Introduction to Computer Networks

CS640

# TCP Reliability Support

https://pages.cs.wisc.edu/~mgliu/CS640/F22/

**Ming Liu**
**mgliu@cs.wisc.edu**

# Today

## Last lecture

- How to tear down the TCP connection?

## Today

- How to ensure reliable data delivery?

## Announcements

- Lab4 is due 12/02/2022, 11:59 PM
- Final exam: Dec 17, 2022 5:05 PM – 7:05 PM

# Q: What is the goal of TCP reliability mechanisms?

**Q: What is the goal of TCP reliability mechanisms?**

**A: Byte stream @sender = Byte stream @receiver**

# Q: What is the goal of TCP reliability mechanisms?

# A: Byte stream @sender = Byte stream @receiver

- #1: TCP segments are delivered with no loss/duplication
- #2: TCP segments are delivered in order
- #3: The sender is not over-running the receiver capability

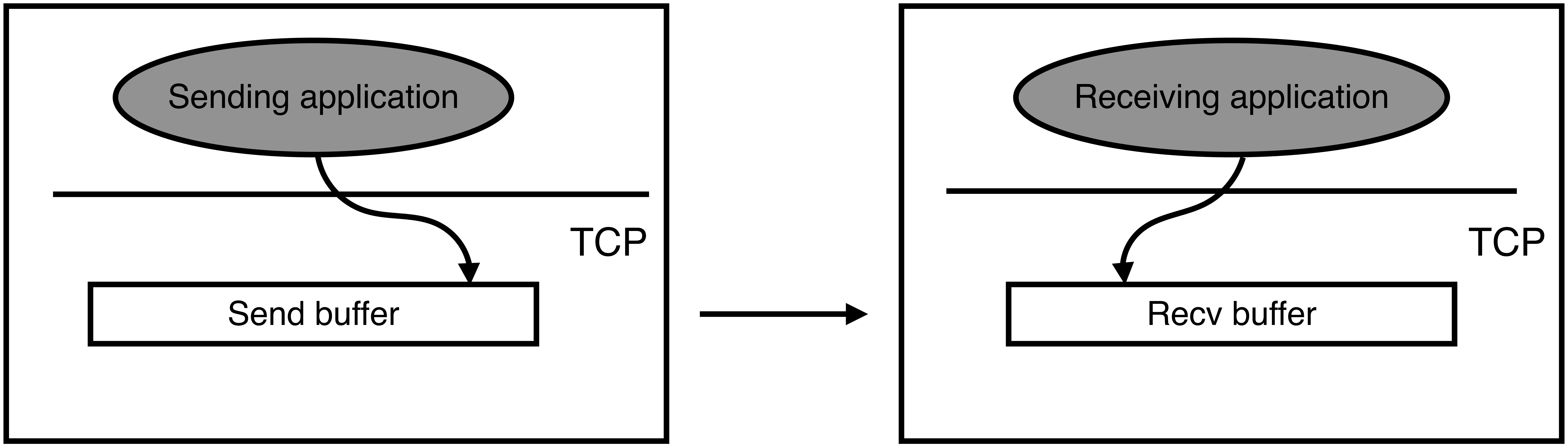
# Q: What is the goal of TCP reliability mechanisms?

## A: Byte stream @sender = Byte stream @receiver

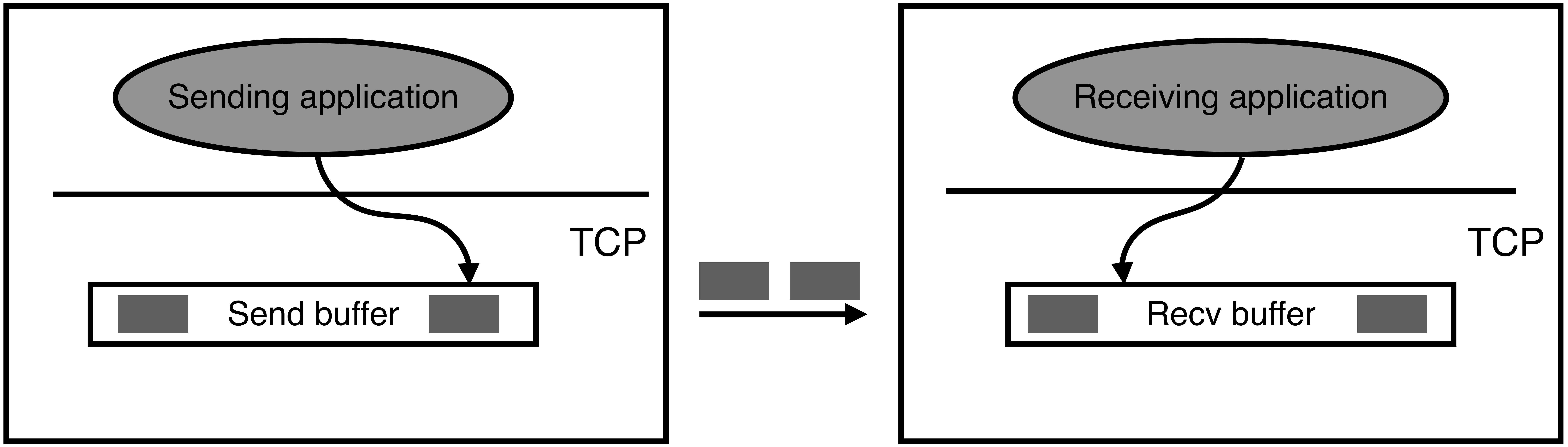
- #1: TCP segments are delivered with no loss/duplication
- #2: TCP segments are delivered in order
- #3: The sender is not over-running the receiver capability

TCP Segment: The smallest data transmission unit under TCP, consisting of a (segment) header and a data payload.

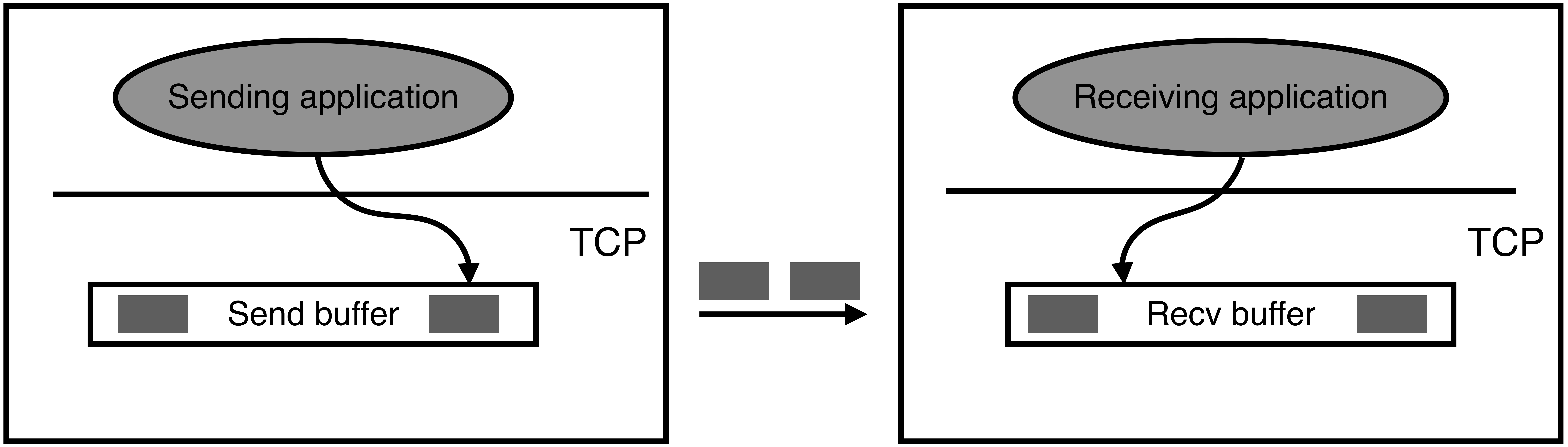
# A TCP Send/Recv Example



# A TCP Send/Recv Example

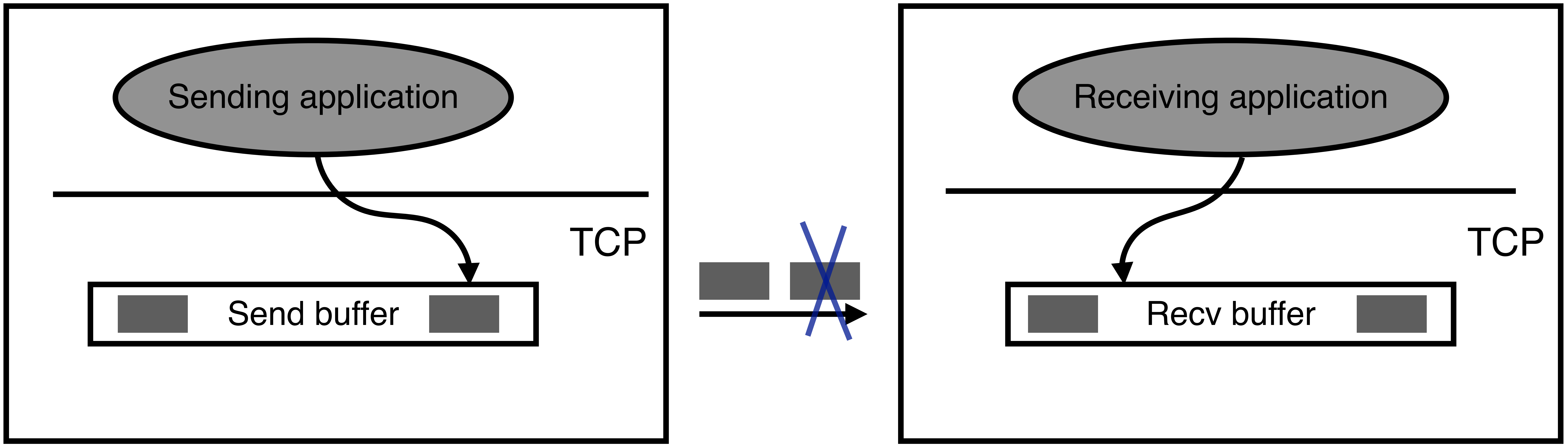


# A TCP Send/Recv Example




Send/Recv buffer is fixed sized
(i.e., MaxSendBuffer and MaxRcvBuffer)

