Computer Science Study

Operating System

Subject 1. Network Overview
Subject 2. Socket Abstraction
Subject 3. TCP in Linux
Subject 4. IP in Linux
Subject 5. Data link in Linux
Subject 6. Network adapter

홍승택

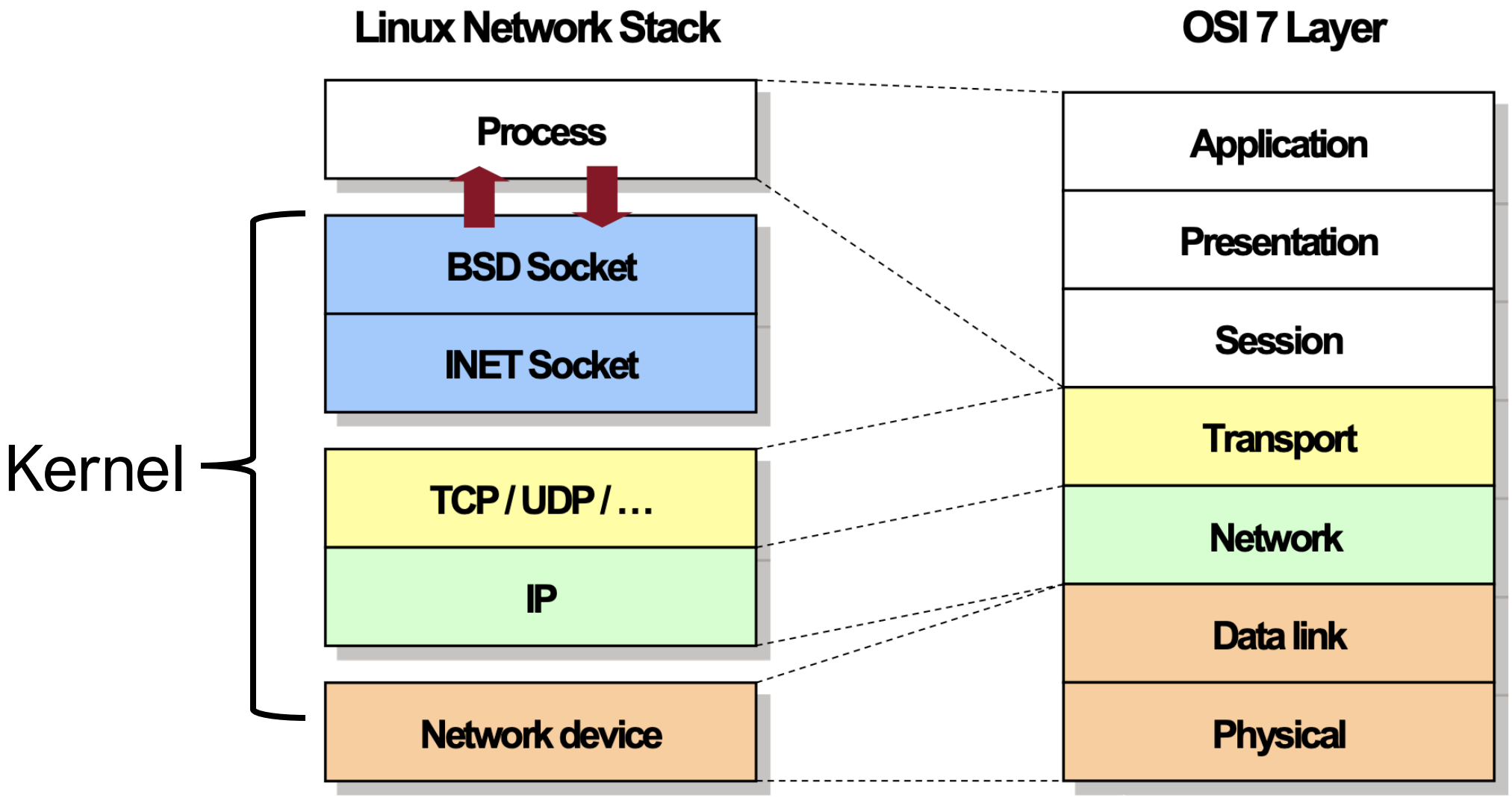
made by Seungtaek

2024. 09. 30. 19시

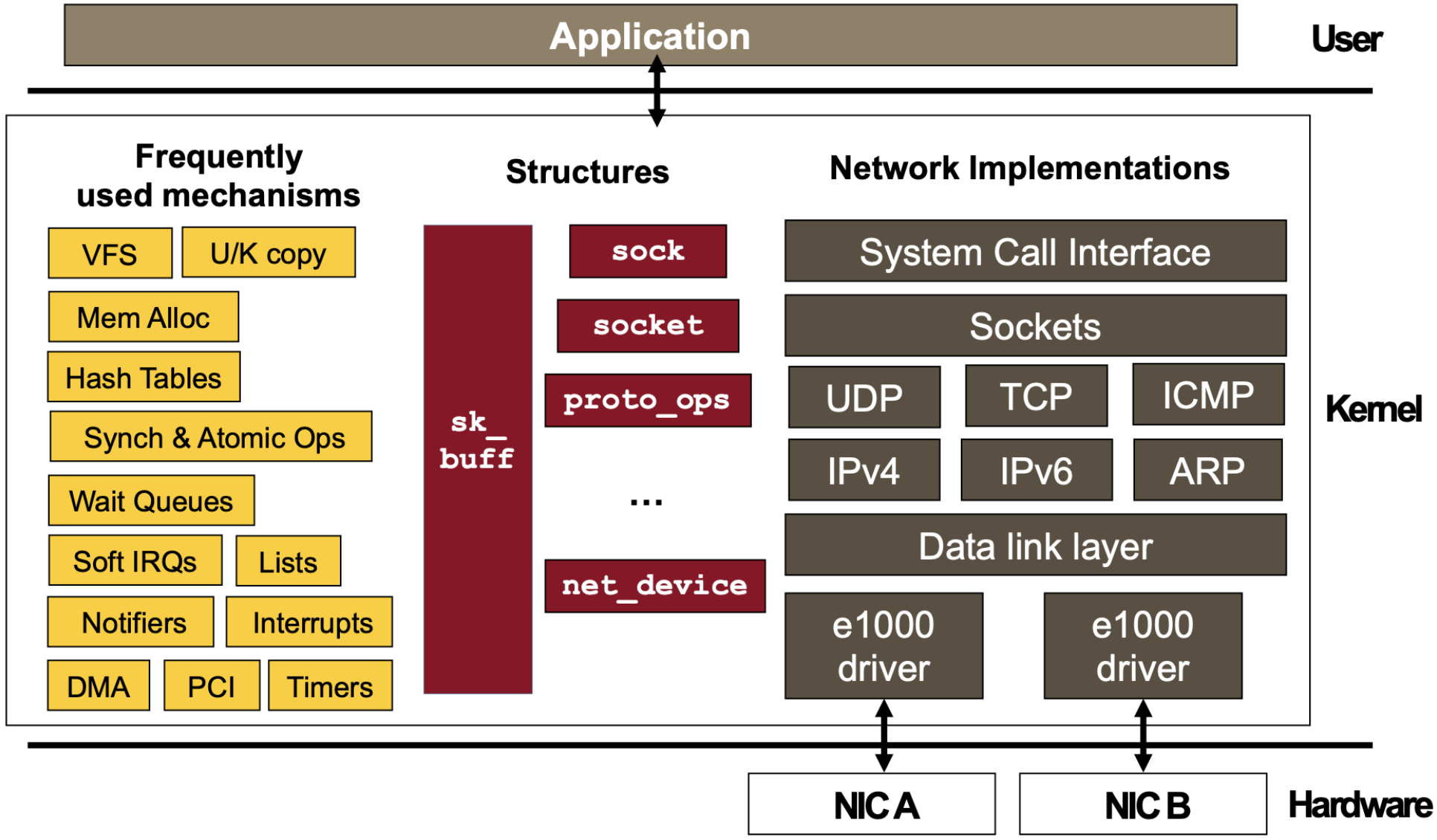
subject 1.

Network Overview

Protocol Stack in Linux Kernel



Kernel Networking



subject 2.

Socket Abstraction

Socket

- An **end-point** for Internet network “connection”
- Socket is defined by the combination of **network address** and **port identifier**

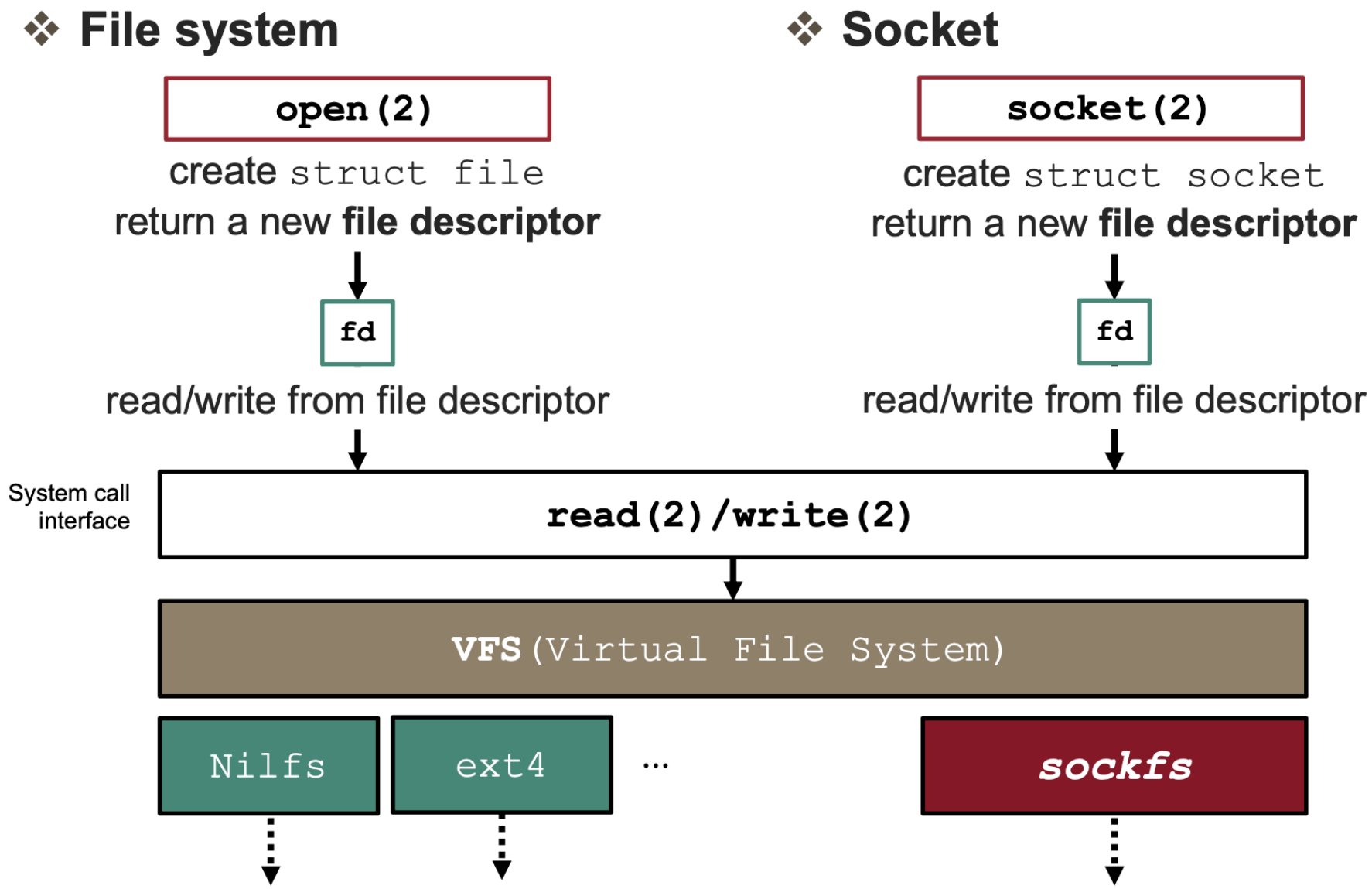
Use file descriptor

- socket(2) returns *fd*
- This *fd* is used to send or receive packets

Does not have open(2)

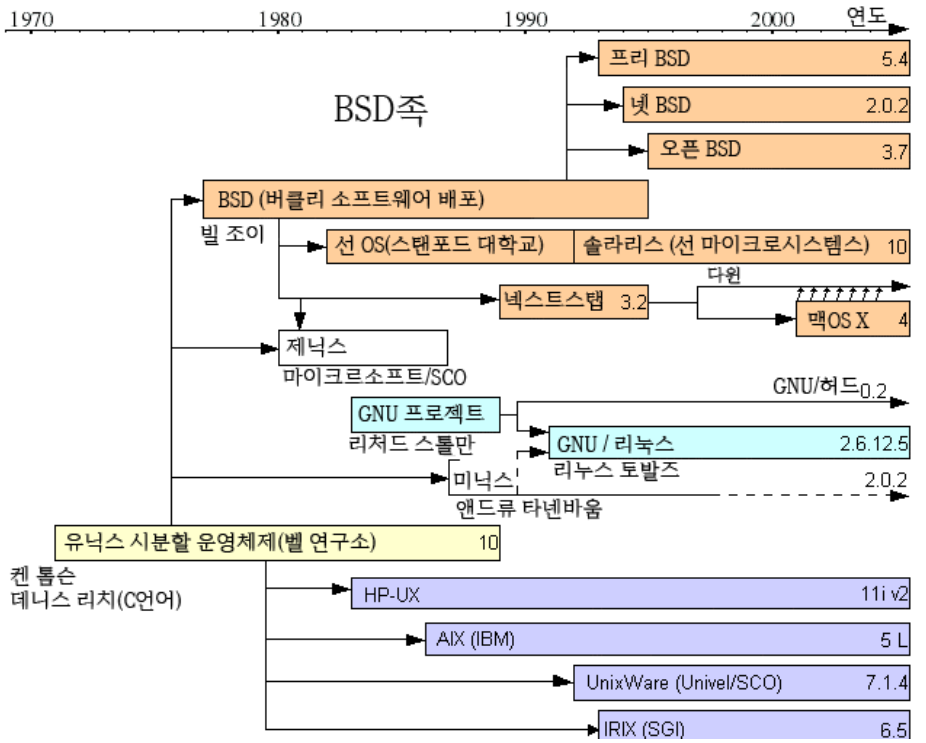
- Instead, connect() and listen()
- Since “connection” is transient and not shared

Socket as a File



Networking with socket

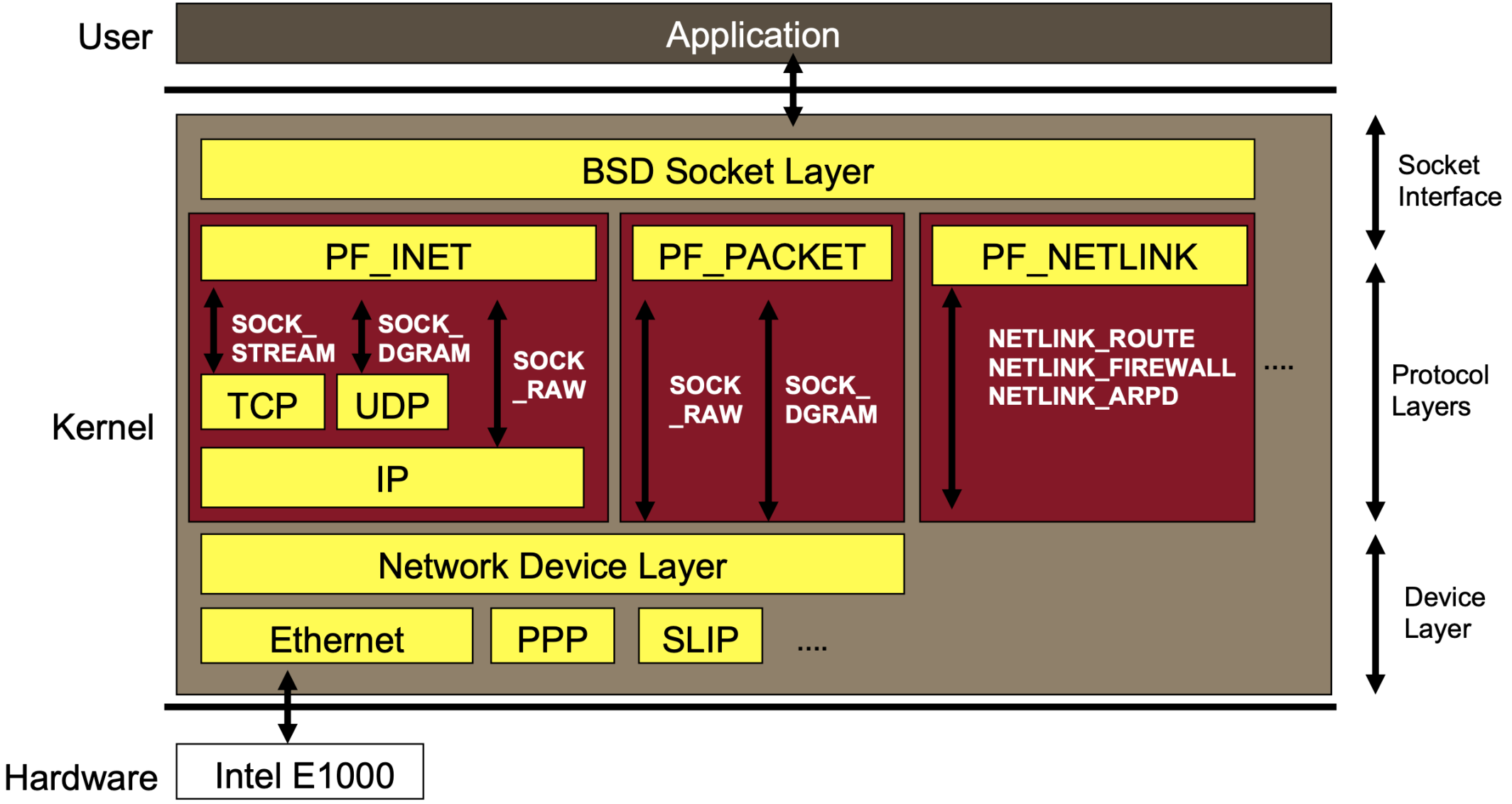
BSD(Berkeley Software Distribution)



