Introduction to Computer Networks

TCP — Connection Management

https://pages.cs.wisc.edu/~mgliu/CS640/F21/

Ming Liu
mgliu@cs.wisc.edu

Today

Last lecture

- TCP/UDP overview

Today

- TCP connection management

Announcements

- Lab3 due on 11/11/2021 at 11:59pm

How TCP solves the three issues of UDP

#1: Arbitrary communication

- Senders and receivers can talk to each other in any ways

#2: No reliability guarantee

- Packets can be lost/duplicated/reordered during transmission
- Checksum is not enough

#3: No resource management

- Each communication channel works as an exclusive network resource owner
- No adaptiveness support for the physical networks and applications

TCP Connection Management

#1: Arbitrary communication

- Senders and receivers can talk to each other in any ways

#2: No reliability guarantee

- Packets can be lost/duplicated/reordered during transmission
- Checksum is not enough

#3: No resource management

- Each communication channel works as an exclusive network resource owner
- No adaptiveness support for the physical networks and applications

The Goal of Connection Management

Dynamically create and destroy a full-duplex communication channel between a sender process and a receiver process for reliable byte stream exchange

The Goal of Connection Management

Dynamically create and destroy a full-duplex communication channel between a sender process and a receiver process for reliable byte stream exchange

- Connection establishment

- Connection termination

Why this is non-trivial?

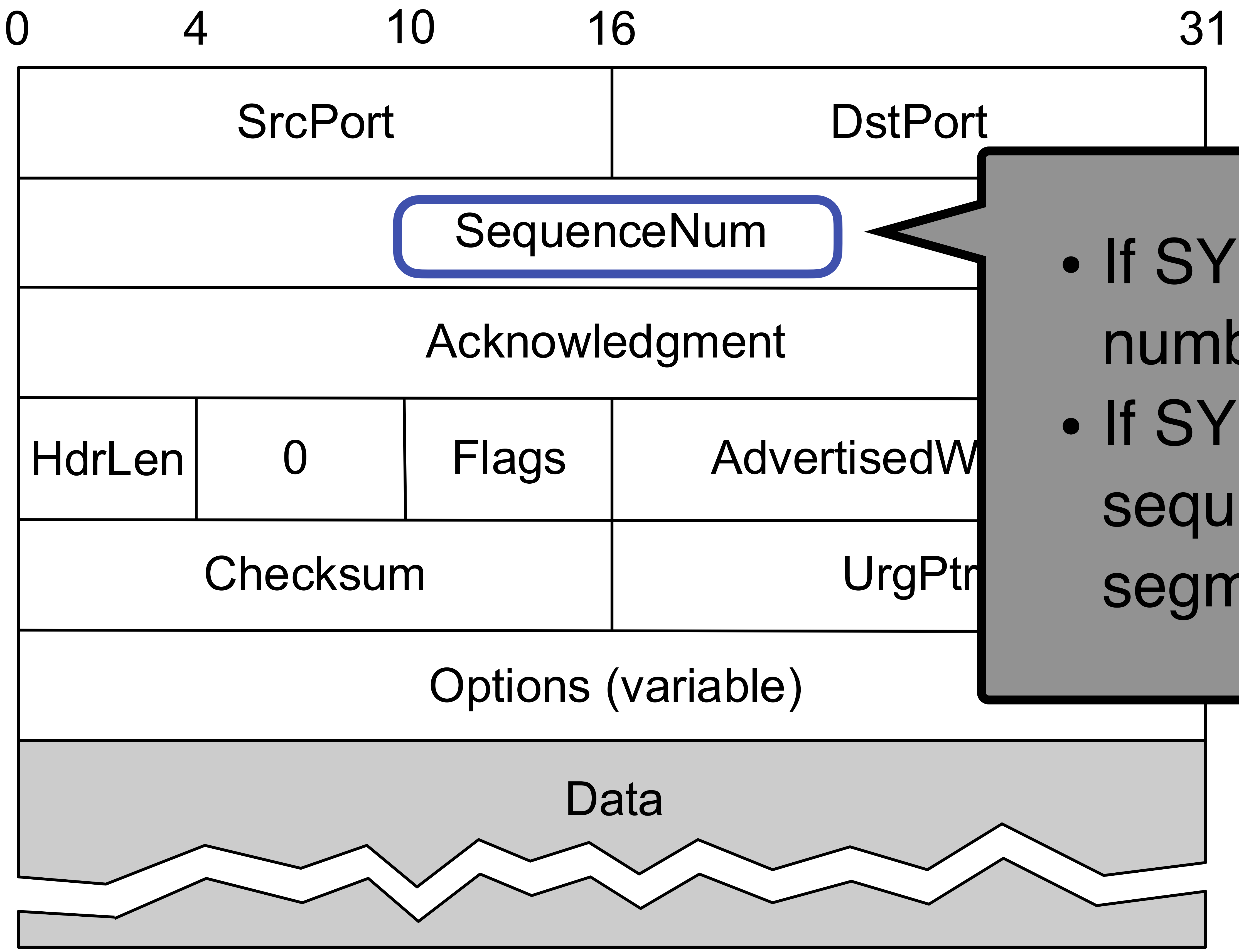
Dynamically create and destroy a full-duplex communication channel between a sender process and a receiver process for reliable byte stream exchange

- On-demand communication

- Client <–> Server

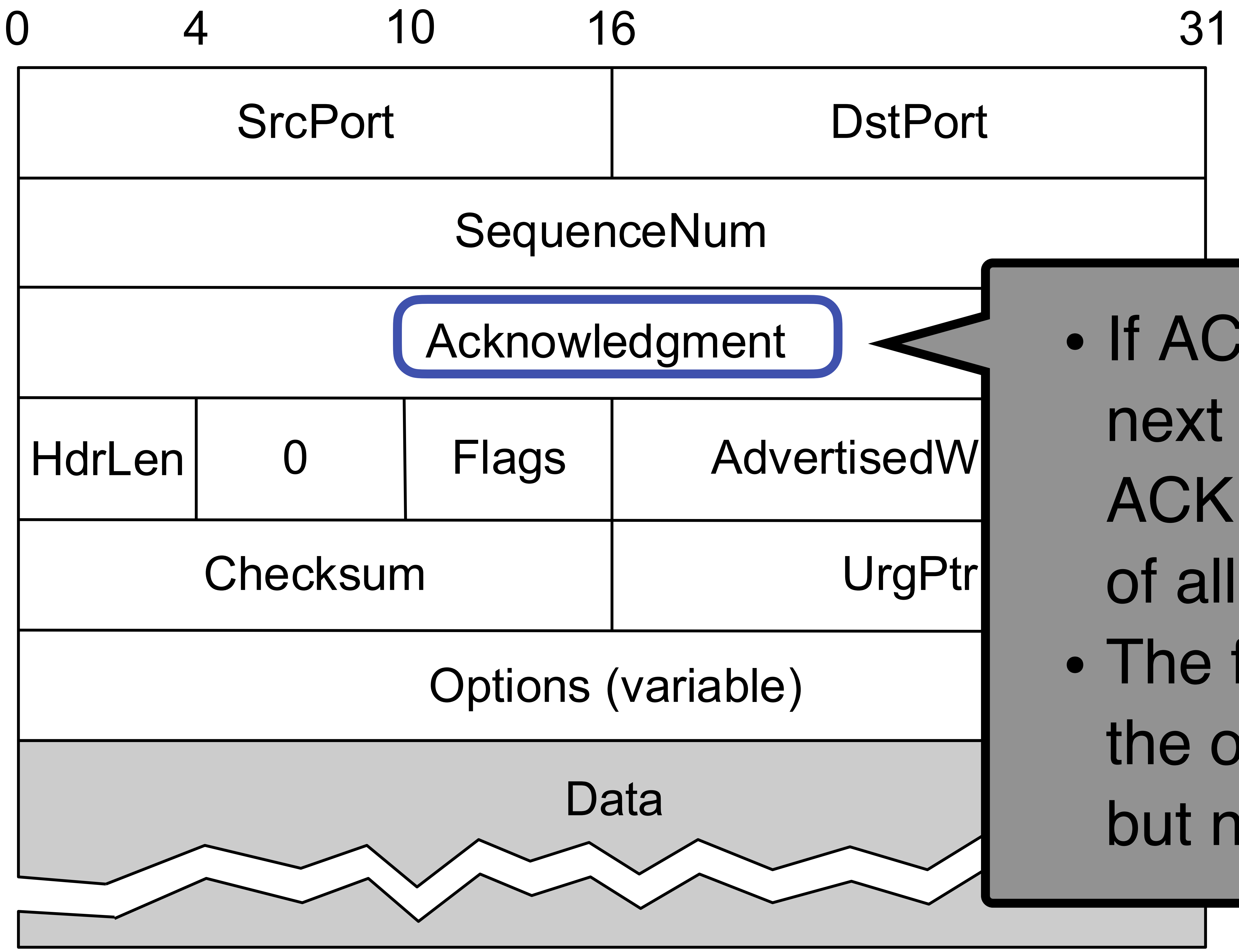
- Client and server agree on the start of byte steams for two directions

Revisit the TCP header



- If SYN flag is set, this is the initial sequence number. The start of a byte stream;
- If SYN flag is clear, this is the accumulated sequence number of the first data byte of this segment for the current session;

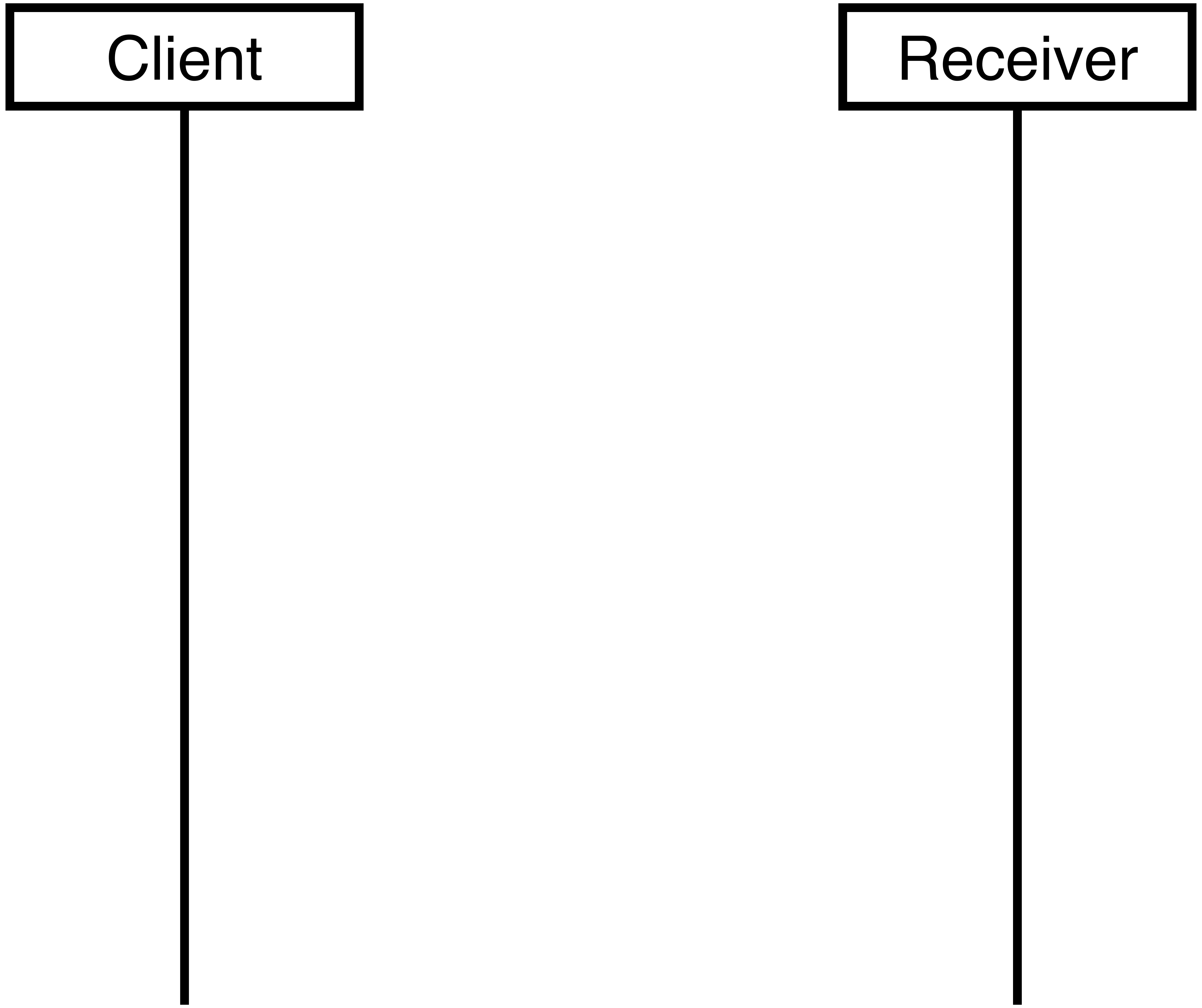
Revisit the TCP header



- If ACK flag is set, the value of this field is the next sequence number that the sender of the ACK is expecting. This acknowledges receipt of all prior bytes (if any)
- The first ACK sent by each end acknowledges the other ends’s initial sequence number itself, but no data

