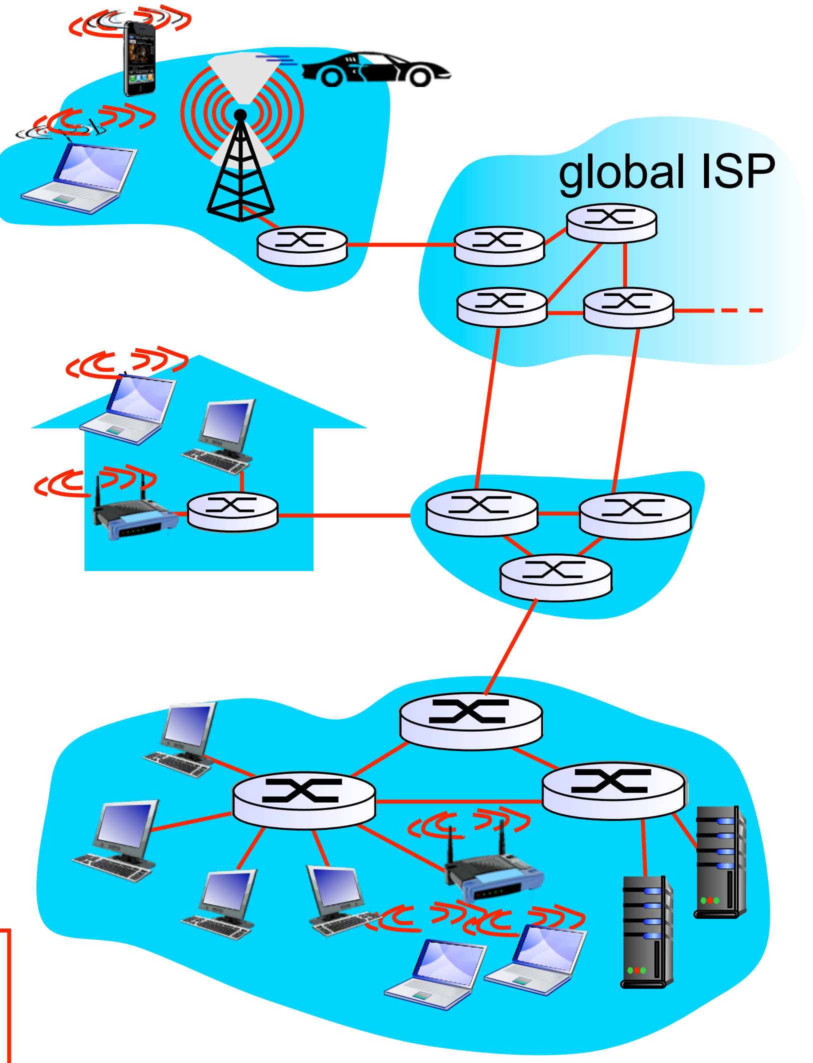
Link layer: introduction

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

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:

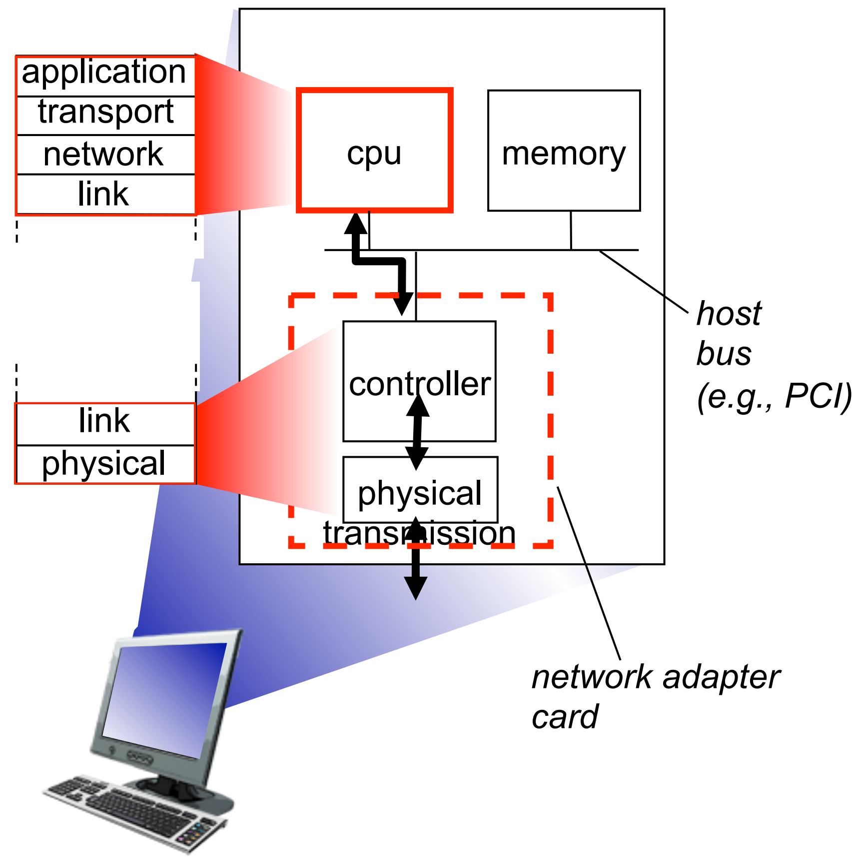
- trip from Princeton to Lausanne
  - limo: Princeton to JFK
  - plane: JFK to Geneva
  - train: Geneva to Lausanne
- tourist = datagram
- transport segment = communication link
- transportation mode = link layer protocol
- travel agent = routing algorithm

Link layer services

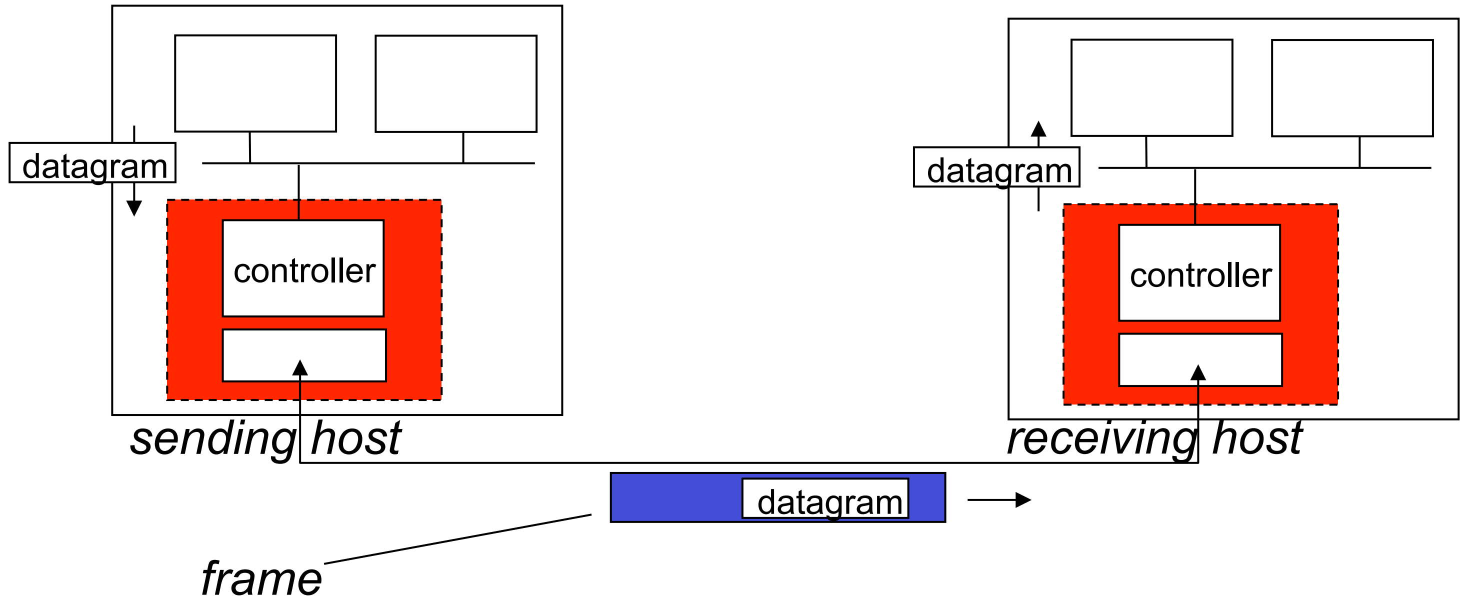
- *framing, link access:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses used in frame headers to identify source, dest
    - different from IP address!

