CS162
Operating Systems and Systems Programming
Lecture 22

E2E Argument, TCP Flow Control

November 26th, 2018

Prof. Ion Stoica

http://cs162.eecs.Berkeley.edu

Goals of Today's Lecture

- End-to-end principle (argument)

- TCP flow control

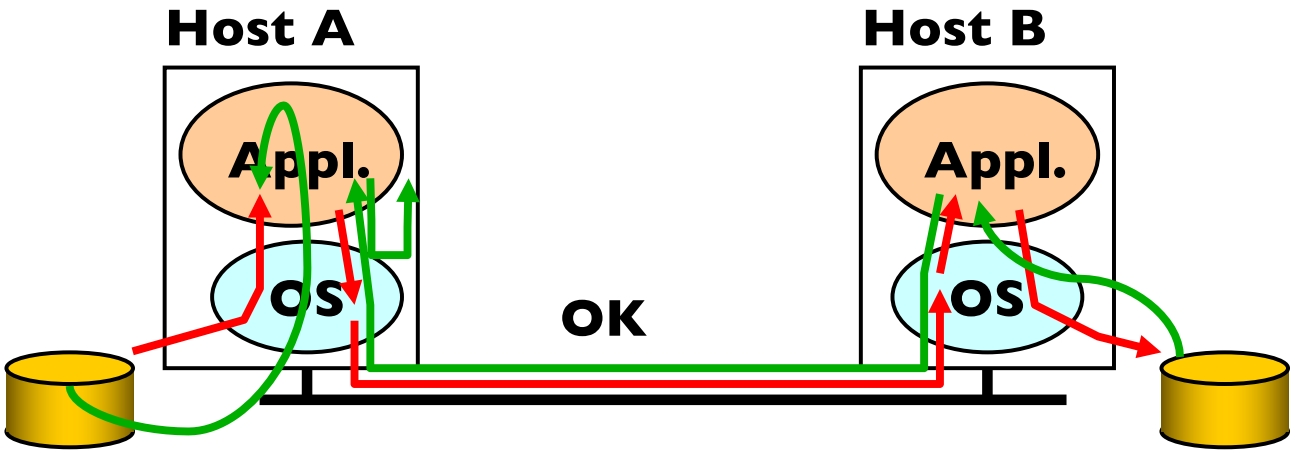
Basic Observation

- Some types of network functionality can only be correctly implemented end-to-end
  - Reliability, security, etc

- Because of this, end hosts:
  - Can satisfy the requirement without network's help
  - Will/**must** do so, since can't ***rely*** on network's help

- Therefore **don't** go out of your way to implement them in the network

Example: Reliable File Transfer



- Solution 1: make each step reliable, and then concatenate them
- Solution 2: end-to-end **check** and try again if necessary

Discussion

- Solution 1 is incomplete
  - What happens if memory is corrupted?
  - Receiver has to do the check anyway!
- Solution 2 is complete
  - Full functionality can be entirely implemented at application layer with no need for reliability from lower layers
- *Is there any need to implement reliability at lower layers?*
  - Well, it could be more efficient

End-to-End Principle

Implementing this functionality in the network:

- Doesn't reduce host implementation complexity
- Does increase network complexity
- Probably imposes delay and overhead on all applications, even if they don't need functionality

- However, implementing in network can enhance performance in some cases
  - E.g., very lossy link

Conservative Interpretation of E2E

- Don't implement a function at the lower levels of the system unless it can be completely implemented at this level
- Unless you can relieve the burden from hosts, don't bother

Moderate Interpretation

- Think twice before implementing functionality in the network
- If hosts can implement functionality correctly, implement it in a lower layer only as a performance enhancement
- But do so only if it does not impose burden on applications that do not require that functionality
- This is the interpretation we are using

BREAK

Administrivia

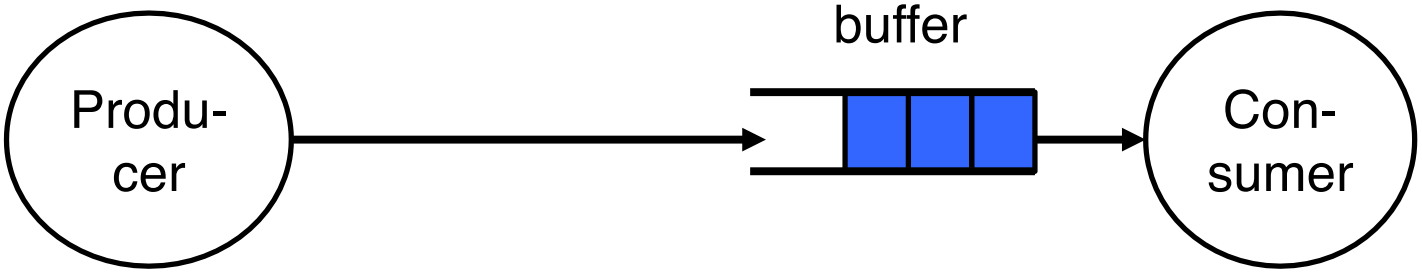
- Midterm 3 coming up on **Wen 11/28 5:00-6:30PM**
  - All topics:
    - » Focus will be on Lectures 18 – 23 and associated readings, and Projects 3
    - » But expect 20-30% questions from materials from Lectures 1-17
  - Closed book
  - 2 pages hand-written notes both sides

Goals of Today's Lecture

- End-to-end principle (argument)
- TCP flow control

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



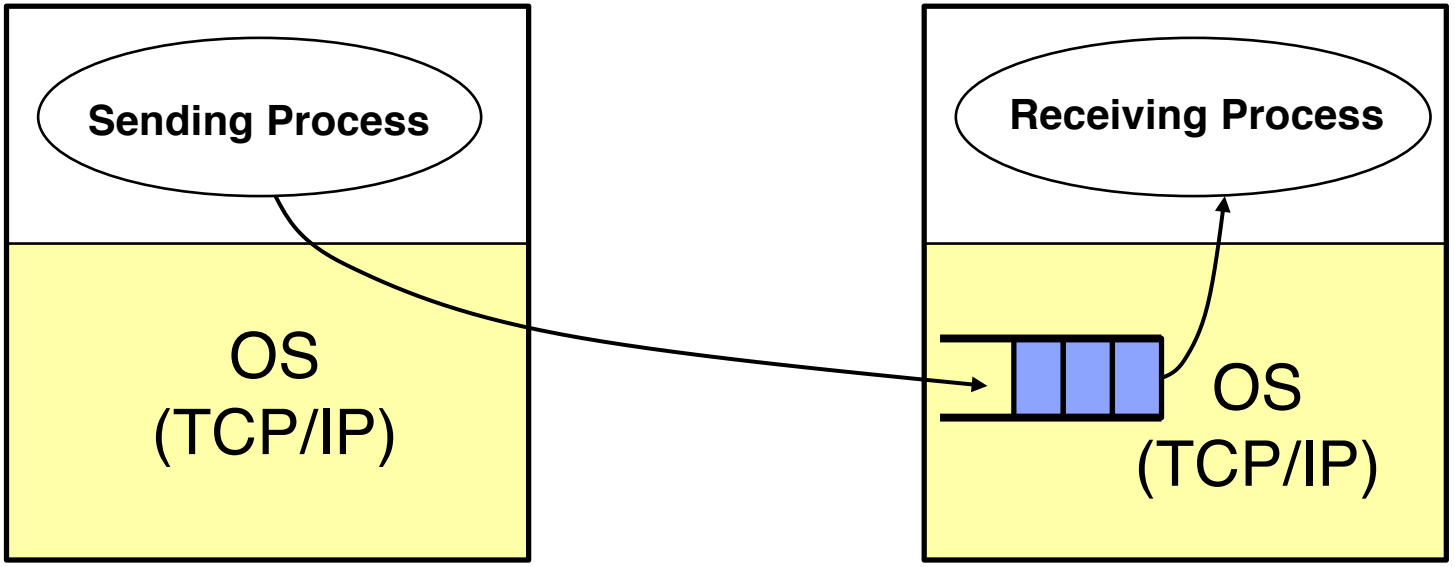
TCP Flow Control

- 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)

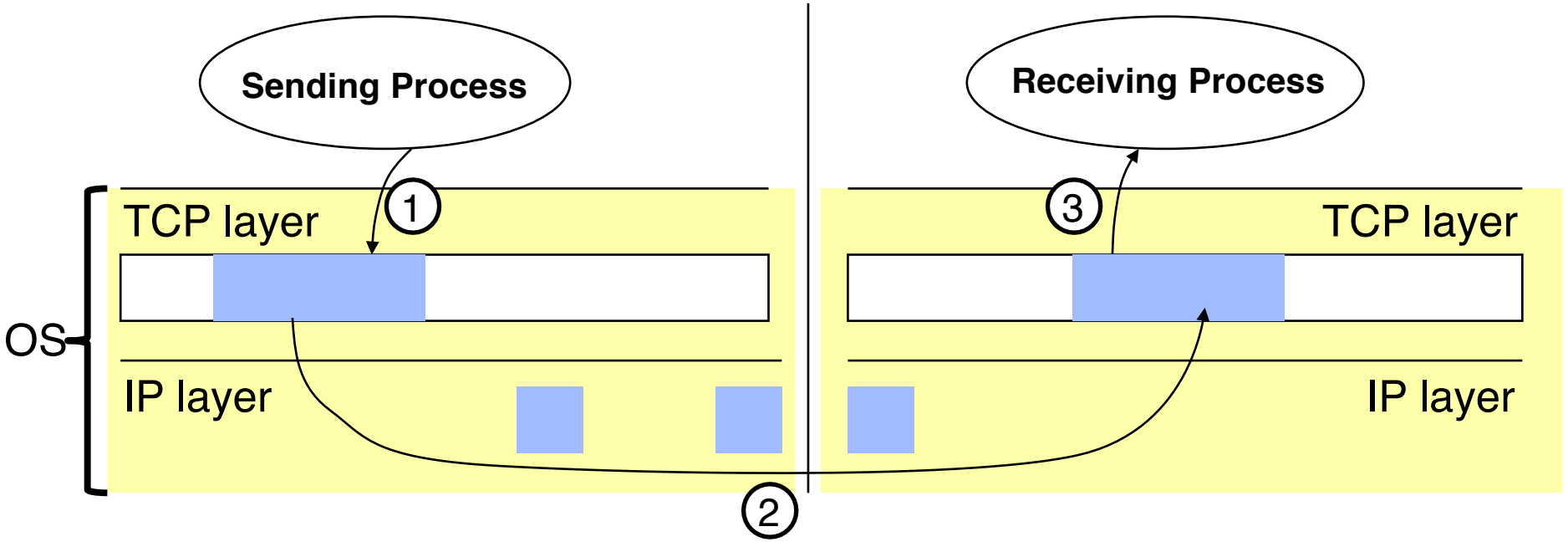
- The ack(nowledgement) contains sequence number N of next byte the receiver expects, i.e., receiver has received all bytes in sequence up to and including N-1

TCP Flow Control



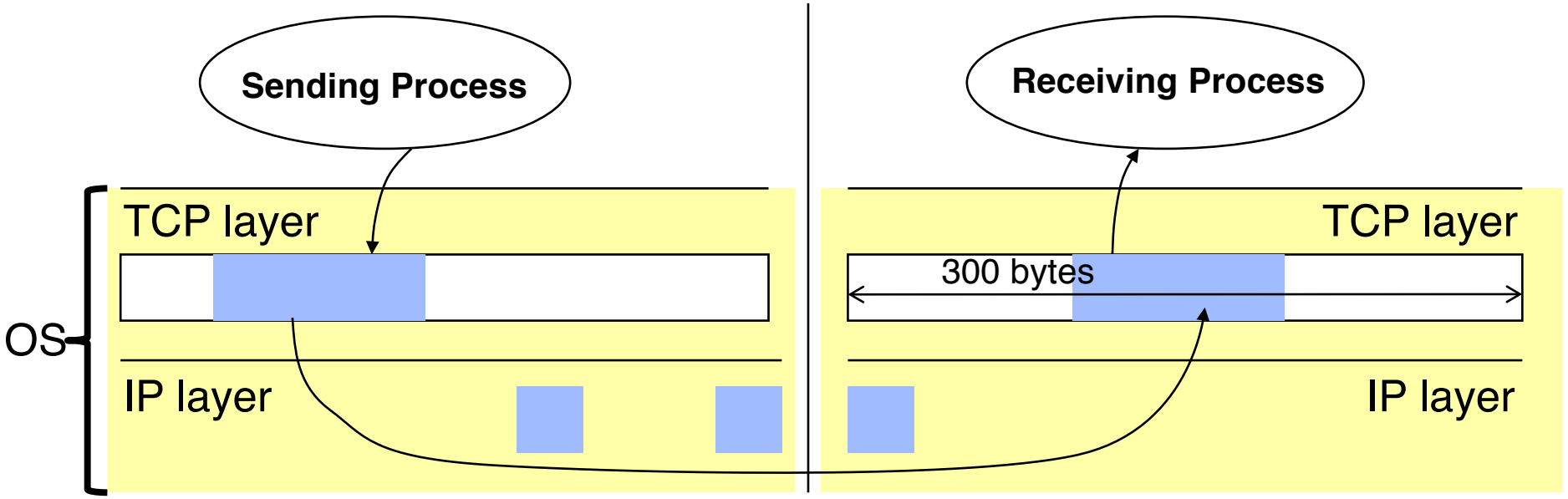
- TCP/IP implemented by OS (Kernel)
  - Cannot do context switching on sending/receiving every packet
    - » At 1Gbps, it takes 12 usec to send an 1500 bytes, and 0.8usec to send an 100 byte packet
- Need buffers to match …
  - sending app with sending TCP
  - receiving TCP with receiving app

