Networking

Prof. Bracy and Van Renesse
CS 4410
Cornell University

based on slides by Prof. Sirer and Van Renesse

Basic Network *Abstraction*

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

Some issues…

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

Network "protocol"

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

Network Layering

- The network abstraction is usually *layered*
  - Essentially the same as OO-style inheritance
- Example:

Application Layer	Clients and servers, remote procedure call
Transport Layer	Reliable networking, retransmission
Network Layer	Abstract networks, routing
Link Layer	Ethernet, etc.
Physical Layer	wires, signal encoding, wireless, etc.

Application Layer
Transport Layer
Network Layer
Link Layer
Physical Layer

Link Layer: Local Area Networking (LAN) and Ethernet

Link Layer

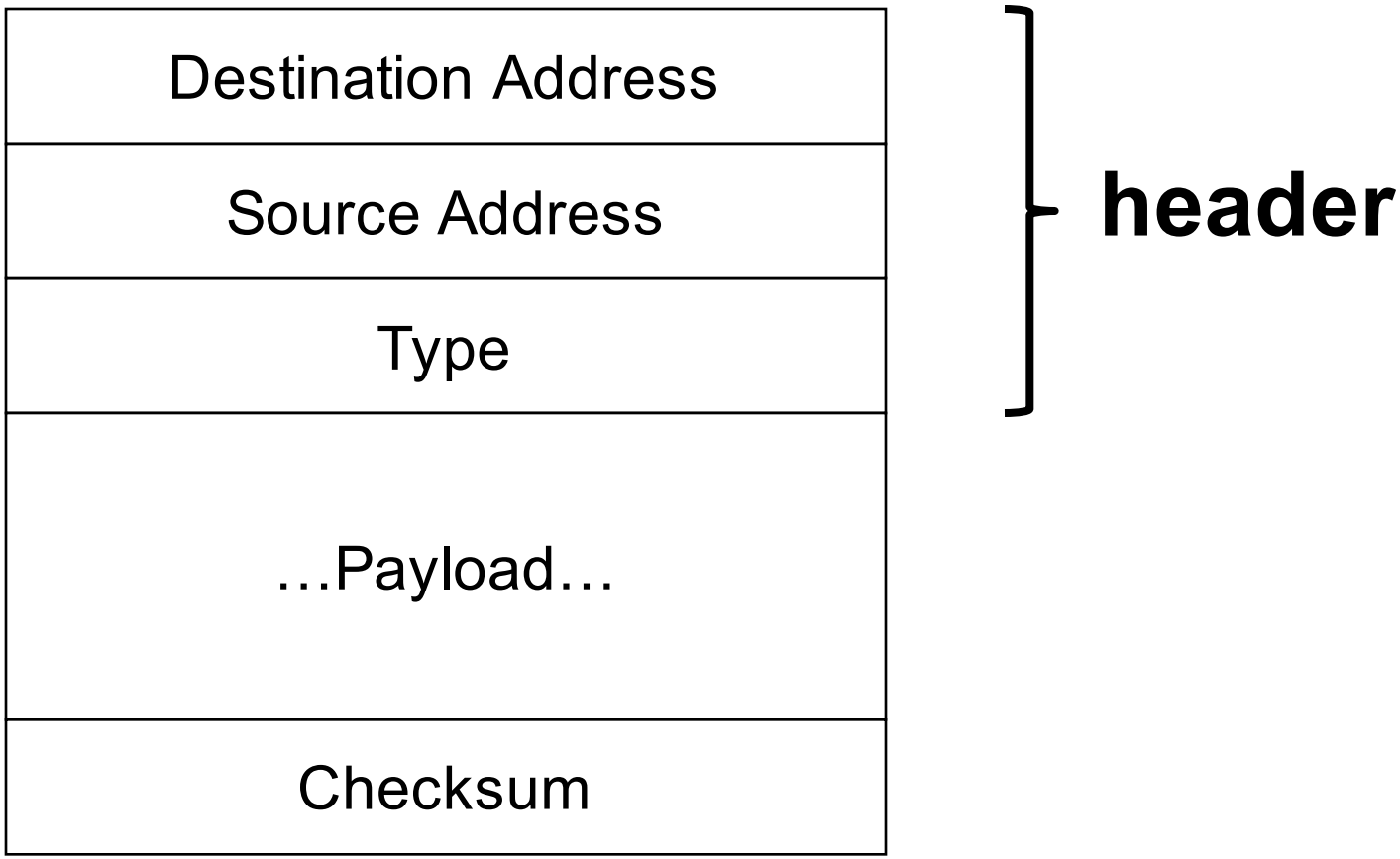
- Each host has one or more *NICs*
  - *Network Interface Cards*
    - Ethernet, 802.11, etc.
- Each NIC has a *MAC address*
  - *Media Access Control* address
  - Ethernet example: b8:e3:56:15:6a:72
  - Unique to network instance
    - often even globally unique
- Messages are *packets* or *frames*

Example: Ethernet

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

Ethernet basics

An Ethernet packet



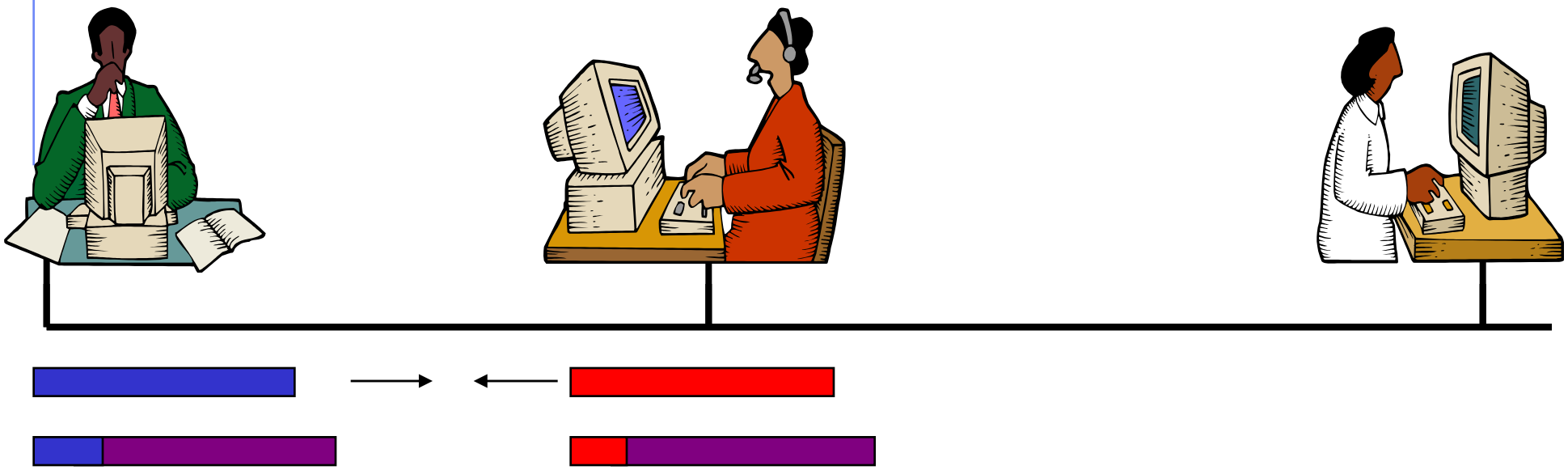
"CSMA/CD"

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

Sending packets

- Carrier sense, broadcast if ether is available

Collisions



- What happens if two people decide to transmit simultaneously ?

Collision Detection & Retransmission

- The hosts involved in the collision stop data transmission, sleep for a while, and attempt to retransmit
- How long they sleep is determined by how many collisions have occurred before
- They abort after 16 retries, hence no guarantee that a packet will get to its destination
- Packets are truncated early to avoid wasting bandwidth

CRC Checksum
(Cyclic Redundancy Check)

- Basically a hash function on the packet
- Added to the end of a packet
- Used to detect malformed packets, e.g. electrical interference, noise

Ethernet Features

- Completely distributed
  - No central arbiter
- Inexpensive
  - No state in the network
  - No arbiter
  - Cheap physical links (twisted pair of wires)

Ethernet Problems

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

Switched Ethernet

- Today's Ethernet deployments are much faster
- In wired settings, *Switched Ethernet* has become the norm
  - All hosts connect to a switch
  - Each p2p connection is a mini Ethernet set-up
  - More secure, no possibility of snooping
  - Switches organize into a spanning tree
- Not to be confused with Ethernet *Hub*
  - A hub simply connects the wires

Wireless

- 802.11 protocols inherit many of the Ethernet concepts
- Full compatibility with Ethernet interface
  - Same address and packet formats

