CS Study 3주차

Netwrok - 2

김신아

TCP 와 UDP

• TCP(Transmission Control Protocol)
서버와 클라이언트간의 데이터를 신뢰성있게 전달하기 위해 만든 프로토콜,
신뢰성있고 **연결지향적**이다.

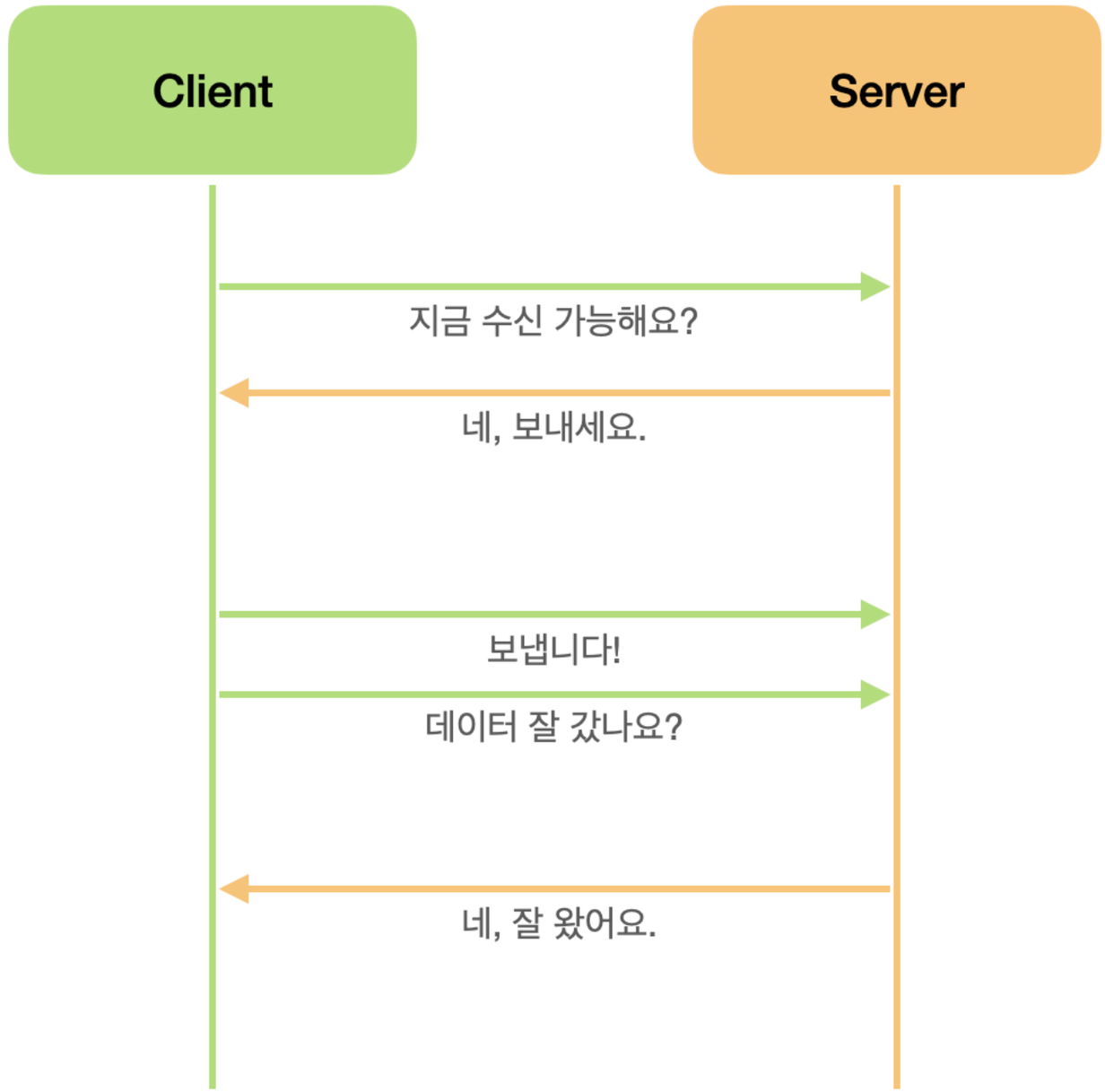
• UDP(User Datagram Protocol)
비연결형 서비스를 지원하는 전송계층 프로토콜로써, 인터넷상에서 서로 정보를 주고받을 때 정보를 보낸다는 신호나 보낸다는 신호 절차를 거치지 않고, 보내는 쪽에서 **일방적으로** 데이터를 전달하는 통신 프로토콜이다.

• TCP와 UDP 비교
TCP는 연속성보다 신뢰성있는 전송이 중요할 때에 사용하는 프로토콜이며,
UDP는 TCP보다 속도가 빠르며 네트워크 부하가 적다는 장점이 있지만,
신뢰성있는 데이터 전송을 보장하지 않는다.

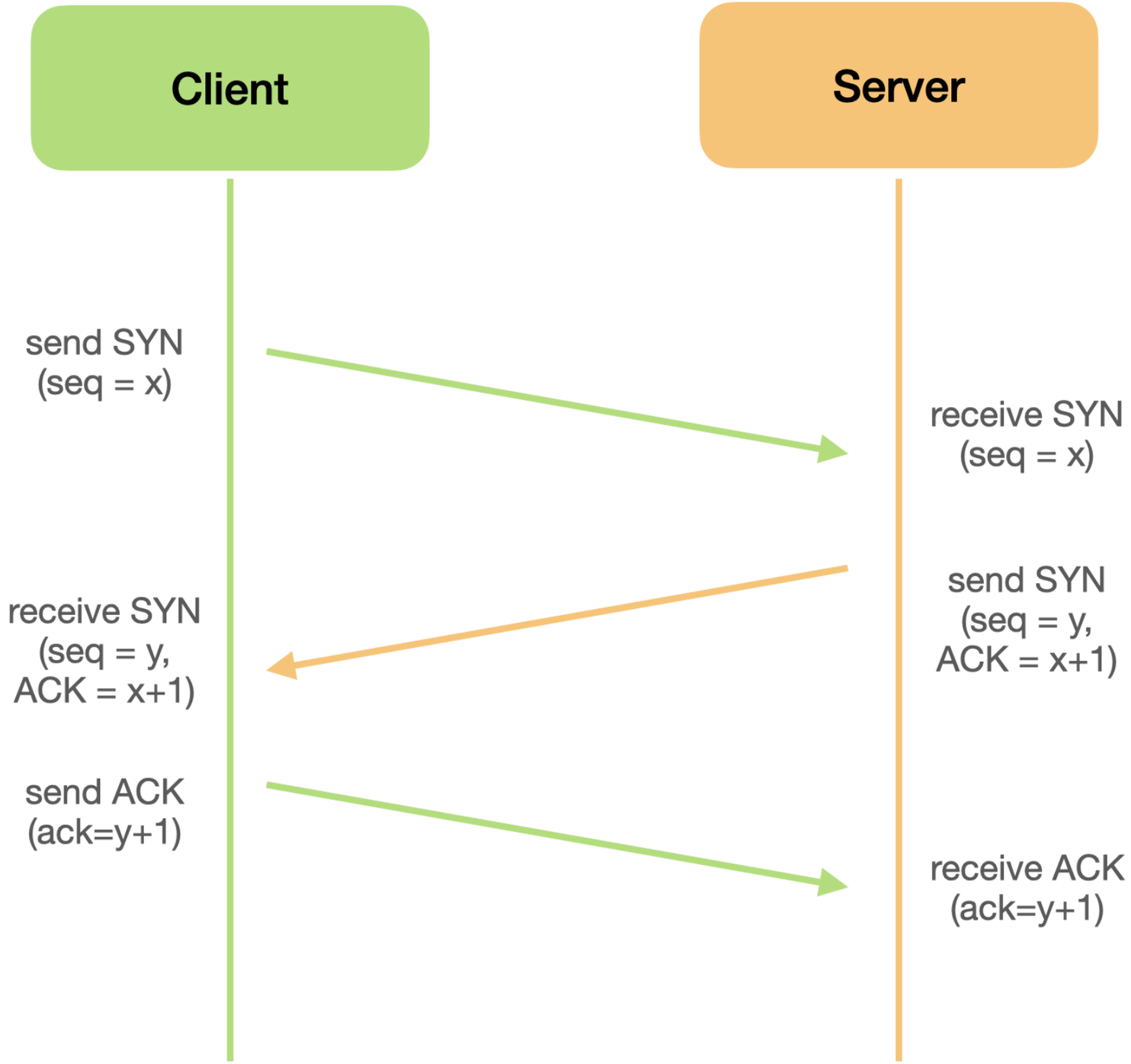
TCP 와 UDP

TCP (Transmission Control Protocol)	UDP (User Datagram Protocol)
연결이 성공해야 통신 가능 (연결형 프로토콜)	연결 없이 통신 가능 (비연결형 프로토콜)
데이터의 경계를 구분하지 않음 (Byte-Stream Service)	데이터의 경계를 구분함 (Datagram Service)
신뢰성 있는 데이터 전송 (데이터의 재전송 존재)	비신뢰성 데이터 전송 (데이터의 재전송 없음)
1 : 1 (Unicast) 통신	1 : 1, 1: N (Broadcast), N : M (Multicast) 통신