TCP Flow Control



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

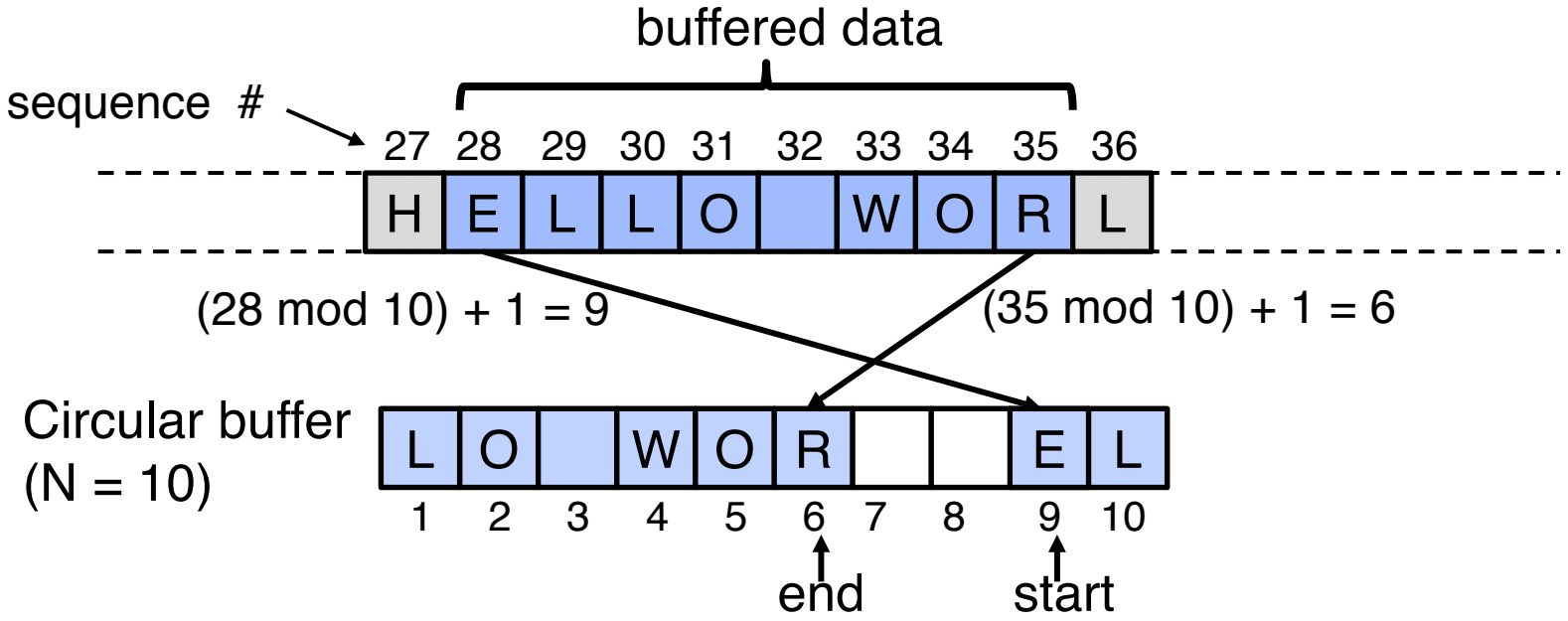
TCP Flow Control



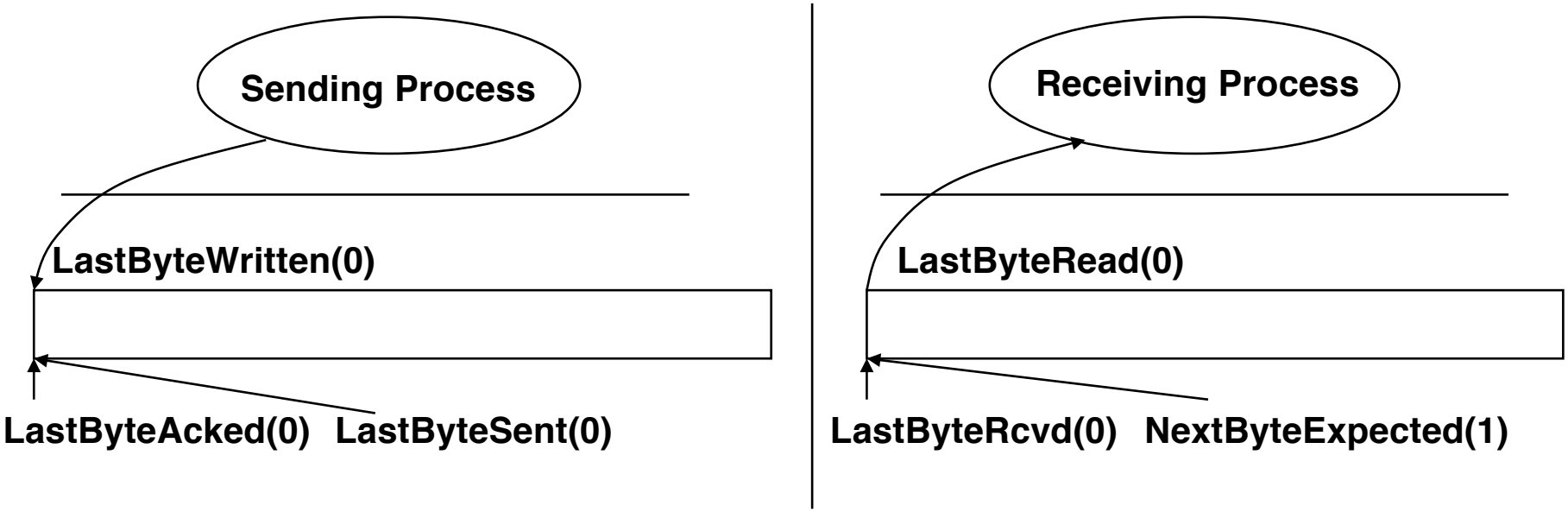
- Example assumptions:
  - Maximum IP packet size = 100 bytes
  - Size of the receiving buffer (MaxRcvBuf) = 300 bytes
- Recall, ack indicates the next expected byte in-sequence, not the last received byte
- Use circular buffers

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

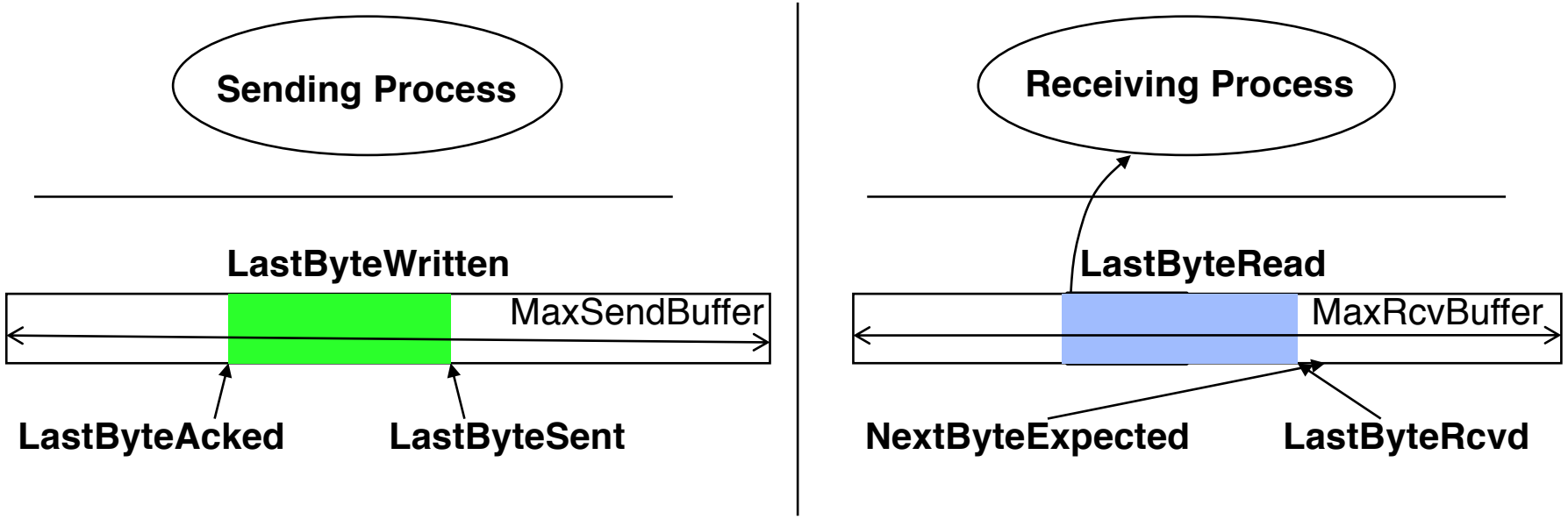


TCP Flow Control



- 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



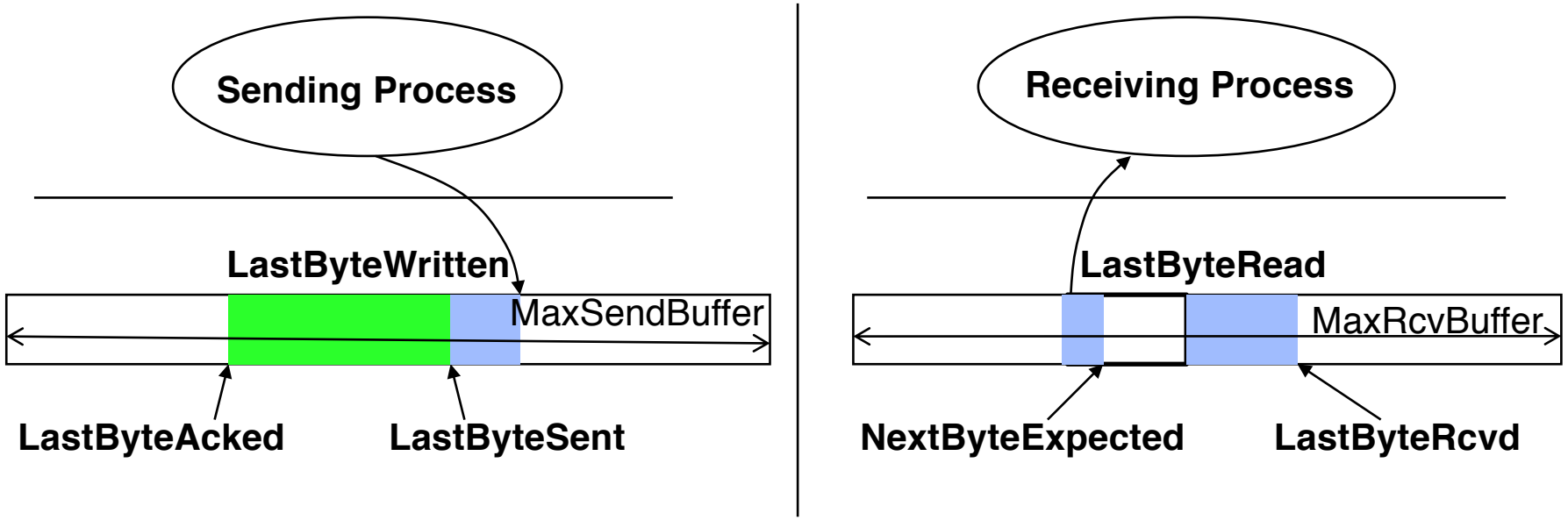
- AdvertisedWindow: number of bytes TCP receiver can receive

AdvertisedWindow = MaxRcvBuffer – (LastByteRcvd – LastByteRead)

- SenderWindow: number of bytes TCP sender can send

SenderWindow = AdvertisedWindow – (LastByteSent – LastByteAcked)

TCP Flow Control



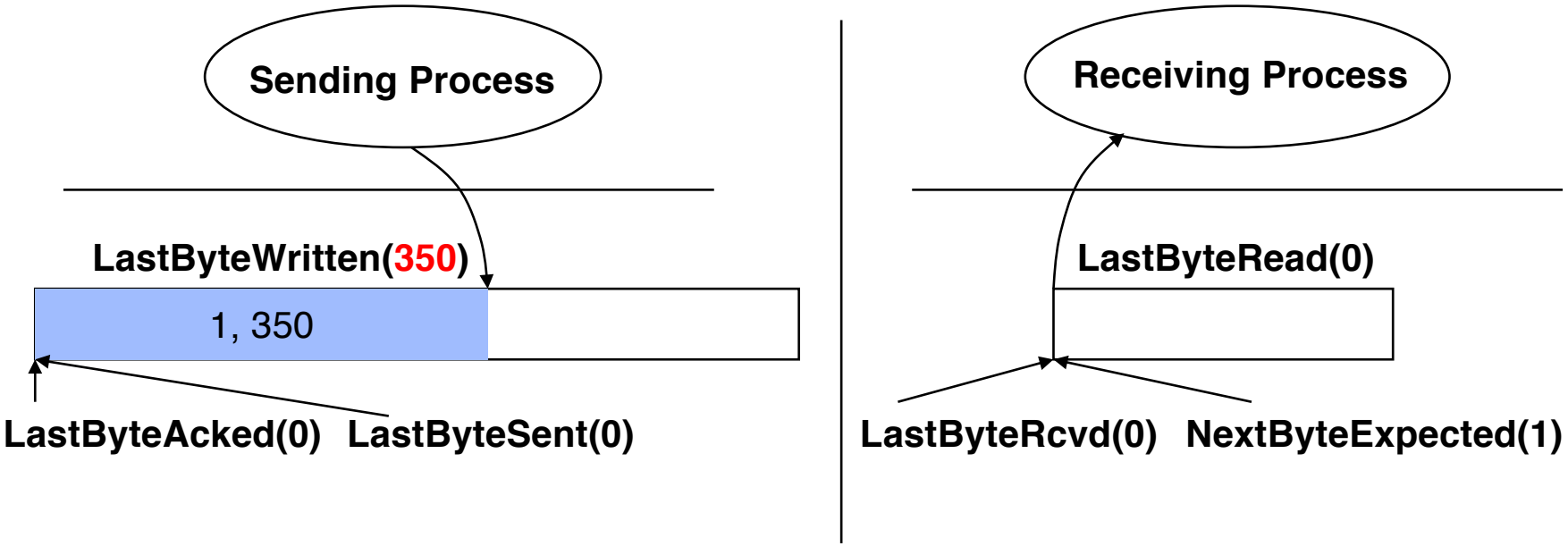
- Still true if receiver missed data….

