Zoltan Juhasz

User-Space Network Stacks

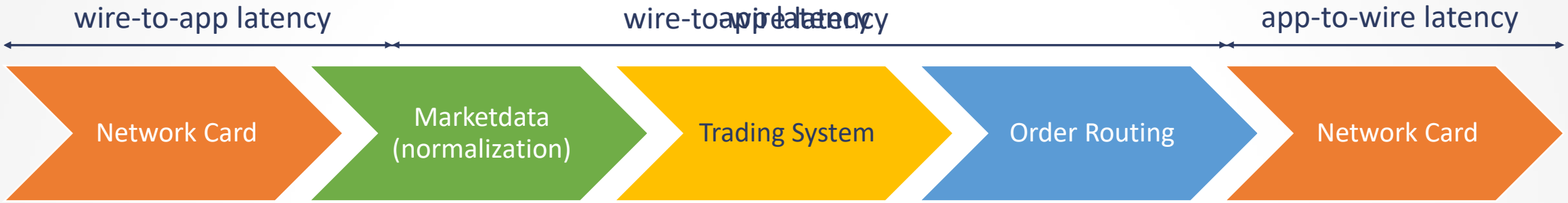
Kernel Bypass Technologies in Linux

Overview

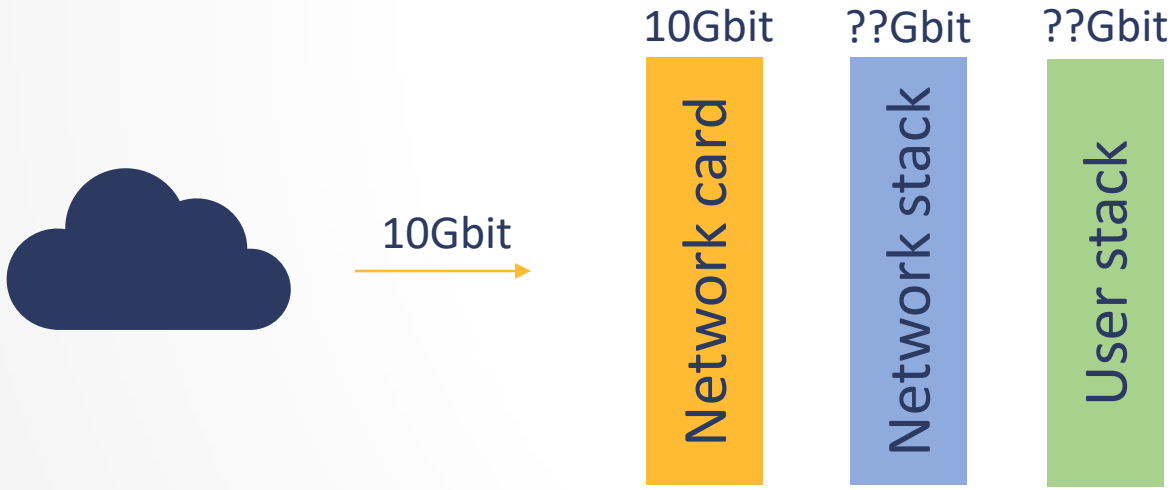
- Not a C++ talk
- Kernel-space networking
- Kernel network stack tuning
- User-space networking
  - Open source libraries
  - Transparent kernel-bypass
  - Vendor specific libraries

Latency and Throughput

What is Latency?



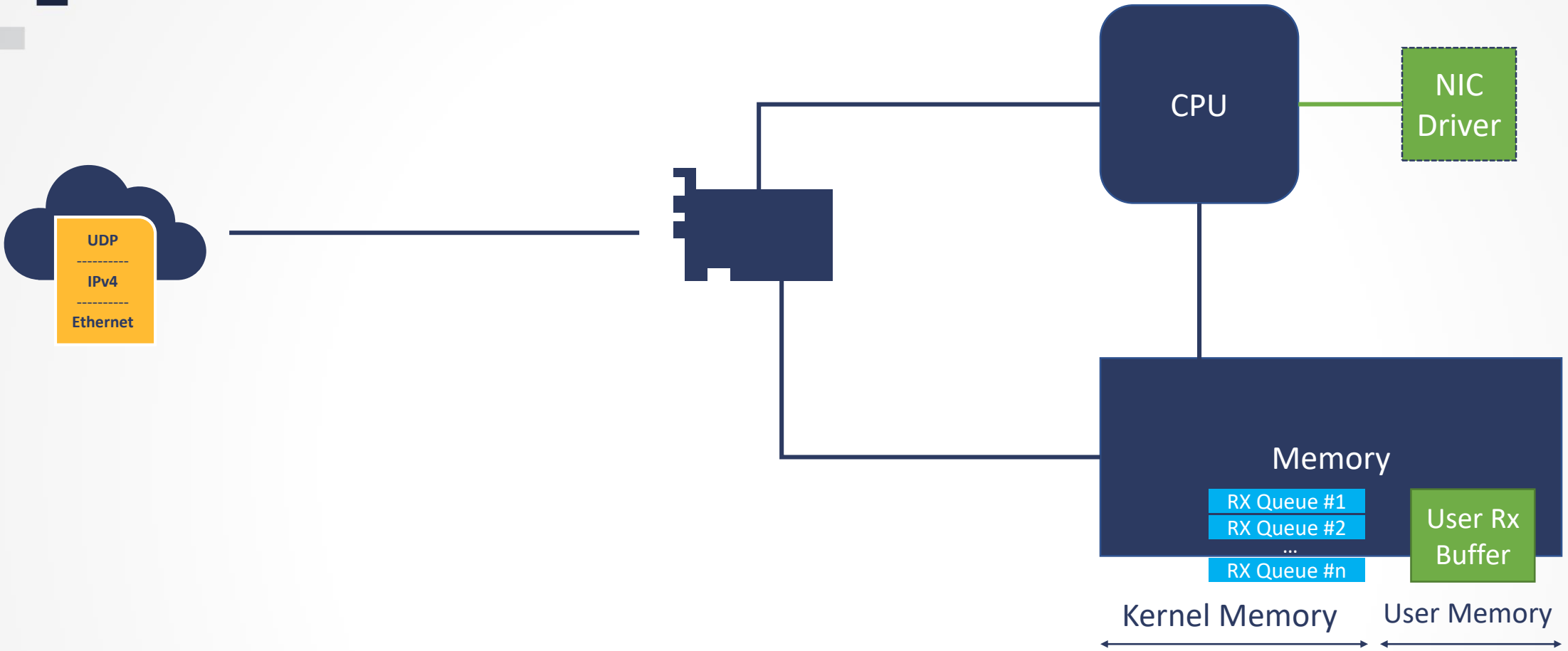
What is Throughput?



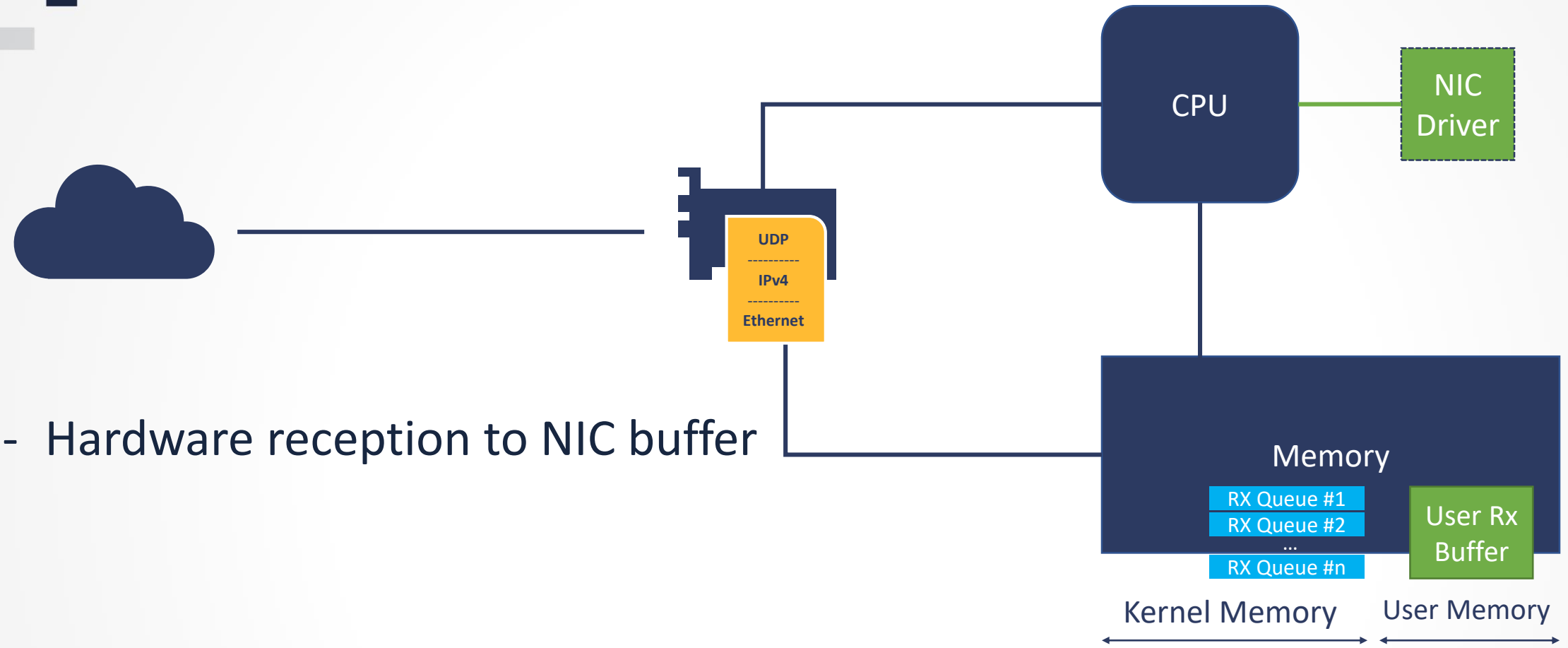
- Affected by packet size
- In general we want to handle **network card (NIC)** line-rate reliably
- Frequently you can sacrifice latency for throughput and vice versa

Linux Packet Handling

Linux Rx Packet Handling Overview

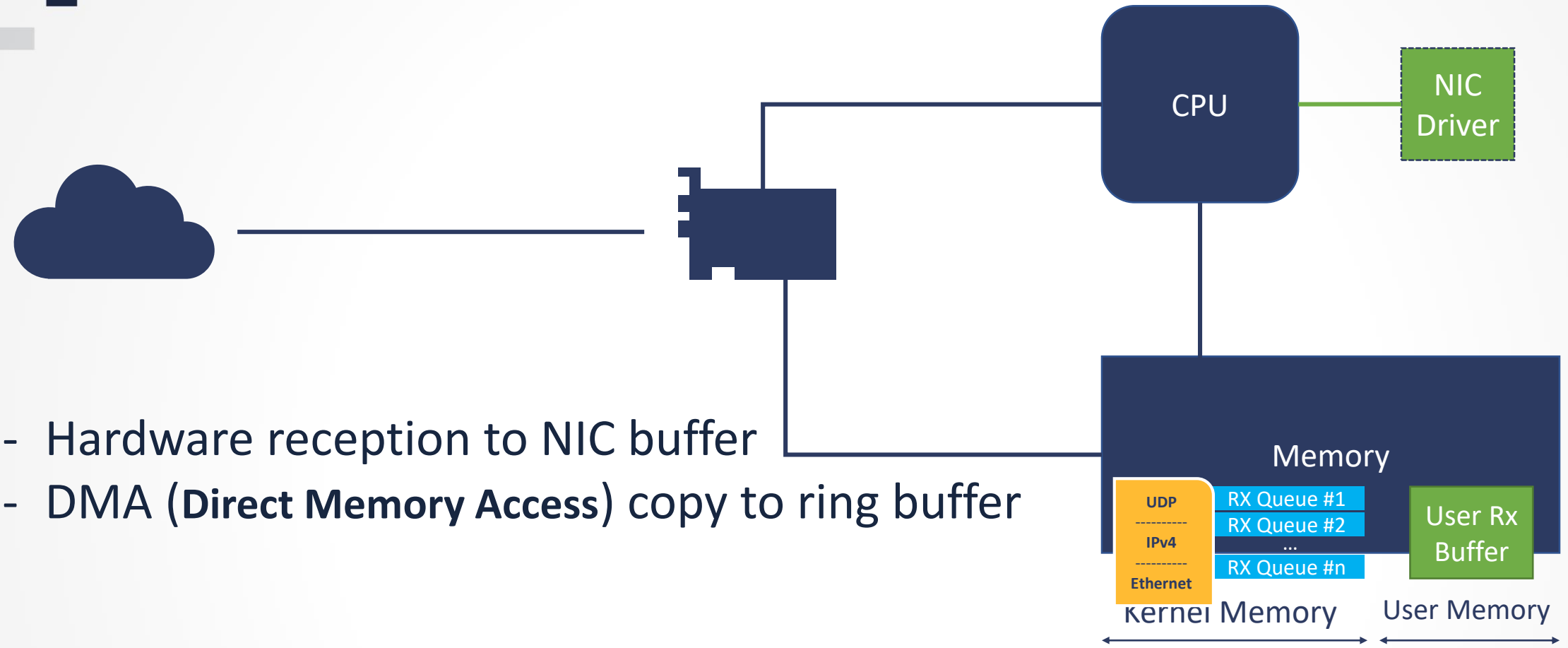


Linux Rx Packet Handling Overview



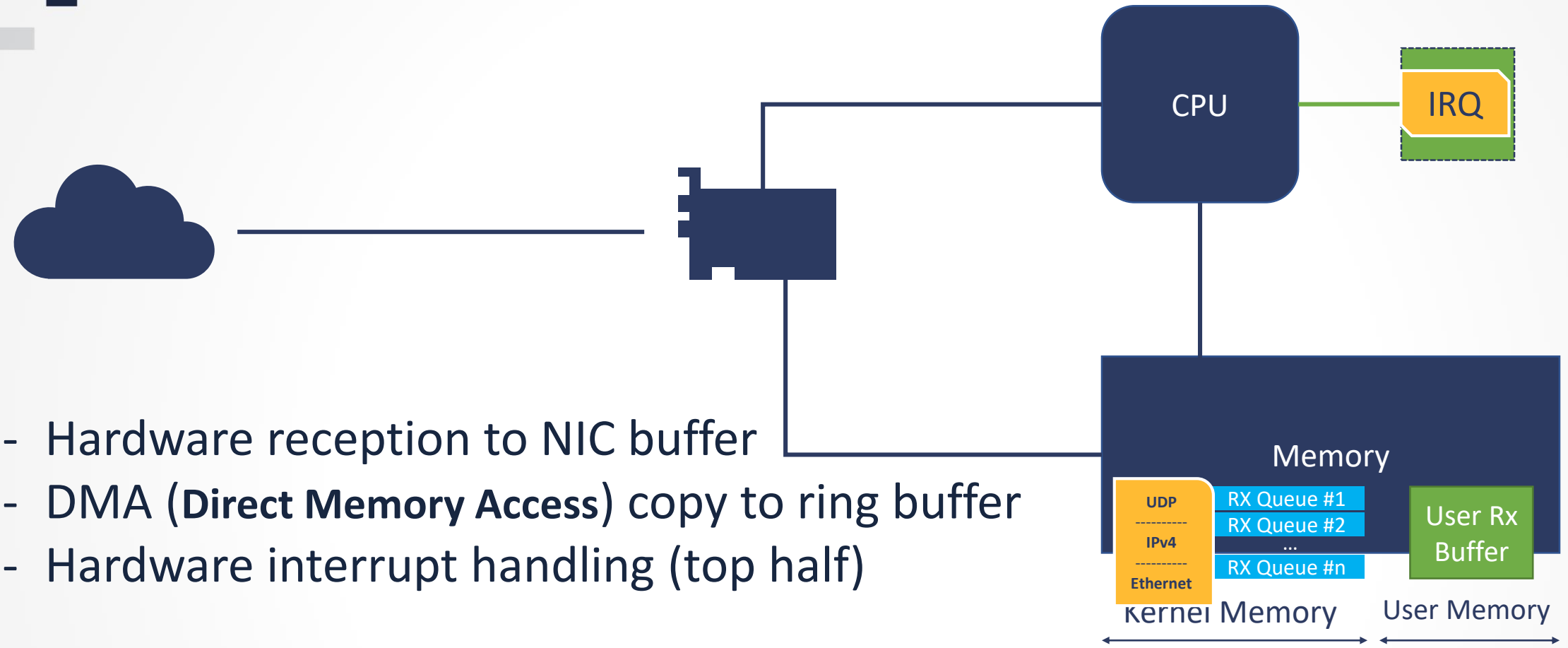
- Hardware reception to NIC buffer

Linux Rx Packet Handling Overview



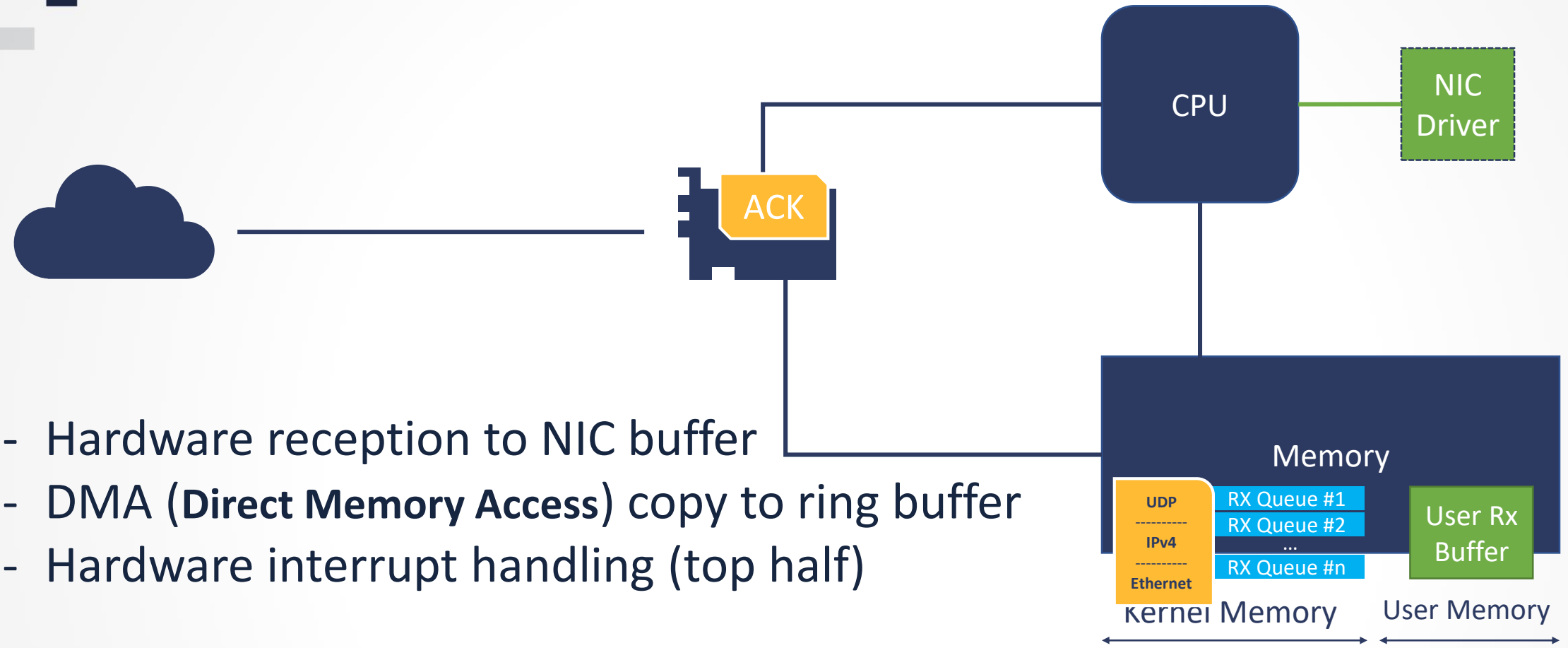
- Hardware reception to NIC buffer
- DMA (**Direct Memory Access**) copy to ring buffer

