CS 352
Reliability: Pipelined Delivery

Lecture 12

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

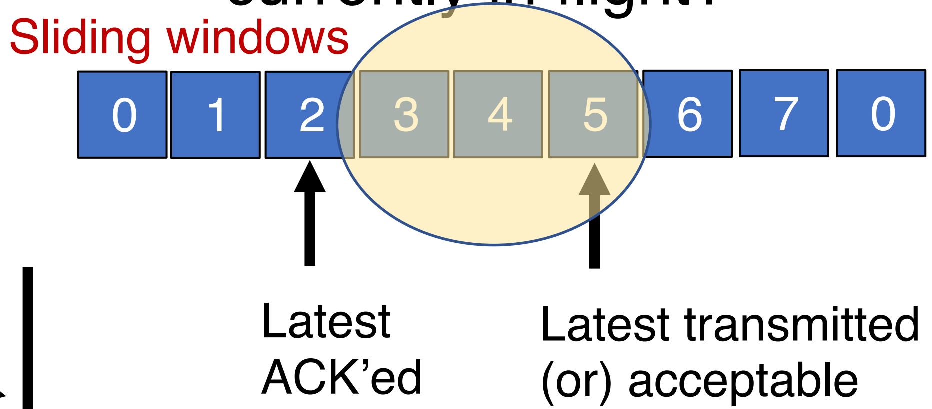
Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

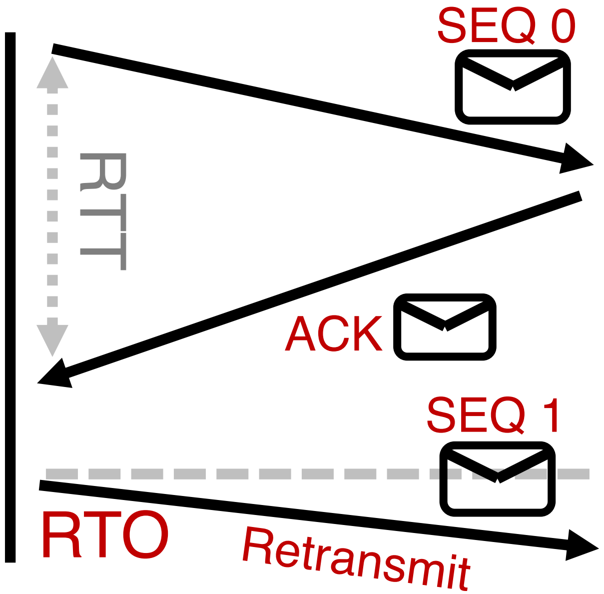
Quick recap of concepts

TCP: Connection-oriented

Q1. Which packets are currently in flight?



Stop and Wait

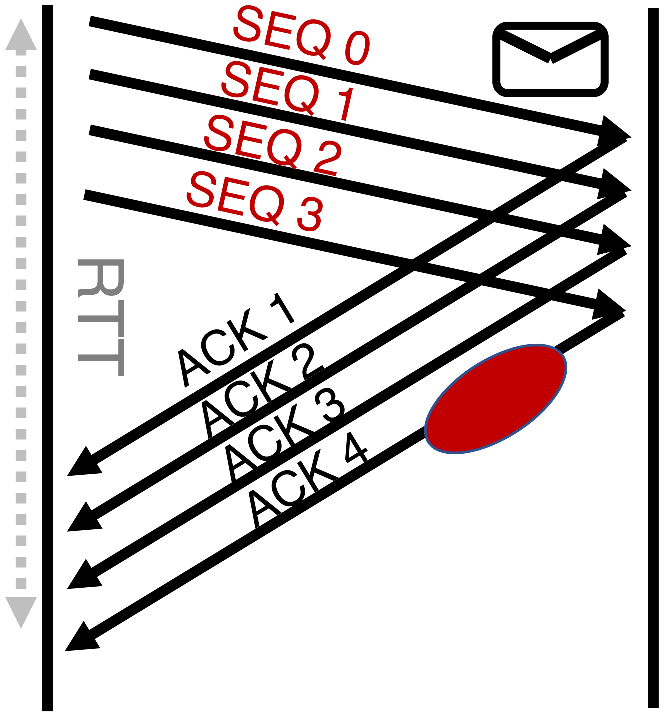


ACK pkts after a drop?

No
??

Yes
What ACK no?

Pipelined Reliability



Q2. Which packets were successfully delivered?

Q3. Which packets should the sender retransmit?

