Fall '20 COSE322-00

System Programming

TCP in Linux

2020. 11. 10.

Contents

- **TCP Review & Basic implementations**
  - TCP review
  - Data structures for TCP
- **Sending segments in TCP**
- **Receiving segments in TCP**

TCP Review & Basic implementations

- TCP review
- Data structures for TCP

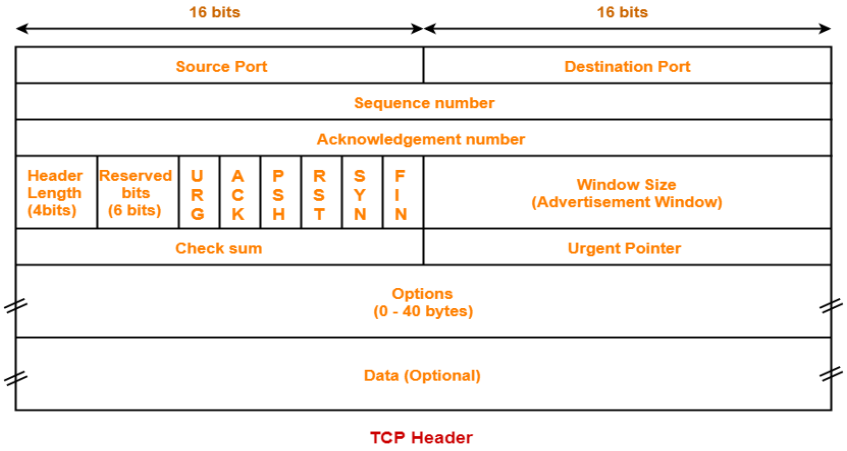
TCP Review

- **The most complex part of the networking stack in the Linux**
- **TCP contributes to the vast majority of the traffic in the Internet**
  - FTP, Telnet, HTTP, etc.
- **Important functions**
  - Establish a reliable communication between a sender and a receiver by retransmitting non-acknowledged packets
  - Implement flow control by sliding window mechanism
  - Implement congestion control by throttling the sending rate when congestion is detected

TCP Review: ACK

- **TCP guarantees reliable and in-order delivery**
  - TCP receivers confirm the receipt of data to the sender
  - Ack is the number to notify how much I received so far
  - TCP uses sequence numbers to keep track of transmitted and acknowledged data

- **For ack,**
  - Receiver writes a sequence number in the Ack field
  - sets the Ack flag

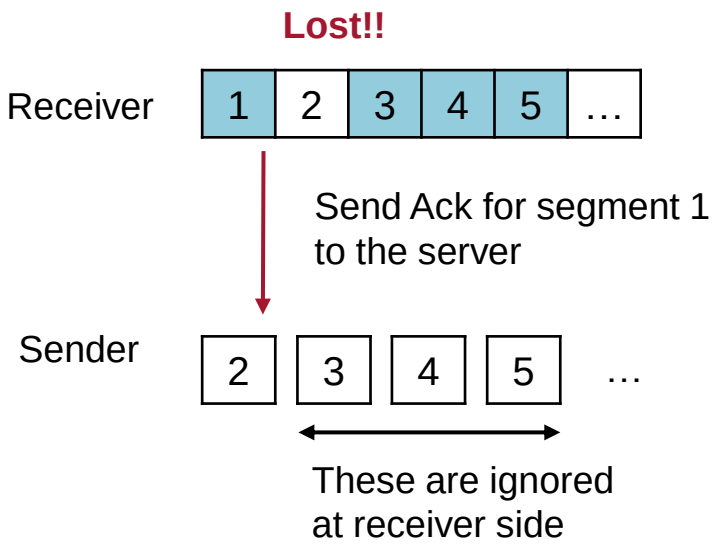


TCP Header

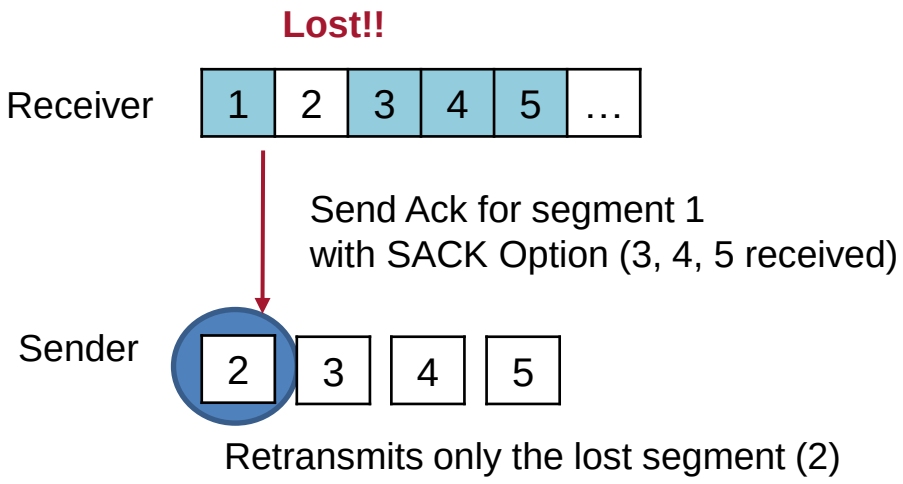
TCP Review: SACK

❖ SACK (Selective ACK)

- The receiver can acknowledge non-continuous blocks of data
- Receiver tells the sender not only the next in-sequence expected byte, but also a range of bytes received out-of-order



