Netmap: A novel framework for fast packet I/O

Rinik Kumar <rinik@mit.edu>

User-space networking

- Network speeds increasing quickly
  - Gilder's Law
  - "Total bandwidth of communication systems triple every 12 months"
- Software architectures are the same:
  - raw sockets, BPF, libpcap
  - `mbuf/sk_buff` encapsulation
  - poor parallelism

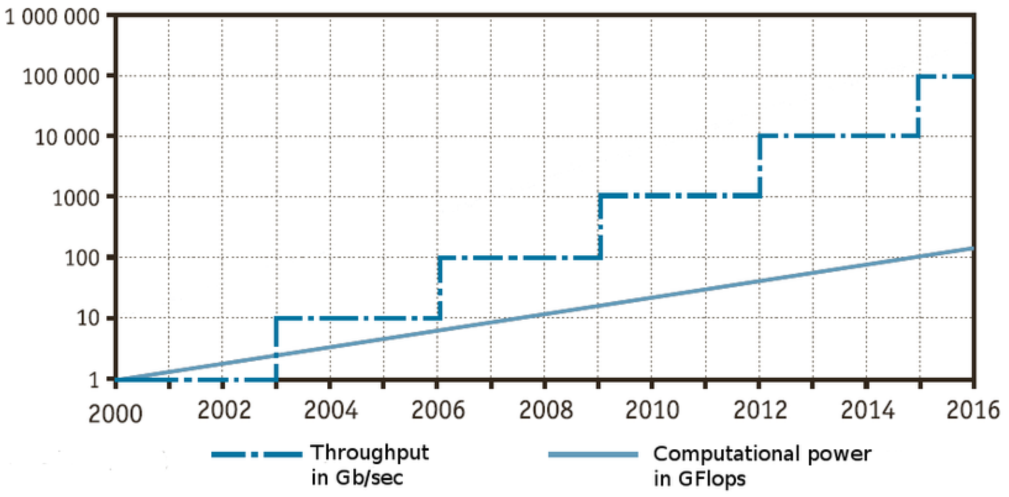


Image source: http://syrcose.ispras.ru/2013/files/submissions/16_syrcose2013.pdf

Netmap overview

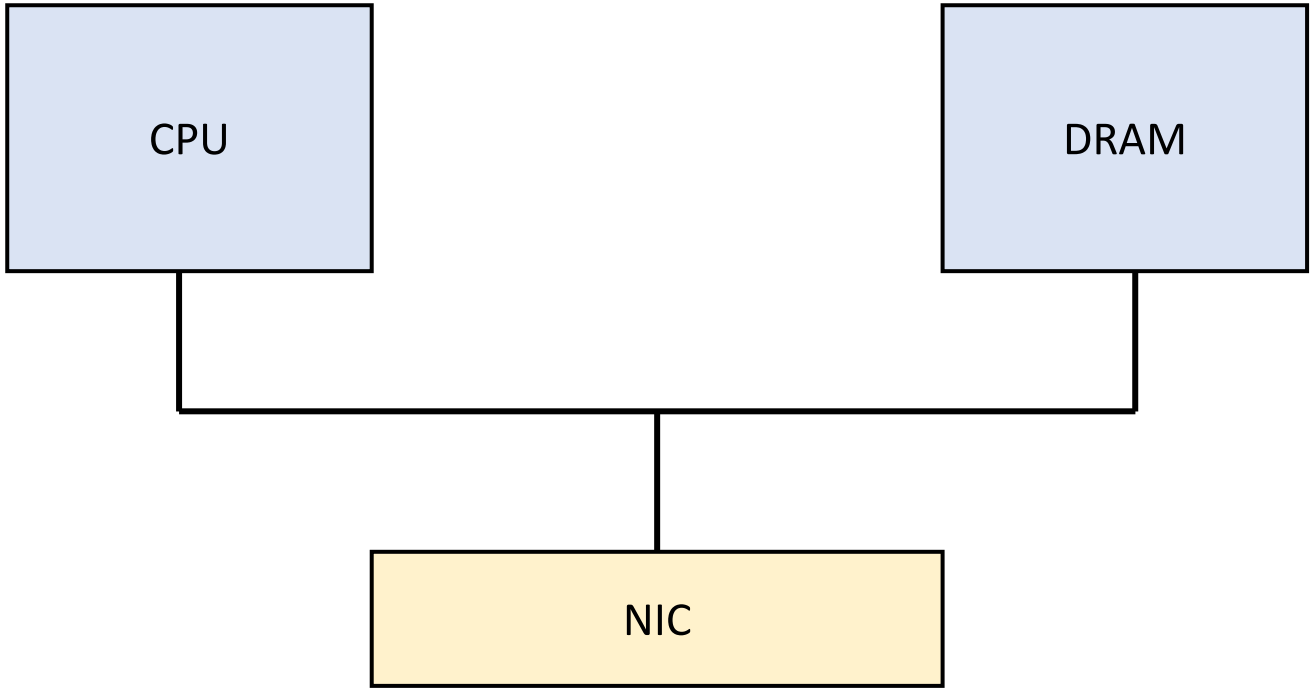
- Identified and removed 3 main packet processing costs:
  - Per-packet dynamic memory allocations
  - Memory copies
  - System call overheads
- Goals:
  - Performance
  - Ease of use
  - Memory safety

Main optimizations

- Metadata representation
  - Abstracts device-specific features
  - Supports batched system calls
- Linear fixed-size packet buffers
  - Eliminates per-packet allocations
- Packet buffers shared between user program and kernel
  - Eliminates memory copy overhead
- Support for useful hardware features
  - E.g. multiple hardware queues

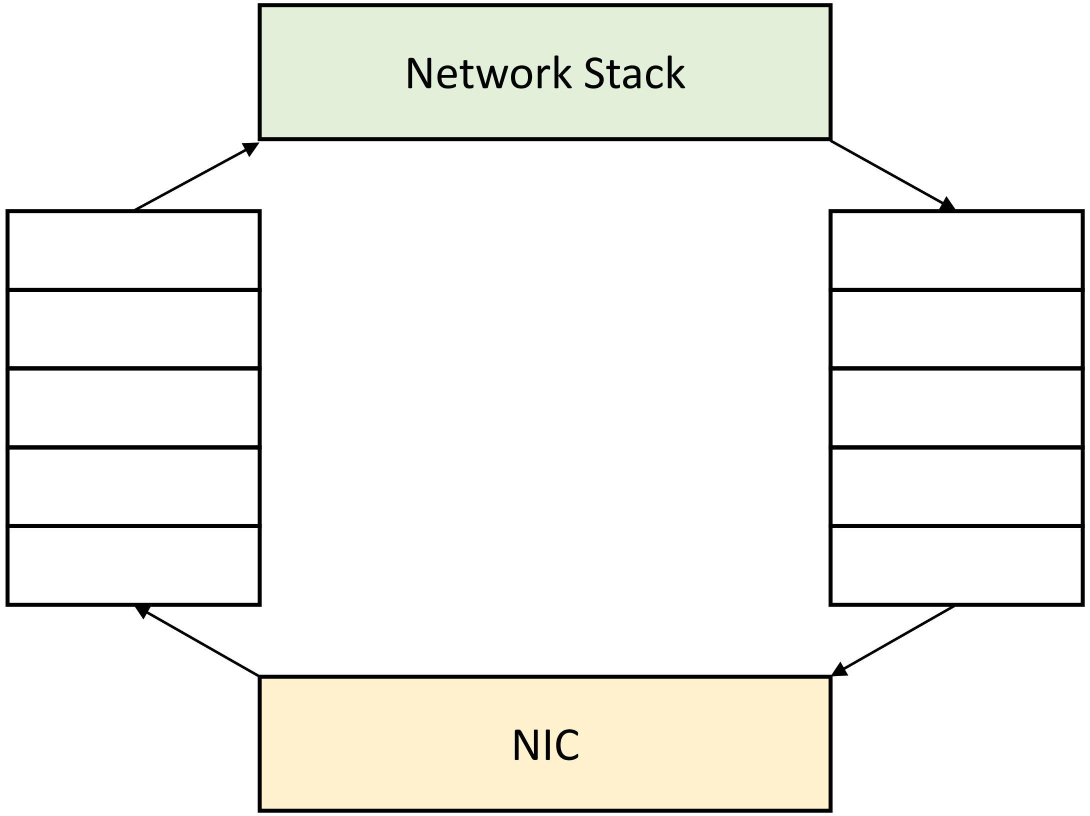
Q: Why doesn’t DPDK solve the same problem as Netmap?