# #1: How to deal with segment loss/duplication?



# #1: How to deal with segment loss/duplication?

L9

**Q3: How to ensure reliable frame delivery?**

**A: Two key ideas:**

- #1: Acknowledgment (ACK) — notify the sender of the receipt of a frame
- #2: Timeout — wait for a reasonable amount of time and generate a signal

**Where? When?**

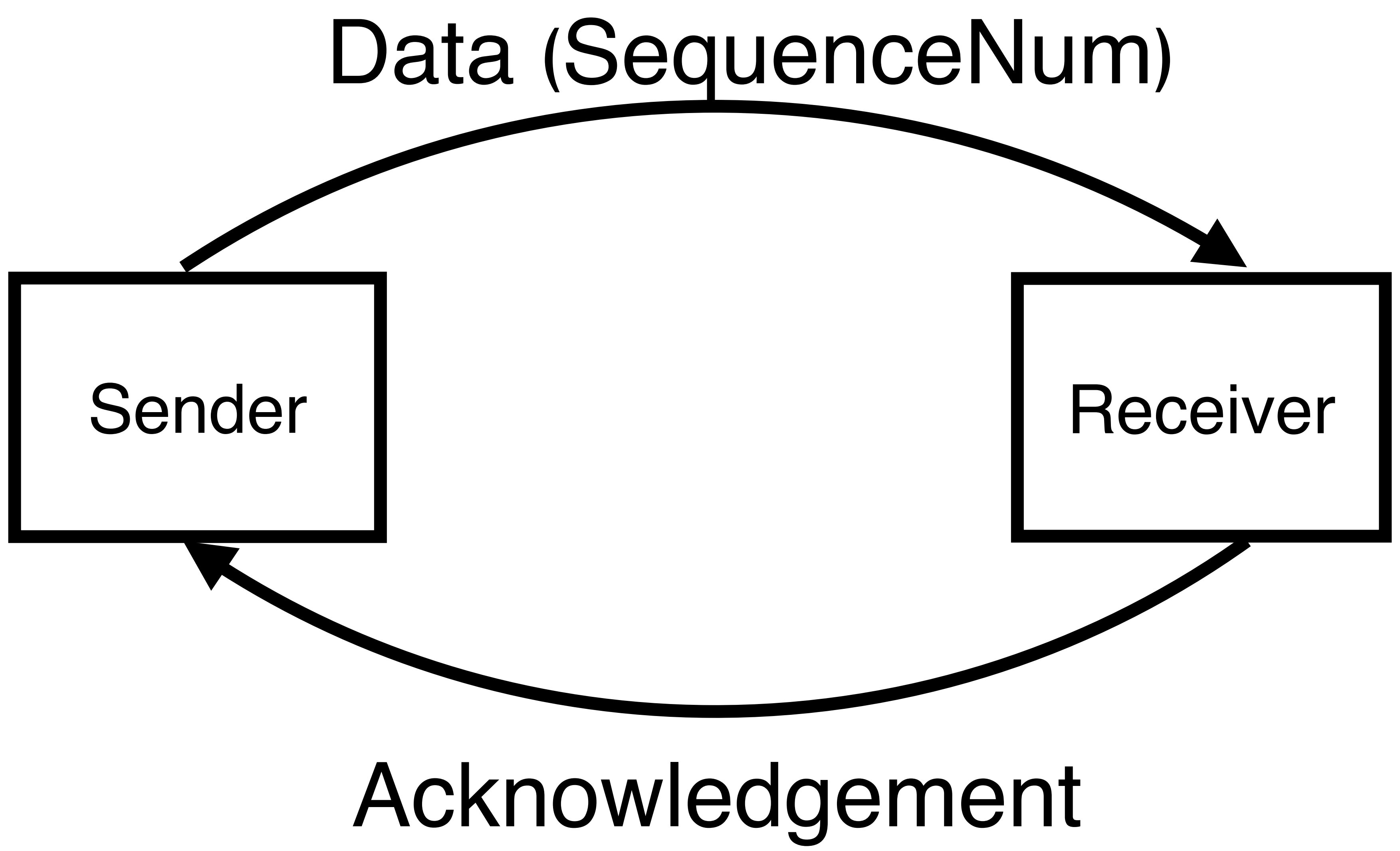
# Acknowledgment

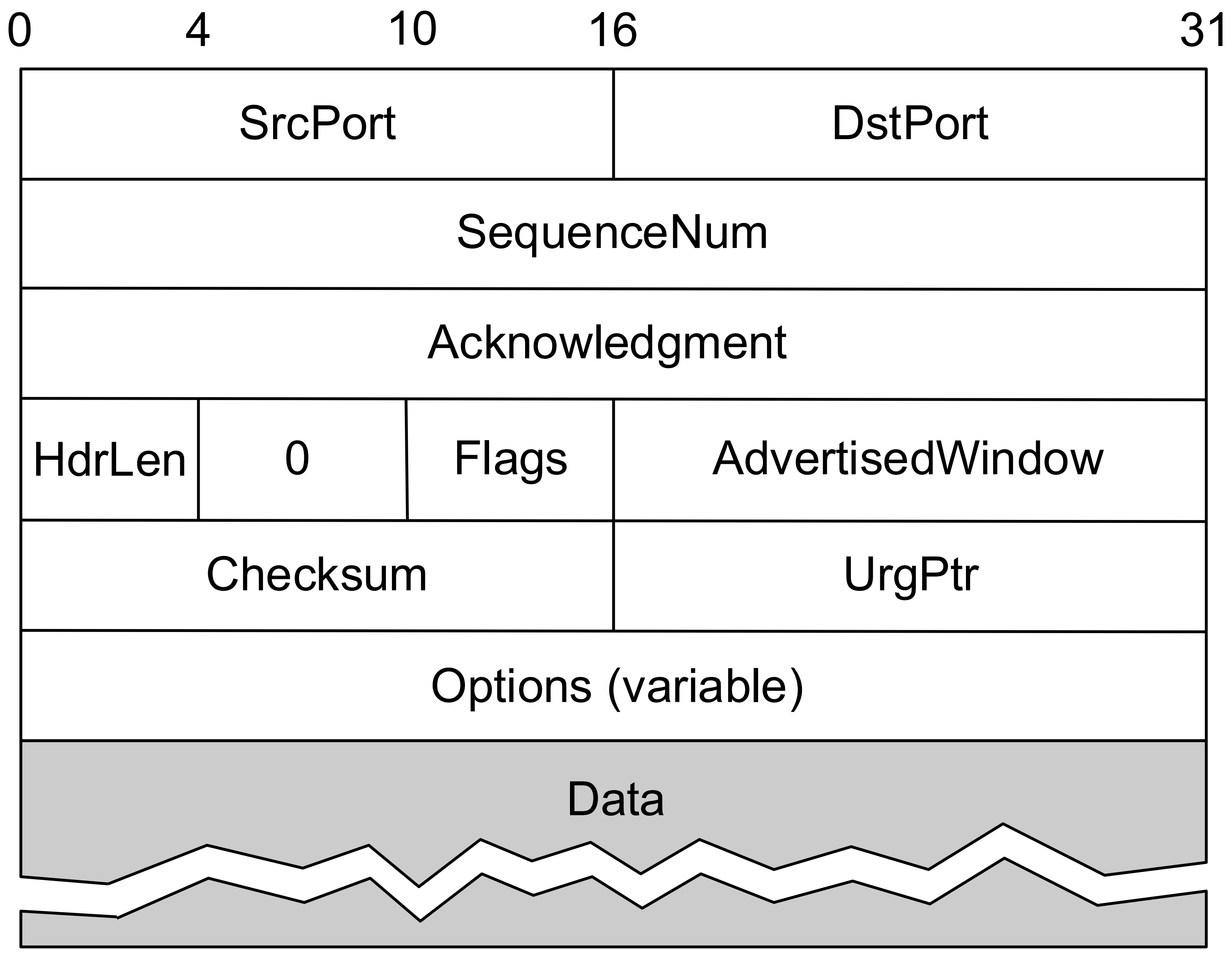
**An acknowledgment (ACK) is a packet sent by one host in response to a packet it has received**

# Acknowledgment

## An acknowledgment (ACK) is a packet sent by one host in response to a packet it has received

- Making a packet an ACK is simply a matter of changing a field in the transport header
- Data can be piggybacked in ACKs



# Timeout

## A timeout is a signal to a packet that was sent but has not received its ACK within a specified time frame

- The packet will be transmitted if a timeout is triggered

How are timers set?

