Kernel Bypass

ECE/CS598HPN

Radhika Mittal

Performance overheads in kernel stack

Performance overheads in kernel stack

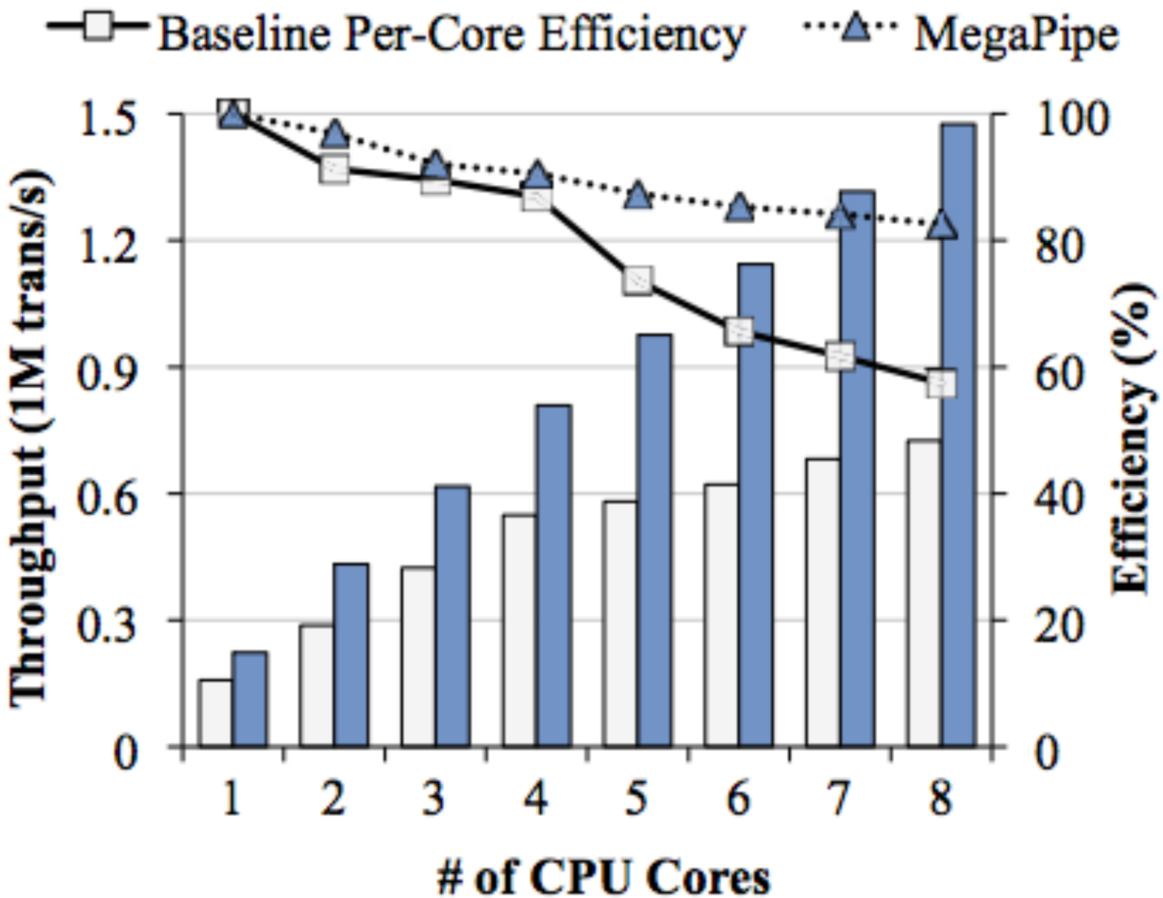
- Shared listening socket.
- Lack of connection affinity.
- System calls (context switching)
- Shared file descriptor space, heavy file descriptors
- Interrupts
- Extra copy and buffering
- Heavy-weight data structures (sk_buff)
- Queuing delays
- CPU scheduling delays.
- Inefficient processing.

Performance overheads in kernel stack

- *Shared listening socket.*
- *Lack of connection affinity.*
- *System calls (context switching).*
- *Shared file descriptor space, heavy file descriptors.*
- Interrupts.
- Extra copy and buffering.
- Heavy-weight data structures (sk_buff).
- Queuing delays.
- CPU scheduling delays.
- Inefficient processing.

Somewhat addressed by MegaPipe

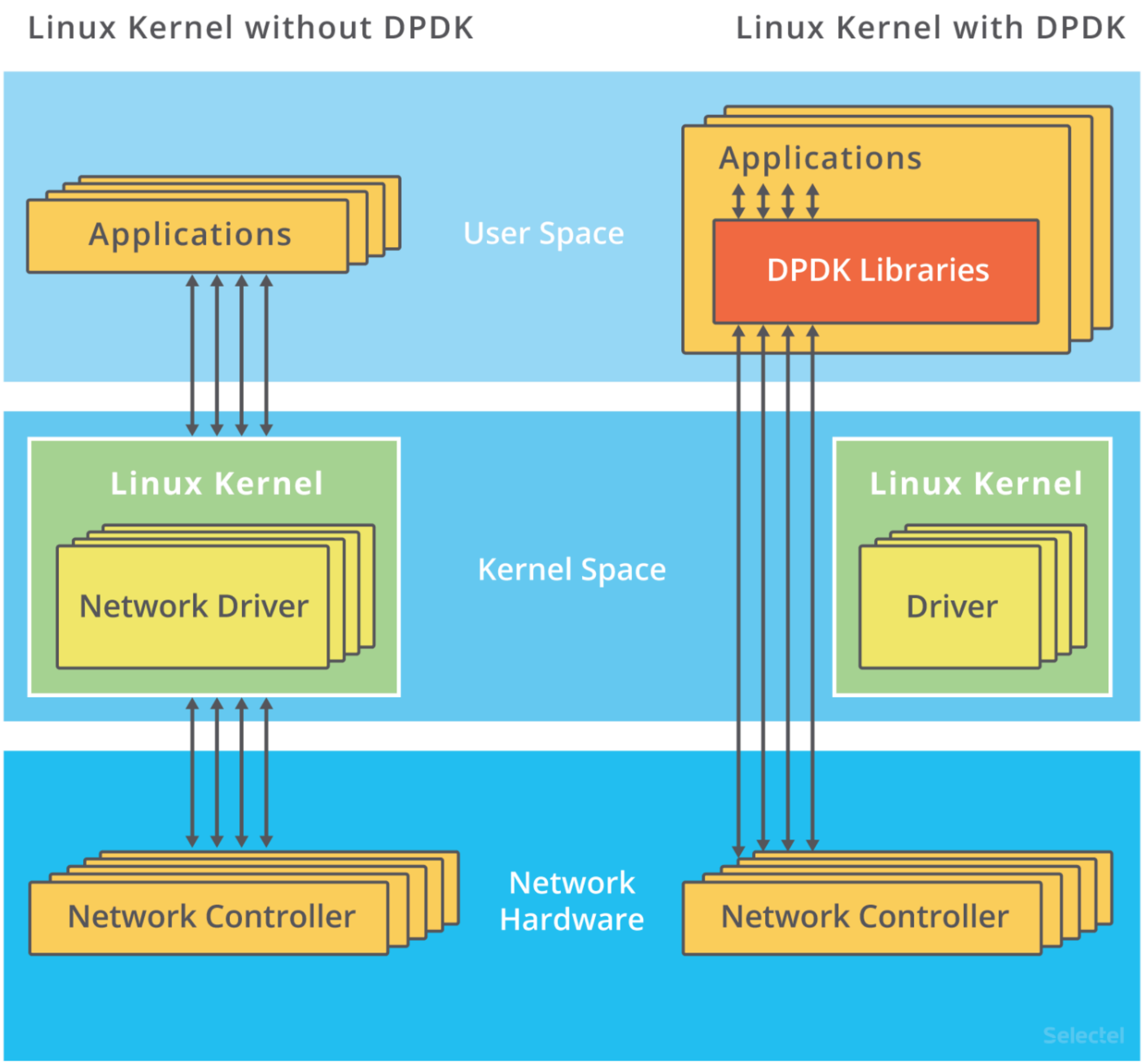
MegaPipe Performace



80% CPU cycles spent in packet processing in kernel.

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
  - In-kernel module for efficient packet processing.
  - Light-weight packet buffers.
  - Fewer memory copies.
  - *Possibly interrupt-driven.*

- Packet Shader
  - Modified packet I/O engine in the kernel.
  - Fetches packets through a combination of interrupts and polling.
  - Processes packets using GPU in userspace.

