CS 352
Ordered Delivery

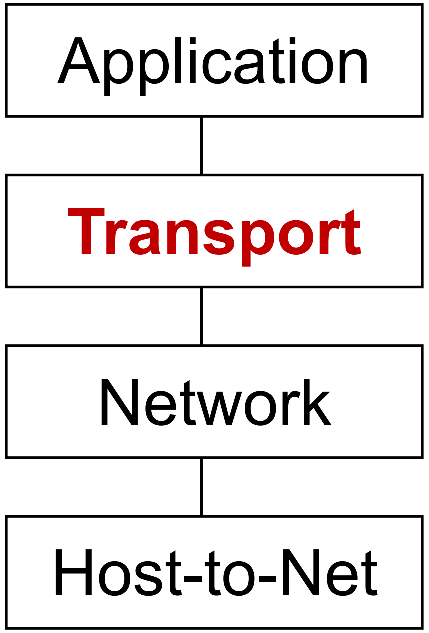
CS 352, Lecture 11.1

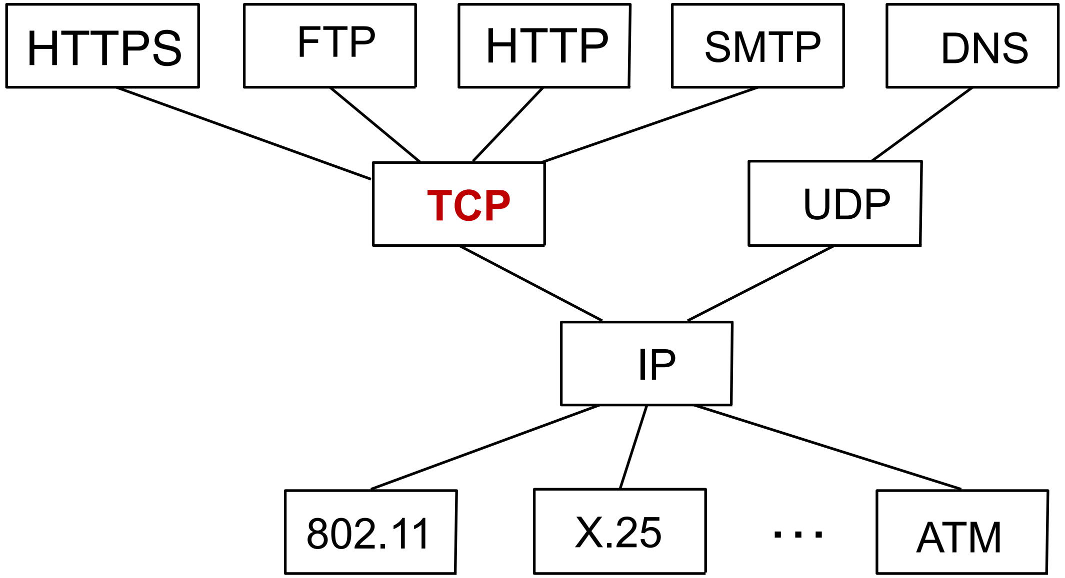
http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana

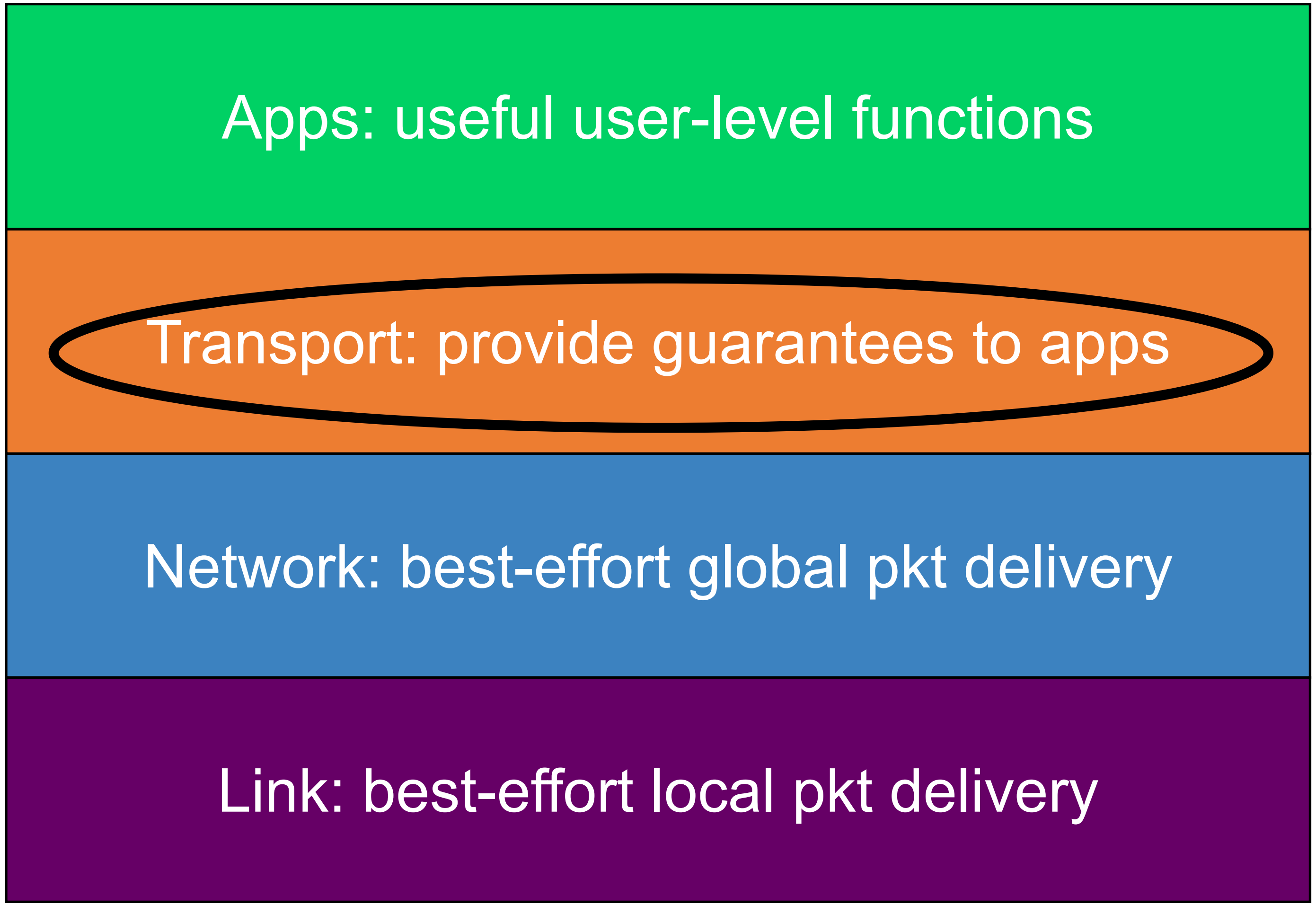
RUTGERS
UNIVERSITY | NEW BRUNSWICK

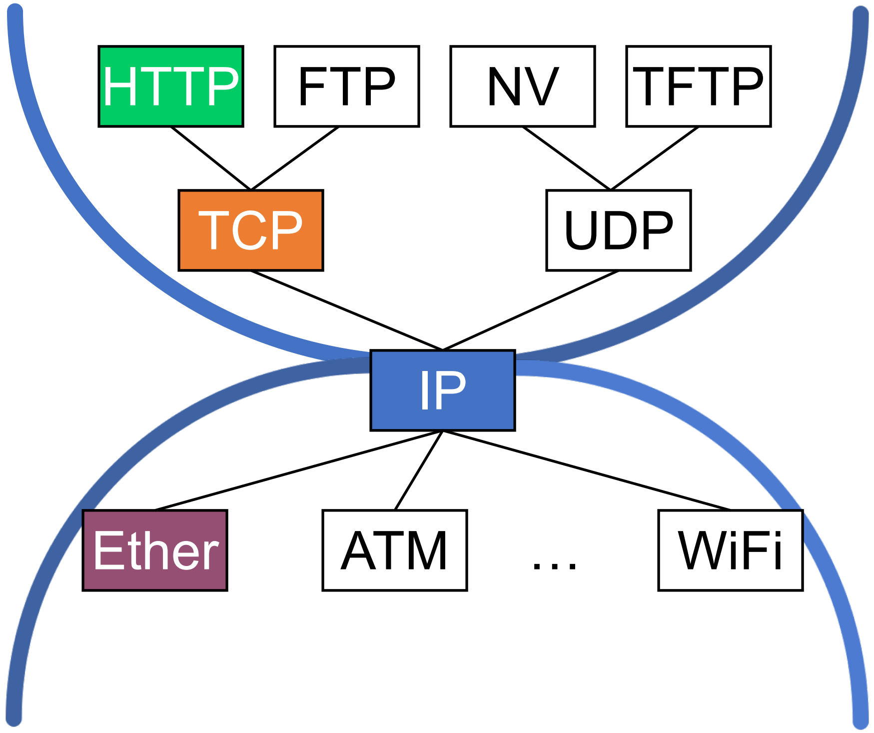
Transport


Application
Transport
Network
Host-to-Net


HTTPS
FTP
HTTP
SMTP
DNS
TCP
UDP
IP
802.11
X.25
...
ATM

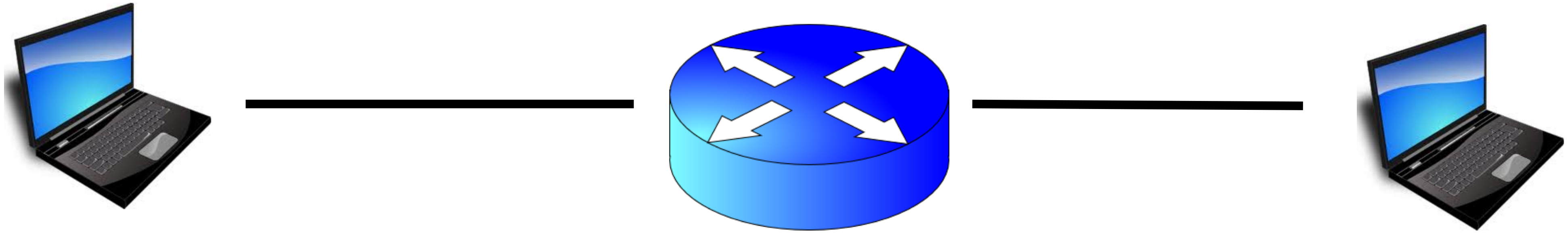
Modularity through layering





How do apps get perf guarantees?

- The network core provides no guarantees on packet delivery

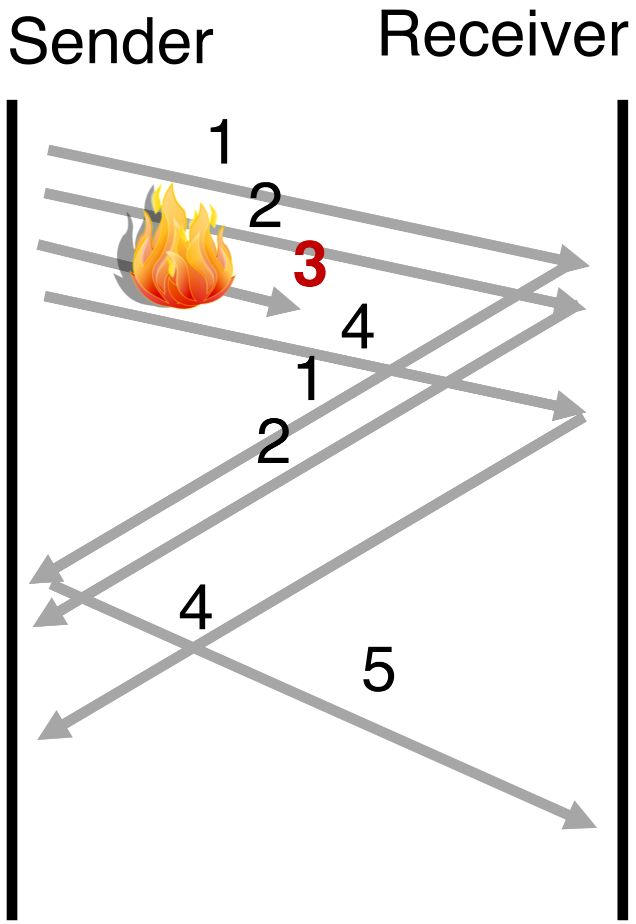


- Transport software on the endpoint oversees implementing guarantees on top of a best-effort network
- Three important kinds of guarantees
  - Reliability
  - Ordered delivery
  - Resource sharing in the network core

Transmission Control Protocol (TCP)

