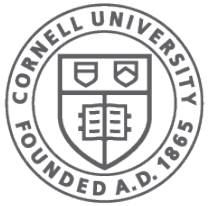
Networking

CS 4410
Operating Systems



Cornell CIS
COMPUTING AND INFORMATION SCIENCE

Outline

- Ethernet and Local Area Networking
- Internet Structure & Protocols
- TCP/IP
- Routing
- Remote Procedure Call

Application
Transport
Network
Link
Physical

Ethernet
and
Local Area Networking

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
- Can use different physical links
- Simple link-level protocol, scales well
- Simple algorithm for sharing the network well under load

Ethernet Goals

- Connect local area networks
  - Few buildings, short distances (<1 km)
- Inexpensively
  - Low infrastructure costs
- Without bottlenecks
  - No expensive routers, bridges, switches etc.
  - No state in the network, no store-and-forward

- Tremendously successful
- Simple conceptual model still in use
  - Despite two orders of magnitude increase in bandwidth

“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

Ethernet basics

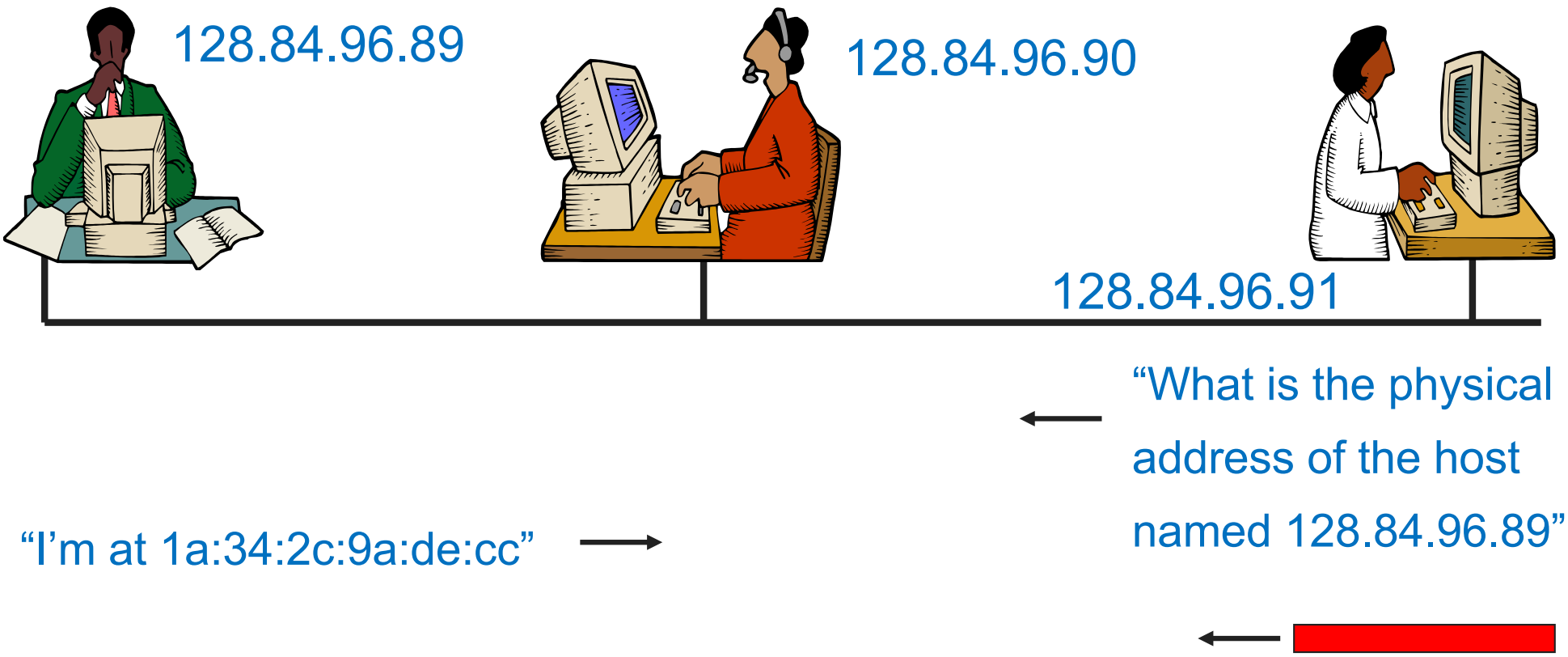
- An ethernet packet

Destination Address
Source Address
Type
…Data…
Checksum

Sending packets

Carrier sense, broadcast if ether is available

Addressing & ARP



ARP is used to discover physical addresses

ARP = Address Resolution Protocol

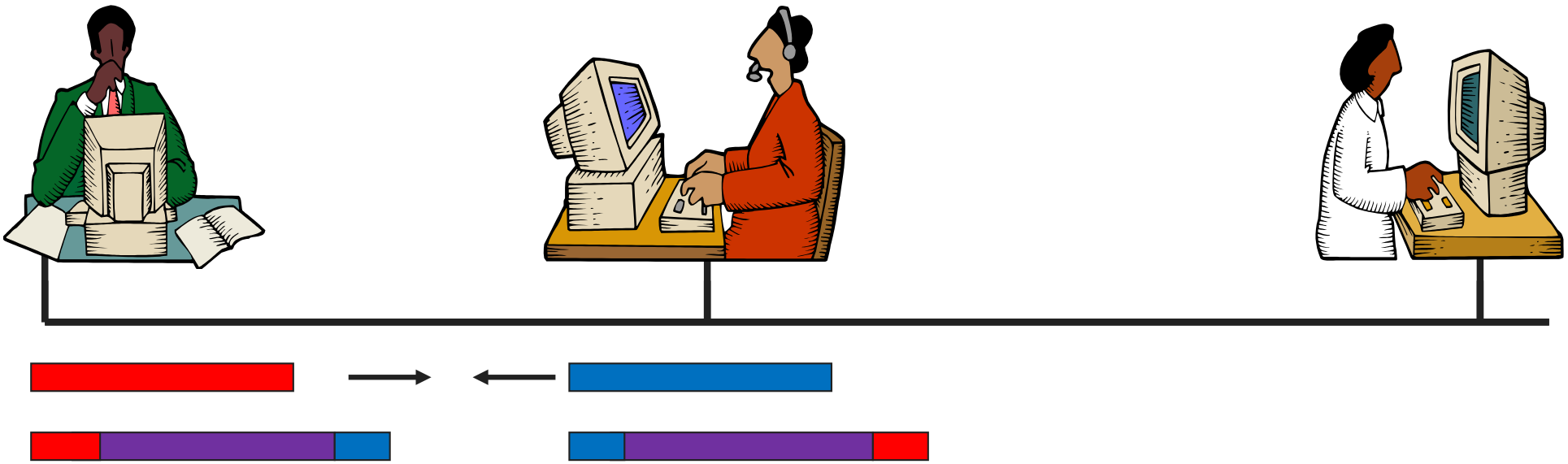
Addressing & DHCP

"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 network addresses

Collisions

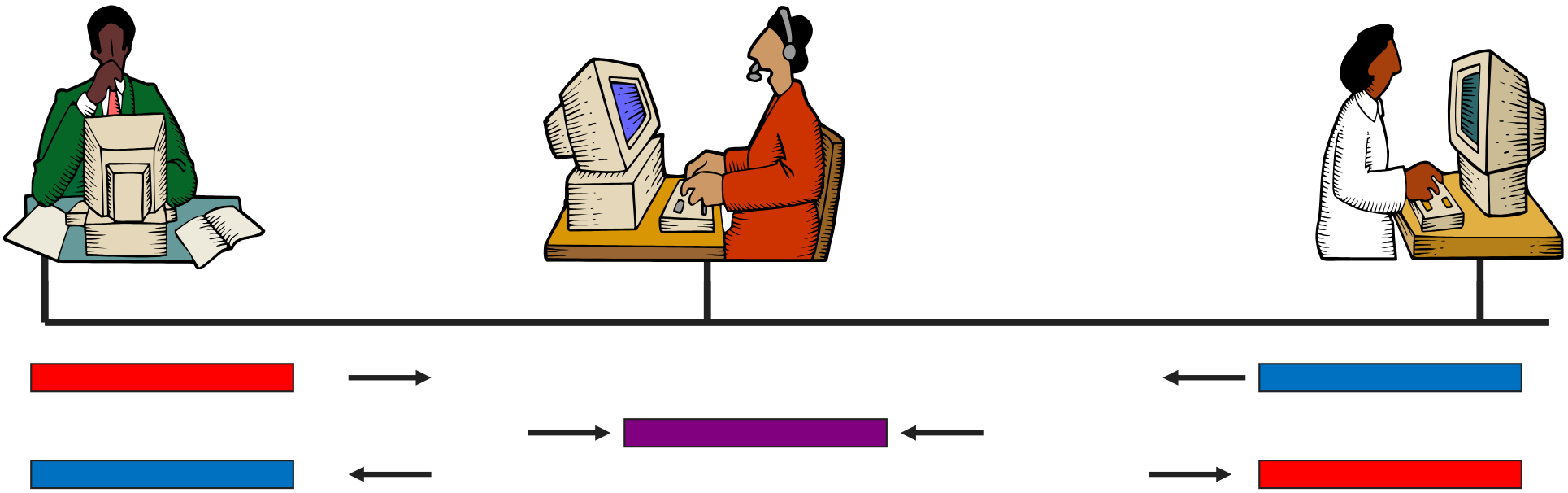


What happens if two people decide to transmit simultaneously ?

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
- Advantages:
  - Packet can be retransmitted at the link level immediately without high-level timeouts,
  - Packets are truncated early to avoid wasting bandwidth
  - Collision rates can be used to gauge net usage

Collisions



What happens if the packets are really short ?

Odds & Ends

- Minimum packet size is 64 bytes, which is just right for the maximum length of an Ethernet wire for all hosts to detect a collision
- Truncated packets are filtered out of the network
- CRC is 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
  - Can place ethernet card in promiscuous mode and listen

Gigabit Ethernet

- Today's Ethernet deployments are much faster
- In wired settings, Switched Ethernet has become the norm
  - All hosts connect to a switch
  - More secure, no possibility of snooping
  - Switches are a single failure point (but they rarely fail)
- In wireless settings, 802.11 and other protocols inherit many of the Ethernet concepts

Ethernet Lessons

- Best-effort delivery simplifies network design
- A simple, distributed protocol can tolerate failures and be easy to administer
- Networking infrastructure represents a large sunk cost
  - Best to keep it simple
  - Interoperable
  - Hard to upgrade means change occurs infrequently, when the gains are sizeable

