CS162
Operating Systems and Systems Programming
Lecture 26

TCP Flow Control (finished)

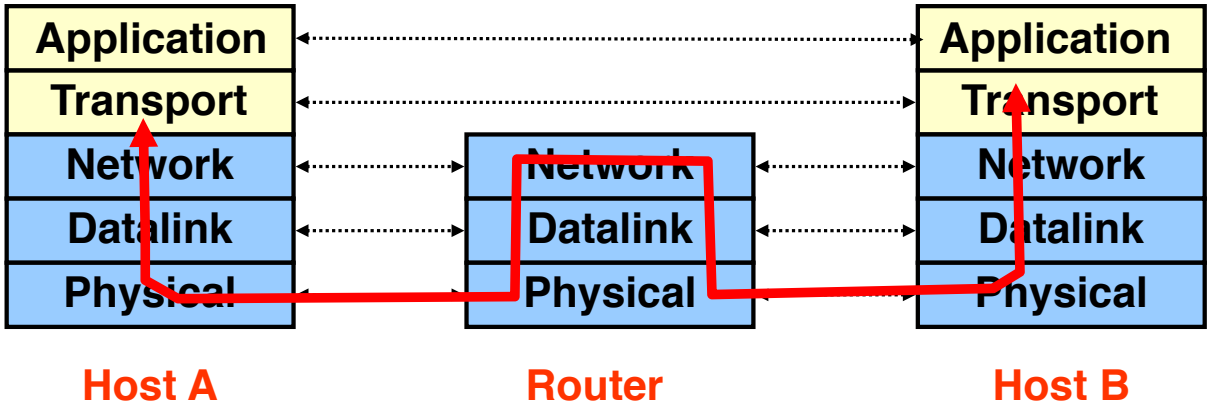
November 30th, 2021

Prof. Ion Stoica

http://cs162.eecs.Berkeley.edu

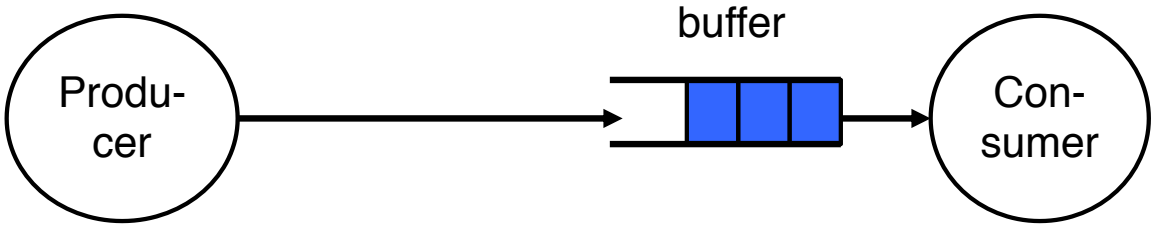
Recall: Transport Layer

- Only implemented at end-hosts (not in the network)
- UDP and TCP: provide multiplexing/demultiplexing to processes using port numbers
- TCP only
  - Flow control: don't over-flow the receiver
  - Congestion control: don't over-flow the network



Recall: Flow Control

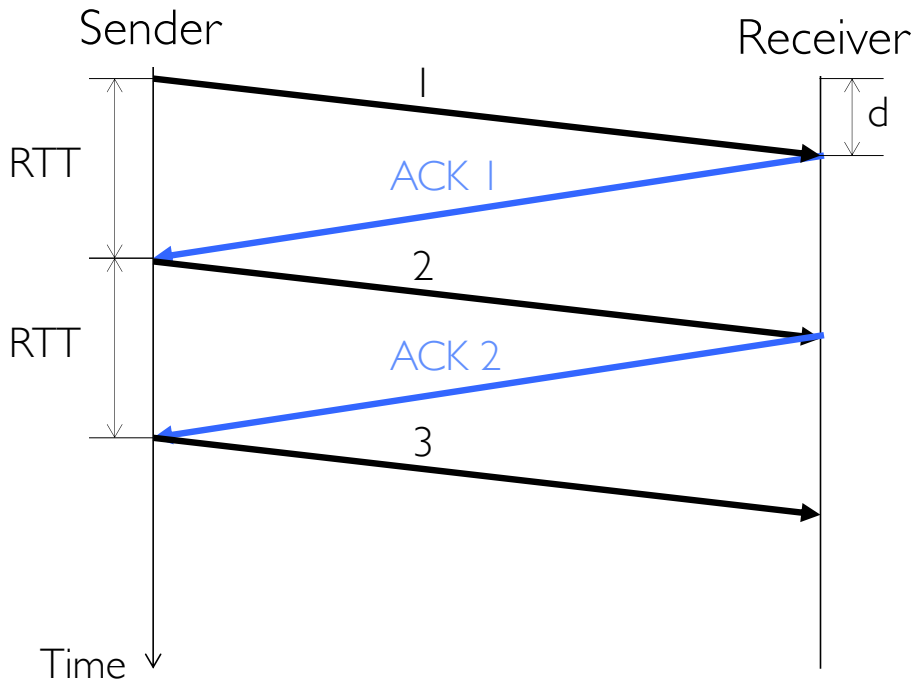
- Recall: Flow control ensures a fast sender does not overwhelm a slow receiver
- Example: Producer-consumer with bounded buffer (Lecture 5)
  - A buffer between producer and consumer
  - Producer puts items into buffer as long as buffer **not full**
  - Consumer consumes items from buffer



Stop-and-Wait

- Send; wait for ACK; repeat
- Round Trip Time (RTT): time it takes a packet to travel from sender to receiver and back
  - One-way latency (d): one way delay from sender and receiver
- For symmetric latency,

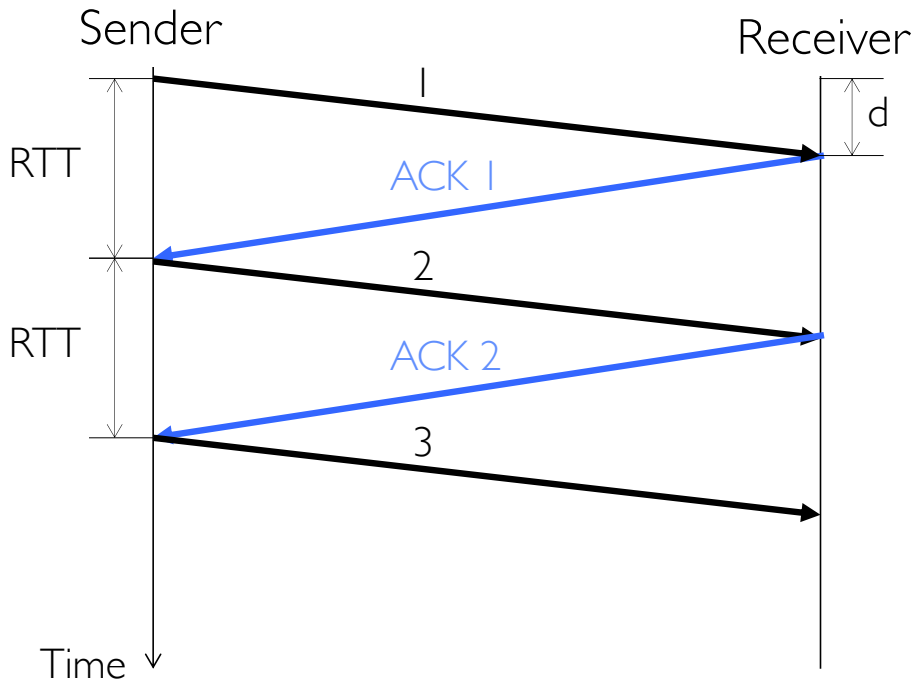
$$RTT = 2d$$



Stop-and-Wait

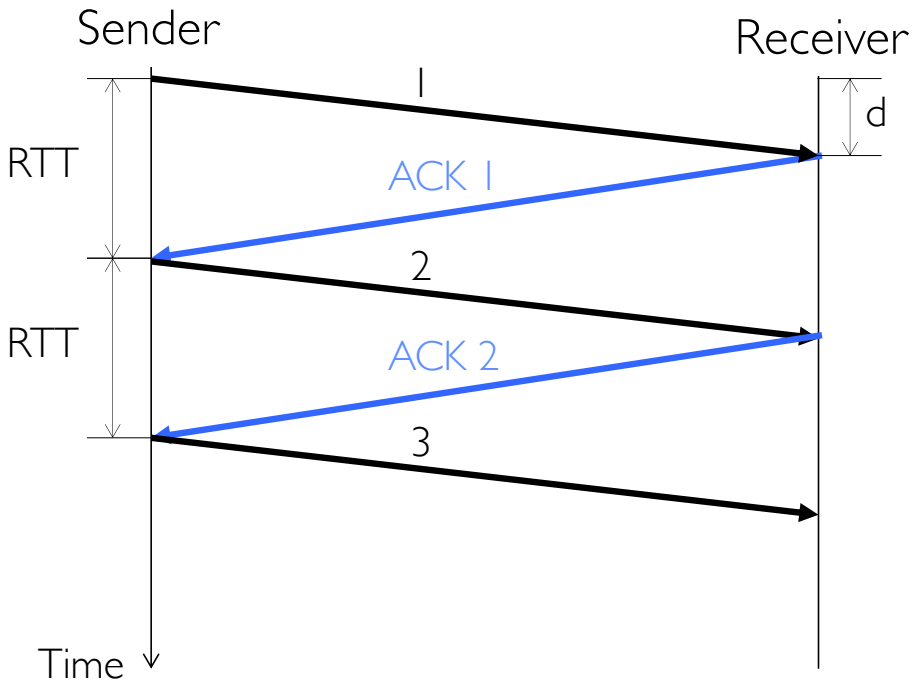
- How fast can you send data?
- Little's Law applied to the network:
$$n = B \cdot \text{RTT}$$
- For Stop-and-Wait, $n = 1$ packet

- So bandwidth is 1 packet per RTT
  - Depends only on latency, not network capacity (!)



Stop-and-Wait

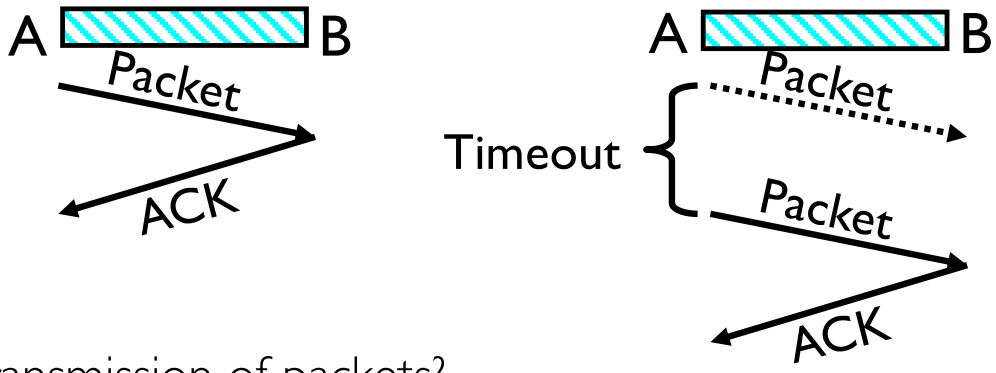
- So bandwidth is 1 packet per RTT
  - Depends only on latency, not network capacity (!)
- Suppose RTT = 100 ms and 1 packet is 1500 bytes
- Throughput = $\frac{1500 \cdot 8}{0.1}$ = 120 Kbps
- Very inefficient if we have a 100 Mbps link!



Problem: Dropped Packets

- All physical networks can garble or drop packets
  - Physical hardware problems (bad wire, bad signal)
- Therefore, IP can garble or drop packets
  - It doesn't repair this itself (end-to-end principle!)
- Building reliable message delivery
  - Confirm that packets aren't garbled
  - Confirm that packets arrive **exactly once**

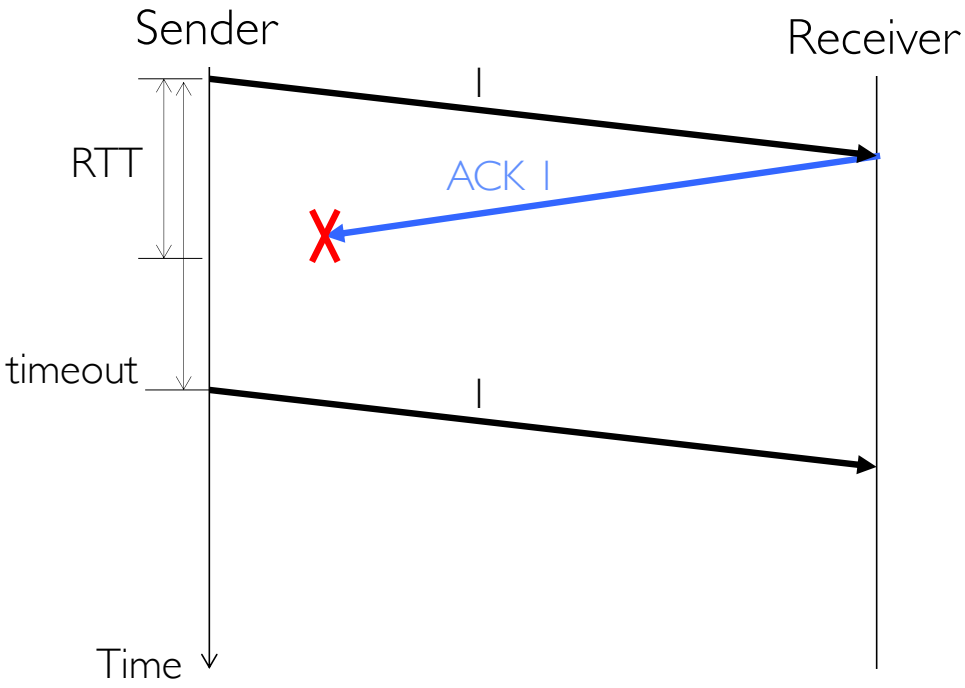
Using Acknowledgements



- How to ensure transmission of packets?
  - Detect garbling at receiver via checksum, discard if bad
  - Receiver acknowledges (by sending "ACK") when packet received properly at destination
  - Timeout at sender: if no ACK, retransmit
- Some questions:
  - If the sender doesn't get an ACK, does that mean the receiver didn't get the original message?
    - » No
  - What if ACK gets dropped? Or if message gets delayed?
    - » Sender doesn't get ACK, retransmits, Receiver gets message twice, ACK each

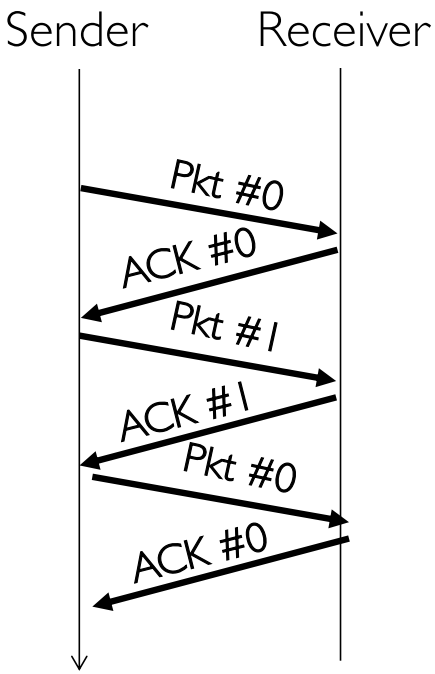
Stop-and-Wait with Packet Loss

- Loss recovery relies on timeouts
- How to choose a good timeout?
  - Too short – lots of duplication
  - Too long – packet loss is really disruptive!
- How to deal with duplication?
  - Retransmission certainly opens up the possibility for



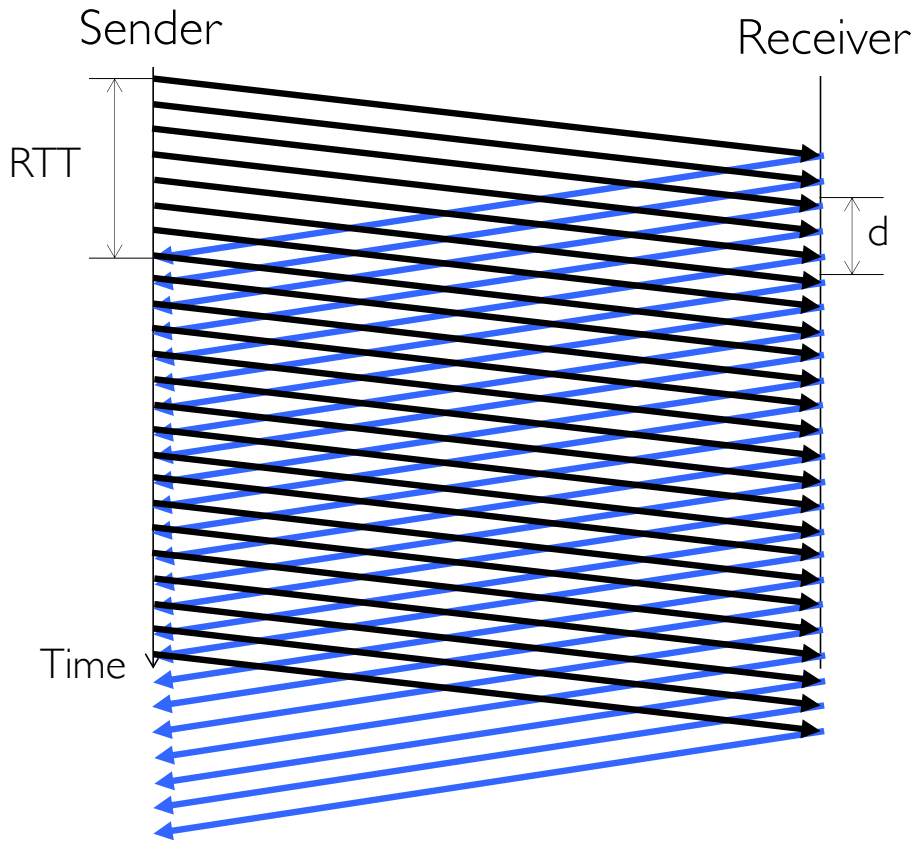
How to Deal with Message Duplication?

