# When Connection Establishment Fails: Syn Floods & Retransmission

In this lesson, we'll look at a couple of loopholes in TCP's implementation and how modern fixes took care of them.

WE'LL COVER THE FOLLOWING



## Hosts Can Refuse Connection Requests #

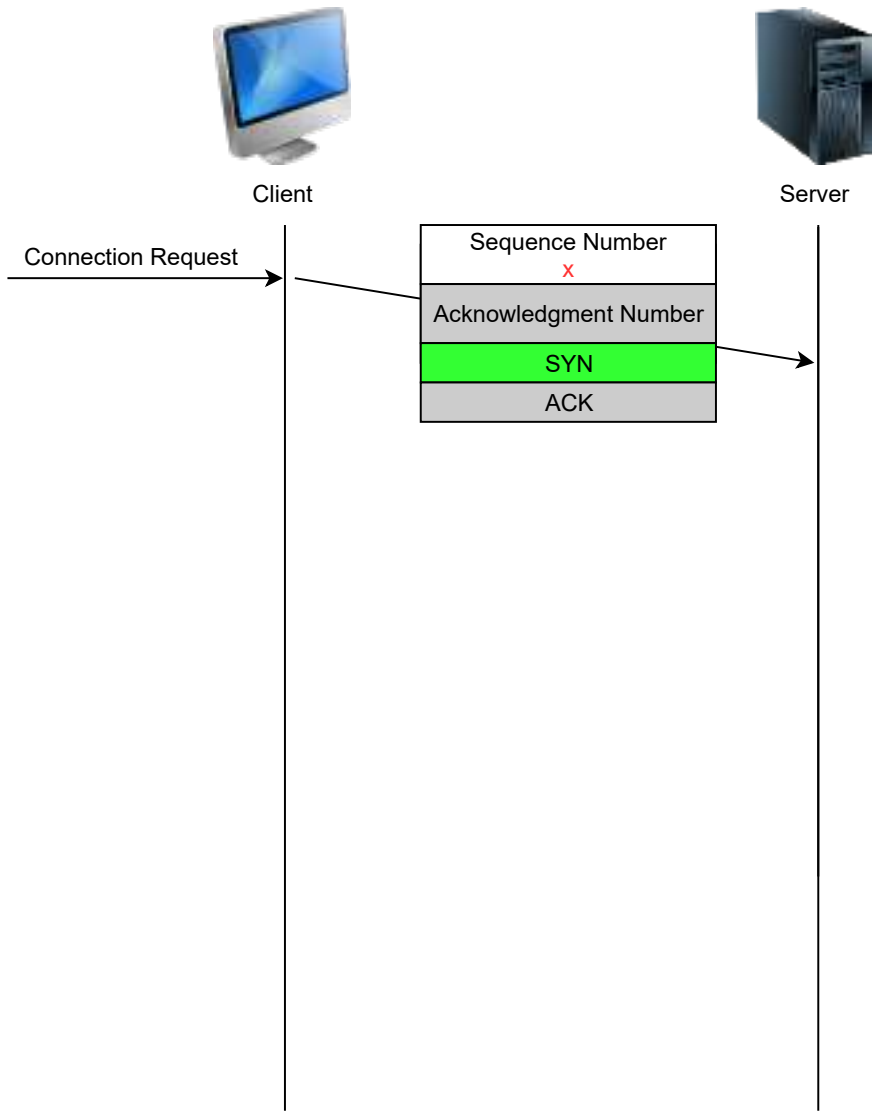
A host could refuse to open a TCP connection upon reception of a *SYN* segment. This refusal may be due to various reasons, for example:

1. There may be **no server process** that's listening on the destination port of the *SYN* segment.
2. The server could always refuse connection establishments from *a* particular client (e.g., due to **security reasons**).
3. The server may **not have enough resources** to accept a new TCP connection at that time.

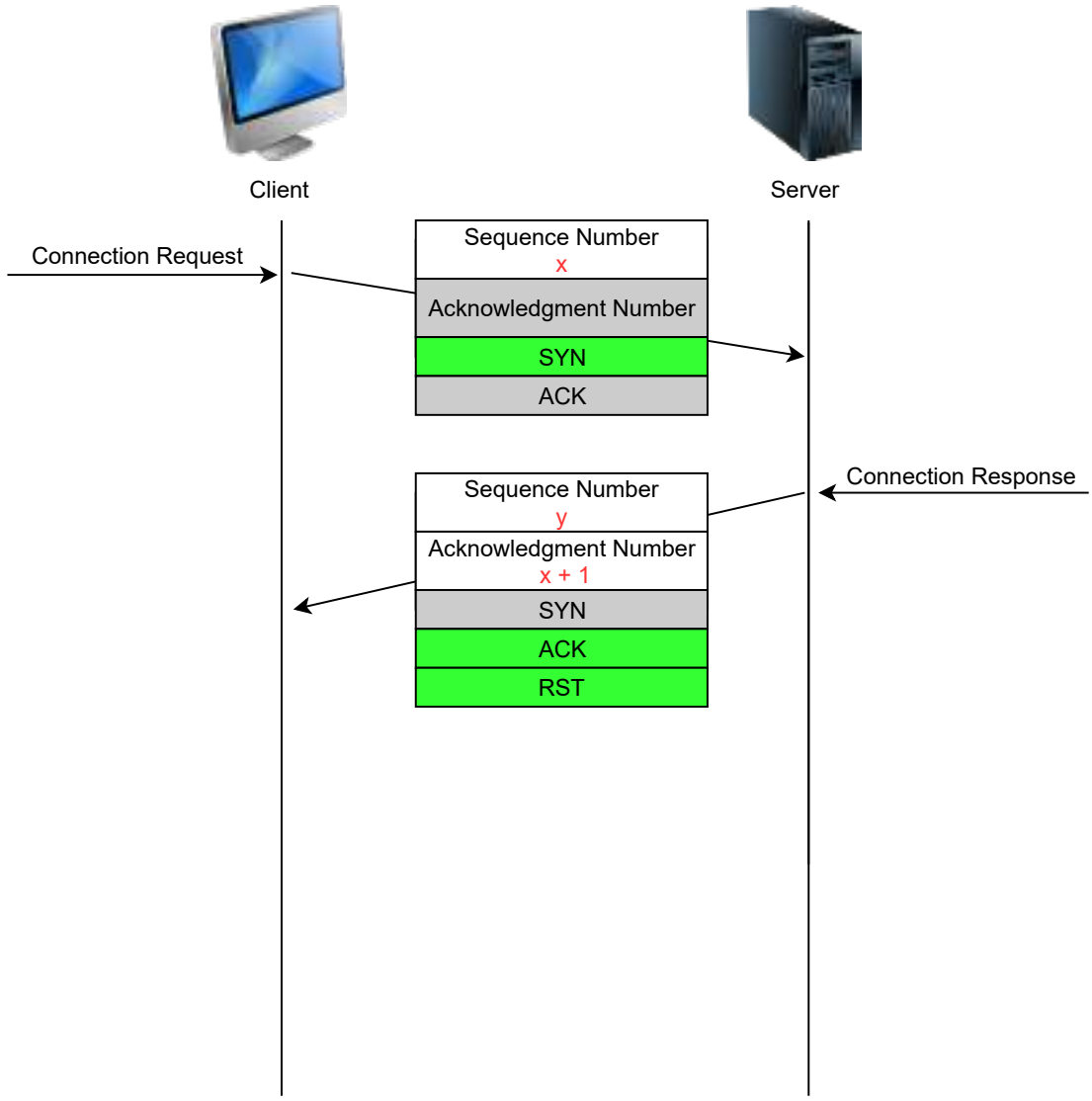
There are other scenarios in which a connection may be refused but these are the common ones. If a process is listening on a port, but the connection is to be refused, the server sends a SYN segment with the following properties:

- Has its *RST* flag set
- Contains the sequence number of the received *SYN* segment as its acknowledgment number.

