netmapによる
実践パケット処理プログラミング

ryo@iij.ad.jp

netmapとは?

- a framework for fast packet I/O
  - ピサ大学のLuigi Rizzo 教授が設計したAPI
  - 送受信パケット用のバッファを予め確保
  - userland/kernelでその領域をmmapで共有
  - index付きring bufferを使うことによるパケット領域の高速swap

そもそも（＊BSDで）パケットを入出力する方法

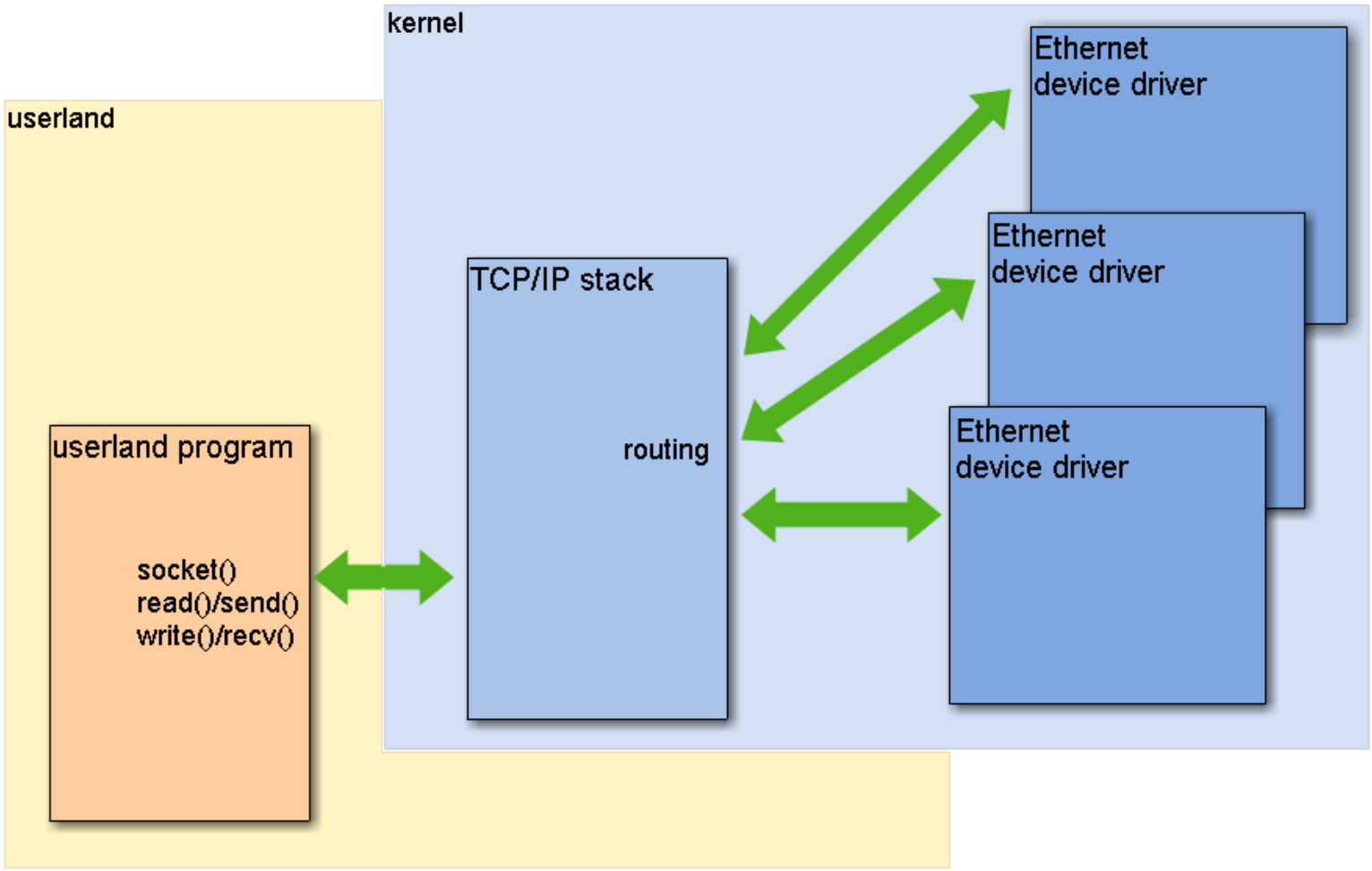
- socket
  - socket(), send()/write(), recv()/read()
- raw-socket
  - socket(AF_INET, SOCK_RAW, …)
- bpf
  - open(“/dev/bpf”, …)
- pfil
  - pfil_add_hook() (kernel)

netmapの特徴

- bpfとpfilの中間くらいに位置する？（bpfとpfilの良いとこ取り）
- userlandで動く
  - /dev/netmap を開いてioctl等で設定
- input/output を横取り/ブロック/書き換えたりできる
- kernel←→userlandでメモリコピーが無いので効率が良い