- Solution: put sequence number in message to identify re-transmitted packets
  - Receiver checks for duplicate number's; Discard if detected
- Requirements:
  - Sender keeps copy of unACK'd messages
    - » Easy: only need to buffer messages
  - Receiver tracks possible duplicate messages
    - » Hard: when ok to forget about received message?
- Alternating-bit protocol:
  - Send one message at a time; don't send next message until ACK received
  - Sender keeps last message; receiver tracks sequence number of last message received
- Pros: simple, small overhead
- Con: doesn't work if network can delay or duplicate messages arbitrarily



Advantages of Moving Away From Stop-and-Wait

- Larger space of acknowledgements
  - Pipelining: don't wait for ACK before sending next packet
- ACKs serve dual purpose:
  - Reliability: Confirming packet received
  - Ordering: Packets can be reordered at destination
- How much data is in flight now?
  - Bytes in-flight: $W_{send} = RTT \times B$
  - Here B is in "bytes/second"
  - $W_{send} \equiv$ Sender's "Window Size"
  - Packets in flight = (W_{send} / packet size)
- How long does the sender have to keep the packets around?
- How long does the receiver have to keep the packets' data?
- What if sender is sending packets faster than the receiver can process the data?



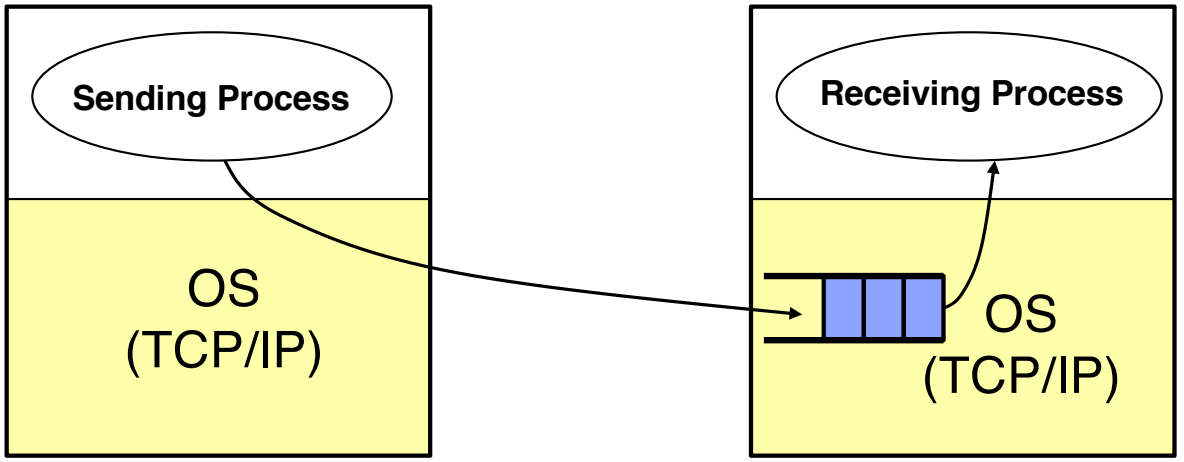
Recall: TCP Flow Control

- TCP: sliding window protocol at byte (not packet) level
  - Go-back-N: TCP Tahoe, Reno, New Reno
  - Selective Repeat (SR): TCP Sack

- Receiver tells sender how many more bytes it can receive without overflowing its buffer (i.e., AdvertisedWindow)

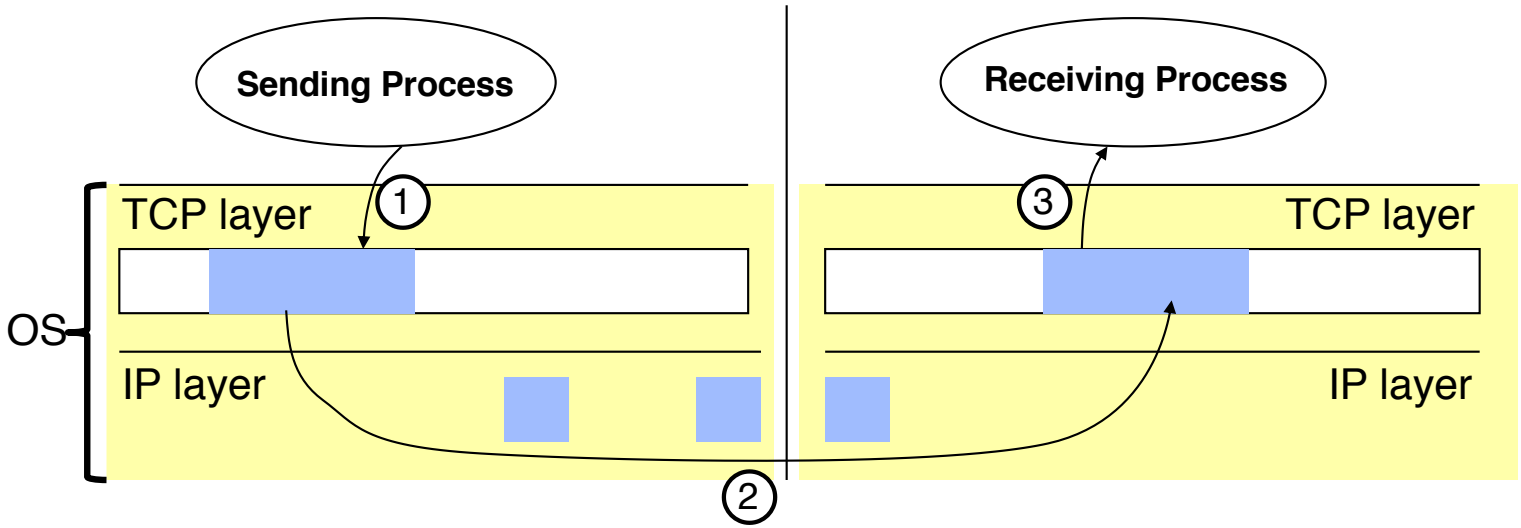
- The ack(nowledgement) contains sequence number N of next byte the receiver expects, i.e., receiver has received all bytes in sequence up to and including N-1

Recall: TCP Flow Control



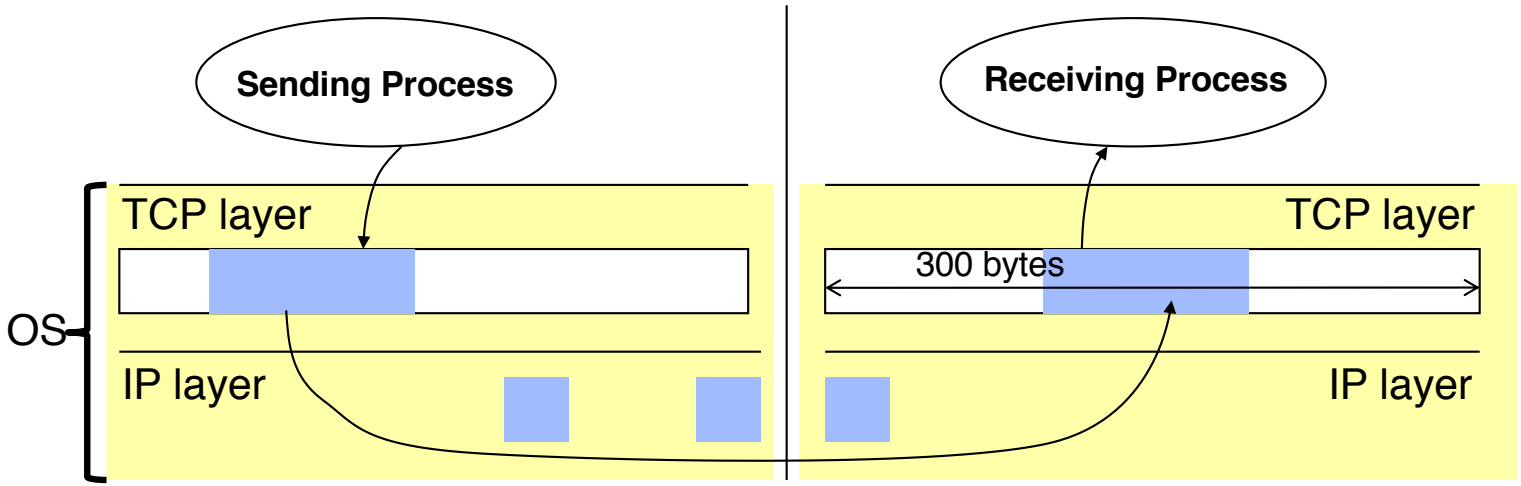
- TCP/IP implemented by OS (Kernel)
  - Cannot do context switching on sending/receiving every packet
    - » At 1Gbps, it takes 12 usec to send a 1500 byte packet, and 0.8usec to send a 100 byte packet
- Need buffers to match …
  - sending app with sending TCP
  - receiving TCP with receiving app

Recall: TCP Flow Control



- Three pairs of producer-consumer's
  - ① sending process → sending TCP
  - ② Sending TCP → receiving TCP
  - ③ receiving TCP → receiving process

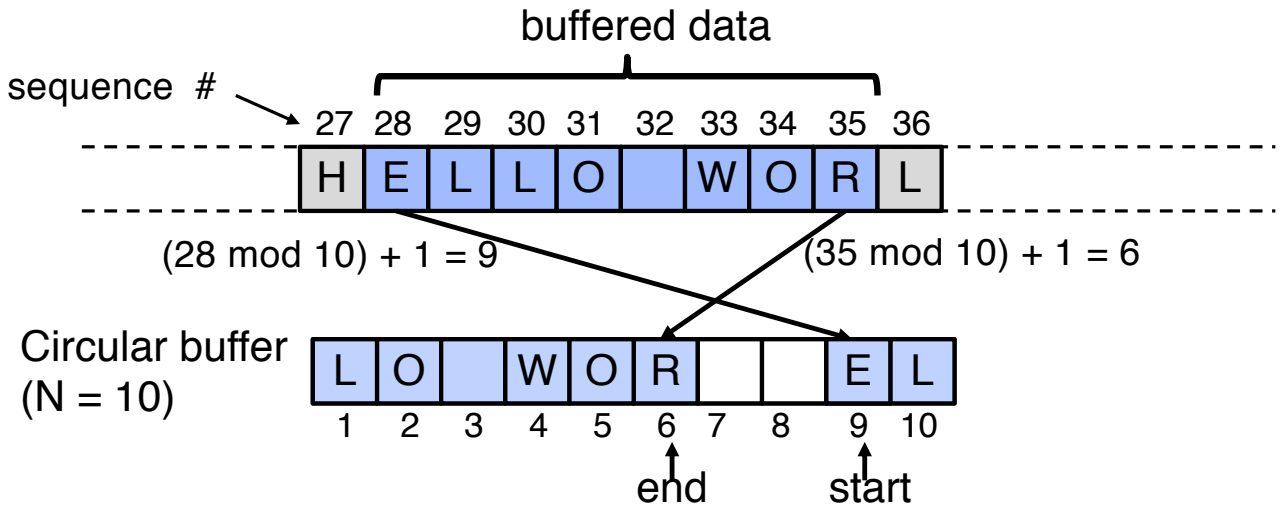
Recall: TCP Flow Control



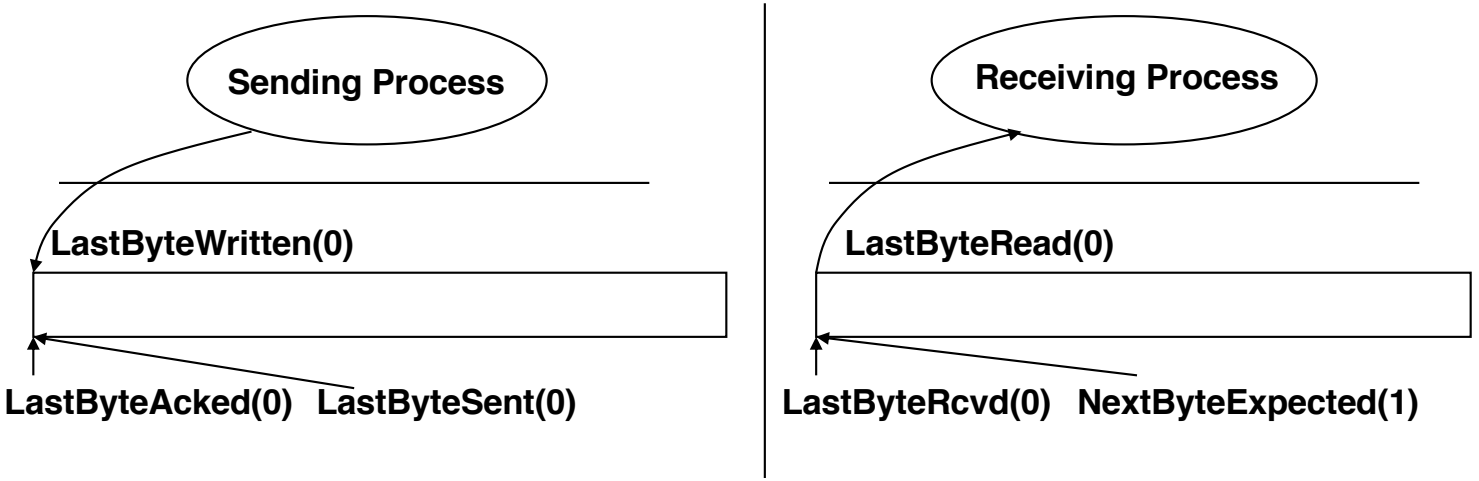
- Example assumptions:
  - Maximum IP packet size = 100 bytes
  - Size of the receiving buffer (MaxRcvBuf) = 300 bytes
- Recall, ack indicates the next expected byte in-sequence, not the last received byte
- Use circular buffers

Recall: Circular Buffer

- Assume
  - A buffer of size N
  - A stream of bytes, where bytes have increasing sequence numbers
    - » Think of stream as an unbounded array of bytes and of sequence number as indexes in this array
- Buffer stores at most N consecutive bytes from the stream
- Byte k stored at position (k mod N) + 1 in the buffer

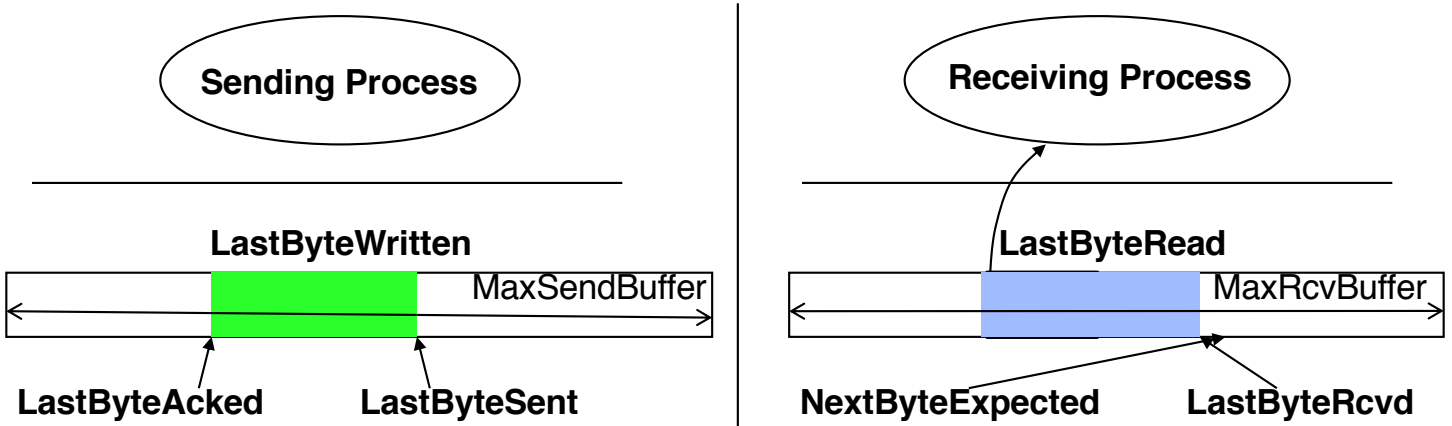


Recall: TCP Flow Control



- LastByteWritten: last byte written by sending process
- LastByteSent: last byte sent by sender to receiver
- LastByteAcked: last ack received by sender from receiver
- LastByteRcvd: last byte received by receiver from sender
- NextByteExpected: last in-sequence byte expected by receiver
- LastByteRead: last byte read by the receiving process

TCP Flow Control



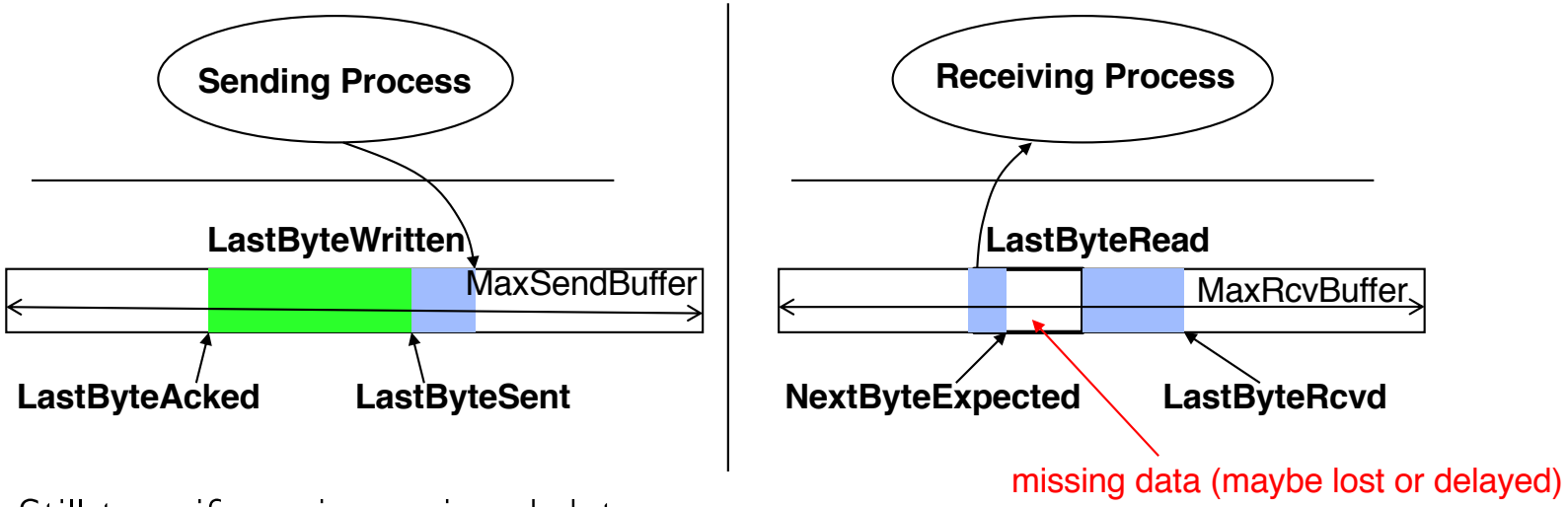
- AdvertisedWindow: number of bytes TCP receiver can receive