Lessons for LAN design

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

Application Layer
Transport Layer
Network Layer
Link Layer
Physical Layer

Network Layer

Network Layer

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

Internetworking Origins

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

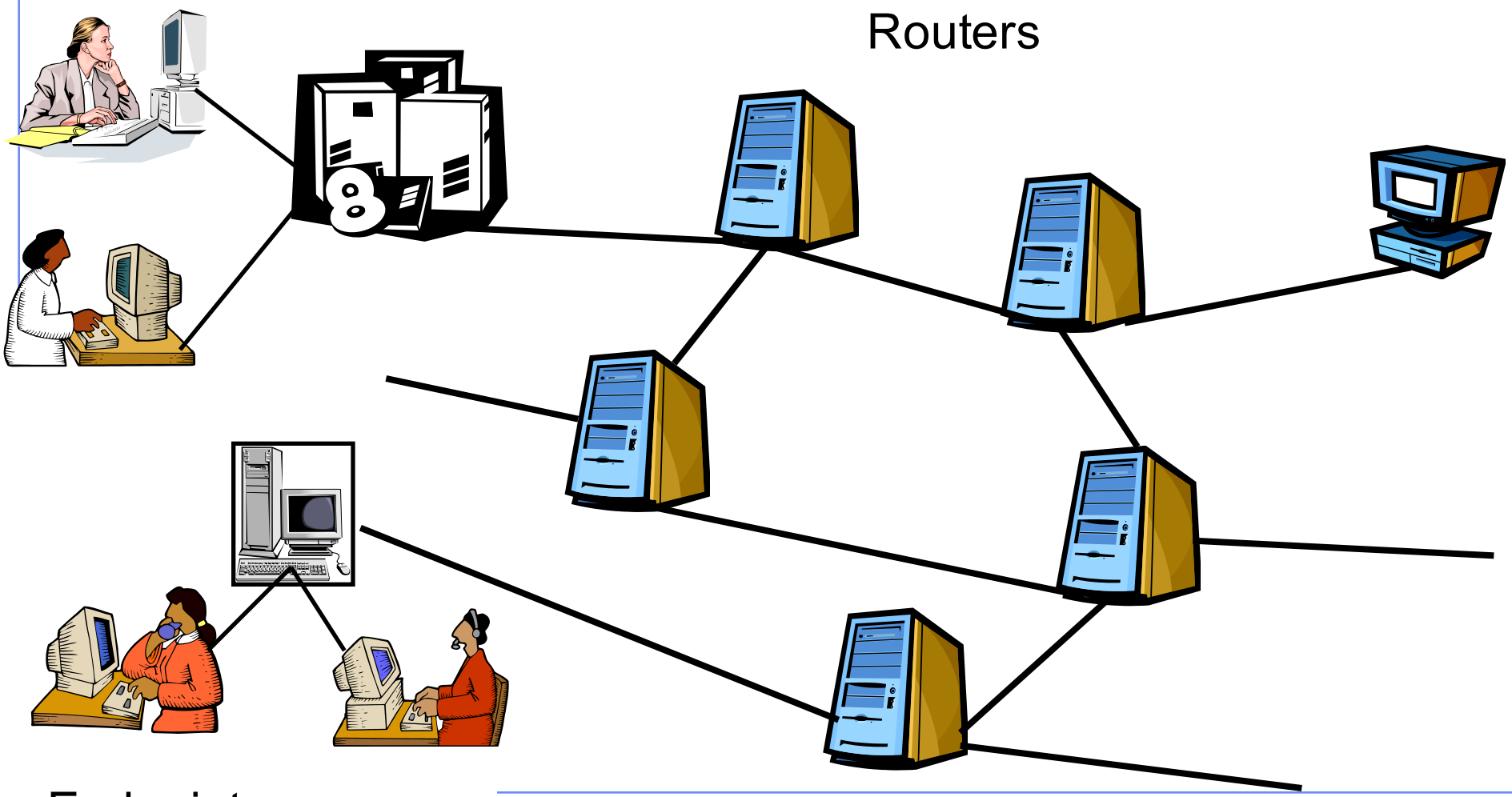
Internetworking Origins

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

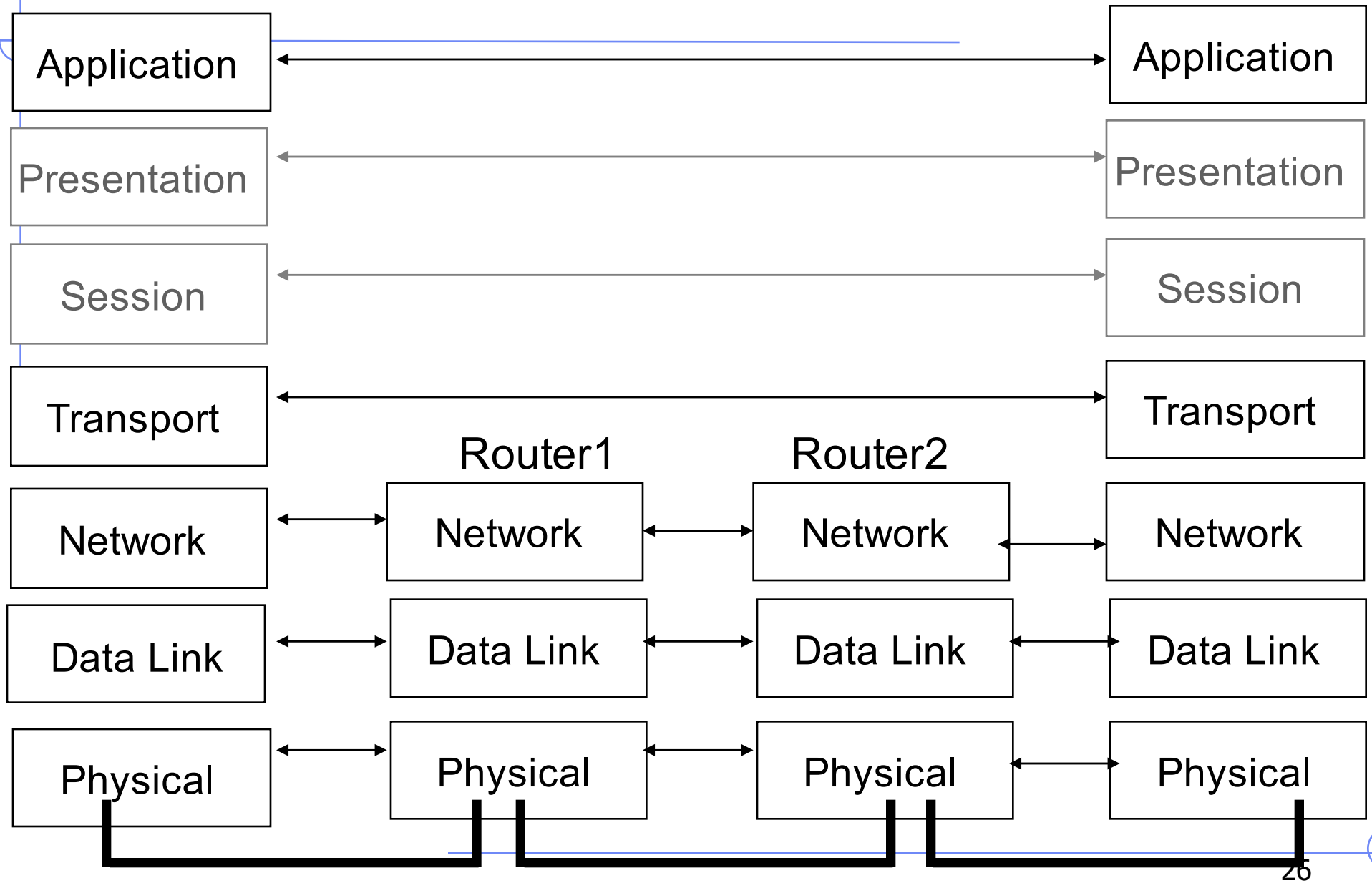
Internet Overview

- Every host is assigned, and identified by, an IP address
- Messages are called datagrams
  - the term *packet* is probably more common though…
- Each datagram contains a header that specifies the destination address
- The network routes datagrams from the source to the destination
- Question: What kinds of properties should the network provide?

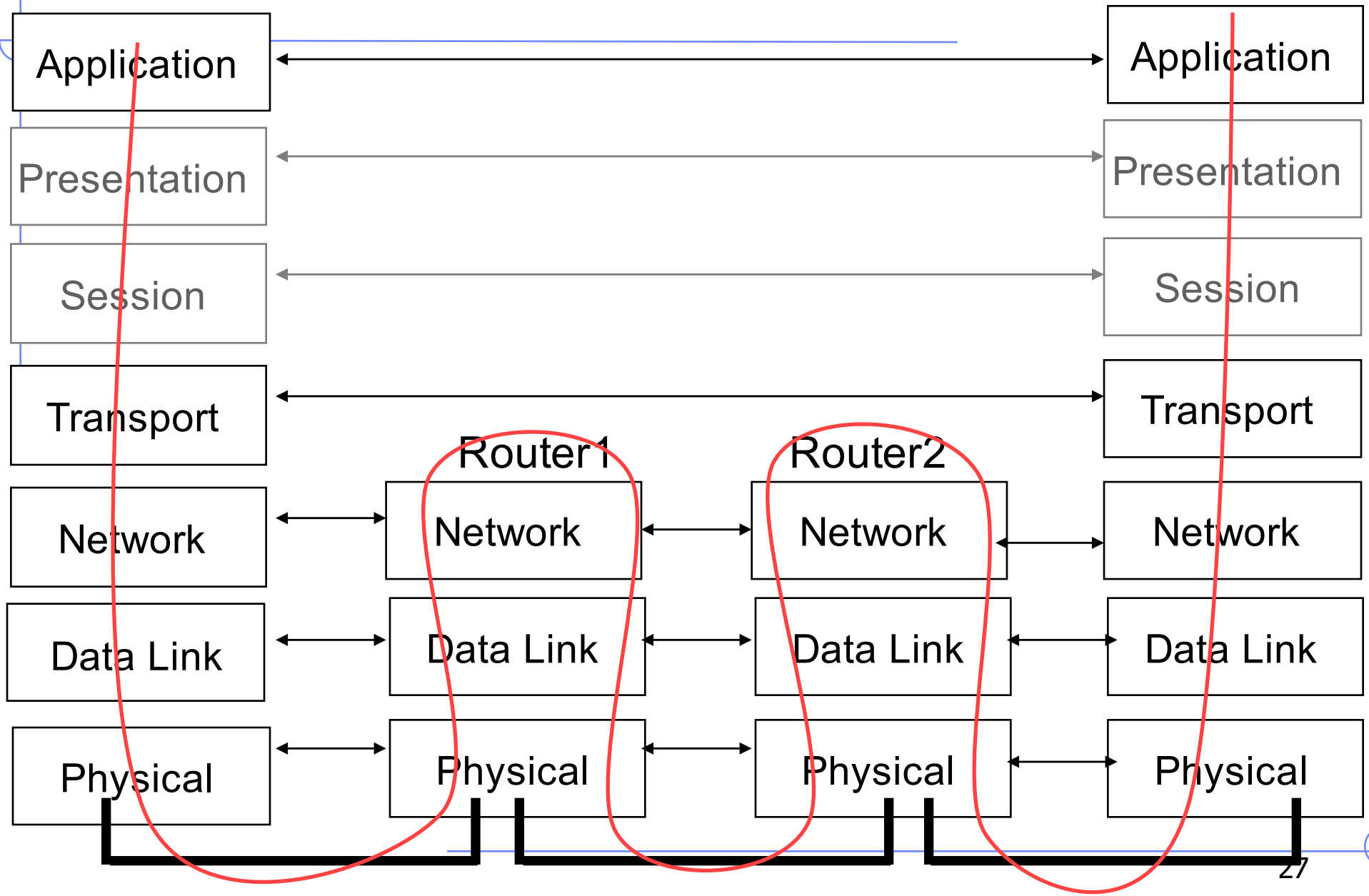
Internet, The Big Picture



The Big Picture



The Big Picture



The OSI Layers
(Open Systems Interconnection)

1. Physical: lowest layer, responsible for transmitting and receiving bits on the media
   (ex: electrical vs optical)
2. Data Link: physical addressing, media access
   (ex: Ethernet)
3. Network: routing across multiple network segments, fragmentation, routing, logical addressing (ex: IP)
4. Transport: data transfer, reliability, streaming, retransmission, etc. (ex: TCP/UDP)
5. Session: connection management
6. Presentation: translation between network and application formats
7. Application: implements application logic

Network Stack – quite literally

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

End-to-End Example

- Should the network guarantee packet delivery ?
  - Think about a file transfer program
  - Read file from disk, send it, the receiver reads packets and writes them to the disk
- If the network guaranteed packet delivery, one might think that the applications would be simpler
  - No need to worry about retransmits
  - But still need to check that the file was written to the remote disk intact
- A check is necessary if nodes can fail
  - Consequently, applications need to be written to perform their own retransmits
  - No need to burden the internals of the network with properties that can, and must, be implemented at the periphery

End-to-End Argument

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

IP

- Internetworking protocol
  - Network layer
- Common address format
- Common packet format for the Internet
  - Specifies what packets look like
  - *Fragments* long packets into shorter packets
  - *Reassembles* fragments into original shape
- IPv4 vs IPv6
  - IPv4 is what most people use
  - IPv6 more scalable and clears up some of the messy parts

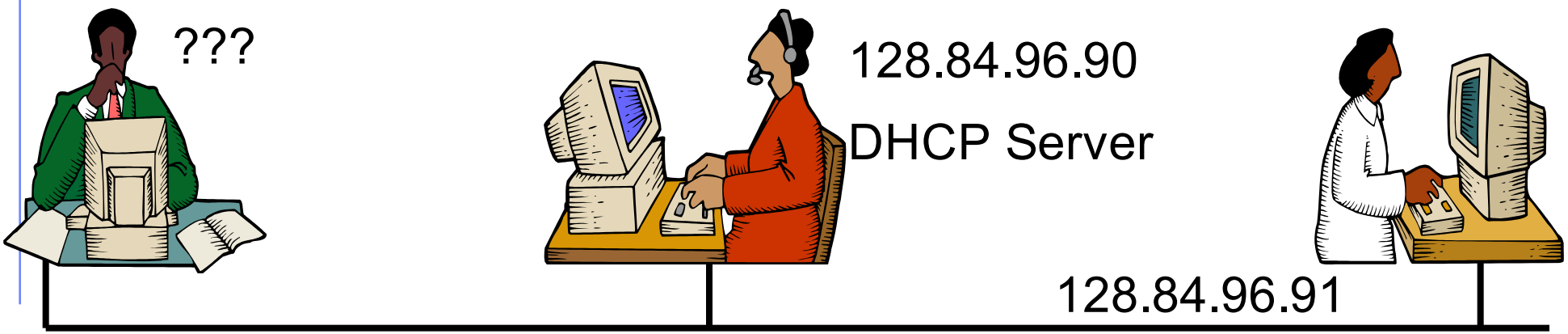
IP Addressing

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

IP "subnetting"

- An IP address consists of a prefix of size *n* and a suffix of size 32 – *n*
  - Either specified by a number, e.g., 128.84.32.00/24
  - Or a "netmask", e.g., 255.255.255.0 (in case n = 24)
- A "subnet" is identified by a prefix and has 2^{32-n} addresses
  - Suffix of "all zeroes" or "all ones" reserved for broadcast
  - Big subnets have a short prefix and a long suffix
  - Small subnets have a long prefix and a short suffix

Addressing & DHCP



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

"Your IP is 128.84.96.89 for the next 24 hours"

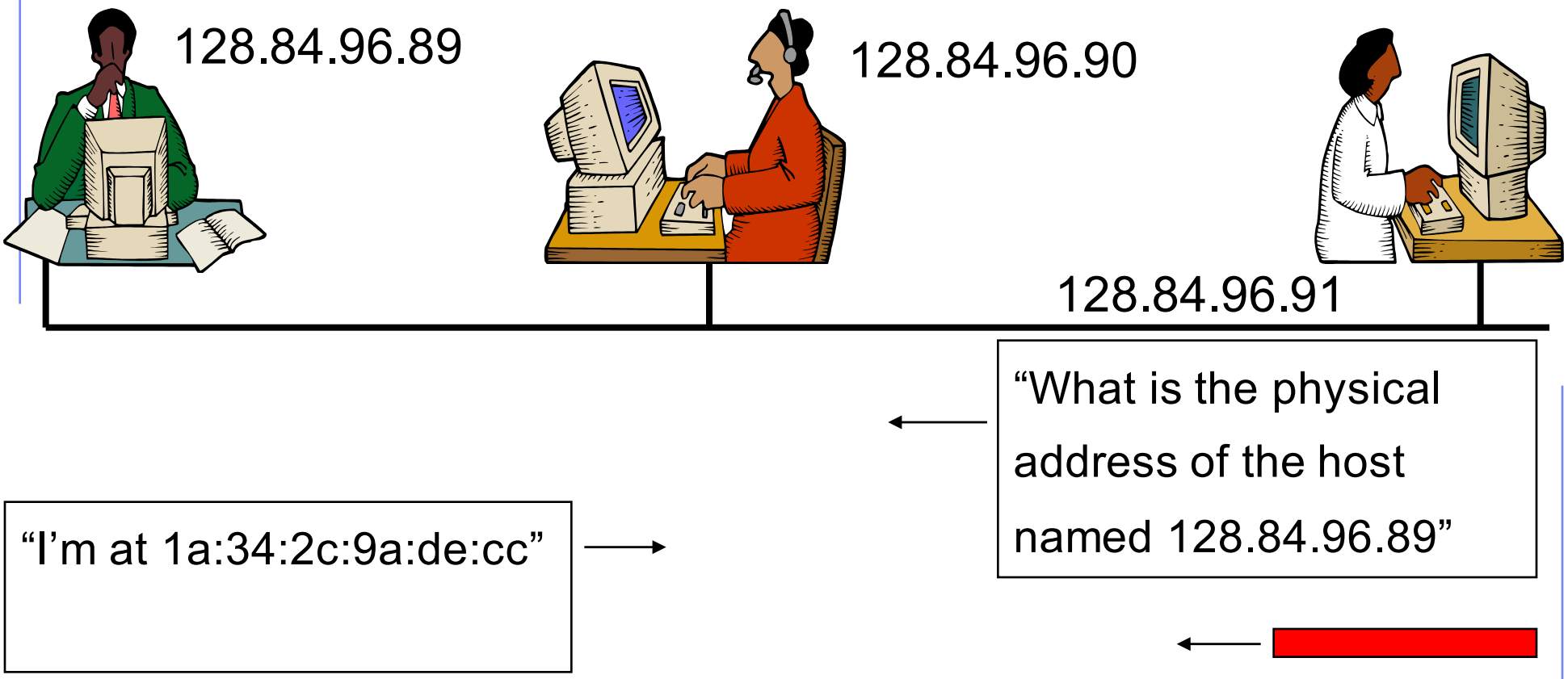
DHCP is used to discover IP addresses (and more)

DHCP = Dynamic Host Configuration Protocol

DHCP

- Each LAN (usually) runs a DHCP server
  - you probably run one at home inside your "router box"
- DHCP server maintains
  - the IP subnet that it owns (say, 128.84.245.00/24)
  - a map of IP address <-> MAC address
    - possibly with a timeout (called a "lease")
- When a NIC comes up, it broadcasts a DHCPDISCOVER message
  - if MAC address in the map, respond with corresponding IP address
  - if not, but an IP address is unmapped and thus available, map that IP address and respond with that
- DHCP also returns the netmask
- Note: NICs can also be statically configured and don't need DHCP

Addressing & ARP



- ARP is used to discover MAC addresses *on same subnet*
  - ARP = Address Resolution Protocol

Scale?