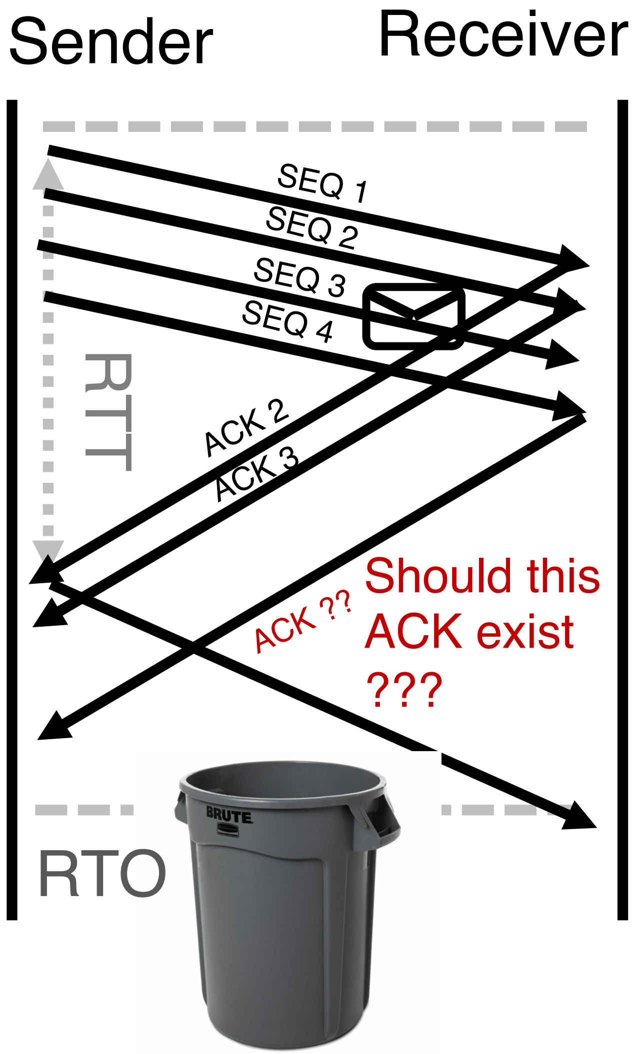
Which packets to retransmit?

How to identify dropped packets?

- Suppose 4 packets sent, but 1 dropped. How does sender know which one(s) dropped?

- Recall: Receiver writes sequence numbers on the ACK indicating successful reception
- Key idea: Sender can infer which data was received successfully using the ACK #s!
  - Hence, sender can know which data to retransmit

- Q1: Should receivers ACK subsequent packets upon detecting data loss?
- Q2: If so, what sequence number should receiver put on the ACK?



Receiver strategies upon packet loss

ACK pkts after a drop?

No

Go-back-N

Yes

Selective Repeat

What # on ACK?

