Introduction to Computer Networks

TCP Reliability Support (II)

https://pages.cs.wisc.edu/~mgliu/CS640/S26/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

- Last
  - TCP Reliability Support (I)
- Today
  - TCP Reliability Support (II)
- Announcements
  - Lab3 due on 03/27/2026

Recap

- Key Questions:
  - What is the goal of the TCP reliability mechanism?

- Terminology
  - Acknowledgement and Timeout
  - EWMA
  - Sequence Number
  - Retransmission

What is the goal of TCP reliability mechanisms?

Byte stream @sender = Byte stream @receiver

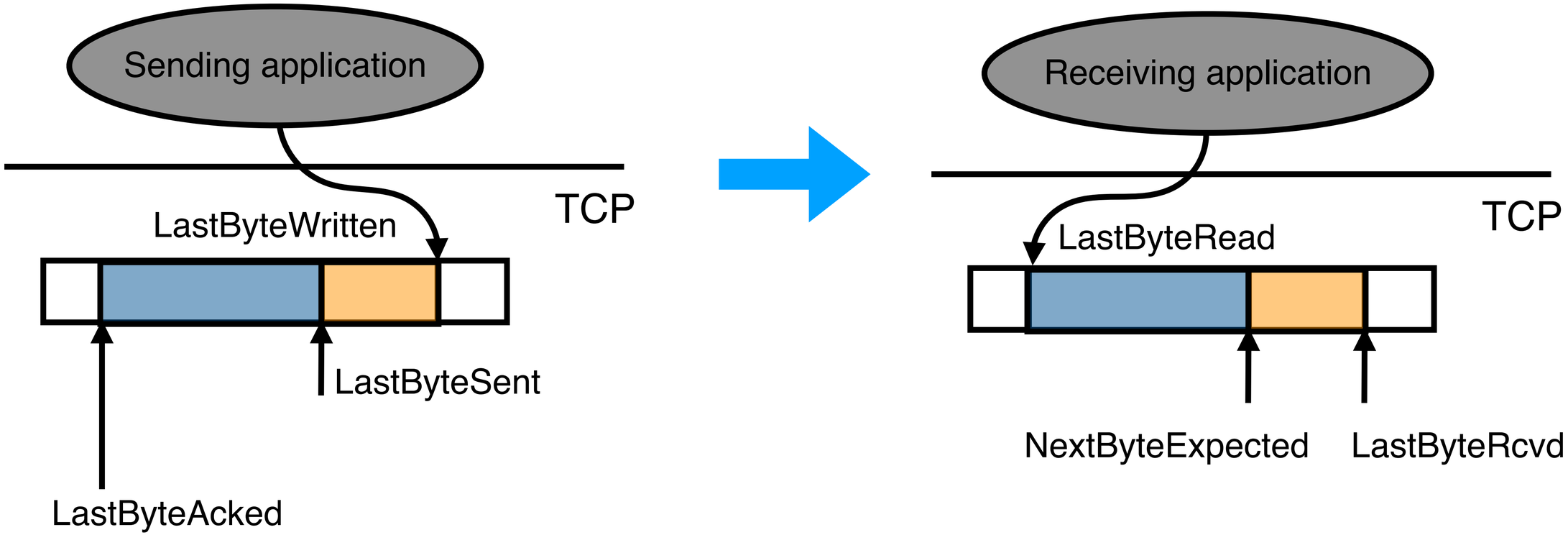
- #1: TCP segments are delivered with no loss/duplication
- #2: TCP segments are delivered in order
- #3: The sender is not over-running the receiver capability

Reliability Mechanism Summary

	Detection	**Fix**
Issue #1: Segement Loss	#1: Sender-side Acknowledgment #2: Sender-side Timeout #3: Receiver-side Sequence number	#1: Retransmit
Issue #2: Duplicated Segment	#1: Sequence Number + Window	#1: Drop
Issue #3: Out-of-Order Segment	#1: Sequence Number + Window	#1: Drop and wait for retransmission #2: Keep it and wait for the missing ones
Issue #4: Receiver Overwhelming	#1: Receiver-side buffer full	#1: Ask the sender to slow down

Combine Everything Together — TCP Sliding Window

- Continuously coordinate sender and receiver during transmission



TCP Sliding Window—Sender Logics

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

- Invariants:
  - **LastByteSent** <=**LastByteWritten**
  - **LastByteAcked** <= **LastByteSent**

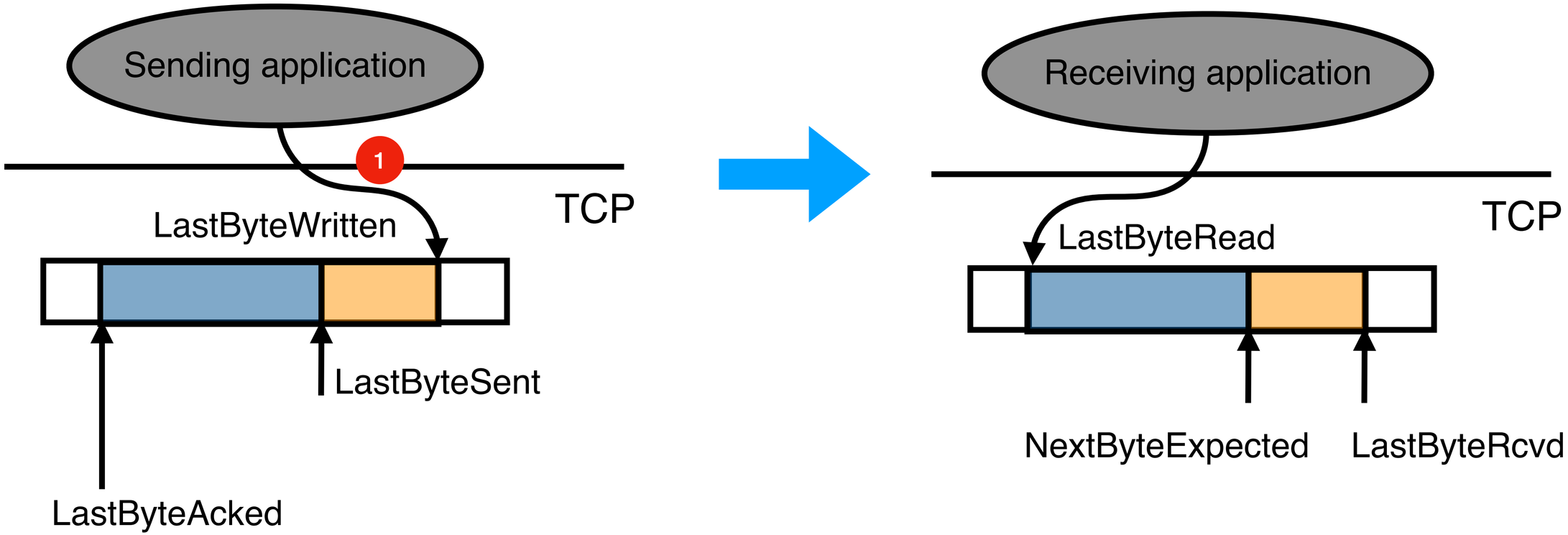
- Buffered bytes:
  - **lLastByteWritten - LastByteAckedl** <= **MaxSendBuffer**

TCP Sliding Window—Receiver Logics

- Three variables manipulations:
  - Advance **LastByteRead** when an app reads
  - Advance **LastByteRcvd** when the segment is received
  - Advance **NextByteExpected** when the next expected segment is received
- Invariants:
  - **LastByteRead** < **NextByteExpected**
  - **NextByteExpected** <= **LastByteRcvd** + 1
- Buffered bytes:
  - **lLastByteRcvd - LastByteReadl <= MaxRcvBuffer**

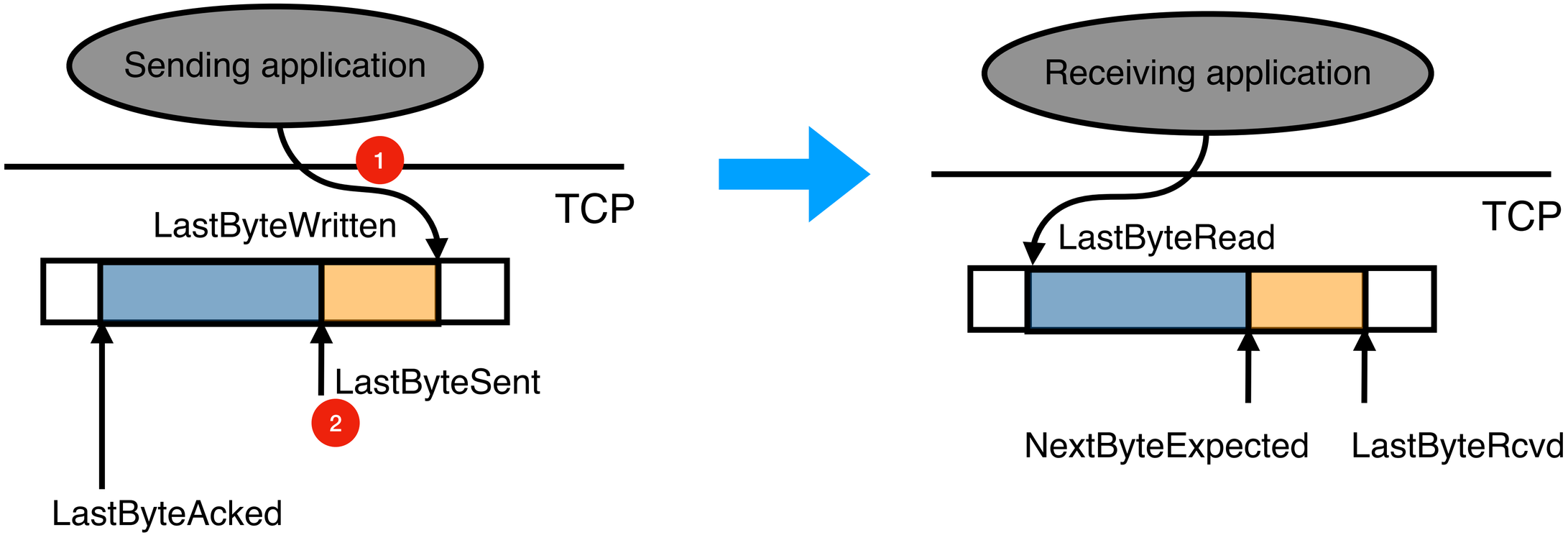
How it works

- Step #1: The sending application writes data to the send buffer
  - **LastByteWritten += sizeof (written data)**