Application
Transport
Network
Link
Physical

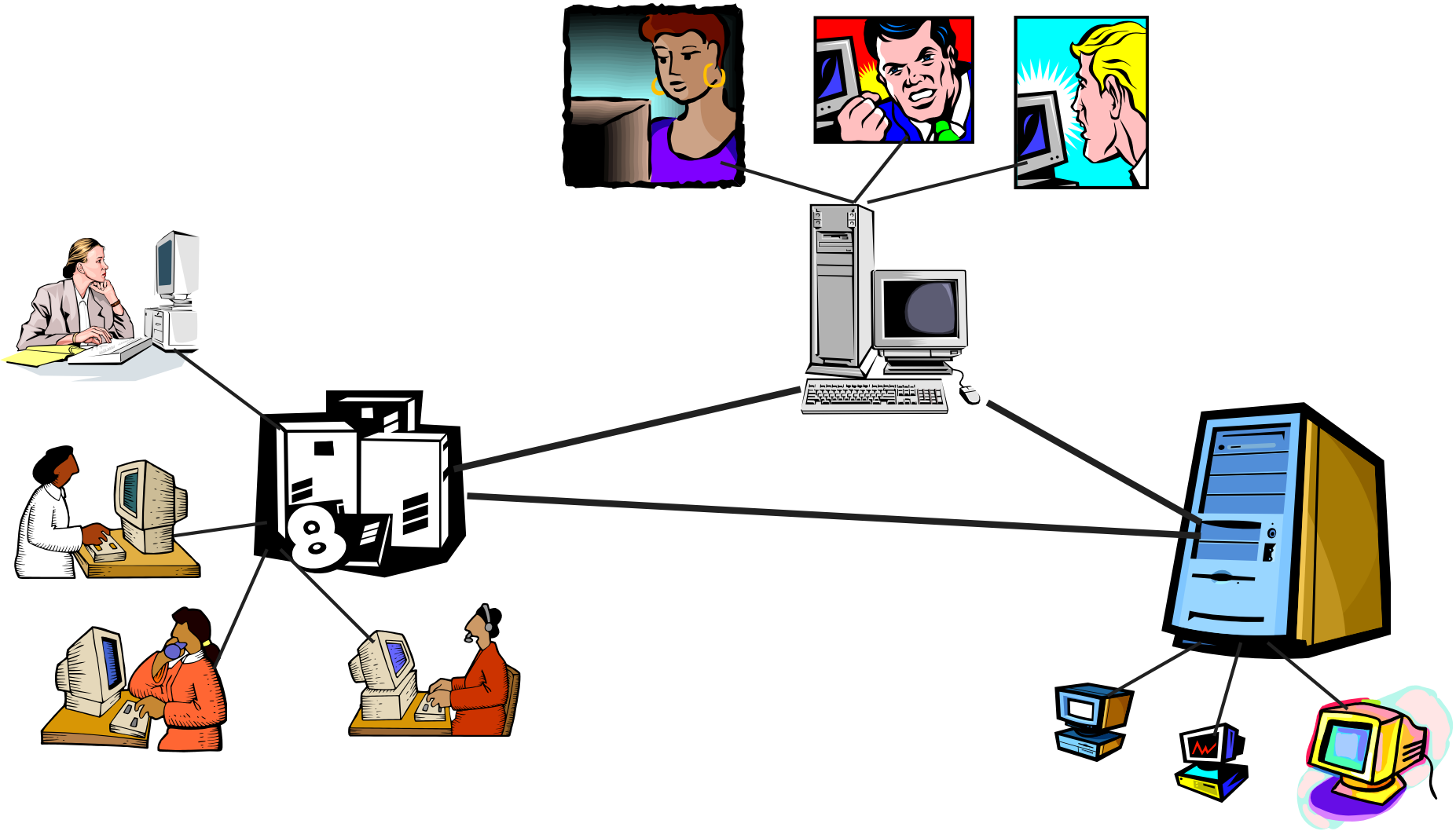
Internet Structure & Protocols

Internetworking Origins

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

- Point-to-point connections might have sufficed

Point to point connections



Internetworking Origins

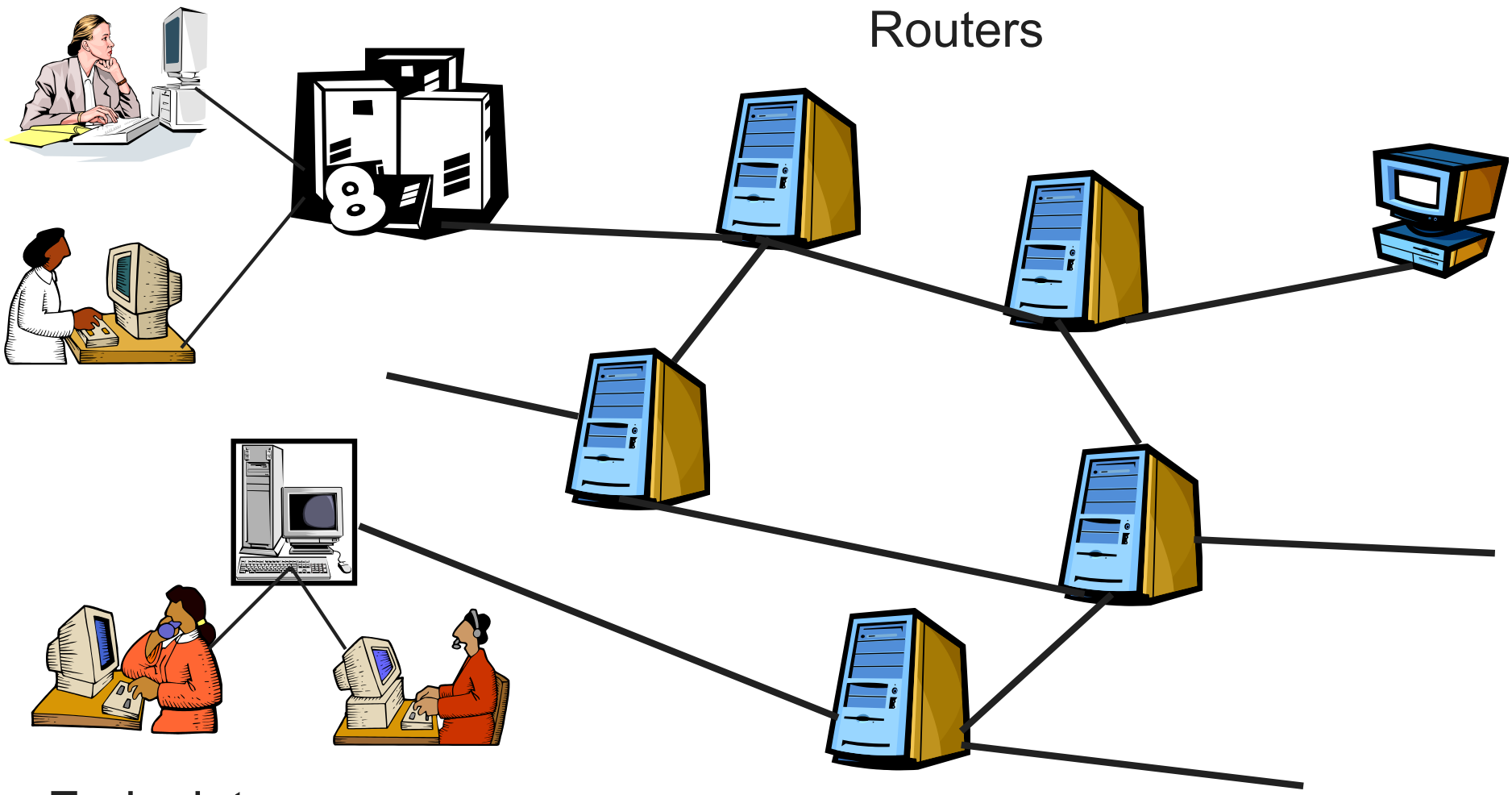
- Department of Defense initiated studies on how to build a resilient global network
  - How do you coordinate a nuclear attack ?
  - Especially, how do you tell people to stop firing missiles during a nuclear war ?
- Interoperability and dynamic routing are a must
  - Along with a lot of other properties
- Result: Internet
- A complex system with simple components

Internet Overview

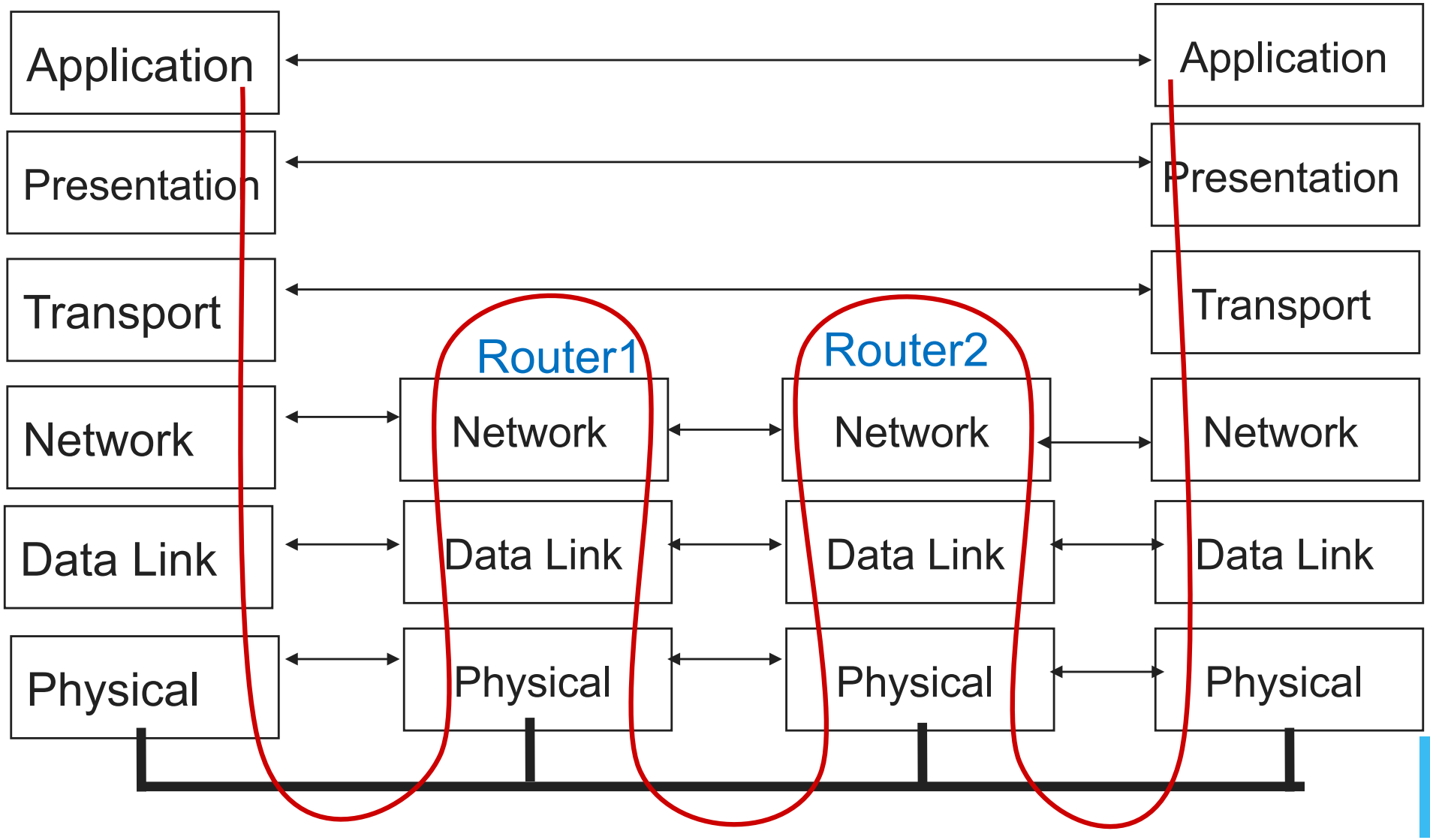
- Every host is assigned, and identified by, an IP address
- Each packet contains a header that specifies the destination address
- The network routes the packets from the source to the destination

- Question: What kinds of properties should the network provide?

Internet, The Big Picture



The Big Picture



The OSI Layers

- **Physical:** lowest layer, transmits and receives bits on the media (ex: electrical vs optical)
- **Data Link:** physical addressing, media access (ex: Ethernet)
- **Network:** Path determination across multiple network segments, routing, logical addressing (ex: IP)
- **Transport:** data transfer, reliability, packetization, retransmission, etc. (ex: TCP/UDP)
- **Session:** connection management (ex: TCP)
- **Presentation:** translation between network and application formats (ex: RPC packages, sockets)
- **Application:** implements application logic

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
  - Still need to check that file was written to 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

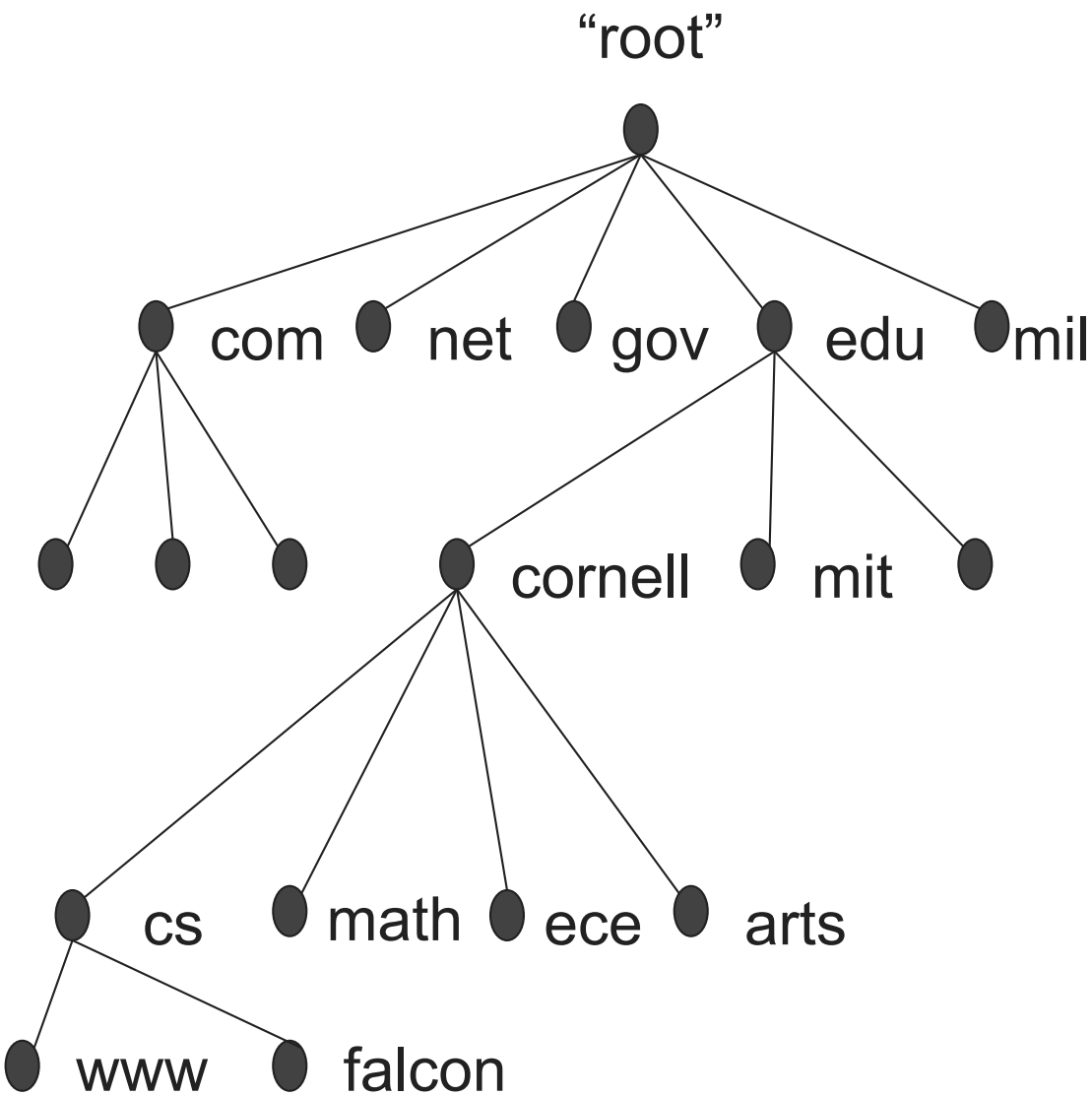
Naming

- Every host on the Internet is identified by an IP address
  - For now, 32-bit descriptor, like a phone number
  - Plans underway to change the underlying protocols to use longer addresses
- IP addresses are assigned to hosts by their internet service providers
  - Not physical addresses: IP address does not identify a single node, can swap machines and reuse the same IP address
  - Not entirely virtual: the IP address determines how packets get to you, and changes when you change your ISP
- Need completely virtual names
  - No one wants to remember a bunch of numbers

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
- 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 nameserver
2. the local nameserver asks one of the *root nameservers*
3. the root nameserver replies with the address of the authoritative nameserver
4. the server then queries that nameserver
5. repeat until host is reached, cache result.

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
- No security!
- Namespace = money
  - Innovations in this space are met with resistance from people who control name resolution

Application
Transport
Network
Link
Physical

IP

IP

- Internetworking protocol
  - Network layer
- Common packet format for the Internet
  - Specifies what packets look like
  - *Fragments* long packets into shorter packets
  - *Reassembles* fragments into original shape
- Some parts are fundamental, and some are arbitrary
  - IPv4 is what most people use
  - IPv6 clears up some of the messy parts, but is not yet in wide use

IPv4 packet layout

<table>
<tr><td>Version</td><td>IHL</td><td>TOS</td><td colspan="2">Total Length</td></tr>
<tr><td colspan="3">Identification</td><td>Flags</td><td>Fragment Offset</td></tr>
<tr><td colspan="2">TTL</td><td>Protocol</td><td colspan="2">Header Checksum</td></tr>
<tr><td colspan="5">Source Address</td></tr>
<tr><td colspan="5">Destination Address</td></tr>
<tr><td colspan="5">Options</td></tr>
<tr><td colspan="5">Data</td></tr>
</table>

IPv4 packet layout

<table>
<tr><td>Version</td><td>IHL</td><td>TOS</td><td colspan="2">Total Length</td></tr>
<tr><td colspan="3">Identification</td><td>Flags</td><td>Fragment Offset</td></tr>
<tr><td colspan="2">TTL</td><td>Protocol</td><td colspan="2">Header Checksum</td></tr>
<tr><td colspan="5">Source Address</td></tr>
<tr><td colspan="5">Destination Address</td></tr>
<tr><td colspan="5">Options</td></tr>
<tr><td colspan="5">Data</td></tr>
</table>

IP Fragmentation

- Networks have different maximum packet sizes
  - Big packets are sometimes desirable – less overhead
  - Huge packets are not desirable – reduced response time for others
- Higher level protocols (e.g. TCP or UDP) could figure out the max transfer unit and chop data into smaller packets
  - The endpoints do not necessarily know what the MTU is on the path
  - The route can change underneath
- Consequently, IP transparently fragments and reassembles packets

IP Fragmentation Mechanics

- IP divides a long datagram into N smaller datagrams
- Copies the header
- Assigns a Fragment ID to each part
- Sets the More Fragments bit
- Receiving end puts the fragments together based on the new IP headers
- Throws out fragments after a certain amount of time if they have not be reassembled

IP Options

- 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

Application
Transport
Network
Link
Physical

UDP & TCP

UDP

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

UDP Packet Layout

IP	Version	IHL	TOS	Total Length	
	Identification			Flags	Fragment Offset
	TTL		Protocol	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
- Applications need to deal with reordering, duplicate suppression, reliable delivery
  - Some apps can ignore these effects and still function
- Unreliable datagrams are the bare-bones network service
  - Good to build on, esp for multimedia applications

TCP

- Transmission Control Protocol
  - Reliable, ordered communication
- Enough applications demand reliable ordered delivery that they should not have to implement their own protocol
- A standard, adaptive protocol that delivers good-enough performance and deals well with congestion
- All web traffic travels over TCP/IP