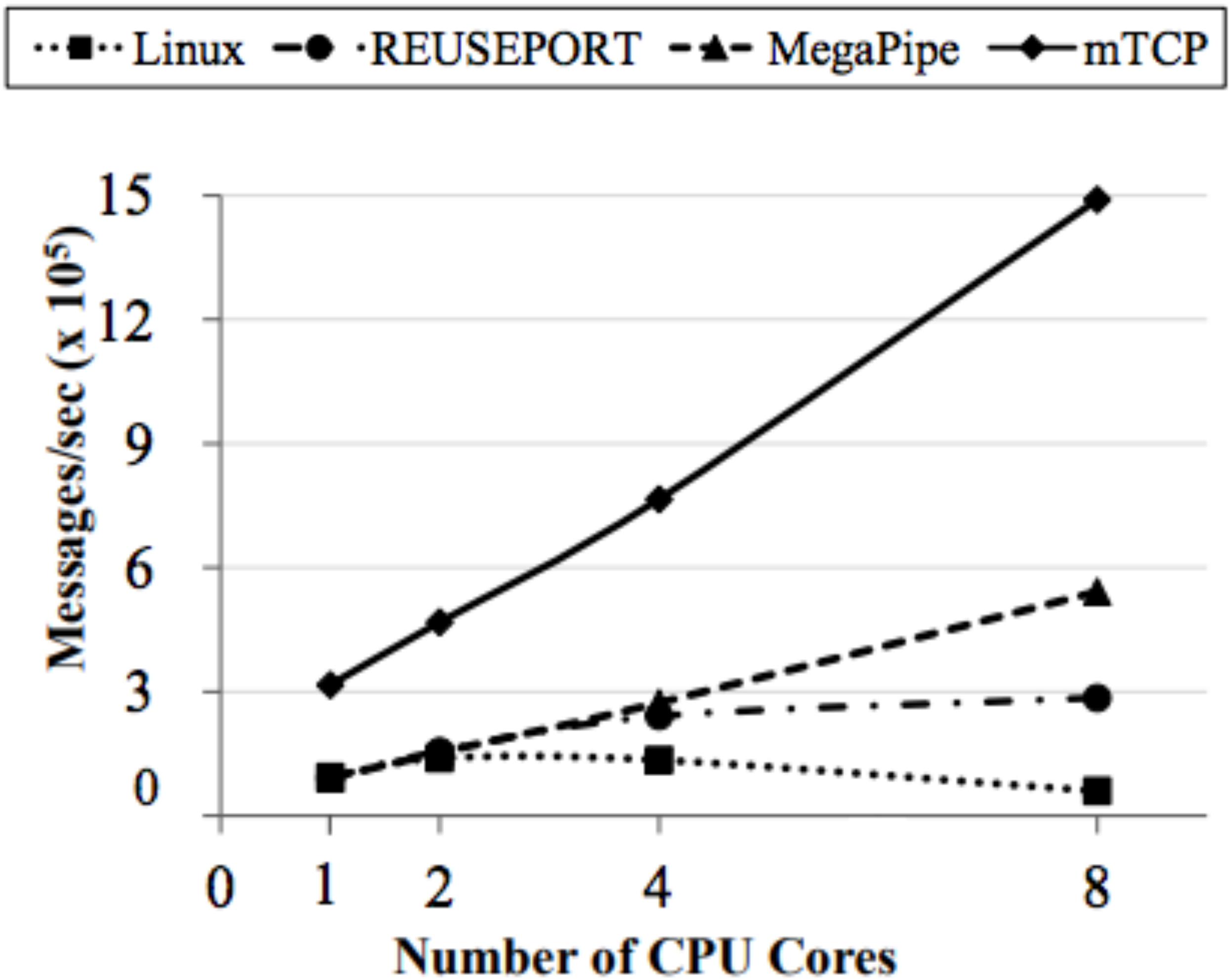
Kernel Bypass Packet I/O Engine

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

mTCP

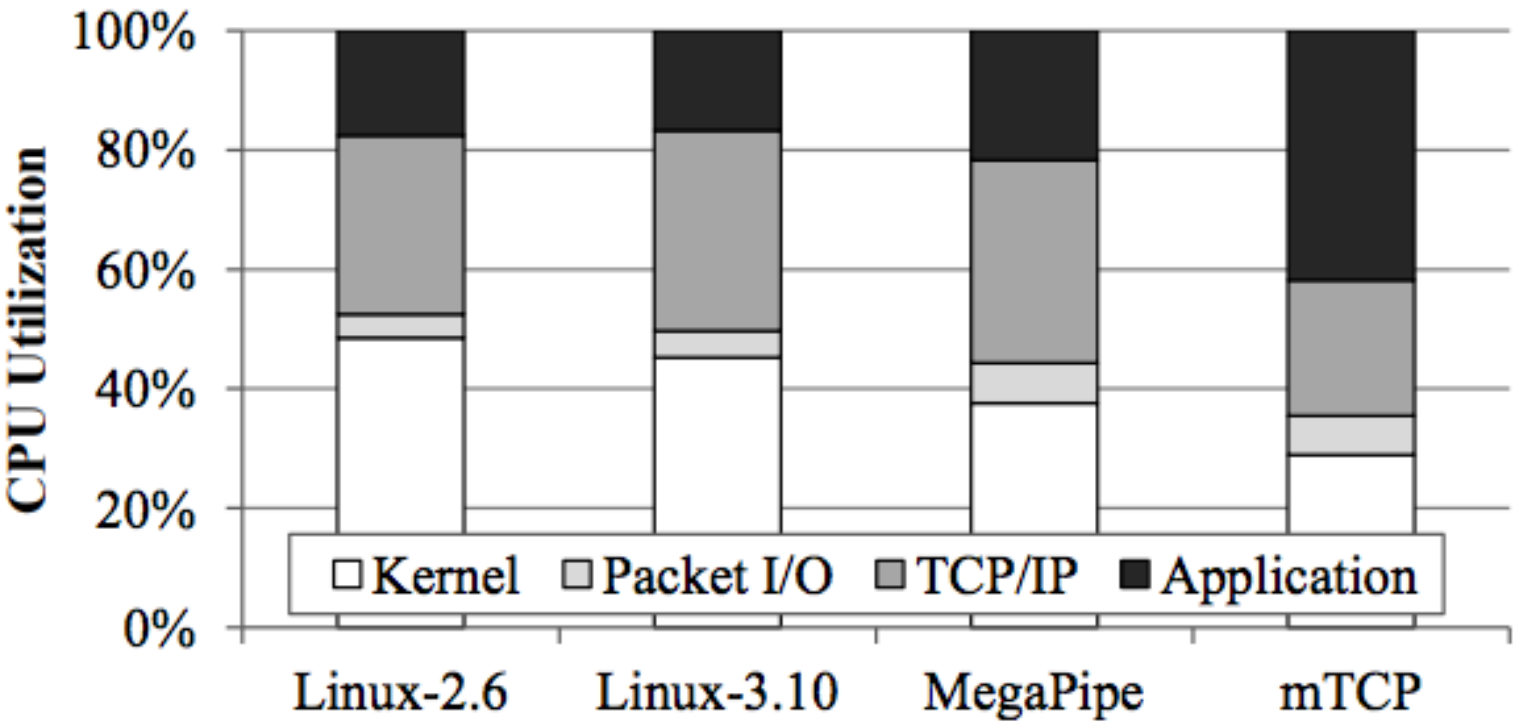
- User-space TCP/IP stack built over kernel-bypass packet I/O engines.
    - Implementation in paper over PacketShader.
    - DPDK based implementation also available.

mTCP



Linux
REUSEPORT
MegaPipe
mTCP
Messages/sec (x 10^5)
15
12
9
6
3
0
0
1
2
4
8
Number of CPU Cores

mTCP



100%
80%
60%
40%
20%
0%
CPU Utilization
Kernel
Packet I/O
TCP/IP
Application
Linux-2.6
Linux-3.10
MegaPipe
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

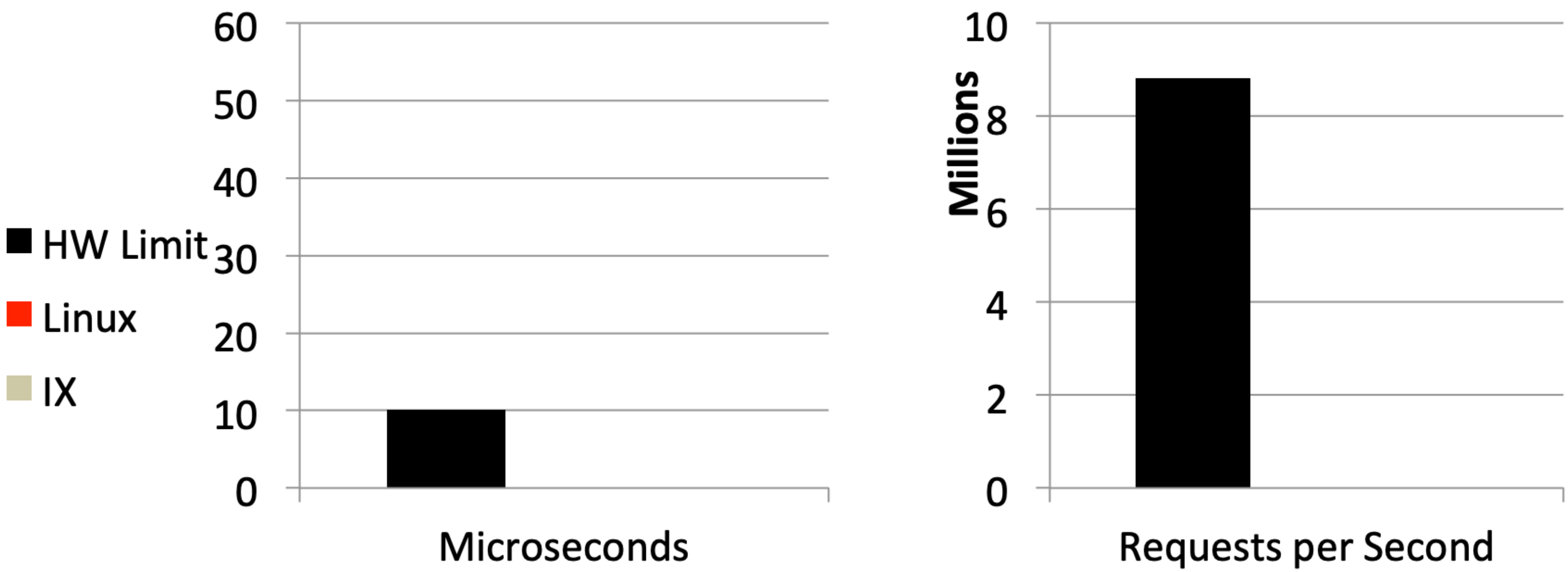
Adam Belay, George Prekas, Ana Klimovic,
Samuel Grossman, Christos Kozyrakis, Edouard Bugnion

OSDI'14 (Best Paper)

Slides borrowed from Adam's OSDI talk.

