Life of a packet in OS

Matěj Grégr

igregr@fit.vutbr.cz

Agenda

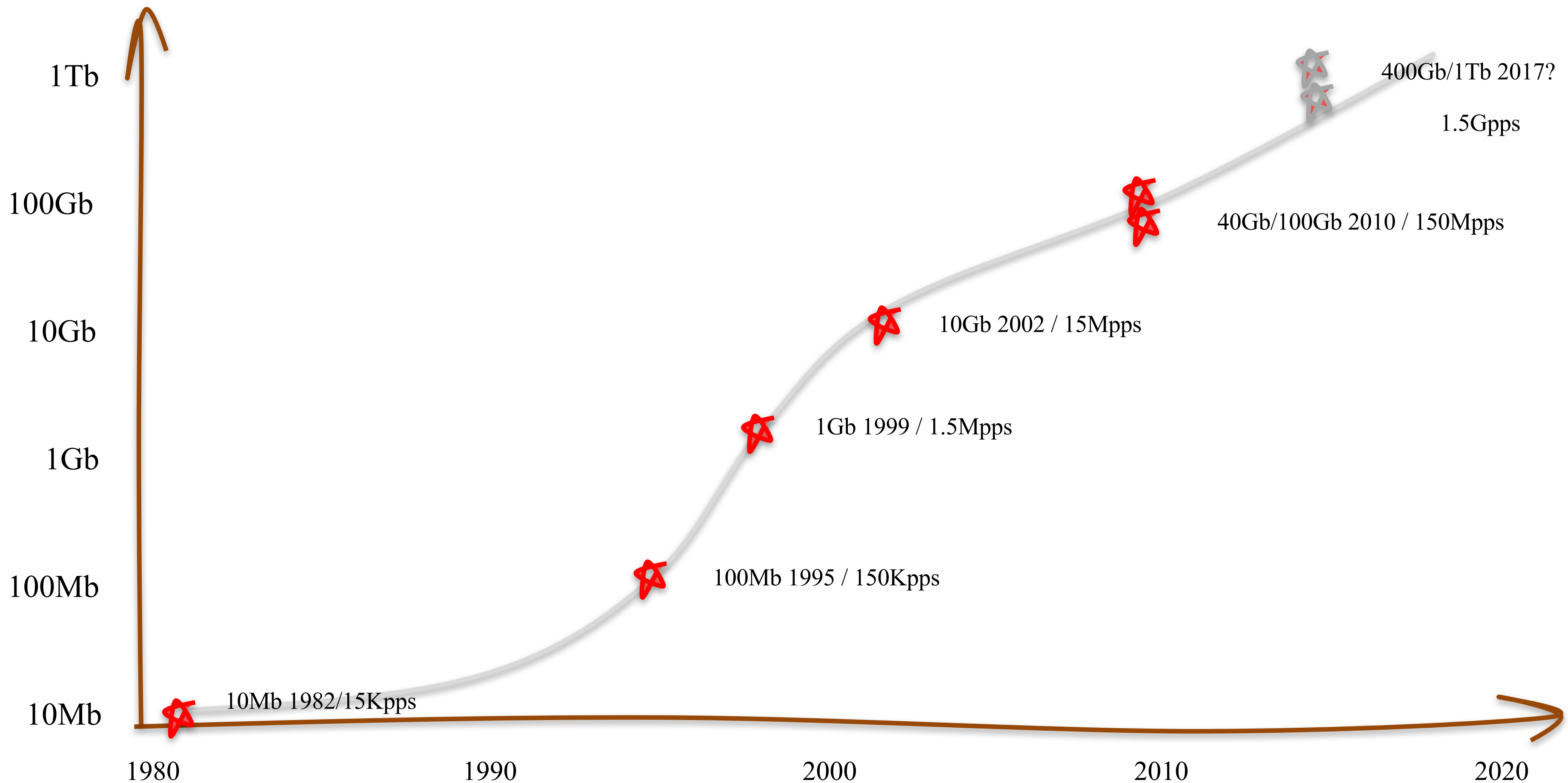
- Ethernet
- Packet processing
- Hooks (NAT/Firewall)
- Speed
- Acceleration techniques

Ethernet

Ethernet!

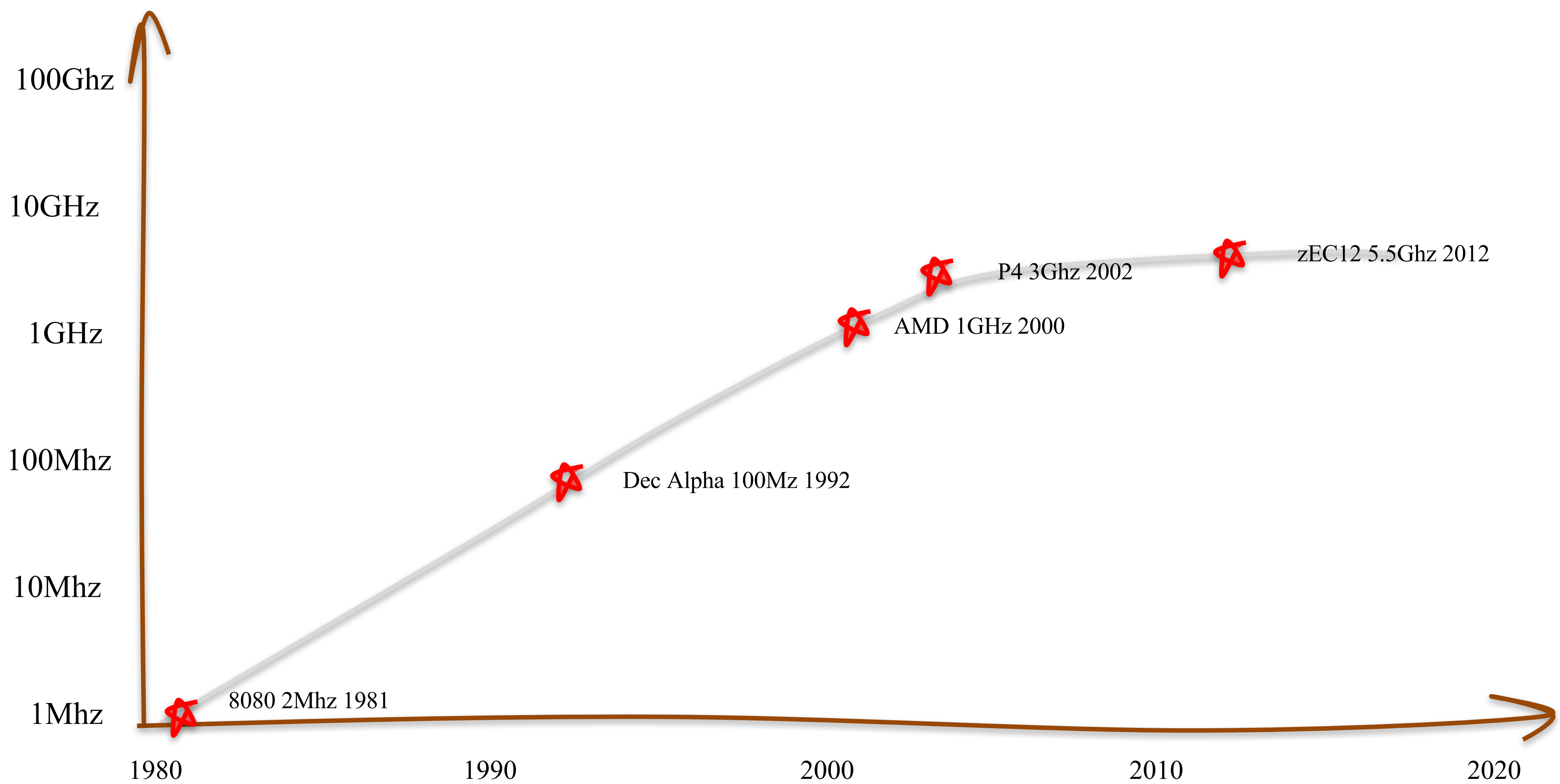
- The same connector since 1976
- Scaled from 10 Mbps to 10Gbps with complete backwards compatibility
- Can supply power.

