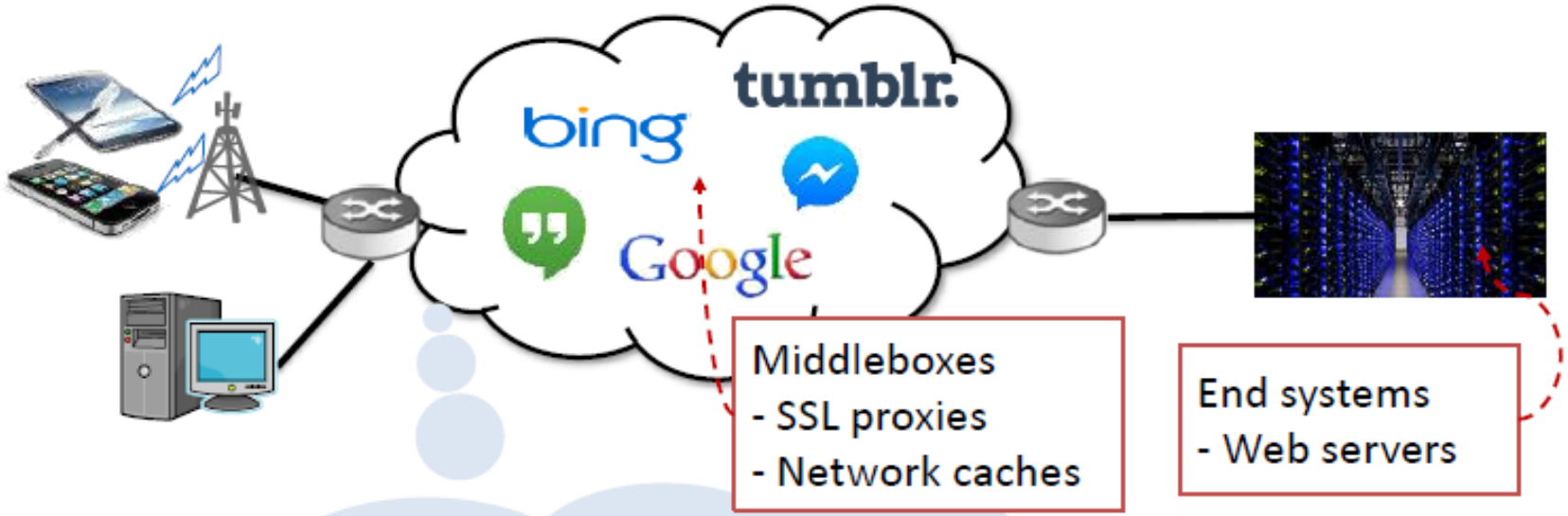
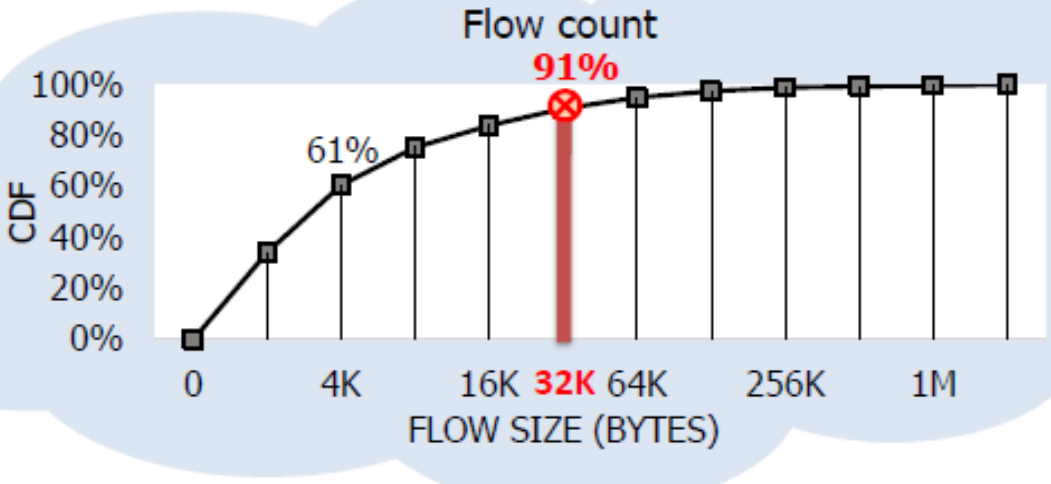
Data Plane Development Kit

One Convergence Devices Ptv. Ltd

Saif

Needs for Handling Many Short Flows

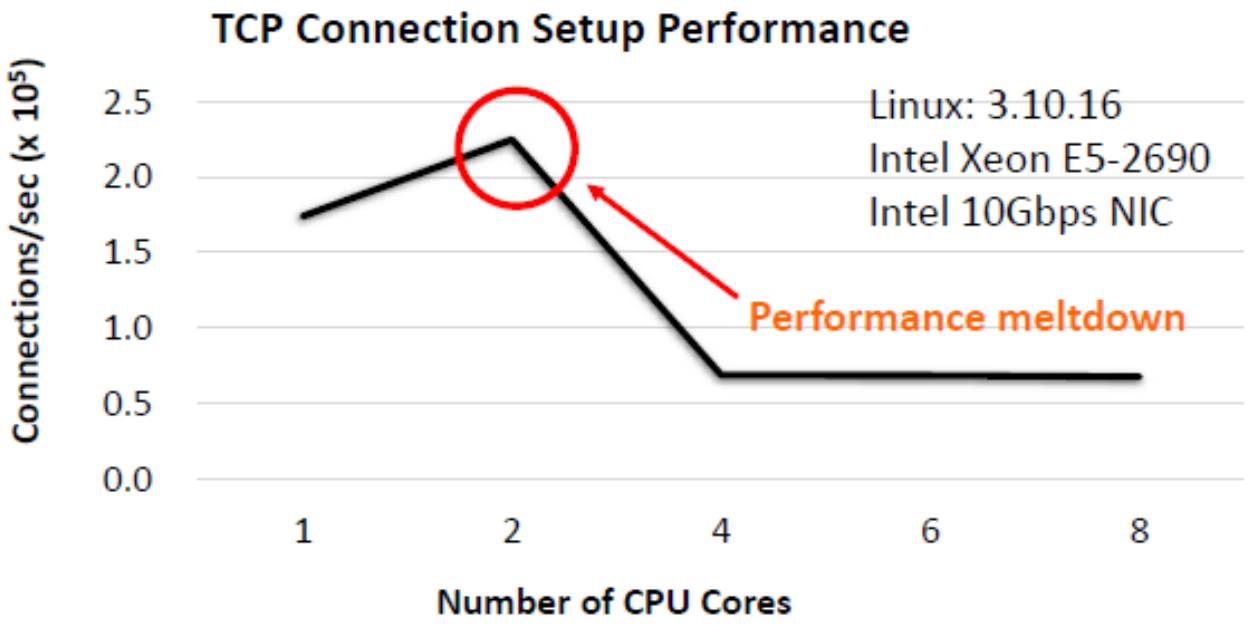




* **Commercial cellular traffic for 7 days**
Comparison of Caching Strategies in Modern Cellular Backhaul Networks, MOBISYS 2013

