Operating System Design and Implementation

Lecture 22: Networking driver

Tsung Tai Yeh

Tuesday: 3:30 – 5:20 pm

Classroom: ED-302

Acknowledgements and Disclaimer

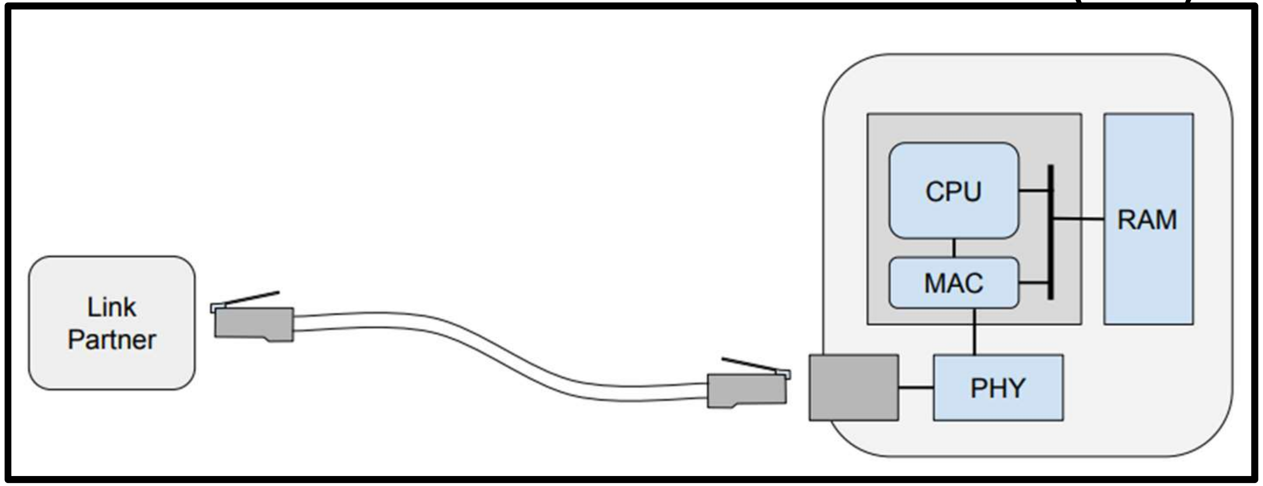
- Slides was developed in the reference with
  MIT 6.828 Operating system engineering class, 2018
  MIT 6.004 Operating system, 2018
  Remzi H. Arpaci-Dusseau etl. , Operating systems: Three easy pieces. WISC
  Onur Mutlu, Computer architecture, ece 447, Carnegie Mellon University
- CSE 506, operating system, 2016,
  https://www.cs.unc.edu/~porter/courses/cse506/s16/slides/sync.pdf

Outline

- The packet flow in NIC
  - RX/TX flow path
  - Kernel space operations
- NIC hardware driver
  - struct sk_buff
  - Reception and transmission operations

The NIC hardware

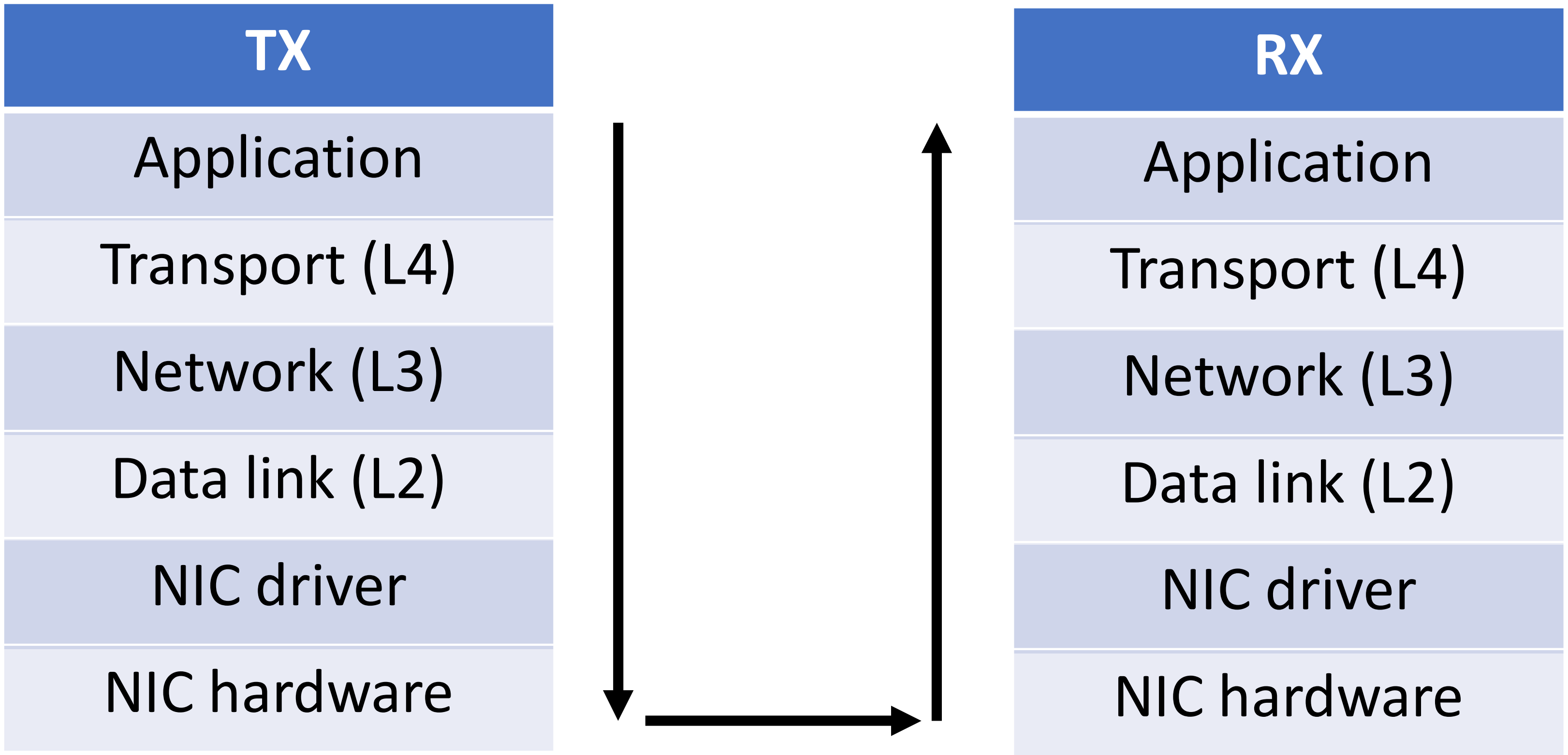
Network Interface Controller (NIC)



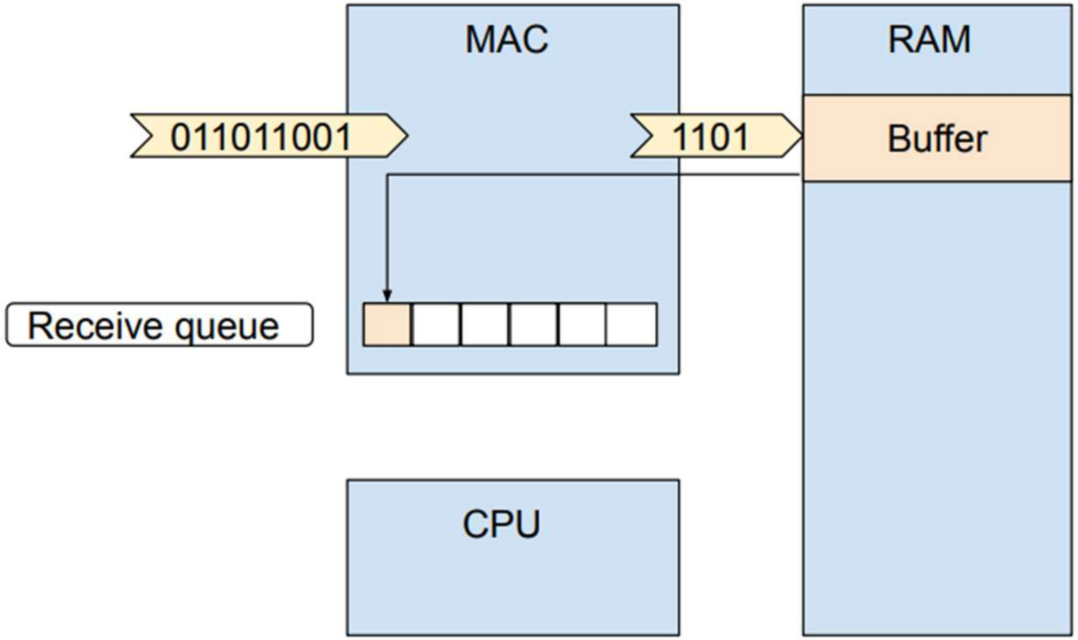
- **The NIC**
  - **Connector**: RJ45 cable, SFP etc.
  - **Media:** Copper, Fiber, Radio
  - **PHY:** Convert media-depend signal into standard data
  - PCIe network card embed a PHY
  - **The MAC**
    - Handle L2 protocol, transfer data to the CPU

https://bootlin.com/pub/conferences/2021/fosdem/chevallier-network-performance-in-the-linux-kernel/chevallier-network-performance-in-the-linux-kernel.pdf