Ethernet Speed



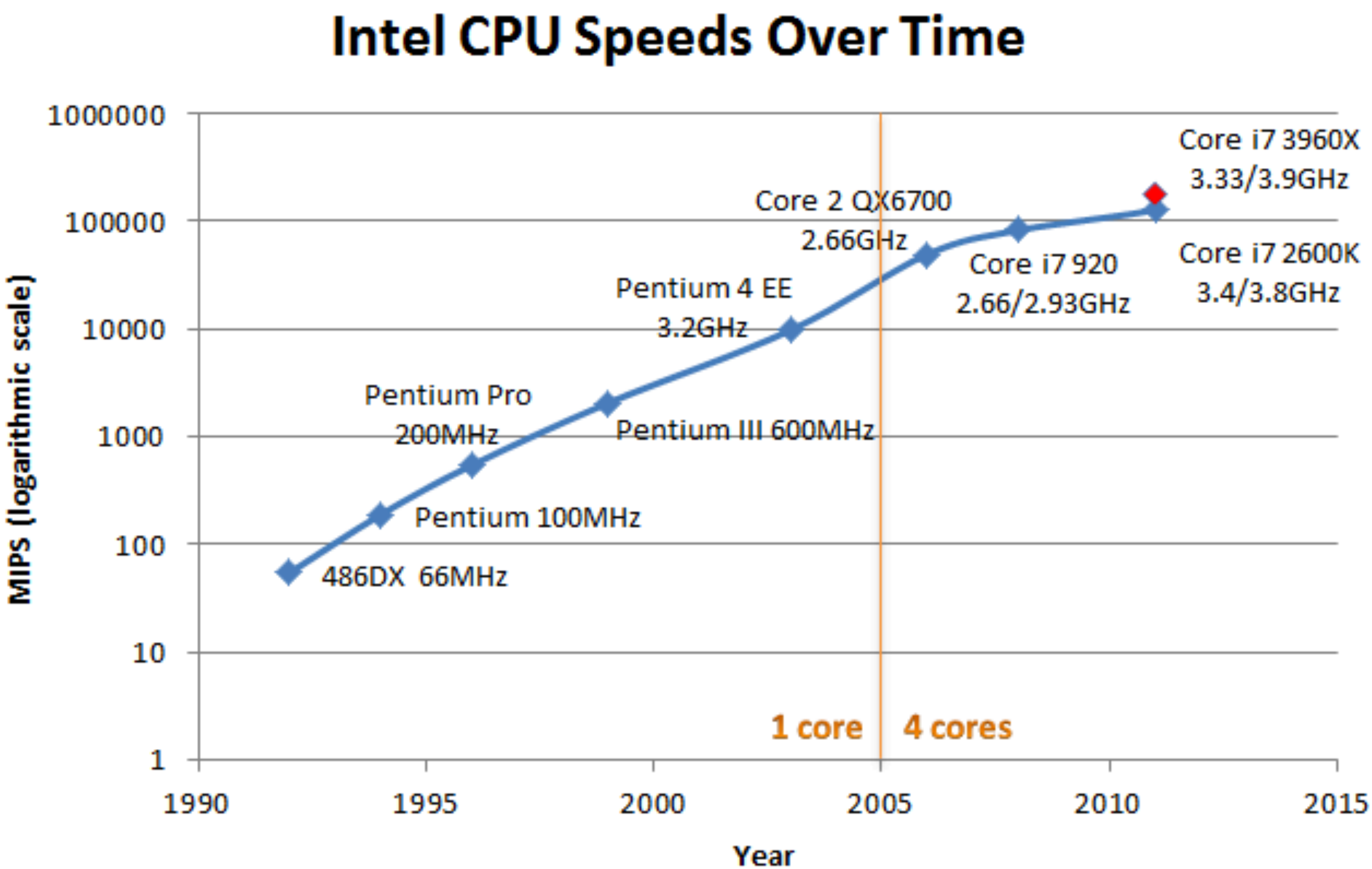
Source: Geoff Huston: Routing 2014, APRICOT 2015

Clock Speed – Processors



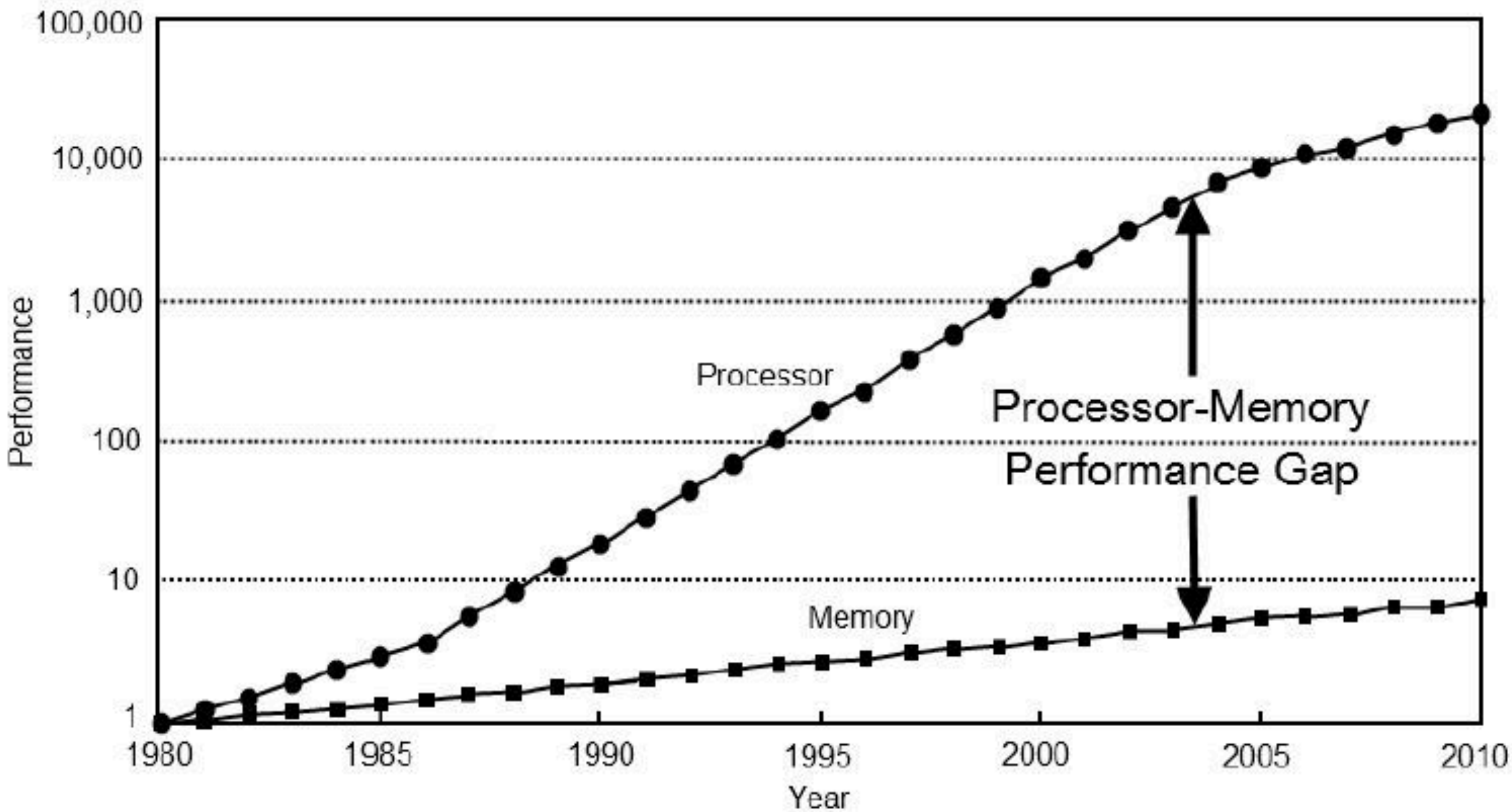
Source: Geoff Huston: Routing 2014, APRICOT 2015