Networking review (DMA)

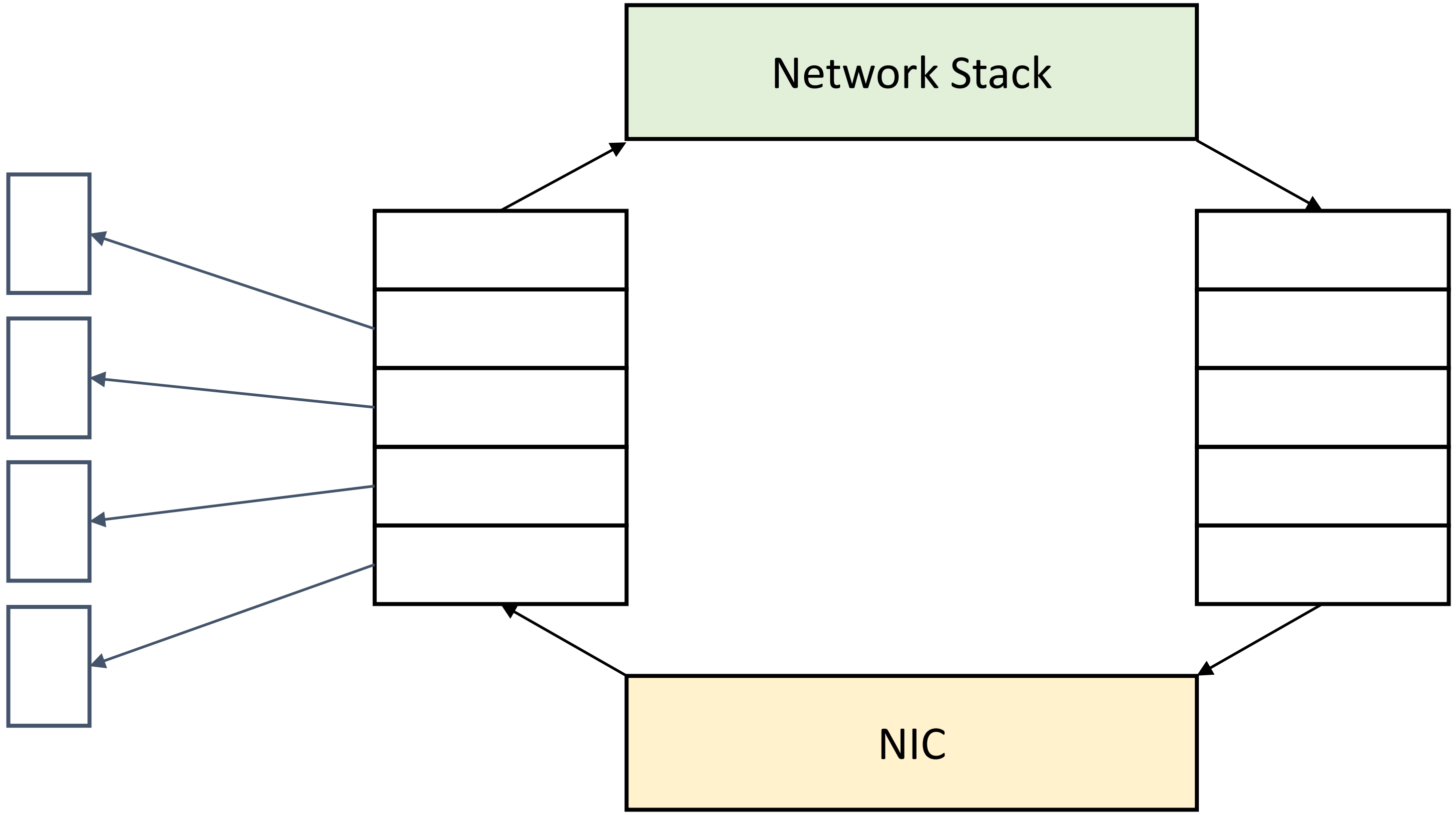


MMIO vs. DMA

Networking review (Ring buffer)

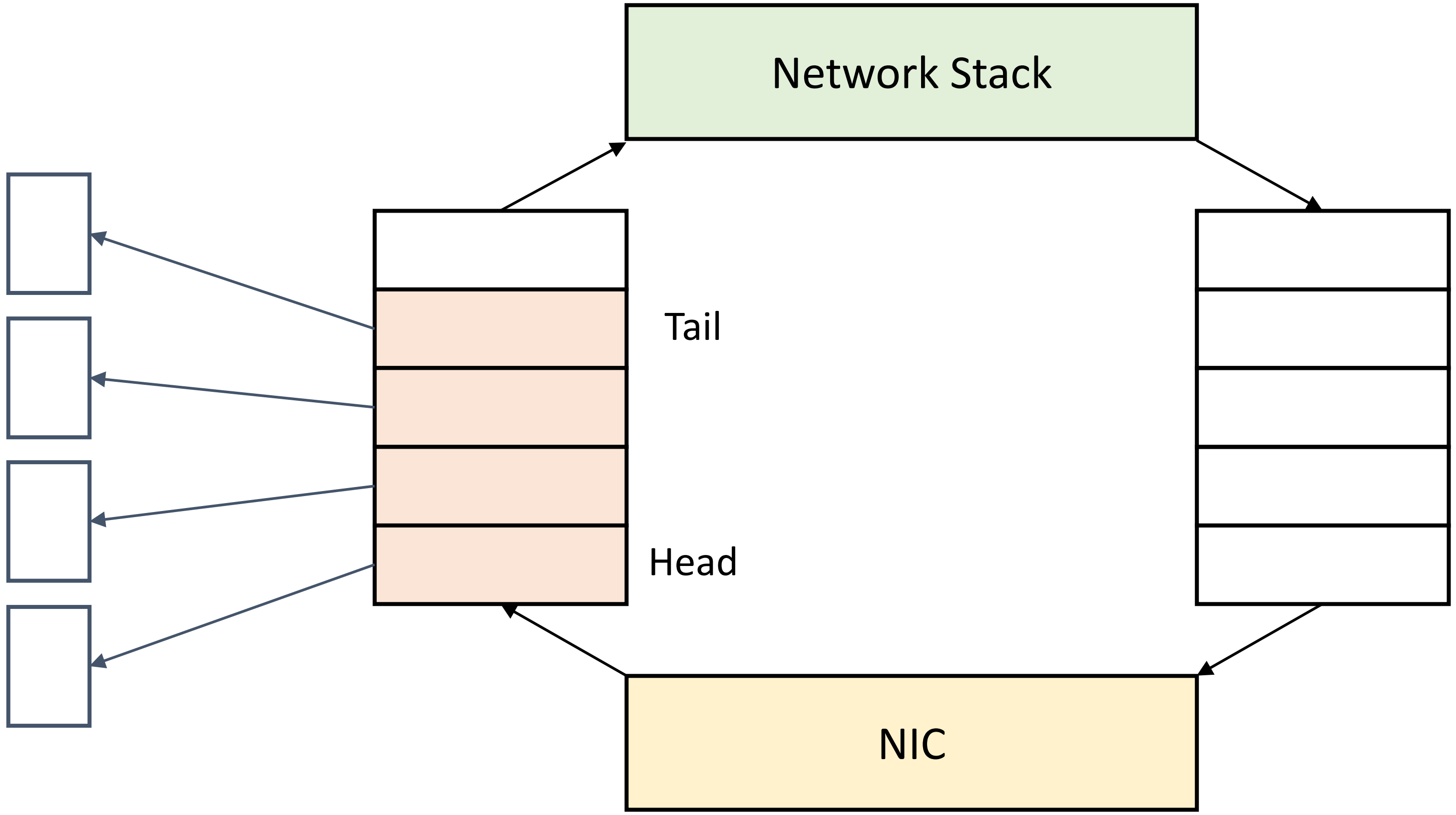


Networking review (Ring buffer)



