Fast Packet Processing

Lecture 11

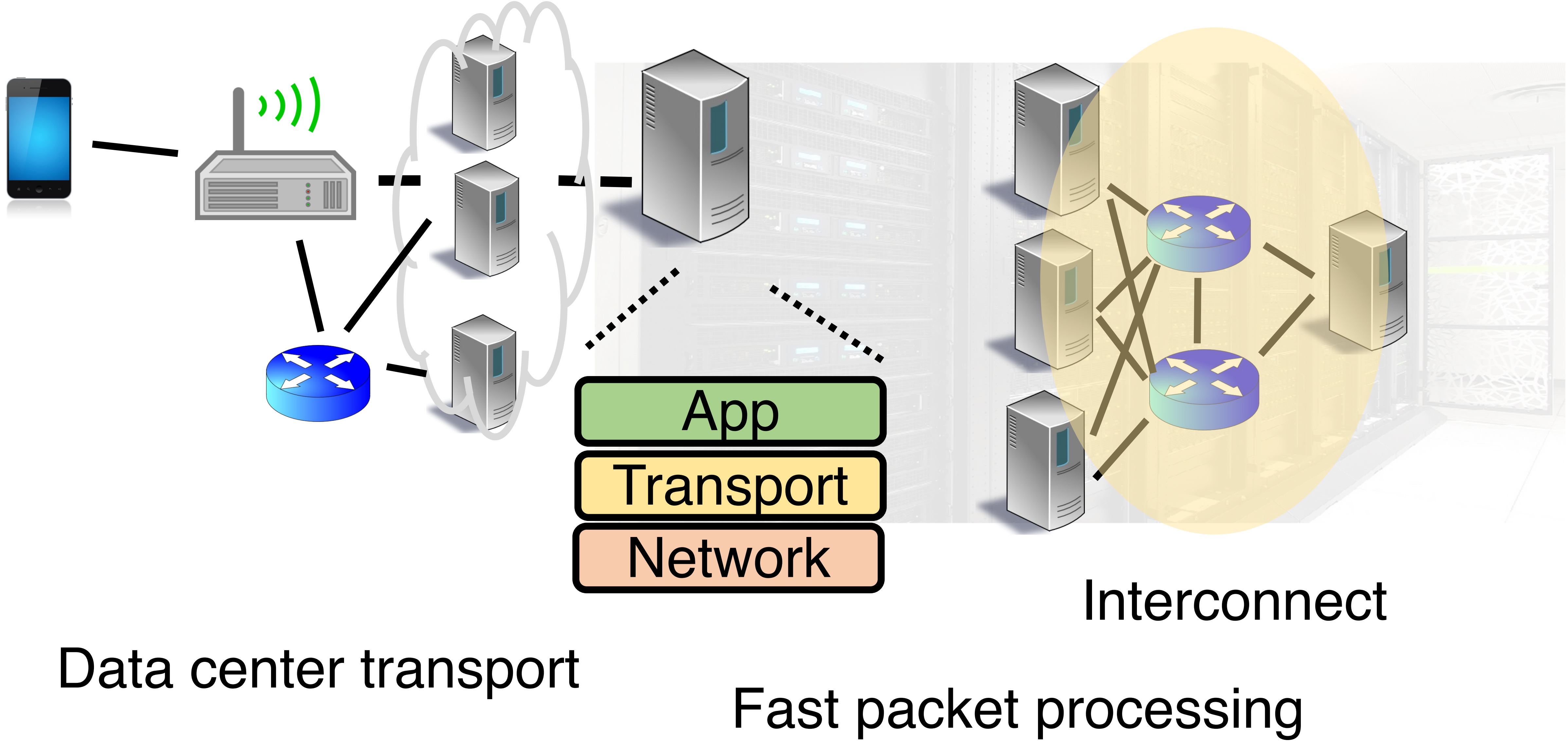
Srinivas Narayana

http://www.cs.rutgers.edu/~sn624/553-S23

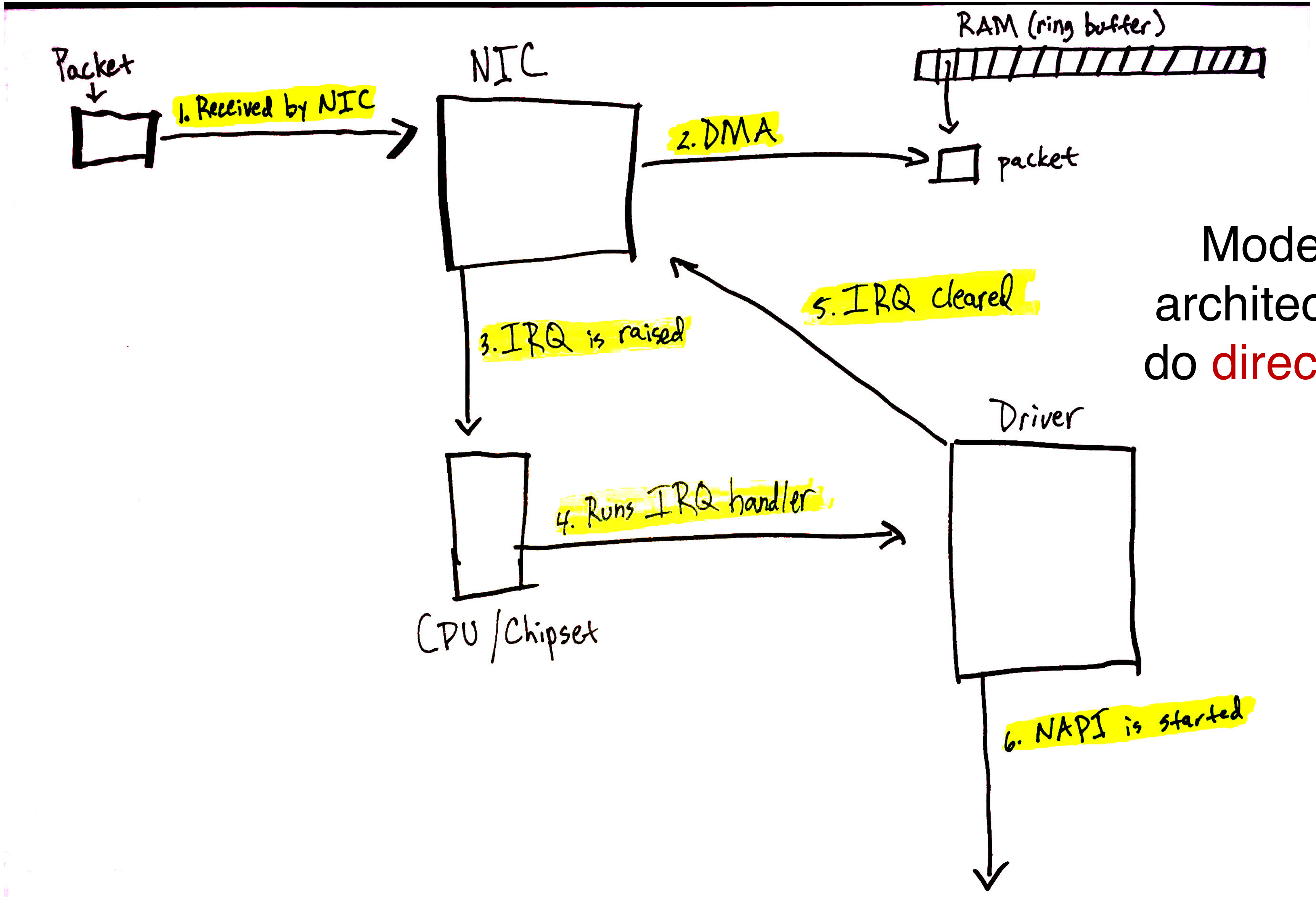
Some slides were adapted from those of Gianni Antichi

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Context: Networking for Internet services



