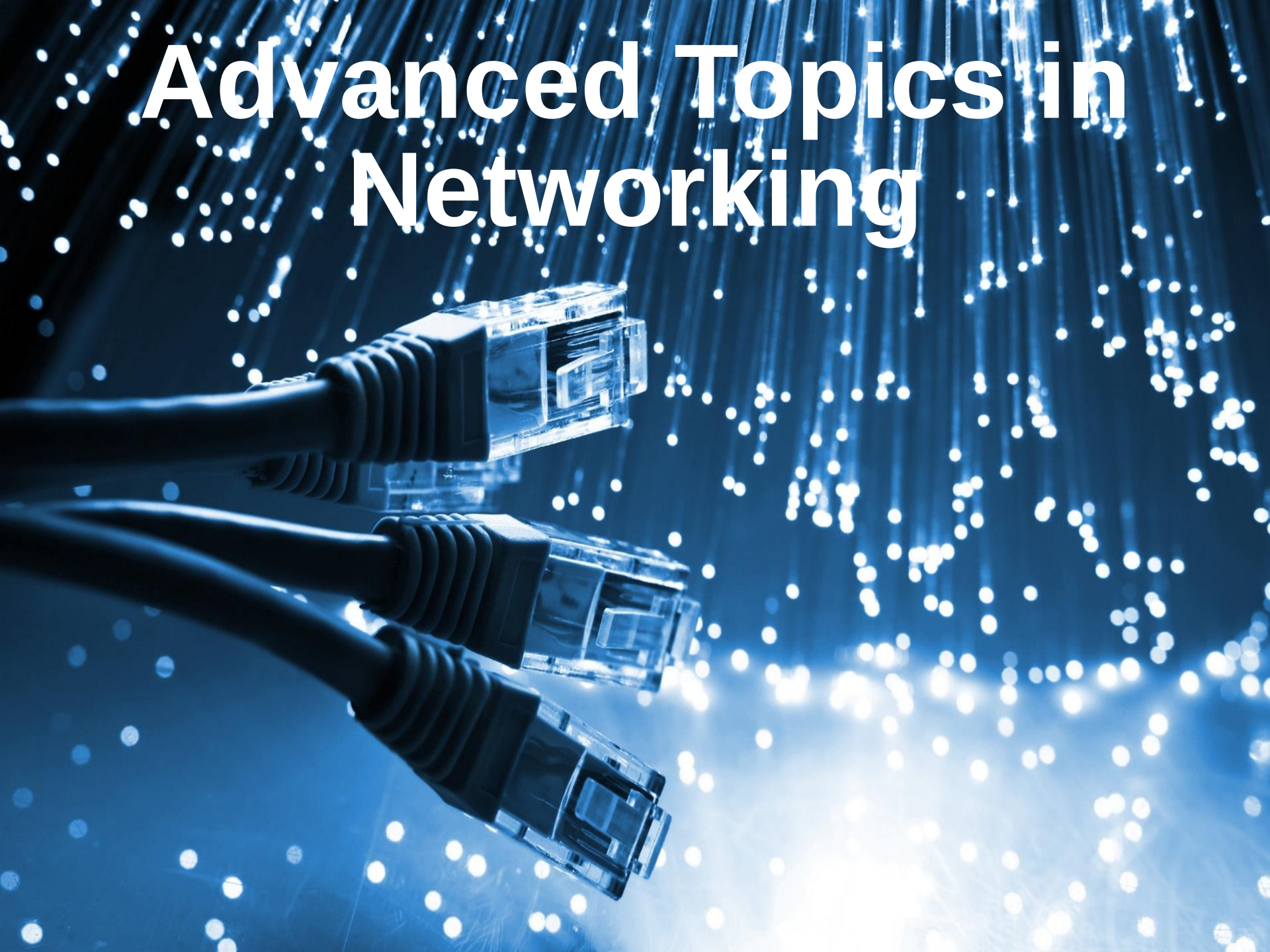
# Advanced Topics in Networking



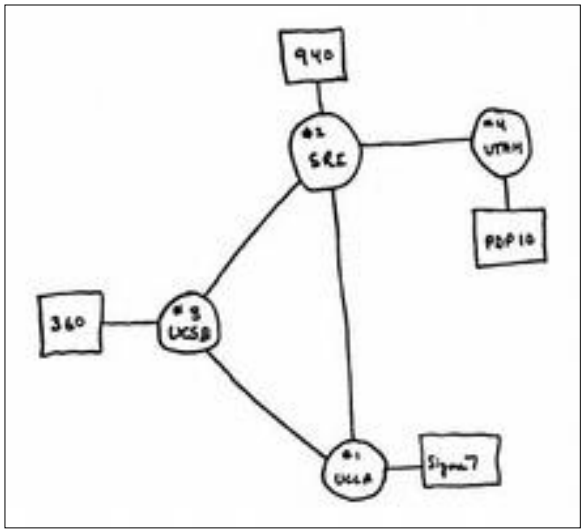
# Overview

- Part I: Become a master!
  - You need to master a topic before you can criticize it: Recap computer networking
- Part II: Criticize it!
  - Advanced topics in networking

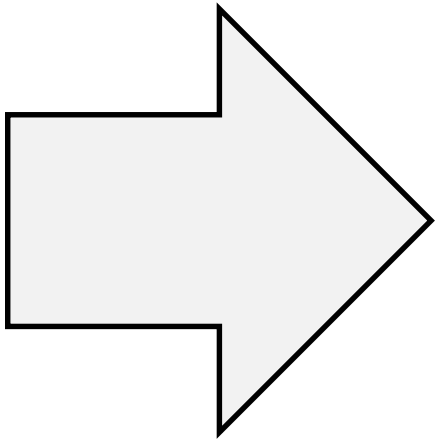
# Part I: Become A Master

# Internet: A Big Success

- Hardly any outages over the last decades: despite huge increase in scale and shift in applications
- Although Internet protocols have hardly changed!




*In the past:*
*connectivity (email, etc.)*

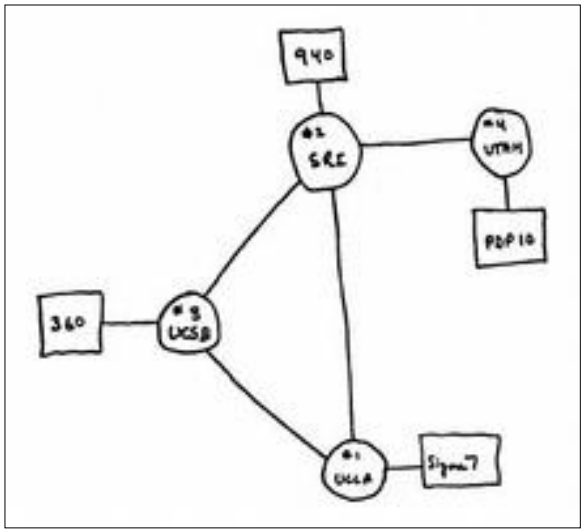


*Now:*
*live streaming, e-voting, cars…*

# Internet: A Big Success

- Hardly any
  huge increa
- Although Internet

What are the design principles that made this possible?

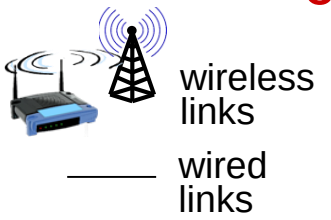



*In the past:*
*connectivity (email, etc.)*

*Now:*
*live streaming, e-voting, cars…*

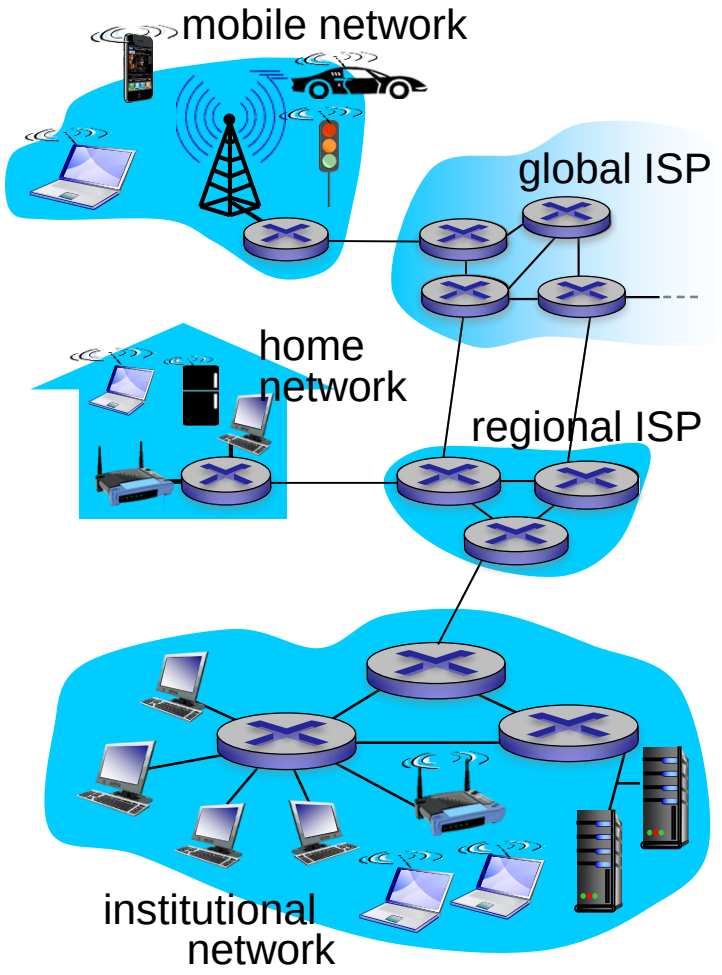
# Communication Networks & Internet


billions of computing devices connected through various networks

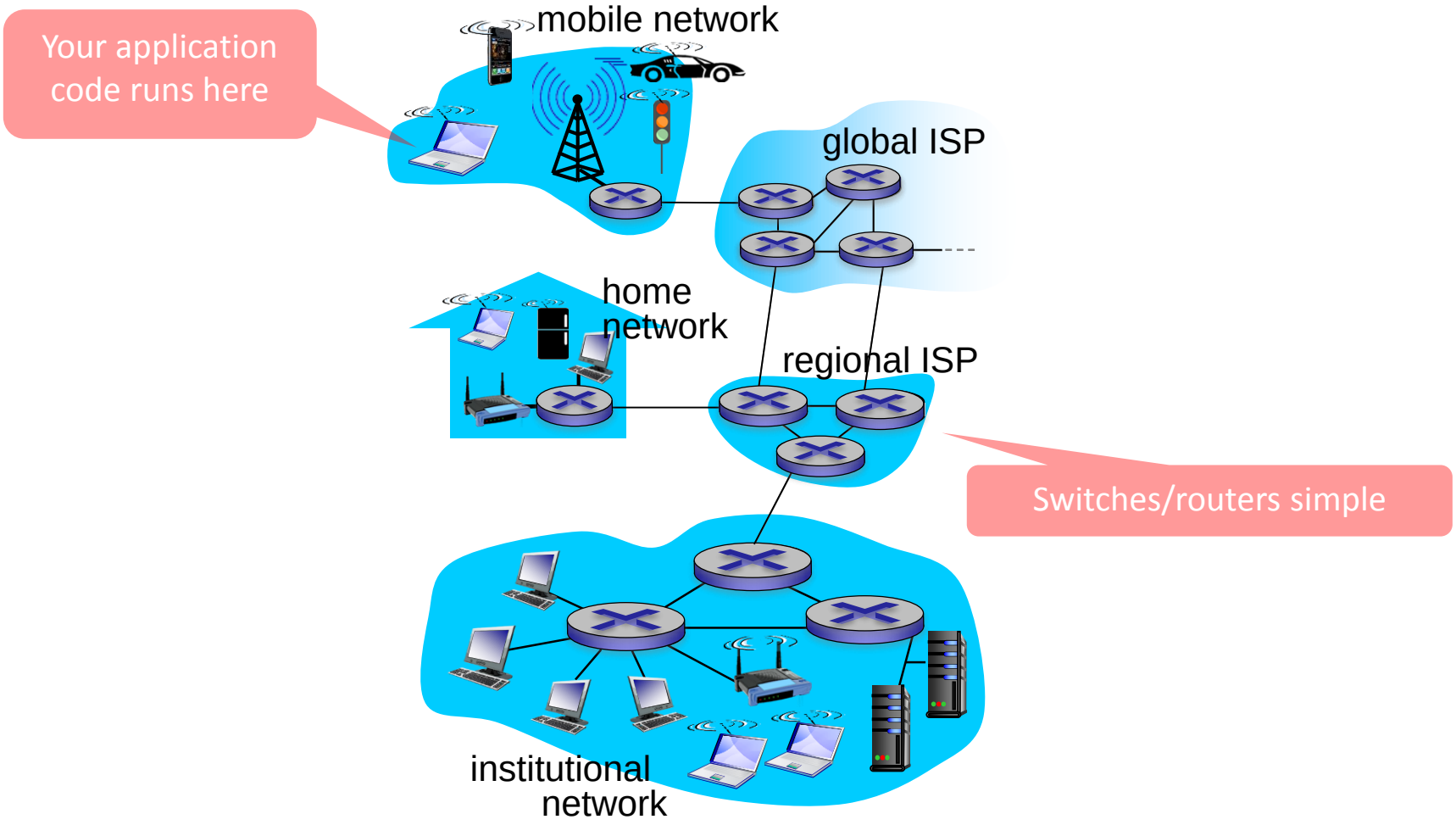



communication links e.g.

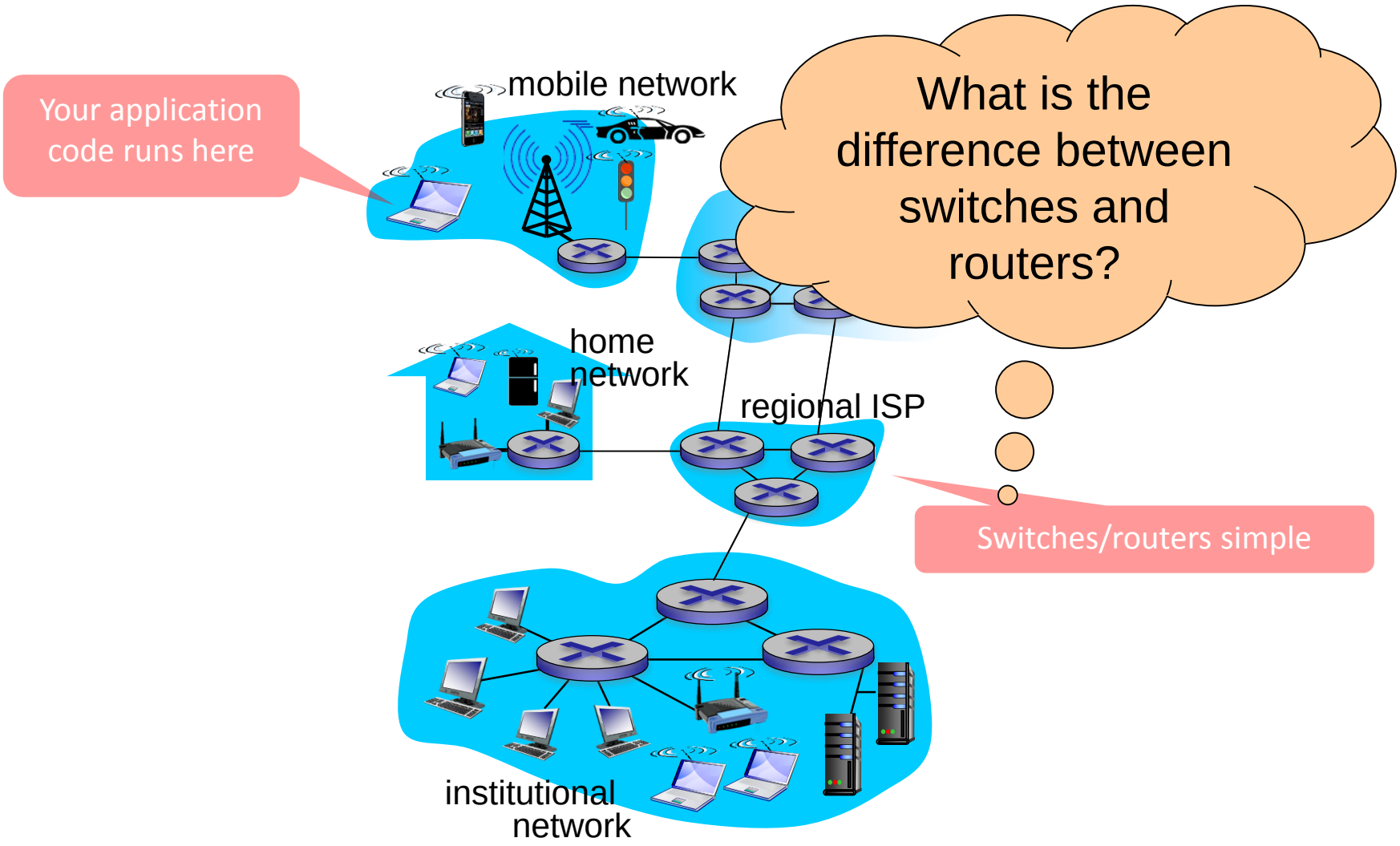
- wired (fiber) and wireless (e.g., radio, satellite)



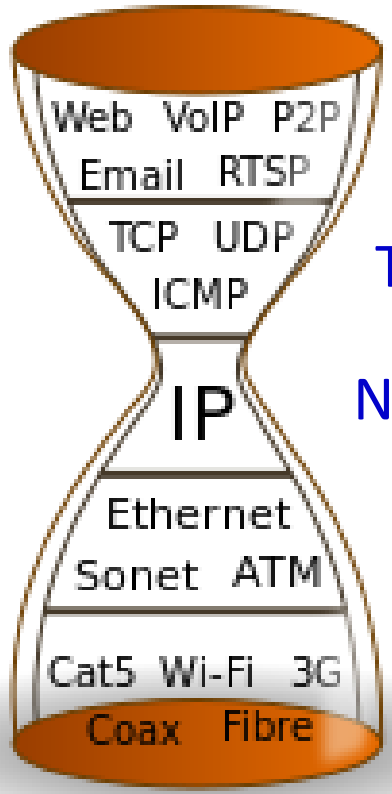
# Design Principle 1: Complexity at Edge



# Design Principle 1: Complexity at Edge



# Design Principle 2: Layering




Application Layer

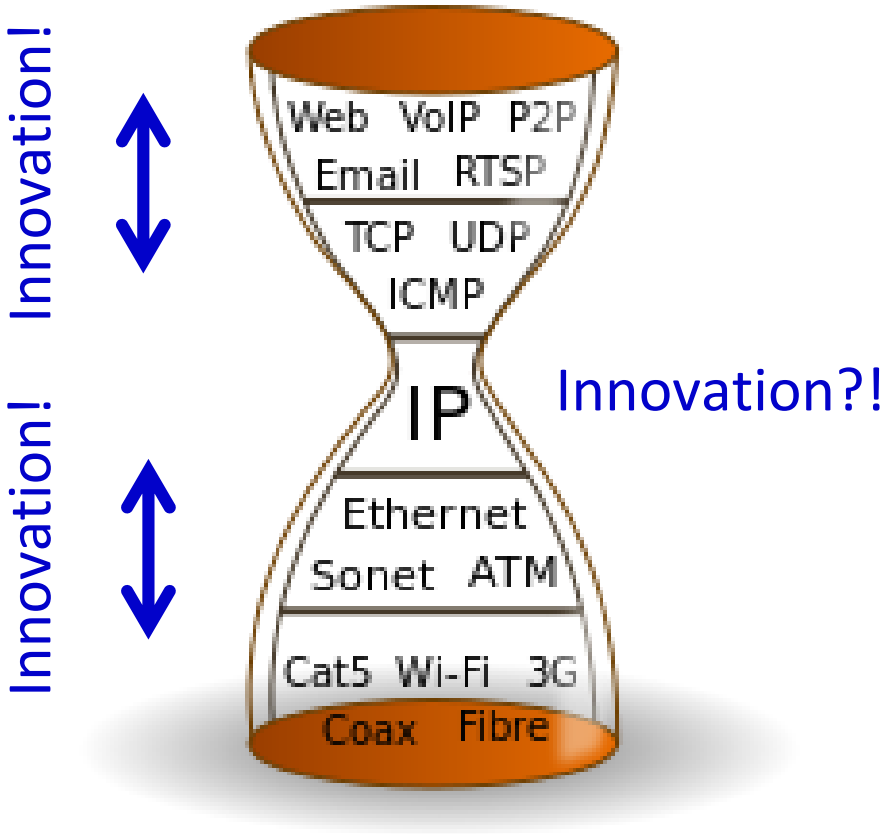
Transport Layer

Network Layer

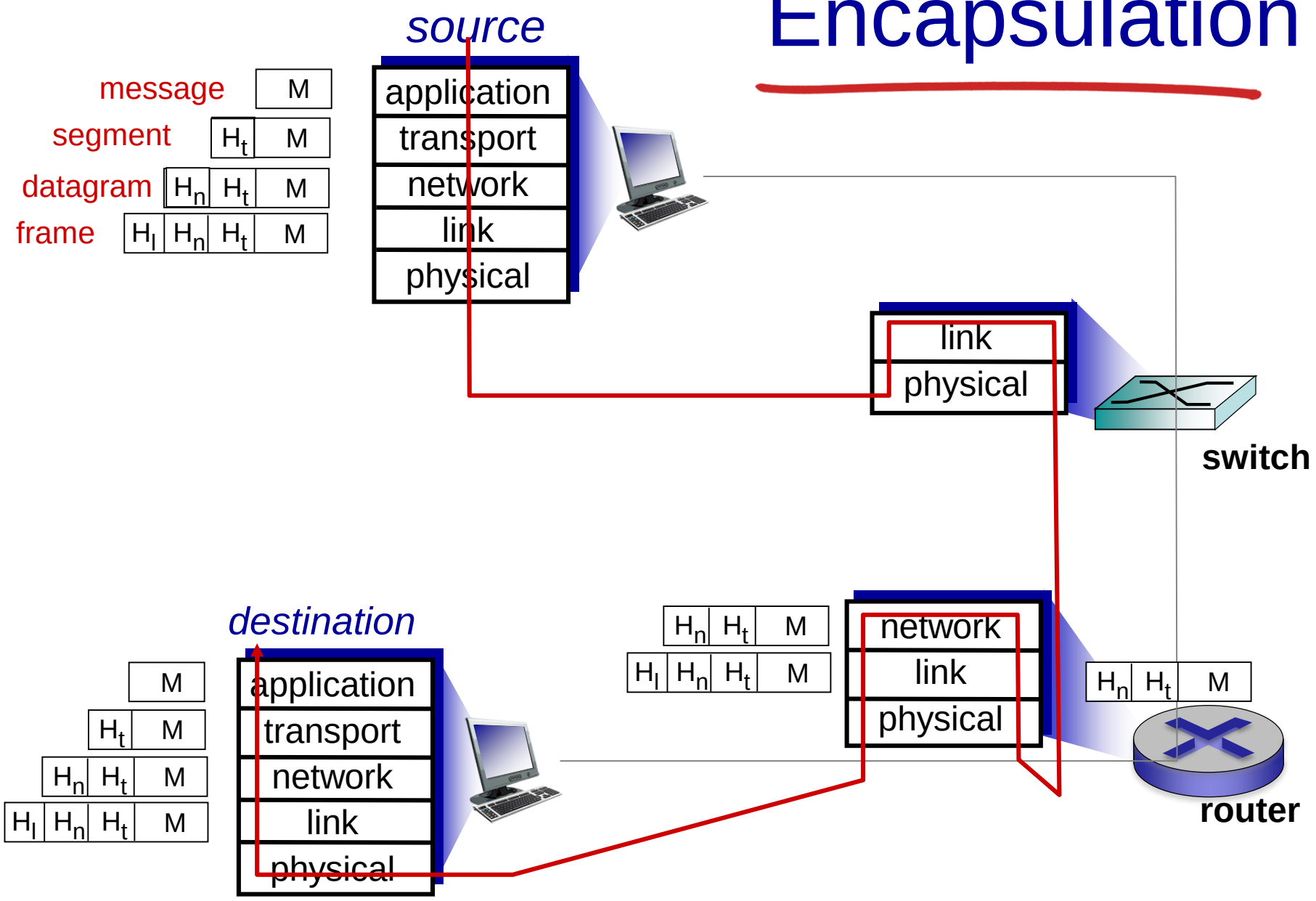
Link Layer

Physical Layer

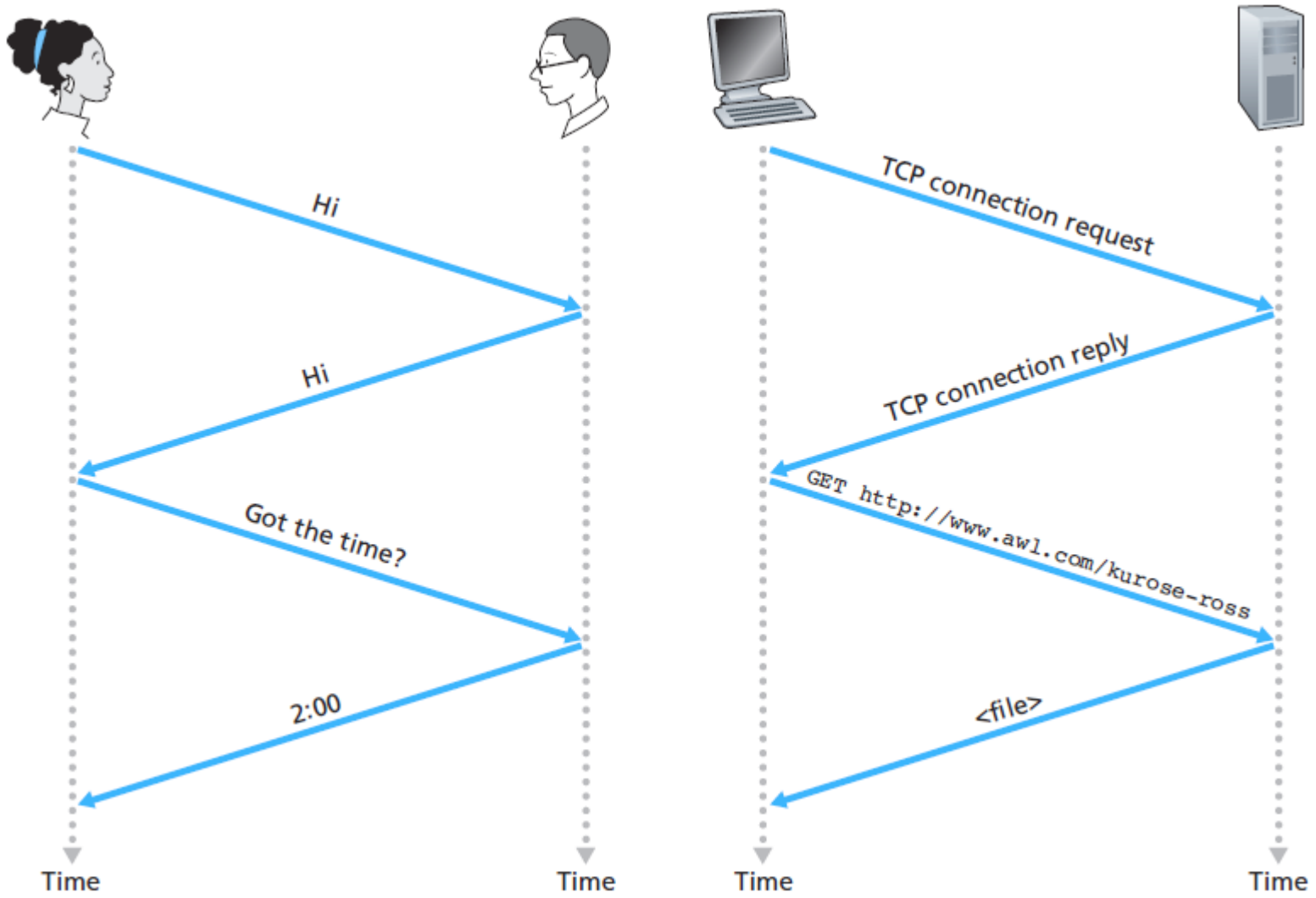
# Design Principle 2: Layering



# Encapsulation



# Design Principle 3: Standardized Protocols



# A Brief Overview of Layers

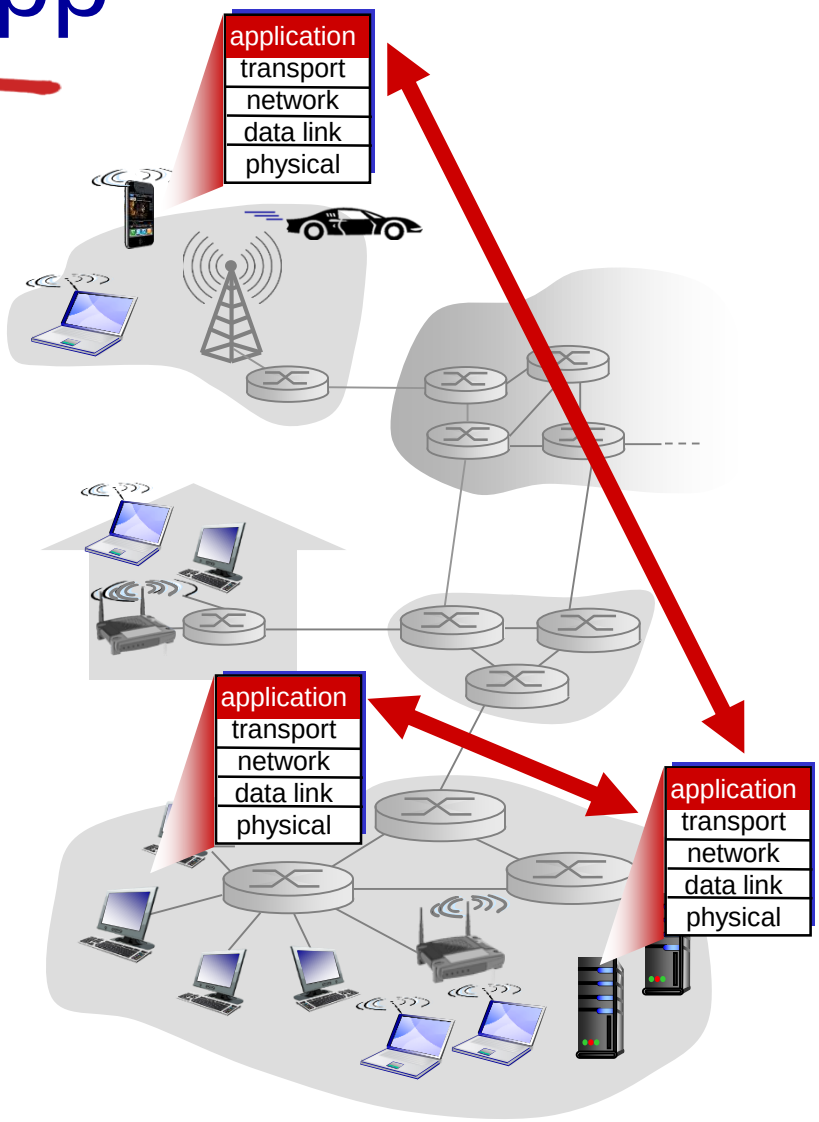
# Some network apps

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- …
- …

# Creating a network app

write programs that:

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software




no need to write software for network-core devices

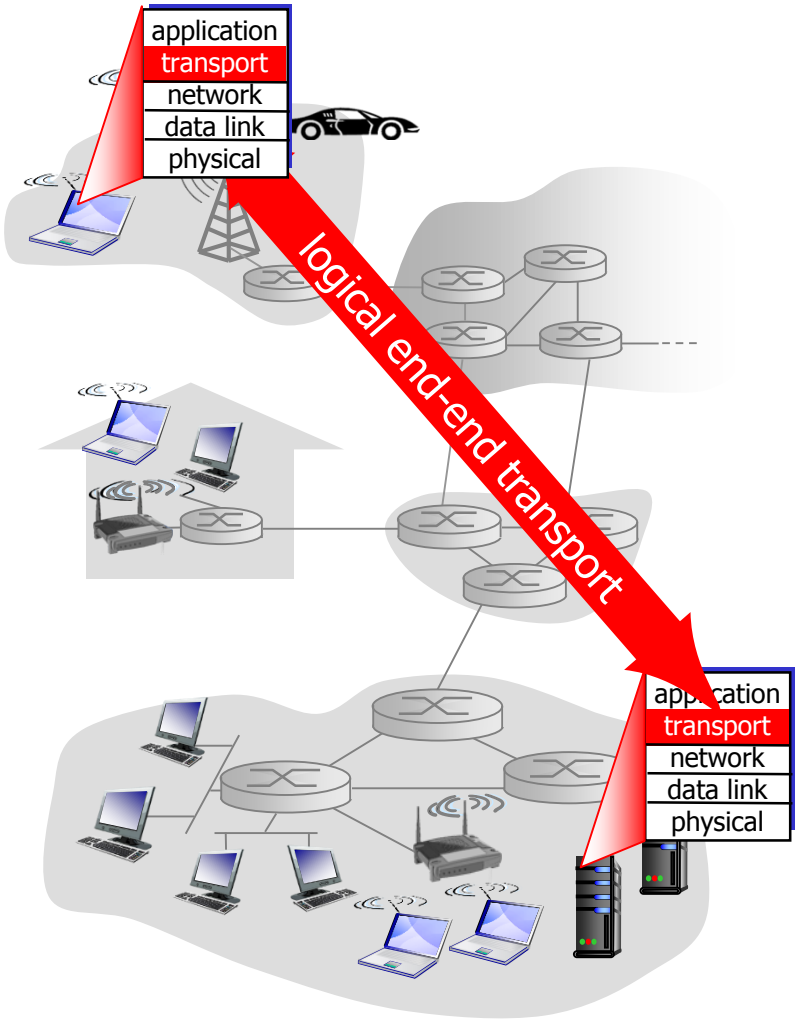
- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation

# Transport vs. network layer

- *transport layer:* logical communication between processes
  - relies on, enhances, network layer services
- *network layer:* logical communication between hosts

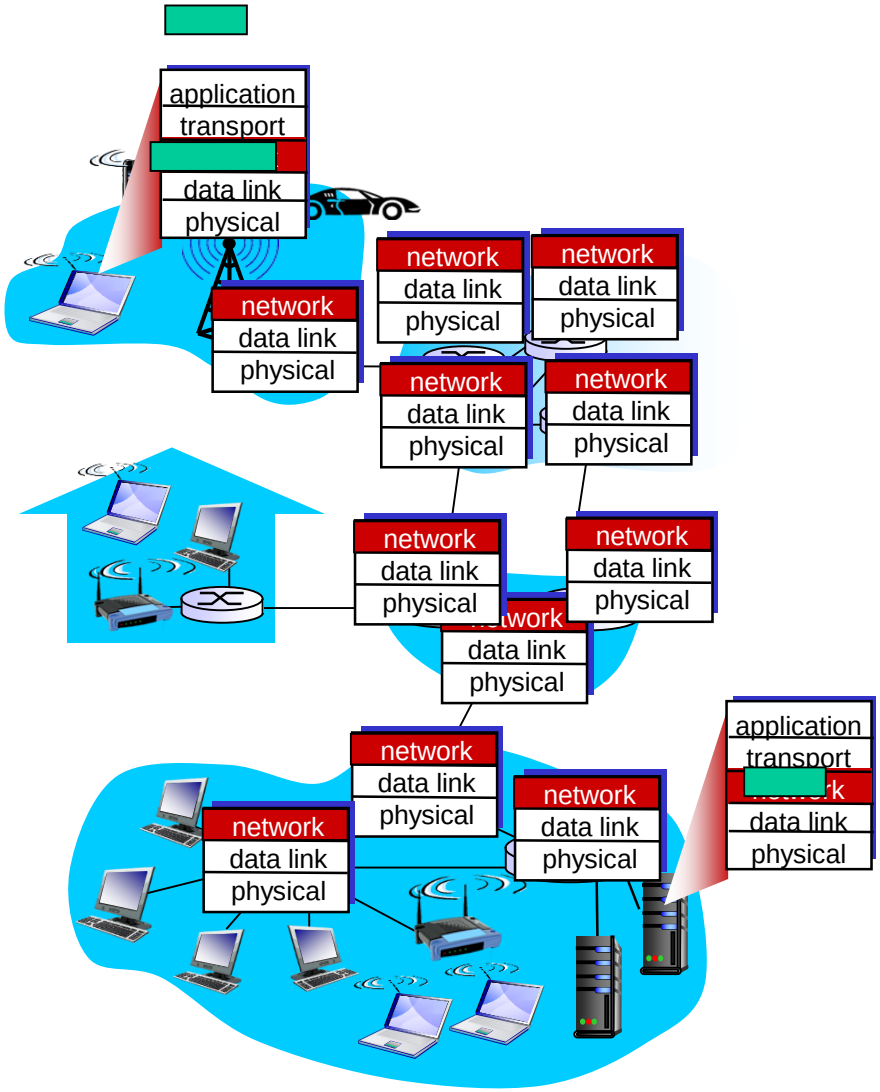
# Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side multiplexing: breaks app messages into *segments*, passes to network layer
  - rcv side demultiplexing : reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



# Network layer

- transport segment from sending to receiving host
- on sending side encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in *every* host, router
- router examines header fields in all IP datagrams passing through it



# Network-layer functions

*Two network-layer functions:*

- *forwarding:* move packets from router's input to appropriate router output
- *routing:* determine route taken by packets from source to destination

*data plane*

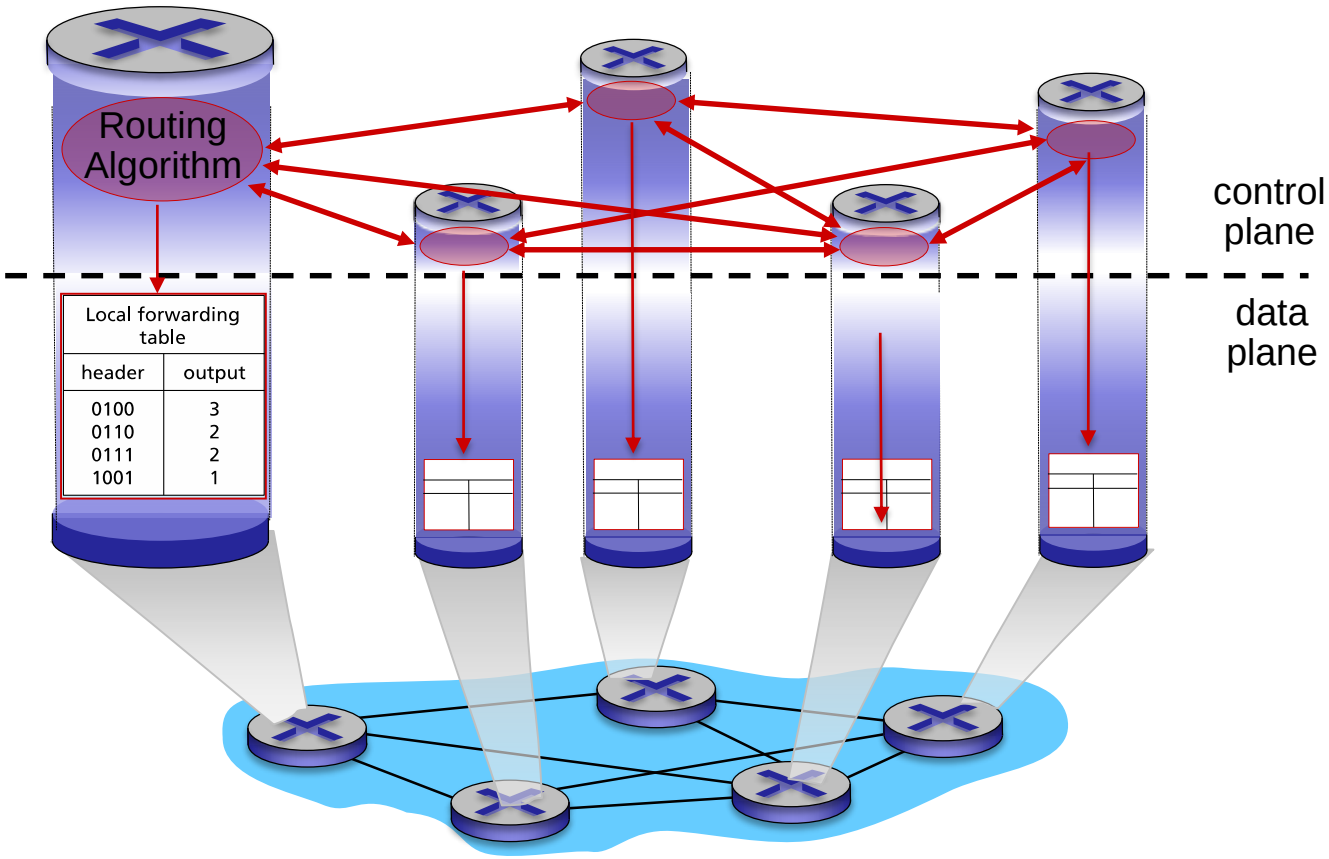
*control plane*

*Traditional network control plane:*

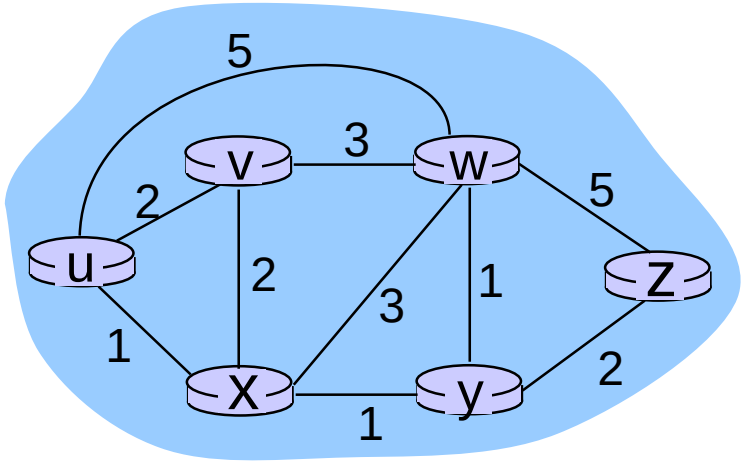
- per-router control

# Traditionally: per-router control plane

Individual (decentralized) routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables



# Routing Inside A Network

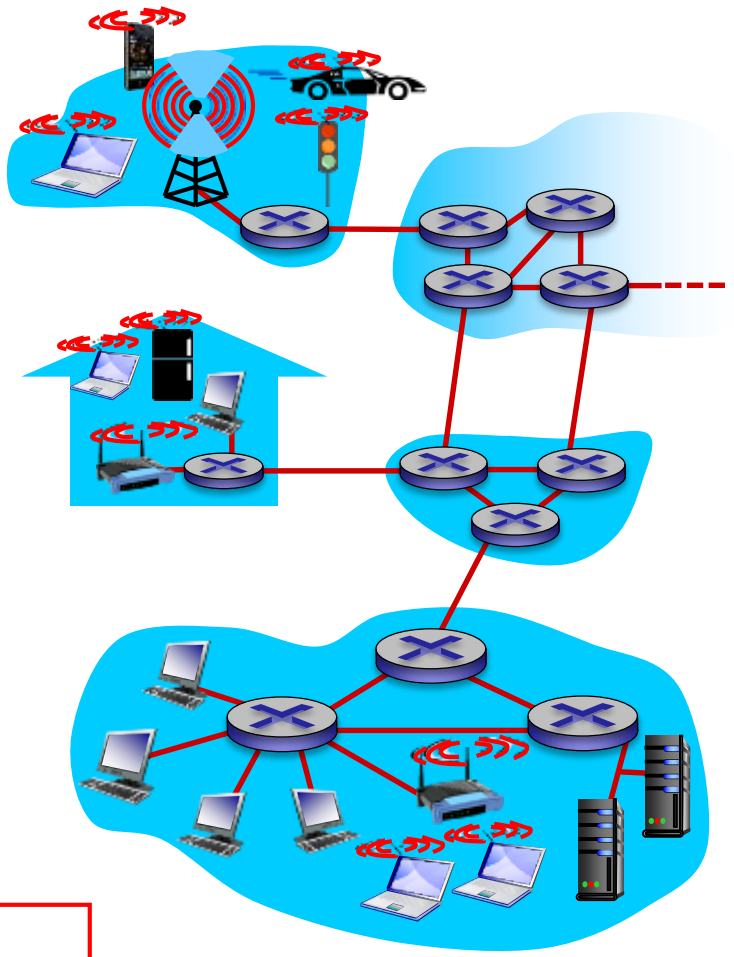



- *Graph algorithms for shortest paths:* Dijkstra's algorithm, distance vector, link state protocols, etc.

# Link layer

*terminology:*

- 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

A single broadcast domain!

