DPDK Intro

Not only “Zero-Copy”

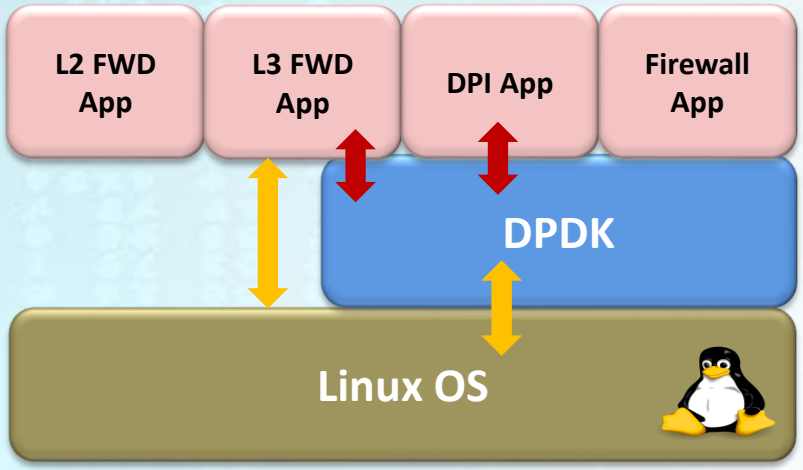
Lior Betzalel
liorbetz@gmail.com

Agenda

- What is DPDK?
- Bypassing the kernel
- Zero-Copy. What else?
- Packet lifecycle in DPDK based Application
- Router application with DPDK and Linux kernel

Data Plan Development Kit

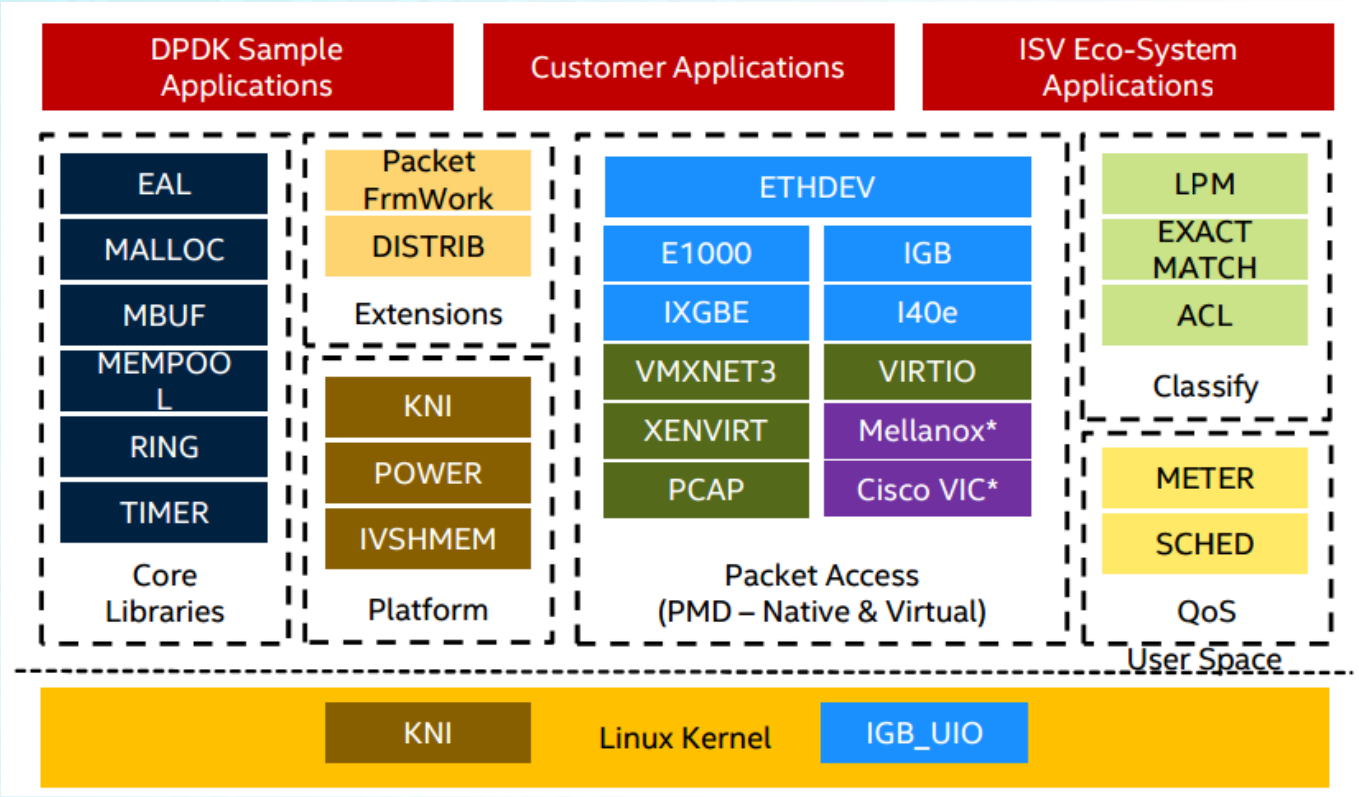
- DPDK is a set of libraries
  - Not only drivers (PMD)
  - Implemented with high attention to performance optimizations
  - Build your own packet processing application



DPDK
DATA PLANE DEVELOPMENT KIT

DPDK libraries

- DPDK is a set of libraries and optimized NIC drivers (PMD) for fast packet processing in **user space**.
- DPDK provides a framework and common API for high speed networking applications.
- Optimized buffer management (huge pages, pre-fetching, spreading over memory channels,…)
- Traffic management: metering, policing, shaping,…
- Classification: hashing, LPM,…



