CS162
Operating Systems and Systems Programming
Lecture 18
TCP's Flow Control, Transactions

April 2, 2012

Anthony D. Joseph and Ion Stoica

http://inst.eecs.berkeley.edu/~cs162

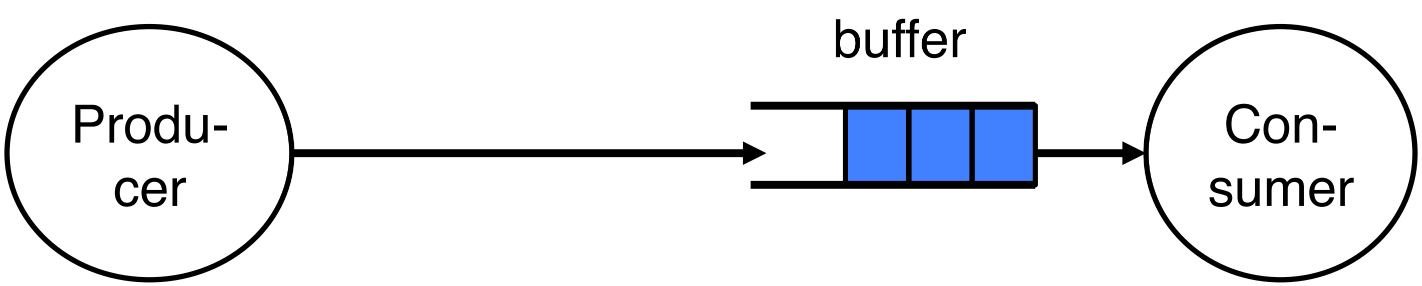
Goals of Today's Lecture

- TCP flow control
- Transactions (ACID semantics)

Note: Some slides and/or pictures in the following are adapted from lecture notes by Mike Franklin.

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



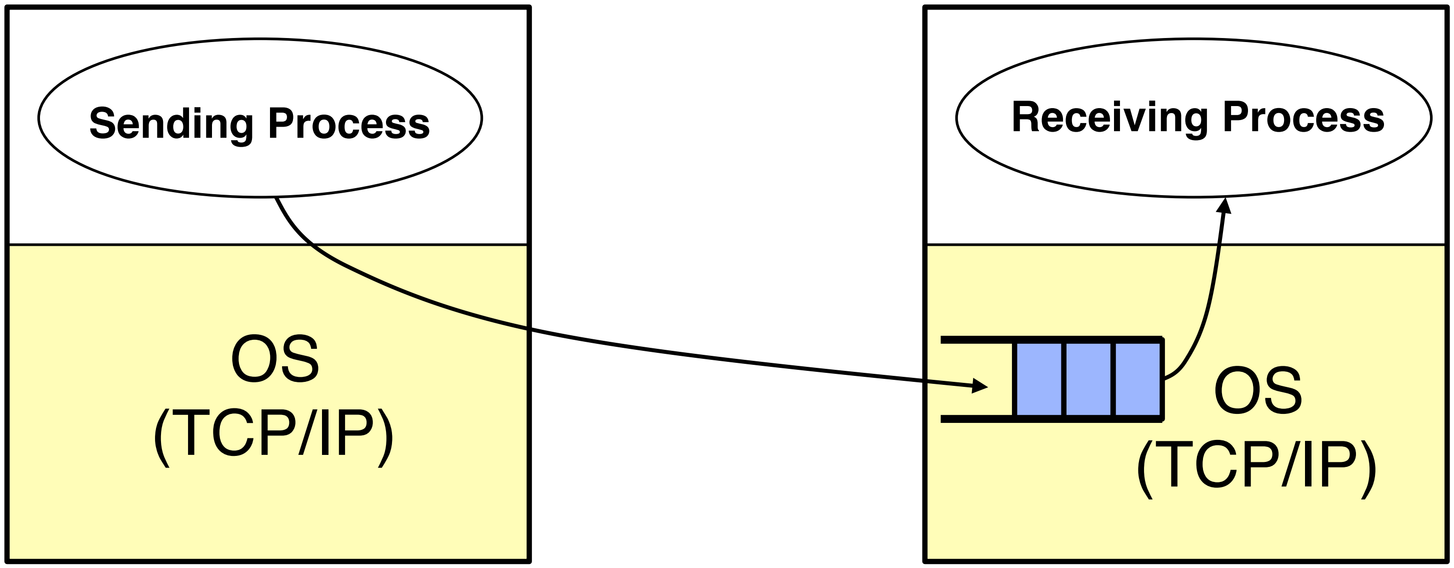
TCP Flow Control

- 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)

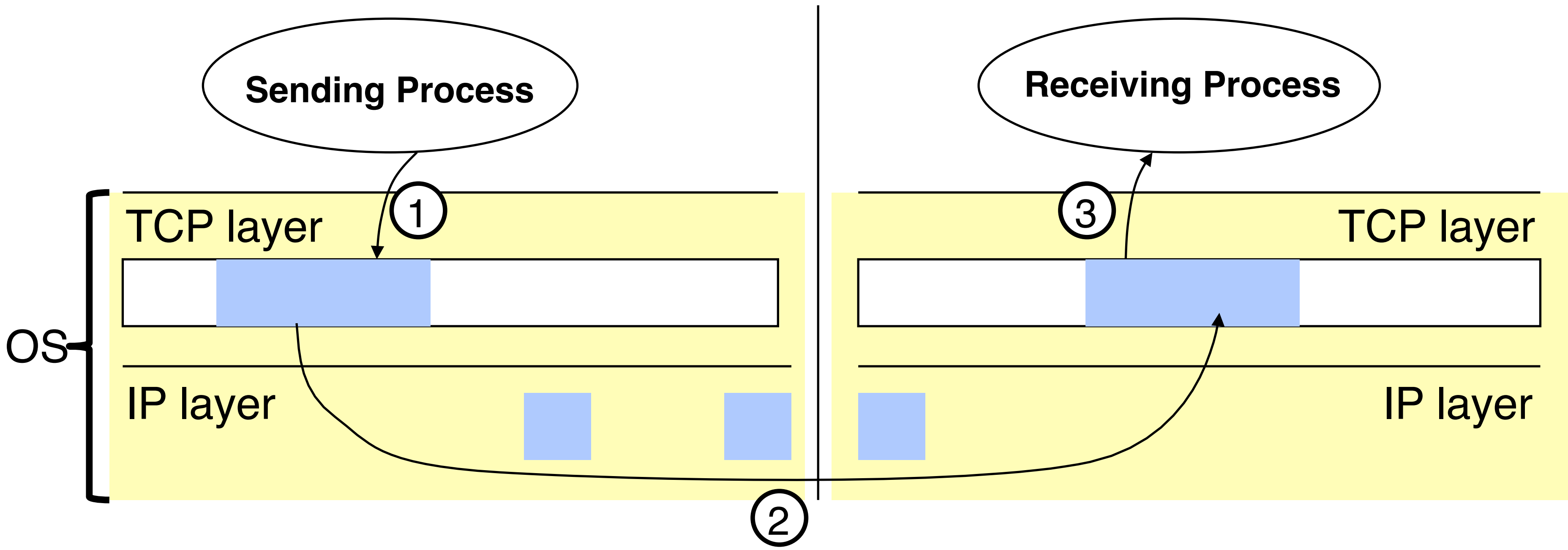
- The ack(nowledgement) contains sequence number N of next byte the receiver expects, i.e., receiver has received all bytes in sequence up to and including N-1

TCP Flow Control



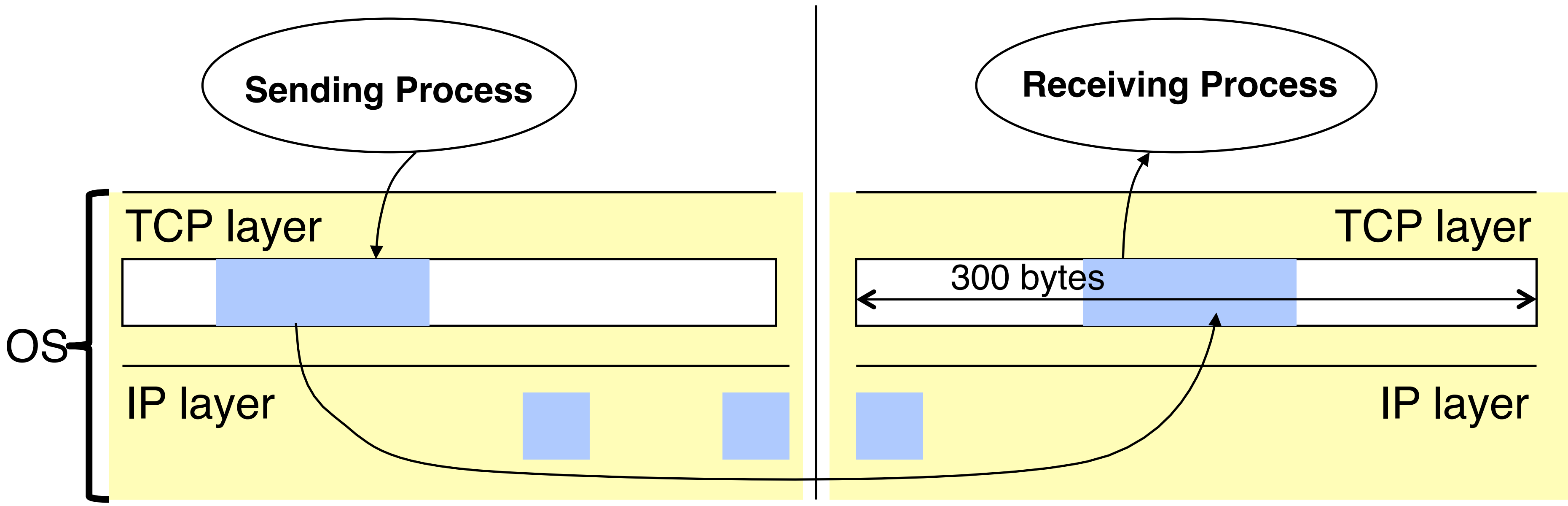
- TCP/IP implemented by OS (Kernel)
  - – Cannot do context switching on sending/receiving every packet
    - » At 1Gbps, it takes 12 usec to send an 1500 bytes, and 0.8usec to send an 100 byte packet
- Need buffers to match …
  - – sending app with sending TCP
  - – receiving TCP with receiving app

TCP Flow Control



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

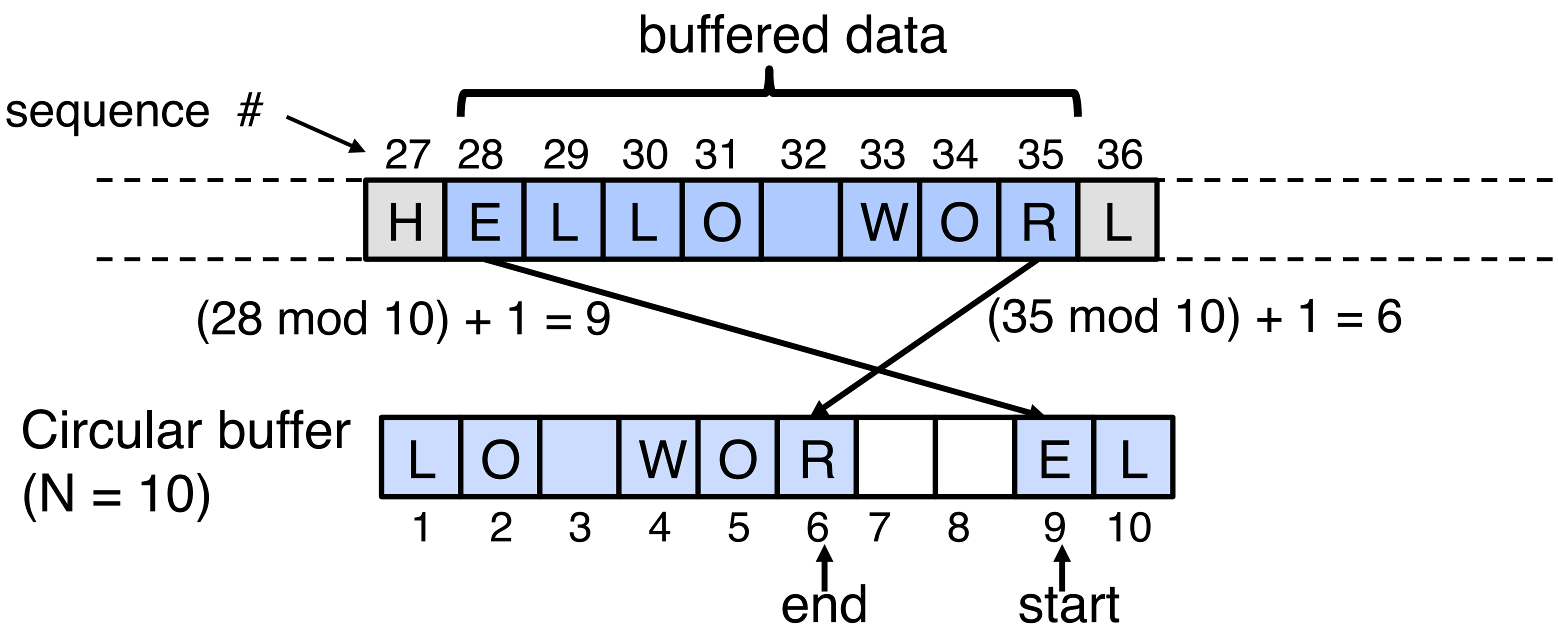
TCP Flow Control



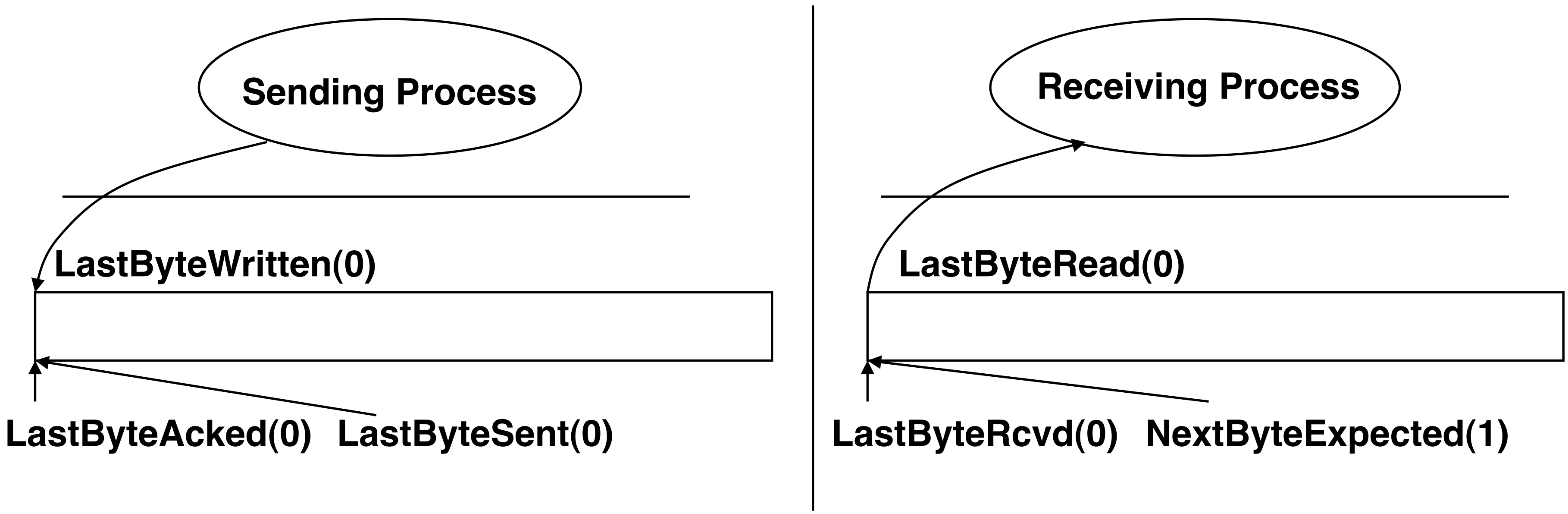
- Example assumptions:
  - Maximum IP packet size = 100 bytes
  - Size of the receiving buffer (MaxRcvBuf) = 300 bytes
- Recall, ack indicates the next expected byte in-sequence, not the last received byte
- Use circular buffers

