ECE 46300 - Fall 2024

Transport Layer III

Vishal Shrivastav

Transmission Control Protocol (TCP)

- **Reliable, In-order Delivery to Applications**
  - Application Multiplexing ✓
  - ACKs and Retransmissions
  - Flow Control
  - Congestion Control

- **Byte Stream Abstraction** ✓
  - Communicates with the applications using a byte stream and *not* packets

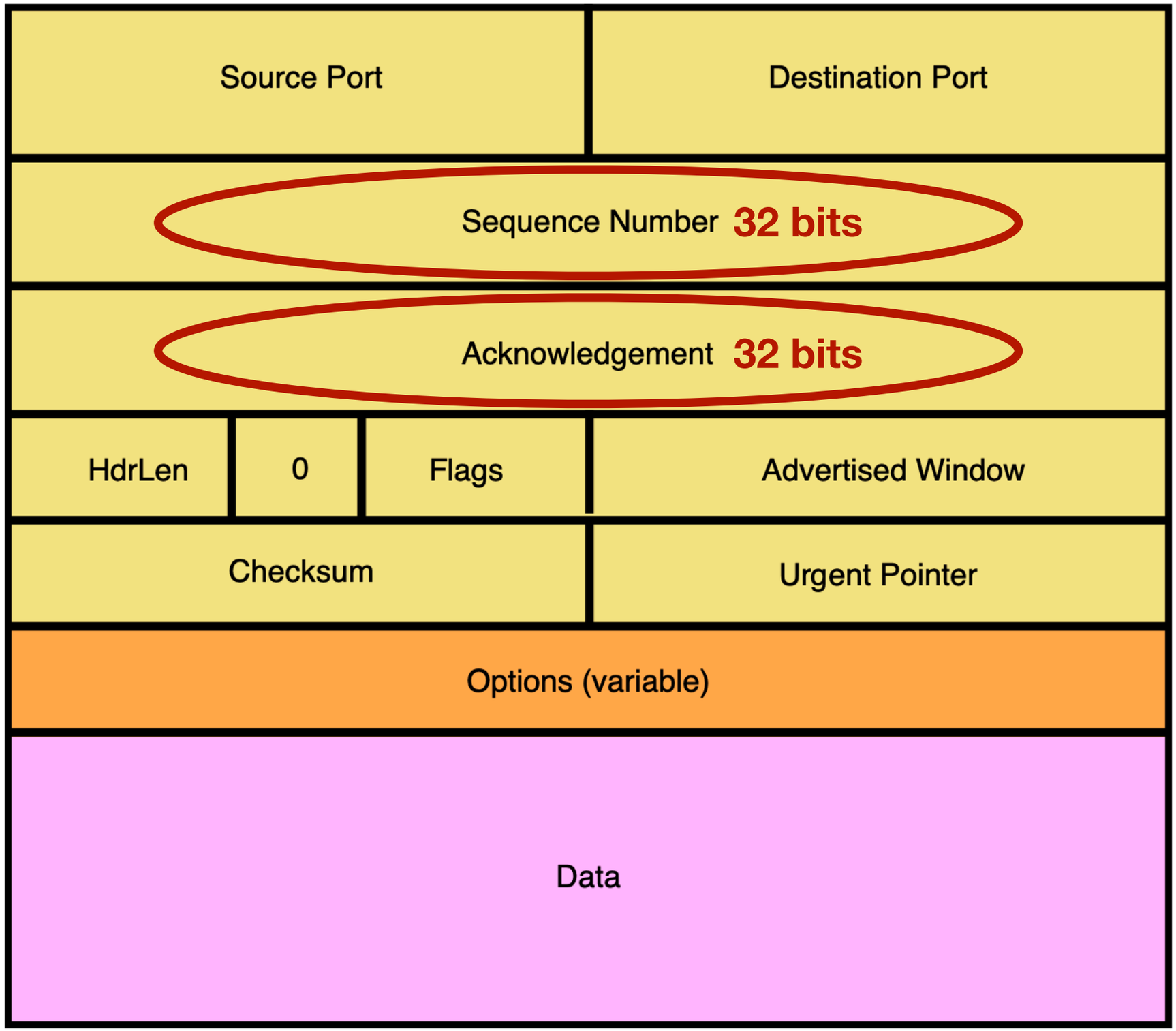
- **Connection-oriented**
  - Pairwise <sender, receiver> connection is established before sending data
  - Used to maintain connection-specific state, e.g., initial seq num, window scaling factor, window size, etc.

Transmission Control Protocol (TCP)

- **Reliable, In-order Delivery to Applications**
  - Application Multiplexing
  - ACKs and Retransmissions
  - Flow Control
  - Congestion Control
- **Byte Stream Abstraction**
  - Communicates with the applications using a byte stream and *not* packets
- **Connection-oriented**
  - Pairwise <sender, receiver> connection is established before sending data
  - Used to maintain connection-specific state, e.g., initial seq num, window scaling factor, window size, etc.

ACKs

Sequence and Acknowledgement Num



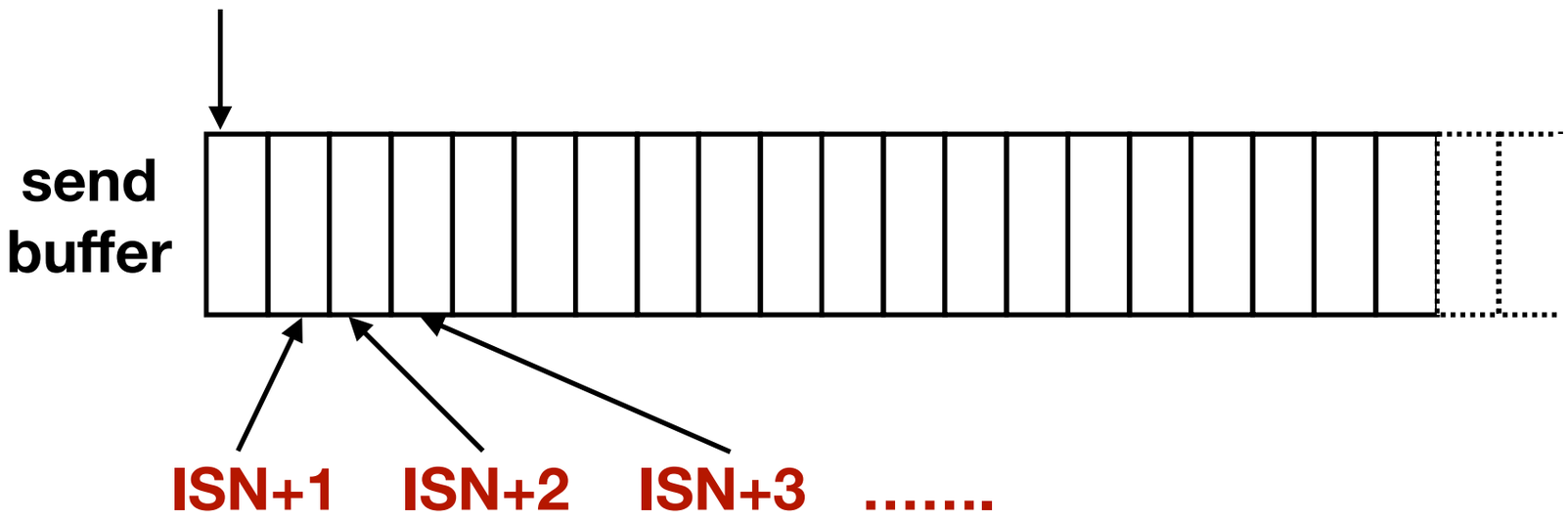
Sequence Number and ACK

- **Previously we focused on <u>packets</u>**
  - Sequence number referred to **packet number**
  - ACKs contained received sequence/packet numbers
  - Window size was expressed in terms of # of packets
- **But, TCP focuses on <u>bytes</u>**
  - Sequence number refers to **byte number** of bytes being sent
  - ACKs contain received sequence/byte numbers
  - Window size expressed in terms of # of bytes

Sequence Number and ACK

Initial Seq Num (ISN)

Each byte to be sent has a sequence number



Suppose sequence number is represented using k bits,

What will be the sequence number of $(2^k + 1)^{th}$ byte?

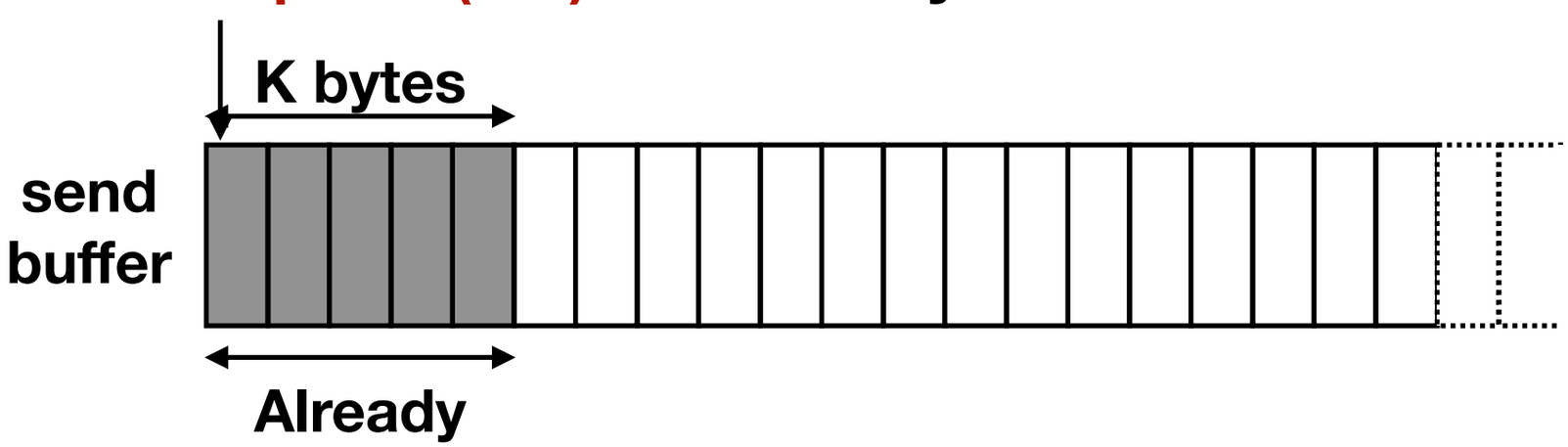
Answer: Same as the sequence number of 1st byte,

i.e., sequence numbers wrap around!