Linux Rx Packet Handling Overview



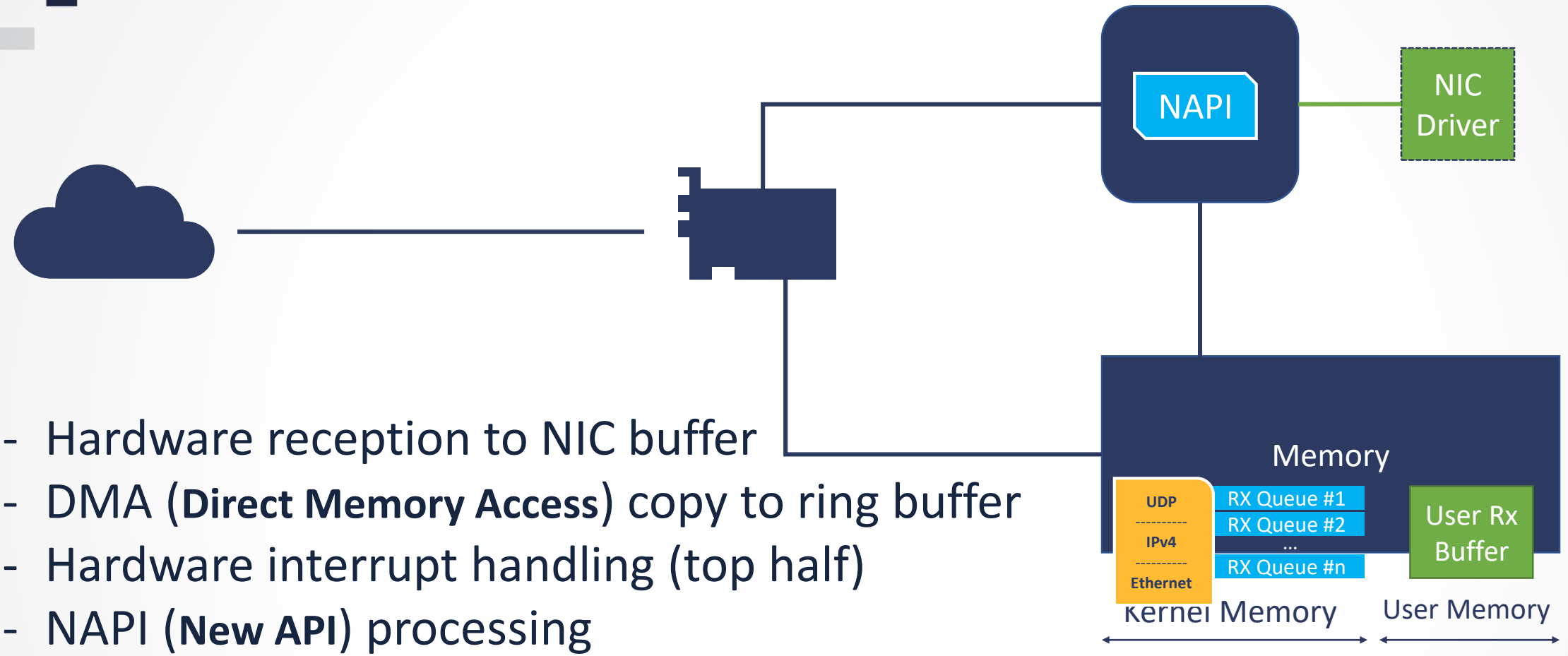
- Hardware reception to NIC buffer
- DMA (**Direct Memory Access**) copy to ring buffer
- Hardware interrupt handling (top half)

Linux Rx Packet Handling Overview



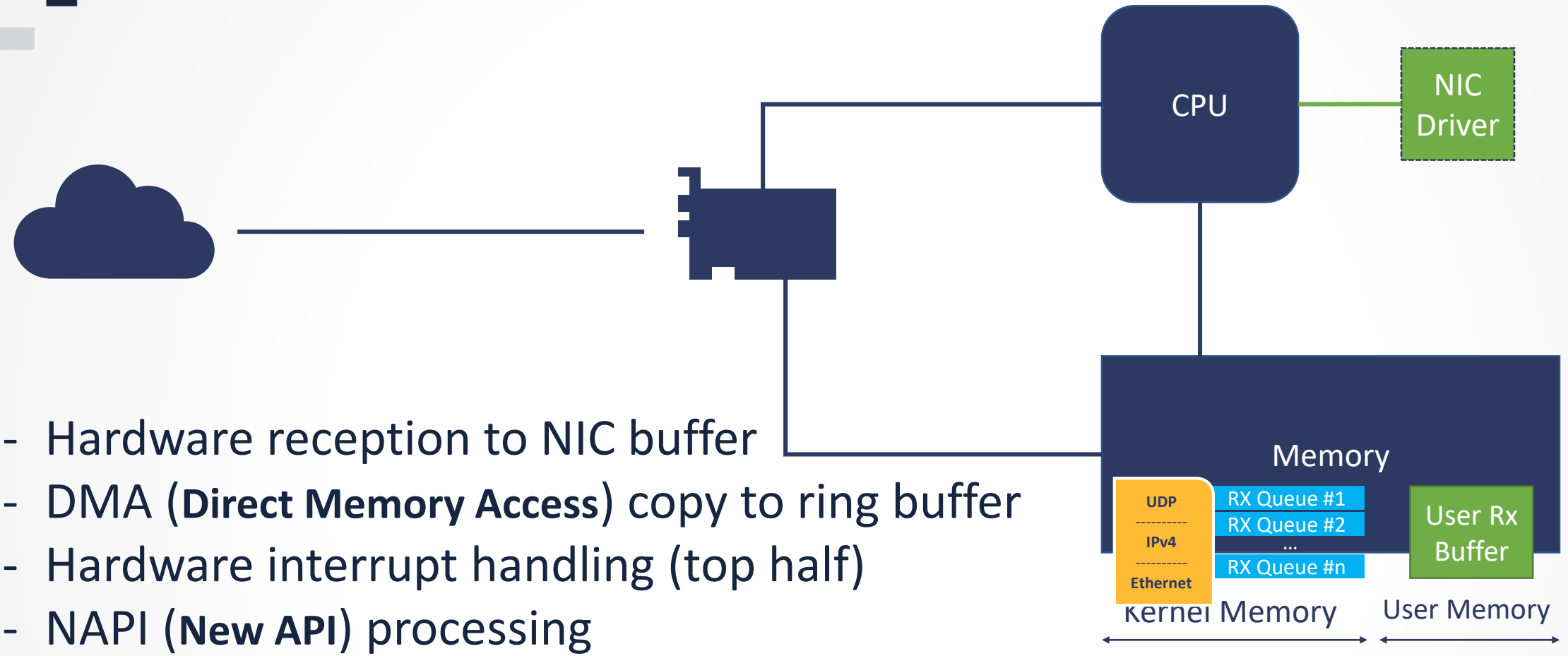
- Hardware reception to NIC buffer
- DMA (**Direct Memory Access**) copy to ring buffer
- Hardware interrupt handling (top half)

Linux Rx Packet Handling Overview



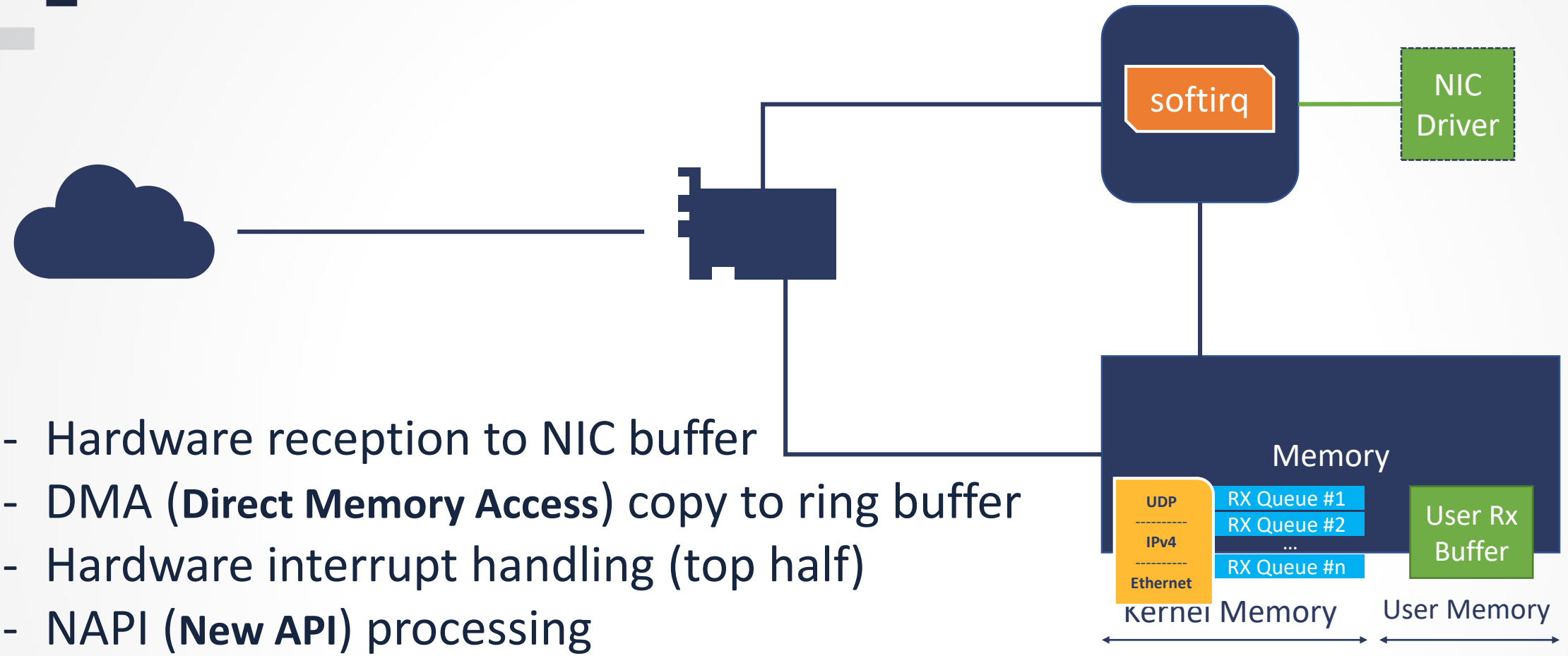
- Hardware reception to NIC buffer
- DMA (**Direct Memory Access**) copy to ring buffer
- Hardware interrupt handling (top half)
- NAPI (**New API**) processing

Linux Rx Packet Handling Overview



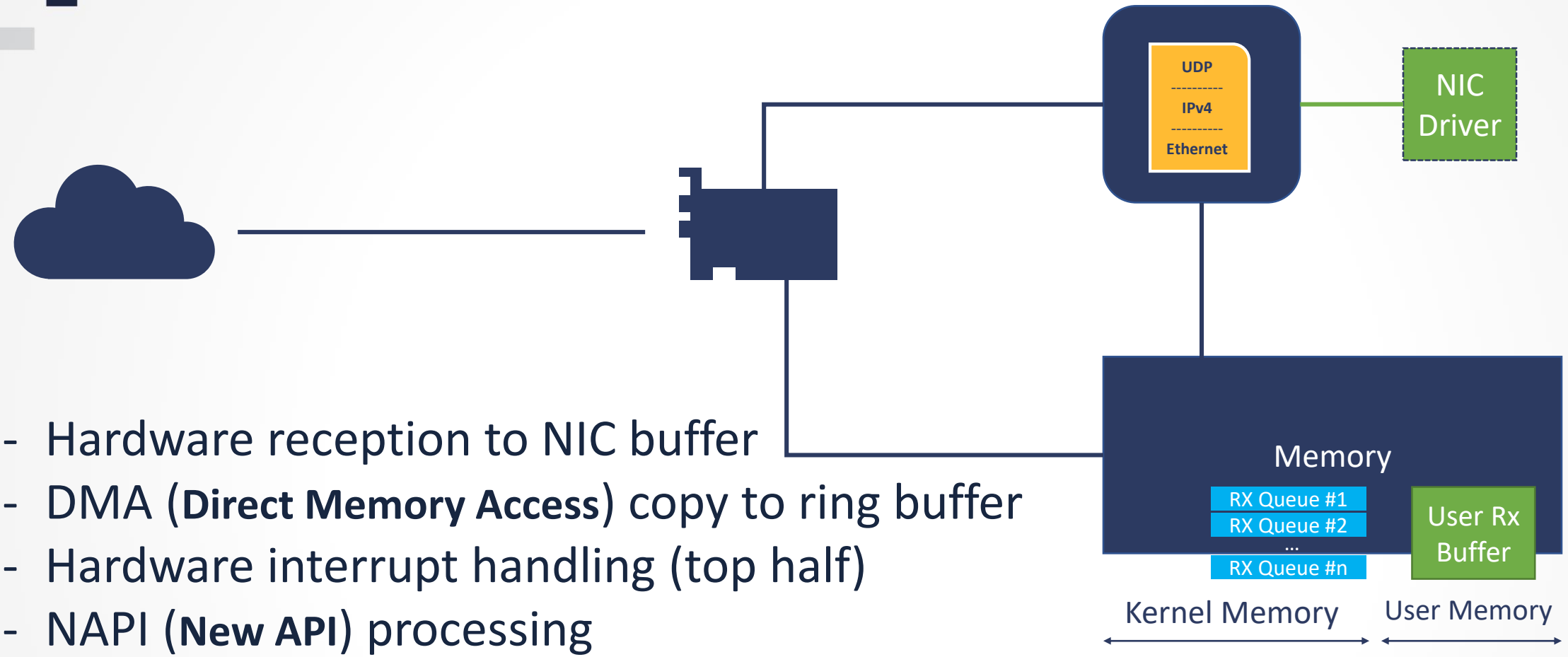
- Hardware reception to NIC buffer
- DMA (**Direct Memory Access**) copy to ring buffer
- Hardware interrupt handling (top half)
- NAPI (**New API**) processing

Linux Rx Packet Handling Overview



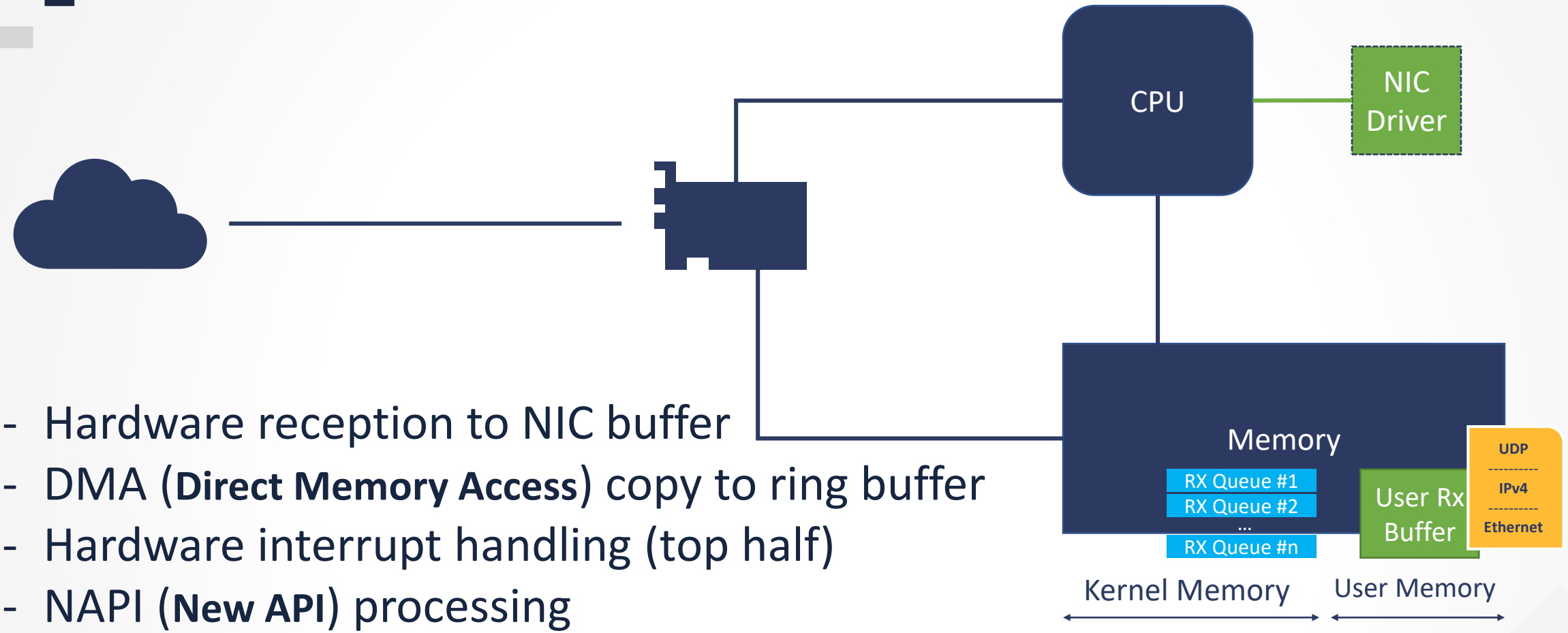
- Hardware reception to NIC buffer
- DMA (**Direct Memory Access**) copy to ring buffer
- Hardware interrupt handling (top half)
- NAPI (**New API**) processing
- Software interrupt handling (bottom half – softirq)

Linux Rx Packet Handling Overview



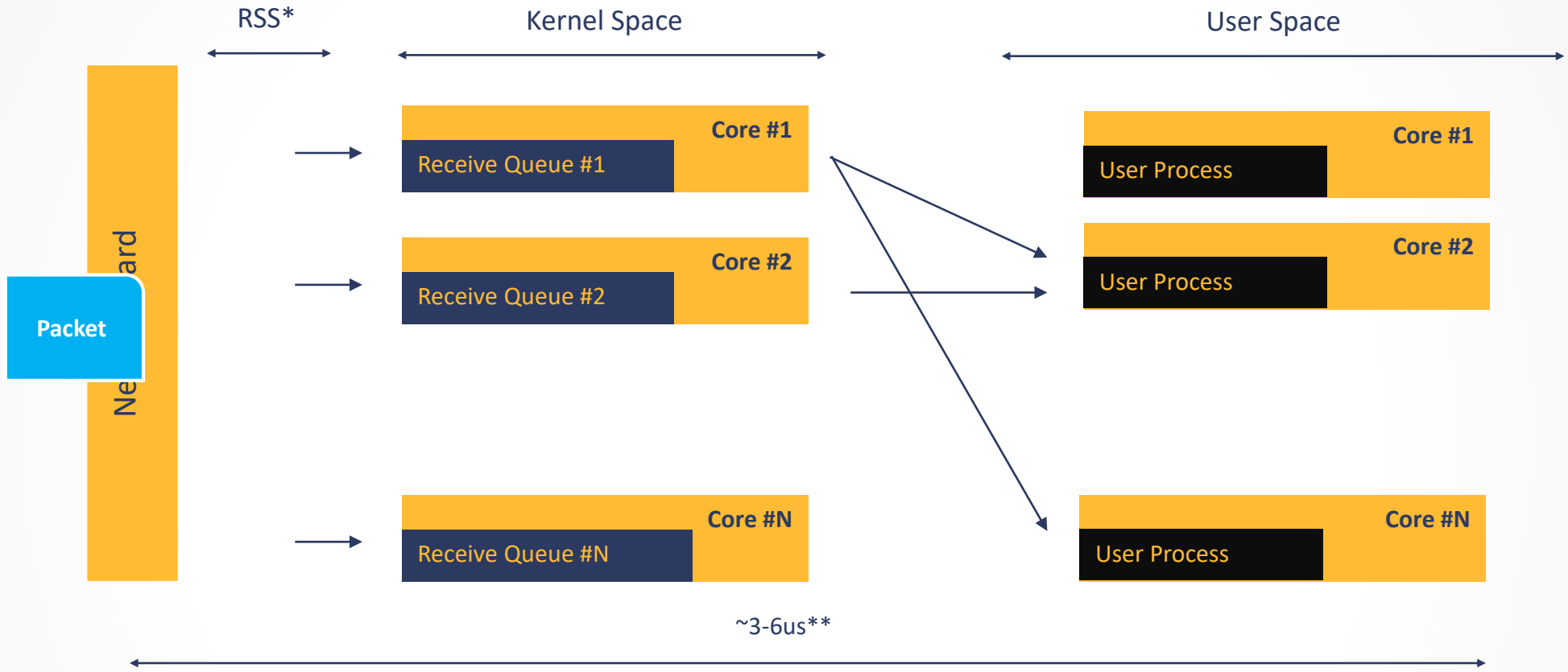
- Hardware reception to NIC buffer
- DMA (**Direct Memory Access**) copy to ring buffer
- Hardware interrupt handling (top half)
- NAPI (**New API**) processing
- Software interrupt handling (bottom half – softirq)
- Protocol processing

