NETWORKS

Who recognizes this?

```
int sockfd;
struct sockaddr_in addr;

addr.sin_family = AF_INET;
addr.sin_addr.s_addr =
          inet_addr(SERV_HOST_ADDR);
addr.sin_port = htons(SERV_TCP_PORT);

sockfd = socket(AF_INET, SOCK_STREAM, 0);
connect(sockfd, (struct sockaddr *) &addr,
          sizeof(serv_addr));
do_stuff(stdin, sockfd);
```

Classic view of network API

- Start with host name (maybe)

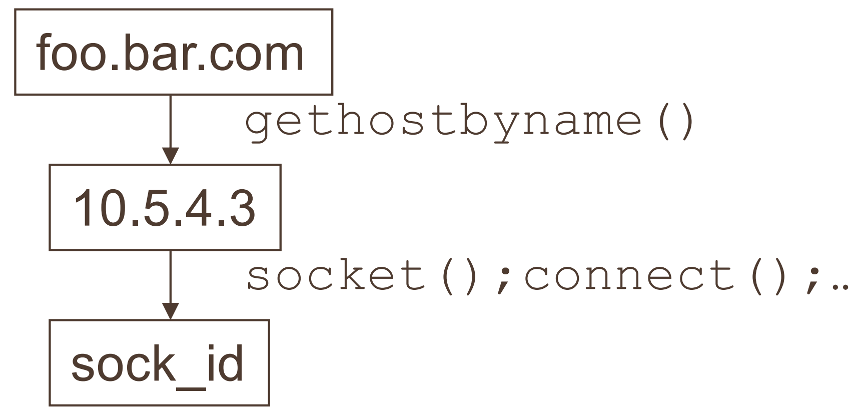
foo.bar.com

Classic view of network API

- Start with host name
- Get an IP address

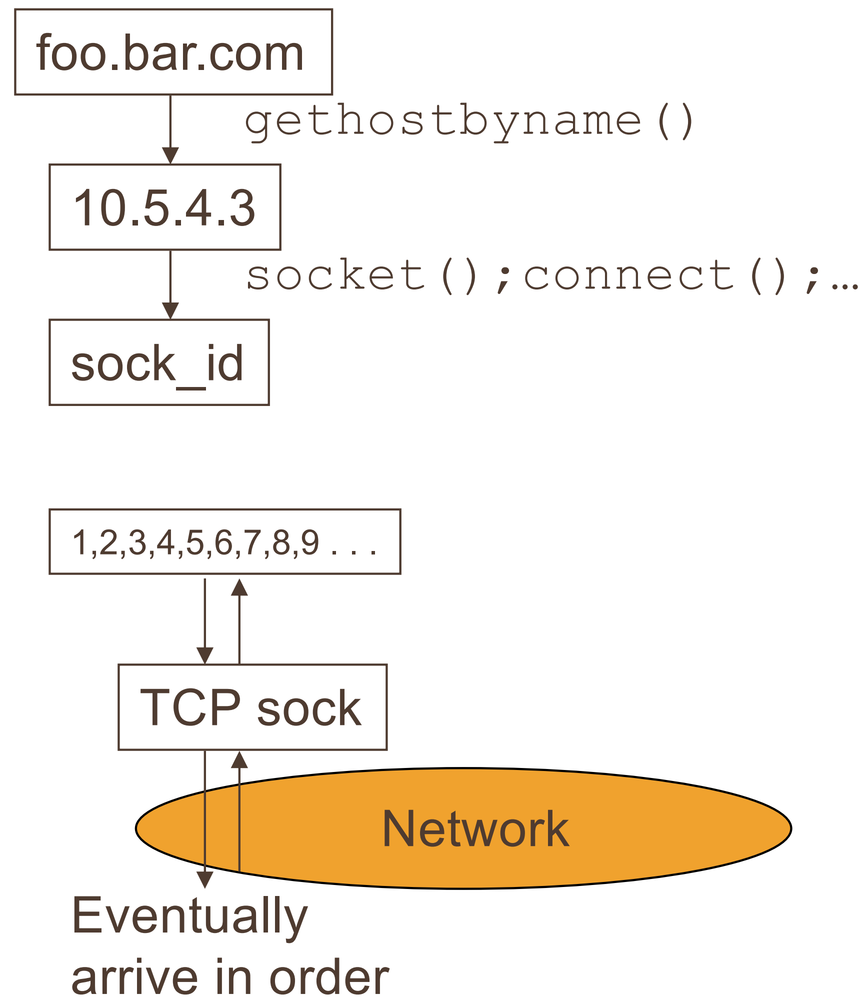
Classic view of network API

- Start with host name
- Get an IP address
- Make a socket (protocol, address)



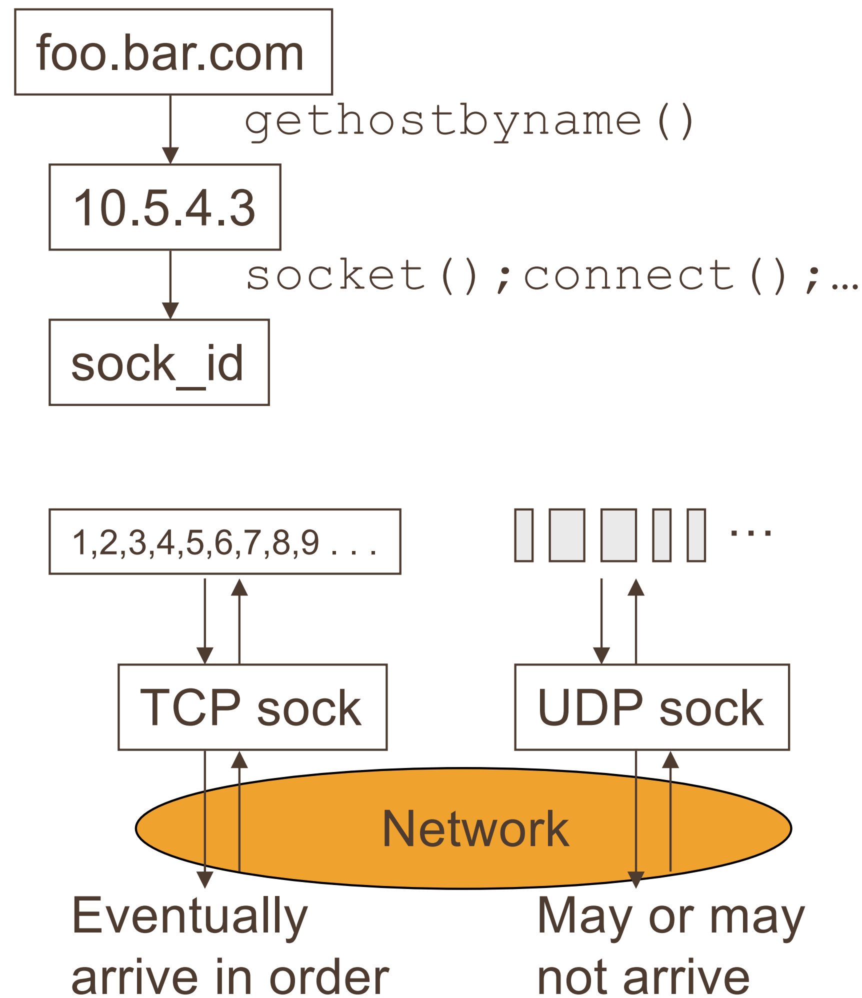
Classic view of network API

- Start with host name
- Get an IP address
- Make a socket (protocol, address)
- Send byte stream (TCP)



Classic view of network API

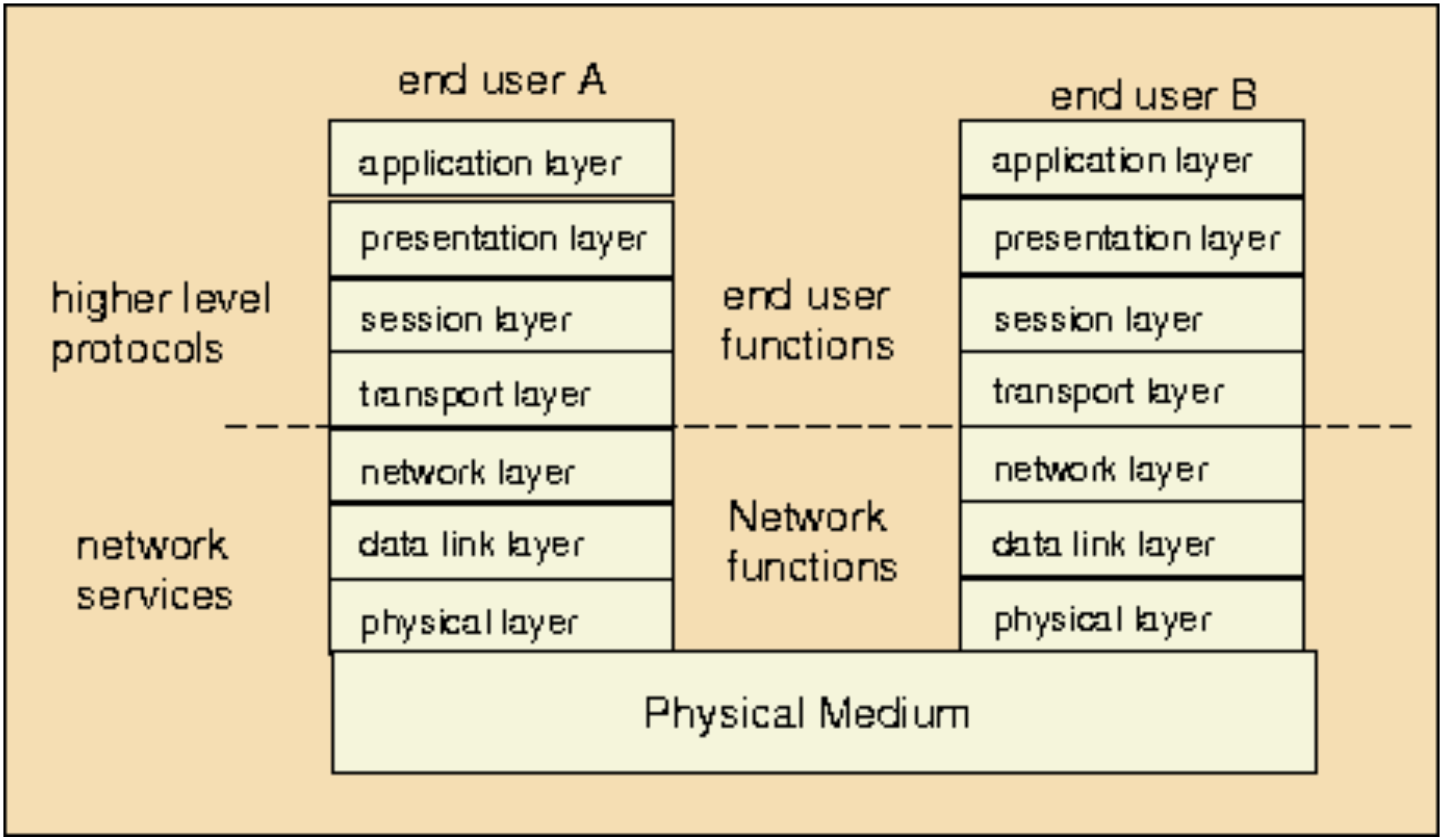
- Start with host name
- Get an IP address
- Make a socket (protocol, address)
- Send byte stream (TCP) or packets (UDP)



