DPDK

Nishanth Shyamkumar

Illinois Institute of Technology

March 2025

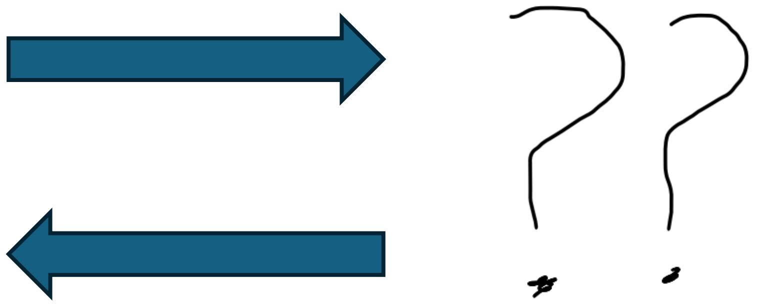
Achieving high network performance on a CPU

End Host Networking

- An ‘end host’ in networks are the end point devices in a network connection.
- These can be your Laptop, Desktop, Mobile phones, Server machines etc.
- They generally rely on Operating Systems to provide networking features
- The main processing hardware is a CPU.

Browsing the web

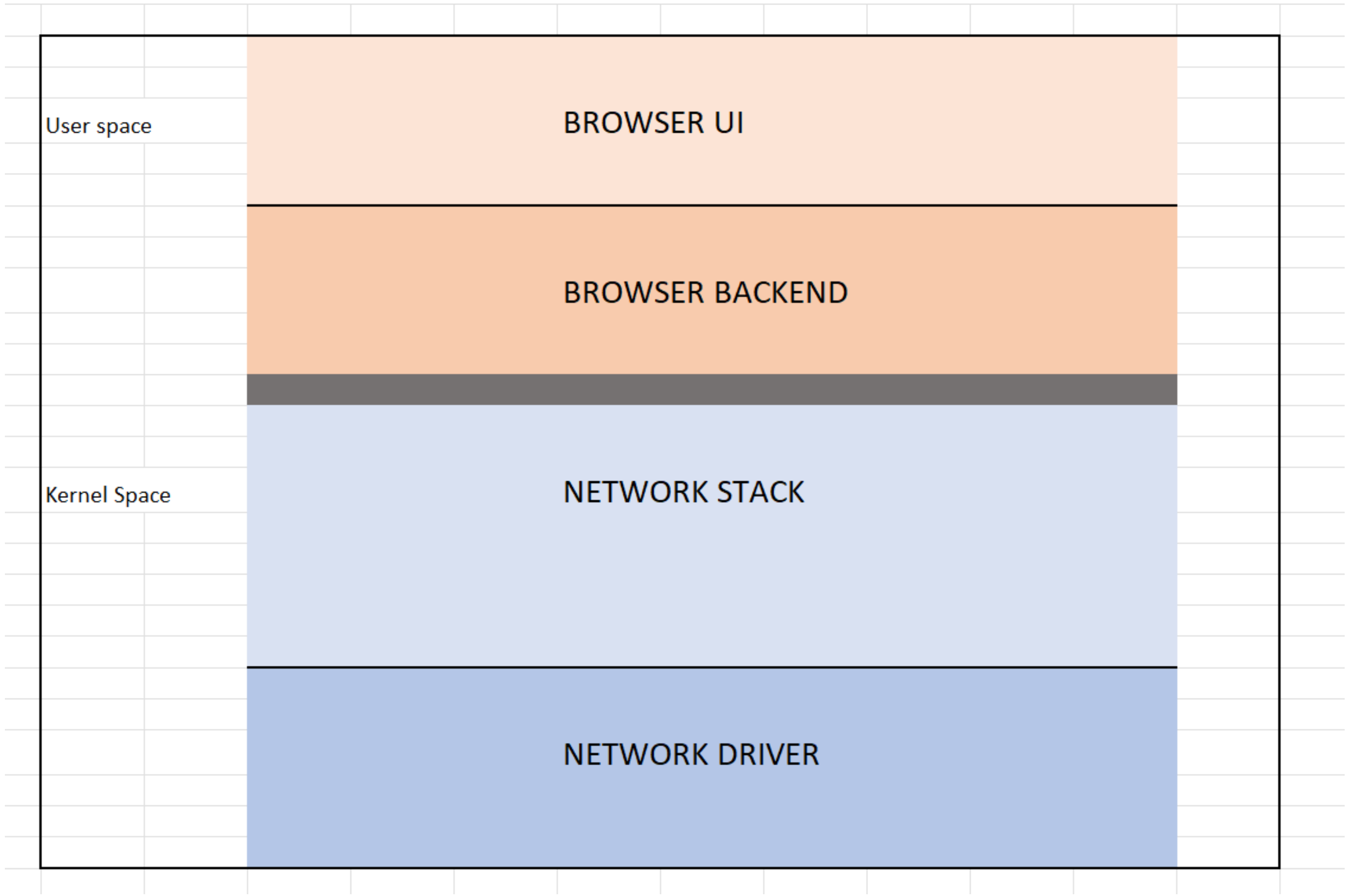
- What happens when you type a URL and click ‘Enter’



System works together

- There is an interplay between the application’s frontend and backend.
- Similarly between the application backend and the OS Kernel
- Finally between the Kernel and the network device
- Specifically, in the case of Mozilla Firefox, they have a backend network engine called ‘Necko’ which deals with Protocol Parsing and using ‘Network Sockets’.

Traditional Browser Stack

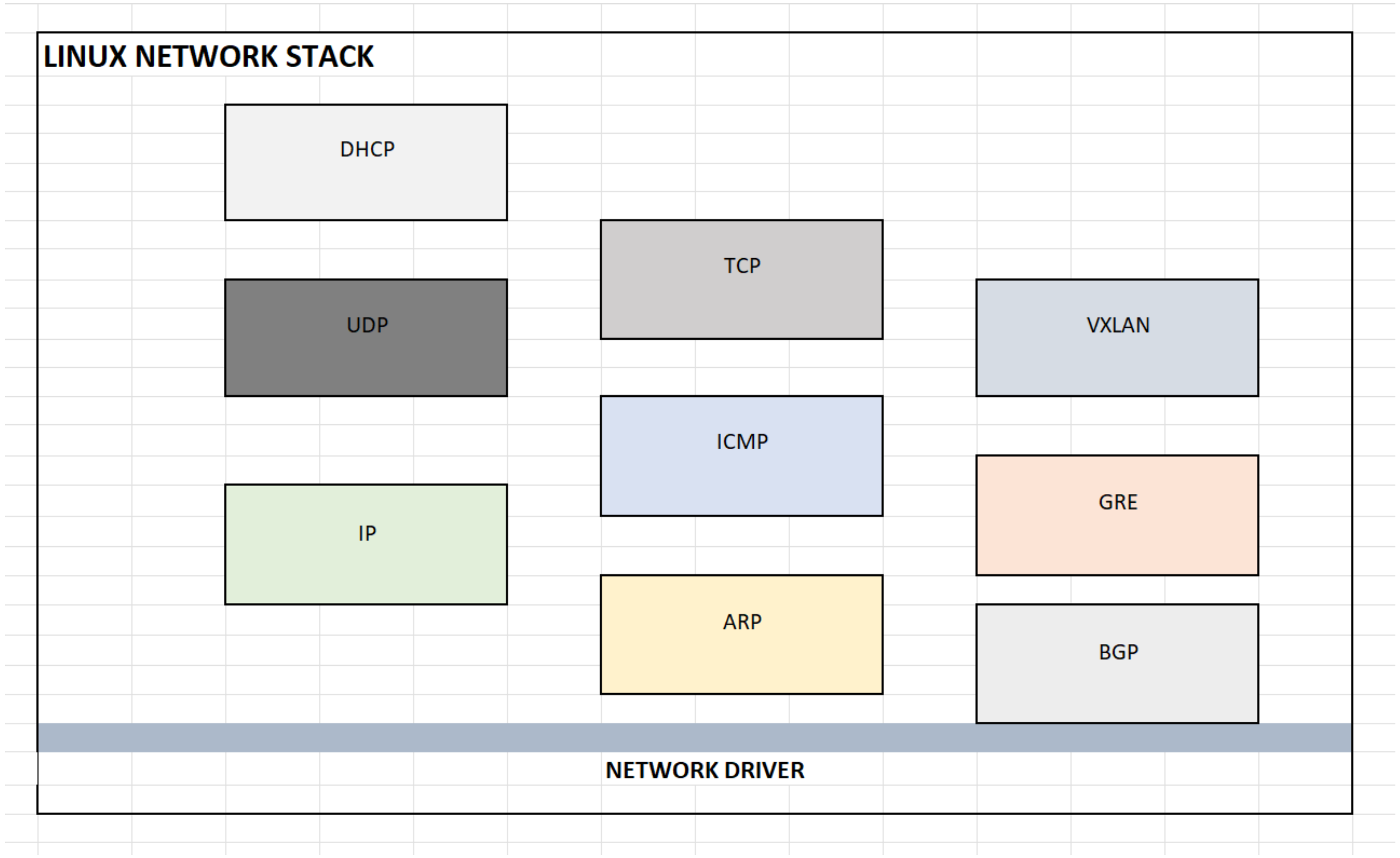


User / Kernel space ??