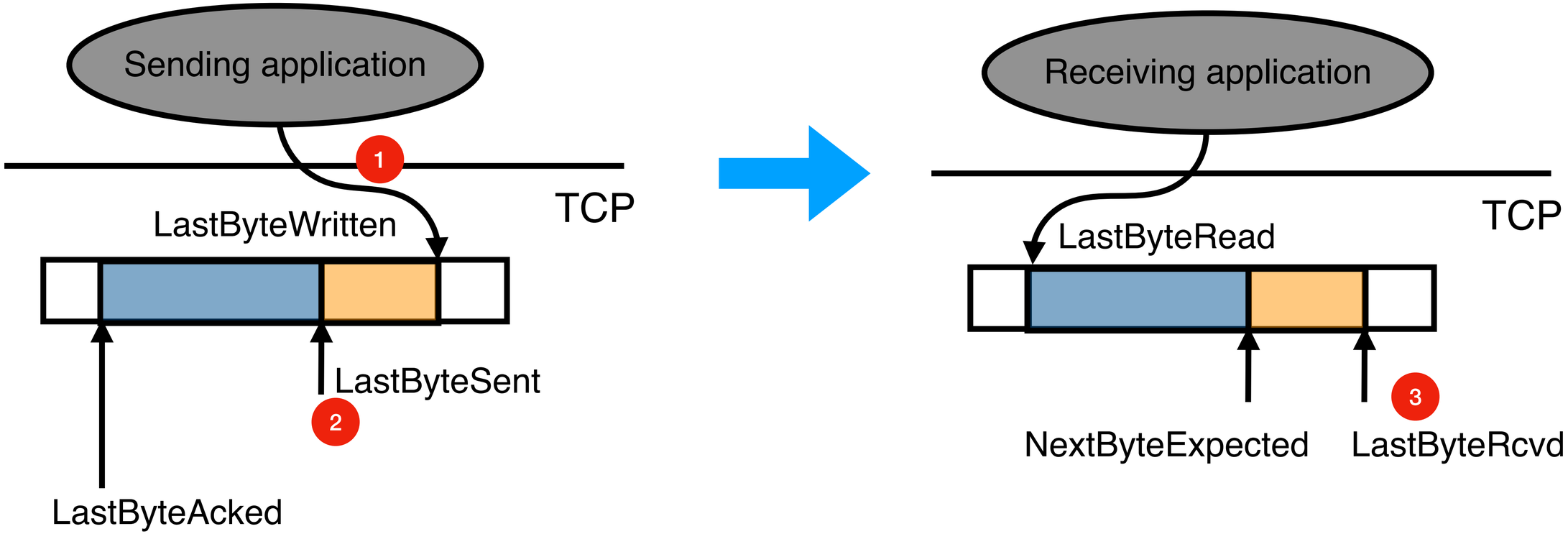
How it works

- Step #2: The buffered data is sent out by OS/NIC
  - **LastByteSent += sizeof (sent data)**



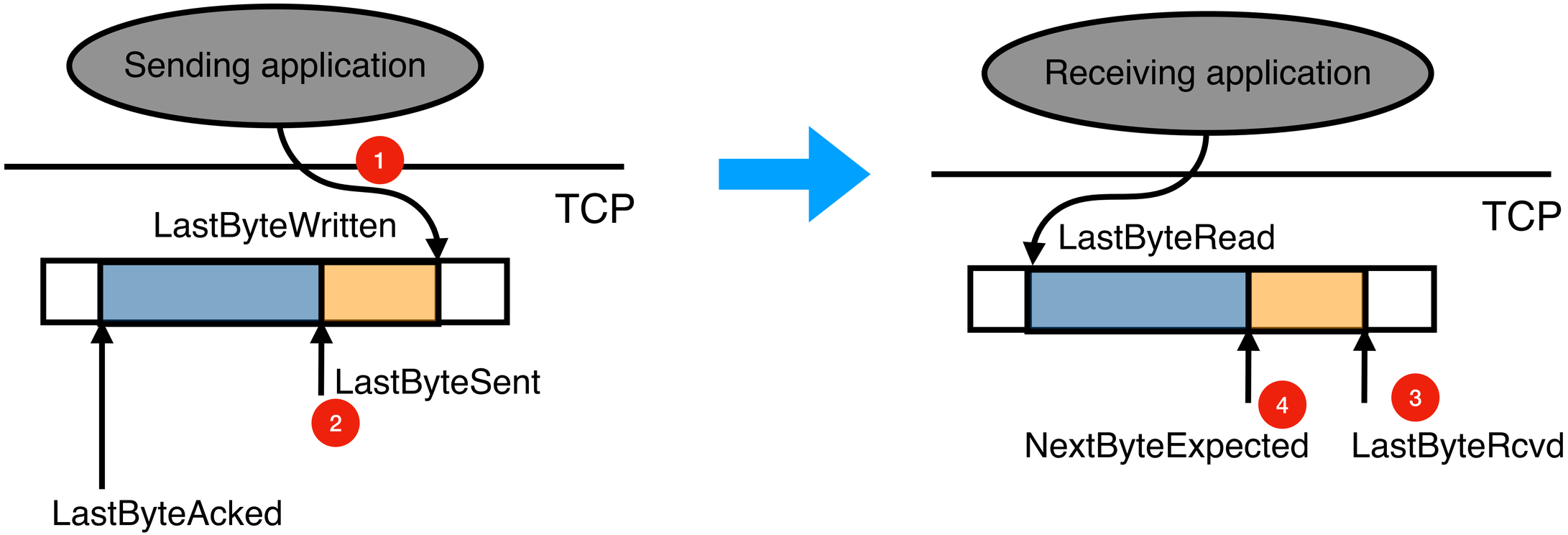
How it works

- Step #3: The data is by the received host and put into the buffer
  - **LastByteRcvd += sizeof (received data)**
  - Advance **NextByteExpected** depending on if there is a hole



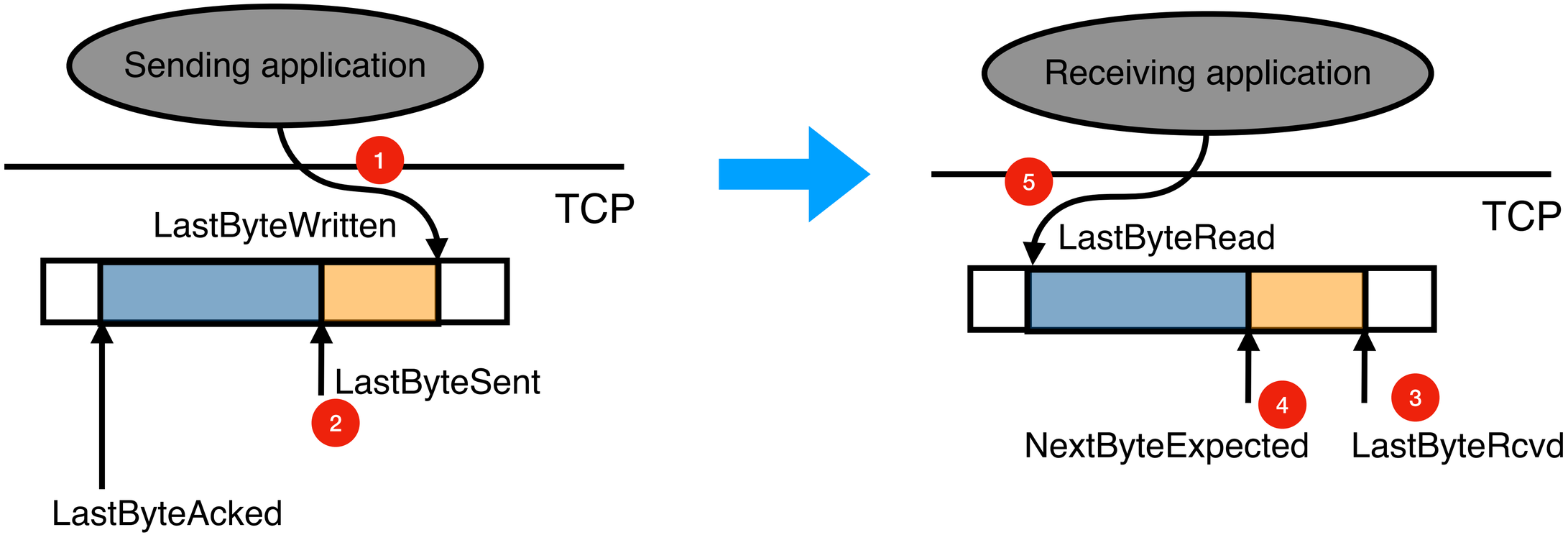
How it works

- Step #4: Received data is sequenced in the buffer
  - Advance **NextByteExpected** when necessary