Sequence Number and ACK

Initial Seq Num (ISN)

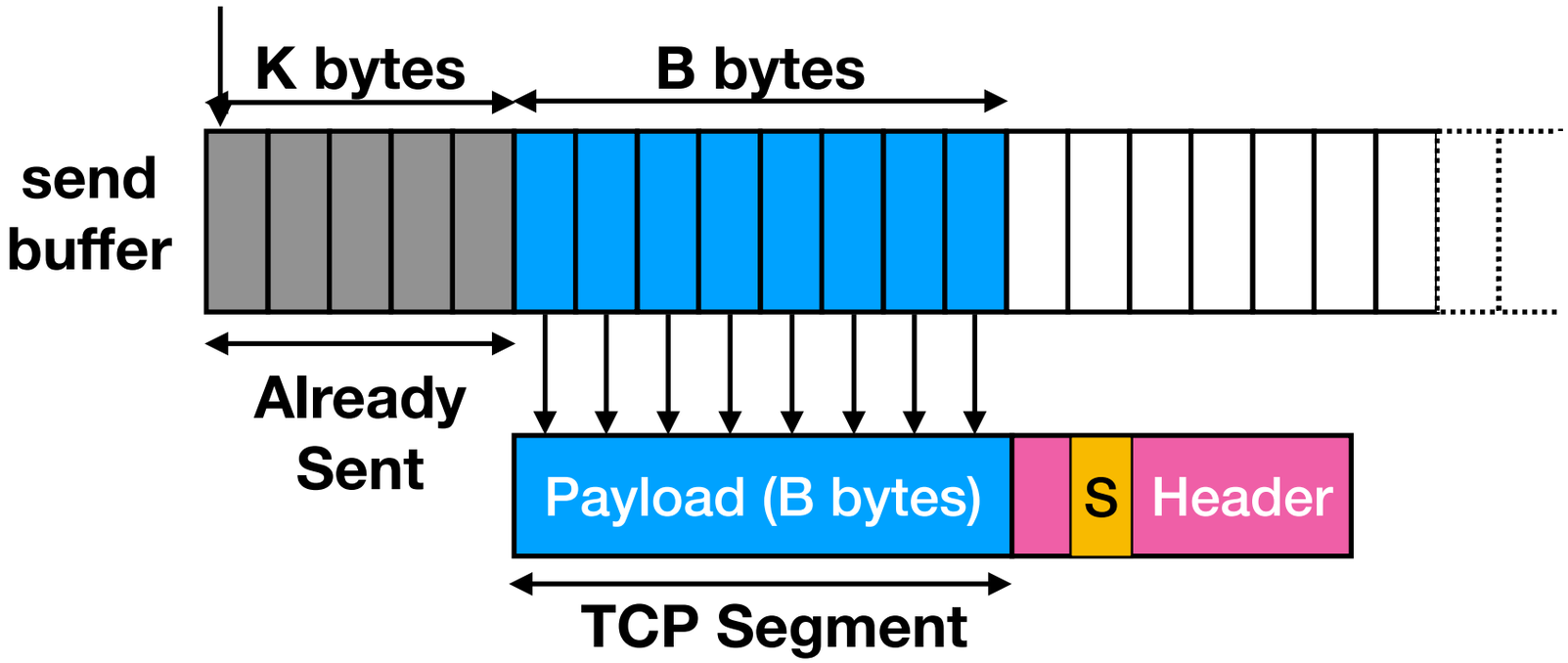
Each byte to be sent has a sequence number



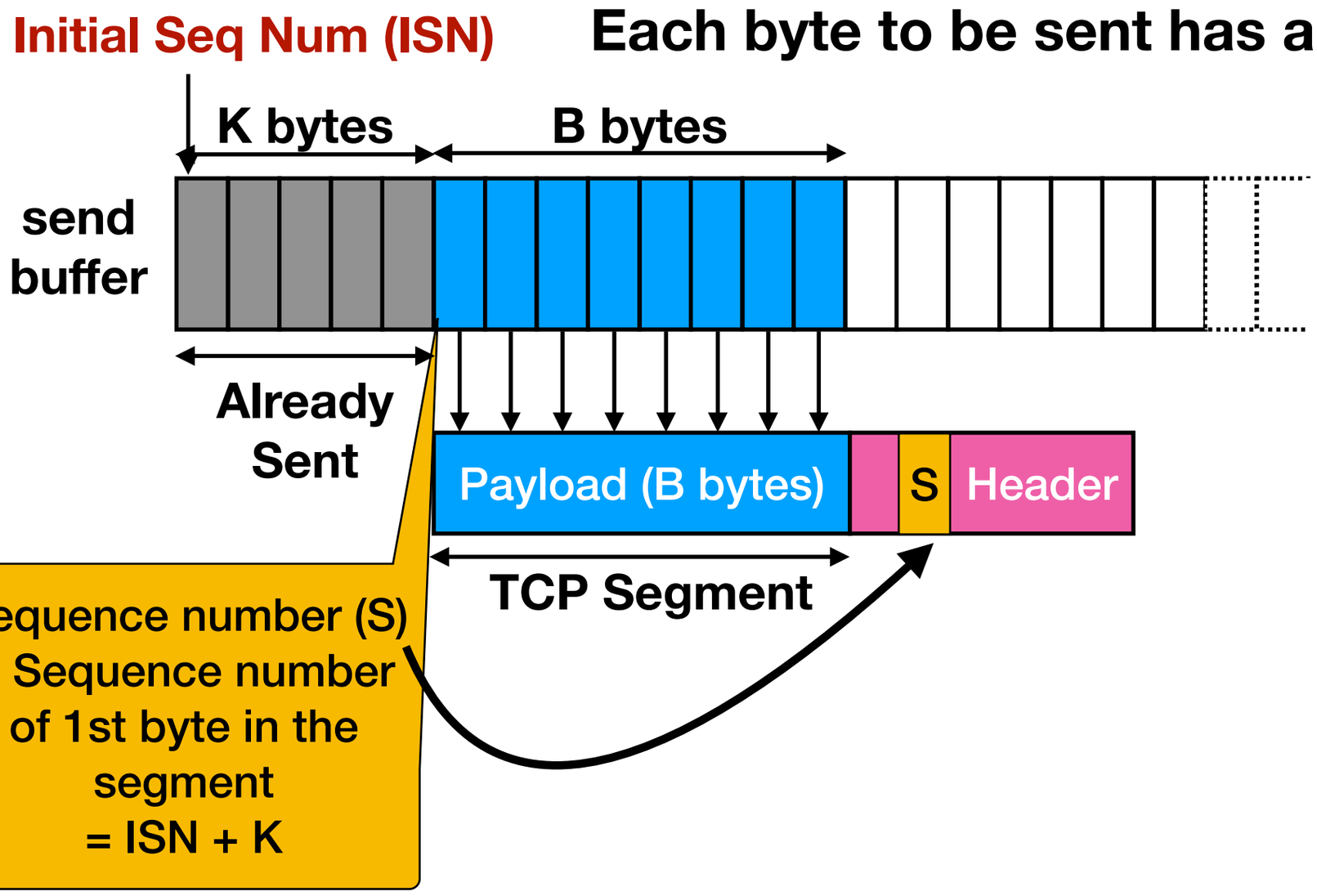
Sequence Number and ACK

Initial Seq Num (ISN)

Each byte to be sent has a sequence number



Sequence Number and ACK

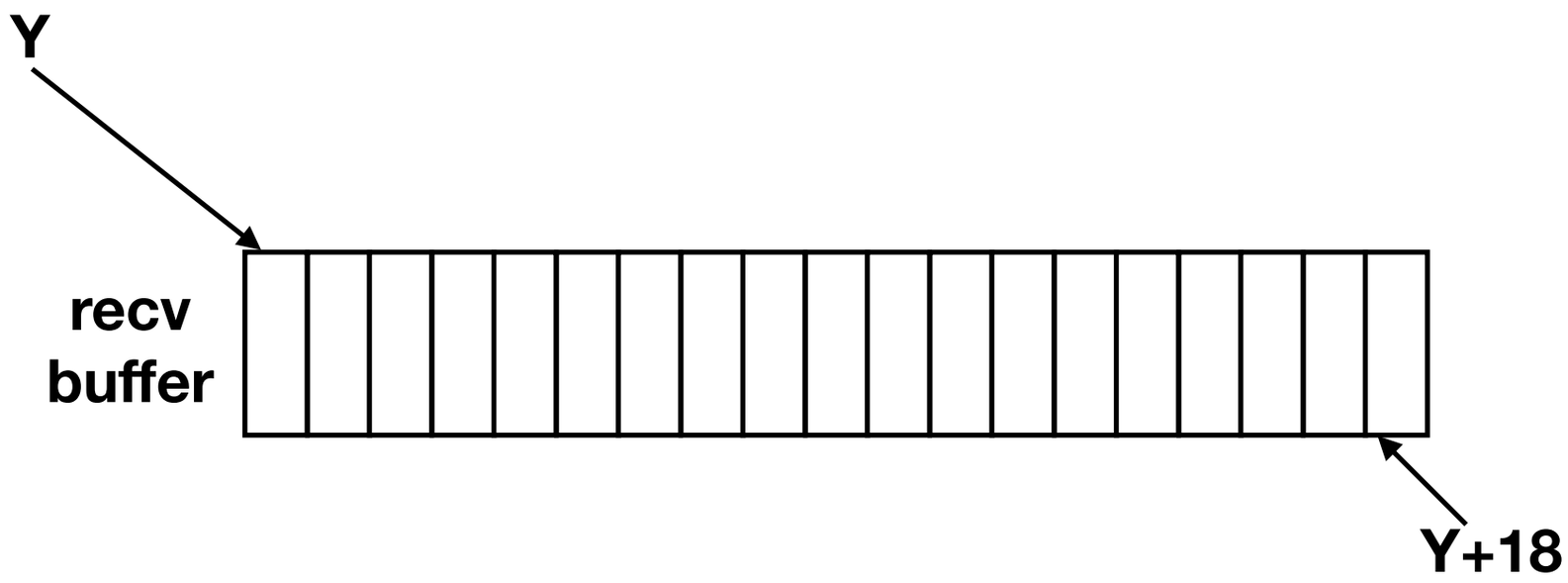


Sequence Number and ACK

- **Sender sends TCP packets with TCP segments (byte stream)**
  - Sequence number field in the packet header equals the sequence number of 1st byte in TCP segment (= S)
  - TCP segment contains B bytes, with sequence numbers
    - S, S+1, S+2, ….., S+B-1