Clock Speed – Processors



Source: Geoff Huston: Routing 2014, APRICOT 2015

CPU vs Memory Speed



Source: Geoff Huston: Routing 2014, APRICOT 2015

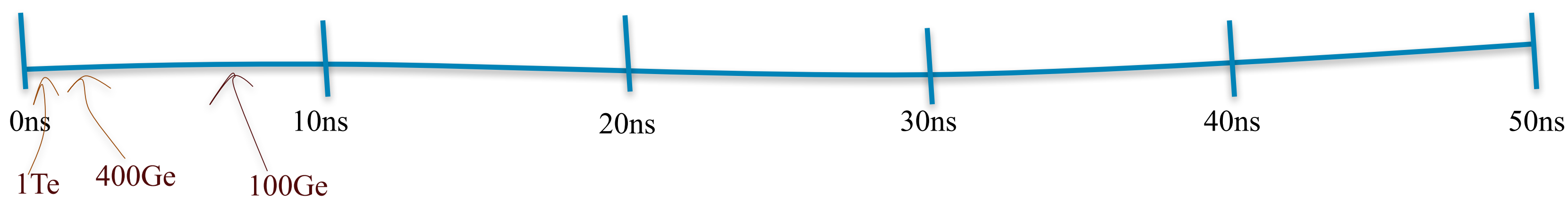
Speed, Speed, Speed

What memory speeds are necessary to sustain a maximal packet rate?

100GE ~ 150Mpps ~ 6.7ns per packet

400Ge ~ 600Mpps ~ 1.6ns per packet

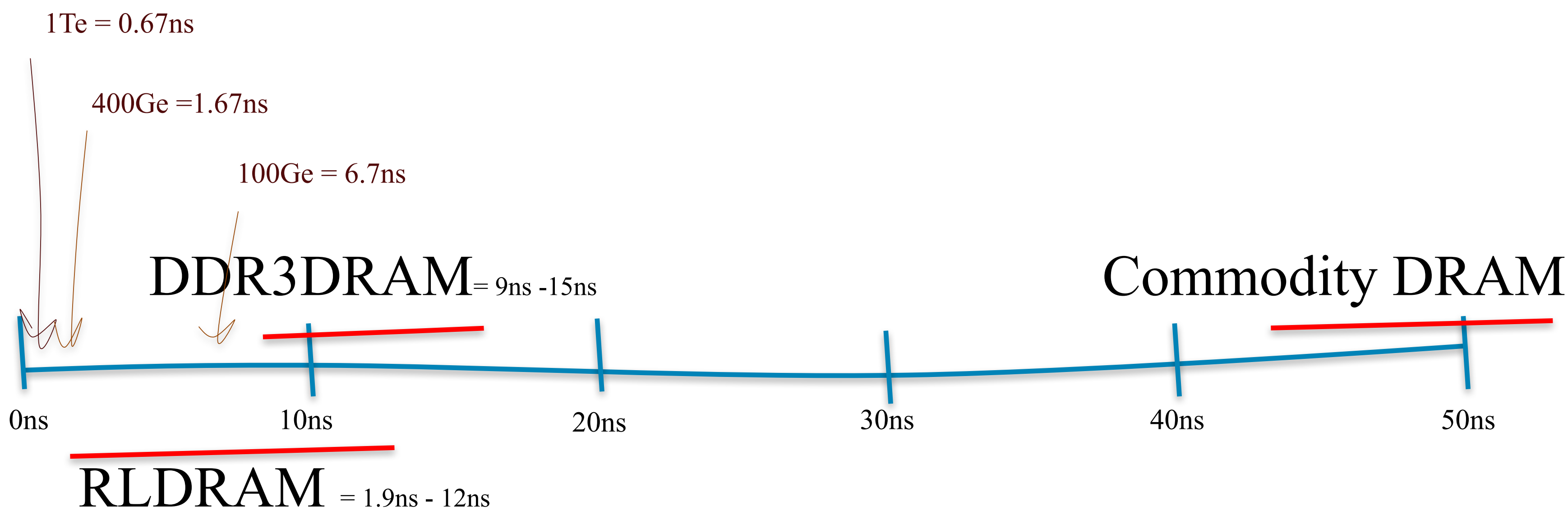
1Te ~ 1.5Gpps ~ 0.67ns per packet



Source: Geoff Huston: Routing 2014, APRICOT 2015

Speed, Speed, Speed

What memory speeds do we have today?



Source: Geoff Huston: Routing 2014, APRICOT 2015

Ethernet

- 10Mb Ethernet had a 64 byte min packet size, plus preamble plus inter-packet spacing

  =14,880 pps

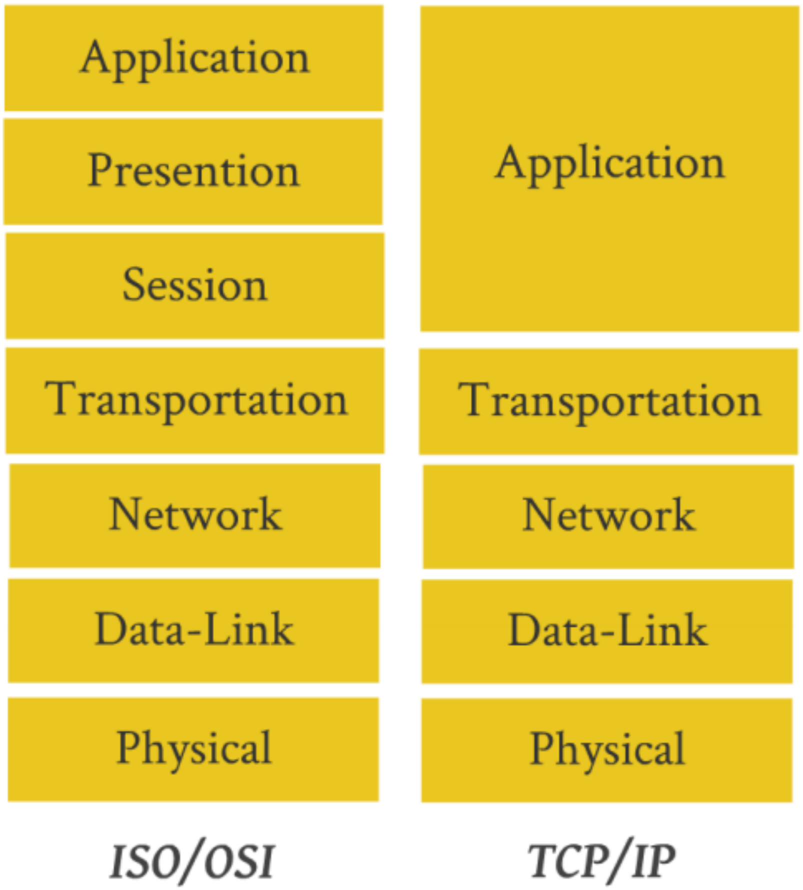
  =1 packet every 67usec

- The speed of circuits was increased, but the Ethernet framing and packet size limits were left unaltered

