

Networking Overview

CS 161: Computer Security

Prof. Vern Paxson

**TAs: Jethro Beekman, Mobin Javed,
Antonio Lupher, Paul Pearce
& Matthias Vallentin**

<http://inst.eecs.berkeley.edu/~cs161/>

February 5, 2013

Focus For Today's Lecture

- Sufficient background in networking to then explore security issues in next ~3 lectures
 - Networking = the **Internet**
- Complex topic with many facets
 - We will omit concepts/details that aren't very security-relevant
 - We'll mainly look at **IP**, **TCP**, **DNS** (and later **DHCP**)
- Networking is full of **abstractions**
 - Goal is for you to develop apt *mental models* / analogies
 - ASK questions when things are unclear
 - o (but we may skip if not ultimately relevant for security, or postpone if question itself is directly about security)

Key Concept #1: *Protocols*

- A protocol is an **agreement on how to communicate**
- Includes **syntax** and **semantics**
 - How a communication is specified & structured
 - o Format, order messages are sent and received
 - What a communication means
 - o Actions taken when transmitting, receiving, or timer expires
- E.g.: making a comment in lecture?
 1. Raise your hand.
 2. Wait to be called on.
 3. Or: wait for speaker to **pause** and vocalize
 4. If unrecognized (after **timeout**): vocalize w/ “excuse me”₃

Key Concept #2: *Dumb Network*

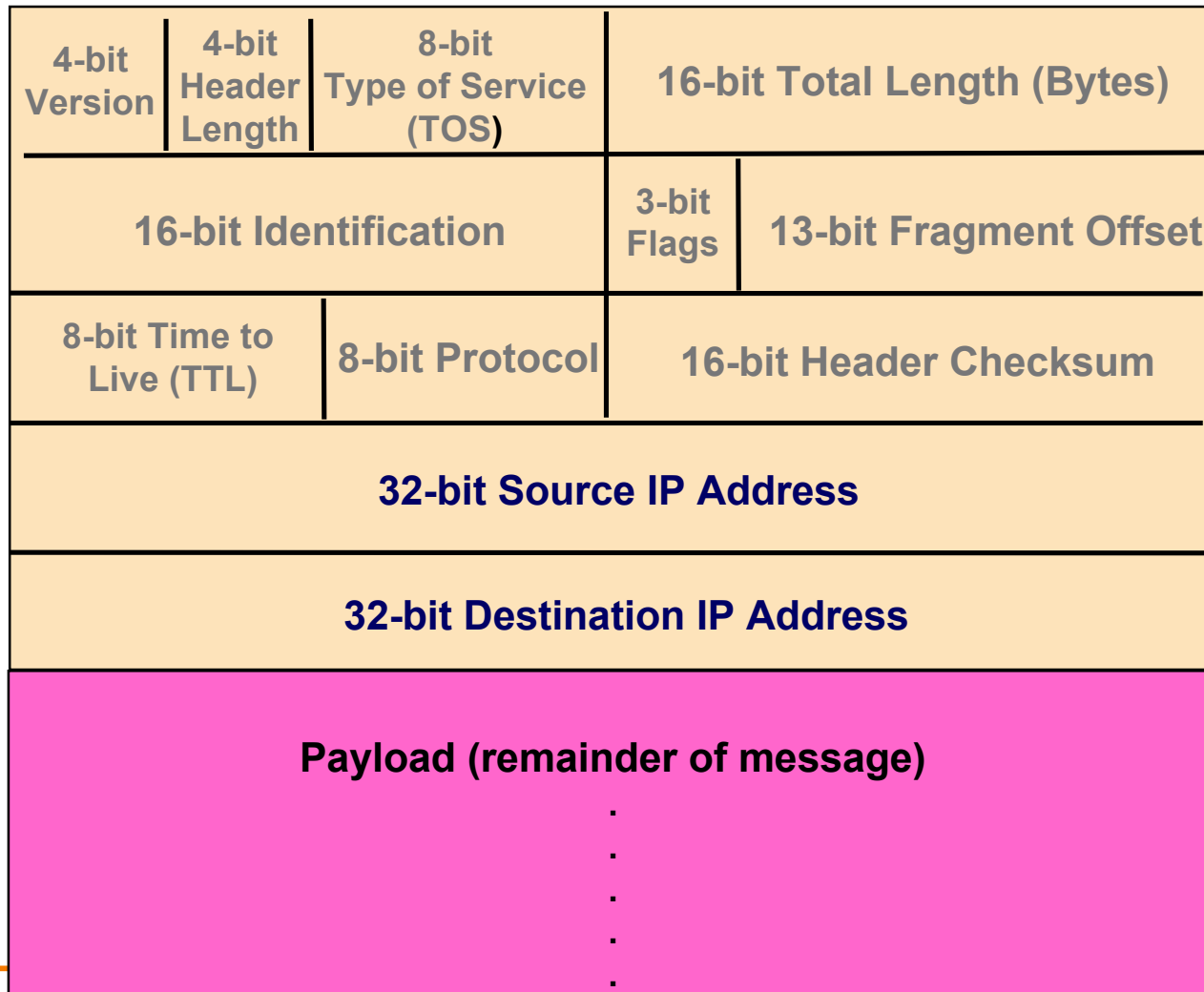
- Original Internet design: interior nodes (“**routers**”) have no knowledge* of ongoing connections going through them
- **Not:** how you picture the telephone system works
 - Which internally tracks all of the active voice calls
- Instead: the **postal system!**
 - Each Internet message (“**packet**”) self-contained

* Today’s Internet is full of hacks that violate this

Self-Contained IP Packet Format



IP = Internet *Protocol*



Header is like a letter envelope: contains all info needed for delivery

Key Concept #2: *Dumb Network*

- Original Internet design: interior nodes (“**routers**”) have no knowledge* of ongoing connections going through them
- **Not:** how you picture the telephone system works
 - Which internally tracks all of the active voice calls
- **Instead:** the **postal system!**
 - Each Internet message (“**packet**”) self-contained
 - Interior routers look at destination address to forward
 - If you want smarts, build it “**end-to-end**”, not “hop-by-hop”
 - Buys simplicity & robustness at the cost of shifting complexity into end systems

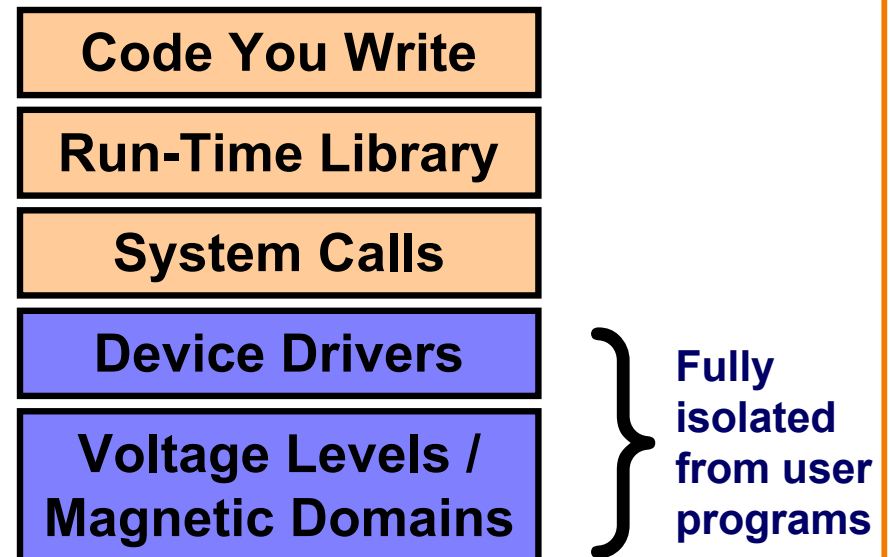
* Today’s Internet is full of hacks that violate this

Key Concept #3: *Layering*

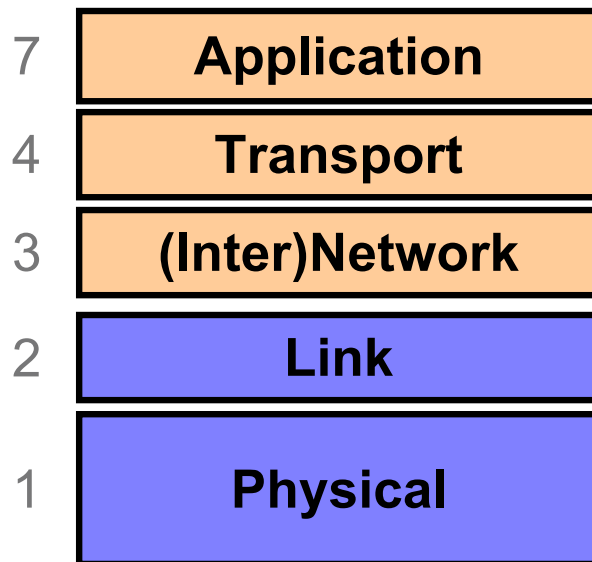
- Internet design is strongly partitioned into layers
 - Each layer relies on services provided by next layer below ...
 - ... and provides services to layer above it

- Analogy:

- Consider structure of an application you've written and the "services" each layer relies on / provides



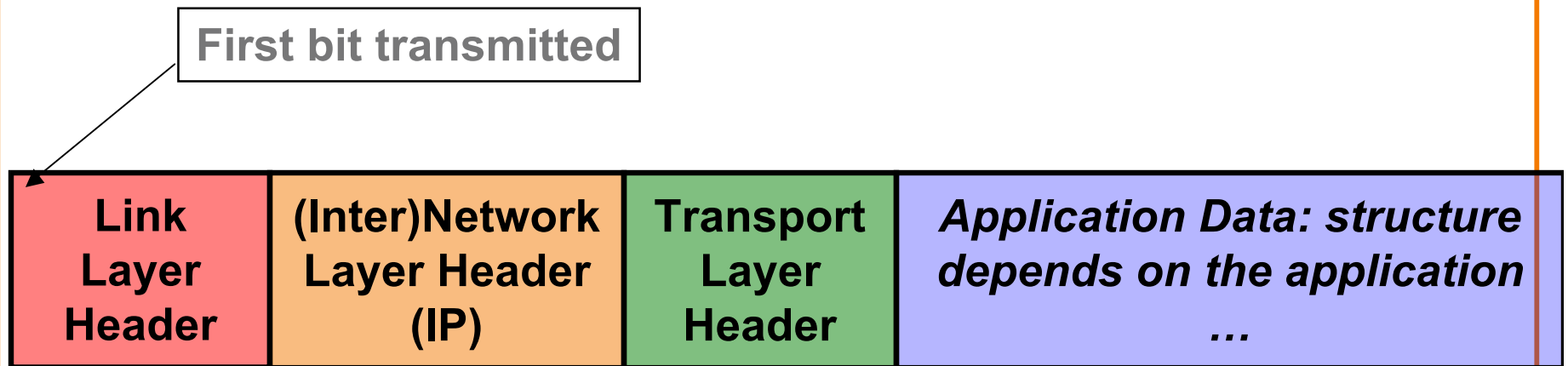
Internet Layering (“Protocol Stack”)



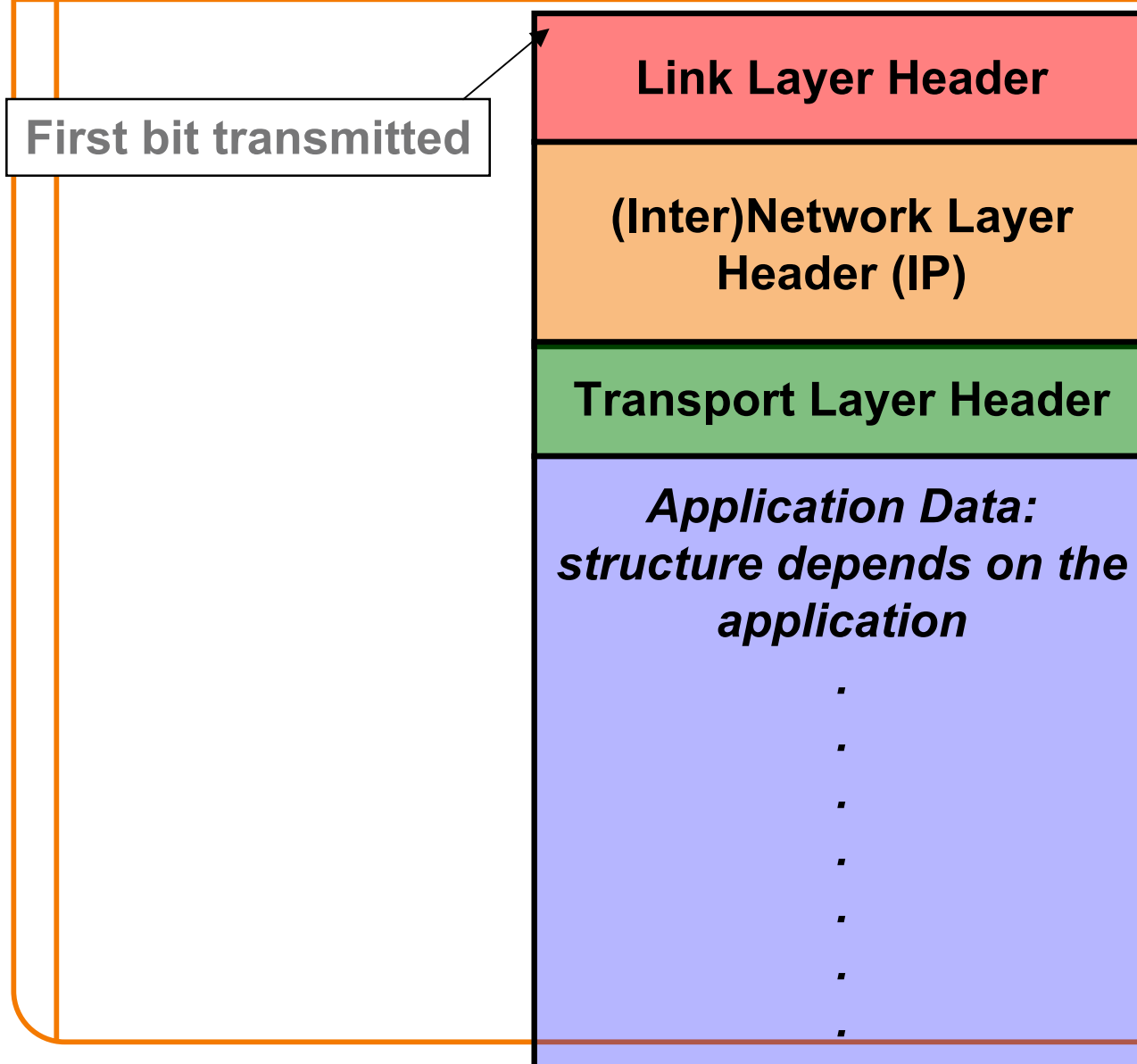
Note on a point of potential confusion: these diagrams are always drawn with lower layers **below** higher layers ...

