Analysis of TCP flags using Wireshark

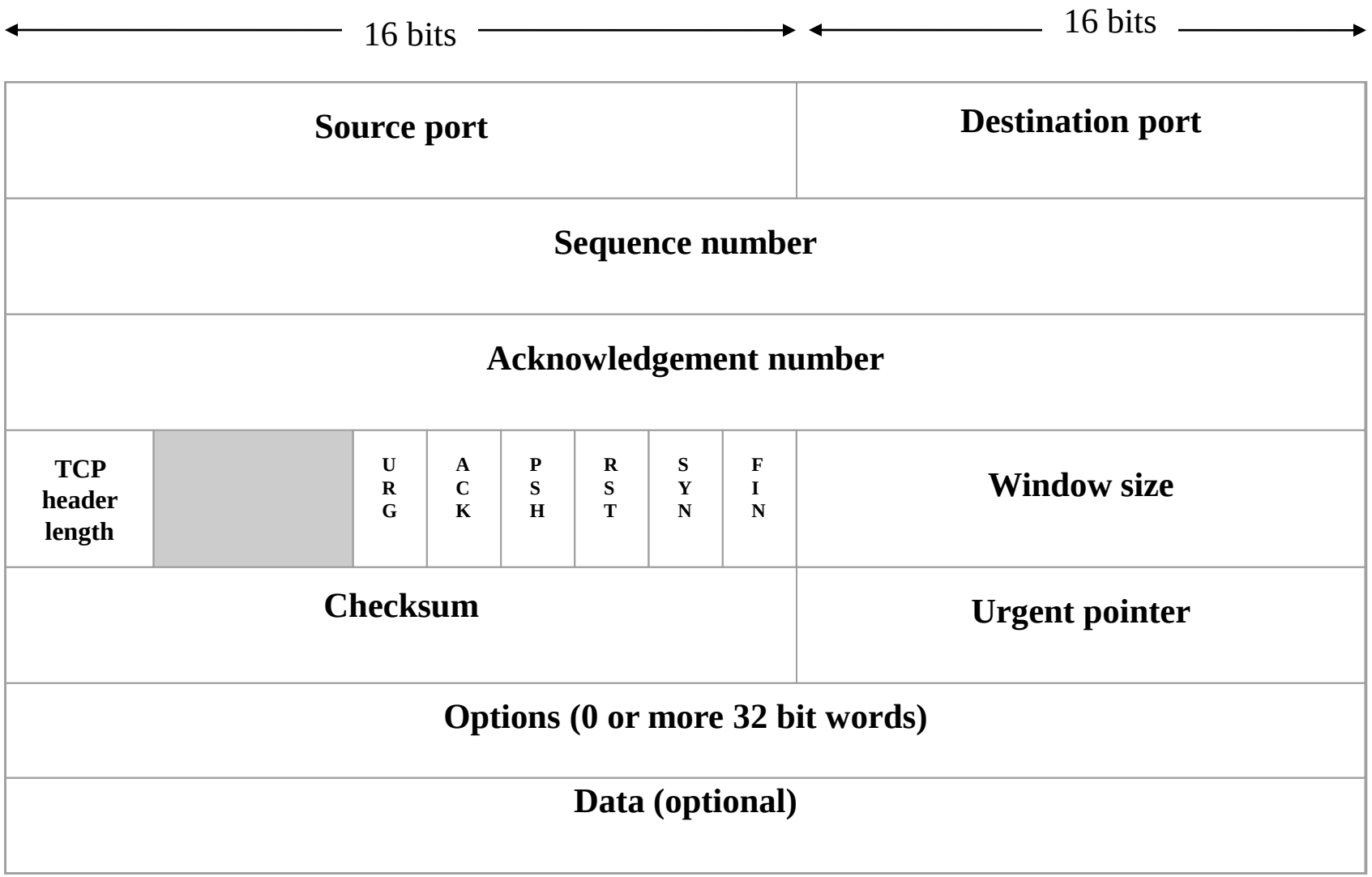
Sujoy Saha

Assistant Professor

National Institute of Technology

NIT Durgapur

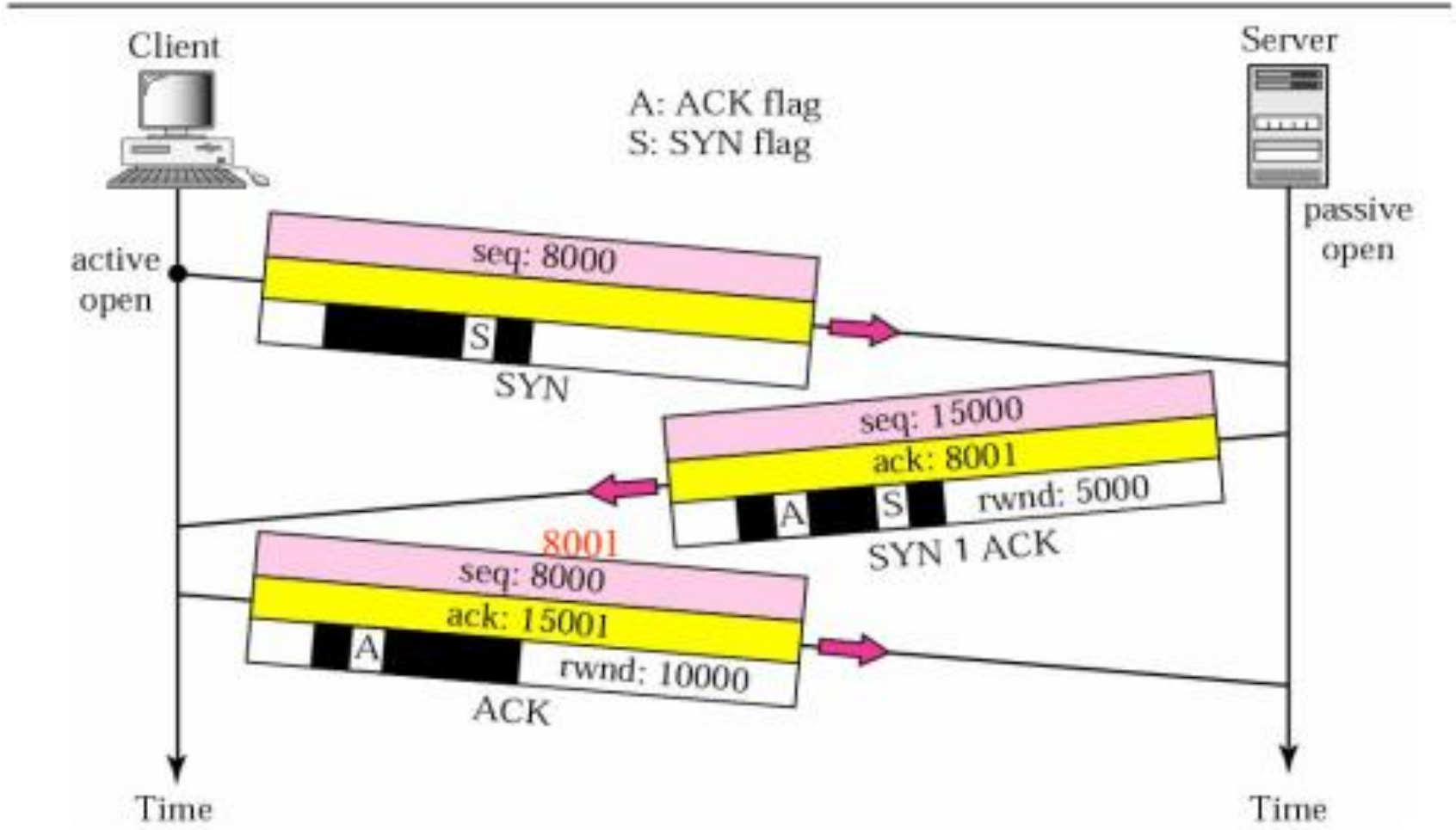
TCP Header


16 bits
16 bits
Source port
Destination port
Sequence number
Acknowledgement number
TCP header length
URG
ACK
PSH
RST
SYN
FIN
Window size
Checksum
Urgent pointer
Options (0 or more 32 bit words)
Data (optional)

Connection Establishment

- Example, a client wants to make a connection to a server
  - Server performs the *passive open*
    - Tell TCP that it is ready to accept a connection
  - Client performs the *active open*
    - Tell TCP that it needs to be connected to the server

Three-way Handshaking



3 Way handshaking

The client sends the first segment, a *SYN* segment

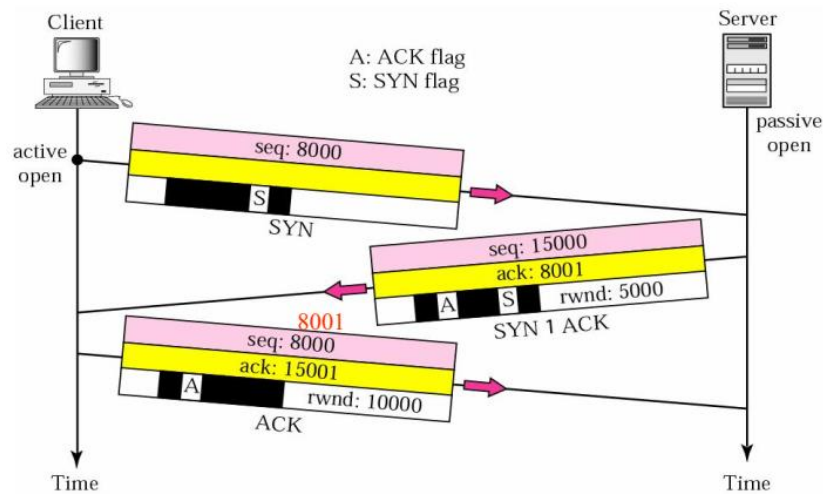
- Set the *SYN* flag
- The segment is used for synchronization of sequence number
  - *Initialization sequence number (ISN)*
- If client wants to define MSS, add MSS option
- Does not contain any acknowledgment number
- Does not define the window size either
- Although a control segment and does not carry data
  - But consumes *one sequence number*

Receiver Window Size (rwnd)

- The **TCP window size**, or as some call it, the **TCP** receiver **window size**, is simply an advertisement of how much data (in bytes) the receiving device is willing to receive at any point in time.
- The receiving device can use this value to control the flow of data, or as a flow control mechanism.
- The **maximum segment size** (**MSS**) is a parameter of the *options* field of the TCP header that specifies the largest amount of data, specified in bytes, that a computer or communications device can receive in a single TCP segment. It does not count the TCP header or the IP header

