I/O is Faster than the CPU
Let's Partition Resources and Eliminate (Most) OS Abstractions

Pekka Enberg, Ashwin Rao, and Sasu Tarkoma
University of Helsinki

HotOS '19

Introduction

- OS abstractions limit **application-level parallelism** and hold back **performance** of today’s **fast I/O devices.**
- **Parakernel** = partition hardware resources and minimise OS participation in data plane operations

Why is the OS a bottleneck now?

NICs are faster than the CPU

- For example, a 40 GbE NIC receives a cache-line sized packet faster than the CPU accesses its last-level cache (12 ns vs 15 ns).
- 400 GbE NICs on the horizon
- The time budget to process a packet is shrinking radically and parallelisation is critical.

Kaufmann *et al*. 2016. High Performance Packet Processing with FlexNIC. ASPLOS ’16

Persistent storage near the speed of DRAM

- Kernel I/O interfaces designed for storage significantly slower than DRAM.

- NVMe access latency 250x slower than DRAM (~20 µs vs 80 ns).

- Intel Optane access latency is 4x slower than DRAM (~80 ns vs. ~320 ns).

- Forces us to rethink OS abstractions and interfaces.

Honda et al. 2018. PASTE: A Network Programming Interface for Non-Volatile Main Memory. NSDI ’18
Intel® 64 and IA-32 Architectures Optimization Reference Manual (April 2019)
https://www.enterprisestorageforum.com/storage-hardware/nvme-speed.html

New application requirements

Predictable tail latency

- Large-scale parallelisation causes the tail latency of a single component to dominate.
- Hard to achieve with monolithic, shared-memory kernels:
    - Background tasks
    - Lack of partitioning
    - Synchronisation of shared state

Dean and Barroso. 2013. The Tail at Scale. CACM
Li *et al.* 2014. Tales of the Tail: Hardware, OS, and Application-level Sources of Tail Latency SOCC ’14

Security

- Monolithic kernels are inherently insecure because of their large trusted computing base (TCB).
- Eliminating kernel abstractions helps shrink TCB.

Biggs *et al.* 2018. The Jury Is In: Monolithic OS Design Is Flawed: Microkernel-based Designs Improve Security. APSys '18

Parakernels

Goals

- Improve application-level parallelism
- Unlock the speed of today’s fast I/O devices

Architecture

- Eliminate most kernel abstractions
- Partition hardware resources
- Minimise OS participation in data plane operations

Process abstraction

- A process in a parakernel is the kernel abstraction for:
  - Application-level parallelism
  - Isolation and multi-tenancy

Parallelism

- Applications must partition their data and work by process.
- Examples of partitioning today:
  - Thread-per-core servers (such as MICA and Seastar)
  - Single-threaded managed runtimes (such as Node.js)

Lim *et al.* 2014. MICA: a holistic approach to fast in-memory key-value storage. NSDI ’14
http://seastar.io/shared-nothing/

