DPDK

Multi Architecture High Performance Packet Processing

M Jay

DPDK Presentation March 1 2017

Legal Disclaimer

General Disclaimer:

Technology Disclaimer:

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

Performance Disclaimers (include only the relevant ones):

Cost reduction scenarios described are intended as examples of how a given Intel- based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

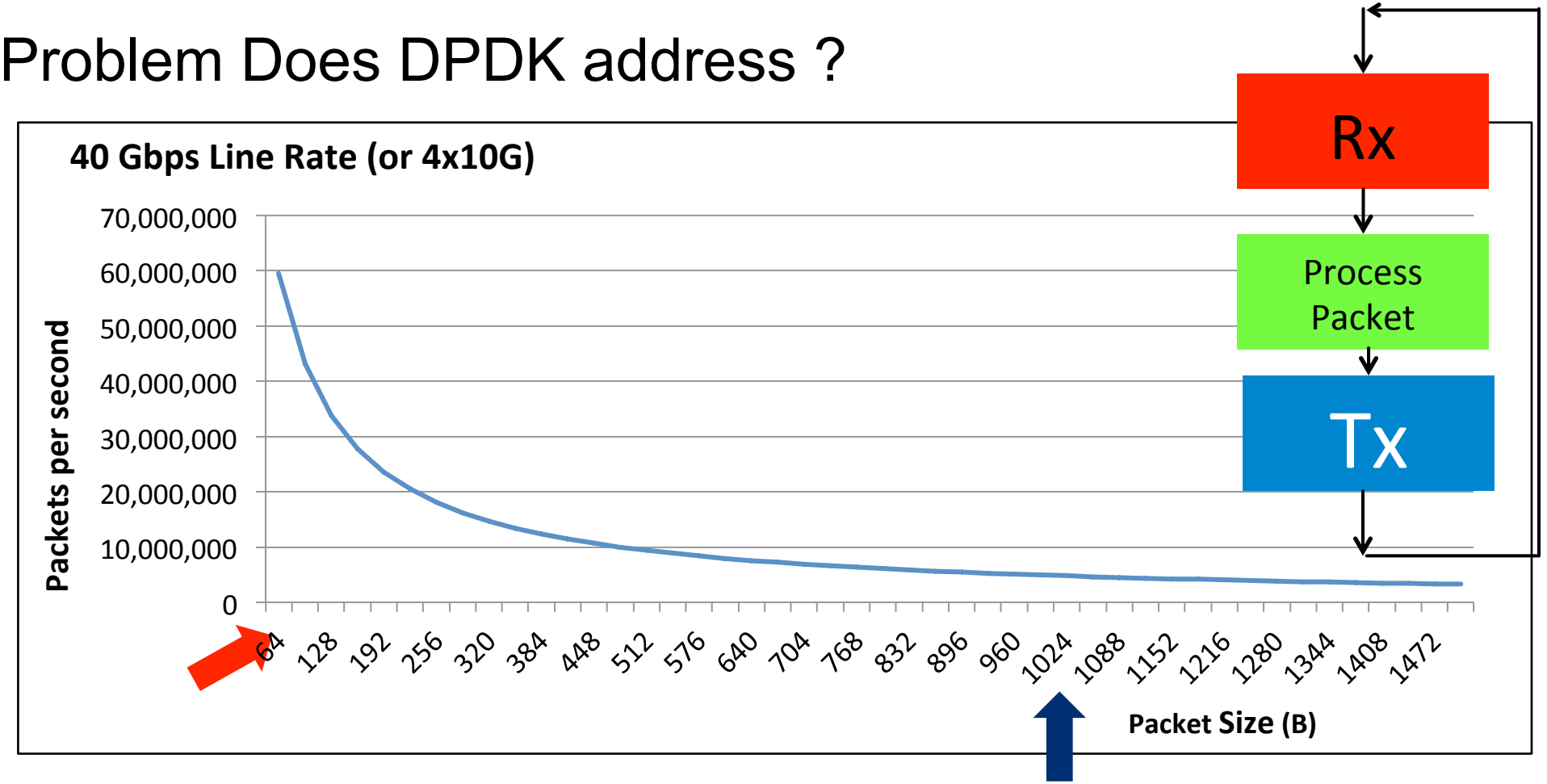
Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

Agenda

- DPDK – Multi Architecture Support
- Why DPDK? - Optimizing Cycles per Packet
- DPDK – Building Block for OVS/NFV
- Enhancing OVS/NFV Infrastructure

- Call To Action

What Problem Does DPDK address ?



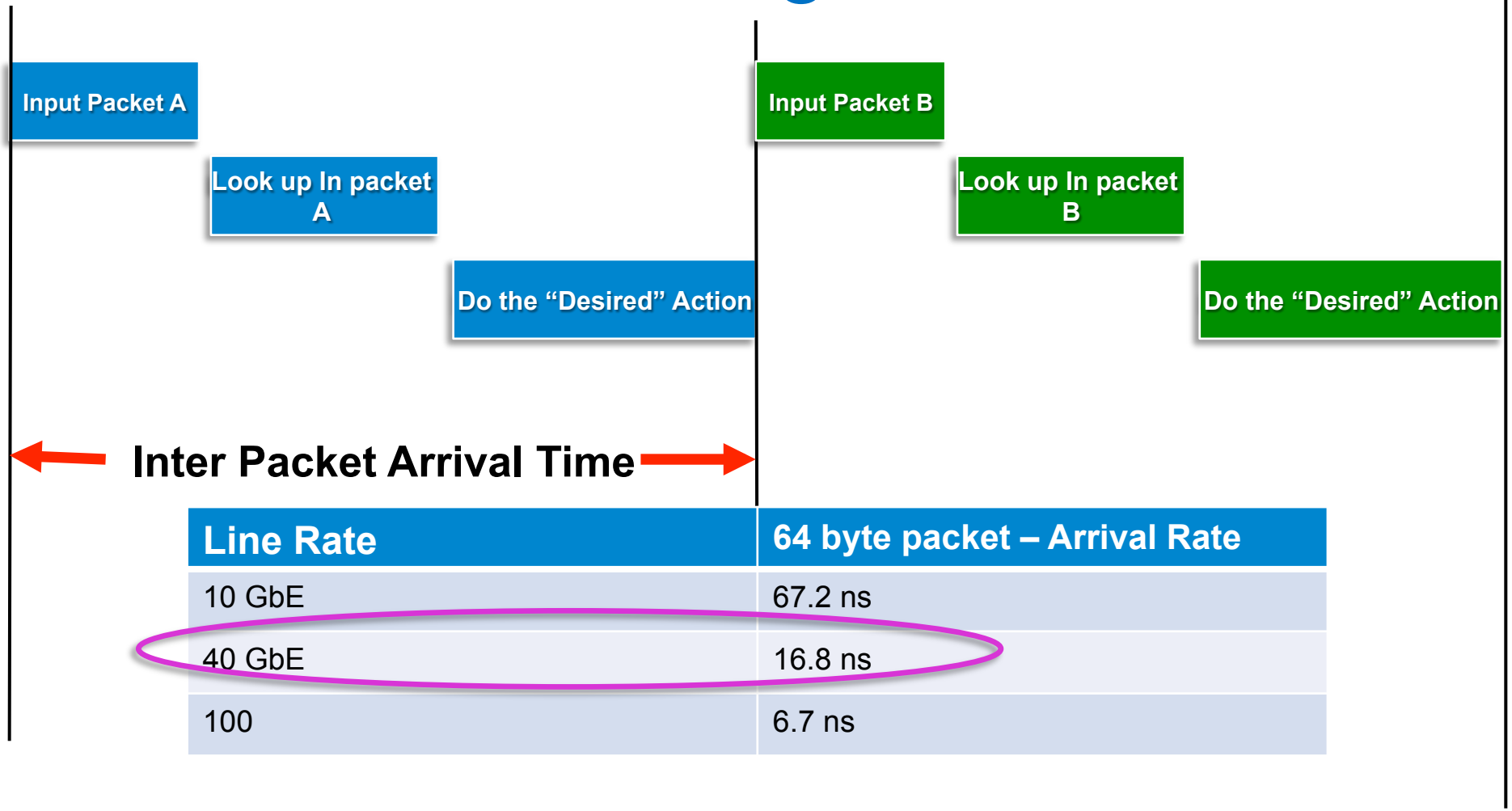
Network Infrastructure Packet Sizes

Packet Size	64 bytes
40G Packets/second	59.5 Million each way
Packet arrival rate	16.8 ns
2 GHz Clock cycles	**33 cycles**

Typical Server Packet Sizes

Packet Size	1024 bytes
40G Packets/second	4.8 Million each way
Packet arrival rate	208.8 ns
2 GHz Clock cycles	**417 cycles**

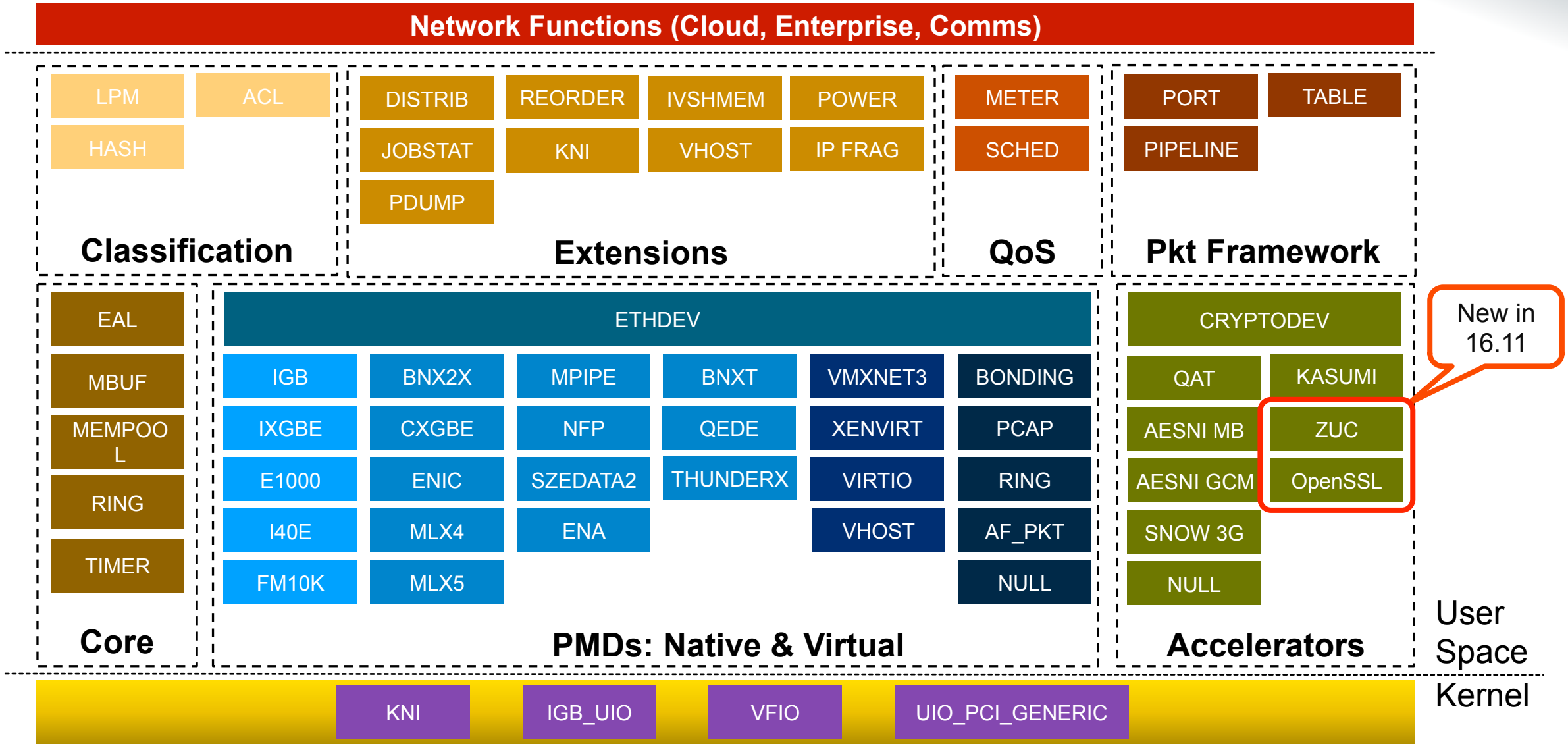
Packet Processing



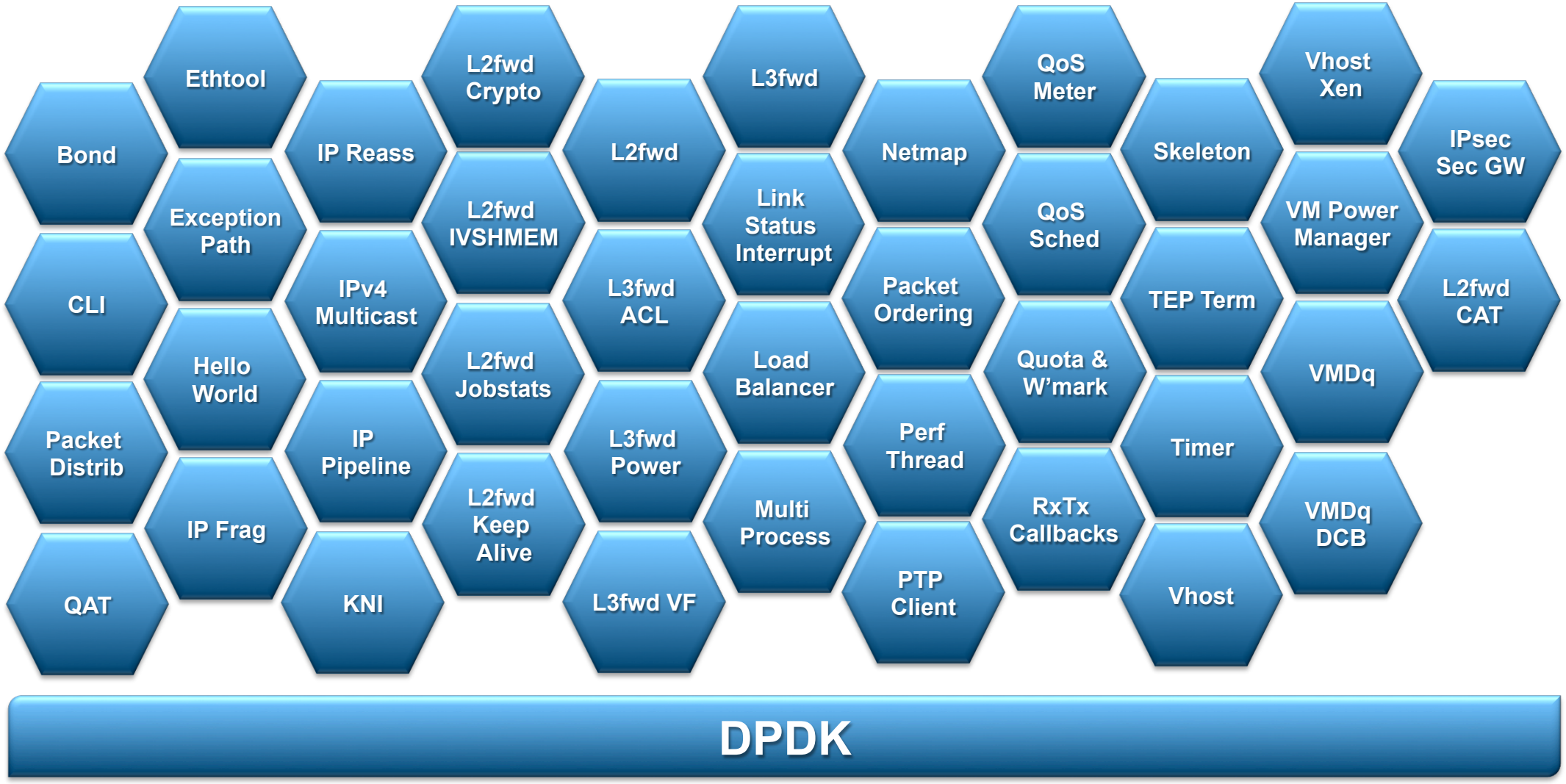
Line Rate	64 byte packet – Arrival Rate
10 GbE	67.2 ns
40 GbE	16.8 ns
100	6.7 ns

Rx Budget = 19 cycles.
Tx Budget = 28 cycles.