Linux Network's Generality

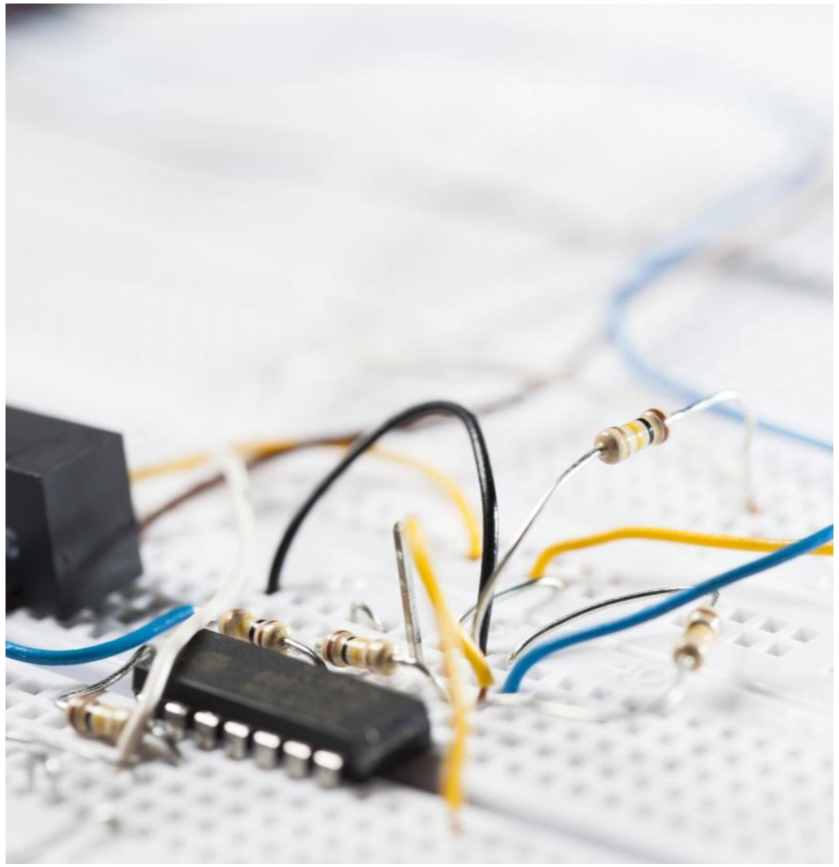
- The Linux Operating System handles multiple network protocols. It HAS to, in order to be useful for all the myriad users.
- Packet header is used by the OS to make decisions on how to handle the packet.
- Increases code complexity(many if-then branches).
- All these conditions and checks are extra instructions running on a CPU.
- Additional instructions for Firewall rule checking.

Linux Networks' Generality



How CPUs affect network performance

- At first following Moore’s Law, transistor density doubled every 2 years
- Single core CPUs kept increasing their clock frequency.
- Higher frequency, more cycles in a unit of time.
- More cycles allows for more instructions to be processed in that unit of time.



How CPUs affect network performance

- Hit Dennard’s scaling and high power draw became a limitation.
- Power *** Voltage^2 * Frequency
- Solution was to use multiple cores running at a lower clock frequency (Multi-processing).
- Meantime, network port speeds increased exponentially from 10Mb/s to 400Gb/s.

Time limitation at high speeds

- Simple mathematical example.

A CPU clock at 2GHz frequency and a network card running at 1Gb/s.

1 CPU cycle = 0.5 nanoseconds (ns)

Time limitation at 1Gb/s throughput

- How many 64B Ethernet packets make 1Gb/s of traffic?

It's 1.48 Mpps (million packets per second).

Amount of time to process 1 packet = 1 / 1.48M ~ 600-700 ns

i.e, for a 2GHz clock frequency = 1200-1400 CPU cycles to process the packet.

What about at 100Gb/s ?

- How many 64B Ethernet packets make 100Gb/s of traffic?

It’s 148 Mpps (million packets per second).

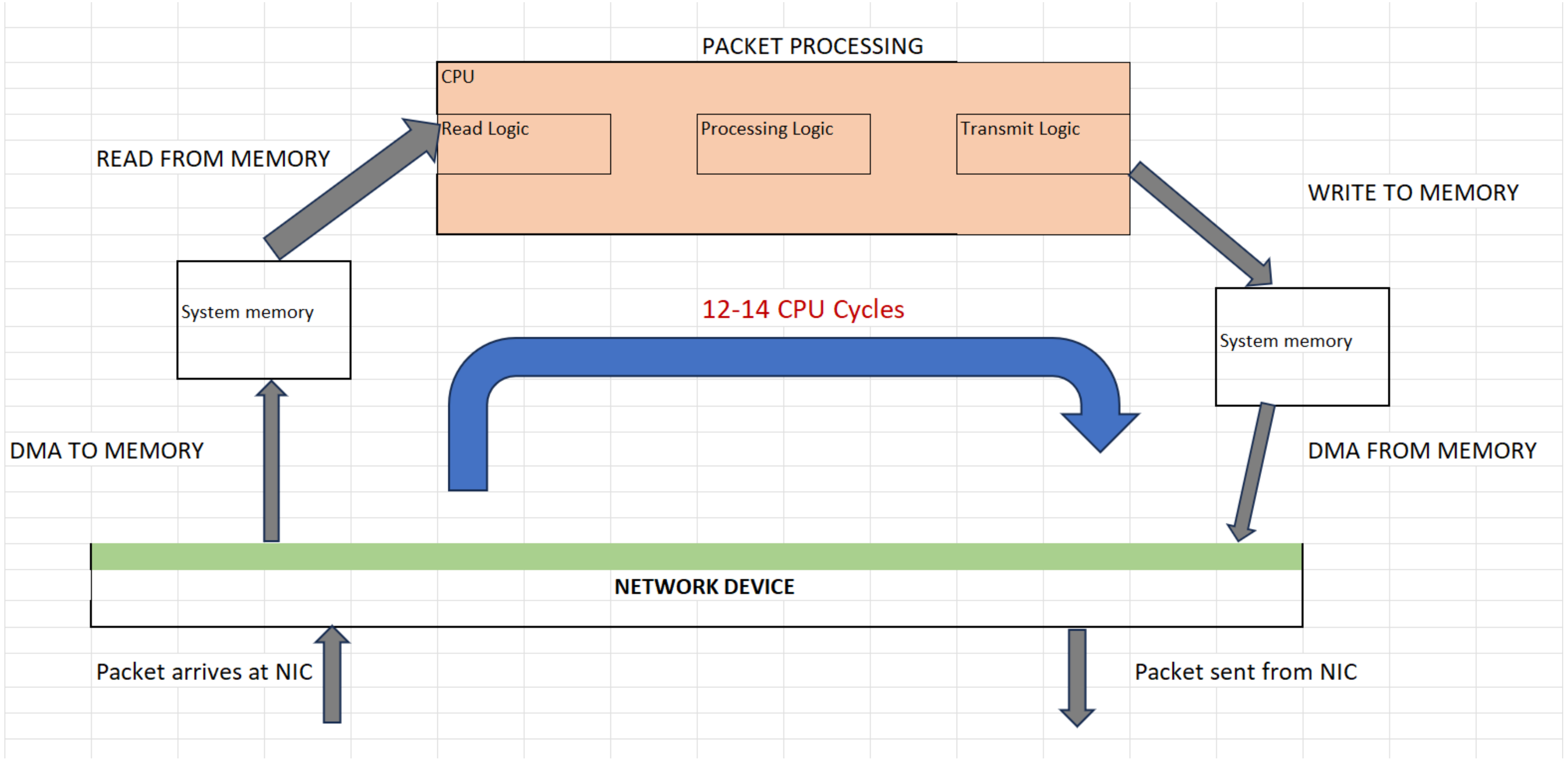
[EC: Can you figure out how it becomes 148Mpps. Think of an Ethernet frame on the wire.]

Amount of time to process 1 packet = 1 / 148M ~ 6-7 ns

i.e, for a 2GHz clock frequency = 12-14 CPU cycles

Packet Lifecycle

DMA ?