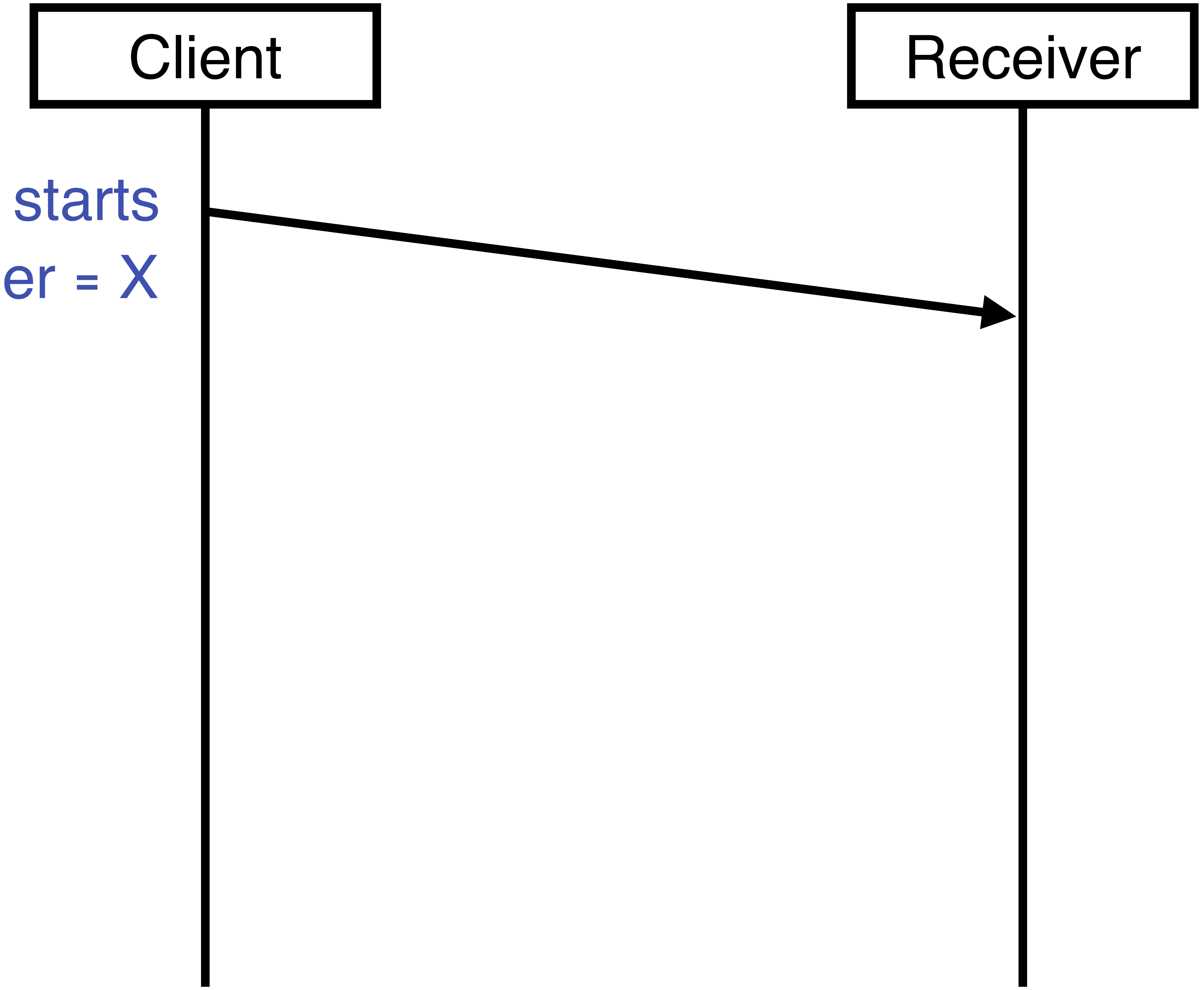
TCP Connection Establishment

Let’s start with a naive approach



TCP Connection Establishment

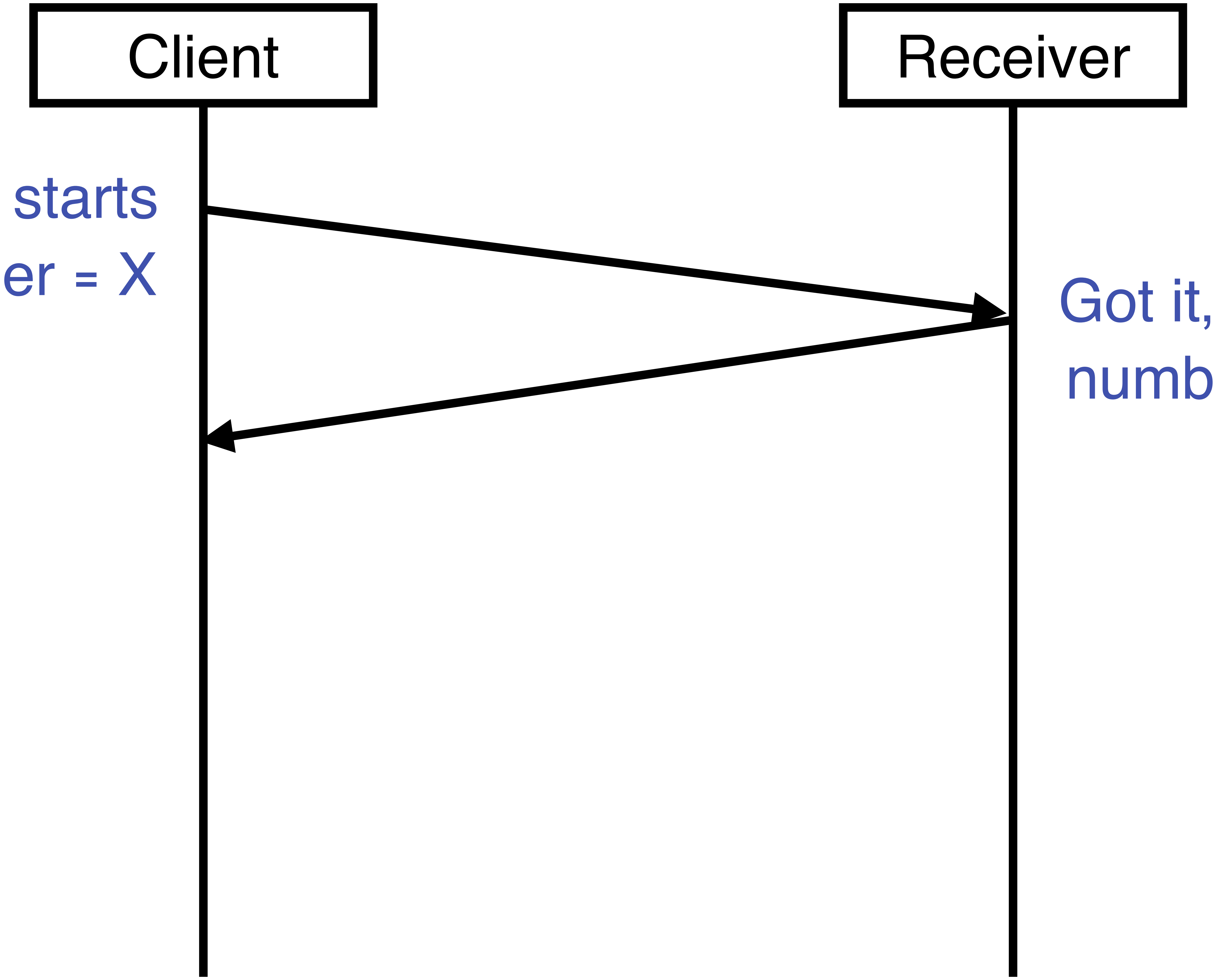
Let’s start with a naive approach



My (client) byte stream starts with a sequence number = X

TCP Connection Establishment

Let’s start with a naive approach

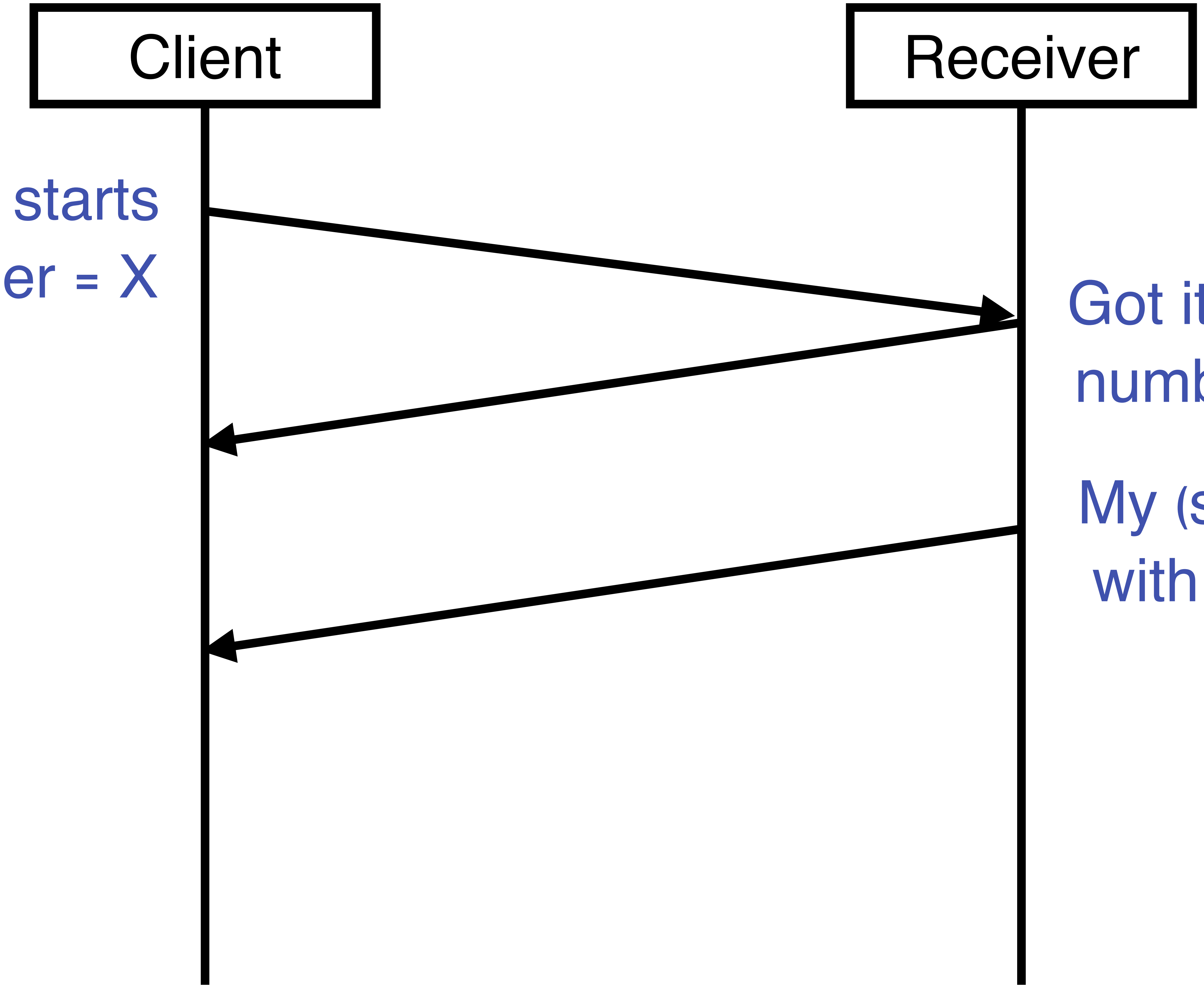


My (client) byte stream starts with a sequence number = X

Got it, I acknowledge the sequence number of your next byte is = X + 1

TCP Connection Establishment

Let’s start with a naive approach



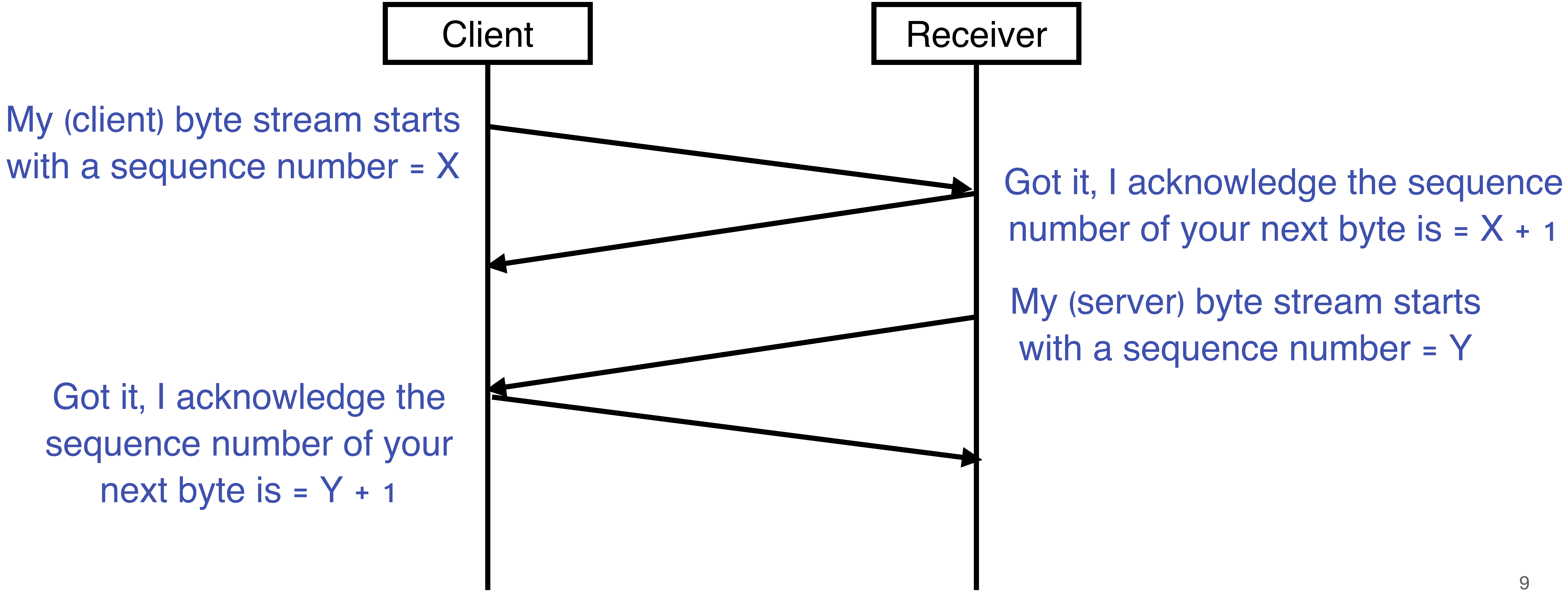
My (client) byte stream starts with a sequence number = X

Got it, I acknowledge the sequence number of your next byte is = X + 1

My (server) byte stream starts with a sequence number = Y

TCP Connection Establishment

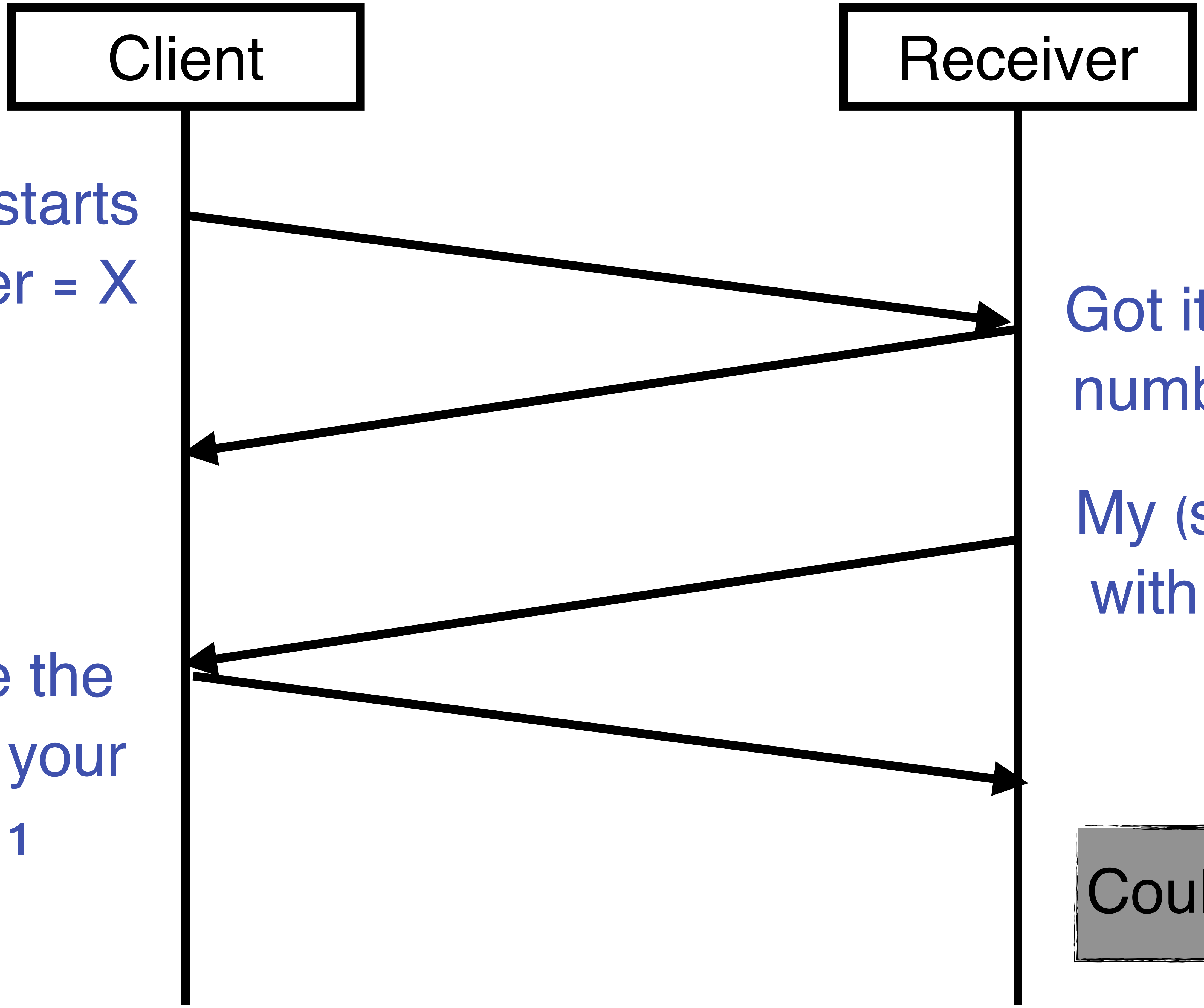
Let’s start with a naive approach



Client
Receiver
My (client) byte stream starts with a sequence number = X
Got it, I acknowledge the sequence number of your next byte is = X + 1
My (server) byte stream starts with a sequence number = Y
Got it, I acknowledge the sequence number of your next byte is = Y + 1

TCP Connection Establishment

Let’s start with a naive approach



Client

Receiver

My (client) byte stream starts with a sequence number = X

Got it, I acknowledge the sequence number of your next byte is = X + 1

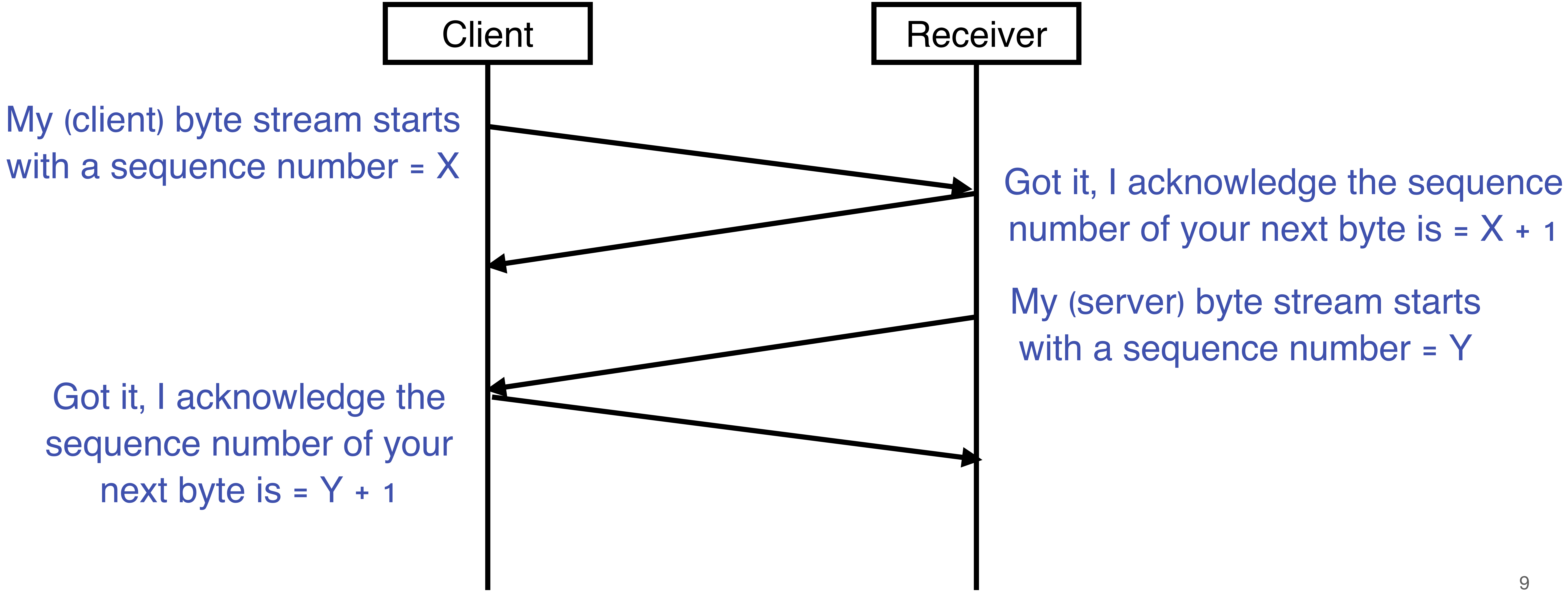
My (server) byte stream starts with a sequence number = Y

Got it, I acknowledge the sequence number of your next byte is = Y + 1

Could we optimize a little bit?

TCP Connection Establishment

Let’s start with a naive approach



My (client) byte stream starts with a sequence number = X

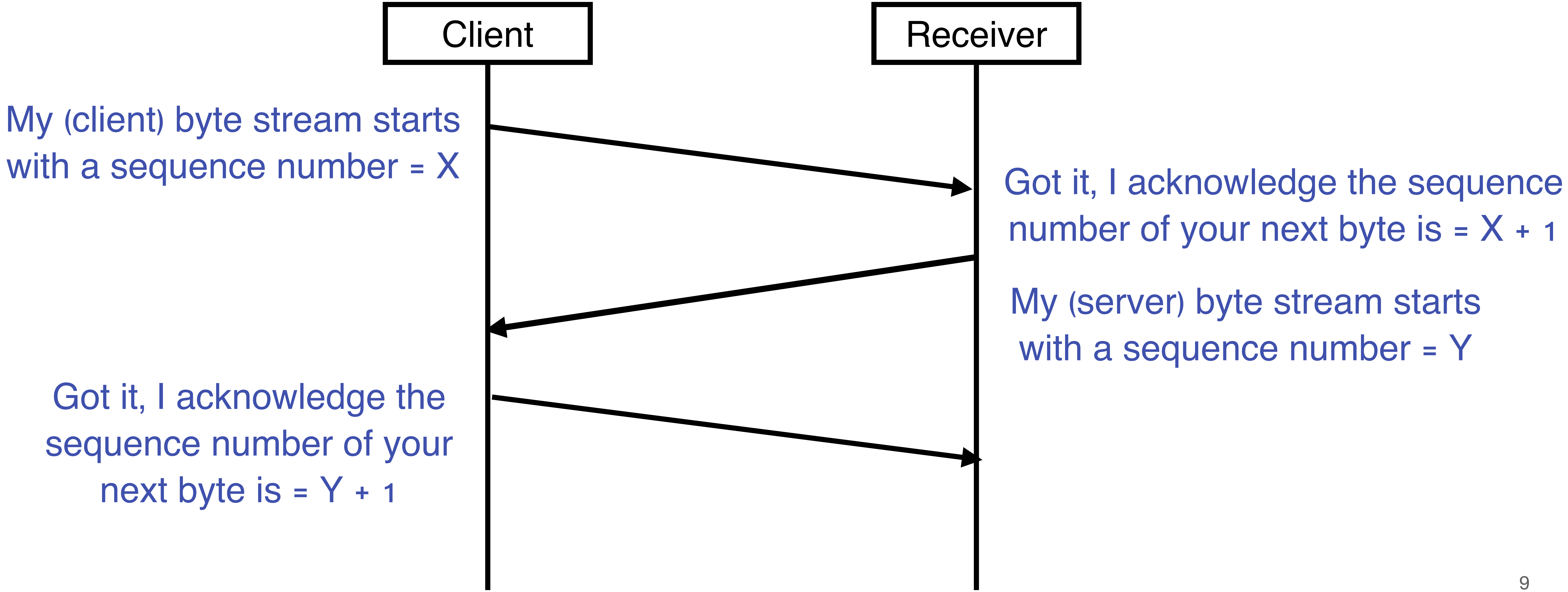
Got it, I acknowledge the sequence number of your next byte is = X + 1

My (server) byte stream starts with a sequence number = Y

Got it, I acknowledge the sequence number of your next byte is = Y + 1

TCP Connection Establishment

Let’s start with a naive approach



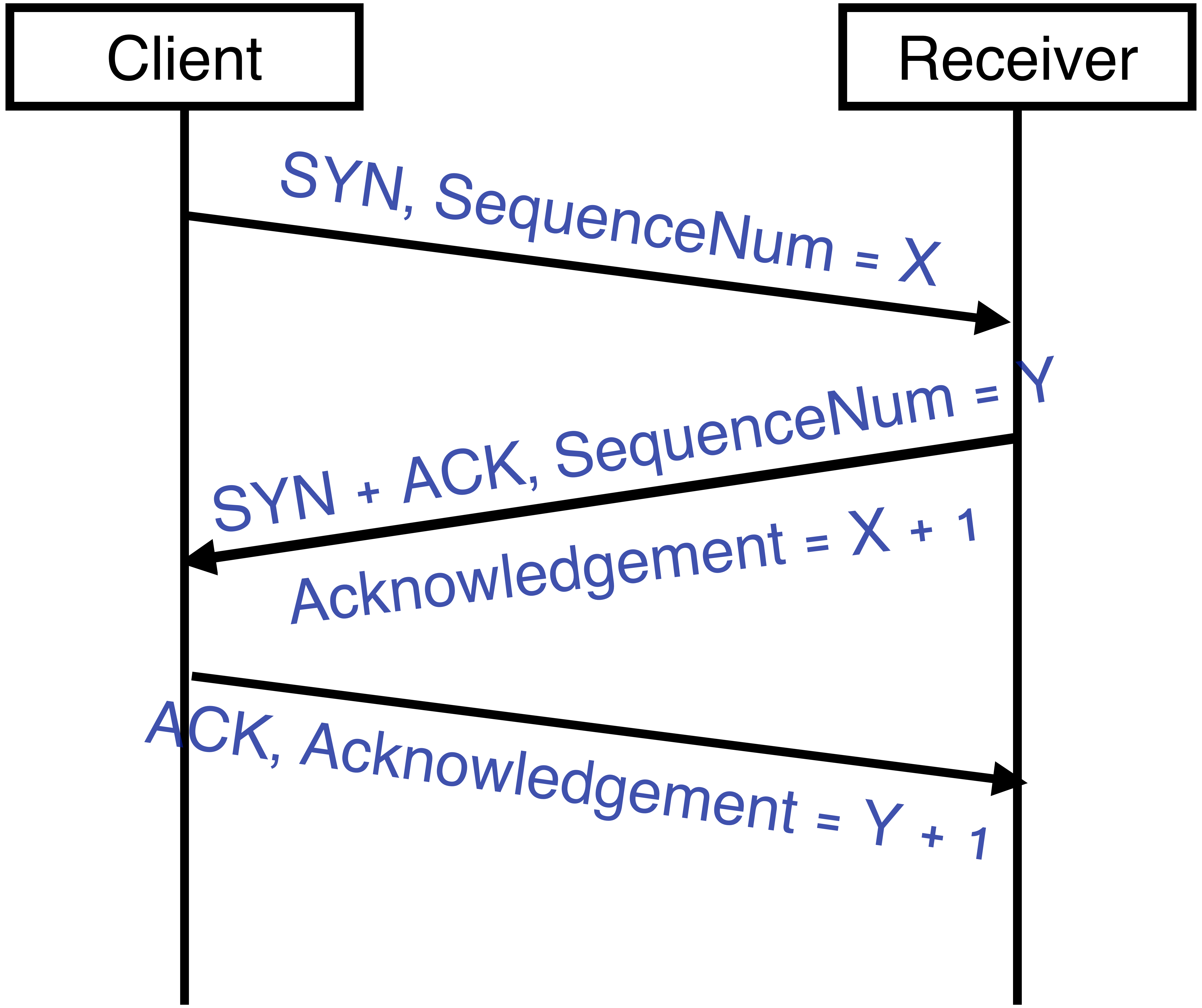
My (client) byte stream starts with a sequence number = X

Got it, I acknowledge the sequence number of your next byte is = X + 1

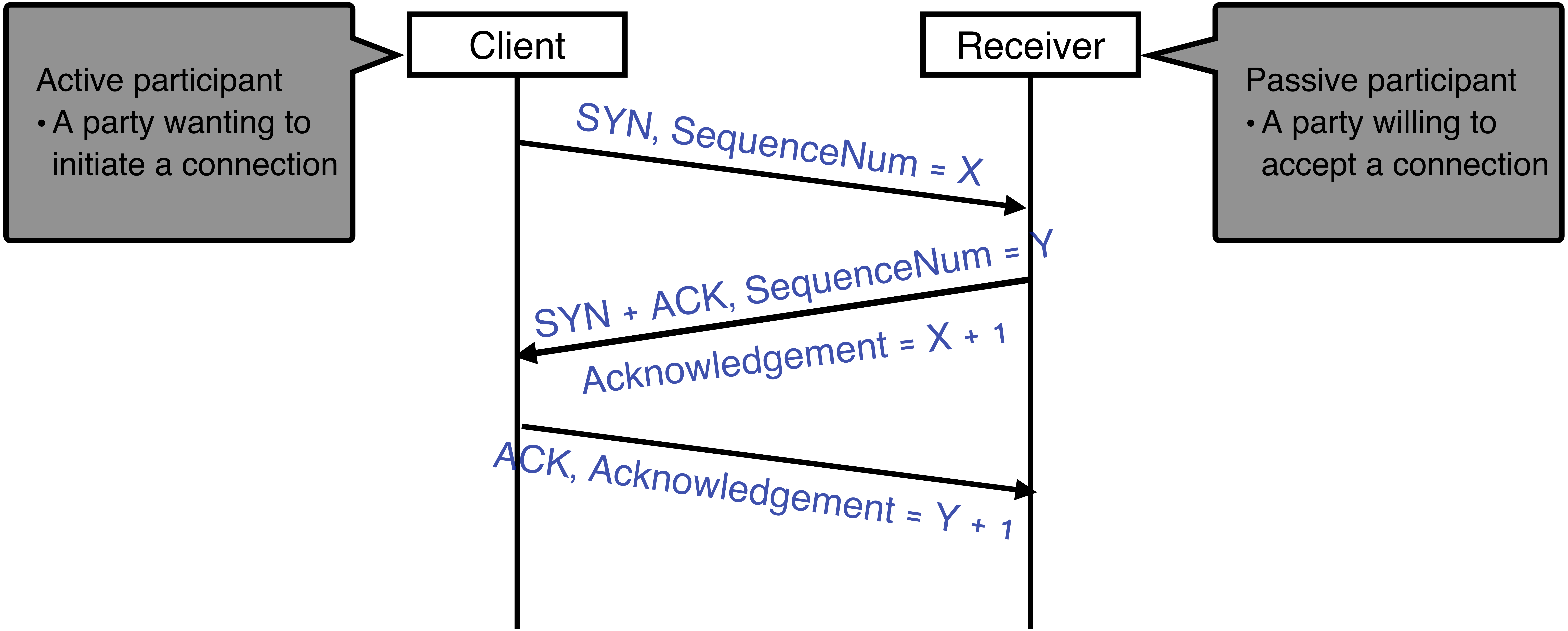
My (server) byte stream starts with a sequence number = Y