- **BSD Socket Layer :** Standard UNIX Socket API
  - Originally developed by UC Berkeley in 80's
- Common interface for network programming
  - Abstraction of all the transport protocols
  - Able to make user apps without specific knowledge about each protocol
- A socket is exposed as a file descriptor to user level applications

https://youtu.be/1Z-VTp4H2Wk?t=97

Networking with socket

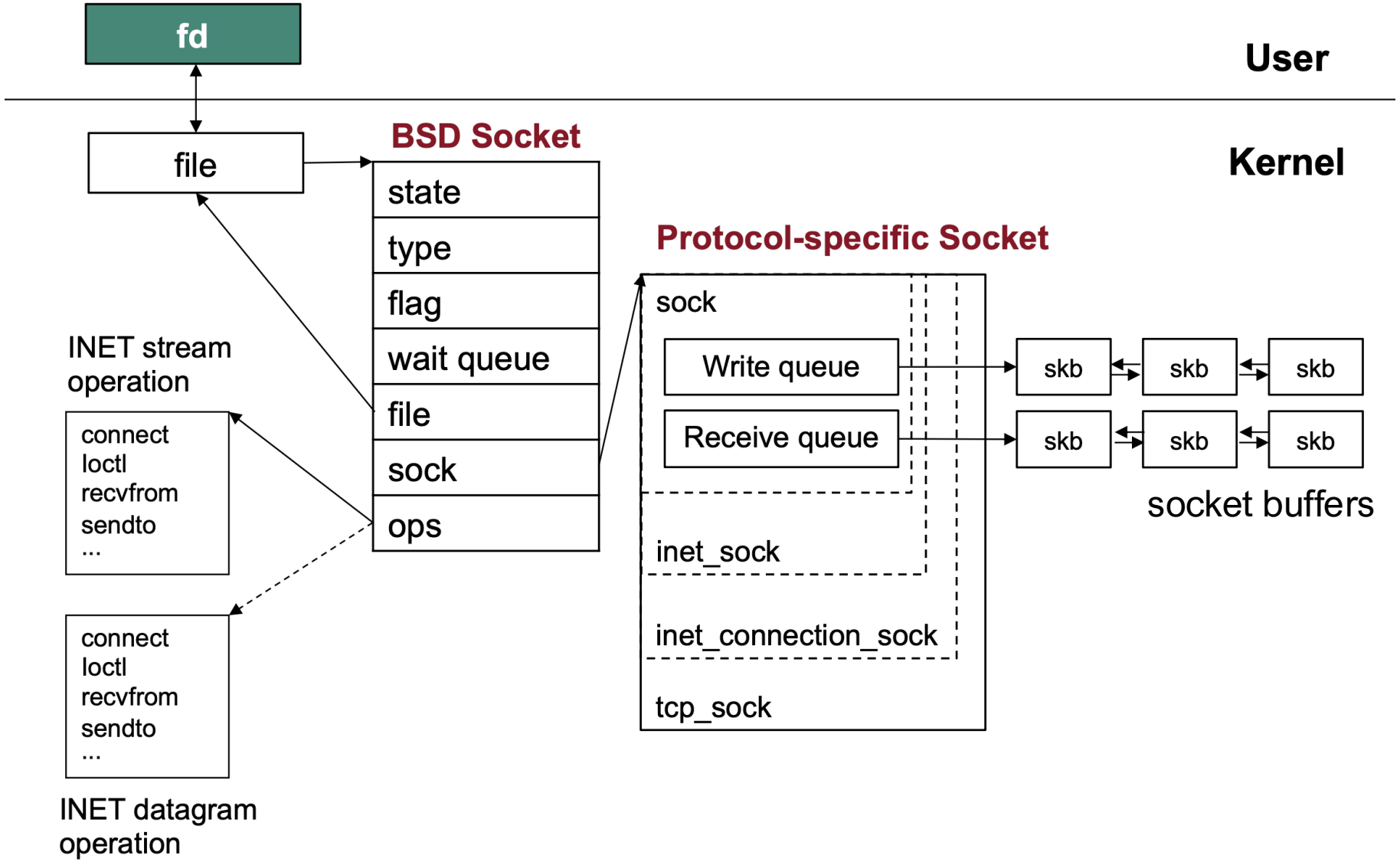


Kernel data structures for socket

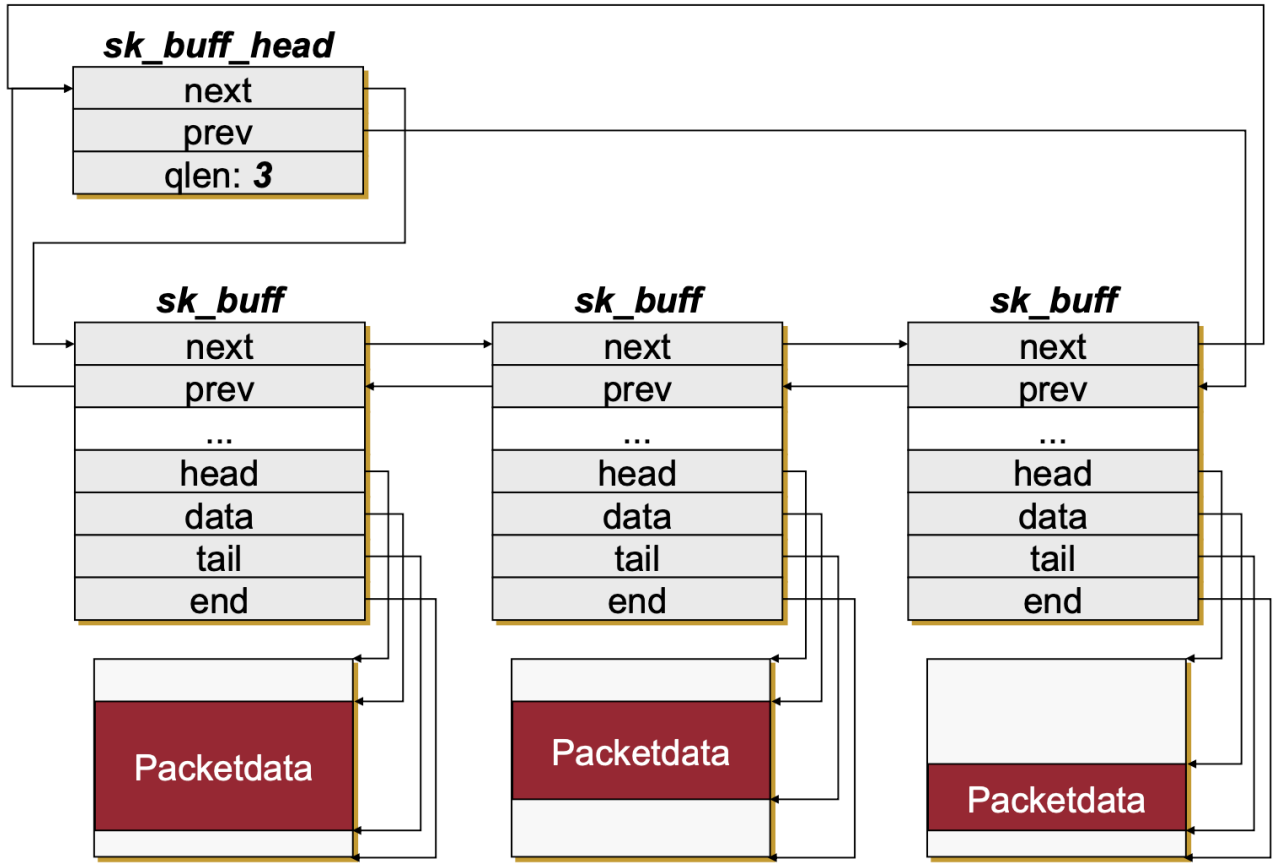
- **BSD Socket (struct socket)**
  - Uniform interface to send or receive data for an application
  - **It's identified by 5 tuples**
    - Source Address, Source Port, Destination Address, Destination Port, Protocol

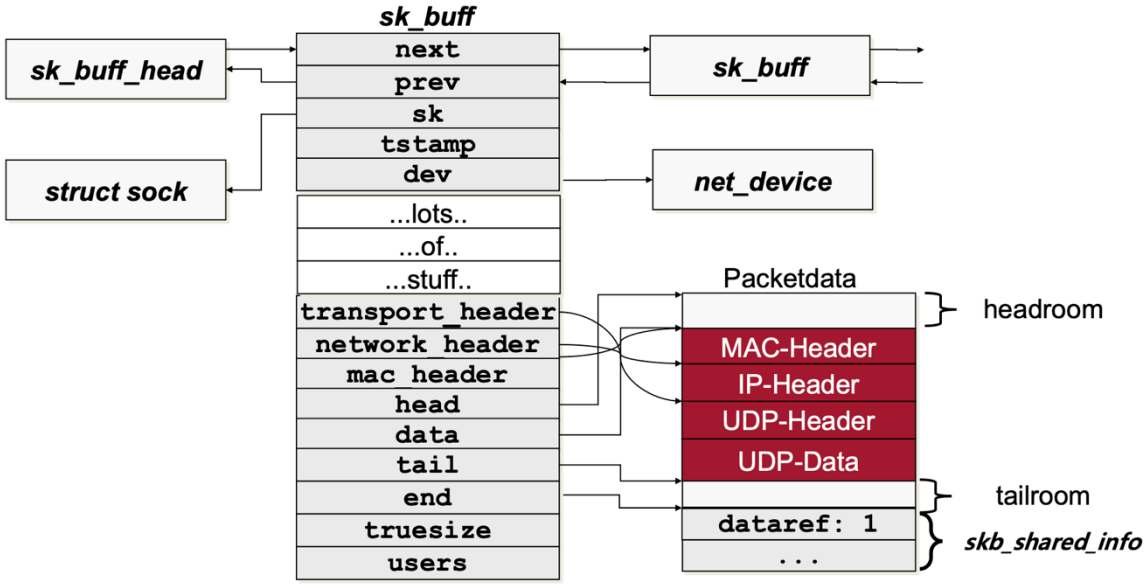
- **Protocol-specific Socket (struct sock)**
  - Data common to each protocol family (e.g., PF_INET)

Overall structure



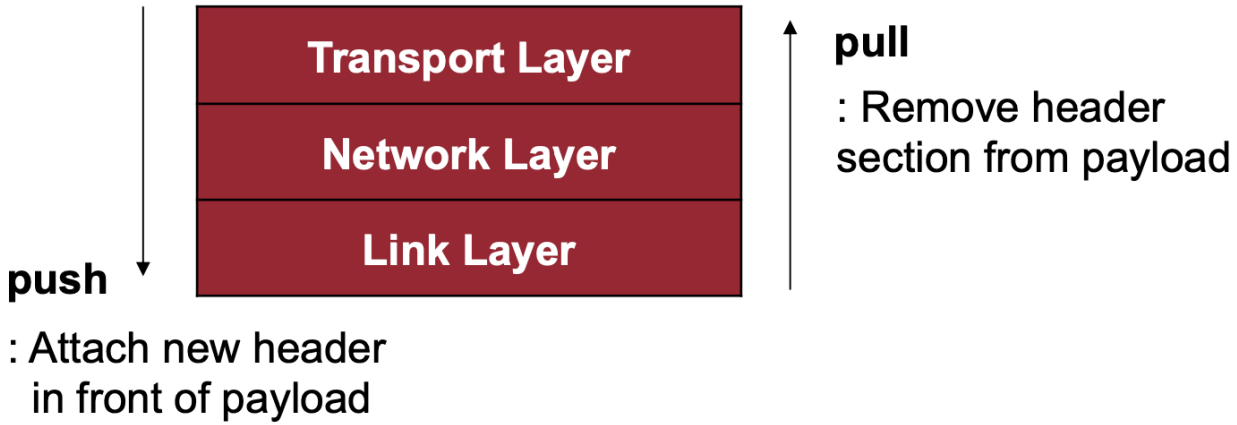
Structure of sk_buff



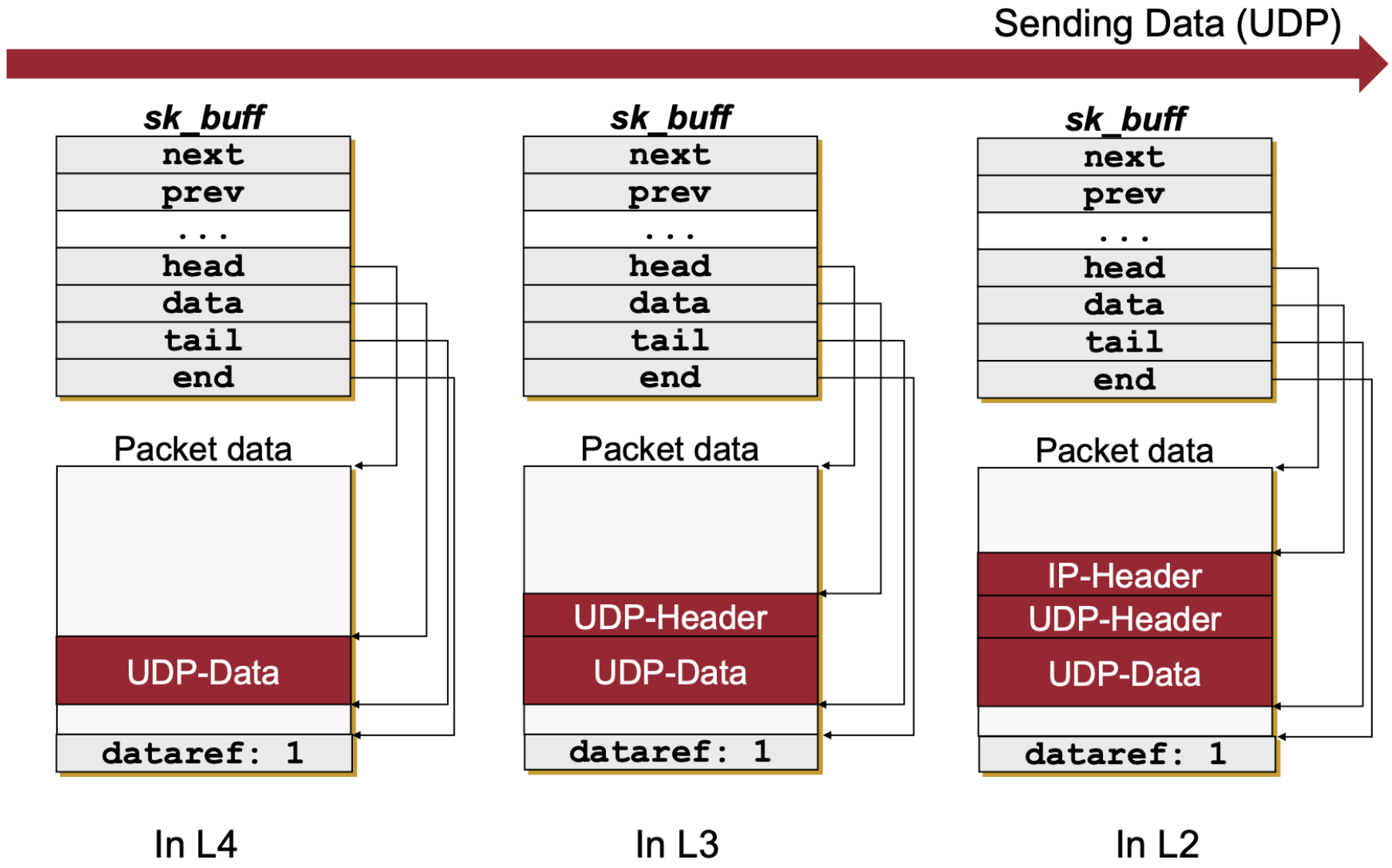


How socket buffers work

- **Sending data**
  - sk_buff에 데이터 copy (user space → kernel)
  - During the travel across the protocol stack, packet header of each layer is inserted in front of the payload
- **Receiving data**
  - When a packet arrives, NIC generates interrupt
  - dev_alloc_skb() allocates sk_buff structure during the interrupt handling
- **The payload is copied twice**
  - user address space ↔ kernel address space
  - kernel address space ↔ network adapter (via DMA)