Linux Rx Packet Handling Overview



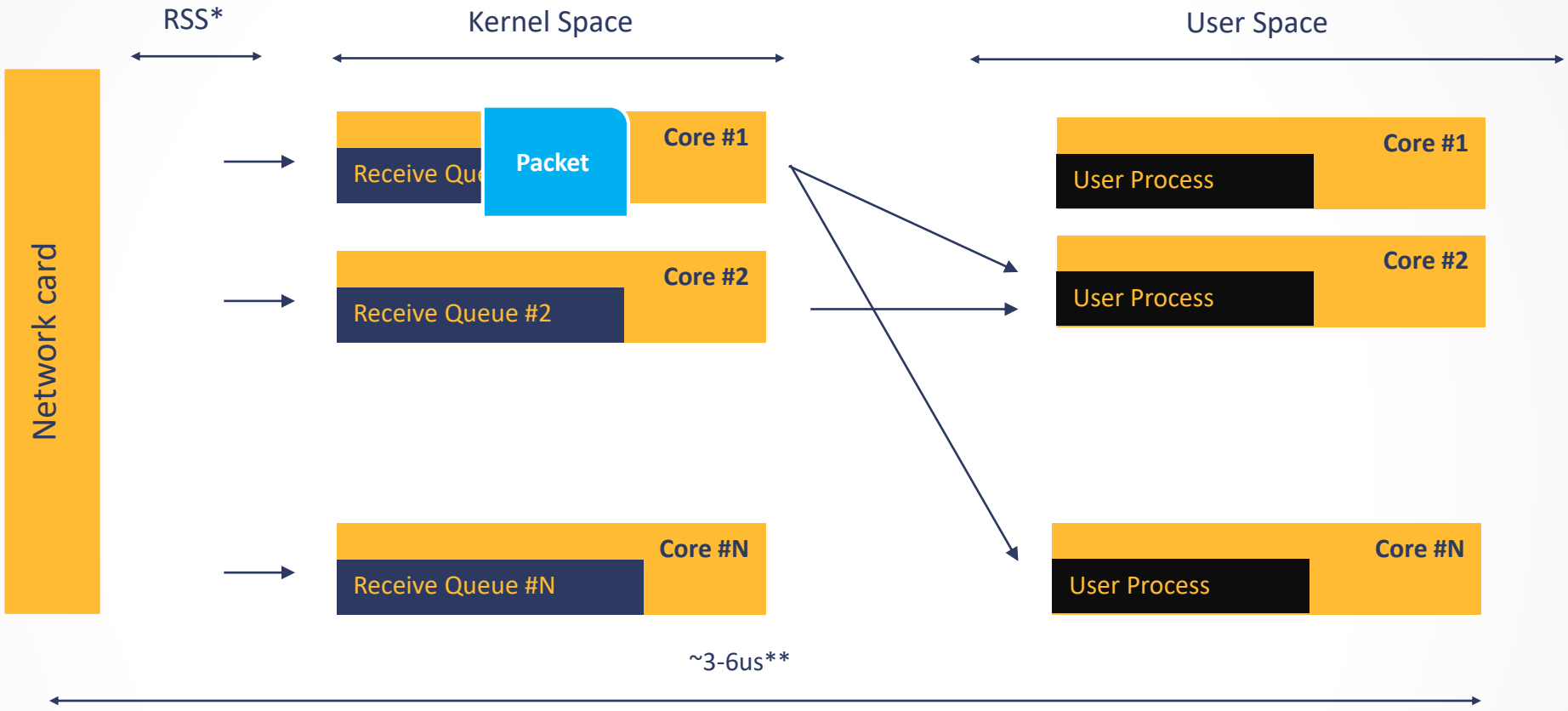
- Hardware reception to NIC buffer
- DMA (**Direct Memory Access**) copy to ring buffer
- Hardware interrupt handling (top half)
- NAPI (**New API**) processing
- Software interrupt handling (bottom half – softirq)
- Protocol processing
- Copy to user Receive (RX) buffer

Interrupts and Cores



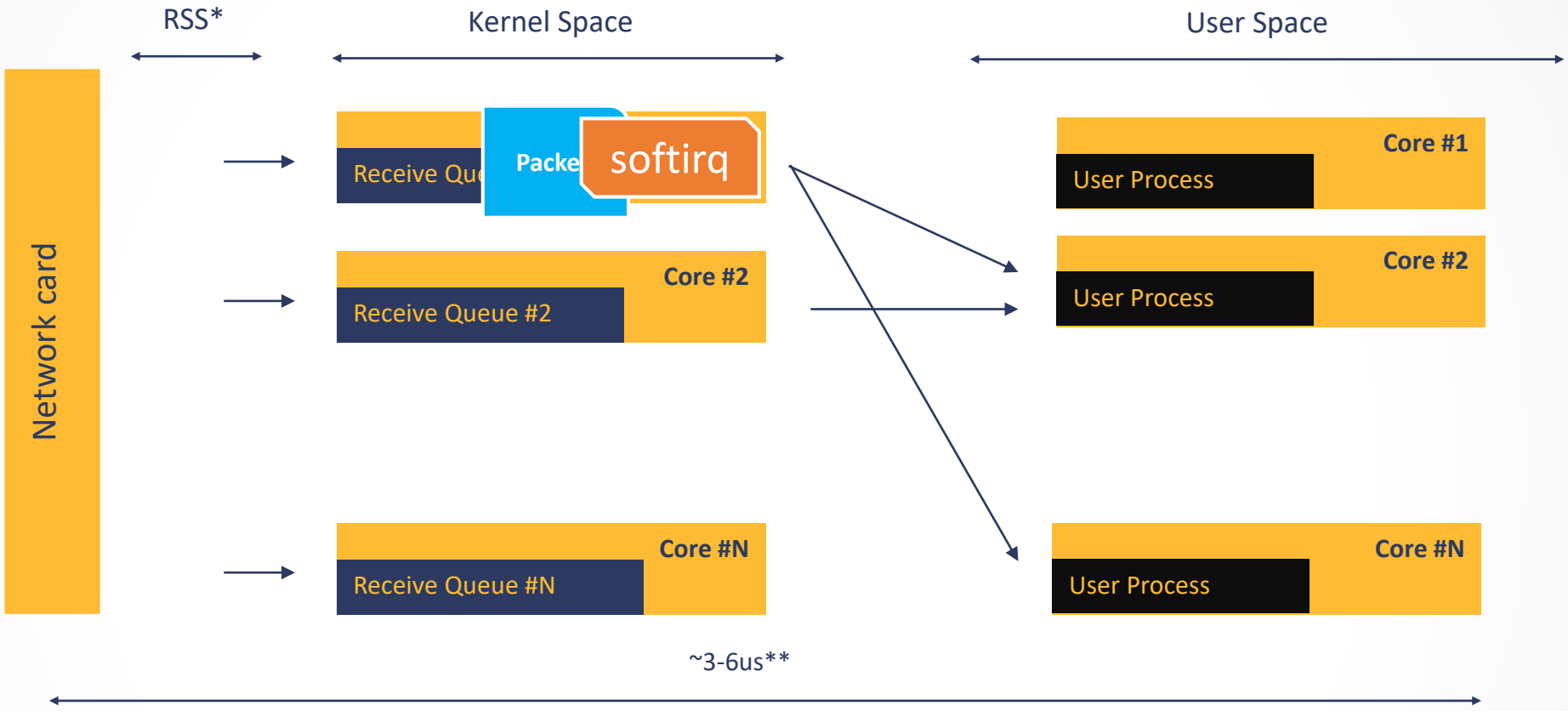
* Receive Side Scaling
** approx., depending on packet size and protocol

Interrupts and Cores



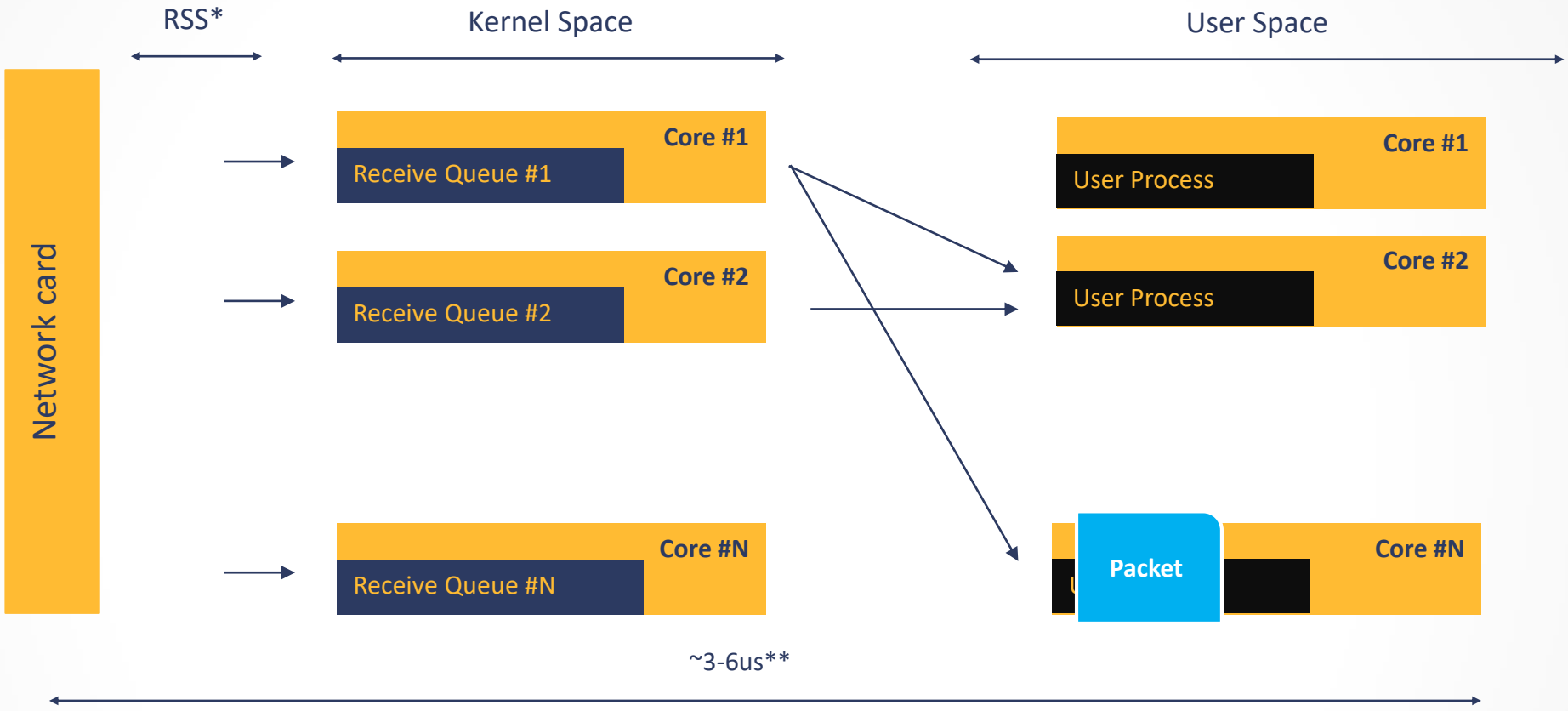
* Receive Side Scaling
** approx., depending on packet size and protocol

Interrupts and Cores



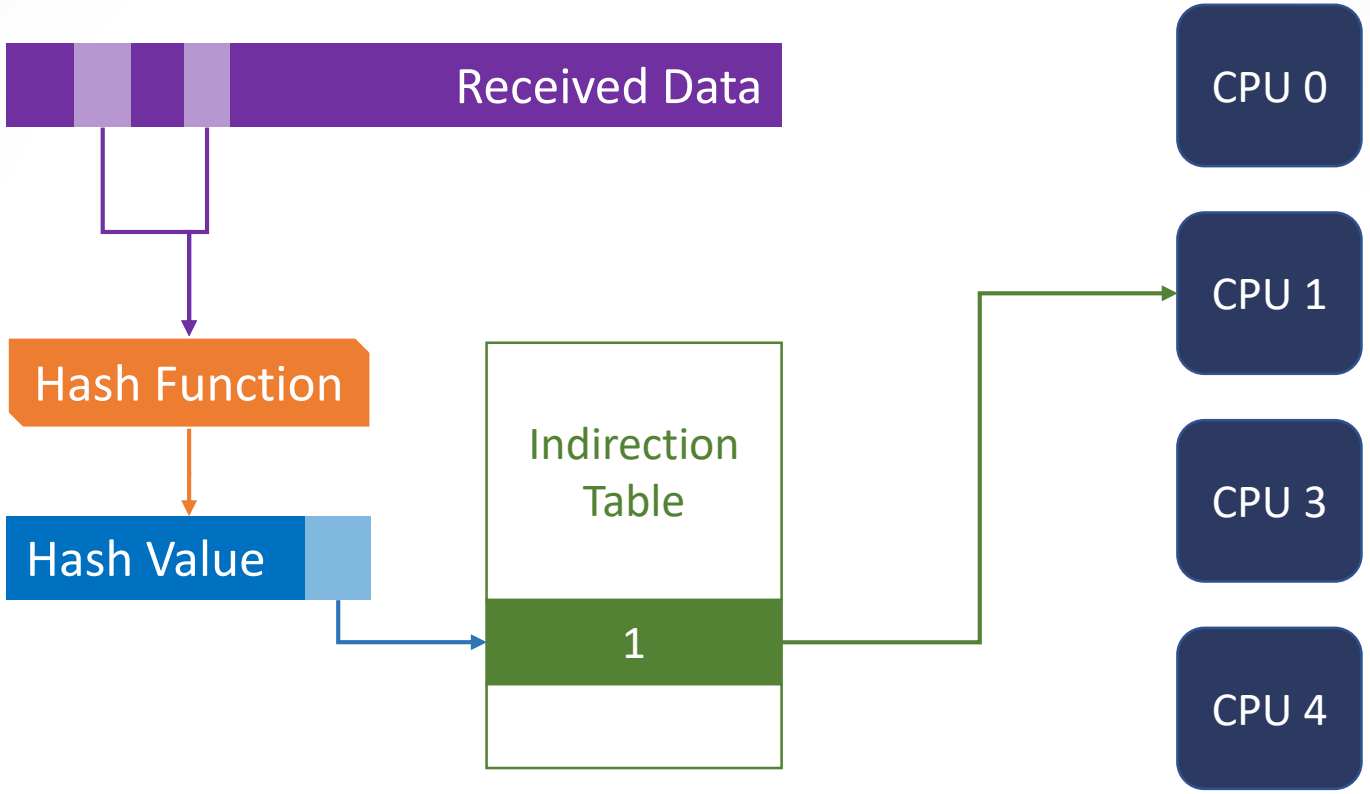
* Receive Side Scaling
** approx., depending on packet size and protocol

Interrupts and Cores



* Receive Side Scaling
** approx., depending on packet size and protocol

Receive Side Scaling (RSS)



Linux Rx Packet Handling Overview

- Receive Packet Scaling (RPS) is Receive Side Scaling (RSS) in software
- `/proc/interrupts`

Linux Rx Packet Handling Overview

- Receive Packet Scaling (RPS) is Receive Side Scaling (RSS) in software

```
cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
 54:          1          0          0          0   eth0
 55:          0          0     269502    1947651   eth0-TxRx-1
 56:          0          0    1439434     704478   eth0-TxRx-2
 57:     292354     389412          0          0   eth0-TxRx-3
 58:    1355854     689235          0          0   eth0-TxRx-4
```

IRQ assigned to this queue

Hardware interrupts handled on this core

Iface / queue id

Linux Rx Packet Handling Overview

- Receive Packet Scaling (RPS) is Receive Side Scaling (RSS) in software
- `/proc/interrupts`
- irqbalancer
- `/proc/softirq`

Linux Rx Packet Handling Overview

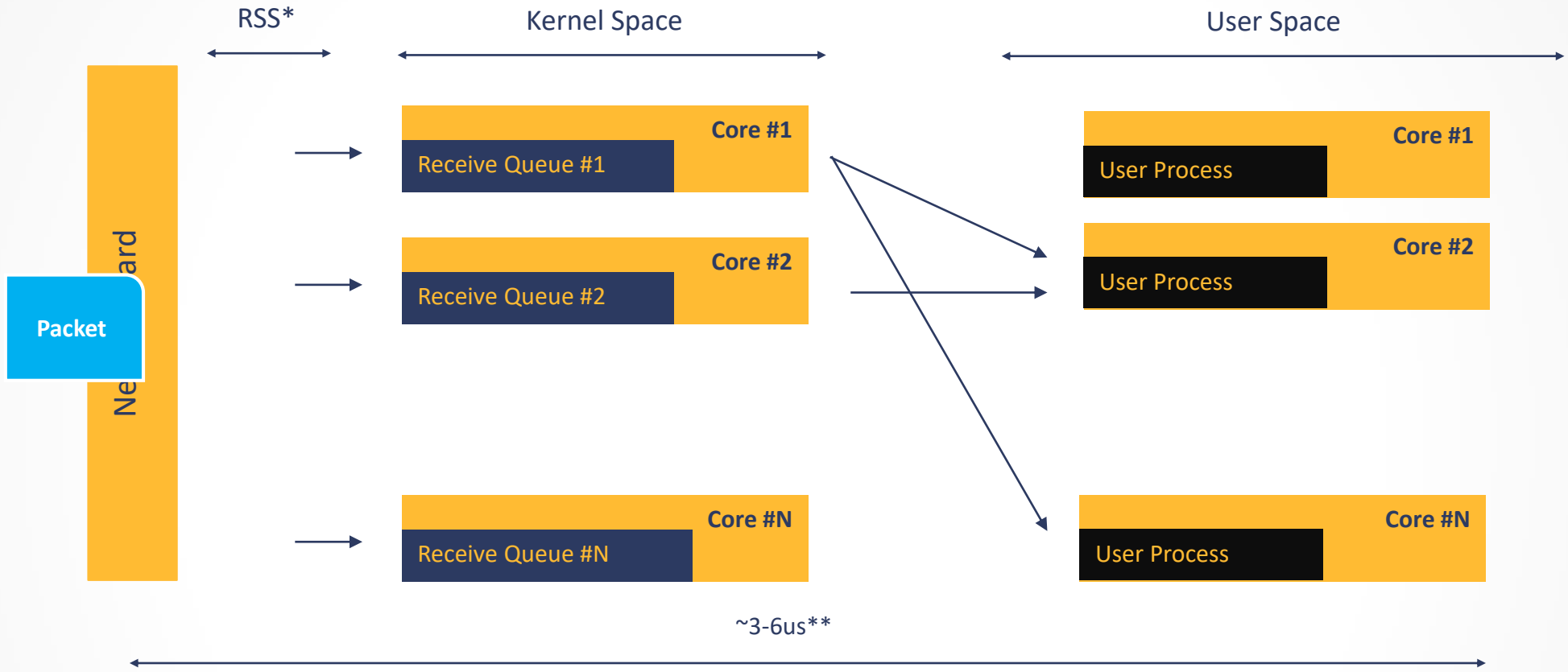
- Receive Packet Scaling (RPS) is Receive Side Scaling (RSS) in software

```
cat /proc/softirqs
                    CPU0       CPU1       CPU2       CPU3
          HI:          0          0          0          0
       TIMER: 2831512516 1337085411 1103326083 1423923272
      NET_TX:   15774435     779806     733217     749512
      NET_RX: 1671622615 1257853535 2088429526 2674732223
       BLOCK: 1800253852    1466177    1791366     634534
BLOCK_IOPOLL:          0          0          0          0
     TASKLET:         25          0          0          0
       SCHED: 2642378225 1711756029  629040543  682215771
     HRTIMER:    2547911    2046898    1558136    1521176
         RCU: 2056528783 4231862865 3545088730  844379888
```

Linux Rx Packet Handling Overview

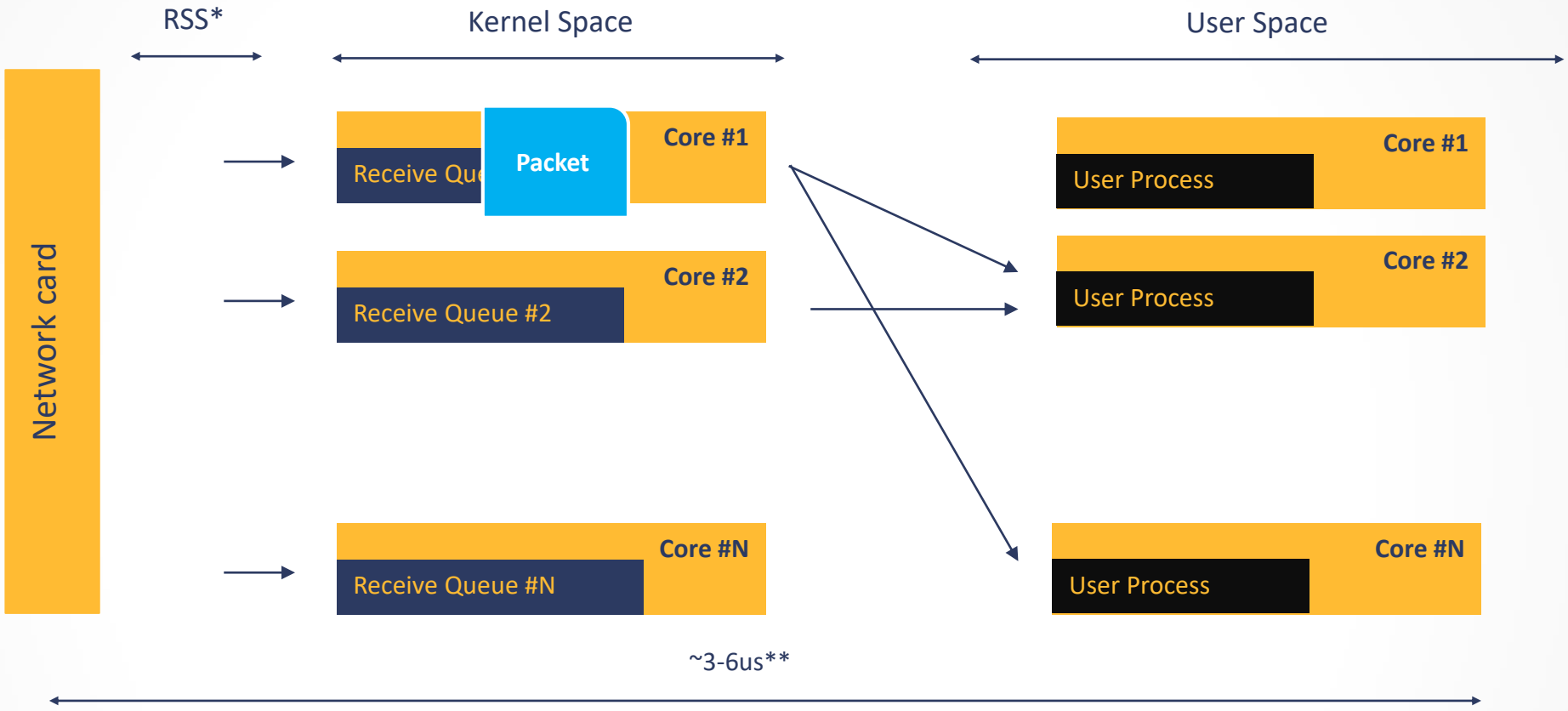
- Receive Packet Scaling (RPS) is Receive Side Scaling (RSS) in software
- `/proc/interrupts`
- irqbalancer
- `/proc/softirq`
- Receive Flow Scaling (RFS and accelerated RFS)