The packet flow



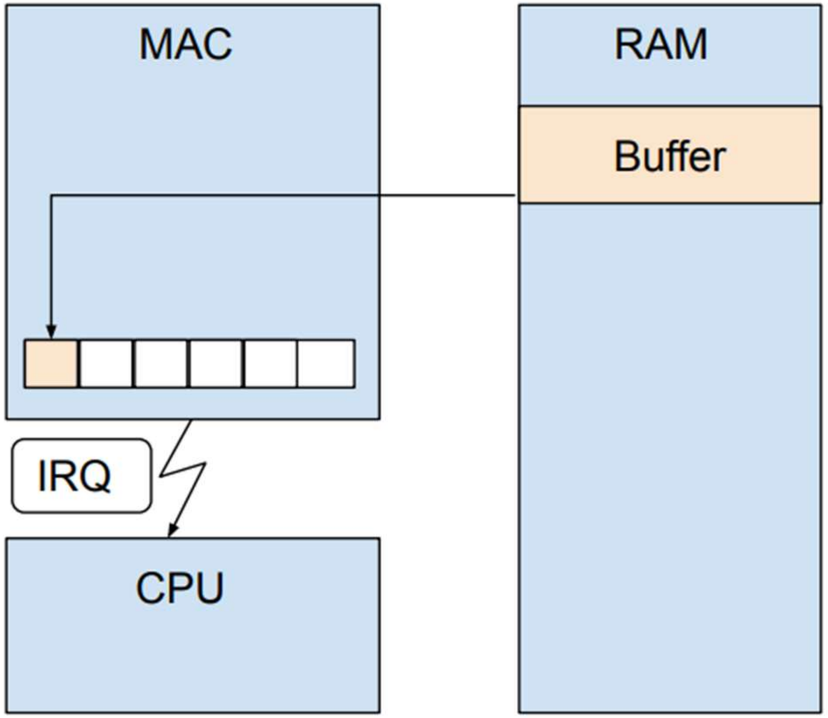
L2 frame reception



https://bootlin.com/pub/conferences/2021/fosdem/chevallier-network-performance-in-the-linux-kernel/chevallier-network-performance-in-the-linux-kernel.pdf

- The **MAC** received data and write it to RAM using **DMA**
- A descriptor is created
- Its address is put in a queue

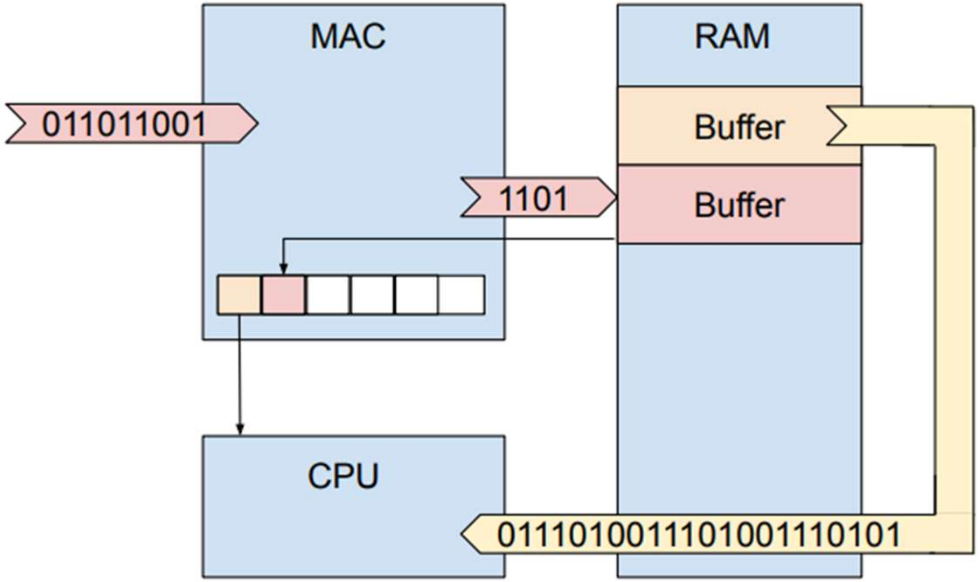
L2 frame reception -- IRQ



https://bootlin.com/pub/conferences/2021/fosdem/chevallier-network-performance-in-the-linux-kernel/chevallier-network-performance-in-the-linux-kernel.pdf

- A interrupt is fired
- One CPU core will handle the interrupt

L2 frame reception -- Unqueue



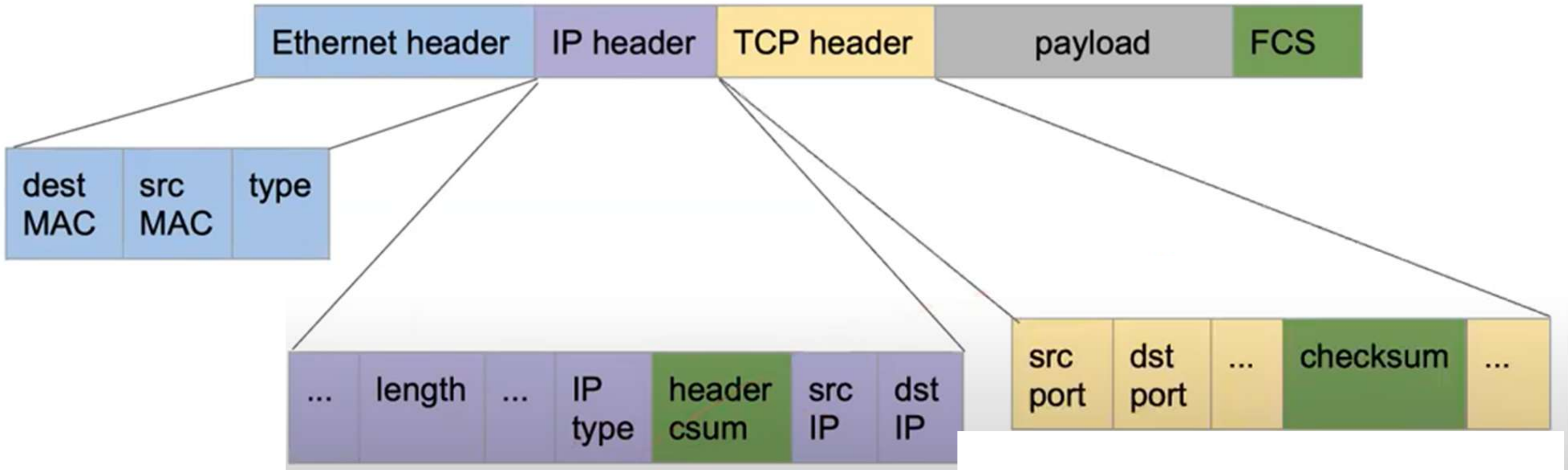
- The interrupt handler acknowledges the interrupt
- The packet is processed in softirq context
- The new frame can be received in parallel

In the NIC driver

- **The CPU**
  - Processes L3 (packets) and above, up to the application
  - The interrupt handler masks interrupts
- **New API (NAPI)**
  - An extension to the device driver in packet processing framework
  - Schedules the processing in batches
  - Stop de-queueing once, because
    - The budget is expired (release the CPU to the scheduler)
    - The queue is empty