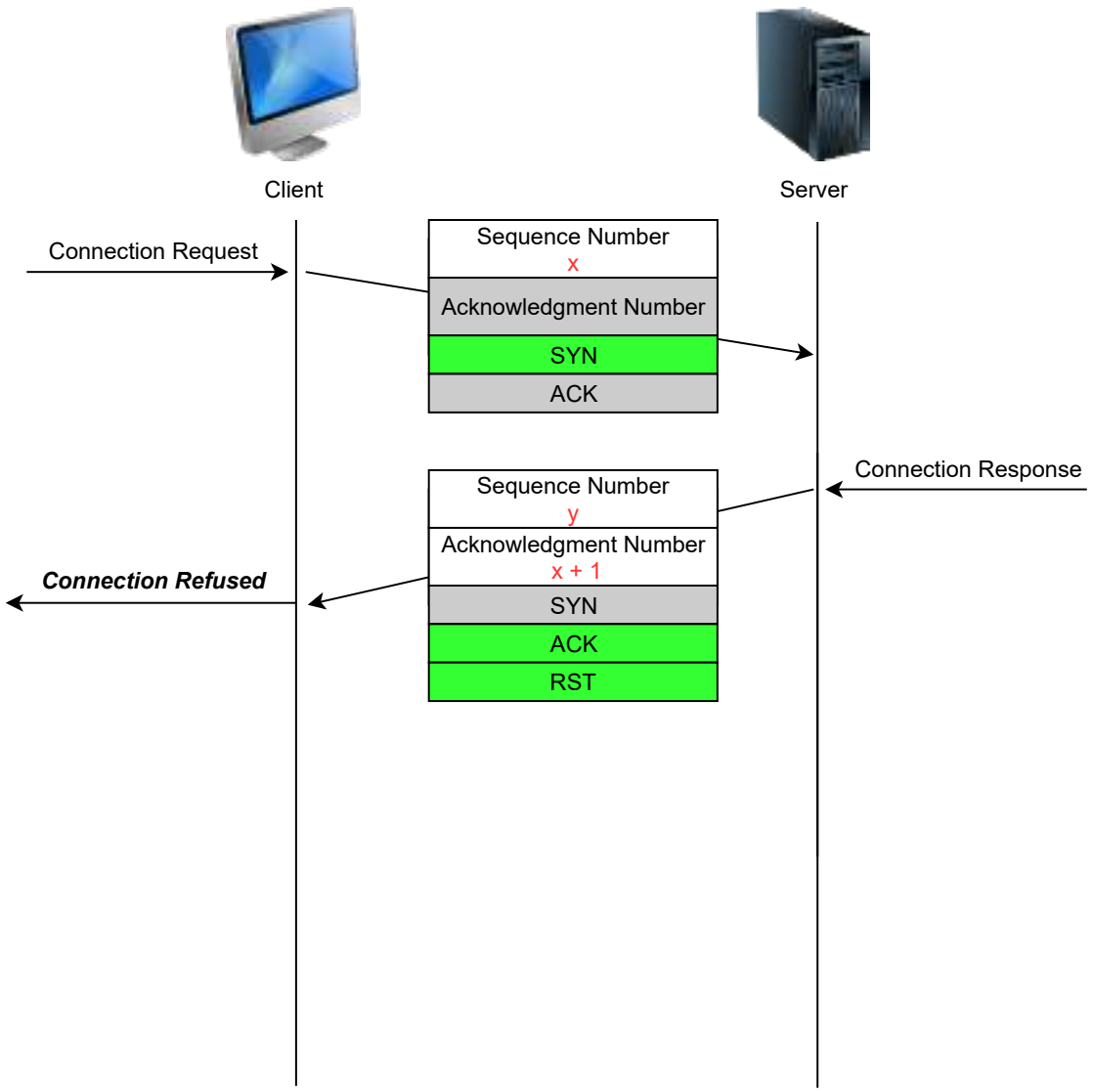
This is illustrated in the slides below. We will discuss the other utilizations of the TCP *RST* flag later in the TCP connection release lesson.




Initiating a Connection




The server refuses the connection with a rst segment



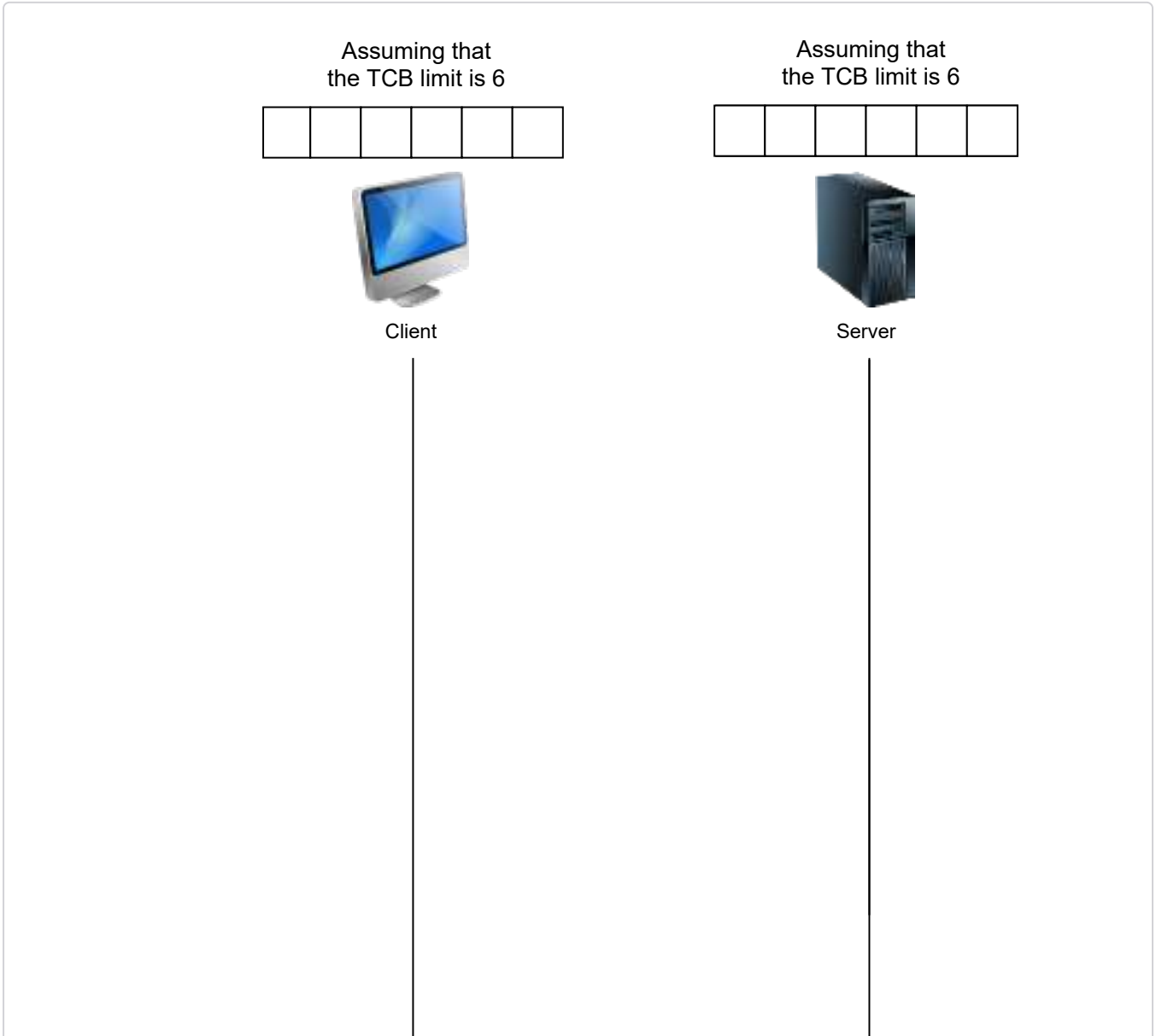

# Syn Flood Attacks #

When a TCP entity opens a TCP connection, it creates a **Transmission Control Block (TCB)** that contains the entire state of the connection, including the local sequence number and sequence number sent by the remote client. Until the mid-1990s, TCP implementations had a limit on the number of 'half-open' TCP connections (TCP connections in the **SYN RCVD** state) which was most commonly at 100. So a machine could only have a 100 'half-open' TCP connections. This was meant to avoid overflowing the entity's memory with TCBs. When the limit was reached, the TCP entity would stop accepting any

*new SYN* segments.