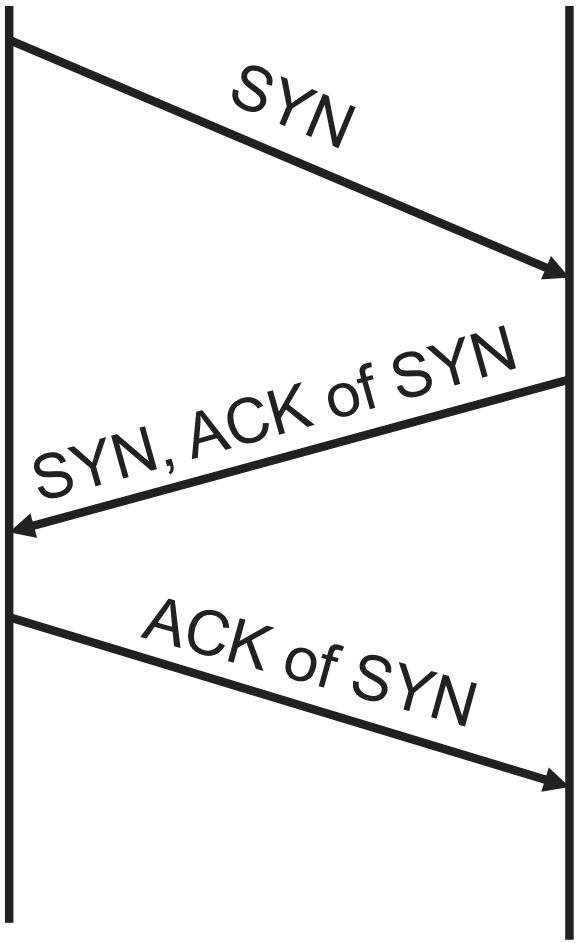
TCP/IP Packets

IP	Version	IHL	TOS	Total Length	
	Identification			Flags	Fragment Offset
	TTL		Protocol	Header Checksum	
	Source Address				
	Destination Address				
TCP	Source Port			Destination Port	
	Sequence Number				
	Acknowledgement Number				
	Offset	ACK|URG|SYN|FIN|RST			Window
	Checksum			Urgent Pointer	
	Options			Padding	

TCP Packets

- Each packet carries a unique ID
  - The initial number is chosen randomly
  - The ID is incremented by the data length
- Each packet carries an acknowledgement
  - Can acknowledge a set of packets by ack'ing the latest one received
- Reliable transport is implemented using these identifiers

TCP Connections

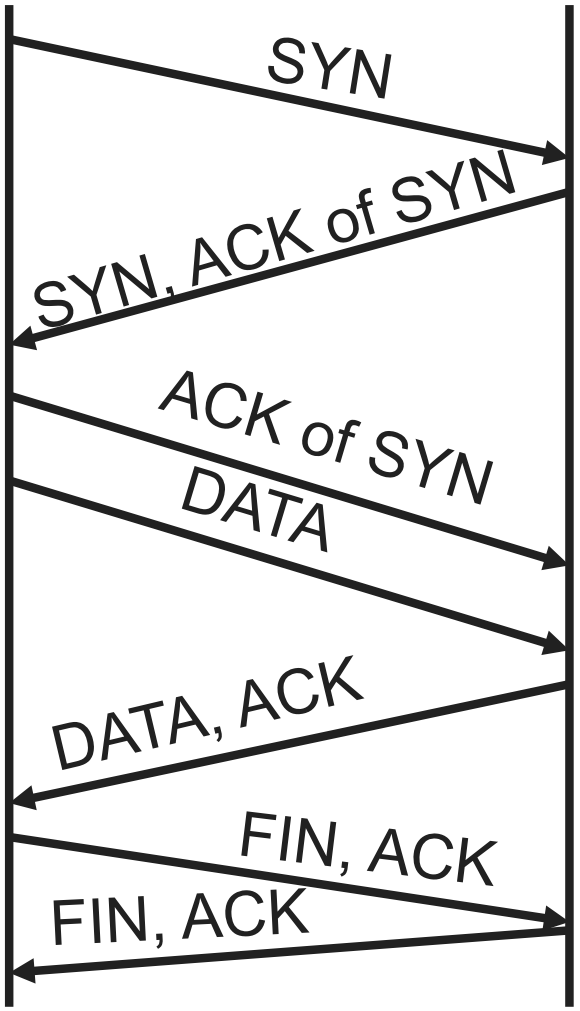


- TCP is *connection* oriented
- A connection is initiated with a three-way handshake
- Three-way handshake ensures against duplicate SYN packets
- Takes 3 packets, 1.5 RTT

TCP Handshakes

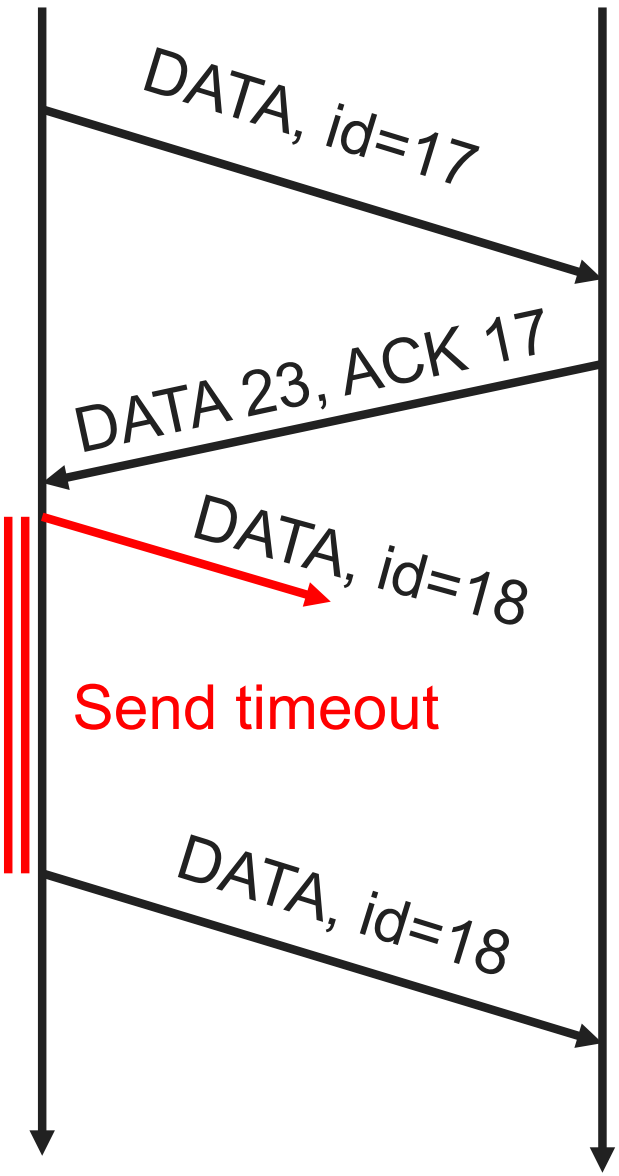
- 3-way handshake establishes common state on both sides of a connection. Both sides will:
  - know that the other side is ready to receive
  - have seen one packet from the other side → know what the first seqno ought to be

Typical TCP Usage



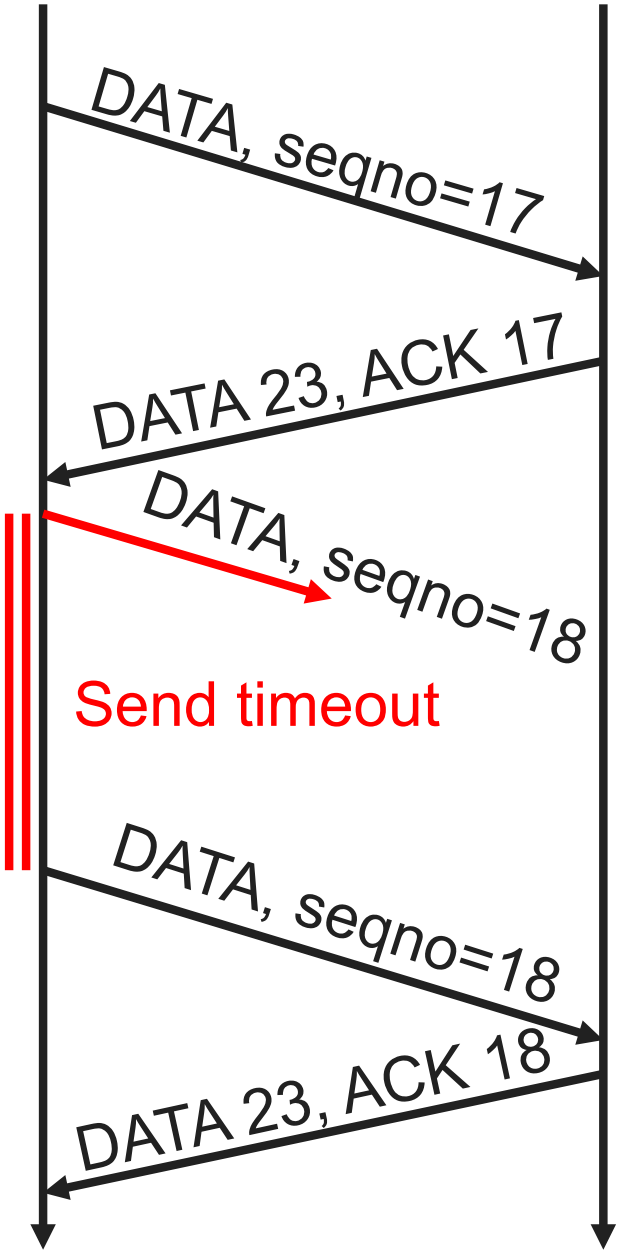
- Three round-trips to set up a connection, send a data packet, receive a response, tear down connection
- FINs work (mostly) like SYNs to tear down connection
  - Need to wait after a FIN for straggling packets

Reliable transport



- TCP keeps a copy of all sent, but unacknowledged packets
- If acknowledgement does not arrive within a “send timeout” period, packet is resent
- Send timeout adjusts to the round-trip delay

Reliable transport



- **Sequence number** corresponds to the number of bytes sent so far
- Each host keeps track of how many bytes it has sent and received
- A packet carrying solely an ACK has the same seqno as a previous packet
- Thus, ACKs do not require ACKs

TCP timeouts

- What is a good timeout period ?
  - Want to improve throughput without unnecessary transmissions

$$\text{NewAverageRTT} = (1 - \alpha)\ \text{OldAverageRTT} + \alpha\ \text{LatestRTT}$$

$$\text{NewAverageDev} = (1 - \alpha)\ \text{OldAverageDev} + \alpha\ \text{LatestDev}$$

where $\text{LatestRTT} = (\text{ack\_receive\_time} - \text{send\_time})$,

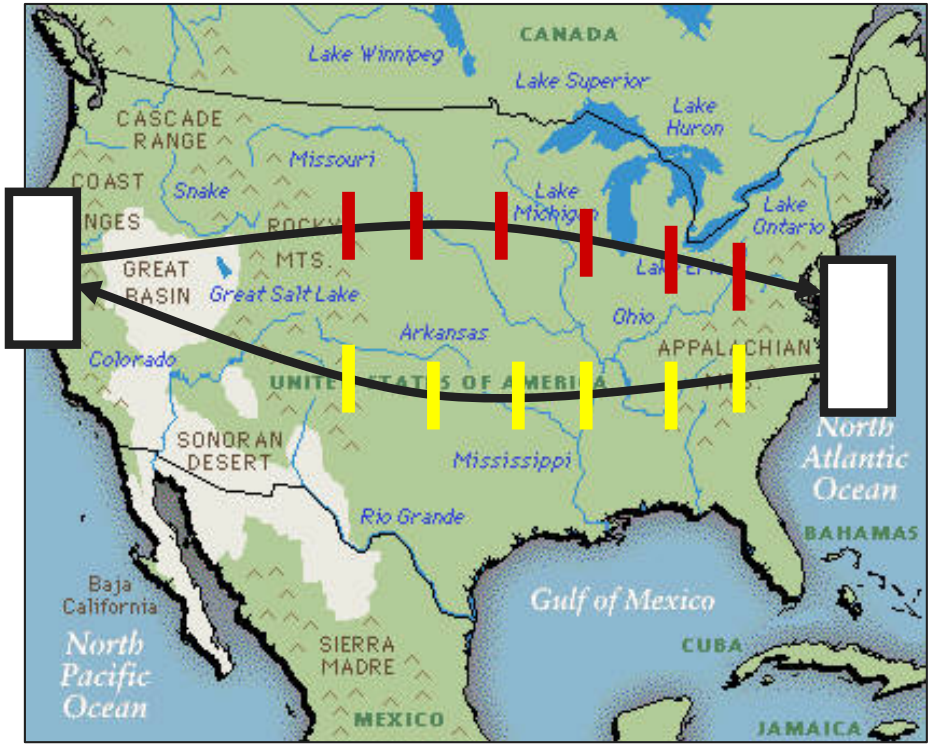
$\text{LatestDev} = |\text{LatestRTT} - \text{AverageRTT}|$,

$\alpha = 1/8$, typically.

$$\text{Timeout} = \text{AverageRTT} + 4 * \text{AverageDev}$$

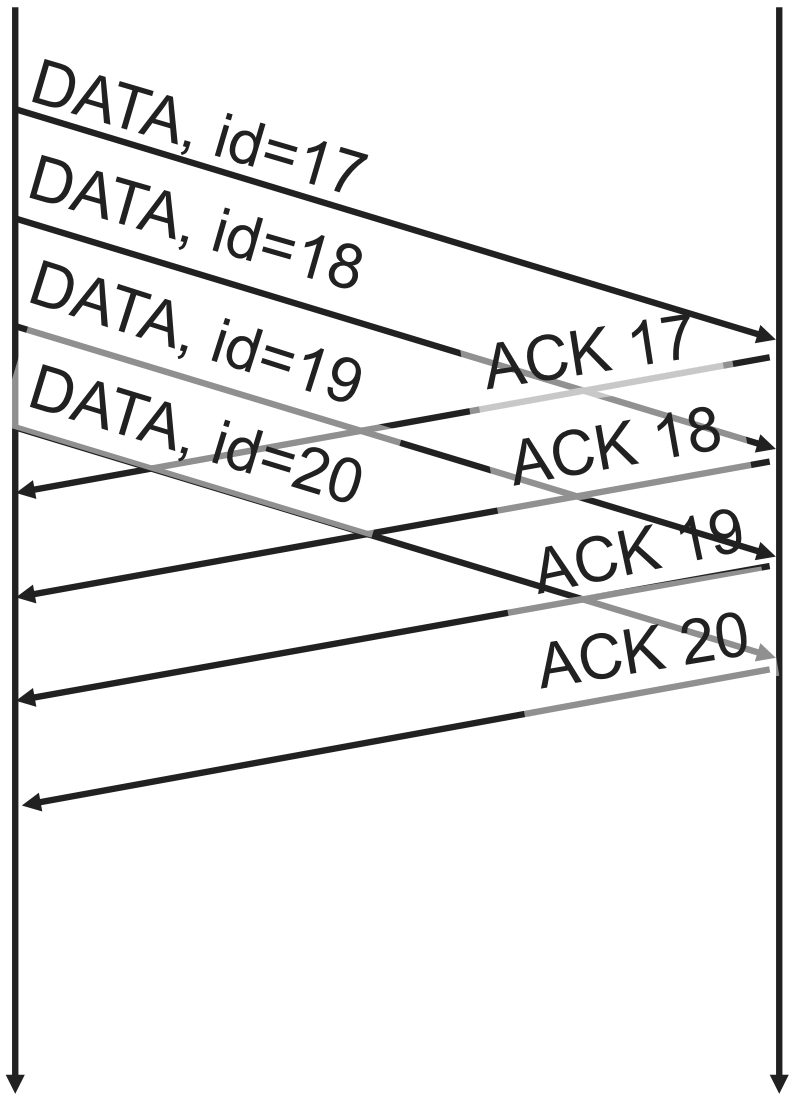
- Timeout is thus a function of RTT and deviation