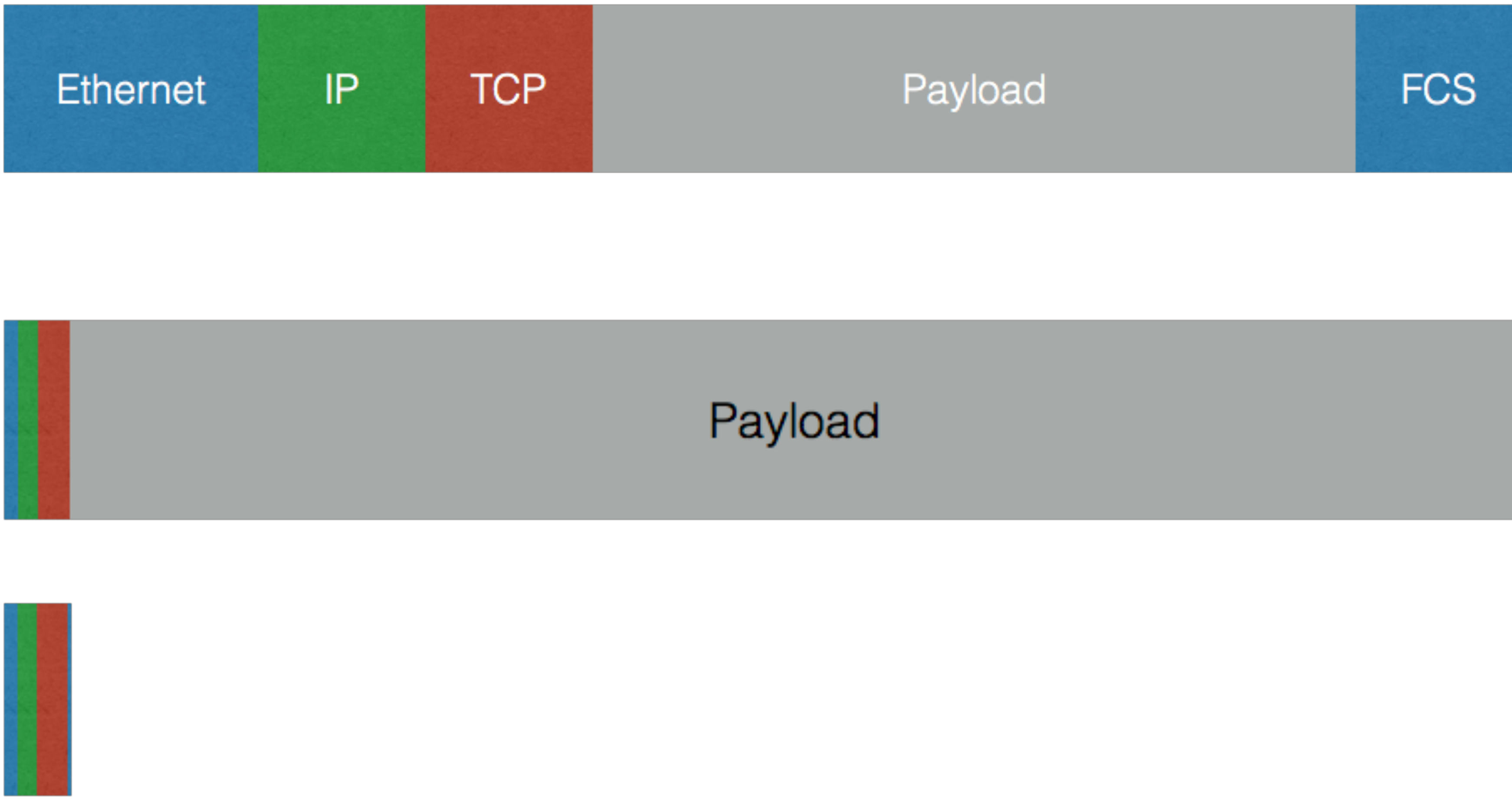
- This have several consequences for processing packets in OS
- It is necessary to increase the speed, but how?

Basic packet processing

Models in theory



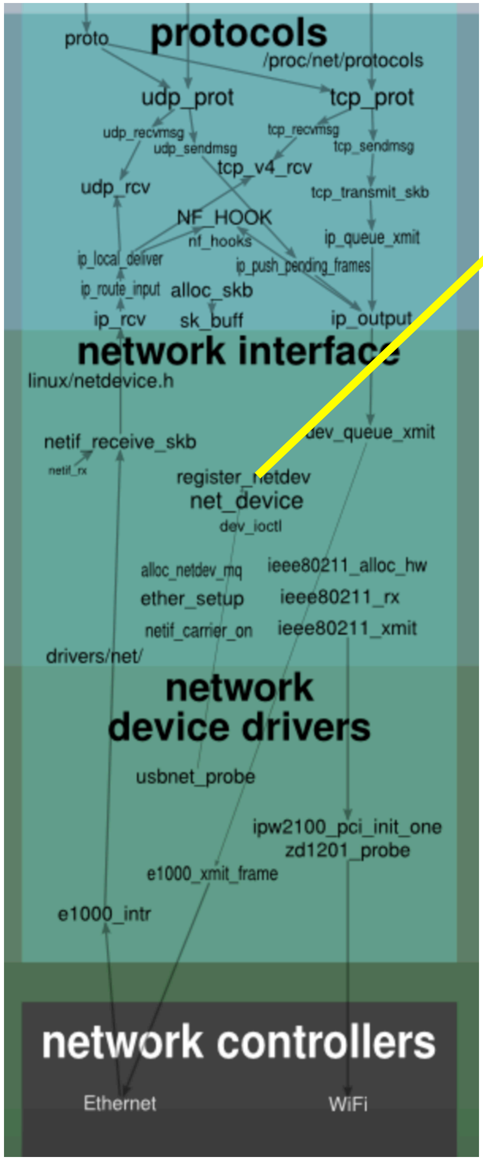
Packets – theory/reality



Networking stack in OS

- Copy the theoretical model – code is divided into layers
- OS typically takes care about layers L2 – L4
- Higher layers are handled by userspace applications
- L1 is handled by HW