AdvertisedWindow = MaxRcvBuffer – (LastByteRcvd – LastByteRead)

- WriteWindow: number of bytes sending process can write

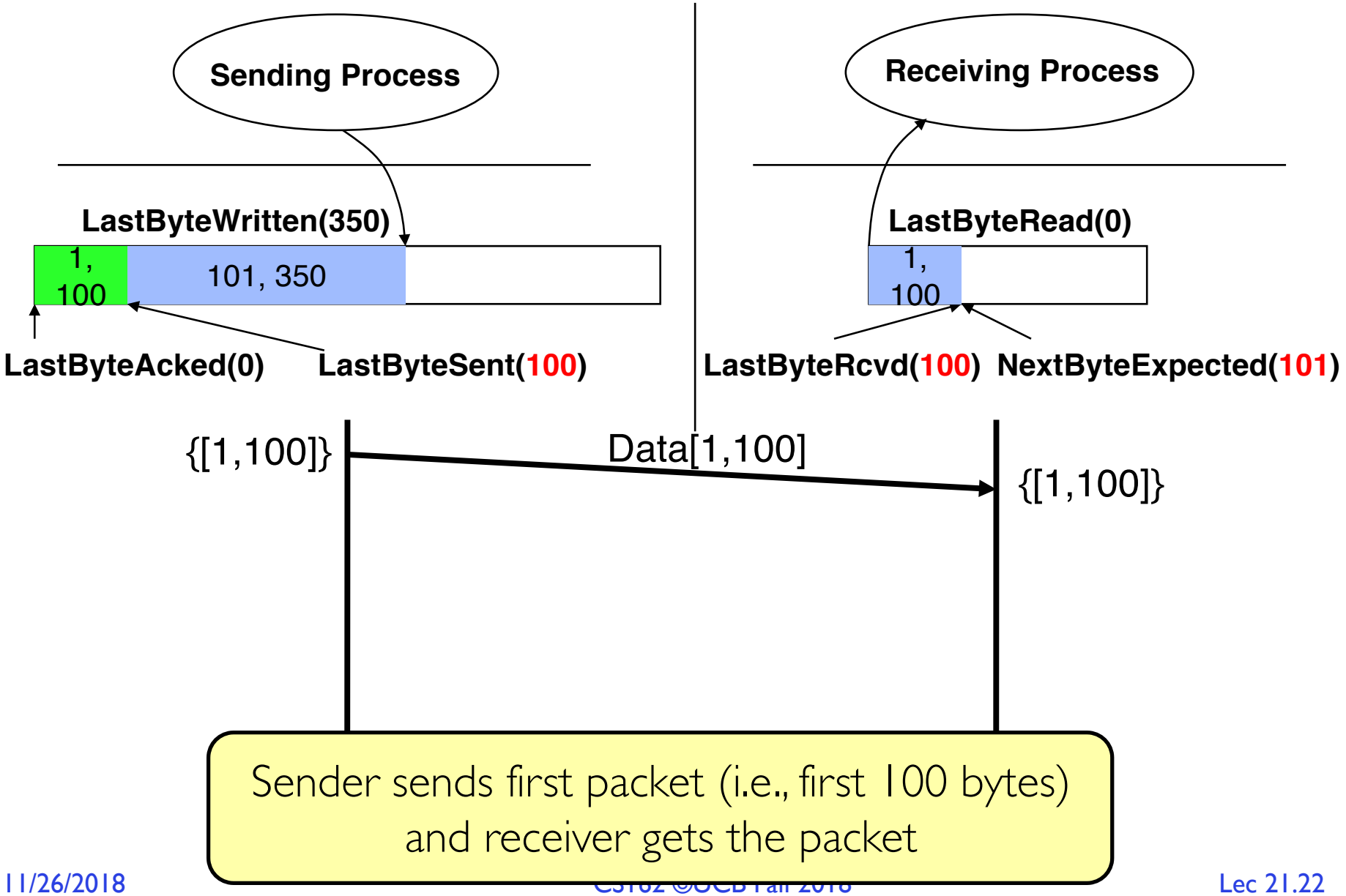
WriteWindow = MaxSendBuffer – (LastByteWritten – LastByteAcked)

TCP Flow Control

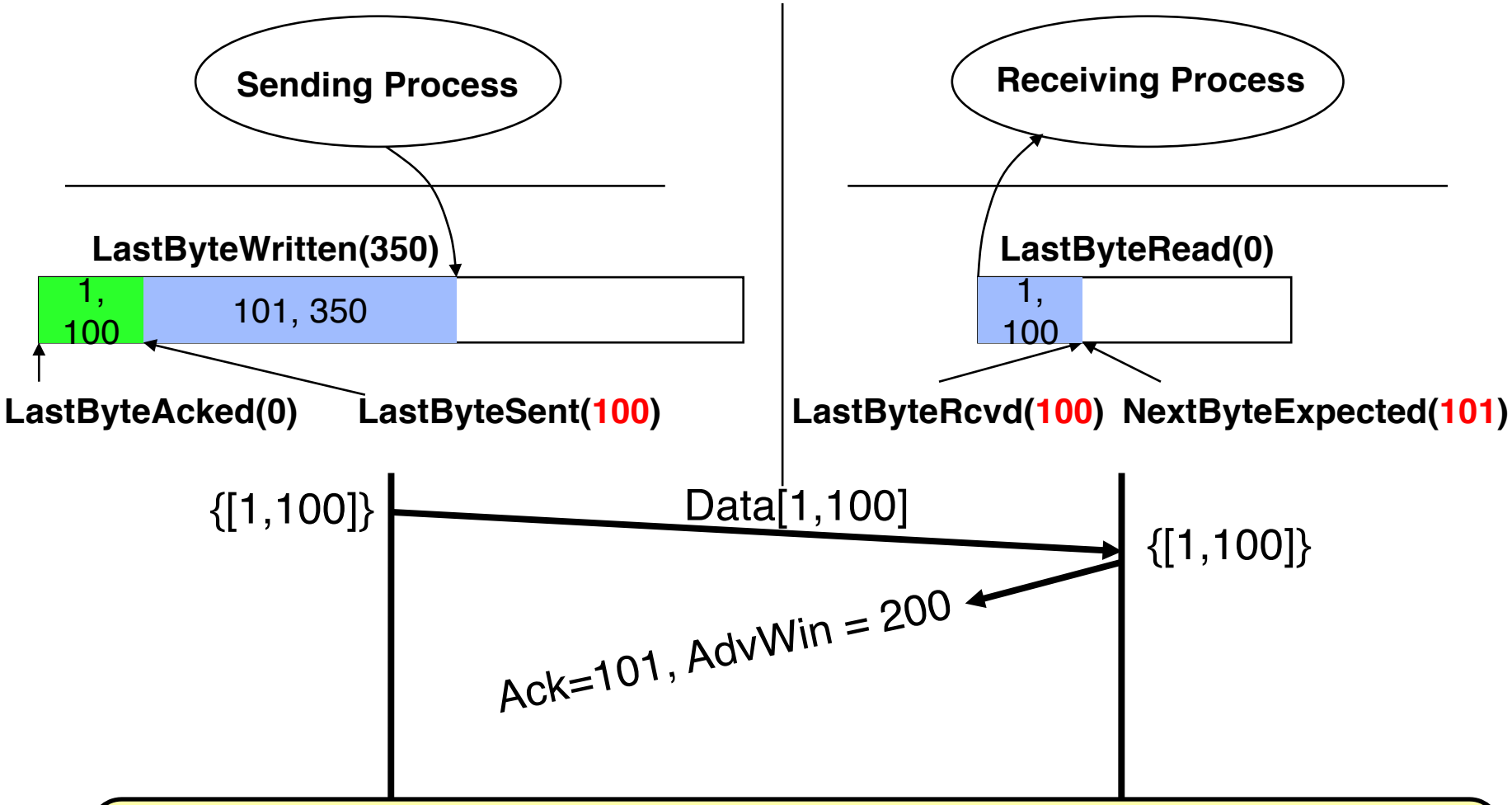


- Sending app sends 350 bytes
- Recall:
  - We assume IP only accepts packets no larger than 100 bytes
  - MaxRcvBuf = 300 bytes, so initial Advertised Window = 300 byets

TCP Flow Control



TCP Flow Control

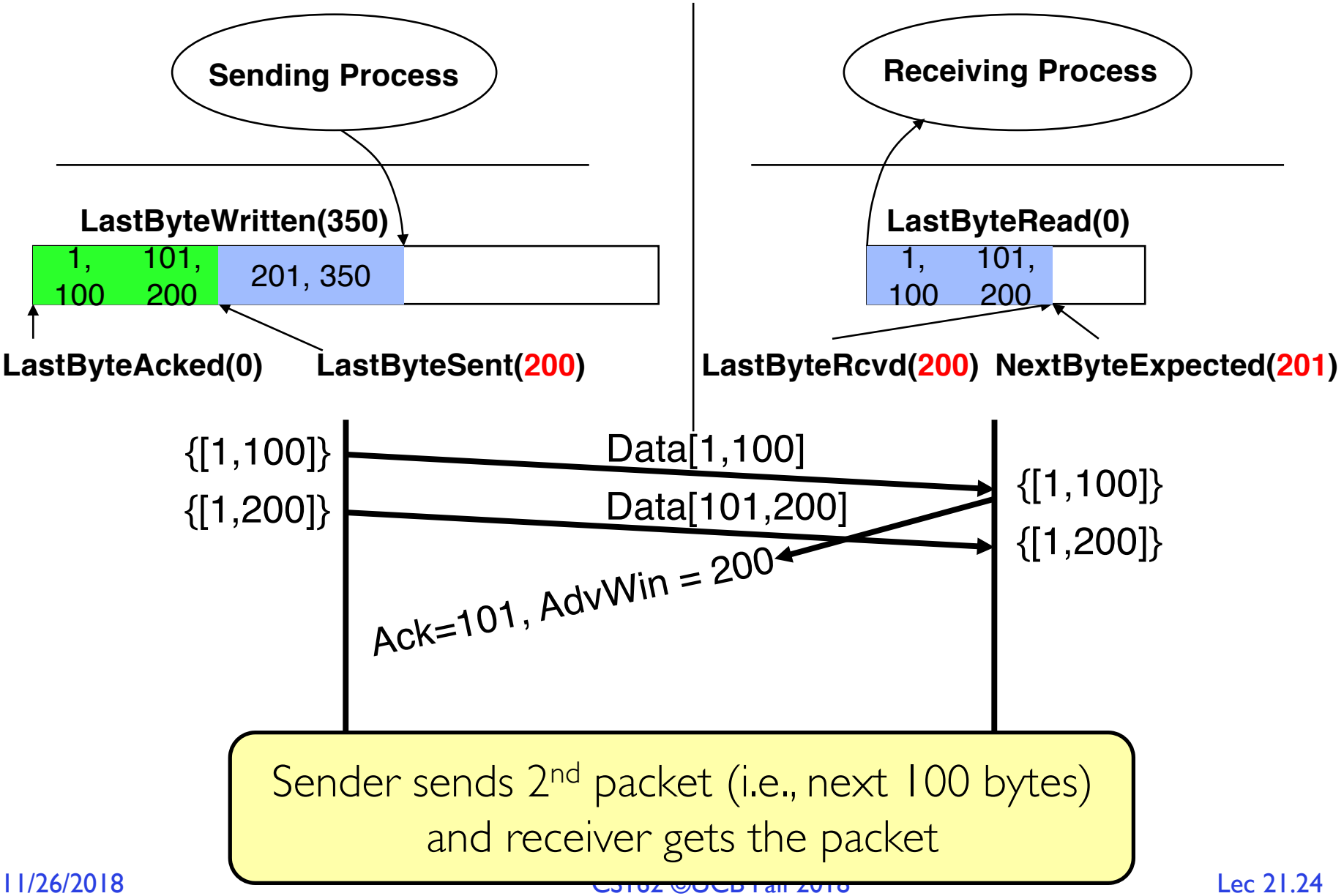


Receiver sends ack for 1st packet

AdvWin = MaxRcvBuffer – (LastByteRcvd – LastByteRead)