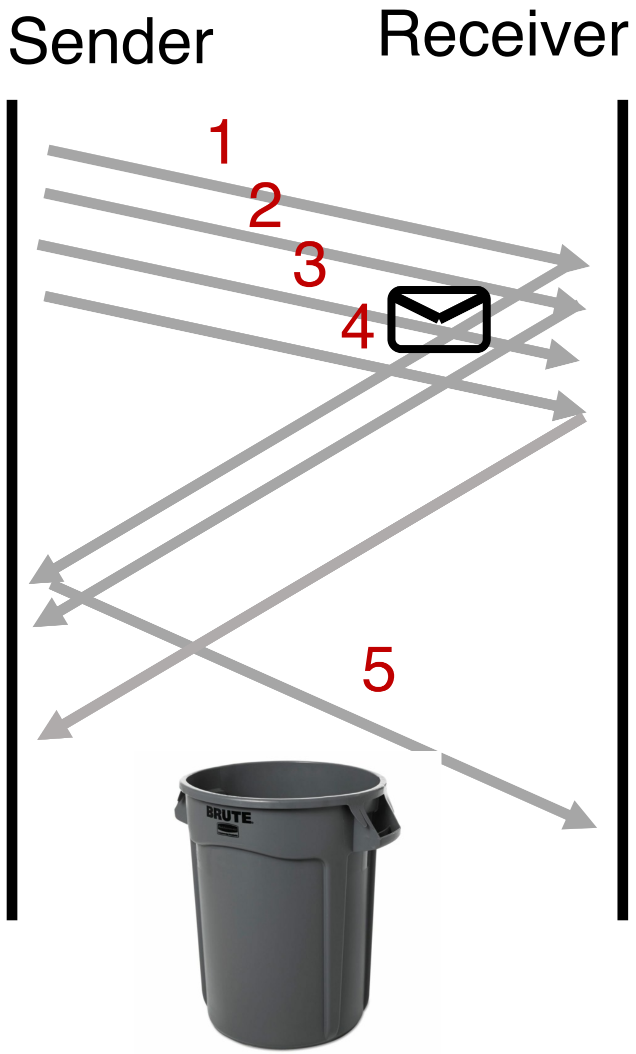
Last seq# in order

Cumulative ACK

Seq# *ranges* received so far

Selective ACK

TCP's default



Sliding Window with Go Back N

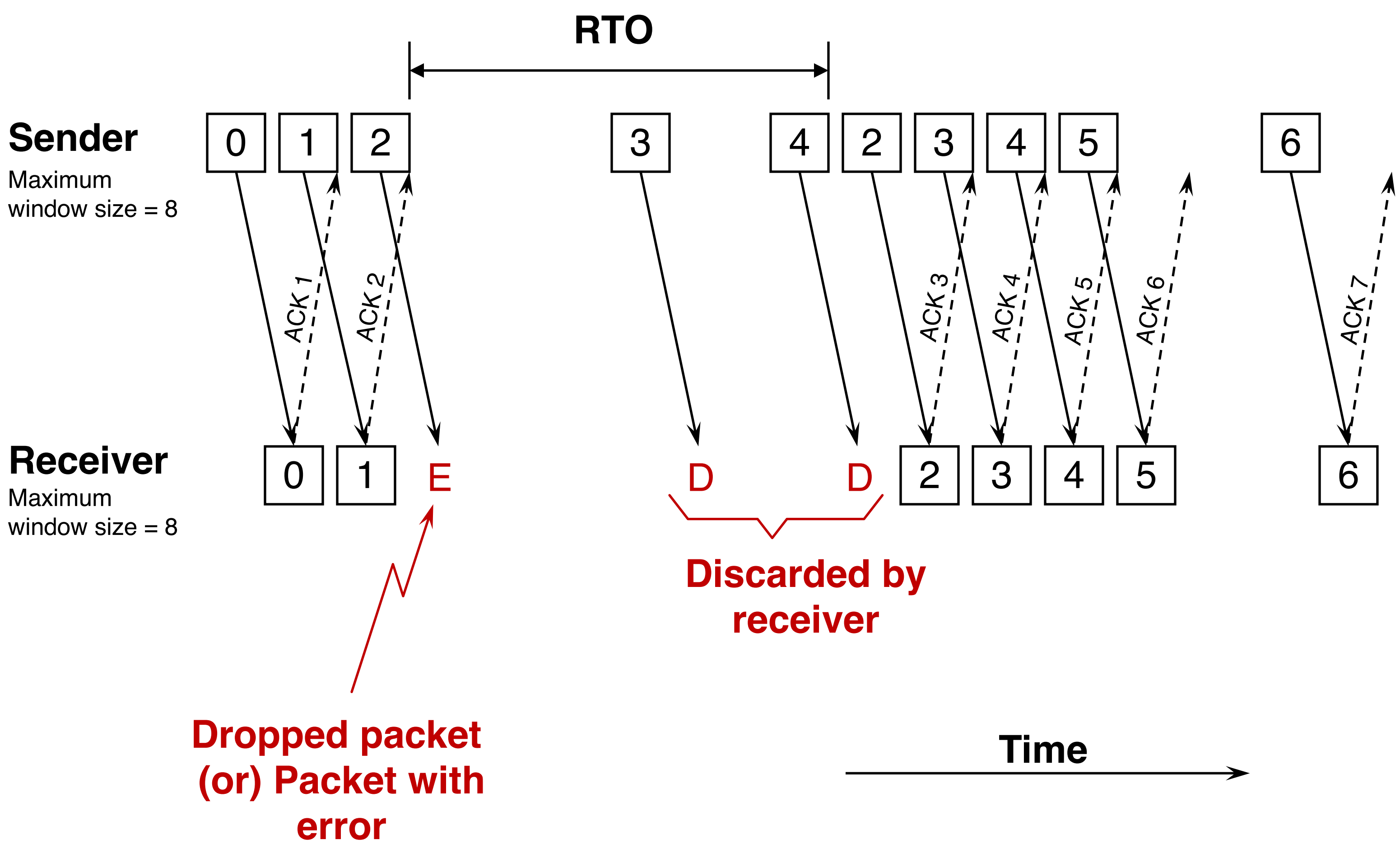
- When the receiver notices missing data:

- It simply discards all data with greater sequence numbers
  - i.e.: the receiver will send no further ACKs

- The sender will eventually time out (RTO) and retransmit all the data in its sending window

- Subtle: conceptually, separate timer per byte to infer RTO

Go back N


RTO
Sender
Maximum
window size = 8
0
1
2
3
4
2
3
4
5
6
ACK 1
ACK 2
ACK 3
ACK 4
ACK 5
ACK 6
ACK 7
Receiver
Maximum
window size = 8
0
1
E
D
D
2
3
4
5
6
Discarded by
receiver
Dropped packet
(or) Packet with
error
Time

Go back N

- Go Back N can recover from erroneous or missing packets.

- But it is wasteful.