Unsatisfactory Performance of Linux TCP

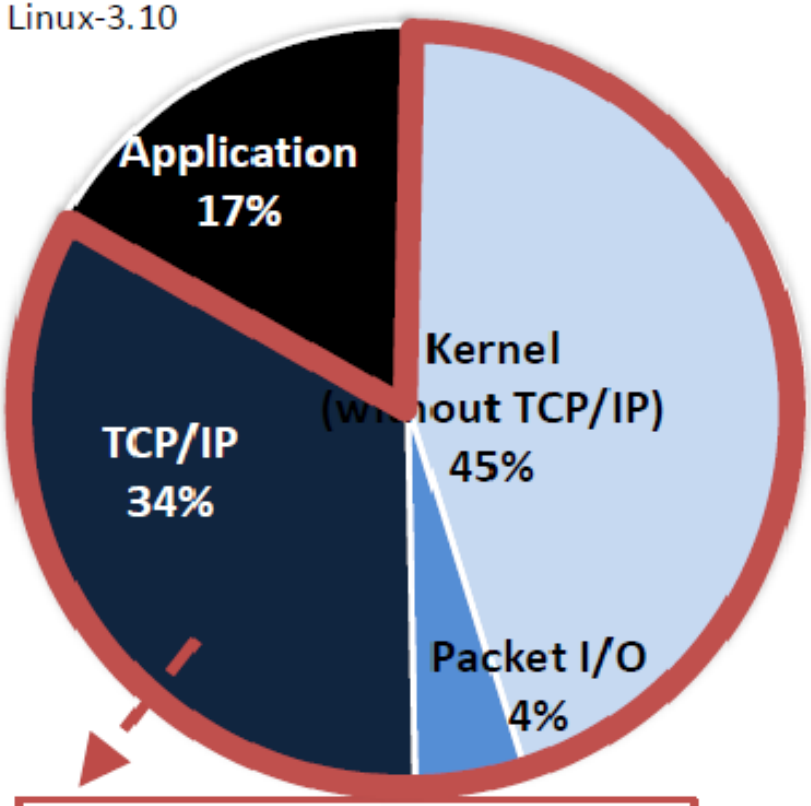
- Large flows: Easy to fill up 10 Gbps
- Small flows: Hard to fill up 10 Gbps regardless of # cores
  - Too many packets:
    14.88 Mpps for 64B packets in a 10 Gbps link
  - Kernel is not designed well for multicore systems



Kernel Uses the Most CPU Cycles

CPU Usage Breakdown of Web Server

Web server (Lighttpd) Serving a 64 byte file
Linux-3.10



83% of CPU usage spent inside kernel!

Performance bottlenecks

1. Shared resources
2. Broken locality
3. Per packet processing

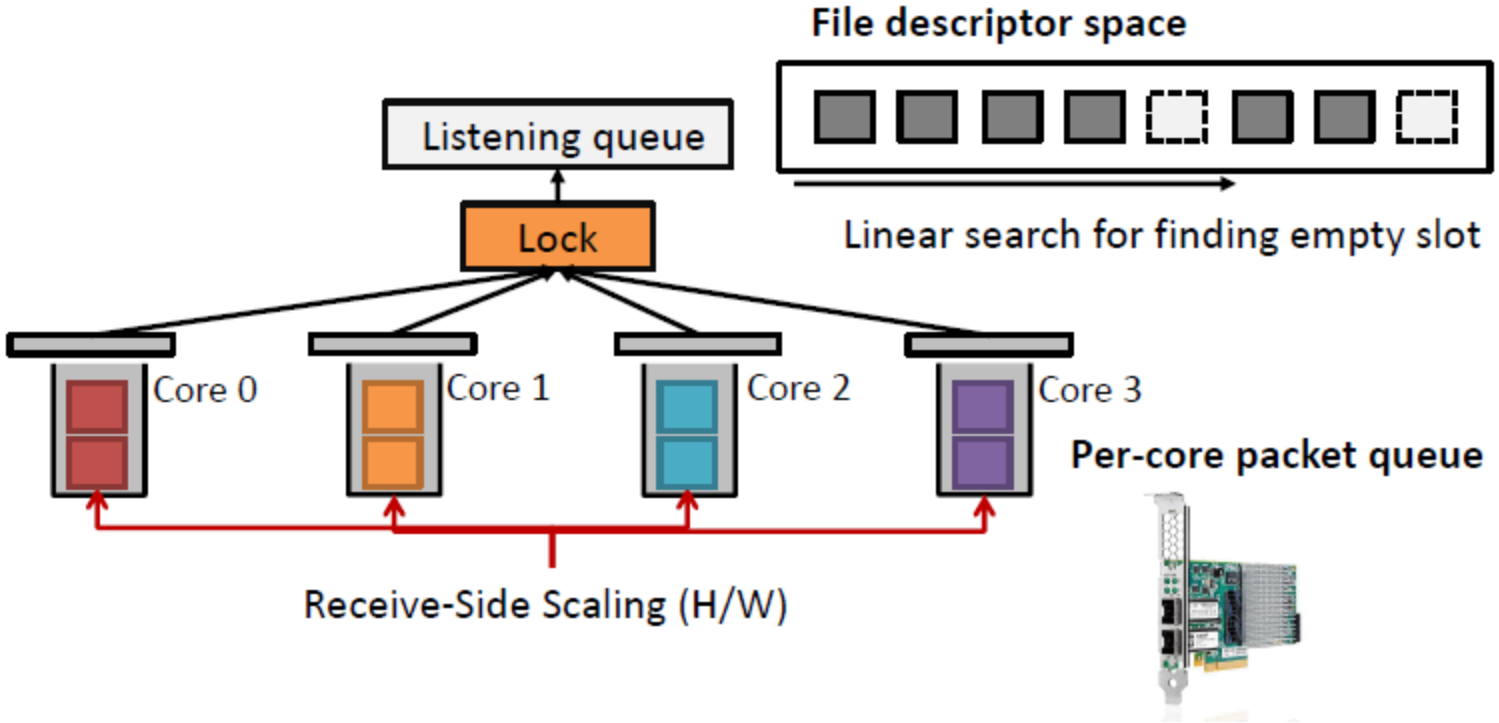
Bottleneck removed by **DPDK**

1) **Efficient use of CPU cycles for TCP/IP processing**
   → **2.35x more CPU cycles for app**
2) **3x ~ 25x better performance**