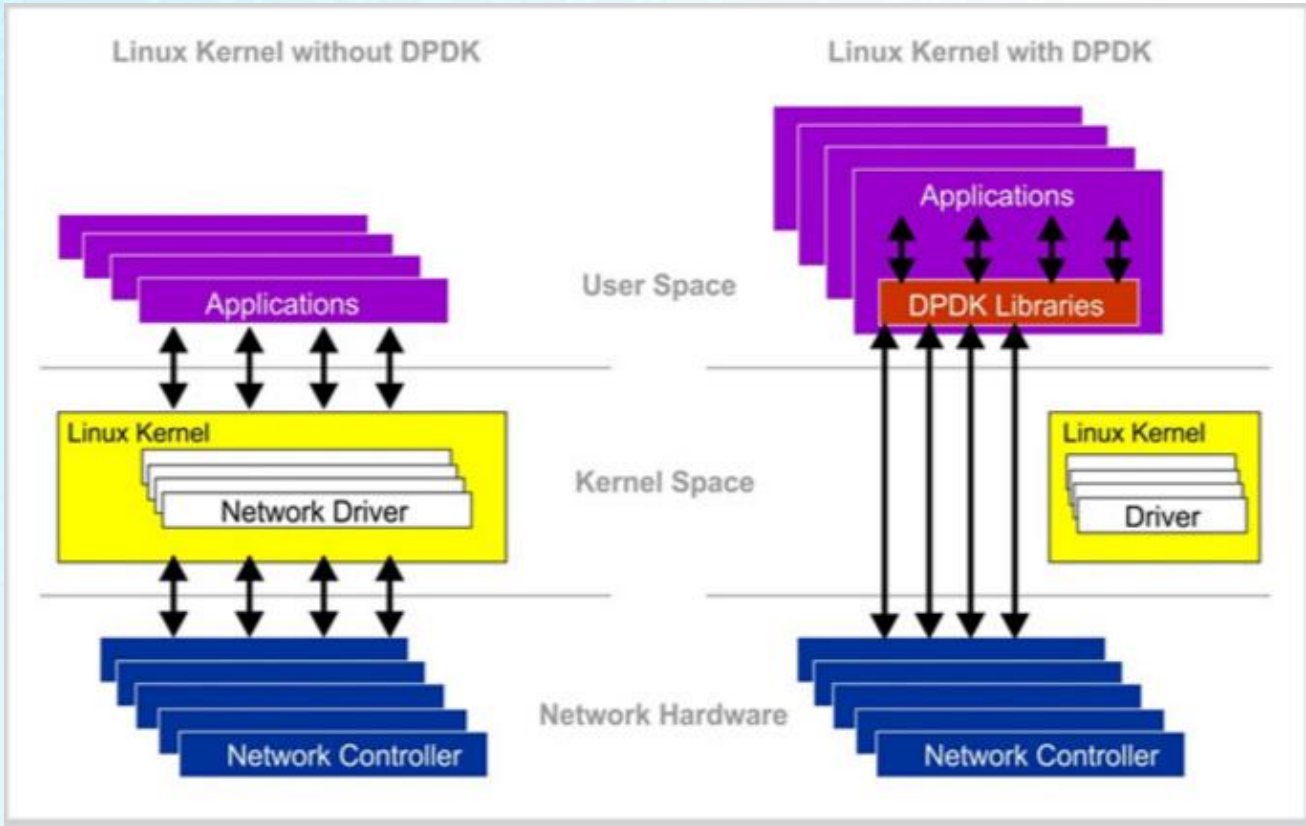
DPDK introduction (Cont.)

“Zero-copy” mode from network to CPU:

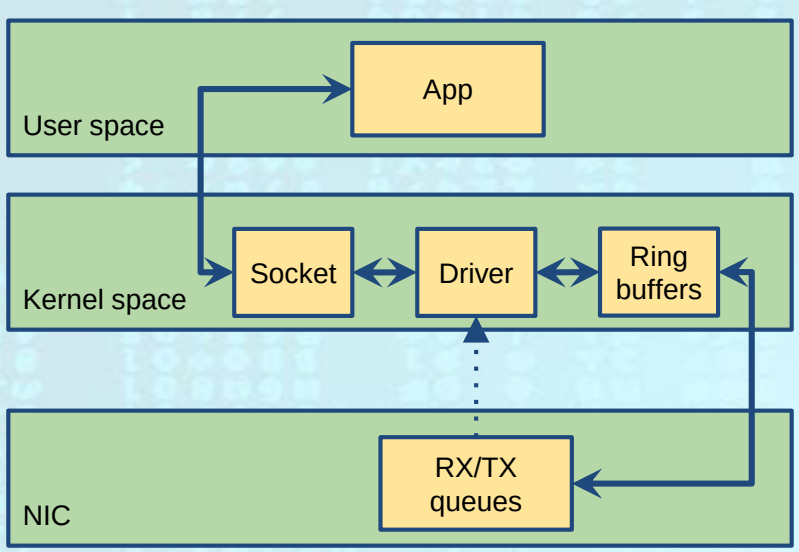
- ✓ Low latency
- ✓ Kernel bypass for the data-path

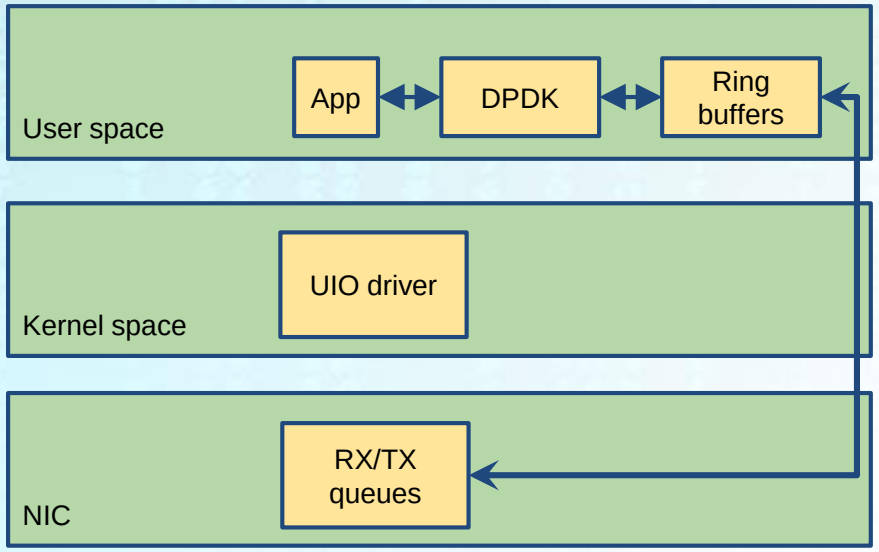
Packets are processed directly in the user space:

- ✓ No interrupts involving RX/TX traffic
- ✓ No scheduler - all devices accessed by polling
- ✓ No IP/TCP stack overhead



Kernel vs User-space Programming





Intel DPDK (Cont.)

- As a result packets arrived directly to user-land without passing the Linux Kernel network stack
  - No Networking stack (protocols) capabilities:
    - No TCP capabilities
    - No ARP capabilities
    - No ICMP capabilities (such as PING)
    - No IPSec capabilities
    - No L3 routing capabilities
    - Etc.
  - No Linux utilities:
    - No TCP-dump capabilities
    - No Netperf
    - No Ethtool
    - Etc.
  - No interface management
    - Ifconfig
    - Etc.

Not only “Zero-Copy”

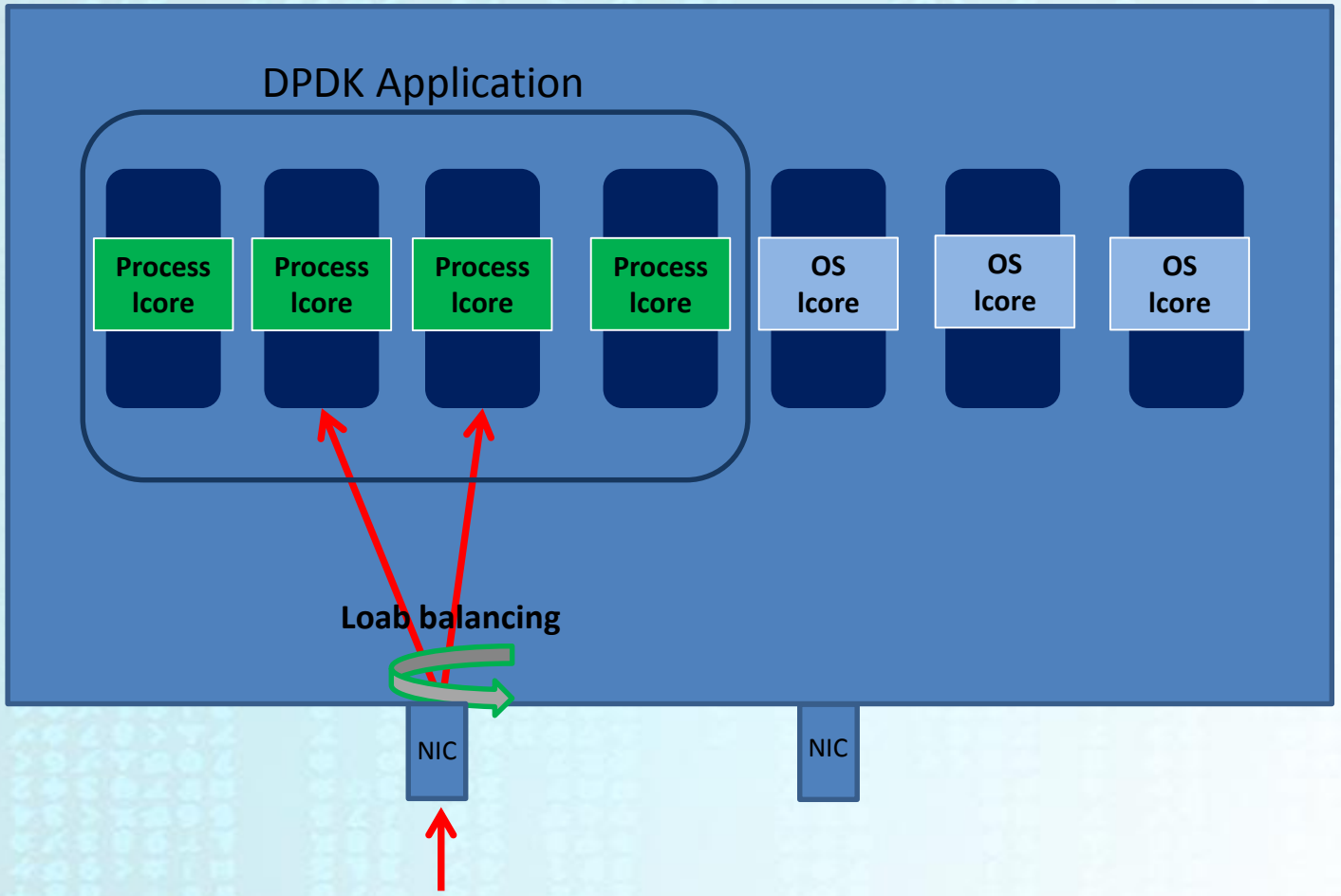
Not only “Zero-Copy” – DPDK uses more technics for accelerating packet processing:

- ✓ DMA directly to user-space memory
- ✓ Polling instead of handling interrupts (with bulk/burst mode)
- ✓ Hugepages to decrease the size of TLB that results in a much faster virtual to physical page conversion
- ✓ Thread affinity to bind threads to a specific core to improve cache utilization

Not only “Zero-Copy” – Cont.