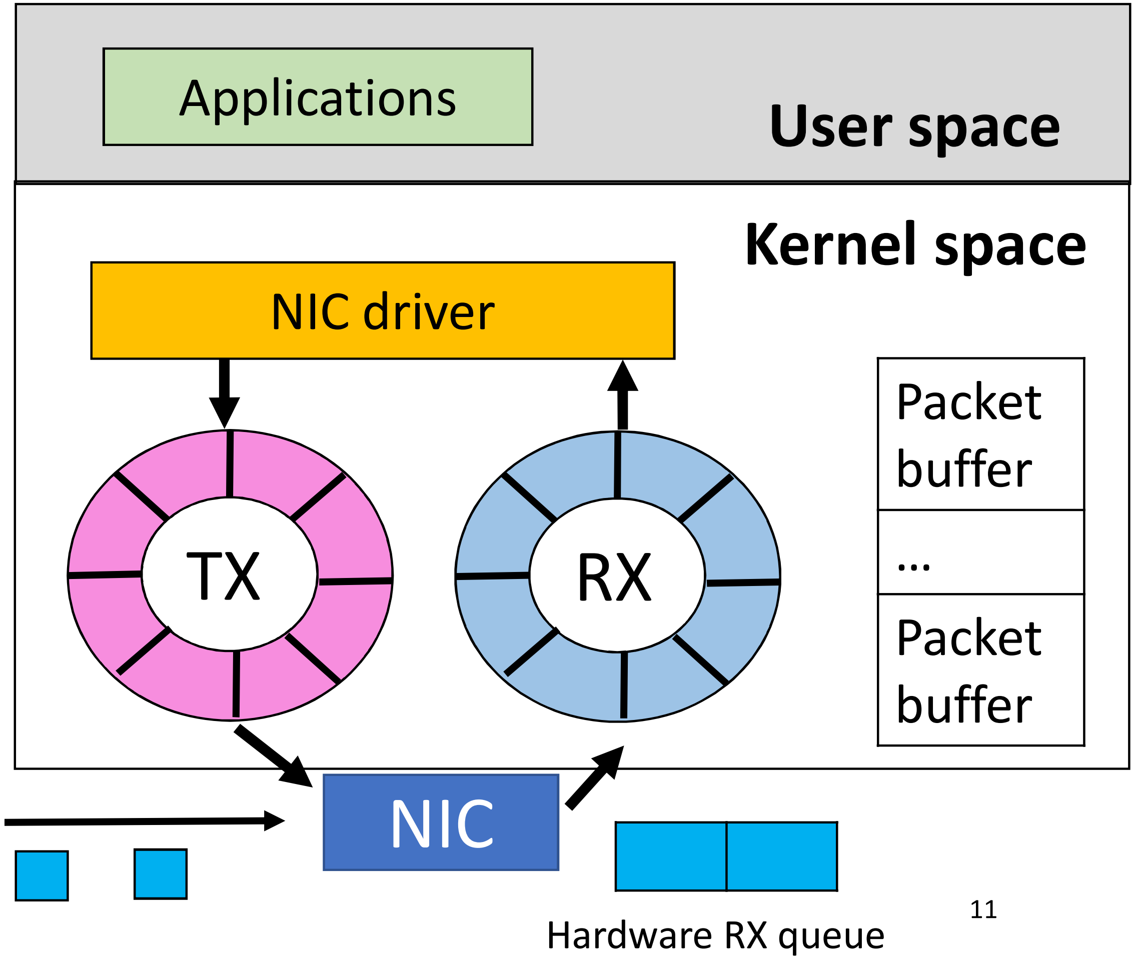
Inside a packet

- **The packet through TCP socket**
- **Frame checksum sequence (FCS):**
  - Detects any in-transit corruption of data



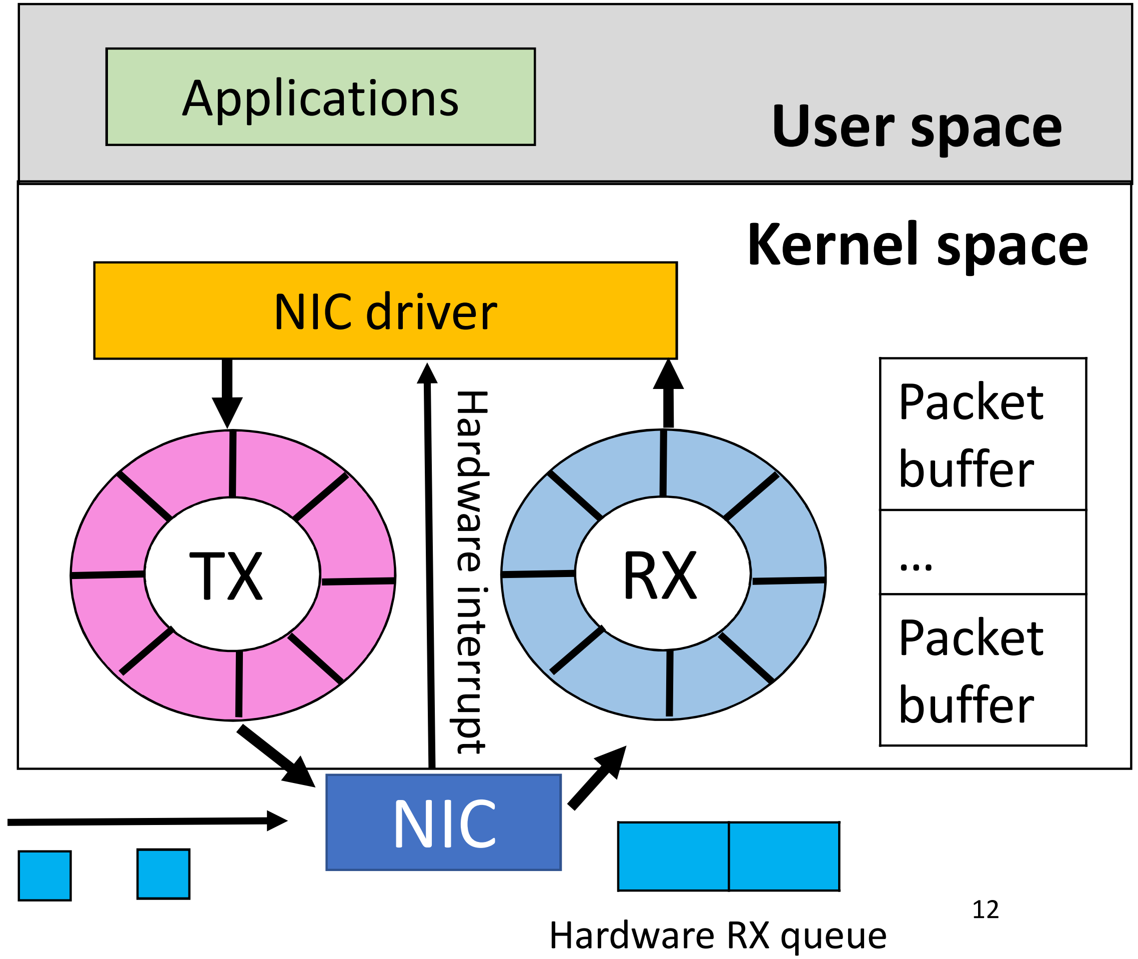
RX path: Packet arrives at the destination NIC

- **NIC receives packets**
  - Match destination MAC address
  - Verify ethernet checksum
- **TX/RX ring**
  - Circular queue
  - Shared between NIC and NIC driver
  - Content
    - Length + packet buffer pointer



RX path: Packet arrives at the destination NIC

- **NIC accepts packets**
  - DMA the packet to RX ring buffer
  - NIC triggers an interrupt



Top-half interrupt processing

CPU interrupts the executed process

Switch from user space to kernel space

Top-half interrupt processing

1. Lookup IDT (interrupt descriptor table)
2. Call ISR (interrupt service routine)
   a. Acknowledge the interrupt
   b. Schedule bottom-half processing
3. Switch back to user space