TCP Windows



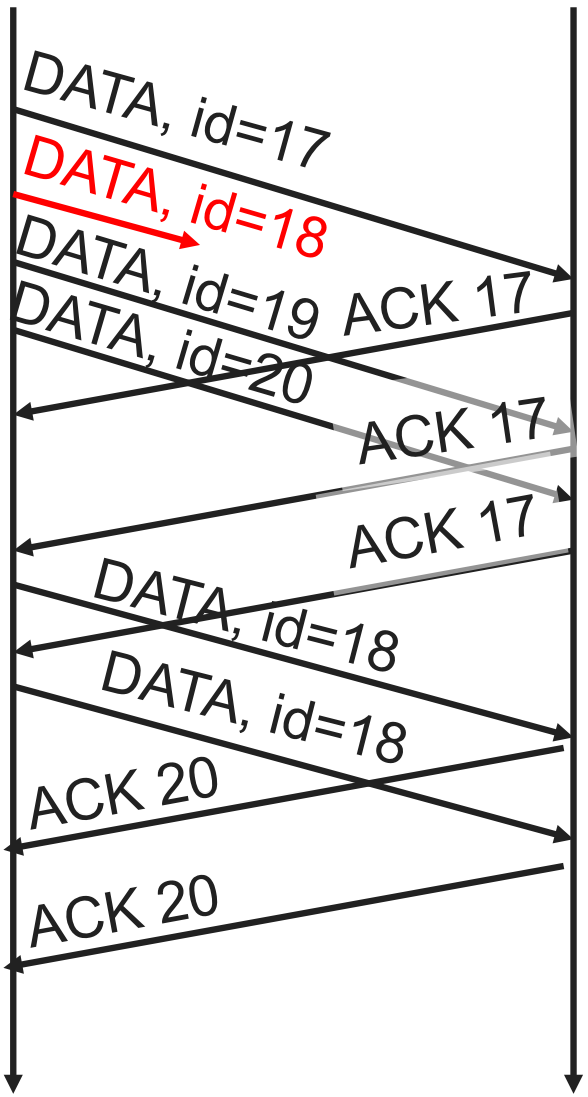
Multiple outstanding packets can increase throughput

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



- Receiver detects a lost packet (*i.e.*, a missing seqno), acks the last seqno it successfully received
- Sender detects the loss without waiting for timeout

TCP Congestion Control

TCP:

- increases window size as long as no packets are dropped
- halves the window size when a packet drop occurs
  - Packet drop evident from the acknowledgements

→ slowly build up to max bandwidth, and hover there

- Does not achieve the max possible

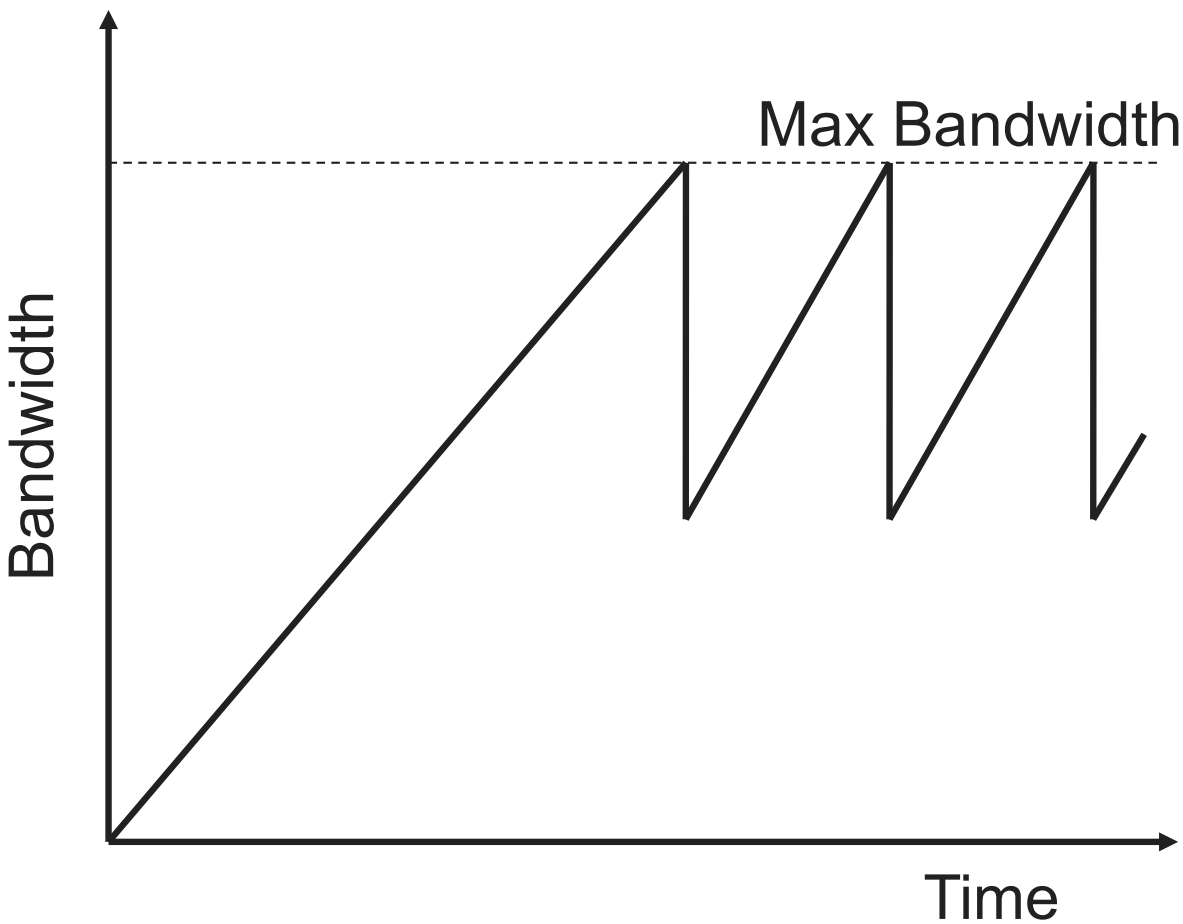
+ Shares bandwidth well with other TCP connections

- This linear-increase, exponential backoff in the face of congestion is termed ***TCP-friendliness***

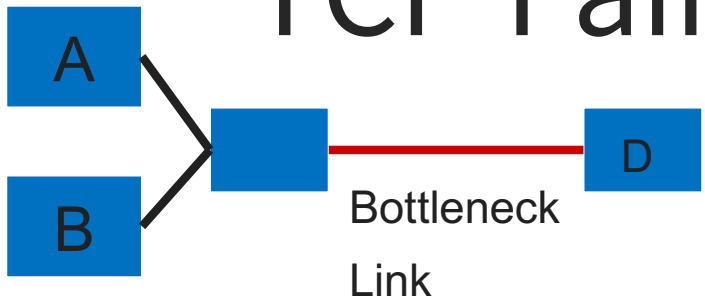
TCP Window Size

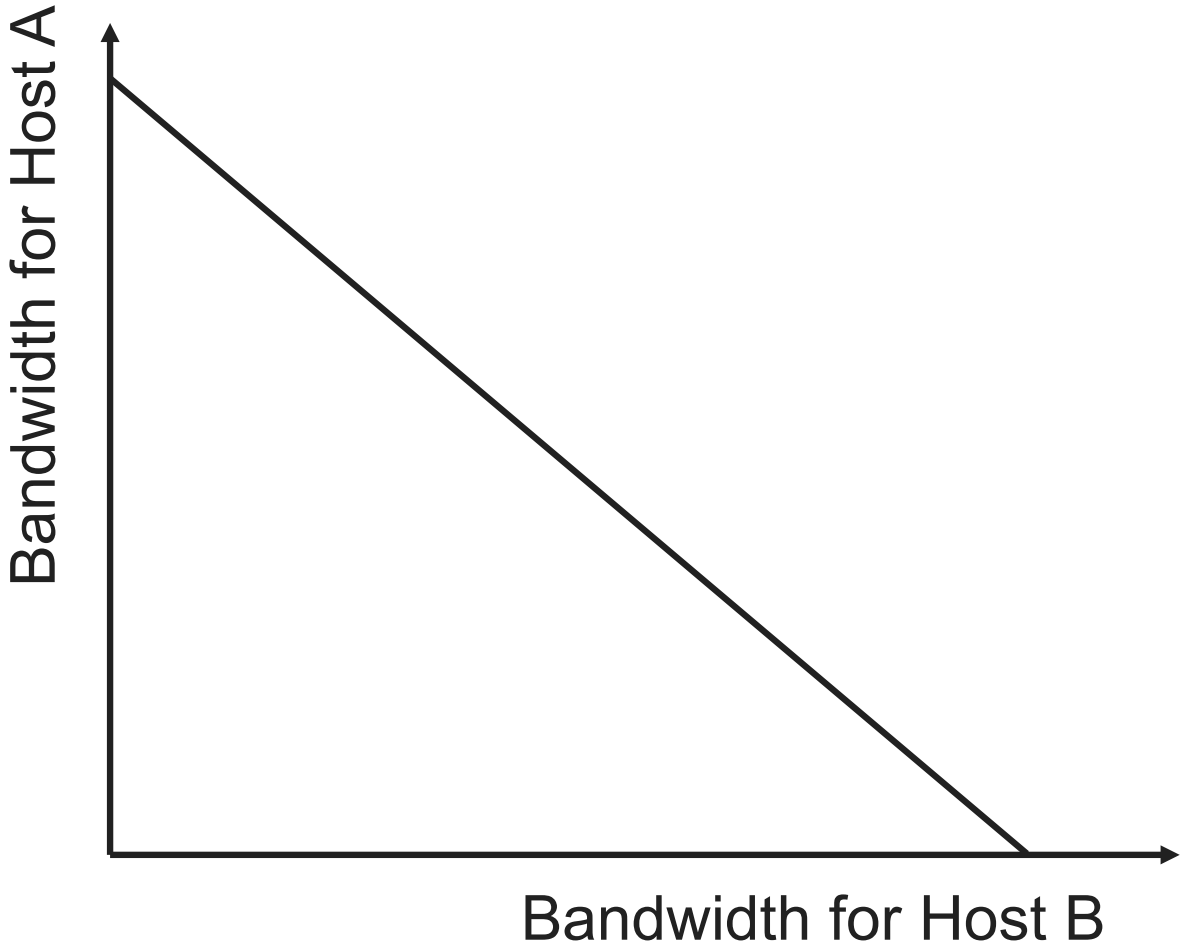
- Linear increase
- Exponential backoff

(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

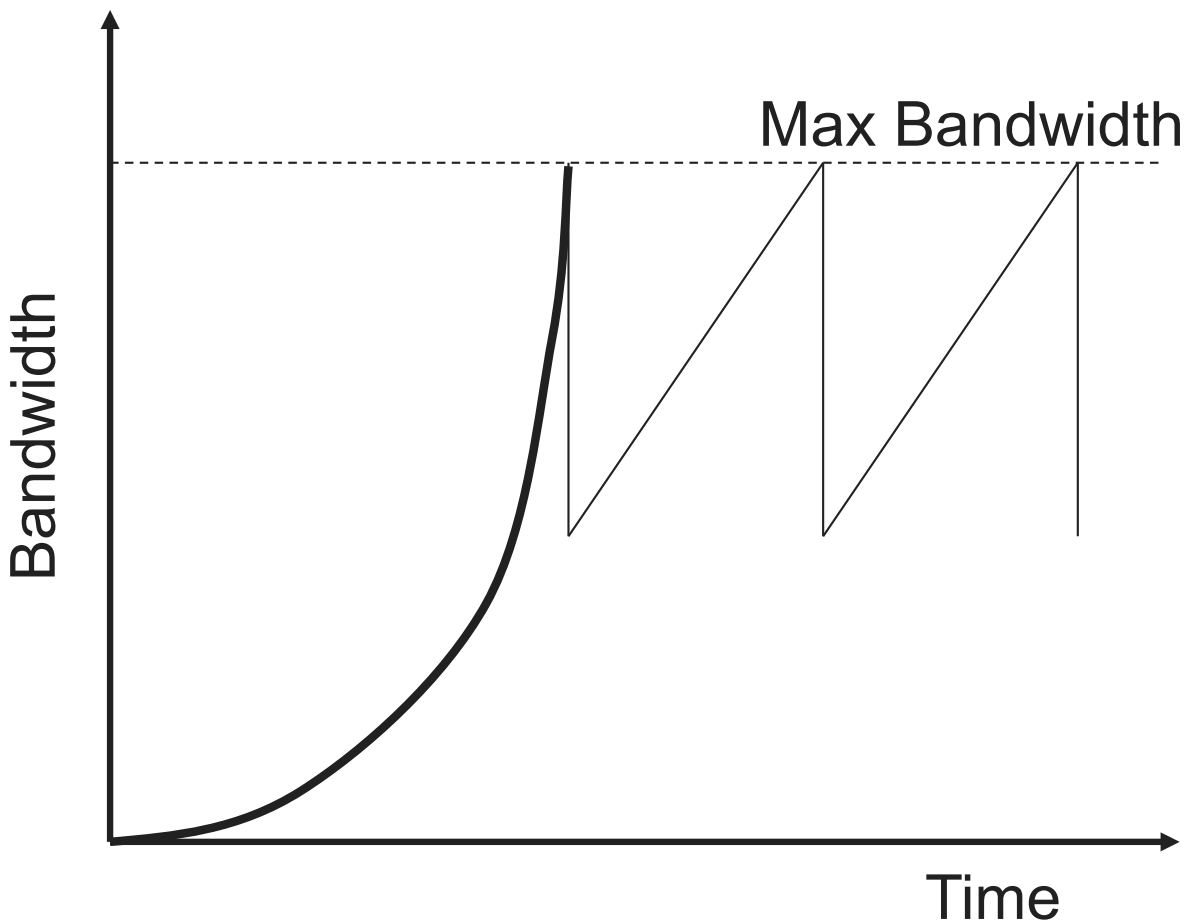
Problem: Linear increase takes a long time to build up a window size that matches the link bandwidth*delay

- Most file transactions are not long enough

→ TCP can spend a lot of time with small windows, never reaching a sufficiently large window size

Fix: Allow TCP to build up to a large window size initially by 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

