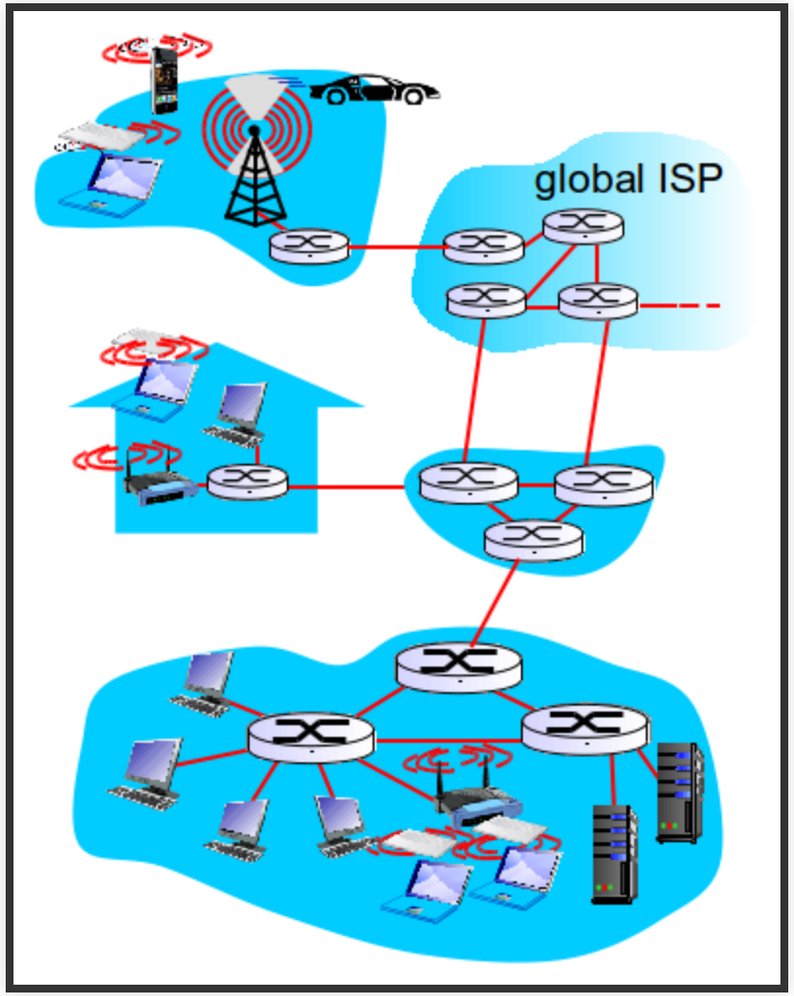
LINK LAYER AND LANS

GOALS

- Understand principles behind link layer services:
  - Error detection, correction
  - Link layer addressing
  - Local area networks: Ethernet, VLANs, and data center networks
- Instantiation, implementation of various link layer technologies

INTRODUCTION, SERVICES



LINK LAYER: INTRODUCTION

Terminology:

- Hosts and routers: **nodes**
- Communication channels that connect adjacent nodes along communication path: links
  - Wired links
  - Wireless links
  - LANs
- Layer-2 packet: **frame**, encapsulates datagram

LINK LAYER: INTRODUCTION

data-link layer has responsibility of transferring datagram from one node to **physically adjacent** node over a link

LINK LAYER: CONTEXT

- Datagram transferred by different link protocols over different links:
  - e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link
- each link protocol provides different services
  - e.g., may or may not provide rdt over link

TRANSPORTATION ANALOGY

- Trip from Princeton to Lausanne
  - limo: Princeton to JFK
  - plane: JFK to Geneva
  - train: Geneva to Lausanne
- tourist = **datagram**
- transport segment = **communication link**
- transportation mode = **link layer protocol**
- travel agent = **routing algorithm**

LINK LAYER SERVICES

Framing, link access:

- Encapsulate datagram into frame, adding header, trailer
- Channel access if shared medium
- "MAC" addresses used in frame headers to identify source, dest
  - Different from IP address!

LINK LAYER SERVICES

Reliability

Reliable delivery between adjacent nodes

- We learned how to do this already (Transport layer)!
- Seldom used on low bit-error link (fiber, some twisted pair)
- Wireless links: high error rates

Q: why both link-level and end-end reliability?

LINK LAYER SERVICES

Flow control

- pacing between adjacent sending and receiving nodes

Half-duplex and full-duplex

- with half duplex, nodes at both ends of link can transmit, but not at same time

LINK LAYER SERVICES

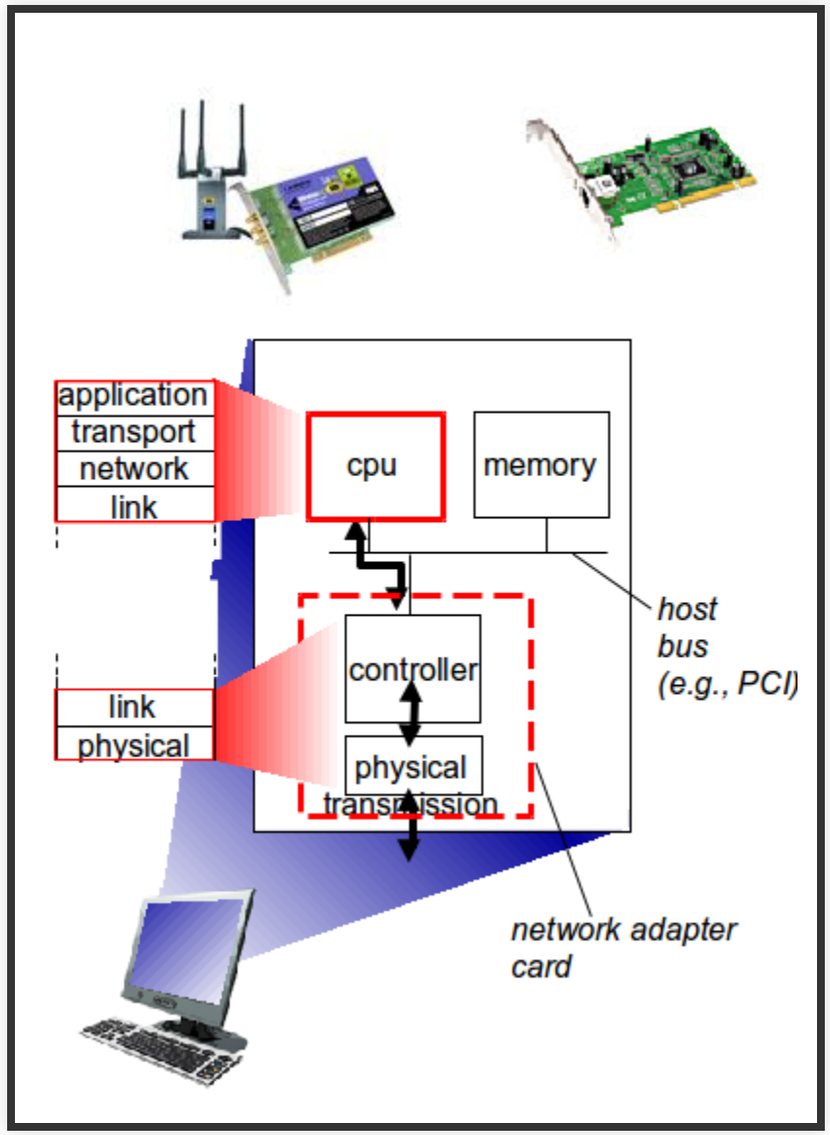
Error detection:

- Errors caused by signal attenuation, noise.
- Receiver detects presence of errors:
    - Signals sender for retransmission or drops frame

Error correction:

- Receiver identifies and corrects bit error(s) without resorting to retransmission

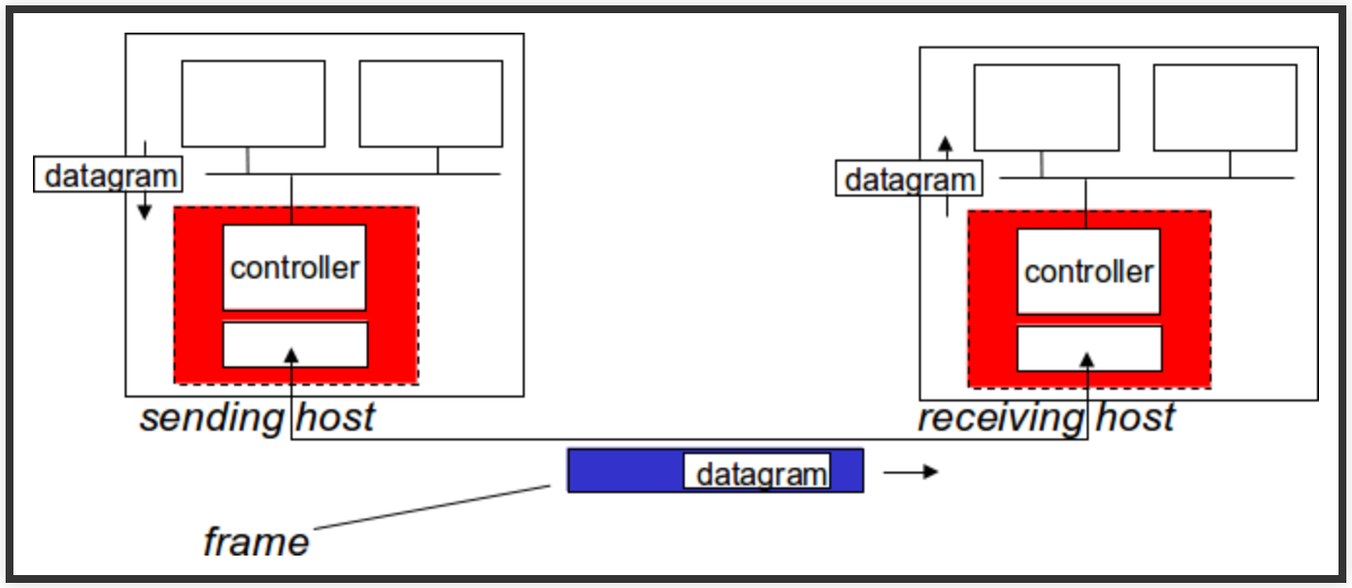
WHERE IS THE LINK LAYER IMPLEMENTED?



WHERE IS THE LINK LAYER IMPLEMENTED?

- In each and every host
- Link layer implemented in "adaptor" (aka **network interface card** NIC) or on a chip
  - Ethernet card, 802.11 card; Ethernet chipset
  - implements link, physical layer
- attaches into host’s system buses
- combination of hardware, software, firmware

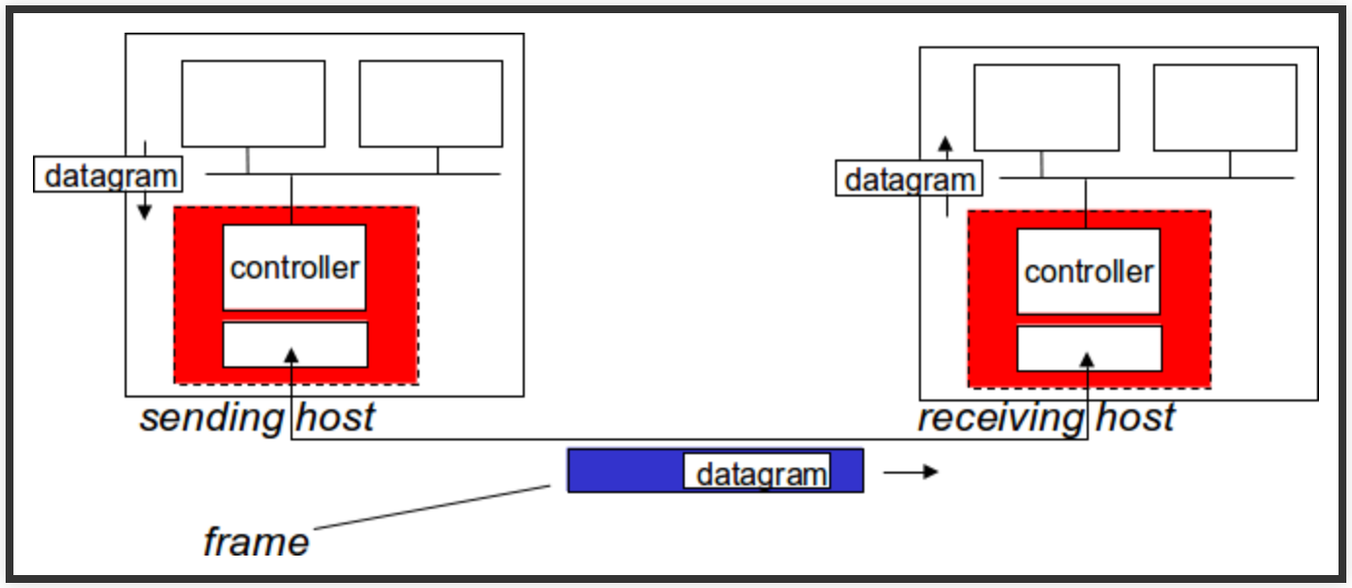
ADAPTORS COMMUNICATING



Sending side:

- Encapsulates datagram in frame
- Adds error checking bits, rdt, flow control, etc.

ADAPTORS COMMUNICATING



Receiving side

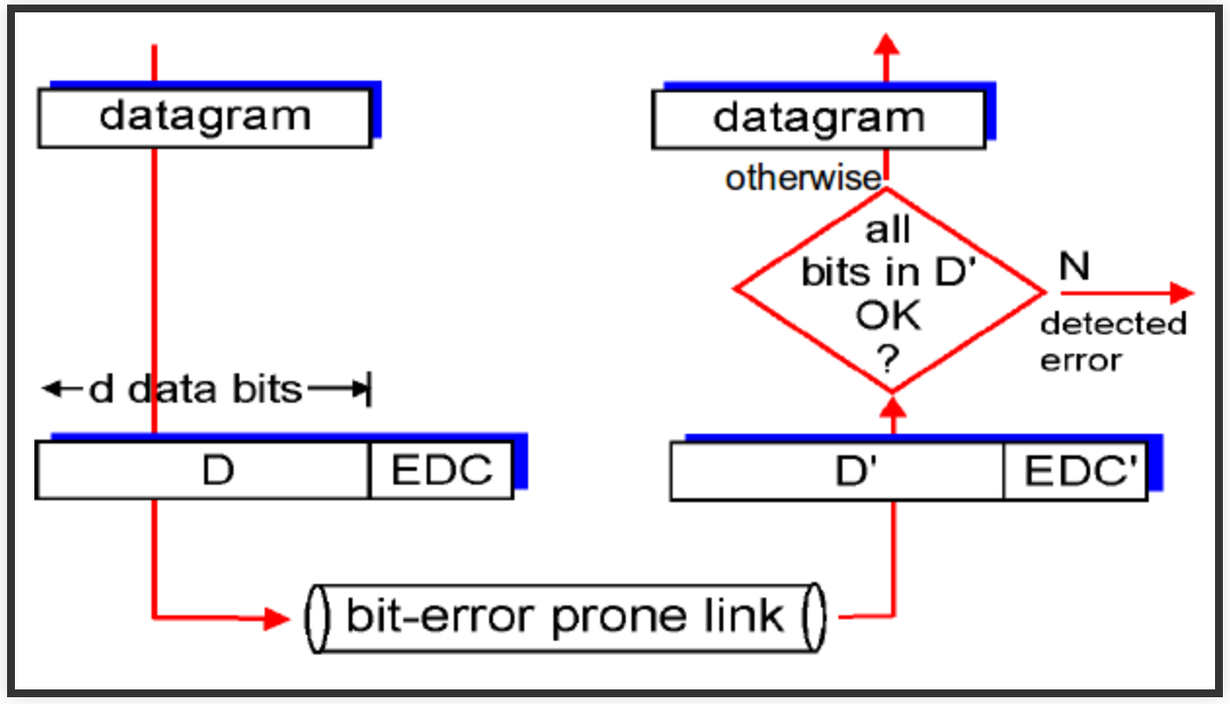
- Looks for errors, rdt, flow control, etc
- Extracts datagram, passes to upper layer at receiving side

ERROR DETECTION, CORRECTION

ERROR DETECTION

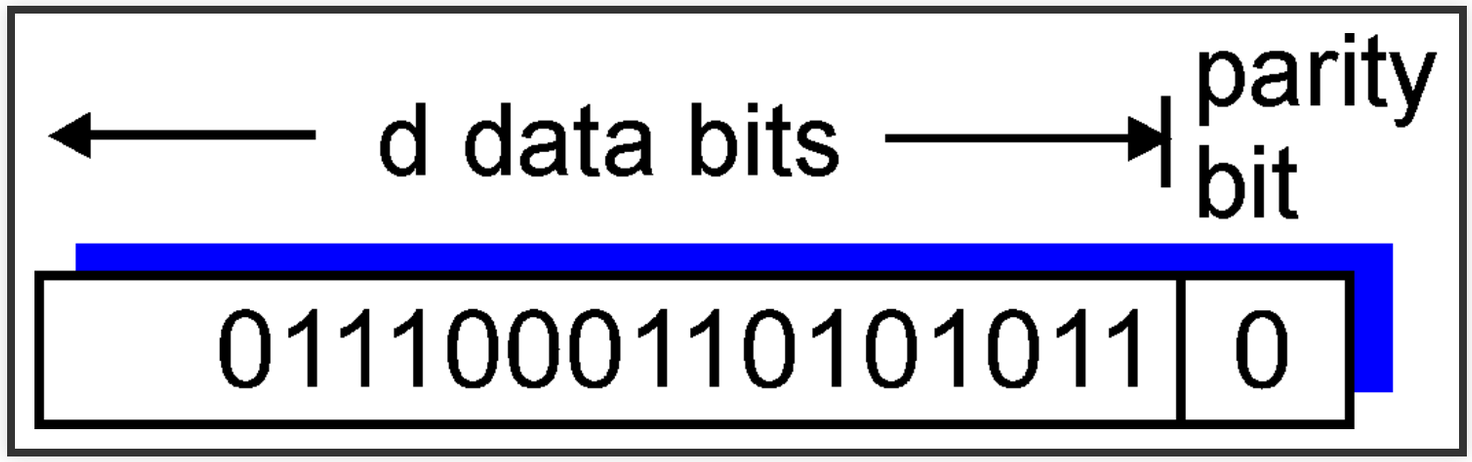
- **EDC** = Error Detection and Correction bits (redundancy)
- **D** = Data protected by error checking, may include header fields
- Error detection not 100% reliable!
  - Protocol may miss some errors, but rarely
  - Larger EDC field yields better detection and correction

ERROR DETECTION



PARITY CHECKING

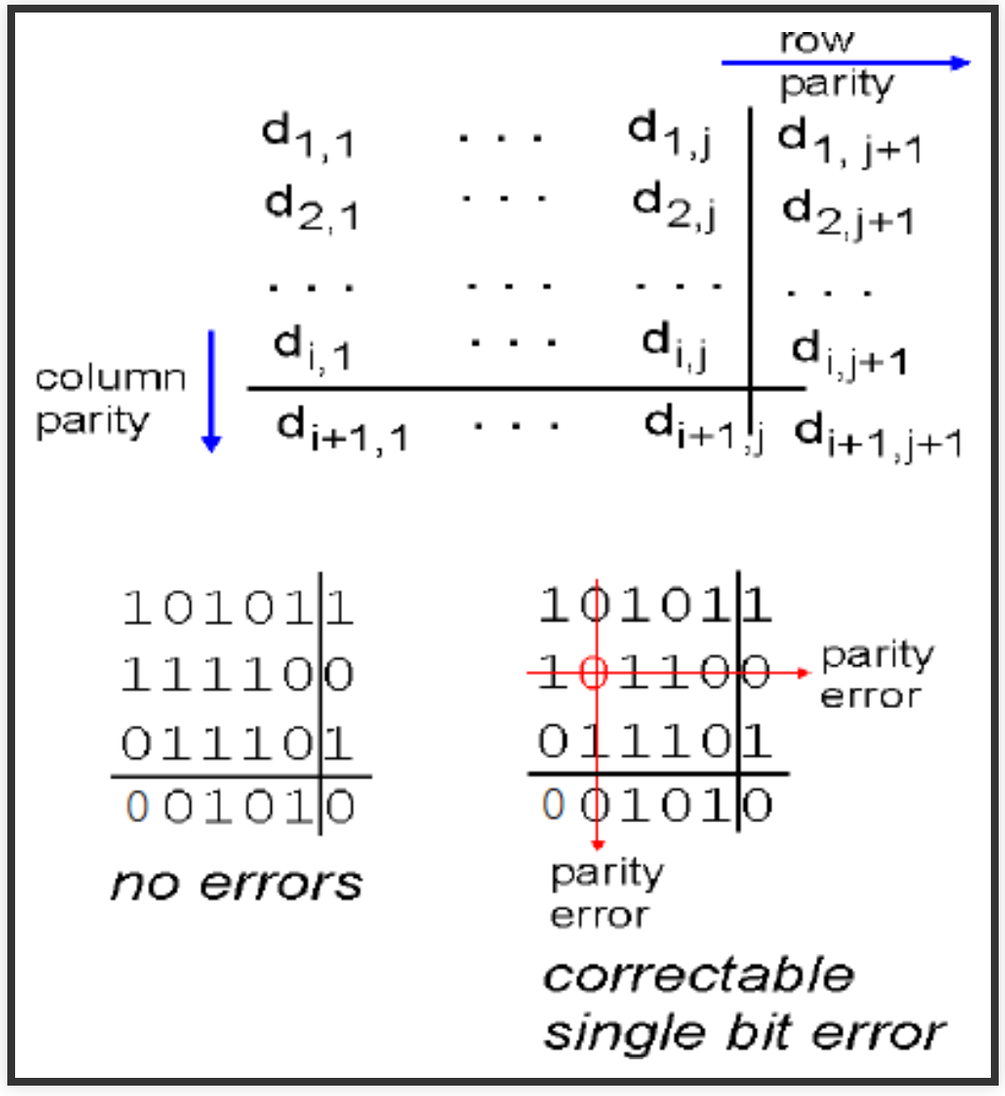
single bit parity: detect single bit errors



PARITY CHECKING

- Two-dimensional bit parity: detect and correct single bit errors

PARITY CHECKING



INTERNET CHECKSUM (REVIEW)

Goal: detect “errors” (e.g., flipped bits) in transmitted packet (note: used at transport layer only)

Sender:

- Treat segment contents as sequence of 16-bit integers
- Checksum: addition (1’s complement sum) of segment contents
- Sender puts checksum value into UDP/TCP checksum field

INTERNET CHECKSUM (REVIEW)

Goal: detect "errors" (e.g., flipped bits) in transmitted packet (note: used at transport layer only)

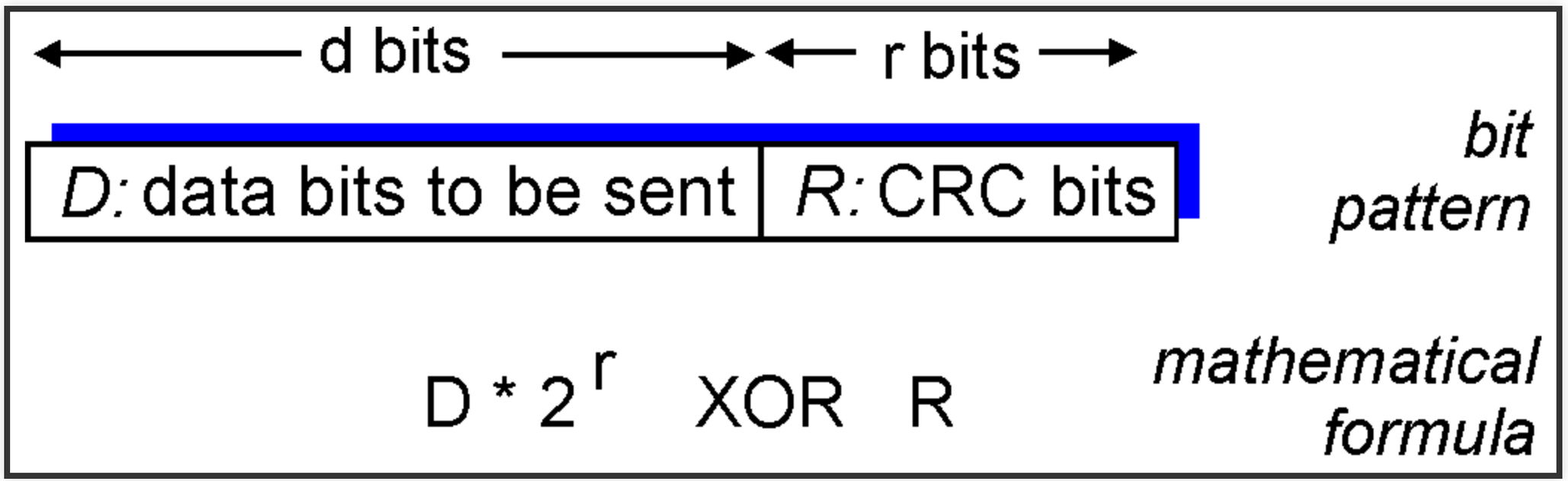
Receiver:

- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - [NO] error detected
  - [YES] no error detected. (But maybe errors nonetheless?)

CYCLIC REDUNDANCY CHECK

- More powerful error-detection coding
- View data bits, **D**, as a binary number
- Choose r+1 bit pattern (generator), **G**
- goal: choose r CRC bits, **R**, such that
  - <D,R> exactly divisible by G (modulo 2)
  - receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
  - can detect all burst errors less than r+1 bits
- **Widely used in practice** (Ethernet, 802.11 WiFi)

CYCLIC REDUNDANCY CHECK



CRC EXAMPLE

want:

$D \cdot 2^r$ XOR R = n · G

equivalently:

$D \cdot 2^r$ = n · G XOR R

equivalently:

if we divide $D \cdot 2^r$ by G, want remainder R to satisfy: R = remainder

[$D \cdot 2^r$ /G]

CRC EXAMPLE

Blackboard calculations?

Lets do the calculation in the book on page 478?

Standards for CRC with 8,1216 and 32 bit generators G. CRC-32 is standard is widely adopted

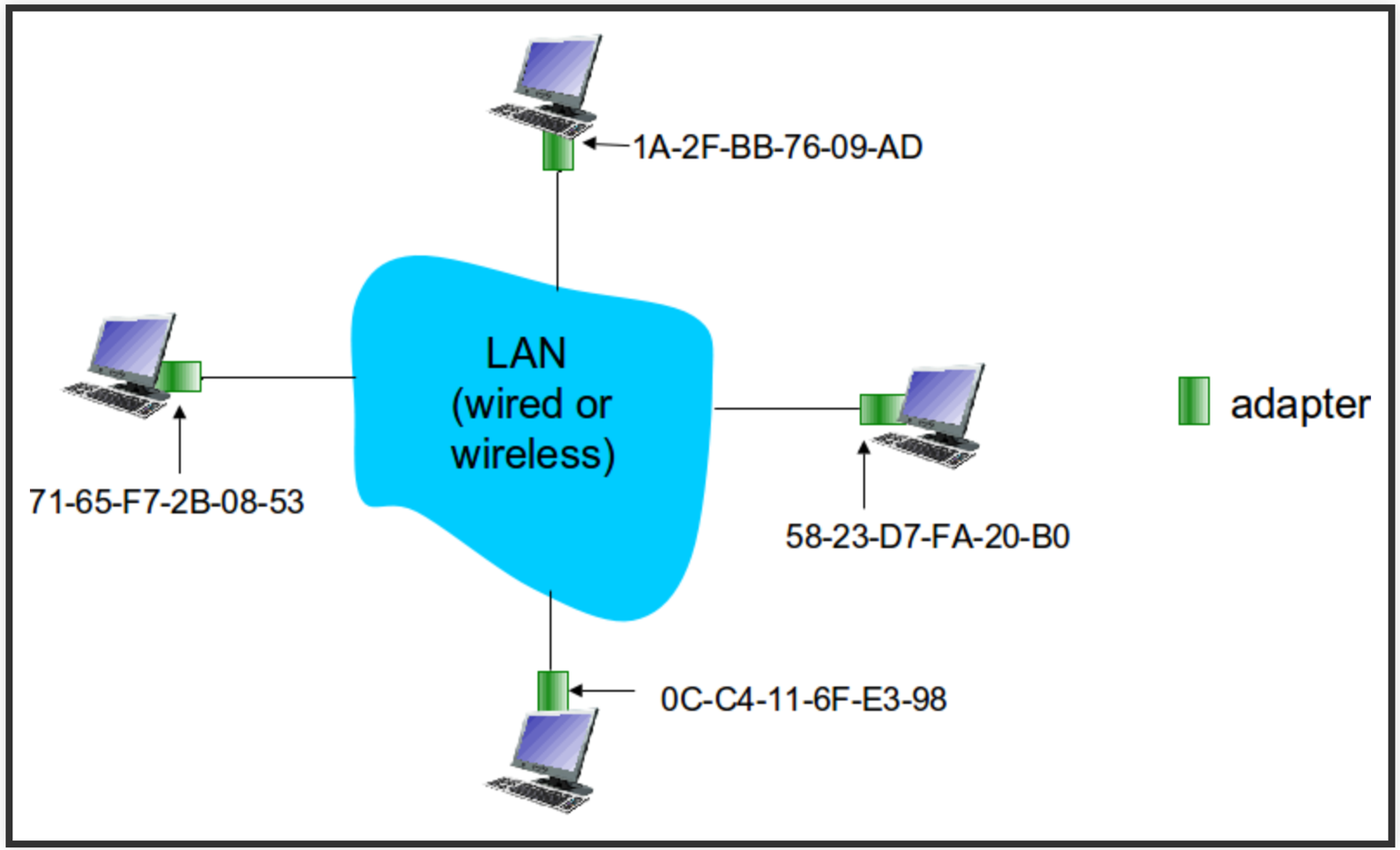
SWITCHED LANS

MAC ADDRESSES AND ARP

- 32-bit IP address:
  - Network-layer address for interface
  - Used for layer 3 (network layer) forwarding
- 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 e.g.: 1A-2F-BB-76-09-AD → hexadecimal (base 16) notation (each "number" represents 4 bits)

LAN ADDRESSES

Each adapter on LAN has unique **LAN** address



LAN ADDRESSES (MORE)

- MAC address allocation administered by IEEE
- Manufacturer buys portion of MAC address space (to assure uniqueness)
- Analogy:
  - MAC address: like Social Security Number
  - IP address: like postal address

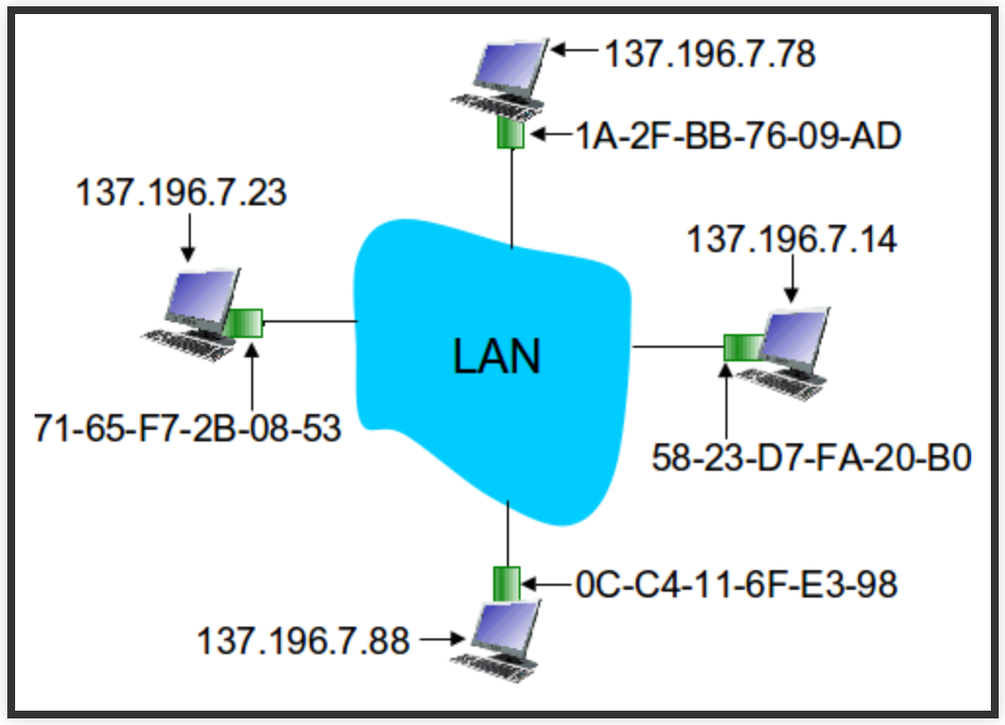
LAN ADDRESSES (MORE)