A chain is as strong as its **weakest** link

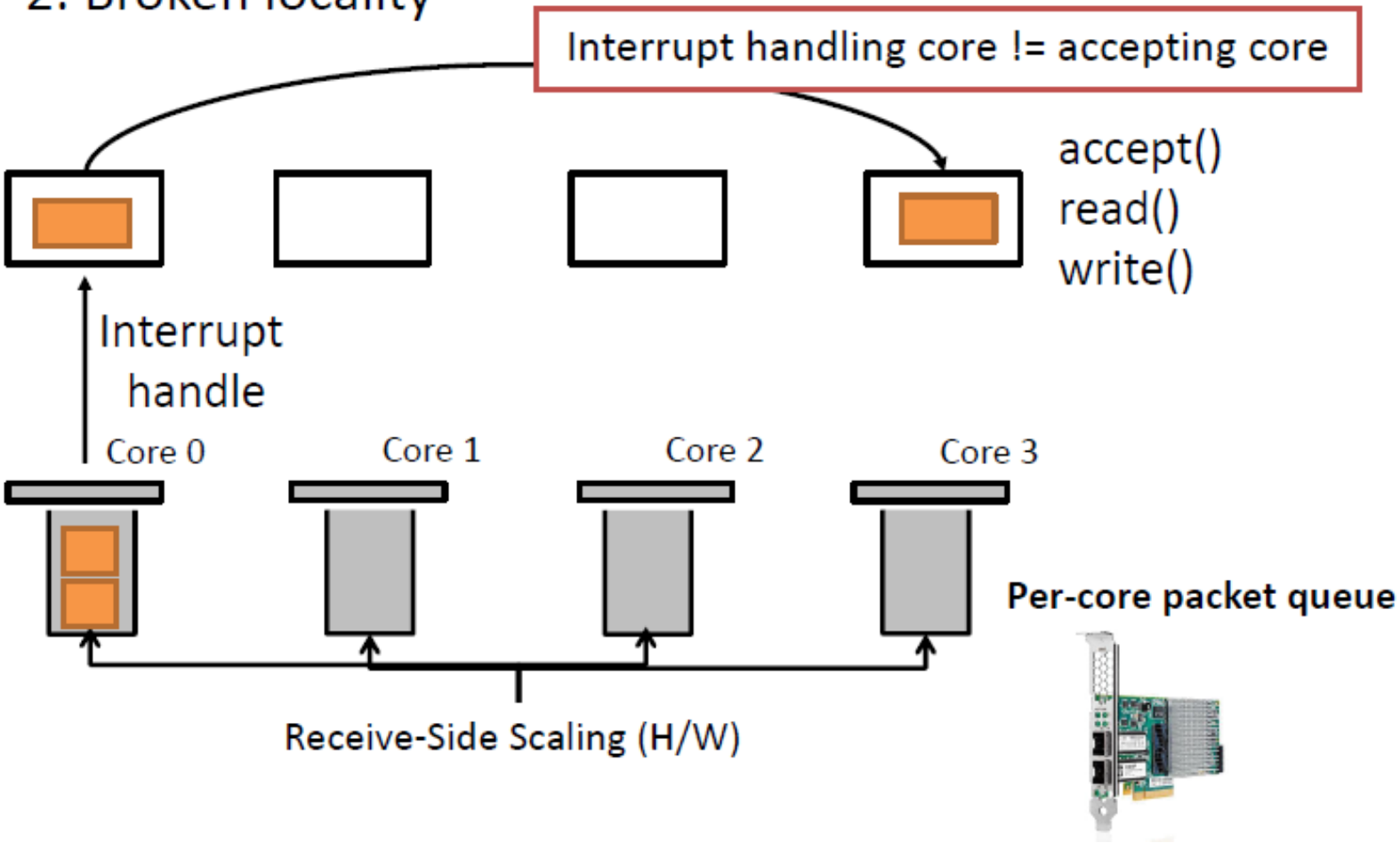
Inefficiencies in Kernel from Shared FD

1. Shared resources
   - Shared listening queue
   - Shared file descriptor space



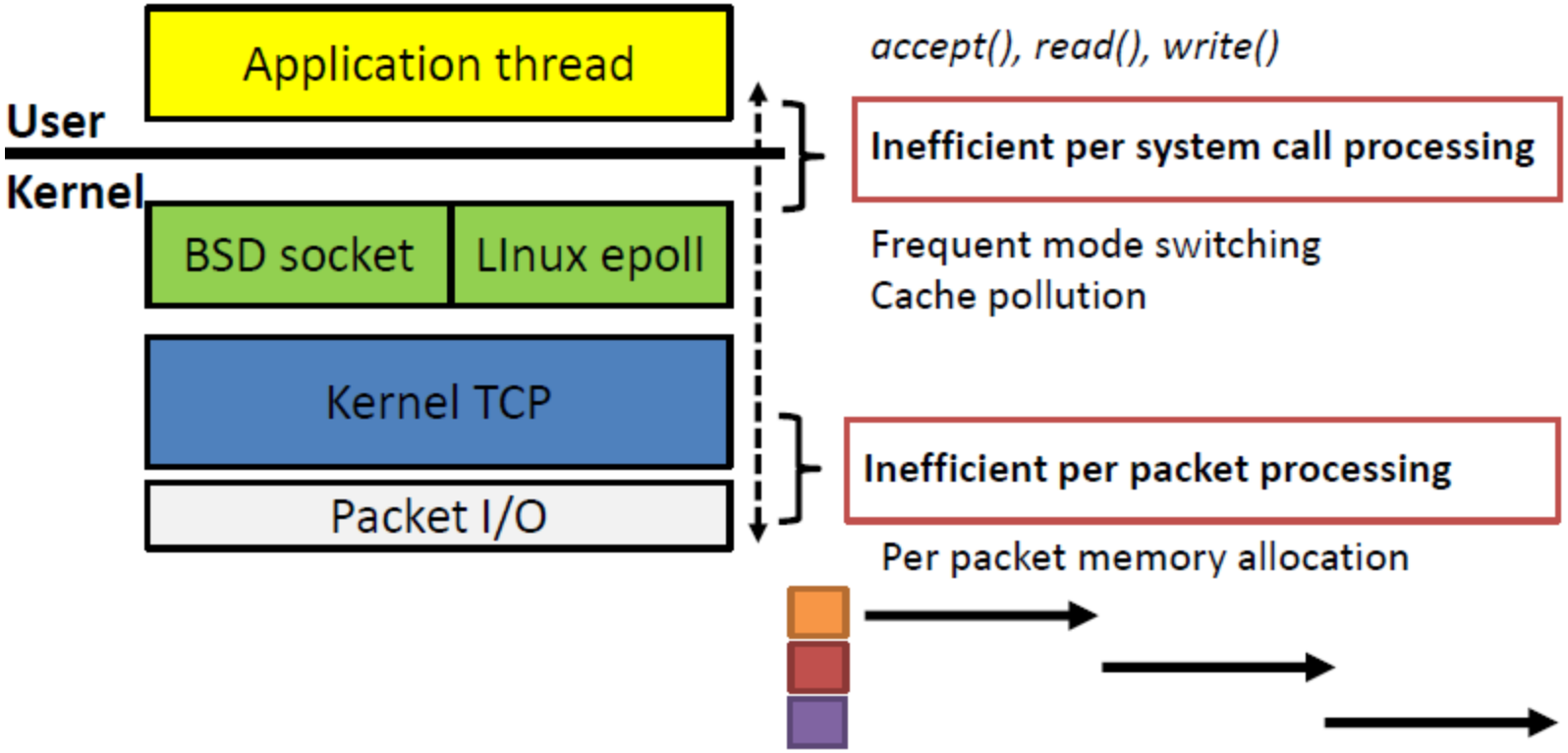
Inefficiencies in Kernel from Broken Locality

2. Broken locality



Inefficiencies in Kernel from Lack of Support for Batching

3. Per packet, per system call processing



Previous Works on Solving Kernel Complexity

	Listening queue	Connection locality	App <-> TCP comm.	Packet I/O	API
Linux-2.6	Shared	No	Per system call	Per packet	BSD
Linux-3.9 SO_REUSEPORT	Per-core	No	Per system call	Per packet	BSD
Affinity-Accept	Per-core	Yes	Per system call	Per packet	BSD
MegaPipe	Per-core	Yes	Batched system call	Per packet	custom

→ Still, 78% of CPU cycles are used in kernel!

