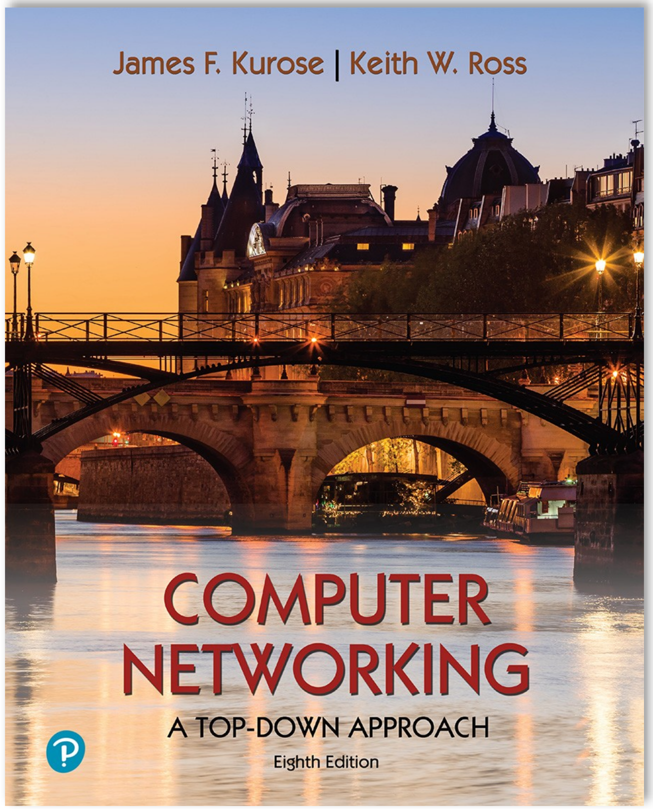
Chapter 6
The Link Layer and LANs



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

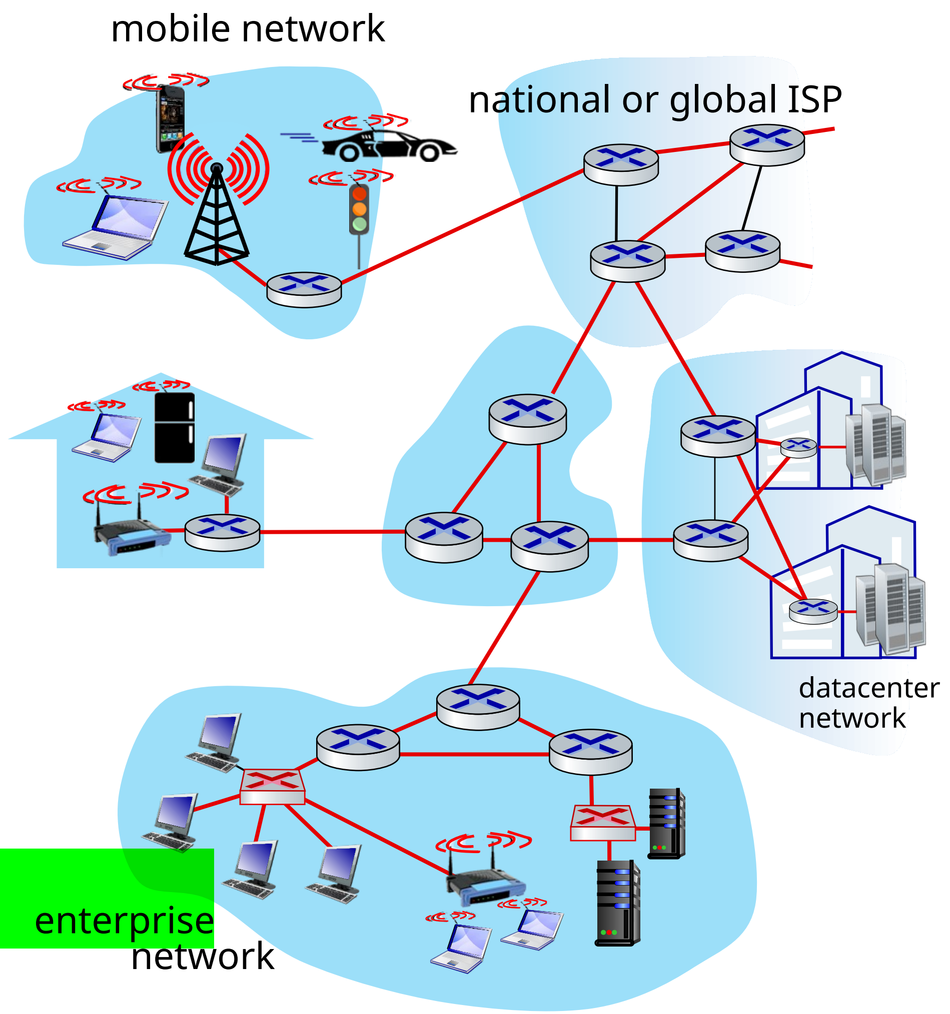
Link layer and LANs: our goals

- understand principles behind link layer services:
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
  - local area networks: Ethernet, VLANs

Link layer: introduction

terminology:

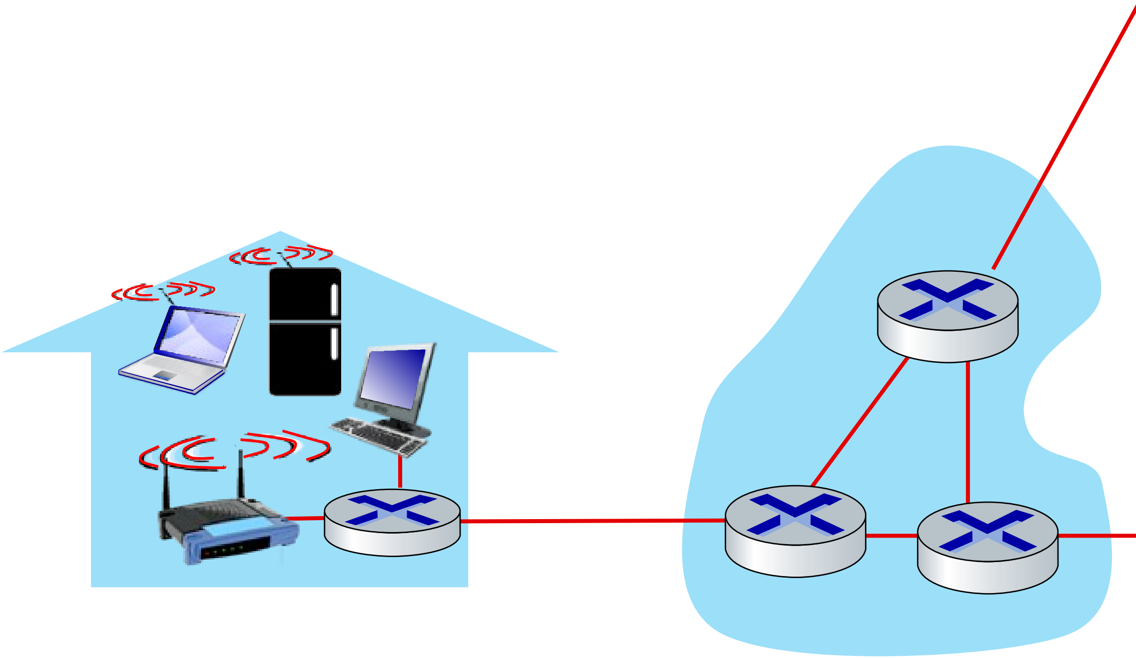
- hosts, routers: nodes
- communication channels that connect adjacent nodes along communication path: links
  - wired , wireless
  - LANs
- layer-2 packet: *frame*, encapsulates datagram



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

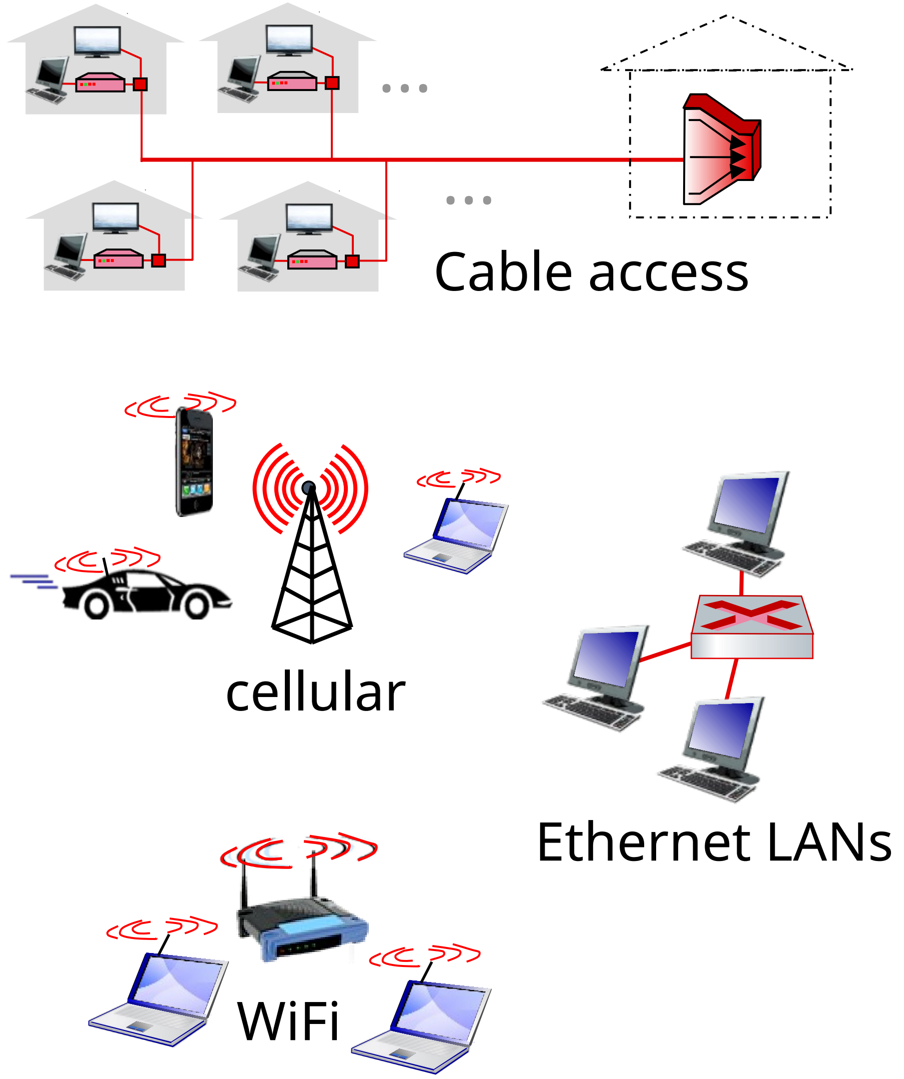
Link layer: context

- datagram transferred by different link protocols over different links:
  - e.g., WiFi on first link, Ethernet on next link
- each link protocol provides different services
  - e.g., may or may not provide reliable data transfer over link



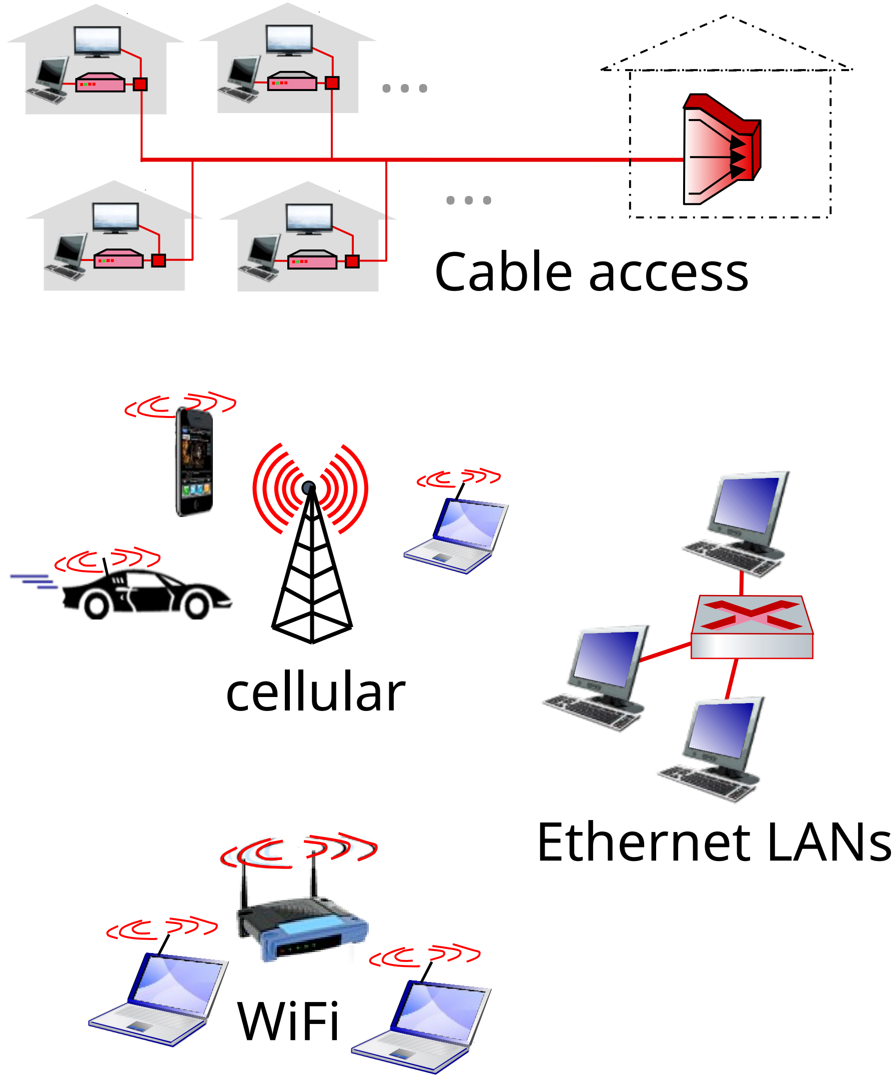
Link layer: services

- framing, link access:
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses in frame headers identify source, destination (different from IP address!)
- reliable delivery between adjacent nodes
  - we already know how to do this!
  - seldom used on low bit-error links
  - wireless links: high error rates



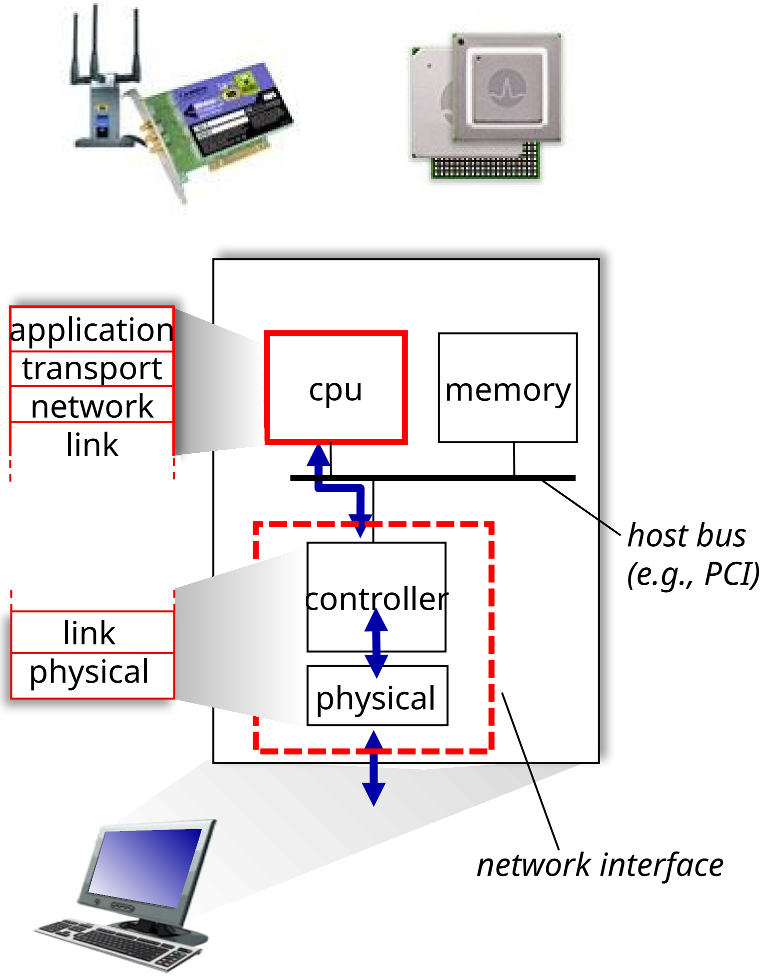
Link layer: services (more)

- **flow control:**
  - pacing between adjacent sending and receiving nodes
- **error detection:**
  - errors caused by signal attenuation, noise.
  - receiver detects errors, signals retransmission, or drops frame
- **error correction:**
  - receiver identifies *and corrects* bit error(s) without retransmission
- **half-duplex and full-duplex:**
  - with half duplex, nodes at both ends of link can transmit, but not at same time