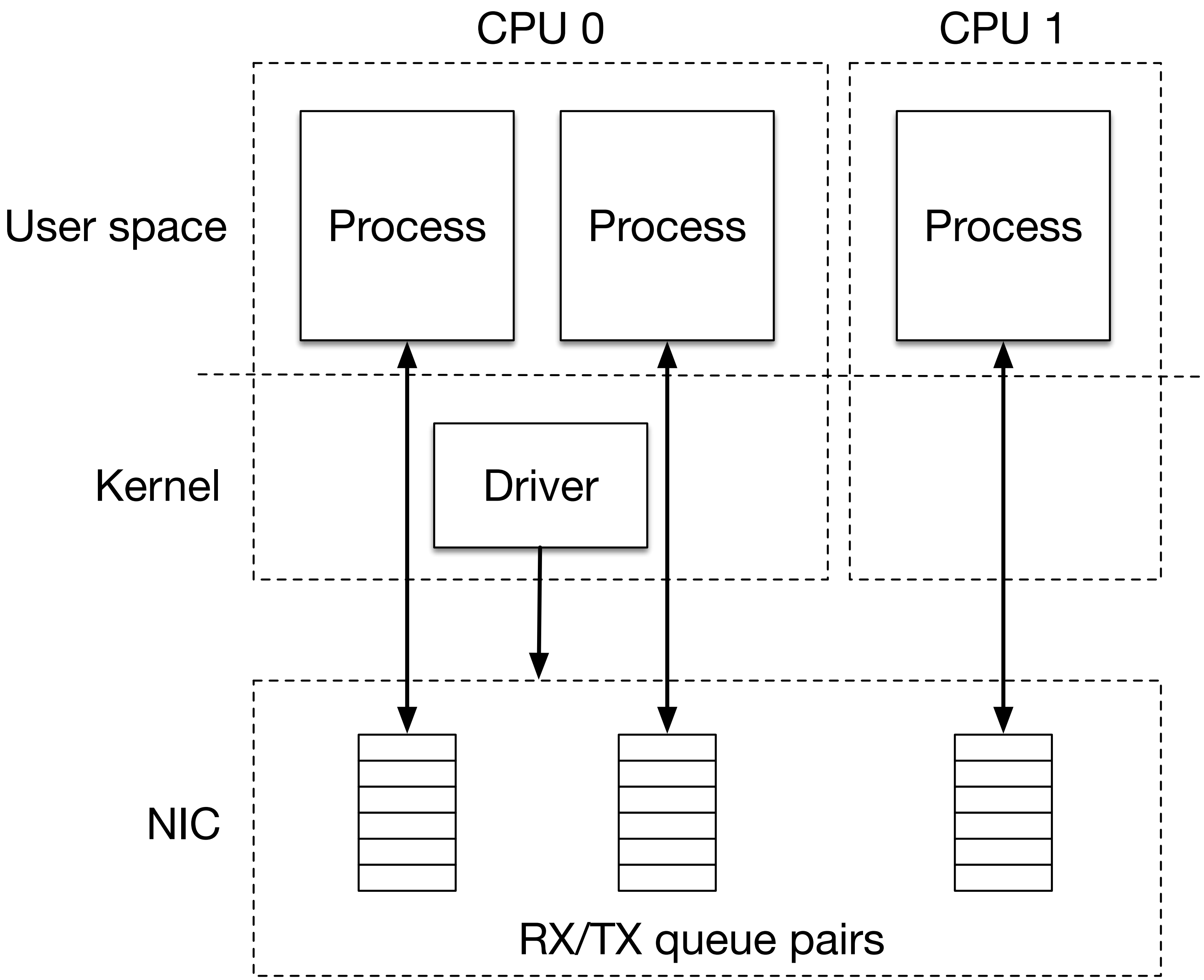
Concurrency

- Parakernel eliminates kernel threads because of their high overheads (context switch and synchronization).
- Parakernel provides non-blocking kernel interfaces.
- Concurrency is managed by user space:
    - Many abstractions available today: event callbacks, fibers, coroutines, and future/promise model.

