CS 352
Ordered Delivery; Flow Control

Lecture 14

http://www.cs.rutgers.edu/~sn624/352-F22

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Recap of concepts


```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          Source Port          |       Destination Port        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        Sequence Number                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Acknowledgment Number                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Data |           |U|A|P|R|S|F|                               |
   | Offset| Reserved  |R|C|S|S|Y|I|            Window             |
   |       |           |G|K|H|T|N|N|                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           Checksum            |         Urgent Pointer        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Options                    |    Padding    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                             data                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                            TCP Header Format

          Note that one tick mark represents one bit position.
```

All these fields may carry useful info on the same packet. TCP is bidirectional

TCP:

Connection-oriented

ACK pkts after a drop?

No — Go-back-N

Yes — Selective repeat

- Cumulative ACK
  - `ACK x`
- Selective ACK
  - `ACK x`
  - `SACK x1-x2,x3-x4,x5-x6`

Cumulative ACK		Selective ACK
	Precision of info:	
	<	
	Time to recover from loss	
	>	
	Complexity, bugs, …	

Observing a TCP exchange

- `sudo tcpdump -i eno1 tcp portrange 56000-56010`
- `curl --local-port 56000-56010 https://www.google.com > output.html`
- Bonus: Try crafting TCP packets with `scapy`!

Buffering and Ordering in TCP

Memory Buffers at the Transport Layer

Sockets need receive-side memory buffers

- Since TCP uses selective repeat, the receiver must buffer data that is received after loss:
  - e.g., hold packets so that only the “holes” (due to loss) need to be filled in later, without having to retransmit packets that were received successfully

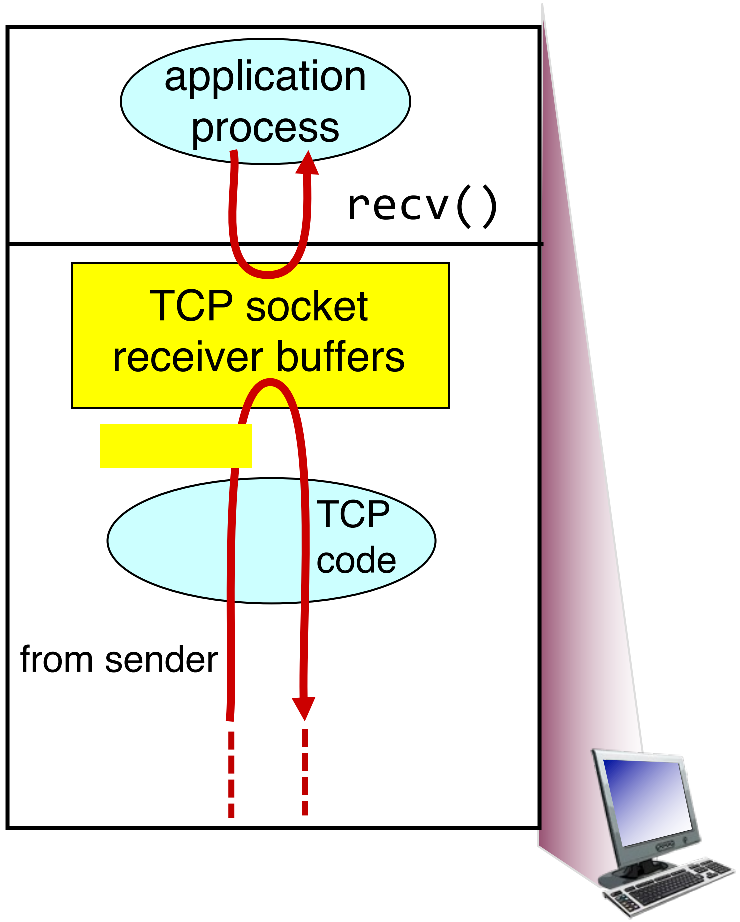
- Apps read from the receive-side socket buffer when you do a `recv()` call.

- Even if data is always reliably received, applications may not always read the data immediately
  - What if you invoked `recv()` in your program infrequently (or never)?
  - For the same reason, UDP sockets also have receive-side buffers

Receiver app’s interaction with TCP

- Upon reception of data, the receiver’s TCP stack deposits the data in the receive-side socket buffer

- An app with a TCP socket reads from the TCP receive socket buffer
  - e.g., when you do `data = sock.recv()`



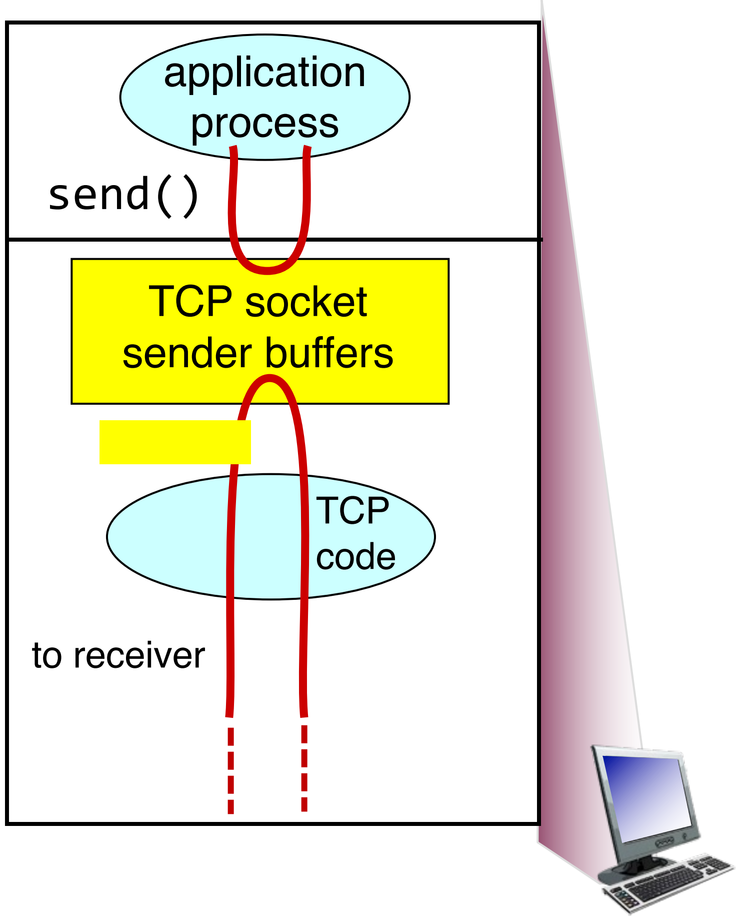
receiver TCP interaction

Sockets need send-side memory buffers

- The possibility of packet retransmission in the future means that data can't be immediately discarded from the sender once transmitted.