  AdvertisedWindow = MaxRcvBuffer – (LastByteRcvd – LastByteRead)

- SenderWindow: number of bytes TCP sender can send

  SenderWindow = AdvertisedWindow – (LastByteSent – LastByteAcked)

TCP Flow Control



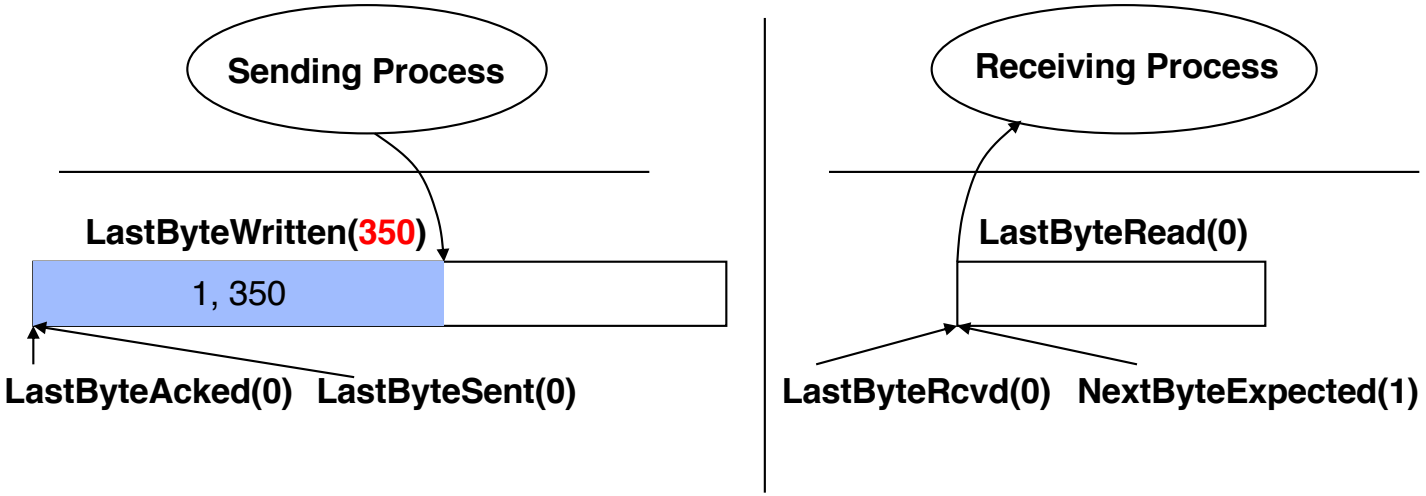
- Still true if receiver missed data….

AdvertisedWindow = MaxRcvBuffer – (LastByteRcvd – LastByteRead)

- WriteWindow: number of bytes sending process can write

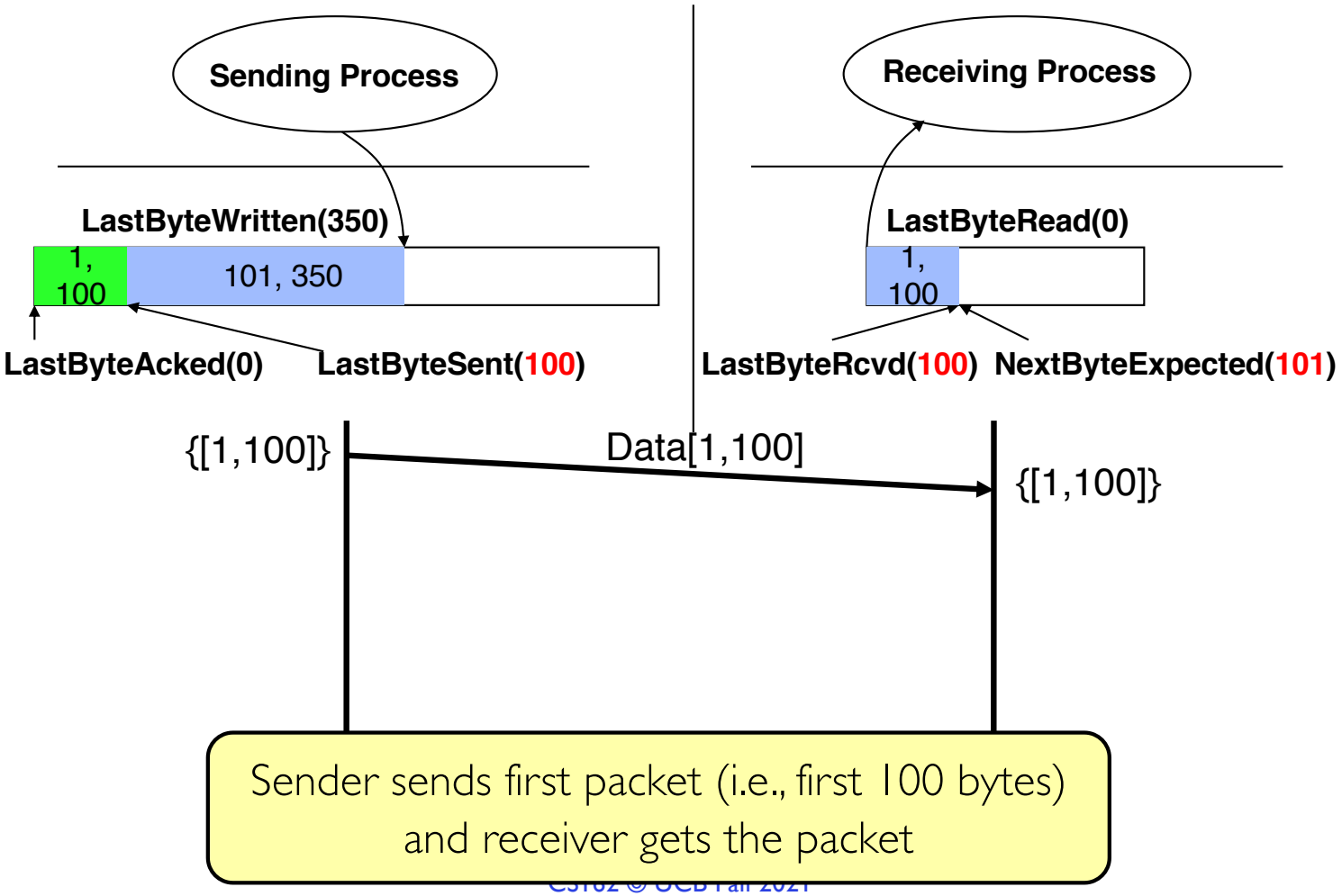
WriteWindow = MaxSendBuffer – (LastByteWritten – LastByteAcked)

TCP Flow Control

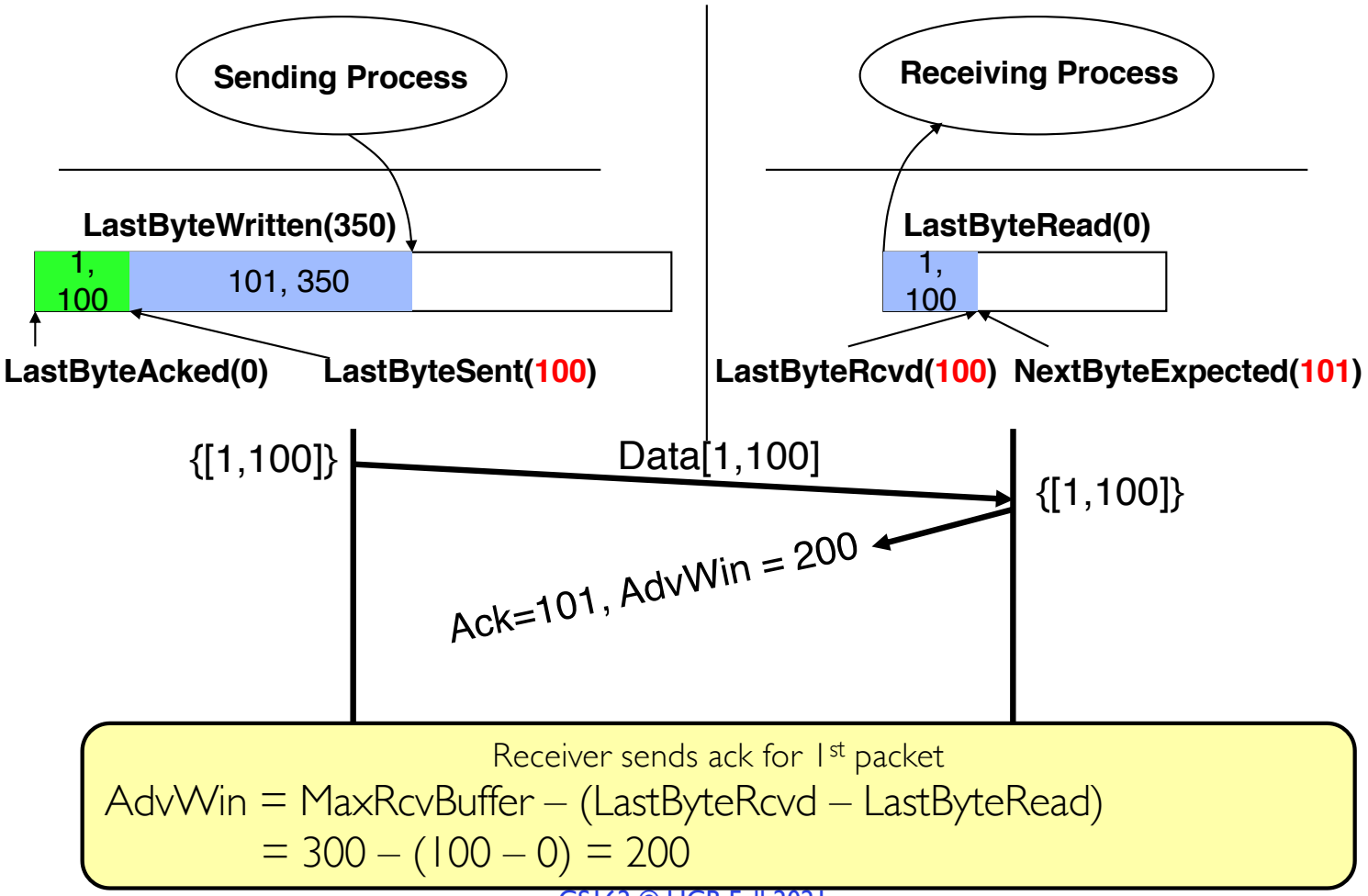


- Sending app sends 350 bytes
- Recall:
  - We assume IP only accepts packets no larger than 100 bytes
  - MaxRcvBuf = 300 bytes, so initial Advertised Window = 300 byets

