

Kernel Bypass

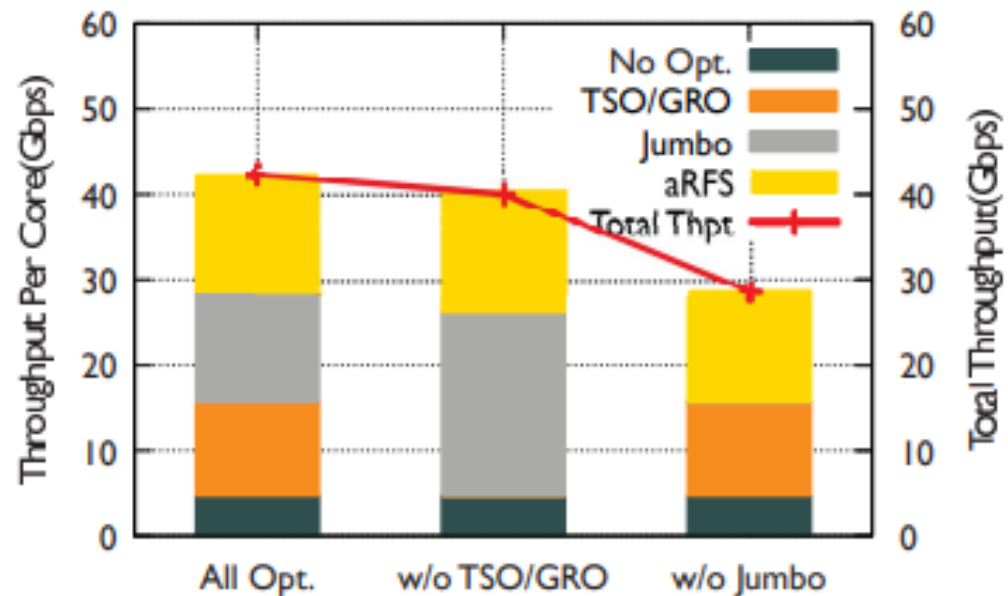
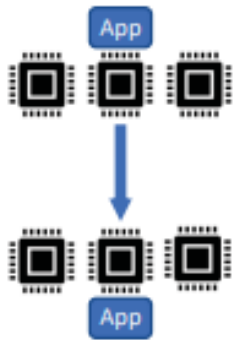
ECE/CS598HPN

Radhika Mittal

Performance overhead in kernel stack

- Protocol processing
- Data copy
- Cache contention (between flows sharing same NUMA node)
- CPU scheduling overheads (locking, context switching)
- Interrupts
- Managing heavy datastructures (skbs)

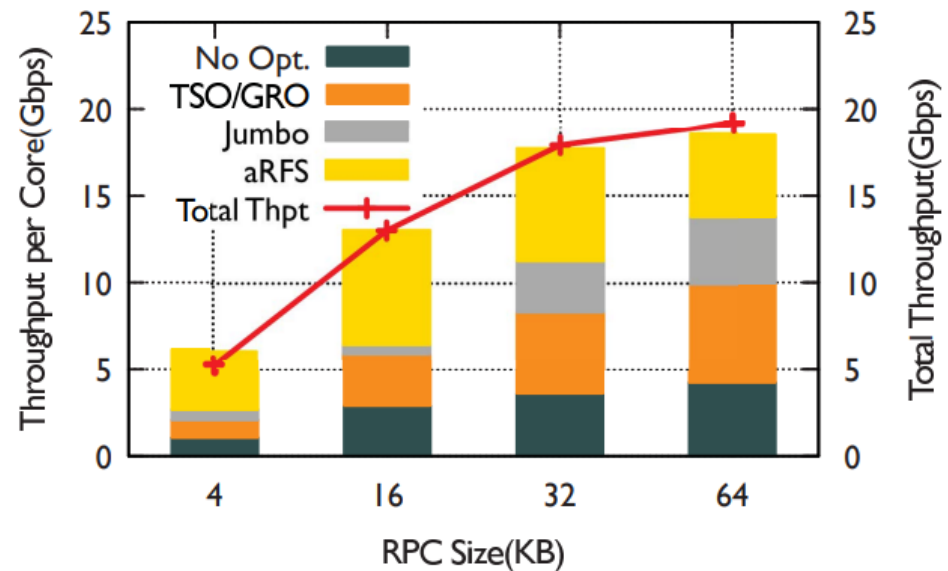
Results from “understanding host network stack overheads”



Results from “understanding host network stack overheads”



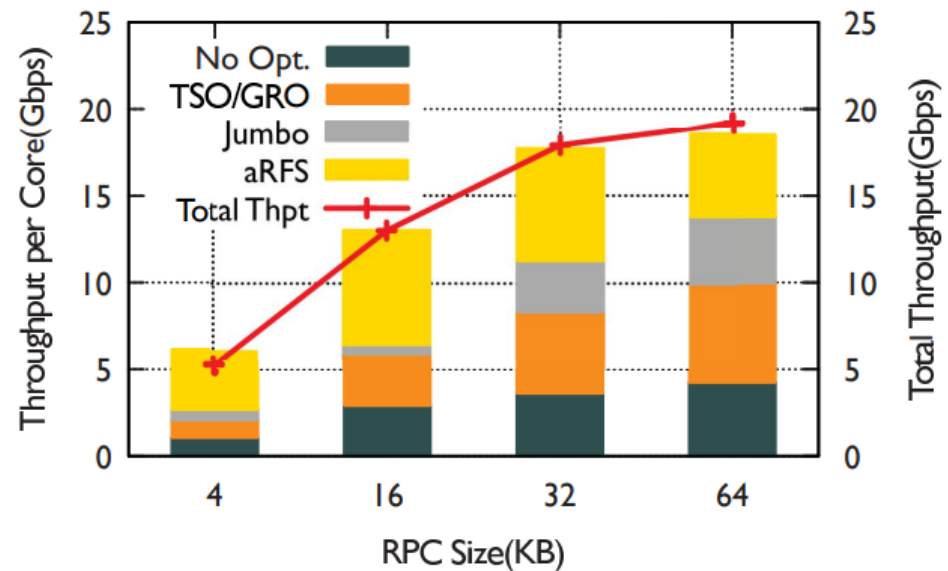
16:1 incast
ping-pong workload
vary flow (RPC) size



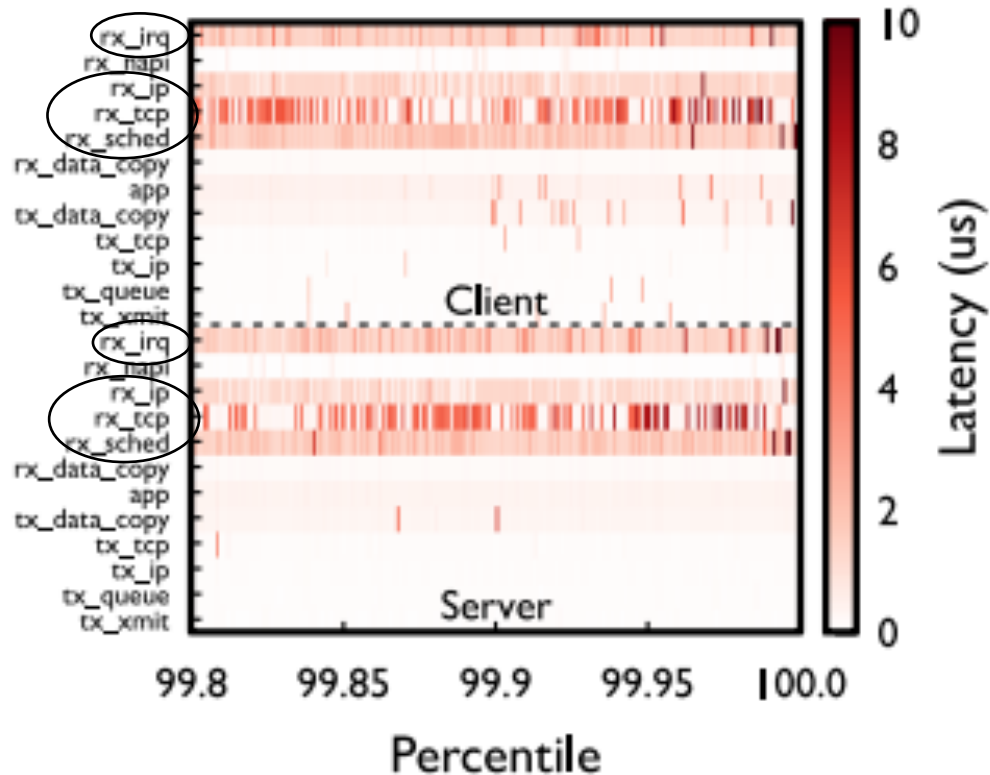
Results from “understanding host network stack overheads”