- App has issued `send()` and moved on; TCP stack must buffer this data

- Transport layer must wait for ACK of a piece of data before reclaiming (freeing) the memory for that data.



sender TCP interaction

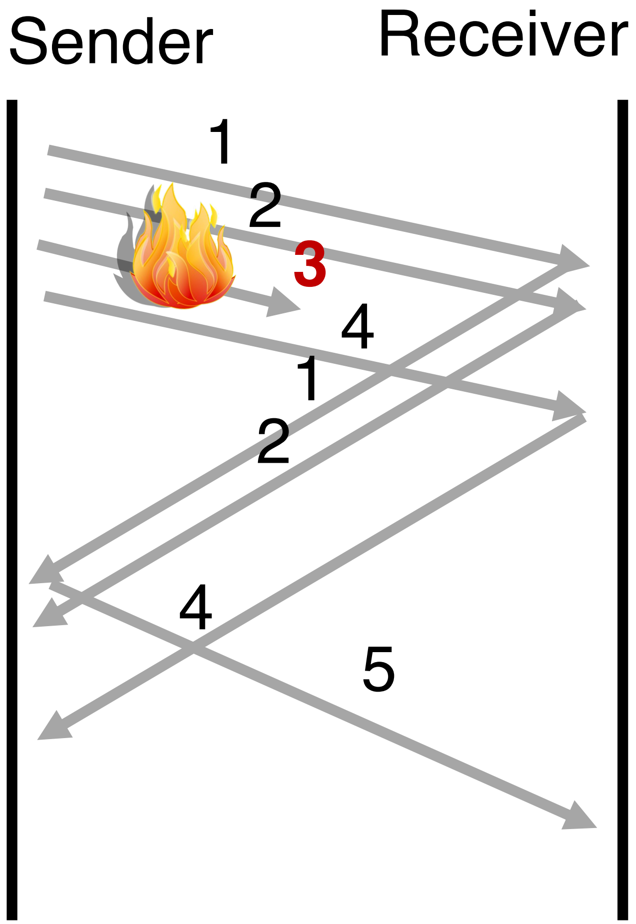
Ordered Delivery

Reordering packets at the receiver side

- Let’s suppose receiver gets packets 1, 2, and 4, but not 3 (dropped)

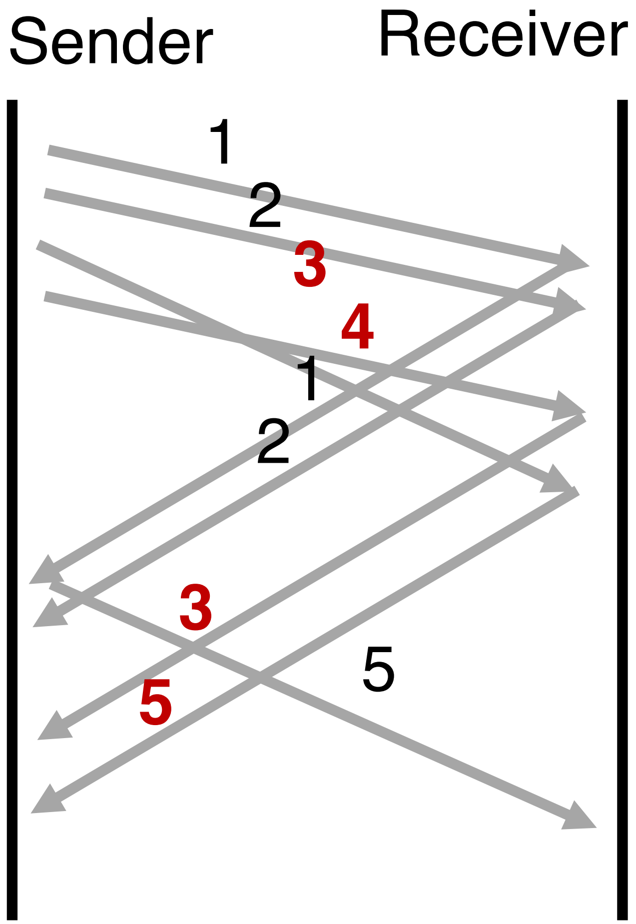
- Suppose you’re trying to download a document containing a report

- What would happen if transport at the receiver directly presents packets 1, 2, and 4 to the application (i.e., receiving 1,2,4 through the `recv()` call)?



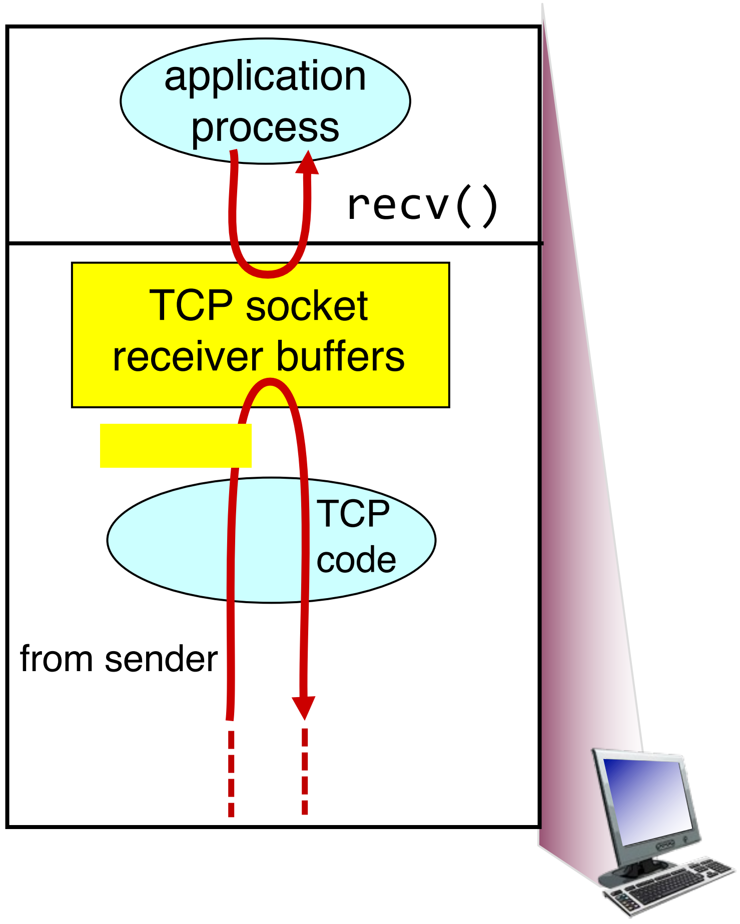
Reordering packets at the receiver side

- Reordering can happen for a few reasons:
  - Drops
  - Packets taking different paths through a network
- Receiver needs a general strategy to ensure that data is presented to the application in the same order that the sender pushed it. Ideas?
- To implement ordered delivery, the receiver uses
  - Sequence numbers
  - Receiver socket buffer
- We've already seen the use of these for reliability; but they can be used to order too!