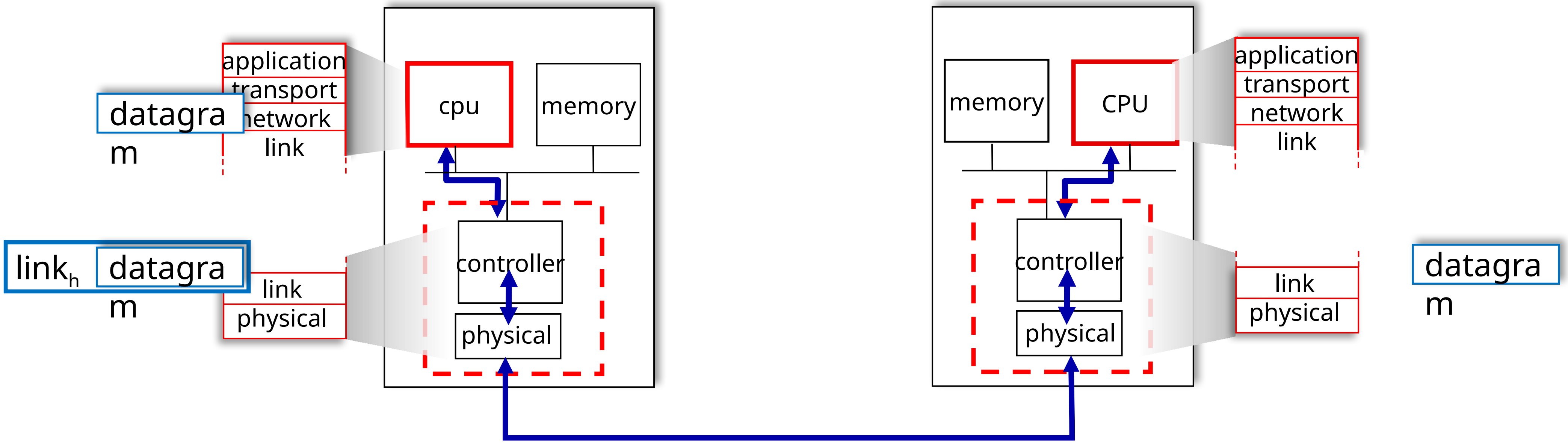


Host link-layer implementation

- in each-and-every host
- link layer implemented on-chip or in network interface card (NIC)
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware



Interfaces communicating



sending side:
- encapsulates datagram in frame
- adds error checking bits, reliable data transfer, flow control, etc.

receiving side:
- looks for errors, reliable data transfer, flow control, etc.
- extracts datagram, passes to upper layer at receiving side

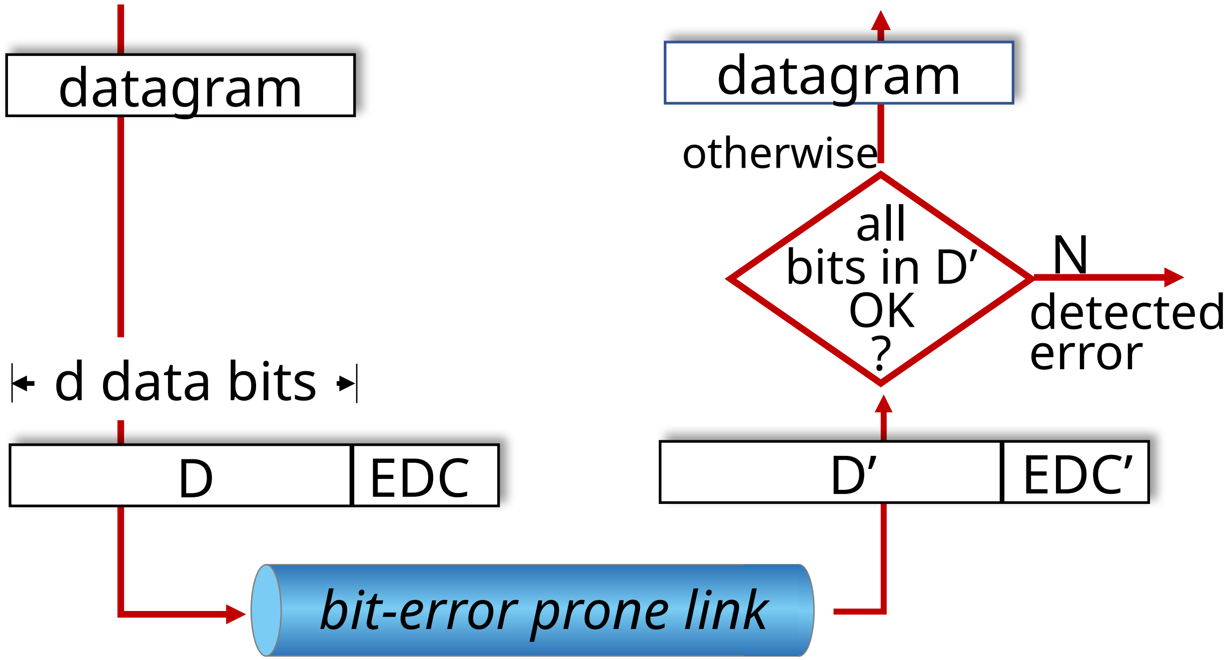
Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- LANs
  - addressing, ARP
  - Ethernet
  - switches

- a day in the life of a web request

Error detection

EDC: error detection and correction bits (e.g., 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

Parity checking

single bit parity:

- detect single bit errors

0111000110101011	1

← *d* data bits → | parity bit

Even/odd parity: set parity bit so there is an even/odd number of 1's

At receiver:

- compute parity of *d* received bits
- compare with received parity bit – if different than error detected

Can detect *and* correct errors (without retransmission!)

- **two-dimensional parity:** detect *and correct* single bit errors

row parity →

column parity ↓

$d_{1,1}$	...	$d_{1,j}$	$d_{1,j+1}$
$d_{2,1}$	...	$d_{2,j}$	$d_{2,j+1}$
...	...	...	...
$d_{i,1}$	...	$d_{i,j}$	$d_{i,j+1}$
$d_{i+1,1}$	...	$d_{i+1,j}$	$d_{i+1,j+1}$

no errors:

1	0	1	0	1	1
1	1	1	1	0	0
0	1	1	1	0	1
1	0	1	0	1	0

detected and correctable single-bit error:

1	0	1	0	1	1
1	0	1	1	0	0
0	1	1	1	0	1
1	0	1	0	1	0

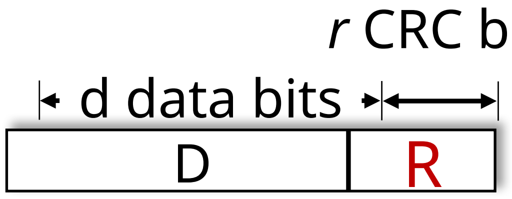
parity error →

parity error ↑

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

Cyclic Redundancy Check (CRC)

- more powerful error-detection coding
- D: data bits (given, think of these as a binary number)
- G: bit pattern (generator), of *r+1* bits (given, specified in CRC standard)



bits to send

<D,R> = $D*2^r$ XOR R

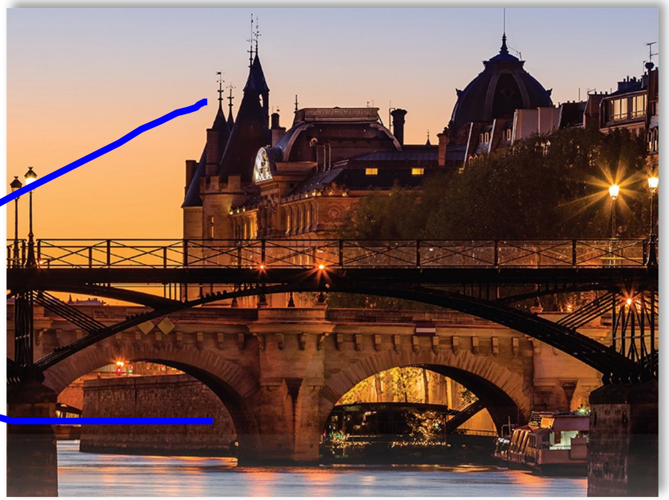
formula for these bits

sender: compute *r* CRC bits, R, such that <D,R> *exactly* divisible by G (mod 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)

Link layer, LANs: roadmap

- introduction
- error detection, correction
- **multiple access protocols**
- LANs
  - addressing, ARP
  - Ethernet
  - switches



- a day in the life of a web request

Multiple access links, protocols

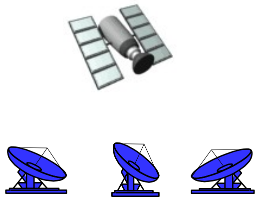
two types of “links”:

- point-to-point
  - point-to-point link between Ethernet switch, host
  - PPP for dial-up access
- broadcast (shared wire or medium)
  - old-school Ethernet
  - upstream HFC in cable-based access network
  - 802.11 wireless LAN, 4G/4G. satellite

shared wire (e.g., cabled Ethernet)

shared radio: 4G/5G

shared radio: WiFi



shared radio: satellite

humans at a cocktail party (shared air, acoustical)

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

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: multiple access channel (MAC) of rate *R* bps

desiderata:

1. when one node wants to transmit, it 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

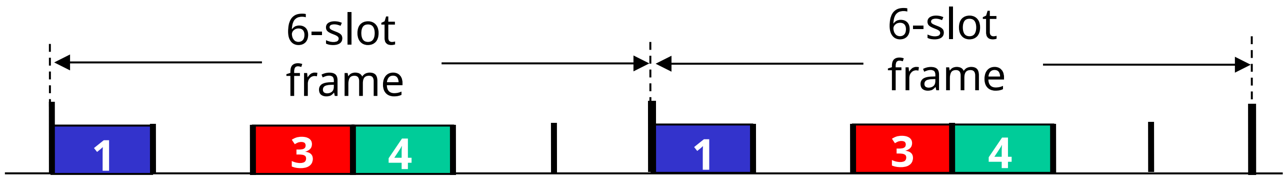
three broad classes:

- 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

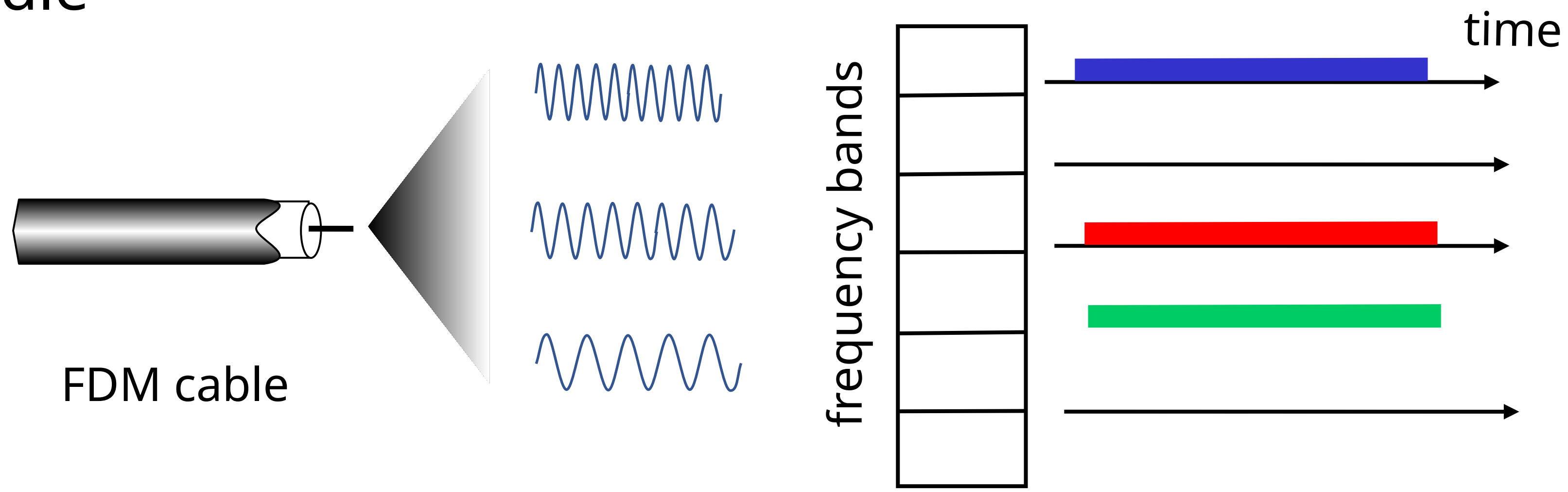
- access to channel in “rounds”
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



Channel partitioning MAC protocols: FDMA

FDMA: frequency division multiple access

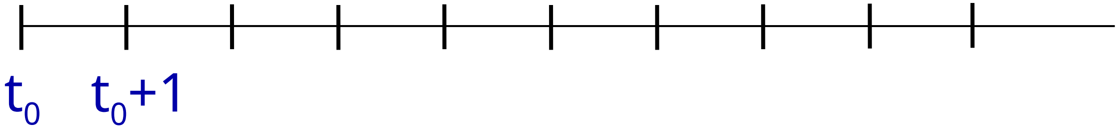
- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



Random access protocols

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

Slotted ALOHA



assumptions:

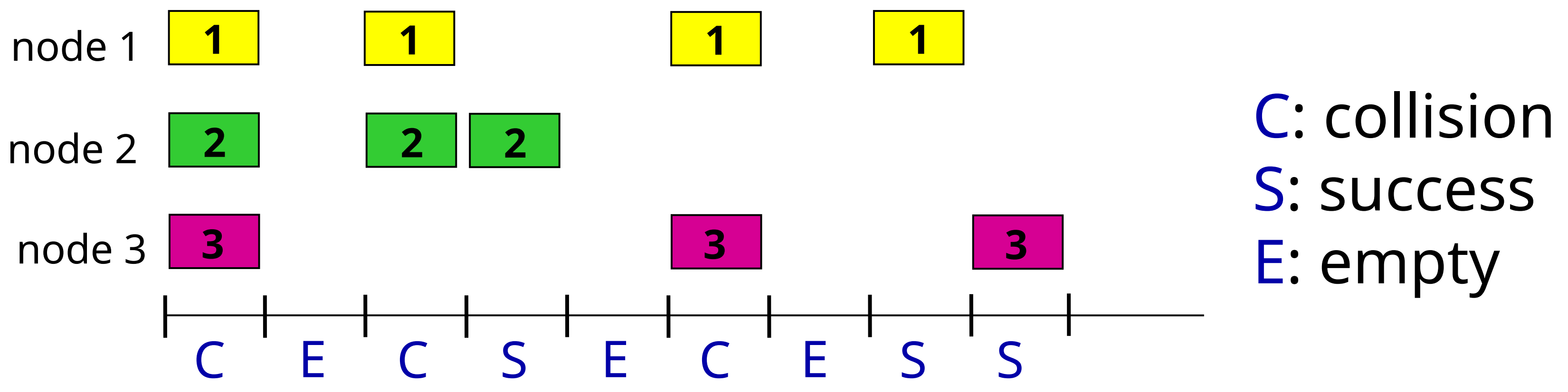
- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

operation:

- when node obtains fresh frame, transmits in next slot
  - *if no collision:* node can send new frame in next slot
  - *if collision:* node retransmits frame in each subsequent slot with probability *p* until success

randomization – *why*?

Slotted ALOHA



Pros:

- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

Cons:

- collisions, wasting slots
- idle slots
- nodes may be able to detect collision in less than time to transmit packet
- clock synchronization

CSMA (carrier sense multiple access)

simple CSMA: listen before transmit:

- if channel sensed idle: transmit entire frame
- if channel sensed busy: defer transmission

- human analogy: don't interrupt others!