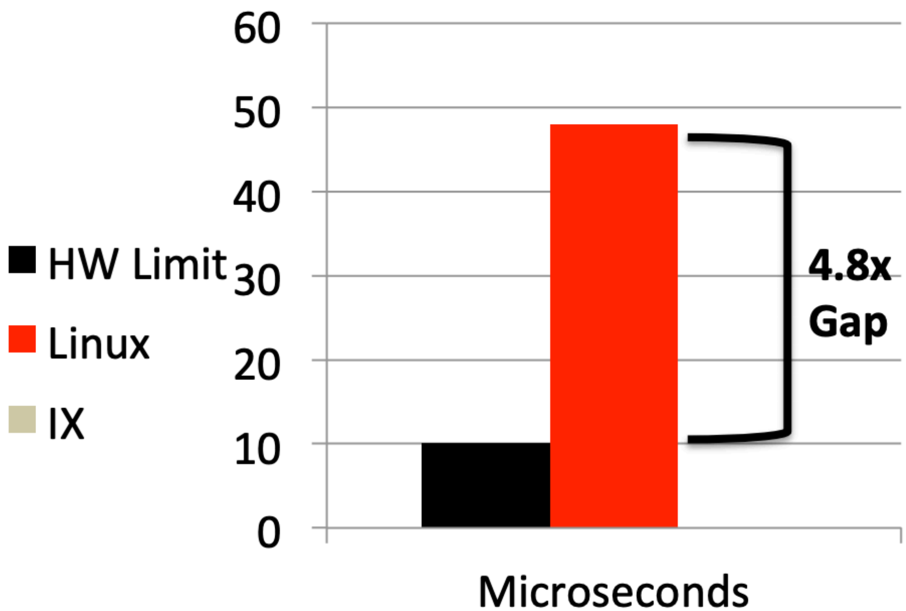
HW is fast

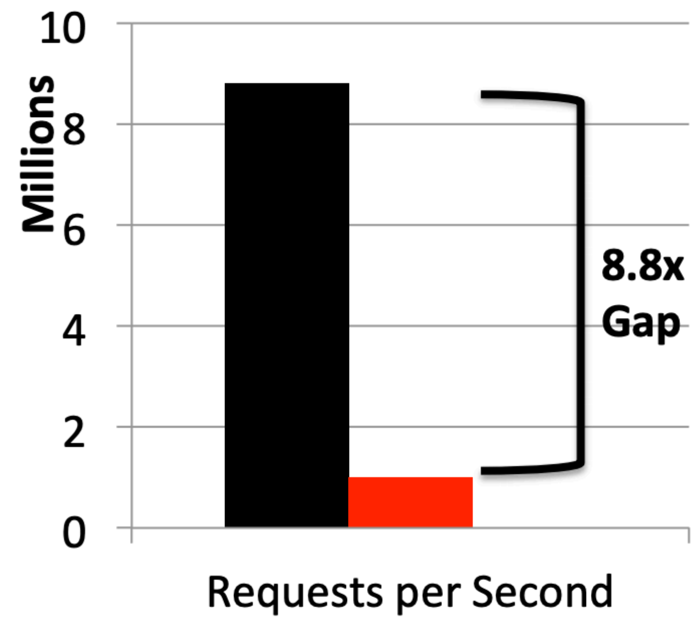
64-byte TCP Echo:



HW is fast, but SW is the bottleneck

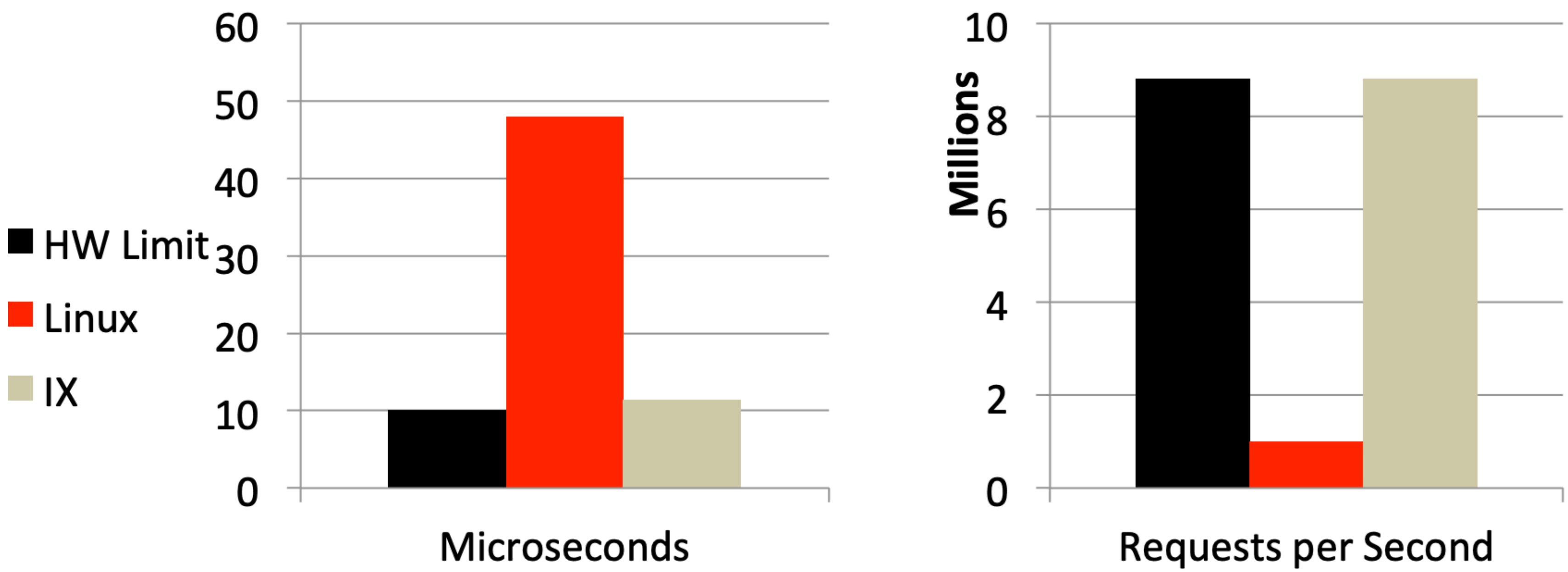
64-byte TCP Echo:



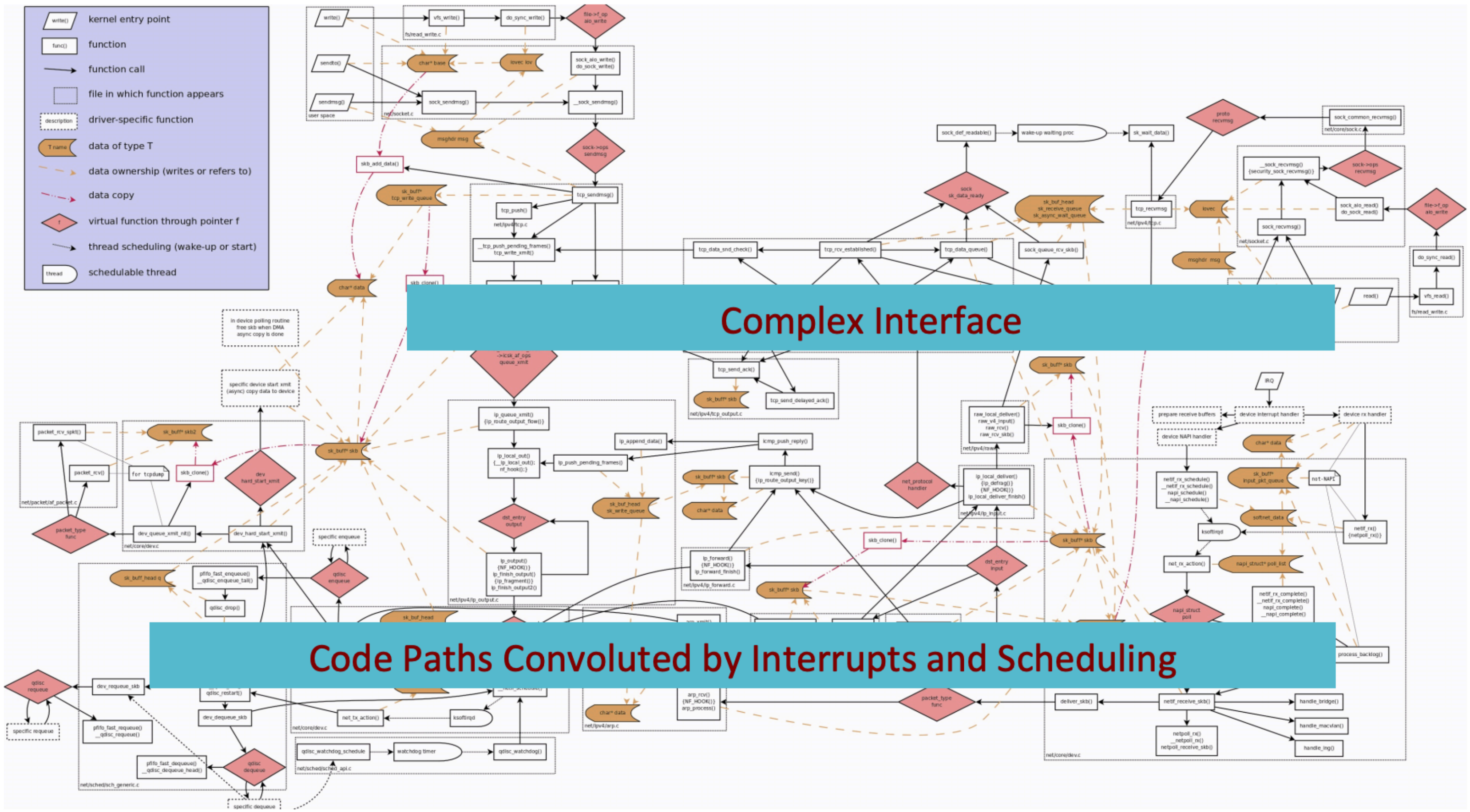


IX closes the SW performance gap

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!

Alternatives

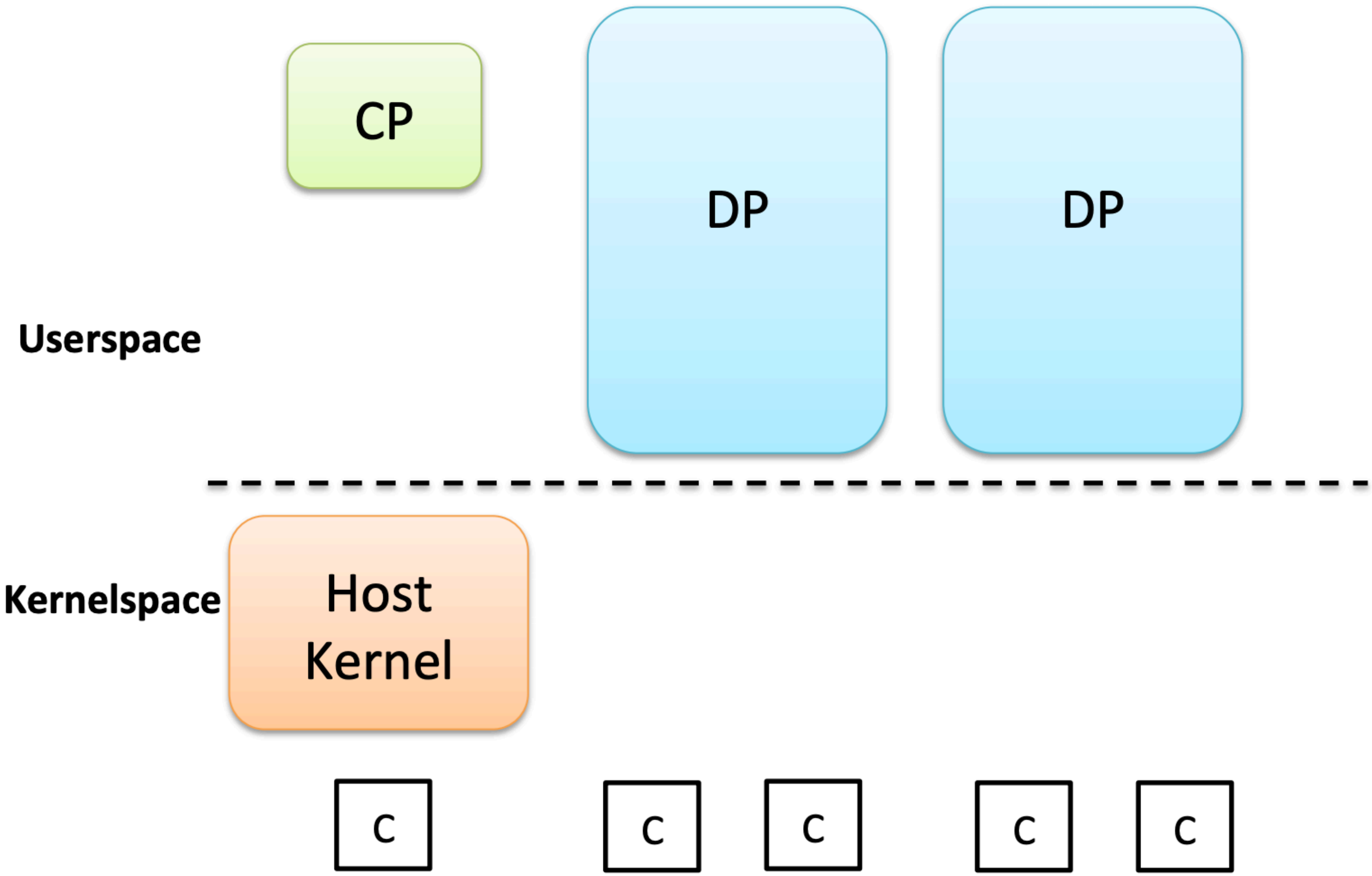
- Kernel-bypass user-space stacks (e.g. mTCP)
  - Lack of protection between app and network stack.
- Hardware support:
  - TCP Offload Engines (TOE)
  - RDMA

IX Key Design Decisions

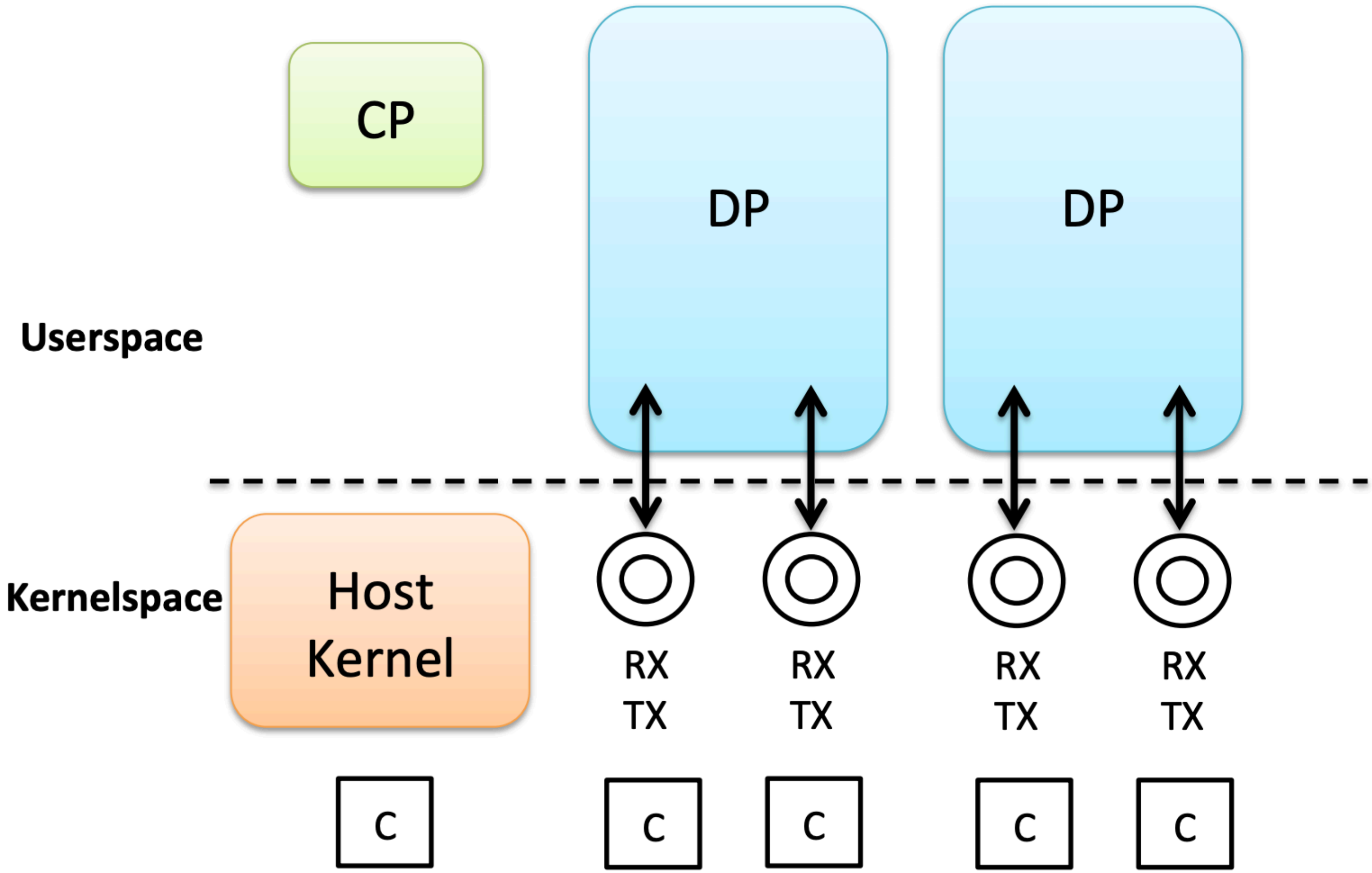
IX Key Design Decisions

- Separation of control plane and dataplane
  - Control plane handles resource allocation.
- Run to completion packet processing.
  - Adaptive Batching
  - Zero-copy
  - Synchronization-free processing