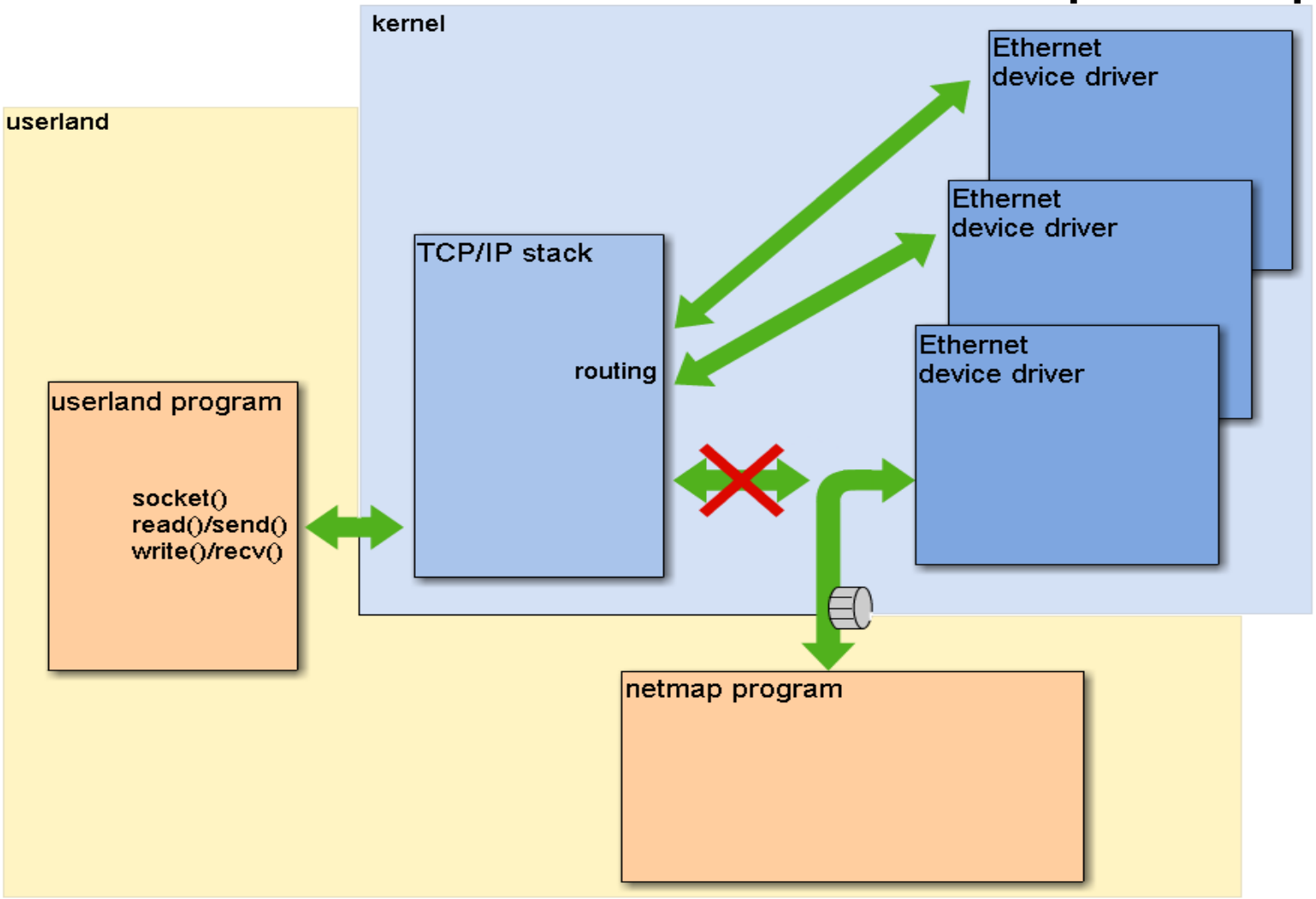
netmapは何ができるのか

- 通常のパケットのフロー



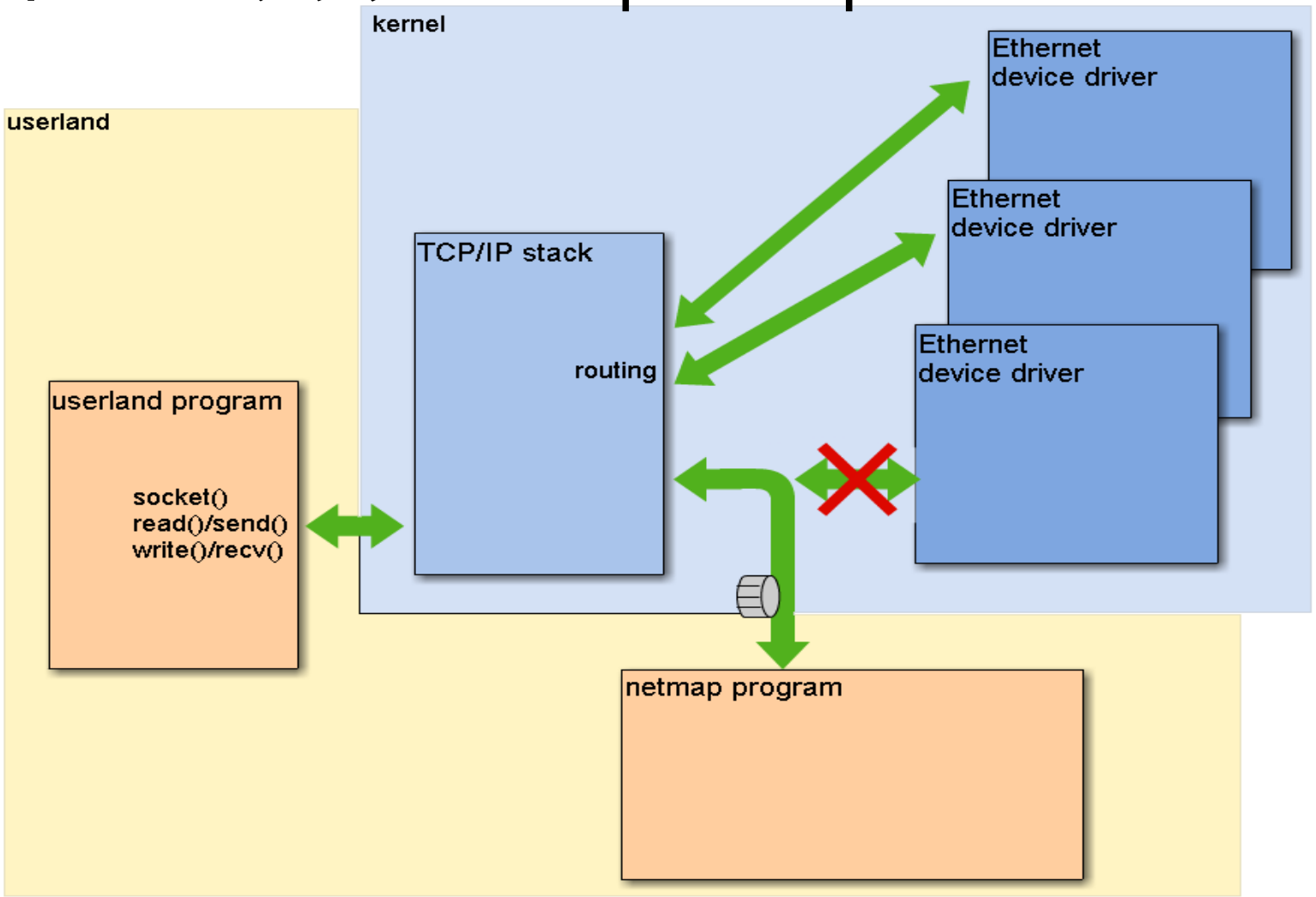
netmapは何ができるのか

- ネットワークインターフェイスへのinput/output



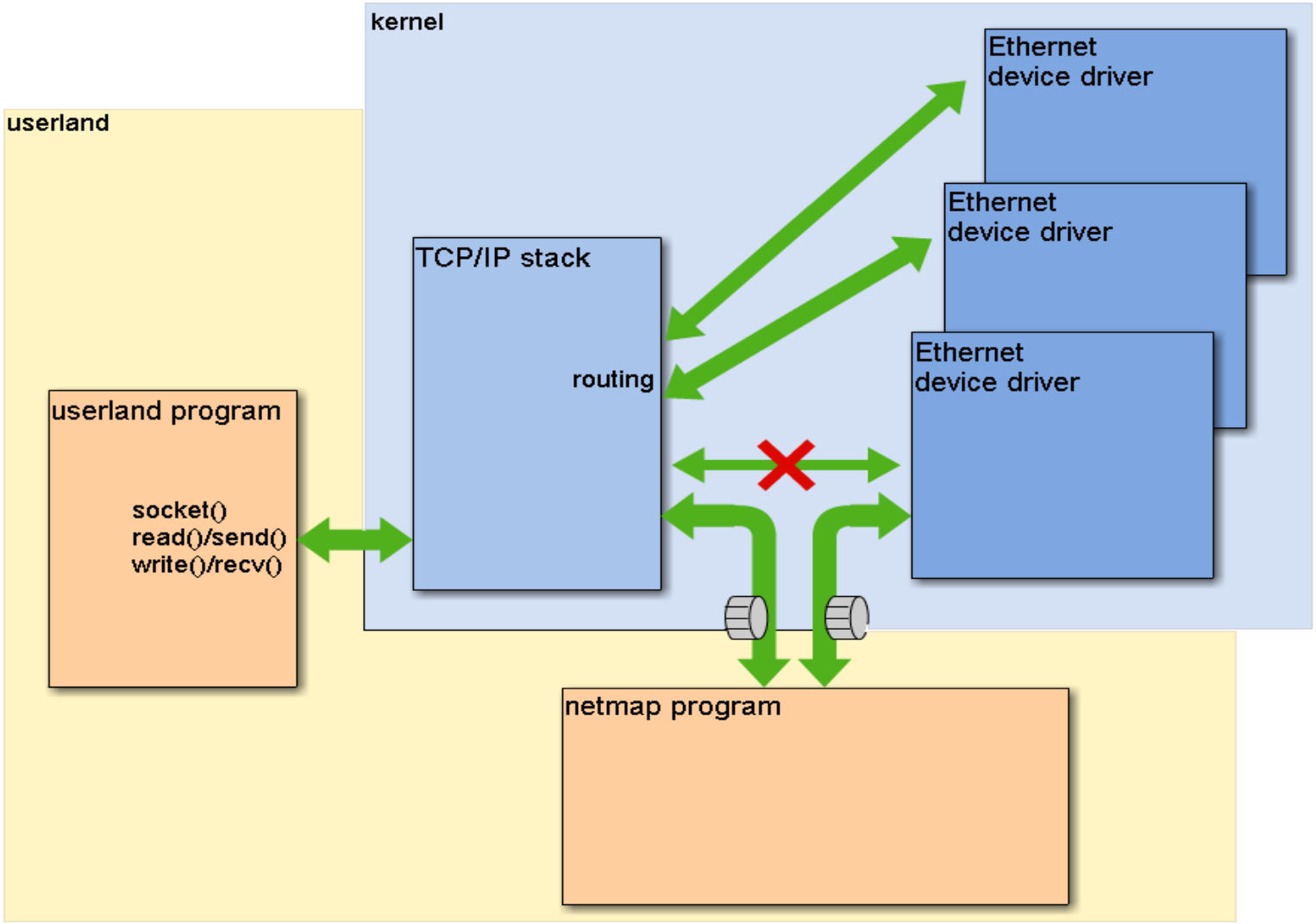
netmapは何ができるのか

- ホストスタックへのinput/output



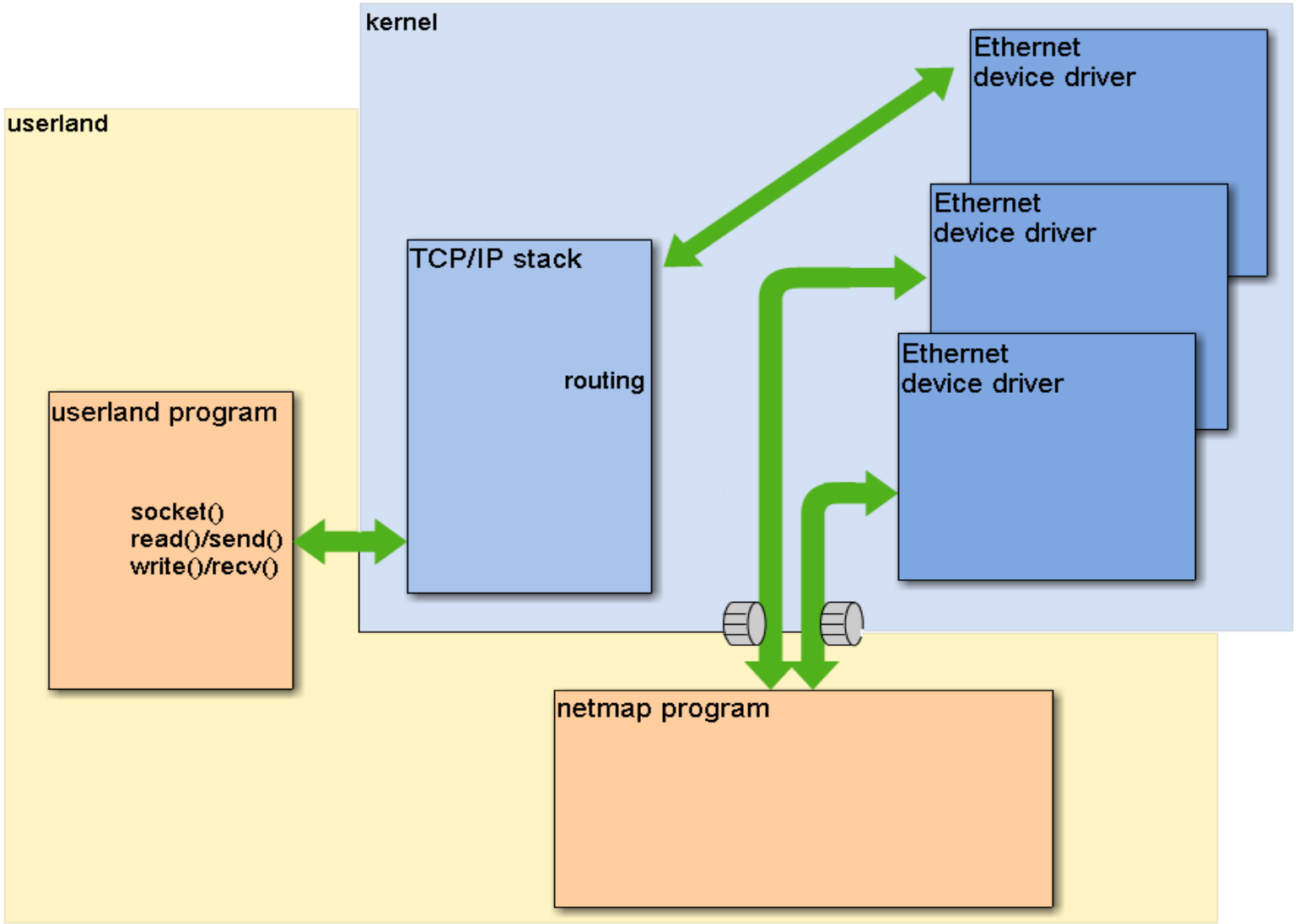
netmapは何ができるのか

- インターフェイスからホストスタックへの転送



netmapは何ができるのか

- インターフェイスからインターフェイスの転送



Hello ~~World~~ Packet

https://github.com/ryo/netmap_sample/ … 01_hexdump

```c