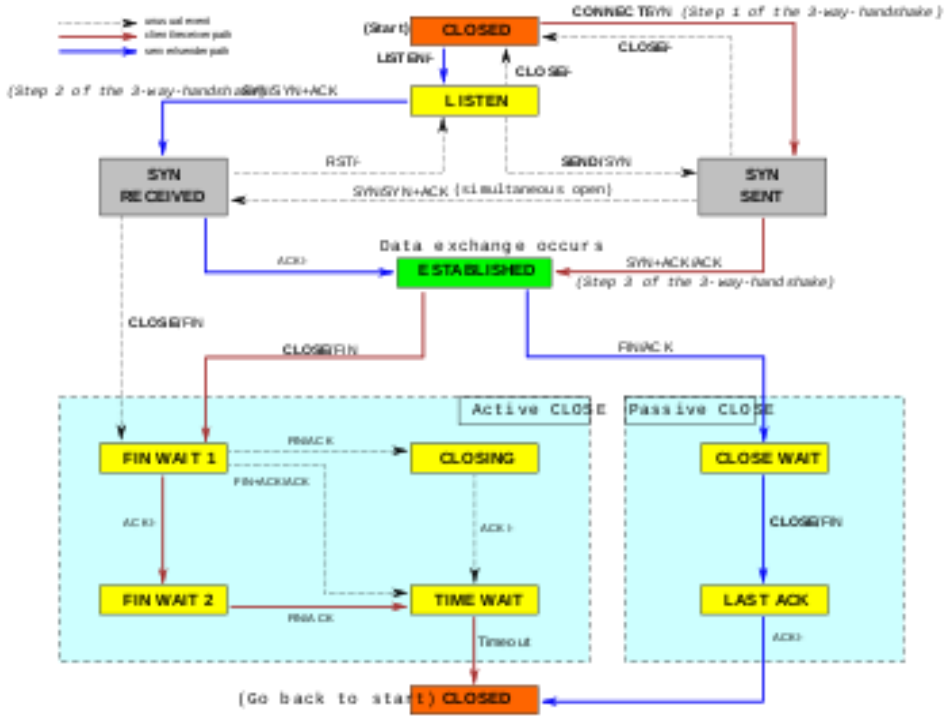
Without SACK



With SACK

TCP state diagram

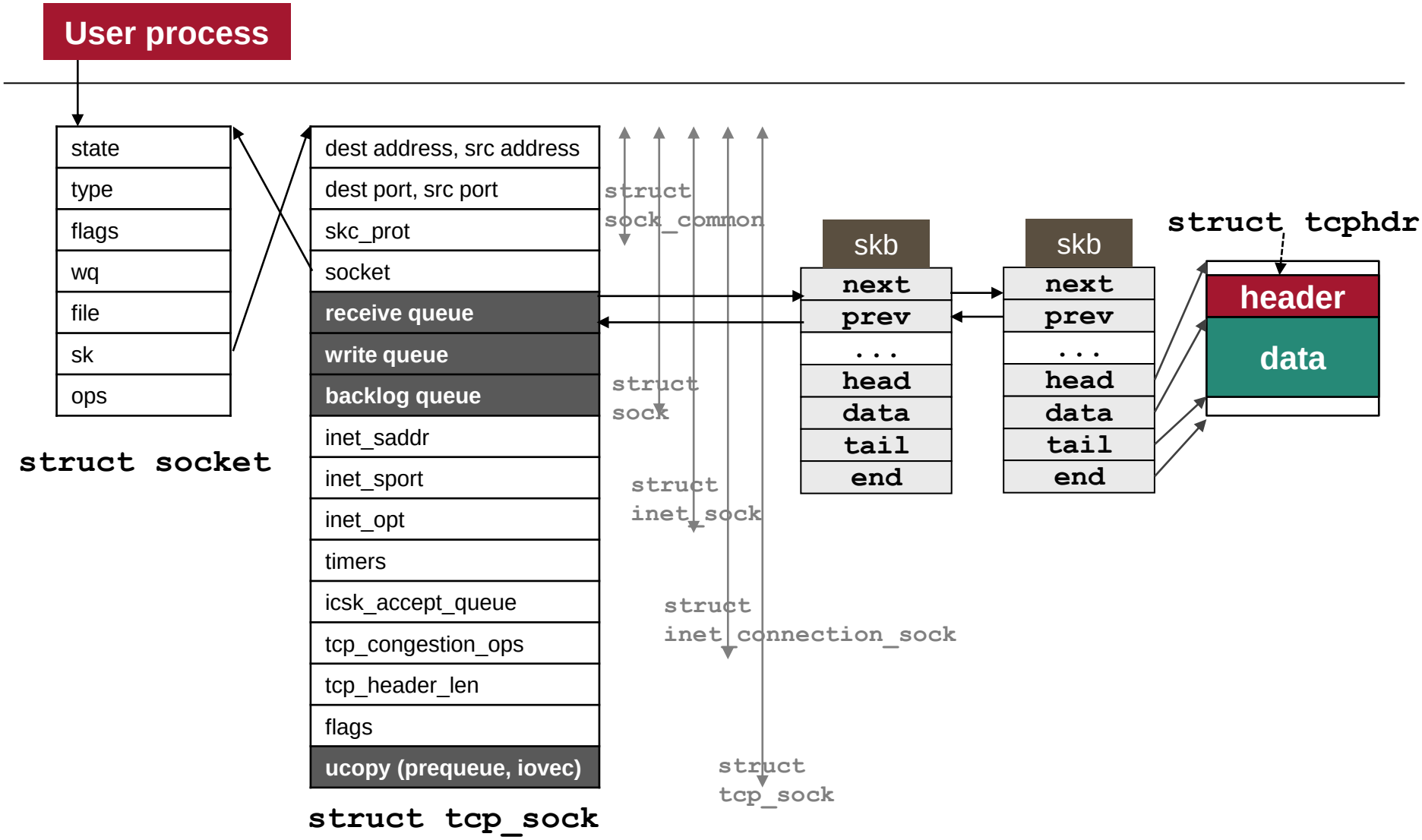
- Connection
  - handshake



Quic

- **Transport protocol based on UDP**
  - transport in http/3
- **Advantages**
  - Reduce latency
    - Connection establishment
    - Connect teardown
  - Better support for multi-streams
    - In http/2, packet loss causes delay for all streams
    - Improves web performance
- **Status**
  - IETF draft

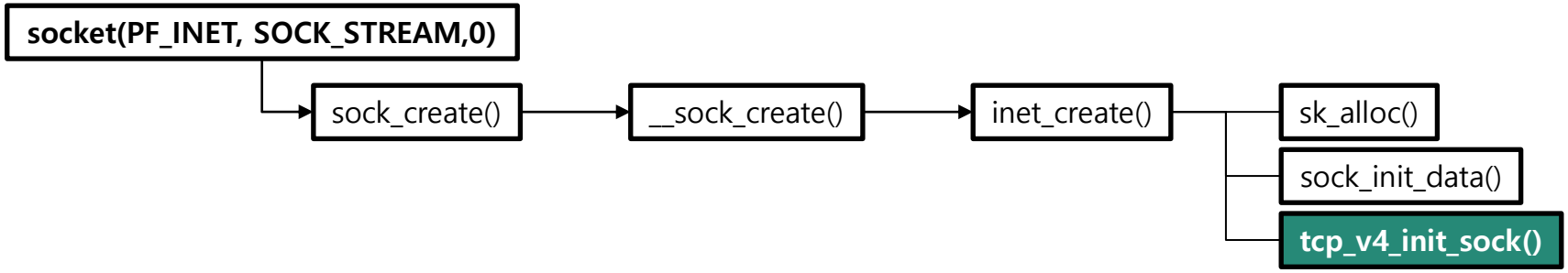
Overall data structures



Socket ↔ TCP

❖ Socket initialization

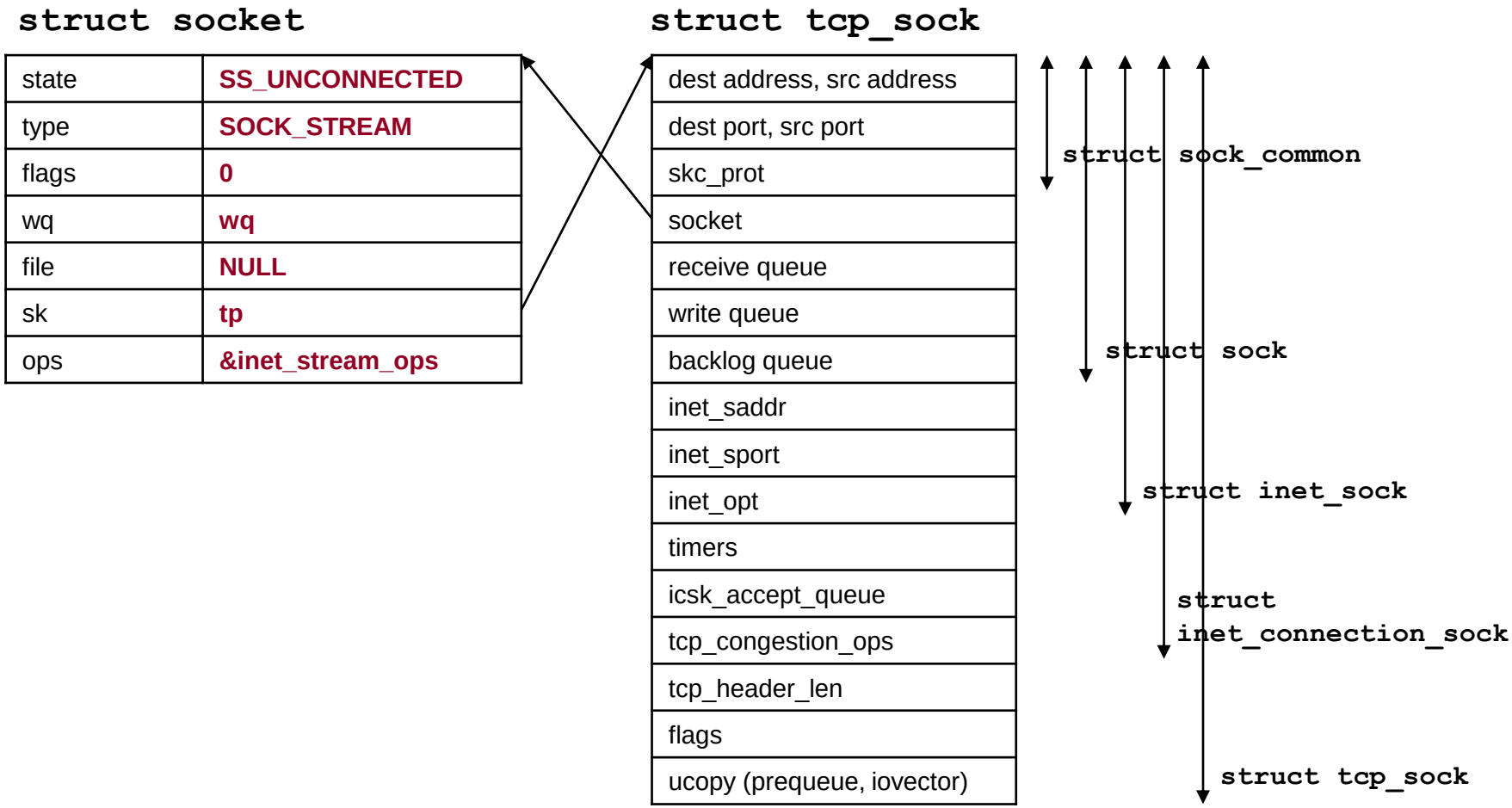
```
int socket (int family, int type, int protocol);
```



