DPDK
Programmable Packet Processing Pipelines

M JAY

Network Platforms Group – November 2015

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
- Optimizing Cycles per Packet
- DPDK – Building Block for Programmable Packet Processing Pipeline
- Further Enhancements

- Call To Action

DPDK – Multi Architecture Support

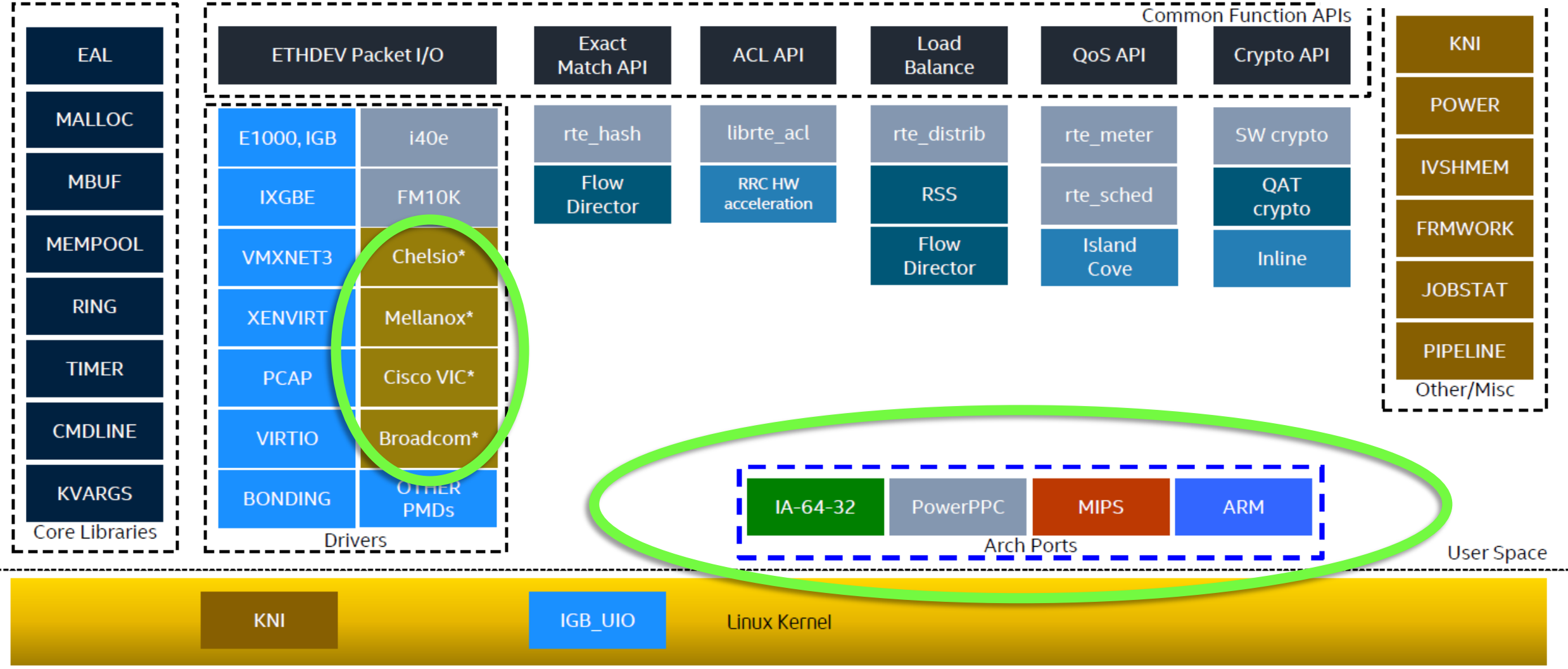
IBM Power 8

TILE-Gx

ARM v7/v8

DPDK 1.8 > DPDK 2.0 > DPDK 2.1 > DPDK 2.2

DATA PLANE DEVELOPMENT KIT ARCHITECTURE



* Other names and brands may be claimed as the property of others.

Testimonials on Throughput and Latency

If Optimizing For Throughput, How Is Latency?

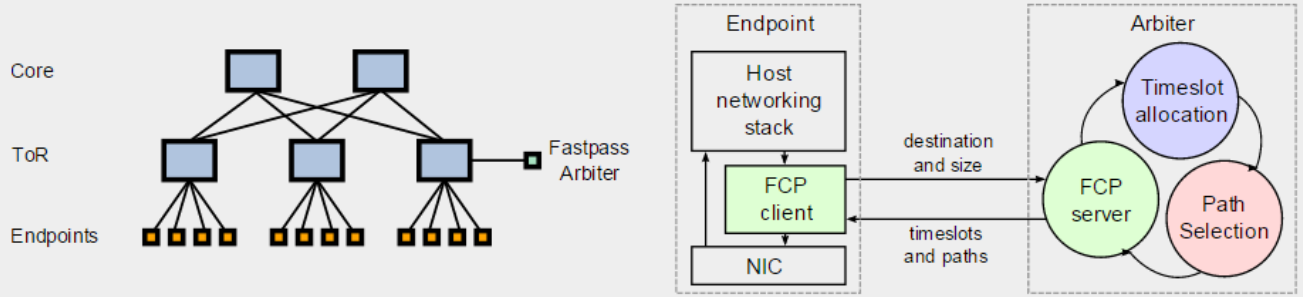
- MIT* white paper on Fast Pass
- Dream of a system with ZERO Queue
- Ultimate testimonial for Latency

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

From Test Apps To Framework – Packet Framework

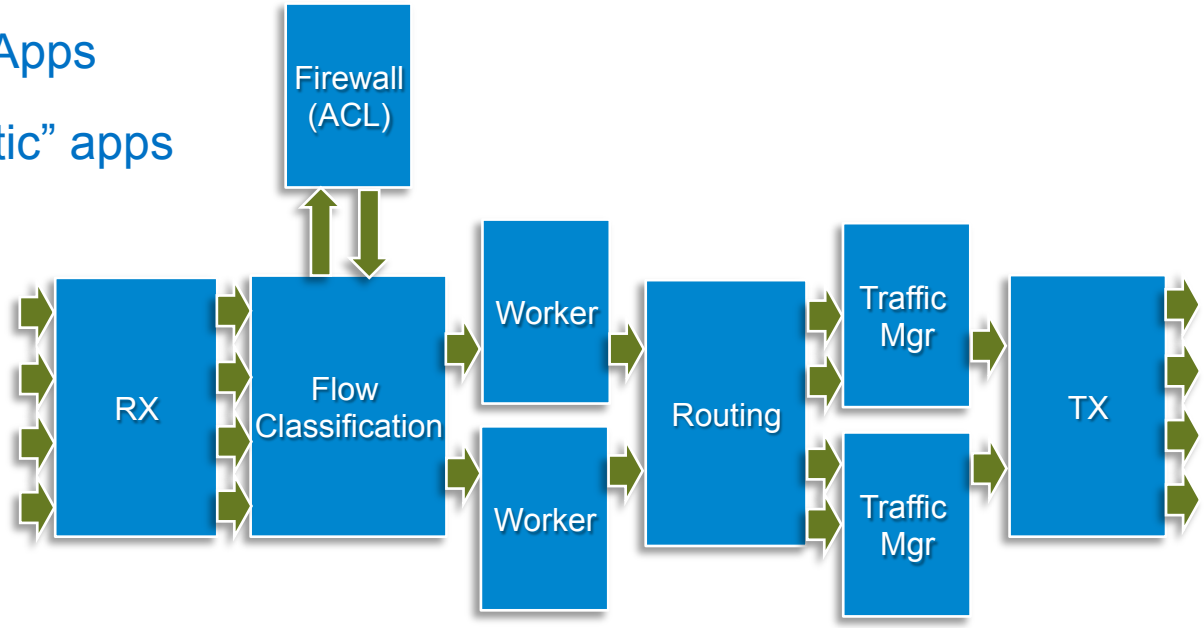
Before Packet Framework:

Simple apps only - L2fwd, L3fwd, test PMD

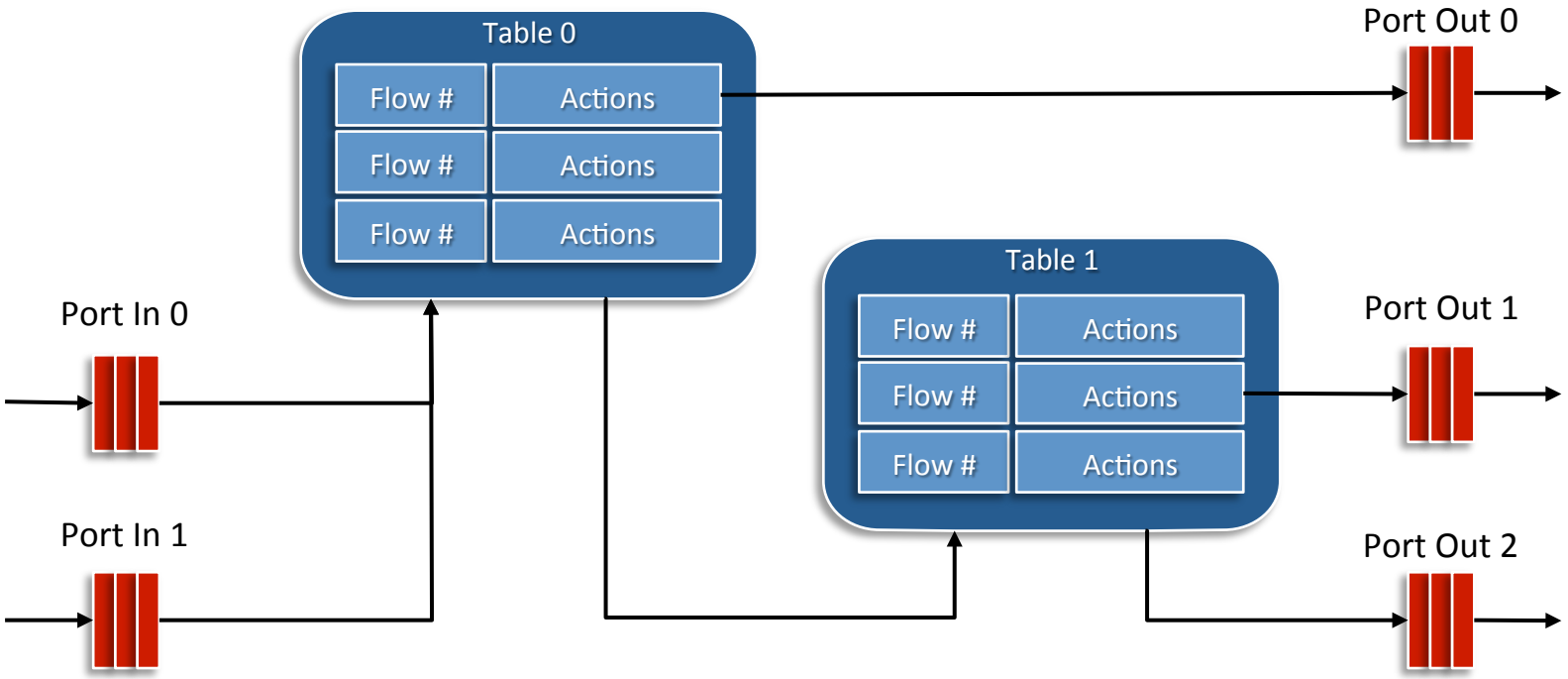
After Packet Framework:

Rapid prototyping of Complex Apps

Quick benchmarking on “realistic” apps



What is it?



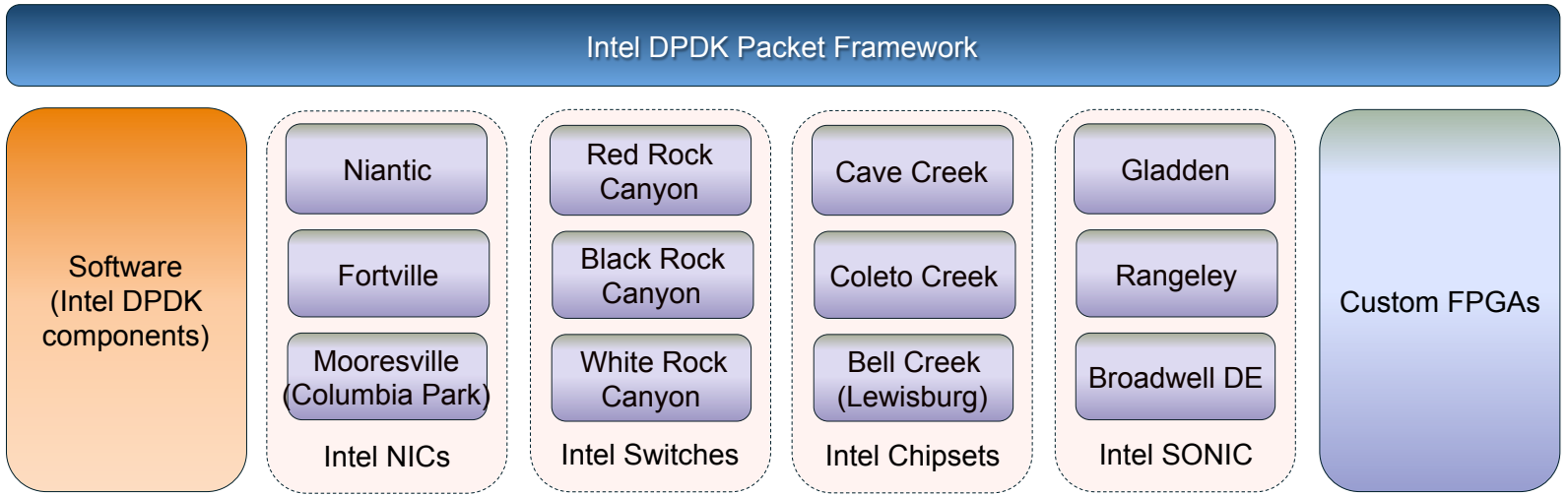
Standard methodology for ***pipeline*** development.
Ports and ***tables*** are connected together in tree-like topologies , with tables providing the ***actions*** to be executed on input packets.

Packet Frame Work Design Objectives

1. **Provide standard methodology to build complex pipelines**
   - Provide reusable and extensible templates for common functions
   - Minimize glue logic and integration effort
2. **Provide capability to switch between pure SW and HW accelerated implementations for the same function**
   - Pick (at run-time) between pure SW implem (no accel), implem on accel A, implem on accel B, etc.
   - Creates preference towards Intel silicon
3. **Provide the best trade-off between flexibility and performance**
   - Hardcoded pipelines provide the best perf, but are not flexible
   - Developing flexible frameworks is never a problem, but perf is low
4. **Provide a framework that is logically similar to Open Flow**
   - Pipeline, ports, tables, actions

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

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)

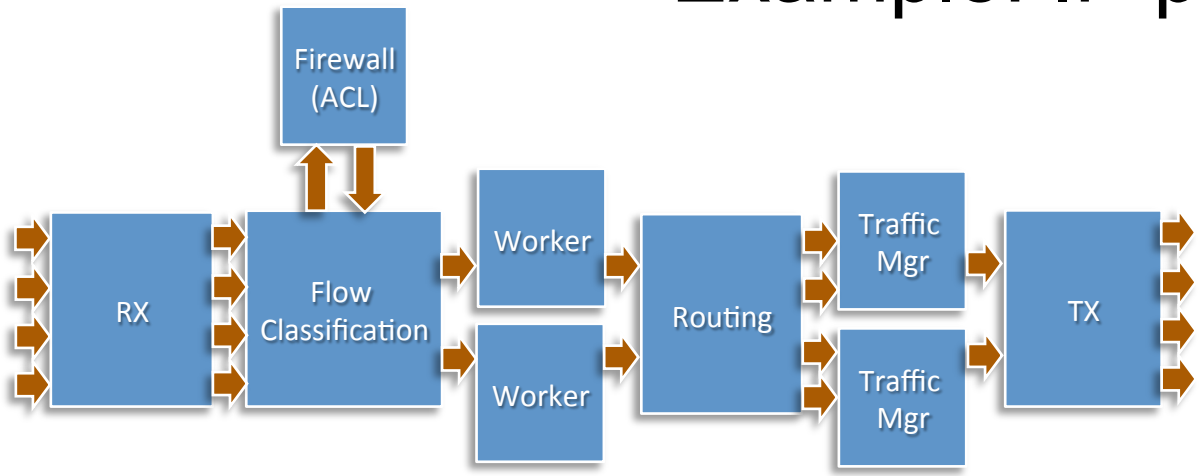
#	Table	Description
1	Hash Table	Exact match table. Used for flow classification tables, ARP caches, tunneling routing tables, etc.
2	ACL Table	Access Control Lists. Used for policy databases (firewall, etc).
3	LPM Table	Longest Prefix Match. Used for IPv4/IPv6 routing tables.
4	Pattern Matching	Pattern 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