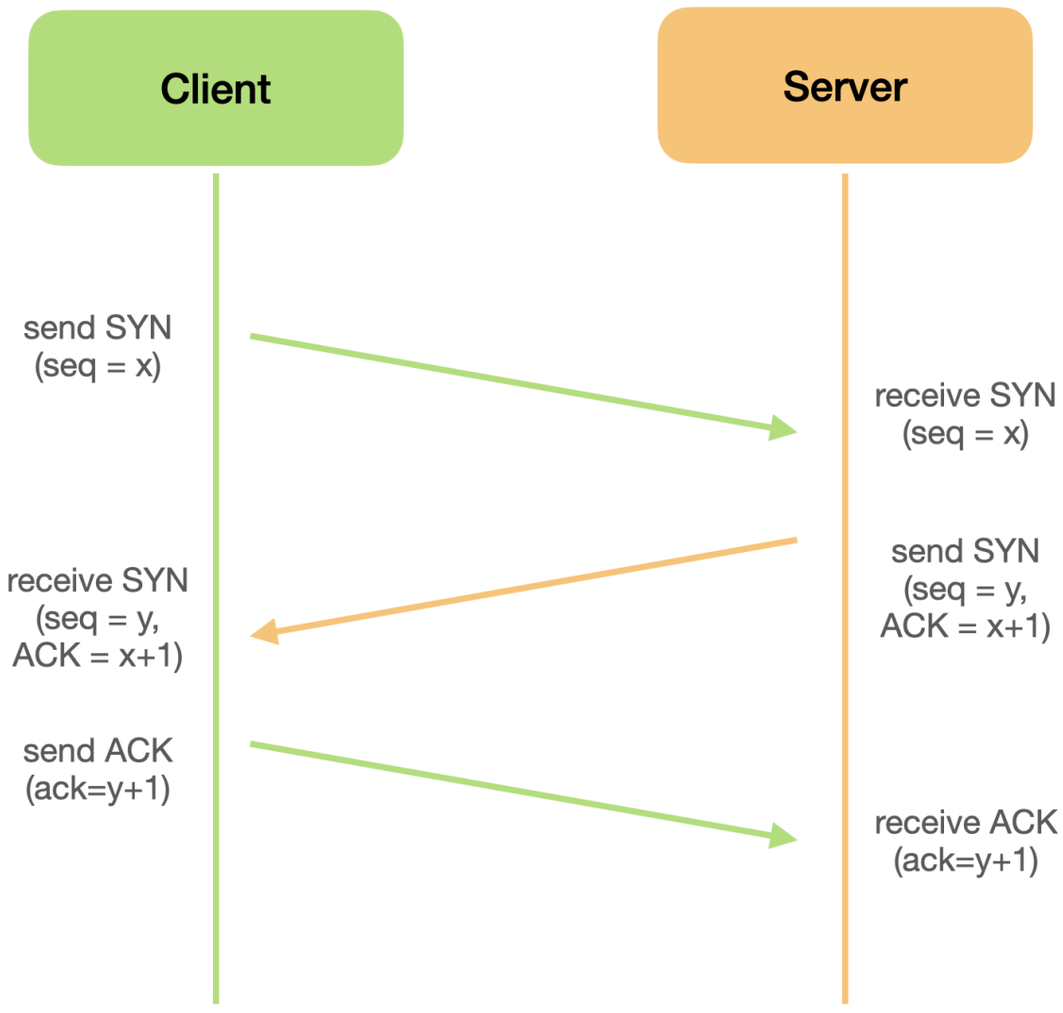
TCP - 3 way handshake (연결 성립)



TCP - 3 way handshake (연결 성립)



TCP - 3 way handshake (연결 성립)

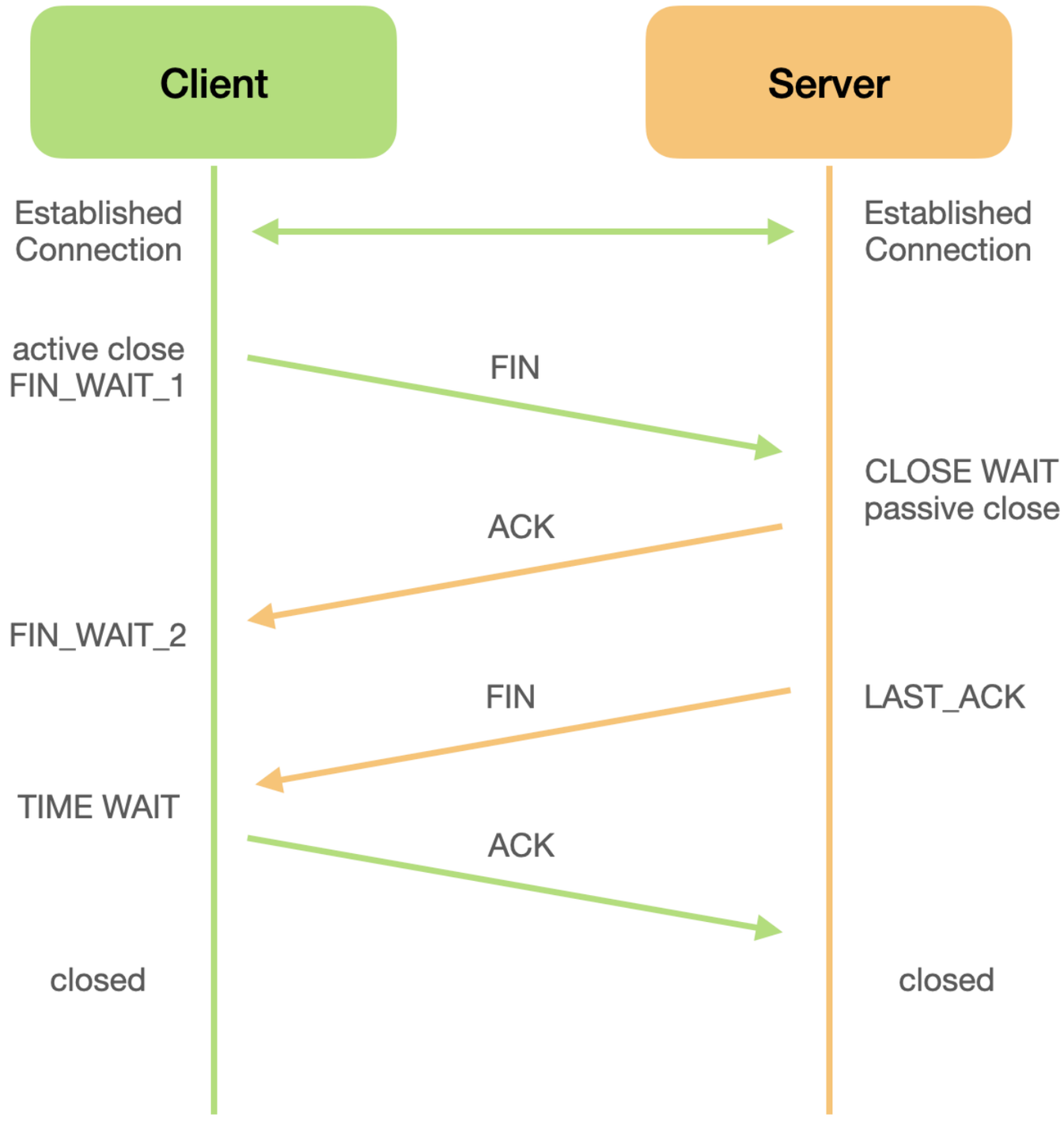


1. 클라이언트가 서버에게 SYN 패킷을 보냄 (sequence : x)