- MAC flat address → portability
  - Can move LAN card from one LAN to another
- IP hierarchical address not portable
  - Address depends on IP subnet to which node is attached

ARP: ADDRESS RESOLUTION PROTOCOL

ARP: ADDRESS RESOLUTION PROTOCOL

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



ARP: ADDRESS RESOLUTION PROTOCOL

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 is "plug-and-play":
  - Nodes create their ARP tables without intervention from net administrator

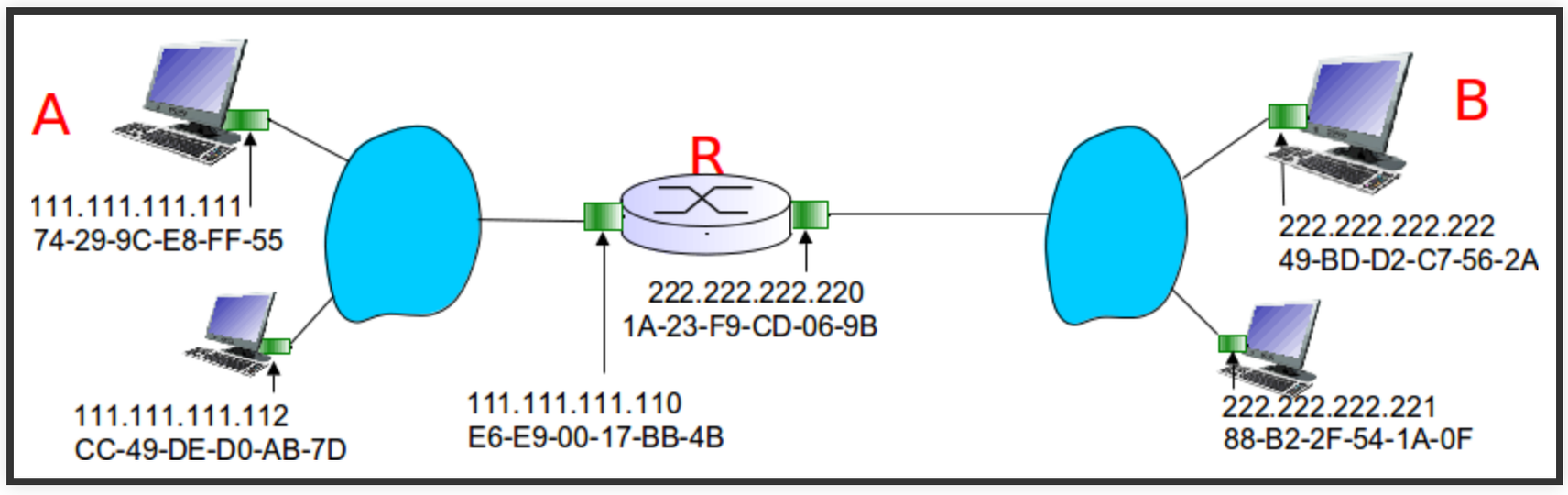
ARP PROTOCOL: SAME LAN

- 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 IP-to-MAC address pair in its ARP table until information times out. (Soft state)

ADDRESSING: ROUTING TO ANOTHER LAN

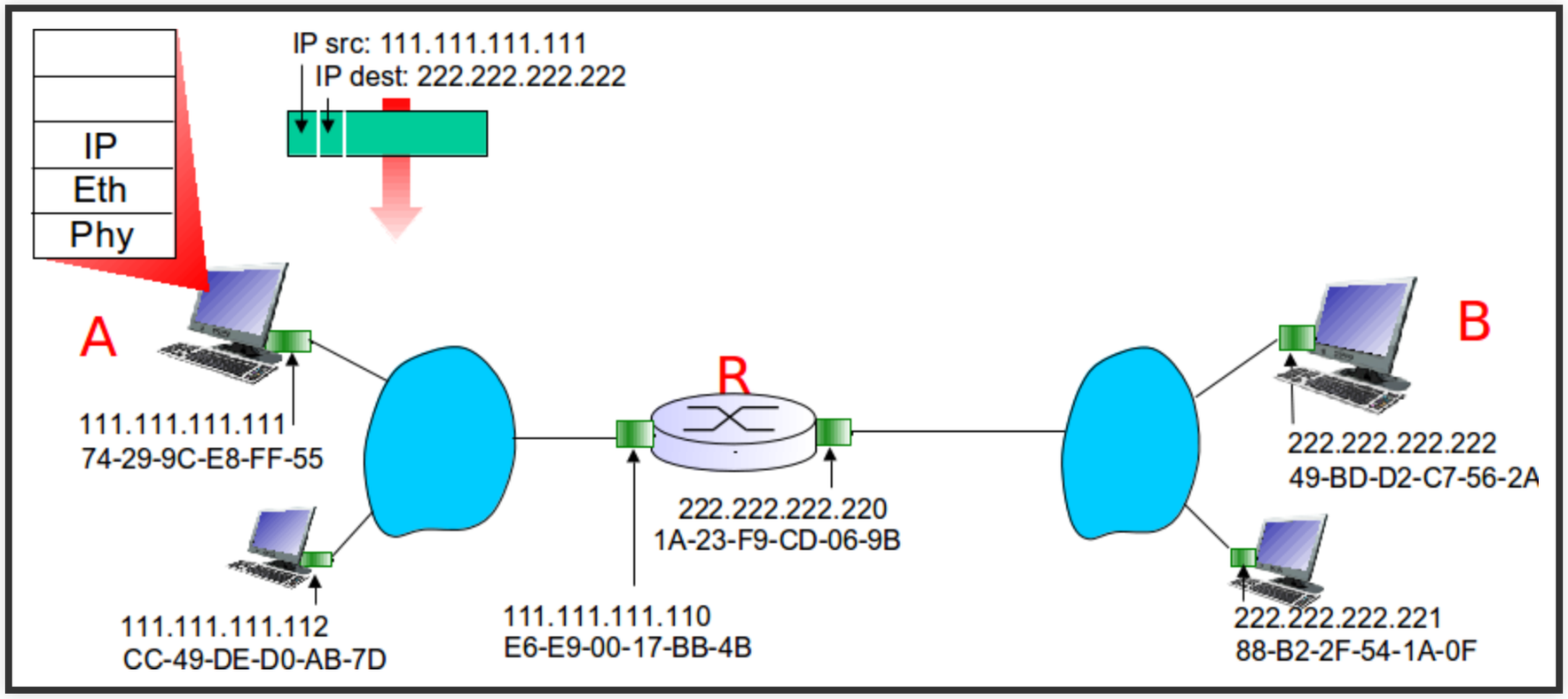
walkthrough: **send datagram from A to B via R**

- 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



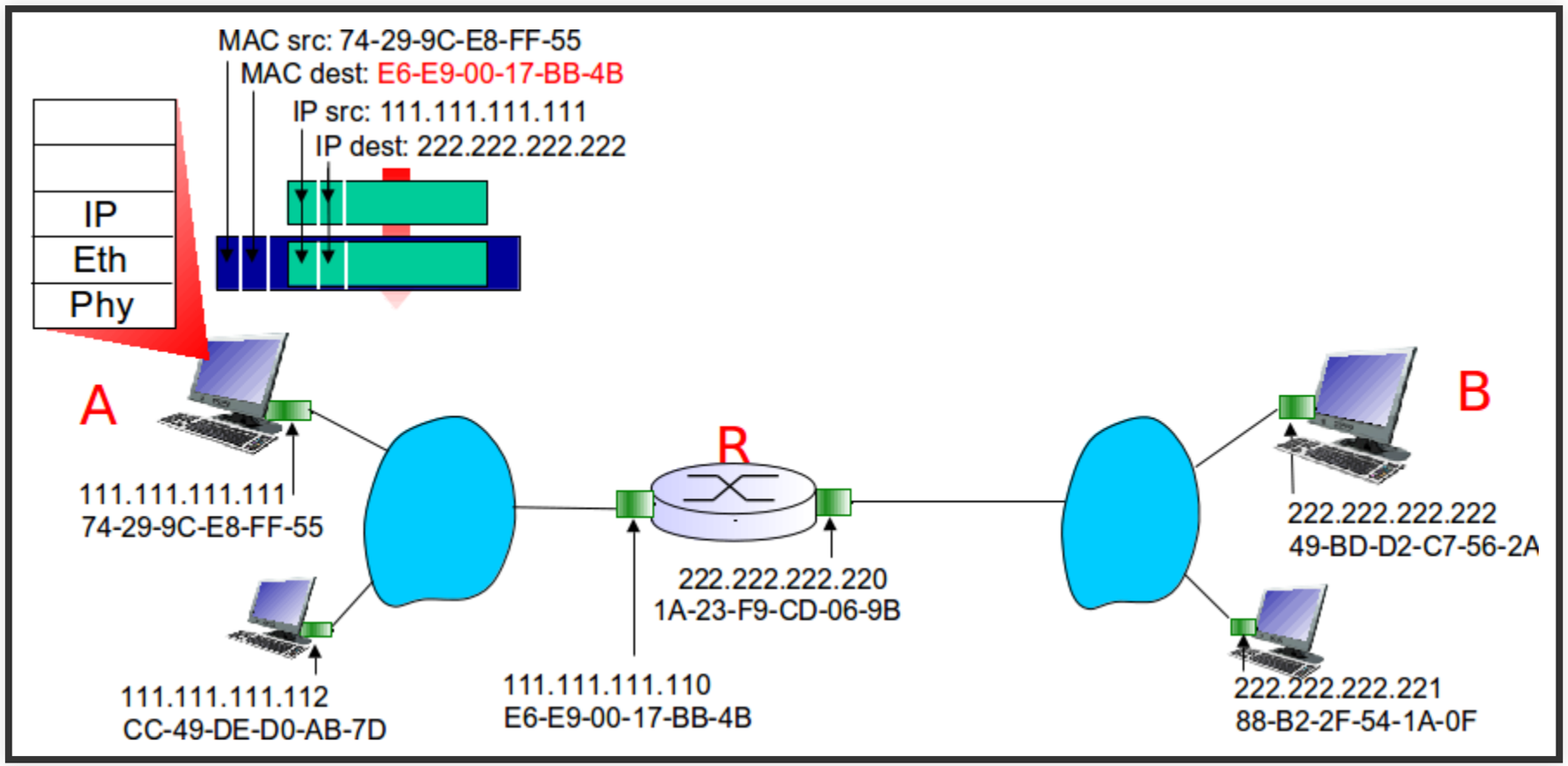
ADDRESSING: ROUTING TO ANOTHER LAN

- A creates IP datagram with IP source A, destination B



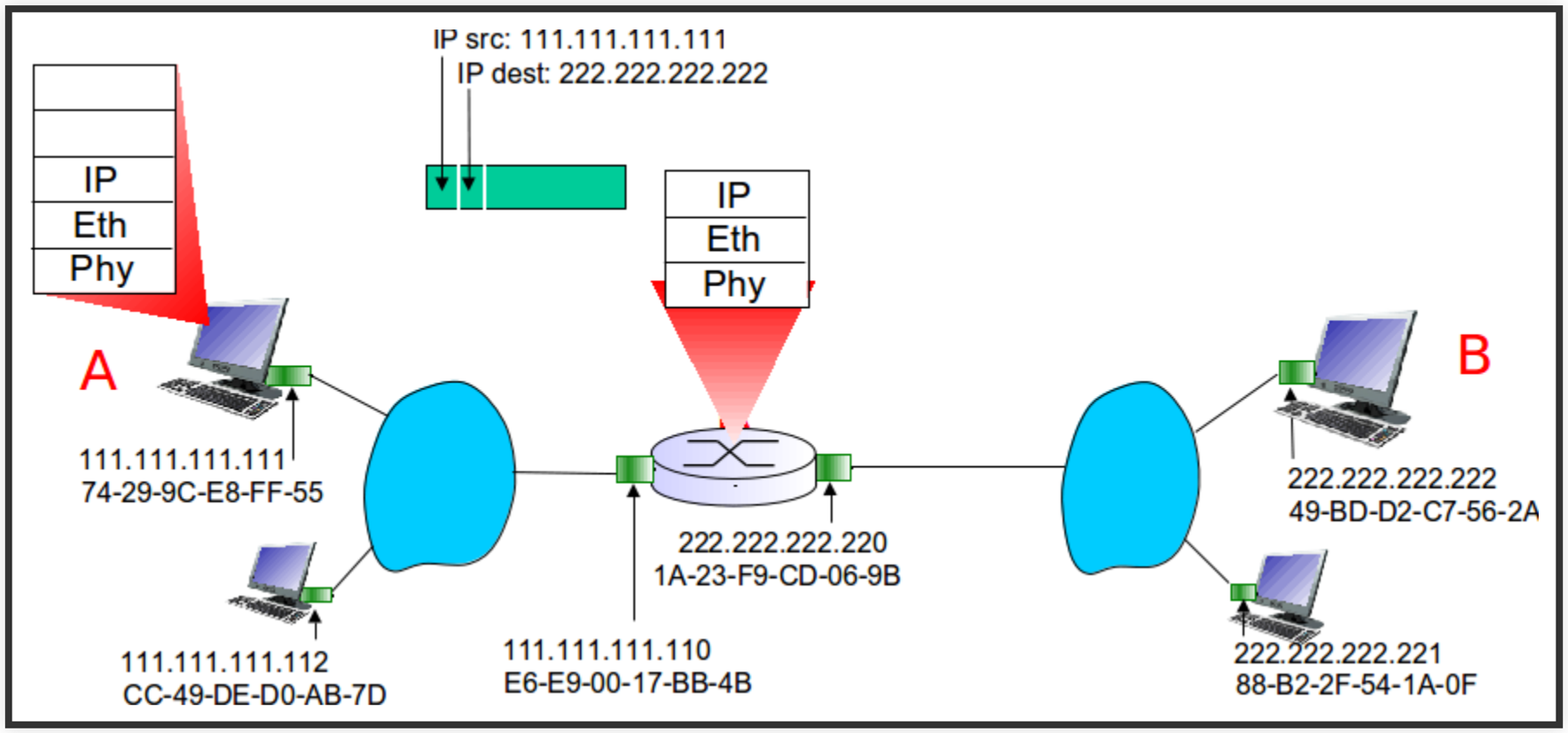
ADDRESSING: ROUTING TO ANOTHER LAN

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



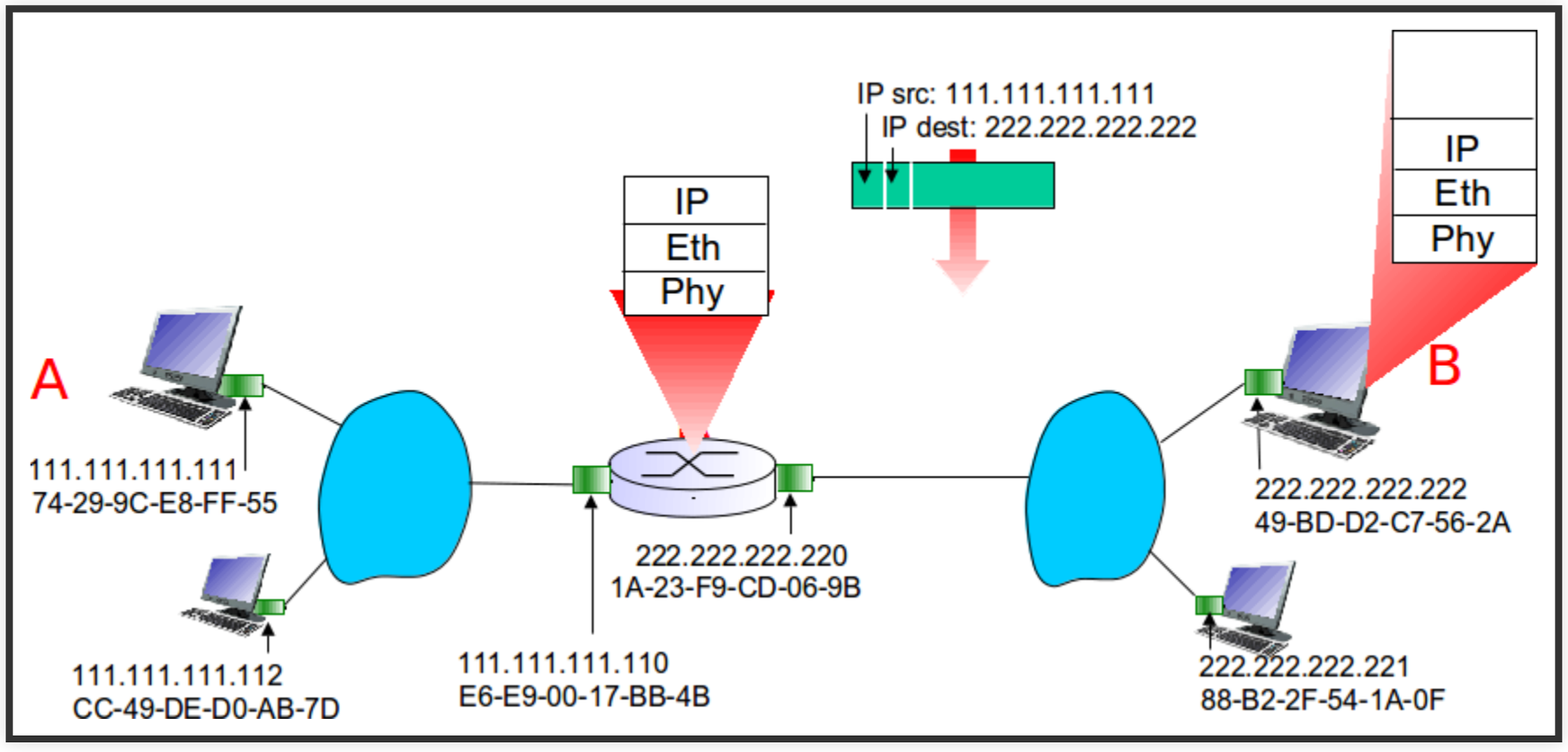
ADDRESSING: ROUTING TO ANOTHER LAN

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



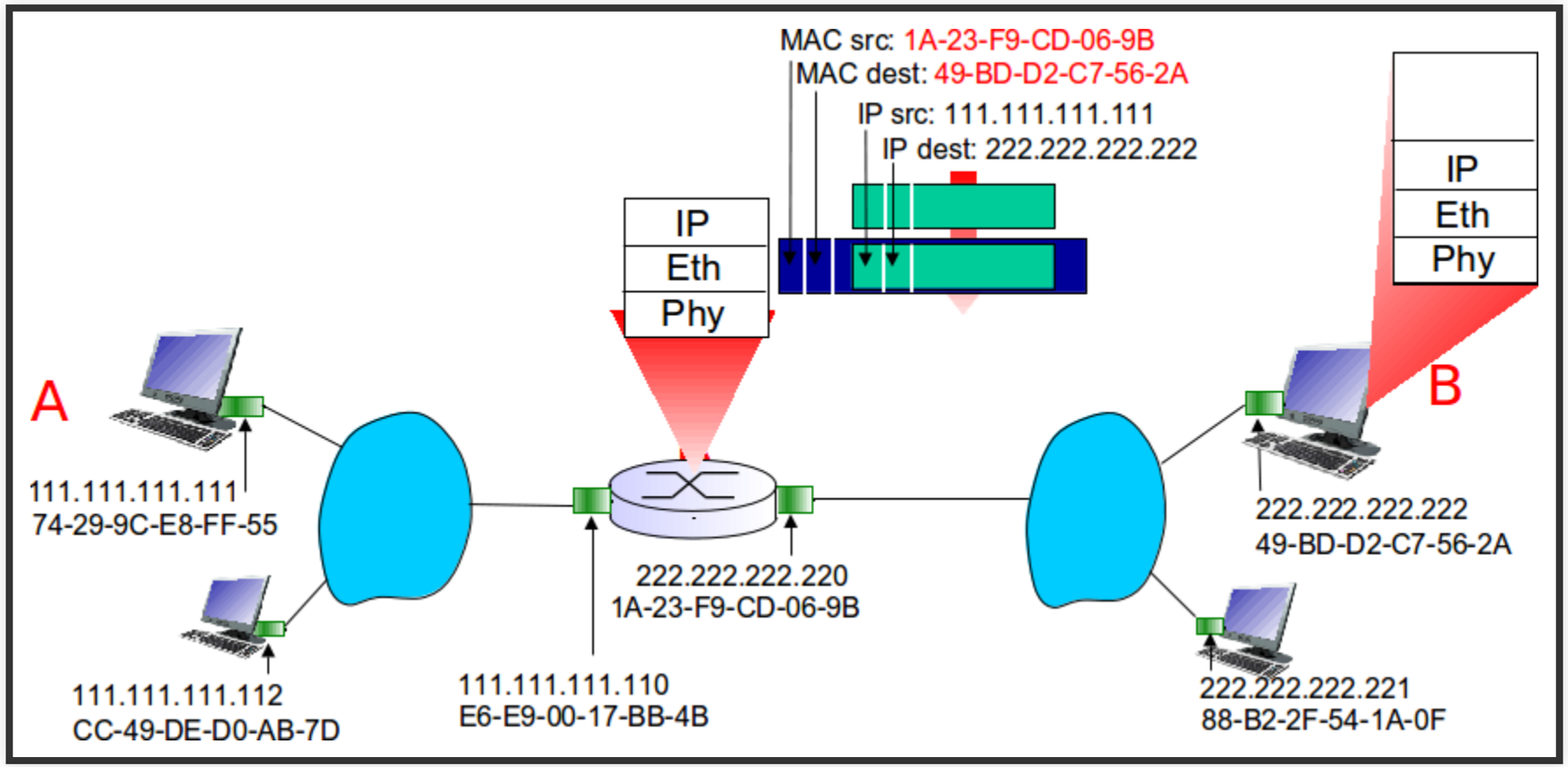
ADDRESSING: ROUTING TO ANOTHER LAN

- R forwards datagram with IP source A, destination B



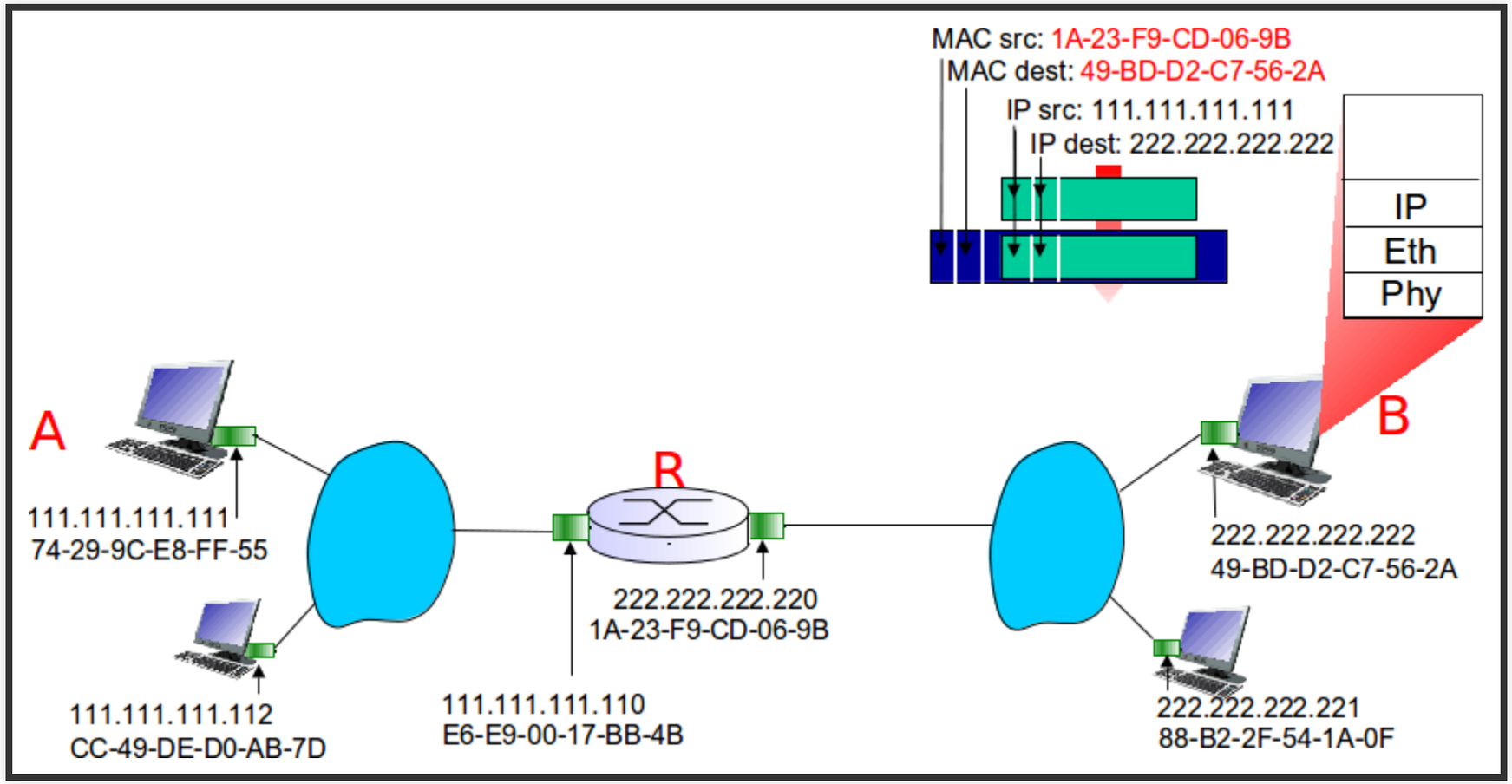
ADDRESSING: ROUTING TO ANOTHER LAN

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



ADDRESSING: ROUTING TO ANOTHER LAN

- Now the frame is adressed within own subnet

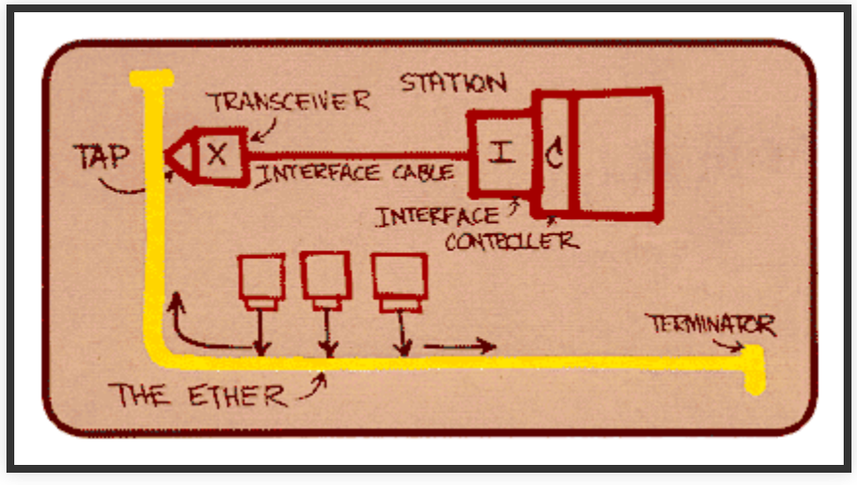


ETHERNET

ETHERNET

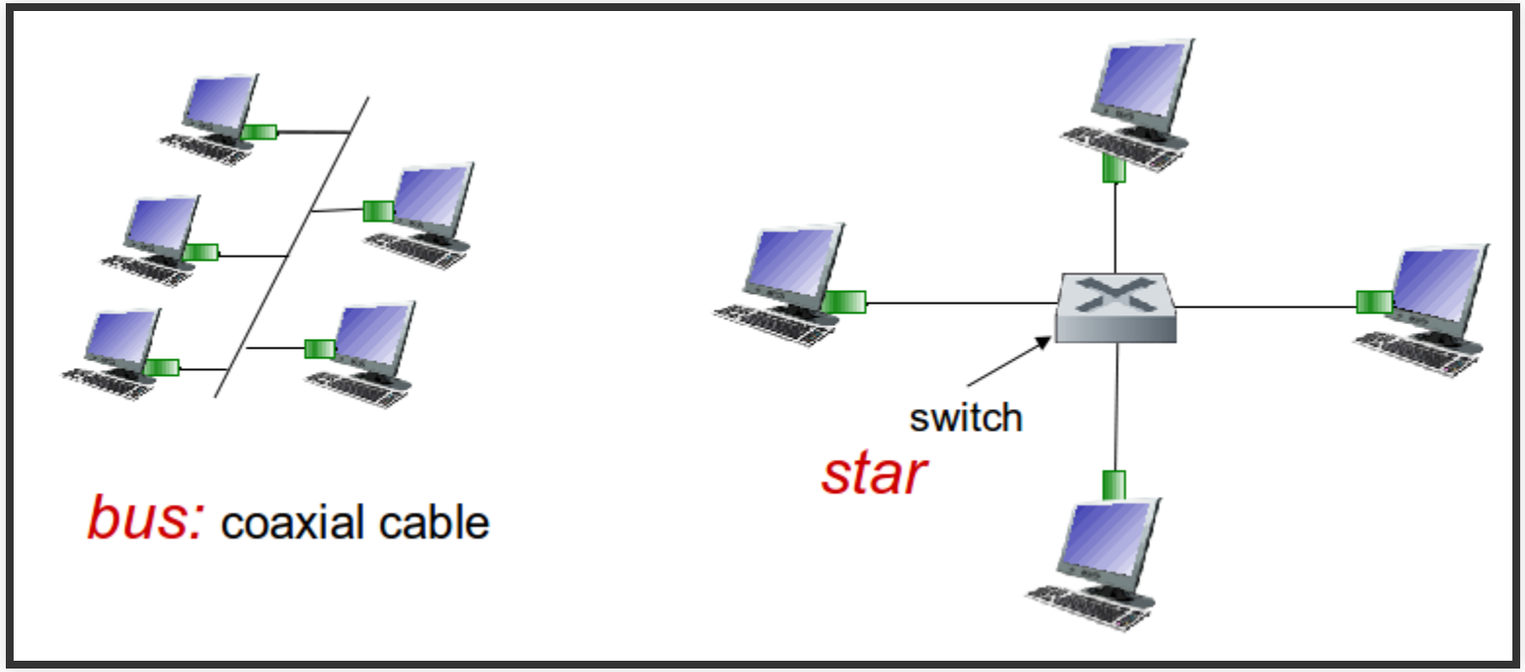
"Dominant" wired LAN technology:

- 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



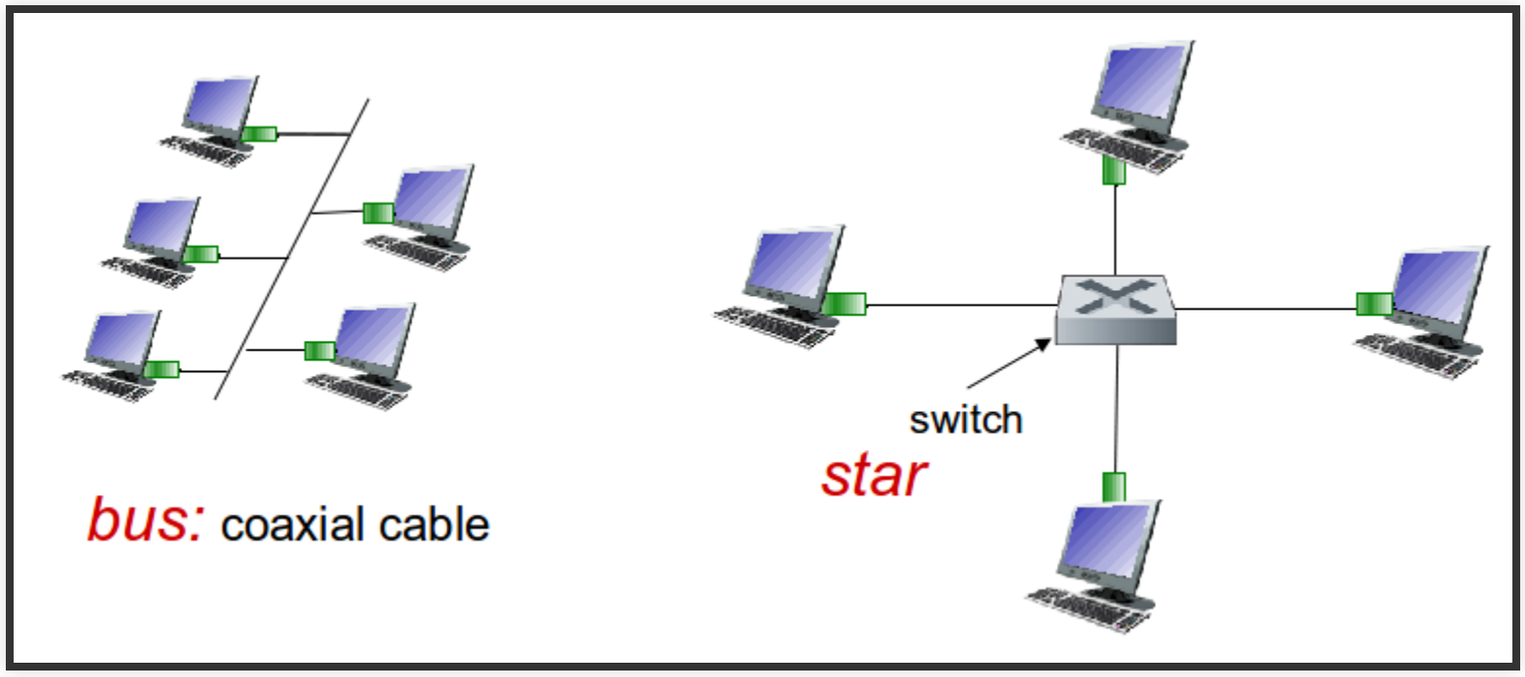
ETHERNET: PHYSICAL TOPOLOGY

- **bus:** popular through mid 90s
  - all nodes in same collision domain (can collide with each other)



