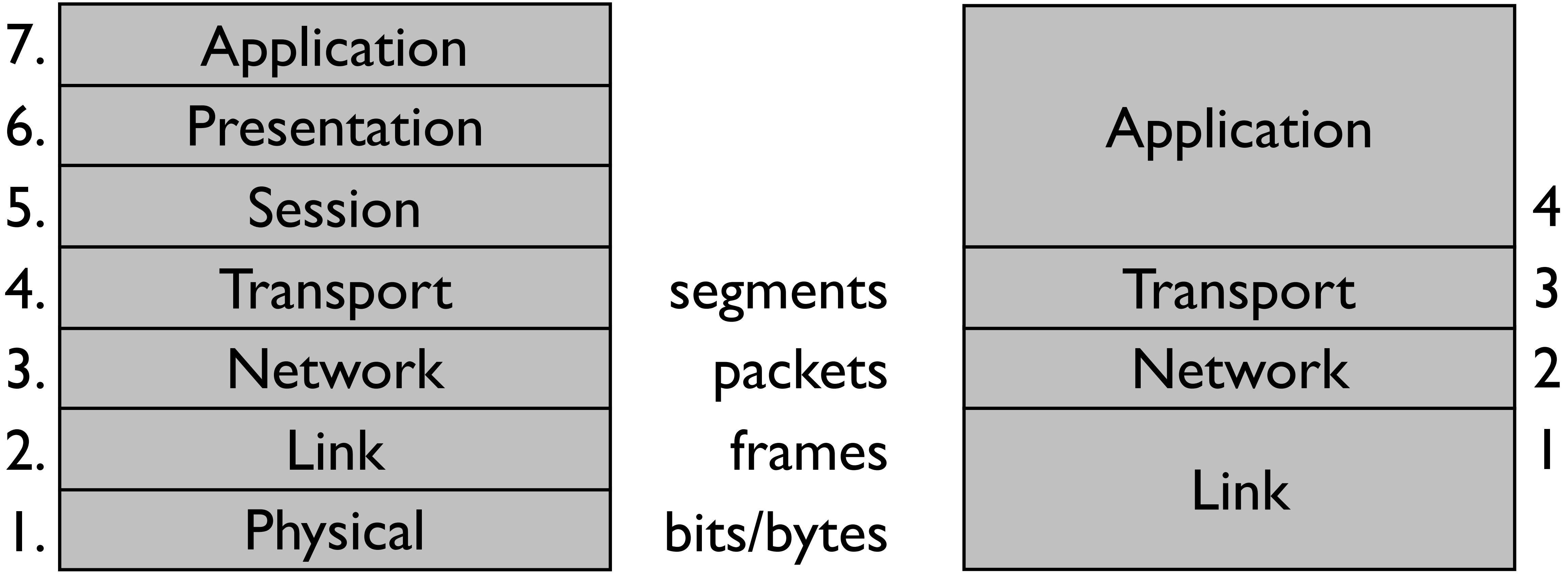
Transport:
How Applications Communicate

Week 2

Philip Levis

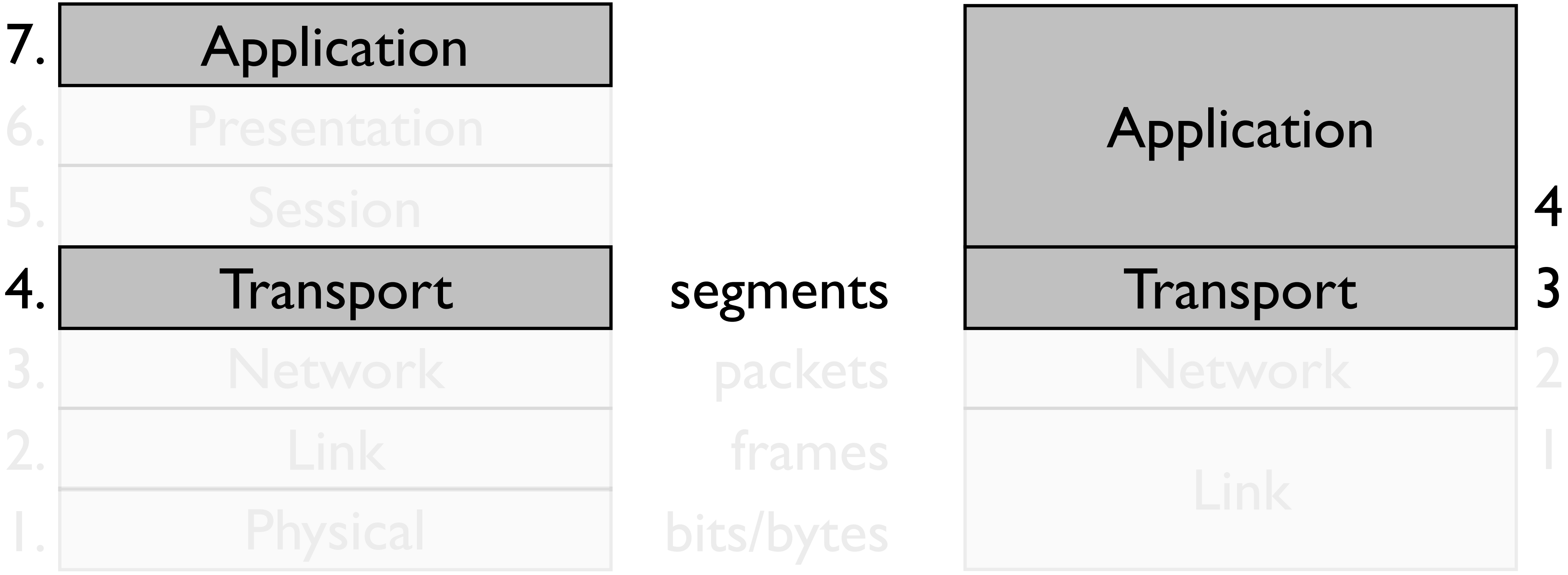
7 Layers (or 4)



7. Application
6. Presentation
5. Session
4. Transport — segments
3. Network — packets
2. Link — frames
1. Physical — bits/bytes

4 Application
3 Transport
2 Network
1 Link

7 Layers (or 4)



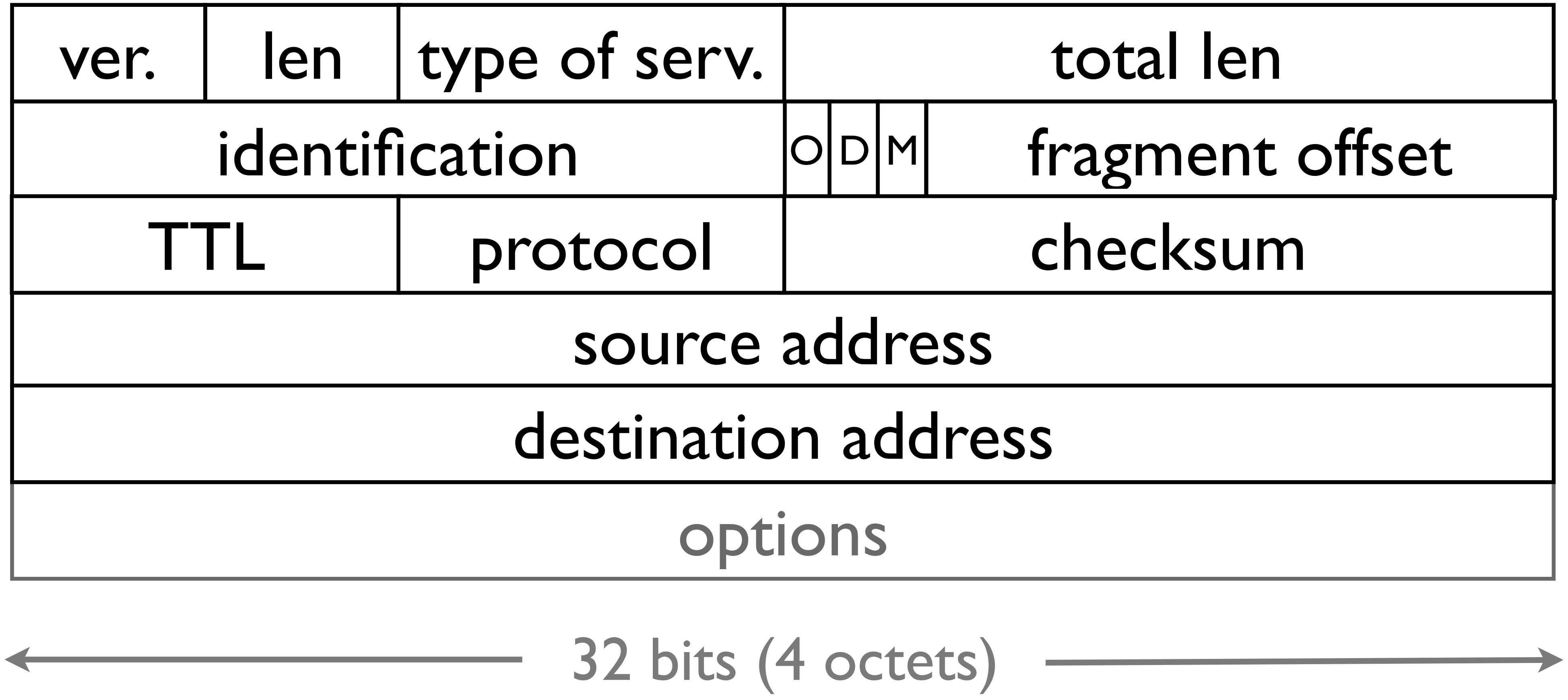
7. Application
6. Presentation
5. Session
4. Transport — segments
3. Network — packets
2. Link — frames
1. Physical — bits/bytes