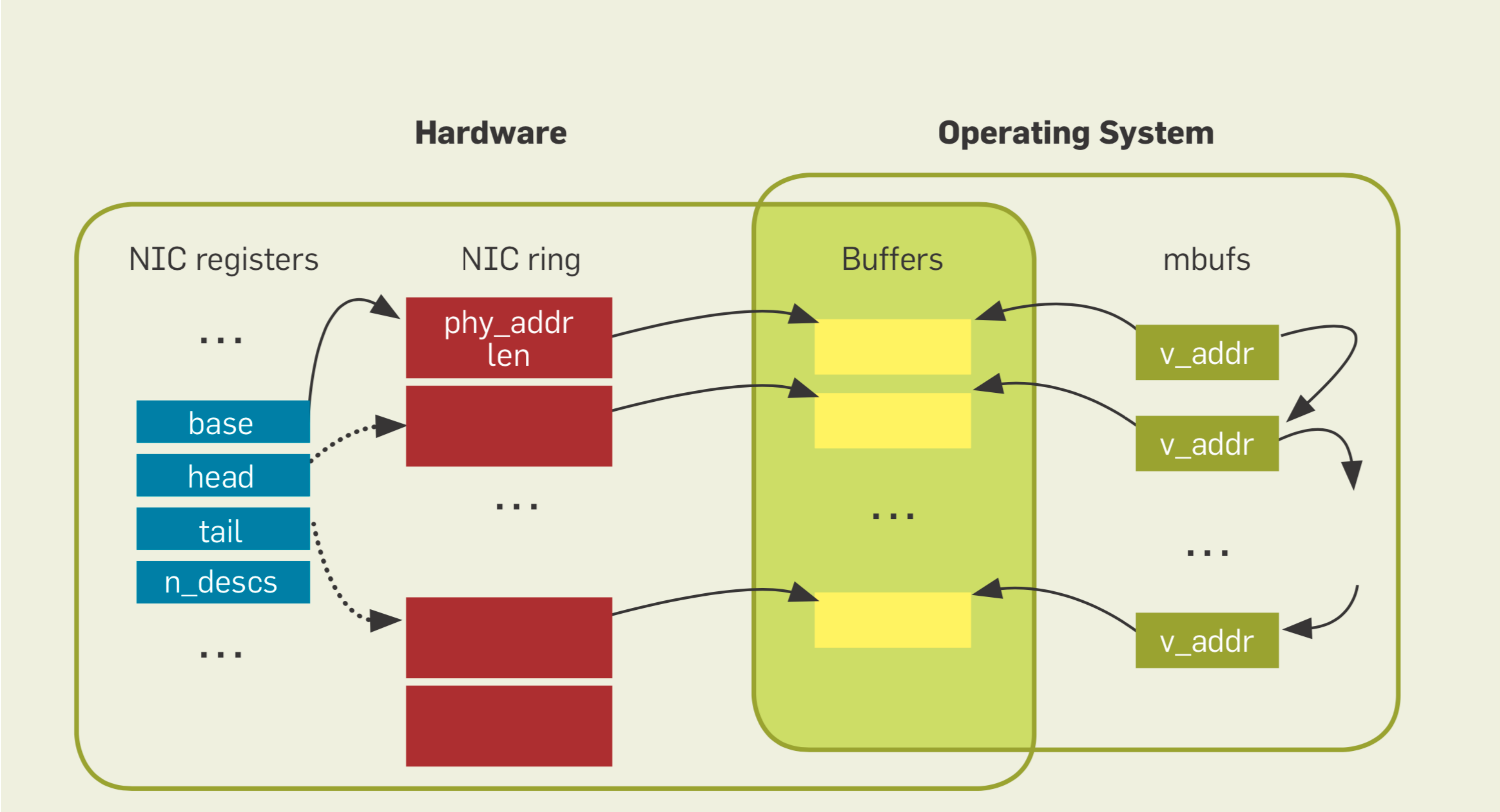
Packet processing on Linux



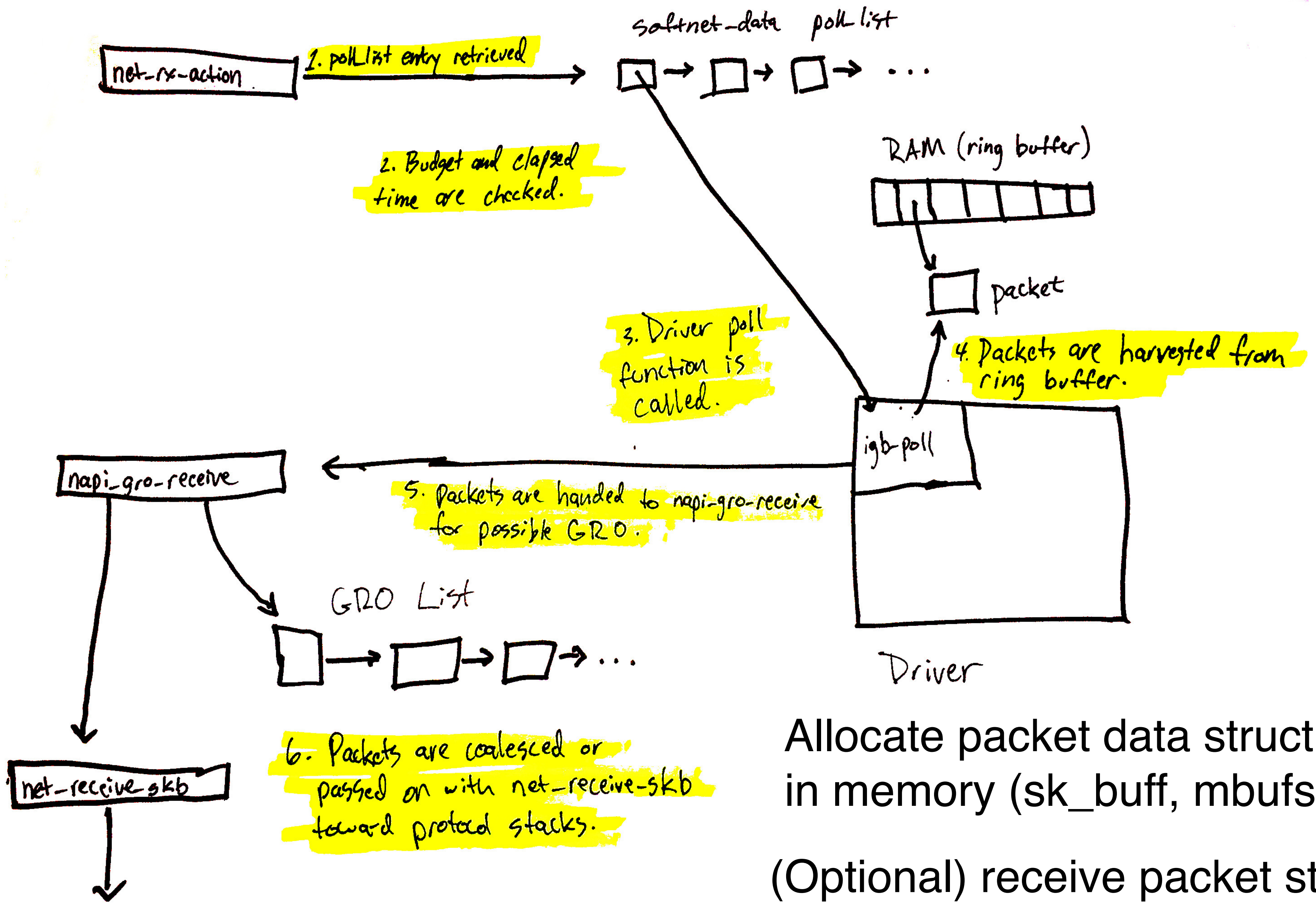
Modern NICs and architectures can also do direct cache access (DCA)

Interrupt mitigation

- Interrupt processing at high rate and priority prevents any other part of the system from progressing (receive livelock).
- Mitigations:
- (1) Interrupt coalescing:
  - Wait (at NIC) for more packets or a timeout until interrupting
- (2) Polling to schedule work across different sources of processing
  - Avoid preemption
- (3) CPU or packet quotas on polling to ensure other parts of the system (user space app) can progress
  - Re-enable interrupts if there is less work than allotted quota