Packet data change



Avoid memory copy between layers

Practice : Server socket

Github : https://github.com/CS-Computer-Science-Study/Operating-System/tree/main/practice/ServerSocket

```
// Compile (.java → .class)
> javac {파일명}.java

// Execution
> java {클래스 파일명}
```

Client

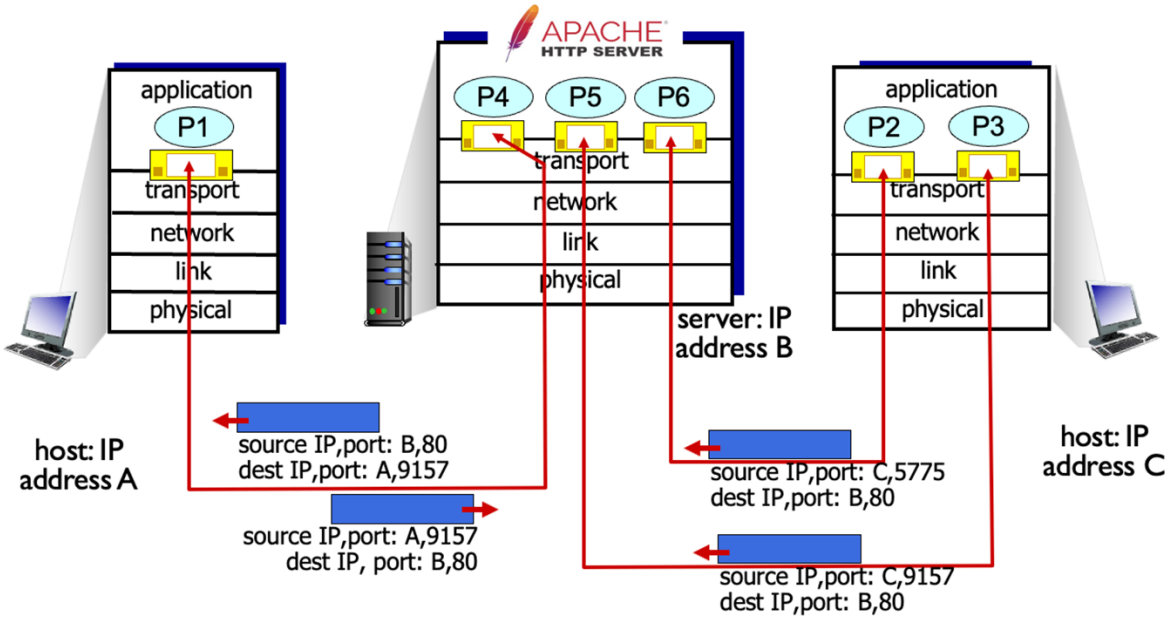
```
javac ClientSocketEx.java
hongseungtaeg ~/Desktop/bookstudy/ope
java ClientSocketEx
서버에 접속되었습니다 : /127.0.0.1:6543
hello!
서버로부터 받은 메시지: hello!
bye
서버로부터 받은 메시지: bye
연결을 종료합니다.
```

Server

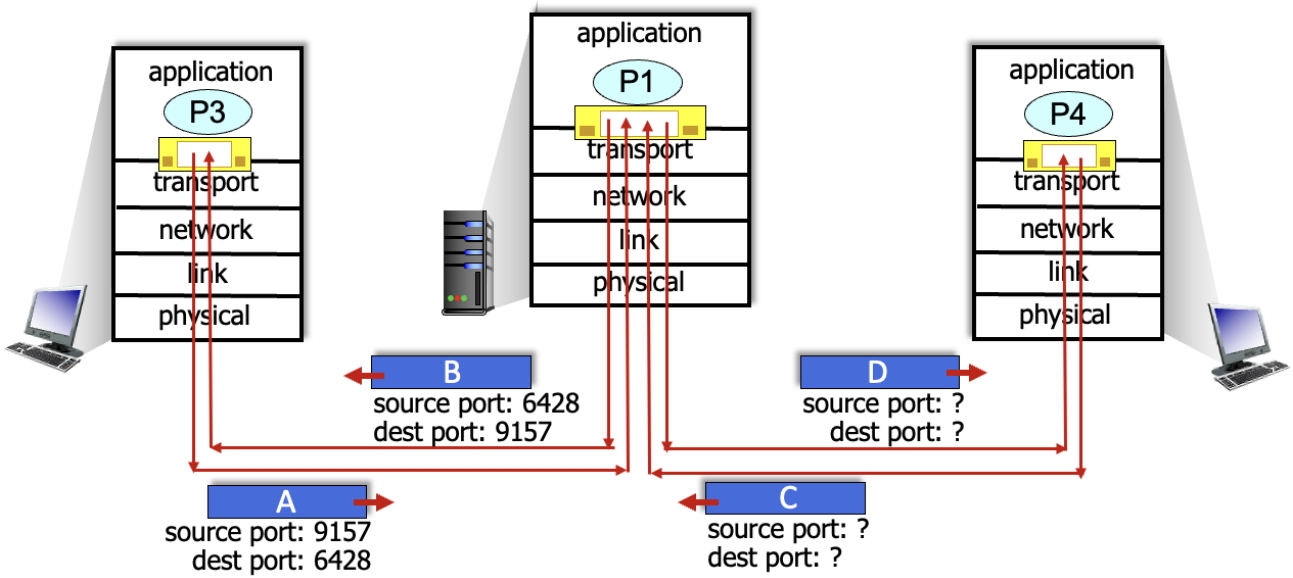
```
클라이언트가 연결되었습니다.
클라이언트가 연결되었습니다.
[/127.0.0.1:58102 → /127.0.0.1:6543] client1
[/127.0.0.1:58098 → /127.0.0.1:6543] client2
클라이언트가 연결되었습니다.
[/127.0.0.1:58108 → /127.0.0.1:6543] client3
```

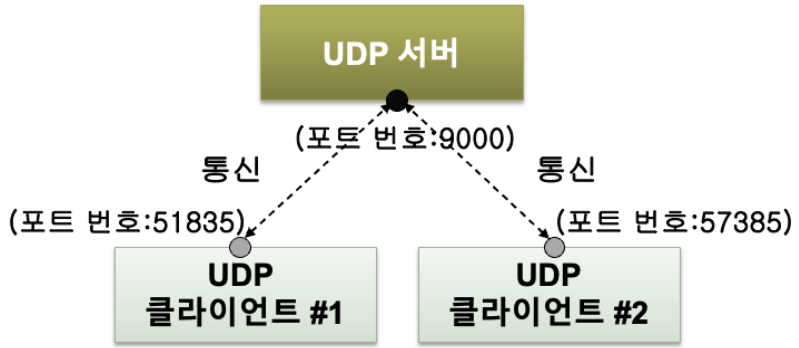
Practice : UDP socket

TCP : connection-oriented



UDP : connectionless





Practice : UDP socket

Github : https://github.com/CS-Computer-Science-Study/Operating-System/tree/main/practice/UDPSocket

```
// Compile (.java → .class)
> javac {파일명}.java

// Execution
> java {클래스 파일명}
```

Client

```
java UDPClient
client1
서버로부터 받은 메시지: client1
bye
서버로부터 받은 메시지: bye
연결을 종료합니다.
```

Server

```
java UDPServer
[/127.0.0.1:58824] client1
[/127.0.0.1:50962] client2
[/127.0.0.1:58824] bye
클라이언트 연결 종료.
```

Practice : Socket identifier

TCP Client

```
java ClientSocketEx
TCPClient
서버로부터 받은 메시지: TCPClient
```

TCP Server

```
java ServerSocketEx
클라이언트가 연결되었습니다.
[/127.0.0.1:9876 -> /127.0.0.1:6543] TCPClient
```

UDP Client

```
java UDPClient
UDPClient
서버로부터 받은 메시지: UDPClient
```

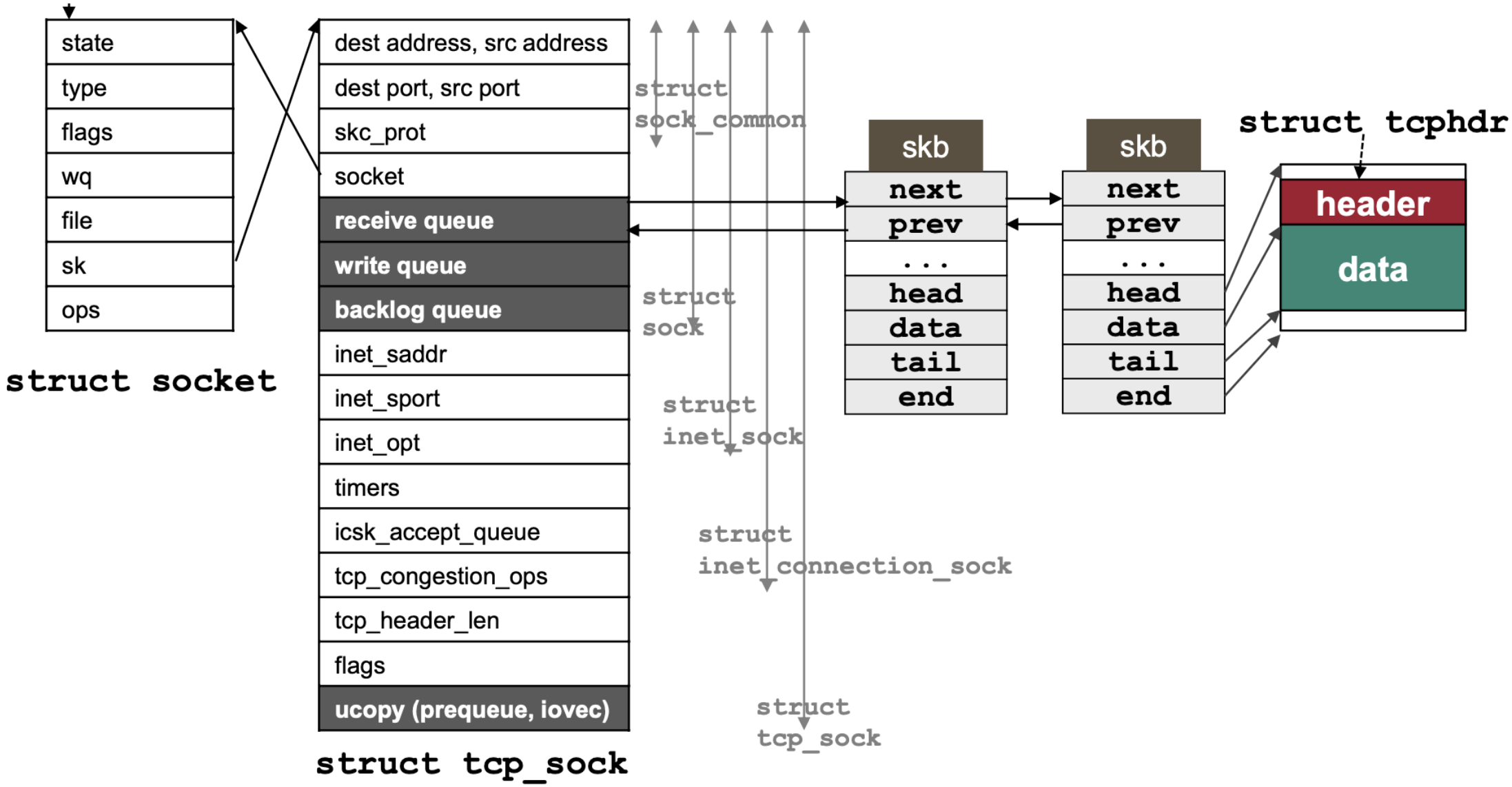
UDP Server

```
java UDPServer
[/127.0.0.1:9876 -> :6543] UDPClient
```

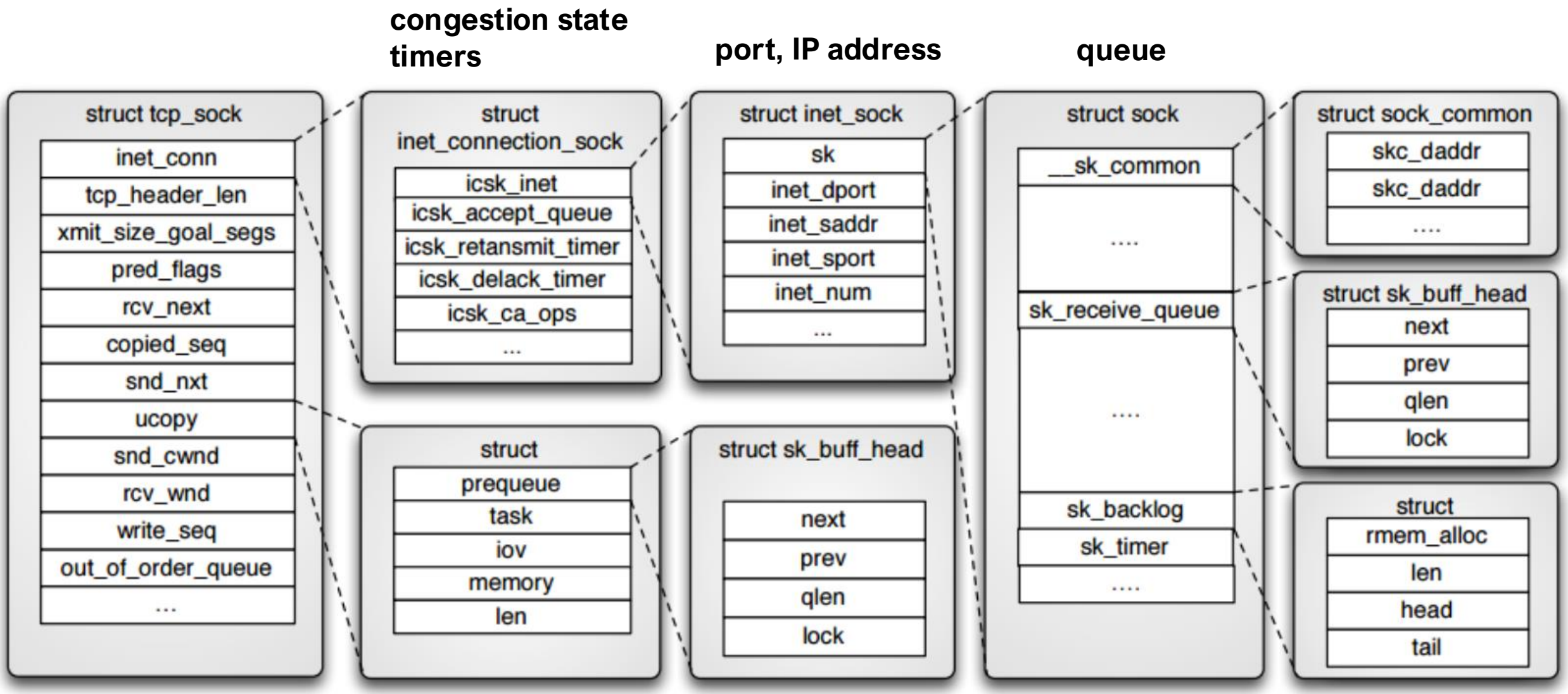
subject 3.