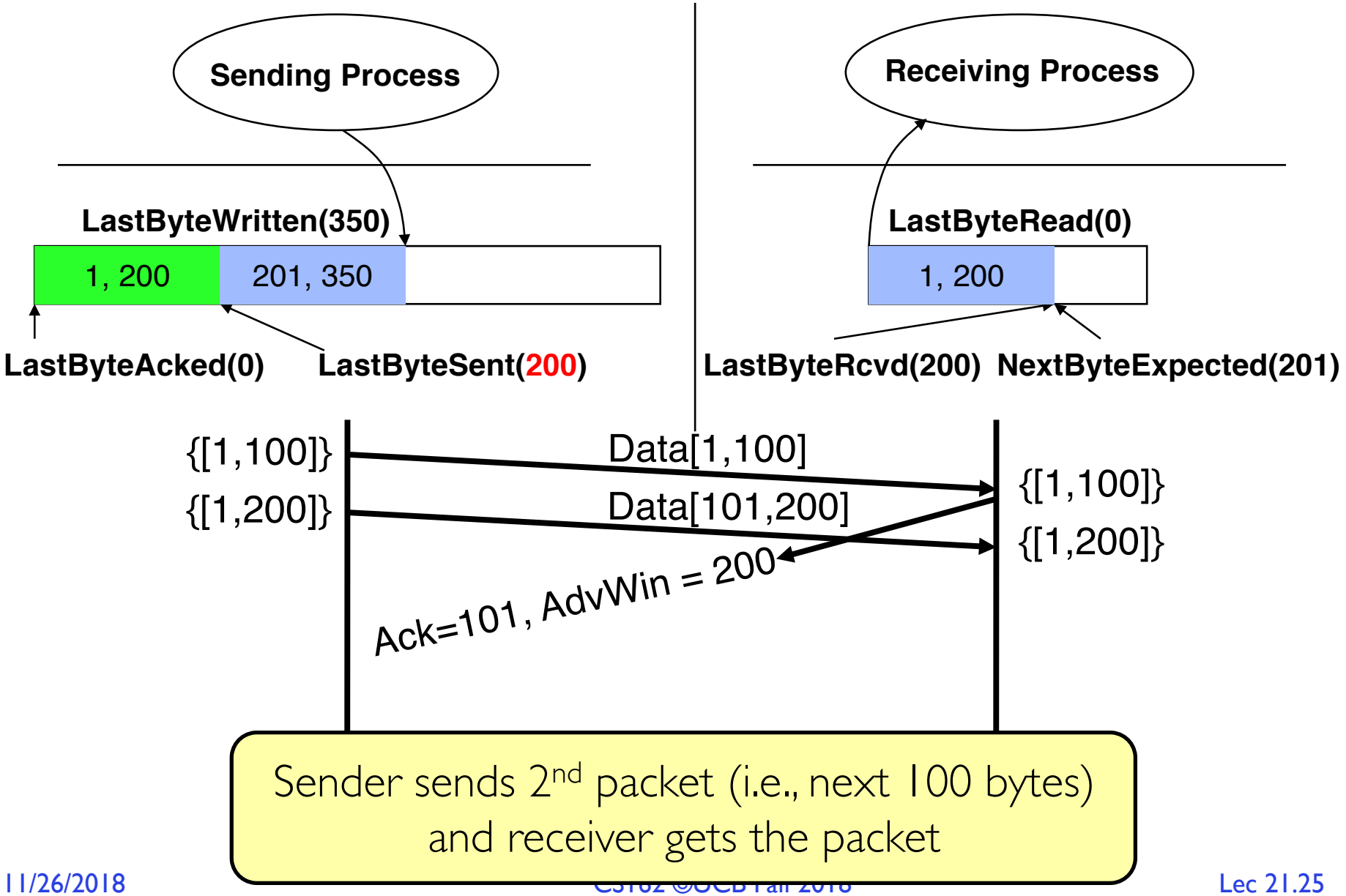
= 300 – (100 – 0) = 200

TCP Flow Control

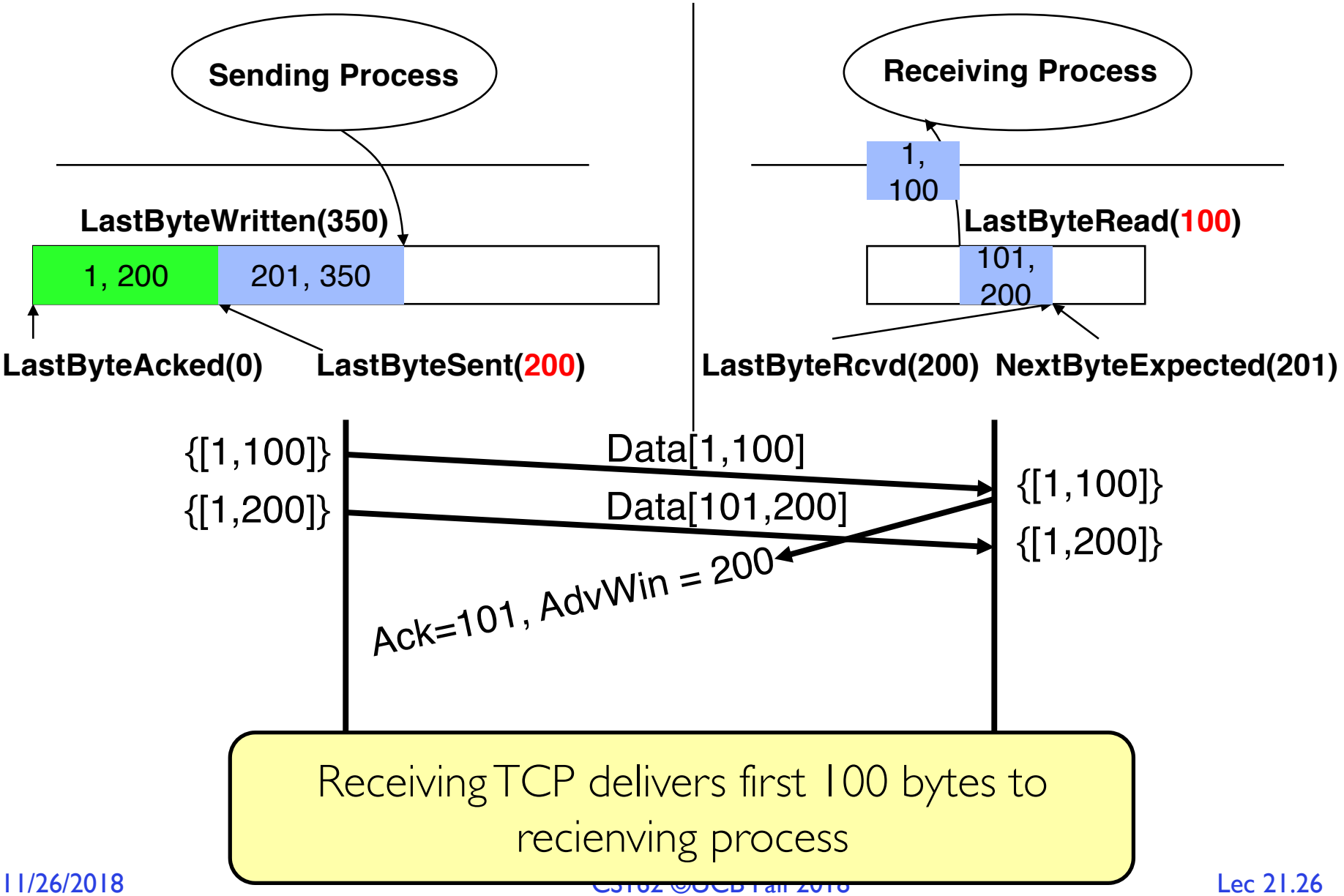


Sender sends 2nd packet (i.e., next 100 bytes) and receiver gets the packet

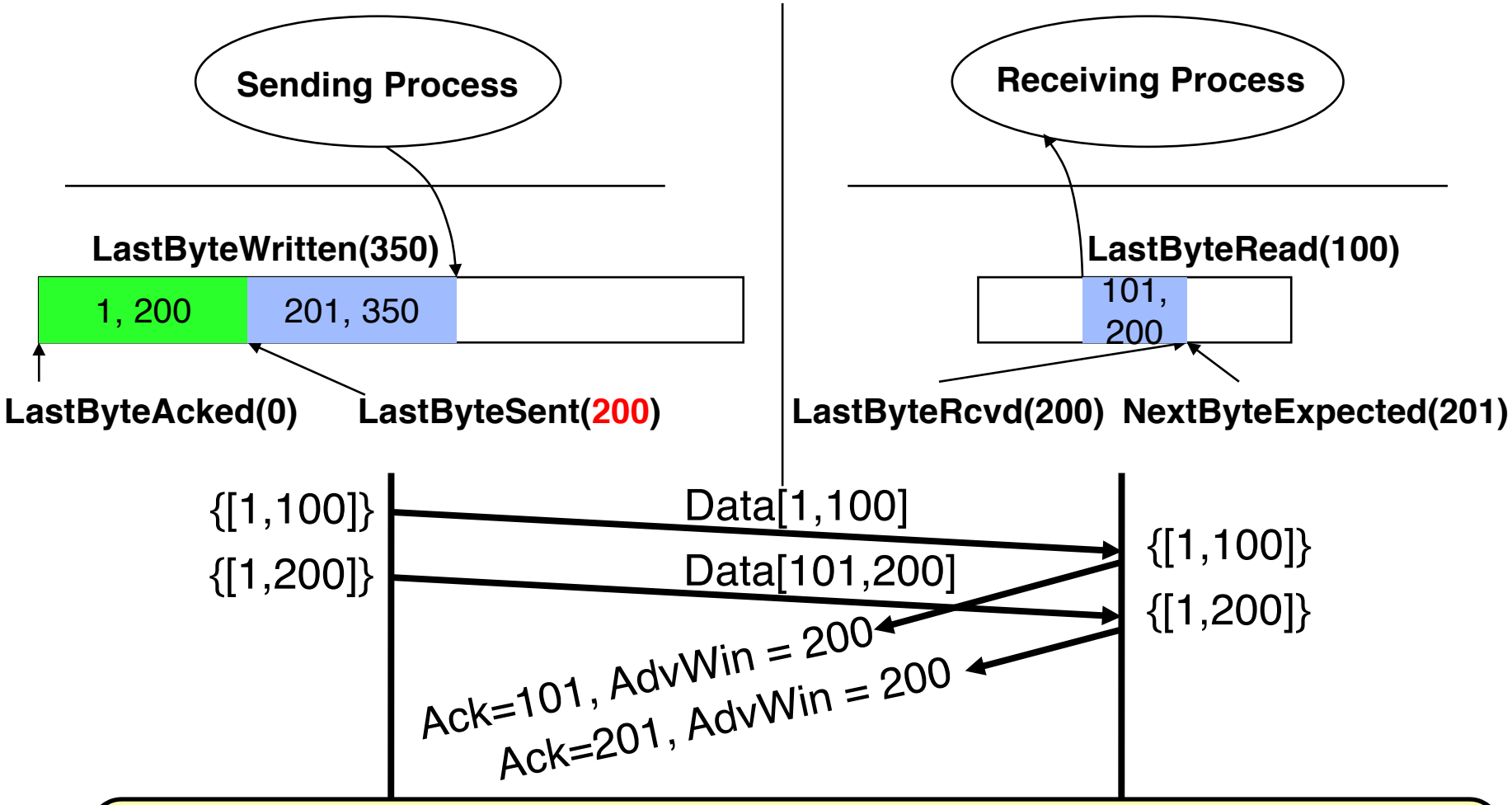
TCP Flow Control



TCP Flow Control



TCP Flow Control

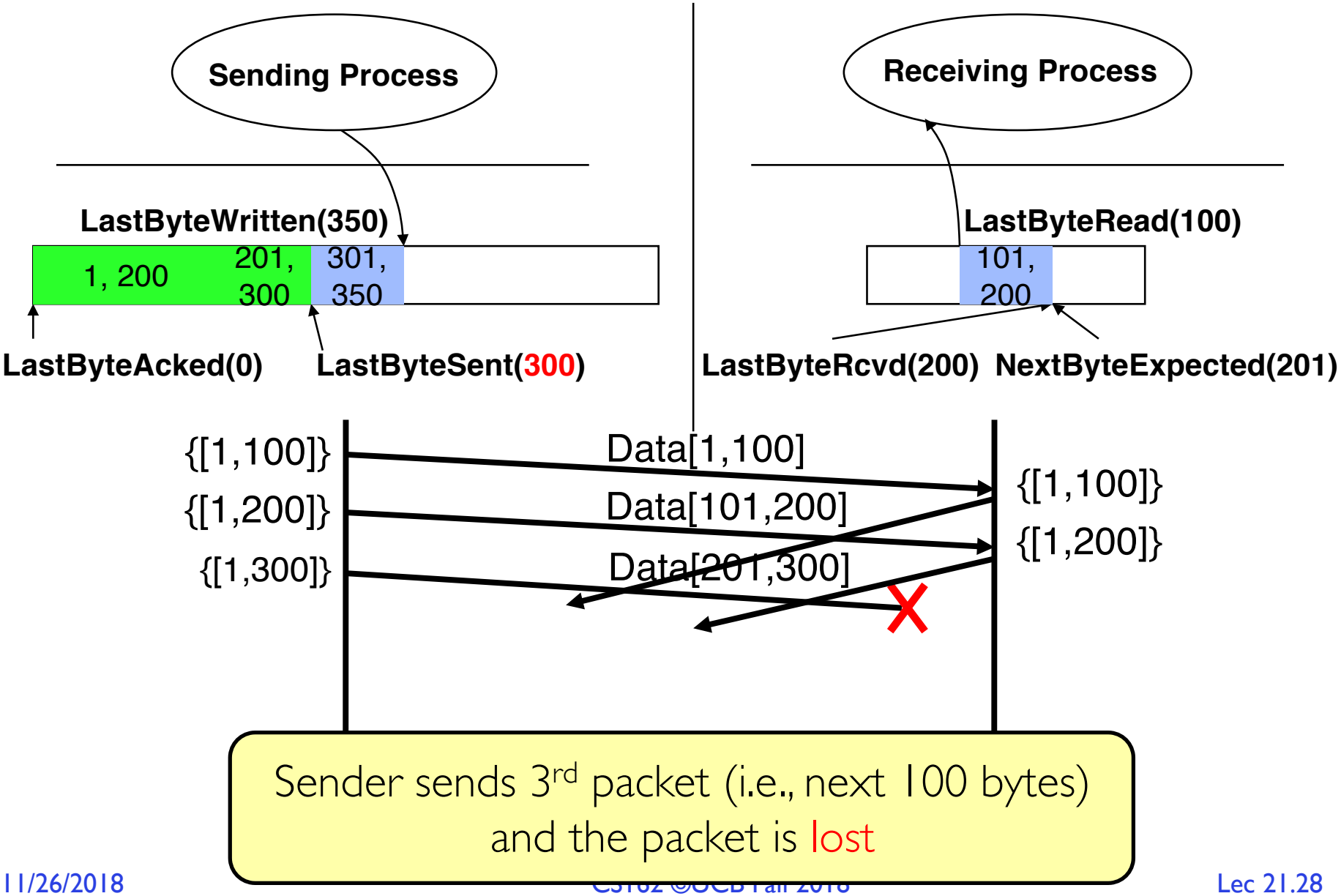


Receiver sends ack for 2nd packet

AdvWin = MaxRcvBuffer – (LastByteRcvd – LastByteRead)