Revisiting network I/O APIs: The netmap framework. CACM'12

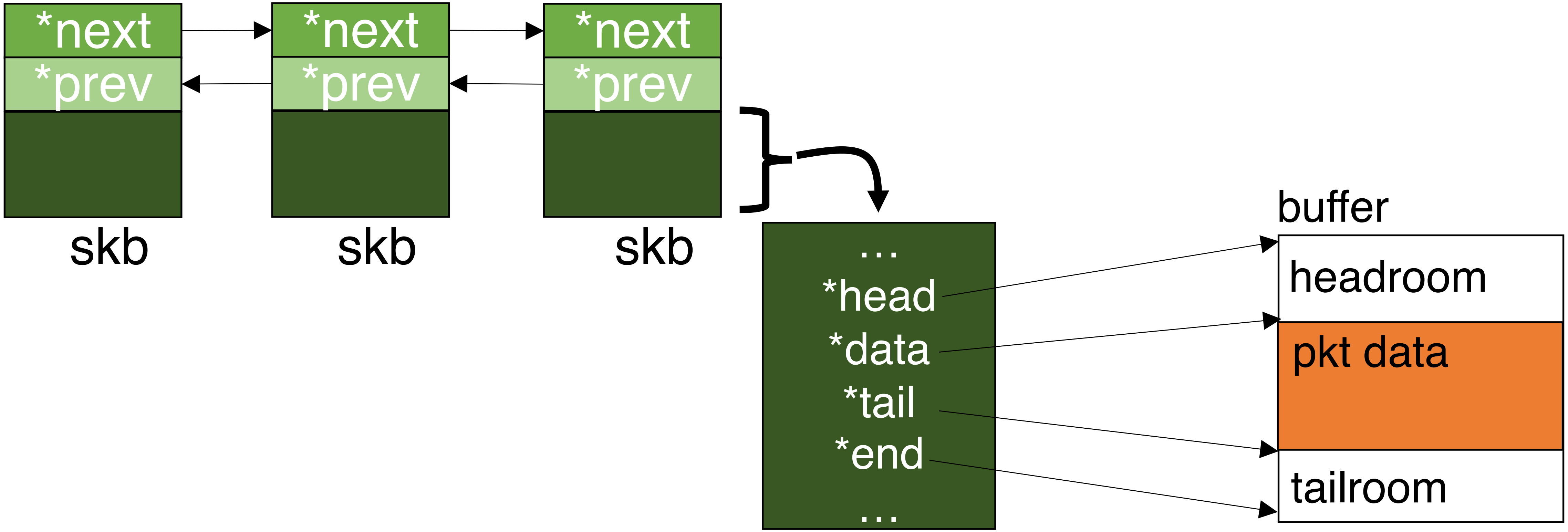


Allocate packet data structures in memory (sk_buff, mbufs, …)

(Optional) receive packet steering

Socket buffers

- Allocate in arbitrary chunks (multiples of 64 bytes)
- Support arbitrary packet sizes, fragments, deferred processing



Other things that happen afterward

- Netfilter: tracking TCP connection state, firewalling, NAT, …
- IP protocol processing: routing
- Transport processing (UDP/TCP protocol layer)

- Some stateless, per-packet work can be done by the NIC:
  - TSO: TCP segmentation offload
  - LRO: Large Receive Offload (also applicable in software)
  - IP checksum
  - Ethernet CRC computation

FreeBSD `sendto()` code path

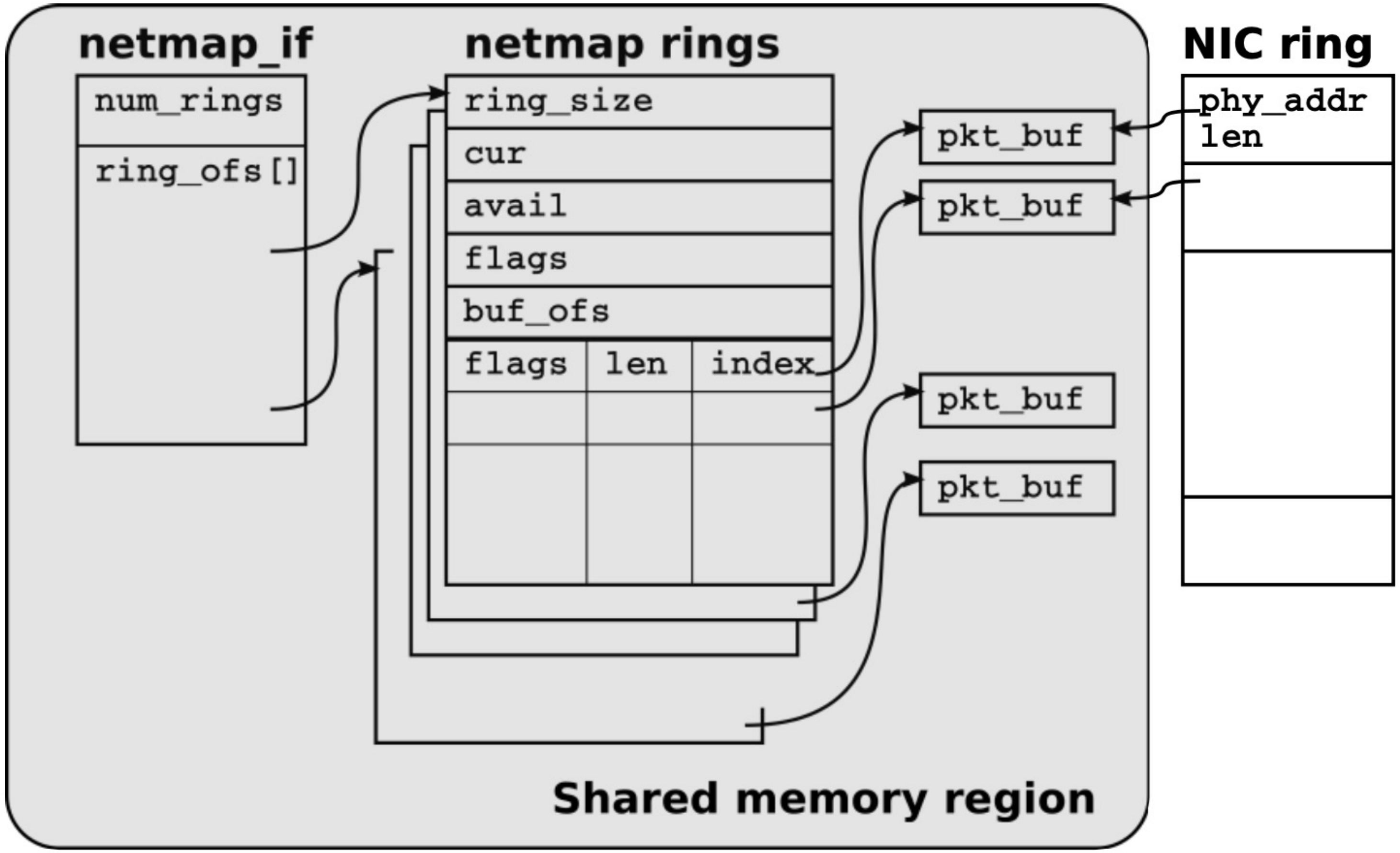
Overheads are sprinkled throughout the packet processing stack.