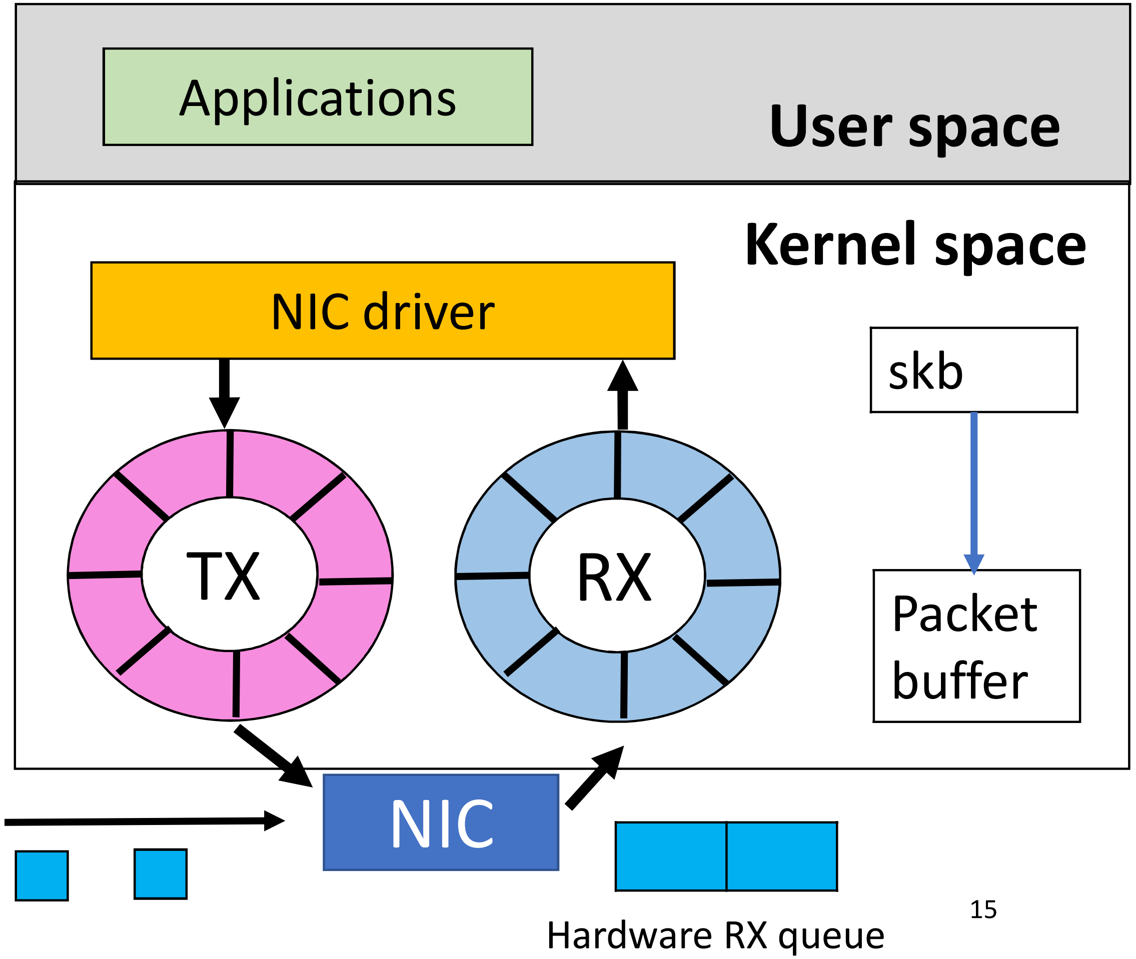
RX
Application
Transport (L4)
Network (L3)
Data link (L2)
NIC driver
NIC hardware

Bottom-half processing

- **CPU** initiates the **bottom-half** when it is free (**soft-irq**)
- Switch from user space to kernel space
- **Driver allocates an sk-buff (skb) dynamically**
- **Sk-buf**
  - In-memory data structure that contains packet metadata
  - Pointers to packet headers and payload
  - Packet related information

Bottom-half processing

- **NIC driver**
  - Handles all packets in the packet buffer
  - **Driver allocates sk-buff**
  - **Update sk-buf** with packet metadata
  - **Remove the Ethernet header**
  - **Pass sk-buff** to the network stack
  - Call L3 protocol handler



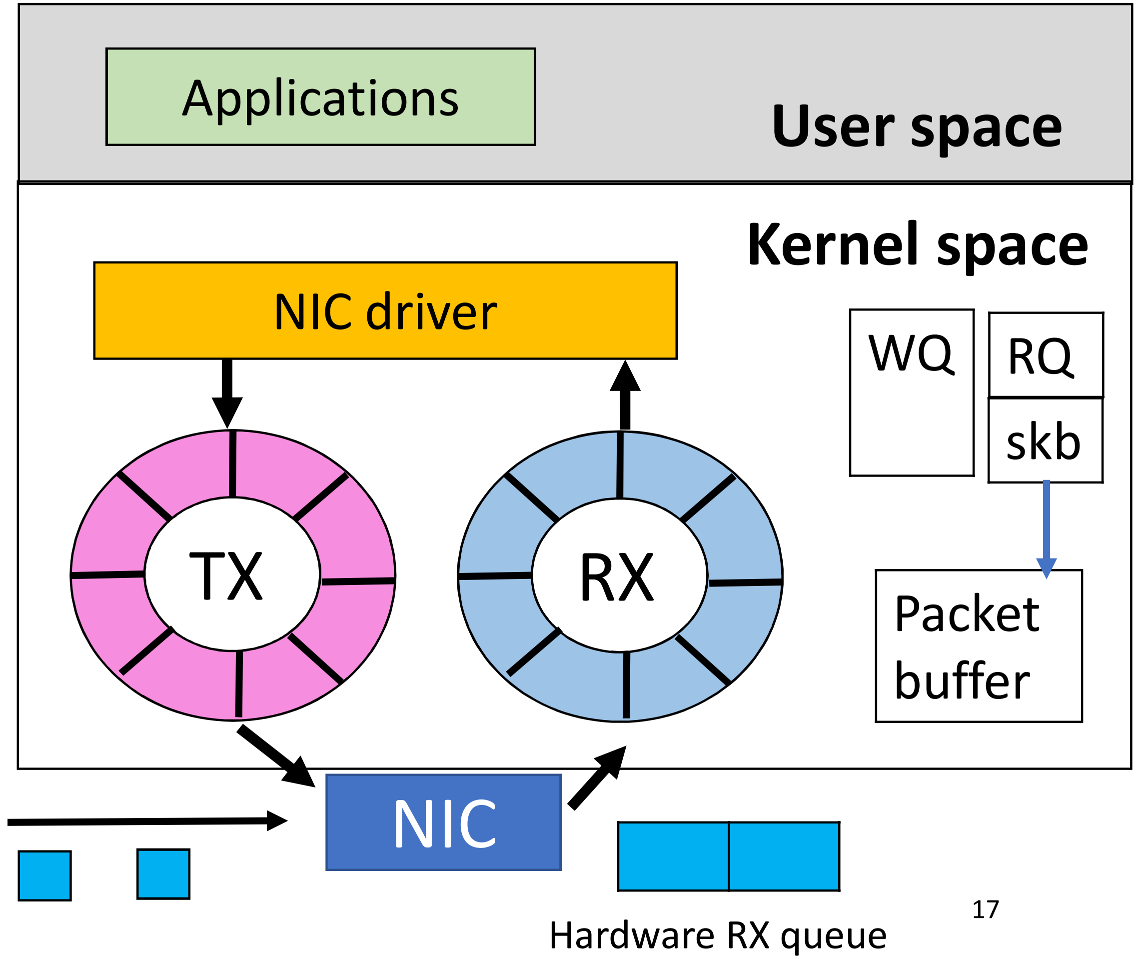
L3 processing

- **L3 common processing**
  - Match destination IP/socket
  - Verify checksum
  - Remove header
- **L3-specific processing**
  - Route lookup
  - Combine fragmented packets
  - Call L4 protocol handler

RX
Application
Transport (L4)
Network (L3)
Data link (L2)
NIC driver
NIC hardware