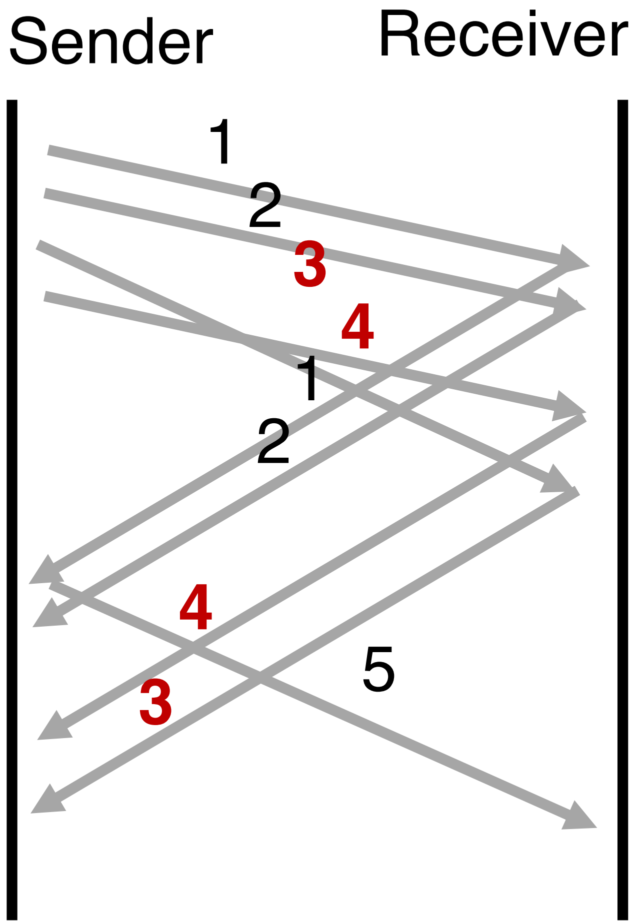
Reordering packets at the receiver side

- Let's suppose receiver gets packets 1, 2, and 4, but not 3 (dropped)
- Suppose you're trying to download a Word document containing a report
- What would happen if transport at the receiver directly presents packets 1, 2, and 4 to the Word application?



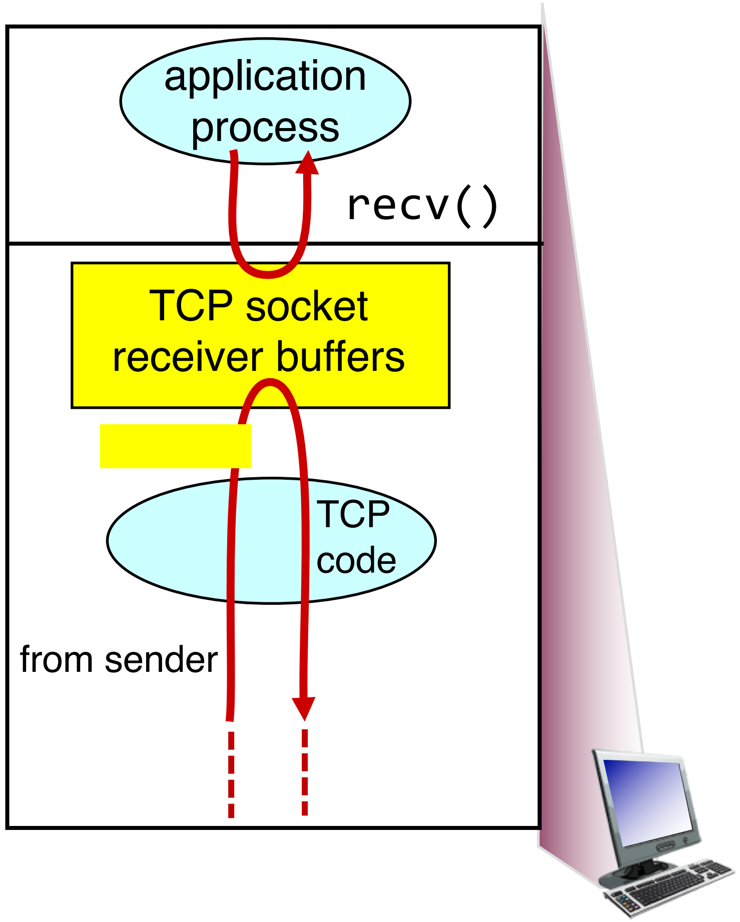
Reordering at the receiver side

- 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 side pushed it
- Receiver uses two mechanisms:
  - Sequence numbers
  - Receiver socket buffer
- We've already seen the use of both of these for reliability