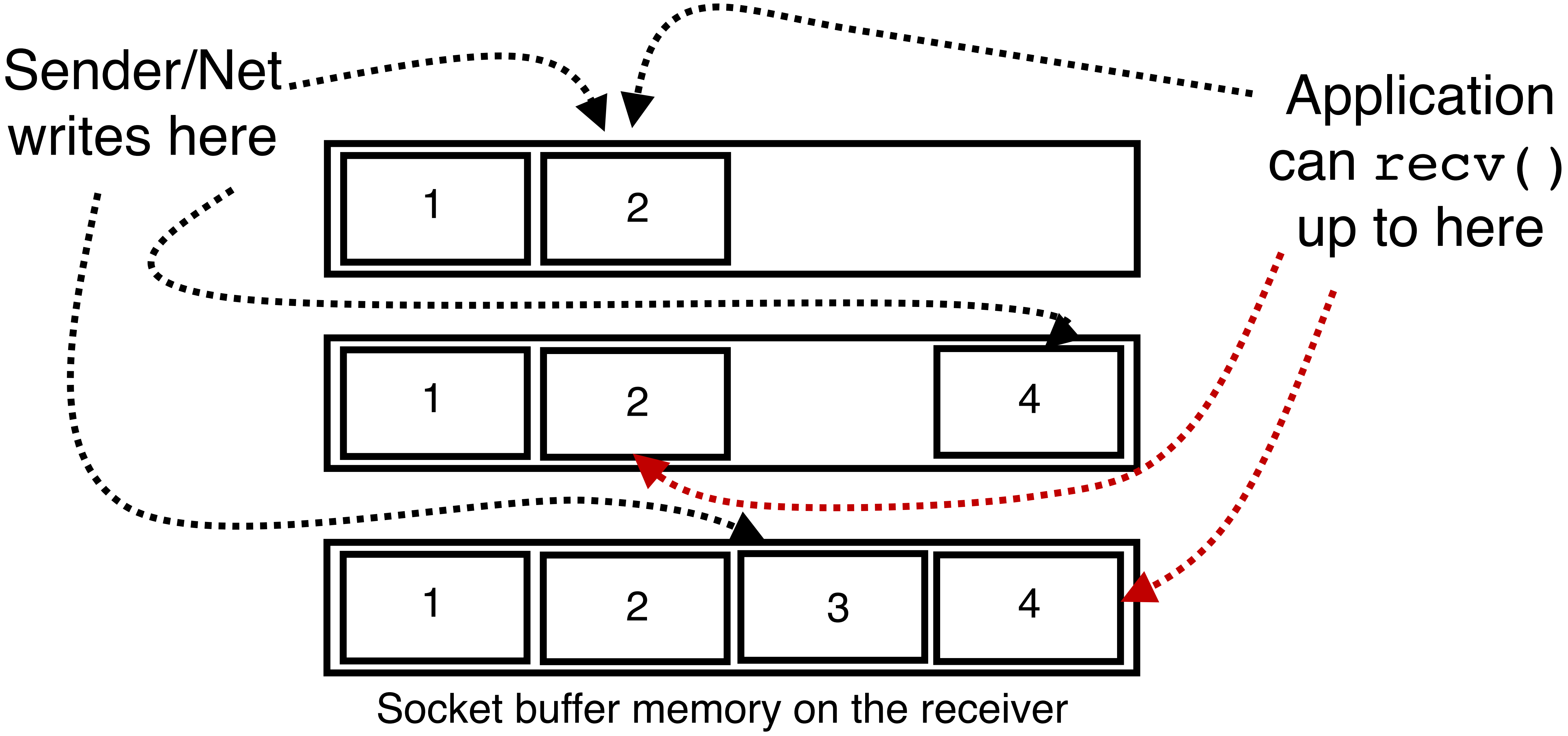
Receive-side app and TCP

- TCP receiver software only releases the data from the receive-side socket buffer to the application if…
    - the data is in order relative to all other data already read by the application
- This process is called TCP reassembly



receiver protocol stack

TCP Reassembly



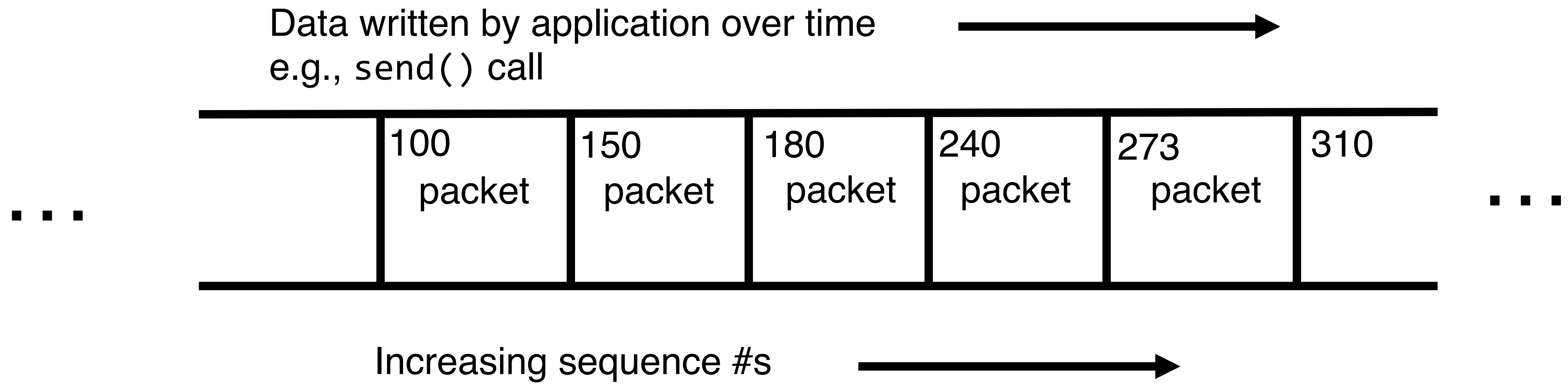
Implications of ordered delivery

- Packets cannot be delivered to the application if there is an in-order packet missing from the receiver's buffer
  - The receiver can only buffer so much out-of-order data
  - Subsequent out-of-order packets dropped
  - It won't matter that those packets successfully arrive at the receiver from the sender over the network

- TCP application-level throughput will suffer if there is too much packet reordering in the network
  - Data may have reached the receiver, but won't be delivered to apps upon a `recv()` (...or may not even be buffered!)