# Timeout Setup

**RTT: the delay between transmission and receipt of packets between hosts**

**RTT can be used to estimate the timeout period**

# EWMA for RTT Estimation

## EWMA: exponentially weighted moving average

# EWMA for RTT Estimation

**EWMA: exponentially weighted moving average**

**#1: Measure SampleRTT for each packet/ACK pair**

# EWMA for RTT Estimation

## EWMA: exponentially weighted moving average

## #1: Measure SampleRTT for each packet/ACK pair

## #2: Compute the weighted average of RTT

- EstimateRTT = alpha x EstimateRTT + beta x SampleRTT, where alpha+beta = 1
- 0.8 <= alpha <= 0.9
- 0.1 <= beta <= 0.2

# EWMA for RTT Estimation

## EWMA: exponentially weighted moving average

## #1: Measure SampleRTT for each packet/ACK pair

## #2: Compute the weighted average of RTT

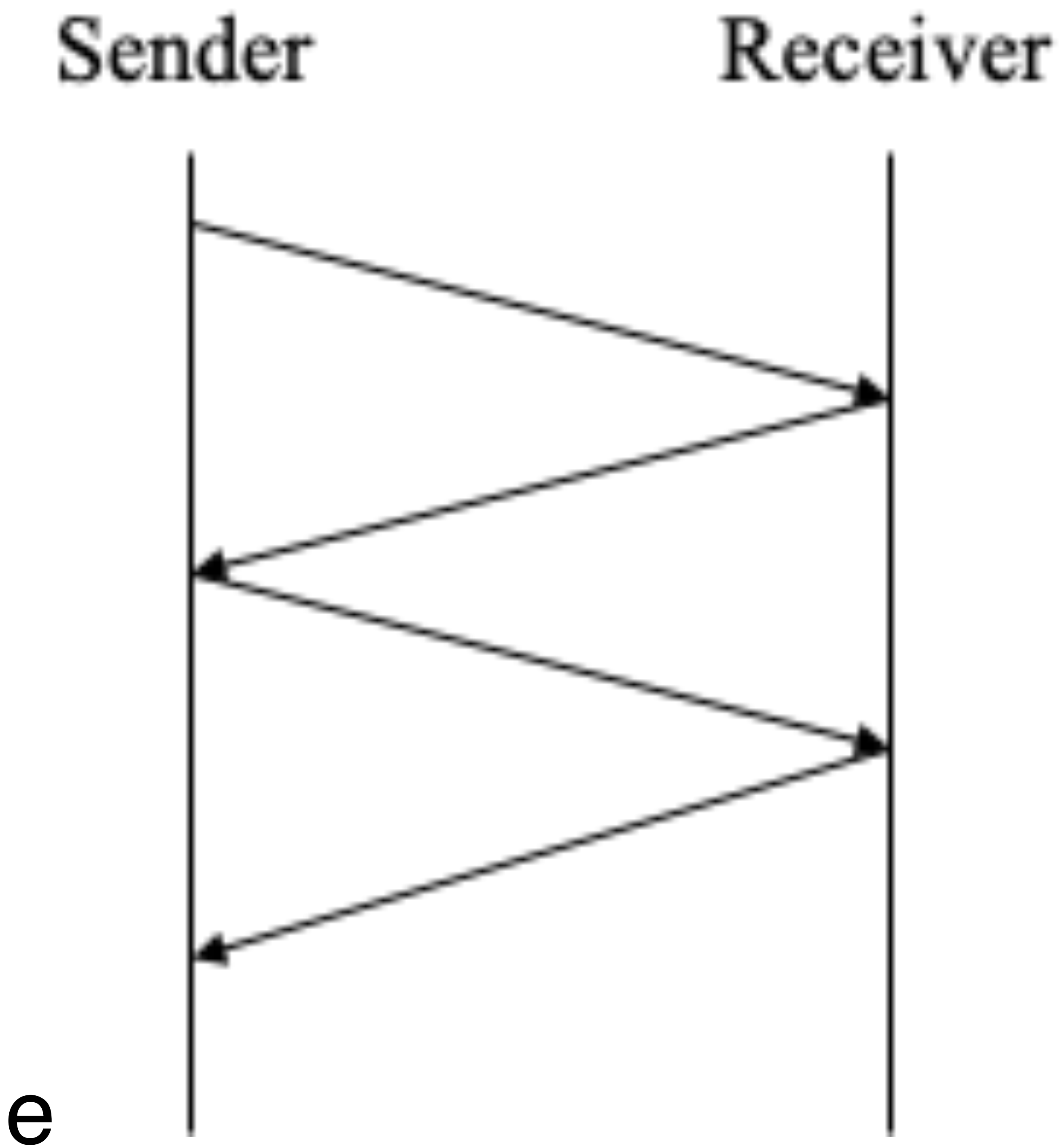
- EstimateRTT = alpha x EstimateRTT + beta x SampleRTT, where alpha+beta = 1
- 0.8 <= alpha <= 0.9
- 0.1 <= beta <= 0.2

## #3: Set timeout based on EstimateRTT

- TimeOut = 2 x EstimateRTT

# Stop-and-Wait Revisited

## Send the next packet only if the last one is successfully delivered




## Acknowledgment

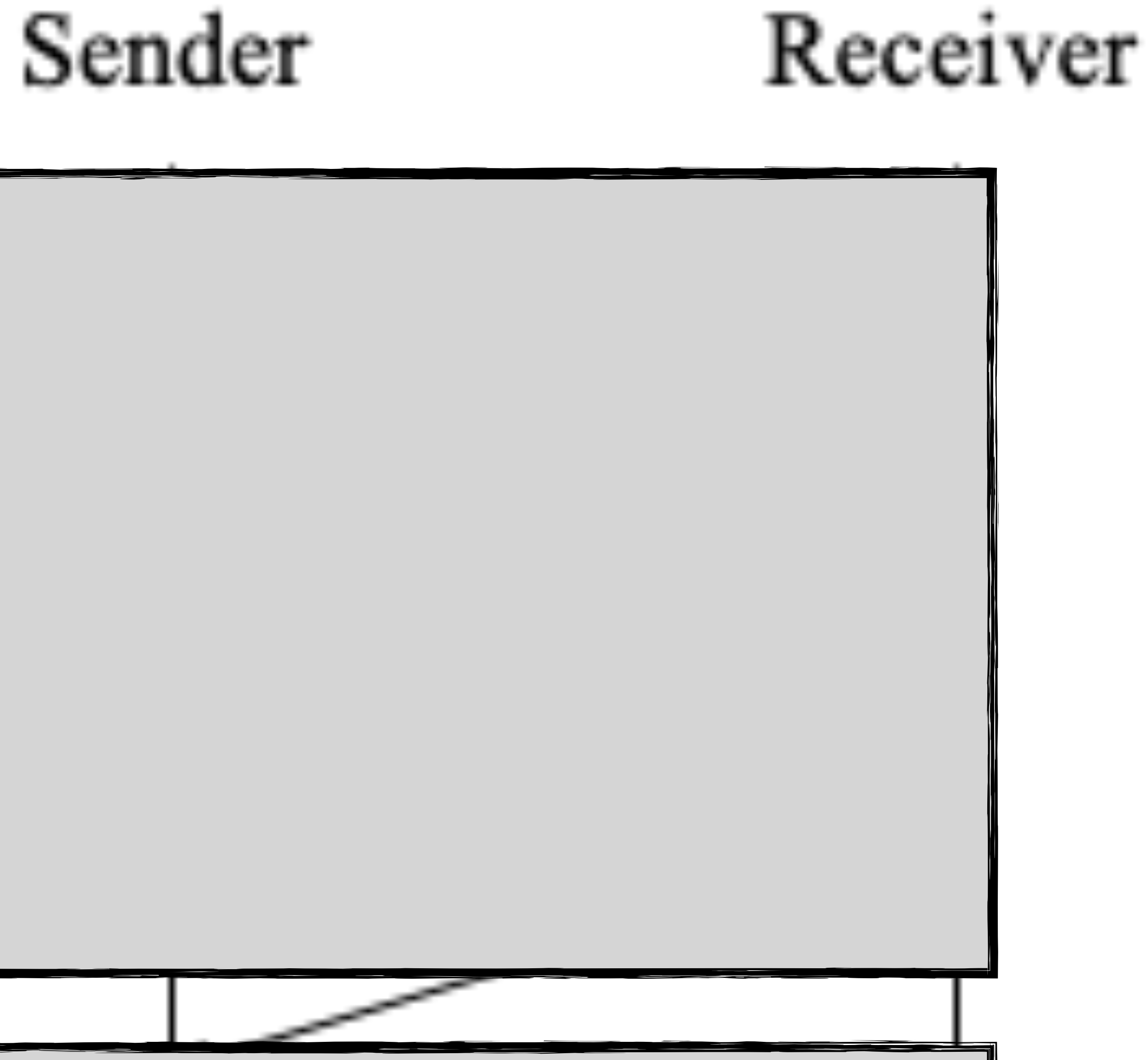
- Where: receiver
- When: after a valid packet is being received

## Timeout

- Where: sender
- When: after the issuing packet not being ACKed for a certain time