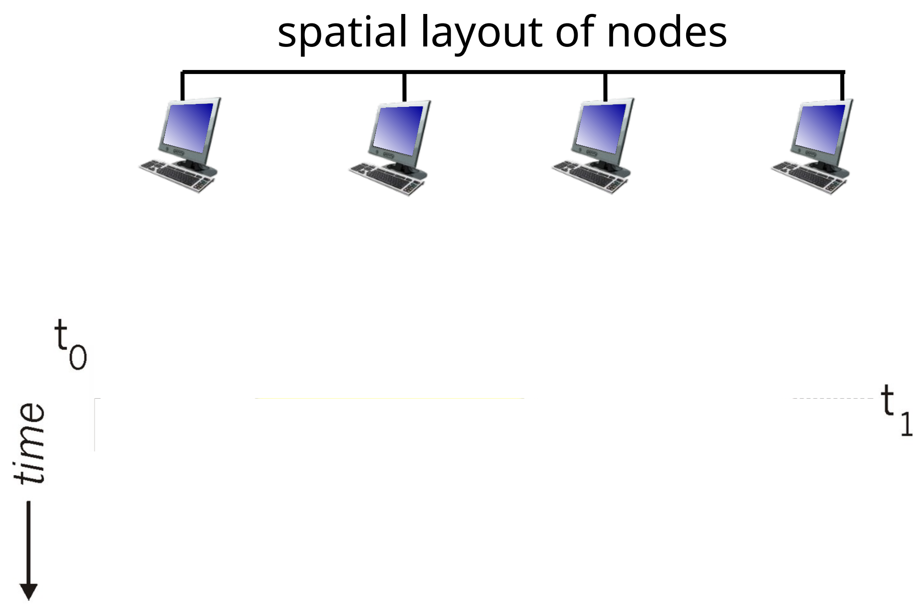
CSMA/CD: CSMA with *collision detection*

- collisions *detected* within short time
- colliding transmissions aborted, reducing channel wastage
- collision detection easy in wired, difficult with wireless

- human analogy: the polite conversationalist

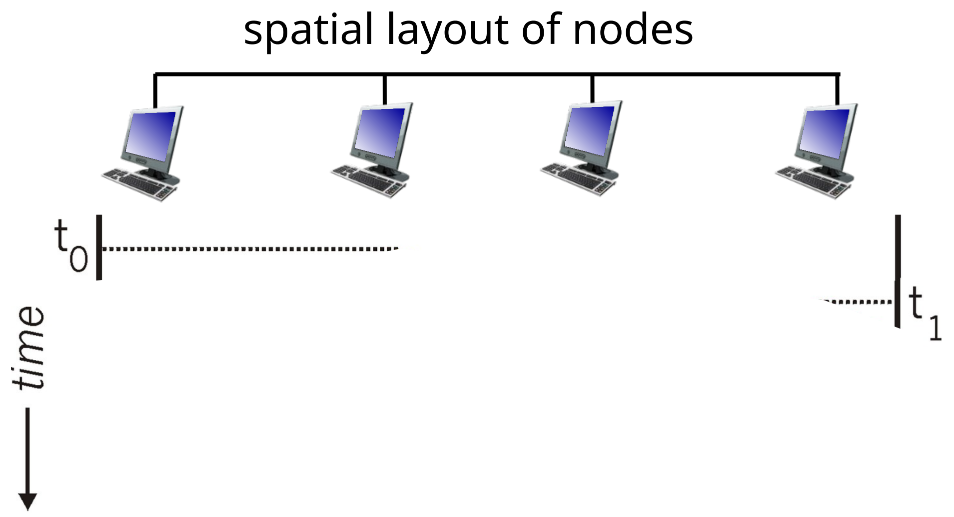
CSMA: collisions

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



CSMA/CD:

- CSMA/CD reduces the amount of time wasted in collisions
  - transmission aborted on collision detection



Ethernet CSMA/CD algorithm

1. Ethernet receives datagram from network layer, creates frame
2. If Ethernet senses channel:
   if idle: start frame transmission.
   if busy: wait until channel idle, then transmit
3. If entire frame transmitted without collision - done!
4. If another transmission detected while sending: abort, send jam signal
5. After aborting, enter *binary (exponential) backoff:*
   - after *m*th collision, chooses *K* at random from $\{0,1,2, \ldots, 2^m-1\}$. Ethernet waits $K \cdot 512$ bit times, returns to Step 2
   - more collisions: longer backoff interval

“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

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

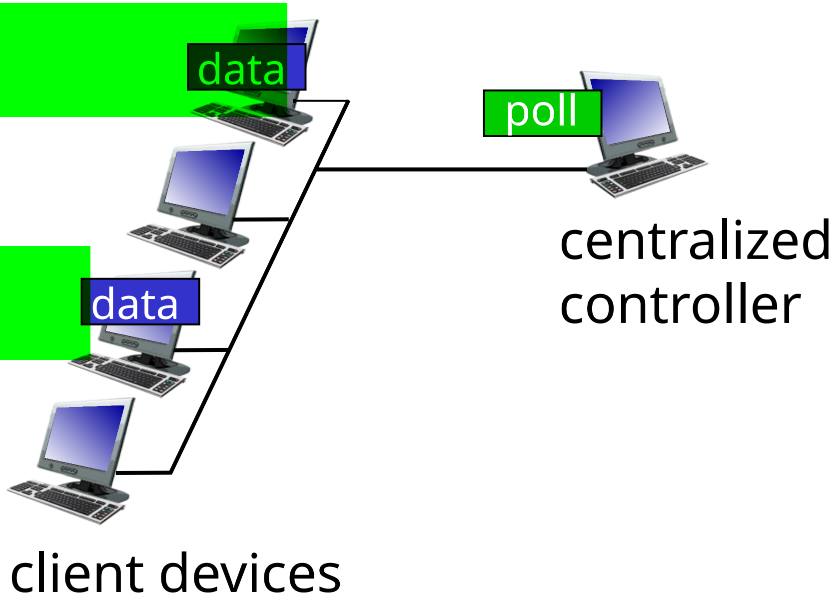
“taking turns” protocols

- look for best of both worlds!

"Taking turns" MAC protocols

polling:

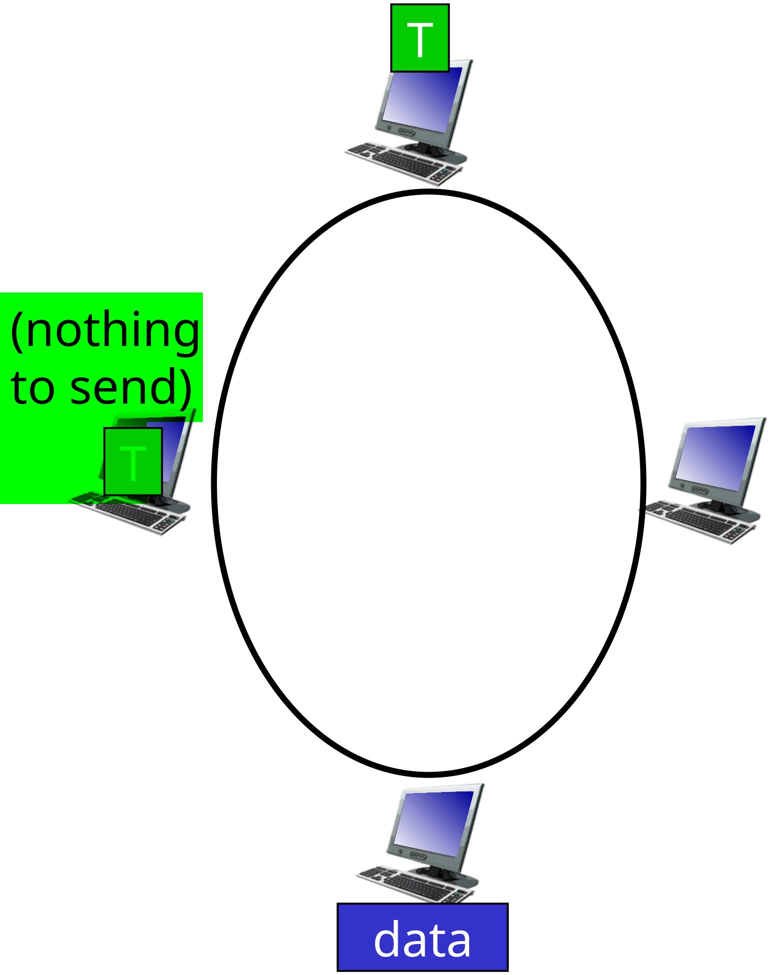
- centralized controller "invites" other nodes to transmit in turn
- typically used with "dumb" devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)
- Bluetooth uses polling



"Taking turns" MAC protocols

token passing:

- control *token* message explicitly passed from one node to next, sequentially
  - transmit while holding token
- concerns:
  - token overhead
  - latency
  - single point of failure (token)

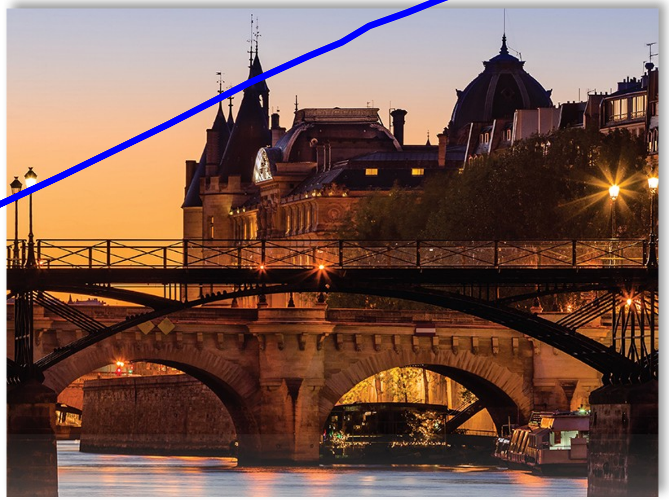


Summary of MAC protocols

- channel partitioning, by time, frequency or code
  - Time Division, Frequency Division
- random access (dynamic),
  - ALOHA, S-ALOHA, CSMA, CSMA/CD
  - carrier sensing: easy in some technologies (wire), hard in others (wireless)
  - CSMA/CD used in Ethernet
  - CSMA/CA used in 802.11
- taking turns
  - polling from central site, token passing
  - Bluetooth, FDDI, token ring

Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- LANs
  - addressing, ARP
  - Ethernet
  - switches



- a day in the life of a web request

MAC addresses

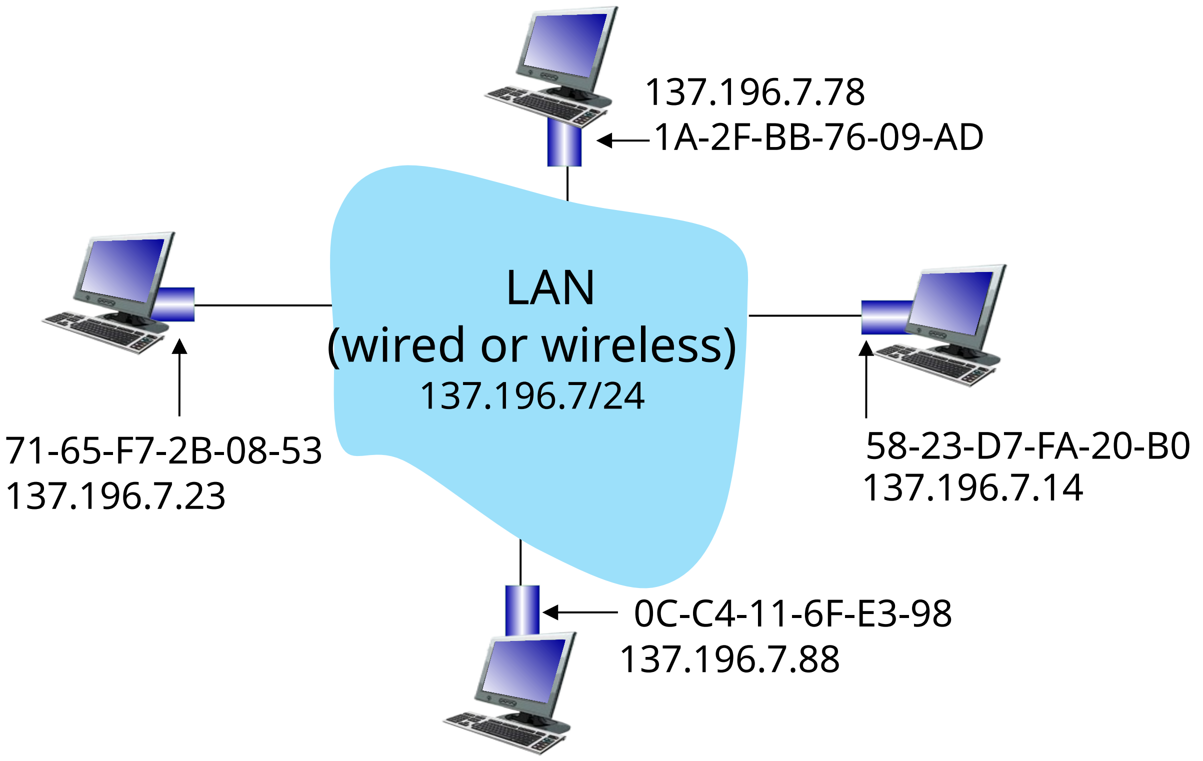
- 32-bit IP address:
  - *network-layer* address for interface
  - used for layer 3 (network layer) forwarding
  - e.g.: 128.119.40.136
- MAC (or LAN or physical or Ethernet) address:
  - function: used "locally" to get frame from one interface to another physically-connected interface (same subnet, 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

hexadecimal (base 16) notation
(each "numeral" represents 4 bits)

MAC addresses

each interface on LAN

- has unique 48-bit MAC address
- has a locally unique 32-bit IP address (as we've seen)

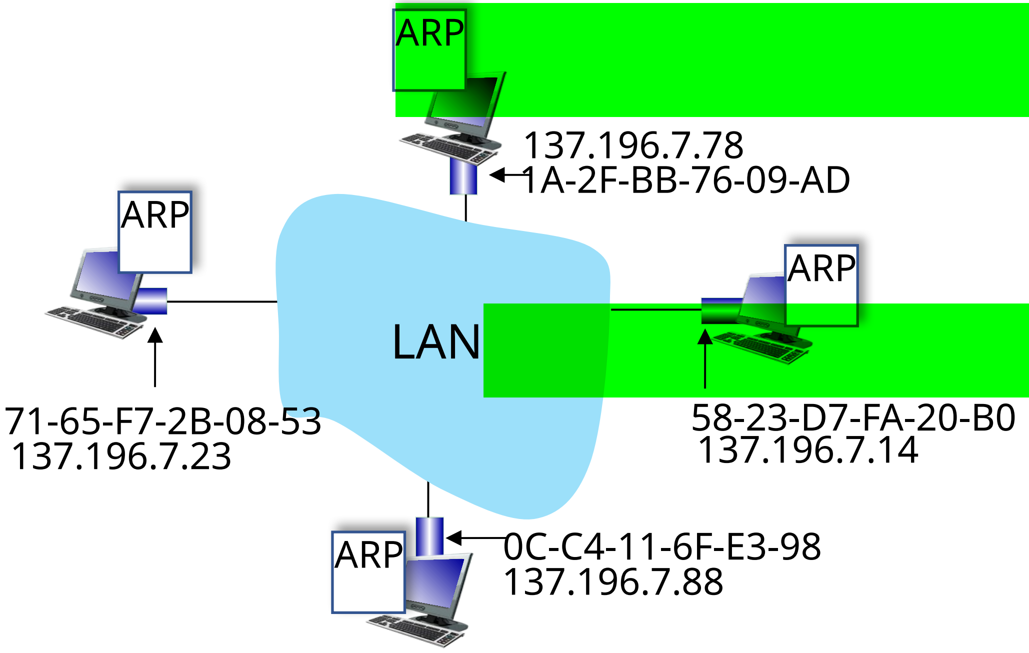


MAC addresses

- 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
- MAC flat address: portability
  - can move interface from one LAN to another
  - recall IP address *not* portable: depends on IP subnet to which node is attached

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 in action

example: A wants to send datagram to B

- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address

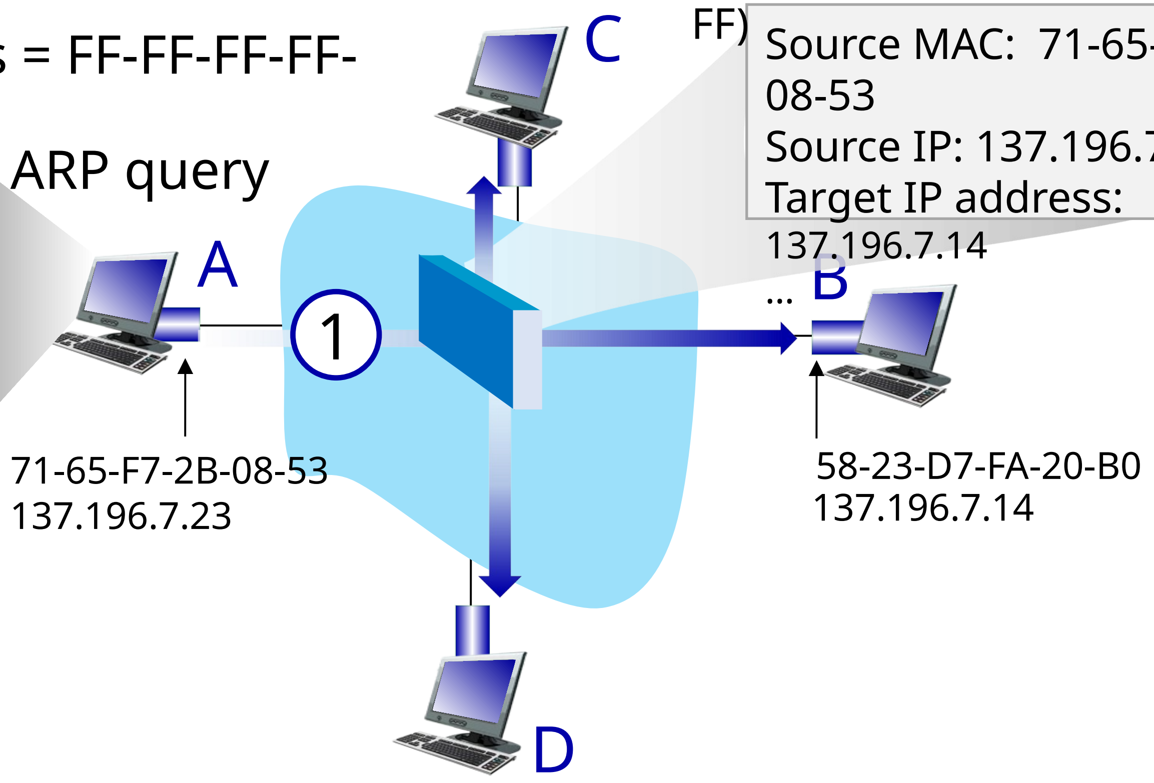
1 A broadcasts ARP query, containing B's IP addr
- destination MAC address = FF-FF-FF-FF-FF-FF
- all nodes on LAN receive ARP query