But diagrams showing the layouts of packets are often the *opposite*, with the lower layers at the **top** since their headers precede those for higher layers

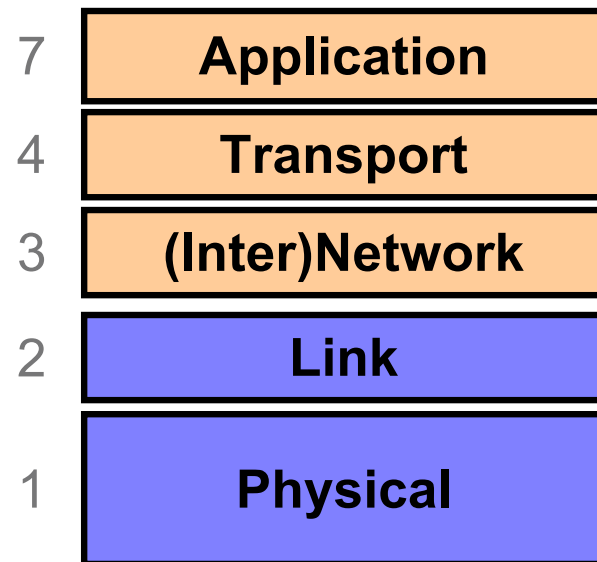
Horizontal View of a Single Packet



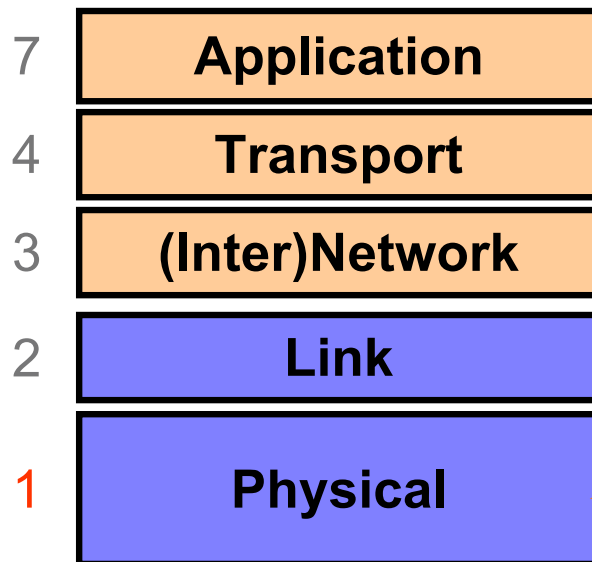
Vertical View of a Single Packet



Internet Layering (“Protocol Stack”)

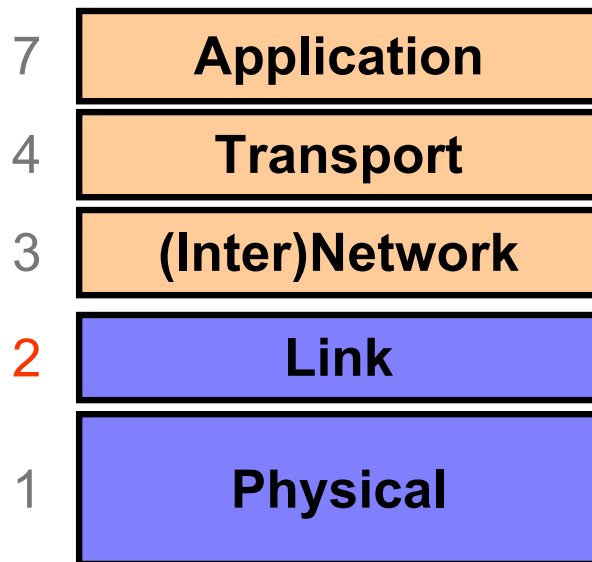


Layer 1: Physical Layer



Encoding **bits** to send them over a single physical link
e.g. patterns of
*voltage levels /
photon intensities /
RF modulation*

Layer 2: Link Layer

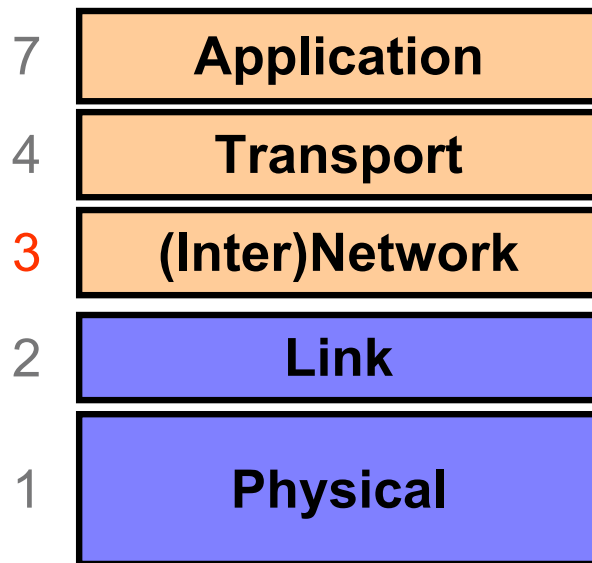


Framing and transmission of a collection of bits into individual **messages** sent across a single “subnetwork” (one physical technology)

Might involve multiple *physical links* (e.g., modern Ethernet)

Often technology supports **broadcast** transmission (**every** “node” connected to subnet receives)

Layer 3: (Inter)Network Layer (*IP*)



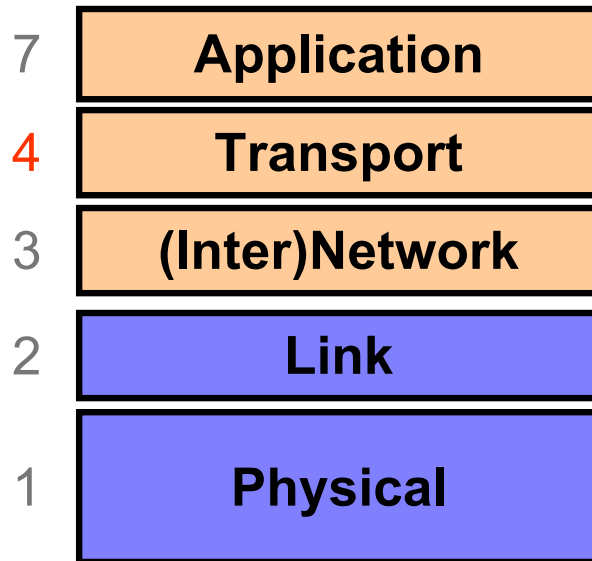
Bridges multiple “subnets” to provide *end-to-end* internet connectivity between nodes

- Provides global addressing

Works across different link technologies

Different for each Internet “hop”

Layer 4: Transport Layer

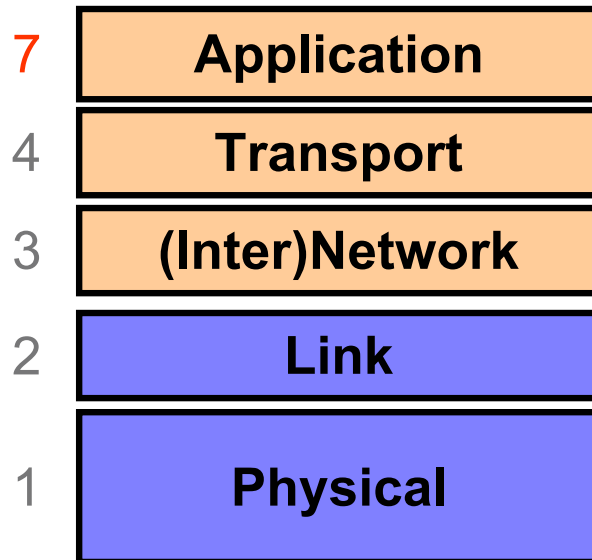


End-to-end communication between processes

Different services provided:
TCP = reliable *byte stream*
UDP = *unreliable datagrams*

(Datagram = single packet message)

Layer 7: Application Layer



Communication of whatever you wish

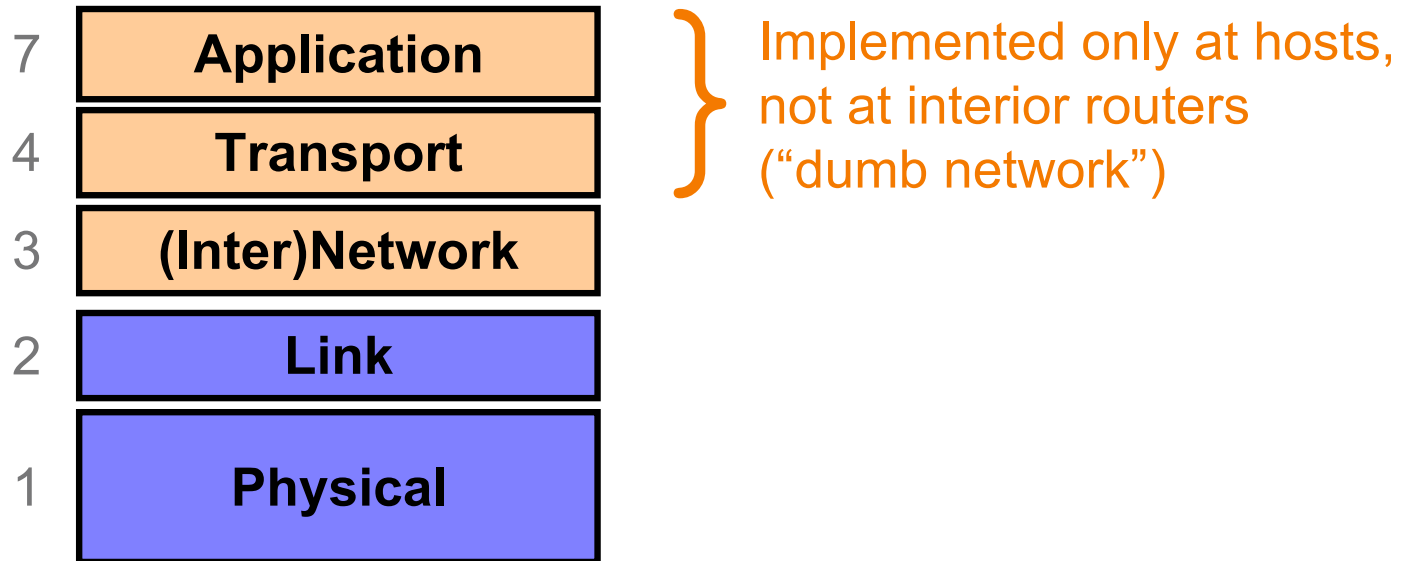
Can use whatever transport(s) is convenient

Freely structured

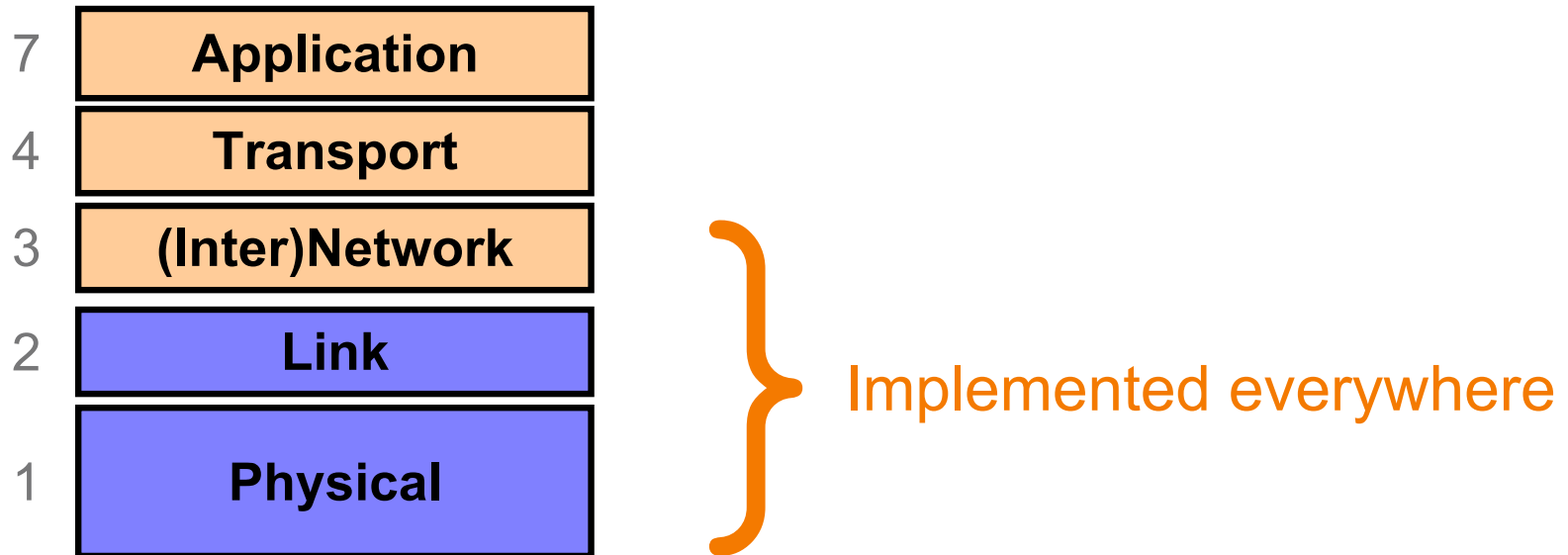
E.g.:

Skype, SMTP (email),
HTTP (Web), Halo, BitTorrent

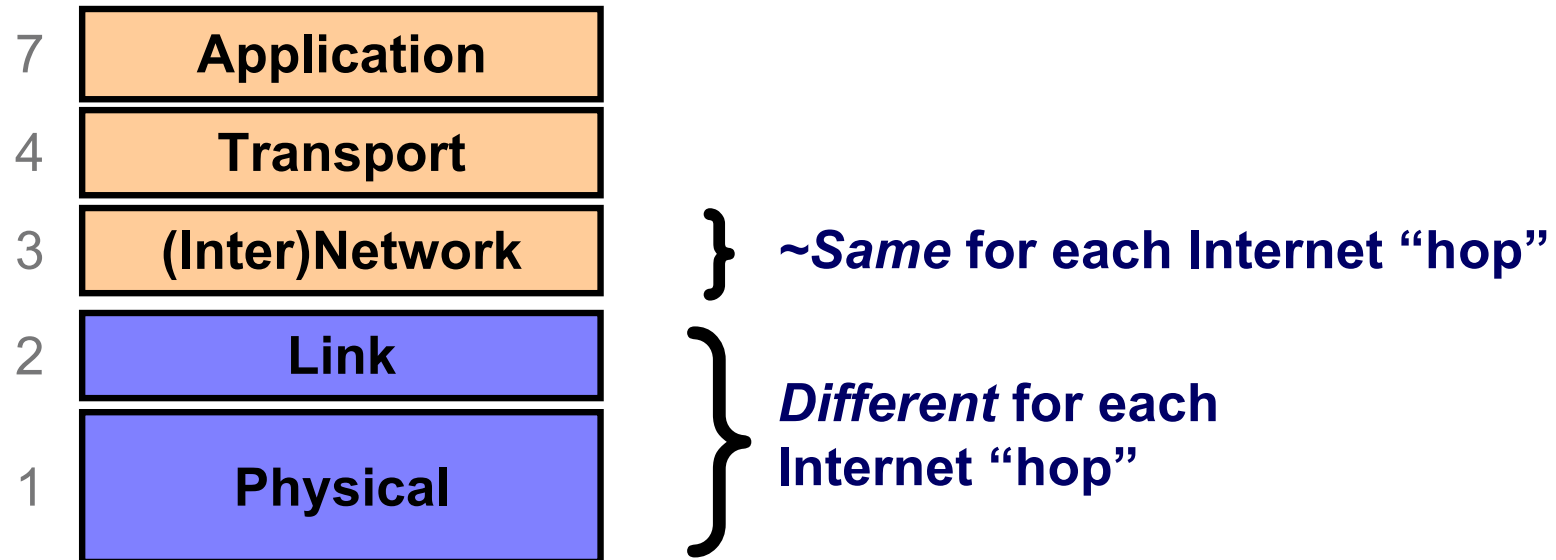
Internet Layering (“Protocol Stack”)