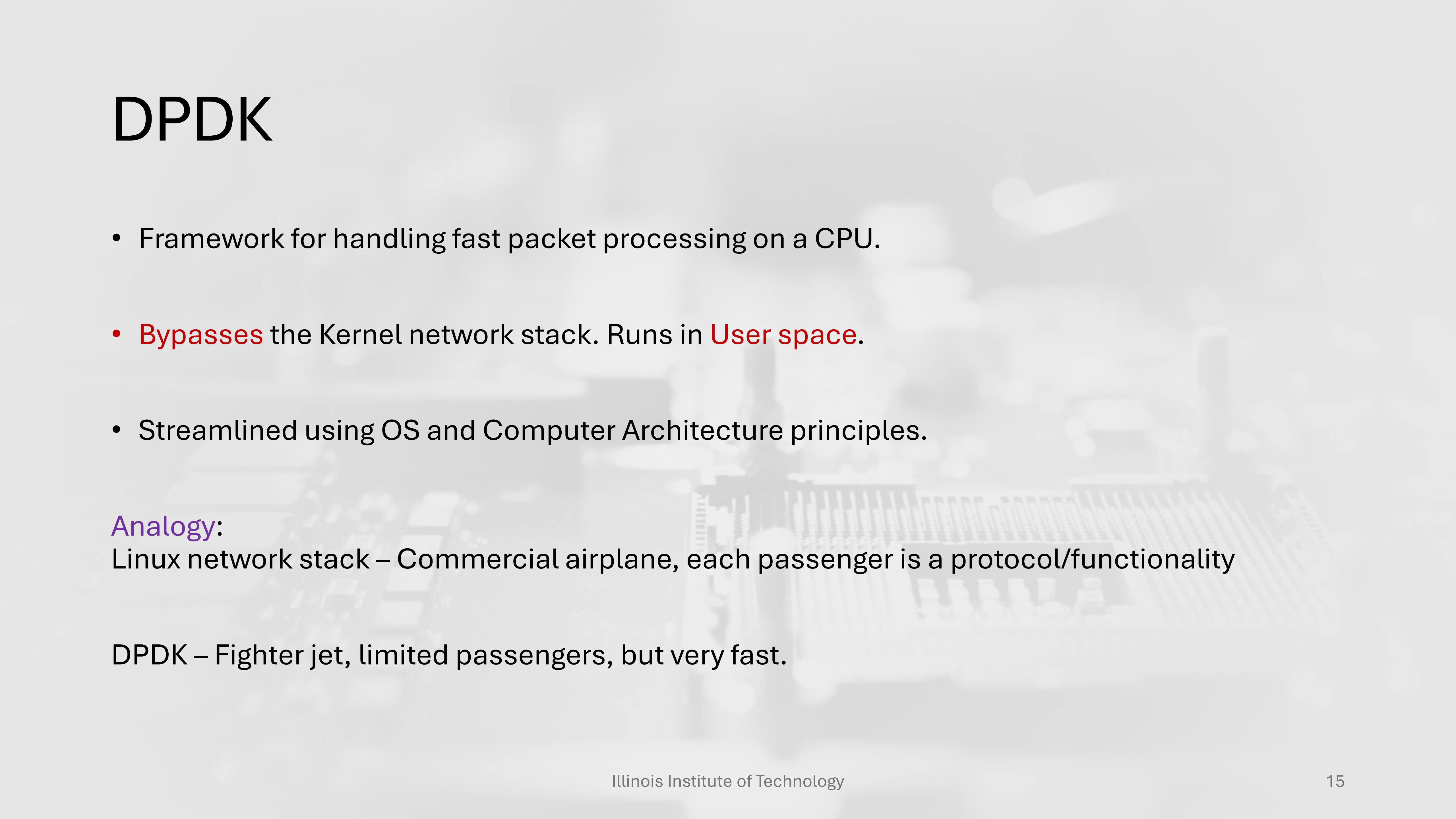
CPU overworked

- CPU needs to be super efficient to handle all the processing in 12-14 cycles

- We have to consider *real limitations* such as:

1. Memory access times (L1 cache – 4 cycles; L3 cache 40 cycles)
2. CPU scheduling
3. Page table walking
4. PCIe read/write latencies

We will address these in the coming slides, but it should be clear now that the CPU has limitations to how quickly it can process packets on a single core.



DPDK

- Framework for handling fast packet processing on a CPU.
- Bypasses the Kernel network stack. Runs in User space.
- Streamlined using OS and Computer Architecture principles.

Analogy:
Linux network stack – Commercial airplane, each passenger is a protocol/functionality

DPDK – Fighter jet, limited passengers, but very fast.

The architecture behind DPDK

The concepts we will cover here are:

- Memory Pools
- Hugepages
- Interrupts and polling
- Packet copying
- NUMA alignment
- PCIe latency

Memory Pools

- DPDK operation:

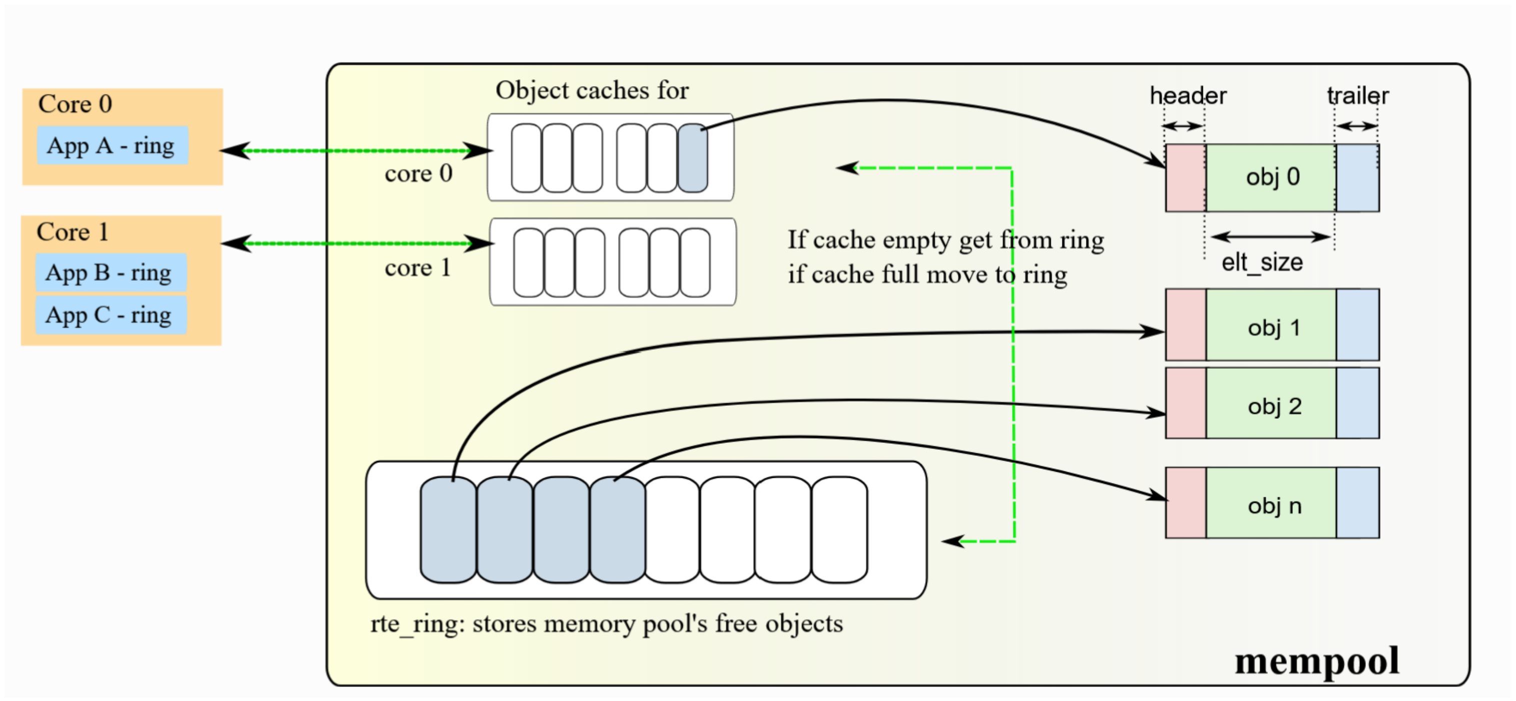
A pool of memory is allocated and assigned when the program starts up, but before any packet processing has begun.

When the program requests a buffer for example, it obtains the memory from the pre-allocated memory pool. Avoids runtime allocation.

When program has finished using the buffer, it isn’t freed, but instead recycled back into the pool, so that it can be used later.

This removes the memory allocation overhead.

Memory Pools



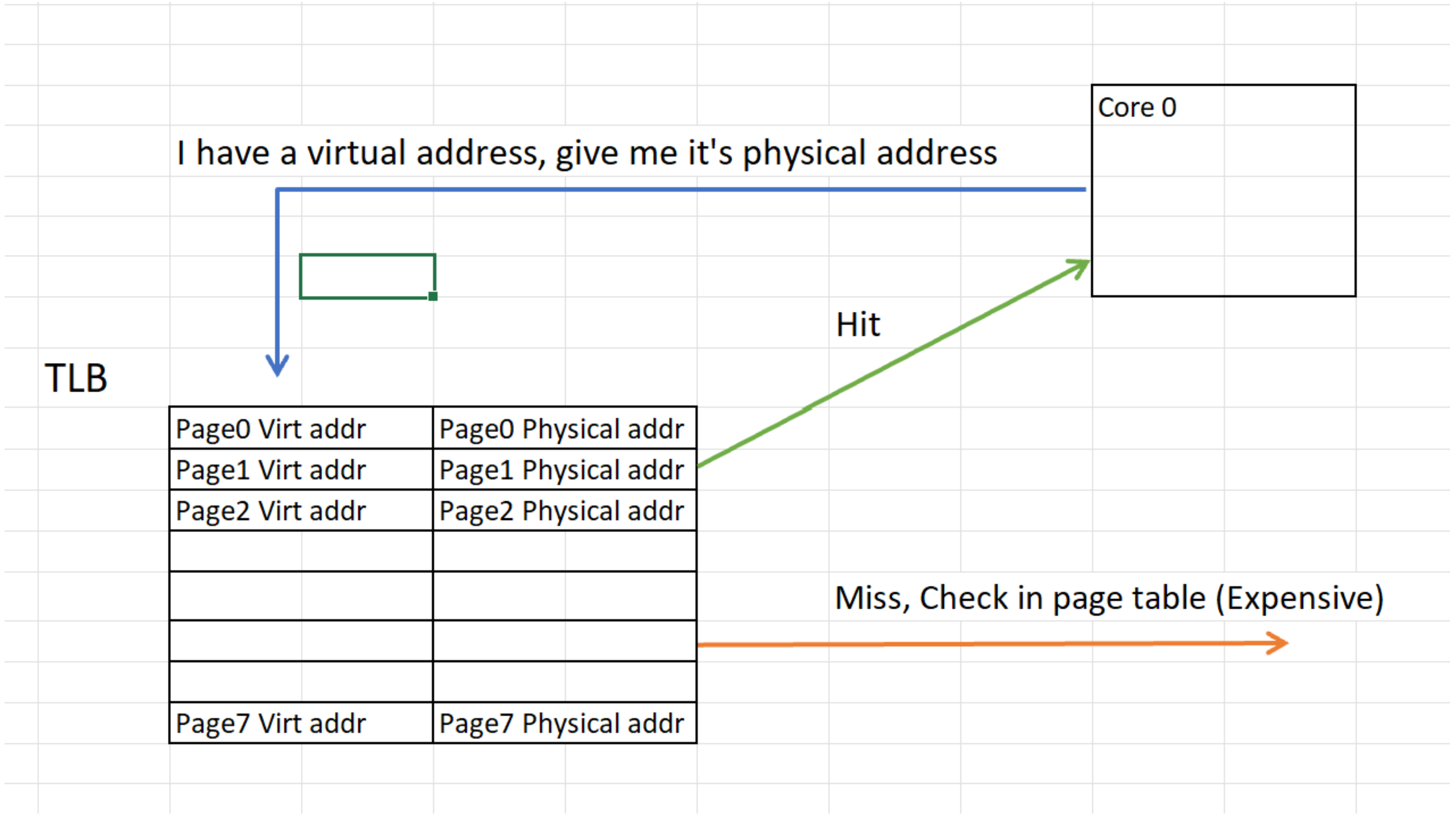
- White – Available objects
- Grey – In use objects