Linux Network Architecture



http://www.makelinux.net/kernel_map/

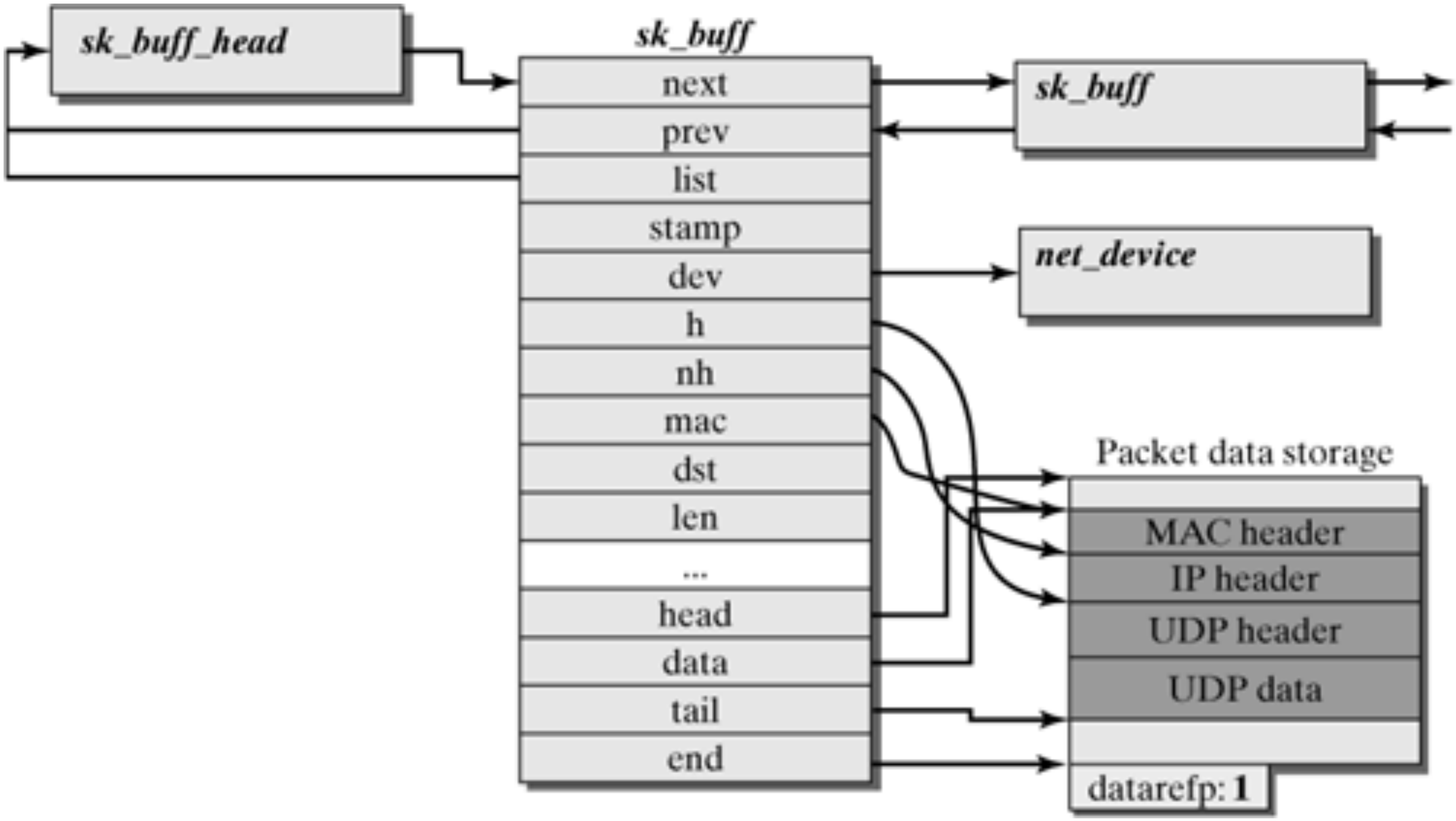
- Network device is represented in `net_device` structure
  - IRQ number
  - MTU
  - MAC address of the device
  - HW features (offloads …)
  - `promiscuity` counter
  - callbacks → function pointers (start/stop device, start trasmit …)
  - `ethtool` callbacks

sk_buff

- Representation of a packet in the Linux kernel
- SKB API – provides set of functions how to access to specific pointers in SKB structure (transport, network)
- Packet is received → `netdev_alloc_skb()`

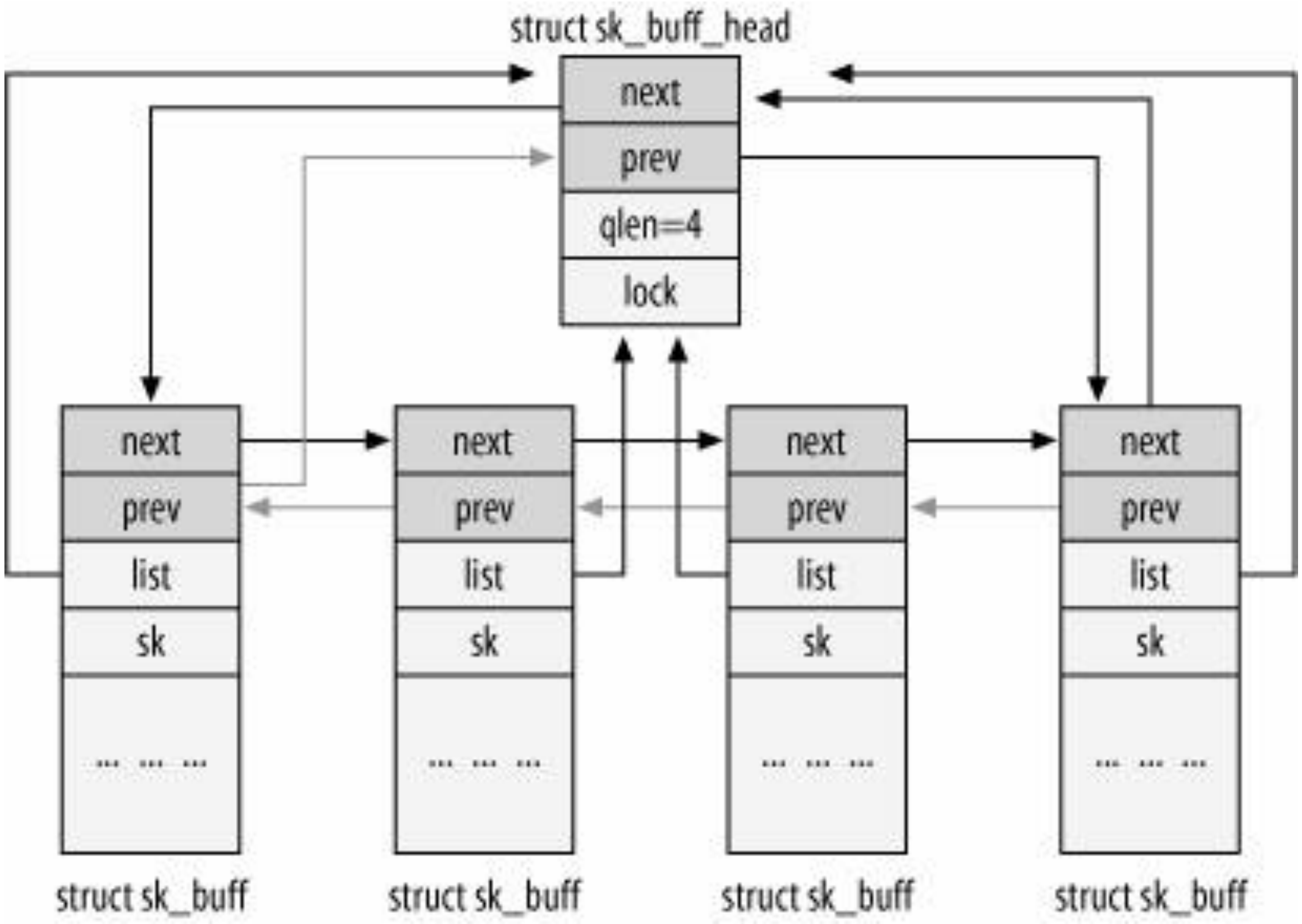
```
struct sk_buff {
        . . .
        struct sock             *sk;
        struct net_device       *dev;
        . . .
        __u8                    pkt_type,
        . . .
        __be16                  protocol;
        . . .
        sk_buff_data_t          tail;
        sk_buff_data_t          end;
        unsigned char           *head,
                                *data;

        sk_buff_data_t          transport_header;
        sk_buff_data_t          network_header;
        sk_buff_data_t          mac_header;
        . . .
}
```

sk_buff



source: Linux Networking Architecture

sk_buff



struct sk_buff_head
next
prev
qlen=4
lock
next
prev
list
sk
... ...
struct sk_buff
next
prev
list
sk
... ...
struct sk_buff
next
prev
list
sk
... ...
struct sk_buff
next
prev
list
sk
... ...
struct sk_buff