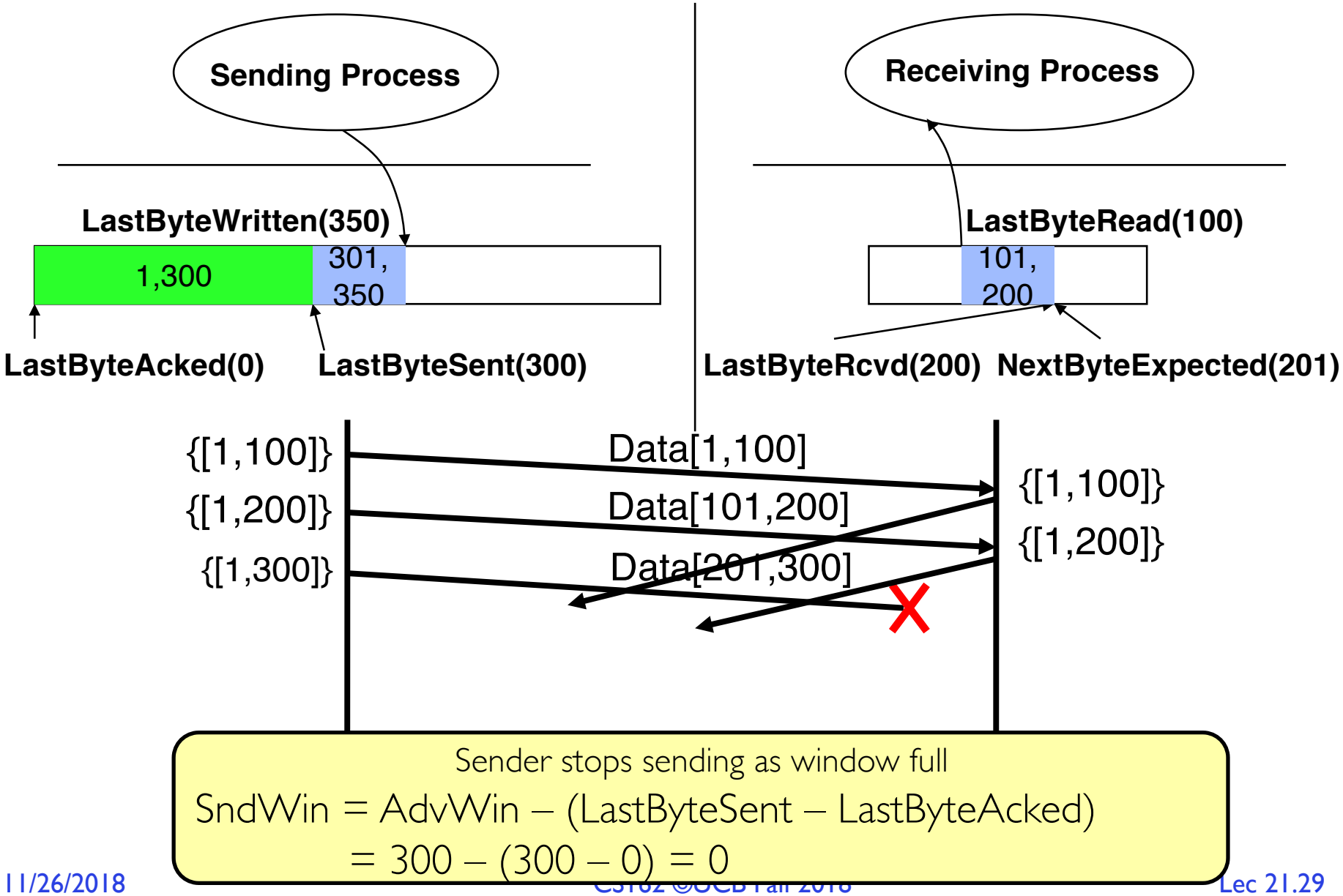
= 300 – (200 – 100) = 200

TCP Flow Control



Sender sends 3rd packet (i.e., next 100 bytes) and the packet is lost

TCP Flow Control

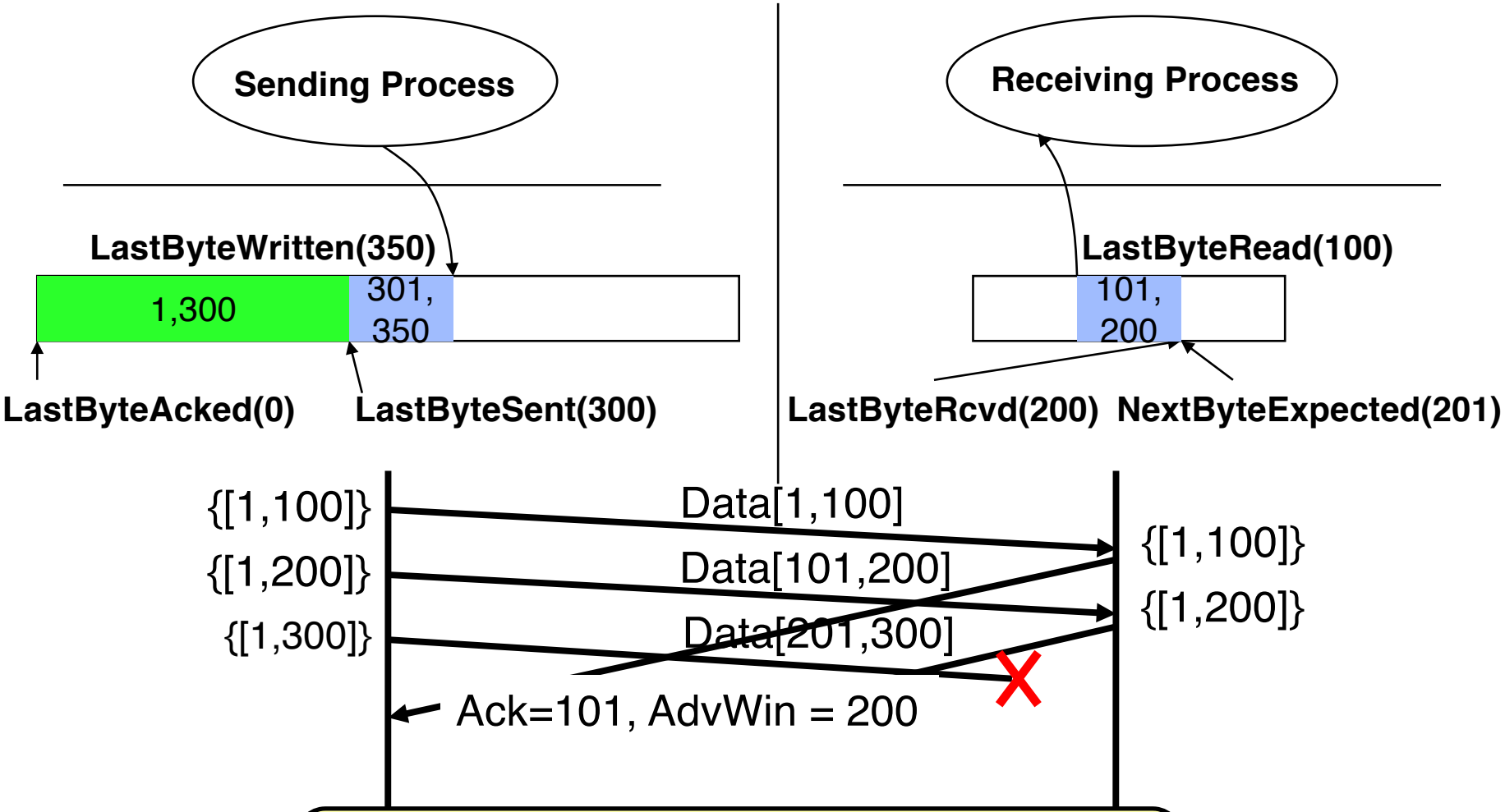


Sender stops sending as window full

SndWin = AdvWin – (LastByteSent – LastByteAcked)