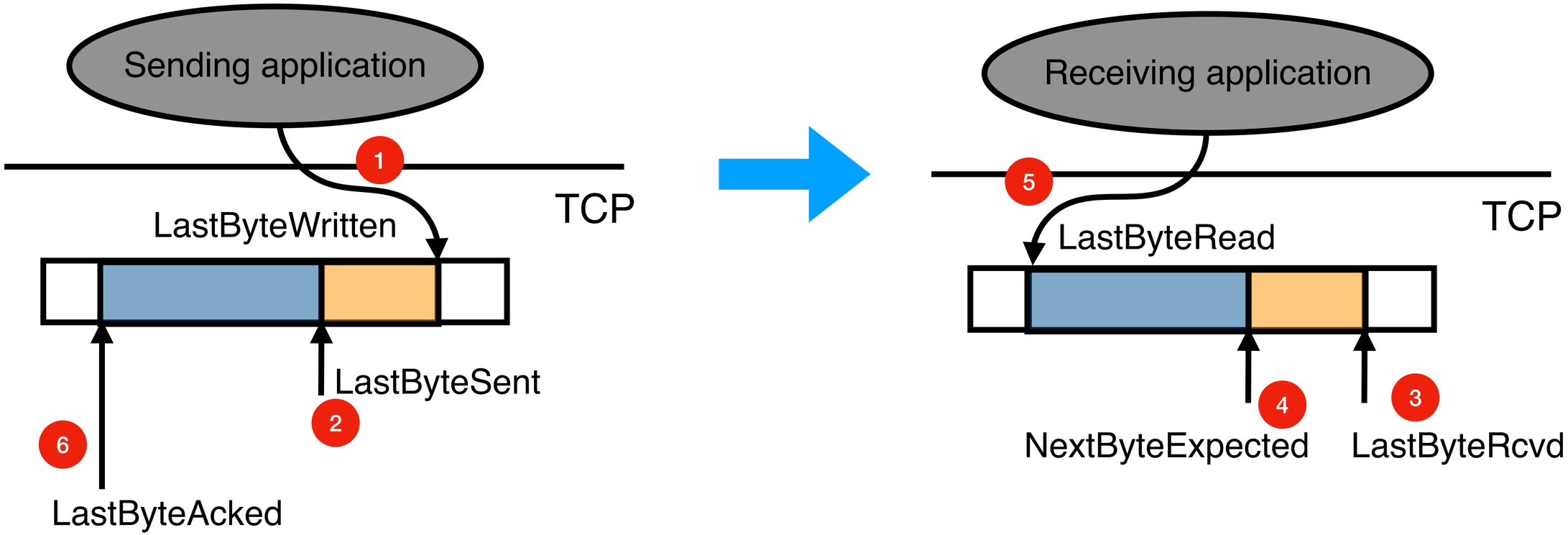
How it works

- Step #5: The receiving application reads data from the buffer
  - **LastByteRead += sizeof (read data)**



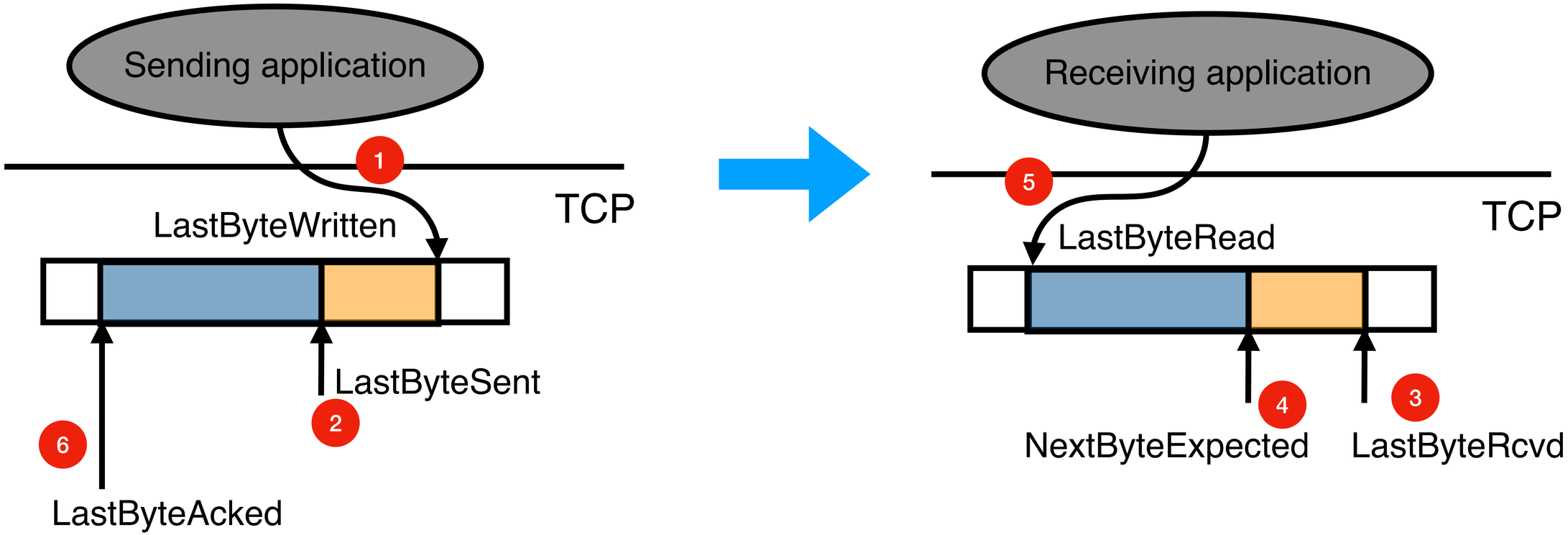
How it works

- Step #6: The receiving application sends ACKs to the sender
  - Advance **LastByteAcked** when necessary



Why Sender Invariants

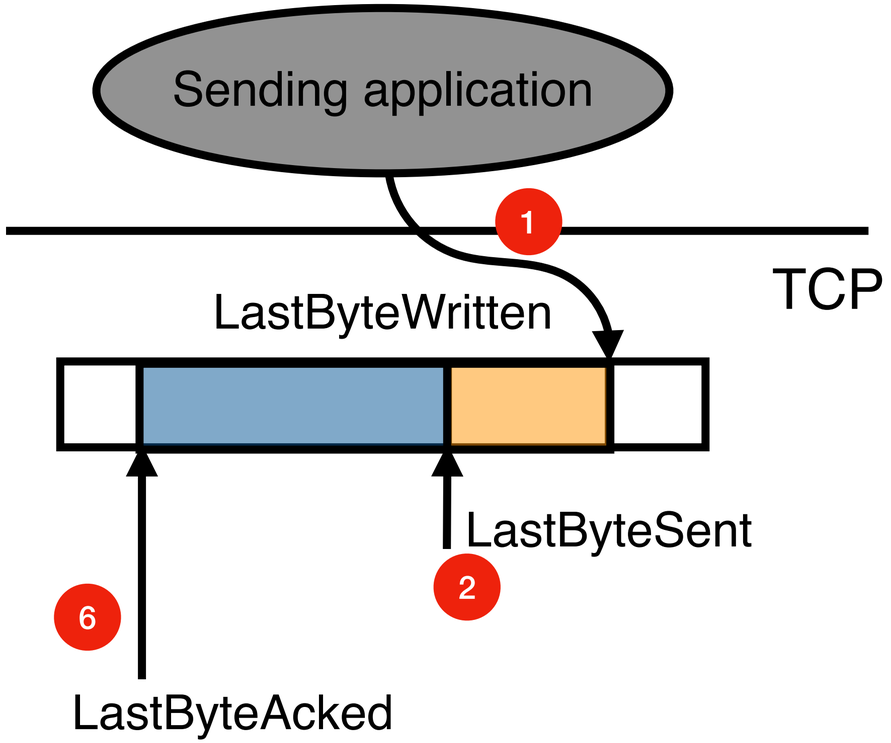
- **LastByteSent <=LastByteWritten**
- **LastByteAcked <= LastByteSent**