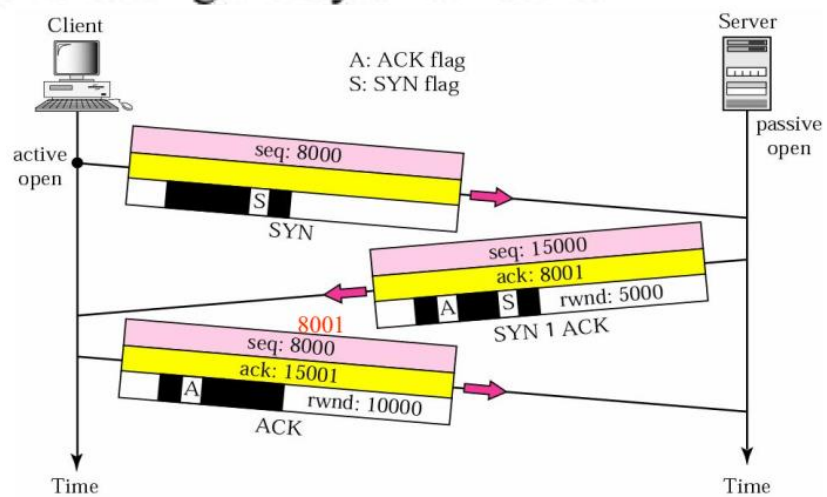
Three-way handshaking (Cont.)

2. The server sends a second segment, a ***SYN + ACK*** segment
   - Set the *SYN* and *ACK* flag
   - *Acknowledge* the receipt of the first segment using the ACK flag and acknowledgment number field
     - Acknowledgment number = client initialization sequence number + 1
     - Must also define the receiver window size for flow control
   - *SYN* information for the server
     - Initialization sequence number from server to client
     - Window scale factor if used
     - MSS is defined



Three-way handshaking (Cont.)

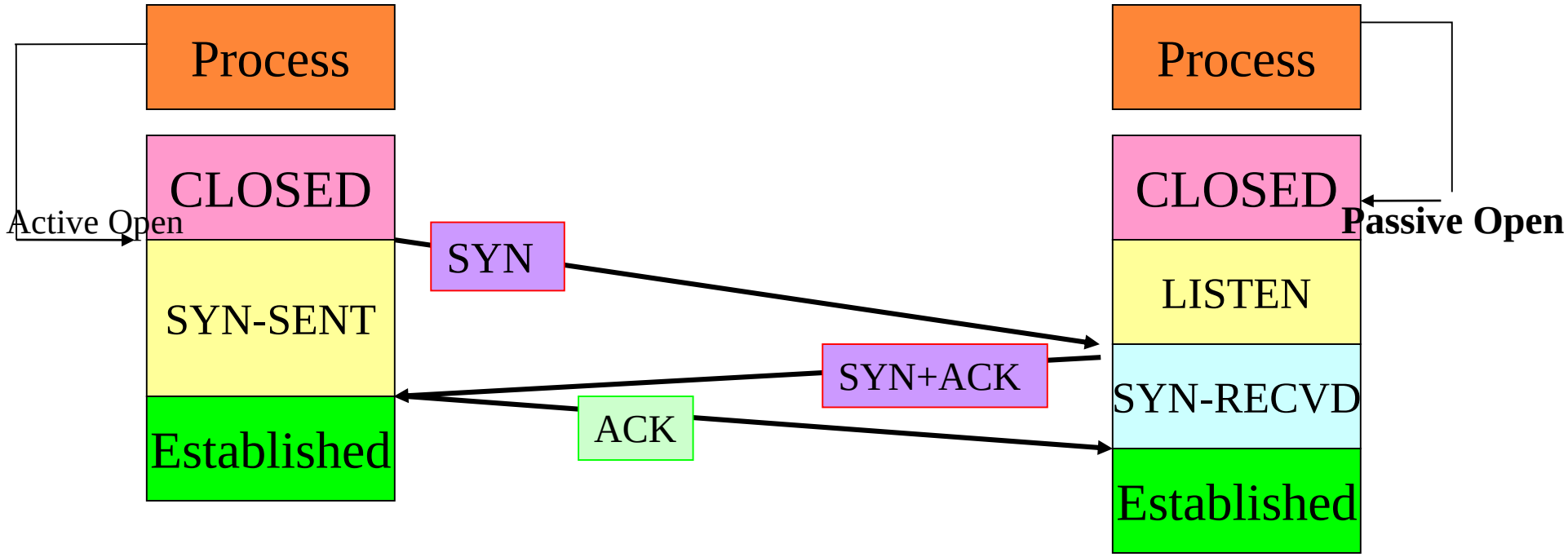
3. The client sends the third segment, ***ACK*** segment
   - *Acknowledge* the receipt of second segment
     - ACK flag is set
       - Acknowledgement number = server initialization sequence number + 1
     - Must also define the server window size
       - Set the window size field
     - The sequence number is the same as the one in the SYN segment
       - ACK segment does not consume any sequence number
   - However, in some implementation, data can be sent with the third packet
     - Must have a new sequence number showing the byte number of the first byte in the data



Note:

A SYN segment cannot carry data, but it consumes one sequence number.