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 chips



Adaptors communicating



- sending side:
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.
- receiving side
  - looks for errors, rdt, flow control, etc
  - extracts datagram, passes to upper layer at receiving side

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 (for most LANs) burned in NIC ROM, also sometimes software settable
  - e.g.: 1~~A~~-2F-BB-76-09-AD

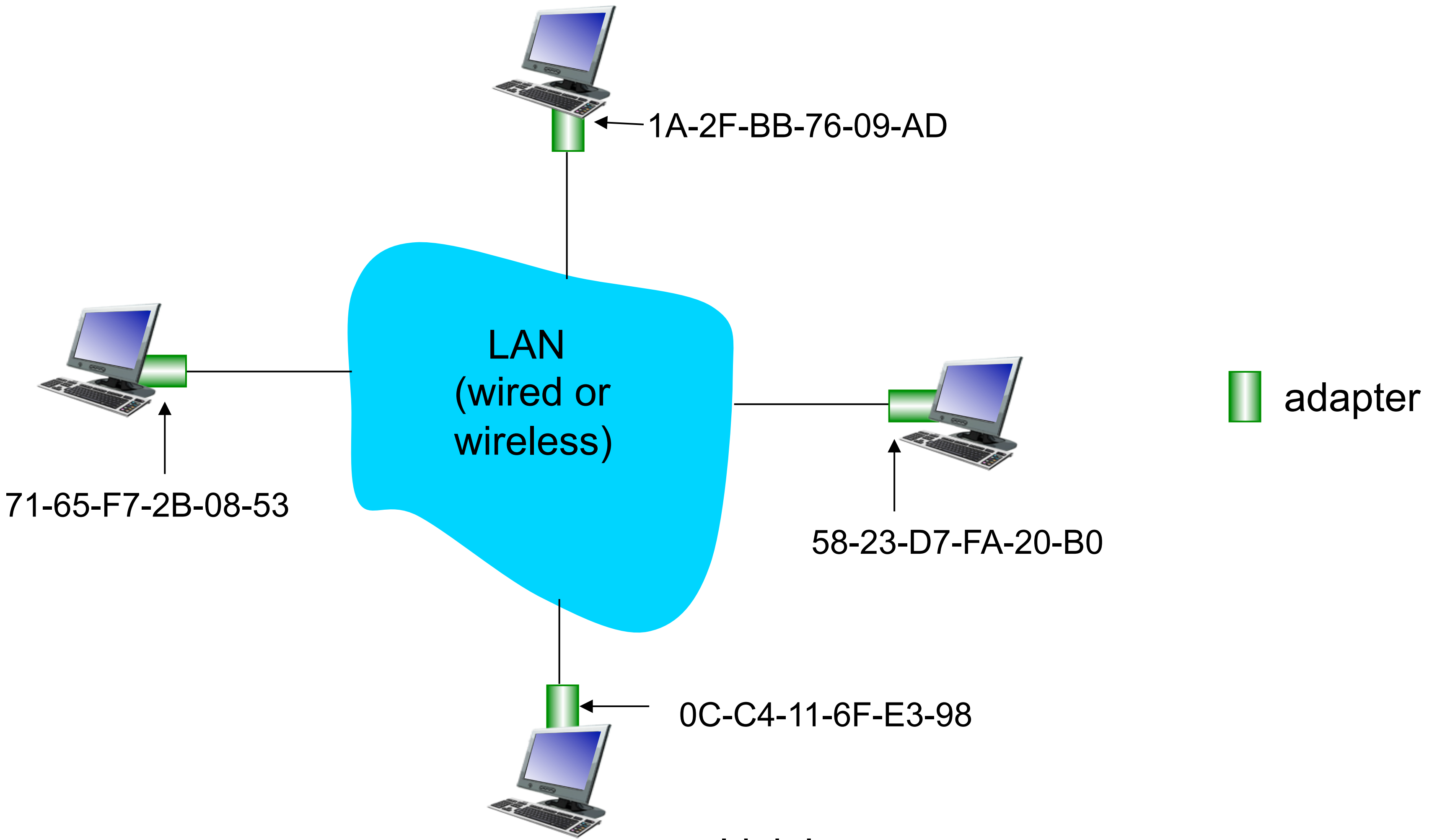
hexadecimal (base 16) notation
(each "number" represents 4 bits)

What is a LAN?

- Local area network
- We will use the term "Local Area Network" (LAN or LAN segment; both are used interchangeably) to refer to each Ethernet shared link
  - The term LAN can also be used to refer to an access point and all hosts associated with it

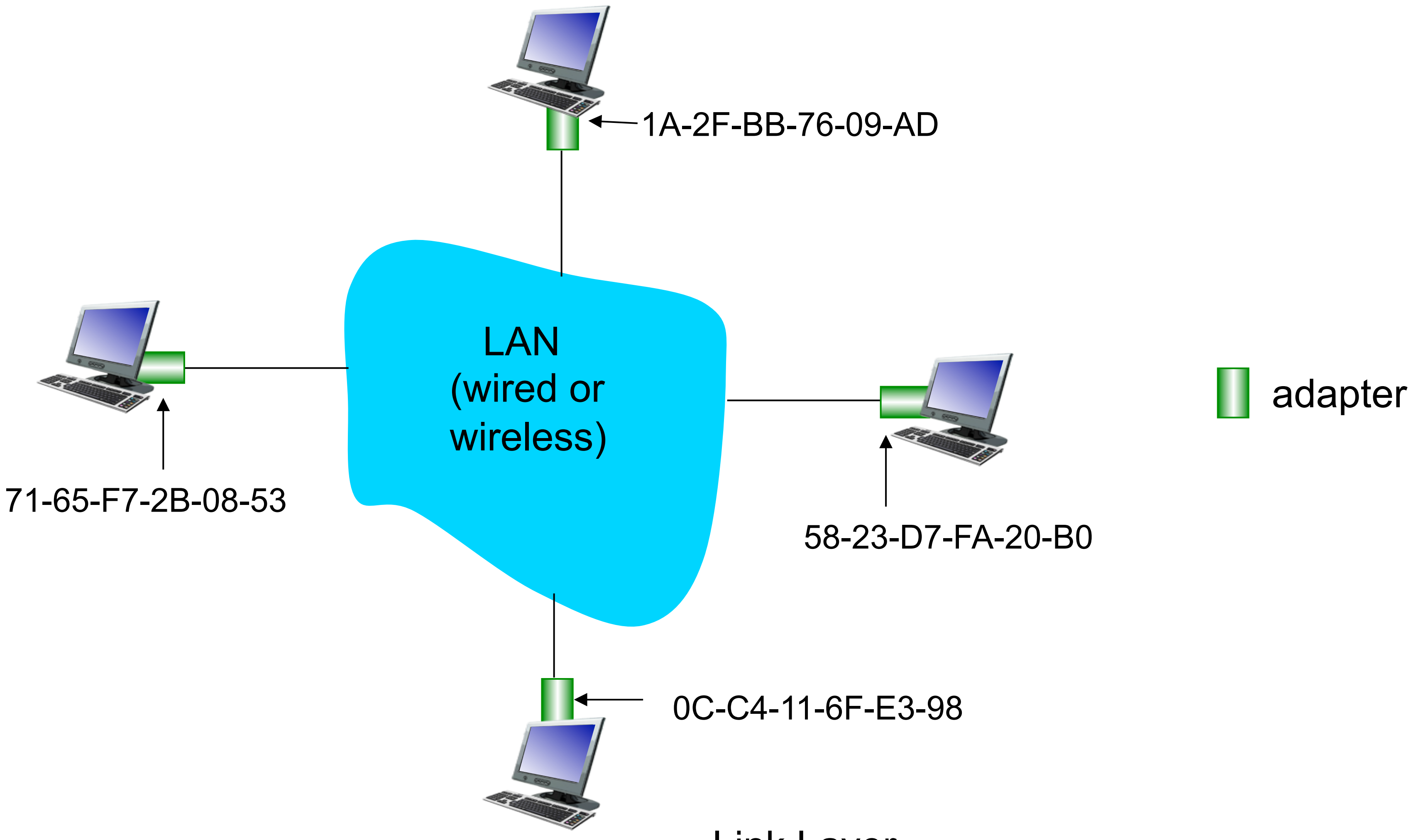
LAN addresses and ARP

each adapter on LAN has unique *LAN* address



LAN addresses and ARP

each adapter on LAN has unique *LAN* address

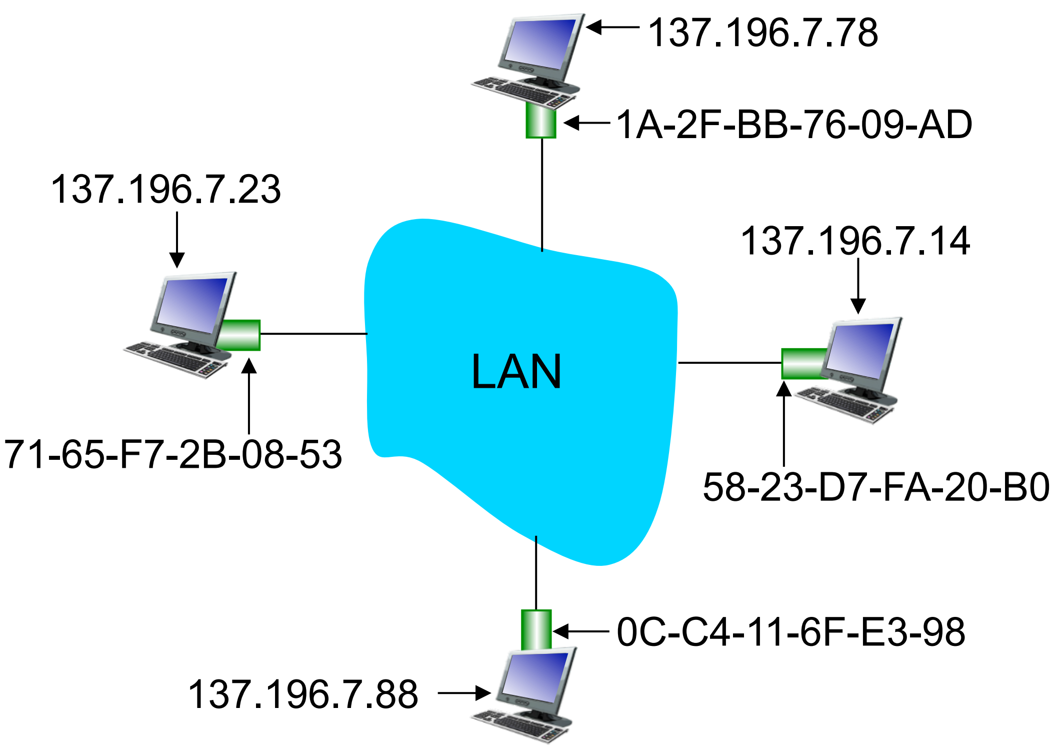


LAN addresses (more)