16:1 incast
ping-pong workload
vary flow (RPC) size



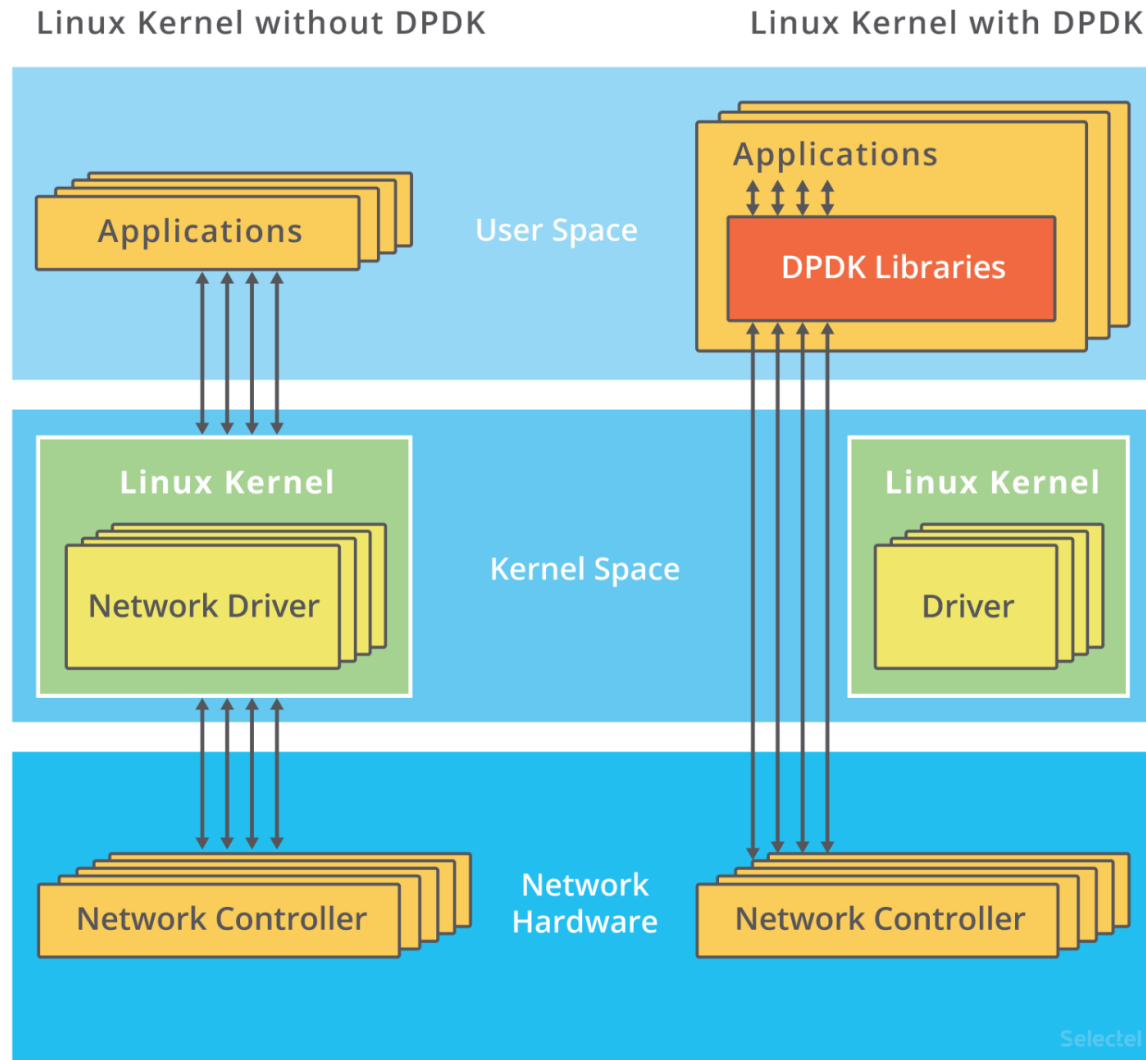
Results from “understanding host network stack latency” (Zuo et. al., SIGCOMM’26)



(b) Tail latency breakdown (P99.8–P100)

Kernel Bypass Packet I/O

Dataplane Development Kit (DPDK)



Source: <https://blog.selectel.com/introduction-dpdk-architecture-principles/>

Dataplane Development Kit (DPDK)

- User-space packet processing (kernel bypass).
 - Avoid context switching overhead.
- Poll Mode Driver (PMD).
 - Avoid interrupt processing overhead.
 - *Keeps a core busy.*
- Memory usage optimizations
 - Light-weight *mbufs*.
 - Memory pools that use hugepages, cache alignment, etc.
 - Lockless ring buffers.

Other examples

- NetMap (Rizzo et. al., ATC'12)
 - In-kernel module for efficient packet processing.
 - Light-weight packet buffers.
 - NIC DMA's packets directly to application memory (no kernel copy)
- Packet Shader (Han et. al, SIGCOM 2010)
 - Modified packet I/O engine in the kernel.
 - Fetches packets through a combination of interrupts and polling.
 - Processes packets using GPU in userspace.

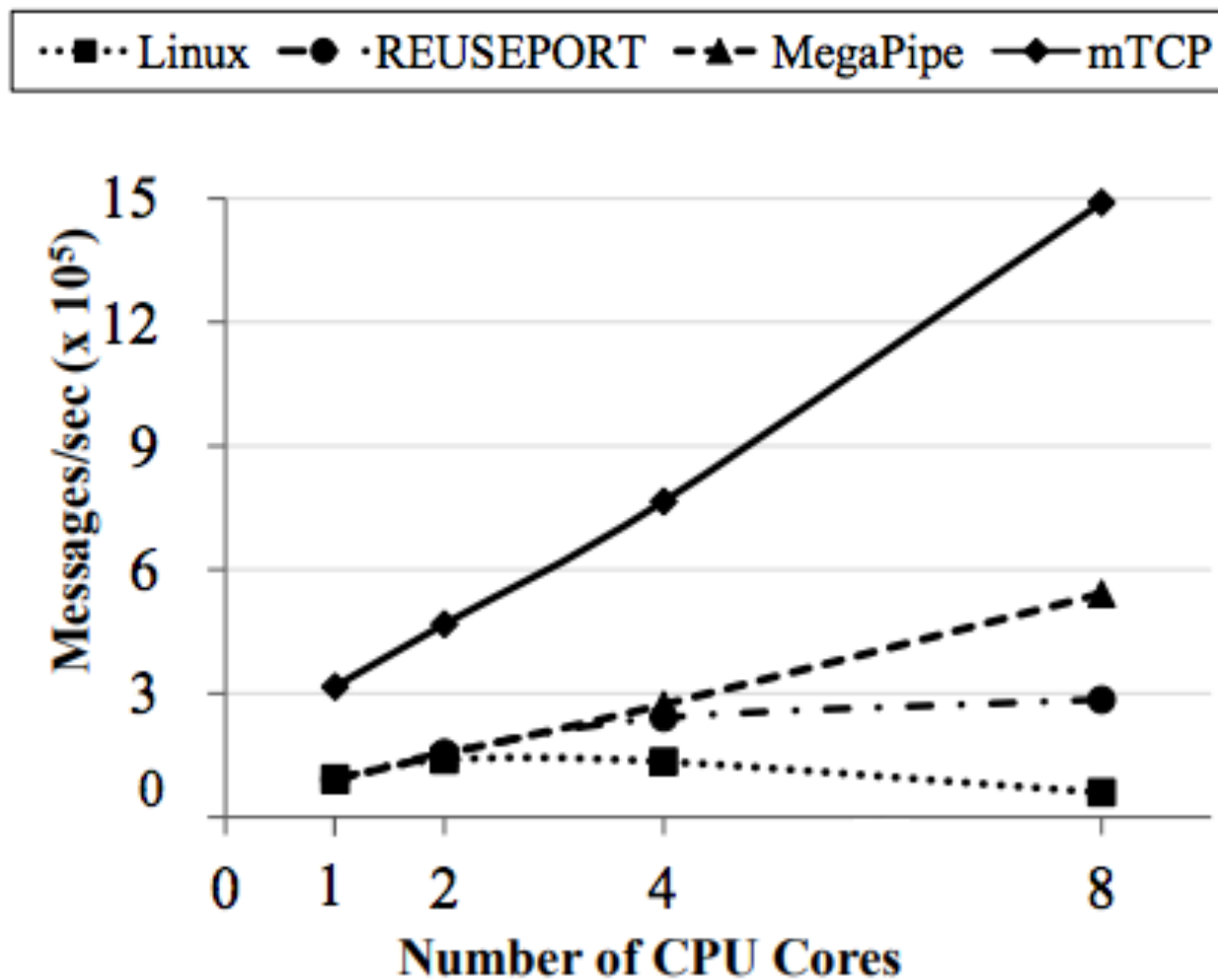
Kernel Bypass Packet I/O Engine

- Provide mechanisms for delivering packets to user space.
- Do not implement a network stack.

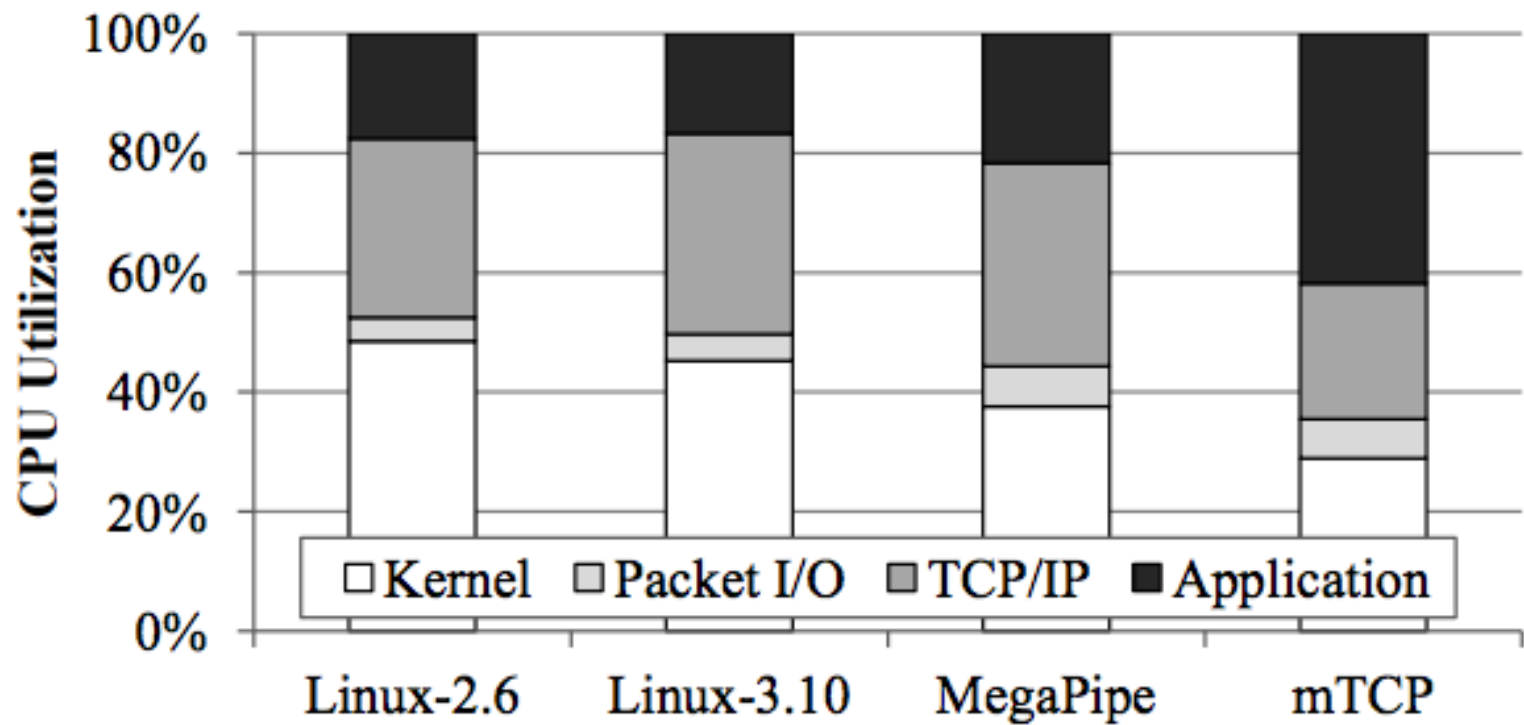
mTCP (Jeong et. al., NSDI'14)