TCP in Linux

Overall data structure (again)

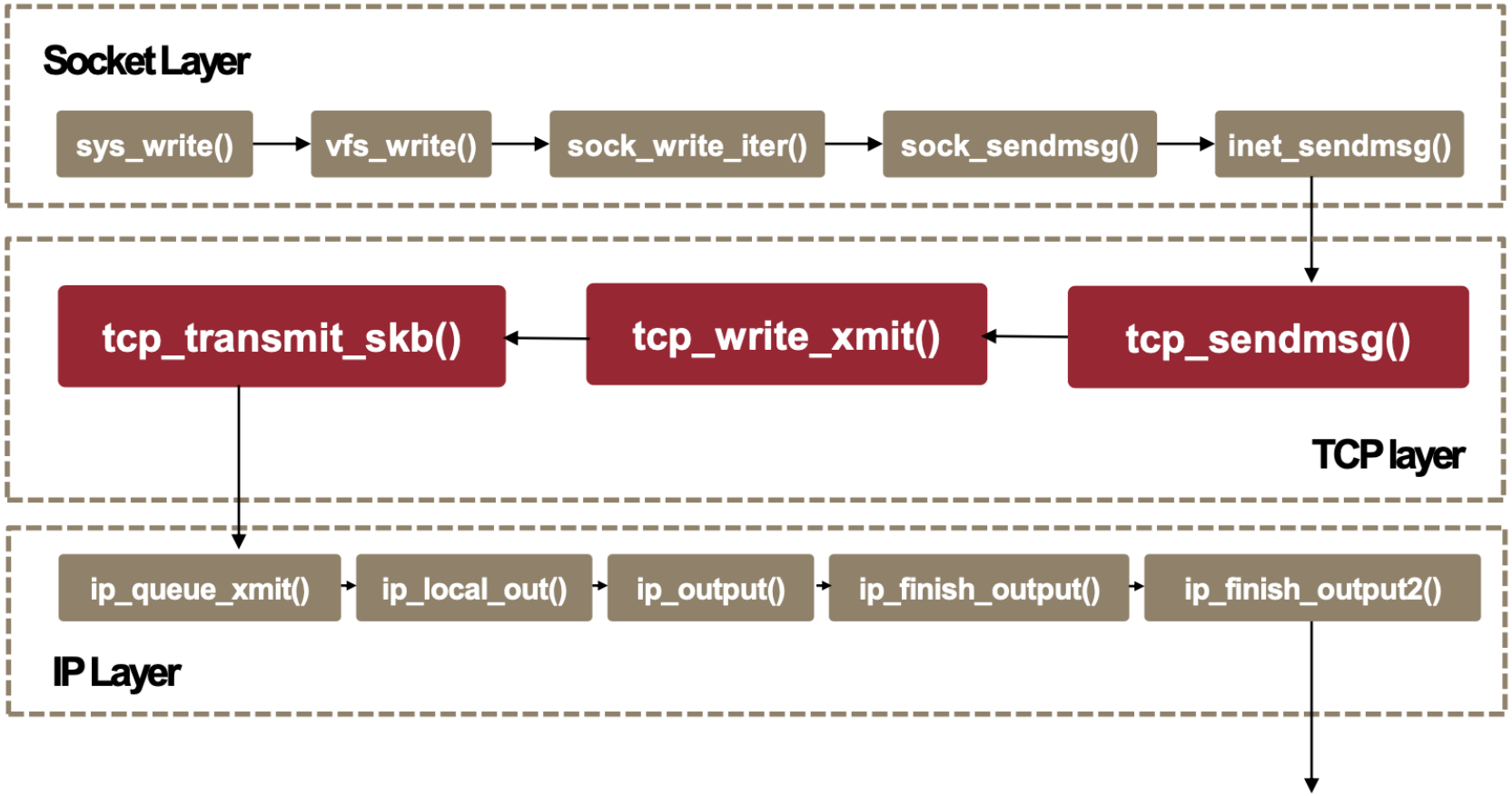


Data structures for TCP



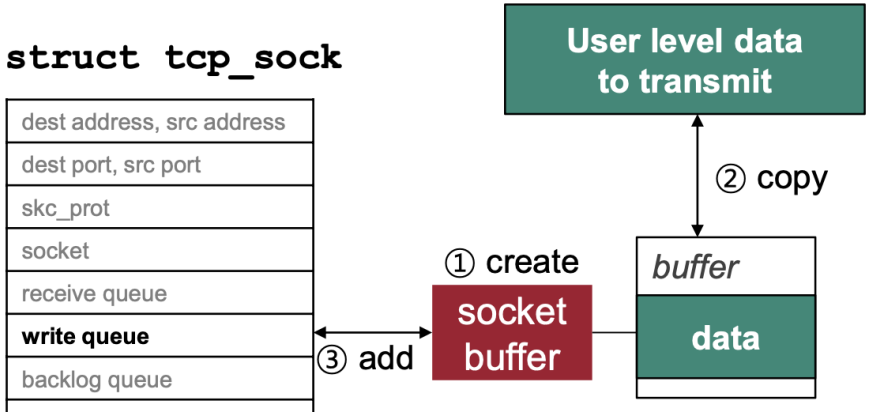
Sending segments in TCP

- Only one type of queue is used for outgoing data (write queue)
- 실제 데이터가 출발했어도, ack가 도착할 때까지는 skb를 유지해야한다.(재전송을 위해)
  - When an acknowledgement arrives from receiver, the corresponding skb is removed



Sending segments in TCP

tcp_sendmsg()

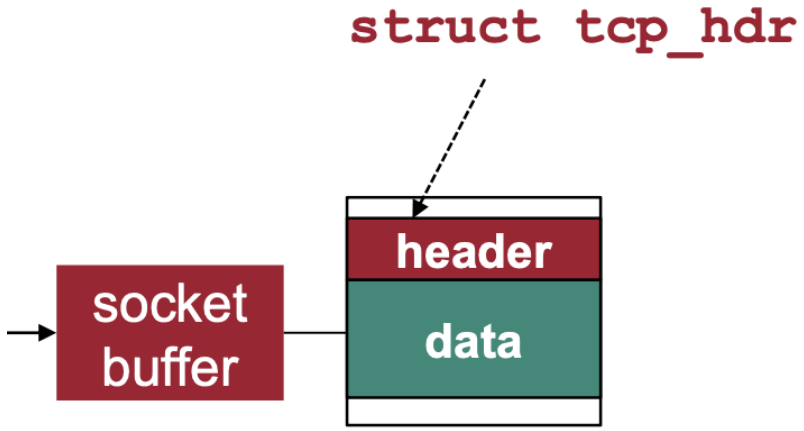


tcp_write_xmit()

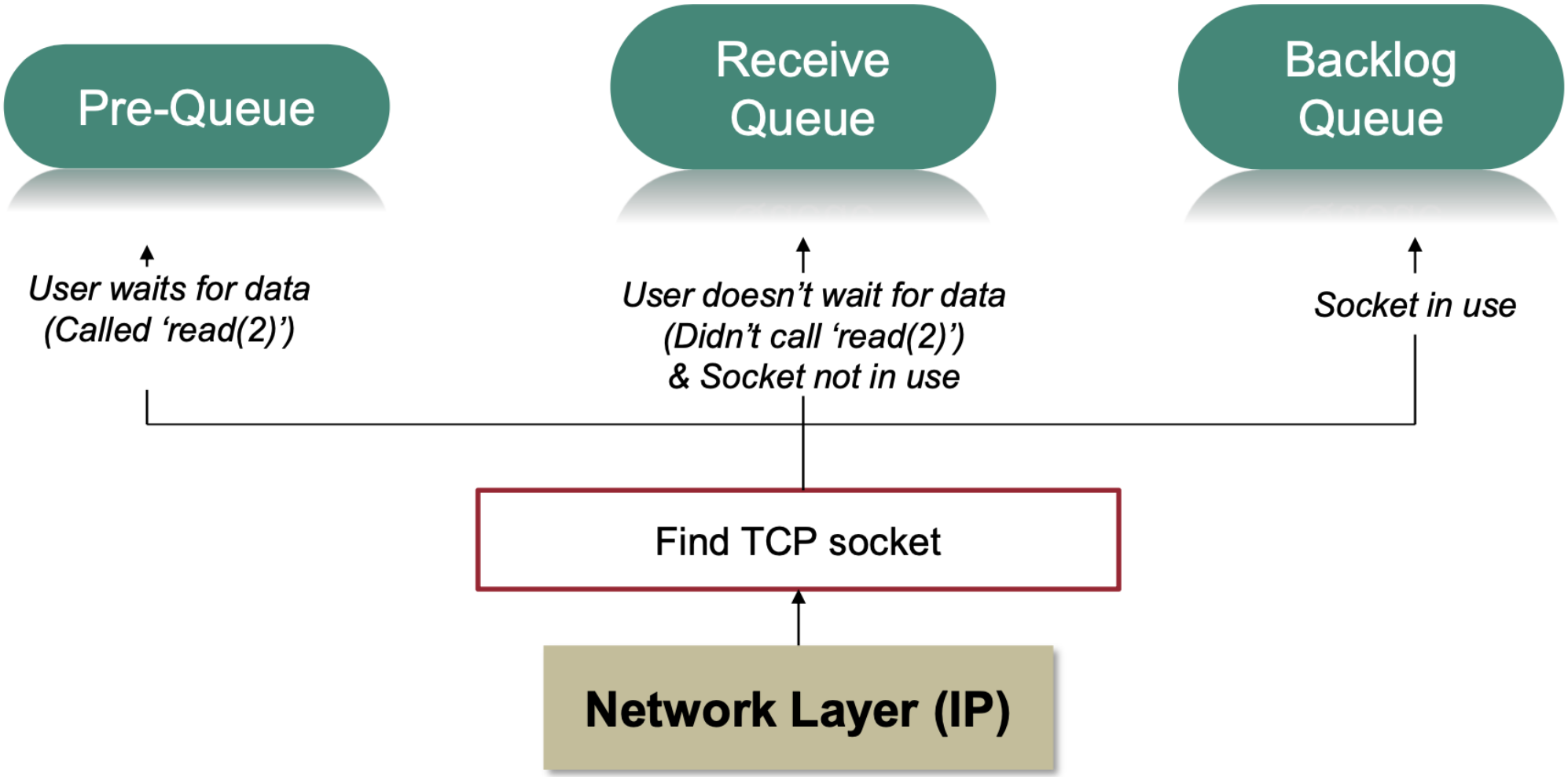
Congestion control,
Retransmission timer set

tcp_transmit_skb()