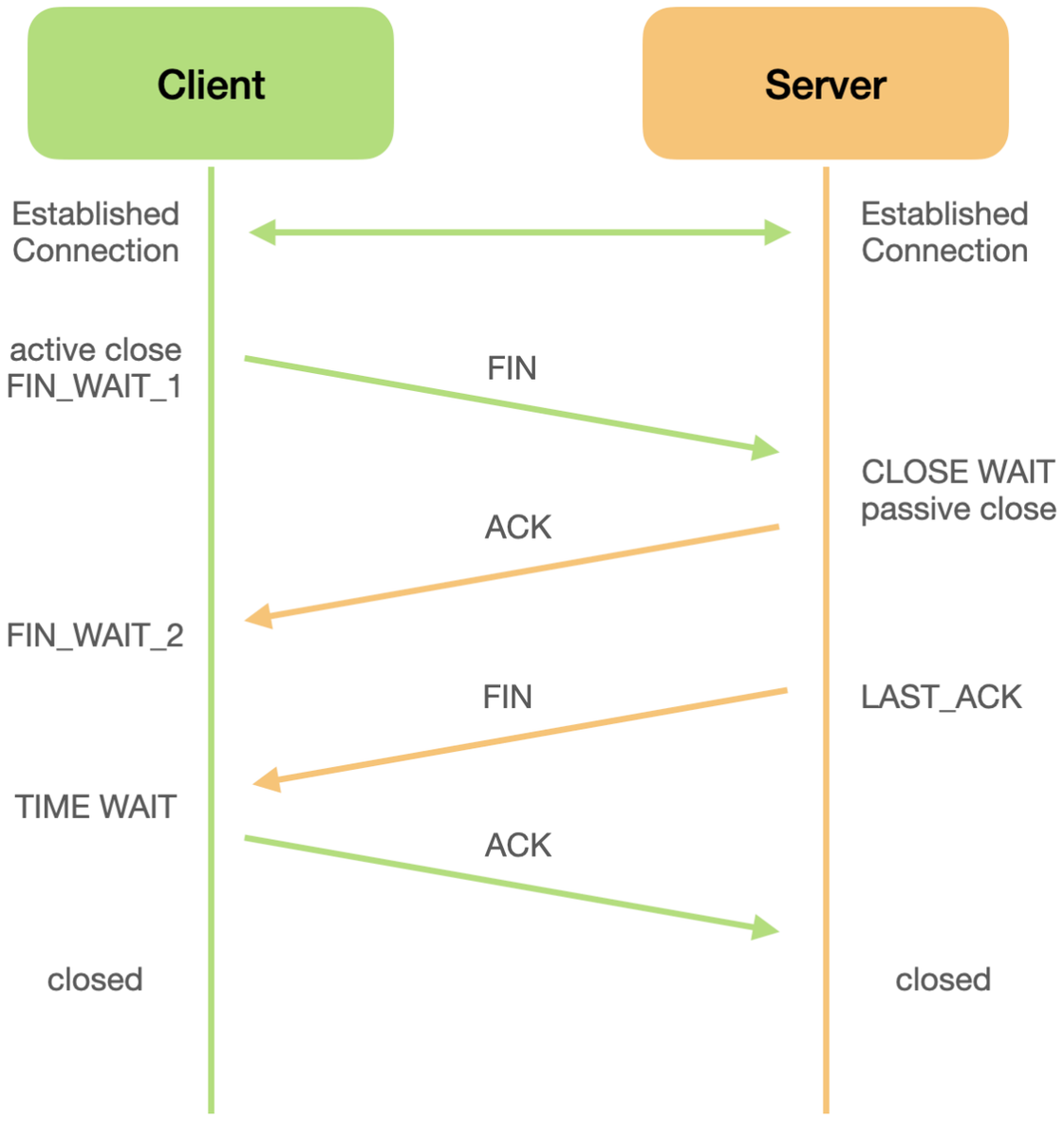
2. 서버가 SYN(x)을 받고, 클라이언트로 받았다는 신호인 ACK와 SYN 패킷을 보냄 (sequence : y, ACK : x + 1)

3. 클라이언트는 서버의 응답 ACK(x+1)와 SYN(y) 패킷을 받고, ACK(y+1)를 서버로 보냄

TCP - 4 way handshake (연결 해제)



TCP - 4 way handshake (연결 해제)



1. 클라이언트는 서버에게 연결을 종료한다는 FIN flag를 보냄

2. 서버는 FIN을 받고, 확인했다는 ACK를 클라이언트에게 보냄

3. 데이터를 모두 보냈다면, 연결이 종료되었다는 FIN flag를 클라이언트에게 보냄

4. 클라이언트는 FIN을 받고, 확인했다는 ACK를 서버에게 보냄 (아직 서버로부터 받지 못한 데이터가 있을 수 있으므로 TIME_WAIT을 통해 기다림)

TCP / IP - Reliable Network

Reliable Network를 보장한다는 것은 4가지 문제점 존재

1. 손실 : packet이 손실될 수 있는 문제
2. 순서 바뀜 : packet의 순서가 바뀌는 문제
3. Congestion : 네트워크가 혼잡한 문제
4. Overload : receiver가 overload 되는 문제

TCP / IP - 흐름제어 / 혼잡제어

- 흐름제어 (endsystem 대 endsystem)
  - 송신측과 수신측의 데이터 처리 속도 차이를 해결하기 위한 기법
  - Flow Control은 receiver가 packet을 지나치게 많이 받지 않도록 조절하는 것
  - 기본 개념은 receiver가 sender에게 현재 자신의 상태를 feedback 한다는 점

- 혼잡제어
  - 네트워크 내에 패킷의 수가 과도하게 증가하는 현상을 혼잡이라 하며,
  혼잡 현상을 방지하거나 제거
  - 송신측의 데이터 전달과 네트워크의 데이터 처리 속도 차이를 해결하기 위한 기법

TCP / IP - 흐름제어

- 흐름제어 (endsystem 대 endsystem)

 - 수신측이 송신측보다 데이터 처리 속도가 빠르면 문제없지만, 송신측의 속도가 빠를 경우 문제 발생

- 수신측에서 제한된 저장 용량을 초과한 이후에 도착하는 데이터는 손실될 수 있으며, 만약 손실 된다면 불필요하게 응답과 데이터 전송이 송/수친측 간에 빈번이 발생

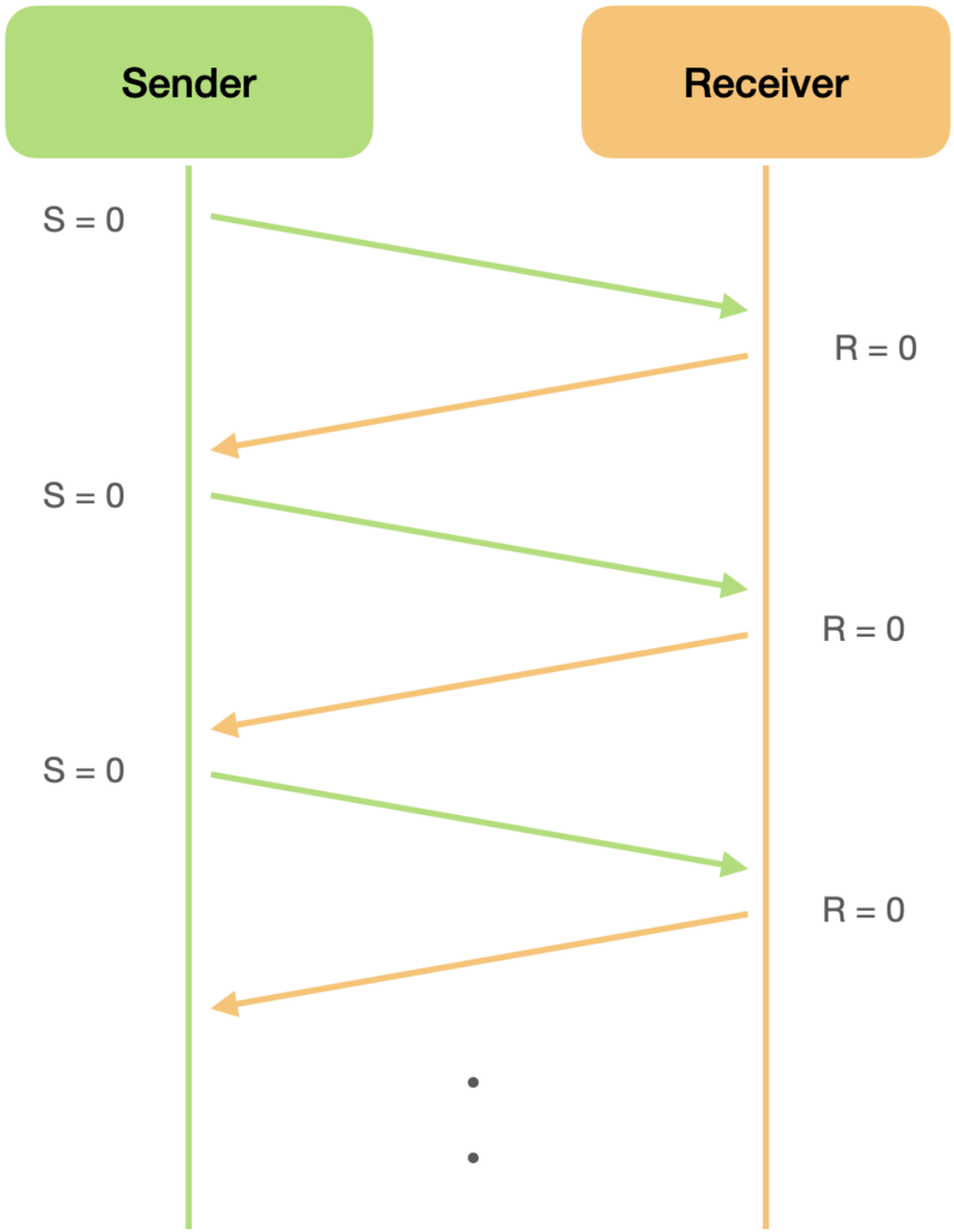
- 이러한 위험을 줄이기 위해 송신 측의 데이터 전송량을 수신측에 따라 조정해야함

TCP / IP - 흐름제어

- 해결방법
  - Stop and Wait
  - Sliding Window (Go Back N ARQ)