Got it, I acknowledge the sequence number of your next byte is = Y + 1

Three-Way Handshake



Three-Way Handshake



Active participant

- A party wanting to initiate a connection

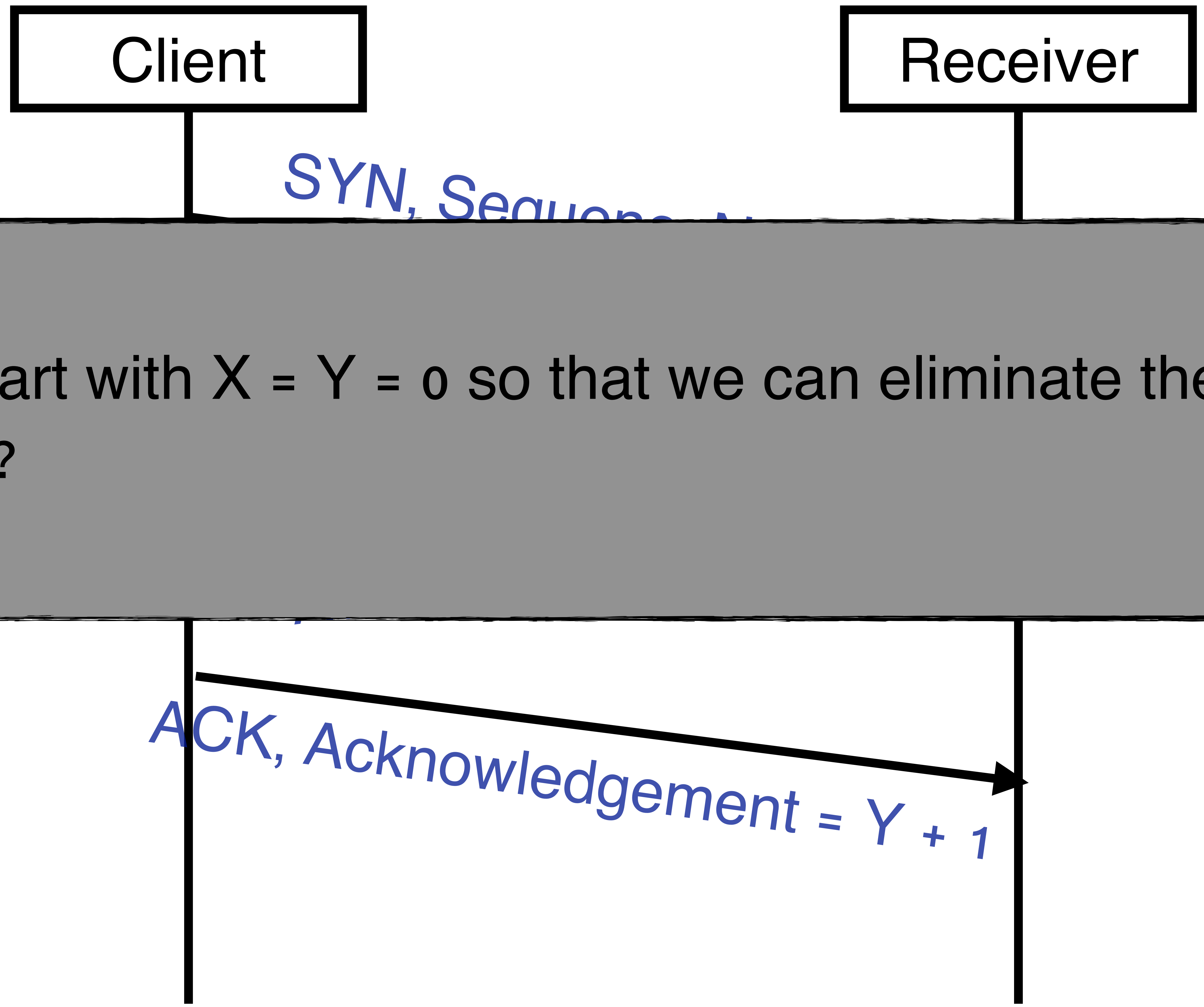
Client

Receiver

Passive participant

- A party willing to accept a connection

Three-Way Handshake



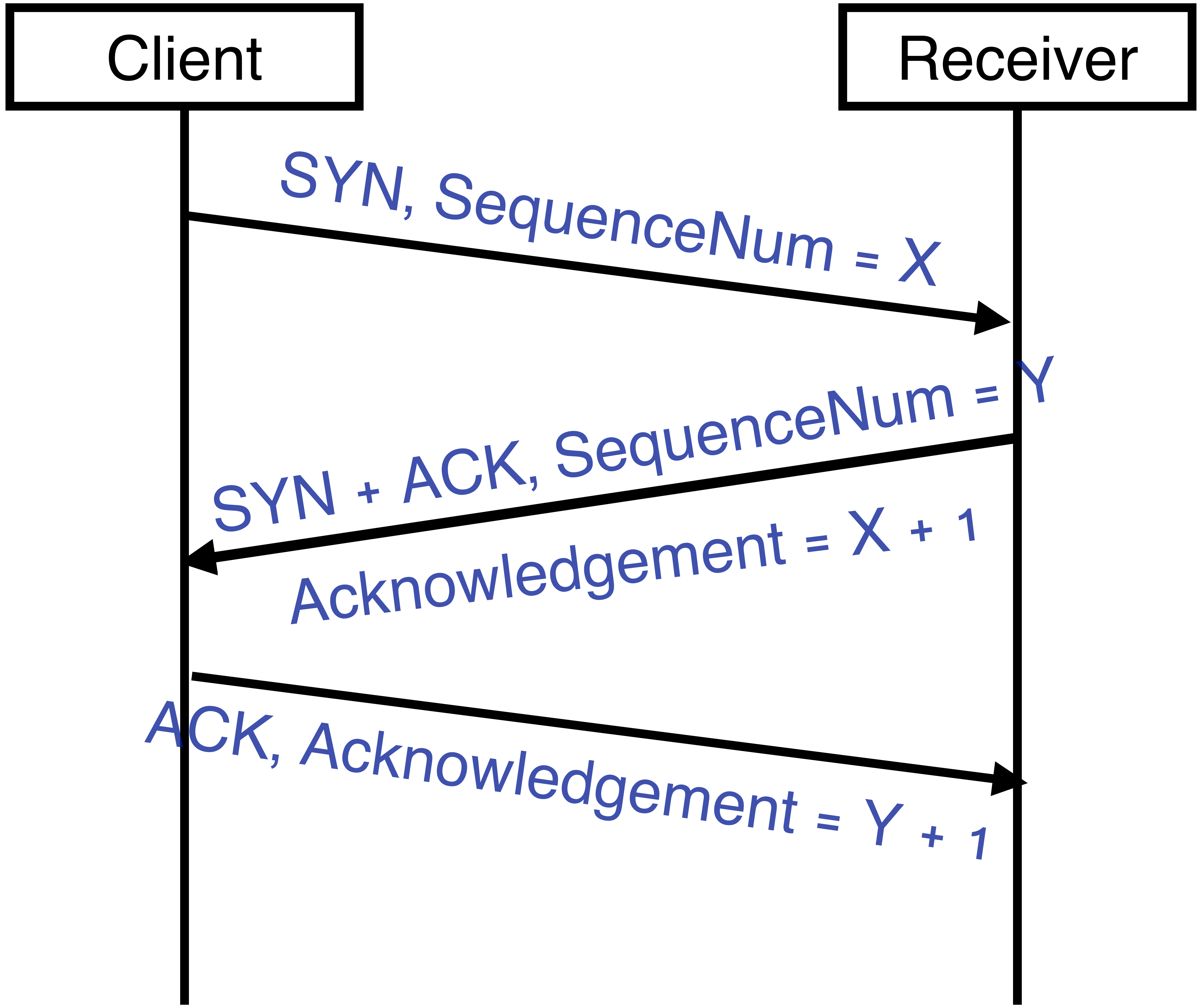
Why not start with X = Y = 0 so that we can eliminate the three-way handshake?

The Incarnation Issue

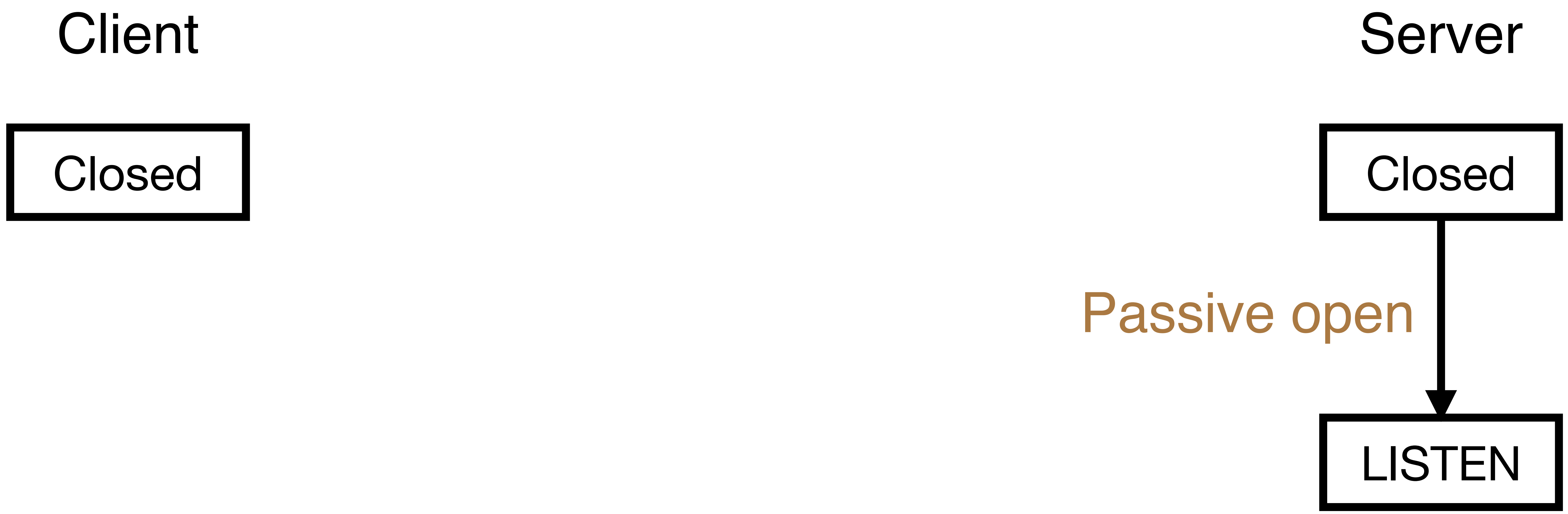
A connection (defined by a particular host and port pair) to be reused again

Solution: initial sequence number is randomly generated

How to implement this?



State Machine (event/action)



Client

Closed

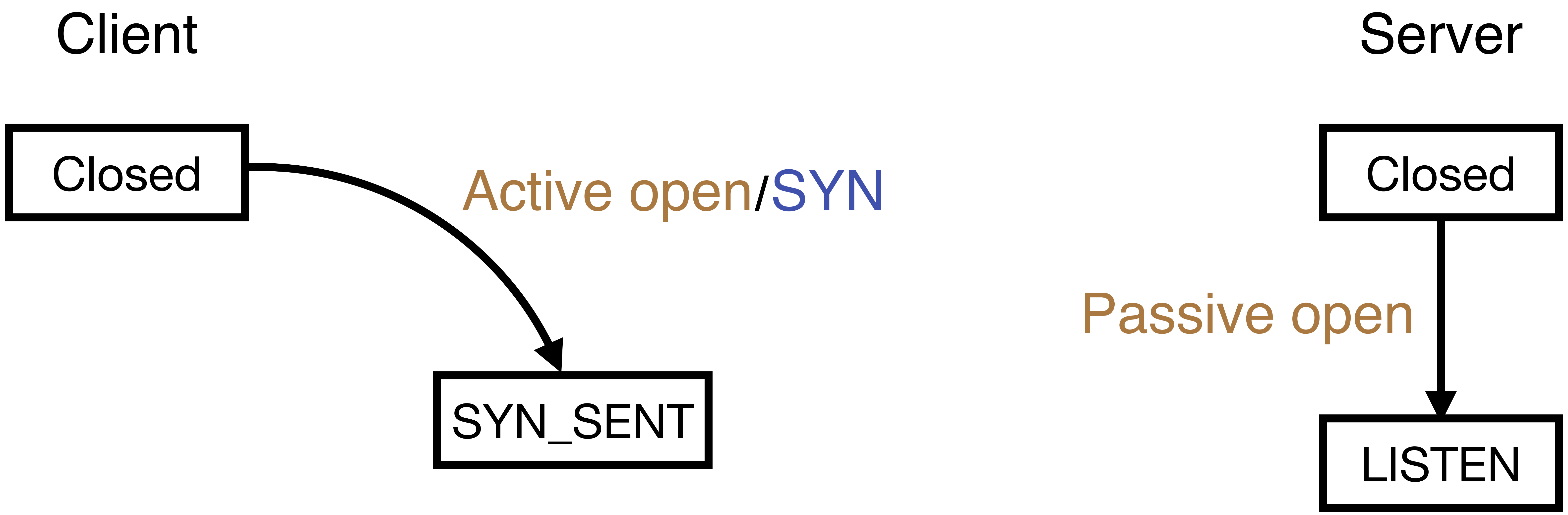
Server

Closed

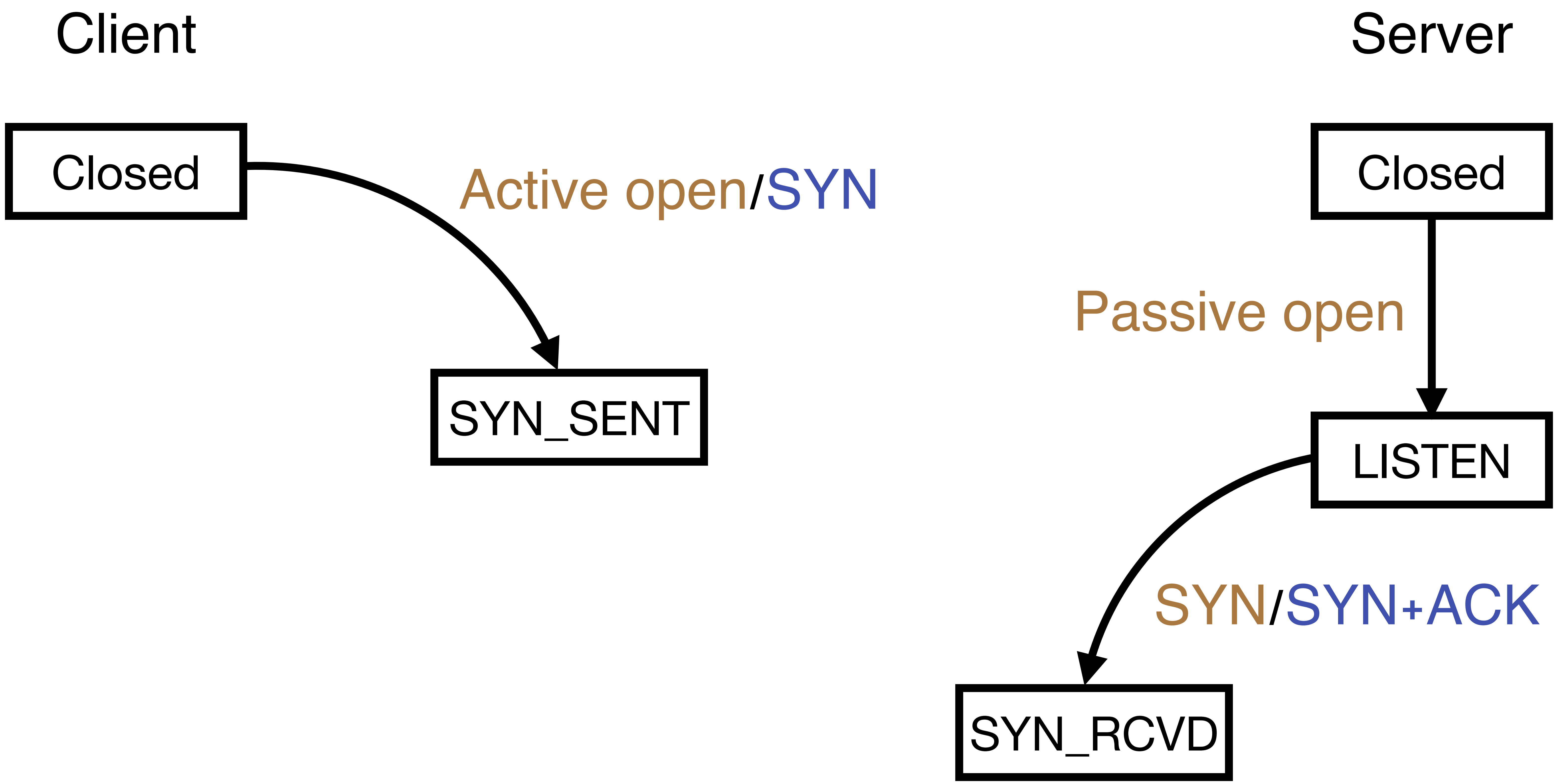
Passive open

LISTEN

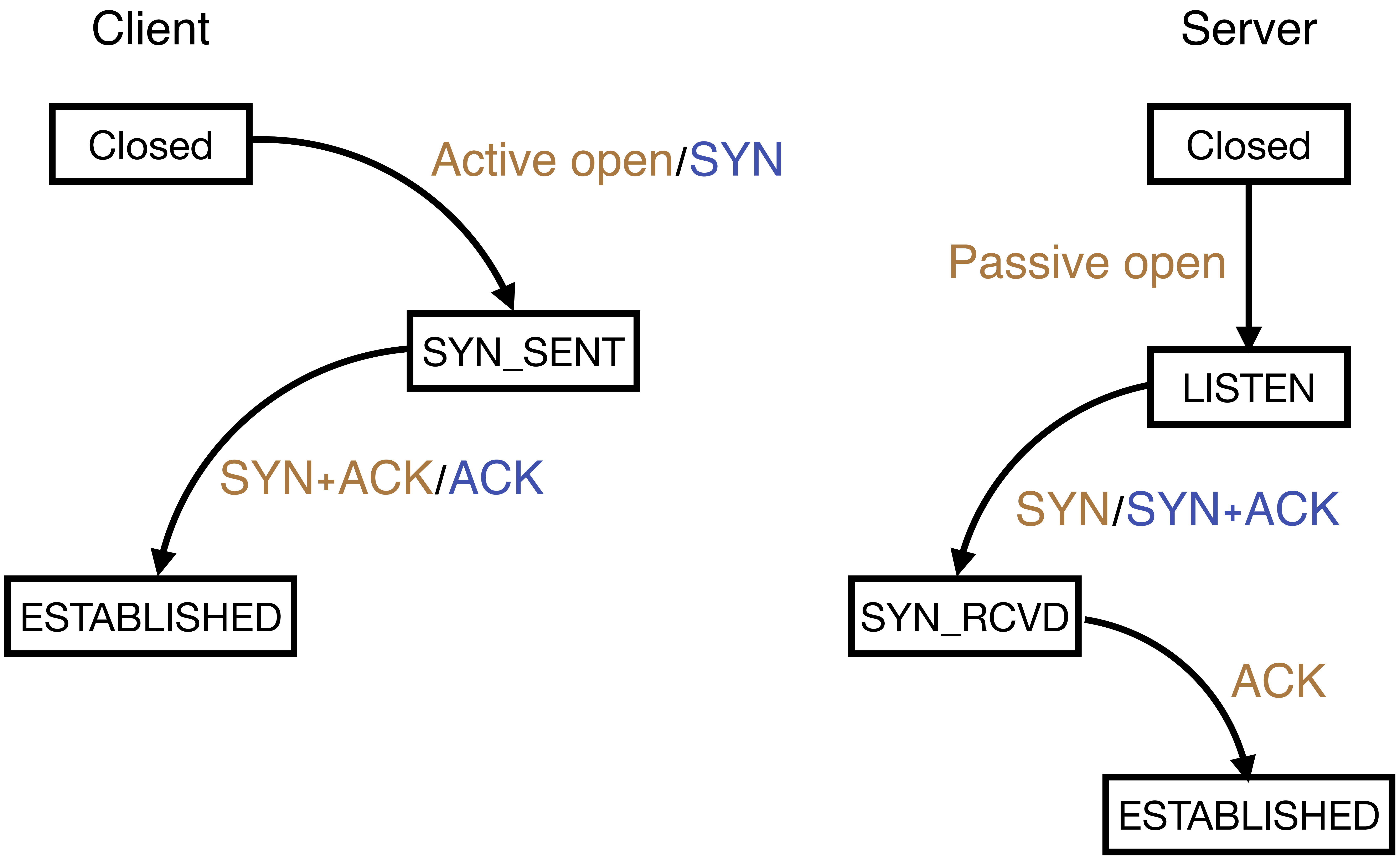
State Machine (Step 1)



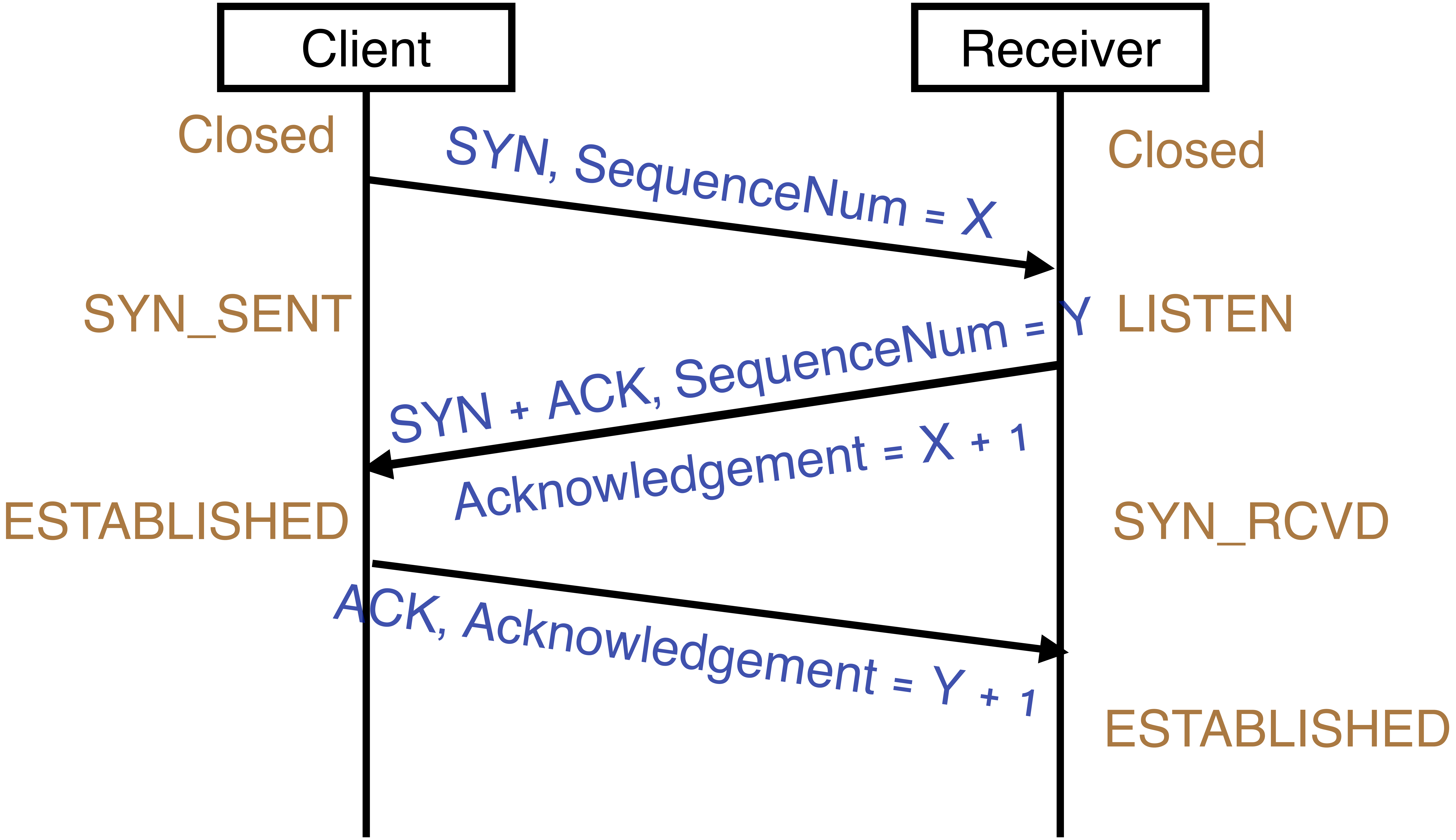
State Machine (Step 2)



State Machine (Step 3)