- During socket creation, **`inet_create()`** function is called
- From there the function **`tcp_v4_init_sock()`** will be called
  - Fields of structure **`tcp_sock`** are initialized to default values
  - Queues and TCP timers will be initialized

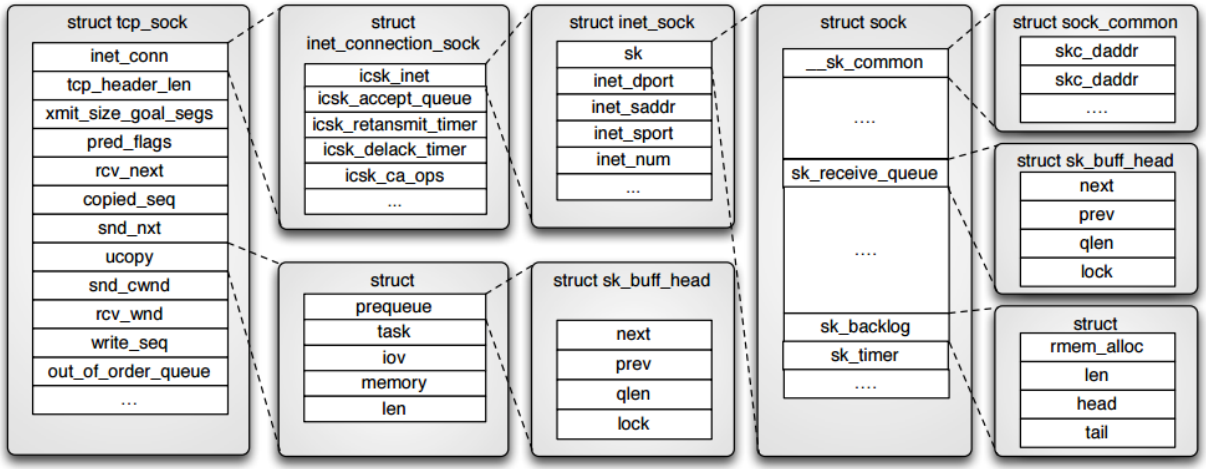
Socket ↔ TCP

❖ After initializing



Data structures for TCP

- `struct tcp_sock`
  - The core structure of TCP
  - Contain all the information and packet buffers for TCP connection
  - Inside tcp_sock there are a few other structures
    - Nested structures
    - Type-casting to different structures



Data structures for TCP

- **`struct inet_connection_sock`**
  - Structure one level down from the tcp_sock
  - Information about protocol congestion state and protocol timers
- **`struct inet_sock`**
  - Information about connection ports and IP addresses
- **`struct sock`**
  - Pointers to receive queue, backlog queue
  - Pointer to write queue for sent data, used with retransmission

Where include/uapi/linux/tcp.h

Data structures for TCP

- `struct tcphdr`
  - Represents the header of each packet

```
24 struct tcphdr {
25         __be16  source;
26         __be16  dest;
27         __be32  seq;
28         __be32  ack_seq;
29 #if defined(__LITTLE_ENDIAN_BITFIELD)
30         __u16   res1:4, doff:4, fin:1, syn:1, rst:1, psh:1,
                   ack:1, urg:1, ece:1, cwr:1;
40 #elif defined(__BIG_ENDIAN_BITFIELD)
41         __u16   doff:4, res1:4, cwr:1, ece:1, urg:1, ack:1,
                   psh:1, rst:1, syn:1, fin:1;
51 #else
52 #error  "Adjust your <asm/byteorder.h> defines"
53 #endif
54         __be16  window;
55         __sum16 check;
56         __be16  urg_ptr;
57 };
```

Sending segments in TCP

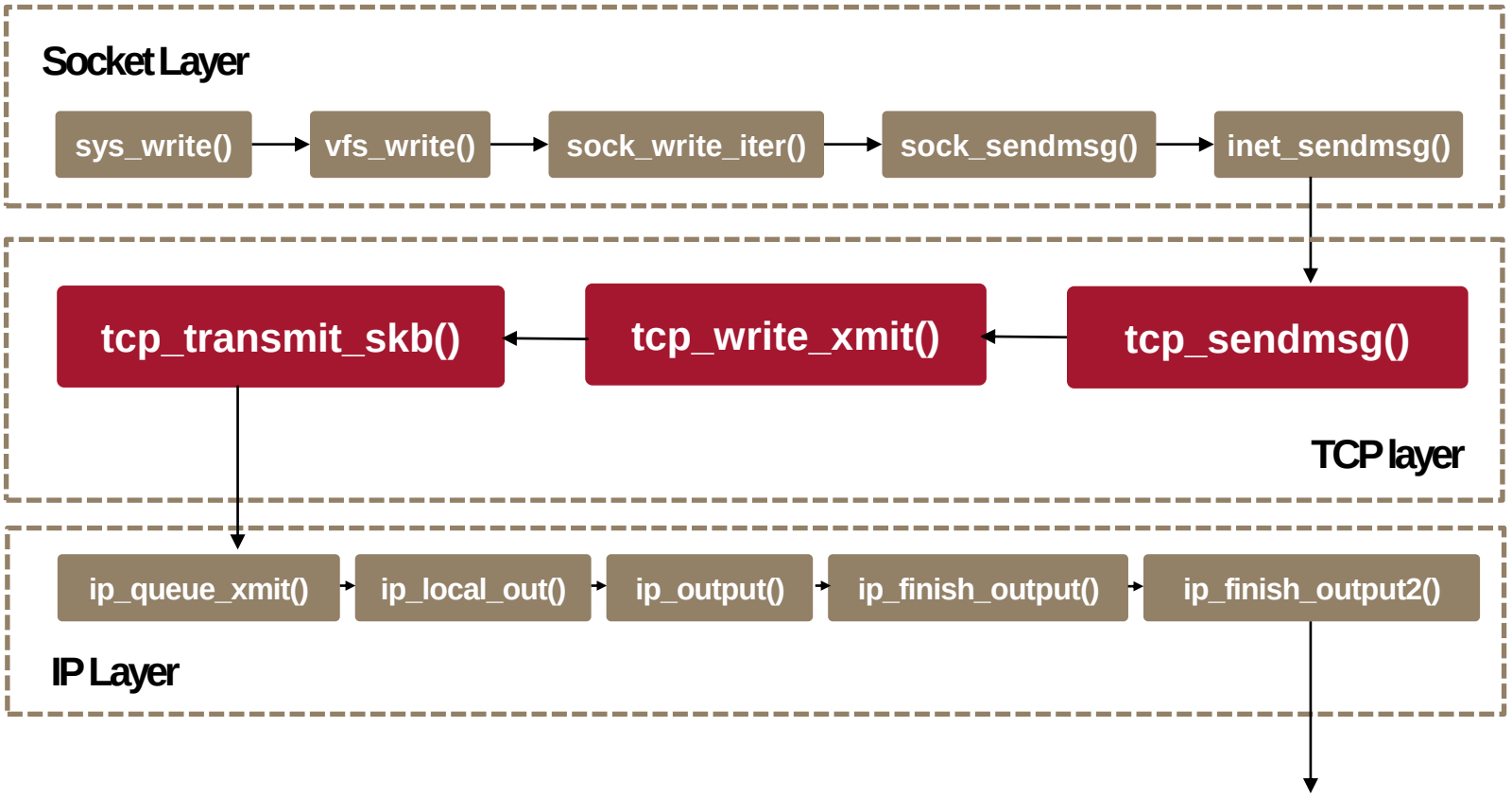
- **TCP Output**
  - tcp_sendmsg()
  - tcp_write_xmit()
  - tcp_transmit_skb()