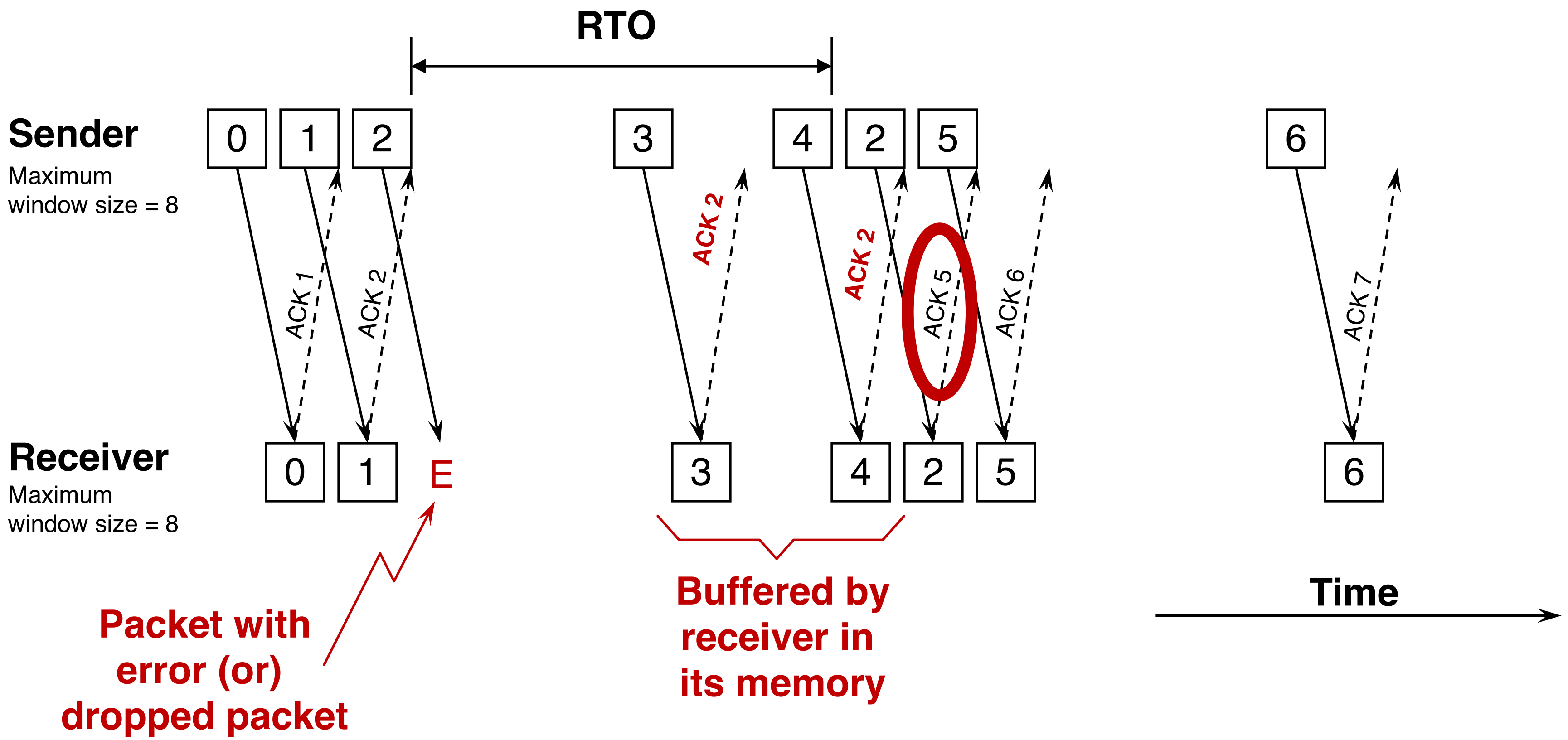
- If there are errors, the sender will spend time and network bandwidth retransmitting data the receiver has already seen.

Selective repeat with cumulative ACK

Idea: sender should only retransmit dropped/corrupted data.