Internet Layering (“Protocol Stack”)

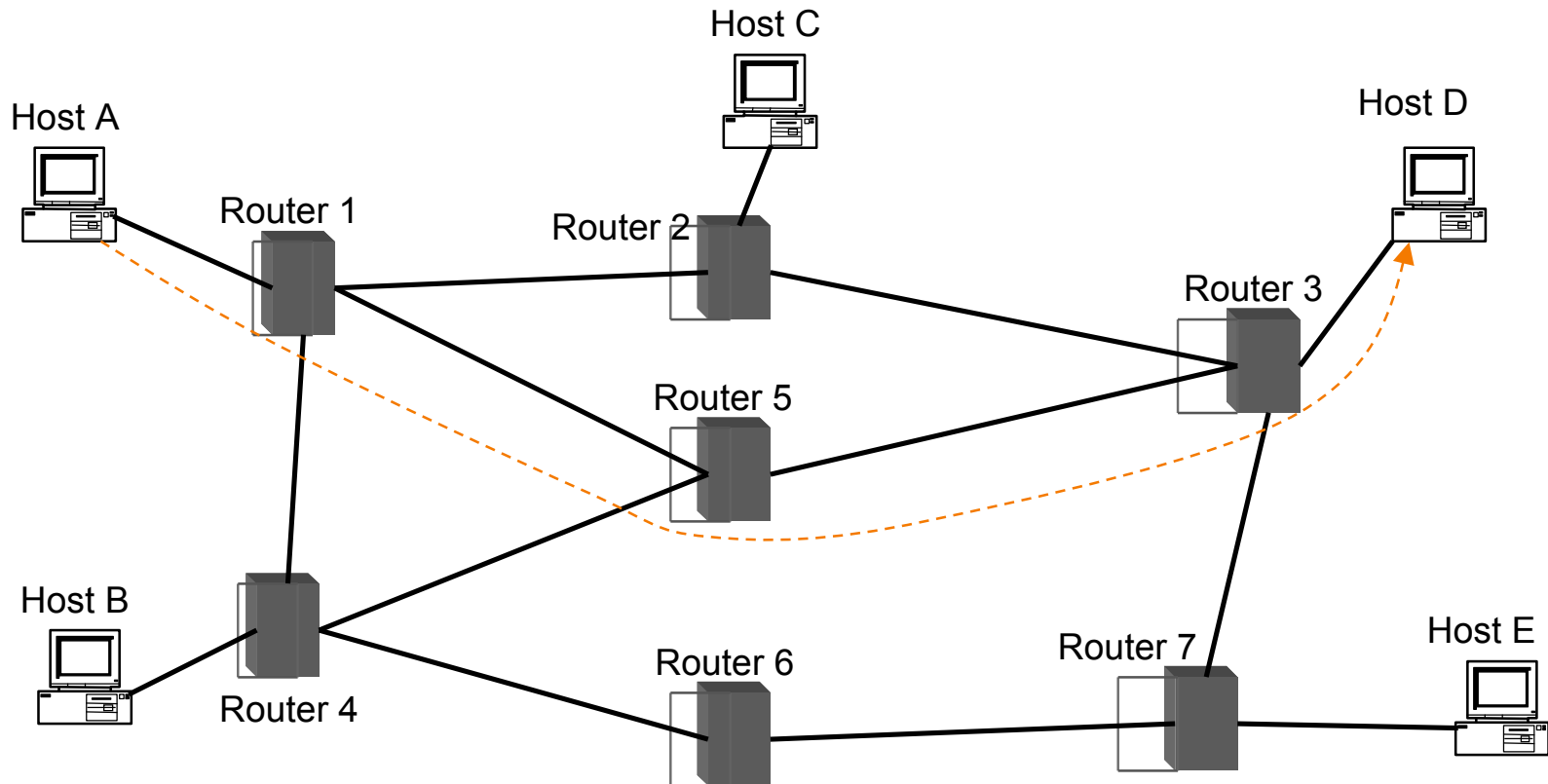


Internet Layering (“Protocol Stack”)



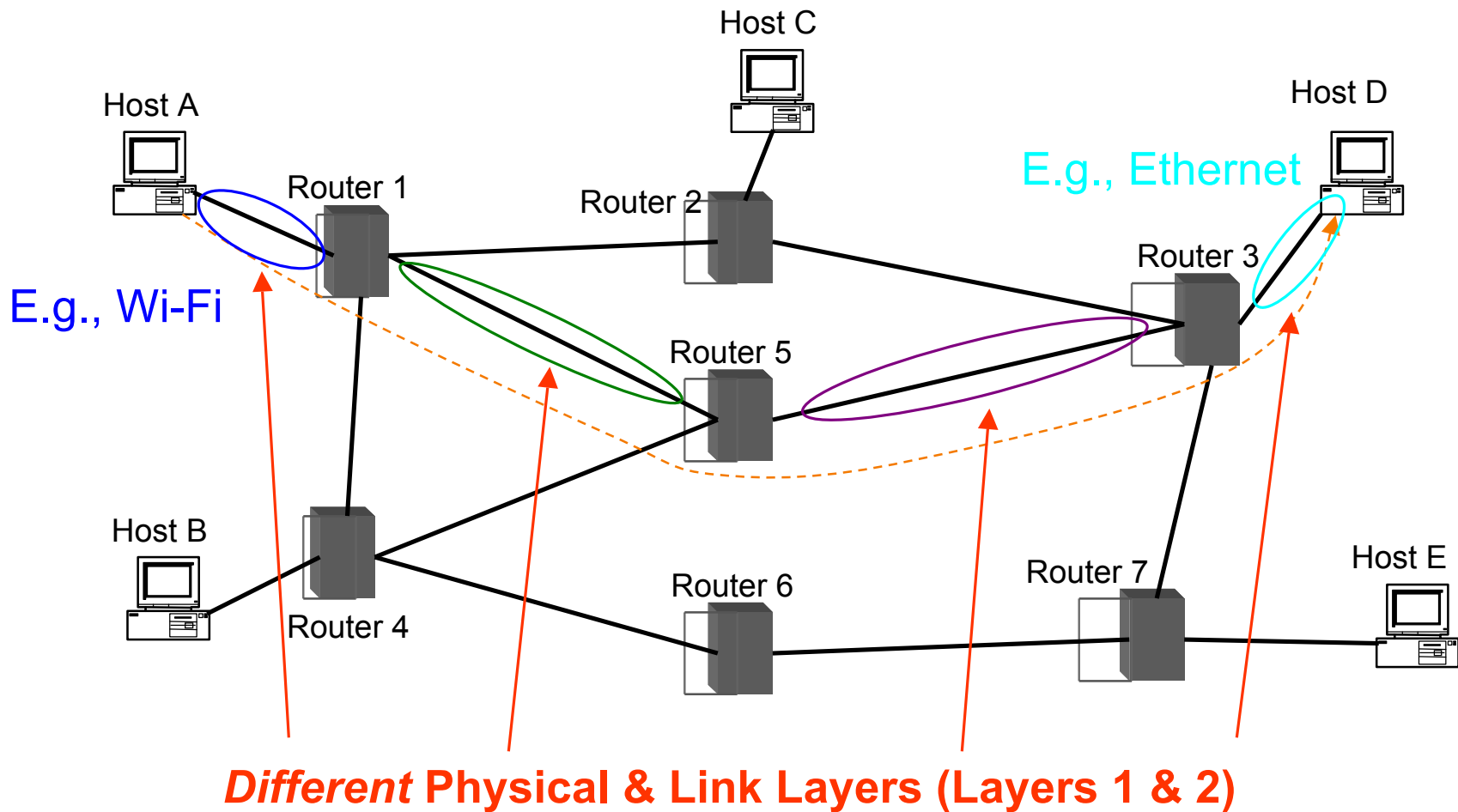
Hop-By-Hop vs. End-to-End Layers

Host A communicates with Host D



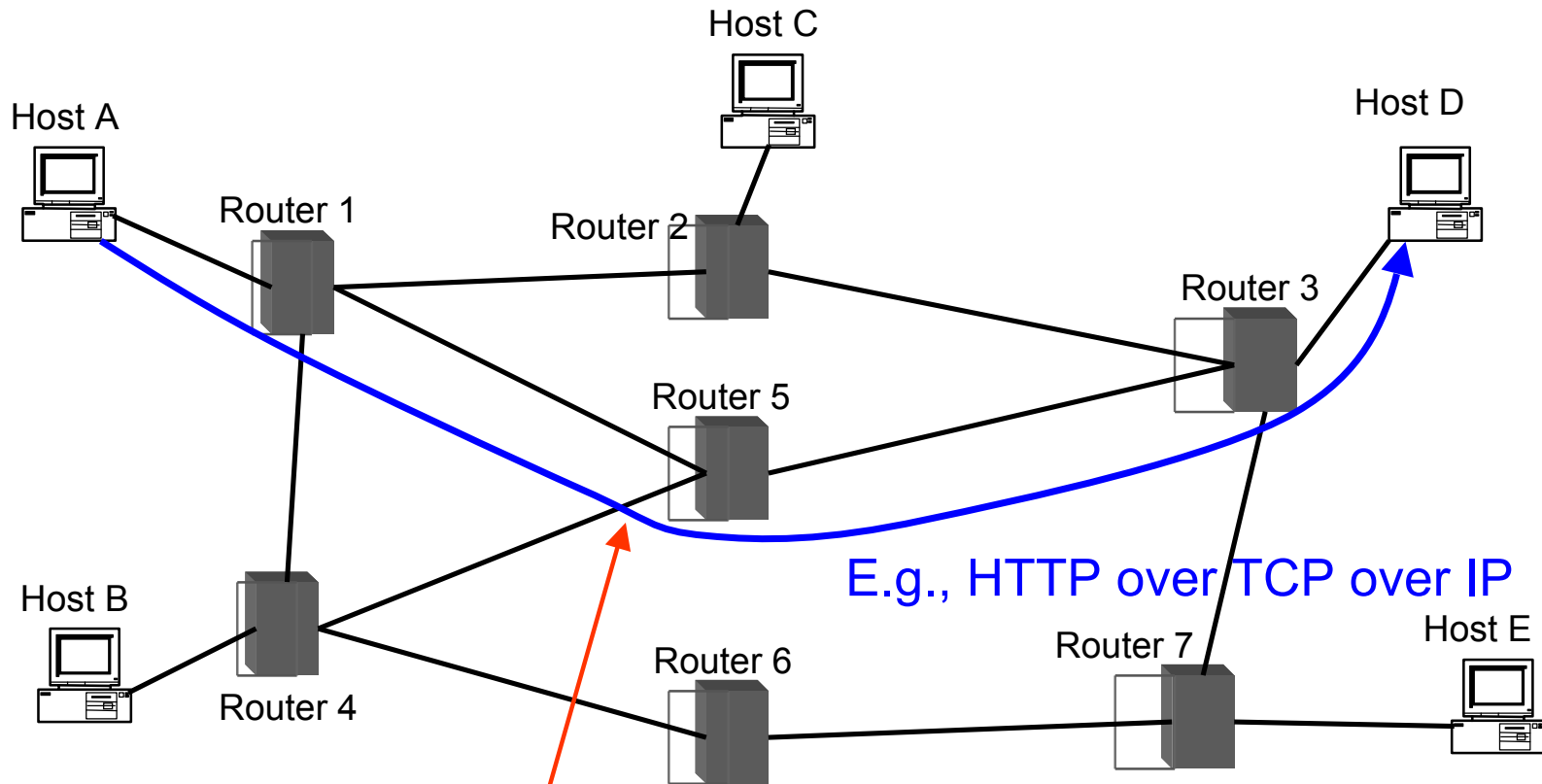
Hop-By-Hop vs. End-to-End Layers

Host A communicates with Host D



Hop-By-Hop vs. End-to-End Layers

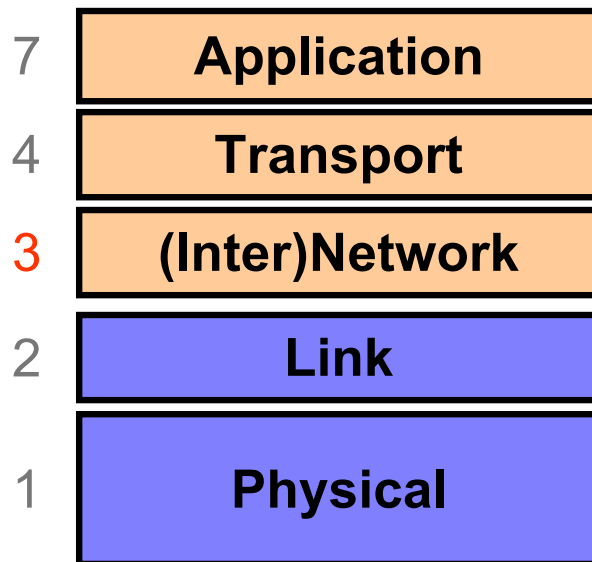
Host A communicates with Host D



E.g., HTTP over TCP over IP

Same Network / Transport / Application Layers (3/4/7)
(Routers **ignore** Transport & Application layers)

Layer 3: (Inter)Network Layer (*IP*)