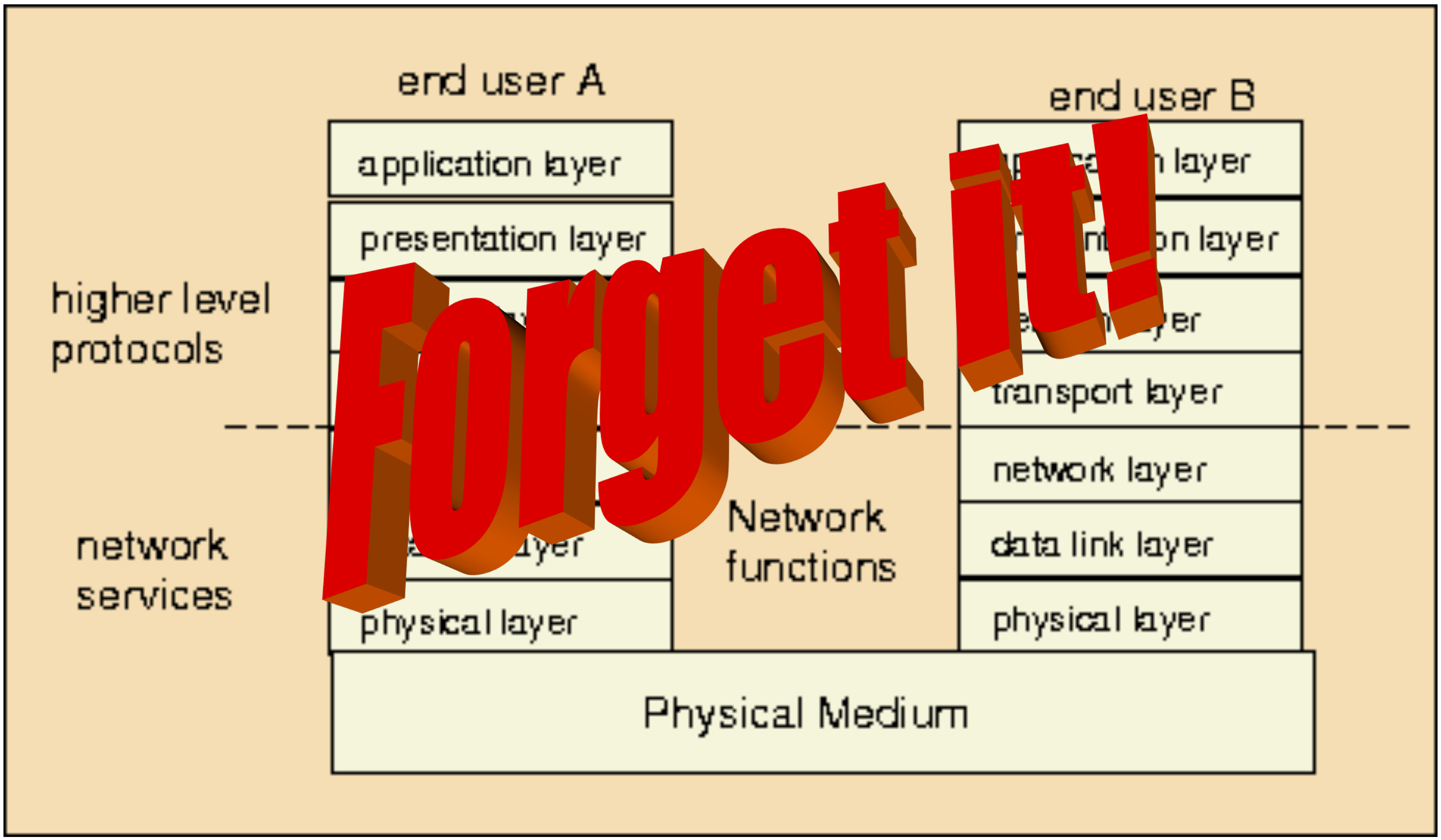
Classic approach “broken” in many ways

- **DNS:** IP address different depending on who asks
- **NAT:** Address may be changed in transit
- **Firewall:** IP address may not be reachable
  - Or may be reachable by you but not another host
- **DHCP:** IP address may change
- **Caches:** Packets may not come from who you think

Classic OSI stack



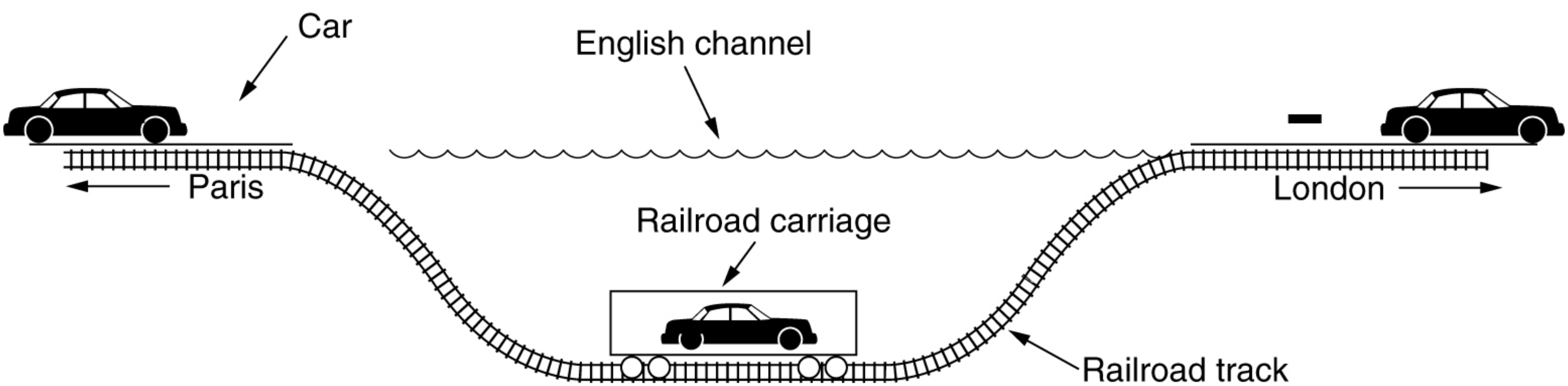
Classic OSI stack



TUNNELING

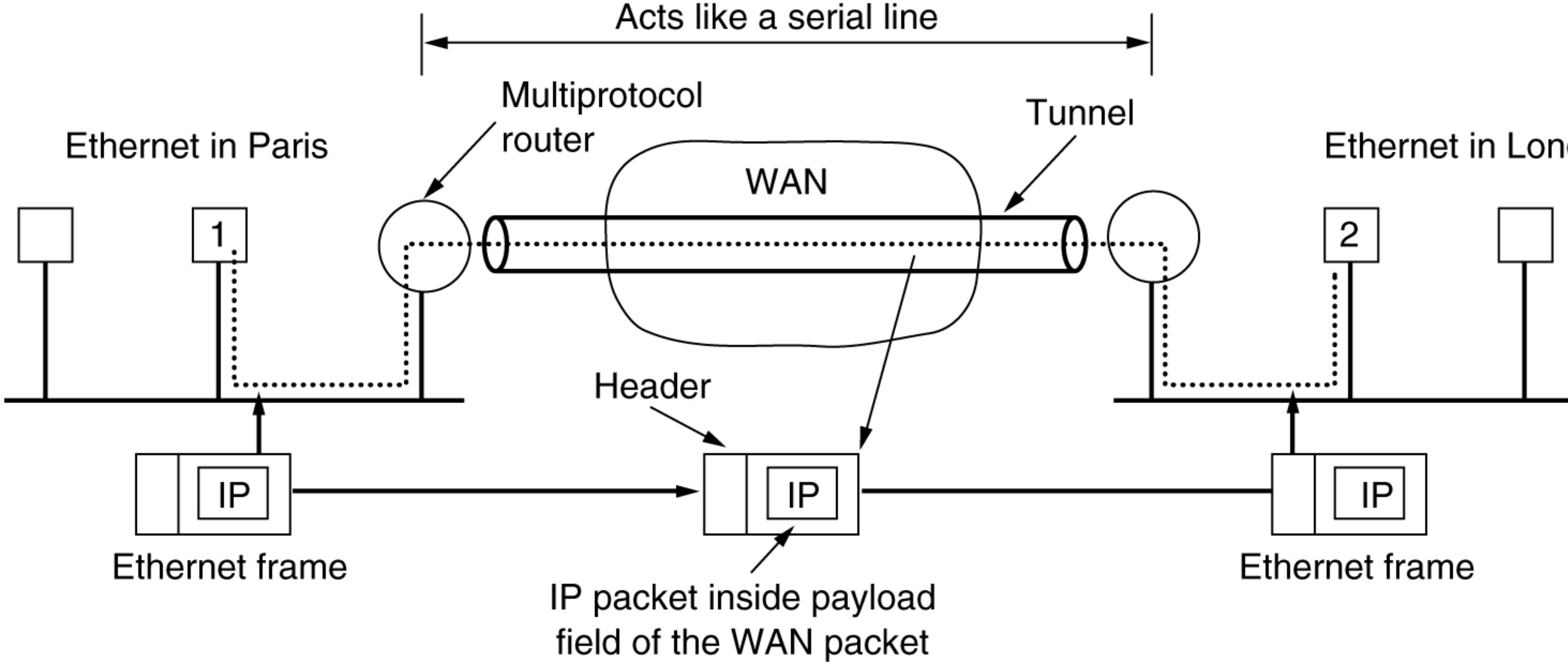
Tunneling Analogy

Tunneling a car from France to England.



Tunneling in Networks

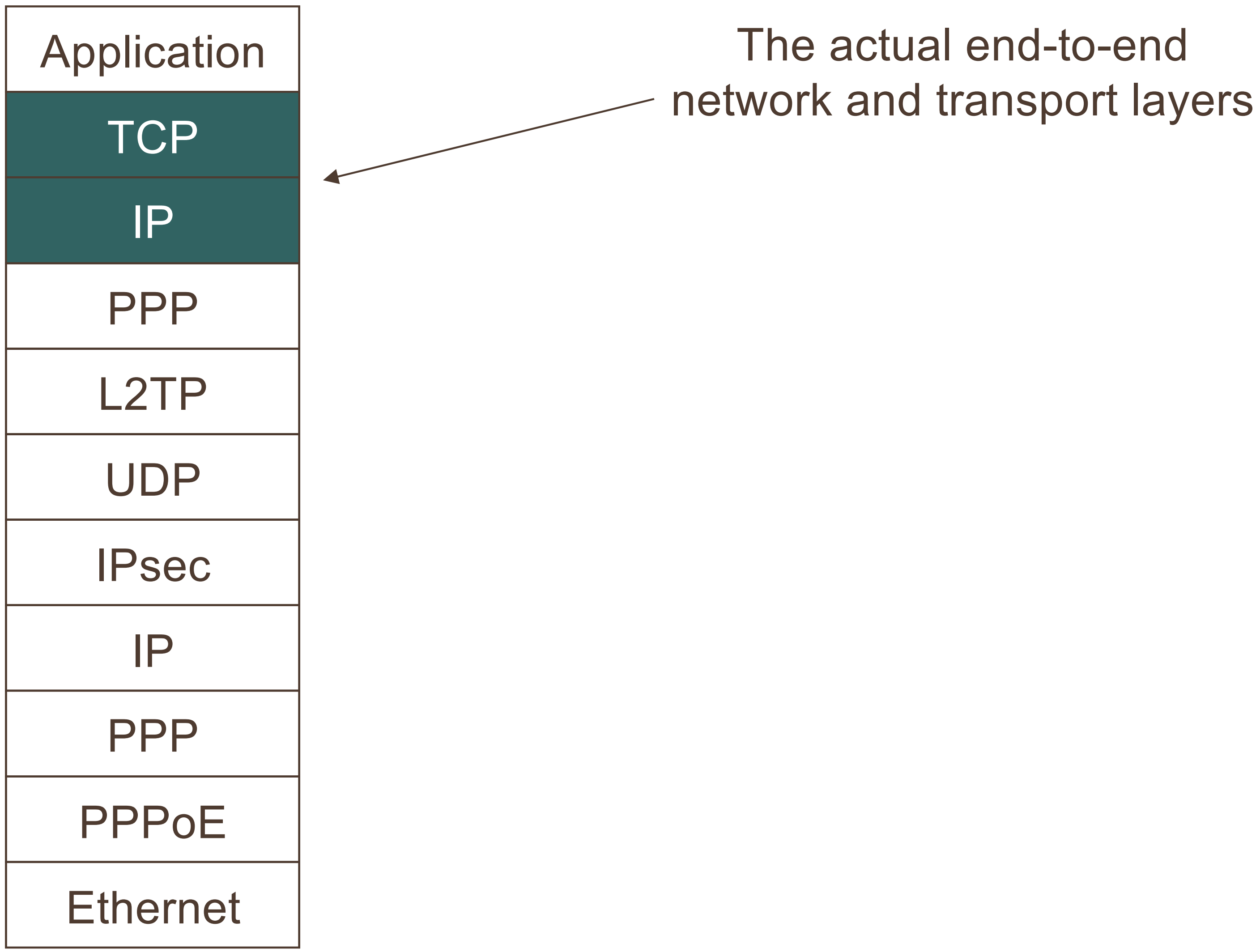
Tunneling a packet from Paris to London.