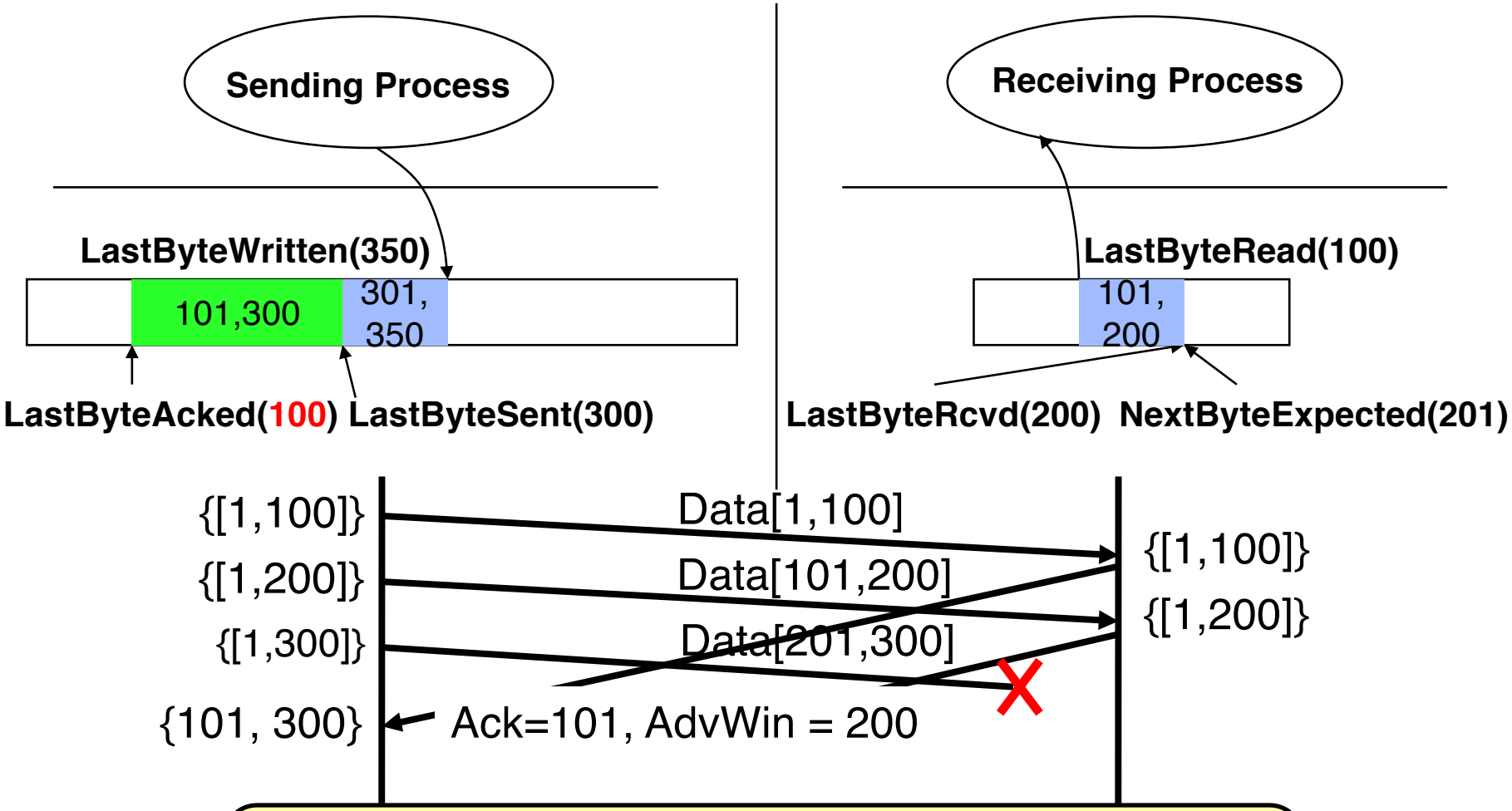
= 300 – (300 – 0) = 0

TCP Flow Control



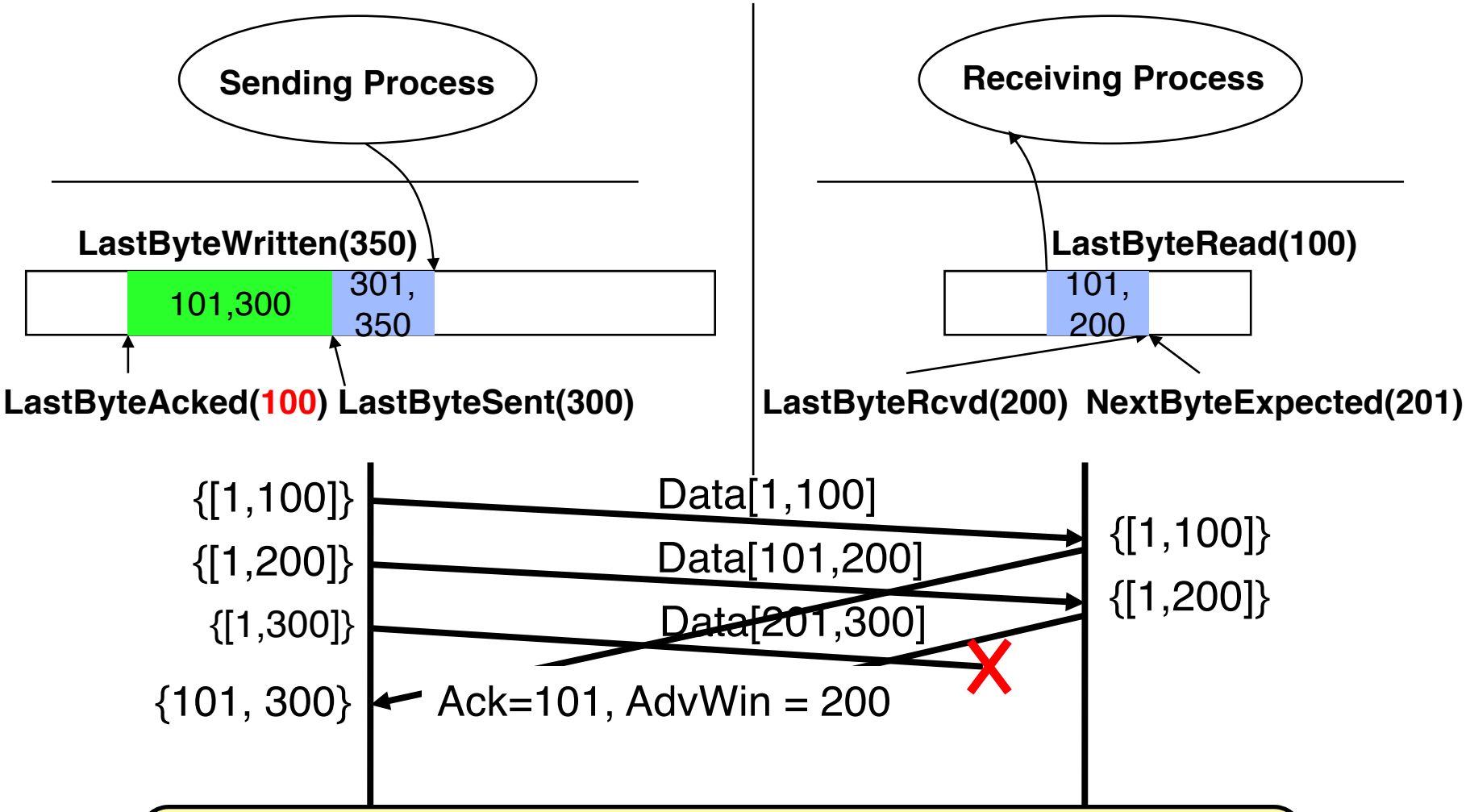
- Sender gets ack for 1st packet
- AdWin = 200

TCP Flow Control



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

TCP Flow Control

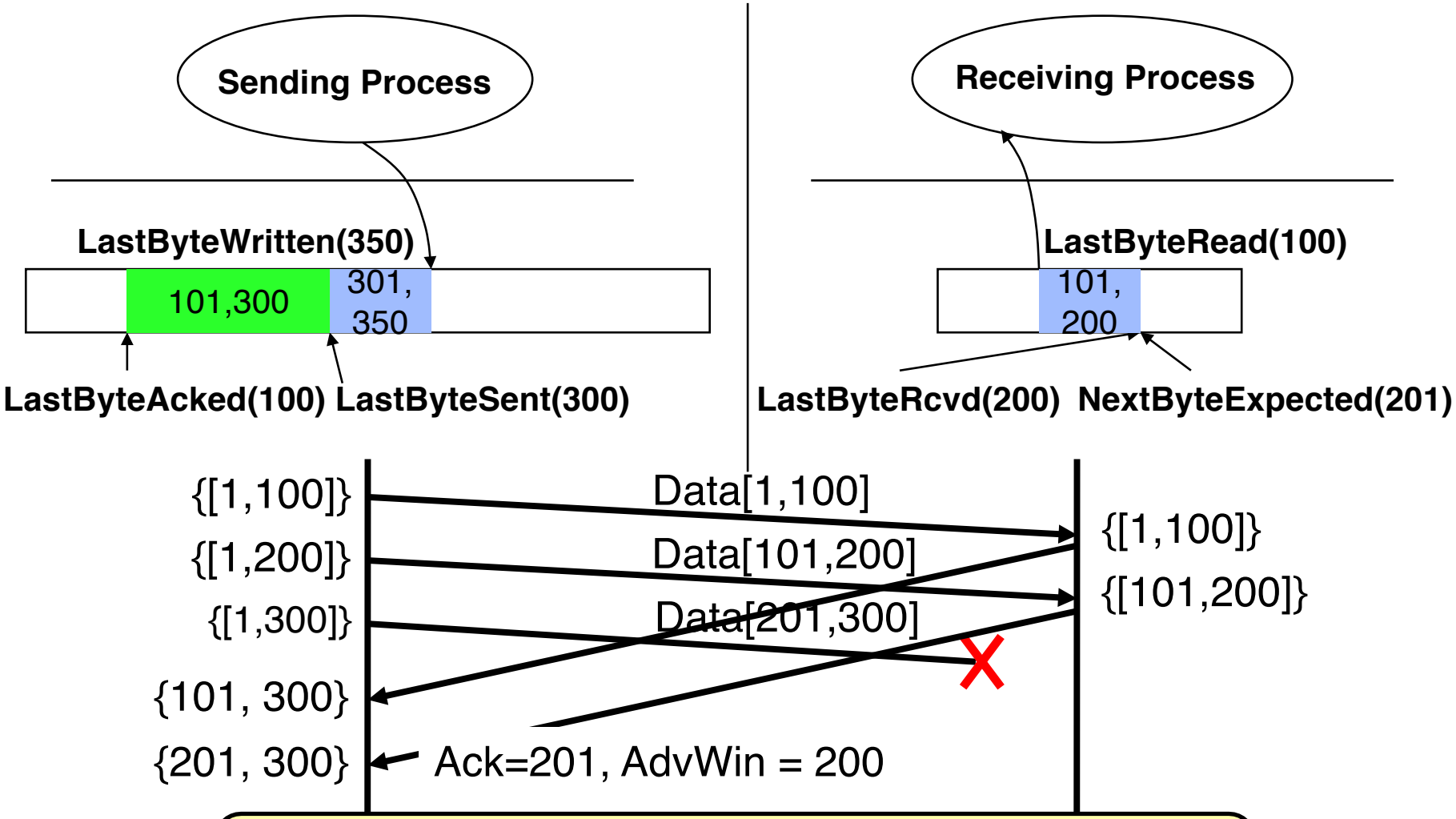


Sender still cannot send as window full

SndWin = AdvWin – (LastByteSent – LastByteAcked)