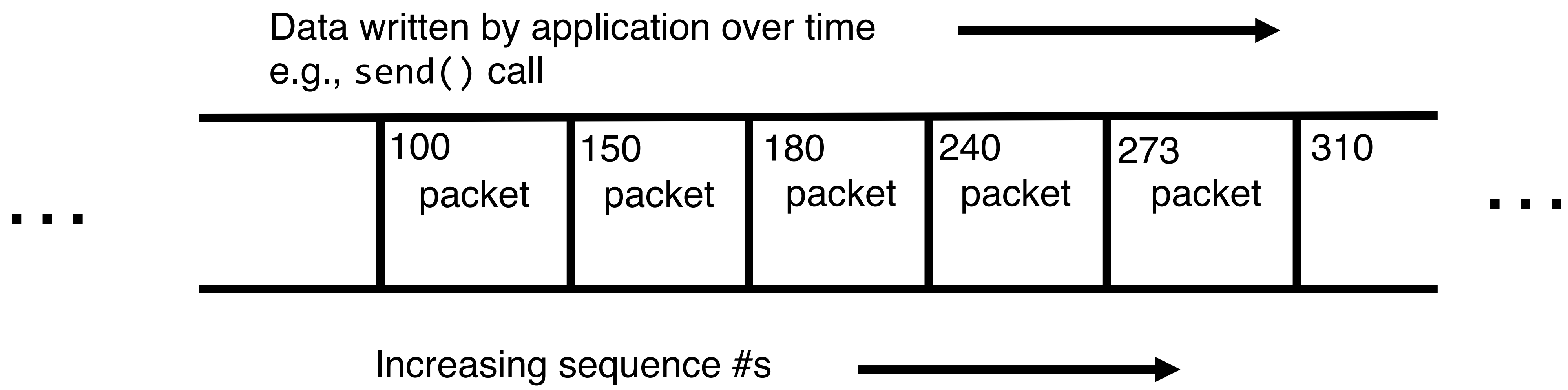
Stream-Oriented Data Transfer

Sequence numbers in the app's stream



TCP uses byte sequence numbers

Sequence numbers in the app's stream

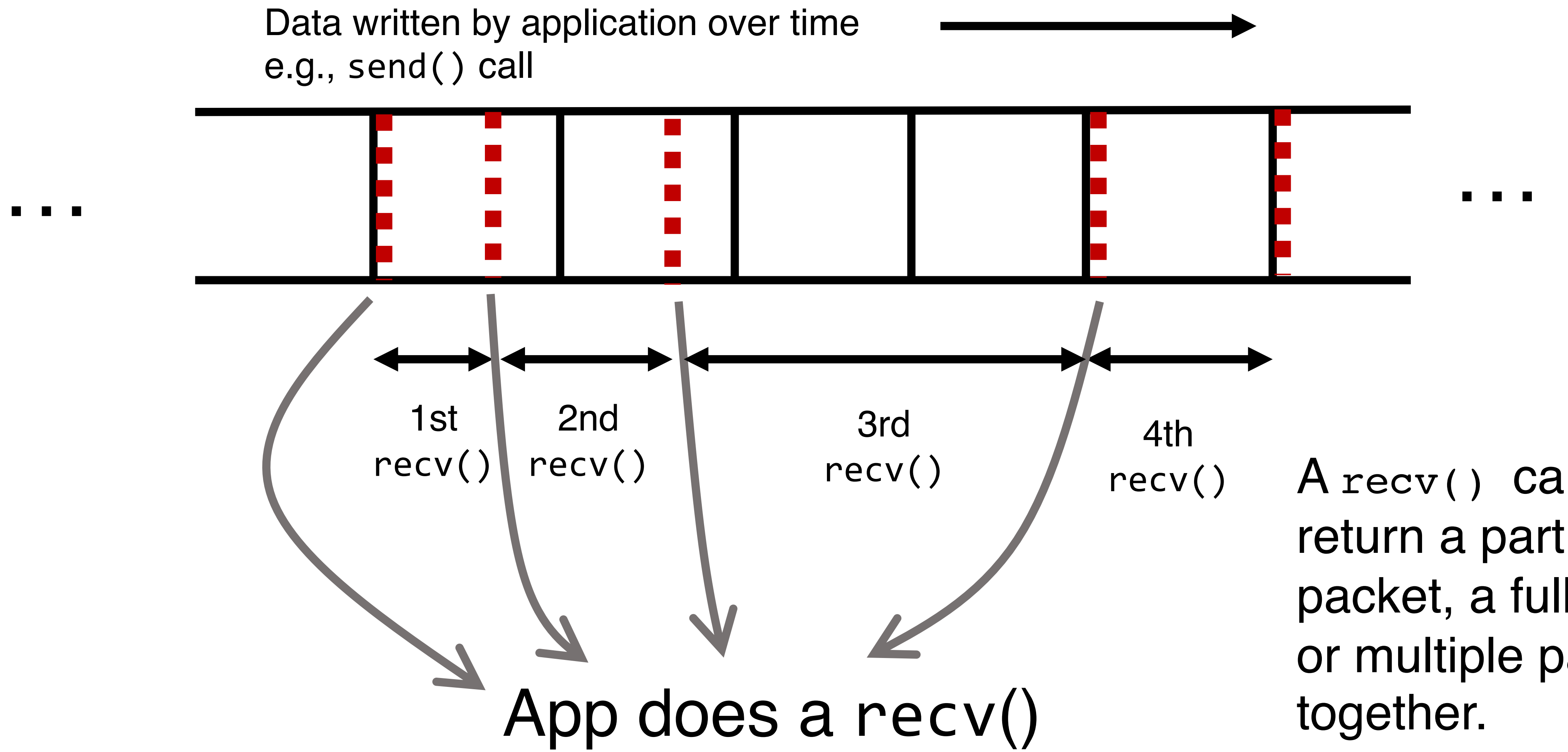


Packet boundaries aren't important for TCP software

TCP is a stream-oriented protocol

(We use `SOCK_STREAM` when creating sockets)

Sequence numbers in the app's stream



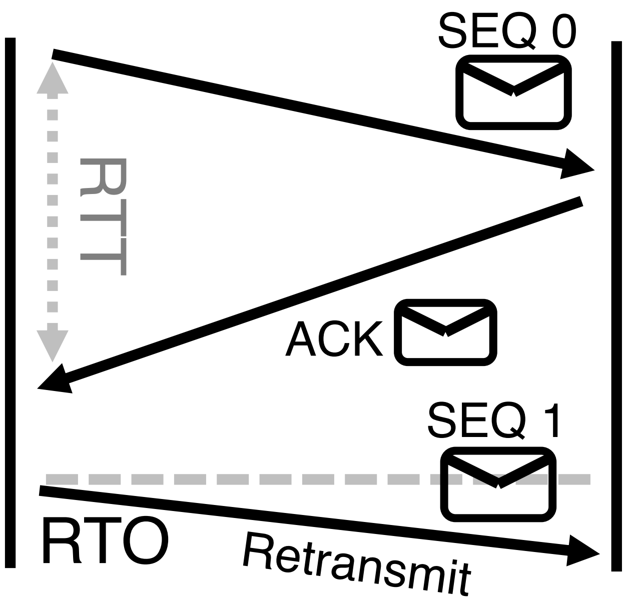
A `recv()` call may return a part of a packet, a full packet, or multiple packets together.