- ARP and DHCP only scale to single subnet
- Need more to scale to the Internet!

IPv4 packet layout

0		1	2	3
Version	IHL	TOS	Total Length	
Identification			Flags	Fragment Offset
TTL		Protocol	Header Checksum	
Source Address				
Destination Address				
Options				
Payload				

IP Header Fields

- Version (4 bits): 4 or 6
- IHL (4 bits): Internet Header Length in 32-bit words
  - usually 5 unless options are present
- TOS (1 byte): type of service (not used much)
- Total Length (2 bytes): length of packet in bytes
- Id (2 bytes), Flags (3 bits), Fragment Offset (13 bits)
  - used for fragmentation/reassembly. Stay tuned
- TTL (1 byte): Time To Live. Decremented at each hop
- Protocol (1 byte): TCP, UDP, ICMP, …
- Header Checksum (2 bytes): to detect corrupted headers

IP Fragmentation

- Networks have different maximum packet sizes
  - "MTU": Maximum Transmission Unit
    - Big packets are sometimes desirable – less overhead
    - Huge packets are not desirable – reduced response time for others
- High-level protocols could try to figure out the minimum MTU along the network path, but
    - Inefficient for links with large MTUs
    - The route can change underneath
- Consequently, IP can transparently fragment and reassemble packets

IP Fragmentation Mechanics

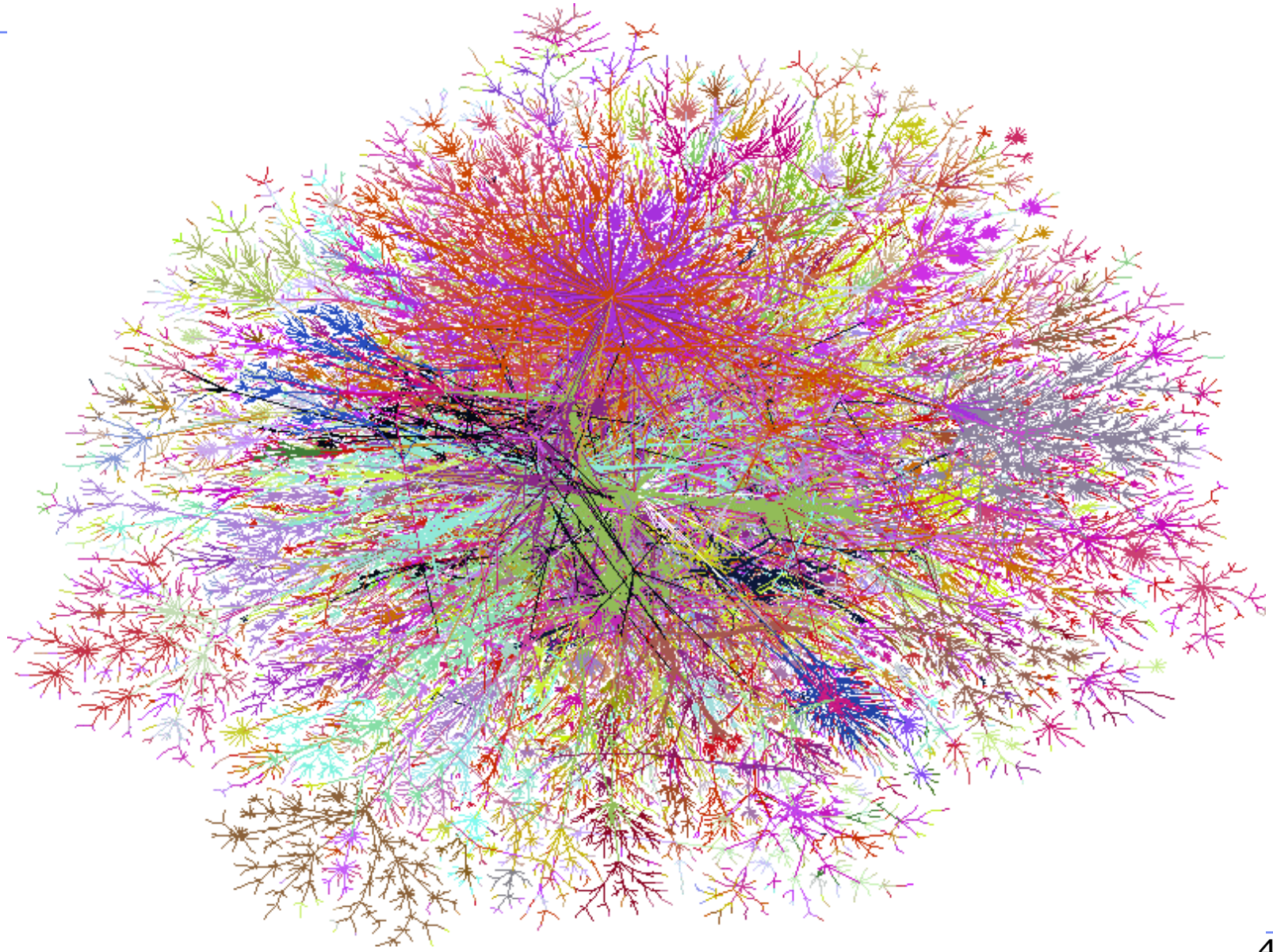
- Source assign each datagram an "identification"
- At each hop, IP can divide a long datagram into N smaller datagrams
- Sets the More Fragments bit except on the last packet
- Receiving end puts the fragments together based on *Identification* and *More Fragments* and *Fragment Offset (times 8)*
- Routers throw out fragments after a certain amount of time if they have not be reassembled

IP Options (not well supported)

- Source Routing: The source specifies the set of hosts that the packet should traverse
- Record Route: If this option appears in a packet, every router along a path attaches its own IP address to the packet
- Timestamp: Every router along the route attaches a timestamp to the packet
- Security: Packets are marked with user info, and the security classification of the person on whose behalf they travel on the network
  - Most of these options pose security holes and are generally not implemented

Routing

The Internet is Big…



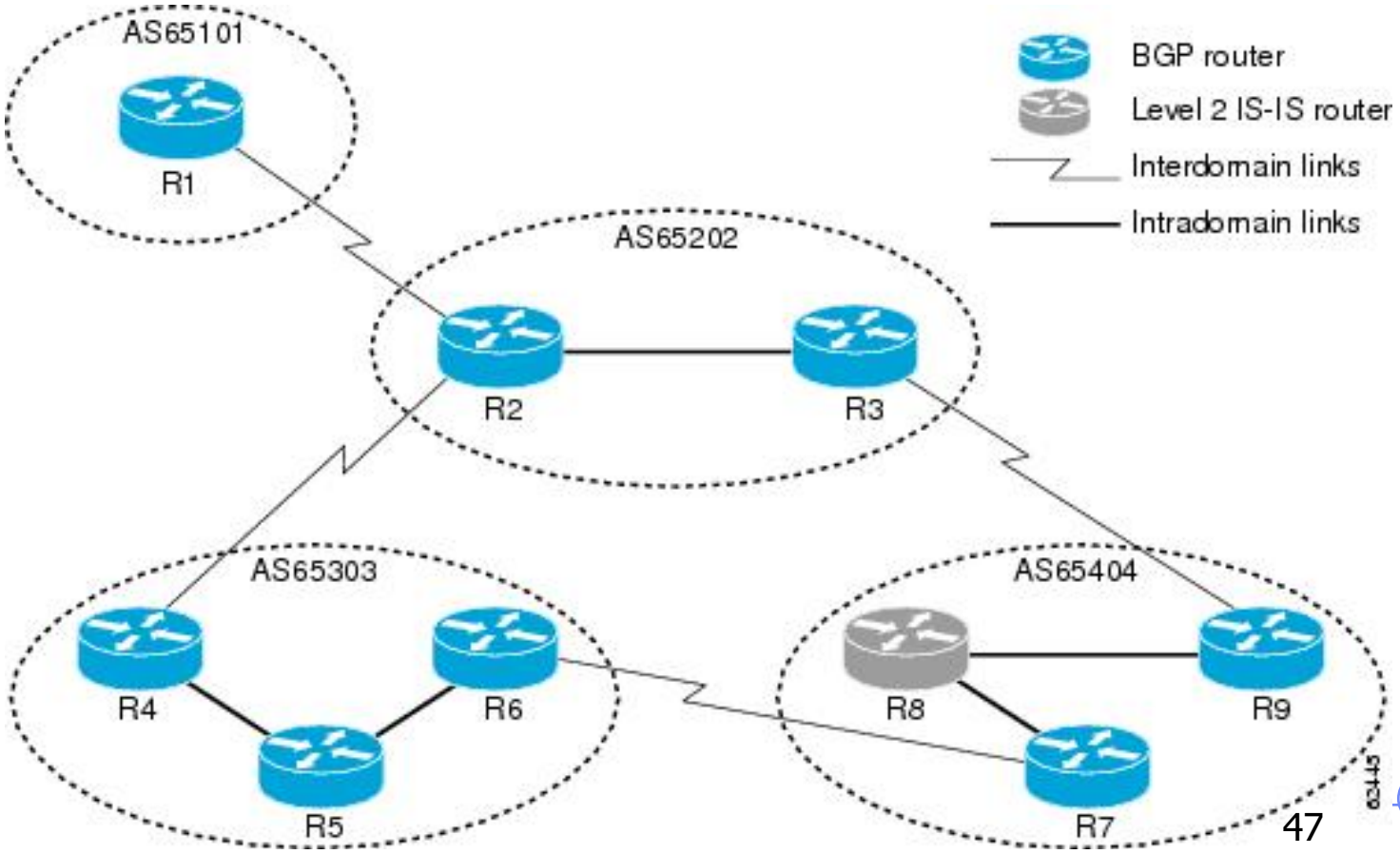
Routing

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

Internet Protocol (IP)

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

Graph of subgraphs



Autonomous Systems

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

Thus routing is hierarchical!

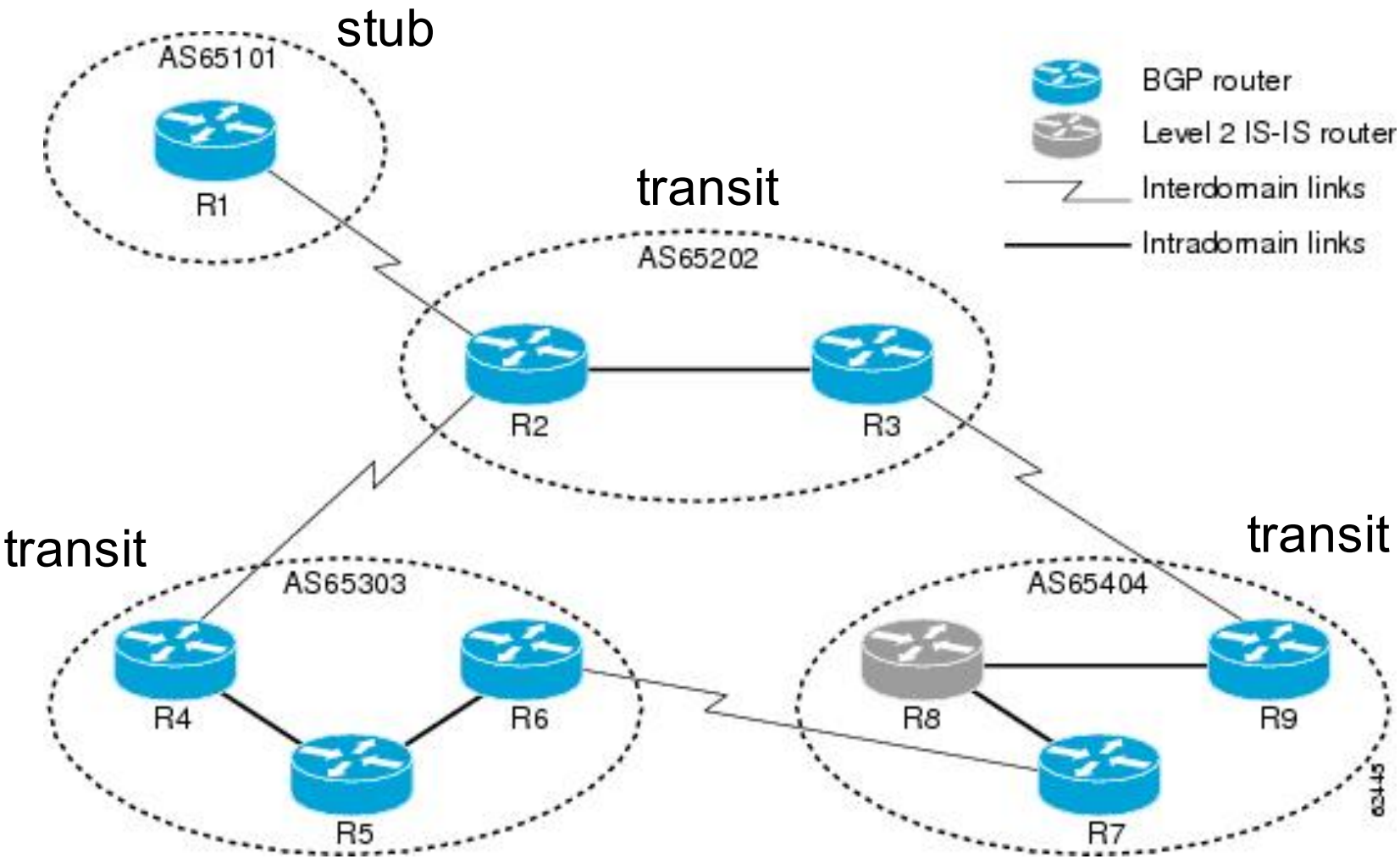
Three steps:

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

Internet Routing, observations

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

Transit ASs


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

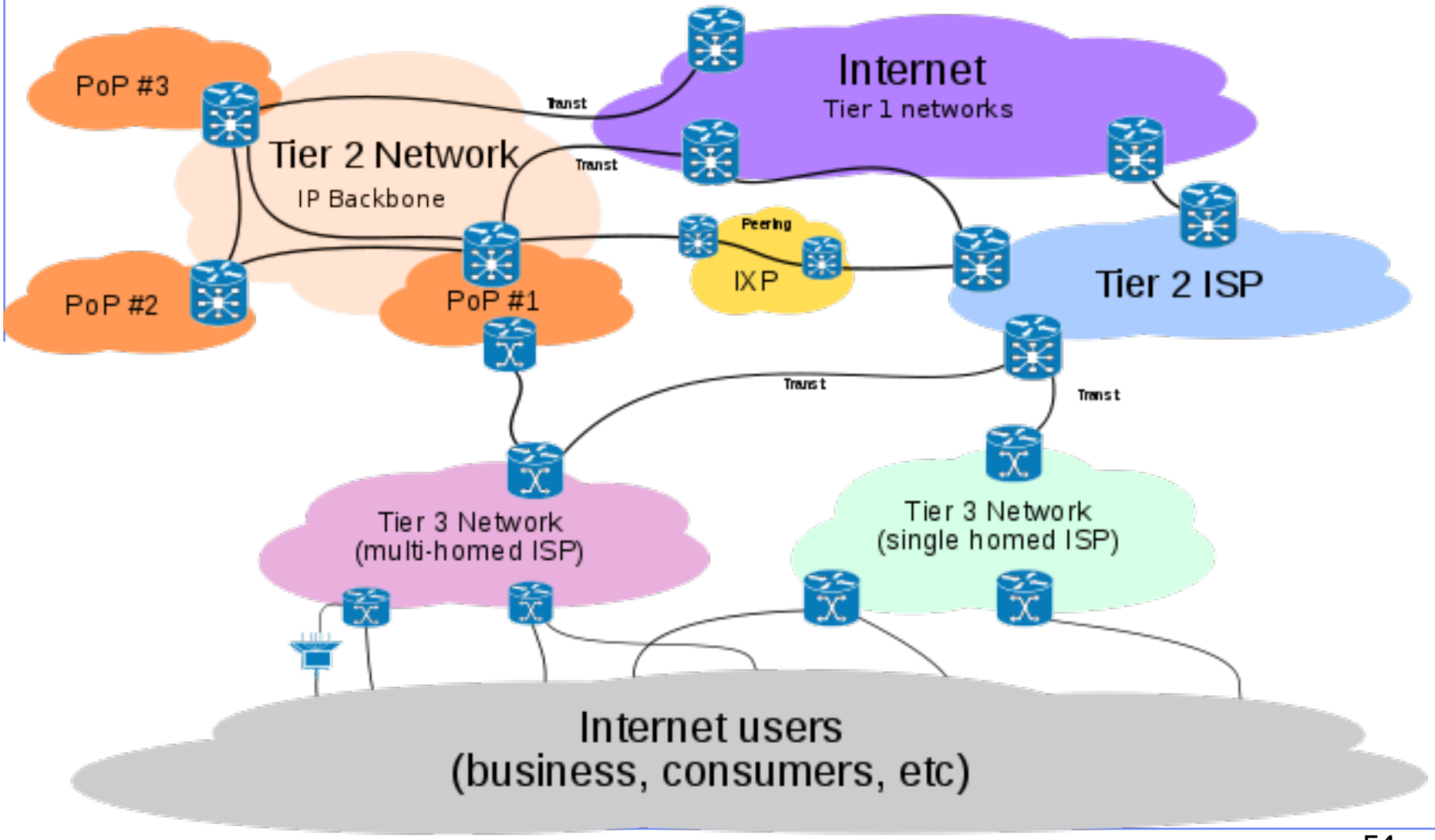
What's an ISP?

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

AS Tiers

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

Tiers


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

Routers (Layer-3 Switches)

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

Routing Table

- Maps IP address to interface or port and to MAC address
- Longest Prefix Matching
- Your laptop/phone has a routing table too!

Address	IF or Port	MAC
128.84.216/23	en0	`c4:2c:03:28:a1:39`
127/8	lo0	`127.0.0.1`
128.84.216.36/32	en0	`74:ea:3a:ef:60:03`
128.84.216.80/32	en0	`20:aa:4b:38:03:24`
128.84.217.255/32	en0	`ff:ff:ff:ff:ff:ff`

Router Function

often implemented in hardware

```
for ever:
    receive IP packet p
    if isLocal(p.dest): return localDelivery(p)
    if --p.TTL == 0: return dropPacket(p)
    matches = { }
    for each entry e in routing table:
        if p.dest & e.netmask == e.address & e.netmask:
            matches.add(e)
    bestmatch = matches.maxarg(e.netmask)
    forward p to bestmatch.port/bestmatch.MAC
```

Routing Loops?

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

How are these routing tables constructed?

- For end-hosts, mostly DHCP and ARP as discussed before
- For routers, using a “routing protocol”

Model for Routing

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

Layer-3 Routing Protocols

- Essentially three types used in practice
  - Link State (e.g., OSPF, IS-IS)
  - Distance Vector (e.g., RIP, IGRP)
  - Path Vector (e.g., BGP)

Link State Routing

- Each node maintains a map of the entire network
- Upon neighbor changes, a node floods its identifier, along with its direct neighbors and a version number, on the network
  - gossip-style convergence
- Recipients update their maps accordingly
- Each node locally runs Dijkstra's algorithm to compute a shortest distance tree with itself as root.
- On receipt of a message, a node uses this graph to select an outgoing neighbor for the next hop.

Most common examples

- OSPF (Open Shortest Path First)
  - Runs on IP, making it easy to deploy
- IS-IS (Intermediate System to Intermediate System)
  - Less chatty, possibly more scalable than OSPF

Distance Vector Routing

- Each node maintains, for each peer node in the network, one outgoing neighbor and the hop count to that peer.
- Each node periodically shares its table with its neighbors.
- Upon receipt, a node uses the neighbor's table to update its own.
  - E.g., if U had a route to Z of length 10 via neighbor X, and U then learns from neighbor Y that it has a route to Z of length 5, then U updates its table to reflect that it has a route of length 6 to Z via neighbor Y.
  - This protocol converges to shortest paths, and is a variant of "Bellman-Ford".
- If a node loses a connection to a neighbor, it notifies its other neighbors so they can remove routes through that node.

Most Common Examples

- RIP (Routing Information Protocol)
  - limited hop count of 15
- IGRP (Interior Gateway Routing Protocol)
  - classful and proprietary

Neither is used much.

Path Vector Routing

- Like distance vector, but each node maintains, for each peer node in the network, an *entire path* to that peer.
- Each node periodically shares its table with its neighbors.
- Upon receipt, a node uses the neighbor's table to update its own.
- If a node loses a connection to a neighbor, it notifies its other neighbors so they can remove routes through that node.
- For this reason each node really has to maintain a set of routes to each other node.

Most Common Example

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

Why BGP?

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

BGP Policy Implementation

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

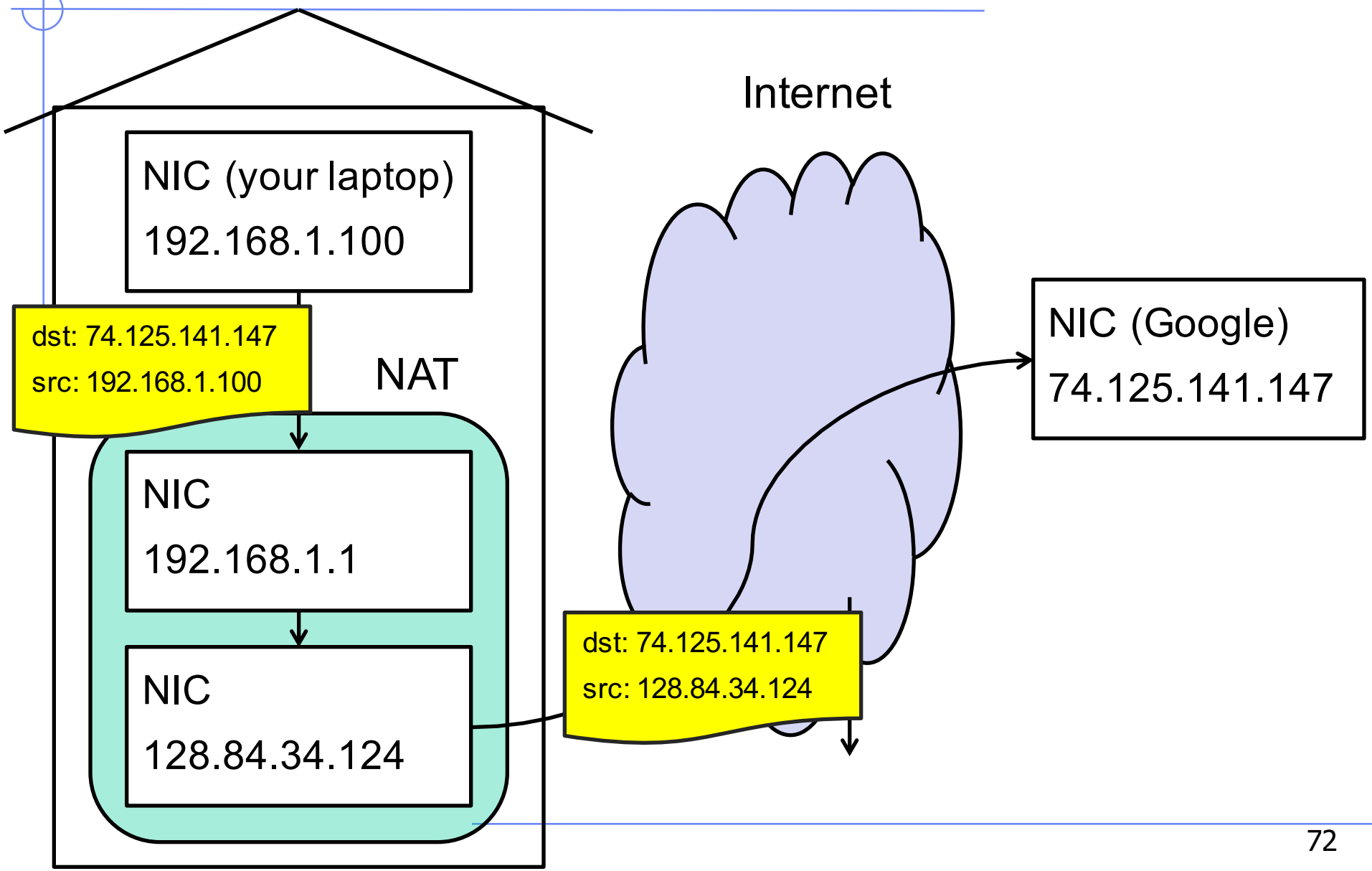
Network Address Translation

- IPv6 adoption is very slow, and IPv4 addresses have run out
- NAT allows entire sites to use a single globally routable IPv4 address for a collection of machines
  - exploits the sparsely used 16-bit TCP/UDP port number space
- A "NAT box" keeps a table that maps global TCP/IP addresses into local ones
- Overwrites the local source address with the globally addressable address

“Private” IP addresses

- The IPv4 addresses 10.x.x.x and 192.168.x.x are freely available for anybody to use
- Many machines have the IP address 192.168.0.100, for example

From your laptop to Google…



Vice versa: punching holes or "game ports"

- When an external host tries to send a message to one of your machines in your house, it first arrives at the NAT box
  - Because you advertise your global IP address
- How does the NAT box know which of your machines to forward the message to?
- Answer: a table. It is indexed by the destination TCP or UDP port in the message

Application Layer
Transport Layer
Network Layer
Link Layer
Physical Layer

Transport Layer

Transport Layer

- For the most part, Network Layer interface not exposed to applications
- Applications see the Transport Layer (UDP, TCP) or higher layers (HTTP, RPC, …)
- Most popular transport layer protocols:
  - UDP: User Datagram Protocol
    - Perhaps better named "Unreliable Datagram Protocol"
  - TCP: Transport Control Protocol
    - Perhaps better name "Trusty Connection Protocol"??