Netmap ATC12.

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	

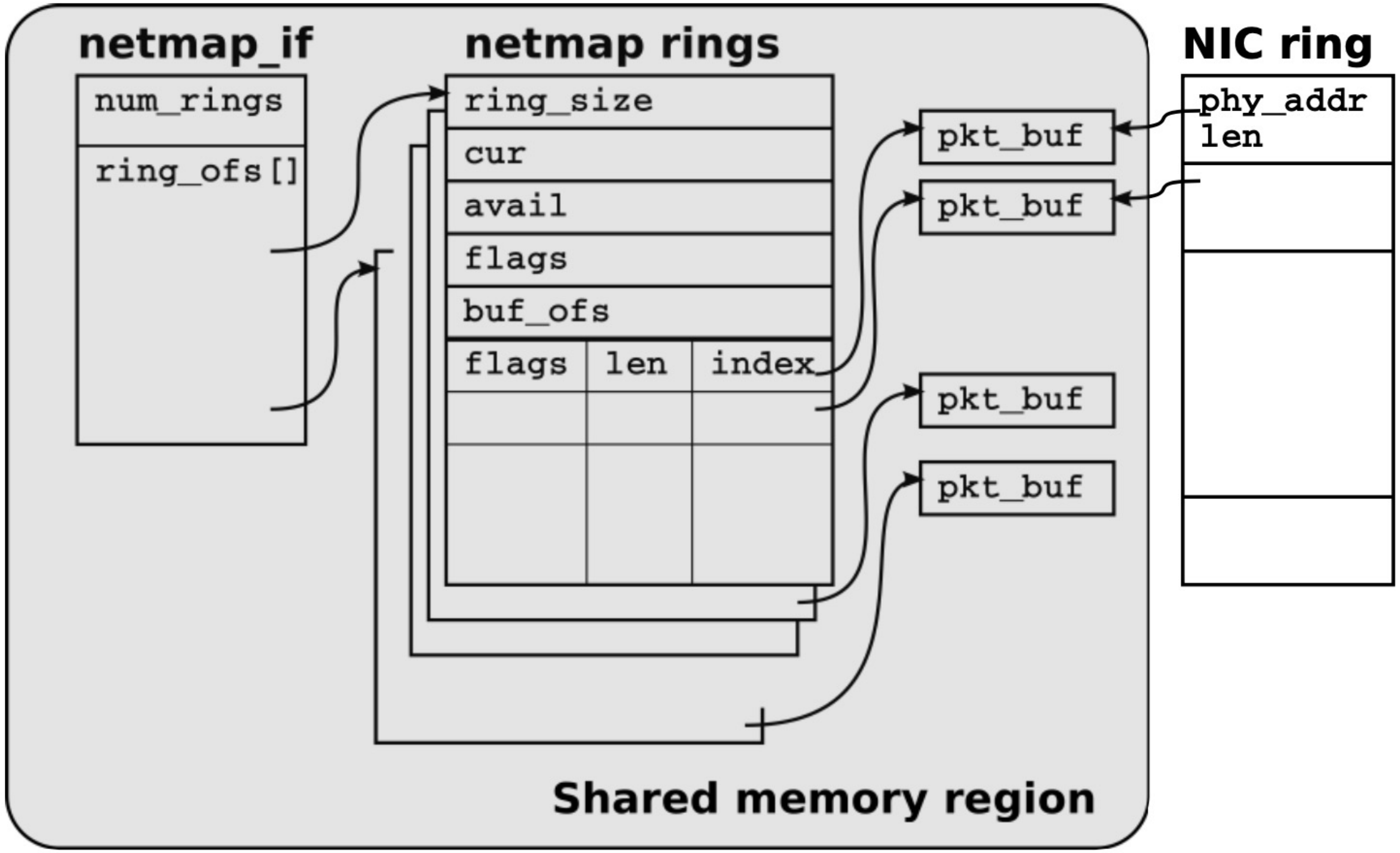
(1) Shared memory: avoid per-byte costs

- Remove user-kernel data copies
- Other systems use similar ideas:
- Finish processing entirely within the kernel (e.g., click-kernel, eBPF)
- Expose kernel buffers directly to user space (PF_RING)