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

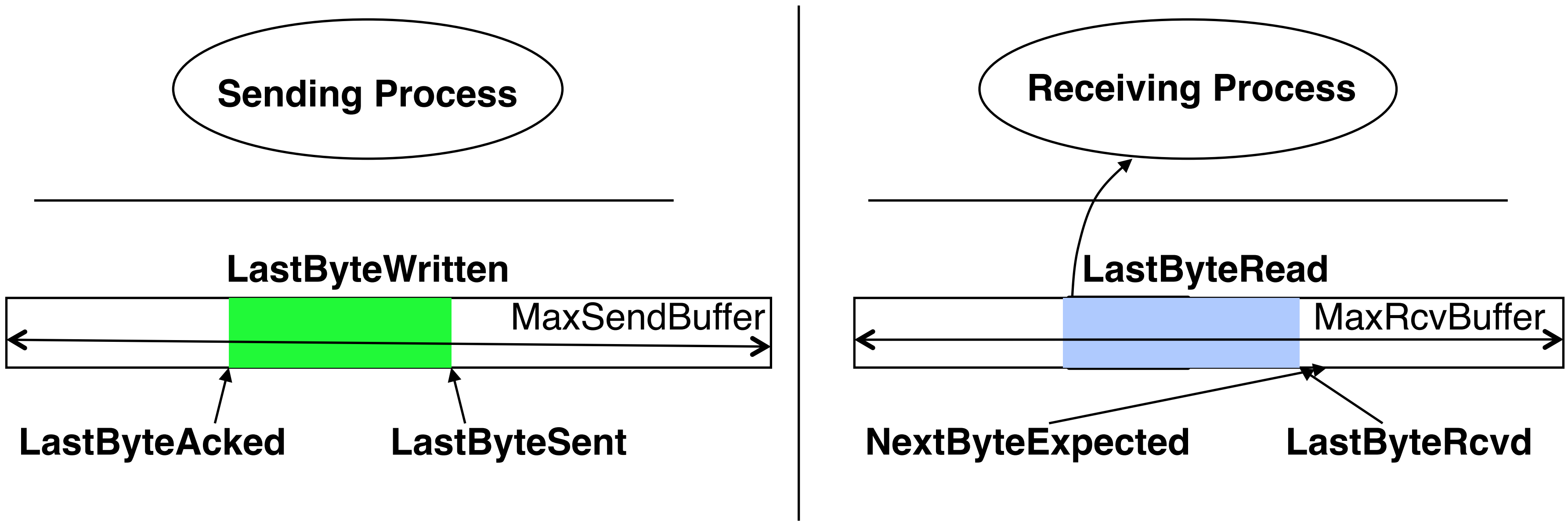


TCP Flow Control



- 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



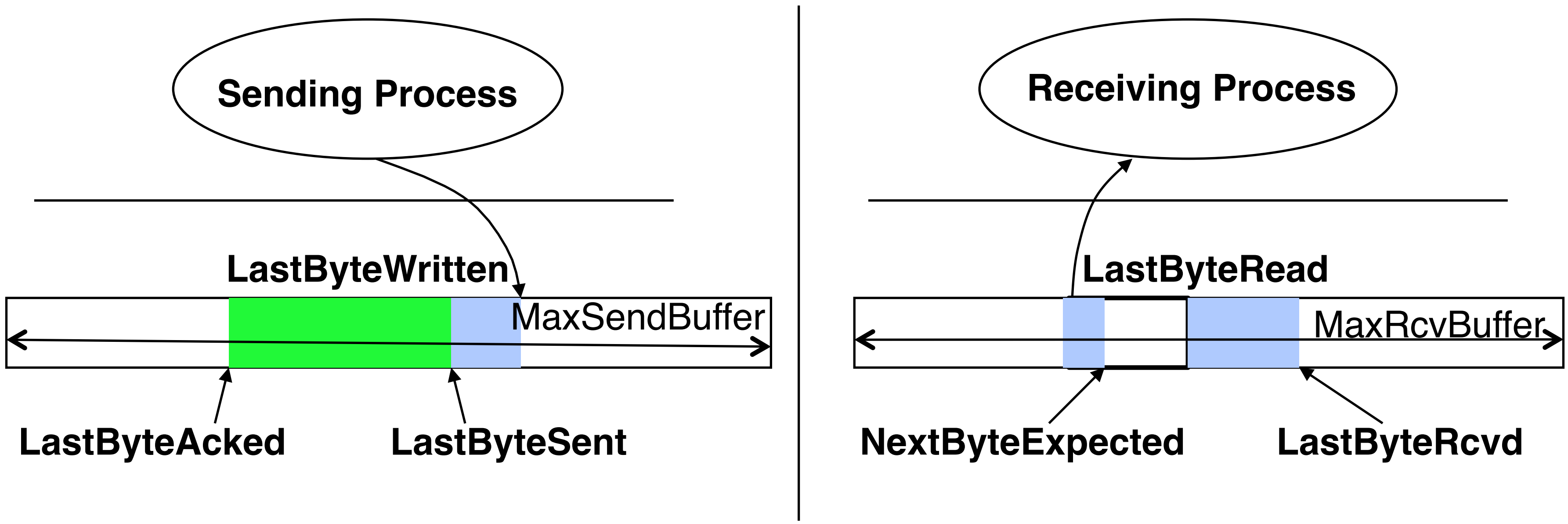
- AdvertisedWindow: number of bytes TCP receiver can receive

AdvertisedWindow = MaxRcvBuffer – (LastByteRcvd – LastByteRead)

- SenderWindow: number of bytes TCP sender can send

SenderWindow = AdvertisedWindow – (LastByteSent – LastByteAcked)

TCP Flow Control



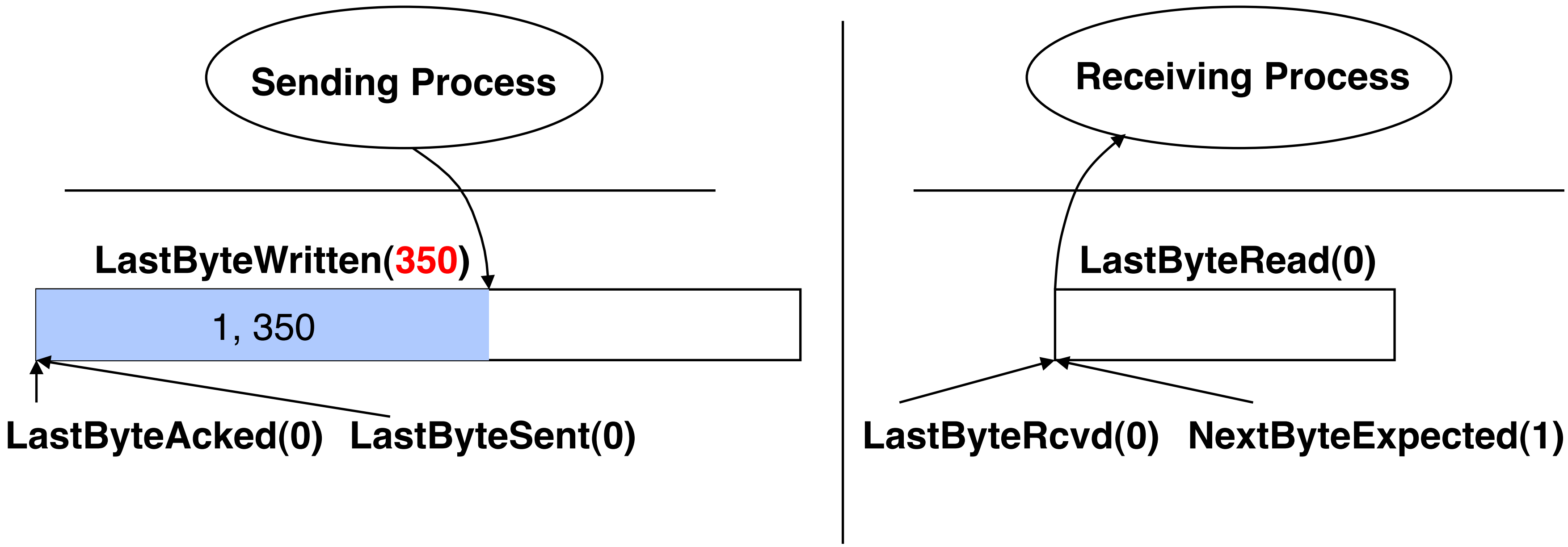
- Still true if receiver missed data….

AdvertisedWindow = MaxRcvBuffer – (LastByteRcvd – LastByteRead)

- WriteWindow: number of bytes sending process can write

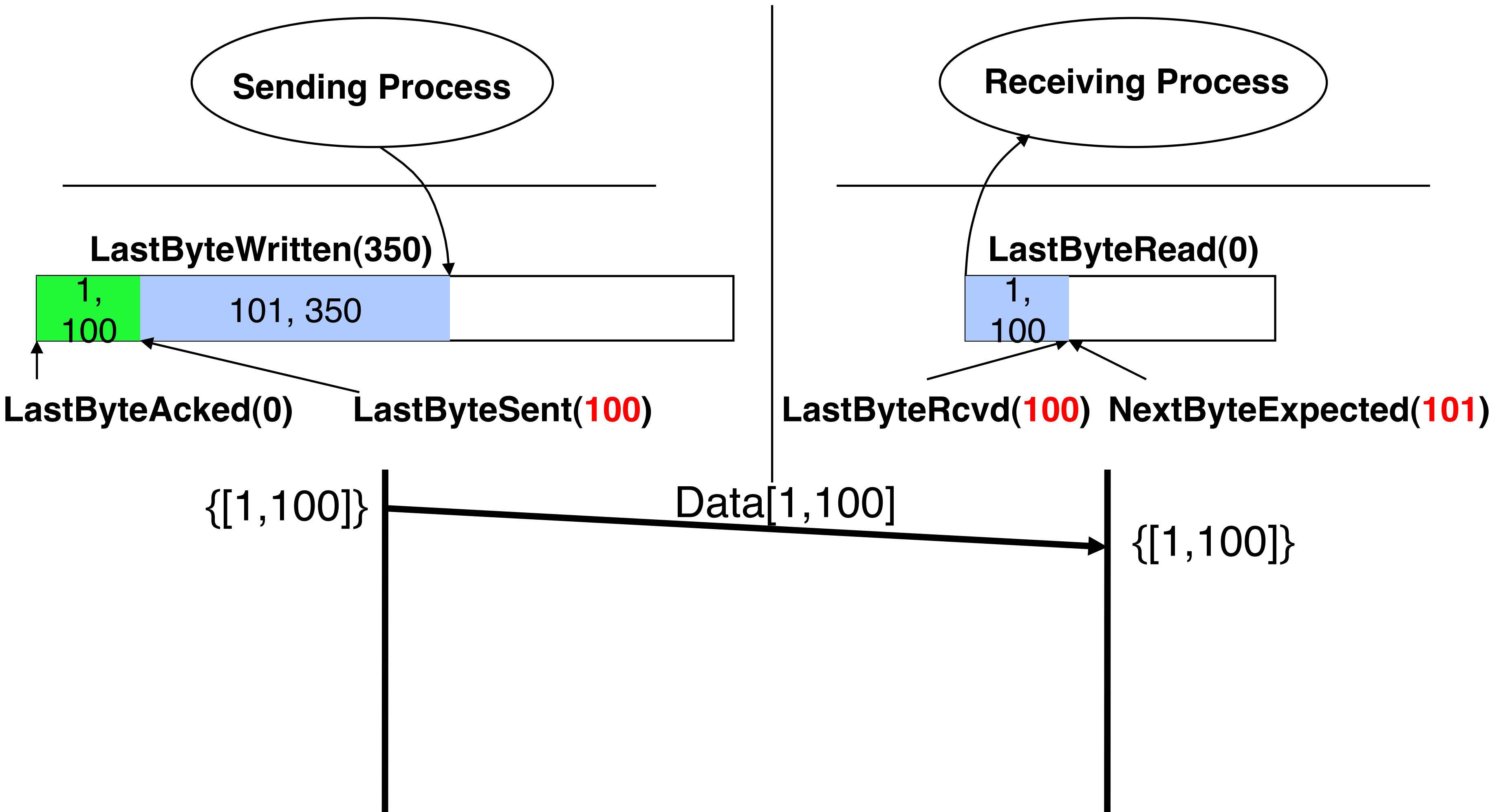
WriteWindow = MaxSendBuffer – (LastByteWritten – LastByteAcked)

TCP Flow Control



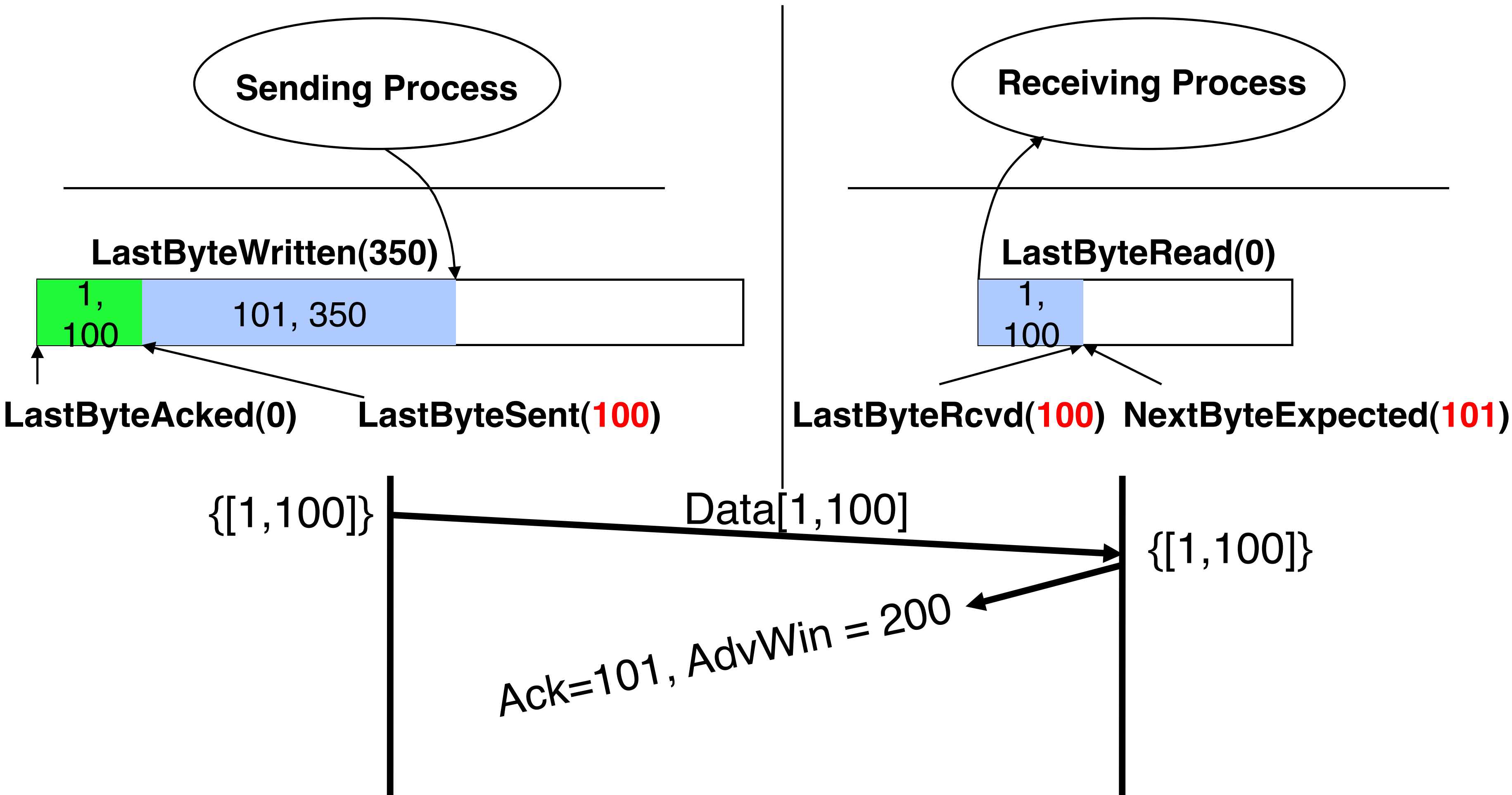
- Sending app sends 350 bytes
- Recall:
  - We assume IP only accepts packets no larger than 100 bytes
  - MaxRcvBuf = 300 bytes, so initial Advertised Window = 300 byets