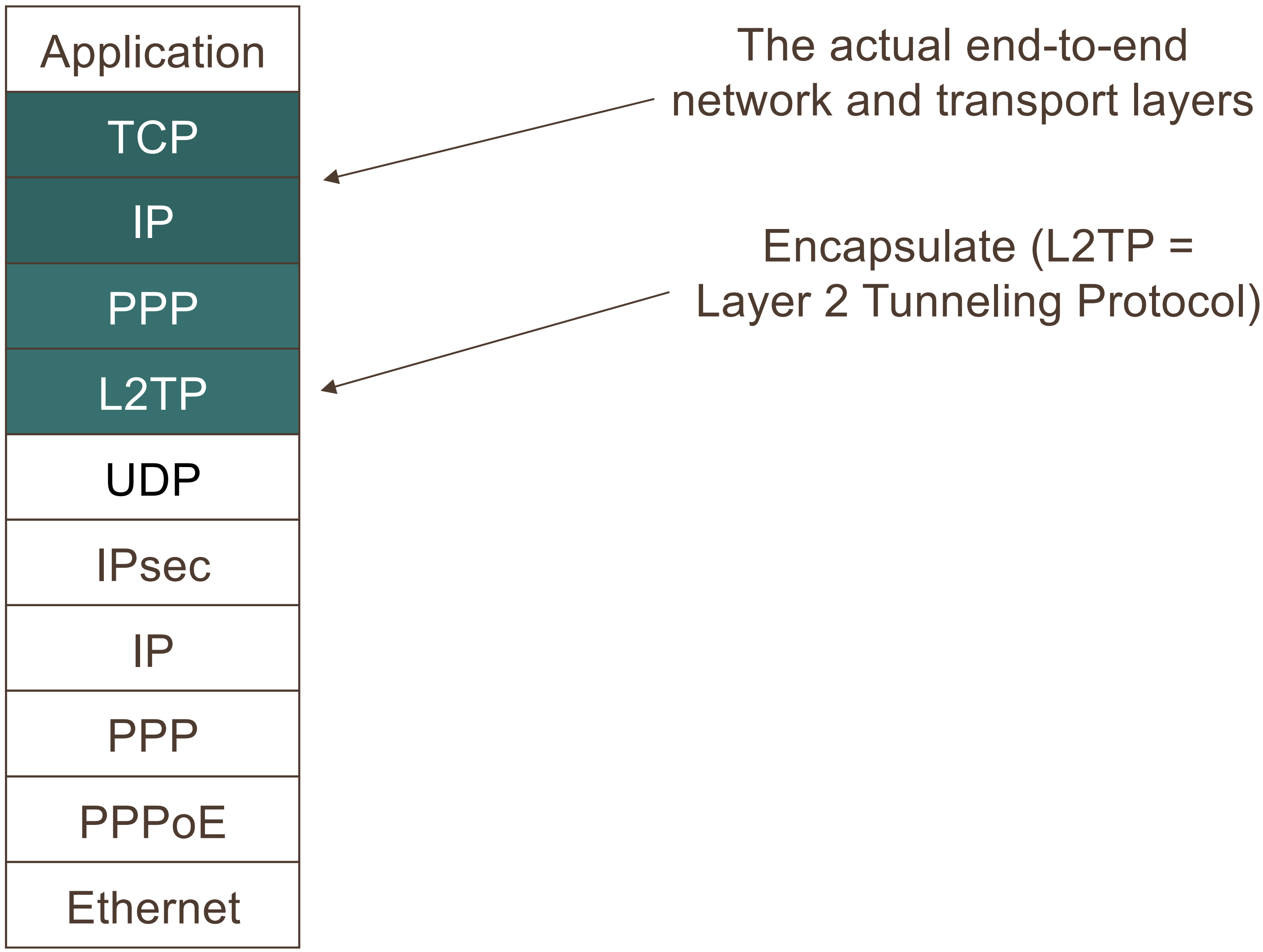
Example Microsoft VPN stack

Application
TCP
IP
PPP
L2TP
UDP
IPsec
IP
PPP
PPPoE
Ethernet

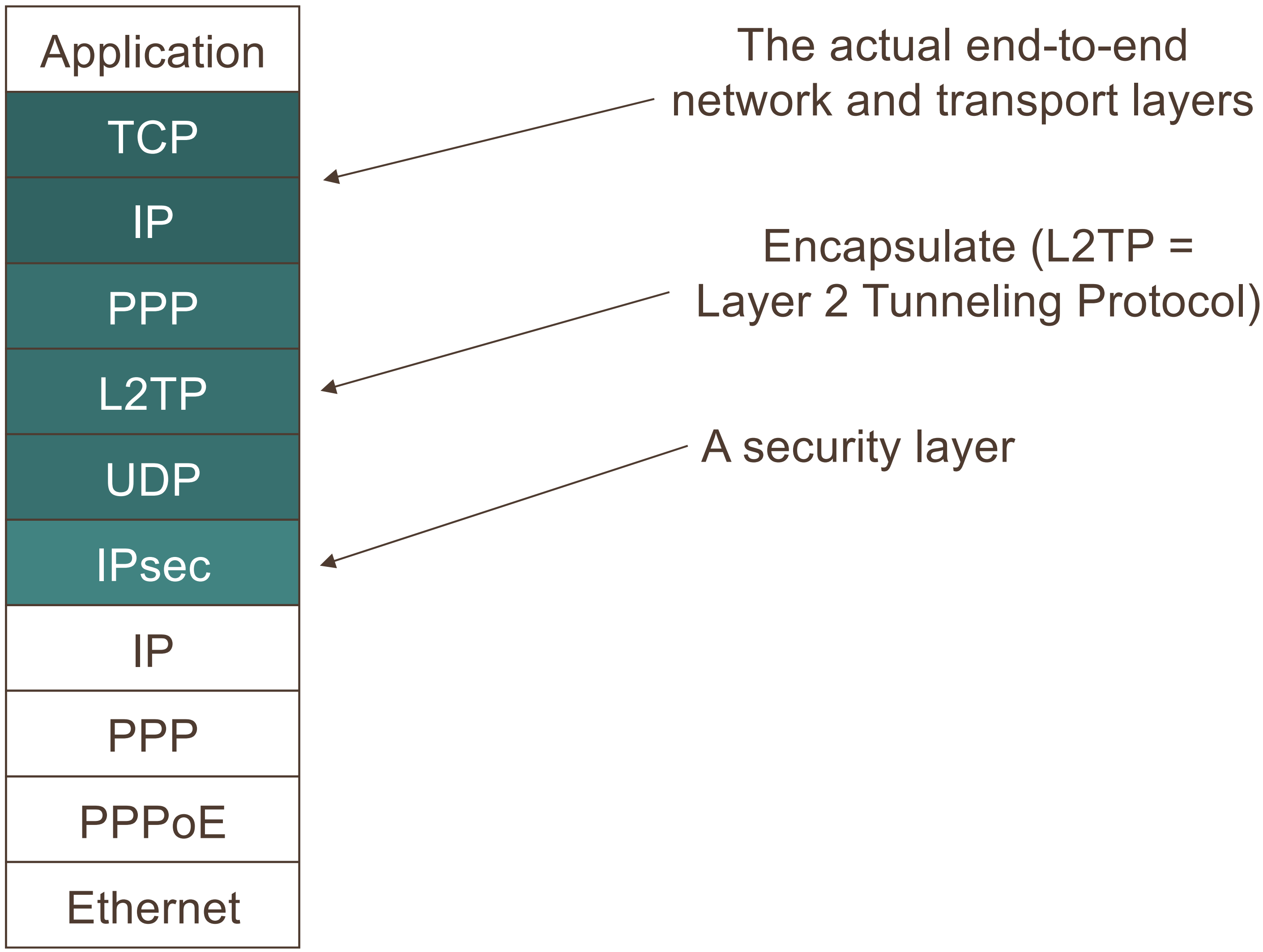
Example Microsoft VPN stack



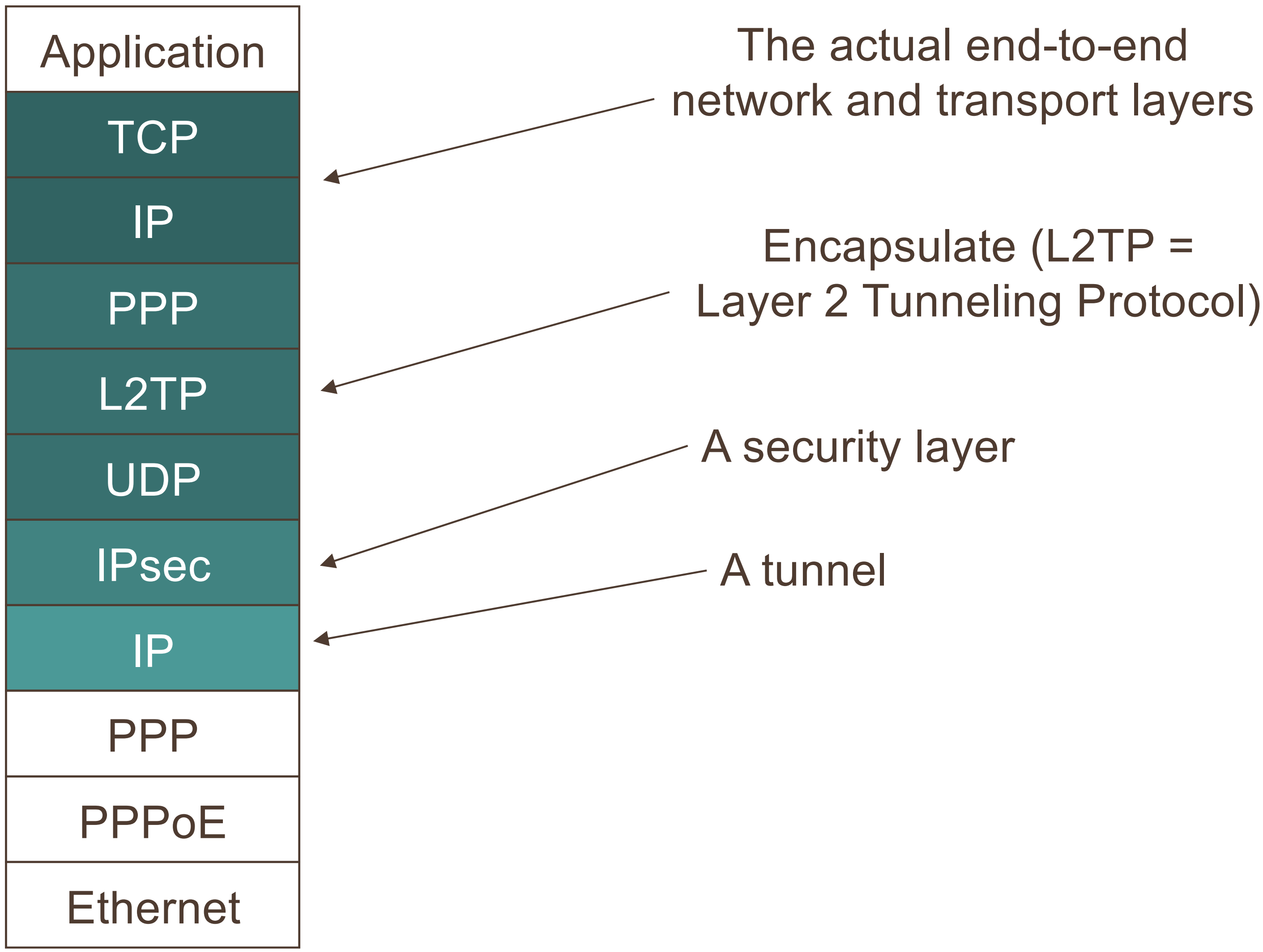
Example Microsoft VPN stack



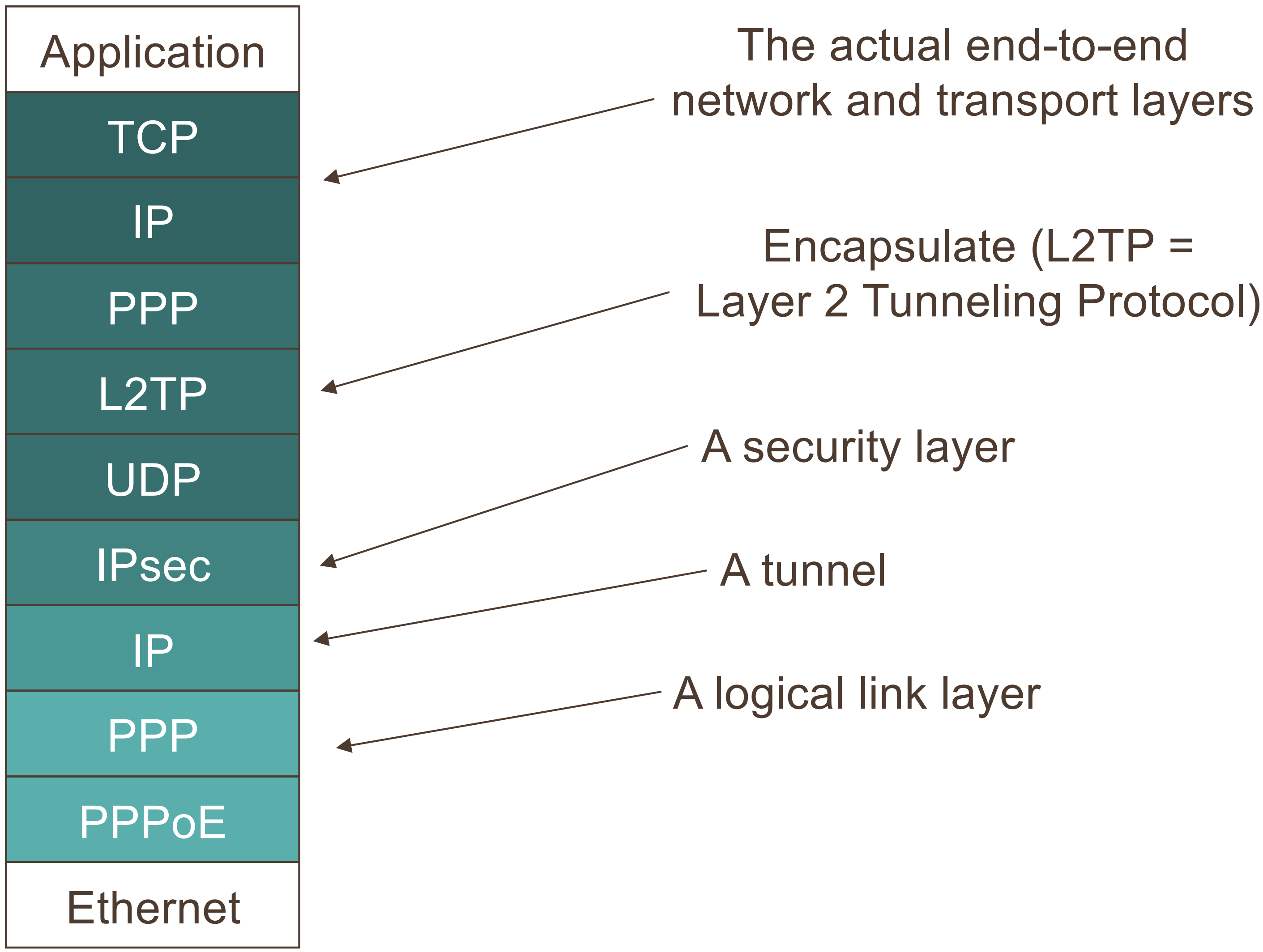
Example Microsoft VPN stack



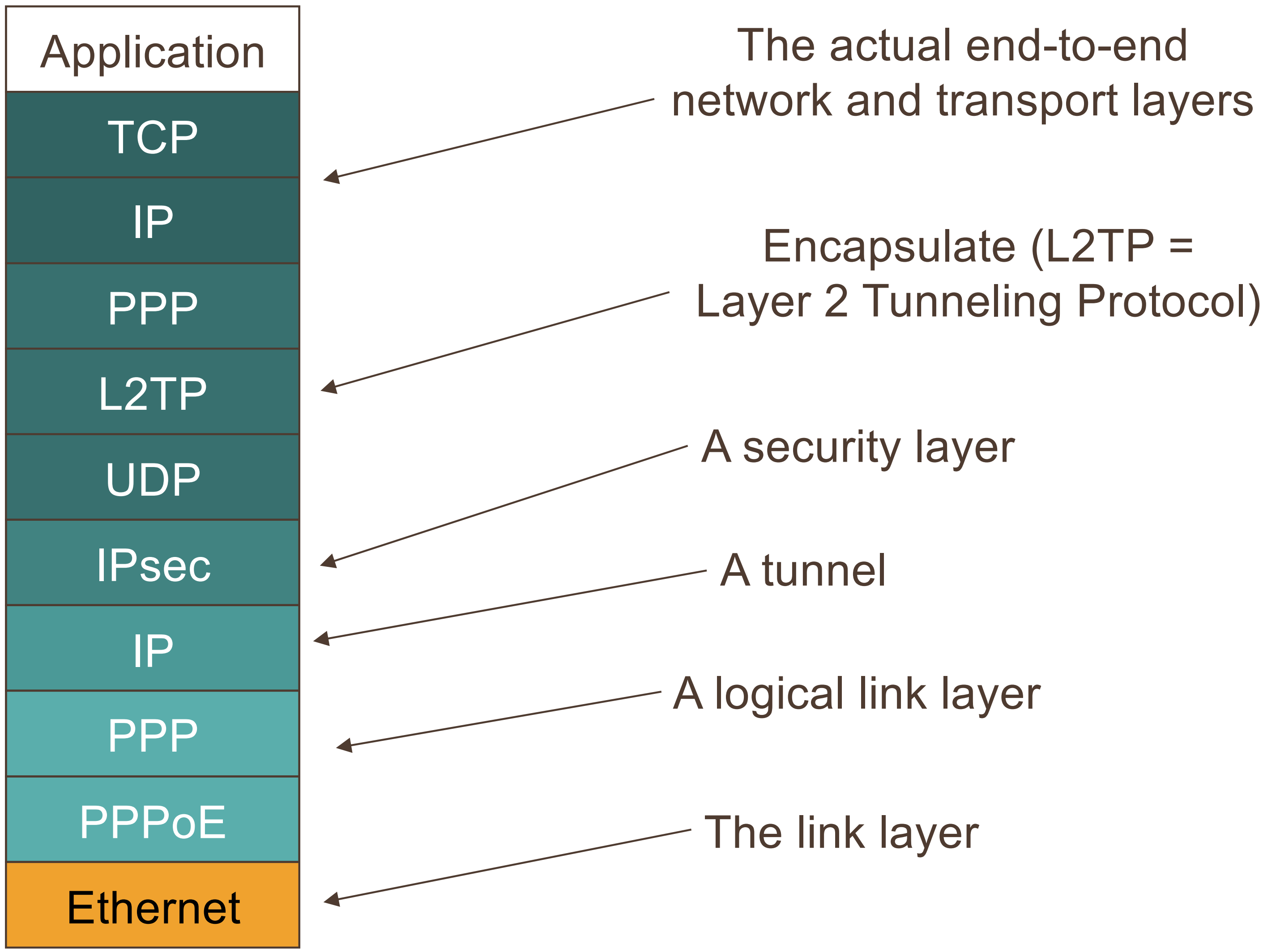
Example Microsoft VPN stack



Example Microsoft VPN stack



Example Microsoft VPN stack



Example Microsoft VPN stack

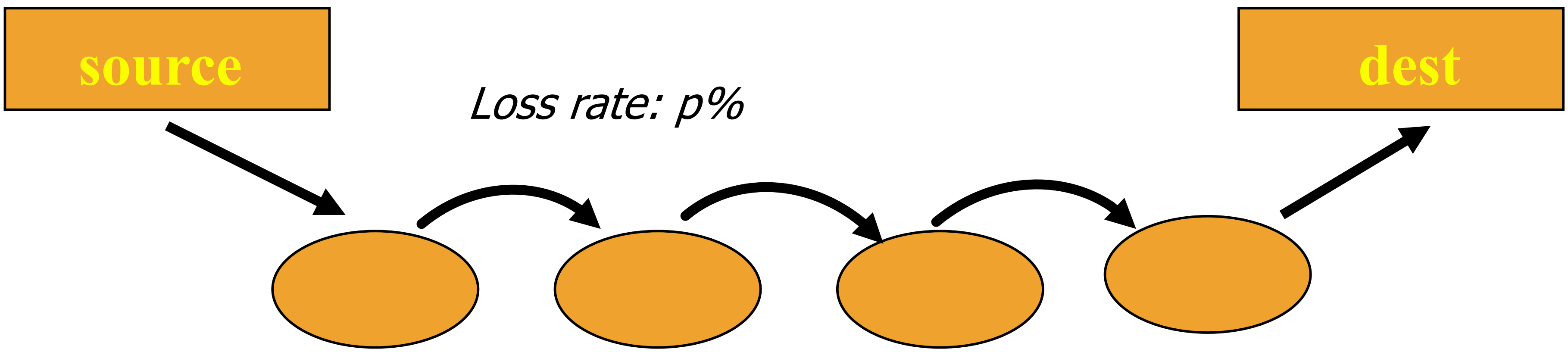
TCP:	Transport Control Protocol
IP:	Internet Protocol
PPP:	Point-to-Point Protocol
L2TP:	Layer 2 Tunneling Protocol
UDP:	User Datagram Protocol
IPsec:	Secure IP
PPPoE:	PPP over Ethernet

END-TO-END ARGUMENT

End-to-End argument

- Internet:
  - Suppose an IP packet will take n hops
  - probability p of loss on each hop
  - Transfer a file of k IP packets