Tx/Rx path

Tx Path

- Sending process context
- Kernel adds:
  - Transport header
  - Network header
  - Link header
  - Driver
  - IRQ

Rx Path

- Different contexts
- IRQ handler
- Softirq
- Network layer - ip_recv()
- Transport layer - tcp_v4_rcv()
- Queue – tcp_data_queue()
- recvmsg
- Copy to user

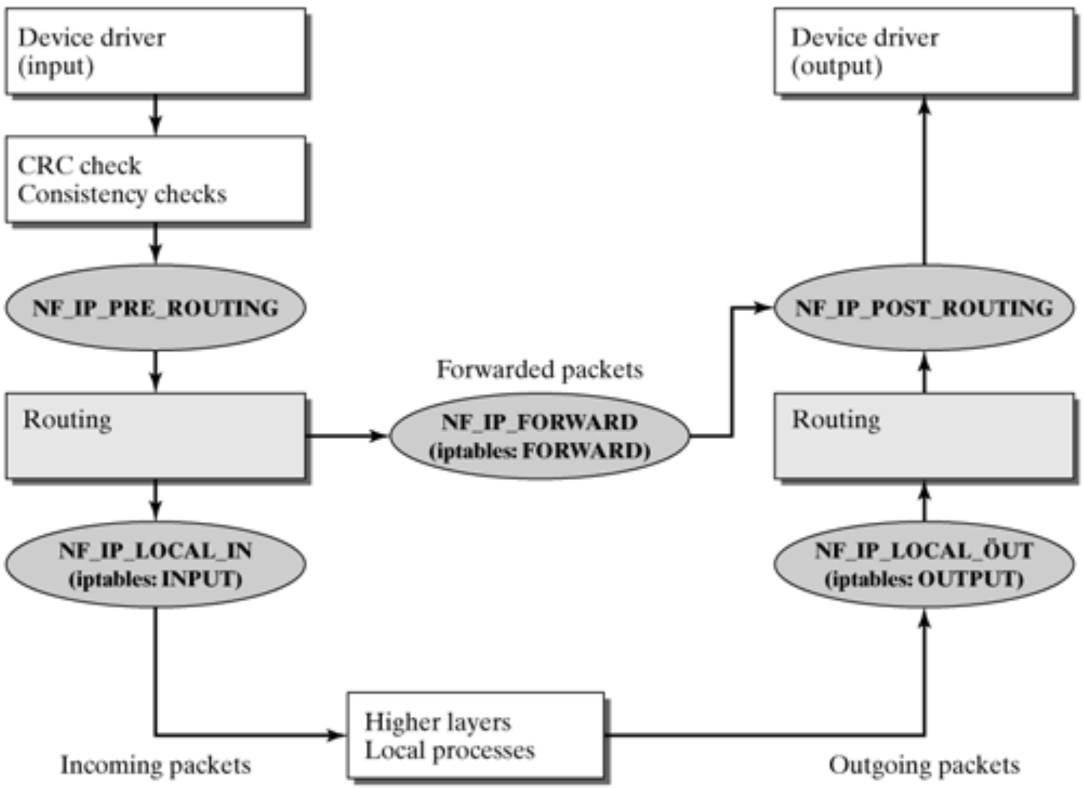
Receiving data

- Data is received by the NIC from the network.
- The NIC uses DMA to write the network data to RAM (ring buffer).
- The NIC raises an IRQ.
- The device driver’s registered IRQ handler is executed.
- The IRQ is cleared on the NIC, so that it can generate IRQs for new packet arrivals.
- NAPI softIRQ poll loop is started with a call to napi_schedule.

NF_HOOKS

Netfilter

- Uniform interface that provides several hooks for packet-filter code
- Can be loaded into the Linux kernel at runtime



Available hooks

- `NF_IP_PRE_ROUTING`
  - called right after the packet has been received
- `NF_IP_LOCAL_IN`
  - packets that are destined for the host
- `NF_IP_FORWARD`
  - packets not addressed to the host
- `NF_IP_POST_ROUTING`
  - packets that have been routed and are ready to leave
- `NF_IP_LOCAL_OUT`
  - Packets sent out from the host

Available codes

- `NF_ACCEPT`
  - accept the packet (continue network stack trip)
- `NF_DROP`
  - drop the packet (don't continue trip)
- `NF_REPEAT`
  - repeat the hook function
- `NF_STOLEN`
  - hook steals the packet (don't continue trip)
- `NF_QUEUE`
  - queue the packet to userspace

iptables

- Provides API for `netfilter` hooks from an userspace app
- Can be used to implement firewall/NAT or mangle the packets according network admin use cases
- Uses tables to organize its rules (e.g., `nat` table, `filter` table …)
- Within each iptables table, rules are further organized within separate chains
  - `PREROUTING`, `INPUT`, `FORWARD`, `OUTPUT`, `POSTROUTING`
    - mirror the names of the `netfilter` hooks

Iptables – Available tables

- **The Filter Table**
  - Used to make decisions about whether to let a packet continue or drop (firewall)
- **The NAT Table**
  - implements network address translation rules (modifies the packet's source or destination addresses, etc.
- **The Mangle Table**
  - Used to alter the IP headers fields, e.g. adjust the TTL
  - Can add MARK to the packet that can be used in other tables and by other networking tools
- **The Raw Table**
  - Purpose is to provide a mechanism for marking packets in order to opt-out of connection tracking.
- **The Security Table**
  - The security table is used to set internal SELinux security context marks on packets

Speed

Speed, speed, speed

- Kernel needs to optimize the packet processing
- NAPI (New API)
- GSO/GRO
- Offloading
- Multiqueue
- Busy looping (low latency socket)

NAPI – New API

- Requires the following features:
  - DMA ring or enough RAM to store packets in software devices.
  - Ability to turn off interrupts or maybe events that send packets up the stack.
- **Interrupt mitigation**
  - some interrupts are disabled during times of high traffic, with a corresponding decrease in system load.
- **Packet throttling**
  - it's better to drop packets before they hit the kernel
  - packets can be dropped in the network adaptor itself

Busy looping

- Previously called low latency sockets
- Extreme low-latency applications need to get packet as soon as possible – poll is too slow for them
- Re-enabling interrupts is not a workable solution, though
- Application can poll the interface for new packets whenever it is prepared to handle new messages
  - Increasing the CPU usage, but decreasing the latency

Multiqueue (RSS)

- Works for multiple different flows
- NIC has several hardware queues
  - Each queue is mapped to a different CPU that handles the interrupts and traffic
  - OS can achieve much higher throughput
- Hash function is typically used to divide the traffic to different queues
  - IP, IP + ports, custom …
- Depends on the NIC hw, PCI lines, etc
- Problems with tunneling, synchronization, locking, cache trashing
- https://blog.cloudflare.com/how-to-receive-a-million-packets/

GSO/GRO (LSO)

- Most of the time, the kernel works with large chunks of data
- GSO: NIC is responsible to cut the data to TCP segments and packets with the correct MTU size
- GRO: NIC is responsible to collect small data packets to one large
- GSO/GRO provided by NIC or network driver