- 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
- 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

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)

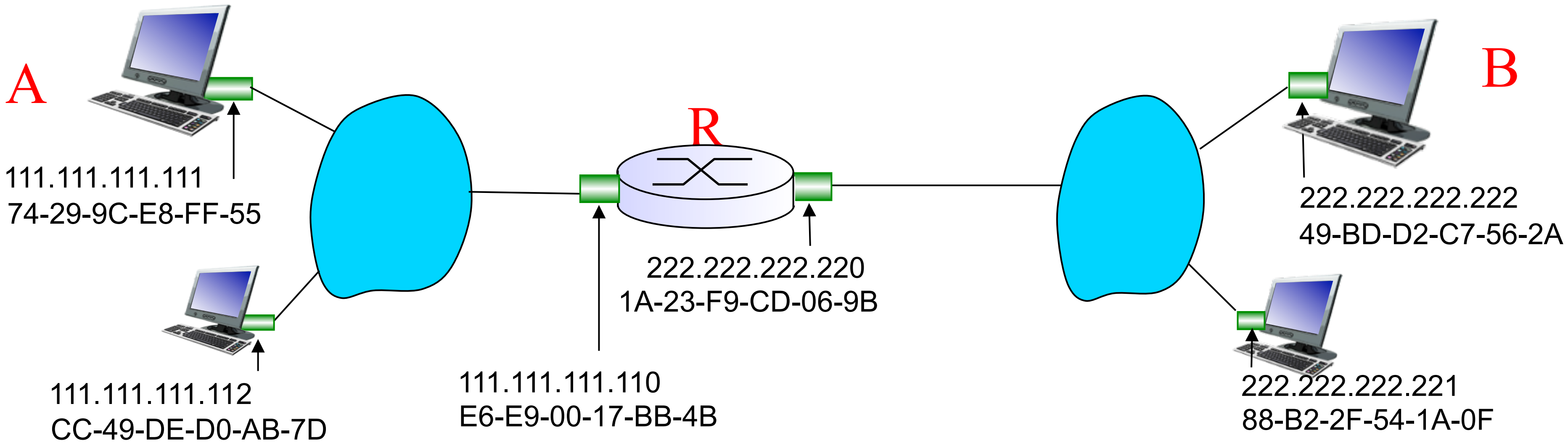
ARP protocol: same LAN

- A wants to send datagram to B
  - B's MAC address not in A's ARP table.
- 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: information that times out (goes away) unless refreshed
- ARP is "plug-and-play":
  - nodes create their ARP tables *without intervention from net administrator*

Addressing: routing to another LAN

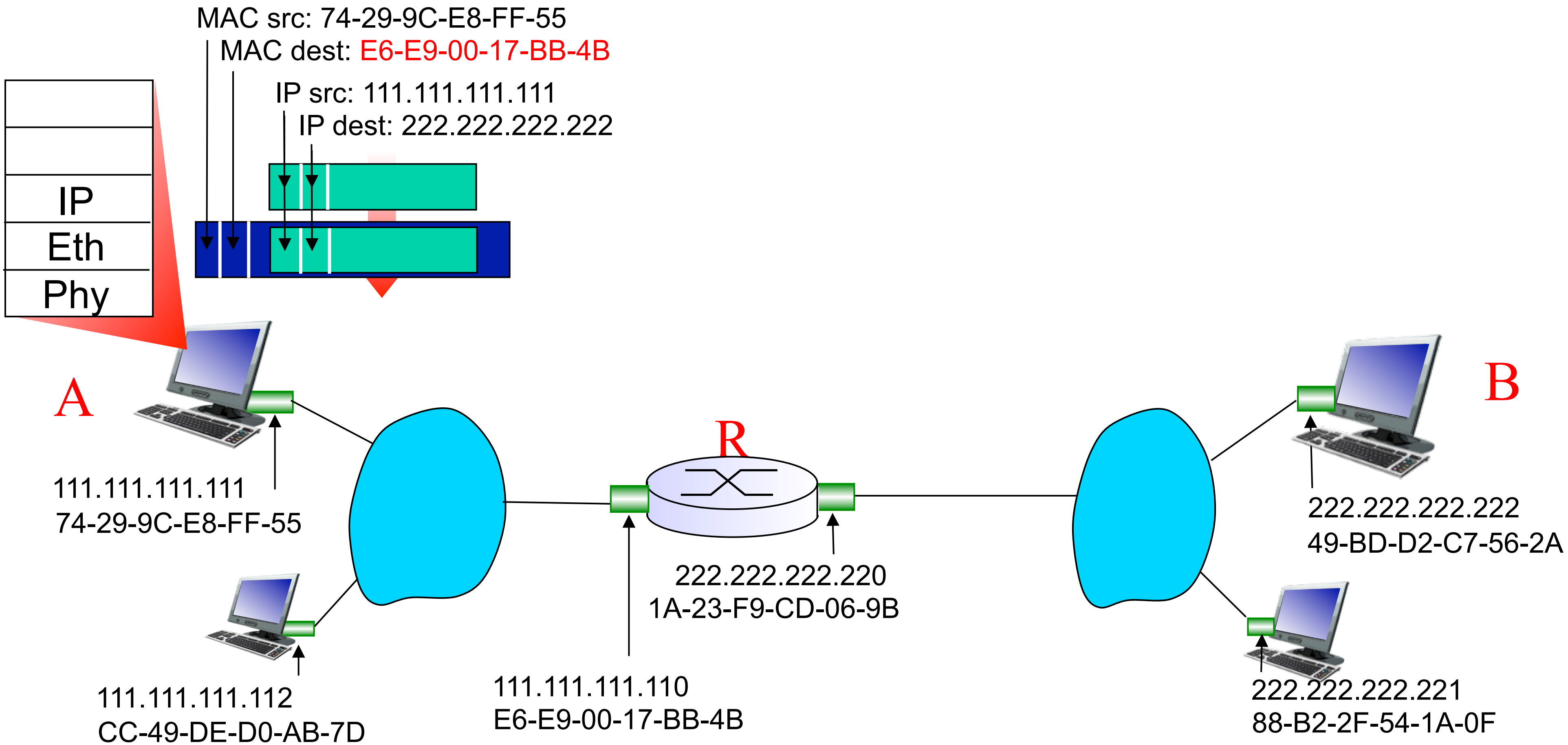
walkthrough: send datagram from A to B via R

- 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 (how?)
- assume A knows R's MAC address (how?)