ARP table in A

IP addr	MAC addr	TTL

Ethernet frame (sent to FF-FF-FF-FF-FF-FF)

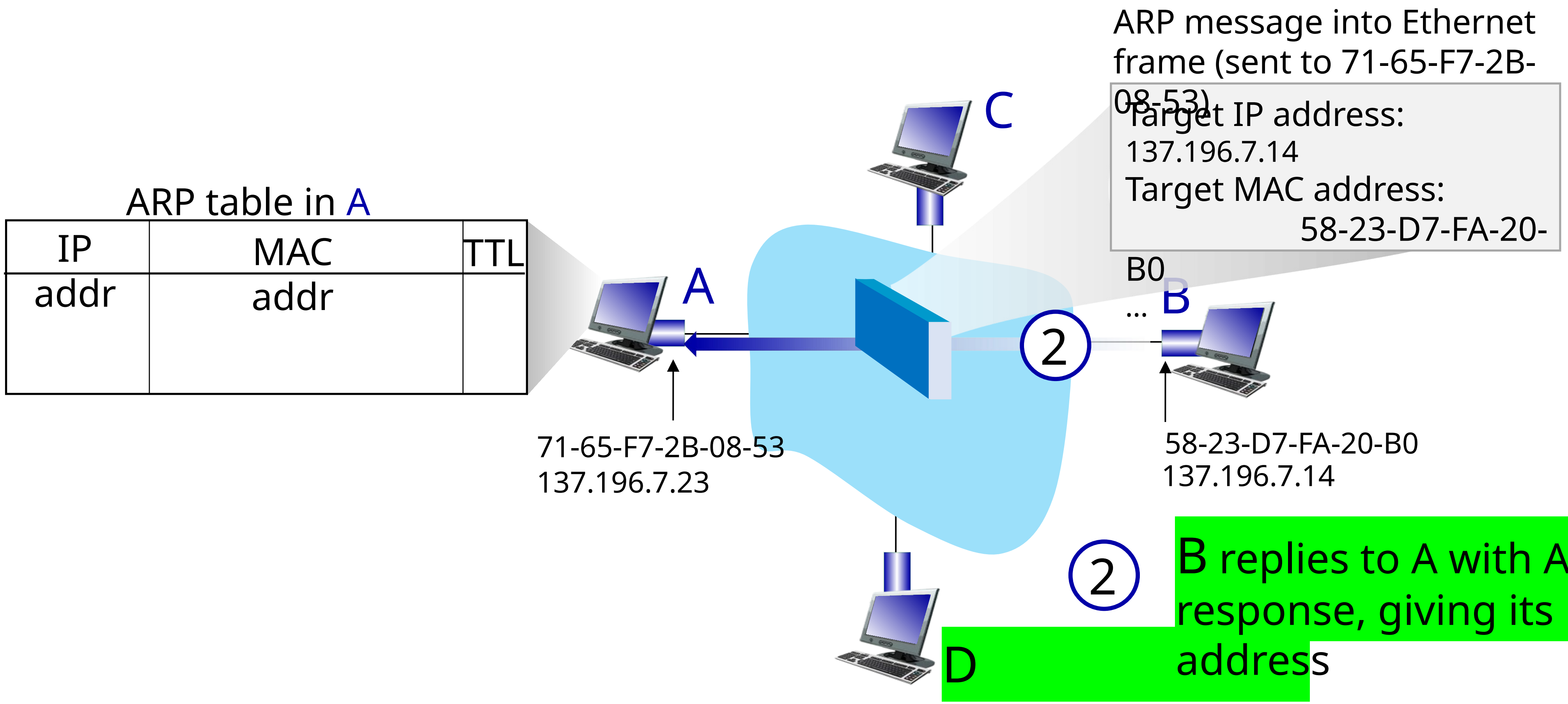
Source MAC: 71-65-F7-2B-08-53
Source IP: 137.196.7.23
Target IP address: 137.196.7.14
…



ARP protocol in action

example: A wants to send datagram to B

- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



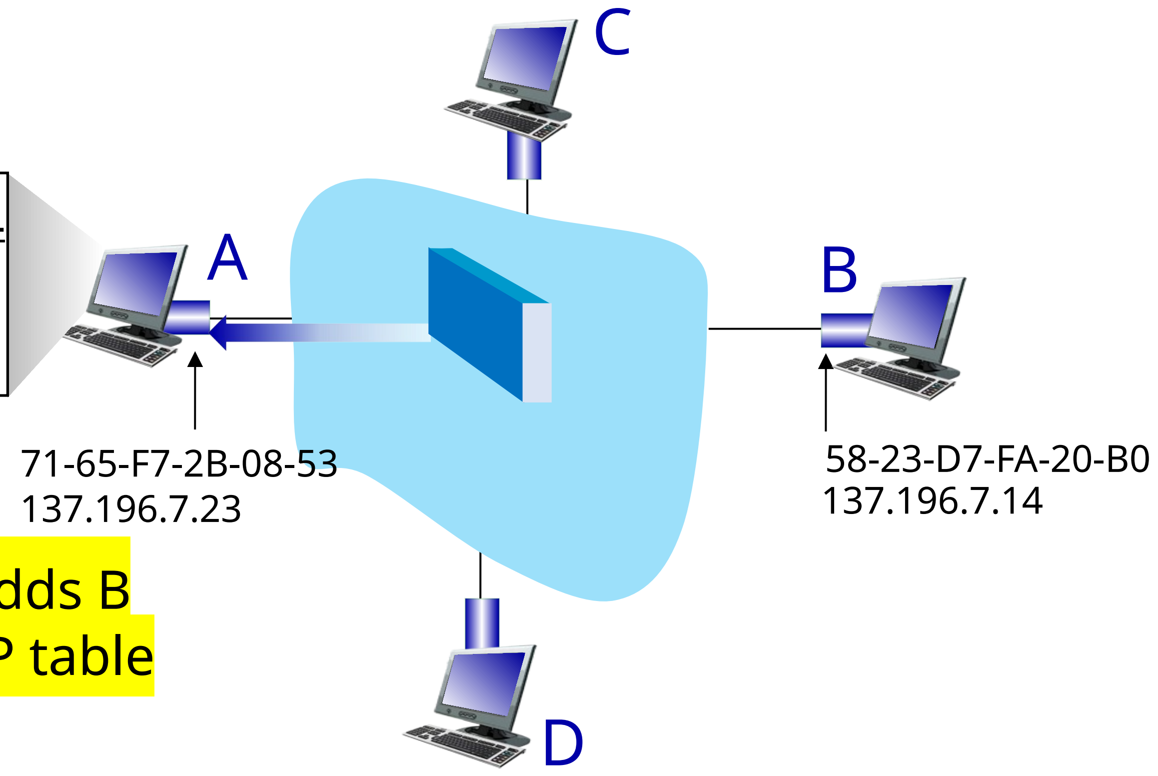
2 B replies to A with ARP response, giving its MAC address

ARP protocol in action

example: A wants to send datagram to B
- B’s MAC address not in A’s ARP table, so A uses ARP to find B’s MAC address

ARP table in A

IP addr	MAC addr	TTL
137.196.7.14	58-23-D7-FA-20-B0	500

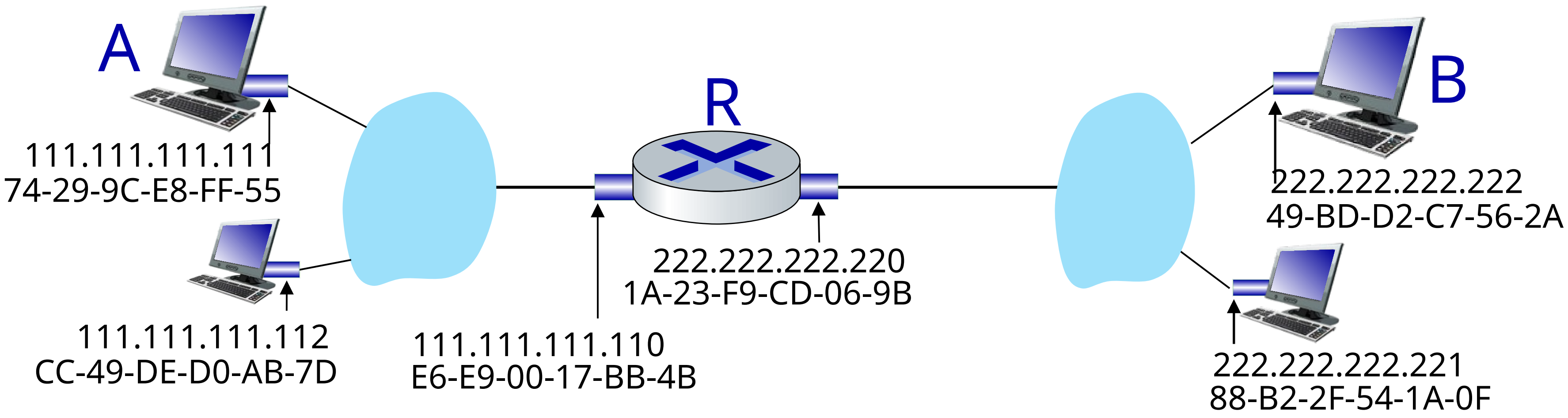


3 A receives B’s reply, adds B entry into its local ARP table

Routing to another subnet: addressing

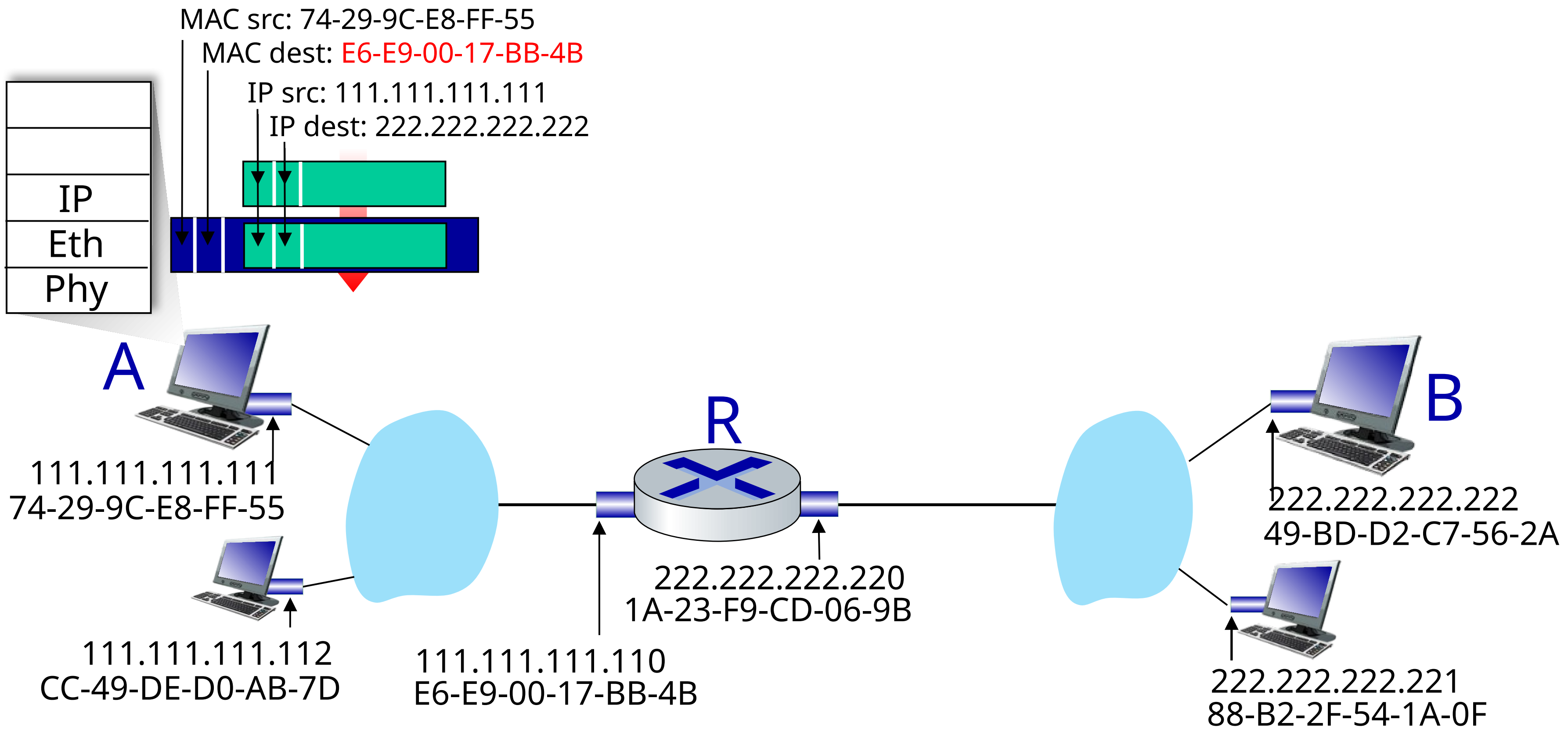
walkthrough: sending a datagram from *A* to *B* via *R*

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



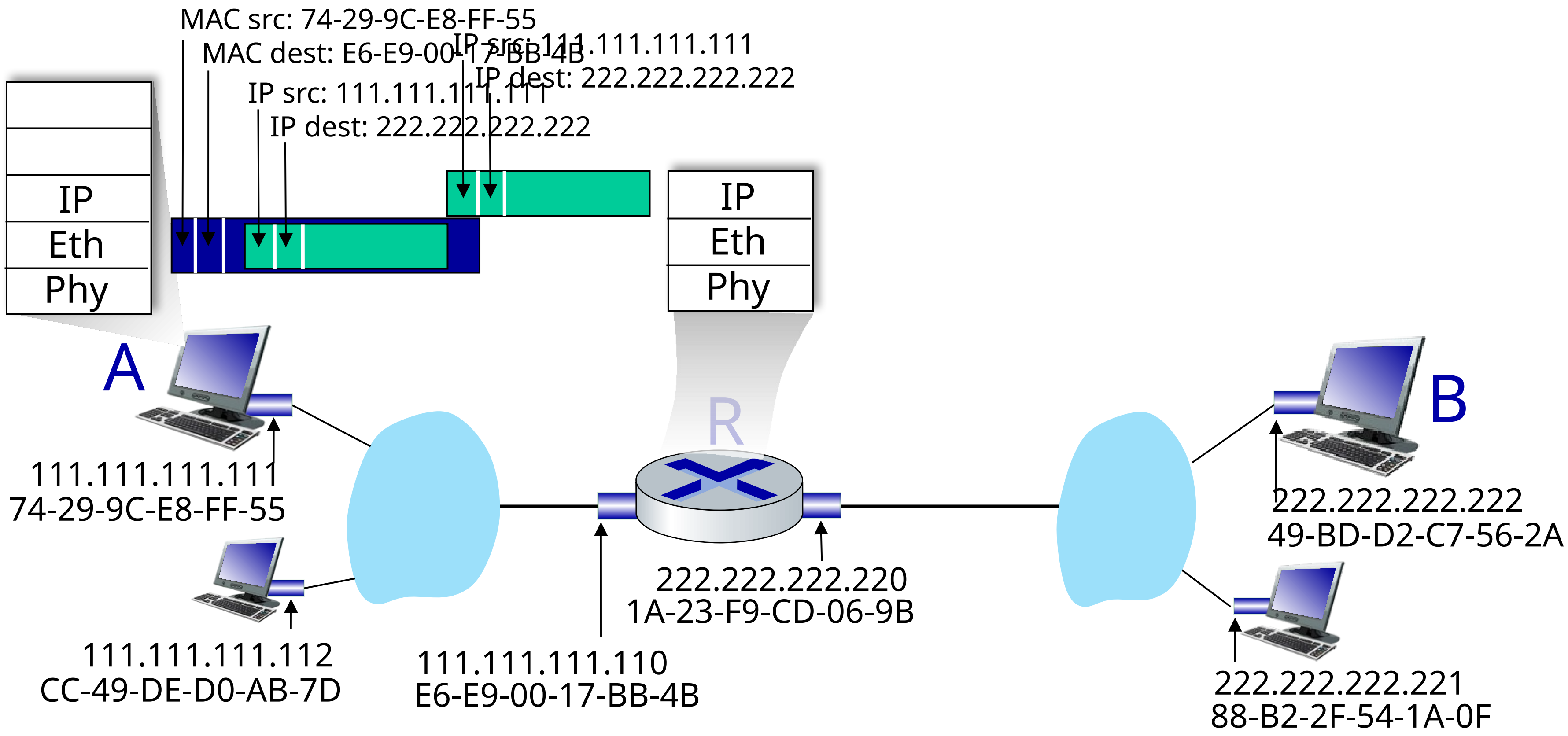
Routing to another subnet: addressing

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame containing A-to-B IP datagram
  - R's MAC address is frame's destination