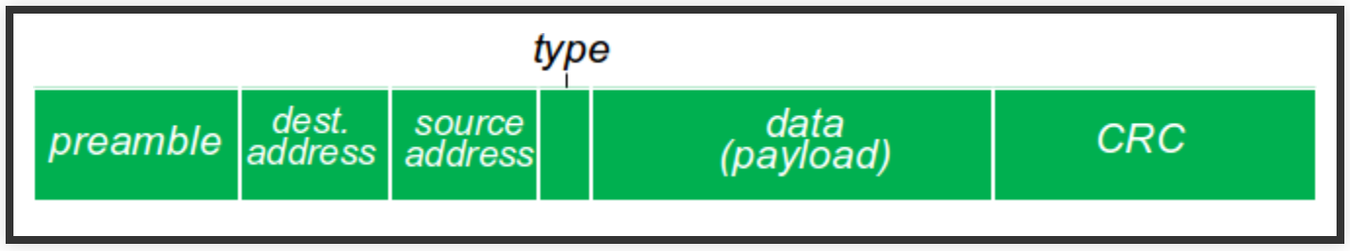
ETHERNET: PHYSICAL TOPOLOGY

- **star:** prevails today
  - active switch in center
  - each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)

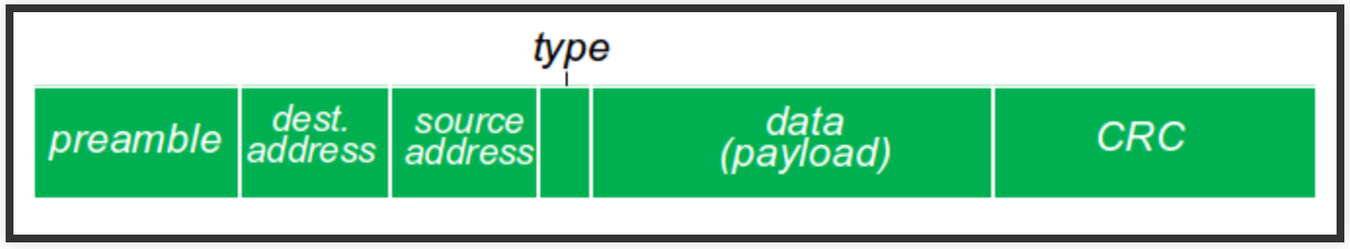


ETHERNET FRAME STRUCTURE

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



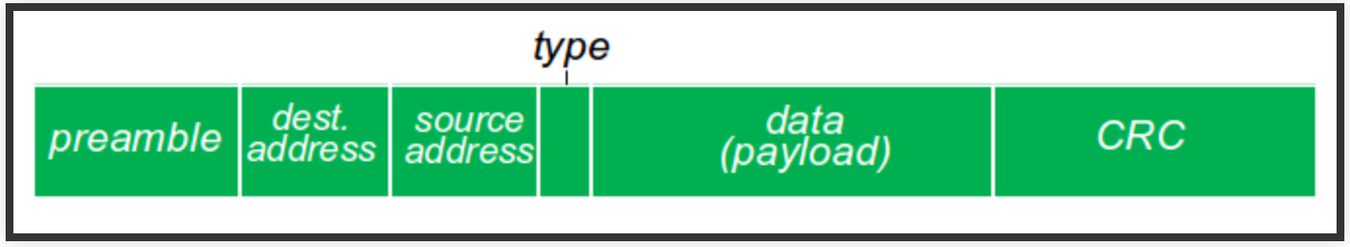
ETHERNET FRAME STRUCTURE



preamble:

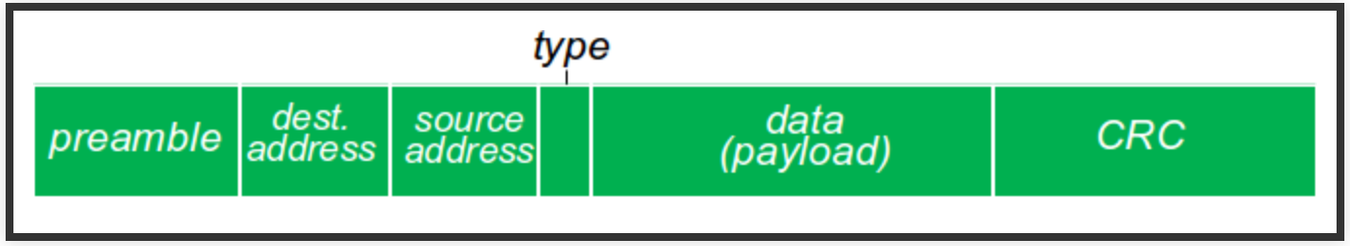
- 7 bytes with pattern 10101010 followed by one byte with pattern 10101011
- Used to synchronize receiver, sender clock rates

ETHERNET FRAME STRUCTURE (MORE)



- **Addresses:** 6 byte source, destination MAC addresses
  - if adapter receives frame with matching destination address, or with broadcast address (e.g. ARP packet), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame

ETHERNET FRAME STRUCTURE (MORE)



- **type:** indicates higher layer protocol (mostly IP but others possible, e.g., Novell IPX, AppleTalk)
- **CRC:** cyclic redundancy check at receiver
  - Error detected: frame is dropped

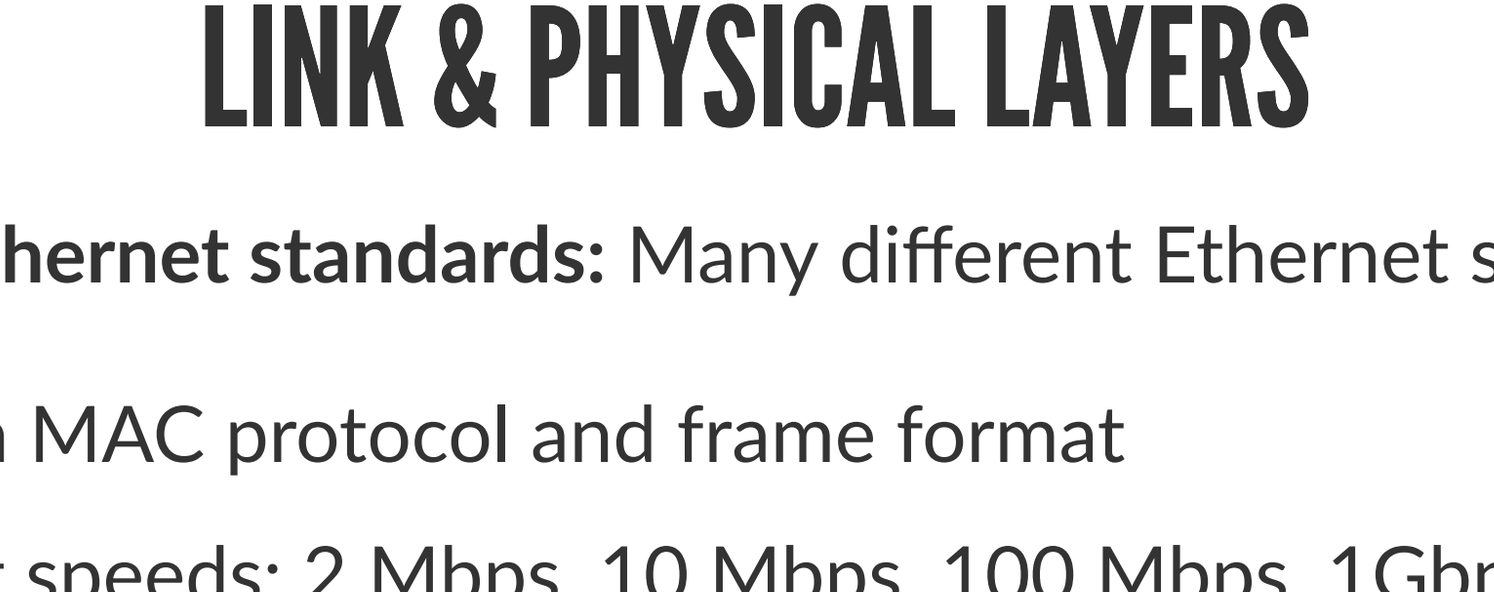
ETHERNET: UNRELIABLE, CONNECTIONLESS

- **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 initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost
- Ethernet's MAC protocol: unslotted **CSMA/CD wth binary backoff** (next lecture!)

LINK & PHYSICAL LAYERS

802.3 Ethernet standards: Many different Ethernet standards

- common MAC protocol and frame format
- different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1Gbps, 10G bps
- different physical layer media: fiber, cable



ETHERNET SWITCH

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

ETHERNET SWITCH

Transparent

- Hosts are unaware of presence of switches

Plug-and-play, self-learning

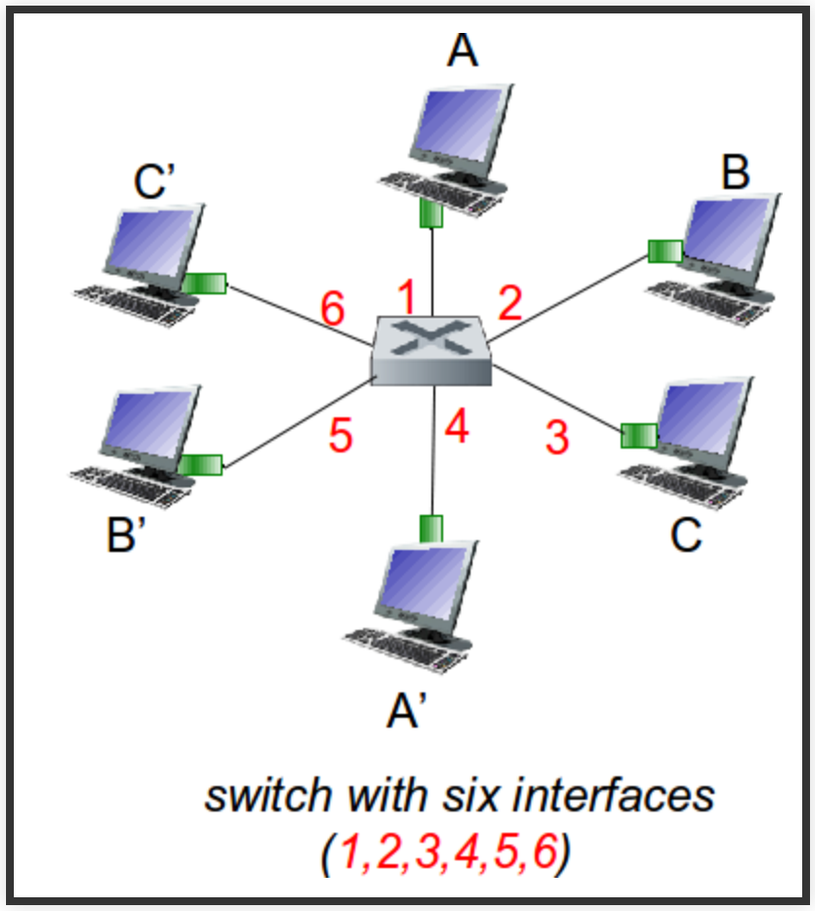
- Switches do not need to be configured

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

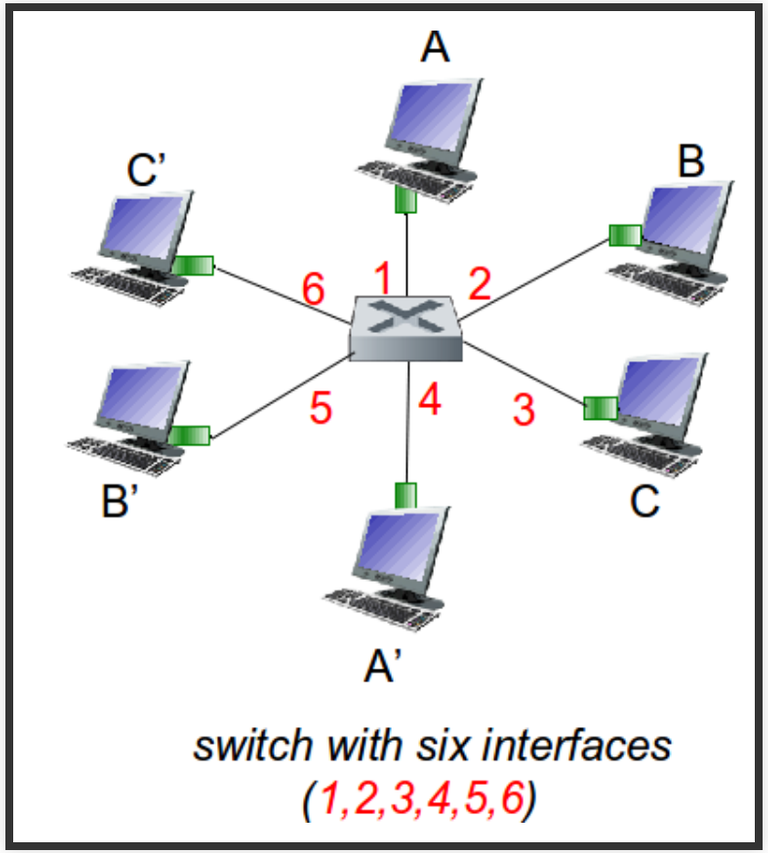
SWITCH: *MULTIPLE* SIMULTANEOUS TRANSMISSIONS

- **switching:** A-to-A' and B-to-B' can transmit simultaneously, without collisions



SWITCH FORWARDING TABLE

- How does switch know A’ reachable via interface 4, B’ reachable via interface 5?
- Each switch has a switch table, each entry: (MAC address of host, interface to reach host, time stamp)



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

SWITCH: FRAME FILTERING/FORWARDING

when frame received at switch:

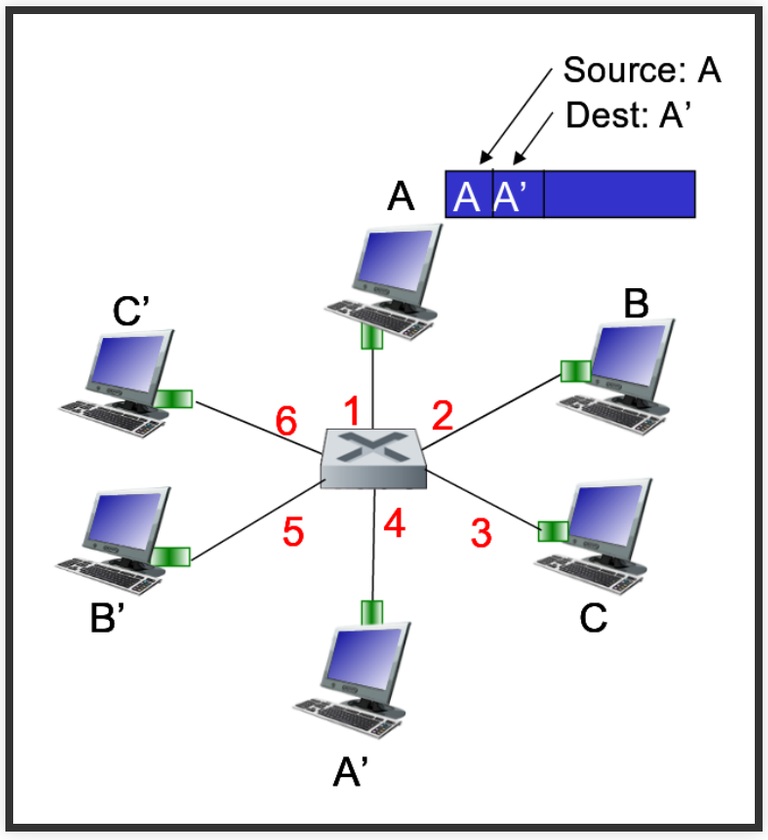
- record incoming link, MAC address of sending host
- index switch table using MAC destination address

```
if entry found for destination
    if destination on segment from which frame arrived
        drop frame
    else
        forward frame on interface indicated by entry
else
   /*forward on all interfaces except arriving interface*/
   flood
```

SWITCH: SELF-LEARNING

MAC Addr	Interface	TTL

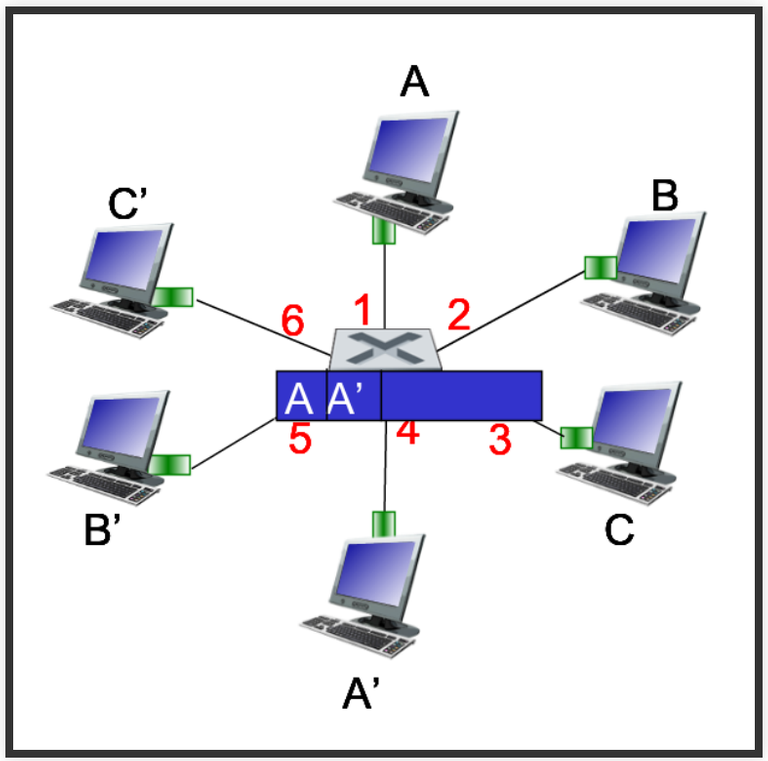
Switch table (initially empty)



SWITCH: SELF-LEARNING

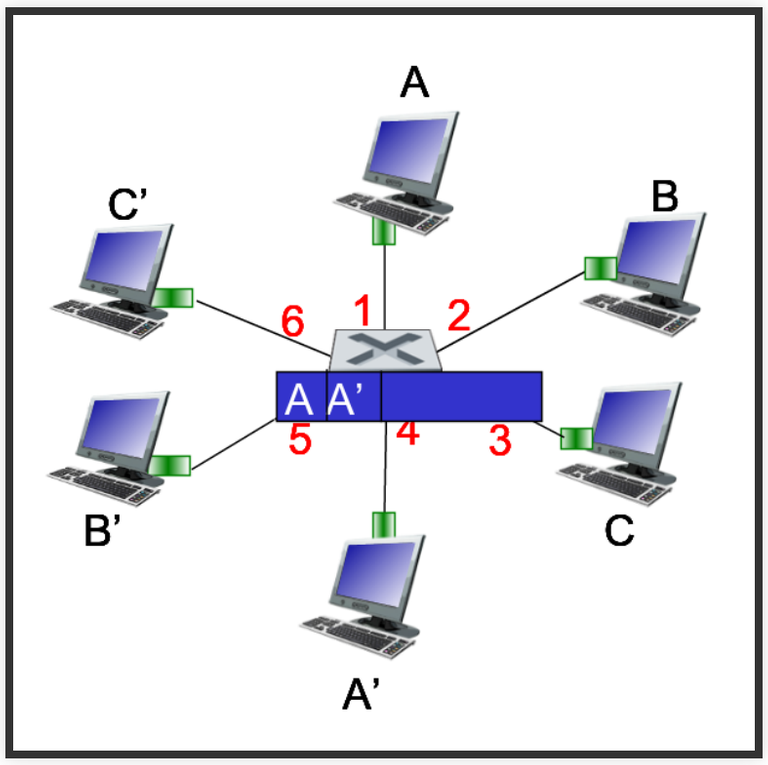
- frame destination, A', location unknown → **flood**
- destination A location known: **selectively send on just one link**

MAC Addr	Interface	TTL



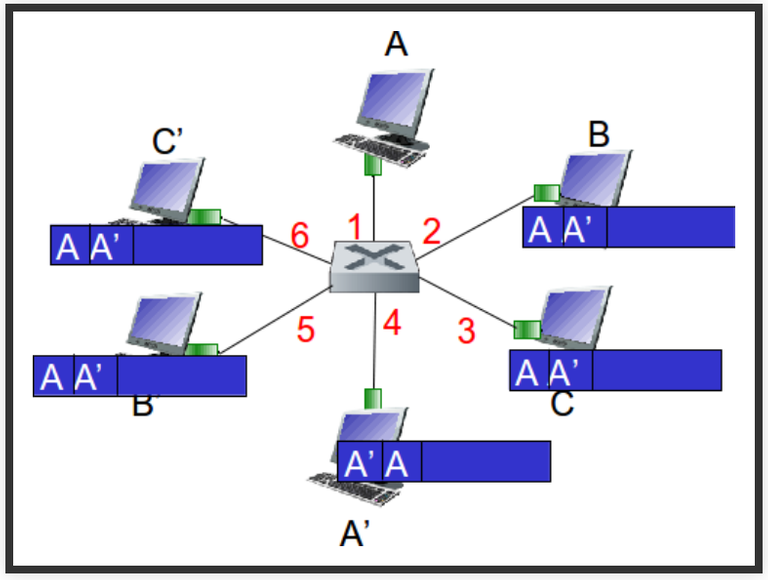
SWITCH: SELF-LEARNING

MAC Addr	Interface	TTL
A	1	60



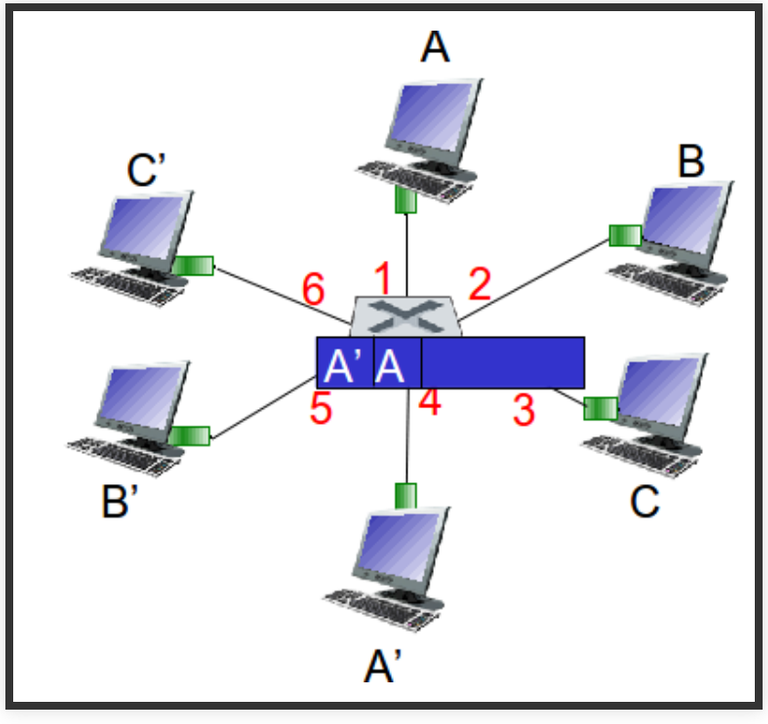
SWITCH: SELF-LEARNING

MAC Addr	Interface	TTL
A	1	60



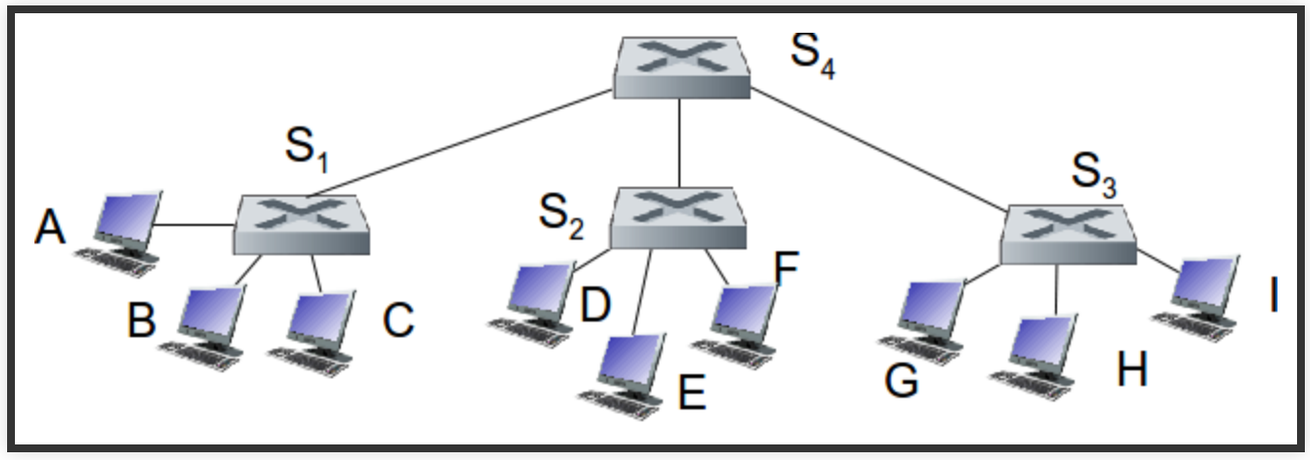
SWITCH: SELF-LEARNING

MAC Addr	Interface	TTL
A	1	60
A'	4	60



INTERCONNECTING SWITCHES

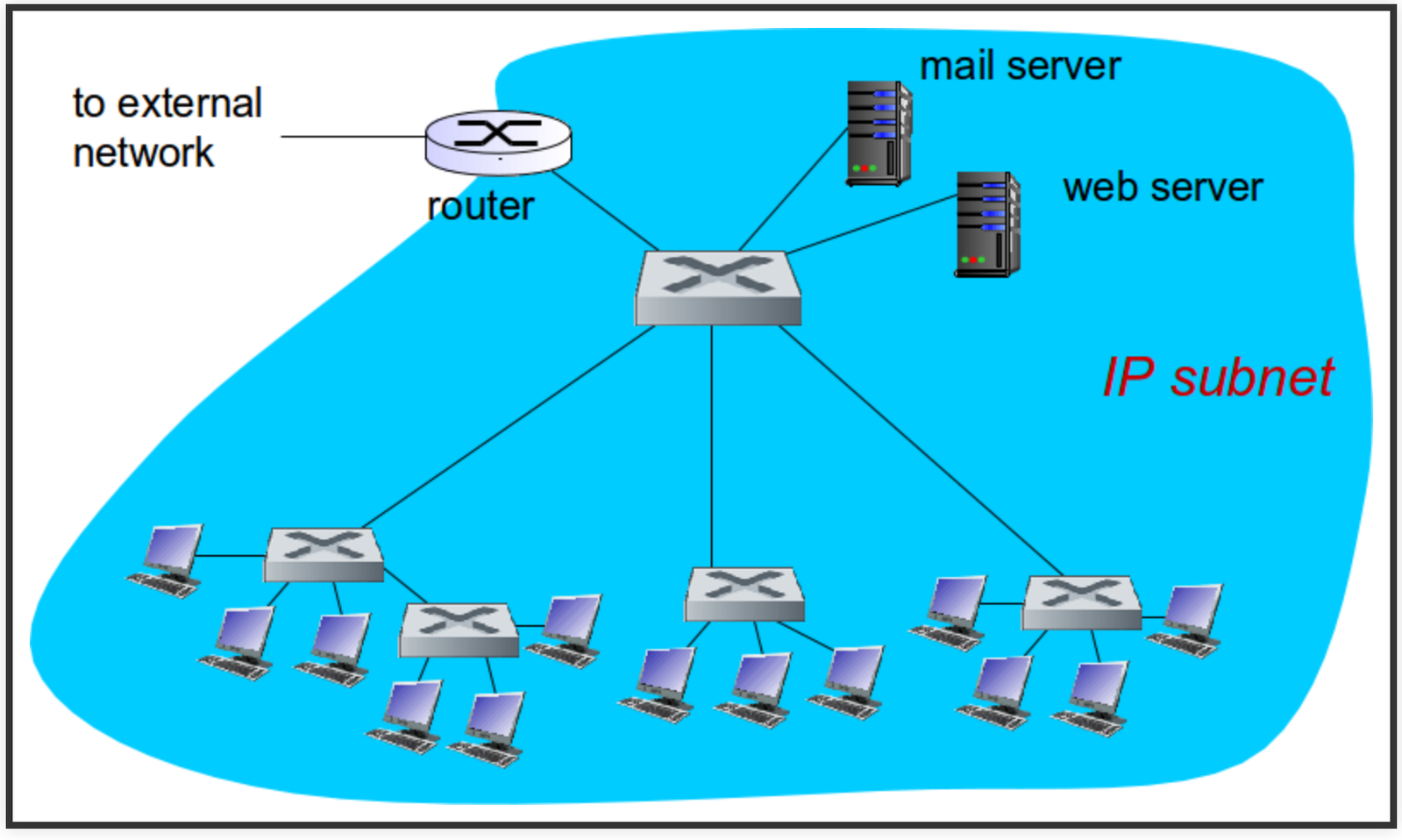
Switches can be connected together



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!)