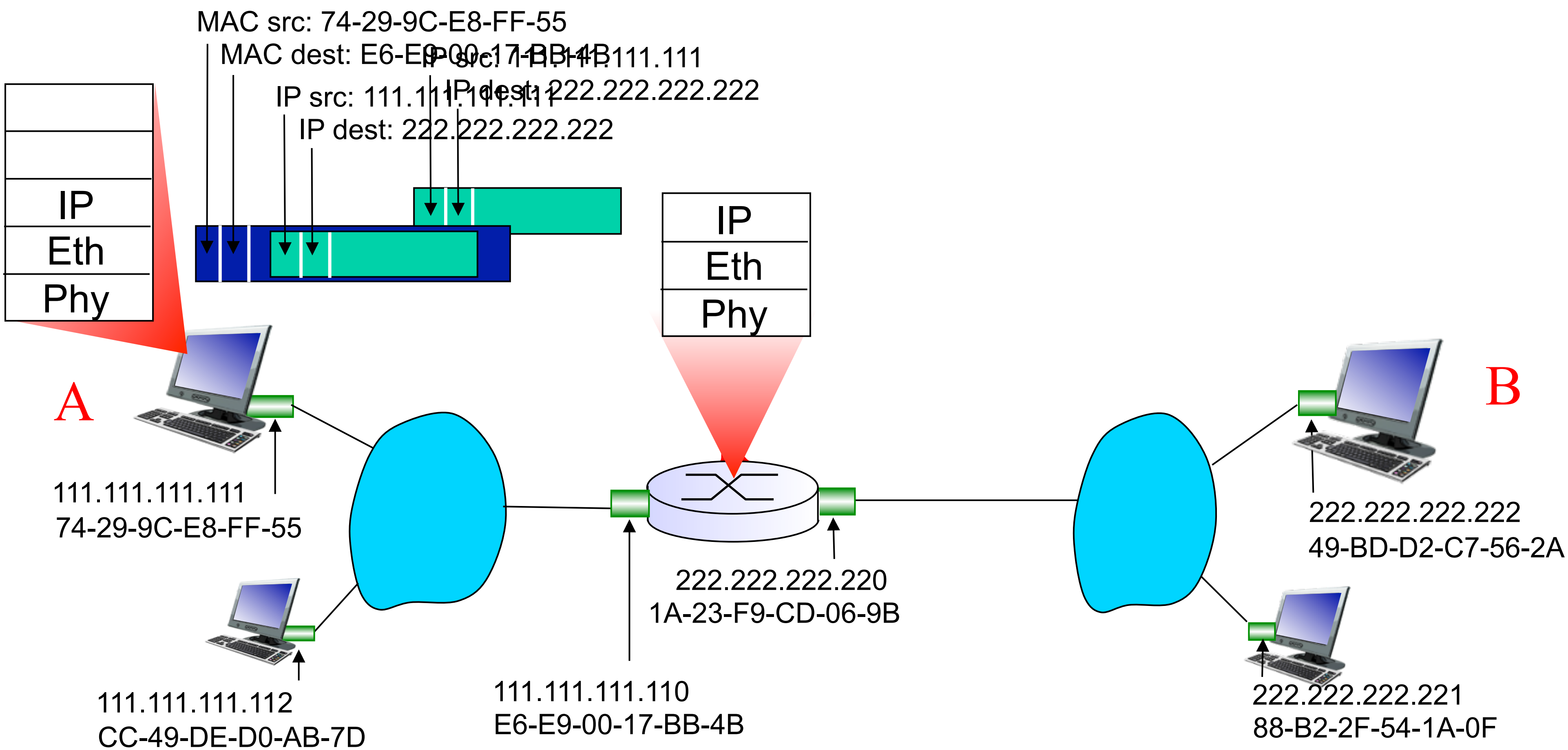
Addressing: routing to another LAN

- 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



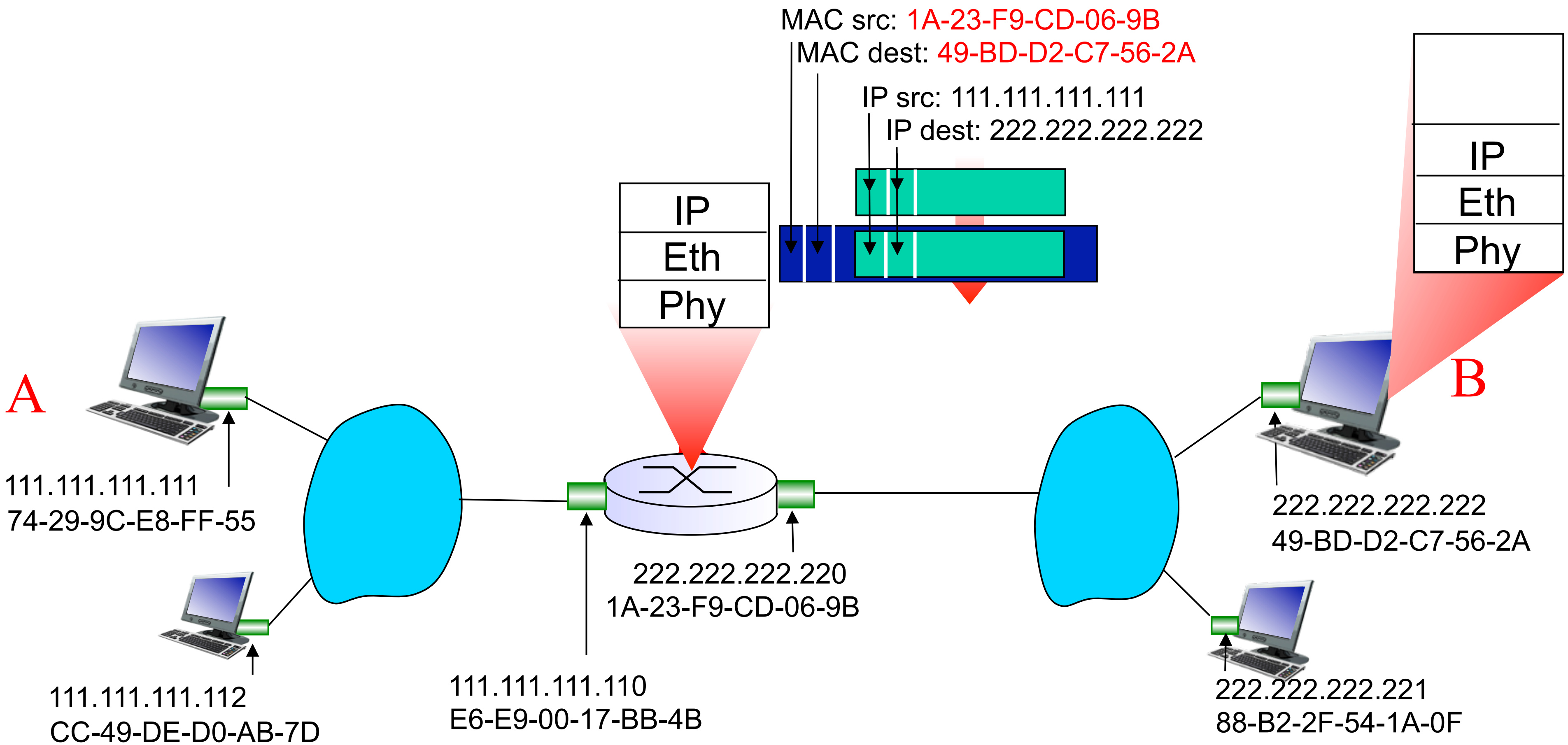
Addressing: routing to another LAN

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



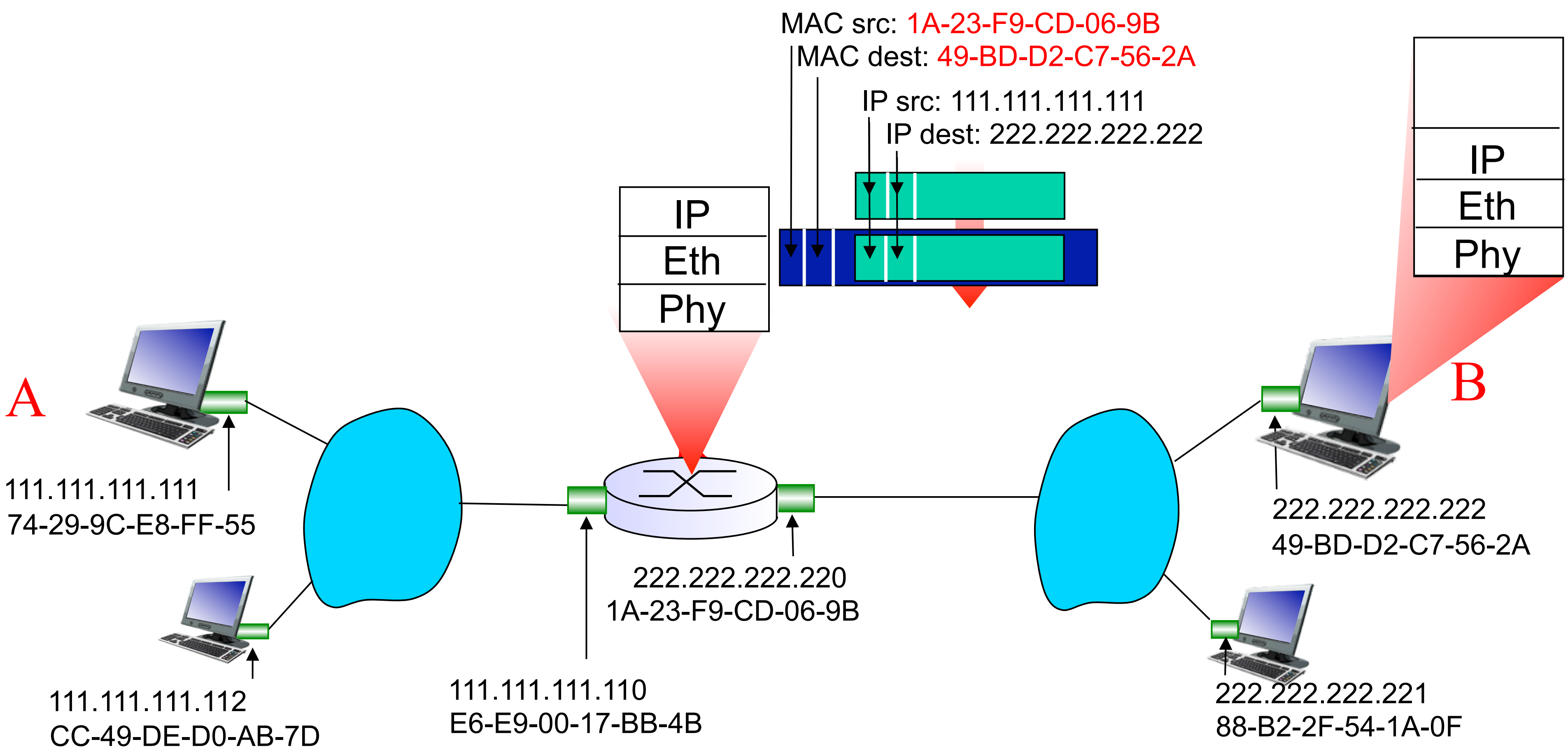
Addressing: routing to another LAN

- 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



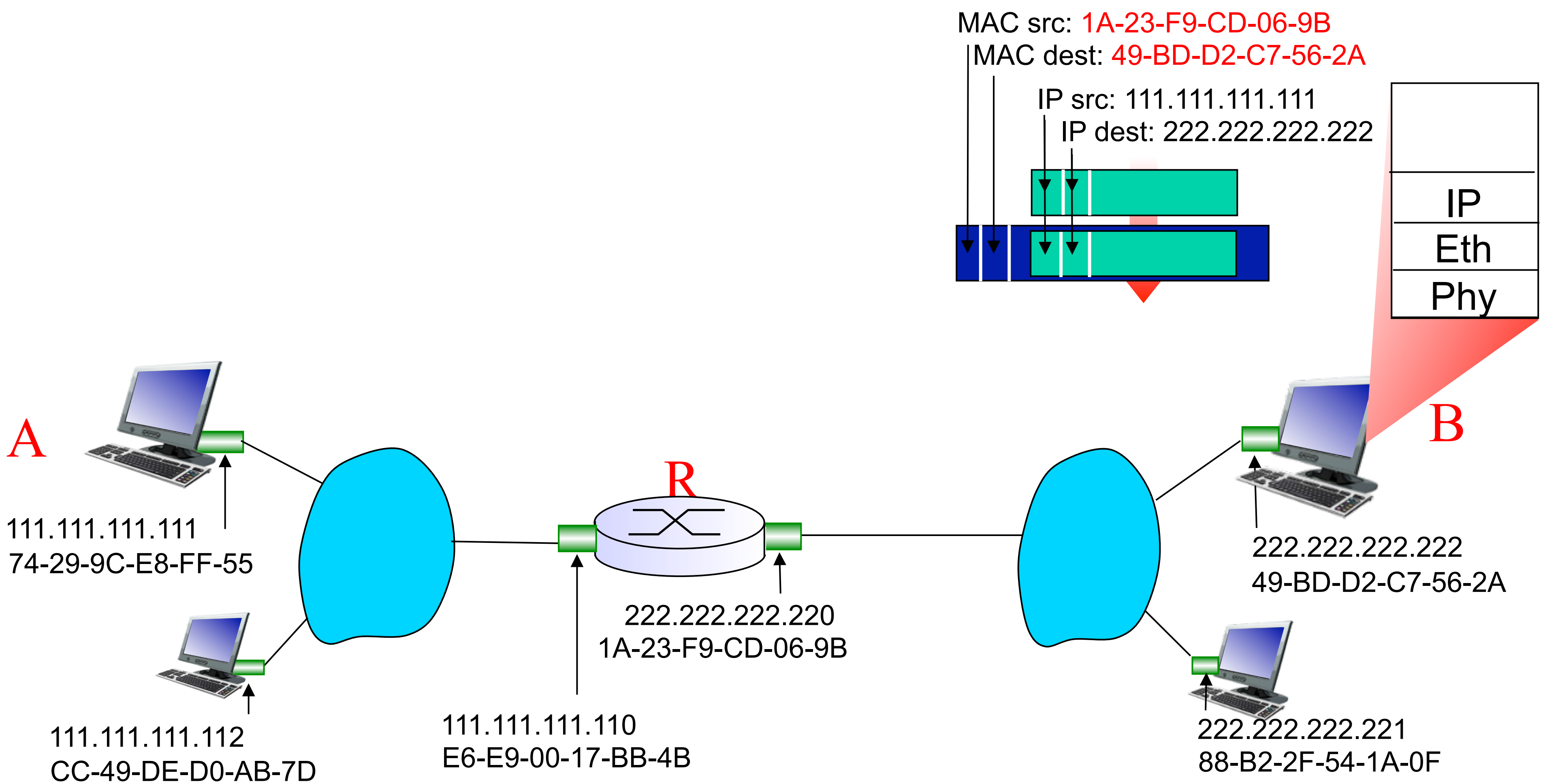
Addressing: routing to another LAN

- 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



Switches and bridges

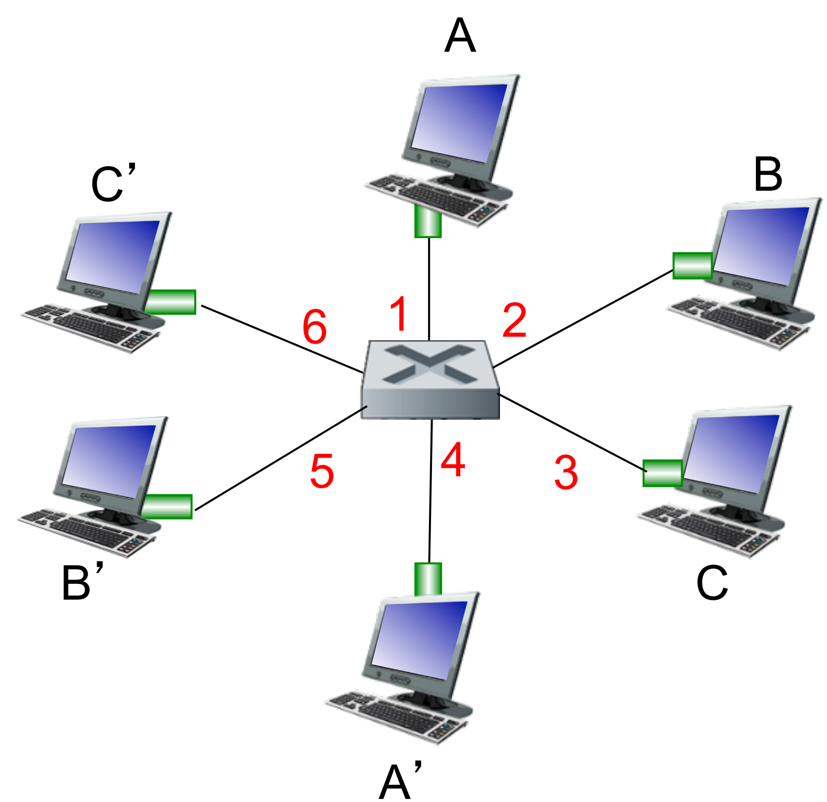
- Switches: Link layer devices that have multiple input/output ports (interfaces) where LAN segments/hosts/other switches can plug in