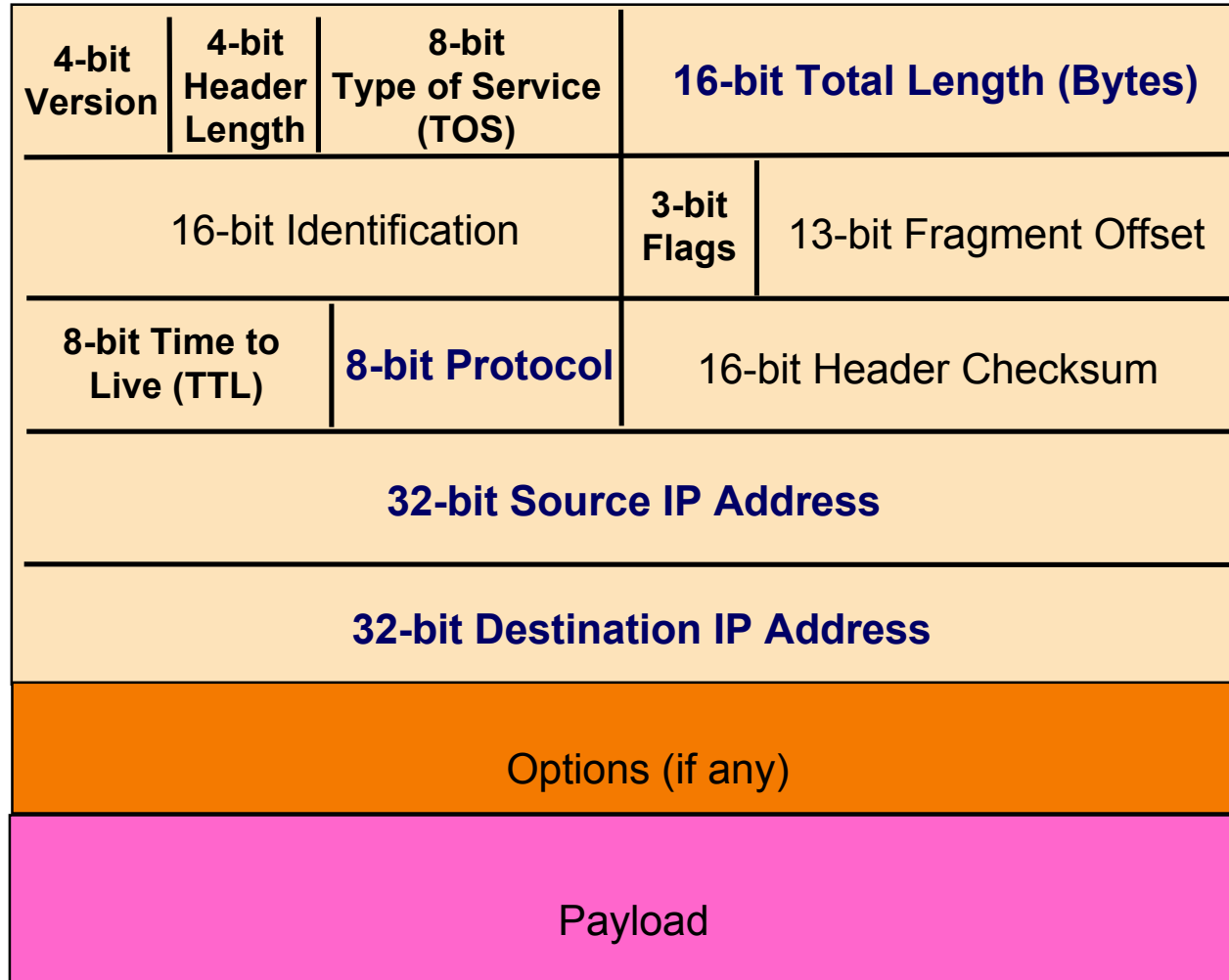


Bridges multiple “subnets” to provide *end-to-end* internet connectivity between nodes

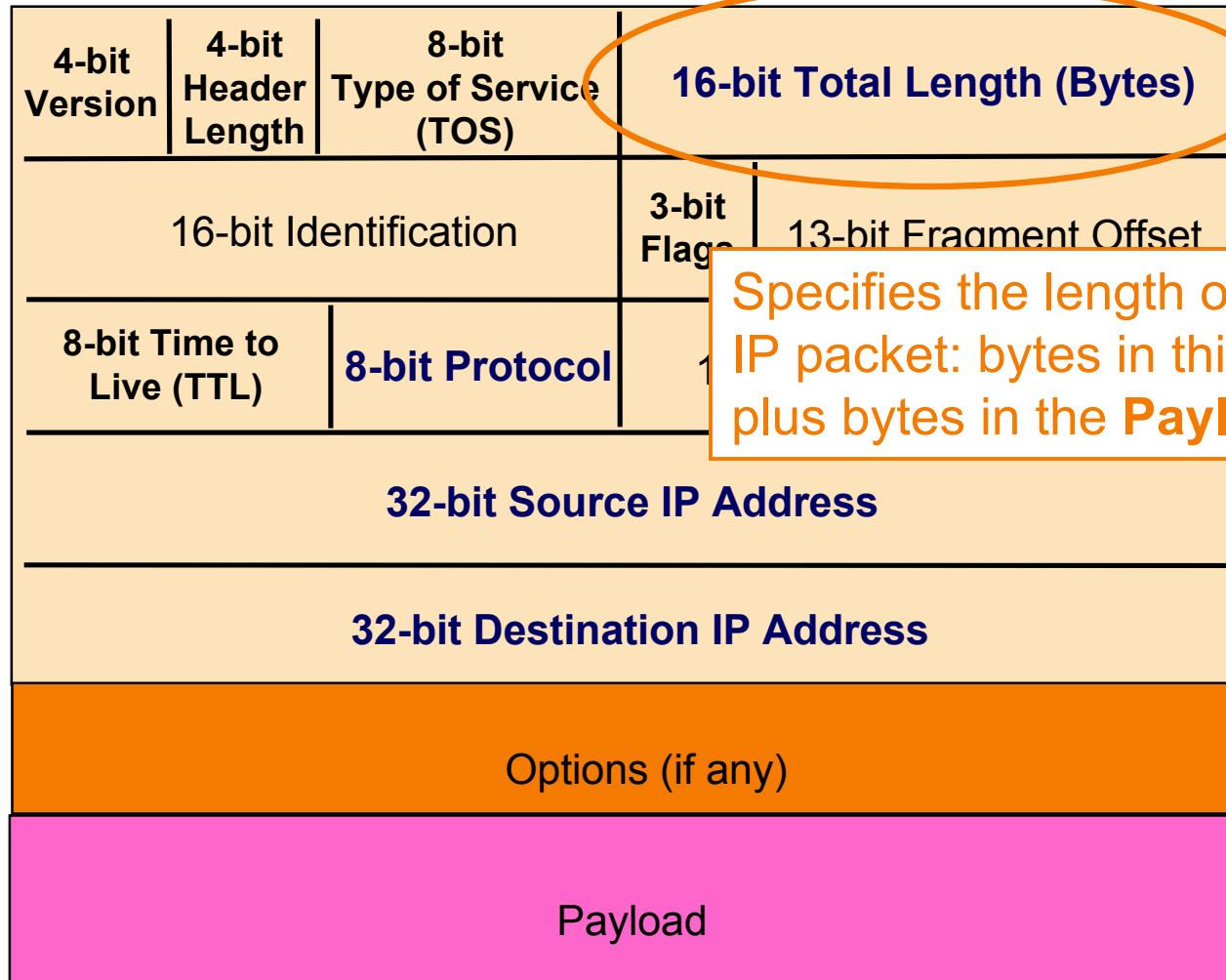
- Provides global addressing

Works across different link technologies

IP Packet Structure

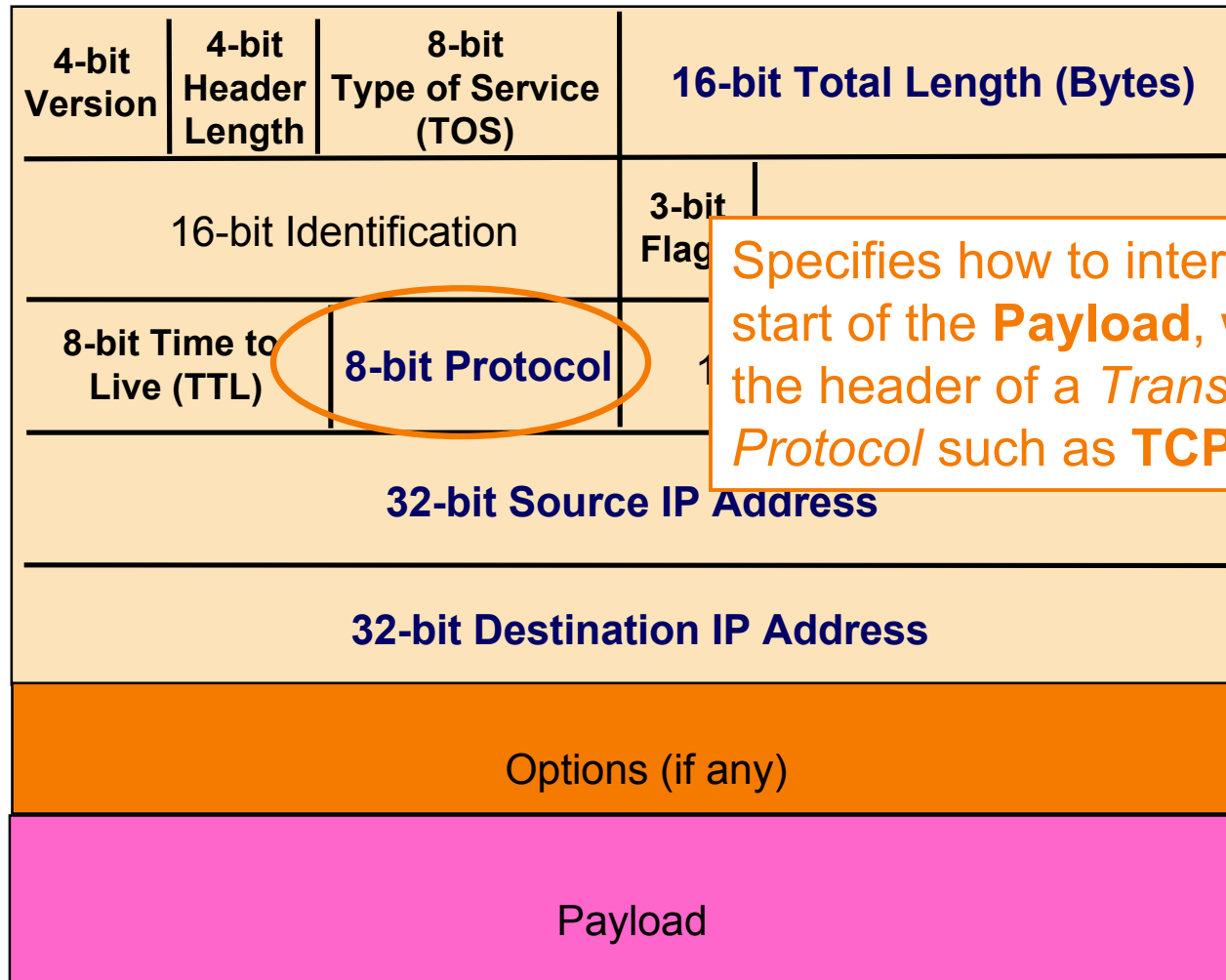


IP Packet Structure



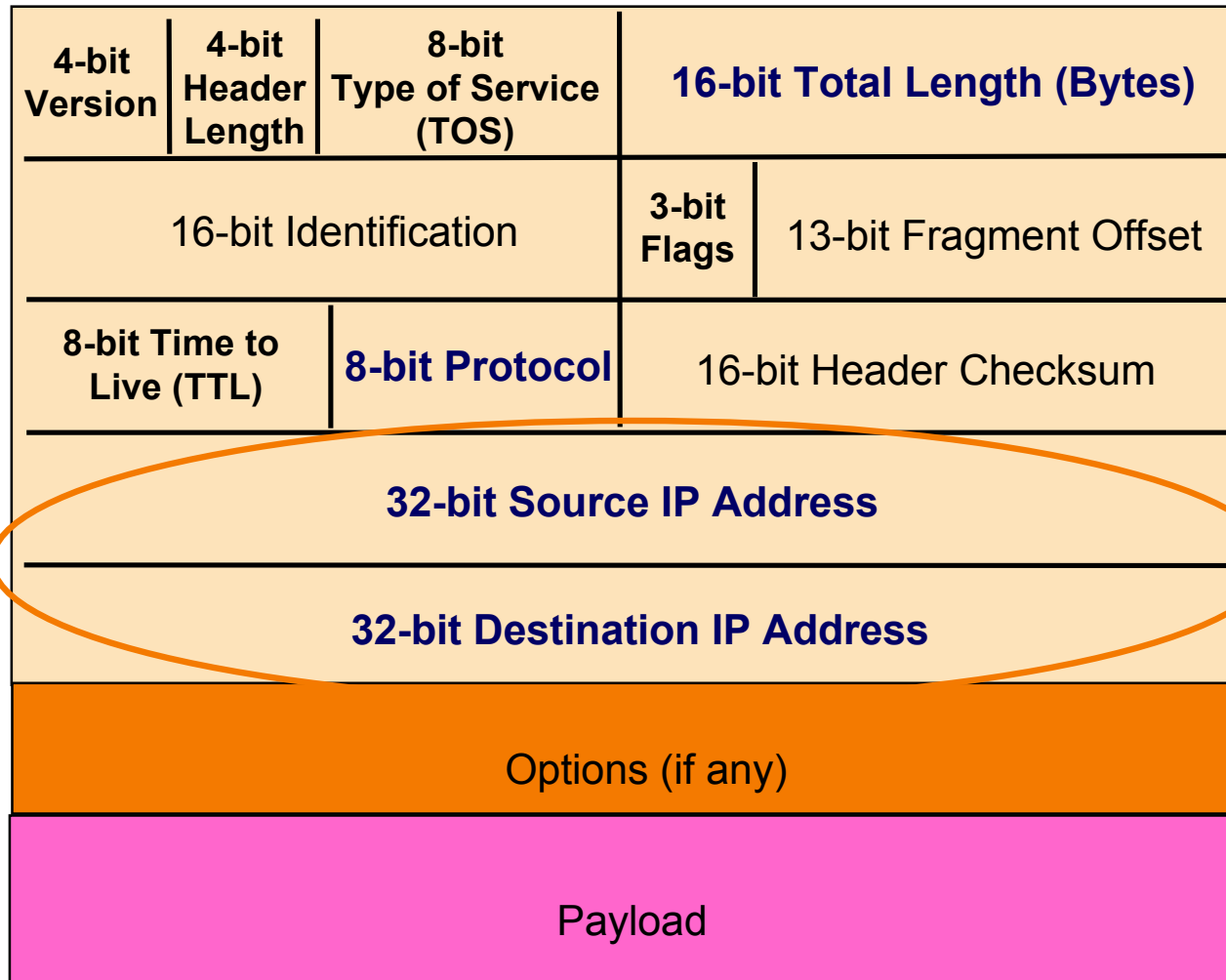
Specifies the length of the entire IP packet: bytes in this header plus bytes in the **Payload**

IP Packet Structure



Specifies how to interpret the start of the **Payload**, which is the header of a *Transport Protocol* such as **TCP** or **UDP**