- User-space TCP/IP stack built over kernel-bypass packet I/O engines.
 - Implementation in paper over PacketShader.
 - DPDK based implementation also available.

mTCP



mTCP



mTCP -- Issues

- Dedicated threads for the TCP stack.
 - Avoid intrusive inter-twining of application and TCP processing.
 - Batching to reduce switching overheads.
 - Adds latency.
- Security vulnerabilities with user-space network stack.

IX: A Protected Dataplane Operating System for High Throughput and Low Latency

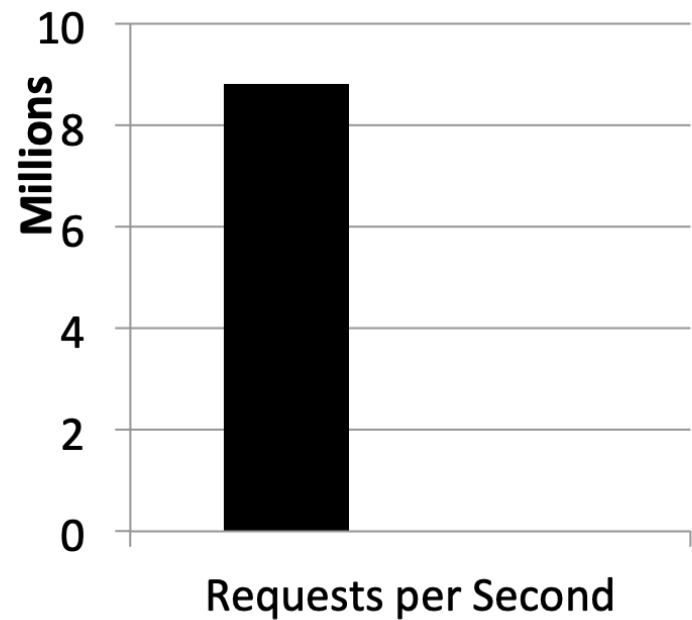
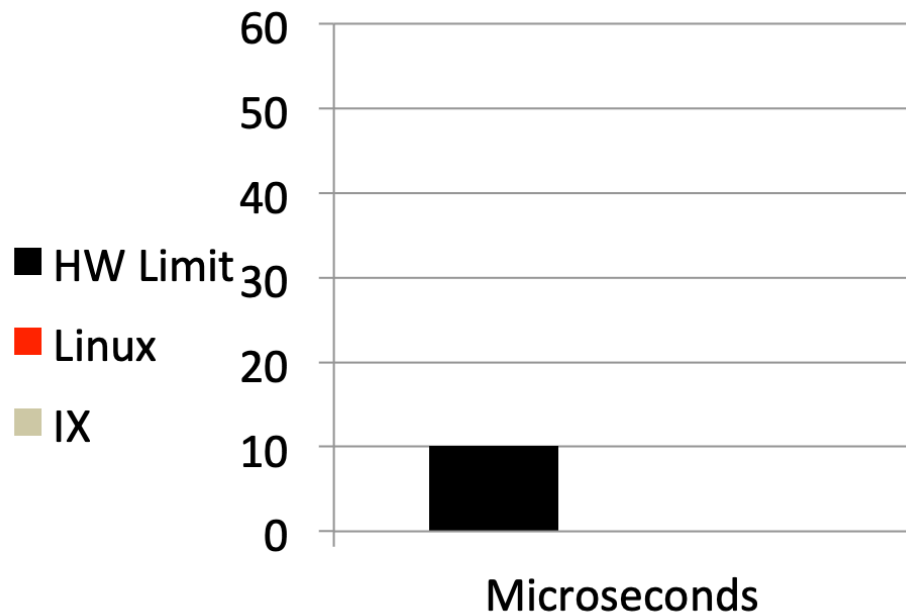
Adam Belay, George Prekas, Ana Klimovic,
Samuel Grossman, Christos Kozyrakis, Edouard Bugnion

OSDI'14 (Best Paper)

Slides borrowed from Adam's OSDI talk.

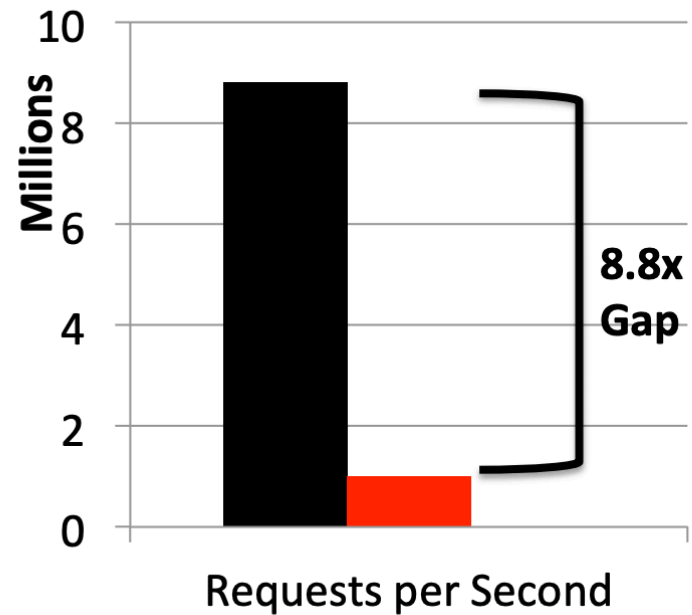
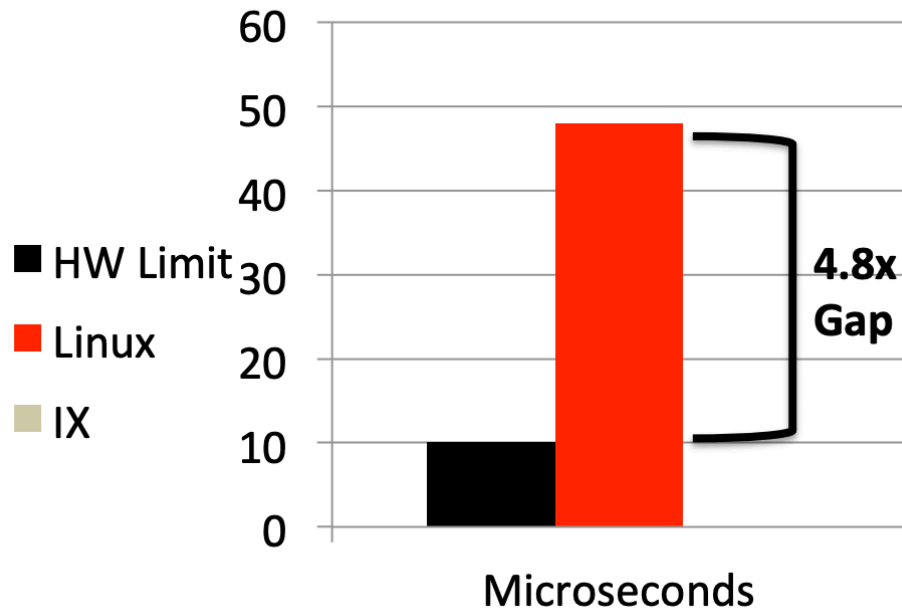
HW is fast

64-byte TCP Echo:



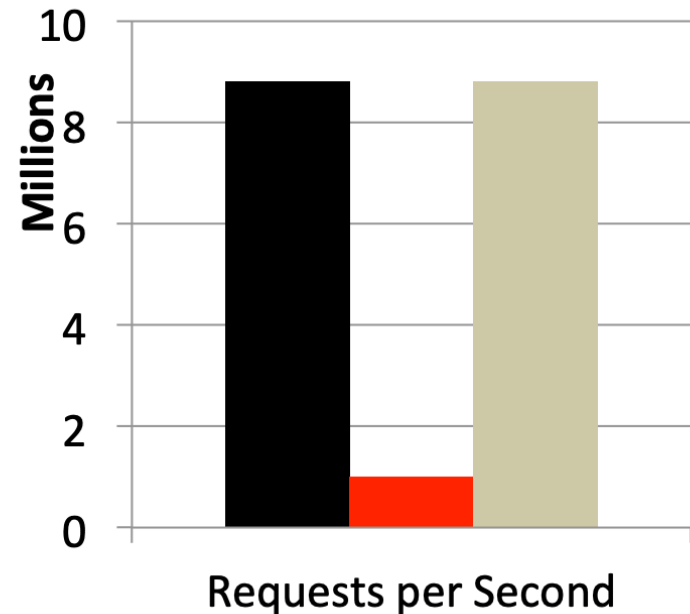
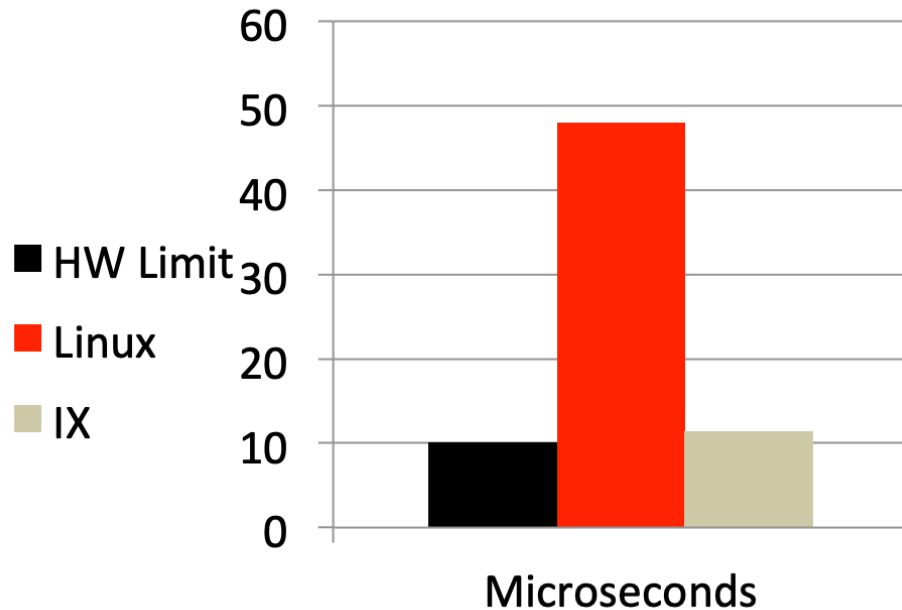
HW is fast, but SW is the bottleneck