TCP Flow Control



Sender sends first packet (i.e., first 100 bytes) and receiver gets the packet

TCP Flow Control

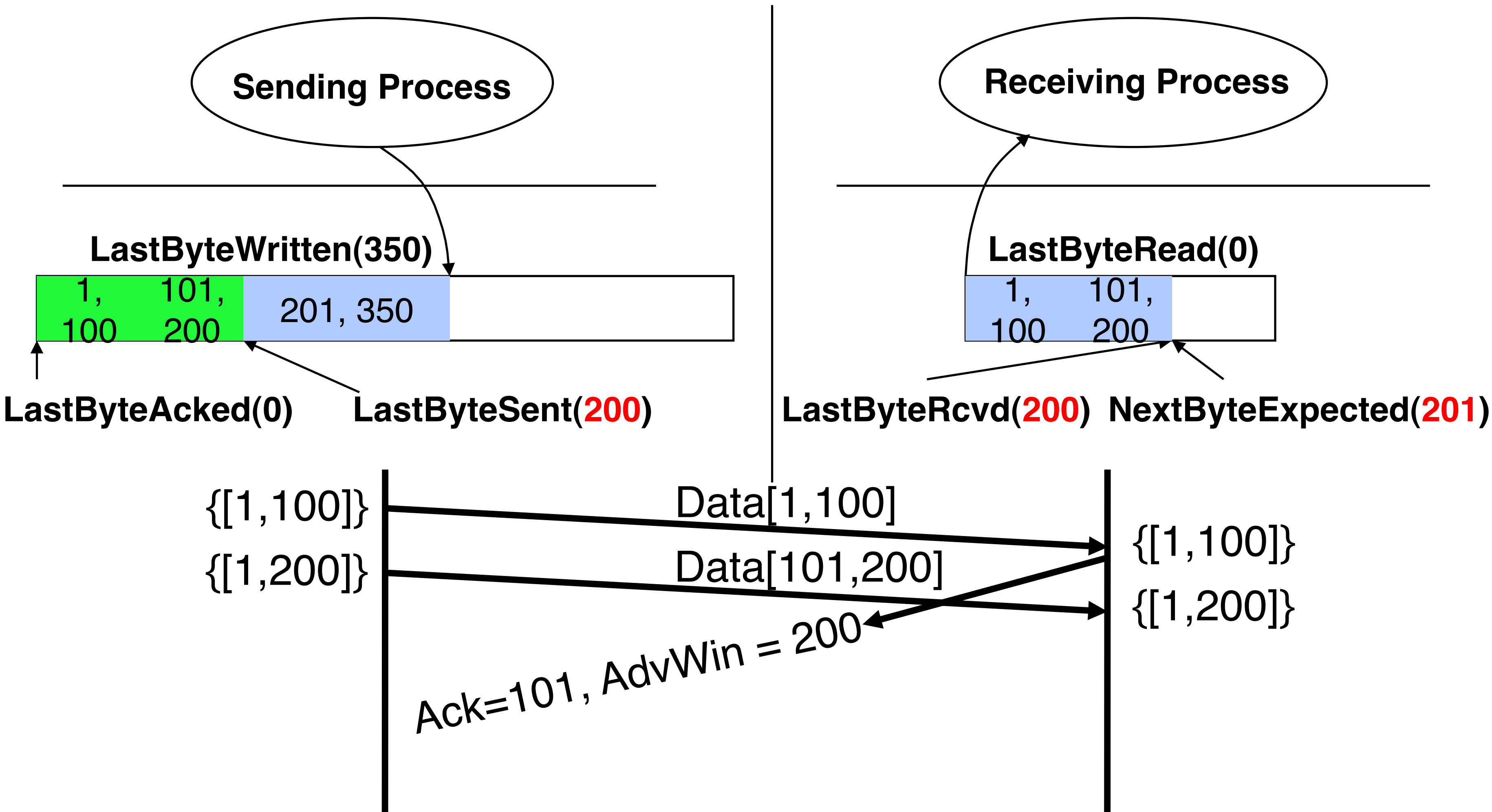


Receiver sends ack for 1st packet

AdvWin = MaxRcvBuffer – (LastByteRcvd – LastByteRead)

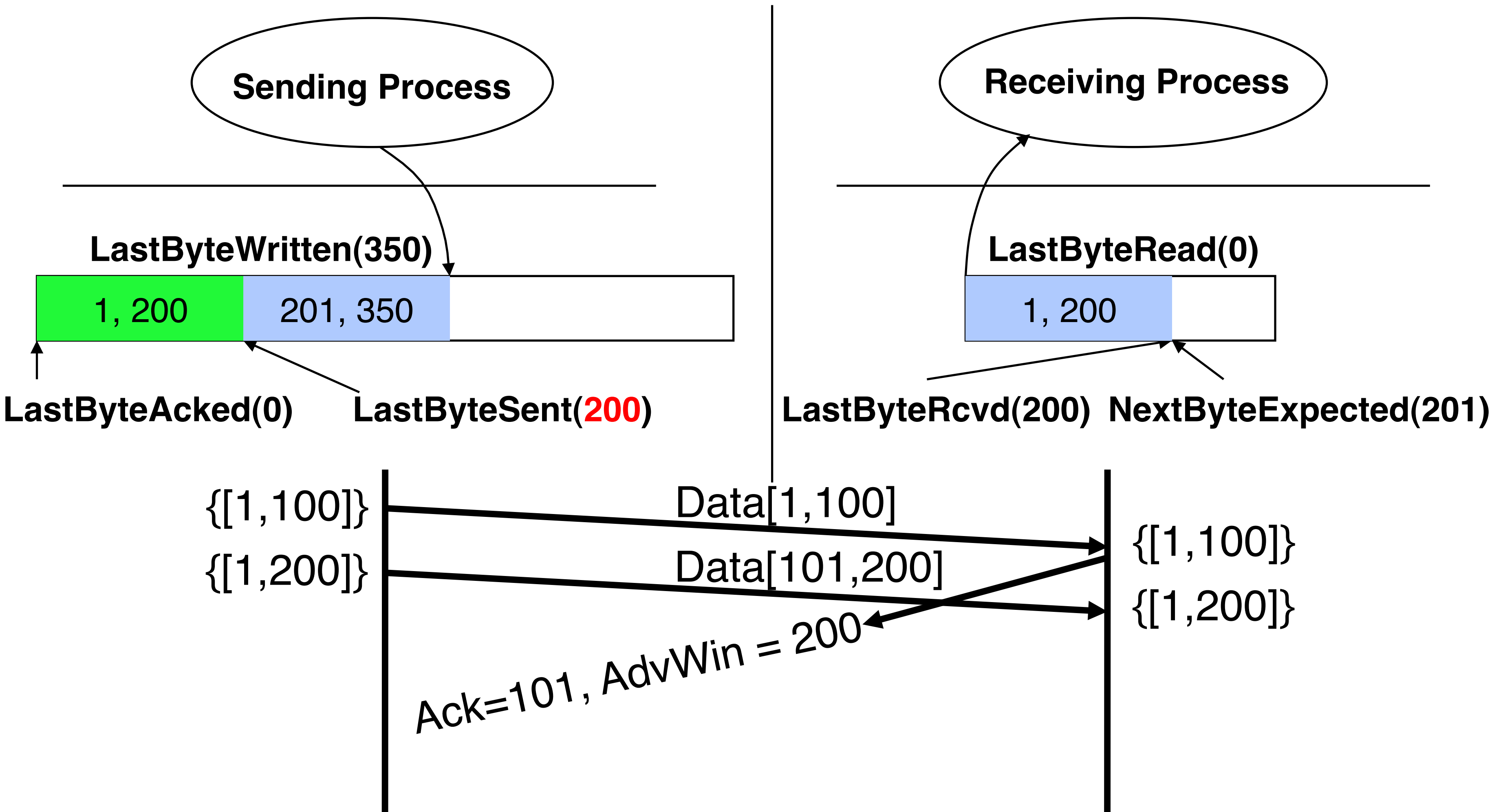
= 300 – (100 – 0) = 200

TCP Flow Control



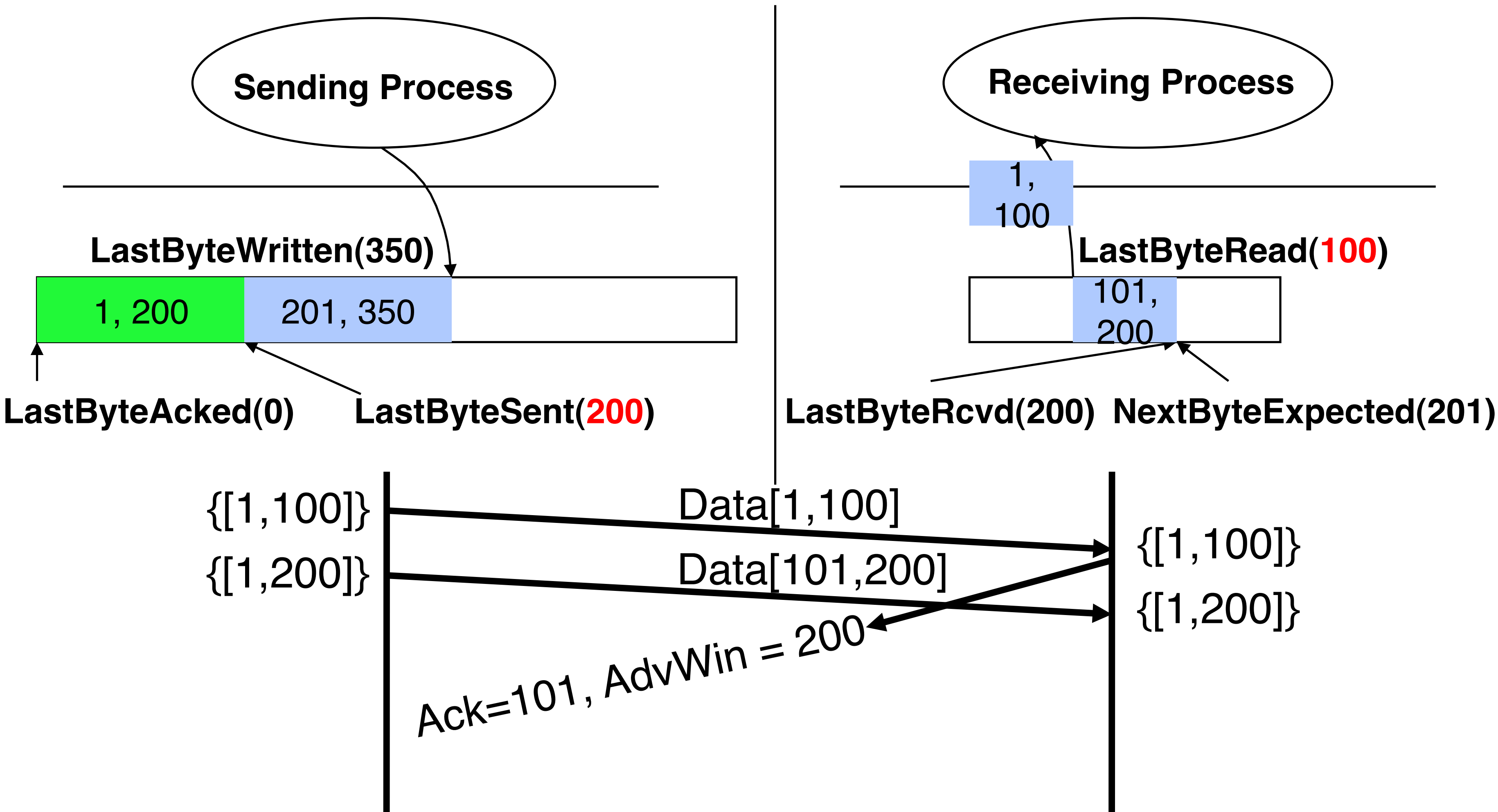
Sender sends 2nd packet (i.e., next 100 bytes) and receiver gets the packet

TCP Flow Control



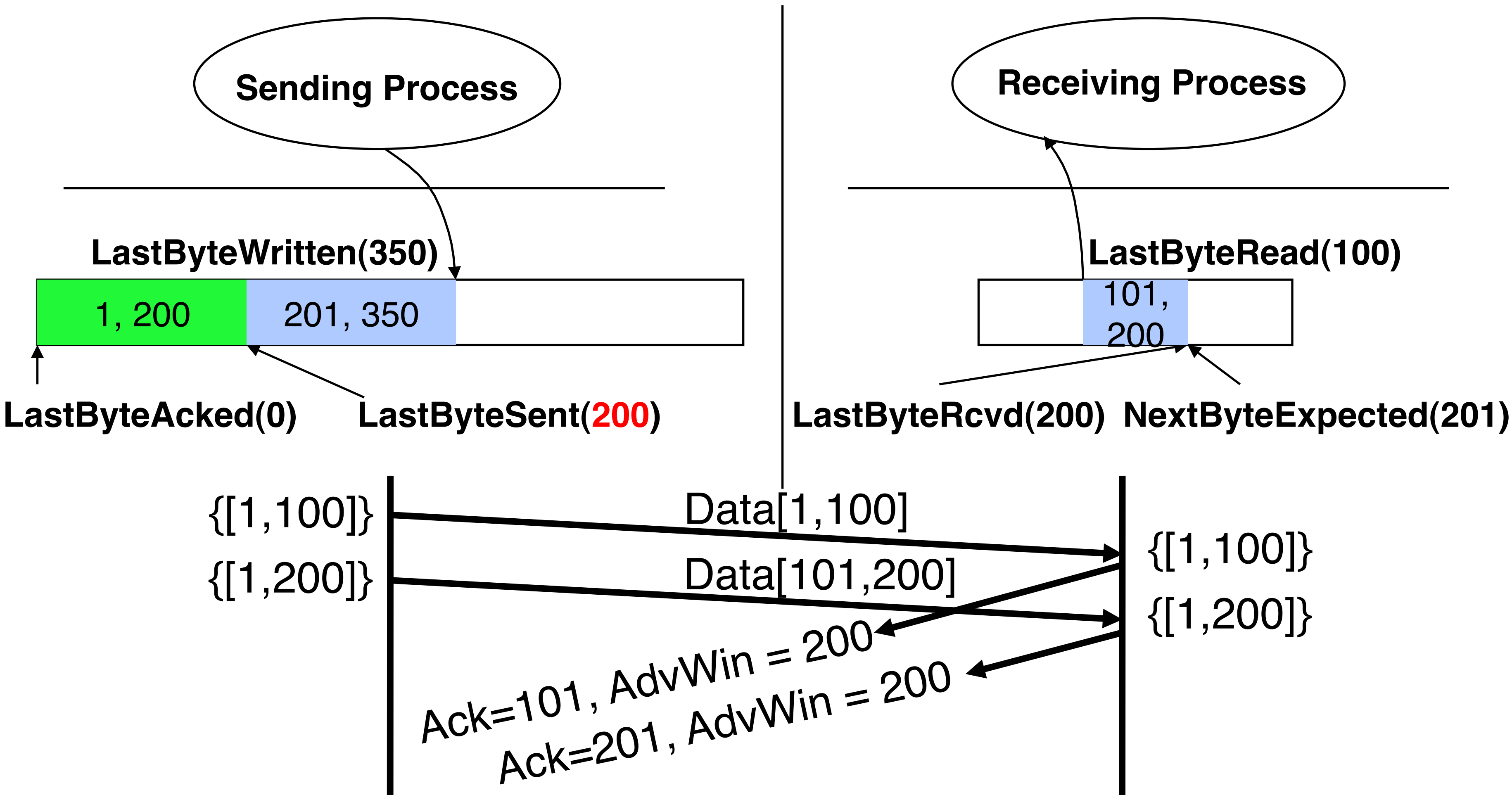
Sender sends 2nd packet (i.e., next 100 bytes) and receiver gets the packet