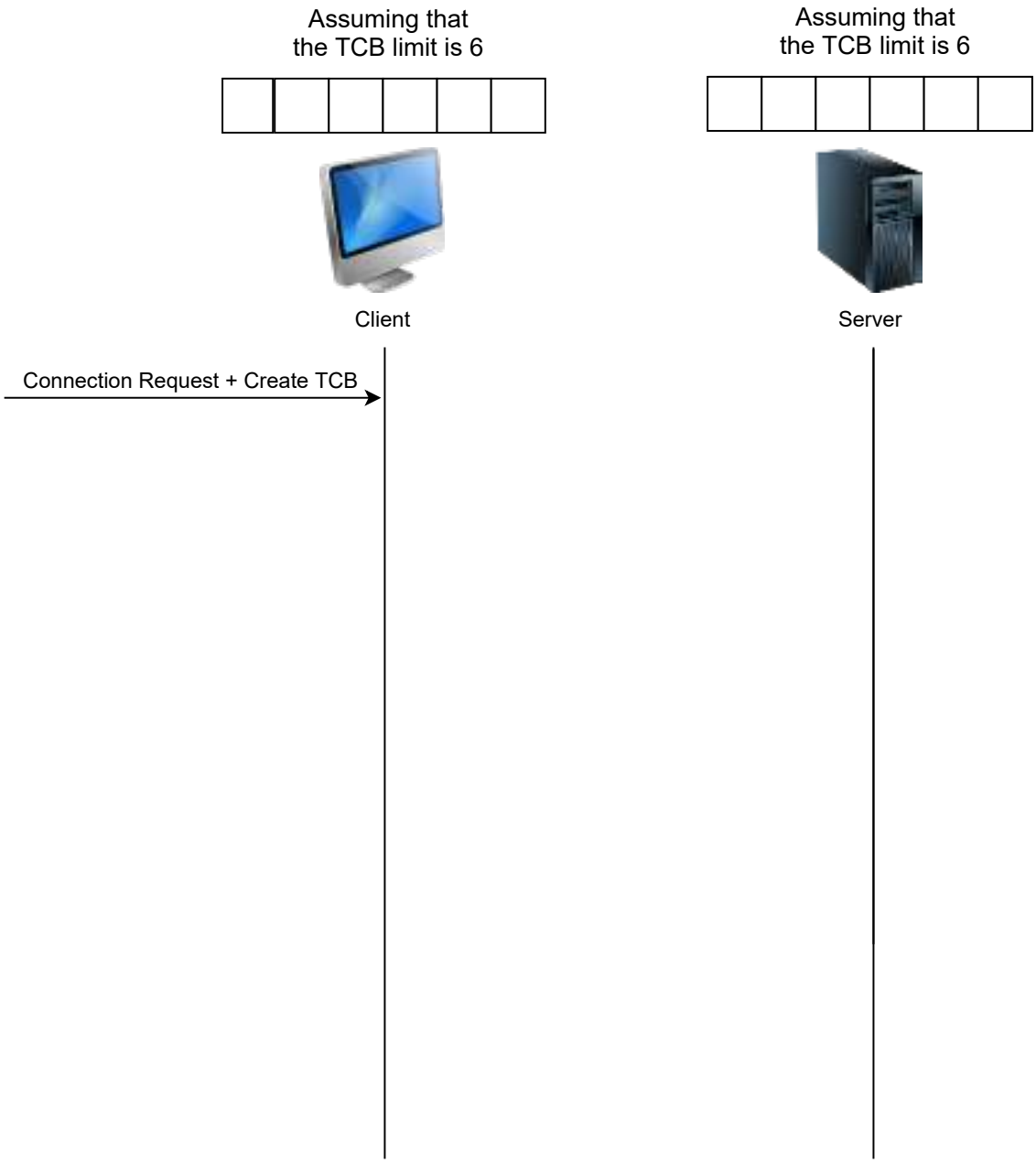
> **Note: The Transmission Control Block:** For each established TCP connection, a TCP implementation must maintain a Transmission Control Block (TCB). A TCB contains all the information required to send and receive segments. This includes:
>
> - the local IP address
> - the remote IP address
> - the local TCP port number
> - the remote TCP port number
> - the current state of the TCP FSM
> - the maximum segment size (MSS)

Here's an illustration of what should happen.