- all layer-2 broadcast traffic (ARP, DHCP, unknown location of destination MAC address) must cross entire LAN
- security/privacy, efficiency issues

# A Less Brief Overview of Layers

# A Less Brief Overview of Layers

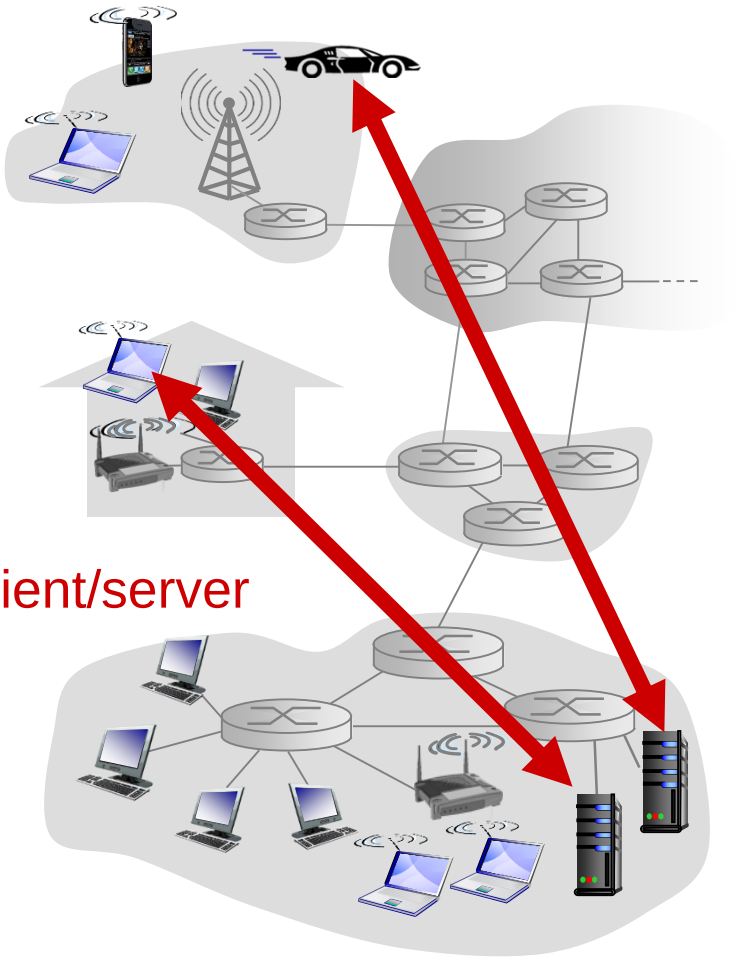
- **Application layer**
- Transport layer
- Network layer
- Link layer
- Example

# Application architectures

possible structure of applications:

- client-server
- peer-to-peer (P2P)

# Client-server architecture



client/server

server:

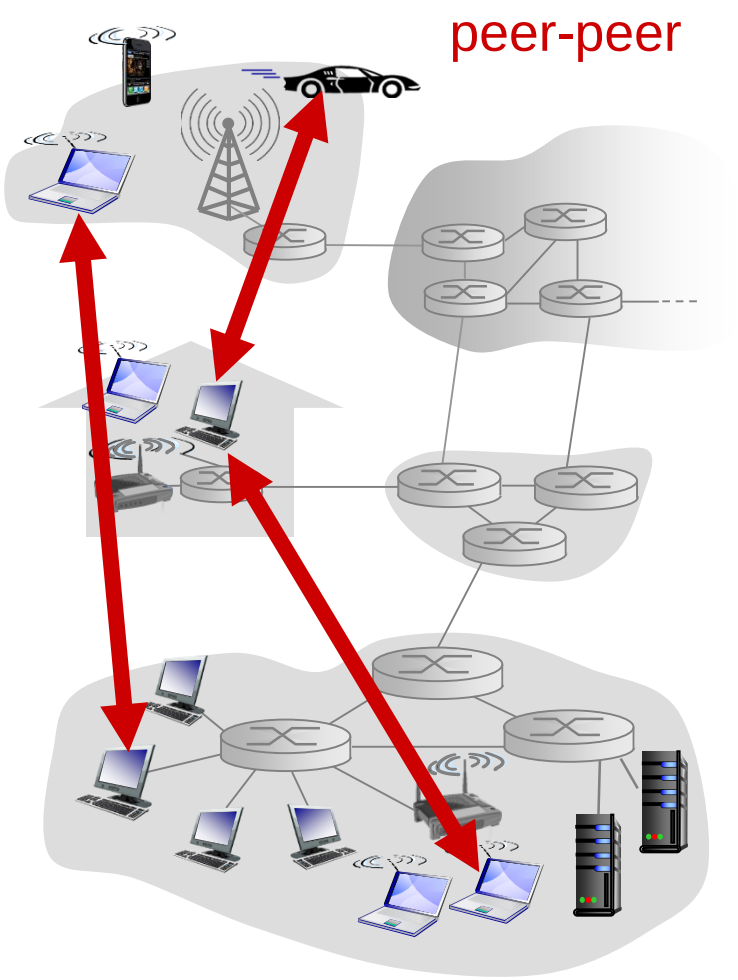
- always-on host
- permanent IP address
- data centers for scaling

clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

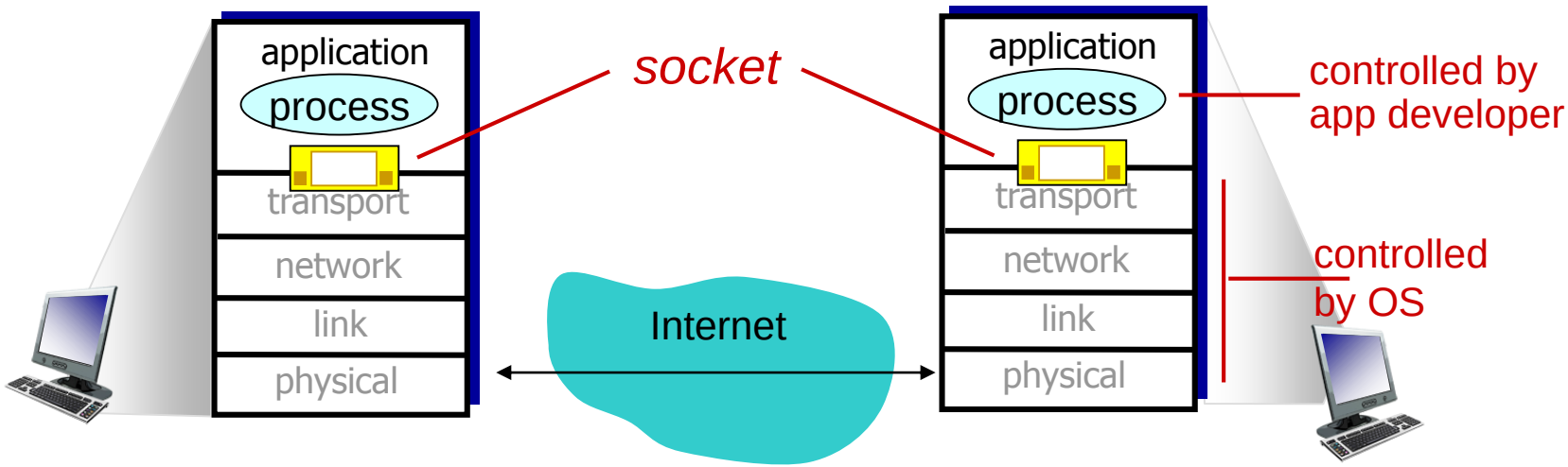
# P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



# Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



# Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* no, *many* processes can be running on same host

- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80

# What transport service does an app need?

## data integrity

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

## timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

## throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

## security

- encryption, data integrity, …

# Transport service requirements: common apps

| application | data loss | throughput | time sensitive |
|---|---|---|---|
| file transfer | no loss | elastic | no |
| e-mail | no loss | elastic | no |
| Web documents | no loss | elastic | no |
| real-time audio/video | loss-tolerant | audio: 5kbps-1Mbps<br>video:10kbps-5Mbps | yes, 100's msec |
| stored audio/video | loss-tolerant | same as above | |
| interactive games | loss-tolerant | few kbps up | yes, few secs |
| text messaging | no loss | elastic | yes, 100's msec<br>yes and no |

# DNS: domain name system

*Internet hosts, routers:*

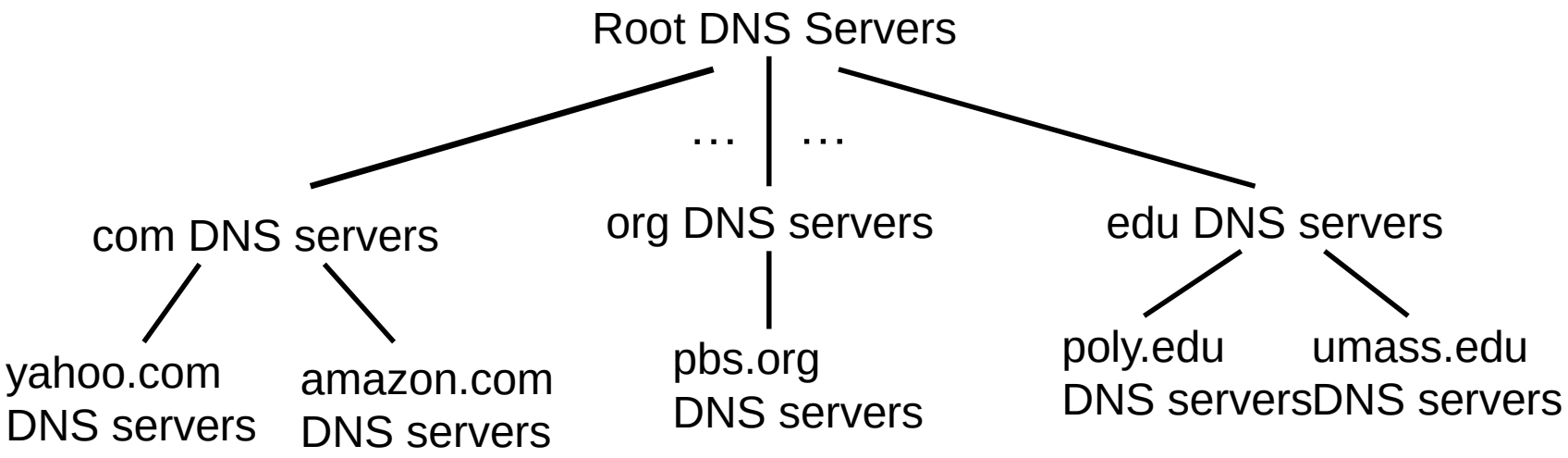
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

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

*Domain Name System:*

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

# DNS: a distributed, hierarchical database




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

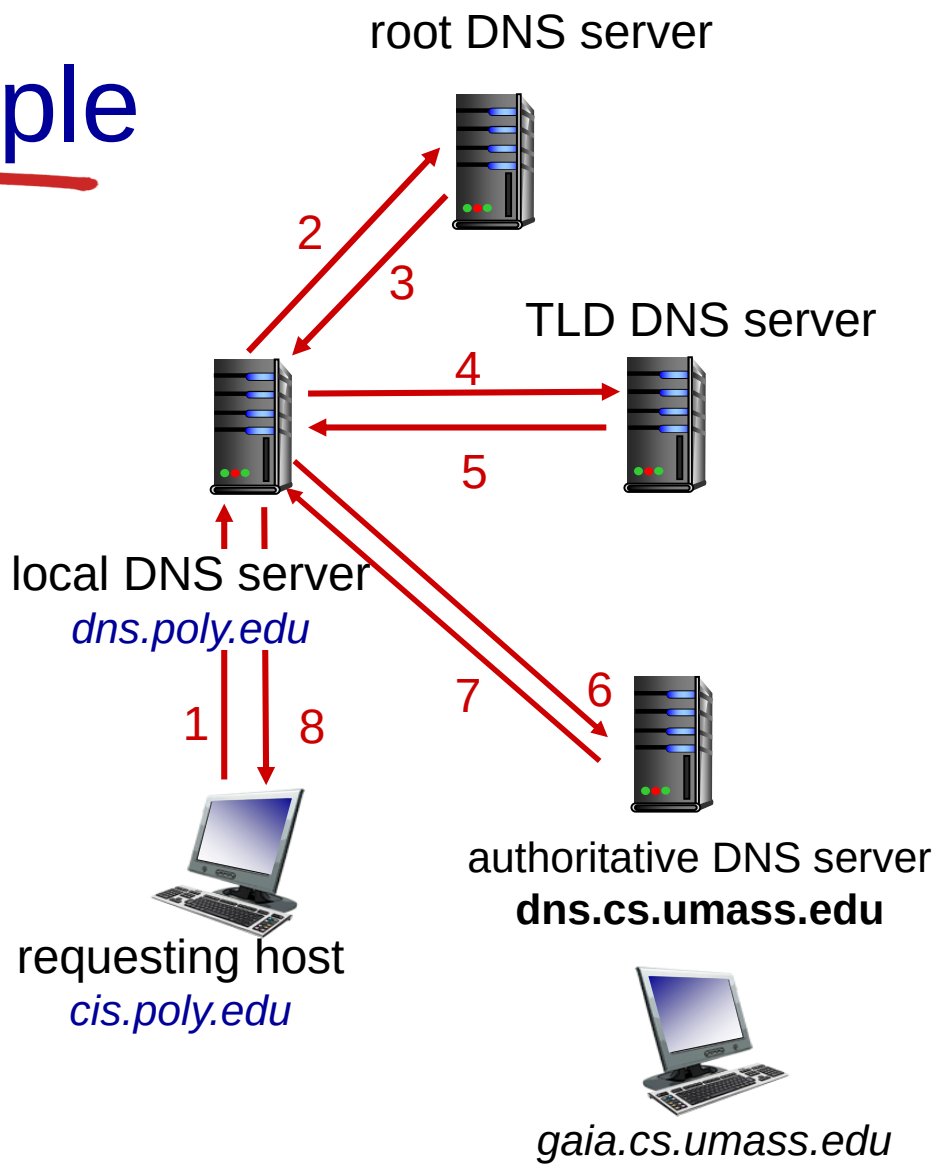
- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

# DNS name resolution example

- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

*iterated query:*

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

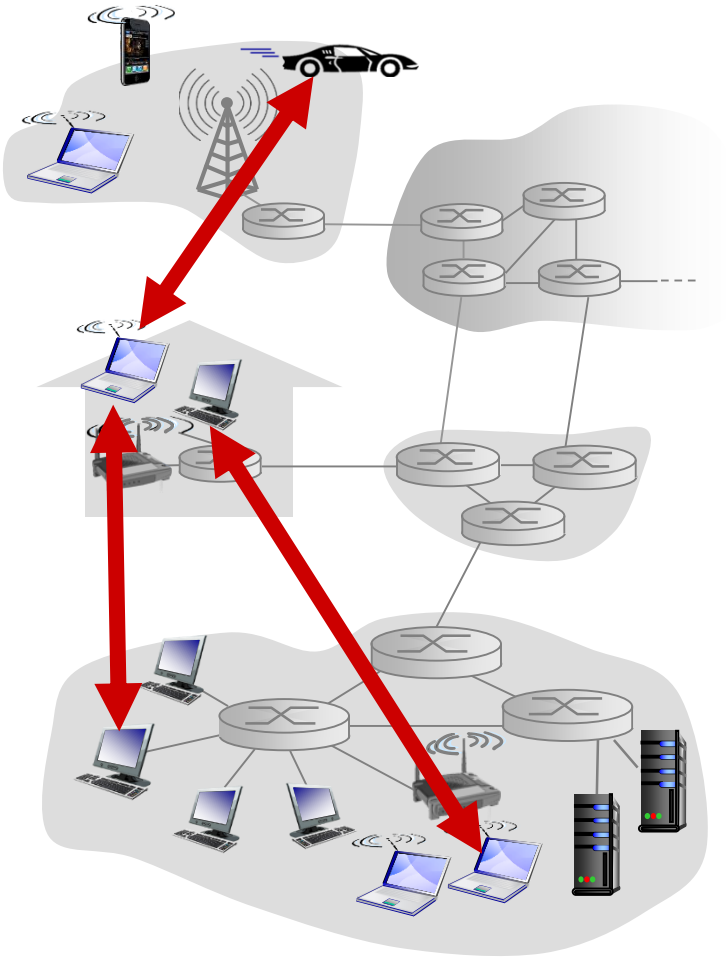


# Peer-to-Peer Applications

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

*examples:*

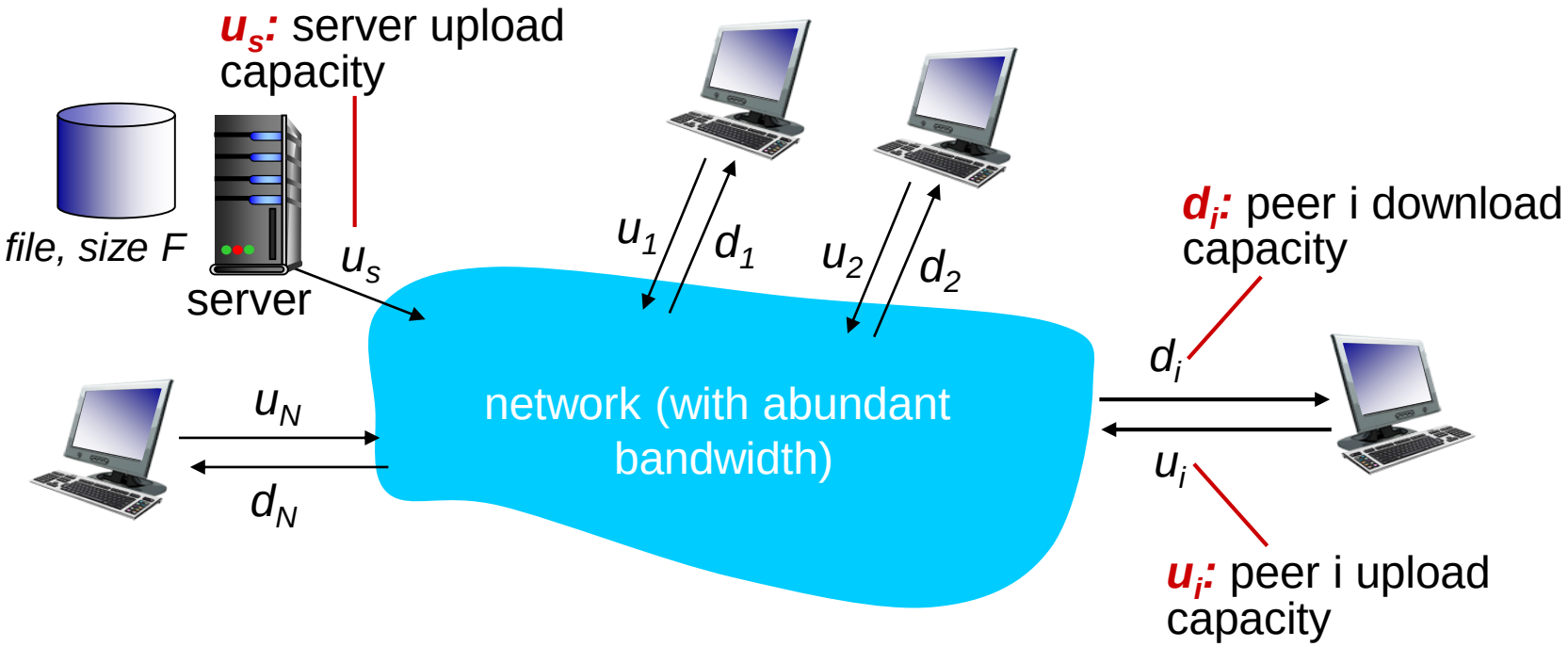
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



# File distribution: client-server vs P2P

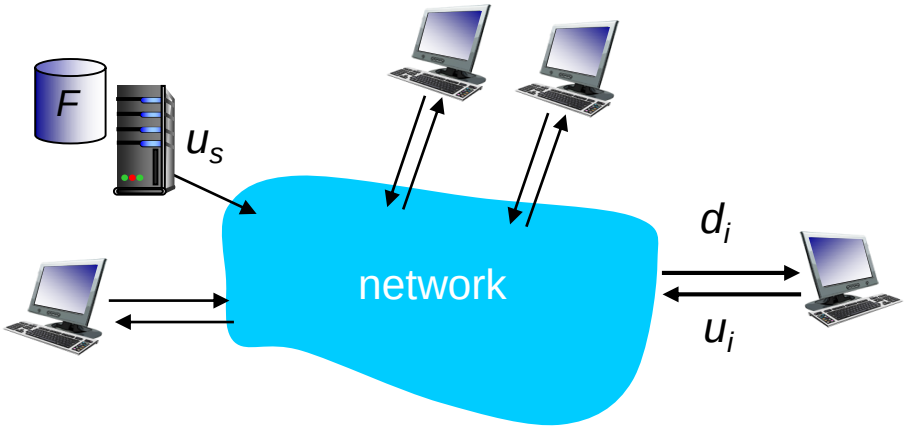
*Question:* how much time to distribute file (size *F*) from one server to *N* *peers*?

- peer upload/download capacity is limited resource



# File distribution time: client-server

- *server transmission:* must sequentially send (upload) *N* file copies:
  - time to send one copy: $F/u_s$
  - time to send *N* copies: $NF/u_s$




- *client:* each client must download file copy
  - $d_{min}$ = min client download rate
  - min client download time: $F/d_{min}$

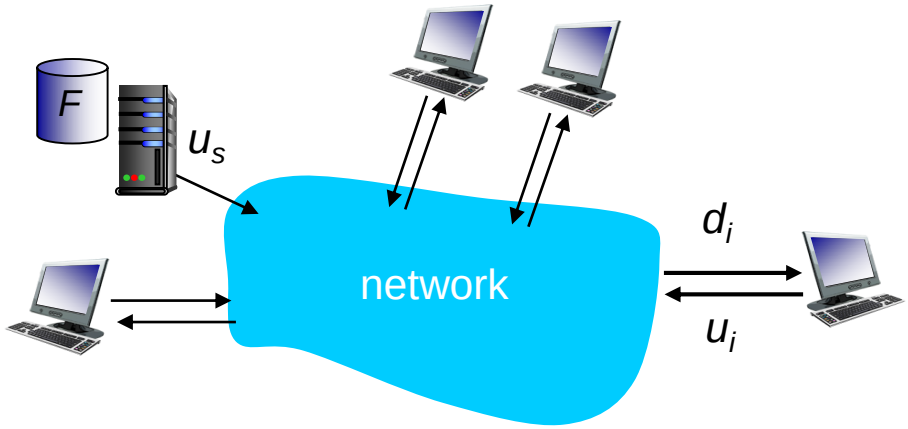
*time to distribute F to N clients using client-server approach*

$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

# File distribution time: P2P

- *server transmission:* must upload at least one copy
  - time to send one copy: $F/u_s$
- *client:* each client must download file copy
  - min client download time: $F/d_{min}$




- *clients:* as aggregate must download *NF* bits
  - max upload rate (limiting max download rate) is $u_s + \Sigma u_i$

*time to distribute F to N clients using P2P approach*

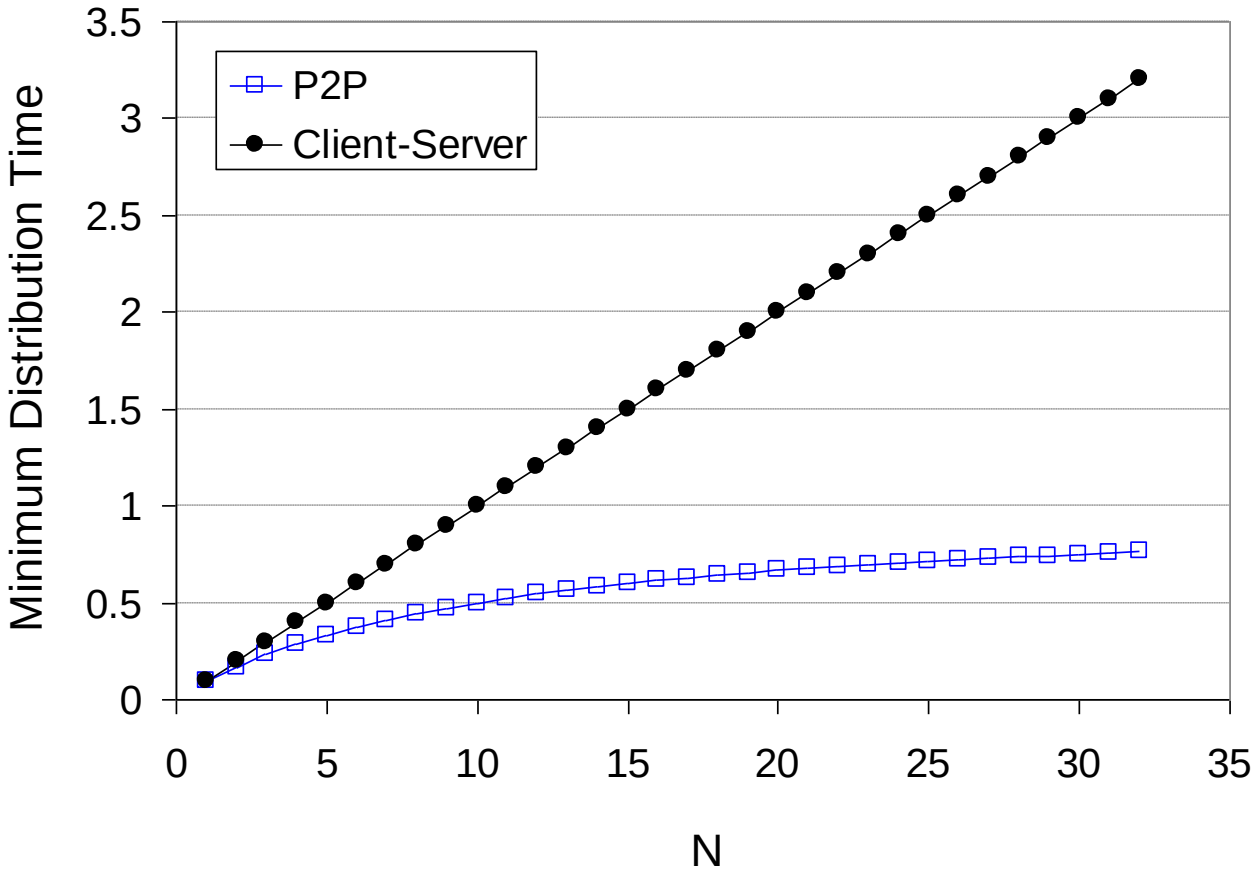
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

increases linearly in *N* …

… but so does this, as each peer brings service capacity

# Client-server vs. P2P: example

client upload rate = $u$, $F/u$ = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$



# A Less Brief Overview of Layers

- Application layer
- **Transport layer**
- Network layer
- Link layer
- Example

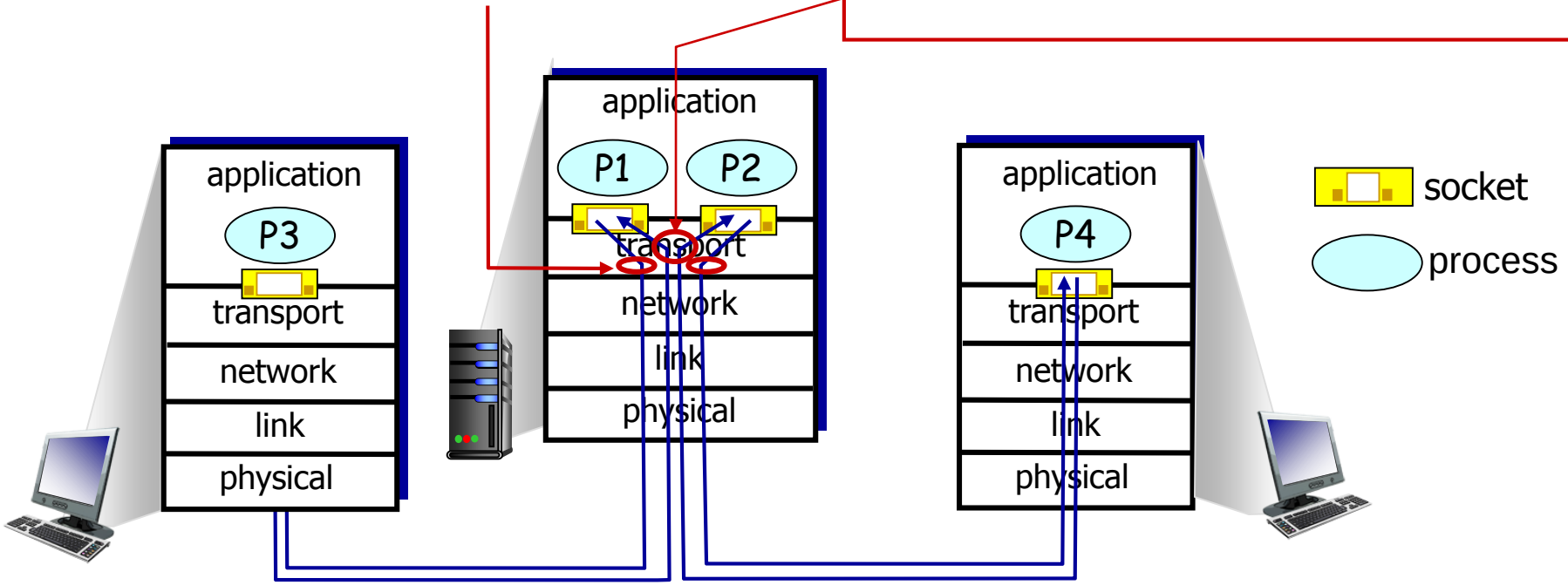
# Multiplexing/demultiplexing

*multiplexing at sender:*

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

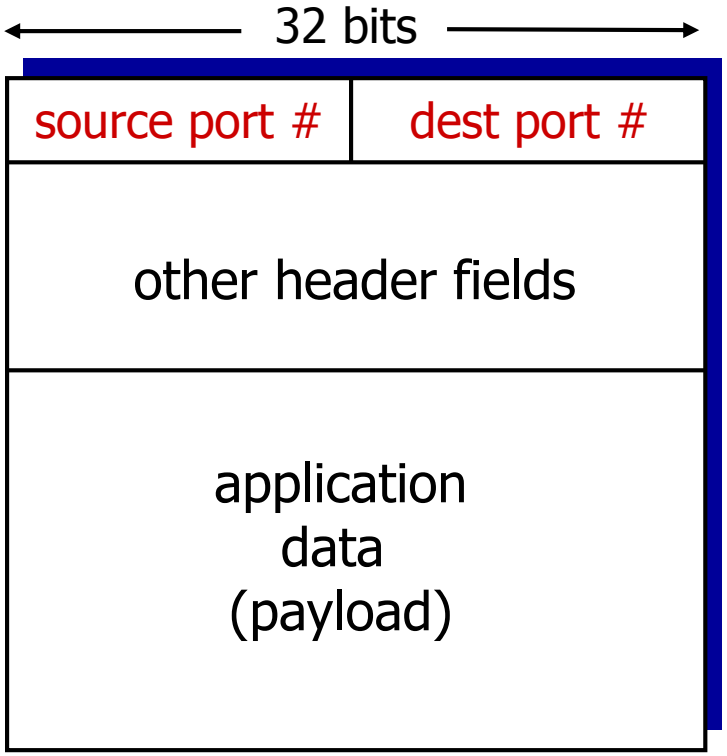
*demultiplexing at receiver:*

use header info to deliver received segments to correct socket



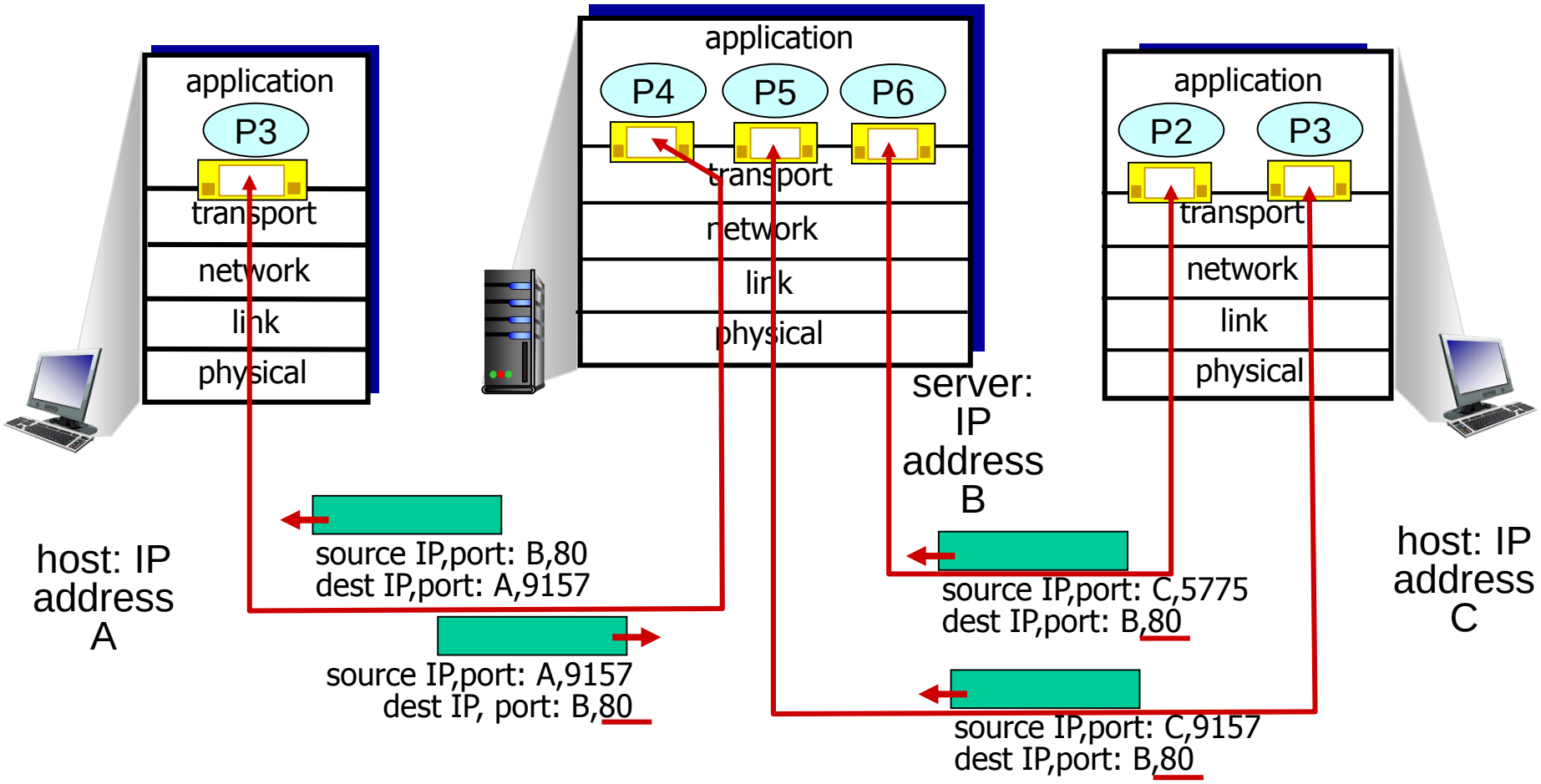
# How demultiplexing works

- host receives IP datagrams
  - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



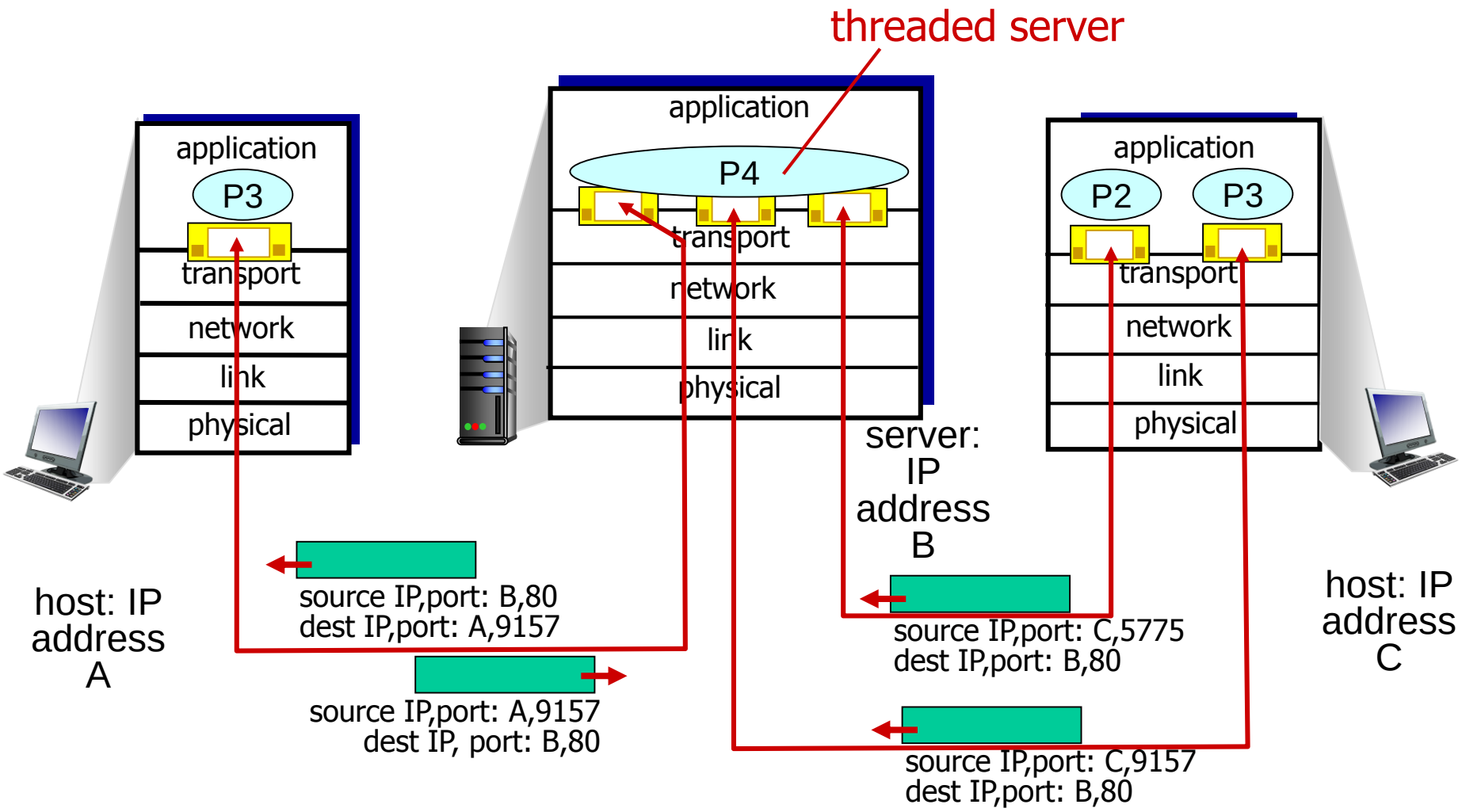

TCP/UDP segment format

# Connection-oriented demux: example

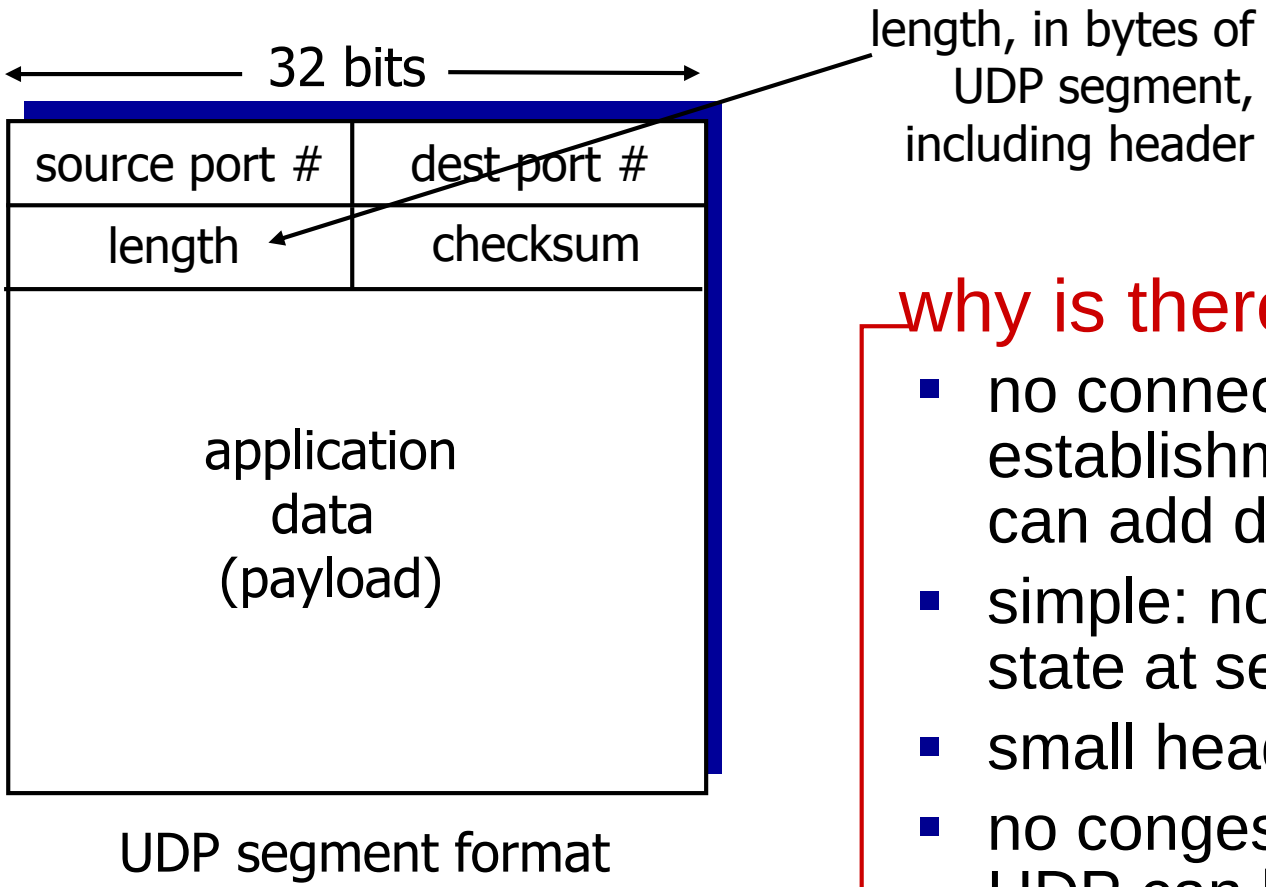



three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

# Connection-oriented demux: example



# UDP: segment header

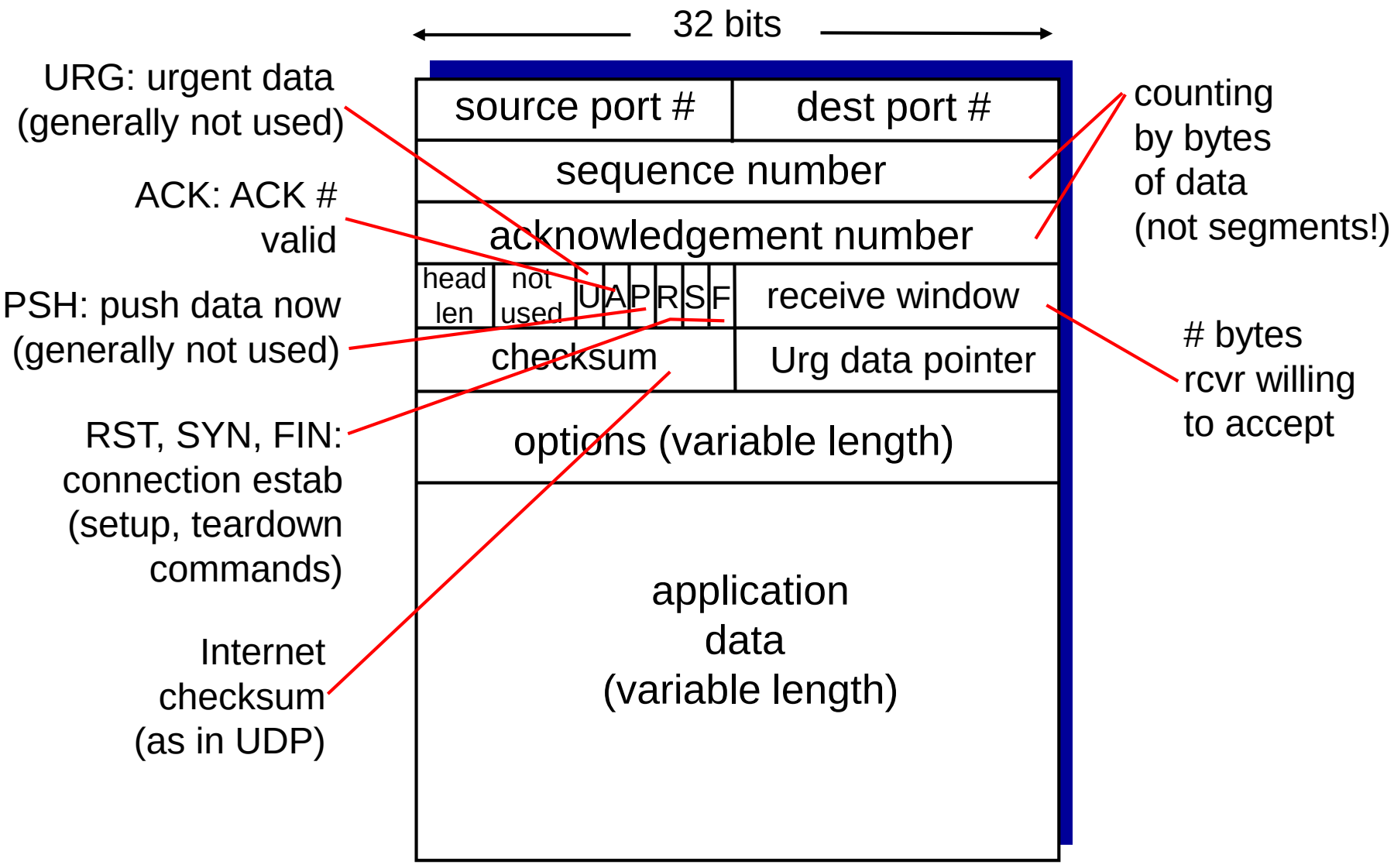



UDP segment format

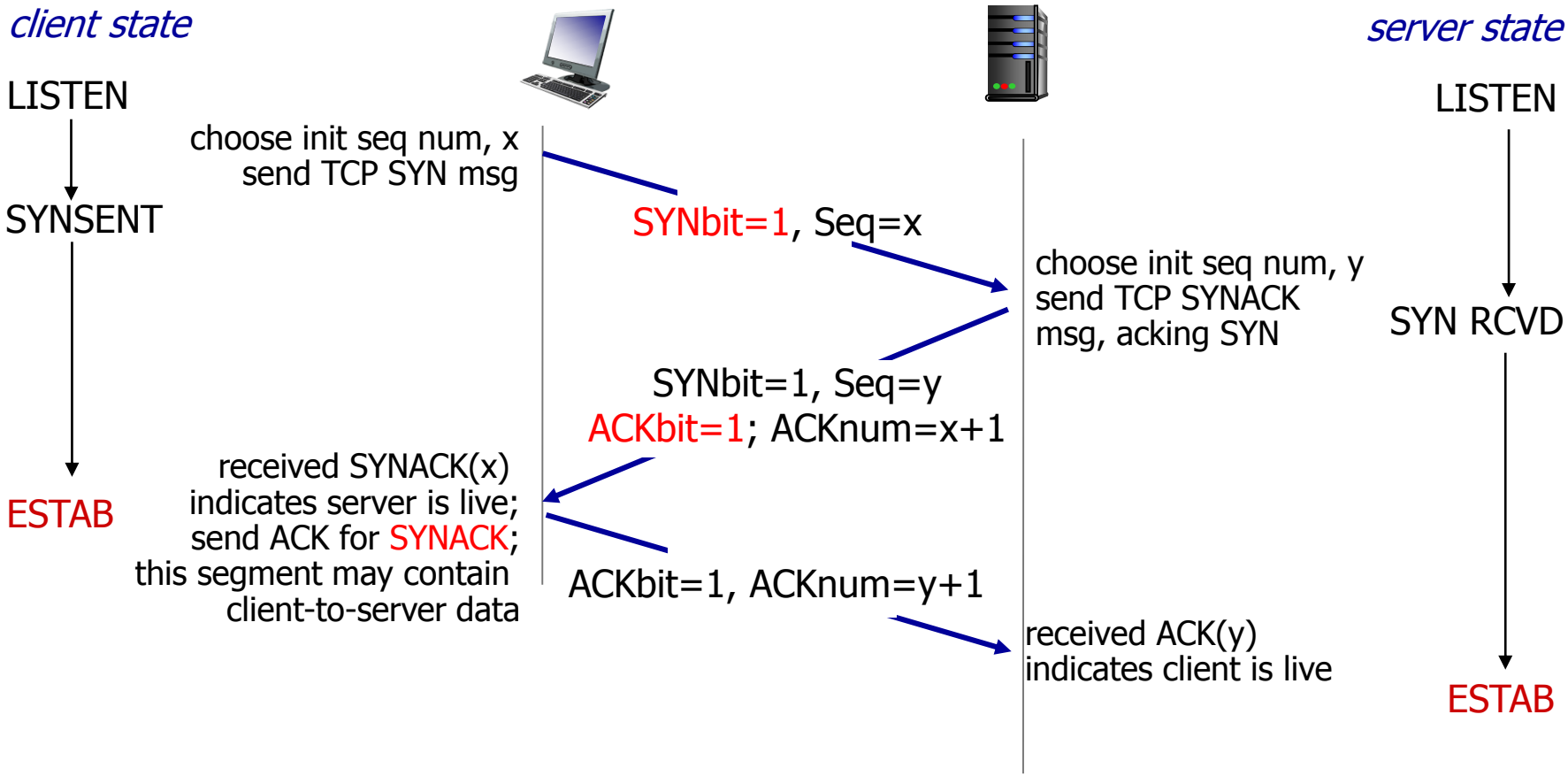
## why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control: UDP can blast away as fast as desired