THREE-WAY HANDSHAKING – Sate Transition Diagram



SYN FLOODING ATTACK

- A malicious attacker sends a larger number of SYN segment to a server
  - Each with a fading source IP address

- The server will runs out of resource and may crash
  - ***Denial of service attack***

SYN FLOODING ATTACK

- This SYN Flooding attack belongs to a group of security attacks known as a **Denial of Service Attack.**

- An attacker monopolizes a system with so many service requests that the system collapses and denies service to every request.

SYN Flooding Attack (Cont.)

- Possible solutions
  - Impose a limit of connections requested during a period of time
  - Filter out datagrams coming from unwanted source addresses
  - Postpone resource allocation until the entire connection is set up
    - SCTP uses strategy, called ***cookie***

Push Flag

- **Normal TCP behavior**: When a sender transmits data over TCP, the data is typically buffered before being sent.
  - The TCP layer waits until there is enough data to fill a buffer (for efficiency) before sending it in a segment.
  - The receiving side, too, will buffer incoming data and may wait for more data before passing it to the application layer.
- **TCP Push flag (PSH)**: When the **PSH flag** is set, it indicates to the receiving end that the data should be processed immediately rather than waiting for the buffer to fill.
  - Essentially, it signals the receiver to deliver the data to the application layer without further delay.

Push Flag

- If you are sending small packets (possibly less than the buffer size), you may want them to be processed immediately, without waiting for the buffer to fill up.
- This is particularly useful in real-time applications, such as interactive sessions (e.g., SSH or telnet), where immediate delivery of each small piece of data is important.
- You can write a simple client and server program to send small data, like a short string. Even in this case, the TCP/IP protocol uses the PUSH flag, as the kernel doesn't wait for the buffer to fill before sending the data.

Urgent Flag and Urgent Pointer

- **URG flag**: The **URG** (urgent) flag in the TCP header indicates that the packet contains urgent data.
- **Urgent pointer**: The **urgent pointer** in the TCP header specifies the position in the data stream where the urgent data ends. The urgent pointer is an offset from the sequence number and helps the receiver determine how much of the incoming data is urgent.
- In the context of BSD sockets (which is widely implemented in most operating systems), out-of-band data (urgent data) is sent and received using the MSG_OOB flag with the send() and recv() functions.

Urgent Flag and Urgent Pointer

- **send(socket_fd, buffer, length, MSG_OOB**);
- **Purpose:** MSG_OOB is used to send out-of-band data (urgent data) in TCP. When you use this flag in the send() function, it sets the URG flag in the TCP header, and the kernel uses the urgent pointer to indicate how much of the data is urgent.

- **Urgent Data in TCP:** TCP does not have a true "out-of-band" data concept like some other protocols. Instead, the urgent data is inline with the rest of the data, but it is marked as urgent using the urgent pointer. The urgent data typically must be handled quickly by the receiver.

send() with MSG_OOB flag

```
// Send urgent data using MSG_OOB
if (send(sock, urgent_message, urgent_len, MSG_OOB) < 0) {
        perror("Send urgent data failed");
}
```

- Here, urgent_message is the data sent with the MSG_OOB flag, signaling that it is urgent. The **urgent pointer** will point to the end of this message in the data stream.

recv() with MSG_OOB flag

- recv(socket_fd, buffer, length, MSG_OOB);
  - When receiving data marked as urgent (out-of-band), you use the MSG_OOB flag with recv() to read the urgent portion of the data.
  - The kernel treats urgent data as a separate logical stream, though it is technically part of the in-line data stream.
  - The recv() call with MSG_OOB will only retrieve the urgent data.

Client & Server Data

Client sends data using:

- send(sock, message, length, 0) for normal data.
- send(sock, urgent_message, length, **MSG_OOB**) for urgent data.

Server receives data using:

- read(new_socket, buffer, BUFFER_SIZE) for normal data.
- recv(new_socket, buffer, BUFFER_SIZE, **MSG_OOB**) for urgent data.

RST flag

- If there is **no server running** on the specified port (K), the TCP stack on the server will respond with an **RST flag**, which will reset the connection, and the client will receive an error indicating that the connection failed.
- The RST flag in TCP is used to **reset a connection** and indicate an error condition.
- It's triggered by situations like **invalid connections, crashes, or abrupt termination** of the communication.
- Applications **cannot directly set the RST flag**, but they can cause conditions where the TCP stack will send an RST, such as attempting to connect to an invalid port or terminating connections unexpectedly.