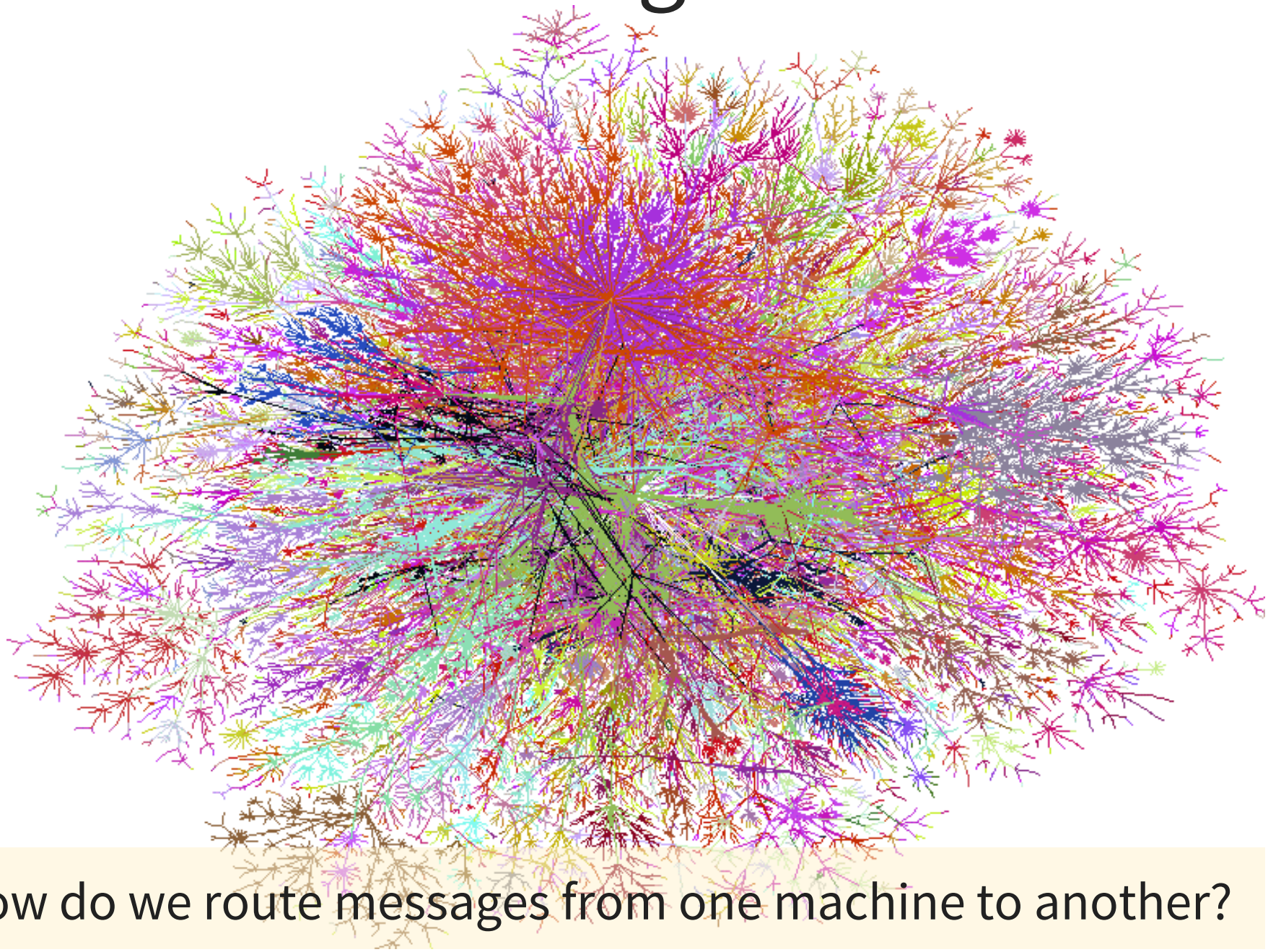
- Reliable ordered message delivery
  - Connection oriented, 3-way handshake
- Transmission window for better throughput
  - Timeouts based on link parameters
- Congestion control
  - Linear increase, exponential backoff
- Fast adaptation
  - Exponential increase in the initial phase

Application
Transport
Network
Link
Physical

Routing

Several figures in this section come from
"Computer Networking: A Top Down Approach"
by Jim Kurose, Keith Ross

The Internet is Big….



How do we route messages from one machine to another?

Routing Challenge

Discover and maintain paths through the network between communicating endpoints.

- Metrics of importance
  - Latency
  - Bandwidth
  - Packet Overhead (“Goodput”)
  - Jitter (packet delay variation)
  - Memory space per node
  - Computational overhead per node

Domains

- Wired networks
  - Stable, administered, lots of infrastructure
    - *e.g.*, the Internet
- Wireless networks
  - Wireless, dynamic, self-organizing
  - Infrastructure-based wireless networks
    - A.k.a. cell-based, access-point-based
    - *e.g.*, Cornell's "rover"
  - Infrastructure-less wireless networks
    - A.k.a. ad hoc

Algorithm Classifications

Route discovery, selection and usage

- Reactive vs. **Proactive**
- **Single path** vs. Multipath
- Centralized vs. **Distributed**

Reactive Routing

- Routes discovered on the fly, as needed
  - Discovery often involves network-wide query
  - Used on many wireless ad hoc networks

- Examples
  - Dynamic source routing (DSR)
  - Ad hoc on-demand distance vector (AODV)

Dynamic Source Routing (DSR) Protocol

Route Discovery:

(1) Source sends neighbors RouteRequest

"I'm Source X looking for Dest Y"

- Path to Y generated as neighbors add themselves to the path & pass RREQ to *their* neighbors
- Nodes drop redundant RREQs

(2) Destination sends back a RouteReply

"I'm Dest Y responding to Source X"

- Source X caches path to Y
- future data packets specify path in header

Route Maintenance:

- Broken links reported
- Affected paths removed from caches

Reactive Routing

- Pros
  - Routers require no state
  - State proportional to # of used routes
  - Communication proportional to # of used routes and failure rate
- Cons
  - Route discovery latency is high
  - Jitter (variance of packet interarrival times) is high

Algorithm Classifications

Route discovery, selection and usage

- Reactive vs. **Proactive**
- **Single path** vs. Multipath
- Centralized vs. **Distributed**

Proactive Routing

- Routes are disseminated from each node to all others, periodically
  - Every host has routes available to every other host, regardless of need
  - Used on the internet, some wireless ad hoc networks

Graph Abstraction of the Network

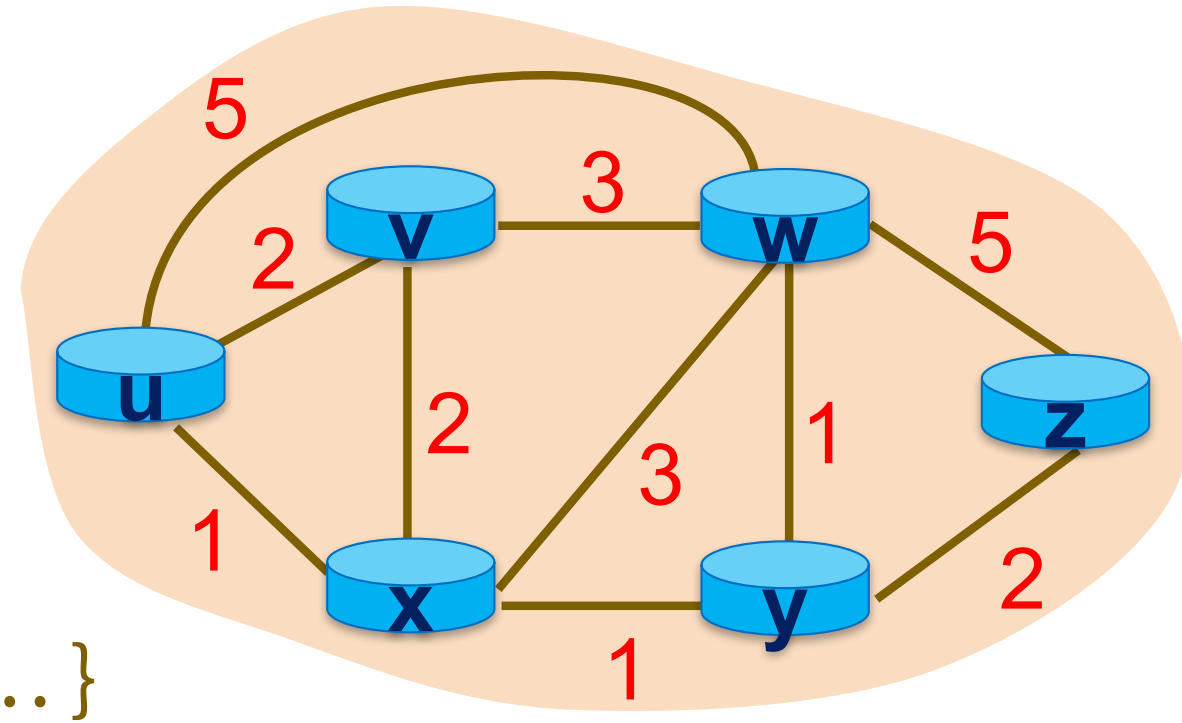
graph G = (V,E)

set of routers

V = { u, v, w, x, y, z }

set of links

E ={ (u,v), (u,x),(u,w)… }



cost of link c(x,x’) e.g., c(w,z) = 5

(cost could always be 1, or inversely related to b/w or congestion)

key question: what is the least-cost path between u and z ?

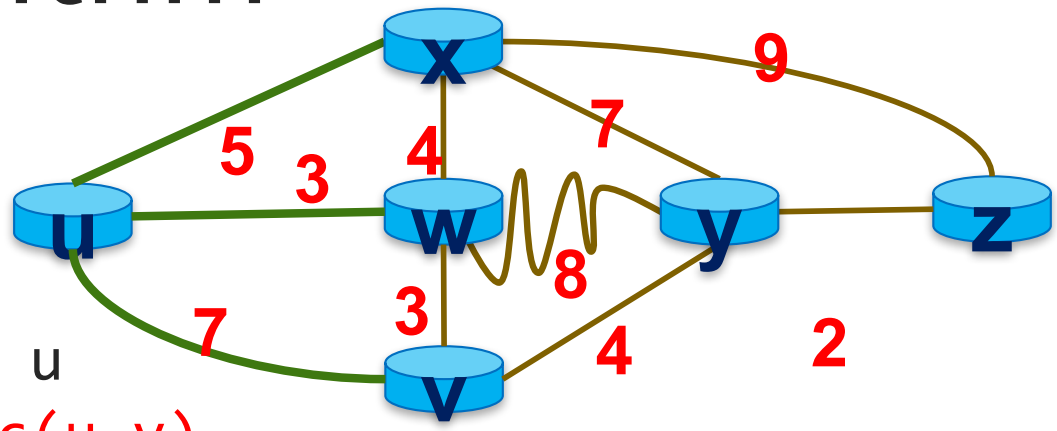
routing algorithm: algorithm that finds that least cost path

Link State (LS) Routing Algorithm

- iterative, centralized
- network topology, all link costs known up front
  - accomplished via “link state broadcast”
  - all nodes have same info
- based on Dijkstra’s (shortest path algorithm)
  - computes least cost paths from one node (‘source”) to all other nodes
- Example: Open Shortest Path First (OSPF) Protocol

```
c(x,y): link cost from node x to y;
        (∞ for non-neighbors)
D(v):   current cost of path from source to v
N':     set of nodes whose least cost path
        definitively known
```

Dijsktra's algorithm



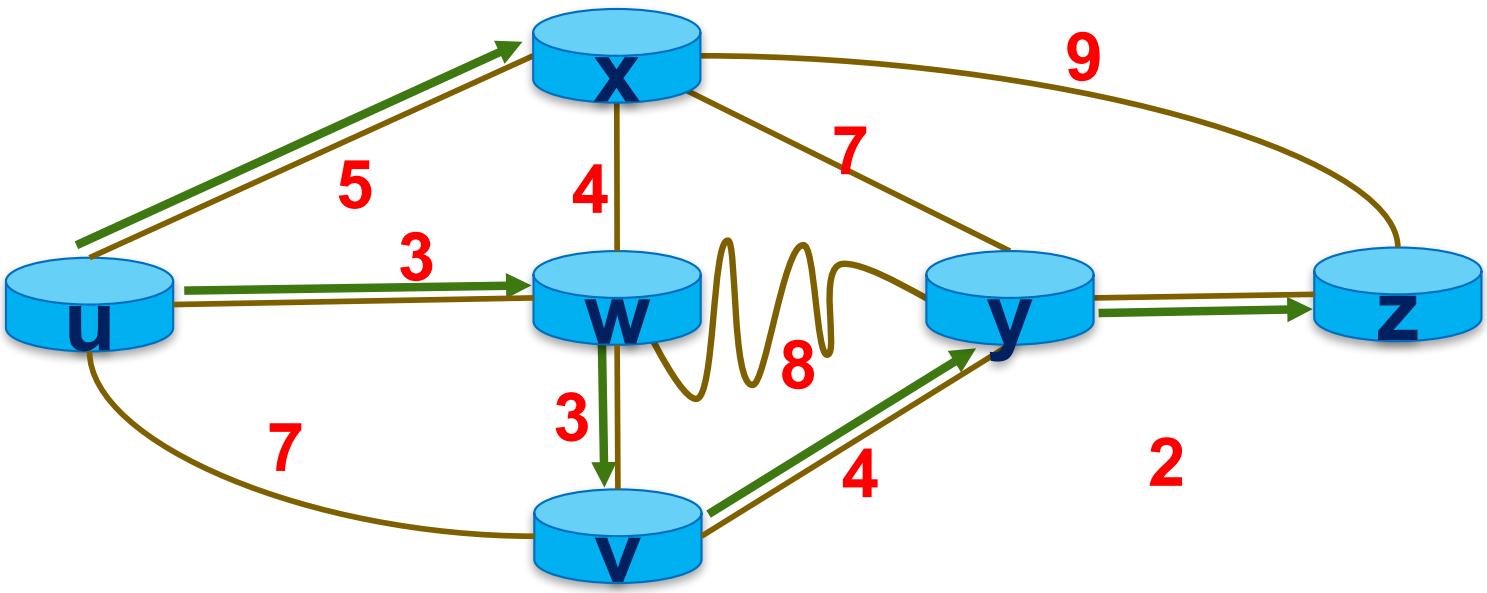

```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    else D(v) = ∞

 Loop
   find w not in N' such that D(w) is a minimum
   add w to N'
   update D(v) for all v adjacent to w & not in N' :
      D(v) = min( D(v), D(w) + c(w,v) )
   /* new cost to v either: old cost to v or known
    shortest path cost to w plus cost from w to v */
until all nodes in N'
```

Dijsktra's in Action

p(x): predecessor node along path from source to node x

Step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					



Algorithm Classifications

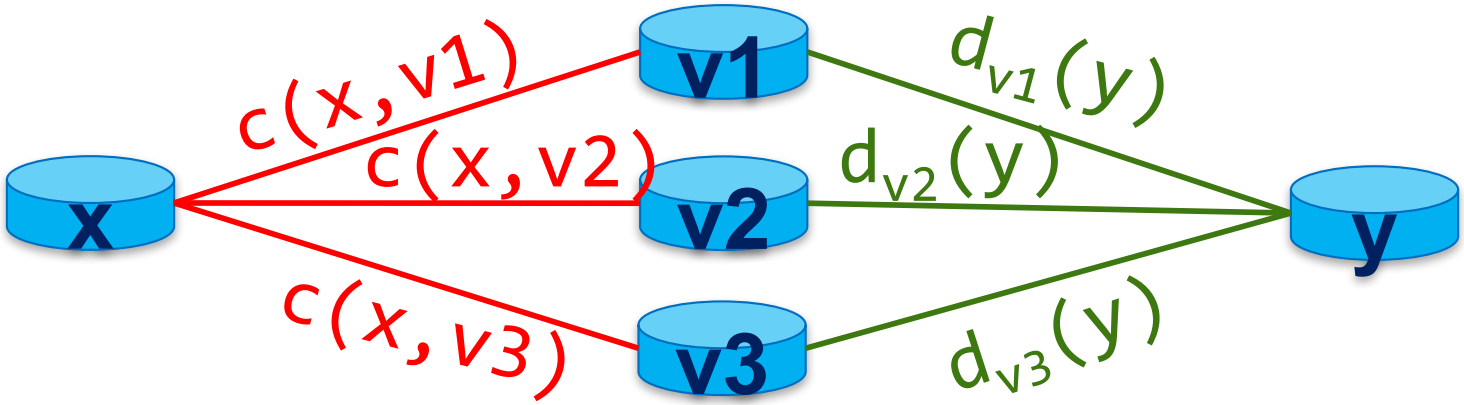
Route discovery, selection and usage

- Reactive vs. **Proactive**
- **Single path** vs. Multipath
- Centralized vs. **Distributed**