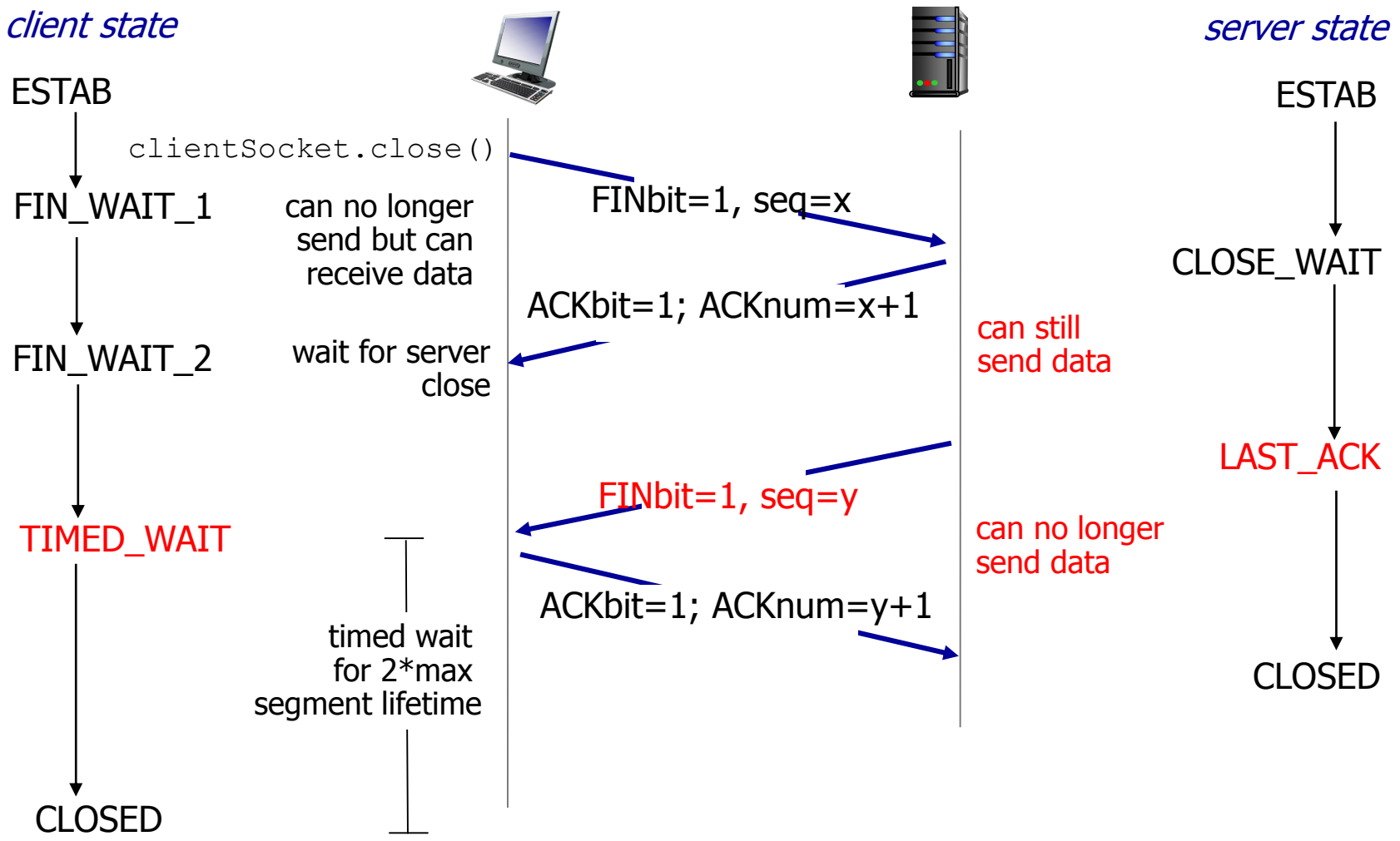
# TCP segment structure



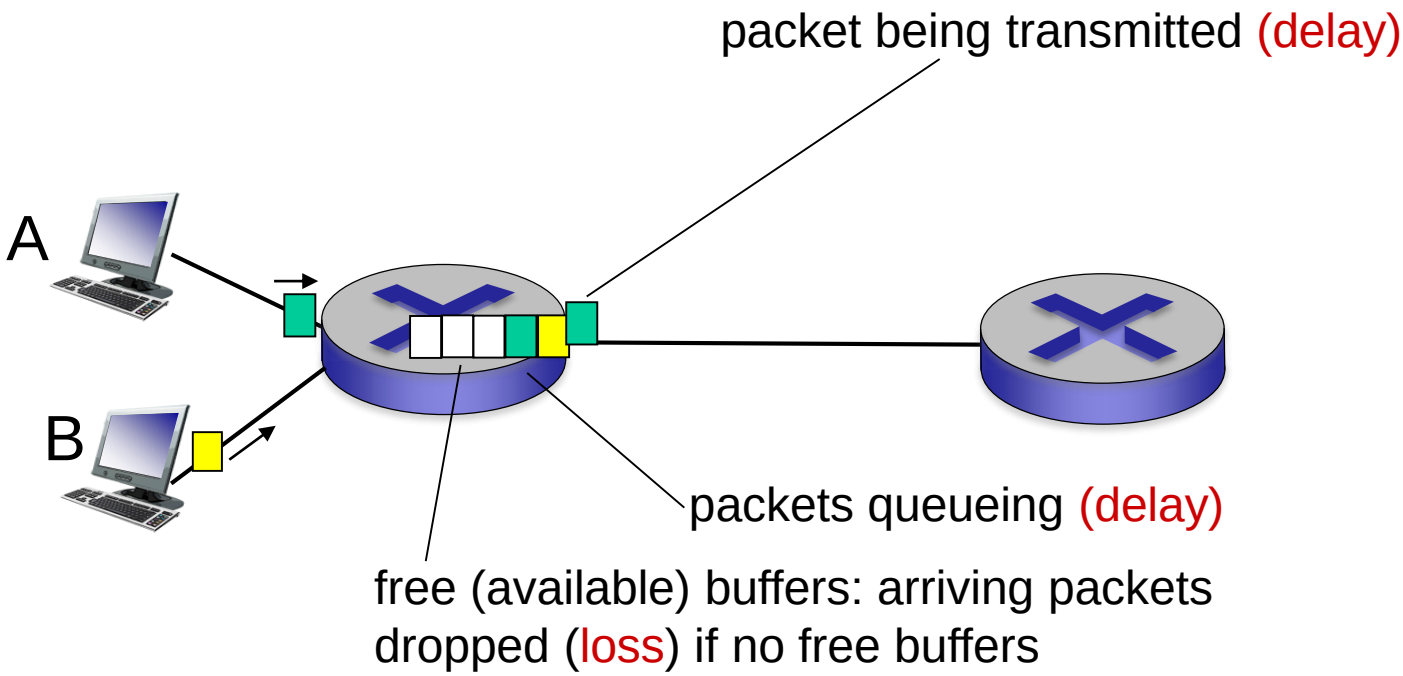
# TCP 3-way handshake



# TCP: closing a connection



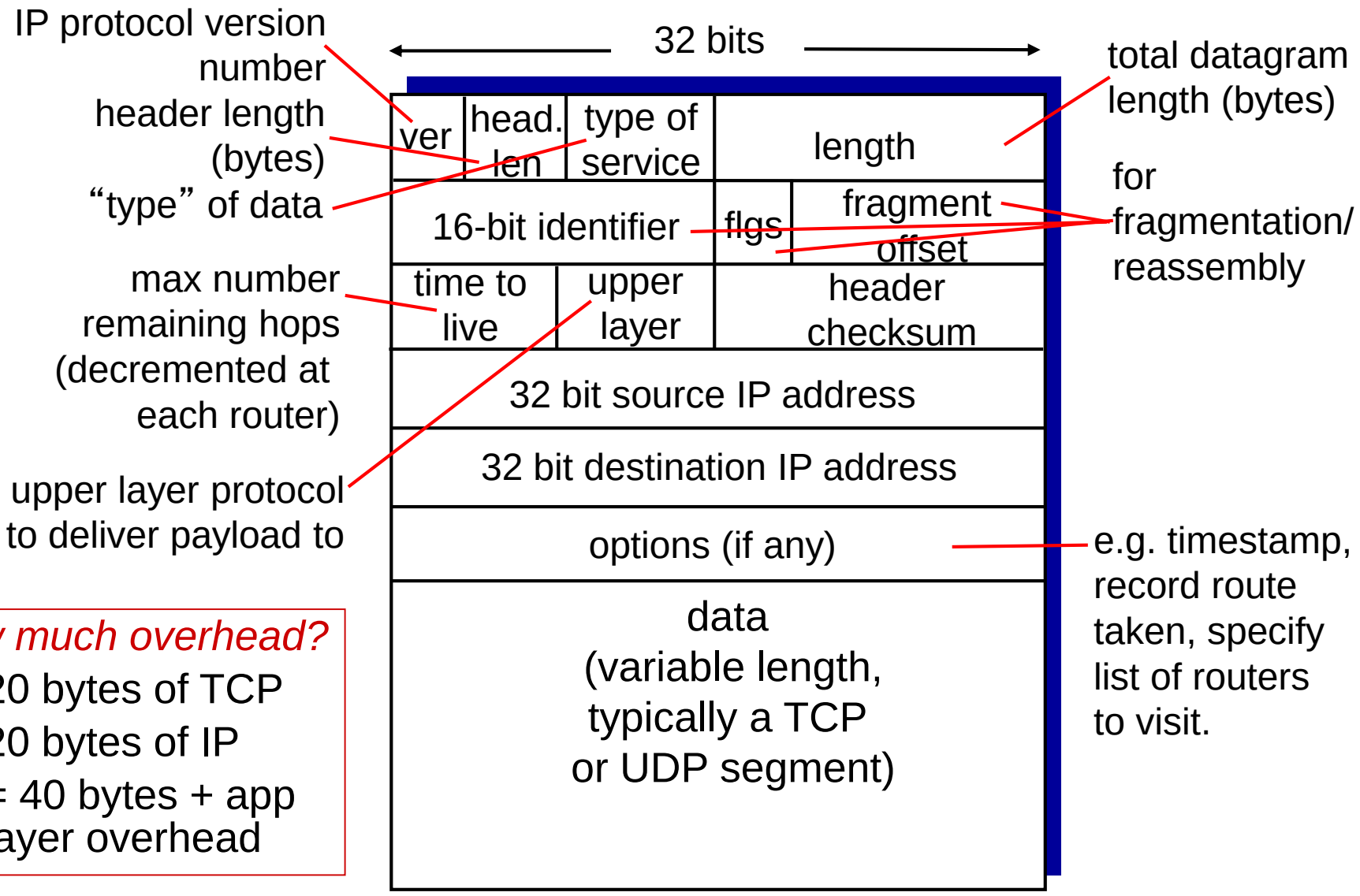
# Why are there delays?



# A Less Brief Overview of Layers

- Application layer
- Transport layer
- **Network layer**
- Link layer
- Example

# IP datagram format

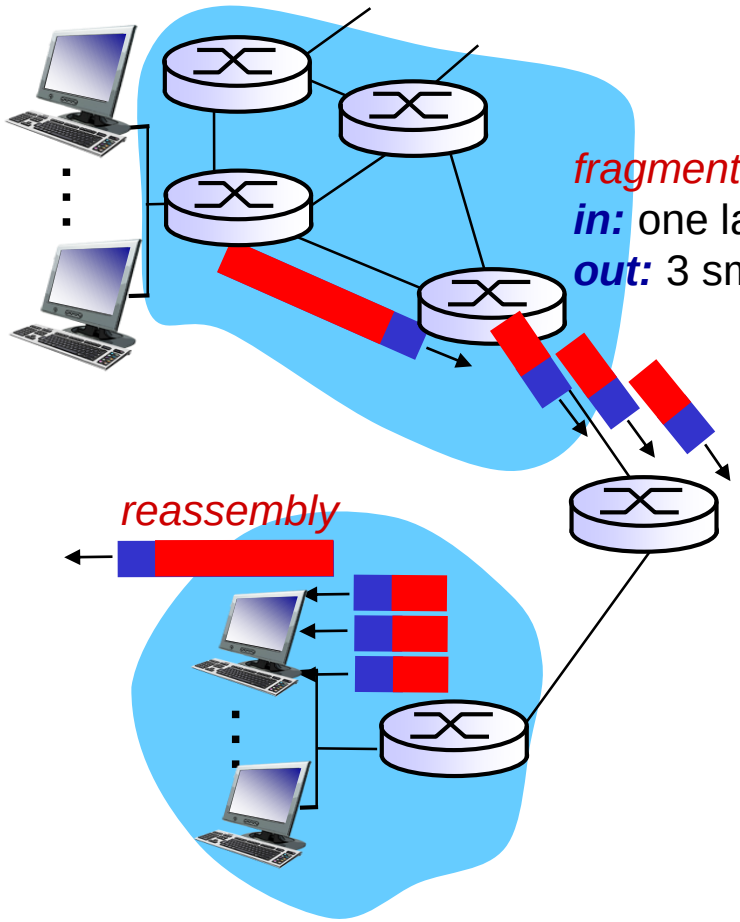



*how much overhead?*

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

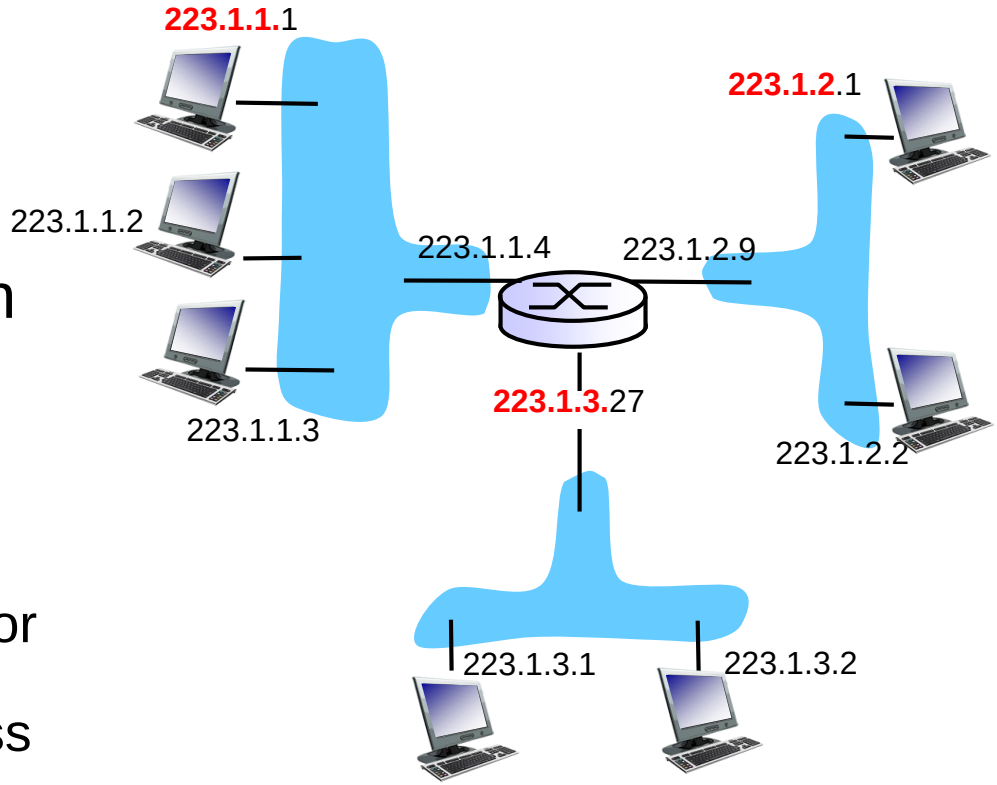
# IP fragmentation, reassembly

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



# IP addressing: introduction

- *IP address:* 32-bit identifier for host, router *interface*
- *interface:* connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)
- *IP addresses associated with each interface*

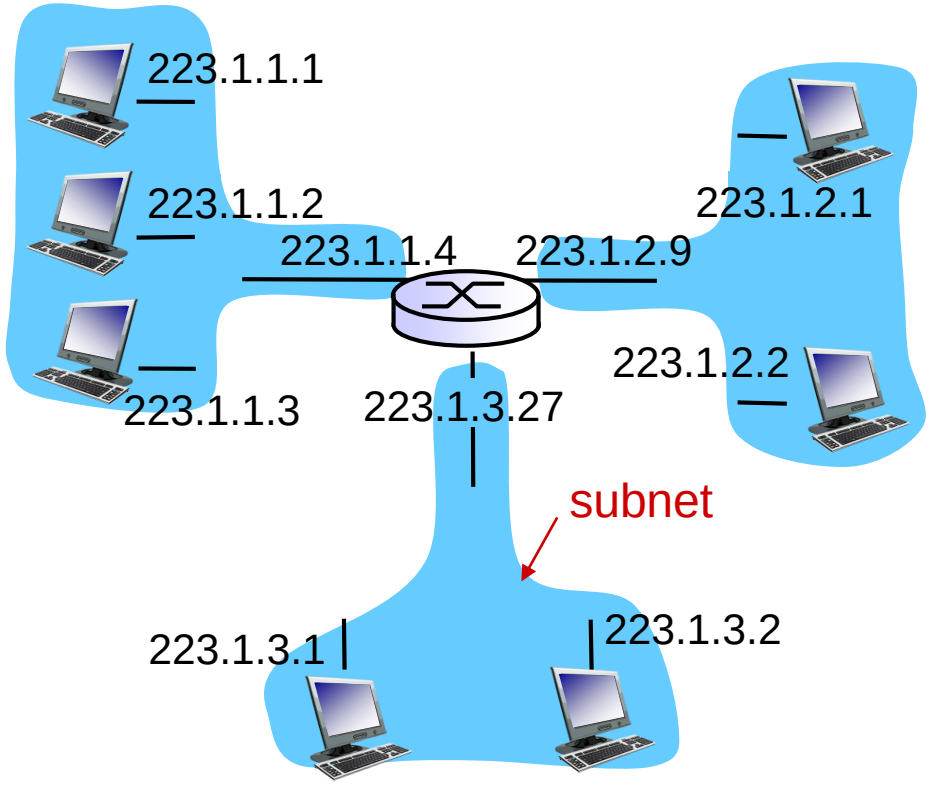



223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

# Subnets

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

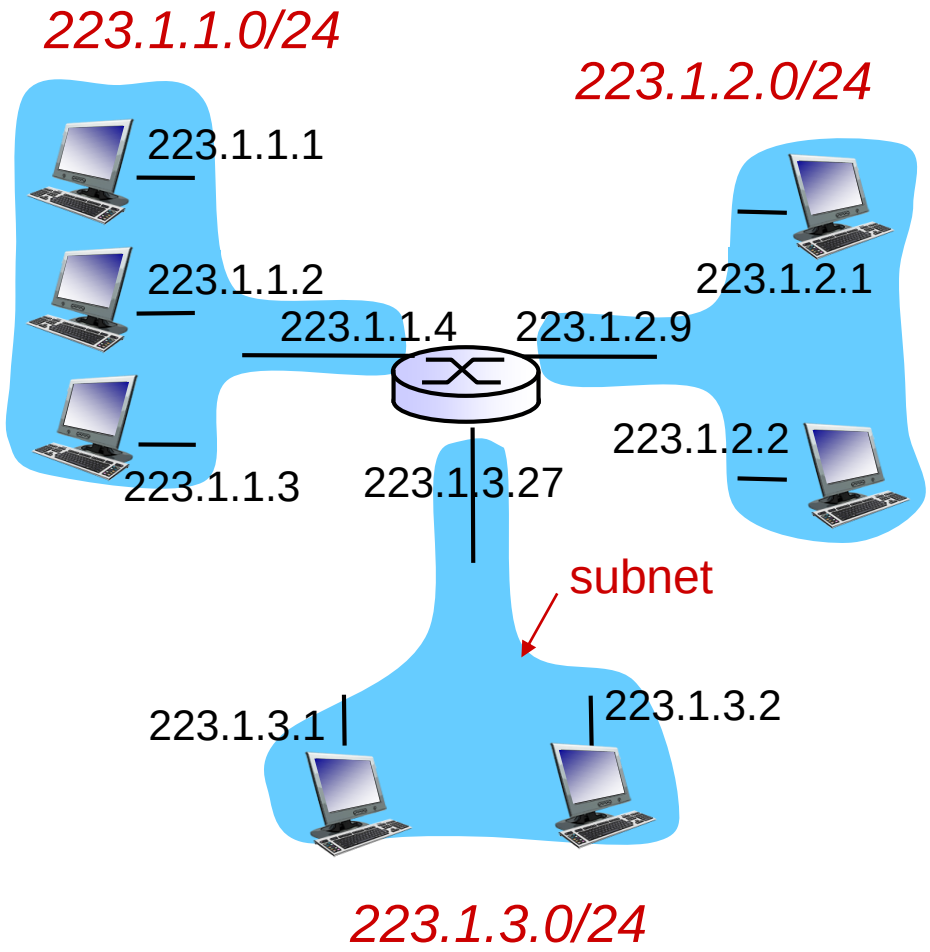



network consisting of 3 subnets

# Subnets

*recipe*

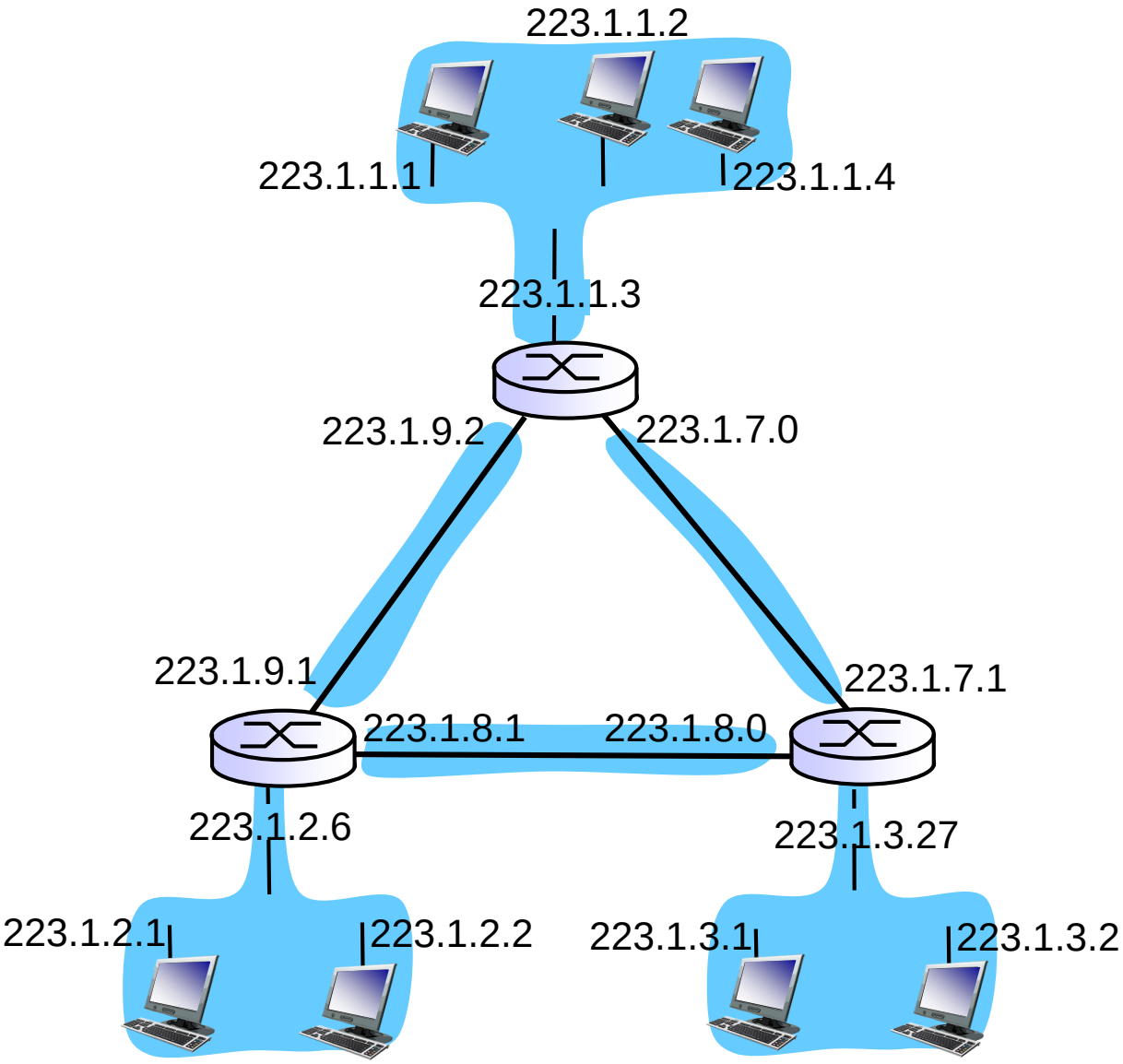
- to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
- each isolated network is called a *subnet*




subnet mask: /24

# Subnets

how many?



# Forwarding: Longest prefix matching

*longest prefix matching*

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

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

examples:

DA: 11001000 00010111 00010110 10100001 which interface?

# Longest prefix matching

*longest prefix matching*

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

| Destination Address Range | Link interface | |
|---|---|---|
| 11001000 00010111 00010*** ********* | 0 | match (longest) |
| 11001000 00010111 00011000 ********* | 1 | no match |
| 11001000 00010111 00011*** ********* | 2 | no match |
| otherwise | 3 | match (but short) |

examples:

DA: 11001000 00010111 00010110 10100001

matches 1st and last: first longer (more specific): interface 1

# Longest prefix matching

*longest prefix matching*

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

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

examples:

DA: 11001000 00010111 00011000 10101010 which interface?

# Longest prefix matching

*longest prefix matching*

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

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

examples:

DA: 11001000 00010111 00011000 10101010

# Longest prefix matching

*longest prefix matching*

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

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

examples:

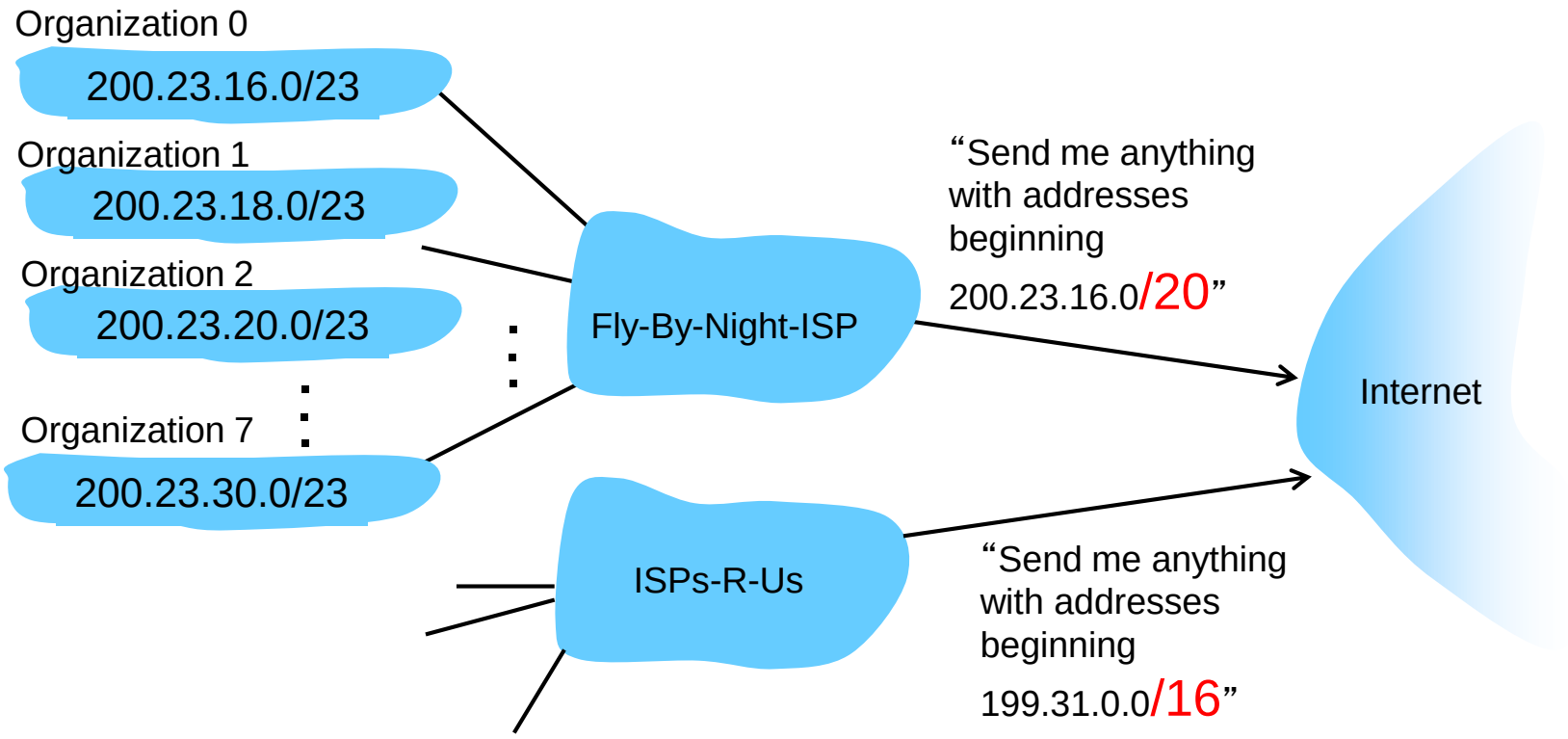
DA: 11001000 00010111 00011000 10101010 So: interface 1!

# Longest prefix matching

- longest prefix matching: often performed using Ternary Content Addressable Memories (TCAMs)
  - *content addressable:* present address to TCAM: retrieve address in one clock cycle, regardless of table size
  - Cisco Catalyst: can up ~1M routing table entries in TCAM

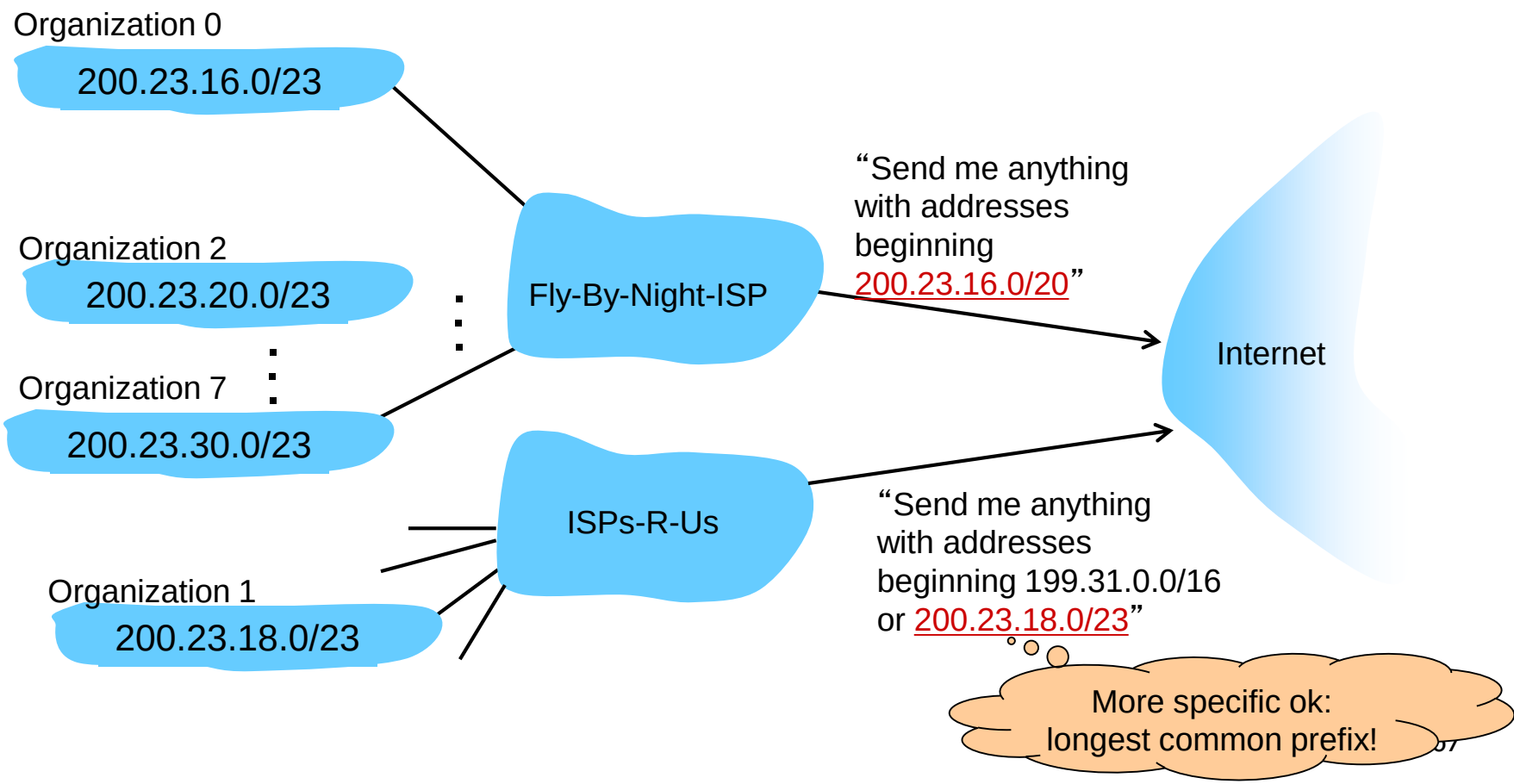
# Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



# Hierarchical addressing: more specific routes

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



# Routing inside a network: Distance vector algorithm

*key idea:*

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

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

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

# Distance vector algorithm

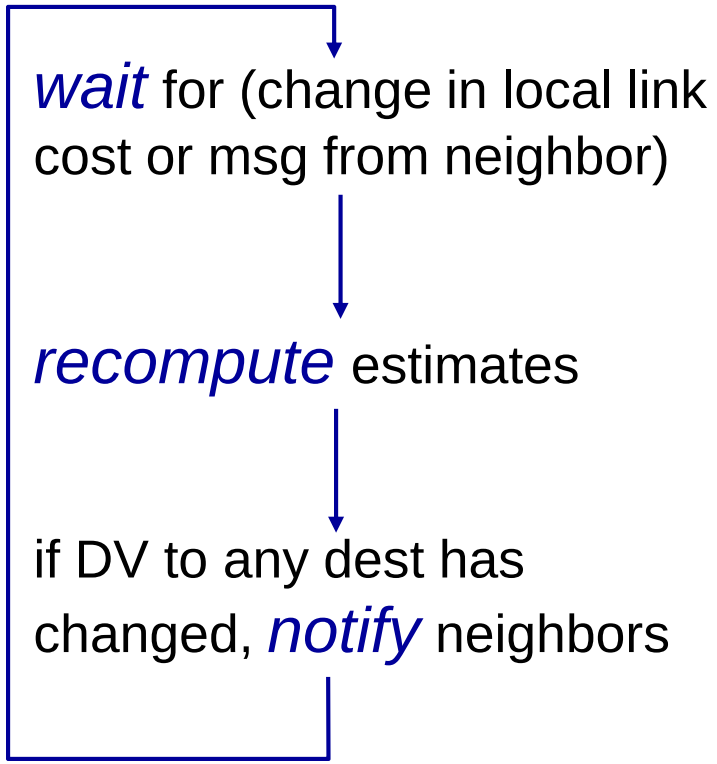
*iterative, asynchronous:* each local iteration caused by:

- local link cost change
- DV update message from neighbor

*distributed:*

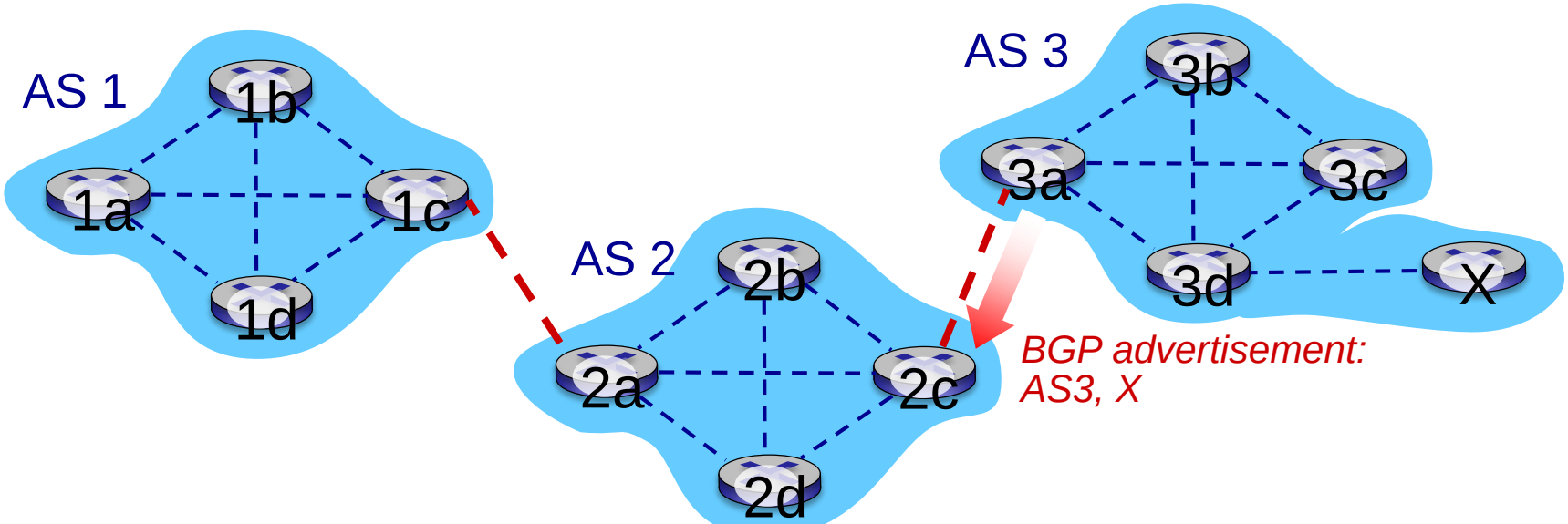
- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

*each node:*



# Between Networks (ASs): BGP

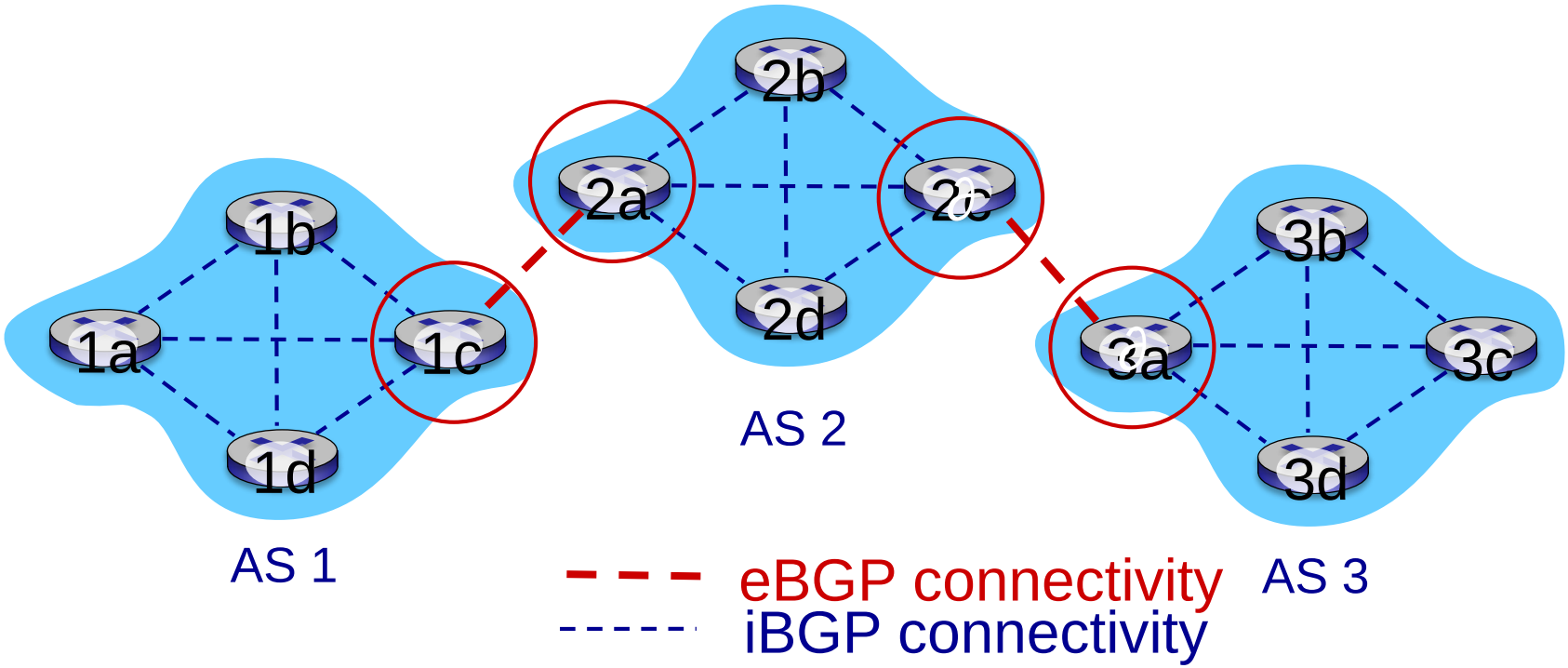
- *Standardized between Autonomous Systems (ASs):* Border-Gateway Protocol (BGP)
- Two BGP routers exchange BGP messages:
  - advertising *paths* to different destination network prefixes (BGP is a "path vector" protocol)
- when AS3 gateway router 3a advertises path AS3,X to AS2 gateway router 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X



# Internet inter-AS routing: BGP

- BGP provides each AS a means to:
  - eBGP: obtain subnet reachability information from neighboring ASes
  - iBGP: propagate reachability information to all AS-internal routers.
  - determine "good" routes to other networks based on reachability information and *policy*
- allows subnet to advertise its existence to rest of Internet: *"I am here"*

# eBGP, iBGP connections



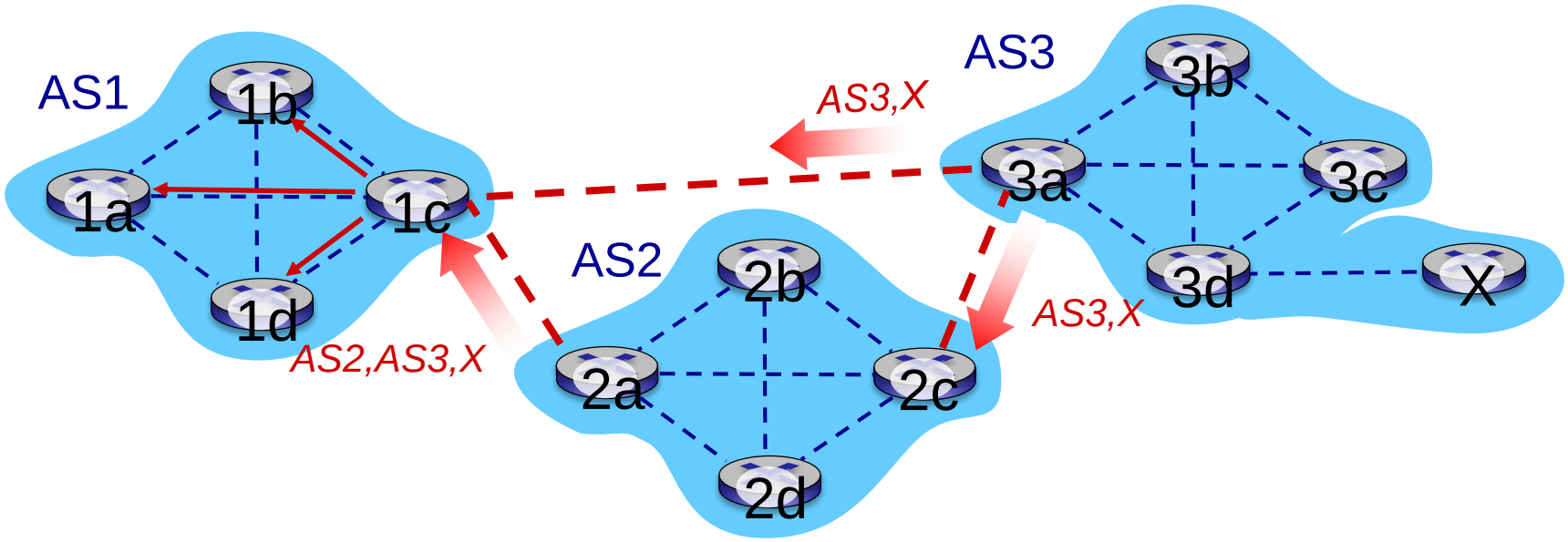


gateway routers run both eBGP and iBGP protools

# Path attributes and BGP routes

- *Policy-based routing:*
  - Import policy: gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
  - Export policy: AS policy also determines whether to *advertise* path to other other neighboring ASes

# BGP path advertisement

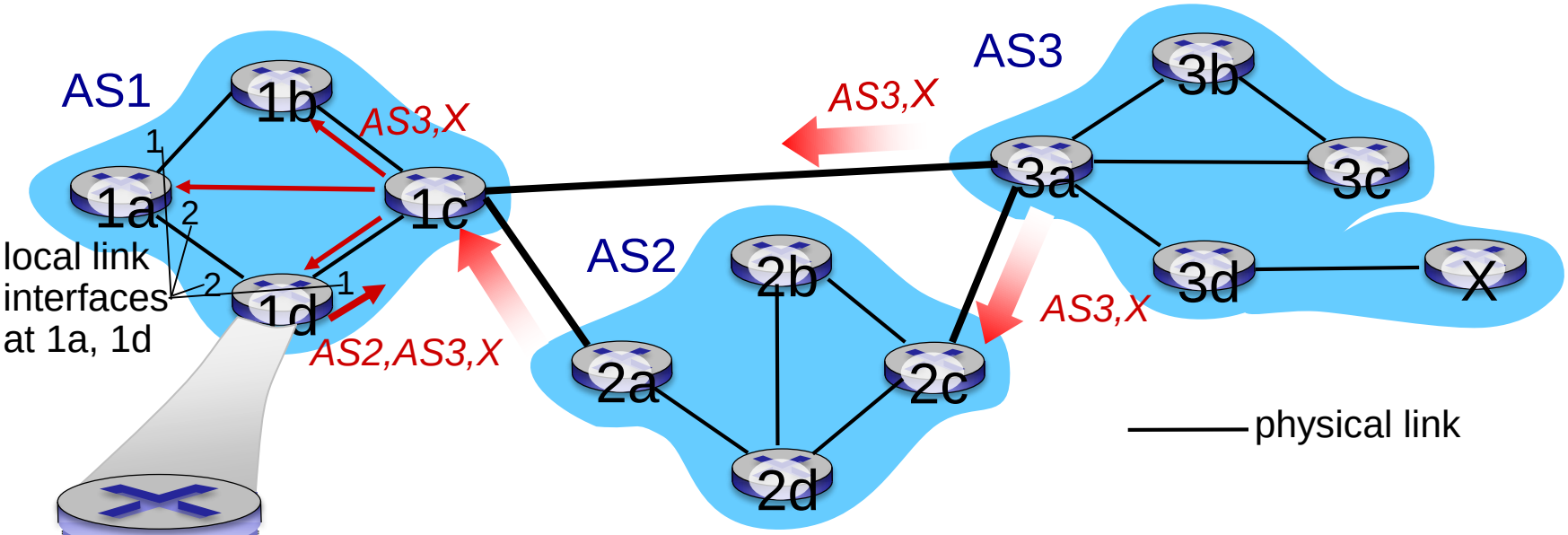


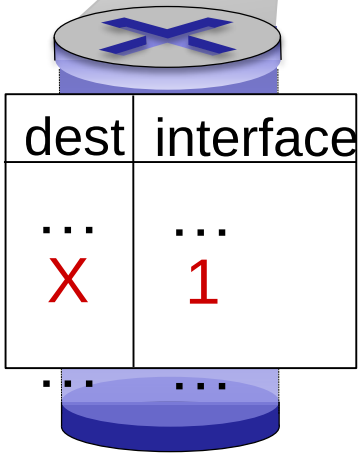

gateway router may learn about multiple paths to destination:

- AS1 gateway router 1c learns path *AS2,AS3,X* from 2a
- AS1 gateway router 1c learns path *AS3,X* from 3a
- Based on policy, AS1 gateway router 1c chooses path *AS3,X, and advertises path within AS1 via iBGP*

# BGP, OSPF, forwarding table entries

Q: how does router set forwarding table entry to distant prefix?