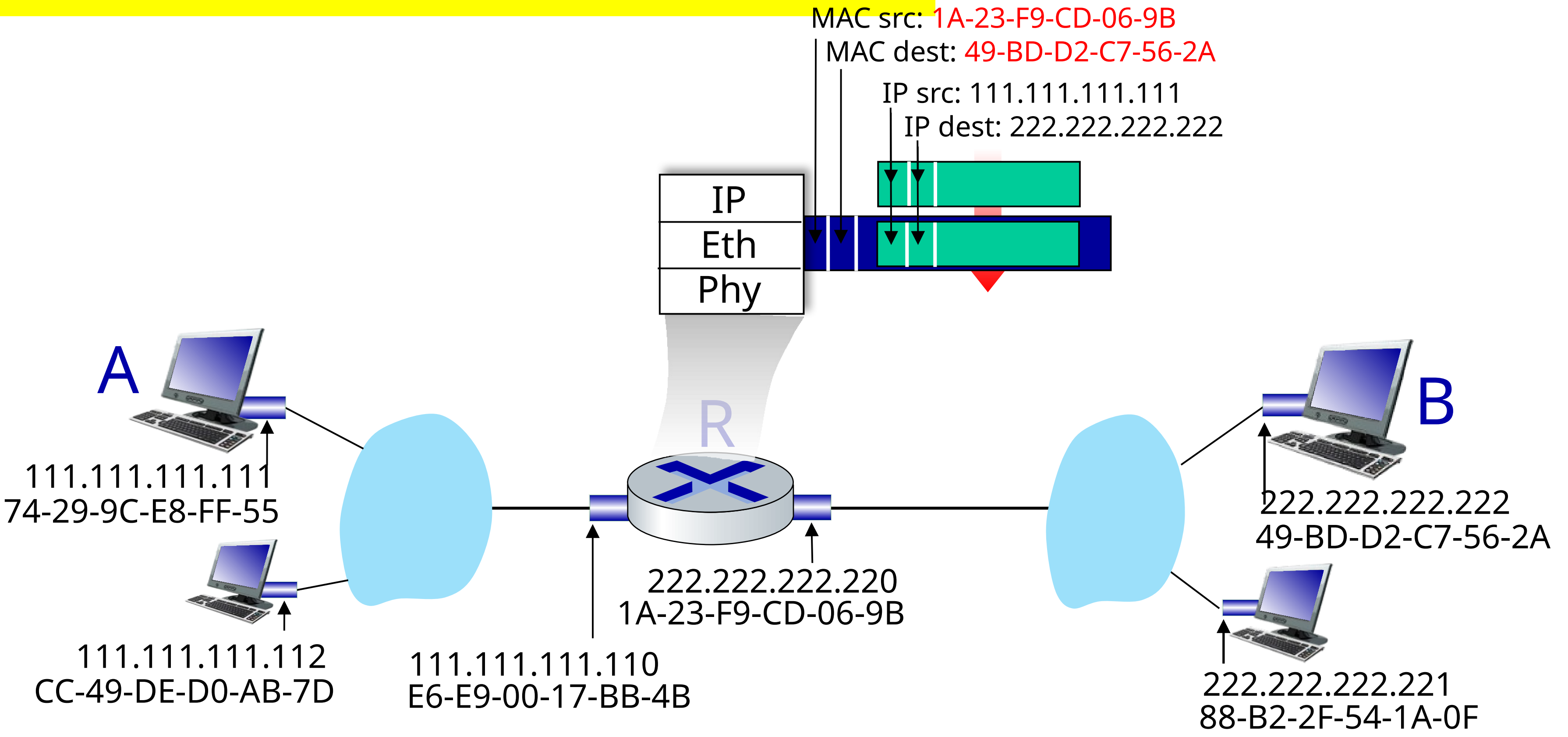
Routing to another subnet: addressing

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



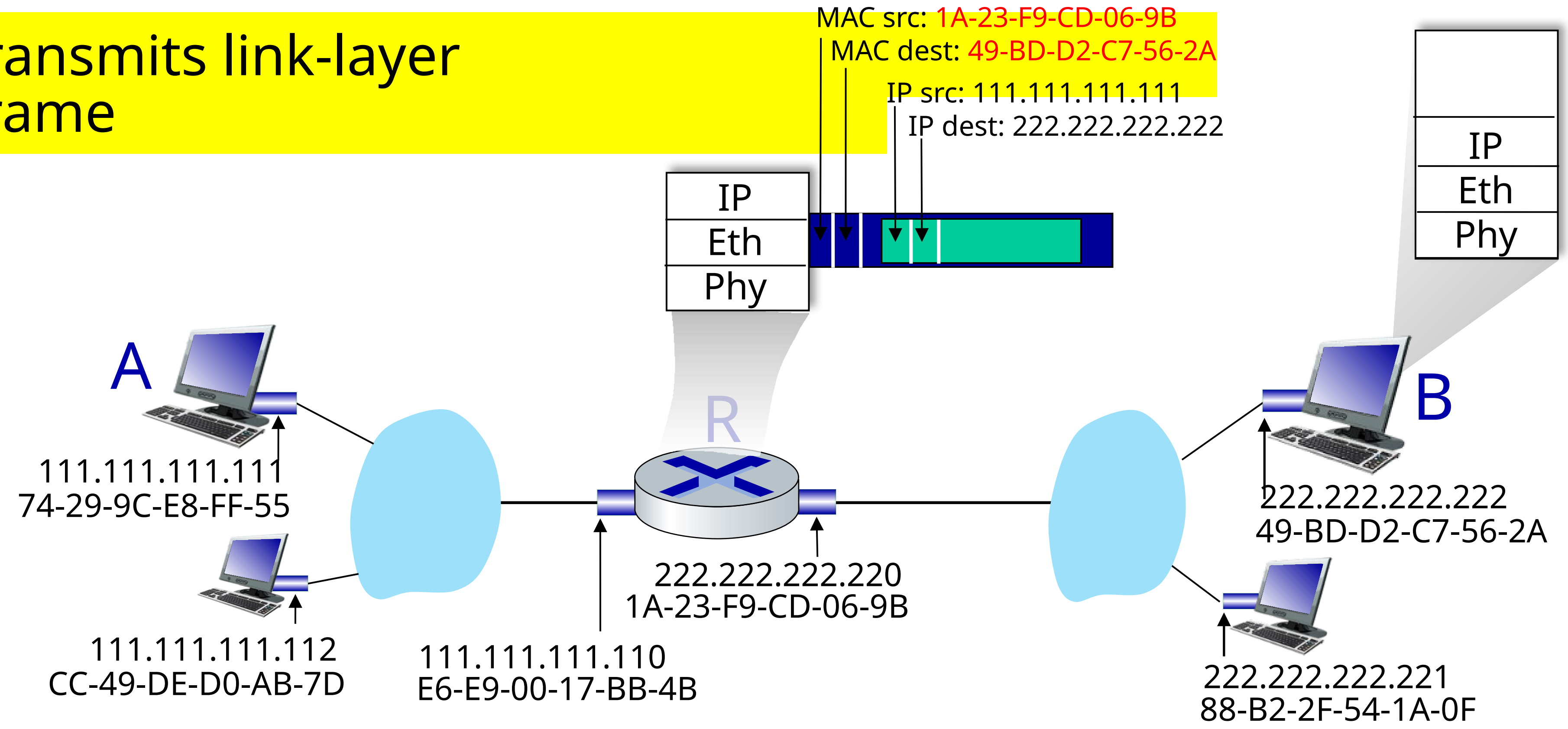
Routing to another subnet: addressing

- R determines outgoing interface, passes datagram with IP source A, destination B to link layer
- R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address



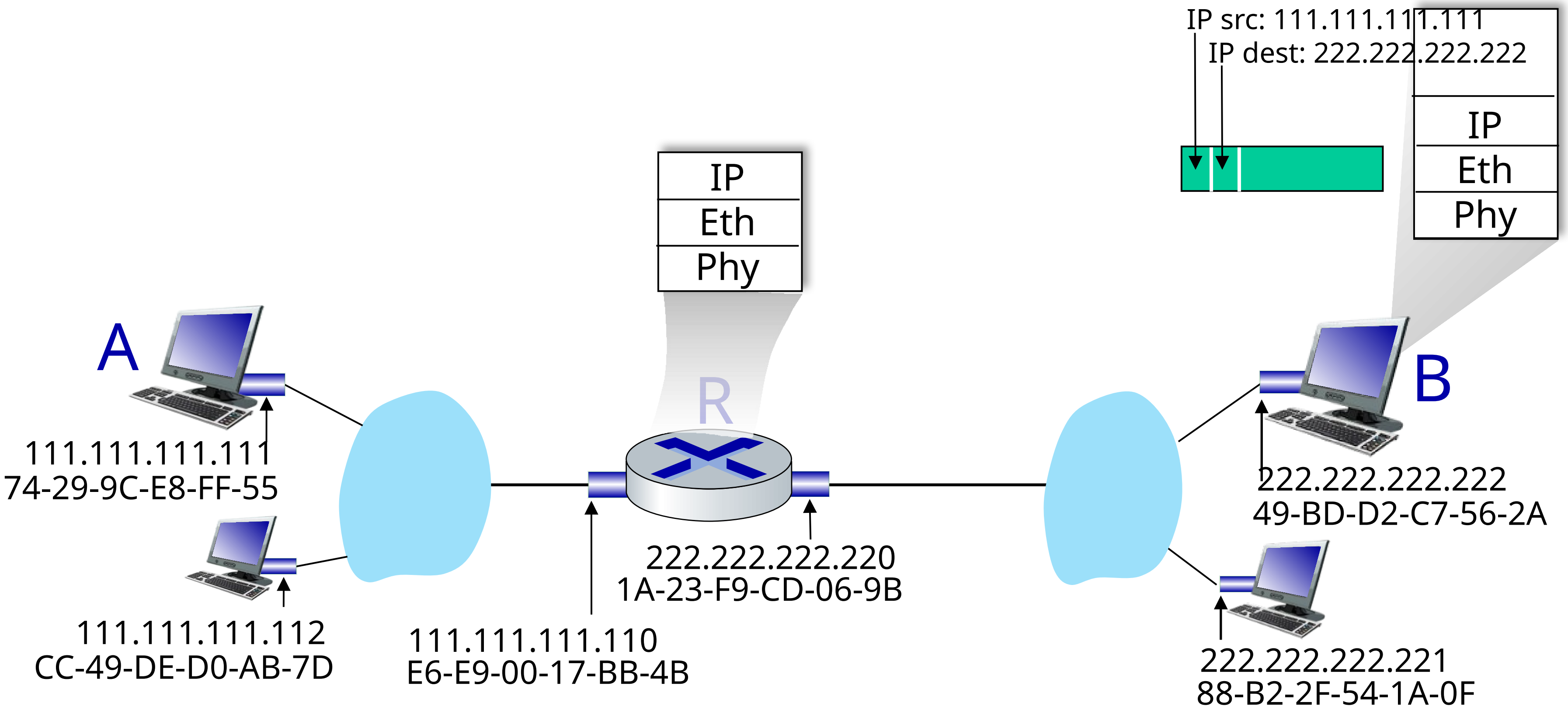
Routing to another subnet: addressing

- R determines outgoing interface, passes datagram with IP source A, destination B to link layer
- R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address
- transmits link-layer frame



Routing to another subnet: addressing

- B receives frame, extracts IP datagram destination B
- B passes datagram up protocol stack to IP



Link layer, LANs: roadmap

- ~~introduction~~
- ~~error detection, correction~~
- ~~multiple access protocols~~
- LANs
  - addressing, ARP
  - Ethernet
  - switches

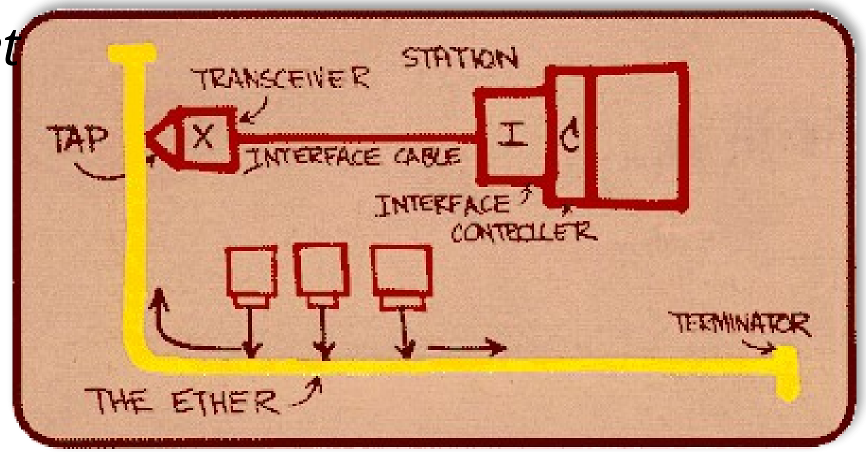
- a day in the life of a web request

Ethernet

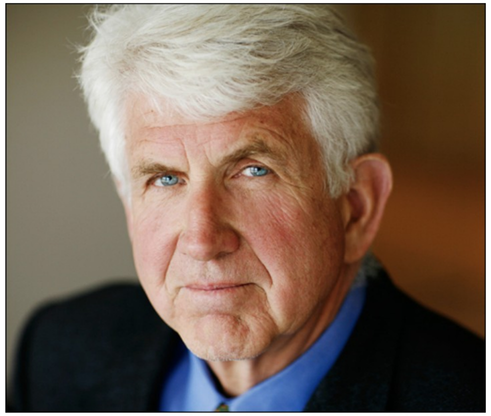
"dominant" wired LAN technology:

- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps – 400 Gbps
- single chip, multiple speeds (e.g., Broadcom BCM5761)

Metcalfe's Ethernet sketch



Bob Metcalfe: Ethernet co-inventor, 2022 ACM Turing Award recipient



https://www.uspto.gov/learning-and-resources/journeys-innovation/audio-stories/defying-doubters

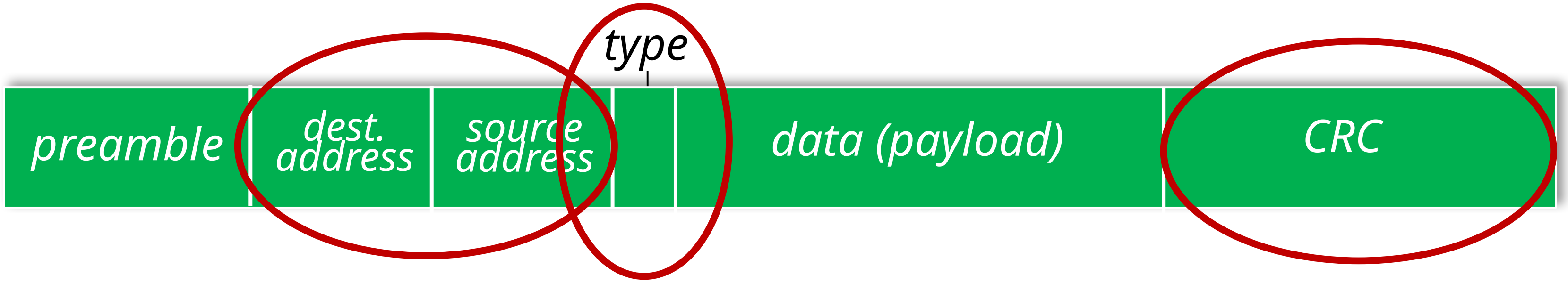
Ethernet frame structure

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

preamble:

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

Ethernet frame structure (more)