TCP Flow Control



Receiving TCP delivers first 100 bytes to recienving process

TCP Flow Control

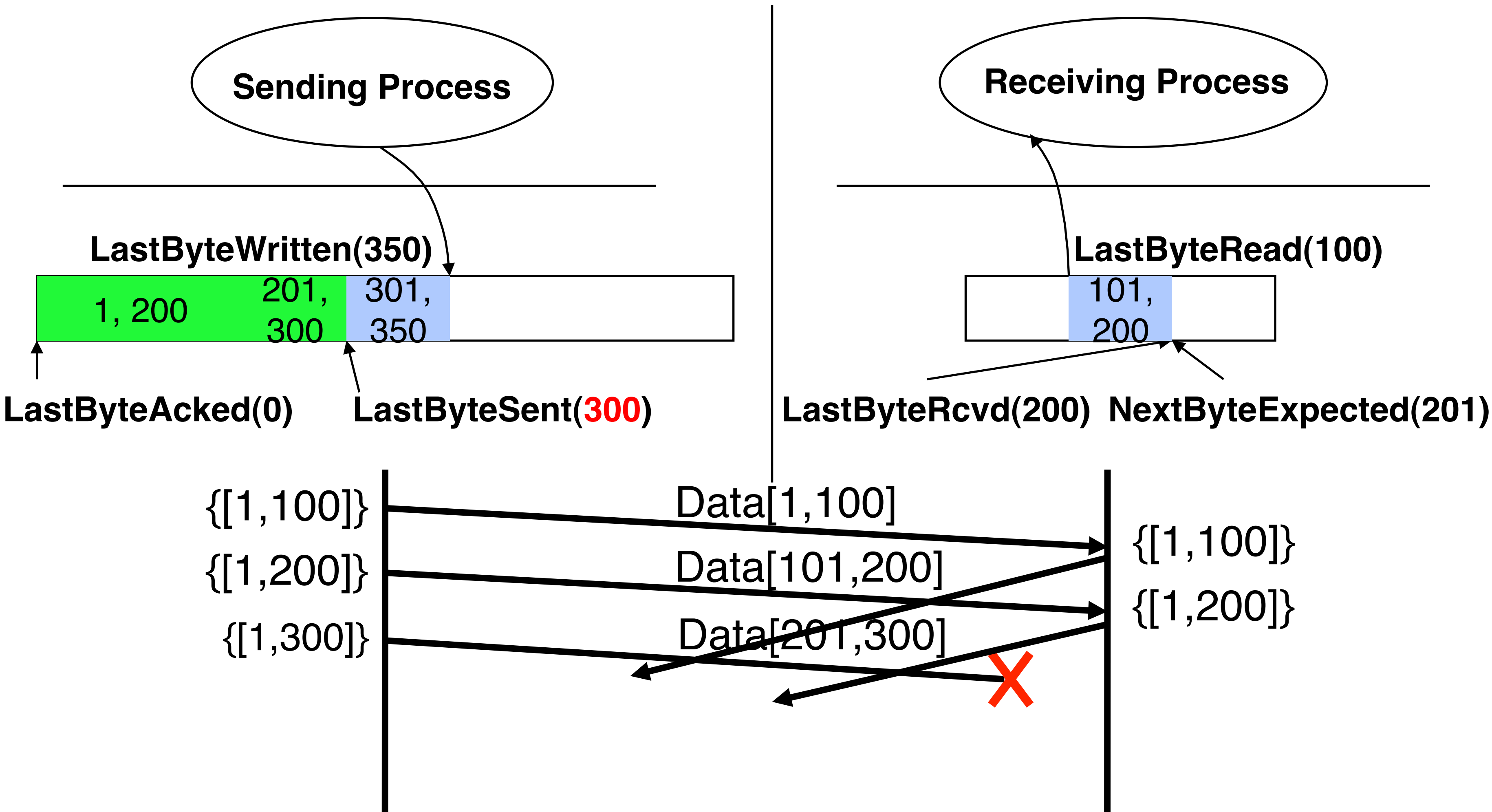


Receiver sends ack for 2nd packet

AdvWin = MaxRcvBuffer – (LastByteRcvd – LastByteRead)

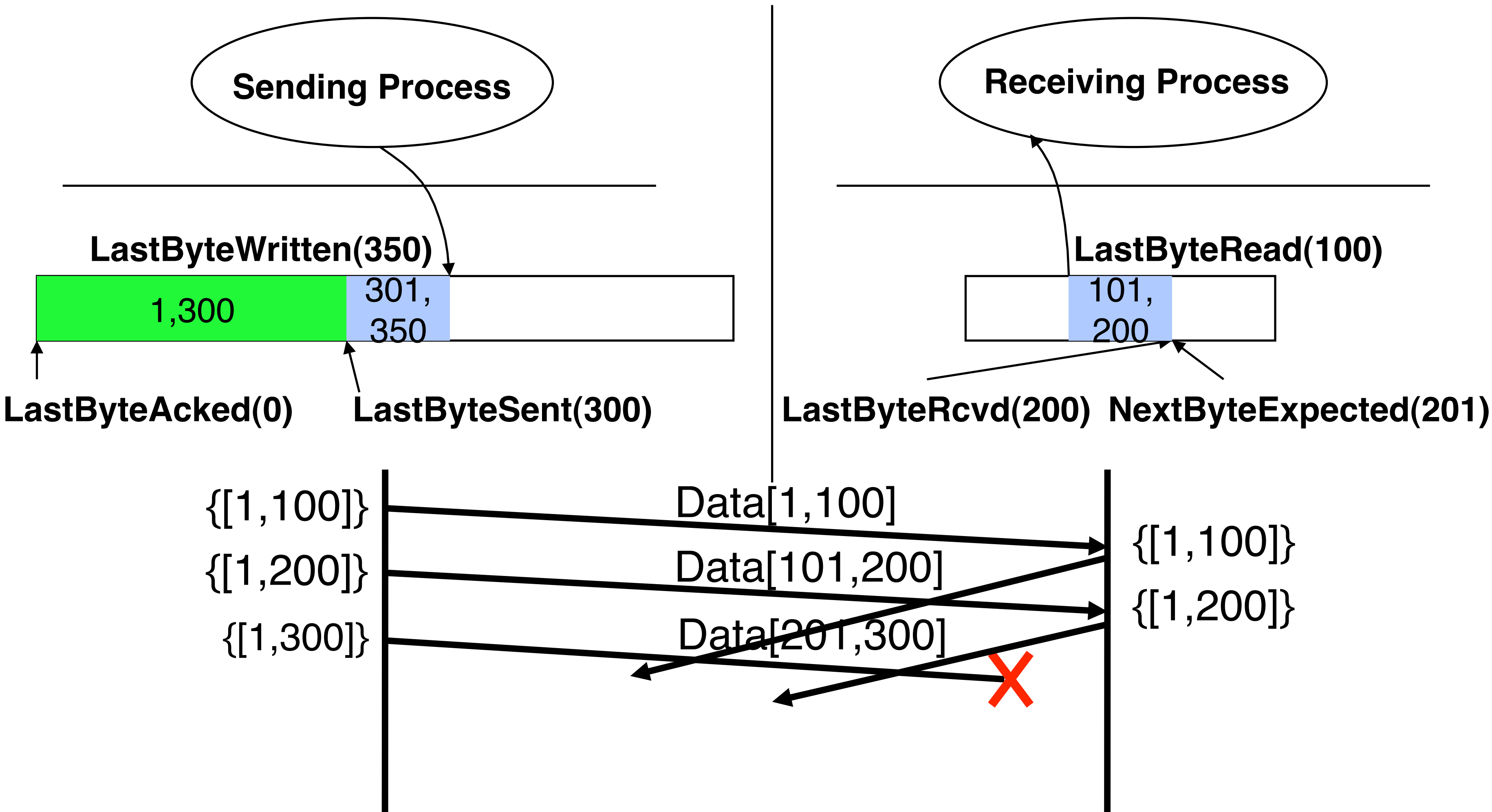
= 300 – (200 – 100) = 200

TCP Flow Control



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

TCP Flow Control

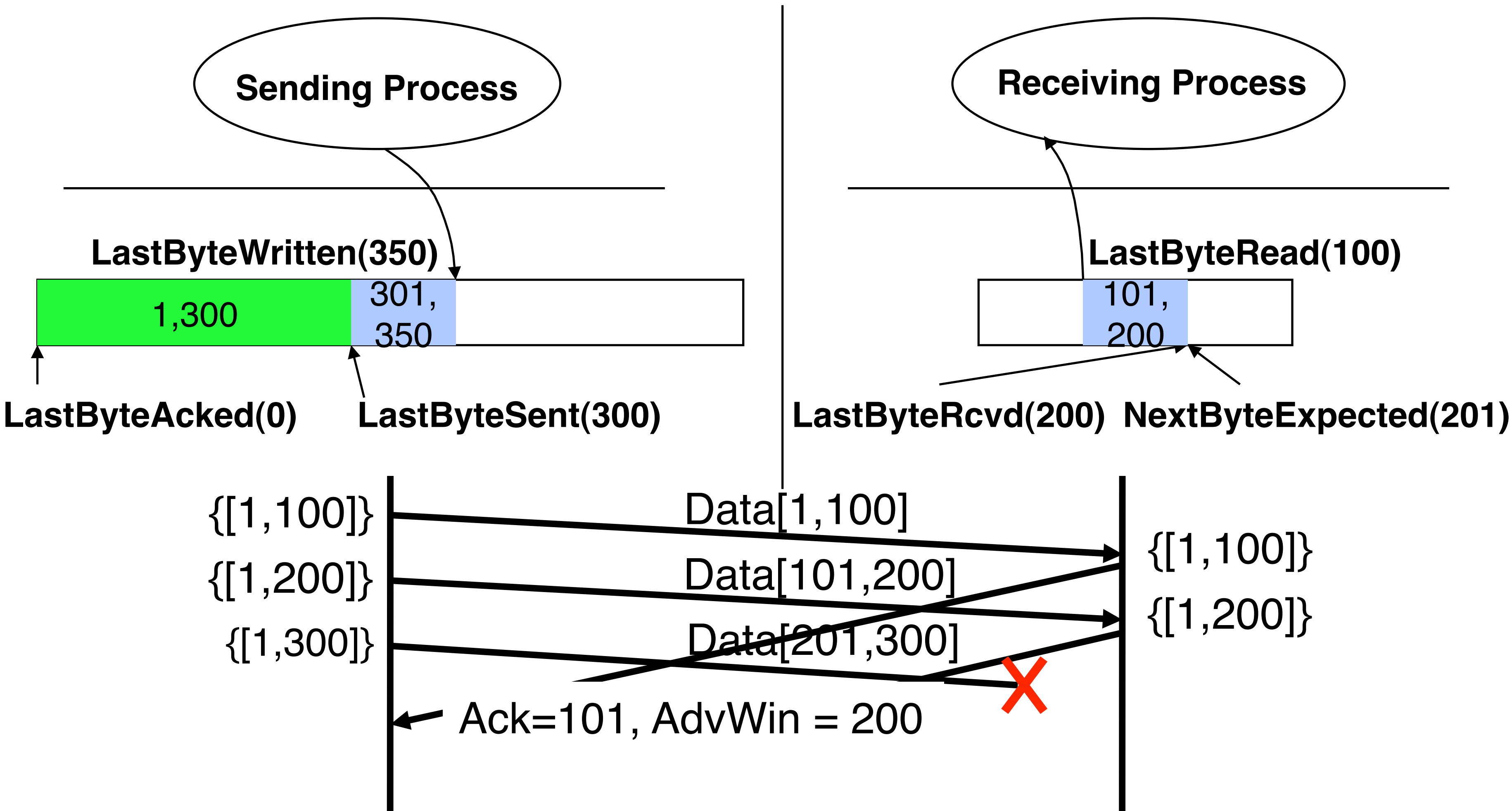


Sender stops sending as window full

SndWin = AdvWin – (LastByteSent – LastByteAcked)