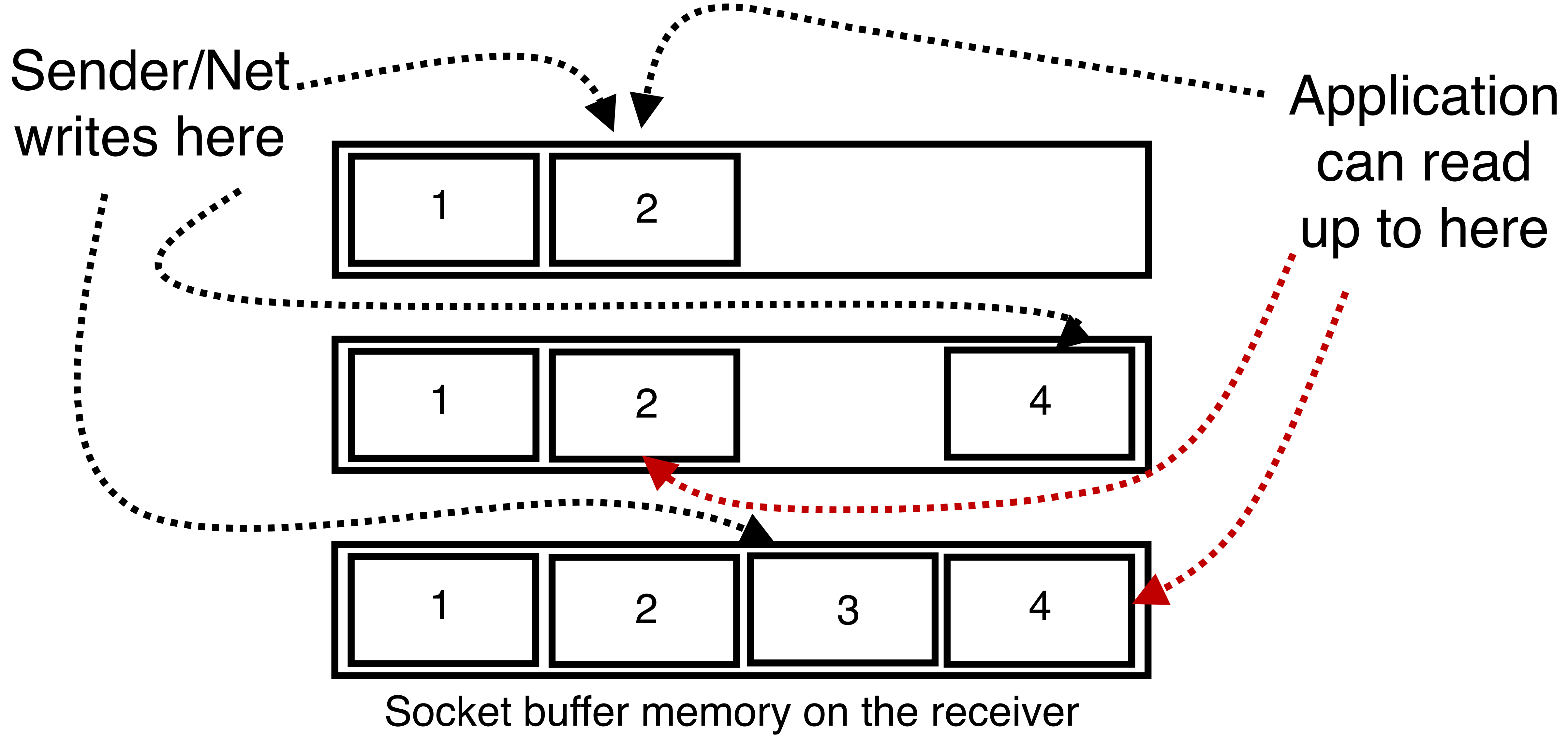
Interaction between apps and TCP

- Sender deposits data in receiver socket buffer
- An app with a TCP socket reads from the TCP receive socket buffer
  - e.g., when you do `data = sock.recv()`
- TCP receiver software only releases this data 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