- ✓ Cache prefetch:
  - ✓ Packet descriptors
  - ✓ Lookup data (Prefetching of Keys in Cache)
- ✓ Lockless Inter-core communication
- ✓ NUMA awareness to avoid expensive data transfers between sockets
- ✓ SSE instruction set to copy big amount of data effectively

Multi Core DPDK Application



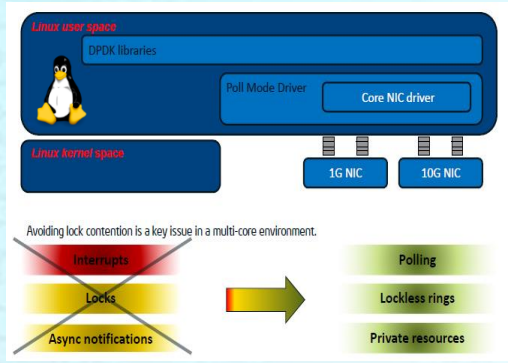
PMD – the gate to DPDK user space application

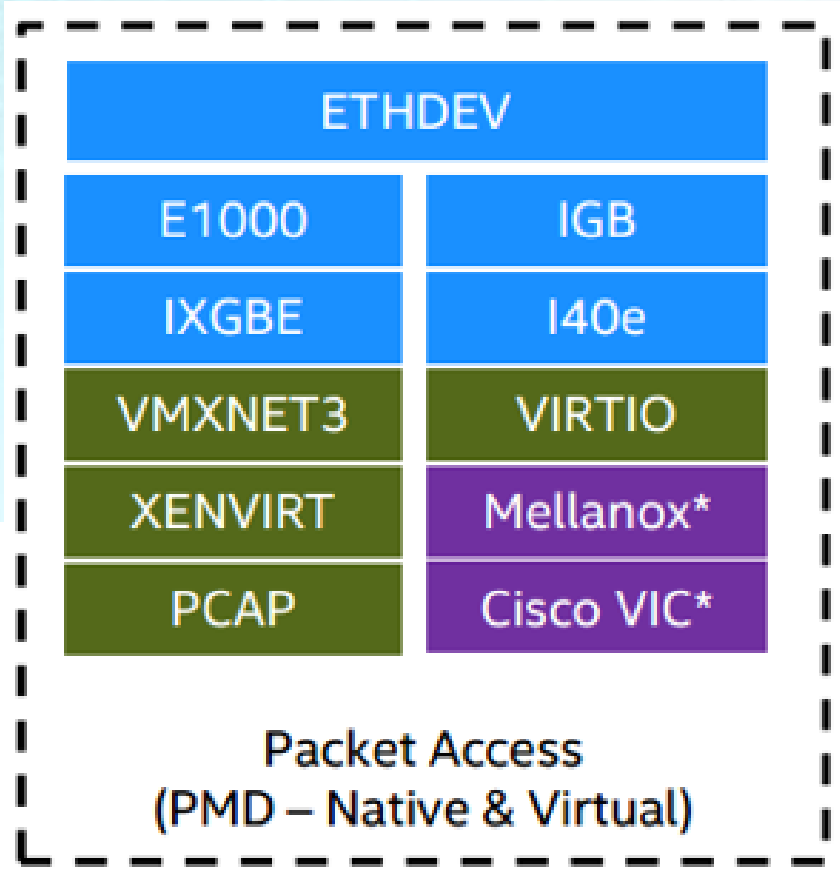
The networking drivers can be classified in two categories:

- Physical for real devices
- Virtual for emulated devices

NIC accelerations:

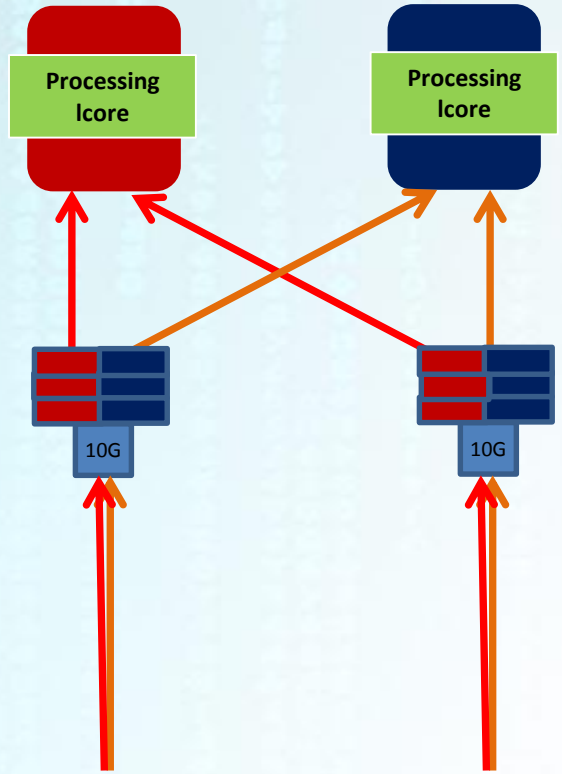
- Specific HW offloading
- Support RX/TX bulk and burst
  - Throughput Vs Latency