TCP Flow Control



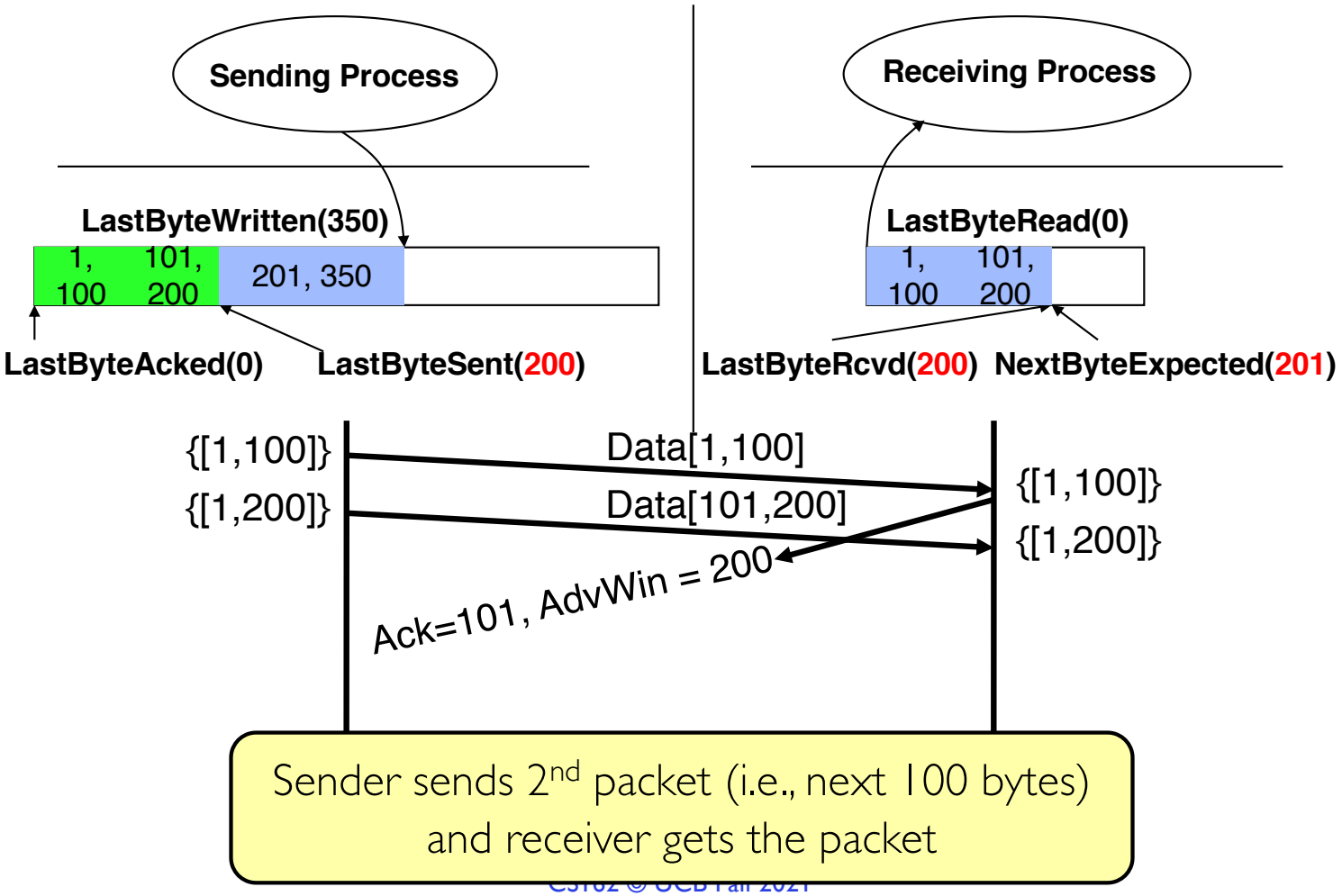
TCP Flow Control



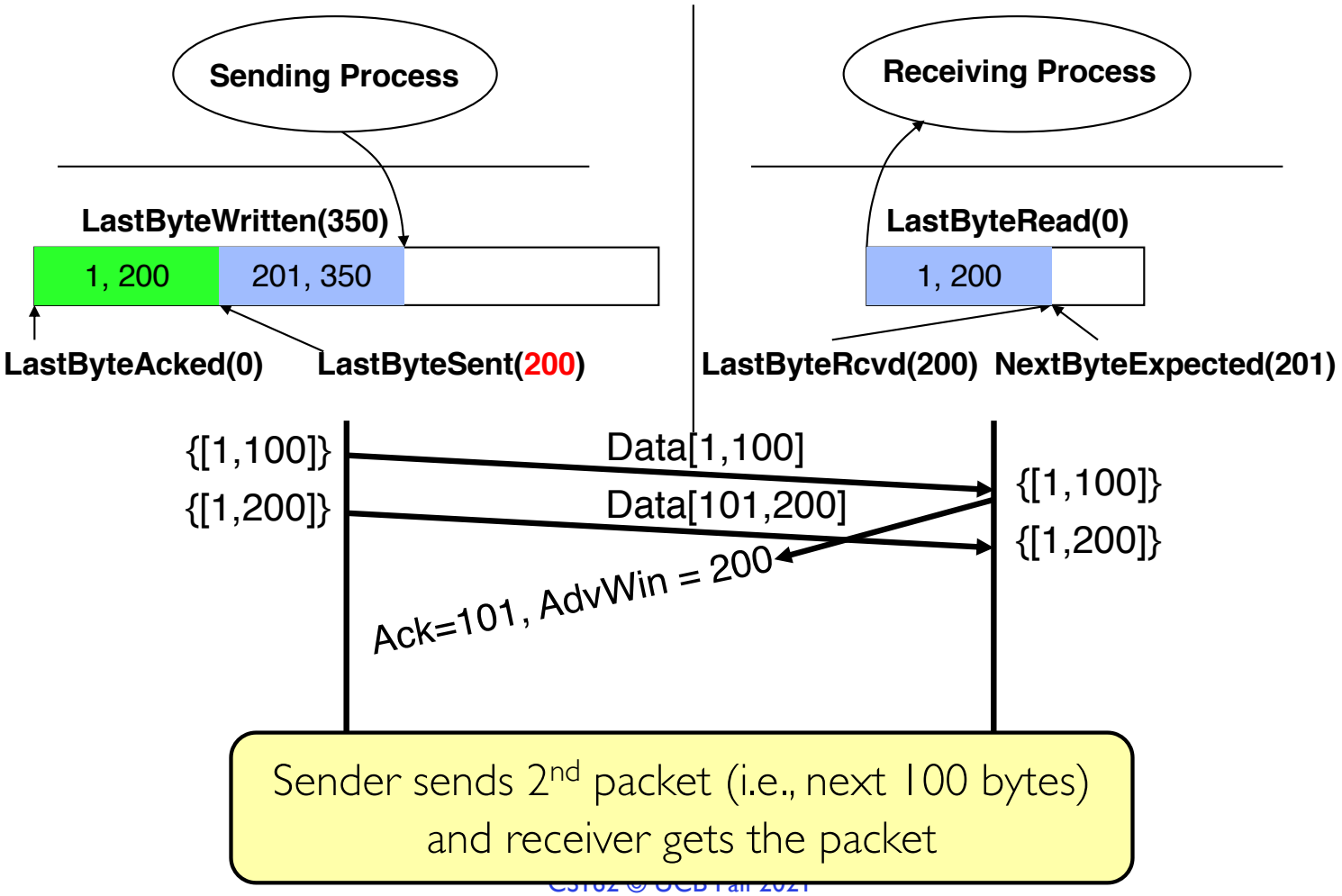
Receiver sends ack for 1st packet

AdvWin = MaxRcvBuffer – (LastByteRcvd – LastByteRead)
= 300 – (100 – 0) = 200

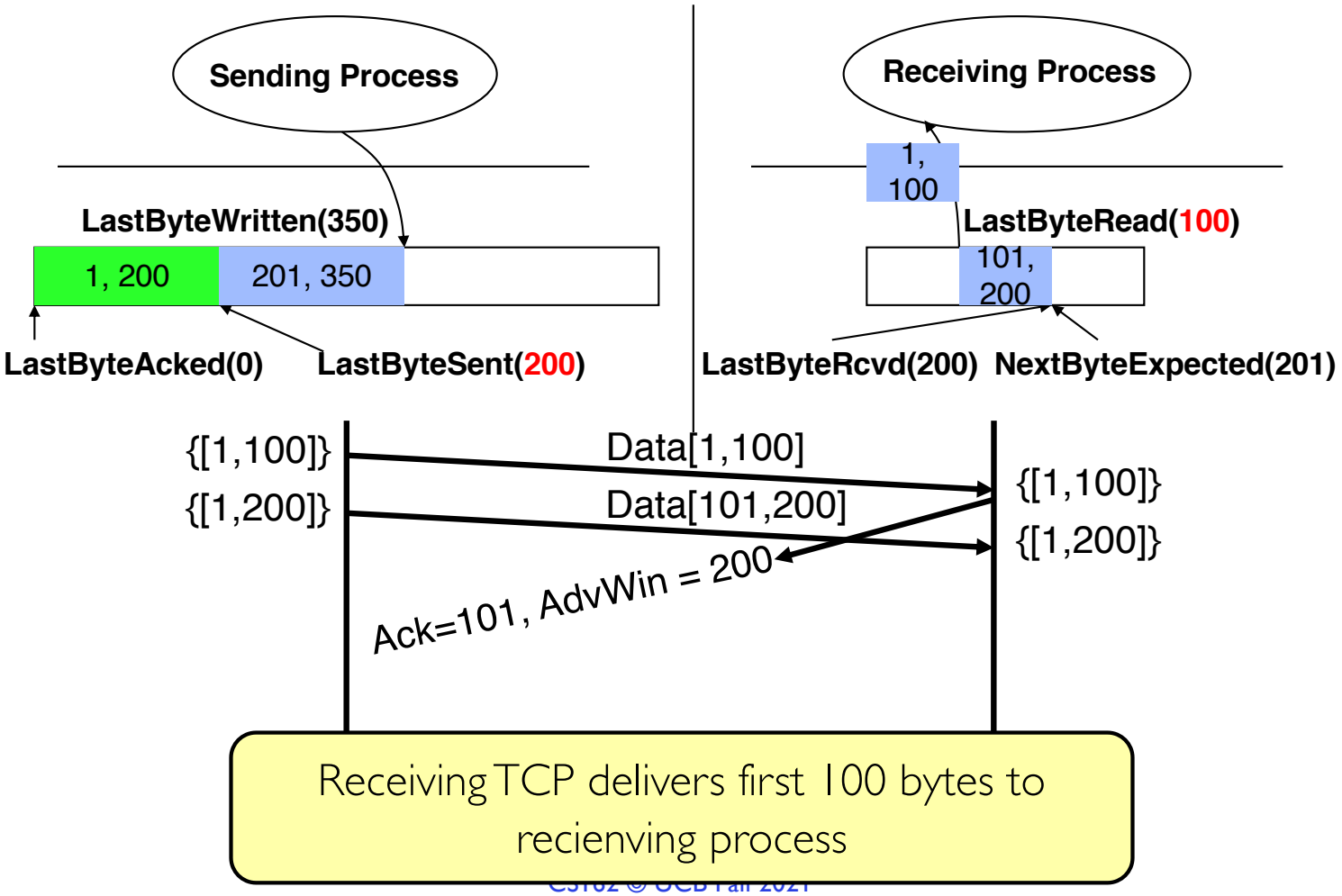
TCP Flow Control



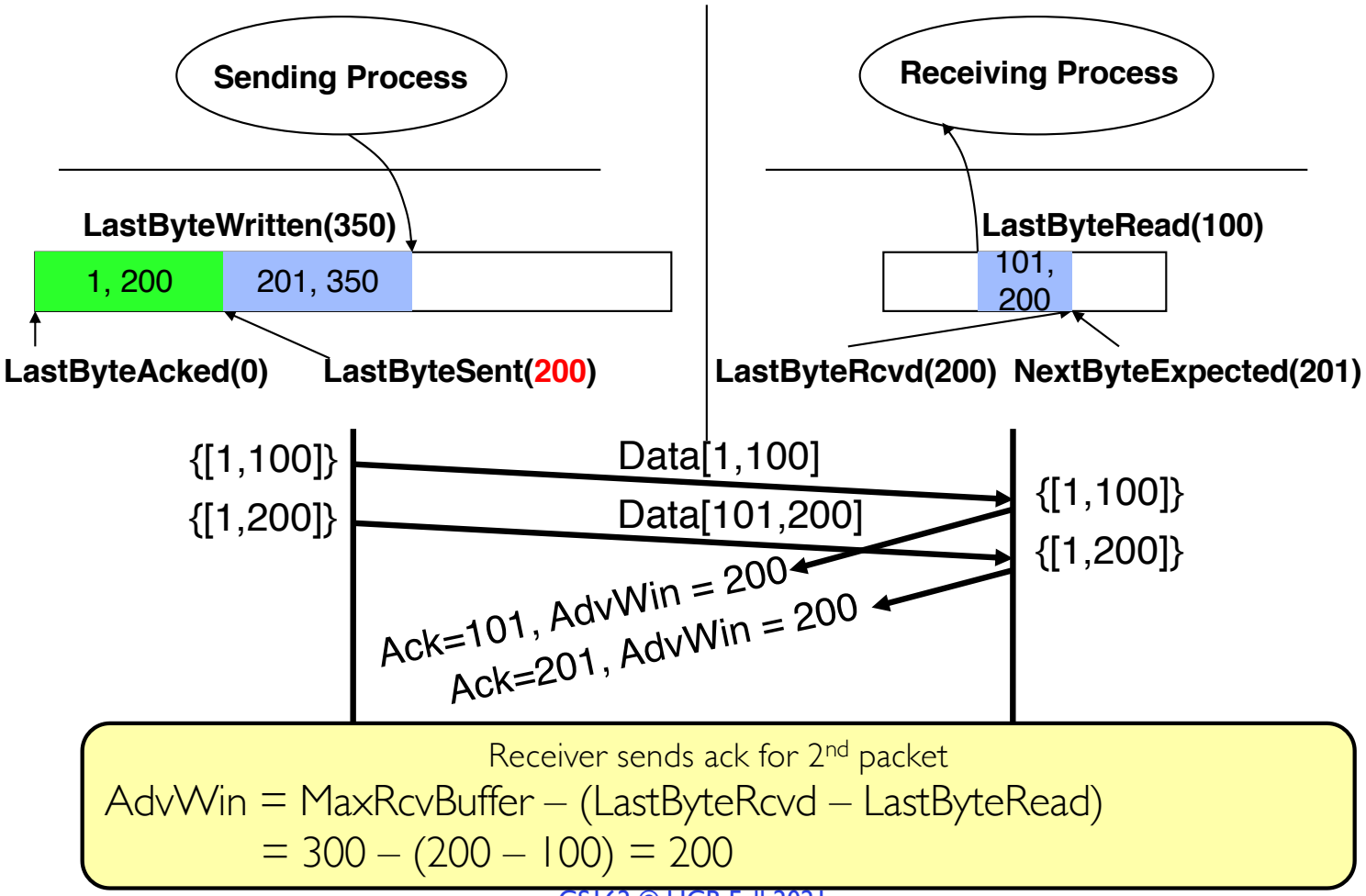
TCP Flow Control



TCP Flow Control



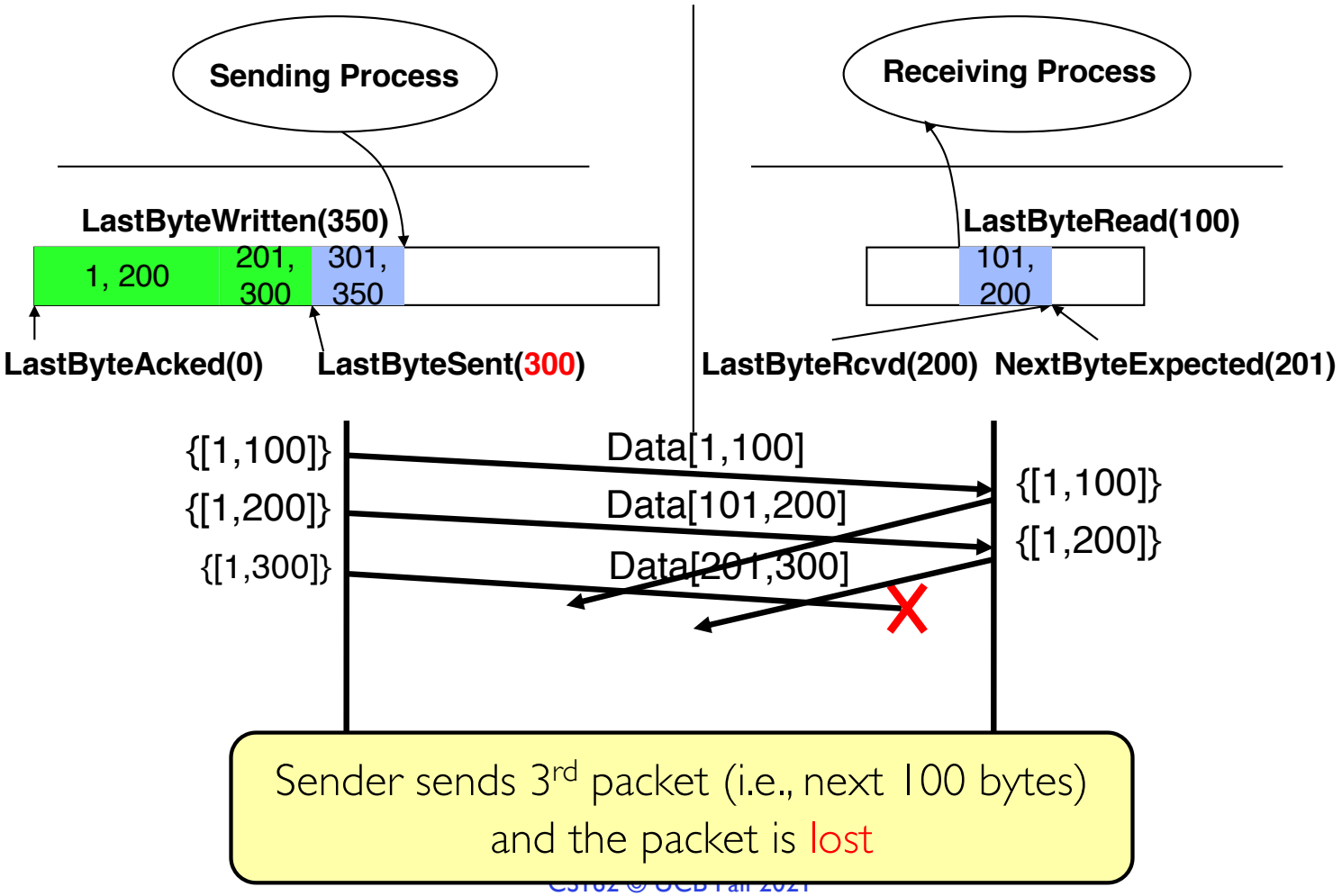
TCP Flow Control



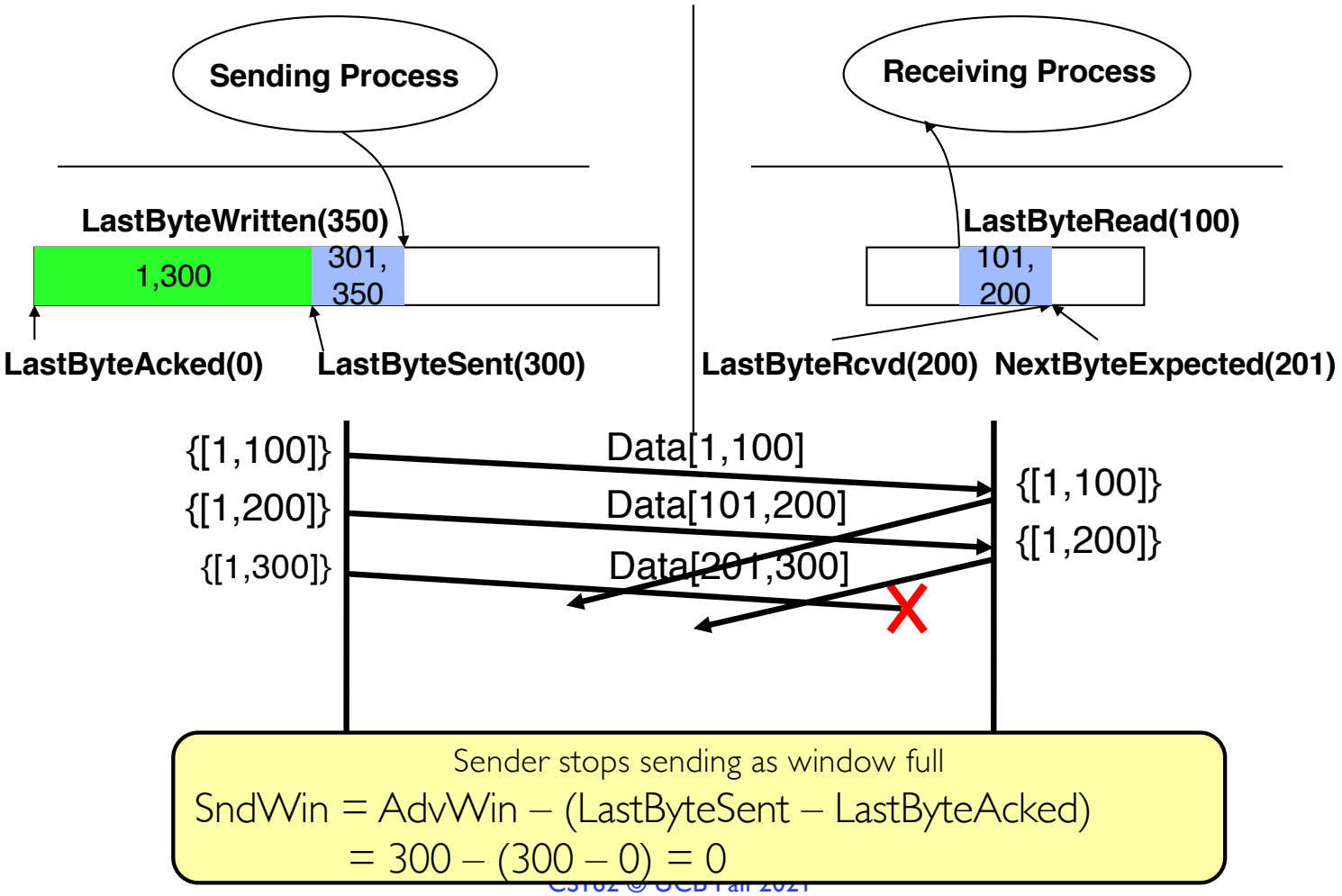
Receiver sends ack for 2nd packet

AdvWin = MaxRcvBuffer – (LastByteRcvd – LastByteRead)