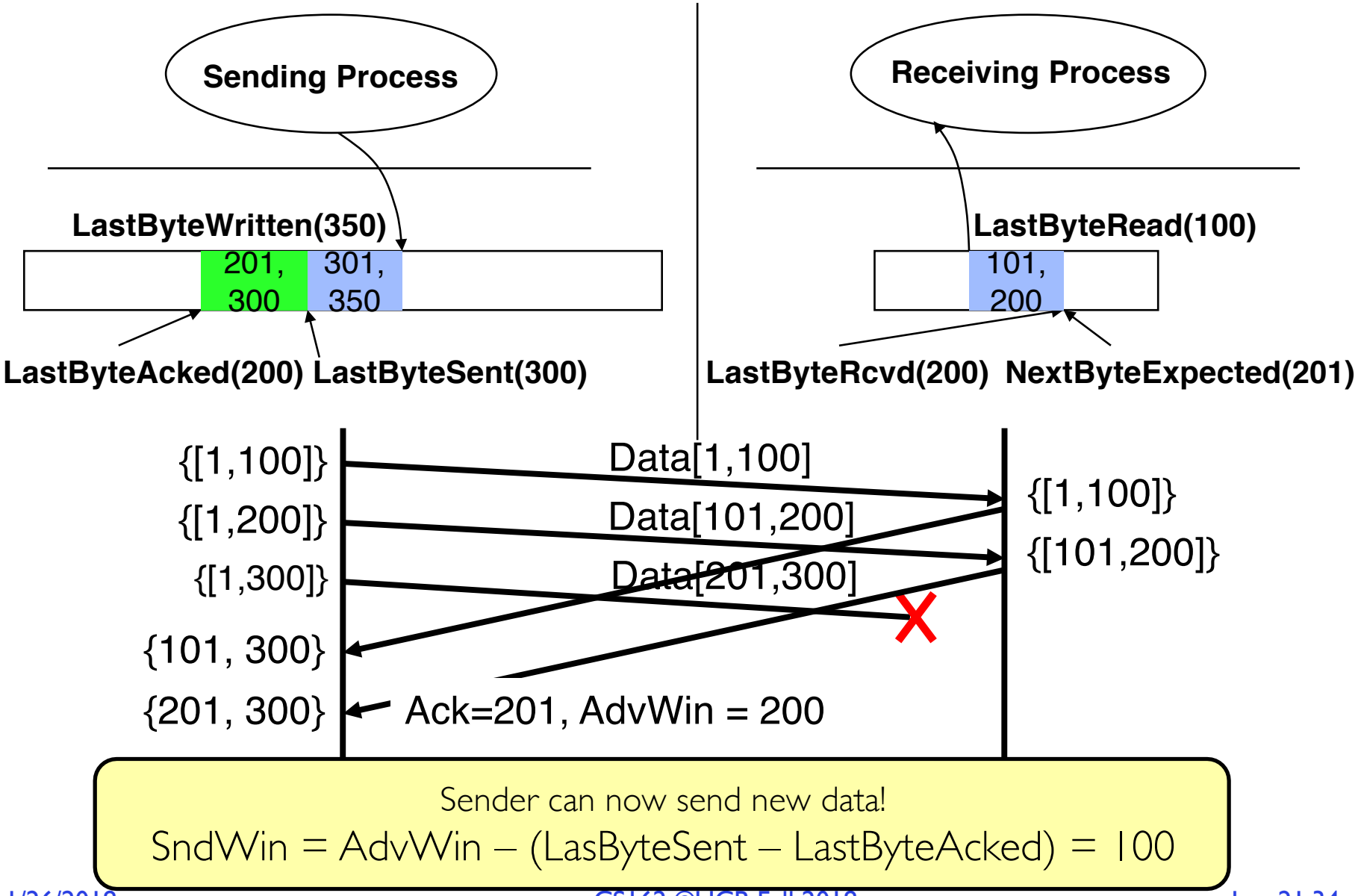
= 200 – (300 – 100) = 0

TCP Flow Control



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

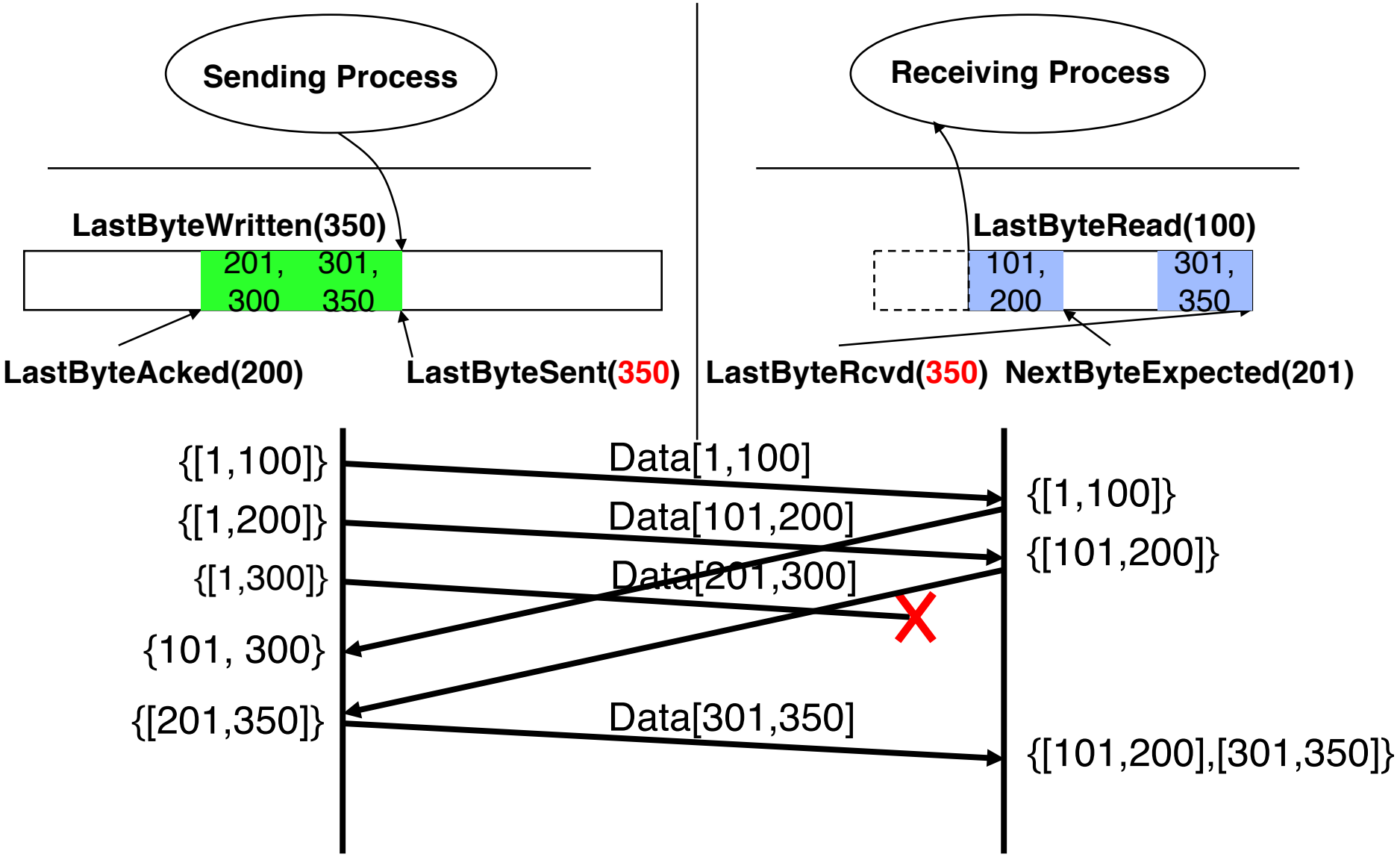
TCP Flow Control



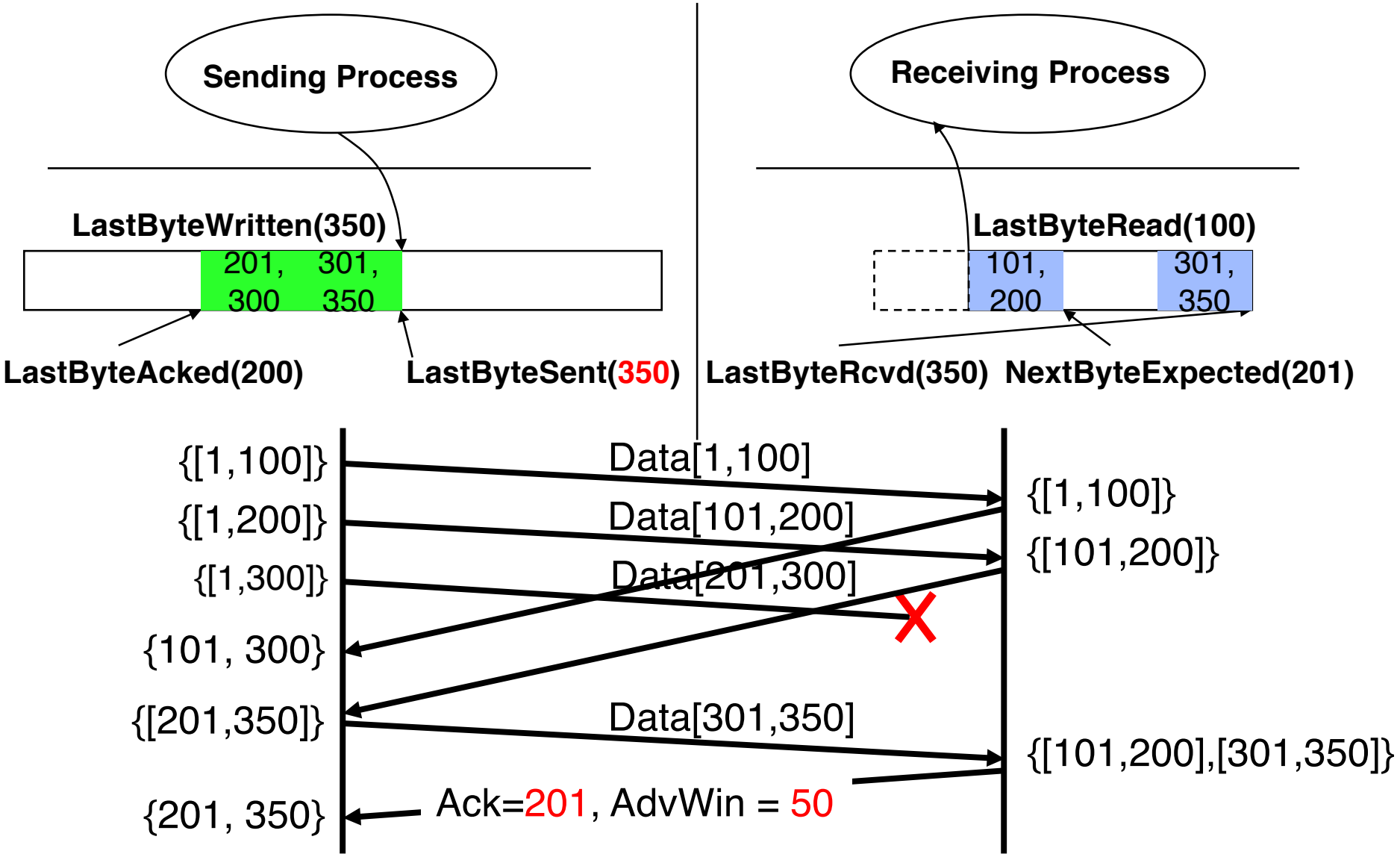
Sender can now send new data!

SndWin = AdvWin – (LasByteSent – LastByteAcked) = 100

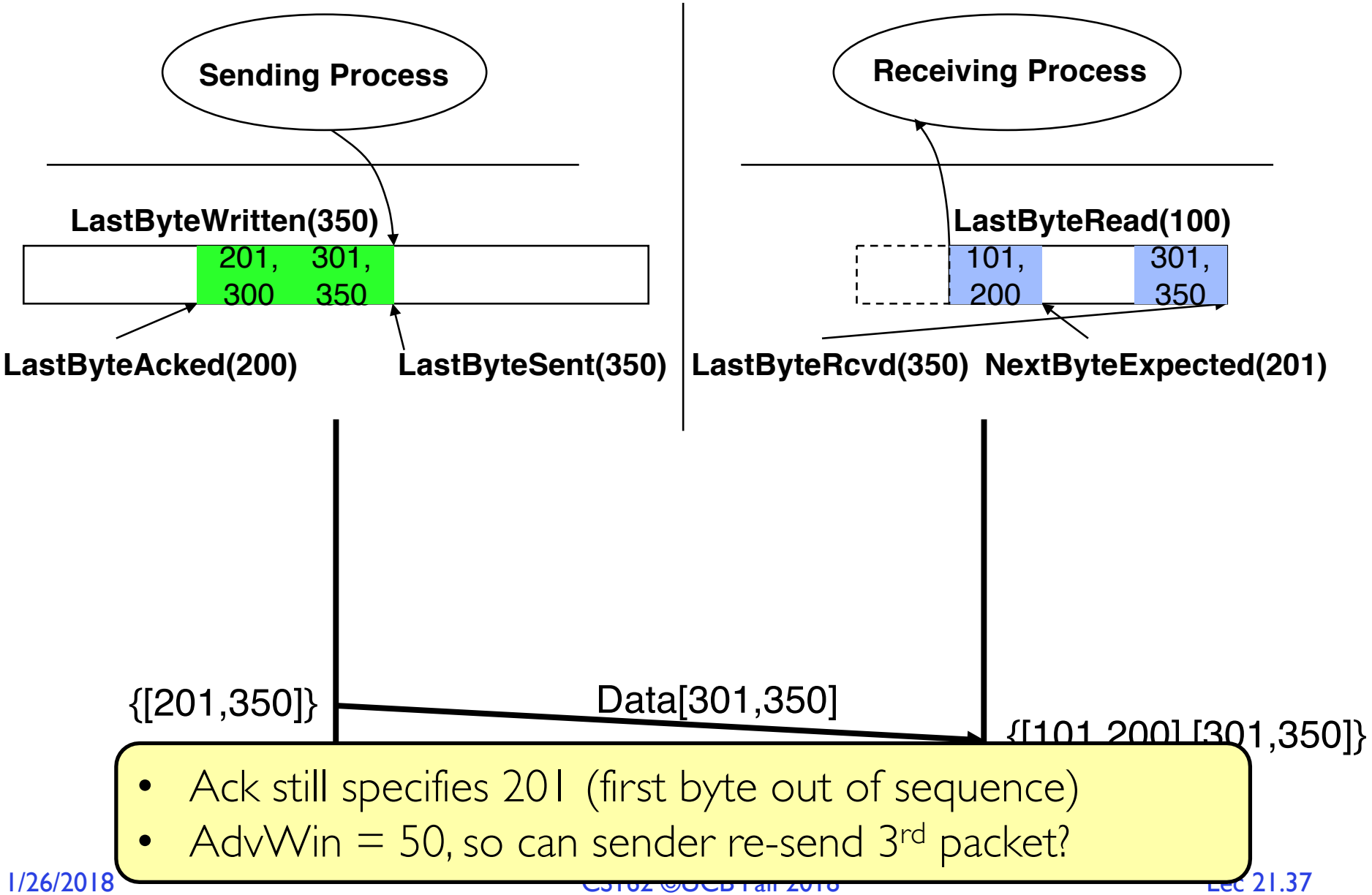
TCP Flow Control



TCP Flow Control

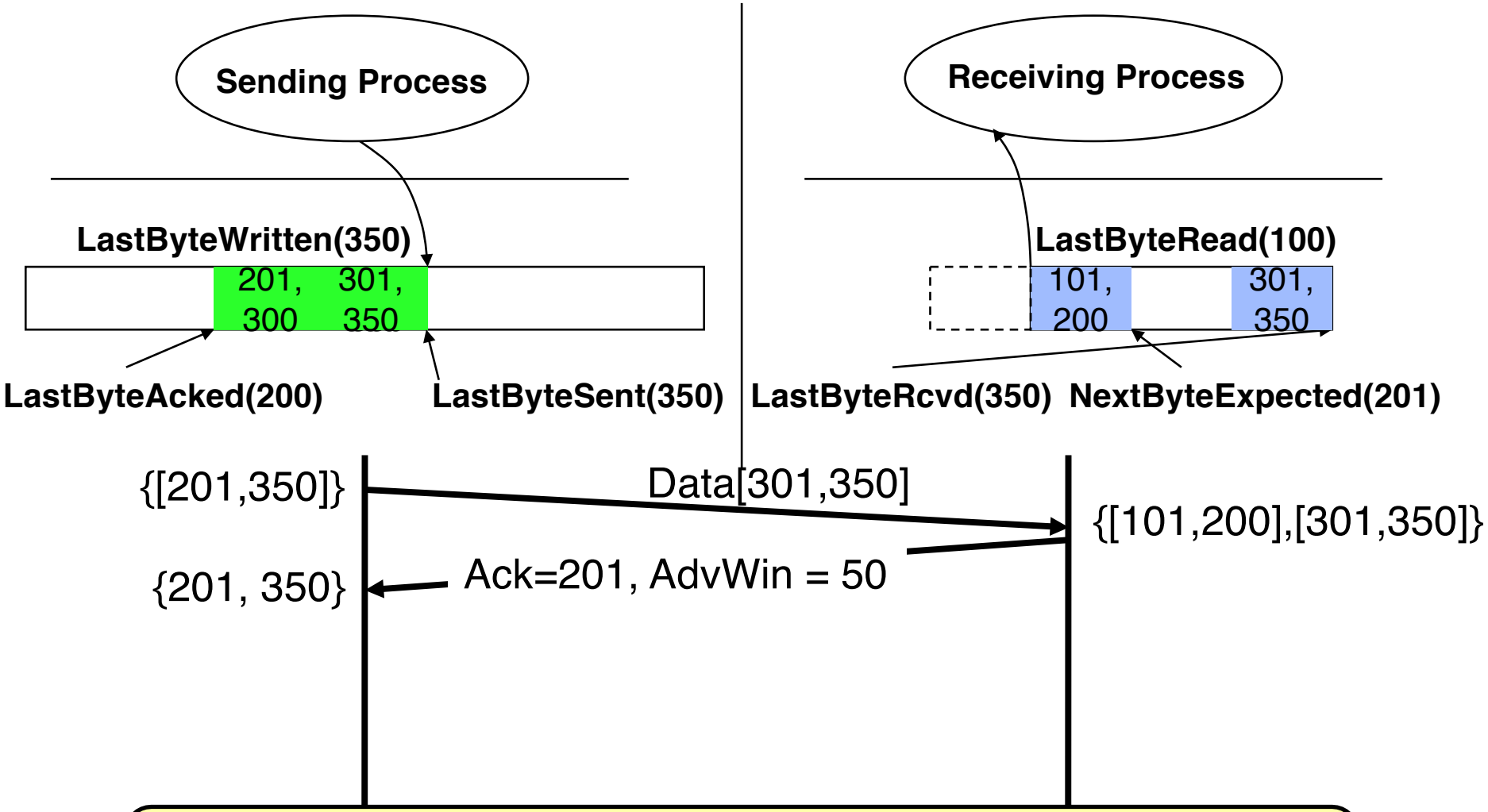


TCP Flow Control



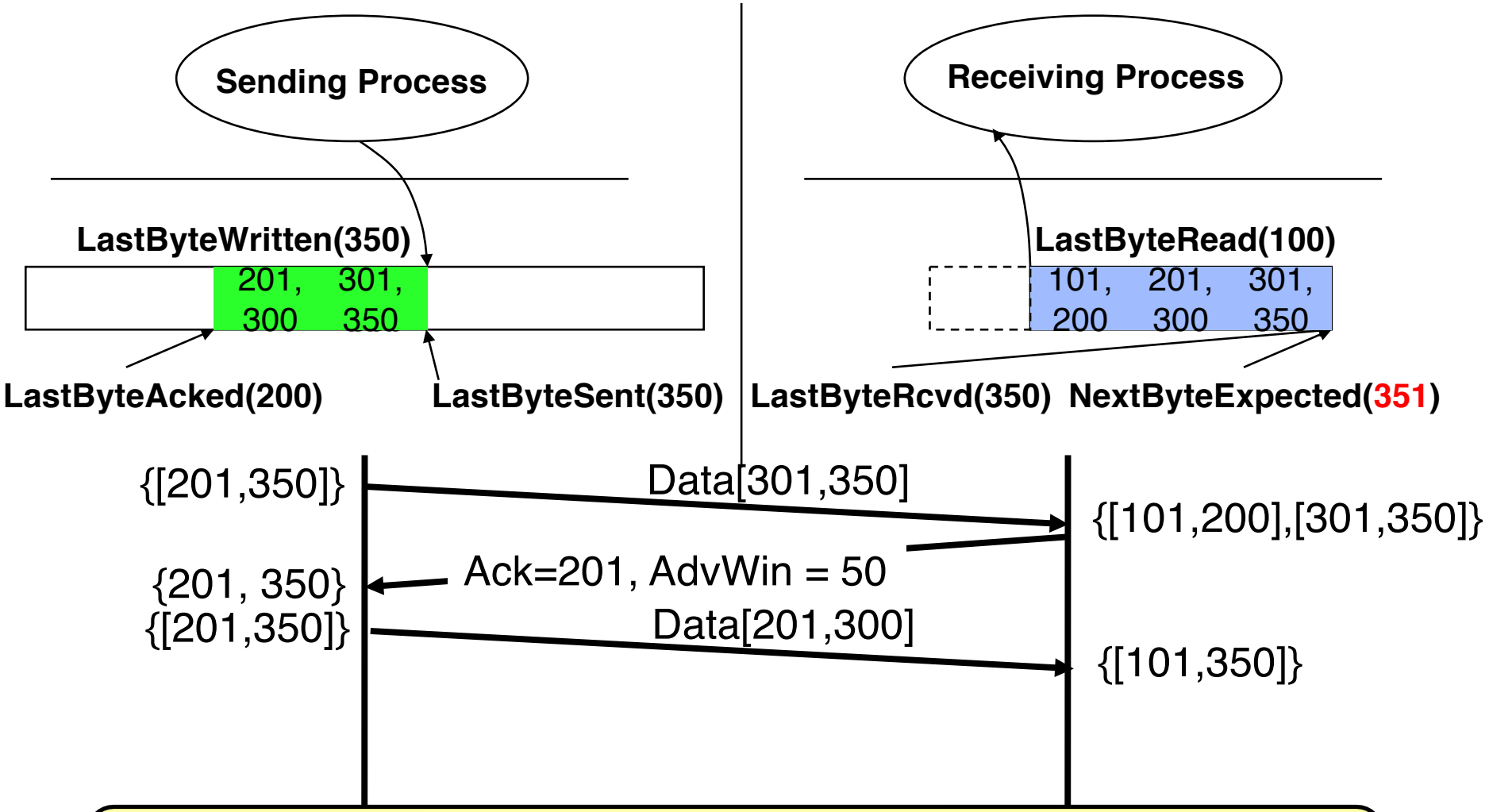
- Ack still specifies 201 (first byte out of sequence)
- AdvWin = 50, so can sender re-send 3rd packet?