4 Application
3 Transport
2 Network
1 Link

Transport

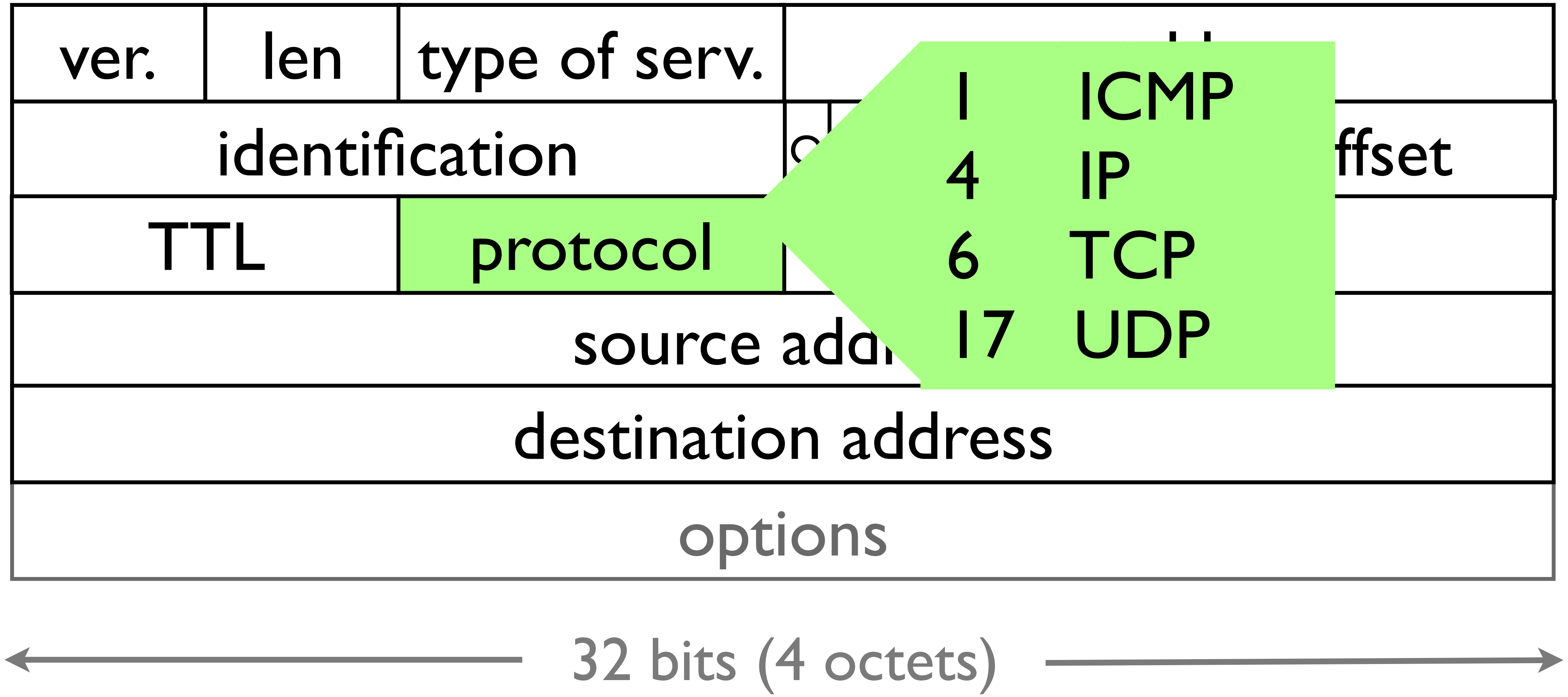
- Provides inter-program communication
  - ICMP: control messages to operating system
  - UDP: unreliable datagrams to user programs
  - TCP: reliable stream to user programs
- Evidenced by *naming*
  - IP packets are addressed to hosts with *addresses*
  - UDP and TCP segments are named to programs with *ports*
  - ICMP is implicitly named to operating system/IP software

IP Header



<table>
<tr><td>ver.</td><td>len</td><td>type of serv.</td><td colspan="4">total len</td></tr>
<tr><td colspan="3">identification</td><td>O</td><td>D</td><td>M</td><td>fragment offset</td></tr>
<tr><td colspan="2">TTL</td><td>protocol</td><td colspan="4">checksum</td></tr>
<tr><td colspan="7">source address</td></tr>
<tr><td colspan="7">destination address</td></tr>
<tr><td colspan="7">options</td></tr>
</table>

IP Header



ver. | len | type of serv.

identification | ffset

TTL | protocol

source add

destination address

options

32 bits (4 octets)

1	ICMP
4	IP
6	TCP
17	UDP

ICMP

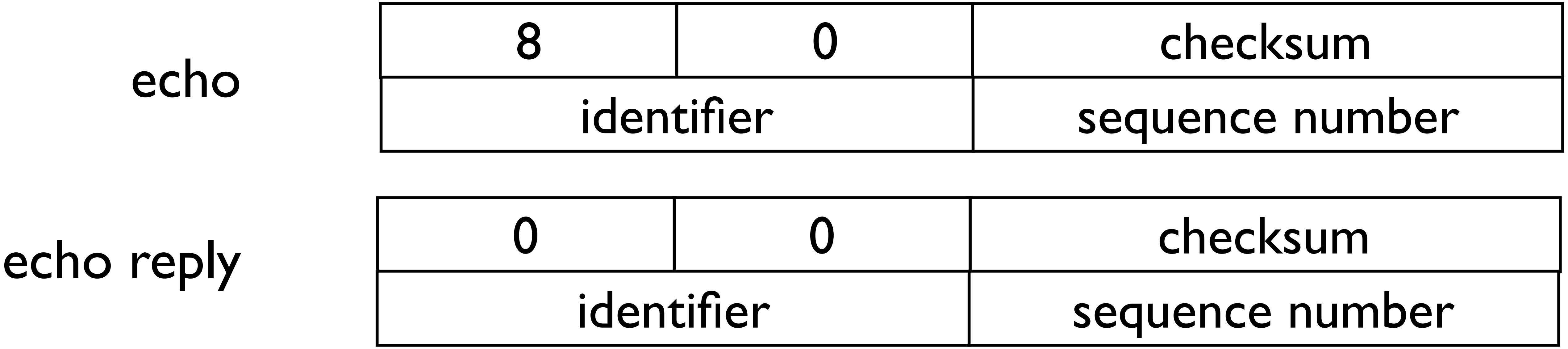
- Internet Control Message Protocol, RFC 792
- Way for Internet hosts to send control information
- You’ll work a lot with ICMP in lab 3 (router)
- Unreliable datagrams

type	code	checksum
data		

Example: type 3 is destination unreachable
  code 0: net unreachable
  code 1: host unreachable
  code 2: protocol unreachable
  code 3: port unreachable…

