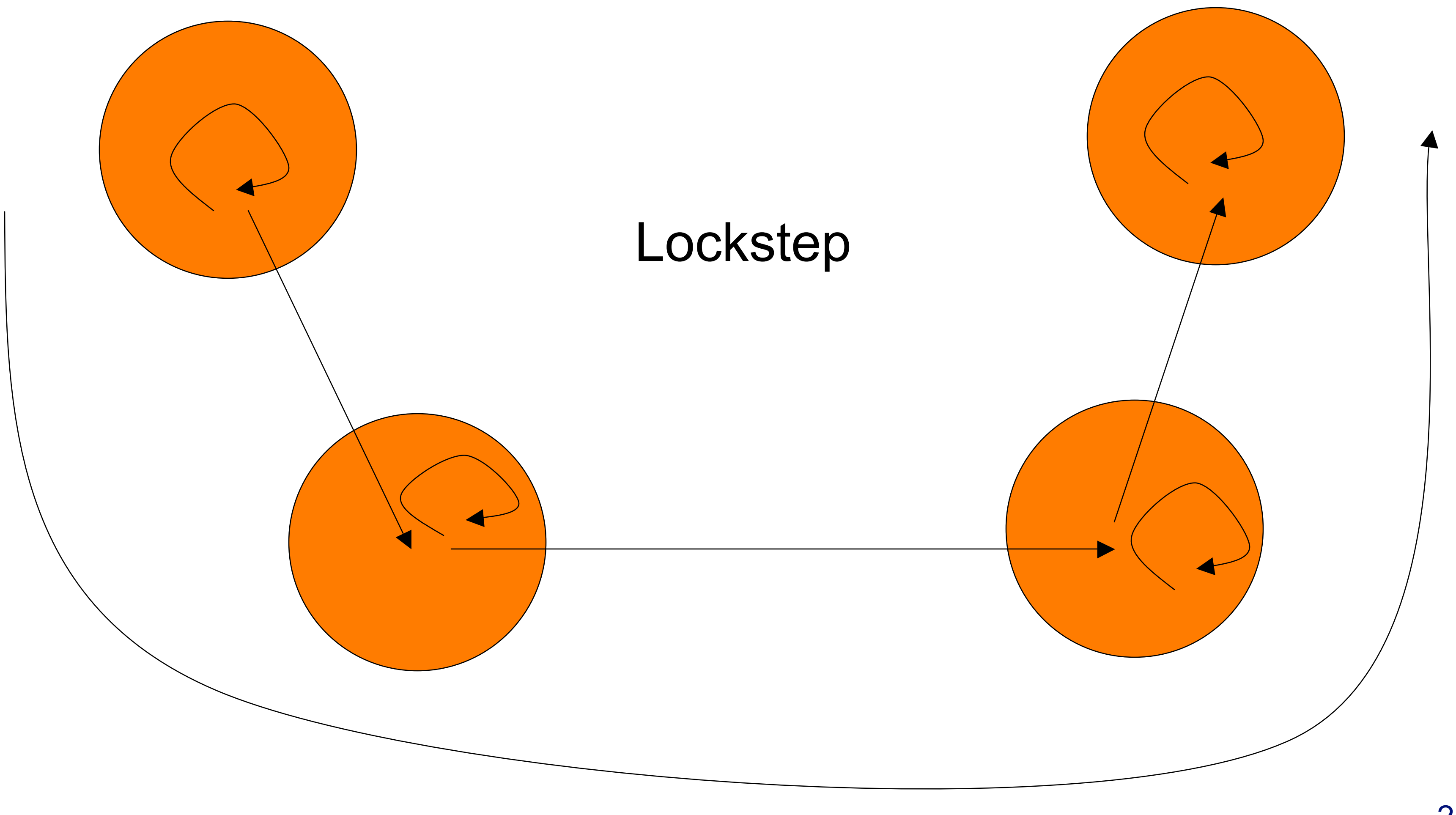
CSE/EE 461

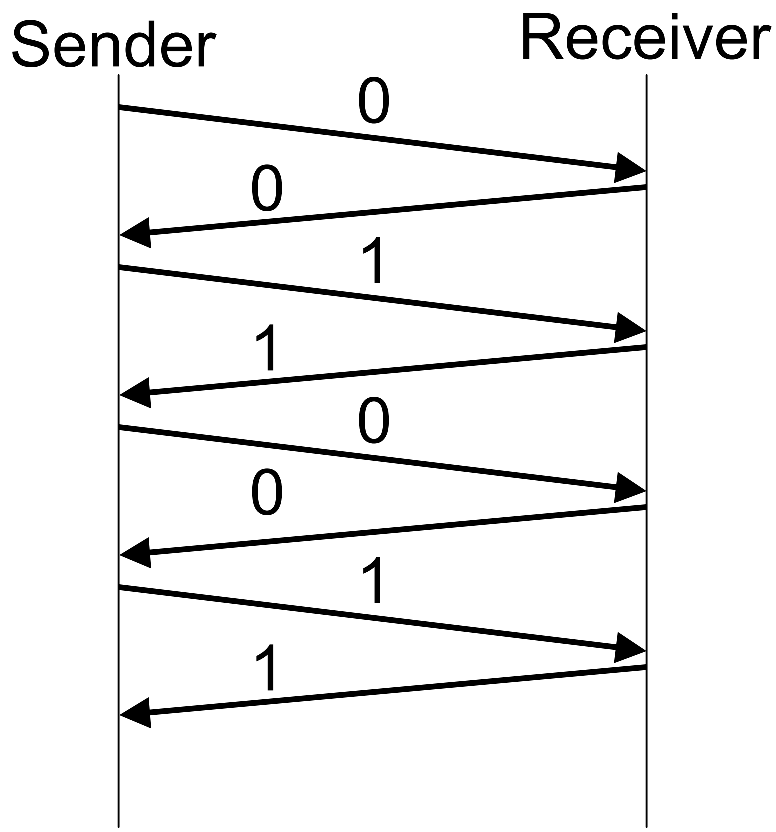
Sliding Windows and ARQ

Slowly Spinning Wheels



Stop-and-Wait

- Only one outstanding packet at a time
- Also called alternating bit protocol
- Reliability
- Flow Control

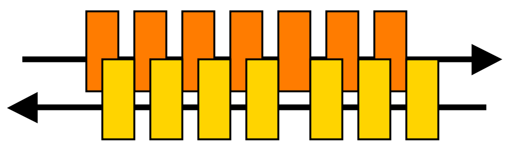


Limitation of Stop-and-Wait

- Lousy performance if wire time << prop. delay
  - Max BW: B
  - Actual BW: M/2D
    - Example: B = 100Mb/s, M=1500Bytes, D=50ms
    - Actual BW = 1500Bytes/100ms --> 15000 Bytes/s --> 100Kb/s
    - 100Mb vs 100Kb?