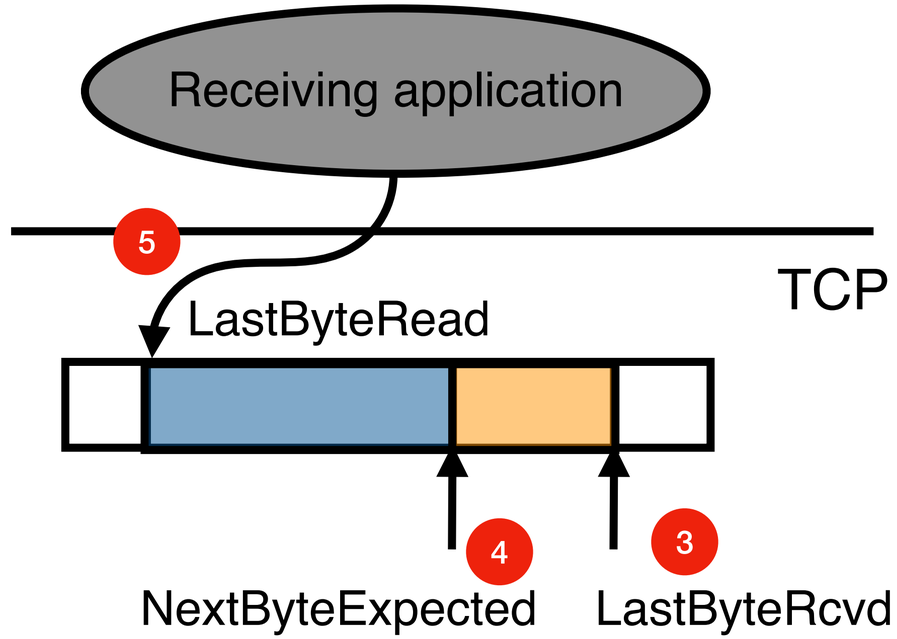


Why Receiver Invariants

- **LastByteSent <=LastByteWritten**
- **LastByteAcked <= LastByteSent**

- **LastByteRead < NextByteExpected**
- **NextByteExpected <= LastByteRcvd + 1**

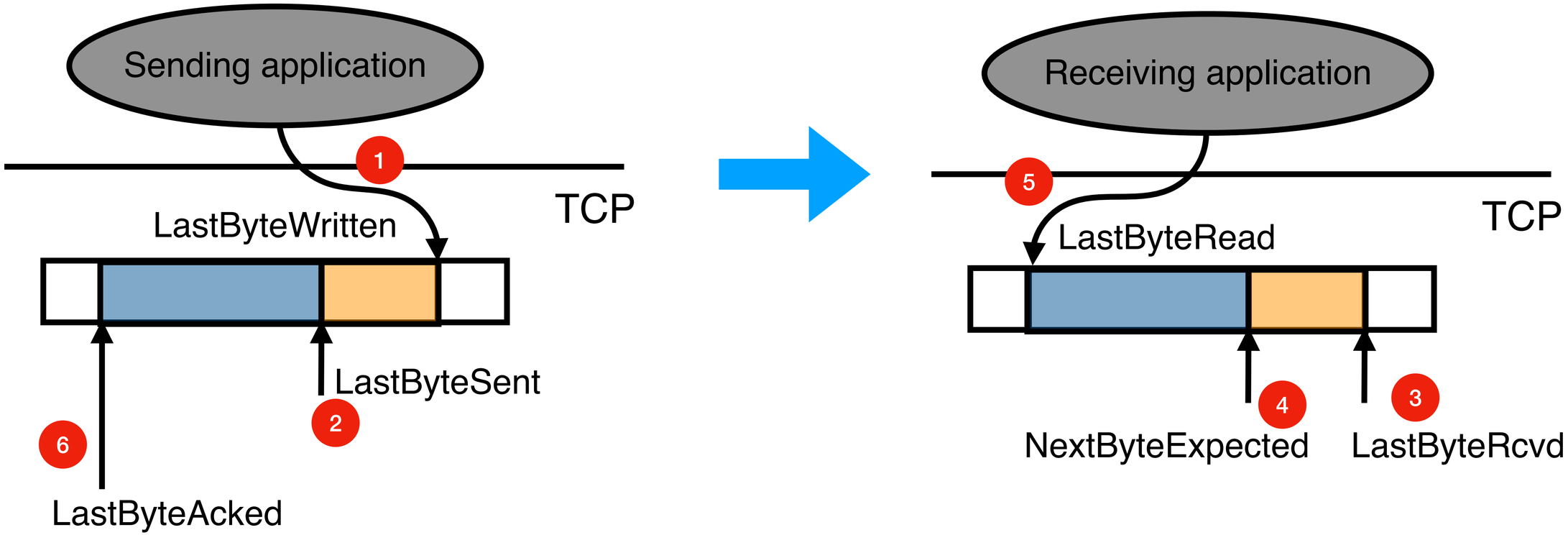




Understanding the Sender Buffer

- **LastByteSent <=LastByteWritten**
- **LastByteAcked <= LastByteSent**

- **LastByteRead < NextByteExpected**
- **NextByteExpected <= LastByteRcvd + 1**

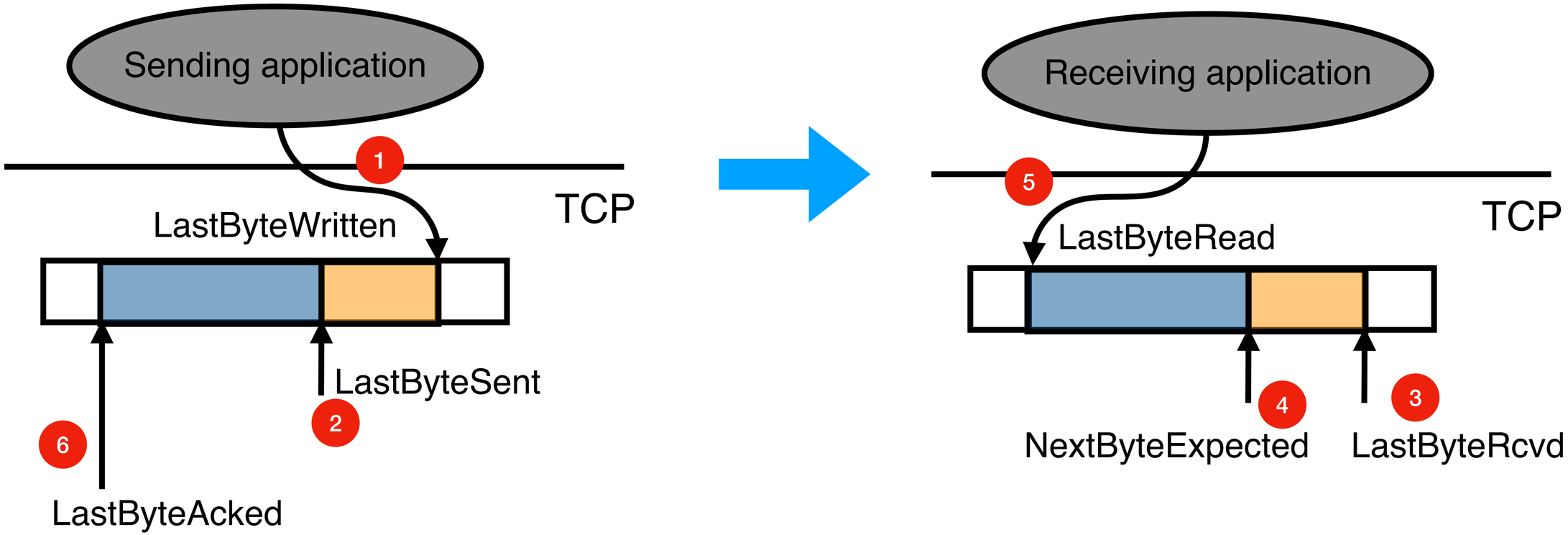


- **lLastByteWritten - LastByteAckedl <= MaxSendBuffer**

Understanding the Receiver Buffer

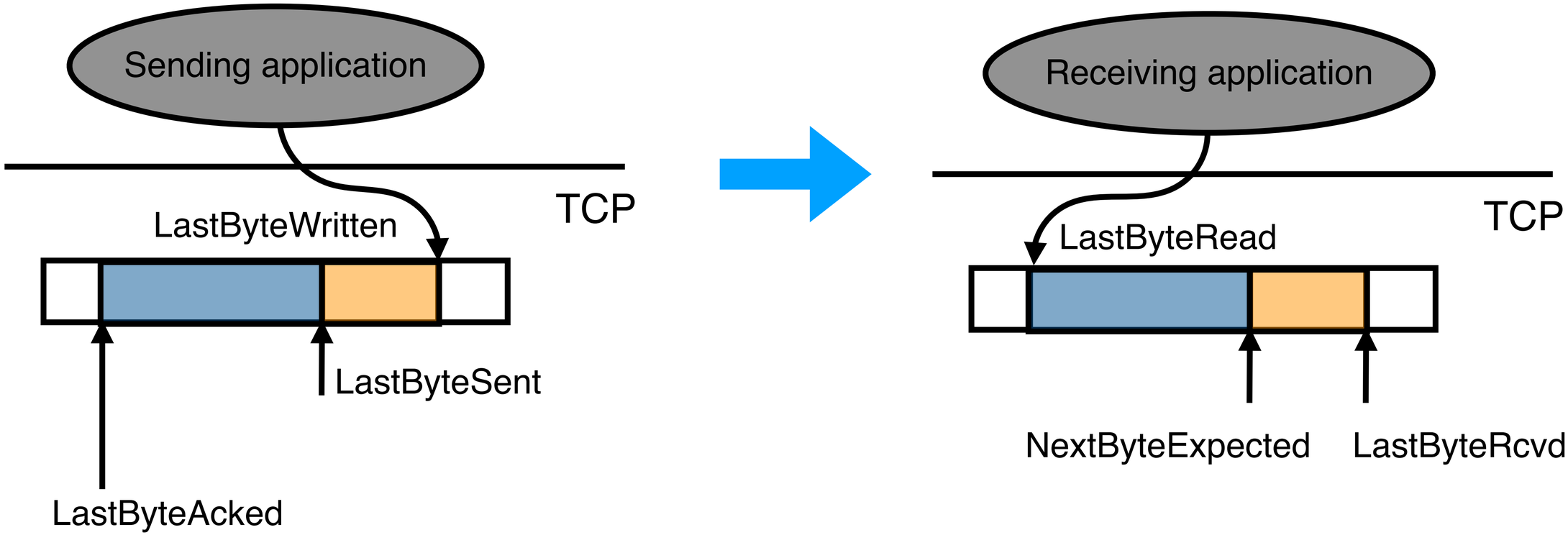
- **LastByteSent <=LastByteWritten**
- **LastByteAcked <= LastByteSent**