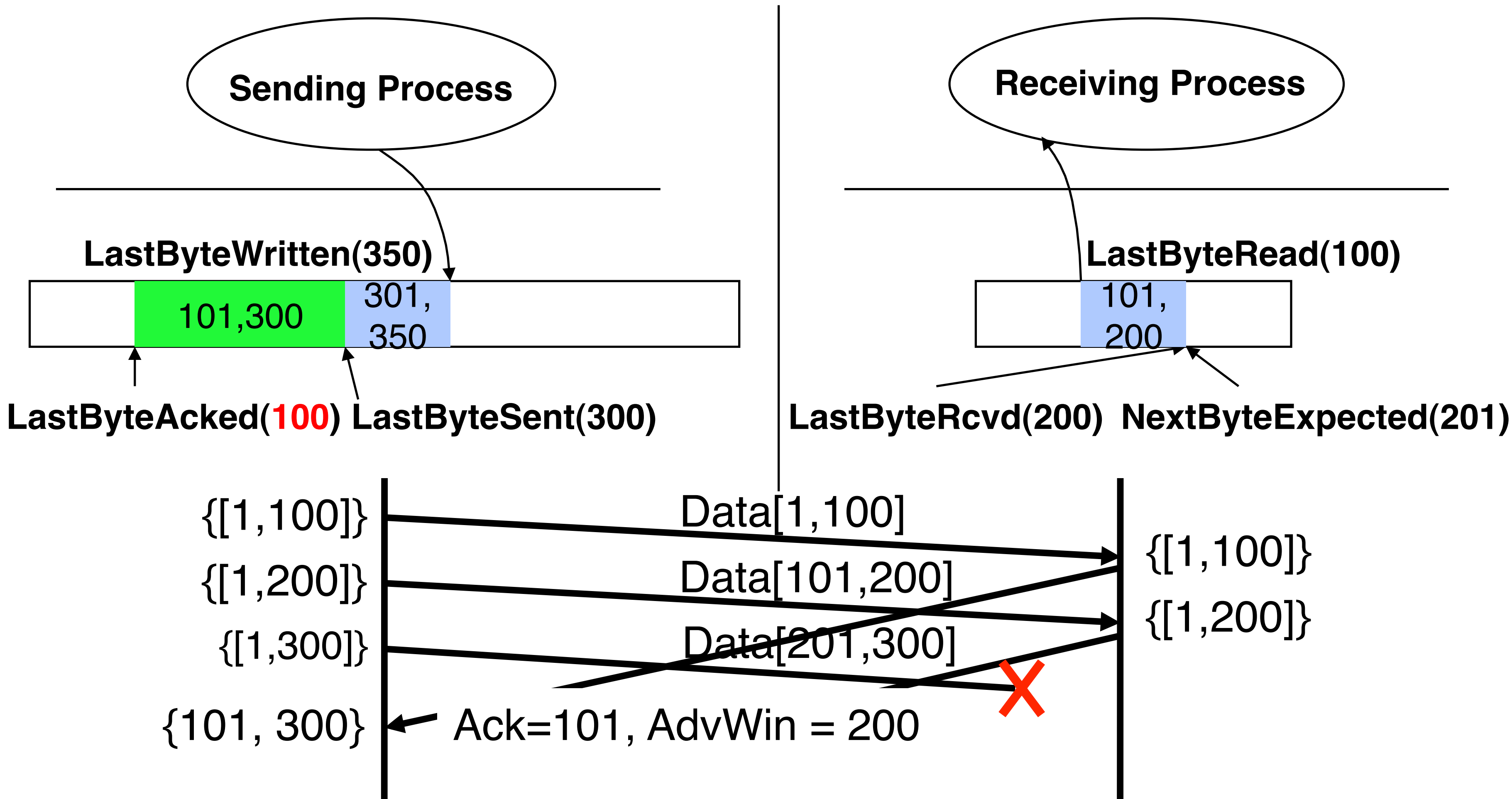
= 300 – (300 – 0) = 0

TCP Flow Control



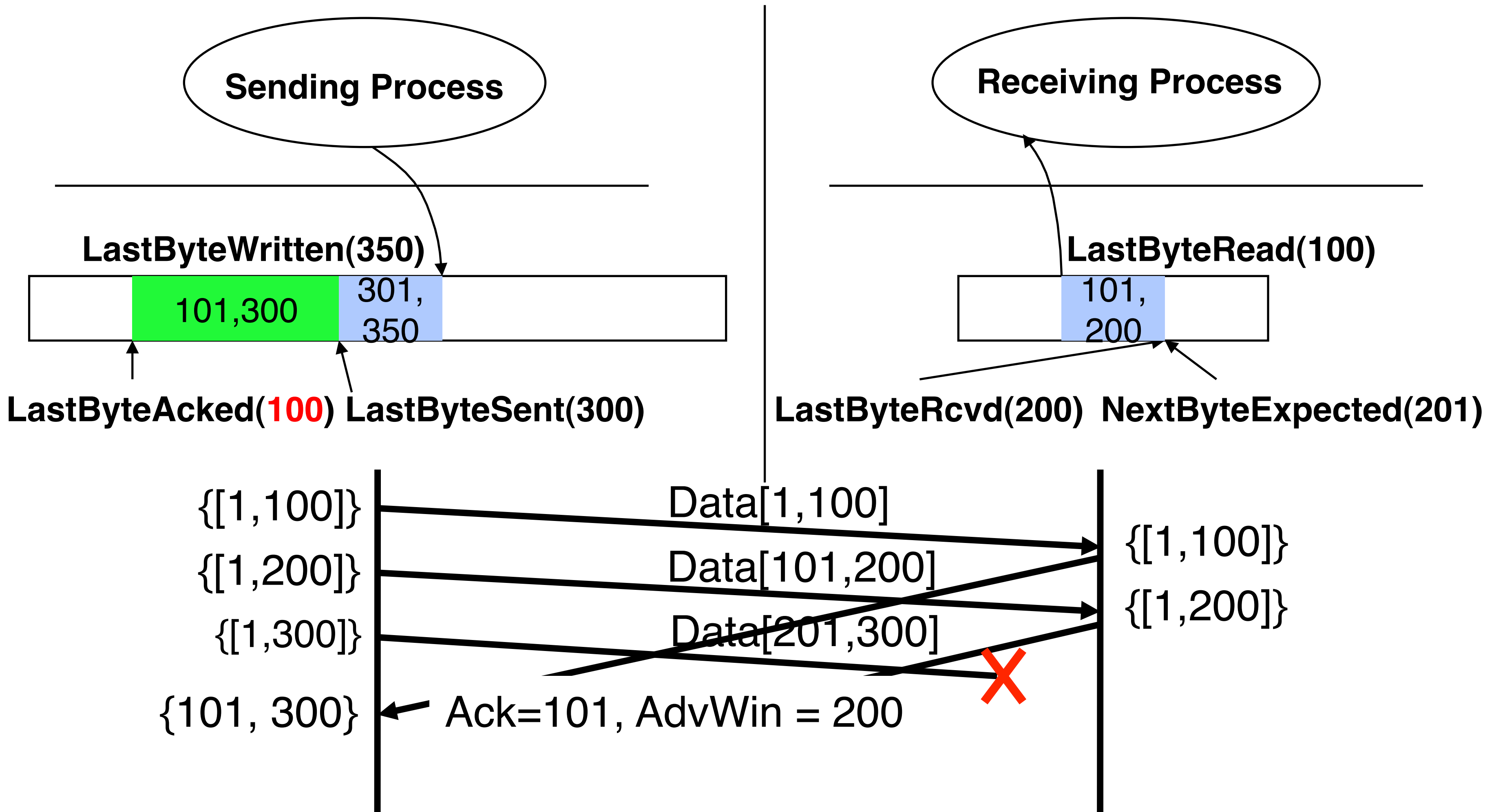
- Sender gets ack for 1st packet
- AdWin = 200

TCP Flow Control



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

TCP Flow Control

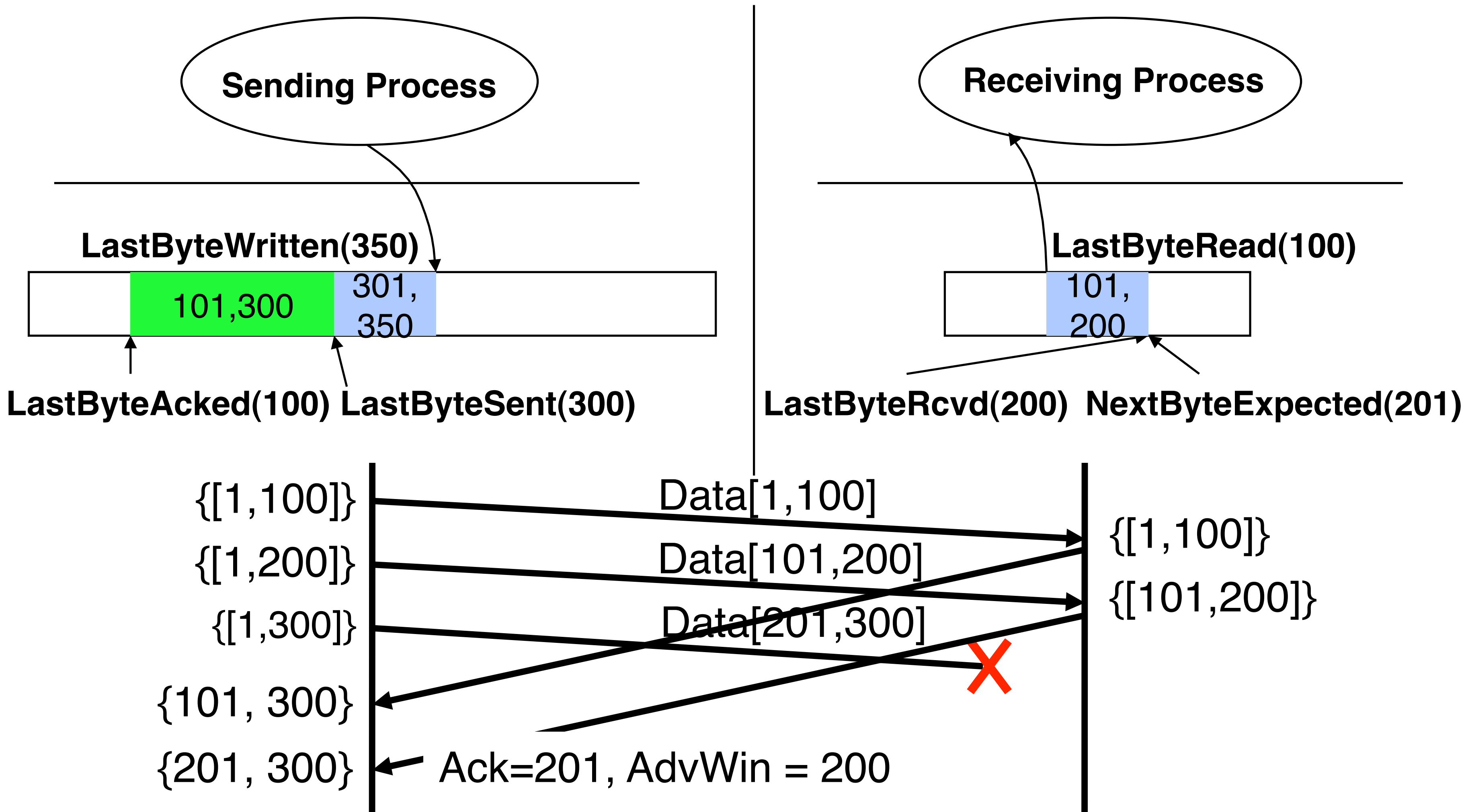


Sender still cannot send as window full

SndWin = AdvWin – (LastByteSent – LastByteAcked)