More BW Please

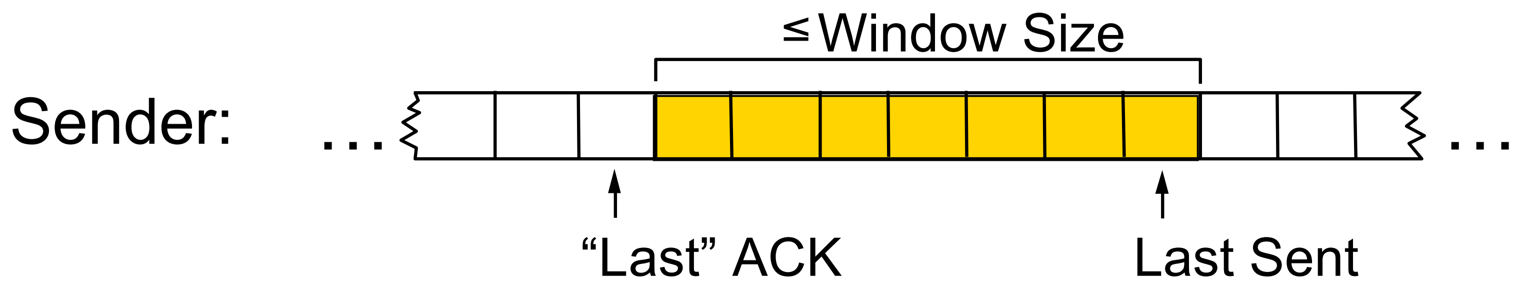
- Want to utilize all available bandwidth
  - Need to keep more data "in flight"
  - How much? Remember the bandwidth-delay product?
- Leads to Sliding Window Protocol
- Window size says how much data can be sent without waiting for an acknowledgement



Sliding Window

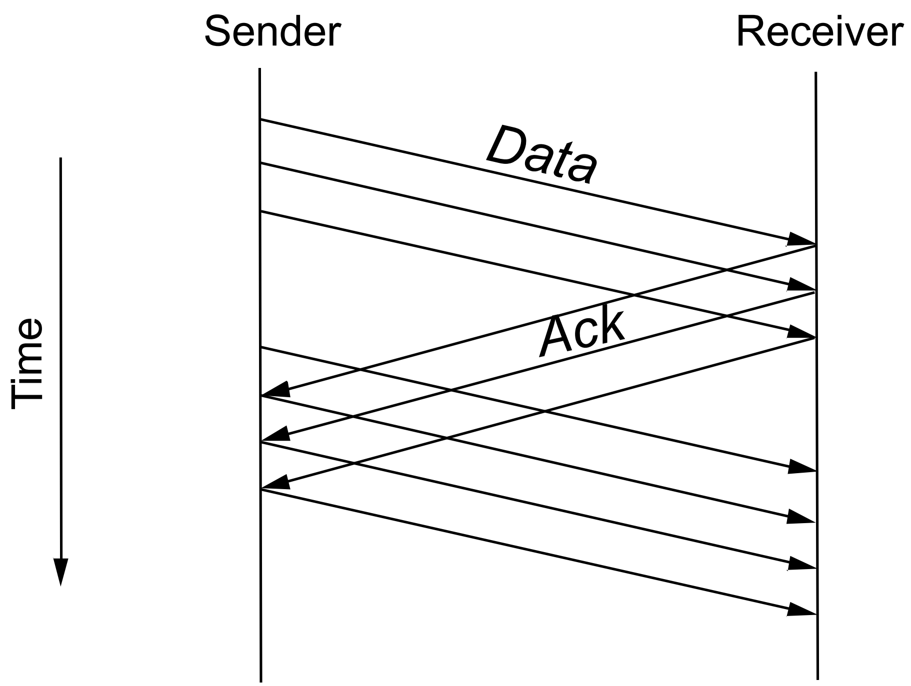
- Sender can send a lot of data before waiting for an ack
  - Amount of data is the window size
    - #pkts, or #bytes, depending
- Sender tries not to send more data than the receiver can handle
  - Window size - sizeof(unacknowledged data)
- It supports multiple functions:
  - Reliable delivery
    - *If I hear you got it, I know you got it.*
    - ACK (Ack # is "next byte expected")
  - In-order delivery
    - *If you get it, you get it in the right order.*
    - SEQ # (Seq # is "the byte this is in the sequence")
  - Flow control
    - *If you don't have room for it, I won't send it.*

Sliding Window – Sender



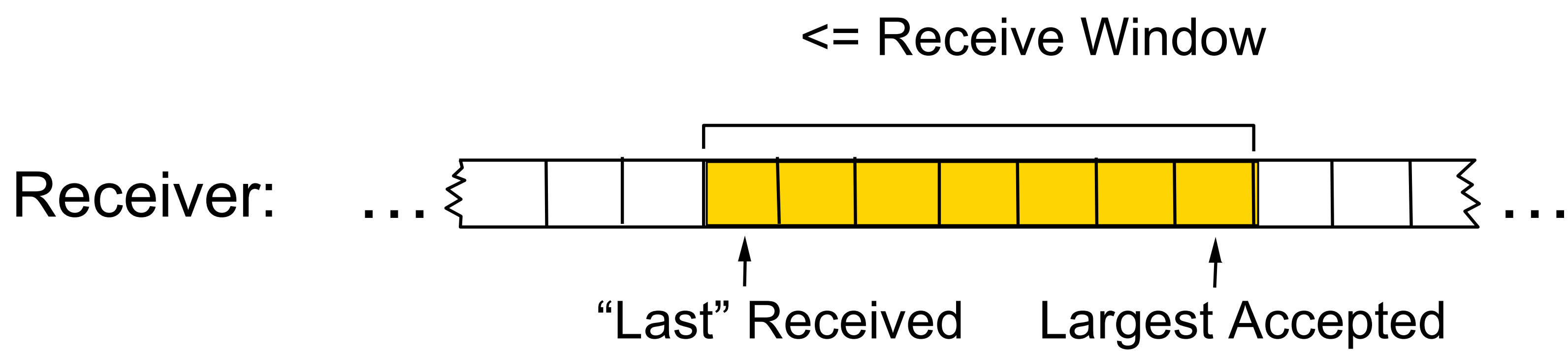
- Window bounds outstanding data
  - Implies need for buffering at sender
    - Specifically, must buffer unacked data
- "Last" ACK applies to in-order data
  - Need not buffer acked data
- Sender maintains timers
  - Go-Back-N: one timer, send all unacknowledged on timeout
  - Selective Repeat: timer per packet, resend as needed