From NIC to lcore

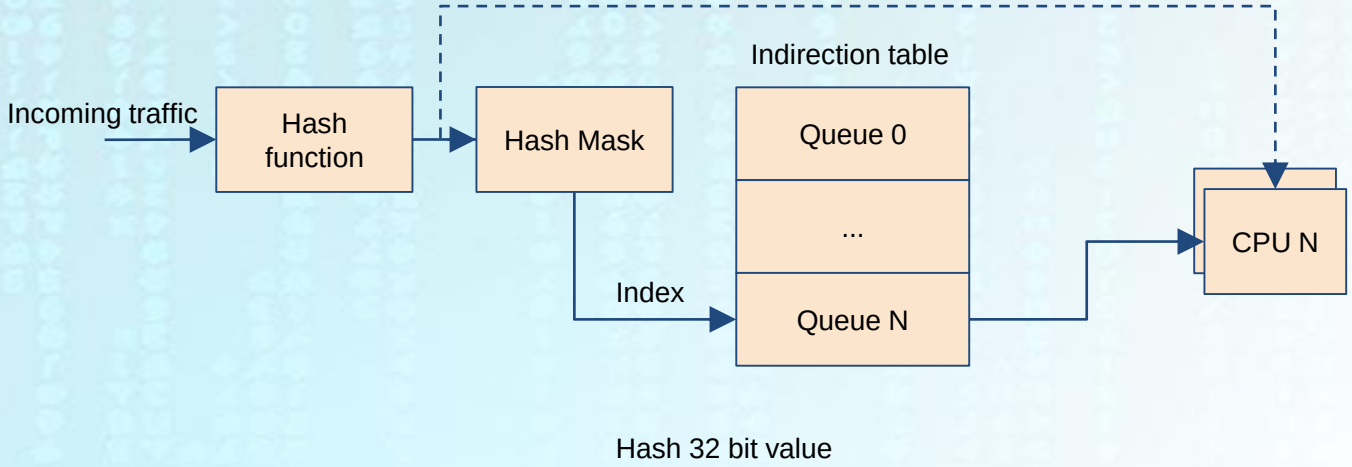
- DMA directly to user-space memory
- Lcore polls the NIC queues
  - Avoid locks - RX queue per NIC per polling LCORE
  - Traffic distributed from NIC to Queue to lcore
  - RSS



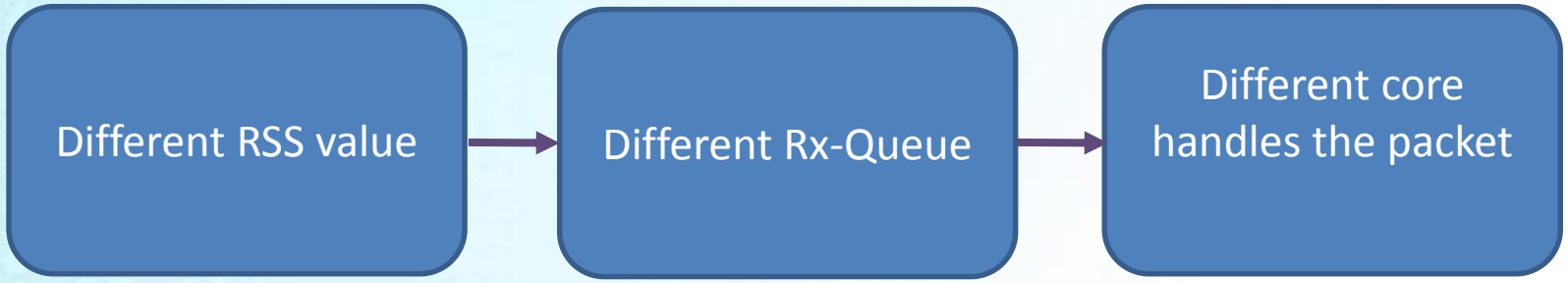
From NIC to lcore - Cont.

Uses for multi-core scaling:

- Depends HW capabilities
- Symmetric RSS key
  - Hashing bi-directional flows

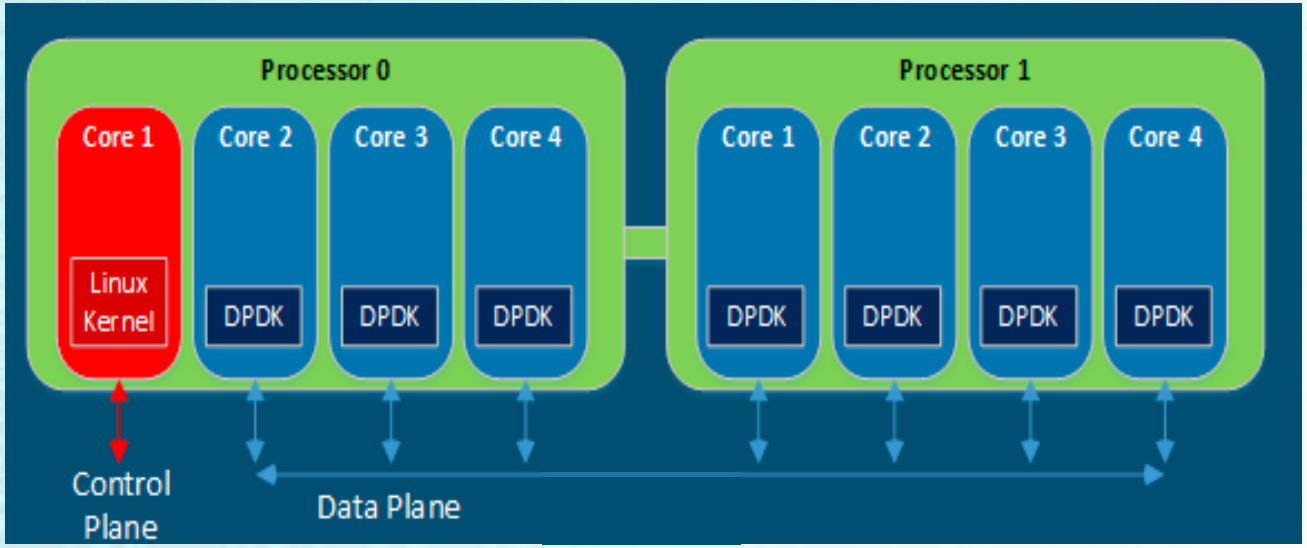


Packet (mbuf) metadata from the HW contains RSS value



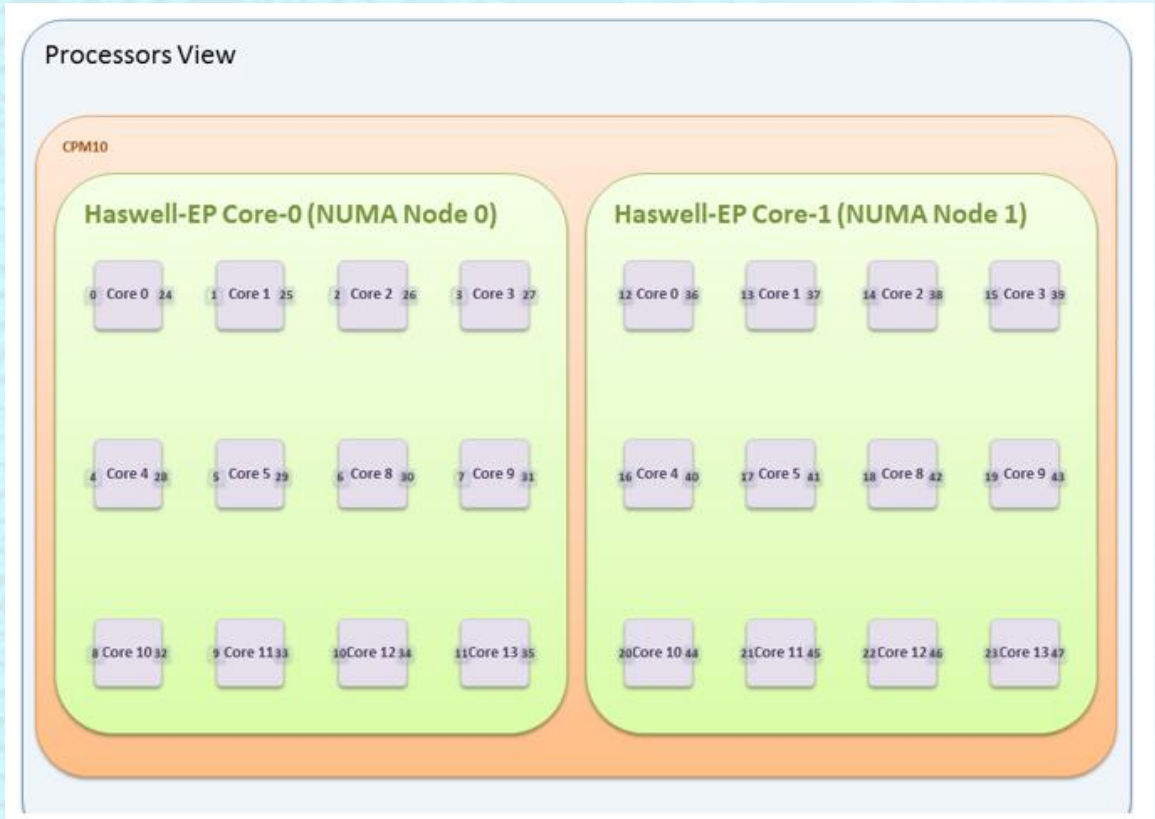
Inside the core - This core is mine!

- Linux **core-affinity**
  - set one processing thread per core
- Linux **core-isolation**
  - Get the OS specific cores while the other cores are used for the data plan
  - No Linux scheduling on isolated lcores



NUMA awareness

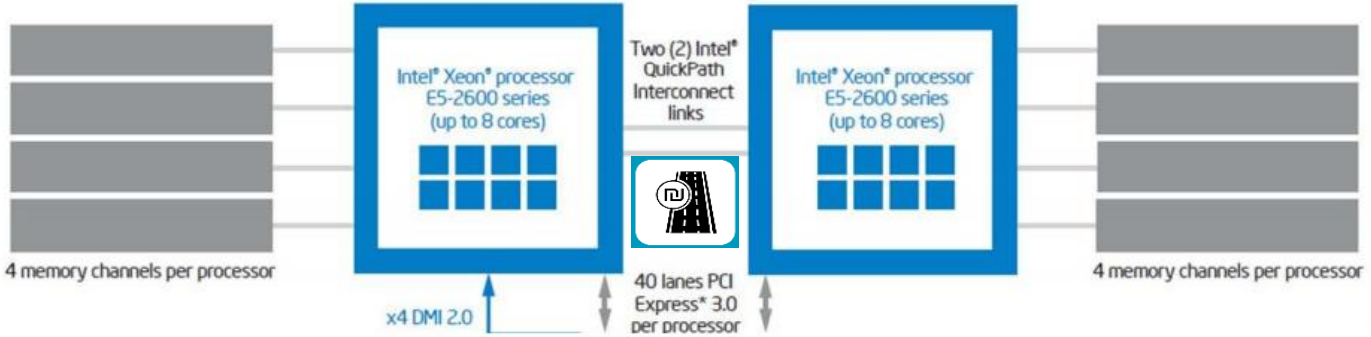
- NUMA node per NIC
  - Need to take into account which core connected to each socket
  - Which NIC connected to each socket? (HW design)
- Number of memory channels
  - Spread load equally over all channels



NUMA awareness – Cont.