Build TCP header, checksum

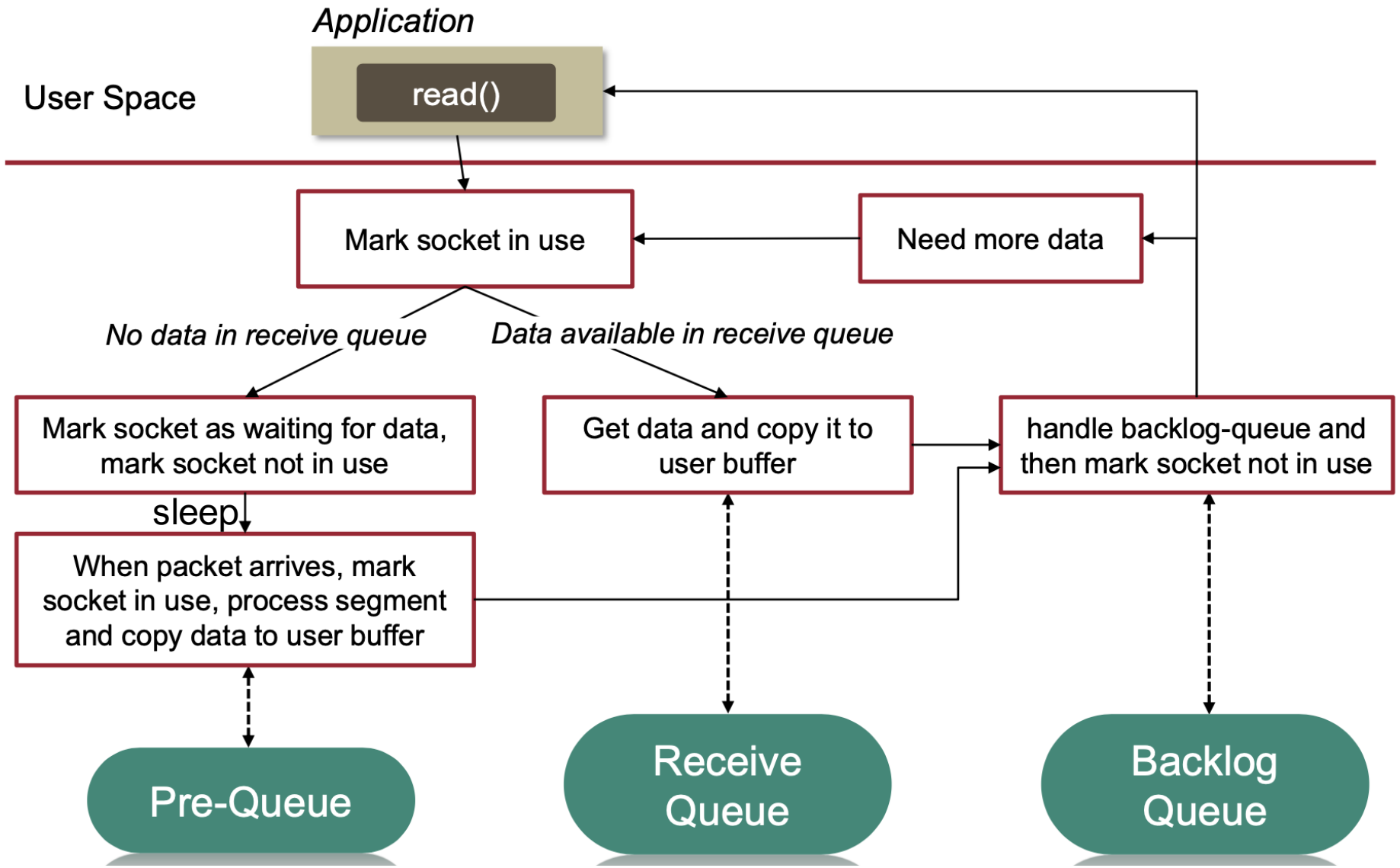


Receiving segments in TCP

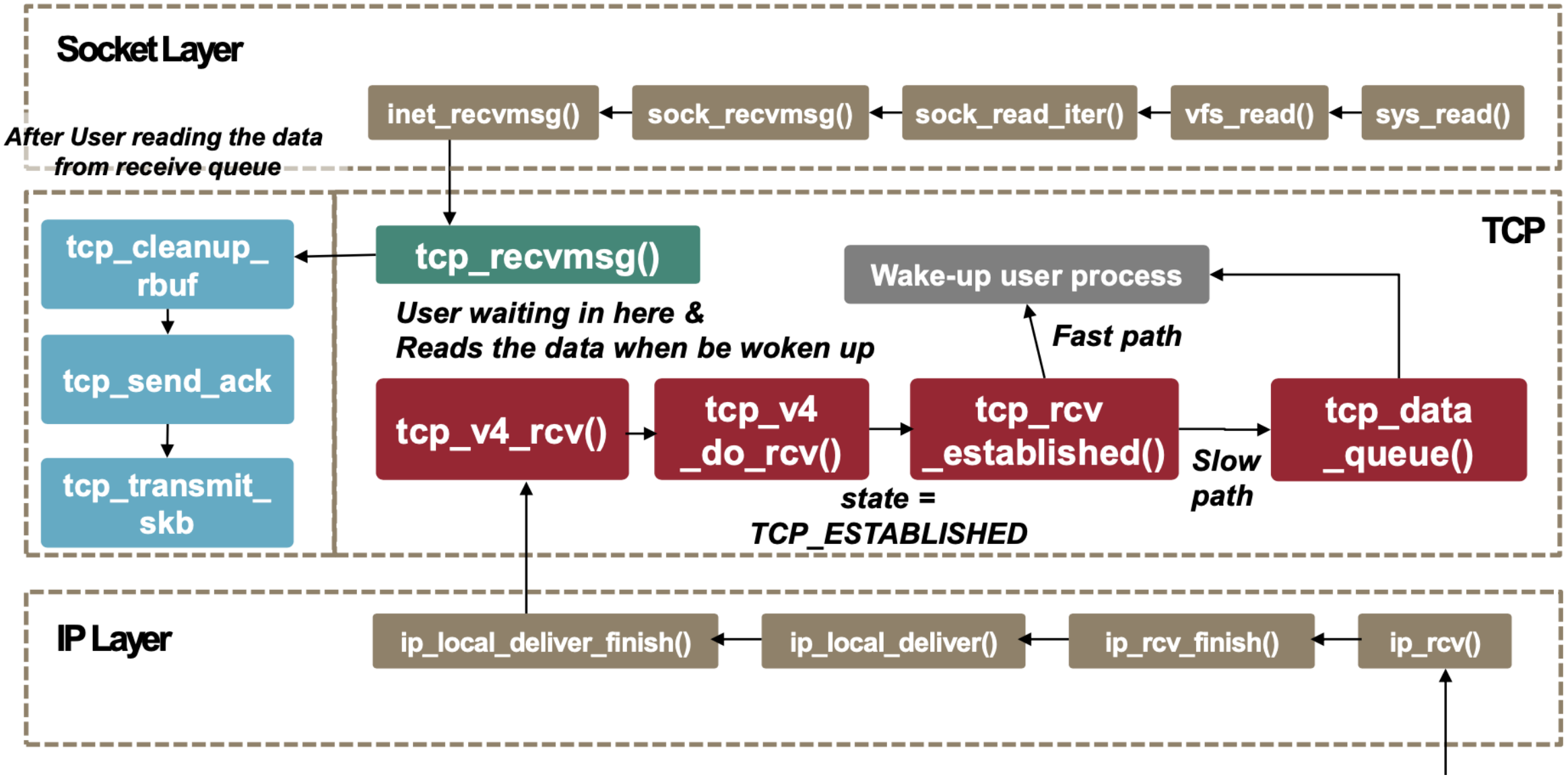


+ Out-of-order queue
: 패킷 순서가 순서대로 도착하지 않았을

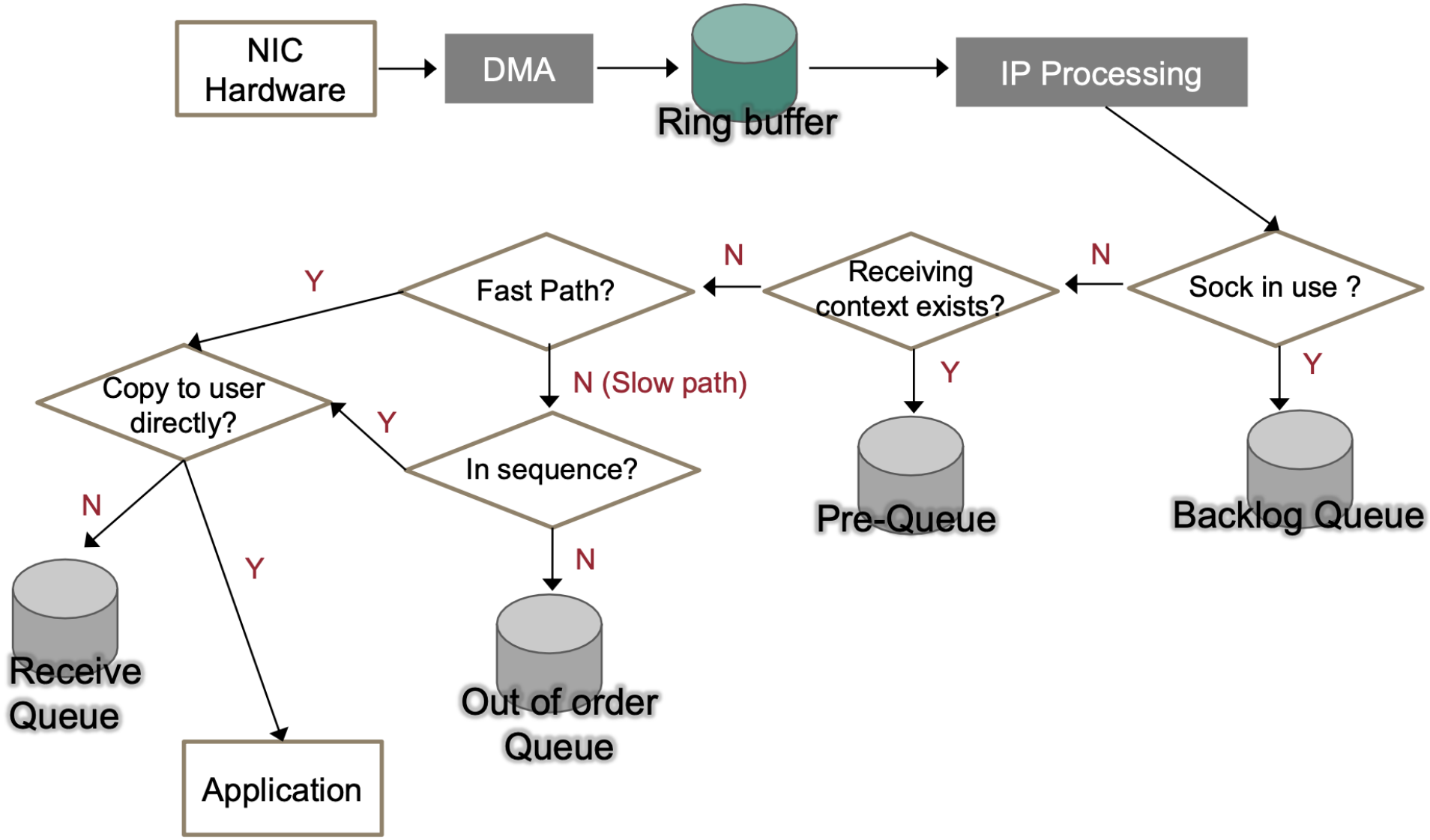
Incoming data queue : dequeue



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()
Slow path
tcp_data_queue()
state = TCP_ESTABLISHED
IP Layer
ip_local_deliver_finish()
ip_local_deliver()
ip_rcv_finish()
ip_rcv()

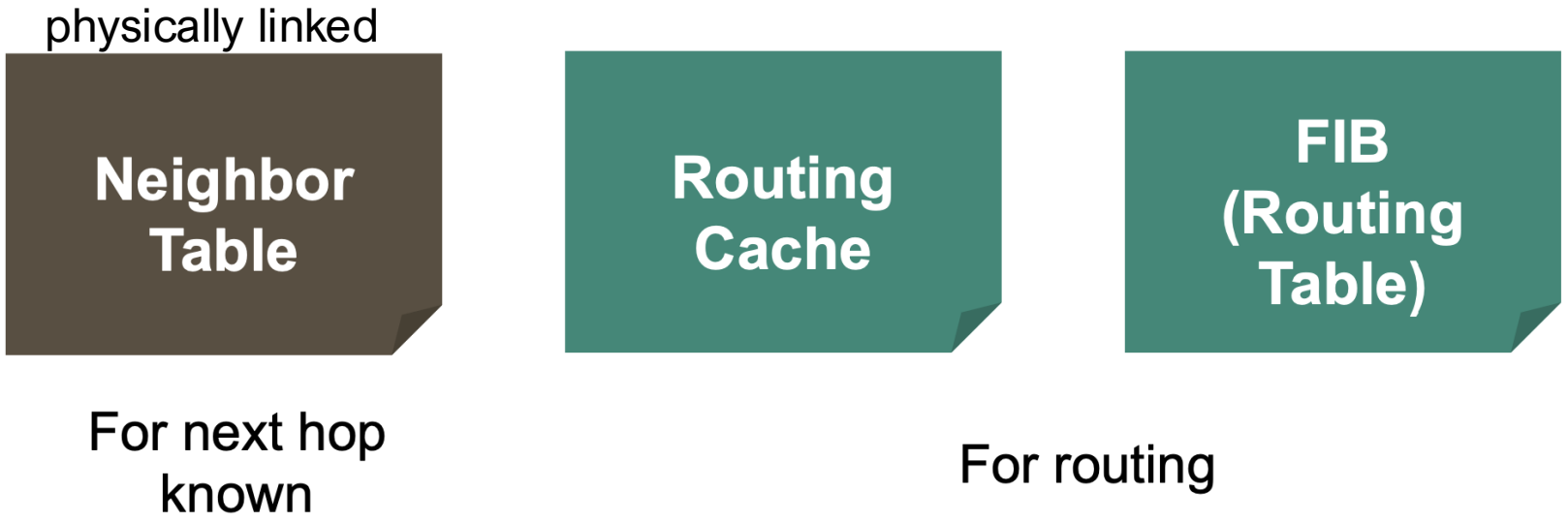
Summary of TCP Input



subject 4.

IP in Linux

Data structures for IP Routing (forwarding)

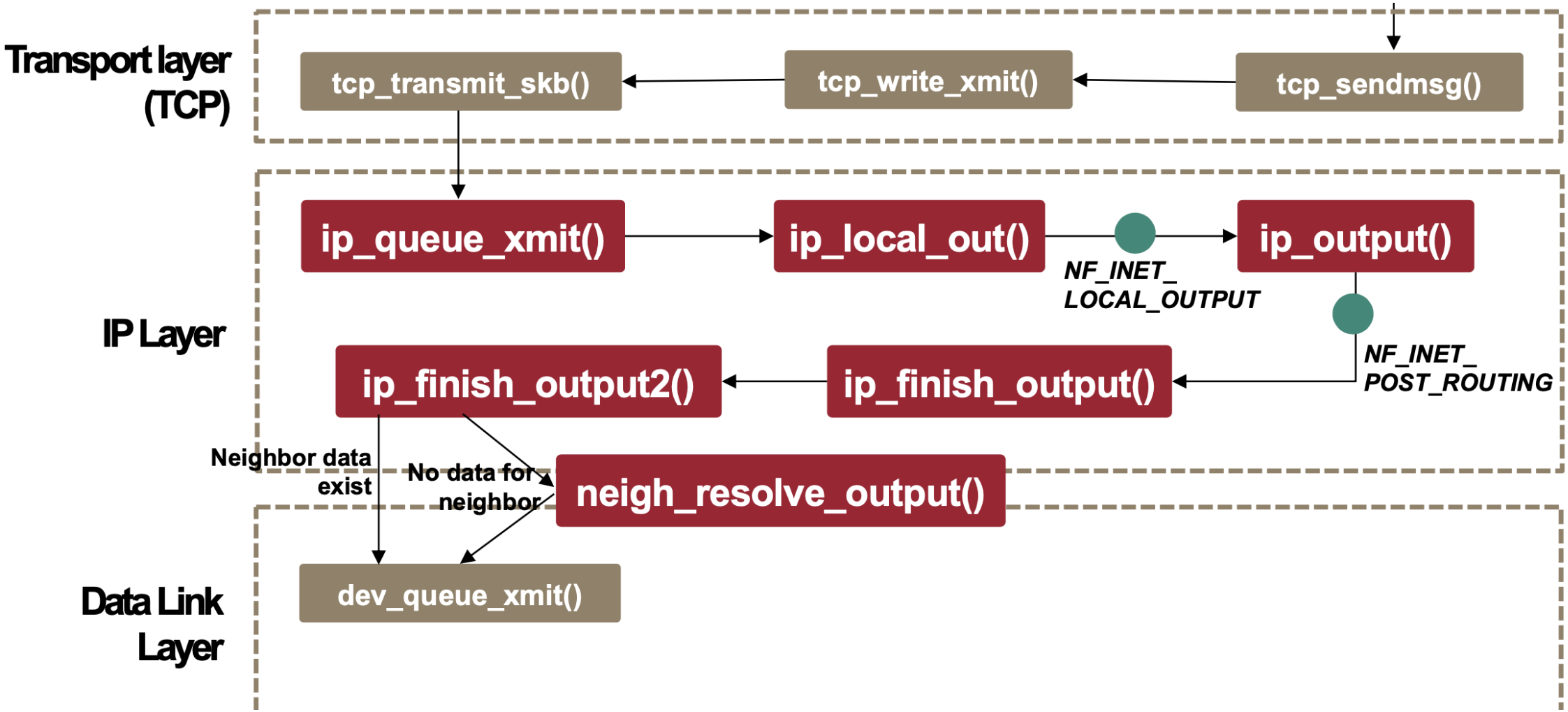


Neighbor Table
Linux uses the Address Resolution Protocol (ARP) to maintain and update this table

Routing Cache
It keeps every route that is currently in use or has been used recently in a hash table

FIB (Forwarding Information Base)
Called routing table
Mechanisms : find interface that matches longest IP address

IP Output


Transport layer (TCP)
tcp_transmit_skb()
tcp_write_xmit()
tcp_sendmsg()
IP Layer
ip_queue_xmit()
ip_local_out()
NF_INET_ LOCAL_OUTPUT
ip_output()
NF_INET_ POST_ROUTING
ip_finish_output2()
ip_finish_output()
Neighbor data exist
No data for neighbor
neigh_resolve_output()
Data Link Layer
dev_queue_xmit()

IP Output

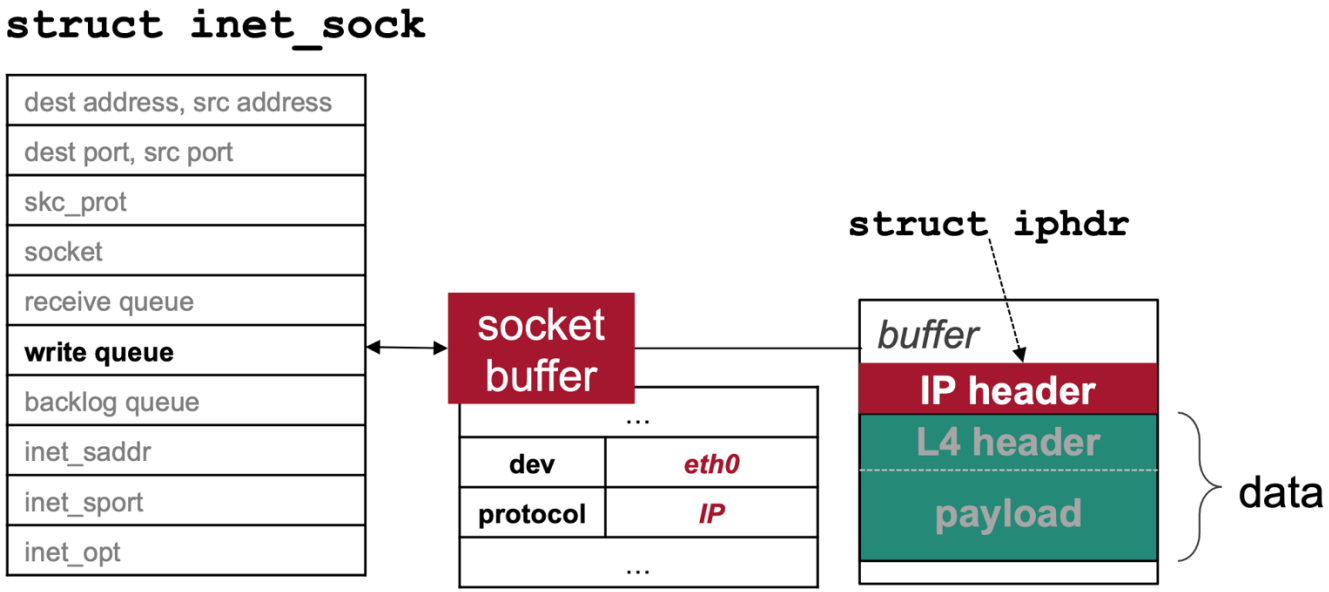
ip_queue_xmit() : initialize IP header, determine destination

ip_local_out() : compute checksum, check loopback

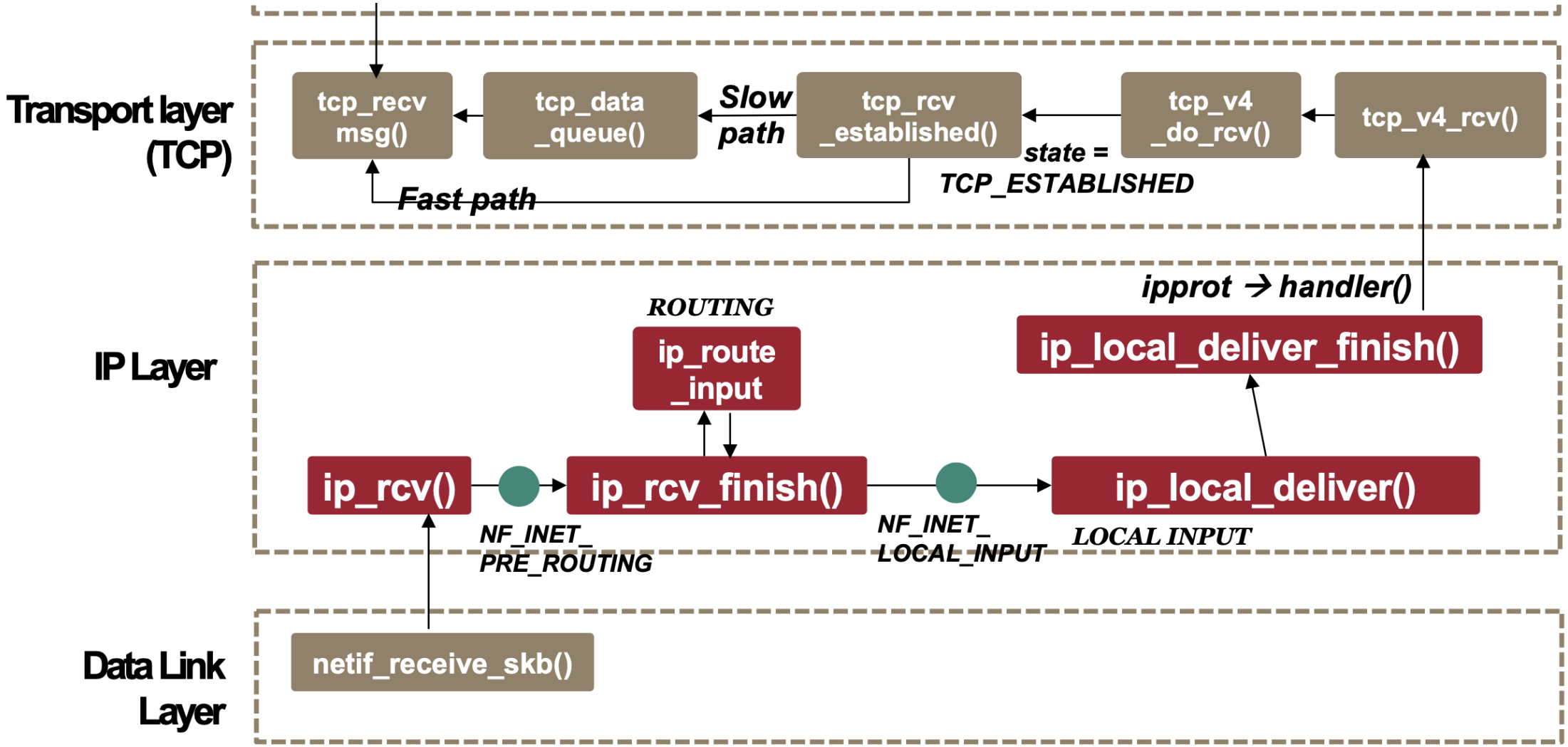
ip_output() : specify which device to use for transmit, specify protocol

ip_finish_output() : need fragmentation?(MTU) -> *ip_fragment()*

ip_finish_ouput2() : make space for L2 header , determine the nexthop(ARP)