Sliding Window – Timeline



- Receiver ACK design choices:
  - –Individual
    - Each packet acked
  - –Cumulative (TCP)
    - Ack says "got everything up to X-1…"
    - really, "my ack means that the next seq# I am expecting is X"
  - –Selective (newer TCP)
    - Acks says "I got X through Y"
    - Easier to keep pipe full with unreceived data
    - More complex
  - – Negative
    - Acks says "I did not get X"
    - Decrease rexmit time

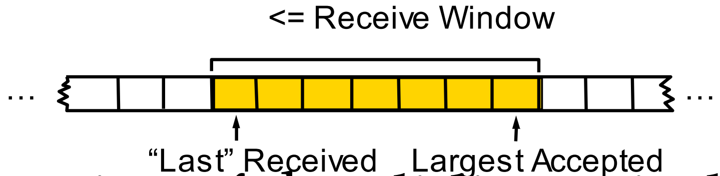
Sliding Window – Receiver



- Receiver buffers too:
  - data may arrive out-of-order
  - or faster than can be consumed by receiving application
    - Drop?
- No sense having more data on the wire than can be buffered at the receiver.
  - In other words, current receiver buffer size limits the window size

Flow Control

- Sender must transmit data no faster than it can be consumed by the receiver
  - Receiver might be a slow machine
  - App might consume data slowly

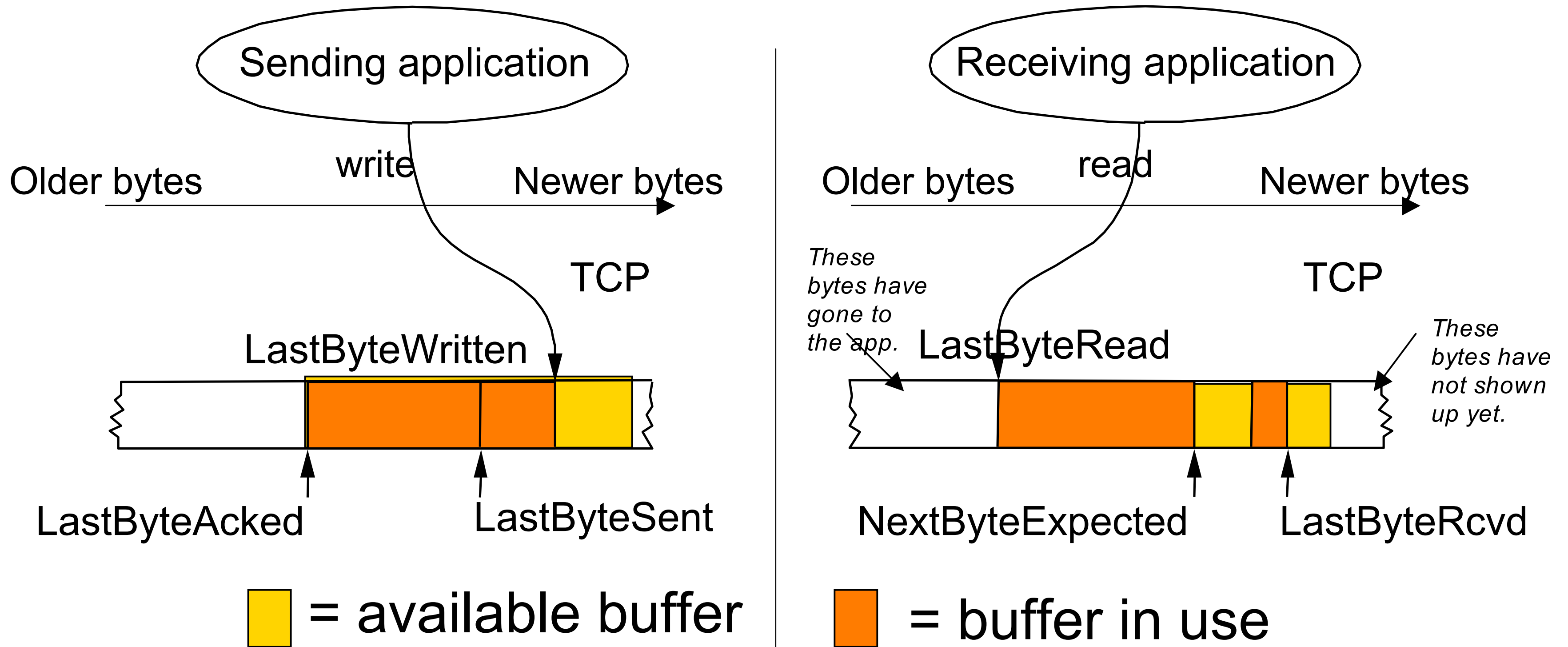


- Implement by adjusting the size of the sliding window used at the sender based on receiver feedback about available buffer space
  - Receiver "advertises" its receive window
  - Piggyback on the ack

One more thing…

- Decouple sending application from sending protocol
- Sender needs to buffer messages anyway in order to resend
- Each side maintains some local and remote buffer state and invariants
- Local buffer state is correct
- Remote buffer state is conservative
- Sending, receiving, reading and writing are allowed to perform according to the states of the buffers and the invariants
  - **Invariants --> *allowed behavior***