Outgoing Data queue

- **Only one type of queue is used for outgoing data**
  - Implemented as double linked-list of **`struct sk_buff`**s
  - When user writes data to socket, segments will be put to write queue
  - When an acknowledgement arrives from receiver, the corresponding skb is removed

TCP Output

- After establishing a connection, User application calls system call for data transmission
- First function that will be called is `tcp_sendmsg()`



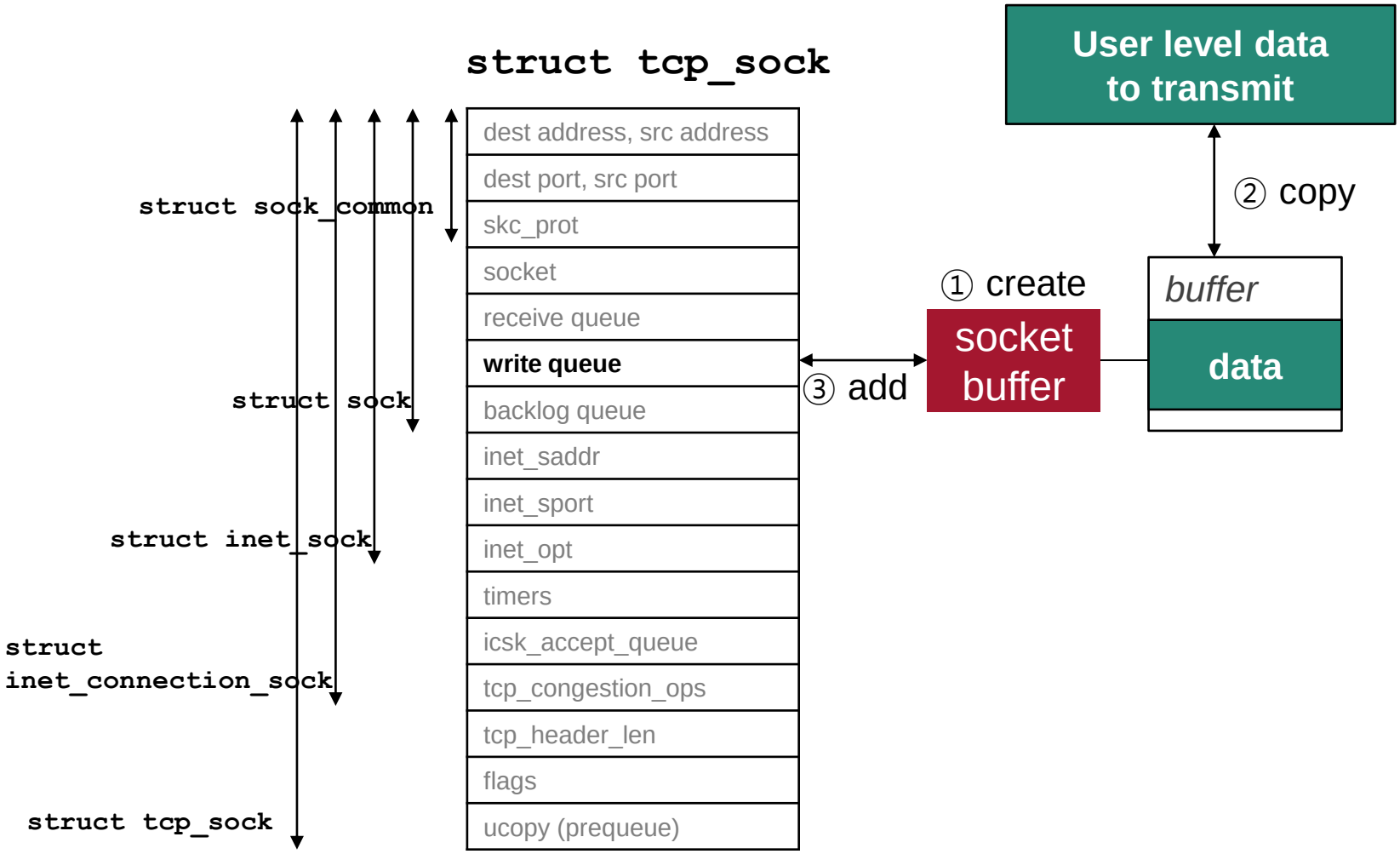
tcp_sendmsg() → tcp_write_xmit() → tcp_transmit_skb()

tcp_sendmsg()

- **Copies payload from the user space into the kernel space and sends it in the form of TCP segments**
  1. Creates **`sk_buff`** buffers of data size
  2. Copies the data from the user space (memcpy_from_msg)
  3. Pushes buffers to write queue

tcp_sendmsg() → tcp_write_xmit() → tcp_transmit_skb()

tcp_sendmsg()



tcp_write_xmit()

- **Segment is sent only when it is allowed to**
  - If congestion control (receiver window or Nagle's algorithm) prevents sending, the data will not go forward
- **Retransmission timers will be set**
- **To pass data to network layer, `tcp_transmit_skb()` is called**
- **After data send, congestion window will be validated**