# Stop-and-Wait Revisited

## Send the next packet only if the last one is successfully delivered




### Benefits:

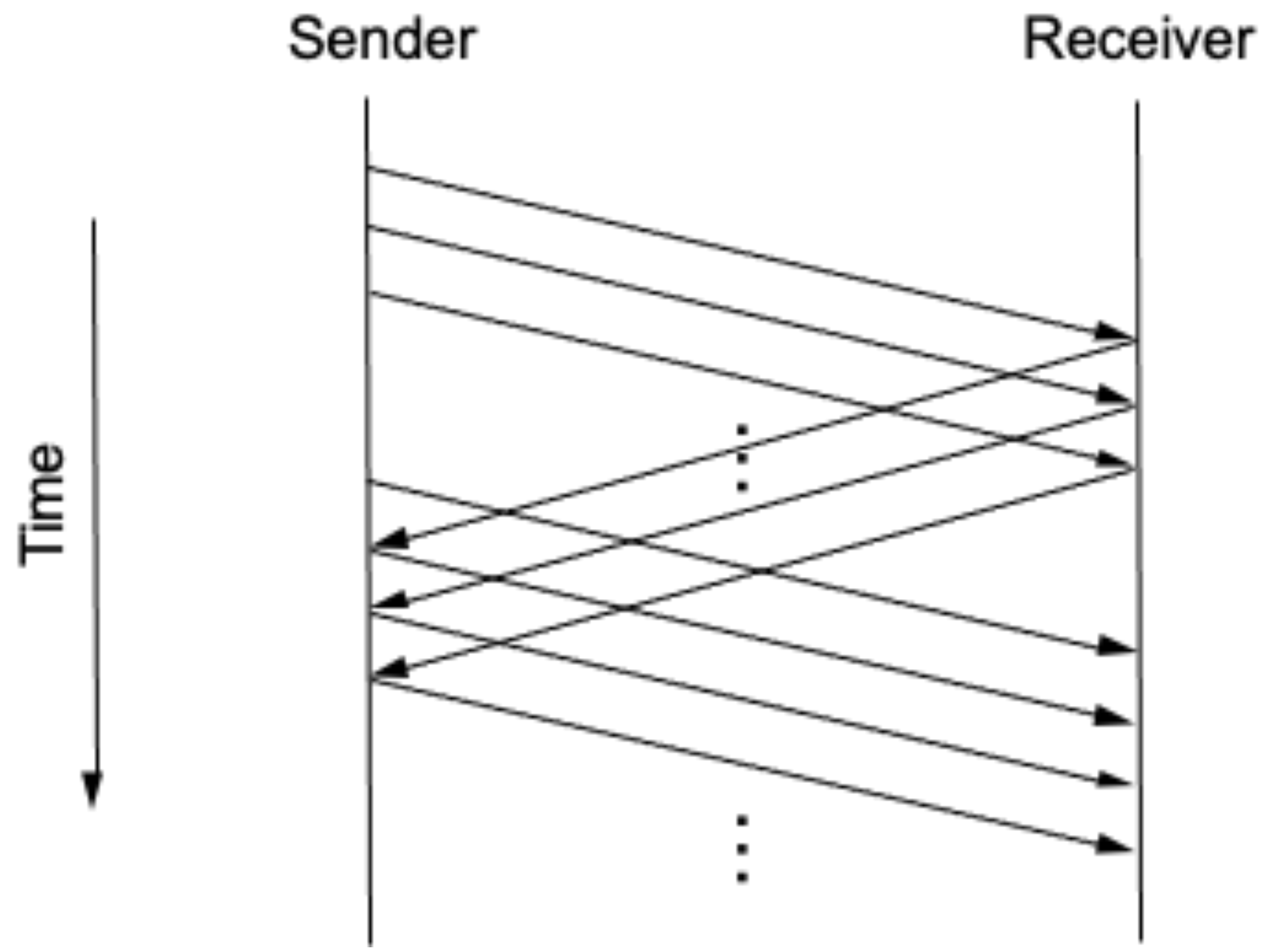
- Sequence numbers help avoid duplicated packets
- No re-ordering: one outgoing packet at a time
- The receiver is not overwhelmed

### Problem:

- How to keep the communication channel full?

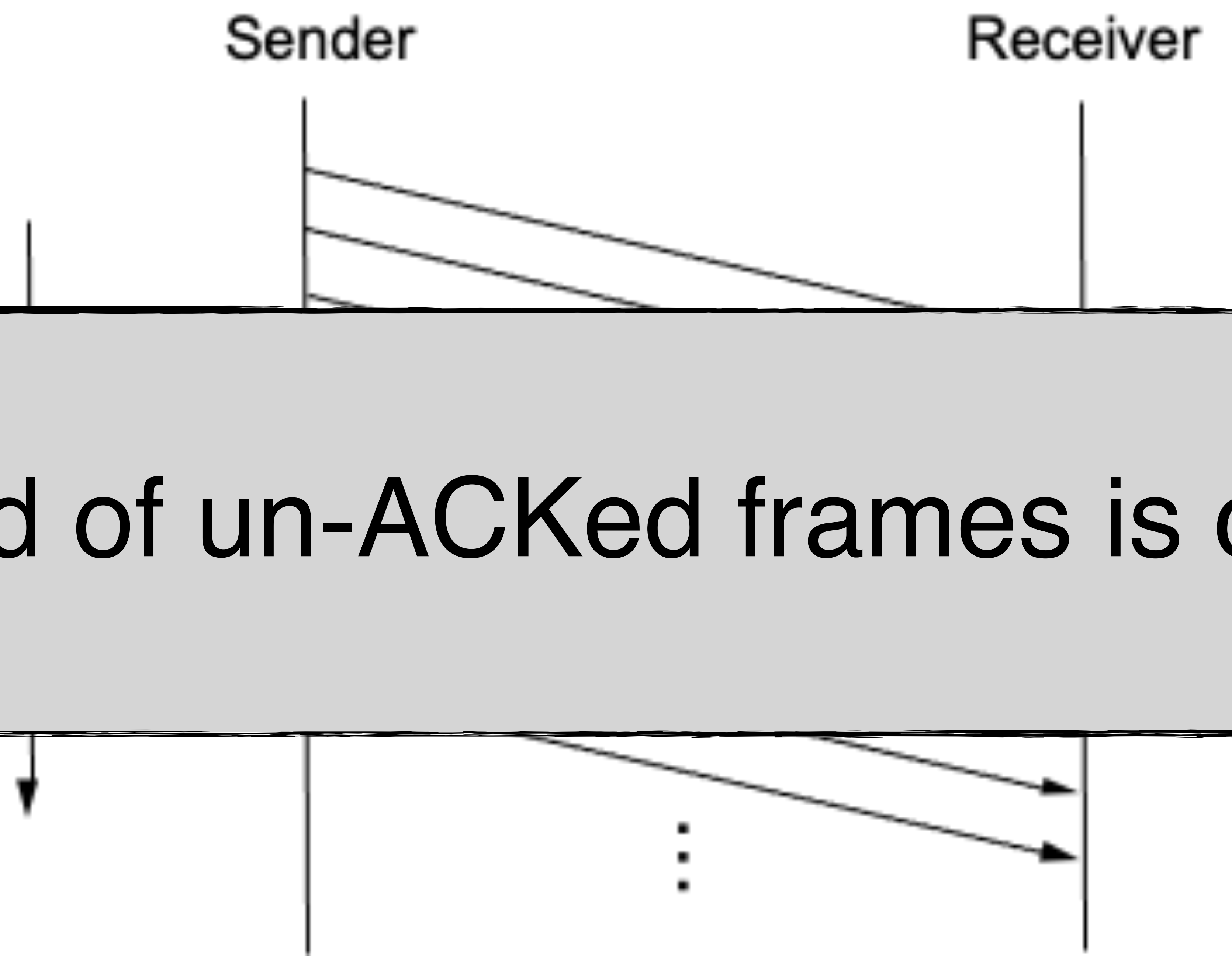
# Solution: Sliding Window

## Allow multiple outstanding (un-ACKed) frames



# Solution: Sliding Window

## Allow multiple outstanding (un-ACKed) frames




The upper bound of un-ACKed frames is called a **window**

# Buffering requirements on Sender and Receiver

**#1: The sender needs to buffer data so that if data is lost, it can be resent**

**#2: The receiver needs to buffer data so that if data is received out of order, it can be held until all packets are received**

# Sliding Window — Sender

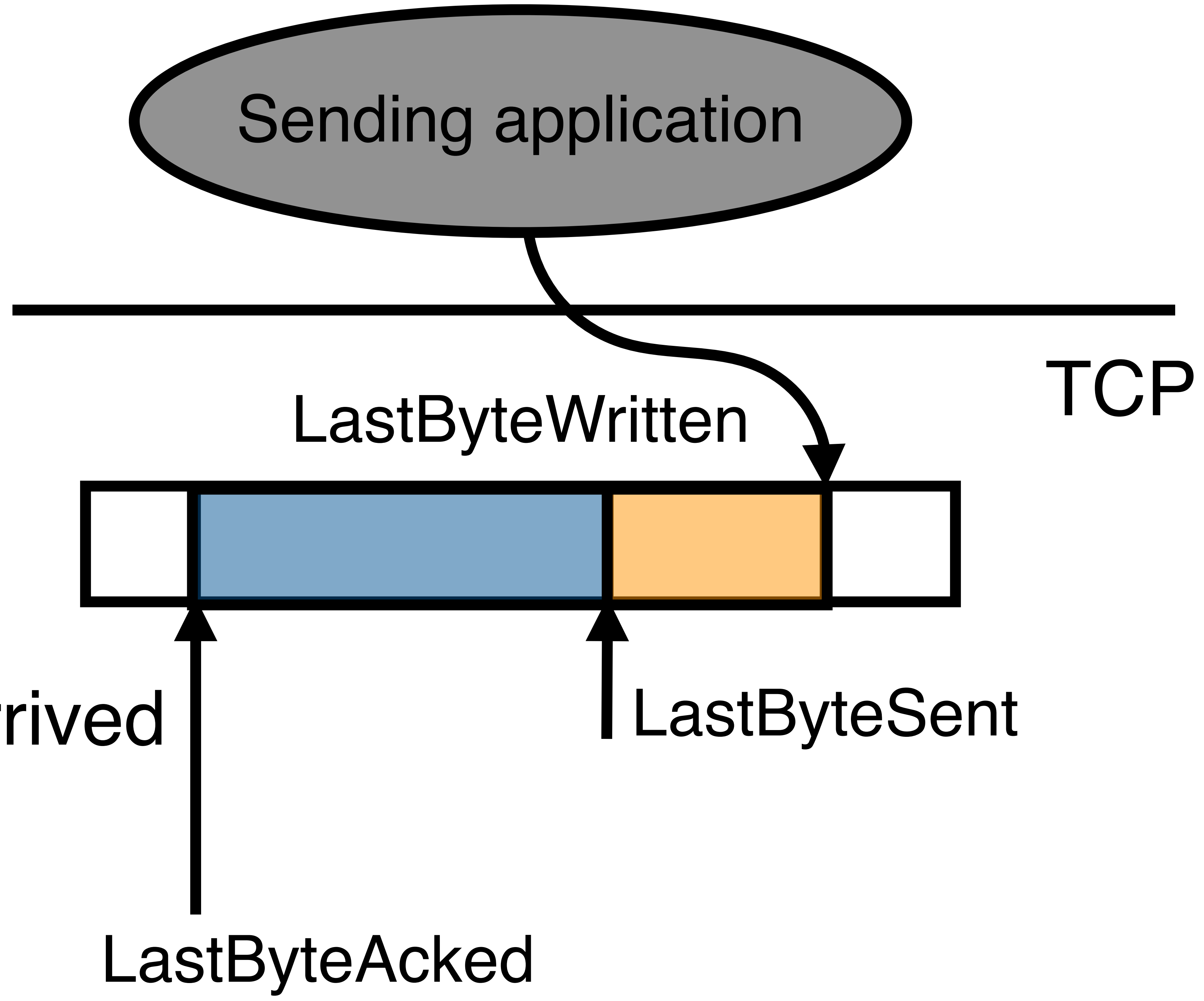
## Assign sequence number to each segment (SeqNum)