#include <poll.h>
#include <libutil.h>
#define NETMAP_WITH_LIBS
#include <net/netmap_user.h>

struct nm_desc *nm_desc;

int
main(int argc, char *argv[])
{
        unsigned int cur, n, i;
        struct netmap_ring *rxring;
        struct pollfd pollfd[1];

        nm_desc = nm_open("netmap:igb2", NULL, 0, NULL);
        for (;;) {
                pollfd[0].fd = nm_desc->fd;
                pollfd[0].events = POLLIN;
                poll(pollfd, 1, 100);

                for (i = nm_desc->first_rx_ring; i <= nm_desc->last_rx_ring; i++) {
                        rxring = NETMAP_RXRING(nm_desc->nifp, i);
                        cur = rxring->cur;
                        for (n = nm_ring_space(rxring); n > 0; n--, cur = nm_ring_next(rxring, cur)) {
                                hexdump(NETMAP_BUF(rxring, rxring->slot[cur].buf_idx), rxring->slot[cur].len, NULL, 0);
                        }
                        rxring->head = rxring->cur = cur;
                }
        }
}
```

パケットをdumpする（だけ）

Hello Packet (1)

- #include <net/netmap_user.h>
  - #define NETMAP_WITH_LIBS してから #include <net/netmap_users.h> すると、便利なマクロやinline関数等が定義される
    - nm_open() / nm_close()
      - 特にこの nm_open() はめんどくさい初期設定を引き受けてくれるので便利。
    - 他にもちょっと便利な関数（今回は未使用）
      - nm_inject()　　パケットを出力する
      - nm_dispatch()　パケット受信処理をcallbackで呼んでくれる
      - nm_nextpkt()　　ringバッファのポインタを次のパケットへと進める