Distance Vector (DV) Routing Algorithm

- iterative, asynchronous, distributed
- based on Bellman-Ford (shortest path algorithm)
- Example: Routing Information Protocol (RIP)

```
let
   dx(y) := cost of least-cost path from x to y
then
   dx(y) = min {c(x,v) + dv(y) }
         for all neighbors v of x
```



Bellman Ford Example

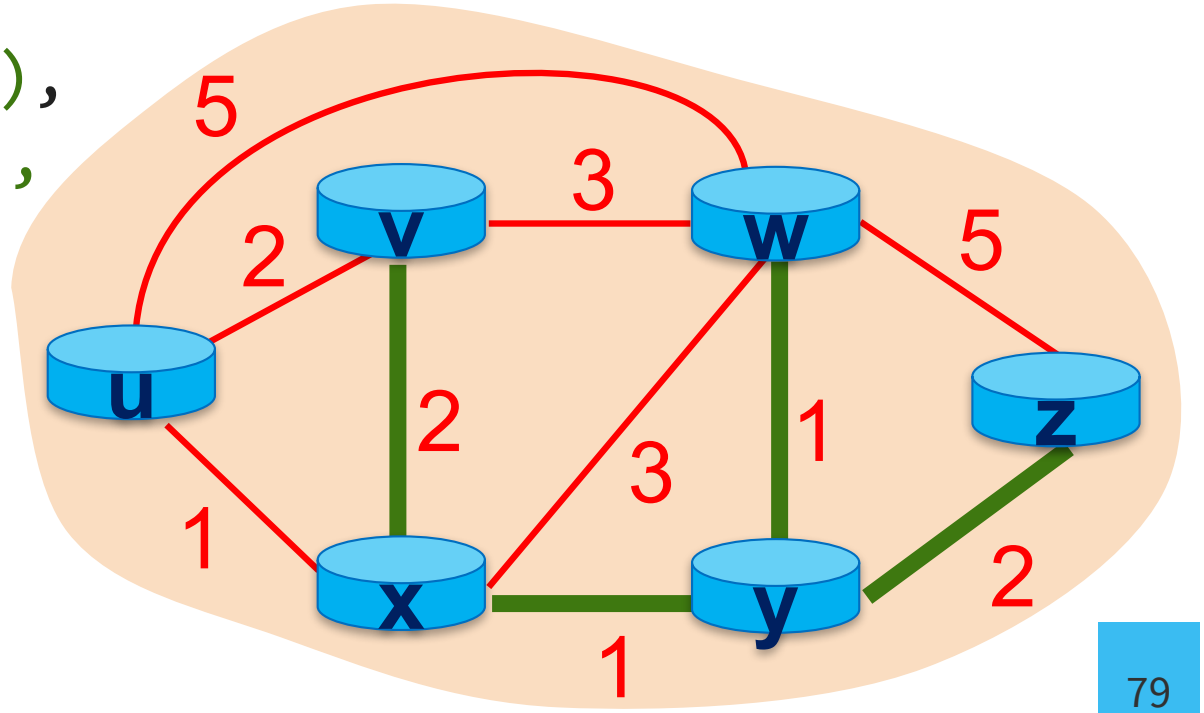
Shortest path from u to z?

Who are u's neighbors? {v, x, w}

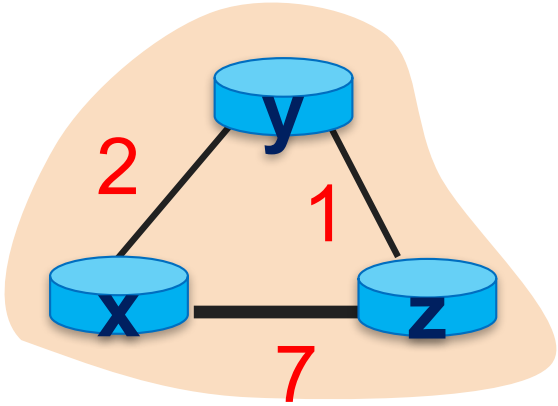
What are *their* shortest paths to z?

$d_v(z) = 5,\ d_x(z) = 3,\ d_w(z) = 3$

```
du(z)=min{c(u,v)+dv(z),
          c(u,x) + dx(z),
          c(u,w) + dw(z)
     = min {2 + 5,
            1 + 3,
            5 + 3}
     = 4
```



DV Algorithm



Each node x:

- knows cost to each neighbor v: c(x,v)
- maintains its neighbors' distance vectors

From time to time (esp. when a change occurs), 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.

DV Algorithm In Action

time

X, t=0		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
	z	∞	∞	∞

Y, t=0		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
	z	∞	∞	∞

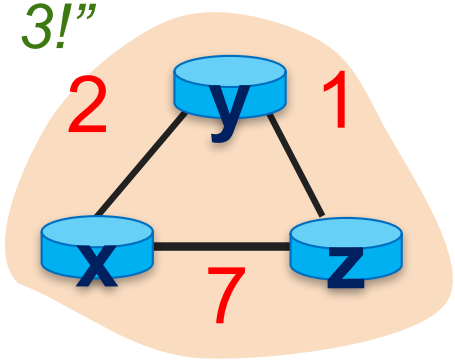
X, t=1		cost to		
		x	y	z
from	x	0	2	**3**
	y	**2**	**0**	**1**
	z	∞	∞	∞

Y sends X its DV

X updates its own DV

"If Y can get to Z in 1,

*then *I* can get to Z in 3!"*



DV Algorithm when costs decrease

time

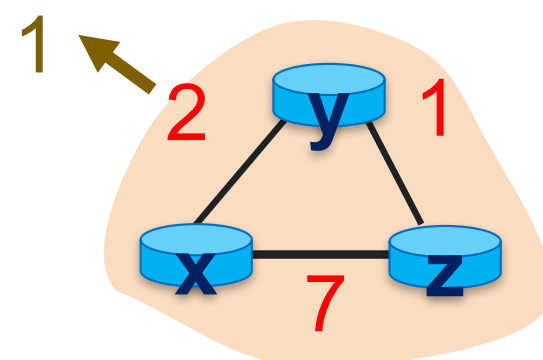
X, t=0		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

Y, t=0		cost to		
		x	y	z
from	x	0	2	3
	y	1 ~~2~~	0	1
	z	3	1	0

Y detects link-cost changes 2 → 1 Updates DV, broadcasts

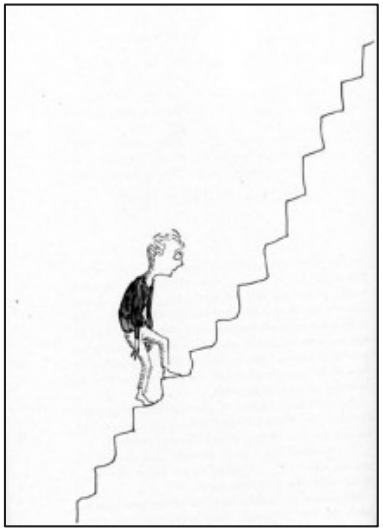
X, t=1		cost to		
		x	y	z
from	x	0	**1**	**2**
	y	**1**	**0**	**1**
	z	3	1	0

X updates its own DV, broadcasts



Counting to Infinity…

What if connections to z are lost?



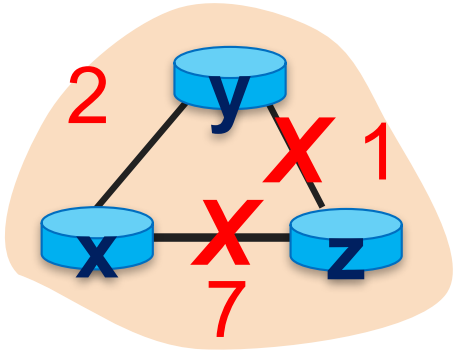
"Well, I can't reach Z anymore, but Y can do that in 1, so I can still get to Z in 3."

X, t=n		*cost to*		
		x	y	z
from	x	0	2	~~3~~ 3
	y	2	0	1
	z	∞	∞	∞

"Well, I can't reach Z anymore, but X can do that in 3, so I can still get to Z in 5."

Y, t=n		*cost to*		
		x	y	z
from	x	0	2	3
	y	2	0	~~1~~ 5
	z	∞	∞	∞

Next: Y sends X its new DV, X updates Y's DV, reruns BF, x → z increases from 3 → 7 … Next…!!



Path Vector (PV) Routing Algorithm

- Distance Vector with paths
- Example: Border Gateway Protocol (BGP)

  "glue that holds the Internet together"

High level:

- Each node x sends its distance vector with the actual path
- Nodes can filter out broken paths

Instead of just shortest path, **BGP** uses other considerations to select which route is best

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)
  - Security

Routing Gone Wrong

- Pakistan, 2008: "*I'll* take you to youtube!"
  - "How Pakistan knocked YouTube offline"
  - "Insecure routing redirects YouTube to Pakistan"

- China, 2010: "*I'll* take you to .gov and .mil"
  - "How China swallowed 15% of 'Net traffic for 18 minutes"
  - "China Hijacks 15% of Internet Traffic?"

Algorithm Classifications

Route discovery, selection and usage

- Reactive vs. **Proactive**
- **Single path** vs. Multipath
- Centralized vs. **Distributed**

Proactive Routing

- Pros
  - Route discovery latency is very low
- Cons
  - O(N) state in every router
  - Constant background communication

Hybrid Routing

- Proactive & Reactive routing have drawbacks
  - Work best under different network conditions
  - Many parameters to pick to get optimal performance

- Perform hybrid routing
- Some routes are disseminated proactively, others discovered reactively
  - Can outperform reactive *and* proactive across many scenarios
    SHARP [Mobihoc 2003]

Application
Presentation (ish)
Transport
Network
Link
Physical

Remote Procedure Call

Several figures in this section come from
"Distributed Systems: Principles and Paradigms"
by Andrew Tanenbaum & Maarten van Steen

Client/Server Paradigm

Common model for structuring distributed computation

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

Typical Pattern:

1. Client first *binds* to the server: locates it in the network & establishes a connection
2. Client sends *requests*: messages that indicate which service is desired, with parameters
3. Server returns *response*

Pros and Cons of Messages

+ Very flexible communication
  - Want a certain message format? *Go for it!*
– Problems with messages:
  - programmer must worry about message formats
  - must be packed and unpacked
  - server must decode to determined request
  - 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

Idea: Let clients call servers like they do procedures

Remote Procedure Call (RPC)

Goal: design RPC to look like a local PC

- A model for distributed communication
- Uses computer/language support
- 3 components on each side:
  - user program (client or server)
  - set of *stub* procedures
  - RPC runtime support

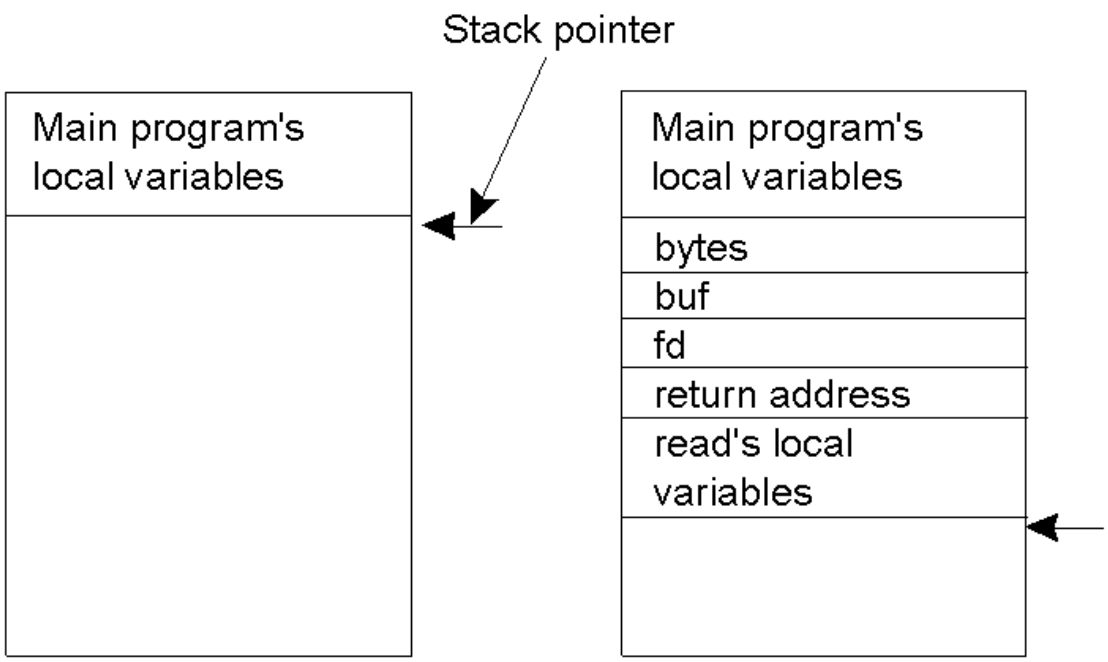
Birrell & Nelson @ Xerox PARC

"Implementing Remote Procedure Calls" (1984)

How does a function call work?