## Maintain three state variables:

- Last byte written by the application (**LastByteWritten**)
- Last byte being acknowledged (**LastByteAcked**)
- Last byte sent (**LastByteSent**)

## Three variables manipulation:

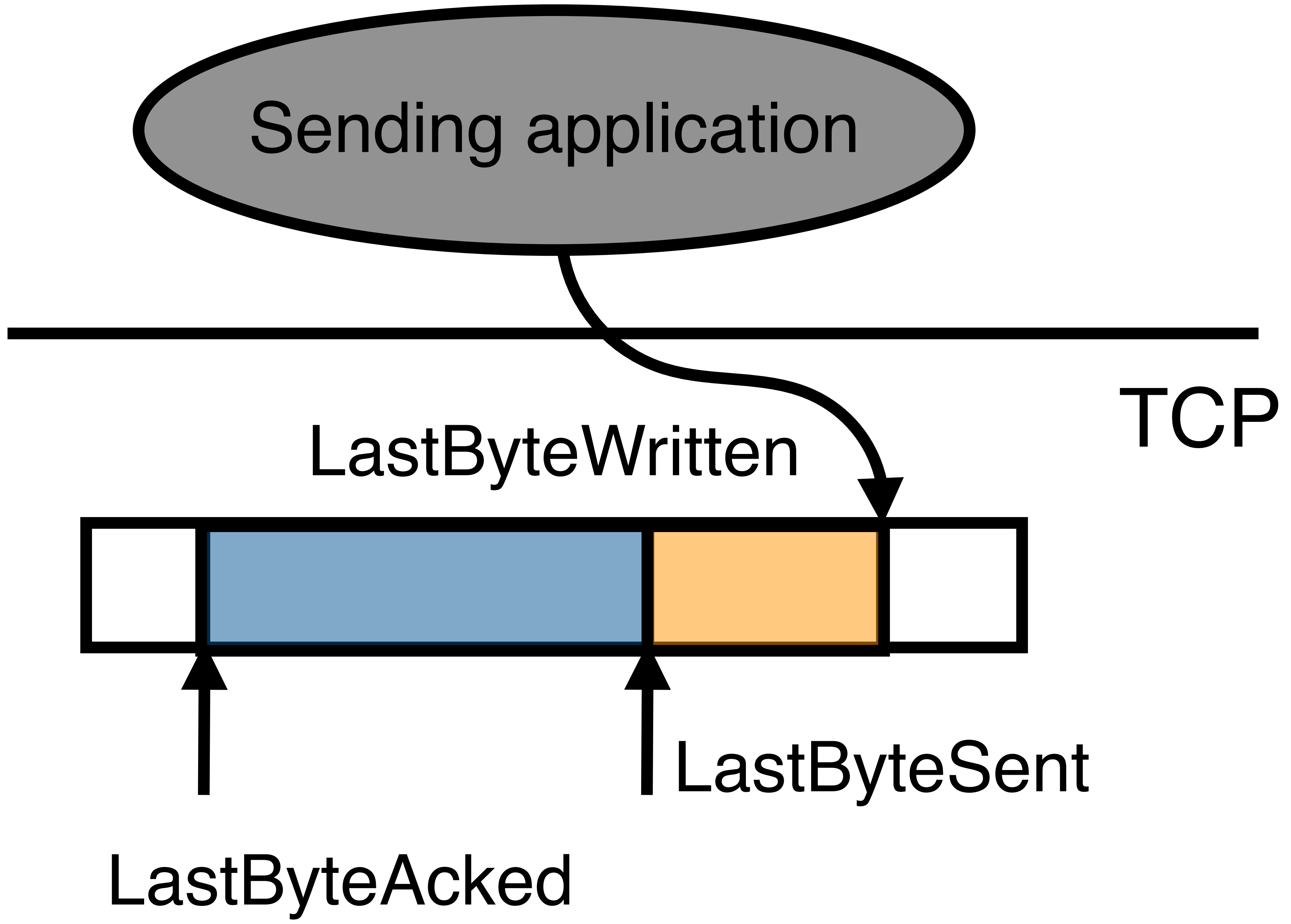
- Advance **LastByteWritten** when an app writes
- Advance **LastByteAcked** when a consecutive ACK arrived
- Advance **LastByteSent** when the segments are sent



# Sliding Window Invariants @Sender

## Invariants:

- LastByteSent <=LastByteWritten
- LastByteAcked <= LastByteSent



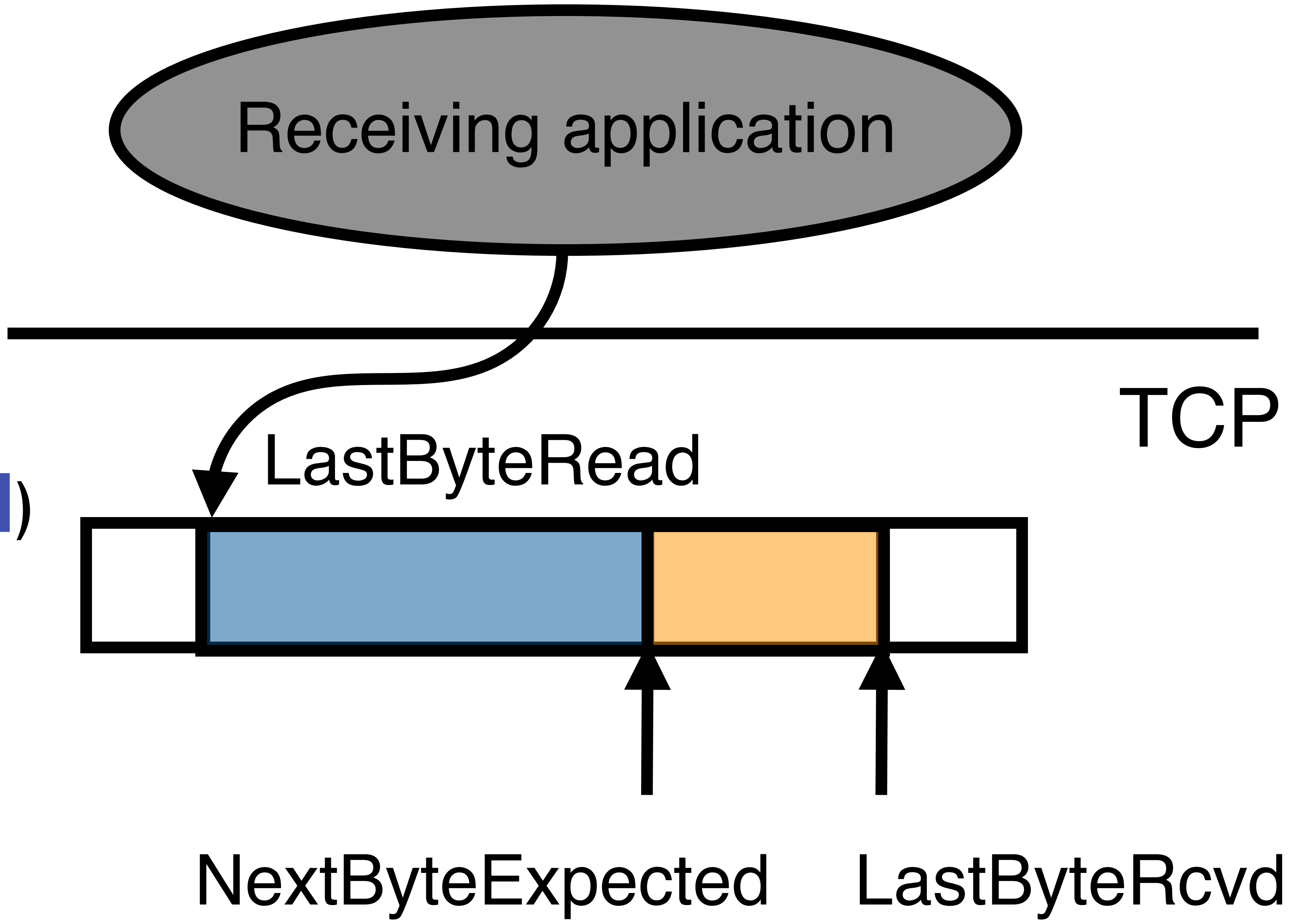

## Buffered bytes:

- |LastByteWritten - LastByteAcked| <= MaxSendBuffer

# Sliding Window — Receiver

## Maintain three state variables:

- Last byte read by the application (**LastByteRead**)
- Last byte received (**LastByteRcvd**)
- Next byte supposed to be received (**NextByteExpected**)