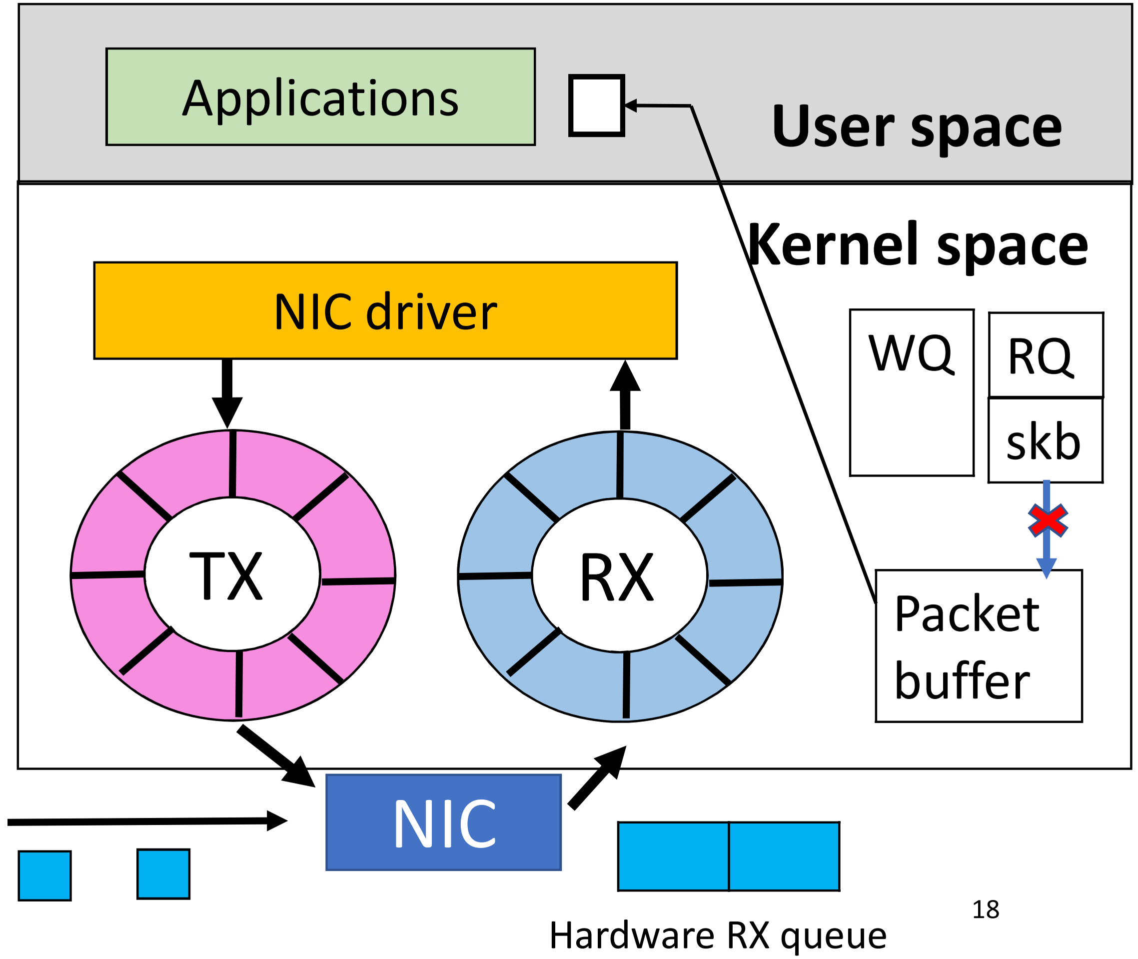
L4 processing

- **L4 specific processing**
  - Handle TCP state machine
  - Enqueue to socket read queue
  - Signal the socket



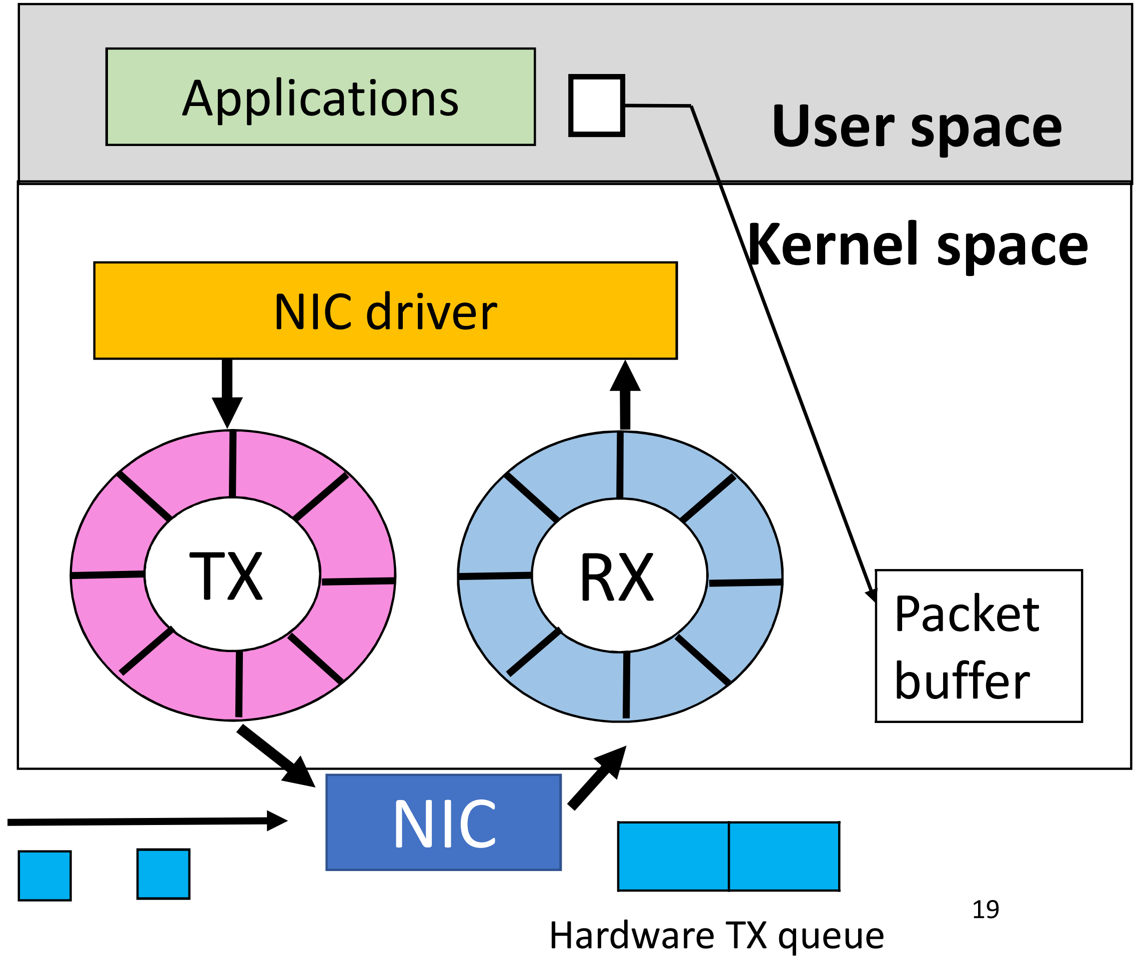
Application Layer processing

- **In the socket read**
  - Switch from user space to kernel space
  - Dequeue packet from socket receive queue (RQ)
  - Copy packet to application buffer (user space)
  - Release sk-buff
  - Return back to the application