TCP / IP - 흐름제어

- Stop and Wait



매번 전송한 패킷에 대해 확인 응답을 받아야만 그 다음 패킷을 전송하는 방법

TCP / IP - 흐름제어

- **Sliding Window (Go Back N ARQ)**

 - 수신측에서 설정한 윈도우 크기만큼 송신측에서 확인응답없이 세그먼트를 전송할 수 있게하여 데이터 흐름을 동적으로 조절하는 제어 기법

 - 목적 : 전송은 되었지만, acked를 받지 못한 byte의 숫자를 파악하지 위해 사용하는 프로토콜

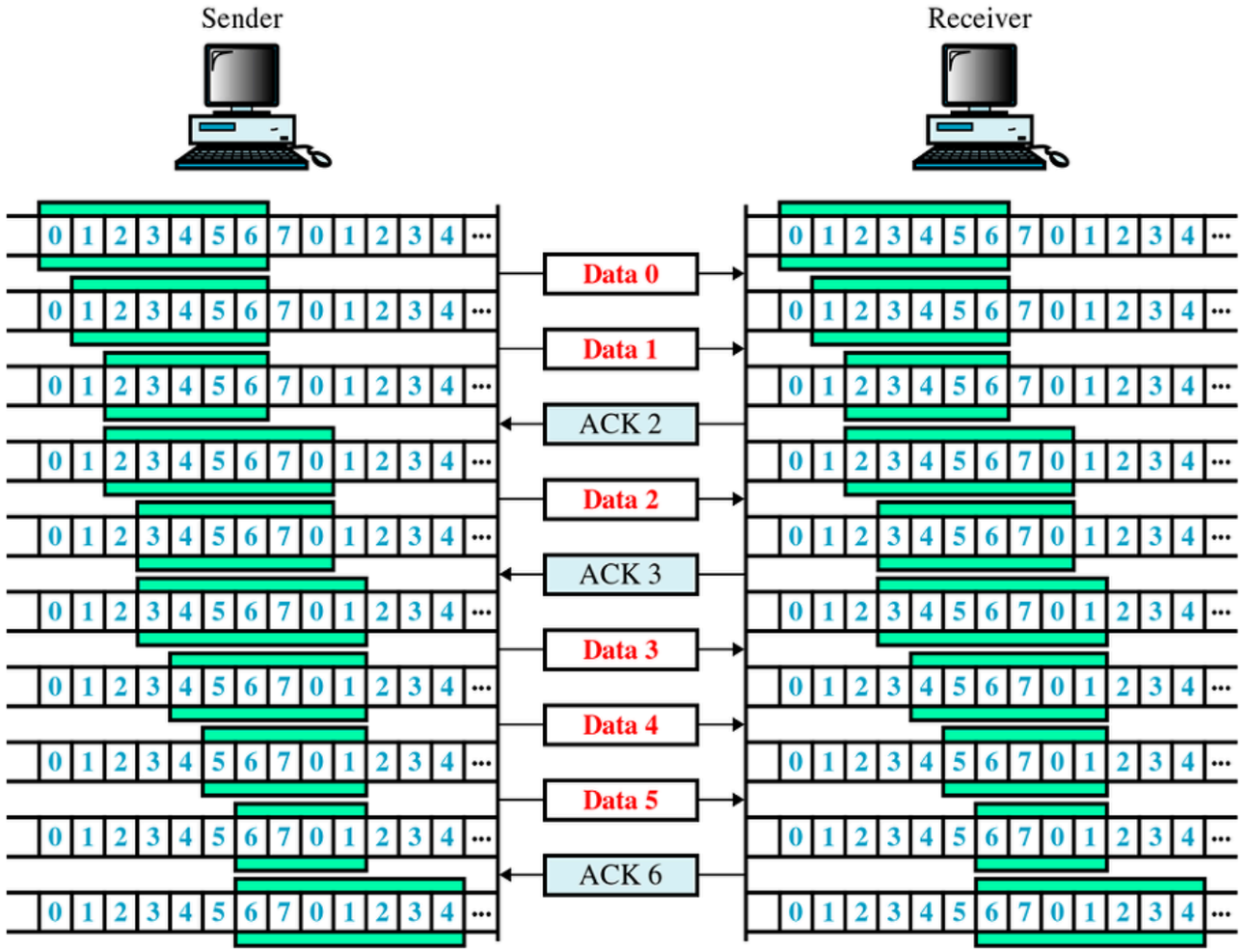
LastByteSent - LastByteAcked < = ReceivecWindowAdvertised

(마지막에 보내진 바이트 - 마지막에 확인된 바이트 < = 남아있는 공간) ==

(현재 공중에 떠있는 패킷 수 < = sliding window)

TCP / IP - 흐름제어

- Sliding Window (Go Back N ARQ)



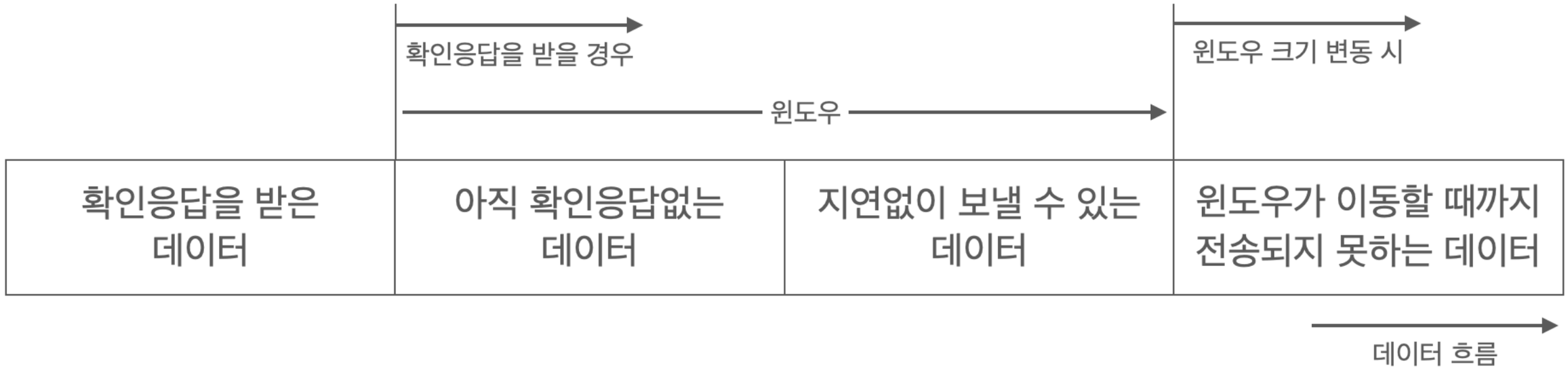
동작방식 : 먼저 윈도우에 포함되는 모든 패킷을 전송하고, 그 패킷들의 전달이 확인되는대로 이 윈도우를 옆으로 옮김으로써 그 다음 패킷들을 전송

Window : TCP/IP를 사용하는 모든 호스트들은 송신하기 위한 2개의 window를 가지고 있음

호스트들은 실제 데이터를 보내기전에 ‘3 way handshaking’을 통해 수신 호스트의 receive window size에 자신의 send window size를 맞추게 됨

TCP / IP - 흐름제어

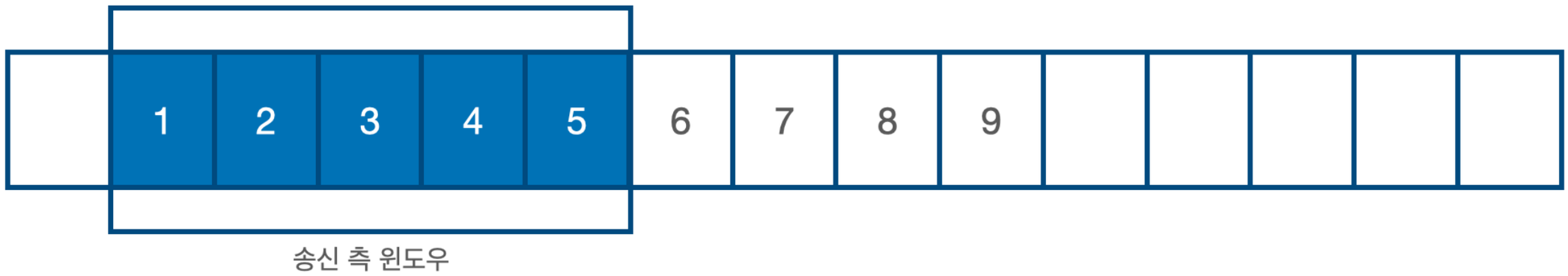
- Sliding Window (Go Back N ARQ)



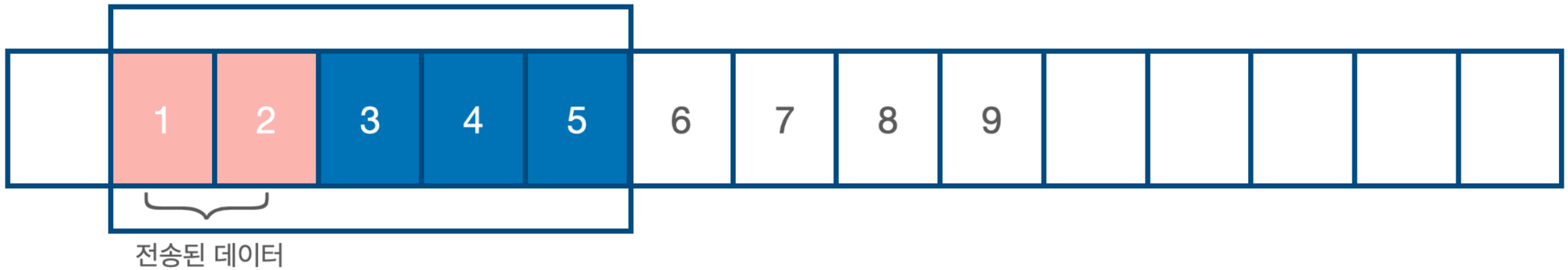
TCP / IP - 흐름제어

- Sliding Window (Go Back N ARQ)