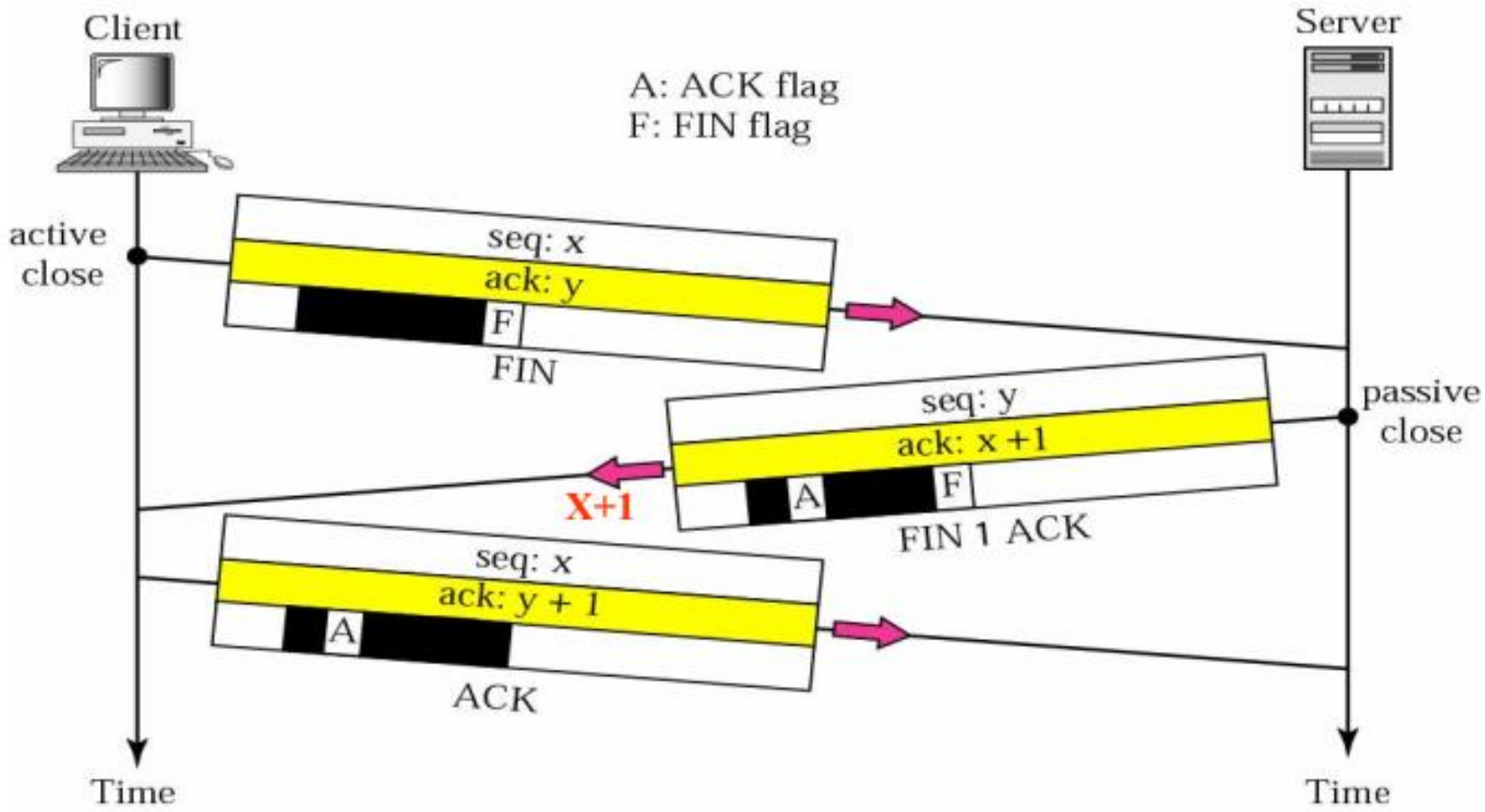
Fin flag

- The **FIN (Finish)** flag in TCP is used to gracefully terminate a connection between two hosts. When a host sends a TCP segment with the **FIN flag**, it indicates that the sender has finished sending data and wants to close the connection.