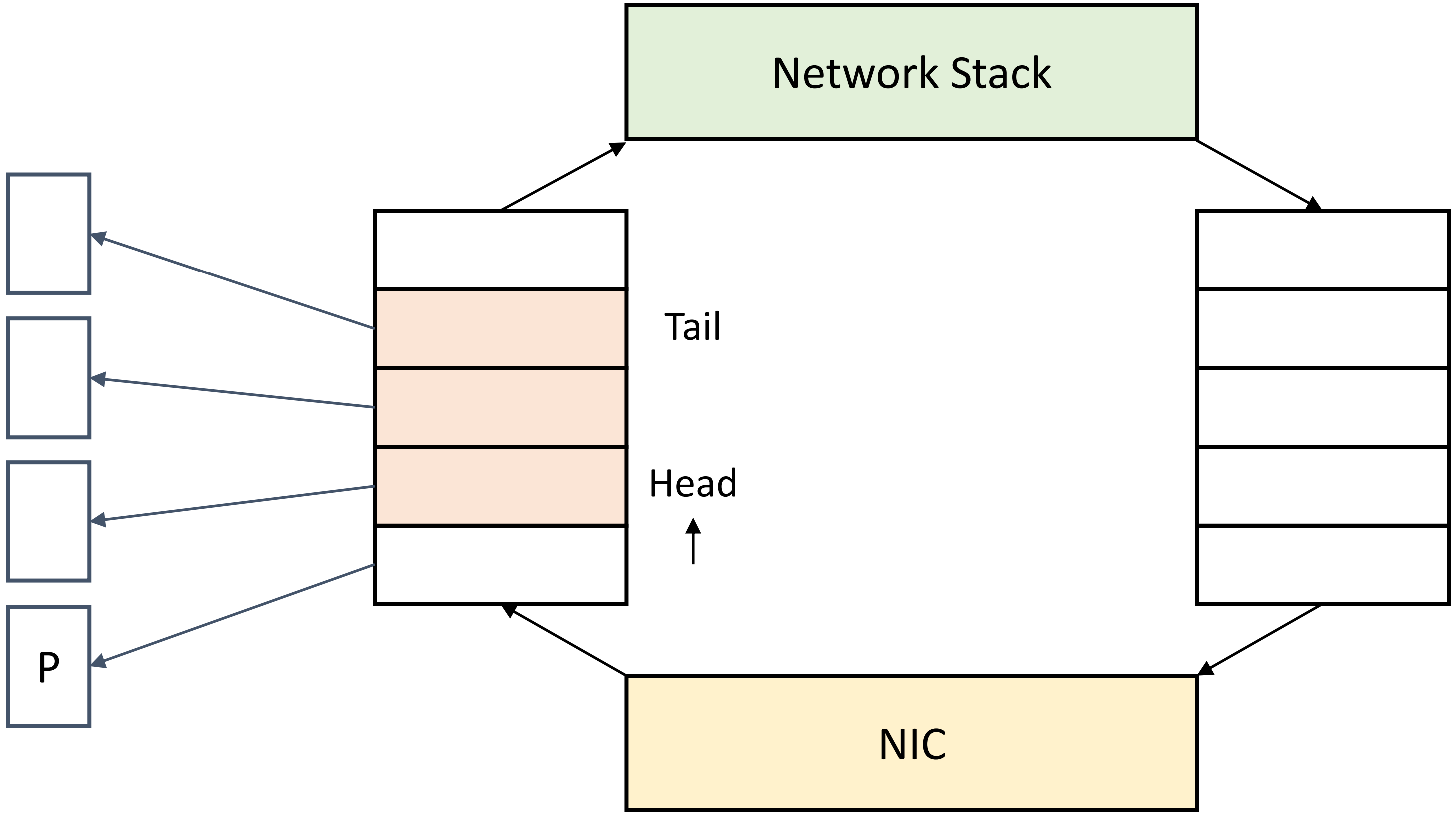
OS allocates `mbufs/sk_buffs` for each ring slot

Networking review (Ring buffer)



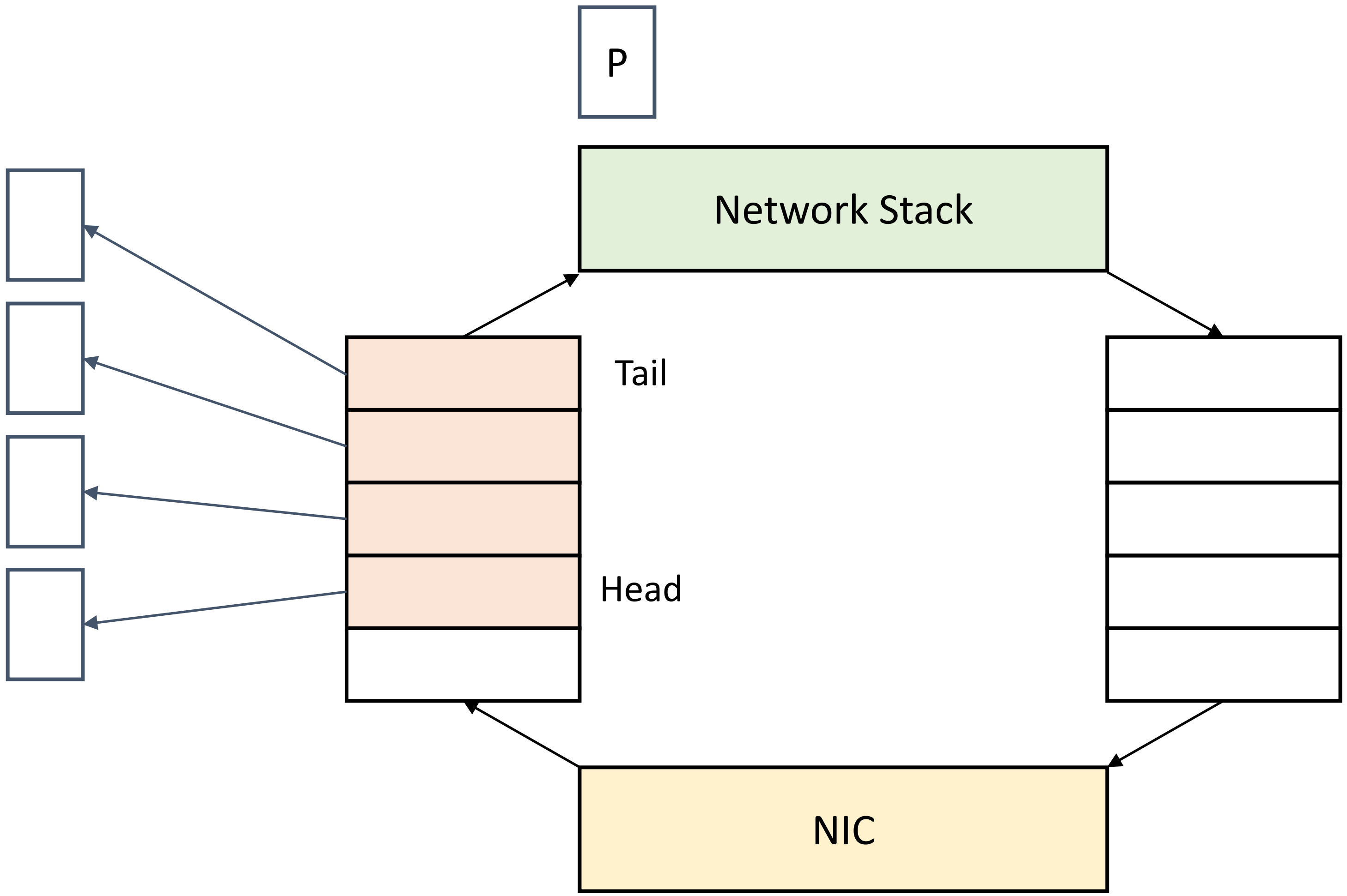
Free slots are in the [Head, Tail) interval

Networking review (Ring buffer)