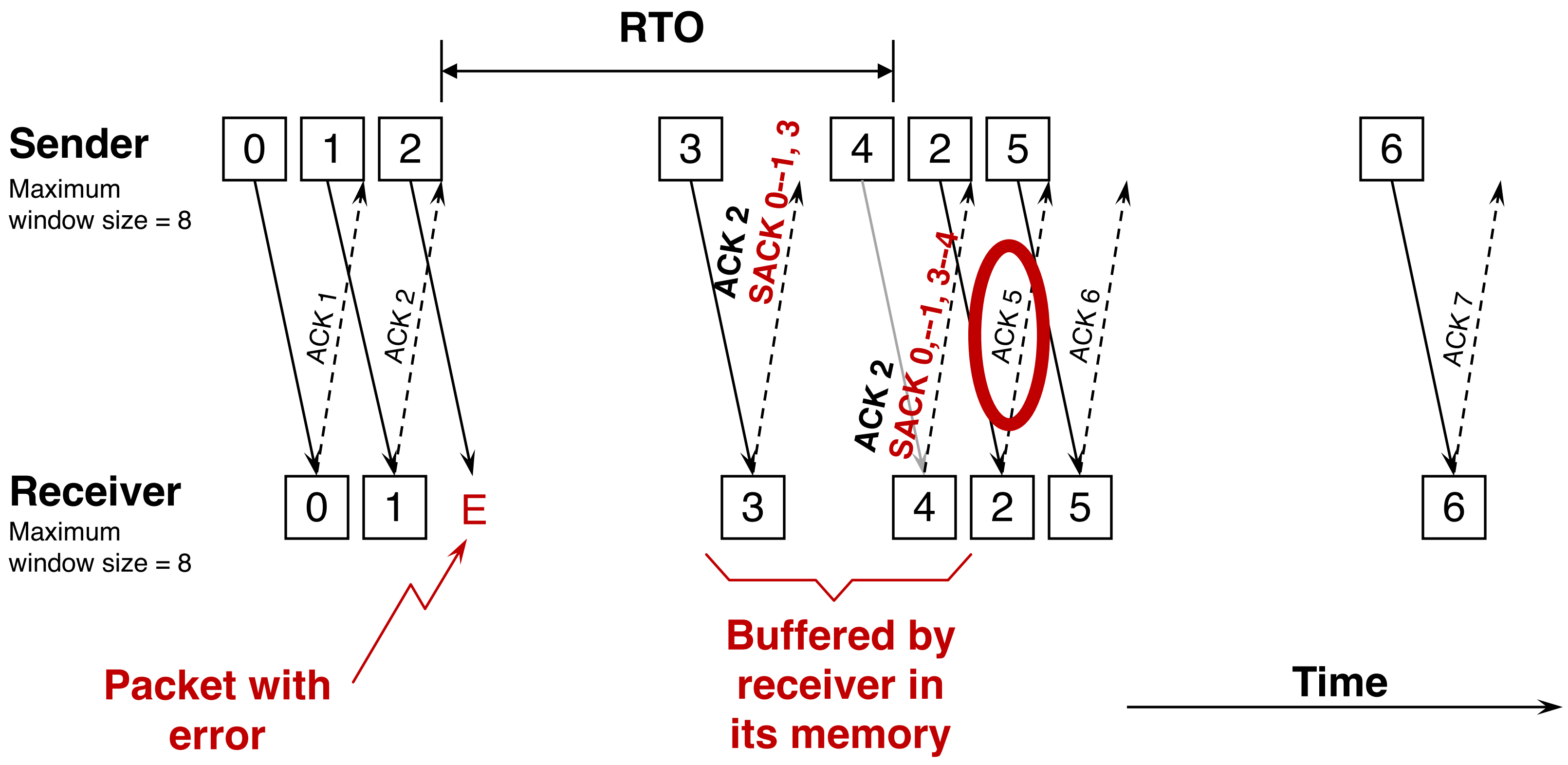
- The receiver stores all the correct frames that arrive following the bad one. (Note that the receiver requires memory to hold data for each sequence number in the receiver window.)
- When the receiver notices a skipped sequence number, it keeps acknowledging the first in-order sequence number it wants to receive. This is termed cumulative ACK.
- When the sender times out waiting for an acknowledgement, it just retransmits the first unacknowledged data, not all its successors.
- Recall that RTO applies independently to each sequence #

Selective repeat with cumulative ACK



Subtle: Even if there were multiple drops, retransmission after an RTO only includes the first dropped sequence number. Recovering each drop will require one RTO after corresponding packet was transmitted.

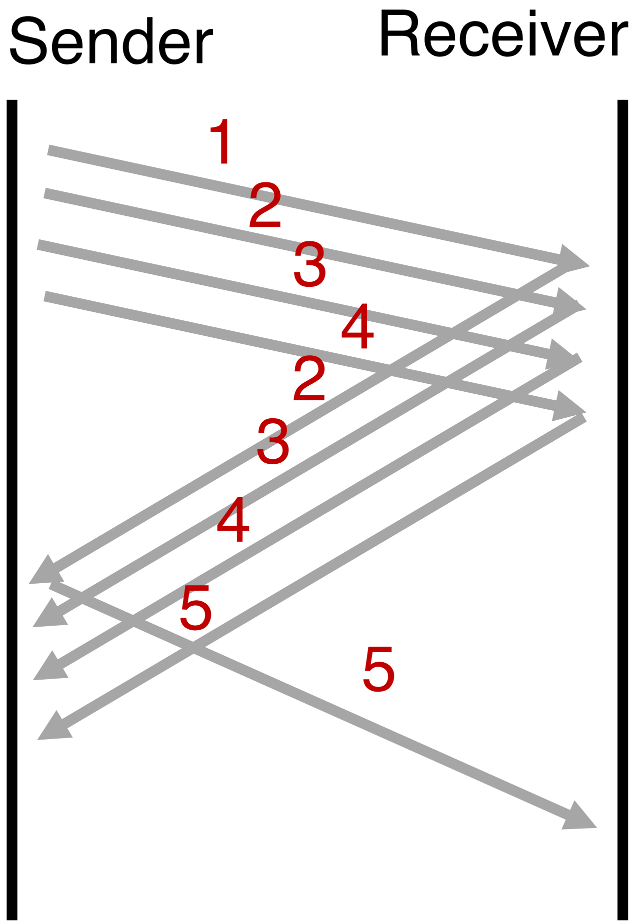
Selective repeat with selective ACK



This slide assumes retransmissions are only triggered by an RTO.
If other signals were to be used to retransmit earlier (e.g., triple dup ACK -- more on this soon),
SACK significantly reduces the number of duplicate transmissions compared to cumulative-only ACKs.

TCP: Cumulative & Selective ACKs

- Sender retransmits the seq #s it thinks aren't received successfully yet
- Pros & cons: selective vs. cumulative ACKs
  - Precision of info available to sender
  - Redundancy of retransmissions
  - Packet header space
  - Complexity (and bugs) in transport software
- On modern Linux, TCP uses selective ACKs by default



TCP reliability metadata

Metadata on TCP packets for Reliability

- TCP uses metadata in the form of sequence #s and ACK #s
- Where are these stored? Naturally, in the packet header!