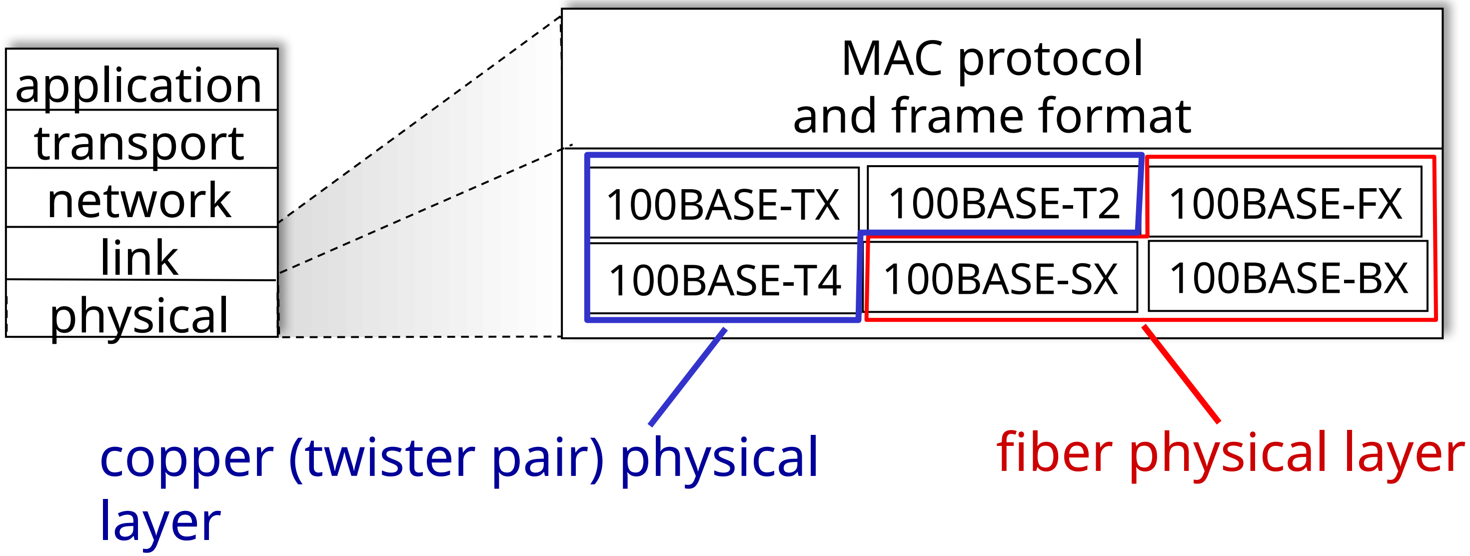
- addresses: 6 byte source, destination MAC addresses
  - if adapter receives frame with matching destination address, or with broadcast address (e.g., ARP packet), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame
- type: indicates higher layer protocol
  - mostly IP but others possible, e.g., Novell IPX, AppleTalk
  - used to demultiplex up at receiver
- 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 doesn't send ACKs or NAKs 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 with binary backoff

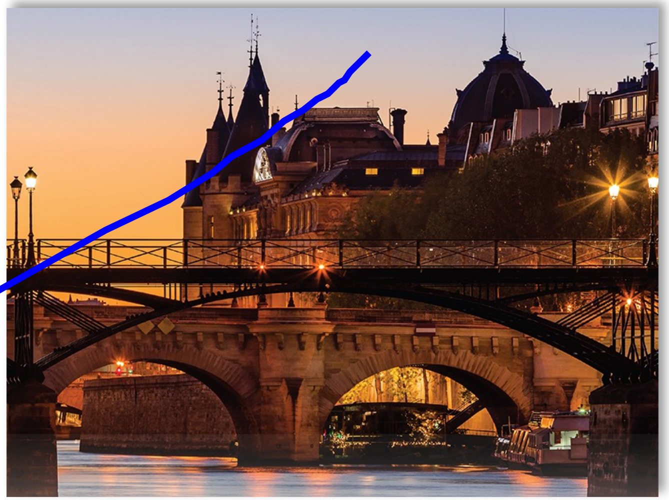
802.3 Ethernet standards: link & physical layers

- *many* different Ethernet standards
  - common MAC protocol and frame format
  - different speeds: 2 Mbps, ... 100 Mbps, 1Gbps, 10 Gbps, 40 Gbps, 80 Gbps
  - different physical layer media: fiber, cable



Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- LANs
  - addressing, ARP
  - Ethernet
  - switches



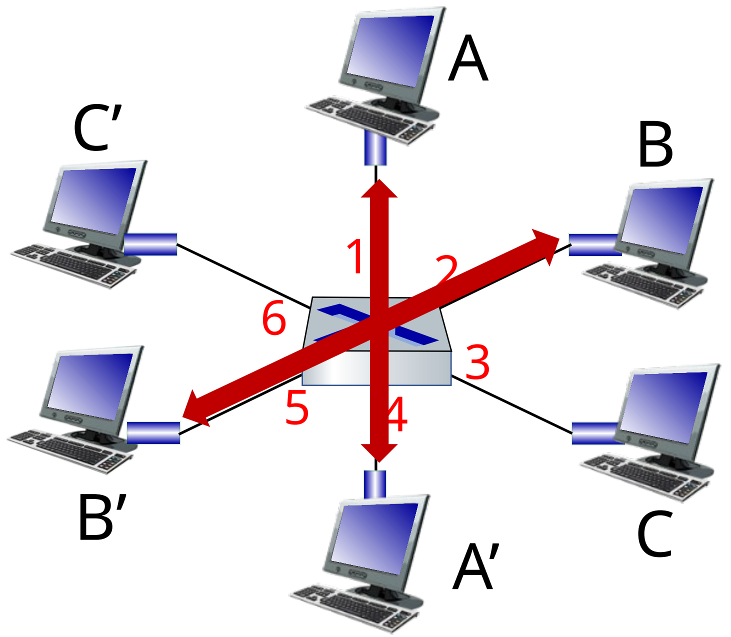
- a day in the life of a web request

Ethernet switch

- Switch is a link-layer device: takes an *active* role
  - store, forward Ethernet (or other type of) 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 *unaware* of presence of switches
- plug-and-play, self-learning
  - switches do not need to be configured

Switch: multiple simultaneous transmissions

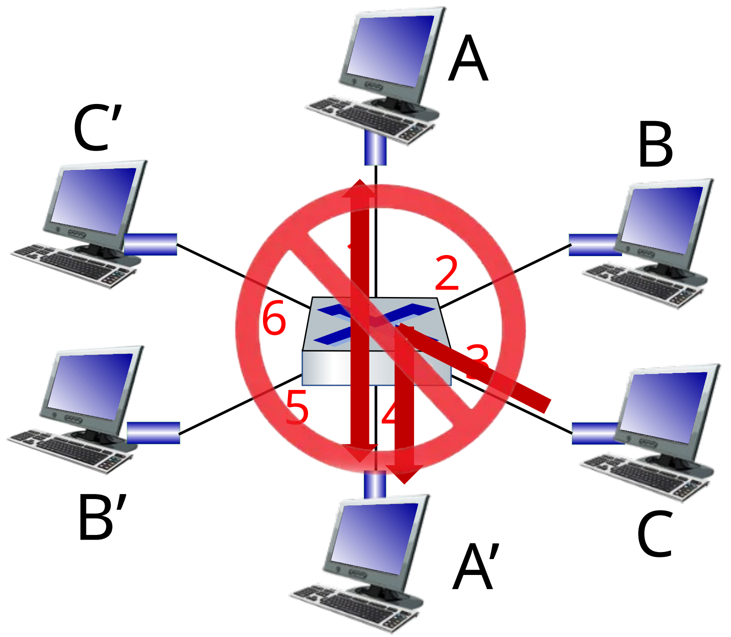
- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, so:
  - no collisions; full duplex
  - each link is its own collision domain
- switching: A-to-A' and B-to-B' can transmit simultaneously, without collisions



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

Switch: multiple simultaneous transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, so:
  - no collisions; full duplex
  - each link is its own collision domain
- switching: A-to-A' and B-to-B' can transmit simultaneously, without collisions
  - but A-to-A' and C to A' can *not* happen simultaneously



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

Switch forwarding table

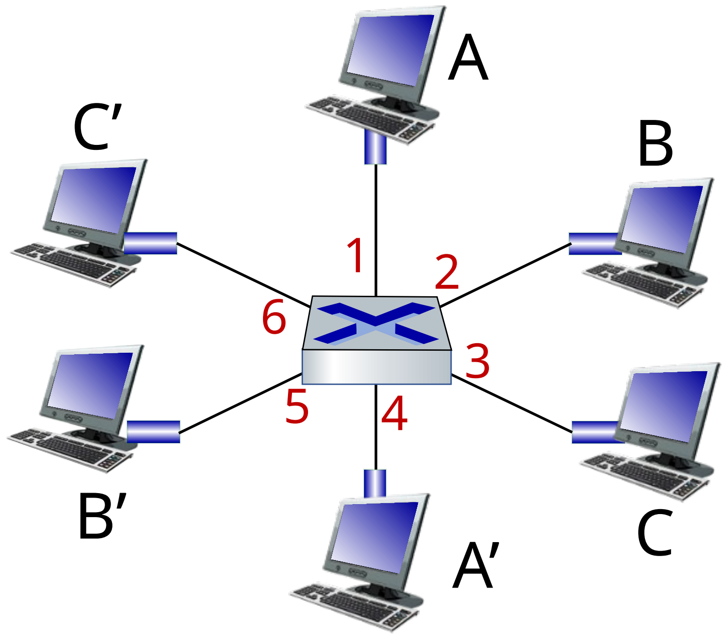
Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

A: each switch has a switch table, each entry:

- (MAC address of host, interface to reach host, time stamp)
- looks like a routing table!

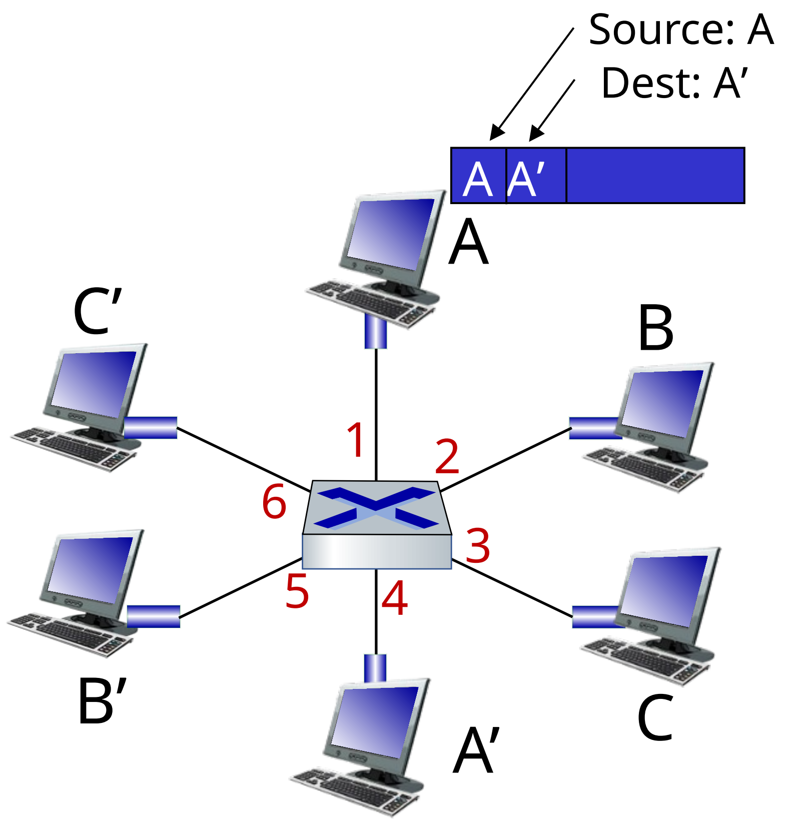
Q: how are entries created, maintained in switch table?

- something like a routing protocol?



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)

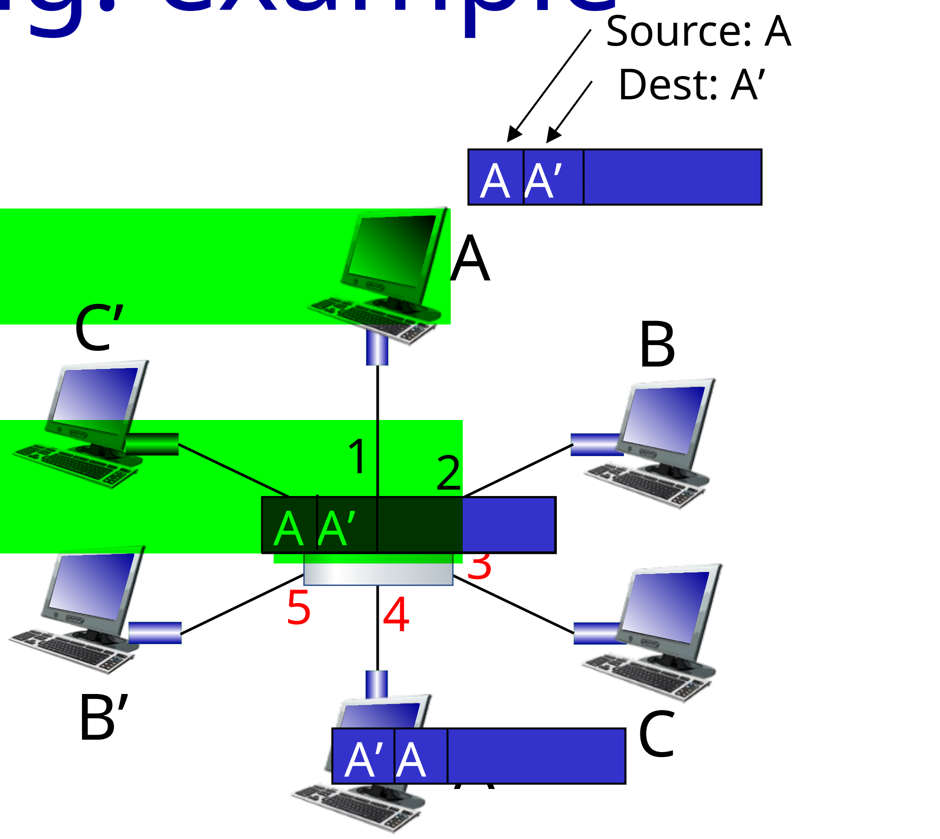
Switch: frame filtering/forwarding

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 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 */

Self-learning, forwarding: example

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

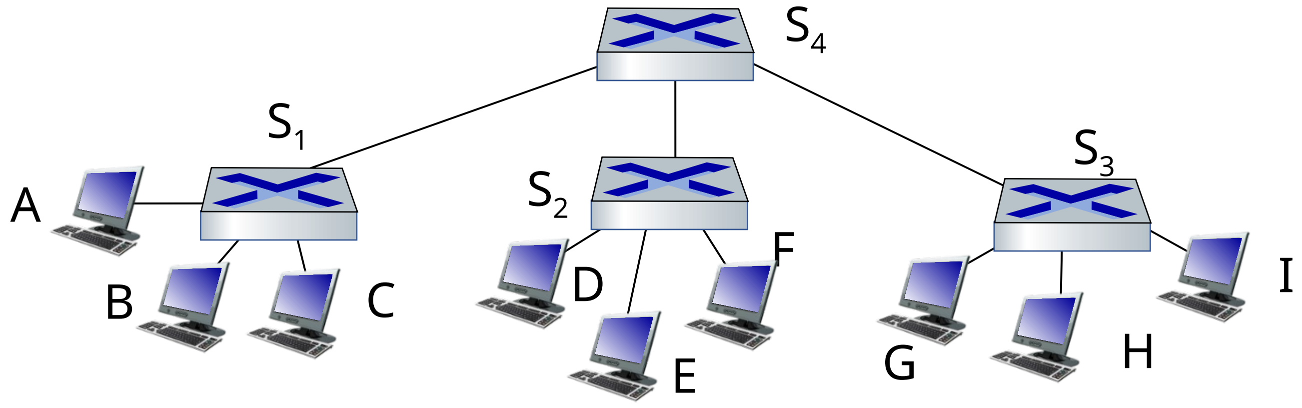


MAC addr	interface	TTL
A	*1*	*60*
A'	*4*	*60*

switch table (initially empty)

Interconnecting switches

self-learning switches can be connected together:

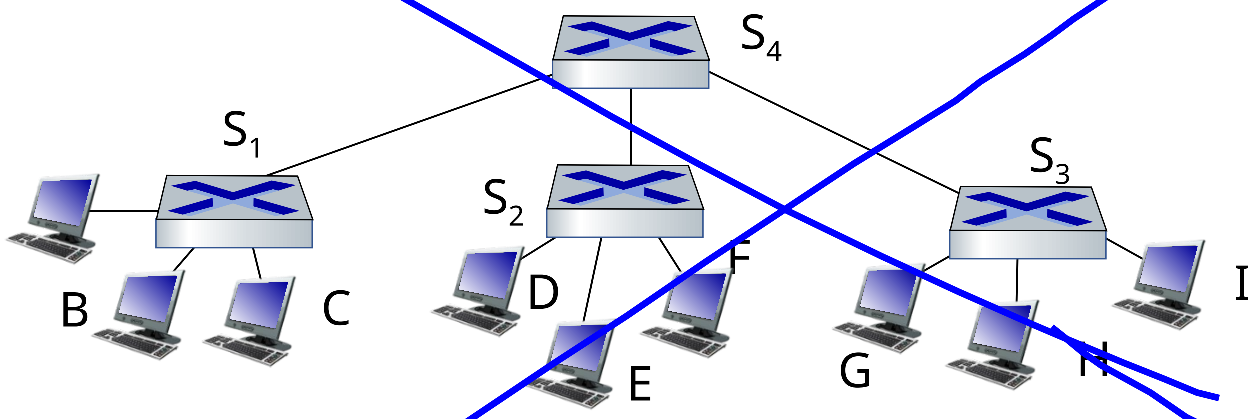


Q: sending from A to G - how does S_1 know to forward frame destined to G via S_4 and S_3?