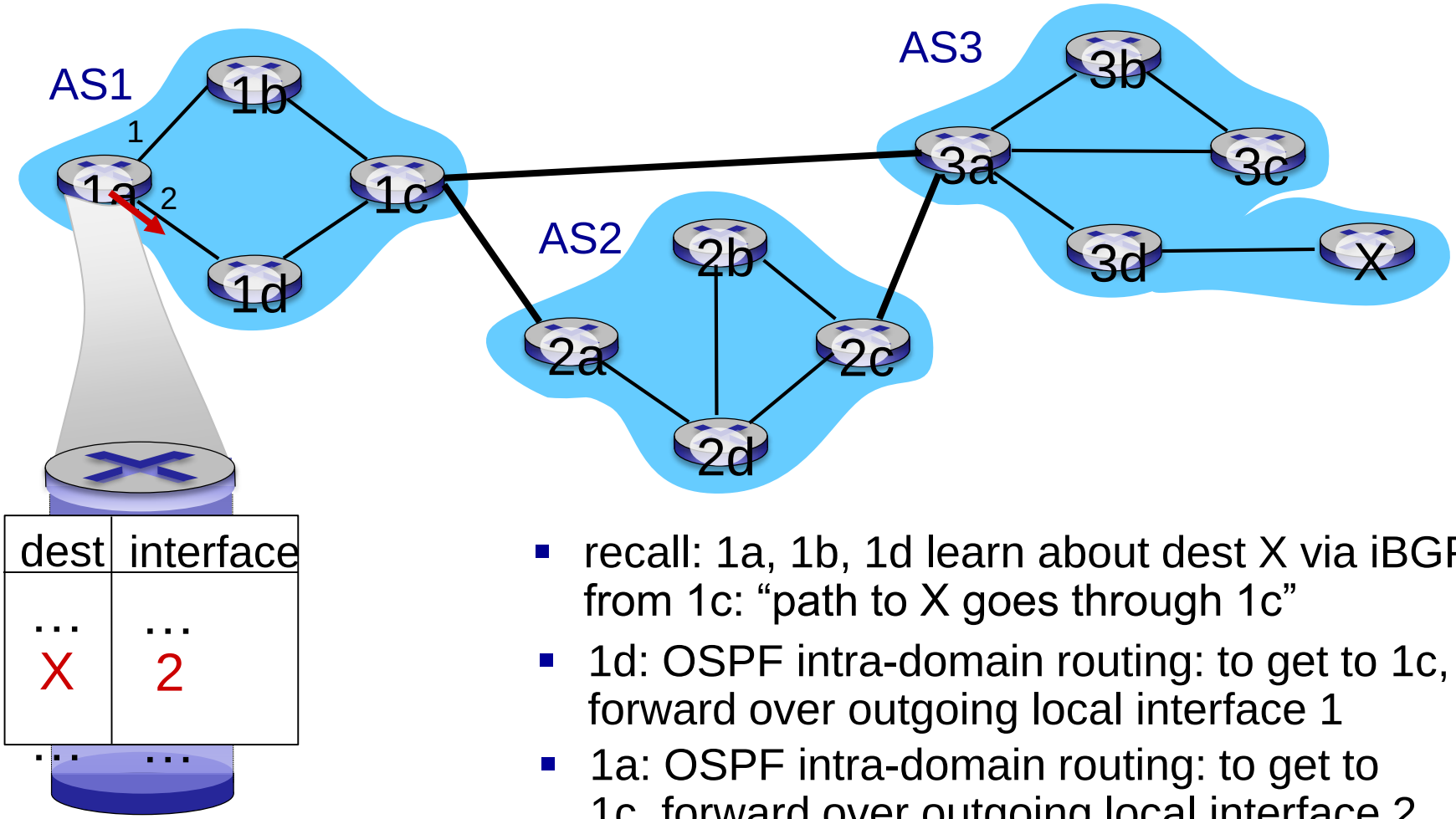



- recall: 1a, 1b, 1d learn about dest X via iBGP from 1c: “path to X goes through 1c”
- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1

# BGP, OSPF, forwarding table entries

Q: how does router set forwarding table entry to distant prefix?

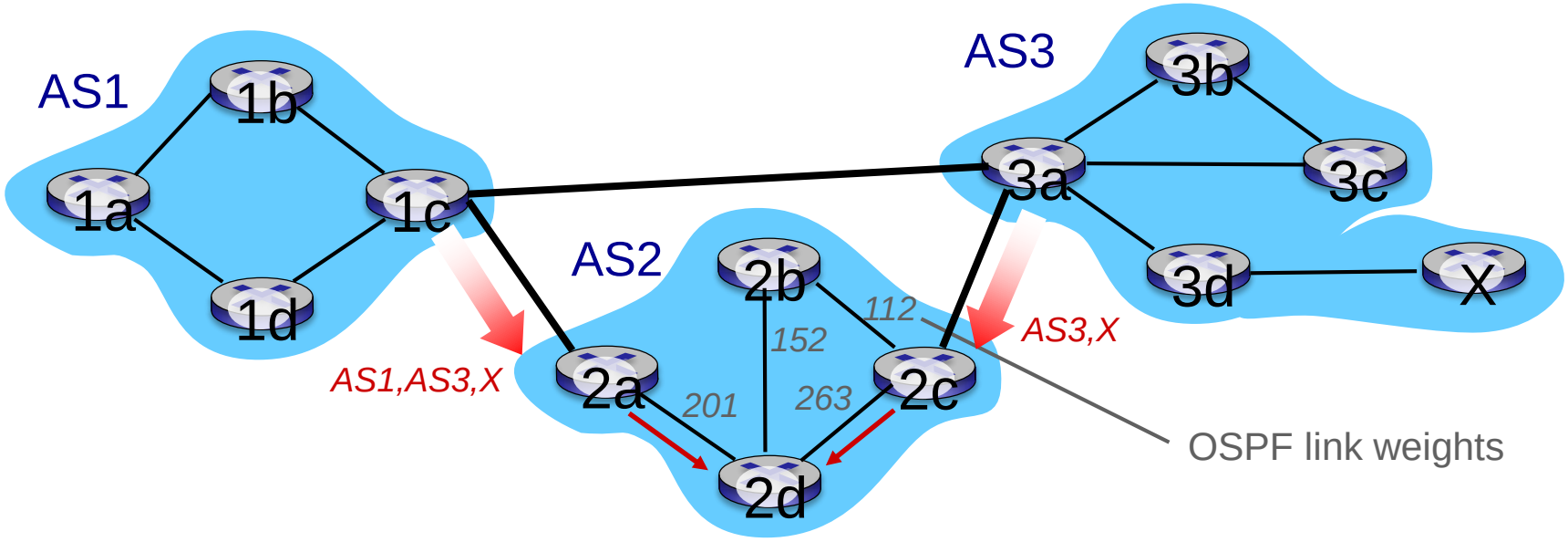



- recall: 1a, 1b, 1d learn about dest X via iBGP from 1c: “path to X goes through 1c”
- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1
- 1a: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 2

# BGP route selection

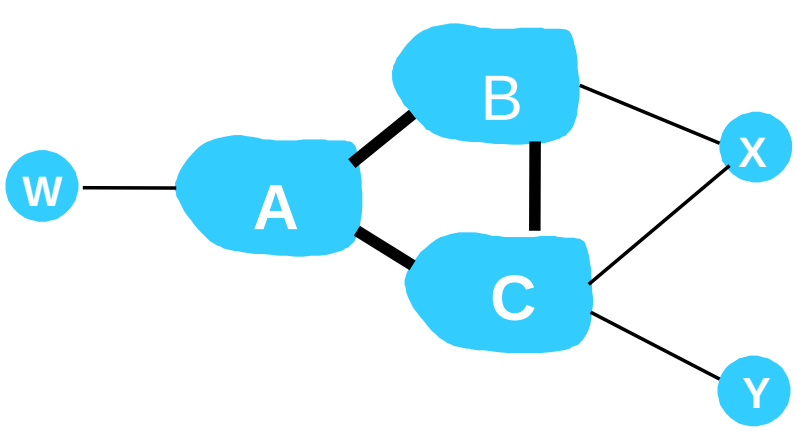
- router may learn about more than one route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria

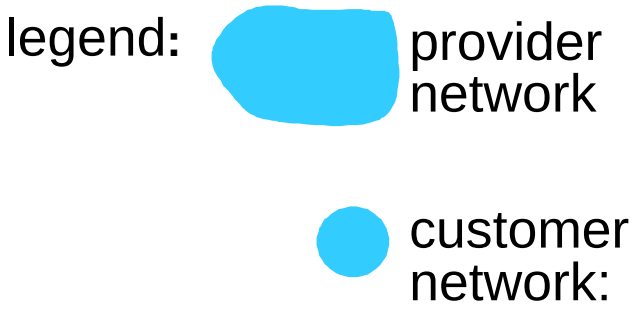
# Hot Potato Routing




- 2d learns (via iBGP) it can route to X via 2a or 2c
- *hot potato routing:* choose local gateway that has least intra-domain cost (e.g., 2d chooses 2a, even though more AS hops to *X*): don't worry about inter-domain cost!

# BGP: achieving policy via advertisements

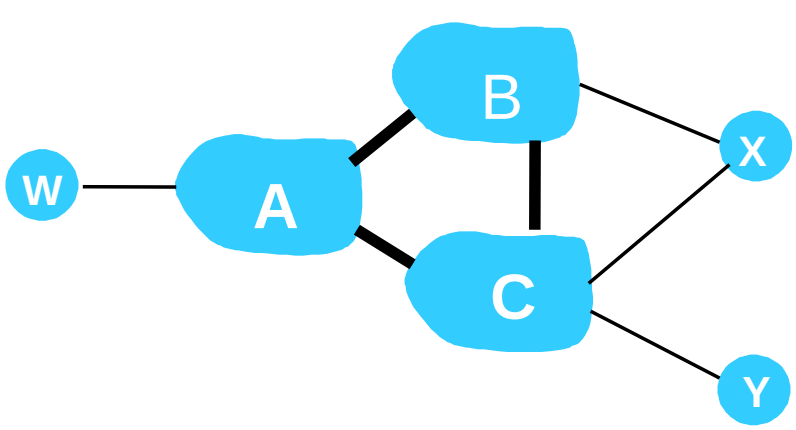


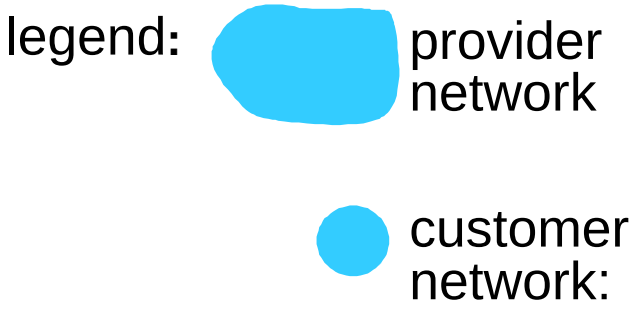




Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- A advertises path Aw to B and to C
- B *chooses not to advertise* BAw to C:
  - B gets no "revenue" for routing CBAw, since none of C, A, w are B's customers
  - C does not learn about CBAw path
- C will route CAw (not using B) to get to w

# BGP: achieving policy via advertisements



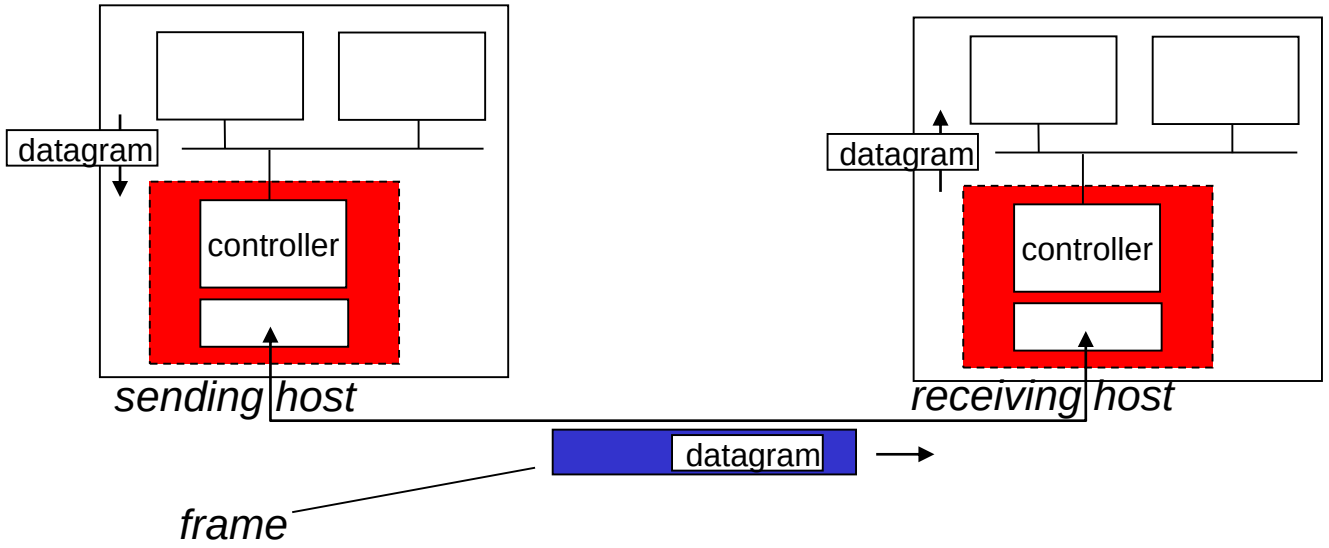

Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- A,B,C are *provider networks*
- X,W,Y are customers (of provider networks)
- X is *dual-homed:* attached to two networks
- *policy to enforce:* X does not want to route traffic from B to C
  - .. so X will not advertise to B a route to C

# A Less Brief Overview of Layers

- Application layer
- Transport layer
- Network layer
- **Link layer**
- Example

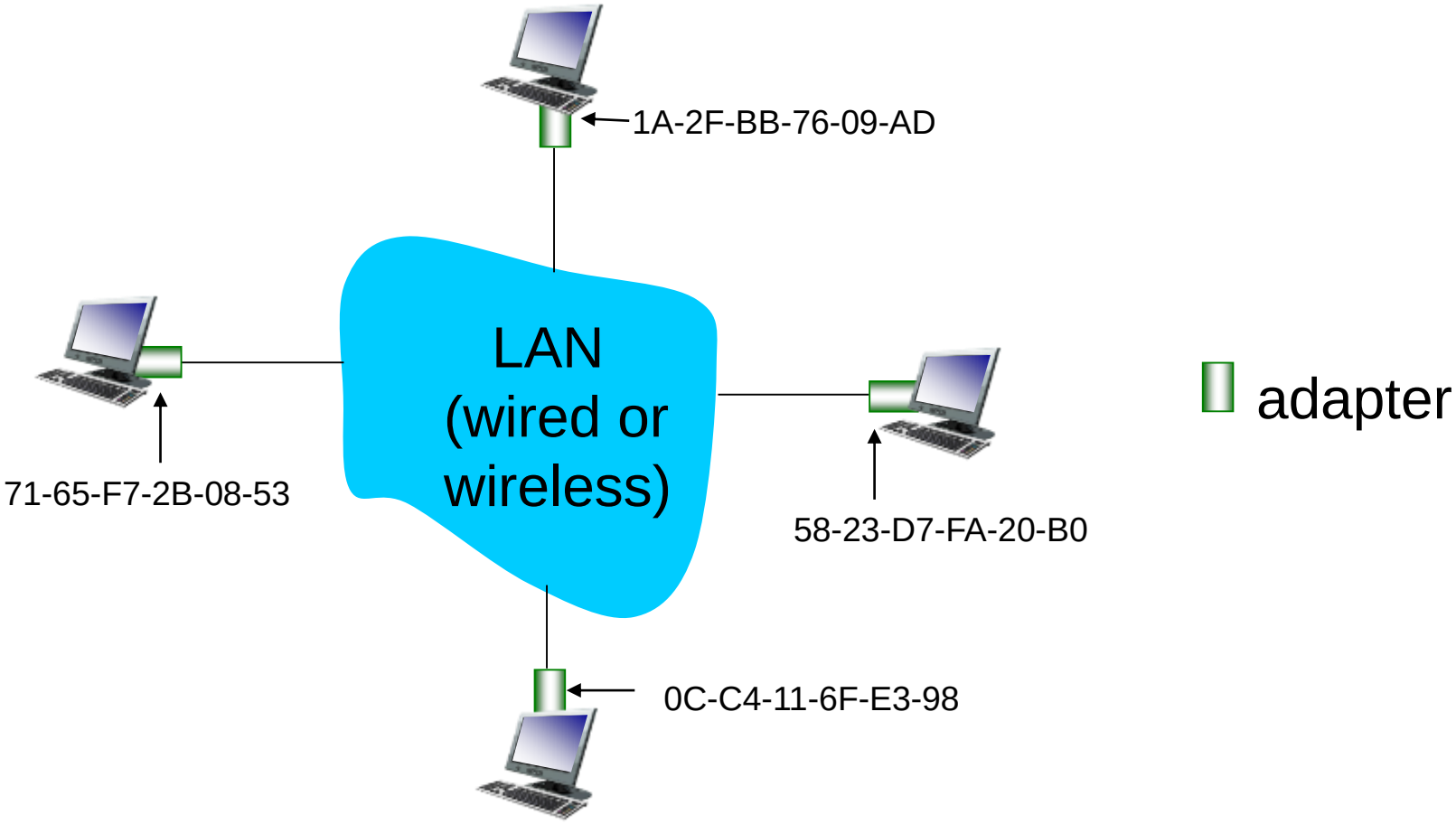
# Adaptors communicating




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

# LAN addresses and ARP

each adapter on LAN has unique *LAN* address

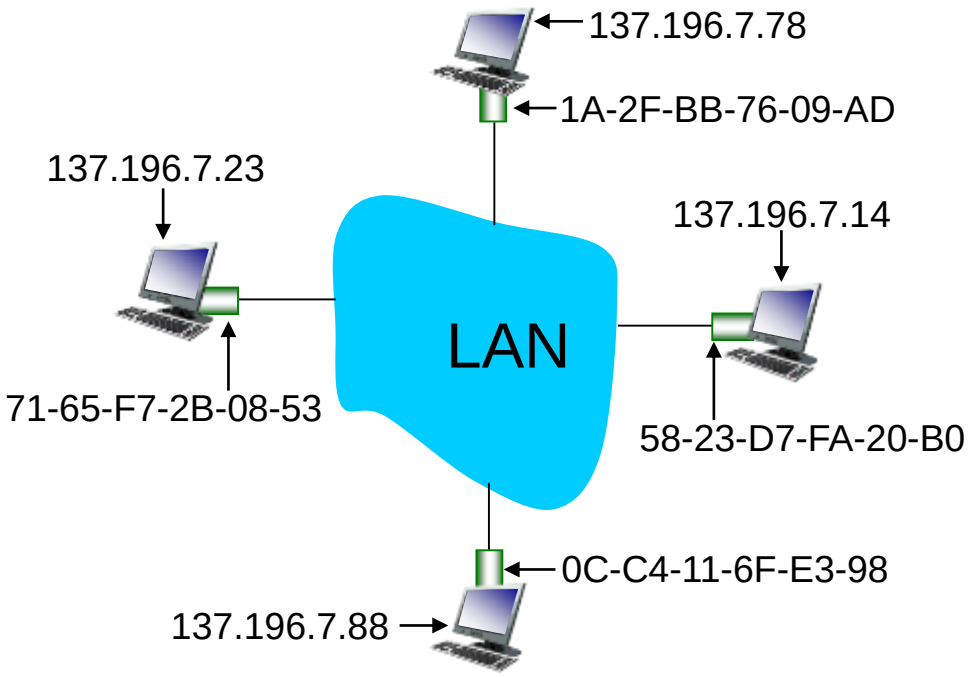


# LAN addresses (more)

- 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

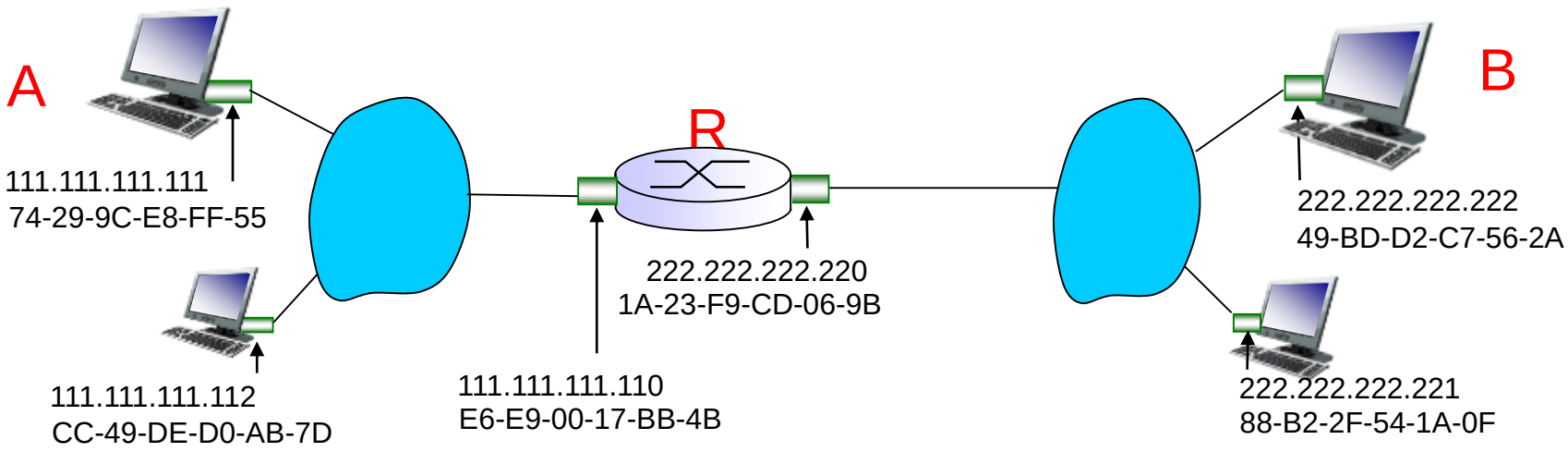
- A wants to send datagram to B
  - B's MAC address not in A's ARP table.
- A broadcasts ARP query packet, containing B's IP address
  - destination MAC address = FF-FF-FF-FF-FF-FF
  - all nodes on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
  - frame sent to 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

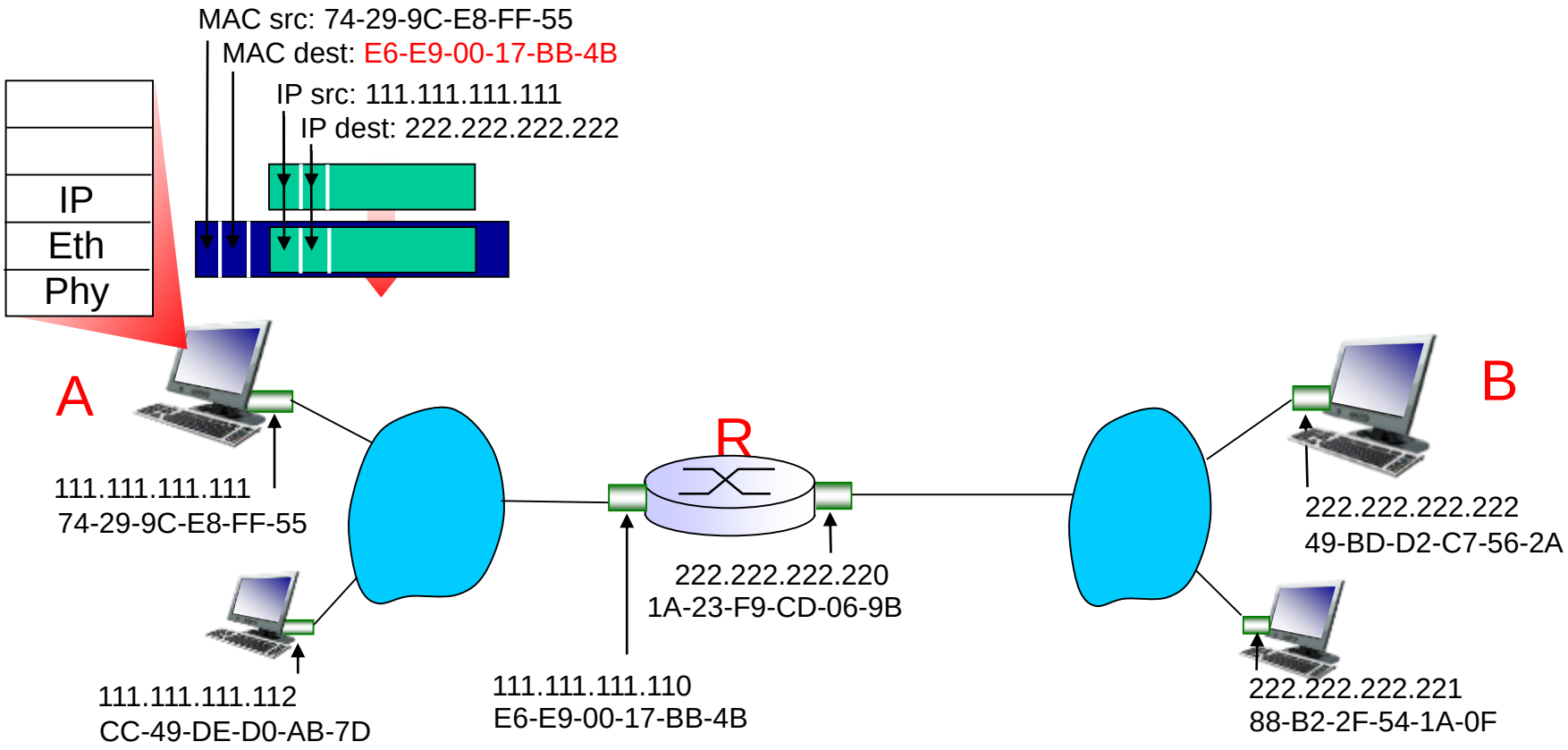
walkthrough: send datagram from A to B via R

- 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
- assume A knows R's MAC address



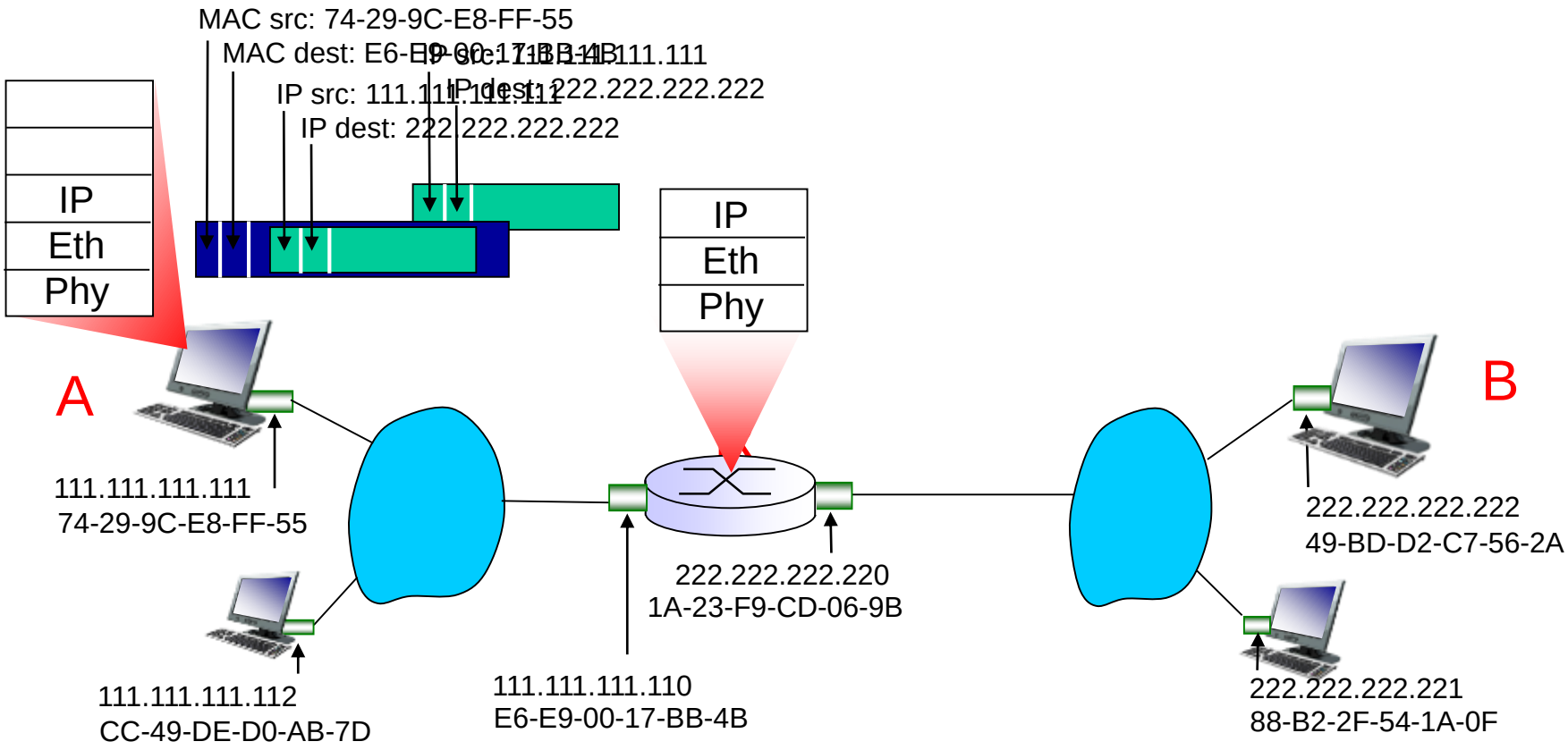
# Addressing: routing to another LAN

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



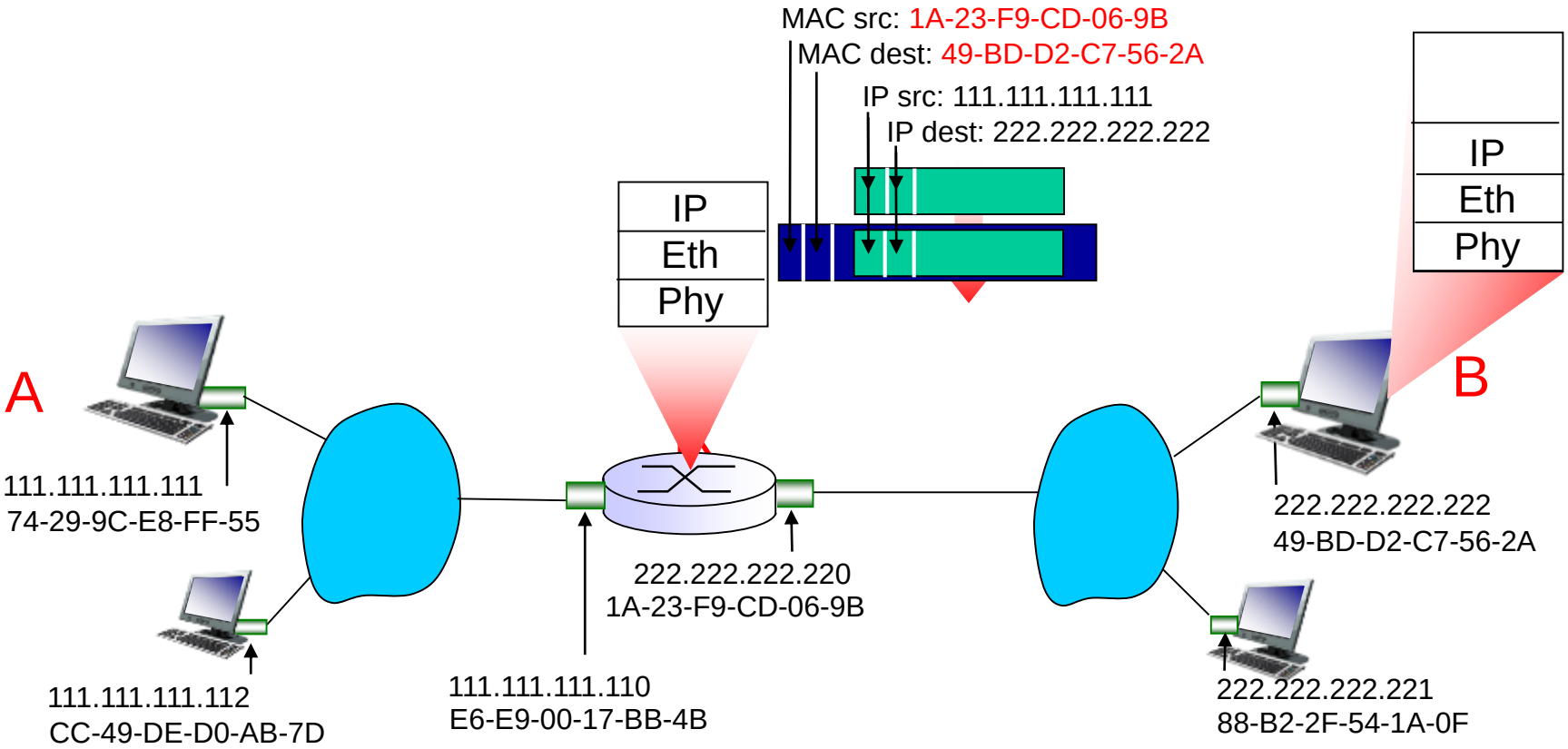
# Addressing: routing to another LAN

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



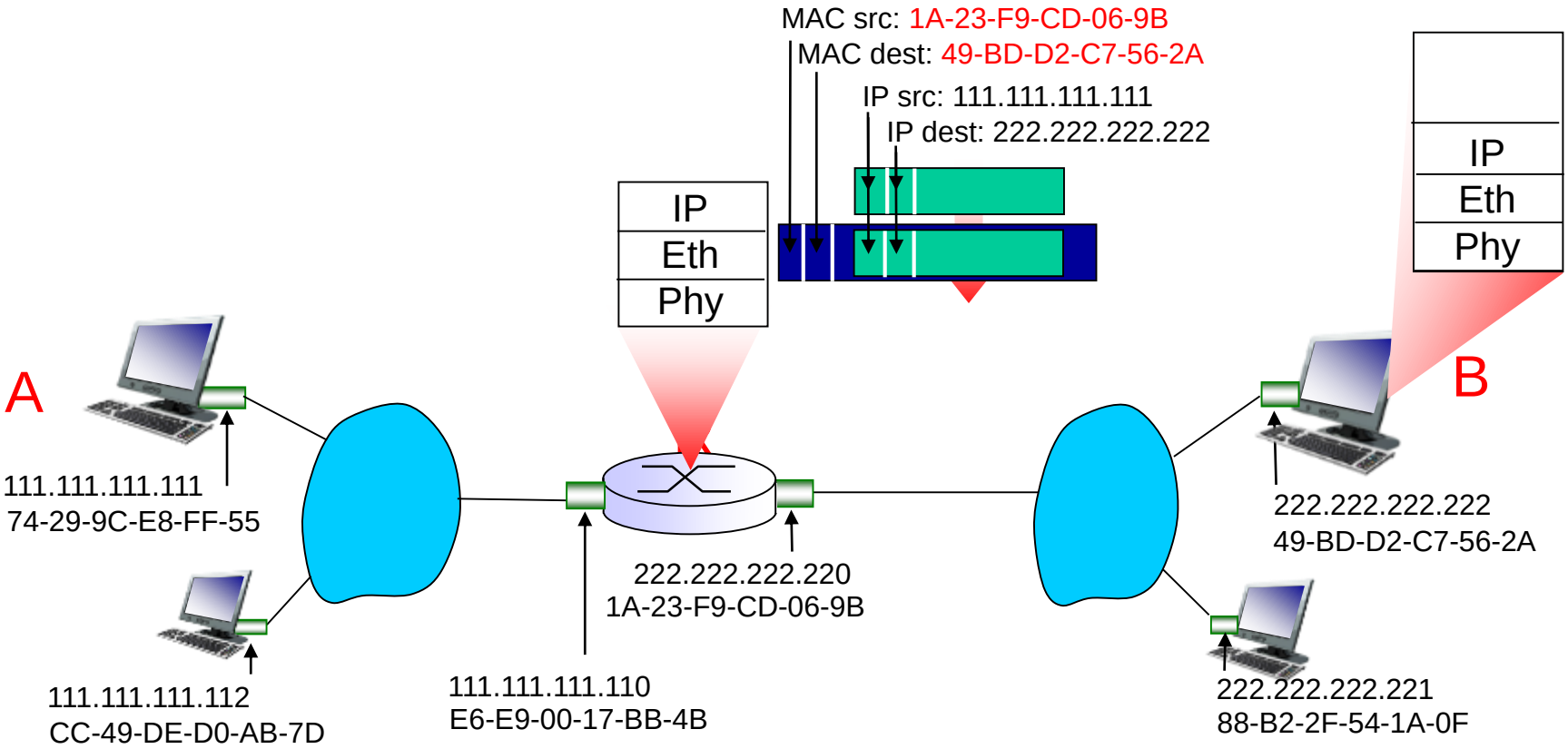
# Addressing: routing to another LAN

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



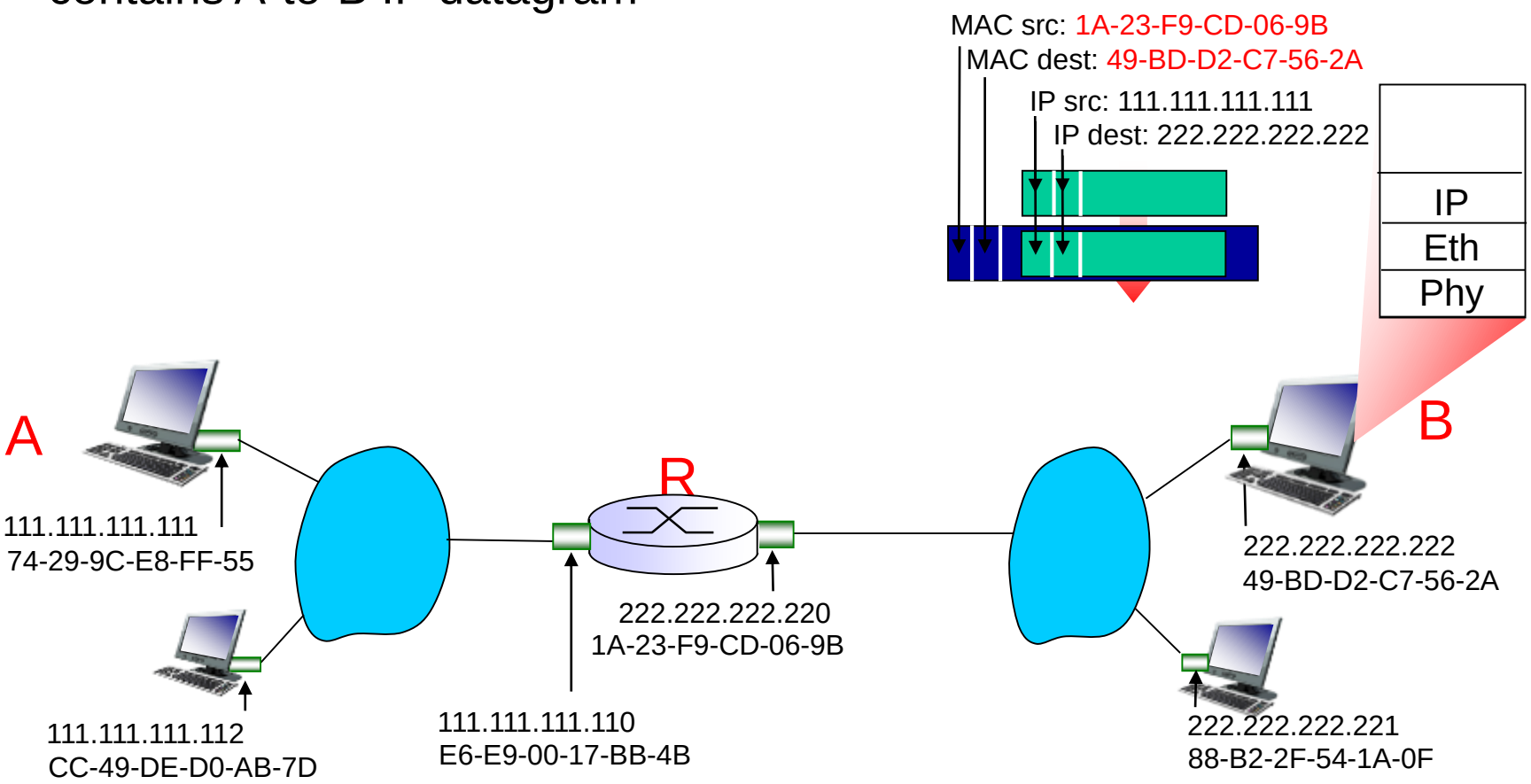
# Addressing: routing to another LAN

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




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

# Part II: Criticize it!

# Challenge: Human errors

## *Complexity: tech-savvy companies struggle to provide reliable networking services*


*We discovered a misconfiguration on this pair of switches that caused what's called a "bridge loop" in the network.*

*A network change was […] executed incorrectly […] more "stuck" volumes and added more requests to the re-mirroring storm*



*Service outage was due to a series of internal network events that corrupted router data tables*

*Experienced a network connectivity issue […] interrupted the airline's flight departures, airport processing and reservations systems*


**Source: Talk by Nate Foster at DSDN Workshop**

# Challenge: Lack of Good Tools

Example: Outage of a data center of a Wall Street investment bank: lost revenue measured in USD $10^6$ / min!