Copy received packet into buf; update head pointer

Networking review (Ring buffer)



Move received packet into network stack; allocate buf; update tail pointer

Networking review (Interrupts vs. Polling)

- How does the OS know packets have arrived?
- Options:
  - **Interrupts:** NIC generates interrupt, OS handles packet upon interrupt
  - **Polling**: NIC sets done flag; OS periodically checks ring buffer
- Interrupts do not scale!
  - Receive livelock
  - CPU spends more time handling interrupts than processing packets

Networking review (Packet buffers)

- `mbuf` (BSD) vs. `sk_buff` (Linux)
- Buffer chains:
  - Packets are typically very small or near the MTU
  - Pool of small buffers (mbufs; ~256 bytes each)
  - Pool of large buffers (mbuf clusters; ~2048+ bytes each)
  - Potential for several allocations per packet

Reference: https://people.sissa.it/~inno/pubs/skb-reduced.pdf

Case study: FreeBSD sendto()

File	Function/description	time ns	delta ns
user program	sendto system call	8	96
uipc_syscalls.c	sys_sendto	104	
uipc_syscalls.c	sendit	111	
uipc_syscalls.c	kern_sendit	118	
uipc_socket.c	sosend	—	
uipc_socket.c	sosend_dgram sockbuf locking, mbuf allocation, copyin	146	137
udp_usrreq.c	udp_send	273	
udp_usrreq.c	udp_output	273	57
ip_output.c	ip_output route lookup, ip header setup	330	198
if_ethersubr.c	ether_output MAC header lookup and copy, loopback	528	162
if_ethersubr.c	ether_output_frame	690	
ixgbe.c	ixgbe_mq_start	698	
ixgbe.c	ixgbe_mq_start_locked	720	
ixgbe.c	ixgbe_xmit mbuf mangling, device programming	730	220
–	on wire	950	

Related networking APIs

- Raw Sockets
  - Direct interface to L3 traffic
- AF_PACKET (Linux)
  - Direct interface to L2 traffic
- BPF (*BSDs)
  - Direct interface to L2 traffic
  - User-space process can provide filter program to kernel
  - Kernel only copies packets that pass filter

Related networking optimizations

- Run application code in the kernel
  - e.g. Click
- Custom device drivers in user-space
  - e.g. DPDK
- Hardware accelerators
  - Hardware offloading (e.g. TCP acceleration)
  - NetFPGA, Catapult
  - Programmable SmartNICs, P4 language

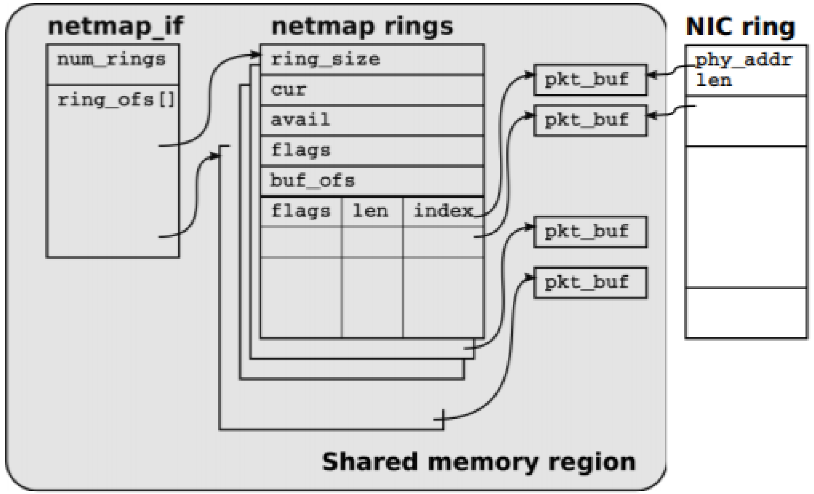
Reference: https://lwn.net/Articles/629155/

Main optimizations (revisited)

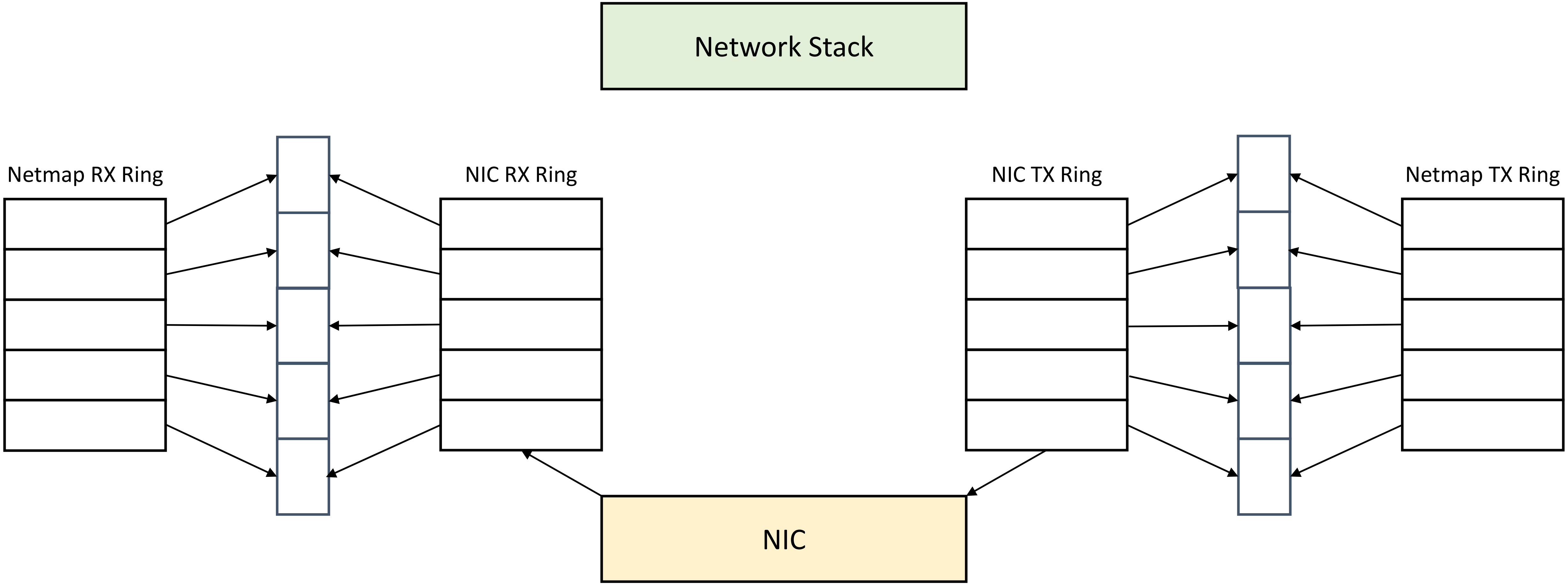
- Metadata representation
  - Abstracts device-specific features
  - Supports batched system calls
- Linear fixed-size packet buffers
  - Eliminates per-packet allocations
- Packet buffers shared between user program and kernel
  - Eliminates memory copy overhead
- Support for useful hardware features
  - E.g. multiple hardware queues