64-byte TCP Echo:

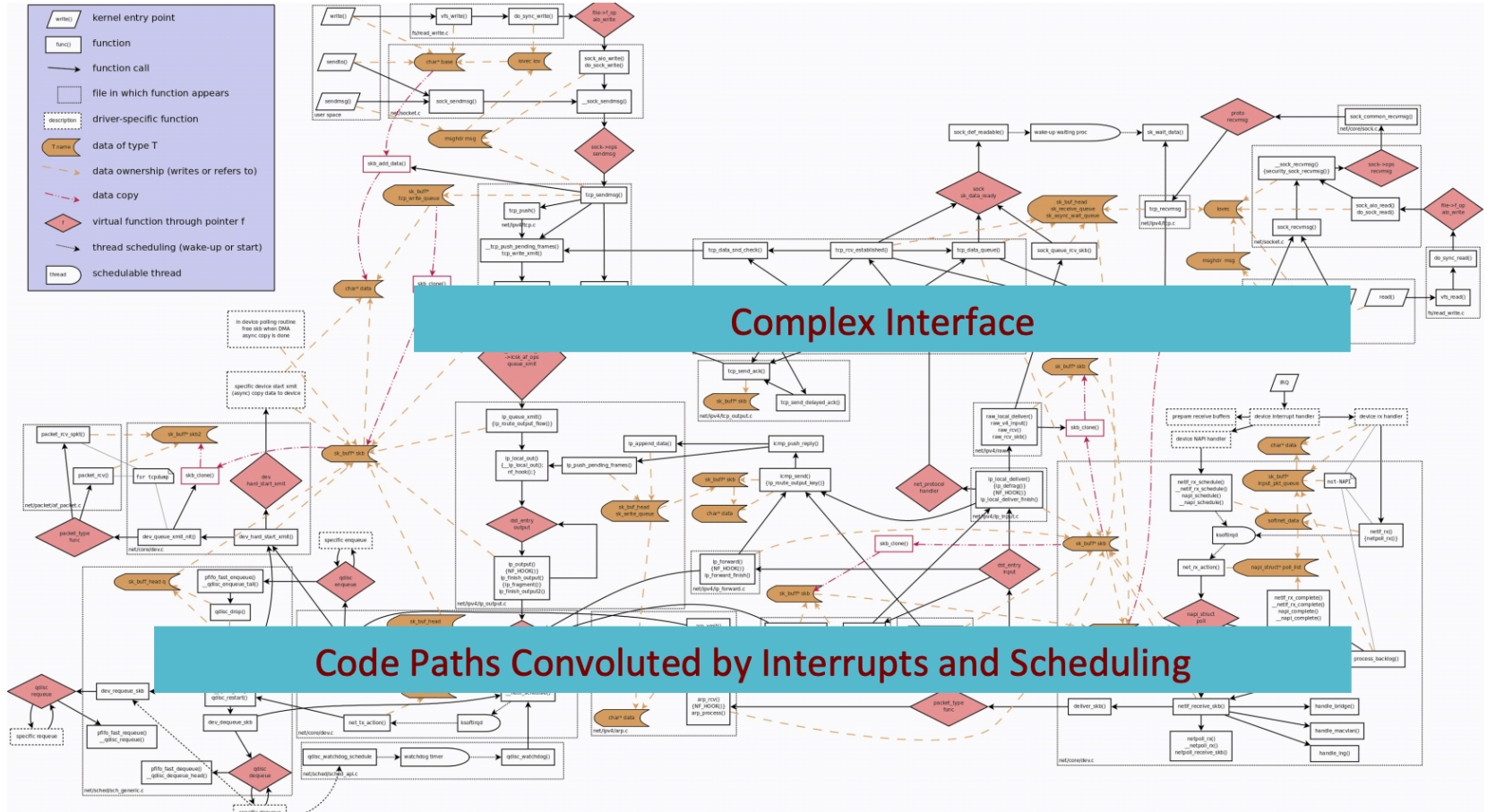


IX closes the SW performance gap

64-byte TCP Echo:



Why is SW slow?



Created by: Arnout Vandecappelle

http://www.linuxfoundation.org/collaborate/workgroups/networking/kernel_flow

Problem: 1980's Software Architecture

- Berkeley sockets, designed for CPU time sharing
- Today's large-scale datacenter workloads:

Hardware: Dense Multicore + 10 GbE (soon 40)

- API scalability critical!
- Gap between compute and RAM -> Cache behavior matters
- Packet inter-arrival times of 50 ns

Scale out access patterns

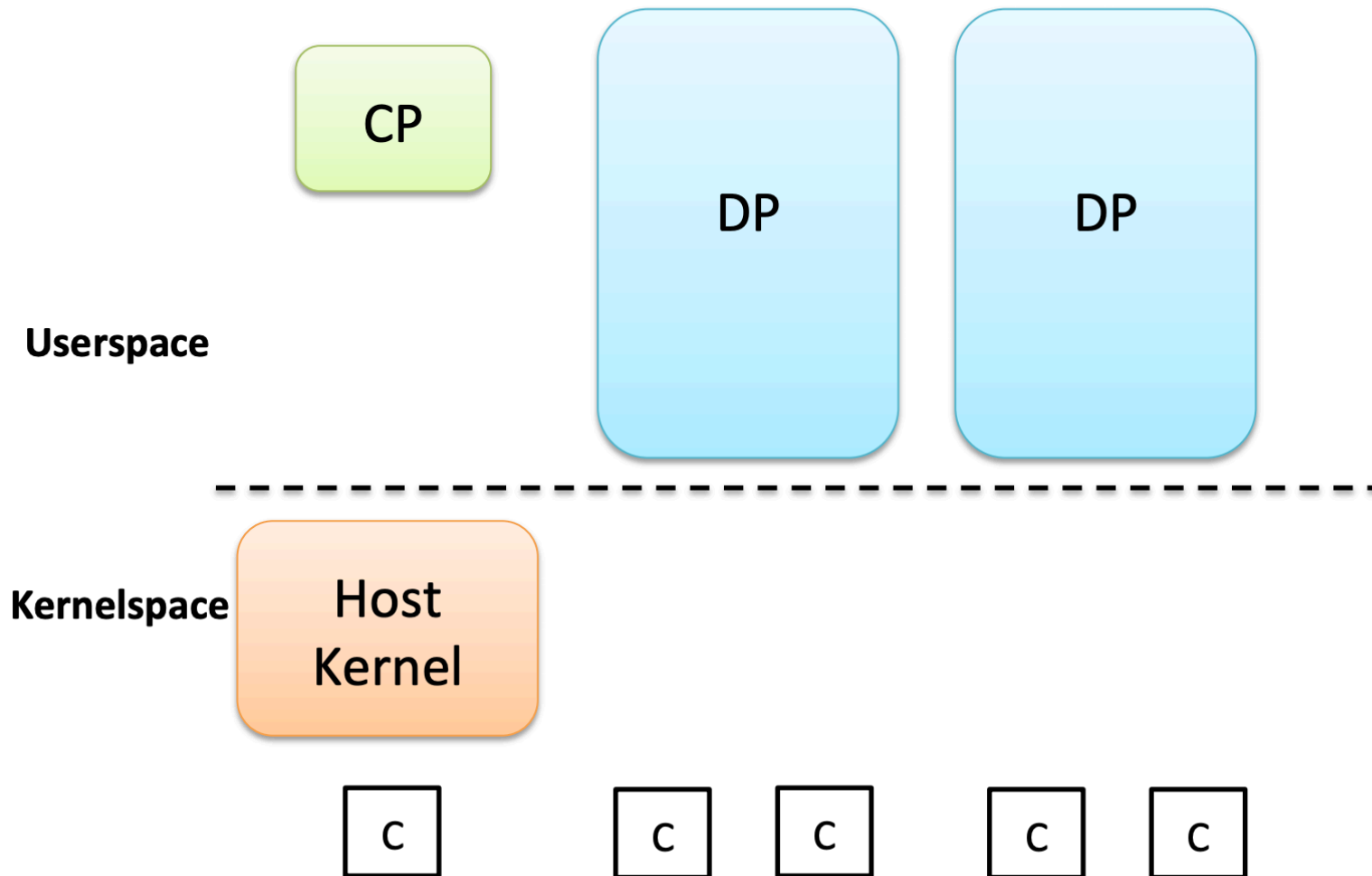
- Fan-in -> Large connection counts, high request rates
- Fan-out -> Tail latency matters!