ICMP: ping

- ping, a very basic tool!
- Source sends an ICMP Echo message
- Destination replies with an ICMP Echo Reply message



echo

8	0	checksum
identifier		sequence number

echo reply

0	0	checksum
identifier		sequence number

ICMP: traceroute

- Send UDP segments to destination with increasing TTL
- ICMP type 11: time to live exceeded

11	0	checksum
first 64 bits of generating packet		

UDP

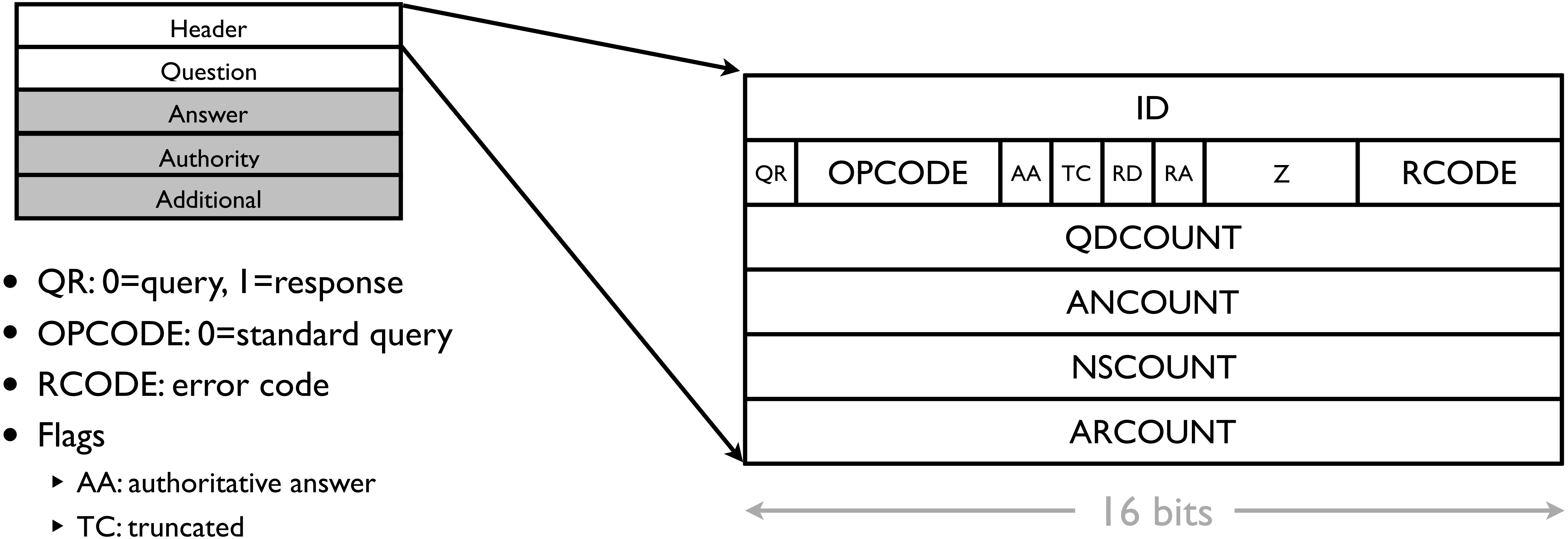
- User Datagram Protocol, RFC 768
- Very thin layer on top of IP, just adds ports
- Unreliable, datagrams

source port	destination port
UDP len	checksum

UDP: DNS

- Example UDP program: Domain Name System (DNS)
- Maps names like cs.stanford.edu to IP addresses
- UDP port 53
- Learn details about DNS in Week 5

DNS Header Structure (RFC1035)



Header

Question

Answer

Authority

Additional

ID

QR | OPCODE | AA | TC | RD | RA | Z | RCODE

QDCOUNT

ANCOUNT

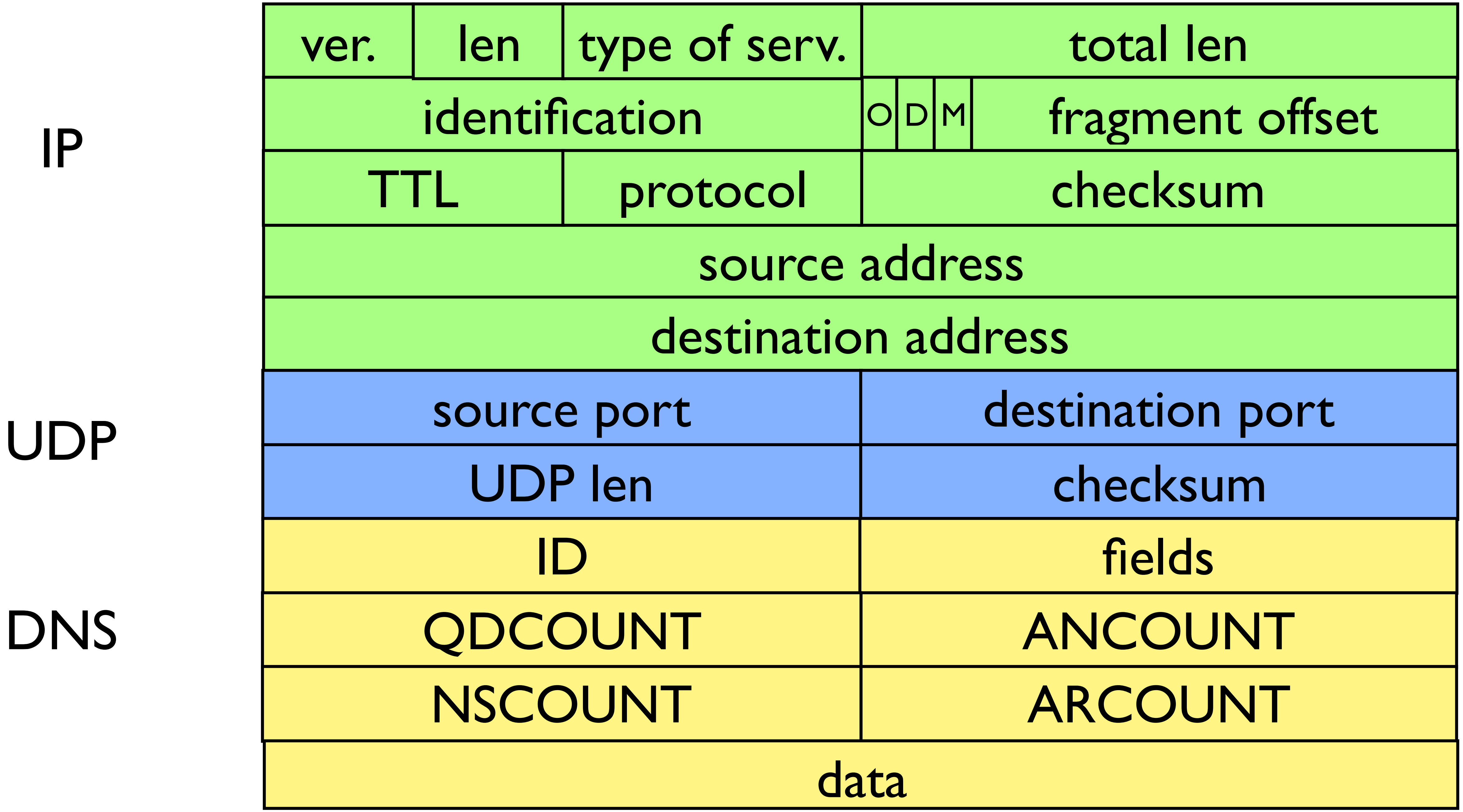
NSCOUNT

ARCOUNT

16 bits

- QR: 0=query, 1=response
- OPCODE: 0=standard query
- RCODE: error code
- Flags
  - AA: authoritative answer
  - TC: truncated
  - RD: recursion desired
  - RA: recursion available