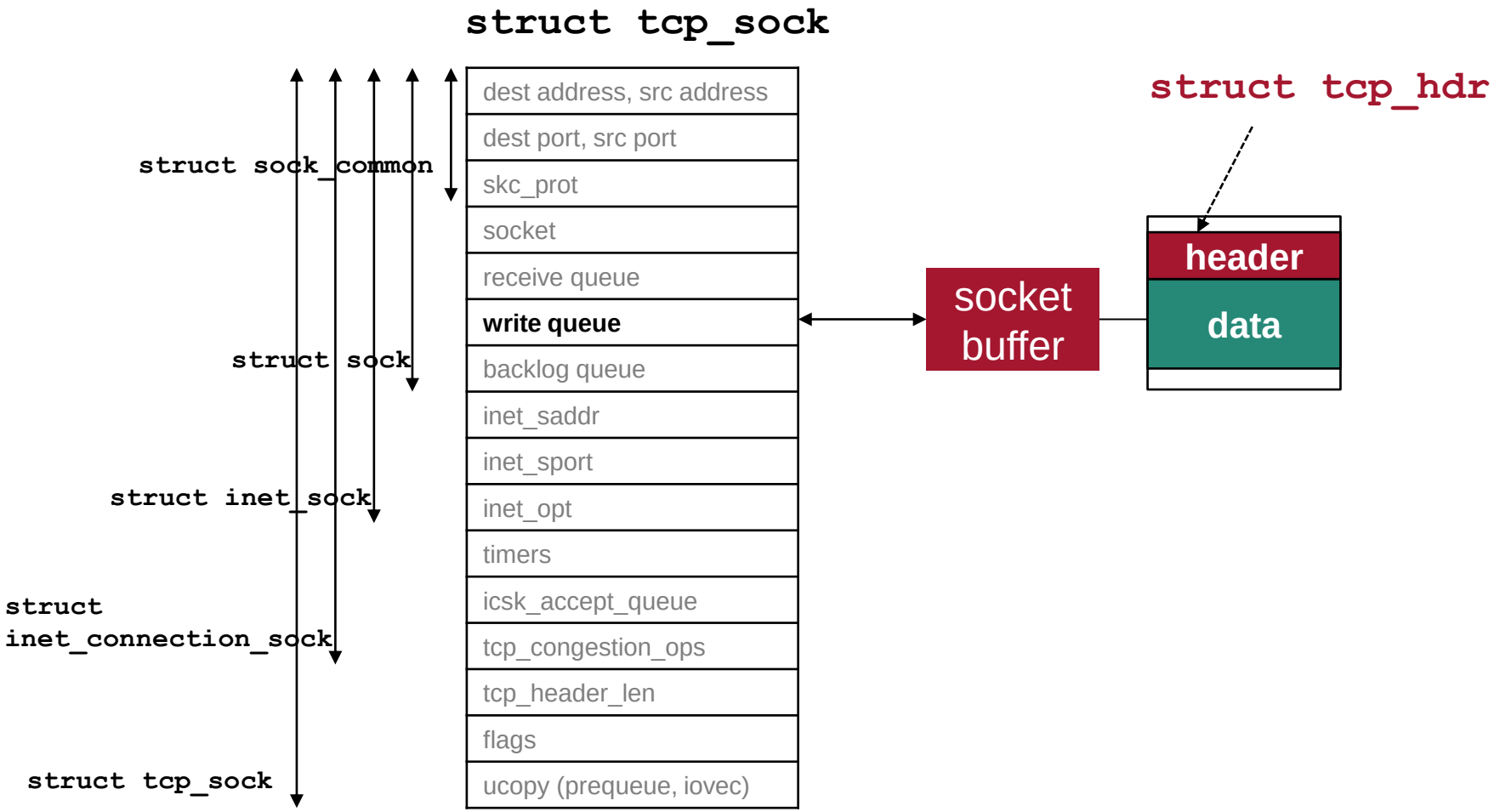
tcp_transmit_skb()

- **Crucial operation in the TCP output**
- **Executes the following tasks**
  - Check flags for timestamps, window scaling and SACK
  - Build **TCP header** and checksum
  - Increment TCP statistics
  - Call **`ip_queue_xmit()`**

tcp_sendmsg() → tcp_write_xmit() → tcp_transmit_skb()

tcp_transmit_skb()

❖ **Encapsulate TCP payload (Attach header)**



Receiving segments in TCP

- **Ways to process received packets**
- **Data queues**
- **TCP Input**
  - tcp_v4_rcv()
  - tcp_v4_do_rcv()
  - tcp_rcv_established()
  - tcp_data_queue()
  - tcp_recvmsg()

TCP Input

- **When TCP receives packets from the IP layer**
  - TCP processes the packets
  - And then, TCP passes the packet to the user process
    - If the current process is the one of the receiving socket's installer, packet is copied to user directly
    - If not, TCP puts the packets into the queues

- **Users use system calls to read data**

Data queues

- **Queues are used to avoid conflicts**
  - User reading data from the socket, socket will be marked as being in use
  - However, incoming segments must be saved somewhere else even when the socket is in use
  - Therefore, socket has several queues for incoming data
  - receive queue, pre-queue, backlog queue and out-of-order queue

Incoming Data queues

- **Receive queue**
  - Segment arrives and user is not waiting for it (General case)
  - Segment is processed immediately and copied to the rcv queue
  - Data will be copied to user's buffer when application reads data from the socket

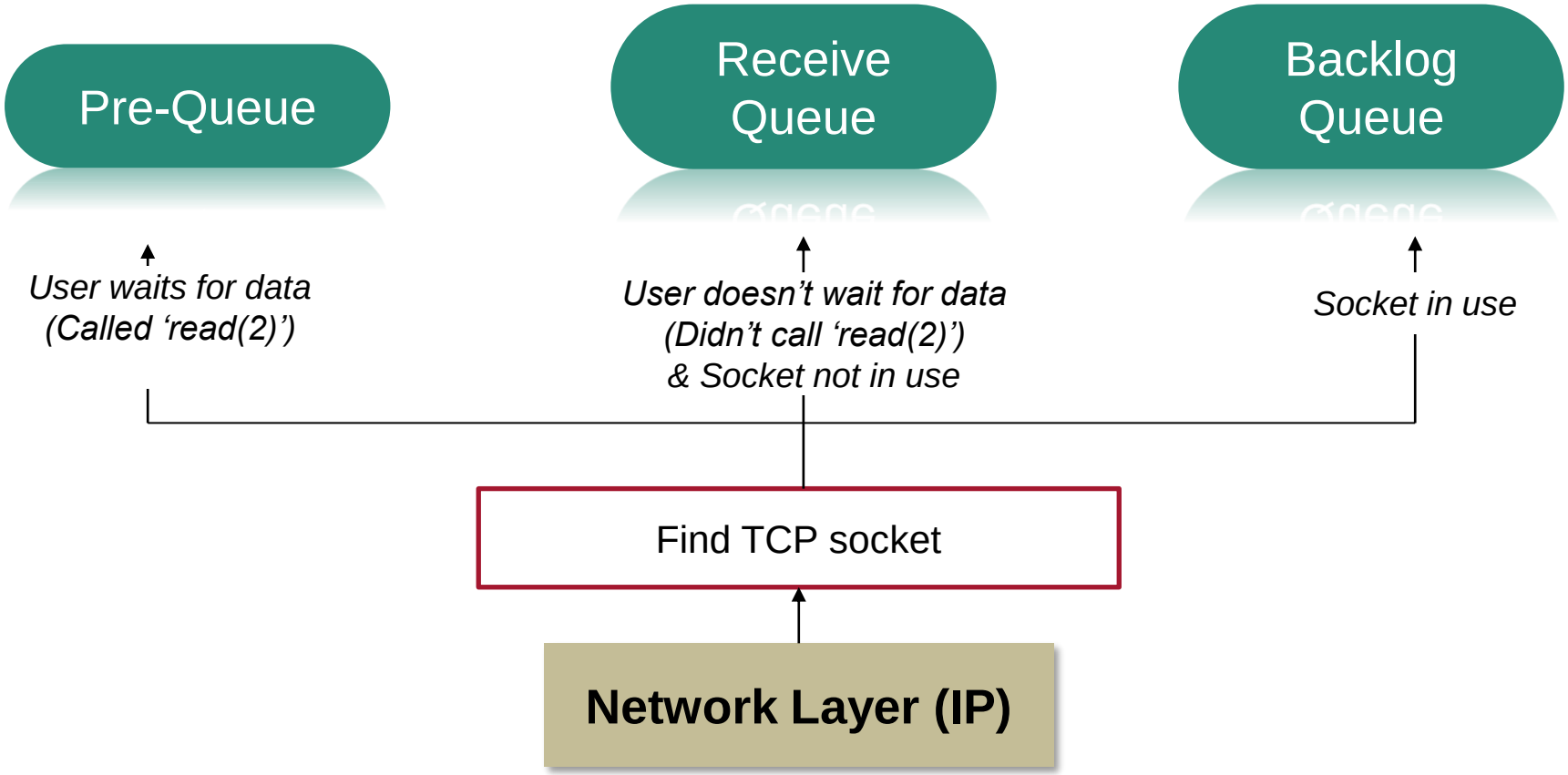
- **Pre-queue**
  - When user is using blocking I/O and the receive queue doesn't have as many bytes as requested, the socket will be marked as waiting for data
  - When new segment arrives, it will be put to pre-queue and waiting process will be awakened
  - Then the data will be handled and copied to user's buffer