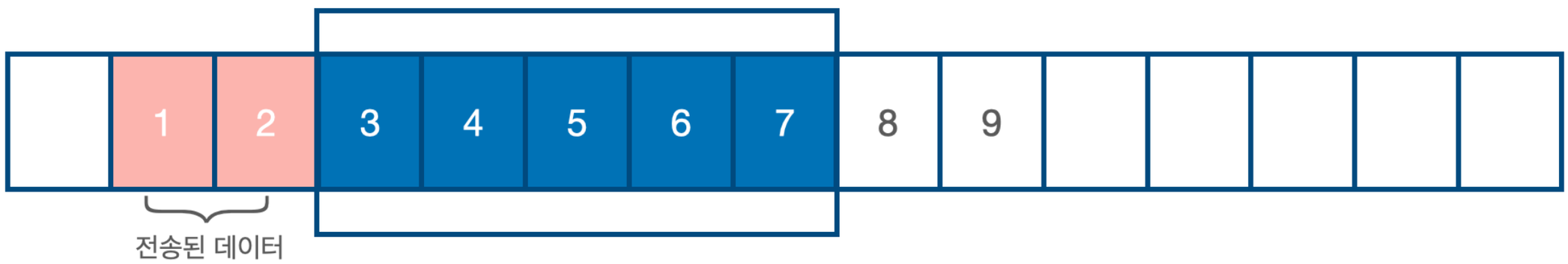
1. 송신 측에서는 1~5까지 프레임 전송가능



2. 데이터 1, 2를 전송하고 아직 데이터 3, 4, 5는 전송하지 않은 상태



3. 데이터 1, 2에 대한 ACK 프레임 수신 후, ACK된 프레임 만큼 윈도우 이동



TCP / IP - 혼잡제어

- 해결방법
  - AIMD (Additive Increase / Multiplicative Decrease)
  - Slow Start (느린 시작)
  - Fast Retransmit (빠른 재전송)
  - Fast Recovery (빠른 회복)

TCP / IP - 혼잡제어

- **AIMD**
  - 처음에 패킷을 하나씩 보내고 이것이 문제없이 도착하면 window 크기(단위 시간 내에 보내는 패킷의 수)를 **1씩** 증가시켜가며 전송하는 방법
  - 패킷 전송에 실패하거나 일정 시간을 넘으면 패킷의 보내는 속도를 절반으로 줄인다.
  - 공평한 방식으로, 여러 호스트가 한 네트워크를 공유하고 있으면 나중에 진입하는 쪽이 처음에는 불리하지만, 시간이 흐르면 평형상태로 수렴하게 되는 특징이 있다.
  - **문제점**은 초기에 네트워크의 높은 대역폭을 사용하지 못하여 오랜 시간이 걸리게 되고, 네트워크가 혼잡해지는 상황을 미리 감지하지 못한다. 즉, 네트워크가 혼잡해지고 나서야 대역폭을 줄이는 방식이다.

TCP / IP - 혼잡제어

• Slow Start (느린 시작)

- Slow Start 방식은 AIMD와 마찬가지로 패킷을 하나씩 보내면서 시작하고, 패킷이 문제없이 도착하면 각각의 ACK 패킷마다 window size를 1씩 늘려준다. 즉, 한 주기가 지나면 window size가 2배로 된다.

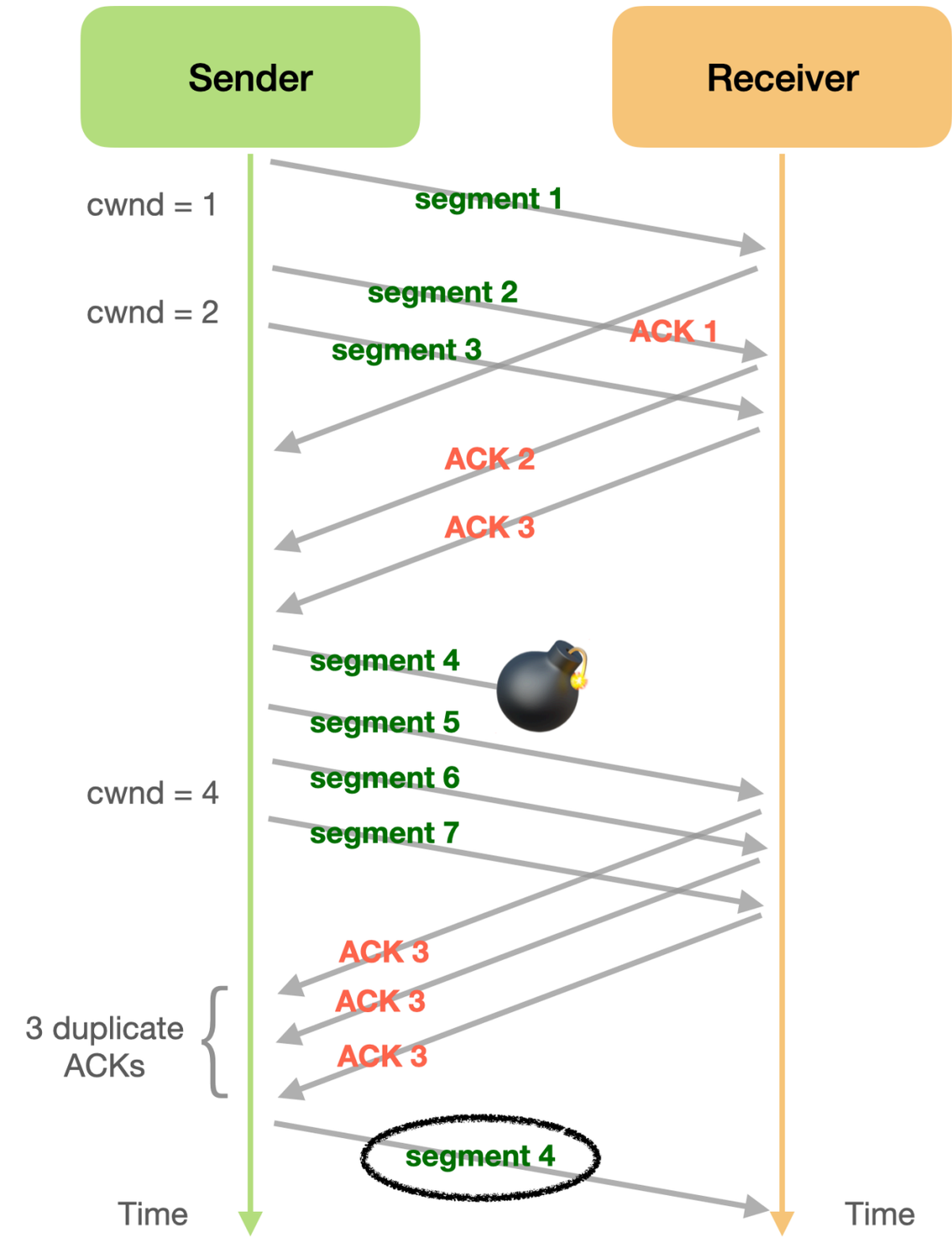
- 전송속도는 AIMD에 반해 지수 함수 꼴로 증가한다. 대신에 혼잡 현상이 발생하면 window size를 1로 떨어뜨리게 된다.

- 처음에는 네트워크의 수용량을 예상할 수 있는 정보가 없지만, 한번 혼잡 현상이 발생하고 나면 네트워크의 수용량을 어느 정도 예상할 수 있다.

- 그러므로 혼잡 현상이 발생하였던 window size의 절반까지는 이전처럼 지수 함수 꼴로 창 크기를 증가시키고 그 이후부터는 완만하게 1씩 증가시킨다.

TCP / IP - 혼잡제어

- Fast Retransmit (빠른 재전송)



TCP / IP - 혼잡제어

- **Fast Retransmit (빠른 재전송)**

 -> 패킷을 받는 수신자 입장에서는 세그먼트로 분할된 내용들이 순서대로 도착하지 않는 경우가 생길 수 있다.

 - 패킷을 받는 쪽에서 먼저 도착해야할 패킷이 도착하지 않고 다음 패킷이 도착한 경우에도 **ACK** 패킷을 보내게 된다.

 - 단, 순서대로 잘 도착한 마지막 패킷의 다음 패킷의 순번을 **ACK** 패킷에 실어서 보내게 되므로, 중간에 하나가 손실되게 되면 송신 측에서는 순번이 중복된 **ACK** 패킷을 받게 된다. 이것을 감지하는 순간 문제가 되는 순번의 패킷을 재전송 해줄 수 있다.