Hello Packet (2)

nm_desc = nm_open("netmap:igb0", NULL, 0, NULL);

/dev/netmapをopenし、ioctlでインターフェイス等を設定、ringバッファをmmapし、初期化してくれる。

“netmap:igb0” ・・・NICのハードウェアTX/RX ringを開く
“netmap:igb0^” ・・・NICに対応するホストリングのTX/RXを開く
“netmap:igb0”* ・・・ハードウェアリングとホストリング両方を開く
“netmap:igb0-1” ・・・NICの1番目のハードウェアリングだけを開く

・・・他にもいろいろ。詳しくは net/netmap_user.h を参照

Hello Packet (3)

```
for (;;) {
        pollfd[0].fd = nm_desc->fd;
        pollfd[0].events = POLLIN;
        poll(pollfd, 1, 100);

        ～～～～～～～～～

}
```

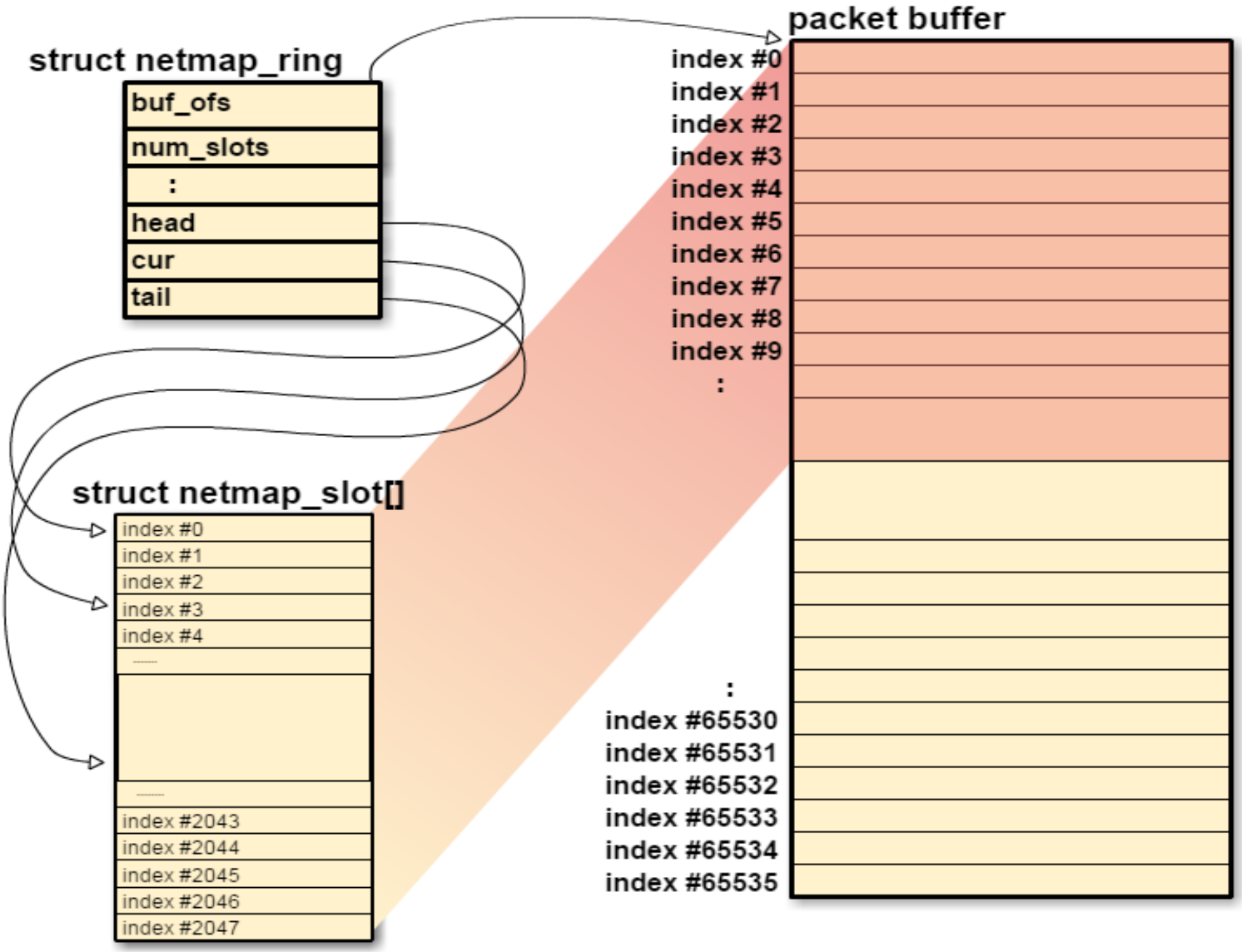
netmapは基本的には poll() で待つ。
poll() で待ってる間にkernelのnetmapドライバがパケットを入出力してくれる。