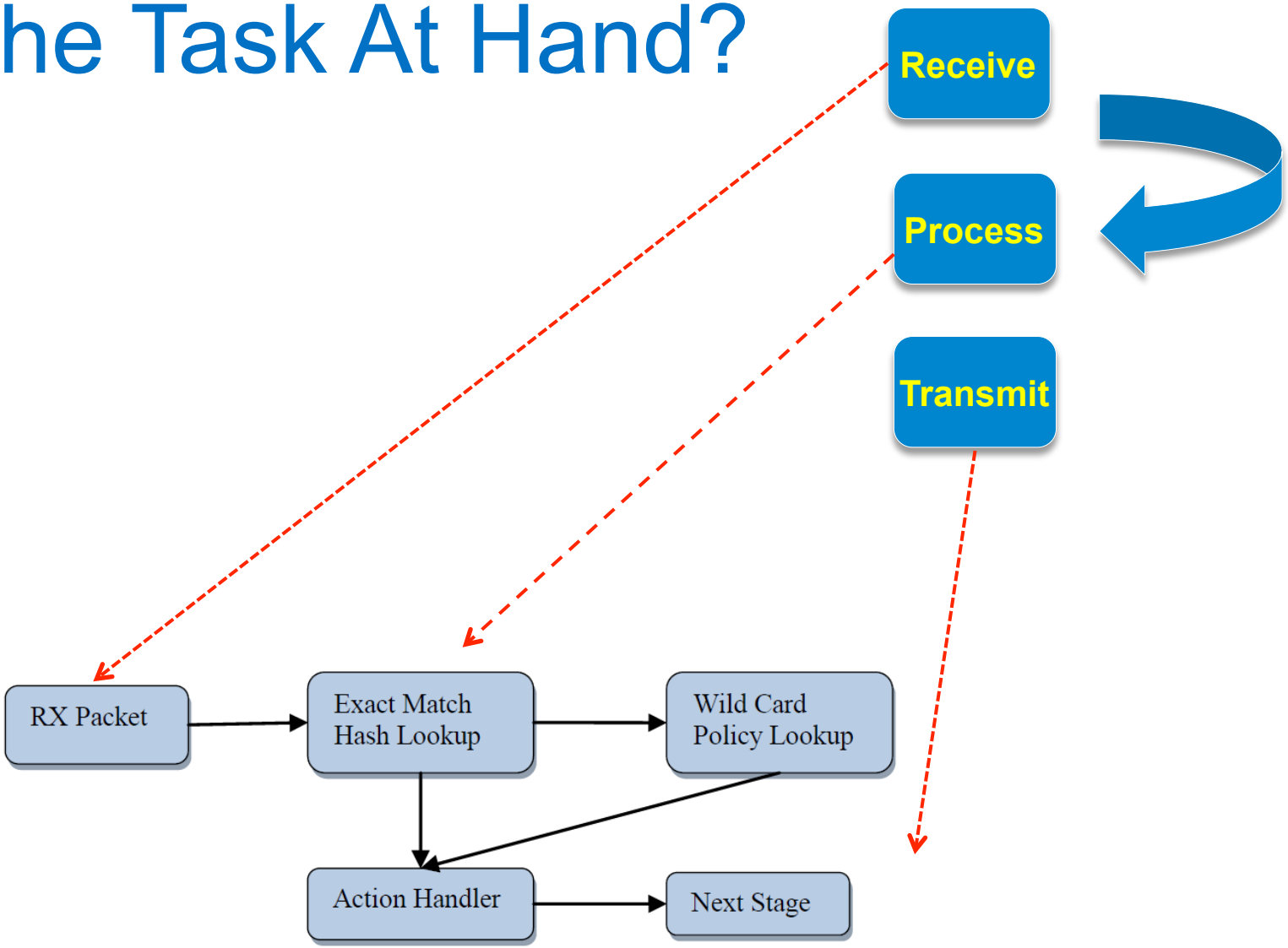
#	User Action	Description
1	Drop	Accept/deny flow
2	NAT / PAT	Translate between internal (LAN) and external IP addresses, ports (and VLAN labels)
3	Meter	Per flow traffic metering (srTCM, trTCM) and policing
4	App ID	Per flow state machine fed by sequence of packets
5	Push/pop	Push/pop VLAN or MPLS labels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

Example: IP pipeline

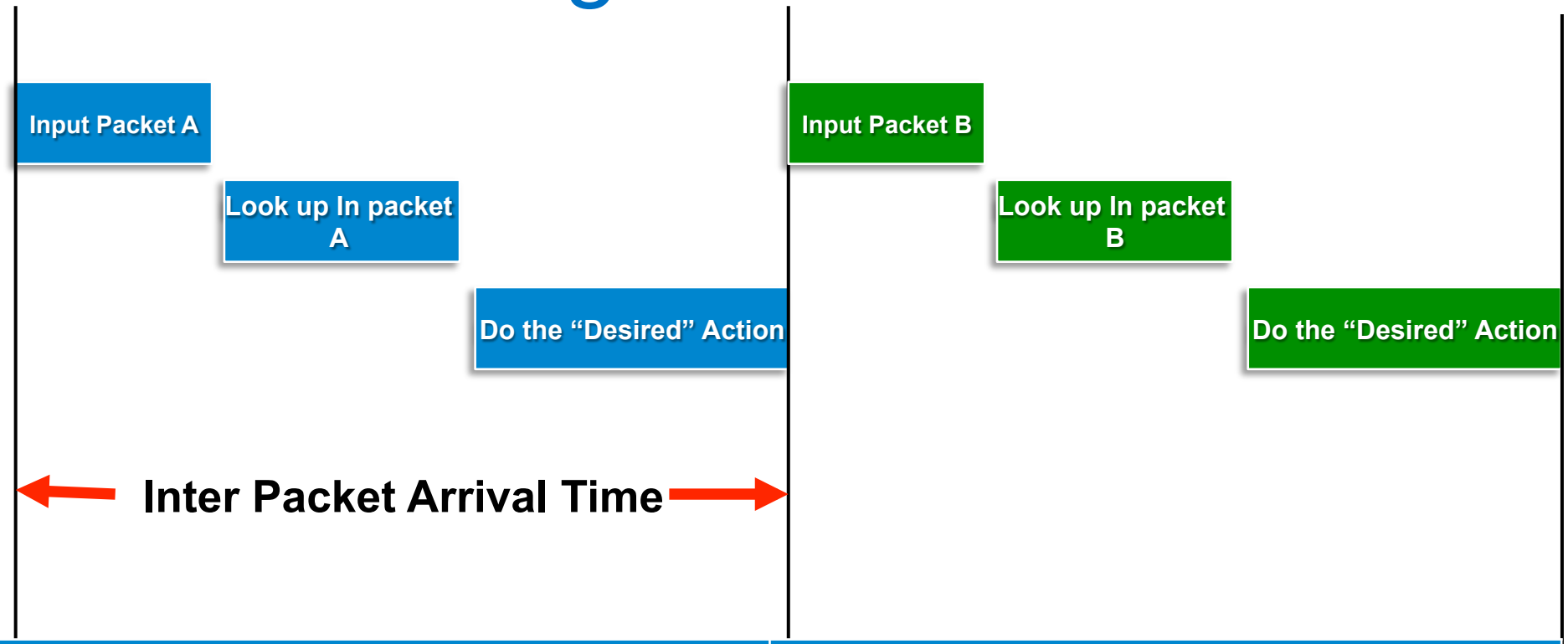


#	Pipeline	Ports IN	Tables	Ports OUT	Actions
1	RX	4x HWQ	4x Stub	4x IP RAS	
2	Flow Classification	4x IP RAS, Nx SWQ	1x Hash	3x SWQ	NAT, Meter, Stats
3	Firewall	1x SWQ	1x ACL	1x SWQ	Add flow
5	Routing & ARP	Nx SWQ	1x LPM, 1x Hash	4x IP FRAG	Dec TTL
6	Traffic Mgr In	4x IP FRAG	4x Stub	4x TM	
7	Traffic Mgr Out	4x TM	4x Stub	4x SWQ	
8	TX	4x SWQ	4x Stub	4x HWQ	

What Is The Task At Hand?