IP Input


Transport layer (TCP)
tcp_recv msg()
tcp_data _queue()
Slow path
tcp_rcv _established()
tcp_v4 _do_rcv()
tcp_v4_rcv()
state = TCP_ESTABLISHED
Fast path
IP Layer
ROUTING
ip_route _input
ipprot → handler()
ip_local_deliver_finish()
ip_rcv()
ip_rcv_finish()
ip_local_deliver()
NF_INET_ PRE_ROUTING
NF_INET_ LOCAL_INPUT
LOCAL INPUT
Data Link Layer
netif_receive_skb()

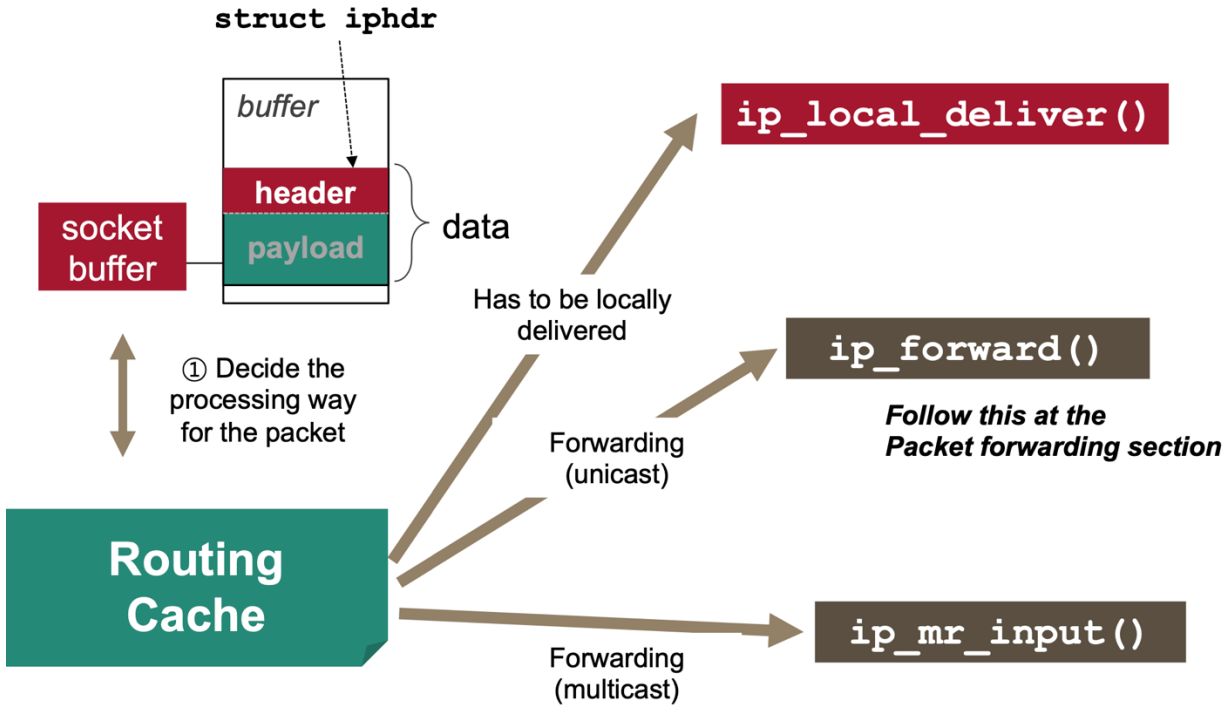
IP Input

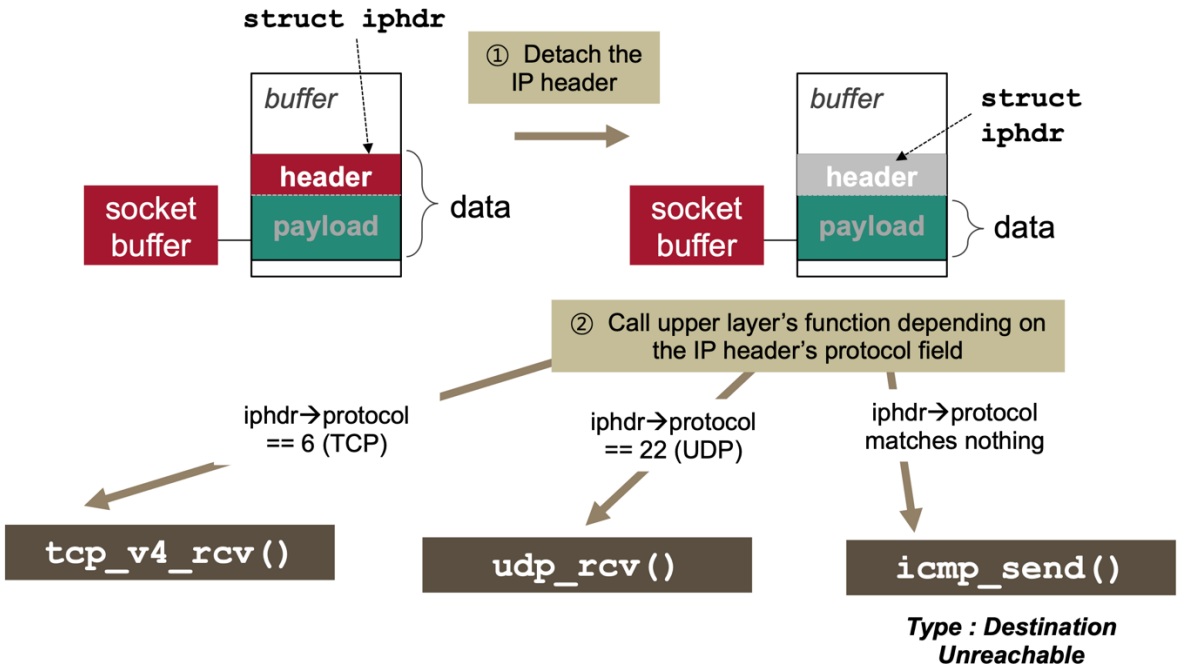
ip_rcv() : checking header size, ip version, checksum, length

ip_rcv_finish() : find the destination (local or forwarding)

ip_local_deliver() : do re-assemble

ip_local_deliver_finish() : remove the IP header from skb,
call upper layer (tcp_v4_rcv(), udp_rcv(), icmp_send())

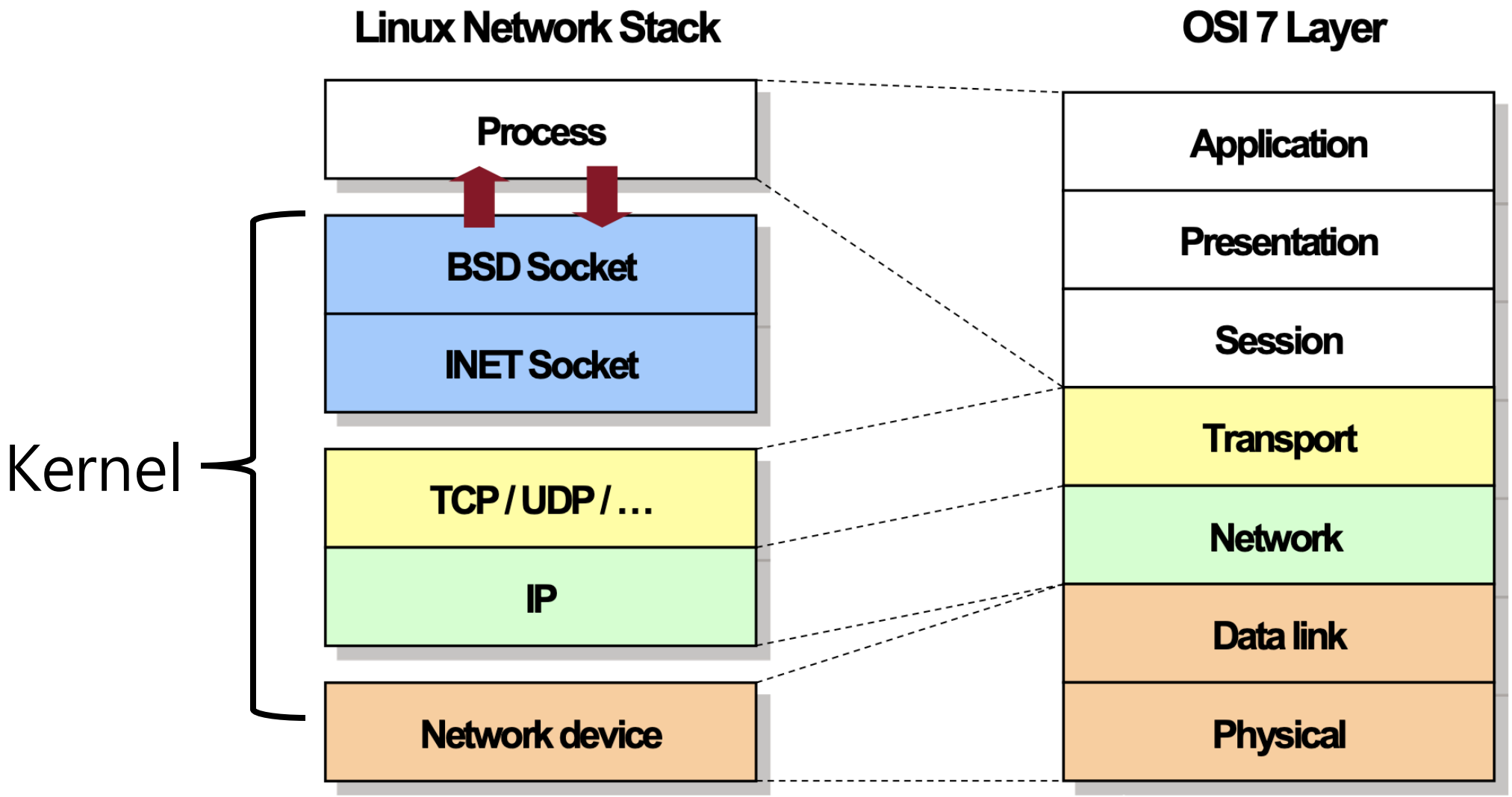




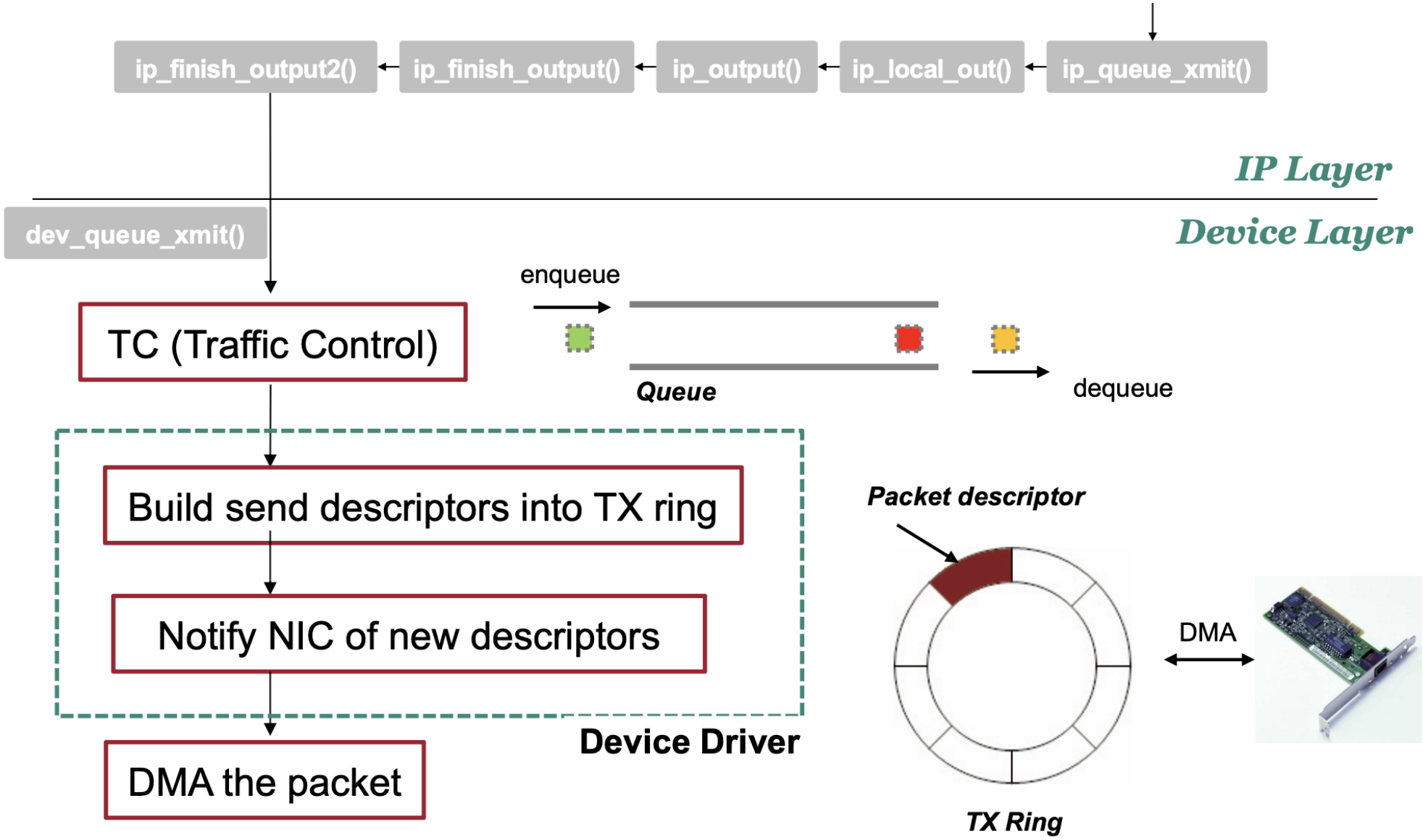
subject 5.