INSTITUTIONAL NETWORK



PROPERTIES OF LINK-LAYER SWITCHING

- Elimination of Collisions
  - Buffer frames, sends only one at a time on a link
- Heterogeneous links
  - Links can run different speed, different media
- Management
  - Managed switches can
    - correct problems by closing ports
    - collect statistics

SWITCHES VS. ROUTERS

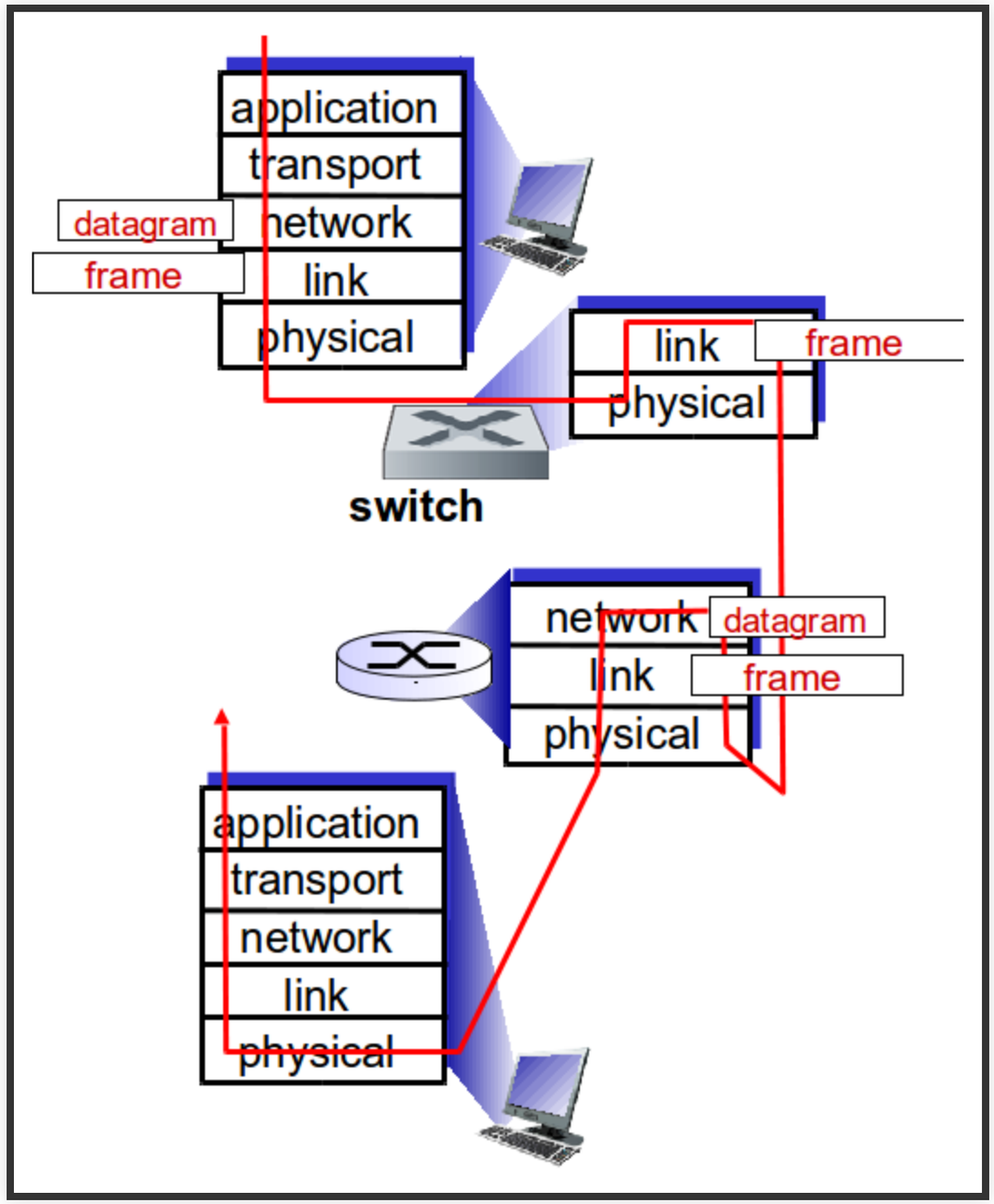
both are store-and-forward:

- **routers:** network-layer devices (examine network-layer headers)
- **switches:** link-layer devices (examine link-layer headers)

both have forwarding tables:

- **routers:** compute tables using routing algorithms, IP addresses
- **switches:** learn forwarding table using flooding, learning, MAC addresses

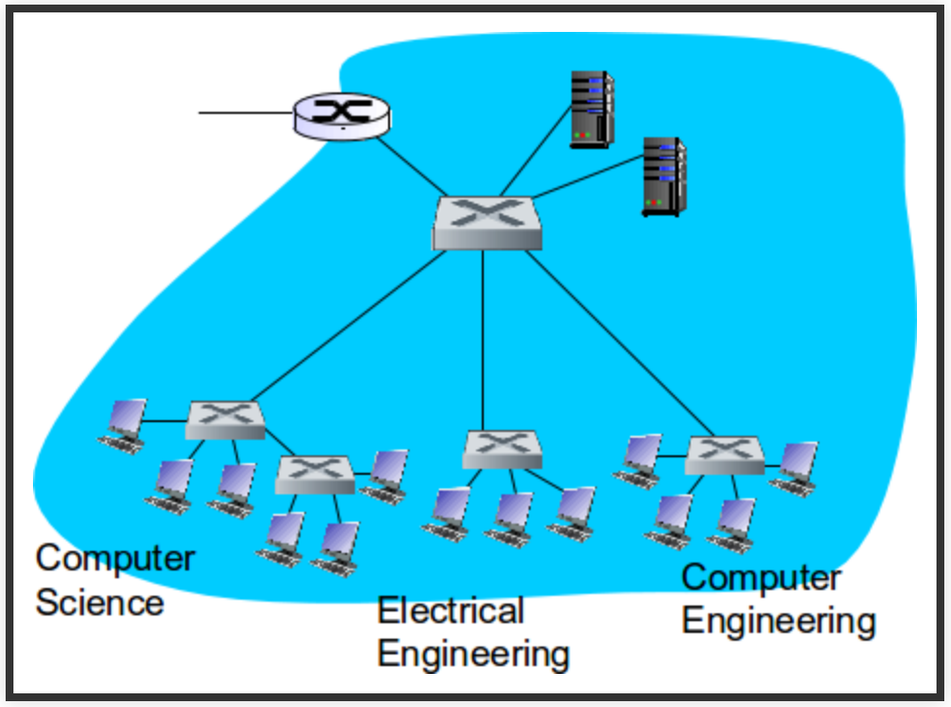
SWITCHES VS. ROUTERS



VLANS

Virtual Local Area Network

VLANS: MOTIVATION



VLANS: MOTIVATION

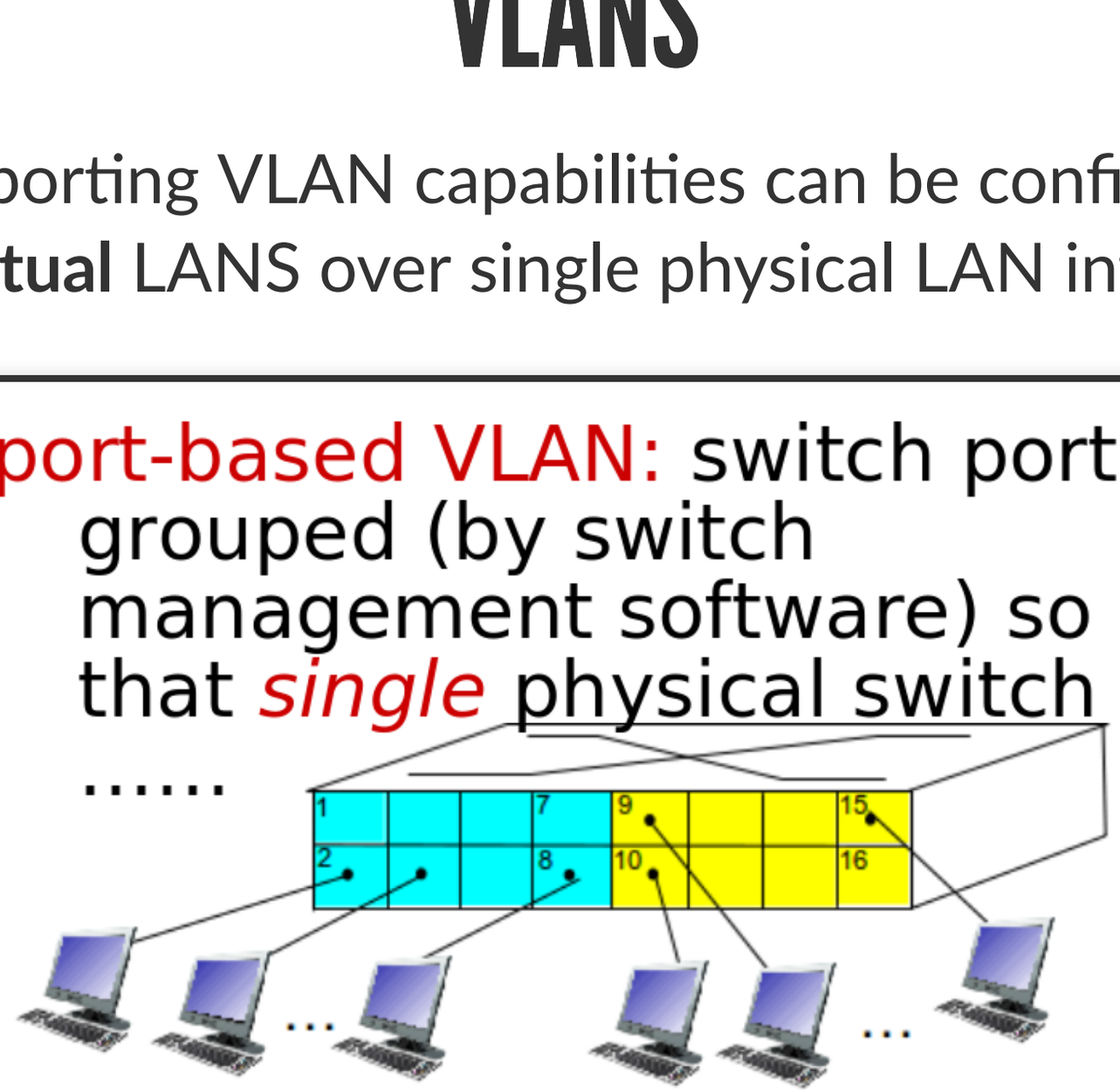
Consider:

- CS user moves office to EE, but wants connect to CS switch?
- Single broadcast domain:
  - All layer-2 broadcast traffic (ARP, DHCP, unknown location of destination MAC address) must cross entire LAN
  - Security/privacy, efficiency issues

VLANS

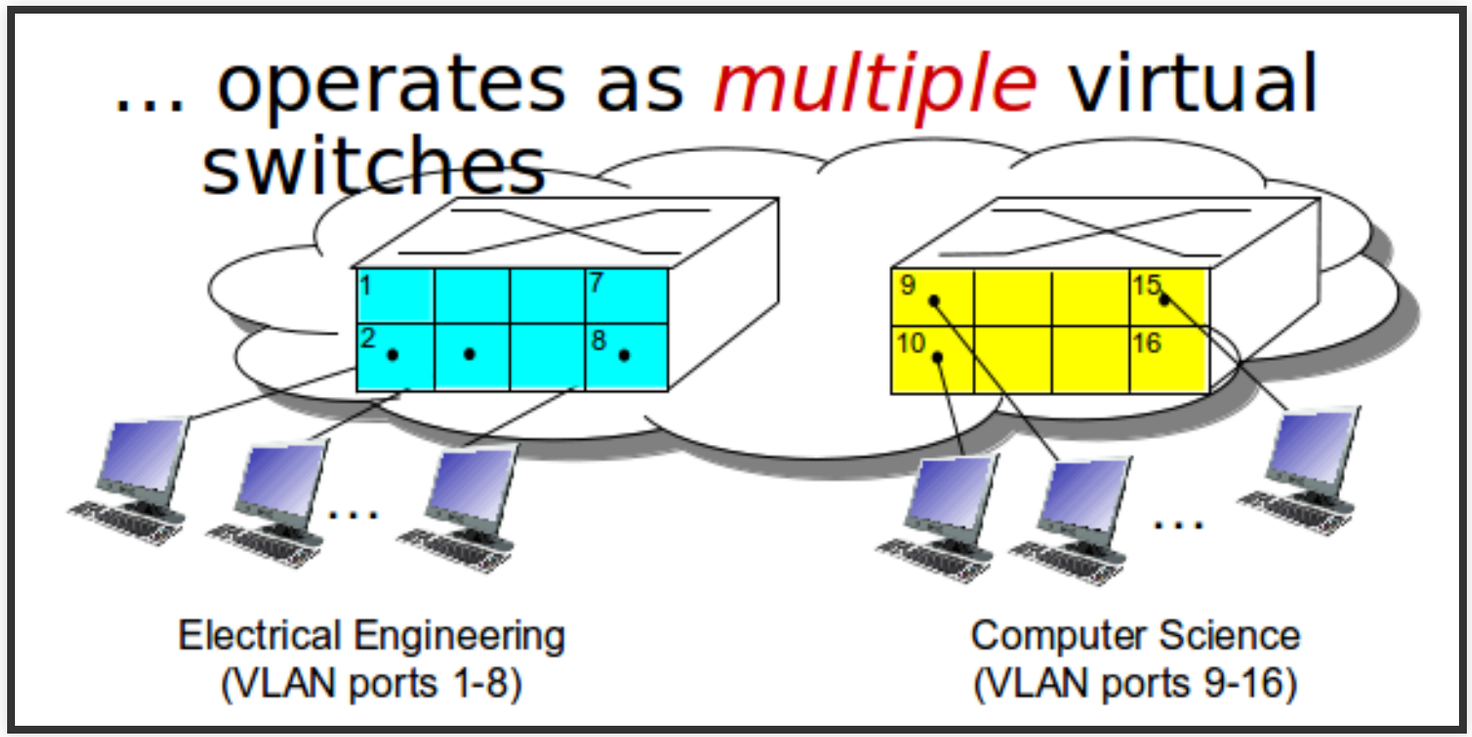
switch(es) supporting VLAN capabilities can be configured to define multiple **virtual** LANS over single physical LAN infrastructure.

port-based VLAN: switch ports grouped (by switch management software) so that *single* physical switch .....



VLANS

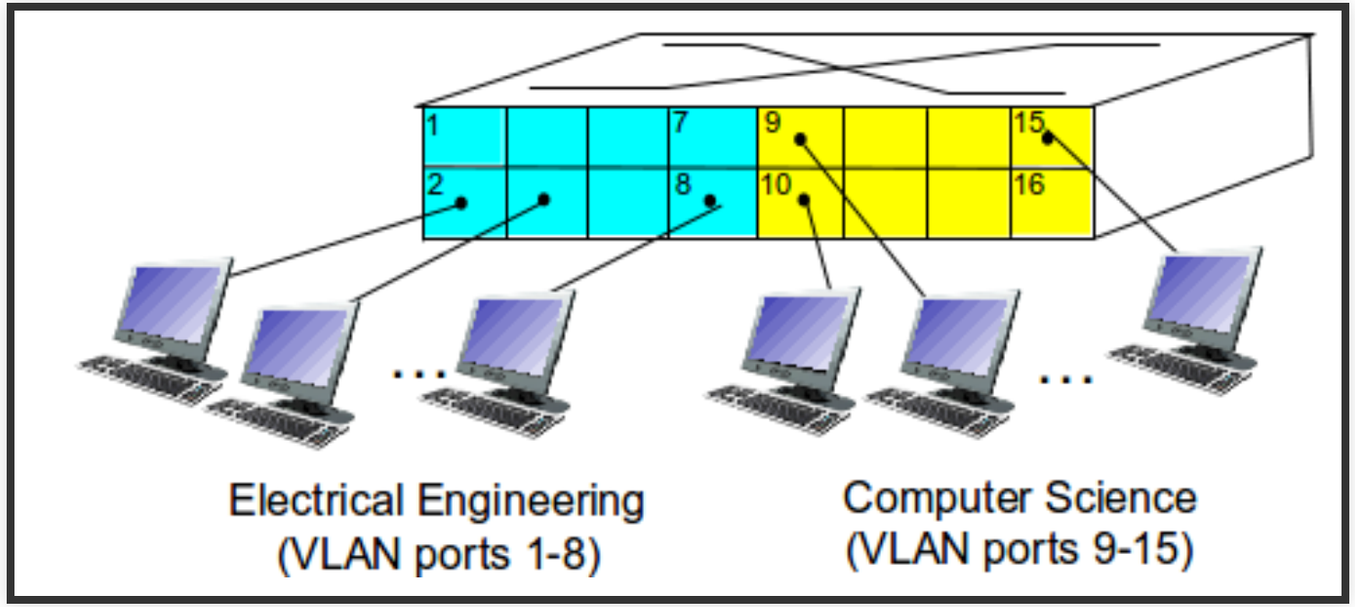
switch(es) supporting VLAN capabilities can be configured to define multiple **virtual** LANS over single physical LAN infrastructure.



PORT-BASED VLAN

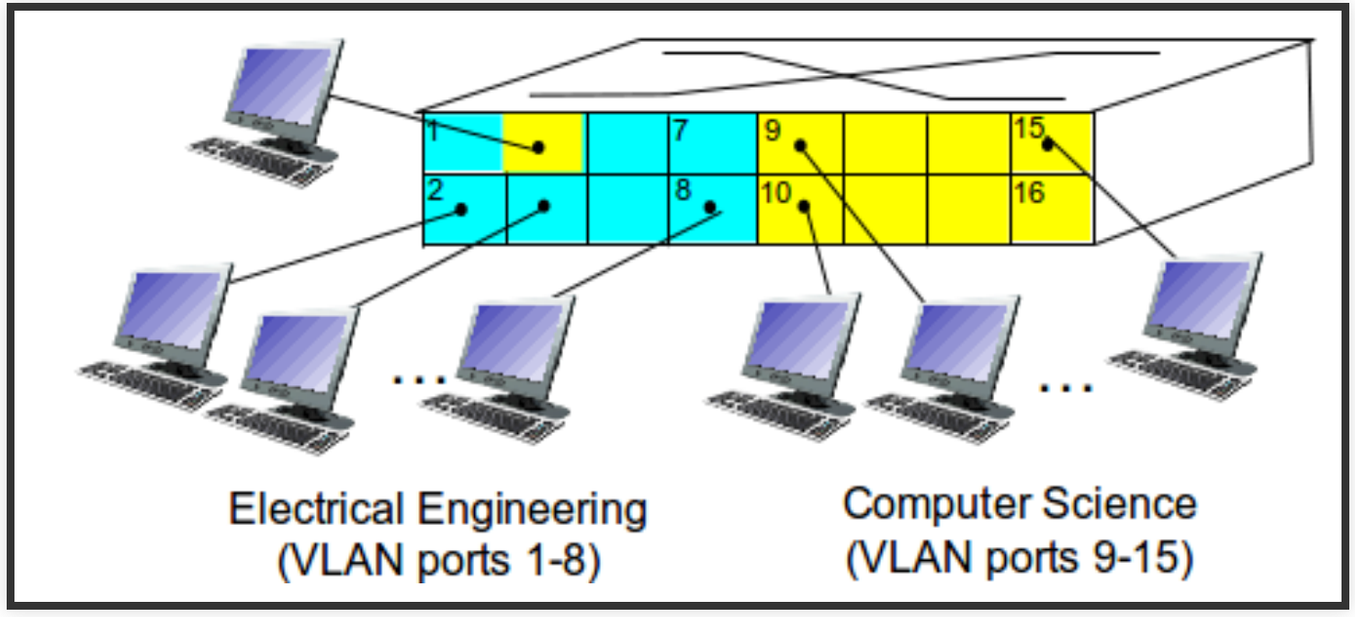
Traffic Isolation: frames to/from ports 1-8 can only reach ports 1-8

- Can also define VLAN based on MAC addresses of endpoints, rather than switch port



PORT-BASED VLAN

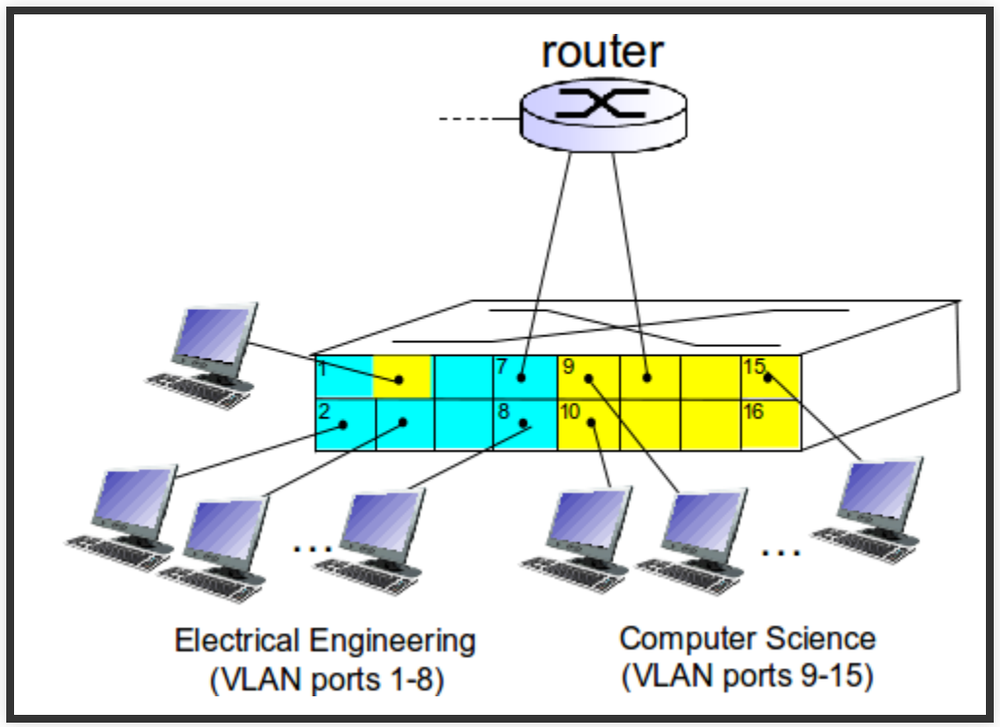
Dynamic membership: ports can be dynamically assigned among VLANs



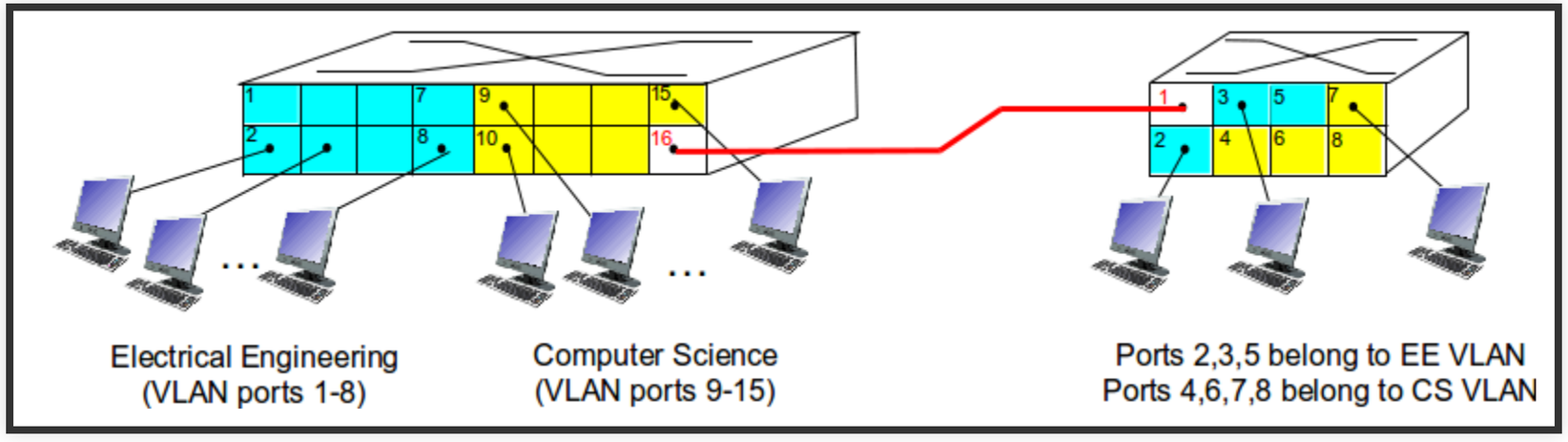
PORT-BASED VLAN

Forwarding between VLANS: Done via routing (just as with separate switches)

- in practice vendors sell combined switches plus routers



VLANS SPANNING MULTIPLE SWITCHES

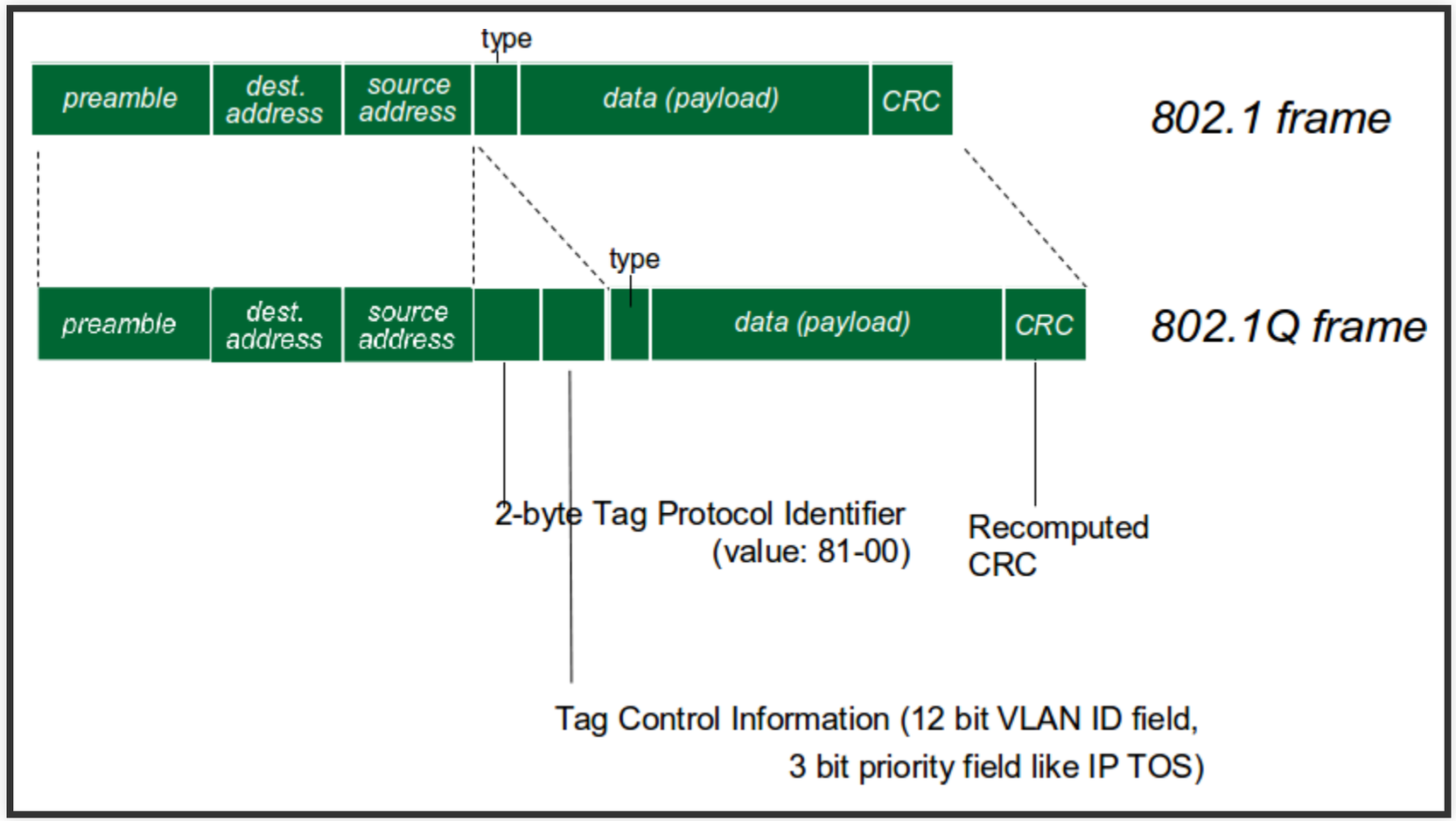


trunk port: carries frames between VLANS defined over multiple physical switches

- frames forwarded within VLAN between switches can’t be vanilla 802.1 frames (must carry VLAN ID info)
- 802.1q protocol adds/removed additional header fields for frames forwarded between trunk ports

VLAN FRAME FORMAT

802.1Q VLAN frame format



LINK VIRTUALIZATION

LINK VIRTUALIZATION

Link Virtualization: A network as a link layer

- Hosts can be blissfully unaware of the connection being to:
  1. Other LAN hosts through single short LAN segment
  2. Geographically dispersed LAN
  3. A VLAN

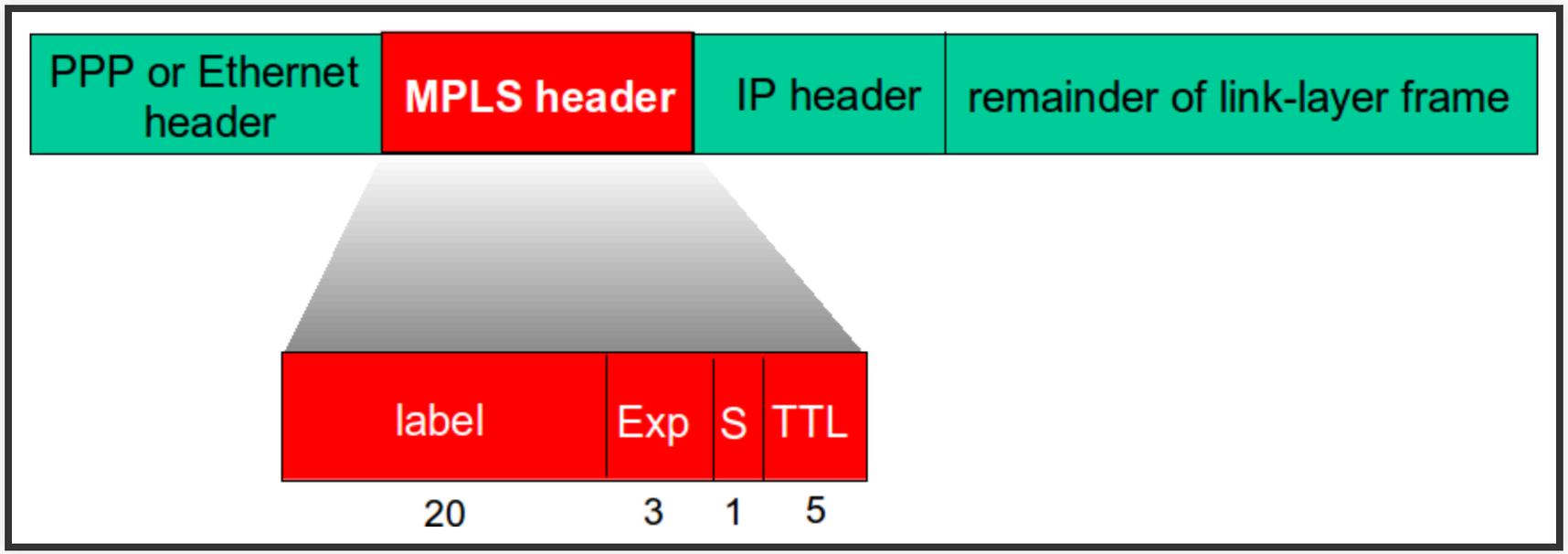
MULTIPROTOCOL LABEL SWITCHING (MPLS)

- Packet-switched virtuel circuit type network
  - Virtuel circuit: Setup fixed connection like in old phone-line networks.
- Has own packet format and forwarding behaviours

MULTIPROTOCOL LABEL SWITCHING (MPLS)

- Initial goal: high-speed IP forwarding using fixed length label (instead of IP address)
  - Fast lookup using fixed length identifier (rather than shortest prefix matching)
  - Borrowing ideas from Virtual Circuit (VC) approach
  - But IP datagram still keeps IP address!