Netmap data structures

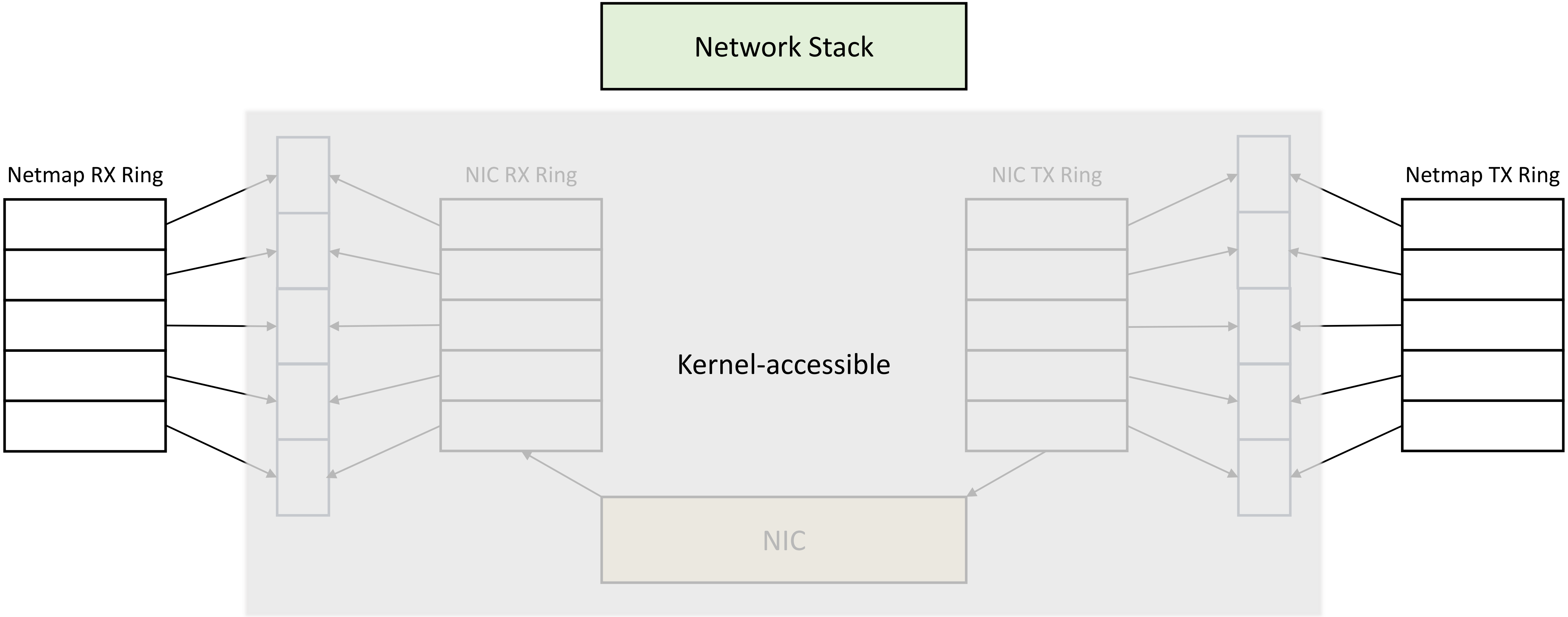
- `netmap_if` describes attributes of an interface
- `netmap_ring` replicates the ring buffer implemented by the NIC
- `pkt_buf` is a fixed-size packet buffer