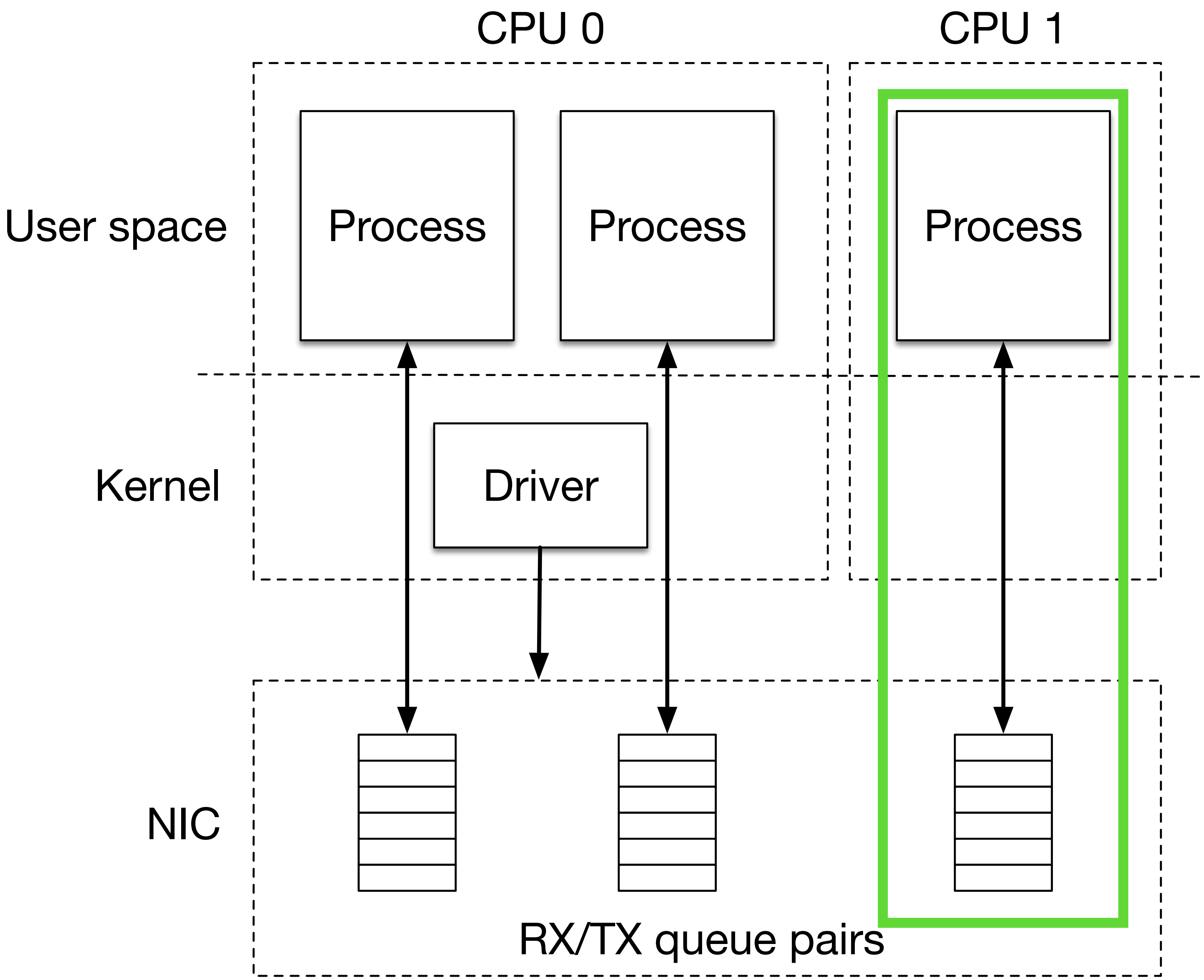
Partitioning

- Hardware resources are partitioned between processes to allow them to run independently.