Quickly, assembled emergency team:

**The compute team:** quickly came armed with reams of logs, showing how and when the applications failed, and had already written experiments to reproduce and isolate the error, along with candidate prototype programs to workaround the failure.

**The storage team:** similarly equipped, showing which file system logs were affected, and already progressing with workaround programs.

**The networking team:** All the networking team had were two tools invented over twenty years ago [*ping* and *traceroute*] to merely test end-to-end connectivity. Neither tool could reveal problems with the switches, the congestion experienced by individual packets, or provide any means to create experiments to identify, quarantine and resolve the problem.

**Source: «The world's fastest and most programmable networks» White Paper Barefoot Networks**

# Challenge: Lack of Good Tools

Example: Outage of a data center of a Wall Street investment bank: lost revenue measured in US

Guess who got blamed?

Quickly, assembled emerg

**The compute team:** quickly came armed with reams of logs, showing how and when the applications failed, and had already written experiments to reproduce and isolate the error, along with candidate prototype programs to workaround the failure.

**The storage team:** similarly equipped, showing which file system logs were affected, and already progressing with workaround programs.

**The networking team:** All the networking team had were two tools invented over twenty years ago [*ping* and *traceroute*] to merely test end-to-end connectivity. Neither tool could reveal problems with the switches, the congestion experienced by individual packets, or provide any means to create experiments to identify, quarantine and resolve the problem.

**Source: «The world's fastest and most programmable networks» White Paper Barefoot Networks**

# Challenge: Security

**The Internet on first sight:**
- Monumental
- Passed the “Test-of-Time”
- Should not and cannot be changed

**The Internet on second sight:**
- Antique
- Britle
- Successful attacks more and more frequent (e.g., based on IoT)

**Source: Slide by Adrian Perrig**

# In the news: DoS, security of DNS/BGP/…

Security / #CyberSecurity

SEP 25, 2016 @ 10:00 AM **41,011** VIEWS

## How Hacked Cameras Are Helping Launch The Biggest Attacks The Internet Has Ever Seen

**Thomas Fox-Brewster,** FORBES STAFF

*I cover crime, privacy and security in digital and physical forms.* **FULL BIO**

Recent "Attack of the (Internet-)Things"
(aka babyphone attack)

# Challenges in More Depth

- IP address space depletion
- Inflexible routing
- Fast failover
- Slow innovation
- Network virtualization

# Challenges in More Depth

Reasons why Google (and others) moved to Software-Defined Networks (SDN):

- Traffic engineering flexibilities
  - Tailor to traffic type!
  - E.g. Google Docs vs datacenter synchronization
- Faster failover

# Challenges in More Depth

- **IP address space depletion**
- Inflexible routing
- Fast failover
- Slow innovation
- Network virtualization

# NAT: network address translation

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

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

# NAT: network address translation

*implementation*: NAT router must:

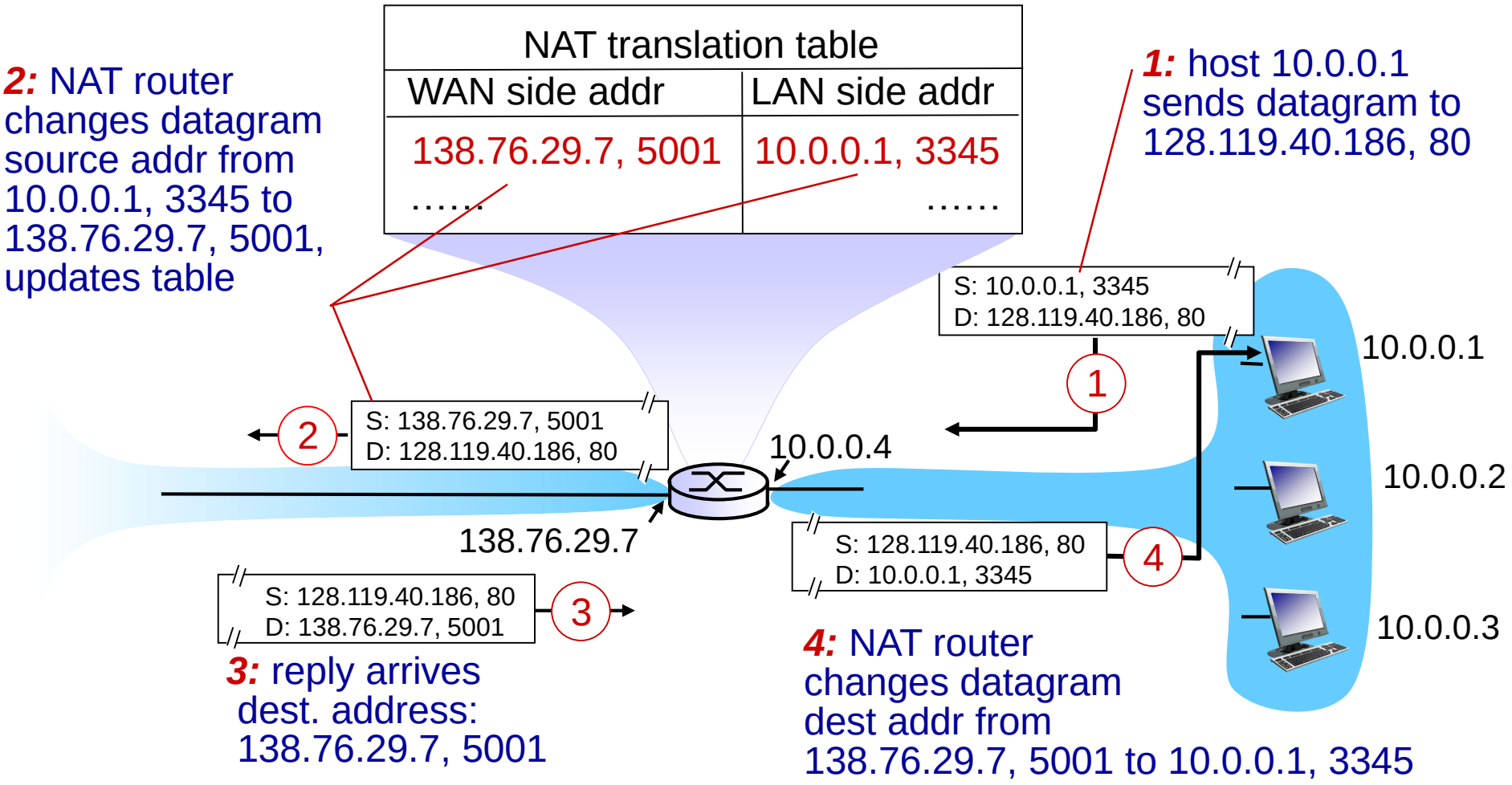
- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)

  . . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr

- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair

- *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

# NAT: network address translation

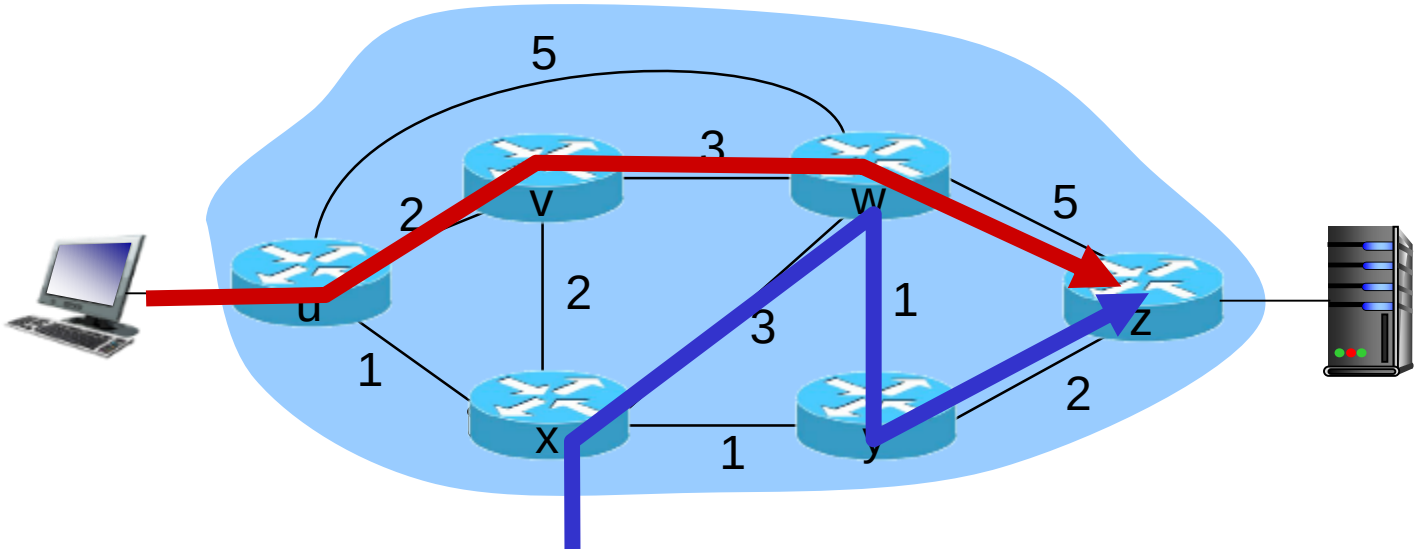



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

# Challenges in More Depth

- IP address space depletion
- **Inflexible routing**
- Fast failover
- Slow innovation
- Network virtualization

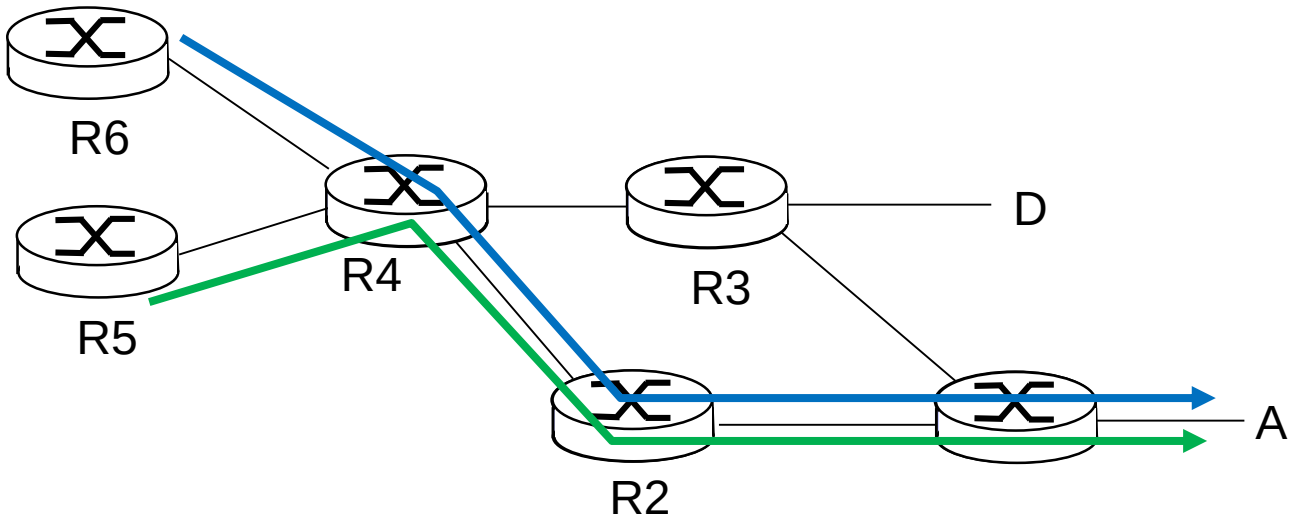
# Traffic engineering: difficult, traditionally!




*Q:* what if w wants to route blue and red traffic differently?

*A:* can't do it (with destination based forwarding, and LS, DV routing)

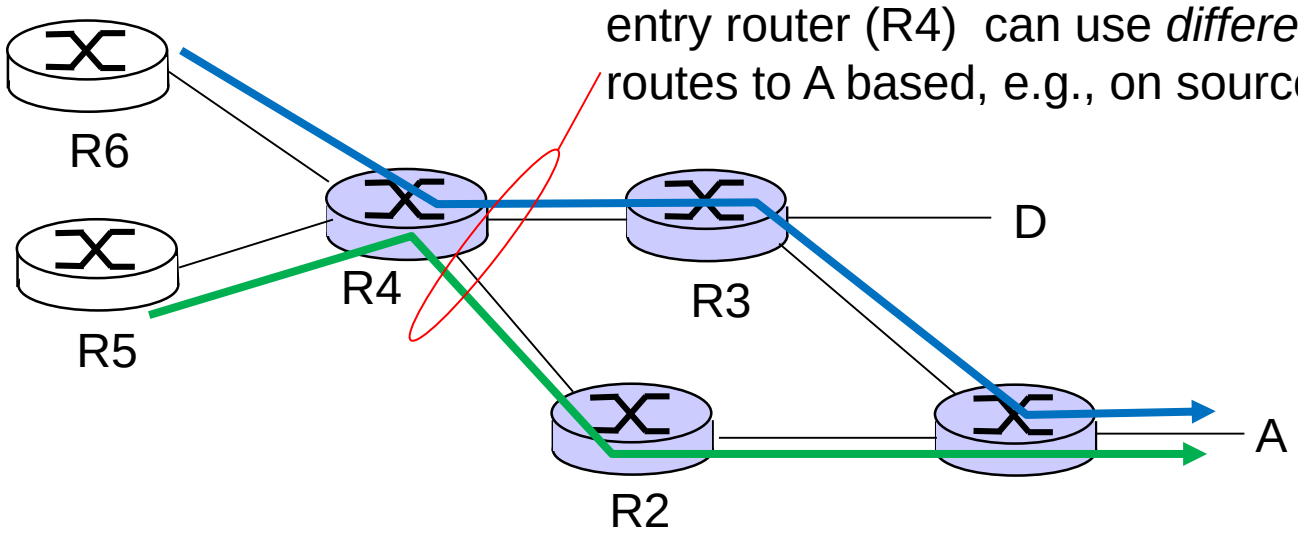
# Traffic engineering: difficult, traditionally!




- *IP routing:* path to *destination determined by destination address alone*

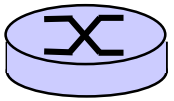

*IP router*

# MPLS versus IP paths




- *IP routing: path to destination determined by destination address alone*
- *MPLS routing:* path to destination can be based on source *and* destination address

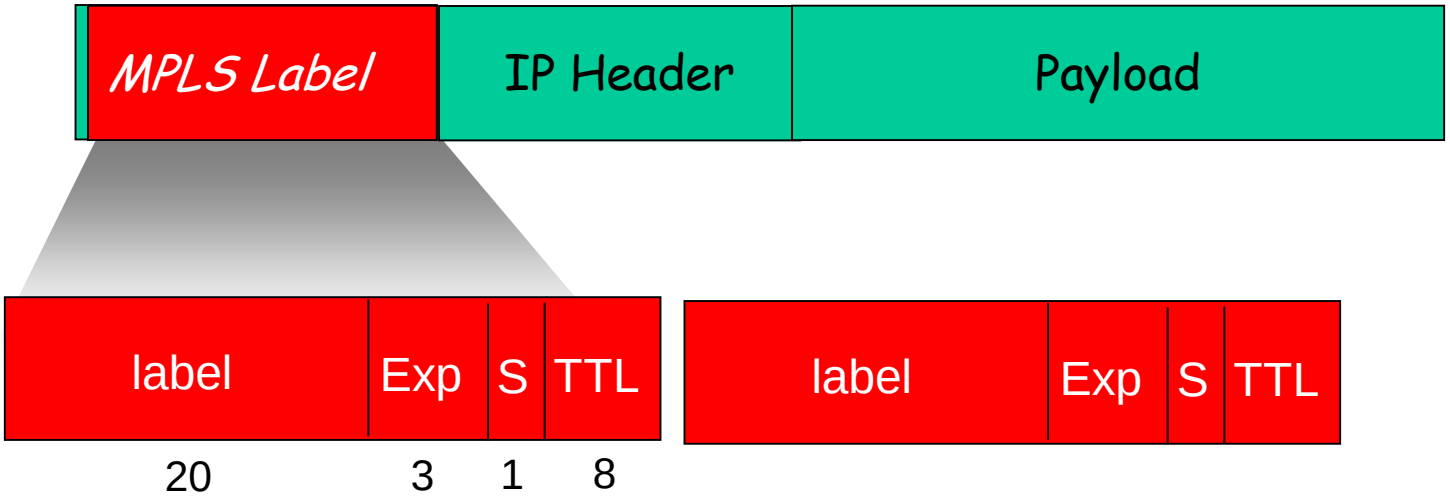
*IP-only router*



*MPLS and IP router*

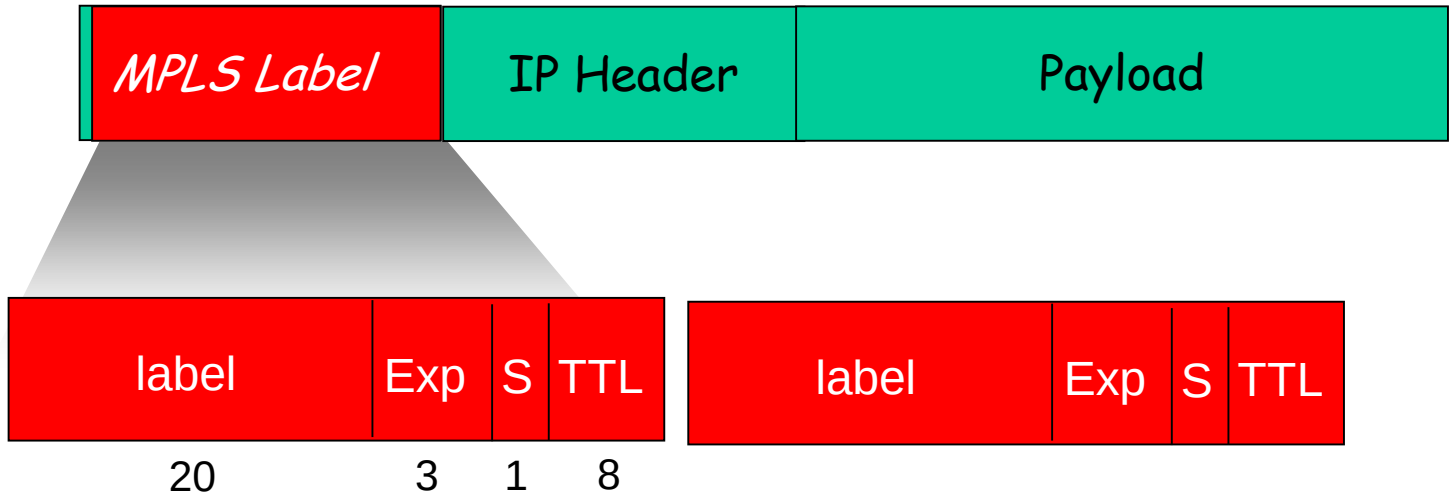
# Solution: MPLS

- Multi Protocol 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!



# A Layer 2.5 Protocol!

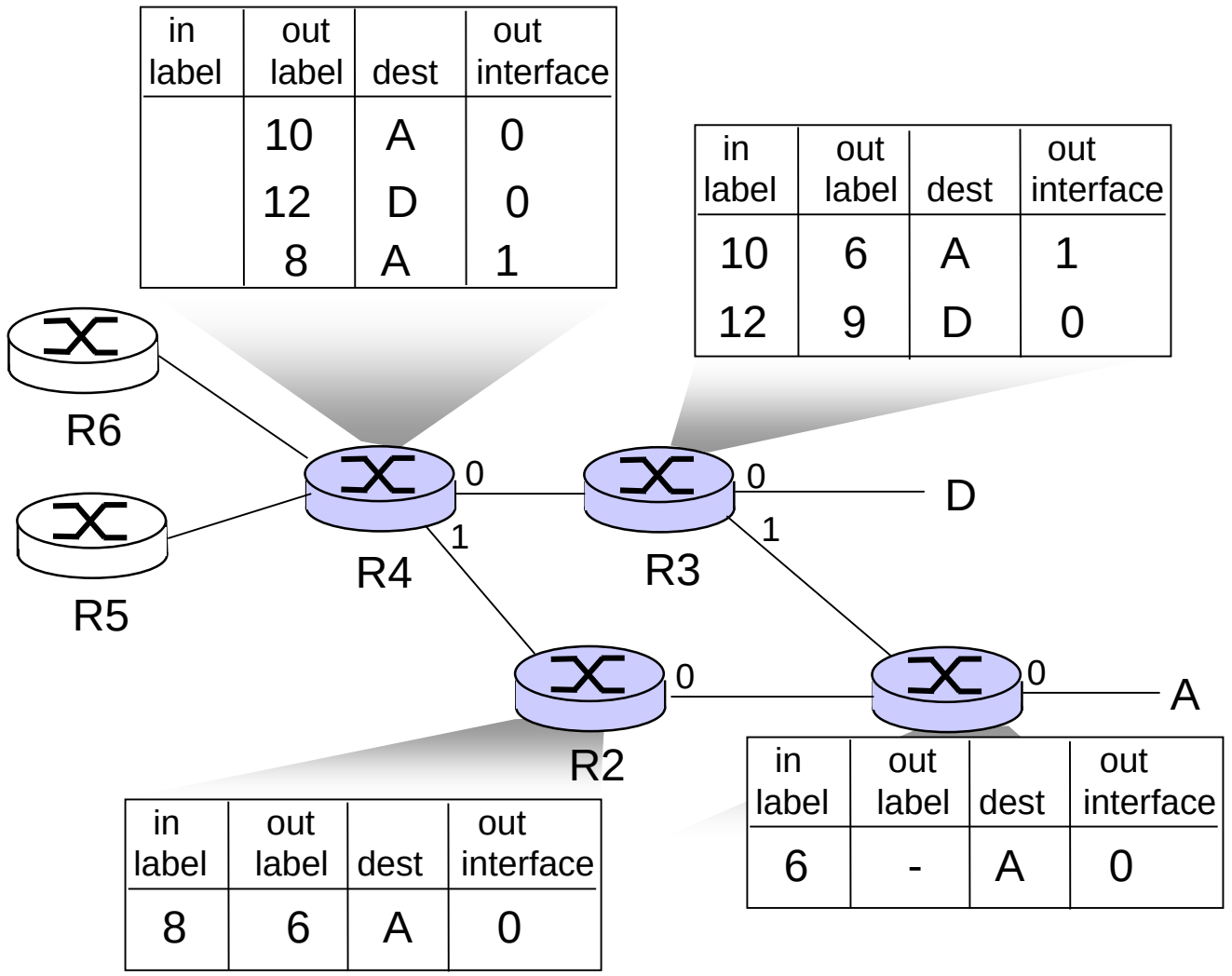
- Insert label between Layer 2 and Layer 3 header
- Fields
  - 20 bit routing label
  - 3 bit “Exp” field carries packet queuing priority for Class of Service
  - 1 bit “Bottom of Stack” field (labels can be stacked)
  - 8 bit Time To Live field
- Can be “stacked”



# 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 forwarding tables



# Challenges in More Depth

- IP address space depletion
- Inflexible routing
- **Fast failover**
- Slow innovation
- Network virtualization

# Traditional Failover: Slow!

*Bellman-Ford equation (dynamic programming)*

let

$d_x(y)$ := cost of least-cost path from x to y

then

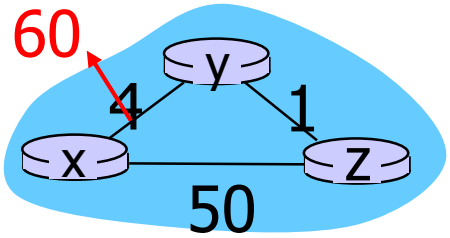
$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

cost from neighbor v to destination y

cost to neighbor v

*min* taken over all neighbors v of x

# Slow!




What will happen?

**node y table**

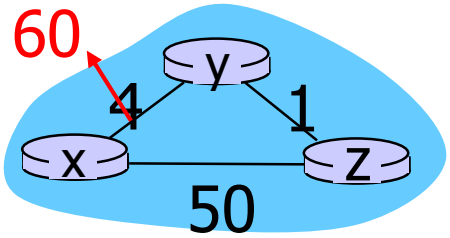
| from \ cost to | x | y | z |
|---|---|---|---|
| x | | | |
| y | 4 | 0 | 1 |
| z | 5 | 1 | 0 |

**node z table**

| from \ cost to | x | y | z |
|---|---|---|---|
| x | | | |
| y | 4 | 0 | 1 |
| z | 5 | 1 | 0 |

time

# Slow!




**node y table**

cost to

| from | x | y | z |
|---|---|---|---|
| x | | | |
| y | ~~4~~ 6 | 0 | 1 |
| z | 5 | 1 | 0 |

Why?!

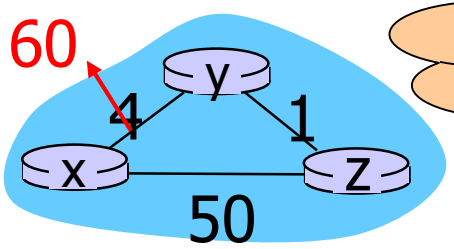
**node z table**

cost to

| from | x | y | z |
|---|---|---|---|
| x | | | |
| y | 4 | 0 | 1 |
| z | 5 | 1 | 0 |

time

# Slow!




Increases 2 by 2: route via z, which then increases too. Same path!

*z still stores old value: can reach x at cost 5. So:*

$D_y(x) = \min\{c(y,x) + D_x(x),\ c(y,z) + D_z(x)\}$
$= \min\{60 + 0\ ,\ 1 + 5\} = 6$

$D_y(x) = \min\{c(y,x) + D_x(x),$
$c(y,z) + D_z(x)\}$
$= \min\{60 + 0\ ,\ 1 + 7\} = 8$

**node y table**

*Why?!*

cost to

| from | x | y | z |
|---|---|---|---|
| x | | | |
| y | ~~4~~ 6 | 0 | 1 |
| z | 5 | 1 | 0 |

cost to

| from | x | y | z |
|---|---|---|---|
| x | | | |
| y | 6 | 0 | 1 |
| z | 5 | 1 | 0 |

cost to

| from | x | y | z |
|---|---|---|---|
| x | | | |
| y | ~~6~~ 8 | 0 | 1 |
| z | 7 | 1 | 0 |

**node z table**

cost to

| from | x | y | z |
|---|---|---|---|
| x | | | |
| y | 4 | 0 | 1 |
| z | 5 | 1 | 0 |

cost to

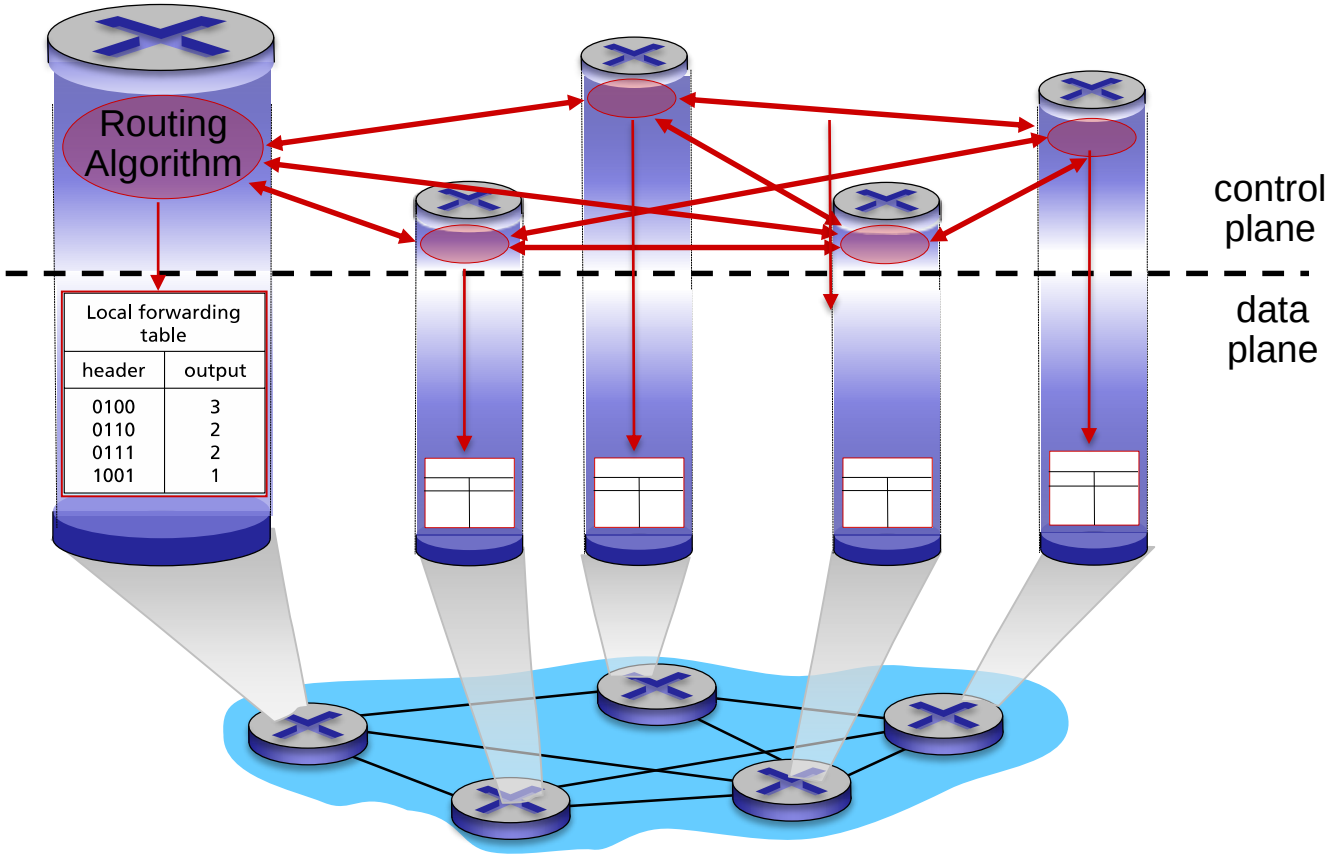
| from | x | y | z |
|---|---|---|---|
| x | | | |
| y | 6 | 0 | 1 |
| z | ~~5~~ 7 | 1 | 0 |

cost to

| from | x | y | z |
|---|---|---|---|
| x | | | |
| y | 6 | 0 | 1 |
| z | 7 | 1 | 0 |

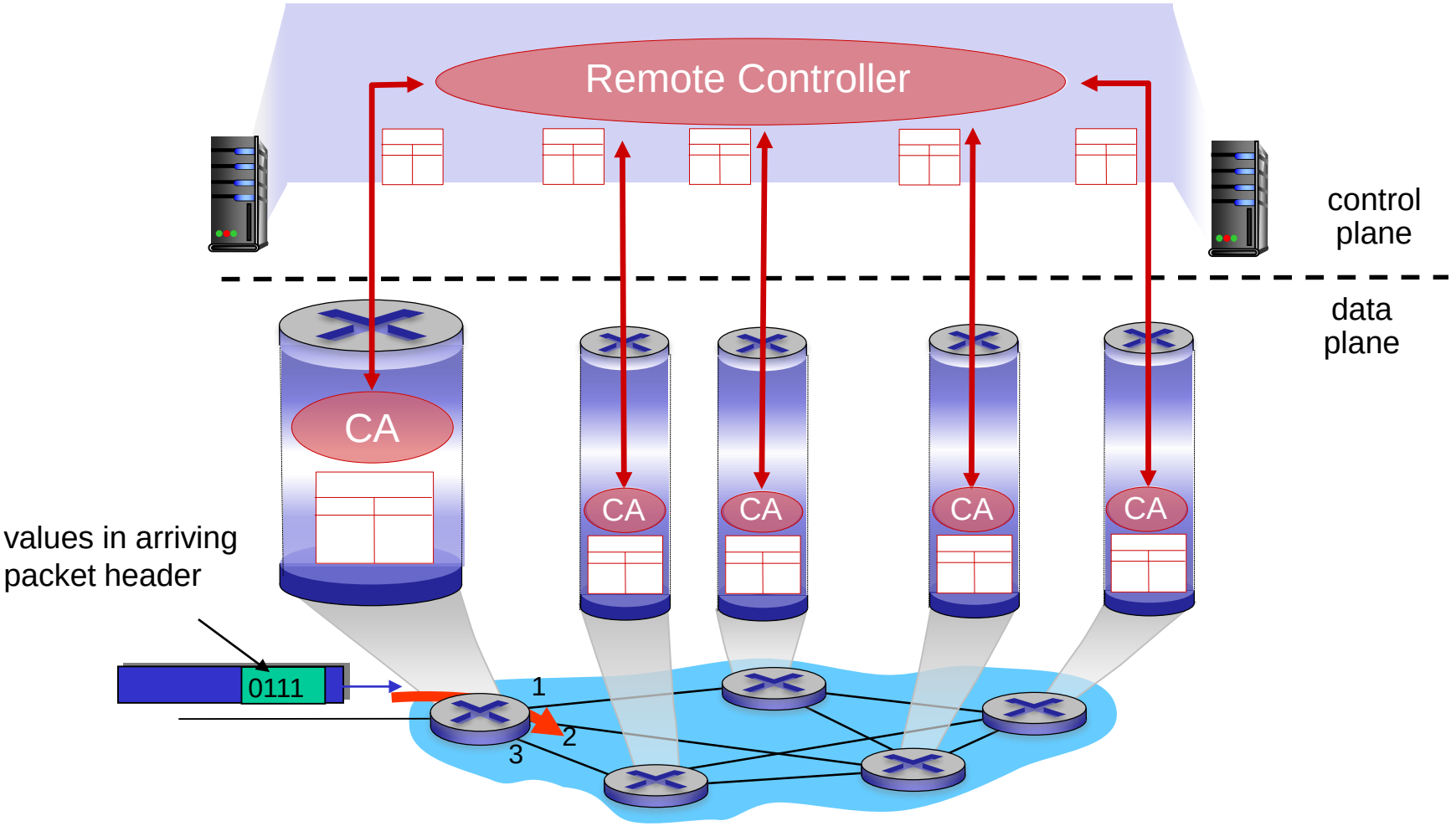
time

# Problem: decentralized control plane



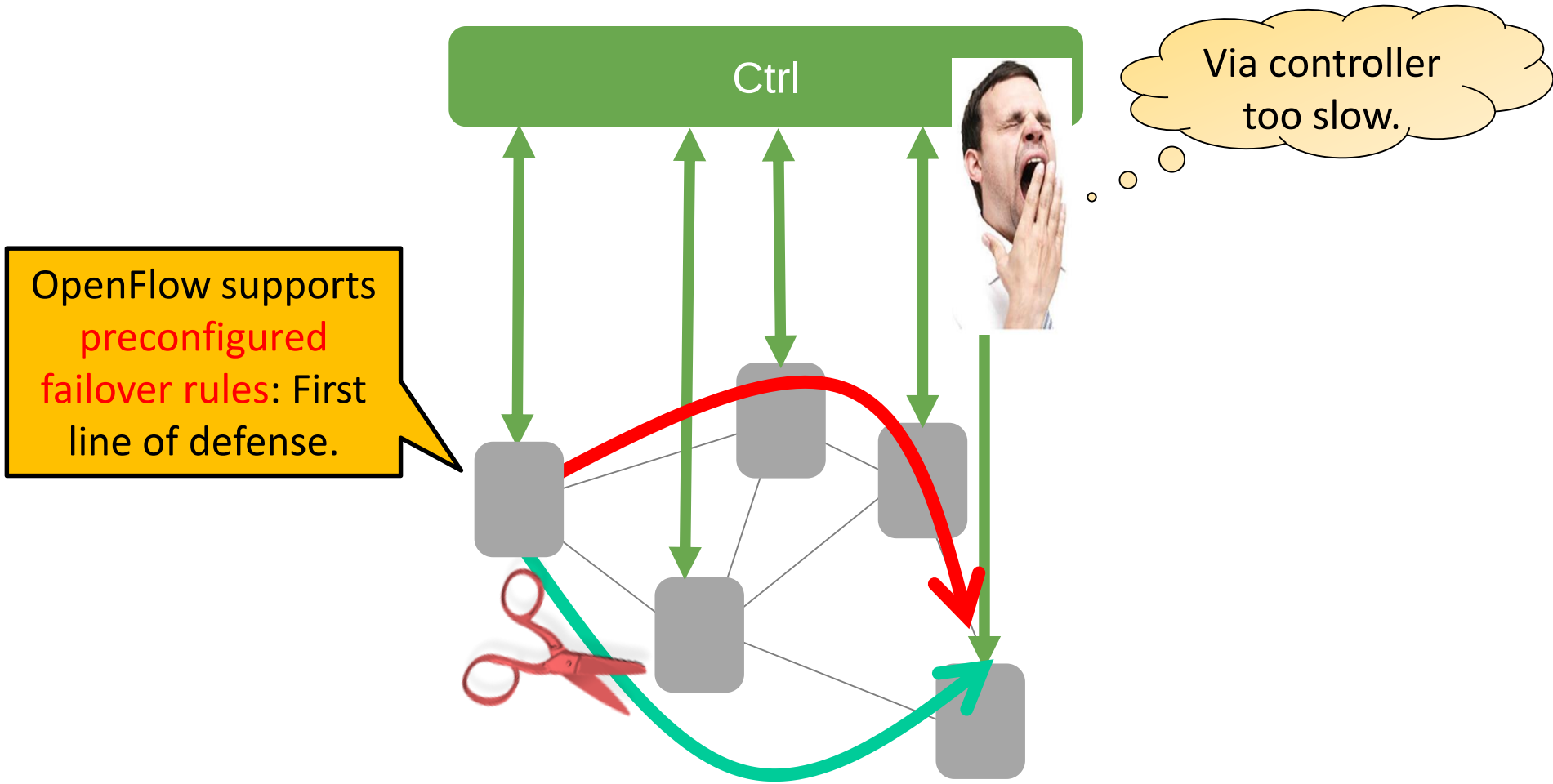
# Solution 1: SDN

A distinct logically centralized (typically remote) controller (servers running software) interacts with local control agents (CAs)



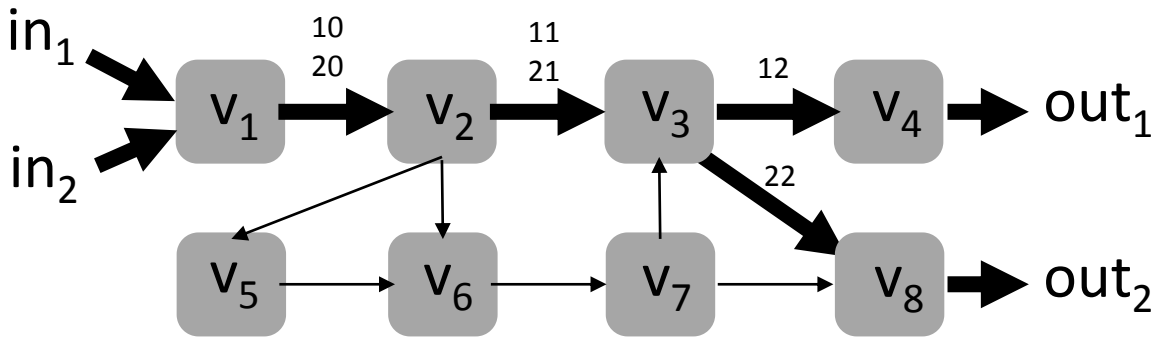
# Solution 2: Local Fast Failover

Supported by MPLS and SDN (controller not involved!)