- **LastByteRead < NextByteExpected**
- **NextByteExpected <= LastByteRcvd + 1**



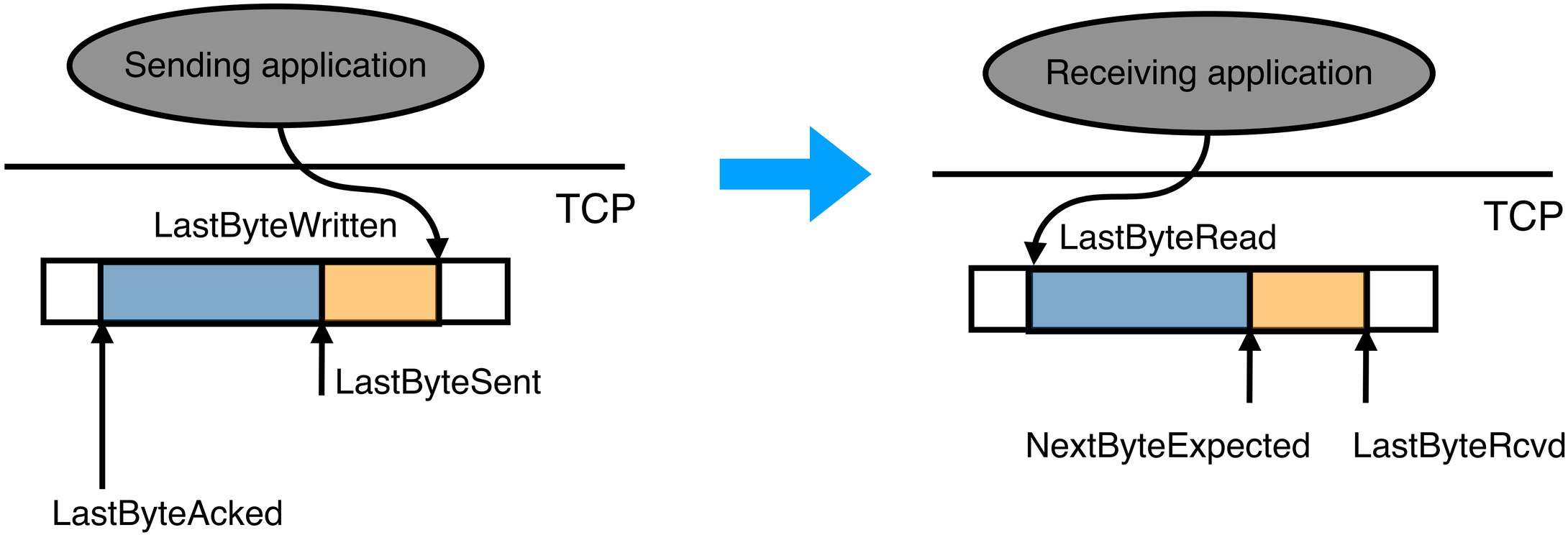
- **lLastByteRcvd - LastByteReadl <= MaxRcvBuffer**

Tackling Issue #1 (Missing Segment)

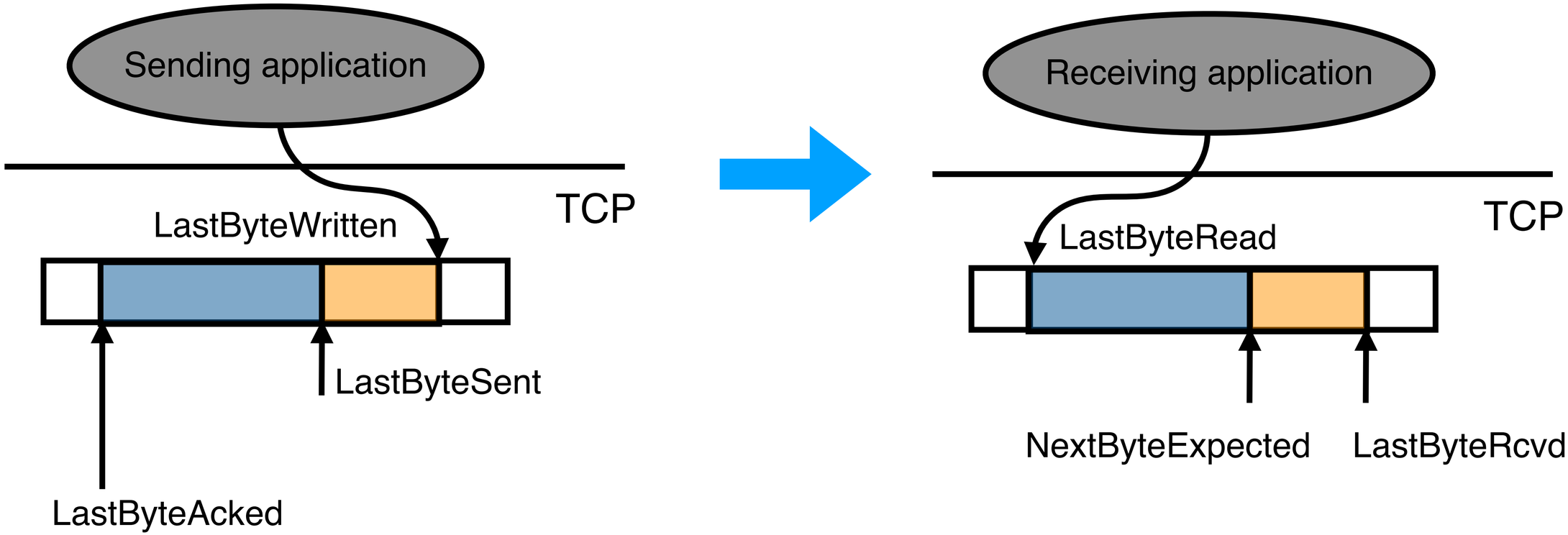


Tackling Issue #1 (Missing Segment)

- Receiver-side detection: [**NextByteExpected**, **LastByteRcvd**]
- Sender-side detection: [**LastByteAcked**, **LastByteSent**]
- Fix: buffered bytes are only freed before **LastByteAcked**

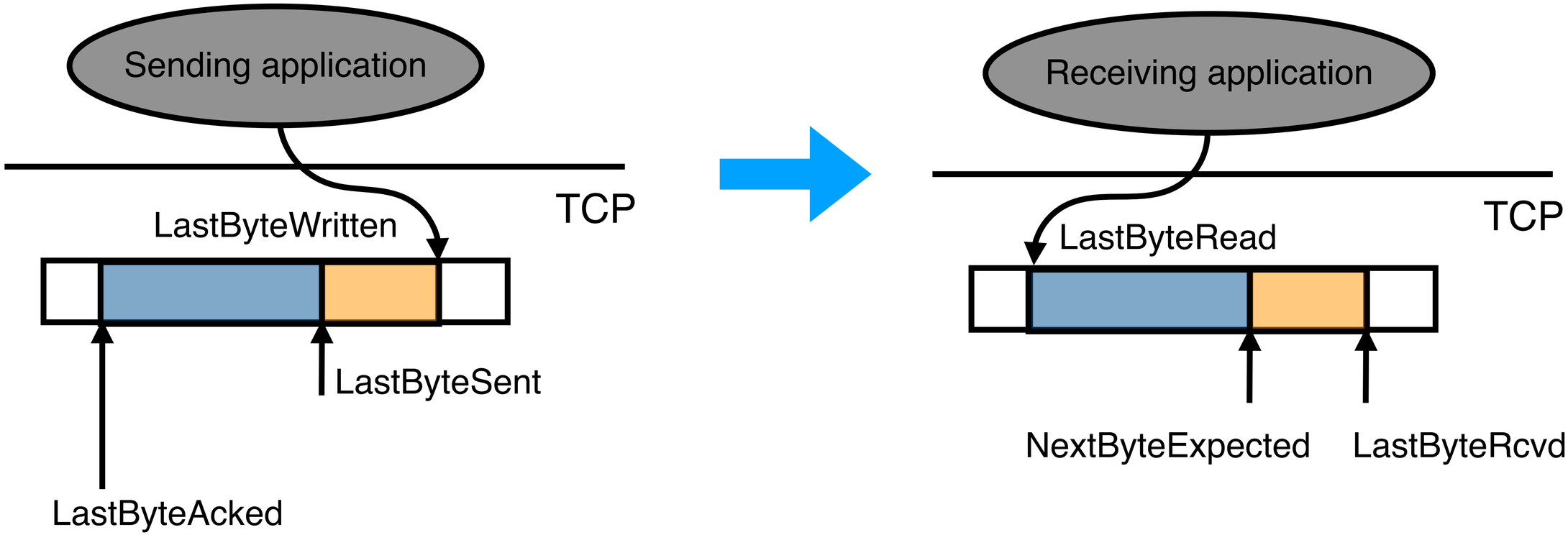


Tackling Issue #2 (Duplicated Segment)