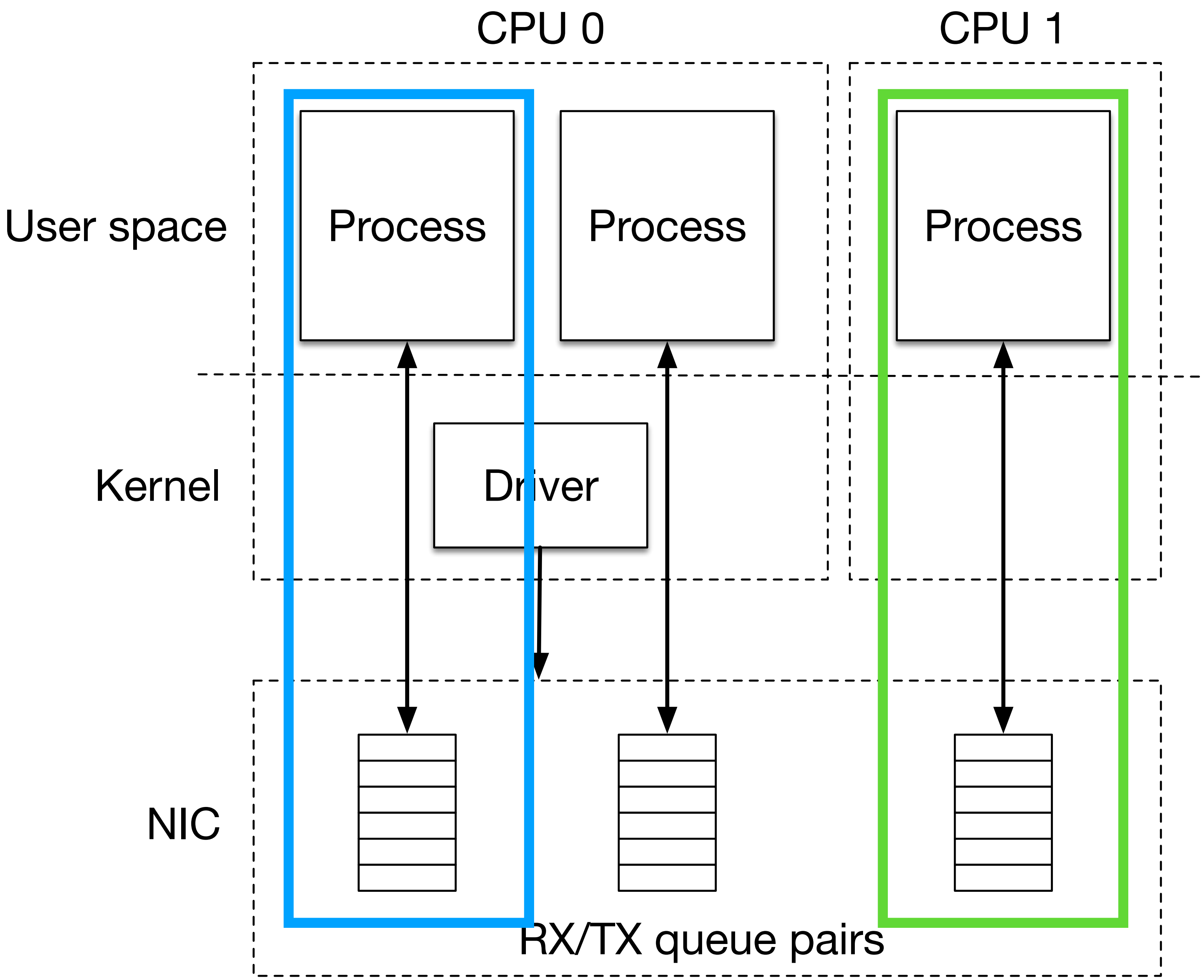
Partitioning



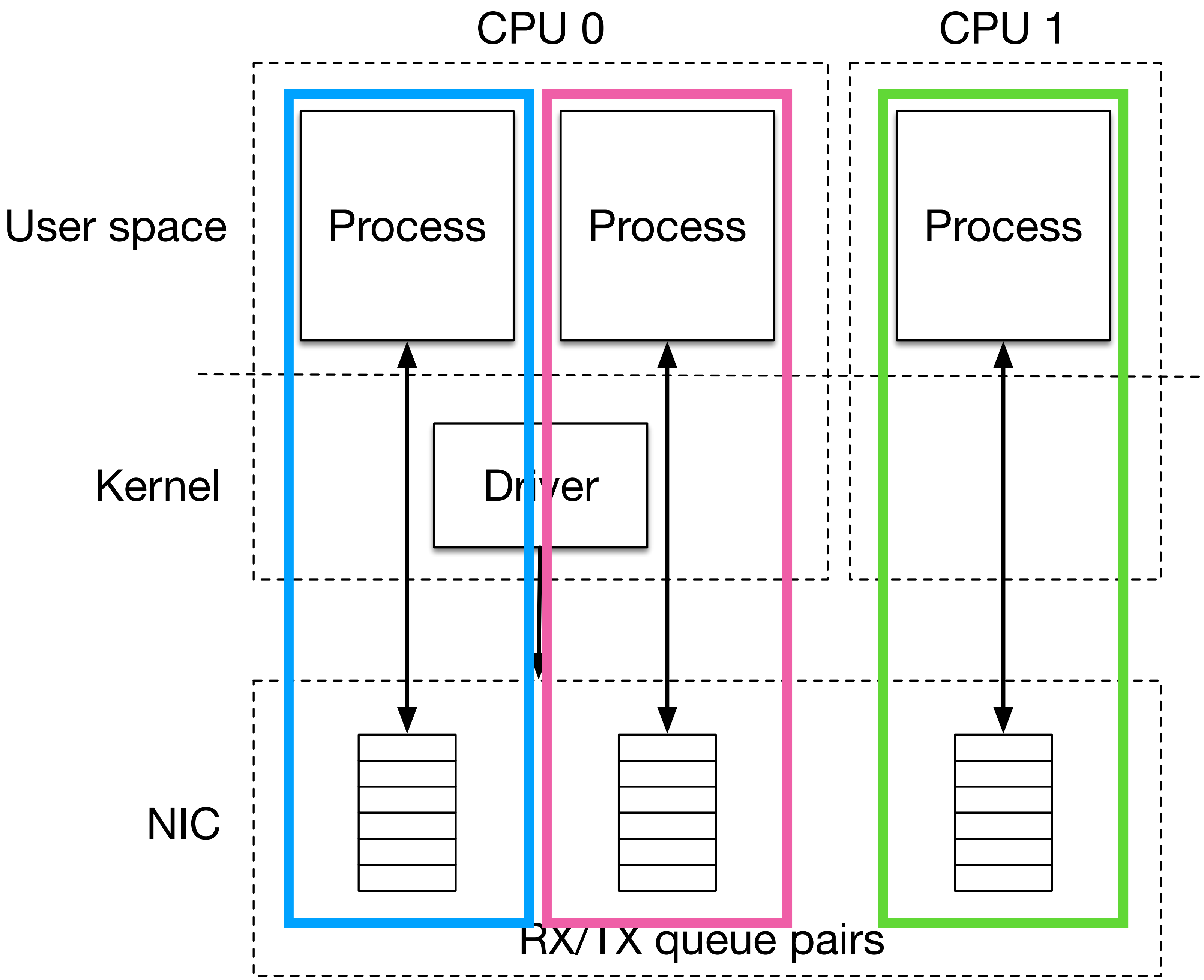
Partitioning



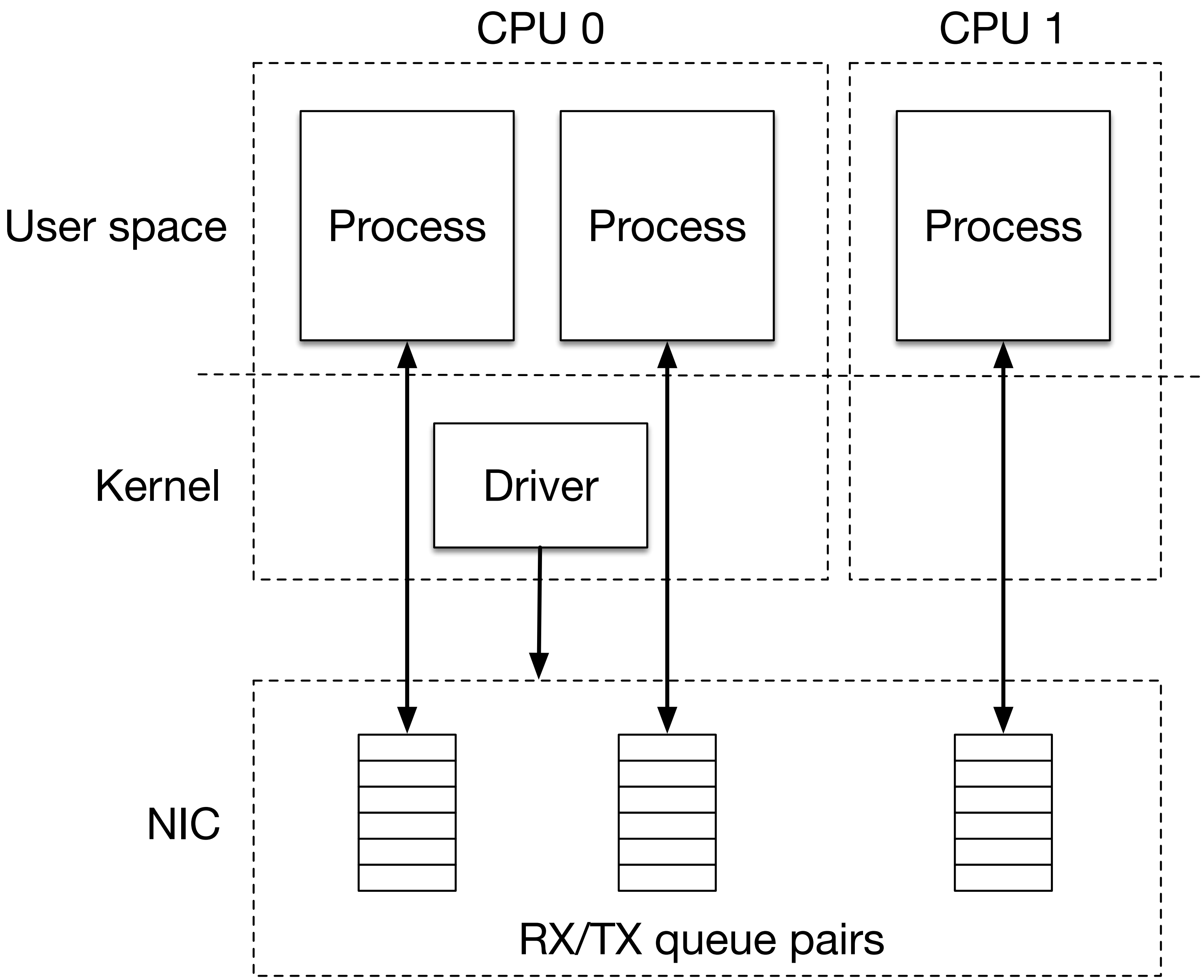
Partitioning



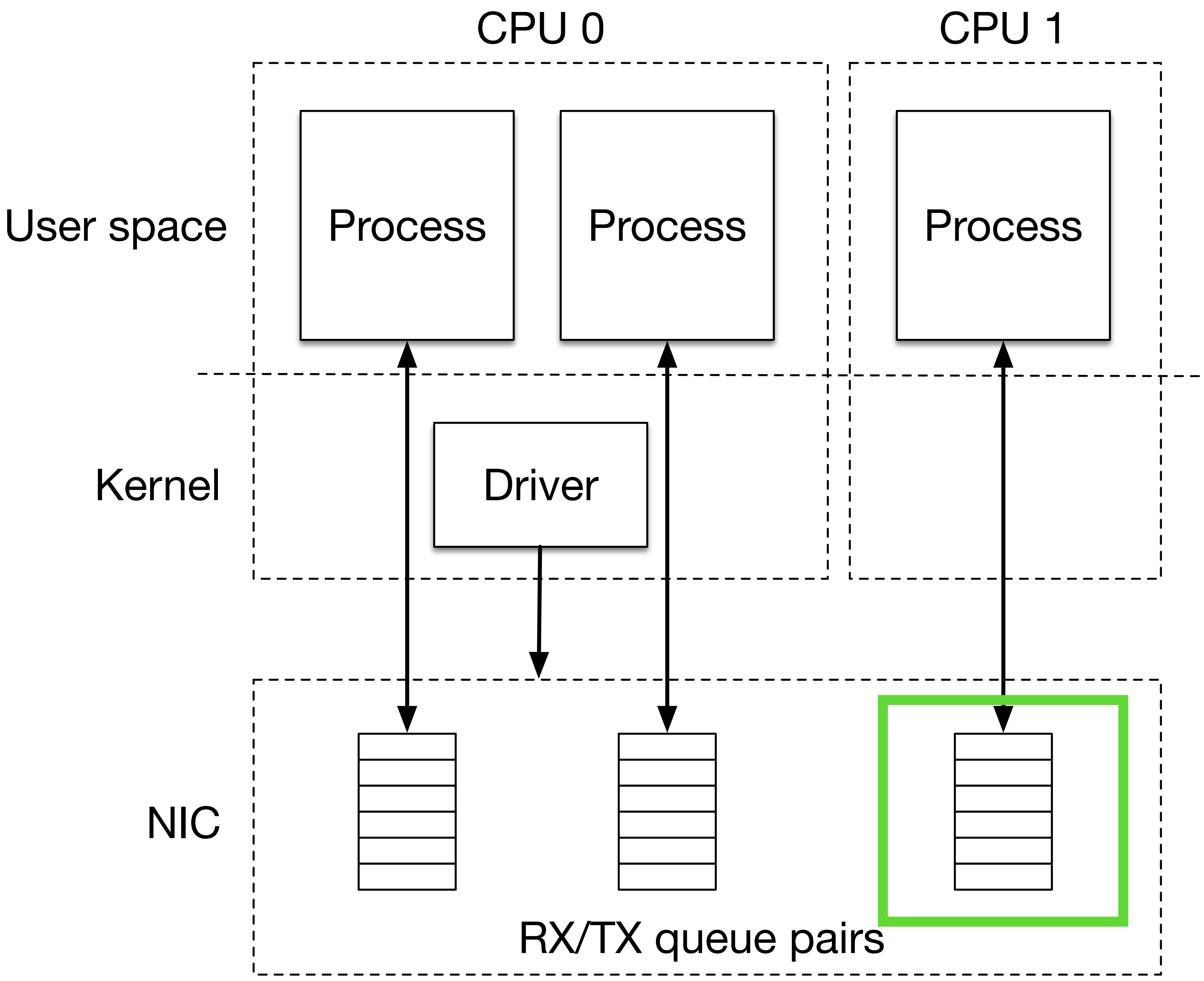
Partitioning



Example: NIC receive

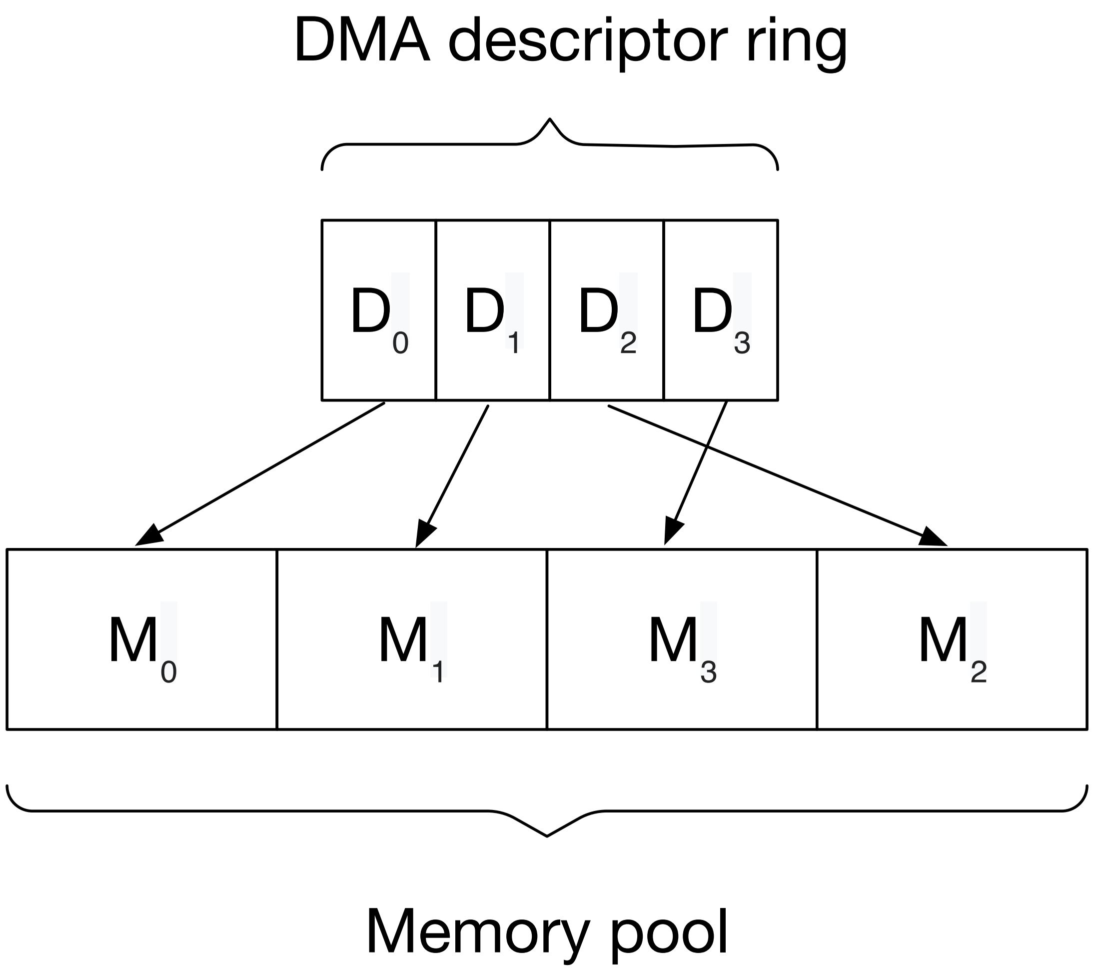


Example: NIC receive



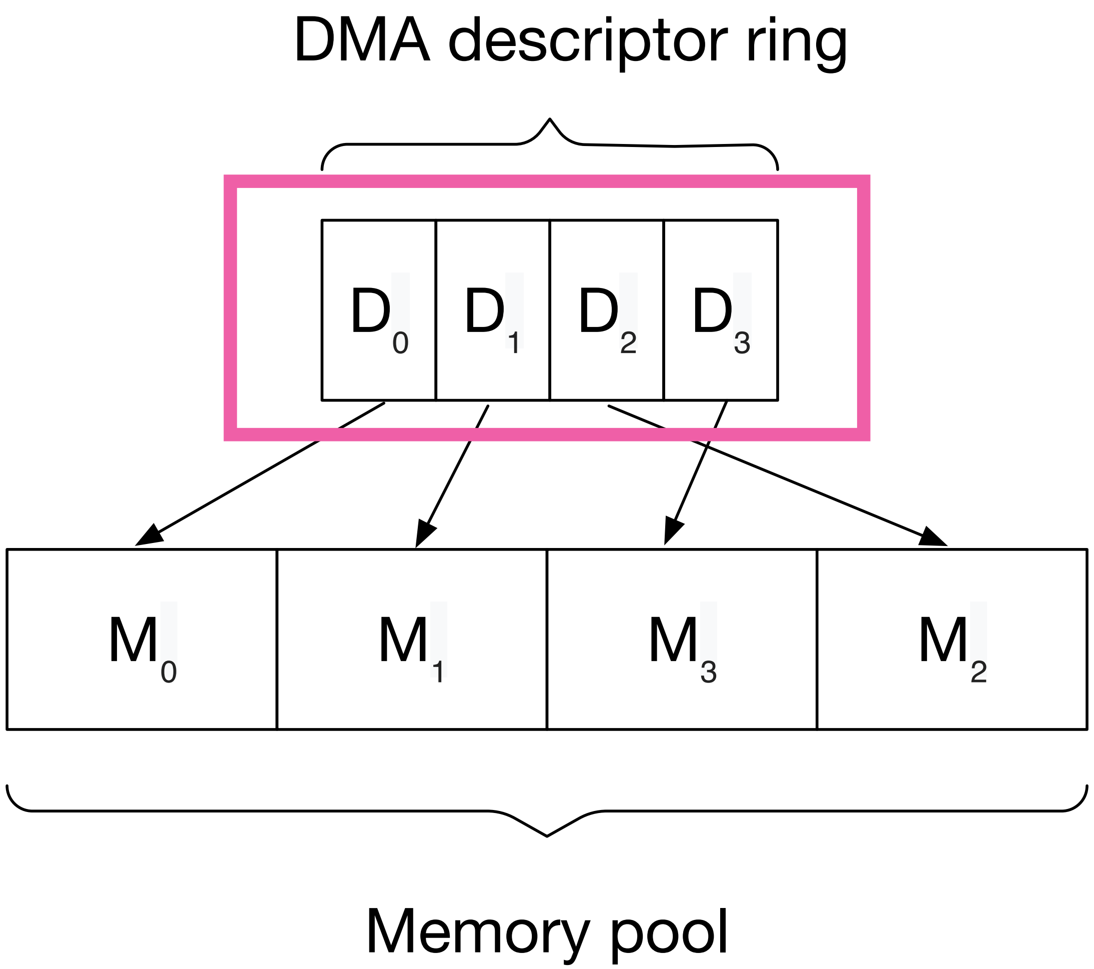
Example: NIC RX ring

- An RX queue consists of two components: DMA descriptor ring and a memory pool.
- On packet receive, NIC takes a DMA descriptor, and DMAs to the memory it points to.



Example: NIC RX ring

- DMA descriptors use physical memory addresses.
- Need to ensure that processes are not able to write to arbitrary physical memory.