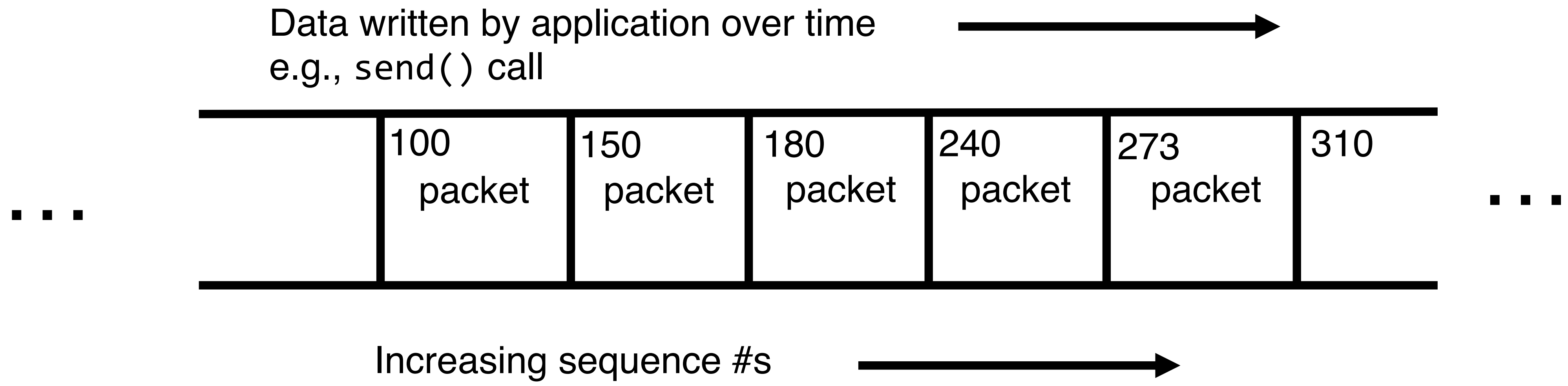


receiver protocol stack

TCP Reassembly

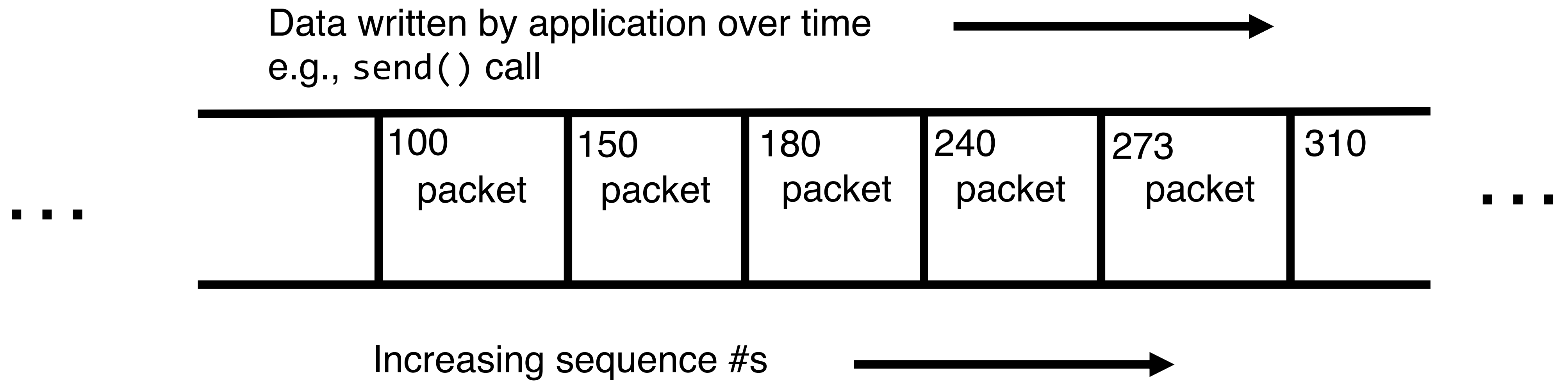


Sequence numbers in the app's stream



TCP uses byte sequence numbers

Sequence numbers in the app's stream

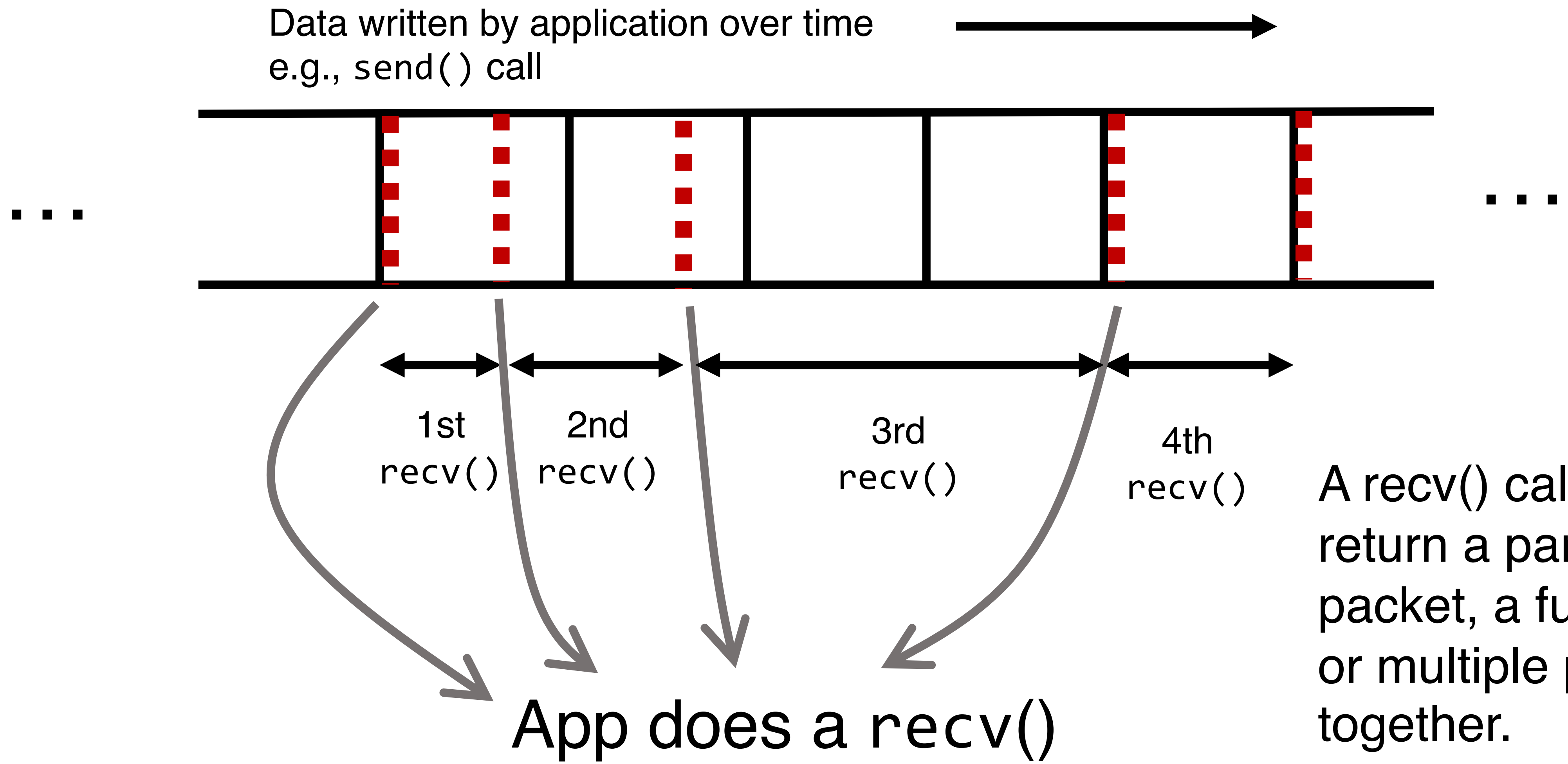


Packet boundaries aren't important for TCP software

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.

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 doesn'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 reach the receiver
  - But won't be delivered to apps upon a `recv()`

Summary of TCP ordered delivery

- In-order delivery accomplished through socket buffer and TCP reassembly at receiver
- TCP is a stream-oriented protocol, where the boundaries between packets aren't important
- Significant packet reordering reduces TCP application throughput

CS 352
Flow Control

CS 352, Lecture 11.2

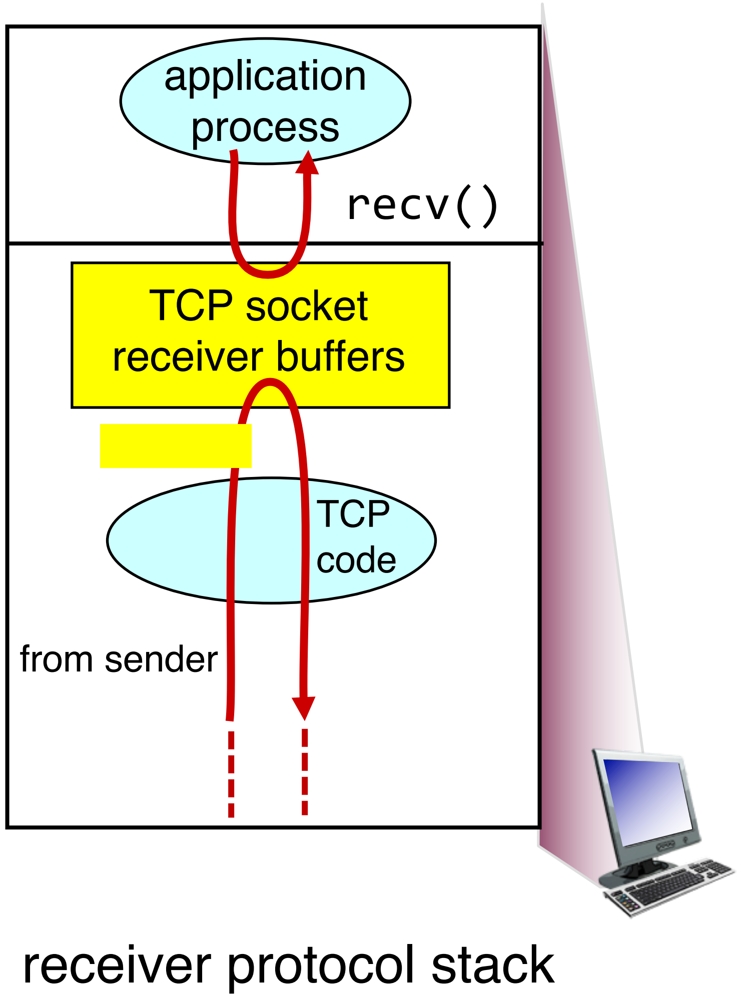
http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