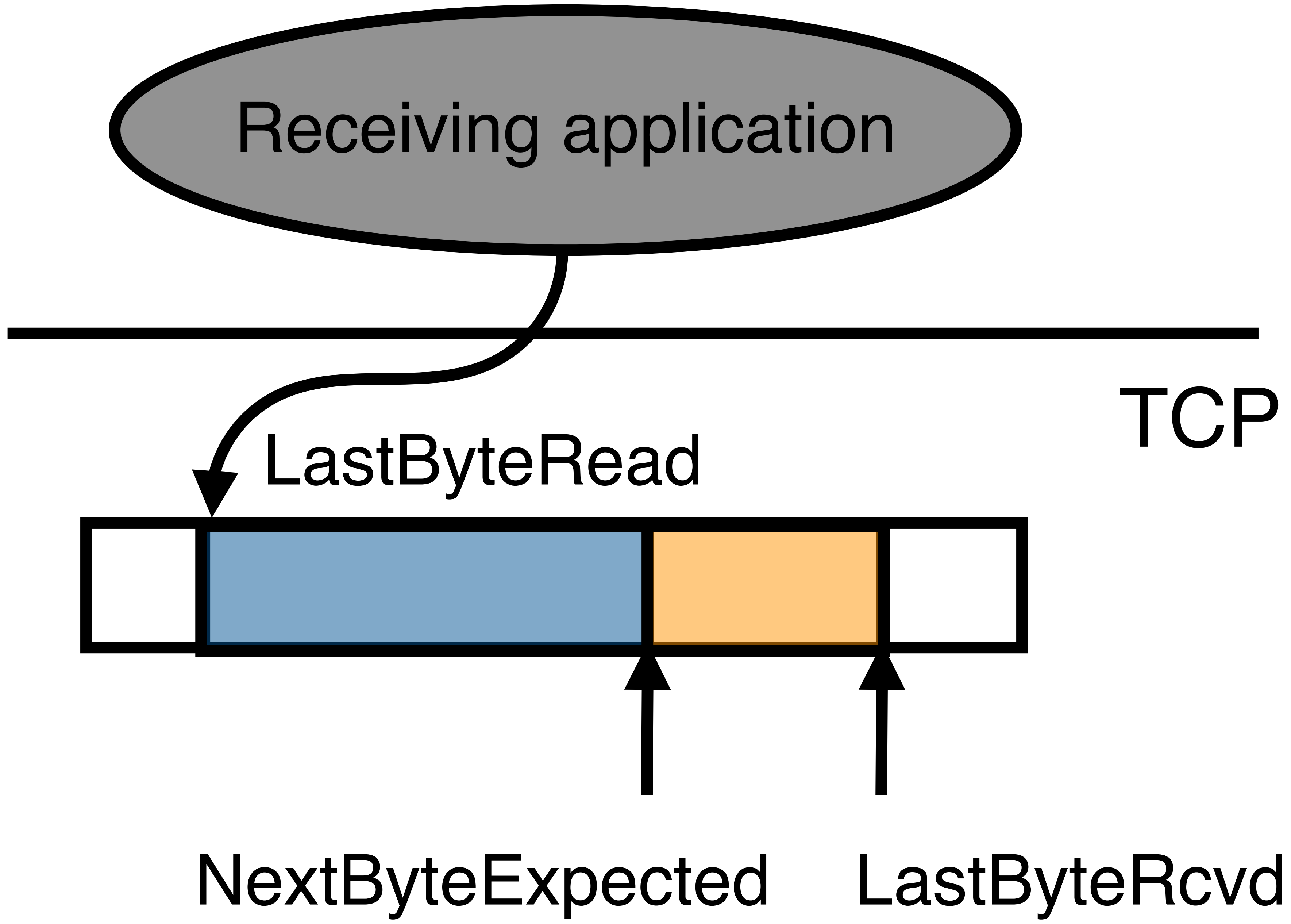

## Three variables manipulation:

- Advance **LastByteRead** when an app reads
- Advance **LastByteRcvd** when the segment is received
- Advance **NextByteExpected** when the next expected segment is received

# Sliding Window Invariants @Receiver

## Invariants:

- LastByteRead < NextByteExpected
- NextByteExpected <= LastByteRcvd + 1




## Buffered bytes:

- |LastByteRcvd - LastByteRead| <= MaxRcvBuffer

# #2: How to deal with the out-of-order delivery?

# #2: How to deal with the out-of-order delivery?

## Sliding Window @ Receiver

- If **SeqNum** of a segment is within the range [**LastByteRead**, **LastByteRcvd**] –> accept
- If **SeqNum** of a segment is less than **LastByteRead** –> discarded
- If a segment causes the required buffer data larger than **MaxRcvBuffer** –> discarded
  - **LastByteRcvd** - **LastByteRead** > **MaxRcvBuffer**