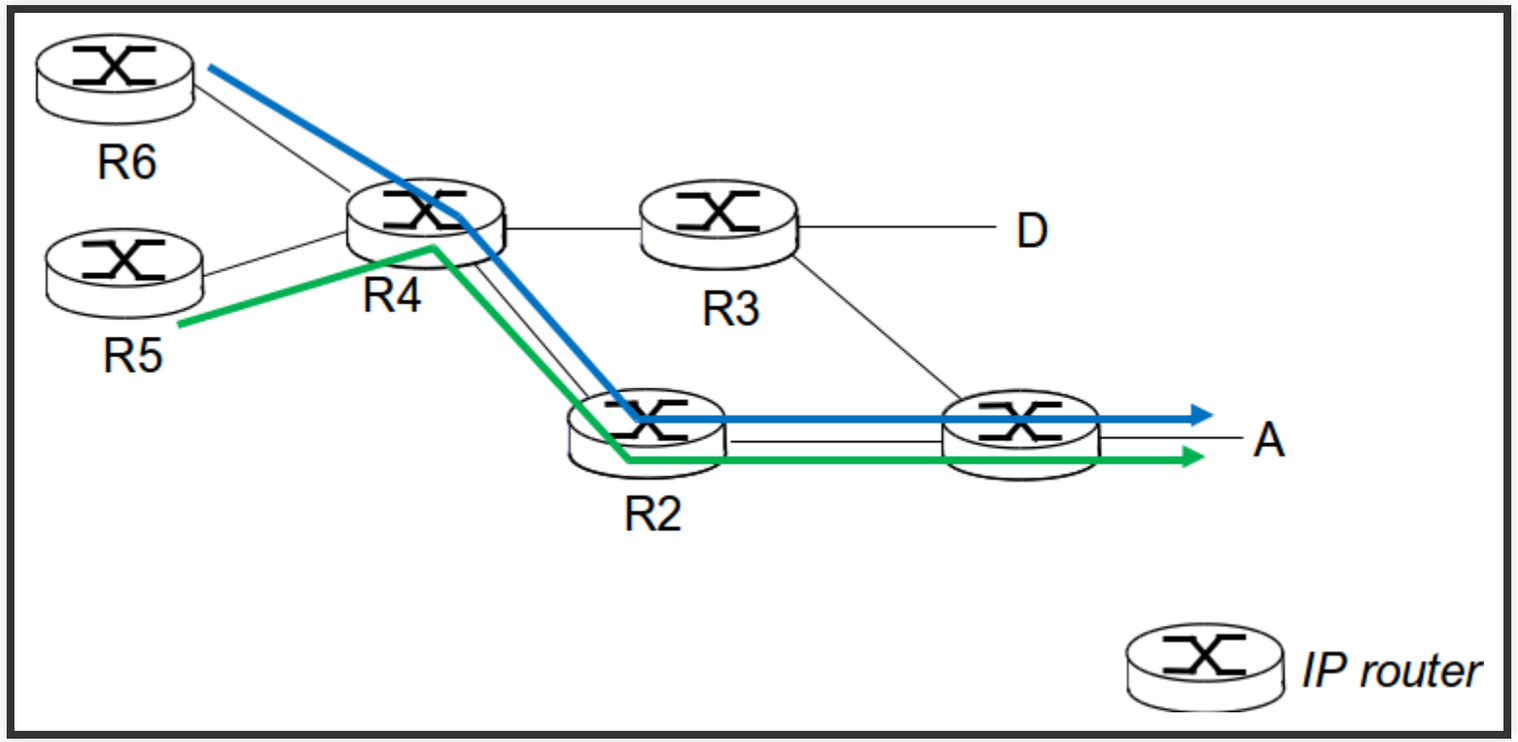
MULTIPROTOCOL LABEL SWITCHING (MPLS)



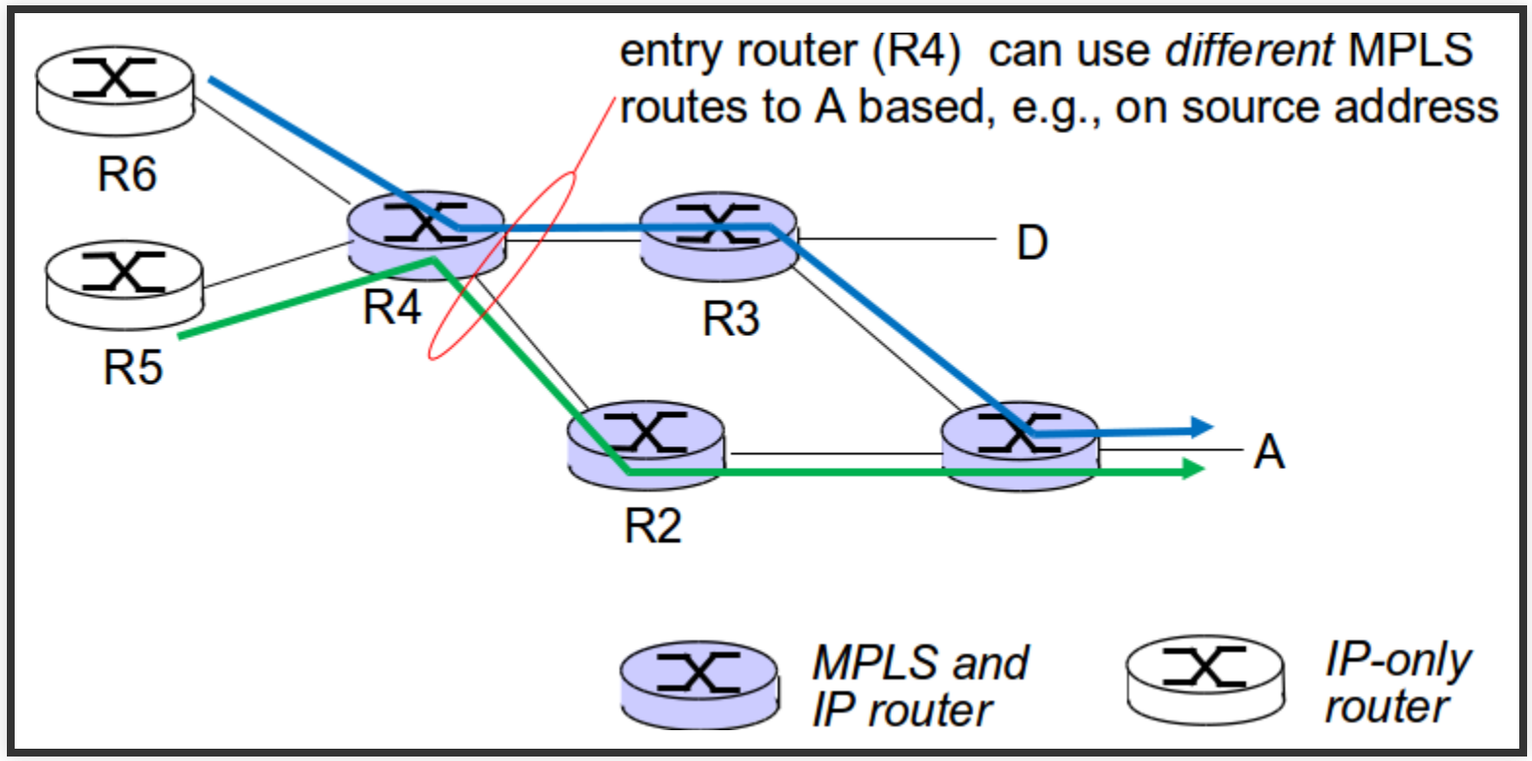
MPLS CAPABLE ROUTERS

- a.k.a. label-switched router
- forward packets to outgoing interface based only on label value (don't inspect IP address)
  - MPLS forwarding table distinct from IP forwarding tables
- **flexibility:** MPLS forwarding decisions can differ from those of IP
  - use destination and source addresses to route flows to same destination differently (traffic engineering)
  - re-route flows quickly if link fails: pre-computed backup paths (useful for VoIP)

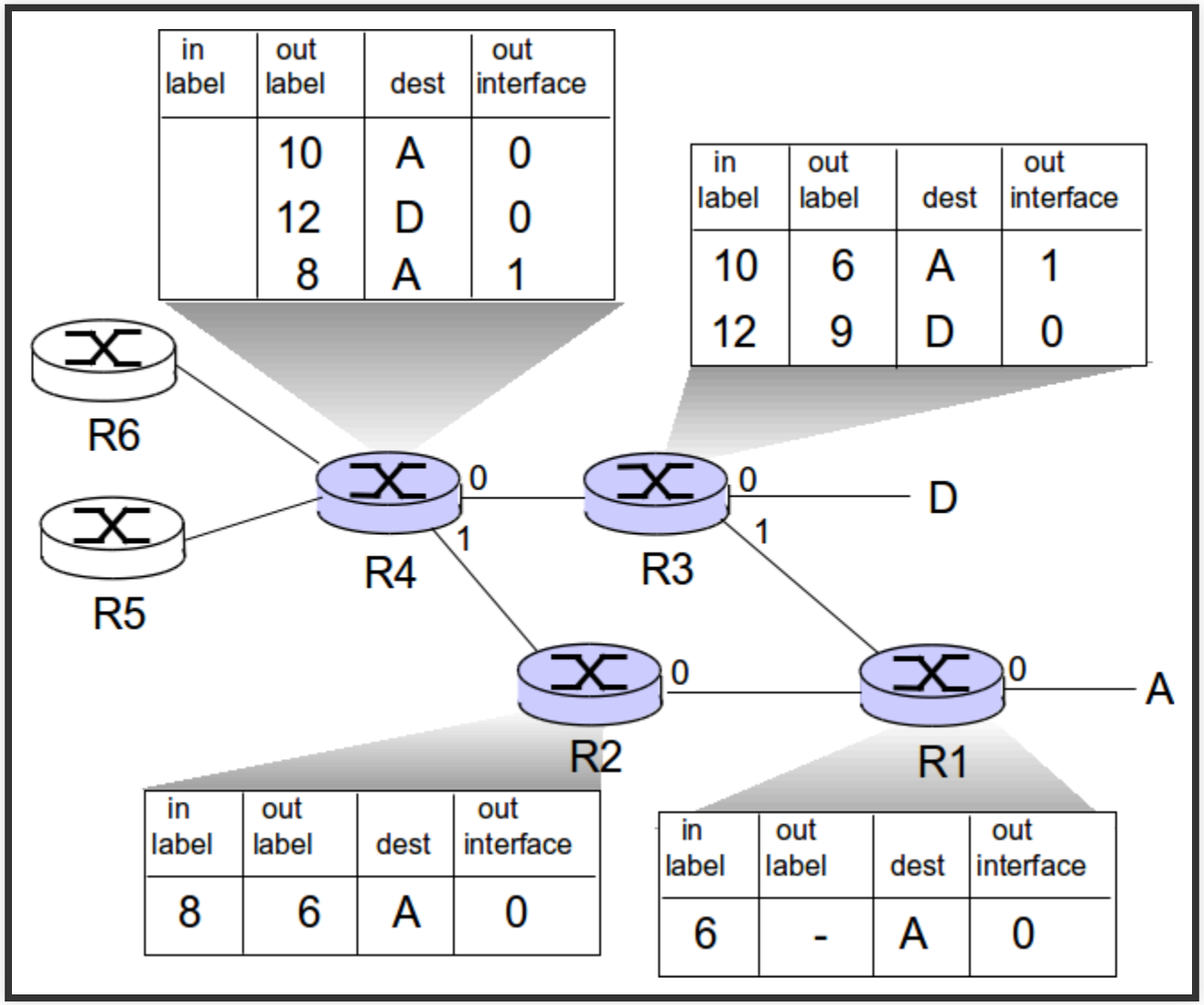
MPLS VERSUS IP PATHS



MPLS VERSUS IP PATHS



MPLS FORWARDING TABLES



DATA CENTER NETWORKING

Inside a 40-ft Microsoft container, Chicago data center

DATA CENTER NETWORKS

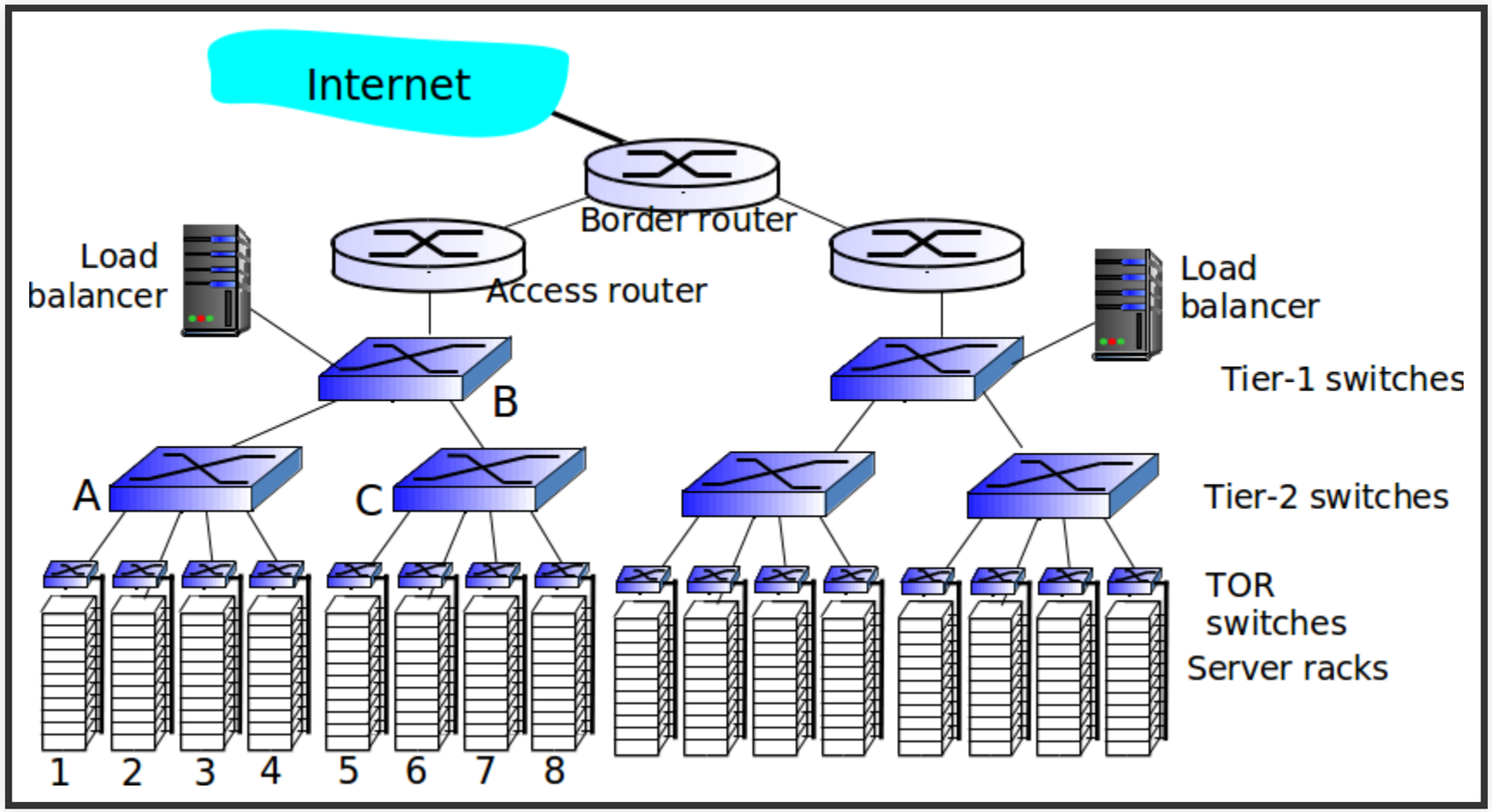
- 10's to 100's of thousands of hosts, often closely coupled, in close proximity:
  - e-business (e.g. Amazon)
  - content-servers (e.g., YouTube, Akamai, Apple, Microsoft)
  - search engines, data mining (e.g., Google)
  - Co-location (one data center hosting servers for many different companies)
- challenges:
  - multiple applications, each serving massive numbers of clients
  - managing/balancing load, avoiding processing, networking, data bottlenecks

DATA CENTER NETWORKS

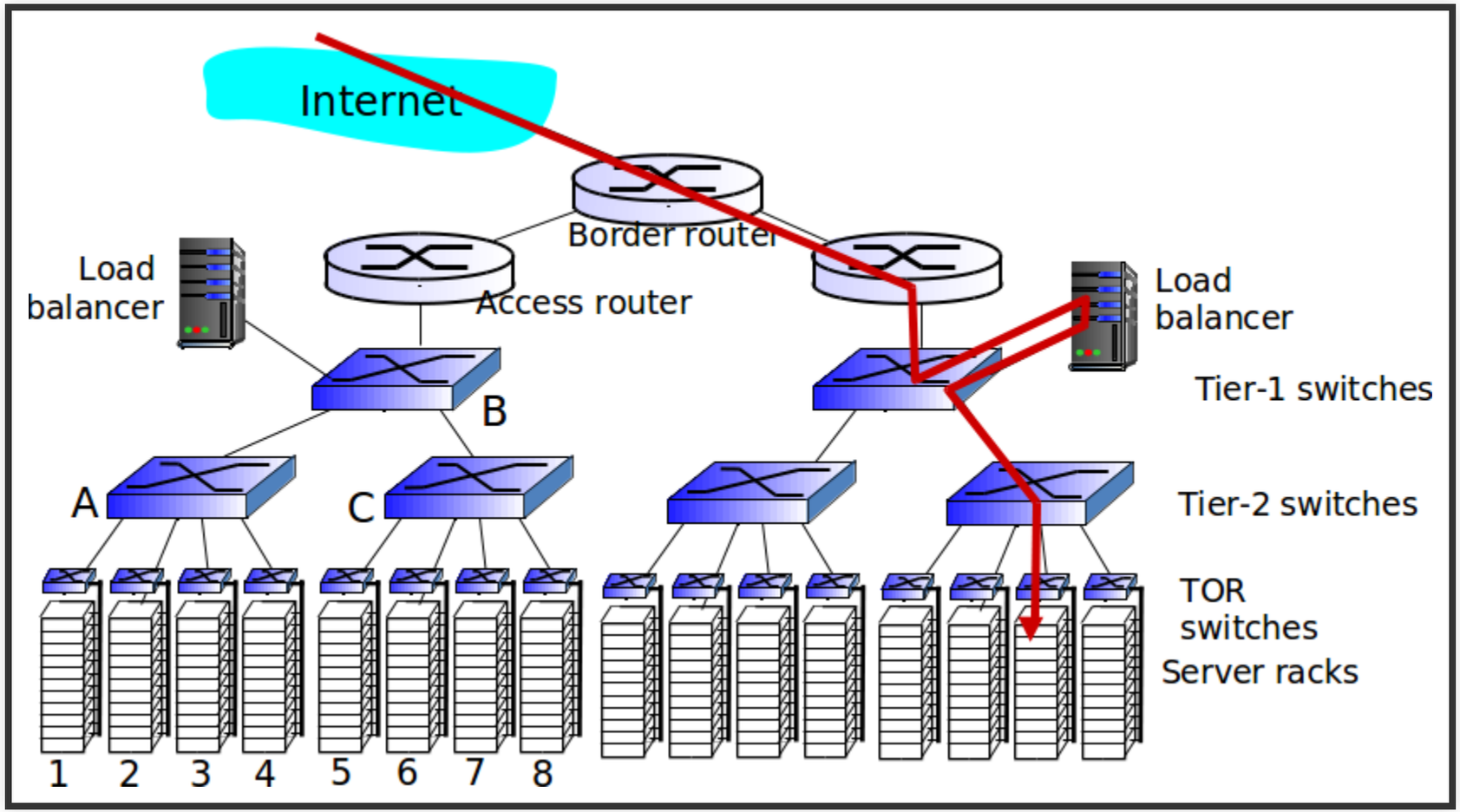
load balancer: application-layer routing

- receives external client requests
- directs workload within data center
- returns results to external client (hiding data center internals from client)

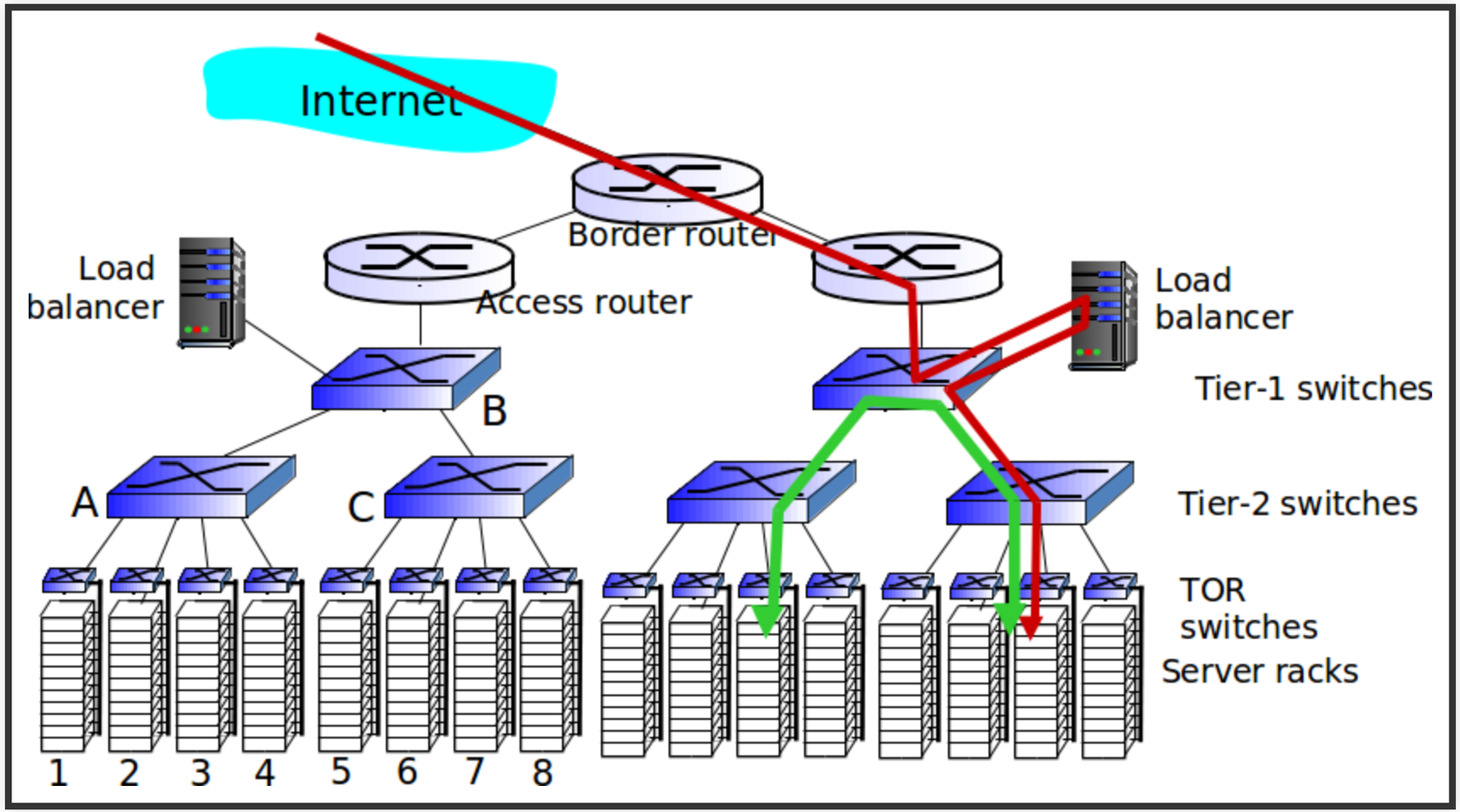
DATA CENTER NETWORKS



DATA CENTER NETWORKS

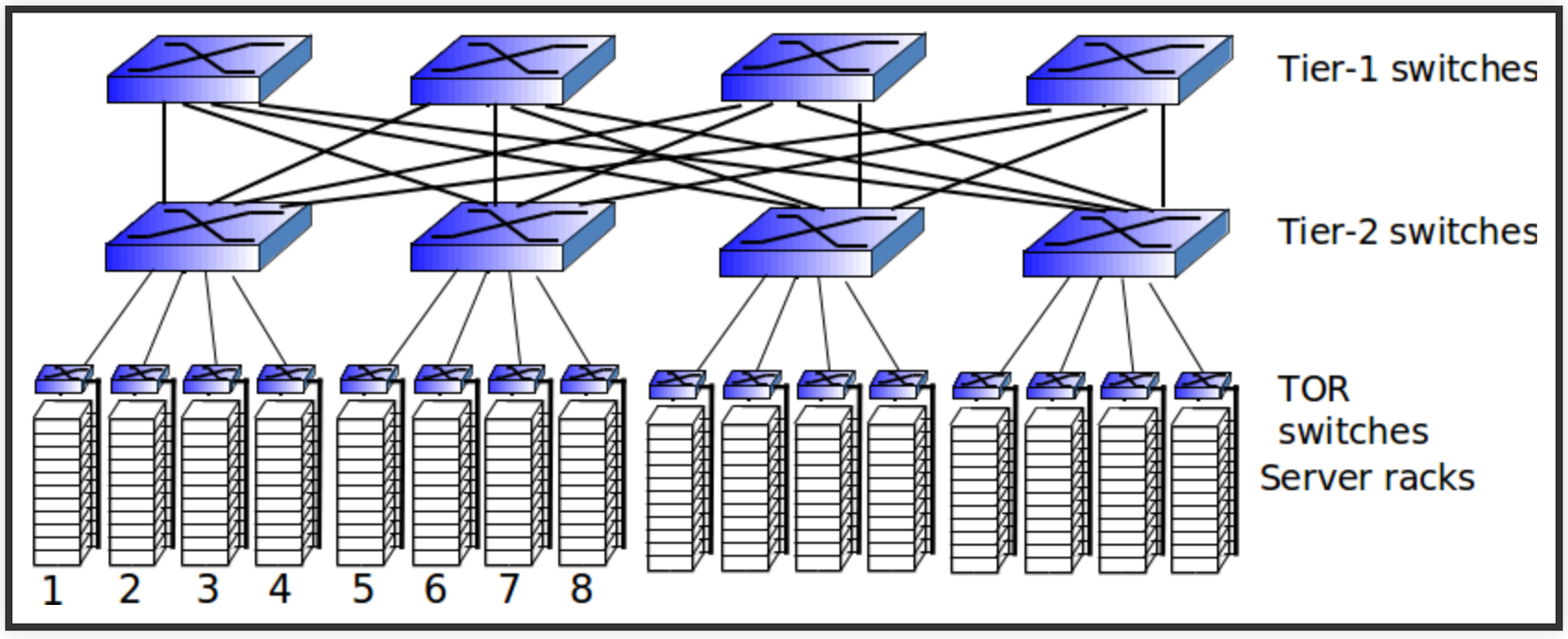


DATA CENTER NETWORKS

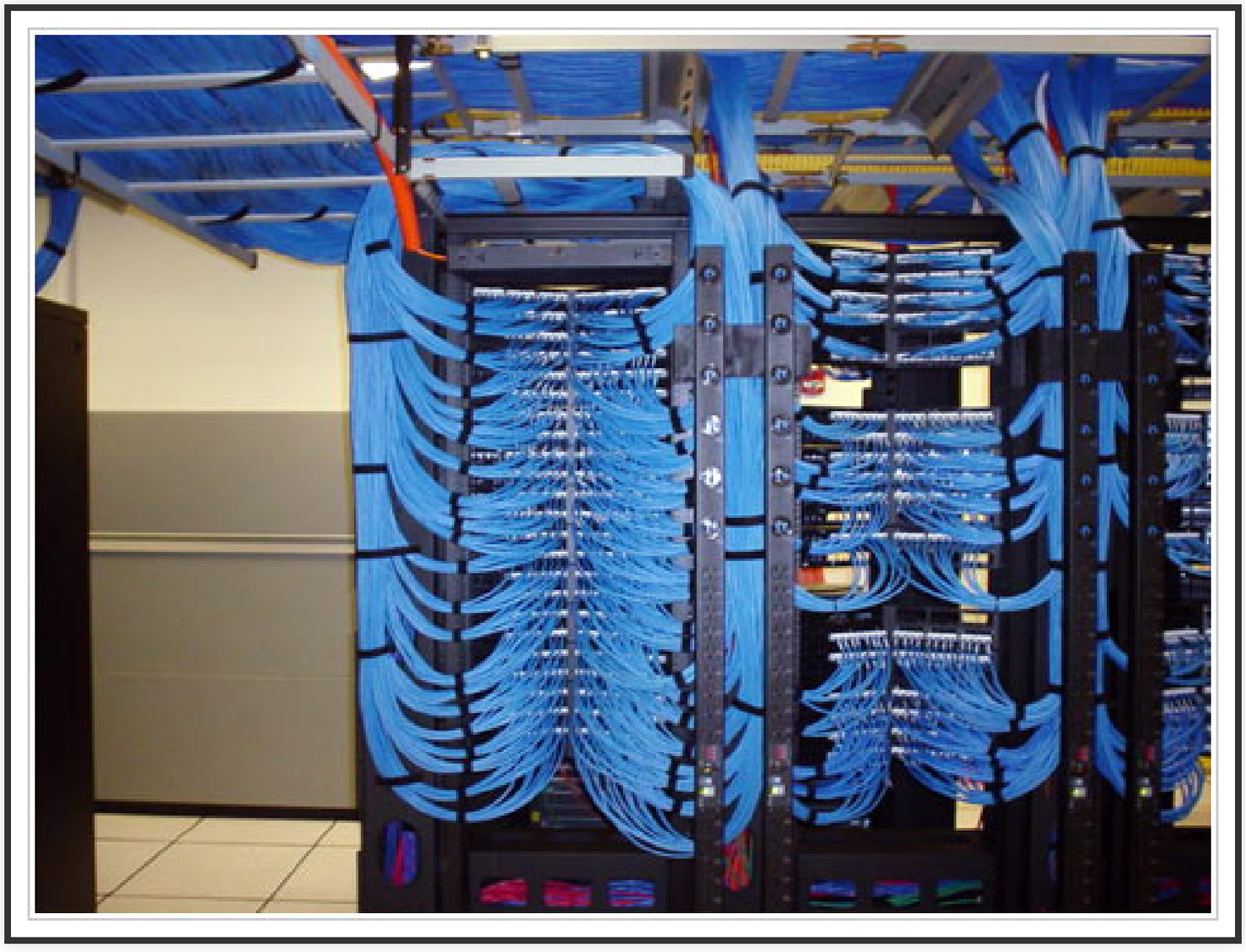


DATA CENTER NETWORKS

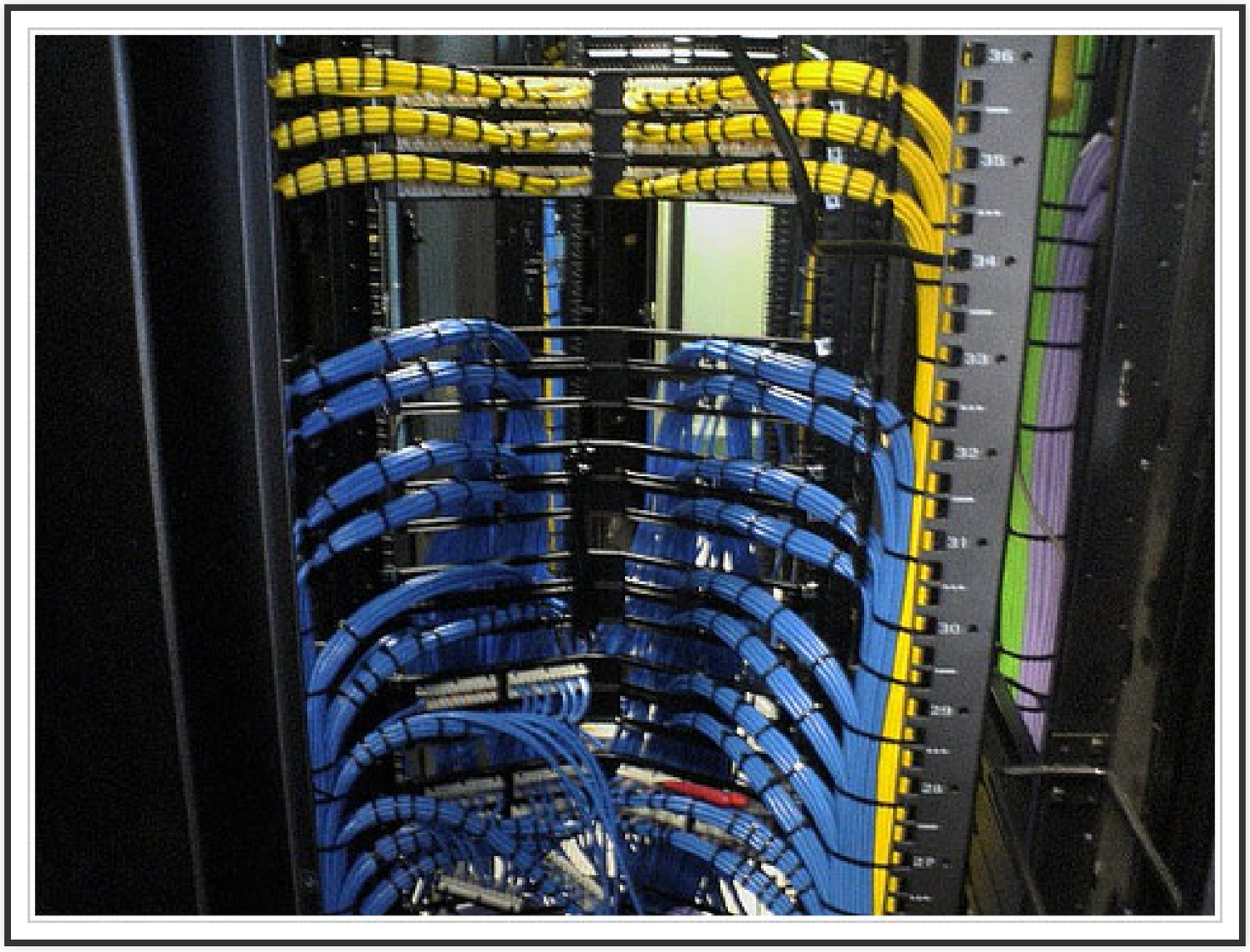
- rich interconnection among switches, racks:
  - increased throughput between racks (multiple routing paths possible)
  - increased reliability via redundancy