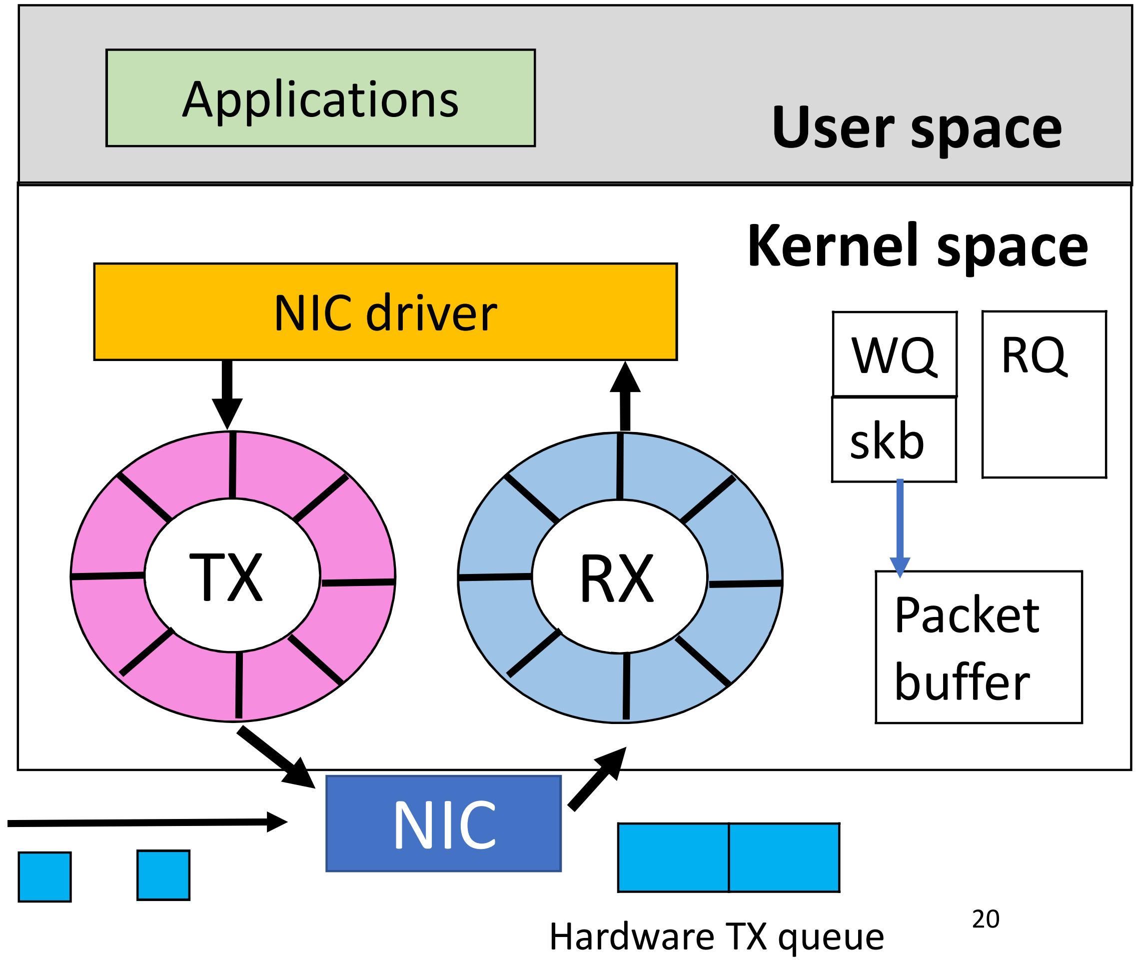
Transmit an application packet

- In the socket writes
  - Switch from user space to kernel space
  - Writes the packet to the kernel buffer
  - Calls socket's send function (sendmsg)



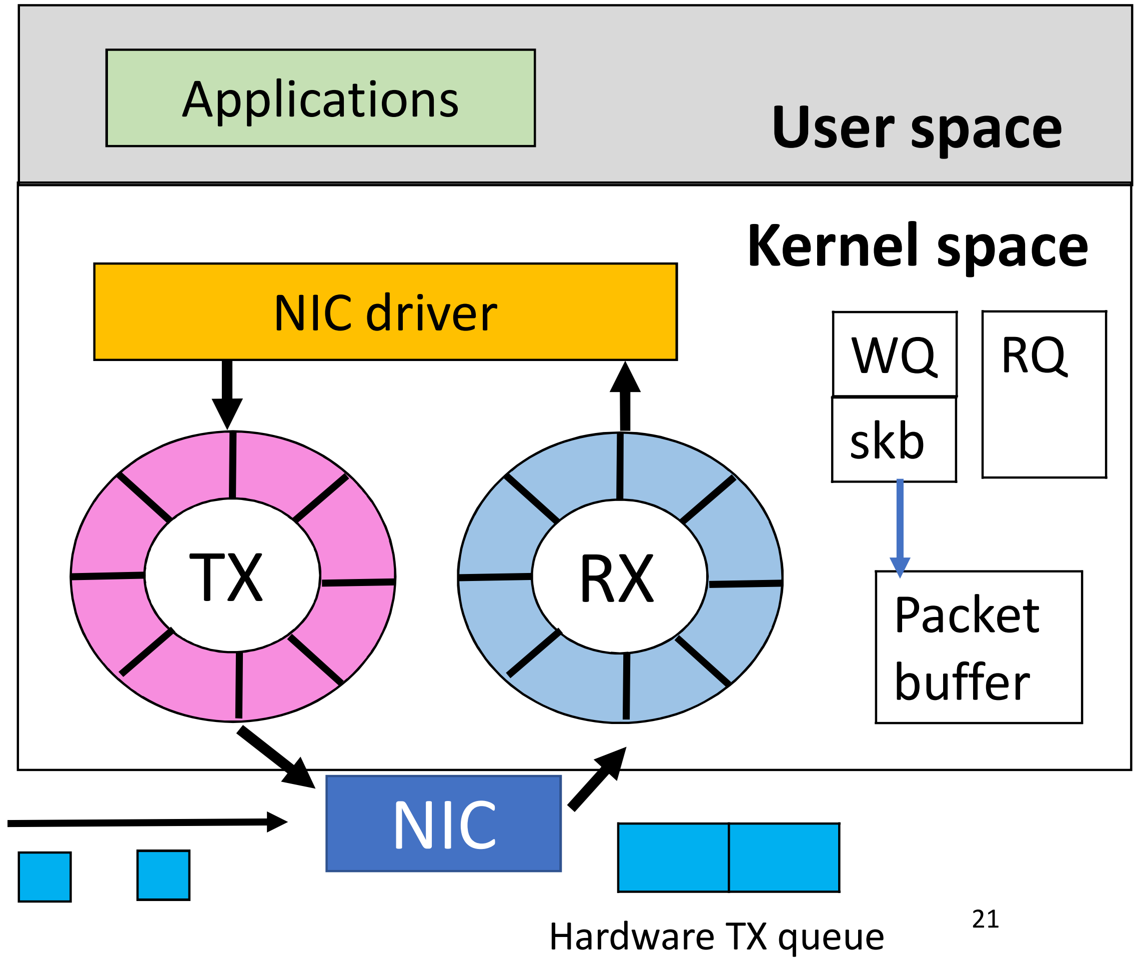
L4 processing in transmit packets

- **L4-specific processing**
  - Allocate sk-buff
  - Enqueue sk-buf to socket write queue
  - Call L3 protocol handler
- **Common processing**
  - Build header
  - Add header to packet buffer
  - Update sk-buf