Incoming Data queues

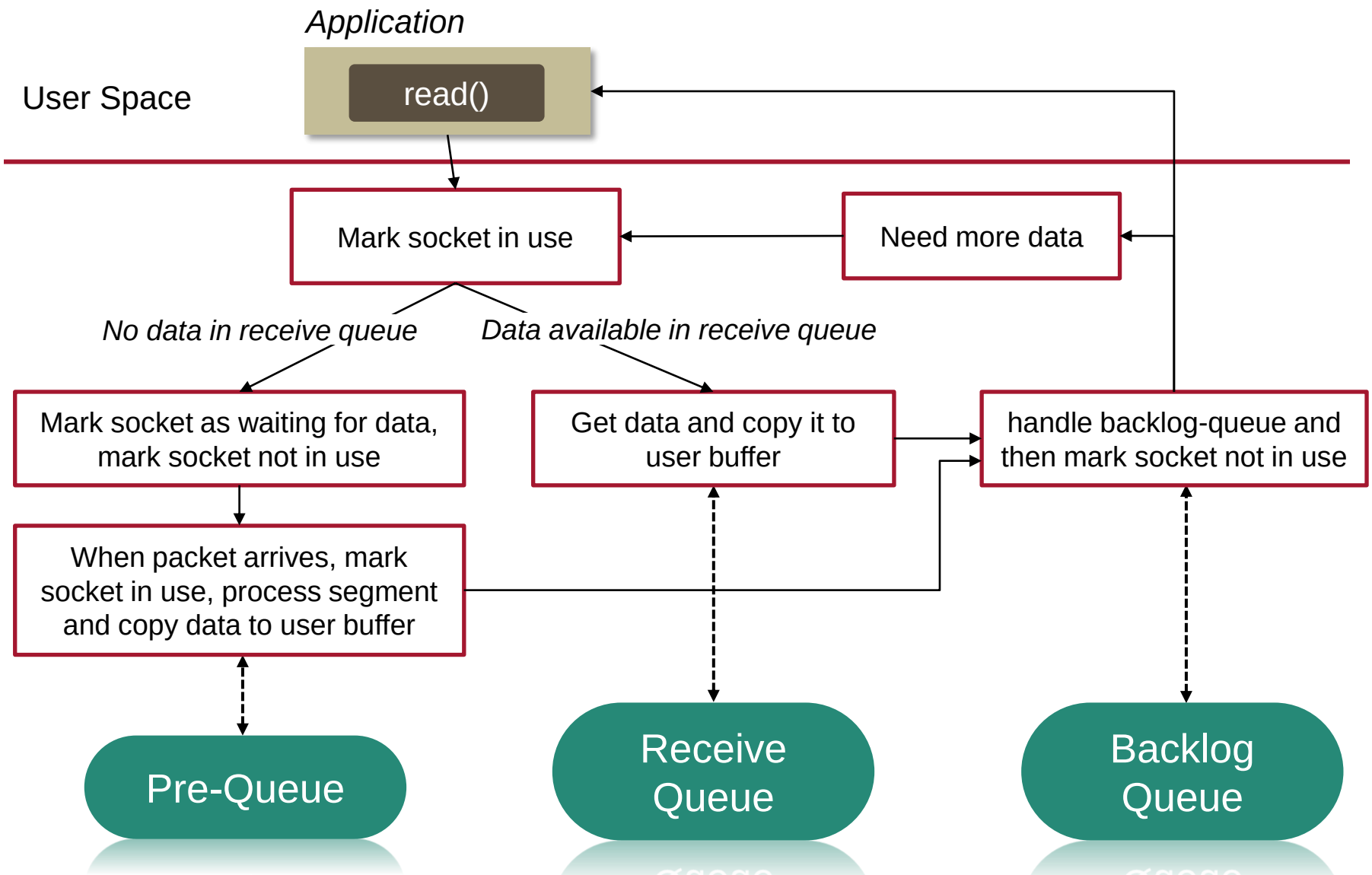
- **Backlog queue**
  - If user is handling segments (socket is marked as being in use) at the same time when we receive a new one, it will be put to the backlog queue
  - User context will handle the segment after it handles all earlier segments from other queues

- **Out-of-order queue**
  - Temporary storage for segments arriving out of order

Use of different incoming data queues: enqueue



Use of different incoming data queues: dequeue



Ways to process received packets

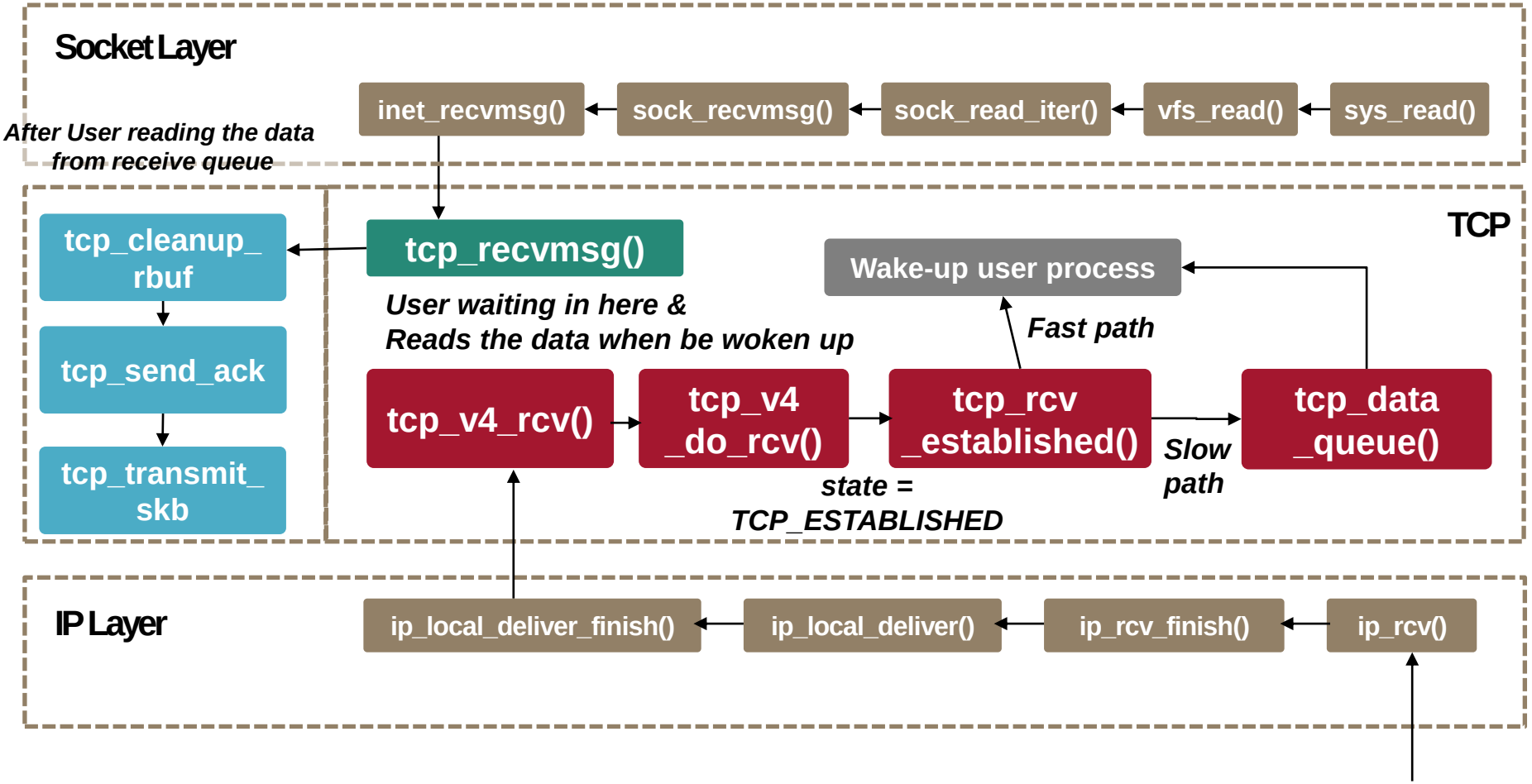
- **Slow path**
  - comprehensive processing route for segments
  - Handles special header flags and out-of-order segments
- **Fast path**
  - TCP optimization used to skip unnecessary packet handling when packet inspection is not needed
- **Header prediction is used to verify segment to fast path**
  - Compare certain bits in the header to check if the segment is valid for fast path
  - Header prediction ensures that there are no special conditions requiring additional processing