# Complexity of What-if Analysis

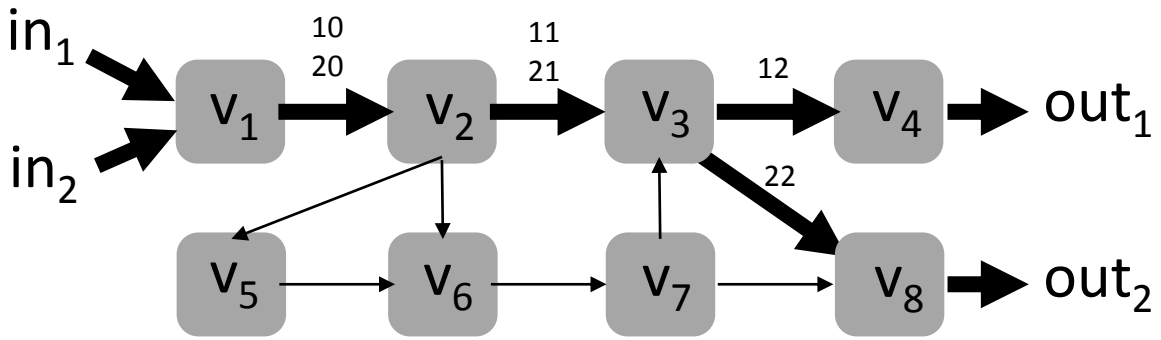
- Simple forwarding based on a label stack
  - Idea: forward according to top label
  - Usually, top label swapped at each hop




Default routing of two flows

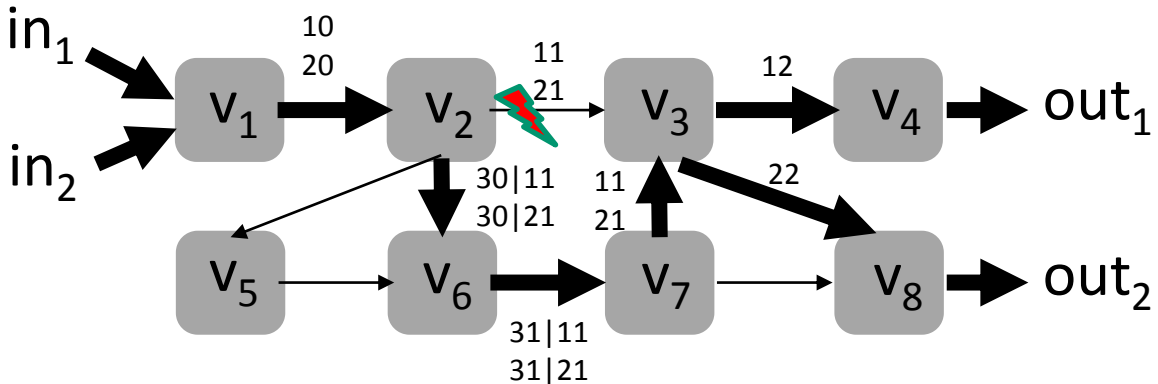
# Complexity of What-if Analysis

- Simple forwarding based on a label stack
  - Idea: forward according to top label
  - Usually, top label swapped at each hop




Default routing of two flows

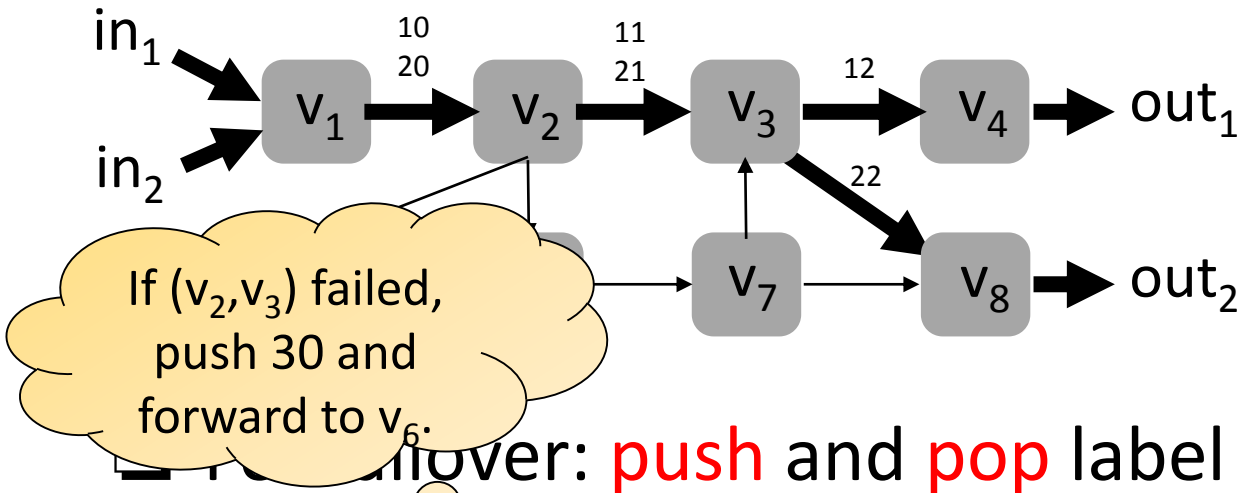
- For failover: push and pop label




One failure: push 30: route around ($v_2$,$v_3$)

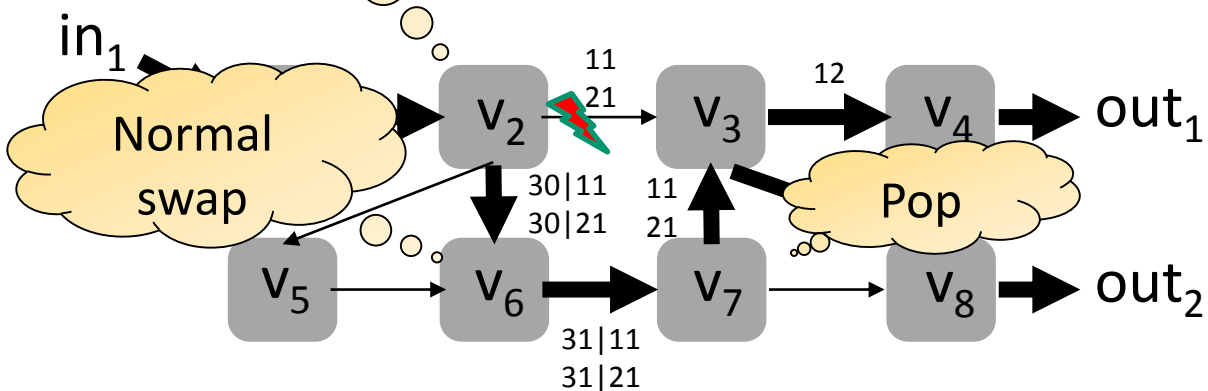
# Complexity of What-if Analysis

- Simple forwarding based on a label stack
  - Idea: forward according to top label
  - Usually, top label swapped at each hop




Default routing of two flows

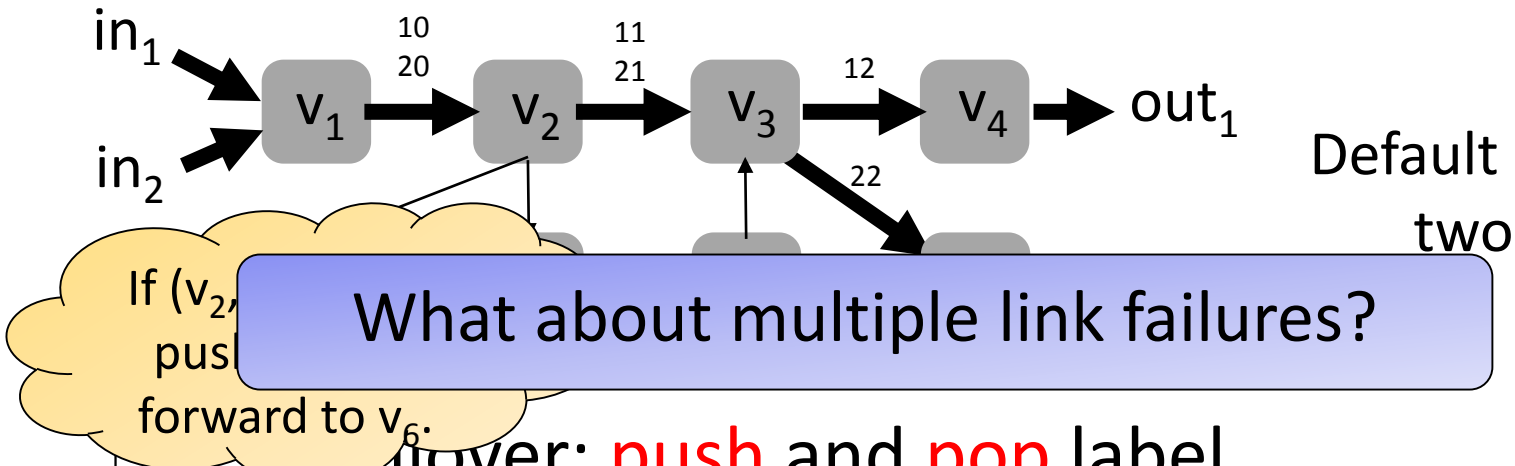
- Fast failover: push and pop label




One failure: push 30: route around $(v_2,v_3)$

# Complexity of What-if Analysis

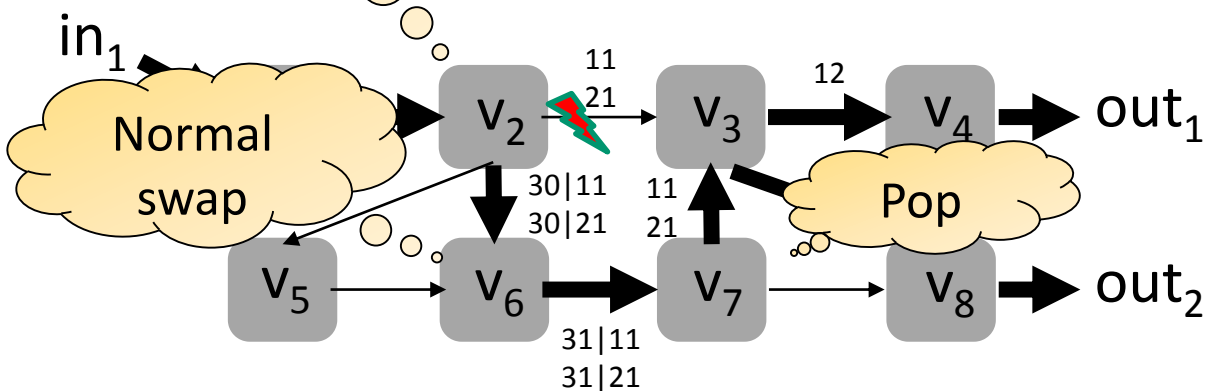
- Simple forwarding based on a label stack
  - Idea: forward according to top label
  - Usually, top label swapped at each hop




Default routing of two flows

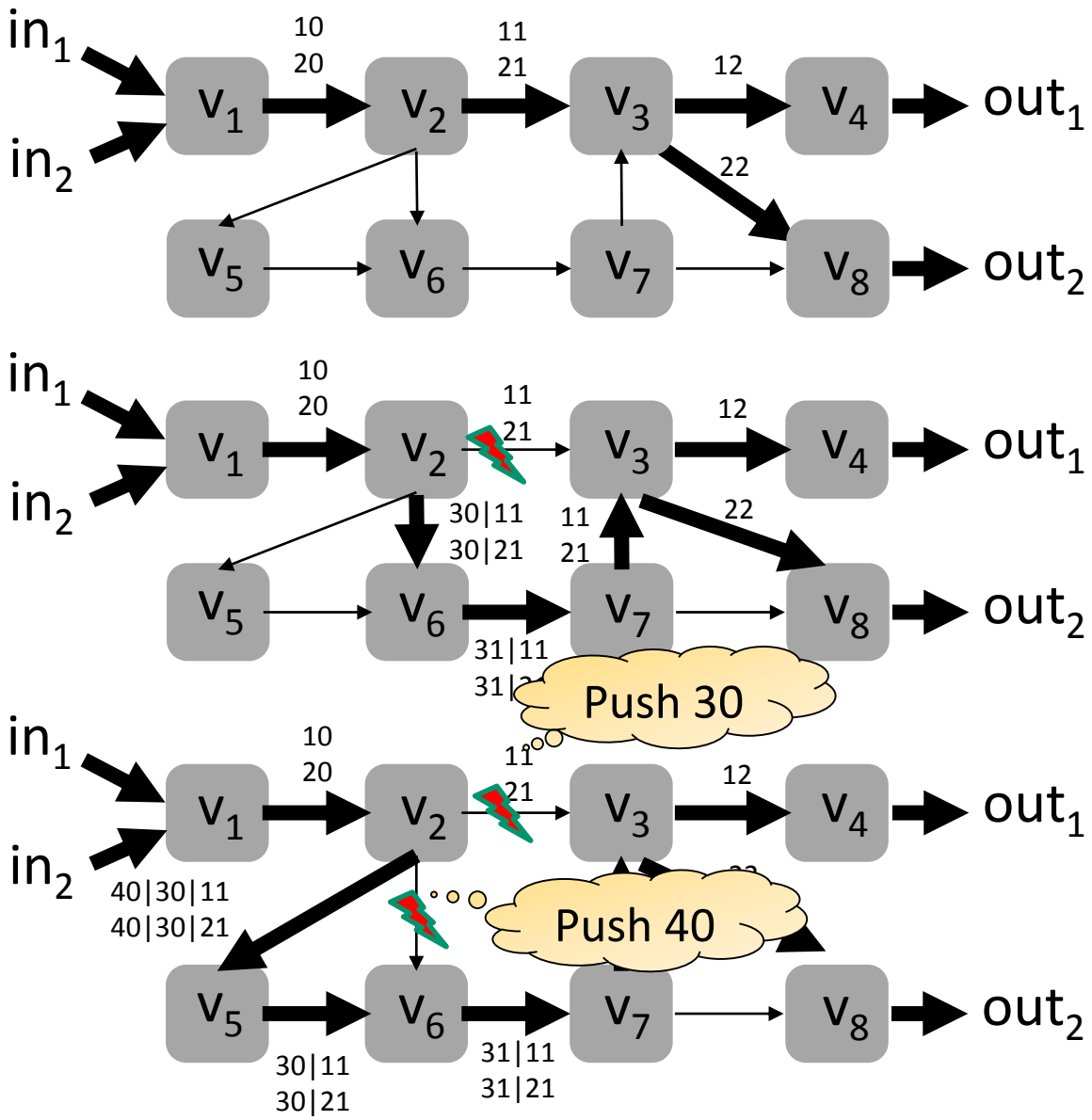
What about multiple link failures?

- Fast failover: push and pop label




One failure: push 30: route around $(v_2,v_3)$

# Push Recursively



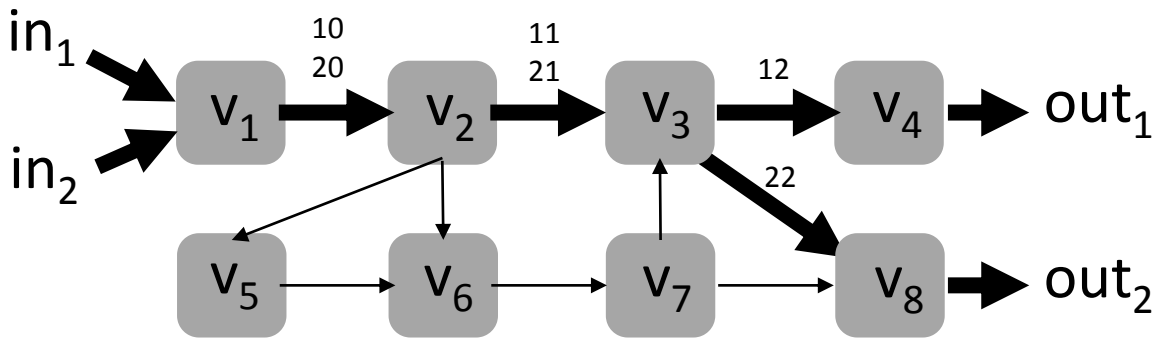

Original Routing

One failure: push 30: route around $(v_2,v_3)$

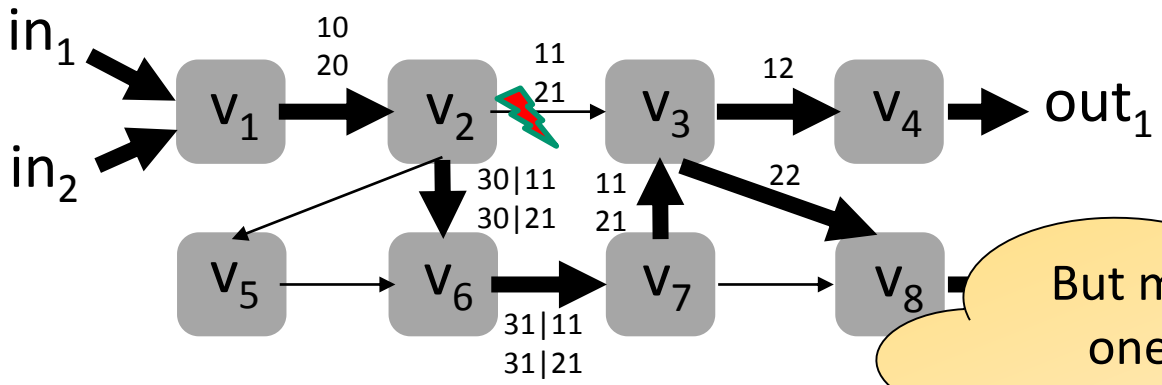
Two failures:
first push 30: route around $(v_2,v_3)$

Recursively push 40: route around $(v_2,v_6)$

# Push Recursively

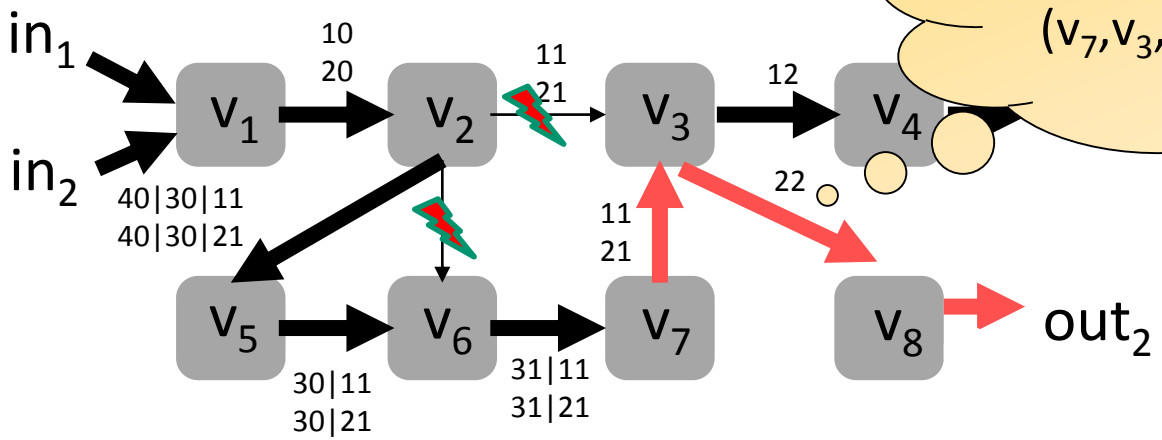



Original Routing




One failure: push 30:

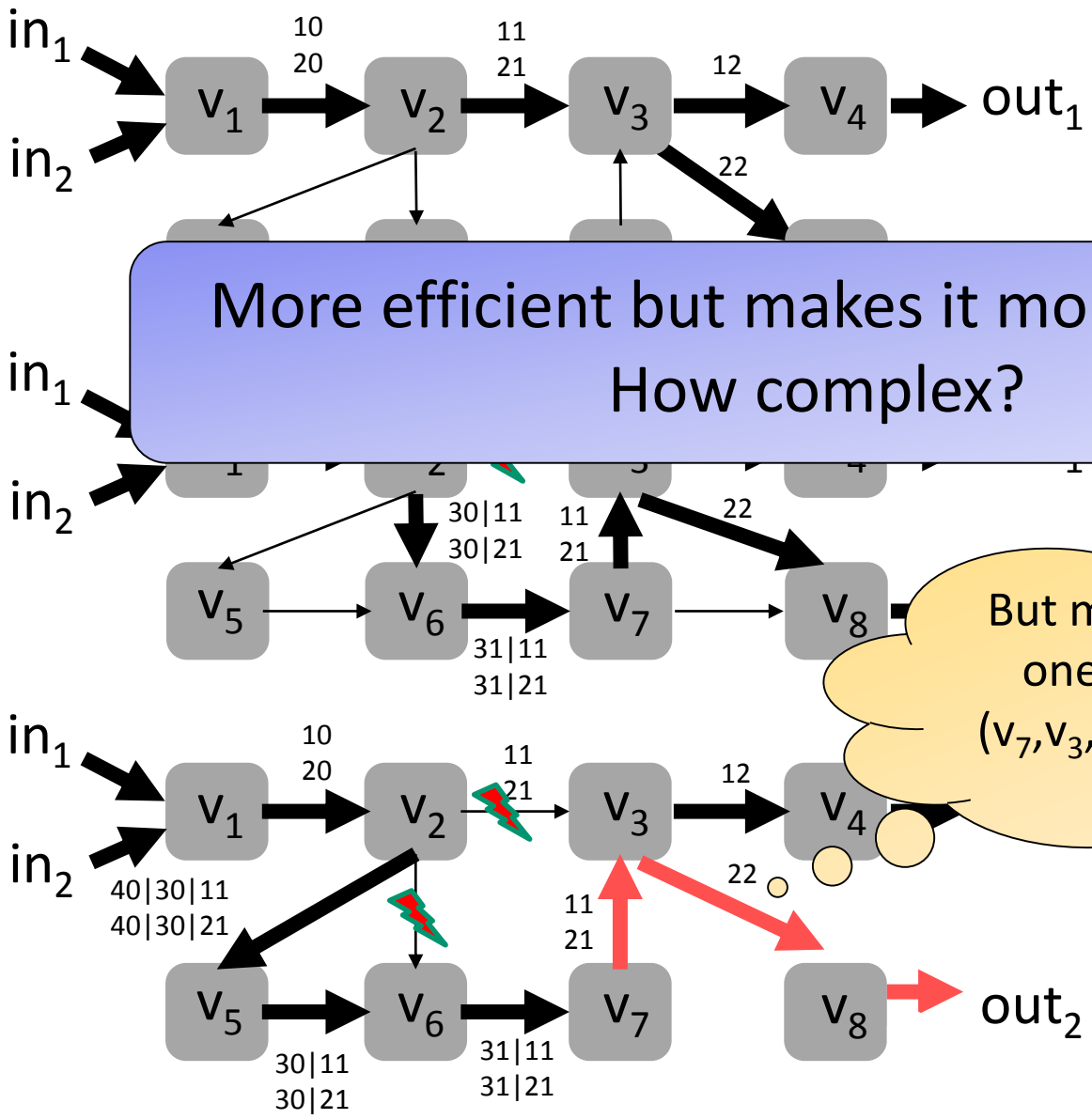
But masking links one-by-one can be inefficient: $(v_7,v_3,v_8)$ could be shortcut to $(v_7,v_8)$.




around $(v_2,v_3)$

Recursively push 40: route around $(v_2,v_6)$

# Push Recursively




More efficient but makes it more complex!
How complex?

But masking links one-by-one can be inefficient: ($v_7$,$v_3$,$v_8$) could be shortcut to ($v_7$,$v_8$).

Original Routing

One failure: push 30: route around ($v_2$,$v_3$)

Recursively push 40: route around ($v_2$,$v_6$)

# Tables

| FT | In-I | In-Label | Out-I | op |
|---|---|---|---|---|
| $\tau_{v_1}$ | $in_1$ | $\perp$ | $(v_1, v_2)$ | $push(10)$ |
| | $in_2$ | $\perp$ | $(v_1, v_2)$ | $push(20)$ |
| $\tau_{v_2}$ | $(v_1, v_2)$ | 10 | $(v_2, v_3)$ | $swap(11)$ |
| | $(v_1, v_2)$ | 20 | $(v_2, v_3)$ | $swap(21)$ |
| $\tau_{v_3}$ | $(v_2, v_3)$ | 11 | $(v_3, v_4)$ | $swap(12)$ |
| | $(v_2, v_3)$ | 21 | $(v_3, v_8)$ | $swap(22)$ |
| | $(v_7, v_3)$ | 11 | $(v_3, v_4)$ | $swap(12)$ |
| | $(v_7, v_3)$ | 21 | $(v_3, v_8)$ | $swap(22)$ |
| $\tau_{v_4}$ | $(v_3, v_4)$ | 12 | $out_1$ | $pop$ |
| $\tau_{v_5}$ | $(v_2, v_5)$ | 40 | $(v_5, v_6)$ | $pop$ |
| $\tau_{v_6}$ | $(v_2, v_6)$ | 30 | $(v_6, v_7)$ | $swap(31)$ |
| | $(v_5, v_6)$ | 30 | $(v_6, v_7)$ | $swap(31)$ |
| | $(v_5, v_6)$ | 61 | $(v_6, v_7)$ | $swap(62)$ |
| | $(v_5, v_6)$ | 71 | $(v_6, v_7)$ | $swap(72)$ |
| $\tau_{v_7}$ | $(v_6, v_7)$ | 31 | $(v_7, v_3)$ | $pop$ |
| | $(v_6, v_7)$ | 62 | $(v_7, v_3)$ | $swap(11)$ |
| | $(v_6, v_7)$ | 72 | $(v_7, v_8)$ | $swap(22)$ |
| $\tau_{v_8}$ | $(v_3, v_8)$ | 22 | $out_2$ | $pop$ |
| | $(v_7, v_8)$ | 22 | $out_2$ | $pop$ |

Flow Table

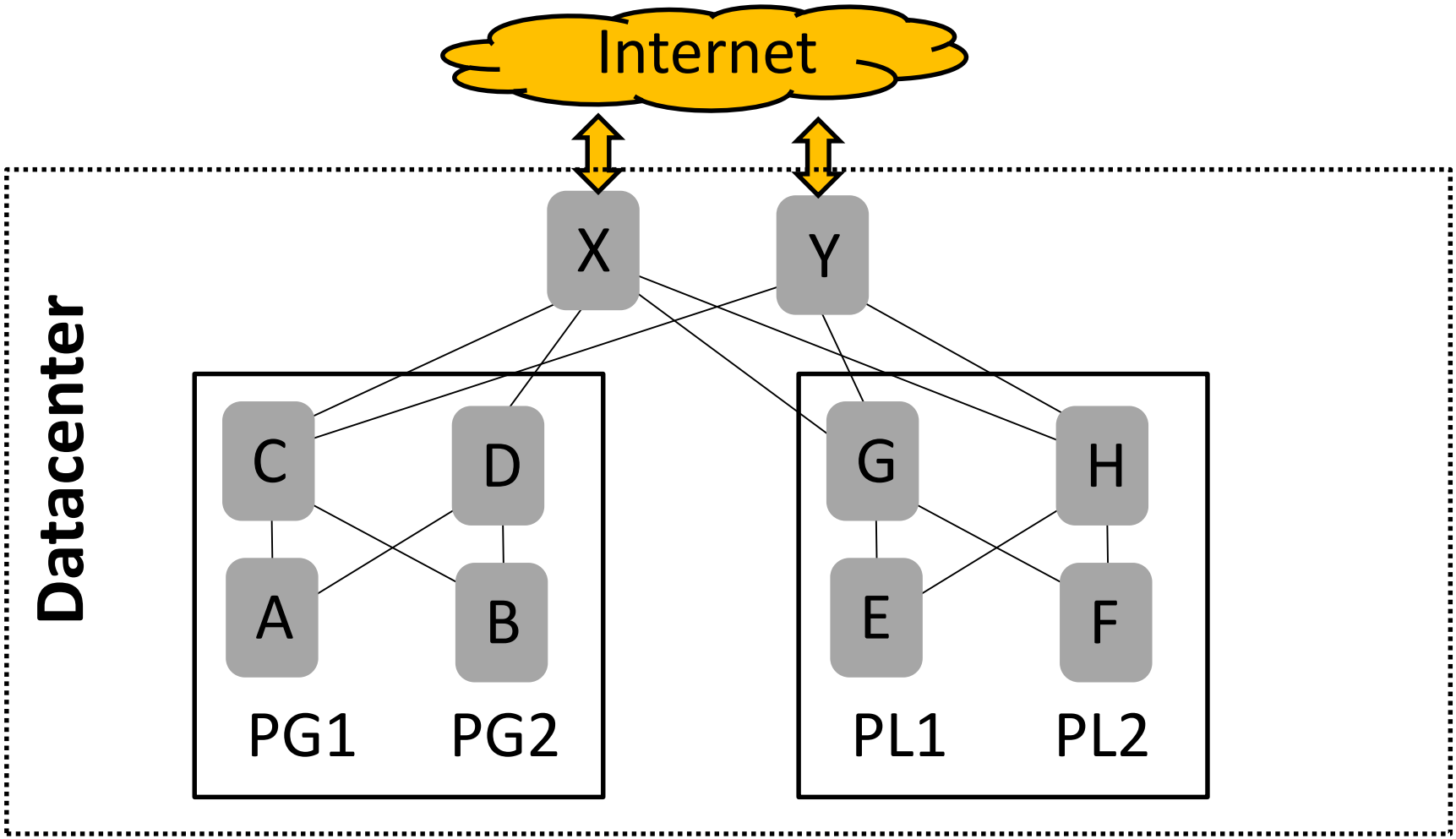
| local FFT | Out-I | In-Label | Out-I | op |
|---|---|---|---|---|
| $\tau_{v_2}$ | $(v_2, v_3)$ | 11 | $(v_2, v_6)$ | $push(30)$ |
| | $(v_2, v_3)$ | 21 | $(v_2, v_6)$ | $push(30)$ |
| | $(v_2, v_6)$ | 30 | $(v_2, v_5)$ | $push(40)$ |

| global FFT | Out-I | In-Label | Out-I | op |
|---|---|---|---|---|
| $\tau'_{v_2}$ | $(v_2, v_3)$ | 11 | $(v_2, v_6)$ | $swap(61)$ |
| | $(v_2, v_3)$ | 21 | $(v_2, v_6)$ | $swap(71)$ |
| | $(v_2, v_6)$ | 61 | $(v_2, v_5)$ | $push(40)$ |
| | $(v_2, v_6)$ | 71 | $(v_2, v_5)$ | $push(40)$ |

Failover Tables

# What-if Analysis Matters

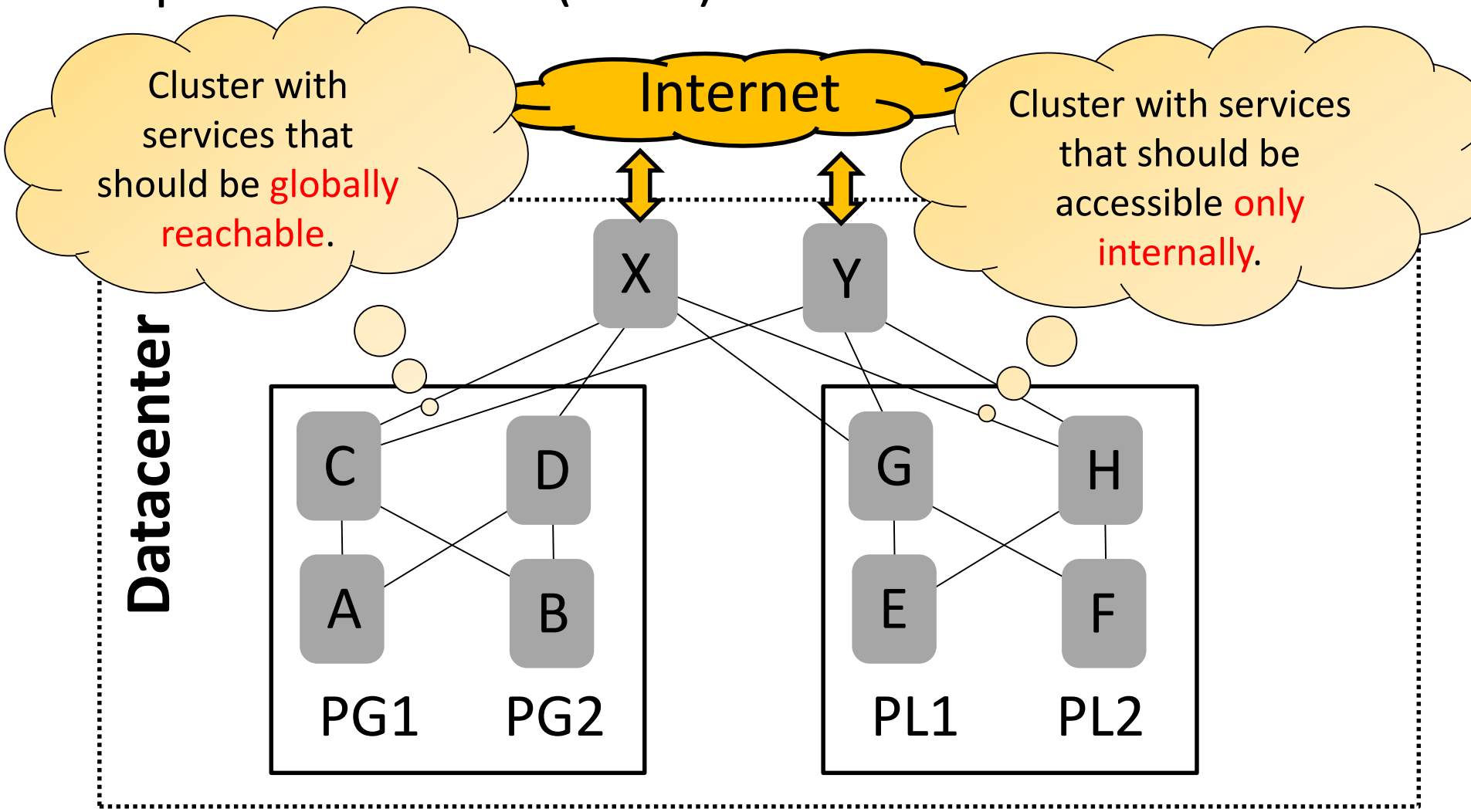
Example: Datacenter (BGP!)*




** Example taken from* **Beckett et al. (SIGCOMM 2016): Don't Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations.**

# What-if Analysis Matters

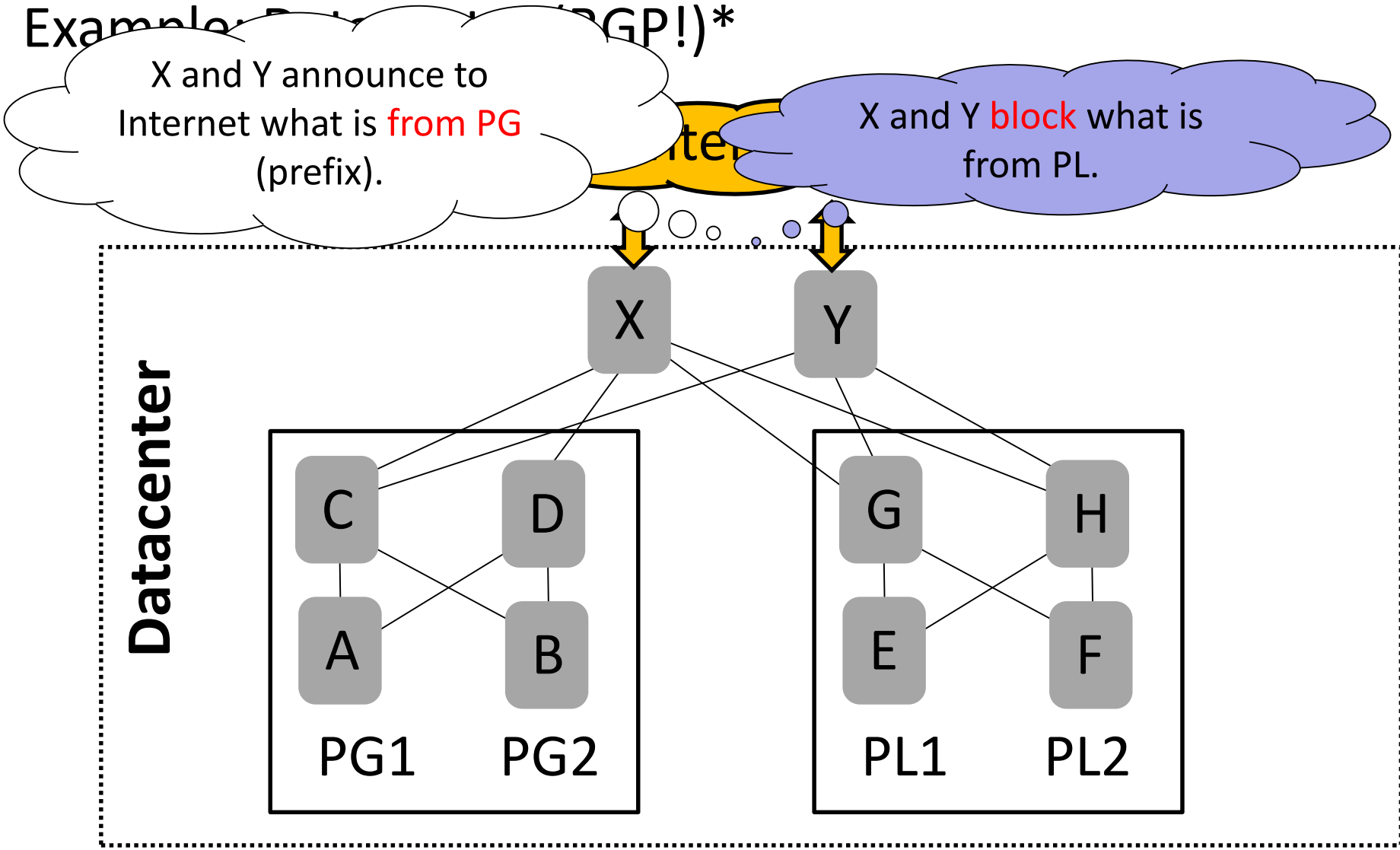
Example: Datacenter (BGP!)*




** Example taken from* **Beckett et al. (SIGCOMM 2016): Don't Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations.**

# What-if Analysis Matters

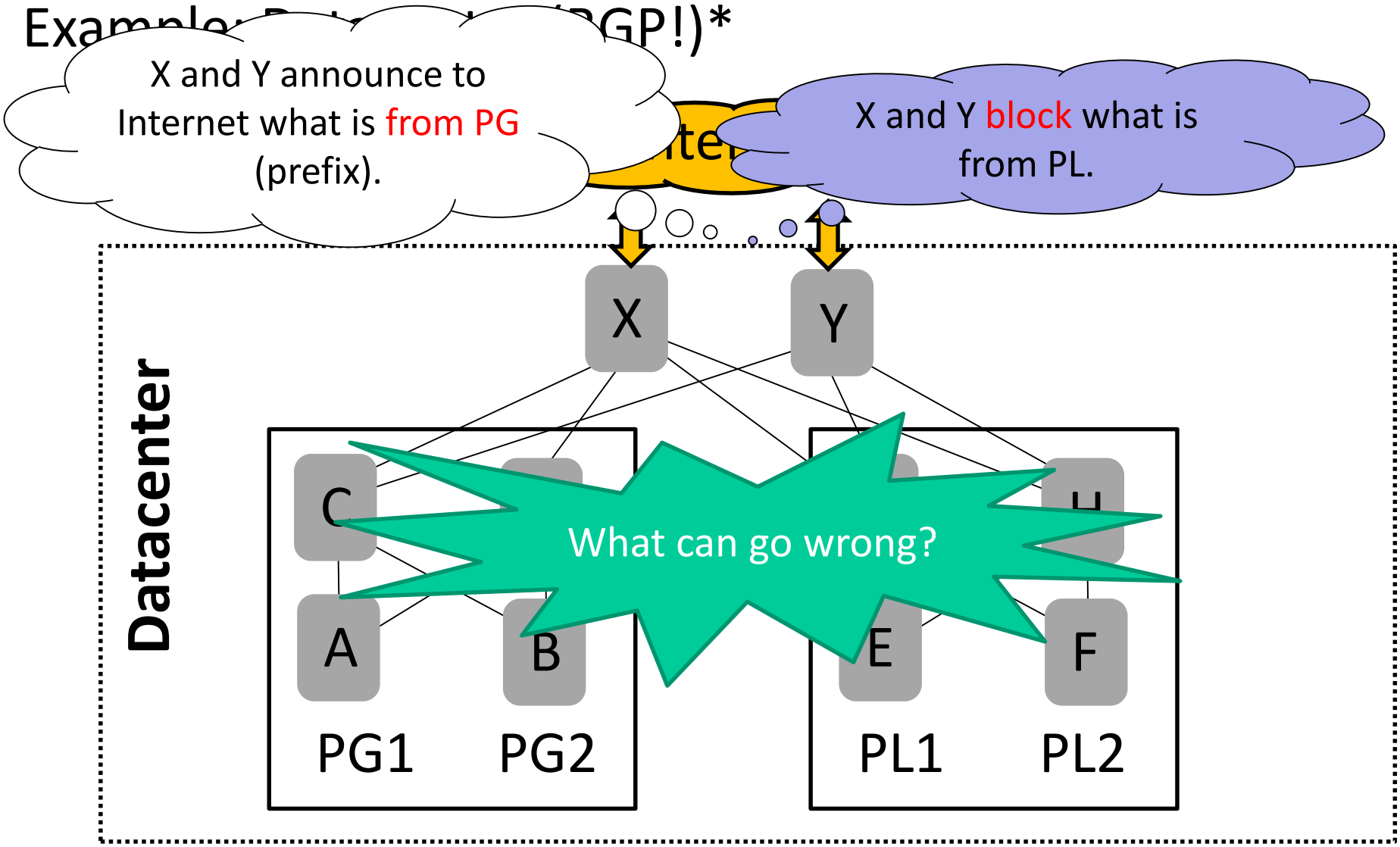
Example: [illegible]BGP!)*




** Example taken from* **Beckett et al. (SIGCOMM 2016): Don't Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations.**

# What-if Analysis Matters

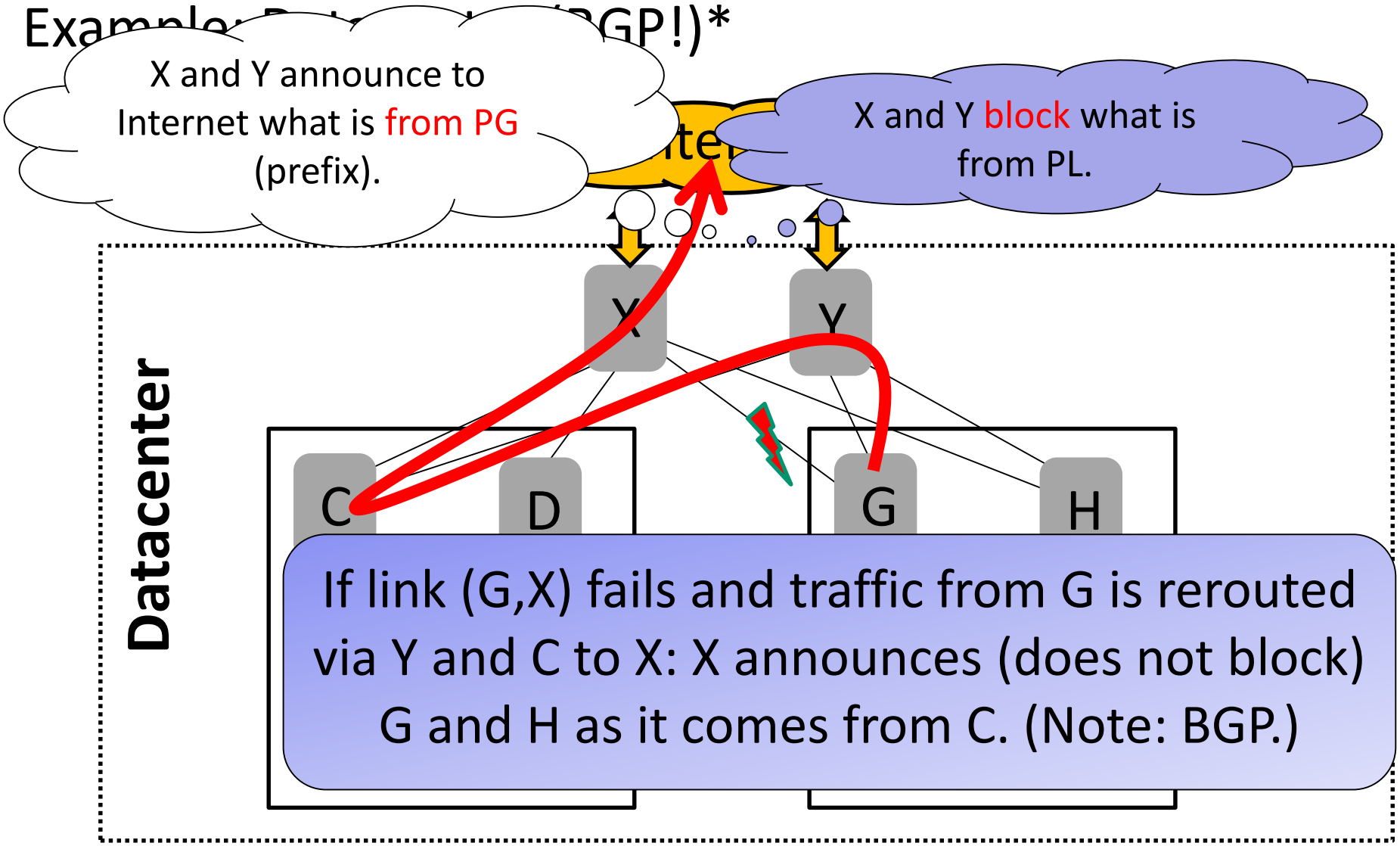
Example: [illegible]BGP!)*




** Example taken from* **Beckett et al. (SIGCOMM 2016): Don't Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations.**

# What-if Analysis Matters

Example: [illegible] BGP!)*



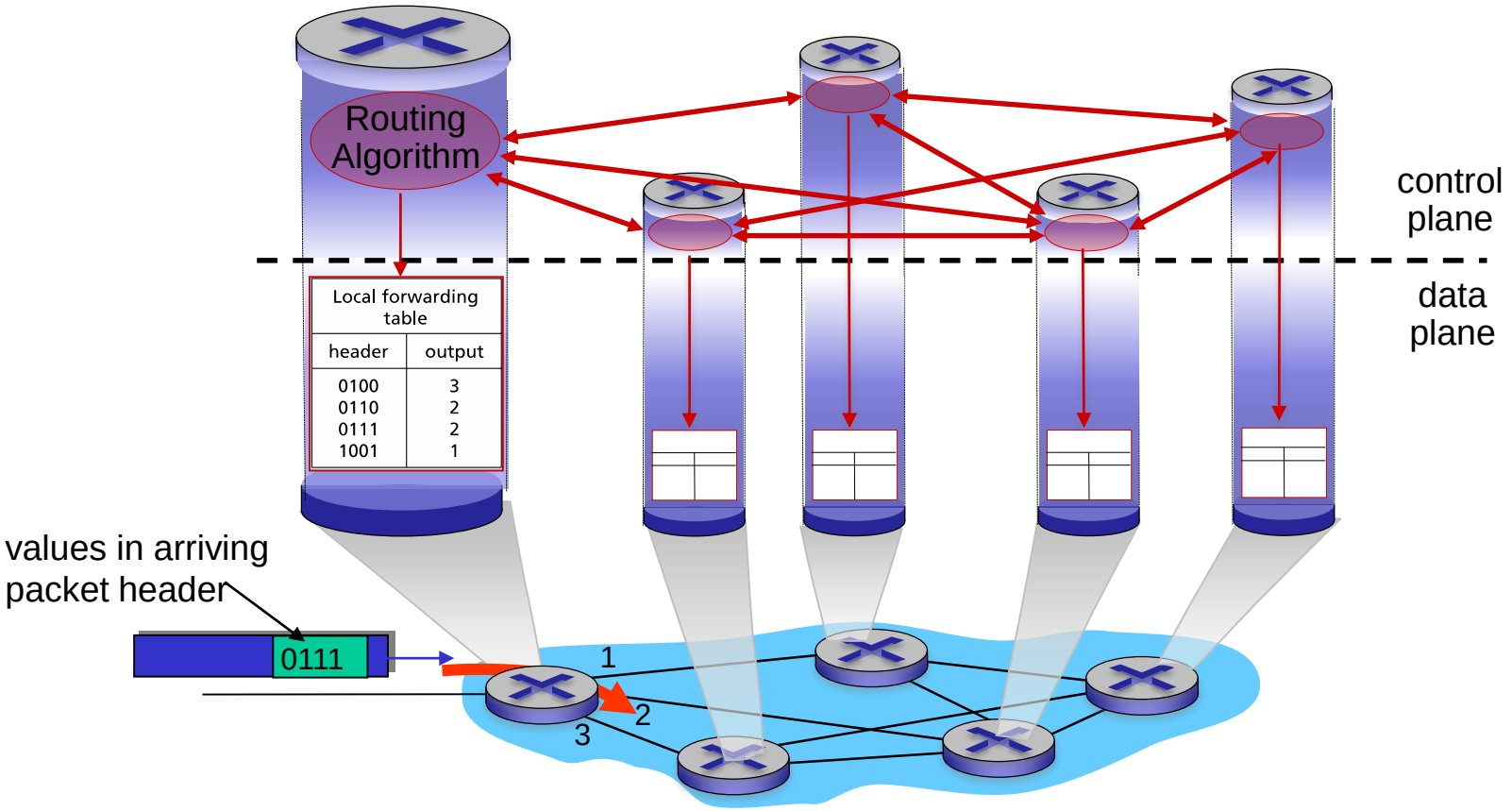

** Example taken from* **Beckett et al. (SIGCOMM 2016): Don't Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations.**

# Challenges in More Depth

- IP address space depletion
- Inflexible routing
- Fast failover
- **Slow innovation**
- Network virtualization

# Traditional networks: Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane



# Drawbacks

- Complex: distributed logic / state (hard to debug!)
- Inflexible: standardized routing protocols only (difficult to innovate Traffic Engineering)
- Slow failover: distributed control protocols are known to re-converge slowly after failures
- Proprietary/blackbox implementations: no open interfaces