Receive Flow Scaling (RFS)



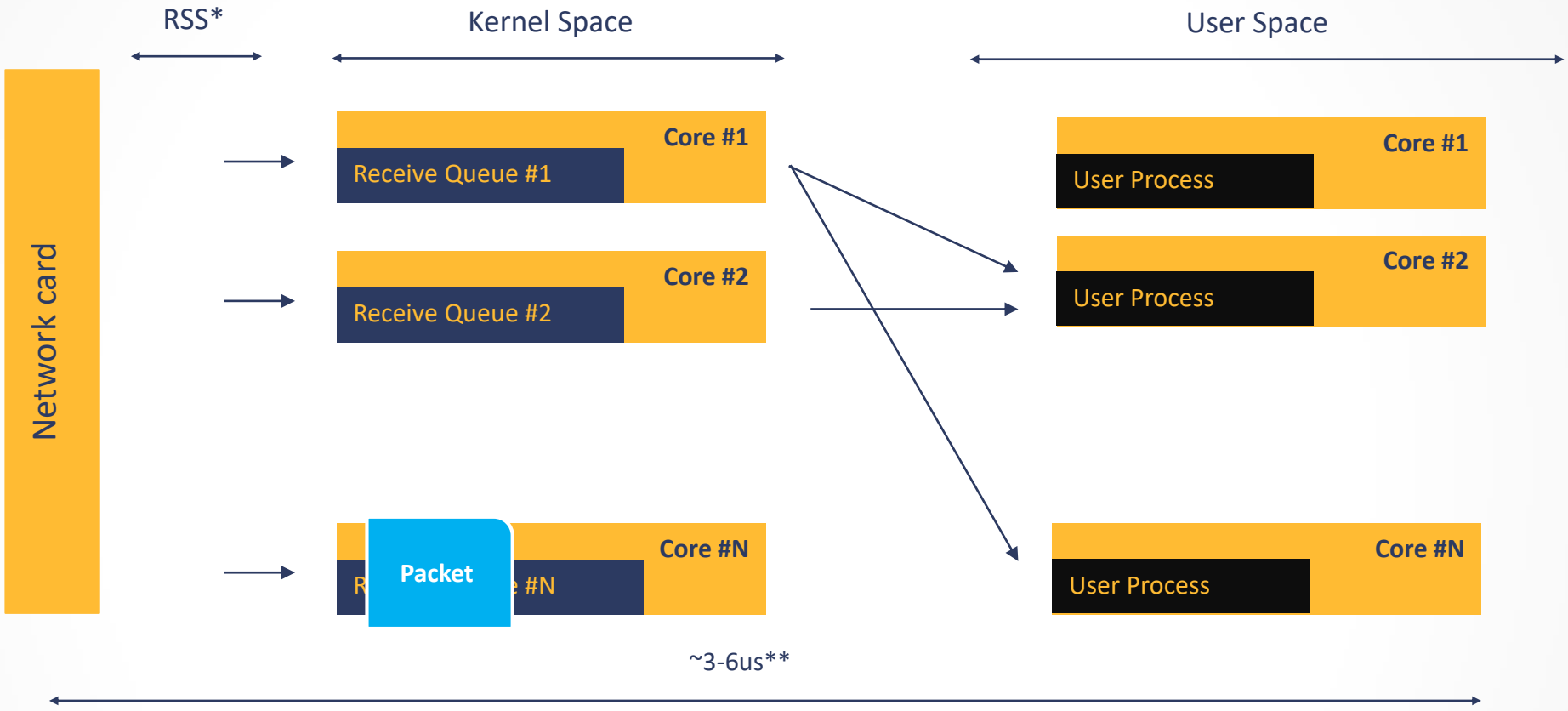
* Receive Side Scaling
** approx., depending on packet size and protocol

Receive Flow Scaling (RFS)



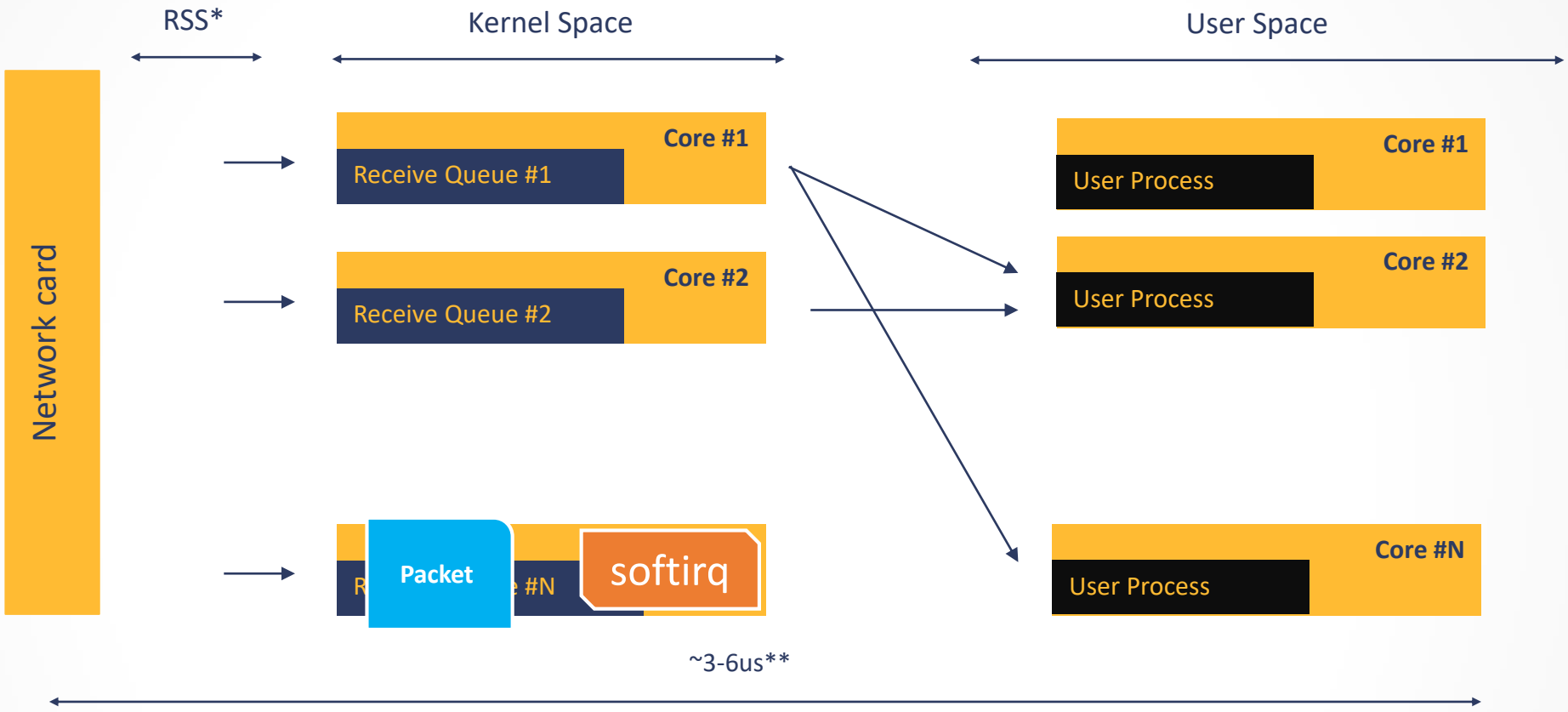
* Receive Side Scaling
** approx., depending on packet size and protocol

Receive Flow Scaling (RFS)



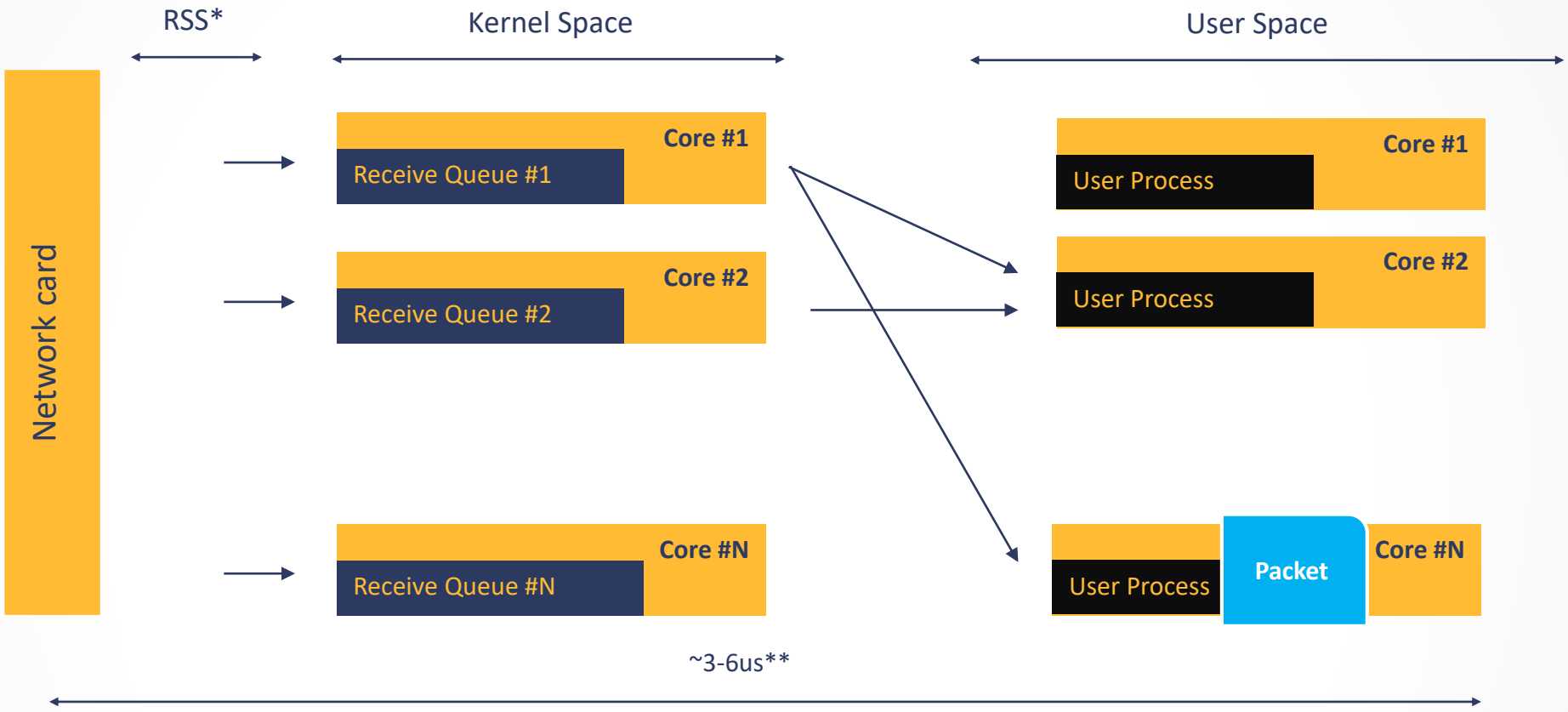
* Receive Side Scaling
** approx., depending on packet size and protocol

Receive Flow Scaling (RFS)



* Receive Side Scaling
** approx., depending on packet size and protocol

Receive Flow Scaling (RFS)



* Receive Side Scaling
** approx., depending on packet size and protocol

Bottlenecks in Packet Reception

- Network card hardware buffer overflow (**NIC to Kernel bottleneck**)
  - Check RSS (Receive Side Scaling) / multi-receive-queue setup
  - Handle interrupts on isolated cores
  - Increase receive queue size
  - Tune / disable `irqbalancer`
  - Check if NIC supports hardware RFS (Receive Flow Steering)
  - Increase softirq budget (`sysctl net.core.netdev_budget`)
- Socket receive queue (**Kernel to application bottleneck**)
  - Profile your app
  - Separate receive and processing logic to threads
  - Isolate and pin to hardware thread the receive thread
  - tune socket buffer sizes
  - Use **`tuned`** for pre-defined profiles

Networking Tools

- **`netstat`**
  - Command-line utility to view network statistics
  - Retrieves information from **`/proc/net`** filesystem

Networking Tools

- n

alf@server:~

```
[alf@server ~]$ netstat -atu
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp        0      0 server:ssh              192.168.1.48:59476      ESTABLISHED
tcp6       0      0 [::]:mysql              [::]:*                  LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
udp        0      0 0.0.0.0:slingshot       0.0.0.0:*
udp        0      0 0.0.0.0:7091            0.0.0.0:*
udp        0      0 0.0.0.0:bootpc          0.0.0.0:*
udp6       0      0 [::]:25087              [::]:*
udp6       0      0 server:dhcpv6-client    [::]:*
udp6       0      0 [::]:59809              [::]:*
[alf@server ~]$
```

Networking Tools

- **`netstat`**
  - Command-line utility to view network statistics
  - Retrieves information from **`/proc/net`** filesystem
- **`dropwatch`**
  - Interactive tool monitors packets free from memory by kernel

Networking Tools

- **netstat**

```
sudo ./dropwatch -l kas
Initalizing kallsyms db
dropwatch> start
Enabling monitoring...
Kernel monitoring activated.
Issue Ctrl-C to stop monitoring

1 drops at tcp_v4_do_rcv+cd (0xffffffff81799bad)
10 drops at tcp_v4_rcv+80 (0xffffffff8179a620)
1 drops at sk_stream_kill_queues+57 (0xffffffff81729ca7)
4 drops at unix_release_sock+20e (0xffffffff817dc94e)
1 drops at igmp_rcv+e1 (0xffffffff817b4c41)
1 drops at igmp_rcv+e1 (0xffffffff817b4c41)
```

Networking Tools

- **`netstat`**
  - Command-line utility to view network statistics
  - Retrieves information from **`/proc/net`** filesystem
- **`dropwatch`**
  - Interactive tool monitors packets free from memory by kernel
- **`ip / ethtool`**
  - Utilities for managing and monitoring routes, devices, tunnels, network card settings (`ip` a replacement for ifconfig)

Networking Tools

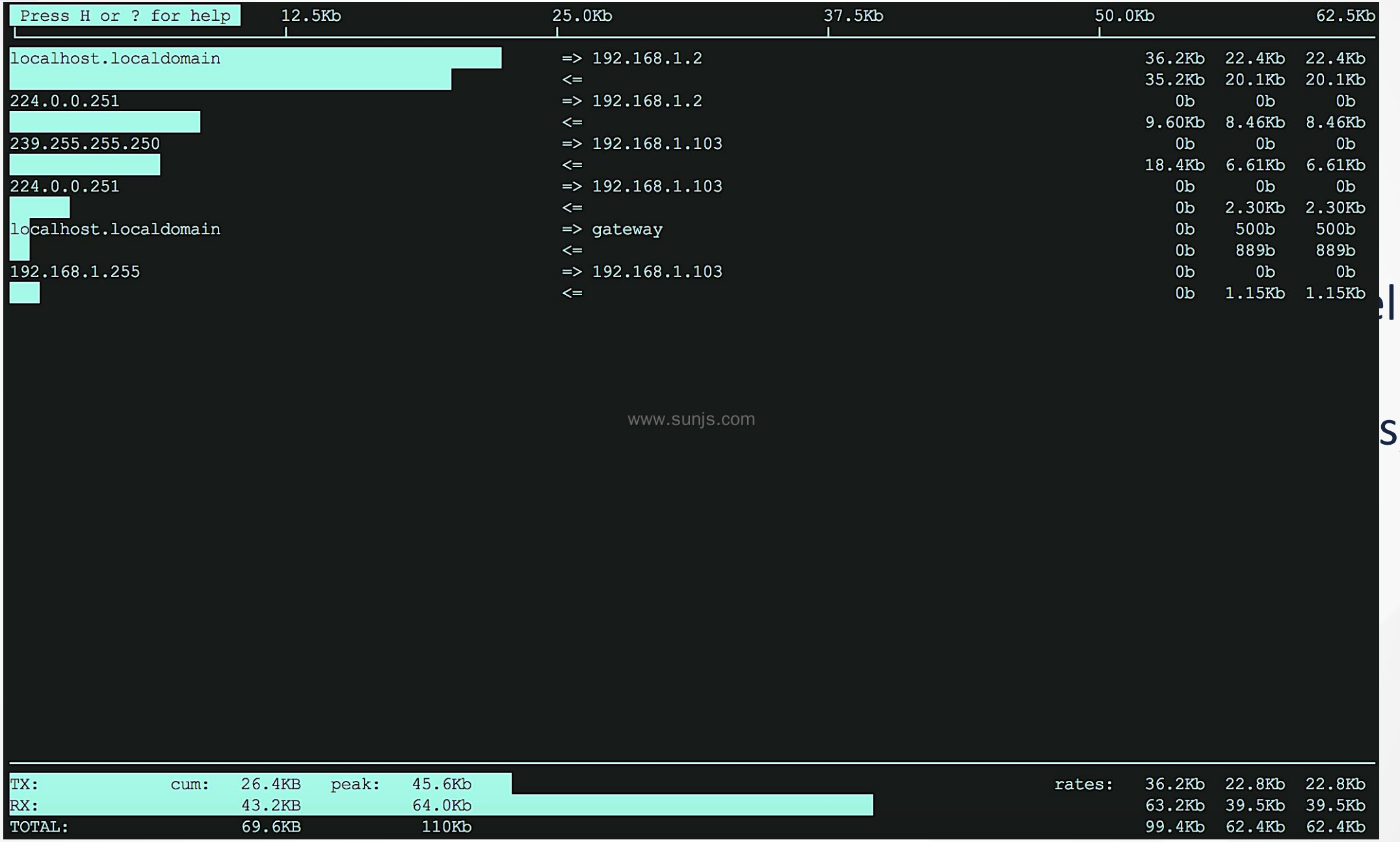
- **netstat**
  - Command-line utility to view network statistics

```
tecmint@tecmint ~ $ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 28:d2:44:eb:bd:98 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.104/24 brd 192.168.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::2ad2:44ff:feeb:bd98/64 scope link
       valid_lft forever preferred_lft forever
3: wlan0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group default qlen 1000
    link/ether 38:b1:db:7c:78:c7 brd ff:ff:ff:ff:ff:ff
tecmint@tecmint ~ $
```

Networking Tools

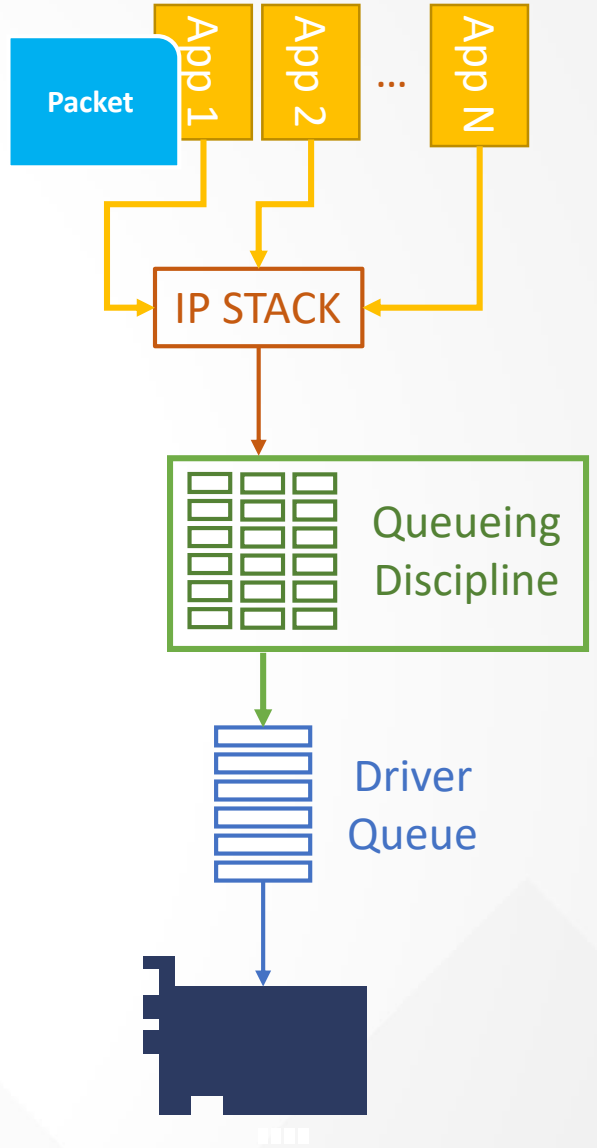
- **`netstat`**
  - Command-line utility to view network statistics
  - Retrieves information from **`/proc/net`** filesystem
- **`dropwatch`**
  - Interactive tool monitors packets free from memory by kernel
- **`ip / ethtool`**
  - Utilities for managing and monitoring routes, devices, tunnels, network card settings (`ip` a replacement for ifconfig)
- **`iftop`**
  - `top`/`htop` for network interfaces