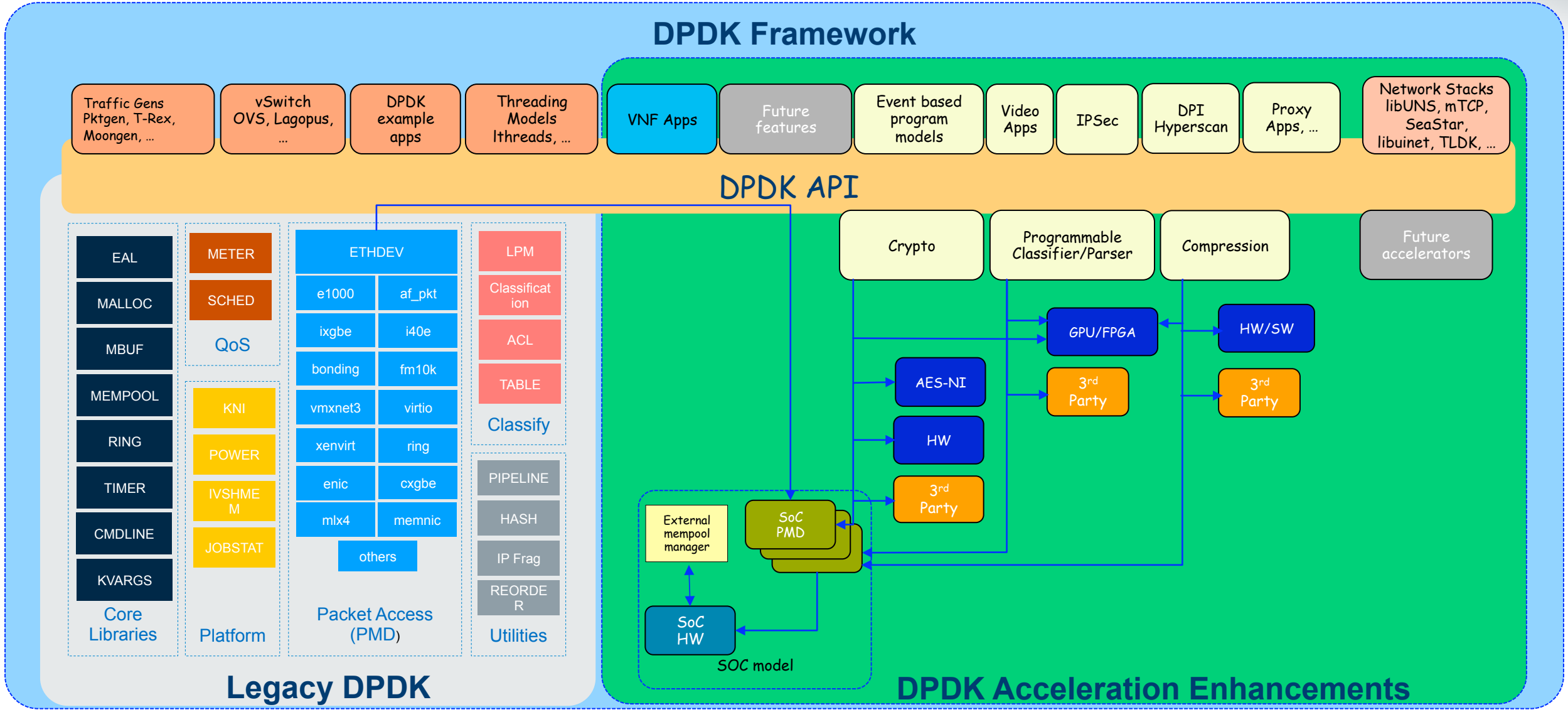
DPDK Framework



DPDK Sample Apps

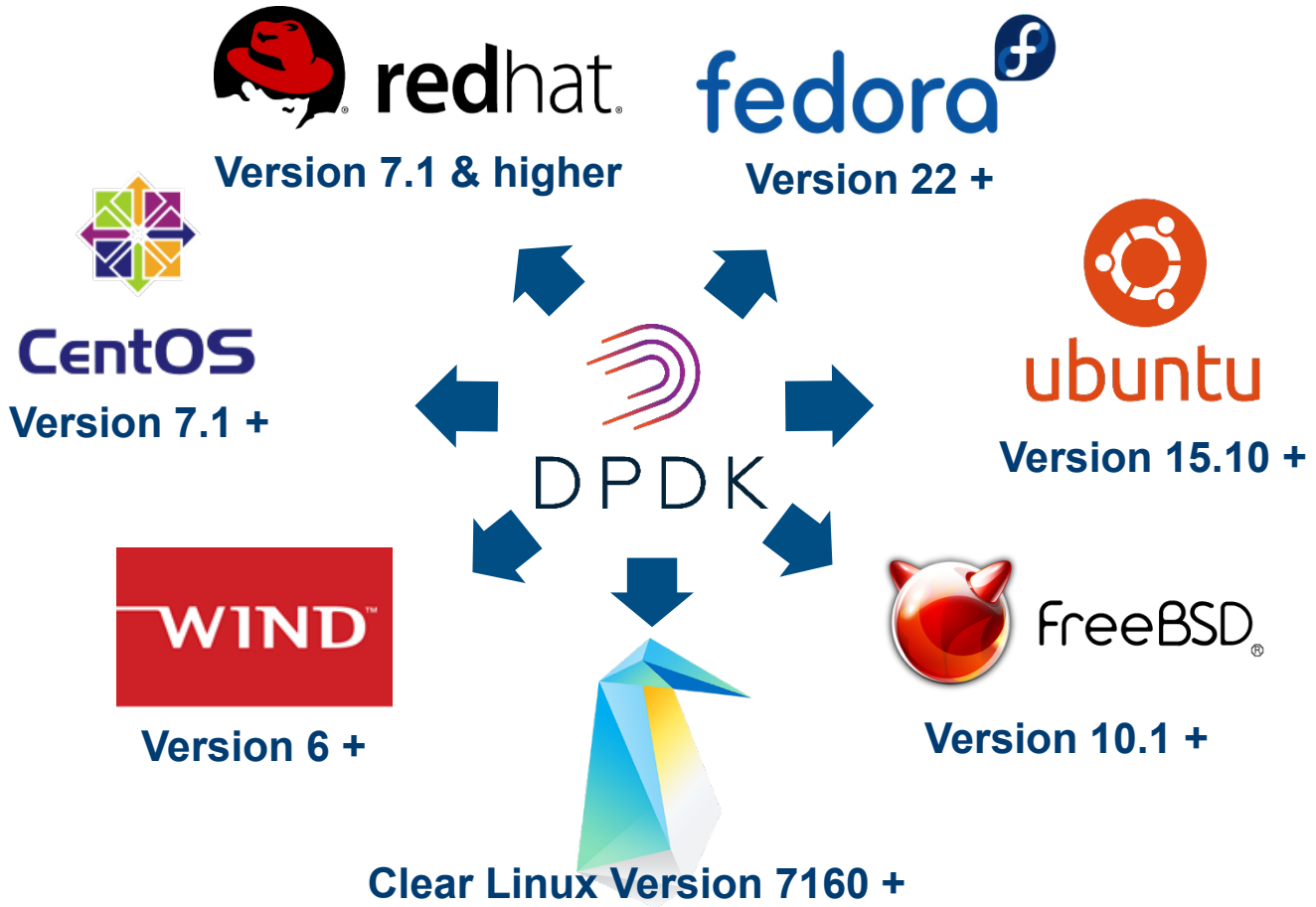


DPDK Acceleration Enhancements



DPDK in OS Distros

DPDK is also available as part of the following OS distributions:

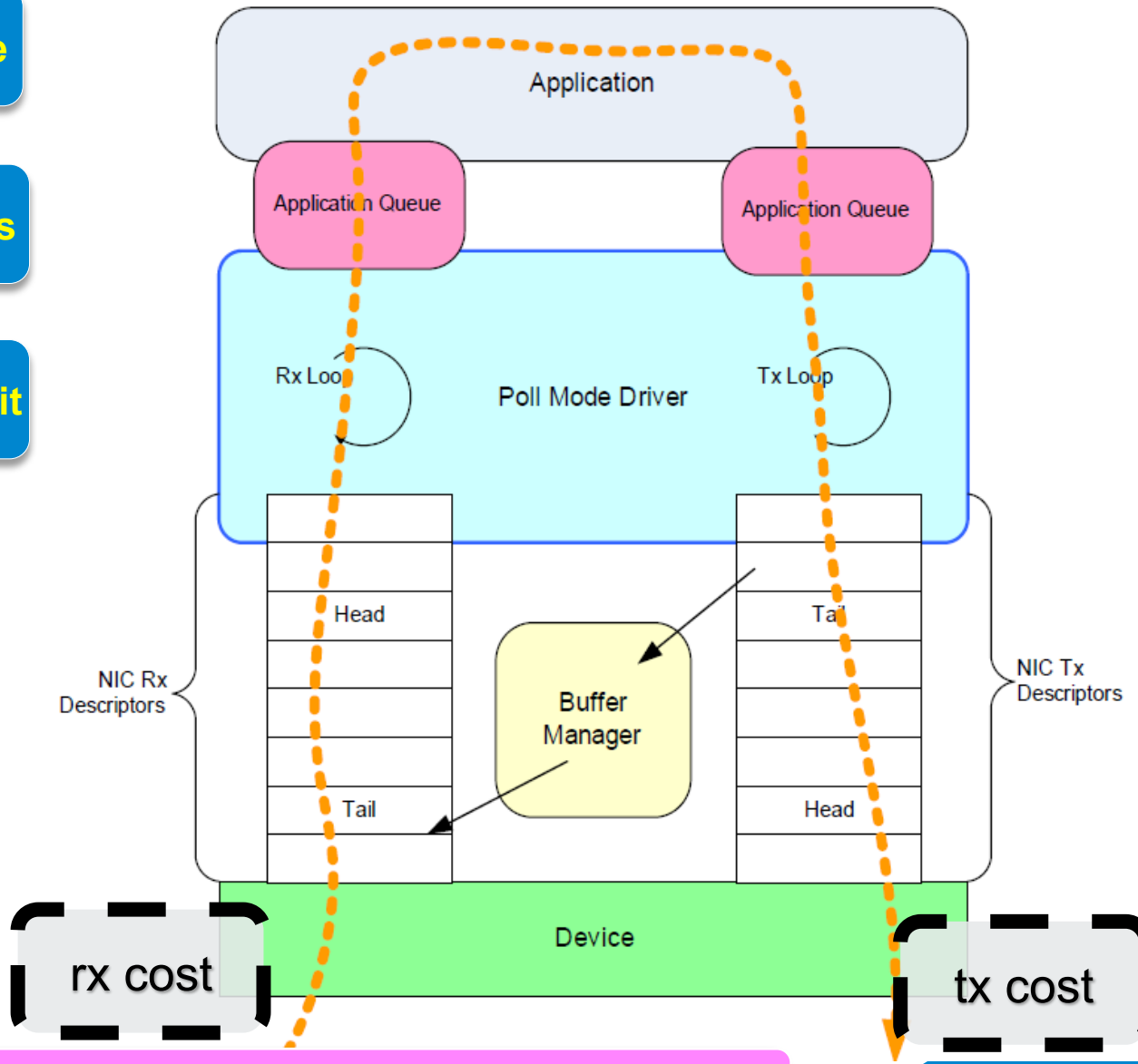


What Is The Task At Hand?

Receive

Process

Transmit



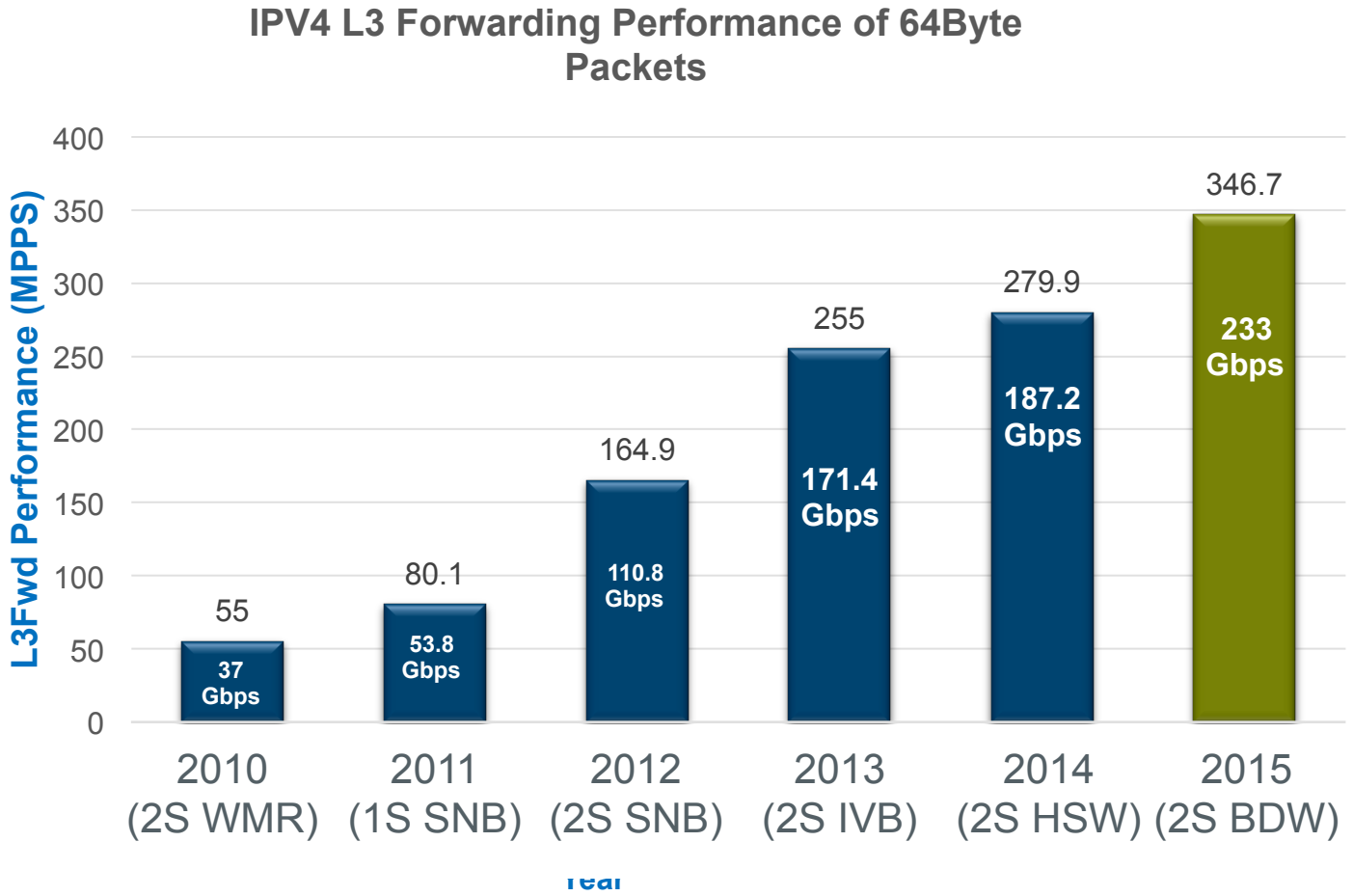
A Chain is only as strong as …..

Benefits – Eliminating / Hiding Overheads

Eliminating	How?	Eliminating /Hiding	How?
Interrupt Context Switch Overhead	Polling	4K Paging Overhead	Huge Page
Kernel User Overhead	User Mode Driver		Lockless Inter-core Communication
Core To Thread Scheduling Overhead	Pthread Affinity	PCI Bridge I/ O Overhead	High Throughput Bulk Mode I/O calls

To Tackle this challenge, what kind of devices /latency we have at our disposal?

DPDK Generational Performance Gains



Broadwell EP System Configuration	
Hardware	
Platform	SuperMicro® - X10DRX
CPU	Intel® Xeon® Processor E5-2658 v4
Chipset	Intel® C612 chipset
Sockets	2
Cores per Socket	14 (28 threads)
LL CACHE	30 MB
QPI/DMI	9.6GT/s
PCIe	Gen3x8
MEMORY	DDR4 2400 MHz, 1Rx4 8GB (total 64GB), 4 Channel per Socket
NIC	10 x Intel® Ethernet CNA XL710-QDA2PCI-Express Gen3 x8 Dual Port 40 GbE Ethernet NIC (1x40G/card)
NIC Mbps	40,000
BIOS	BIOS version: 1.0c (02/12/2015)
Software	
OS	Debian 8.0
Kernel version	3.18.2
Other	DPDK2.2.0

Disclaimer: Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/performance.