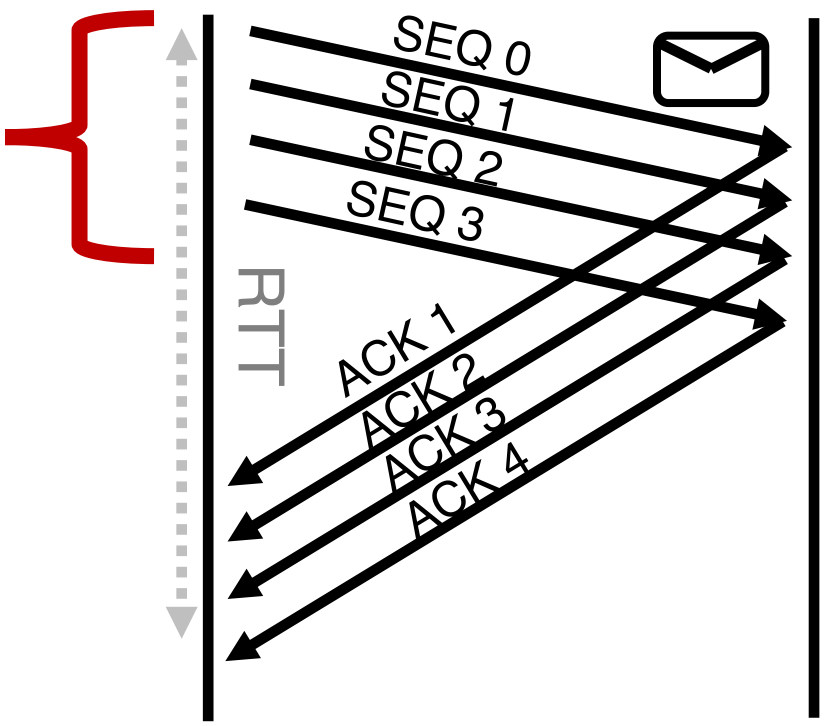
= window size

Proportional to throughput

How much data to keep in flight?