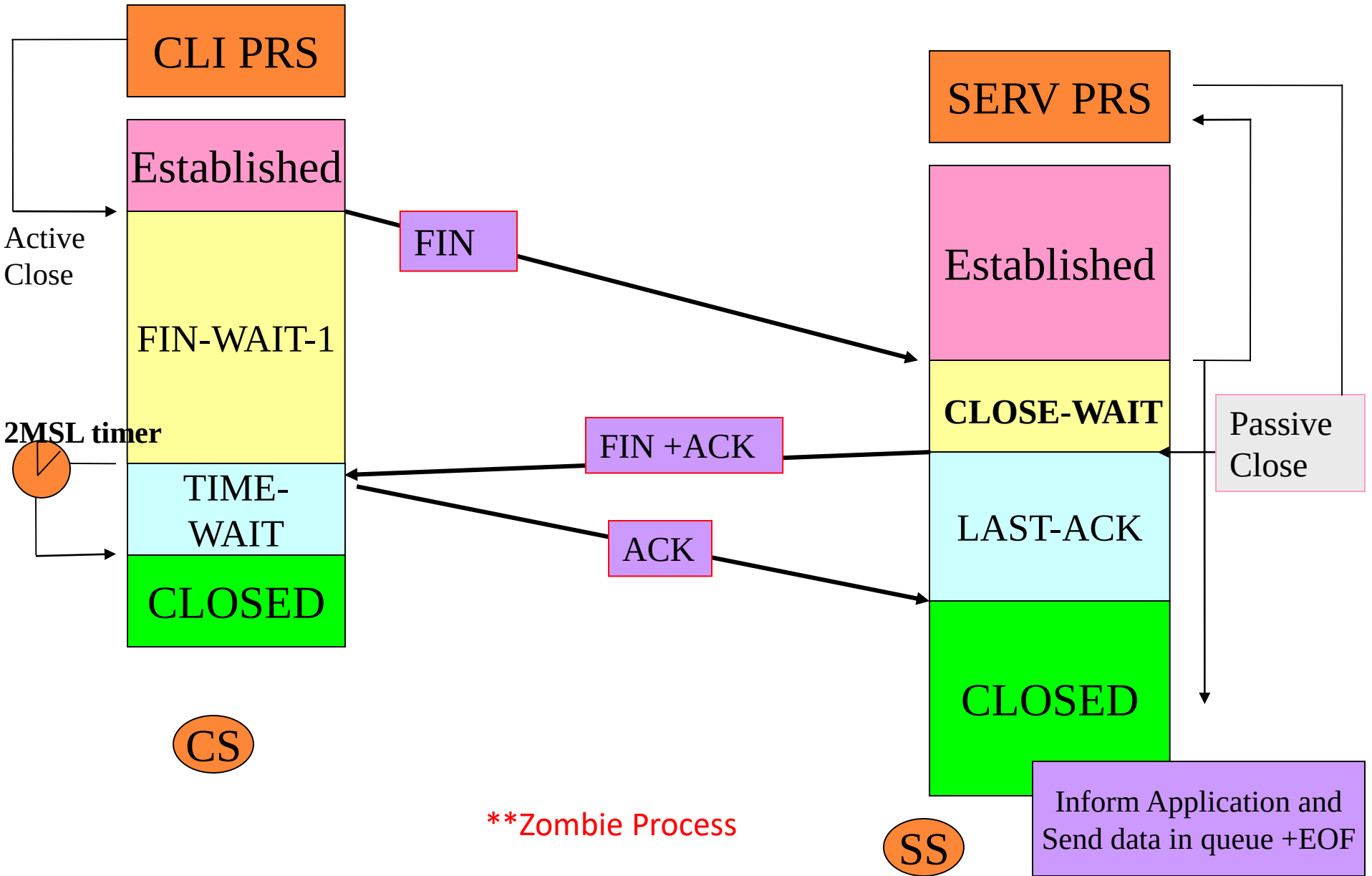
CONNECTION TERMINATION

- Three-way Handshaking
- Simultaneous Close
- Four Way Handshaking

Three-Way Handshaking



THREE-WAY HANDSHAKING (CT)



**Zombie Process

2MSL TIMER

MSL is the maximum time a segment can exist in the internet before it is dropped.

The common value for MSL is between 30 seconds and 1 minutes.

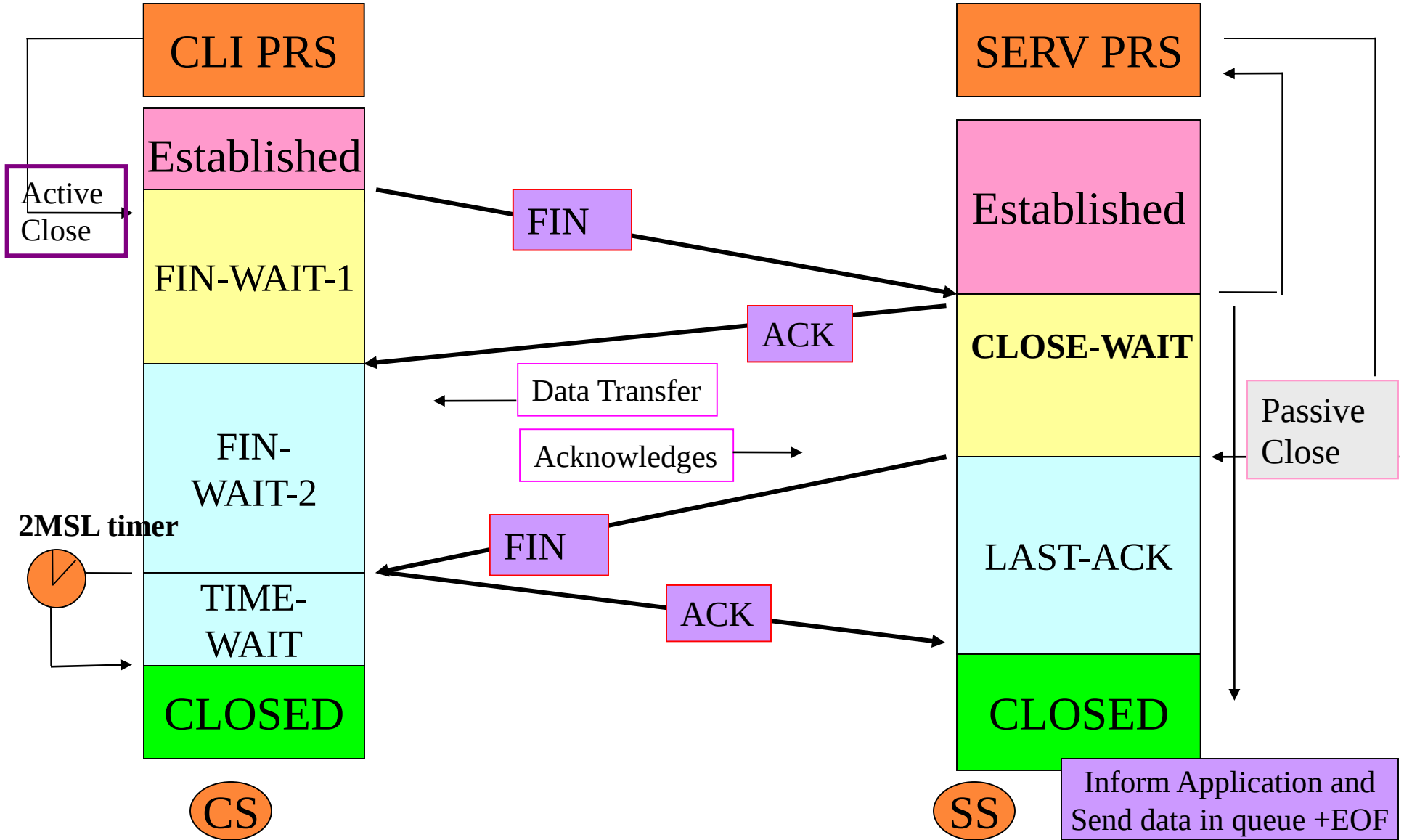
Main Reasons for the existence of the TIME-WAIT and the 2MSL Timer:-

If The Last ACK segment is lost → server assumes that last FIN (FIN+ACK) segment is lost and resends it.