How much **performance improvement** can we get
if we implement **a user-level TCP stack** with all optimizations?

Clean-slate Design Principles of **DPDK**

- DPDK : A high-performance user-level TCP designed for multicore systems
- Clean-slate approach to divorce kernel's complexity

Problems		**Our contributions**
1. Shared resources	→	Each core works independently
2. Broken locality		– No shared resources
		– Resources affinity
3. Lack of support for batching	→	Batching from flow processing from packet I/O to user API
	+	Easily portable APIs for compatibility

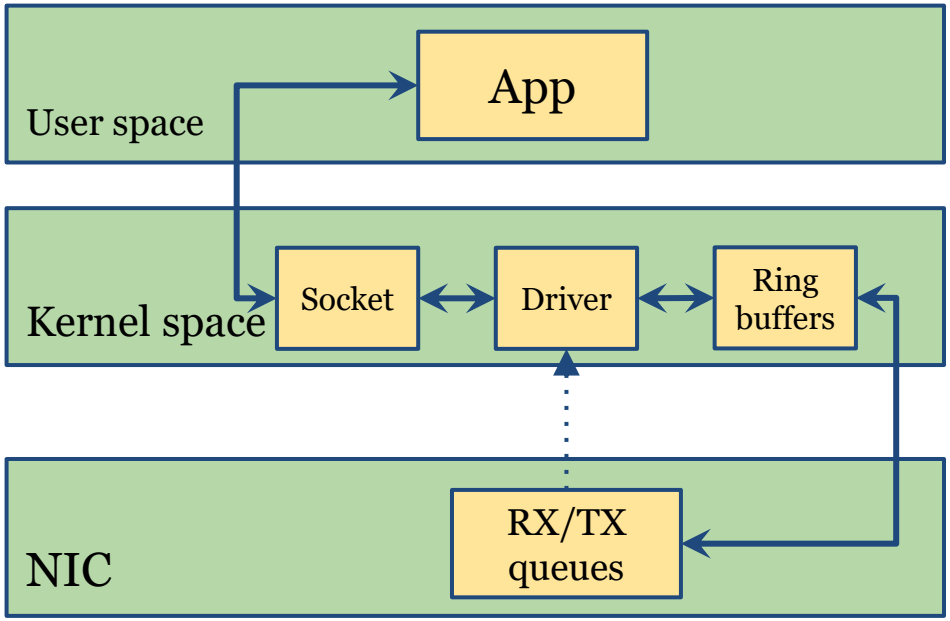
DPDK Boosts Packet Processing, Performance, and Throughput

- Data Plane Development Kit (DPDK) greatly boosts packet processing performance and throughput, allowing more time for data plane applications.
- DPDK can improve packet processing performance by up to ten times. **It's possible to achieve over 80 Mbps throughput on a single Intel® Xeon® processor**, and double that with a dual-processor configuration.[1] As a result, telecom and network equipment manufacturers (TEMs and NEMs) can lower development costs, use fewer tools and support teams, and get to market faster.

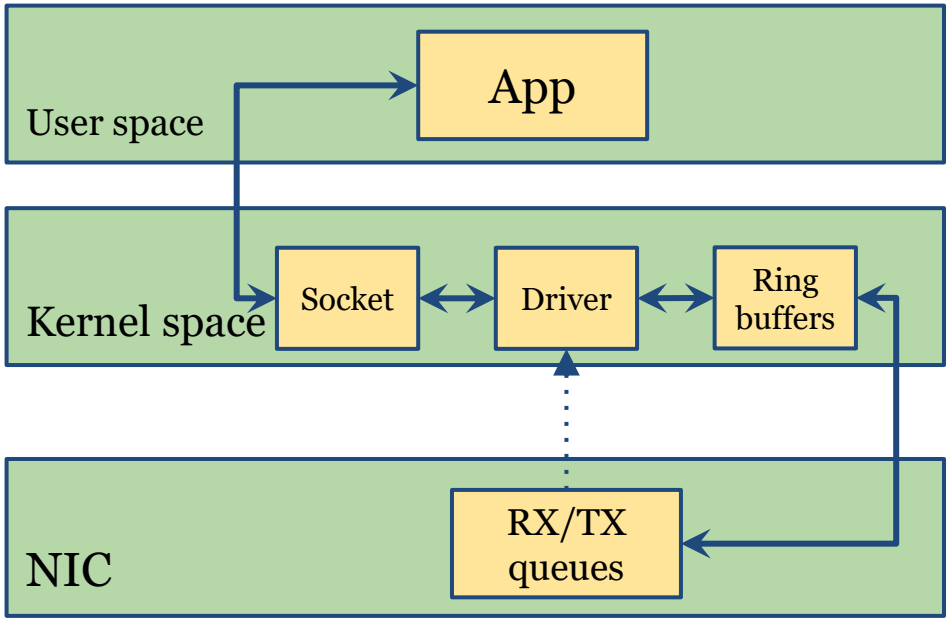
Benefits – Eliminating / Hiding Overheads