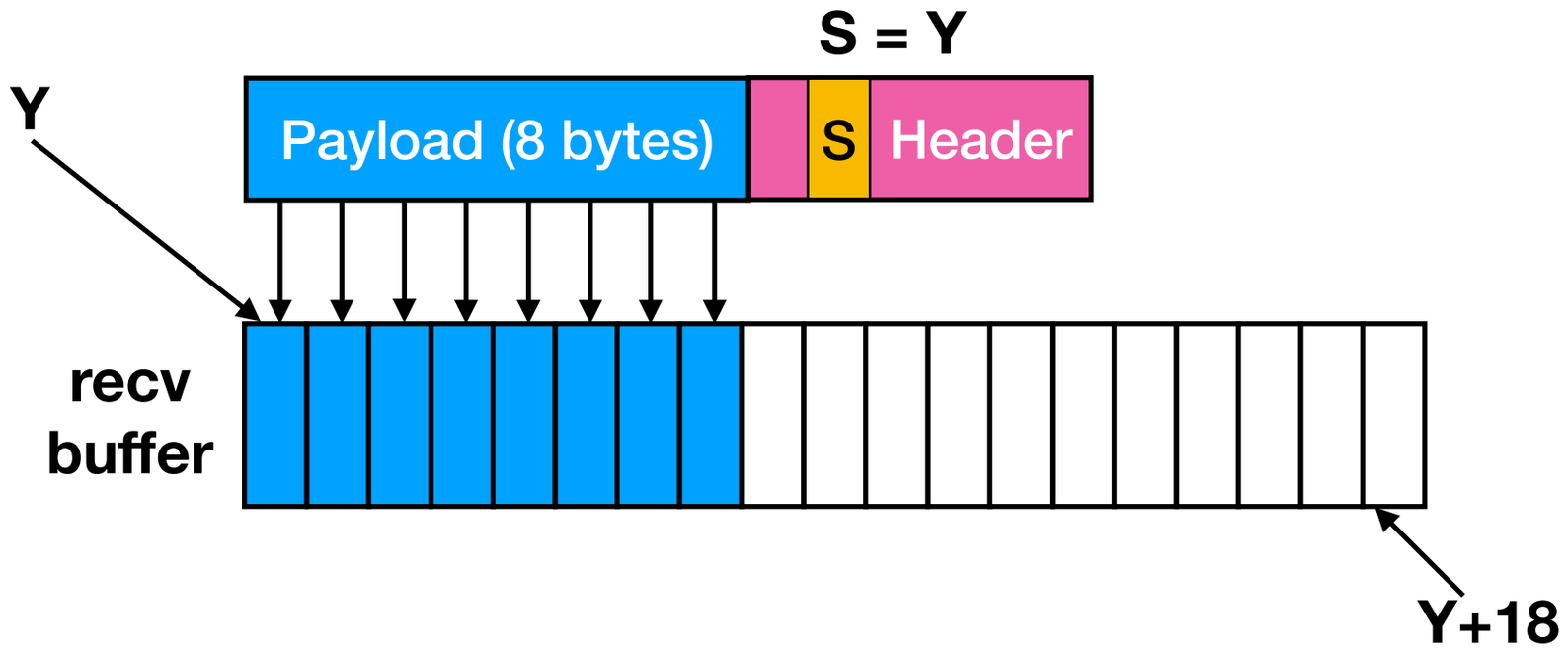
TCP Receive Buffer

- Suppose application has read all bytes until sequence number **Y-1**
  - Receive buffer can hold bytes with sequence numbers starting from **Y** to **Y + len(recv buffer) - 1**



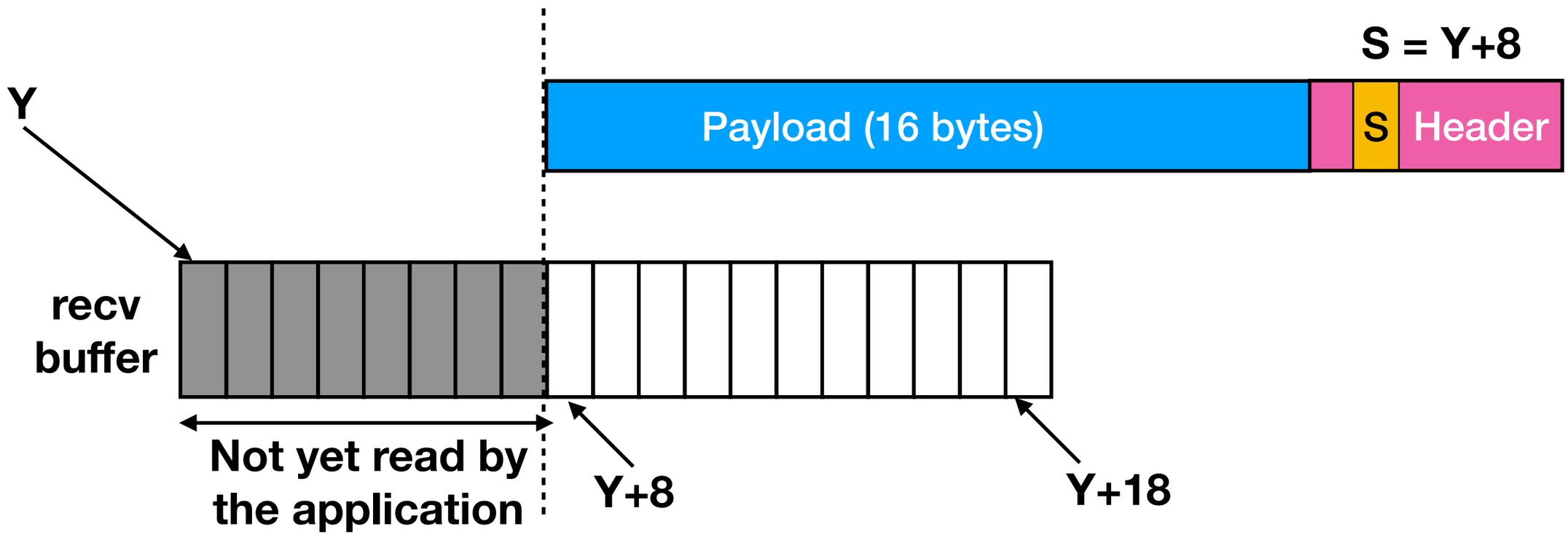
TCP Receive Buffer

- Suppose application has read all bytes until sequence number **Y-1**
  - Receive buffer can hold bytes with sequence numbers starting from **Y** to **Y + len(recv buffer) - 1**



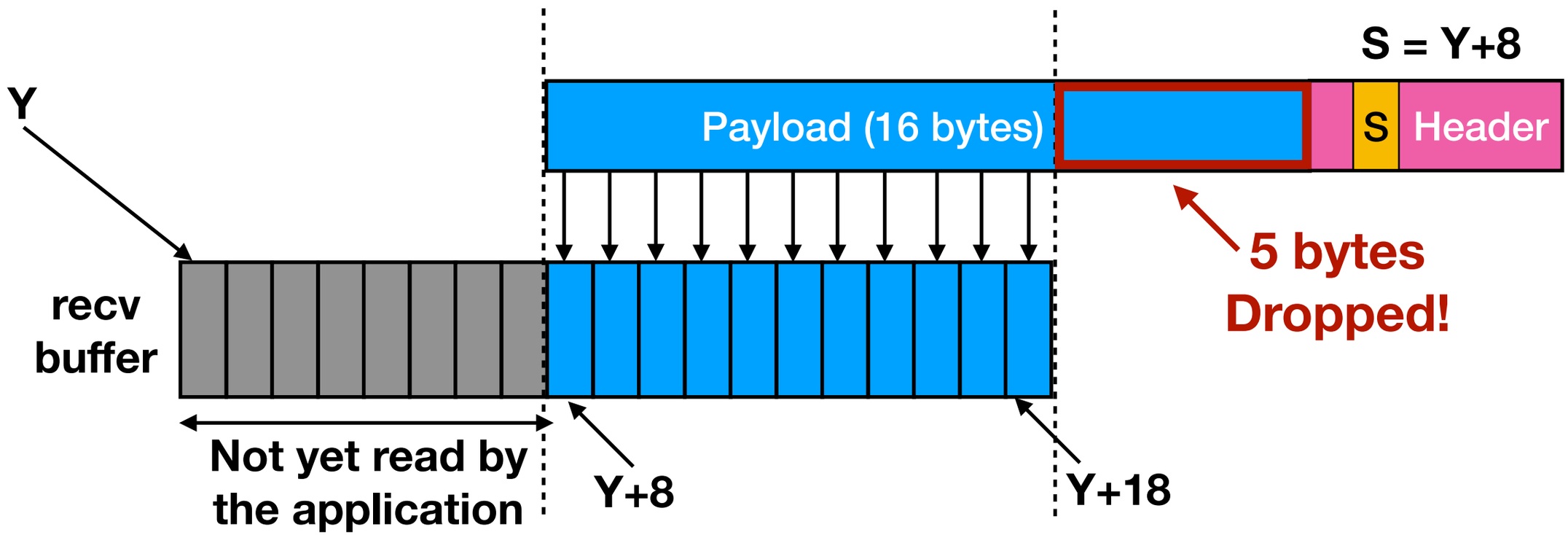
TCP Receive Buffer

- Suppose application has read all bytes until sequence number **Y-1**
  - Receive buffer can hold bytes with sequence numbers starting from **Y** to **Y + len(recv buffer) - 1**



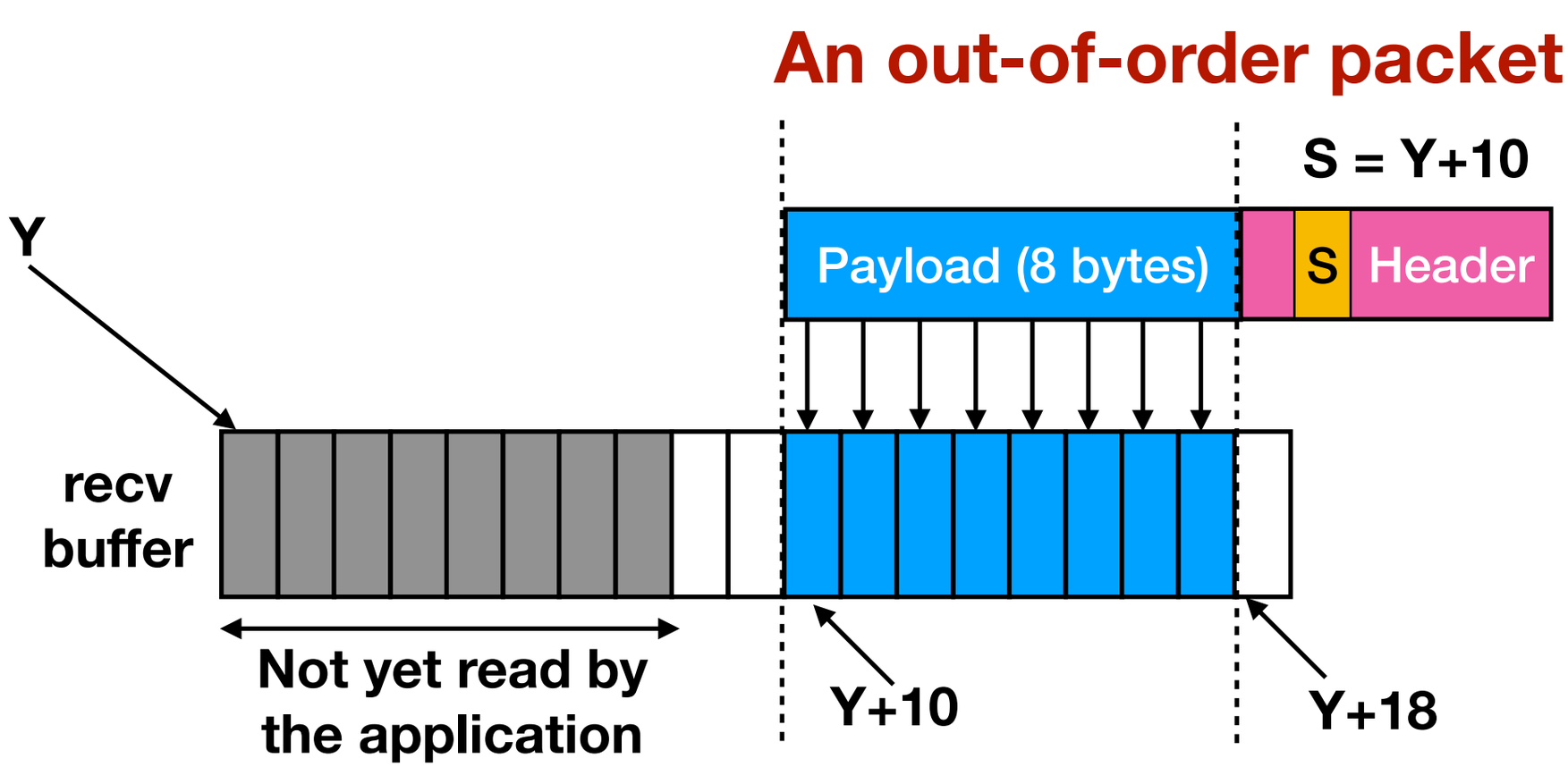
TCP Receive Buffer

- Suppose application has read all bytes until sequence number **Y-1**
  - Receive buffer can hold bytes with sequence numbers starting from **Y** to **Y + len(recv buffer) - 1**