Fast and slow path 조건

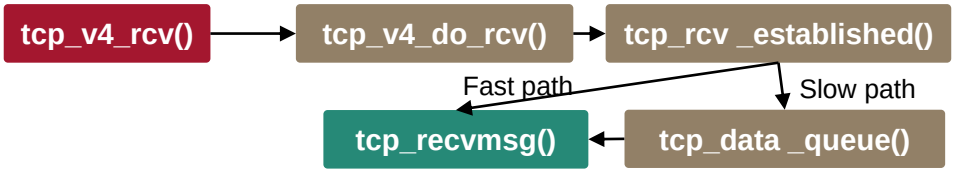
- **Packets are processed in fast path if**
  - The segment received is a pure ACK segment for the data sent last
  - The segment received is the data expected (Packet's SN is the same with expected number)

- **Packets are processed in slow path if**
  - SYN, URG, FIN, RST flag is set (detected in Header Prediction)
  - The sequence number of the segment is inconsistent with `tcp→rcv_nxt` (Out of order)
  - The communication is two-way (ACK + Data)
  - The segment contains TCP options

TCP Input

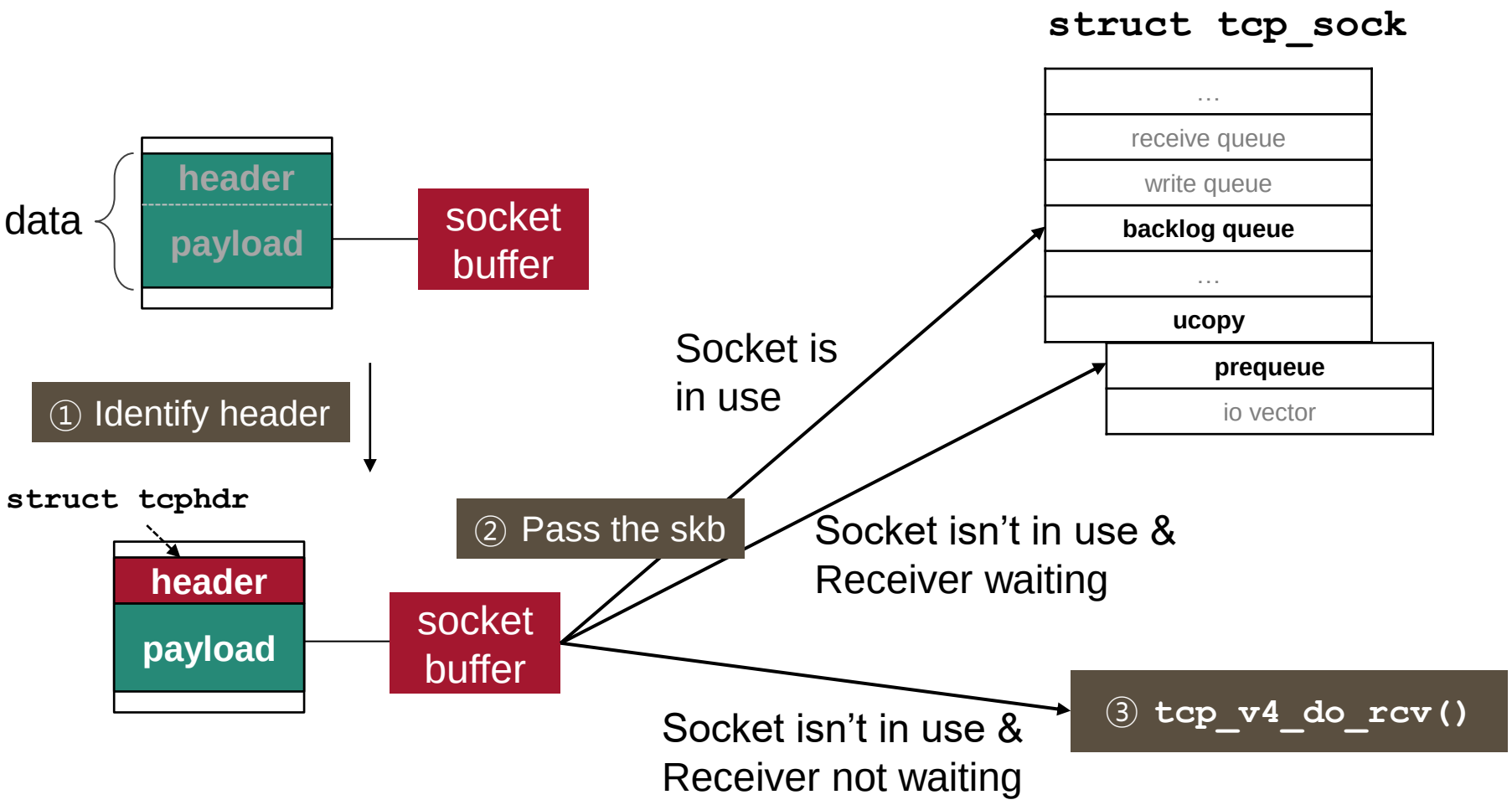

Socket Layer
inet_recvmsg()
sock_recvmsg()
sock_read_iter()
vfs_read()
sys_read()
After User reading the data from receive queue
TCP
tcp_cleanup_rbuf
tcp_send_ack
tcp_transmit_skb
tcp_recvmsg()
User waiting in here &
Reads the data when be woken up
Wake-up user process
Fast path
tcp_v4_rcv()
tcp_v4_do_rcv()
tcp_rcv_established()
tcp_data_queue()
Slow path
state = TCP_ESTABLISHED
IP Layer
ip_local_deliver_finish()
ip_local_deliver()
ip_rcv_finish()
ip_rcv()

tcp_v4_rcv()