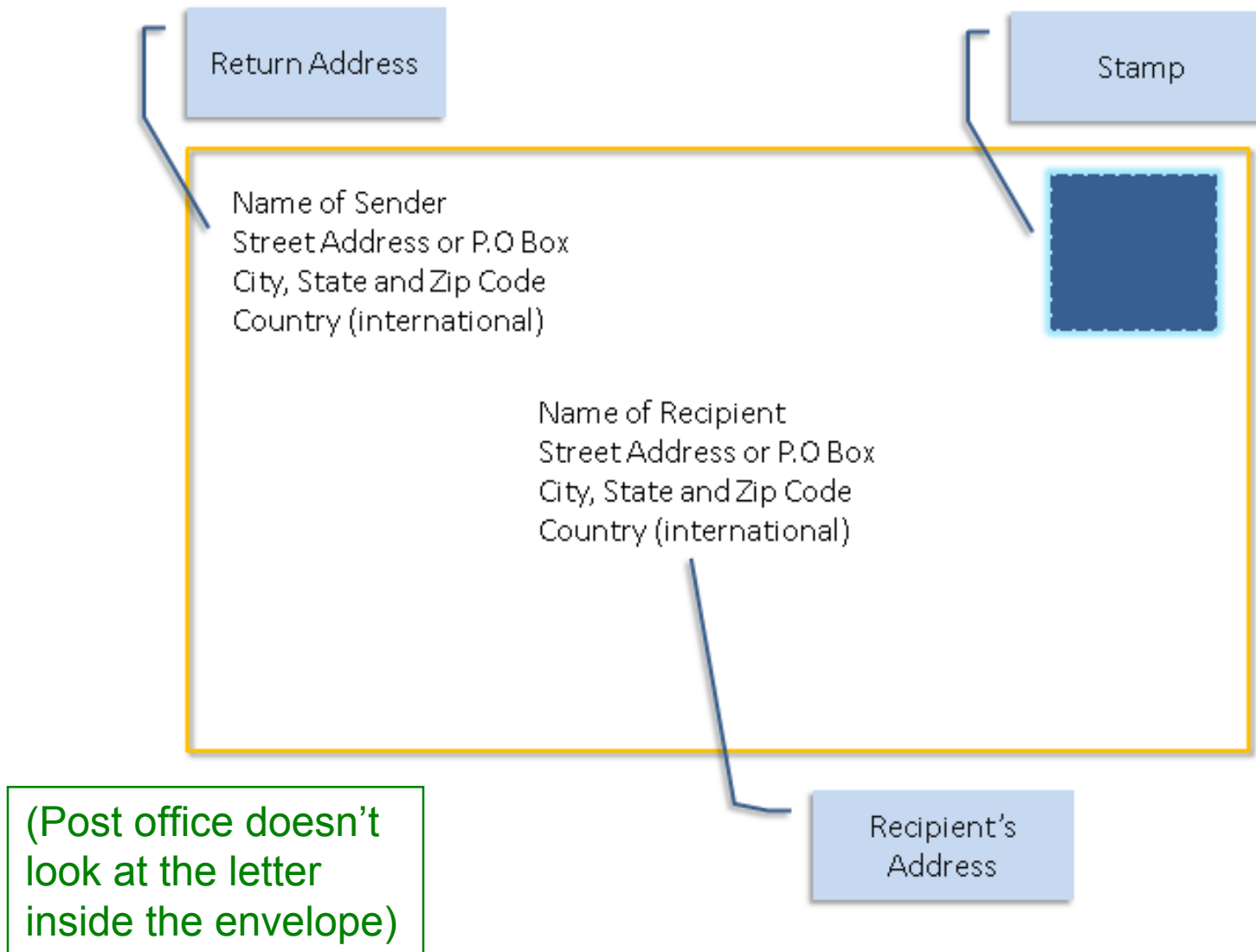
IP Packet Structure



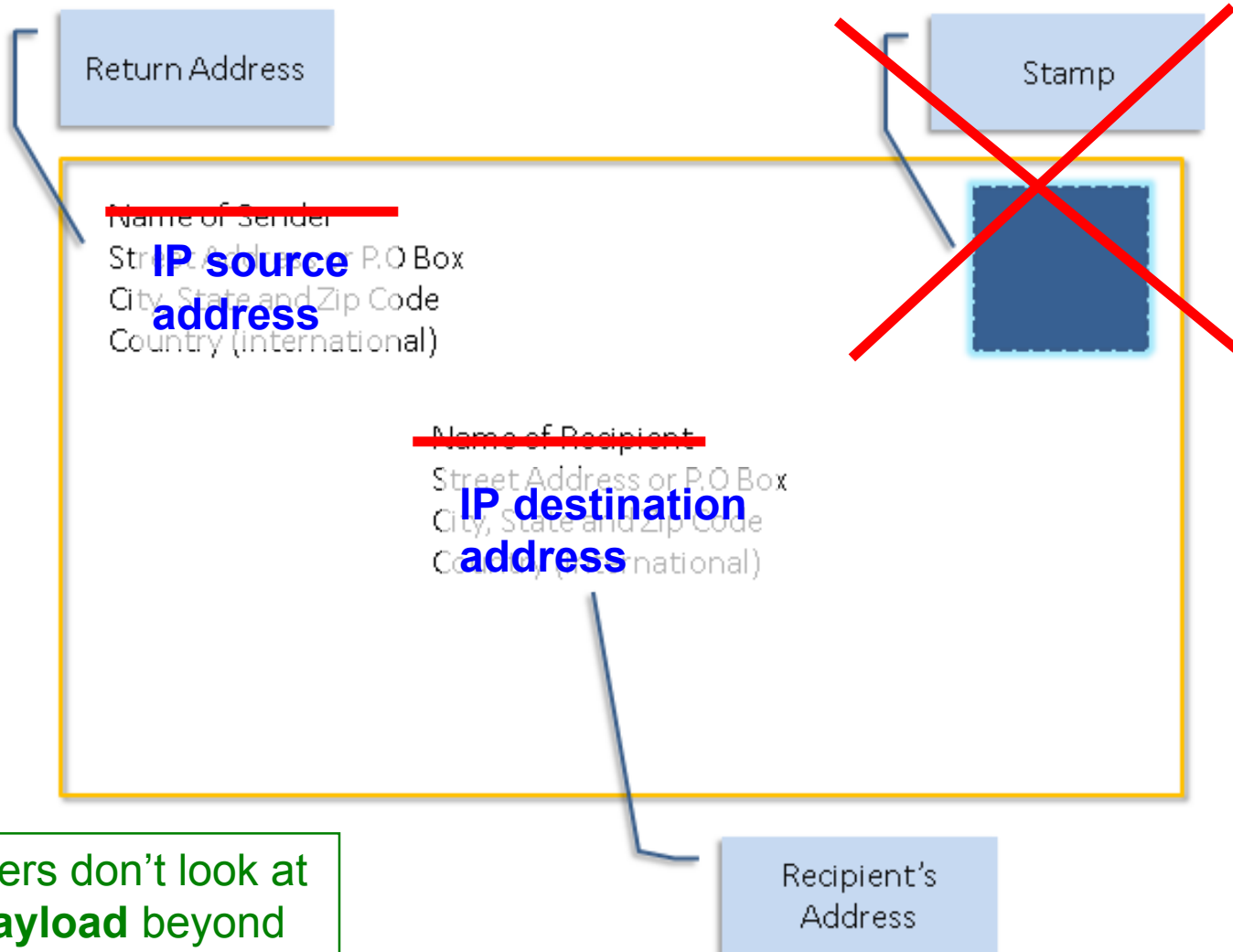
IP Packet Header (Continued)

- Two IP addresses
 - Source IP address (32 bits)
 - Destination IP address (32 bits)
- Destination address
 - Unique **identifier/locator** for the receiving host
 - Allows each node to make forwarding decisions
- Source address
 - Unique identifier/locator for the sending host
 - Recipient can decide whether to accept packet
 - Enables recipient to send a reply back to source

Postal Envelopes:



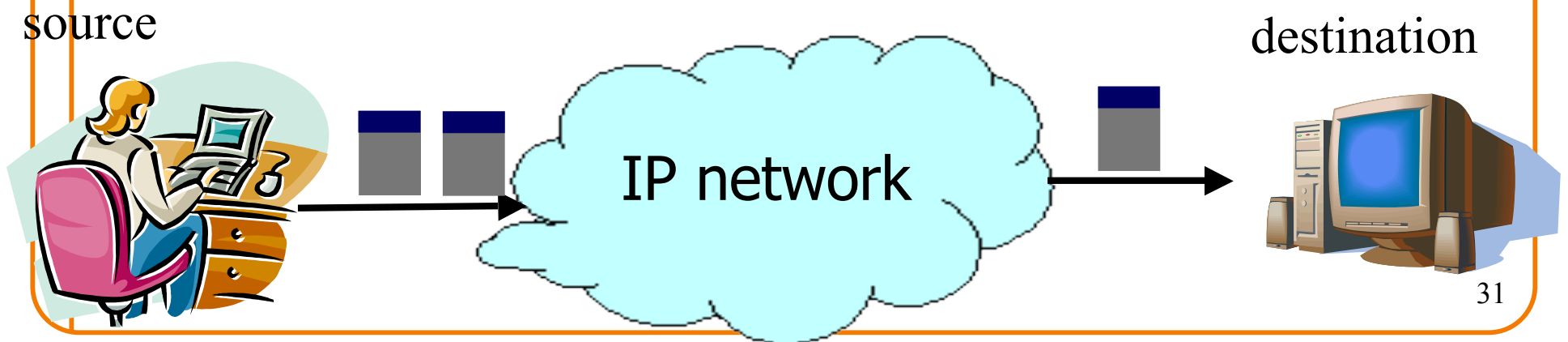
Analogy of IP to Postal Envelopes:



(Routers don't look at the **payload** beyond the IP header)

IP: “*Best Effort*” Packet Delivery

- Routers inspect destination address, locate “next hop” in forwarding table
 - Address = ~unique **identifier/locator** for the receiving host
- Only provides a “*I’ll give it a try*” delivery service:
 - Packets may be lost
 - Packets may be corrupted
 - Packets may be delivered out of order



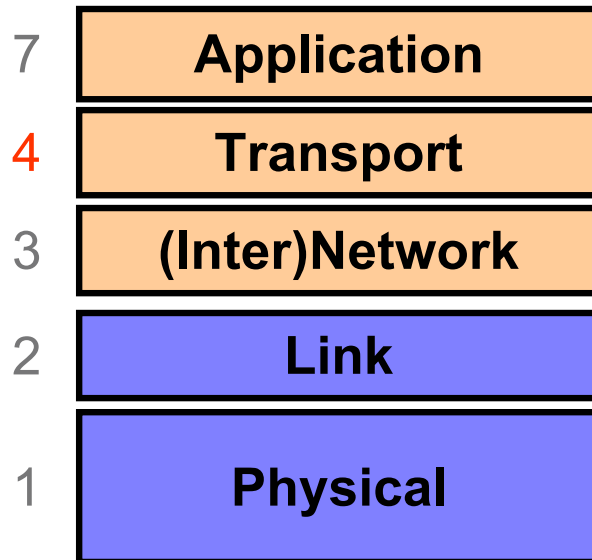
5 Minute Break

Questions Before We Proceed?

“Best Effort” is Lame! What to do?

- It's the job of our Transport (layer 4) protocols to build services our apps need out of IP's modest layer-3 service

Layer 4: Transport Layer



End-to-end communication between processes

Different services provided:

TCP = reliable *byte stream*

UDP = unreliable *datagrams*

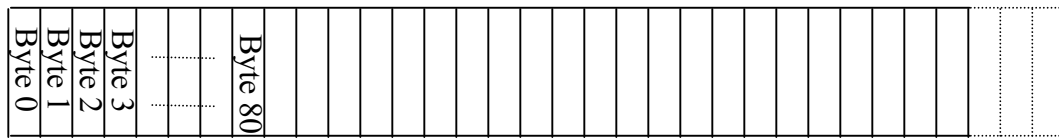
(Datagram = single packet message)

“Best Effort” is Lame! What to do?

- It's the job of our Transport (layer 4) protocols to build services our apps need out of IP's modest layer-3 service
- #1 workhorse: TCP (Transmission Control Protocol)
- Service provided by TCP:
 - Connection oriented (explicit set-up / tear-down)
 - o End hosts (processes) can have multiple concurrent long-lived communication
 - **Reliable**, in-order, *byte-stream* delivery
 - o Robust detection & retransmission of lost data

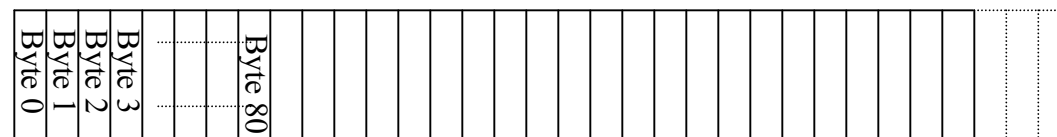
TCP “Bytestream” Service

Process A on host H1



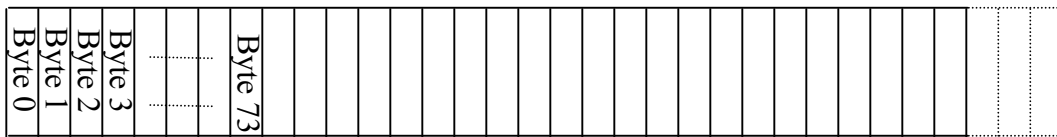
Hosts don't ever see packet boundaries, lost or corrupted packets, retransmissions, etc.

Process B on host H2



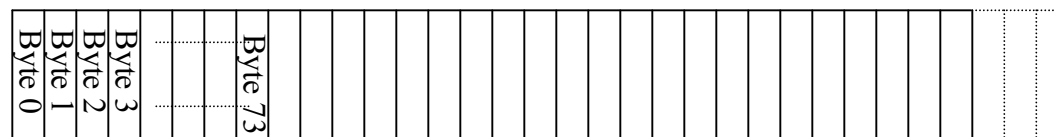
Bidirectional communication:

Process B on host H2

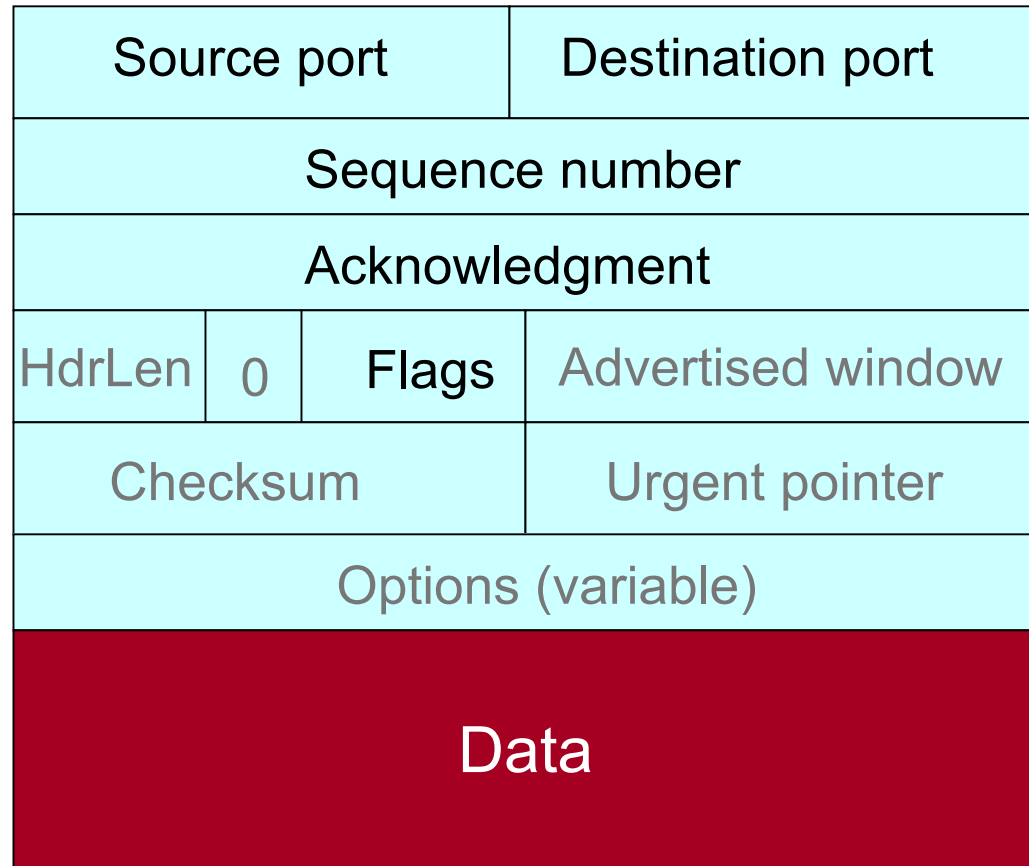


There are two separate **bytestreams**, one in each direction

Process A
on host H1

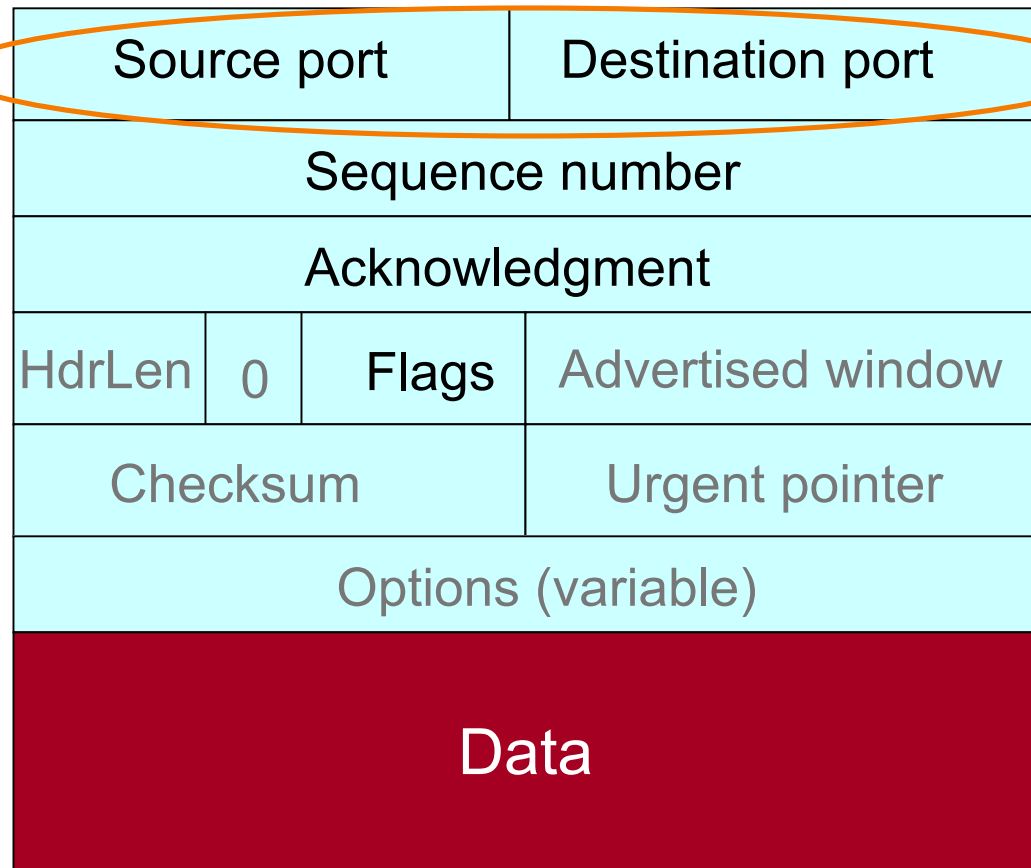


TCP Header

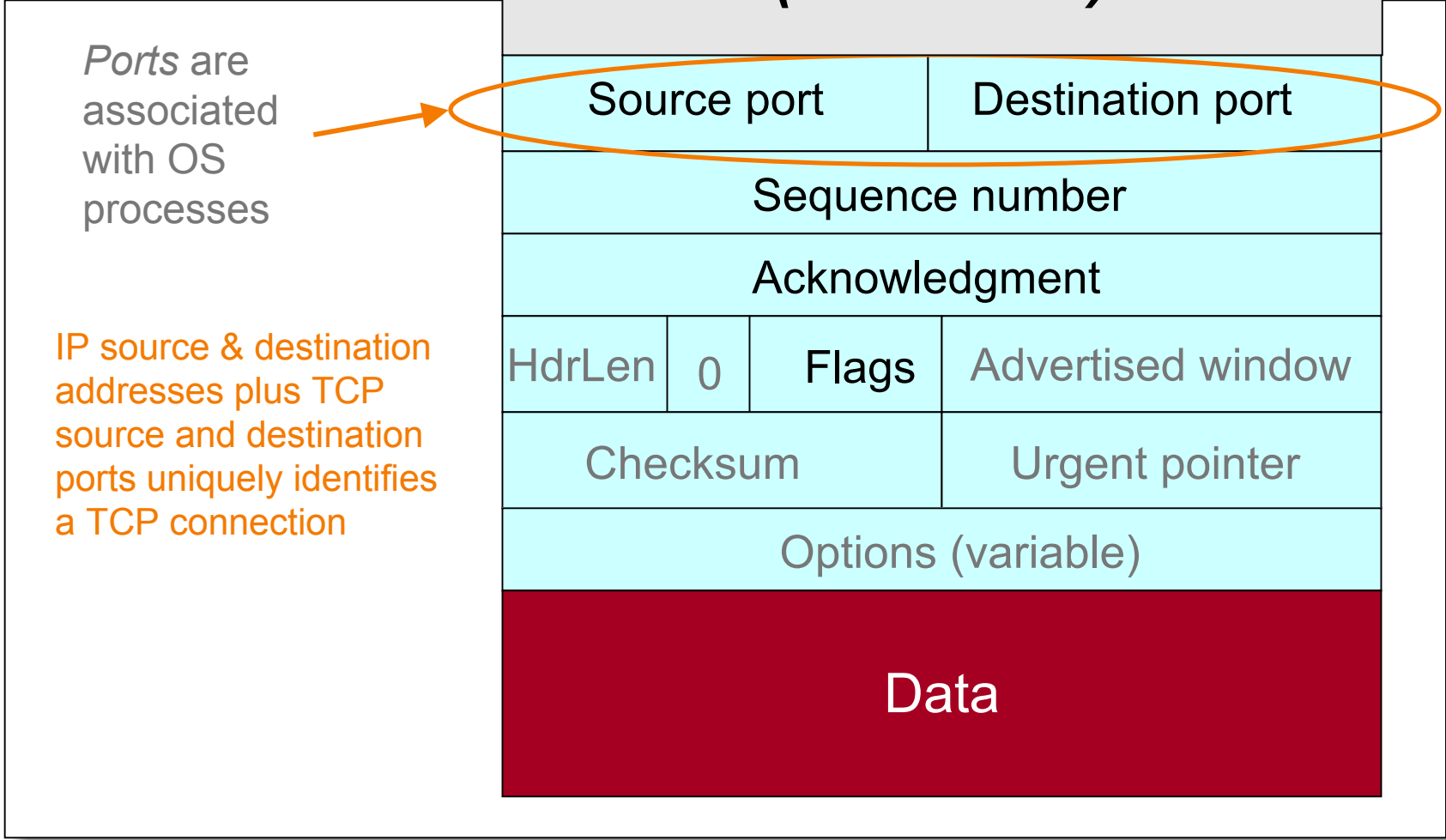


TCP Header

Ports are associated with OS processes



TCP Header



Ports are associated with OS processes

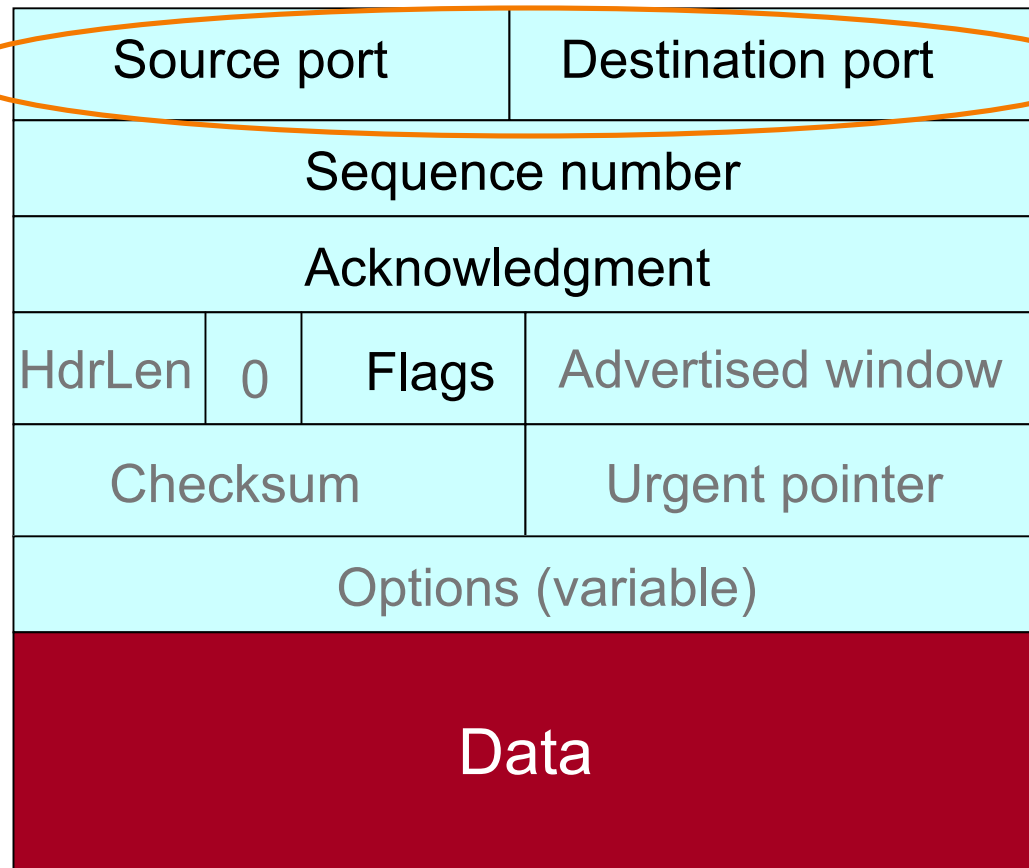
IP source & destination addresses plus TCP source and destination ports uniquely identifies a TCP connection

TCP Header

Ports are associated with OS processes

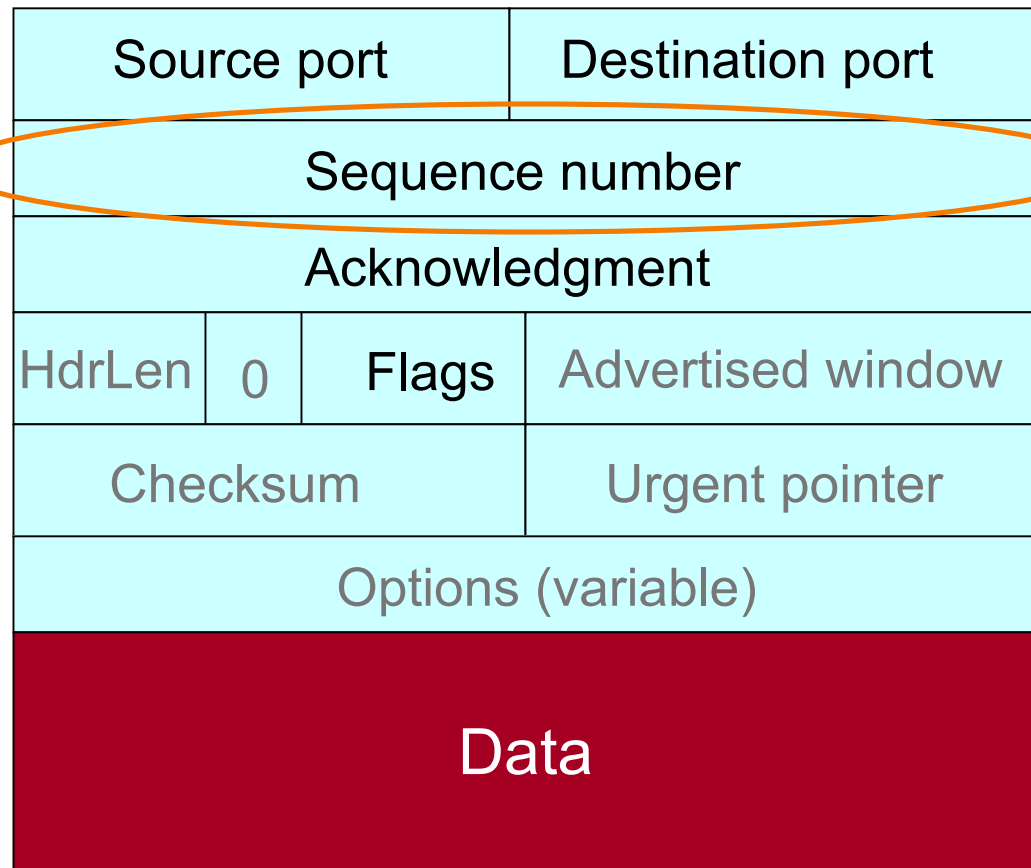
IP source & destination addresses plus TCP source and destination ports uniquely identifies a TCP connection

Some port numbers are "well known" / reserved
e.g. port 80 = HTTP



TCP Header

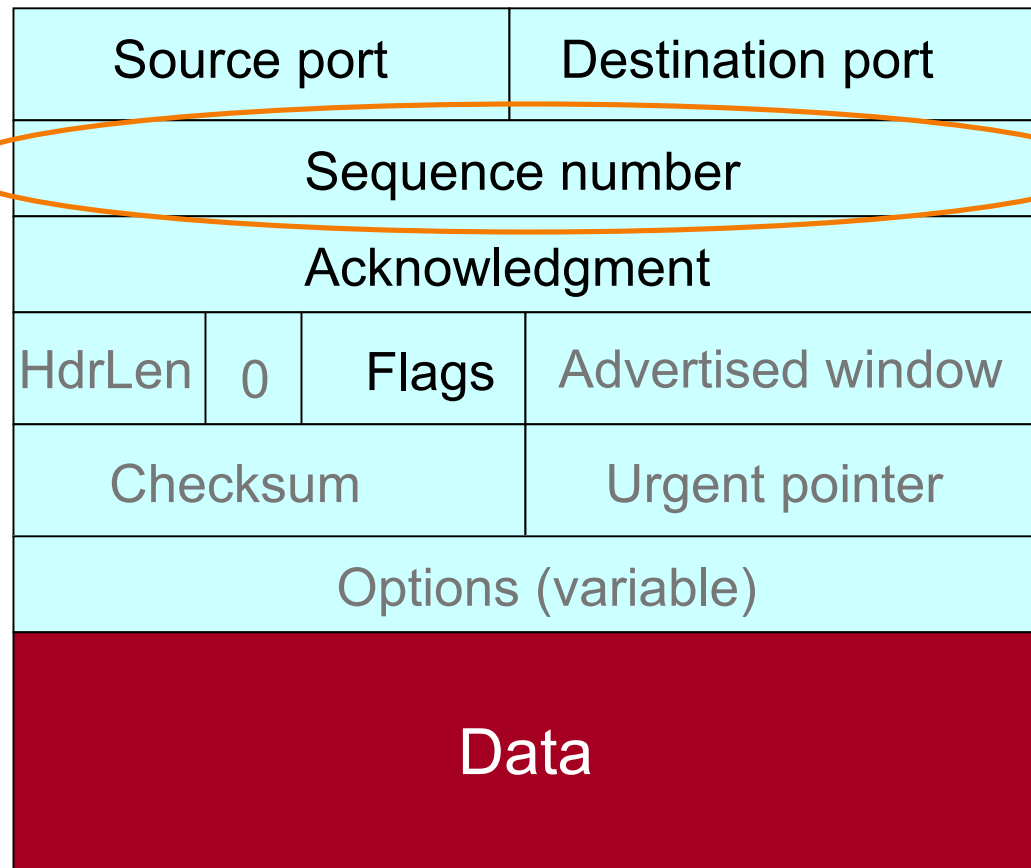
Starting sequence number (byte offset) of data carried in this packet