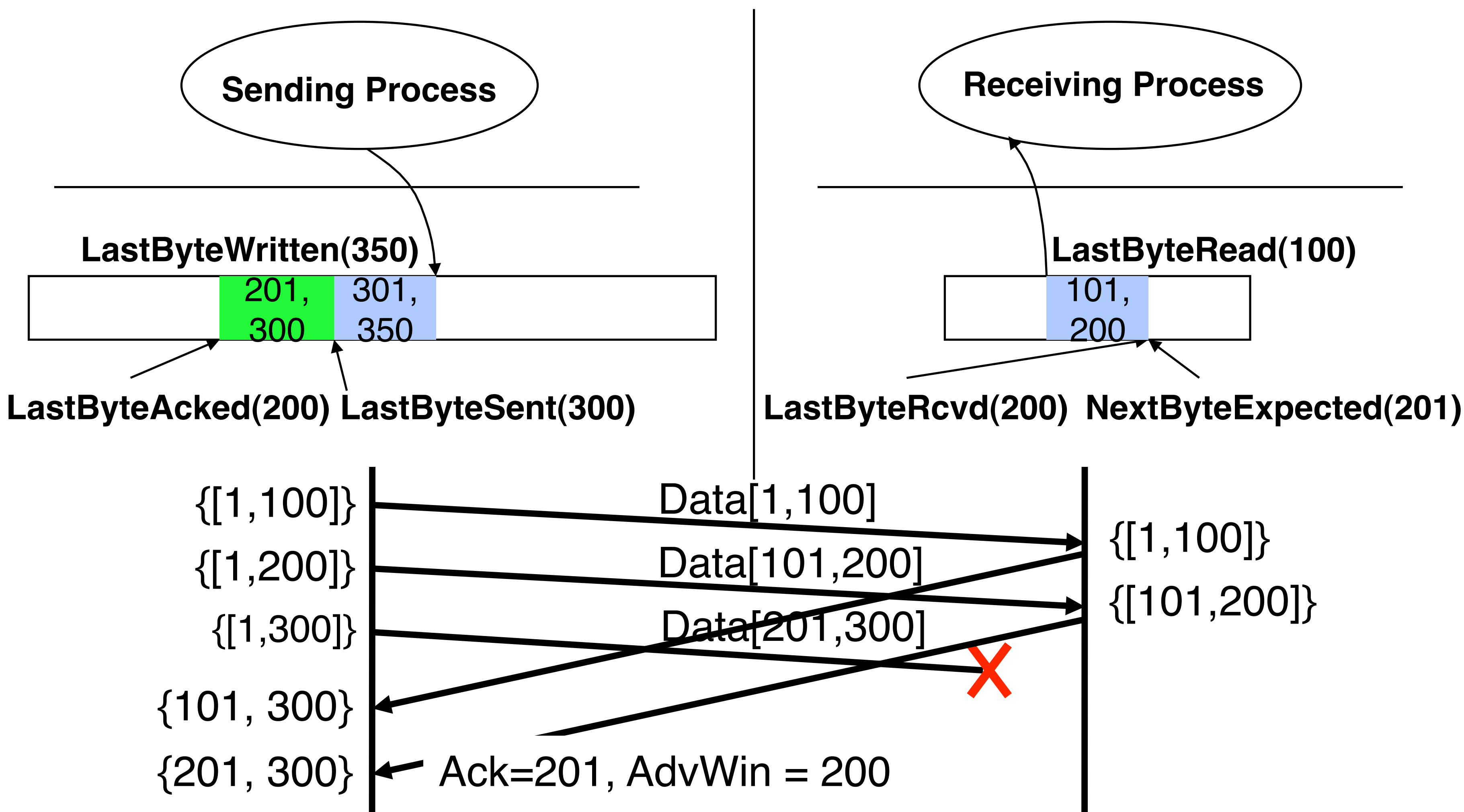
= 200 – (300 – 100) = 0

TCP Flow Control



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

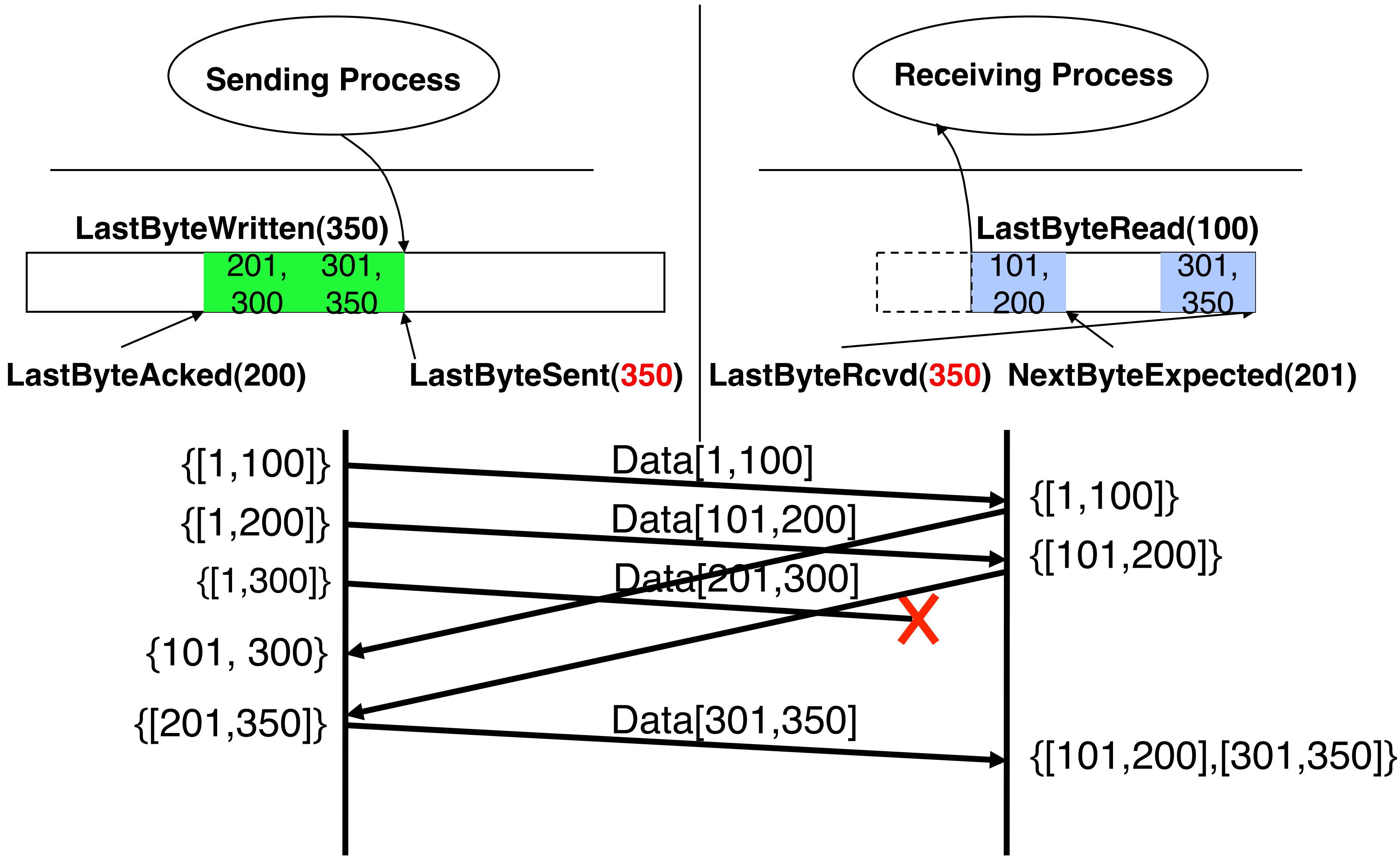
TCP Flow Control



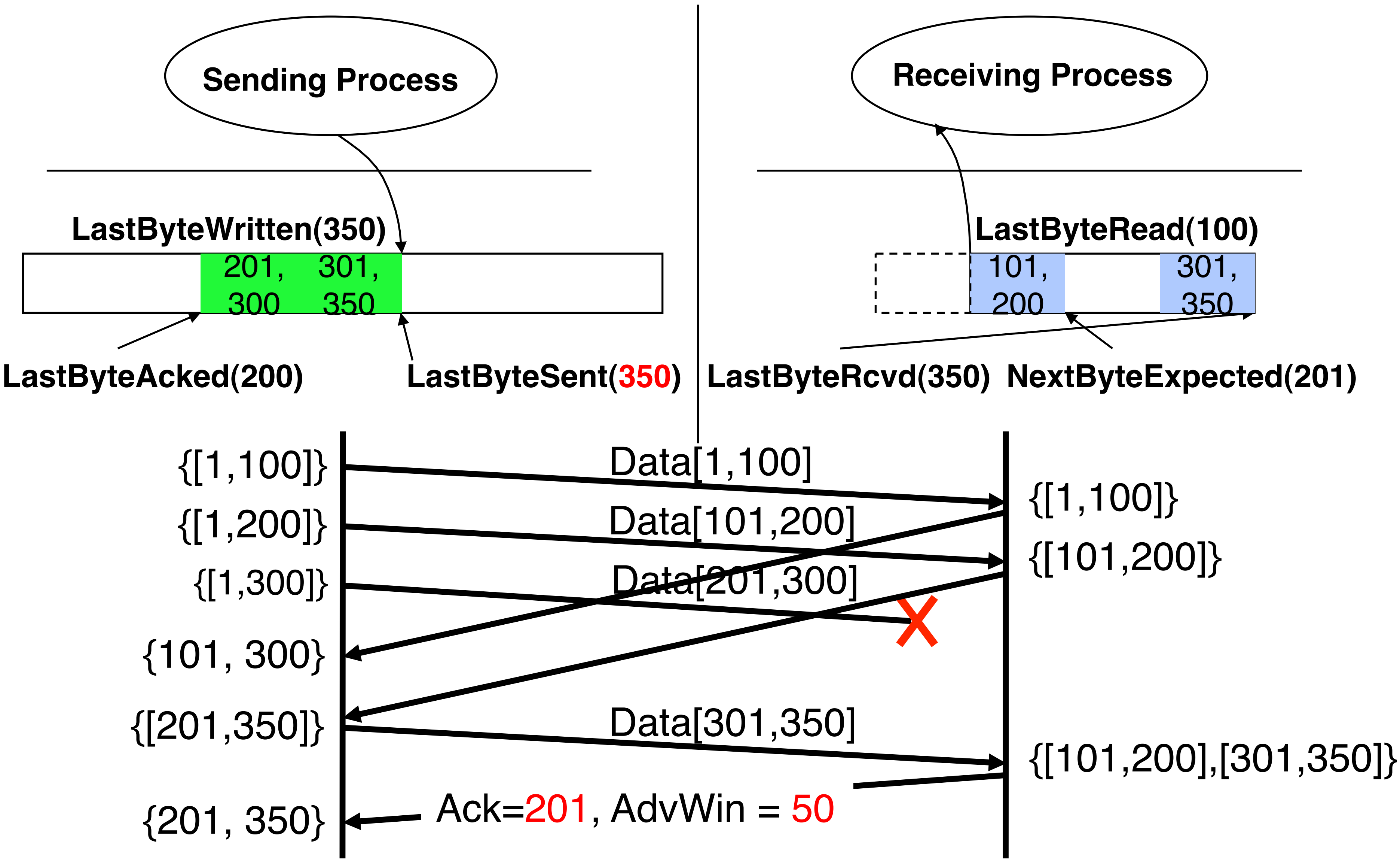
Sender can now send new data!

SndWin = AdvWin – (LasByteSent – LastByteAcked) = 100

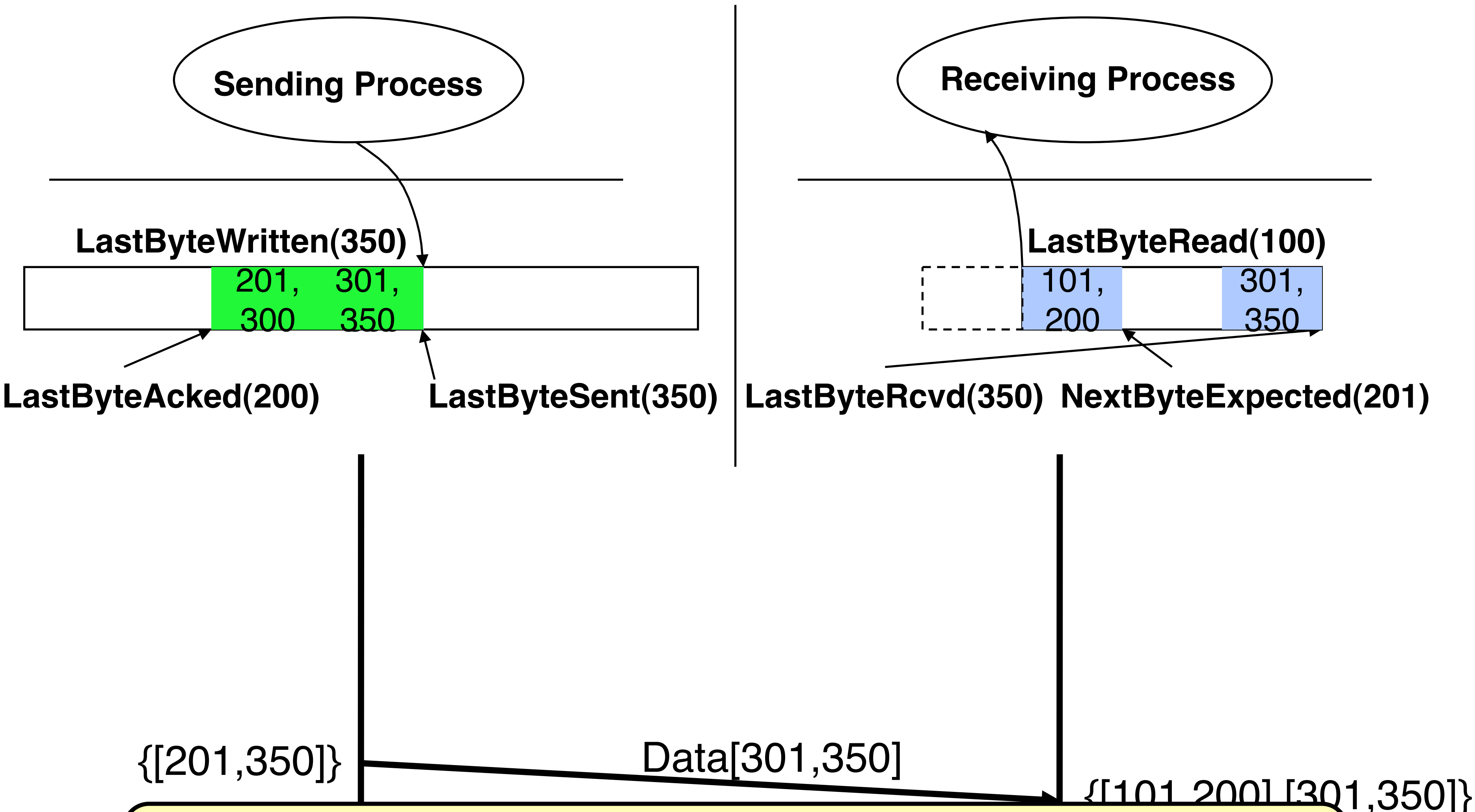
TCP Flow Control



TCP Flow Control

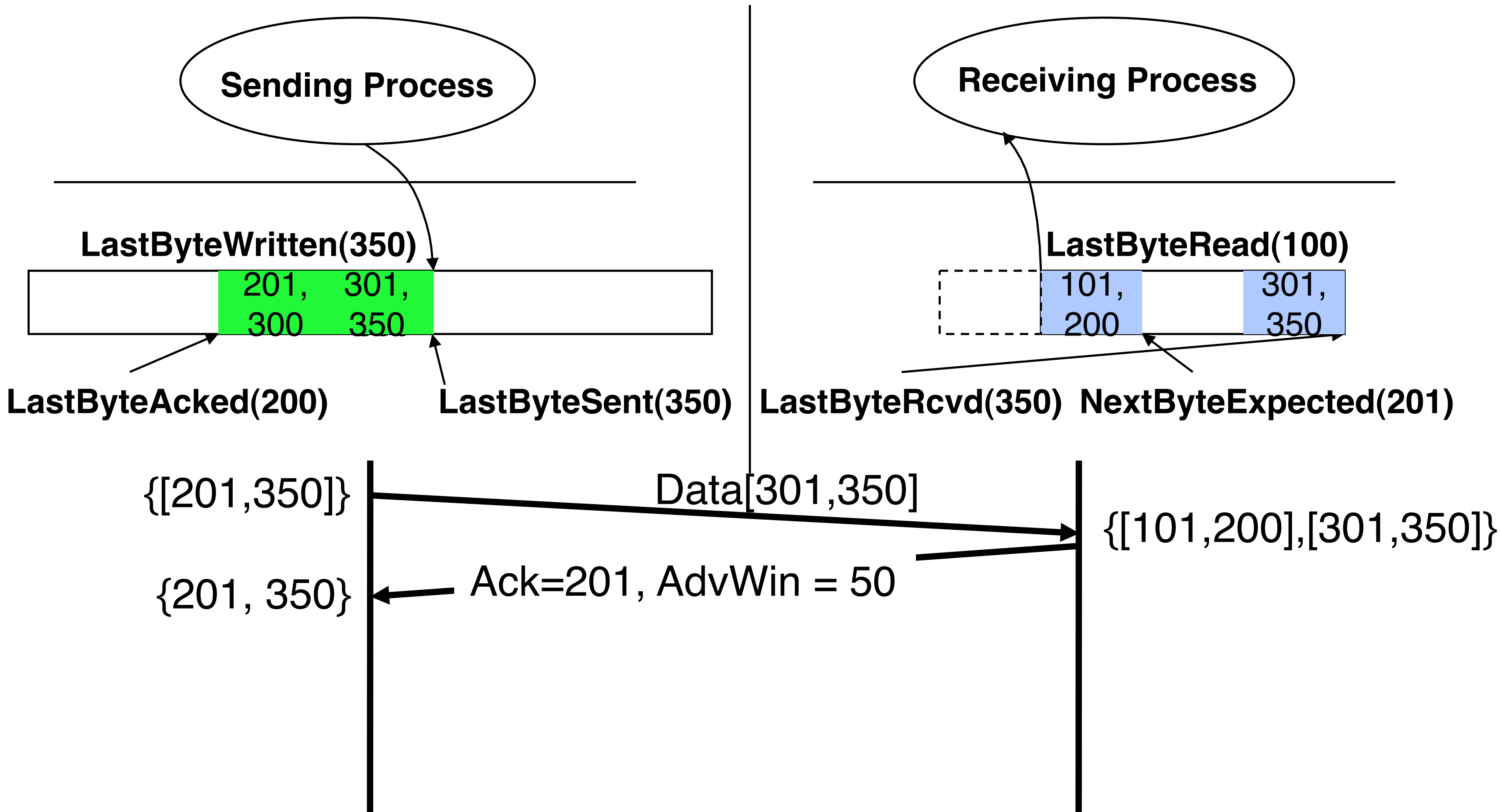


TCP Flow Control



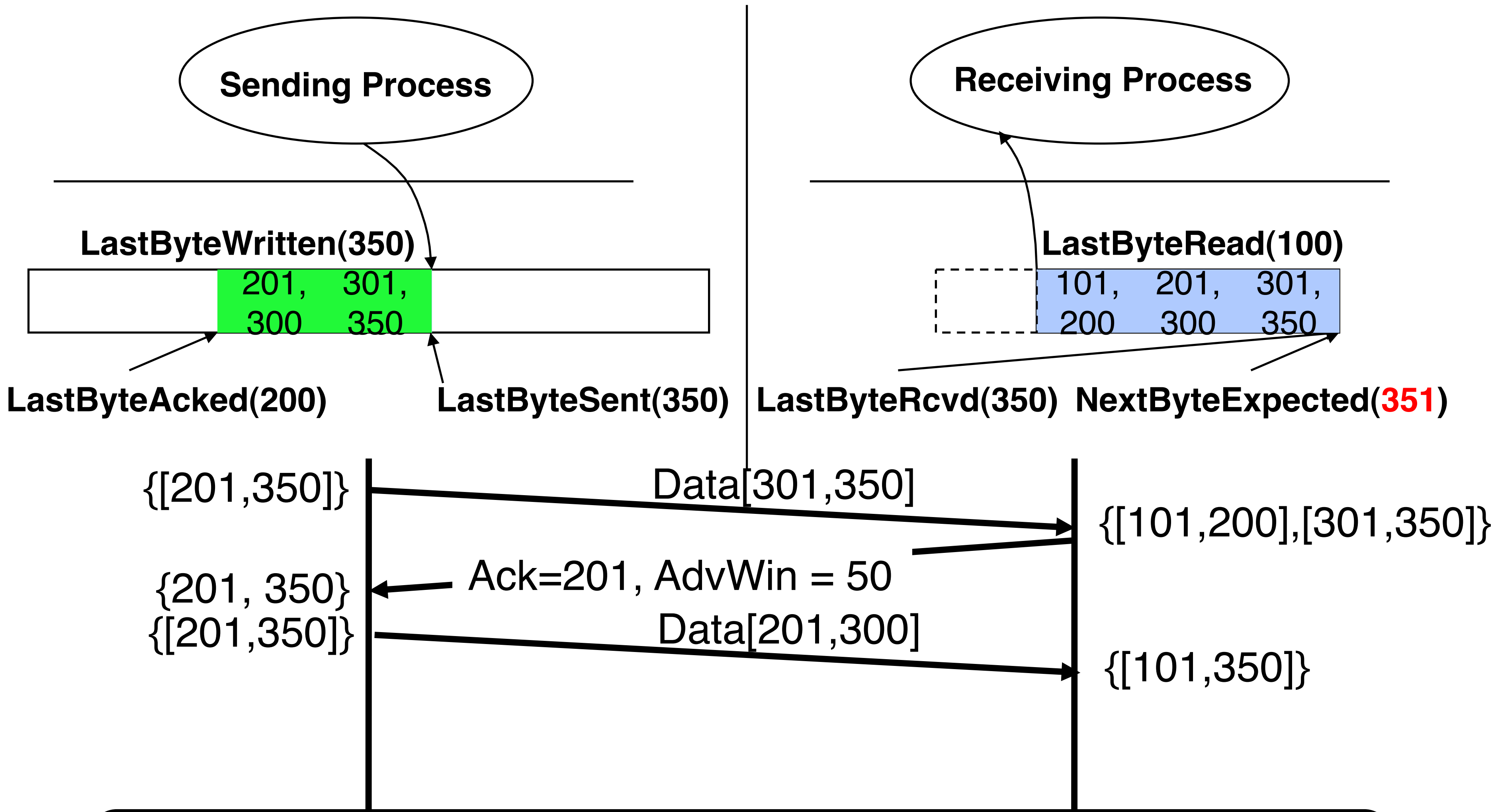
- Ack still specifies 201 (first byte out of sequence)
- AdvWin = 50, so can sender re-send 3rd packet?