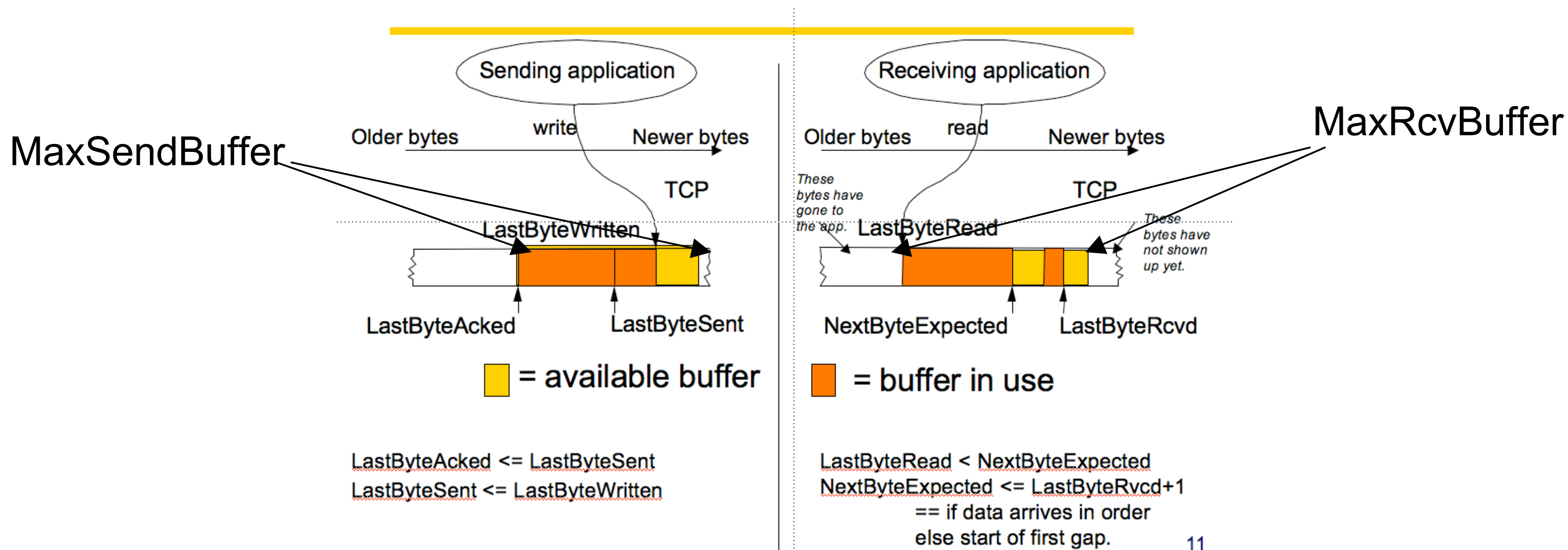
Sender and Receiver Buffering



LastByteAcked <= LastByteSent
LastByteSent <= LastByteWritten

LastByteRead < NextByteExpected
NextByteExpected <= LastByteRcvd+1
== if data arrives in order
else start of first gap.

Flow Control



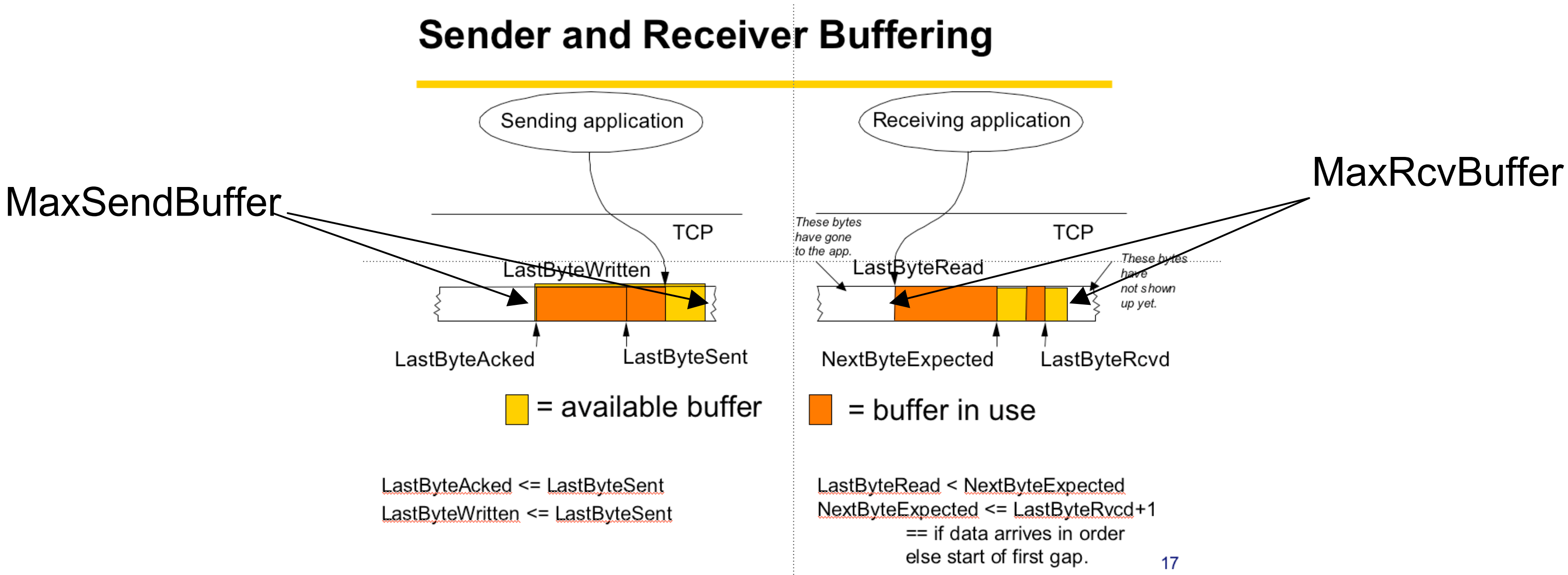
LastByteRcvd - LastByteRead <= MaxRcvBuffer

AdvertisedWindow = MaxRcvBuffer - ((NextByteExpected -1) - LastByteRead

"All the buffer space minus the buffer space that's in use."

As data arrives, receiver acknowledges it so long as all preceding bytes have also arrived. Advertised Window potentially shrinks depending on how fast receiving app is drawing out Data.