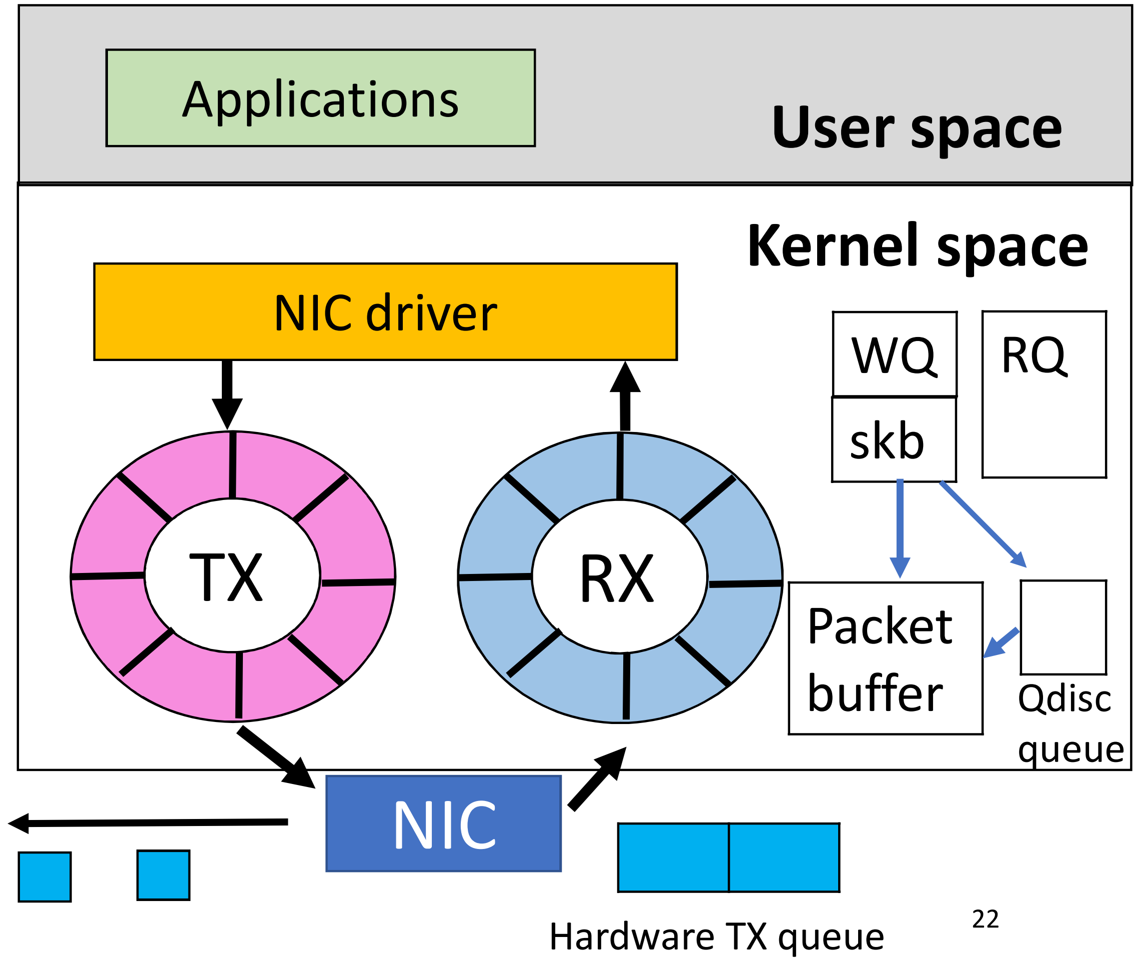
L3 processing in transmit packets

- **L3-specific processing**
  - Fragment, if needed
  - Call L2 protocol handler



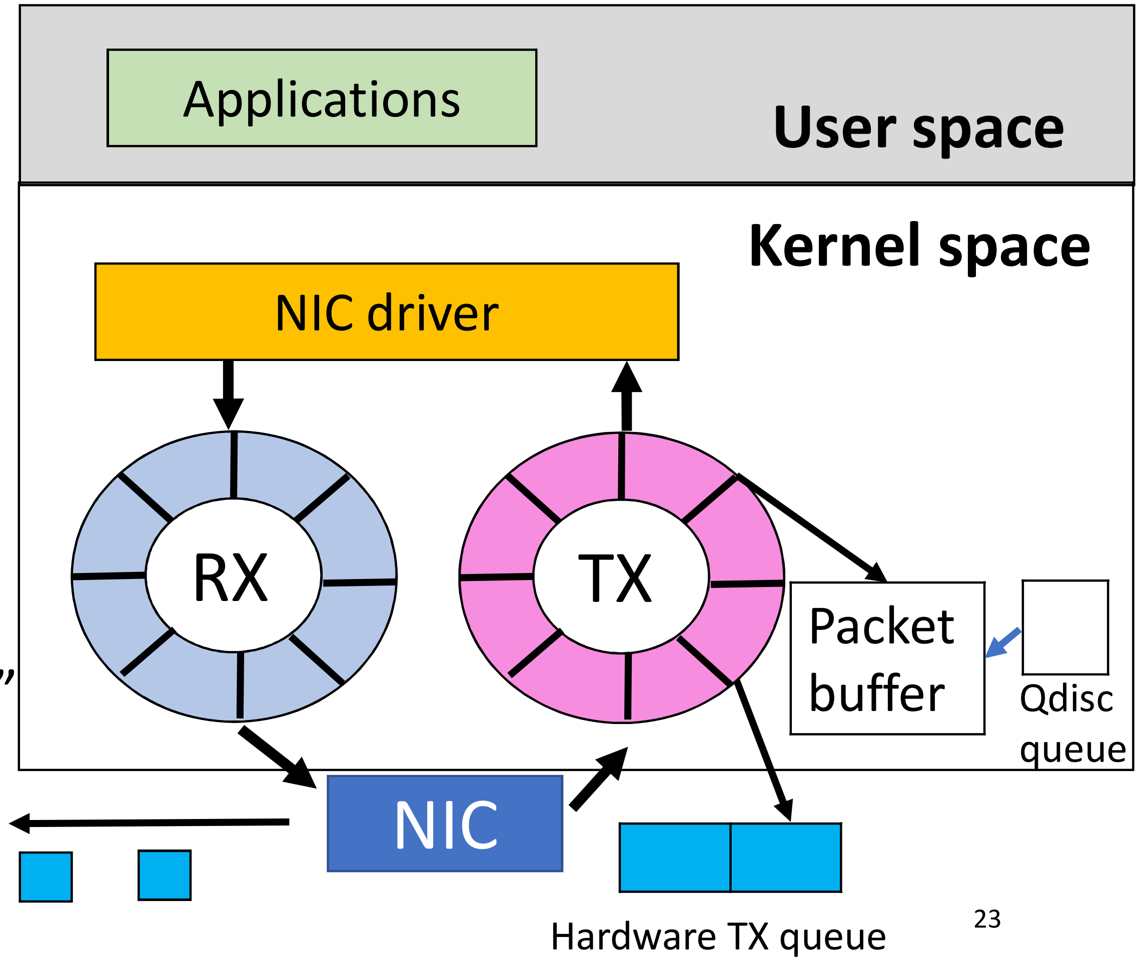
L2 processing in transmit packets

- **Enqueue packet to queue discipline (qdisc)**
  - Hold packets in a queue
  - FIFO, priority scheduling policy
- **Qdisc**
  - Dequeue sk-buff (if NIC has free buffers)
  - Calculate TCP/IP checksum
  - Call NIC driver's send function