UDP

- User Datagram Protocol
- IP goes from host to host
- We need a way to get datagrams from one process to another
- How do we identify processes on the hosts?
  - Assign *port numbers*
  - E.g. port 13 belongs to the time service, port 88 is Kerberos, etc.

UDP Packet Layout

IP	Version	IHL	TOS	Total Length	
	Identification			Flags	Fragment Offset
	TTL		Protocol = 17	Header Checksum	
	Source Address				
	Destination Address				
UDP	Source Port			Destination Port	
	Length			Checksum	
	Data				

UDP adds Ports, Data Length and Data checksum

UDP

- UDP is unreliable
  - A UDP packet may get dropped at any time
  - It may get duplicated
  - A series of UDP packets may get reordered
- Unreliable datagrams are the bare-bones network service
  - Good to build on, esp for multimedia applications
- Most applications would prefer reliable, in-order delivery
  - Some apps can ignore these effects and still function

TCP

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

Application Layer
Transport Layer
Network Layer
Link Layer
Physical Layer

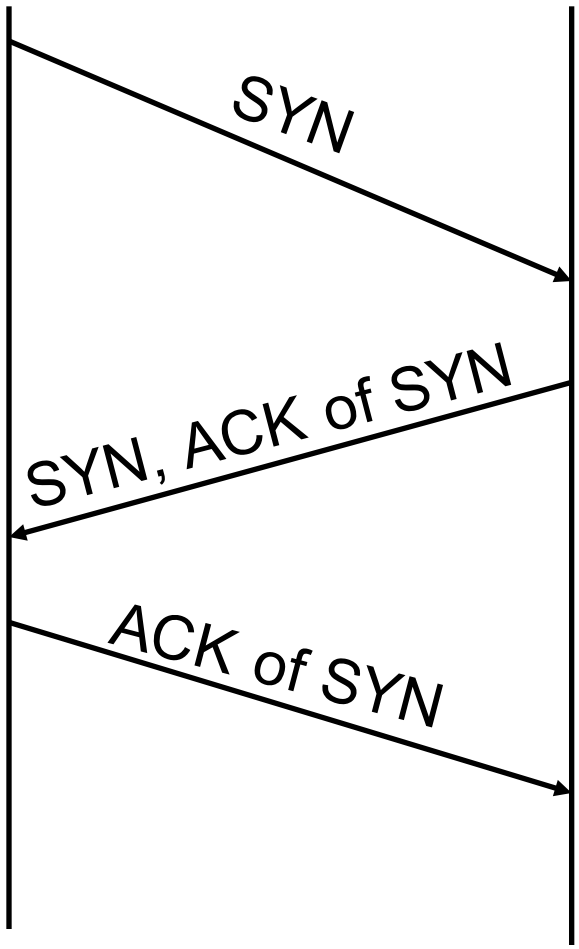
TCP/IP Packets

IP	Version	IHL	TOS	Total Length	
	Identification			Flags	Fragment Offset
	TTL		Protocol = 6	Header Checksum	
	Source Address				
	Destination Address				
TCP	Source Port			Destination Port	
	Sequence Number				
	Acknowledgment Number				
	Hdr-Len ACK|URG|SYN|FIN|RST			Window	
	Checksum			Urgent Pointer	
	Options			Padding…	
	Data				

TCP Packets

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

TCP Connections



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

SYN = Synchronize

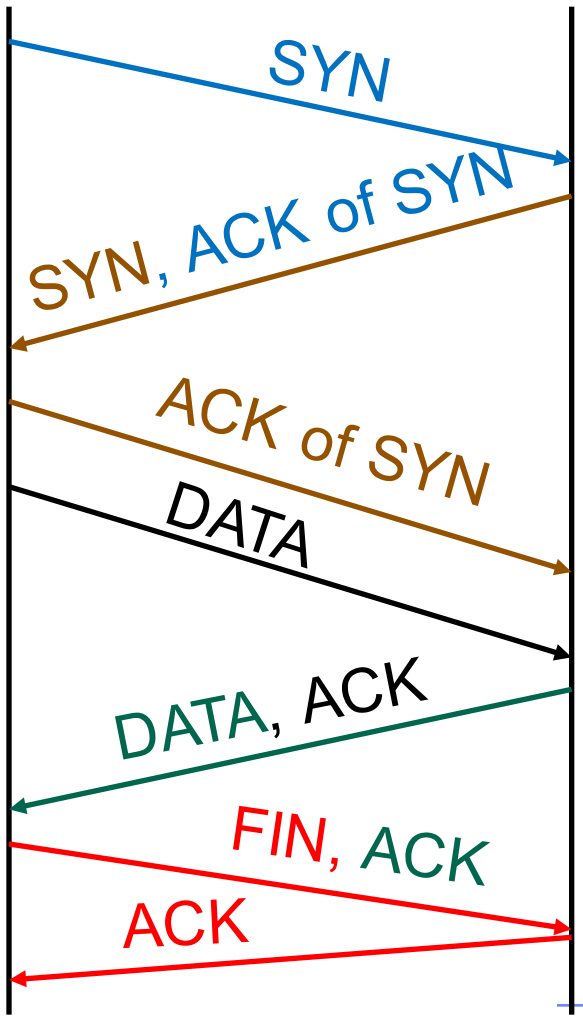
ACK = Acknowledgement

TCP Handshakes

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

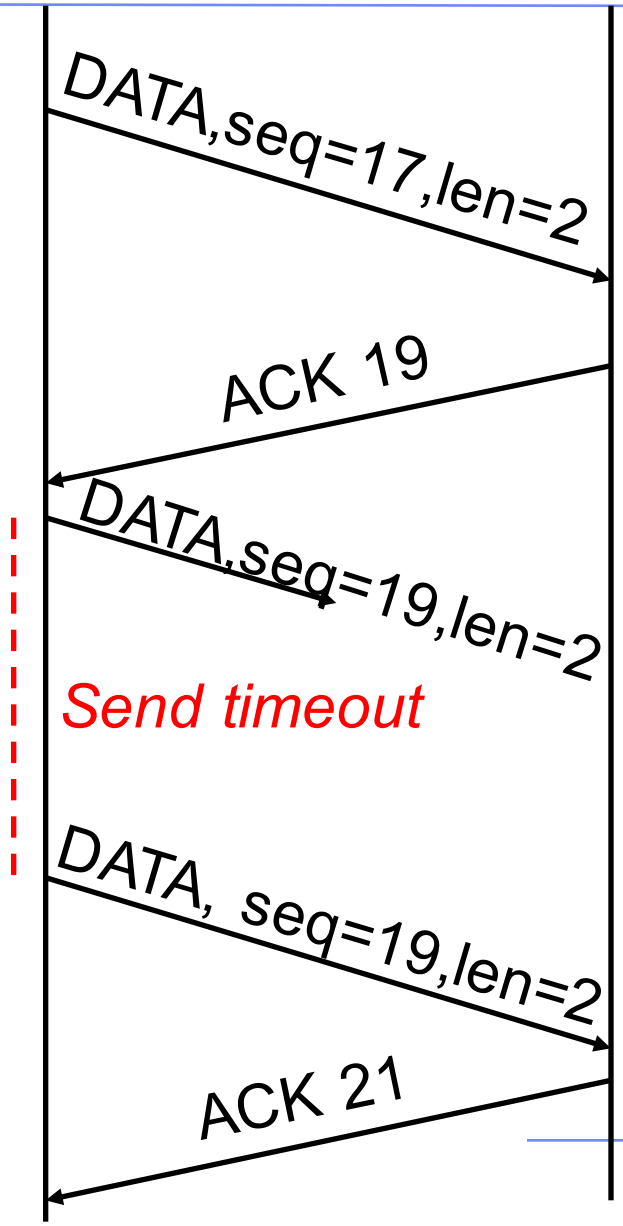
- Both sides will have seen one packet from the other side, thus know what the first seqno ought to be
- SYN-ACK also typically carries a new port for the server
- Both sides will know that the other side is ready to receive

Typical TCP Usage



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

Reliable transport



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

TCP timeouts

What is a good timeout period ?

- Want improved throughput w/o unnecessary transmissions

AverageRTT := (1 - α) AverageRTT + α LatestRTT

AverageVar := (1 - β) AverageVar + β LatestVar

where LatestRTT = (ack_receive_time – send_time),

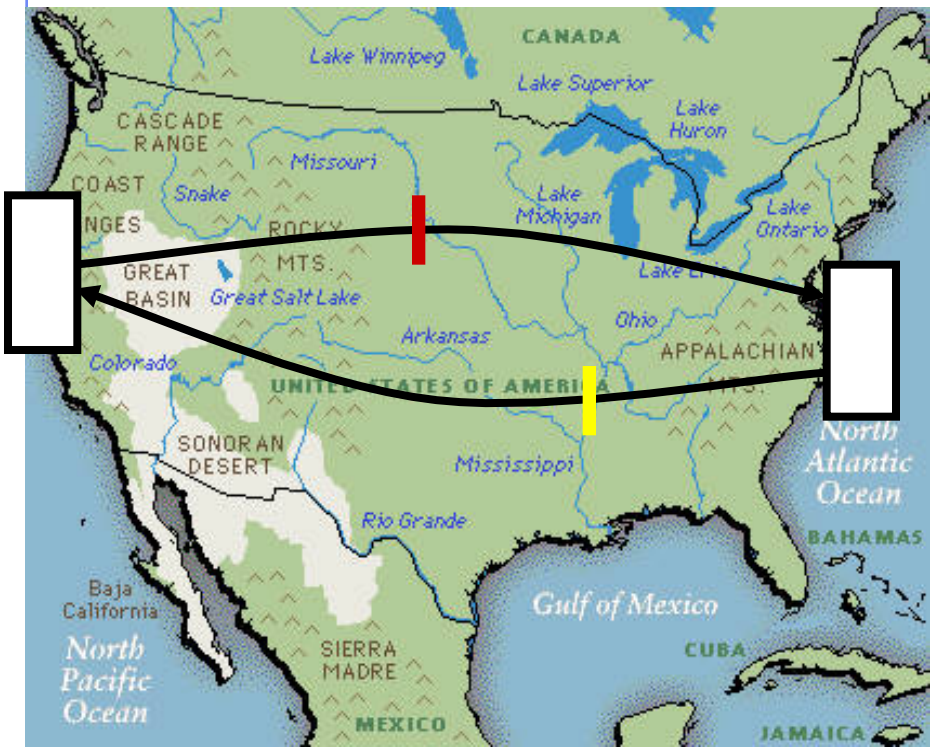
LatestVar = |LatestRTT – AverageRTT|,

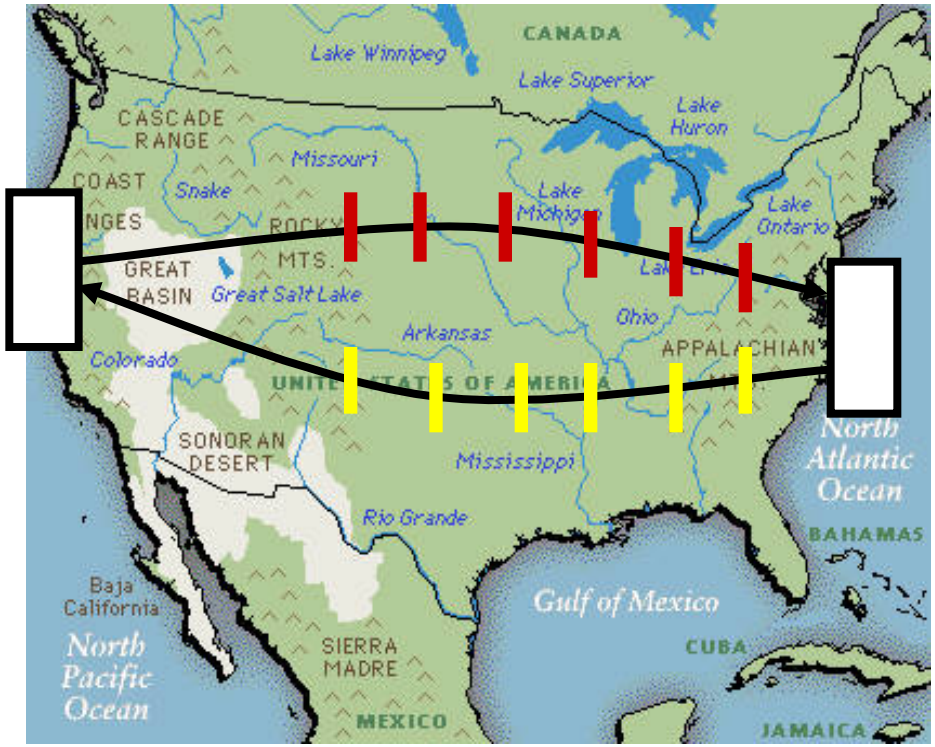
α = 1/8, β = 1/4 typically.