Flow Control On the Sender

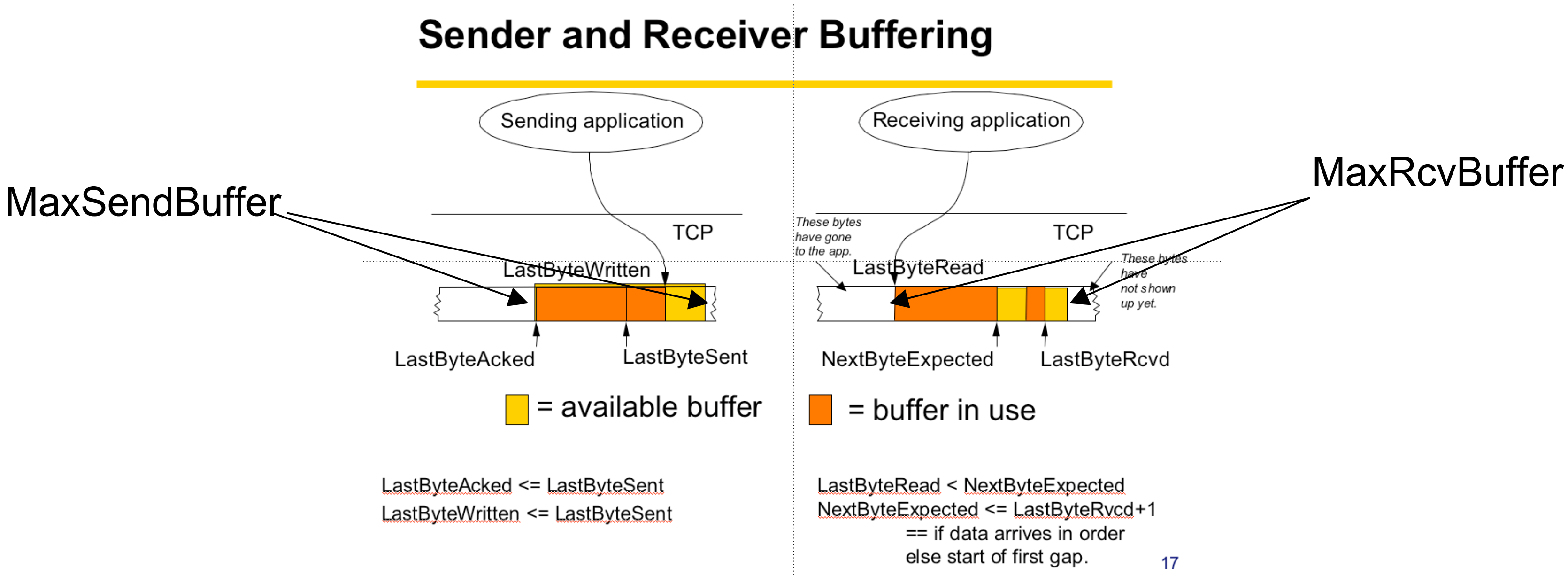


LastByteSent - LastByteAcked <= AdvertisedWindow *'don't send that which is unwanted.'*

EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)

OK to send that which there is room for, which is that which was advertised minus that which I've already sent since receiving the last advertisement.

Sending Side -- One last detail



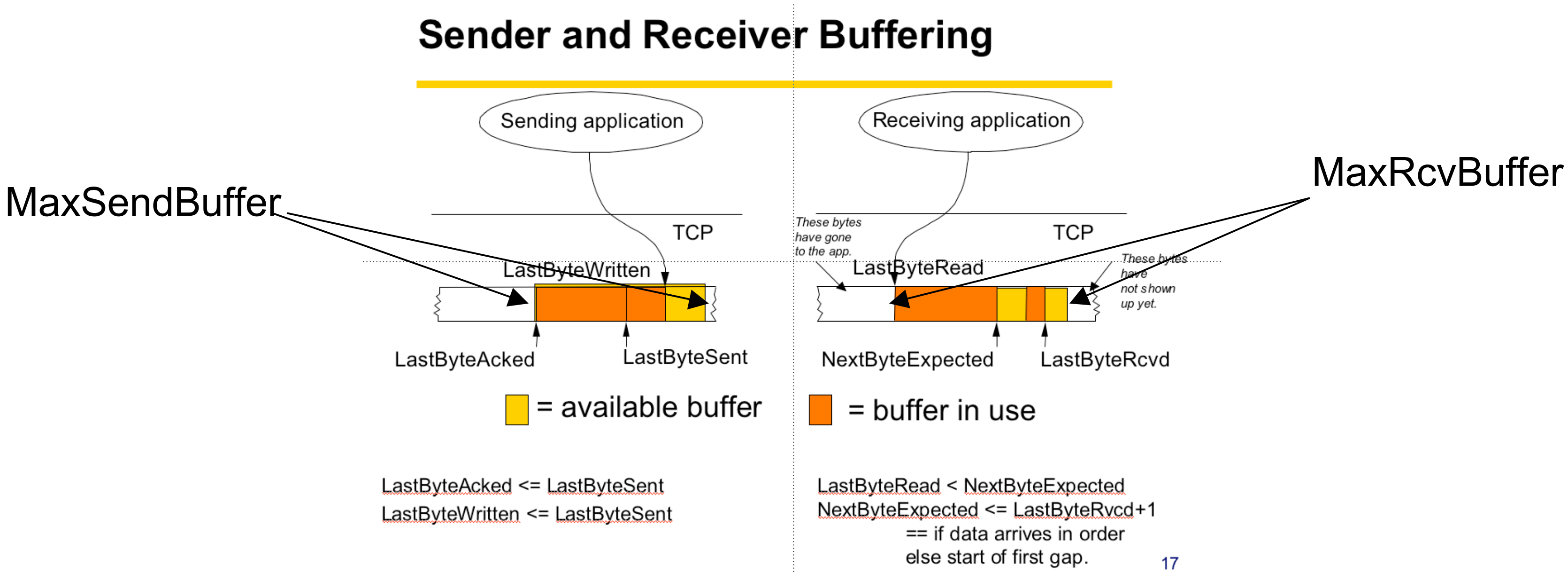
LastByteWritten - LastByteAcked <= MaxSendBuffer

Can only hang on to unsent and unacked data if there's room for it.