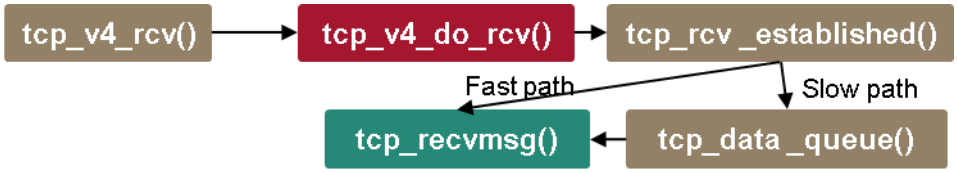
- **Checks if packet is for us**
  - Divide header and data region in socket buffer (①)
  - Check checksum
- **Finds the matching TCP socket using IP address and ports to pass the skb (②)**
  - If the socket is now being handled, packet is left on the backlog queue
  - If the socket is not owned by user
    - Check if the user context is waiting for the data
      - If yes, packet is left on the pre-queue and wakes the waiting context
      - If not, we do further processing (**`tcp_v4_do_rcv()`**)

tcp_v4_rcv()


struct tcp_sock
…
receive queue
write queue
backlog queue
…
ucopy
prequeue
io vector
data
header
payload
socket buffer
① Identify header
struct tcphdr
header
payload
socket buffer
② Pass the skb
Socket is in use
Socket isn't in use & Receiver waiting
Socket isn't in use & Receiver not waiting
③ tcp_v4_do_rcv()

tcp_v4_do_rcv()

- **Calls function depending on the TCP state of the connection**
  - If the connection is established (stateTCP_ESTABLISHED), it calls **tcp_rcv_established()**
  - All other states are processed by **tcp_rcv_state_process()**

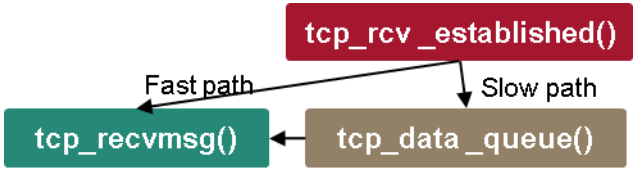


tcp_rcv_established()

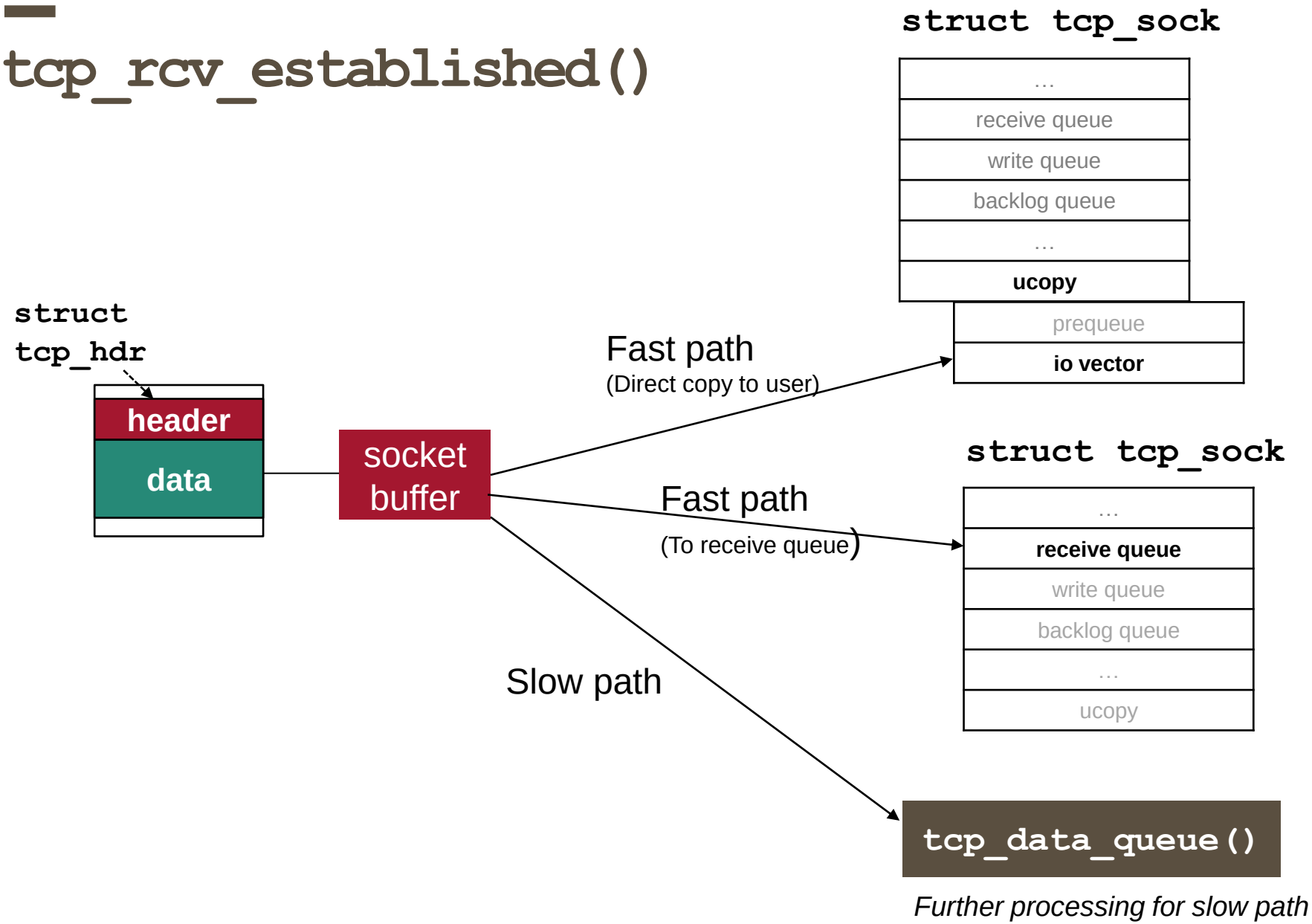
- **Dispatches packets to fast path or slow path based on header prediction**

- **If the packet is for fast path**
  - If the current process is the one of the receiving socket's installer, packet is copied to user directly
  - If not, put to the receive queue

- **If the packet is for slow path**
  - Call `tcp_data_queue()`

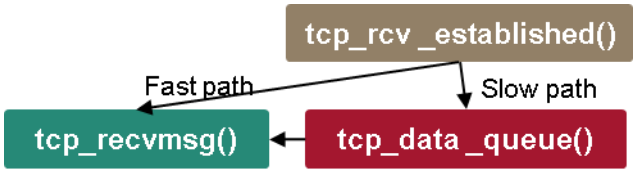


tcp_rcv_established()



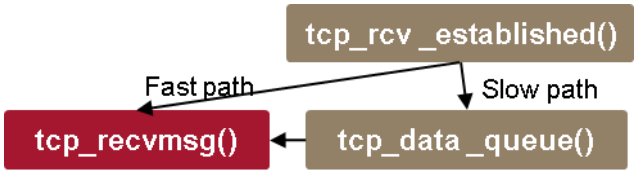
tcp_data_queue()

- **tcp_data_queue()**
  - Process the tcp segment options and flags
  - If the packet did not arrive in order, it puts it in the out-of-order queue
    - If a gap in the queue is filled, send an ACK immediately
  - Segment is copied to user or put to the receive queue



tcp_recvmsg()

- **When user calls `sys_read()`, received packets in the receive queue are copied to user**
- **If there aren't sufficient packets as requested, process sleeps waiting for data to be arrived**
  - If packet arrives, the waiting process is awakened
  - The packets in the pre-queue, receive queue, backlog queues are copied to user



Summary of TCP Input