Operator says:

**I need extended VTP (VLAN Trunking Protocol) / a 3rd spanport etc. !**

Cisco's answer:

**Buy one of these!**

# Drawbacks

- Complex: distributed logic / state (hard to debug!)
- Inflexible: standardized routing protocols only (difficult to innovate Traffic Engineering)
- Slow failover: distributed control protocols are known to re-converge slowly after failures
- Proprietary/blackbox implementations: no open interfaces

Operator says:

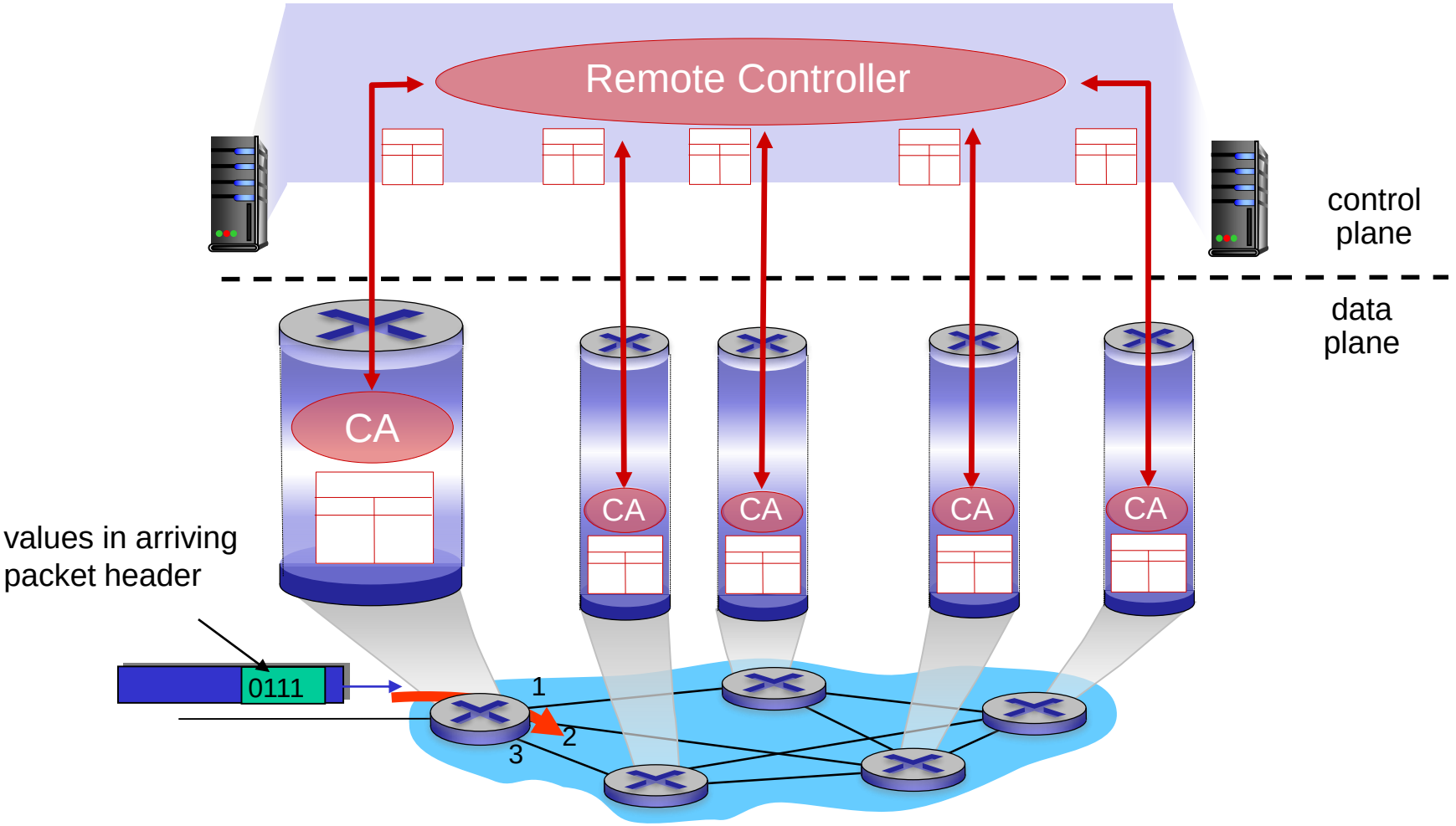

Cisco's answer:

# On a more serious note

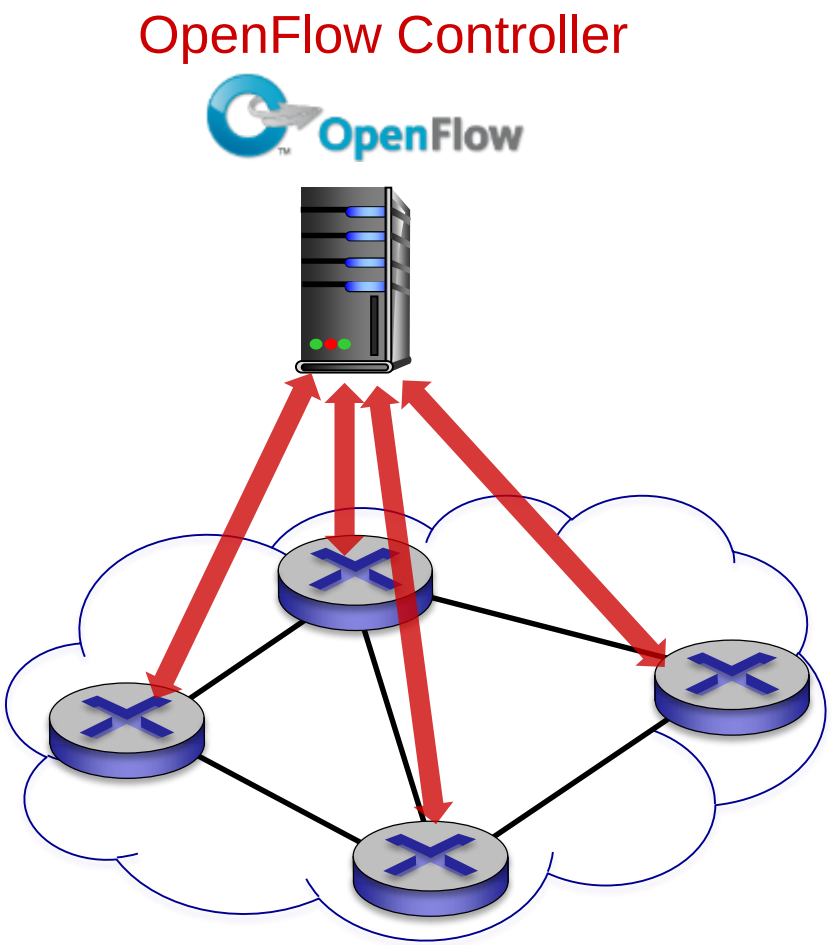
- Closed HW/SW stack hurts innovation
- Researchers can't try their ideas on scale (no, NetFPGA is not 'at scale')
- Data center ops can't differentiate by building customized network

# SDN: logically centralized control plane

A distinct logically centralized (typically remote) controller (servers running software) interacts with local control agents (CAs)

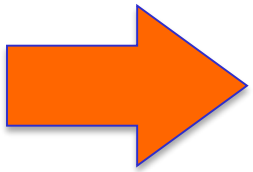


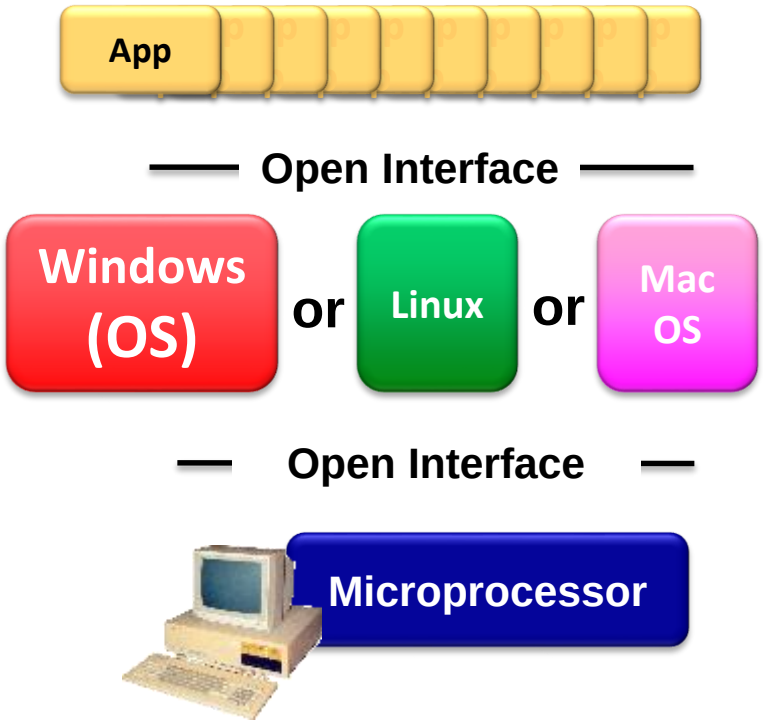
# OpenFlow protocol



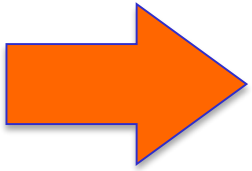

- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc)

# Analogy: mainframe to PC evolution*




Vertically integrated
Closed, proprietary
Slow innovation
Small industry



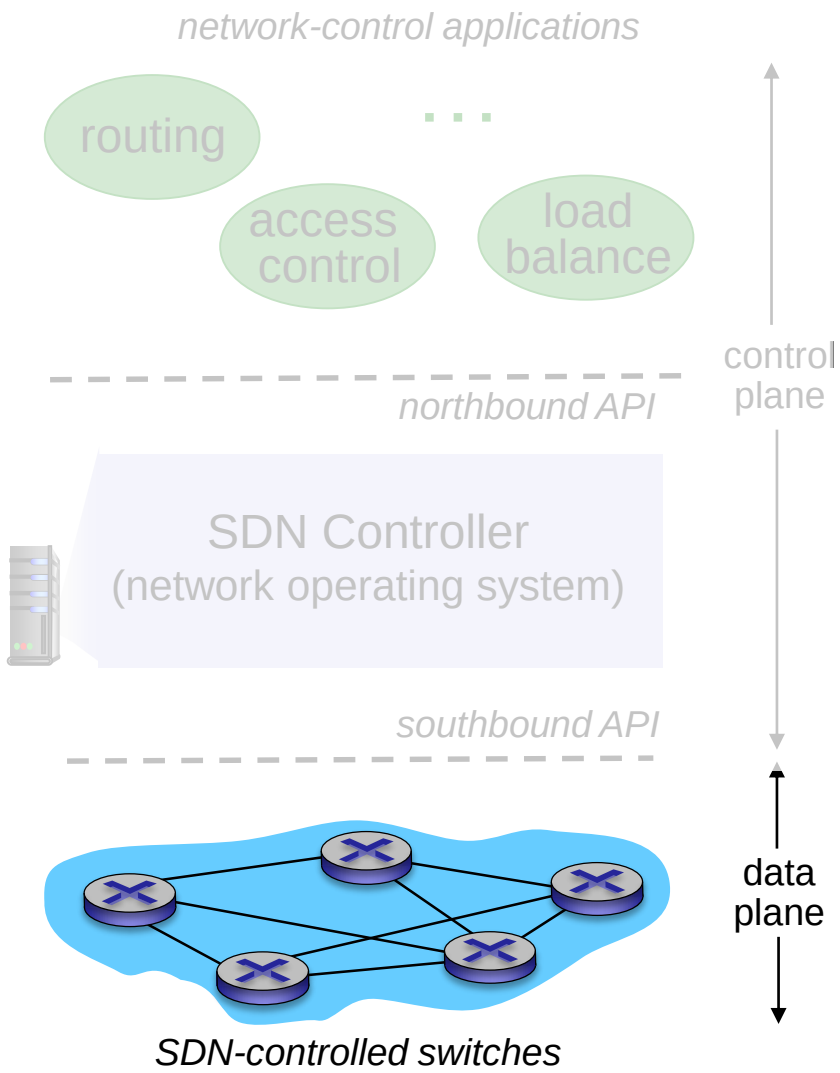
Horizontal
Open interfaces
Rapid innovation
Huge industry

* Slide courtesy: N. McKeown

# SDN perspective: data plane switches

## *Data plane switches*

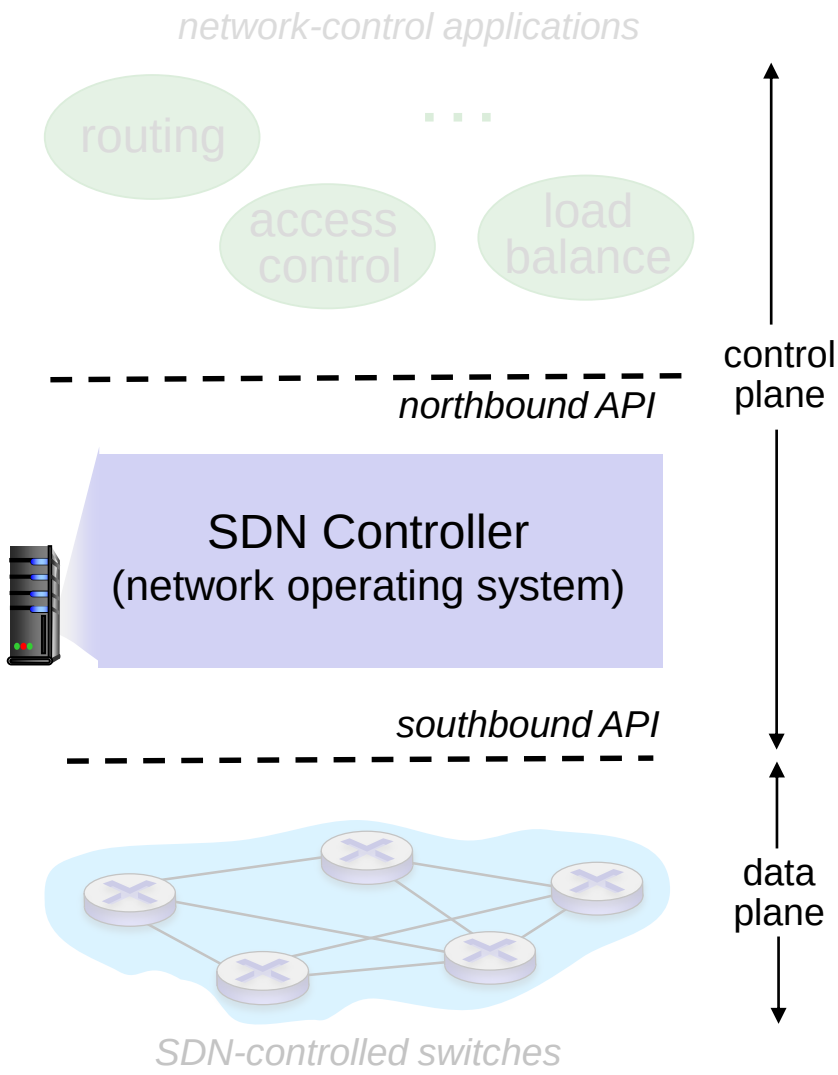
- fast, simple, commodity switches implementing generalized data-plane forwarding in hardware
- switch flow table computed, installed by controller
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable and what is not
- protocol for communicating with controller (e.g., OpenFlow)



# SDN perspective: SDN controller

*SDN controller ("network OS"):*

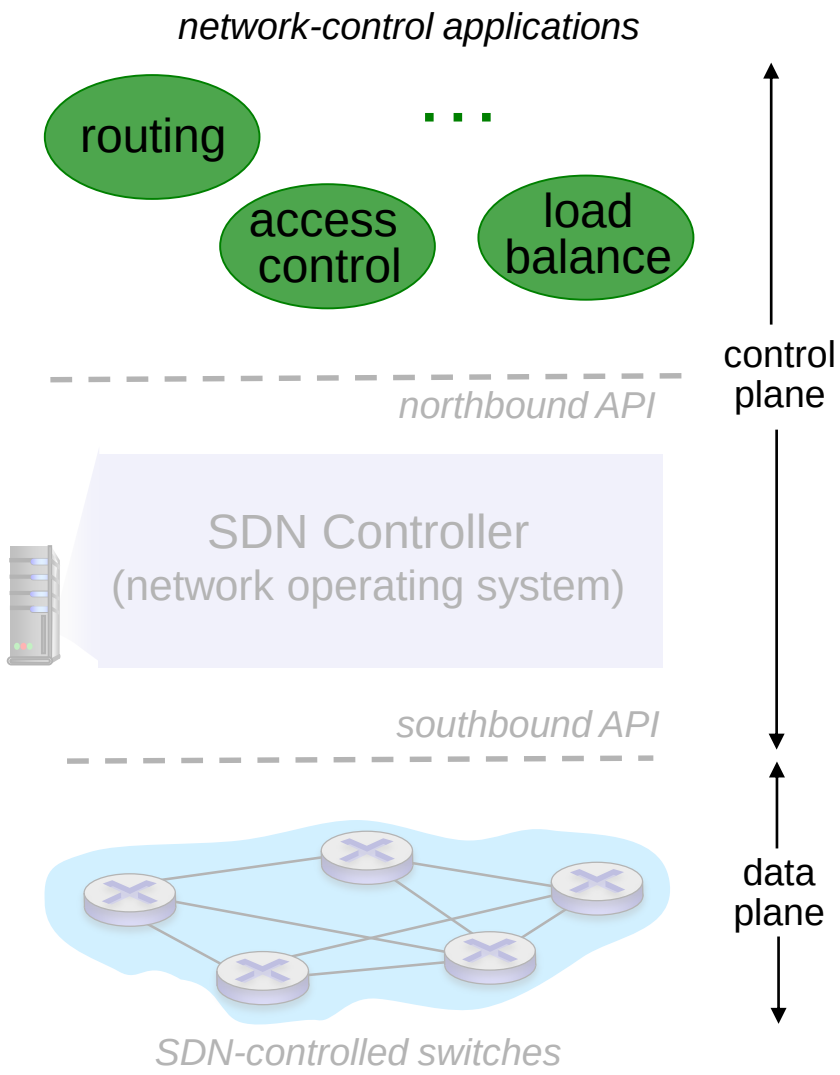
- maintain network state information
- interacts with network control applications "above" via northbound API
- interacts with network switches "below" via southbound API (OpenFlow)
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



# SDN perspective: control applications

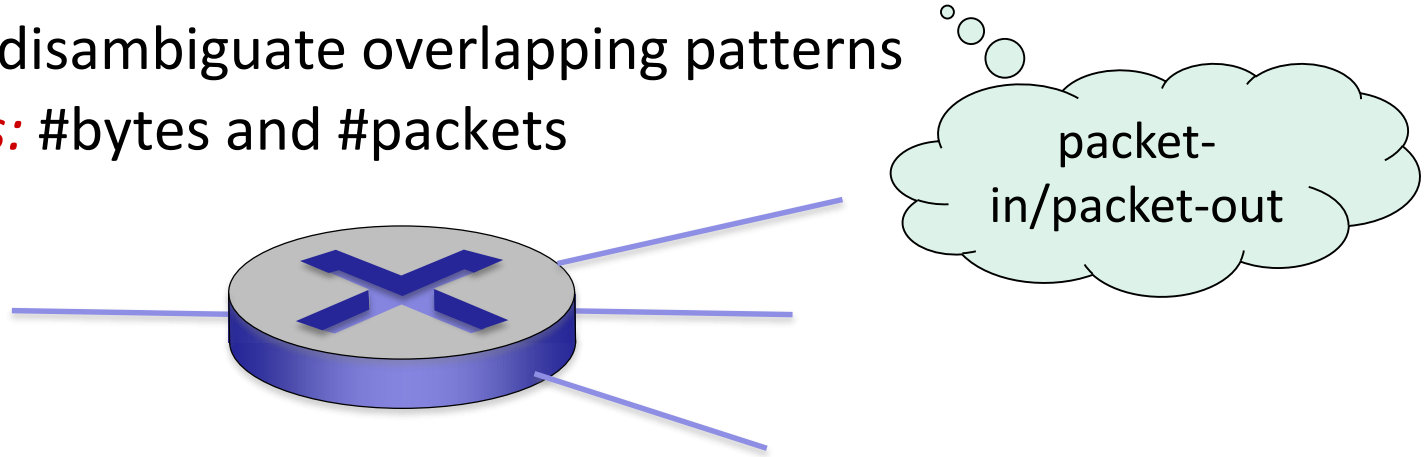
*network-control apps:*

- "brains" of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller



# OpenFlow data plane abstraction

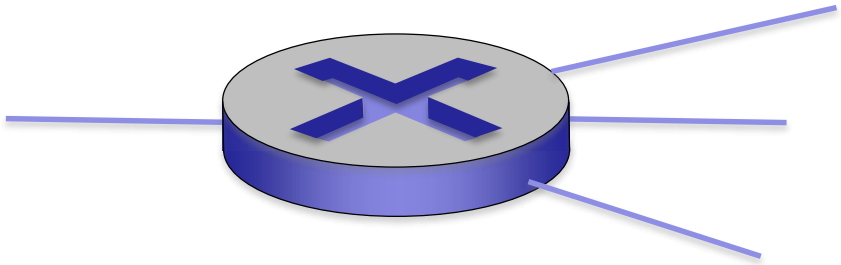
- *flow*: defined by header fields (layer 2, 3 and 4!)
- generalized forwarding: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, modify matched packet, or send (un)matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters:* #bytes and #packets




*Flow table in a router (computed and distributed by controller) define router's match+action rules*

# OpenFlow data plane abstraction

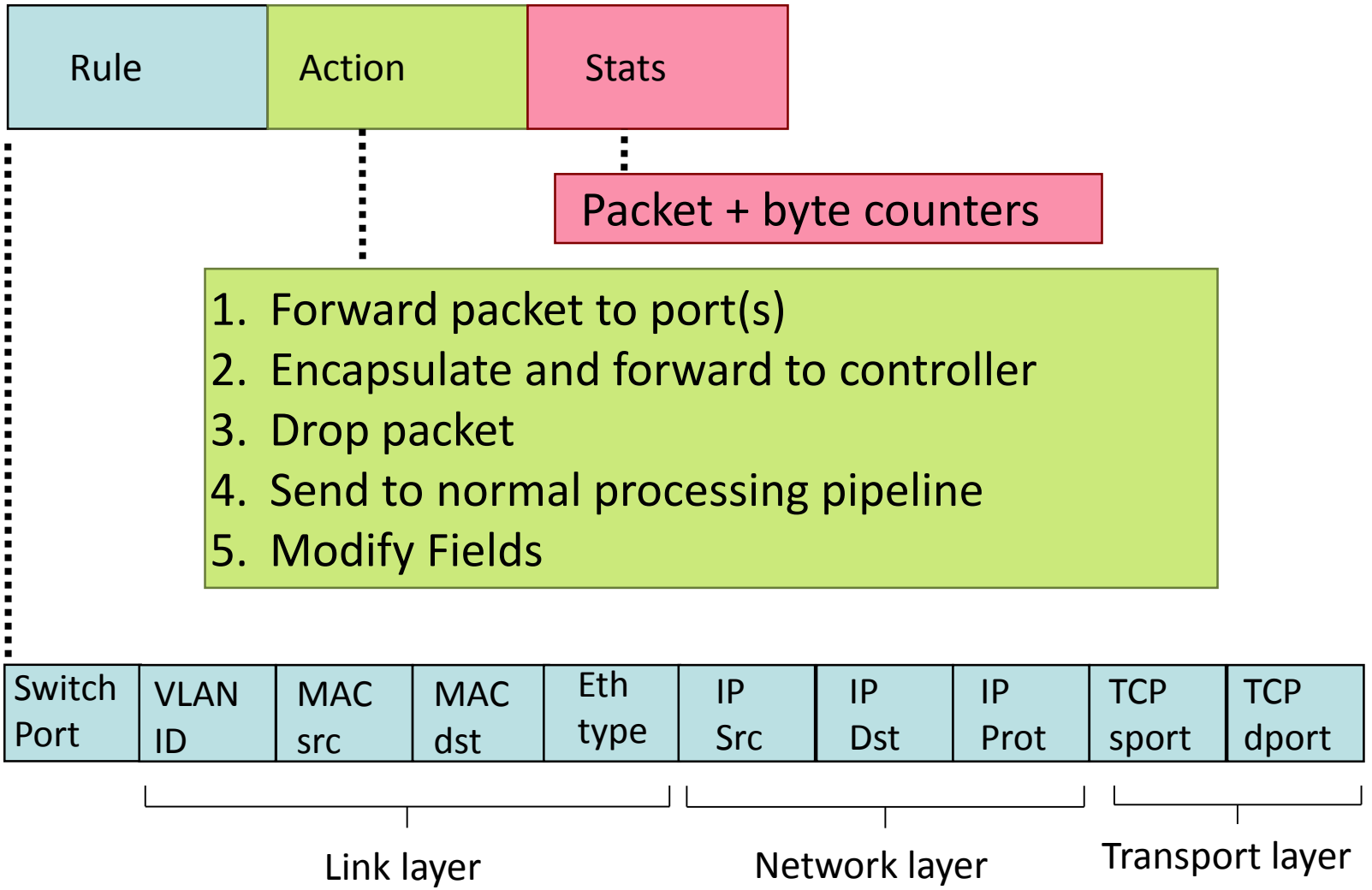
- *flow*: defined by header fields (layer 2, 3 and 4!)
- generalized forwarding: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, modify matched packet, or send (un)matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters:* #bytes and #packets



* : wildcard

1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

# OpenFlow: Flow Table Entries



# Examples

## Destination-based forwarding:

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport | Forward |
|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | * | * | * | 51.6.0.8 | * | * | * | port6 |

*IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port* 6

## Firewall:

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport | Drop |
|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | * | * | * | * | * | * | 22 | drop |

*do not forward (block) all datagrams destined to TCP port 22*

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport | Drop |
|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | * | * | 128.119.1.1 | * | * | * | * | drop |

*do not forward (block) all datagrams sent by host 128.119.1.1*

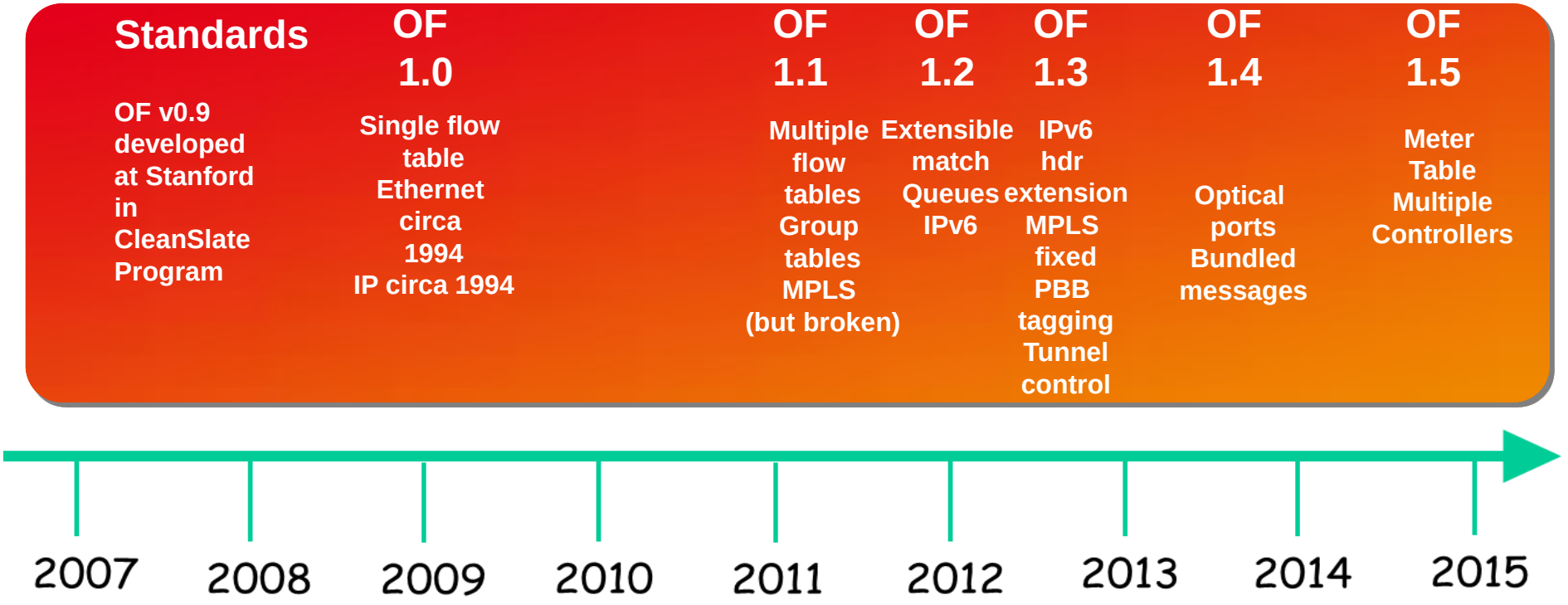
# Examples

## Destination-based layer 2 (switch) forwarding:

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport | Forward |
|---|---|---|---|---|---|---|---|---|---|---|
| * | 22:A7:23:11:E1:02 | * | * | * | * | * | * | * | * | port3 |

*layer 2 frames from MAC address 22:A7:23:11:E1:02 should be forwarded to output port* 6

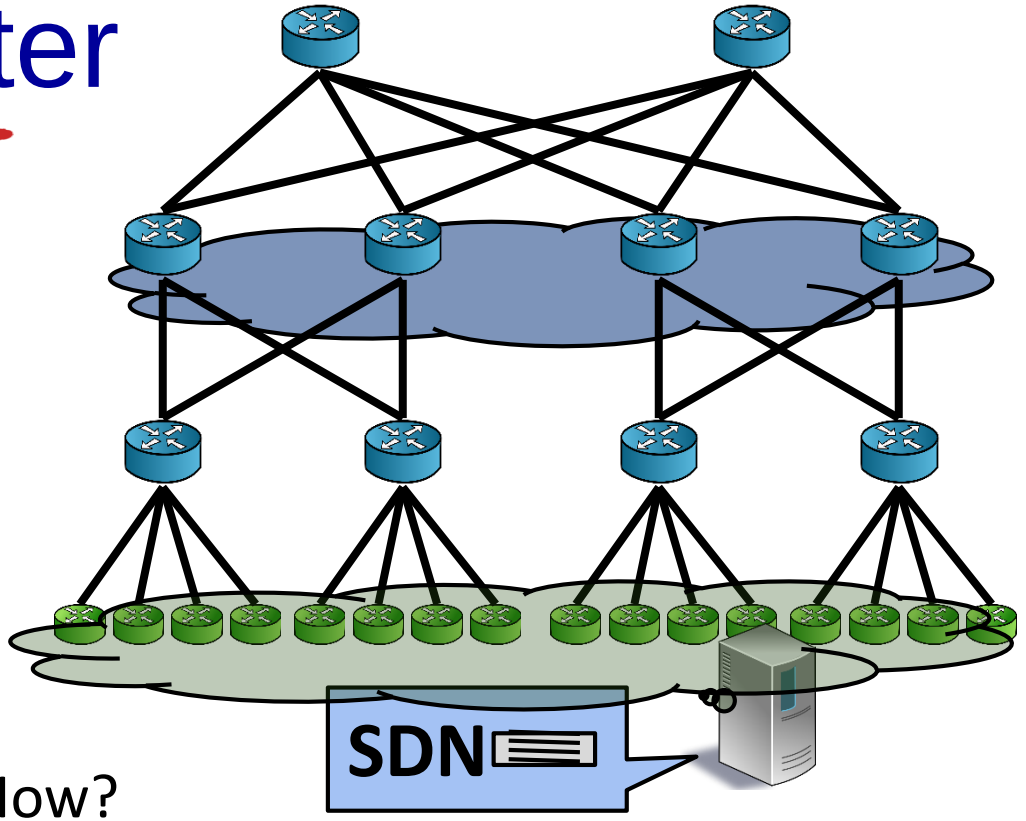
# Development of OpenFlow Specification



# SDN in Datacenter

## Characteristics

- Already highly virtualized
- Quite homogeneous
- Scalability a challenge



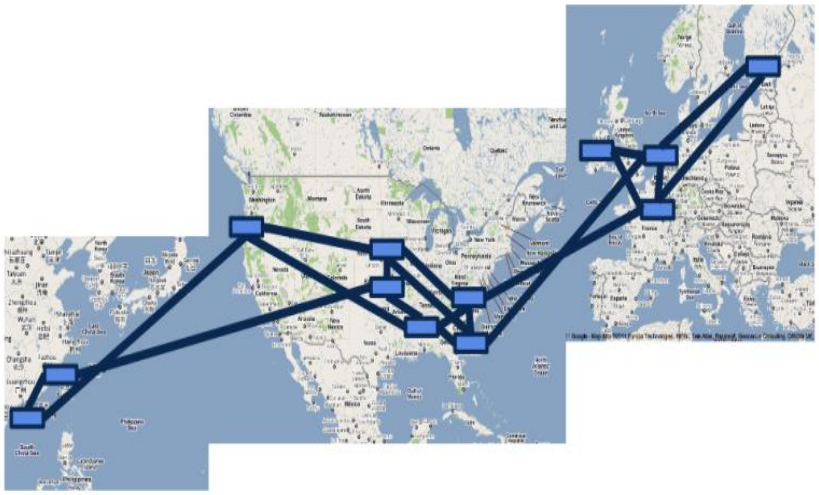

## Why SDN?

- Decouple application from physical infrastructure
- Enable virtual networks (e.g., Nicira): own addresses for tenants, isolation, support for seamless VM migration
- Performance: improve throughput

## How?

- Two separate control planes at edge and in core; «innovation» only at edge
- Provide simple fabric abstraction to tenant: classify packets at ingress and providing tunnels (through ECMP fabric)
- SDN deployment easy: software switches (Open vSwitch) at edge, software update

# SDN in WAN



## Characteristics

- Small: not many sites
- Many different applications and requirements, latency matters

- Bandwidth precious (WAN traffic grows fastest): 1G fiber connection from San Jose costs USD 3000/month

## Why SDN?

- Improve utilization (e.g., Google B4) and safe costs (e.g., Microsoft SWAN)
- Differentiate applications (latency sensitive Google docs vs datacenter synchronization)

## How?

- Replace IP "core" routers (running BGP) at border of datacenter (end of long-haul fiber)
- Gradually replace routers

# Challenges in More Depth

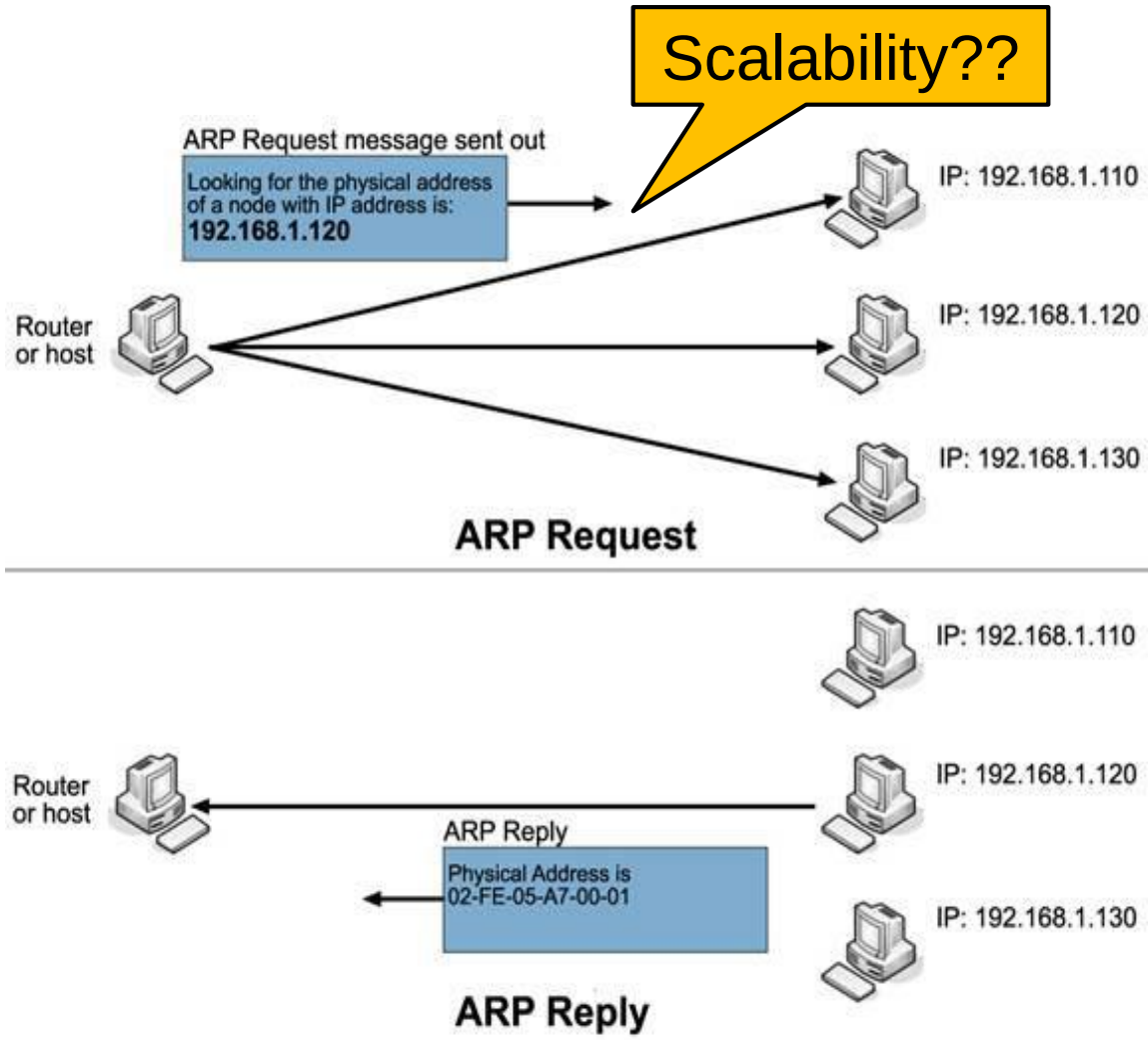
- IP address space depletion
- Inflexible routing
- Fast failover
- Slow innovation
- **Network virtualization**

# The Data Center Network Problem

- Example: a single virtualized data center built with cheap commodity servers
  - 32 VMs per server
  - 100,000 servers
  - 32 x 100,000 = 3.2 million VMs!
- Each VM needs a MAC address and an IP address
- Infrastructure needs IP and MAC addresses too
  - Routers, switches
  - Physical servers for management
- Clearly a scaling problem!

# Datacenter Problem #1:ARP/ND Handling

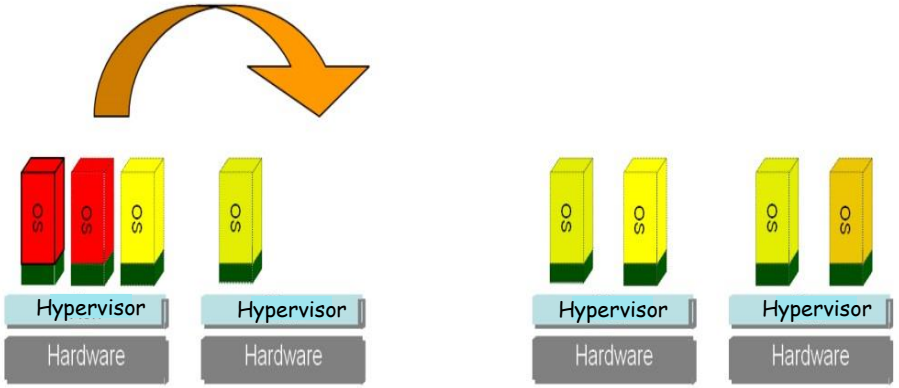
- One big Layer-2 datacenter network might be convenient: much flexibility
- IP nodes use ARP (IPv4) and N(eighbor) D(iscovery) for resolving the IP to MAC address
  - Broadcast (ARP) and Multicast (ND)
- Problem:
  - Broadcast forwarding load on large, flat L2 networks can be overwhelming




Source: http://www.louiewong.com/wp-content/uploads/2010/09/ARP.jpg

# Datacenter Problem #2: VM Movement

- Data center operators need to move VMs around
  - Reasons: server maintenance, server optimization for energy use, performance improvement, etc.
  - MAC address can stay fixed (provided it is unique in the data center)
  - If subnet changes, IP address must change because it is bound to the VM's location in the topology
    - For "hot" migration, the IP address cannot change




- Problem:
  - How broadcast domains are provisioned affects where VMs can be moved

Tradeoff: larger IP subnets = high mobility, but also higher broadcast overhead!

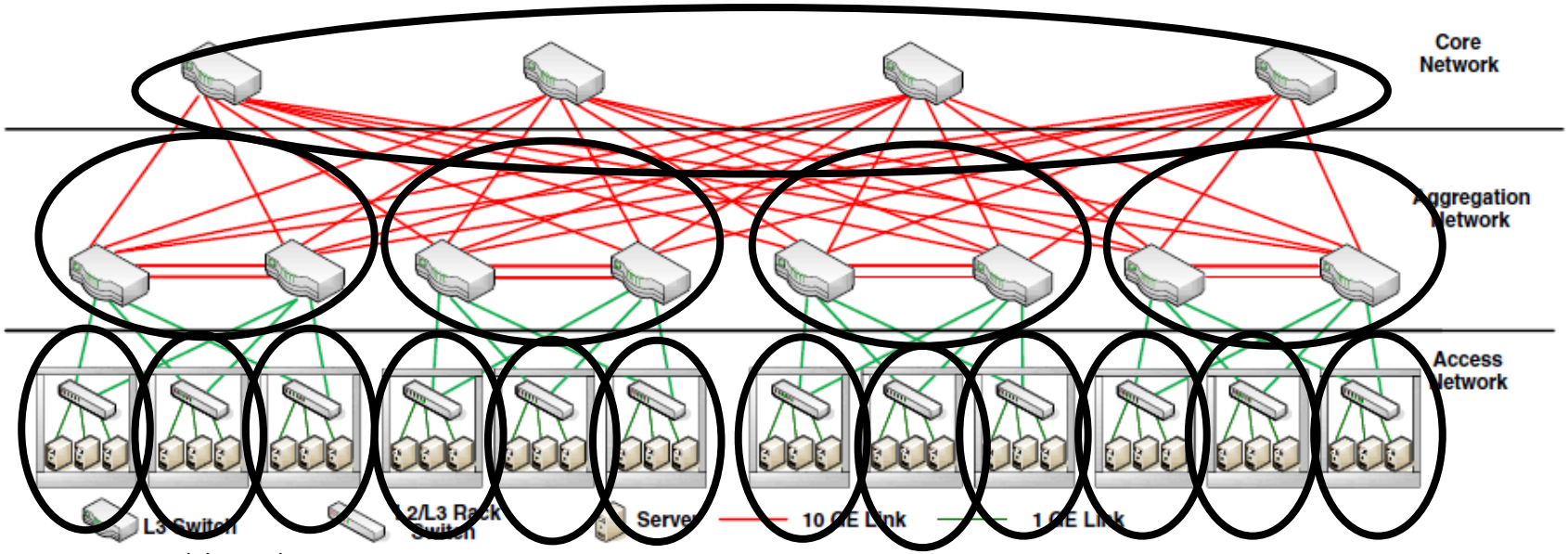
Source: http://www.freesoftwaremagazine.com/files/nodes/1159/slide4.jpg

# Solutions based on IP Subnets: How to define subnets?

- ToR == last hop router
  - Subnet (broadcast domain) limited to rack
  - Good broadcast/multicast limitation
  - Poor VM mobility
- Aggregation Switch == last hop router
  - Subnet limited to racks controlled by aggregation switch
  - Moderate broadcast/multicast limitation
  - Moderate VM mobility
    - To any rack covered
- Core Switch/Router == last hop router
  - Poor broadcast/multicast limitation
  - Good VM mobility

Efficiency
vs
migration flexibility

Also note:
Without isolation, these solutions only work if the data center is single tenant!
Otherwise: need to configure complex VLANs (see later)

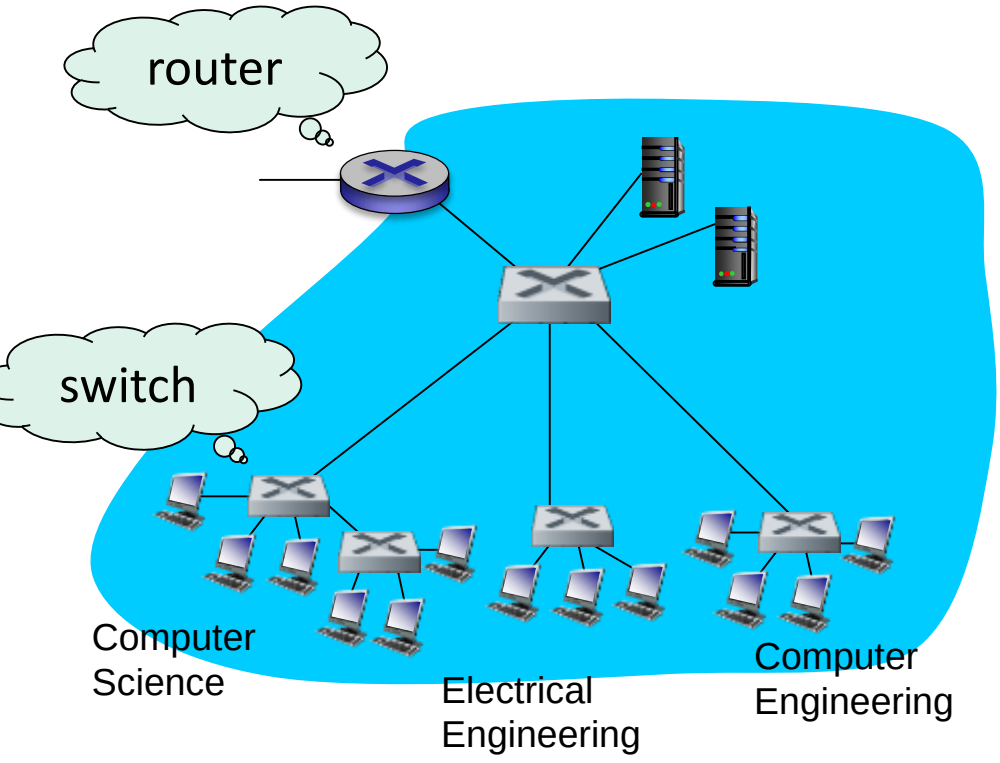



Source: Bilal, et. al.