Eliminating	How?
	Polling
	User Mode Driver
	Pthread Affinity

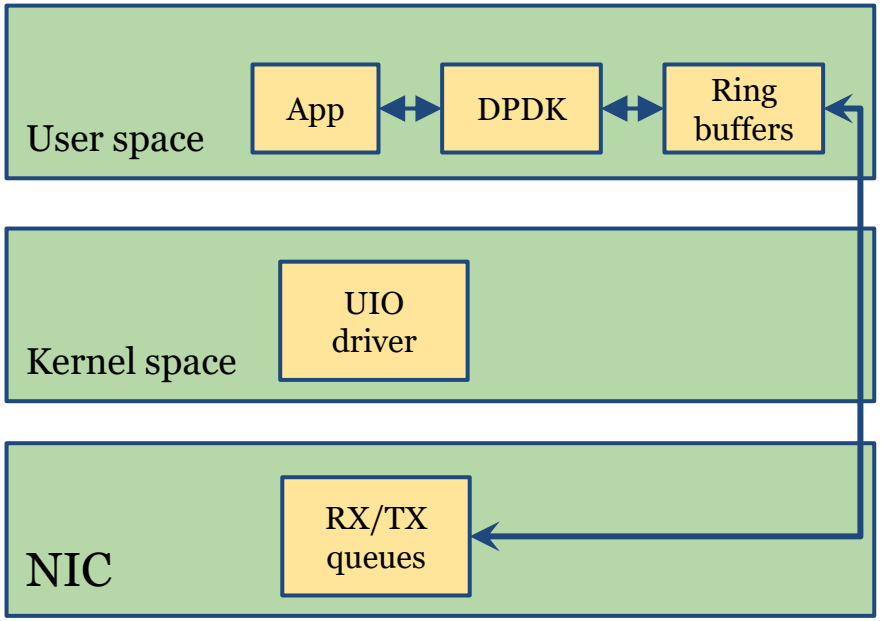
Packet processing in Linux



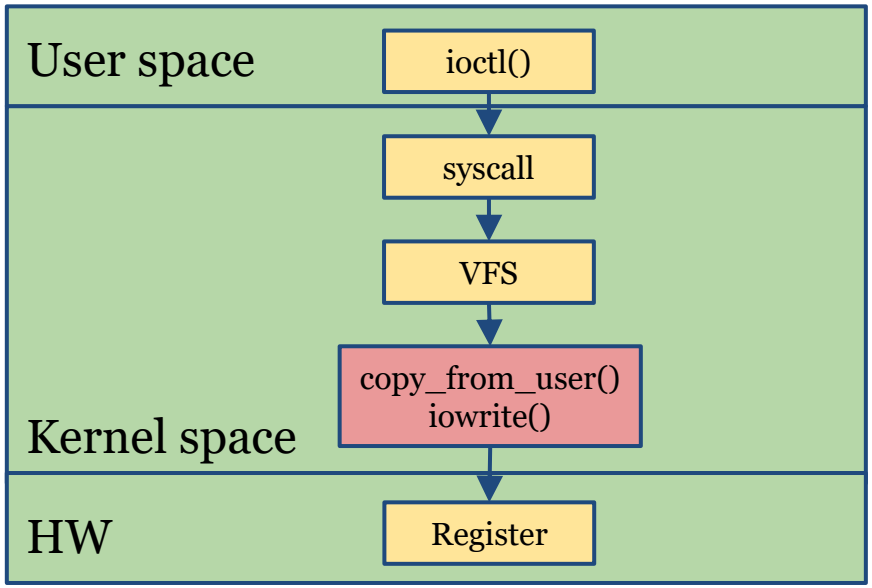
Packet processing in Linux



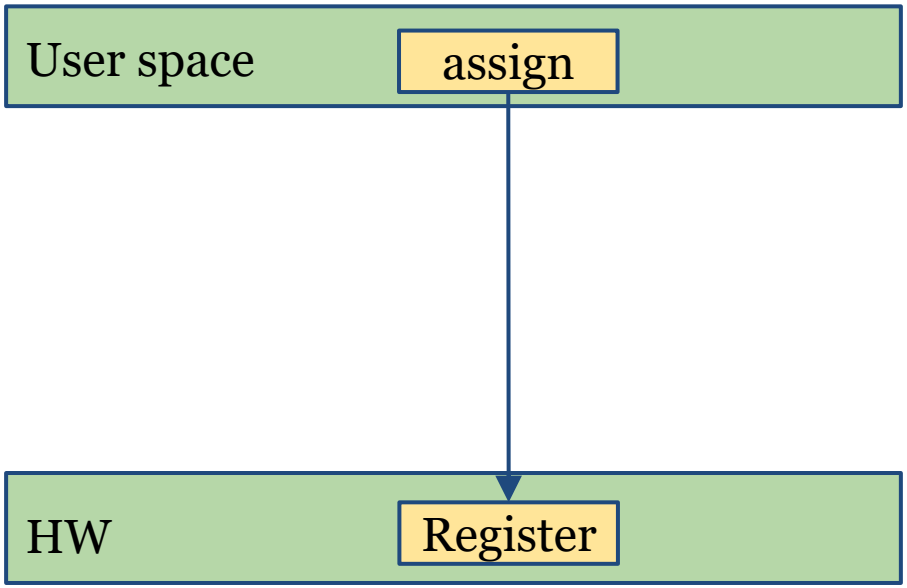
Packet processing with DPDK



Updating a register in Linux



Updating a register with DPDK



DPDK Packet Processing Concepts

- DPDK is designed for high-speed packet processing on IA. This is achieved by optimizing the software libraries to IA with some of the following concepts
  - Huge Pages
  - Prefetching
  - Intel® DDIO
  - Cache alignment
  - New Instructions
  - Memory Interleave
  - Ptheads with Affinity
  - NUMA
  - Memory Channel

DPDK Packet Processing Concepts

- DPDK is designed for high-speed packet processing on IA. This is achieved by optimizing the software libraries to IA with some of the following concepts
  - Huge Pages
  - Prefetching
  - Intel® DDIO
  - Cache alignment
  - New Instructions
  - Memory Interleave
  - Ptheads with Affinity
  - NUMA
  - Memory Channel

- Pthreads
  - On startup of the DPDK specifies the cores to be used via the Pthread call with affinity to tie an application to a core. Reducing the kernel's ability of moving the application to another local or remote core affecting performance.
  - The user may still use Ptheads or Fork calls after the DPDK has started to allow threads to float or multiple thread to be tied to a single core.

DPDK Packet Processing Concepts

- DPDK is designed for high-speed packet processing on IA. This is achieved by optimizing the software libraries to IA with some of the following concepts
  - Huge Pages
  - Prefetching
  - Intel® DDIO
  - Cache alignment
  - New Instructions
  - Memory Interleave
  - Ptheads with Affinity
  - NUMA
  - Memory Channel

- Pthreads
  - On startup of the DPDK specifies the cores to be used via the Pthread call with affinity to tie an application to a core. Reducing the kernel's ability of moving the application to another local or remote core affecting performance.
  - The user may still use Ptheads or Fork calls after the DPDK has started to allow threads to float or multiple thread to be tied to a single core.

- NUMA
  - DPDK utilizes NUMA memory for allocation of resources to improve performance for processing and PCIe I/O local to a processor.
  - With out the NUMA set in a dual socket system memory is interleaved between the two sockets.

- Huge Pages
  - DPDK utilizes 2M and 1G hugepages to reduce the case of TLB misses which can significantly affect a cores overall performance.

DPDK Packet Processing Concepts

- DPDK is designed for high-speed packet processing on IA. This is achieved by optimizing the software libraries to IA with some of the following concepts
  - Huge Pages
  - Prefetching
  - Intel® DDIO
  - Cache alignment
  - New Instructions
  - Memory Interleave
  - Ptheads with Affinity
  - NUMA
  - Memory Channel

- Cache Alignment
  - Better performance by aligning structures on 64 Byte cache lines.