= 300 – (200 – 100) = 200

TCP Flow Control



TCP Flow Control

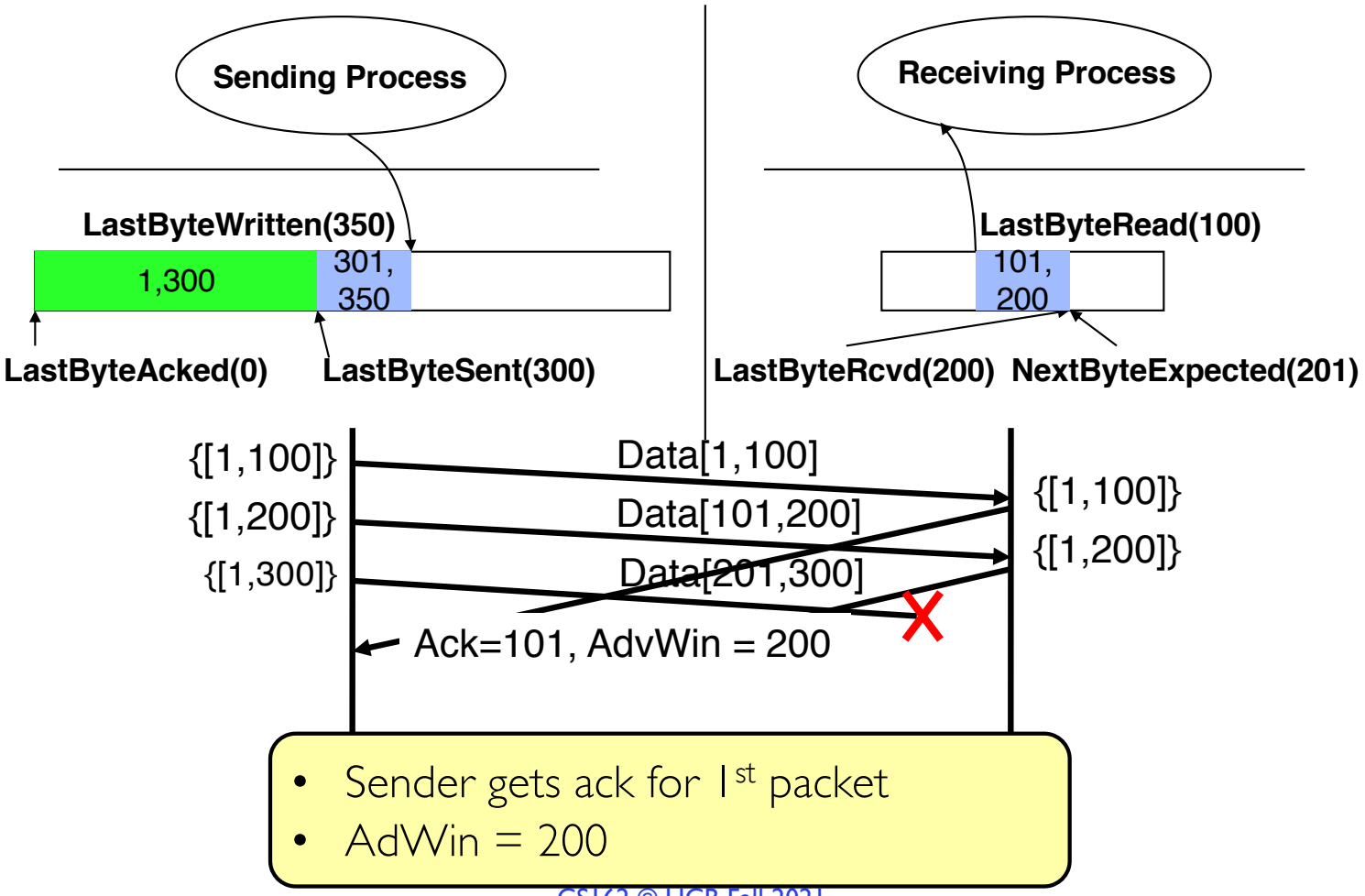


Sender stops sending as window full

SndWin = AdvWin – (LastByteSent – LastByteAcked)

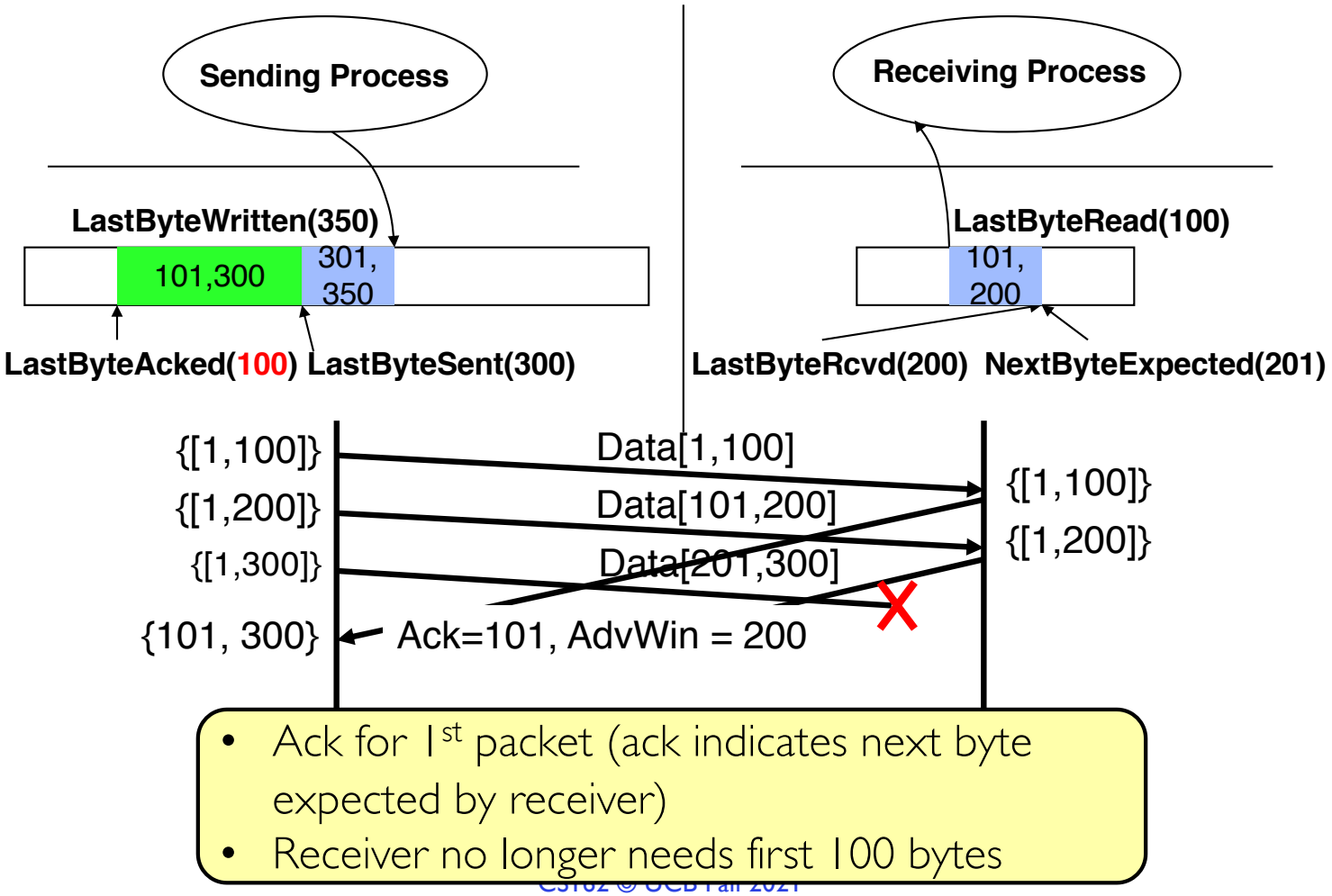
= 300 – (300 – 0) = 0

TCP Flow Control



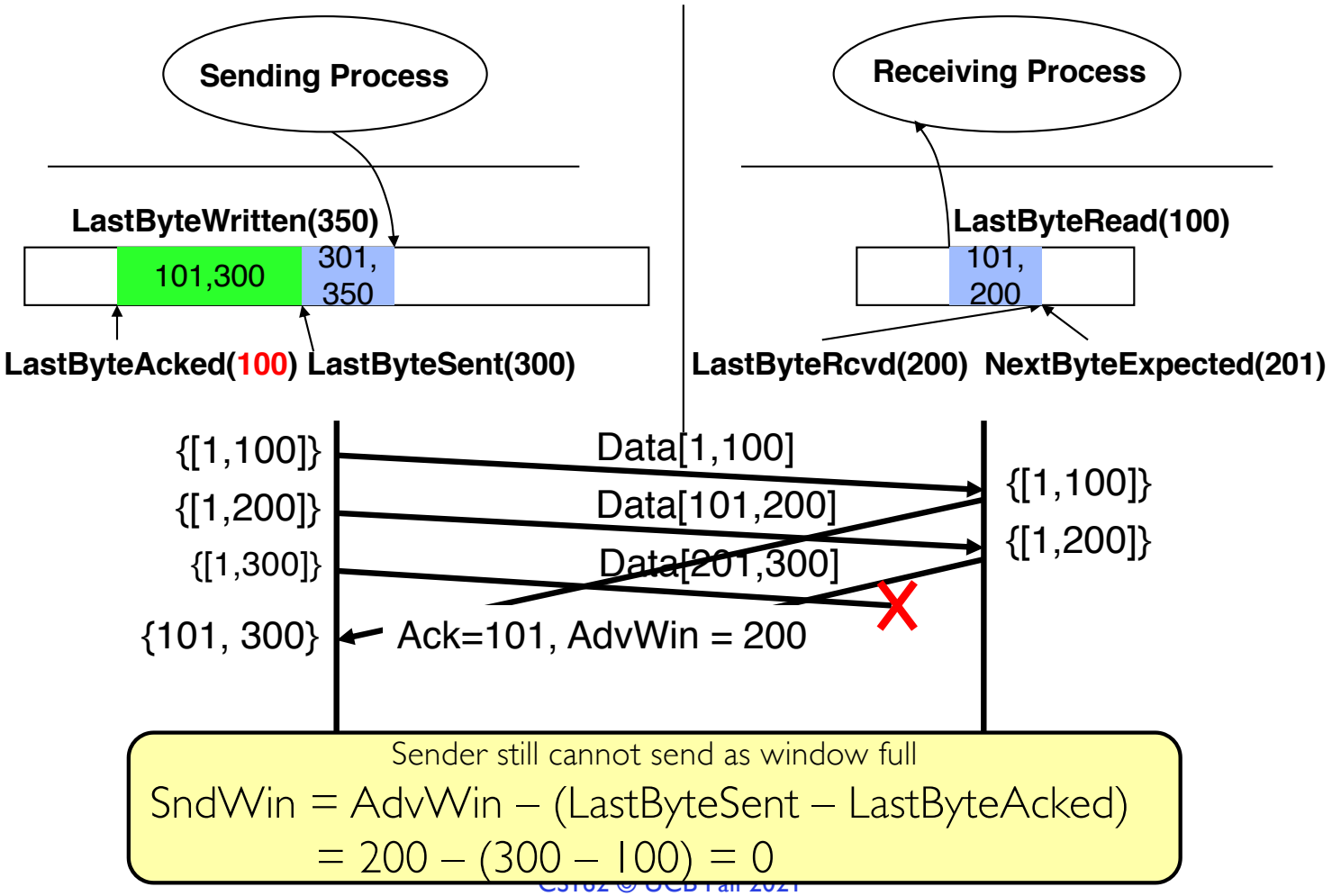
- Sender gets ack for 1st packet
- AdWin = 200

TCP Flow Control

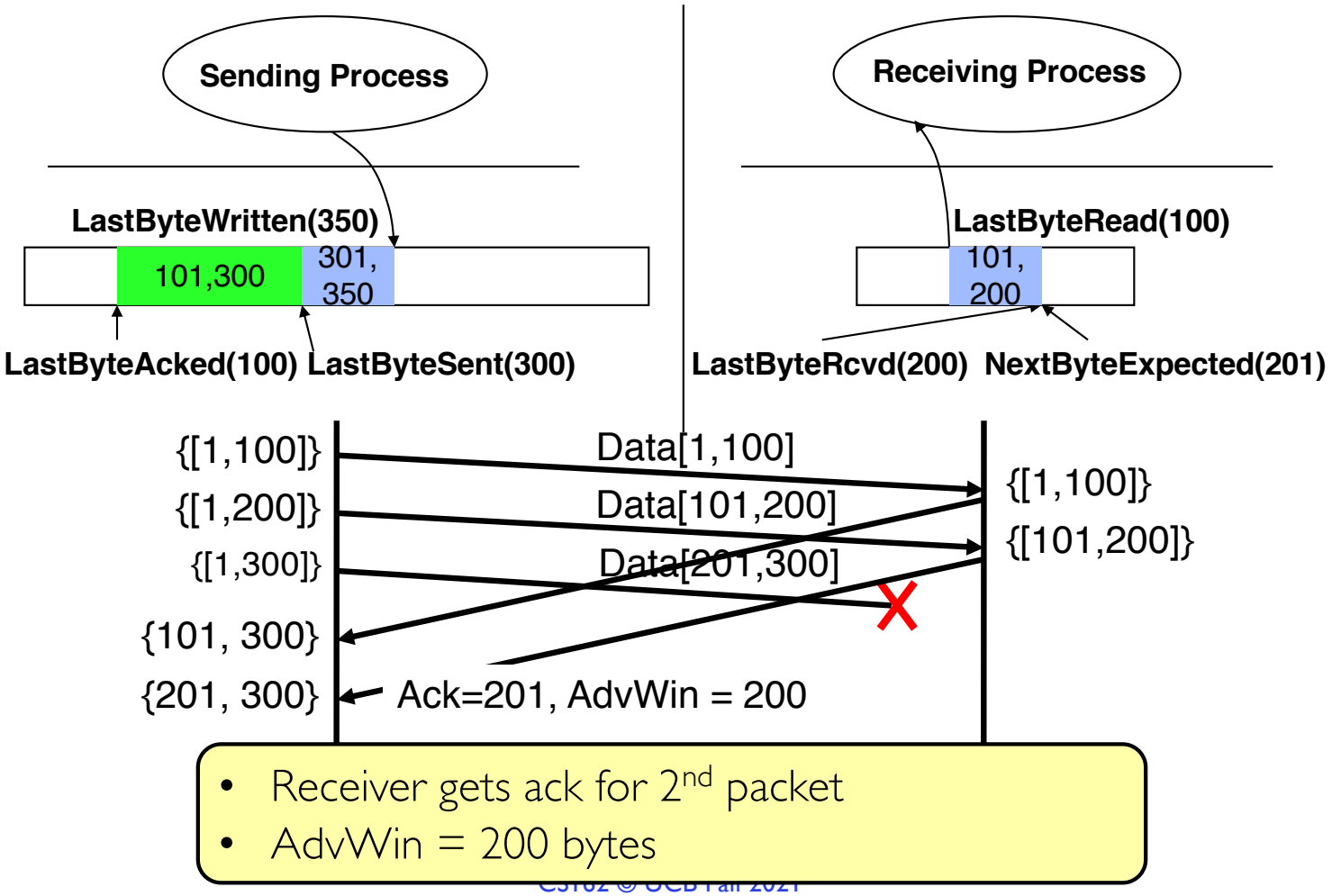


- Ack for 1st packet (ack indicates next byte expected by receiver)
- Receiver no longer needs first 100 bytes