 - 중복된 순번의 패킷을 3개 받으면 재전송을 하게 된다. 약간 혼잡한 상황이 일어난 것이므로 혼잡을 감지하고 **window size**를 줄이게 된다.

TCP / IP - 혼잡제어

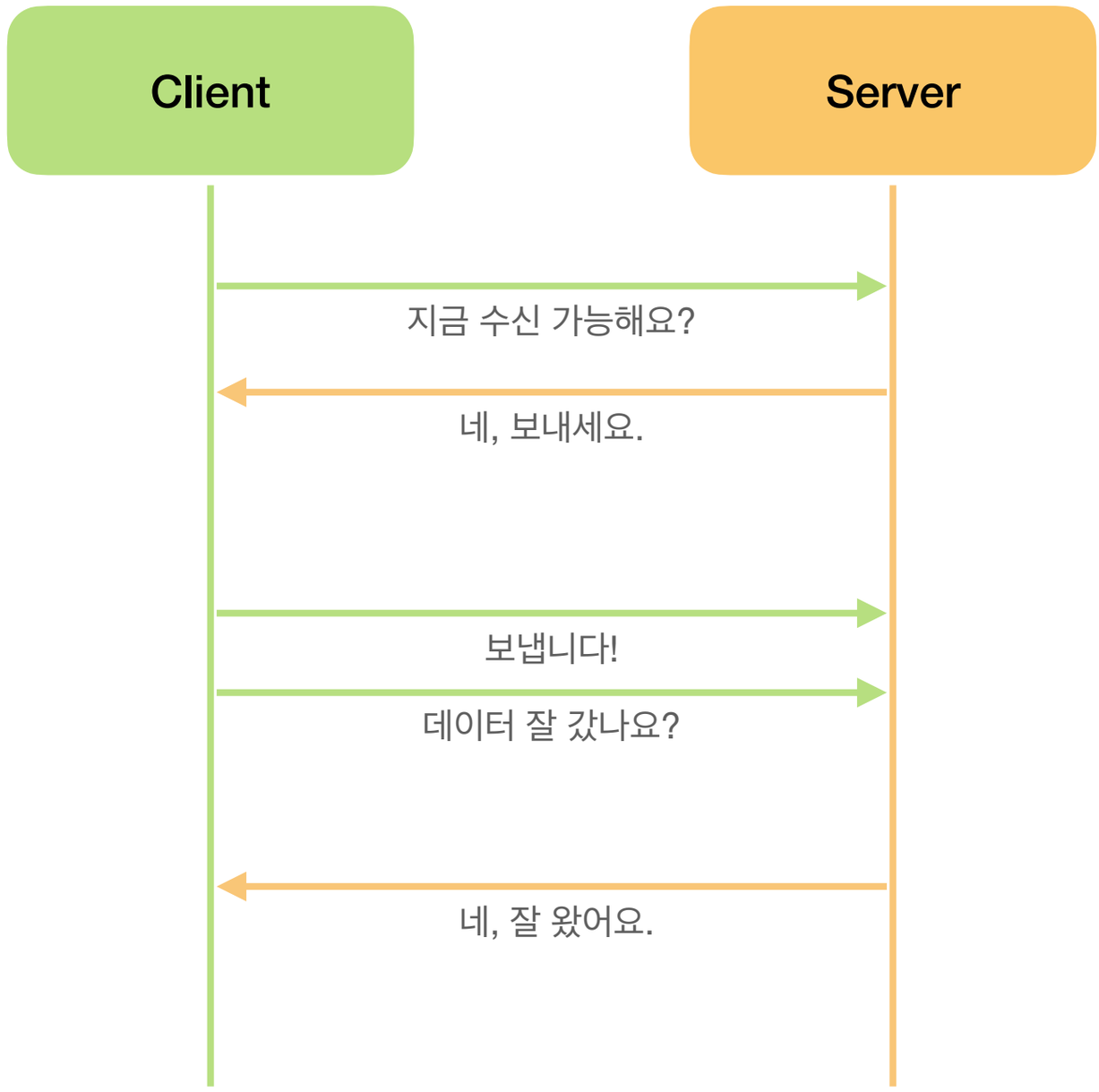
- Fast Recovery (빠른 회복)

  - 혼잡한 상태가 되면 window size를 1로 줄이지 않고 반으로 줄이고 선형증가시키는 방법이다.
  이 정책까지 적용하면 혼잡 상황을 한번 겪고 나서부터는 순수한 AIMD 방식으로 동작하게 된다.