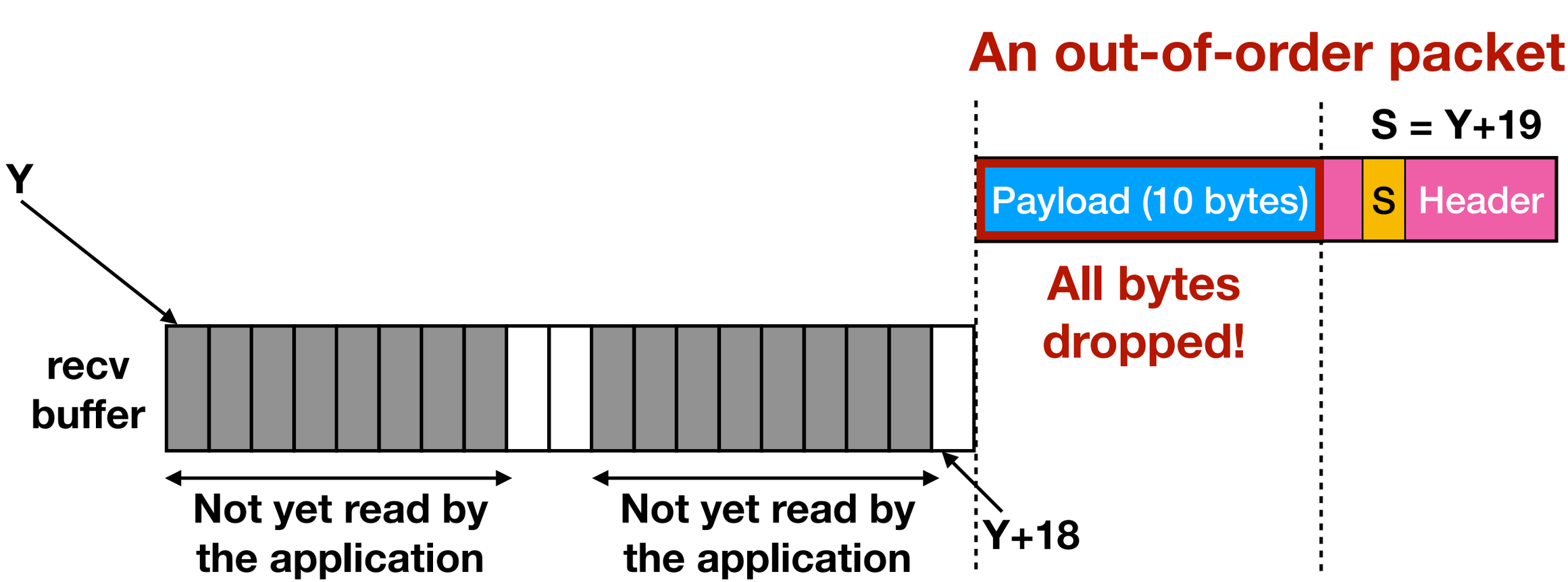
TCP Receive Buffer

- Suppose application has read all bytes until sequence number **Y-1**
  - Receive buffer can hold bytes with sequence numbers starting from **Y** to **Y + len(recv buffer) - 1**



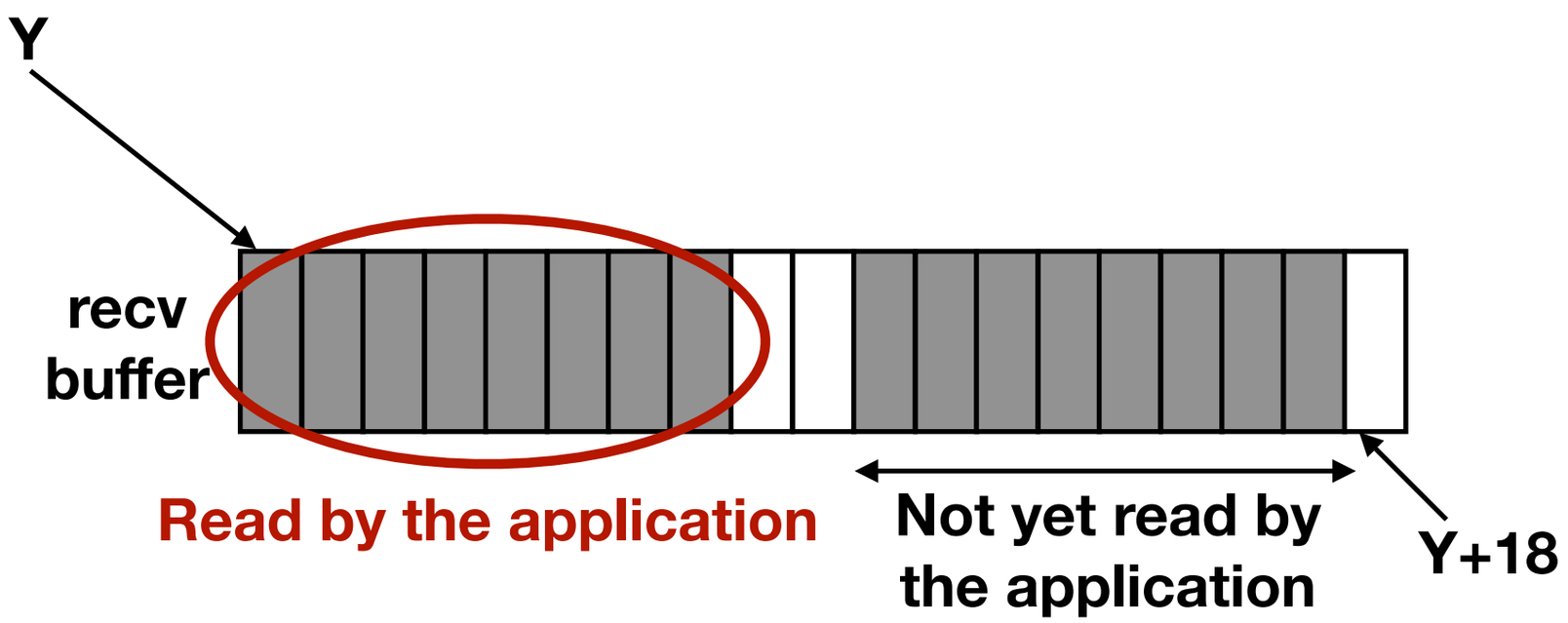
TCP Receive Buffer

- Suppose application has read all bytes until sequence number **Y-1**
  - Receive buffer can hold bytes with sequence numbers starting from **Y** to **Y + len(recv buffer) - 1**



TCP Receive Buffer

- Suppose application has read all bytes until sequence number **Y+7**

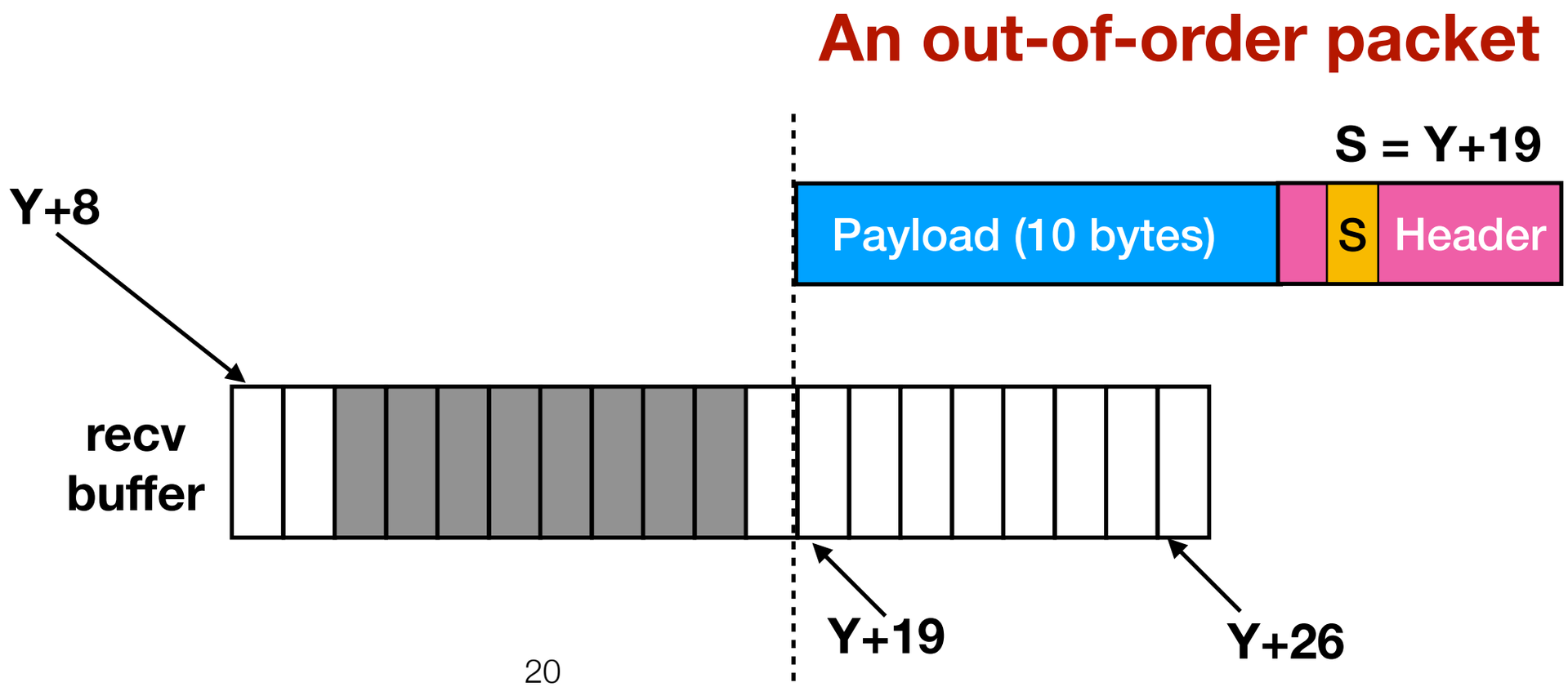


TCP Receive Buffer

- Suppose application has read all bytes until sequence number **Y+7**
  - Receive buffer can now hold bytes with sequence numbers starting from **Y+8** to **Y+8 + len(recv buffer) - 1**