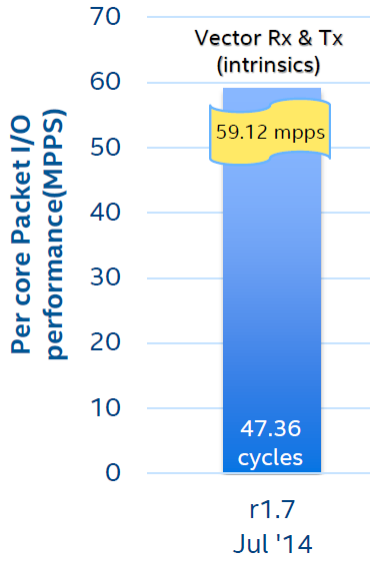
Packet Processing

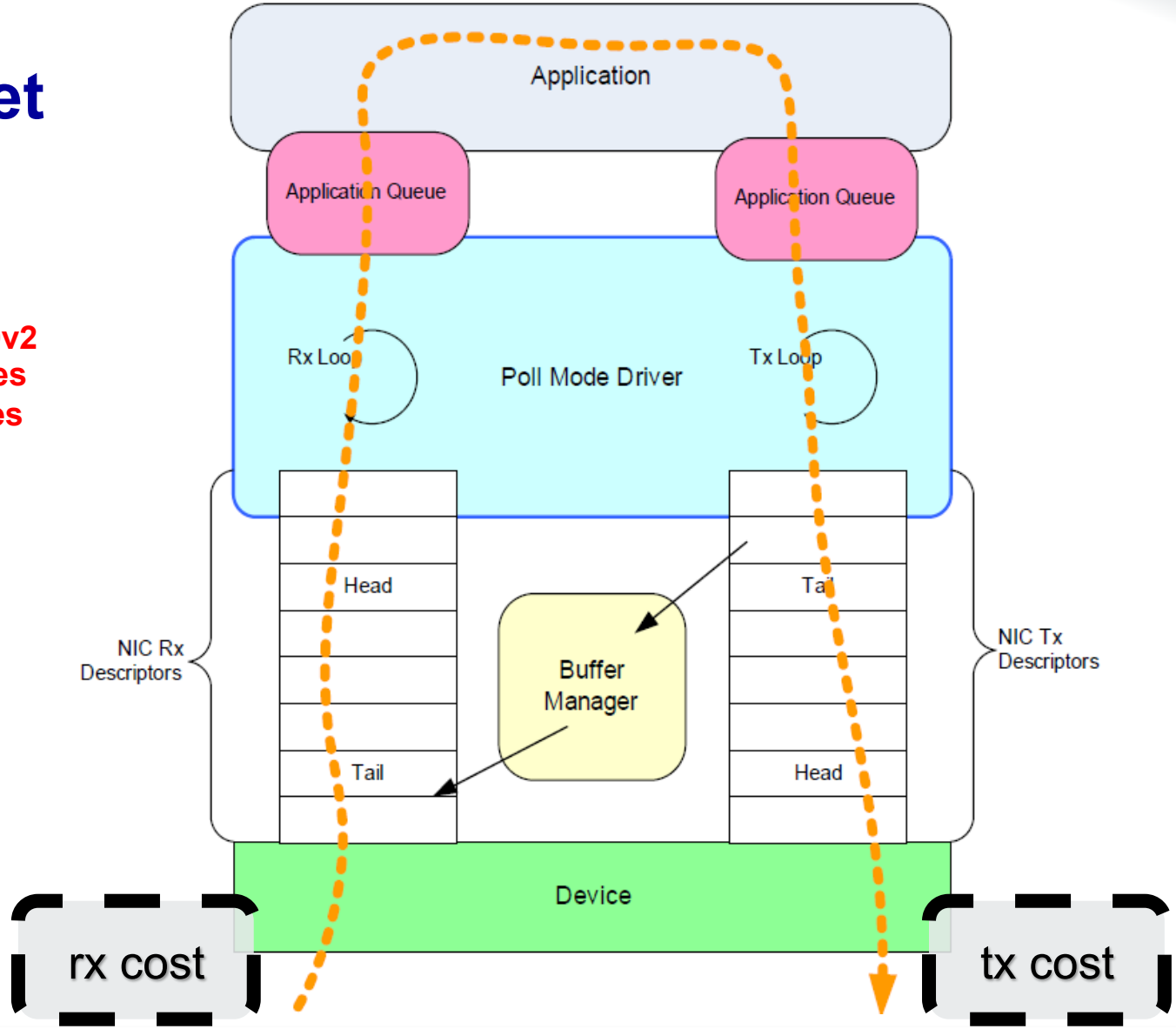


Line Rate	64 byte packet – Arrival Rate
10 GbE	67.2 ns
40 GbE	16.8 ns
100	6.7 ns

Life Of A DPDK Packet

With 2.8 GHz E5-2680v2
Rx Budget = 19 cycles
Tx Budget = 28 cycles





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

Thanks !

Back Up

A Chain is Only As Strong As ….

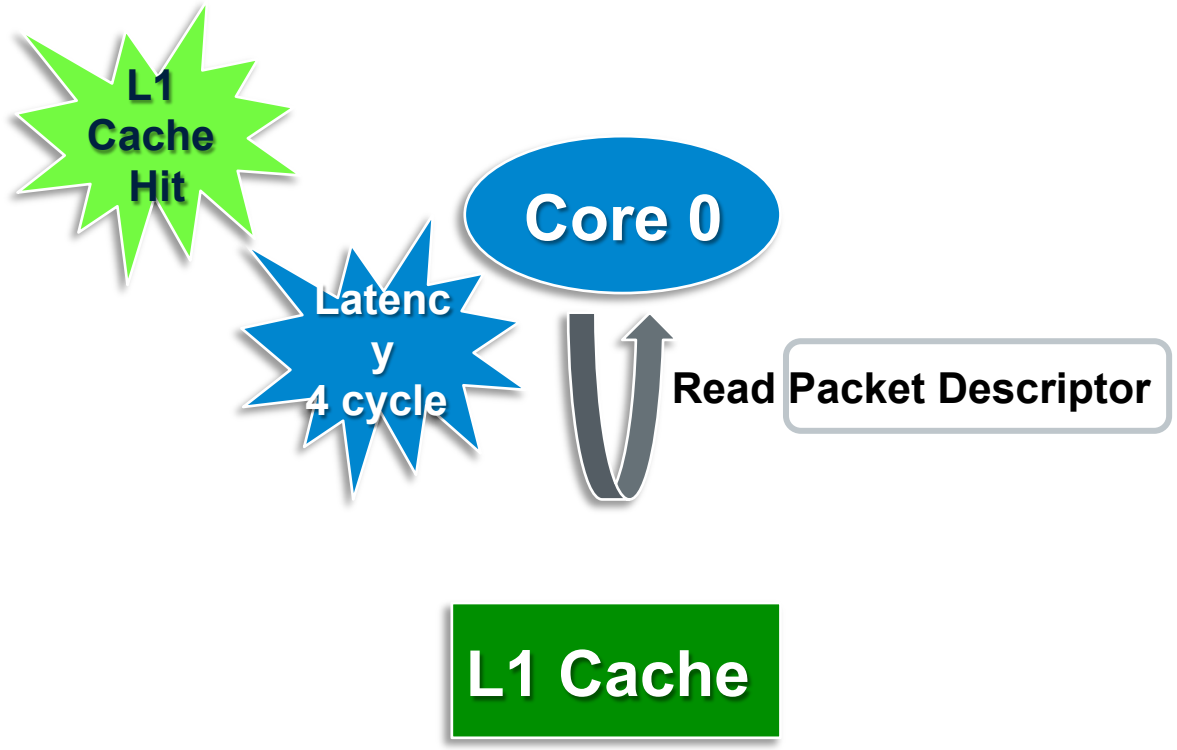
…… its Weakest Link

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

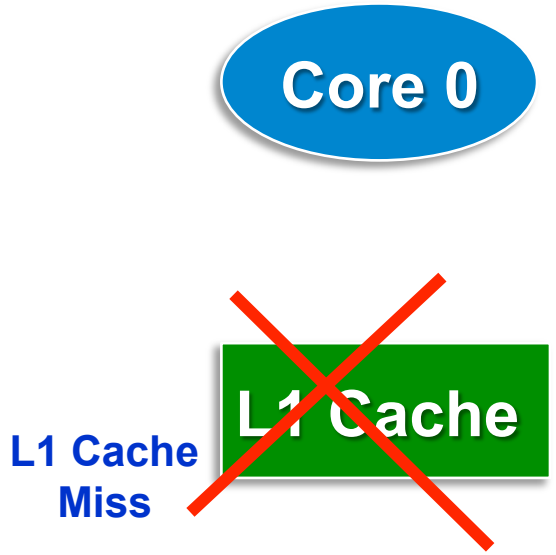
To Tackle this Challenge,
What Kind Of Devices & Latency We Have At Hand?