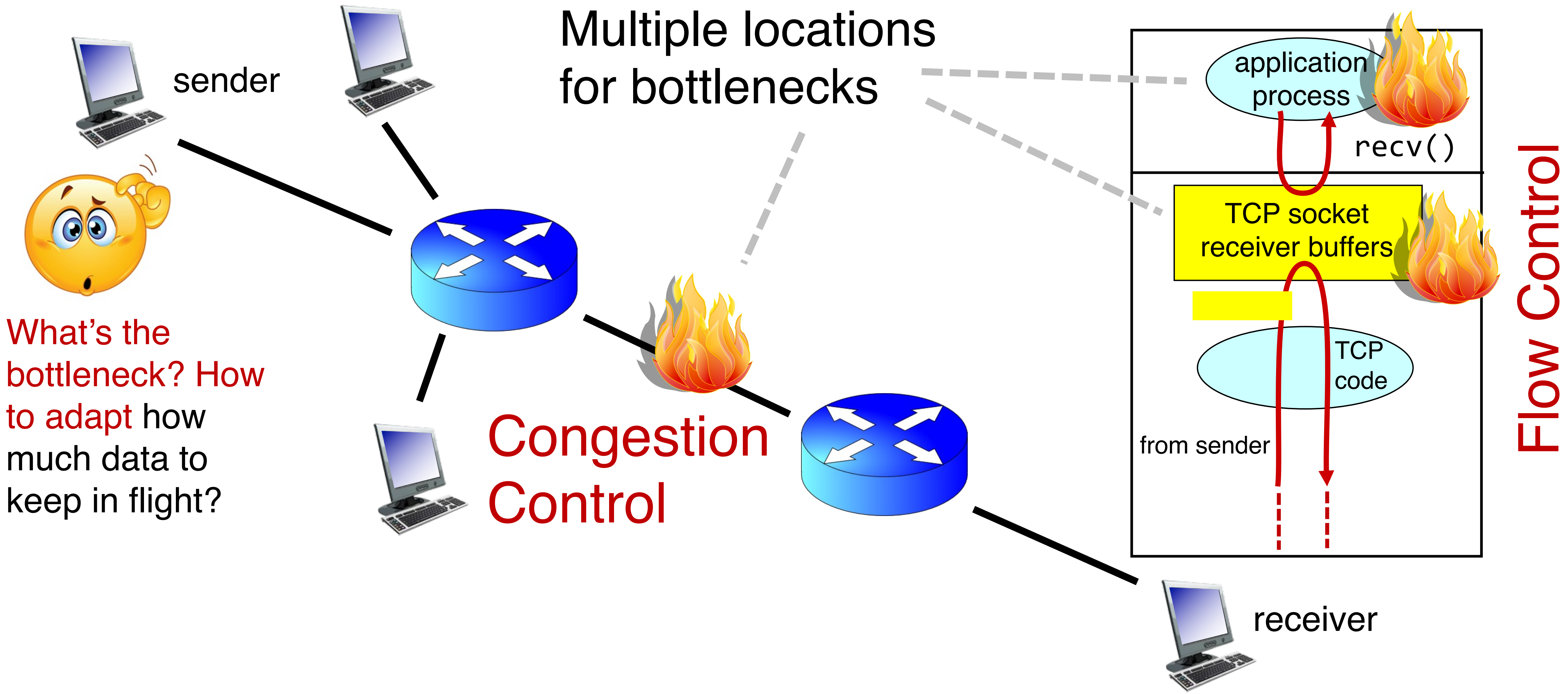
Stop and Wait





Pipelined Reliability

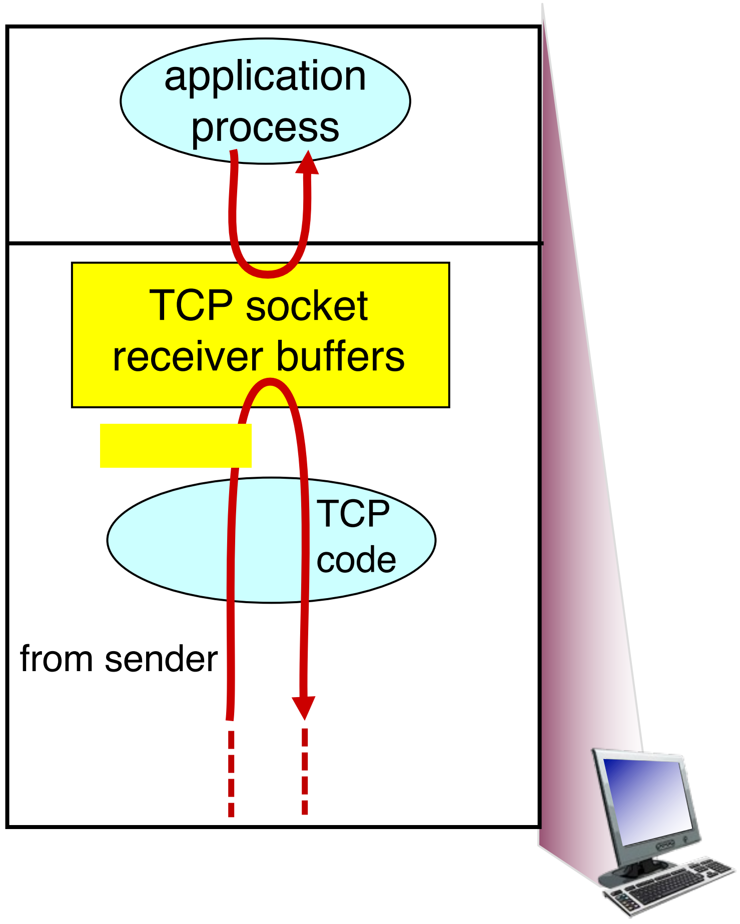
We want to increase throughput, but …



Flow Control

Socket buffers can become full

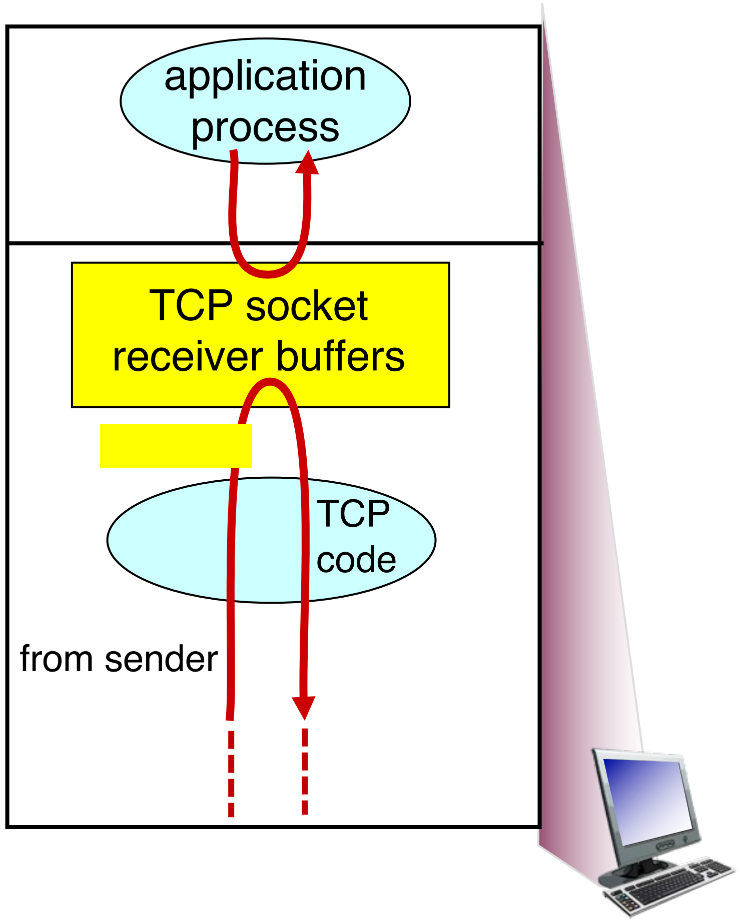
- Applications may read data slower than the sender is pushing data in
  - Example: what if an app infrequently or never calls `recv()`?
- There may be too much reordering or packet loss in the network
  - What if the first few bytes of a window are lost or delayed?
- Receivers can only buffer so much before dropping subsequent data



TCP receiver

Goal: avoid drops due to buffer fill

- Have a TCP sender only send as much as the free buffer space available at the receiver.
- *Amount of free buffer varies over time!*
- TCP implements flow control
- Receiver's ACK contains the amount of data the sender can transmit without running out the receiver's socket buffer
- This number is called the advertised window size



receiver protocol stack

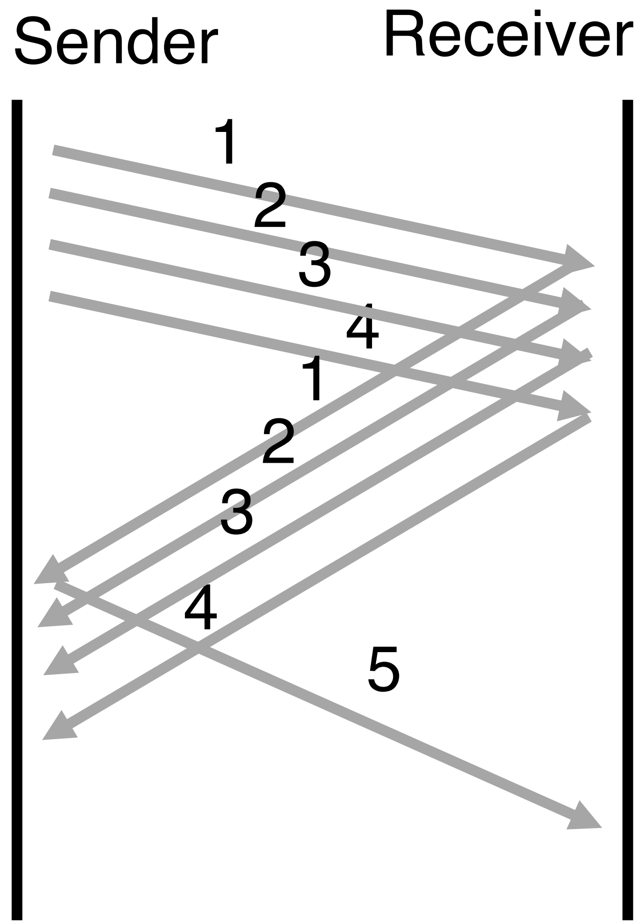
Flow control in TCP headers

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                      TCP Header Format