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

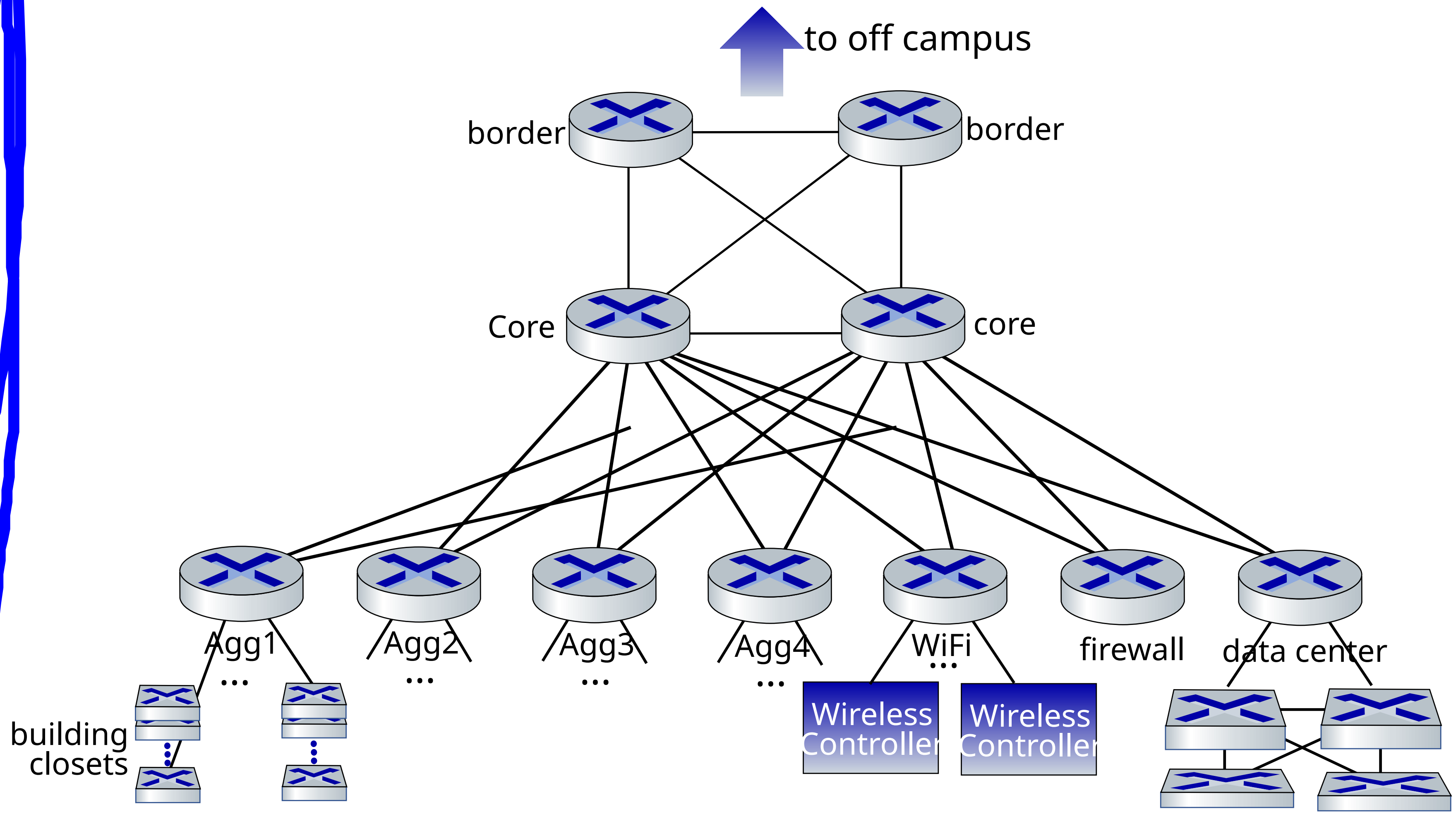
Self-learning multi-switch example

Suppose C sends frame to I, I responds to C



Q: show switch tables and packet forwarding in S_1, S_2, S_3, S_4

UMass Campus Network - Detail

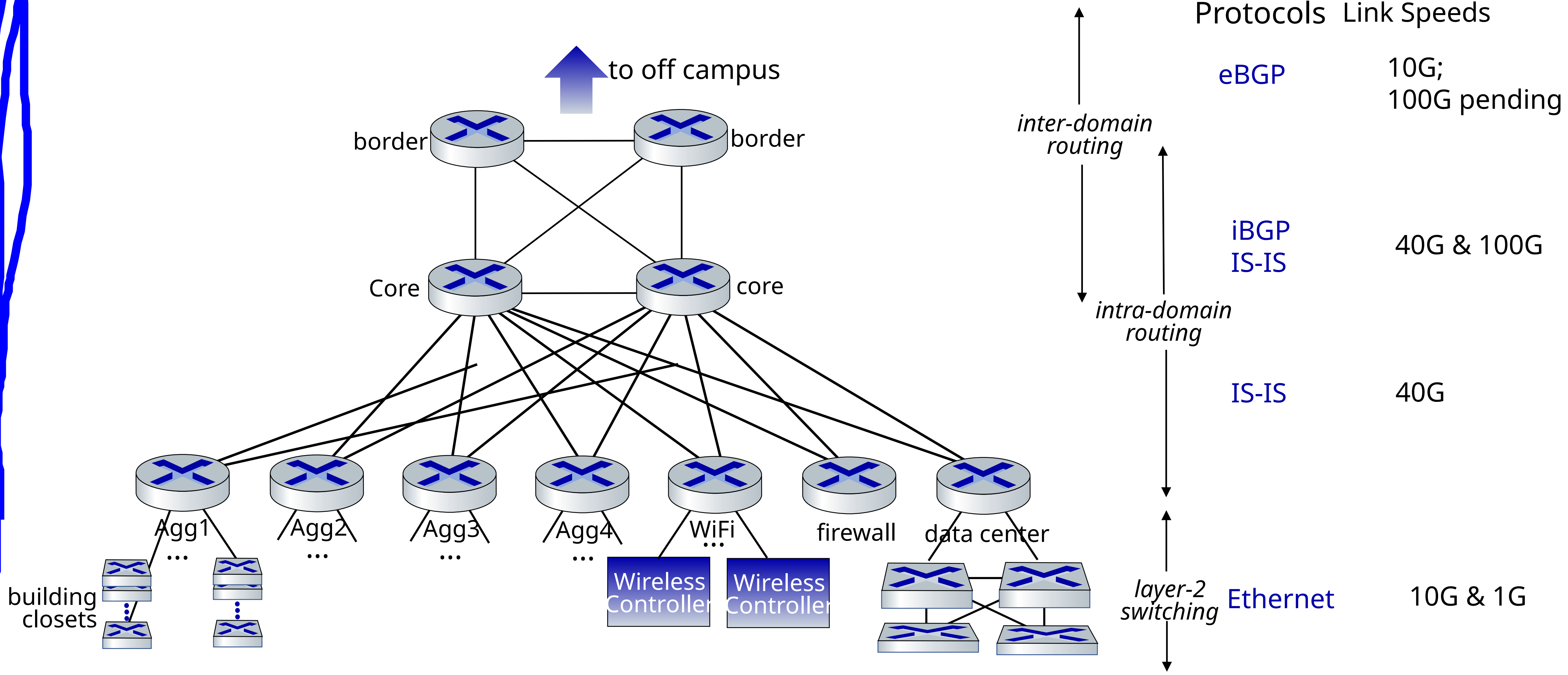


UMass network:

- 4 firewalls
- 10 routers
- 2000+ network switches
- 6000 wireless access points
- 30000 active wired network jacks
- 55000 active end-user wireless devices

… all built, operated, maintained by ~15 people

UMass Campus Network - Detail



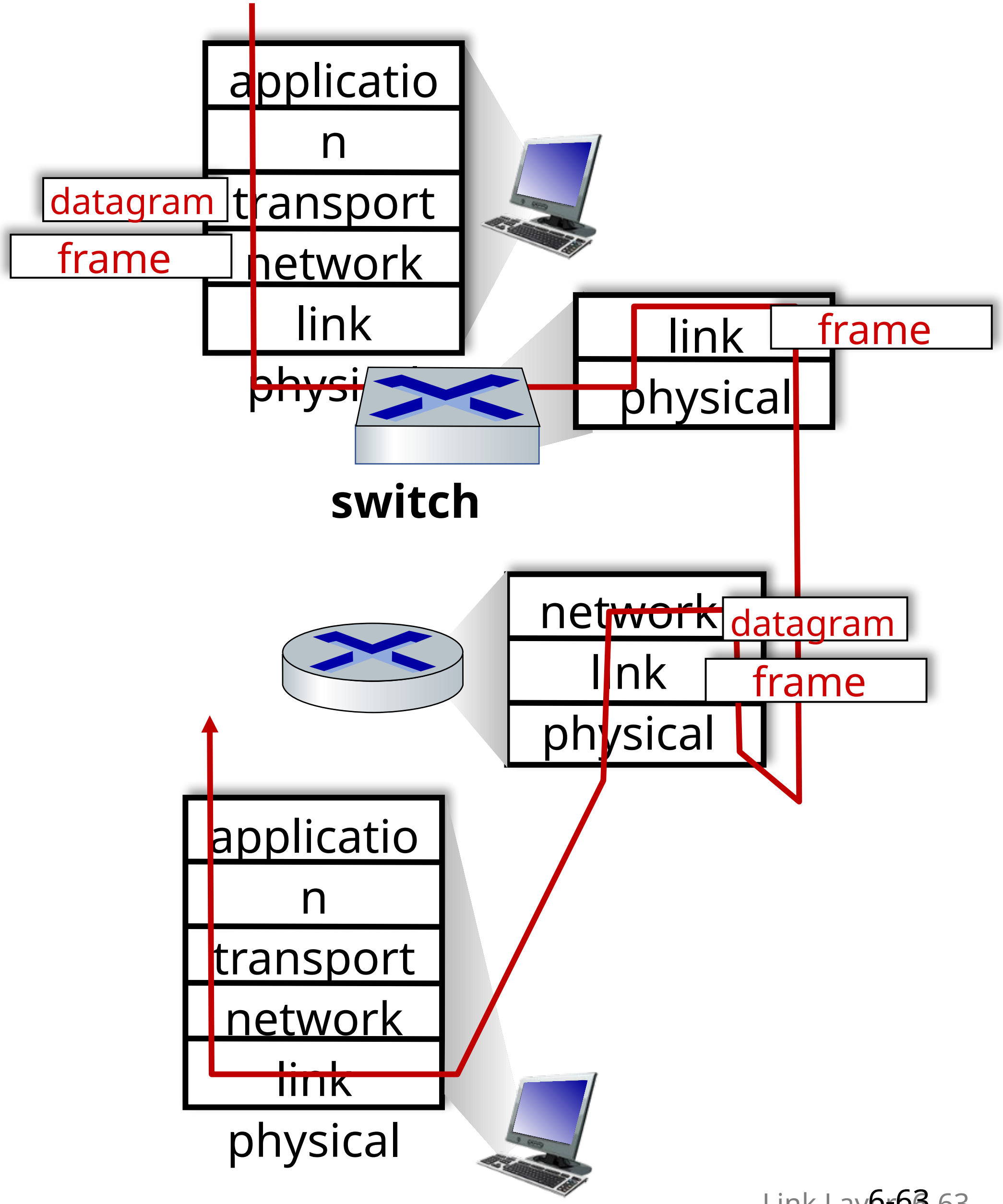
Switches vs. routers

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



Link layer, LANs: roadmap

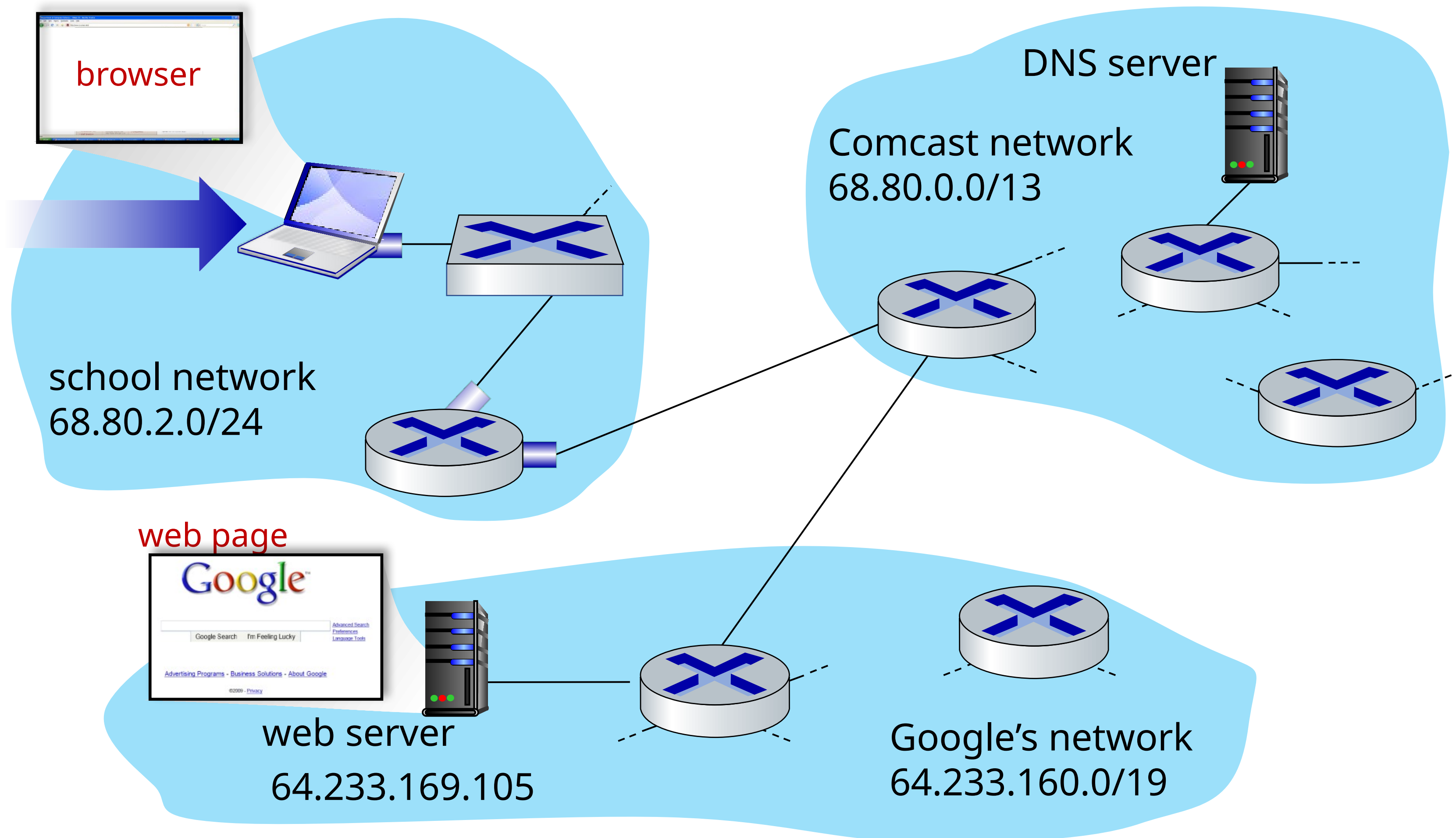
- ~~introduction~~
- ~~error detection, correction~~
- ~~multiple access protocols~~
- ~~LANs~~
  - ~~addressing, ARP~~
  - ~~Ethernet~~
  - ~~switches~~

- a day in the life of a web request

Synthesis: a day in the life of a web request

- our journey down the protocol stack is now complete!
  - application, transport, network, link
- putting-it-all-together: synthesis!
  - *goal:* identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
  - *scenario:* student attaches laptop to campus network, requests/receives www.google.com