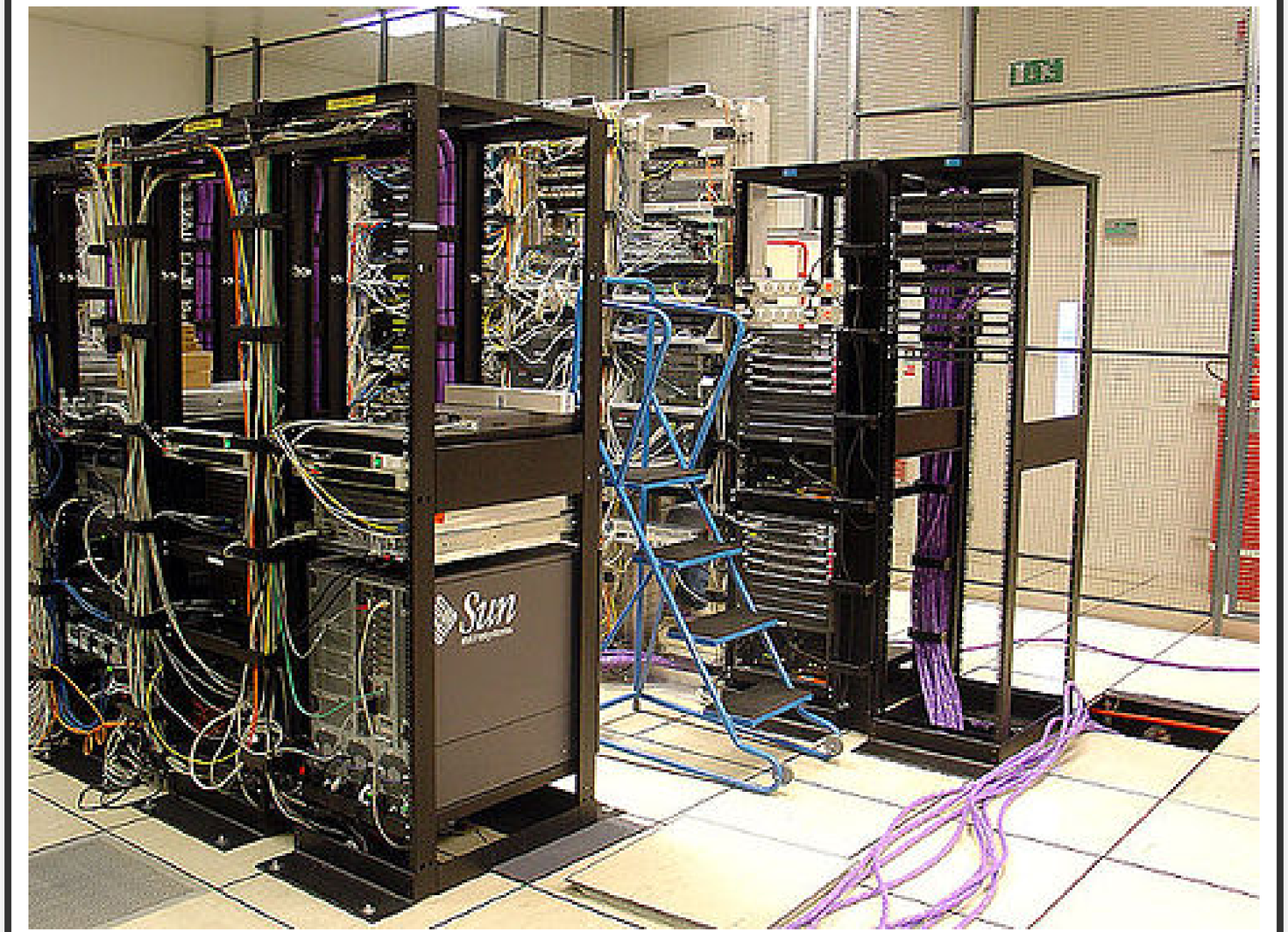
NETWORK CABLING EXAMPLES



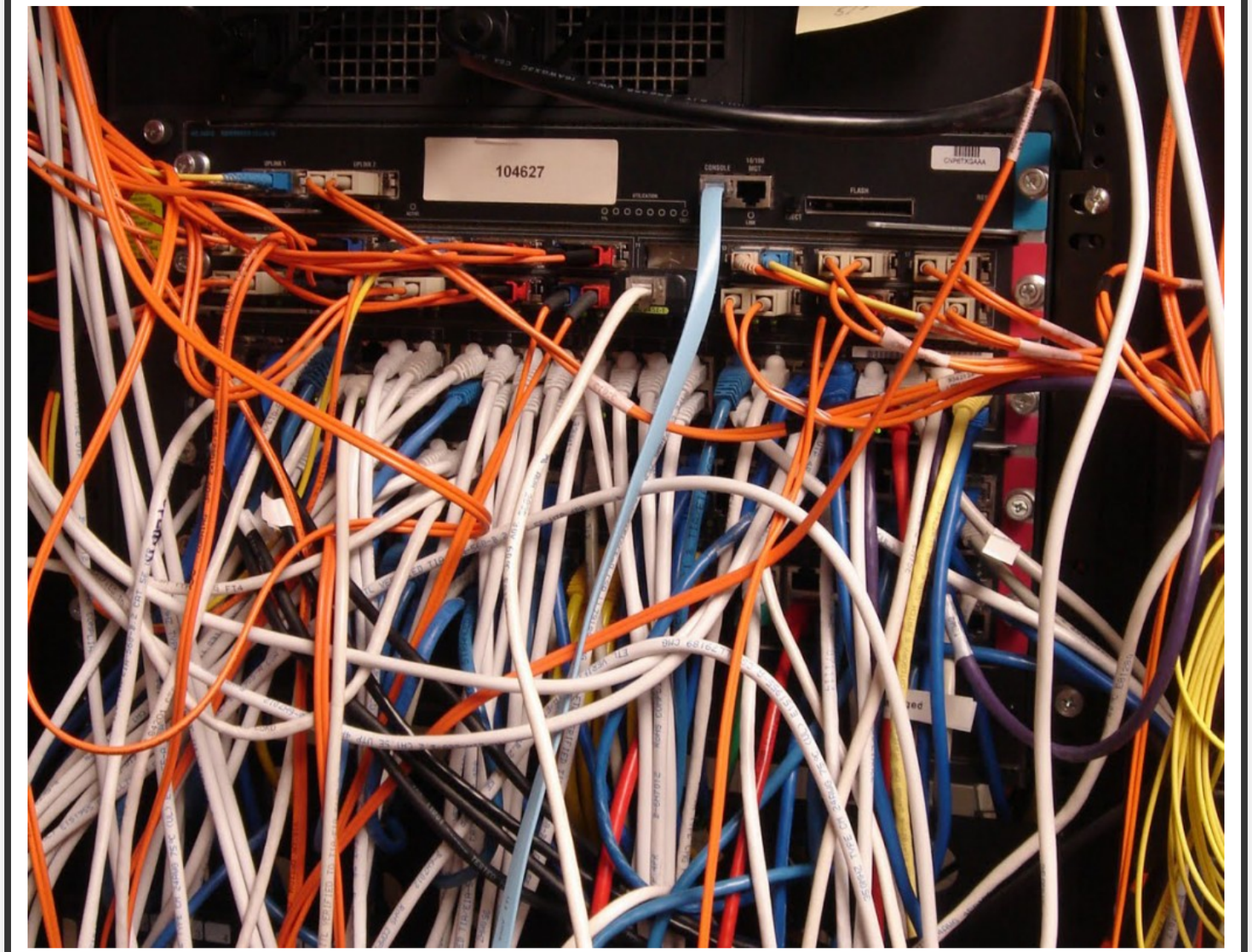
NETWORK CABLING EXAMPLES



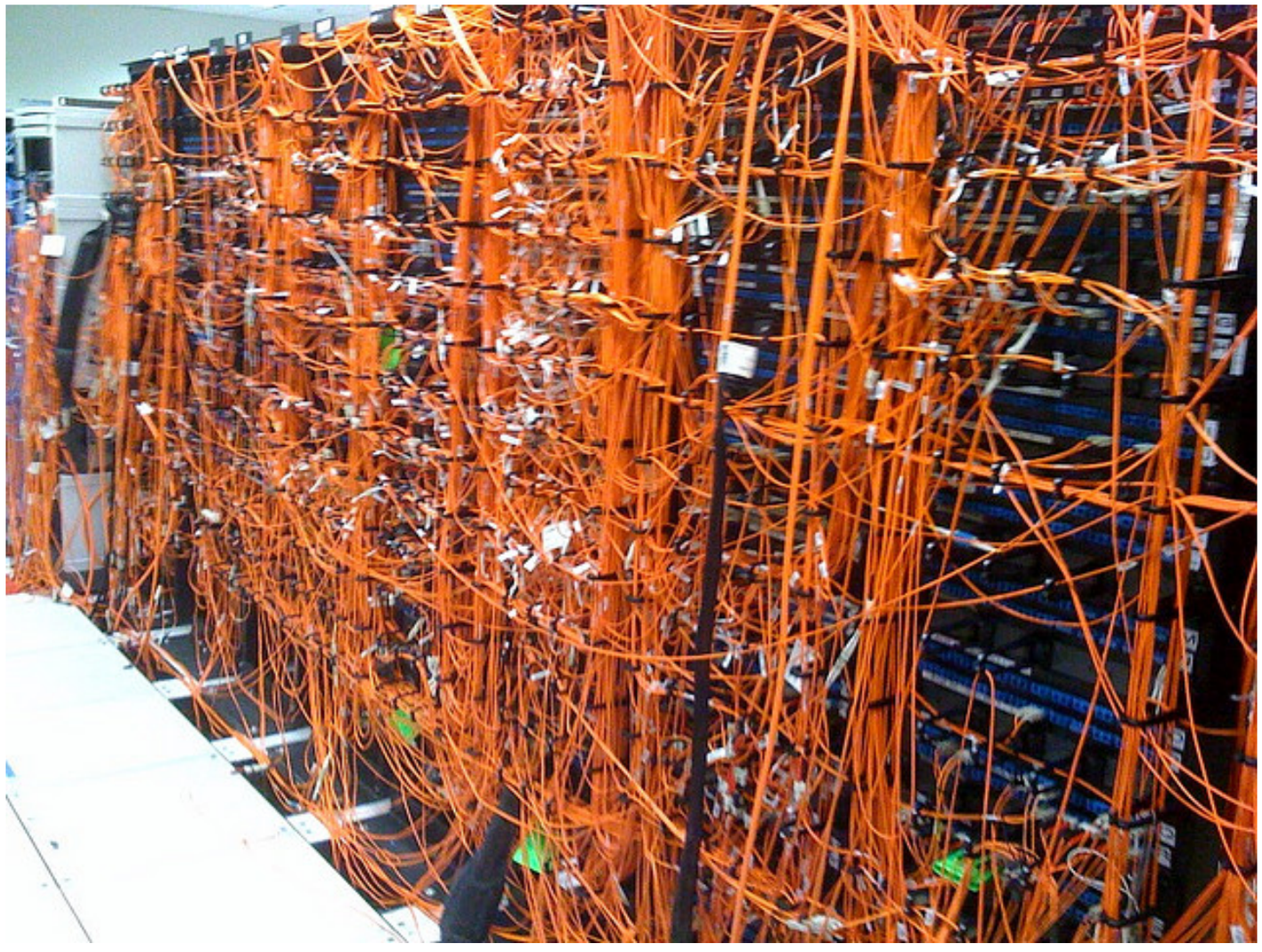
NETWORK CABLING EXAMPLES



NETWORK CABLING EXAMPLES

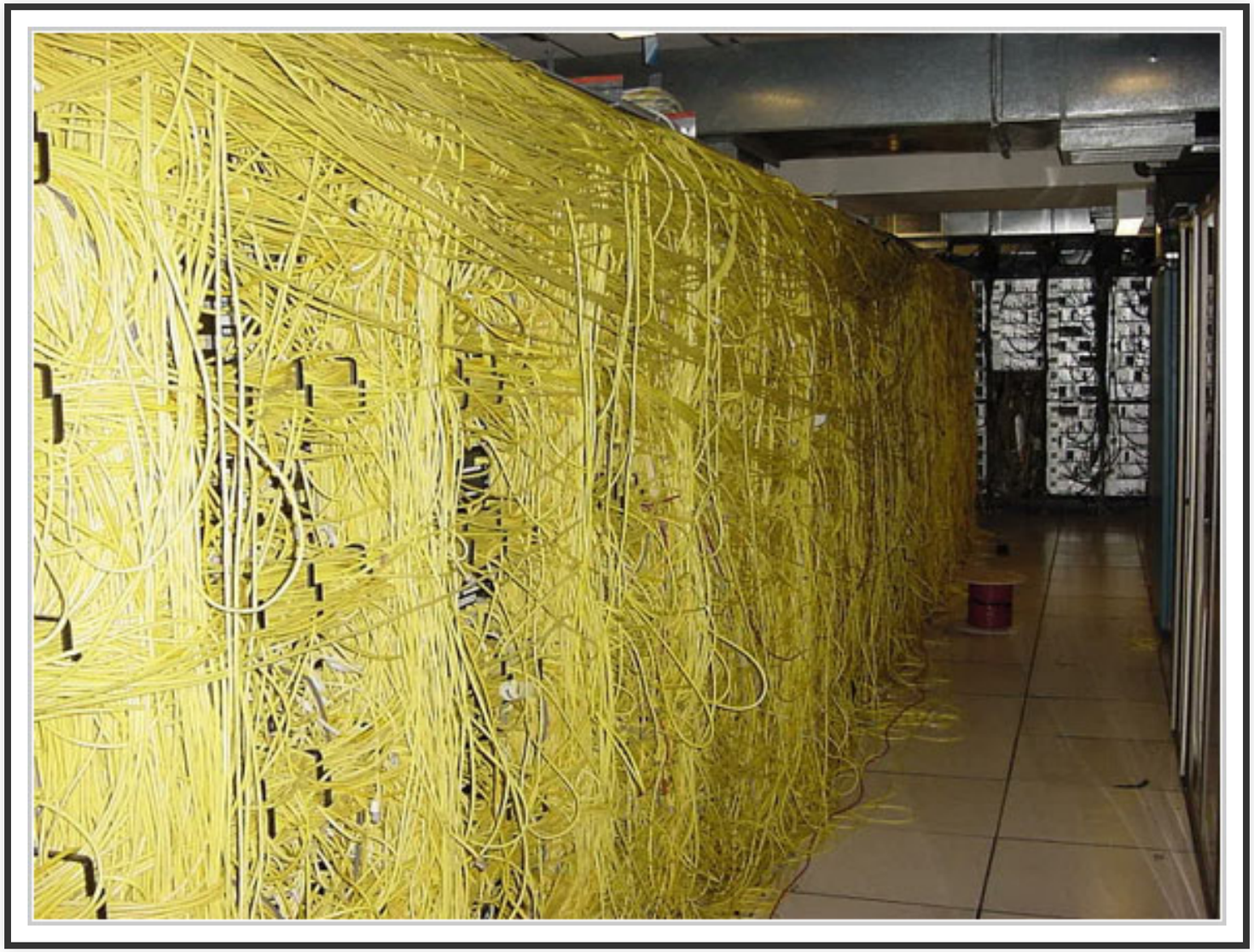


NETWORK CABLING EXAMPLES



NETWORK CABLING EXAMPLES

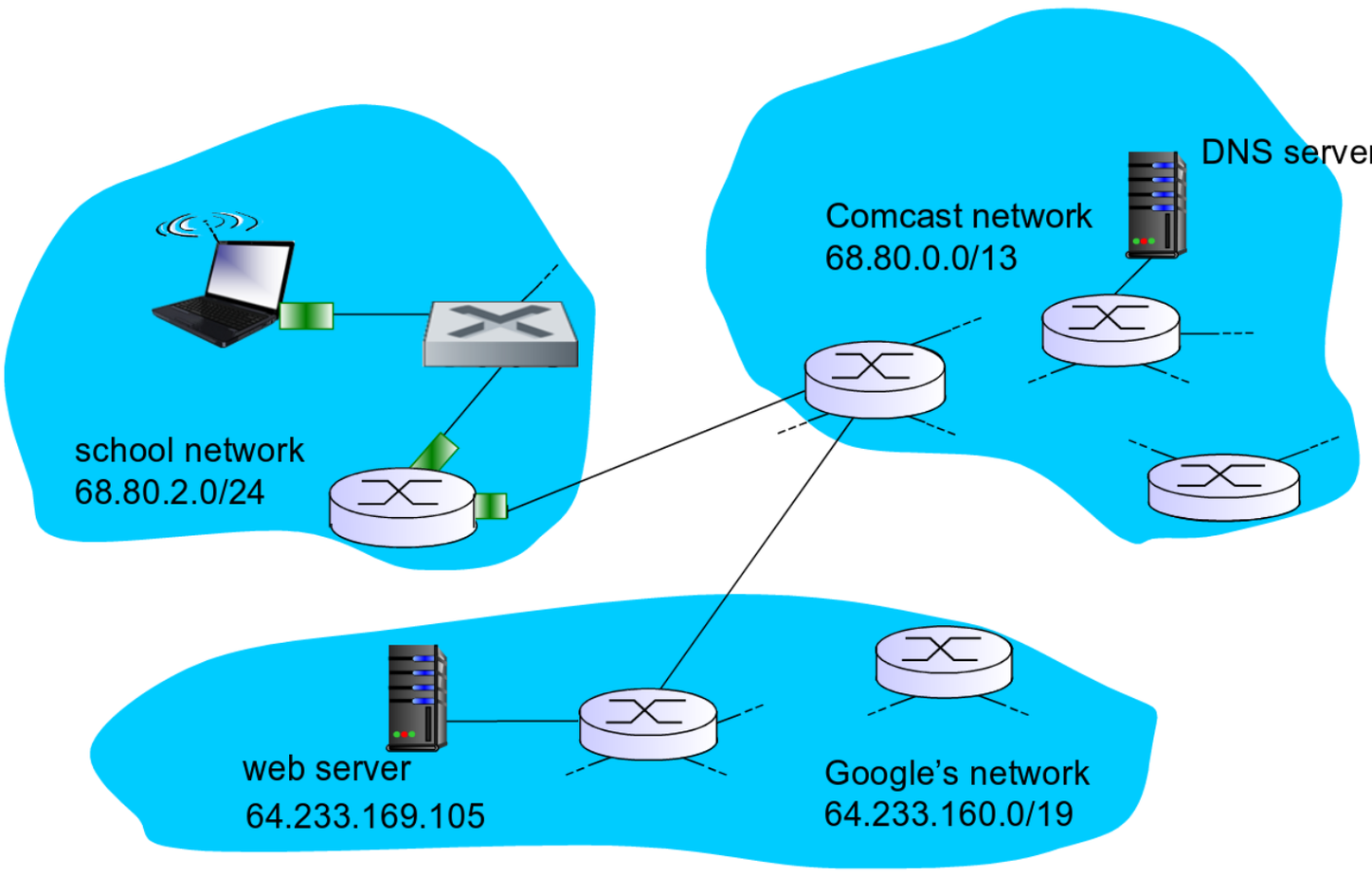
NETWORK CABLING EXAMPLES



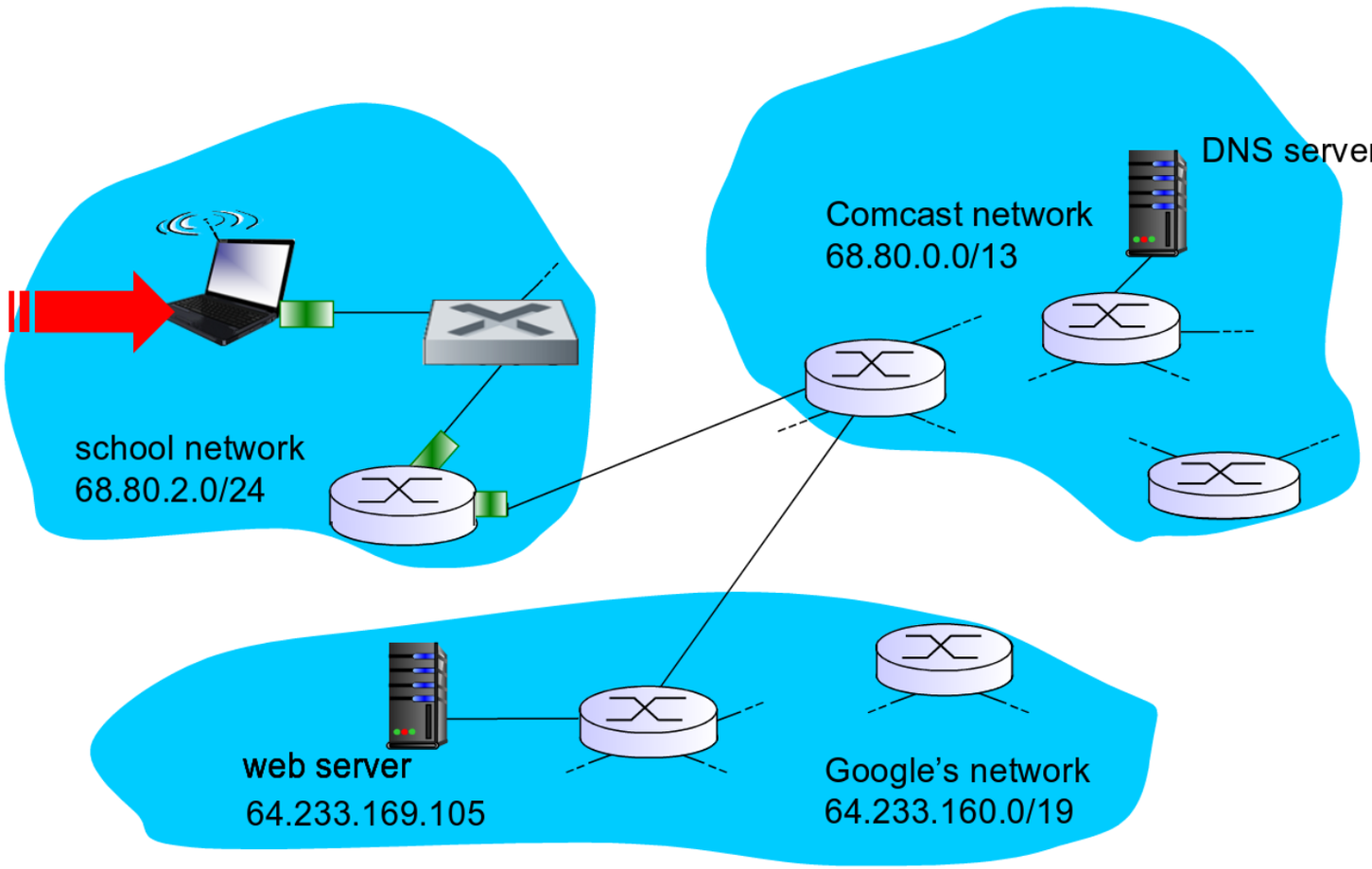
SYNTHESIS: A DAY IN THE LIFE OF A WEB REQUEST

- Journey down protocol stack complete!
  - Application, transport, network, link
- Putting-it-all-together: synthesis!
  - **Goal:** identify, review, understand protocols (at all layers) involved in seemingly simple scenario: *Requesting a webpage*
  - **Scenario:** student attaches laptop to campus network, requests/receives www.google.com

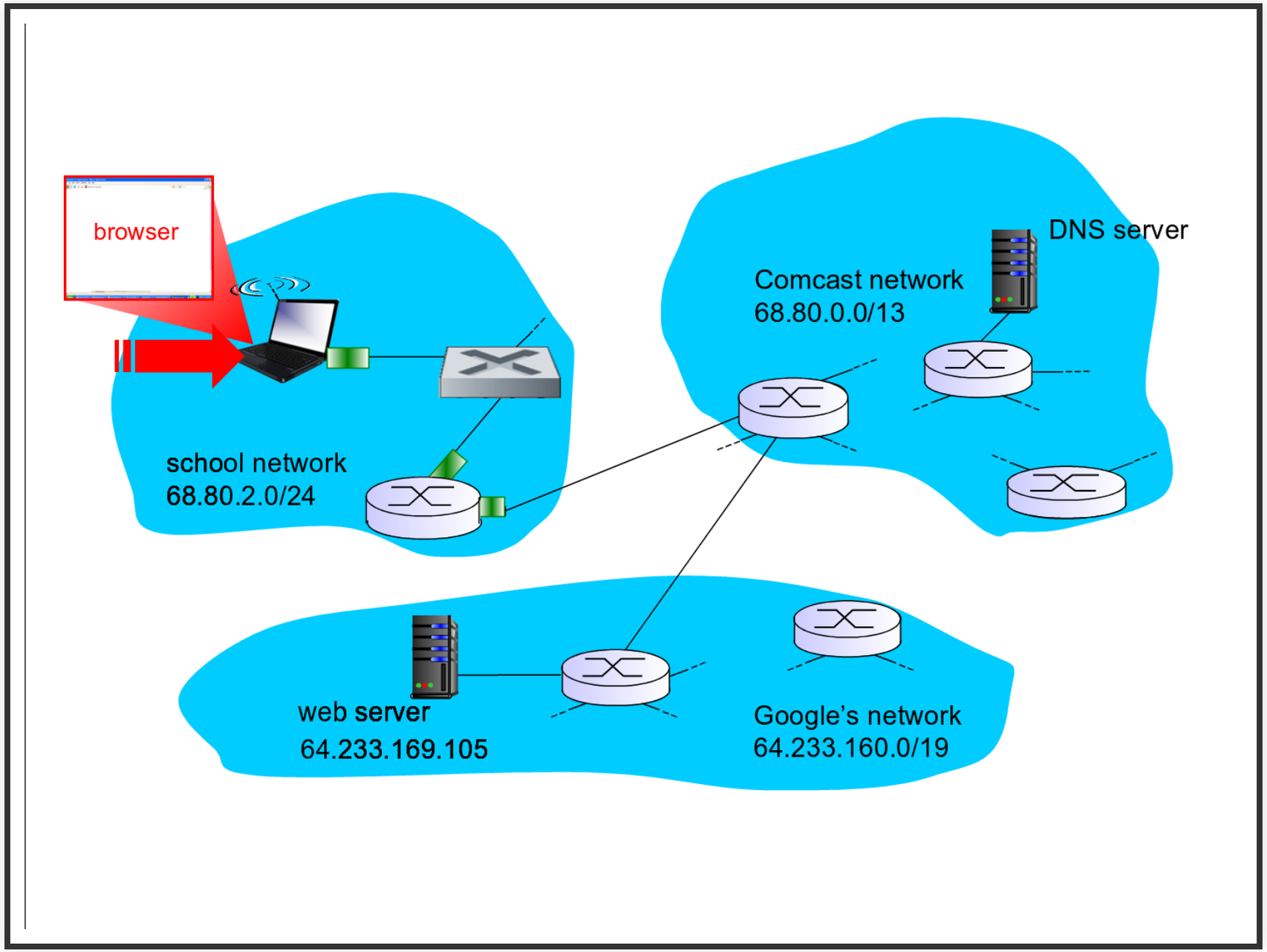
SCENARIO


DNS server
Comcast network
68.80.0.0/13
school network
68.80.2.0/24
web server
64.233.169.105
Google's network
64.233.160.0/19

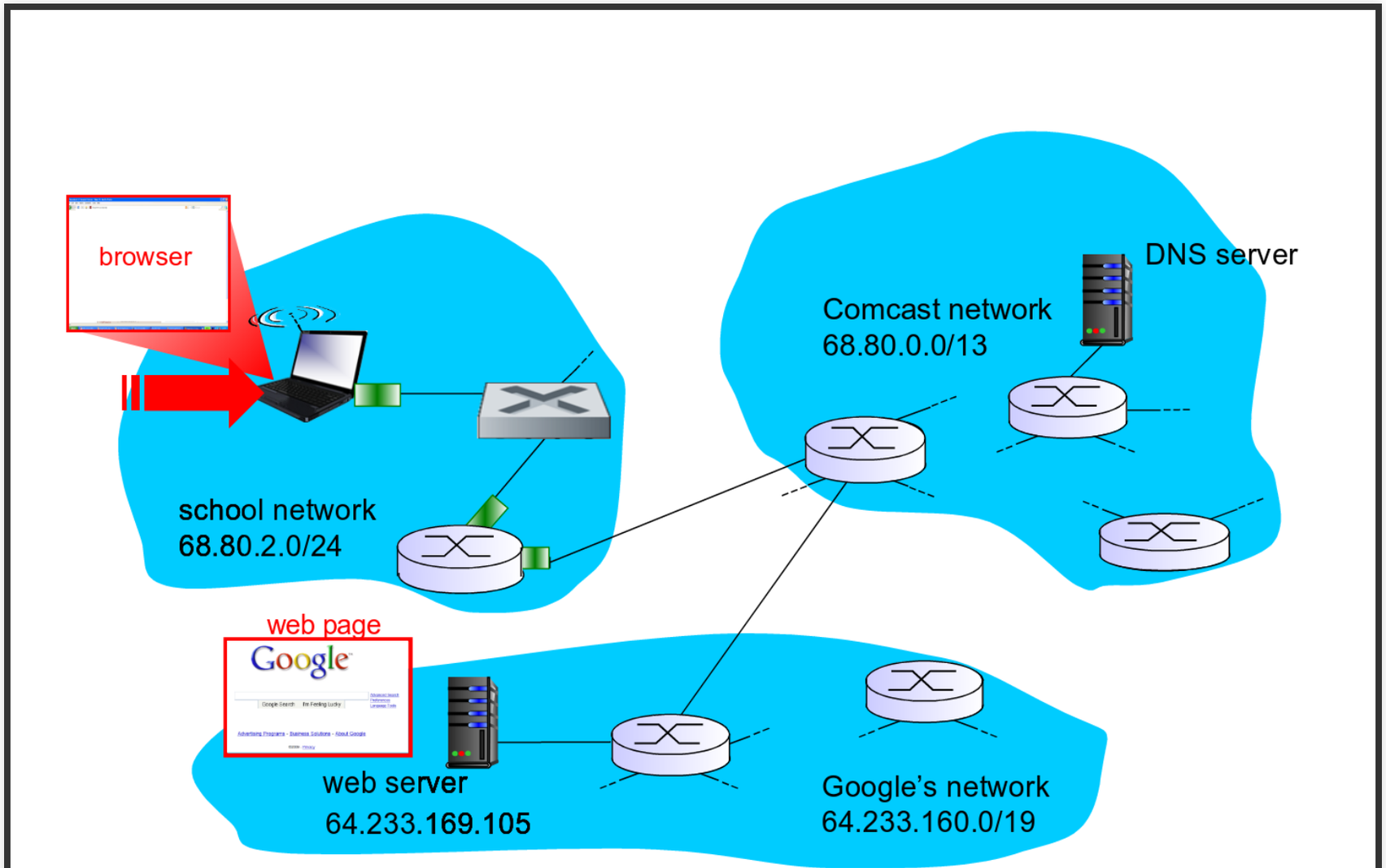
SCENARIO


DNS server
Comcast network
68.80.0.0/13
school network
68.80.2.0/24
web server
64.233.169.105
Google's network
64.233.160.0/19

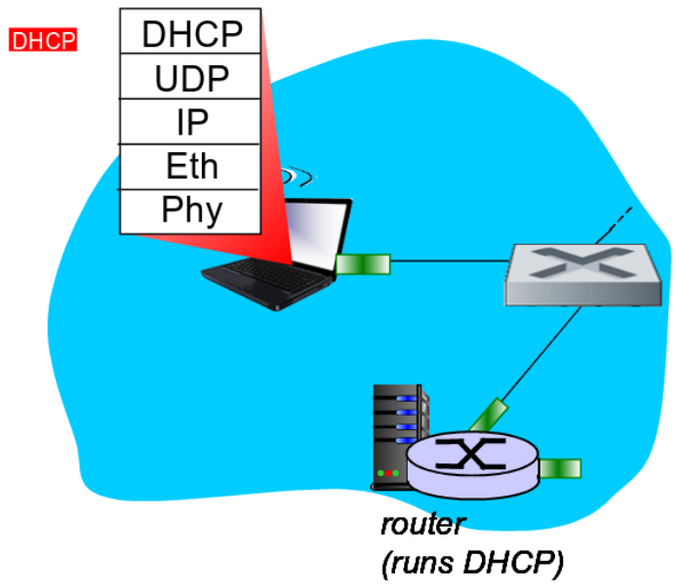
SCENARIO


browser
school network
68.80.2.0/24
Comcast network
68.80.0.0/13
DNS server
web server
64.233.169.105
Google's network
64.233.160.0/19

SCENARIO

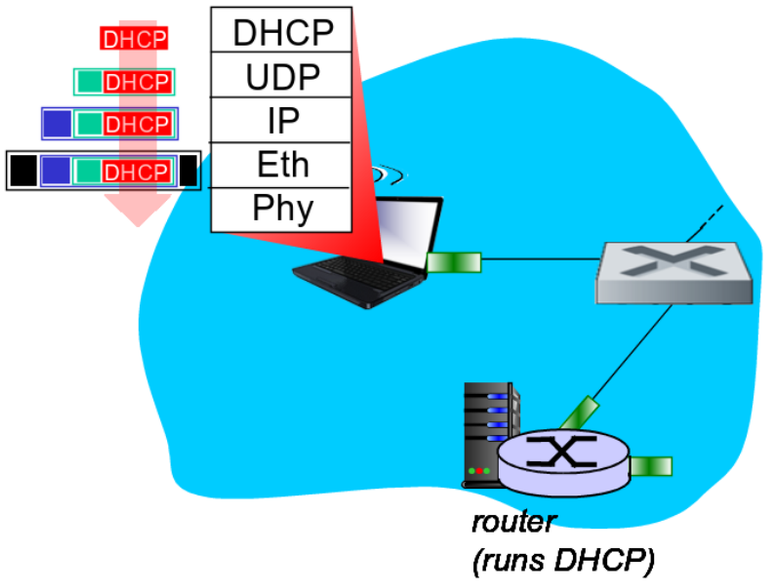

browser
school network
68.80.2.0/24
Comcast network
68.80.0.0/13
DNS server
web page
Google
web server
64.233.169.105
Google's network
64.233.160.0/19