Encapsulation



IP

ver. | len | type of serv. | total len

identification | O | D | M | fragment offset

TTL | protocol | checksum

source address

destination address

UDP

source port | destination port

UDP len | checksum

DNS

ID | fields

QDCOUNT | ANCOUNT

NSCOUNT | ARCOUNT

data

TCP

- Transmission Control Protocol, RFC 793
- Different abstraction: bidirectional, reliable byte stream
  - Building block of most applications today
- Abstracts away entire network -- just a pipe between two programs
  - One side reads what the other writes
- Application level controls communication pattern and payloads
  - World Wide Web (HTTP)
  - Skype
  - BitTorrent

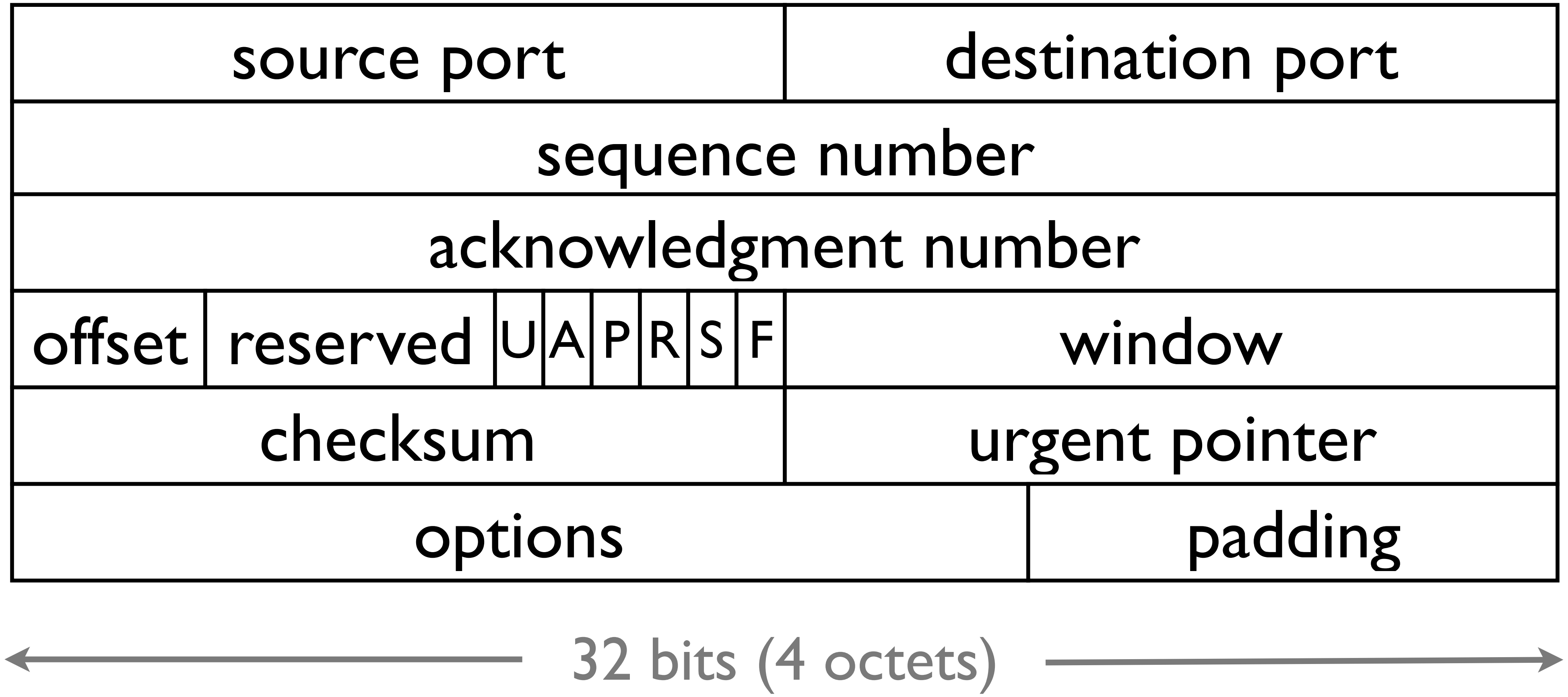
Audacious Idea

- TCP: make a reliable data stream out of an unreliable network
  - Can fail, but almost always explicitly detected (connection breaks)
  - Assumes random errors, not malicious ones
- Part of a larger theme in computer systems, making robust, high performance computing out of cheap, unreliable parts
  - TCP from IP datagrams
  - RAID: Redundant Array of Inexpensive Disks
  - Early cloud computing systems (MapReduce, Hadoop, etc.)
  - Domain Name System

How to Start?

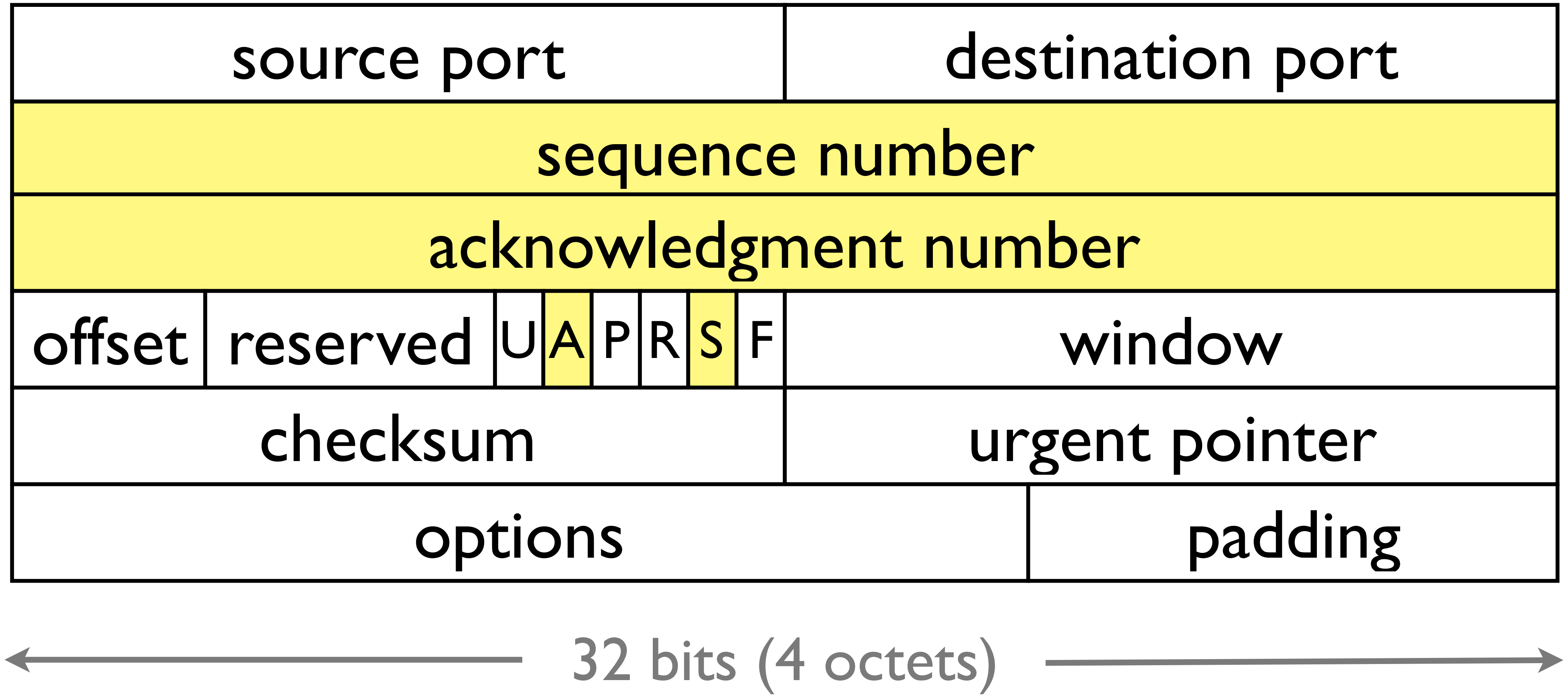
- Reliable communication typically benefits from have some state on each end of a connection
  - Need to be able to identify data to determine if it's been delivered
  - For a stream, need to know where in stream data is
- Problem: connection establishment
  - How do you set up this state?
- Problem: connection teardown
  - How do you clean up (reuse ports, etc.)?