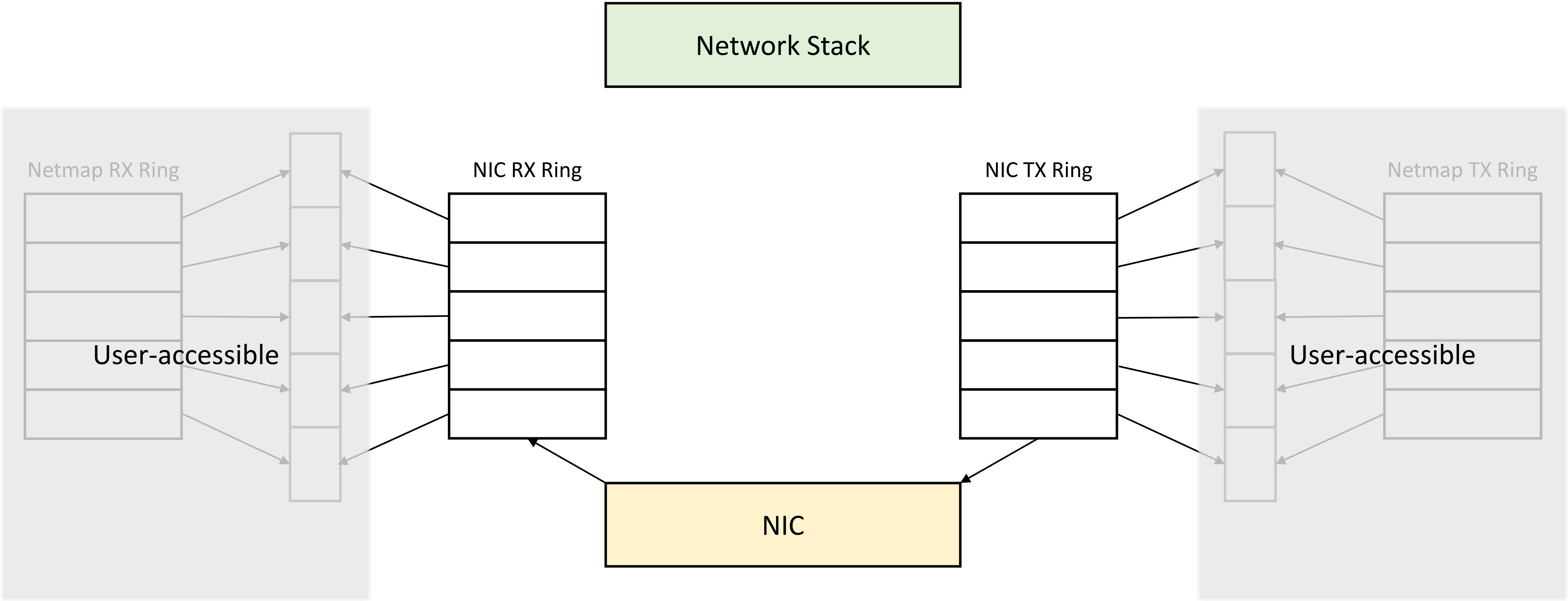
Netmap data structures



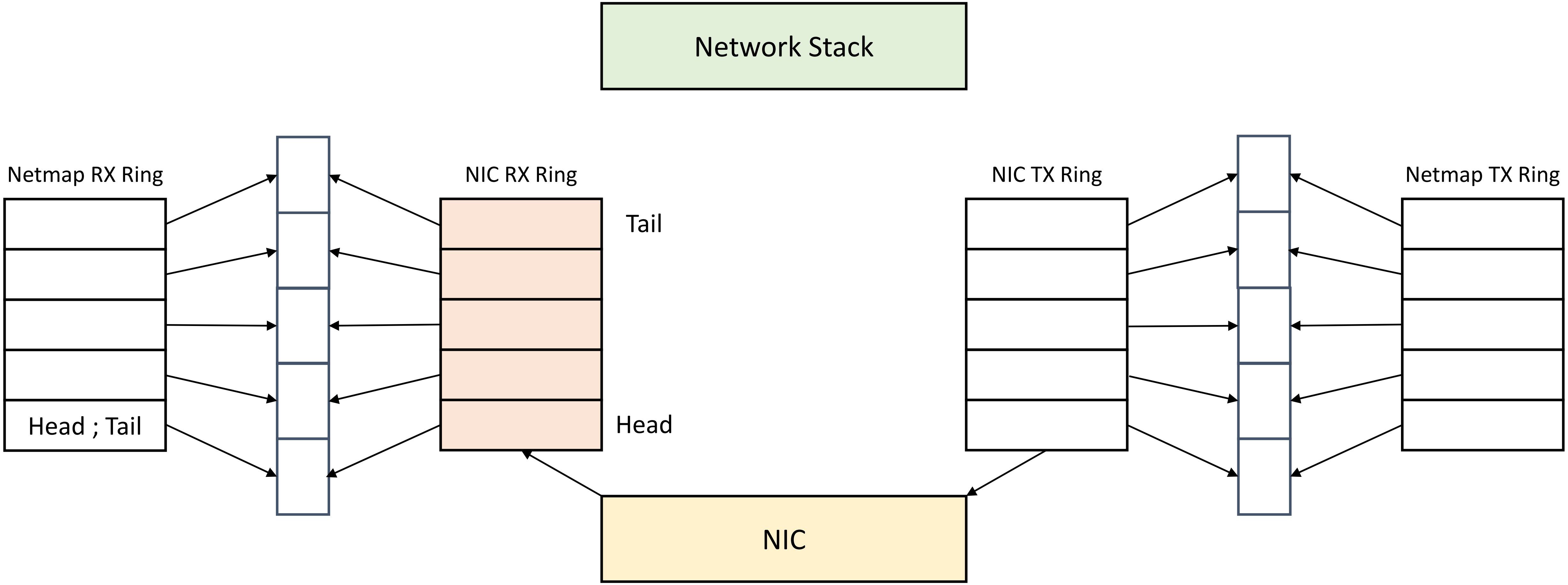
Netmap data structures



Netmap data structures

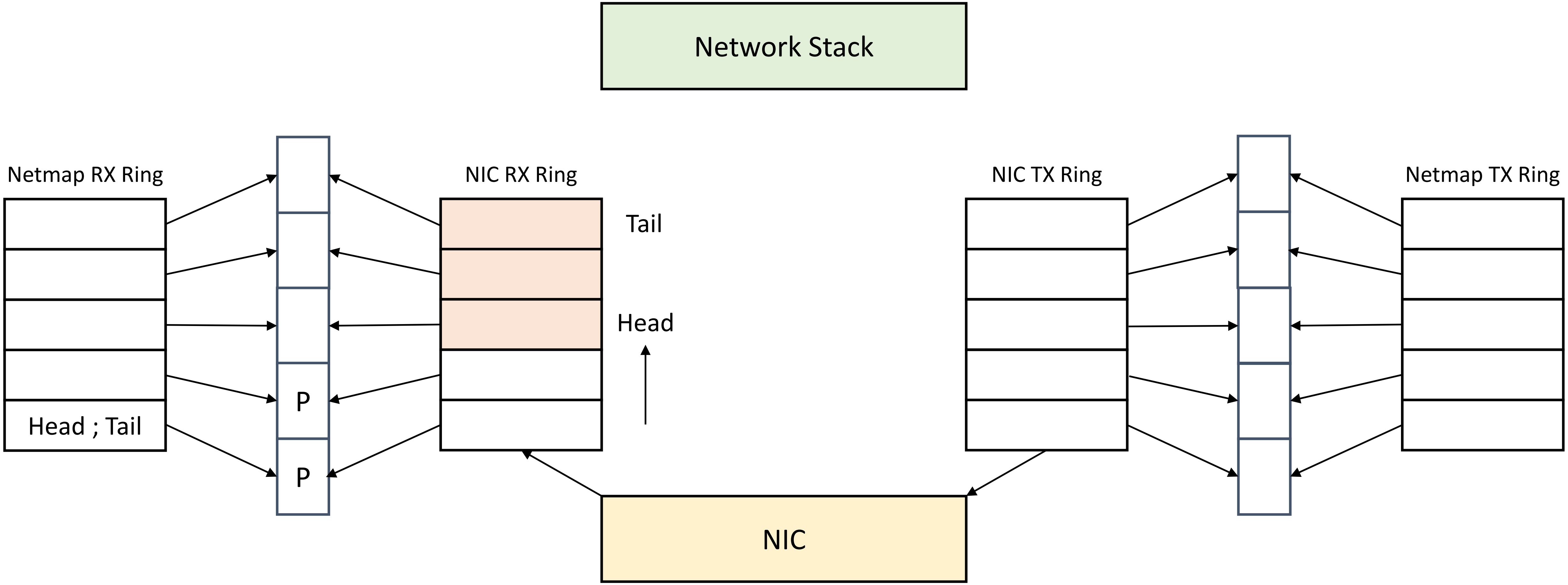


Netmap data structures



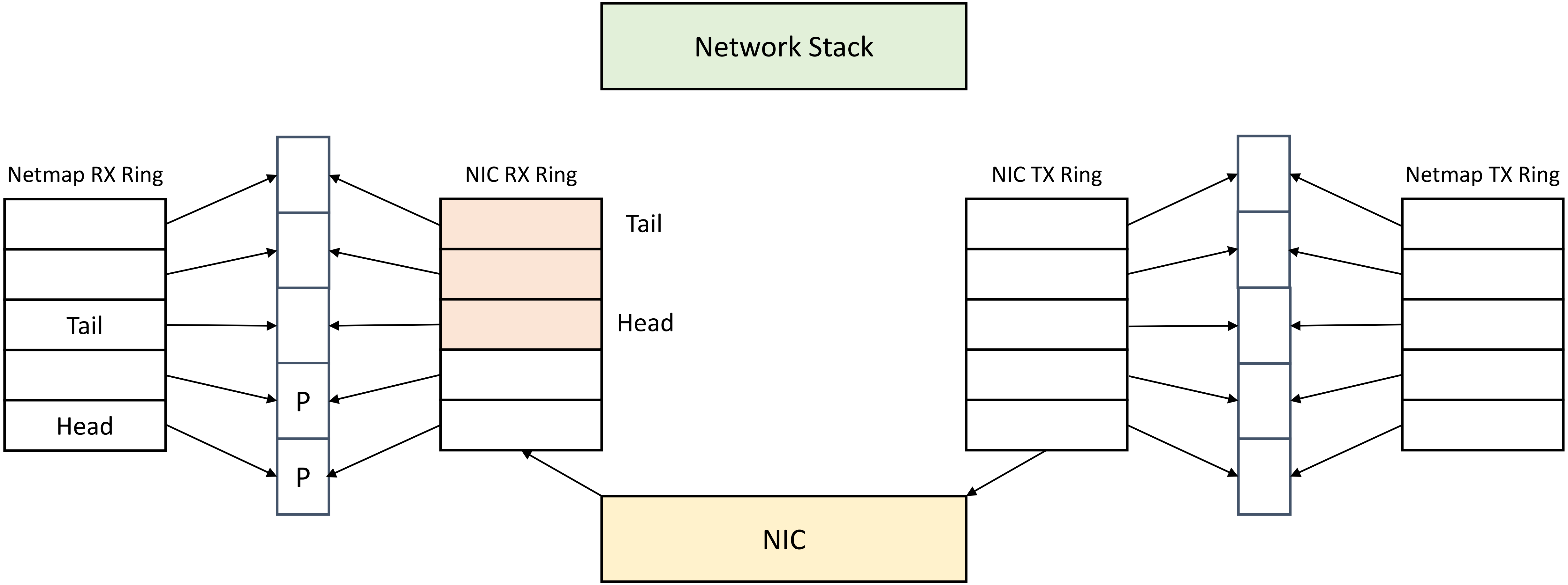
Netmap applications access slots in [head, tail)

Netmap data structures



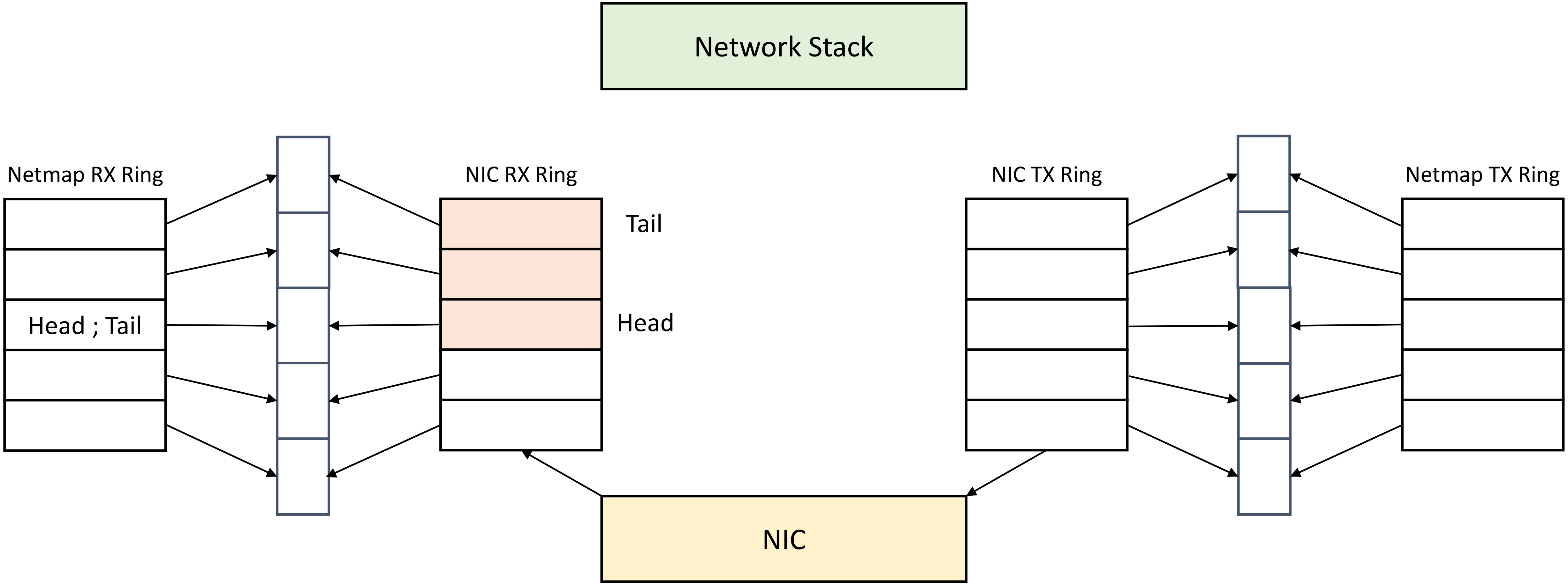
Copy 2 received packets into buf; update head pointer