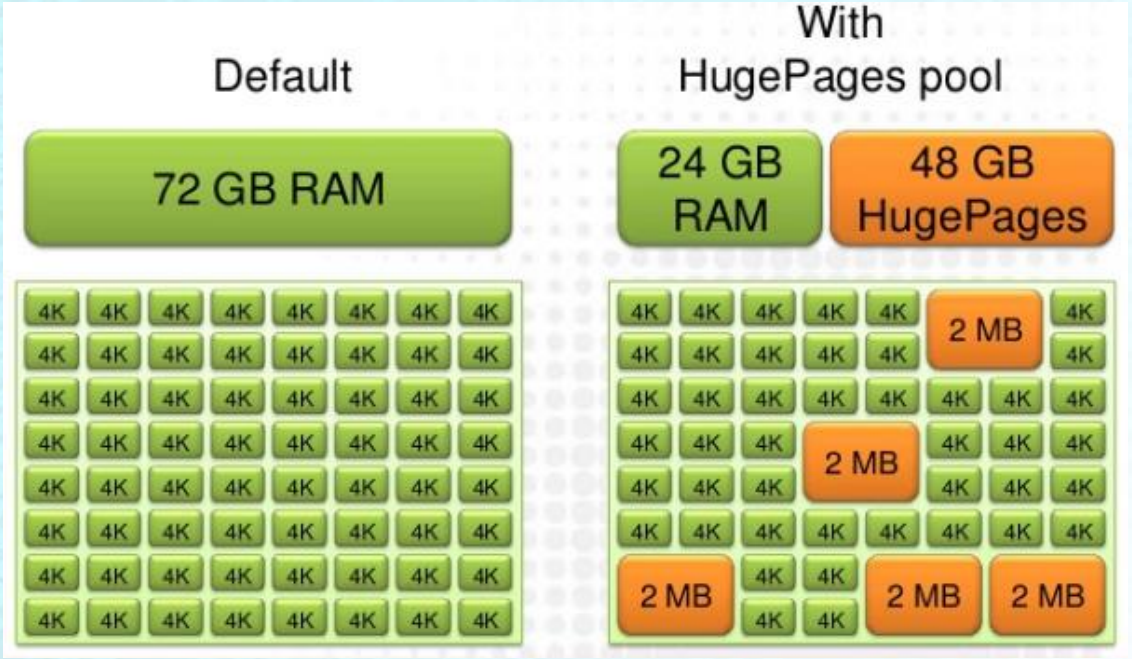
Design Tips:

- Need to considered core layout for each platform when choosing the coremask to use in each case
- Keep host process hugepage memory and QEMU process on the same NUMA node
- Create memory pools & rings on specific NUMA node



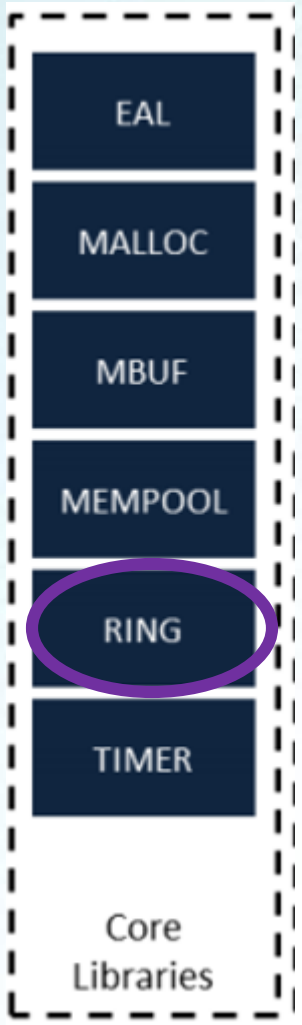
Huge pages

- The page size is increased to 2MB-**1G**
  - Reducing the total number of pages to be managed by the kernel
  - Less TLB misses



Communication Between lcores

- Ring core component:
  - Communications between applications/lcores in the DPDK (as pipeline)
  - Used by the memory pool allocator
  - Provides a lockless ring implementation
  - Supports bulk and burst access
    - Only one costly atomic operation
    - Performance is greatly improved when using bulk access operations



Core components – Ring lib

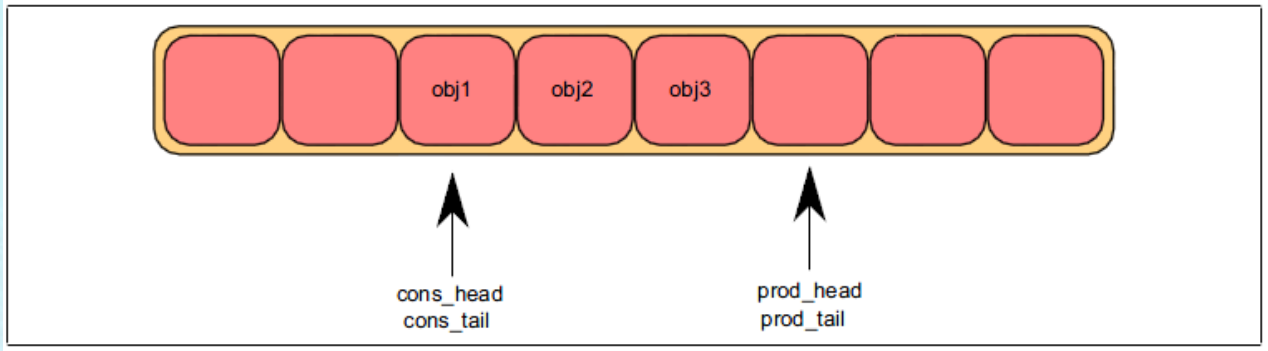
Properties:

- FIFO
- Max size is fixed
- Lockless (uses a "compare and set" instruction to move cons/prod index atomically)
- Multi/Single consumer/producer
- Bulk Enqueue / Dequeue
- Supports watermarking / threshold