Image source: https://doc.dpdk.org/guides/prog_guide/mempool_lib.html

2. Hugepages

- The normal page size in Linux is generally 4KB.
- Pages are abstractions used by the OS to give the illusion that every process has access to more than available RAM.
- To maintain this illusion, a page table is required.
- To speed up page table lookup, a Translation Lookaside Buffer(TLB) is used.
- TLB has super fast lookup, but expensive memory and logic.

TLB


Core 0
I have a virtual address, give me it's physical address
Hit
TLB
Miss, Check in page table (Expensive)

Page0 Virt addr	Page0 Physical addr
Page1 Virt addr	Page1 Physical addr
Page2 Virt addr	Page2 Physical addr
Page7 Virt addr	Page7 Physical addr

TLB size:

2048 entries for Intel Xeon Gen 3

Hugepages

- Too much data accessed in a short time will fill up the TLB and cause page table walking on a MISS.
- Increases latency drastically, since page table resides in main memory.
- To avoid this scenario, DPDK always uses hugepages.

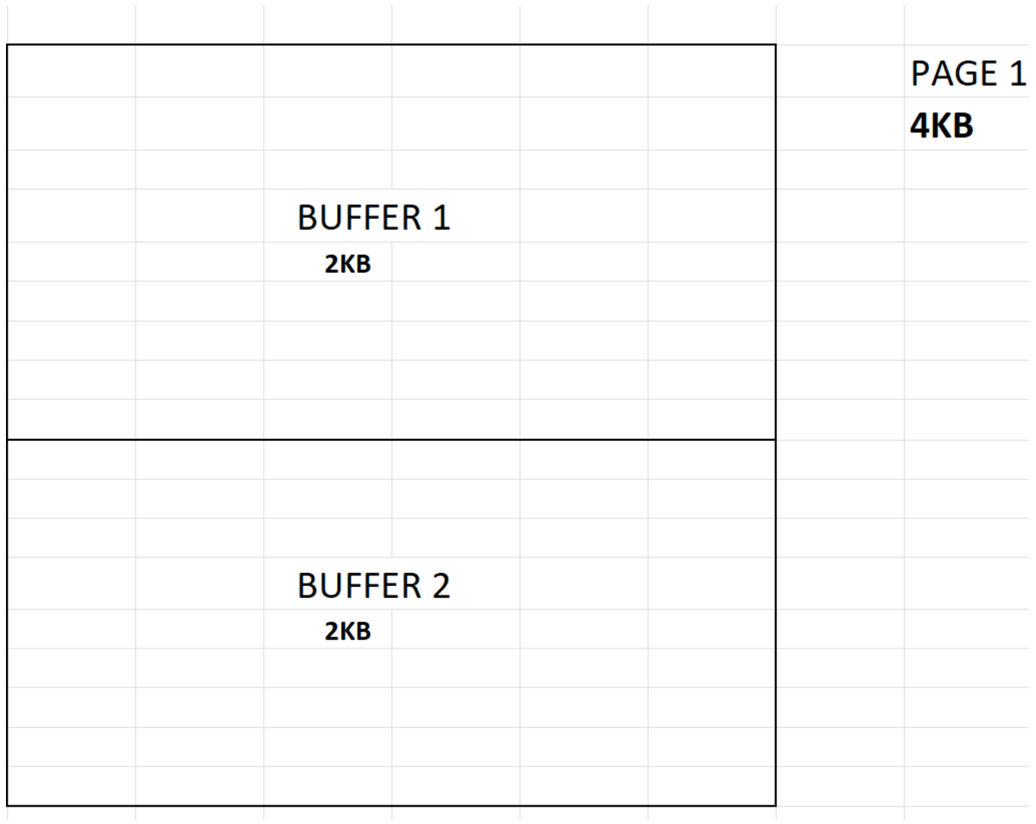
Hugepages

- Instead of the traditional 4KB, hugepages are 2MB or 1GB in size.
- More data in memory is covered by a single mapping.
- Mbufs(DPDK Packet buffer representation) are 2KB in size.

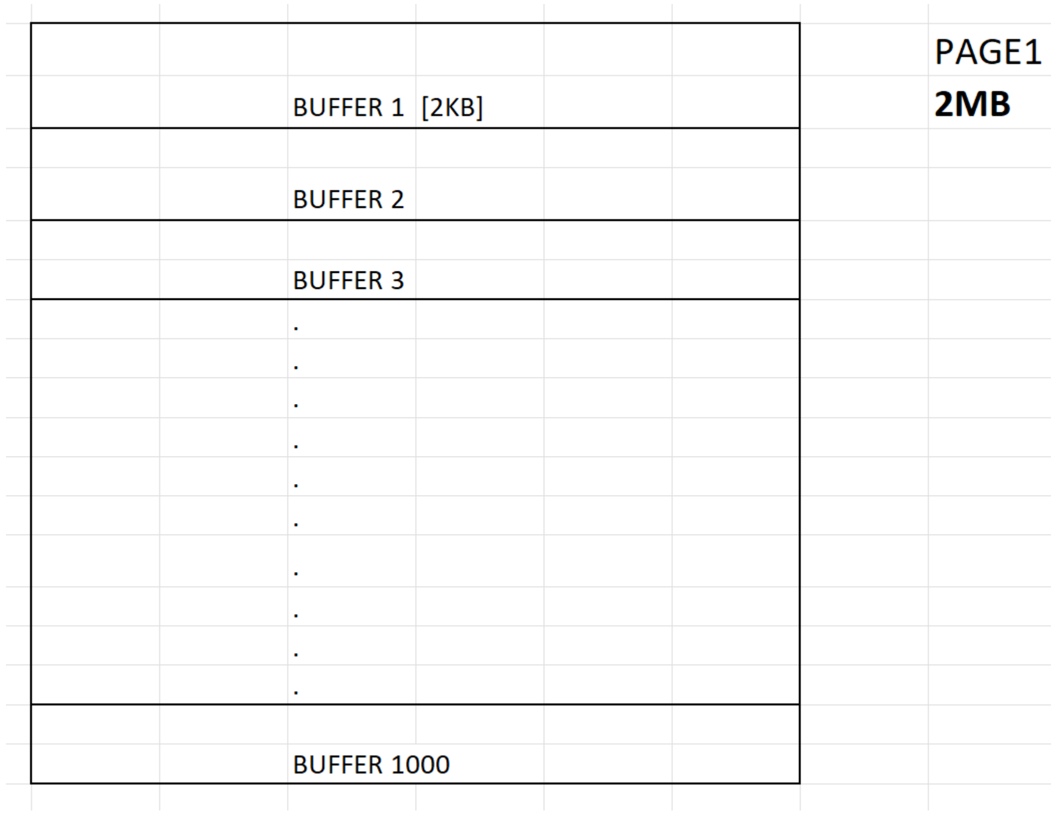
1000 Mbufs being accessed for a 4KB page will require 500 entries.

- 1000 Mbufs being accessed for a 2MB page will require only 1 entry.
- Reduces need to do page table walking.