TCP header structure

Source port, destination port (connection demultiplexing)

Size of the TCP header (in 32-bit words)

Basic error detection through checksums (similar to UDP)

```
 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 header structure

Identifies data in the packet from sender's perspective

TCP uses byte seq #s

Identifies the data being ACKed from the receiver's perspective.

TCP uses next seq # that the receiver is expecting.

```
 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.
```

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

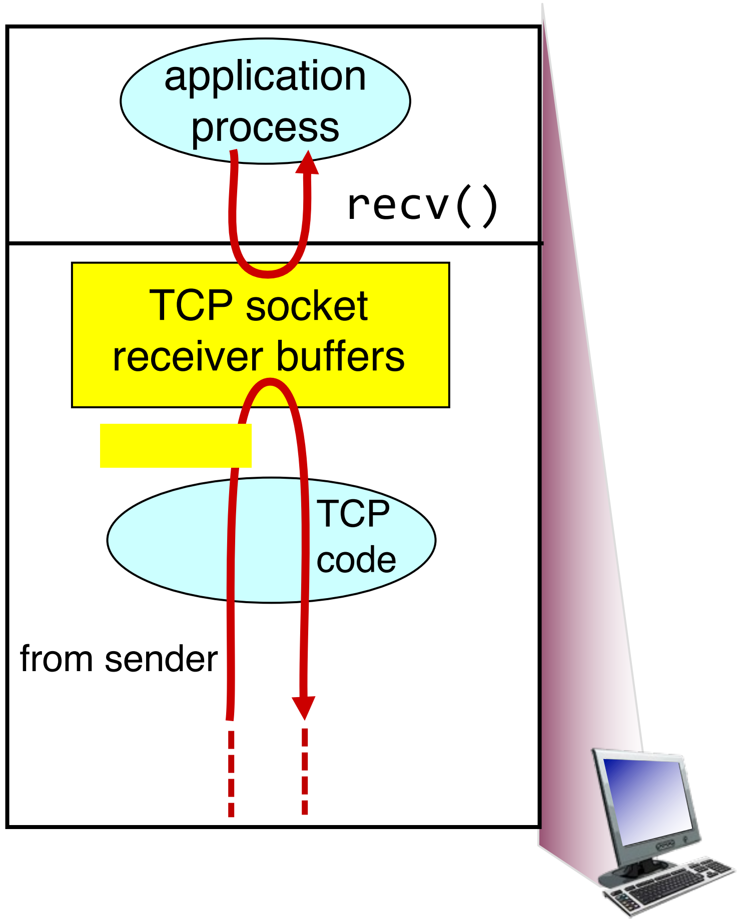
- Apps read from the receive-side socket buffer when you do a `recv()` call.

- 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()`



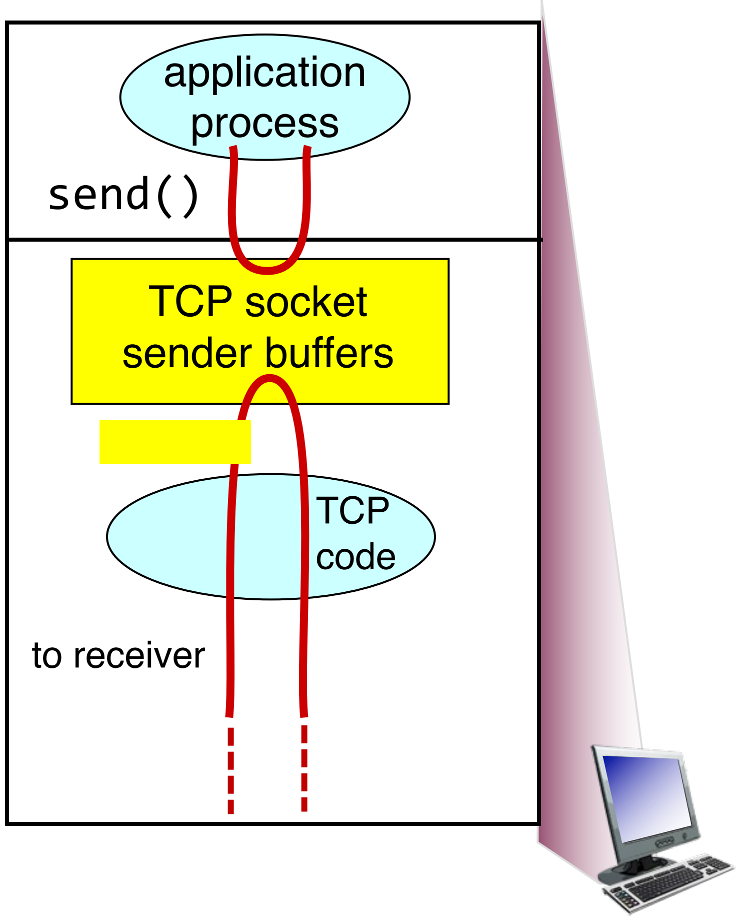
receiver TCP interaction

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.

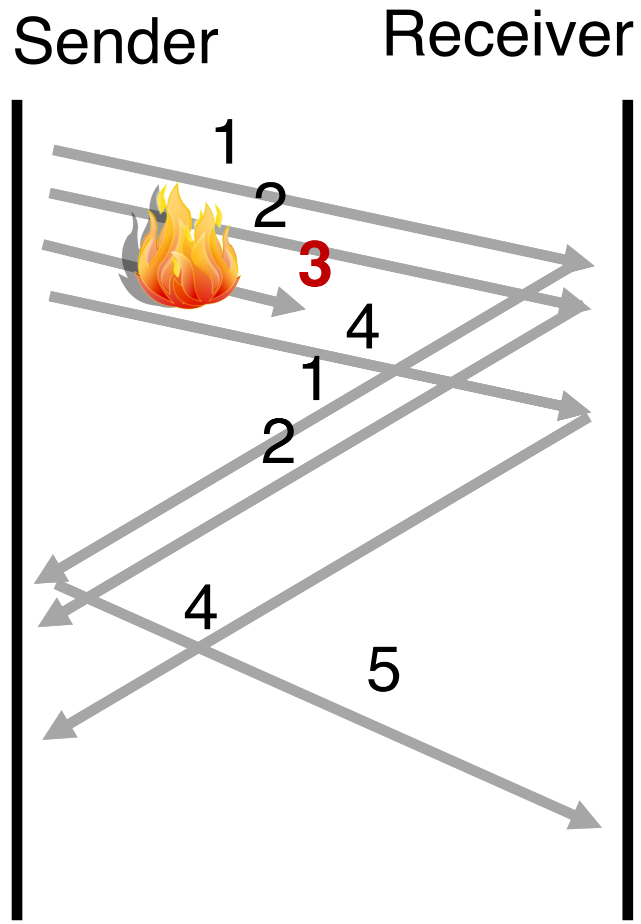


sender TCP interaction