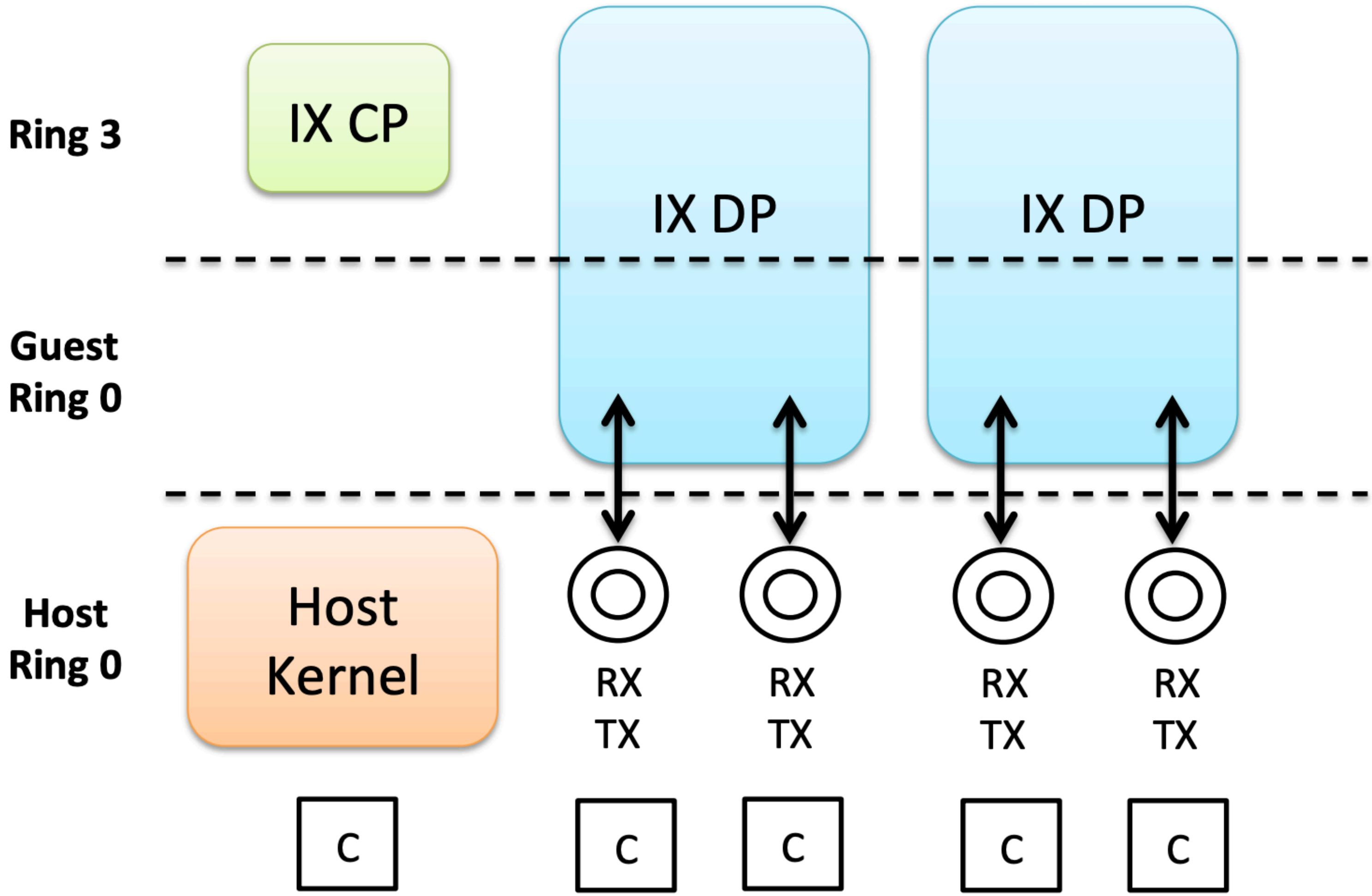
IX: Separation of Control and Data Plane



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.

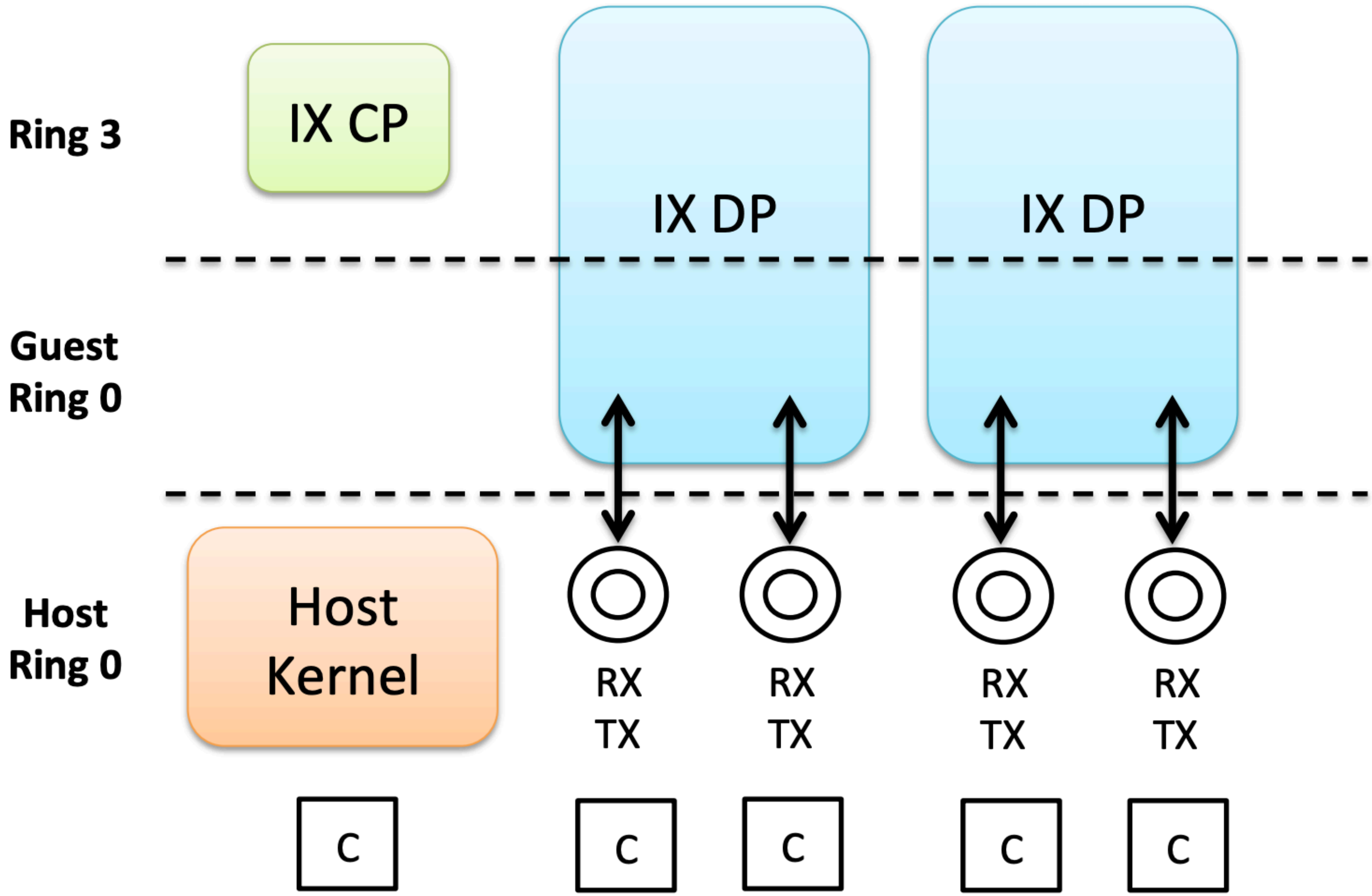
Detour: what is virtualization?

- Trick a guest OS into believing it has direct access to hardware (CPU, NIC, etc).