Netmap data structures



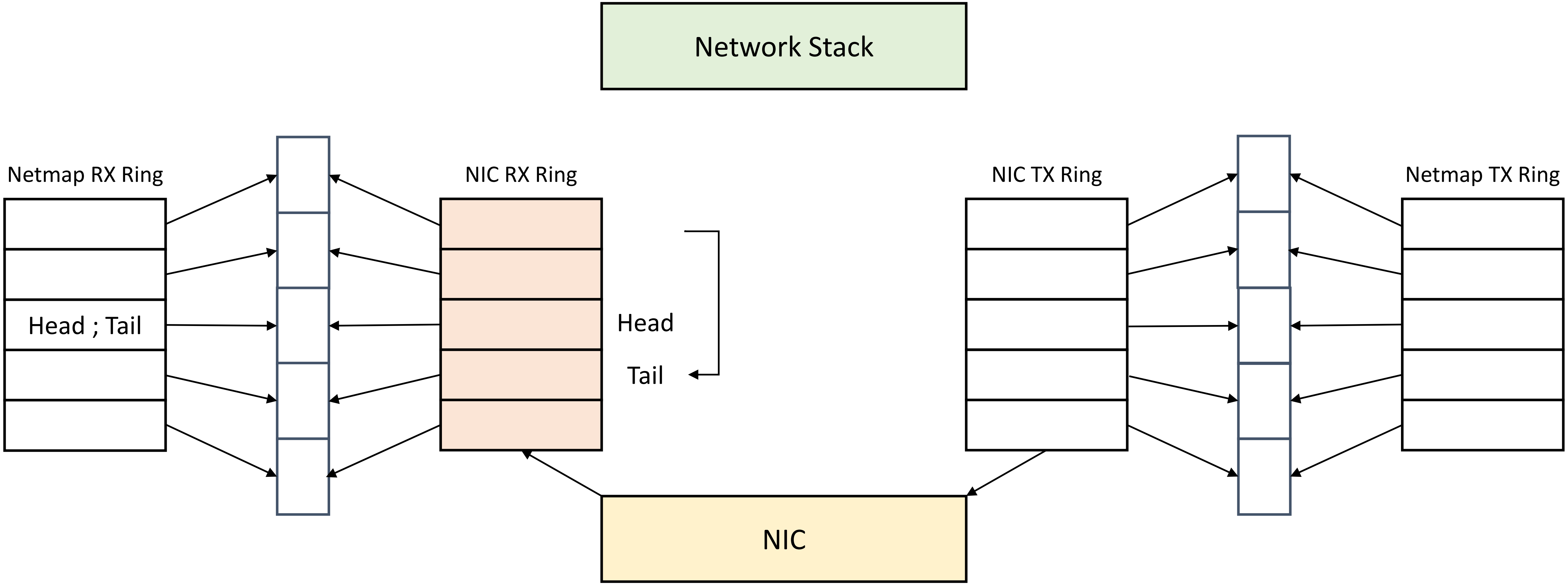
`ioctl(..., NIOCRXSYNC);` update netmap RX ring tail pointer

Netmap data structures



Netmap application reads 2 packets

Netmap data structures



`ioctl(..., NIOCRXSYNC);` update NIC RX ring tail pointer

Removing allocations

- Packet buffers are pre-allocated during initialization
- Metadata associated to packets are stored in netmap ring slots
- Each slot in netmap ring associated to a fixed-size packet buffer
  - 2K-byte buffer supports typical MTU of 1500
  - Buffers are reused as new packets arrive

Batching system calls

- System calls are significant source of latency
- Send/receive multiple packets in a single system call
- User program and kernel coordinate at synchronization points
  - `ioctl(..., NIOCTXSYNC)`
  - `ioctl(..., NIOCRXSYNC)`
  - Calls to `ioctl()` are nonblocking
  - Validate and update `netmap_ring` fields

Removing copies

- Netmap data structures are shared between user-space programs and the kernel
- Example 1:
  - Packet buffers accessible from user-space and kernel-space
  - Eliminates need to copy packets into user-space
- Example 2:
  - Packet buffers for all interfaces in same memory region
  - Eliminates need to copy forwarded packets

Protecting shared memory

- `netmap_ring` always owned by user-space application, except during system call
- Packet buffers between `cur` and `cur+avail–1` are owned by user-space application
  - Generally, `[cur, cur+avail-1] = [head, tail]`
- Program can break invariants or corrupt netmap data structures, but cannot cause kernel to crash

Q: Is this sufficient protection?

Supporting real hardware

- Multiple hardware queues
  - Abstracted using `netmap_if`
- Netmap requires some modifications to device drivers
  - Minimal changes needed
  - Drivers must support synchronization routines (`NIOCTXSYNC` and `NIOCRXSYNC`)
  - Initialization of rings in netmap mode
  - Export device driver locks

Netmap API

- No user-space library
  - All data structures, prototypes, macros in header file (`netmap.h`)
  - Also provides `netmap_user.h`, containing additional utilities to manipulate netmap data structures in user-space
- Compatibility libraries to support existing packet processing libraries:
  - E.g. libpcap
  - Map essential libpcap functions to netmap calls

Netmap API

```
fds.fd = open("/dev/netmap", O_RDWR);
strcpy(nmr.nm_name, "ix0");
ioctl(fds.fd, NIOCREG, &nmr);
p = mmap(0, nmr.memsize, fds.fd);
nifp = NETMAP_IF(p, nmr.offset);
fds.events = POLLOUT;
for (;;) {
  poll(fds, 1, -1);
  for (r = 0; r < nmr.num_queues; r++) {
    ring = NETMAP_TXRING(nifp, r);
    while (ring->avail-- > 0) {
      i = ring->cur;
      buf = NETMAP_BUF(ring, ring->slot[i].buf_index);
      ... store the payload into buf ...
      ring->slot[i].len =  ... // set packet length
      ring->cur = NETMAP_NEXT(ring, i);
    }
  }
}
```