TCP Connection Establishment Summary



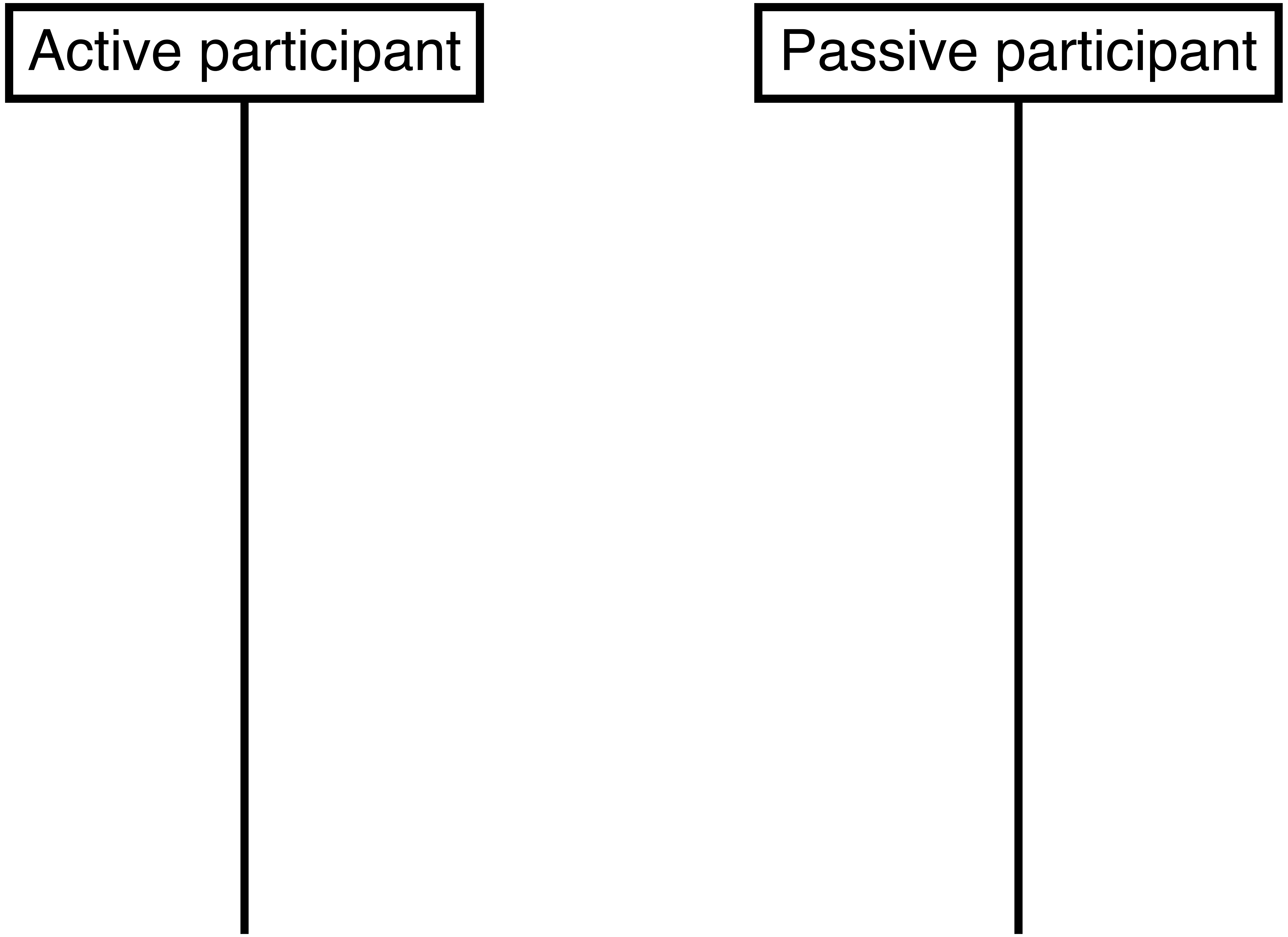
Connection Termination

Three cases:

- Case #1: One-side closes first
- Case #2: Both sides close simultaneously
- Case #3: Both sides close simultaneously (special)

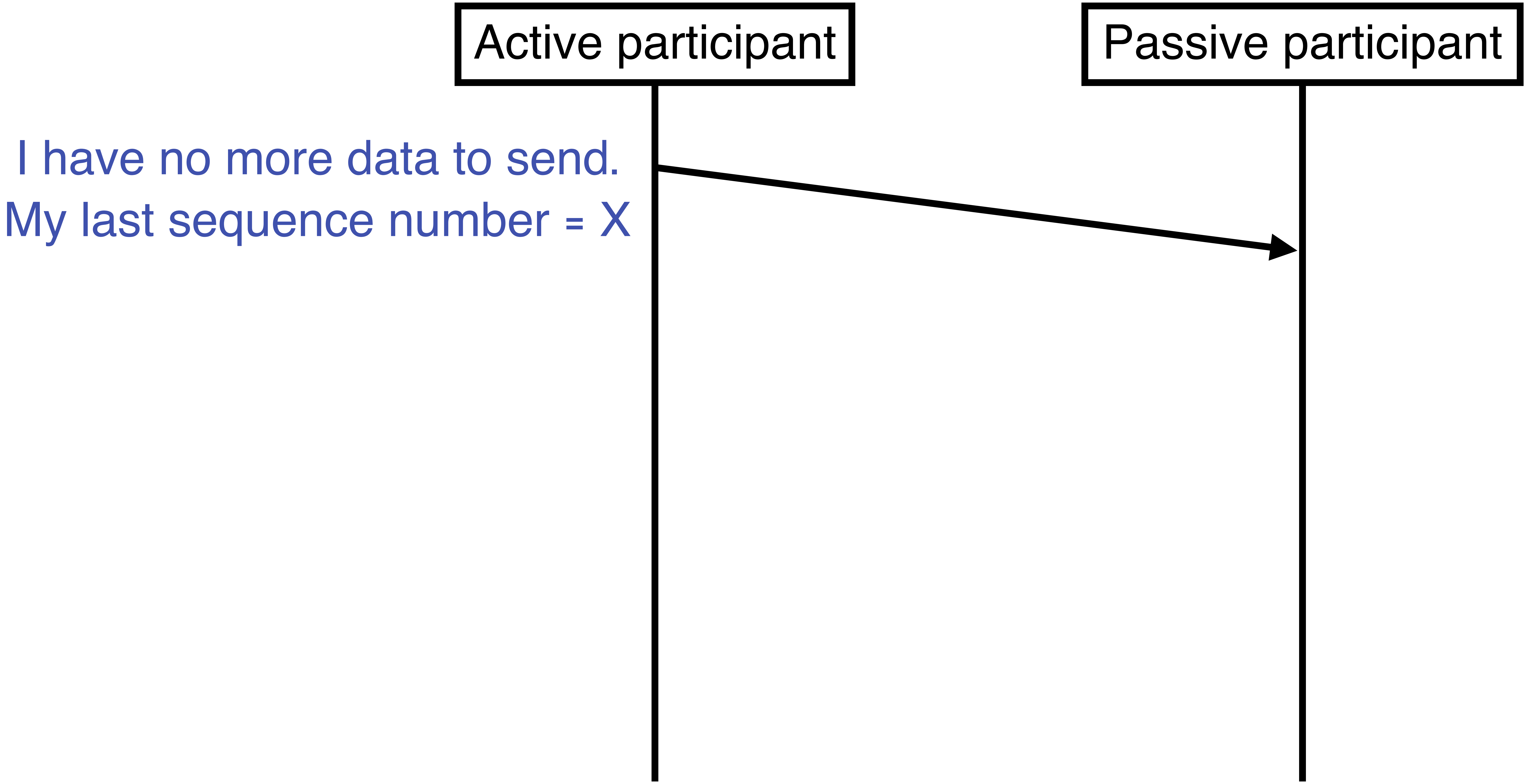
Case 1: One-side Closes First

4-way handshake



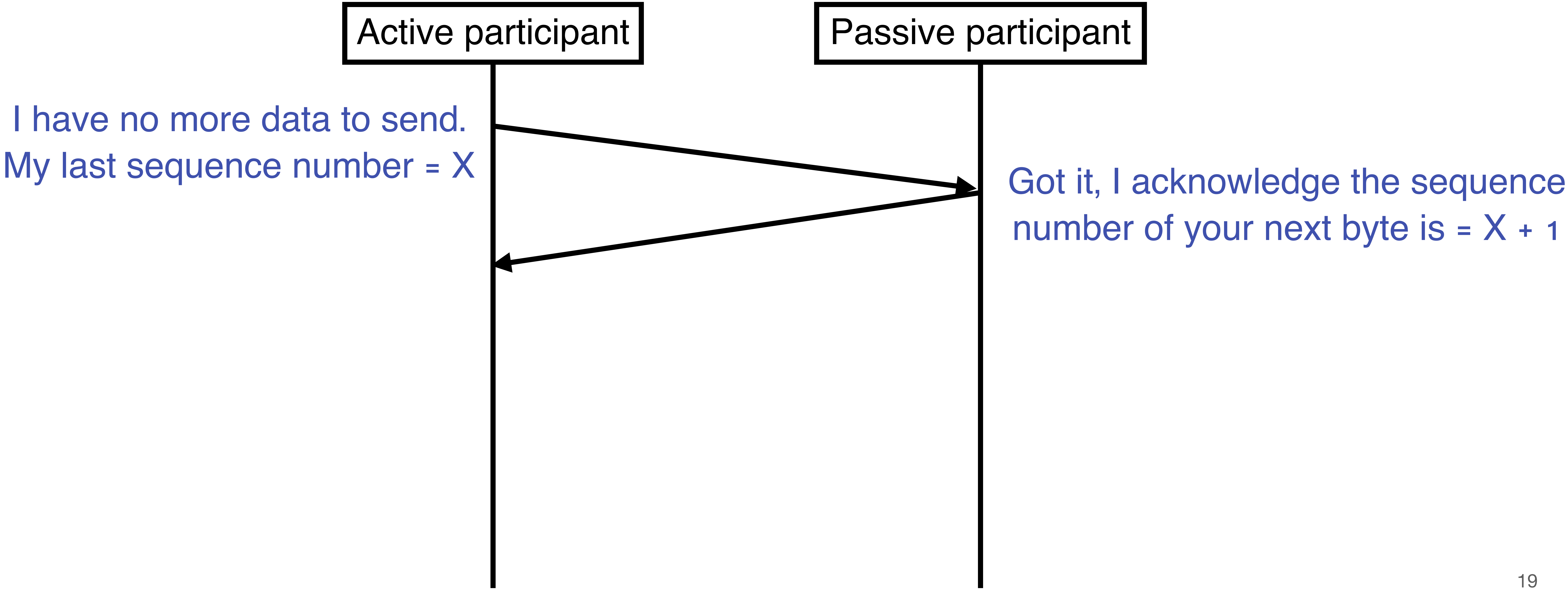
Case 1: One-side Closes First

4-way handshake



Case 1: One-side Closes First

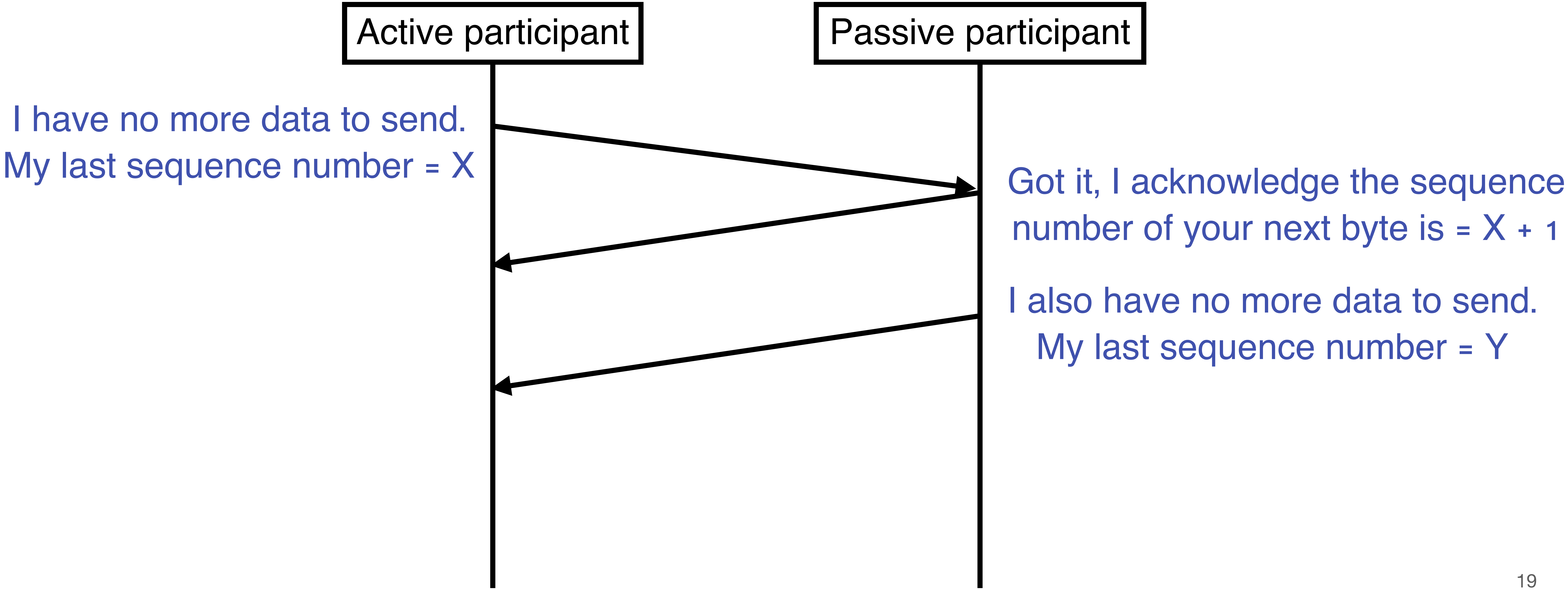
4-way handshake



Active participant
Passive participant
I have no more data to send.
My last sequence number = X
Got it, I acknowledge the sequence
number of your next byte is = X + 1

Case 1: One-side Closes First

4-way handshake



Active participant

Passive participant

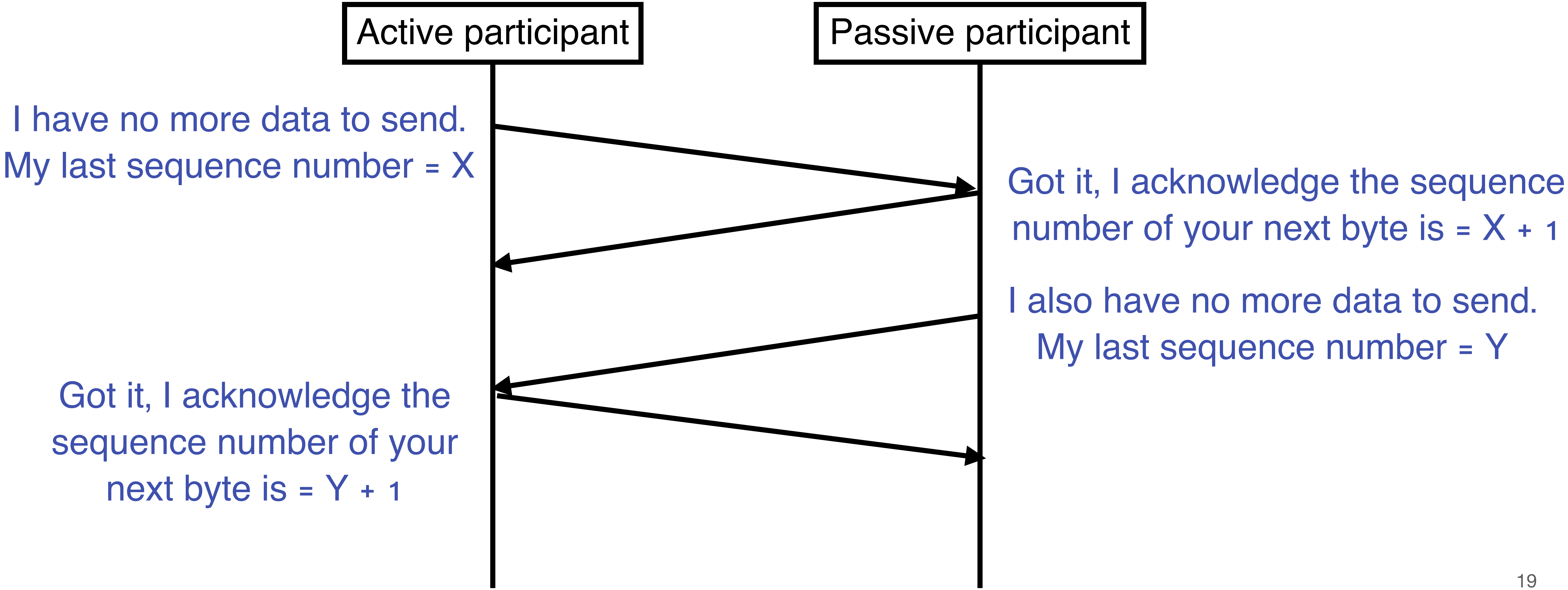
I have no more data to send.
My last sequence number = X

Got it, I acknowledge the sequence
number of your next byte is = X + 1

I also have no more data to send.
My last sequence number = Y

Case 1: One-side Closes First

4-way handshake



Active participant

Passive participant

I have no more data to send.
My last sequence number = X

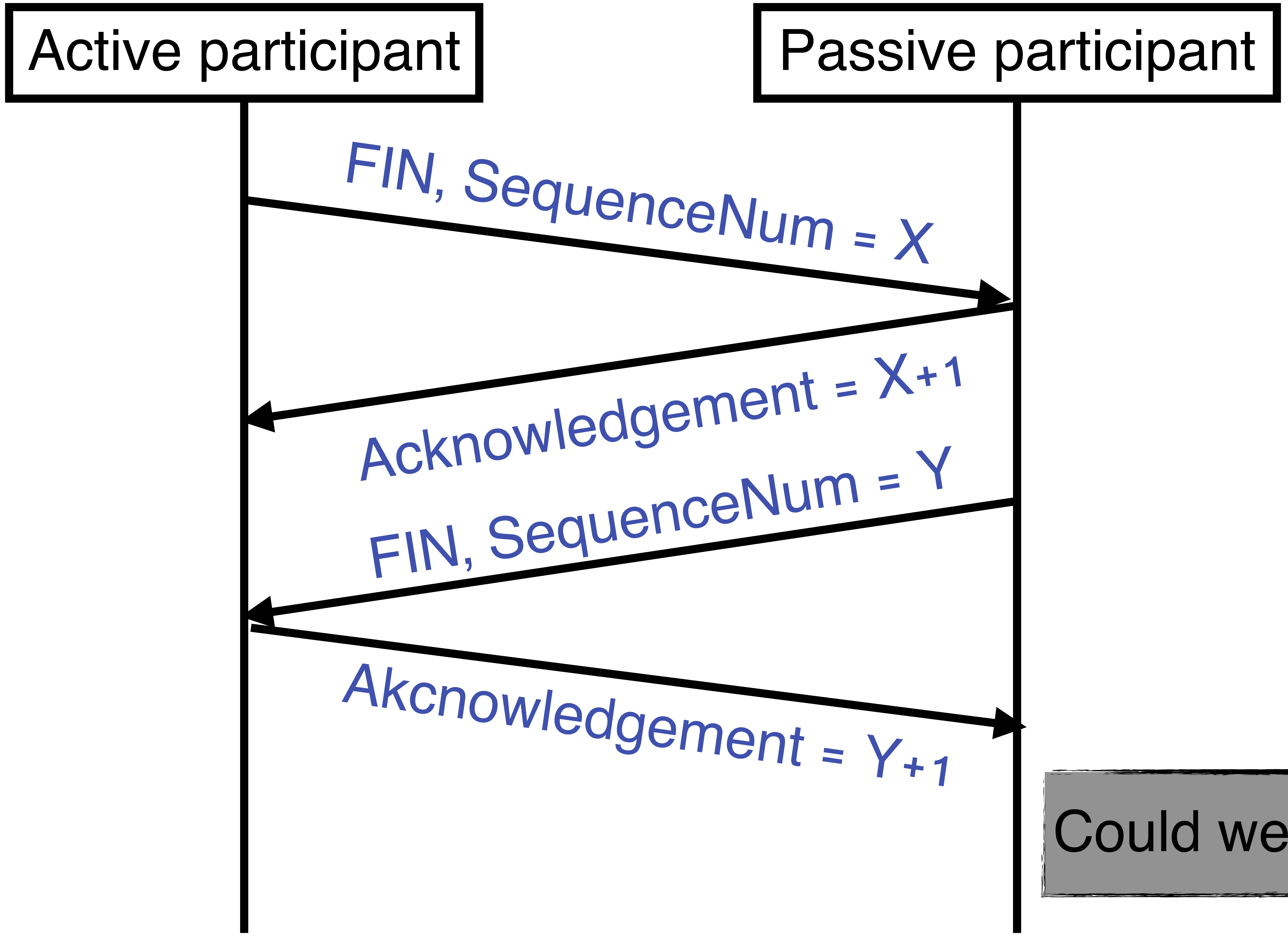
Got it, I acknowledge the sequence
number of your next byte is = X + 1

I also have no more data to send.
My last sequence number = Y

Got it, I acknowledge the
sequence number of your
next byte is = Y + 1

Case 1: One-side Closes First

4-way handshake



Could we do a 3-way handshake?