Timeout := AverageRTT + 4*AverageVar

→ Timeout is thus a function of RTT and variance

TCP Windows



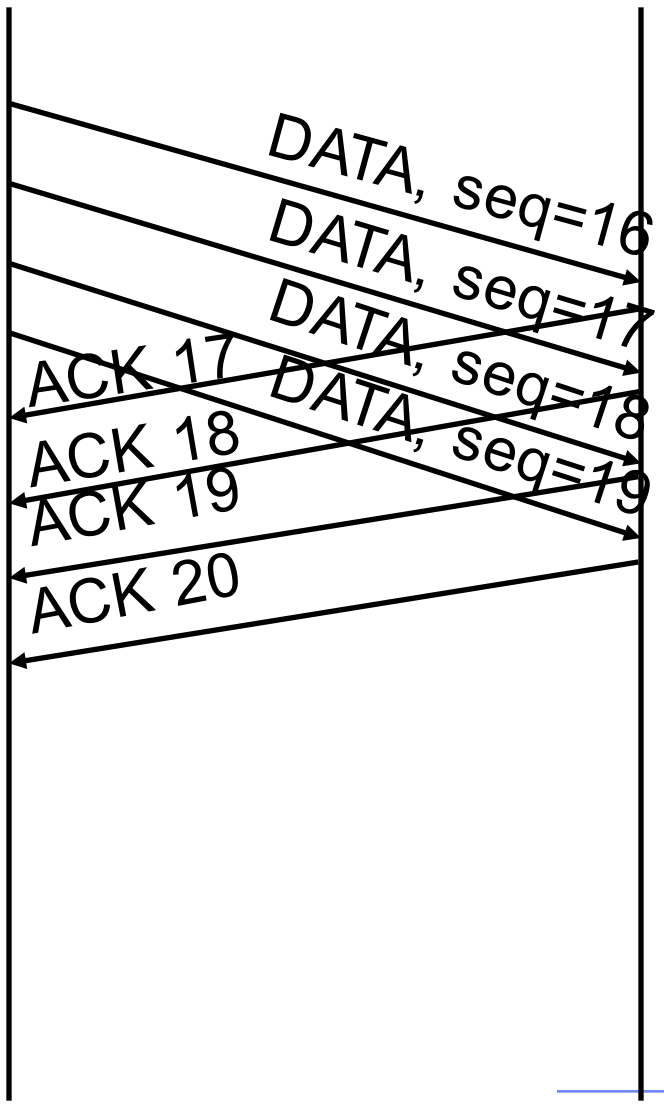


Multiple outstanding packets can increase throughput

How much data "fits" in a pipe?

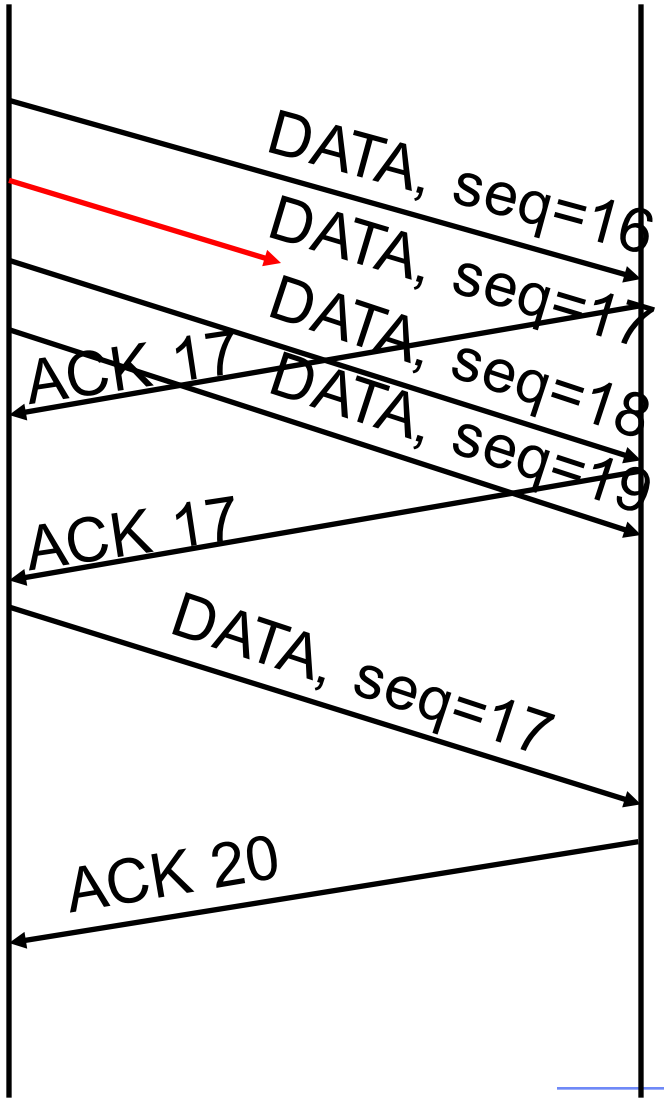
- Suppose the b/w is b bytes / second
- Suppose the RTT is r seconds
- Suppose an ACK is a small message
  - you can send b * r bytes before receiving an ACK for the first byte
- But b/w and RTT are both variable…

TCP Windows



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

TCP Windows and Fast Retransmit

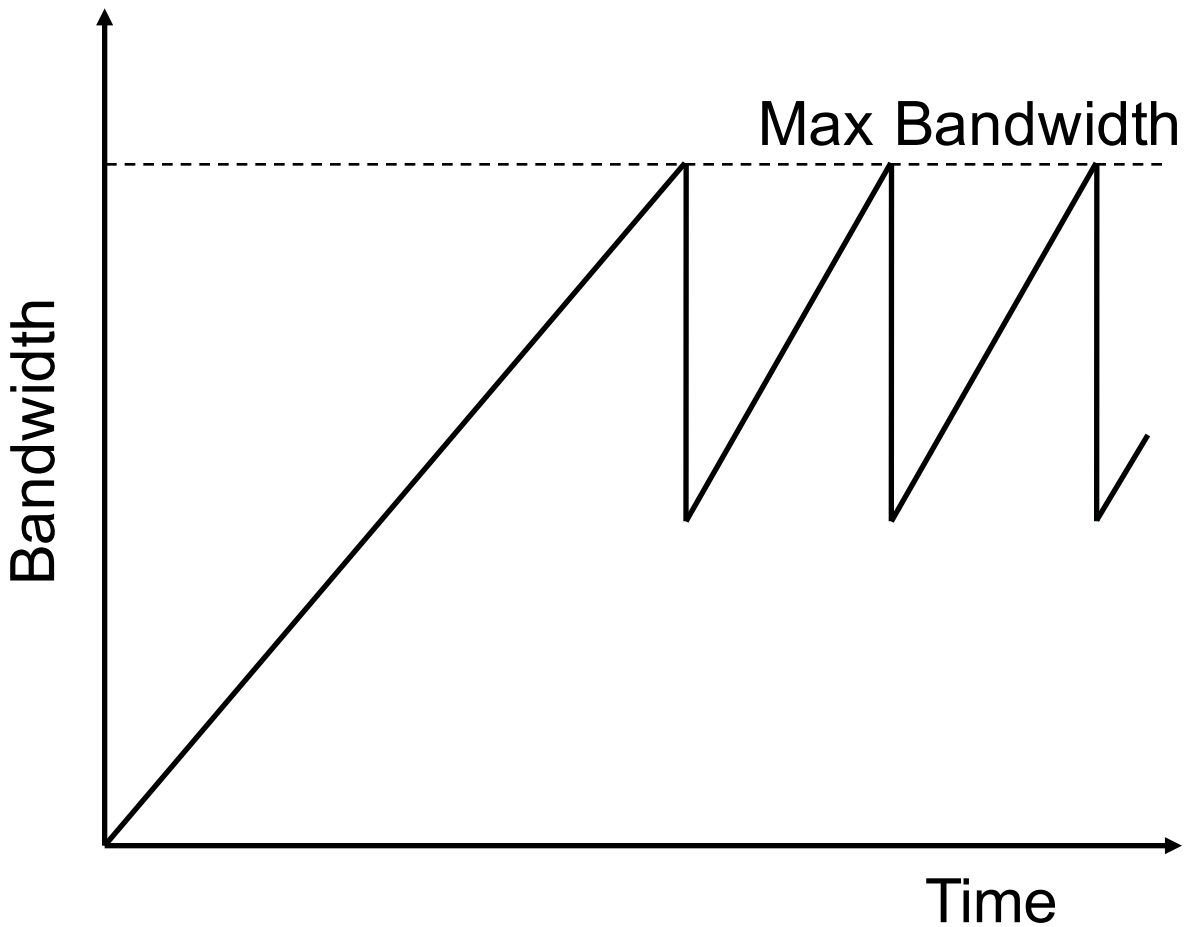


- When receiver detects a lost packet (i.e. a hole in the seqno space), it acks the last seqno it successfully received
- Sender can quickly detect that a loss occurred without waiting for a timeout

TCP Congestion Control

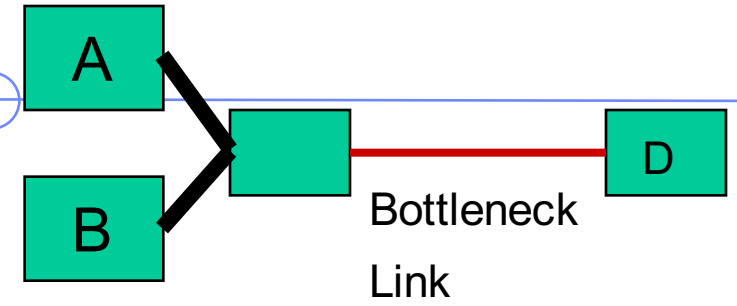
- TCP typically increases its window size by one MTU (Maximum Transmission Unit) every RTT
- It typically halves the window size when a packet drop occurs
  - A packet drop is evident from the acknowledgments
- Therefore, it will slowly build up to the max bandwidth, and hover around the max
  - It doesn't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

TCP Window Size

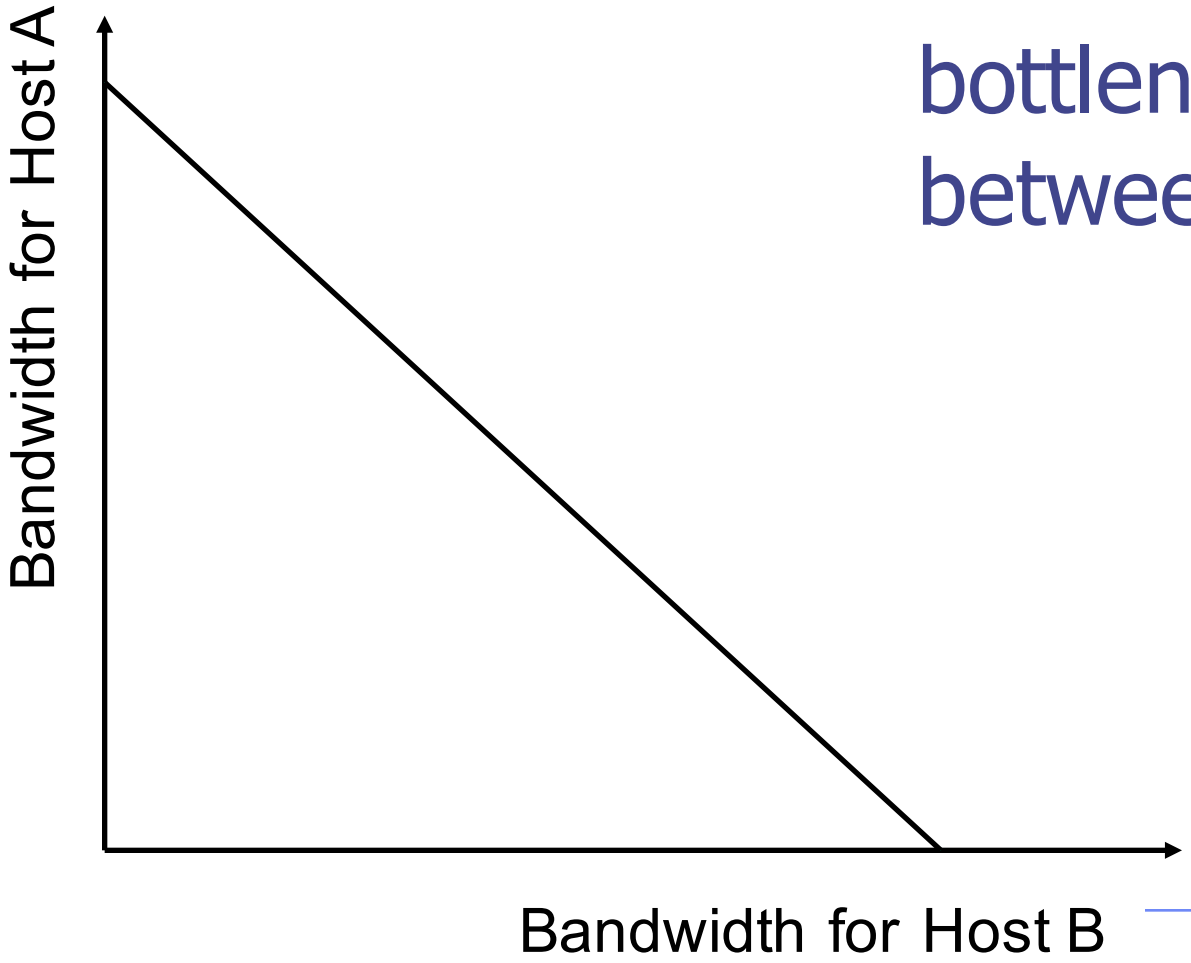


- Linear increase
- Exponential backoff
- Assuming no other losses in the network except those due to bandwidth