What are the top two key performance metrics?

1) Latencies come in all shapes and sizes – What to do?

How Is Latency?

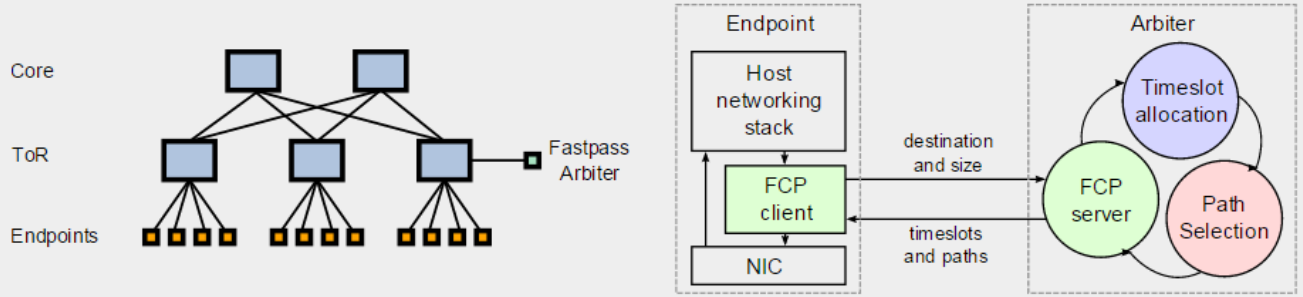
- MIT* white paper on Fast Pass
- Dream of a system with ZERO Queue
- Ultimate testimonial for Latency

Fastpass

A Centralized "Zero-Queue" Datacenter Network

Fastpass is a datacenter network framework that aims for high utilization with zero queueing. It provides low median and tail latencies for packets, high data rates between machines, and flexible network resource allocation policies. The key idea in Fastpass is fine-grained control over packet transmission times and network paths.

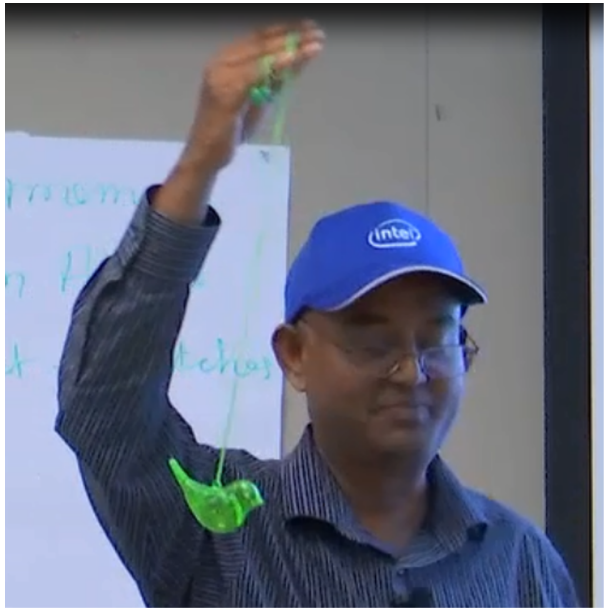
A logically centralized *arbiter* controls and orchestrates all network transfers.



See How DPDK Can Solve Your Latency Concern

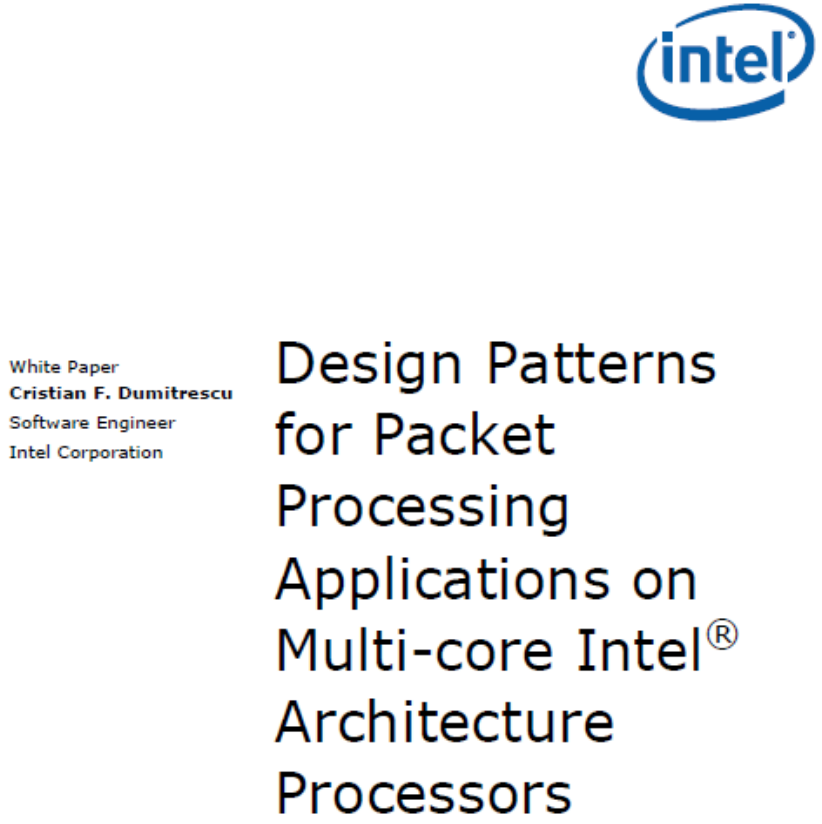
http://fastpass.mit.edu

2) Throughput



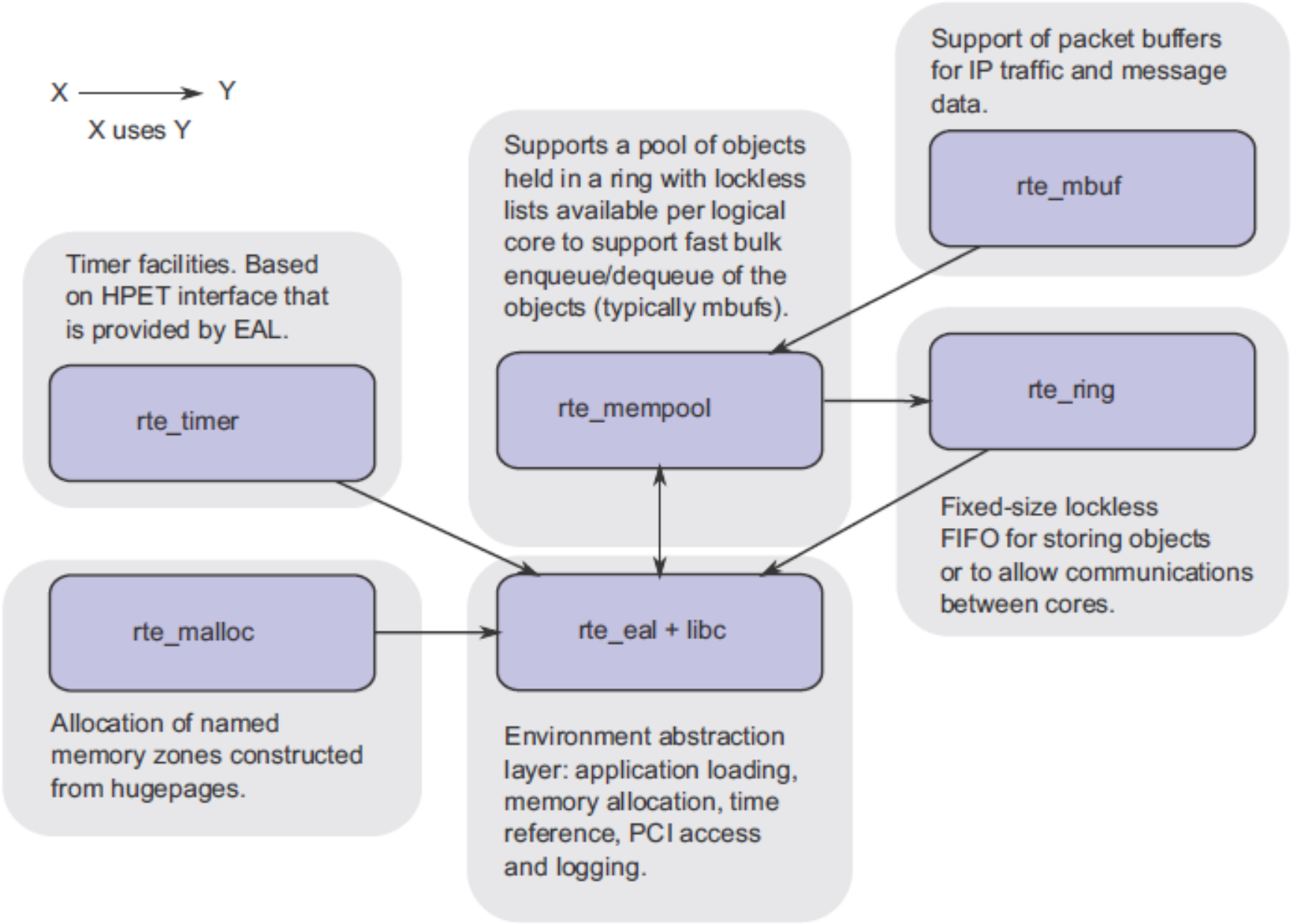
Why go for 1s and 2s while you can take a truck load with **BULK API** ?

Great White Paper – A Must To Read



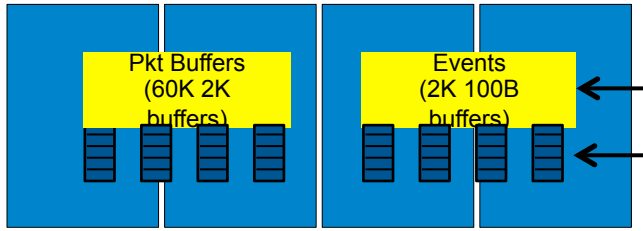
http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-multicore-packet-processing-paper.pdf

Core Components Architecture



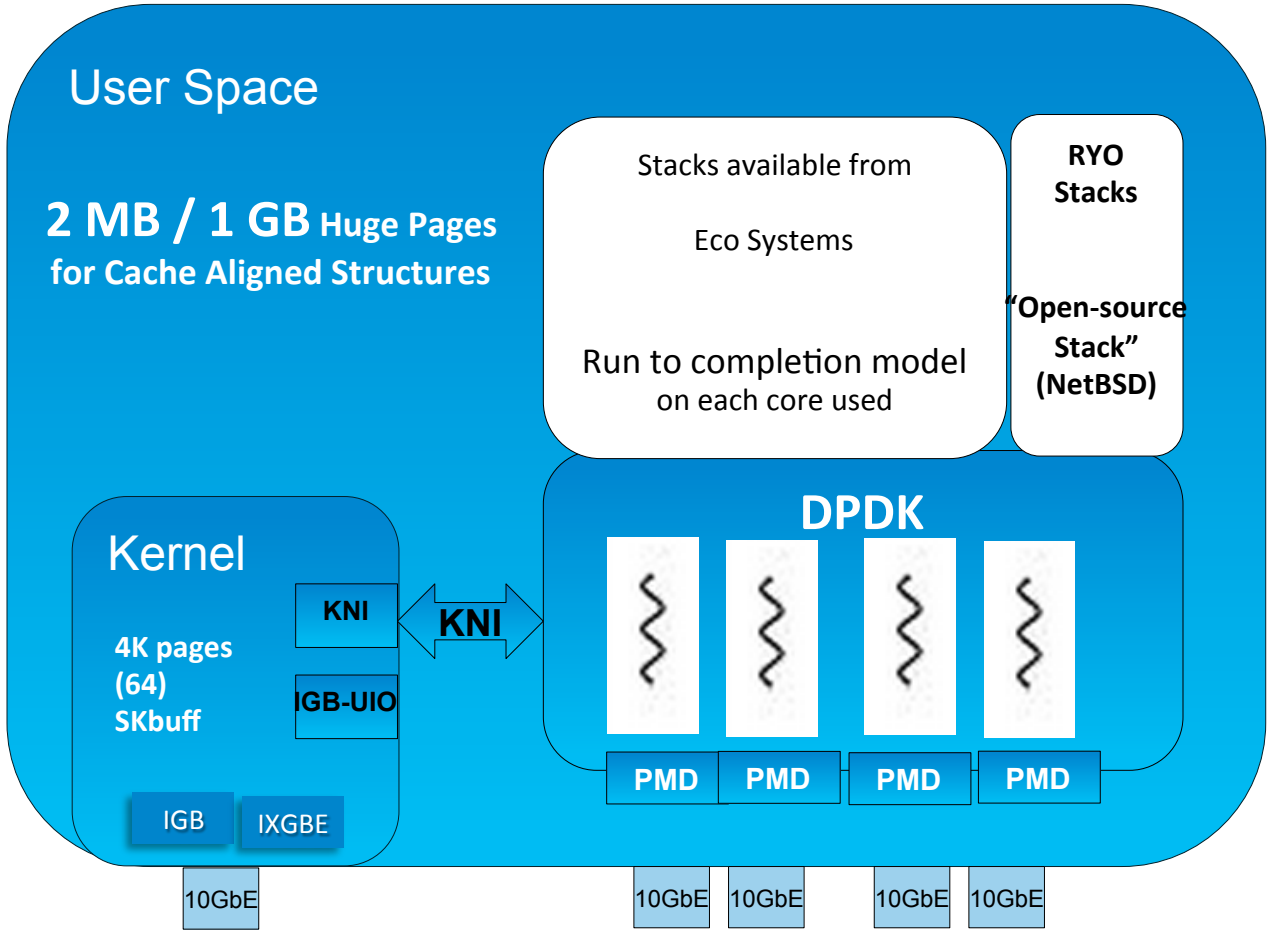
DPDK model

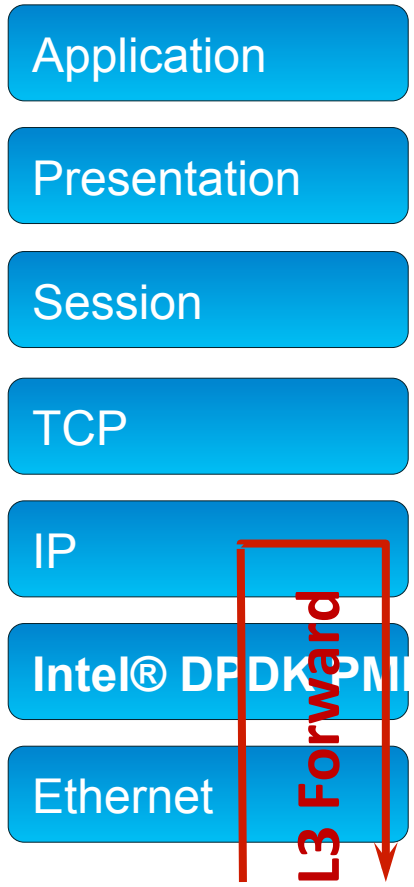
Intel® DPDK allocates packet memory equally across 2, 3, 4 channels.
Aligned to have equal load over channels



Rings for cached buffers

Per core lists, unique per lcore. Allows packet movement without locks





High Performance Components of DPDK

Environment Abstraction Layer

- Abstracts huge-page file system, provides multi-thread and multi-process support, etc.

Memory Manager

- Responsible for allocating pools of objects in memory. A pool is created in huge page memory space and uses a ring to store free objects. It also provides an alignment helper to ensure that objects are padded to spread them equally on all DRAM channels.

Buffer Manager

- Reduces by a significant amount the time the operating system spends allocating and de-allocating buffers. The Intel® DPDK pre-allocates fixed size buffers which are stored in memory pools.