TCP Flow Control



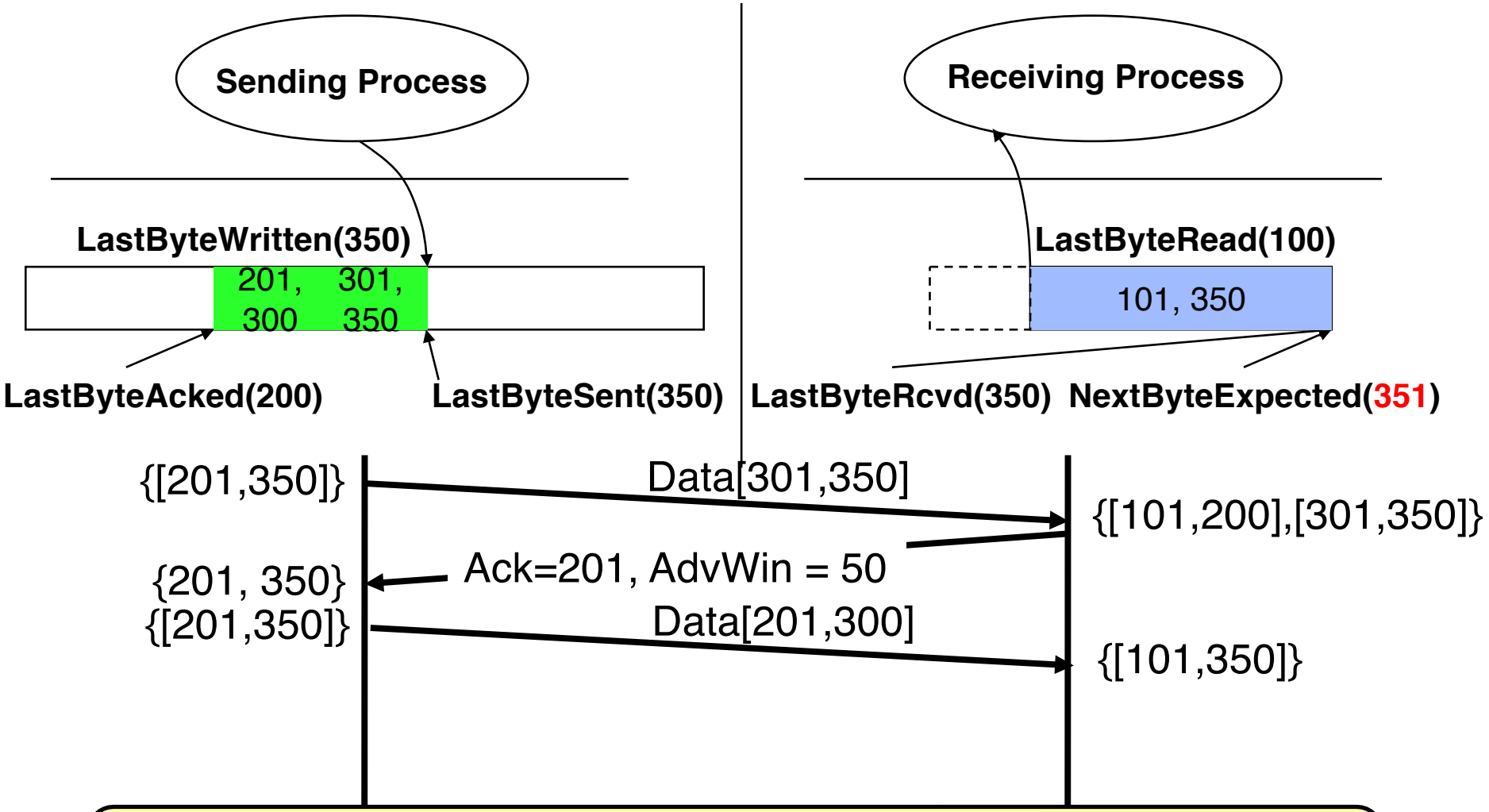
- Ack still specifies 201 (first byte out of sequence)
- AdvWin = 50, so can sender re-send 3rd packet?

TCP Flow Control



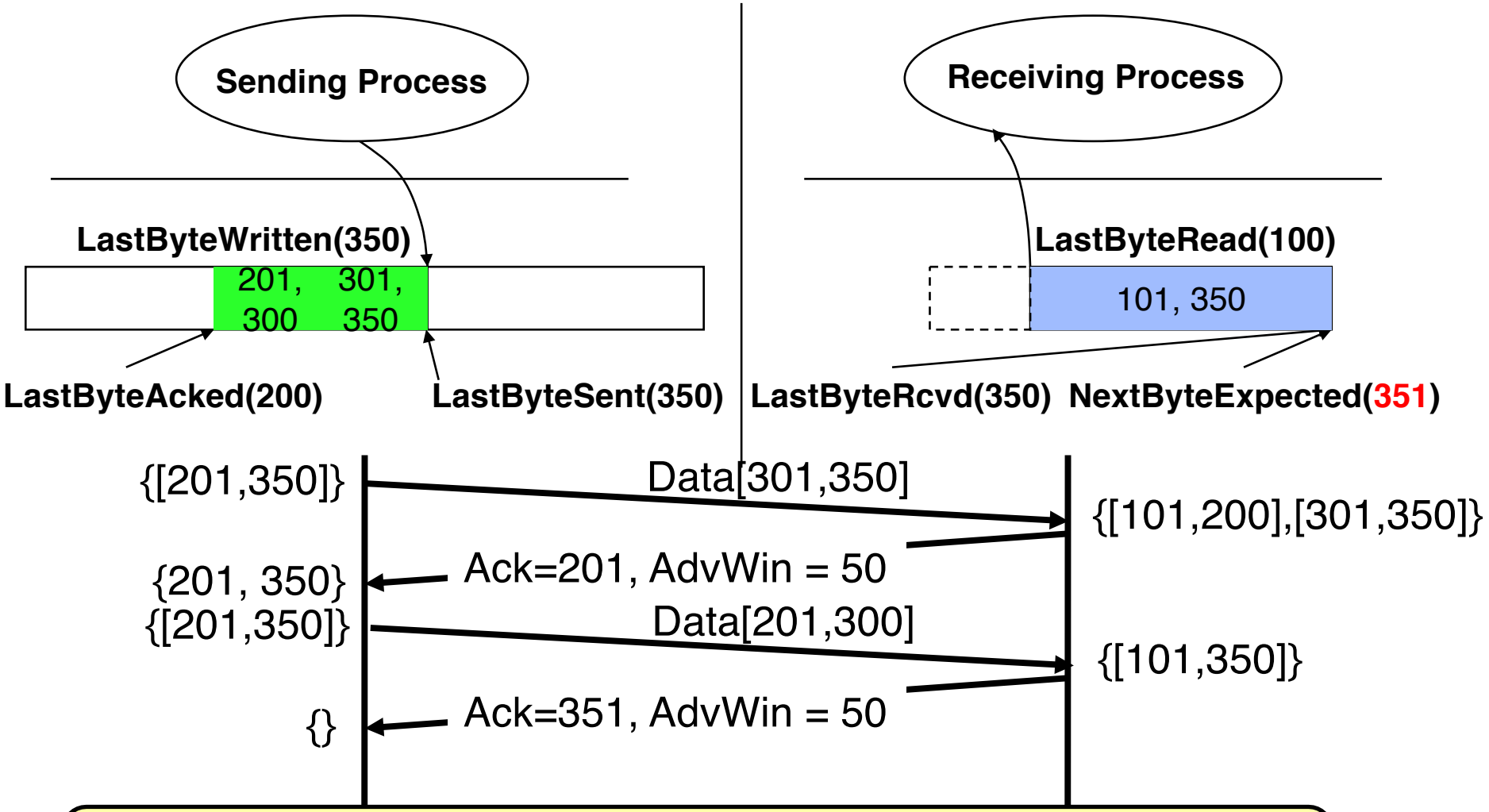
Yes! Sender can re-send 2nd packet since it's in existing window – won't cause receiver window to grow

TCP Flow Control



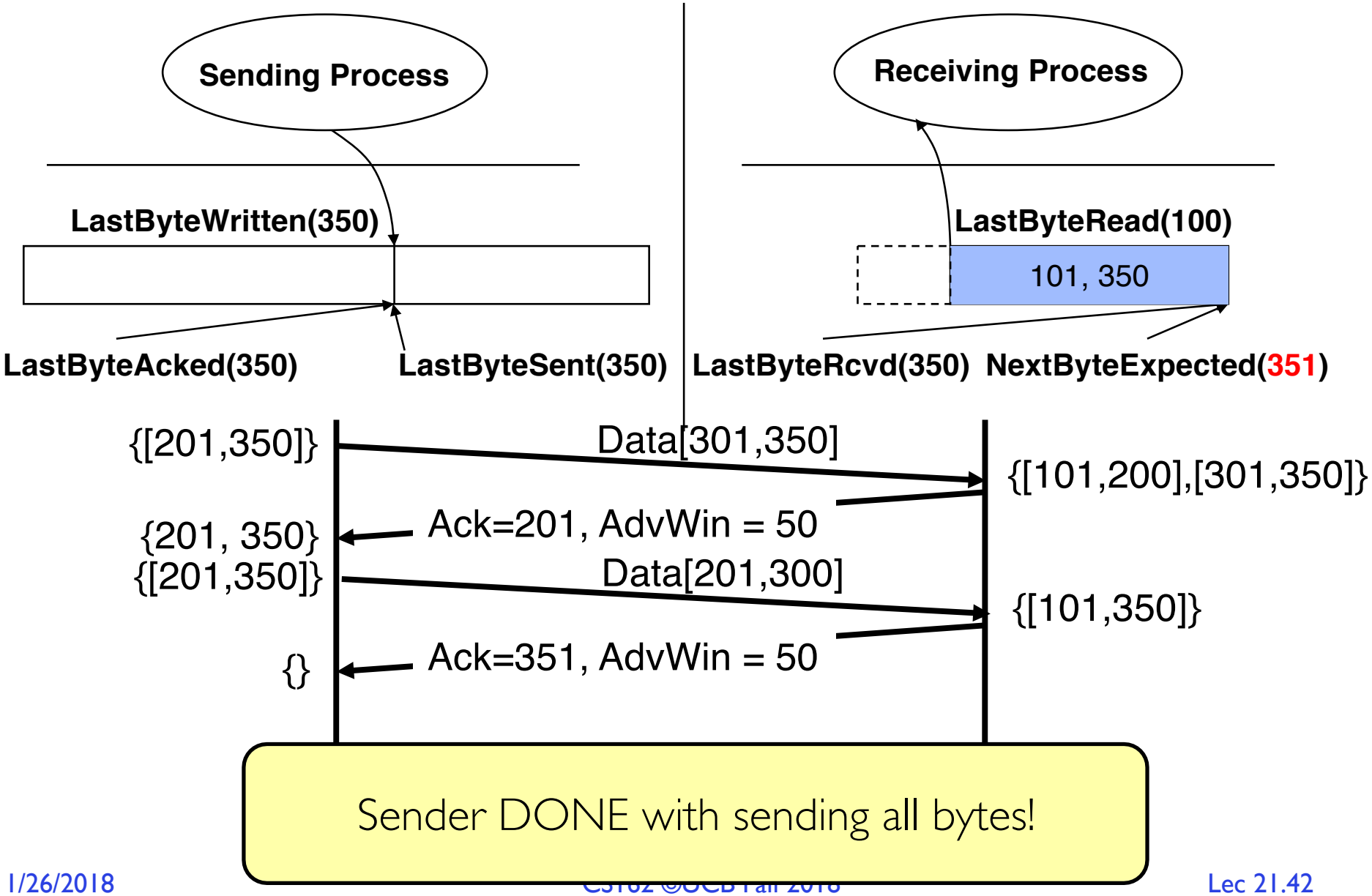
Yes! Sender can re-send 2nd packet since it's in existing window – won't cause receiver window to grow

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 (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)

Summary

- 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

THANKS, AND GOOD LUCK!