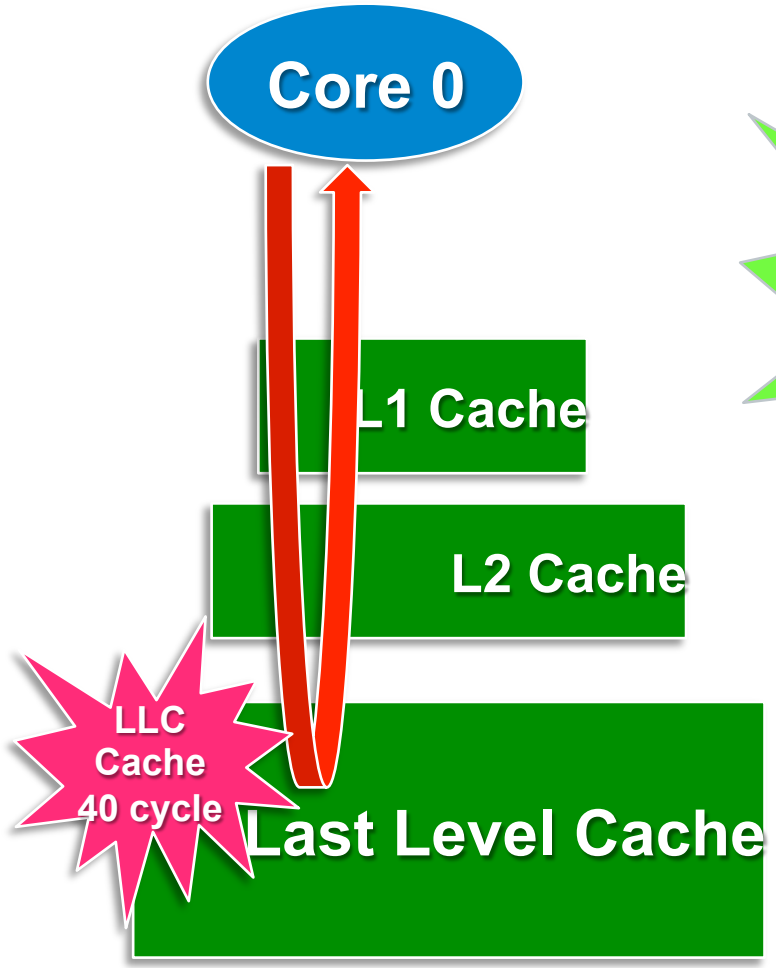
L1 Cache With 4 Cycle Latency



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

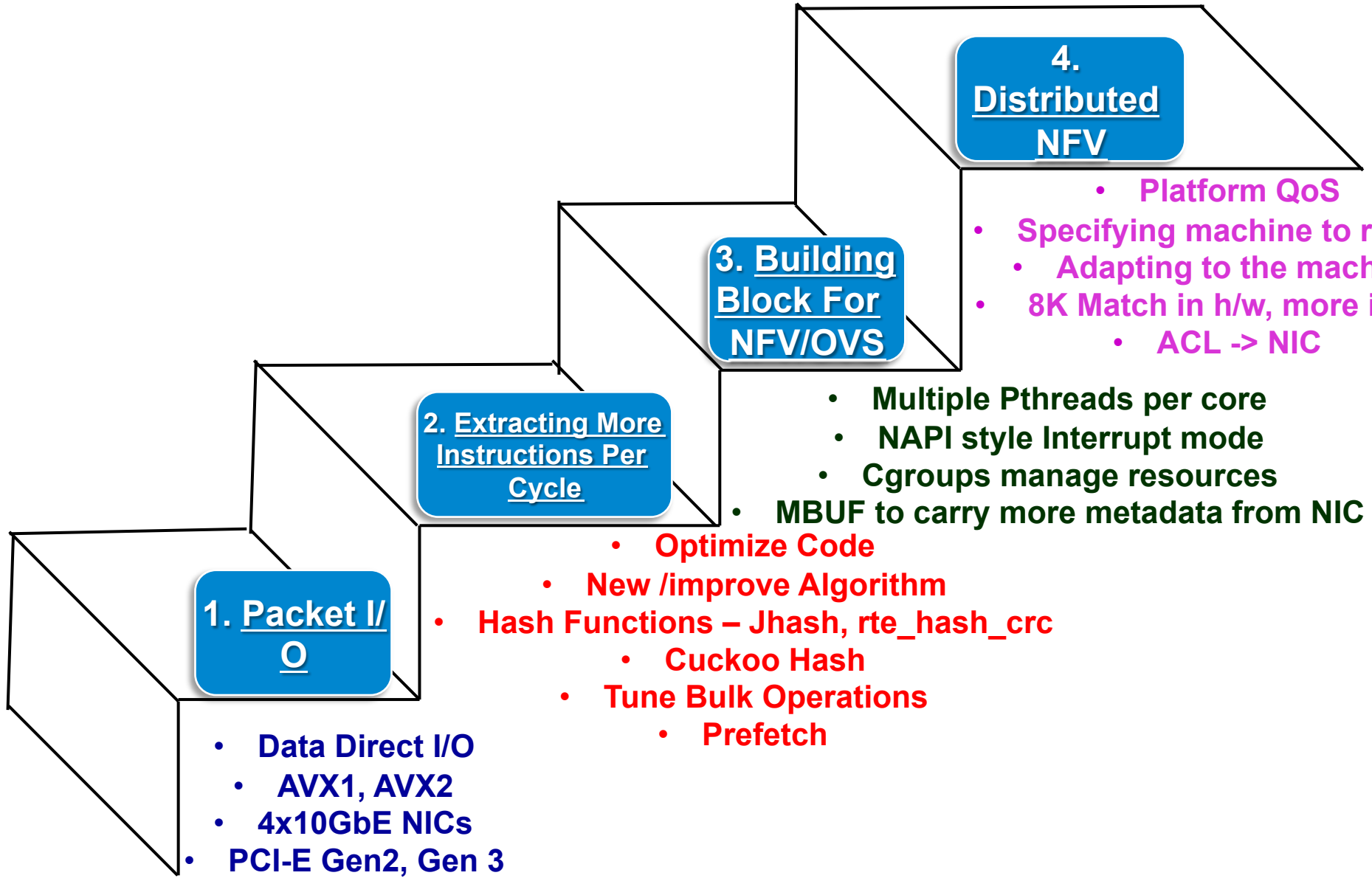
Challenge: What if there is L1 Cache Miss and LLC Hit?





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

DPDK Trail Blazing - Performance & Functionality



1. Packet I/O

Solution – Amortizing Over Multiple Descriptors

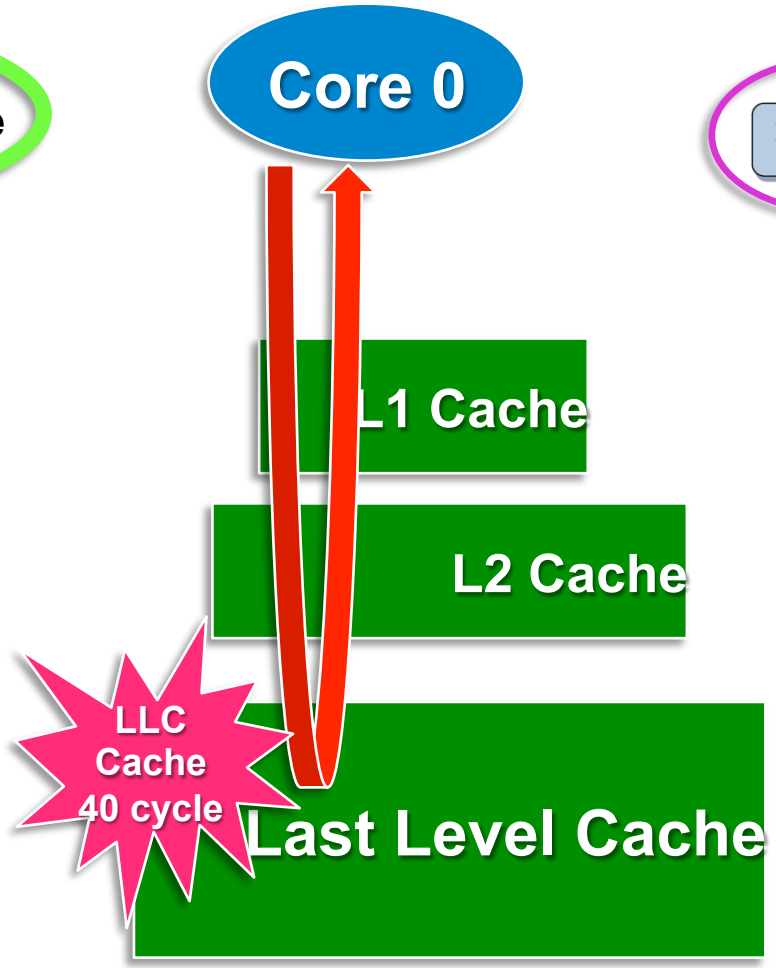
- 40 ns gets Amortized Over Multiple Descriptors
- Roughly getting back to the latency of L1 cache hit per packet
- Similarly for packet i/o, Go For Burst Read

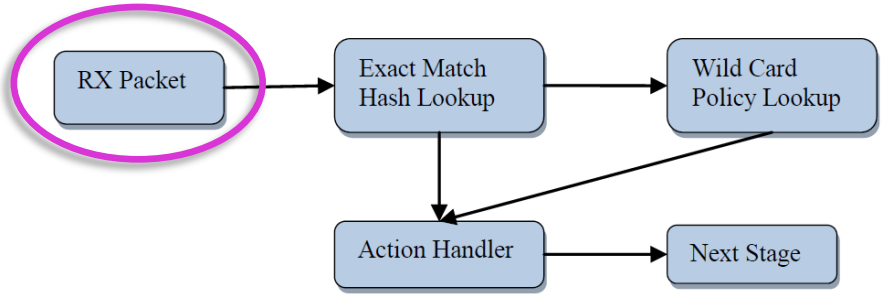
Examine Bunch Of Descriptors At A Time

DPDK DATA PLANE DEVELOPMENT KIT

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

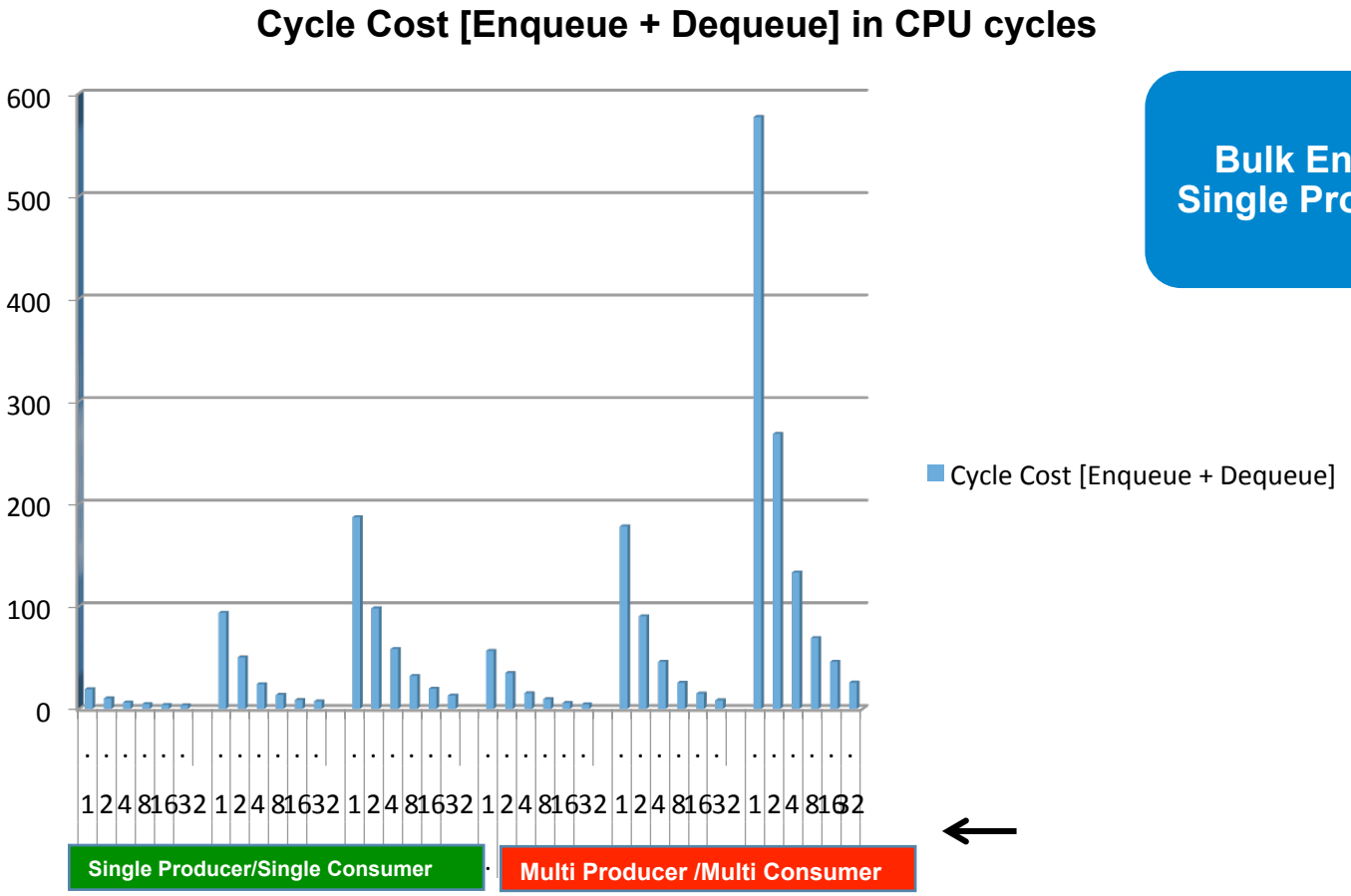
Design Principle In Packet I/O Optimization