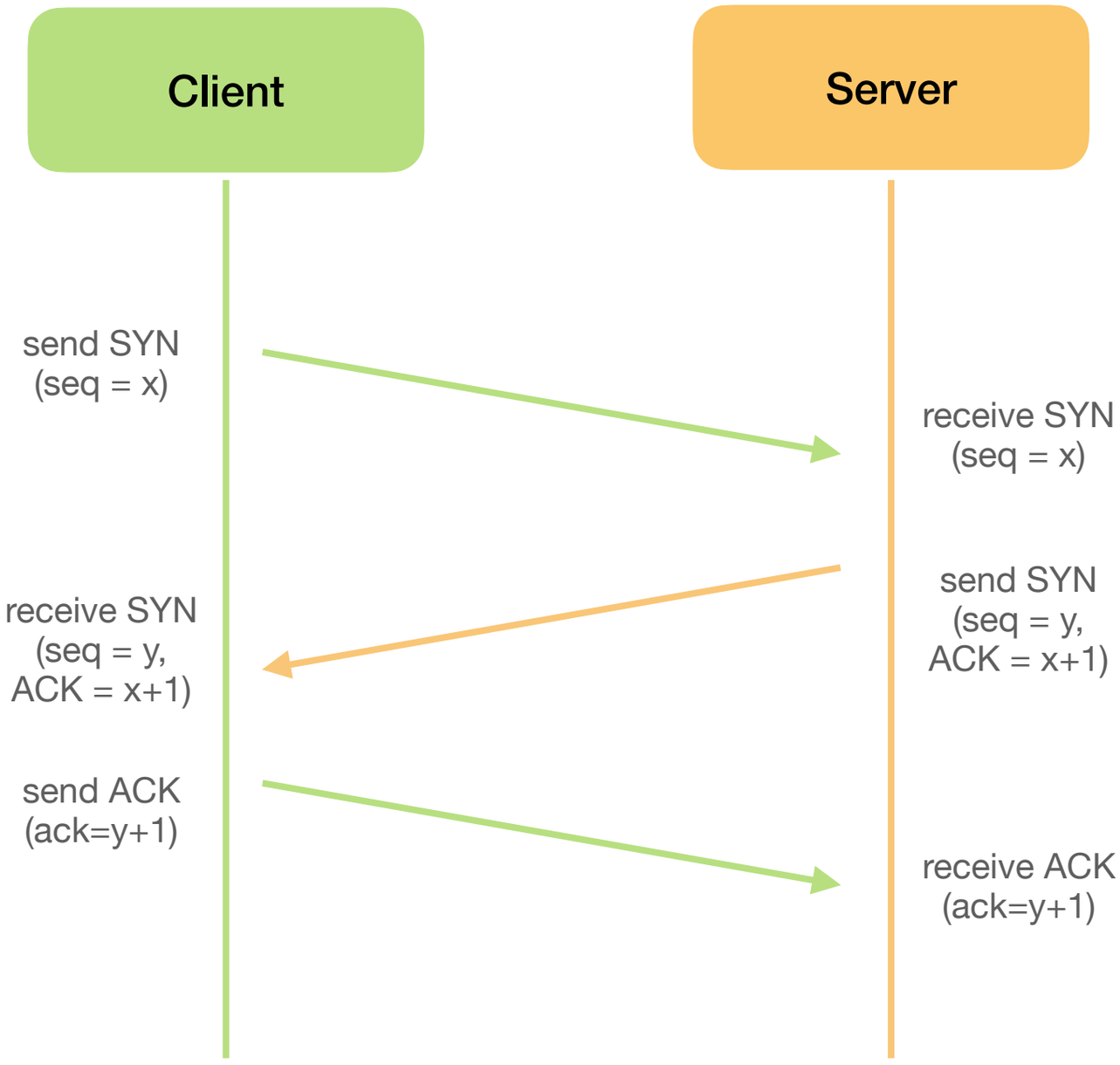
Thank You

Q & A

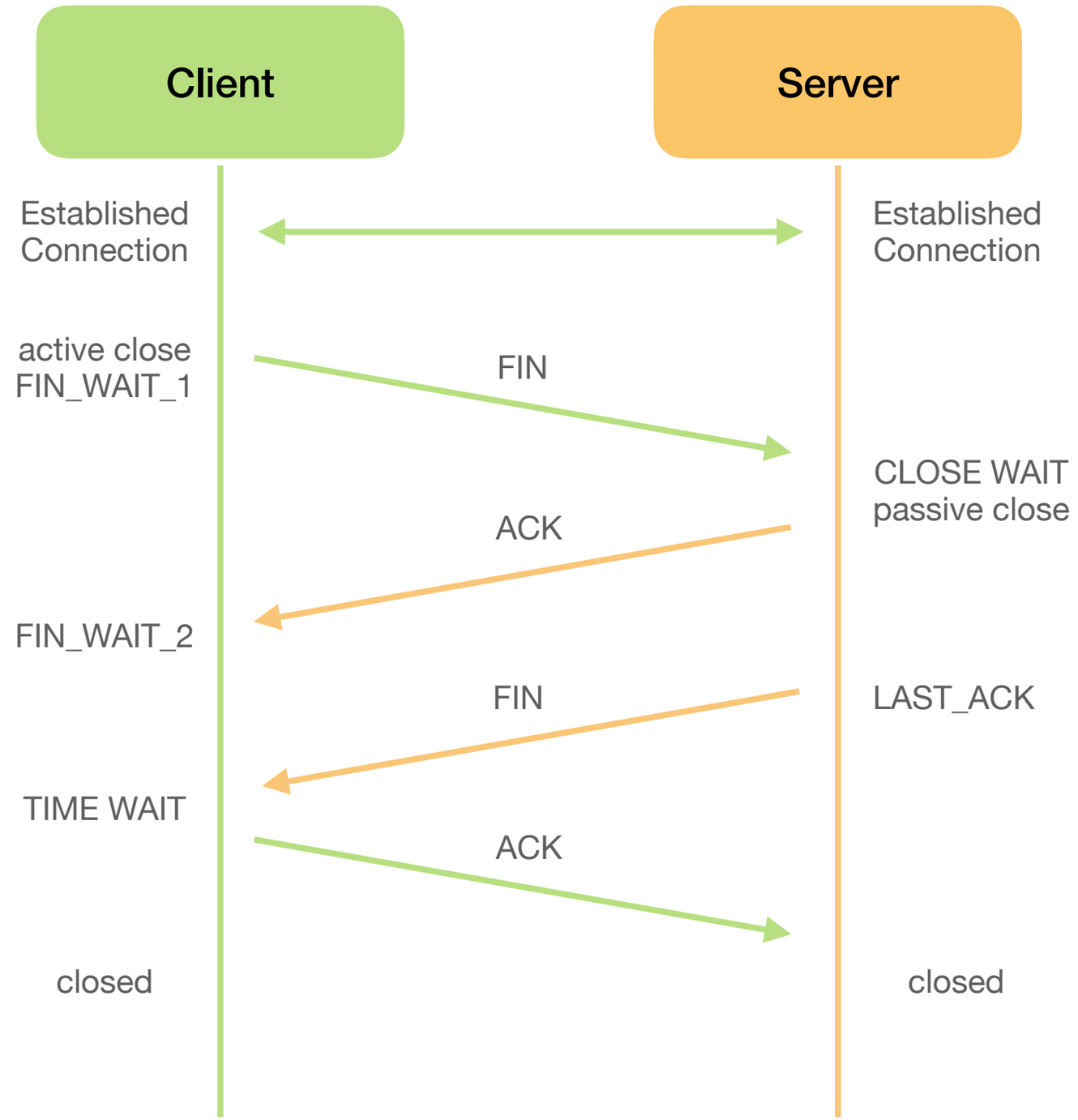
별첨



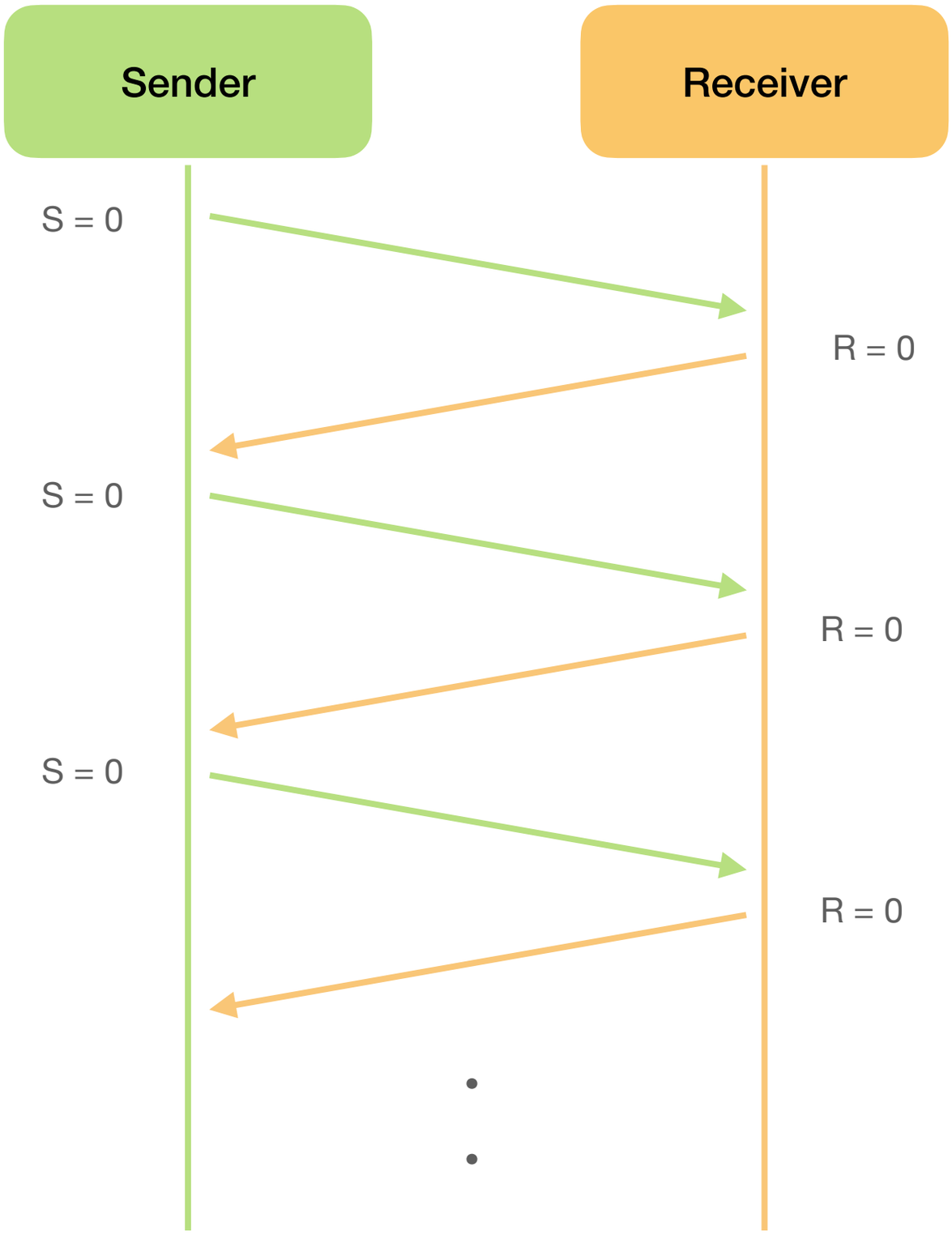
별첨


Client
Server
send SYN
(seq = x)
receive SYN
(seq = x)
receive SYN
(seq = y,
ACK = x+1)
send SYN
(seq = y,
ACK = x+1)
send ACK
(ack=y+1)
receive ACK
(ack=y+1)