Networking Tools



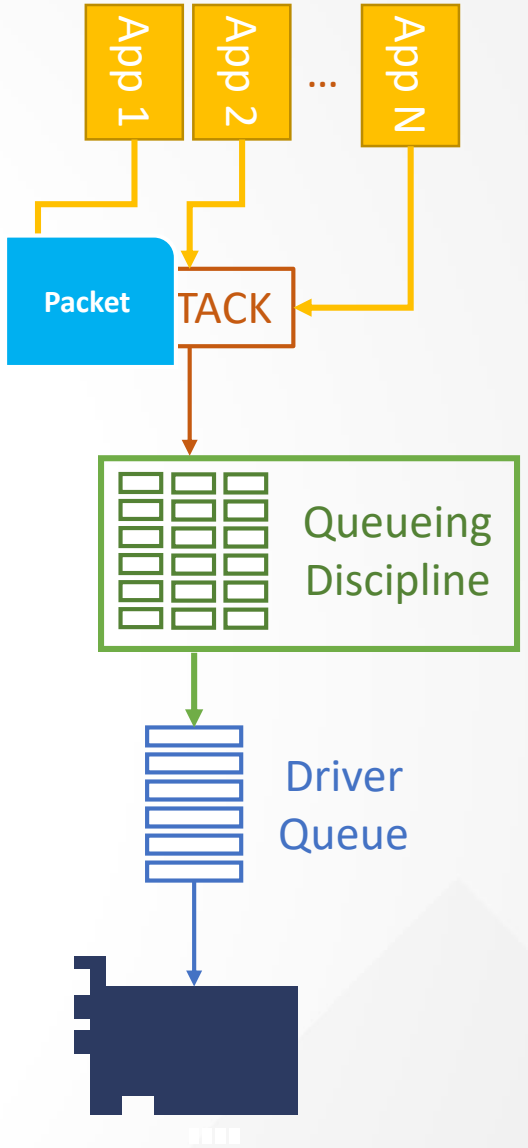
Linux Tx Packet Handling Overview

- App calls `write/sendto/sendmsg`



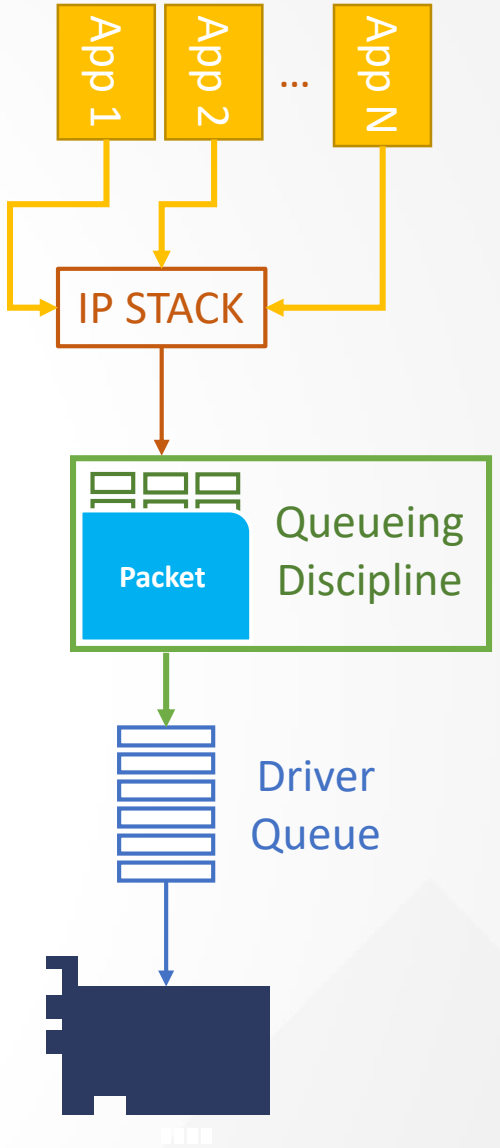
Linux Tx Packet Handling Overview

- App calls `write/sendto/sendmsg`
- copy payload to kernel space
  - when there is no space, it blocks the thread
- Routing / security / permission checks
- Protocol stack



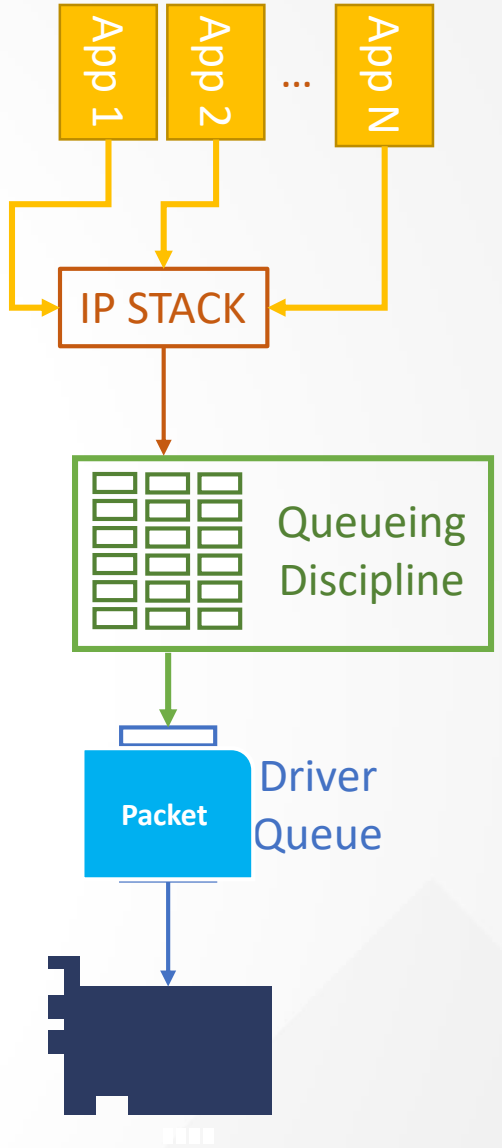
Linux Tx Packet Handling Overview

- App calls `write/sendto/sendmsg`
- copy payload to kernel space
  - when there is no space, it blocks the thread
- Routing / security / permission checks
- Protocol stack
- Schedules packet for transfer
  - Queue discipline



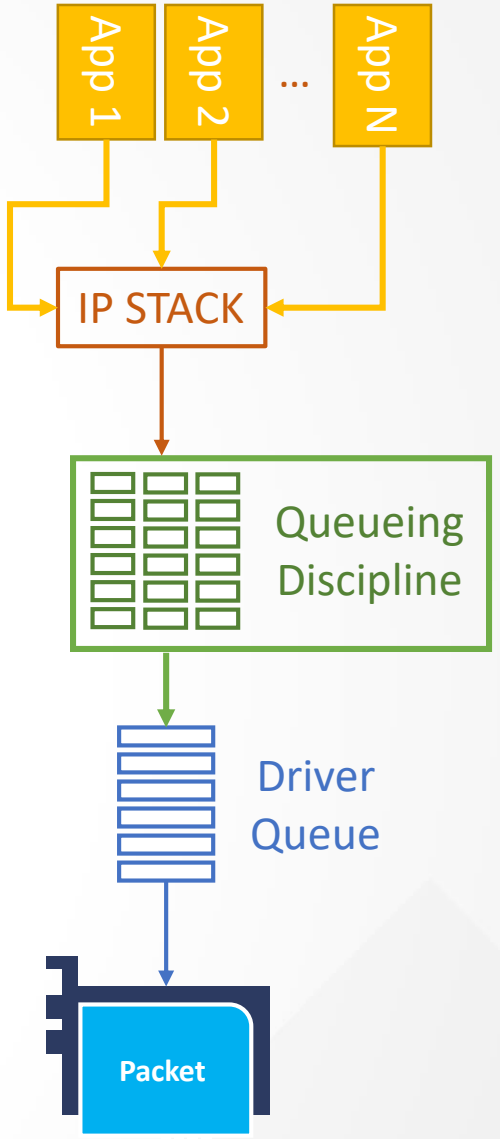
Linux Tx Packet Handling Overview

- App calls `write/sendto/sendmsg`
- copy payload to kernel space
  - when there is no space, it blocks the thread
- Routing / security / permission checks
- Protocol stack
- Schedules packet for transfer
  - Queue discipline
- Copied to Driver Queue



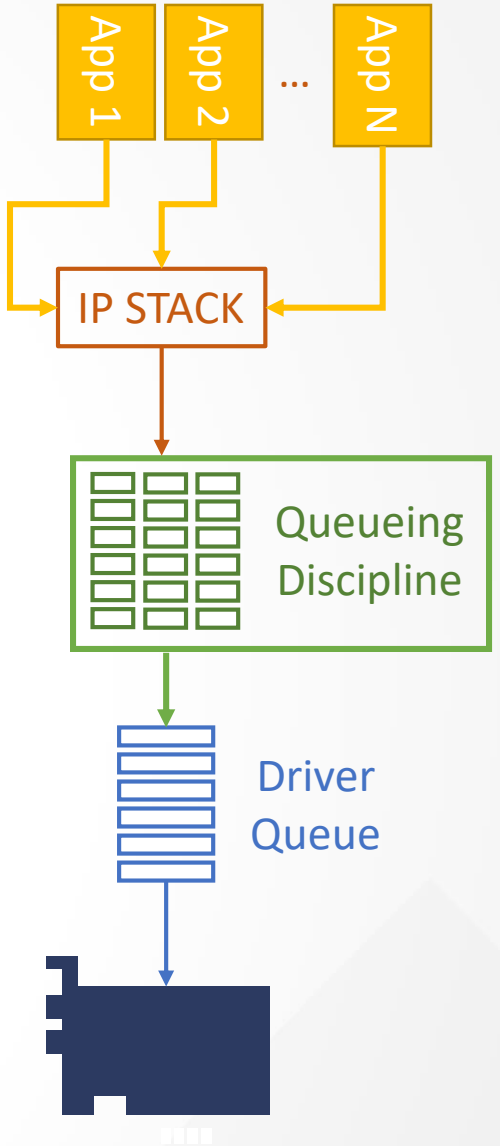
Linux Tx Packet Handling Overview

- App calls `write/sendto/sendmsg`
- copy payload to kernel space
  - when there is no space, it blocks the thread
- Routing / security / permission checks
- Protocol stack
- Schedules packet for transfer
  - Queue discipline
- Copied to Driver Queue
- DMA (Direct Memory Access) transfer to hardware buffer



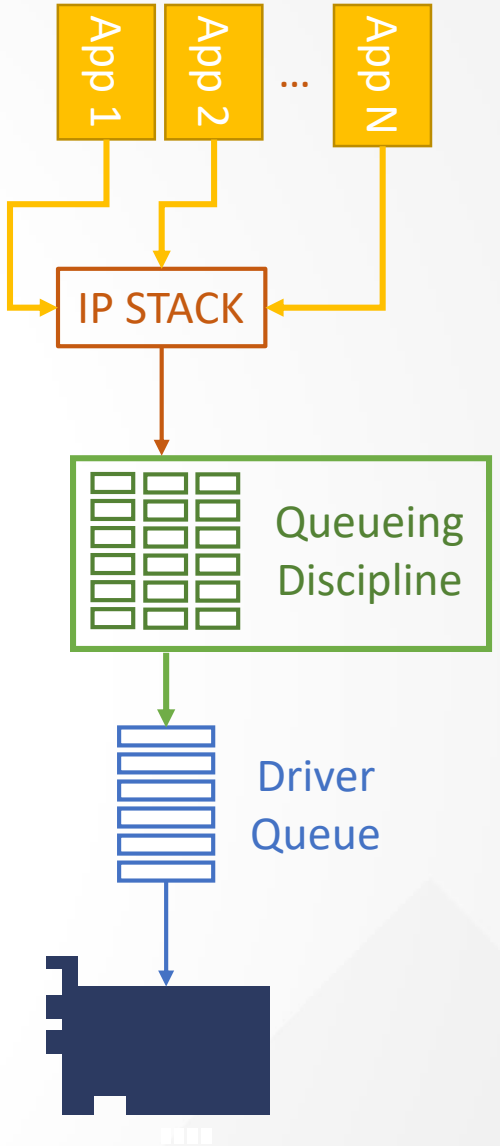
Linux Tx Packet Handling Overview

- App calls `write/sendto/sendmsg`
- copy payload to kernel space
  - when there is no space, it blocks the thread
- Routing / security / permission checks
- Protocol stack
- Schedules packet for transfer
  - Queue discipline
- Copied to Driver Queue
- DMA (Direct Memory Access) transfer to hardware buffer
- Data sent out by the network card (NIC)

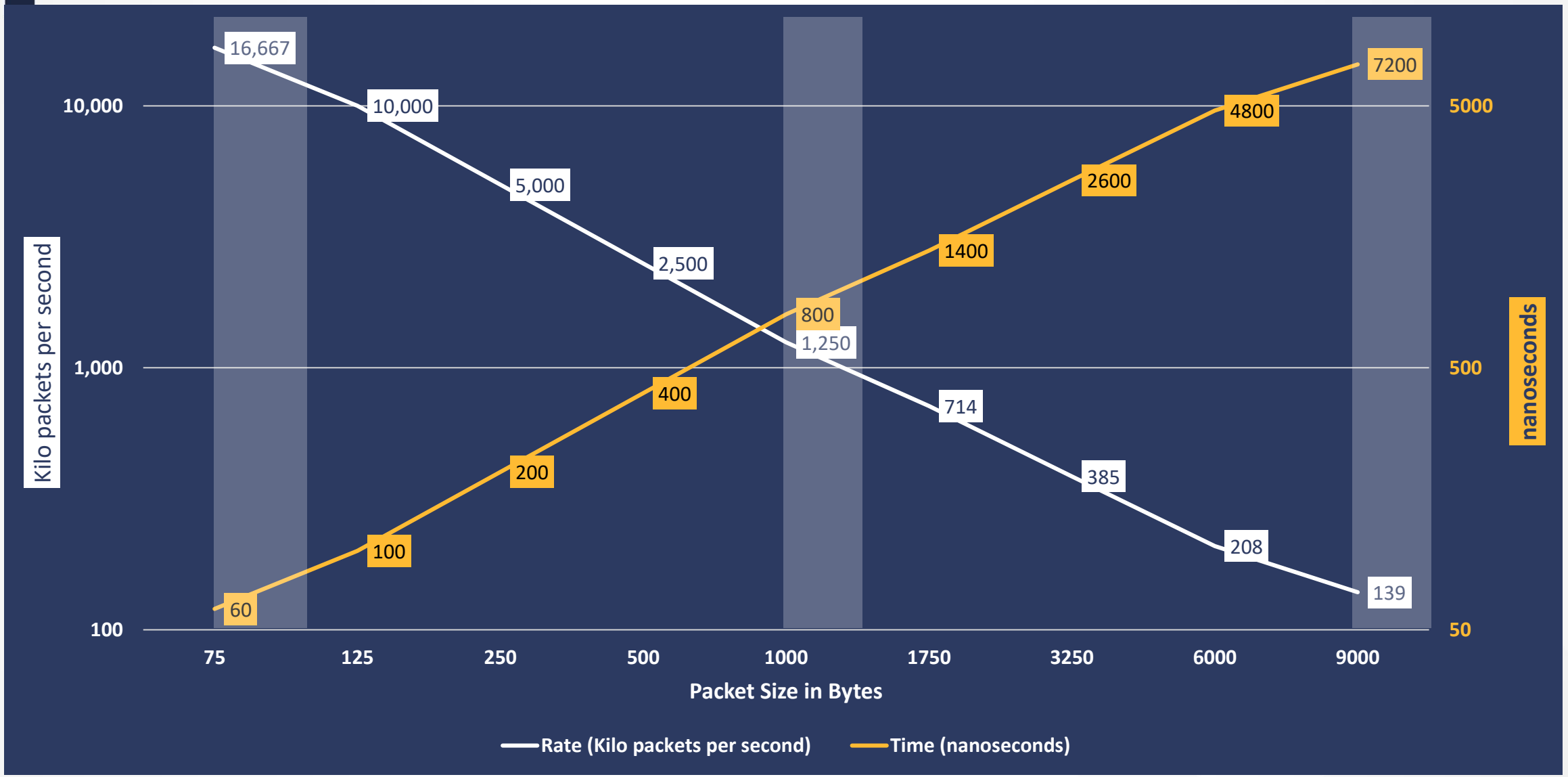


Linux Tx Packet Handling Overview

- App calls `write/sendto/sendmsg`
- copy payload to kernel space
  - when there is no space, it blocks the thread
- Routing / security / permission checks
- Protocol stack
- Schedules packet for transfer
  - Queue discipline
- Copied to Driver Queue
- DMA (Direct Memory Access) transfer to hardware buffer
- Data sent out by the network card (NIC)
- NIC reports the data transfer result

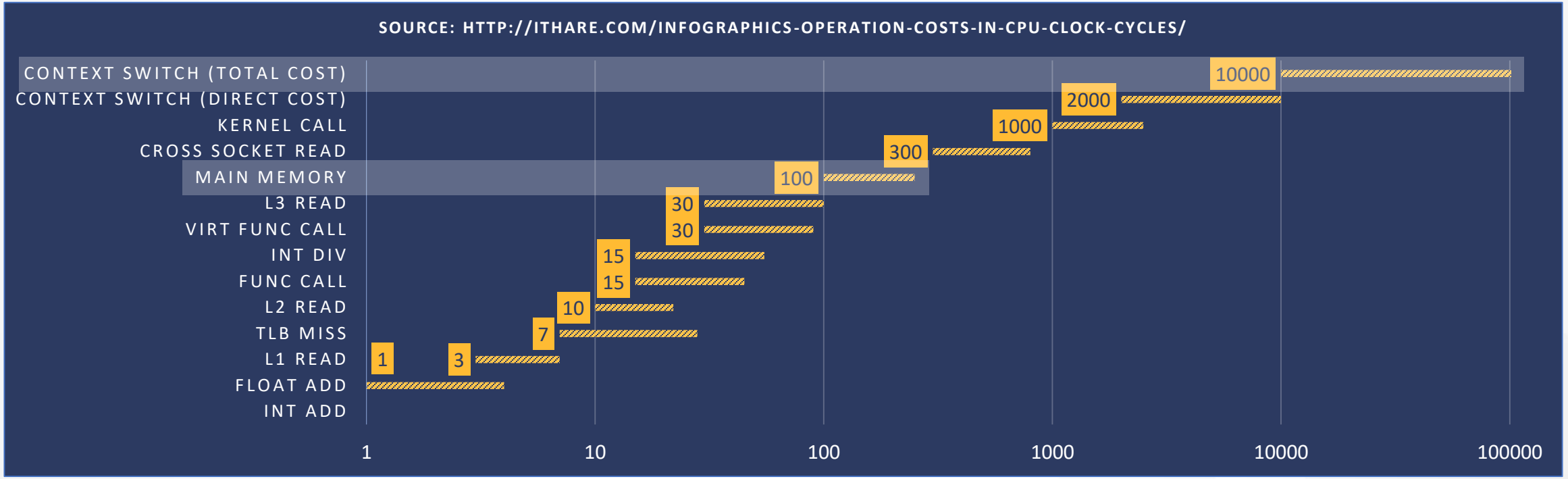


Packet Size vs Packet Rate and Time between packets



How Could We Improve?