        Note that one tick mark represents one bit position.
```

TCP flow control

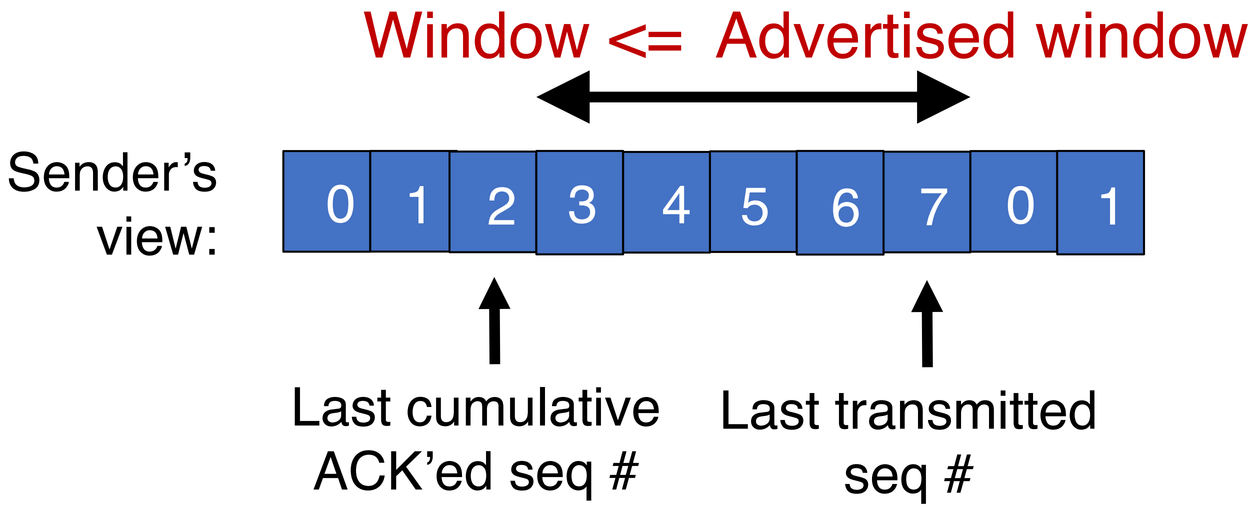
- Receiver advertises to sender (in the ACK) how much free buffer is available

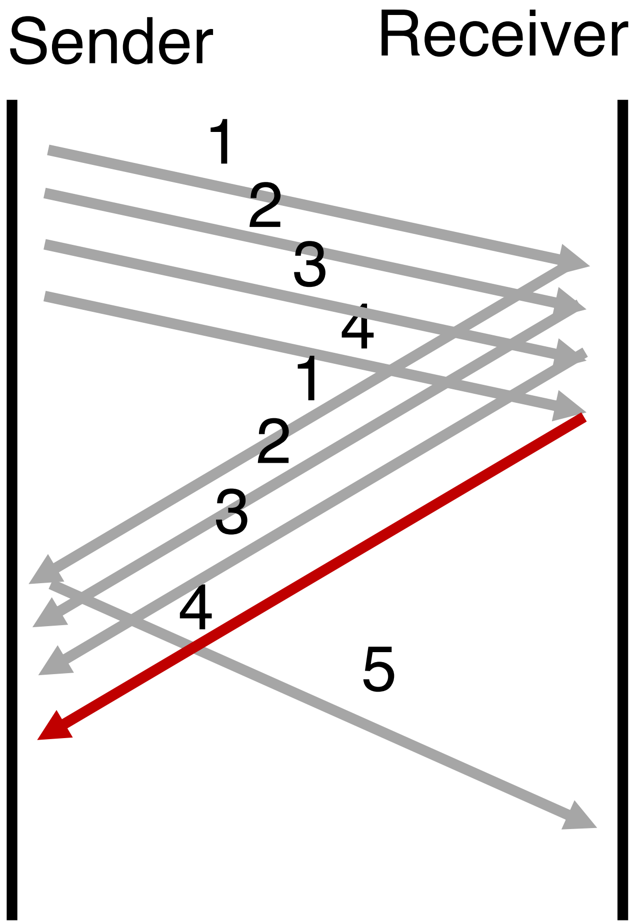
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                    TCP Header Format

      Note that one tick mark represents one bit position.
```



TCP flow control

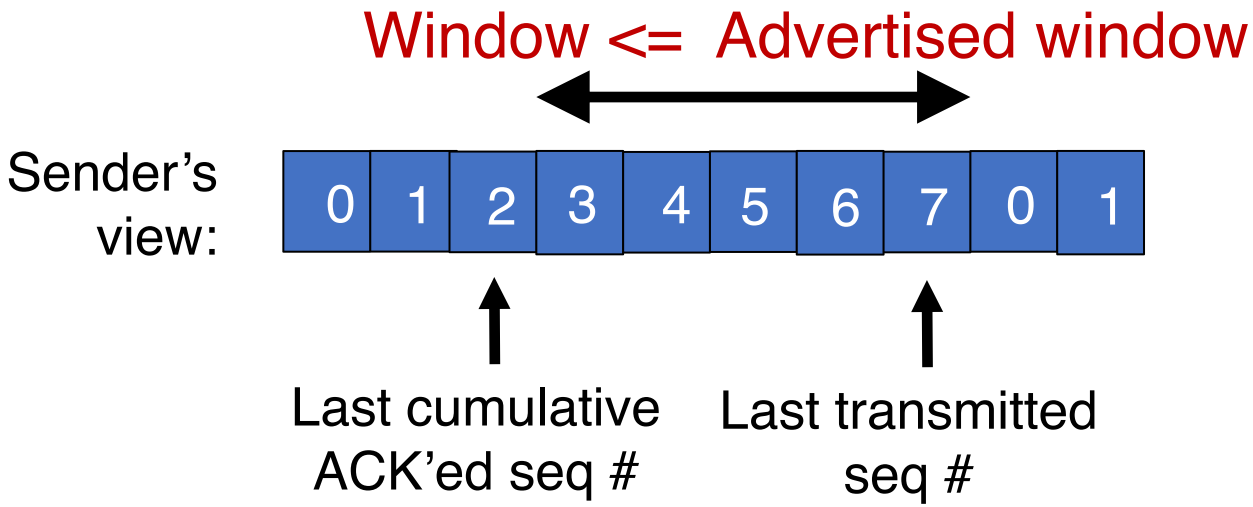
- Subsequently, the sender's sliding window cannot be larger than this value
- Restriction on new sequence numbers that can be transmitted
- == restriction on sending rate!