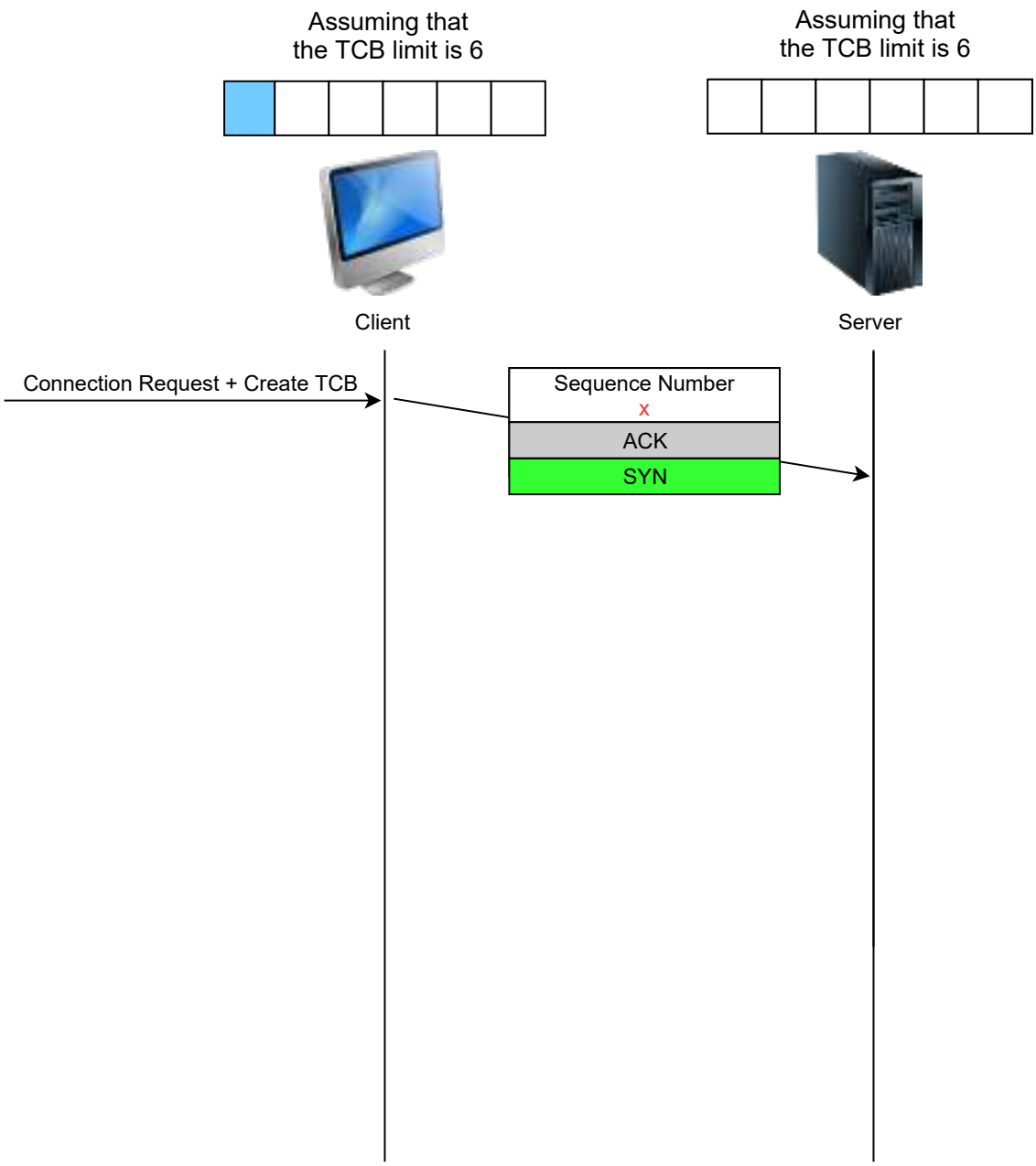


Normal behavior with TCBs

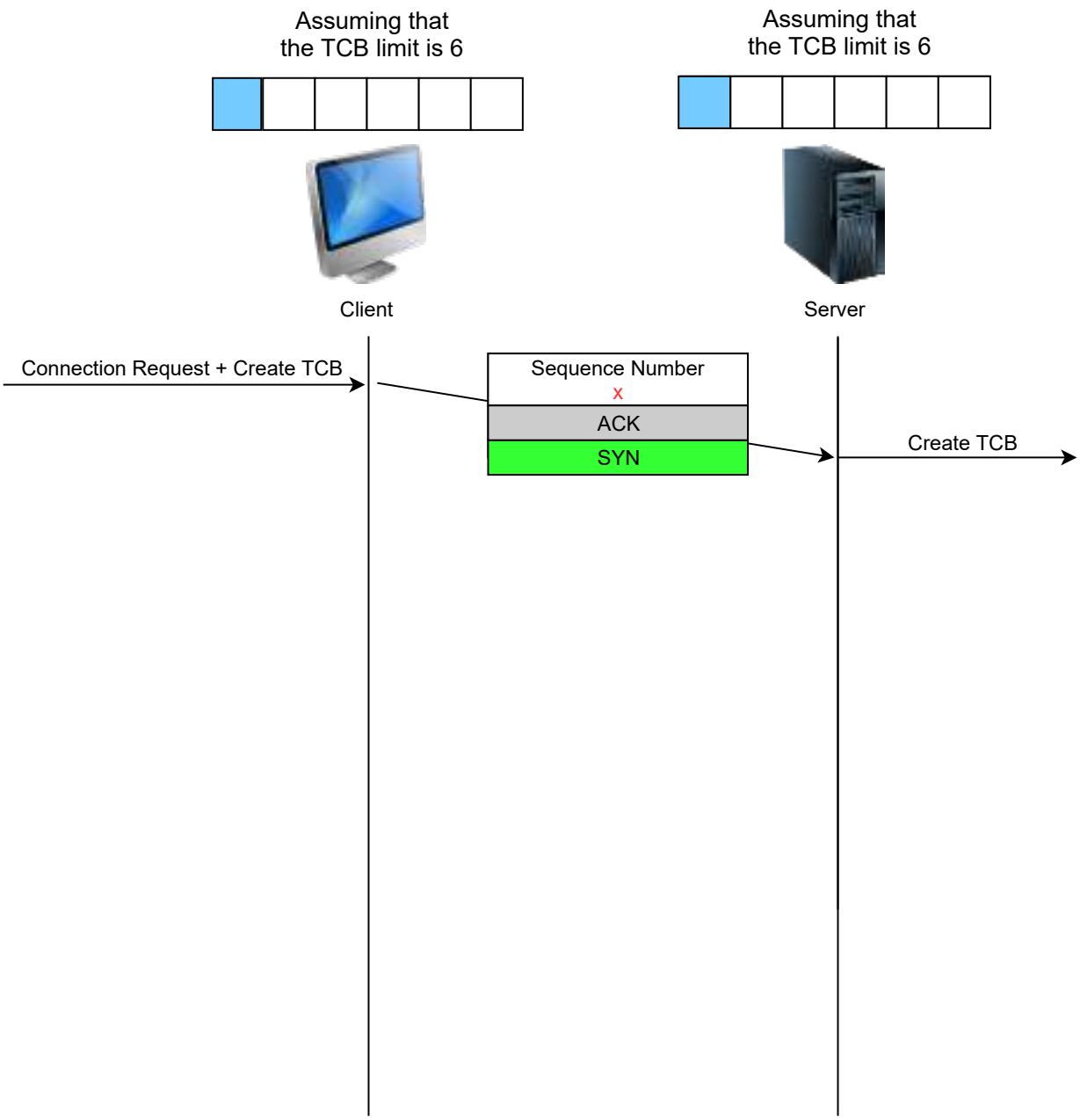




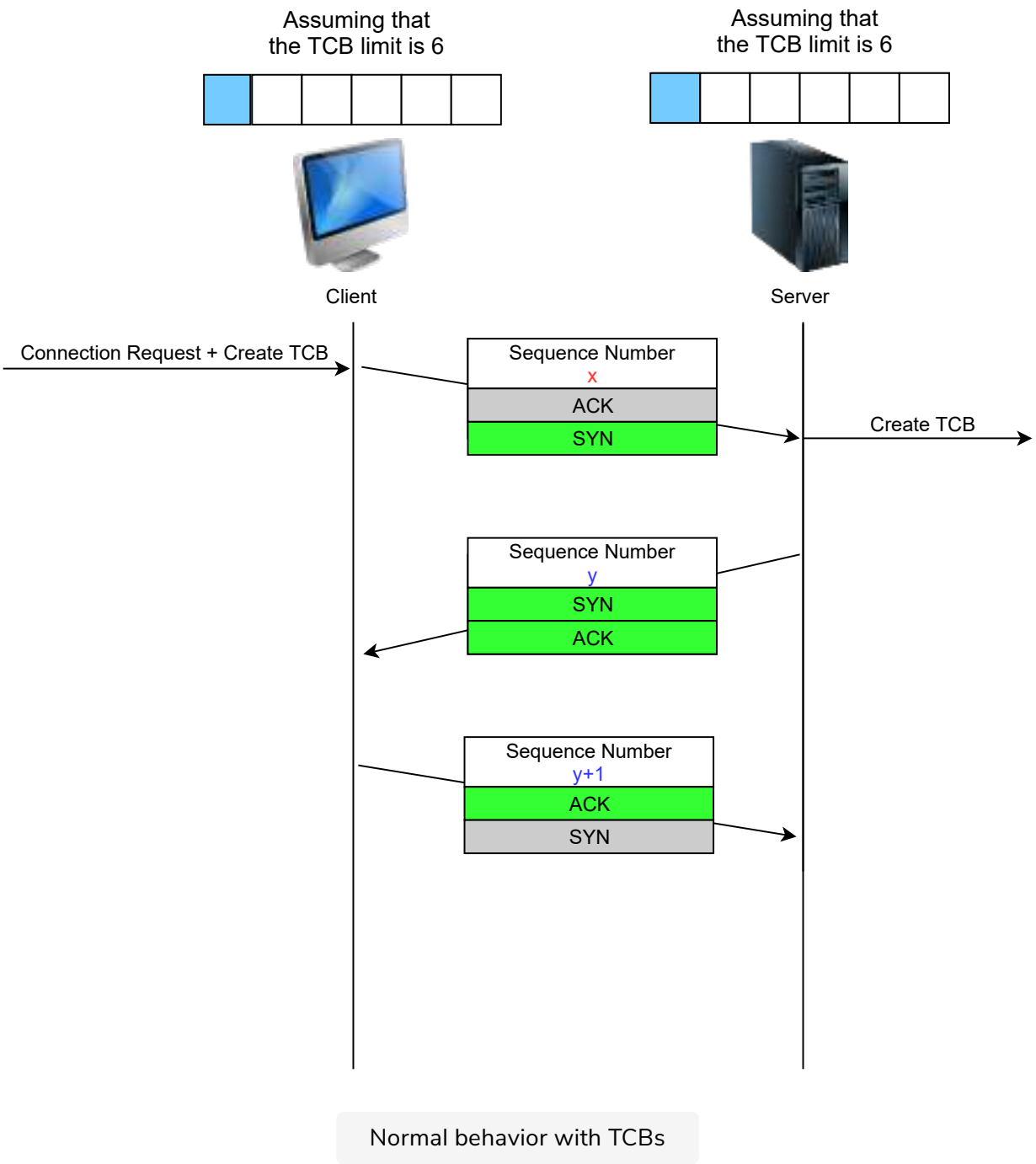

Normal behavior with TCBs




Normal behavior with TCBs




Normal behavior with TCBs




Normal behavior with TCBs

**5** of 5

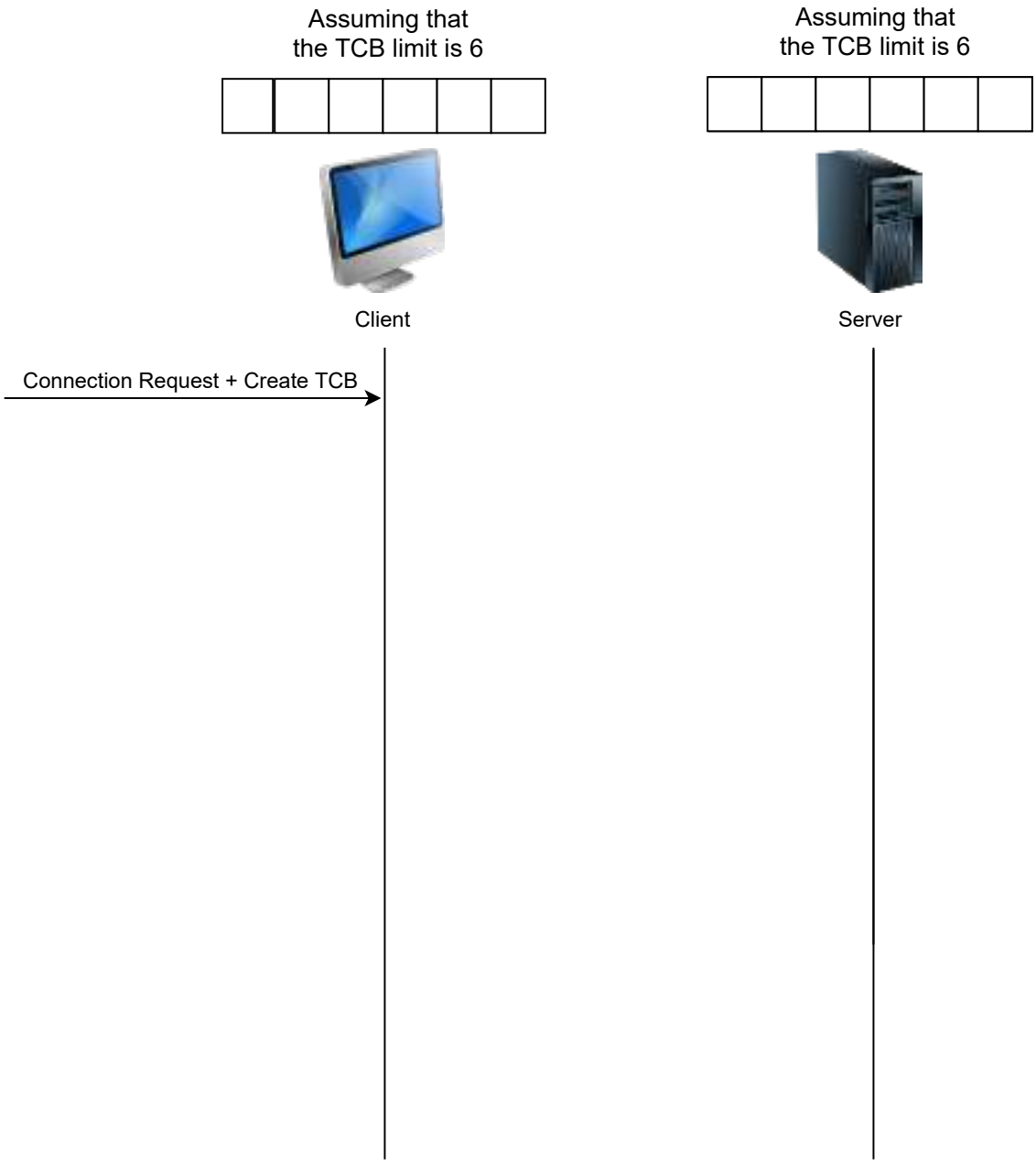
## Syn Cookies #

However, this allowed attackers to carry out an attack where they could render a resource unavailable in the network by sending it valid messages. Such attacks are called **Denial of Service (DoS)** attacks because they deny the user(s) a service. Here's how this one was carried out:

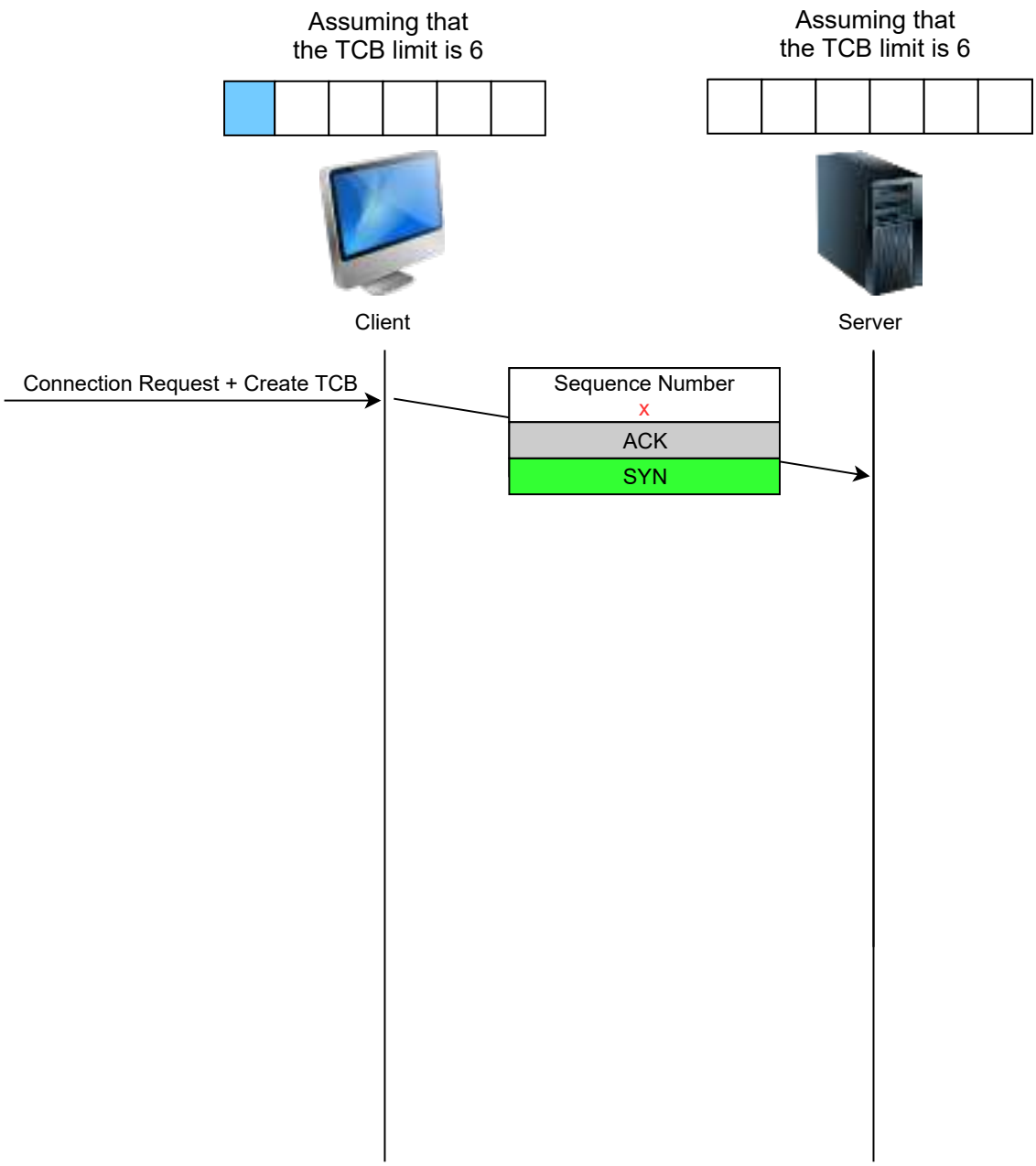
1. The attacker would send a few 100 SYN segments every second to a server

2. The attacker would not reply to any received *SYN+ACK* segments
3. To avoid being caught, the attacker would send these *SYN* segments with a different IP address from their own IP address.
4. Once a server entered the **SYN RCVD** state, it would remain in that state for several seconds, waiting for an ACK and not accepting any new, possibly genuine connections, thus being rendered unavailable.