Hugepages



2 entries per page

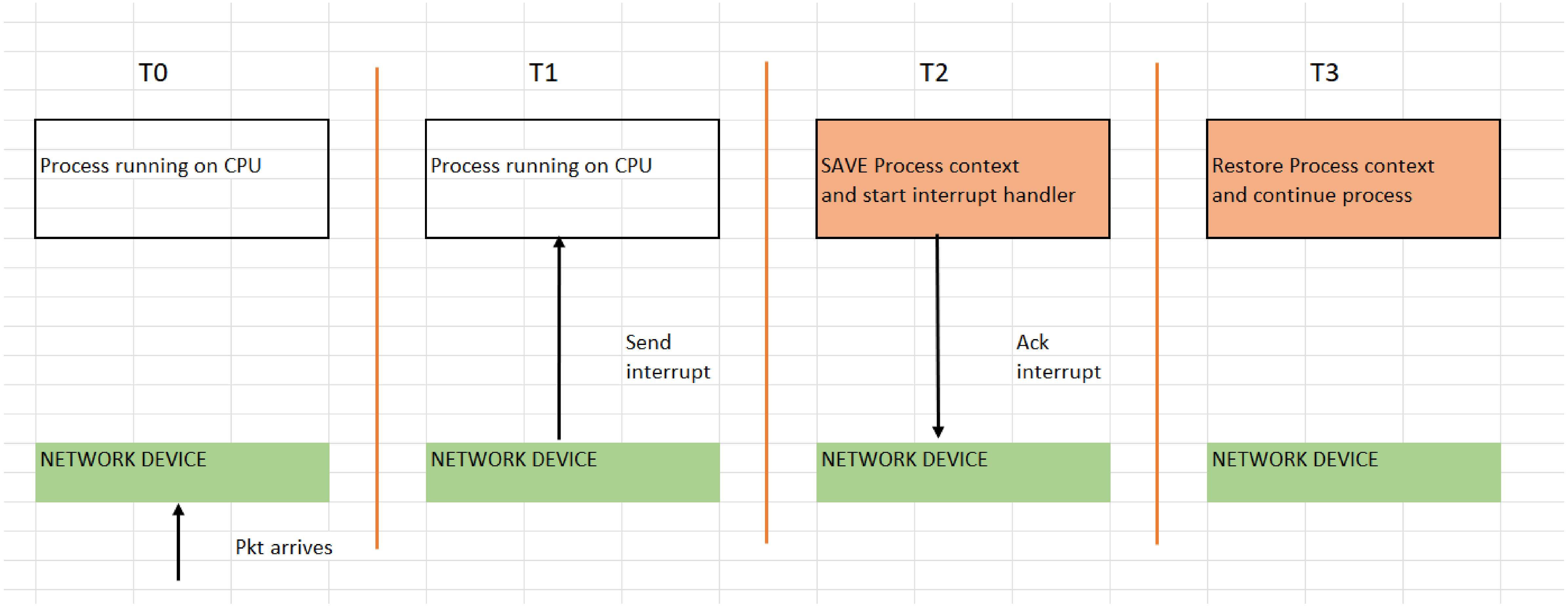


1000 entries per page

3. Polling

- When the network device receives a packet, it notifies the CPU with an interrupt.
- The CPU processes this interrupt and consumes the packet.
- Interrupts interrupt the running process on the CPU to handle the packet. This context switching is expensive in terms of instruction cache warmth.
- An interrupt for every packet for 148Mpps, can stunt the performance.

Interrupt driven

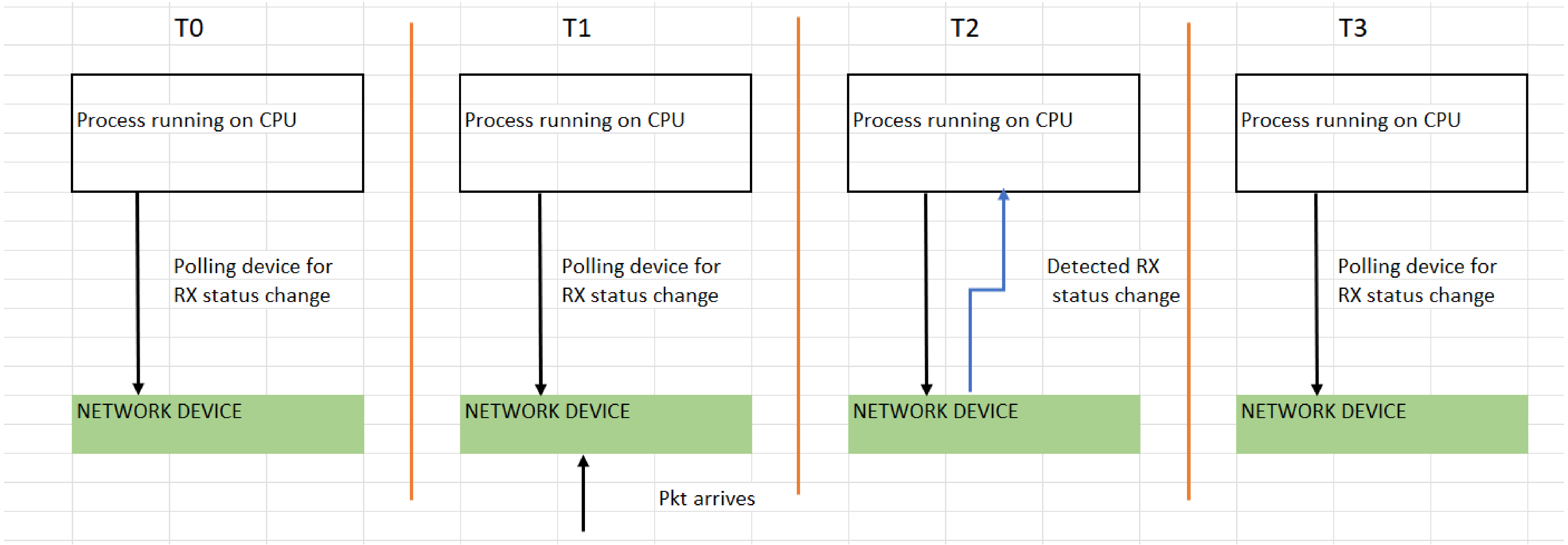


What does Saving and Restoring Process context mean ?

Polling

- Instead of the device notifying the CPU, the driver code in the CPU continuously polls the device registers for change of state.
- If register state change is observed, the packets are then read and processed.
- Avoids polluting instruction cache.
- Prevents context switching per packet.
- Polling code runs even when no packets arrive, making it waste CPU cycles. This is the reason why any DPDK process will show 100% CPU utilization

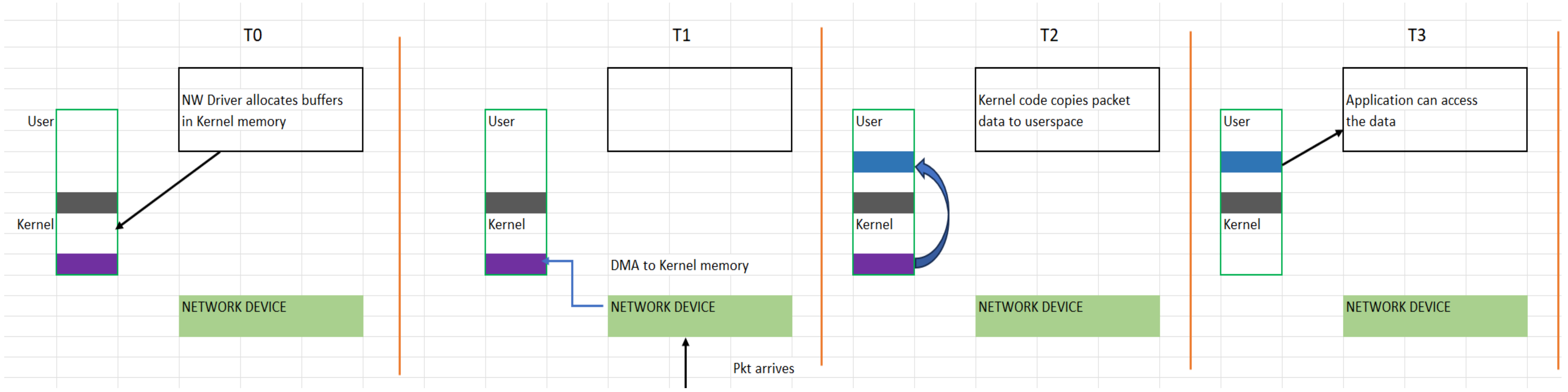
Poll mode


T0
Process running on CPU
Polling device for RX status change
NETWORK DEVICE
T1
Process running on CPU
Polling device for RX status change
NETWORK DEVICE
Pkt arrives
T2
Process running on CPU
Detected RX status change
NETWORK DEVICE
T3
Process running on CPU
Polling device for RX status change
NETWORK DEVICE

4. Zero copy

- Before a packet arrives, the network driver will instruct the device to transfer the packet to kernel memory.
- When packet arrives, the device DMAs the packet to this kernel memory
- Kernel memory is not visible to user space.
- So if user application needs to use the data, it should provide a user space buffer.
- The kernel code will then copy the packet from kernel memory to user provided buffer memory.
- This byte by byte copying of data uses many clock cycles, hurting performance