CONNECTING TO THE INTERNET



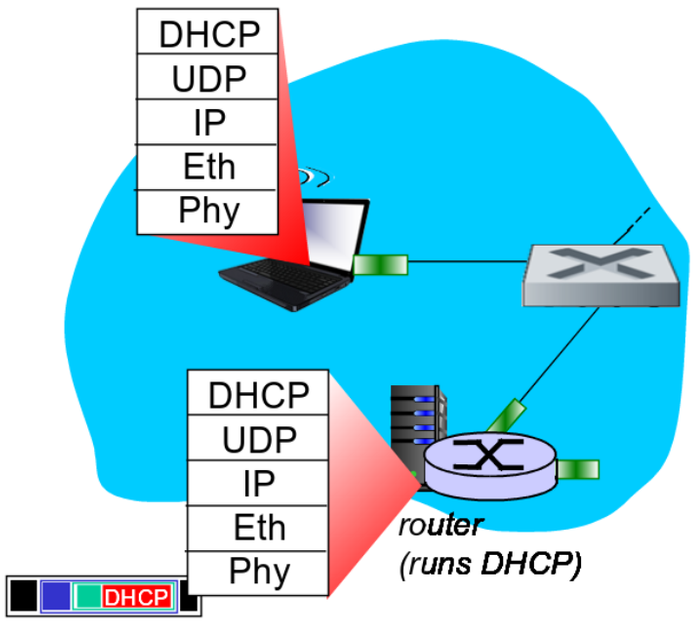
- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use *DHCP*

CONNECTING TO THE INTERNET



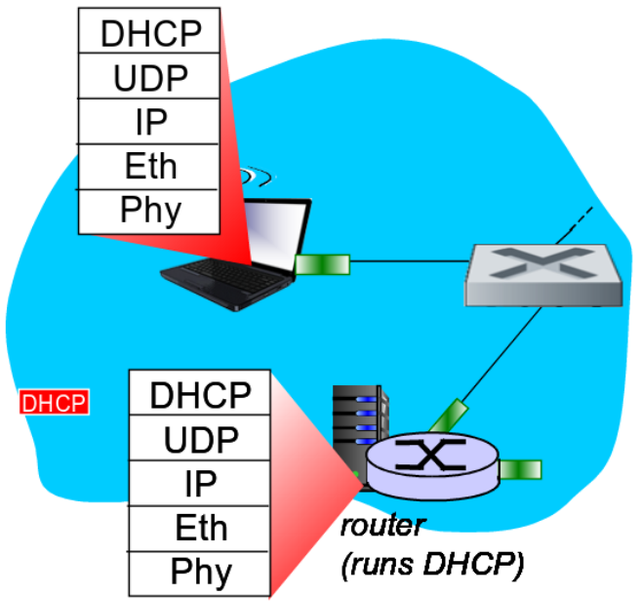
- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use *DHCP*
- DHCP request *encapsulated* in *UDP*, encapsulated in *IP*, encapsulated in *802.3* Ethernet

CONNECTING TO THE INTERNET



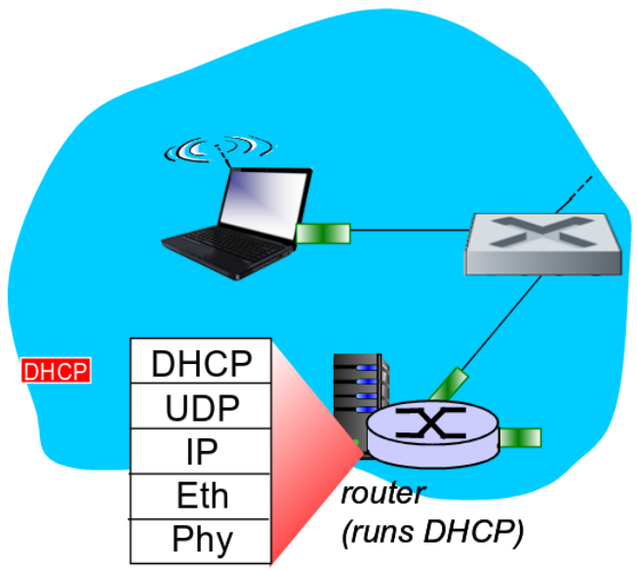
- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use *DHCP*
- DHCP request *encapsulated* in *UDP*, encapsulated in *IP*, encapsulated in *802.3* Ethernet
- Ethernet frame *broadcast* (dest: FFFFFFFFFFFF) on LAN, received at router running *DHCP* server

CONNECTING TO THE INTERNET



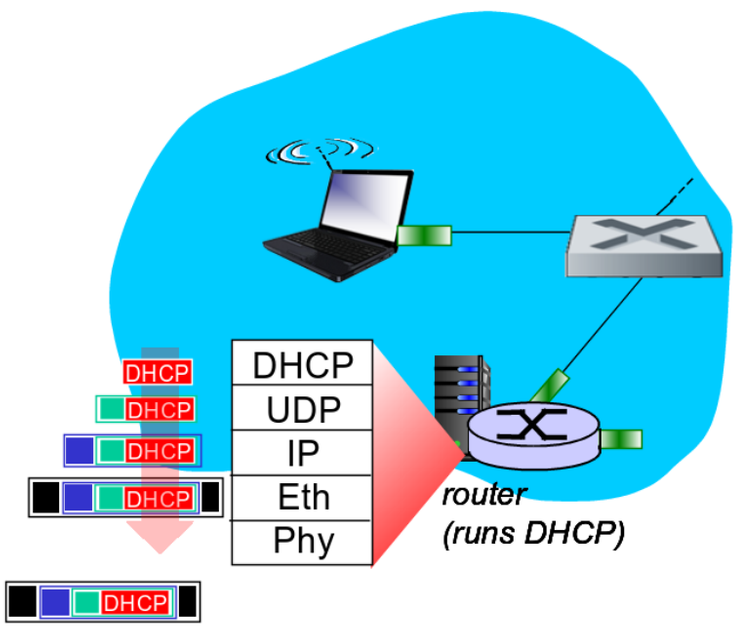
- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use *DHCP*
- DHCP request *encapsulated* in *UDP*, encapsulated in *IP*, encapsulated in *802.3* Ethernet
- Ethernet frame *broadcast* (dest: FFFFFFFFFFFF) on LAN, received at router running *DHCP* server
- Ethernet *demuxed* to IP demuxed, UDP demuxed to DHCP

CONNECTING TO THE INTERNET



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

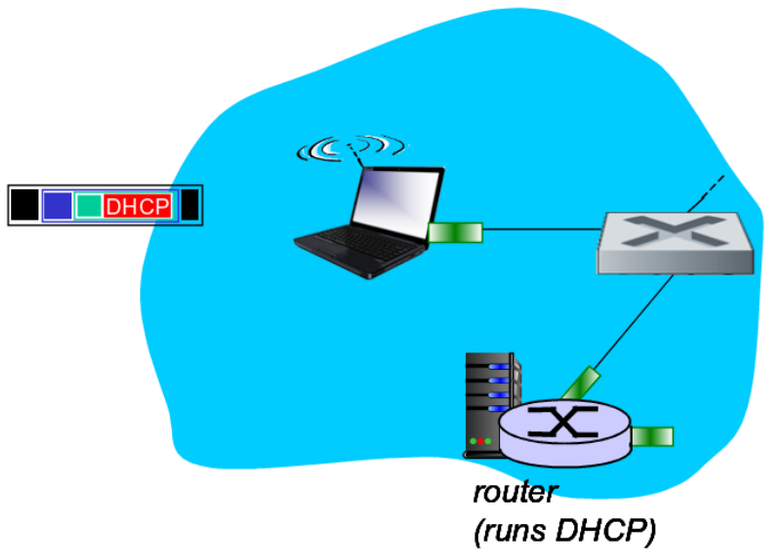
CONNECTING TO THE INTERNET



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

- encapsulation at DHCP server, frame forwarded (*switch learning*) through LAN, demultiplexing at client

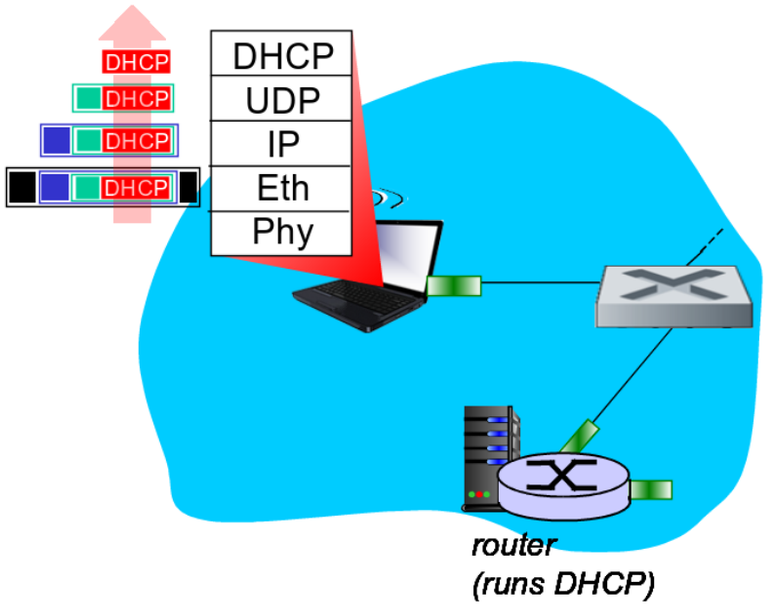
CONNECTING TO THE INTERNET



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

- encapsulation at DHCP server, frame forwarded (*switch learning*) through LAN, demultiplexing at client

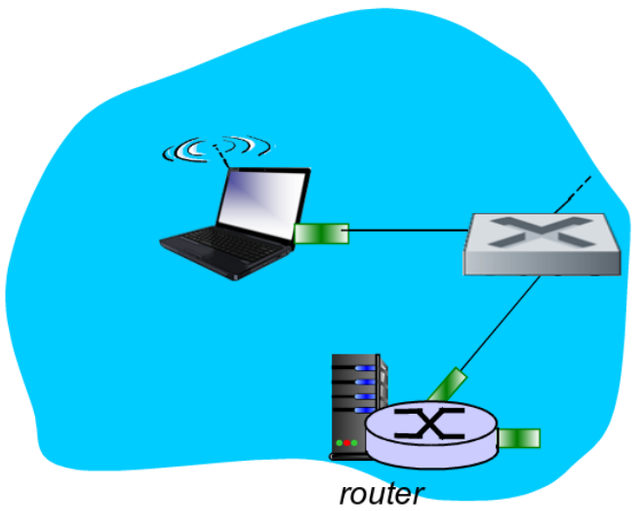
CONNECTING TO THE INTERNET



- DHCP server formulates *DHCP ACK* containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation at DHCP server, frame forwarded (*switch learning*) through LAN, demultiplexing at client
- DHCP client receives DHCP ACK reply

Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

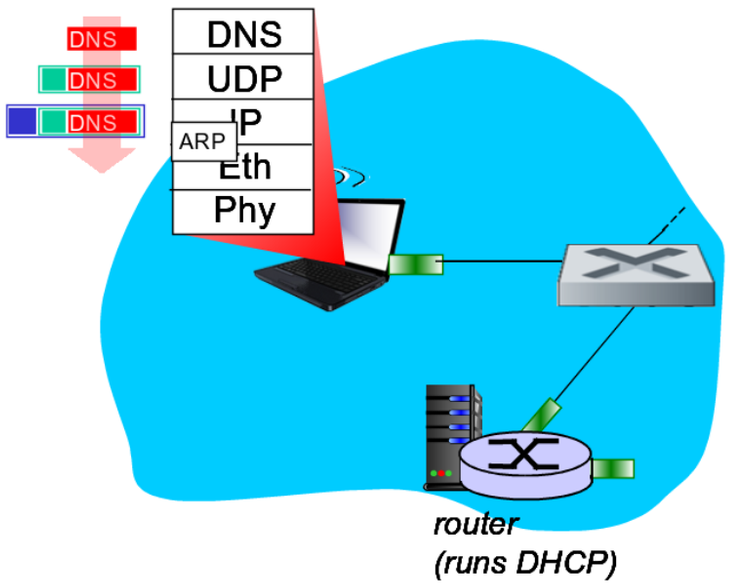
ARP (BEFORE DNS, BEFORE HTTP)



router
(runs DHCP)

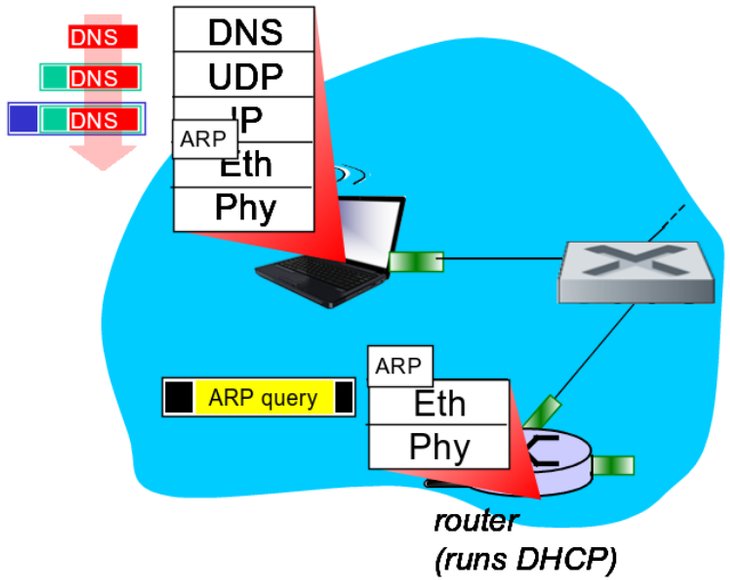
- before sending *HTTP* request, need IP address of www.google.com: *DNS*

ARP (BEFORE DNS, BEFORE HTTP)



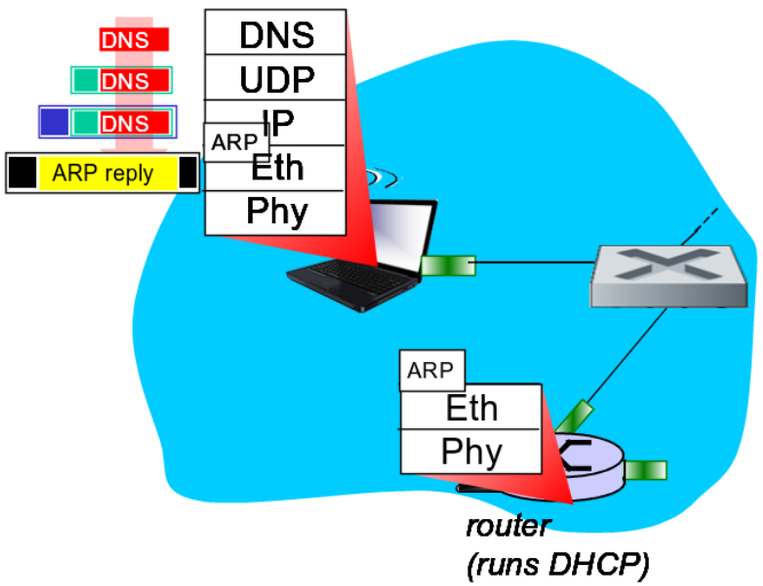
- before sending *HTTP* request, need IP address of www.google.com: *DNS*
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: *ARP*

ARP (BEFORE DNS, BEFORE HTTP)



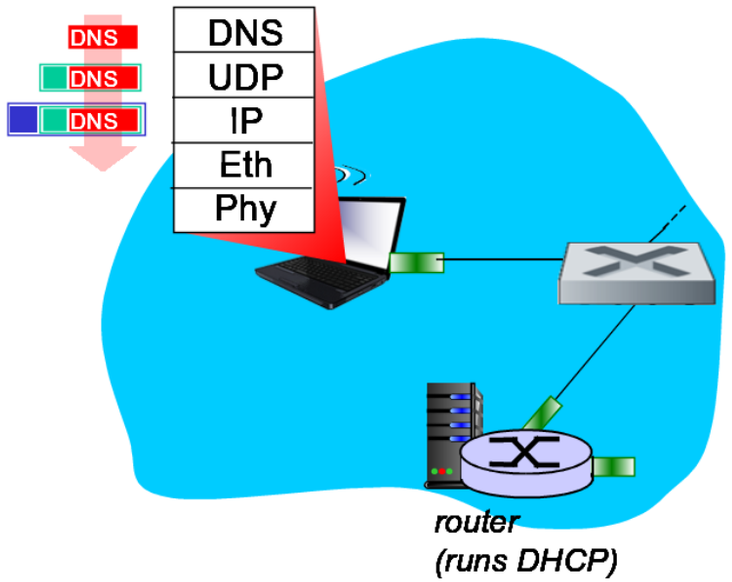
- before sending *HTTP* request, need IP address of www.google.com: *DNS*
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: *ARP*
- *ARP query* broadcast, received by router, which replies with *ARP reply* giving MAC address of router interface

ARP (BEFORE DNS, BEFORE HTTP)



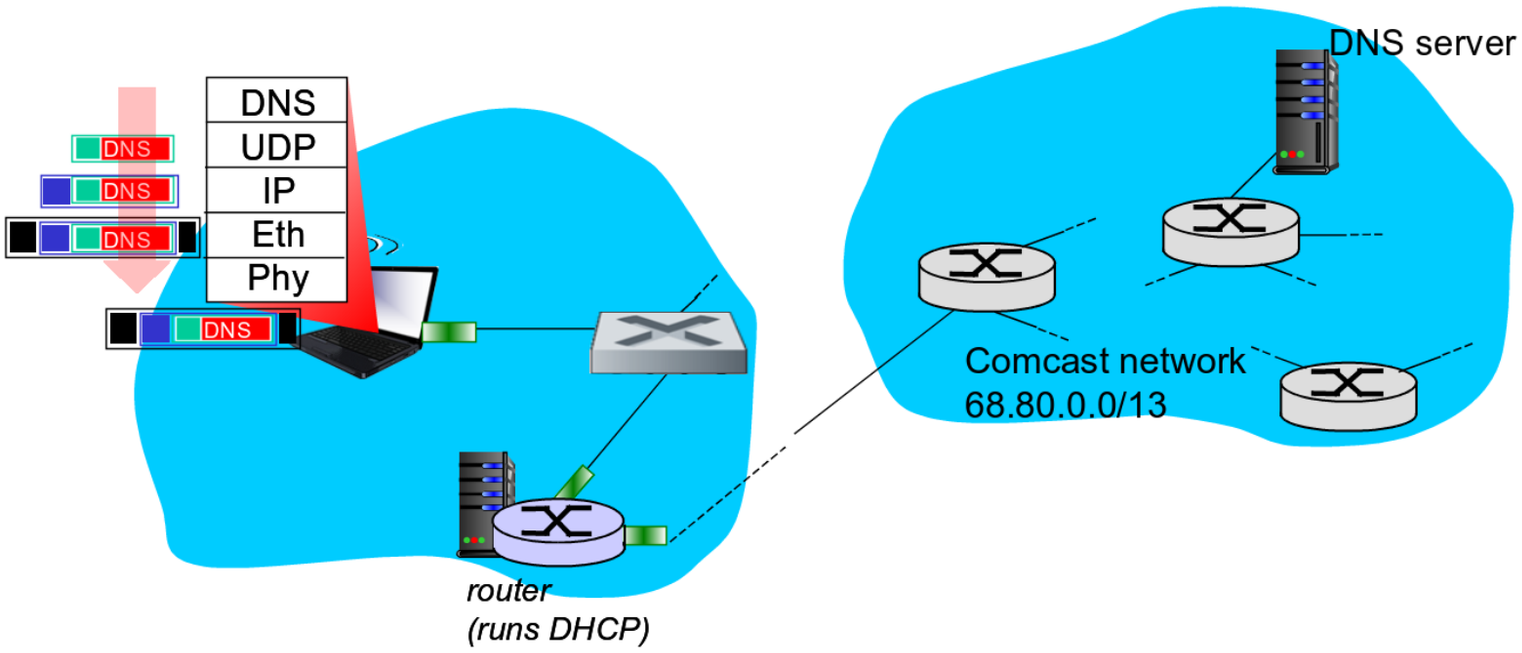
- before sending *HTTP* request, need IP address of www.google.com: *DNS*
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: *ARP*
- *ARP query* broadcast, received by router, which replies with *ARP reply* giving MAC address of router interface

ARP (BEFORE DNS, BEFORE HTTP)



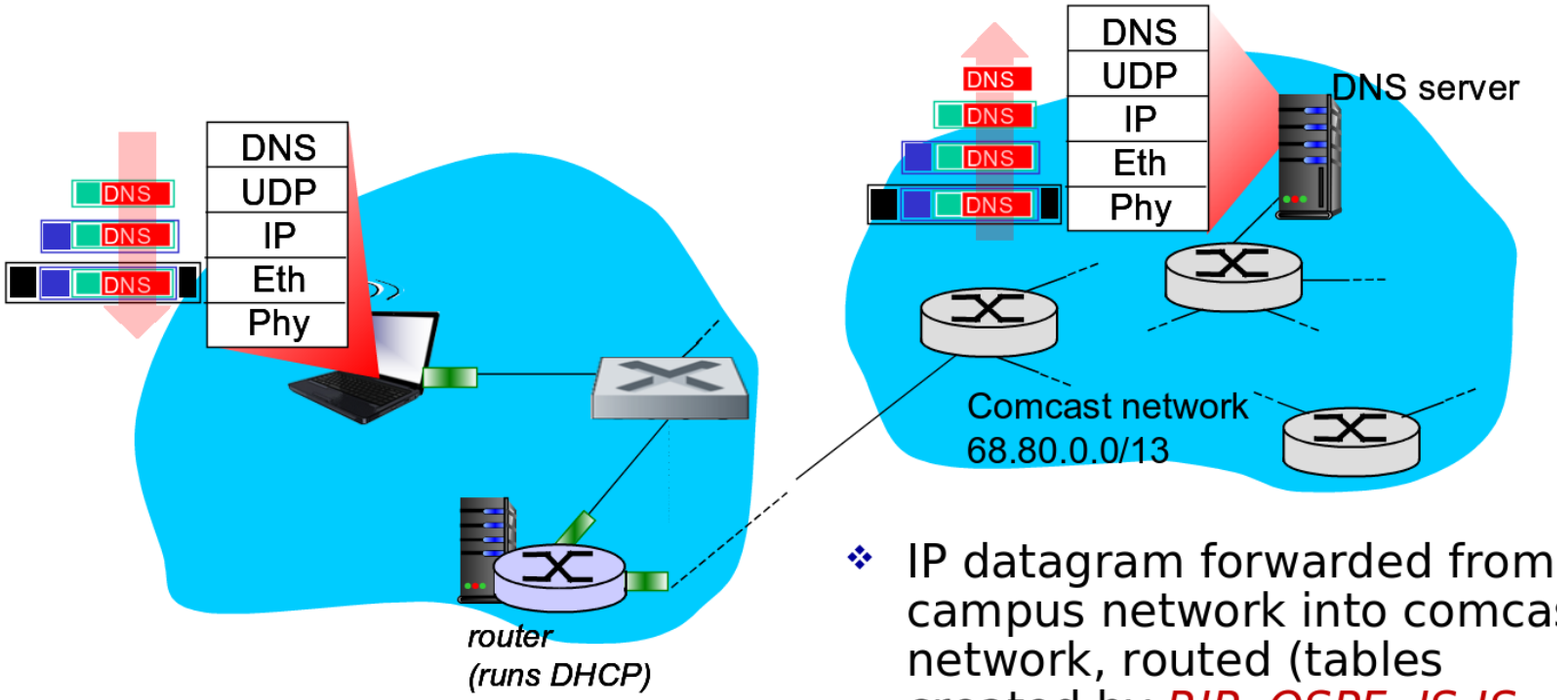
- before sending *HTTP* request, need IP address of www.google.com: *DNS*
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: *ARP*
- *ARP query* broadcast, received by router, which replies with *ARP reply* giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

USING DNS



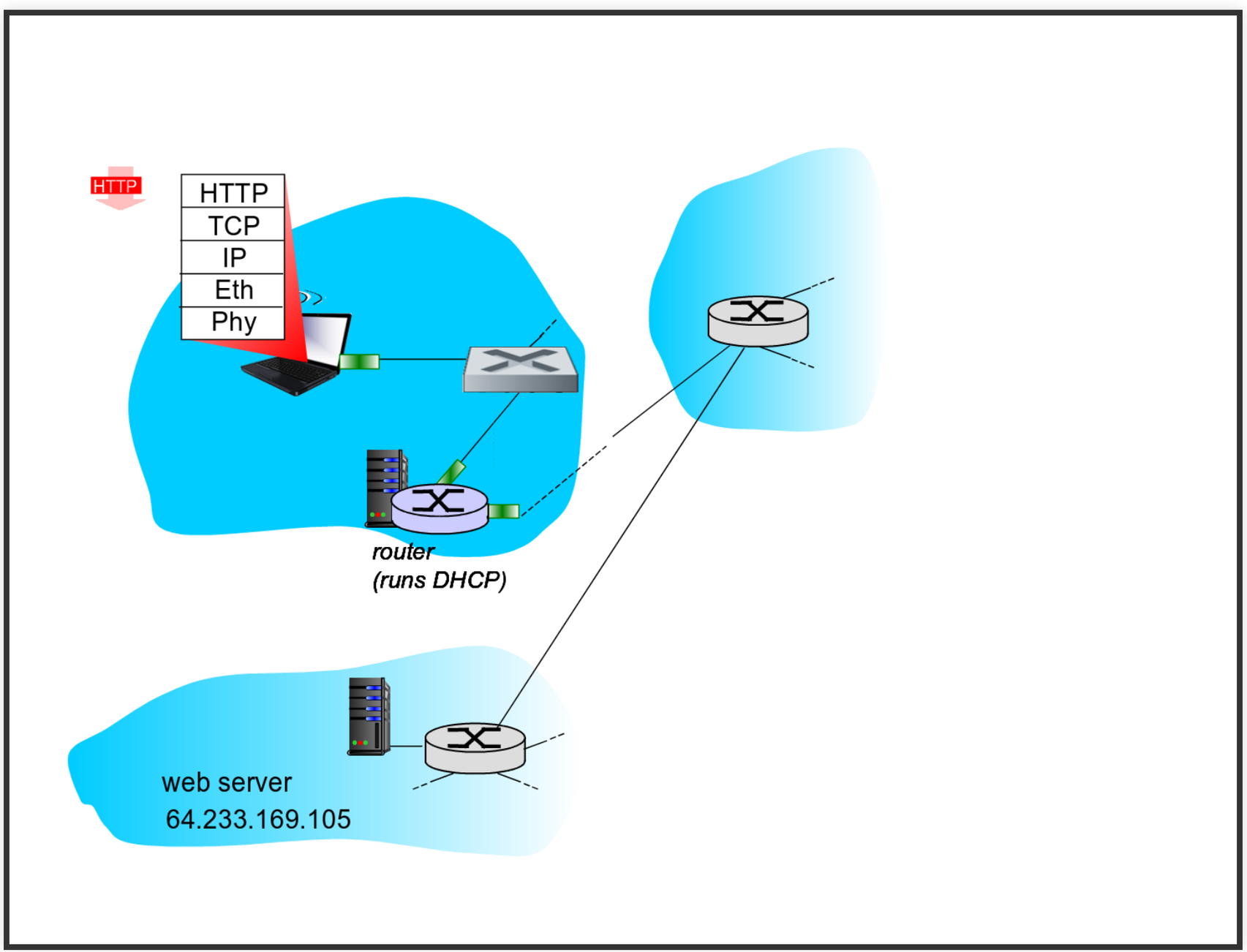
- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router

USING DNS

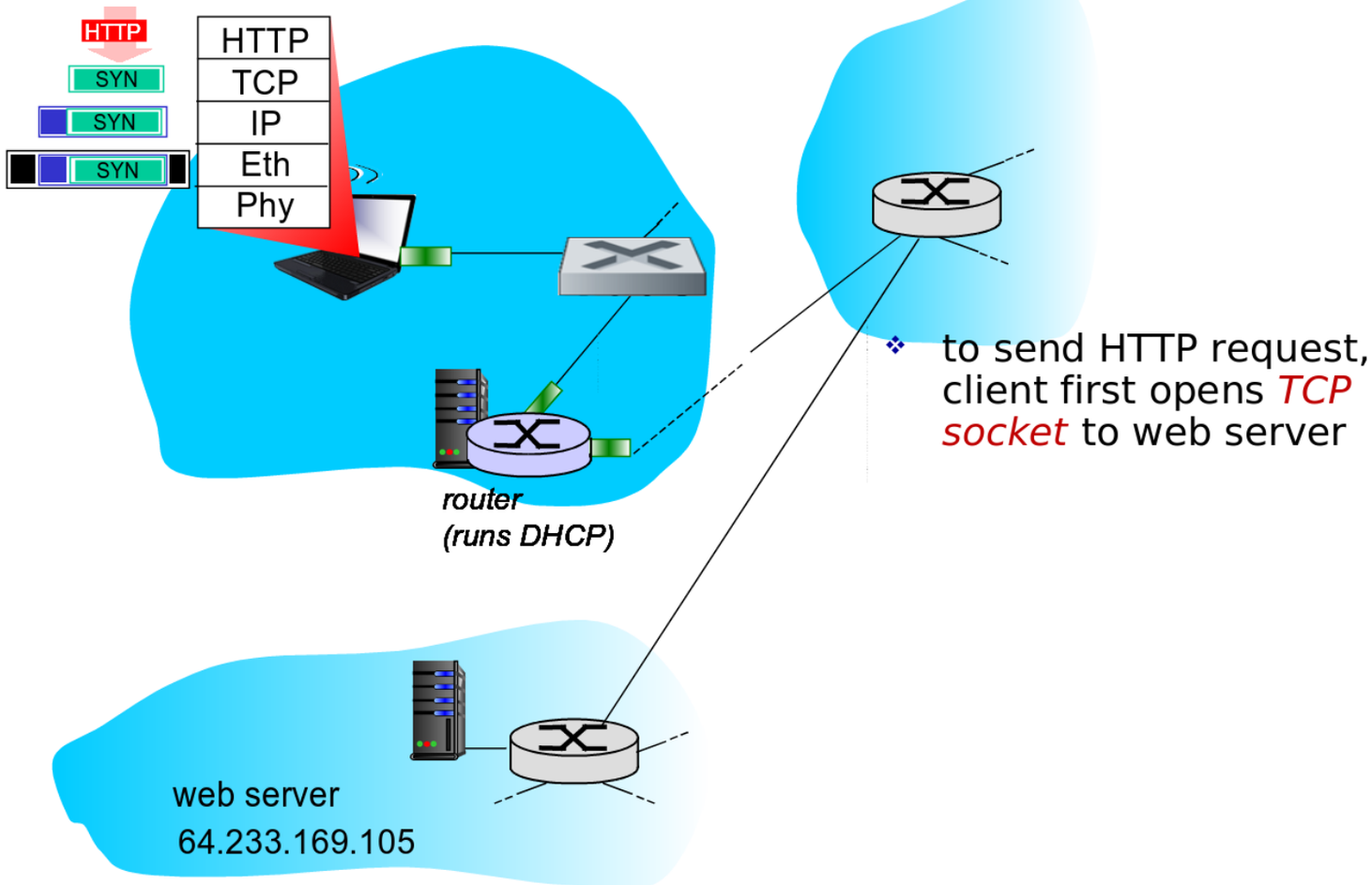


- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
- IP datagram forwarded from campus network into comcast network, routed (tables created by *RIP, OSPF, IS-IS* and/or *BGP* routing protocols) to DNS server
- demux'ed to DNS server
- DNS server replies to client with IP address of www.google.com