Data link layer in ~~Linux~~

Protocol Stack in Linux Kernel



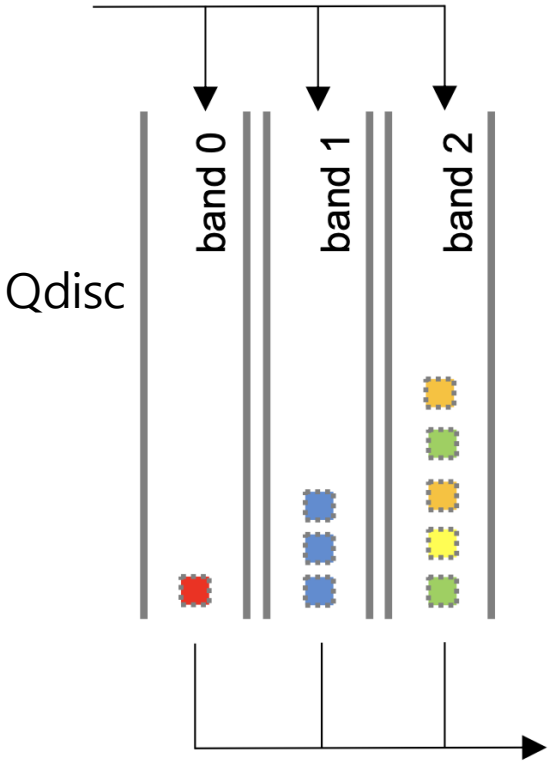
Sending data frames



Traffic Control (TC)

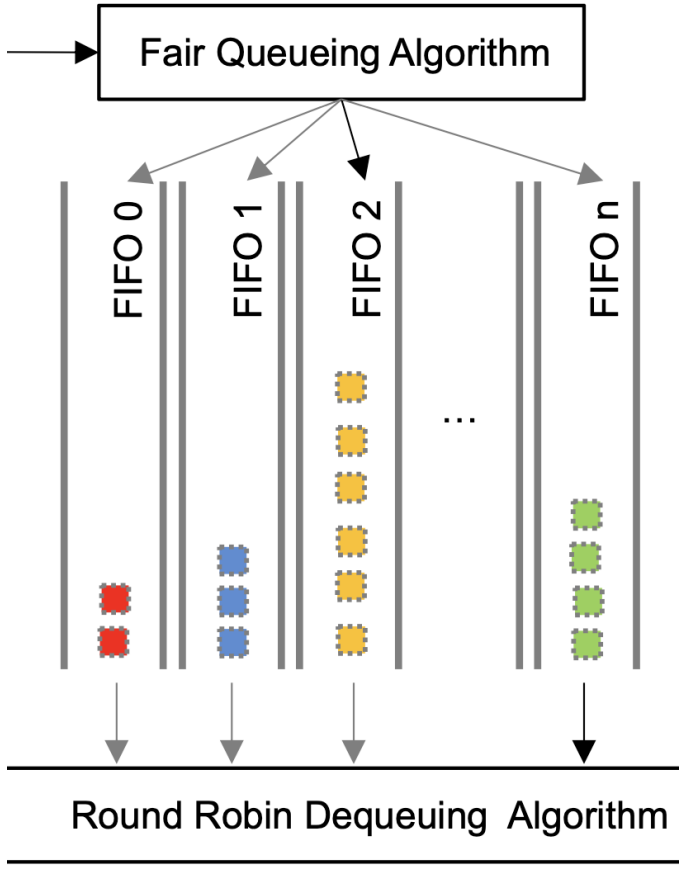
Throttle frames : Control to transmit and receive packets

pfifo_fast

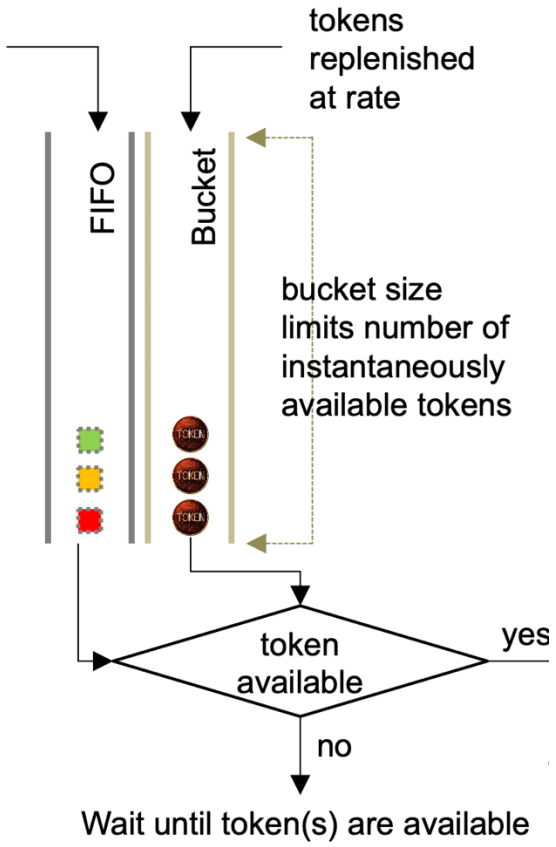


priority
band 0 > band 1 > band2

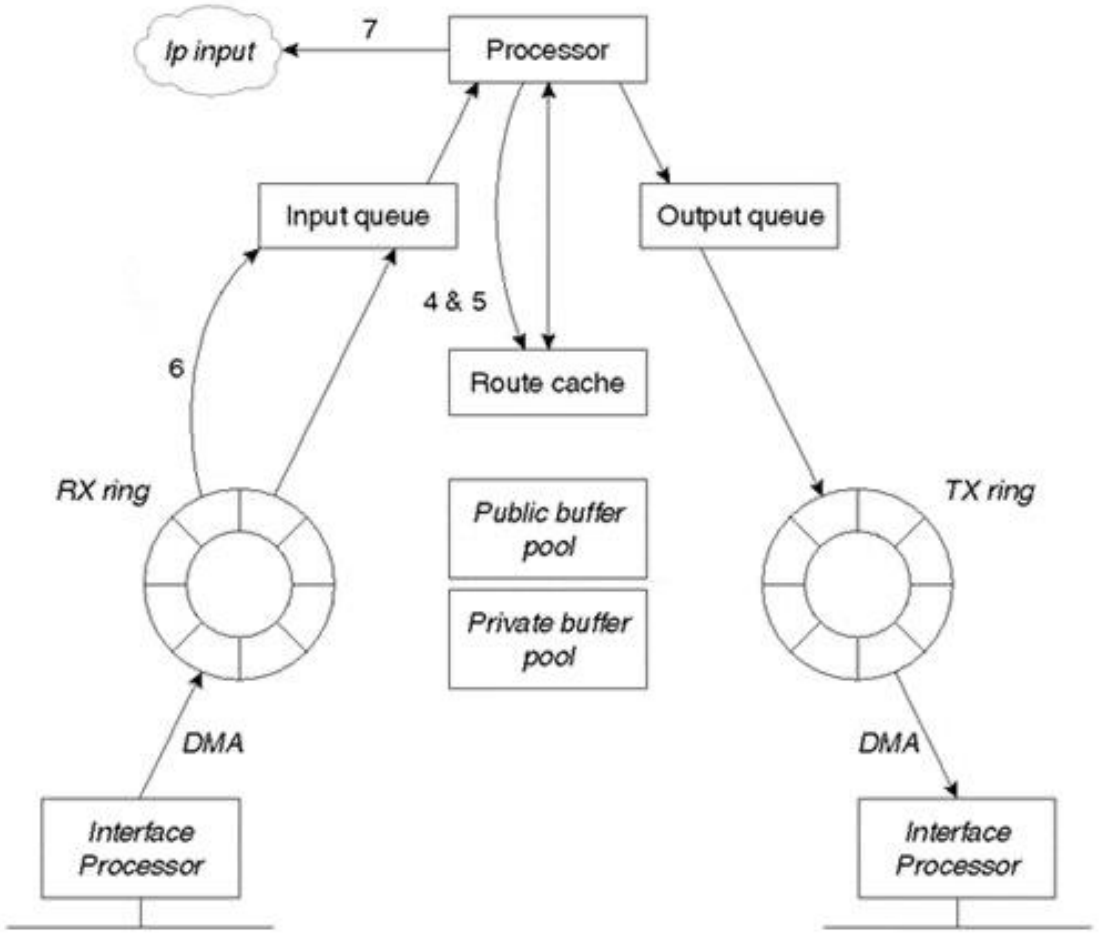
SFQ(Stochastic Fair Queueing)



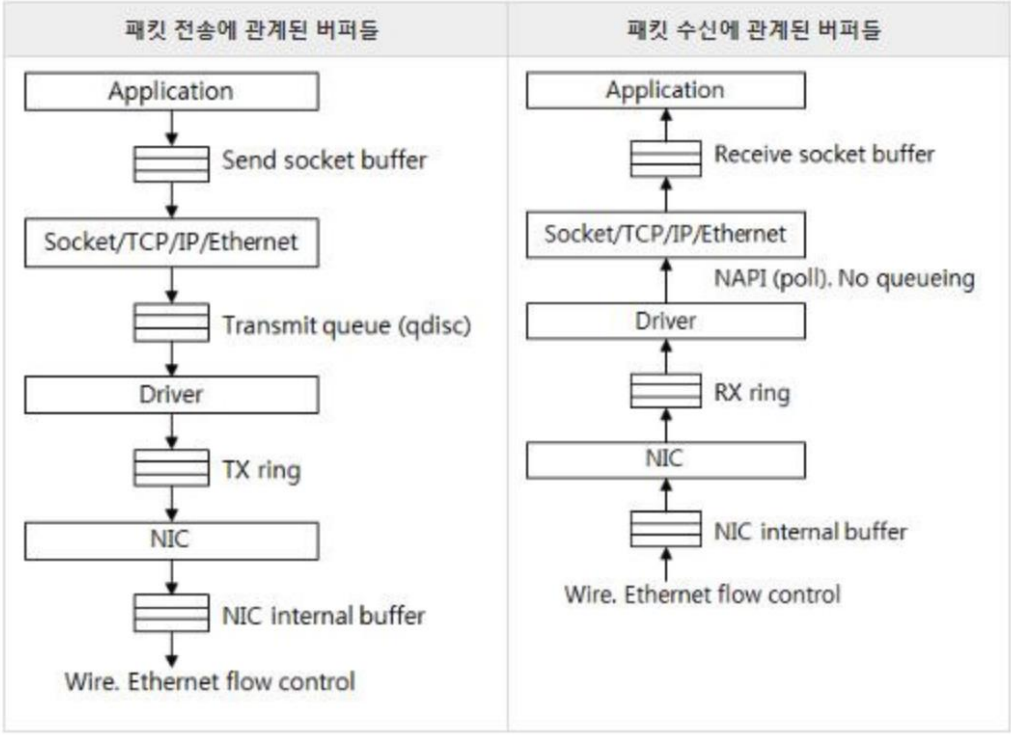
TBF(Token Bucket Filter)