- IOMMUs are device-based, not queue based.
- Parakernel is responsible for DMA descriptor ring writes.



Example: timers

- Hardware timers, such as the APIC timer, are not partitionable.
- The kernel could provide a periodic timer, but this is inefficient.
- Timers as a kernel abstraction are worth keeping.

Discussion

Evaluation plan

- Network-intensive application (for example, MICA KVS)
- Application running on a managed runtime (for example, Node.js)
- Target platform is undecided

Open issues

- What other kernel abstractions are needed?
- How to provide backwards compatibility?
    - Linux API as a library

How are parakernels different?

- **Demikernels** propose an unified kernel-bypass device interface, but retain POSIX for backward compatibility.

- **Exokernels** eliminate all kernel abstractions, but parakernels keep for I/O devices partitioning by queues.

- **Arrakis** partitions virtual devices, parakernels partition I/O queues.

- **µ-kernels** keep kernel abstraction only if it is required for correctness.

Summary

- Parakernels aims to improve **application-level parallelism** and **unlock** today's **fast I/O devices** by *partitioning* hardware resources and *minimising OS participation* in data plane operations.
- We are working on a prototype parakernel in Rust.

Thanks!

penberg.org

Backup slides

What devices are partition able?

- If a device has internal parallelism that is exposed to the programmers, it is partitionable for parakernels.
- I/O queues that run independently are TODO
  - Multi-queue NICs, NVMe devices, and GPUs.

Parakernel vs. demikernel

- Demikernels propose a new high-level I/O kernel abstraction for kernel-bypass devices.

- Demikernel retains the monolithic, shared-memory kernel for control plane and legacy applications, parakernel wants to eliminate it.

Parakernel vs. exokernel

- Exokernels aim for performance and flexibility by eliminating all kernel abstractions.
- Ideal kernel interface is the hardware interface.
- Parakernels are inspired by exokernels, but take a more relaxed approach to kernel abstractions.

Parakernel vs. Arrakis

- Arrakis uses hardware virtualisation (SR-IOV) to partition devices between applications.
- The problem with NICs, for example, is that each application has its own vNIC (and own MAC address), which exposes partitioning to clients.
- Parakernel partitions the NIC by its queues, which is transparent to clients.

Parakernel vs. µ-kernel

- µ-kernels tolerate a kernel abstraction only if it is require for correctness, parakernel does not take this strict view.
- Parakernels could be implemented as µ-kernels, but partitioning I/O devices by queue requires a multiserver approach.
- It is not clear what the performance overhead is, which is why current implementation of parakernel uses kernel drivers.