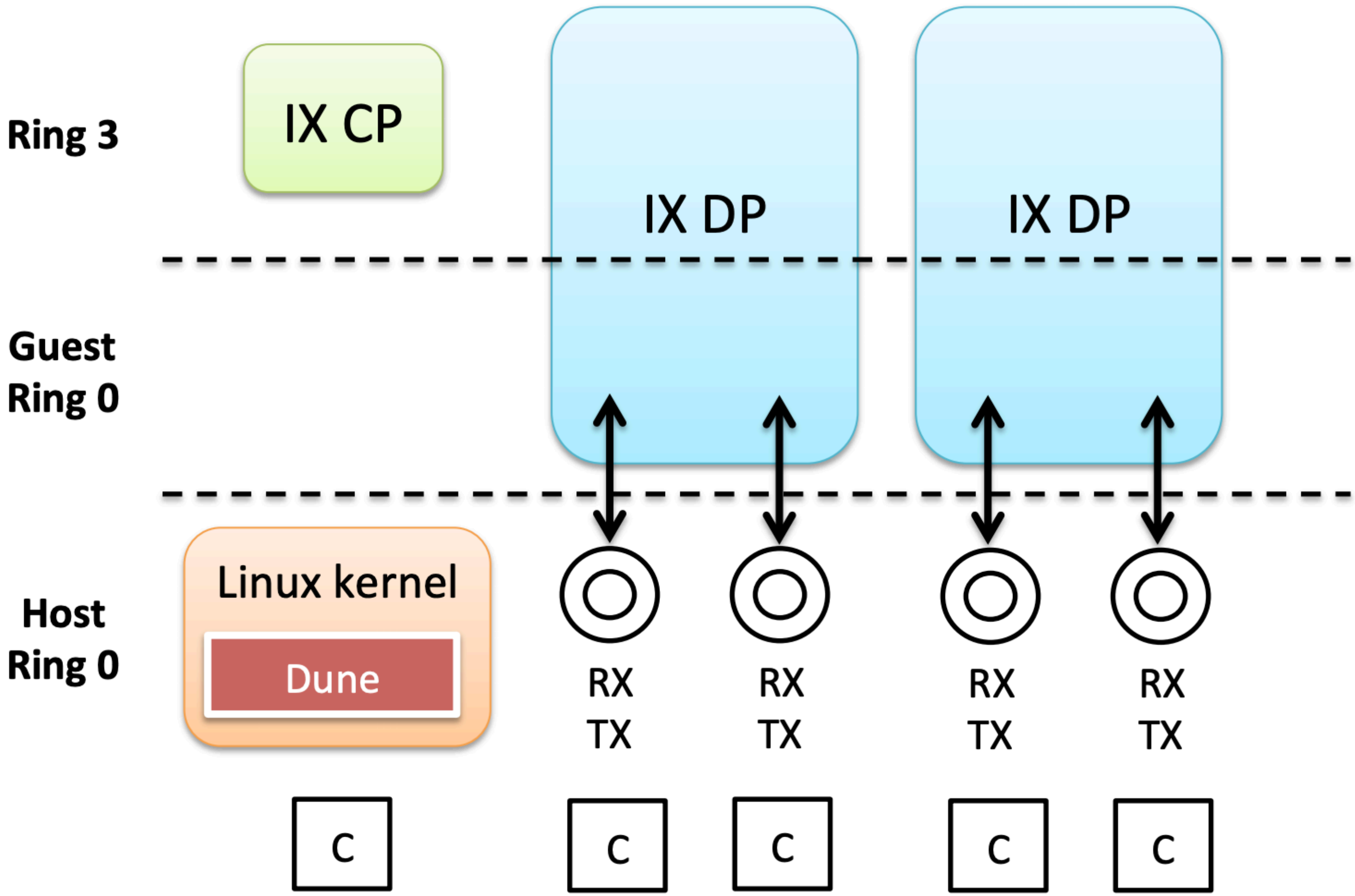
- Hypervisor or Virtual Machine Monitor (VMM) controls the guest VM's access, provides isolation, etc.

- 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)

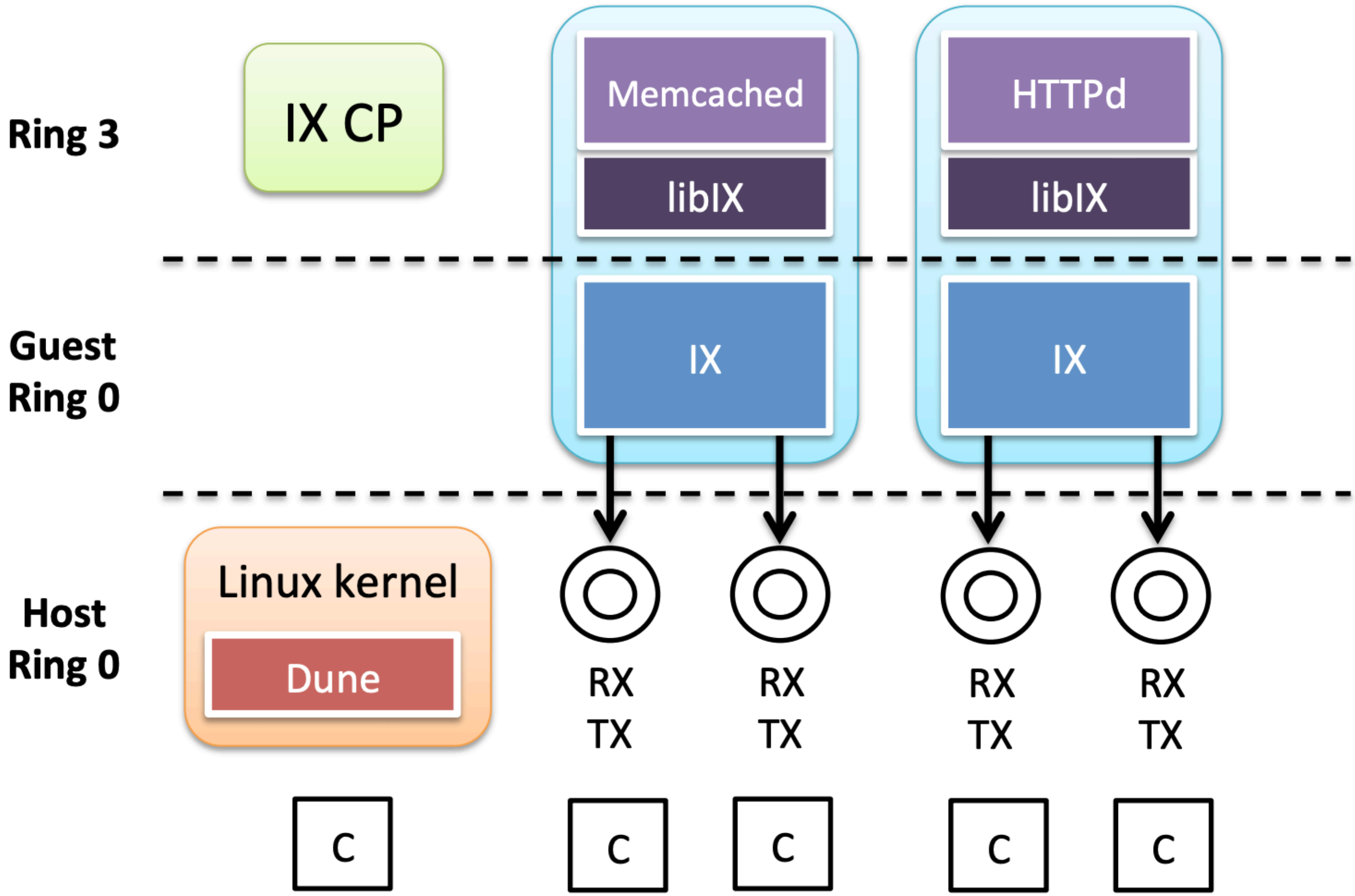
IX: Separation of Control and Data Plane