- Software Prefetching
  - Needs to be issued "appropriately" ahead of time to be effective. Too early could cause eviction before use
  - Allows cache to be populated before data is accessed

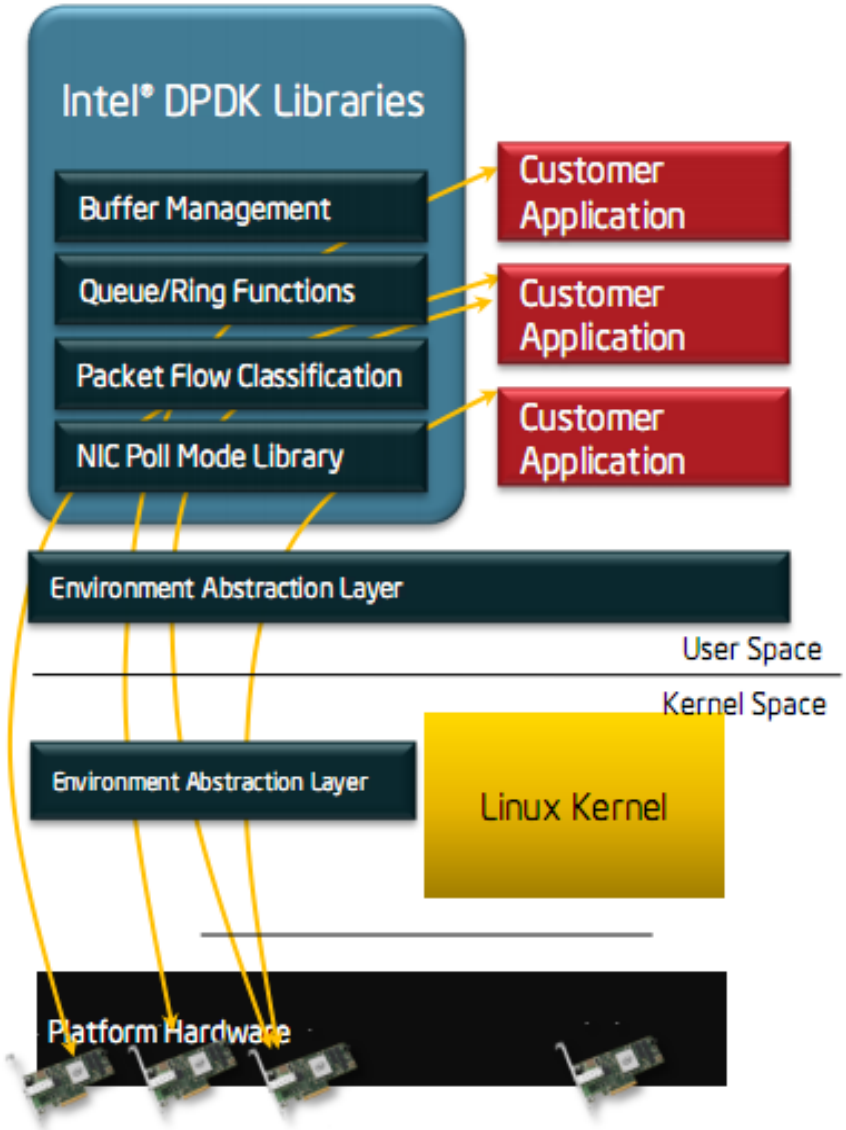
- Memory channel use
  - Memory pools add padding to objects to ensure even use of memory channels
  - Number of channels specified at application start up

What additionaly inside DPDK?

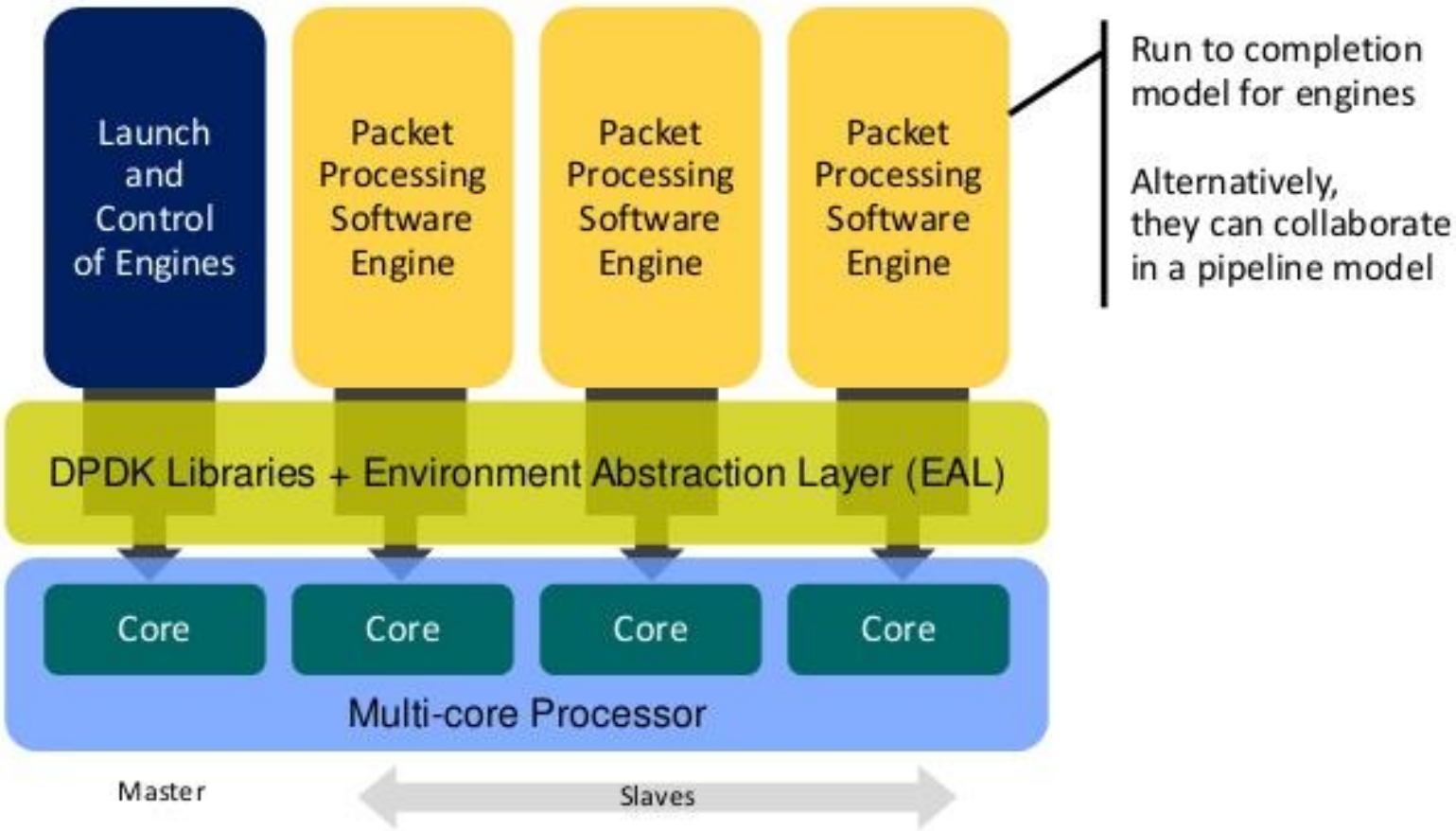
- Processor affinity (separate cores)
- UIO (no copying from kernel)
- Polling (no interrupts overhead)
- Lockless synchronization (avoid waiting)