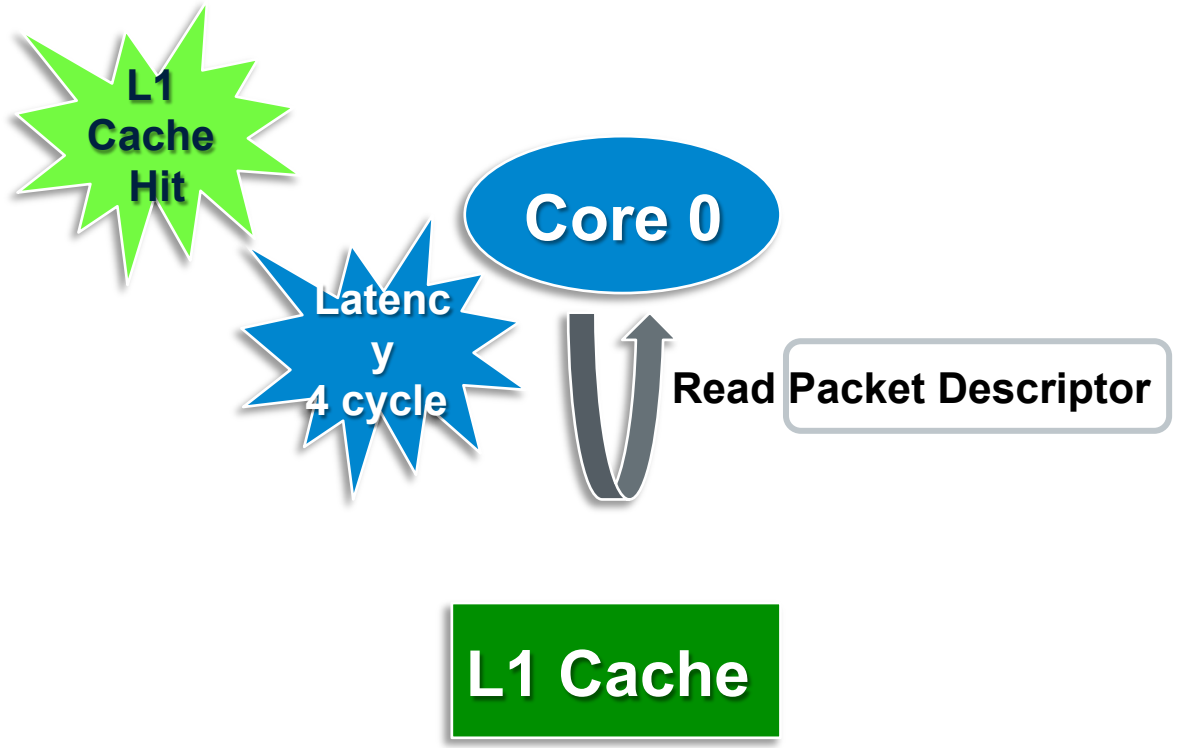
Queue Manager

- Implements safe lockless queues, instead of using spinlocks, that allow different software components to process packets, while avoiding unnecessary wait times.

Flow Classification

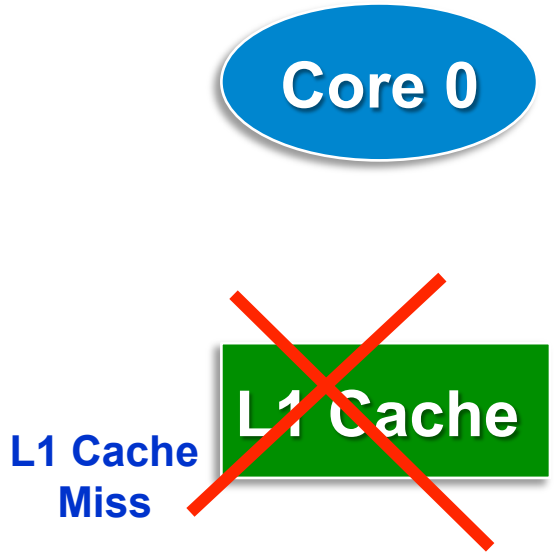
- Provides an efficient mechanism which incorporates Intel® Streaming SIMD Extensions (Intel® SSE) to produce a hash based on tuple information so that packets may be placed into flows quickly for processing, thus greatly improving throughput.

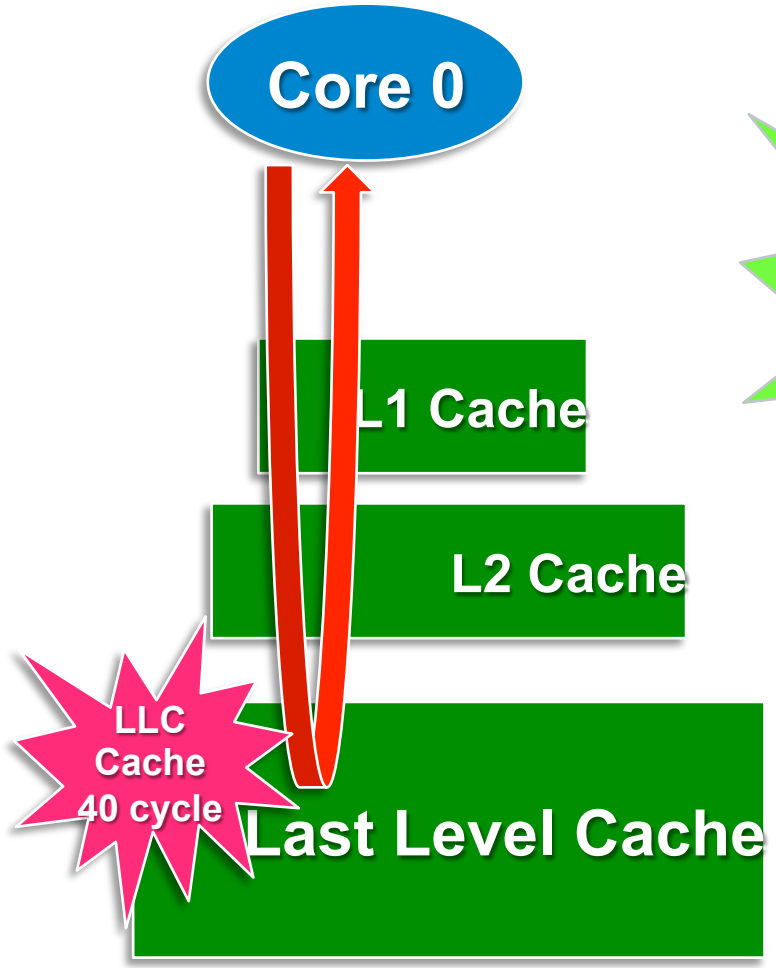
L1 Cache With 4 Cycle Latency



With 4 cycles Latency, achieving Rx budget of 19 cycles seems within reach.

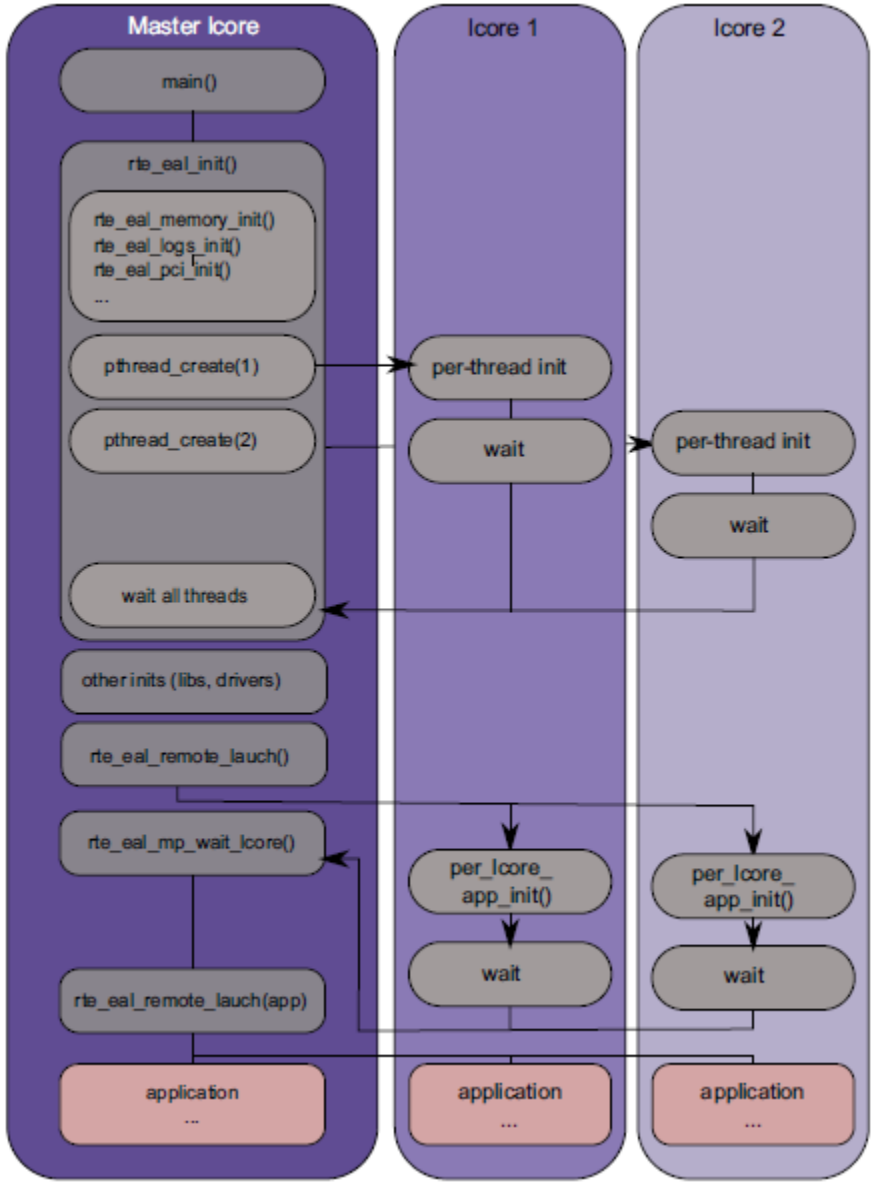
Challenge: What if there is L1 Cache Miss and LLC Hit?



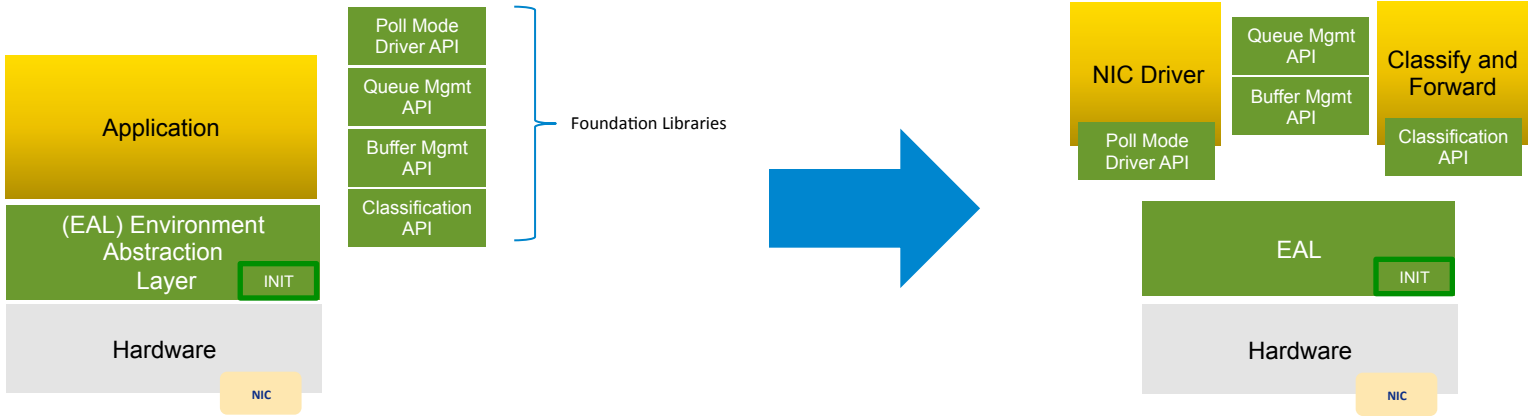


With 40 cycles LLC Hit, How will you achieve Rx budget of 19 cycles ?

EAL Initialization in a Linux Application Environment



DPDK: Overview of components



Generalized Overview

EAL is primarily initialization code

- Bootstrap processor startup, MP-Init
- PCI Scan for supported devices (NIC, CPM)
- Console, Keyboard, other services initialization

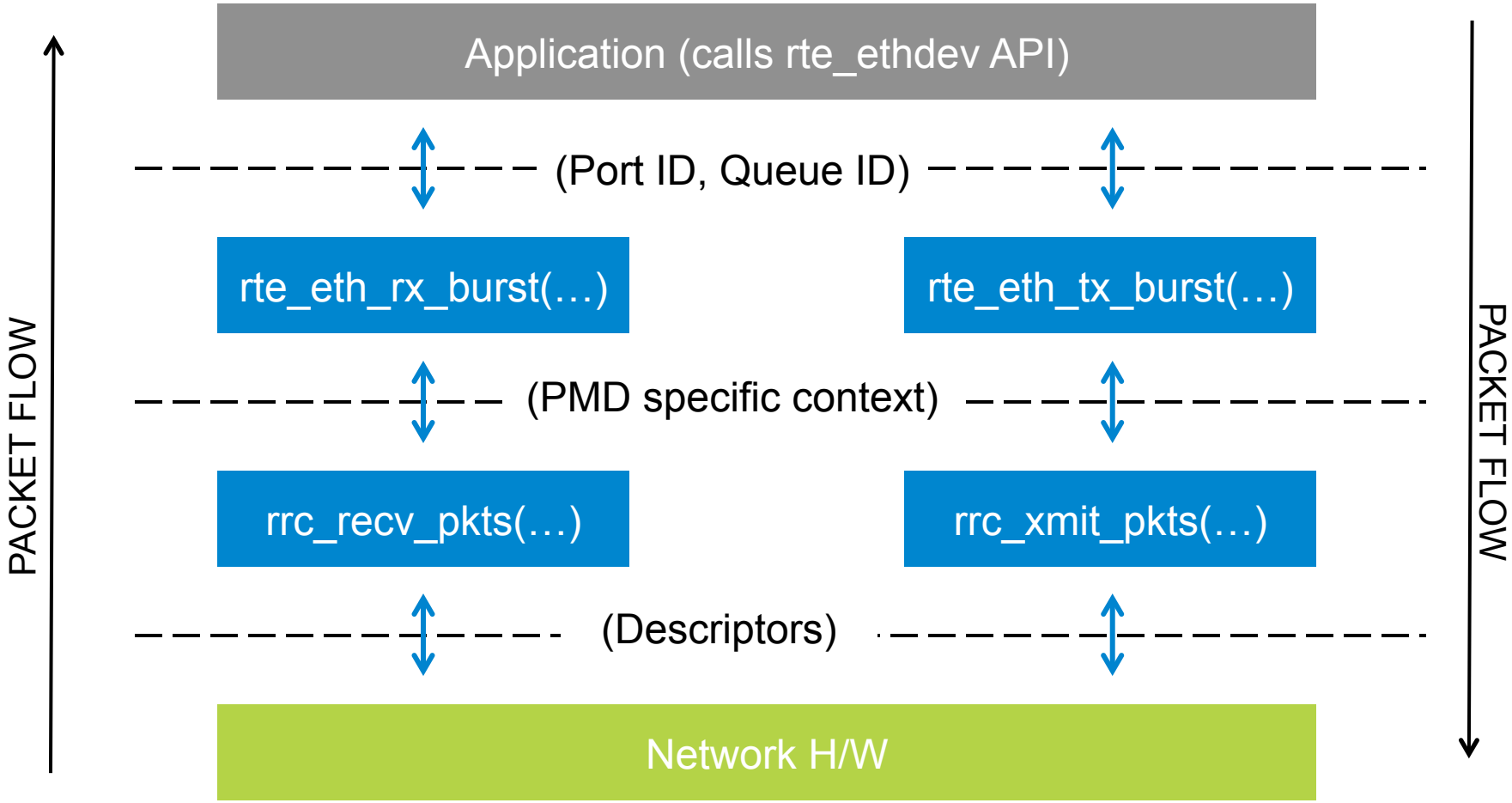
Ends with each logical core (execution unit) running its own dispatch loop

- Typically bootstrap core (EU0) initializes all foundation library services

Example of a IPv4 L3 Forwarding Instantiation

- EAL dispatch loop has two applications
  - NIC Driver & Classify+Forward apps
  - Can be run on one core or split across multiple cores
  - Queue Management library serves as means of communication between applications