Case 1: State Machine Transition

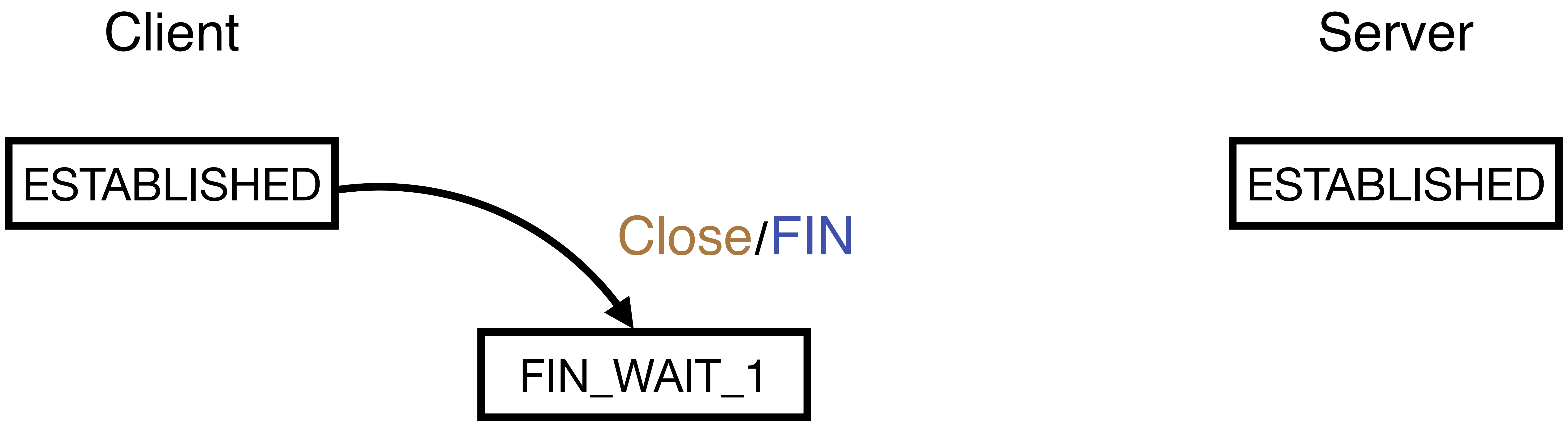
Client

ESTABLISHED

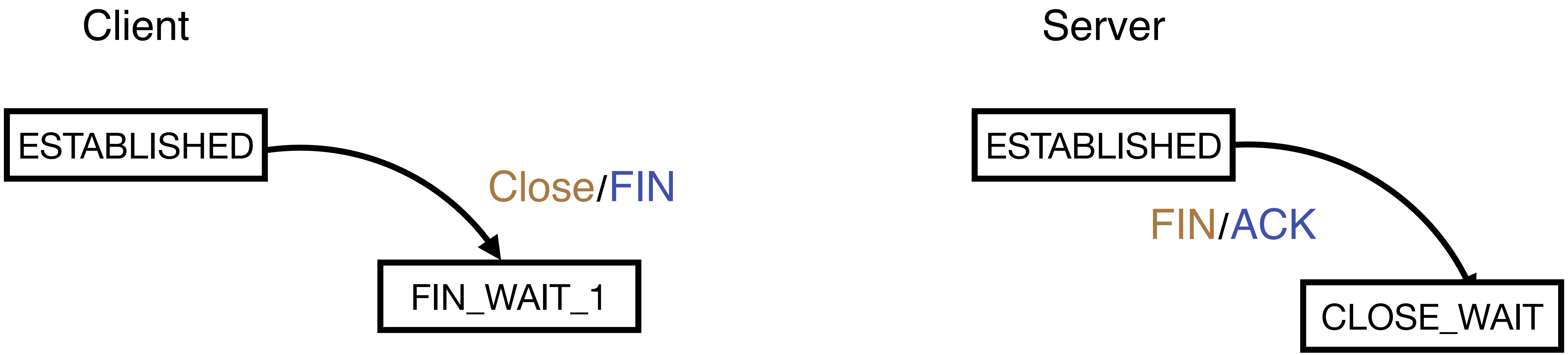
Server

ESTABLISHED

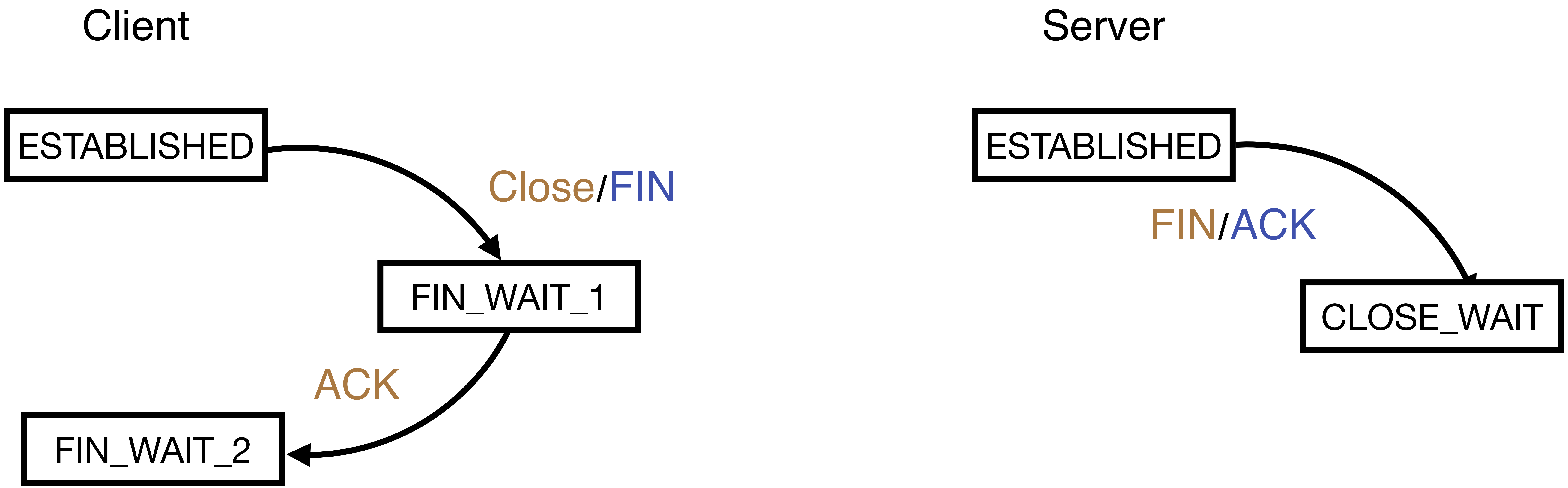
Case 1: State Machine Transition (Step 1)



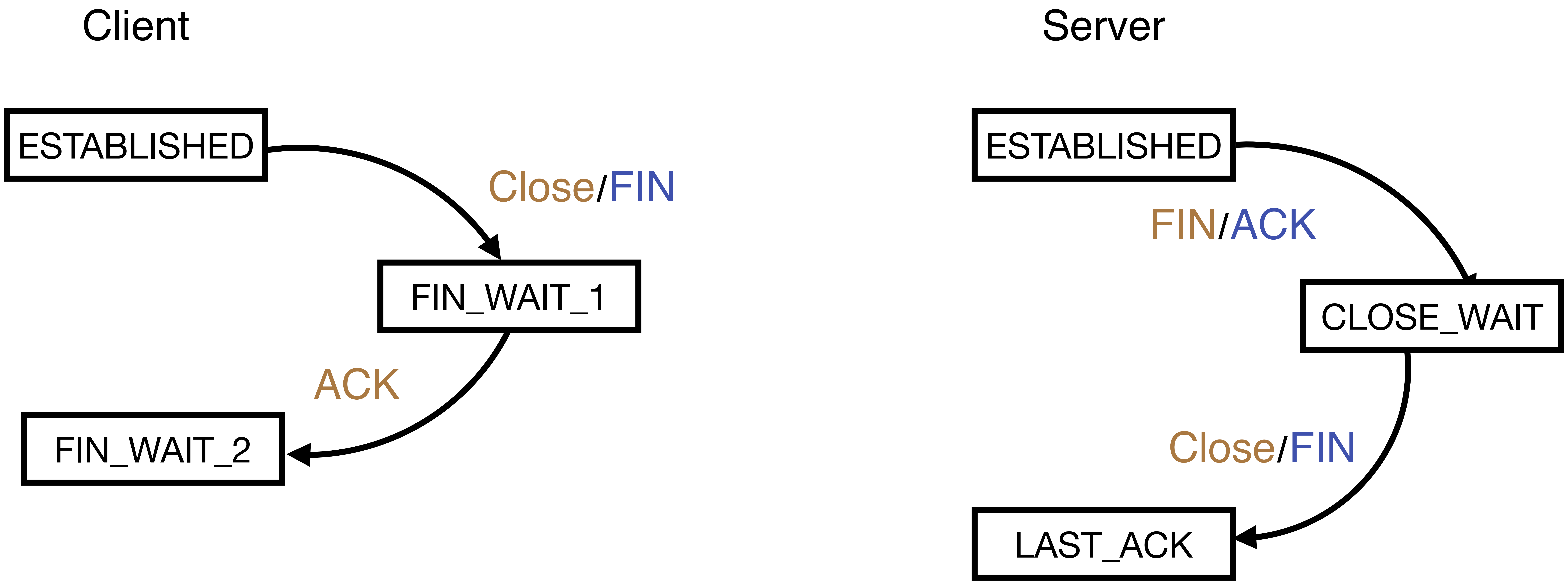
Case 1: State Machine Transition (Step 1)



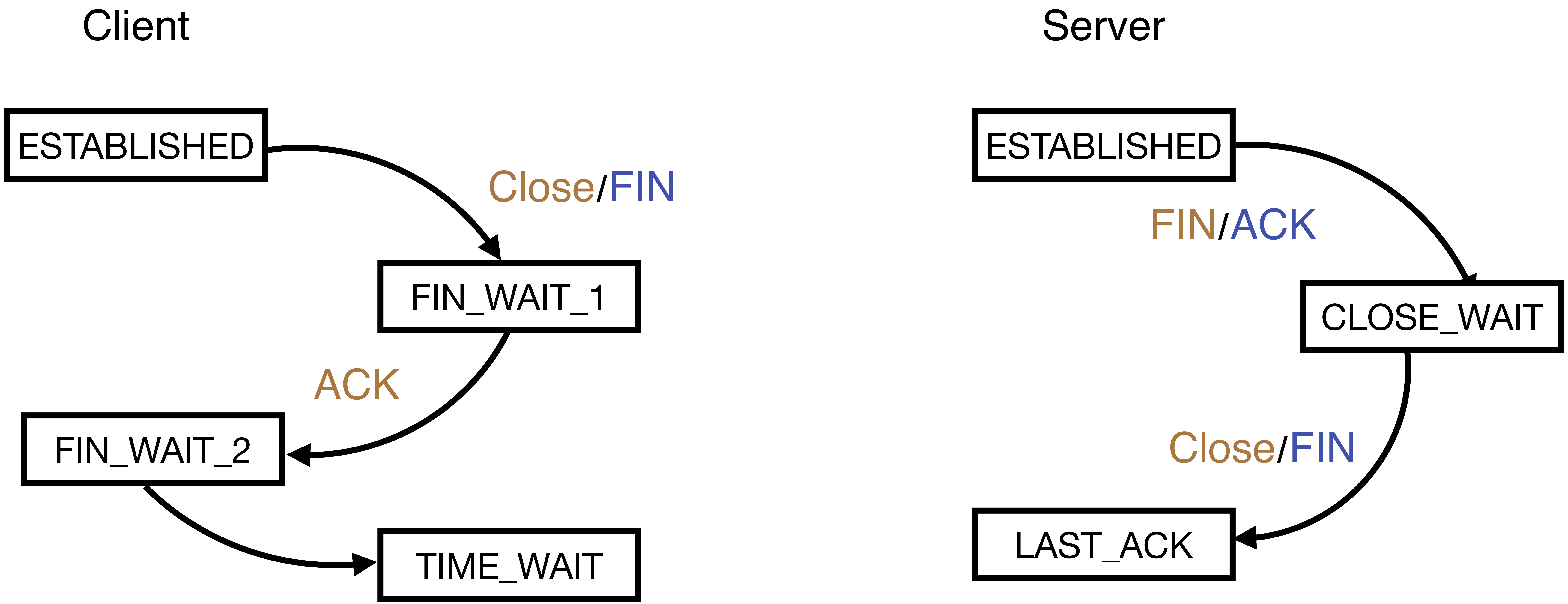
Case 1: State Machine Transition (Step 2)



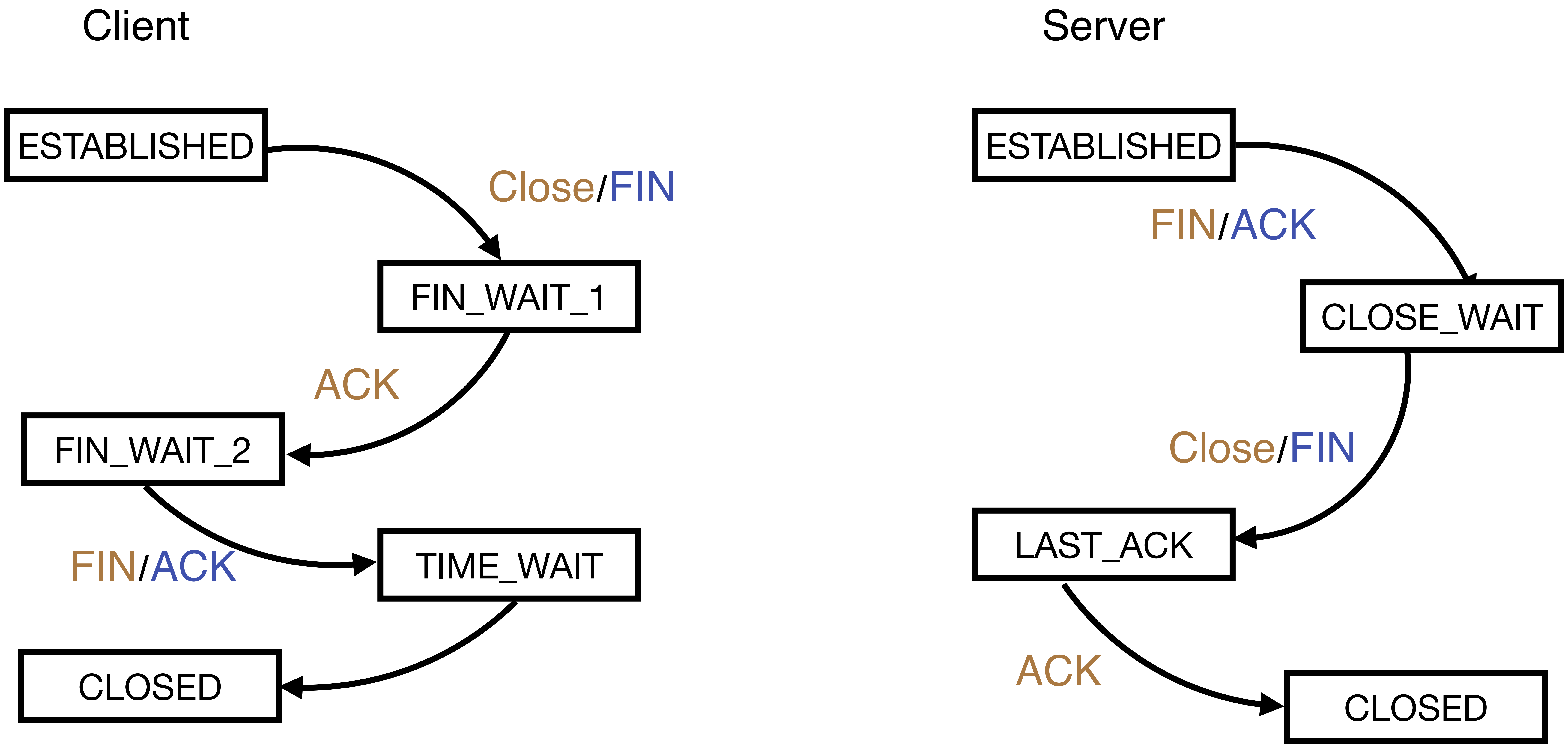
Case 1: State Machine Transition (Step 3)



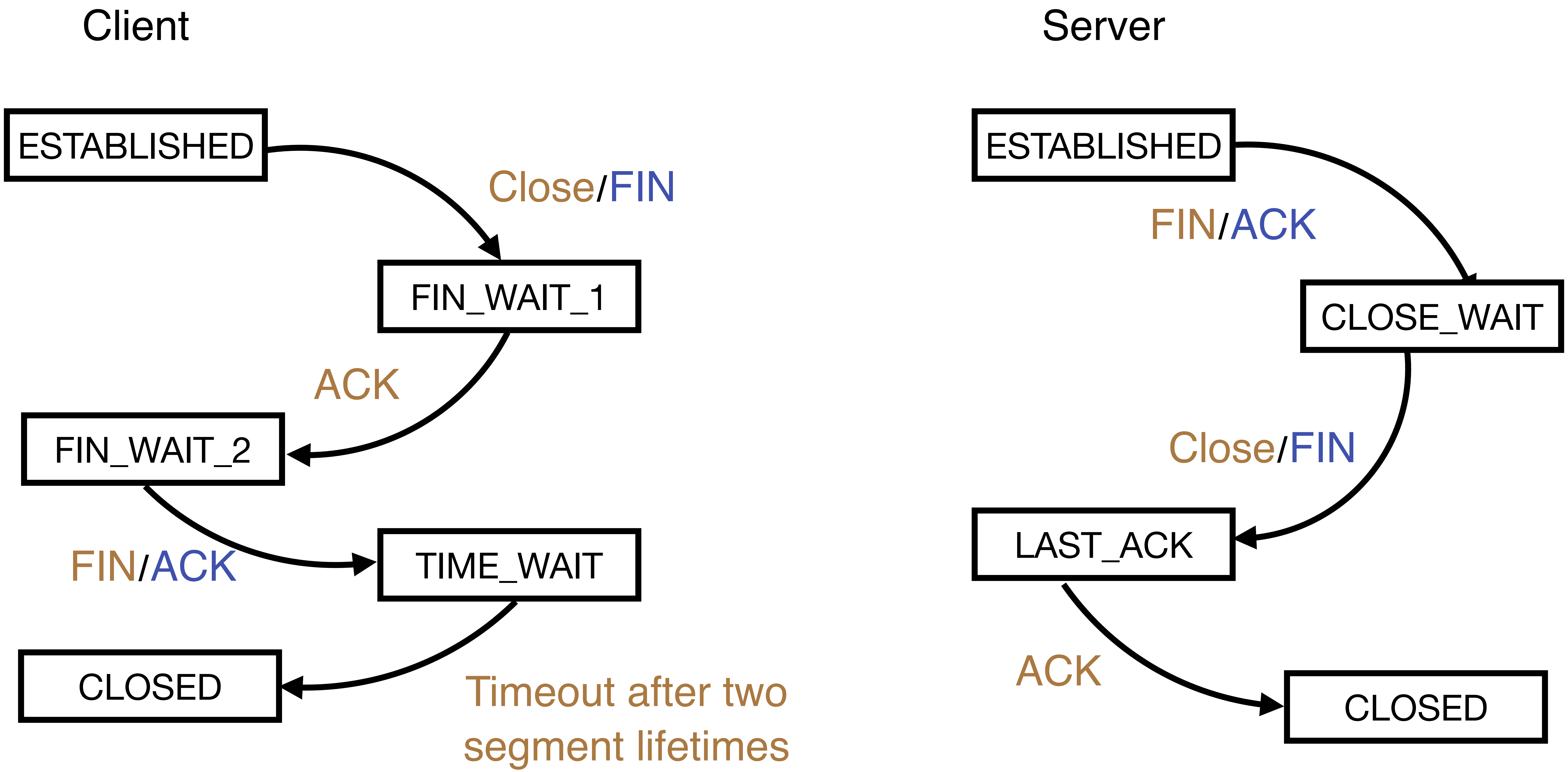
Case 1: State Machine Transition (Step 3)



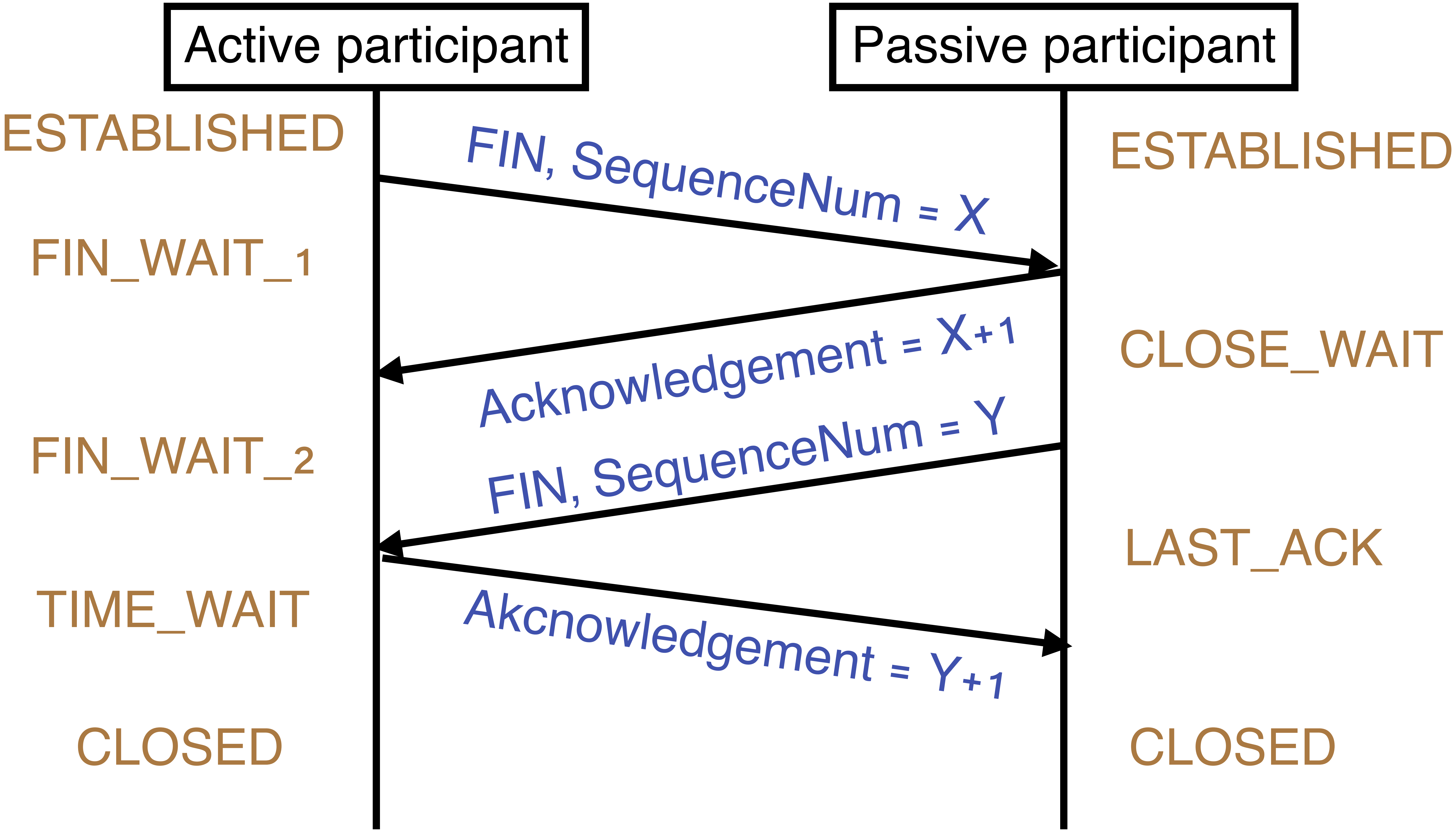
Case 1: State Machine Transition (Step 4)



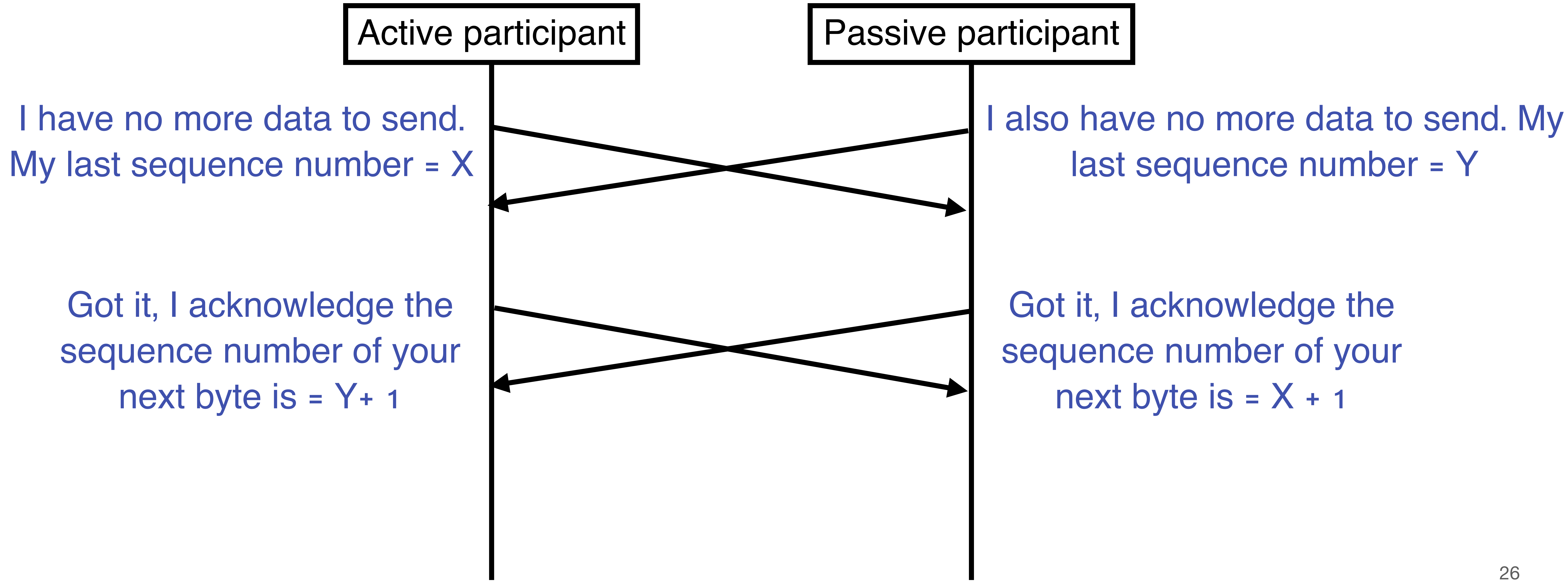
Case 1: State Machine Transition (Step 4)