IX Key Design Decisions

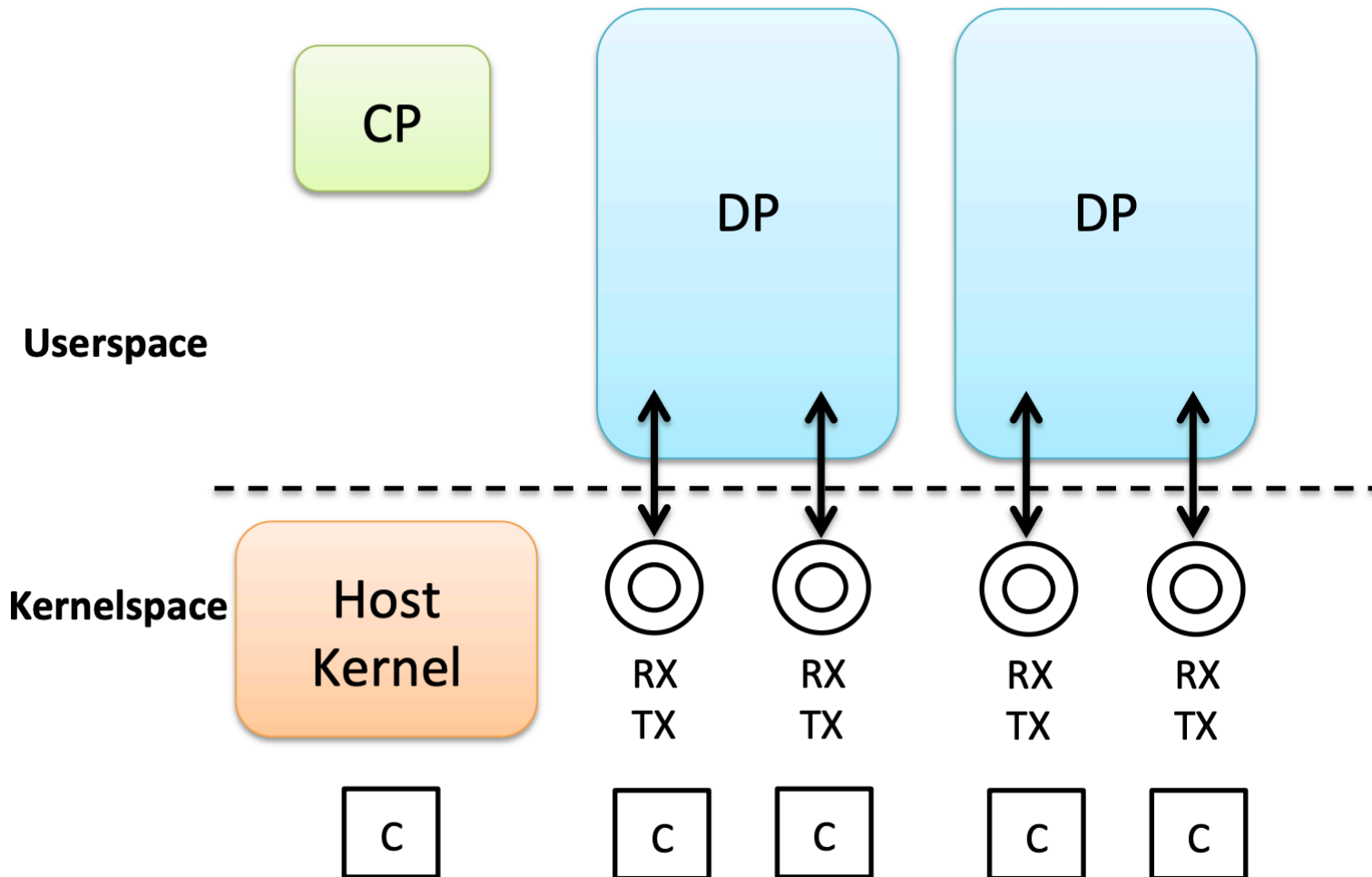
IX Key Design Decisions

- Separation of control plane and dataplane
 - Control plane handles resource allocation at coarse granularity (memory, core, and NIC queues).
- Run to completion packet processing.
 - Adaptive Batching
 - Zero-copy
 - Synchronization-free processing

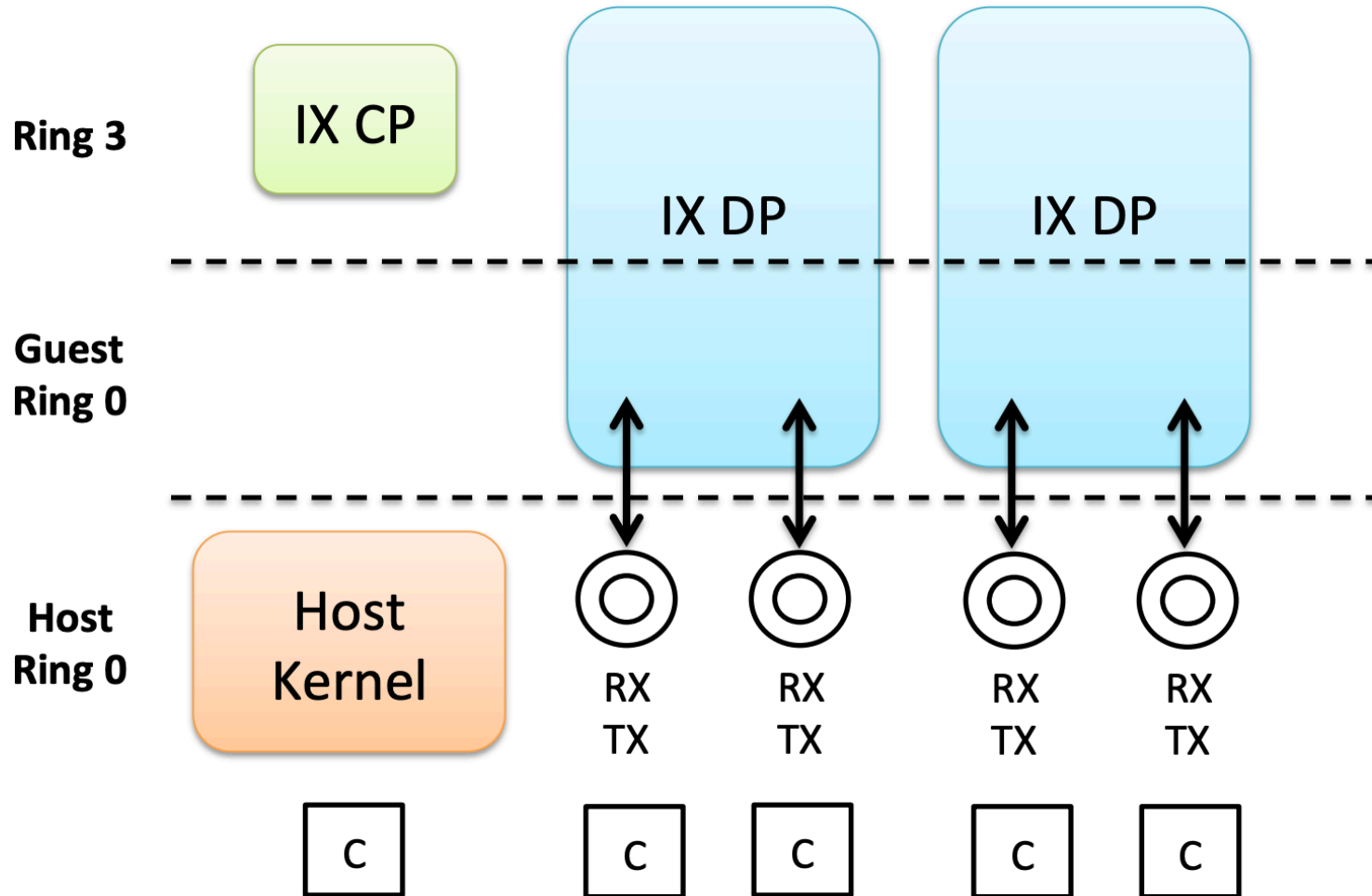
IX: Separation of Control and Data Plane