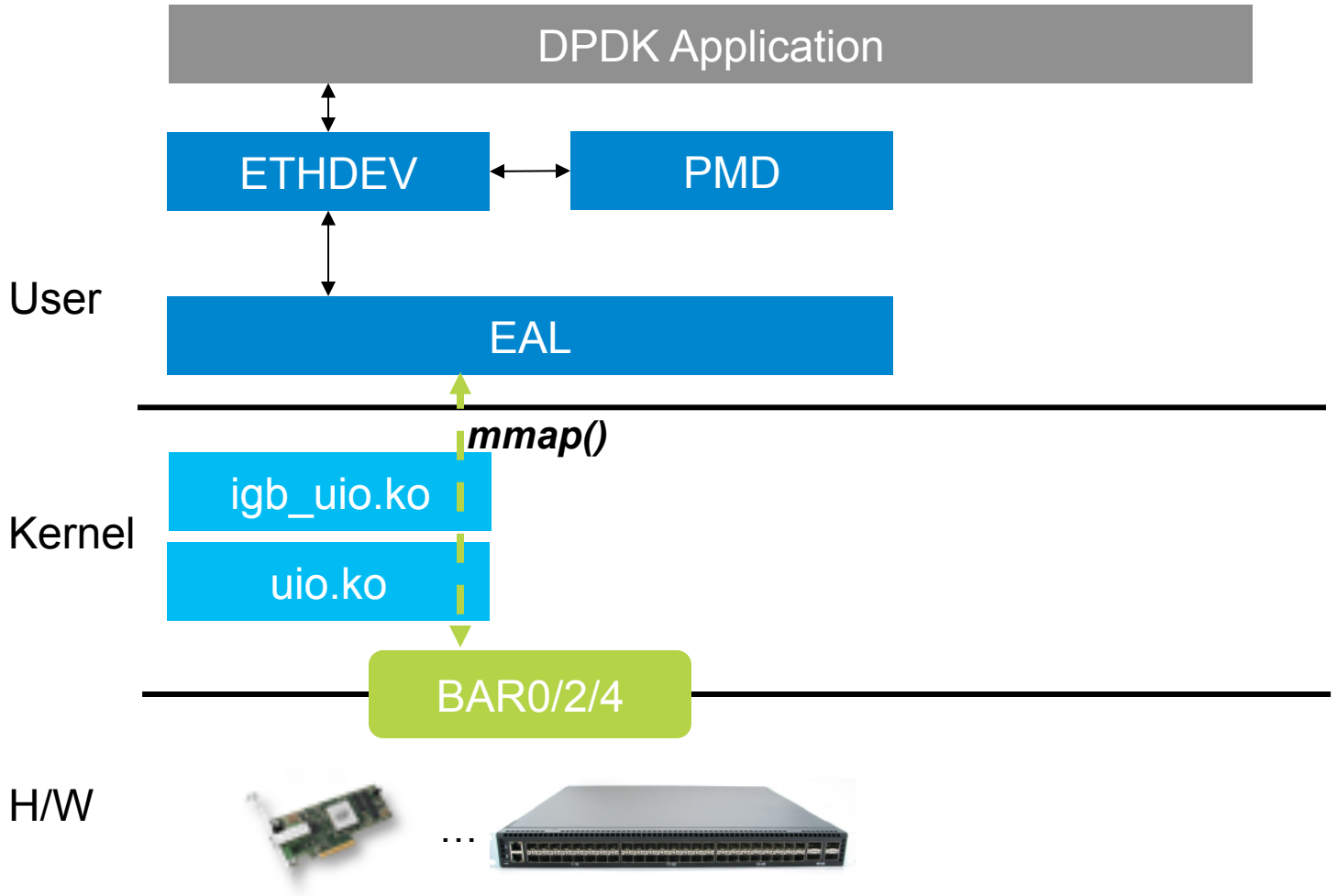
Ethernet Device Framework



Userspace I/O (UIO)

https://www.kernel.org/doc/htmldocs/uio-howto/

UIO Picture


DPDK Application
ETHDEV
PMD
User
EAL
mmap()
igb_uio.ko
Kernel
uio.ko
BAR0/2/4
H/W
...

UIO

Only one small kernel module to write and maintain (igb_uio.ko).

Develop the main part of the driver in user space, with all the tools and libraries you're used to.

Bugs in the driver won't crash the kernel.

Updates of the driver can take place without recompiling the kernel.

User/Admin binds PCI devices to igb_uio

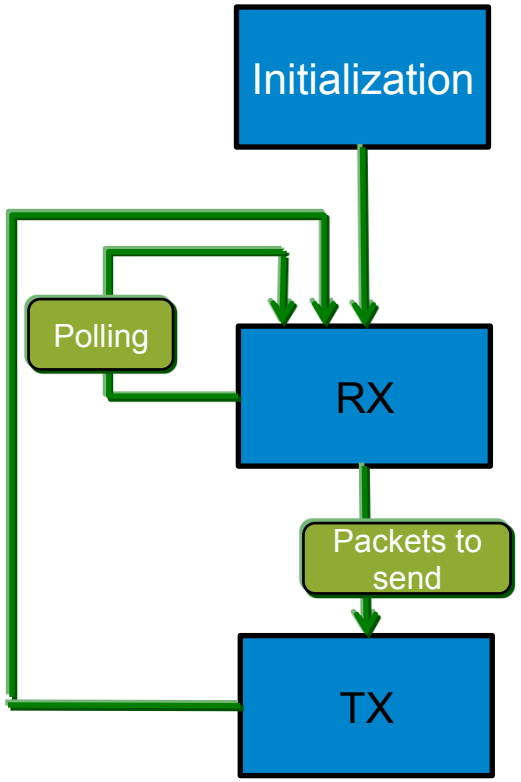
UIO Framework creates /dev/uioX, and sysfs files describing BAR regions (address, size)

DPDK scans PCI bus looking for devices matching any of it’s PMDs

If a matching Driver is found, DPDK maps BAR regions into Userspace, and calls the initialization function originally registered by the PMD

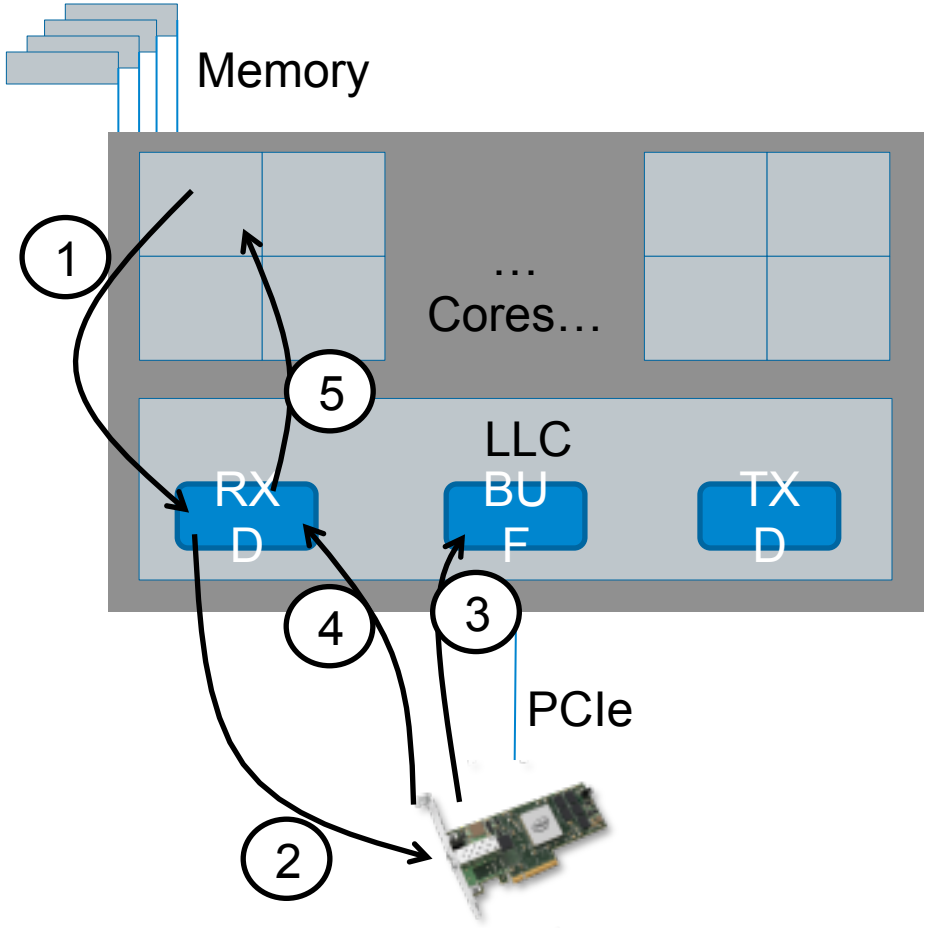
Poll Mode Driver (PMD) - Rx & Tx Overview

30,000 ft overview of packet flow



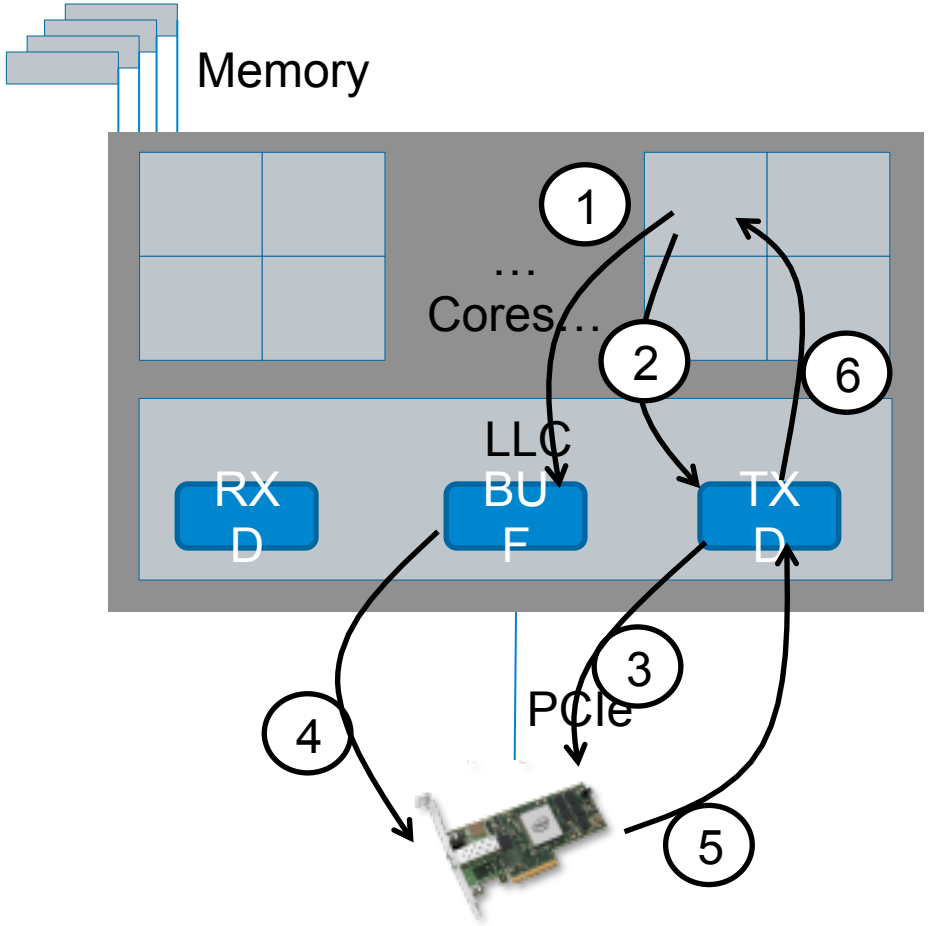
1. Initialization
   - Init Memory Zones and Pools
   - Init Devices and Device Queues
   - Start Packet Forwarding Application
2. Packet Reception (RX)
   - Poll Devices' RX queues and receive packets in bursts
   - Allocate new RX buffers from per queue memory pools to stuff into descriptors
3. Packet Transmission (TX)
   - Transmit the received packets from RX
   - Free the buffers that we used to store the packets

Rx Overview



1. CPU Write Rx descriptor
2. NIC Read Rx descriptor to get buffer address
3. NIC Write Rx packet to buffer address
4. NIC Write Rx descriptor
5. CPU Read Rx descriptor (polling)

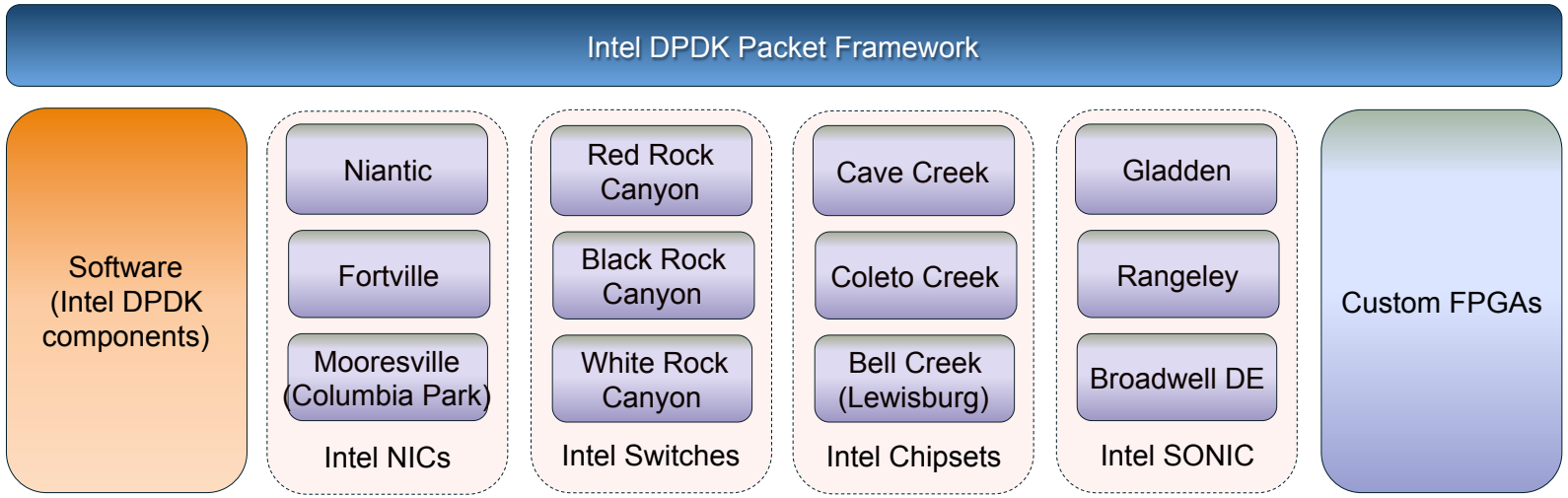
Tx Overview



1. CPU Write data
2. CPU Write Tx descriptor
3. NIC Read Tx descriptor to get buffer address
4. NIC Read Tx packet from buffer address
5. NIC Write Tx descriptor
6. CPU Read Tx descriptor

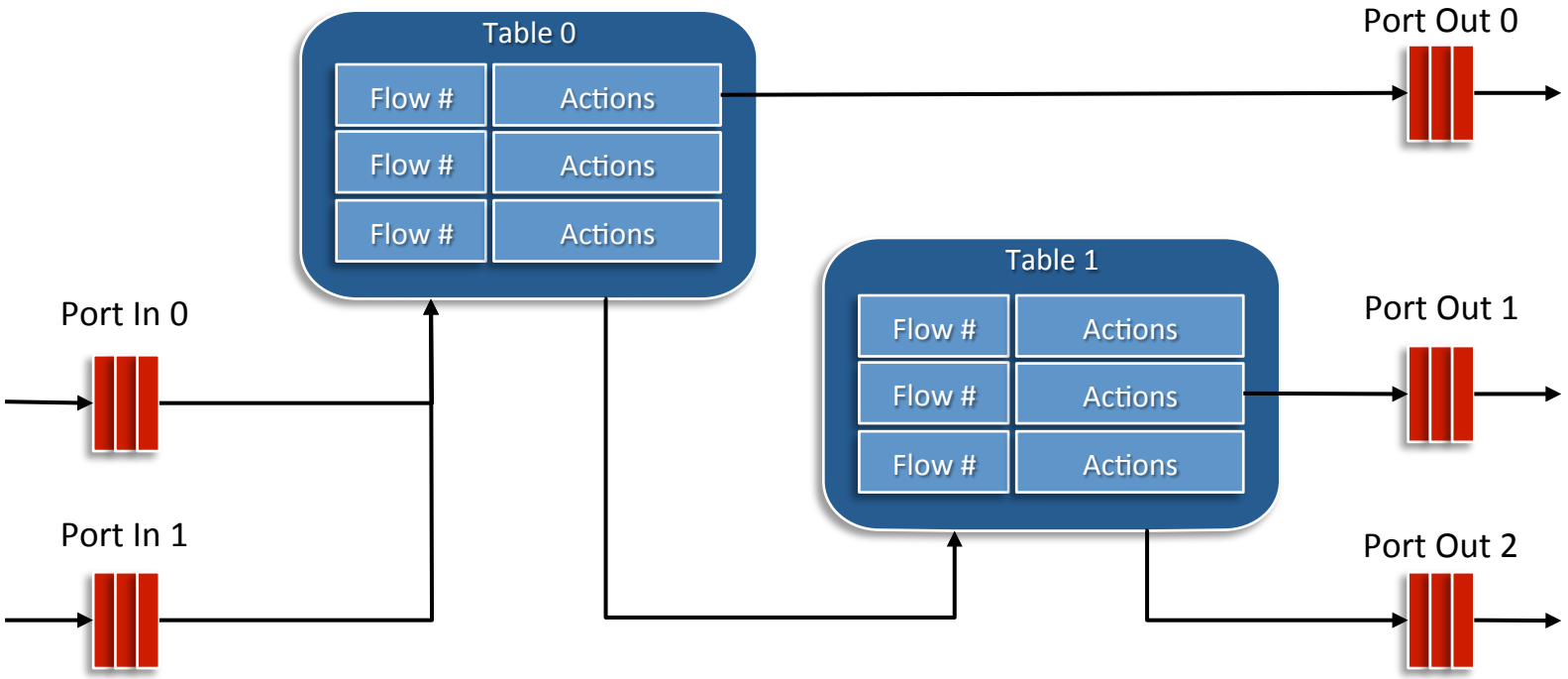
Why Packet Framework?