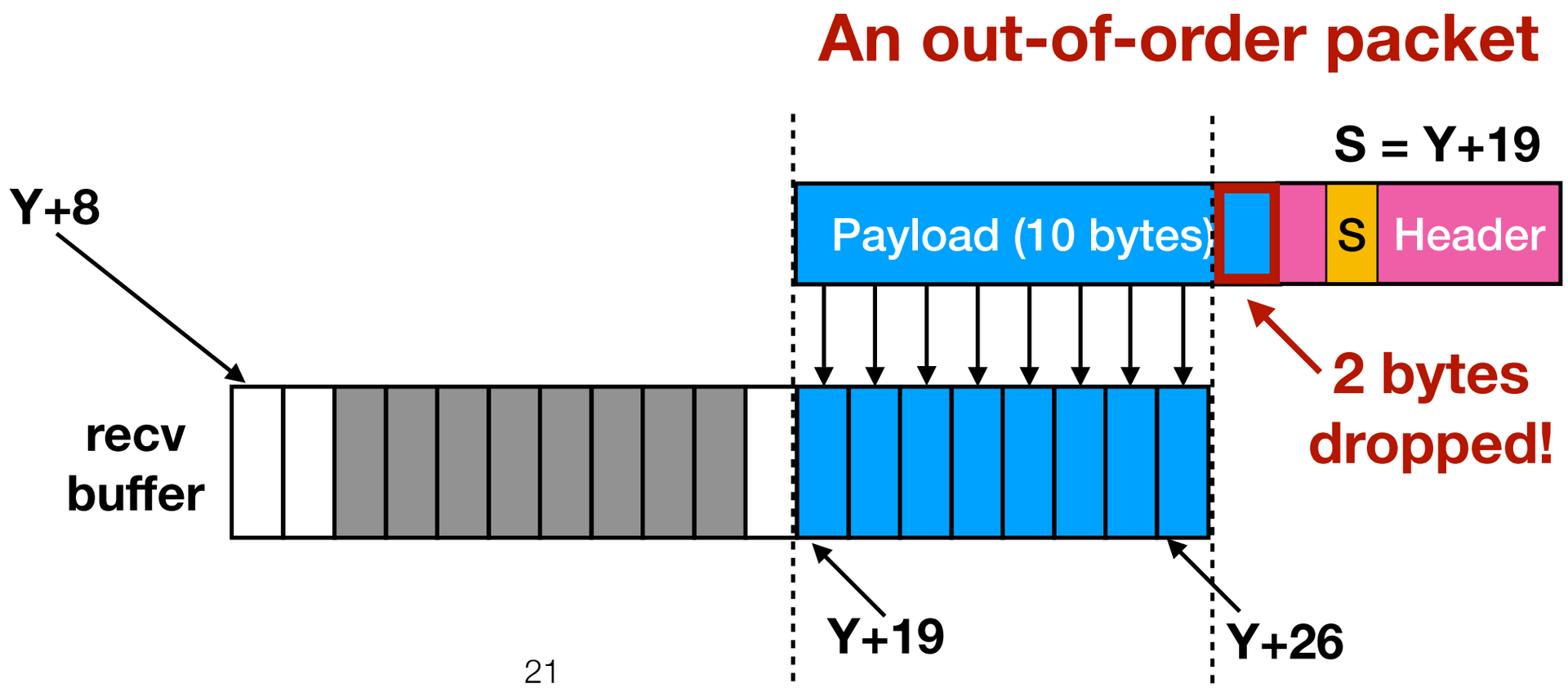
TCP Receive Buffer

- Suppose application has read all bytes until sequence number **Y+7**
  - Receive buffer can now hold bytes with sequence numbers starting from **Y+8** to **Y+8 + len(recv buffer) - 1**



TCP Receive Buffer

- Suppose application has read all bytes until sequence number **Y+7**
  - Receive buffer can now hold bytes with sequence numbers starting from **Y+8** to **Y+8 + len(recv buffer) - 1**



Sequence Number and ACK

- **Sender sends TCP packets with TCP segments (byte stream)**
  - Sequence number field in the packet header equals the sequence number of 1st byte in TCP segment (= S)
  - TCP segment contains B bytes, with sequence numbers
    - S, S+1, S+2, ….., S+B-1

- **Upon receipt of a TCP packet, receiver sends an ACK**
  - Most popular TCP implementations use **Cumulative ACK**

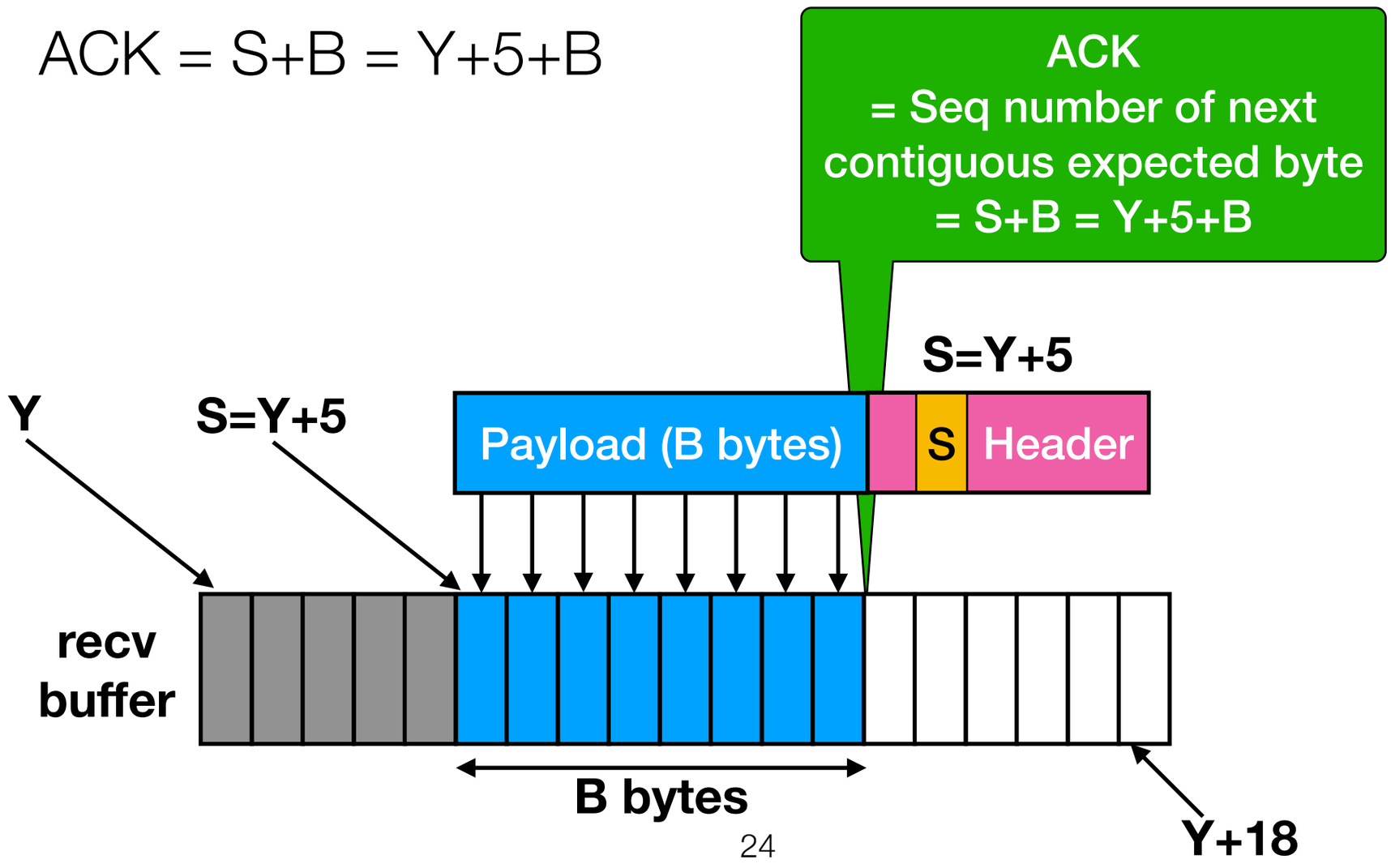
Sequence Number and ACK

- **Sender sends TCP packets with TCP segments (byte stream)**
  - Sequence number field in the packet header equals the sequence number of 1st byte in TCP segment (= S)
  - TCP segment contains B bytes, with sequence numbers
    - S, S+1, S+2, ….., S+B-1

- **Upon receipt of a TCP packet, receiver sends an ACK**
  - Most popular TCP implementations use **Cumulative ACK**
    - ACK contains the sequence number of ***next*** **contiguous** byte expected by the receiver