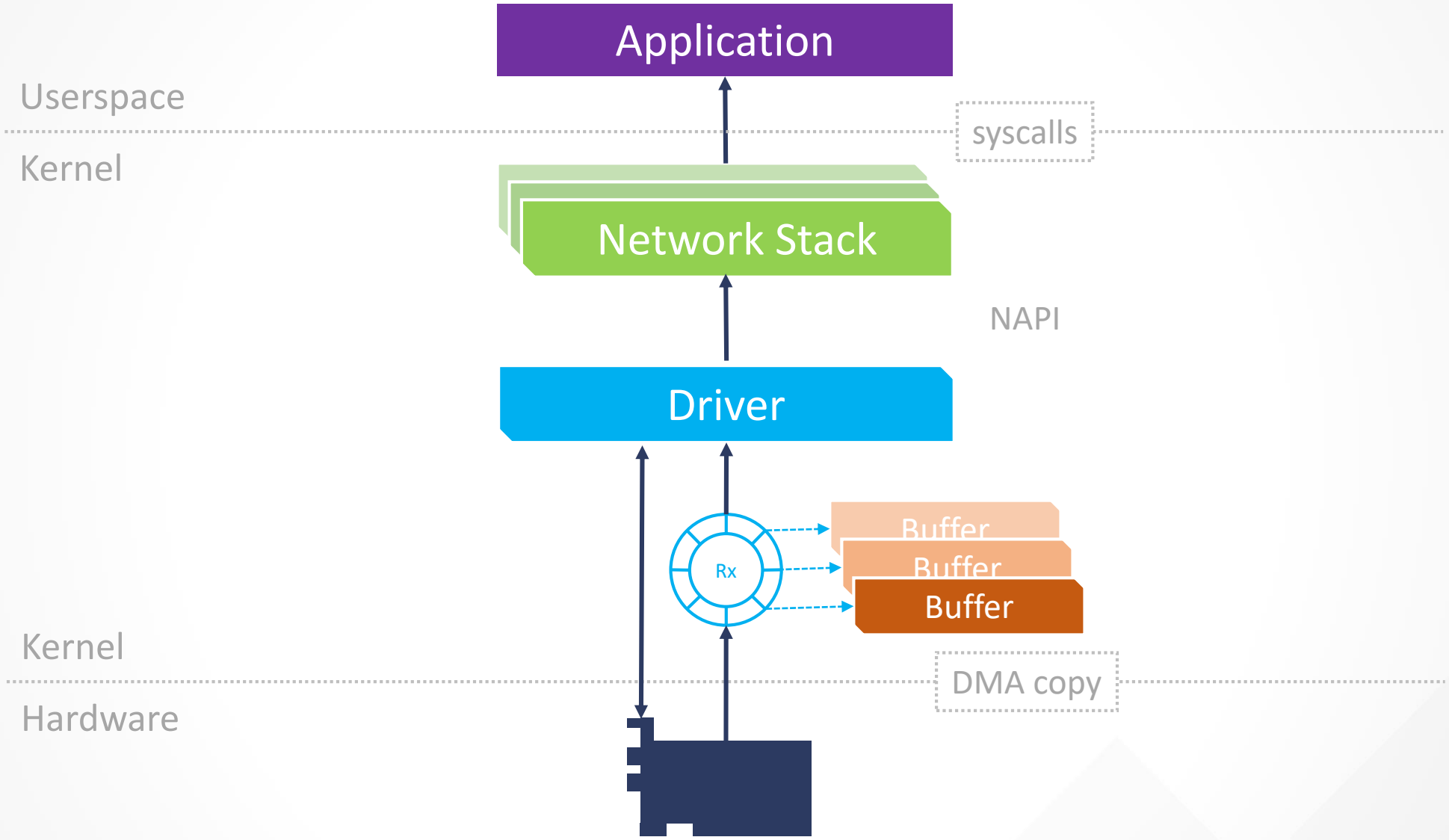
- Reduce number of protection ring / context switches
- Pre-allocate everything
- Offload certain work to NIC
- “Zero copy”



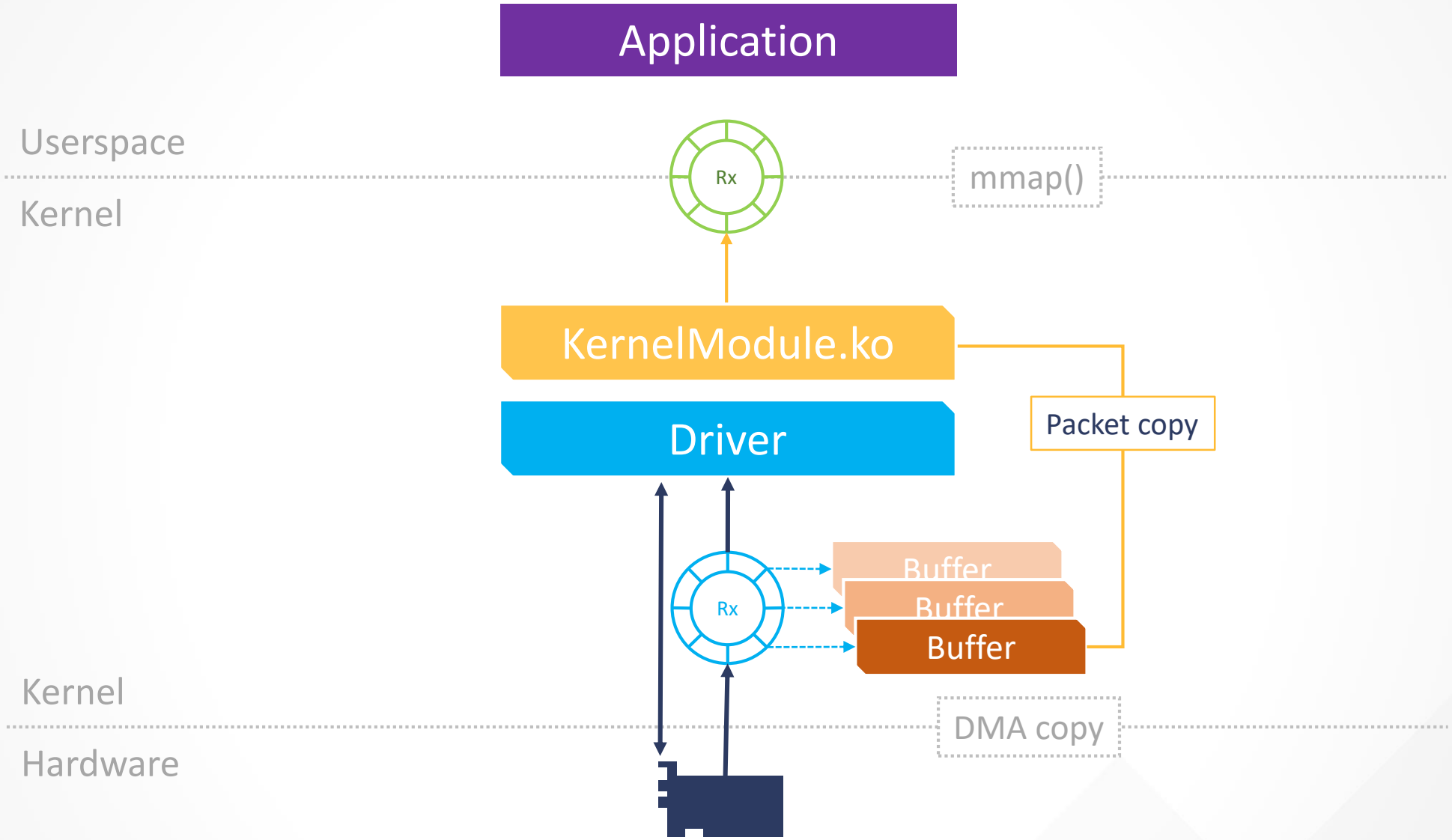
User-Space Network Stacks

Basic Architectural Overview

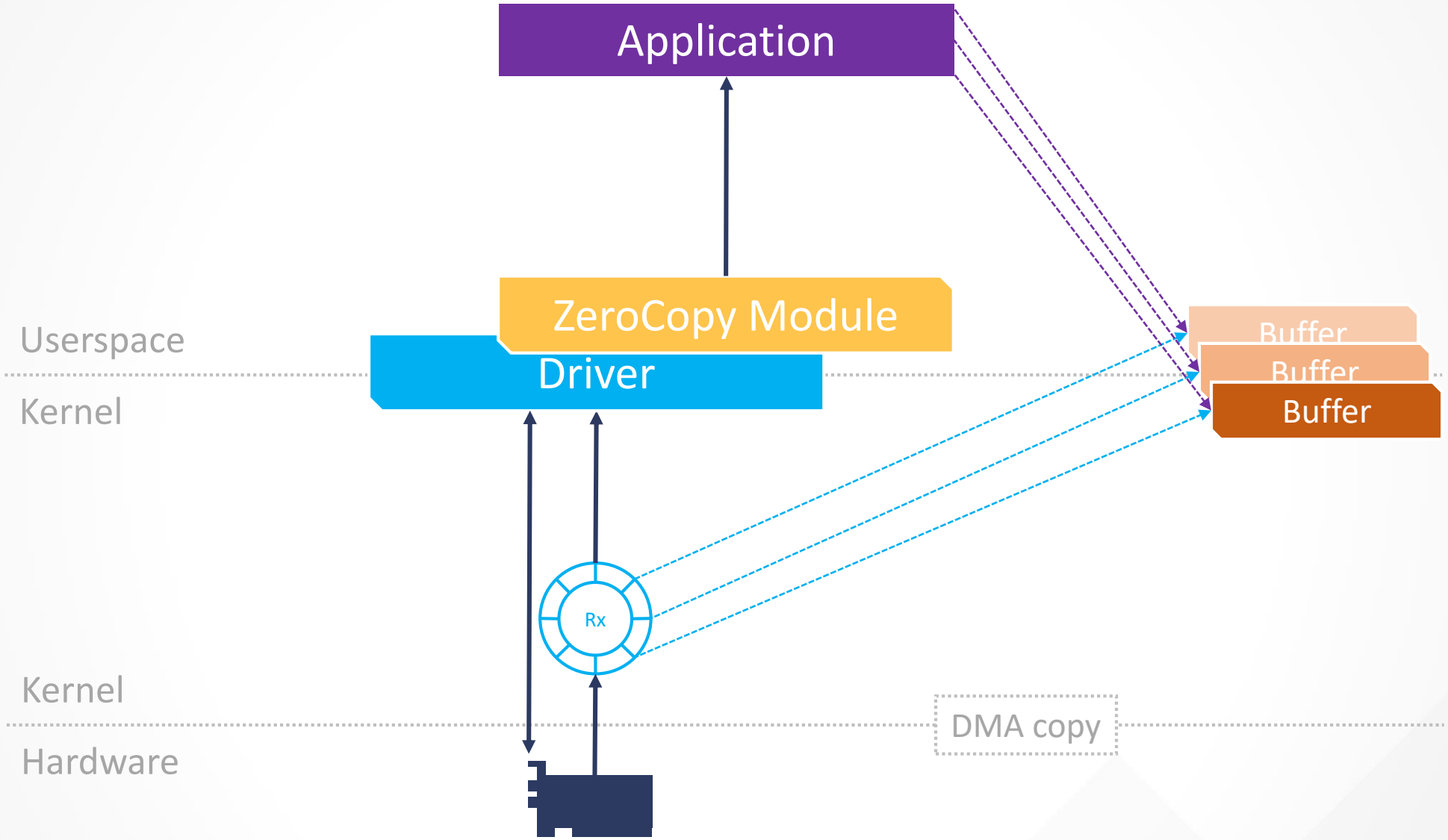
Quick Reminder of Linux Network Stack



Eliminating syscalls



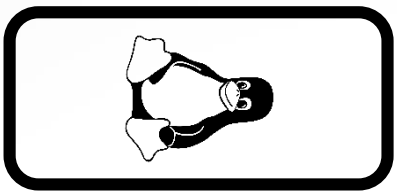
Zero-copy


Application
ZeroCopy Module
Driver
Userspace
Kernel
Buffer
Buffer
Buffer
Rx
Kernel
DMA copy
Hardware

User-Space Overview

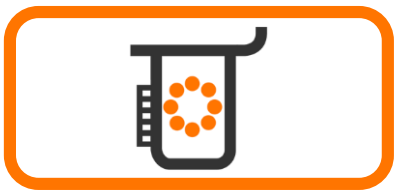
L2 L3 L4

DPDK
DATA PLANE DEVELOPMENT KIT



AF_PACKET, XDP, AF_XDP

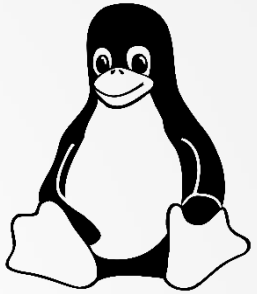
netmap



PF_RING

VPP
Vector Packet Processing

AF_PACKET and XDP

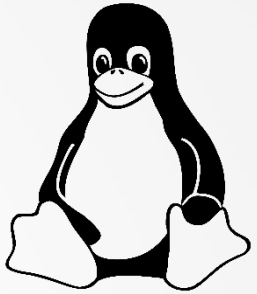


- AF_PACKET v1-v3
  - socket that allows communication at the link layer
  - part of Linux Kernel, same socket API
  - User-Space packet buffer via mmap

- XDP (eXpress Data Path)
  - allows packets to be reflected, filtered or redirected without traversing the network stack
  - Not a receive / send API, but an extended Berkley Packet Filter (eBPF) processor
  - Batched I/O operation
  - Goal is 100Gbit with GRO (Generic Receive Offload)
  - Has infrastructure to offload eBPF to NIC (Network Card)

```
int s = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL)); // receive all protocols

struct sockaddr_ll socket_address; // link-layer socket address

memset( & socket_address, 0, sizeof(socket_address));
socket_address.sll_family = AF_PACKET;
socket_address.sll_ifindex = if_nametoindex("eth0"); // use eth0 network interface

bind(s, (struct sockaddr * ) & socket_address, sizeof(socket_address));
char buf[1600];

while (1) {
  memset( & buf, 0, sizeof(buf));

  recv_size = recv(s, & buf, sizeof(buf), 0);
  if (recv_size == -1) {
    perror("Socket receive");
    exit(0);
  }

  printf("\n");
  for (i = 0; i & lt; recv_size; i++) {
    printf("%02x ", buf[i]);
  }
}
```

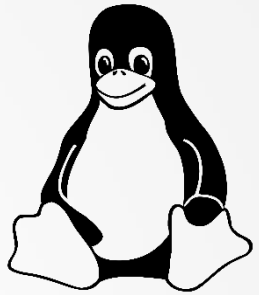
AF_PACKET and XDP

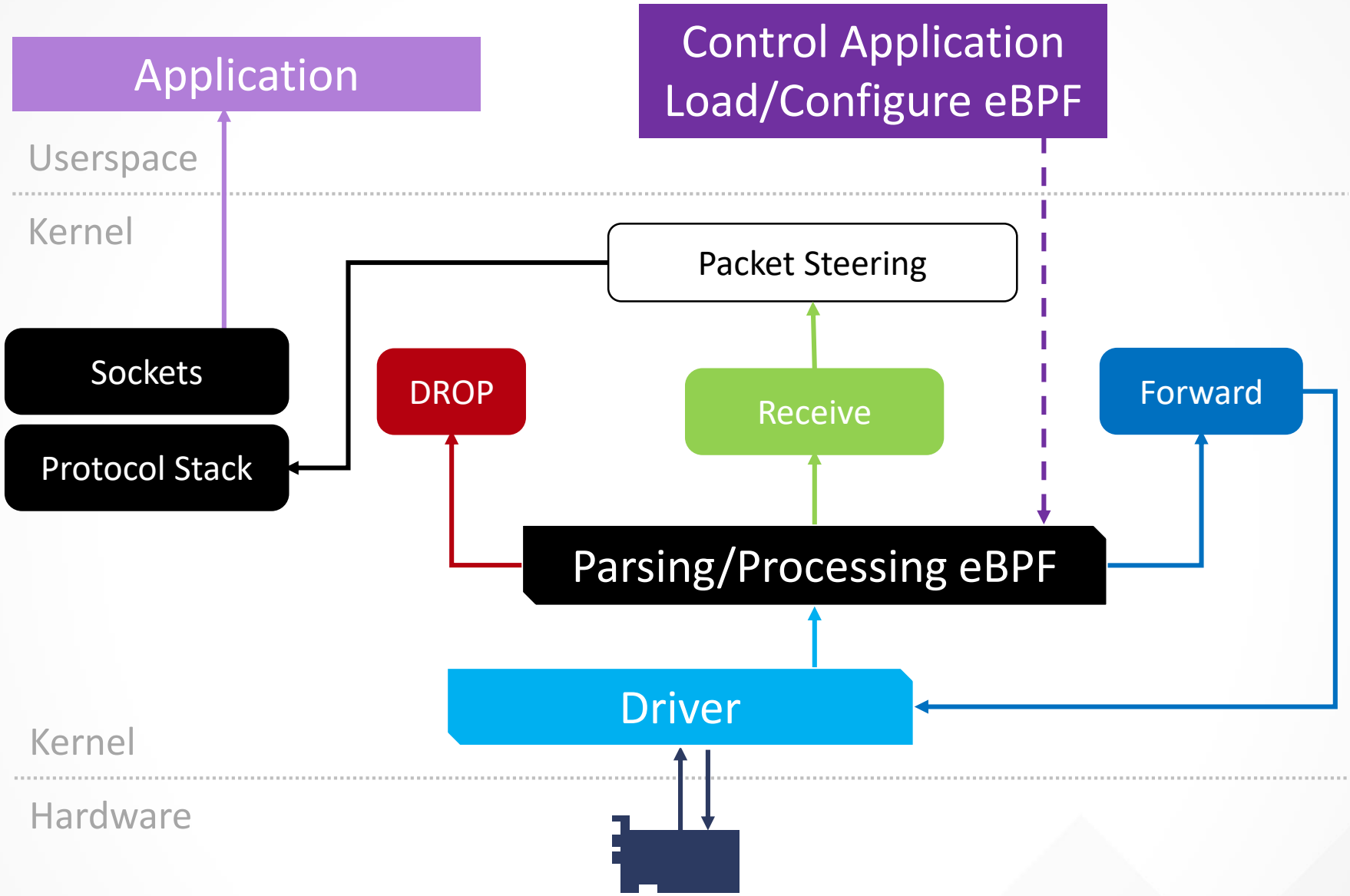


- AF_PACKET v1-v3
  - socket that allows communication at the link layer
  - part of Linux Kernel, same socket API
  - User-Space packet buffer via mmap

- XDP (eXpress Data Path)
  - allows packets to be reflected, filtered or redirected without traversing the network stack
  - Not a receive / send API, but an extended Berkley Packet Filter (eBPF) processor
  - Batched I/O operation
  - Goal is 100Gbit with GRO (Generic Receive Offload)
  - Has infrastructure to offload eBPF to NIC (Network Card)

XDP Receive

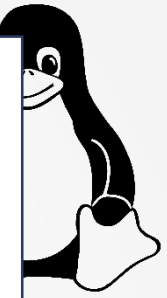



Application
Control Application
Load/Configure eBPF
Userspace
Kernel
Packet Steering
Sockets
Protocol Stack
DROP
Receive
Forward
Parsing/Processing eBPF
Driver
Kernel
Hardware

XDP Receive

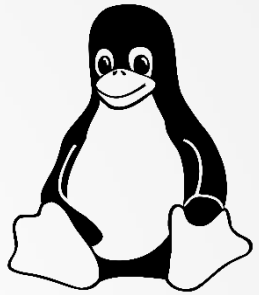
```
switch (action) {
case XDP_PASS:
  break; /* Normal netstack handling */
case XDP_TX:
  if (driver_xmit(dev, page, length) == NETDEV_TX_OK)
    goto consumed;
  goto xdp_drop; /* Drop on xmit failure */
default:
  bpf_warn_invalid_xdp_action(action);
case XDP_ABORTED: // eBPF program error
case XDP_DROP:
  xdp_drop:
    if (driver_recycle(page, ring))
      goto consumed;
  goto next; /* Drop */
case XDP_REDIRECT:
      /*...*/
      goto consumed;
}
```

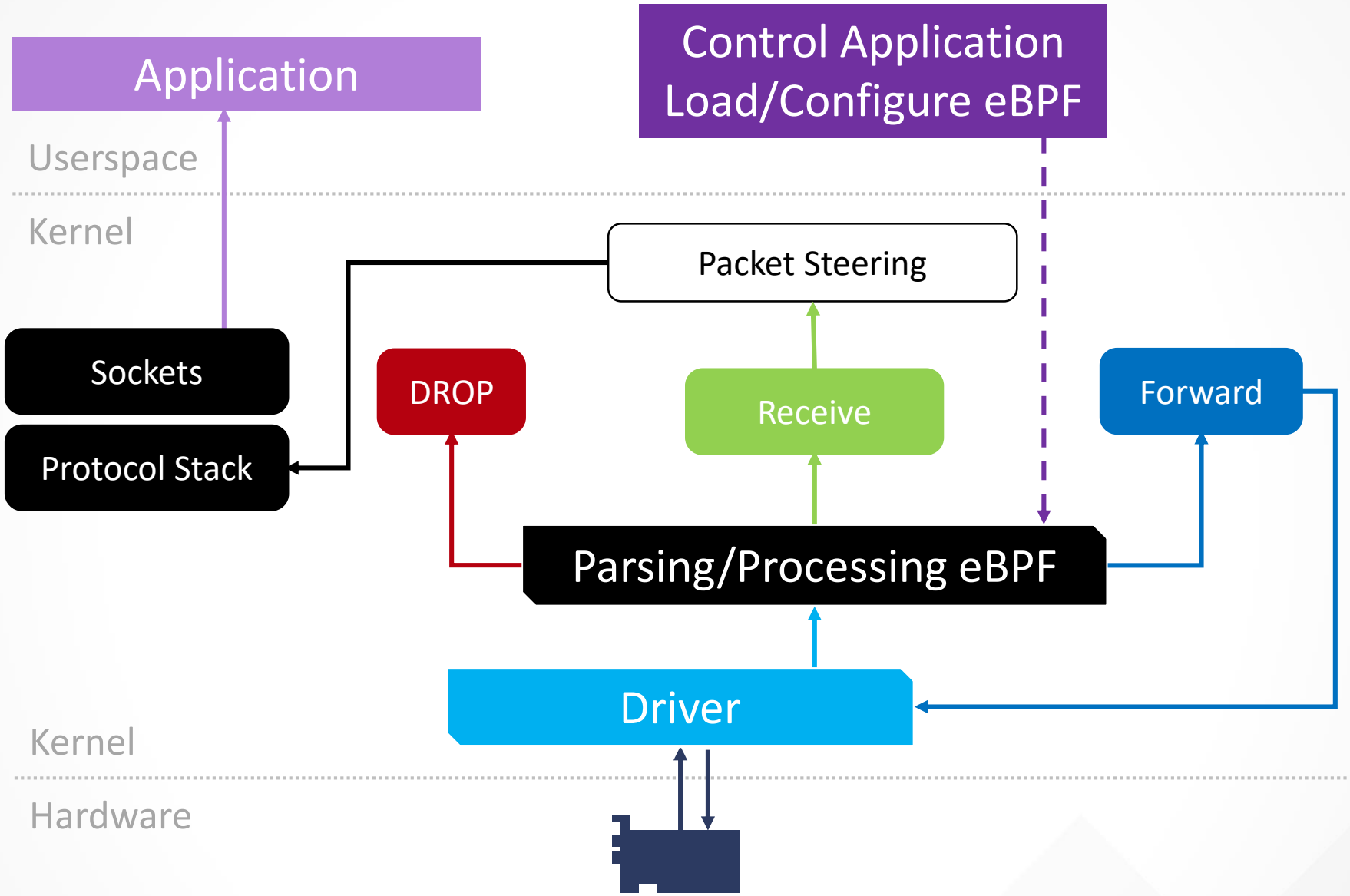
XDP Receive



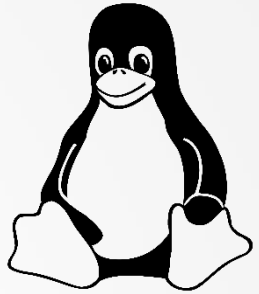
```
int xdp_prog_drop_all_UDP(struct xdp_md * ctx)
{
  void * data_end = (void * )(long) ctx - > data_end;
  void * data = (void * )(long) ctx - > data;
  struct ethhdr * eth = data;
  u64 nh_off;
  u32 ipproto = 0;
  nh_off = sizeof( * eth); /* ETH_HLEN == 14 */

  /* Verifier use this boundry check */
  if (data + nh_off > data_end)
       return XDP_ABORTED;
  if (eth - > h_proto == htons(ETH_P_IP))
       ipproto = parse_ipv4(data, nh_off, data_end);
  if (ipproto == IPPROTO_UDP)
       return XDP_DROP;
  return XDP_PASS;
}
```