- **Intel devices with increased acceleration capability need to be complemented by SW to enable complete functionality**
- **Intel DPDK provides highly optimized SW primitives that can be further accelerated by Intel HW**



Combine the best Intel HW with the best Intel SW to achieve the best functionality and performance

What is it?



Standard methodology for ***pipeline*** development.
Ports and ***tables*** are connected together in tree-like topologies , with tables providing the ***actions*** to be executed on input packets.

Example

#	Port	Description
1	SW Ring	Circular memory buffer used for message passing between CPU cores
2	HW Ring	Circular memory buffer used for sending/receiving packets from/to NIC ports
3	IP Reassembler	Input packets are IP fragments. Output packets are reassembled IP datagrams.
4	IP Fragmentator	Input packets are IP datagrams with length bigger than MTU. Output packets are IP fragments.
5	Traffic Manager	Traffic manager attached to a NIC output port performing congestion management and hierarchical scheduling.
6	KNI Port	Send/receive packets to/from Linux kernel devices
7	Berkeley Socket	TCP/IP stack endpoint
8	Source Port	Packet generator port (/dev/zero)
9	Sink Port	Packet terminator port (/dev/null)

#	Table	Description
1	Hash Table	Exact match table. Used for flow classification tables, ARP caches, tunneling routing tables, etc.
2	ACL Table	Access Control Lists. Used for policy databases (firewall, etc).
3	LPM Table	Longest Prefix Match. Used for IPv4/IPv6 routing tables.
4	Pattern Matching	Pattern database. Used for IPS, anti-virus, etc.
5	Load Balancer	Distribute the stream of input packets to a set of output ports while preserving the packet order for each flow.

Actions

- Assigned per table
- executed in priority order on all packets that share the current action before moving to the next action (as opposed to all actions for one packet at a time)
- If (fn0) call next fn() else stop

#	User Action	Description
1	Drop	Accept/deny flow
2	NAT / PAT	Translate between internal (LAN) and external IP addresses, ports (and VLAN labels)
3	Meter	Per flow traffic metering (srTCM, trTCM) and policing
4	App ID	Per flow state machine fed by sequence of packets
5	Push/pop	Push/pop VLAN or MPLS labels
6	Decrement TTL	Decrement IPv4/IPv6 TTL (and update checksum)
7	Statistics	Update byte or packet statistics
8	Encrypt/ decrypt	
9	Compress/ decompress	

Memory Pools & Per-Core Cache

Object

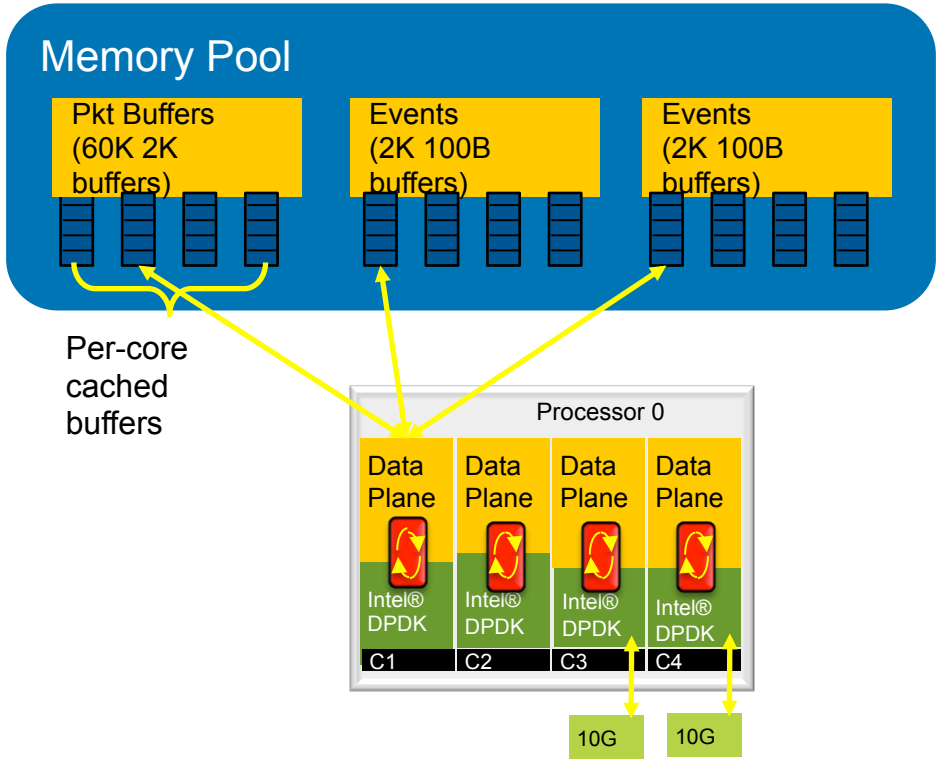
Size fixed at creation time:
- Fixed size elements
- Fixed number of elements

Multi-producer/multi-consumer safe

Safe for fast-path use

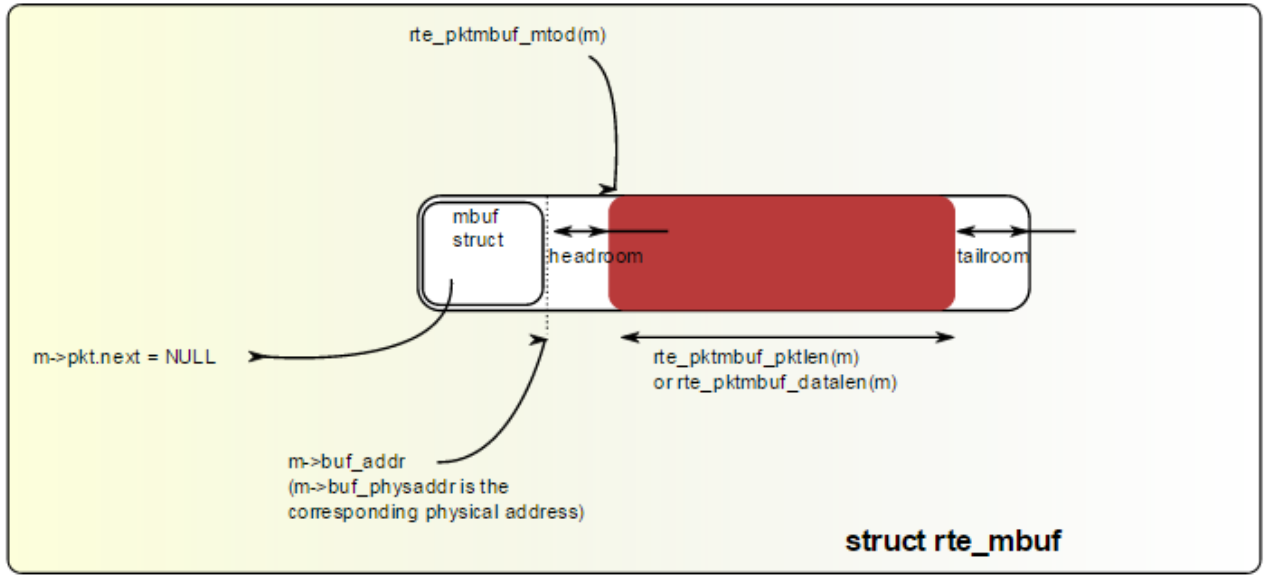
Typical usage is packet buffers

Optimized for performance:
- No locking, use CAS instructions
- All objects cache aligned
- Per core caches to minimise contention/use of CAS instructions
- Support for bulk allocation/freeing of buffers

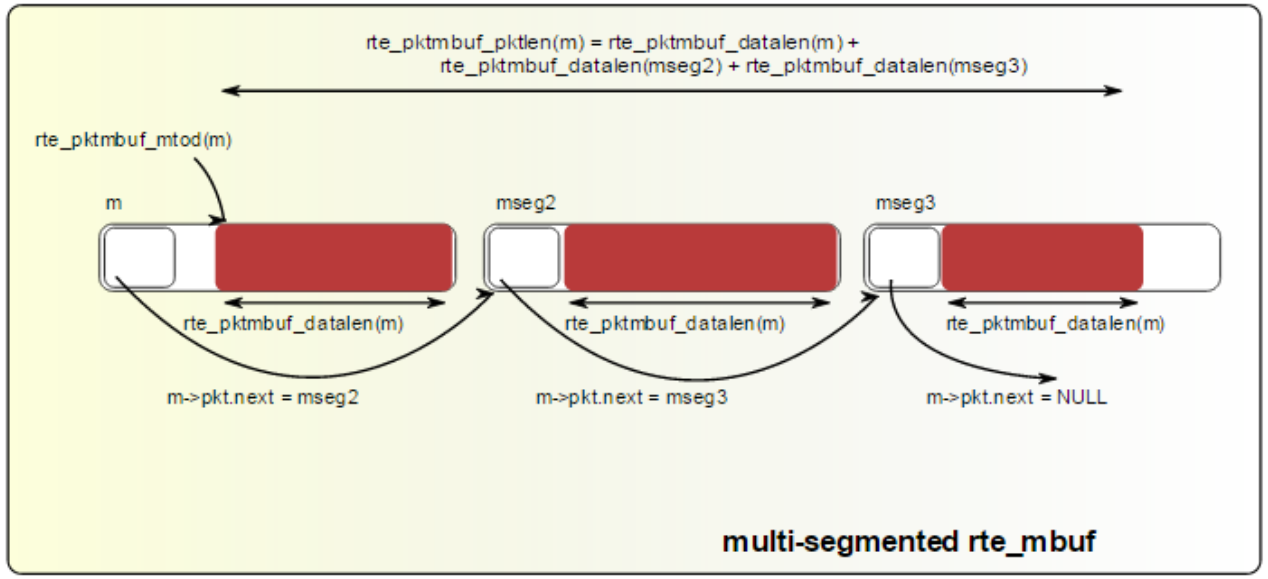


You can implement S/W caches of Large Structures private to Each Core

An mbuf with One Segment



An mbuf with Three Segments



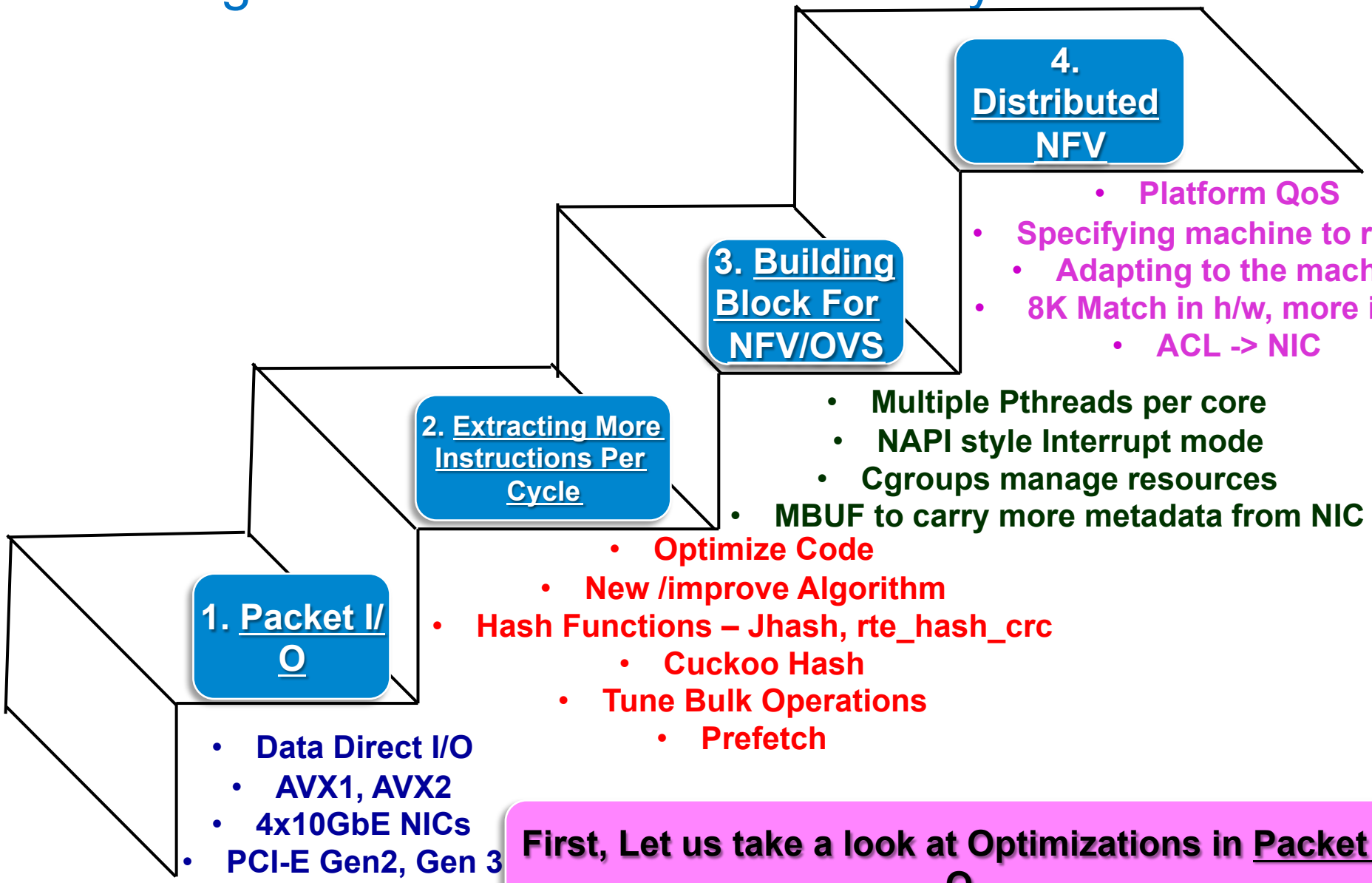
3. Building Block For NFV/OVS

Mbuf To Carry More Metadata From NIC

```
#define RTE_PTYPE_TUNNEL_VXLAN          0x00003000
/**
 * NVGRE (Network Virtualization using Generic Routing Encapsulation) tunneling
 * packet type.
 *
 * Packet format:
 * <'ether type'=0x0800
 * | 'version'=4, 'protocol'=47
 * | 'protocol type'=0x6558>
 * or,
 * <'ether type'=0x86DD
 * | 'version'=6, 'next header'=47
 * | 'protocol type'=0x6558'>
 */
#define RTE_PTYPE_TUNNEL_NVGRE          0x00004000
/**
 * GENEVE (Generic Network Virtualization Encapsulation) tunneling packet type.
 *
 * Packet format:
 * <'ether type'=0x0800
 * | 'version'=4, 'protocol'=17
 * | 'destination port'=6081>
 * or,
 * <'ether type'=0x86DD
 * | 'version'=6, 'next header'=17
 * | 'destination port'=6081>
 */
#define RTE_PTYPE_TUNNEL_GENEVE         0x00005000
/**
 * Tunneling packet type of Teredo, VXLAN (Virtual eXtensible Local Area
 * Network) or GRE (Generic Routing Encapsulation) could be recognized as this
 * packet type, if they can not be recognized independently as of hardware
 * capability.
 */
```