Hello Packet (4)

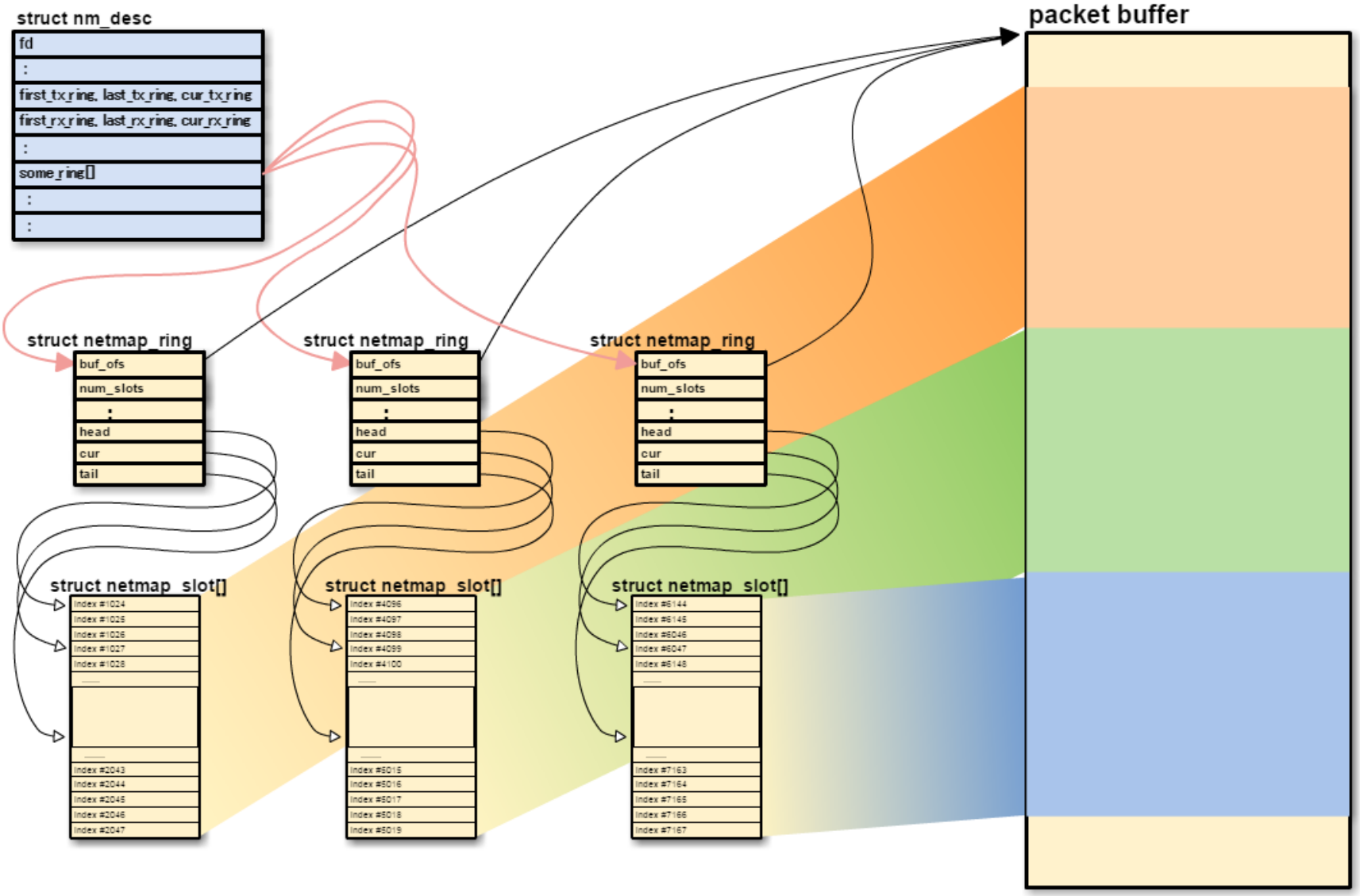
```
for (i = nm_desc->first_rx_ring; i <= nm_desc->last_rx_ring; i++) {

        rxring = NETMAP_RXRING(nm_desc->nifp, i);
        cur = rxring->cur;
        for (n = nm_ring_space(rxring); n > 0; n--) {
                hexdump(NETMAP_BUF(rxring, rxring->slot[cur].buf_idx),
                        rxring->slot[cur].len, NULL, 0);
                cur = nm_ring_next(rxring, cur);
        }
        rxring->head = rxring->cur = cur;
}
```