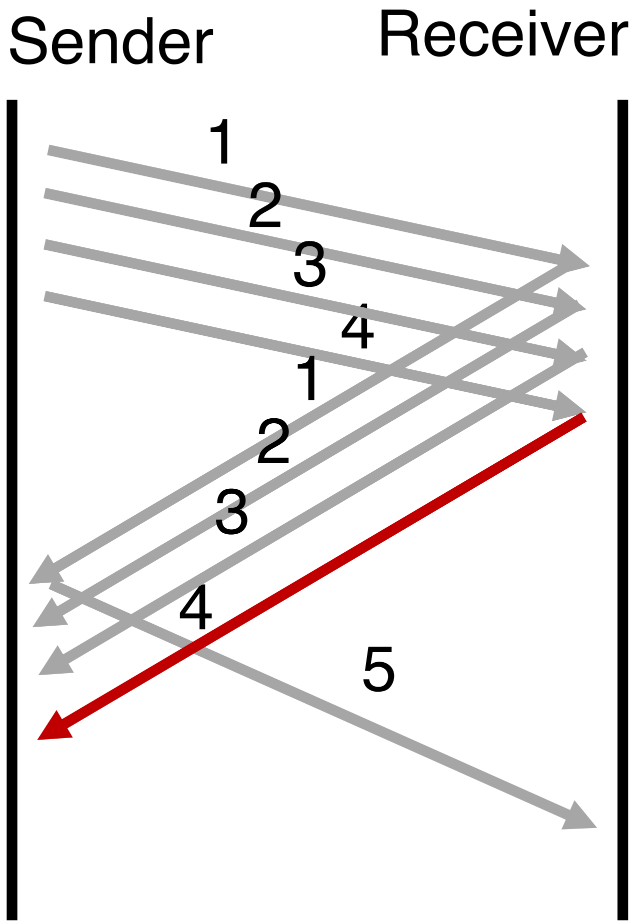




TCP flow control

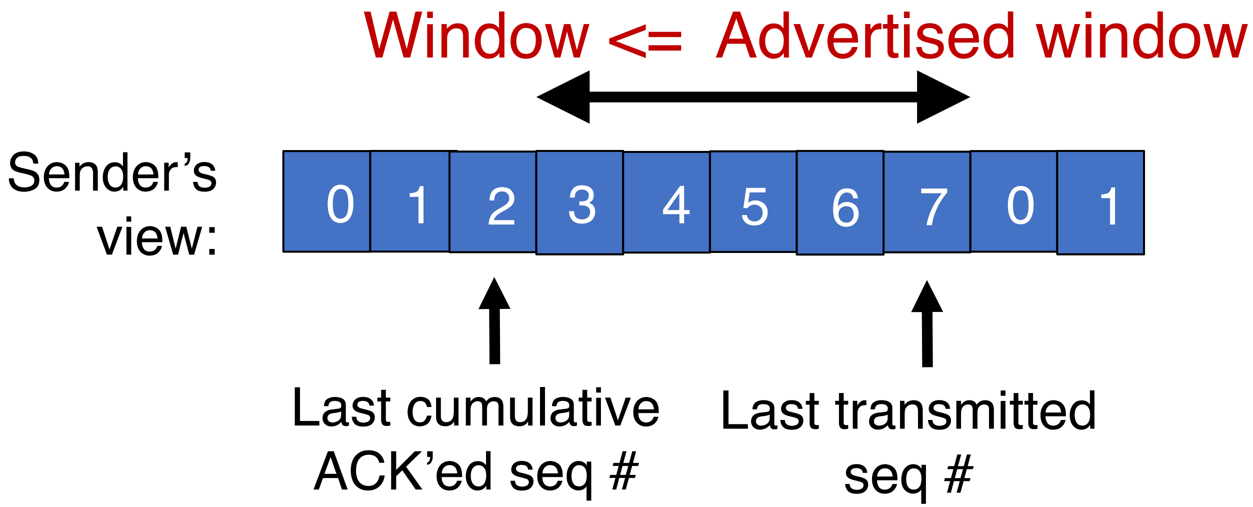
- If receiver app is too slow reading data:
  - receiver socket buffer fills up
  - So, advertised window shrinks
  - So, sender's window shrinks
  - So, sender's sending rate reduces

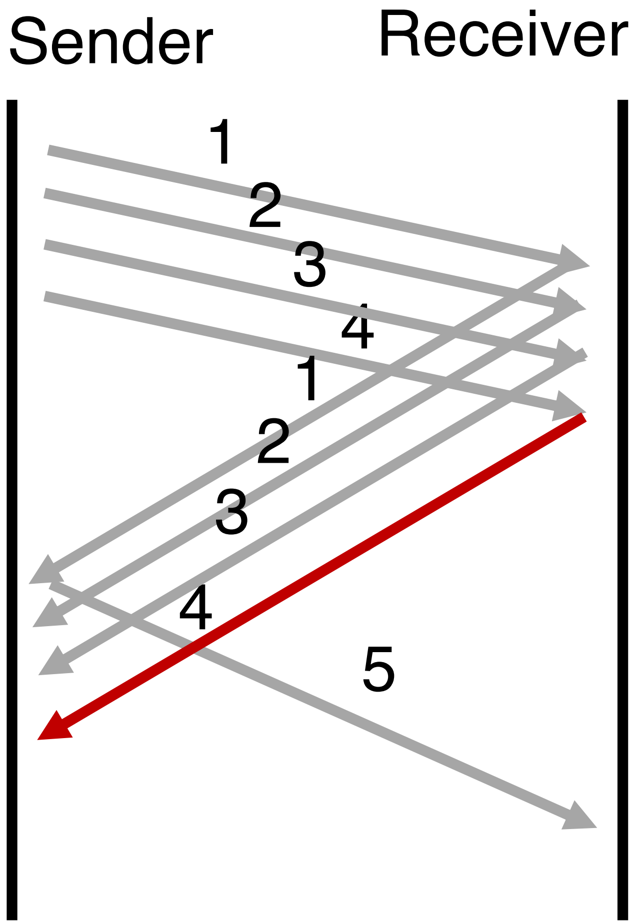




TCP flow control

Flow control matches the sender's write speed to the receiver's read speed.





Sizing the receiver's socket buffer

- Operating systems have a default receiver socket buffer size
  - Listed among `sysctl -a | grep net.inet.tcp` on MAC
  - Listed among `sysctl -a | grep net.ipv4.tcp` on Linux

- If socket buffer is too small, sender can't keep too many packets in flight ➔ lower throughput

- If socket buffer is too large, too much memory consumed per socket

- How big should the receiver socket buffer be?

Sizing the receiver's socket buffer

- Case 1: Suppose the receiving app is reading data too slowly:
  - no amount of receiver buffer can prevent low sender throughput if the connection is long-lived!

Sizing the receiver's socket buffer

- Case 2: Suppose the receiving app reads sufficiently fast *on average* to match the sender's writing speed.
  - Assume the sender has a window of size W.
  - The receiver must use a buffer of size at least W. Why?

- Captures two cases:
- (1) When the first sequence #s in the window are dropped
  - *Selective repeat*: data in window buffered until the ACKs of delivered data (within window) reach sender. Adv. win reduces sender's window
- (2) When the sender sends a burst of data of size W
  - Receiver may not match the *instantaneous* rate of the sender

Summary of flow control

- Keep memory buffers available at the receiver whenever the sender transmits data
- Buffers needed to hold for selective repeat and reassemble data in order
- Inform the sender on an on-going basis (each ACK)
- Function: match sender speed to receiver speed

- Correct socket buffer sizing is important for TCP throughput

Info on (tuning) TCP stack parameters

- https://www.ibm.com/support/knowledgecenter/linuxonibm/liaag/wkvm/wkvm_c_tune_tcpip.htm

- https://cloud.google.com/solutions/tcp-optimization-for-network-performance-in-gcp-and-hybrid