http://www.dpdk.org/browse/dpdk/tree/lib/librte_mbuf/rte_mbuf.h

DPDK Trail Blazing - Performance & Functionality



First, Let us take a look at Optimizations in Packet I/O

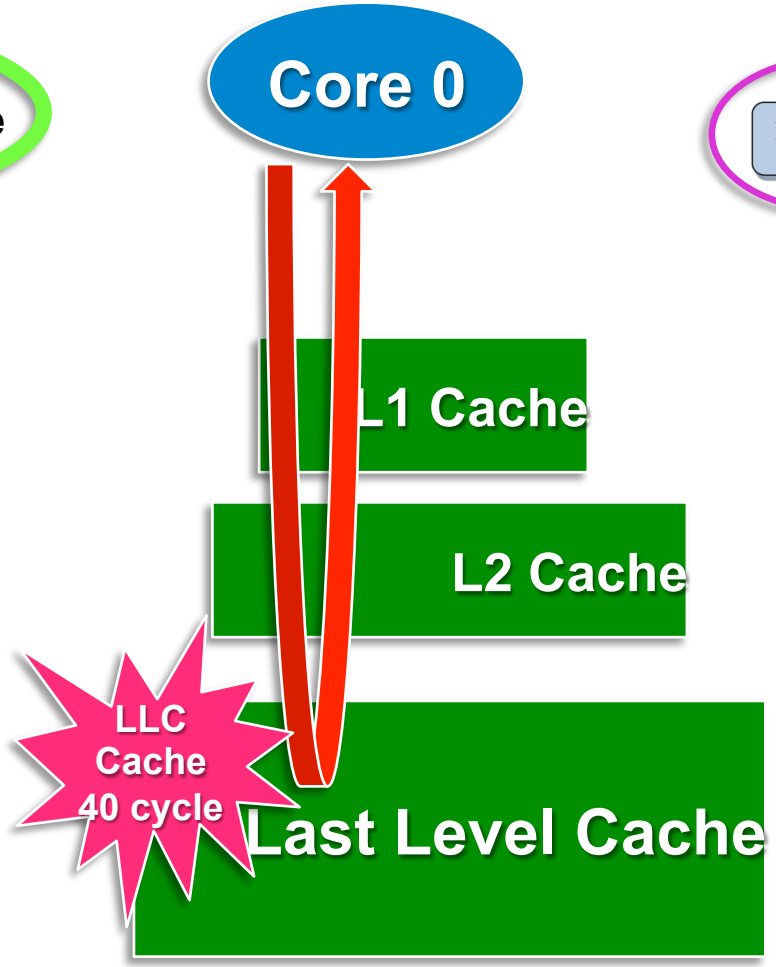
Solution – Amortizing Over Multiple Descriptors

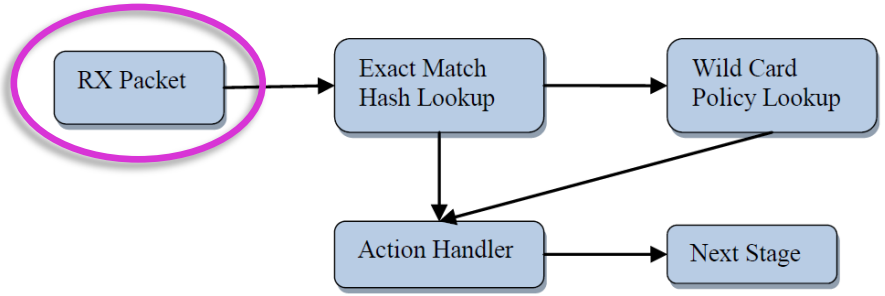
- 40 ns gets Amortized Over Multiple Descriptors
- Roughly getting back to the latency of L1 cache hit per packet
- Similarly for packet i/o, Go For Burst Read

Examine Bunch Of Descriptors At A Time

Read 8 Packet Descriptors at a time

- Packet Descriptor 0
- Packet Descriptor 1
- Packet Descriptor 2
- Packet Descriptor 3
- Packet Descriptor 4
- Packet Descriptor 5
- Packet Descriptor 6
- Packet Descriptor 7





With 8 Descriptors, 40 ns gets amortized over 8 Descriptors

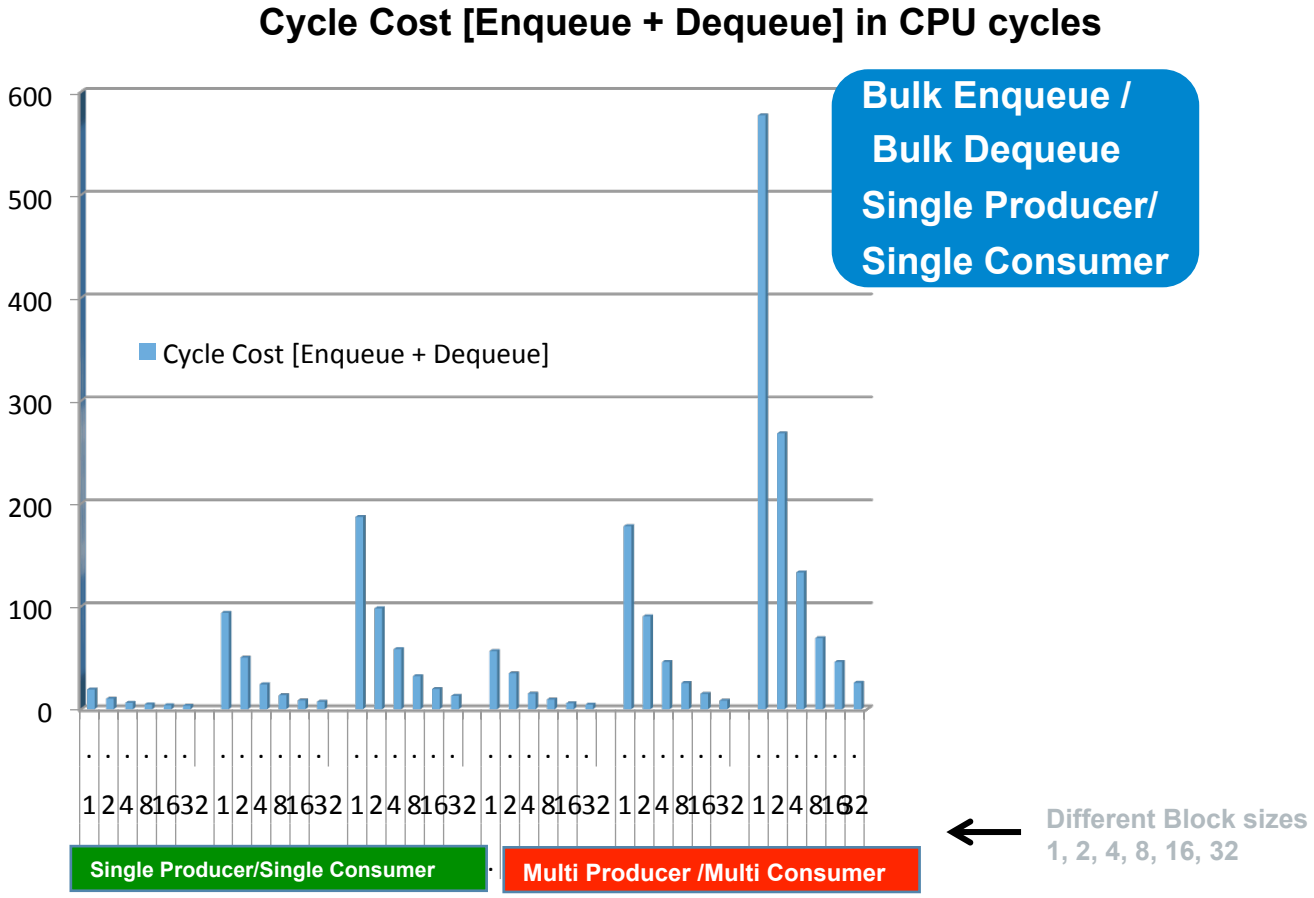
Design Principle In Packet I/O Optimization

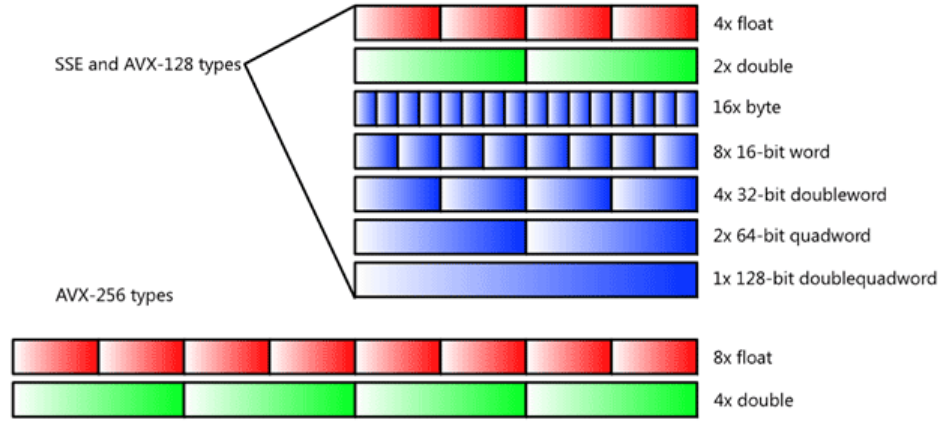
L3fwd default tuning is for performance

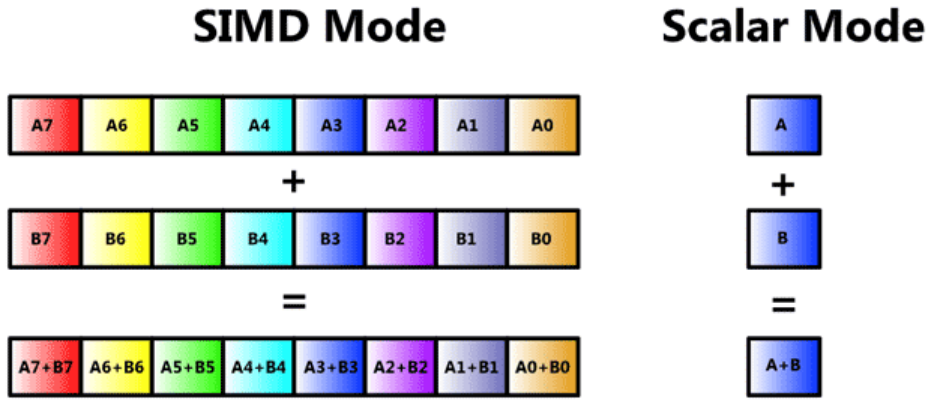
- Coalesces packets up to 100 us
- Receives and transmits at least 32 packets at a time
  - nb_rx = rte_eth_rx_burst(portid, queueid, pkts_burst, **MAX-PKT_BURST**)

Could bunch 8,4, 2 (or 1) packets

Micro BenchMarks – The Best Kept Secre



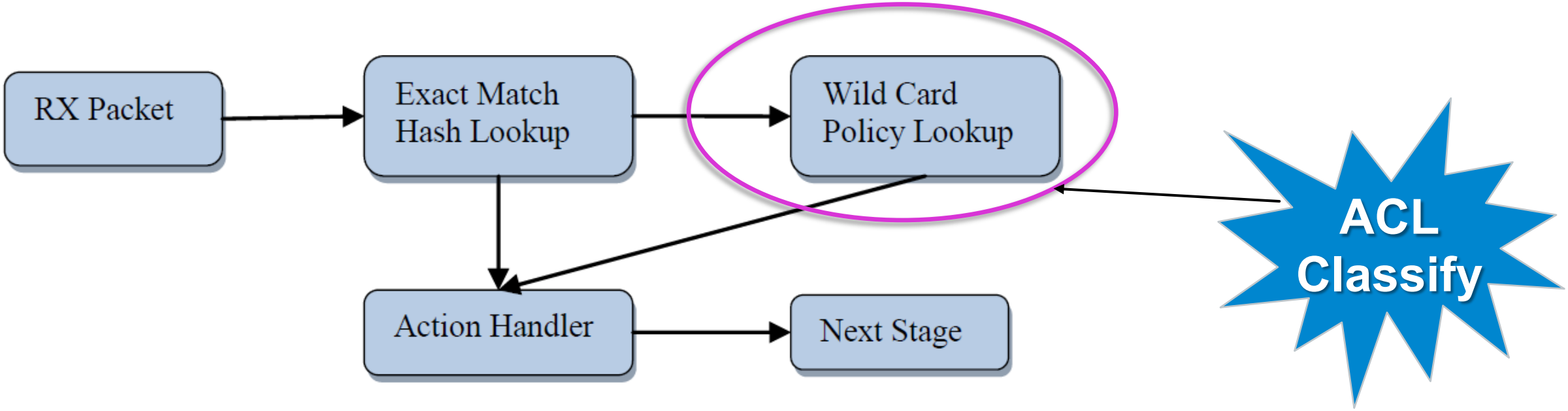




SSE – 4 Lookups in Parallel

Next: 2. Extracting More Instructions Per Cycle

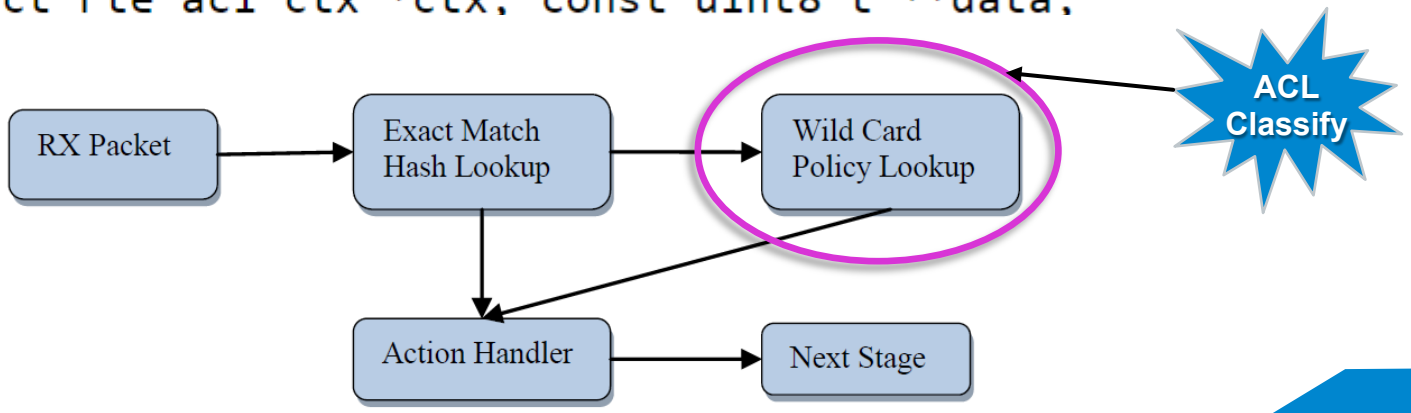
How Can Your NFV Application Benefit From SSE and AVX ?