Ordered Delivery

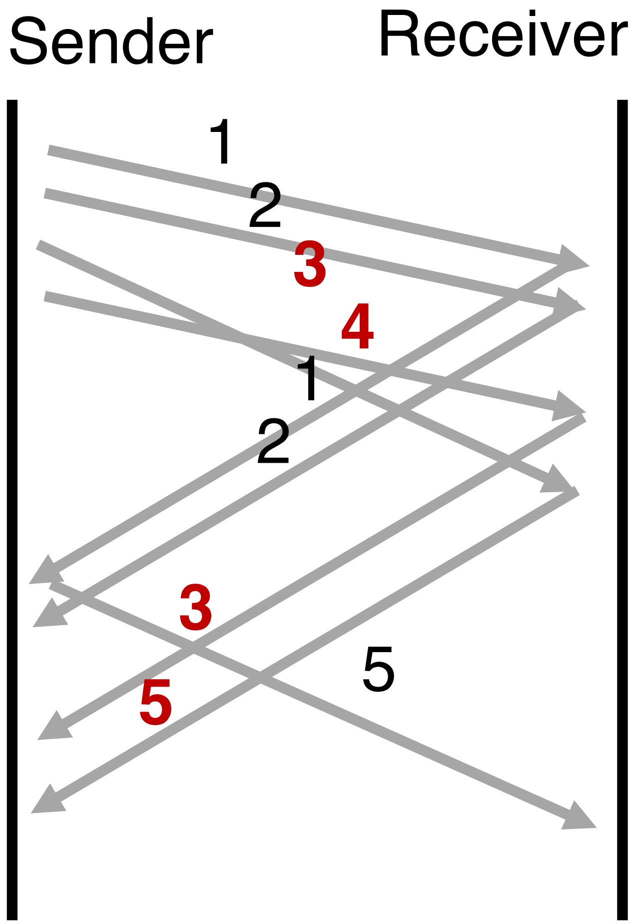
Reordering packets at the receiver side

- 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)?



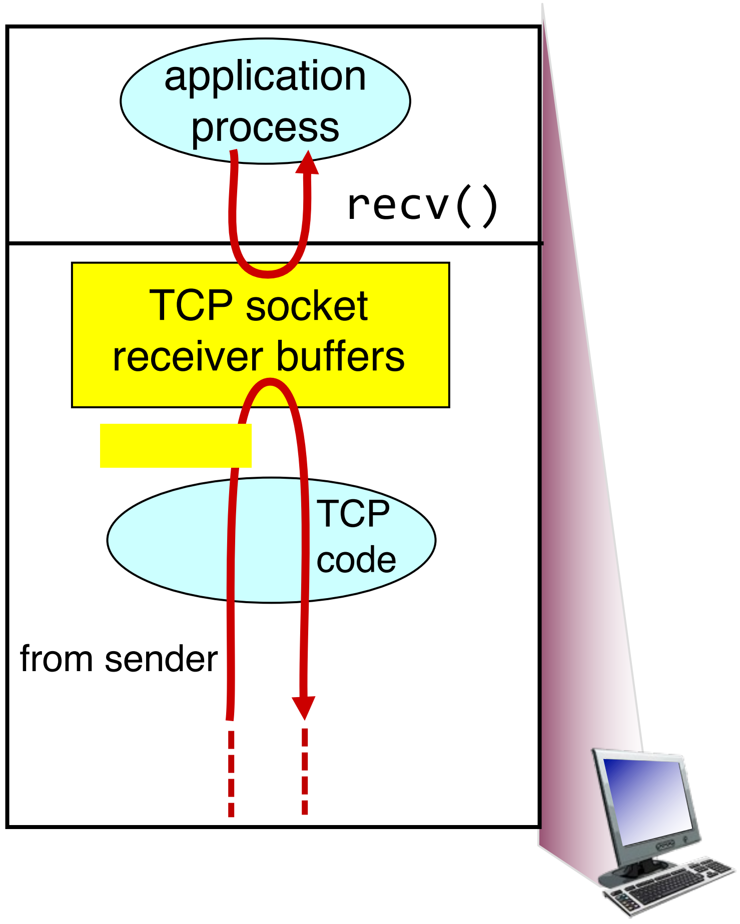
Reordering packets at the receiver side

- 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
- 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!



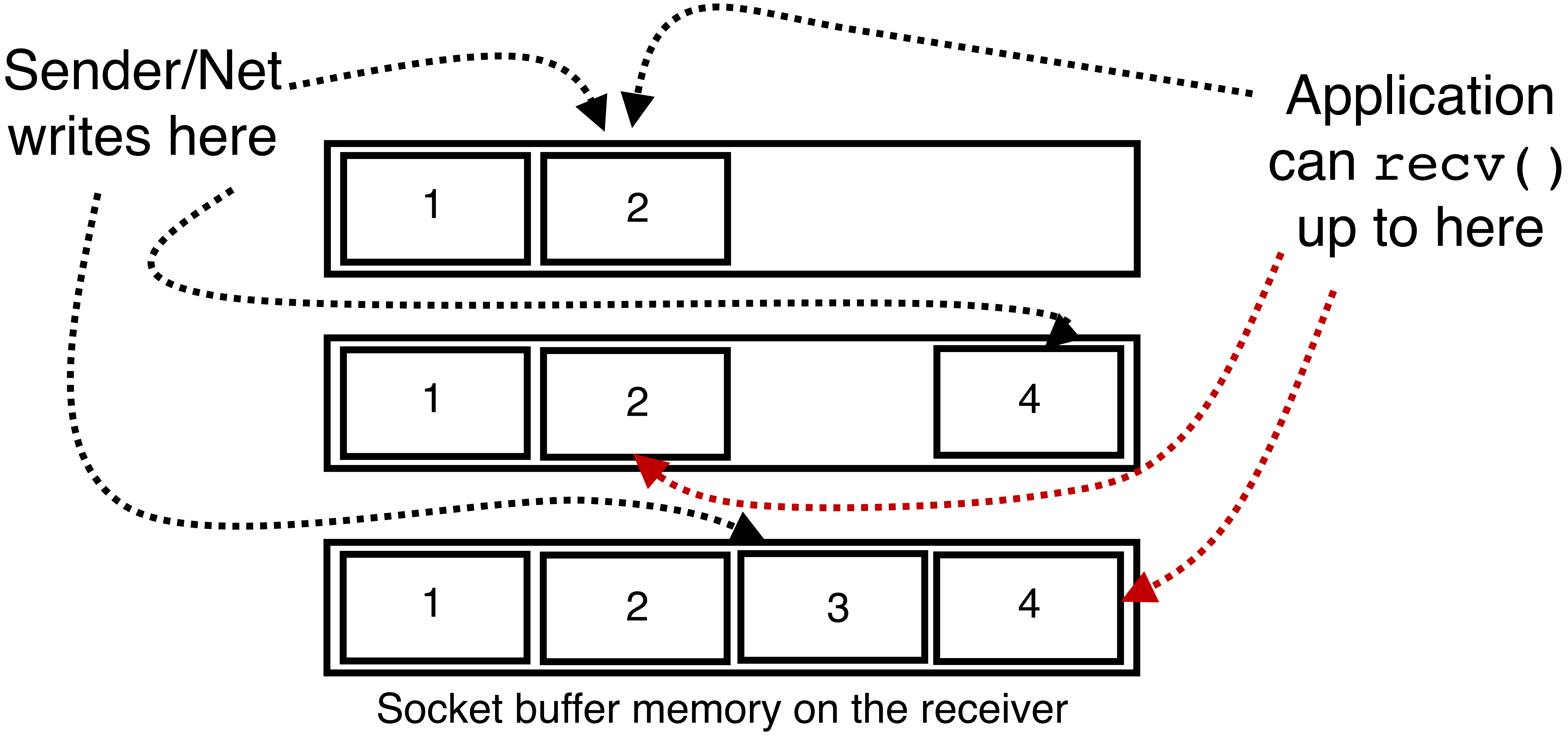
Receive-side app and TCP

- 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!)