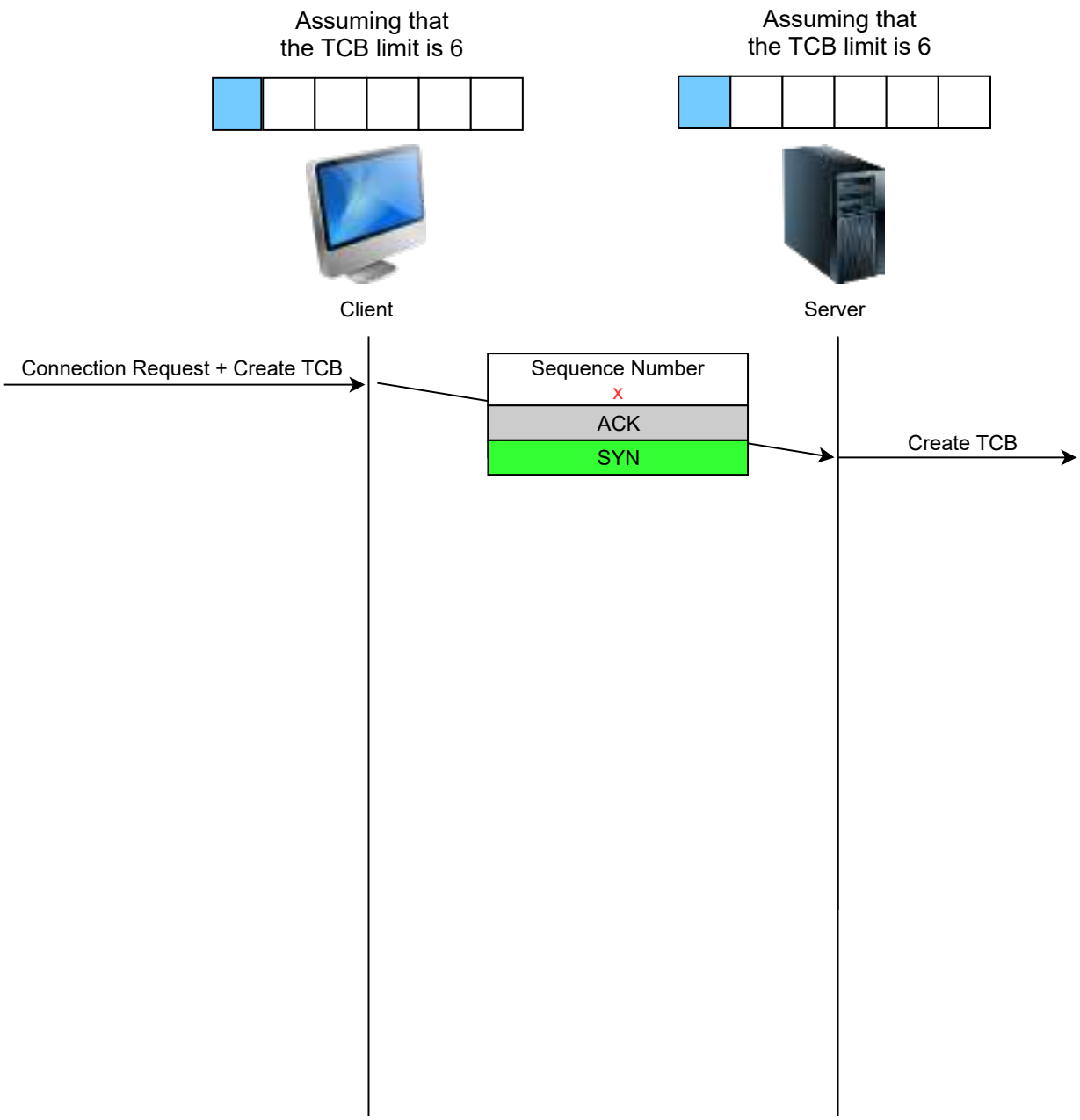
Here are some slides depicting a **SYN flood attack**:



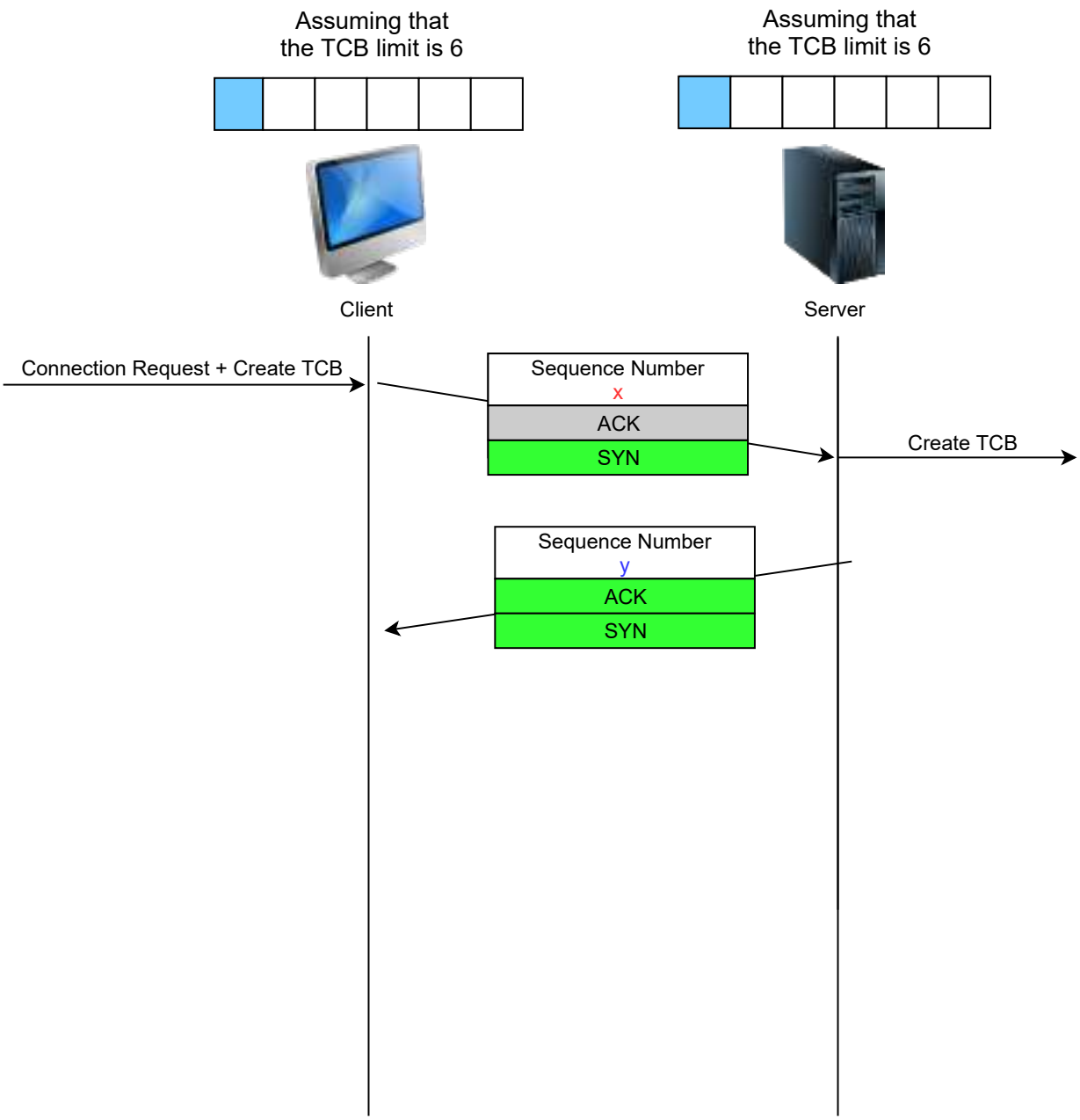

syn flood attack



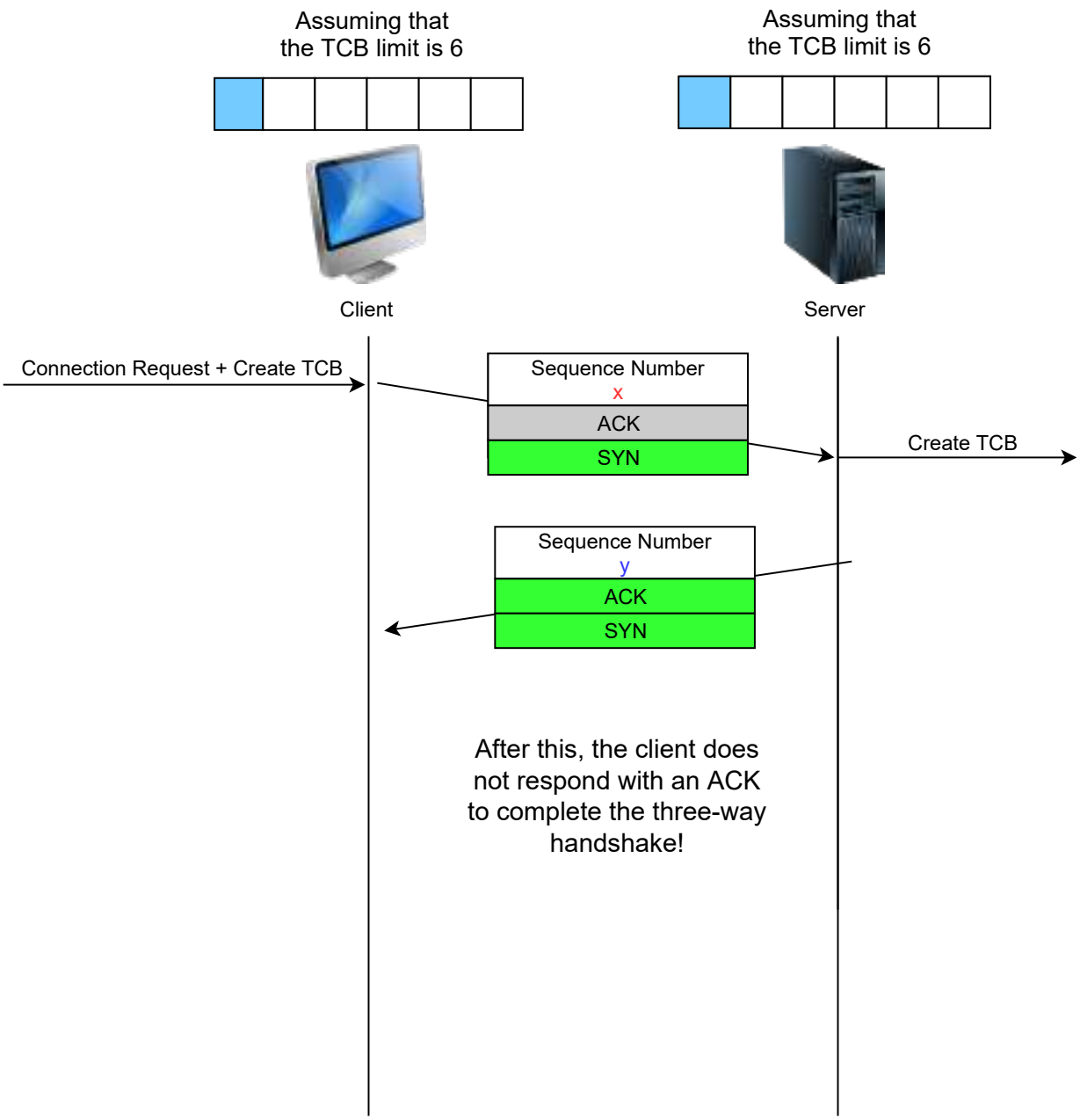

syn flood attack



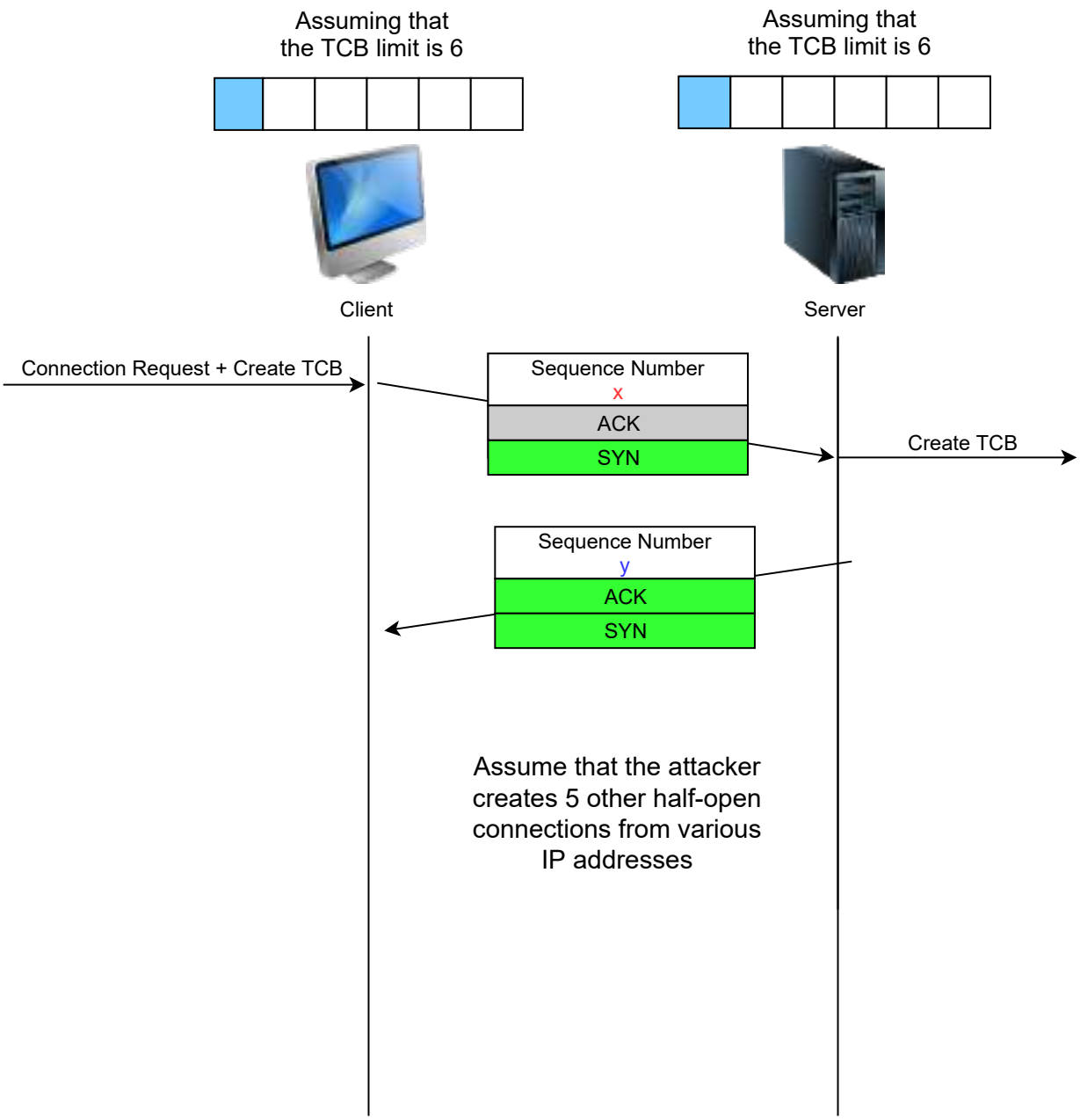

syn flood attack



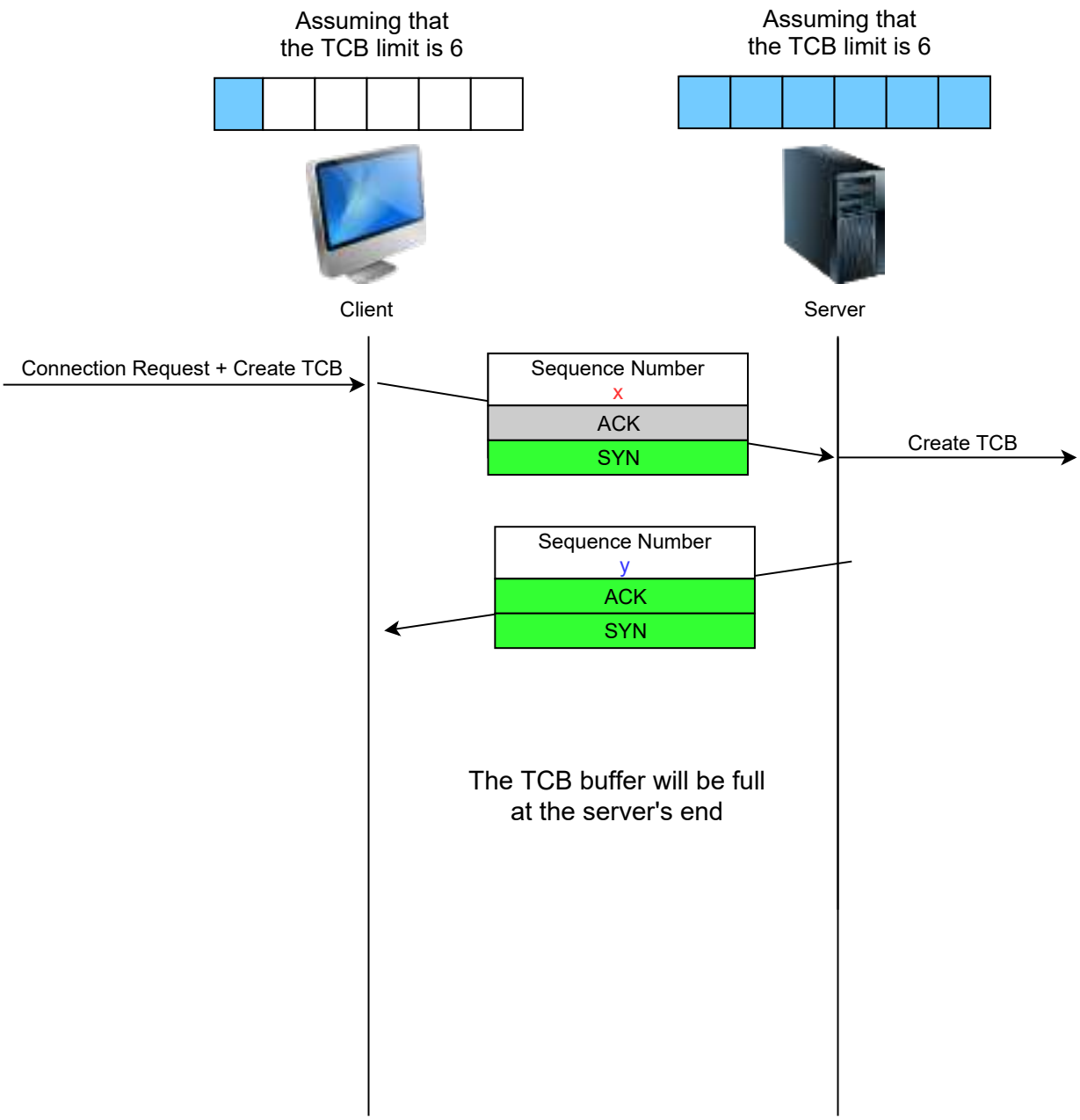

syn flood attack



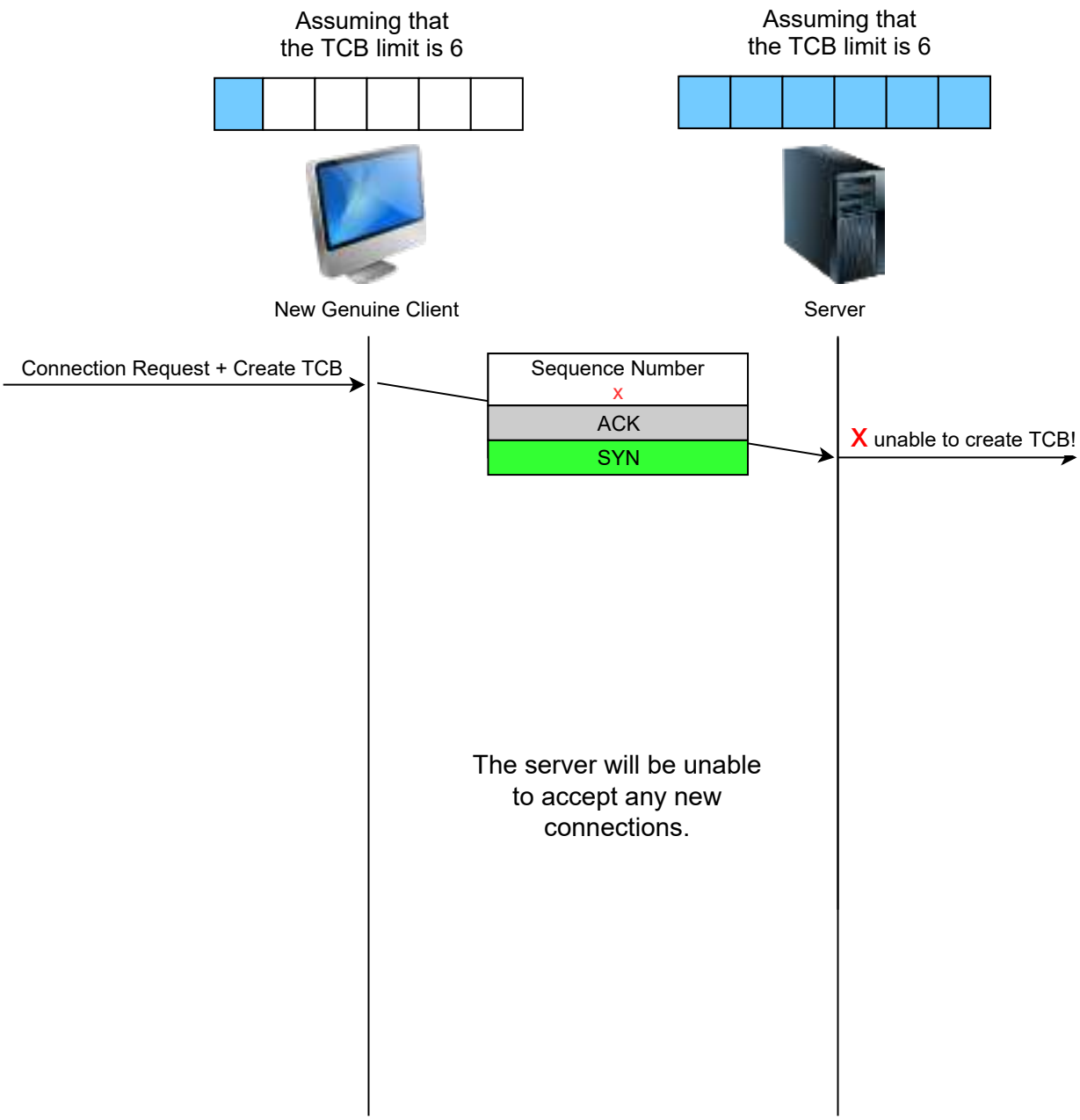

syn flood attack




syn flood attack




syn flood attack




To avoid these **SYN flood attacks**, newer TCP implementations reply directly with SYN+ACK segments and wait until the reception of a valid ACK to create a TCB.

The goal is to not store connection state on the server immediately upon reception of a *SYN* packet. But, without this information, the server cannot tell if a subsequent *ACK* it receives is from a legitimate client that had sent a benign *SYN* packet. One way to do it is to verify that if the acknowledgement

number contained in the *ACK* packet is $y$, then the server had sent a sequence number $y - 1$ in the *SYN+ACK* packet. But, again, if we are remembering the initial sequence number for each *SYN* packet, we are back to square one - remembering connection state. The way **SYN Cookie** solves this problem is to use a function that uses some information from the client's *SYN* packet and some information from the server side to calculate a random initial sequence number. This number, say, $y - 1$ is sent to the client in a *SYN + ACK* message. If an *ACK* packet is later received with a sequence number $y$, using some packet header fields and some server side information, a reverse function can verify that the acknowledgement number is valid. If not, the connection is refused, otherwise a TCB is created and a connection is established.

The advantage of *SYN* cookies is that the server would not need to create and store a TCB upon reception of the *SYN* segment.

## Retransmitting Lost Segments #

Since the underlying Internet protocol provides an unreliable service, the *SYN* and *SYN+ACK* segments sent to open a TCP connection could be lost. Current TCP implementations start a **retransmission timer** when the first *SYN* segment is sent. This timer is often set to three seconds for the first retransmission, and then doubles after each retransmission (RFC 2988). When the timer expires, the segment is retransmitted. TCP implementations also enforce a maximum number of retransmissions for the initial *SYN* segment. Note that the same technique is used for all TCP segments, not just connection establishment segments.

We'll look at this in detail when we study TCP reliability in an upcoming lesson!

## Quick Quiz! #

1 A TCP connection establishment request may be refused due to which of the following reasons?

That is it for TCP connection establishment. What we'll cover next is how TCP releases a connection.