- Switches are also called “bridges”. We will mostly use “switches” to avoid confusion, but realize that it means the same as “bridges”.
- Often connected using a star topology
- This forms the basis for scalable connectivity as nodes can be added without impacting efficiency
- Key Functionality:
    - Forwarding, Learning, Spanning Tree construction

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 contention resolution 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

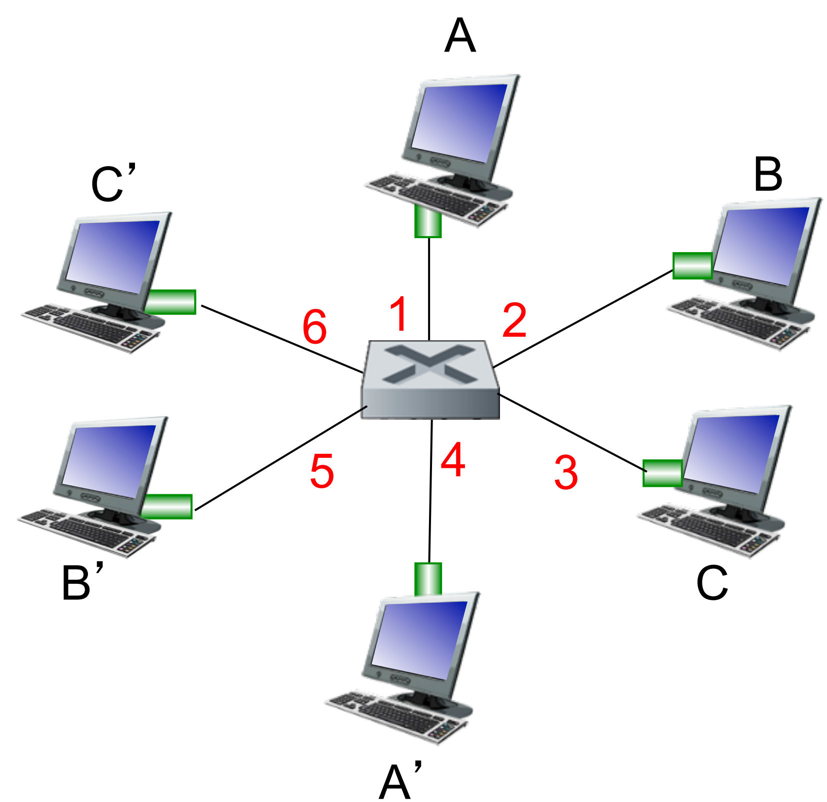


switch with six interfaces (1,2,3,4,5,6)

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)
  - looks like a routing table!



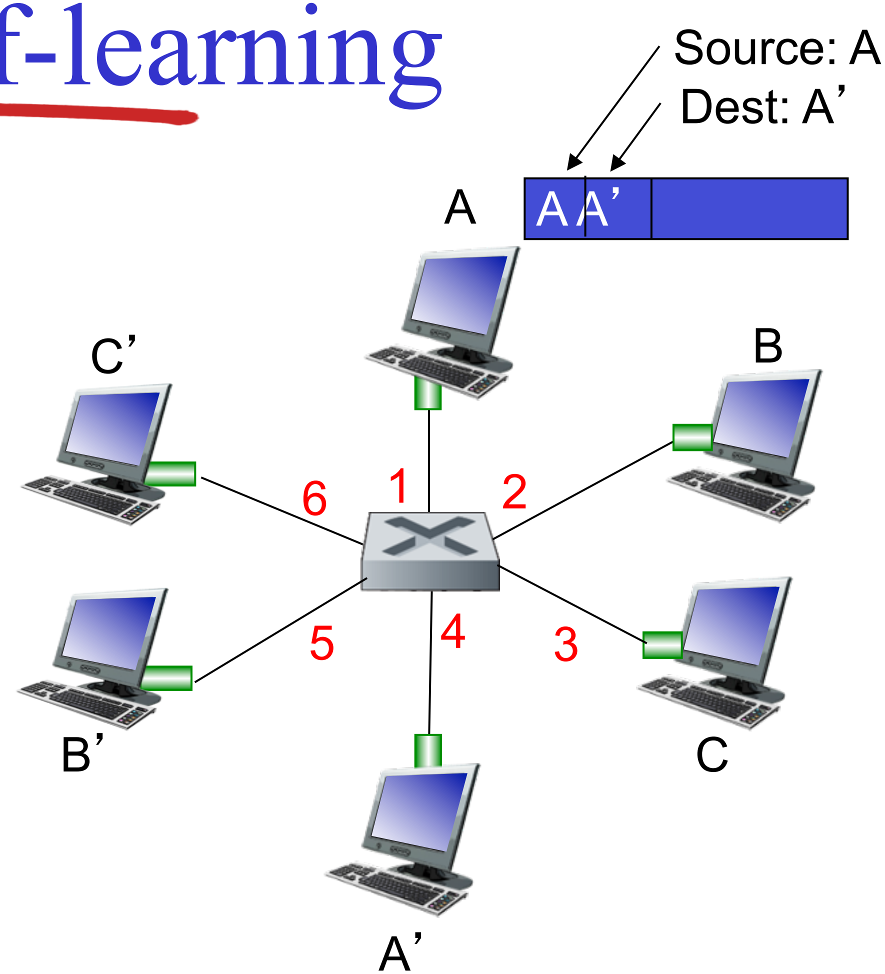
switch with six interfaces (1,2,3,4,5,6)