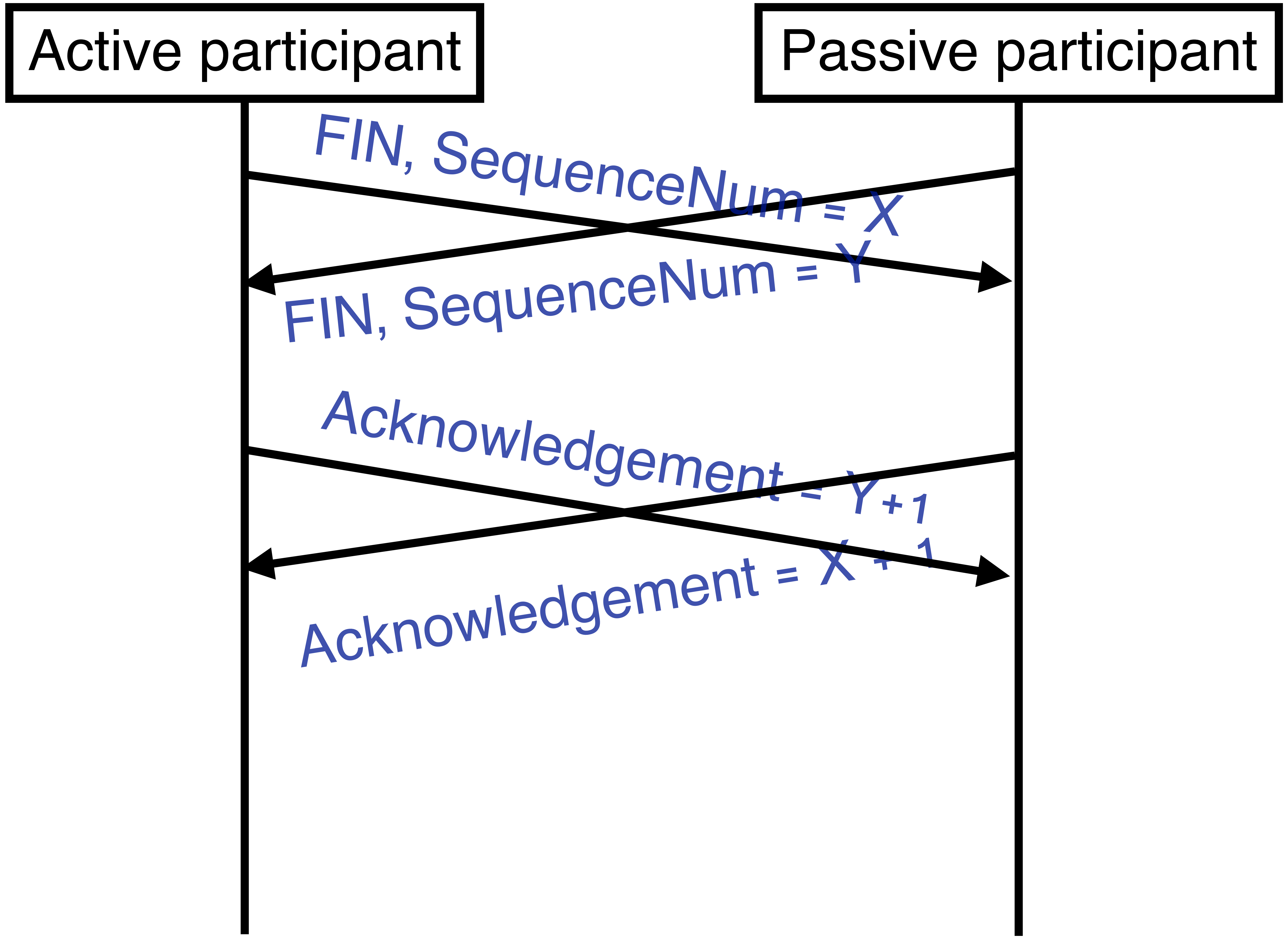
TCP Connection Termination (Case1) Summary



Case 2: Both Sides Close Simultaneously


Active participant
Passive participant
I have no more data to send. My last sequence number = X
I also have no more data to send. My last sequence number = Y
Got it, I acknowledge the sequence number of your next byte is = Y+ 1
Got it, I acknowledge the sequence number of your next byte is = X + 1

Case 2: Both Sides Close Simultaneously



Case 2: State Machine Transition (Step 1)

Client

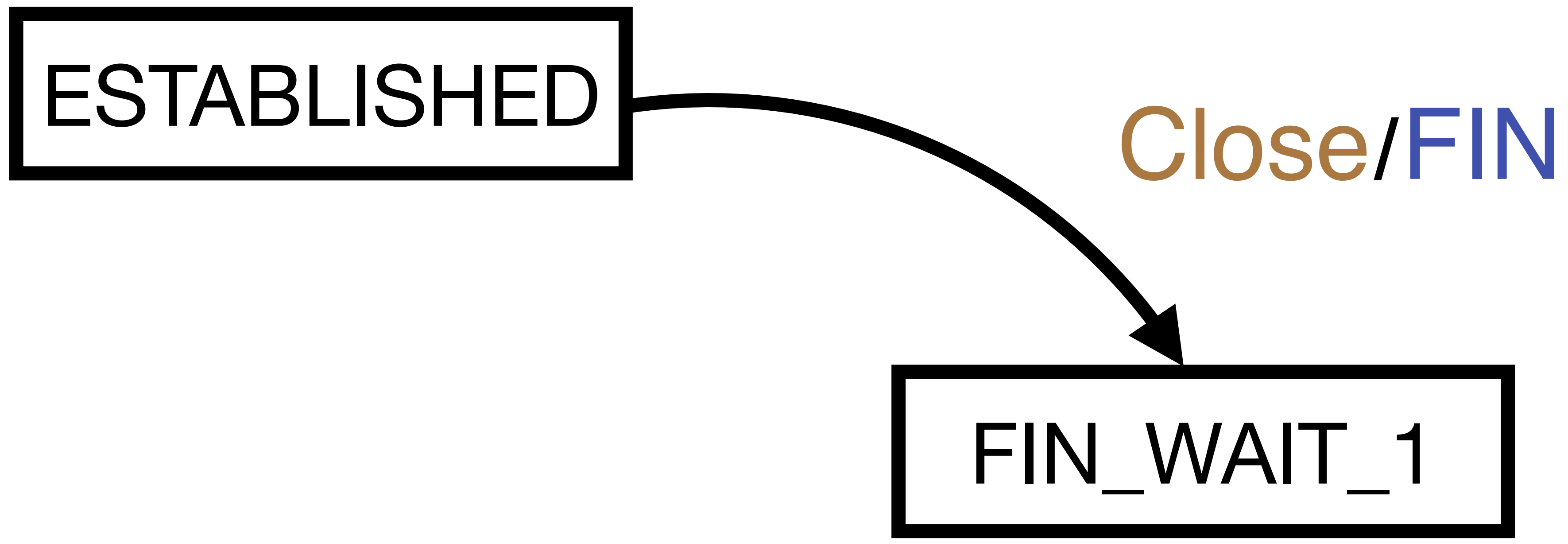
ESTABLISHED

Server

ESTABLISHED

Case 2: State Machine Transition (Step 1)

Client

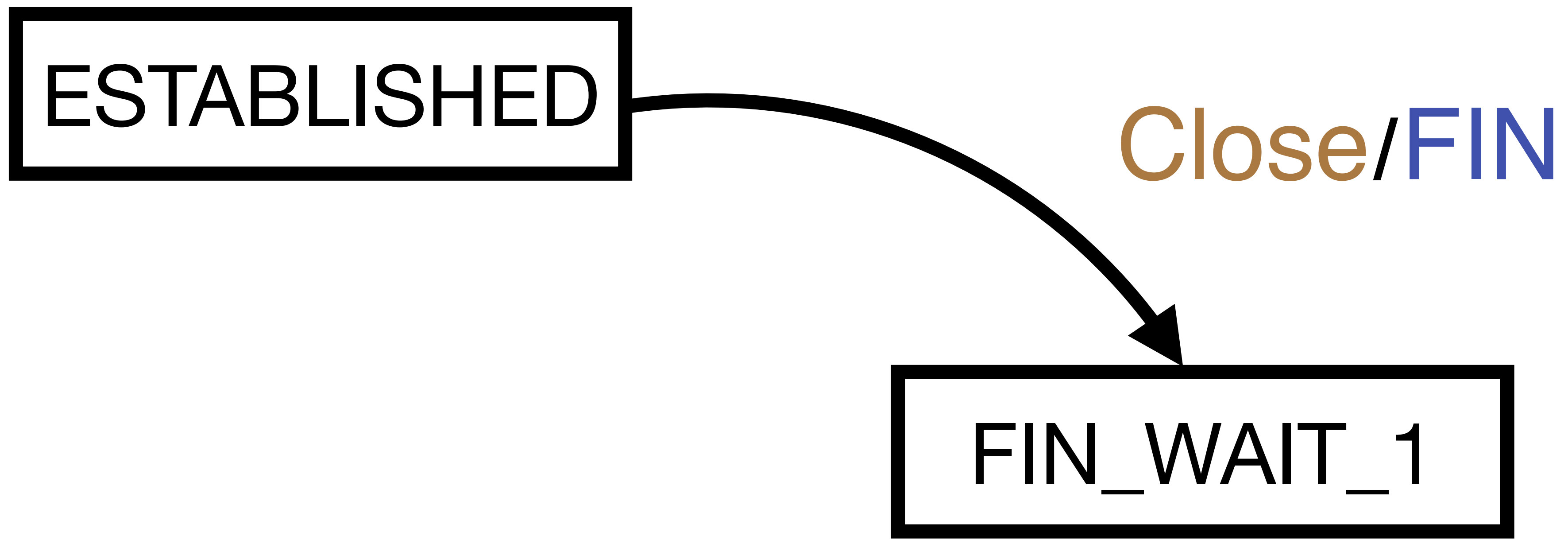


ESTABLISHED

Close/FIN

FIN_WAIT_1

Server

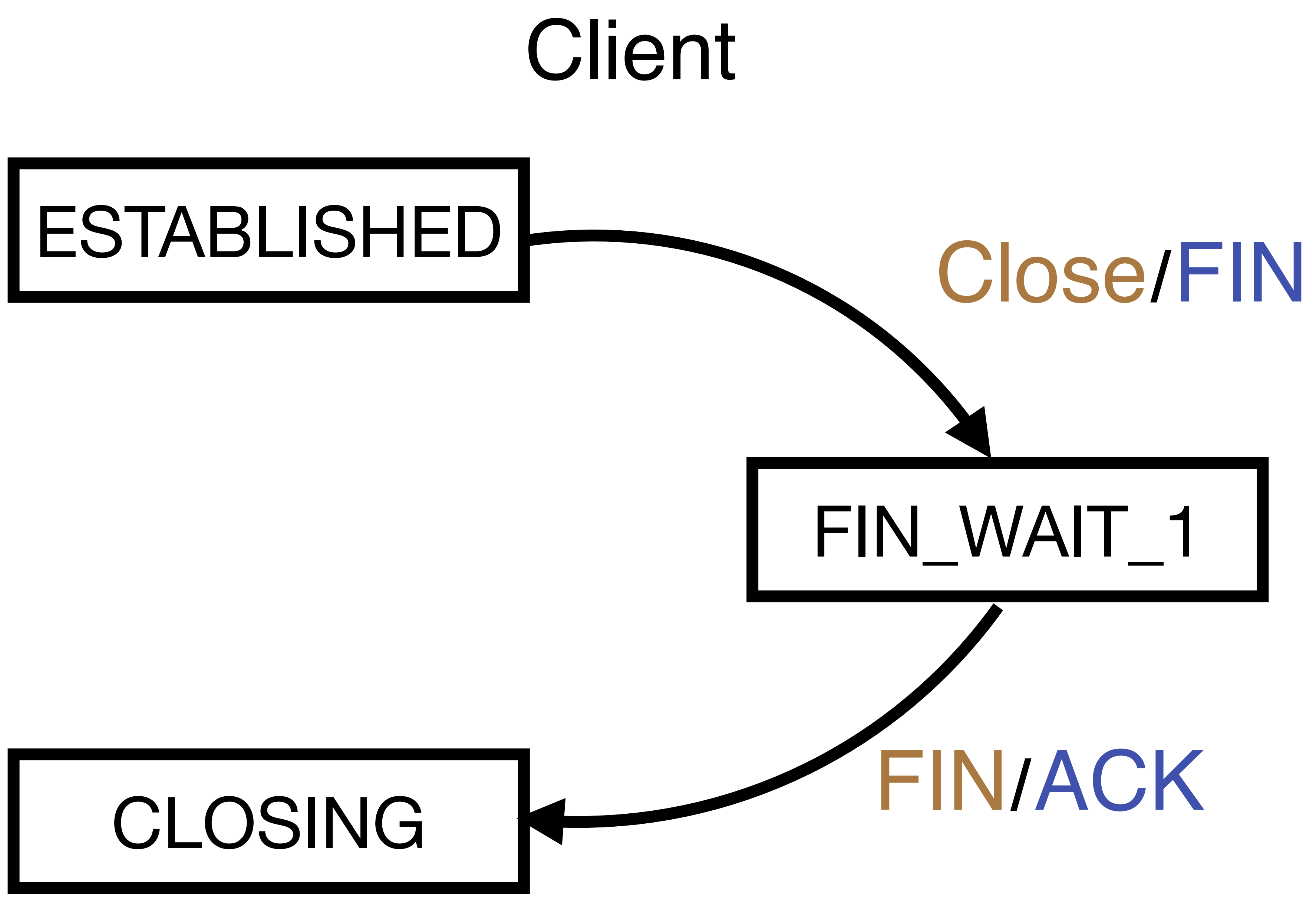


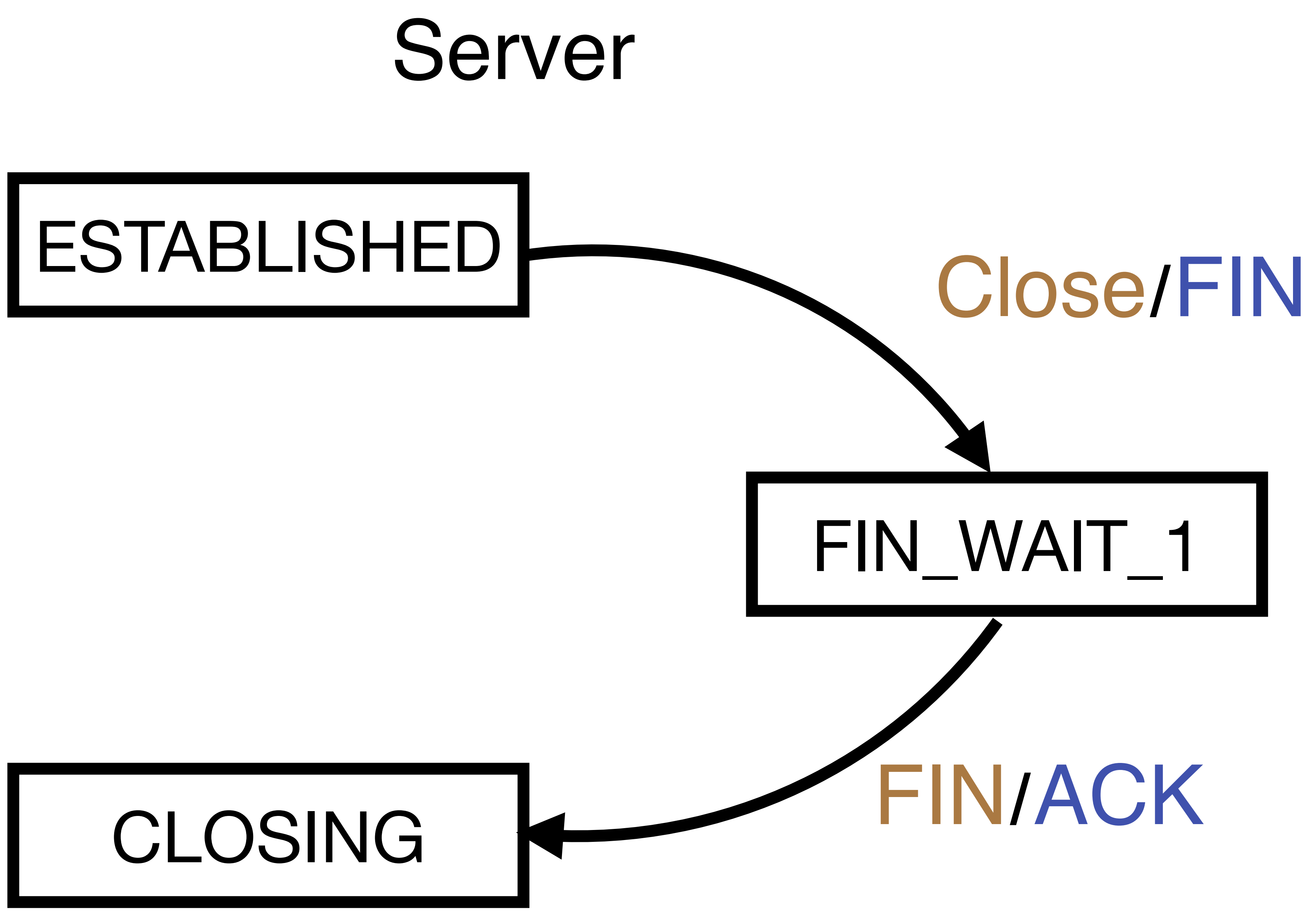
ESTABLISHED

Close/FIN

FIN_WAIT_1

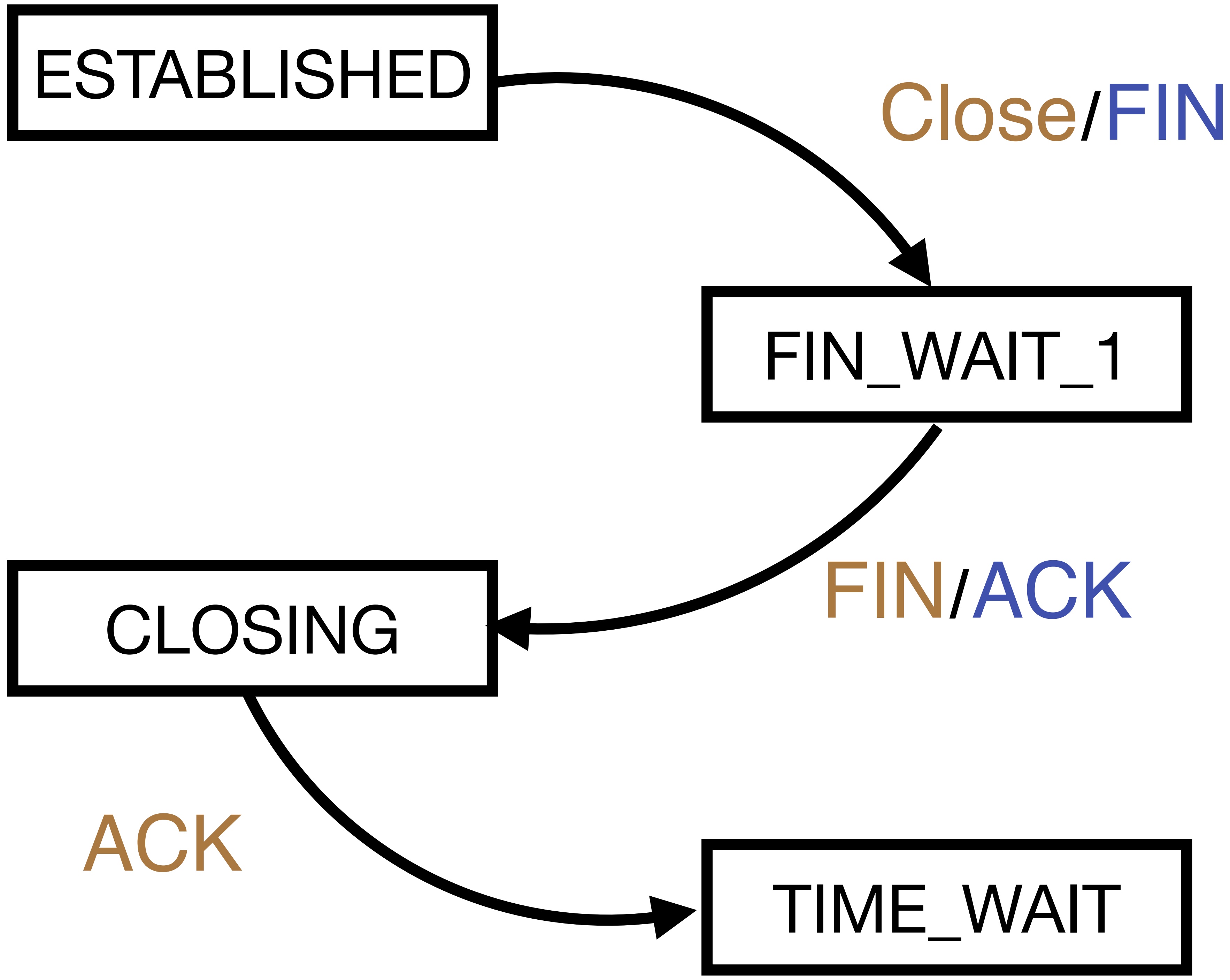
Case 2: State Machine Transition (Step 2)





Case 2: State Machine Transition (Step 3)