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

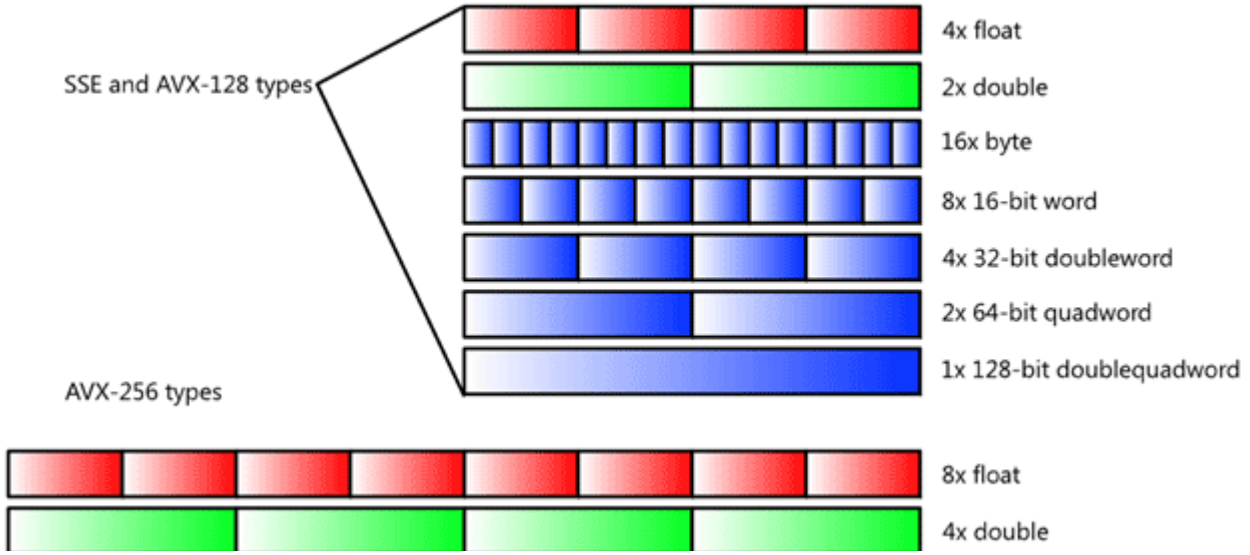


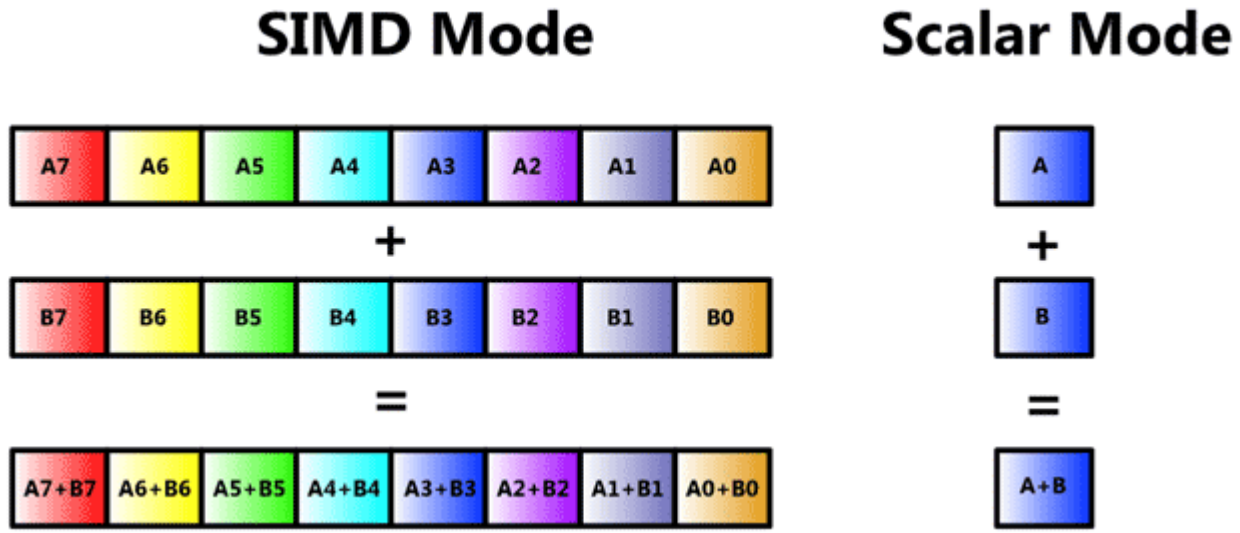
Bulk Enqueue / Bulk Dequeue
Single Producer/Single Consumer