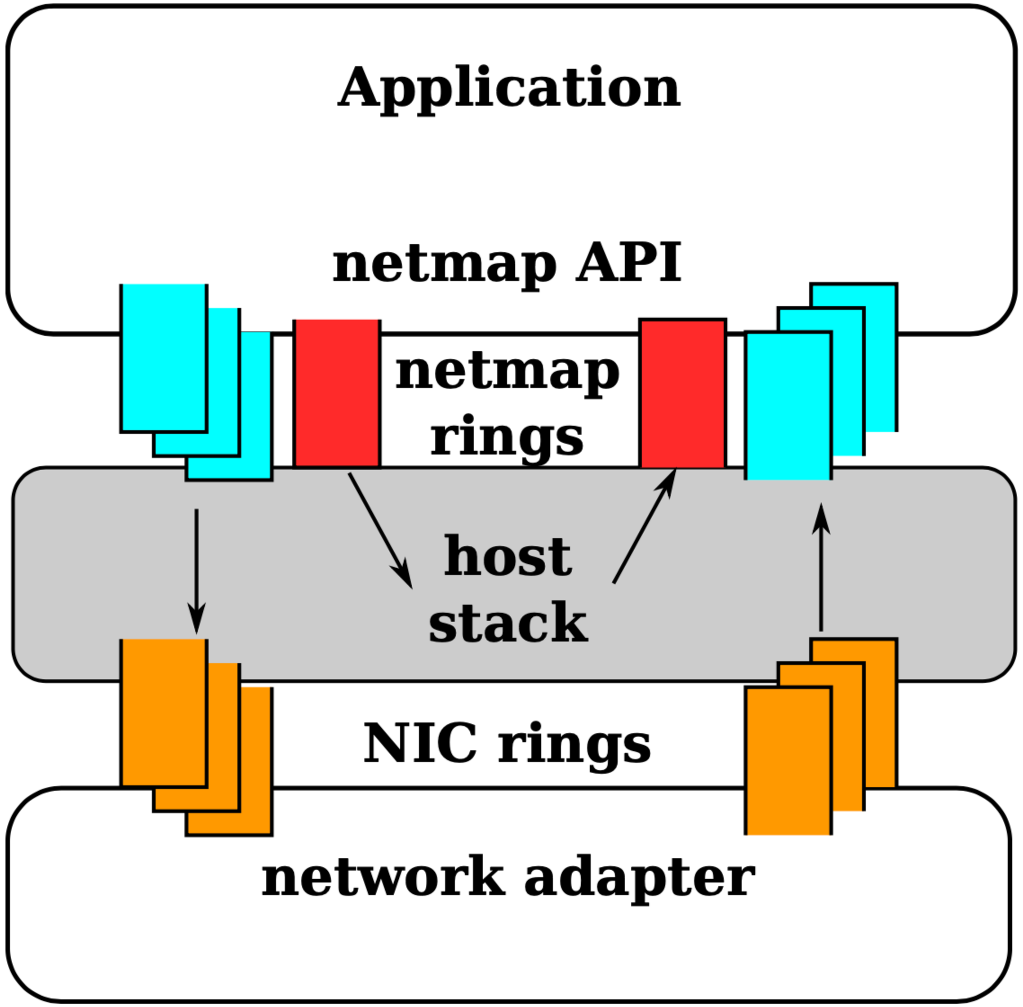
(2) Data representation: pre-allocated fixed size buffers and rings

- Avoid per-byte costs by pre-allocating chunks of a fixed size (max packet size)
- No allocation and freeing mbuf/sk_buff at run time



(3) NIC/netmap ring separation

- Validate netmap ring inputs provided by applications
- System call still needed to copy netmap ring descriptor to NIC ring descriptor (per-packet operation)
- Some systems avoid even this (DPDK, PF-RING, Solarflare openonload) by having apps directly program NIC rings (security & fault implications)



(4) Amortize system calls by batching

- Notify the kernel about packets written for transmission or available for receiption

```
ioctl(.., NIOCTXSYNC)

ioctl(.., NIOCRXSYNC)

select()/poll()
```

Pkt gen

- Associate shared buffers with fd's
- Poll file descriptor
- Walk through the netmap ring to identify available packet buffers. Write and notify
- Poll automatically synchronizes rings. No more system calls needed

```
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);
    }
  }
}
```

DPDK basic forwarding

```
for (;;) {
	/*
	 * Receive packets on a port and forward them on the paired
	 * port. The mapping is 0 -> 1, 1 -> 0, 2 -> 3, 3 -> 2, etc.
	 */
	RTE_ETH_FOREACH_DEV(port) {

		/* Get burst of RX packets, from first port of pair. */
		struct rte_mbuf *bufs[BURST_SIZE];
		const uint16_t nb_rx = rte_eth_rx_burst(port, 0,
				bufs, BURST_SIZE);

		if (unlikely(nb_rx == 0))
			continue;

		/* Send burst of TX packets, to second port of pair. */
		const uint16_t nb_tx = rte_eth_tx_burst(port ^ 1, 0,
				bufs, nb_rx);

		/* Free any unsent packets. */
		if (unlikely(nb_tx < nb_rx)) {
			uint16_t buf;
			for (buf = nb_tx; buf < nb_rx; buf++)
				rte_pktmbuf_free(bufs[buf]);
		}
	}
}
```