XDP Receive




Application
Control Application
Load/Configure eBPF
Userspace
Kernel
Packet Steering
Sockets
Protocol Stack
DROP
Receive
Forward
Parsing/Processing eBPF
Driver
Kernel
Hardware

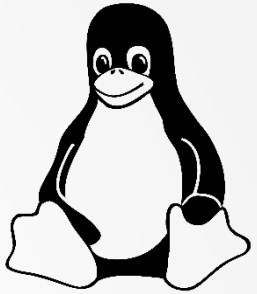
AF_XDP



- AF_XDP
  - Not a kernel bypass, interface at user-space for XDP
  - Kernel 4.18+
  - No system calls in data path
  - True zero copy (DMA + HW RSS)
  - Goal is 25Gbit/s for 64 byte packets, 40Gbit for large packets
  - Special focus on security and isolation
  - Control plane is still in Kernel, security and isolation is important
  - Does not expose hardware details (ease of use)
  - Requires modified device drivers for zero copy
  - XDP_SKB < XDP_DRV < XDP_DRV + ZC

* Fast Packet Processing in Linux with AF_XDP [FOSDEM2018]

AF_XDP



- AF_XDP
  - Not a kernel bypass, interface at user-space for XDP
  - Kernel 4.18+
  -

	V3	XDP_SKB	XDP_DRV	XDP_DRV+ZC
rxdrop	0.73 Mpps	3.3 Mpps	11.6 Mpps	16.9 Mpps
txpush	0.98 Mpps	2.2 Mpps	-	21.8 Mpps
L2fwd	0.71 Mpps	1.7 Mpps	-	10.3 Mpps

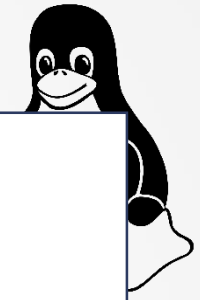
  - Control plane is still in Kernel, security and isolation is important
  - Does not expose hardware details (ease of use)
  - Requires modified device drivers for zero copy
  - XDP_SKB < XDP_DRV < XDP_DRV + ZC

* Fast Packet Processing in Linux with AF_XDP [FOSDEM2018]

AF_XDP

```
sfd = socket(PF_XDP, SOCK_RAW, 0);
buffs = calloc(num_buffs, FRAME_SIZE);
// ..Pin memory with umem character device...
setsockopt(sfd, SOL_XDP, XDP_RX_RING, & req, sizeof(req));
setsockopt(sfd, SOL_XDP, XDP_TX_RING, & req, sizeof(req));
mmap(..., sfd); /* map kernel Tx/Rx rings */
// ..Post Rcv buffers..
struct sockaddr_xdp addr = { PF_XDP, ifindex, queue_id };
bind(sfd, addr, sizeof(addr));
for (;;) {
  read_messages(sfd, msgs, ....);
  process_messages(msgs);
  send_messages(sfd, msgs, ....);
};
```

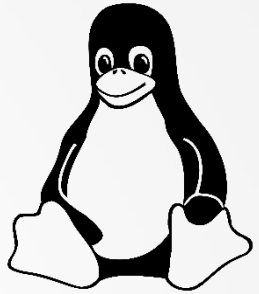
AF_XDP



-
```
int dequeue_one(RING *ring, RING_TYPE *item)
{
    __u32 entries = *ring->producer - *ring->consumer;

    if (entries == 0)
        return -1;

    // read-barrier!
    *item = ring->desc[*ring->consumer & (RING_SIZE - 1)];
    (*ring->consumer)++;
    return 0;
}
```

- XDP_SKB < XDP_DRV < XDP_DRV + ZC

AF_XDP



- AF_XDP
  - Not a kernel bypass, interface at user-space for XDP
  - Kernel 4.18+
  - No system calls in data path
  - True zero copy (DMA + HW RSS)
  - Goal is 25Gbit/s for 64 byte packets, 40Gbit for large packets
  - Special focus on security and isolation
  - Control plane is still in Kernel, security and isolation is important
  - Does not expose hardware details (ease of use)
  - Requires modified device drivers for zero copy
  - XDP_SKB < XDP_DRV < XDP_DRV + ZC

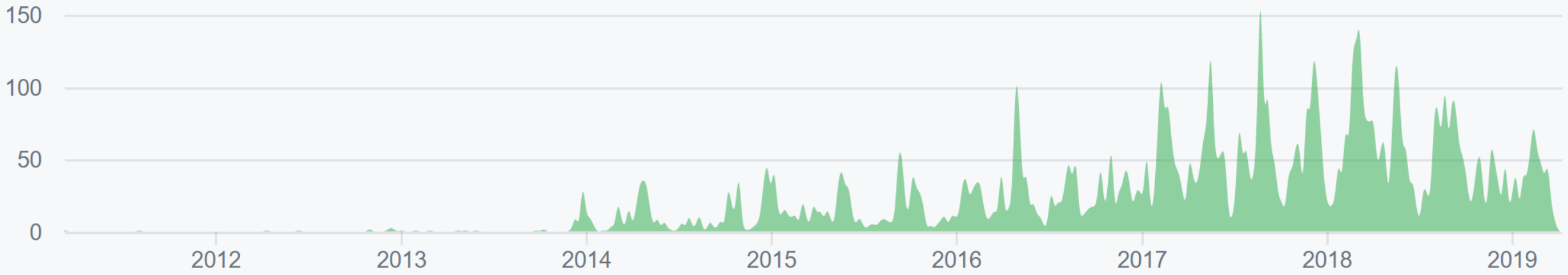
* Fast Packet Processing in Linux with AF_XDP [FOSDEM2018]

DPDK

- Data Plane Development Kit is a collection of libraries with custom API
- Open-source and managed by the Linux Foundation (FD.io)
- No transparent mode (i.e. once driver loaded DPDK takes over the NIC)
- Well documented, vibrant community with large companies as active contributors
- Initially by Intel (with Intel NIC/CPU support only), but now supports a variety of CPUs and NICs
- Claims line rate for 40G+ with small packet sizes

DPDK

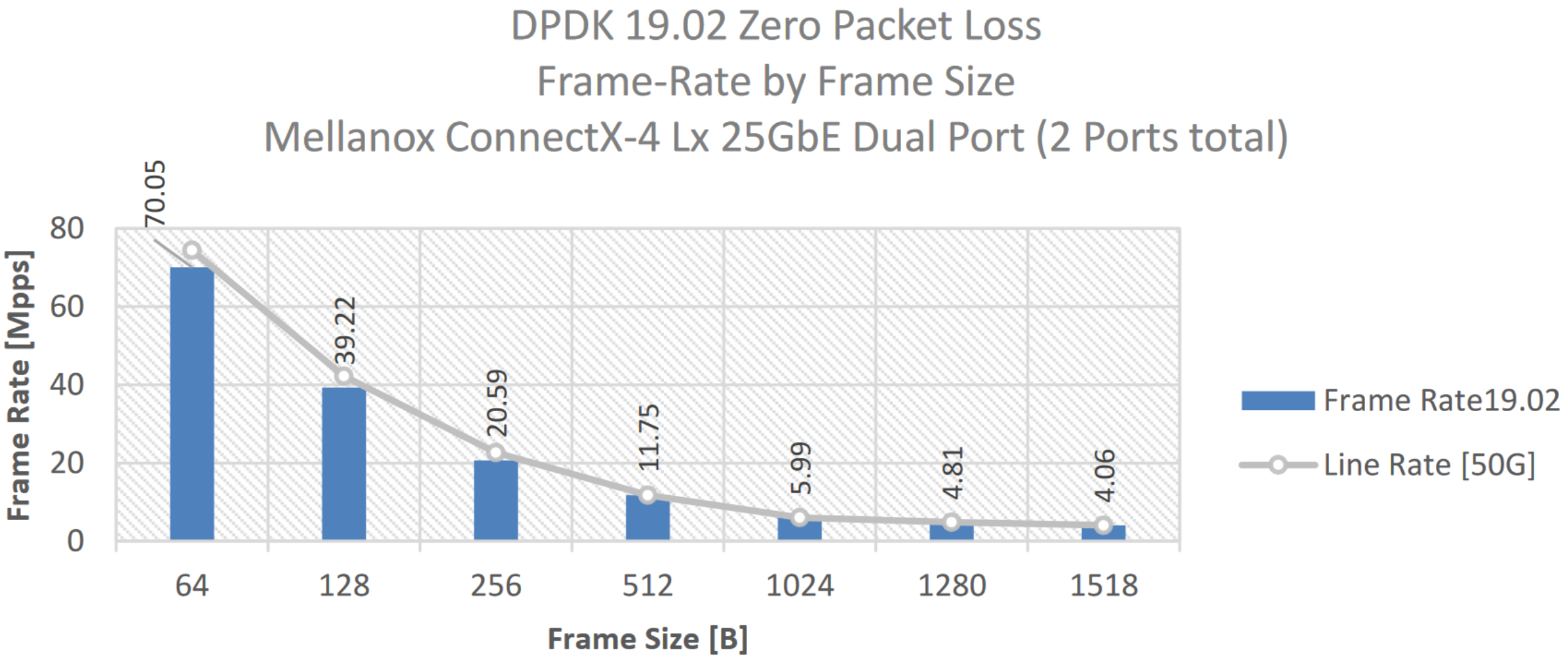
- Data Plane Development Kit is a collection of libraries with custom API
- Open-source and managed by the Linux Foundation (FD.io)
- No transparent mode (i.e. once driver loaded DPDK takes over the NIC)



DPDK

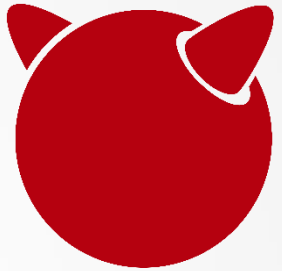
```
/* Run until the application is quit or killed. */
for (;;) {
  /* Receive packets on a port and forward them on the paired port. */
  RTE_ETH_FOREACH_DEV(port) {

    /* Get burst of RX packets, from first port of pair. */
    struct rte_mbuf * bufs[BURST_SIZE];
    const uint16_t nb_rx = rte_eth_rx_burst(port, 0, bufs, BURST_SIZE);
    if (unlikely(nb_rx == 0)) continue;

    /* Send burst of TX packets, to second port of pair. */
    const uint16_t nb_tx = rte_eth_tx_burst(port ^ 1, 0, bufs, nb_rx);

    /* Free any unsent packets. */
    if (unlikely(nb_tx < nb_rx)) {
      uint16_t buf;
      for (buf = nb_tx; buf < nb_rx; buf++)
        rte_pktmbuf_free(bufs[buf]);
    }
  }
}
```

DPDK


DPDK 19.02 Zero Packet Loss
Frame-Rate by Frame Size
Mellanox ConnectX-4 Lx 25GbE Dual Port (2 Ports total)
Frame Rate [Mpps]
80
60
40
20
0
70.05
39.22
20.59
11.75
5.99
4.81
4.06
64
128
256
512
1024
1280
1518
Frame Size [B]
Frame Rate19.02
Line Rate [50G]

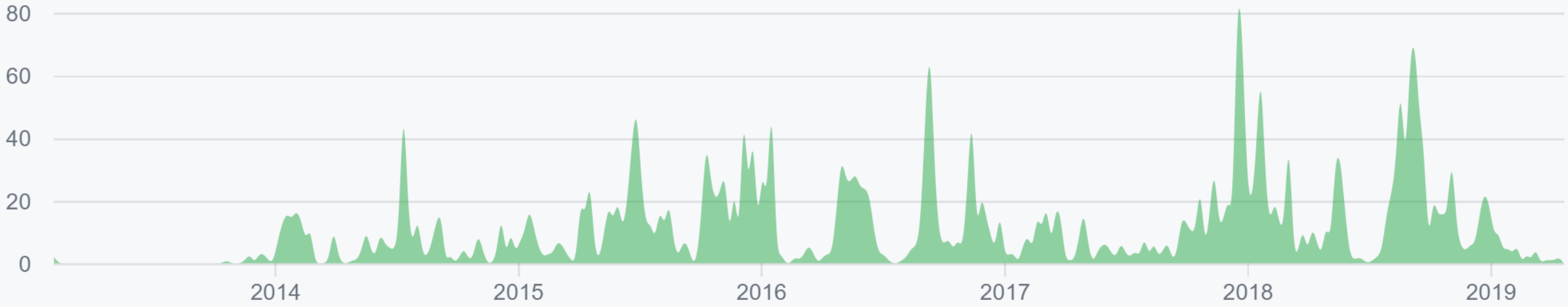
Netmap

- Open-source, part of FreeBSD, external module for Linux/Windows
- 10G with 64bytes packets, 30 Mpps on 40G NICs (160bytes packets)
- New API, rx/tx rings shared by application and NIC
- API is simple and generally interacts with rx/tx rings directly
- Supports both non-blocking I/O and blocking
- Supports select, poll, epoll, kqueue
- Takes exclusive ownership of the NIC (no packets delivered to the OS)



Netmap

- Open-source, part of FreeBSD, external module for Linux/Windows
- 10G with 64bytes packets, 30 Mpps on 40G NICs (160bytes packets)
- New API, rx/tx rings shared by application and NIC



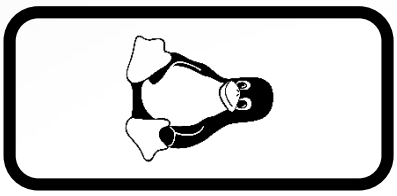
Netmap

```
static int
receive_packets(struct netmap_ring *ring, u_int limit, int dump)
{
        u_int cur, rx, n;
        cur = ring->cur;
        n = nm_ring_space(ring);
        if (n < limit) limit = n;
        for (rx = 0; rx < limit; rx++) {
                struct netmap_slot *slot = &ring->slot[cur];
                char *p = NETMAP_BUF(ring, slot->buf_idx);
                if (dump) dump_payload(p, slot->len, ring, cur);
                cur = nm_ring_next(ring, cur);
        }
        ring->head = ring->cur = cur;

        return (rx);
}
```

User-Space Overview

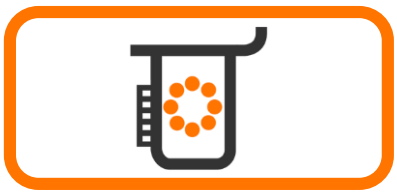
L2 L3 L4

DPDK
DATA PLANE DEVELOPMENT KIT



AF_PACKET, XDP, AF_XDP

netmap



PF_RING

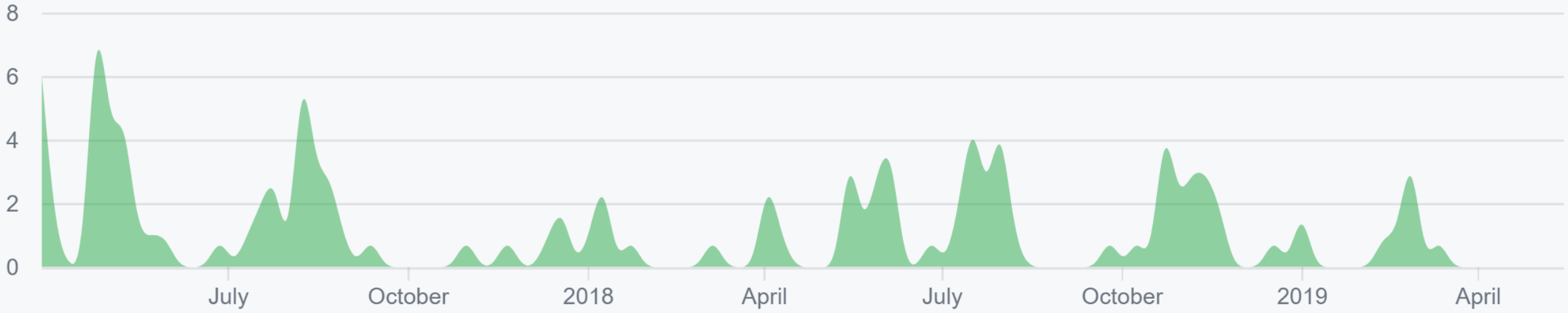
VPP
Vector Packet Processing

F-Stack

- **F-Stack** = **DPDK** + **FreeBSD TCP/IP** + **POSIX** API + **Coroutines**
- Full feature TCP/IP stack, stable and well tested
- Developed by Tencent, deployed in production
- Supports BSD-like socket API (epoll etc.)
- Library level co-routine API via libco
- LD_PRELOAD soon
- No support for IPv6
- Performance claims:
  - 10 million concurrent connections
  - 5 million Request Per Second (RPS)
  - 1 million Connections Per Second (CPS)

F-Stack

- **F-Stack** = **DPDK** + **FreeBSD TCP/IP** + **POSIX** API + **Coroutines**
- Full feature TCP/IP stack, stable and well tested
- Developed by Tencent, deployed in production
- Supports BSD-like socket API (epoll etc.)



- 5 million Request Per Second (RPS)
- 1 million Connections Per Second (CPS)

F-Stack

```
int main() {
  sockfd = ff_socket(AF_INET, SOCK_STREAM, 0);
  ff_ioctl(sockfd, FIONBIO, & on);

  struct sockaddr_in my_addr;
  bzero( & my_addr, sizeof(my_addr));
  my_addr.sin_family = AF_INET;
  my_addr.sin_port = htons(80);
  my_addr.sin_addr.s_addr = htonl(INADDR_ANY);

  ff_bind(sockfd, (struct linux_sockaddr * ) & my_addr, sizeof(my_addr));

  ff_listen(sockfd, MAX_EVENTS);

  ev.data.fd = sockfd;
  ev.events = EPOLLIN;
  ff_epoll_ctl(epfd, EPOLL_CTL_ADD, sockfd, & ev);
  ff_run(loop, NULL);
}
```

- 1 million connections per second (cps)

F-Stack

```
int loop(void * arg) {
  int nevents = ff_epoll_wait(epfd, events, MAX_EVENTS, 0);
  for (int i = 0; i < nevents; ++i) {
     if (events[i].data.fd == sockfd) { /* Handle new connect */; }
     else {
      if (events[i].events & EPOLLERR) { /* Handle error */; }
      else if (events[i].events & EPOLLIN) {
        char buf[256];
        size_t readlen = ff_read(events[i].data.fd, buf, sizeof(buf));
        if (readlen > 0)
          ff_write(events[i].data.fd, html, sizeof(html));
        else { /* Handle close */; } }
      else { /* Handle unknown events */;}
    }
  }
}
```

- 1 million Connections Per Second (CPS)

Seastar

SEASTAR.

- Open-source uses C++14/17, used in ScyllaDB
- Future/Promise/Continuation model, but not `std::promise/std::future`
- Dictates application architecture (much like boost::asio would do)
- Can use native Linux or DPDK as data-plane backend
- Zero-Copy support
- Primary goal is linear scaling performance on multi-core systems (share-nothing architecture)
- Most recently added support for **C++20 co-routines**

Seastar

- Open-source uses C++14/17, used in ScyllaDB
- Future/Promise/Continuation model, but not `std::promise/std::future`

SEASTAR.