## Data is delivered only if there is a continuous byte stream without gap

# #3: How to not over-run the receiver?

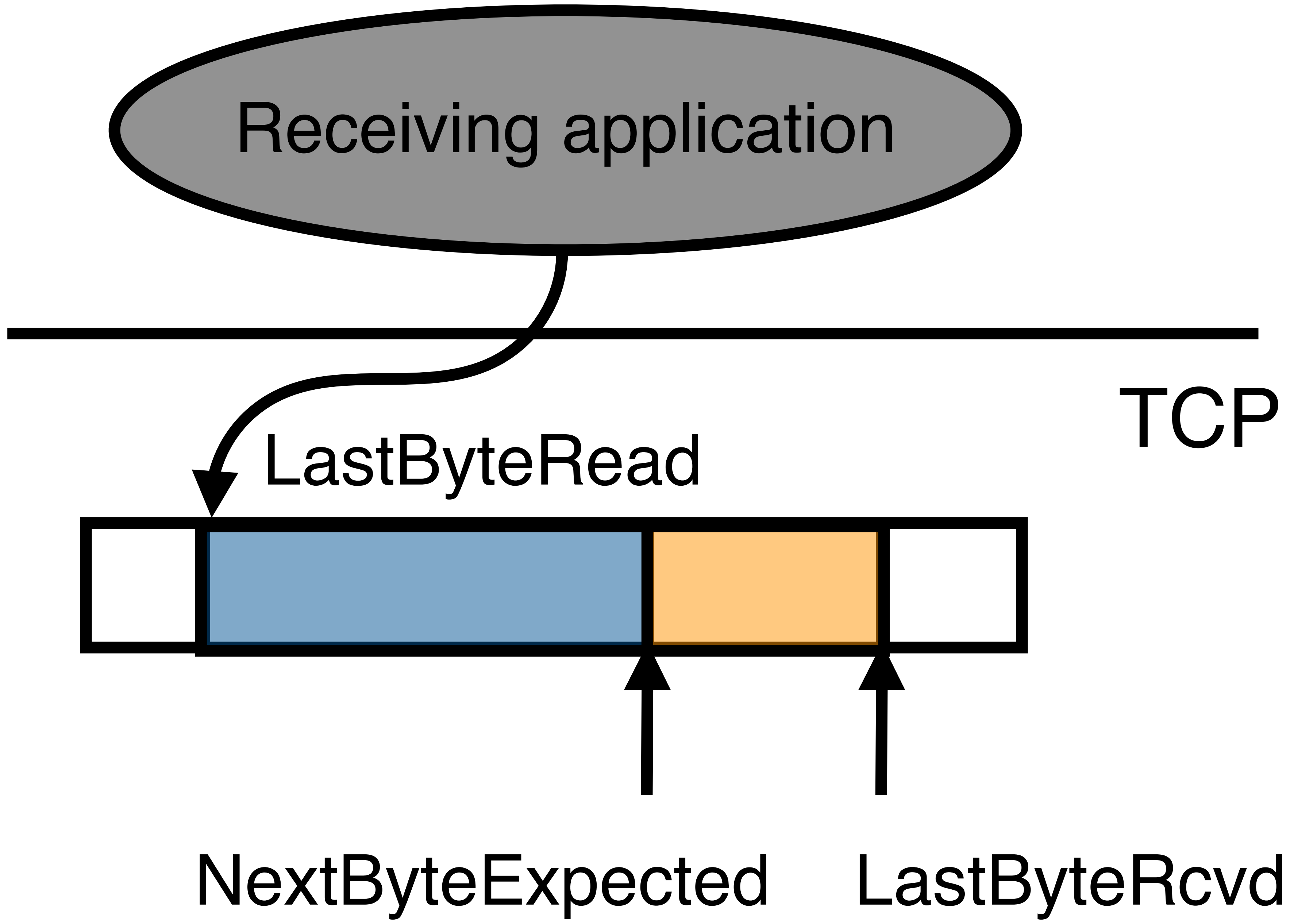
## Flow control

- Prevent server from overflowing the receiver's buffer

@Receiver: **LastByteRcvd** - **LastByteRead** <= **MaxRcvBuffer**

# Solution: the receiver advertises this window

LastByteRcvd - LastByteRead <= MaxRcvBuffer

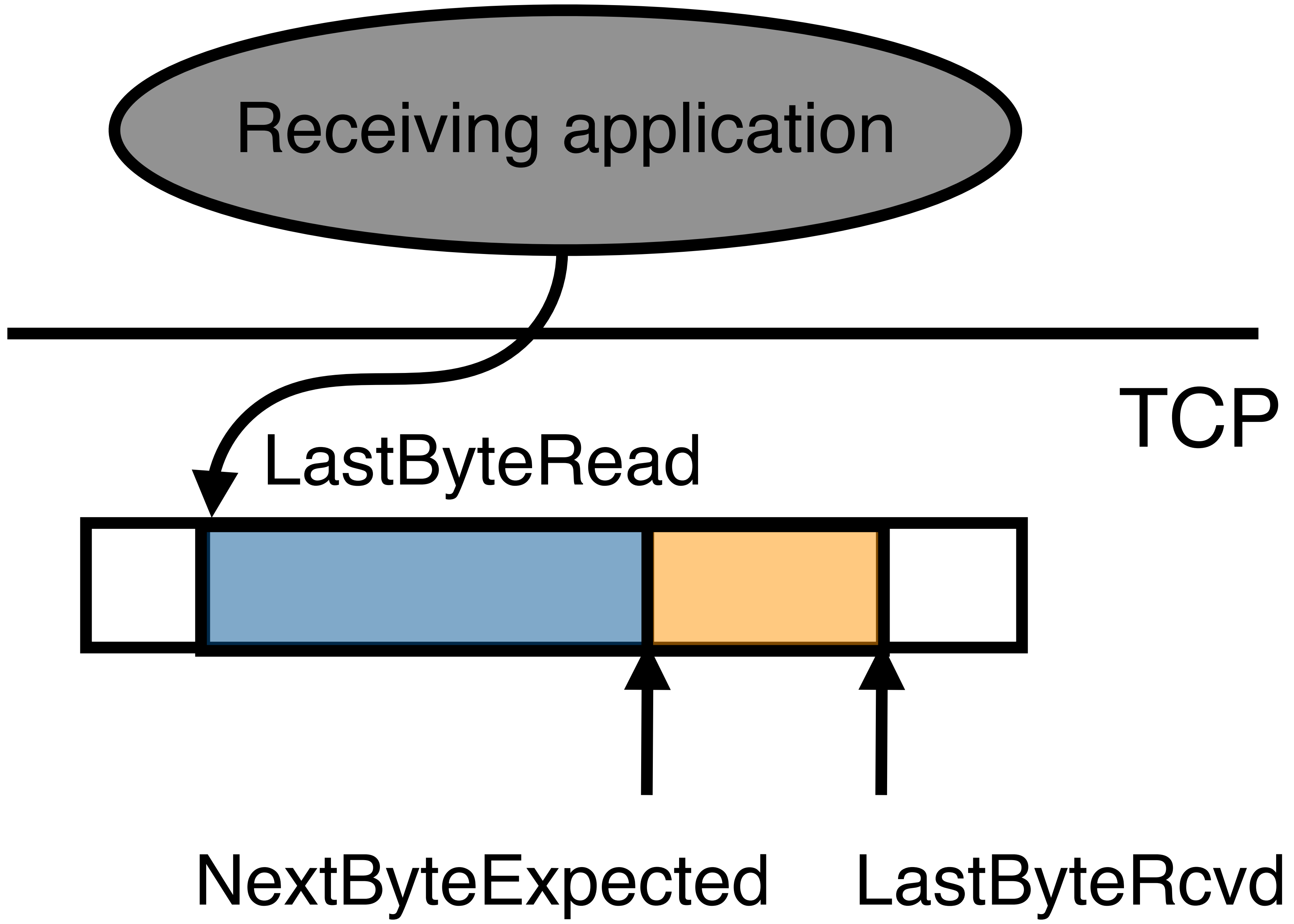


# Solution: the receiver advertises this window

**LastByteRcvd** - **LastByteRead** <= **MaxRcvBuffer**

⇓

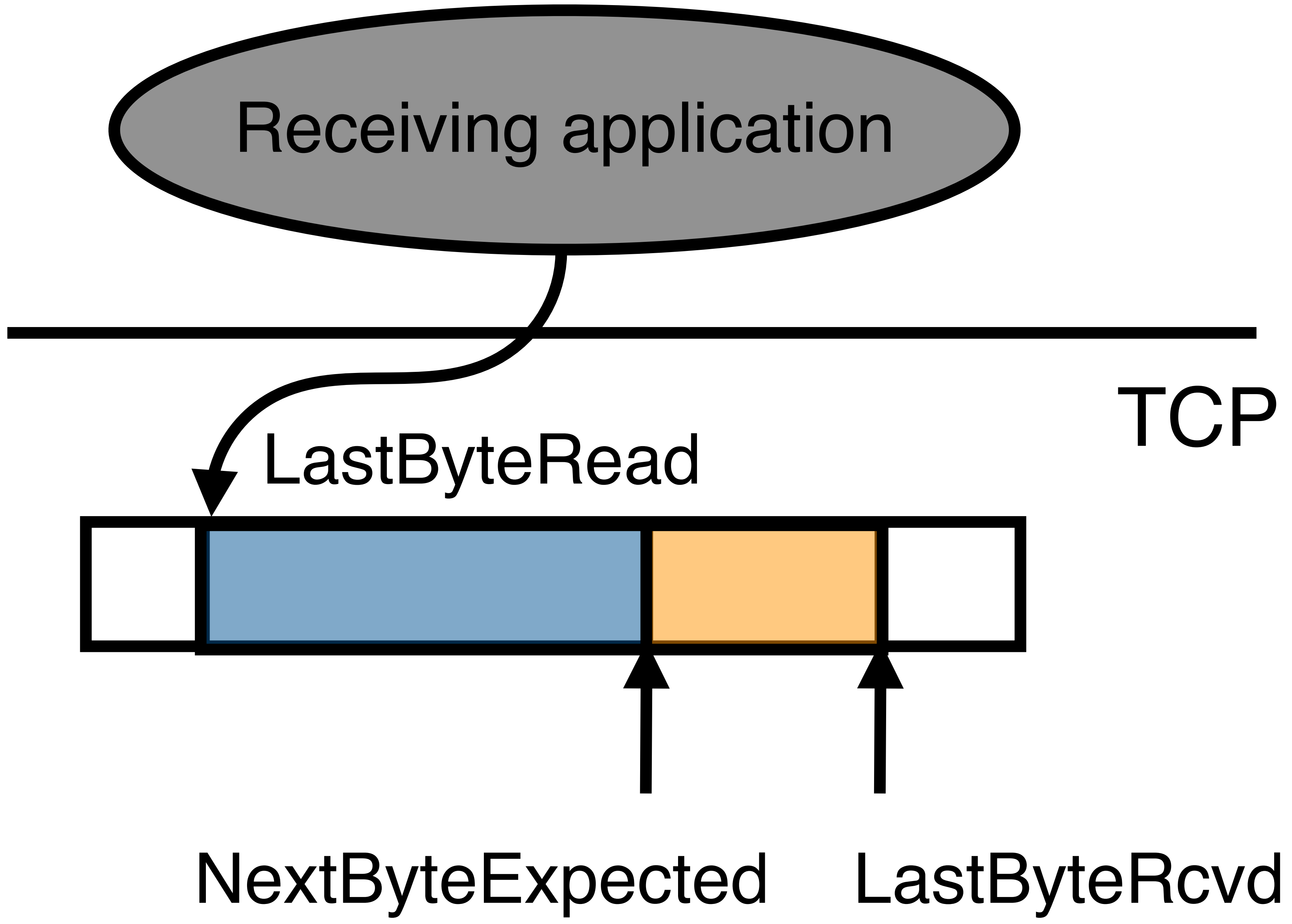
**AdvertisedWindow** = **MaxRcvBuffer** - (**LastByteRcvd** - **LastByteRead**)



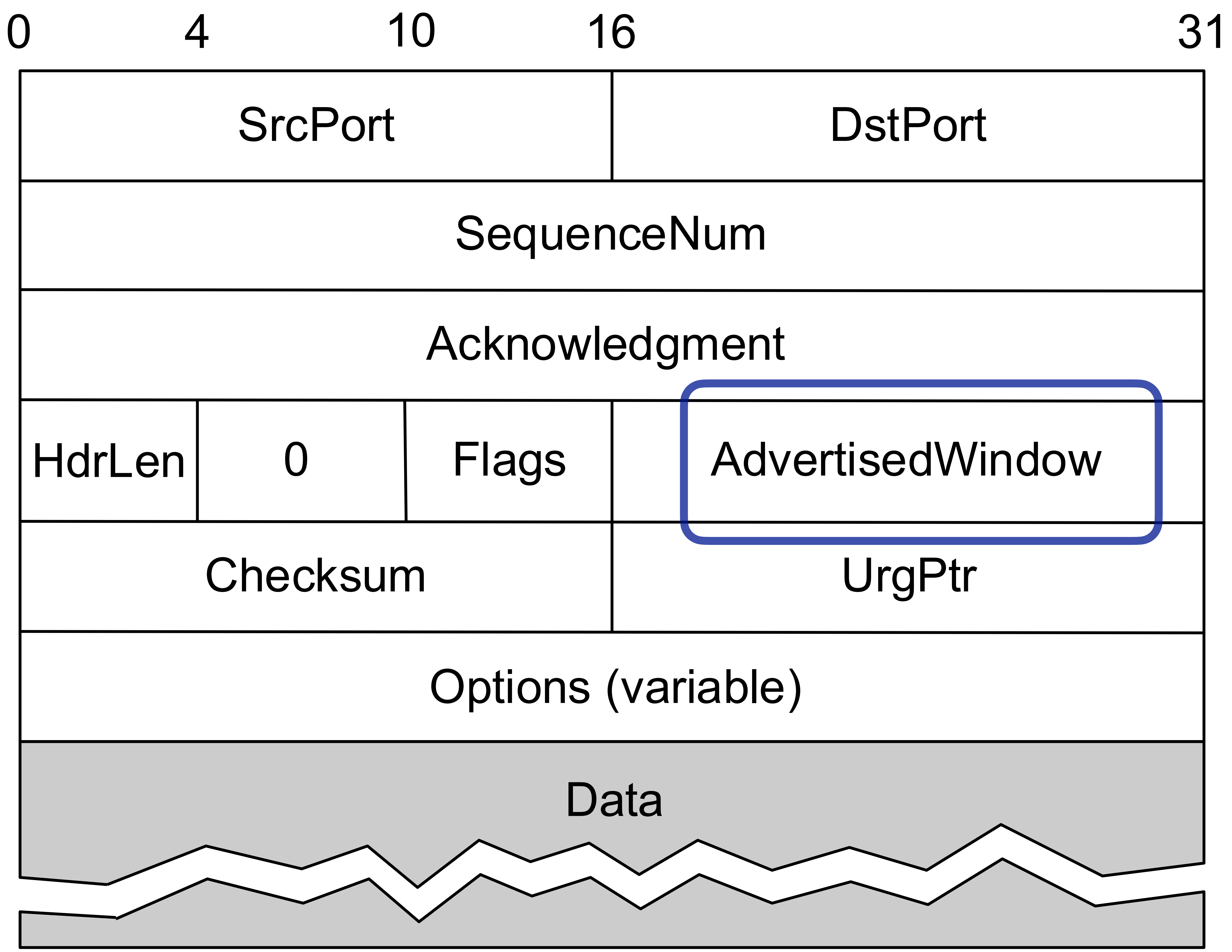
# Solution: the receiver advertises this window