Disadvantages

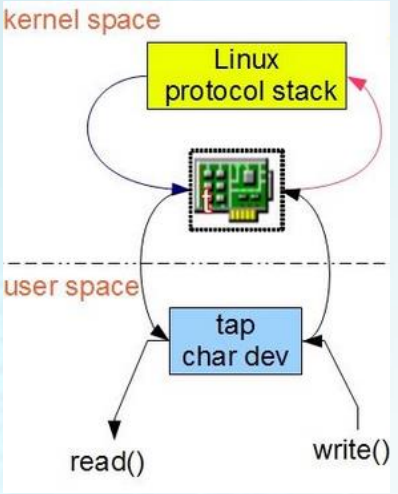
- Fixed
- Memory costs

Tip:
- Single Vs Multi

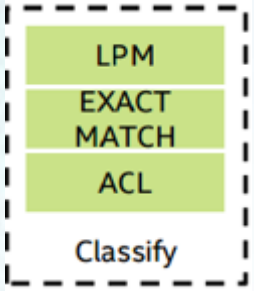


Processing lcore & kernel communication

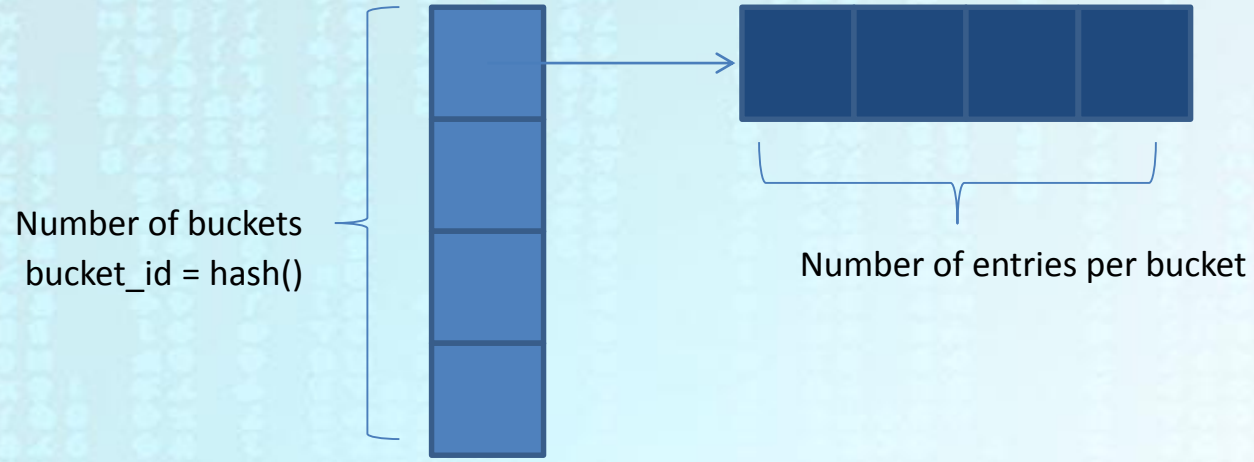
- Tap/TUN devices
- KNI based DPDK driver (Kernel NIC Interface)
- Netlink
  - Netlink socket used for communication between the kernel and user-space processes
    - Can receive kernel events to user land such as:
      - Interface up/down events
      - MTU change on interfaces
      - Routing updates
      - Etc…



Flow classification Core libraries



- LPM Library (Longest Prefix Match)
- Hash:

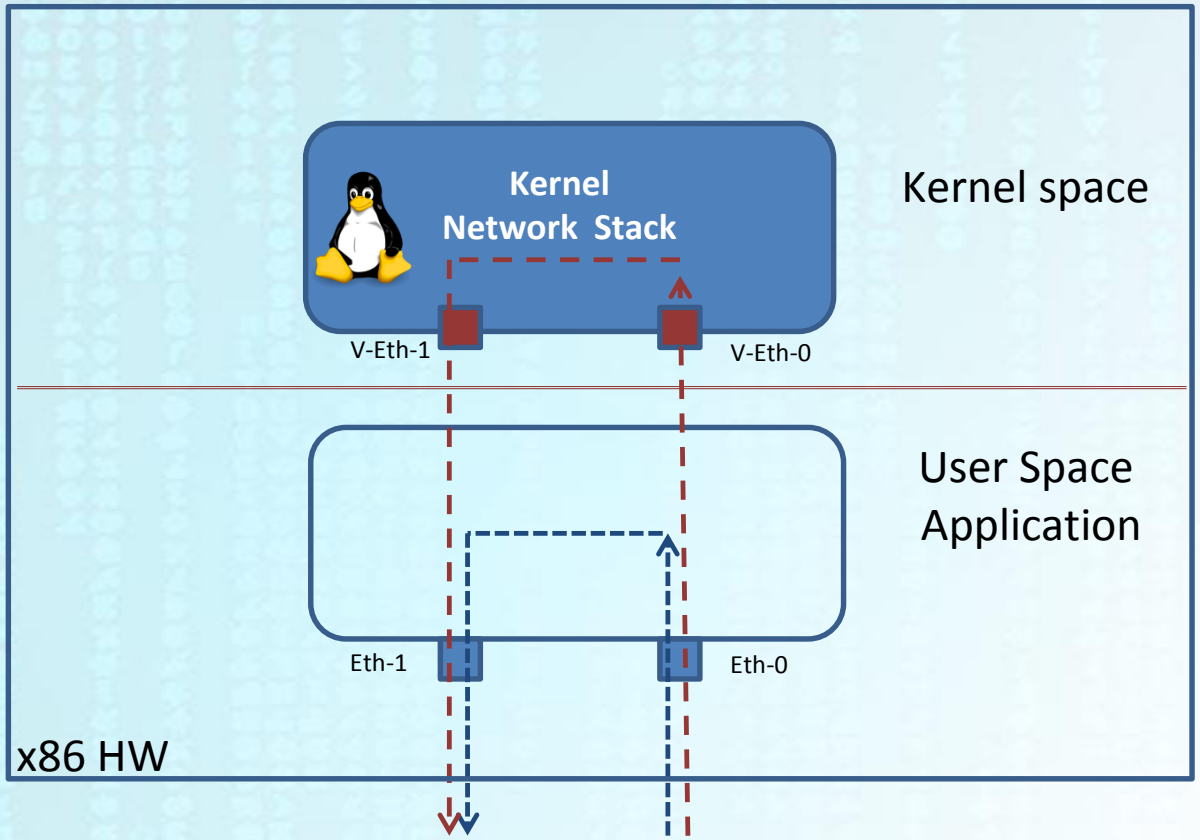


- From DPDK Ver 2.2 – Cuckoo hash:
  - Optimized Flow Table Design
  - Allows for more flows to be inserted in the flow table
  - Using Intel AVX instructions for scaling lookup performance

Application usage example:

Routing Application:

- Hash for flow classification
- KNI/TAP for kernel IP stack



Thank You

Lior Betzalel
liorbetz@gmail.com