  - Should we:
    - use a retransmission protocol running “end-to-end” or
    - n TCP protocols in a chain?

End-to-End argument



Probability of successful transit: $(1\text{-}p)^n$,

Expected packets lost: $k\text{-}k^*(1\text{-}p)^n$

Saltzer et. al. analysis

- If p is <u>very</u> small, then even with many hops most packets will get through
  - Overhead of using TCP protocols in the links
  - End-to-end recovery mechanism

Generalized End-to-End view?

- Low-level mechanisms should focus on speed, not reliability
- The application should worry about "properties" it needs
- In general, add additional functionality *end-to-end in the application*
  - Not in the network

What can we learn from this?

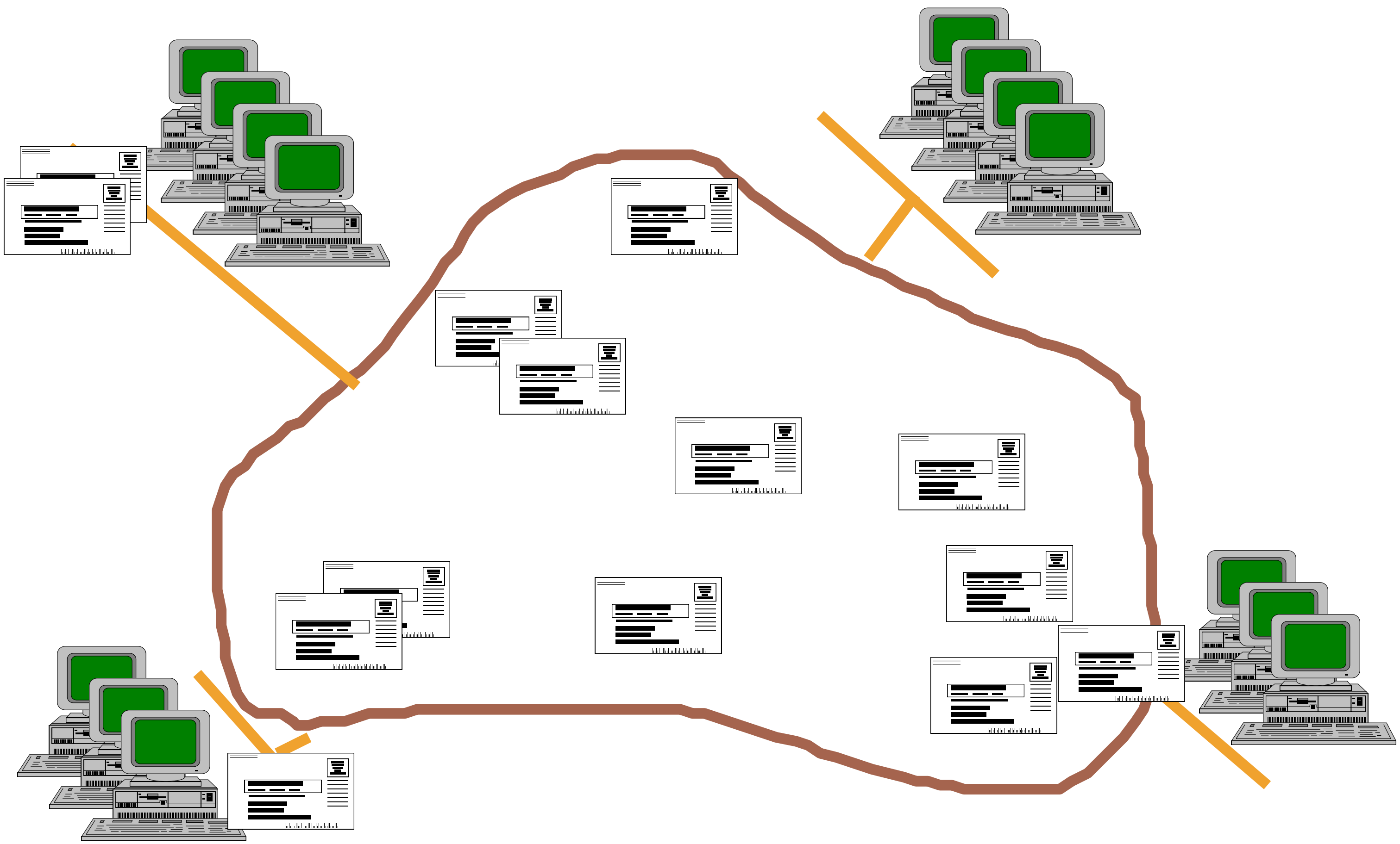
- That the internet is a mature technology
  - Kludges on kludges
- *That the end-to-end argument actually works!*

When should the network do more?