Intel DPDK

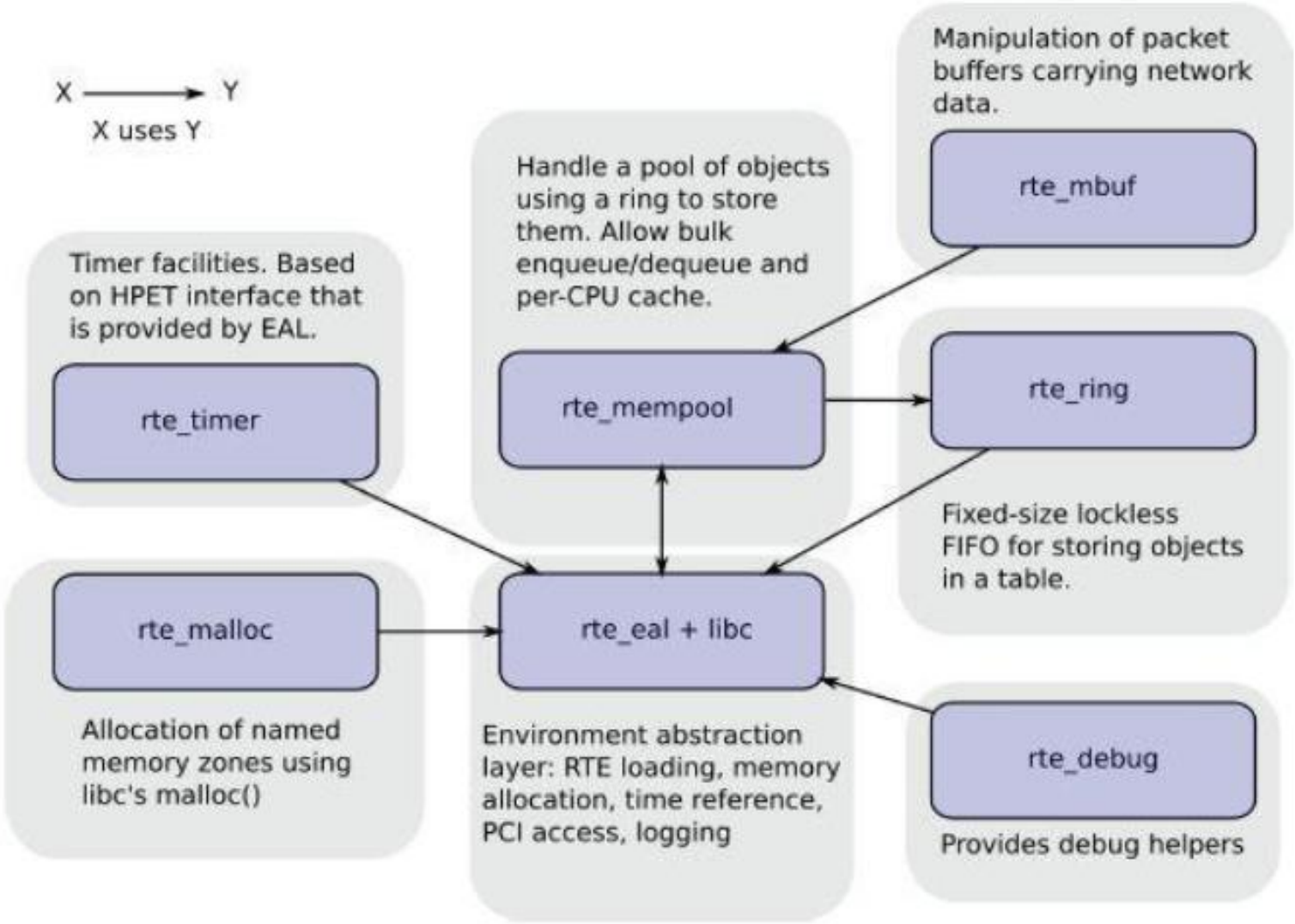
- Buffer and Memory Manager
  Manage the allocation of objects non-*NUMA* using *huge pages* through *rings*, reducing TLB access, also, perform a pre-allocation of fixed buffer space for each core
- Queue Manager
  Implements *lockless queues*, allow packets to be processed by different software components with no contention
- Flow Classification
  Implements hash functions from information tuples, allow packets to be positioned rapidly in their flow paths. Improves *throughput*
- Pool Mode Driver
  Temporary hold times thus avoiding raise NIC interruptions