- Problems
  - Tunnels
  - Can corrupt the data

Offloading

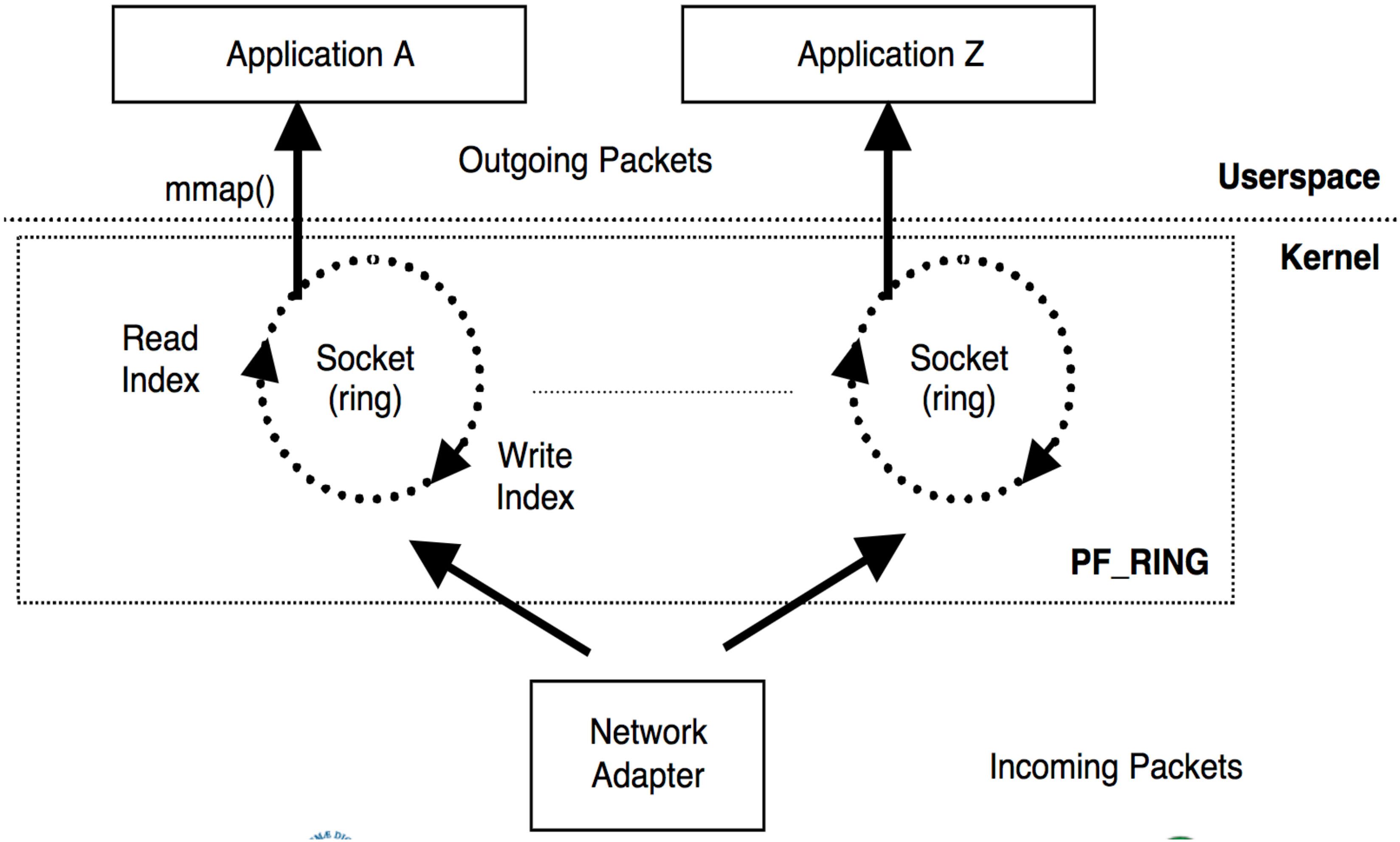
- Several computing functions can be offload to the NIC
- Checksum – especially TCP/UDP (all data checksum)
- VLAN stripping

- Problems with tunneling techniques
  - TCP inside GRE
  - UDP inside VLAN inside VxVLAN inside GRE

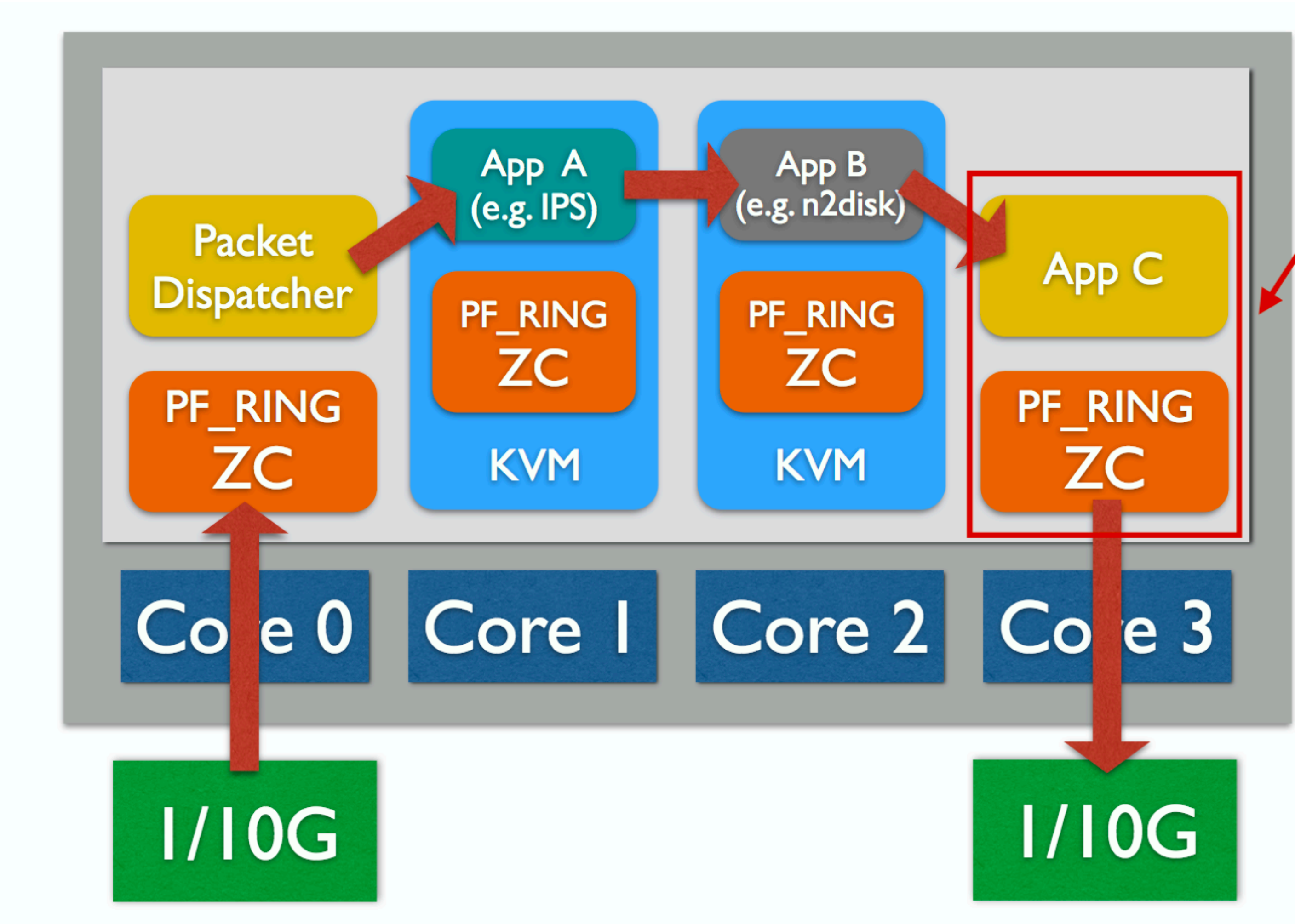
- Strongly depends on the NIC hardware

Accelerating techniques

PF_RING


Application A
Application Z
Outgoing Packets
mmap()
Userspace
Kernel
Read Index
Socket (ring)
Socket (ring)
Write Index
PF_RING
Network Adapter
Incoming Packets