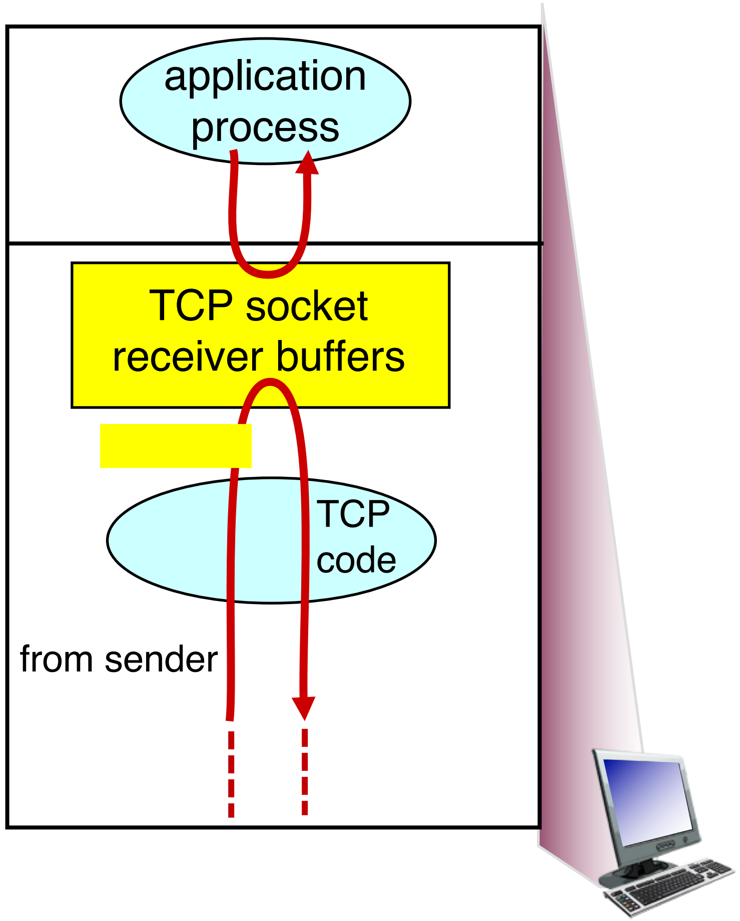
Review: app and socket buffer interaction

- Sender deposits data in receiver socket buffer
- An app with a TCP socket reads from the TCP receive socket buffer
  - e.g., when you do `data = sock.recv()`
- Buffers used for ordering & reliability
- Ordering: only release data to app when data in order with everything else app has read previously
- Reliability: avoid wasteful sender retransmissions using selective repeat



But socket buffers can get full…

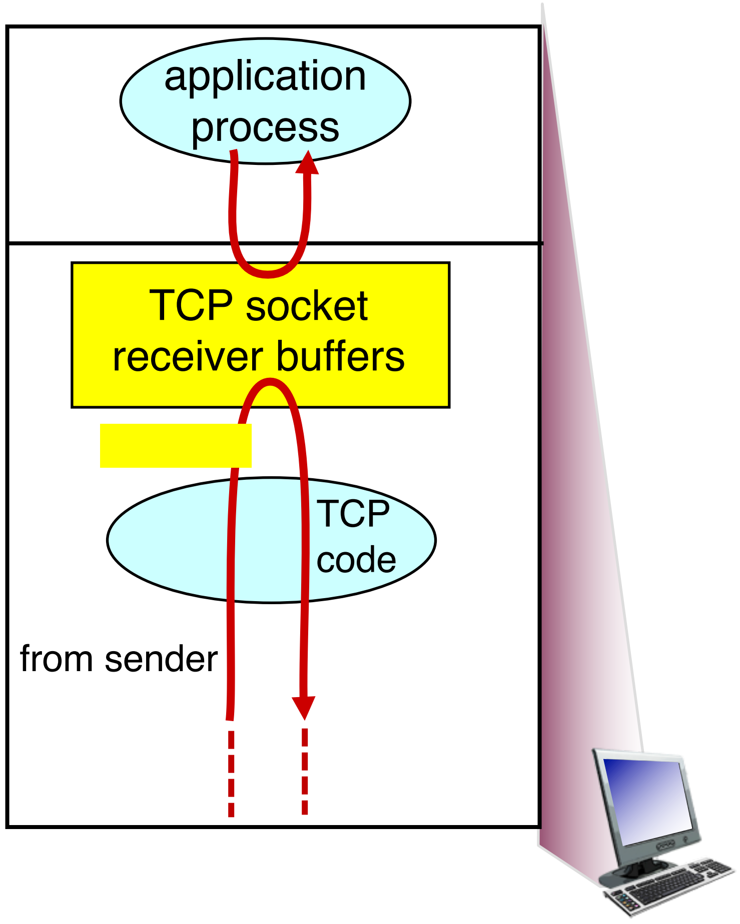
- Applications may read data slower than the sender is pushing data in
  - Example: what if an app infrequently or never calls `recv()`?
- There may be too much reordering or packet loss in the network
  - What if the first few bytes of a window are lost or delayed?
- Receivers can only buffer so much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