2. Extracting More Instructions Per Cycle

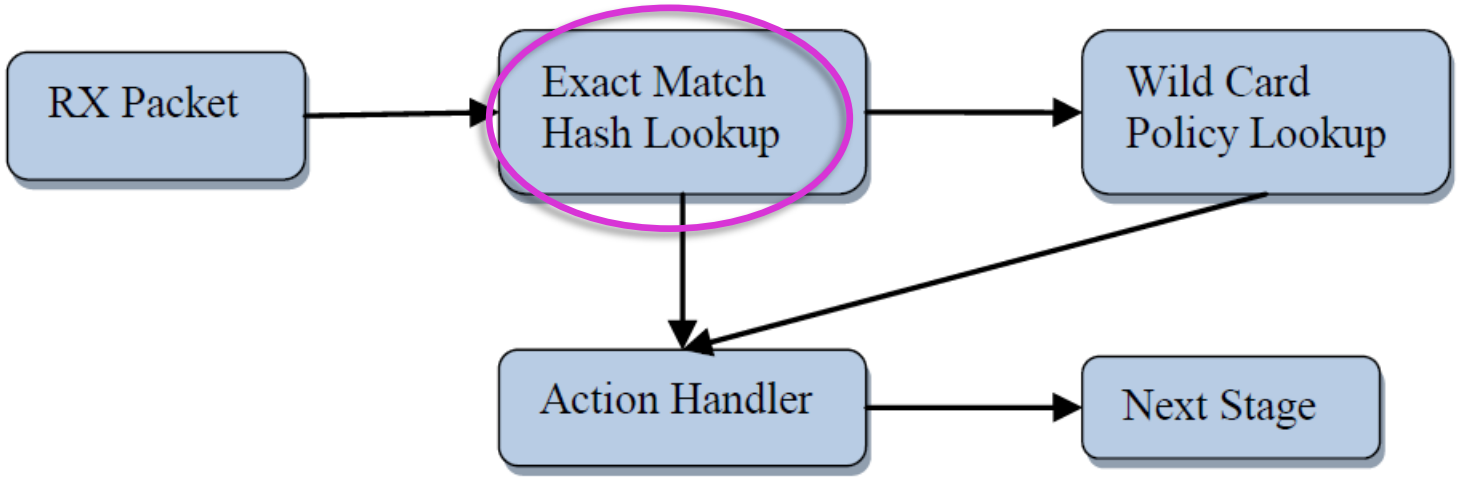
Exploiting Data Parallelism

```
218 /*
219  * Different implementations of ACL classify.
220  */
221 int
222 rte_acl_classify_scalar(const struct rte_acl_ctx *ctx, const uint8_t **data,
223         uint32_t *results, uint32_t num, uint32_t categories);
224
225 int
226 rte_acl_classify_sse(const struct rte_acl_ctx *ctx, const uint8_t **data,
227         uint32_t *results, uint32_t num, uint32_t categories);
228
229 int
230 rte_acl_classify_avx2(const struct rte_acl_ctx *ctx, const uint8_t **data,
231         uint32_t *results, ui
232
233 #ifdef __cplusplus
234 }
235 #endif /* __cplusplus */
236
237 #endif /* _ACL_H_ */
```

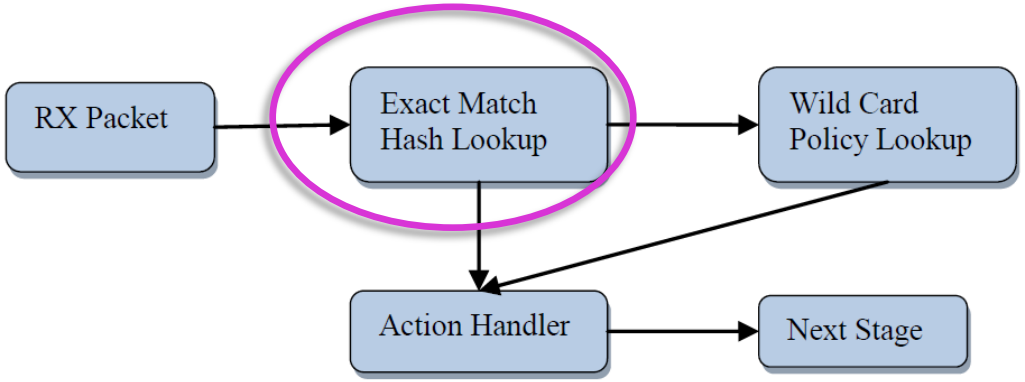


DPDK
DATA PLANE DEVELOPMENT KIT

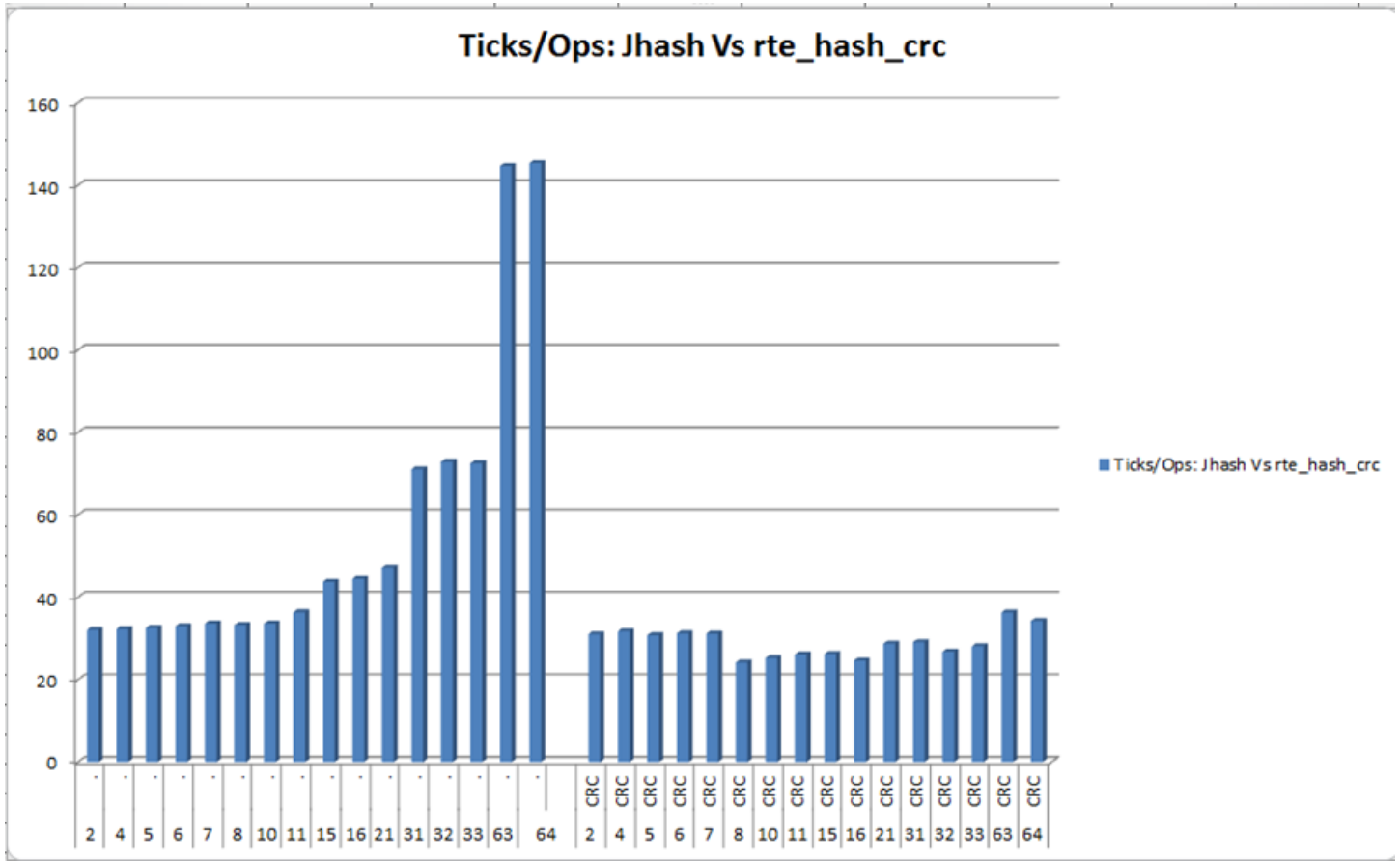
What About Exact Match Lookup Optimization?



Comparison of Different Hash Implementations



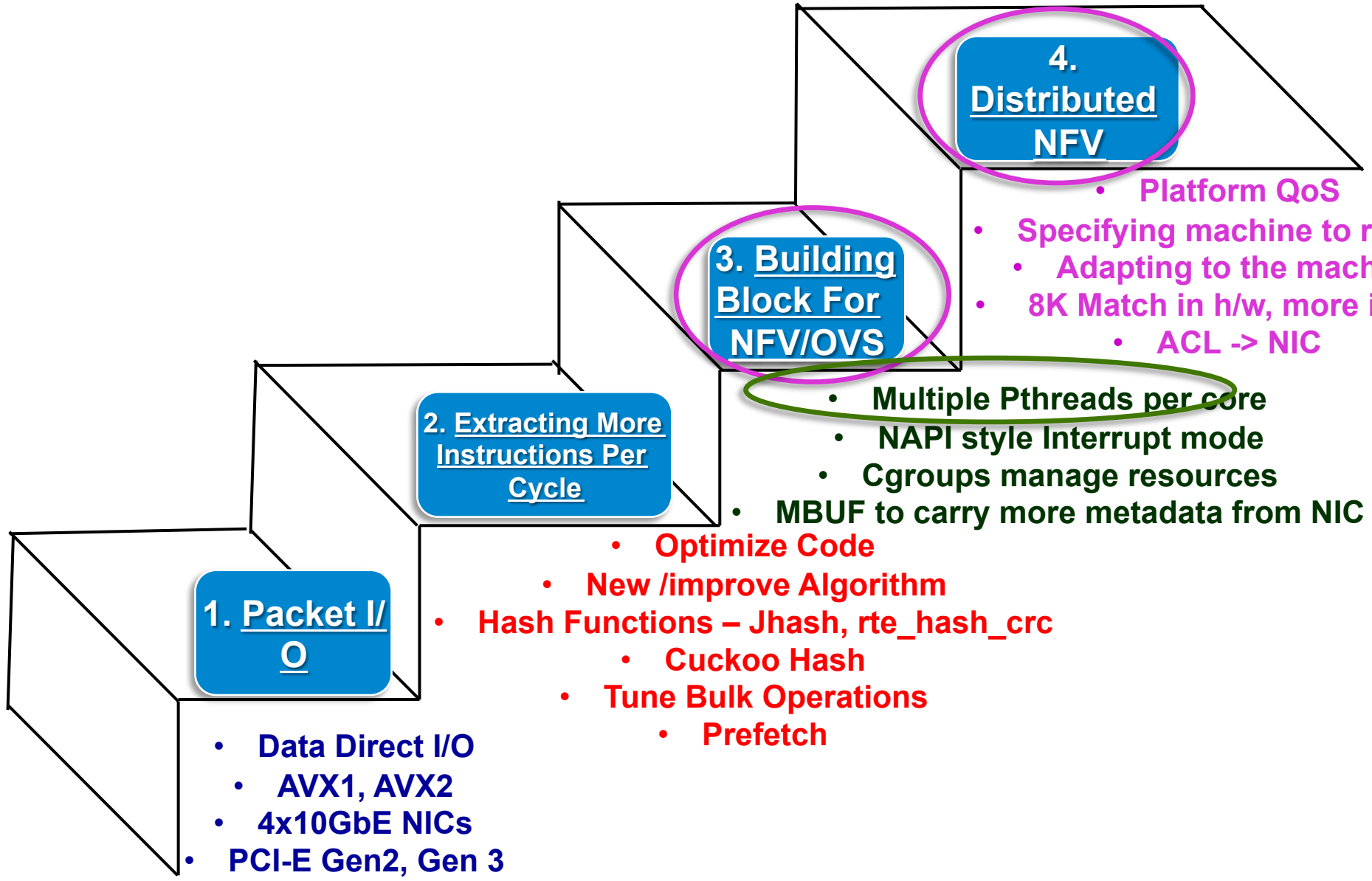
Configuration:
intel® Core™ i7 – 2 sockets
Frequency – 3 GHz
Memory: 2 Meg Huge Page –
2 Gig each socket
82599 10 Gig NIC



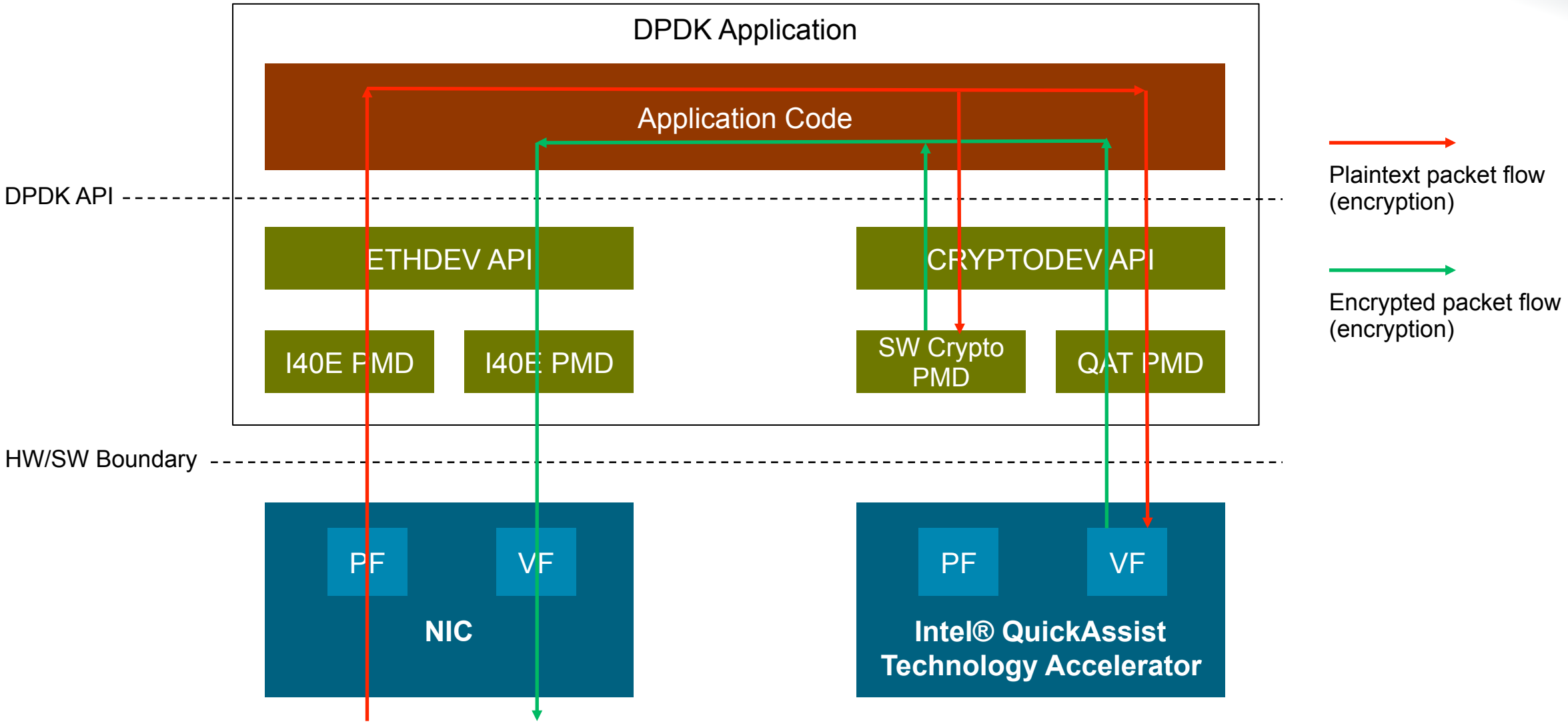
Faster Hash Functions
Higher Flow Count (16M, 32M Flows)

1 Billion Entries? Bring it on !! - DPDK & Cuckoo Switch

Trail Blazing - Performance & Functionality



Cryptodev Packet Processing Flow



4. Distributed NFV

What DPDK Features To Enhance NFV ?

- What About Specifying Which Machine (with capabilities) to Run on?
- If not available, how about adapting to the Machine where NFV was placed?
- What About …
- To Know More Register For Free in www.dpdk.org community

Summary

- DPDK offers the best performance for packet processing.
- OVS Netdev-DPDK is progressing with new features and performance enhancements.
- Ready for deployments today.

CALL TO ACTION - Thank YOU Fo
Painting The NFV World With DPD..

DPDK
DATA PLANE DEVELOPMENT KIT

1. Register in DPDK Community - http://dpdk.org/ml/listinfo/dev

   - Collaborate with Intel in Open Source and Standard Bodies.
   - DPDK, Virtual Switch, Open DayLight, Open Stack etc.

2. Develop applications with DPDK for a Programmable & Scalable VNF

Let's Collaborate and Accelerate DPDK Deployments