==> *BLOCK write(y) if*

(LastByteWritten - LastByteAcked) + y > MaxSendBuffer

Receiving Side -- One last detail



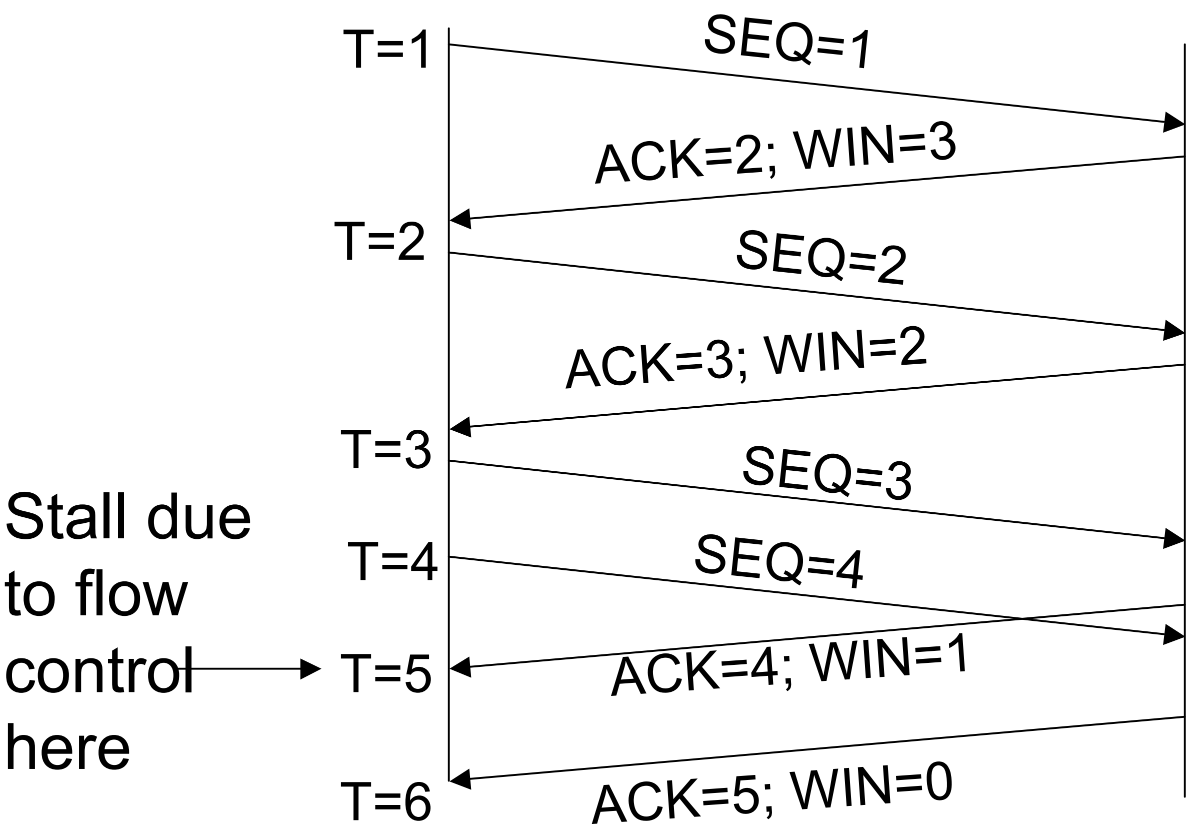
LastByteRead < NextByteExpected

Can't read data if it hasn't arrived.