TCP Flow Control

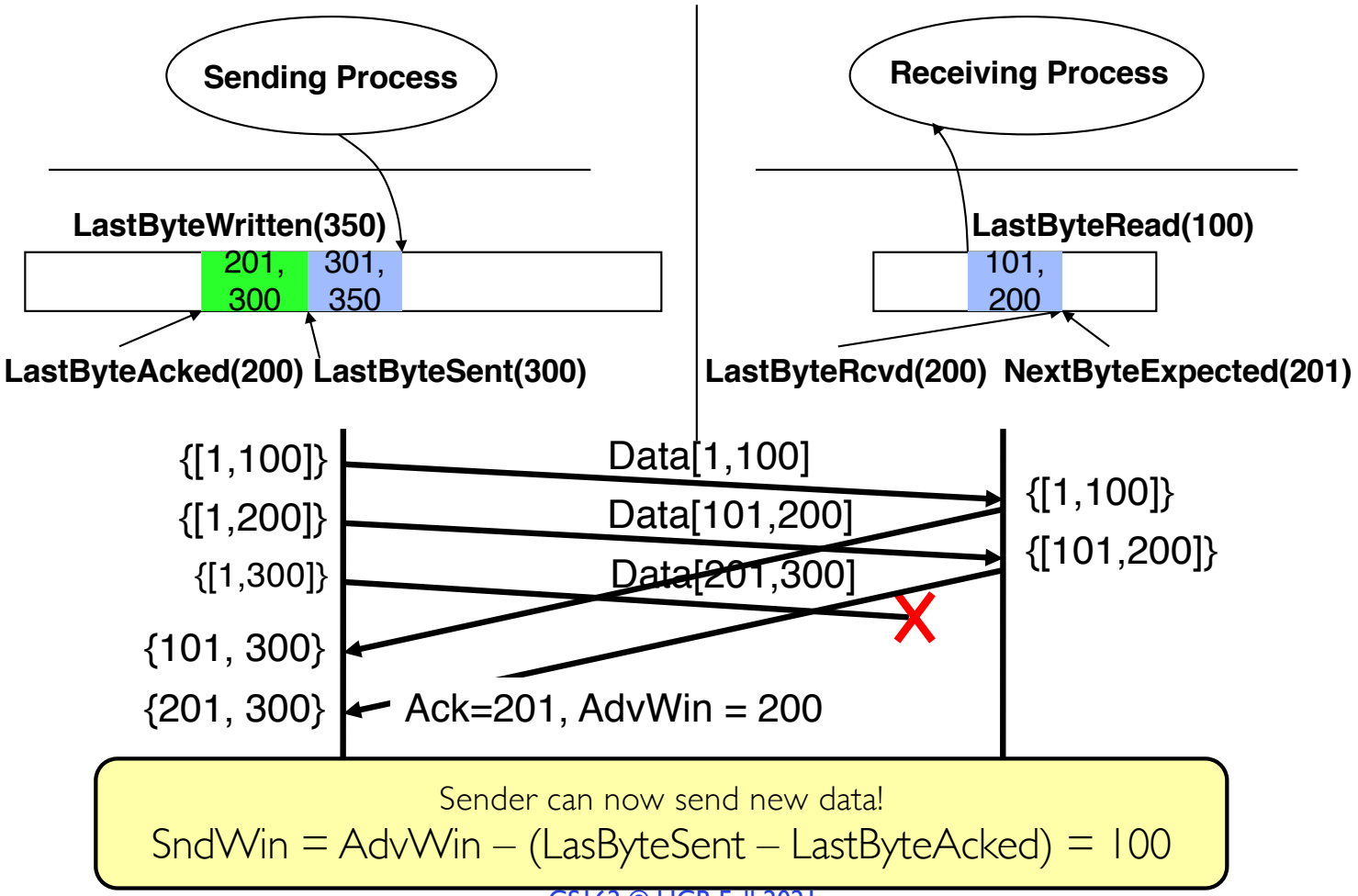


TCP Flow Control

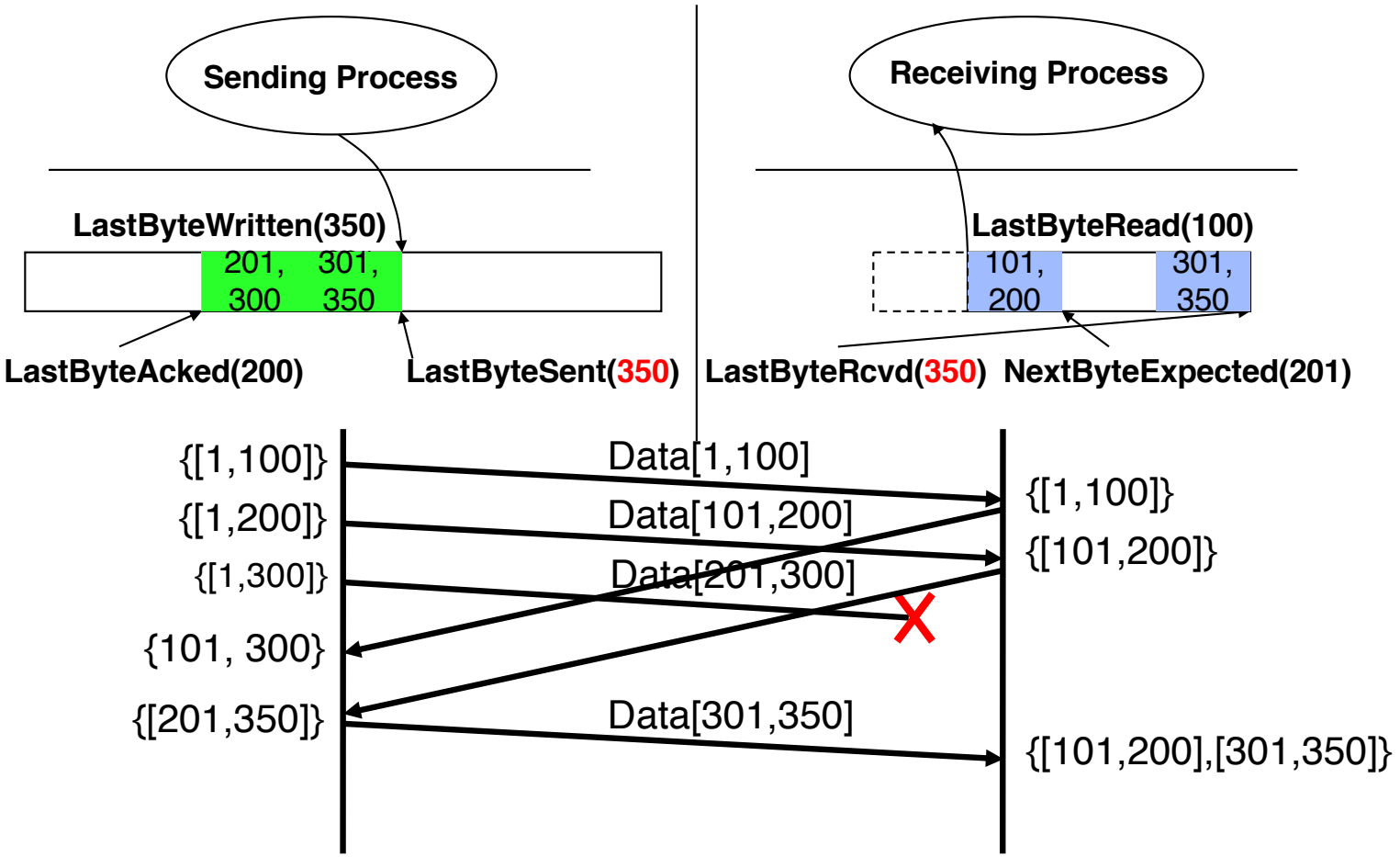


- Receiver gets ack for 2nd packet
- AdvWin = 200 bytes

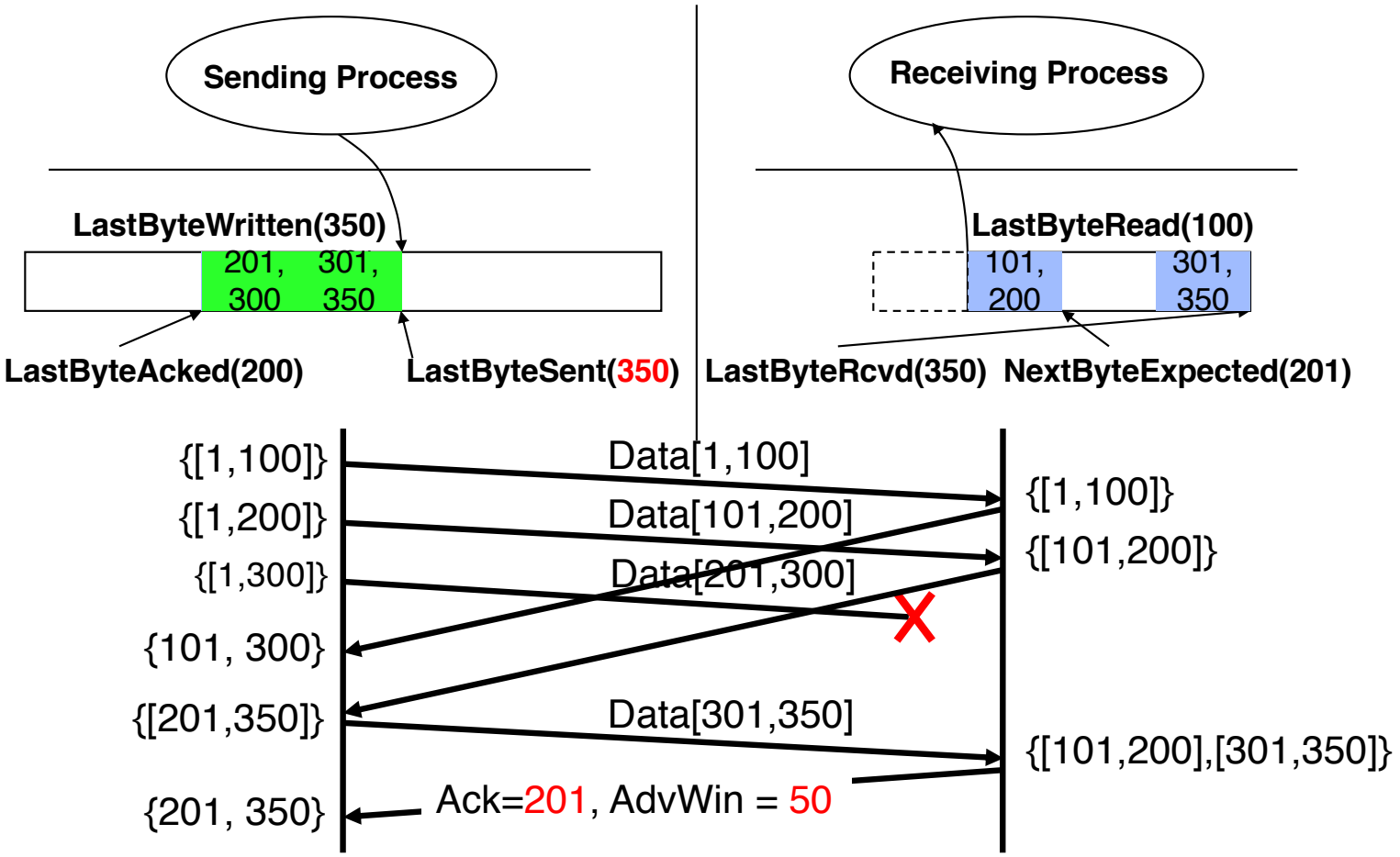
TCP Flow Control



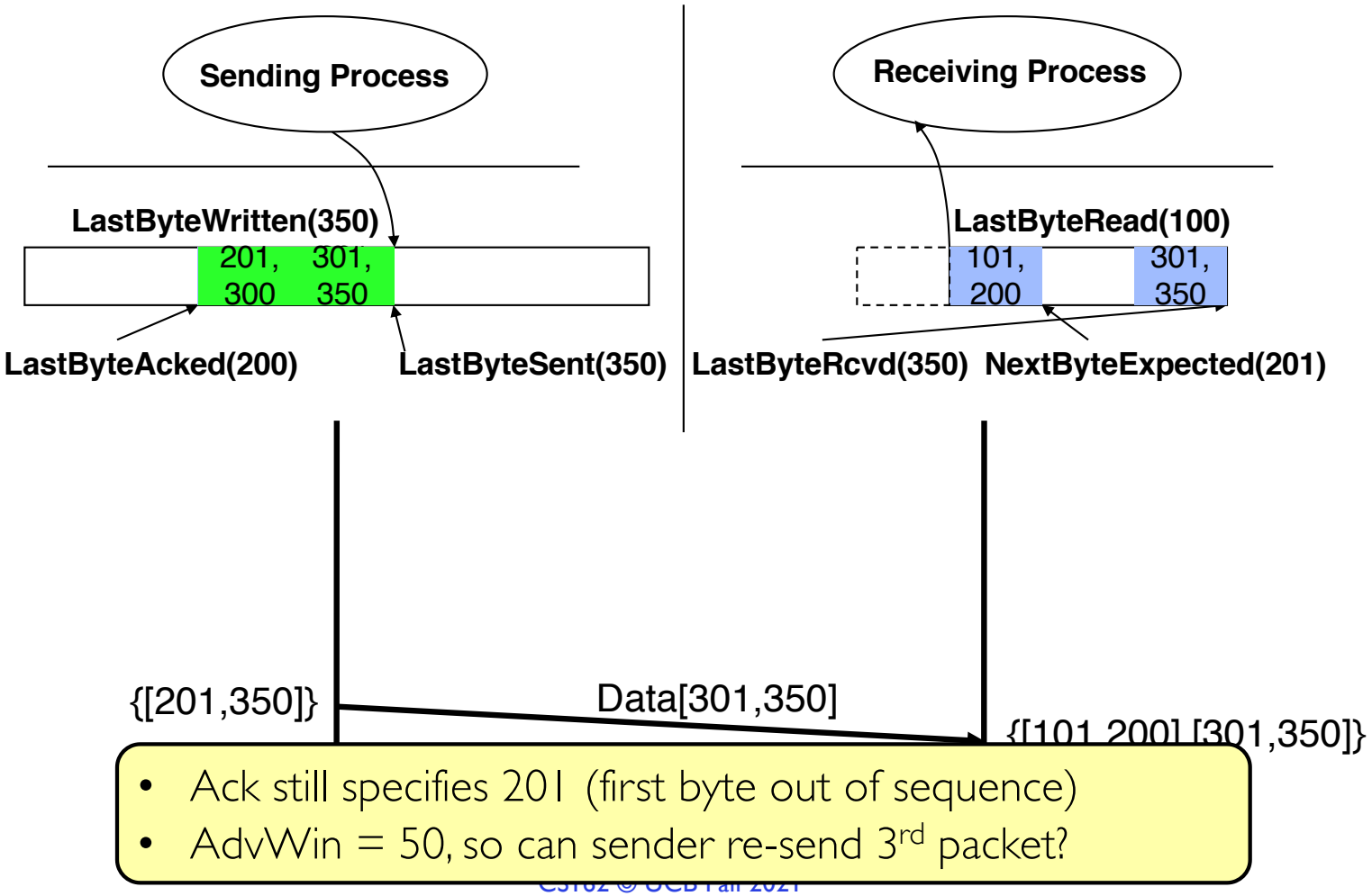
TCP Flow Control



TCP Flow Control

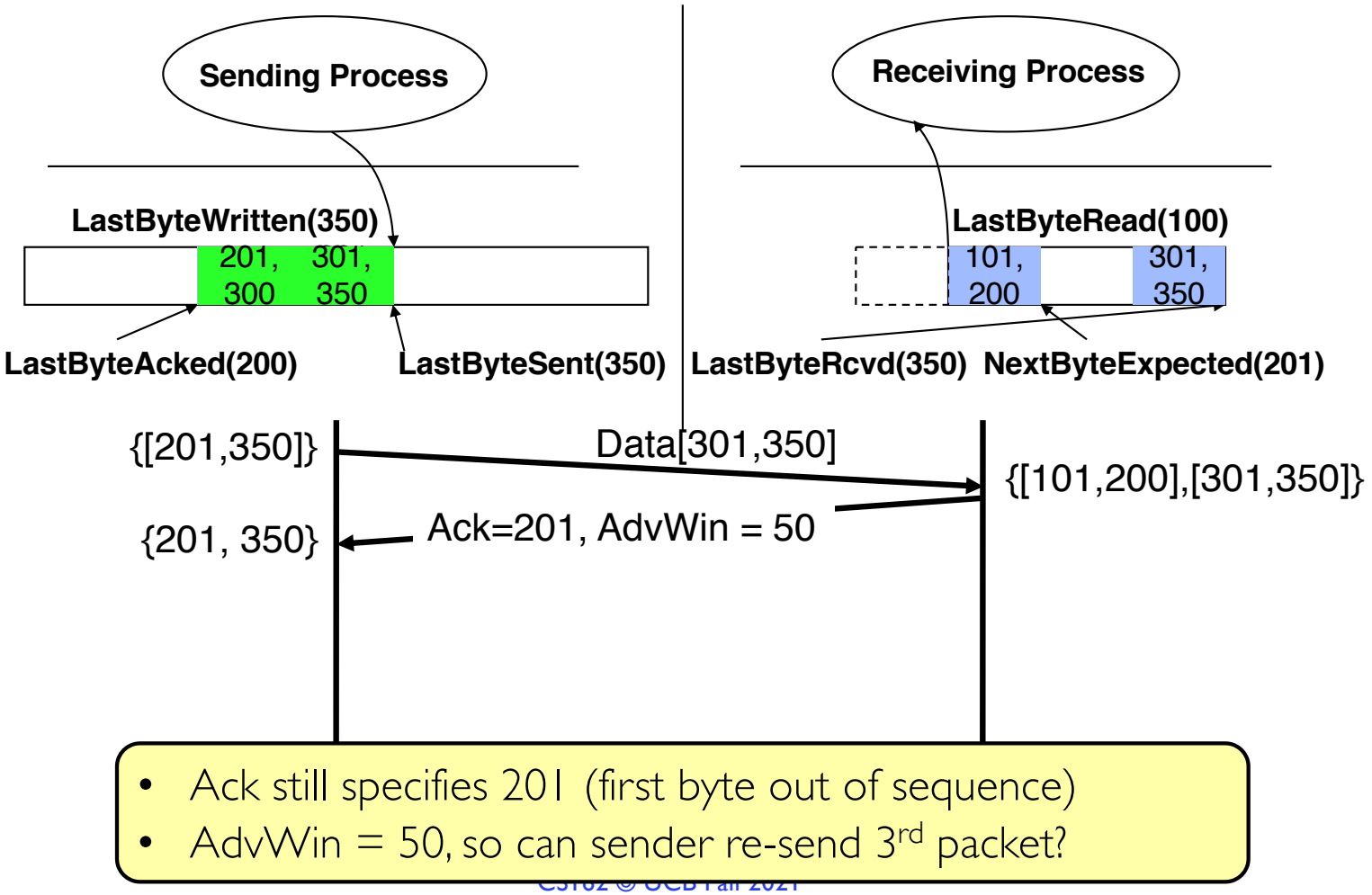


TCP Flow Control



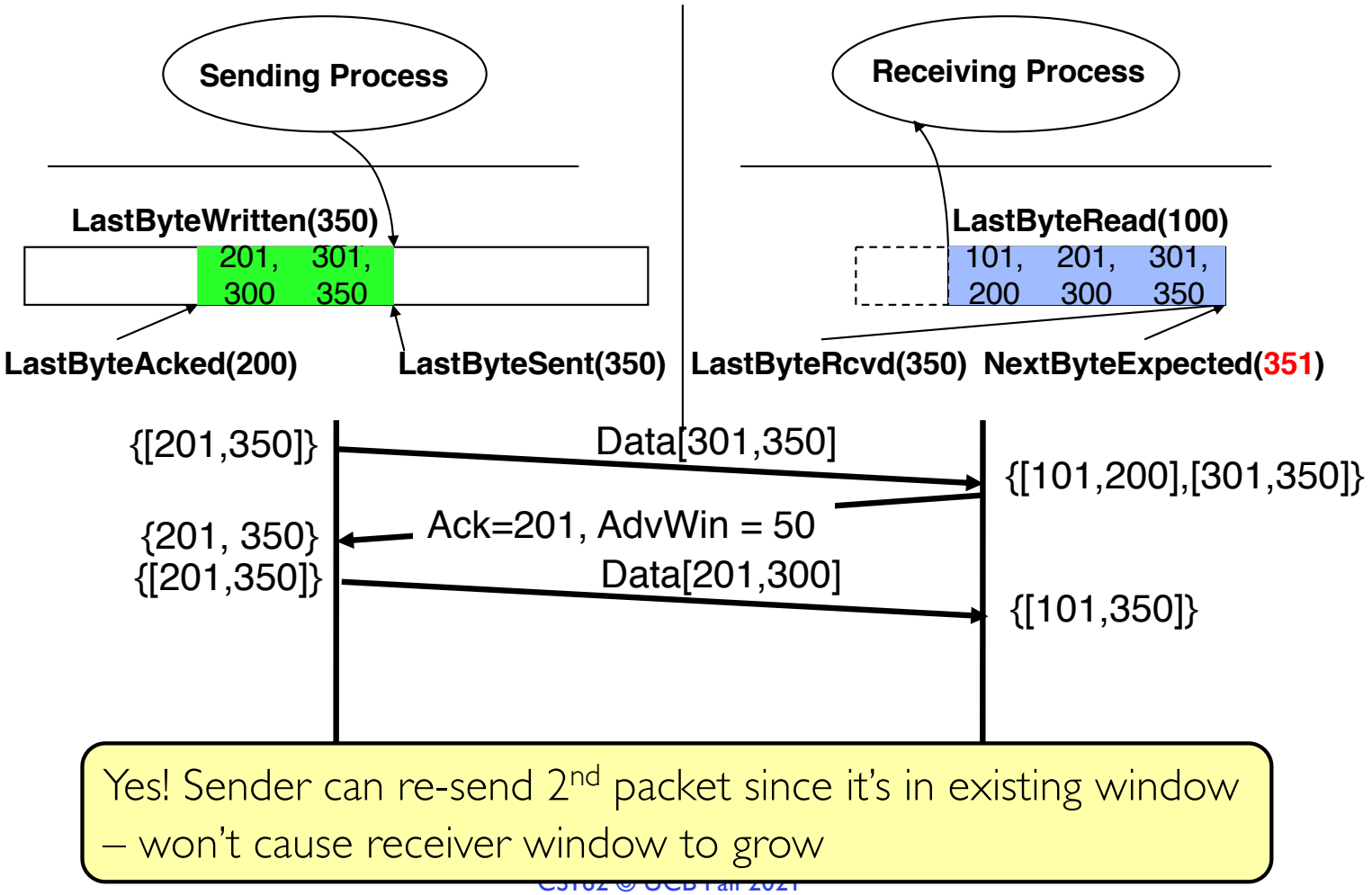
- Ack still specifies 201 (first byte out of sequence)
- AdvWin = 50, so can sender re-send 3rd packet?