Next: 2. Extracting More Instructions Per Cycle

2. Extracting More Instructions Per Cycle

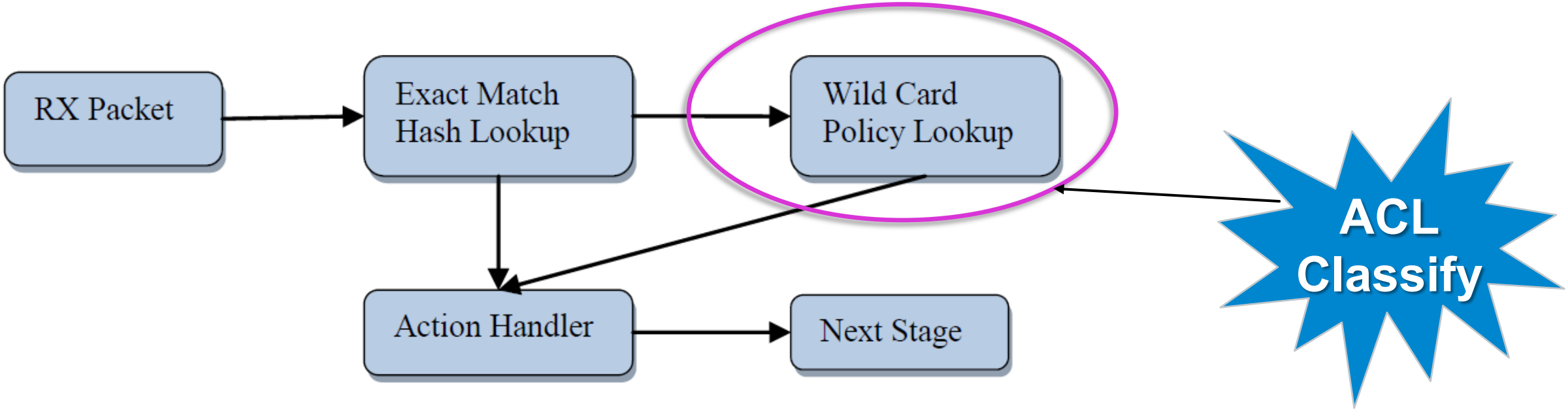
SSE – 4 Lookups in Parallel





DPDK
DATA PLANE DEVELOPMENT KIT

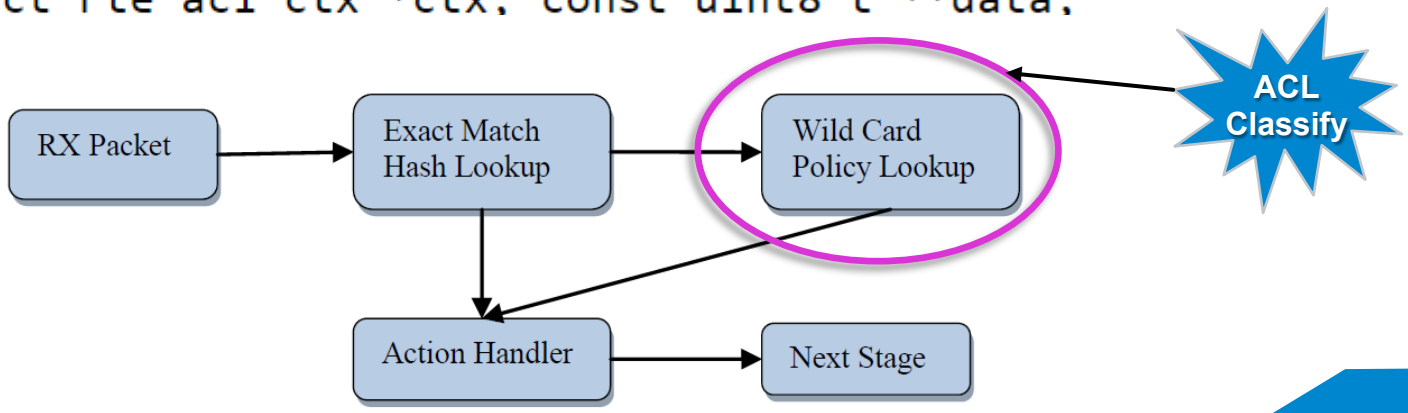
How Can Your NFV Application Benefit From SSE and AVX ?



2. Extracting More Instructions Per Cycle

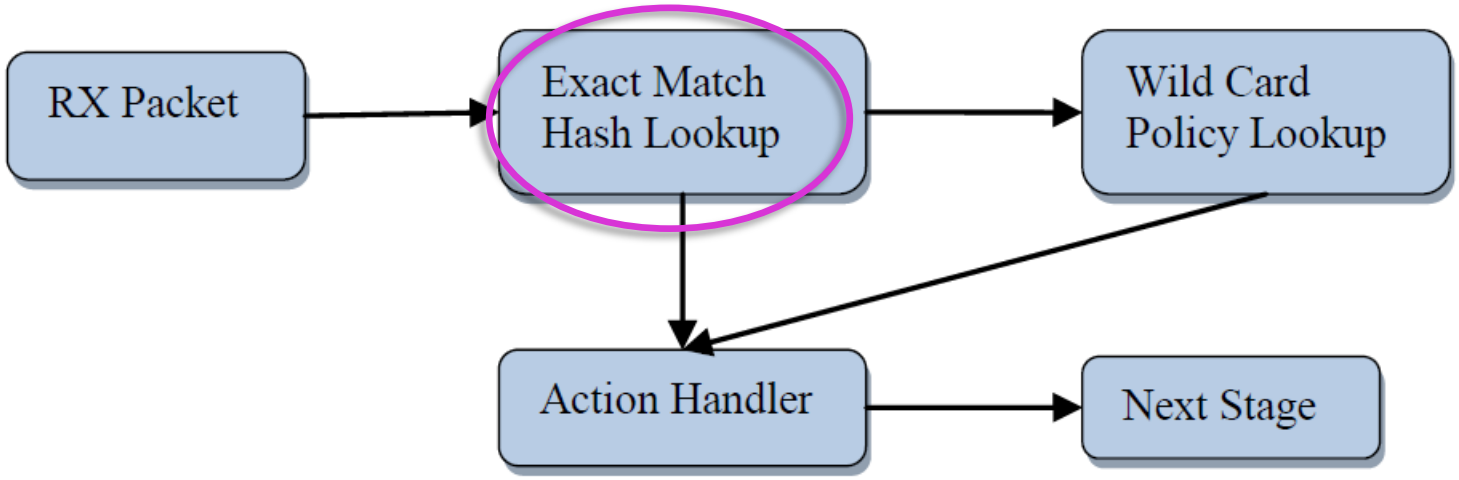
Exploiting Data Parallelism

```
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_ */
```

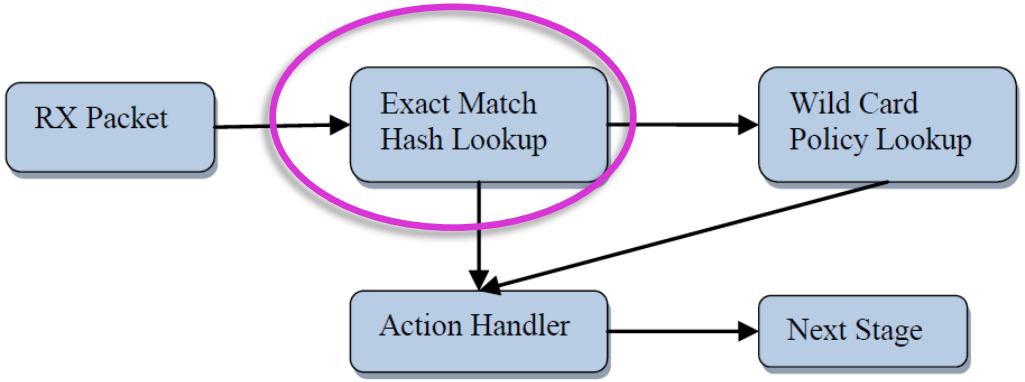


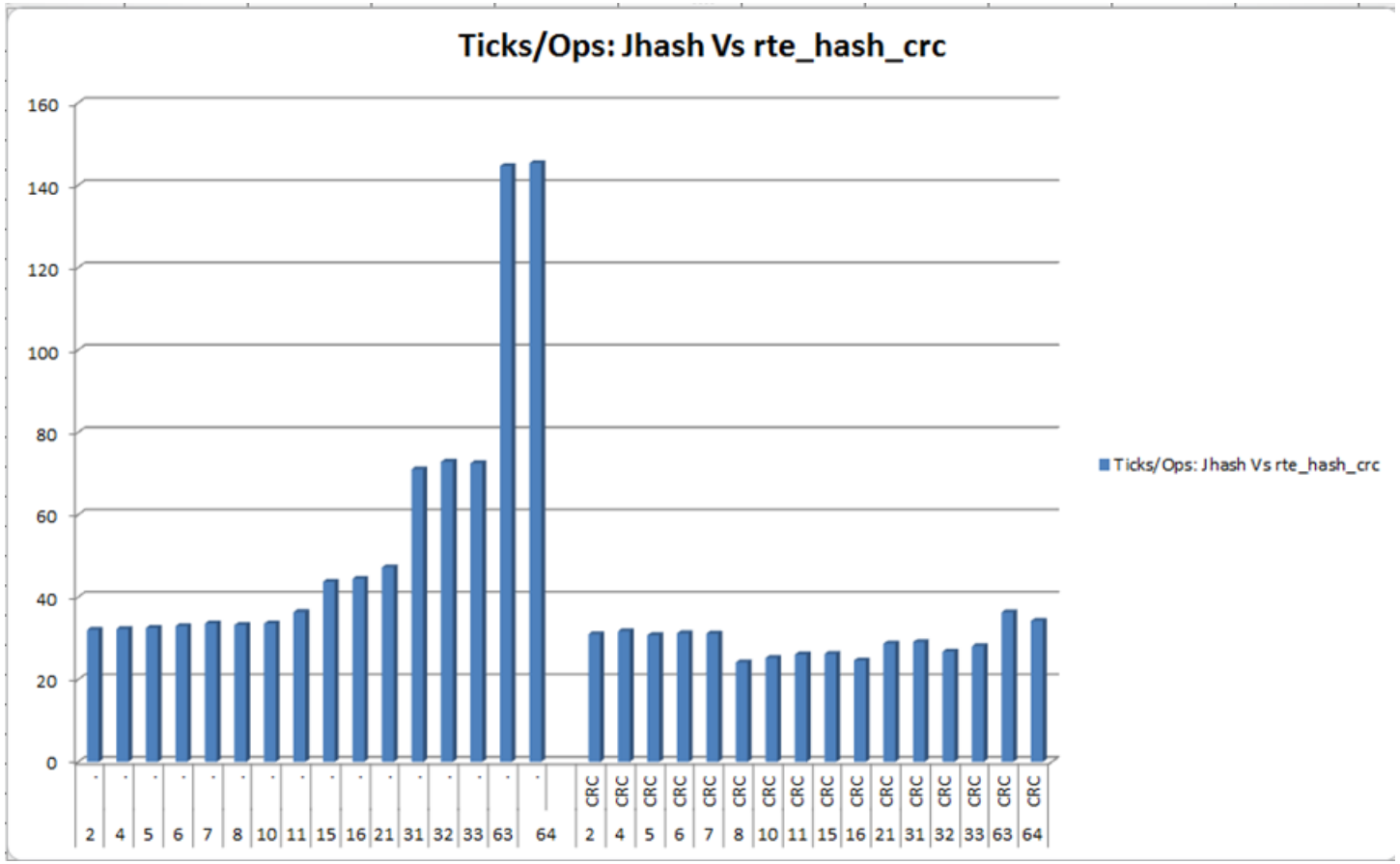
DPDK
DATA PLANE DEVELOPMENT KIT

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)

DPDK
DATA PLANE DEVELOPMENT KIT

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

http://www.cs.cmu.edu/~dongz/papers/cuckooswitch.pdf

Cuckoo Hash

Description

- New hash algorithm implemented as part of the Cuckoo Switch project
- Built around a memory-efficient, high-performance & highly-concurrent hash table