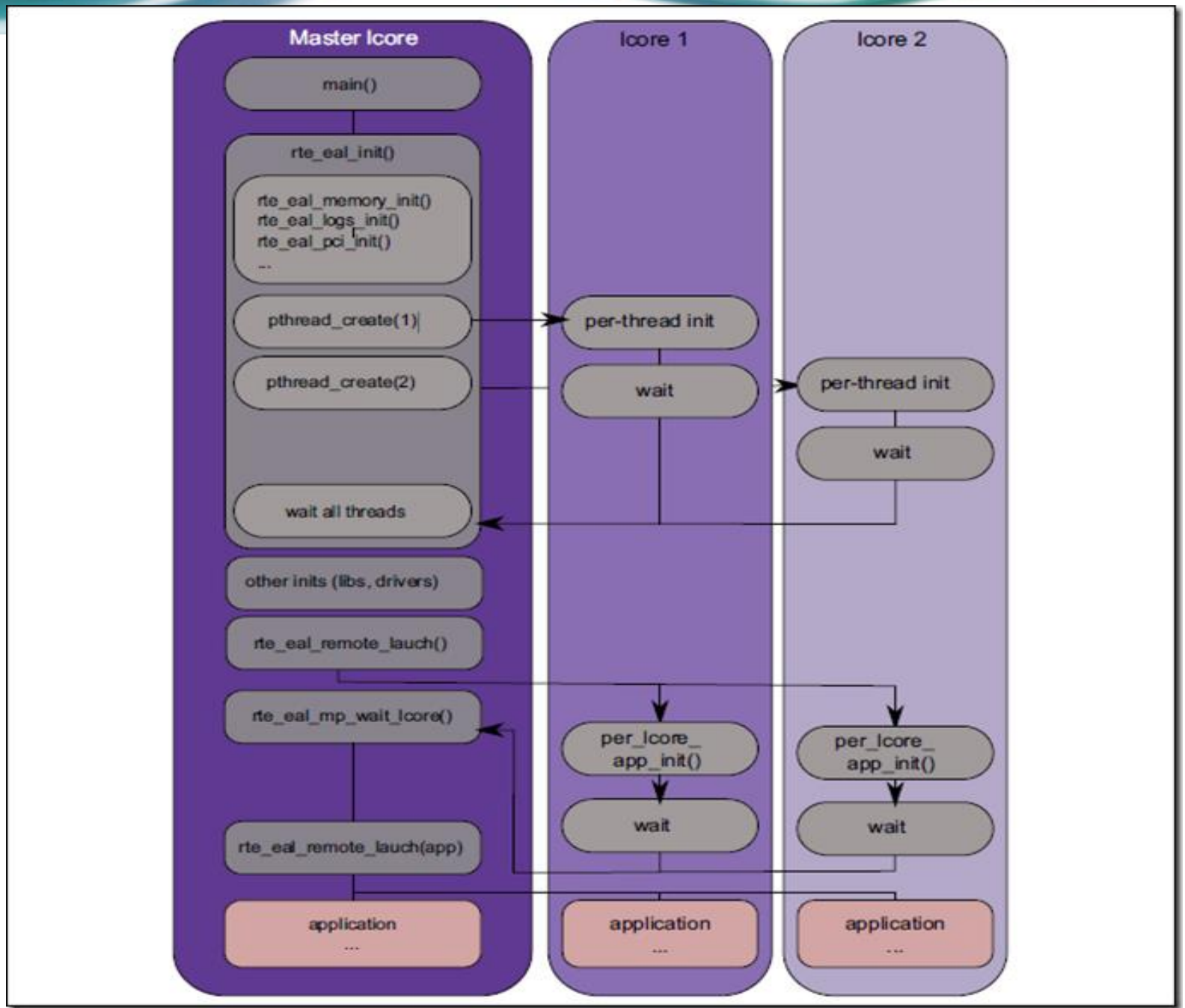
DPDK Programming Model



Relationship of DPDK Libraries

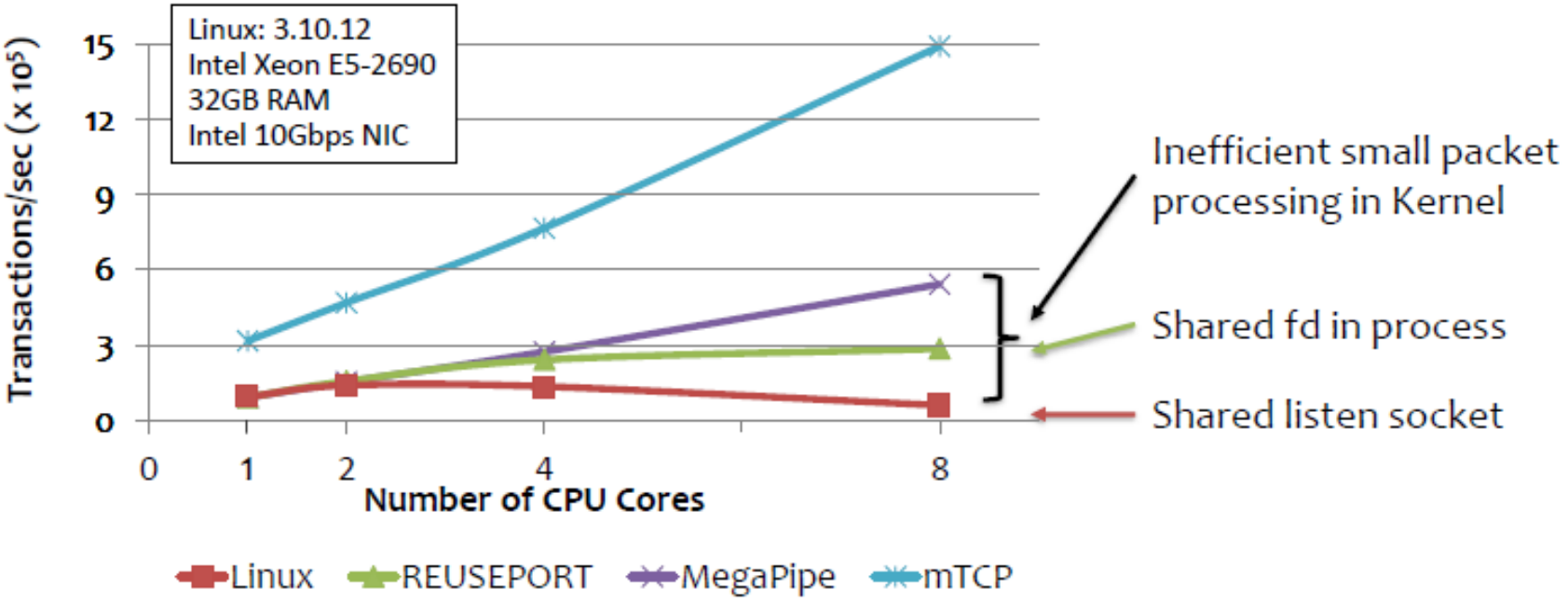


Initialization and Core Launching

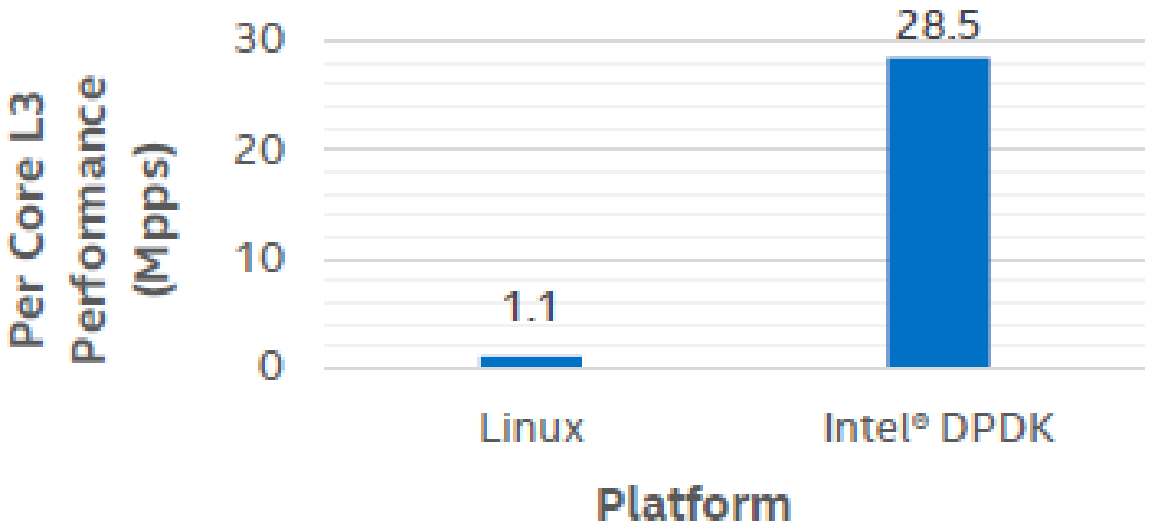


Multicore Scalability

- 64B ping/pong messages per connection
- Heavy connection overhead, small packet processing overhead
- 25x Linux, 5x SO_REUSEPORT*[LINUX3.9], 3x MegaPipe*[OSDI'12]



Performance



Questions?

Thank you!