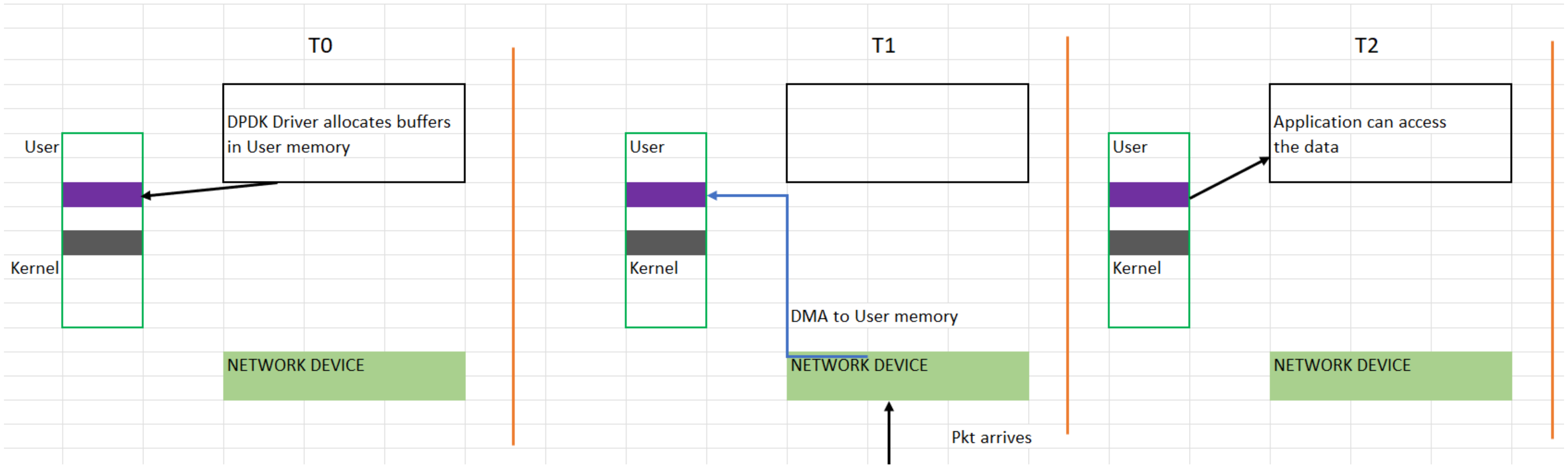
Copy operation



Zero copy

- DPDK avoids using any kernel memory.
- Instead uses user space memory and instructs the network device to directly send the packet over there.
- Avoids copying from kernel to user-space. Which is why it is called Zero copy.
- Disadvantage is the loss of security. There is no protection against a misbehaving DPDK driver.
- But great from a network performance perspective.

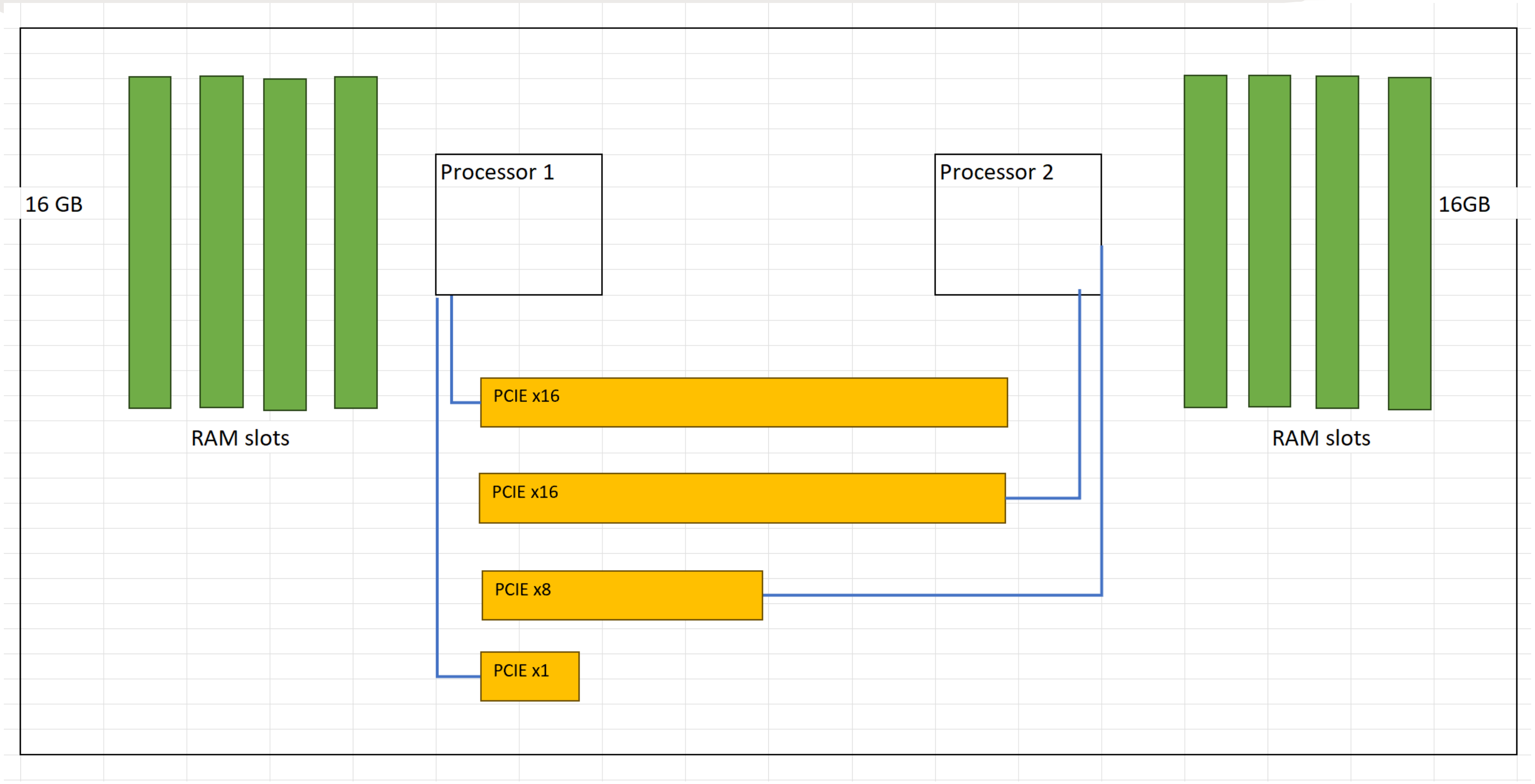
Zero copy


T0
DPDK Driver allocates buffers in User memory
User
Kernel
NETWORK DEVICE
T1
User
Kernel
DMA to User memory
NETWORK DEVICE
Pkt arrives
T2
Application can access the data
User
Kernel
NETWORK DEVICE

5. NUMA alignment

- In modern processors, there can be multiple CPU sockets, each having it’s own memory and IO controllers.
- Data in memory closer to the socket is accessed quickly improving performance.
- Data in memory in another socket takes an interconnect penalty, reducing performance.
- That is, the memory access is non uniform. Which is why it’s called Non Uniform Memory Access (NUMA).

NUMA sockets



NUMA alignment

- On DPDK application bootup, it ensures that data structures and packet buffers are assigned to the right CPU socket in order to improve performance.
- Users need to make sure that the CPU cores assigned to run the DPDK program are aligned with the PCIe device.

6. PCIe

- Common high speed communication bus used to transfer data between IO device and main memory/CPU.
- Has its’ own communication protocol(TLP) with its’ own overheads.
- DPDK drivers take this into account, and try to minimize the overheads by batching PCIe reads and writes.
- Tries to avoid the communication bus becoming the bottleneck.
- User has to ensure PCIe bandwidth is sufficient.

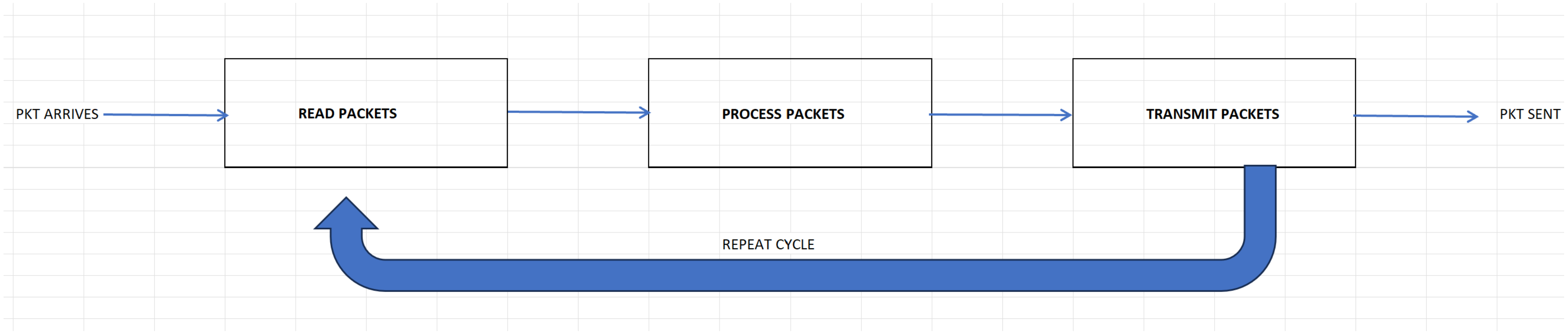
DPDK modes of operation (RTC)

Run to Completion (RTC)