IX: Separation of Control and Data Plane



IX: Separation of Control and Data Plane



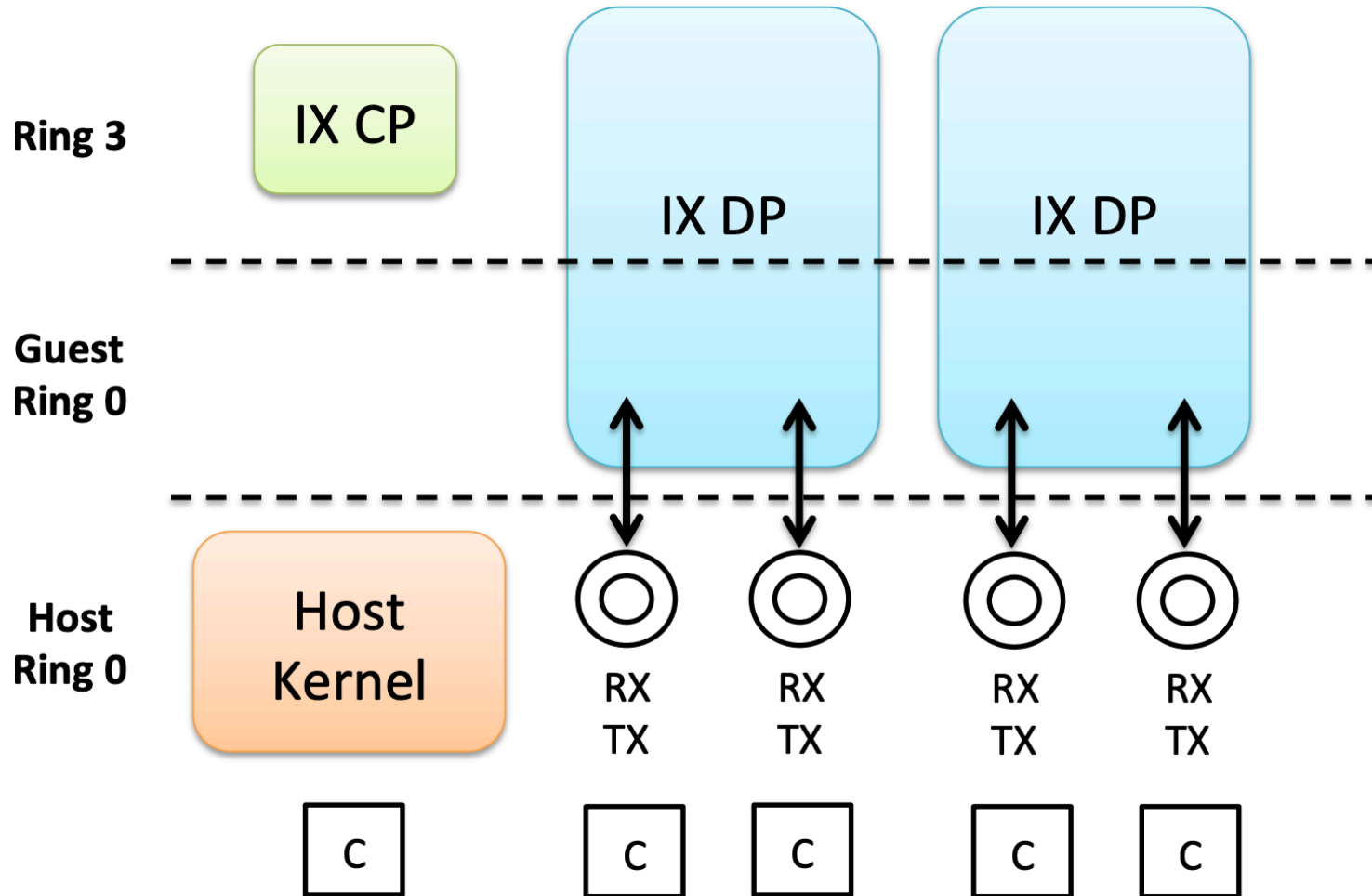
Three-way isolation

- Between IX control plane, dataplane, and untrusted user code.
- Use modern hardware virtualization techniques.

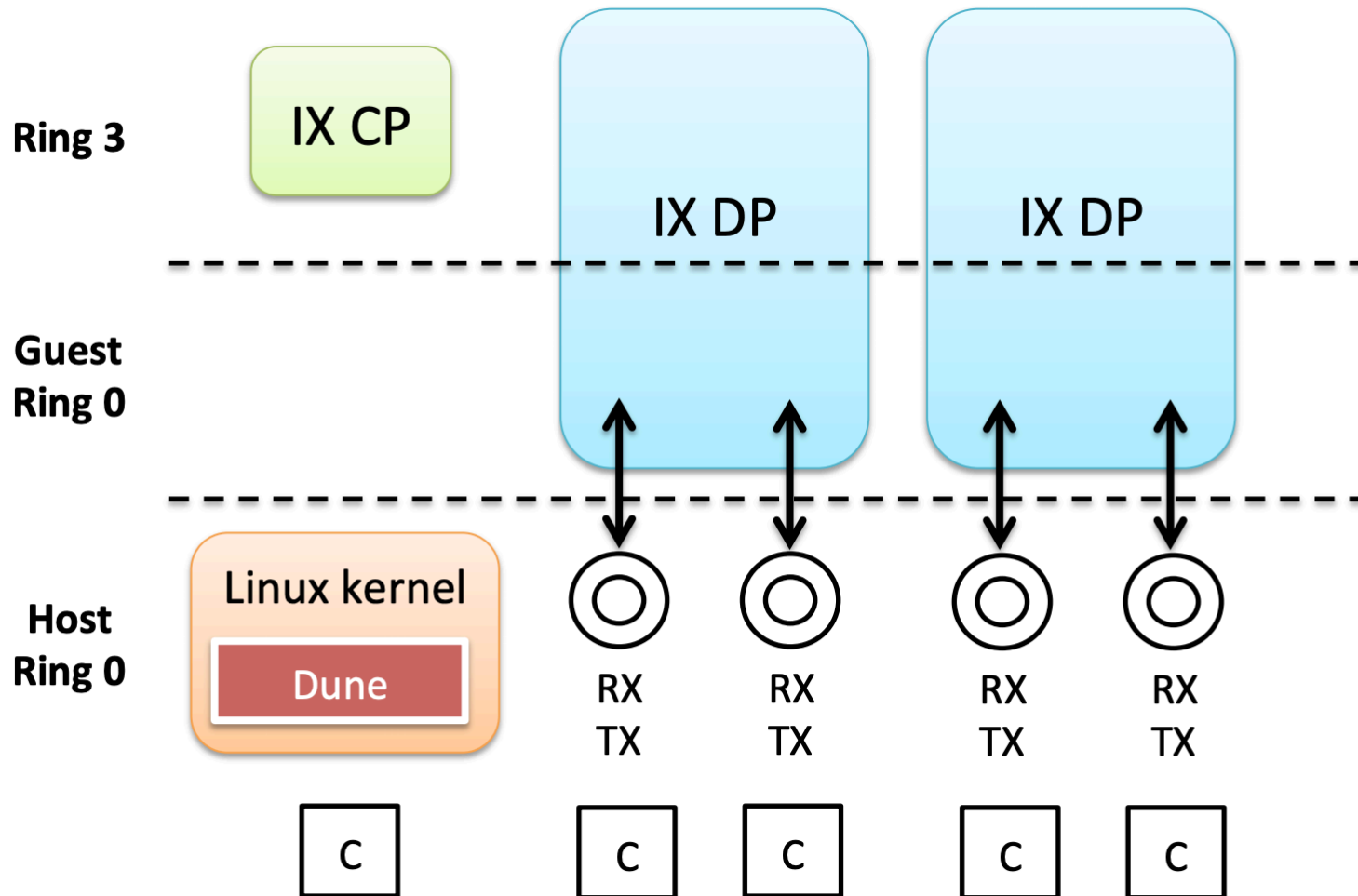
Detour: what is virtualization?

- Trick a guest OS into believing it has direct access to hardware (CPU, NIC, etc).
- Hypervisor or Virtual Machine Monitor (VMM) controls the guest VM's access, provides isolation, etc.
- Hardware virtualization techniques (e.g. Intel's VT-x) allow guest VMs to directly access hardware in a controlled manner.
 - Through extra privilege level (non-root ring 0) for guest OS.
 - Less privileged than root ring 0 (Host OS / Hypervisor)
 - More privileged than ring 3 (guest applications)

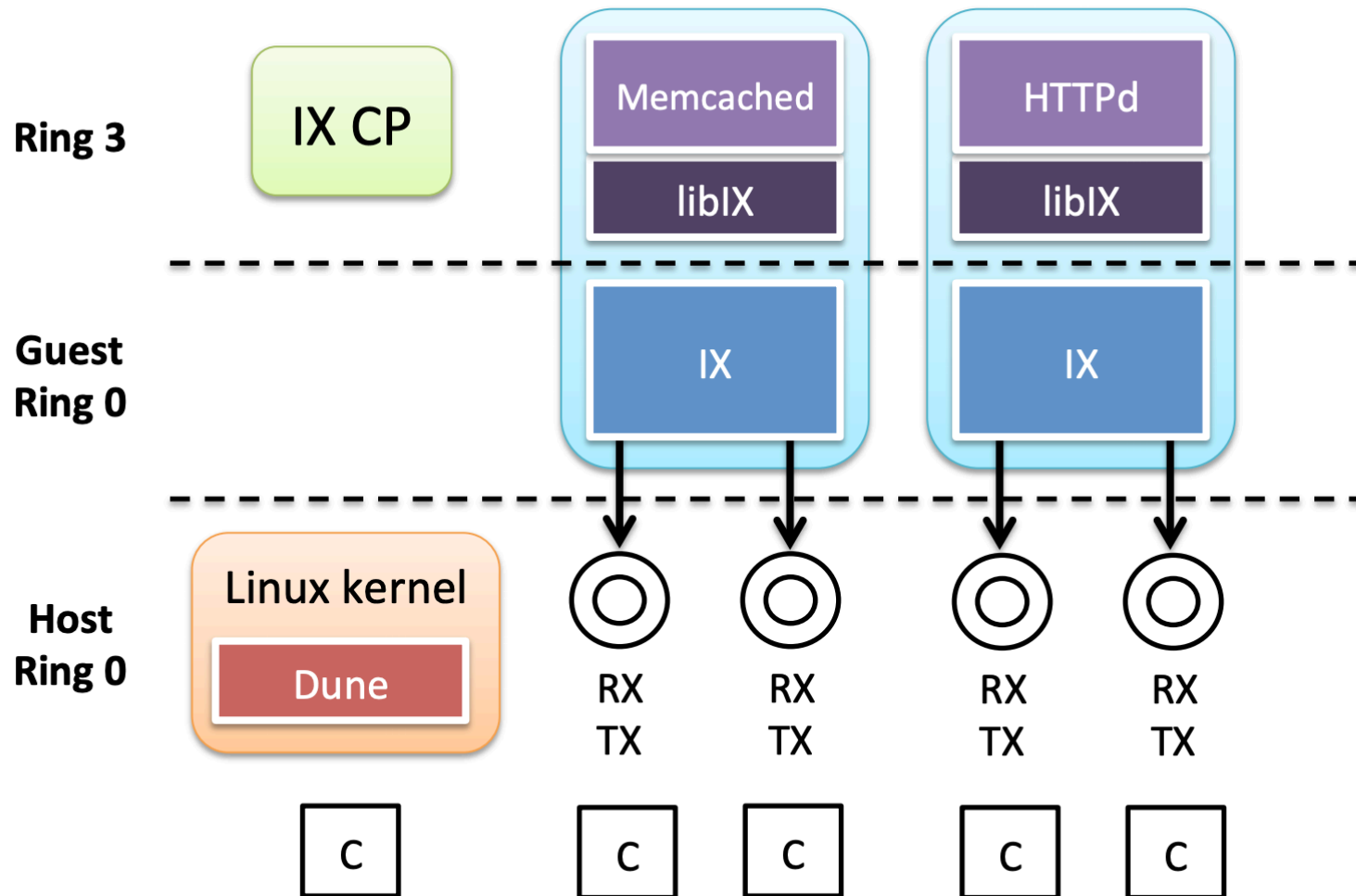
IX: Separation of Control and Data Plane