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

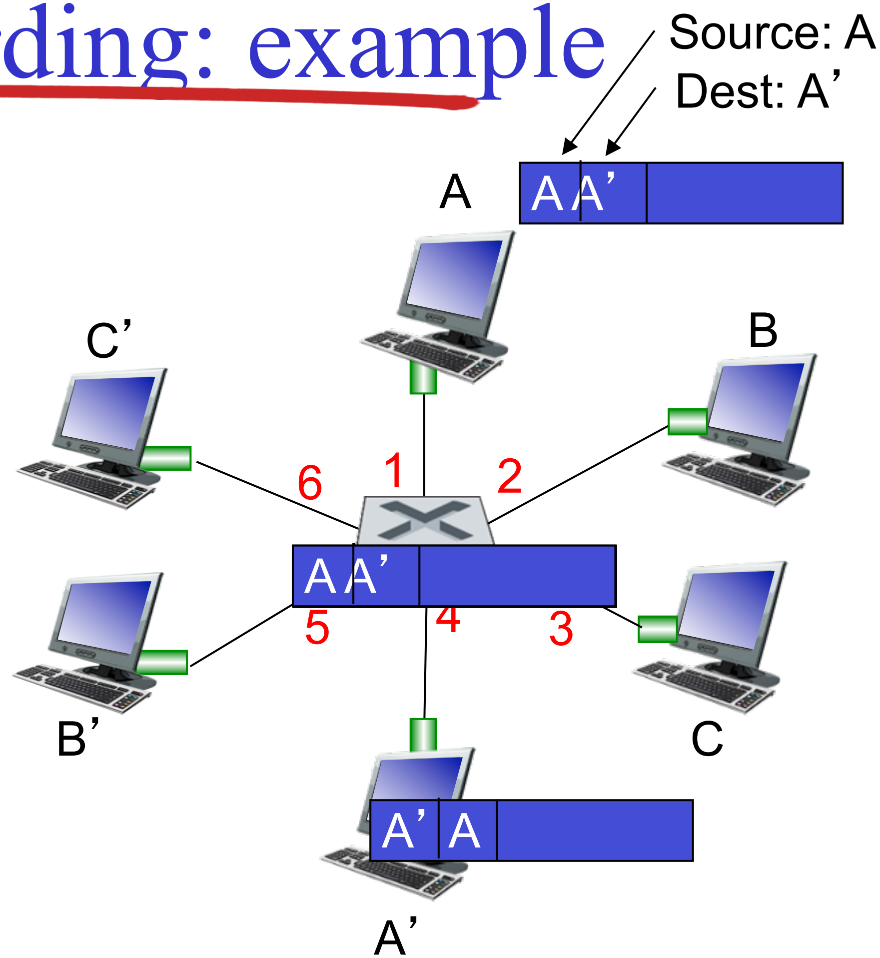
when frame received at switch:

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', locaton unknown: *flood*

❖ destination A location known: selectively send on just one link

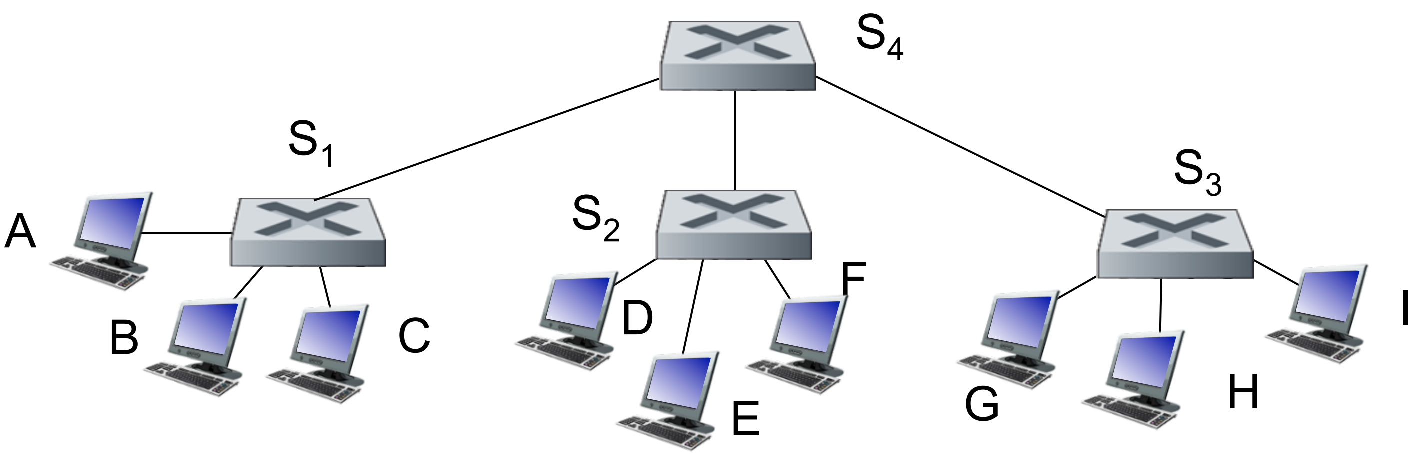


MAC addr	interface	TTL
A	*1*	*60*
A'	*4*	*60*

switch table (initially empty)

Interconnecting switches

- switches can be connected together

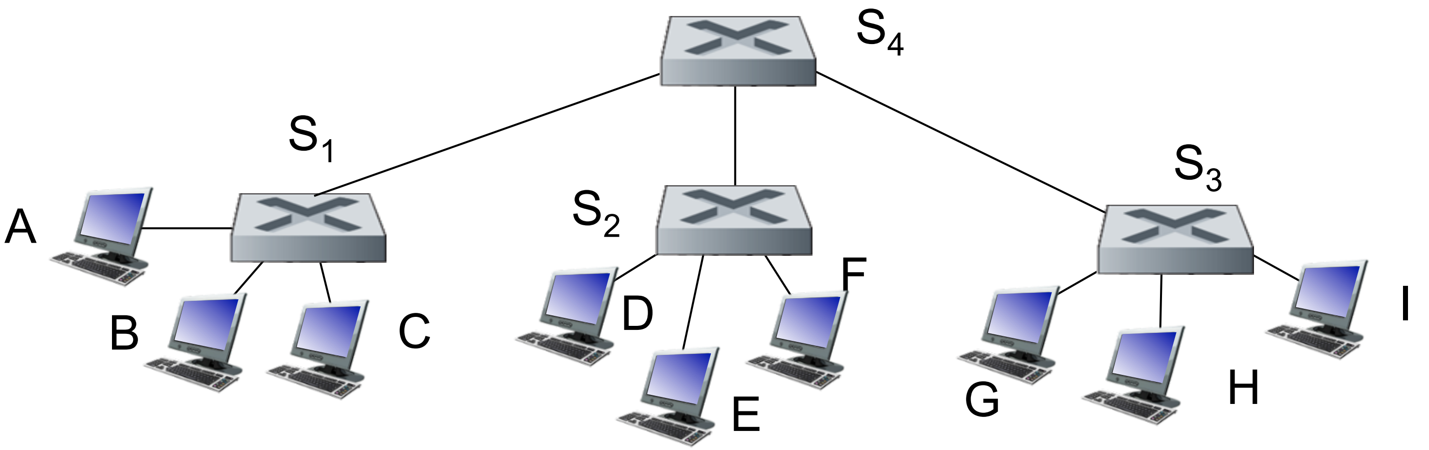


Q: sending from A to G - how does S_1 know to forward frame destined to F via S_4 and S_3?