TCP Fairness



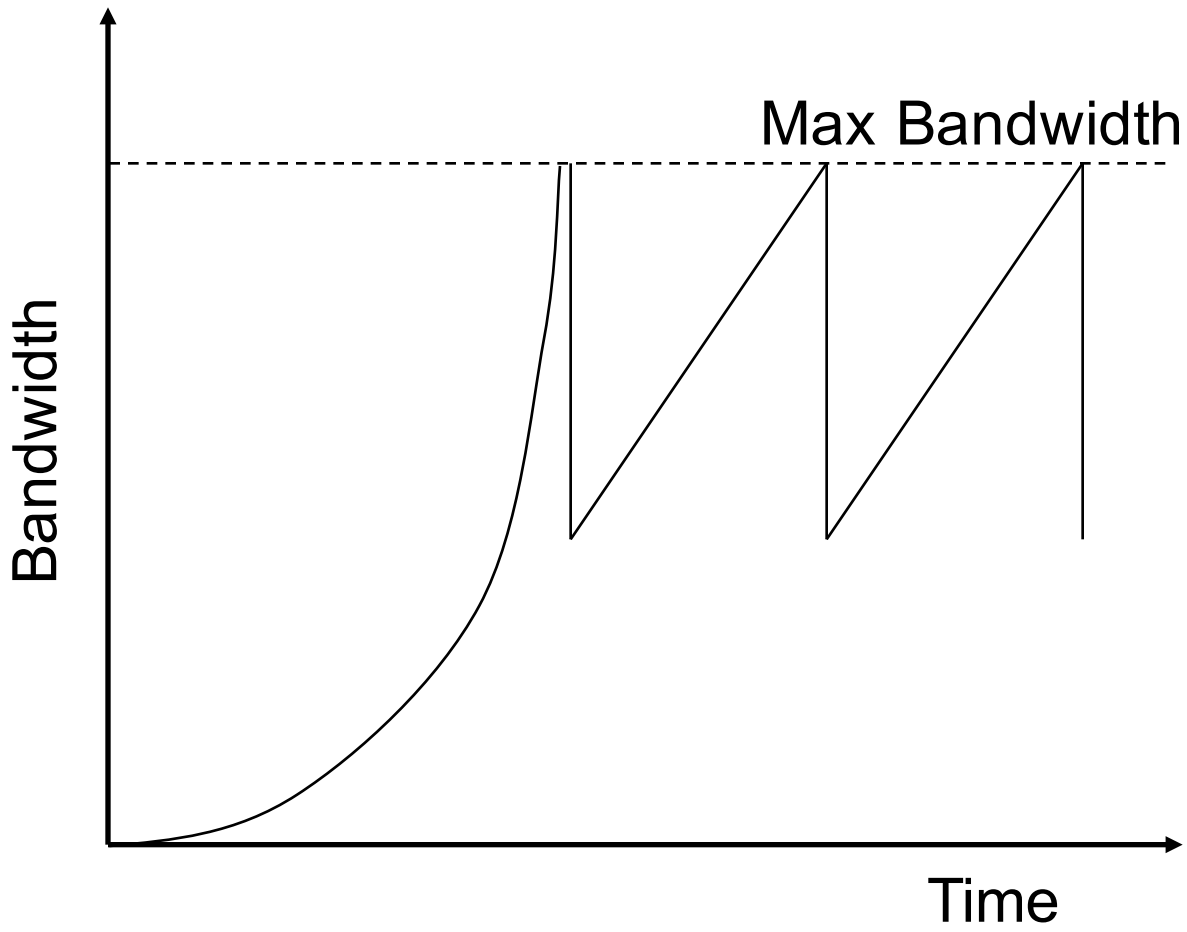
- Want to share the bottleneck link fairly between two flows



TCP Slow Start

- Linear increase takes a long time to build up a window size that matches the link bandwidth*delay
- Most file transactions are not long enough
- Consequently, TCP can spend a lot of time with small windows, never getting the chance to reach a sufficiently large window size
- Fix: Allow TCP to build up to a large window size initially by increasing the window size linearly for each ack received
  - Effectively doubling the window size until first loss

TCP Slow Start



- Initial phase of exponential increase
- Assuming no other losses in the network except those due to bandwidth

TCP Summary

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

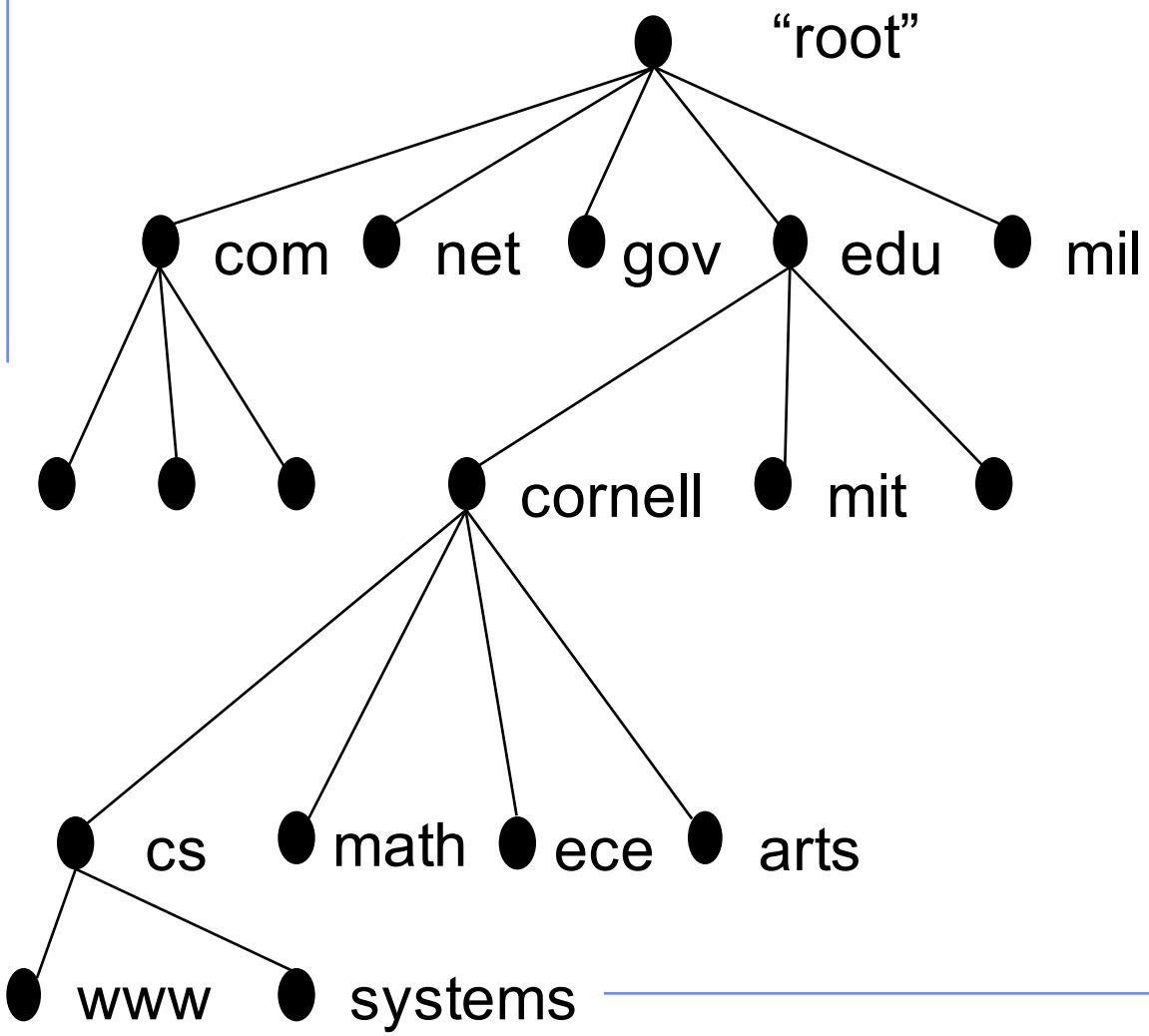
Application Layer
Transport Layer
Network Layer
Link Layer
Physical Layer

Application Layer

DNS

- Protocol for converting textual names to IP addresses
  - www.cnn.com = 207.25.71.25
- Namespace is hierarchical, i.e. a tree.
- Names are separated by dots into components
  - Not to be confused with dots in IP addresses. If anything, the order of least significant to most significant is reversed!
  - Components are looked up from the right to the left

DNS Tree

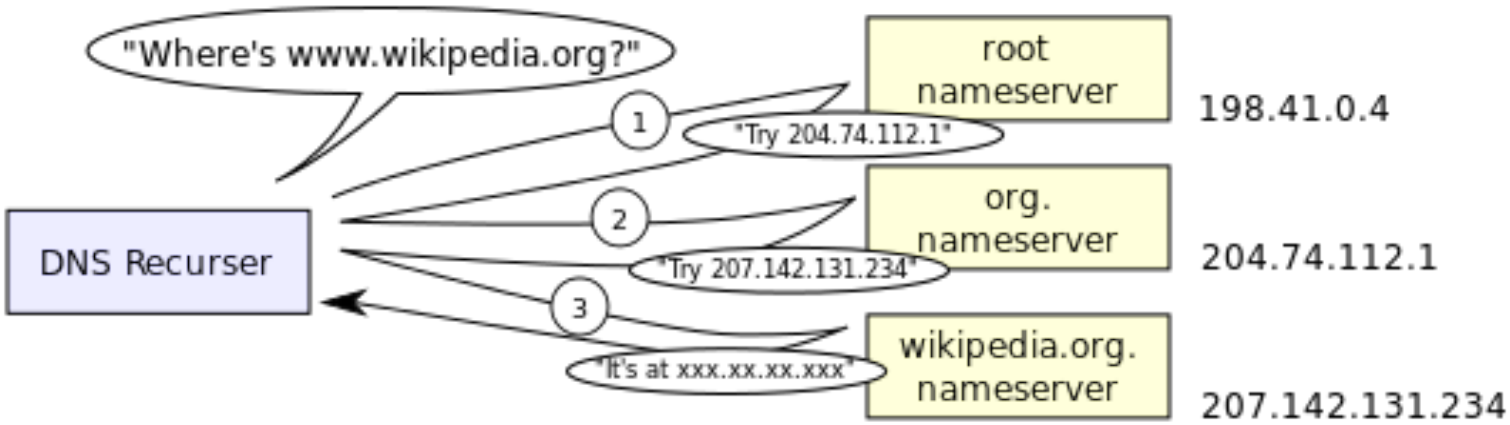


- All siblings must have unique names
- Root is owned by ICANN
- Lookup occurs from the top down
- DNS stores arbitrary tuples (***resource records***)
- The address field contains the IP address, other fields contain mail routing info, owner info, etc.
- One field stores the cache timeout value

DNS Lookup

1. the client asks its local name server
   - Address acquired with DHCP or statically configured
2. the local name server asks one of the *root name servers*
3. the root name server replies with the address of the authoritative name server
4. the server then queries that name server
5. repeat until final host is reached
6. each step caches result until timeout expires

Example (from wikipedia)



DNS Lessons

- Simple, hierarchical namespace works well
  - Can name anything, can share names
- Scales OK
  - Caching
  - Even though it was meant to be hierarchical, people like short names, and use it like a flat namespace
- Arbitrary tuple database
  - Can delegate selected services to other hosts
  - Email is a good example
- Little or no security!
  - DNSSEC could help

Remote Procedure Call

Client/Server Paradigm

Common model for structuring distributed computation

- ***Server:*** program (or collection of programs) that provides some *service*, *e.g.*, file service, name service
  - may exist on one or more nodes
- ***Client:*** program that uses the service

Typical pattern:

1. Client *binds* to the server, i.e., locates it in the network and establishes a connection
2. Client sends *requests* to perform actions; sends messages that indicate which service is desired, along with parameters
3. Server returns a *response*

The Pro and Cons of Messages

+ Very flexible communication
- Problems with messages:
  - require that programmer worry about message formats
  - must be packed and unpacked
  - have to be decoded by server to figure out what is requested
  - may require special error handling functions

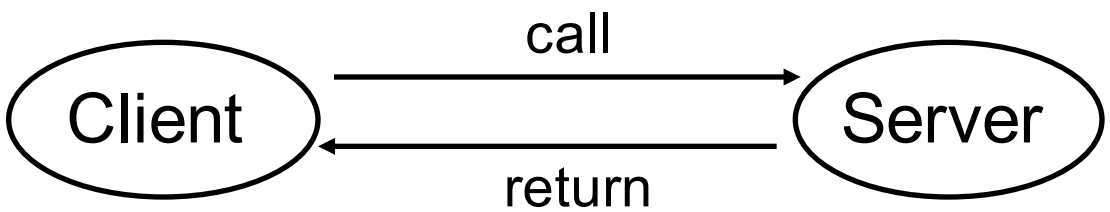
Messages are not a natural programming model for most programmers.