Client



ESTABLISHED

Close/FIN

FIN_WAIT_1

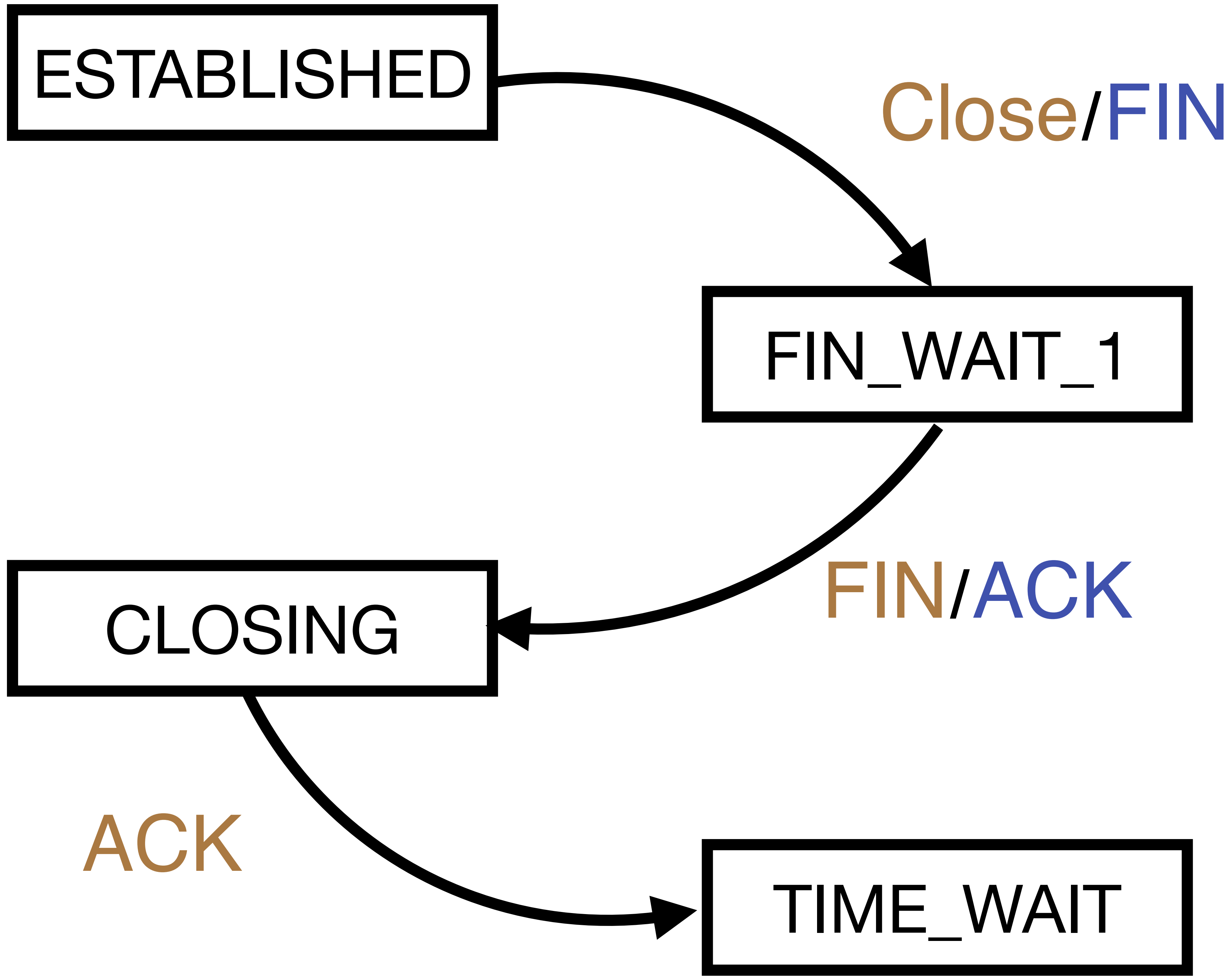
FIN/ACK

CLOSING

ACK

TIME_WAIT

Server



ESTABLISHED

Close/FIN

FIN_WAIT_1

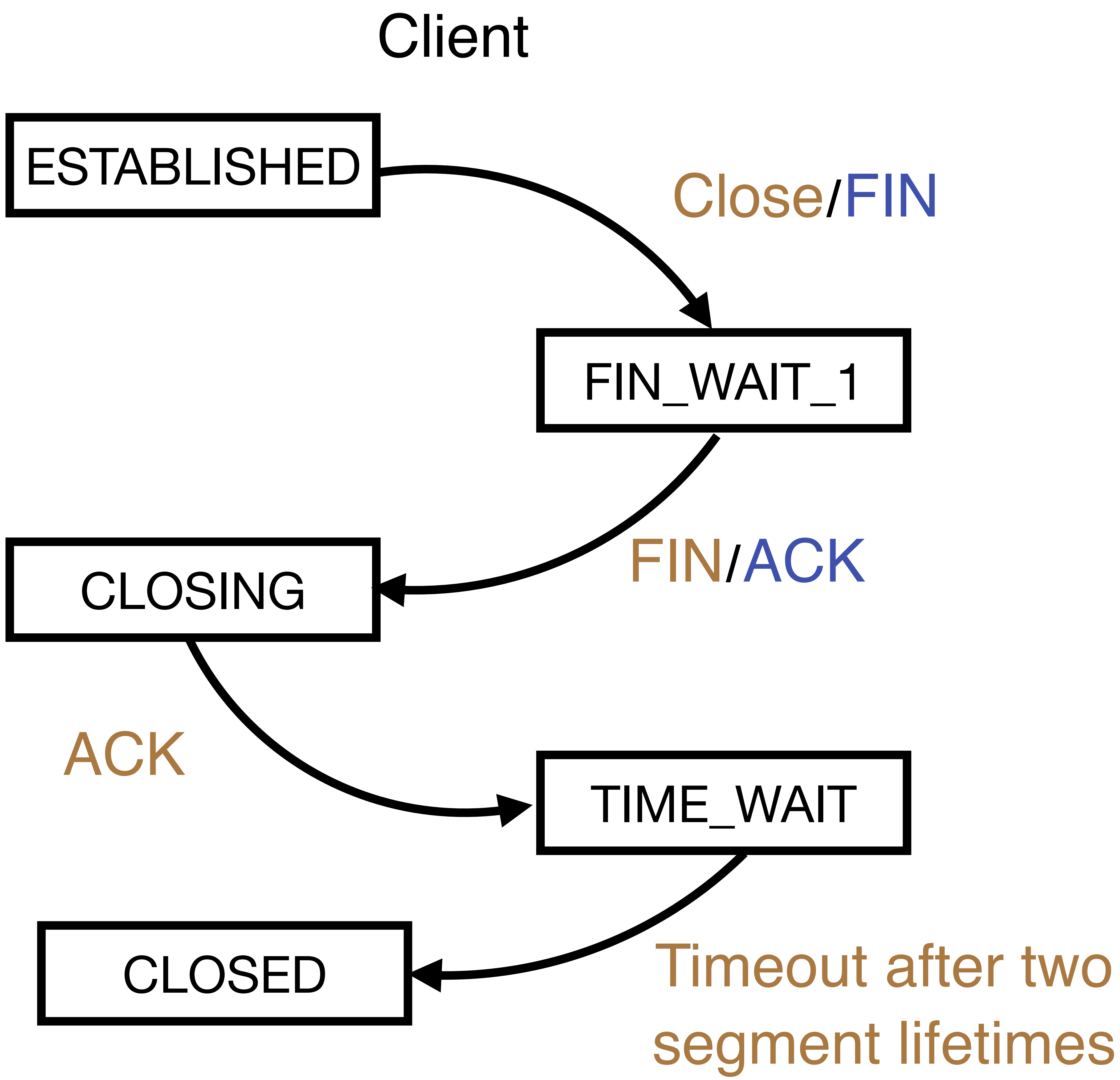
FIN/ACK

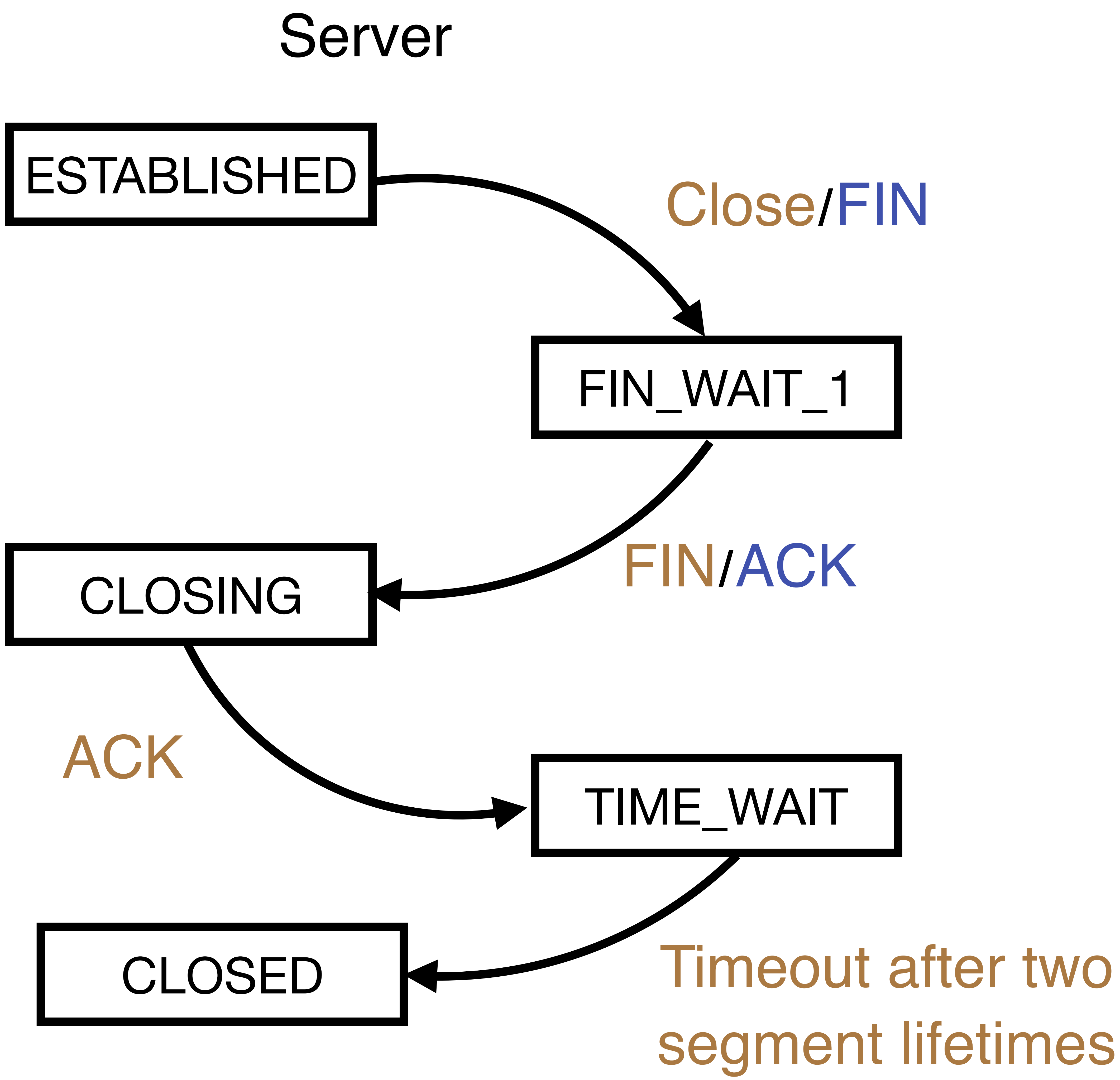
CLOSING

ACK

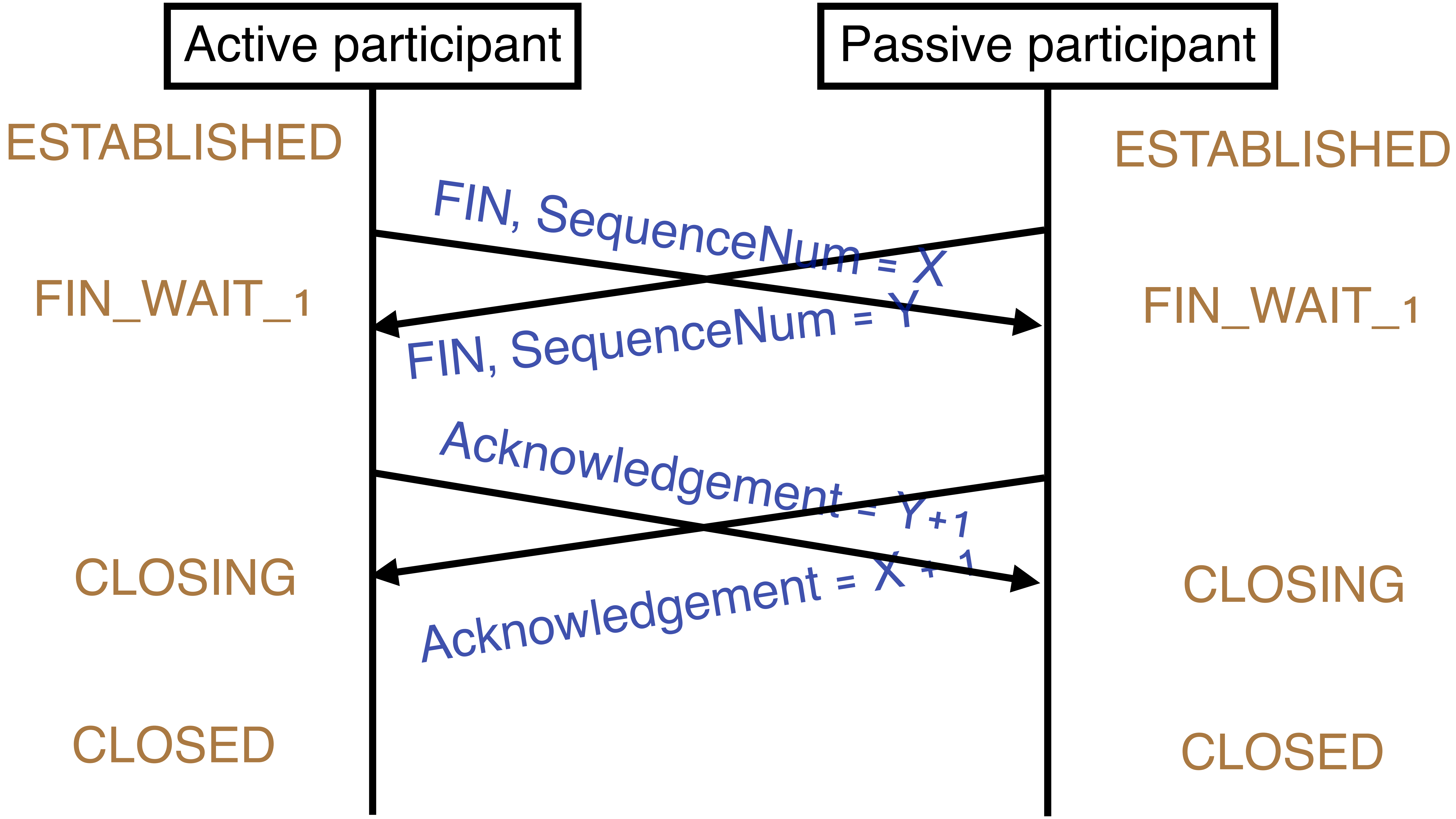
TIME_WAIT

Case 2: State Machine Transition (Step 4)

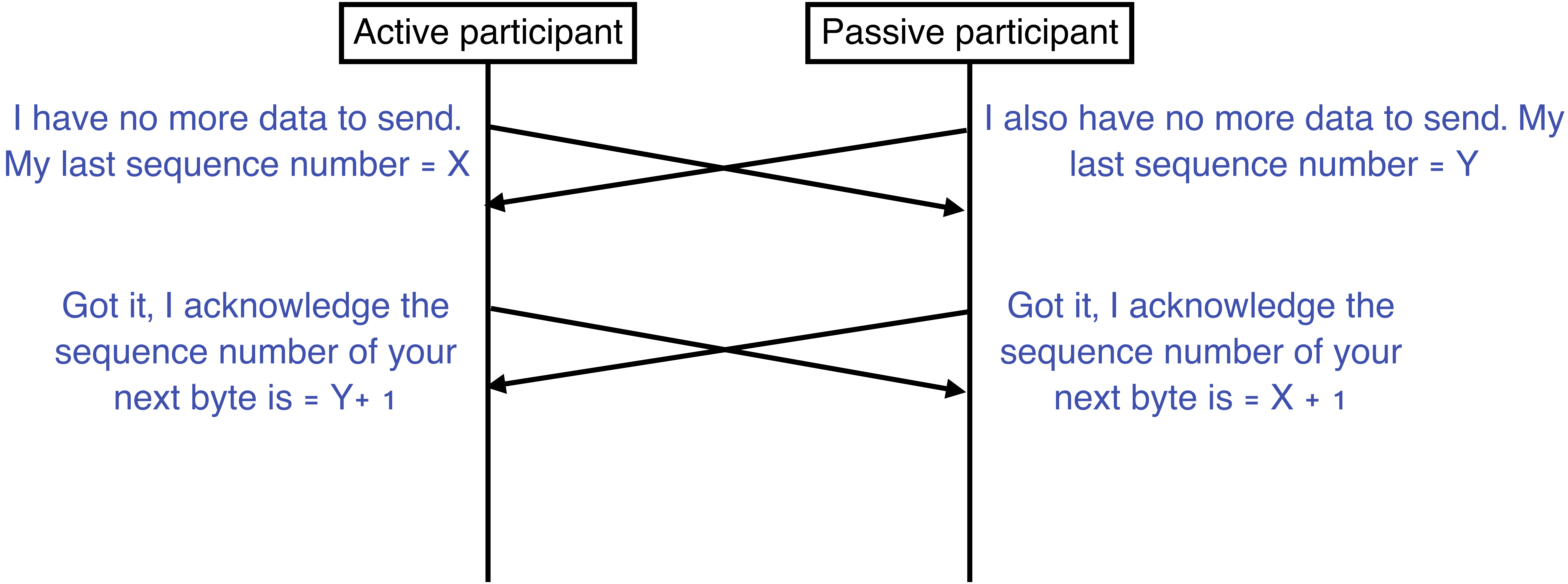




TCP Connection Termination (Case 2) Summary



Case 3: Both Sides Close Simultaneously, but



Active participant

Passive participant

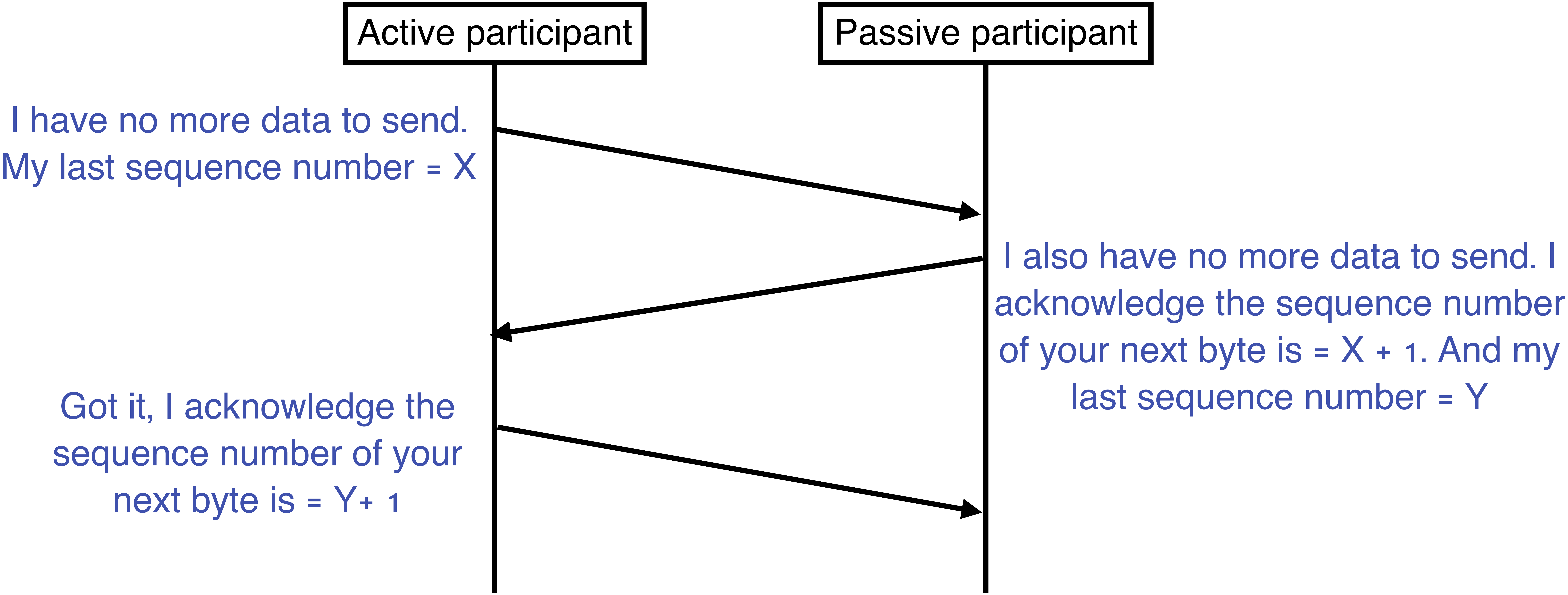
I have no more data to send. My last sequence number = X

I also have no more data to send. My last sequence number = Y

Got it, I acknowledge the sequence number of your next byte is = Y+ 1

Got it, I acknowledge the sequence number of your next byte is = X + 1

Case 3: Both Sides Close Simultaneously, but



Active participant

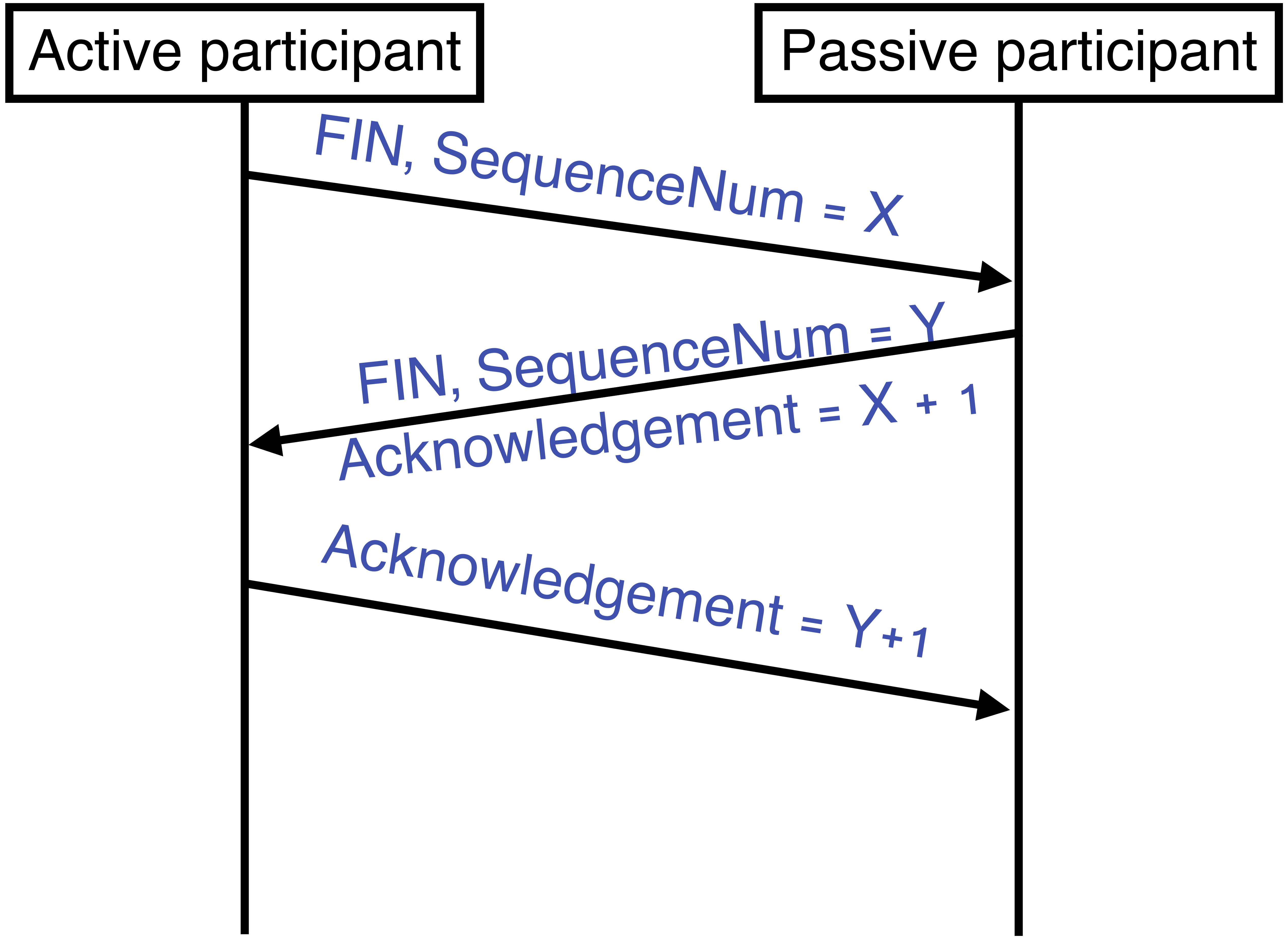
Passive participant

I have no more data to send.
My last sequence number = X

I also have no more data to send. I acknowledge the sequence number of your next byte is = X + 1. And my last sequence number = Y