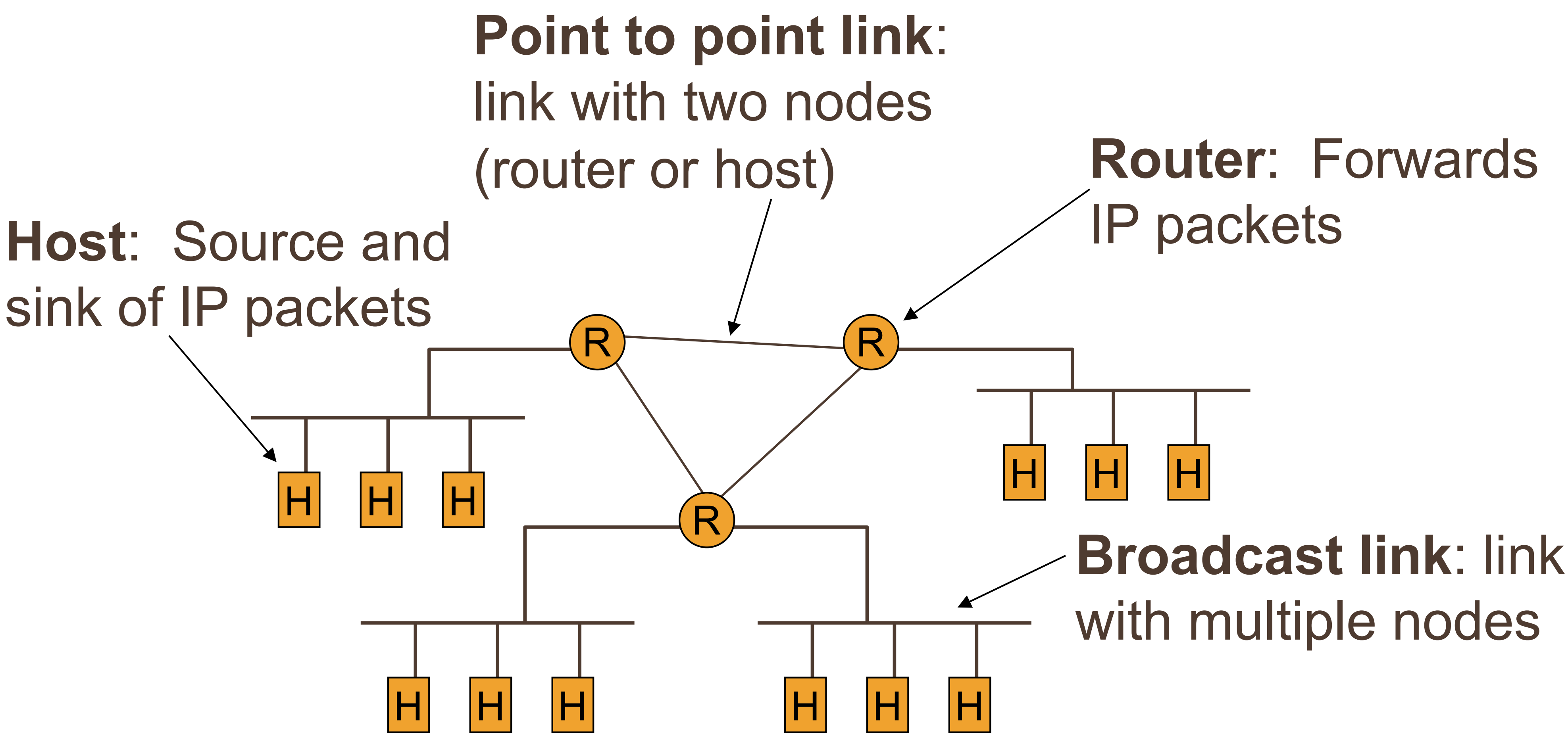
- When you get performance gains
  - Link-level retransmissions over a lossy link
  - Ex: wireless network
- Also
  - When the network doesn’t trust the end user
    - Corporation or military
  - Some things can’t be done at the end
    - Routing algorithms
    - Billing
    - User authentication

NETWORK INFRASTRUCTURE

A network is like a “mostly reliable” post office



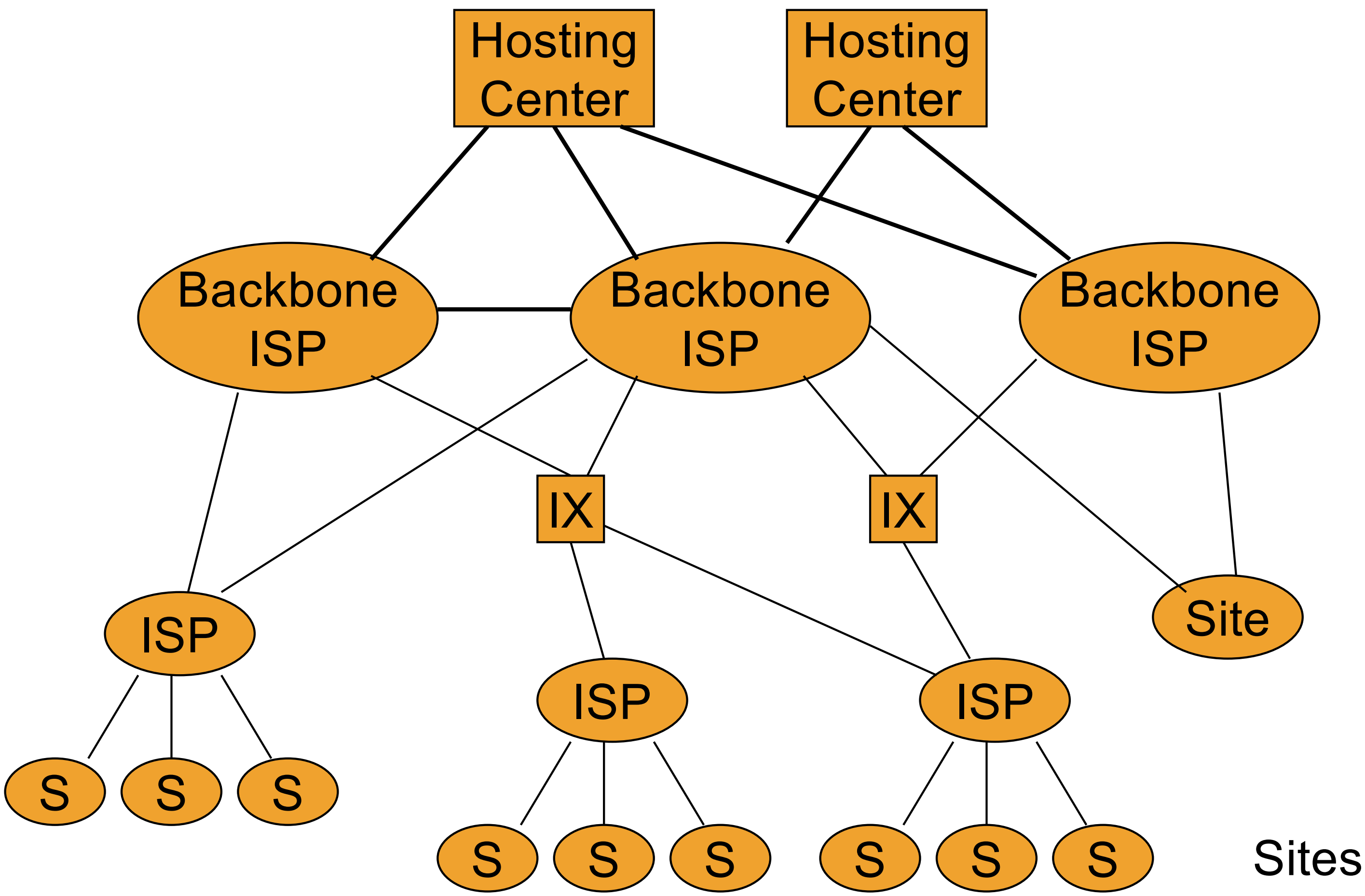
Network components



Network components

- **Network:** Collection of hosts, links, and routers
- **Site:** Stub network, typically in one location and under control of one administration
- **Firewall/NAT:** Box between the site and ISP
- **ISP:** Internet Service Provider. Transit network that provides IP connectivity for sites
- **Backbone ISP:** Transit network for regional ISPs and large sites
- **Inter-exchange (peering point):** Broadcast link where multiple ISPs connect and exchange routing information (peering)
- **Hosting center:** Stub network that supports lots of hosts (web services), high speed connections to many backbone ISPs.
- **Bilateral peering:** Direct connection between two backbone ISPs

Internet topology



IXs came first

IXs tend to be performance bottlenecks

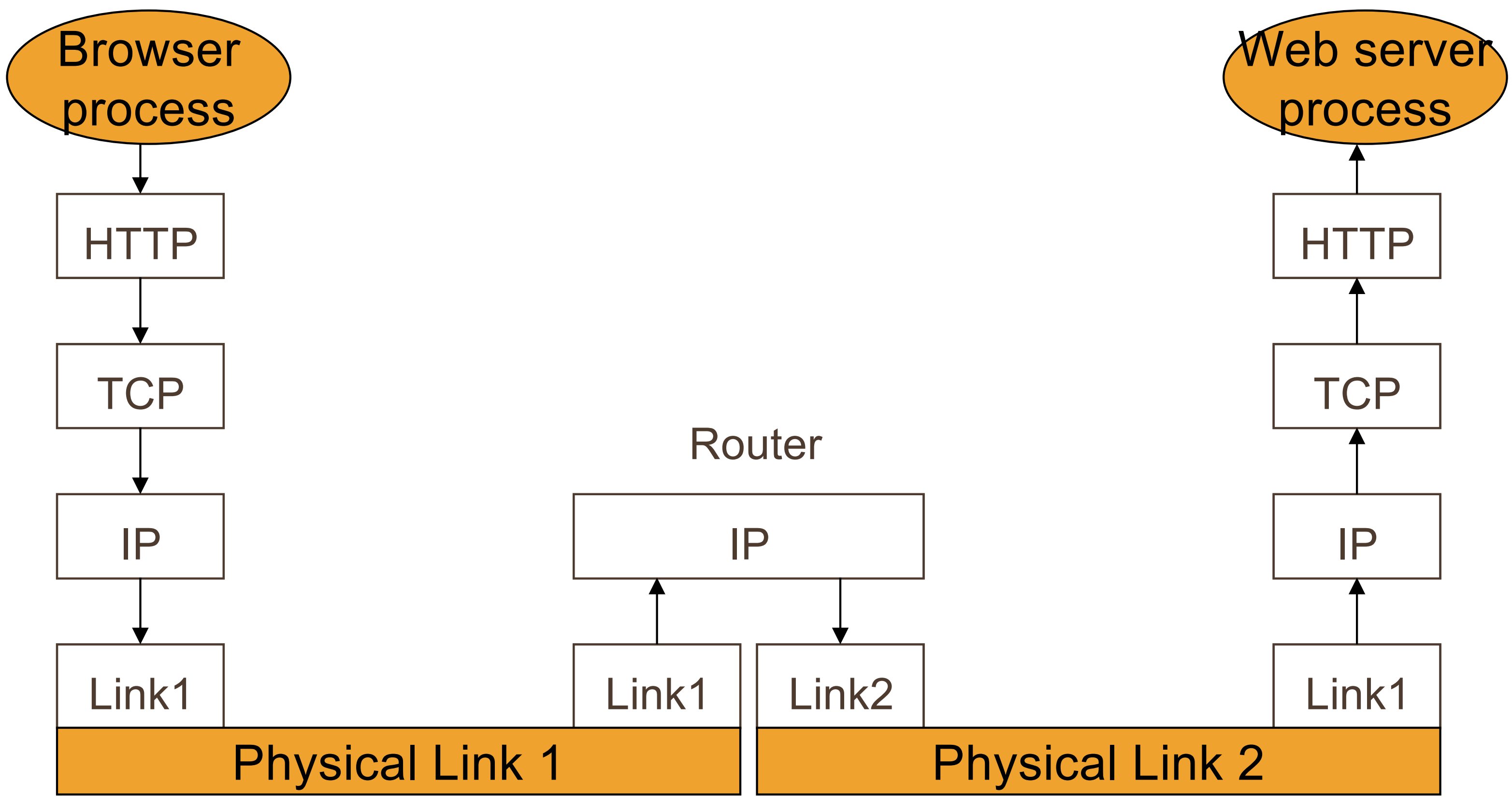
Hosting centers and bilateral peering are a response to poor IXs