`read(int fd, char* buf, int nbytes)`

- `File descriptor`
- `character array`
- `how much to read`



Stack before procedure call

Stack during procedure call

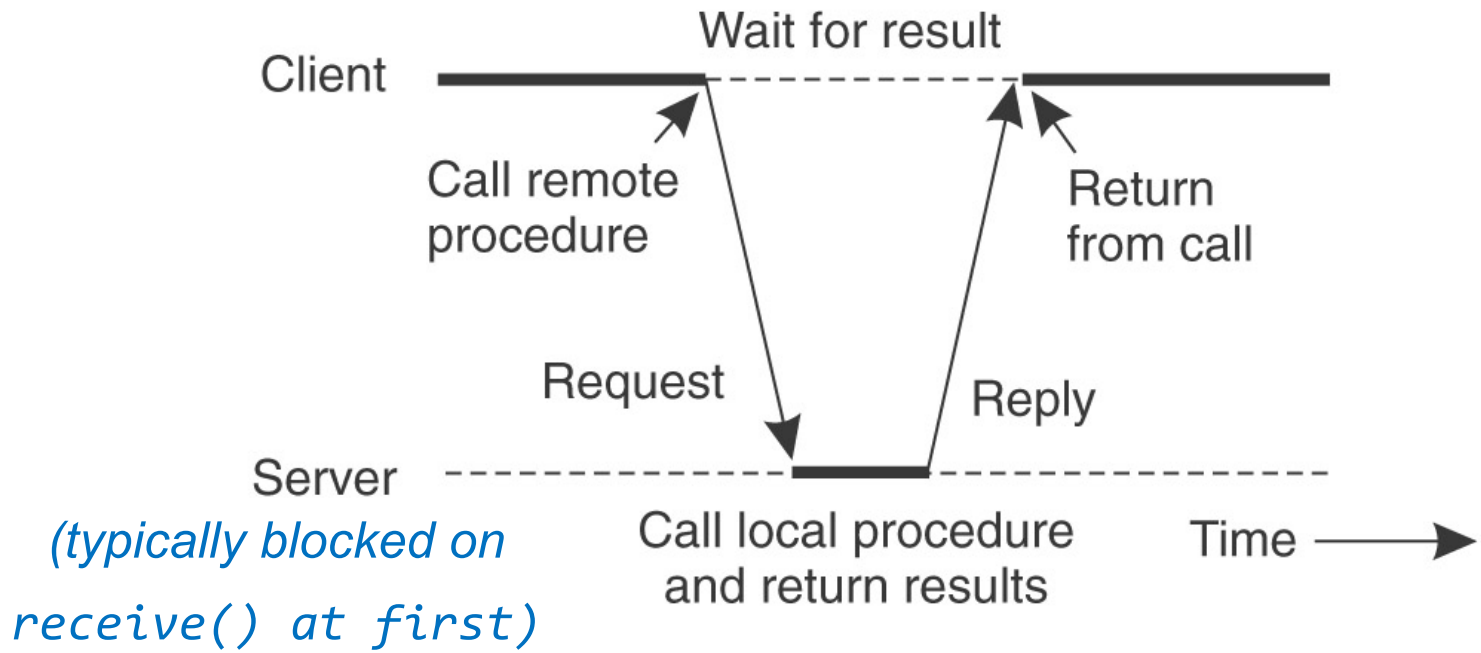
- Linker inserts read implementation into obj file
- Implementation usually invokes a system call

[Tanenbaum & van Steen, Fig 4-5]

How does a RPC work?

Basic idea:

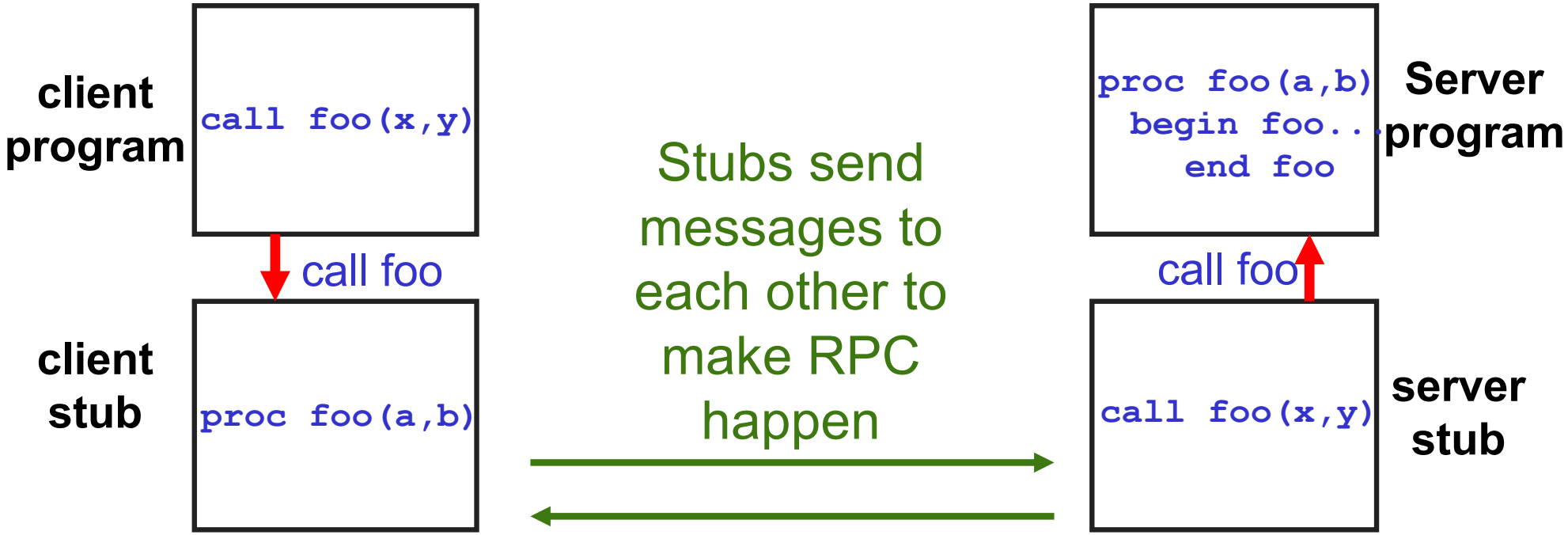
- Server *exports* a set of procedures
- Client calls these procedures, as if they were local functions



- Message passing details hidden from client & server (like system call details are hidden in libraries)

[Tanenbaum & van Steen, Fig 4-6]

RPC Stubs



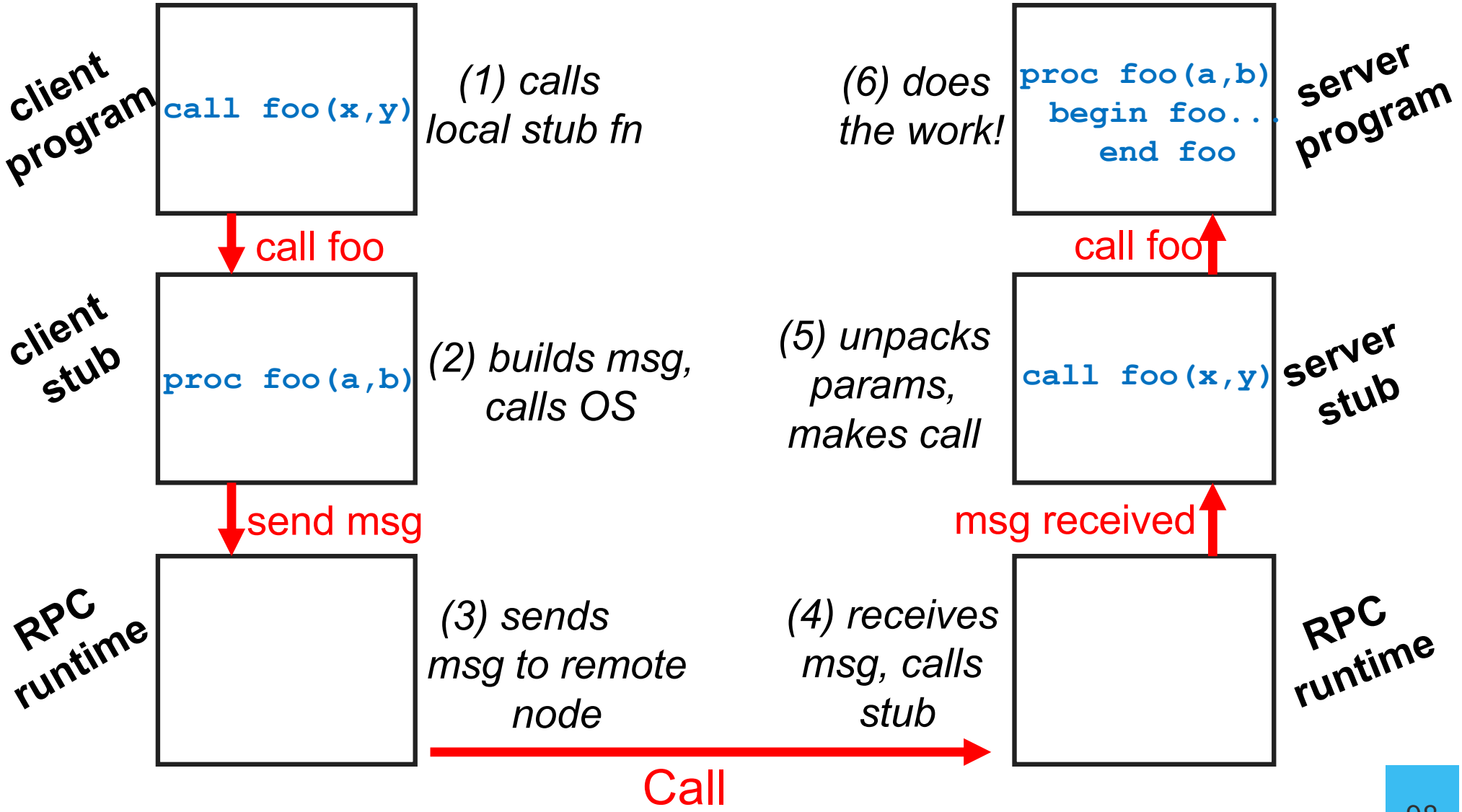
Client-side stub:

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

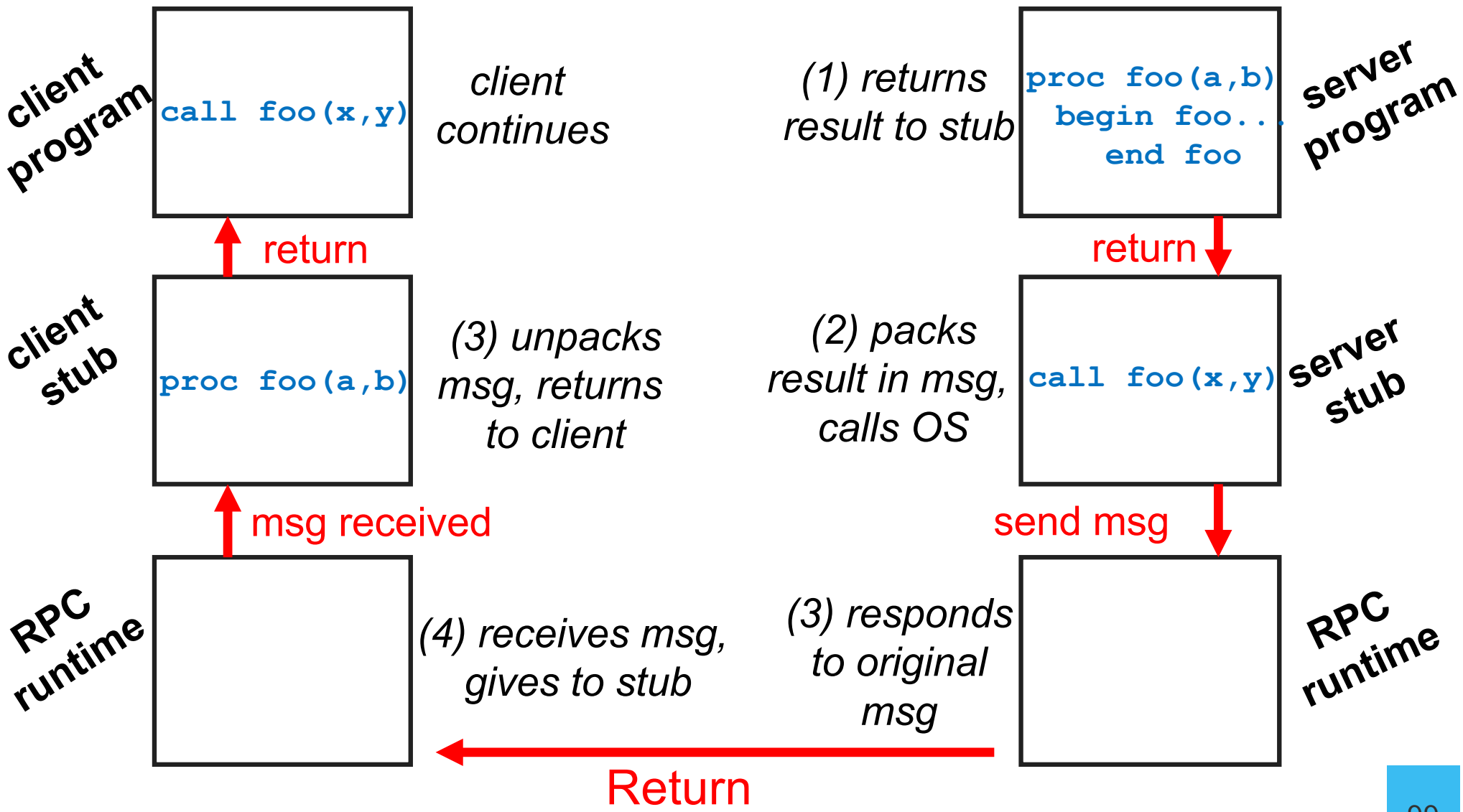
Server-side stub:

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

RPC Call Structure



RPC Return Structure

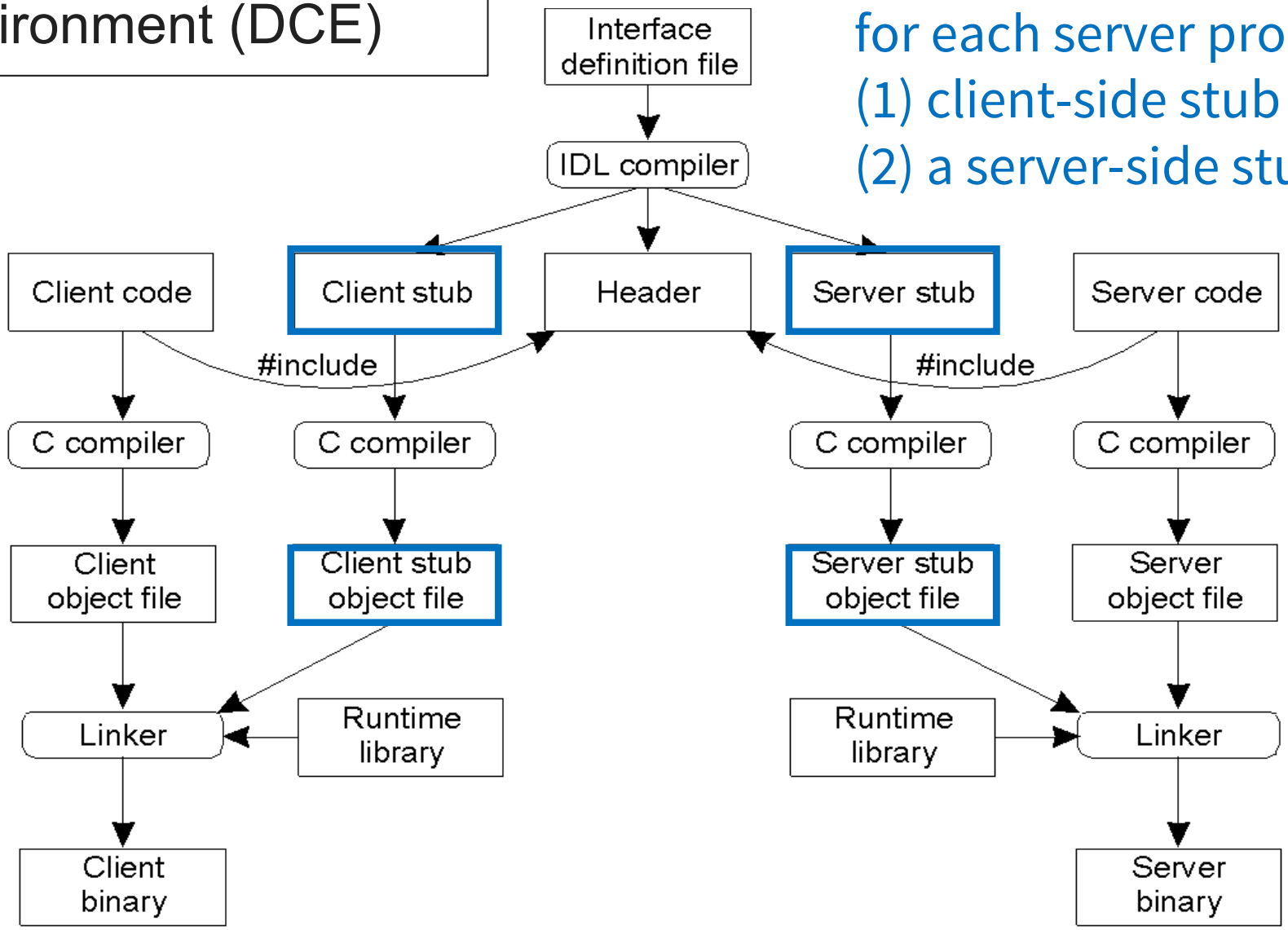


Example RPC system:

Distributed Computing Environment (DCE)

Stub compiler

- reads IDL
- produces 2 stub procedures for each server procedure
  (1) client-side stub
  (2) a server-side stub

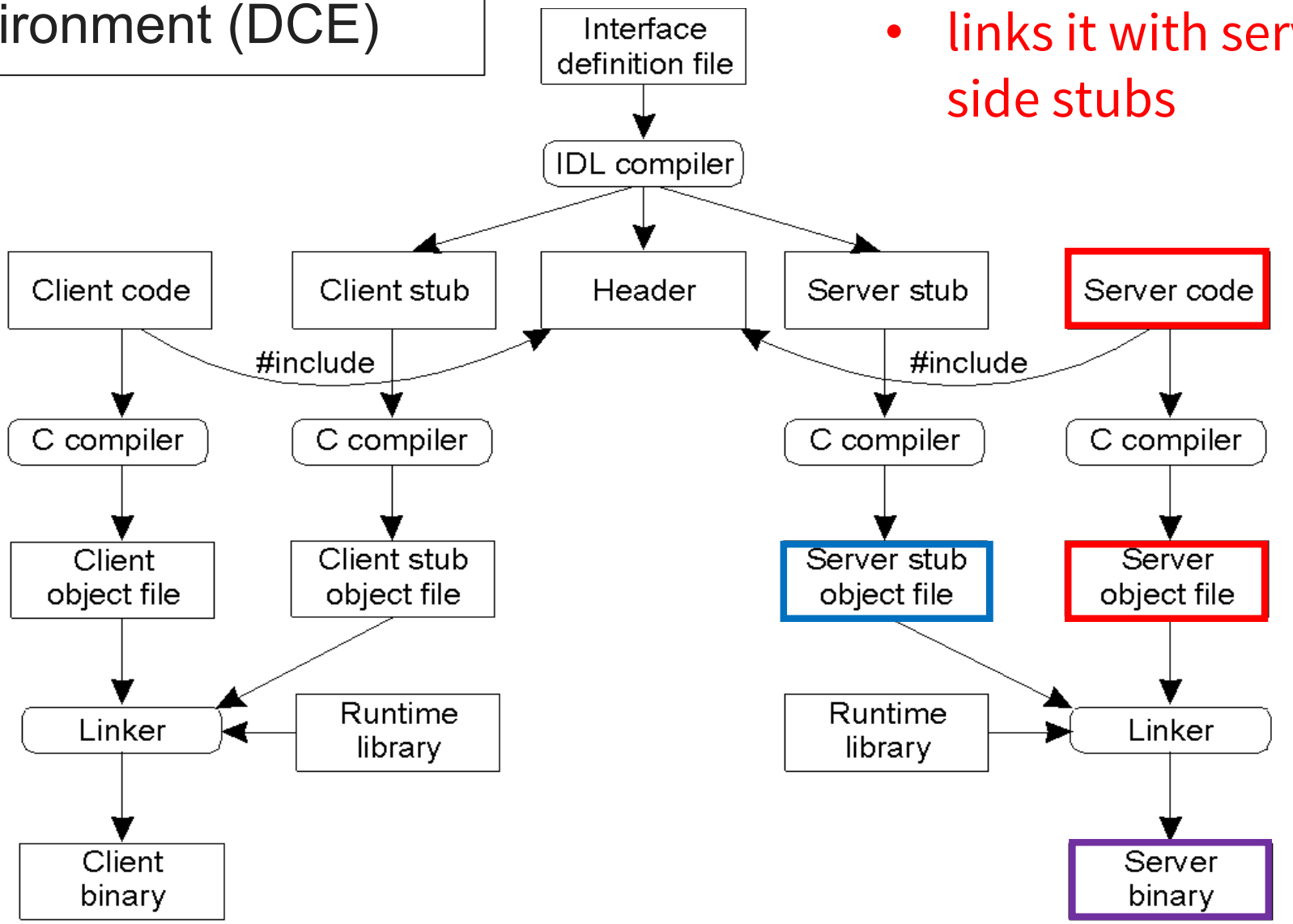


Example RPC system:

Distributed Computing Environment (DCE)

Server writer:

- writes server
- links it with server-side stubs



Binding: Connecting Client & Server

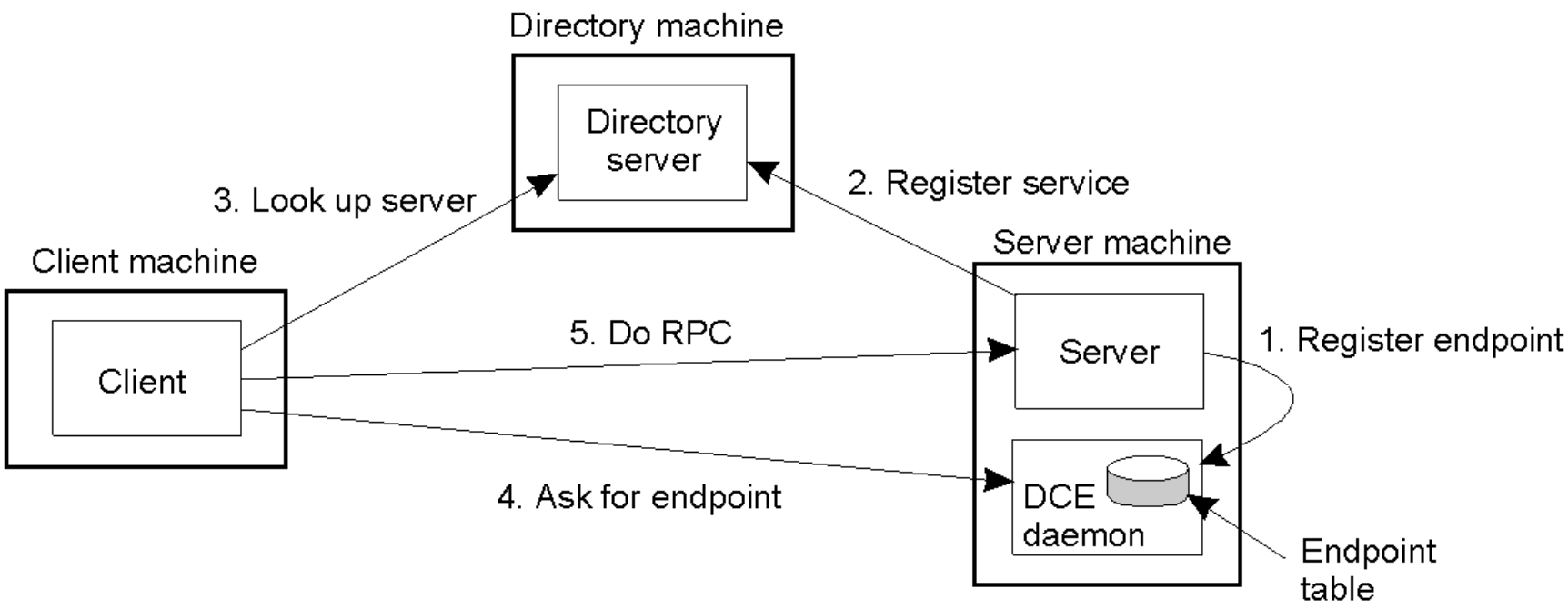
Server *exports* its interface:

- identifying itself to a network name server
- telling the local runtime its dispatcher address

Client *imports* the server. RPC runtime:

- looks up the server through the name service
- contacts requested server to set up a connection

Import and *export* are explicit calls in the code



RPC Concerns

- Parameter Passing
- Failure Cases
- Performance

Your function call has been secretly replaced with a remote function call. Is this okay?

RPC Marshalling

Packing parameters into a message packet

- RPC stubs call type-specific procedures to marshall (or unmarshall) all of the parameters to the call

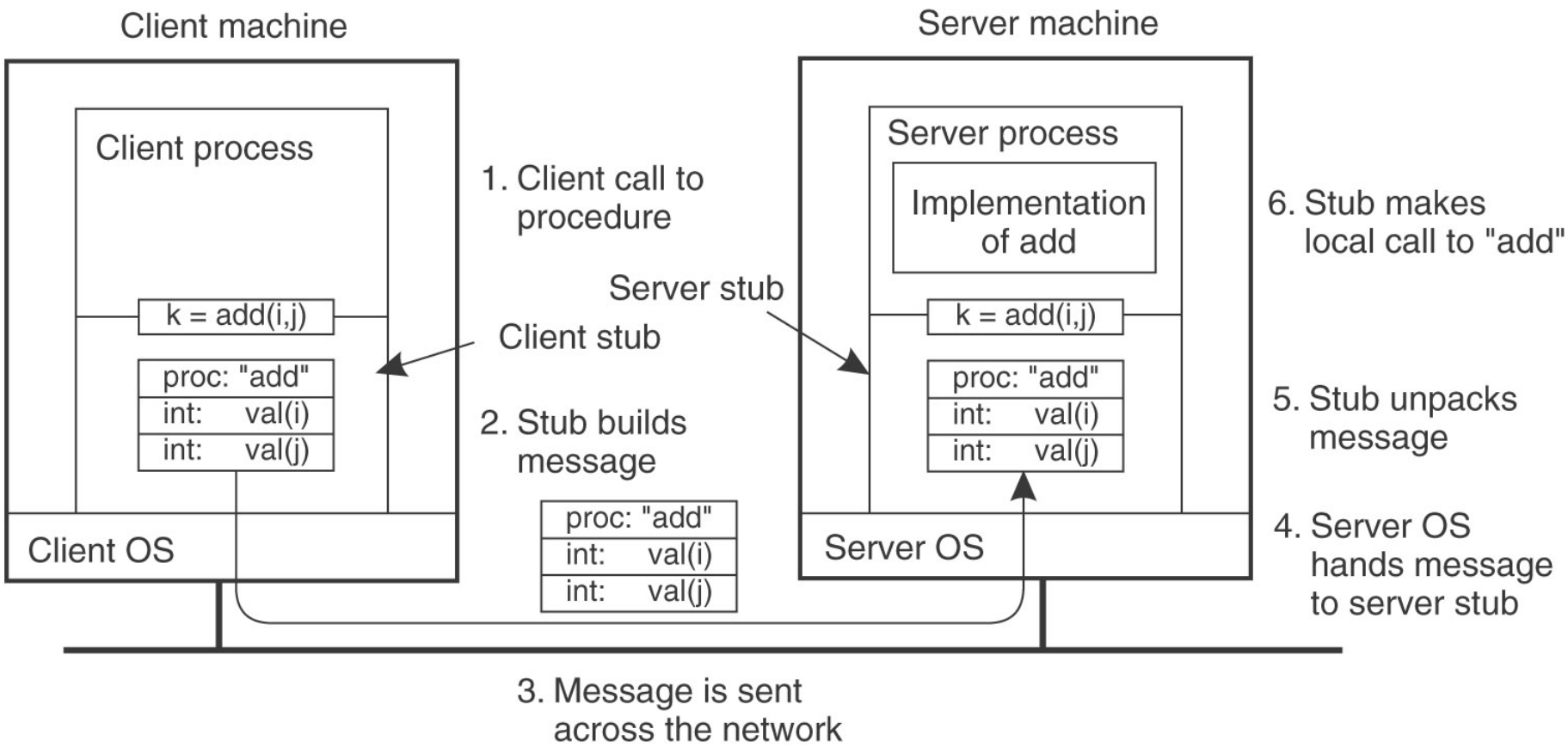
On Call:

- Client stub marshalls parameters into the call packet
- Server stub unmarshalls parameters to call server's fn

On return:

- Server stub marshalls return values into return packet
- Client stub unmarshalls return values, returns to client

Parameter Passing



What could go wrong?

[Tanenbaum & van Steen, Fig 4-7]

RPC Concerns

- Parameter Passing
  - Data Representation
  - Passing Pointers
  - Global Variables
- Failure Cases
- Performance

Data Representation

Data representation?

ASCII vs. Unicode, structure alignment, n-bit machines, floating-point representations, endianness

→Server program defines interface using an *interface definition language* (IDL)

For all client-callable functions, IDL specifies:

- names
- parameters
- types

Passing Pointers

- Forbid pointers? (breaks transparency)
- Have server call client and ask it to modify when needed (breaks transparency)
- Have stubs replace call-by-reference semantics with Copy/Restore
  - Optimization: if stub knows that a reference is exclusively input/output copy only on call/return
  - Only works for simple arrays & structures
    - Union types? YUCK
    - Multi-linked structures? YUCK
    - Raw pointers? YUCK

RPC Concerns

- Parameter Passing
- Failure Cases
- Performance

RPC Failure Cases

Function call failure cases:

- Called fn crashes → so does the caller

RPC Failure cases:

- server fine, client crashes? (orphans)
- client fine, server crashes?
  - Client just hangs?
  - Stub supports a timeout, error after n tries?
  - Client deals w/failure (breaks transparency)

Aside: Idempotency

Multiple calls yields the same result

What's idempotent?

- read block 50

What's not?

- appending a file
- most I/O

How many times will a function be executed?

A calls B. B never responds… Should A resend or not?

2 Possibilities:

(1) B never got the call:

- Resend → B executes the procedure *once*
- Don't resend → B executes the procedure *zero times*

(2) B performed the call then crashed:

- Resend → B executes the procedure *twice*
- Don't resend → B executes the procedure *once*

Can we even promise transparency?

What semantics will RPC support?

A calls B. B responds… What does A assume about how many times the function was executed?

Exactly once:

- system guarantees local semantics
- at best expensive, at worst, impossible

At-least-once:

+ easy: no response? A re-sends
− only works for idempotent functions
− server operations must be stateless

At-most-once:

− requires server to detect duplicate packets
+ works for non-idempotent functions

RPC Concerns

- Parameter Passing
- Failure Cases
- Performance
  - Remote is not cheap
  - Lack of parallelism (on both sides)
  - Lack of streaming (for passing data)

RPC Concluding Remarks

RPC:

- Common model for distributed application communication
- ***language support*** for distributed programming
- relies on a *stub compiler* & IDL server description
- commonly used, *even on a single node*, for communication between applications running in different address spaces (most RPCs *are* intra-node!)

"Distributed objects are different from local objects, and keeping that difference visible will keep the programmer from forgetting the difference and making mistakes."
–Jim Waldo+, "A Note on Distributed Computing" (1994)