==> *BLOCK read(y) if*

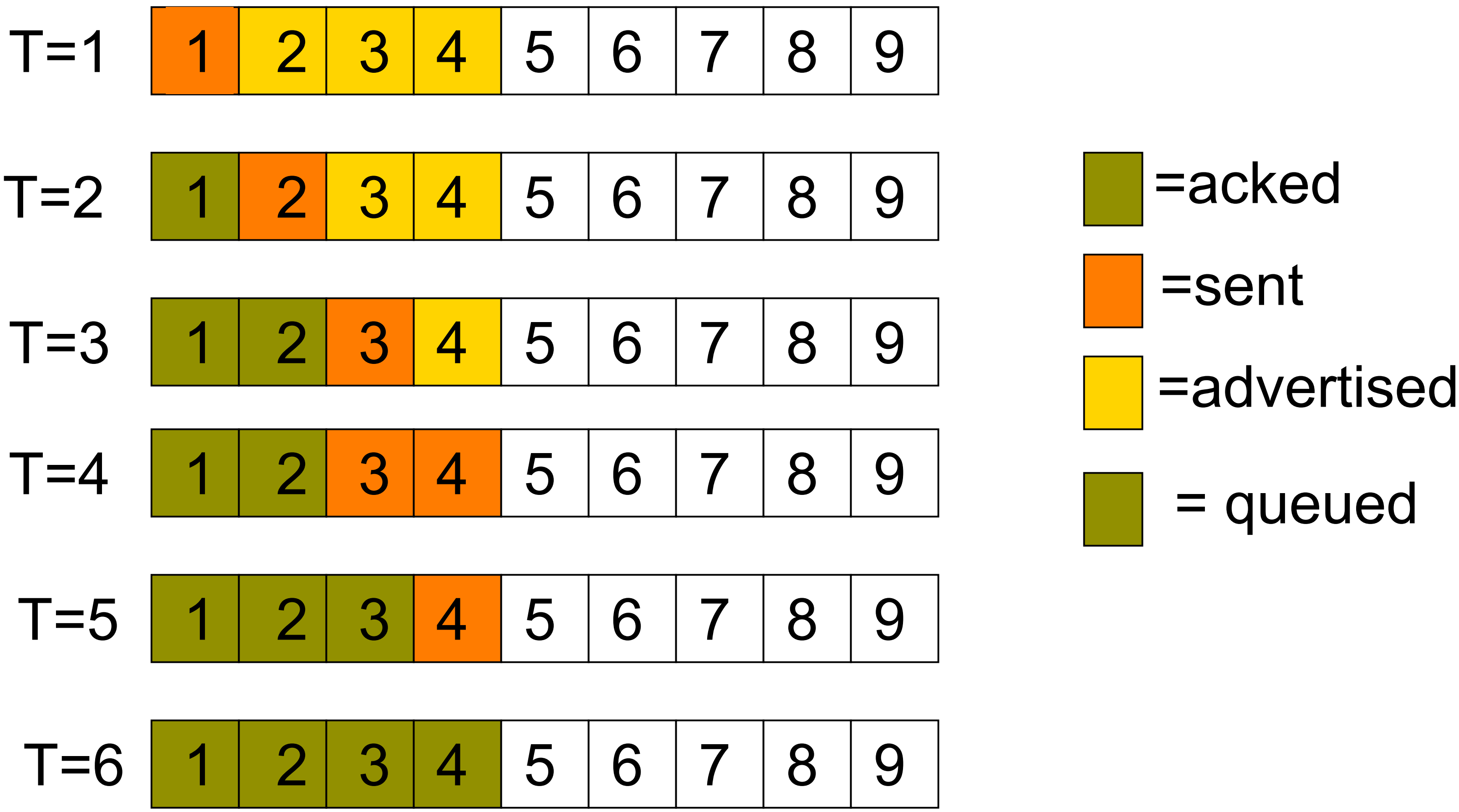
LastByteRead < NextByteExpected

Example – Exchange of Packets

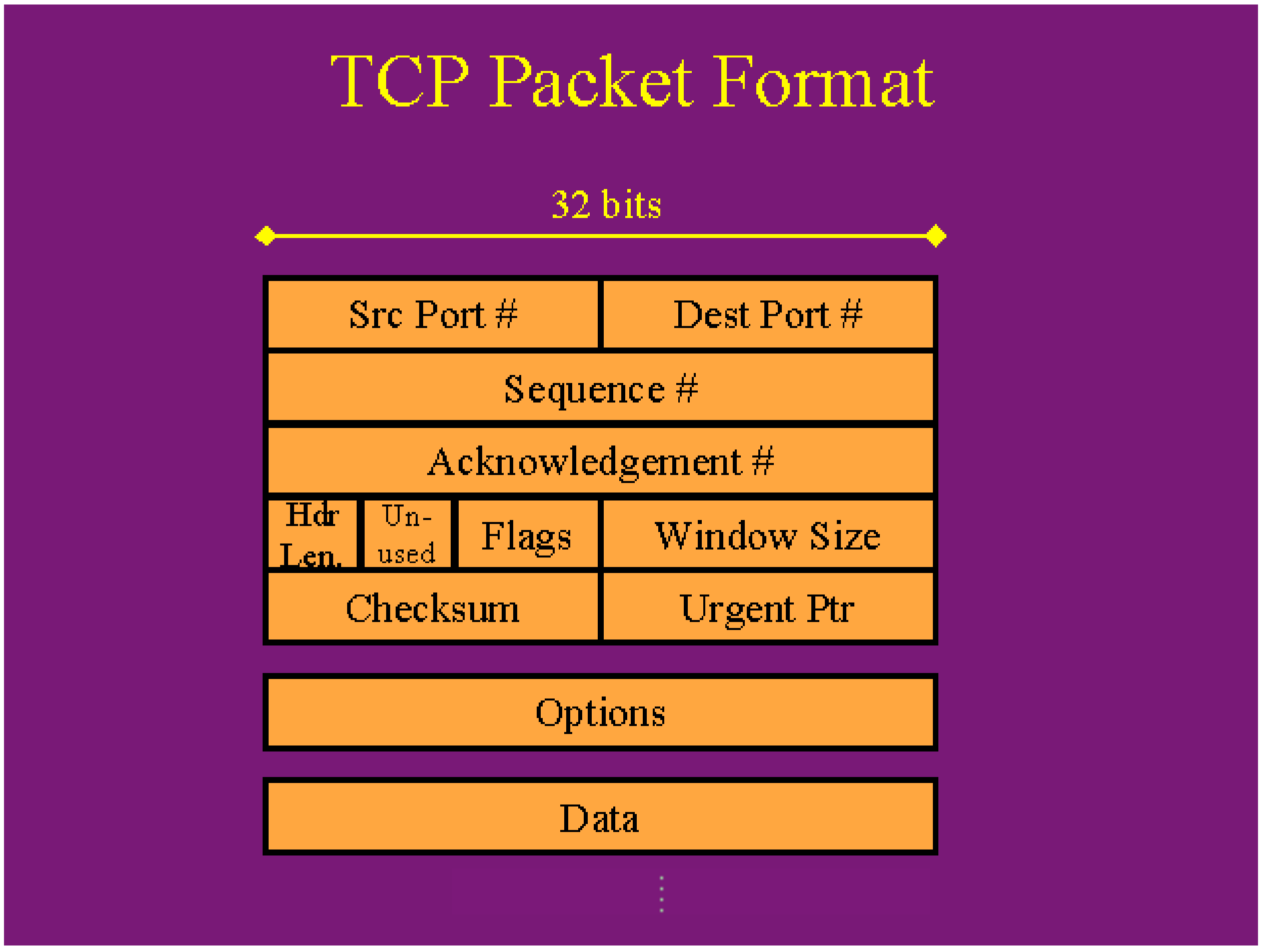


Receiver has buffer of size 4 and application doesn't read

Example – Buffer at Sender



TCP Packet Format



16 bit window size gets
Cramped with large
Bandwidth x delay

16 bits --> 64K
BD ethernet: 122KB
STS24 (1.2Gb/s): 14.8MB

32 bit sequence number
must not wrap around faster
than the maximum packet
lifetime. (120 seconds)
-- 622Mb/s link: 55 seconds

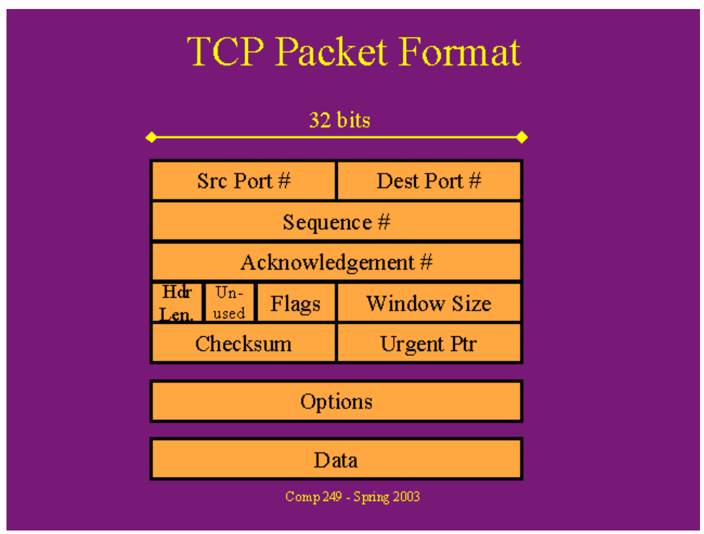
What to do?