NETWORK PROTOCOLS

Protocol layering

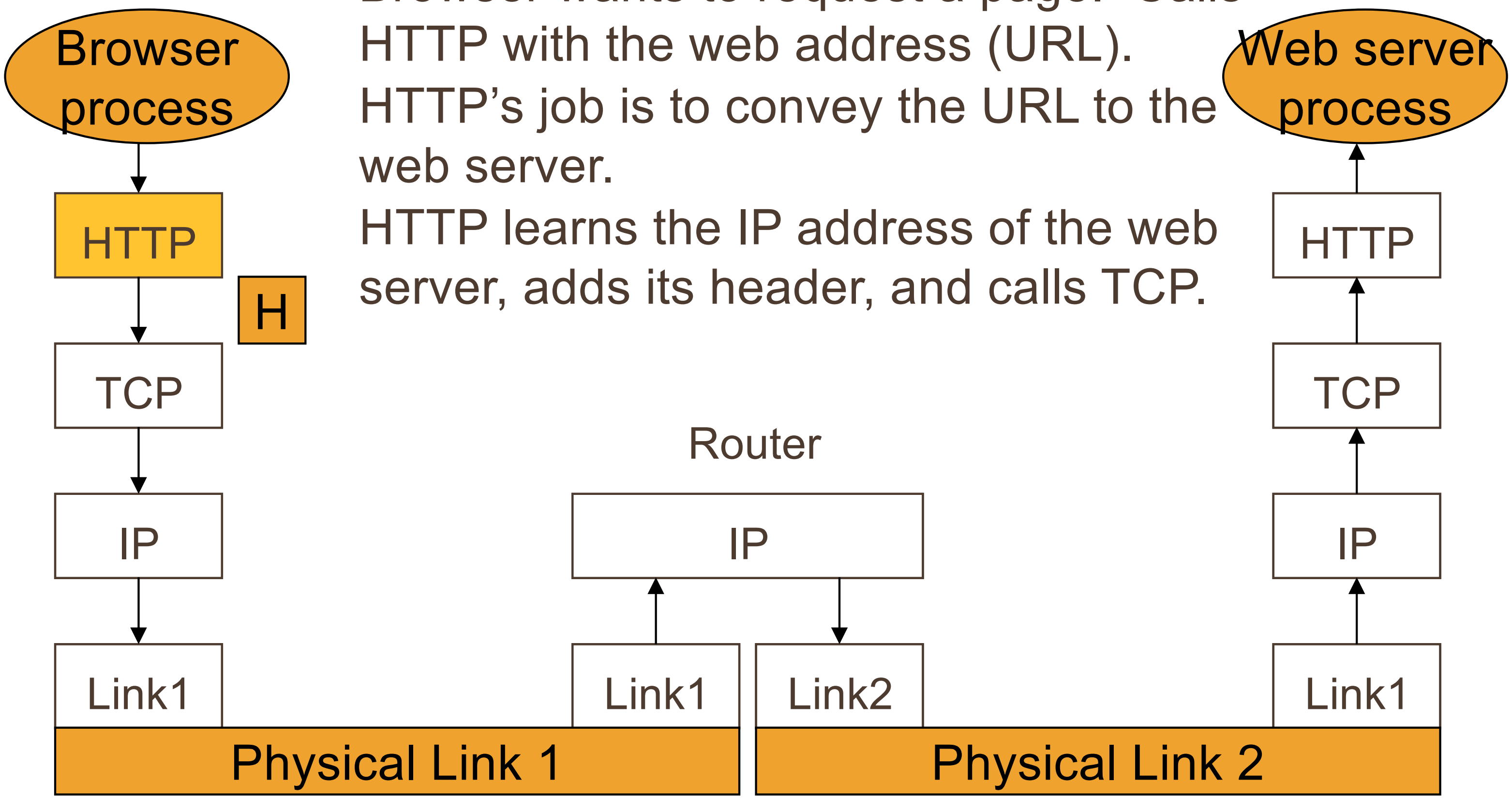
- Communications stack consists of a set of services
  - Each providing a service to the layer above
  - Using services of the layer below
  - Each service has a programming API
- Each service has to convey information for one or more peers across the network
- This information is contained in a header

Protocol layering example



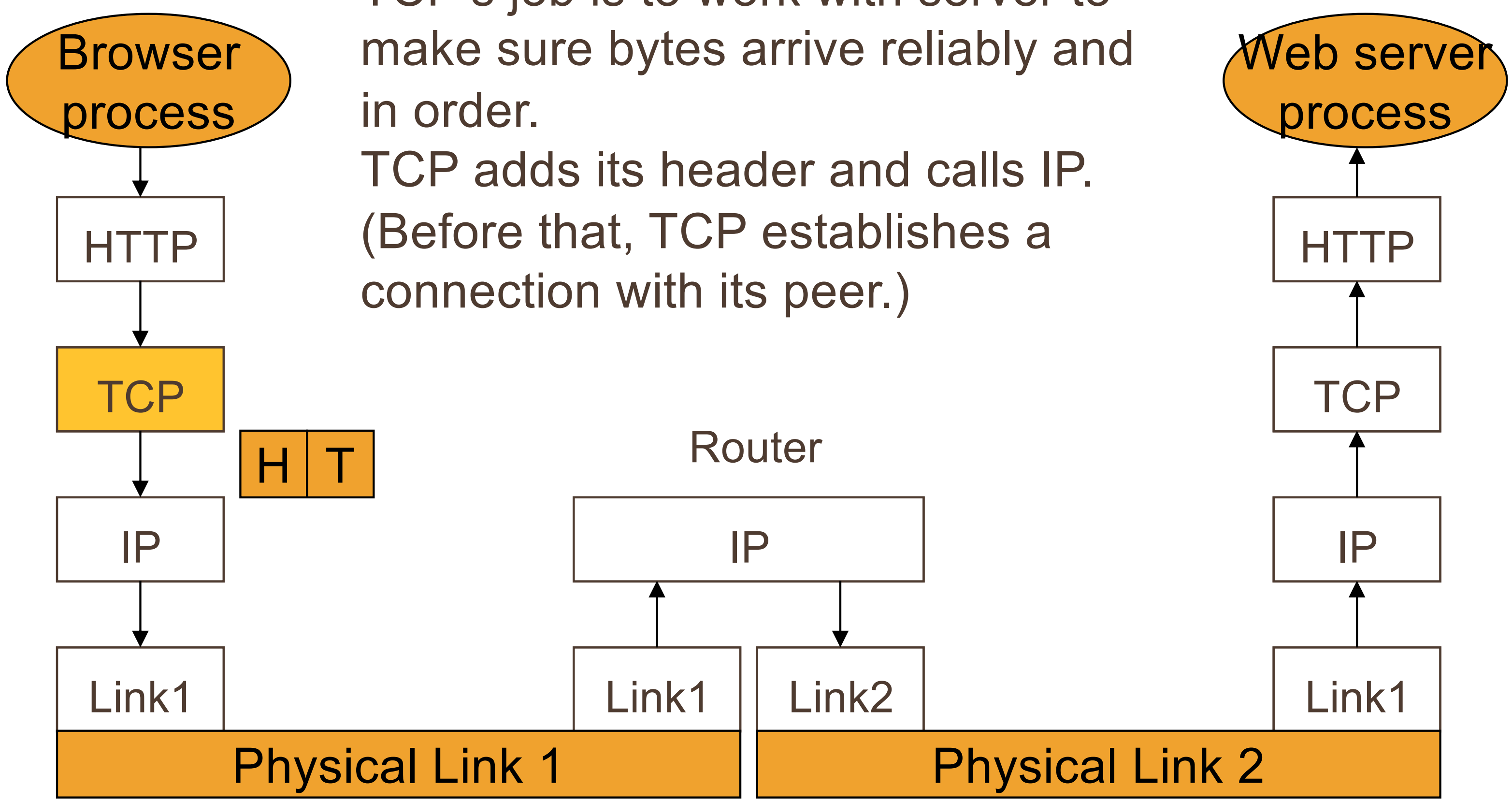
Protocol layering example

Browser wants to request a page. Calls HTTP with the web address (URL). HTTP's job is to convey the URL to the web server. HTTP learns the IP address of the web server, adds its header, and calls TCP.



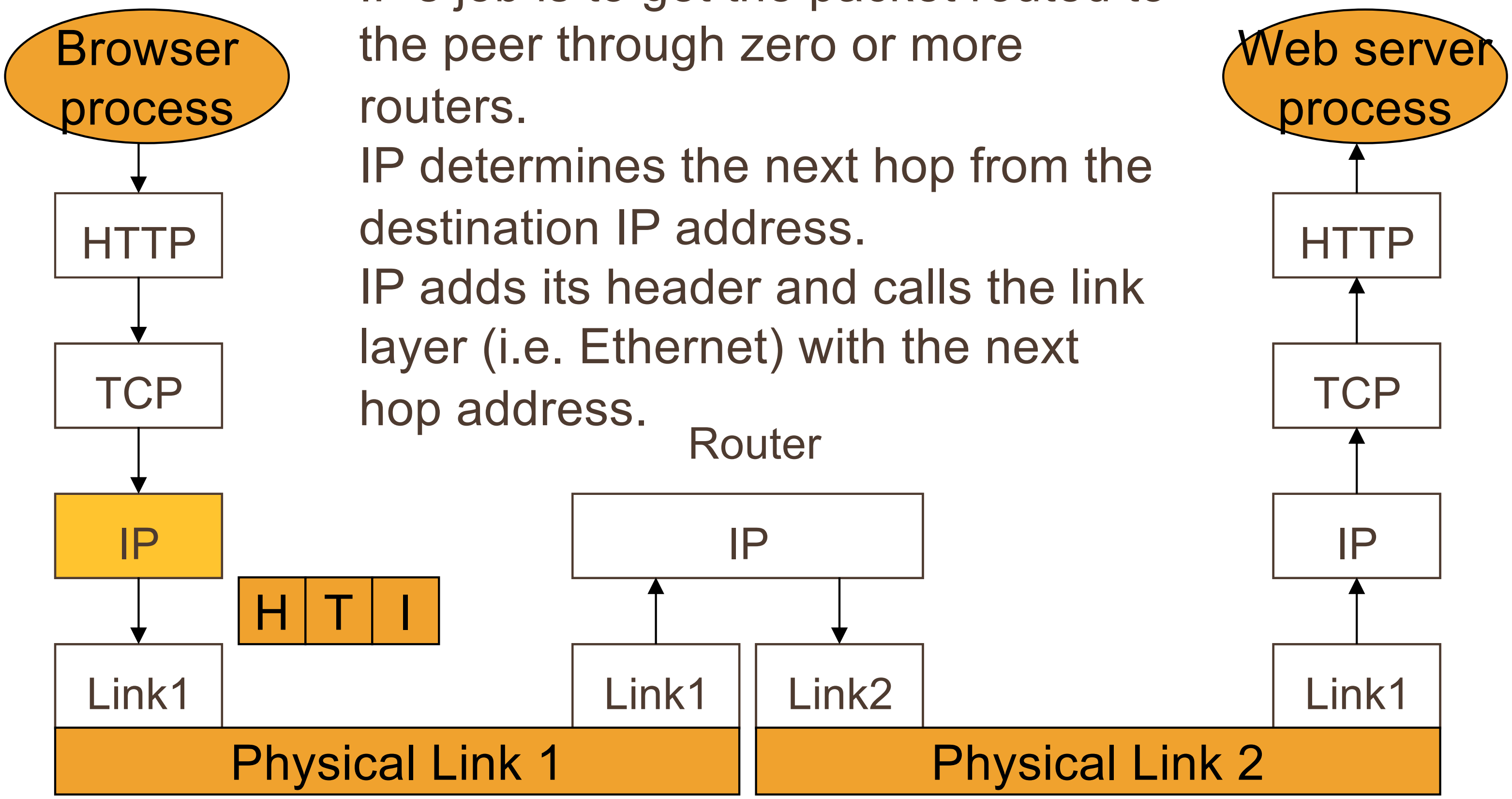
Protocol layering example

TCP's job is to work with server to make sure bytes arrive reliably and in order.
TCP adds its header and calls IP.
(Before that, TCP establishes a connection with its peer.)