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

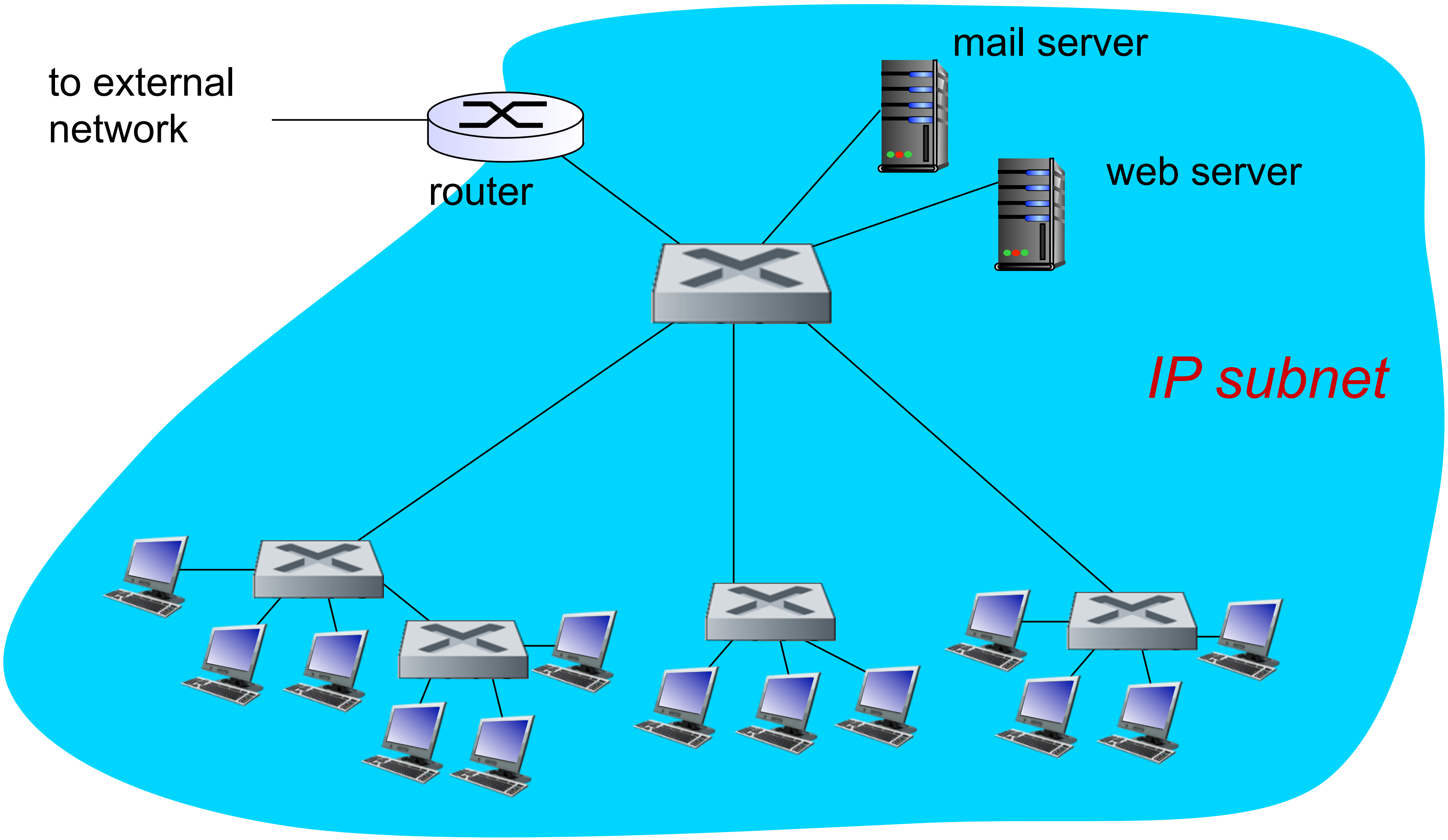
Self-learning multi-switch example

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

Institutional network



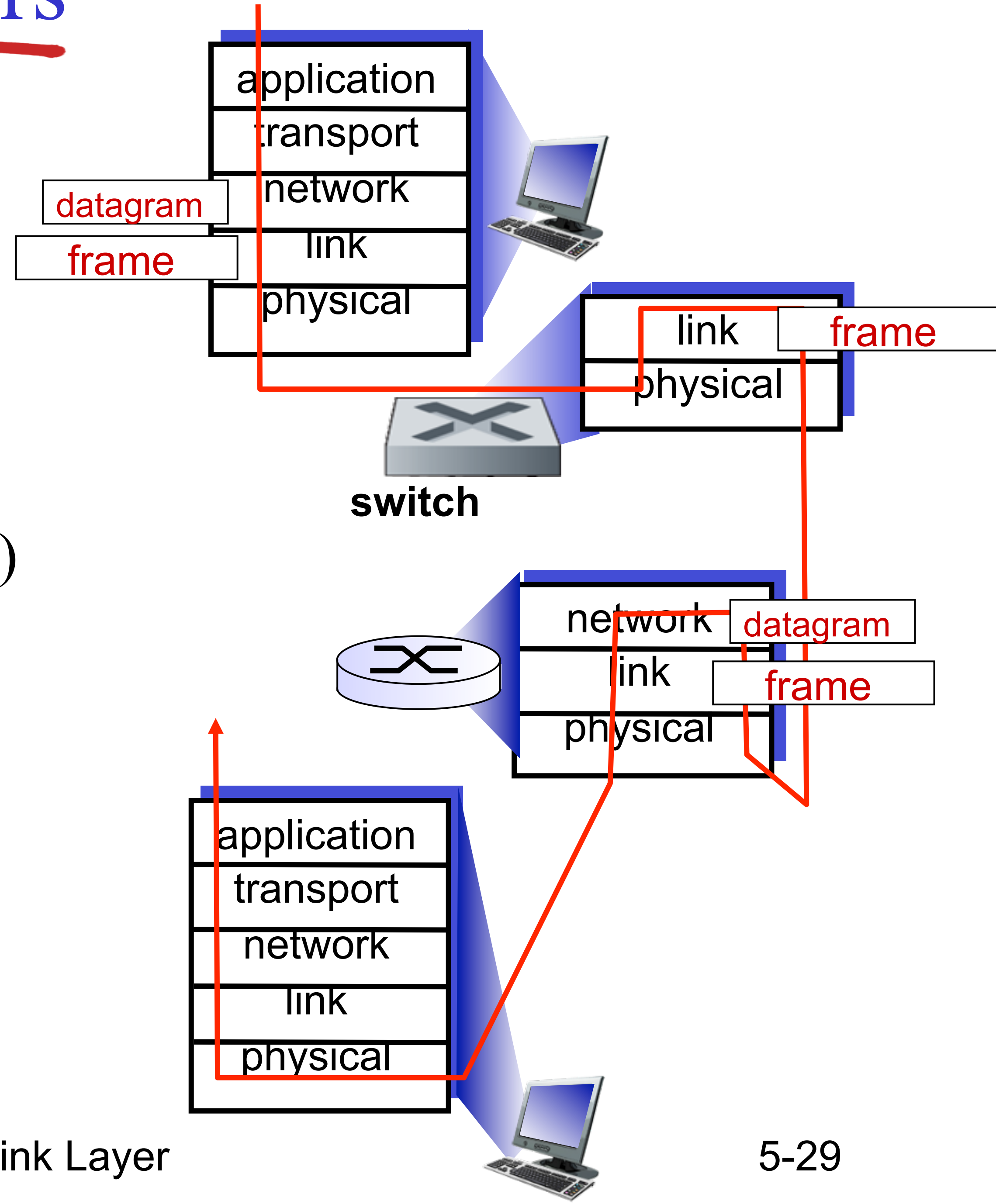
Switches vs. routers

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

- Packet switching:
  - Frame comes in on interfaces
  - Switch looks at destination LL address
  - Looks address up in a table and forwards it along
  - Runs contention resolution method if needed
  - Size of table in worst case == number of nodes
  - If there is no entry in the table, then broadcast to all ports

Switches

- Learning
  - Problem: How to build this table
  - Constraint: Hosts can move or go offline
  - Option: Manual configuration is painful and has no mobility support
- So switches use learning
  - Keep track of source addresses of packets (S) arriving on interface (I)
  - If packets arrive with destination address S then we know which interface to use
  - Time out – accounts for mobility

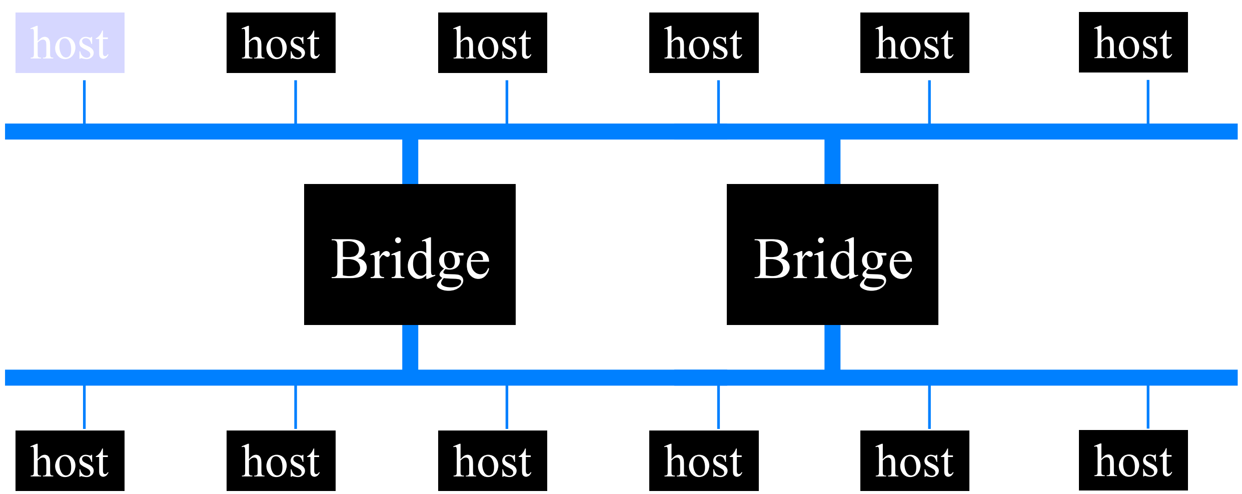
Spanning Tree

How to handle forwarding in complex Layer 2 topologies?

Each LAN segment can have many bridges

- More complex topologies can provide redundancy.
  - But can also create loops.
    - E.g. What happens when there is no table entry?
  - Multiple copies of data

→ Could crash the network → has happened often!



What is a Spanning Tree?

- Reduce our topology graph to a tree:
  - Make sure there are no loops in the topology
  - All LAN segments are still connected to the LAN and can receive messages
- Main idea: Bridges choose the ports over which they have to forward frames.