receiver protocol stack

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

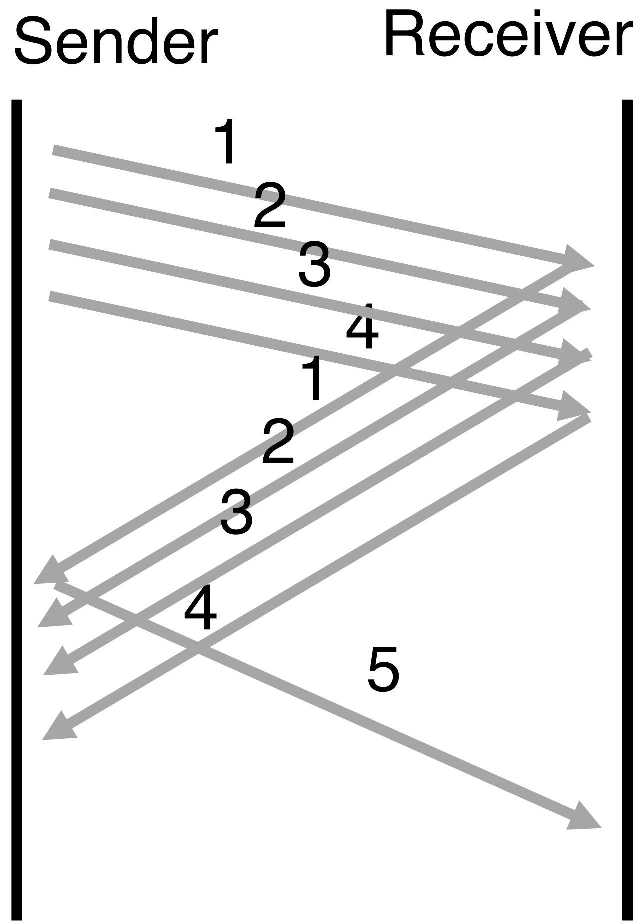
Flow control in TCP headers

```
 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

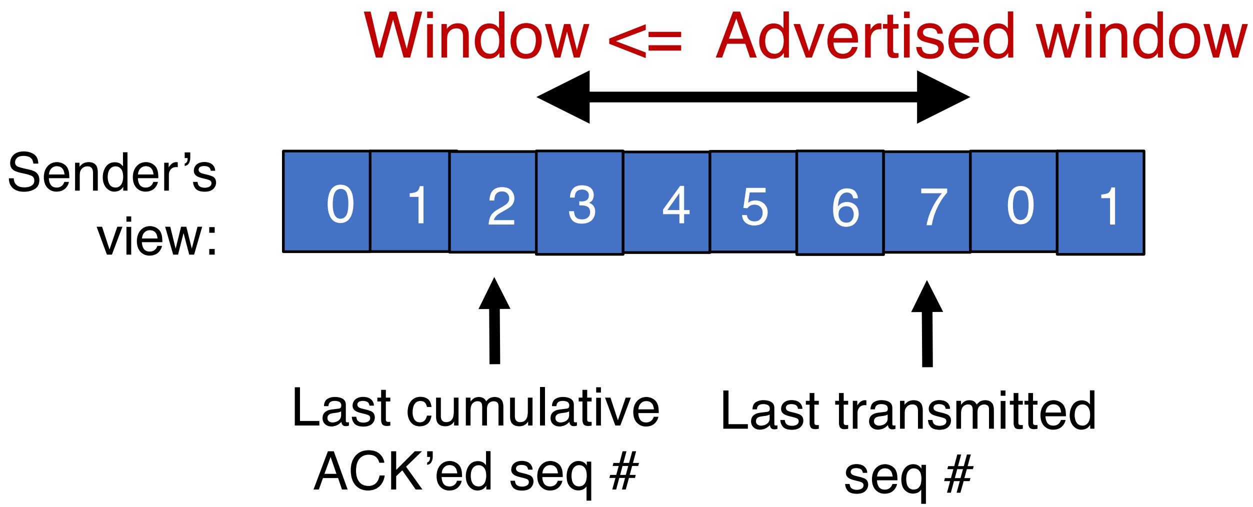
- Receiver advertises to sender (in the ACK) how much free buffer is available

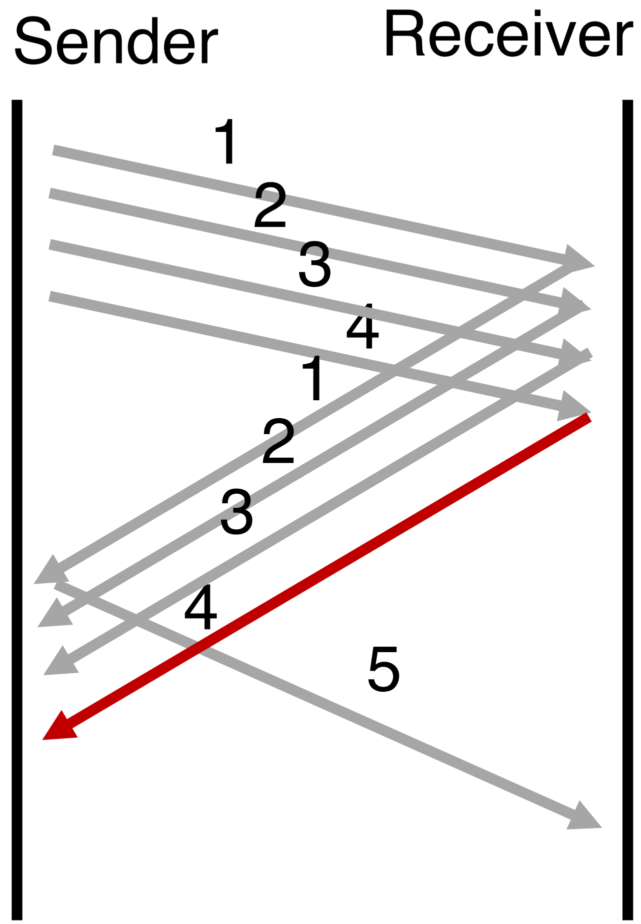
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        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

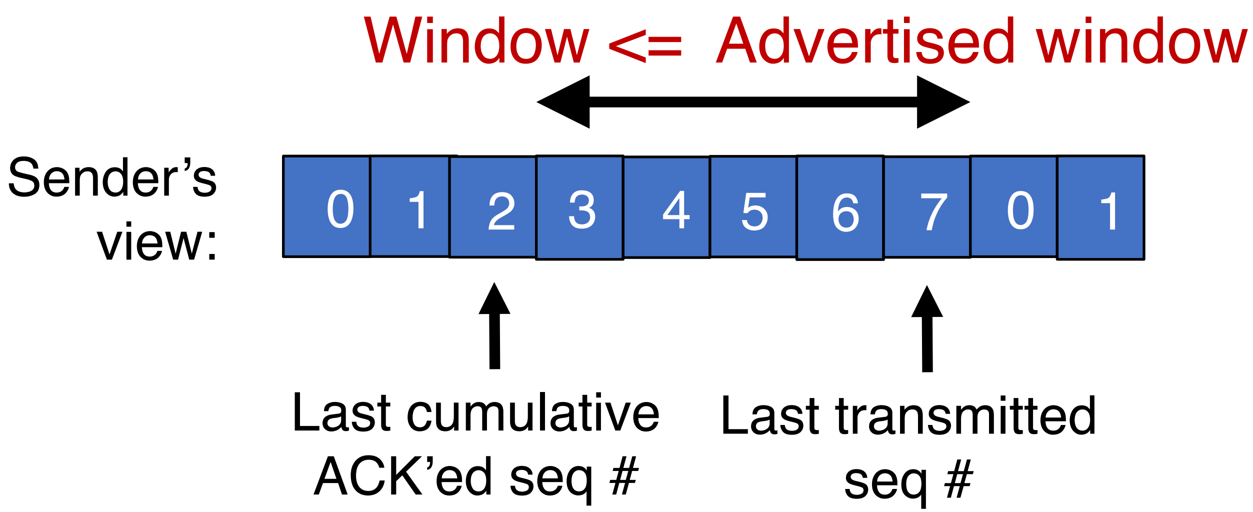
- Subsequently, the sender's sliding window cannot be larger than this value
- Restriction on new sequence numbers that can be transmitted
- Restriction on TCP sending rate

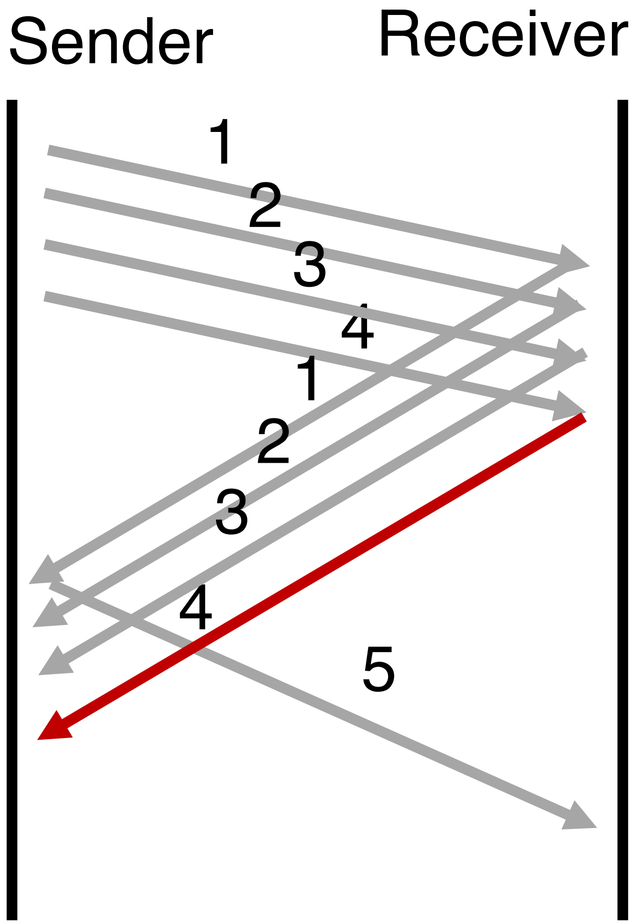




TCP flow control

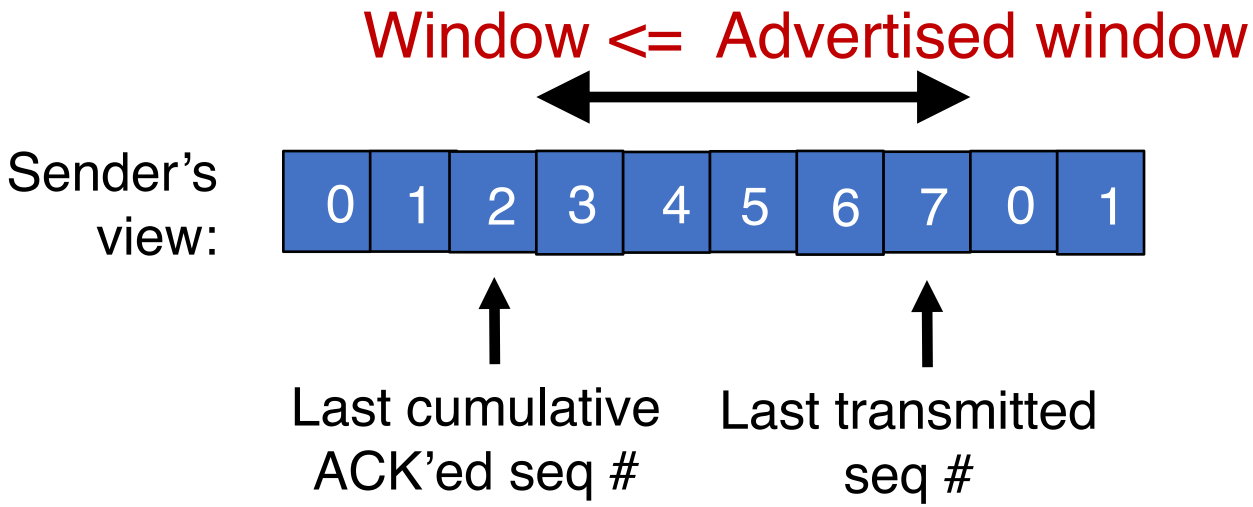