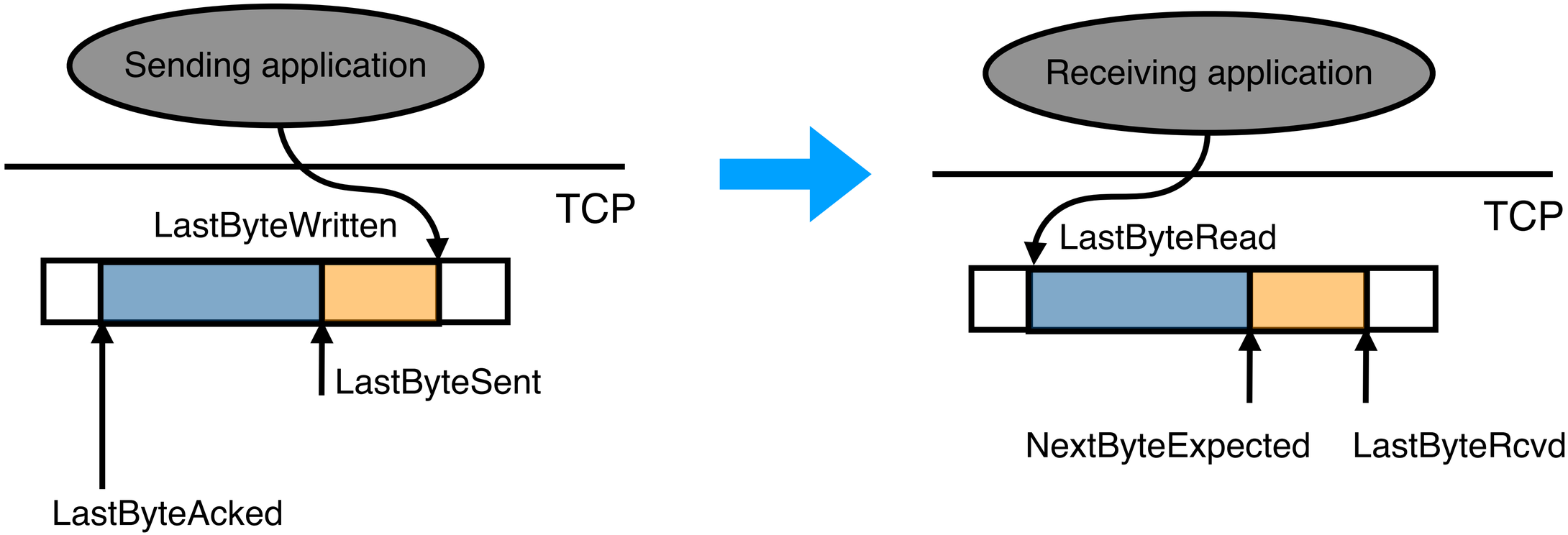


Tackling Issue #2 (Duplicated Segment)

- Detection: [**LastByteRead**, **NextByteExpected**]
- Fix: drop

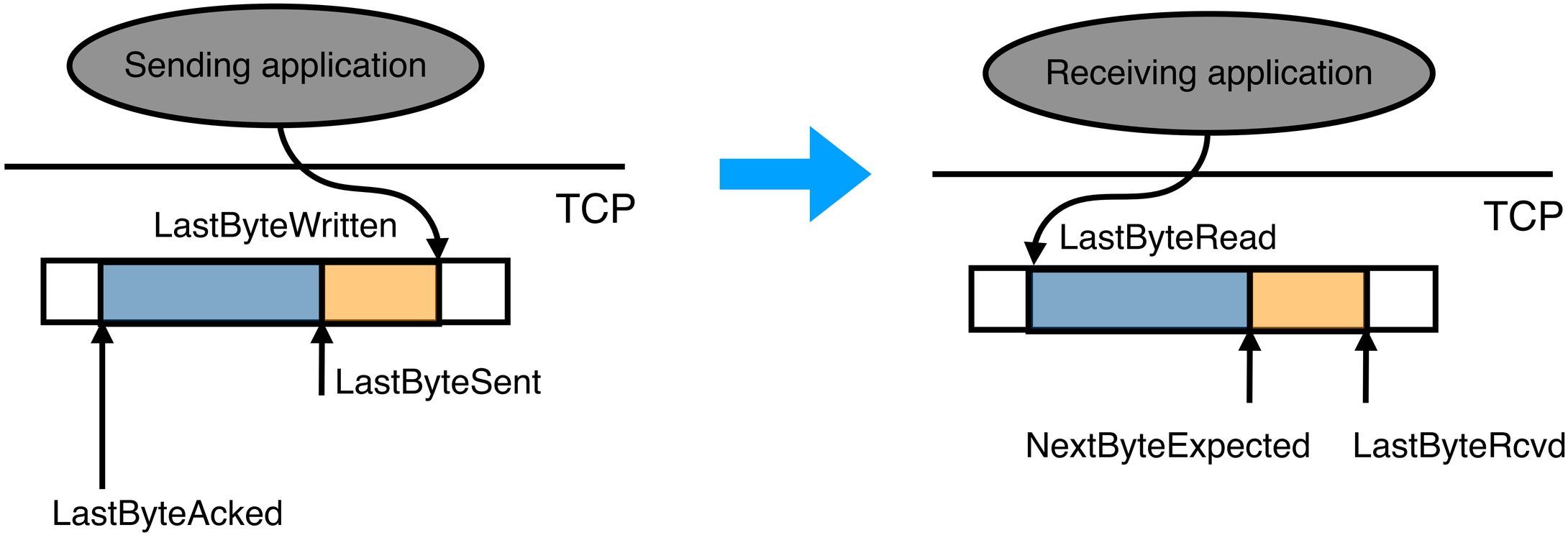


Tackling Issue #3 (Out-of-order Segment)

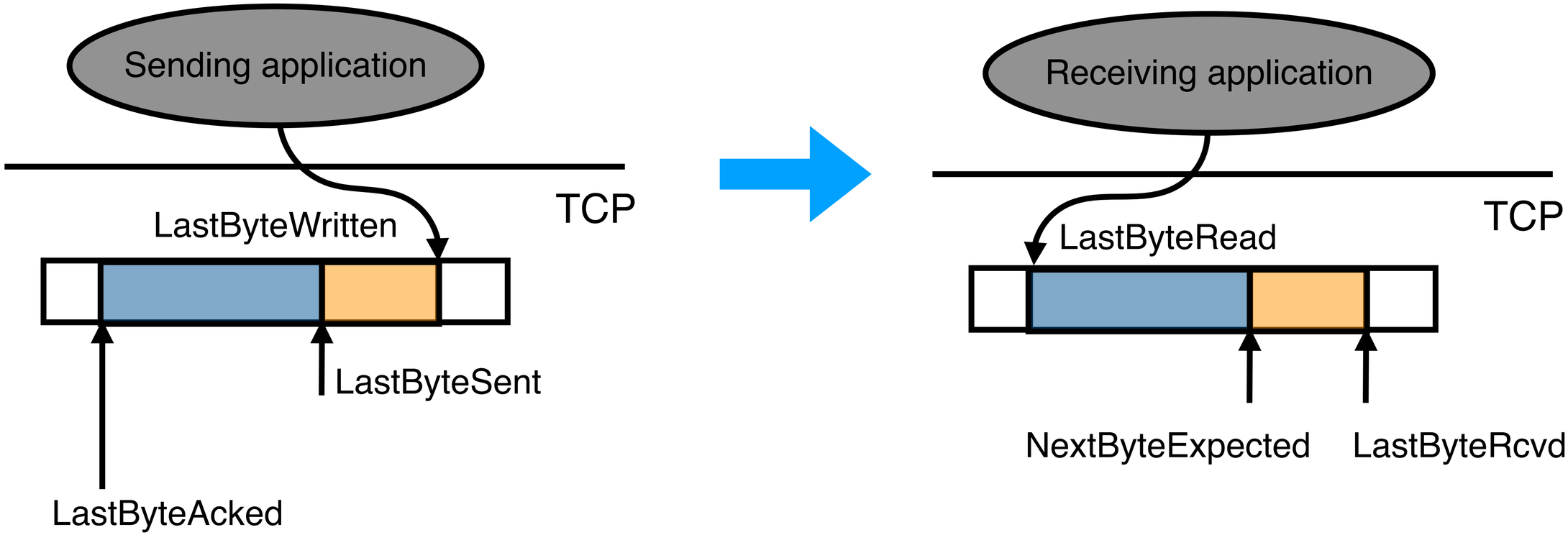


Tackling Issue #3 (Out-of-order Segment)