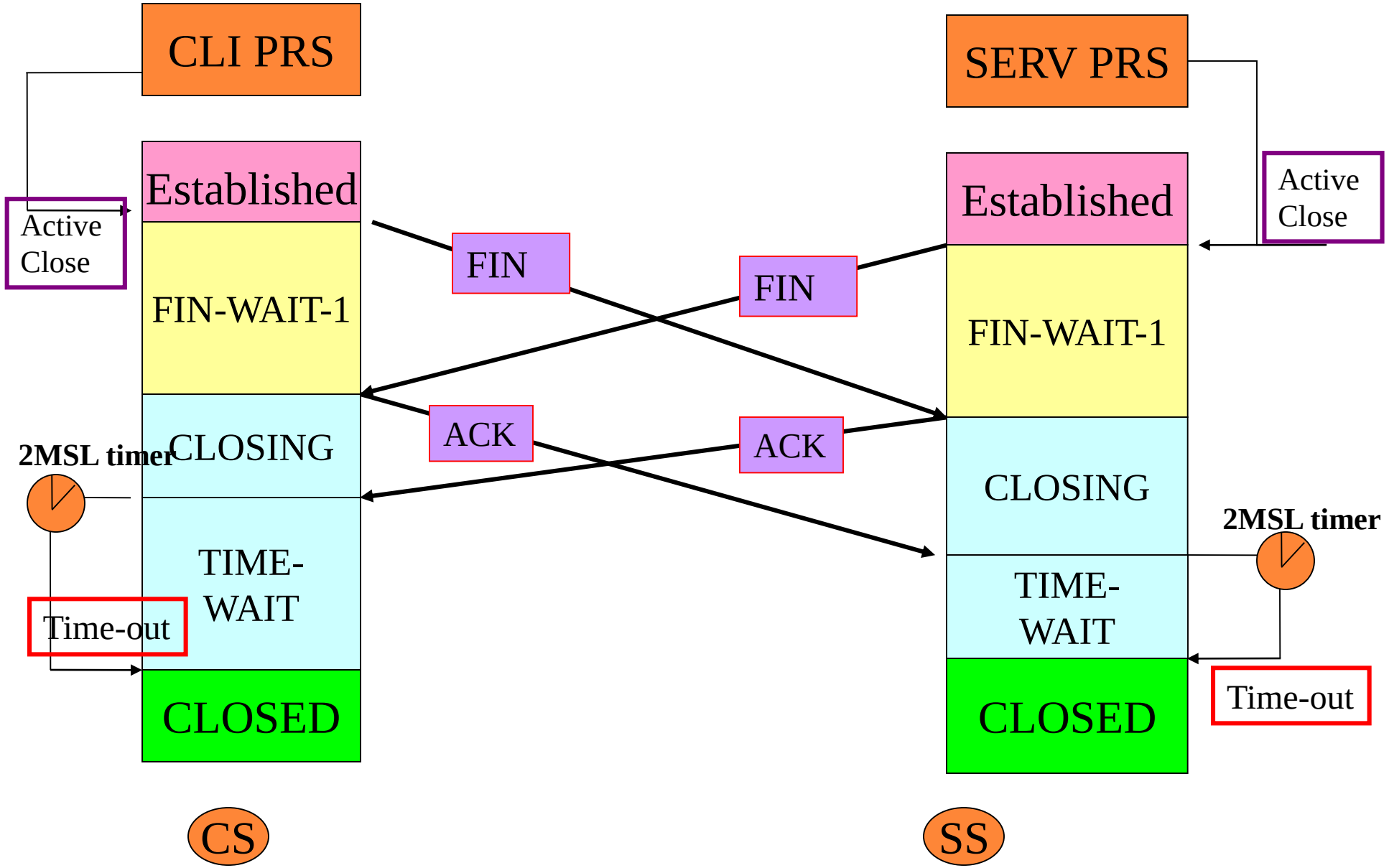
if the client goes to the CLOSED state and close the connection Before the 2MSL timer then server never close the connection

During the TIME-WAIT state if a new FIN arrives the client sent New ACK & restarts the 2MSL timer.