- If receiver app is too slow reading data:
  - receiver socket buffer fills up
  - So, advertised window shrinks
  - So, sender’s window shrinks
  - So, sender’s sending rate reduces

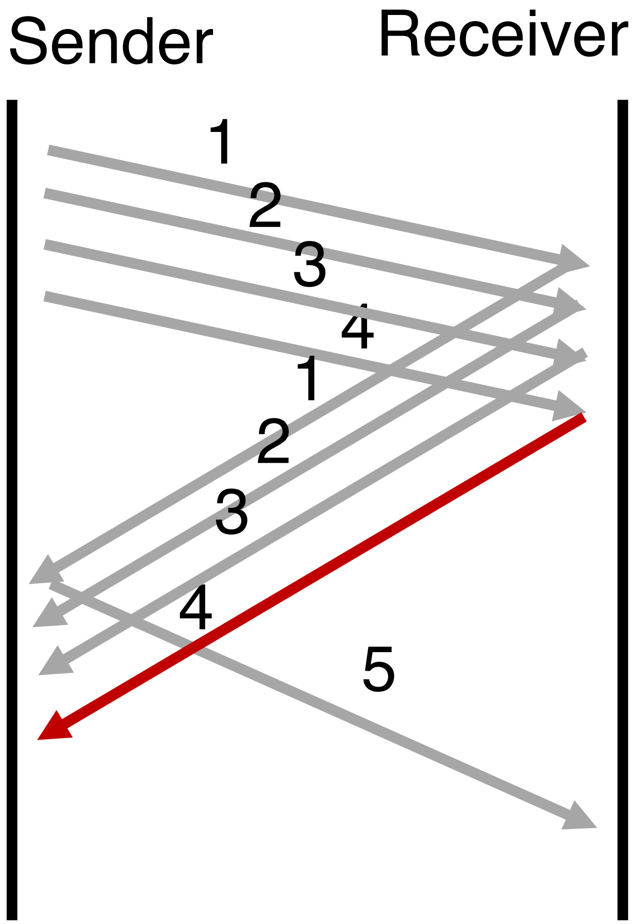




TCP flow control

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 the parameters in `sysctl -a | grep net.inet.tcp` on MAC, `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
  - The rest of the window must be buffered until the ACKs (of the rest of the window) reach sender. Adv. window in ACKs 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

- A mechanism to keep buffers available at the receiver whenever the sender transmits data
- Main function: match sender speed to receiver speed
- Socket buffer sizing is important for throughput