- Detection: [**NextByteExpected**, **LastByteRcvd**]
- Fix: take if **lLastByteRcvd - LastByteReadl <= MaxRcvBuffer**

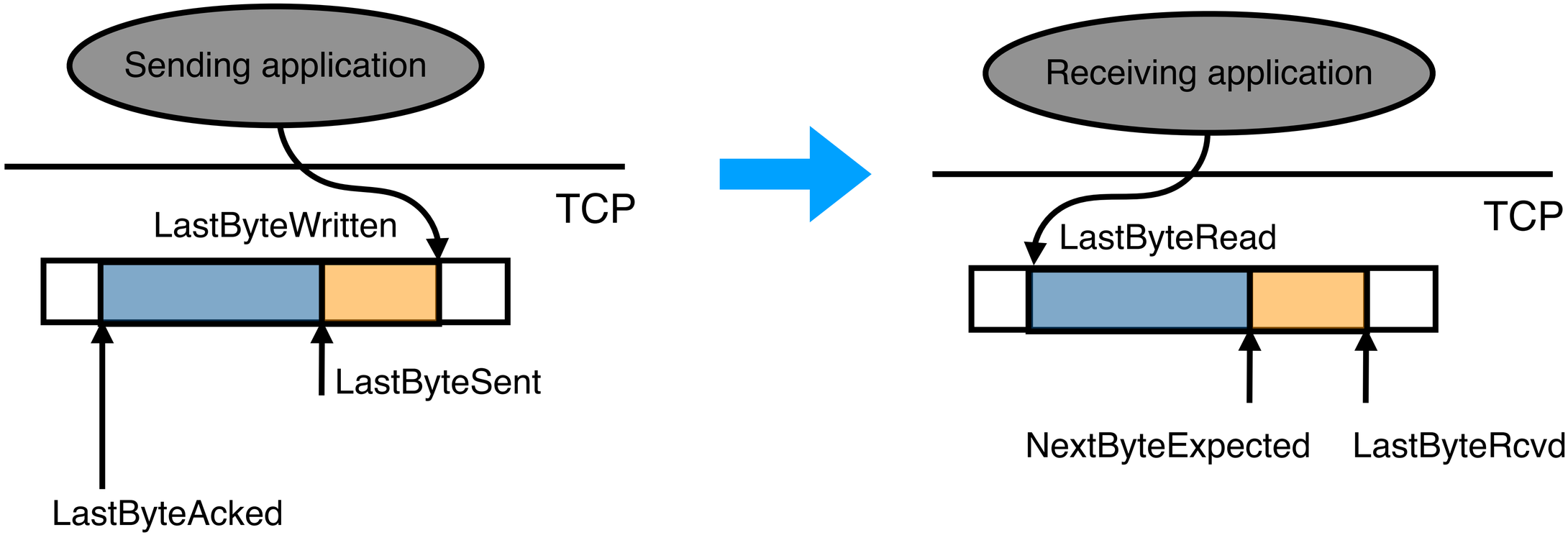


Tackling Issue #4 (Receiver Overwhelming)



Tackling Issue #4 (Receiver Overwhelming)

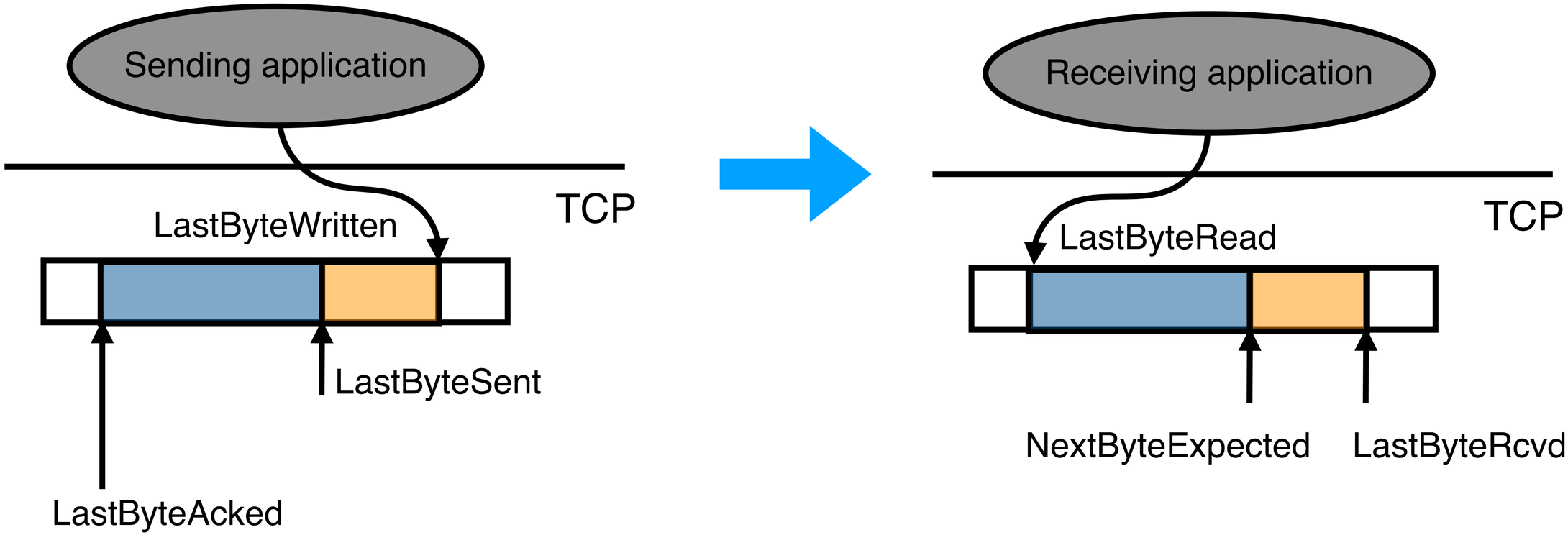
- Detection: **lLastByteRcvd - LastByteReadl <= MaxRcvBuffer**
- Fix: tell the sender the available space (**AdvertisedWindow**)



Tackling Issue #4 (Receiver Overwhelming)

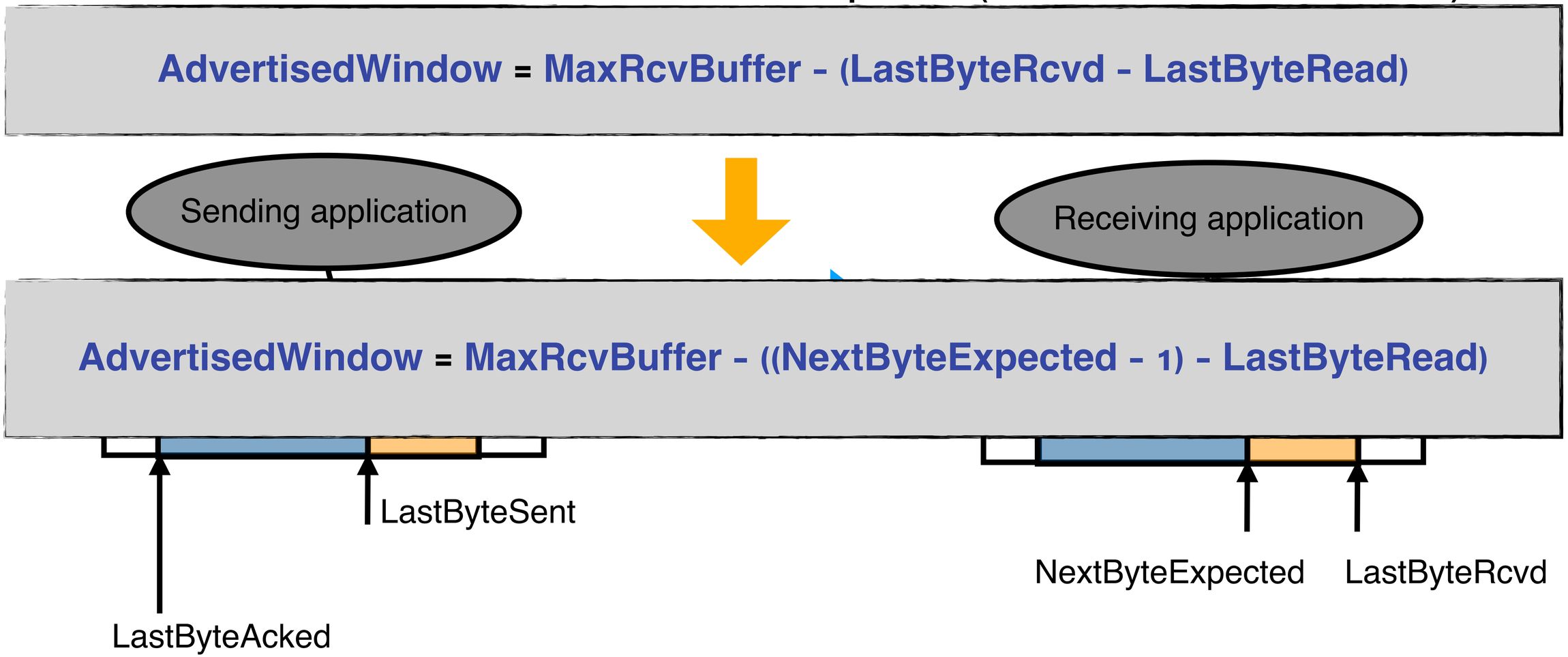
- Detection: **lLastByteRcvd - LastByteReadl <= MaxRcvBuffer**
- Fix: tell the sender the available space (**AdvertisedWindow**)