Spanning Tree Protocol Overview

Embed a tree that provides a single unique default path to each destination:

Bridges designate ports over which they will or will not forward frames

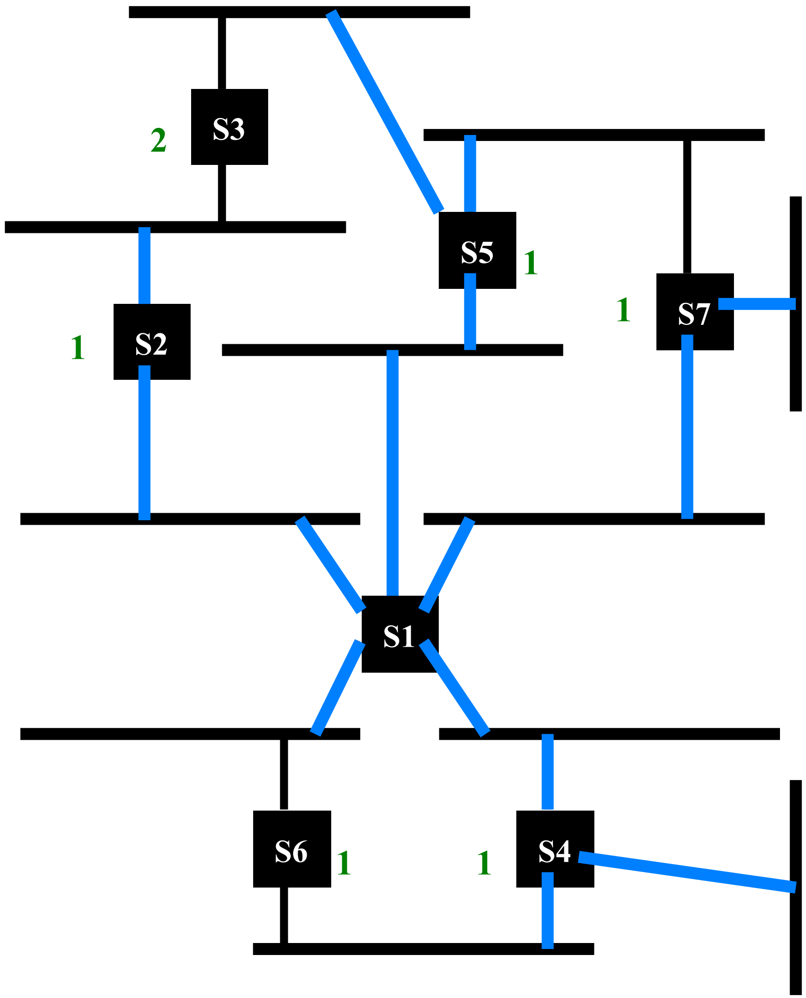
By removing ports, extended LAN is reduced to a tree

Addresses the crashing problem; but tree is not resilient

When switch/link fails, rerun protocol to converge to new tree

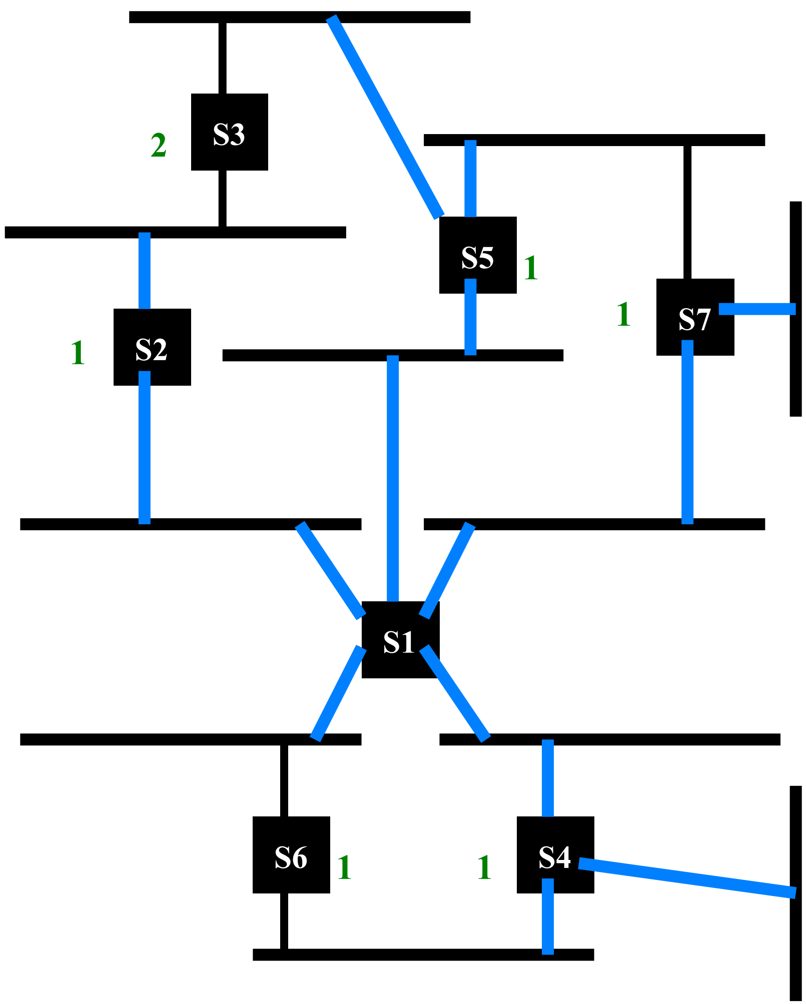
Spanning Tree Algorithm

- Root of the spanning tree is elected first → the bridge with the lowest identifier.
  - All ports are part of tree
- Bridges designate ports on which they will/not forward
- Each bridge finds shortest path to the root.
  - Remembers port that is on the shortest path
  - Used to forward packets
- Select for each LAN a designated bridge that will forward frames to root
  - Has the shortest path to the root.
  - Identifier as tie-breaker



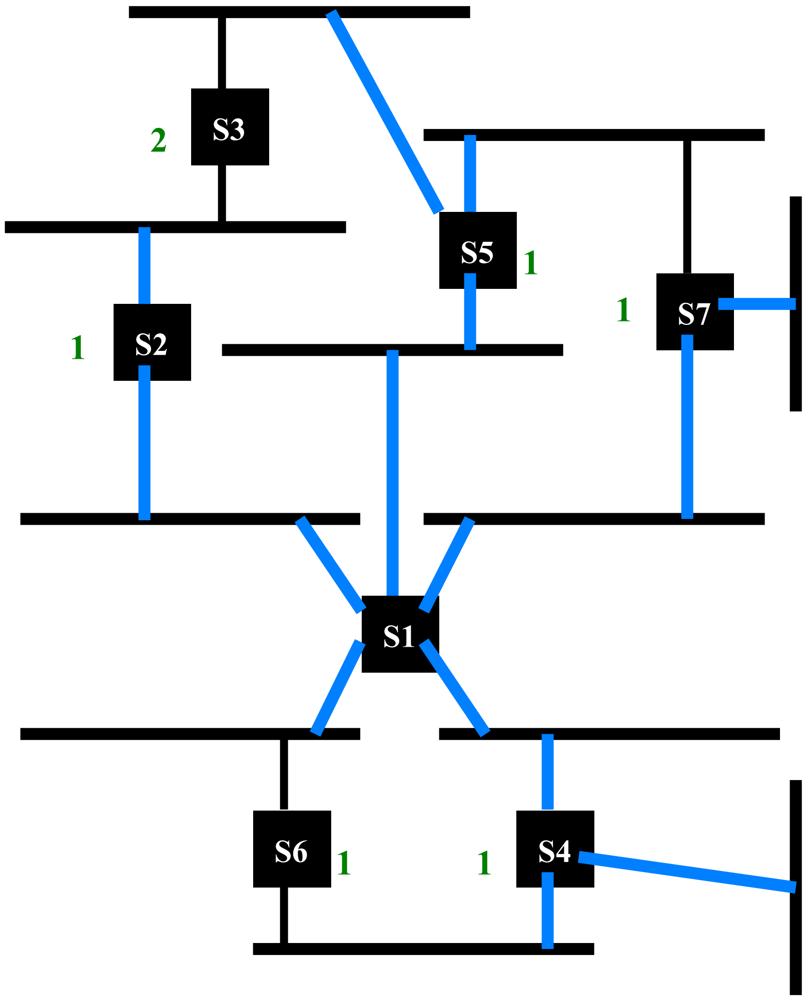
Spanning Tree Algorithm

- Each node sends configuration message to all neighbors.
  - Identifier of the sender
  - Id of the presumed root
  - Distance to the presumed root
- Initially each bridge thinks it is the root.
  - B5 sends (B5, B5, 0)
- When B receive a message, it decide whether the solution is better than their local solution.
  - A root with a lower identifier?
  - Same root but lower distance?
  - Same root, distance but sender has lower identifier?
- Message from bridge with smaller root ID
  - Not root; stop generating config messages, but can forward
- Message from bridge closer to root
  - Not designated bridge; stop sending any config messages on the port
  - Block port



Spanning Tree Algorithm

- Each bridge B can now select which of its ports make up the spanning tree:
  - B's root port
  - All ports for which B is the designated bridge on the LAN
- States for ports on bridges
  - *Forward state* or *blocked state*, depending on whether the port is part of the spanning tree
- Root periodically sends configuration messages and bridges forward them over LANs they are responsible for
- Any bridge failure => Start over



Spanning Tree Algorithm Example

- –B3 receives (B2,B2,0)
- –Since 2<3 B3 accepts B2 as a root
- –B3 adds one to the distance advertised by B2(0) and thus sends (B3,B2,1) toward B5
- –Meanwhile B2 accepts B1 as the root and sends (B2,B1,1)
- –B5 accepts B1 as the root and sends (B5,B1,1)
- –B3 accepts B1as the root and figures that B1 and B2 are closer to the root. So stops forwarding on both interfaces.