A single RTC cycle comprises of Read packet → Process packet → Send packet

Very efficient for keeping the L1 Data cache warm

RTC


PKT ARRIVES
READ PACKETS
PROCESS PACKETS
TRANSMIT PACKETS
PKT SENT
REPEAT CYCLE

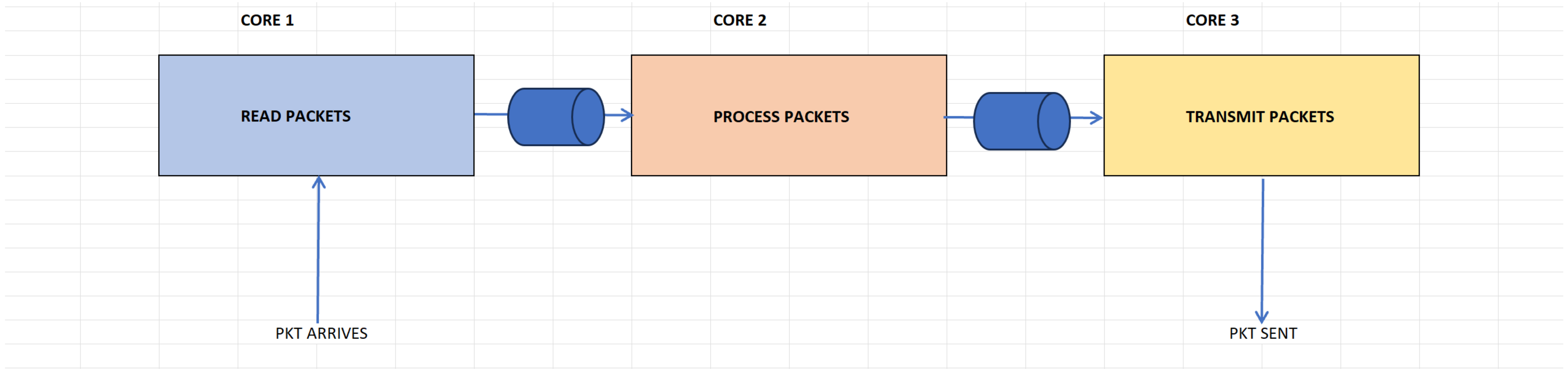
DPDK modes of operation (Pipeline)

Multiple stages where each stage runs on a specific core.

One stage for reading packets, a 2nd stage for processing packets and a 3rd stage for transmitting packets.

DPDK Rings used to synchronize data between the cores

Pipelining


CORE 1
CORE 2
CORE 3
READ PACKETS
PROCESS PACKETS
TRANSMIT PACKETS
PKT ARRIVES
PKT SENT

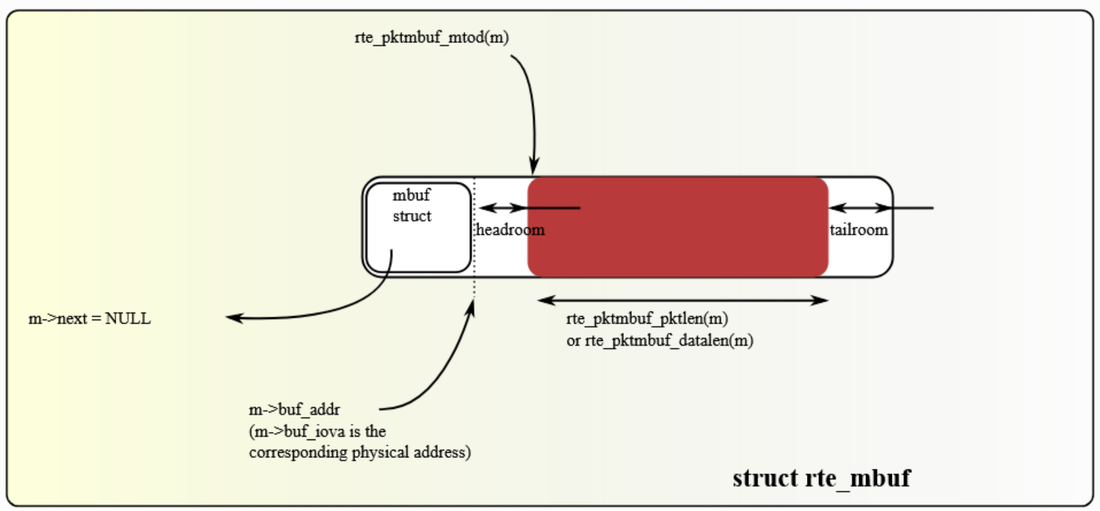
DPDK modes of operation (Pipeline)

- More stages can be added or removed as needed.
- Great for L1 Instruction cache warmth. [Why ?]
- Stages are connected using DPDK rings

DPDK mbuf

- DPDK packet buffer used to store packet data and packet metadata.
- Packet metadata is stored in the mbuf header.
- Each mbuf is 2KB in size (including mbuf header).
- Can be chained to hold jumbo packets.

DPDK mbuf


rte_pktmbuf_mtod(m)
mbuf struct
headroom
tailroom
m->next = NULL
rte_pktmbuf_pktlen(m) or rte_pktmbuf_datalen(m)
m->buf_addr (m->buf_iova is the corresponding physical address)
struct rte_mbuf

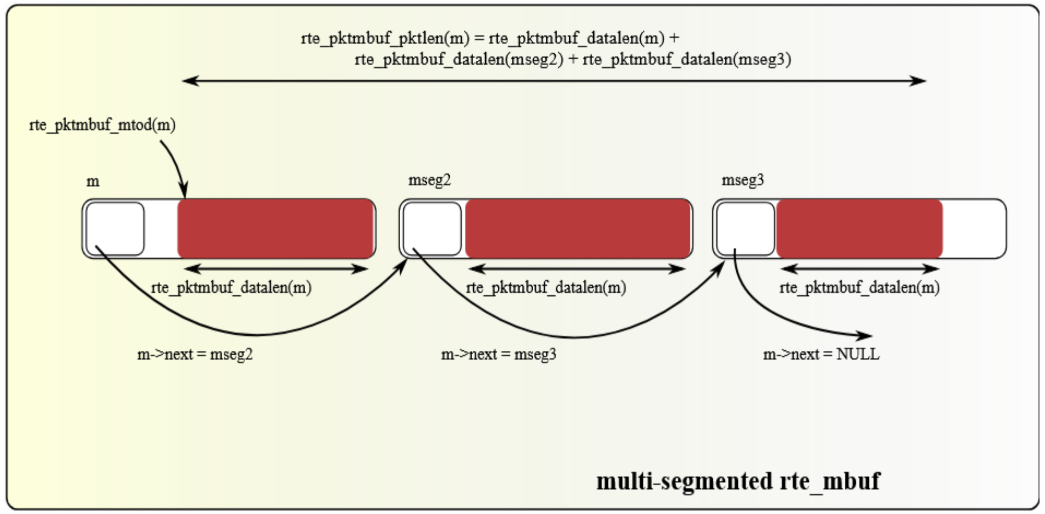


Image source:
https://doc.dpdk.org/guides/prog_guide/mbuf_lib.html

Horizontal scaling

- As mentioned earlier , 12-14 cycles is too small a time to process a packet on the CPU.

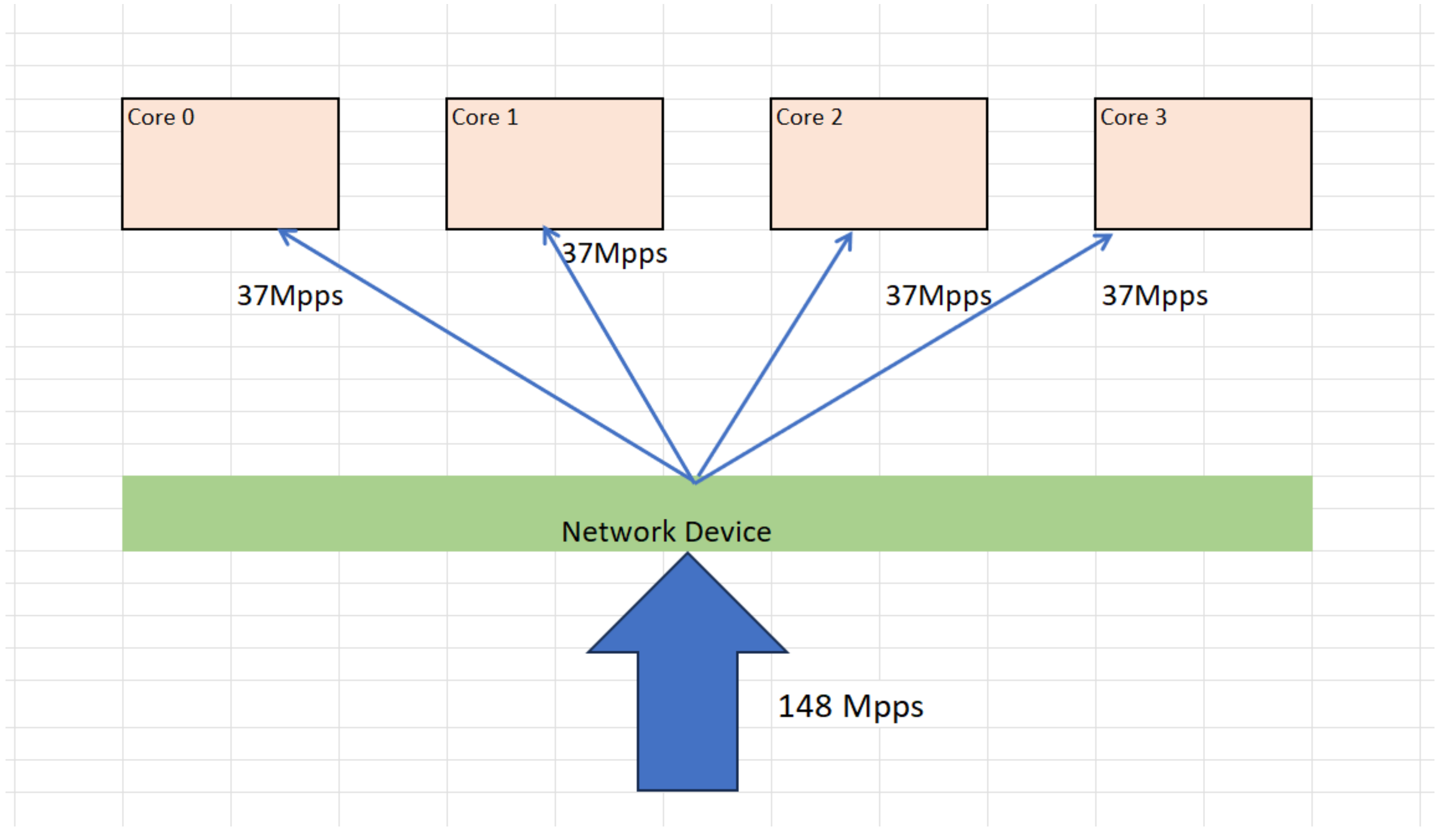
  What can be done to mitigate this?