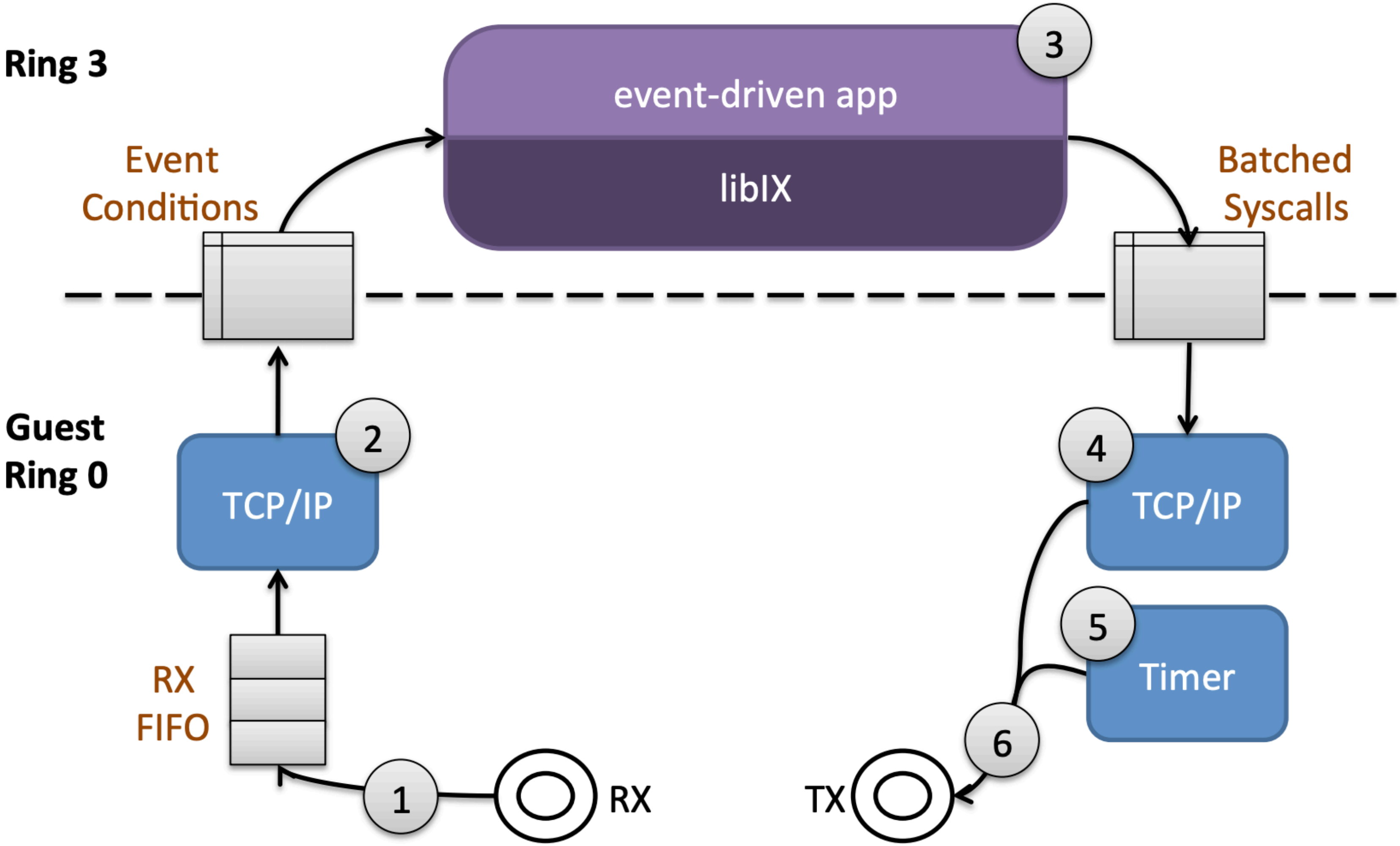
IX: Separation of Control and Data Plane



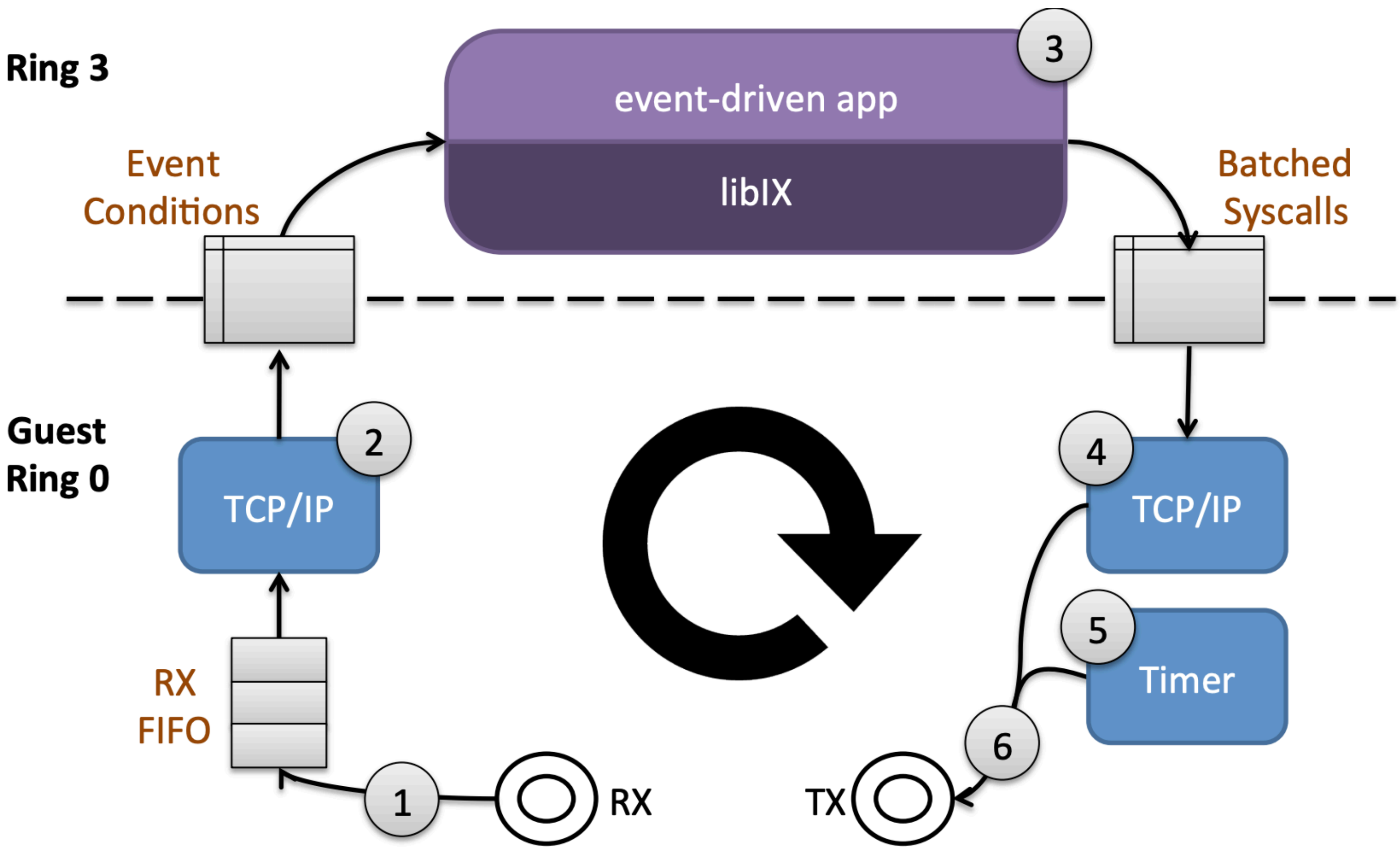
IX: Separation of Control and Data Plane