LastByteRcvd - LastByteRead <= MaxRcvBuffer

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



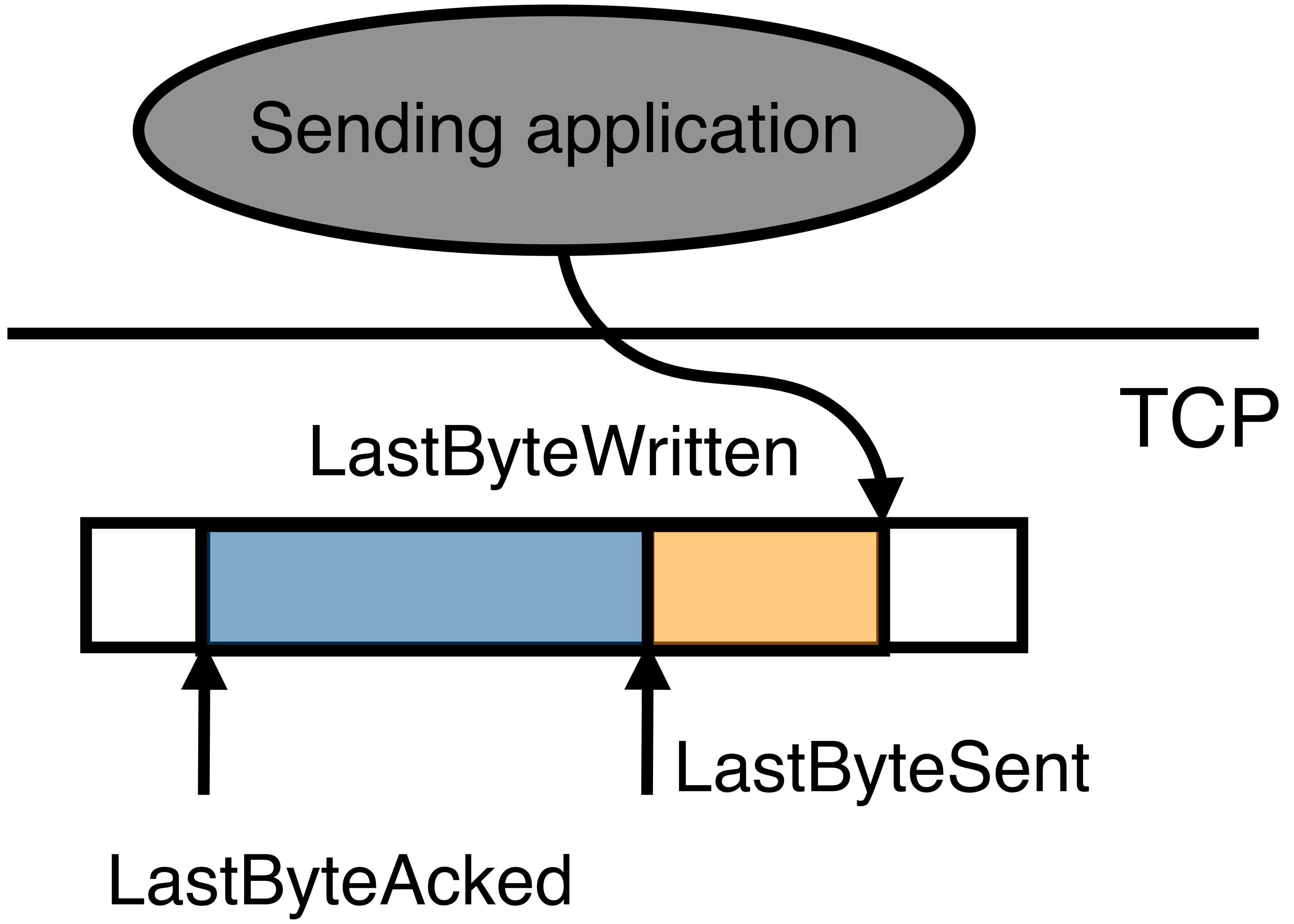
# Flow Control: Sender Side



# Flow Control: The sender controls the rate

## Sending side @TCP

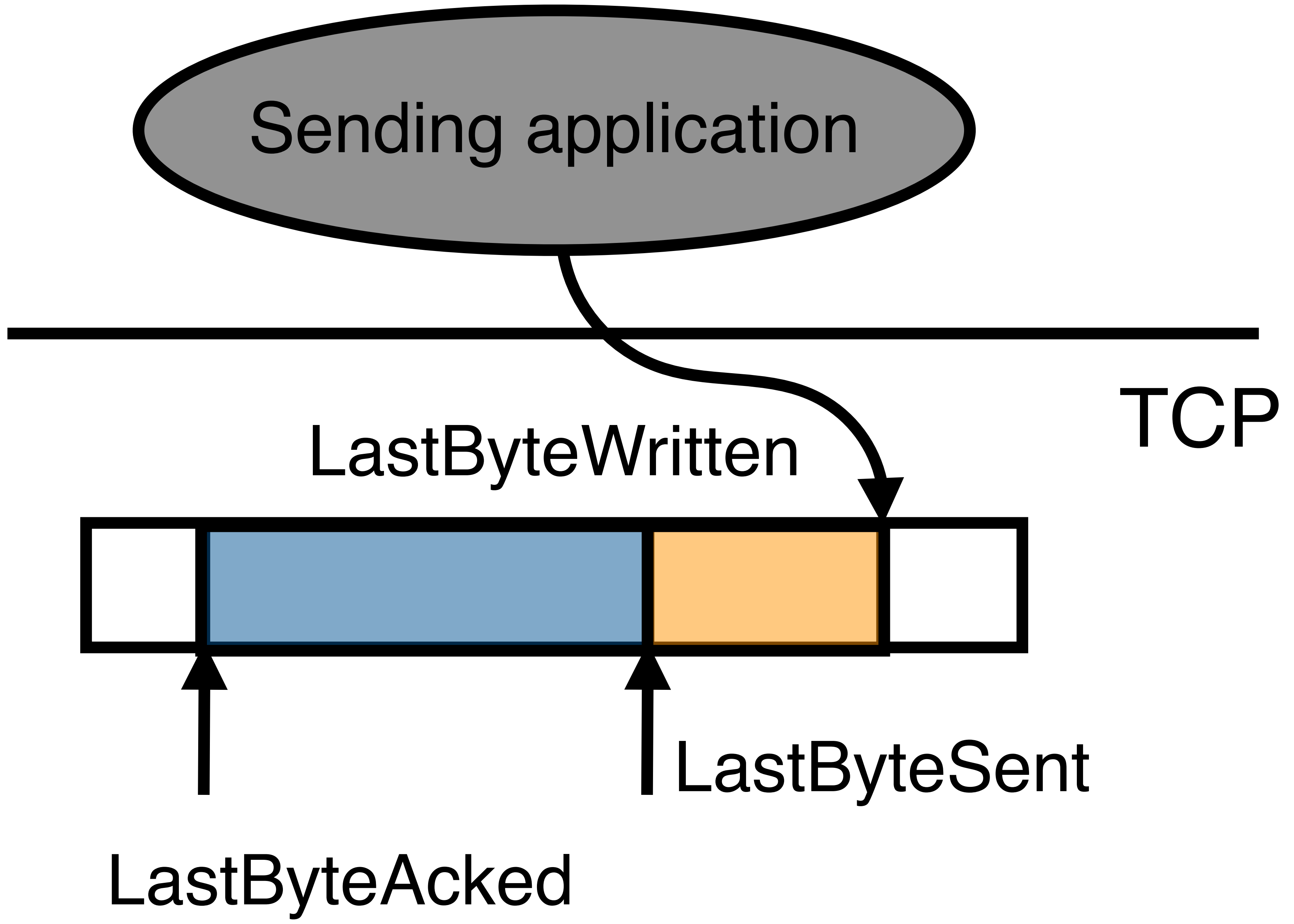
- **LastByteSent - LastByteAcked <= AdvertisedWindow**
- **EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)**



# Flow Control: The sender controls the rate

## Sending side @Application

- LastByteWritten - LastByteAcked <= MaxSendBuffer
- Block sender if (LastByteWritten - LastByteAcked) + y > MaxSendBuffer



# Flow Control Discussion

## The receiver

- Always send ACK in response to arriving data segment

## The sender

- Persist sending one byte segment when **AdvertiseWindow** `= 0`

# How TCP solves the first issue?

## #1: Arbitrary communication ✓

- Senders and receivers can talk to each other in any ways

## #2: No reliability guarantee ✓

- Packets can be lost/duplicated/reordered during transmission
- Checksum is not enough

## #3: No resource management

- Each communication channel works as an exclusive network resource owner
- No adaptiveness support for the physical networks and applications

## Terminology

1. Host
2. NIC
3. Multi-port I/O bridge
4. Protocol
5. RTT
6. Packet
7. Header
8. Payload
9. BDP
10. Baud rate
11. Frame/Framing
12. Parity bit
13. Checksum
14. Ethernet
15. MAC
16. (L2) Switch
17. Broadcast
18. Acknowledgement
19. Timeout
20. Datagram
21. TTL
22. MTU
23. Best effort
24. (L3) Router
25. Subnet mask
26. CIDR
27. Converge
28. Count-to-infinity
29. Line card
30. Network processor
31. Gateway
32. Private network
33. IPv6
34. Multicast
35. IGMP
36. SDN
37. (Transport) port
38. Pseudo header
39. SYN/ACK
40. Incarnation
41. Flow
42. SYN flood
43. TCP Segment
44. Window
45. Advertised Window

## Principle

1. Layering
2. Minimal States
3. Hierarchy

# Technique

1. NRZ Encoding
2. NRZI Encoding
3. Manchester Encoding
4. 4B/5B Encoding
5. Byte Stuffing
6. Byte Counting
7. Bit Stuffing
8. 2-D Parity
9. CRC
10. MAC Learning
11. Store-and-Forward
12. Cut-through
13. Spanning Tree
14. CSMA/CD
15. Stop-and-Wait
16. Sliding Window
17. Fragmentation and Reassembly
18. Path MTU discovery
19. DHCP
20. Subnetting
21. Supernetting
22. Longest prefix match
23. Distance vector routing (RIP)
24. Link state routing (OSPF)
25. Boarder gateway protocol (BGP)
26. Network address translation (NAT)
27. User Datagram Protocol (UDP)
28. Transmission Control Protocol (TCP)
29. Three-way Handshake
30. TCP state transition
31. EWMA
32. Sliding window
33. Flow control

# Summary

## Today's takeaways

**#1:** TCP employs the sliding window technique to achieve reliable data delivery

**#2:** TCP flow control ensures the sender never over-runs the receiver

## Next lecture

- TCP congestion control (I)