# Traditional solution for multi-tenancy: use VLANs!

A short excursion!

# VLANs: Virtualization on Layer 2




*consider:*

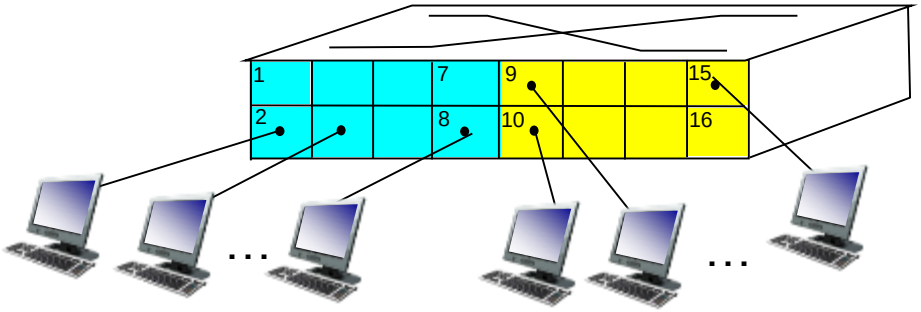
- CS user moves office to EE, but wants connect to CS switch?
- single broadcast domain:
  - all layer-2 broadcast traffic (ARP, DHCP, unknown location of destination MAC address) must cross entire LAN
  - security/privacy, efficiency issues

# VLANs

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

***Virtual Local Area Network***

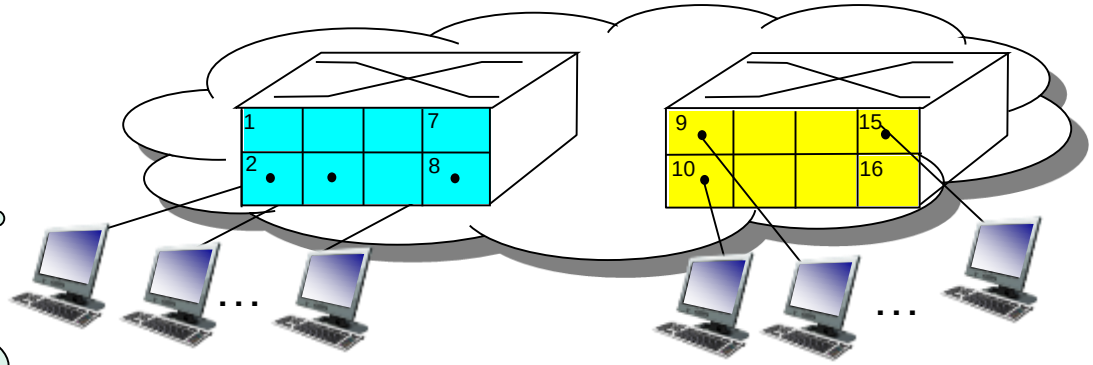
switch(es) supporting VLAN capabilities can be configured to define multiple ***virtual*** LANs over single physical LAN infrastructure.




Electrical Engineering
(VLAN ports 1-8)

Computer Science
(VLAN ports 9-15)

… operates as multiple virtual switches



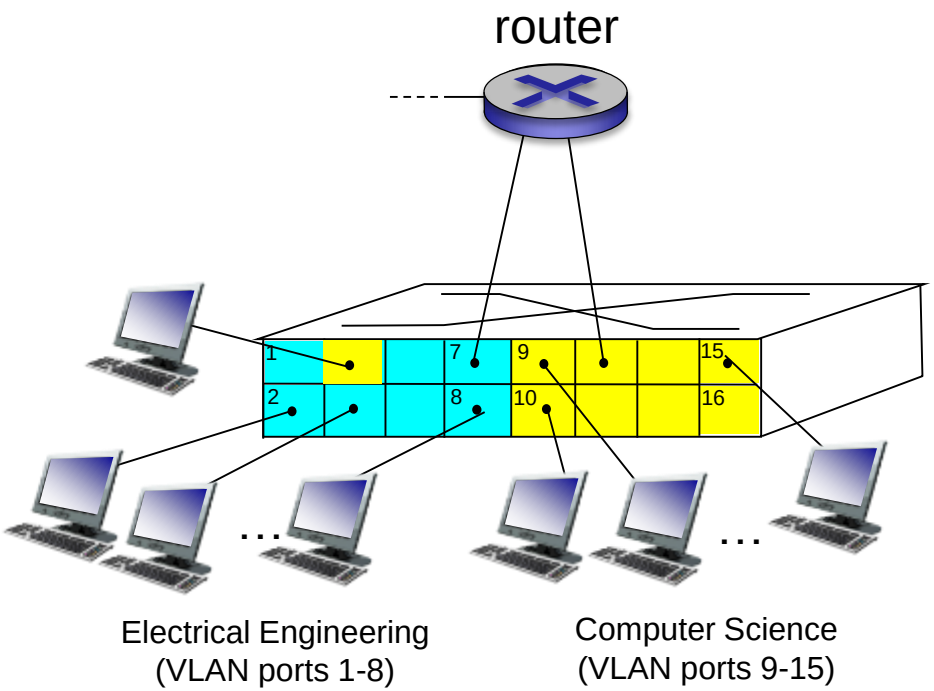

Electrical Engineering
(VLAN ports 1-8)

Computer Science
(VLAN ports 9-16)

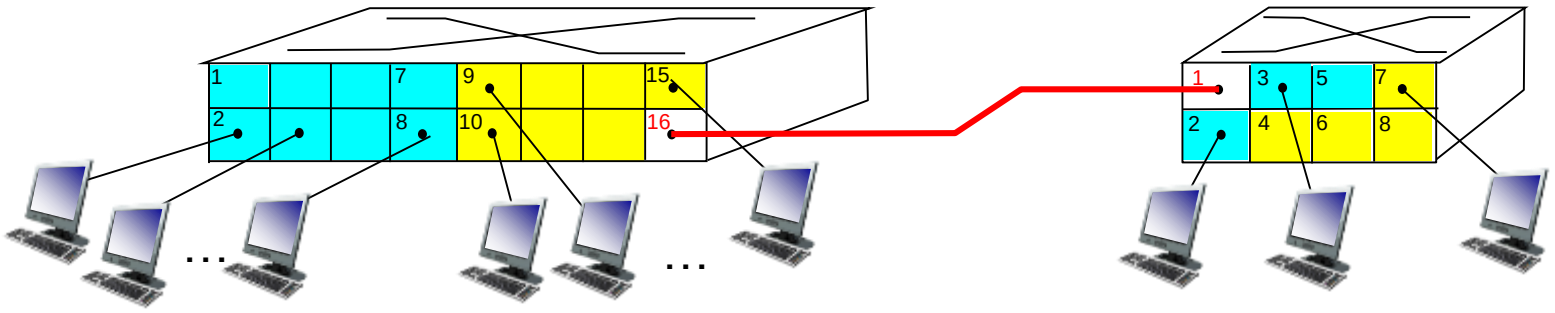
Looks like 2 physically disconnected switches!

# 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
- dynamic membership: ports can be dynamically assigned among VLANs
- forwarding between VLANs: done via routing/router (just as with separate switches)
  - in practice vendors sell combined switches plus routers



# VLANs spanning multiple switches




Electrical Engineering
(VLAN ports 1-8)

Computer Science
(VLAN ports 9-15)

Ports 2,3,5 belong to EE VLAN
Ports 4,6,7,8 belong to CS VLAN

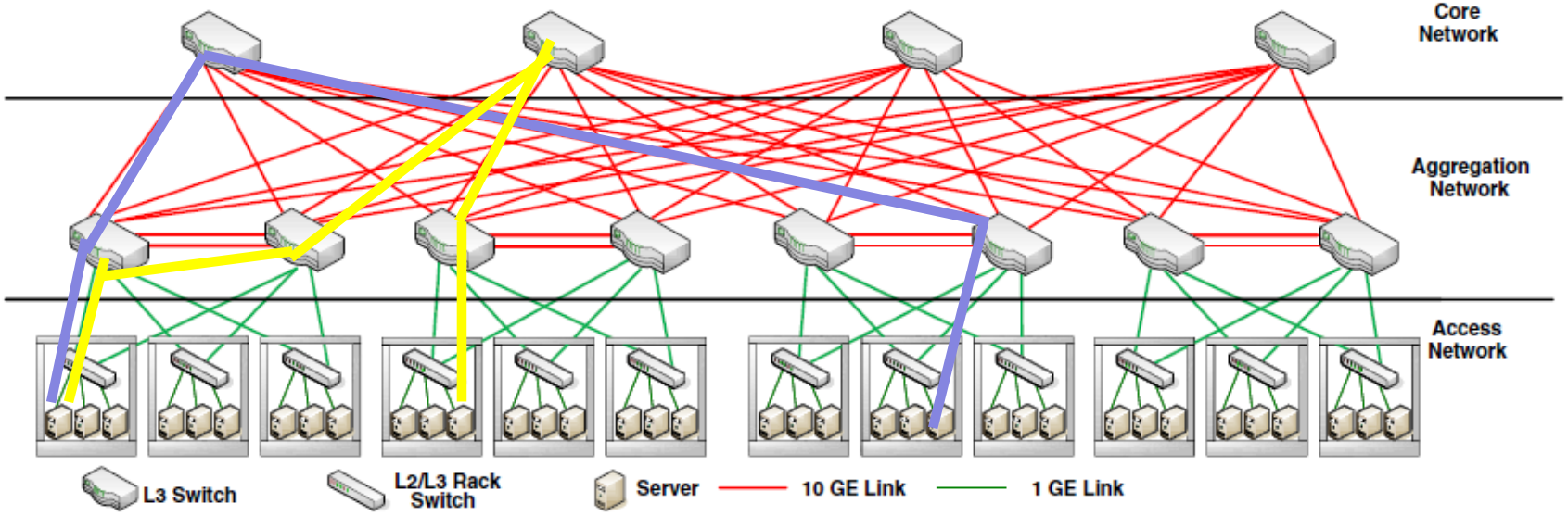
- *trunk port:* carries frames between VLANs defined over multiple physical switches
  - frames forwarded within VLAN between switches can't be vanilla 802.1 frames (must carry VLAN ID info)
  - 802.1q protocol adds/removed additional header fields for frames forwarded between trunk ports

# Conclusion: VLAN for isolation?

- Solution: use a different VLAN for each tenant network
- Problem #1: scalability
  - There are only 4096 VLAN tags for 802.1q VLANs*
- Problem #2: mostly static and complex
  - For fully dynamic VM placement, each ToR-server link must be dynamically configured as a trunk
- Problem #3: complex physical configurations
  - Can only move VMs to servers where VLAN tag is available
    - Dependencies: Ties VM movement to physical infrastructure

# Virtual Networks through Overlays

- Basic idea of an overlay:
  - Tunnel (e.g., using IP) tenant packets through underlying physical Ethernet or IP network
  - Overlay forms a conceptually separate network providing a separate service from underlay
- L2 service like VPLS or EVPN
  - Overlay spans a separate broadcast domain
- L3 service like BGP IP VPNs
  - Different tenant networks have separate IP address spaces
- Dynamically provision and remove overlay as tenants need network service
- Multiple tenants with separate networks on the same server

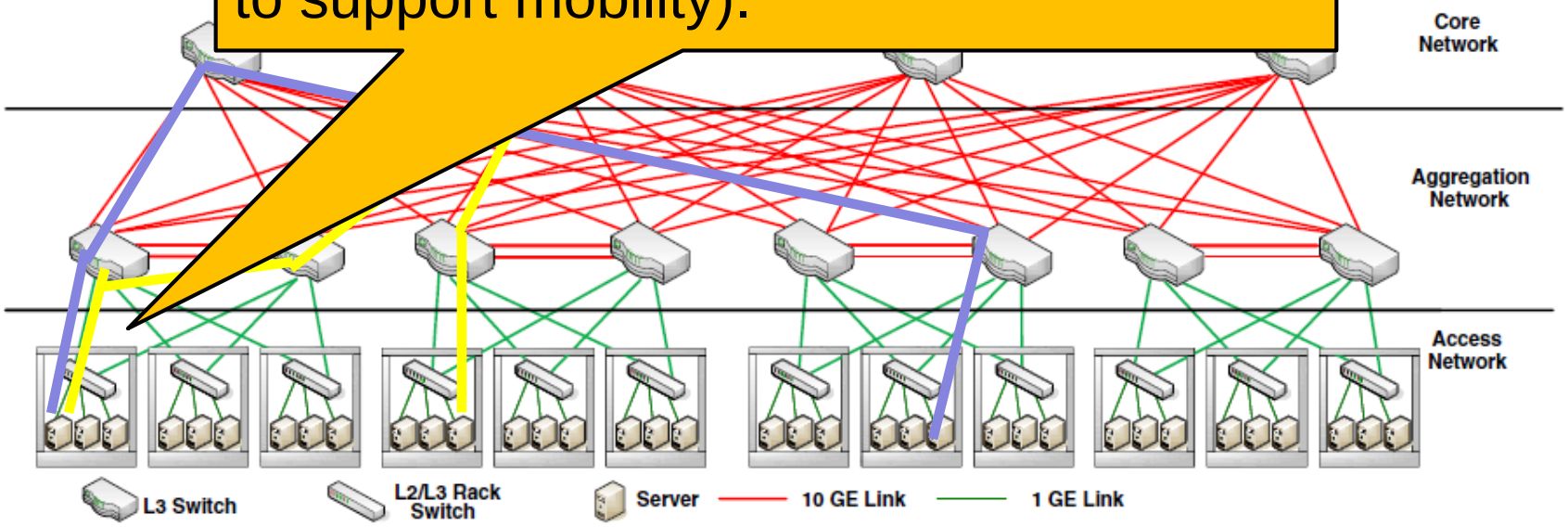



Source: Bilal, et. al.

# Virtual Networks through Overlays

- Basic idea of an overlay:
  - Tunnel (e.g., using IP) tenant packets through underlying physical Ethernet or IP network
  - Overlay forms a conceptually separate network providing a separate service from underlay
- L2 service like VPLS or EVPN
  - Overlay spans a separate broadcast domain
- L3 servic
  - Differe
- Dynamic eed network
- Multiple erver

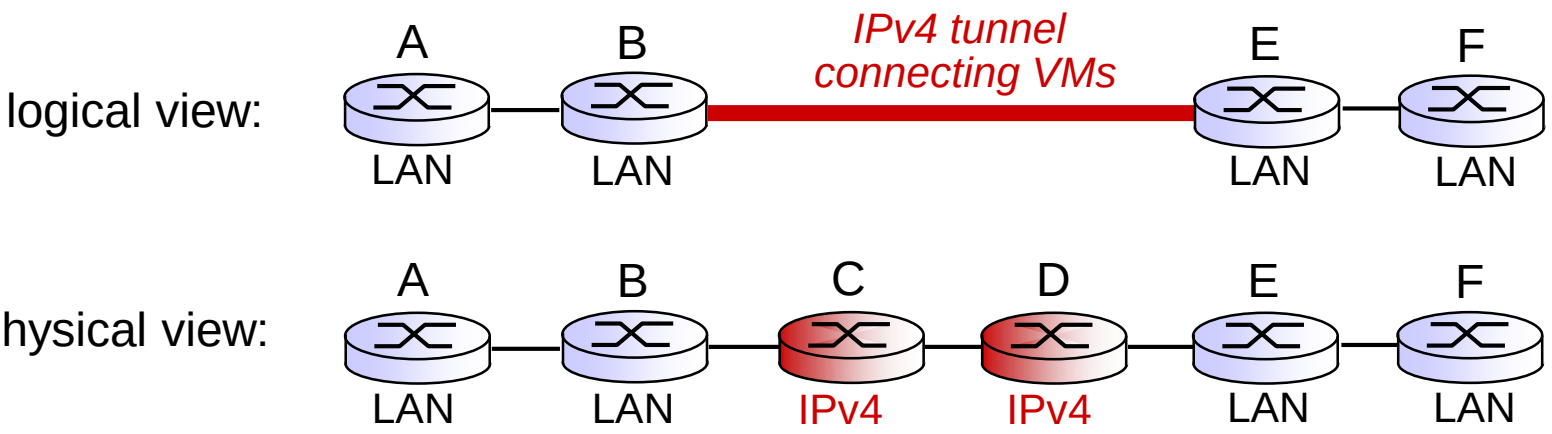
Tunnel logic at servers: encapsulation (e.g., IP)! Simple fabric/interconnect otherwise, no need to reconfigure (e.g., to support mobility).




Source: Bilal, et. al.

Blue Tenant Network  Yellow Tenant Network

# Tunneling


logical view:
A
B
IPv4 tunnel
connecting VMs
E
F
LAN
LAN
LAN
LAN
physical view:
A
B
C
D
E
F
LAN
LAN
IPv4
IPv4
LAN
LAN

# VxLAN: Virtual eXtensible LAN

- Virtual Extensible LAN (VXLAN) is an evolution of efforts to standardize on an overlay encapsulation protocol, increasing scalability up to 16 million logical networks
- Concretely: VLAN-like encapsulation technique to encapsulate MAC-based Layer-2 frames with Layer-4 UDP
- VxLAN segments constitute a broadcast domain
- VXLAN endpoints terminate tunnels and may be both virtual or physical switch ports
  - E.g., Open Vswitch (OVS)

# Advantages of Overlays

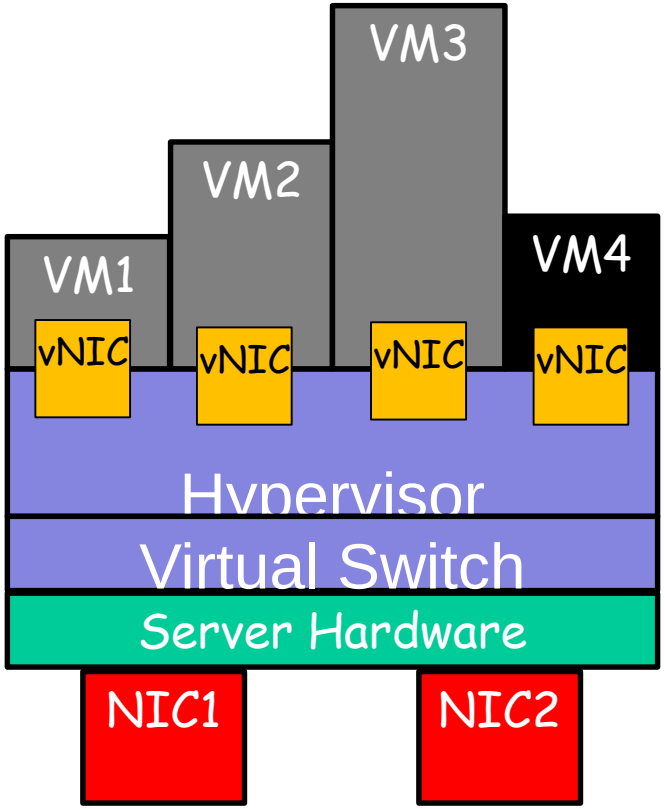
- Overlays can potentially support large numbers of tenant networks
- Virtual network state and end node reachability are handled in the end nodes (the servers, “fabric”)
- Tenant addresses hidden from other tenants
  - Multiple tenants with the same IP address space
- Addresses in underlay are hidden from the tenant
  - Inhibits unauthorized tenants from accessing data center infrastructure
- Tunneling is used to aggregate traffic

# Virtualization of dataplane: Virtual Switches

- Recall: datacenter network virtualization often relies on virtual switches
- Virtual switch: software that emulates a physical switch
  - Conceptually sits in the hypervisor
  - Downlink: Emulates NICs toward VMs (vNICs)
  - Uplink: Connects to hardware NICs through hypervisor
- Provides virtualized networking support to VMs
  - Bridges traffic between VMs on the physical server
  - Switches off-server traffic from the ToR to the VMs

# Virtual Switches

- Hypervisor provides a compute abstraction layer
  - OSes run as multiple Virtual Machines (VMs) on single server
  - Looks like hardware to operating system
- Hypervisor maps VM to processors
  - Virtual cores (vCores)
- Virtual switch (vSwitch) provides networking between VMs and to DC network
  - Virtual NICs (vNICS)
- W.o. oversubscription, usually as many VMs as cores
- VMs can be moved from one machine to another



# Example Virtual Switch: Open Virtual Switch (OVS)

- History
  - 2007- First released by Nicira
  - 2010 - OpenFlow 1.0 support
  - 2011- Ported to hardware (Pica8)
  - 2012 – VMWare acquires Nicira, commits to maintaining OVS
  - 2012 – OVS becomes part of Linux release
  - 2014 – OpenFlow 1.5 support
- Open source, maintained by VMWare
  - Included with Xen, KVM open source hypervisors
- Proprietary competitors
  - Cisco 1000V (number one)
  - VMWare vDS (but trending downward)
  - IBM DVS 5000V
  - Microsoft Hyper-V vswitch

# Security of OVS

**New attack vector:**

- The virtualized data plane

**Background:**

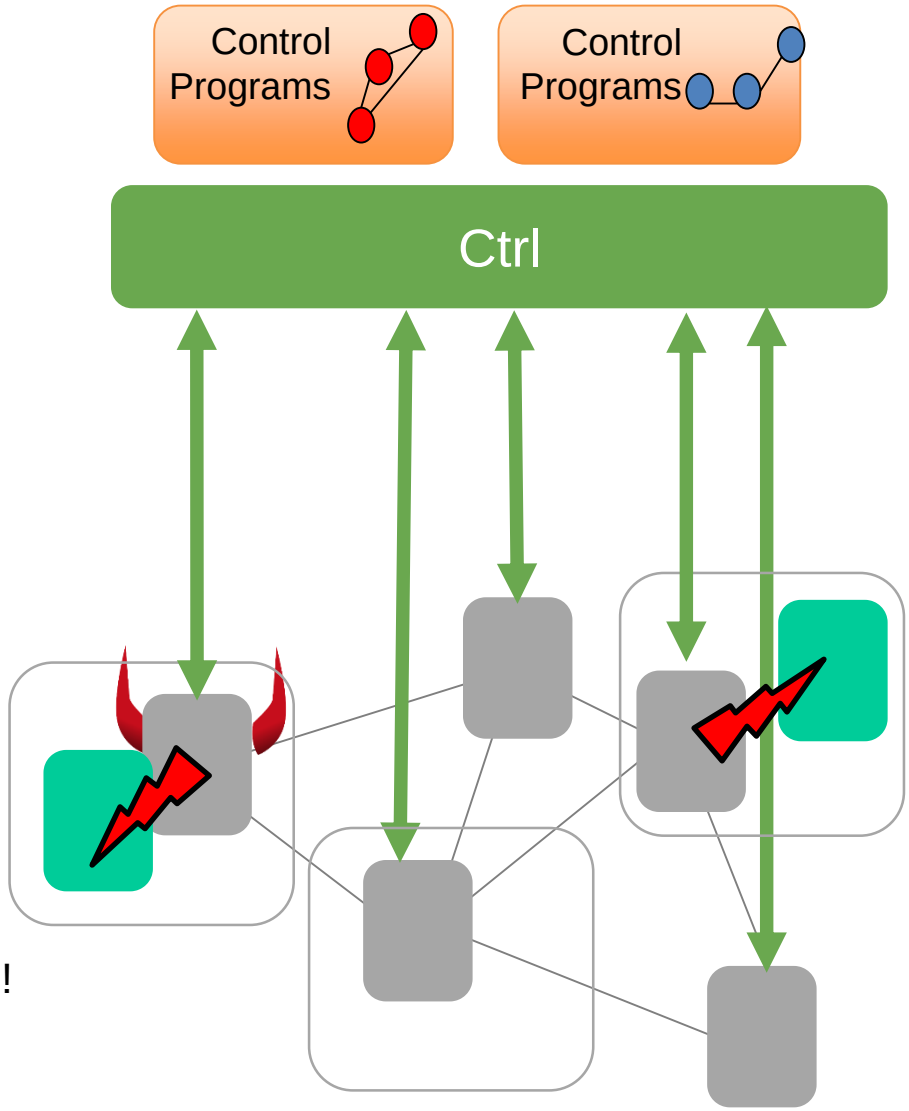
- Packet processing and other network functions are more and more virtualized
- E.g., they run on servers at the edge of the datacenter
- Example: OVS

**Advantage:**

- Cheap and performance ok!
- Fast and easy deployment

**Disadvantage:**

- New vulnerabilities, e.g., collocation!



# Security of OVS

**New attack vector:**

- The virtualized data plane

**Background:**

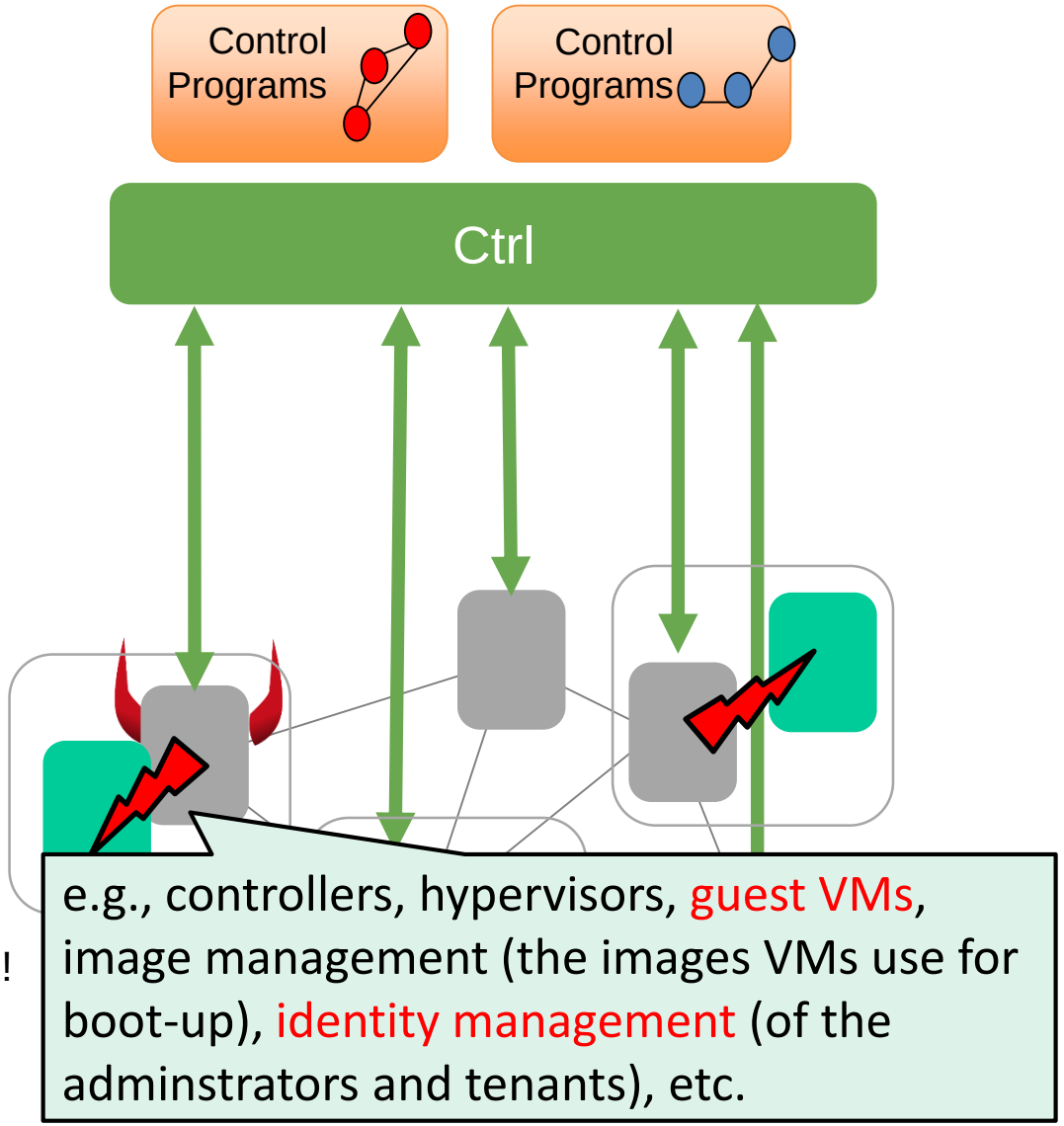
- Packet processing and other network functions are more and more virtualized
- E.g., they run on servers at the edge of the datacenter
- Example: OVS

**Advantage:**

- Cheap and performance ok!
- Fast and easy deployment

**Disadvantage:**

- New vulnerabilities, e.g., collocation!



# Case Study OVS

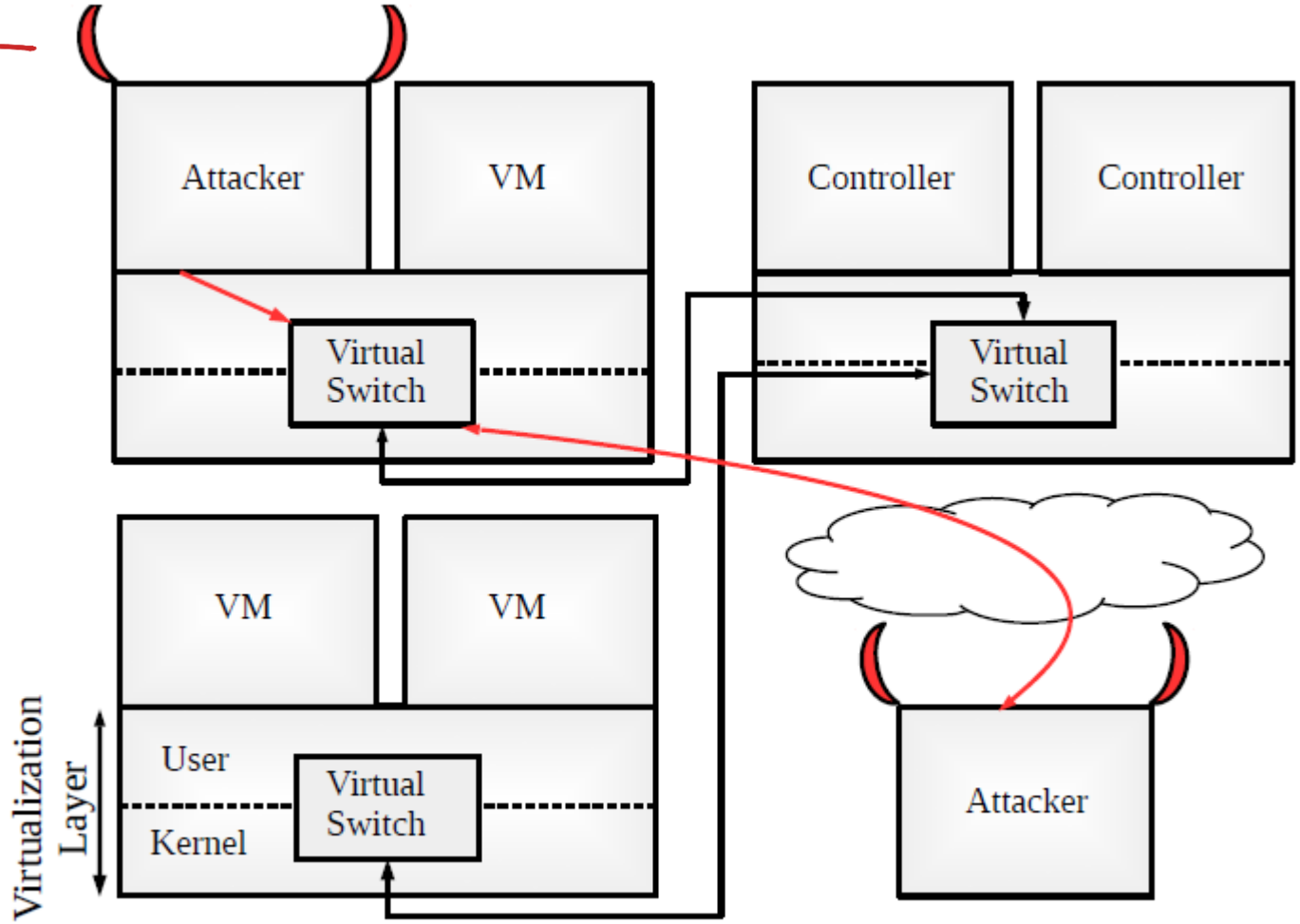
- OVS: a production quality switch, widely deployed in the Cloud
- After fuzzing just 2% of the code, we found major vulnerabilities:
  - E.g., two stack overflows when malformed MPLS packets are parsed
- These vulnerabilities can easily be weaponized:
  - Can be exploited for arbitrary remote code execution
  - E.g., our «reign worm» compromised cloud setups within 100s
- Significance
  - It is often believed that only state-level attackers (with, e.g., control over the vendor's supply chain) can compromise the data plane
  - Virtualized data planes can be exploited by very simple, low-budget attackers: e.g., by renting a VM in the cloud and sending a single malformed MPLS packet

# Analysis

Exploits 4 problems:

1. **Security assumptions:** Virtual switches often run with elevated (root) priviledges by design.
2. **Collocation:** virtual switchs reside in virtualized servers (Dom0), and are hence collocated with other and possibly critical cloud software, including controller software
3. **Logical centralization:** the control of data plane elements is often outsourced to a centralized software. The corresponding bidirectional communication channels can be exploited to spread the worm further.
4. **Support for extended protocol parsers:** Virtual switches provide functionality which goes beyond basic protocol locations of normal switches (e.g., handling MPLS in non-standard manner)

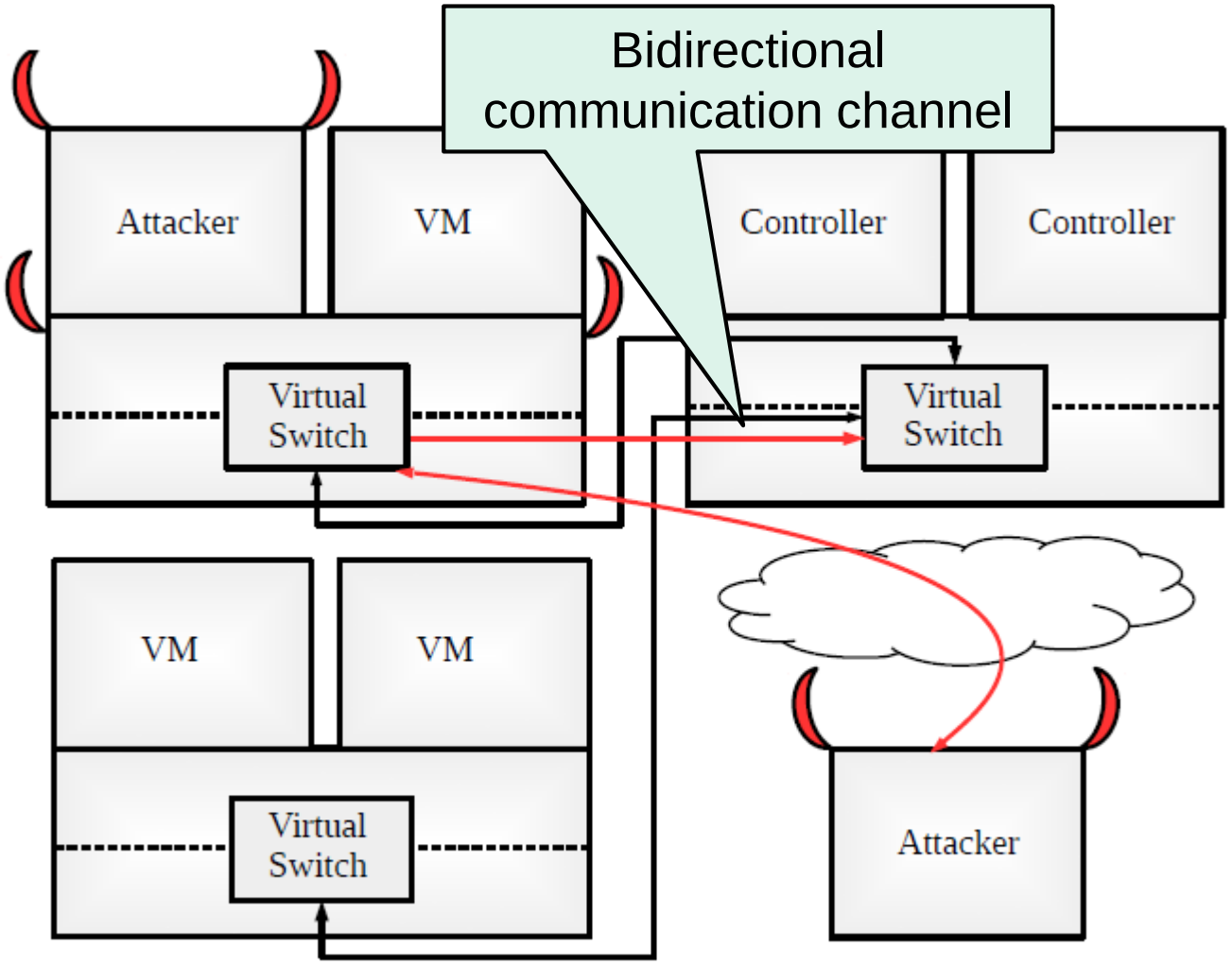
# Worm




(a) Step 1.

Attacker VM sends a malicious packet that compromises its server, giving the remote attacker control of the server.

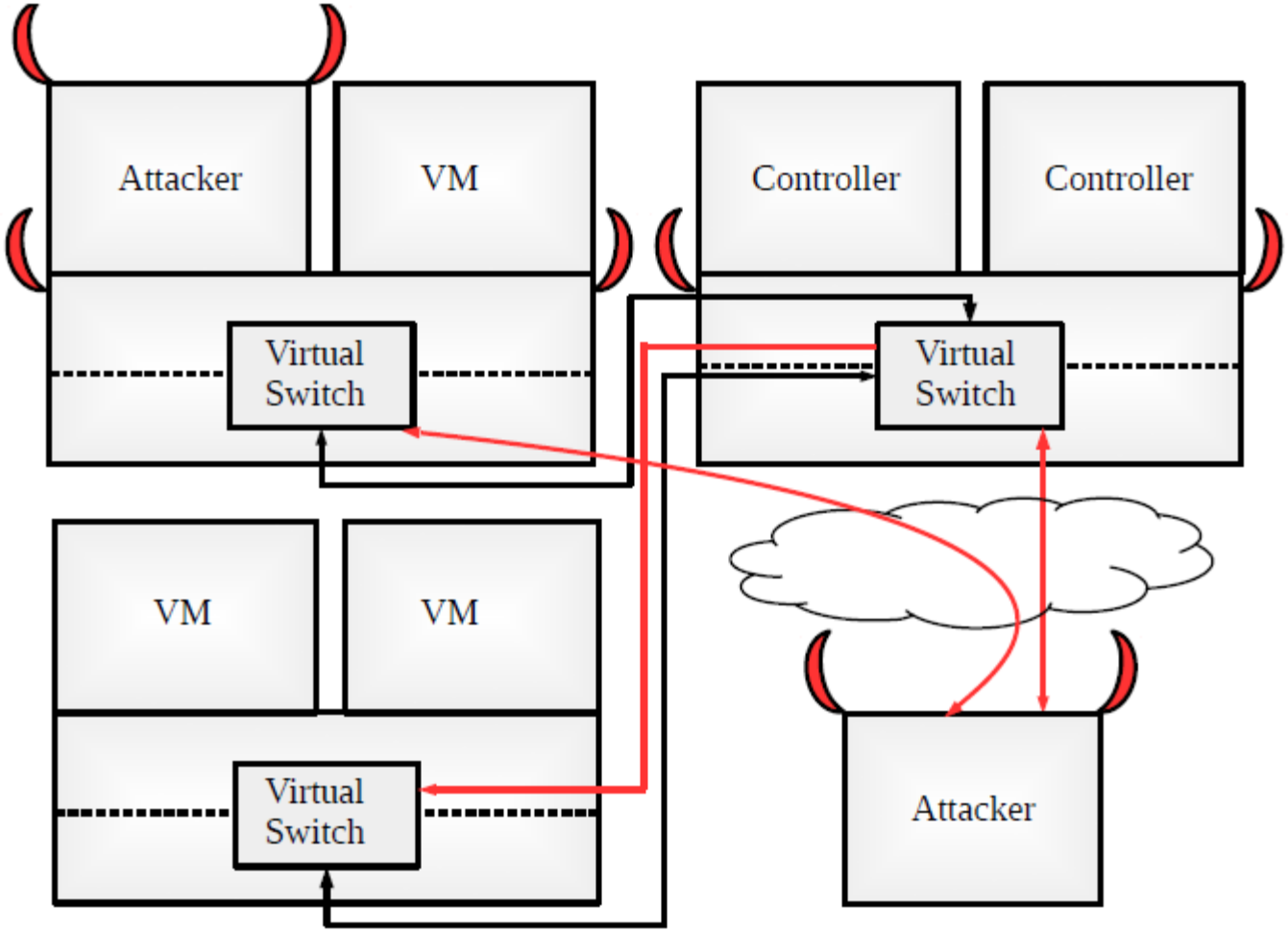
# Worm




(b) Step 2.

Attacker controlled server compromises the controllers' server, giving the remote attacker control of the controllers' server.

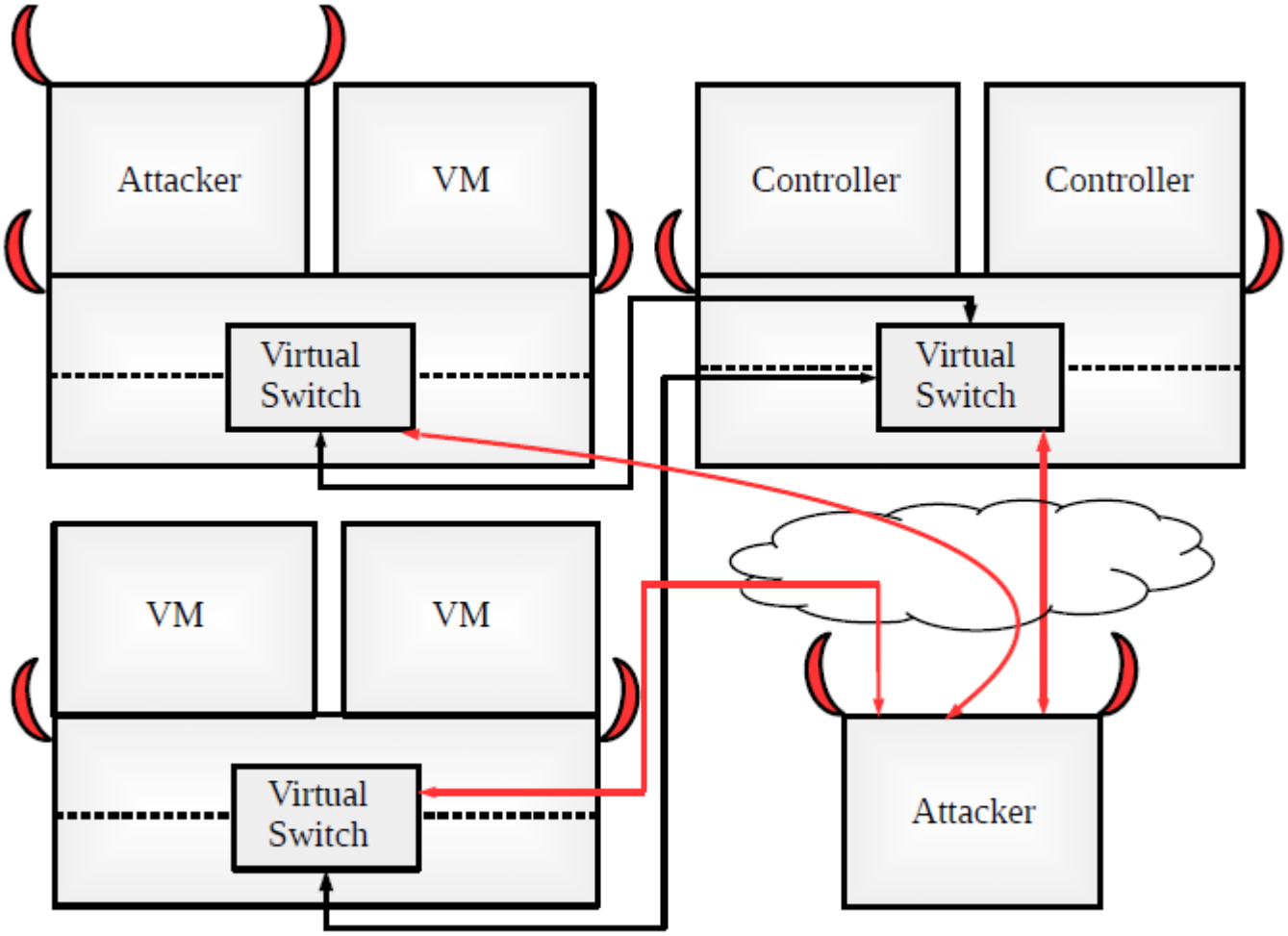
# Worm




(c) Step 3.

The compromised controllers' server propagates the worm to the remaining uncompromised server.

# Worm




(d) Step 4.

All the servers are controlled by the remote attacker.

# That’s all

- Become a master!
- Become a critic!