- Use multiple CPU cores and share the workload aka horizontal scaling.

- For example, 4 cores can share the number of arriving packets equally, such that each core is processing 148/4 = 37Mpps.

- This means each core now can process a packet in 12-14 * 4 = 48 – 56 CPU cycles. A lot more breathing room.

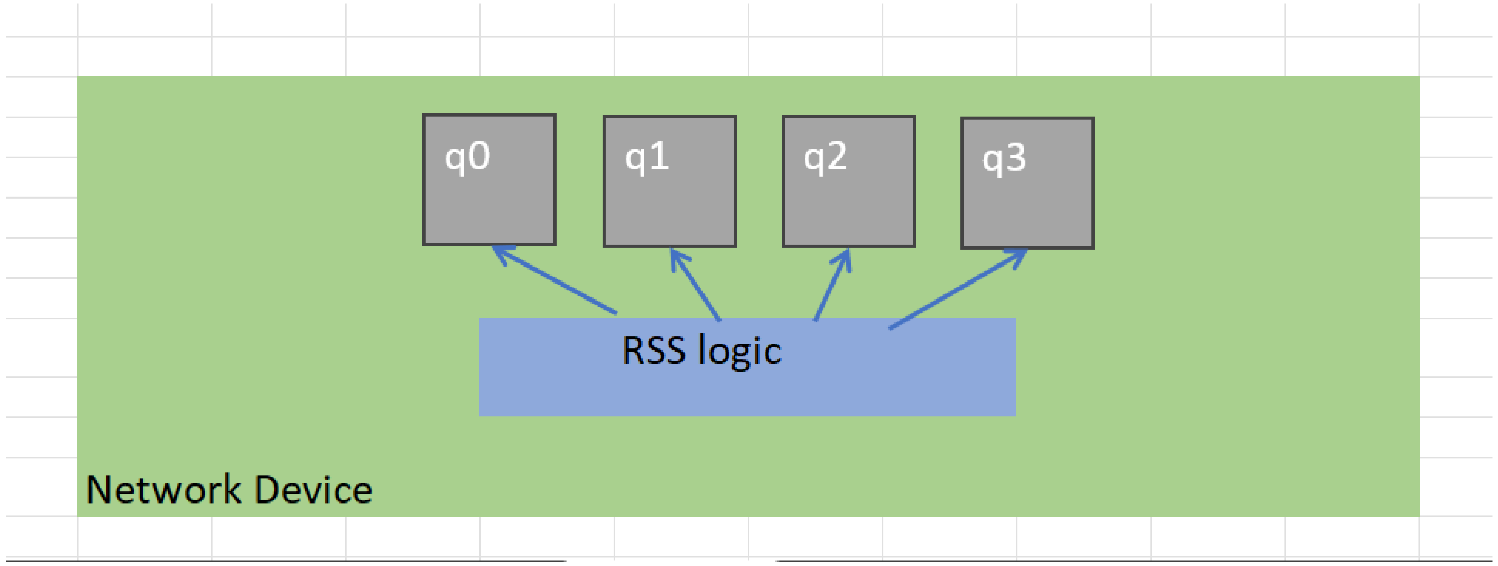
Horizontal scaling



HW queues and RSS

- The question now is ‘How do we split the packets’ ?
- Modern Network Interface Cards (NICs) have multiple HW queues.
- The arriving packets can be split by the NIC to each of it’s HW queues.
- A generally supported algorithm to do the splitting on NICs, is Receive Side Scaling (RSS).

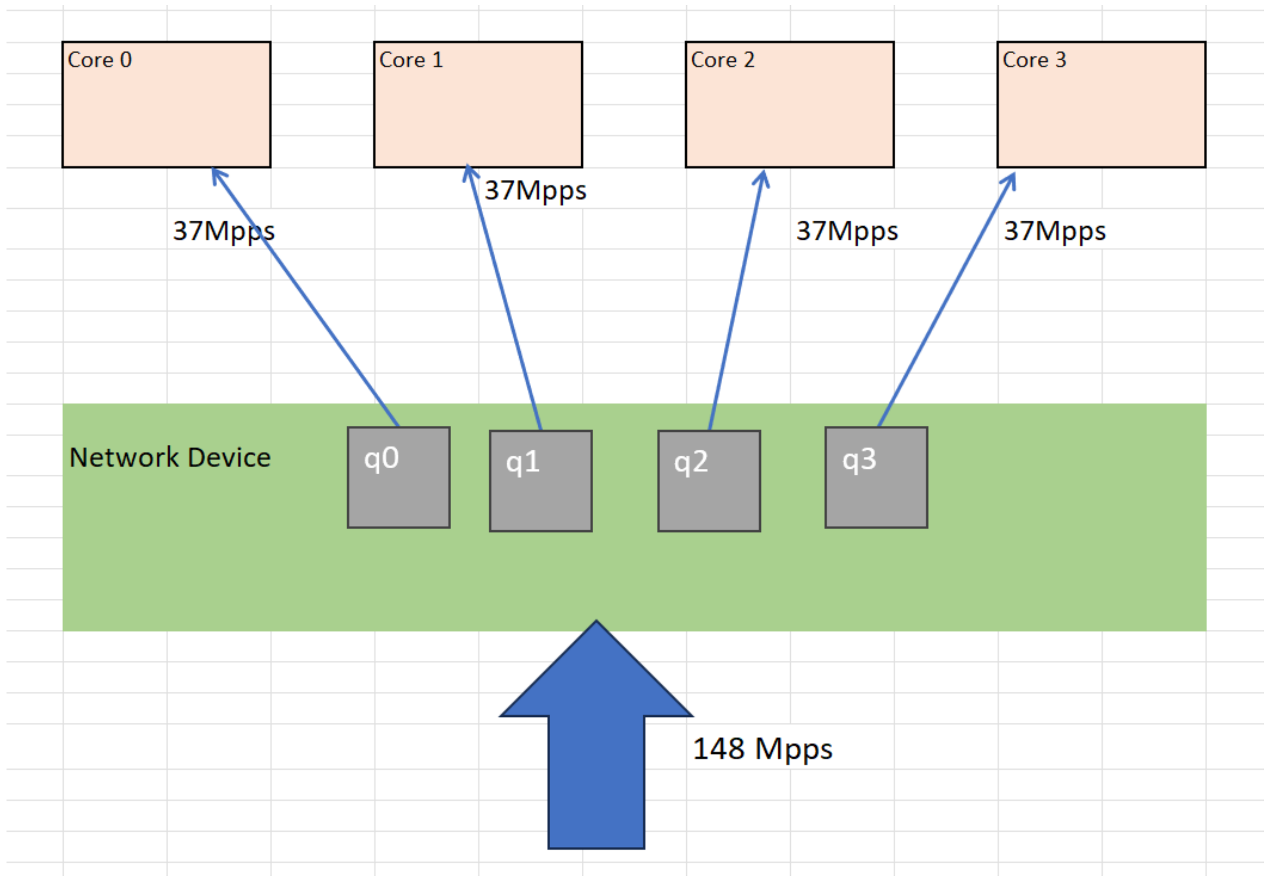
RSS



HW queues and RSS

- Each processing core in DPDK is assigned to a specific HW queue.
- It reads packets from that queue, and processes it.
- If no packets arrive on that queue, it will not process any packets.
- Enabling RSS makes efficient use of cores.

Core-queue mapping



CPU:Queues Can be 1:1 or 1:n, but not n:1

DPDK Demo on FABRIC

References

- A look at Intel’s Dataplane Development Kit – Dominik Scholz

- Data Plane Development Kit (DPDK) – *A Software Optimization guide to the user space based network applications* – Heqing Zhu

- DPDK Programmer’s Guide
  https://doc.dpdk.org/guides/prog_guide/index.html

- . Skeleton code documentation
  https://doc.dpdk.org/guides/sample_app_ug/skeleton.html

- DPDK API documentation
  https://doc.dpdk.org/api/index.html

THANK YOU