TCP Header



<table>
<tr><td colspan="8">source port</td><td>destination port</td></tr>
<tr><td colspan="9">sequence number</td></tr>
<tr><td colspan="9">acknowledgment number</td></tr>
<tr><td>offset</td><td>reserved</td><td>U</td><td>A</td><td>P</td><td>R</td><td>S</td><td>F</td><td>window</td></tr>
<tr><td colspan="8">checksum</td><td>urgent pointer</td></tr>
<tr><td colspan="8">options</td><td>padding</td></tr>
</table>

Connection Setup



source port | destination port

sequence number

acknowledgment number

offset | reserved | U | A | P | R | S | F | window

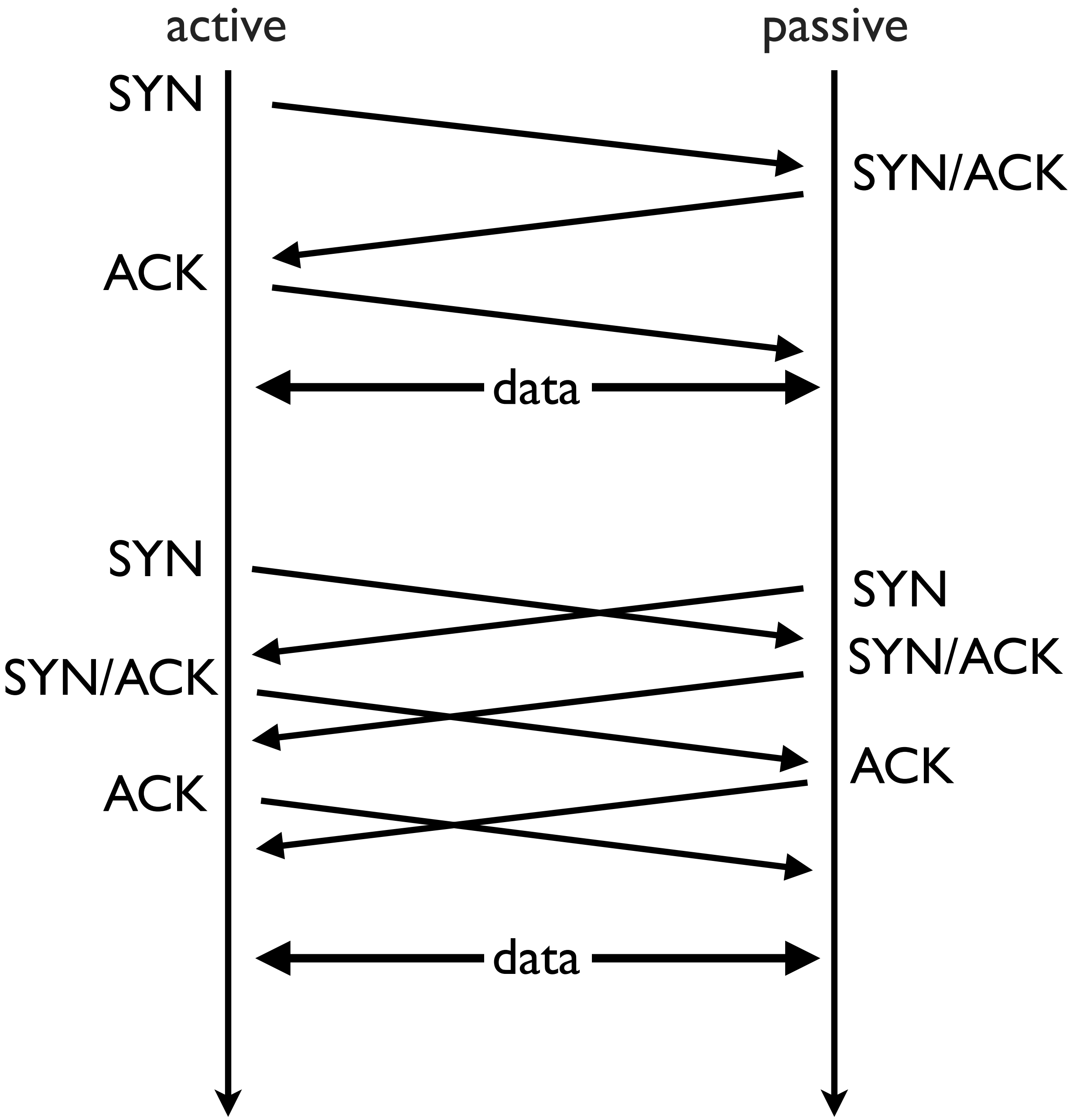
checksum | urgent pointer

options | padding

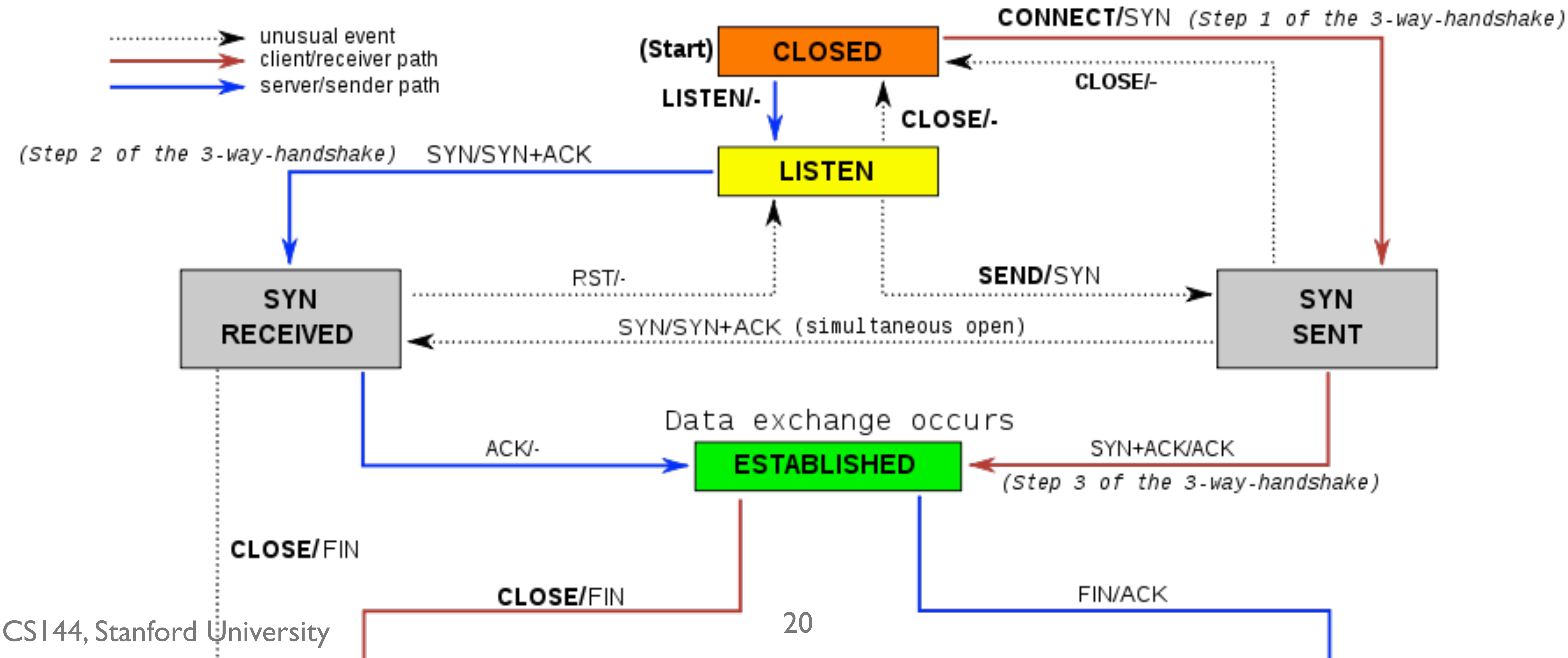
32 bits (4 octets)

3-way Handshake

- Active opener sends first packet
  - SYN with sequence number
- Passive opener responds
  - SYN with sequence number
  - ACKs active opener's SYN packet
- Active opener responds
  - ACKs passive opener's SYN packet
- Also support "simultaneous open"
  - Two SYNs pass each other
  - Each side ACKs the other



TCP Setup FSM



Conceptual Model

- SYN message tells destination endpoint the starting sequence number
  - Can't send data until it acknowledges it knows the starting sequence number
- ACK of SYN tells source that this endpoint knows the starting seq no.
- Happens in both directions: bidirectional dream

Connection established!