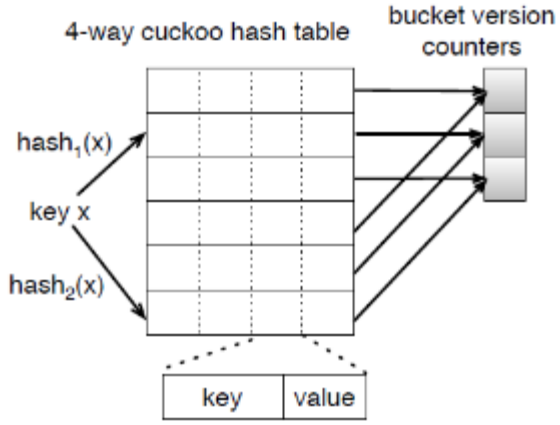
Benefits

- Compact and fast FIB lookup
- 90 Million Mpps with forwarding table of 1 billion forwarding entries
- Improve performance by using single hash table used by multiple threads/processes

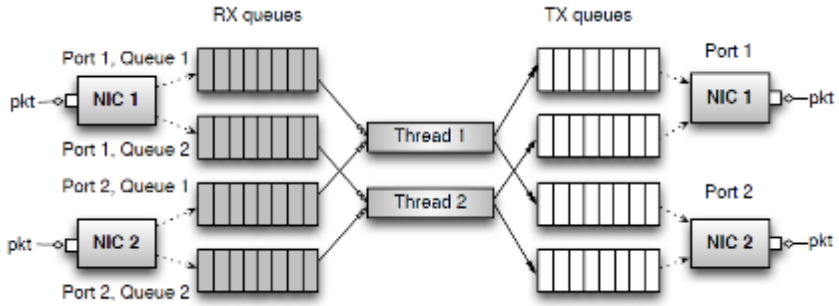
Use case

- Any packet processing application which comprise of multiple threads accessing common table.
- Complex packet processing pipelines implemented with packet Framework
- SDN and CCN (Content Centric Networking)
  - Require extremely large, fast forwarding tables
- With 40 Gbps links
  - Not just more lookups / sec, but into increasingly large tables

http://www.cs.cmu.edu/~dongz/papers/cuckooswitch.pdf

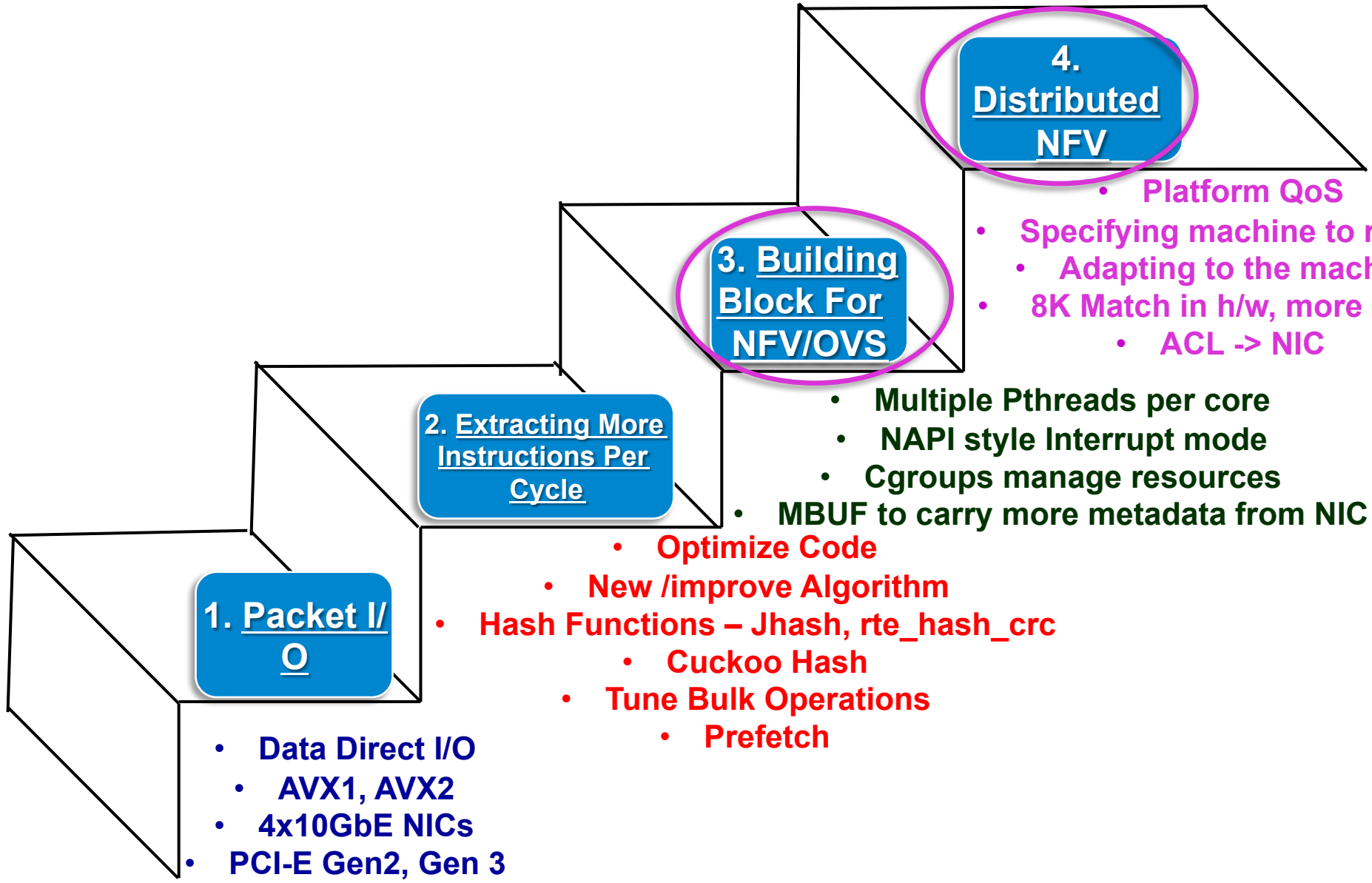


A 2,4 Cuckoo hash table



Packet Processing Pipeline of Cuckoo Switch

Trail Blazing - Performance & Functionality



DPDK DATA PLANE DEVELOPMENT KIT

Multiple pThreads per Core (cgroups)

Core Oversubscription

Description

- Allow multiple DPDK pThreads to be assigned to a logical core or float, to better use resources.
- Use cgroups (control groups) to control the resources (e.g. Pre-emptive multitasking)

Benefits

- Many operations don't require 100% of a CPU so share it smartly
  - Utilize CPU resources more efficiently instead of 1 pThread per logical core
  - Better control of application performance, lower or higher performance when needed
- *Cgroups allows Prioritization where* groups may get different shares of CPU resources

Use case

- Assign DPDK pThreads to float for infrequent tasks such as management applications
- Assign multiple pThreads to an lcore and determine the utilization each requires. Utilization may change over the period of time and so may the pThreads requirements of the core.

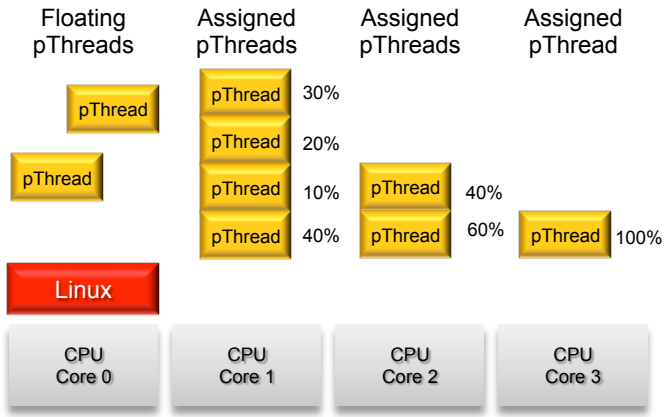


Figure 1: pThread affinity model

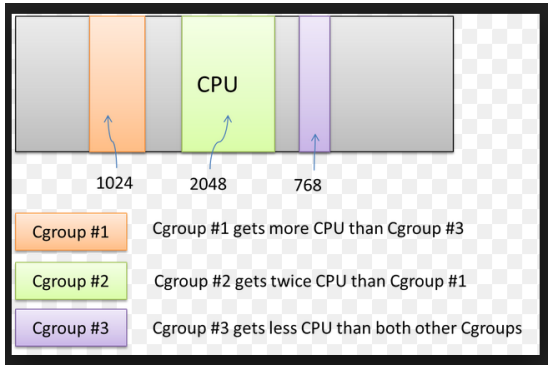


Figure 2: cgroup – Pre-emptive multitasking

http://eecatalog.com/embeddedlinux/2012/06/28/improve-linux-real-time-performance-in-multicore-devices-with-light-weight-threading/

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

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

Thank YOU For Painting The NFV With DPDK

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

3. Evaluate Intel Open Network Platform for best-in-class NFVi
   - Download from 01.org & evaluate
   - Become familiar with data plane benchmarks on Intel® Xeon platforms

Let's Collaborate and Accelerate NFV Deployments