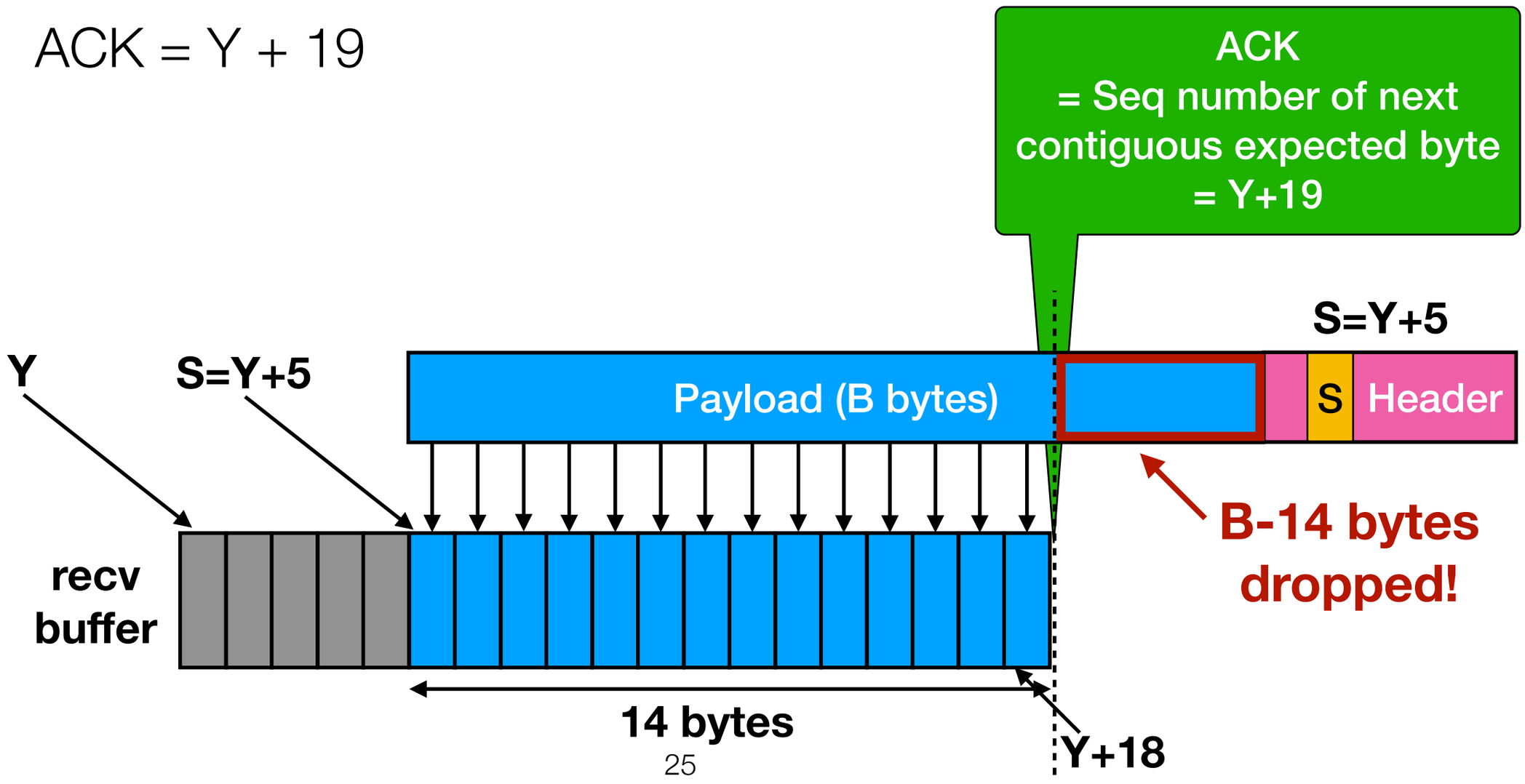
TCP Cumulative ACK

- Suppose bytes upto Y+4 received and bytes upto Y-1 read by application
  - On receiving a TCP packet with sequence num S and B bytes of data,
    - **Case 1a: All bytes prior to S have been received, i.e., S = Y+5, and received packet fits in the available recv buffer space**
      - ACK = S+B = Y+5+B



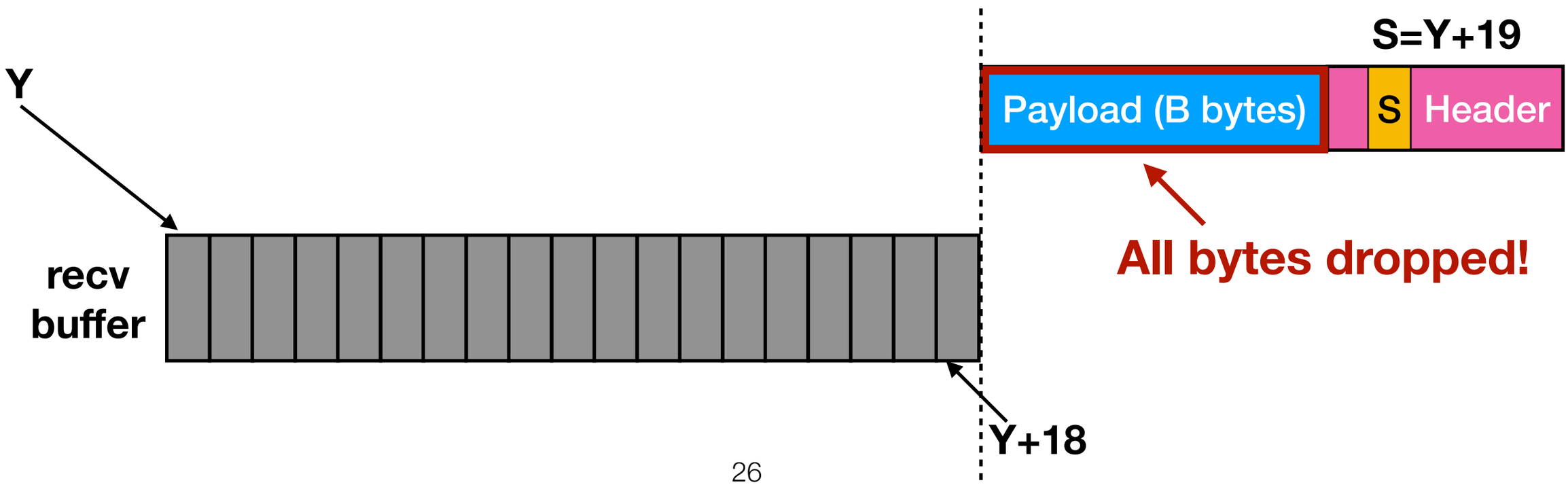
TCP Cumulative ACK

- Suppose bytes upto Y+4 received and bytes upto Y-1 read by application
  - On receiving a TCP packet with sequence num S and B bytes of data,
    - **Case 1b: All bytes prior to S have been received, i.e., S = Y+5, and received packet doesn't fit in available recv buffer space**
      - ACK = Y + 19



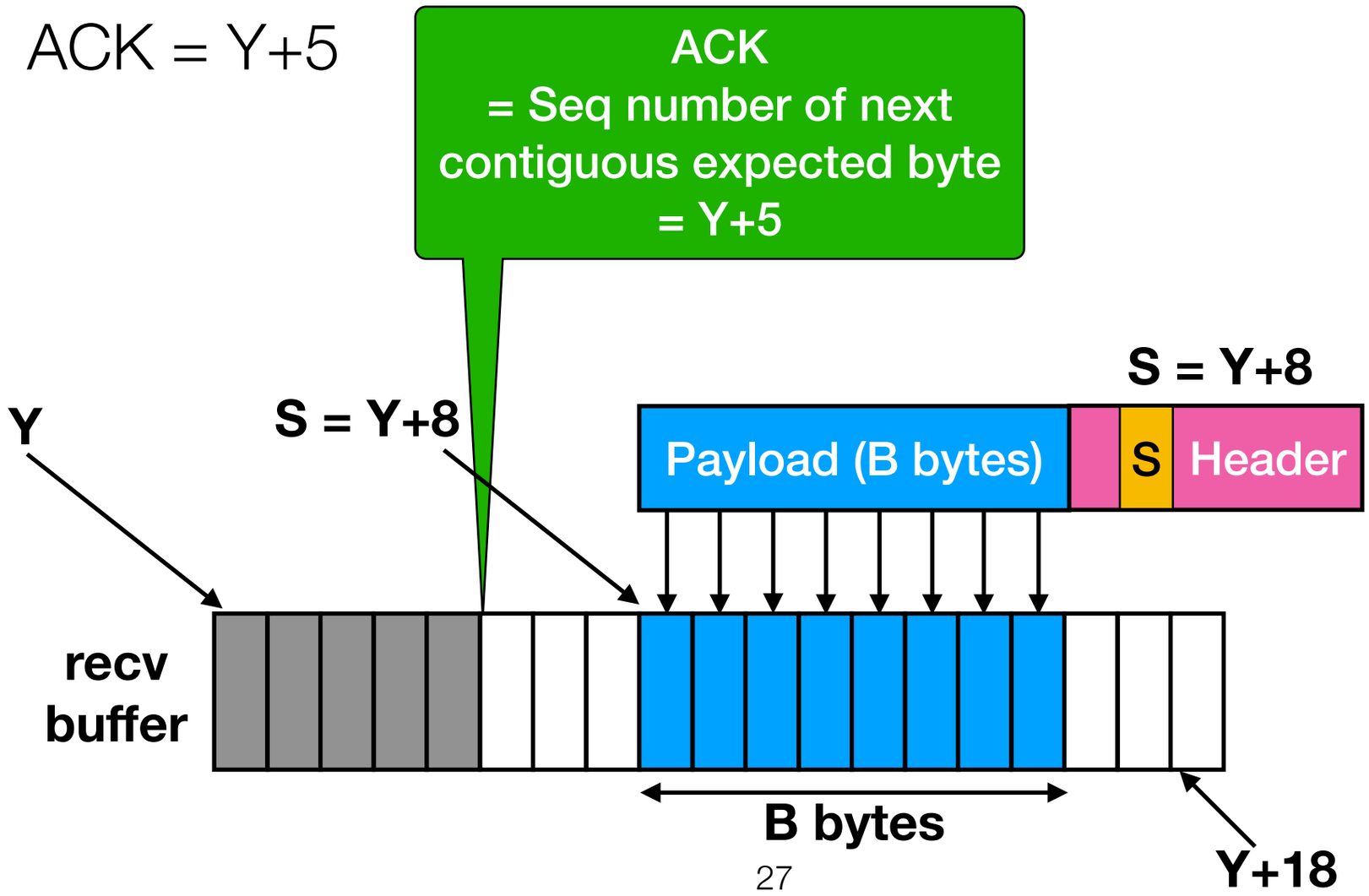
TCP Cumulative ACK

- Suppose bytes upto Y+4 received and bytes upto Y-1 read by application
  - On receiving a TCP packet with sequence num S and B bytes of data,
    - **Case 1c: All bytes prior to S have been received, i.e., S = Y+19, and recv buffer is full**
      - Entire packet is discarded … No bytes received … No ACK sent!