A day in the life: scenario

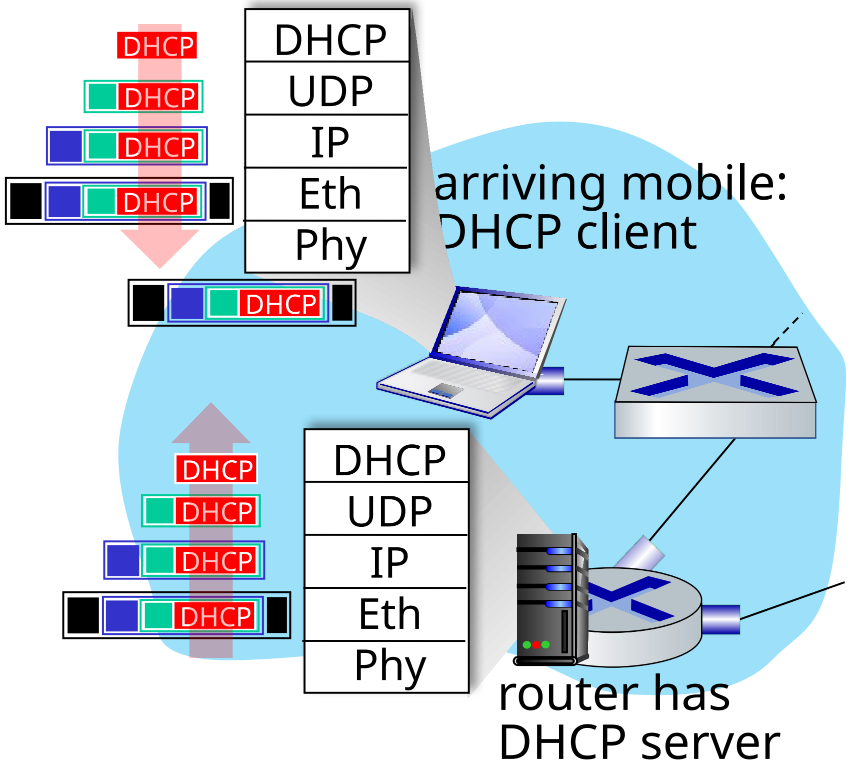


scenario:

- arriving mobile client attaches to network …
- requests web page: www.google.com

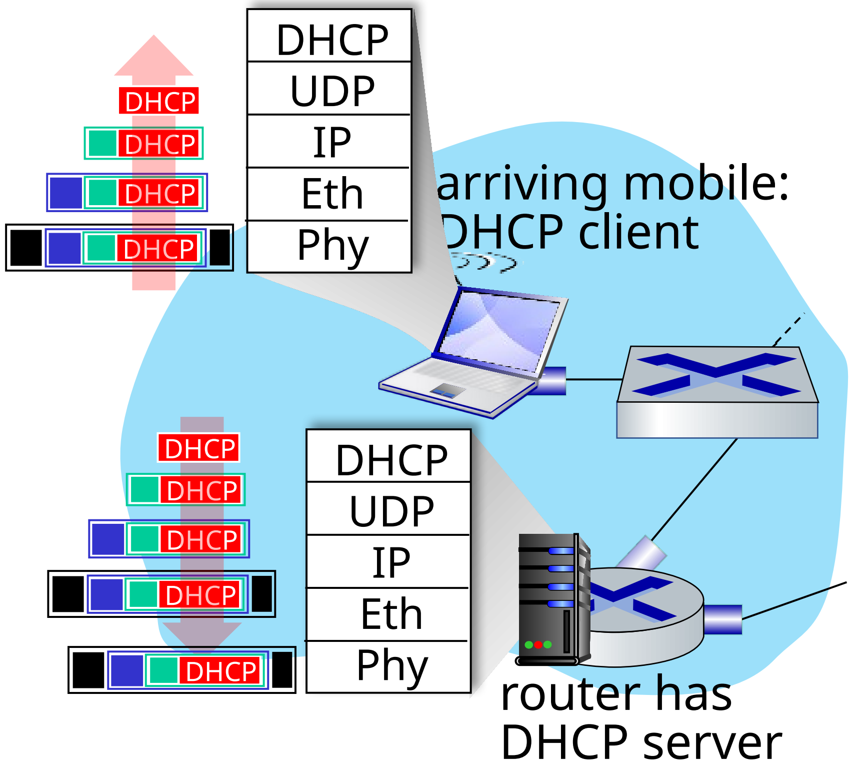
Sounds simple!

A day in the life: 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 de-muxed to IP de-muxed, UDP de-muxed to DHCP

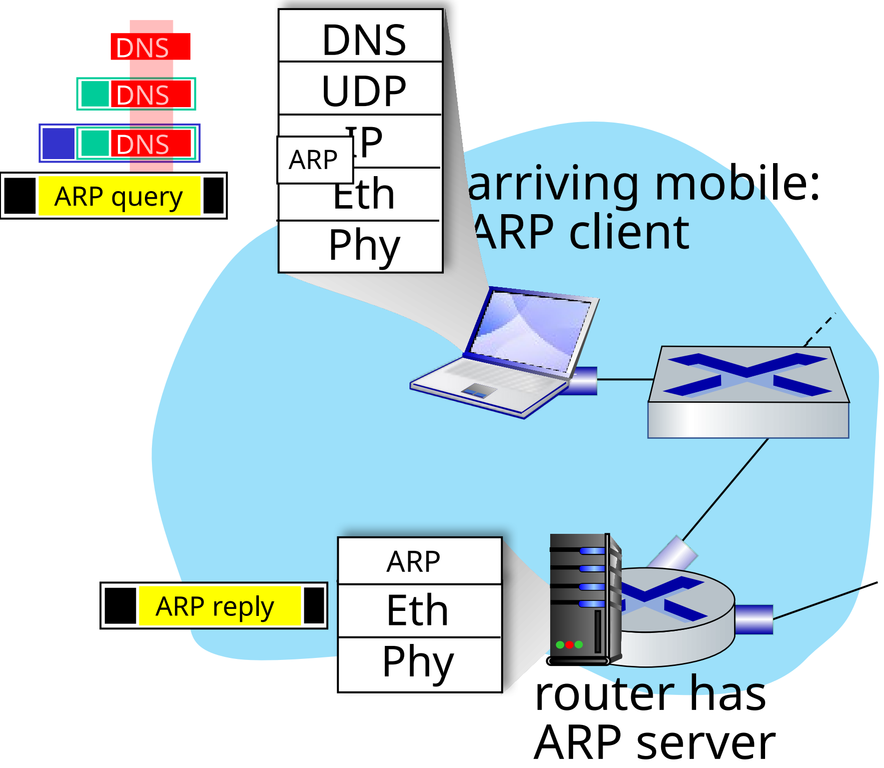
A day in the life: connecting to the Internet



- 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

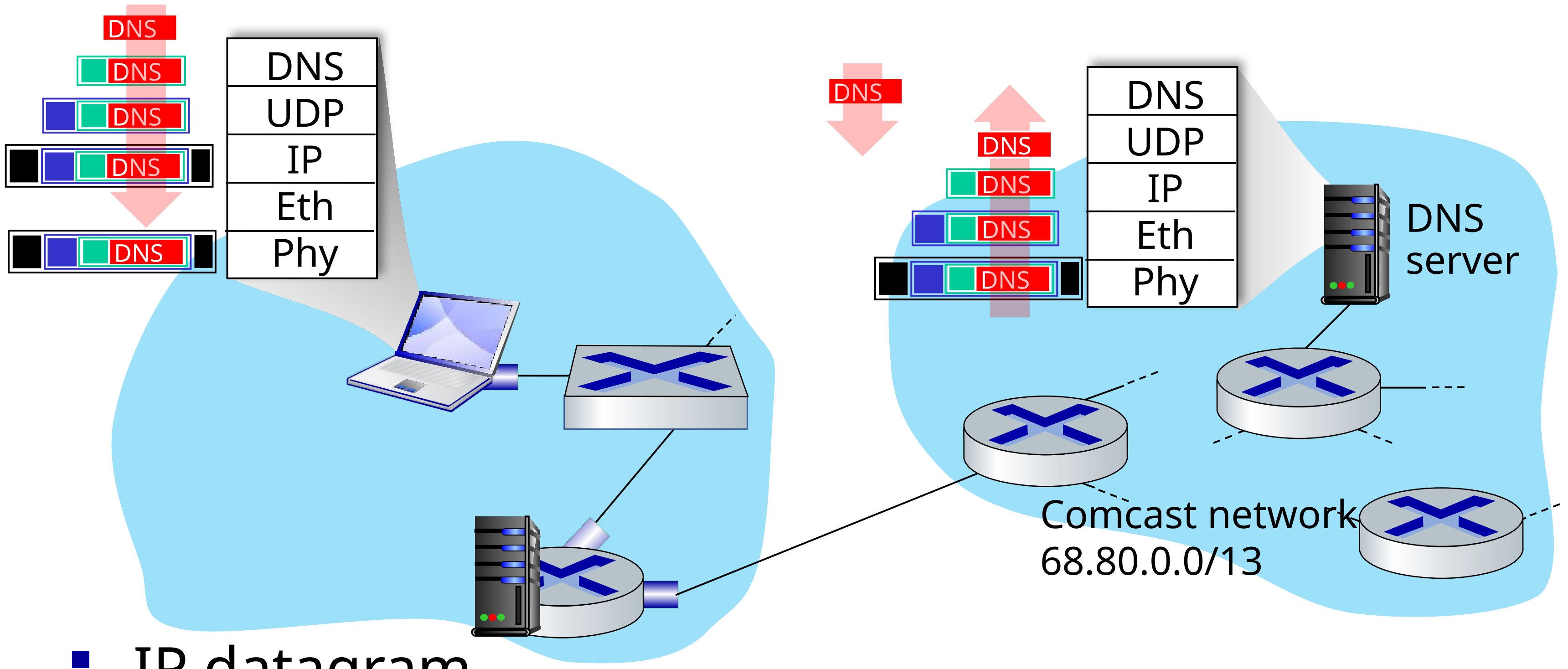
Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

A day in the life... 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

A day in the life... using DNS

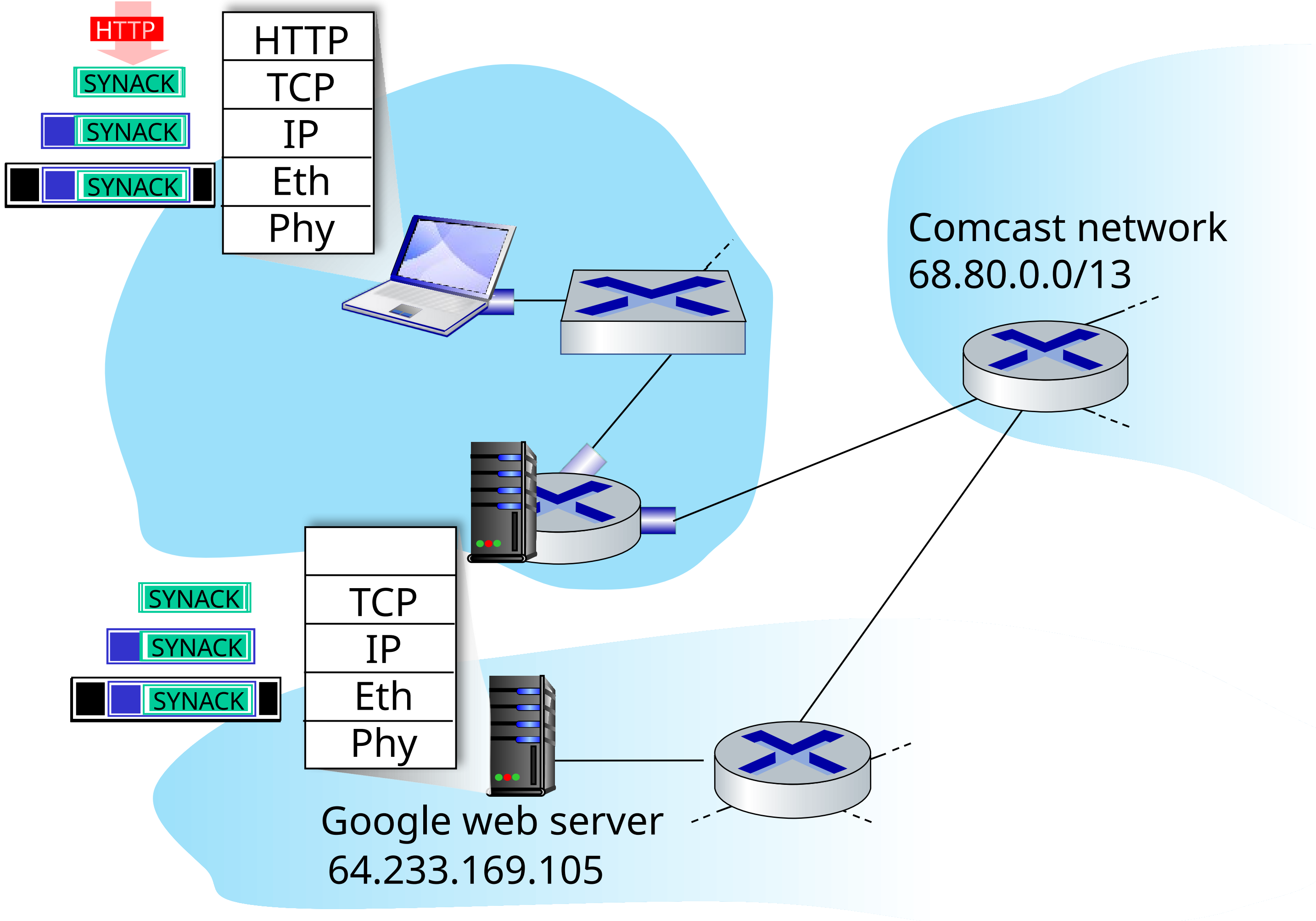


- de-muxed to DNS
- DNS replies to client with IP address of www.google.com

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

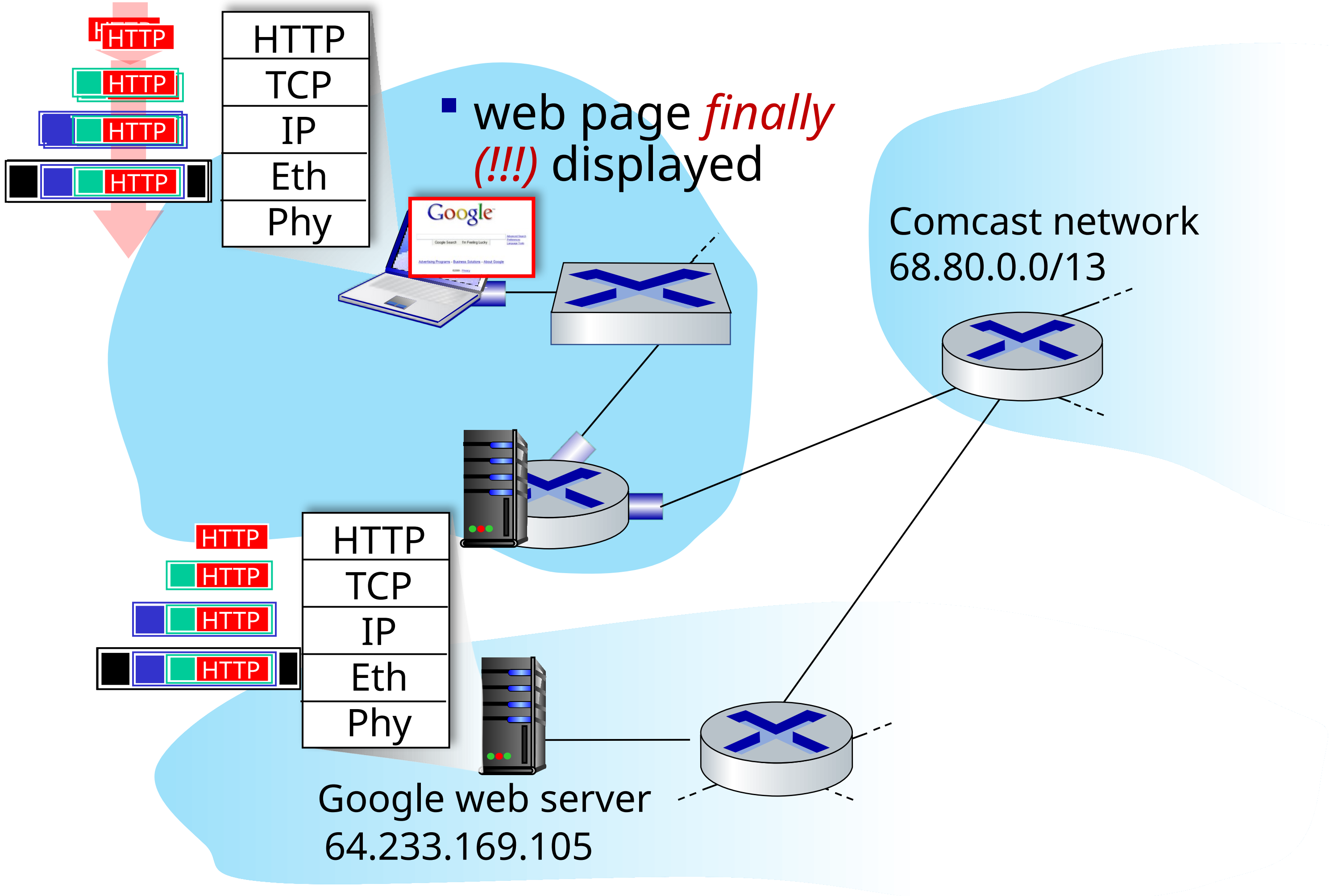
- IP datagram forwarded from campus network into Comcast network, routed (tables created by RIP, OSPF, IS-IS and/or BGP routing protocols) to DNS server

A day in the life…TCP connection carrying HTTP



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

A day in the life… 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