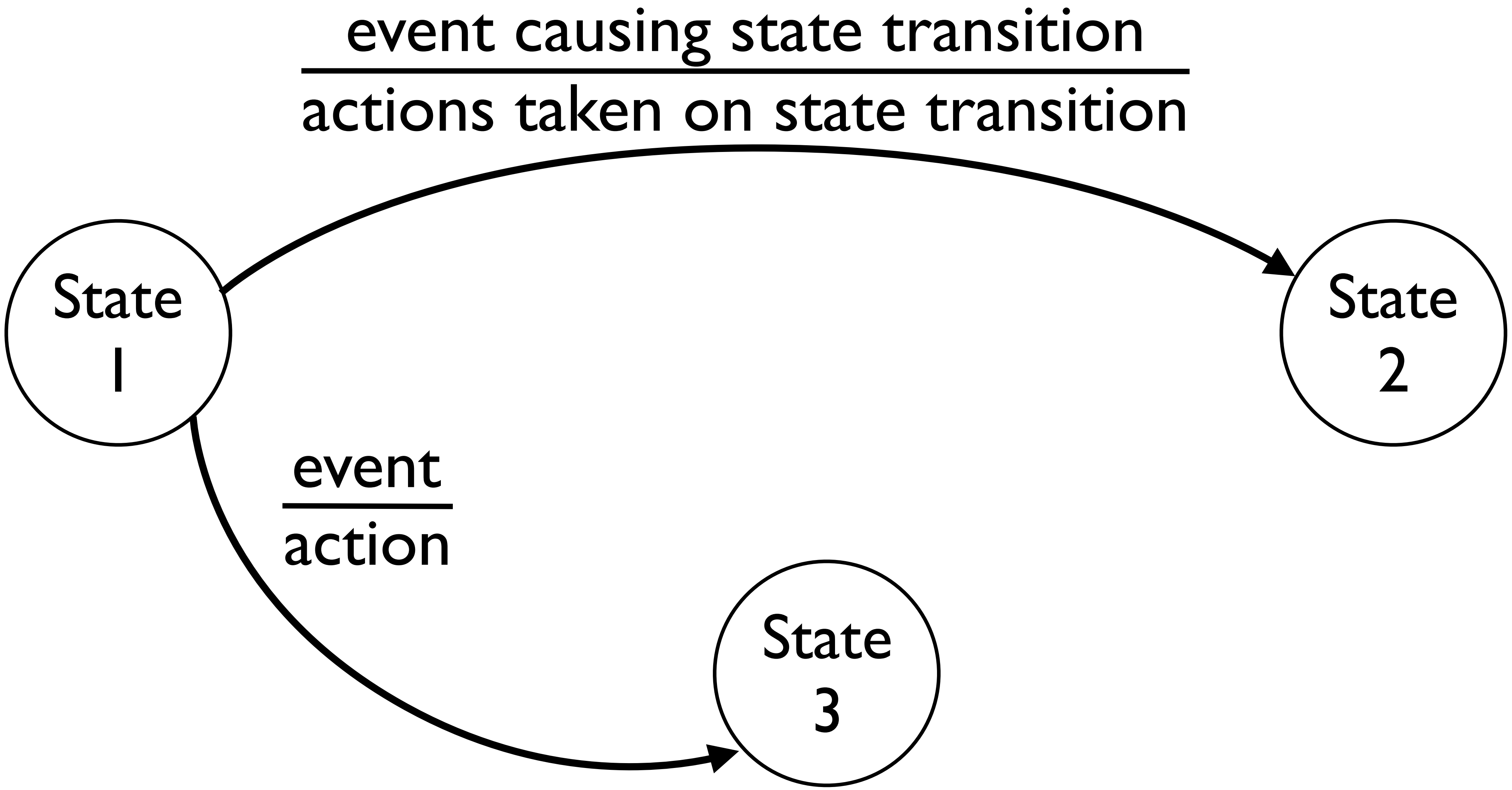
Now what?

How do we send data?

Flow Control

- Don’t send more packets than receiver can process
- Receiver gives sender feedback
- Two basic approaches
  - Stop and wait (lab 1)
  - Sliding window (lab 2)

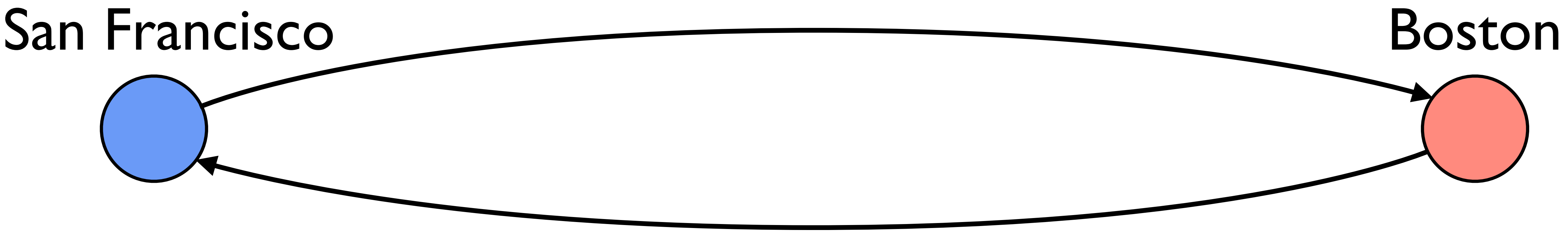
Finite State Machines


event causing state transition
actions taken on state transition
State 1
State 2
event
action
State 3

Stop and Wait

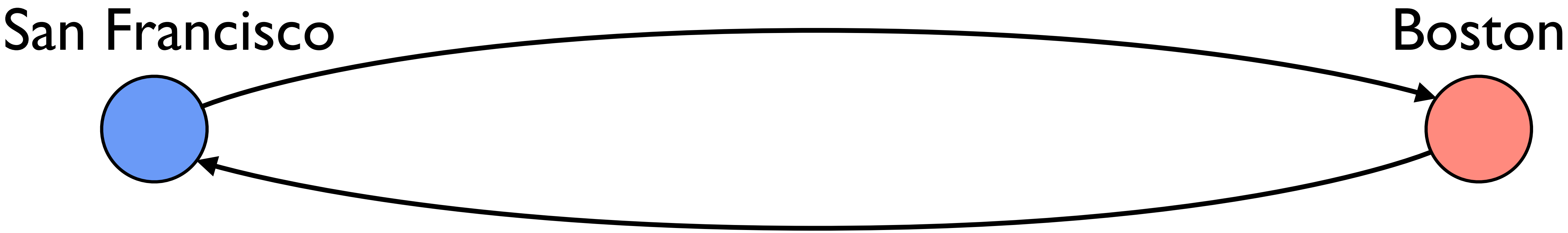
- At most one packet in flight at any time
- Sender sends one packet
- Receiver sends acknowledgment packet when it receives data
- On receiving acknowledgment, sender sends new data
- On timeout, sender resends current data