TCP CONN. CARRYING HTTP

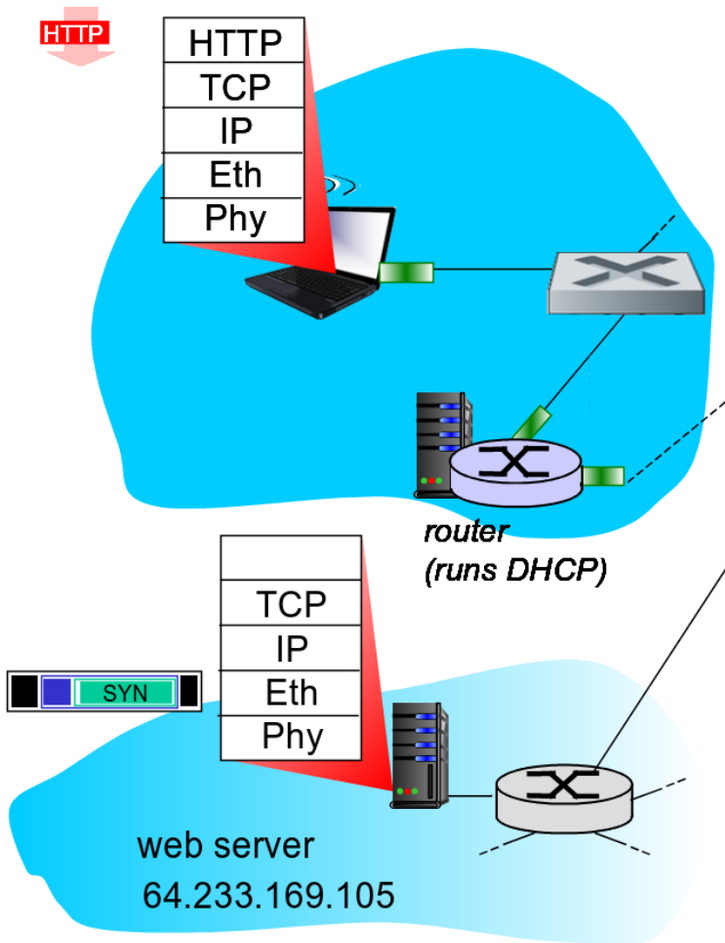


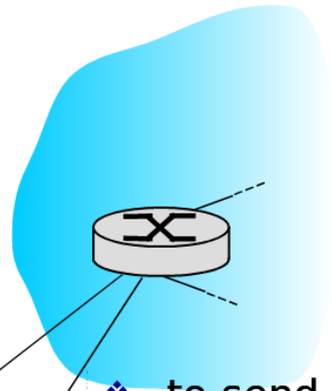
TCP CONN. CARRYING HTTP



- to send HTTP request, client first opens *TCP socket* to web server

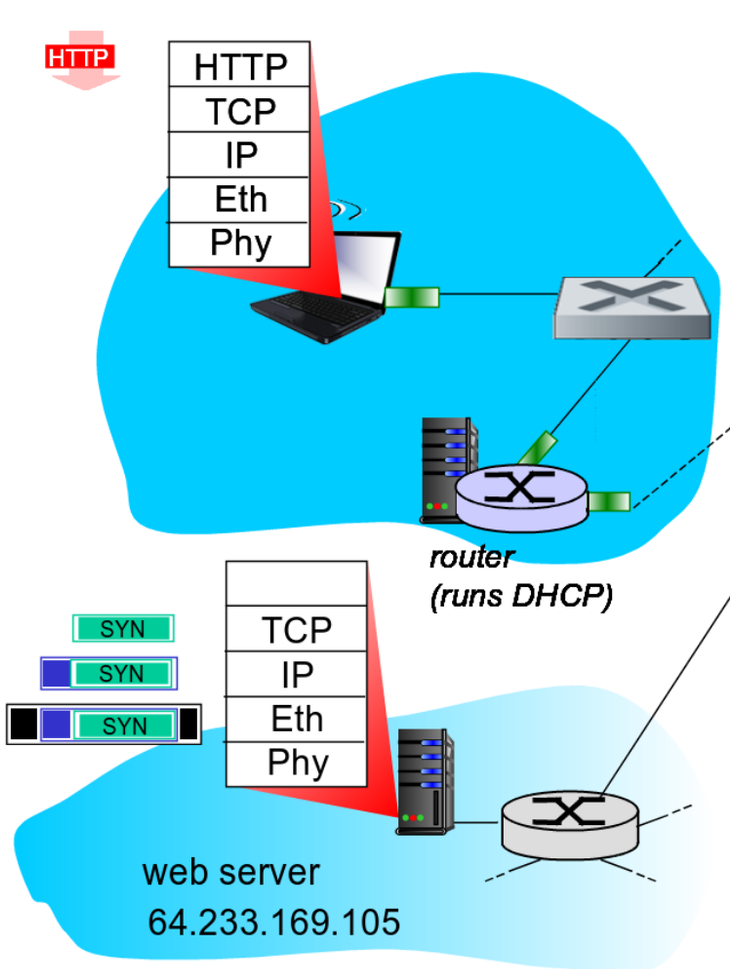
TCP CONN. CARRYING HTTP





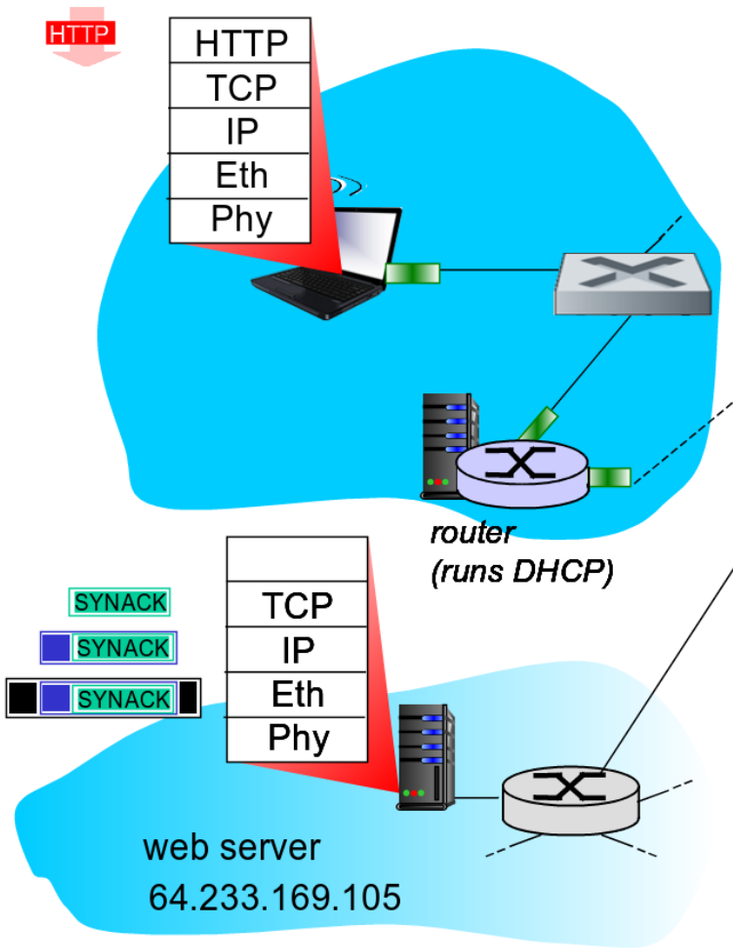
- to send HTTP request, client first opens *TCP socket* to web server
- TCP *SYN segment* (step 1 in 3-way handshake) *inter-domain routed* to web server

TCP CONN. CARRYING HTTP



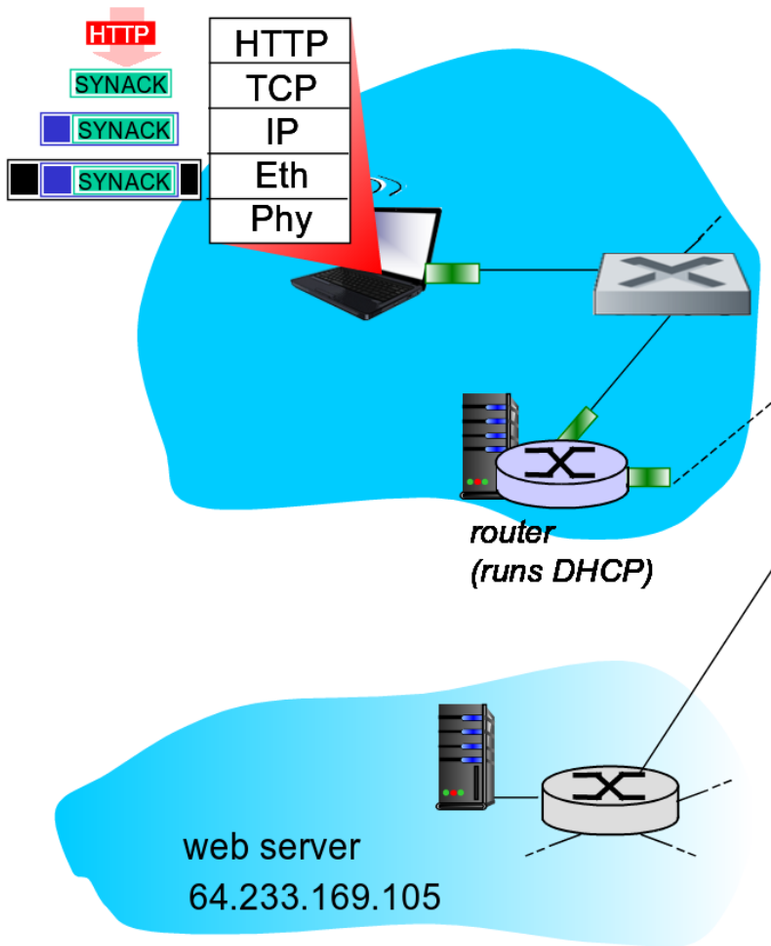
- to send HTTP request, client first opens *TCP socket* to web server
- TCP *SYN segment* (step 1 in 3-way handshake) *inter-domain routed* to web server

TCP CONN. CARRYING HTTP



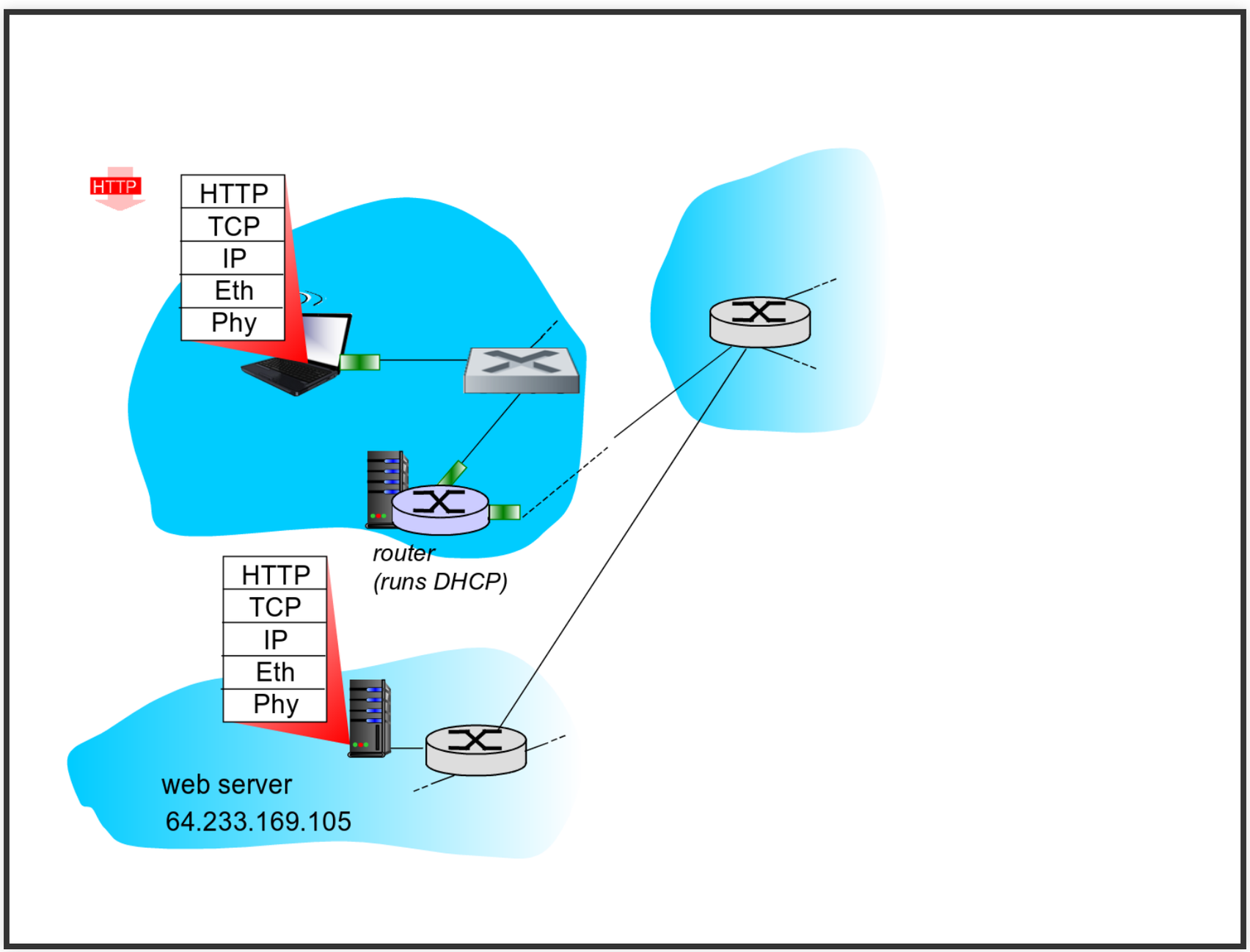
- to send HTTP request, client first opens *TCP socket* to web server
- TCP *SYN segment* (step 1 in 3-way handshake) *inter-domain routed* to web server
- web server responds with *TCP SYNACK* (step 2 in 3-way handshake)

TCP CONN. CARRYING HTTP

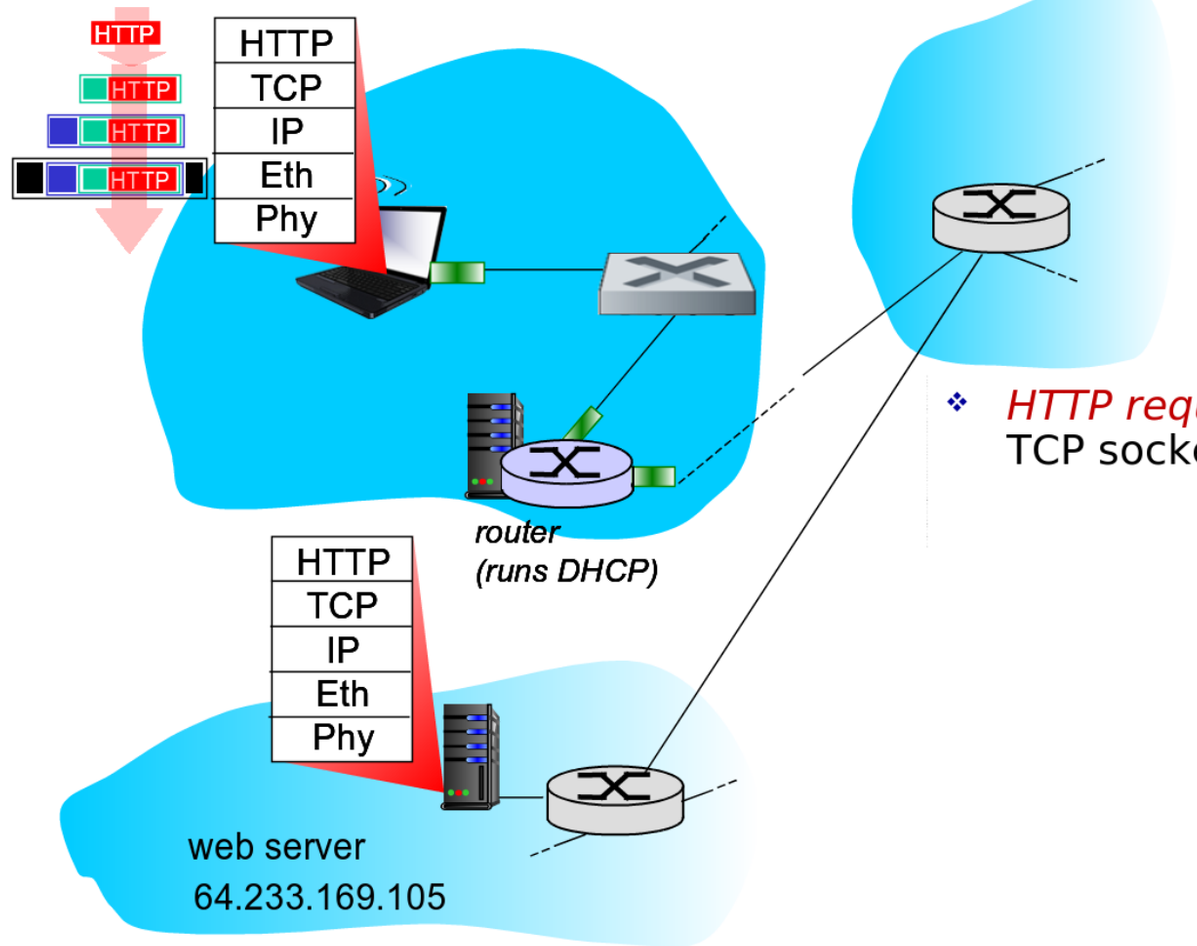


- to send HTTP request, client first opens *TCP socket* to web server
- TCP *SYN segment* (step 1 in 3-way handshake) *inter-domain routed* to web server
- web server responds with *TCP SYNACK* (step 2 in 3-way handshake)
- TCP *connection established!*

HTTP REQUEST/REPLY

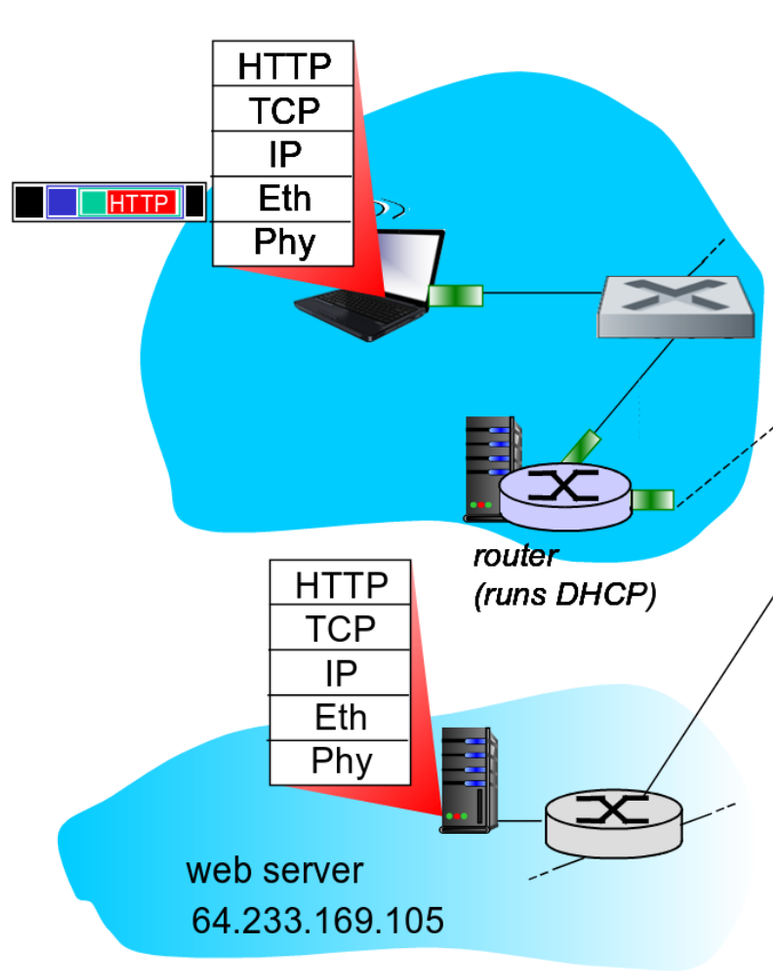


HTTP REQUEST/REPLY



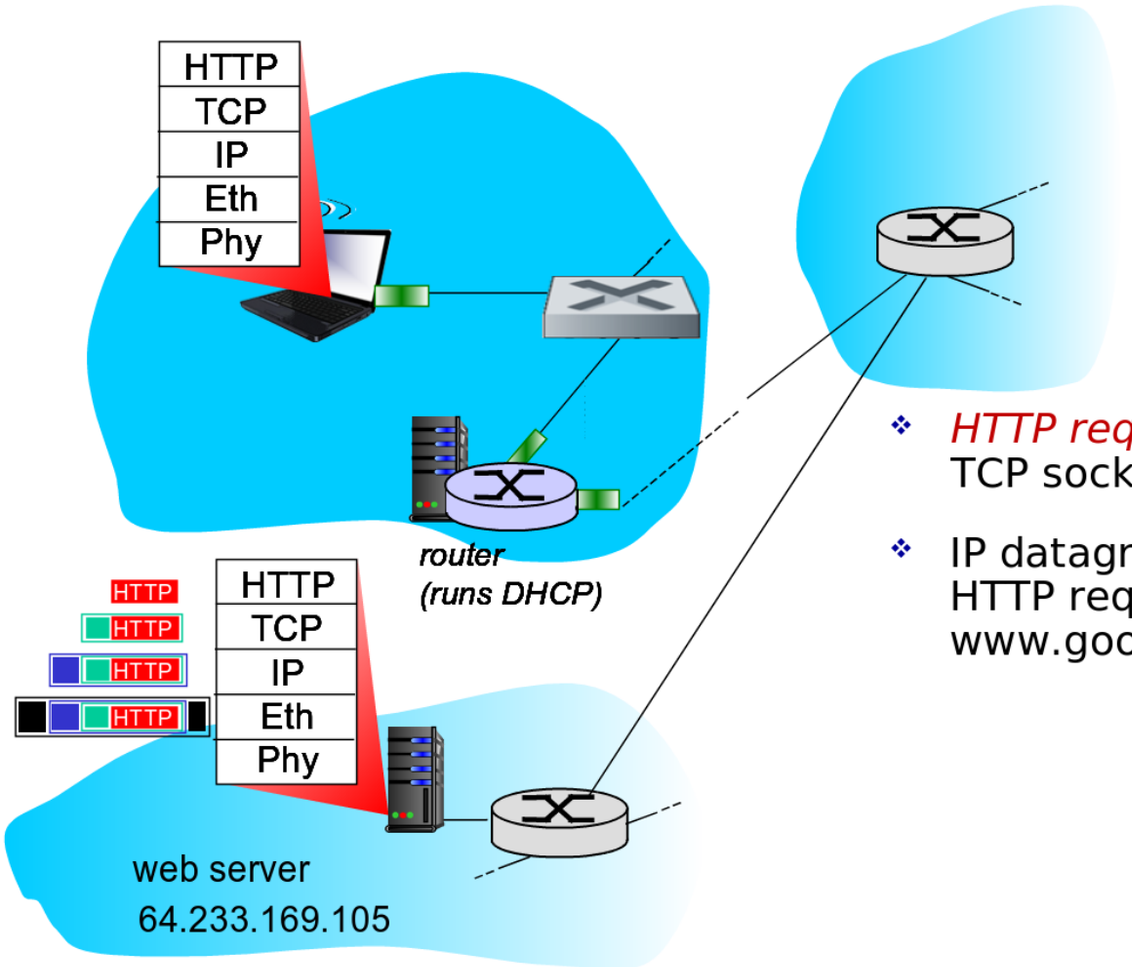
- *HTTP request* sent into TCP socket

HTTP REQUEST/REPLY



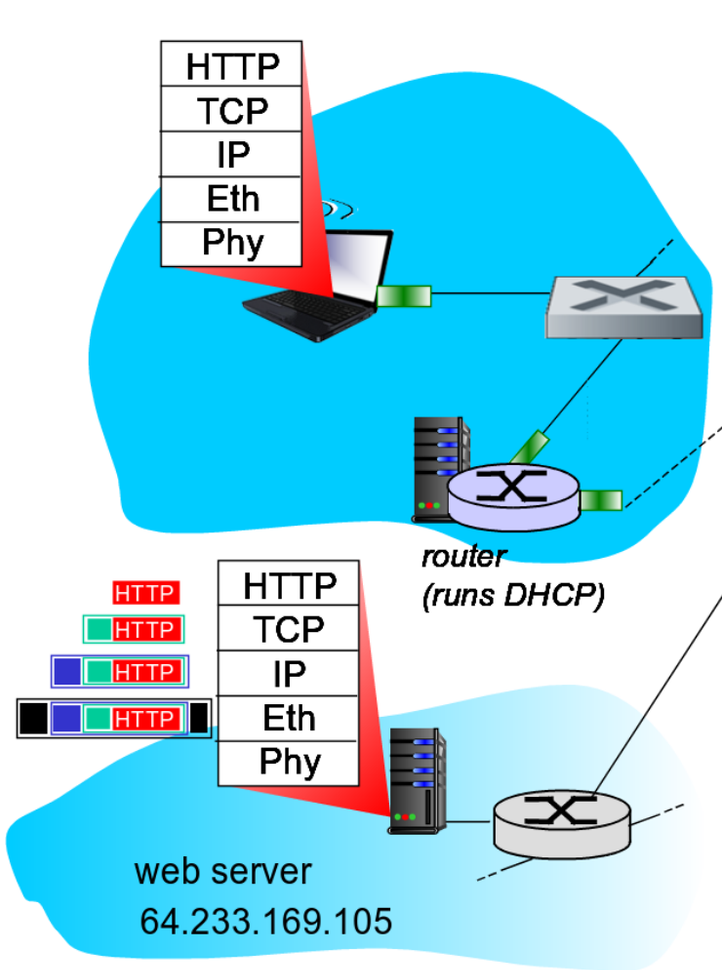
- *HTTP request* sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com

HTTP REQUEST/REPLY



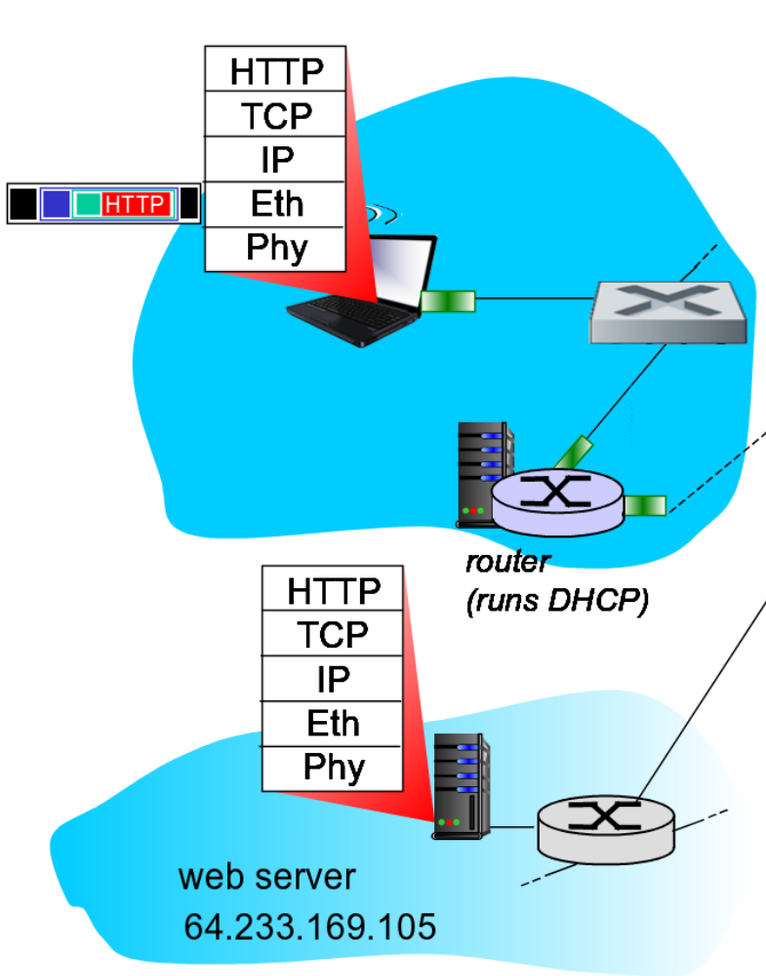
- *HTTP request* sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com

HTTP REQUEST/REPLY



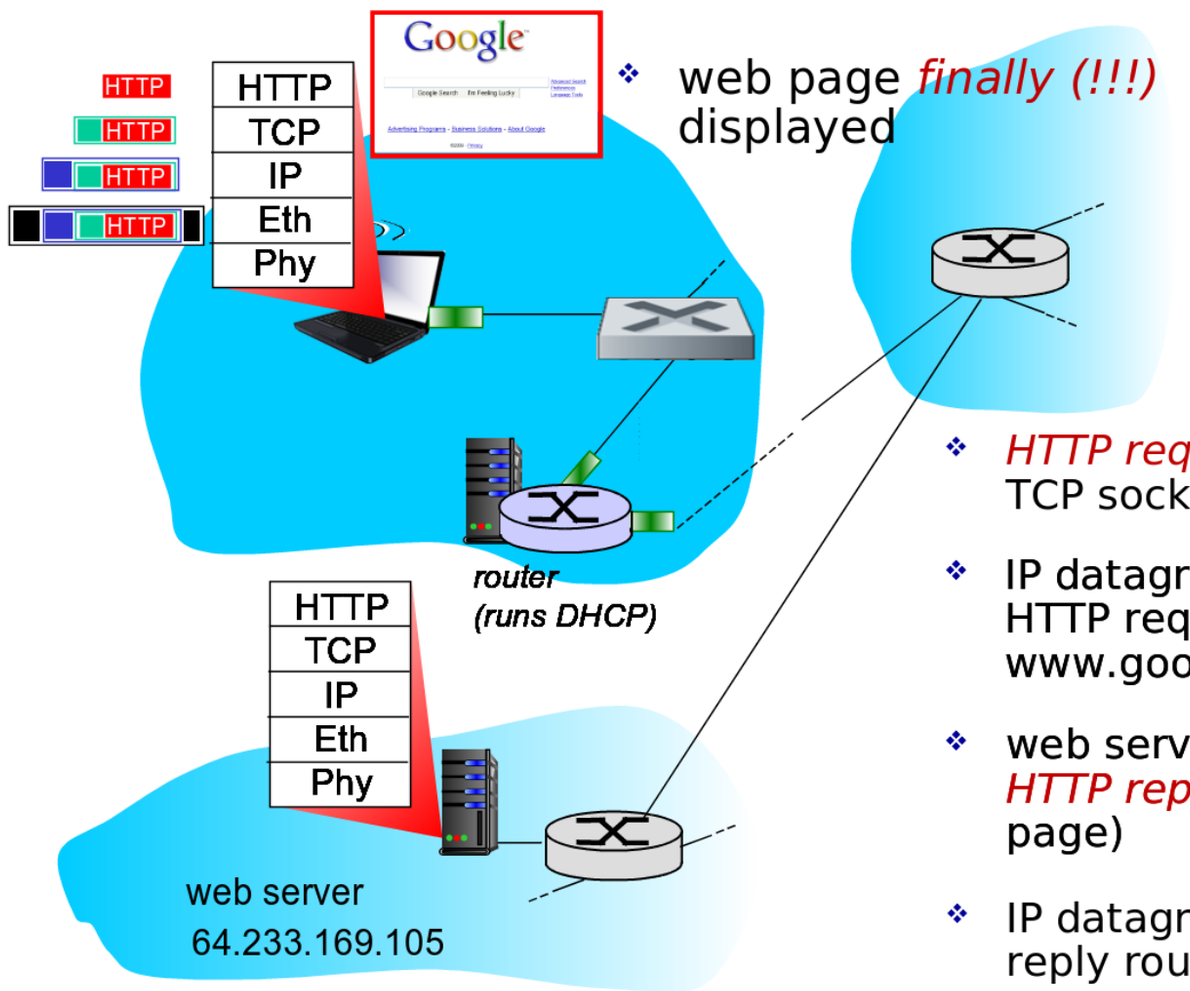
- *HTTP request* sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with *HTTP reply* (containing web page)

HTTP REQUEST/REPLY



- *HTTP request* sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with *HTTP reply* (containing web page)
- IP datagram containing HTTP reply routed back to client

HTTP REQUEST/REPLY



- *HTTP request* sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with *HTTP reply* (containing web page)
- IP datagram containing HTTP reply routed back to client

SUMMARY

- Principles behind data link layer services:
  - Error detection, correction
  - Link layer addressing
- Instantiation and implementation of various link layer technologies
  - Ethernet
  - Switched LANS, VLANs
  - Virtualized networks as a link layer: MPLS
- Synthesis: a day in the life of a web request

LET'S TAKE A BREATH

- Journey down protocol stack **complete** (except PHY)
- Solid understanding of networking principles, practice
- … could stop here … but lots of interesting topics!
  - Wireless
  - Multimedia
  - Security
  - Network management