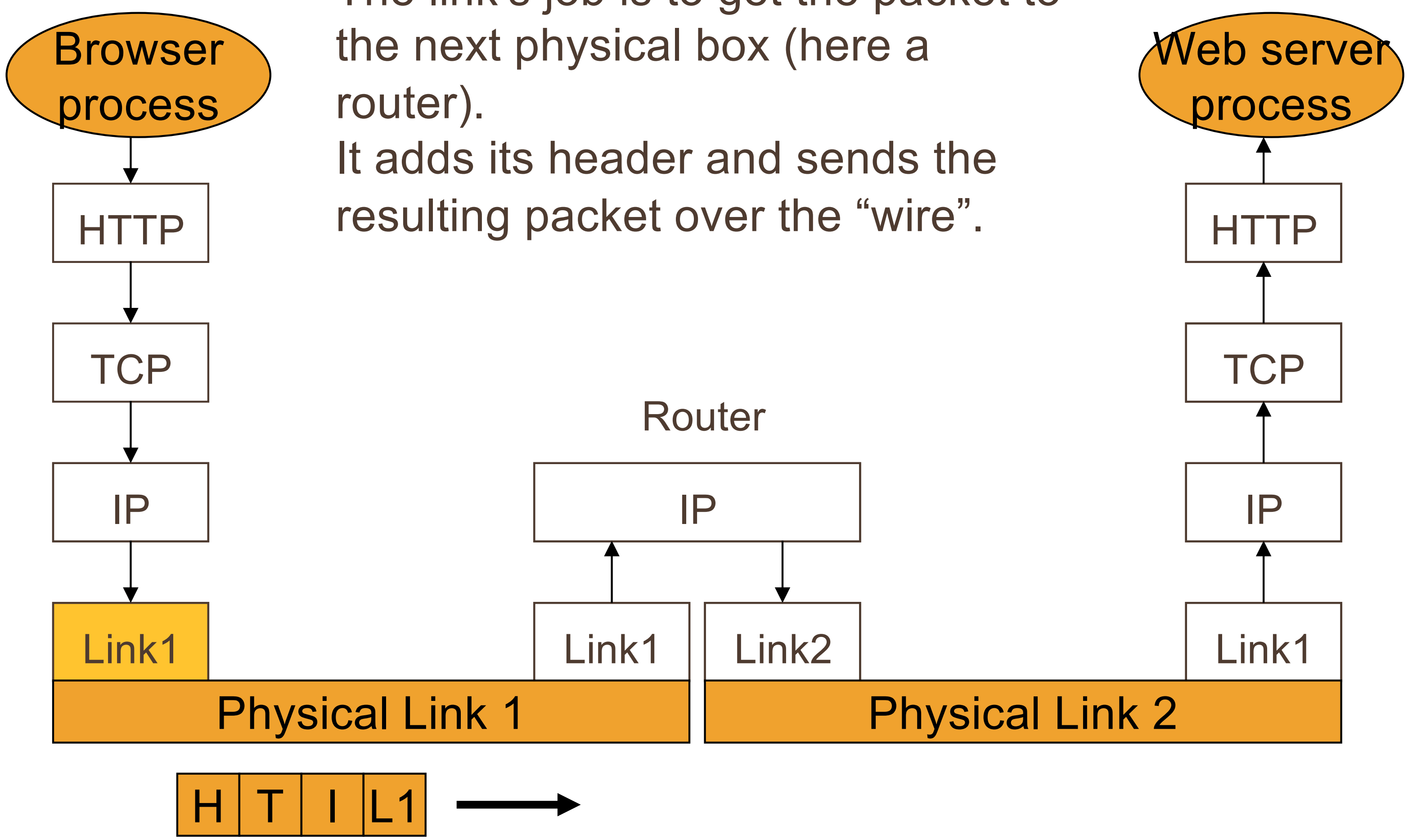
Protocol layering example

IP's job is to get the packet routed to the peer through zero or more routers.
IP determines the next hop from the destination IP address.
IP adds its header and calls the link layer (i.e. Ethernet) with the next hop address.



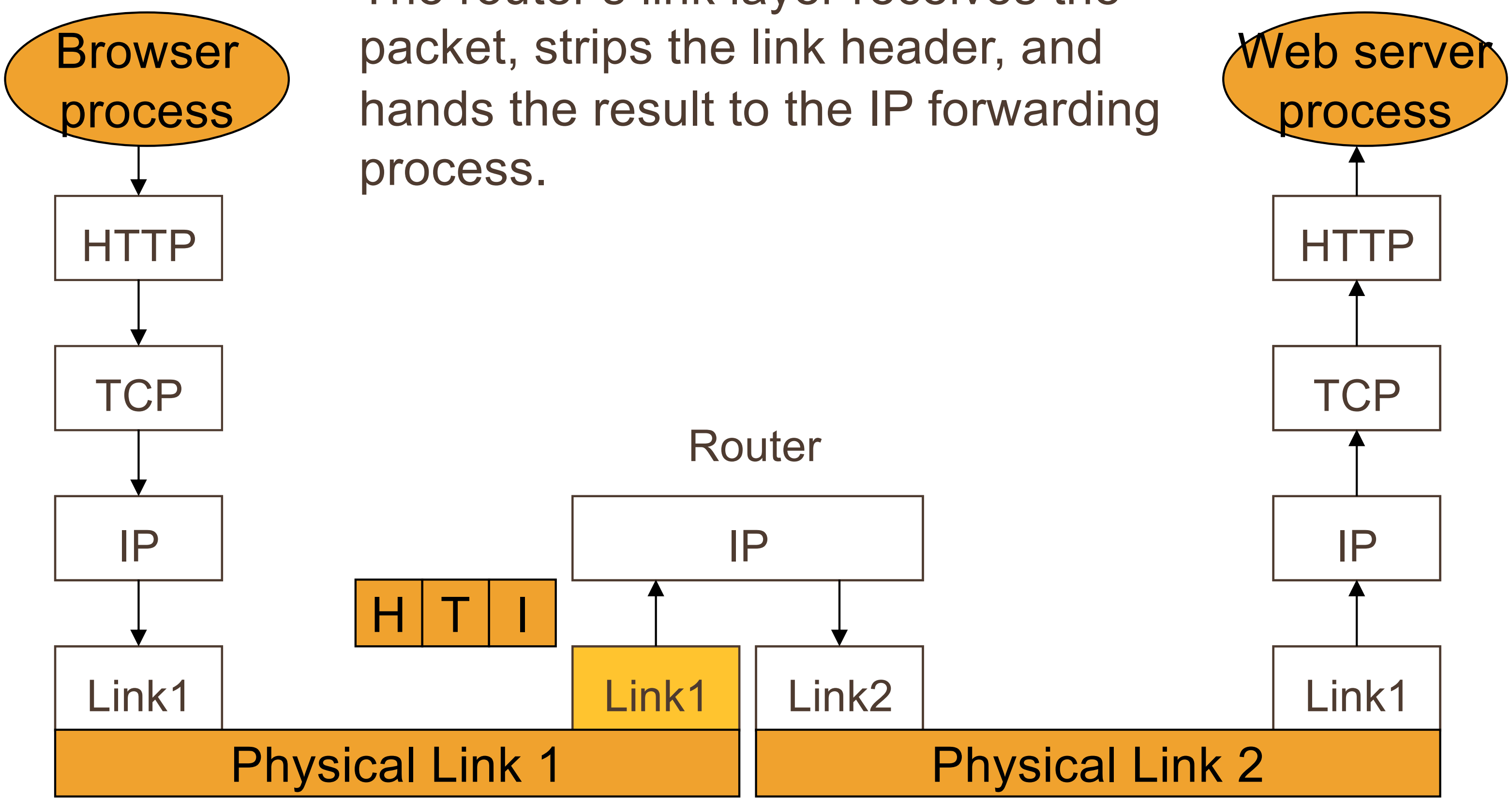
Protocol layering example

The link's job is to get the packet to the next physical box (here a router).
It adds its header and sends the resulting packet over the "wire".



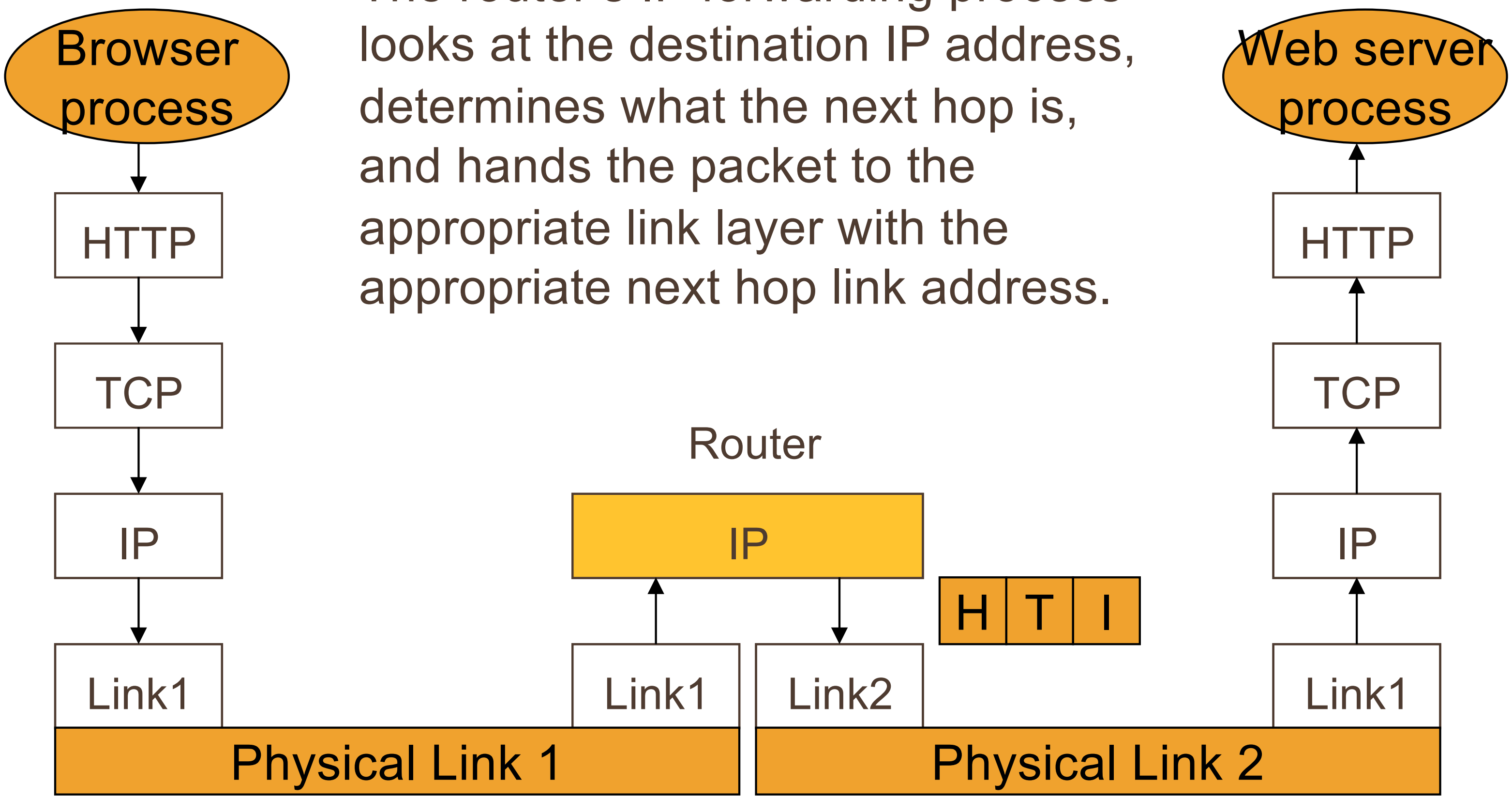
Protocol layering example

The router's link layer receives the packet, strips the link header, and hands the result to the IP forwarding process.



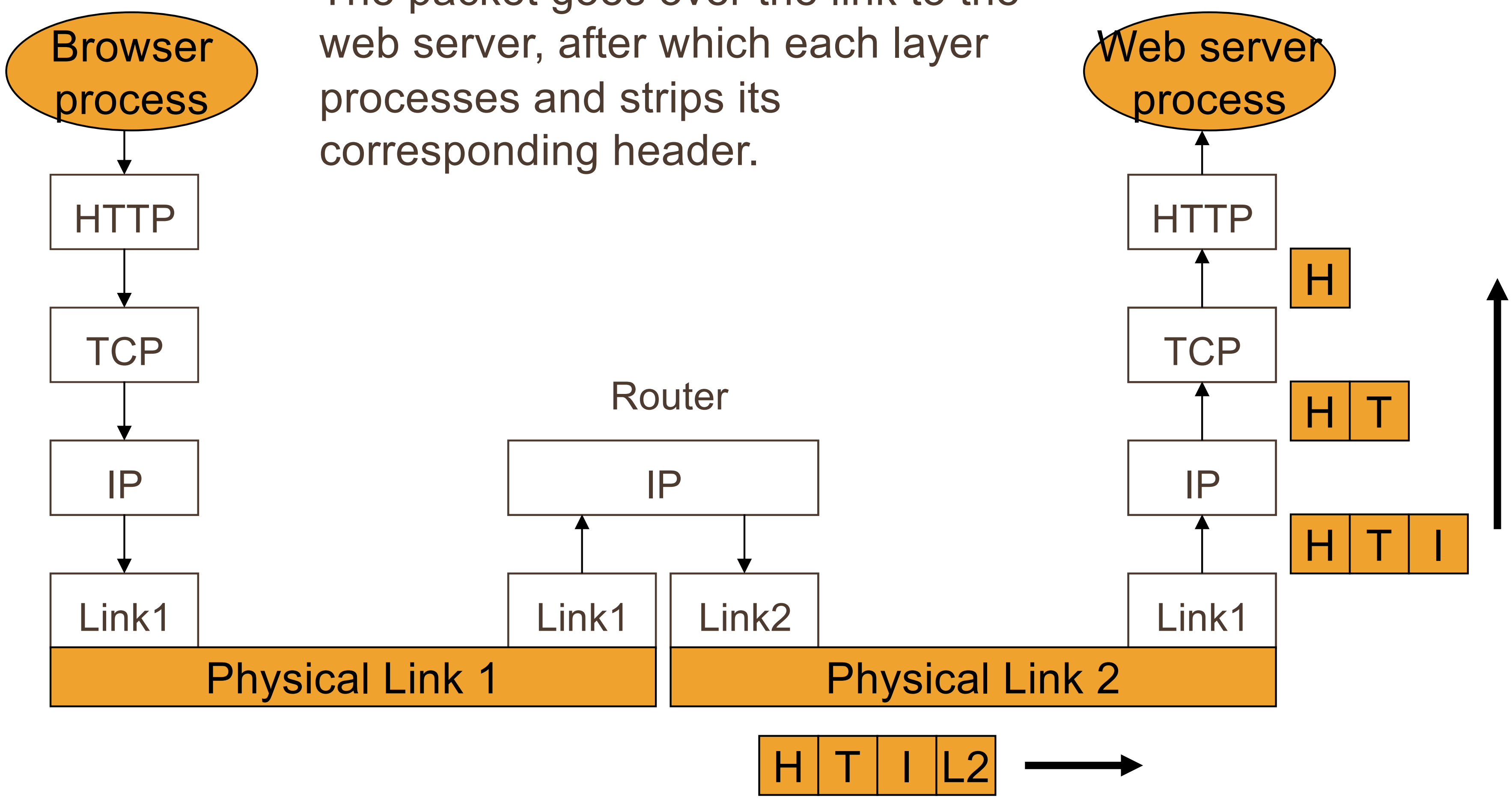
Protocol layering example

The router's IP forwarding process looks at the destination IP address, determines what the next hop is, and hands the packet to the appropriate link layer with the appropriate next hop link address.