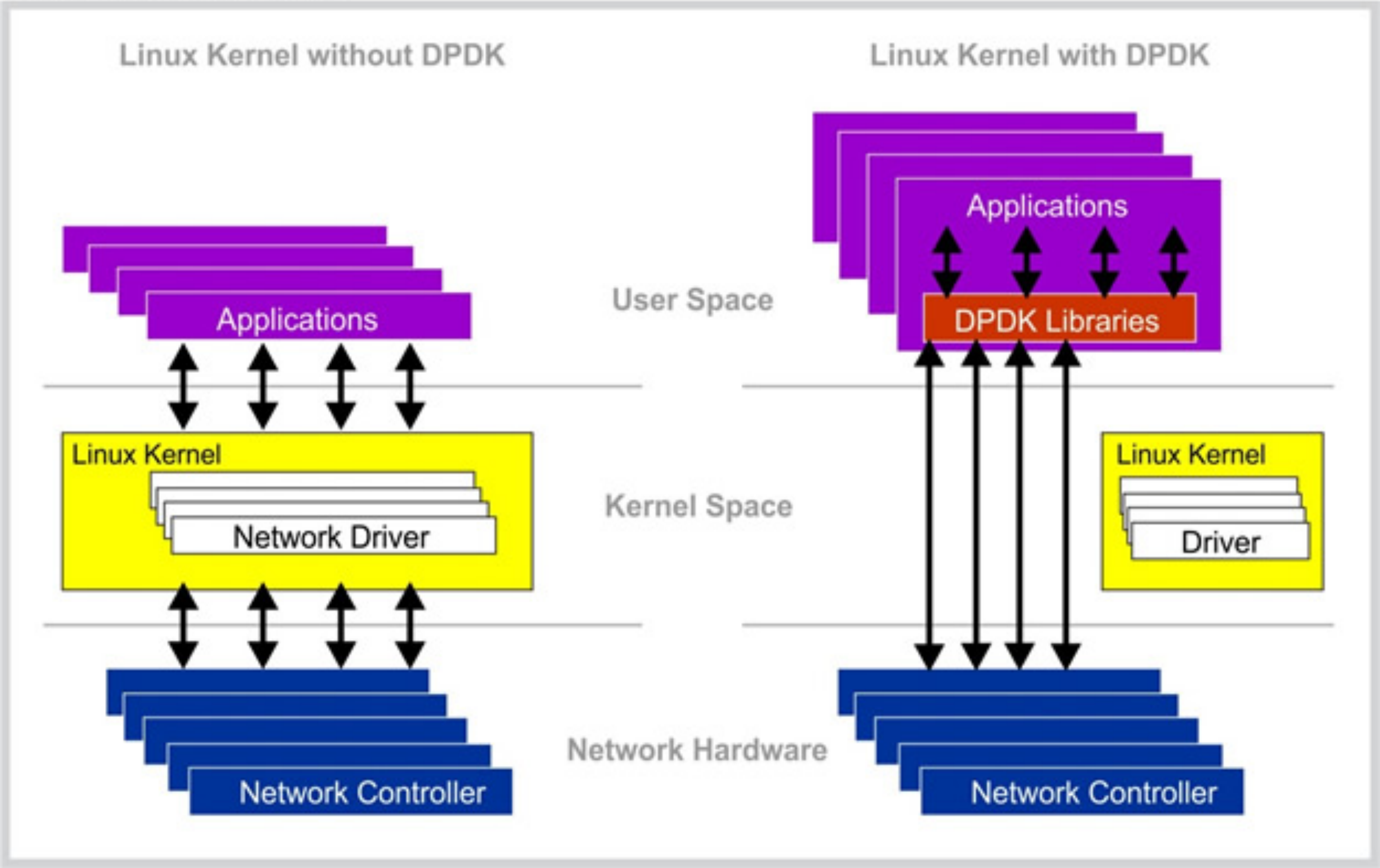
PF_RING ZC



Optional Packet Transmission in 0-copy

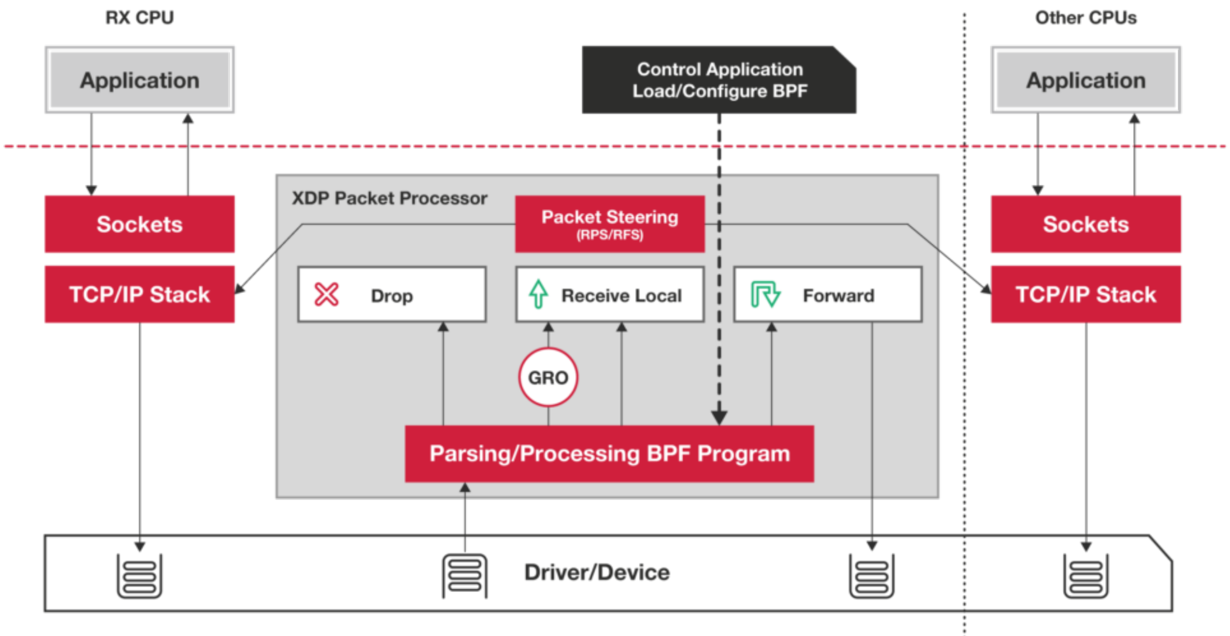
DPDK

Linux Kernel with DPDK



XDP

- A programmable, high performance, specialized application, packet processor in the Linux networking data path
- XDP is not kernel bypass. It is an integrated fast path in the kernel stack



https://www.iovisor.org/technology/xdp