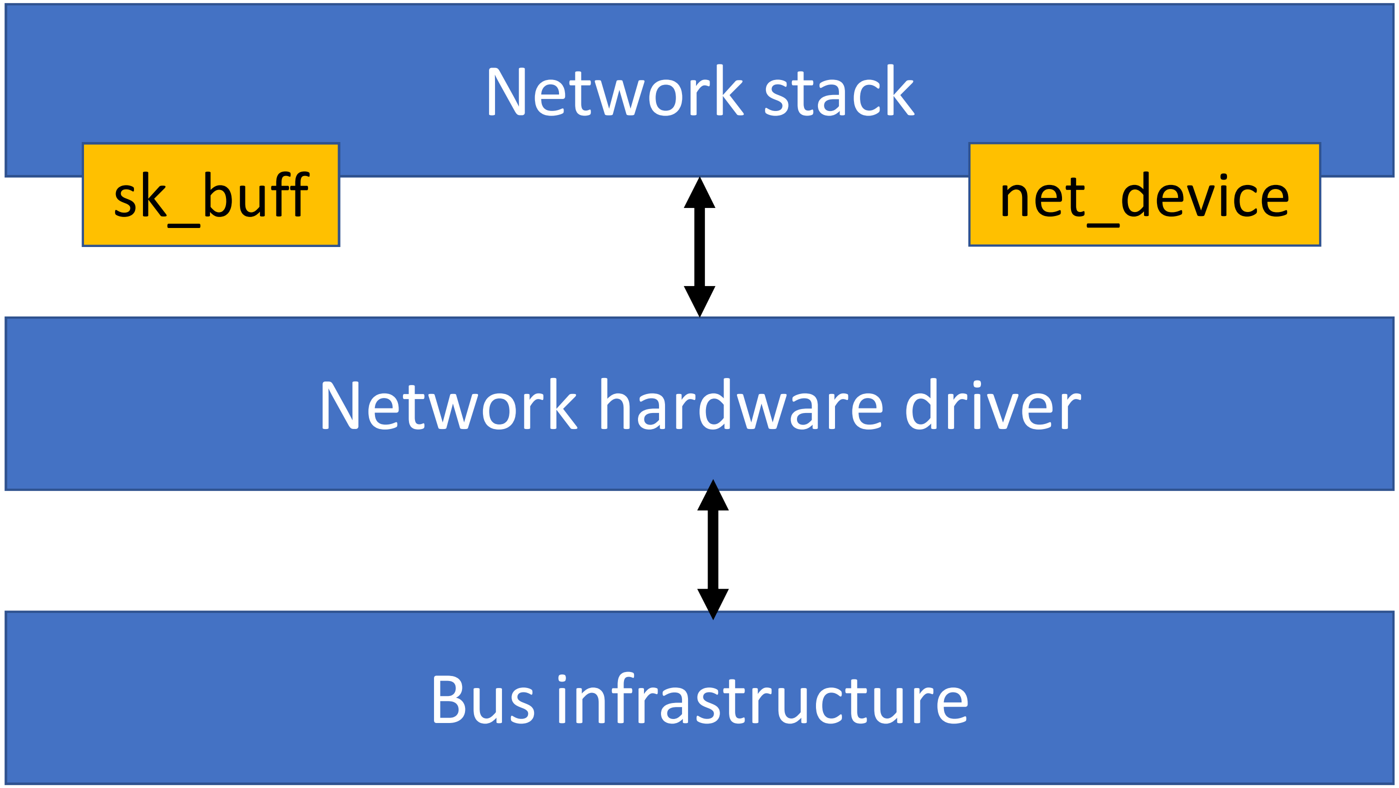


NIC processing

- **NIC driver**
  - If hardware TX queue full
    - Stop qdisc queue
  - Else
    - Map packet data to DMA
    - Tells NIC to send the packet
- **NIC**
  - Calculate FCS
  - Send packet to the wire
  - Sends an interrupt “packet is sent” (kernel space to user space)
  - Driver frees the sk-buf, start the qdisc queue



NIC packet processing flow



sk_buff

- **struct sk_buff**
  - Represents **a network packet**
  - Support encapsulation/decapsulation of data through the protocol layers
  - **Maintain data structures**
    - **Head**: the start of the packet
    - **Data**: the start of the packet payload
    - **Tail:** the end of the packet payload
    - **End:** the end of the packet
    - **Len:** the amount of data in a packet

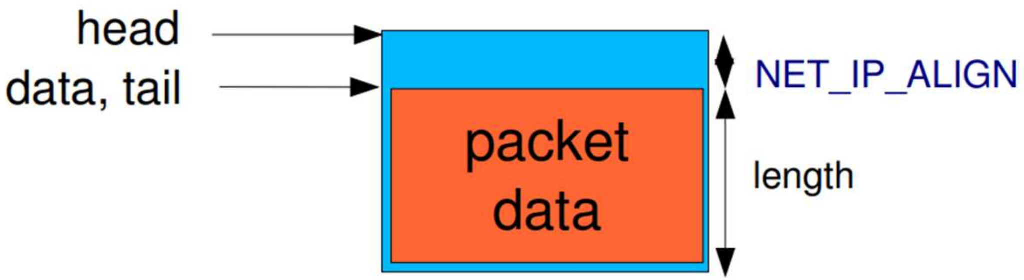
Allocating a skb

- **Allocate an SKB**
  - **dev_alloc_skb()**
  - Called from an **interrupt handler**
  - On Ethernet, the size allocated is the length of packet + 2
  - So, the IP header is word-aligned (the Ethernet header is 14 bytes)
    - skb = dev_alloc_skb (length + NET_IP_ALIGN)

Copy the received data

- Copy the packet payload from the DMA buffer to the skb
  - static inline vokd **skb_copy_to_linear_data** (struct sk_buff *skb, const void *from, const unsigned int len);
  - static inline vokd **skb_copy_to_linear_data_offset** (struct sk_buff *skb, const void *from, const unsigned int len);

```
skb_copy_to_linear_data(skb, dmabuffer,
                        length);
```



https://bootlin.com/doc/legacy/network-drivers/network-drivers.pdf

struct net_device

- **struct net_device**
  - **Represents a single network interface**
  - Allocation with **alloc_etherdev()**
  - Registration with **register_netdev()**
  - Unregistration with **unregister_netdev()**
  - Liberation with **free_netdev()**

struct net_device_ops

- **Methods of a network interface**
  - **ndo_open(),** called when the network interface is up
  - **ndo_close(),** called when the network interface is down
  - **ndo_start_xmit(),** start the transmission of a packet
  - **ndo_get_stats(),** gets statistics
  - **ndo_do_ioctl(),** implement device specific operations
  - **ndo_set_rx_mode(),** select promiscuous, multicast, etc.
  - **ndo_set_mac_address(),** set the MAC address

Transmission

- The **ndo_start_xmit()** starts the transmission of a packet
  - The driver **sets up DMA buffers**
  - The driver can also **stop the queue with netif_stop_queue()** depending on the number of free DMA buffers available
- When the packet has been sent, an interrupt is raised, the driver will do
  - **Acknowledging the interrupt**
  - **Freeing the used DMA buffers**
  - **Free the skb with dev_kfree_skb_irq()**
  - If the queue was stopped, start it again

Reception

- **Reception is notified by an interrupt**. The interrupt handler should
  - **Allocate an skb** with dev_alloc_skb()
  - **Reserve the 2 bytes offset** with skb_reserve ()
  - **Copy the packet data** from the DMA buffers to the skb through skb_copy_to_linear_data () or skb_copy_to_linear_data_offset ()
  - **Update the skb pointers** with skb_put()
  - **Update the skb->protocol field** with eth_type_trans(skb, netdevice)
  - **Give the skb to the kernel network stack** with netif_rx(skb)

Reception: NAPI mode

- The NAPI mode allows to switch to polled mode when the interrupt rate is too high
  - Add a struct napi_struct in the network interface private structure
  - At driver initialization, register the NAPI poll operation
  - netif_napi_add (dev, &bp->napi, macb_poll, 64)
    - **dev**: the network interface
    - **&bp->napi**: the struct napi_struct
    - **macb_poll** is the NAPI poll operation
    - **64 is the weight** that represents the importance of the network interface

Reception: NAPI mode

- **When a packet has been received, interrupt handler will do**

```
if (napi_schedule_prep (&bp->napi)) {
        /*Disable reception interrupts*/
        __napi_schedule (&bp->napi); }
```

  - The kernel will **call our poll() operation** regularly (macb_poll())
  - **Push packets to the network stack** using **netif_receive_skb()** when receive at most budget packets
  - **Switch back to interrupt mode** using **napi_complete ()** if less than budget packets have been received, re-enable interrupts
  - **Must return the number of packets received**

Communication with the PHY

- **Ethernet controller handles layer 2 (MAC**) communication
- An **external PHY is responsible for layer 1** communication
- The MAC and PHY are connected using a **MII or RMII interface**
  - **MII = Media independent interface**
  - **RMII = Reduced media independent interface**
- This interface contains **two wires used for the MDIO (management data input/output) bus**
  - Ethernet driver needs to communicate with the PHY to get information about the link (up, down, speed, full or half duplex)

PHY in the kernel

- **The kernel provides a framework that**
  - Exposes an API to communicate with the PHY
  - Allows to implement PHY drivers
  - Implements a basic generic PHY driver that works with all PHY
  - See ‘drivers/net/phy’

Connection to the PHY

- The ‘**mdiobus_register()’** function
  - Filled the mii_bus->phy_map[] array with struct phy_device * pointer
  - The appropriate PHY must be selected
  - Connecting to the PHY allows to register a callback that will be called when the link changes

```
int phy_connect_direct (
        struct net_device *dev,
        void (*handler) (struct net_device *),
        u32 flags,
        phy_interface_t interface
)
```

Start and stop the PHY

- To make poll regularly in the PHY hardware, one must start
  - **phy_start** (phydev)
- When the network is stopped, the PHY must also be stopped
  - **phy_stop** (phydev)

Suspend and resume the PHY

- **The suspend () operation**
  - Call netif_device_detach ()
  - Do the hardware-dependent operations to suspend the devices (like disable the clocks)
- **The resume() operation**
  - Call netif_device_attach()
  - Do the hardware-dependent operations (like enable the clocks)

ethtool

- **Ethtool is a userspace tool**
  - Allows to query low-level information from an Ethernet interface and to modify its configuration
- **On the kernel side**
  - A struct ethtool_ops can be declared and connected to the struct net_device
  - These operations can be implemented using the PHY interface (phy_ethtool_gset (), phy_ethtool_sset()) or using generic operations (ethtool_op_get_link())

Summary

- NIC driver
  - The kernel space interrupt handler controls the TX/RX packet flow
  - TX/RX ring circular queue
  - Packet buffer
- struct sk_buff
  - A network packet entry
- set_device ()