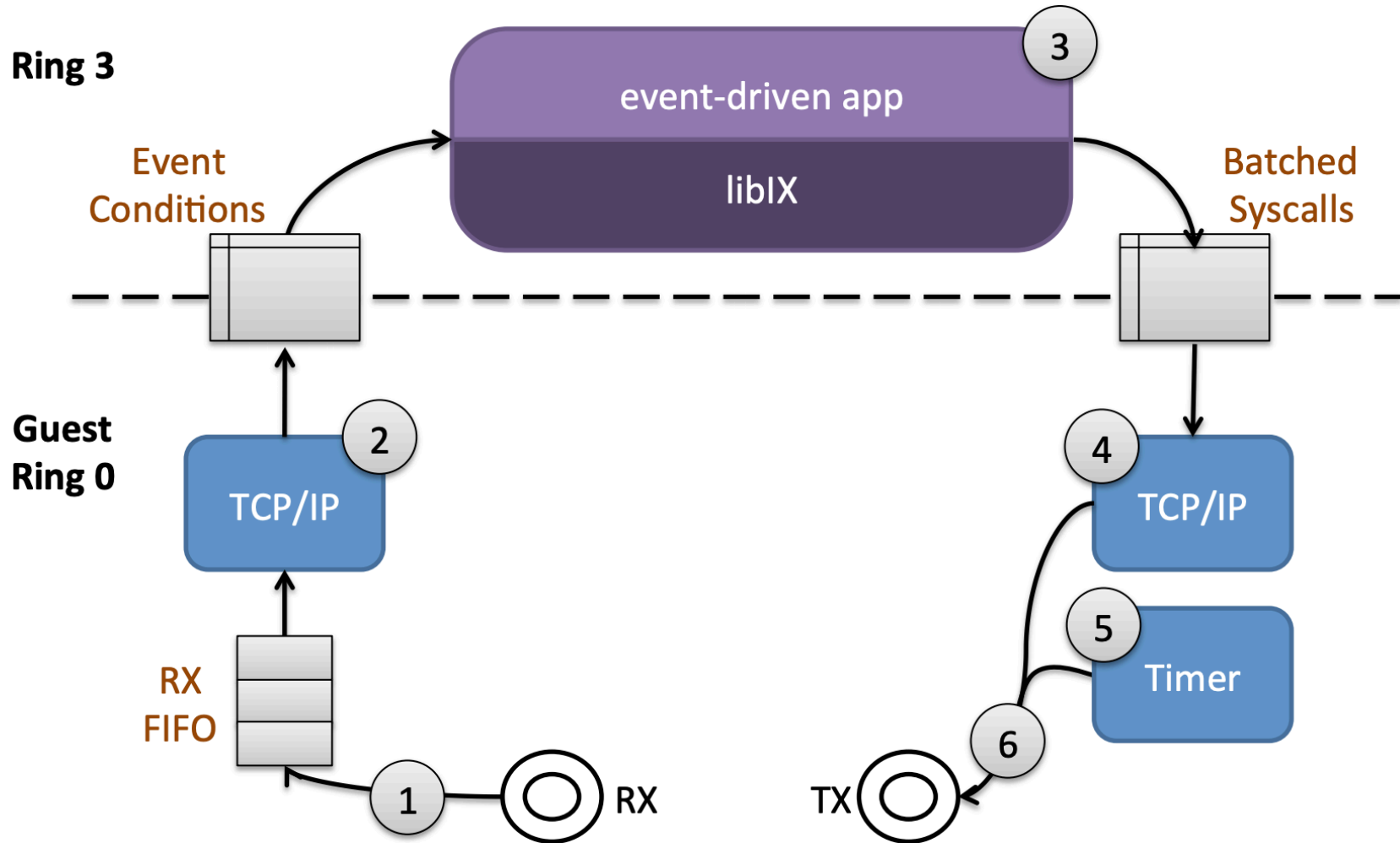
IX: Separation of Control and Data Plane



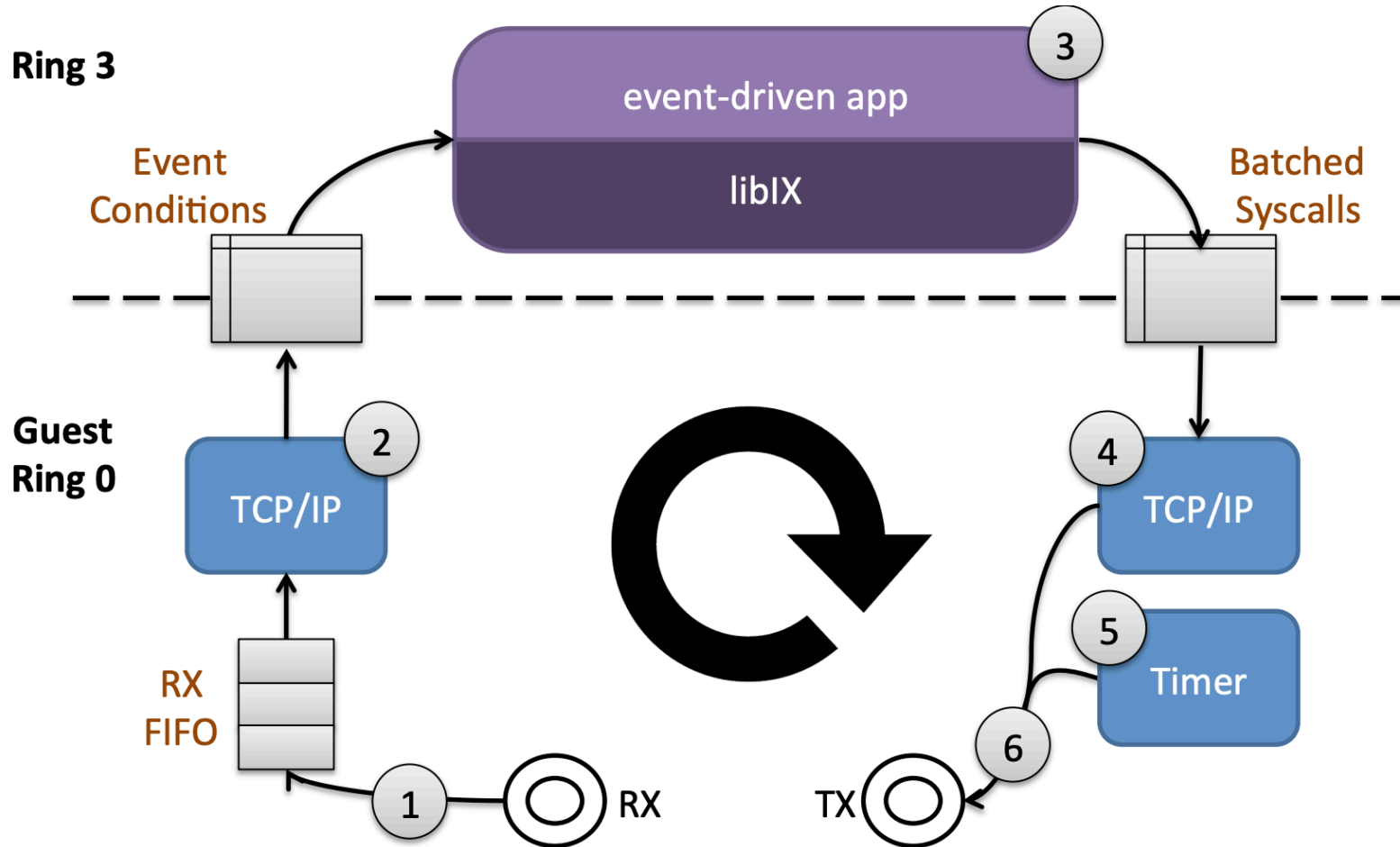
IX: Separation of Control and Data Plane