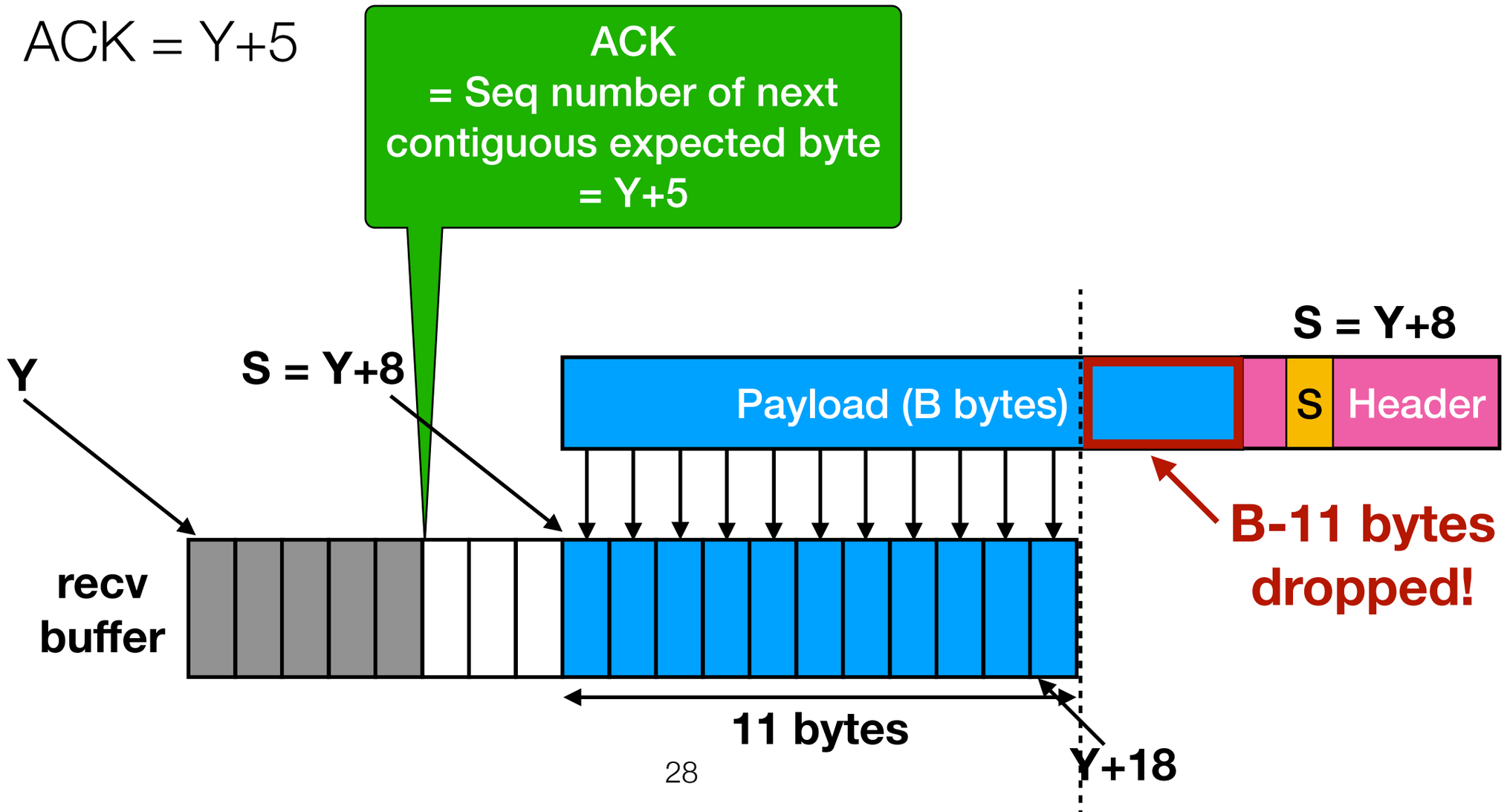
TCP Cumulative ACK

- Suppose bytes upto Y+4 received and bytes upto Y-1 read by application
  - On receiving a TCP packet with sequence num S and B bytes of data,
    - **Case 2a: Received an out-of-order packet, i.e., S > Y+5, and the received packet fits in available recv buffer space**
      - ACK = Y+5



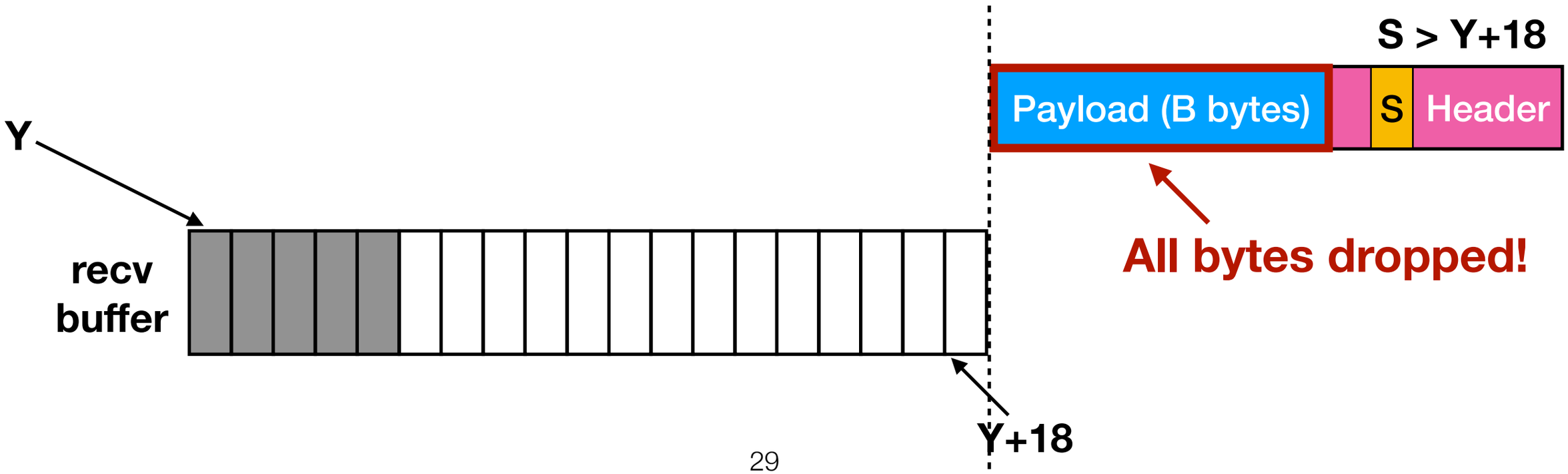
TCP Cumulative ACK

- Suppose bytes upto Y+4 received and bytes upto Y-1 read by application
  - On receiving a TCP packet with sequence num S and B bytes of data,
    - **Case 2b: Received an out-of-order packet, i.e., S > Y+5, and received packet doesn't fit in available recv buffer space**
      - ACK = Y+5



TCP Cumulative ACK

- Suppose bytes upto Y+4 received and bytes upto Y-1 read by application
  - On receiving a TCP packet with sequence num S and B bytes of data,
    - **Case 2c: Seq num of out-of-order pkt exceeds recv buffer size**
      - Entire packet is discarded … No bytes received … No ACK sent!



Retransmissions

TCP Retransmission Mechanism

- **Under what conditions should a sender retransmit a packet?**
  - **Assuming Cumulative ACKing,**
    - Duplicate ACKs
    - Timeout
- **What packets should a sender retransmit?**
  - Assuming Cumulative ACKing, use **Go-Back-N**

TCP Retransmission Mechanism

- **Under what conditions should a sender retransmit a packet?**
  - **Assuming Cumulative ACKing,**
    - Duplicate ACKs
    - Timeout
- **What packets should a sender retransmit?**
  - Assuming Cumulative ACKing, use **Go-Back-N**

Duplicate ACKs (Fast Retransmit)

- **Example:**
  - Sender sends 100B packets with ISN 100
  - Sequence numbers of sent packets:
    - 100, 200, 300, 400, 500, 600, 700, 800, 900, …..
  - Assume 5th packet (sequence num 500) is lost but no others
  - Stream of ACKs will be:
    - 200, 300, 400, 500, 500, 500, 500, 500, 500, …..

Stream of duplicate ACKs; implies 500 has not been delivered

TCP retransmits after receiving 3 Duplicate ACKs

Why not retransmit immediately after receiving the first Duplicate ACK?

500 could have been **reordered/delayed** in the network, and ***not*** lost…

Waiting for 3 Dup ACKs before retransmitting can avoid unnecessary retransmissions

Timeout

- **Why need timeouts? Why (Duplicate) ACKs are not enough?**

Timeout

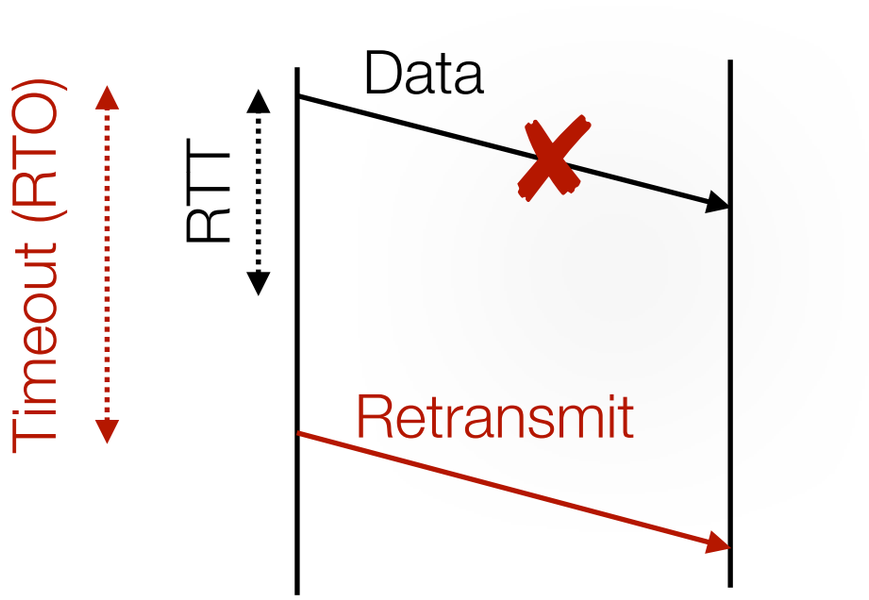
- **Why need timeouts? Why (Duplicate) ACKs are not enough?**
  - ACKs can only be generated if **some** packets are being received
  - What if a TCP stream only has 1 packet, and that packet is lost?
  - Or, the last packet in a TCP stream is lost?
  - Or, a retransmitted packet is lost?
  - … will need a timer to detect loss!

Timeout

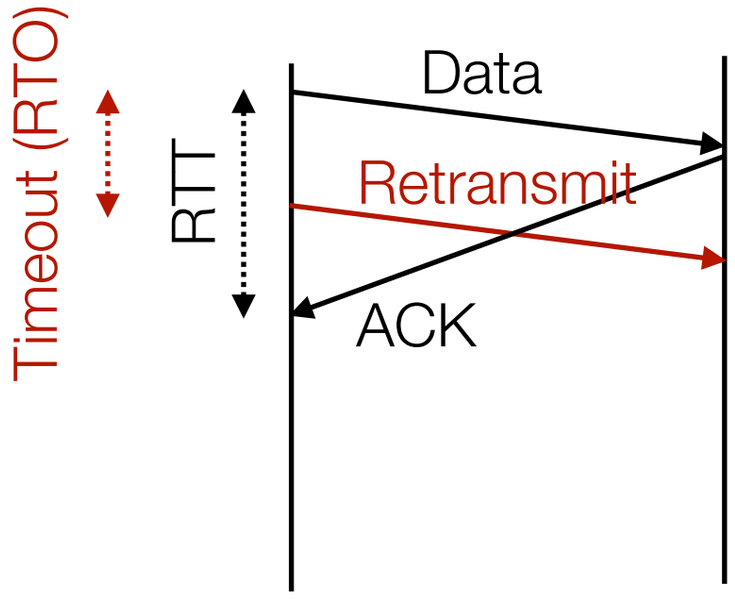
- **Why need timeouts? Why (Duplicate) ACKs are not enough?**
  - ACKs can only be generated if **some** packets are being received
  - What if a TCP stream only has 1 packet, and that packet is lost?
  - Or, the last packet in a TCP stream is lost?
  - Or, a retransmitted packet is lost?
    - … will need a timer to detect loss!
- Most TCP implementations have a **single** timer
- TCP resets timer whenever ***new*** data is ACKed
- Retransmits when the timer expires
- **RTO** (Retransmit Time Out) value inside the TCP code is used to configure the timeout value

Timeout

- **How to set the value of RTO?**



Too large => waiting too long to retransmit



Too small => unnecessarily retransmitting

- **Ideally, RTO = RTT … but need a very accurate estimation of RTT**
  - Very hard to do in practice, because of variable delays such as queuing delay
- **Most TCP implementations use a conservative value for RTO, e.g., 500ms**
  - Hence, retransmission by timeout is very slow!
  - So, TCP relies on Duplicate ACKs for **"Fast Retransmit"**