TCP Flow Control



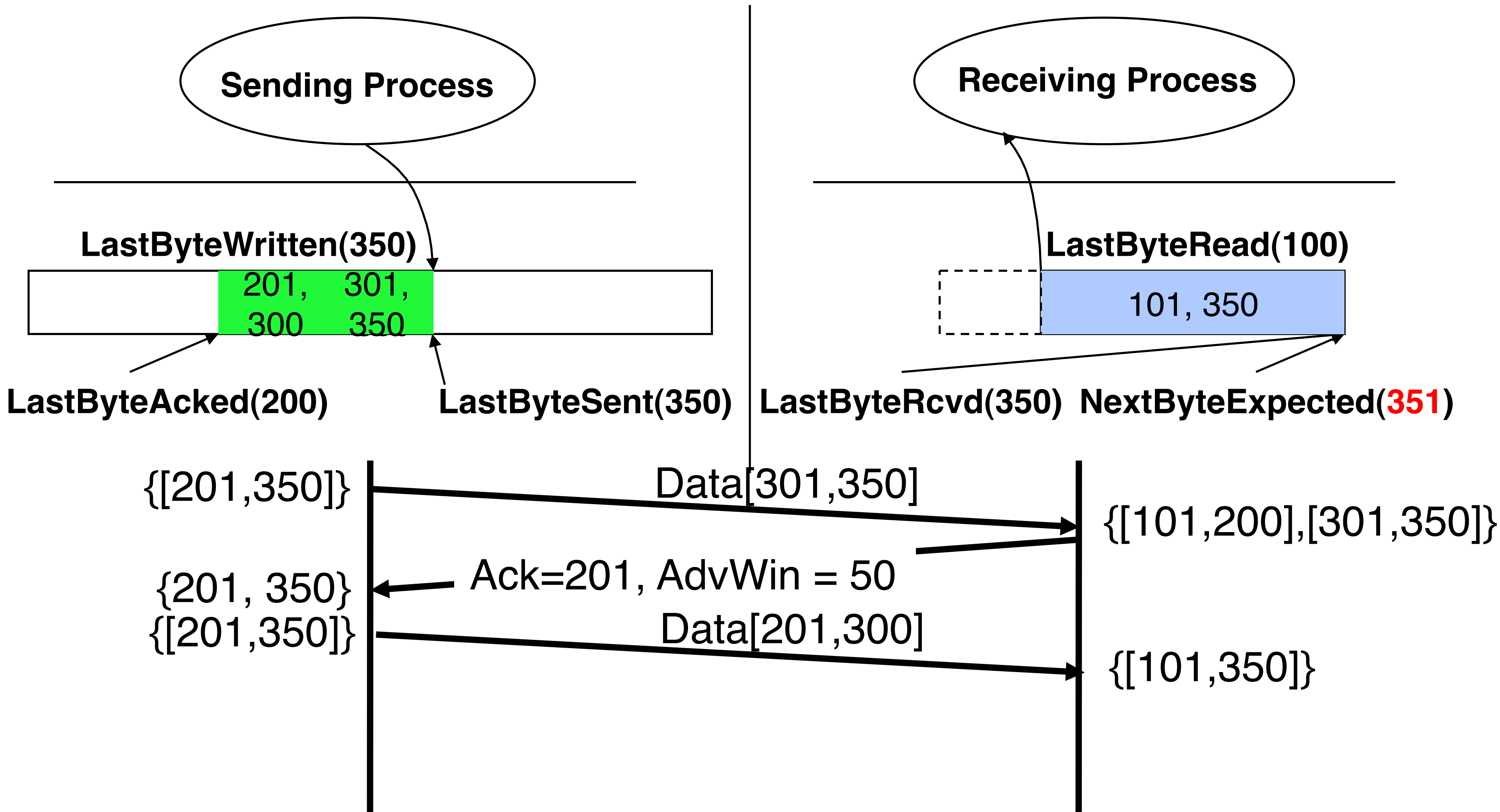
- Ack still specifies 201 (first byte out of sequence)
- AdvWin = 50, so can sender re-send 3rd packet?

TCP Flow Control



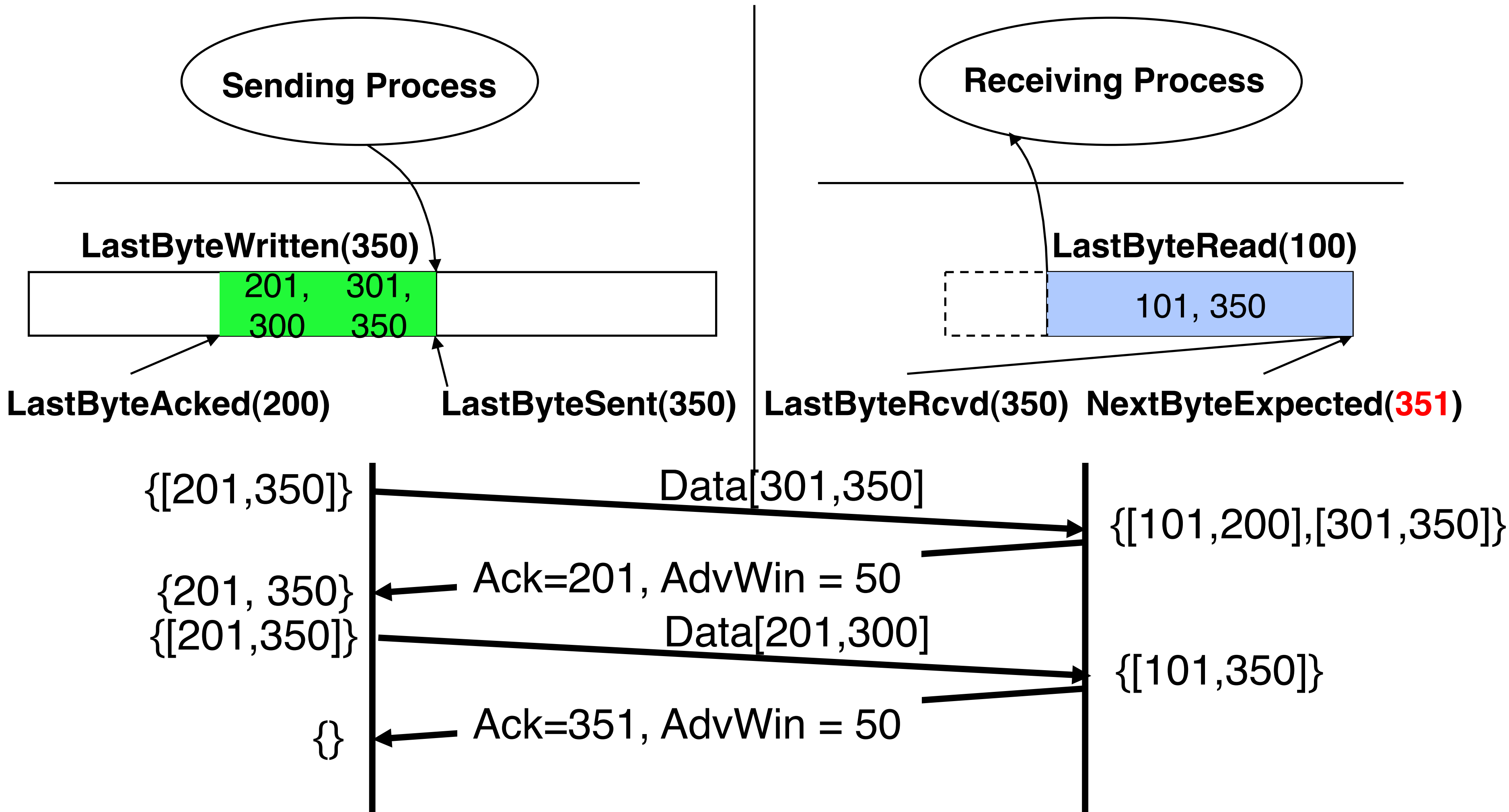
Yes! Sender can re-send 2nd packet since it's in existing window – won't cause receiver window to grow

TCP Flow Control



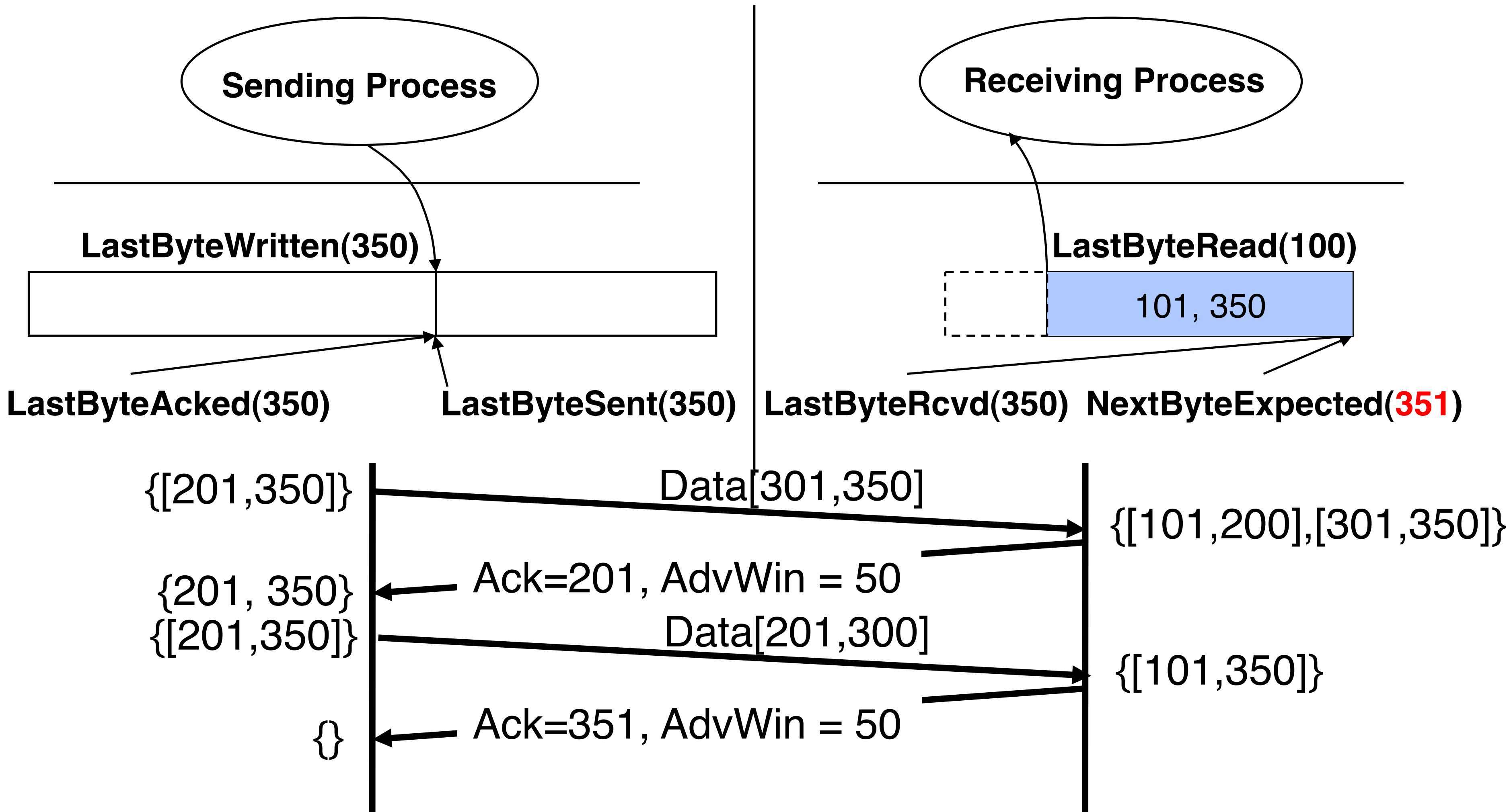
Yes! Sender can re-send 2nd packet since it’s in existing window – won’t cause receiver window to grow

TCP Flow Control



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

TCP Flow Control



Sender DONE with sending all bytes!

Discussion

- Why not have a huge buffer at the receiver (memory is cheap!)?