The IP Packet

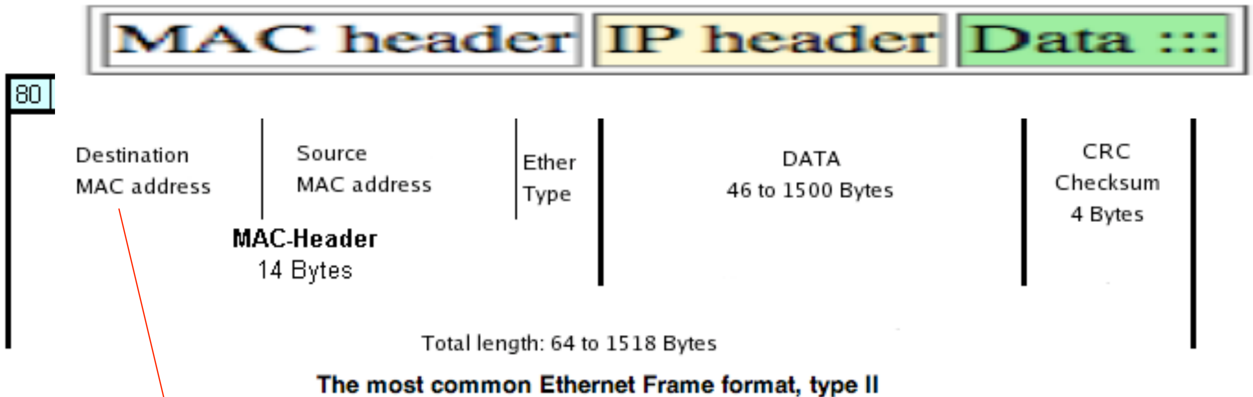
MAC header	IP header	Data :::

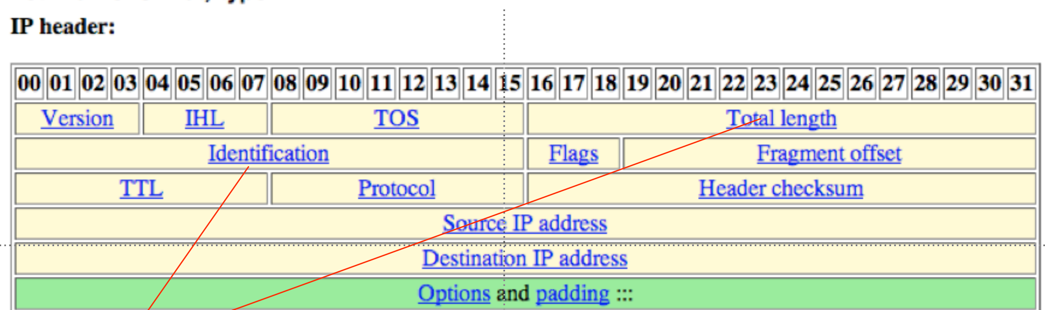
IP header:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Version				IHL				TOS								Total length															
Identification																Flags			Fragment offset												
TTL								Protocol								Header checksum															
Source IP address																															
Destination IP address																															
Options and padding :::																															

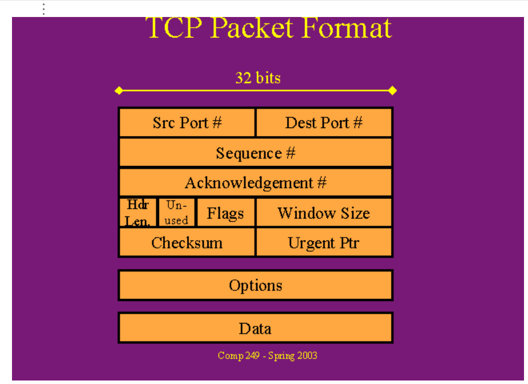


An Entire Ether Packet






```
[bigmac:Web461/HW/HW3] bershad% sudo tcpdump -xx host www.cs.washington.edu
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en0, link-type EN10MB (Ethernet), capture size 96 bytes
08:16:03.030582 IP 10.0.1.4.63604 > www.cs.washington.edu.http: S 1795021890:179
>
        0x0000:  0003 93e7 3f30 000a 95c9 340a 0800 4500  ....?0....4...E.
        0x0010:  003c 35e4 4000 4006 75ac 0a00 0104 80d0  .<5.@.@.u.......
        0x0020:  0358 f874 0050 6afd dc42 0000 0000 a002  .X.t.Pj..B......
        0x0030:  ffff 5b88 0000 0204 05b4 0103 0300 0101  ..[.............
        0x0040:  080a 6a78 b5d6 0000 0000                 ..jx......
```



EtherTypes

Links: Ethernet assigned numbers.

Ethertype	Protocol
0x0000 - 0x05DC	IEEE 802.3 length.
0x0600	XEROX NS IDP.
0x0660 0x0661	DLOG.
0x0800	IP, Internet Protocol.
0x0801	X.75 Internet.
0x0802	NBS Internet.
0x0803	ECMA Internet.
0x0804	Chaosnet.
0x0805	X.25 Level 3.
0x0806	ARP, Address Resolution Protocol.
0x8035	DRARP, Dynamic RARP. RARP, Reverse Address Resolution Protocol.
0x80F3	AARP, AppleTalk Address Resolution Protocol.
0x8100	EAPS, Ethernet Automatic Protection Switching.
0x8137	IPX, Internet Packet Exchange.
0x814C	SNMP, Simple Network Management Protocol.
0x86DD	IPv6, Internet Protocol version 6.
0x880B	PPP, Point-to-Point Protocol.
0x880C	GSMP, General Switch Management Protocol.
0x8847	MPLS, Multi-Protocol Label Switching (unicast).
0x8848	MPLS, Multi-Protocol Label Switching (multicast).
0x8863	PPPoE, PPP Over Ethernet (Discovery Stage).
0x8864	PPPoE, PPP Over Ethernet (PPP Session Stage).
0x88BB	LWAPP, Light Weight Access Point Protocol.
0x88CC	LLDP, Link Layer Discovery Protocol.
0x8E88	EAPOL, EAP over LAN.
0xFFFF	reserved.

IP header:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Version				IHL				TOS								Total length															
Identification																Flags			Fragment offset												
TTL								Protocol								Header checksum															
Source IP address																															
Destination IP address																															
Options and padding :::																															

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en0, link-type EN10MB (Ethernet), capture size 96 bytes
08:16:03.030582 IP 10.0.1.4.63604 > www.cs.washington.edu.http: S 1795021890:1
>
        0x0000:  0003 93e7 3f30 000a 95c9 340a 0800 4500  ....?0....4...E.
        0x0010:  003c 35e4 4000 4006 75ac 0a00 0104 80d0  .<5.@.@.u.......
        0x0020:  0358 f874 0050 6afd dc42 0000 0000 a002  .X.t.Pj..B......
        0x0030:  ffff 5b88 0000 0204 05b4 0103 0300 0101  ..[.............
        0x0040:  080a 6a78 b5d6 0000 0000                 ..jx......
```

Key Concepts

- Transport layer allows processes to communicate with stronger guarantees, e.g., reliability
- Basic reliability is provided by ARQ mechanisms
  - Stop-and-Wait through Sliding Window plus retransmissions