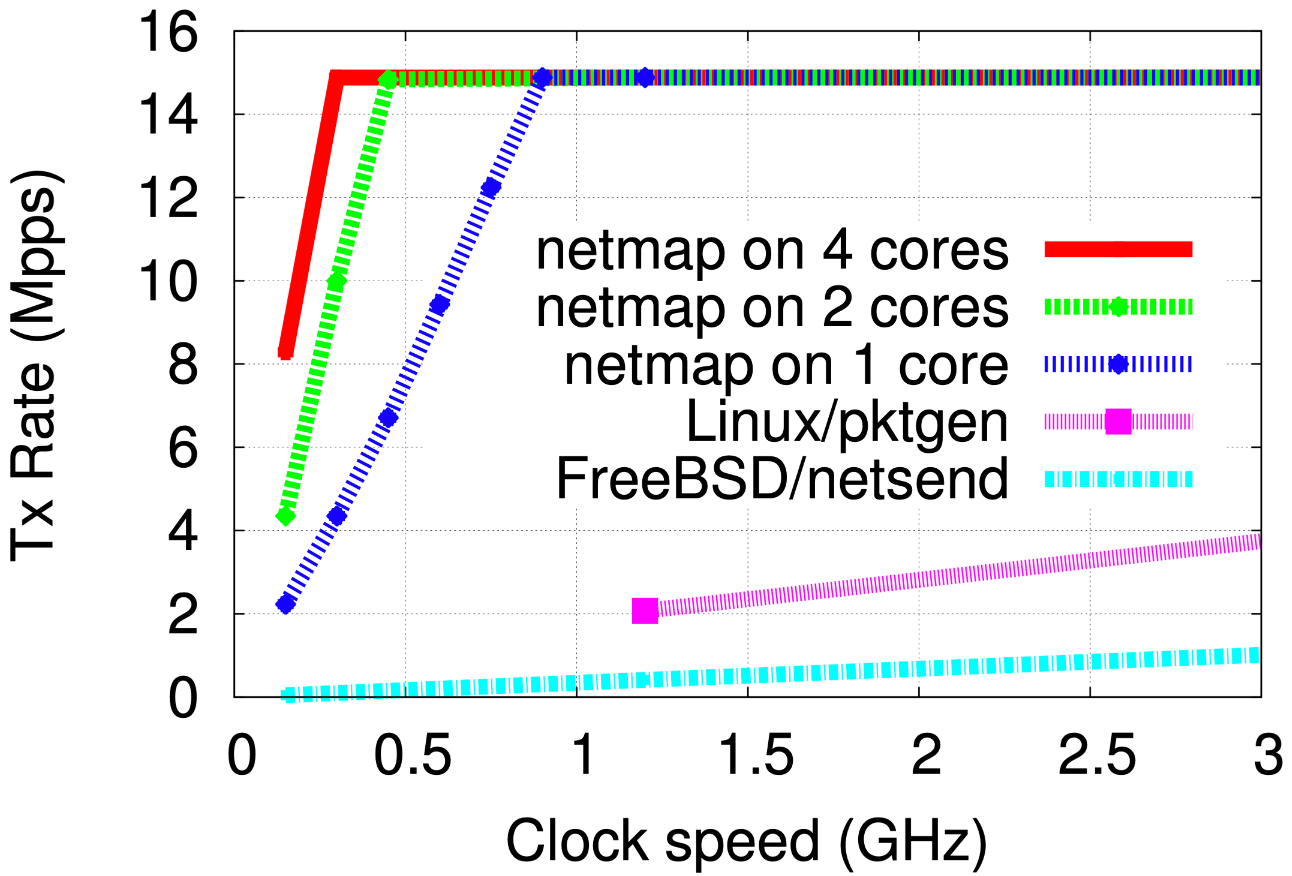
Forwarding between two interfaces

- Move descriptors, no data copies

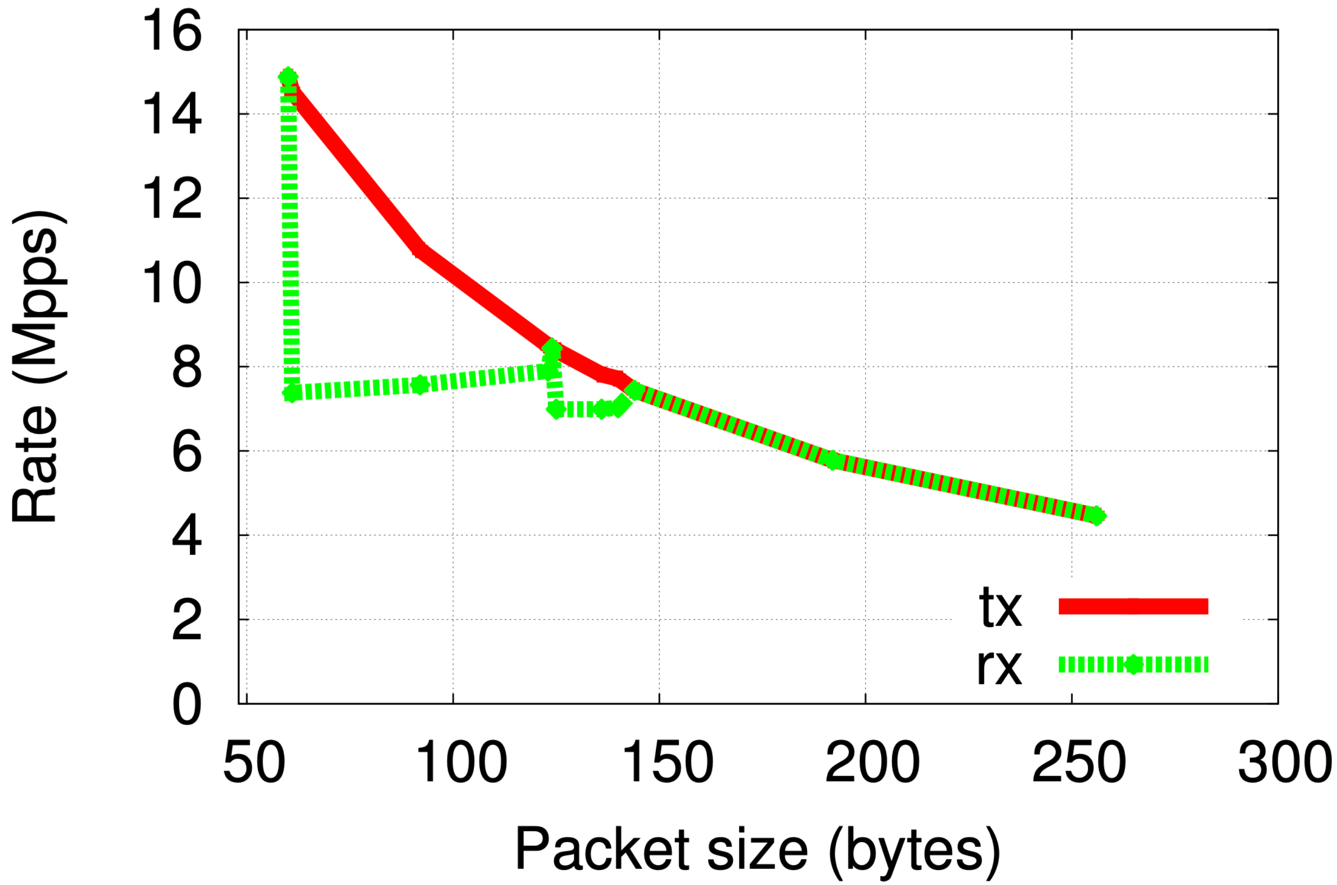
```
...
src = &src_nifp->slot[i]; /* locate src and dst slots */
dst = &dst_nifp->slot[j];
/* swap the buffers */
tmp = dst->buf_index;
dst->buf_index = src->buf_index;
src->buf_index = tmp;
/* update length and flags */
dst->len = src->len;
/* tell kernel to update addresses in the NIC rings */
dst->flags = src->flags = BUF_CHANGED;
...
```