- Most recently added support for **C++20 co-routines**

Seastar

- Open-source uses C++14/17, used in ScyllaDB

```
int main(int ac, char ** av) {
  server s;
  app_template app;
  // configure application
  return app.run_deprecated(ac, av, [ & app, & s] {
    auto && config = app.configuration();
    auto chunk_size = config["chunk-size"].as < int > ();
    auto mem_size = (size_t) config["mem-size"].as < int > () * MB;
    auto copy = config.count("copy");
    s.start(chunk_size, copy, mem_size);
  });
}
```

- Most recently added support for **C++20 co-routines**

Seastar

```
class server {
  private:
  udp_channel _chan;
  uint64_t _n_sent {};
  size_t _chunk_size;
  std::vector < packet > _packets;
  std::unique_ptr < output_stream < char >> _out;
  size_t _packet_size = 8 * KB;
  char * _mem;
  size_t _mem_size;

  public:
    future < > send(ipv4_addr dst, packet p) {
      return _chan.send(dst, std::move(p)).then([this] {
        _n_sent++;
      });
    }
  void start(int chunk_size, bool copy, size_t mem_size);
};
```

Seastar

```
void server::start(int chunk_size, bool copy, size_t mem_size) {
  _chan = engine().net().make_udp_channel(ipv4_addr(), 10000));
_out = std::make_unique <output_stream<char>> (
  data_sink(std::make_unique <vector_data_sink> (_packets)), _packet_size);
_mem = new char[mem_size];
_mem_size = mem_size;

keep_doing([this] {
  return _chan.receive().then([this](udp_datagram dgram) {
    auto chunk = next_chunk();
    scattered_message < char > msg;
    msg.reserve(3);
    msg.append_static(chunk, chunk_size);
    //...
    return send(dgram.get_src(), std::move(msg).release());
  });
});
}
```

R.

Seastar

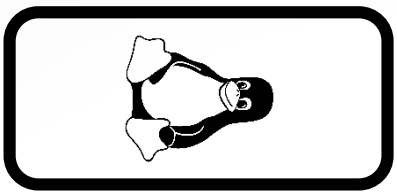
SEASTAR.

- Open-source uses C++14/17, used in ScyllaDB
- Future/Promise/Continuation model, but not `std::promise/std::future`
- Dictates application architecture (much like boost::asio would do)
- Can use native Linux or DPDK as data-plane backend
- Zero-Copy support
- Primary goal is linear scaling performance on multi-core systems (share-nothing architecture)
- Most recently added support for **C++20 co-routines**

User-Space Overview

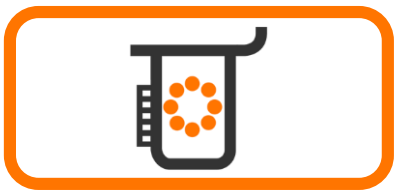
L2 L3 L4

DPDK
DATA PLANE DEVELOPMENT KIT



AF_PACKET, XDP, AF_XDP

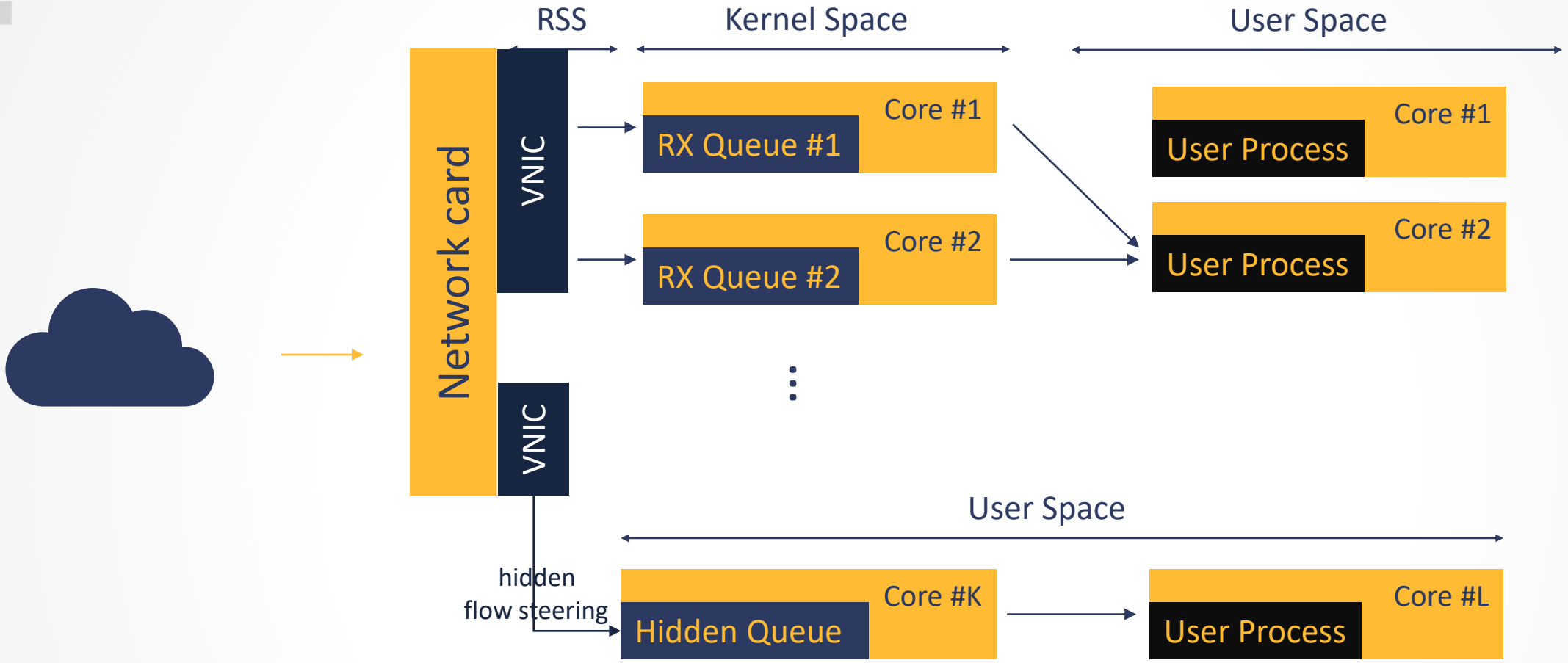
netmap



PF_RING

VPP
Vector Packet Processing

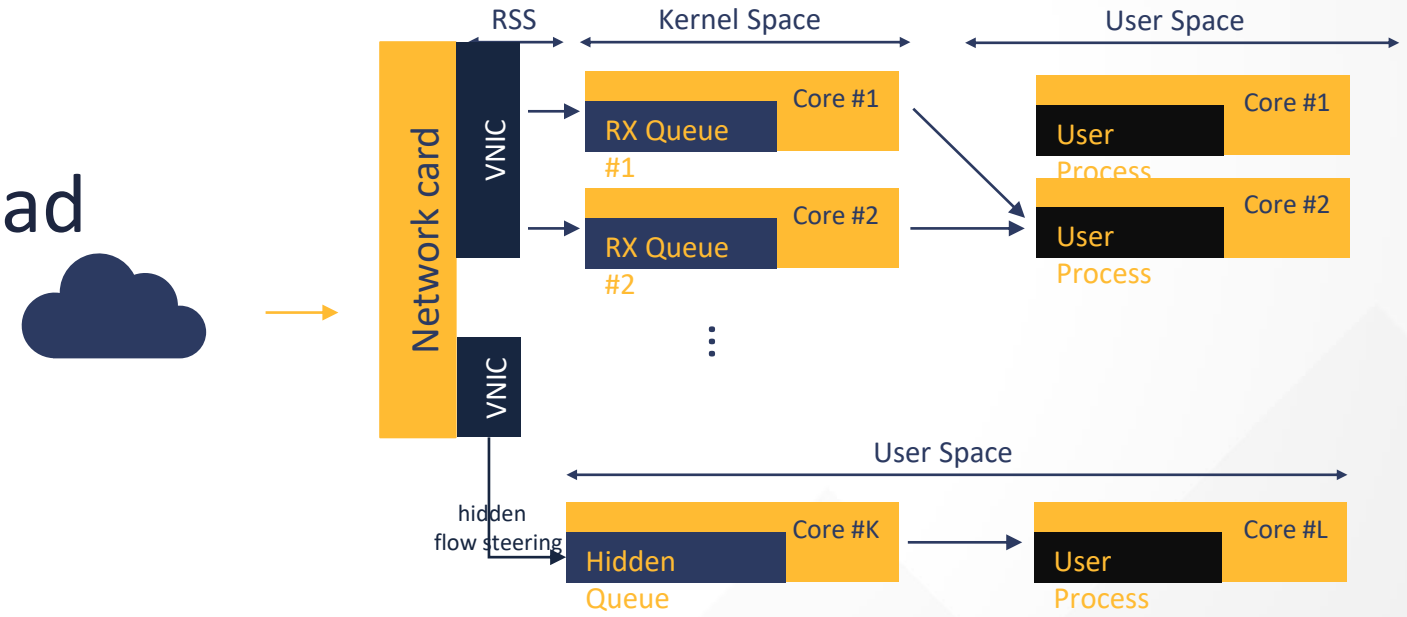
Transparent Kernel-Bypass



Transparent Kernel-Bypass

- LD_PRELOAD accelerated library
- Intercepts subset of libc calls
  (socket, bind, listen, accept, ioctl, recv, poll, epoll, epoll_wait, close, ...)
- Hidden Queue is mapped at user space
- Interface and route configuration taken from kernel

- Mellanox – VMA
- Solarflare – OpenOnload
- Myricom – DBL
- Exablaze – exasock
- Chelsio - WireDirect



Userspace-only Design

- Fully implemented in user space
- Library creates thread(s) to handle background work
- Benefits:
  - Lower latency (<0.7-1.3us us from 3-6 us by kernel*)
- Issues:
  - Extra threads needs attention from user (thread affinity)
  - When application does not exit cleanly, connections not get cleaned up
- Example: VMA from Mellanox

* Depending on settings, packet size

Hybrid Userspace-Kernel Design

- Userspace handles hot-path (send/recv)
- Kernel handles administrative work and async operations
  Examples: TCP timers, socket clean up, etc.
- Accelerated socket is a kernel object with all the guarantees
- No need for (library) background threads
- Connection cleanup is taken care by the kernel

- Example: SF OpenOnload, Exablaze exasock

Vendor specific APIs

Kernel-Bypass API

- Custom APIs provide direct access to Ethernet frames
- Direct access to the buffers that hardware use
- Very high throughput, allows for filtering and forwarding

- Bare metal you need to worry about Layer 2 / Layer 3
- Example use case: Rx Multicast (UDP) packet capture

- Example: Solarflare ef_vi, Mellanox VERBS, Exablaze libexanic

Kernel-Bypass API

```
int main(int argc, char* argv[])
{
  pthread_t thread_id;
  struct resources* res;

  /* Open driver and allocate a VI. */
  ef_driver_open(&res->dh);

  /* Allocate memory for DMA transfers. */
  size_t alloc_size = res->pkt_bufs_n * PKT_BUF_SIZE;
  res->pkt_bufs = mmap(NULL, alloc_size, PROT_READ | PROT_WRITE,
                       MAP_ANONYMOUS | MAP_PRIVATE | MAP_HUGETLB, -1, 0);

  pthread_create(&thread_id, NULL, monitor_fn, res)

  if( cfg_eventq_wait ) event_loop_blocking(res);
  else if( cfg_fd_wait )event_loop_blocking_poll(res);
  else if( cfg_low_latency ) event_loop_low_latency(res);
  else event_loop_throughput(res);
}
```

Kernel-Bypass API

```
static void event_loop_throughput(struct resources * res) {
  const int ev_lookahead = EV_POLL_BATCH_SIZE + 7;
  while (1) {
    refill_rx_ring(res);
    if (ef_eventq_has_many_events( & (res - > vi), ev_lookahead) ||
      (res - > batch_loops) -- == 0) {
      poll_evq(res);
      res - > batch_loops = 100;
    }
  }
}

static void event_loop_low_latency(struct resources * res) {
  while (1) {
    refill_rx_ring(res);
    poll_evq(res);
  }
}
```

```
static int poll_evq(struct resources * res) {
  ef_event evs[EV_POLL_BATCH_SIZE];
  ef_request_id ids[EF_VI_RECEIVE_BATCH];
  int i, j, n_rx;
  int n_ev = ef_eventq_poll( & res - > vi, evs, EV_POLL_BATCH_SIZE);
  for (i = 0; i < n_ev; ++i) {
    switch (EF_EVENT_TYPE(evs[i])) {
    case EF_EVENT_TYPE_RX:
      handle_rx(res, EF_EVENT_RX_RQ_ID(evs[i]),
        EF_EVENT_RX_BYTES(evs[i]) - res - > rx_prefix_len);
      break;
    case EF_EVENT_TYPE_RX_MULTI:
    case EF_EVENT_TYPE_RX_MULTI_DISCARD:
    case EF_EVENT_TYPE_RX_DISCARD: /* ... */ break;
    default:
      LOGE("ERROR: unexpected event type=%d\n", (int) EF_EVENT_TYPE(evs[i]));
      break;
    }
  }
  return n_ev;
}
```

Kernel-Bypass API

```
static void handle_rx(struct resources * res, int pkt_buf_i, int len) {
  struct pkt_buf * pkt_buf;
  LOGV("PKT: received pkt=%d len=%d\n", pkt_buf_i, len);
  pkt_buf = pkt_buf_from_id(res, pkt_buf_i);

  /* Do something useful with packet contents here! */
  if (cfg_hexdump)
    hexdump(pkt_buf - > rx_ptr, len);

  pkt_buf_free(res, pkt_buf);
  res - > n_rx_pkts += 1;
  res - > n_rx_bytes += len;
}
```

Kernel-Bypass API

- Custom APIs provide direct access to Ethernet frames
- Direct access to the buffers that hardware use
- Very high throughput, allows for filtering and forwarding

- Bare metal you need to worry about Layer 2 / Layer 3
- Example use case: Rx Multicast (UDP) packet capture

- Example: Solarflare ef_vi, Mellanox VERBS, Exablaze libexanic

Why Use User-Space Network Stacks

- Performance
- Multiple programming models
- Community driven
- Quick releases
- User-space toolchain
- Isolate Control and Data plane

Challenges

- High barrier to enter, complex non BSD-like APIs
- Code quality is sometimes questionable (depending on framework, not in general!)
- Commercial support might not be available
- No standard, porting to another library is non-trivial
- Vendor specific API is tied to vendor's hardware
- Does not interact well with Kernel-space network stack

Thanks!

Andrew Morrow (MongoDB)
Arthur O'Dwyer
Craig Inglis
Nathan Myers

Performance Consideration

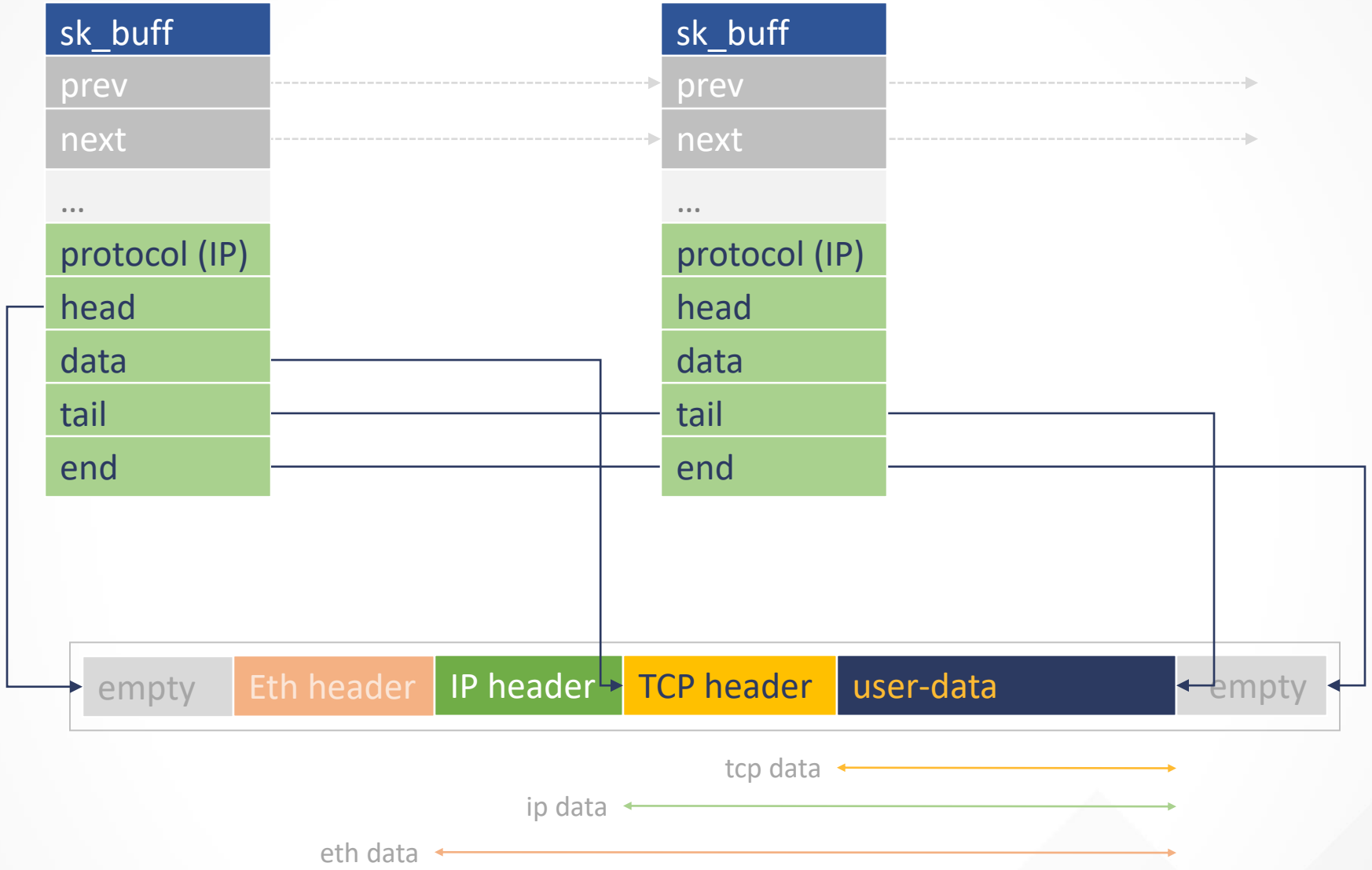
- Not for generic tuning
- Use a software or hardware supported kernel-bypass technology
- Measure! Record performance metrics (hw timestamping)
- Check your NUMA setup
- Disable C-states / SMT / Frequency Scaling in CPU
- Pin critical threads / isolate cores
- Check your interrupt affinity / disable irq balancer
- Disable swapping
- Consider using Huge Pages
- Control the impact of Meltdown, Spectre mitigation
- Profile, profile and profile

epoll

- Usually receiving data from 100+ connection
  E.g.: US Equity Market Data
- Primary goal is to handle line-rate traffic (10GbE) without dropping packets
- Secondary goal is to minimize latency
- epoll is used to monitor multiple file descriptor

- 32 BATS channel for every NASDAQ channel
- Potentially 32 notification for every 1 NASDAQ
- Requires some non-trivial code to handle it
- Fundamentally however you want to simply process each packet as they arrive

sk_buff


sk_buff
prev
next
...
protocol (IP)
head
data
tail
end
sk_buff
prev
next
...
protocol (IP)
head
data
tail
end
empty
Eth header
IP header
TCP header
user-data
empty
tcp data
ip data
eth data

1. RedHat Tunning Guides:
    1. Red Hat Performance Tuning Guide – Networking
    2. Red Hat Enterprise Linux Network Performance Tuning Guide
    3. Red Hat - Linux Network Receive StackMonitoring and Tuning Deep Dive
    4. Red Hat Developer - Achieving high-performance, low-latency networking with XDP
2. Network Stacks
    1. DPDK
    2. github – dpdk
    3. VPP - fd.io
    4. tldk - fd.io
    5. [Video] FD.io - DPDK Overview
    6. PR_RING
    7. github - pf_ring
    8. F-Stack
    9. OpenFastPath
    10. Seastar.io
    11. gtihub - mtcp
    12. io_uring
    13. man7 - AF_PACKET
    14. [Video] AF_PACKET V4 (pre-AF_XDP)
    15. AF_XDP
    16. FreeBSD – netmap
    17. github – netmap

1. Vendor Specific Network Stacks
    1. Introduction OpenOnload White Paper
    2. OpenOnload
    3. Mellanox VMA
    4. Exablaze - exasock
2. The Linux Kernel
    1. Understanding the Linux Kernel, 3rd Edition
    2. Linux Kernel Development (3rd Edition)
3. Others
    1. cloudflare - Kernel bypass
    2. fd.io
    3. Technische Universität München - A Look at Intel's Dataplane Development Kit
    4. TechTalks - Receive Side Scaling and Receive Packet Steering
    5. packagecloud:blog - Monitoring and Tuning the Linux Networking Stack: Receiving Data
    6. CLÉMENT BERTIER - Linux Kernel Packet Transmission Performance in High-speed Networks
    7. Fermilab - Potential Performance Bottleneck in Linux TCP
    8. Technische Universität München - Comparison of Frameworksfor High-Performance Packet IO
    9. Operation Costs In Cpu Clock Cycles
    10. [Video] The brief case for User-space Network Stacks