Got it, I acknowledge the sequence number of your next byte is = Y+ 1

Case 3: Both Sides Close Simultaneously, but



Case 3: State Machine Transition

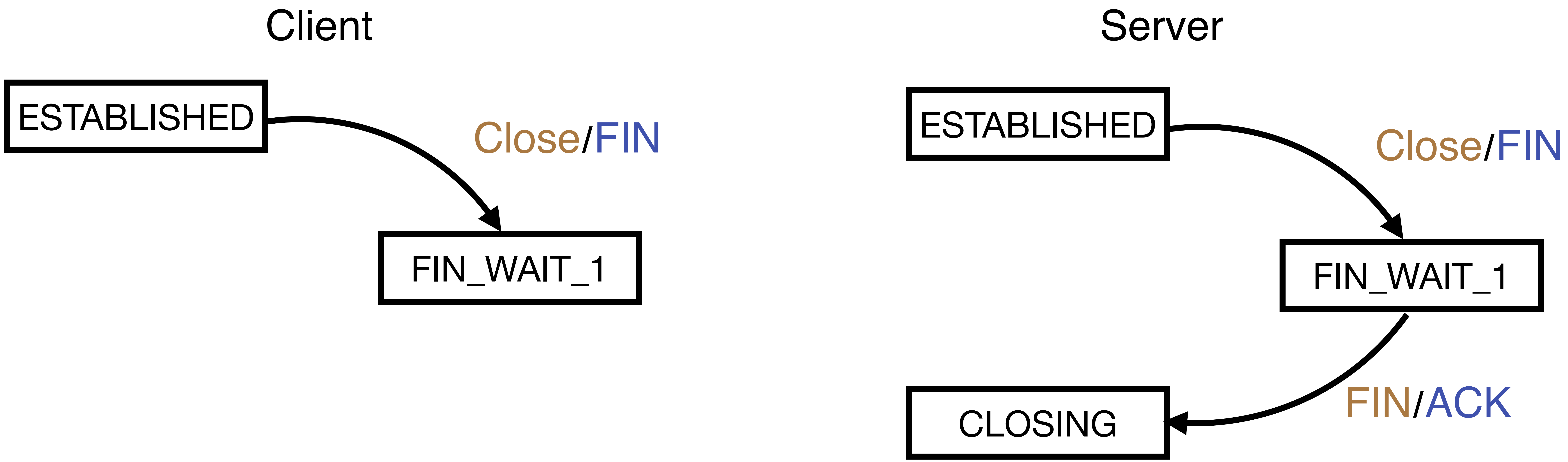
Client

ESTABLISHED

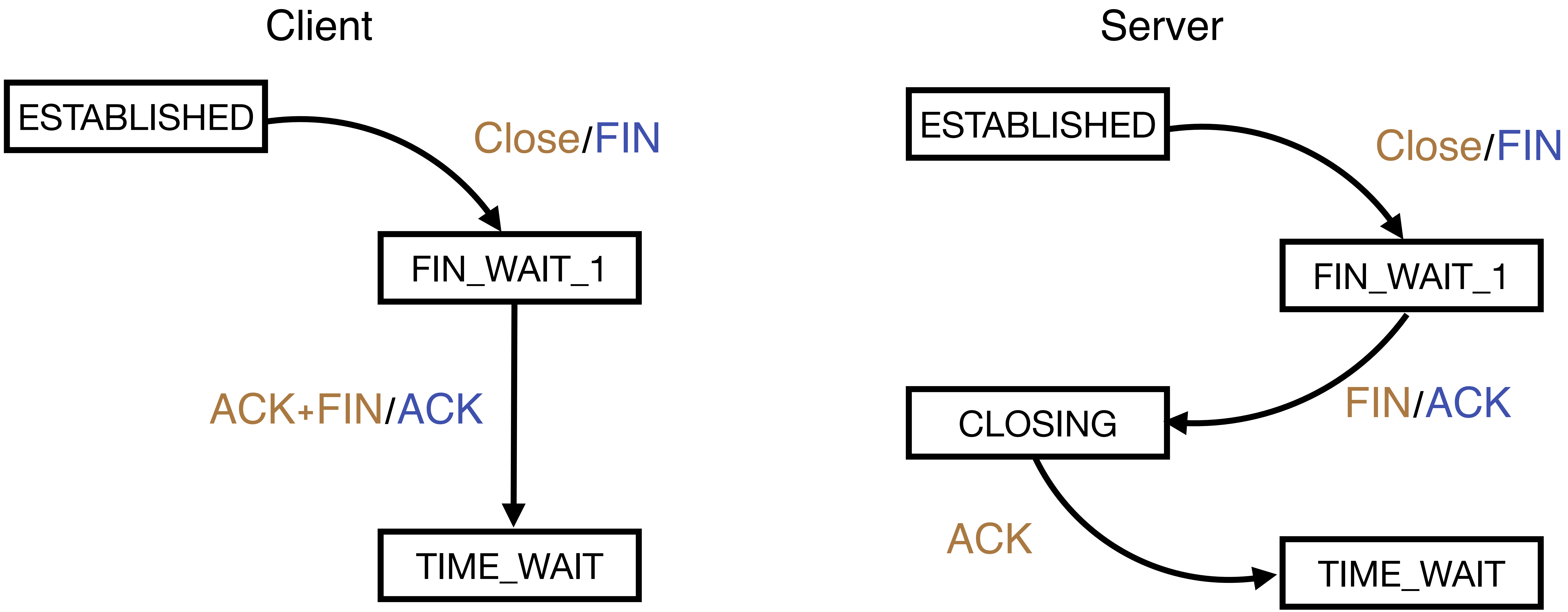
Server

ESTABLISHED

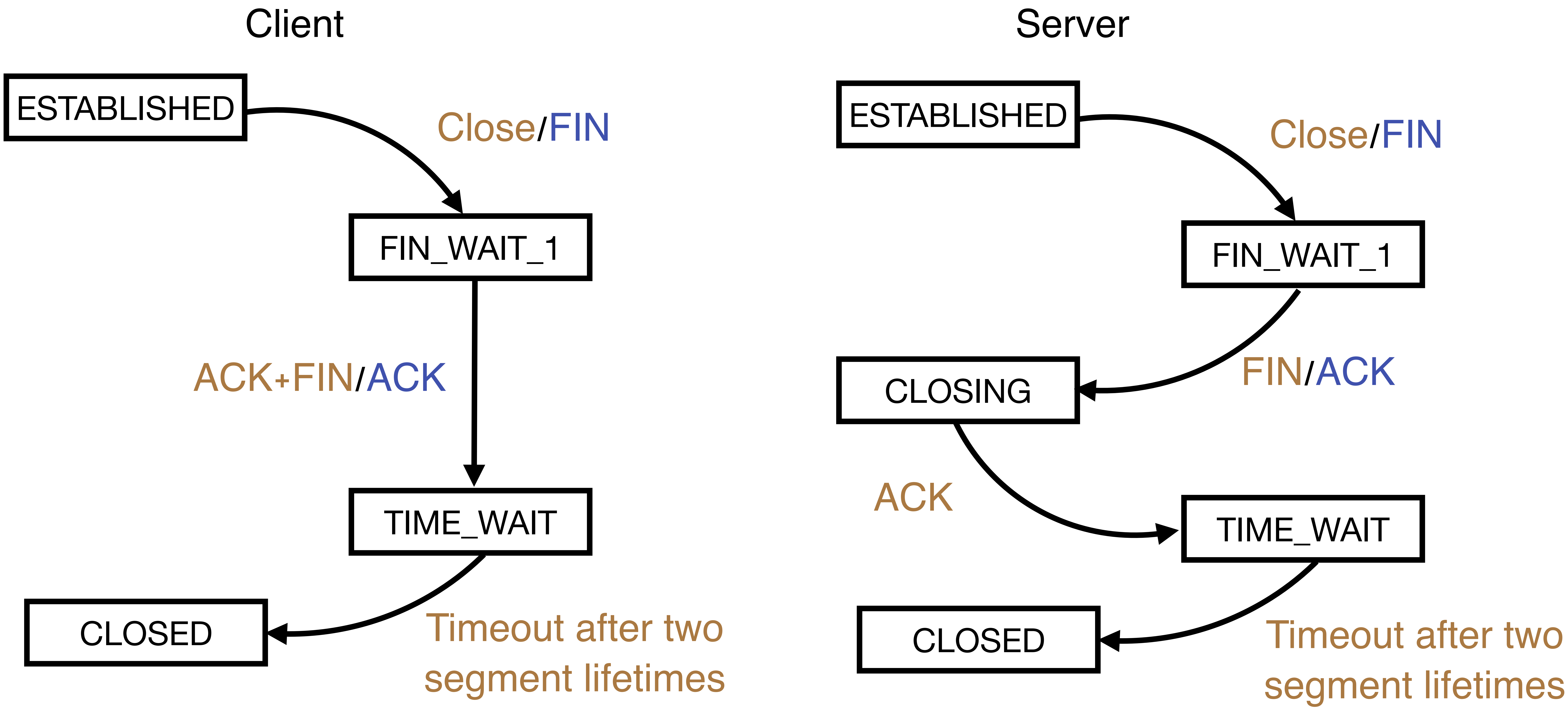
Case 3: State Machine Transition (Step 1)



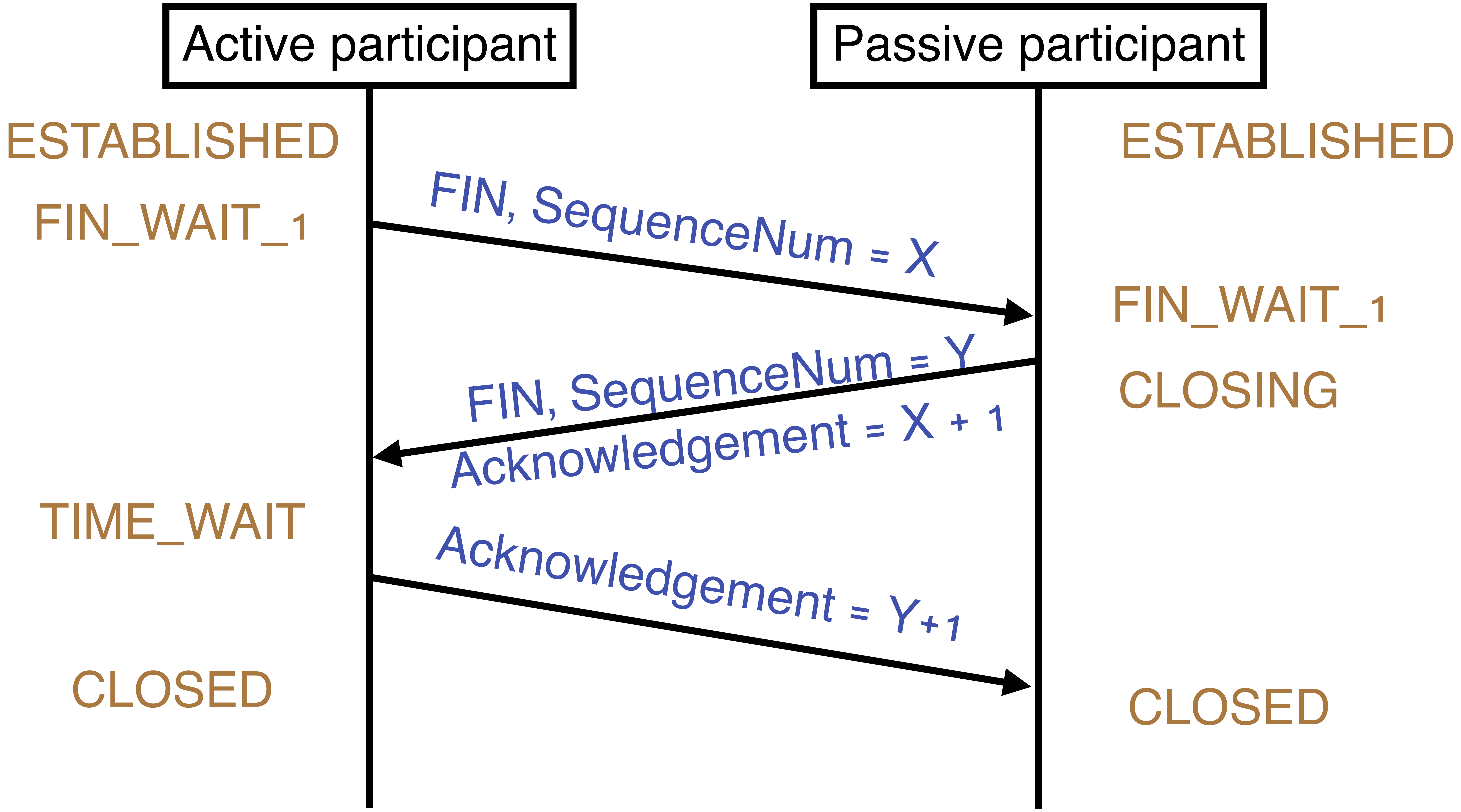
Case 3: State Machine Transition (Step 2)



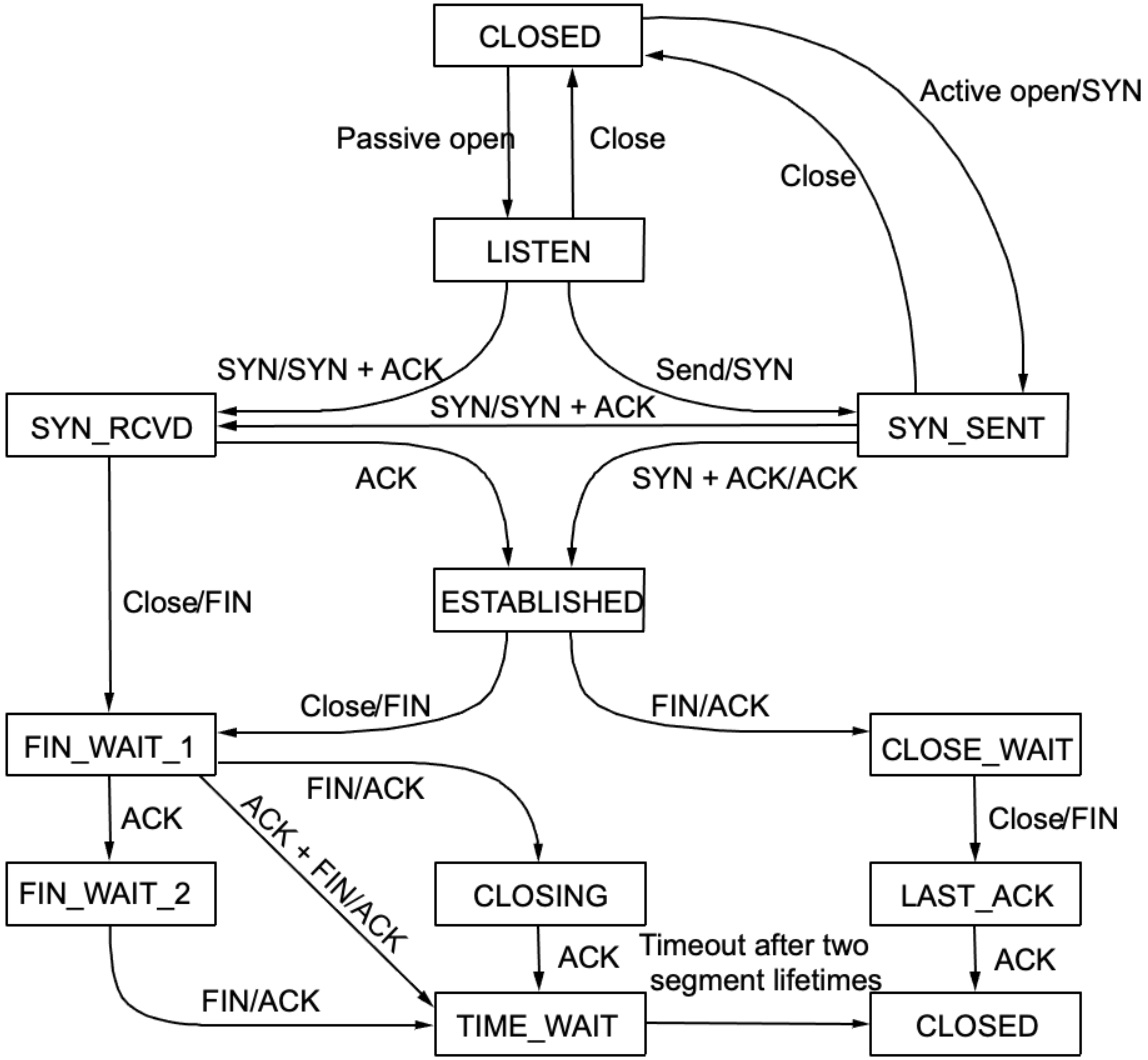
Case 3: State Machine Transition (Step 3)



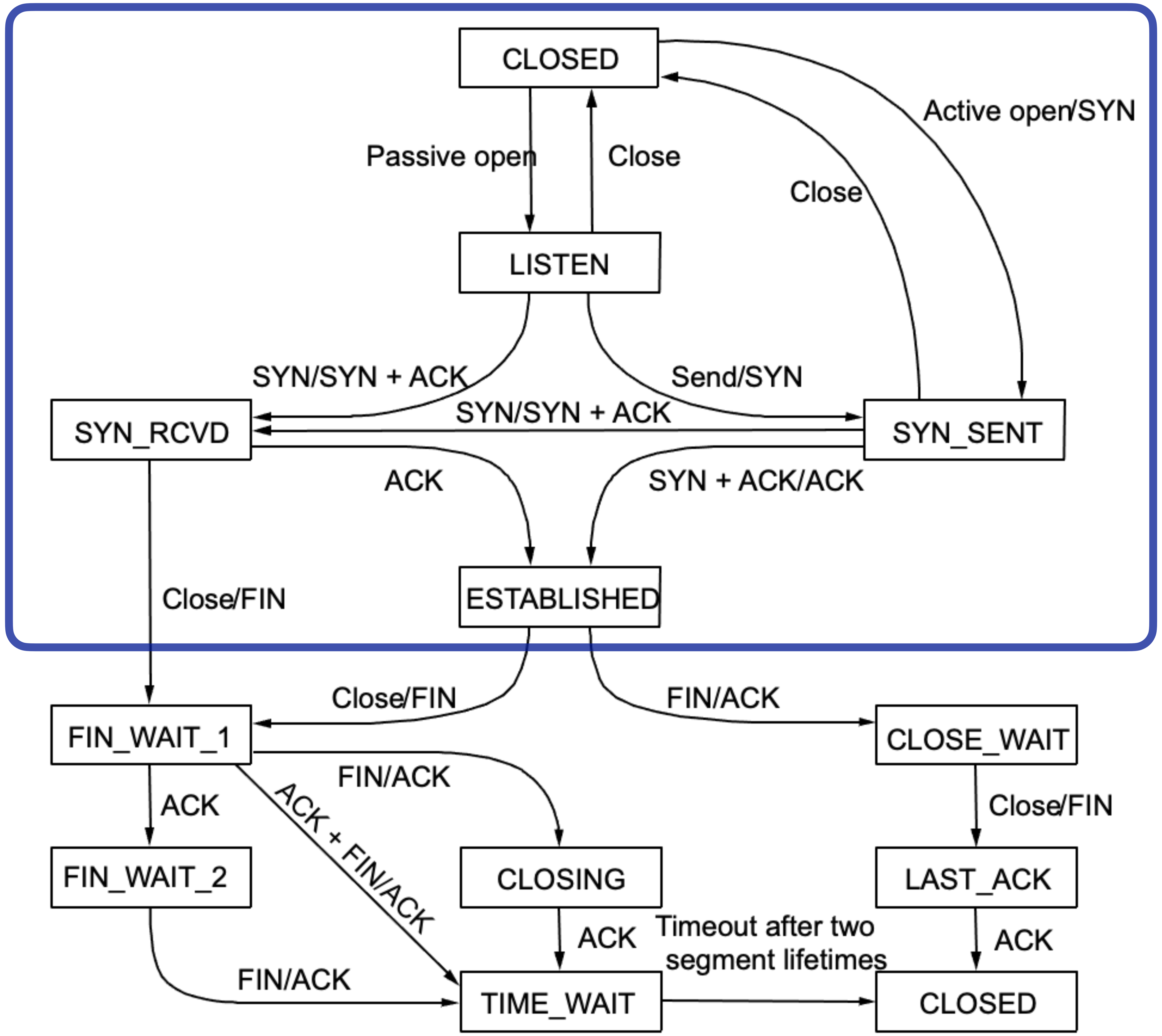
TCP Connection Termination (Case 3) Summary



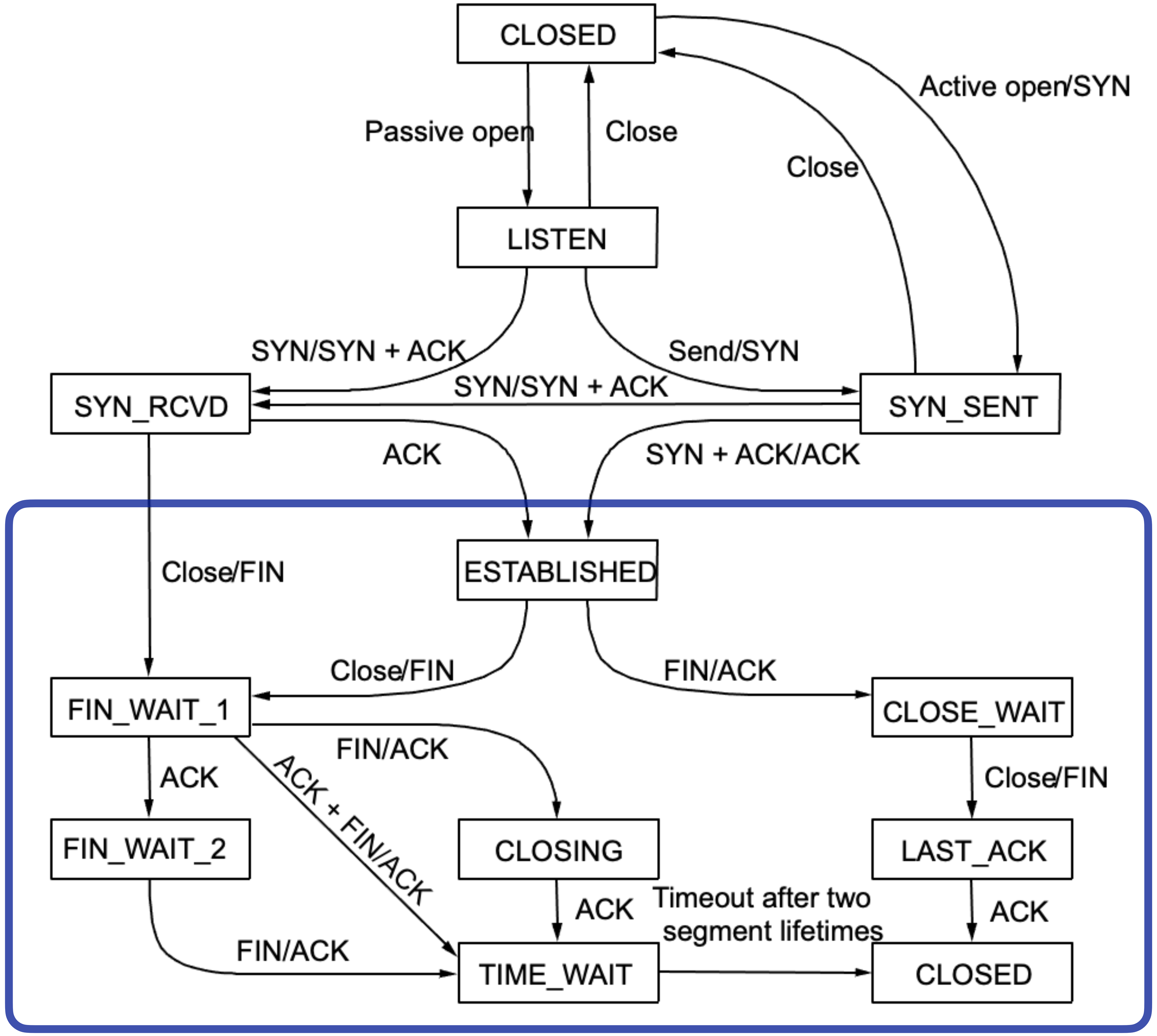
TCP State Transition Diagram Overall



TCP State Transition Diagram Overall



TCP State Transition Diagram Overall



TCP Connection Management Summary

Connection setup is asymmetric, where one side does a passive open the other side does an active open

Connection teardown is symmetric, where each side has to close the connection independently

Most of the states schedule a timeout, eventually causing the segment to be present if the expected response does not happen

Summary

Today

- TCP connection management

Next lecture

- TCP reliability mechanisms