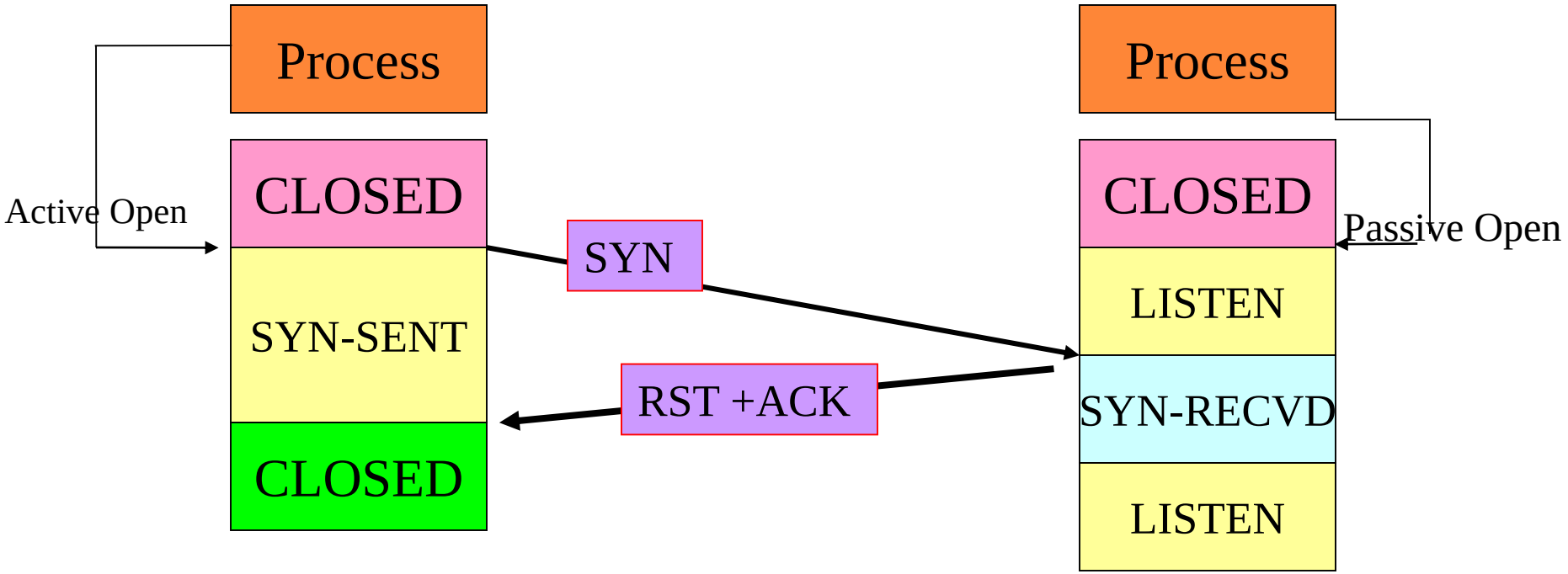
FOUR-WAY HANDSHAKING (CT)



SIMULTANEOUS CLOSE

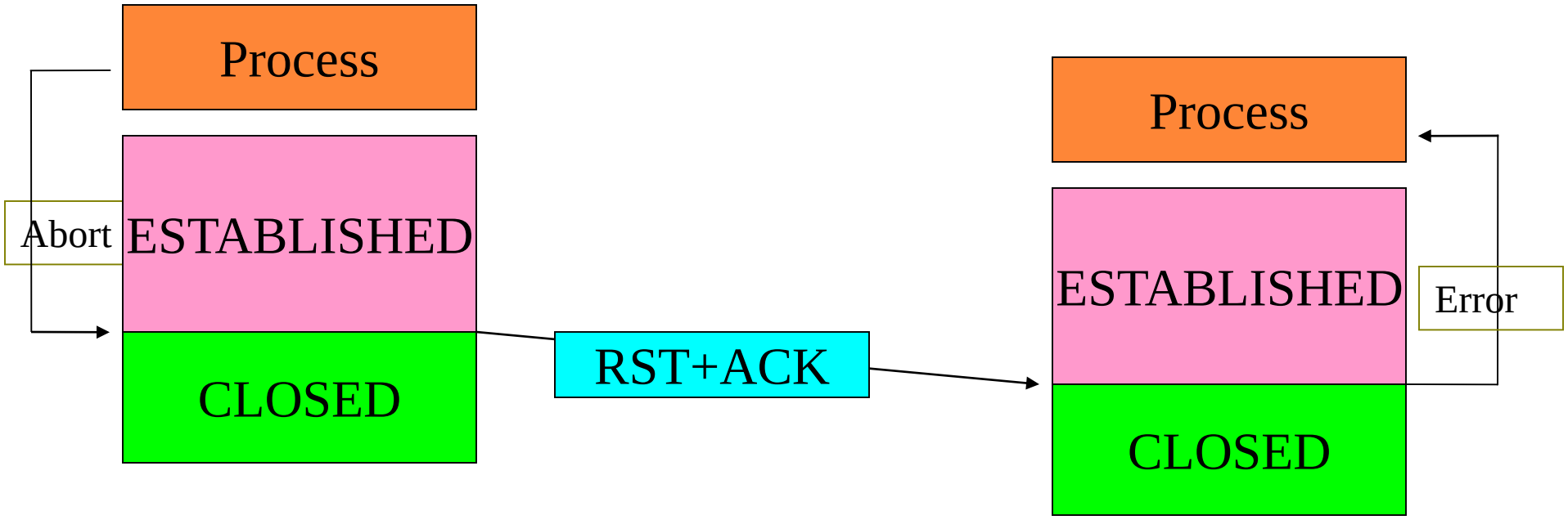


DENYING A CONNECTION



Server Denies the connection because the destination port number
In the SYN segment defines a server that is not in the LISTEN state
At the moment.

ABORTING A CONNECTION



This can happen if the process has failed (infinite loop) or does not Want the data in queue to be sent. (Example ctrl +c)

Thanks