TCP Flow Control



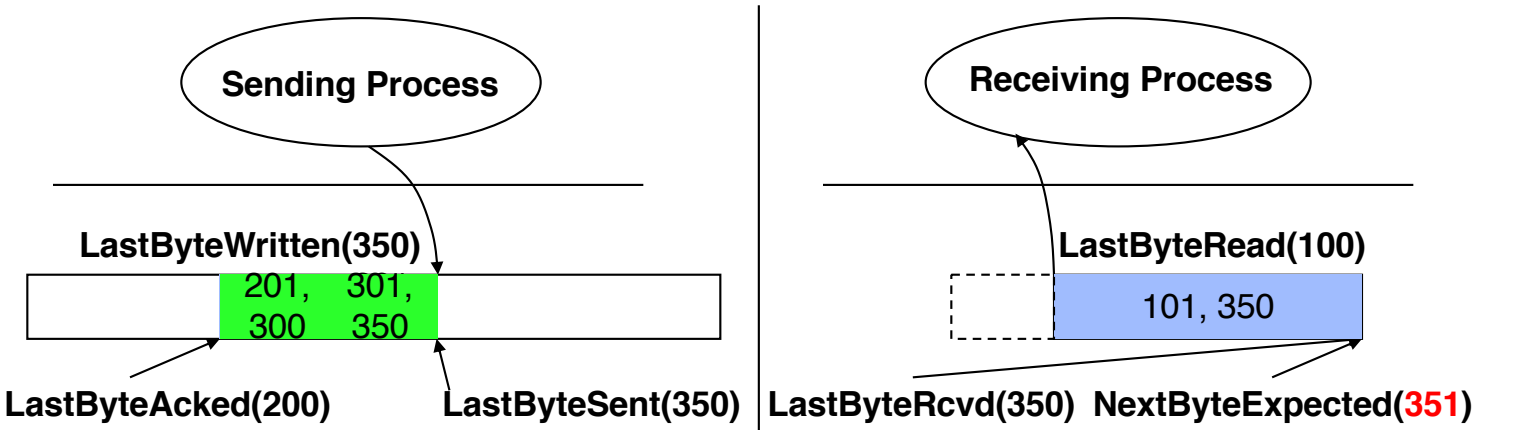
- Ack still specifies 201 (first byte out of sequence)
- AdvWin = 50, so can sender re-send 3rd packet?

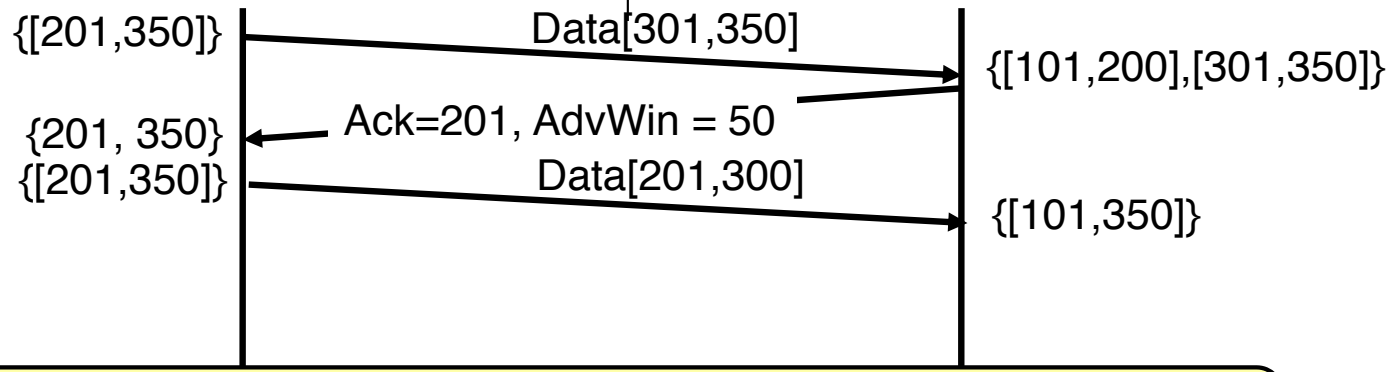
TCP Flow Control



Yes! Sender can re-send 2nd packet since it's in existing window – won't cause receiver window to grow

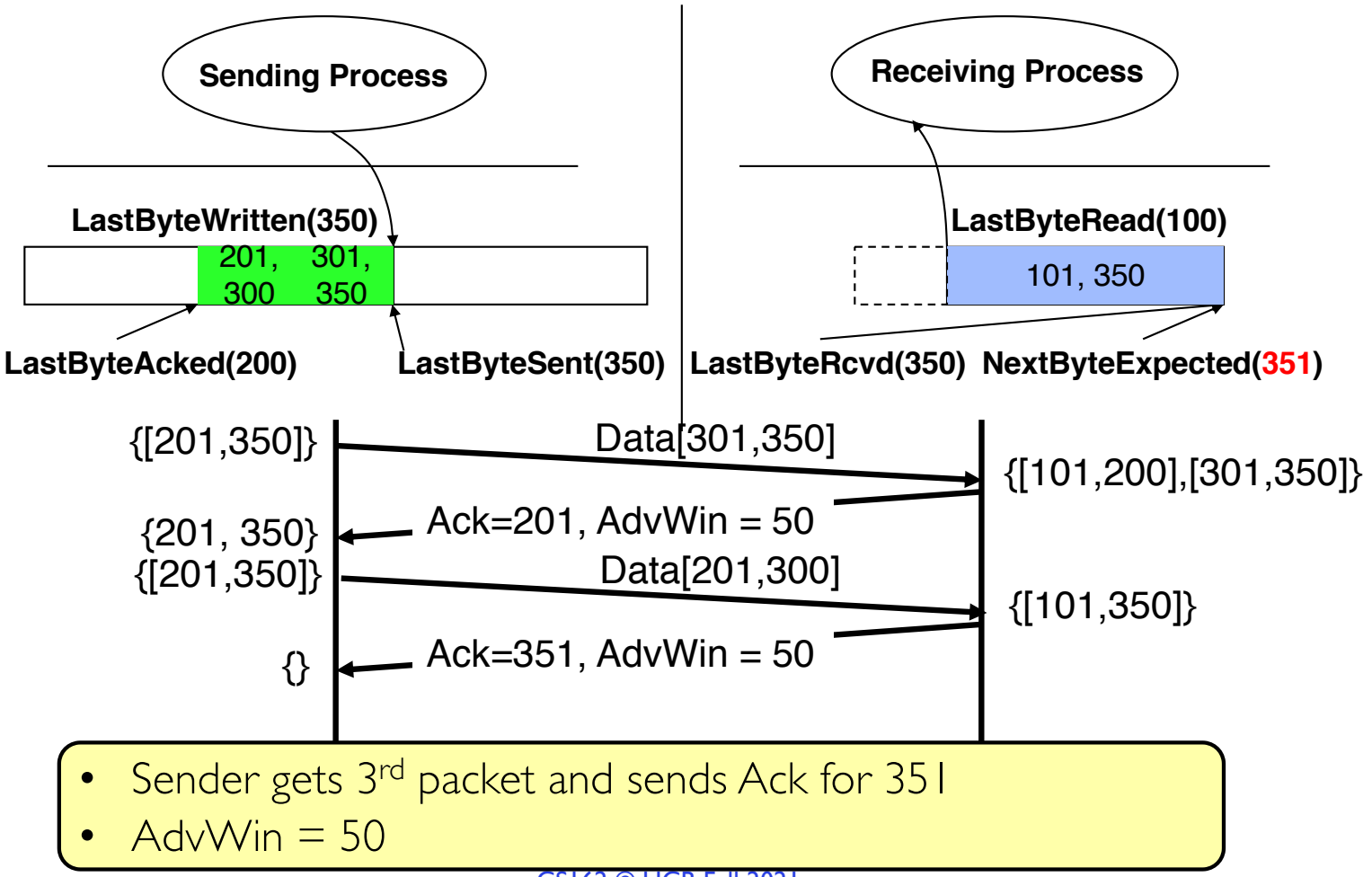
TCP Flow Control





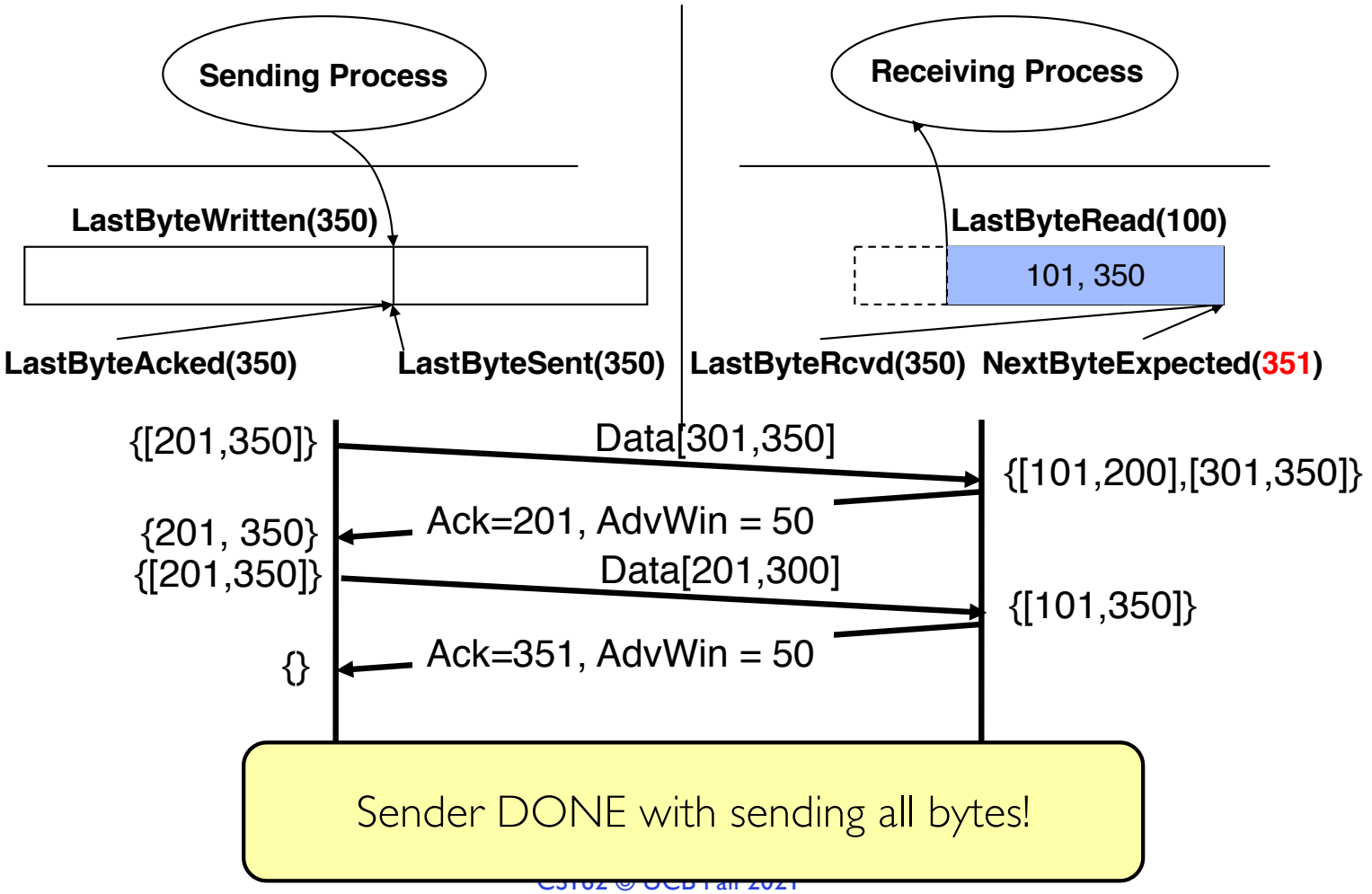
Yes! Sender can re-send 2nd packet since it's in existing window – won't cause receiver window to grow

TCP Flow Control



- Sender gets 3rd packet and sends Ack for 351
- AdvWin = 50

TCP Flow Control



Discussion

- Why not have a huge buffer at the receiver (memory is cheap!)?
- Sending window (SndWnd) also depends on network congestion
  - **Congestion control**: ensure that a fast receiver doesn't overwhelm a router in the network (see next; discussed in detail in cs168)
- In practice there is another set of buffers in the protocol stack, at the **link layer** (i.e., Network Interface Card)

Announcements

Midterm 3 on Wed 12/1 from 7-9 PM

Project 3 Party on Sunday 12/5 from 10AM-2PM at Woz Lounge

Discussions converted to office hours this week

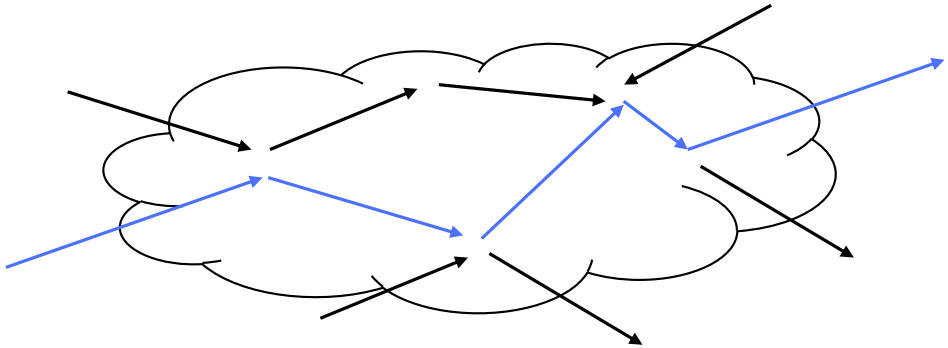
Office hours will continue into dead week (with possibly modified schedule)

Homework 6 due Friday 12/3 11:59 PM

Project 3 due Wednesday 12/8 11:59 PM

Congestion

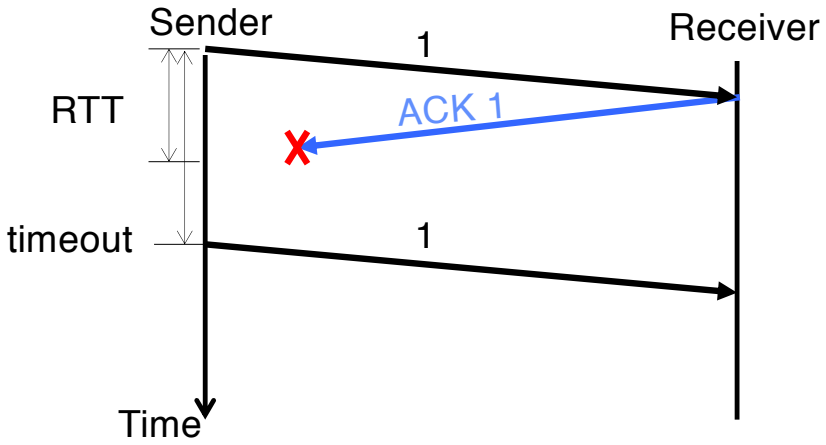
- Too much data trying to flow through some part of the network



- How do you detect congestion?
  - **Dropped packets**: routers drop packets when their buffers overflow (most used)
  - **Early Congestion Notification (ECN)**: Set a bit in the packet if the buffer is about to overflow, e.g., buffer occupancy exceed some threshold
  - **Delayed packets**: when the router buffers are becoming large it takes longer to the packet to get to the destination

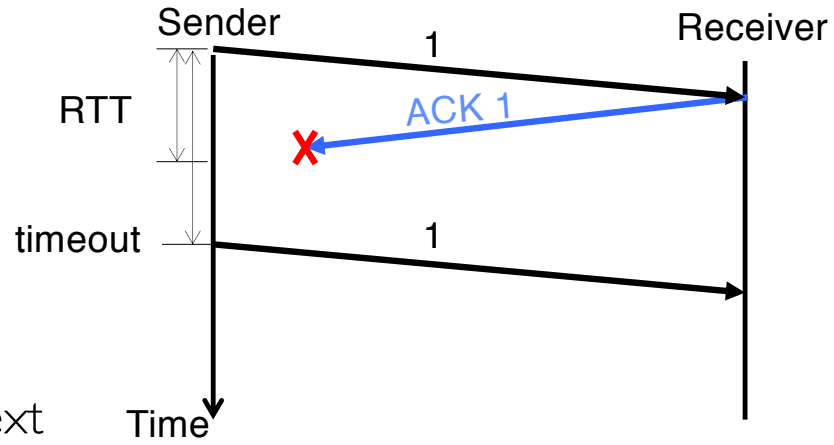
How do you detect dropped packets?