IX Execution Pipeline

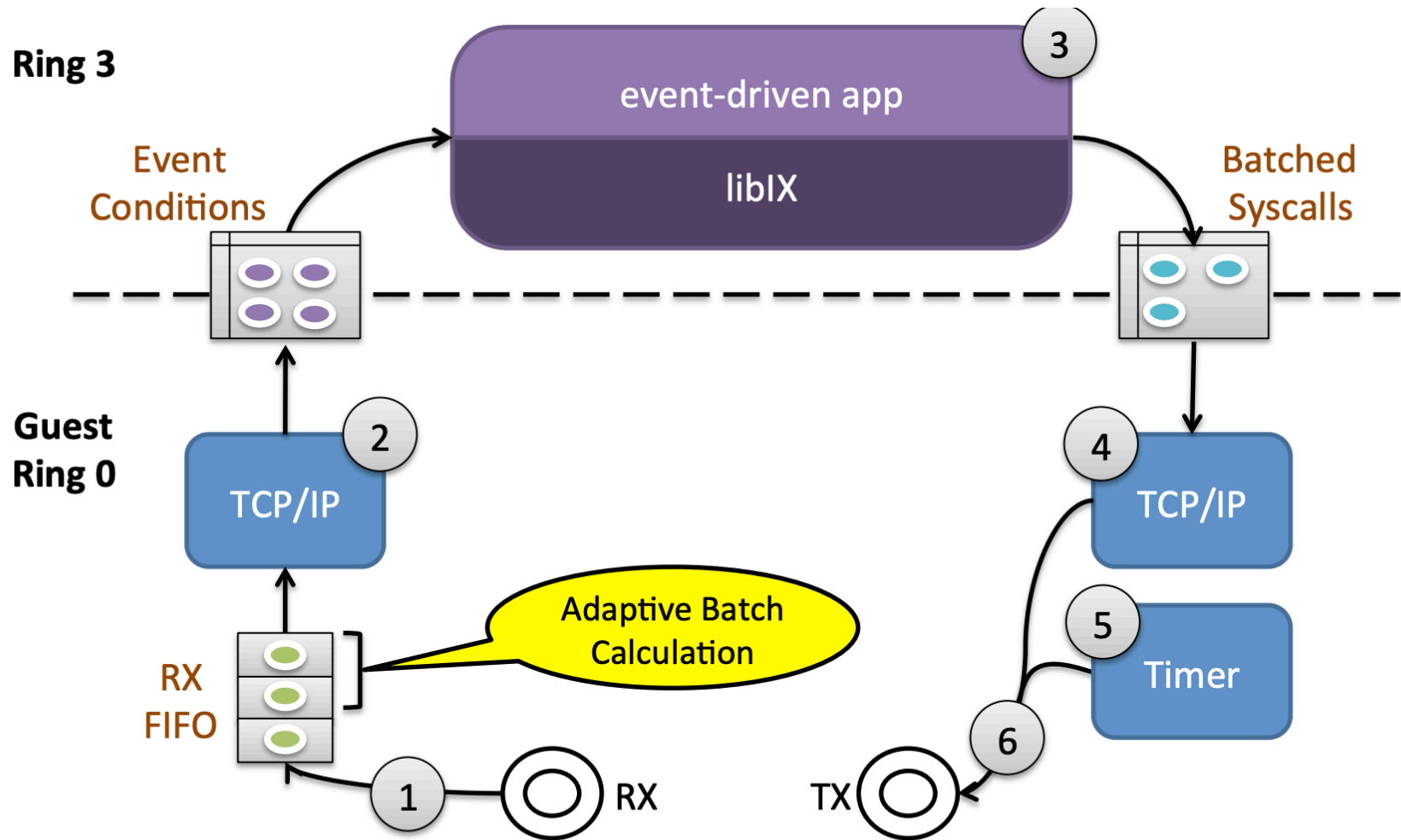


Design (I): Run to Completion



Improves Data-Cache Locality
Removes Scheduling Unpredictably

Design (2): Adaptive Batching



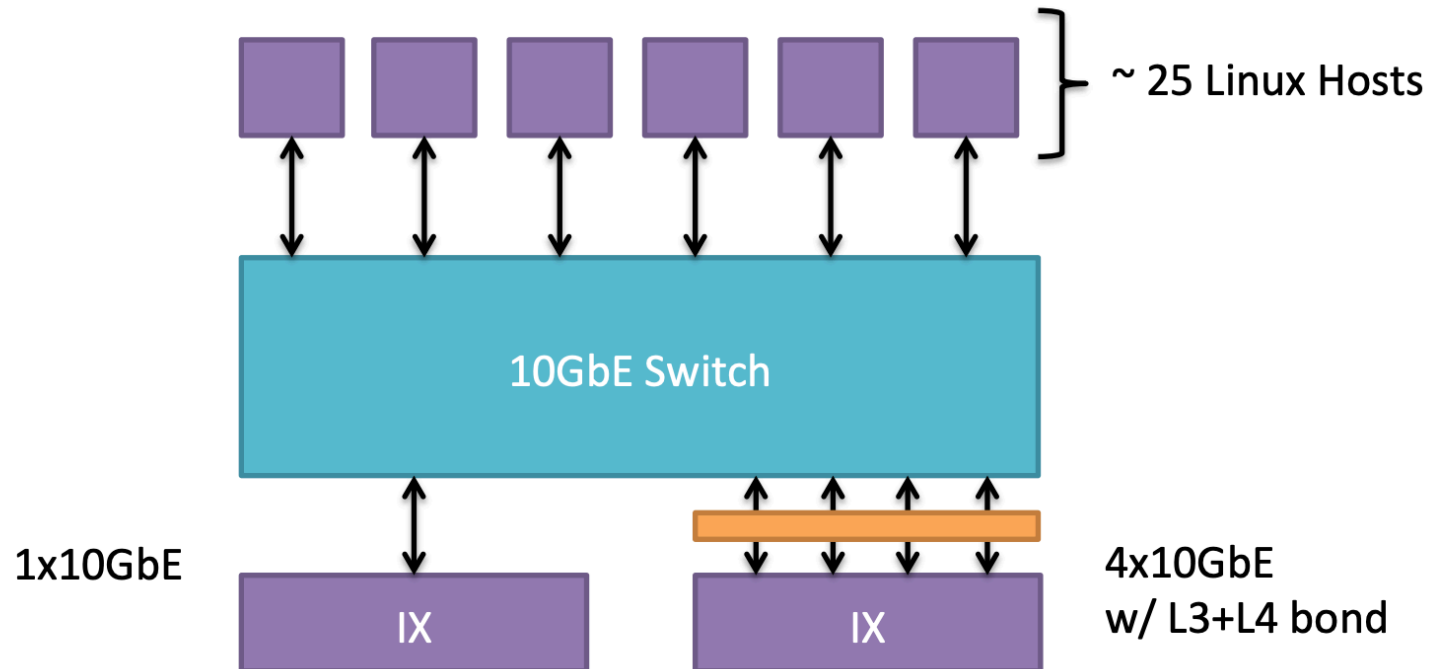
Improves Instruction-Cache Locality and Prefetching 17

Other Aspects

- Design (3): Flow consistent hashing
 - Synchronization & coherence free operation
- Design (4): Native zero-copy API
 - Flow control exposed to application
- Libix: Libevent-like event-based programming
- IX prototype implementation
 - Dune, DPDK, LWIP, ~40K SLOC of kernel code

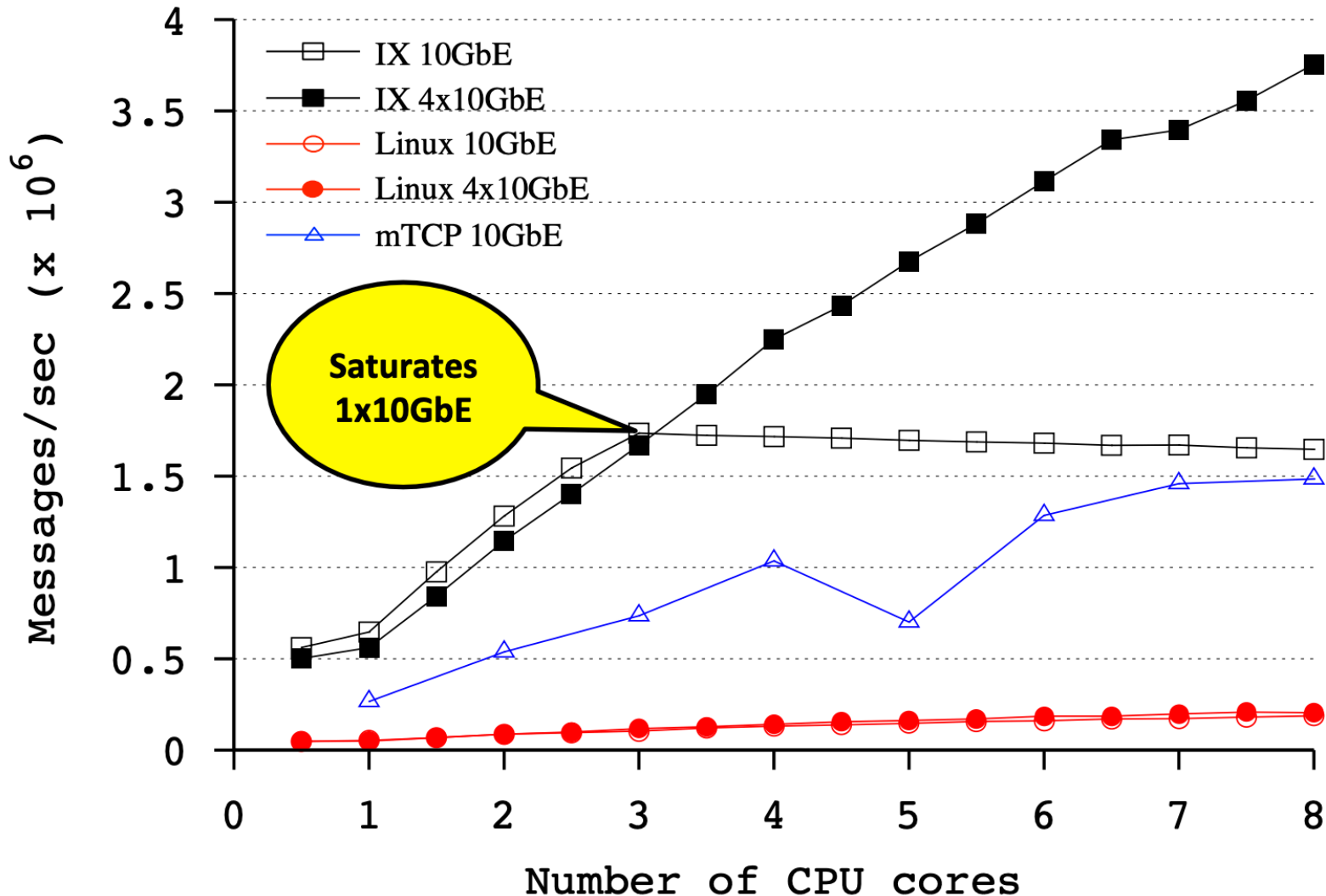
Evaluation

- Comparison IX to Linux and mTCP [NSDI '14]
- TCP microbenchmarks and Memcached



TCP Echo: Multicore Scalability for Short Connections

64 byte ping-pong messages per connection



What did you like about the paper?

Limitations?

Other subsequent work

- TAS (Kauffman et. al., Eurosys'19)
 - Differentiates between fast-path and slow-path processing
 - Leverages kernel-bypass using DPDK.
- Kernel-bypass CPU scheduling across applications
 - ZygOS (Prekas et. al, SOSP'27)
 - Shenango (Ousterhout et. al., NSDI'19)
 - Shinjuku (Kaffes et. al., NSDI'19)
 - Caladan (Fried et. al., OSDI'20)
 - **SNAP (Google, SOSP'19)**
 -

Kernel bypass vs more efficient kernel processing?