Stop and Wait Problem



Bottleneck is 10Mbps
RTT is 100ms

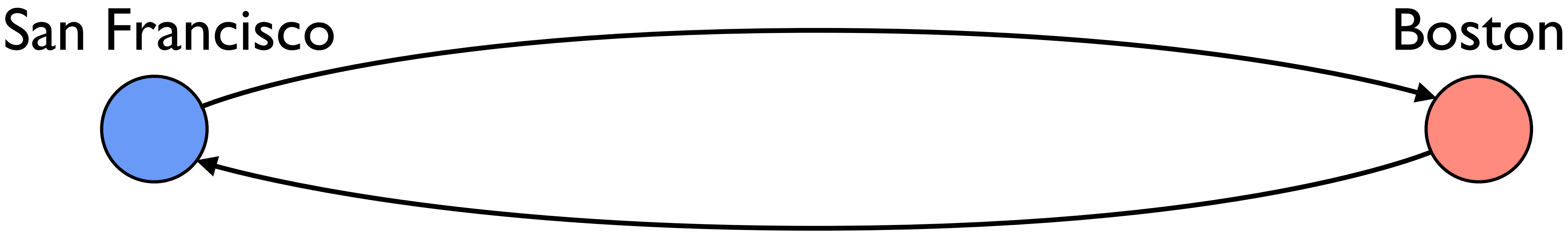
Stop and Wait Problem



Bottleneck is 10Mbps
RTT is 100ms

At most 10 packets/second!
120Kbps << 10Mbps

Sliding Window



Bottleneck is 10Mbps
RTT is 50ms

- Generalization of stop-and-wait: allow multiple un-acked segments
- Bound on number of un-acked segments, called *window*
- Can keep pipe full

Sliding Window Sender

- Every segment has a sequence number (SeqNo)
- Maintain 3 variables
  - Send window size (SWS)
  - Last acknowledgment received (LAR)
  - Last segment sent (LSS)
- Maintain invariant: (LSS - LAR) ≤ SWS
- Advance LAR on new acknowledgment
- Buffer up to SWS segments

Sliding Window Receiver

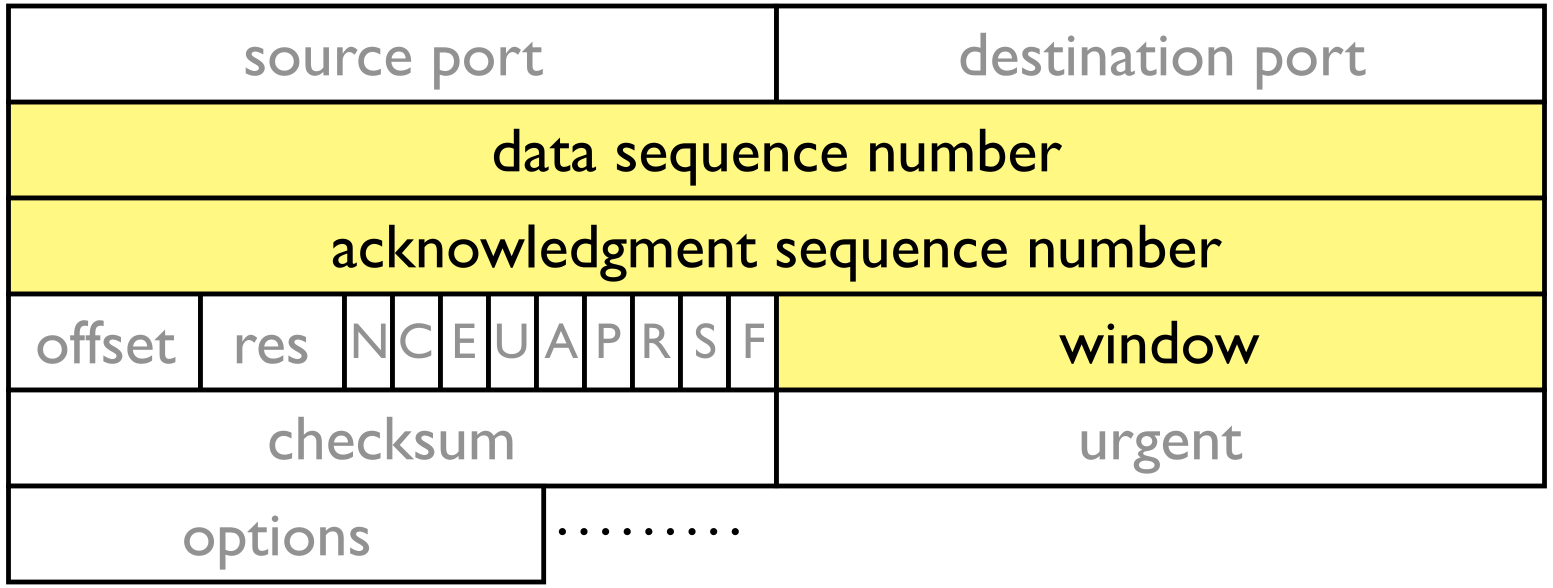
- Maintain 3 variables
  - Receive window size (RWS)
  - Last acceptable segment (LAS)
  - Last segment received (LSR)
- Maintain invariant: (LAS - LSR) $\leq$ RWS
- If received packet is < LAS, send acknowledgment
  - Send *cumulative* acks: if received 1, 2, 3, 5, acknowledge 3
  - NOTE: TCP acks are next *expected* data (e.g., ack 4 in above example)

RWS, SWS, and Sequence Space

- RWS ≥ 1, SWS ≥ 1, RWS ≤ SWS
- Assuming packets not more than 2 RTTs:
  - If RWS = 1, "go back N" protocol, need SWS+1 sequence numbers
  - If RWS = SWS, need 2SWS sequence numbers
- Generally need RWS+SWS sequence numbers per 2 RTTs of delay

TCP Flow Control

- Receiver advertises RWS using window field
- Sender can only send data up to LAR + window