TX / RX Ring

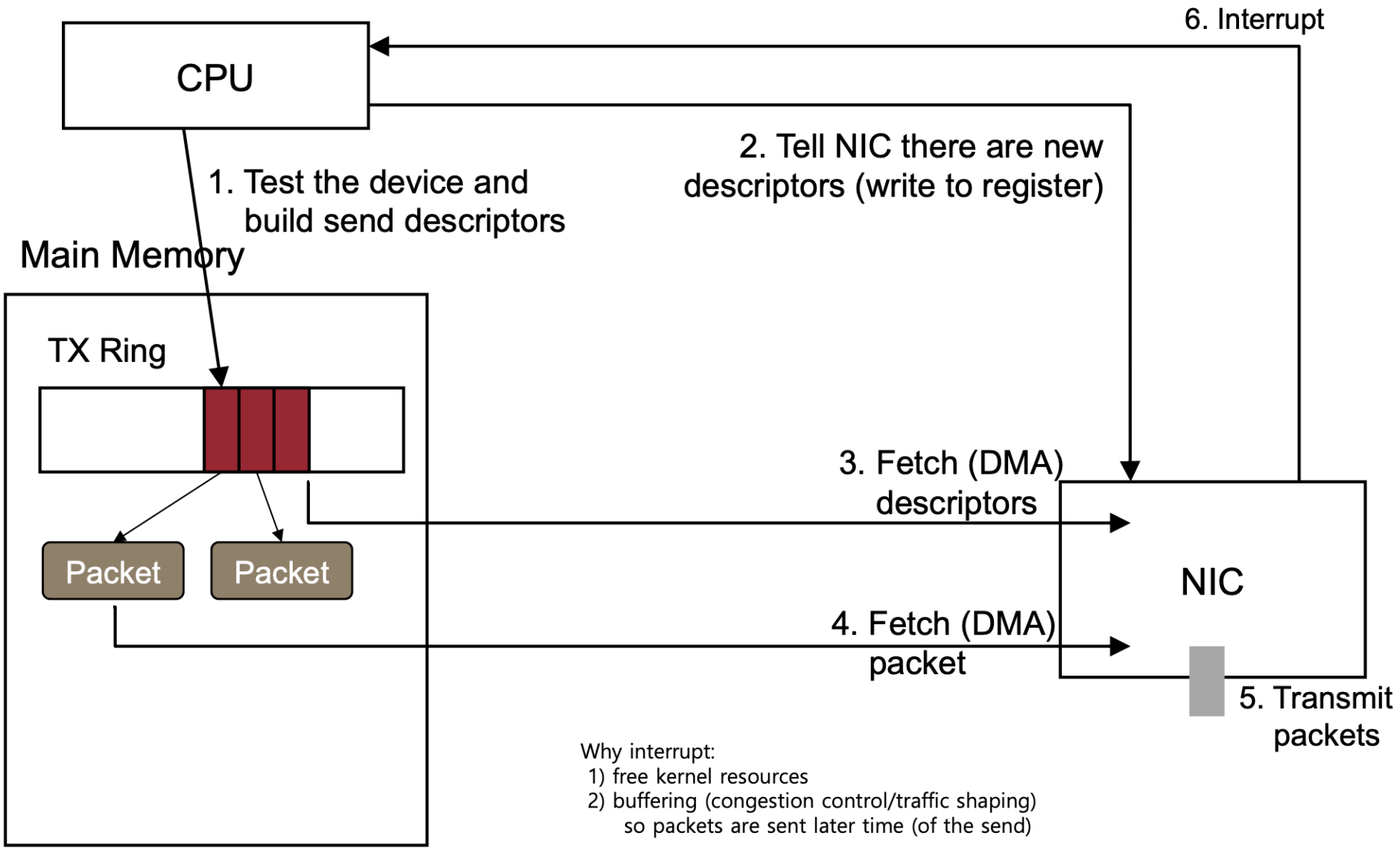



```
root@CT100:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.39  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::be24:11ff:fed0:bd8a  prefixlen 64  scopeid 0x20<link>
        ether bc:24:11:d0:bd:8a  txqueuelen 1000  (Ethernet)
        RX packets 718994  bytes 316807709 (316.8 MB)
        RX errors 0  dropped 14  overruns 0  frame 0
        TX packets 442852  bytes 58145609 (58.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```



CF) NIC ring buffer는 maximum 값으로 설정 권장

Sending data between device driver and NIC



Why interrupt:
1) free kernel resources
2) buffering (congestion control/traffic shaping)
so packets are sent later time (of the send)

New packet receiving mechanism

- Originally, Linux takes one interrupt per received frame.
- If packets **arrive too fast**, system spends most time servicing interrupts coming from packets received
  - Actual packet processing (SoftIRQ) is preempted continuously
  - When the queue for received packet is full, packets will be dropped in the interrupt handler
  - Eventually, throughput begins to fall

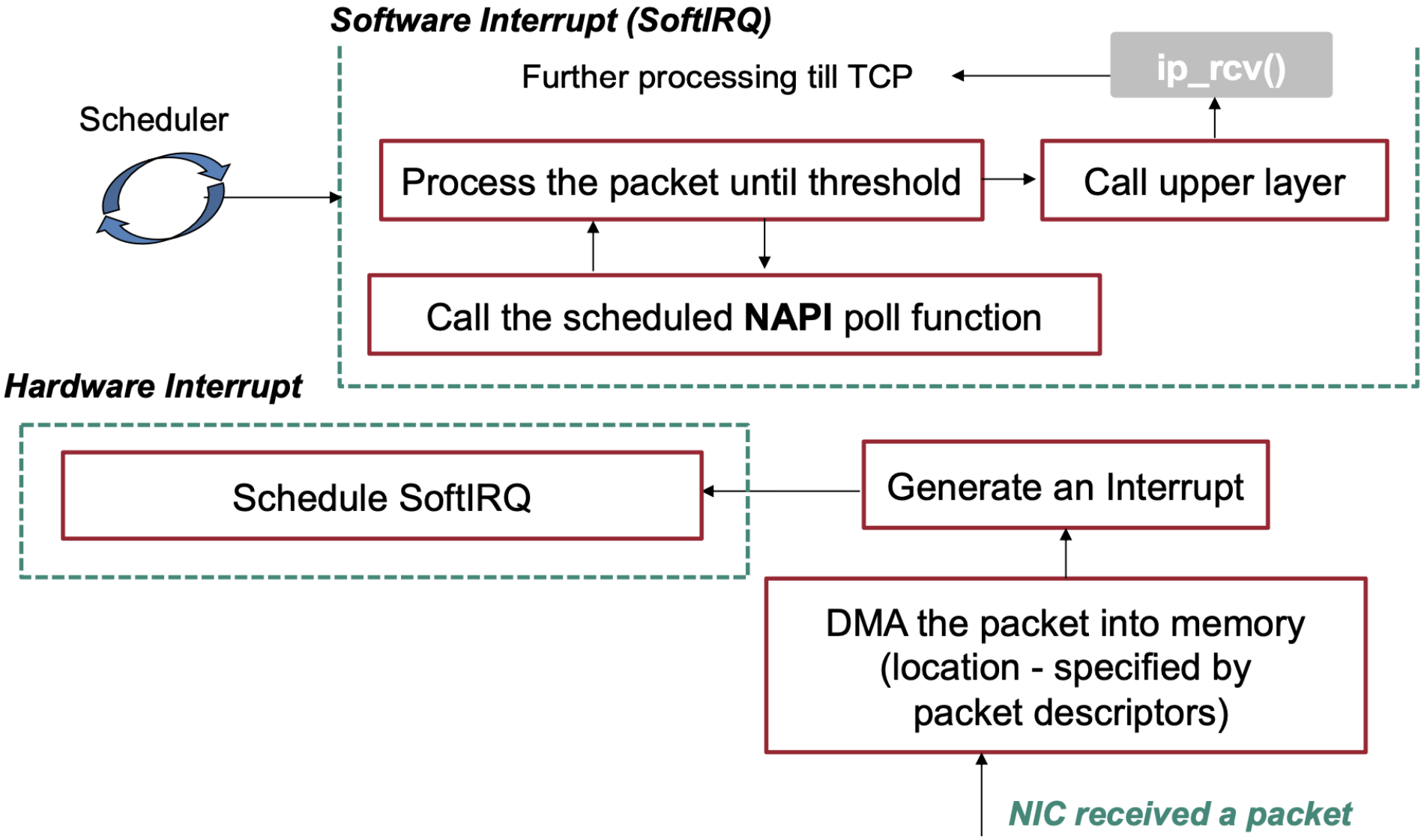
NAPI is provided to solve this problem

Packet reception with NAPI

NAPI: “New API”

- Packet processing mechanism designed to better support the drivers for the "fast" network adapters
- Implementation
  - Generate an interrupt immediately on the first frame received
  - Switch into **polling mode (disable interrupts)** (대부분의 패킷은 뭉쳐서 들어온다.)
  - Re-enable interrupts after polled for some number of frames or some time interval
- Most drivers support this now

Receiving data between device driver and NIC



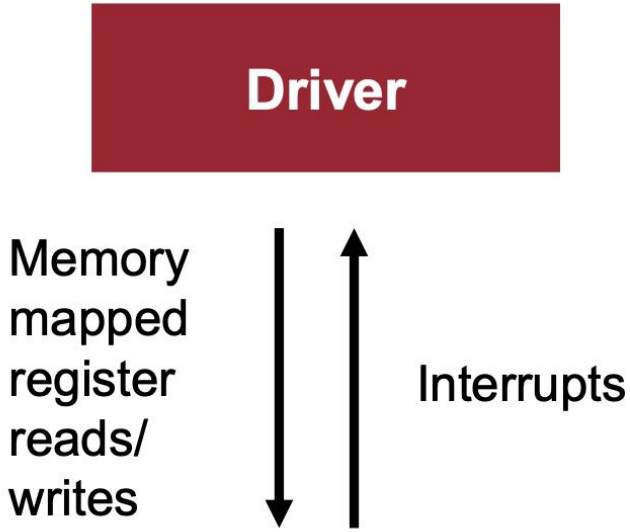
subject 6.

Network adapter (NIC)

Driver <-> Network Adapter Interface

Driver(SW) talks to the device(HW)

- Writing commands to memory-mapped *control registers*
- Setting aside buffers for packet transmission/reception
- buffers described in *descriptor rings*



Device talks to driver

- Place values in control registers
- DMA packets to/from available buffers
- Generate *interrupts* (both on send and receive)
- Update status in descriptor rings(receive)

Interrupt handling philosophy in Kernel

- **Do as little as possible in the interrupt handler**
- **Defer non-critical actions till later**
- **Structure: top and bottom halves**
  - Top-half (Hardware Interrupt): do minimum work and return
    - ISR(Interrupt Service Routine)
  - Bottom-half (Software Interrupt): deferred processing
    - softirqs, tasklets, workqueues

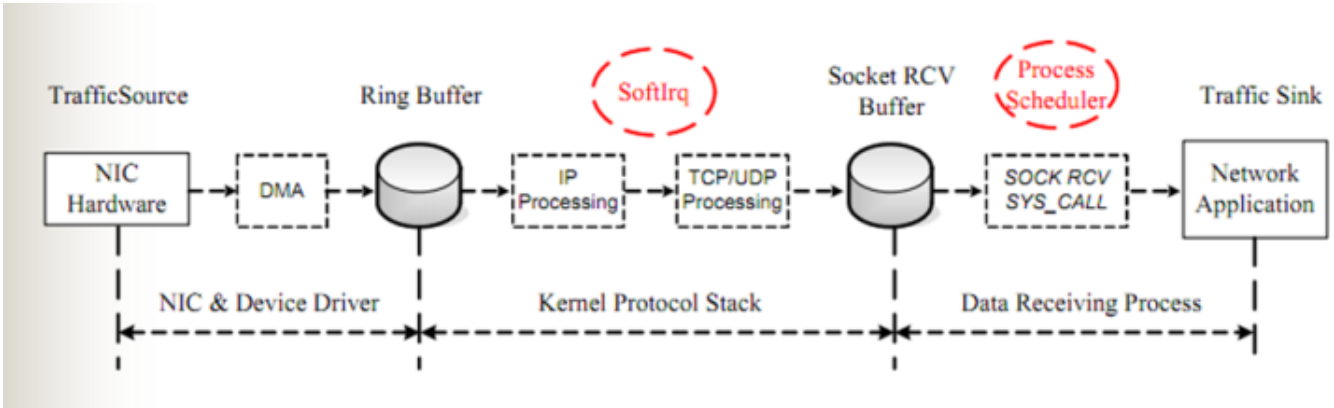
Top Half / Bottom Half

Top Half (hardware interrupt)

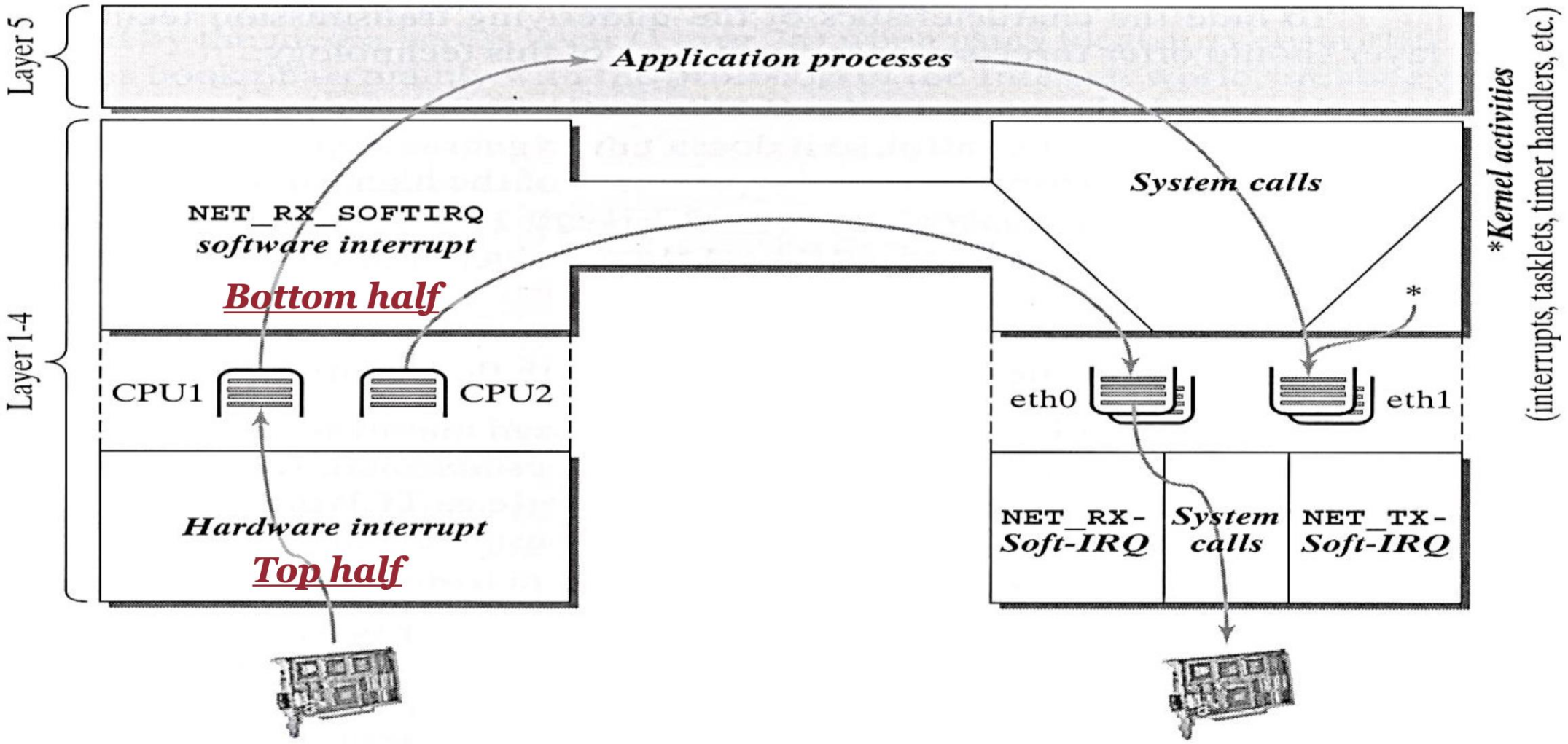
- it preempts other tasks
- 인터럽트 핸들러에서 최소한의 연산을 수행
- IRQ is typically masked for duration of top half

Bottom Half (software interrupt)

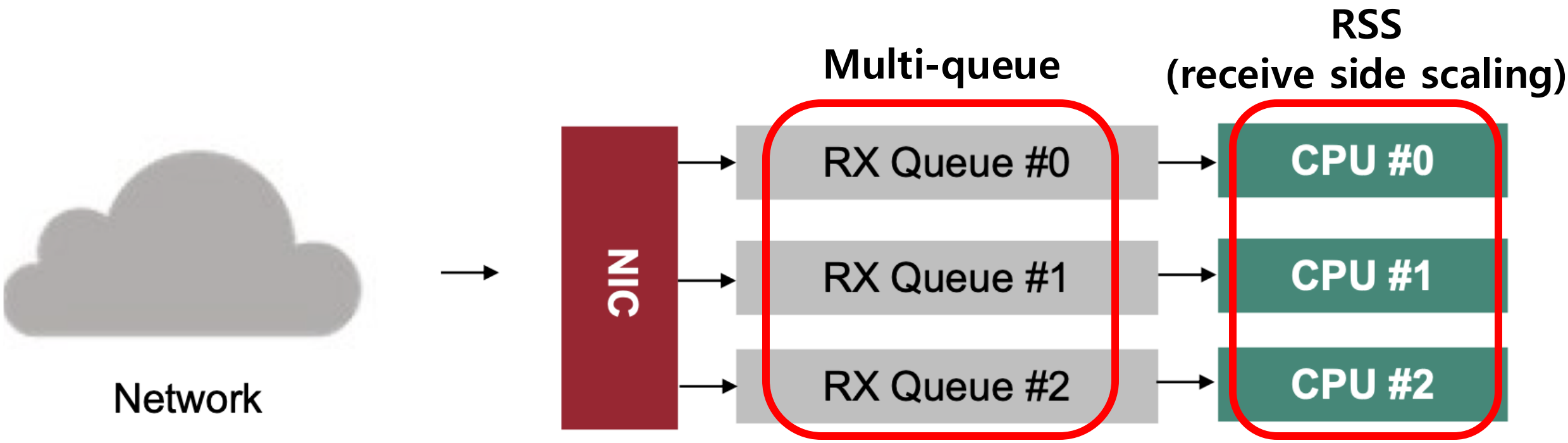
- Top half 에서는 최소한의 일을 수행하고, 나머지는 Bottom Half 부분에서 처리
- Network transmission and reception is handled with softirq



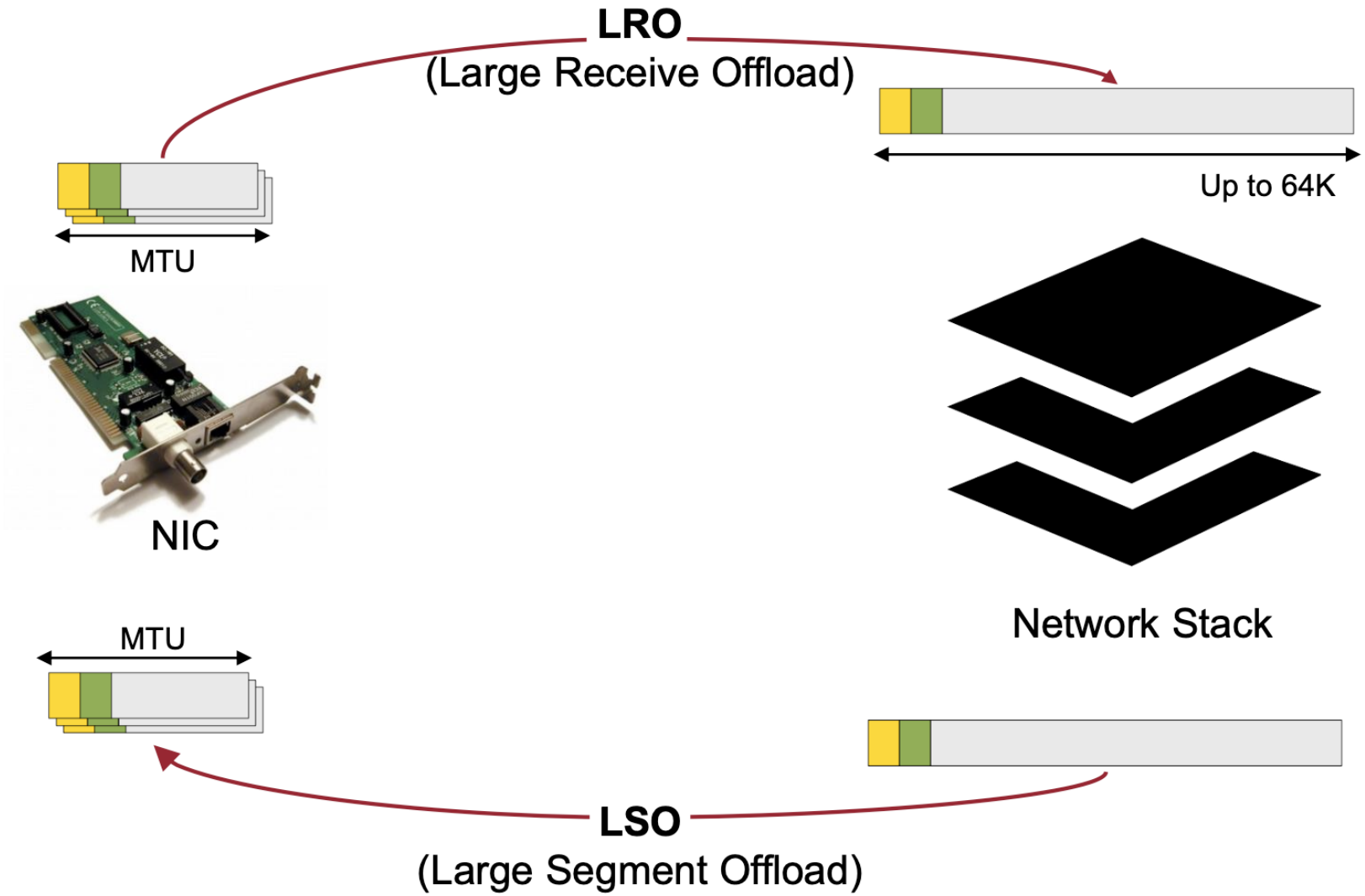
Network interrupt



Advanced functionalities – Multi queue, RSS



Advanced functionalities – LSO, LRO



Advanced functionalities – else

checksum offloading

- NIC computes the checksum of transmitting IPv4 packets and verify the checksum of receiving IPv4 packets
- Takes the load for checksum calculation off from the CPU

Promiscuous mode

- A mode that causes the NIC to read all traffic arriving at the host, rather than reading only the frames that are destined to the host
- Normally used for packet sniffing

vectored I/O (scatter-gather I/O)

- sequentially reads data from multiple buffers and writes it to a single data stream (gather)
- reads data from a data stream and writes it to multiple buffers (scatter)

Reference

https://www.youtube.com/watch?v=o59PLhxIdjo

system programming (korea university, 유혁)

Operating System Concepts (10/E, Silberschatz)

Computer Systems A Programmer's Perspective (3/E, Randal E. Bryan)

Computer Organization And Design (6/E, David A. Patterson)