IX Execution Pipeline



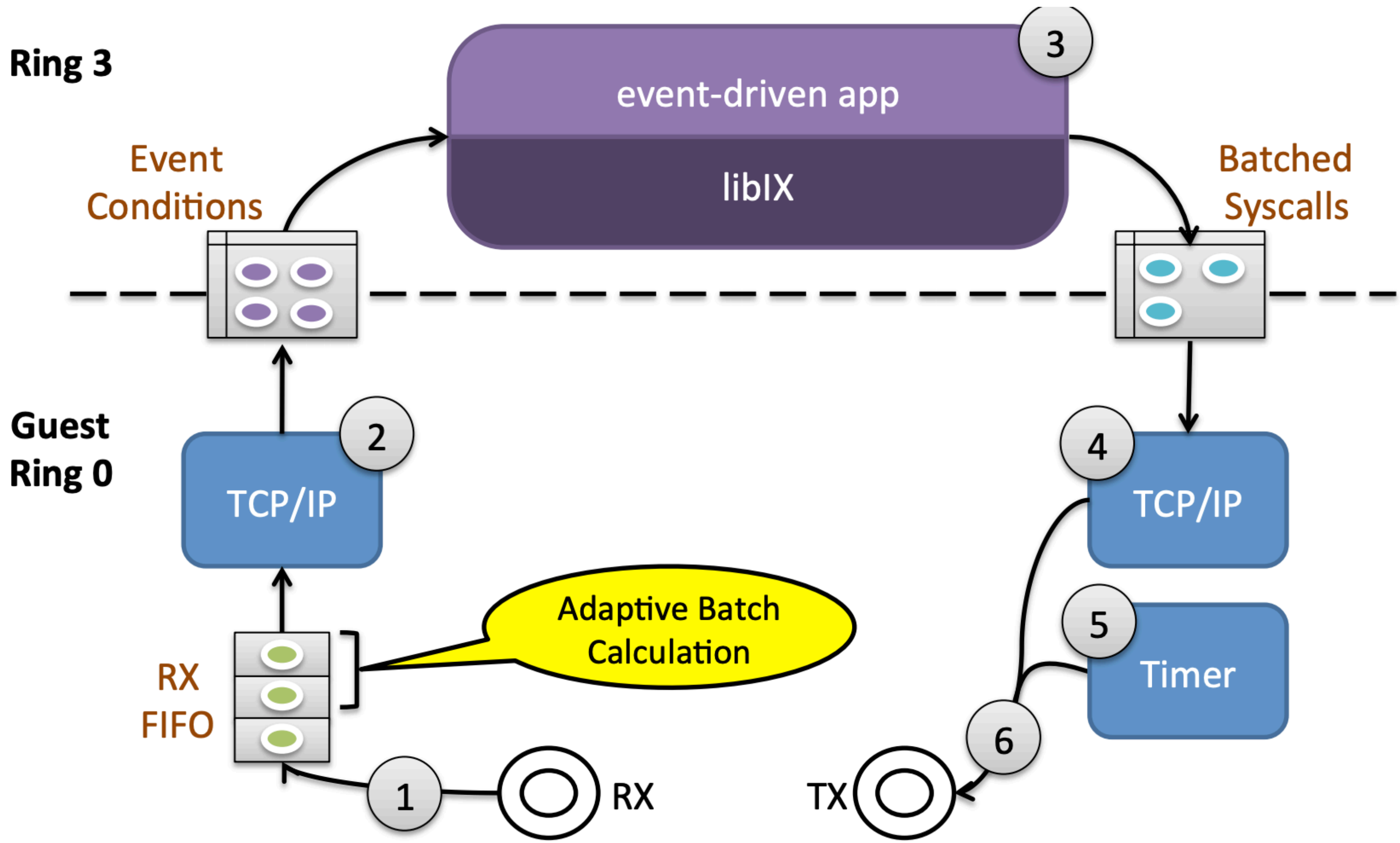
Design (1): Run to Completion



Improves Data-Cache Locality

Removes Scheduling Unpredictably

Design (2): Adaptive Batching



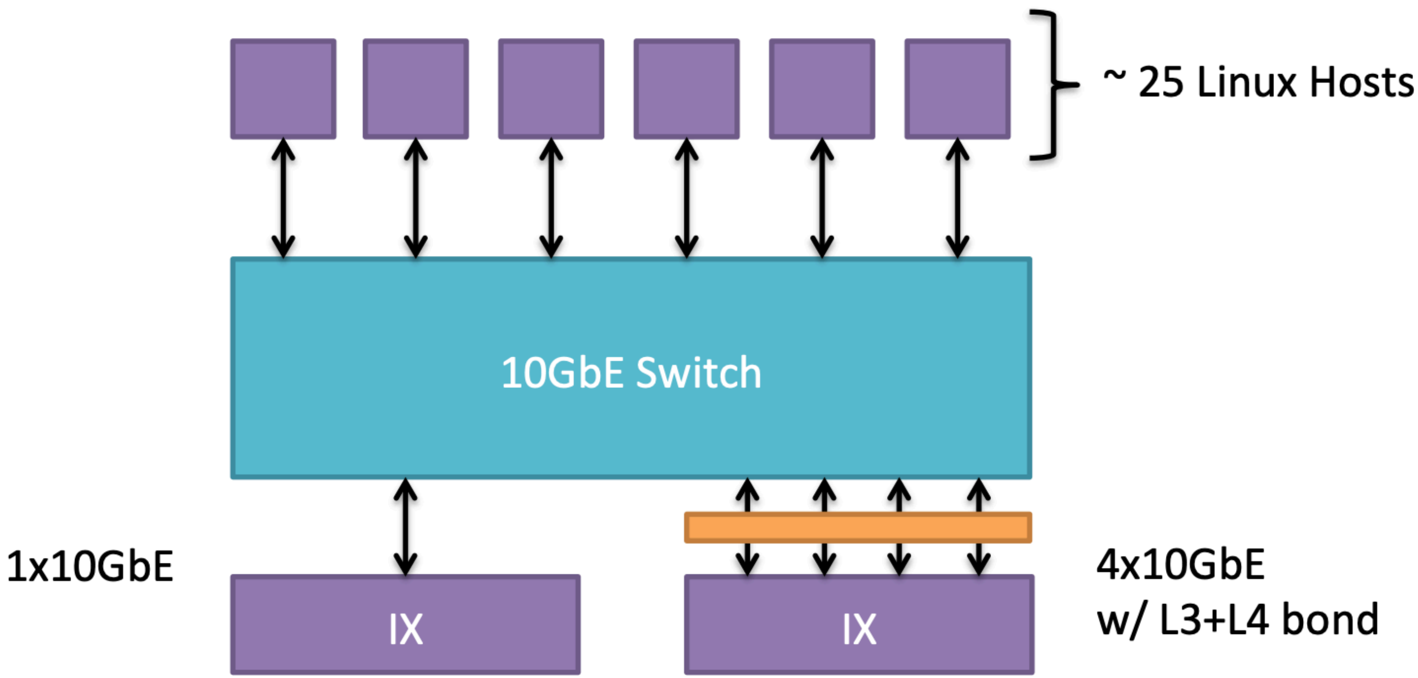
Improves Instruction-Cache Locality and Prefetching

Other Aspects

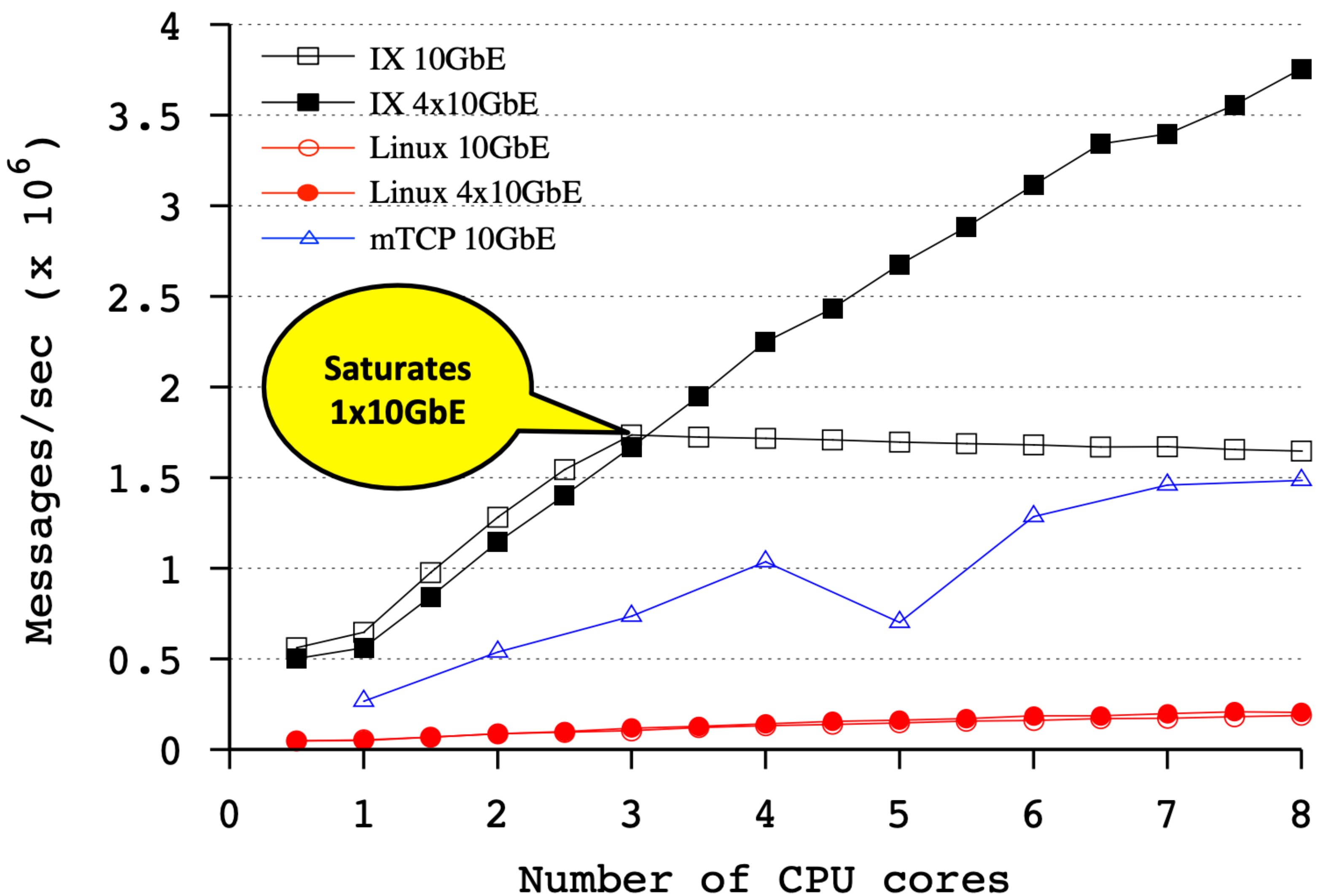
- Design (3): Flow consistent hashing
  - Synchronization & coherence free operation
- Design (4): Native zero-copy API
  - Flow control exposed to application
- Libix: Libevent-like event-based programming
- IX prototype implementation
  - Dune, DPDK, LWIP, ~40K SLOC of kernel code

Evaluation

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



TCP Echo: Multicore Scalability for Short Connections



Your Opinions

- Pros
  - Protection, along with low latency and high throughput.
    - Run-to-completion
    - Adaptive batching
    - Synchronization-free processing
    - Zero-copy
  - Innovative reuse the idea of control and data plane separation.

Your Opinions

- Cons
  - Possibility of internal memory fragmentation.
  - How are batching bounds determined?
  - Performance comparison with RDMA?
  - Insufficient details about control plane.
  - Needs new API
  - Certain assumptions from the application:
    - What if application is too slow?
  - Co-existence with conventional VM settings.

Your Opinions

- Ideas
  - Resource allocation policies!
  - Redo evaluation after super-optimizing Linux.
  - Better memory management.
  - Use similar hardware virtualization techniques for other usecases.
  - Are there any backwards compatible solution to the problem?

Next Class: RDMA

- FaRM: heavy on systems concepts.
- IRN: heavy on networking concepts.