Timeout: An ack was not received for a time much longer than expected, i.e., RTT



How do you detect dropped packets?

Timeout: An ack was not received for a time much longer than expected, i.e., RTT



Duplicate acknowledgement:

- Recall receiver acks with the # of the next expected byte in sequence
- If there is a gap, the receiver will always sent back the same ack # (201 in our example)
- **3 Duplicate Acks** == Loss

Congestion Control

Key idea: sender

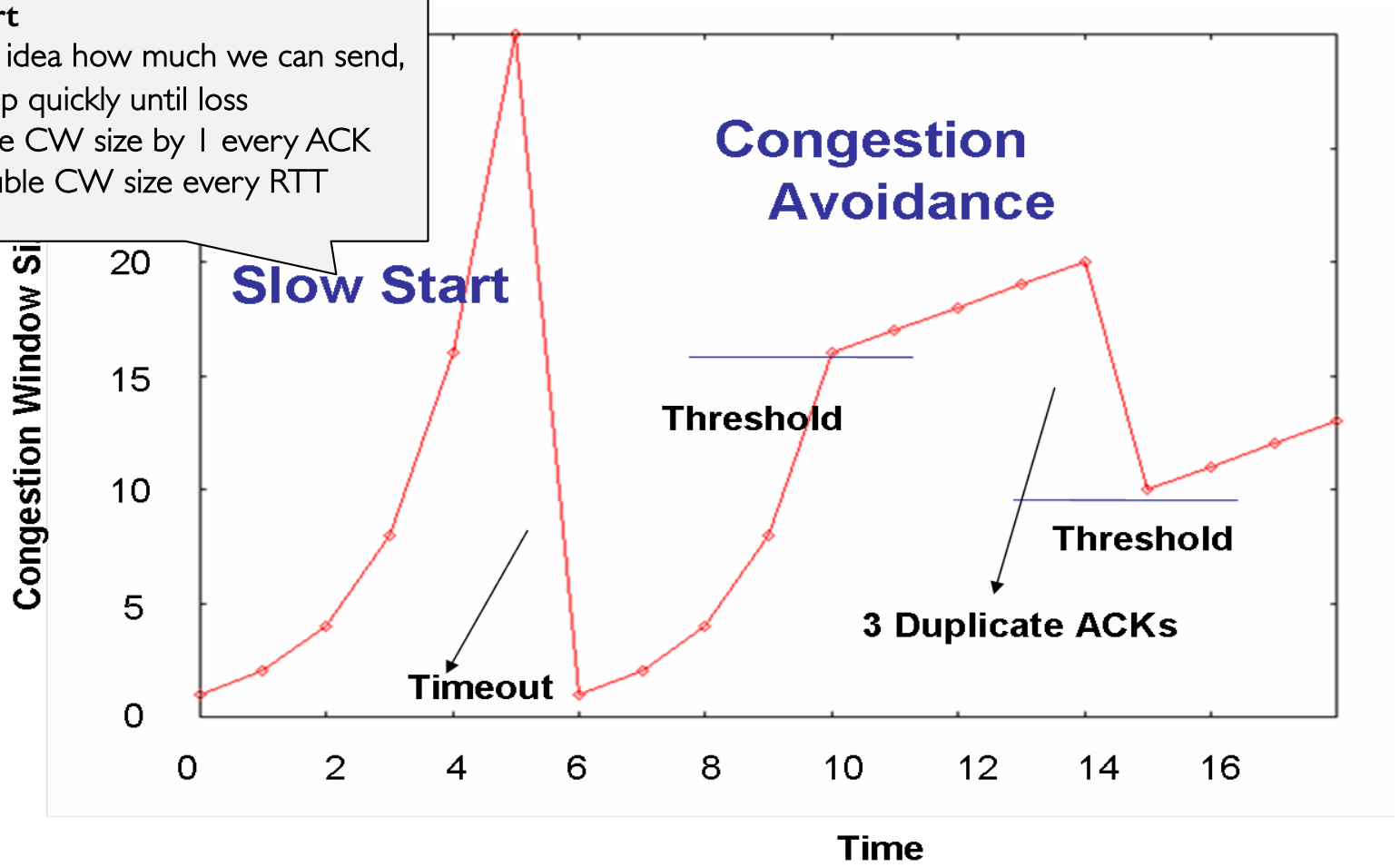
- Increase sending rate until congestion
- Reduce sending rate if the network is congested

- How does the sender control sending rate? Sender (congestion) window size!
  - Must be less than receiver's advertised buffer size
    - » Increase size of window until congestion
    - » Reduce size of the window if congestion

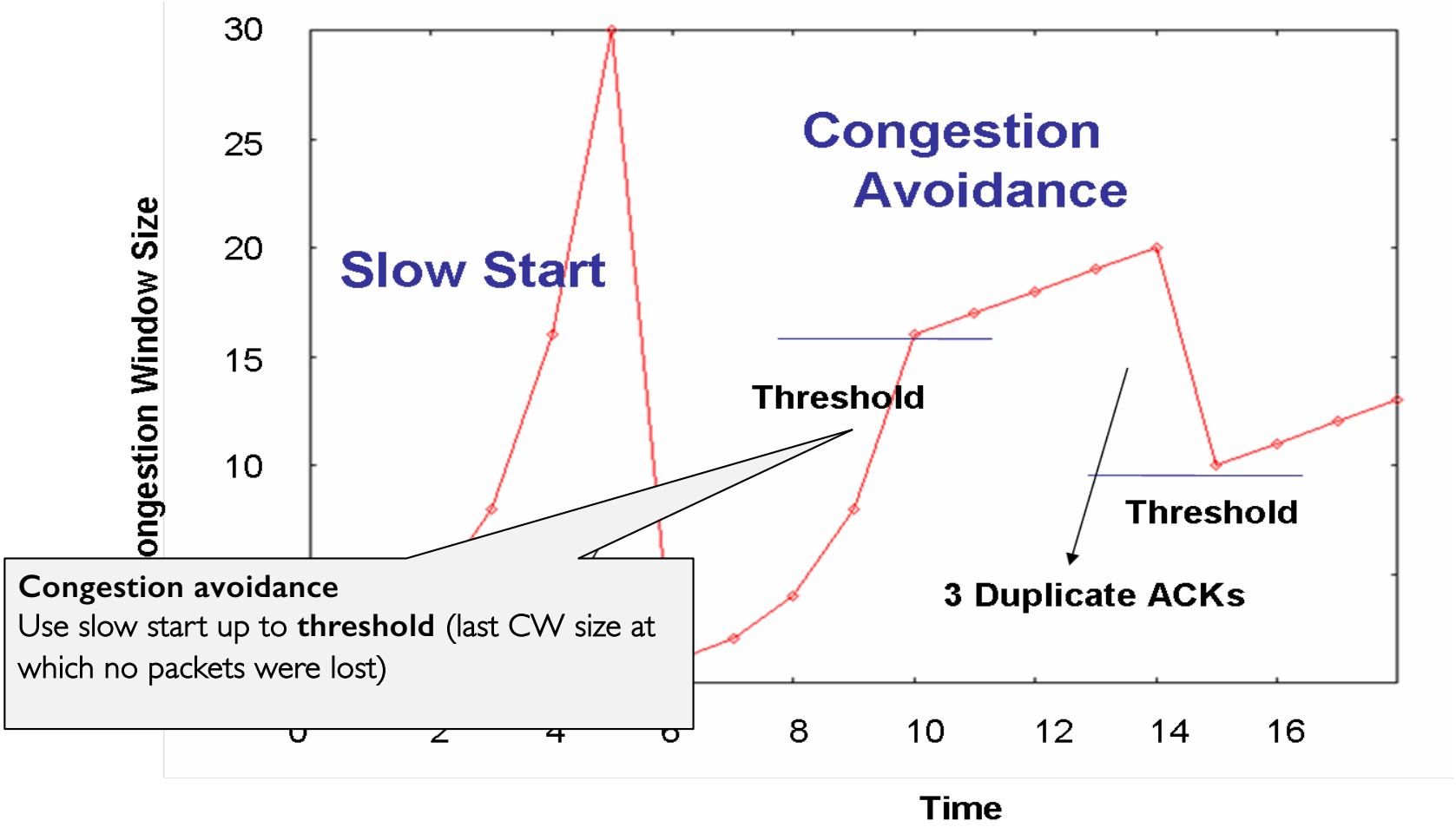
Congestion Control at a glance

Slow Start
Initially, no idea how much we can send, so ramp-up quickly until loss

- Increase CW size by 1 every ACK
  ➔ double CW size every RTT

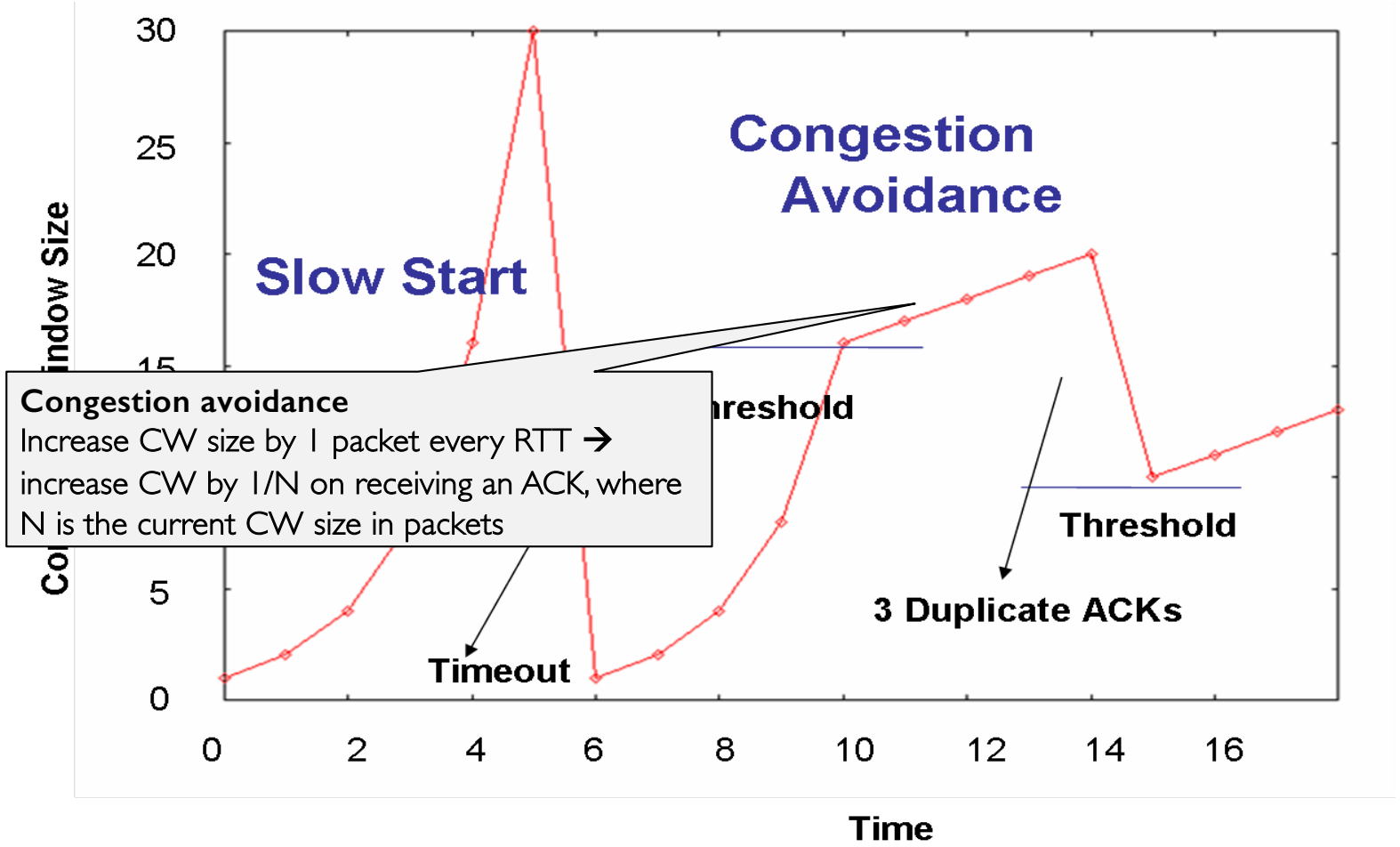


Congestion Control at a glance

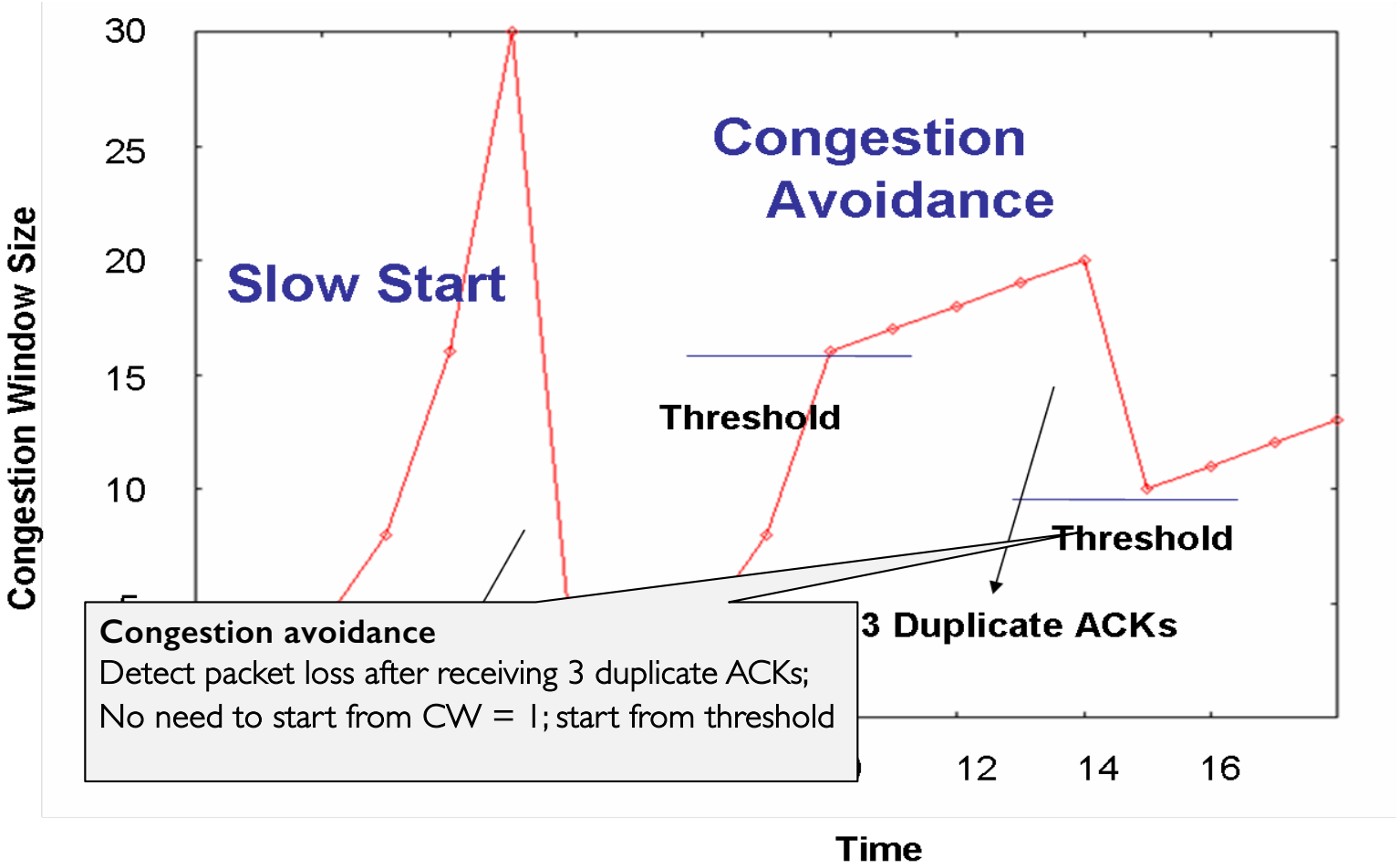


Congestion avoidance
Use slow start up to **threshold** (last CW size at which no packets were lost)

Congestion Control at a glance



Congestion Control at a glance



Congestion avoidance
Detect packet loss after receiving 3 duplicate ACKs;
No need to start from CW = 1; start from threshold

Networking Summary (1/2)

- Layered architecture powerful abstraction for organizing complex networks
- Internet: 5 layers
  - Physical: send bits
  - Datalink: Connect two hosts on same physical media
  - Network: Connect two hosts in a wide area network
  - Transport: Connect two processes on (remote) hosts
  - Applications: Enable applications running on remote hosts to interact
- Unified Internet layering (Application/Transport/ Internetwork/Link/Physical) decouples apps from networking technologies

Networking Summary (2/2)

- E2E argument encourages us to keep IP simple
- If higher layer can implement functionality correctly, implement it in a lower layer only if
  - it improves the performance significantly for application that need that functionality, and
  - it does not impose burden on applications that do not require that functionality
- Flow control
  - Avoid the sender over-flowing the receiver buffer
  - Receiver only reads in-sequence data, and acks with the next sequence number is waiting for
  - Sender never sends more data than the receiver can hold in its buffer
- Congestion control:
  - Avoid server over-flowing the network
  - Increase sending rate until congestion; reduce sending rate when congestion
  - Use packet loss to detect congestion in the network