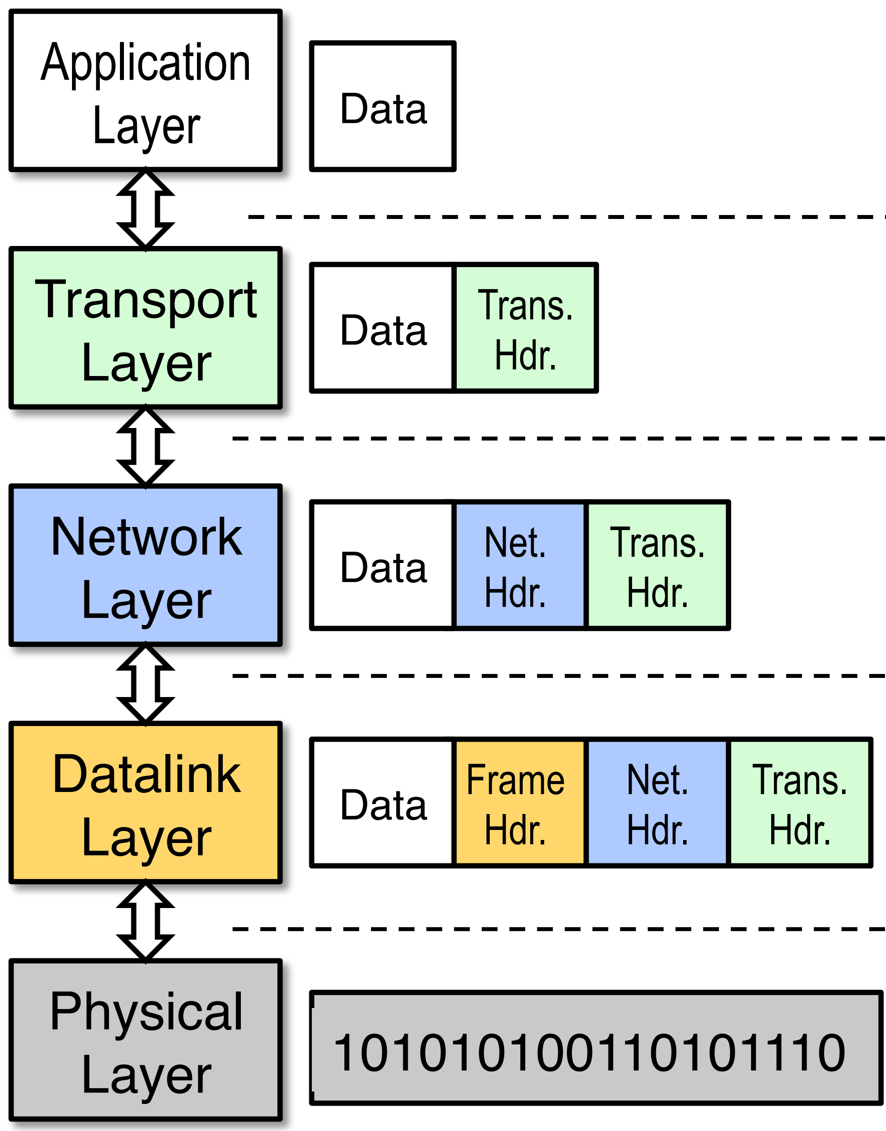
- Sending window (SndWnd) also depends on network congestion
  – **Congestion control**: ensure that a fast receiver doesn't overwhelm a router in the network (discussed in detail in ee122)

- In practice there is another set of buffers in the protocol stack, at the **link layer** (i.e., Network Interface Card)

Summary: Reliability & Flow Control

- Reliable transmission
  – S&W not efficient for links with large capacity (bandwidth) delay product
  – Sliding window far more efficient
- TCP: Reliable Byte Stream
  – Open connection (3-way handshaking)
  – Close connection: no perfect solution; no way for two parties to agree in the presence of arbitrary message losses (Byzantine General problem)
- Flow control: three pairs of producer consumers
  – Sending process → sending TCP
  – Sending TCP → receiving TCP
  – Receiving TCP → receiving process

Summary: Networking (Internet Layering)



Any distributed protocol (e.g., HTTP, Skype, p2p, KV protocol in your project)

Send *segments* to another *process* running on same or different node

Send *packets* to another node possibly *located* in a different network

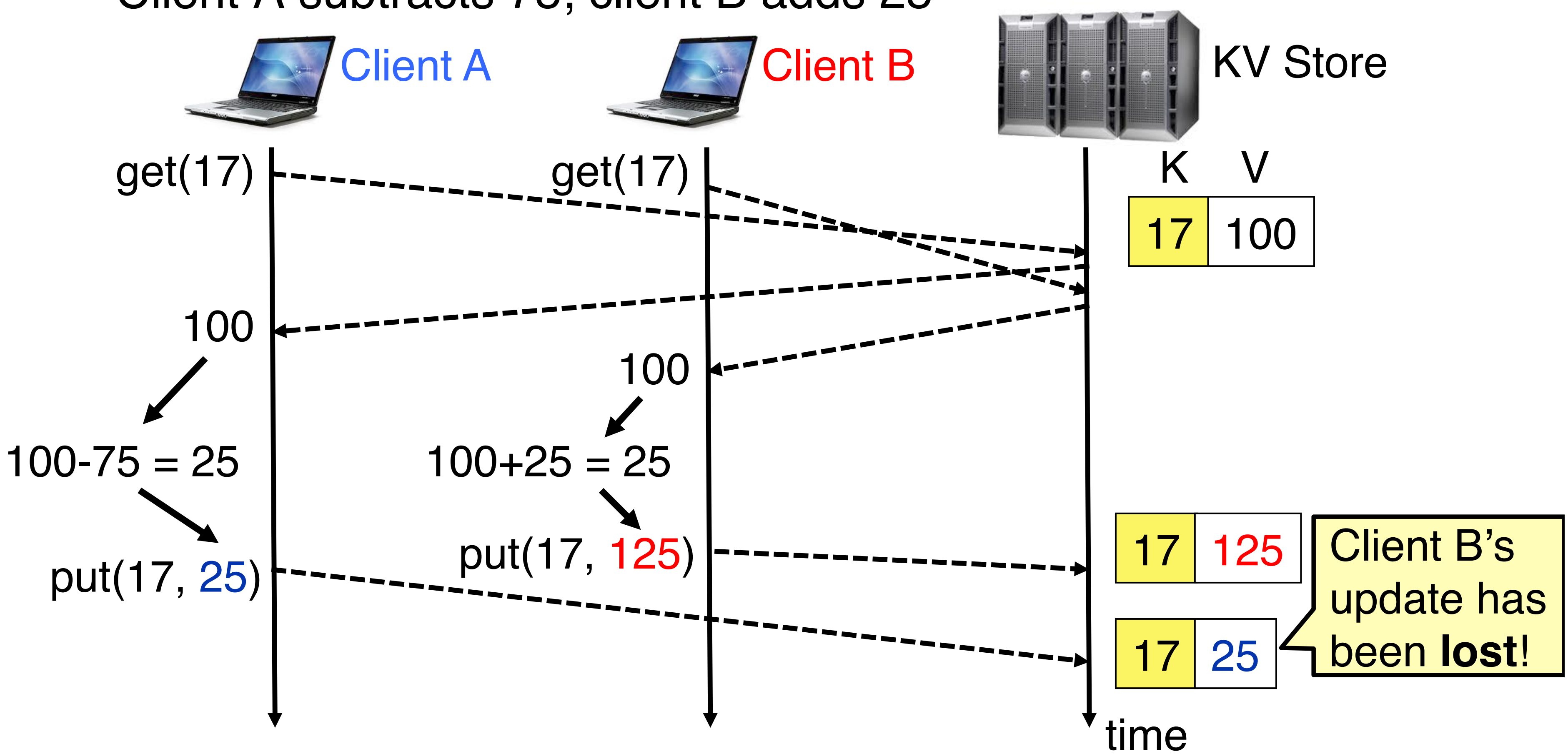
Send *frames* to other node directly connected to same physical network

Send *bits* to other node directly connected to same physical network

5min Break

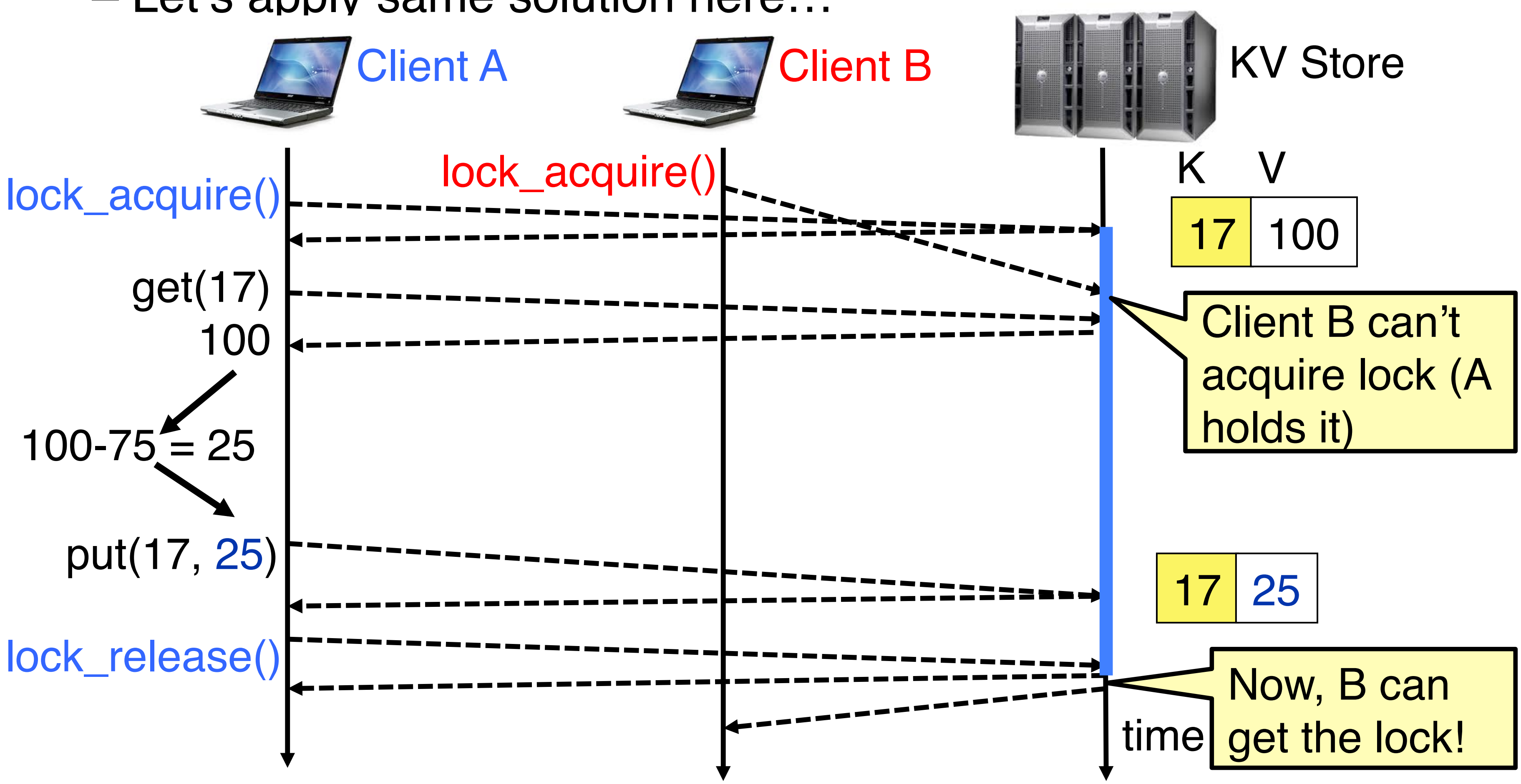
Need for Transactions

- Example: assume two clients updating same value in a key-value (KV) store at the same time
  - Client A subtracts 75; client B adds 25



Solution?

- How did we solve such problem on a single machine?
  – Critical section, e.g., use locks
  – Let's apply same solution here…



Discussion

- How does client B get the lock?
  - – Pooling: periodically check whether the lock is free
  - – KV storage system keeps a list of clients waiting for the lock, and gives the lock to next client in the list
- What happens if the client holding the lock crashes?
- Network latency might be higher than update operation
  - – Most of the time in critical section spent waiting for messages
- What is the lock granularity?
  - – Do you lock every key? Do you lock the entire storage?
  - – What are the tradeoffs?

Better Solution

- Interleave reads and writes from different clients

- Provide the same semantics as clients were running one at a time

- **Transaction** – database/storage sytem's abstract view of a user program, i.e., a sequence of reads and writes

"Classic" Example: Transaction

```
BEGIN;    --BEGIN TRANSACTION
 UPDATE accounts SET balance = balance -
   100.00 WHERE name = 'Alice';

 UPDATE branches SET balance = balance -
   100.00 WHERE name = (SELECT branch_name
   FROM accounts WHERE name = 'Alice');

 UPDATE accounts SET balance = balance +
   100.00 WHERE name = 'Bob';

 UPDATE branches SET balance = balance +
   100.00 WHERE name = (SELECT branch_name
   FROM accounts WHERE name = 'Bob');
COMMIT;    --COMMIT WORK
```

Transfer $100 from Alice's account to Bob's account

The ACID properties of Transactions

- **Atomicity:** all actions in the transaction happen, or none happen
- **Consistency:** if each transaction is consistent, and the database starts consistent, it ends up consistent, e.g.,
  - Balance cannot be negative
  - Cannot reschedule meeting on February 30
- **Isolation:** execution of one transaction is isolated from that of all others
- **Durability:** if a transaction commits, its effects persist

Atomicity

- A transaction
  - might *commit* after completing all its operations, or
  - it could *abort* (or be aborted) after executing some operations

- Atomic Transactions: a user can think of a transaction as always either *executing all its* operations, or *not executing any* operations at all
  - Database/storage system *logs* all actions so that it can *undo* the actions of aborted transactions

Consistency

- Data follows integrity constraints (ICs)
- If database/storage system is consistent before transaction, it will be after
- System checks ICs and if they fail, the transaction rolls back (i.e., is aborted)
  - A database enforces some ICs, depending on the ICs declared when the data has been created
  - Beyond this, database does not understand the semantics of the data (e.g., it does not understand how the interest on a bank account is computed)

Isolation

- Each transaction executes as if it was running by itself
  - Concurrency is achieved by database/storage, which interleaves operations (reads/writes) of various transactions

- Techniques:
  - Pessimistic – don't let problems arise in the first place
  - Optimistic – assume conflicts are rare, deal with them *after* they happen

Durability

- Data should survive in the presence of
  - System crash
  - Disk crash → need backups

- All committed updates and only those updates are reflected in the database
  - Some care must be taken to handle the case of a crash occurring during the recovery process!

This Lecture

- Deal with **(I)solation**, by focusing on **concurrency control**

- Next lecture focus on (A)tomicity, and partially on (D)urability

Example

- Consider two transactions:
  - T1: moves $100 from account A to account B

```
T1:A := A-100; B := B+100;
```

  - T2: moves $50 from account B to account A

```
T2:A := A+50;  B := B-50;
```

- Each operation consists of (1) a read, (2) an addition/ subtraction, and (3) a write
- Example: A = A-100

```
Read(A); // R(A)
A := A - 100;
Write(A); // W(A)
```

Example (cont'd)

- Database only sees reads and writes

Database View

`T1: A:=A-100; B:=B+100;`	→	`T1:R(A),W(A),R(B),W(B)`
`T2: A:=A+50; B:=B-50;`	→	`T2:R(A),W(A),R(B),W(B)`

- Assume initially: A = \$1000 and B = \$500
- What is the legal outcome of running T1 and T2?
  - A = \$950
  - B = \$550

Example (cont'd)

```
T1: A:=A-100; B:=B+100;
```

```
T2: A:=A+50; B:=B-50;
```

Initial values:

`A:=1000`

`B:=500`

- What is the outcome of the following execution?

```
T1:R(A),W(A),R(B),W(B)
T2:                     R(A),W(A),R(B),W(B)
```

A=900

B=600

A=950

B=550

- What is the outcome of the following execution?

```
T1:                     R(A),W(A),R(B),W(B)
T2:R(A),W(A),R(B),W(B)
```

A=950

B=550

A=1050

B=450

Example (cont' d)

```
T1: A:=A-100; B:=B+100;
```

```
T2: A:=A+50; B:=B-50;
```

Initial values:

A:=1000

B:=500

- What is the outcome of the following execution?

```
T1:R(A),W(A),                          R(B),W(B)
T2:          R(A),W(A),R(B),W(B)
```

A=900 A=950 B=450 B=550

- What is the outcome of the following execution?

```
T1:R(A),                     W(A),R(B),W(B)
T2:     R(A),W(A),R(B),W(B)
```

A=1050 B=450 A=900 B=550

Lost $50!

Transaction Scheduling

- Why not run only one transaction at a time?
- Answer: low system utilization
  - Two transactions cannot run simultaneously even if they access different data
- Goal of transaction scheduling:
  - Maximize system utilization, i.e., concurency
    - » Interleave operations from different transactions
  - Preserve transaction semantics
    - » Logically the sequence of all operations in a transaction are executed atomically
    - » Intermediate state of a transaction is not visible to other tranasctions

Summary

- Transaction: a sequence of storage operations

- ACID:
  - Atomicity: all operations in a transaction happen, or none happens
  - Consistency: if database/storage starts consistent, it ends up consistent
  - Isolation: execution of one transaction is isolated from another
  - Durability: the results of a transaction persists