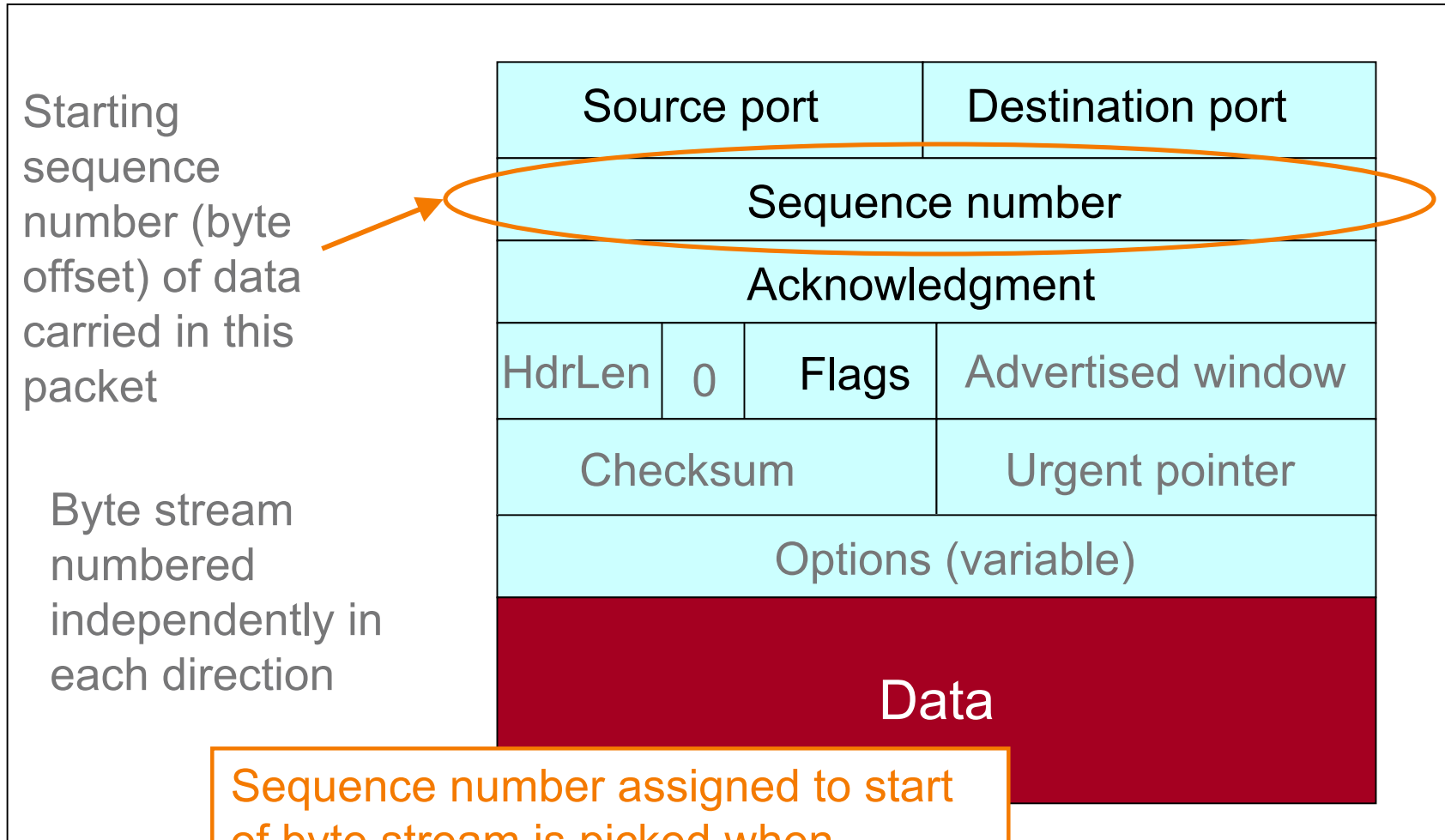
TCP Header

Starting sequence number (byte offset) of data carried in this packet

Byte streams numbered independently in each direction



TCP Header

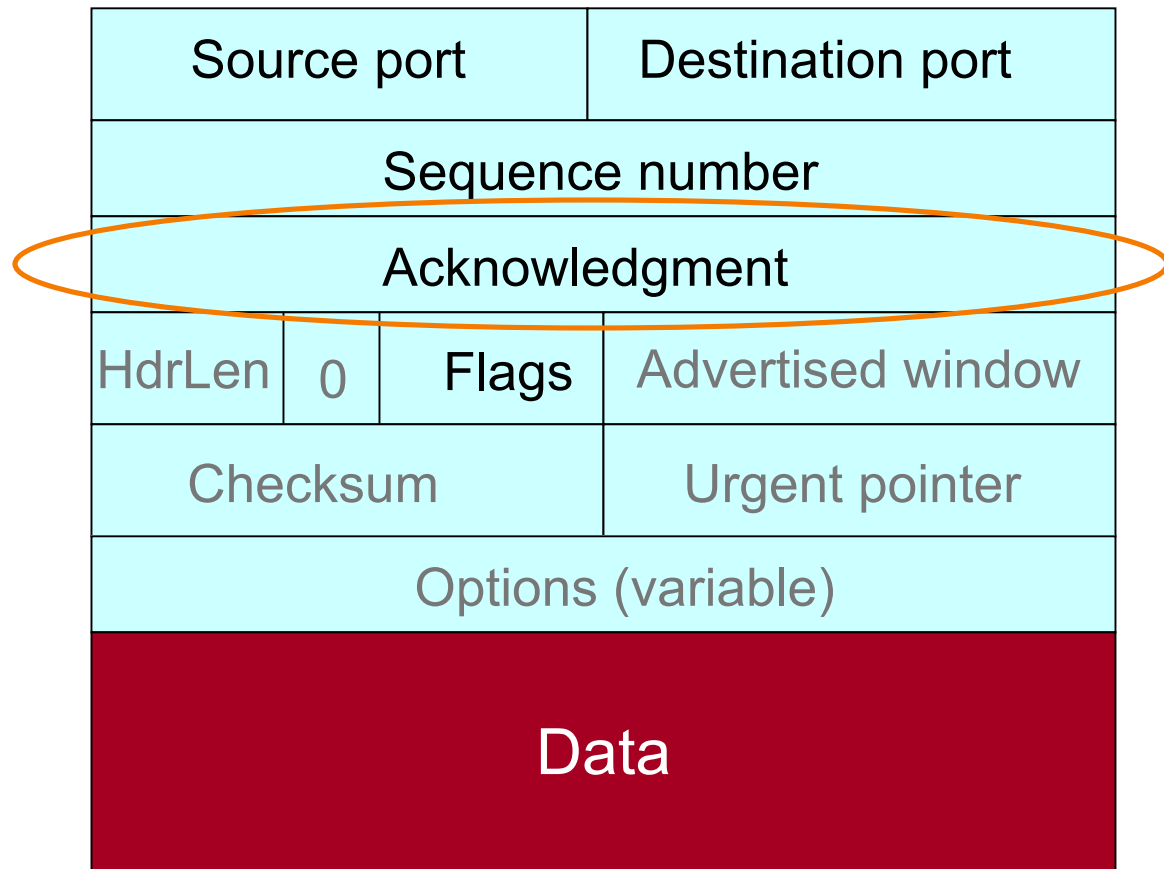


Sequence number assigned to start of byte stream is picked when connection begins; **doesn't** start at 0

TCP Header

Acknowledgment gives seq # **just beyond** highest seq. received **in order**.

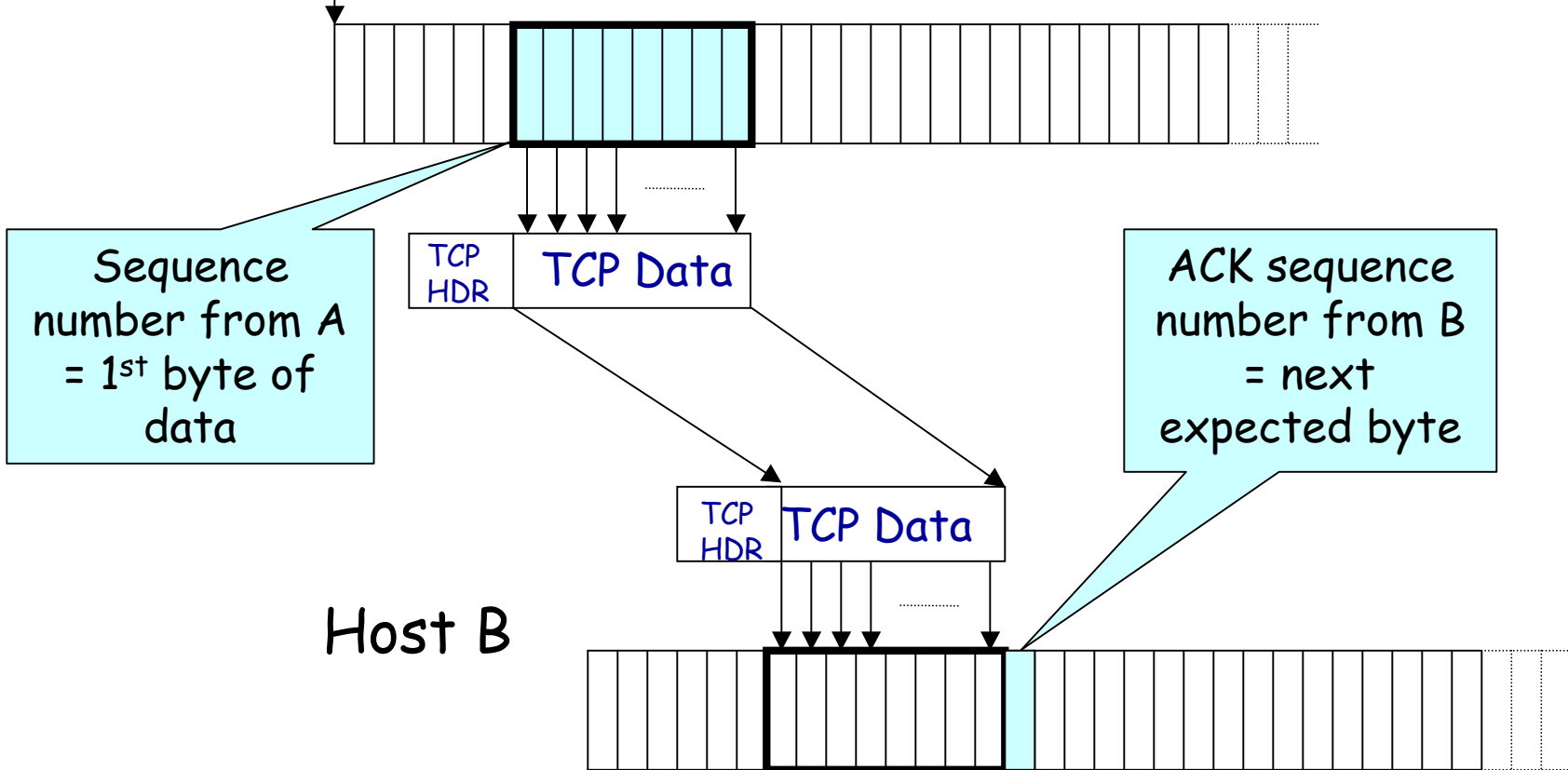
If sender sends **N** bytestream bytes starting at seq **S** then “ack” for it will be **S+N**.



Sequence Numbers

Host A

ISN (initial sequence number)



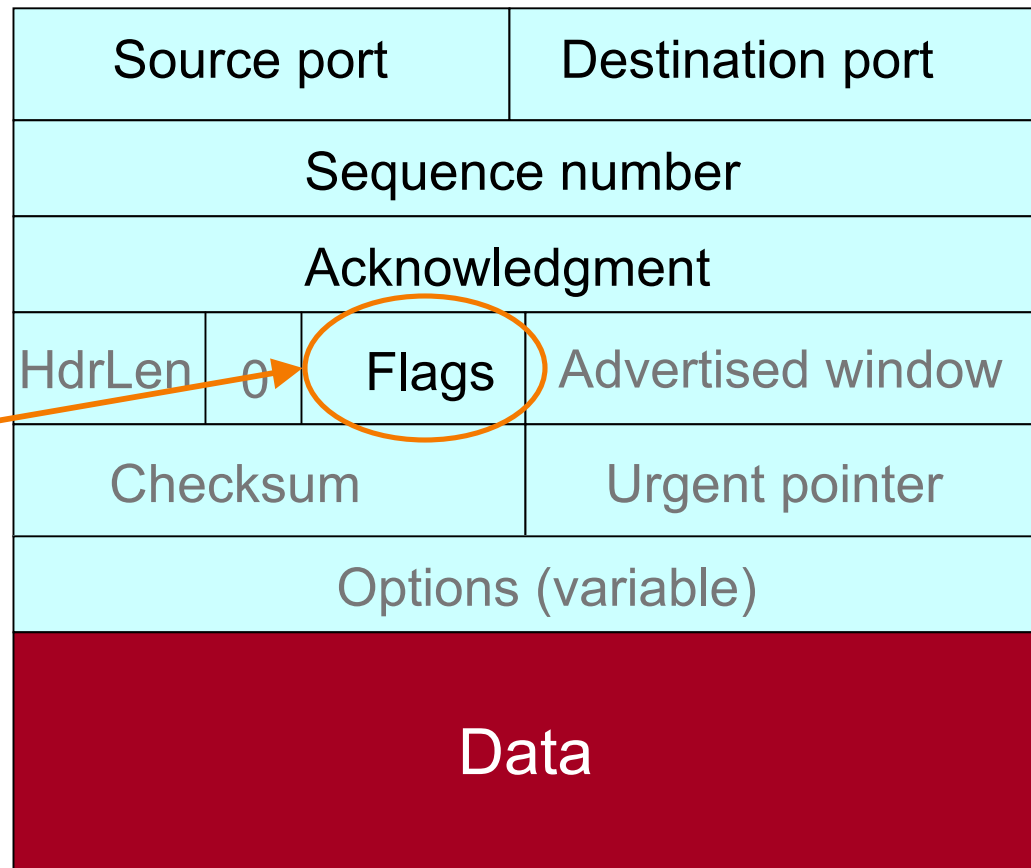
Host B

TCP Header

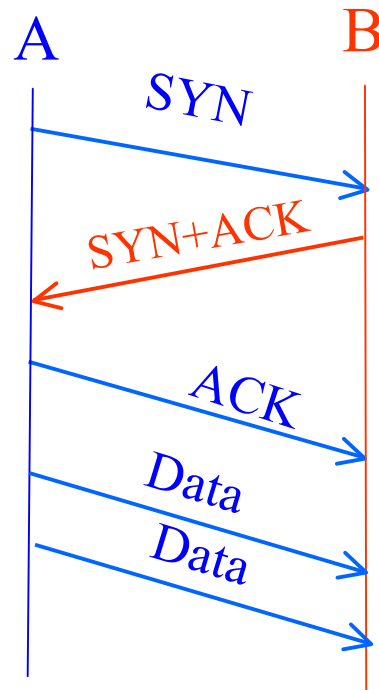
Uses include:

acknowledging data (“**ACK**”)

setting up (“**SYN**”) and closing connections (“**FIN**” and “**RST**”)