Protocol layering example

The packet goes over the link to the web server, after which each layer processes and strips its corresponding header.



TCP/IP PROTOCOLS

Basic elements of any protocol header

- Demuxing field
  - Indicates which is the next higher layer (or process, or context, etc.)
- Length field or header delimiter
  - For the header, optionally for the whole packet
- Header format may be text (HTTP, SMTP (email)) or binary (IP, TCP, Ethernet)

Demuxing fields

- Ethernet: Protocol Number
  - Indicates IPv4, IPv6, (old: Appletalk, SNA, Decnet, etc.)
- IP: Protocol Number
  - Indicates TCP, UDP, SCTP
- TCP and UDP: Port Number
  - Well known ports indicate FTP, SMTP, HTTP, SIP, many others
  - Dynamically negotiated ports indicate specific processes (for these and other protocols)
- HTTP: Host field
  - Indicates "virtual web server" within a physical web server
  - (Well, more like an identifier than a demuxing field)

IP (Internet Protocol)

- Three services:
  - Unicast: transmits a packet to a specific host
  - Multicast: transmits a packet to a group of hosts
  - Anycast: transmits a packet to one of a group of hosts (typically nearest)
- Destination and source identified by the IP address (32 bits for IPv4, 128 bits for IPv6)
- All services are unreliable
  - Packet may be dropped, duplicated, and received in a different order

IP address

- Both source and destination address may be modified in transit
  - By NAT boxes
  - But even so, can reply to the source IP address
  - Unless source address is spoofed
- IP (unicast) address is hierarchical, but host can treat it as a flat identifier (given net mask)
  - Can't tell how close or far a host is by looking at its IP address

IP(v4) address format

- In binary, a 32-bit integer
- In text, this: “155.246.89.22”
  - Each decimal digit represents 8 bits (0 – 255)
- a.b.c.d/n
  - Last n bits identify machine
  - First 32-n bits identify network on the Internet
  - Ex: <u>155.246.89</u>.0/8 for most Stevens hosts

IP(v4) address format

- “Private” addresses are not globally unique:
  - Used behind NAT boxes
  - 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
- Multicast addresses start with 1110 as the first 4 bits (Class D address)
  - 224.0.0.0/4
- Unicast and anycast addresses come from the same space

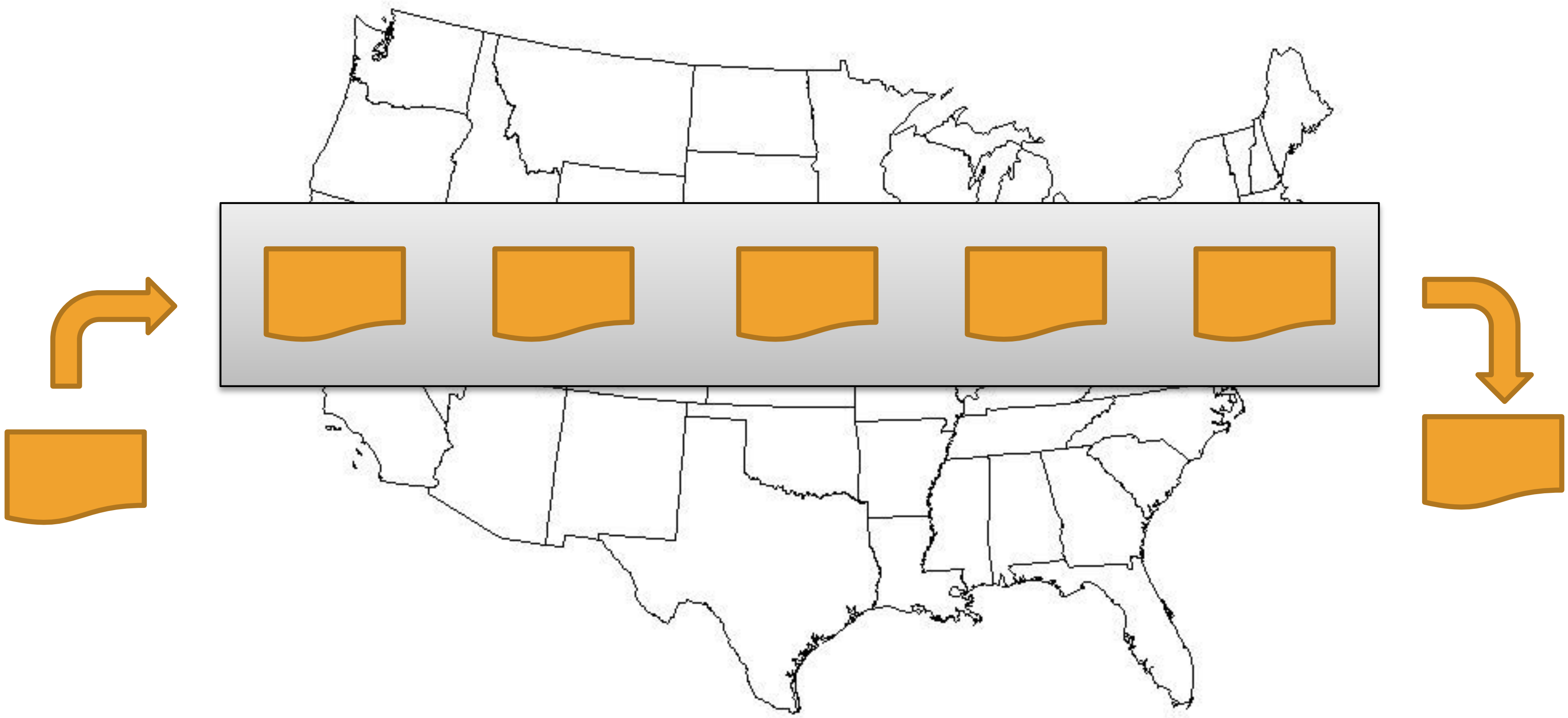
UDP (User Datagram Protocol)

- Runs above IP
- Same unreliable service as IP
  - Packets can get lost anywhere:
    - Outgoing buffer at source
    - Router or link
    - Incoming buffer at destination
- But adds port numbers
  - Mailboxes
  - Used to identify “application layer” processes
- Also a checksum, optional

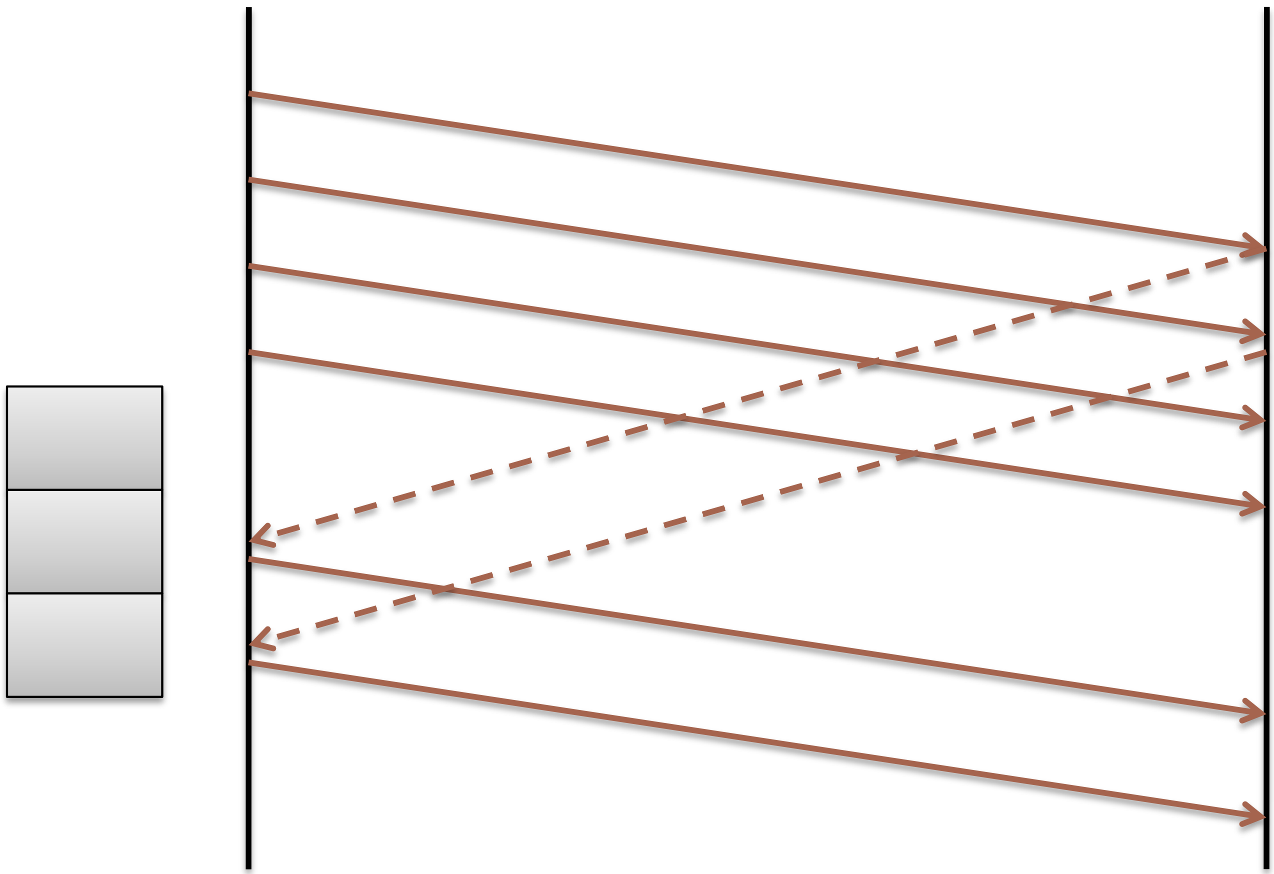
TCP (Transmission Control Protocol)

- Runs above IP
  - Port number and checksum like UDP
- Service is in-order byte stream
  - Bytes transparently packaged in packets
- Flow control and congestion control
- Connection setup and teardown phases
- Can be considerable delay between bytes in at source and bytes out at destination
  - Because of timeouts and retransmissions
- Works only with unicast (not multicast or anycast)

TCP Pipeline



TCP Windowing



UDP vs. TCP

- UDP is more real-time
  - Packet is sent or dropped, but is not delayed
- UDP has more of a "message" flavor
  - One packet = one message
  - But must add reliability mechanisms over it
- TCP good for transferring a file
  - Frustrating for messaging
  - Interrupts to application don't conform to message boundaries
  - No "Application Layer Framing"
- TCP is vulnerable to DoS (Denial of Service) attacks
  - SYN flood

SCTP (Stream Control Transmission Protocol)

- IETF standard
- Overcomes many limitations of TCP
  - Motivation is SS7 signaling over IP
- Message oriented---supports message framing
- Multiple streams for a given session
  - Interruption in one stream does not effect others
- Cookie mechanism for DoS attacks

Summary

- TCP, UDP, IP provide a nice set of basic tools
  - Key is to understand concept of protocol layering
- But problems/limitations exist
  - IP has been compromised by NAT, can't be used as a stable identifier
  - Firewalls can block communications
  - TCP has vulnerabilities
  - Network performance highly variable