TCP Retransmission Mechanism

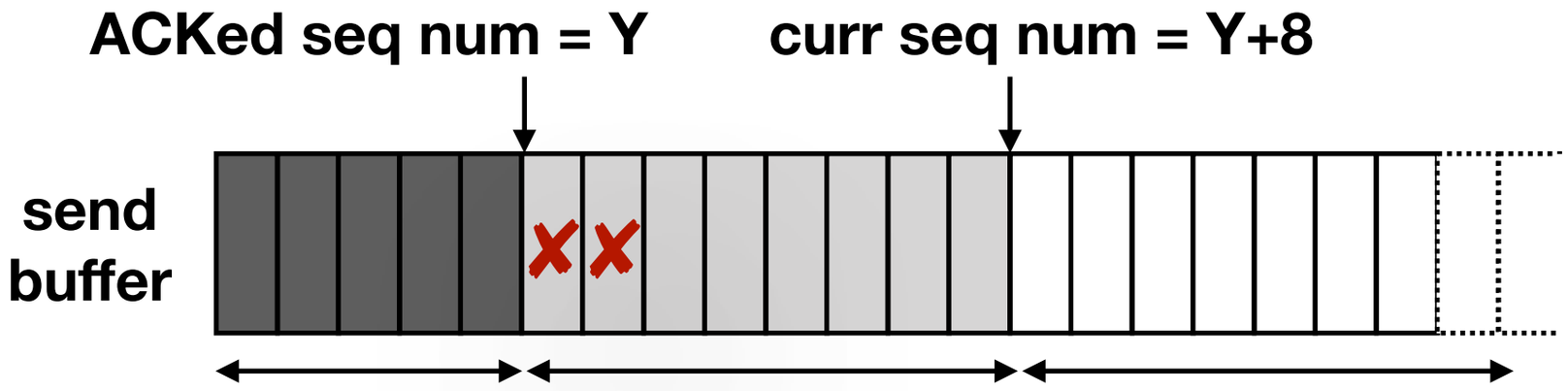
- **Under what conditions should a sender retransmit a packet?**
  - **Assuming Cumulative ACKing,**
    - Duplicate ACKs
    - Timeout

- **What packets should a sender retransmit?**
  - Assuming Cumulative ACKing, use **Go-Back-N**

Go-Back-N

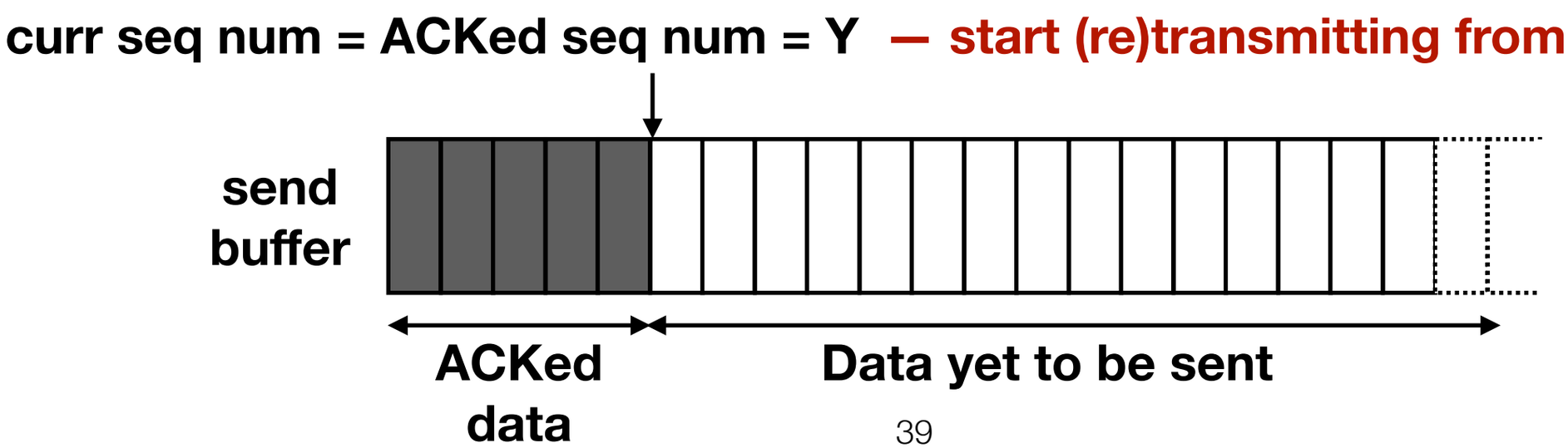
ACKed seq num = highest contiguous sequence num ACKed by receiver

curr seq num = sequence num of next byte that sender would send



Suppose some bytes starting from Y are lost → Will receive Dup ACKs with value Y

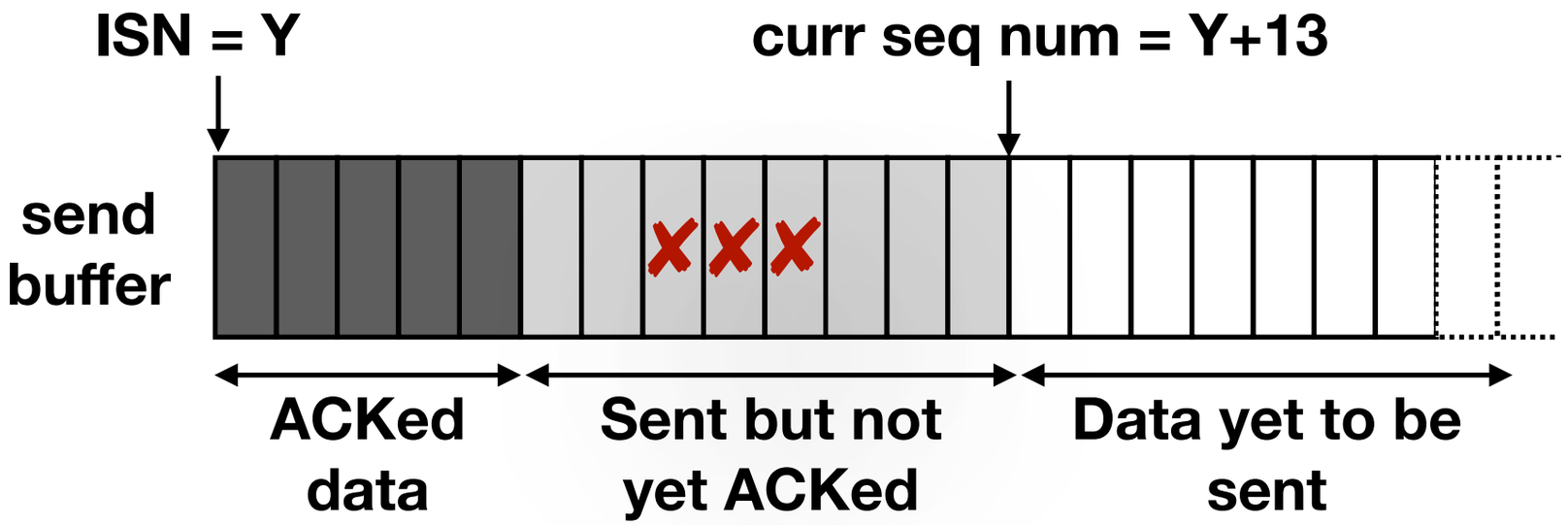
On receiving 3 Dup ACKs, sender will do **Go-Back-N**



Other TCP ACK and Retransmission Mechanisms

- Most TCP implementations use **Cumulative ACK** with **Go-Back-N**
- However, some TCP implementations also use
  - **Selective ACK** with **Selective Retransmit**
    - May result in better performance, e.g., fewer retransmitted packets
    - … at the cost of higher ACKing overhead
      - Selective ACK vs. Cumulative ACK
    - … and higher implementation complexity
      - Selective Retransmit vs. Go-Back-N

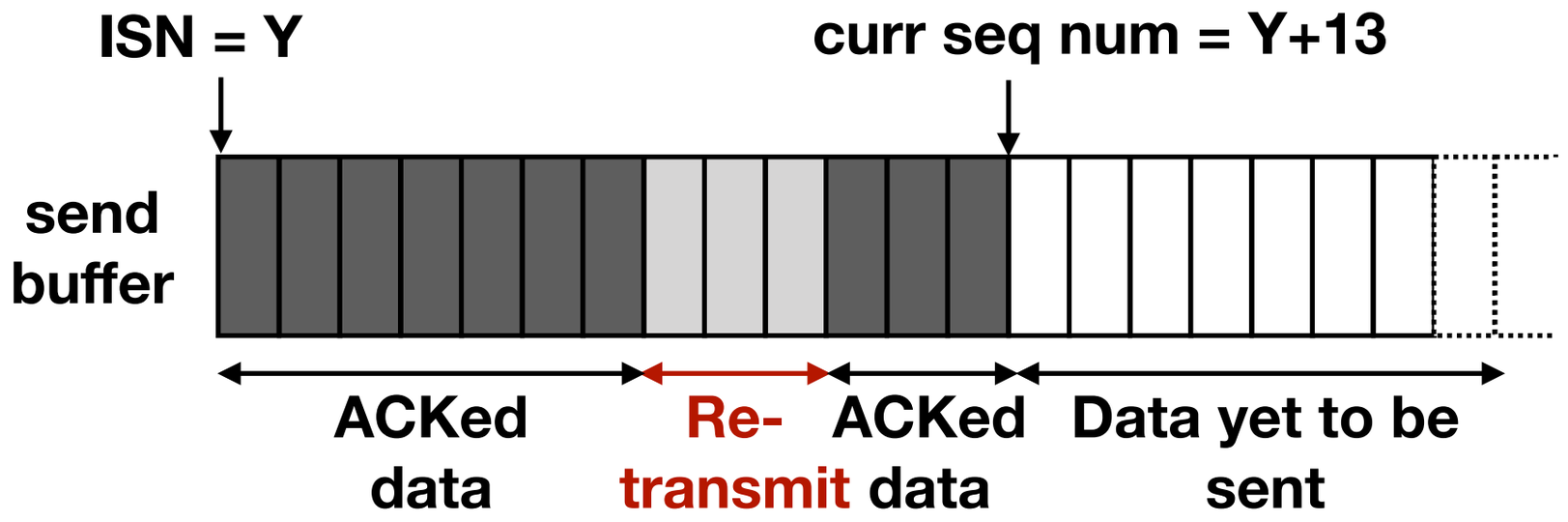
Selective ACK w/ Selective Retransmit



Suppose bytes Y+7, Y+8, Y+9 are lost

Will receive **Selective ACK** = (up to Y+6, plus Y+10, Y+11, Y+12)

On receiving the Selective ACK, sender will do **Selective Retransmit**



Transmission Control Protocol (TCP)

- **Reliable, In-order Delivery to Applications**
  - Application Multiplexing ✓
  - ACKs and Retransmissions ✓
  - Flow Control
  - Congestion Control

- **Byte Stream Abstraction** ✓
  - Communicates with the applications using a byte stream and *not* packets

- **Connection-oriented**
  - Pairwise <sender, receiver> connection is established before sending data
  - Used to maintain connection-specific state, e.g., initial seq num, window scaling factor, window size, etc.

Questions?