source port | destination port

data sequence number

acknowledgment sequence number

offset | res | N | C | E | U | A | P | R | S | F | window

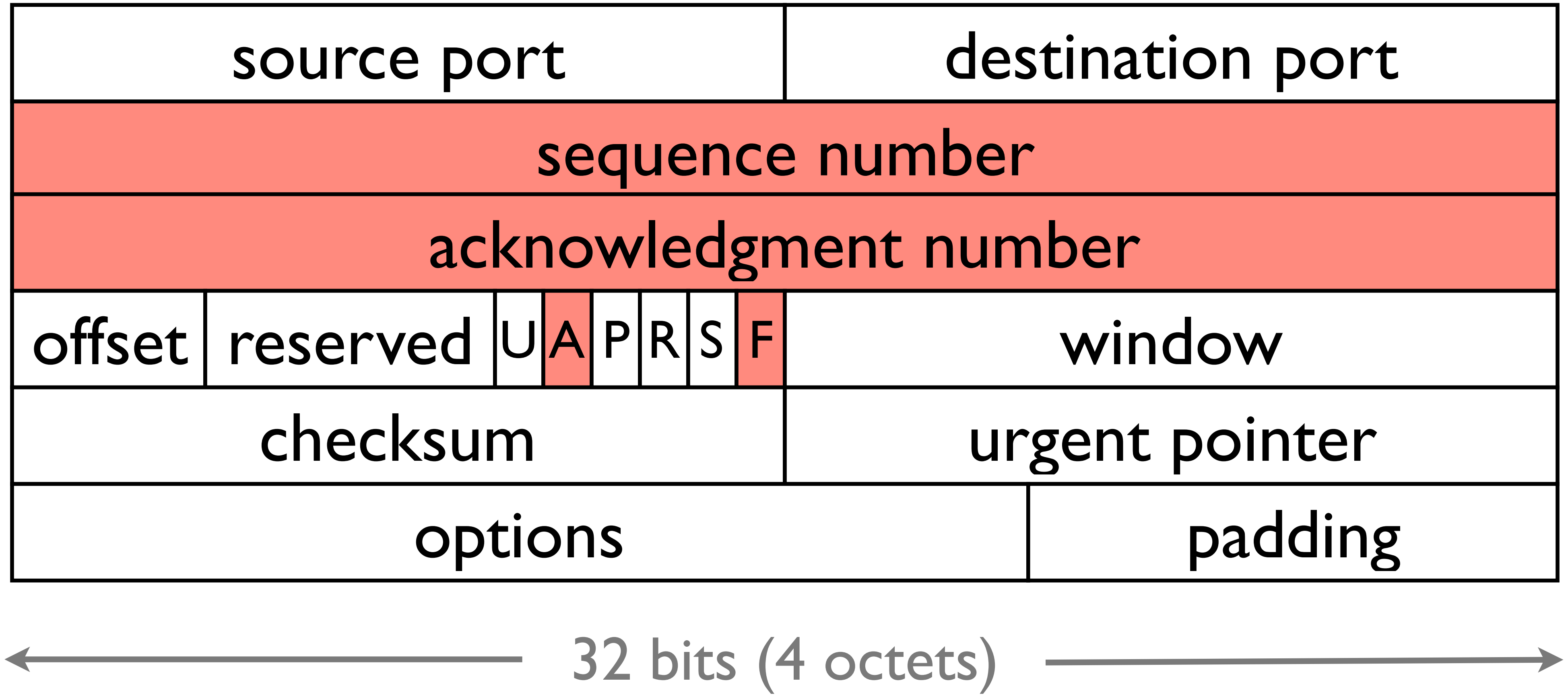
checksum | urgent

options | ....

Sequence numbers

- TCP sequence numbers are in bytes: specifies where in the stream the data in this particular segment resides
    - Denotes state of forward stream, from source to destination of packet
    - Sequence number 2,032, length 800 is bytes 2032–2831
    - Sequence number 123,400, length 1200 is bytes 123,400–124,599
- Acknowledgement number specifies state of stream in reverse direction
    - Cumulative acknowledgements: specifies the first byte of the stream that hasn't been received
    - If stream from A to B started at sequence number 5,000, acknowledgement 15,201 sent from B means that B has received bytes 5,000–15,200 successfully

Connection Teardown



source port | destination port

sequence number

acknowledgment number

offset | reserved | U | A | P | R | S | F | window

checksum | urgent pointer

options | padding

32 bits (4 octets)

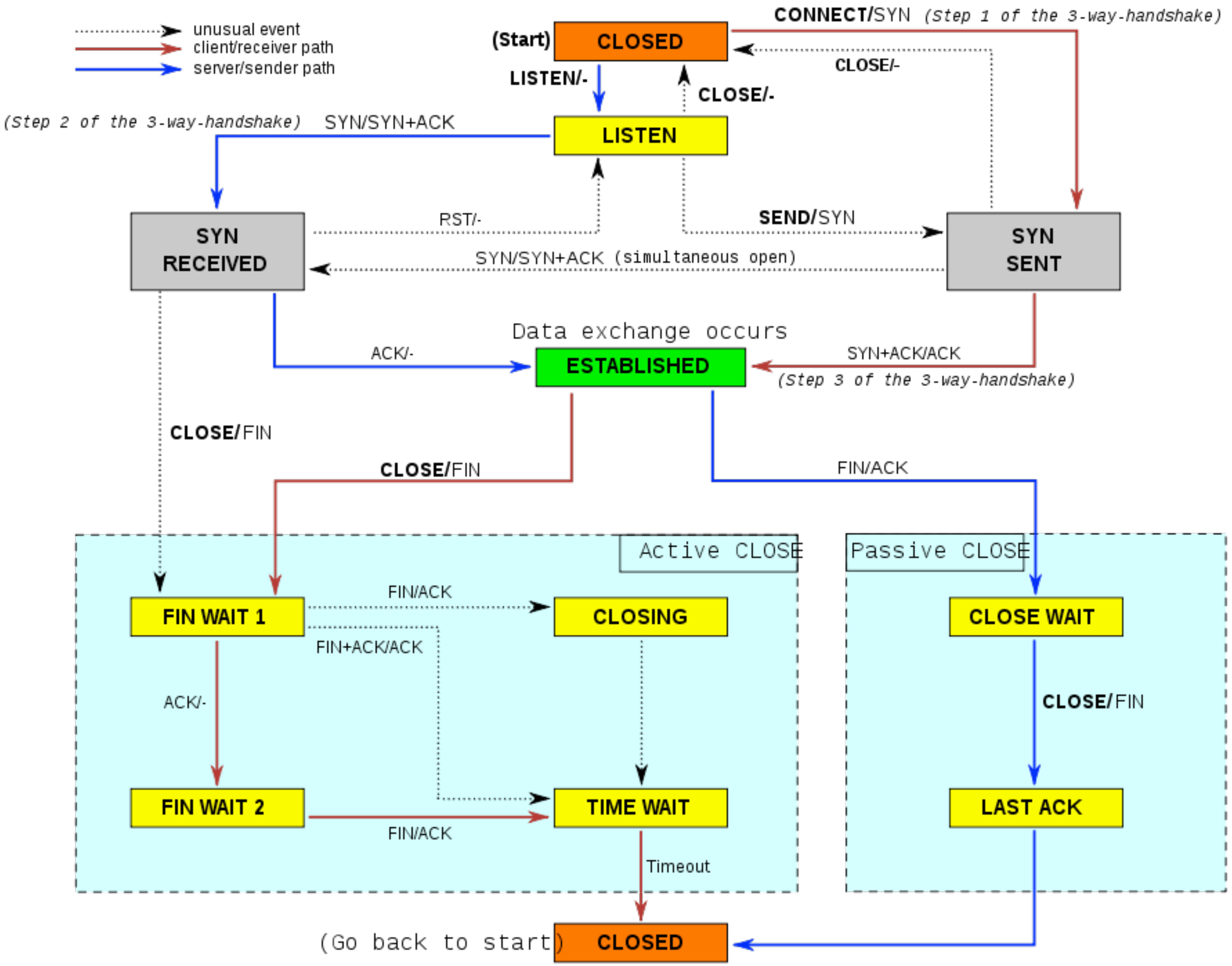
Connection Teardown

- FIN bit says no more data to send
  - Caused by close() or shutdown() on other end
- Both sides must send FIN to terminate a connection
- Typical teardown exchange:
  - A → B: FIN, seq S_A, ack S_B
  - B → A: ack S_{A+1}
  - B → A: FIN, seq S_B, ack S_{A+1}
  - A → B: ack S_{B+1}
- Can also have simultaneous close
- Can A and B forget about closed socket after final message?

Cleaning Up Safely

- Problems with closed socket
  - What if final ack is lost in the network?
  - What if the same port pair is immediately reused for a new connection?
- Solution: "active" closer goes into TIME WAIT
  - Active close is sending FIN before receiving one
  - Keep socket around for 2MSL (twice the "maximum segment lifetime")
- Can pose problems with servers
  - OS has too many sockets in TIME WAIT, slows things down
  - Hack: Can send RST and delete socket, set SO_LINGER socket option to time 0
  - OS won't let you re-start server because port still in use (SO_REUSEADDR option lets you re-bind used port number)

Full TCP FSM



http://en.wikipedia.org/wiki/File:Tcp_state_diagram_fixed.svg

Transport

- Provides inter-program communication
  - ICMP: control messages to operating system
  - UDP: unreliable datagrams to user programs
  - TCP: reliable stream to user programs
- Evidenced by *naming*
  - IP packets are addressed to hosts with *addresses*
  - UDP and TCP segments are named to programs with *ports*
  - ICMP is implicitly named to operating system/IP software

Transport Abstractions

- ICMP: unreliable datagrams, control messages between IP software
- UDP: unreliable datagrams, application data
- TCP: reliable stream, application data
  - Need to establish connections: 3-way handshake
  - Data transfer: stop and wait
  - Data transfer: sliding window
    - Receiver states current window size
    - Sender can have up to window size unacknowledged bytes in flight
  - Connection teardown