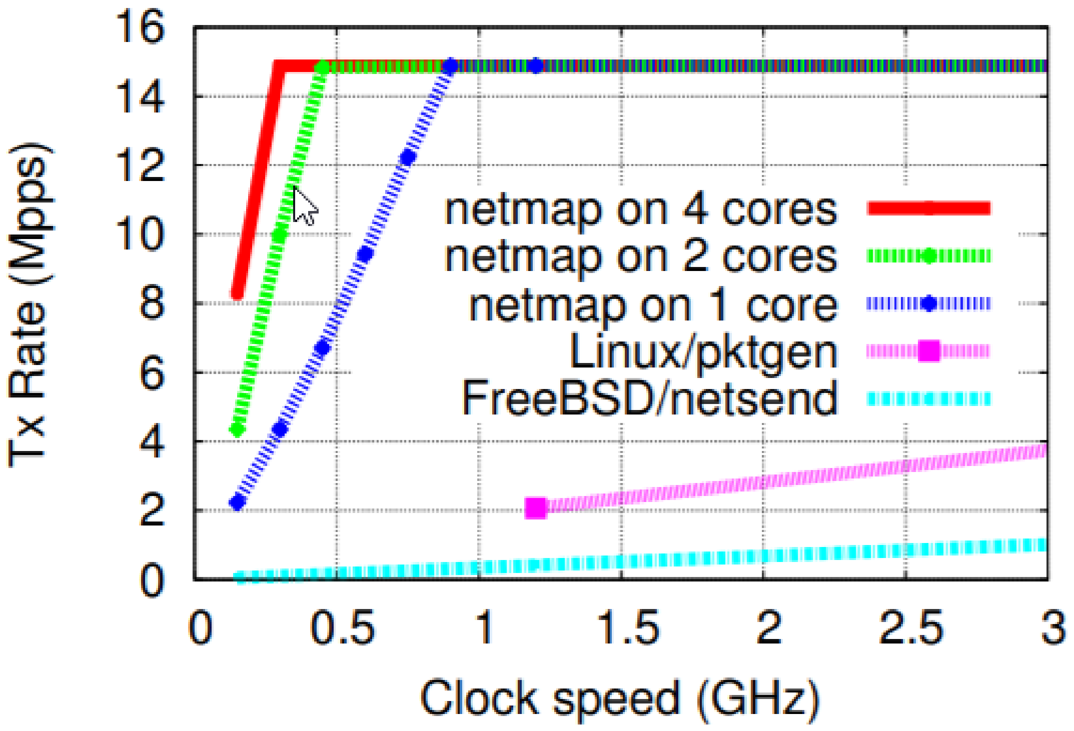
Netmap demo

https://github.com/rk9109/netmap-demo (WIP)

Q: How can existing applications take advantage of Netmap API?

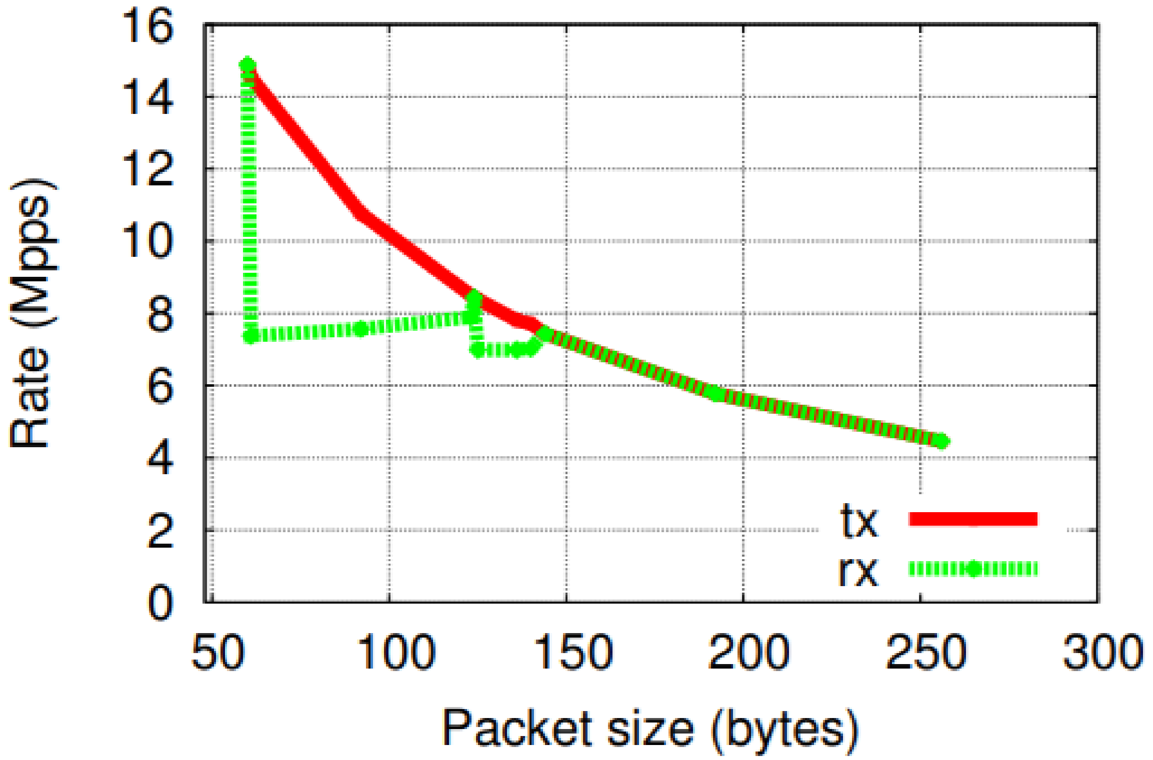
Performance (pt. 1)

- Transmit speed vs. clock rate



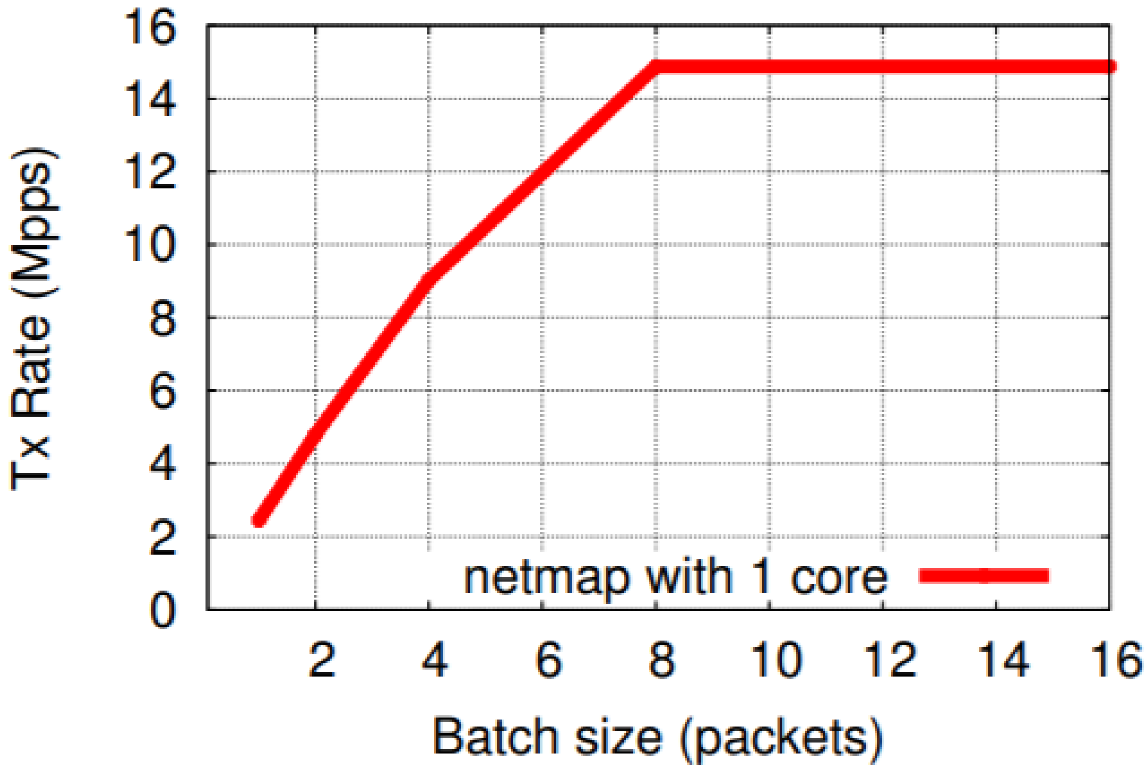
Performance (pt. 2)

- Transmit speed vs. packet size



Performance (pt. 3)

- Transmit speed vs. batch size



Packet forwarding performance

- Tested using existing packet forwarding applications

Configuration	Mpps
netmap-fwd (1.733 GHz)	14.88
netmap-fwd + pcap	7.50
click-fwd + netmap	3.95
click-etherswitch + netmap	3.10
click-fwd + native pcap	0.49
openvswitch + netmap	3.00
openvswitch + native pcap	0.78
bsd-bridge	0.75

Conclusion

- Netmap provides improved performance without using dedicated hardware acceleration/features.
  - 4 to 40 times faster compared to similar APIs
- Simple and accessible API
  - Merged into FreeBSD HEAD
  - Available as Linux kernel module
- Combination of the correct set of simple optimizations can result in impressive performance!