Establishing a TCP Connection

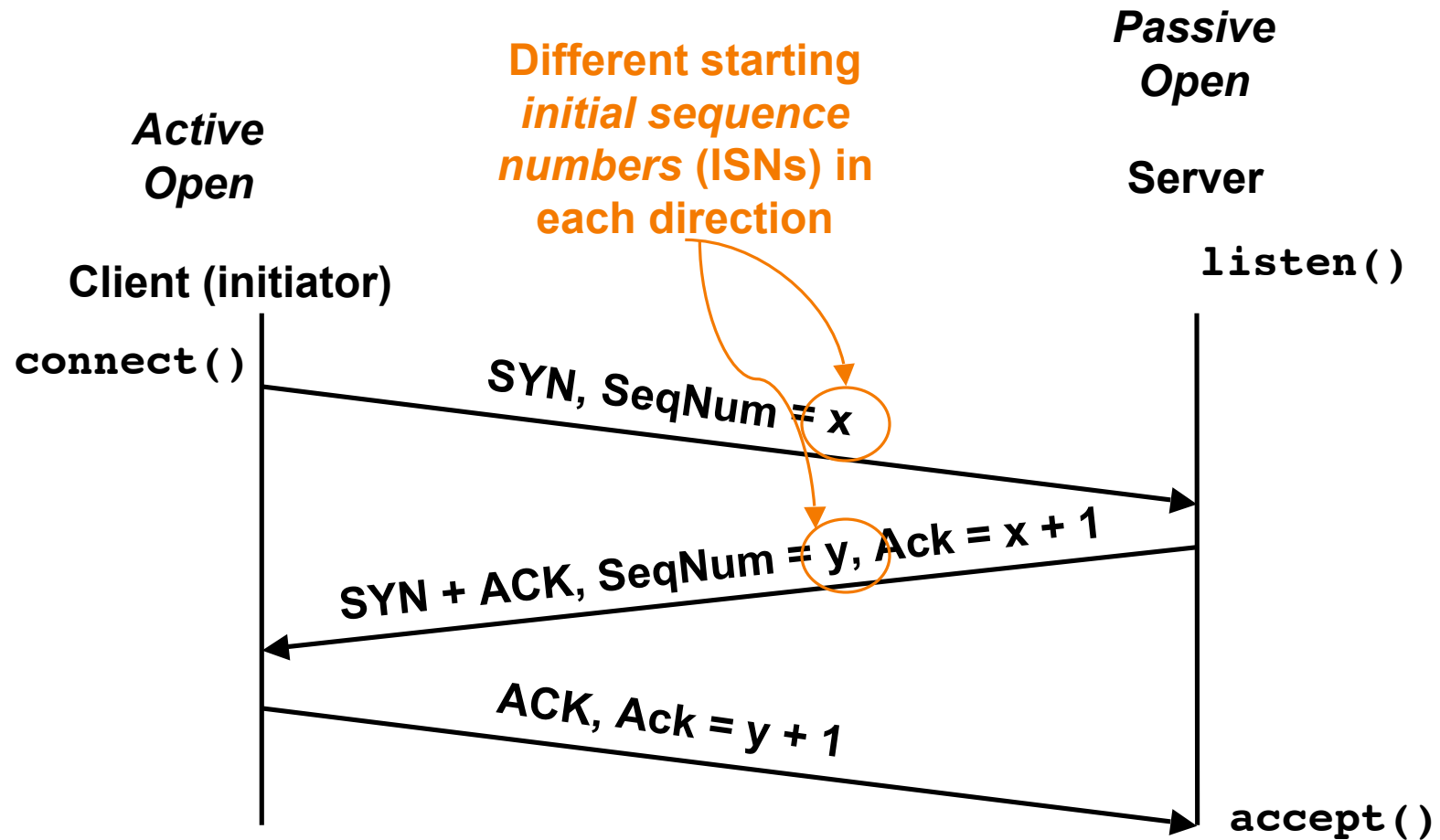


Each host tells its *Initial Sequence Number* (ISN) to the other host.

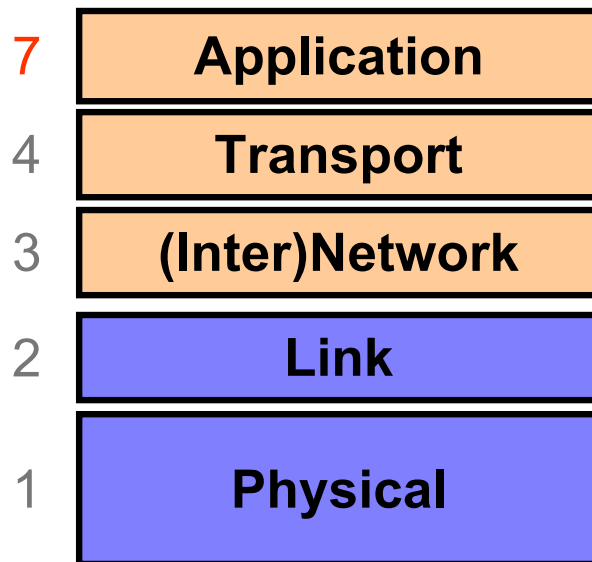
(Spec says to pick based on local clock)

- Three-way handshake to establish connection
 - Host A sends a **SYN** (open; “synchronize sequence numbers”) to host B
 - Host B returns a SYN acknowledgment (**SYN+ACK**)
 - Host A sends an **ACK** to acknowledge the SYN+ACK

Timing Diagram: 3-Way Handshaking



Layer 7: Application Layer



Communication of whatever you wish

Can use whatever transport(s) is convenient

Freely structured

E.g.:

Skype, SMTP (email),
HTTP (Web), Halo, BitTorrent

Web (HTTP) Request

Method

Resource

HTTP version

Headers

GET /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*

Accept-Language: en

Connection: Keep-Alive

User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)

Host: www.example.com

Referer: http://www.google.com?q=dingbats

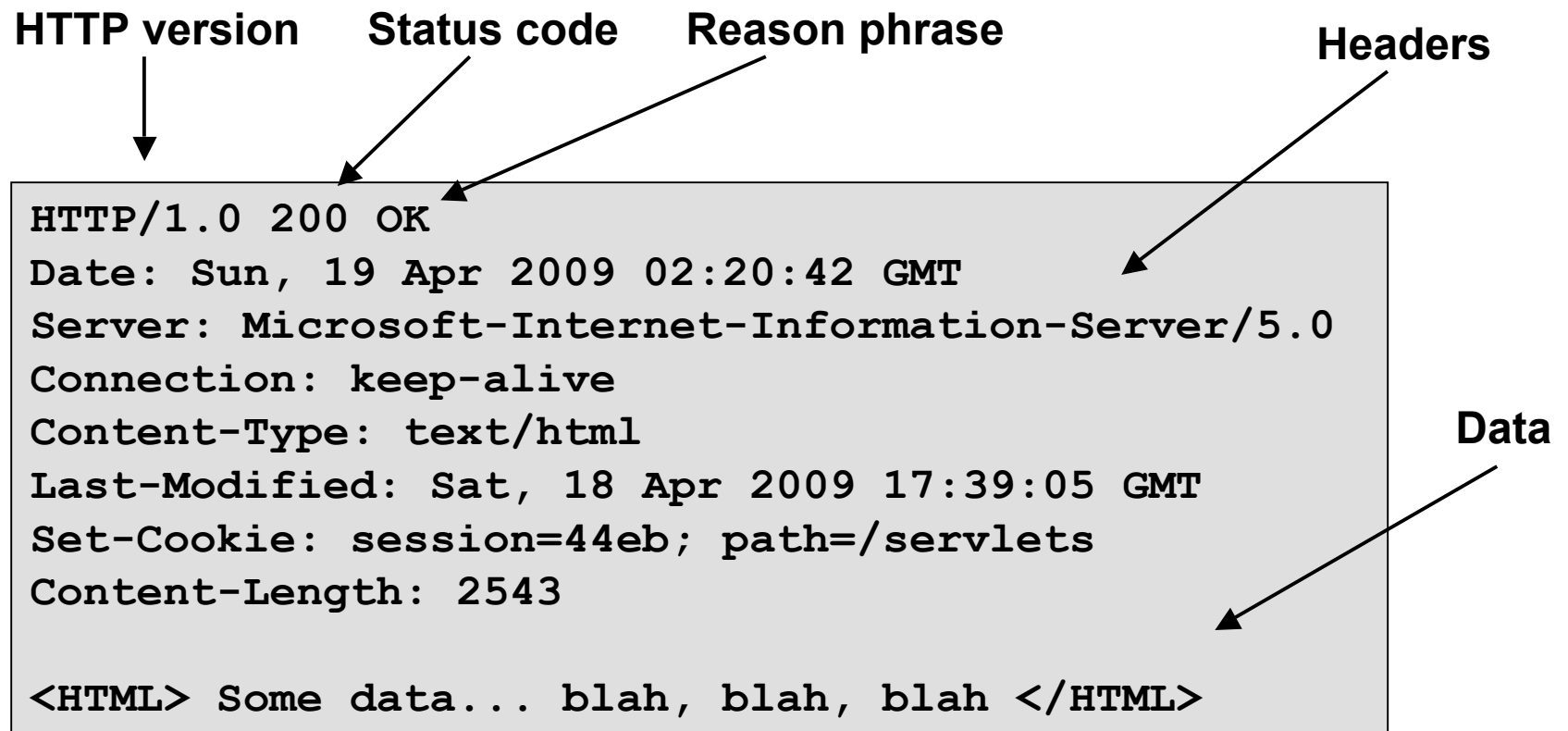
Blank line

Data (if POST; none for GET)

GET: download data.

POST: upload data.

Web (HTTP) Response



Host Names vs. IP addresses

- Host names
 - Examples: `www.cnn.com` and `bbc.co.uk`
 - Mnemonic name appreciated by **humans**
 - Variable length, full alphabet of characters
 - Provide little (if any) information about location
- IP addresses
 - Examples: `64.236.16.20` and `212.58.224.131`
 - Numerical address appreciated by **routers**
 - Fixed length, binary number
 - Hierarchical, related to host location

Mapping Names to Addresses

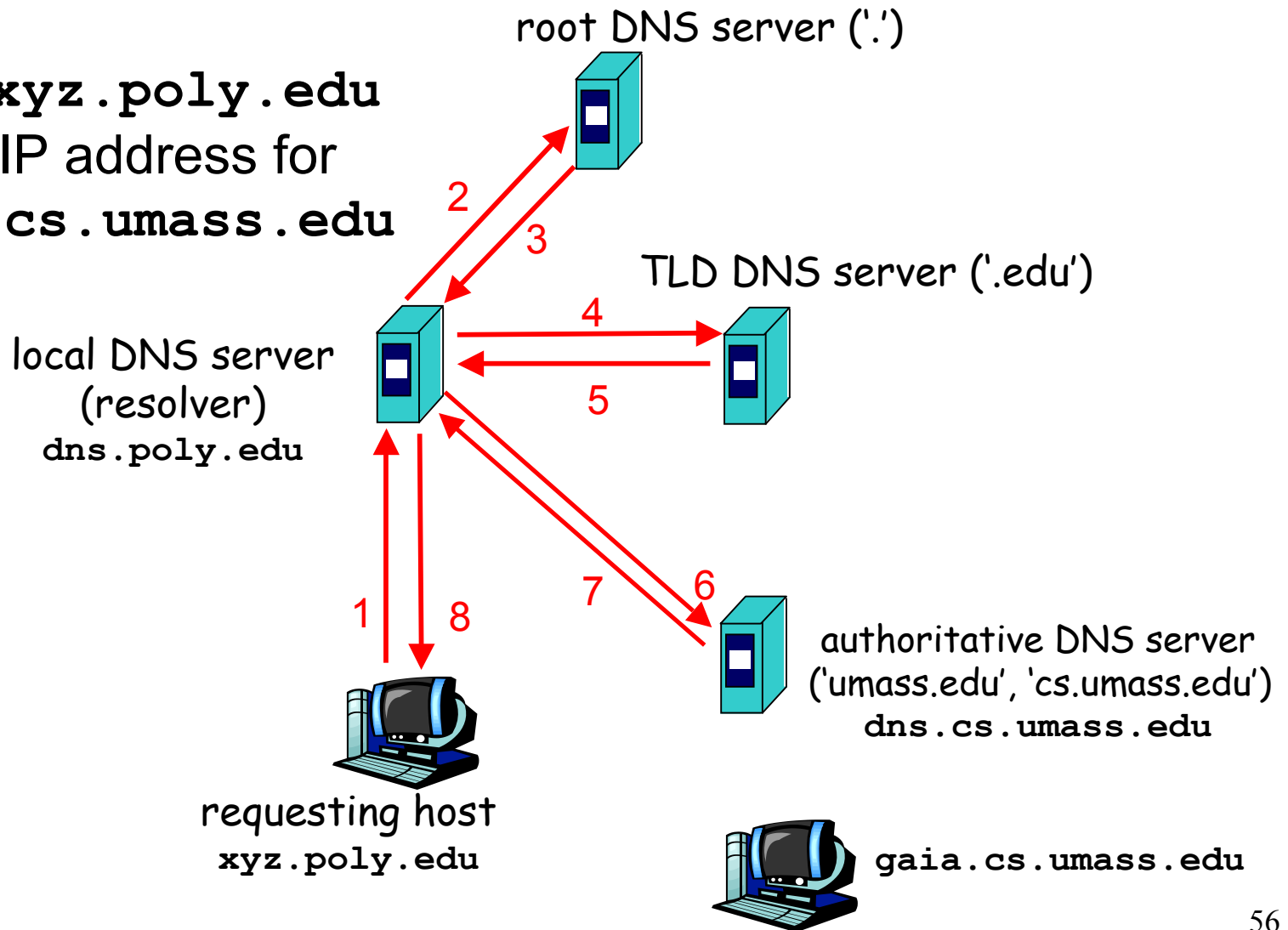
- Domain Name System (DNS)
 - Hierarchical name space divided into sub-trees (“zones”)
 - Zones distributed over collection of DNS *name servers*
- Hierarchy of DNS servers
 - Root (**hardwired** into other servers)
 - Top-level domain (**TLD**) servers
 - “Authoritative” DNS servers (e.g. for *berkeley.edu*)

Mapping Names to Addresses

- Domain Name System (DNS)
 - Hierarchical name space divided into *zones*
 - Zones distributed over collection of DNS name servers
- Hierarchy of DNS servers
 - Root (hardwired into other servers)
 - Top-level domain (TLD) servers
 - “Authoritative” DNS servers (e.g. for *berkeley.edu*)
- Performing the translations
 - Each computer configured to contact a *resolver*

Example

Host at **xyz.poly.edu**
wants IP address for
gaia.cs.umass.edu



DNS Protocol

DNS protocol: *query* and *reply* messages, both with *same message format*

(Mainly uses UDP transport rather than TCP)

Message header:

- **Identification:** 16 bit # for query, reply to query uses same #
- Replies can include “Authority” (name server responsible for answer) and “Additional” (info client is likely to look up soon anyway)
- Replies have a **Time To Live** (in seconds) for **caching**

16 bits	16 bits
Identification	Flags
# Questions	# Answer RRs
# Authority RRs	# Additional RRs
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

Questions?