AdvertisedWindow = MaxRcvBuffer - (LastByteRcvd - LastByteRead)

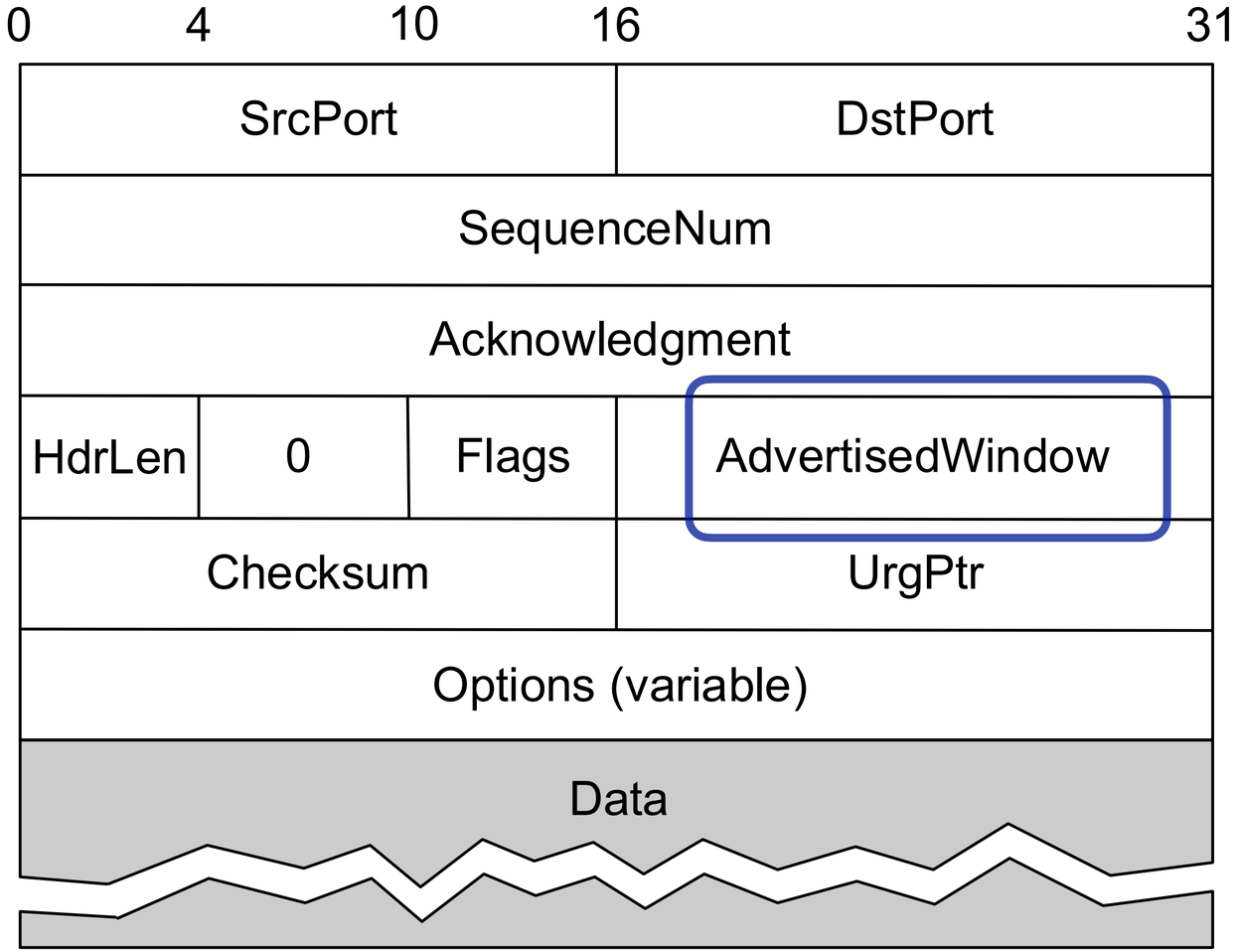


Tackling Issue #4 (Receiver Overwhelming)

- Detection: **lLastByteRcvd - LastByteReadl <= MaxRcvBuffer**
- Fix: tell the sender the available space (**AdvertisedWindow**)

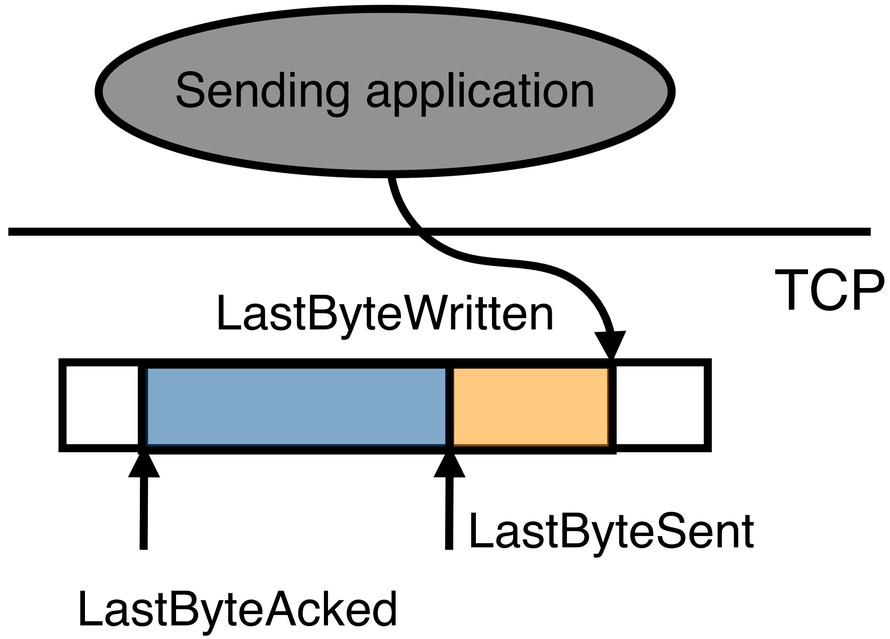


Revisiting TCP Header



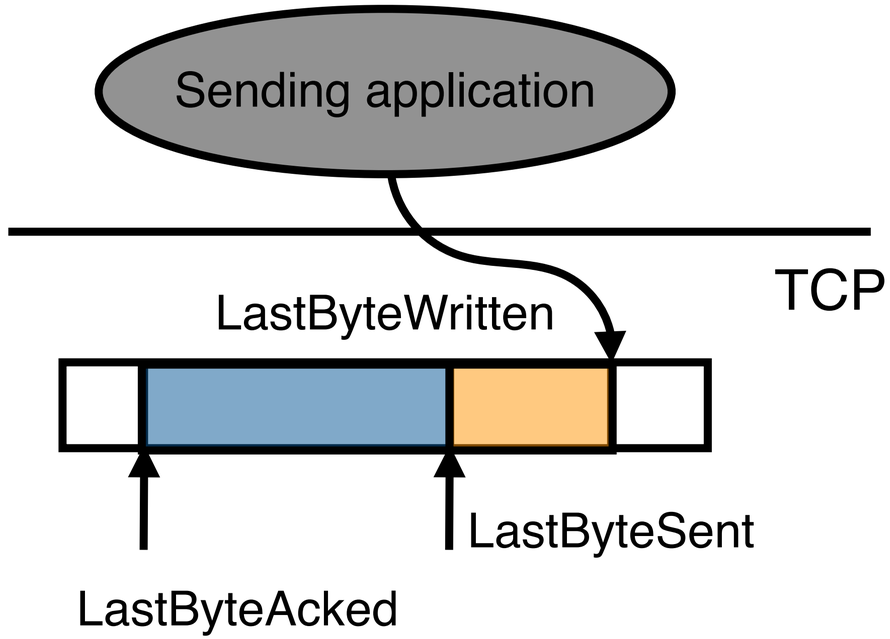
TCP Flow Control

- The sender controls the transmission rate
  - LastByteSent - LastByteAcked <= AdvertisedWindow
  - EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)



TCP Flow Control Affects Application Performance

- The application speed is throttled
  - LastByteWritten - LastByteAcked <= MaxSendBuffer
  - Block sender if (LastByteWritten - LastByteAcked) + y > MaxSendBuffer



Flow Control More

- The receiver
  - Always send ACKs in response to arriving data segments

- The sender
  - Persistent sending at least one byte when AdvertisedWindow = 0

How does TCP solve the second 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**
  - **A checksum is not enough**

- #3: No resource management
  - Each channel works as an exclusive network resource owner
  - No adaptive support for the physical networks and applications

Summary

- Today
  - TCP reliability support (II)

- Next lecture
  - TCP congestion control (I)