Performance (pkt gen throughput)

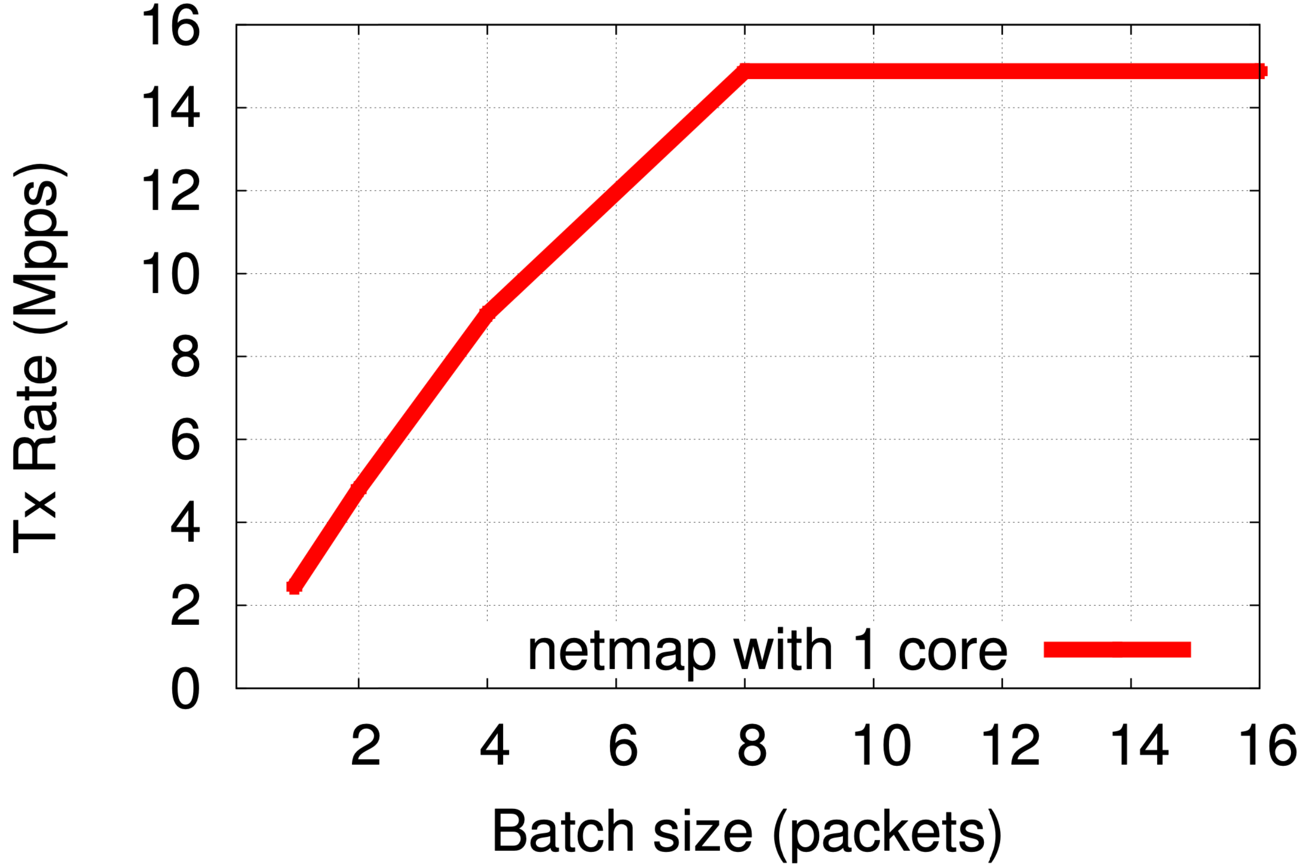
- Vary clock rate to make the workload CPU bound



Varying packet size



Performance with batching



Outlook: fast packet processing

- Get rid of software if you can
- Application-kernel API change: application must be modified
- Device drivers must often be modified
- Utilities in the host networking stack?
  - Libpcap, Netfilter, Routing, Socket lookup/packet demuxing?
- Multitenancy: serious implications to weakening fault isolation
- Can we get isolation with efficiency?