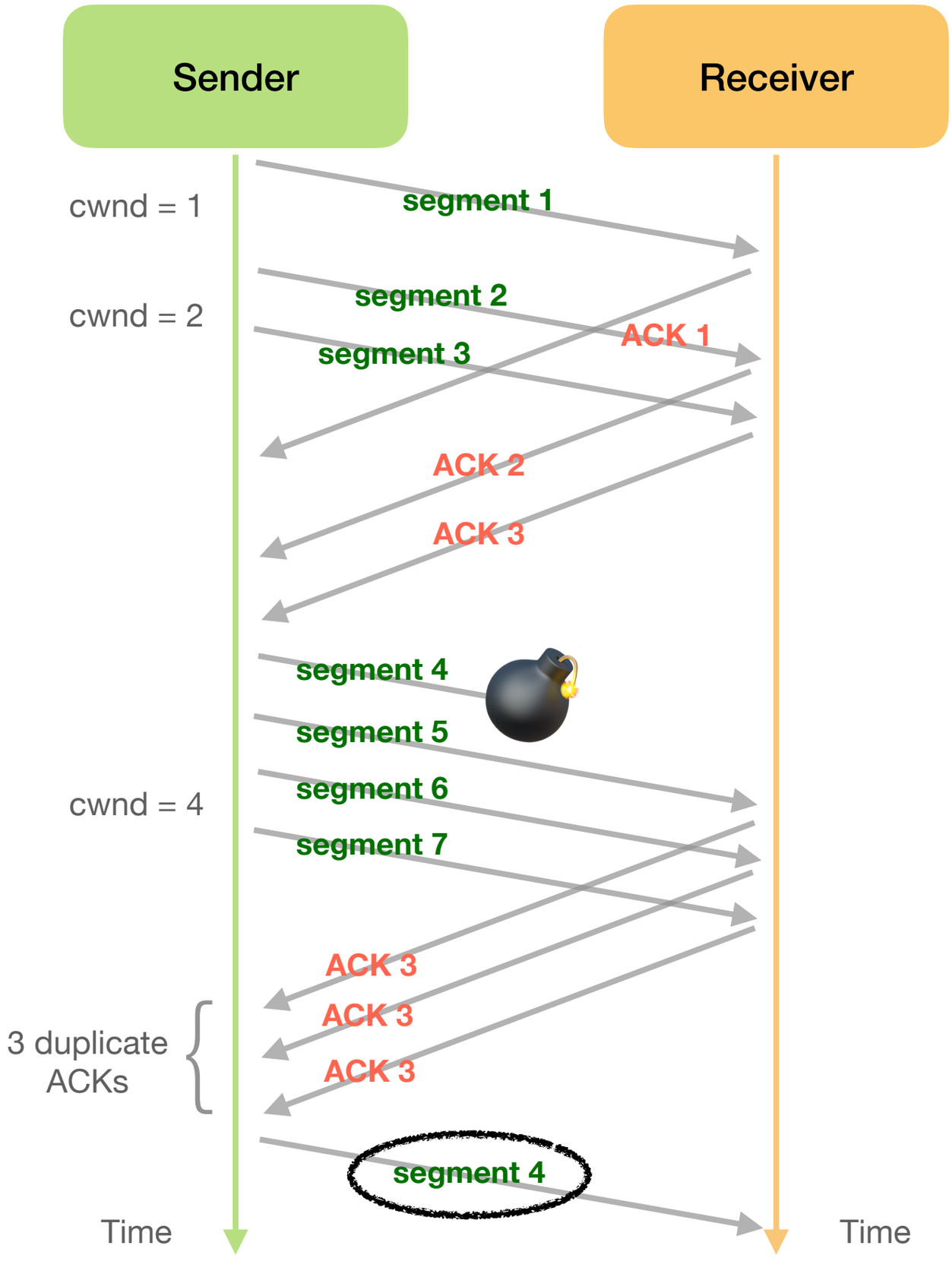
별첨


Client
Server
Established Connection
Established Connection
active close FIN_WAIT_1
FIN
CLOSE WAIT passive close
ACK
FIN_WAIT_2
FIN
LAST_ACK
TIME WAIT
ACK
closed
closed

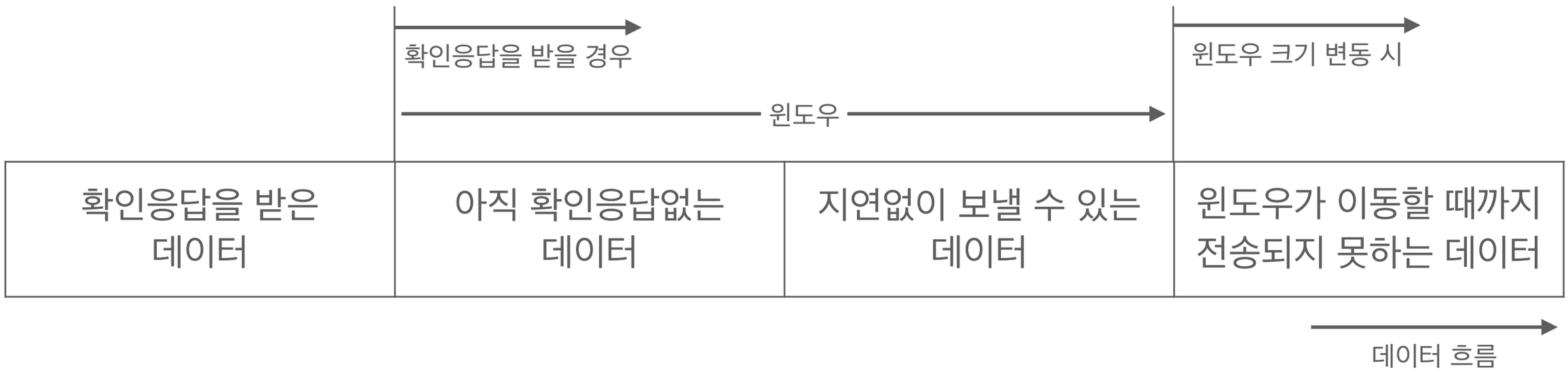
별첨



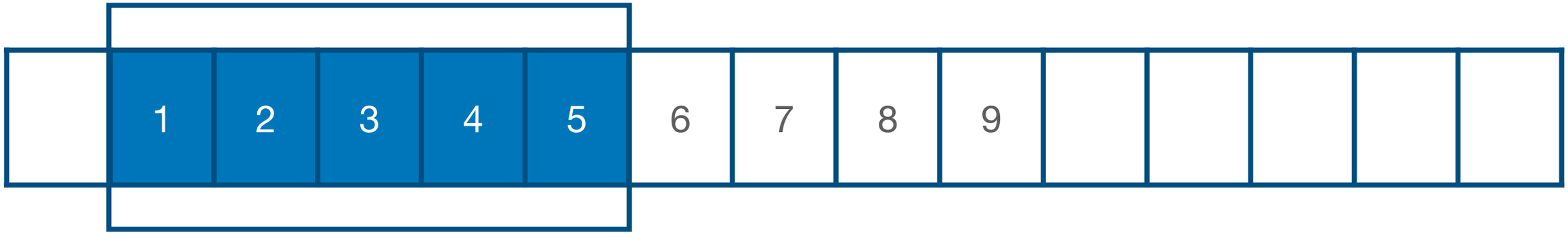
별첨

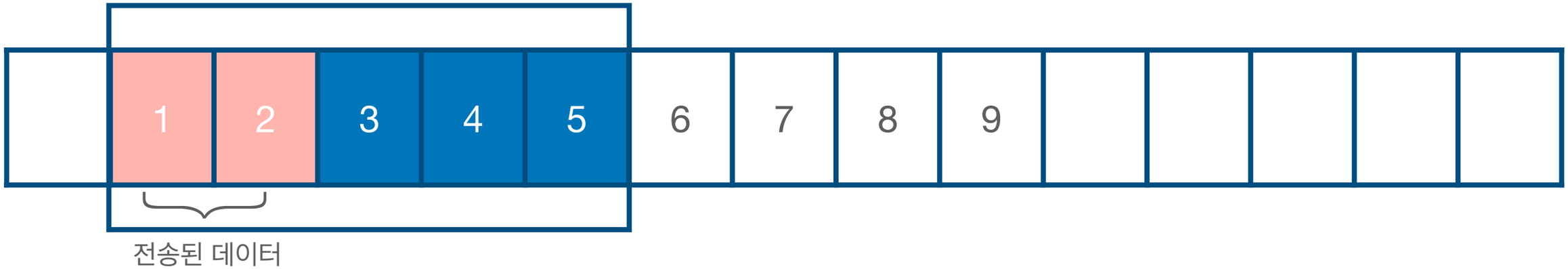

Sender
Receiver
cwnd = 1
segment 1
cwnd = 2
segment 2
segment 3
ACK 1
ACK 2
ACK 3
segment 4
segment 5
segment 6
segment 7
cwnd = 4
ACK 3
ACK 3
ACK 3
3 duplicate ACKs
segment 4
Time
Time

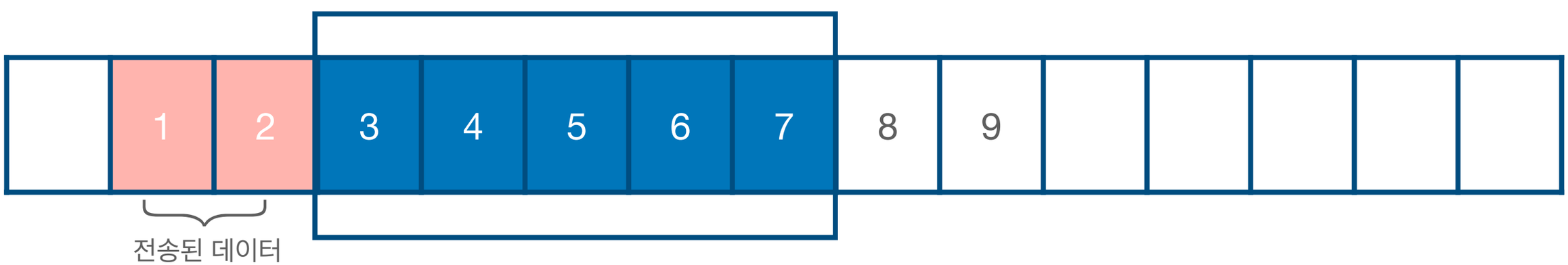
별첨


확인응답을 받을 경우
윈도우 크기 변동 시
윈도우
확인응답을 받은 데이터
아직 확인응답없는 데이터
지연없이 보낼 수 있는 데이터
윈도우가 이동할 때까지 전송되지 못하는 데이터
데이터 흐름

별첨







별첨

Application	• End User layer • HTTP, FTP, SMTP, POP3, IMAP, SSH, DNS	Data
Presentation	• Syntax layer • SSL, MPEG, JPEG	Data
Session	• Synch & send to port • API's, Sockets, WinSock	Data
Transport	• End-to-end-connections • TCP, UDP	Segments
Network	• Packets • IP, ICMP, IPsec, IGMP	Packets
Data Link	• Frames • Ethernet, PPP, Switch, Bridge	Frames
Physical	• Physical structure • Coax, Fiber, Wireless, Hubs, Repeaters	Bits