Procedure Call

A more natural way to communicate:

- every language supports it
- semantics are well defined and understood
- natural for programmers to use

- Basic idea: define a server as a module that *exports* a set of procedures that can be called by client programs
- To use the server, the client just does a procedure call, as if it were linked with the server



(Remote) Procedure Call

Goal: use procedure call as a model for distributed communication

Issues:

- how do we make this invisible to the programmer?
- what are the semantics of parameter passing?
- how is binding done (locating the server)?
- how do we support heterogeneity (OS, architecture, programming language)

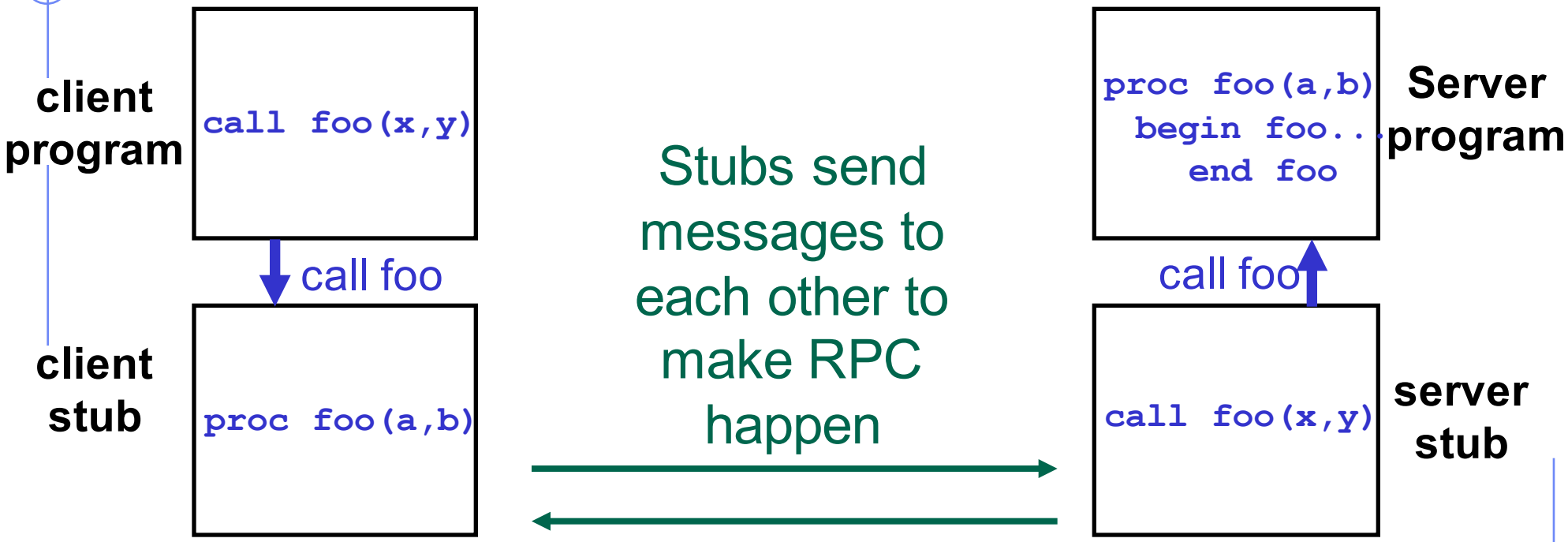
Remote Procedure Call (RPC)

- Basic model for RPC was described by Birrell and Nelson in 1980, based on work done at Xerox PARC.
- Goal: make RPC look as much like local PC as possible
- Used computer/language support
- 3 components on each side:
  - user program (client or server)
  - set of *stub* procedures
  - RPC runtime support

Basic process for building a server

- Server program defines the server's interface using an *interface definition language* (IDL)
- **IDL:** specifies the names, parameters, and types for all client-callable server procedures
- *Stub compiler* reads the IDL and produces two stub procedures for each server procedure: a client-side stub and a server-side stub
- Server writer writes the server and links it with the server-side stubs; Client writes her program and links it with the client-side stubs
- Stubs: manages all details of the remote communication between client and server

RPC Stubs



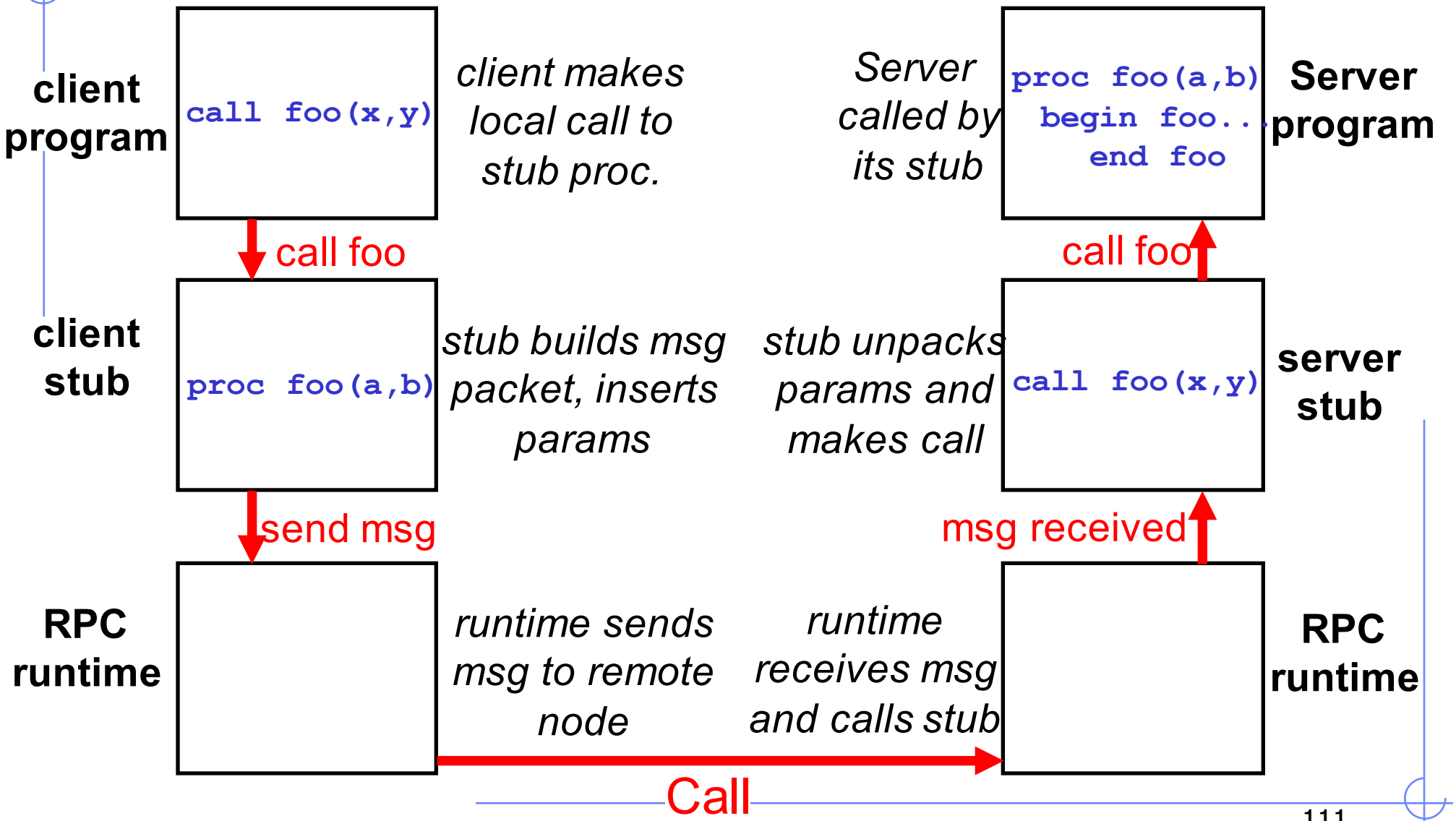
Client-side stub:

- Looks (to the client) like a callable server procedure
- Client program thinks it is calling the server

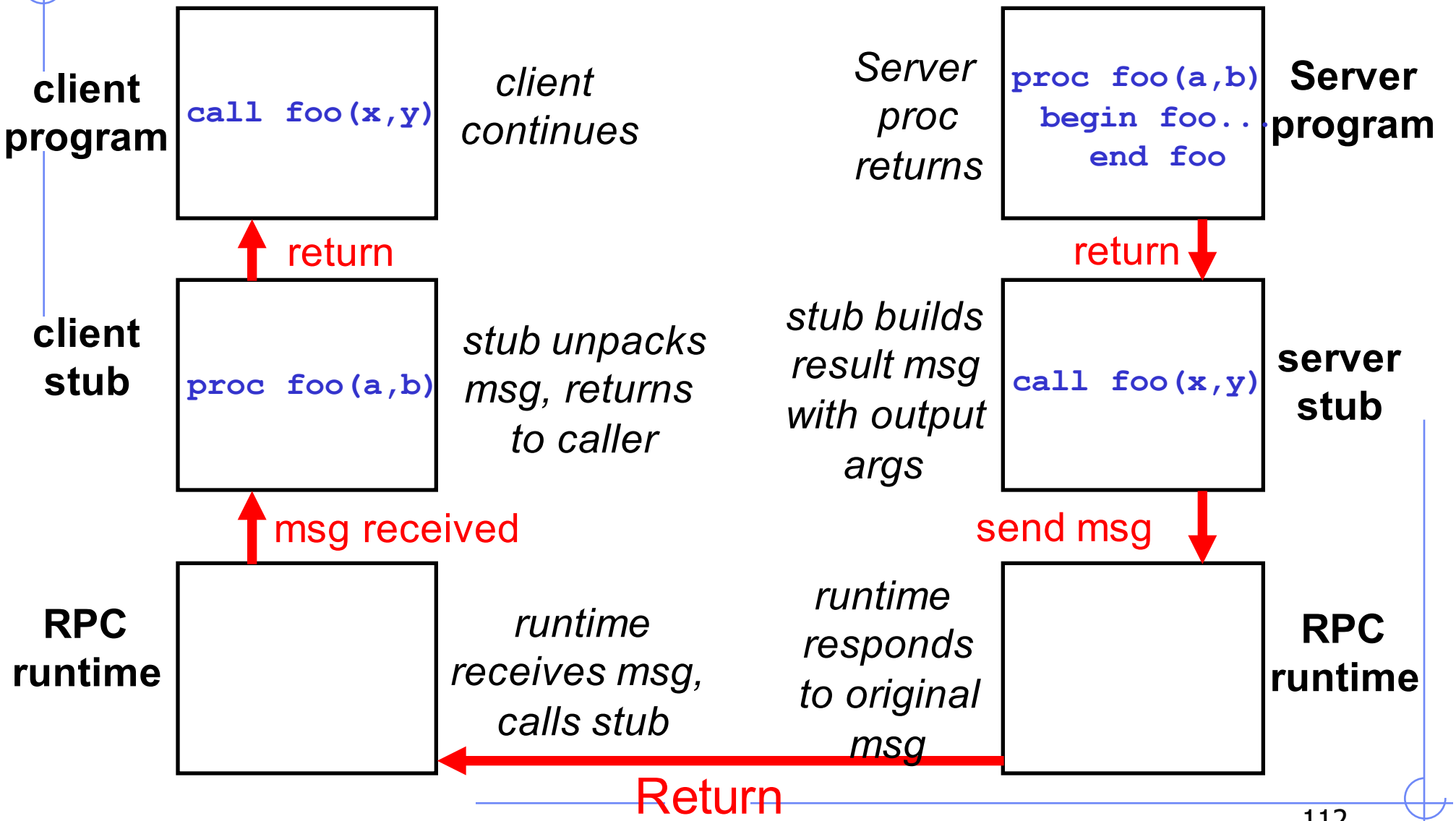
Server-side stub:

- Server program thinks it is called by the client
- `foo` actually called by the server stub

RPC Call Structure



RPC Return Structure

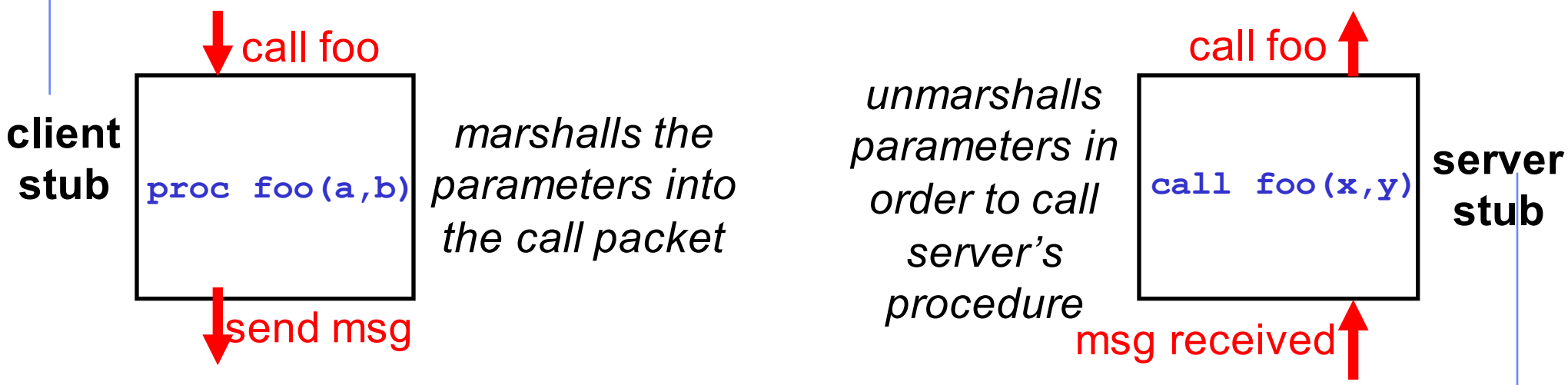


RPC Binding

- The process of connecting the client and server
- **Server:** on start up, *exports* its interface:
  - Identifies itself to a network name server
  - Tells local runtime its dispatcher address
- **Client:** before issuing any calls, *imports* the server:
  - Causes the RPC runtime to lookup the server through the name service and contact the requested server to setup a connection
- *Import* and *export* are explicit calls in the code

RPC Marshalling

- Packing of procedure parameters into a message packet
  - Also: pickling (python), serialization (Java)
- RPC stubs call type-specific procedures to marshall/unmarshall call parameters



On return: server stub marshalls return parameters into the return packet; client stub unmarshalls return parameters and returns to the client

RPC Concluding Remarks

RPC:

- A common model for communications in distributed applications
- *language support* for distributed programming
- relies on a *stub compiler* to automatically produce client/server stubs from the IDL server description
- commonly used, *even on a single node*, for communication between applications running in different address spaces.