RX ringから受信パケットを1パケットづつ取り出してhexdumpする。
最初のループは、NICによってはringが複数ある場合のため。

netmap ringの構造

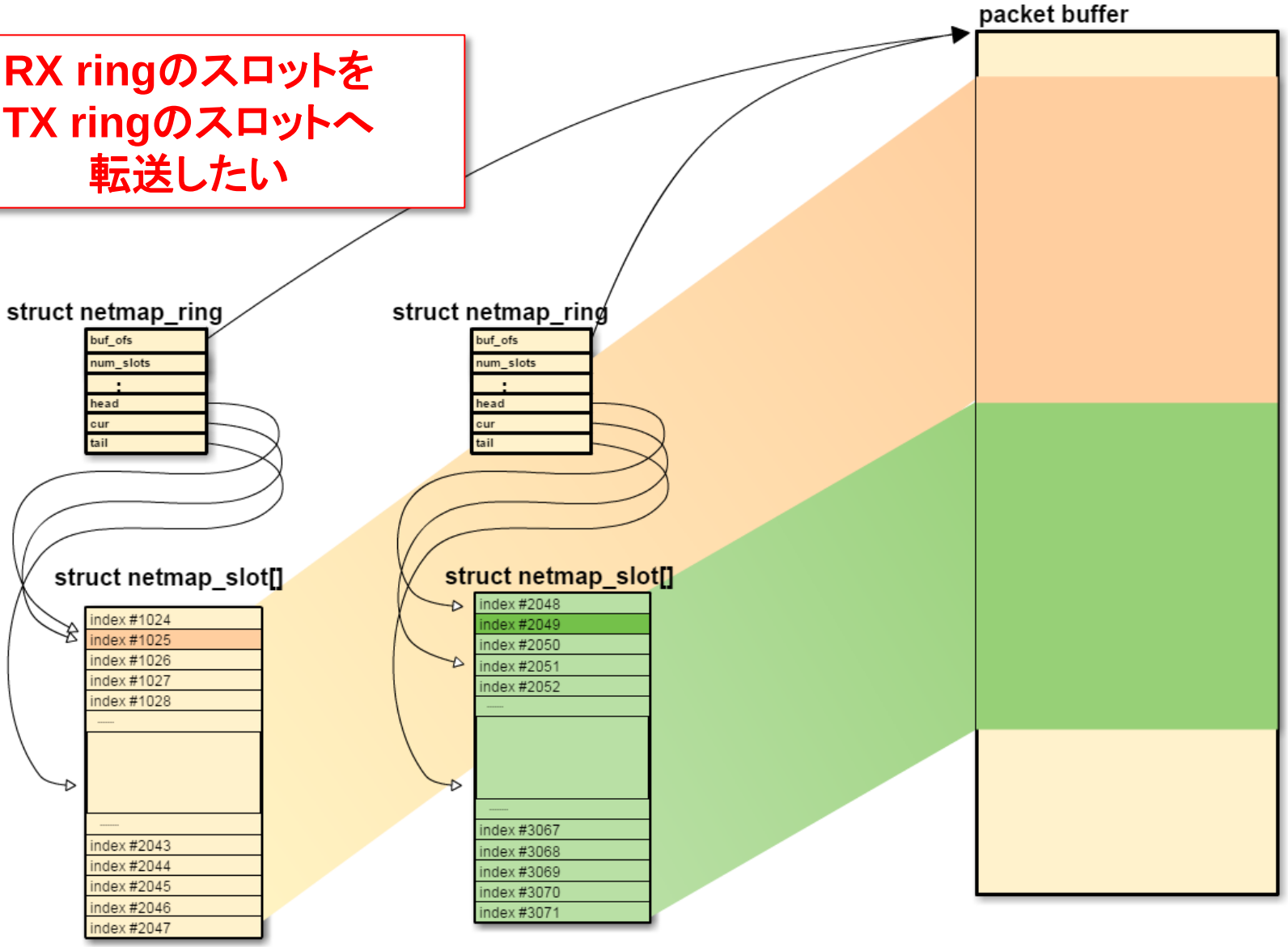


netmap ringの構造（全体）



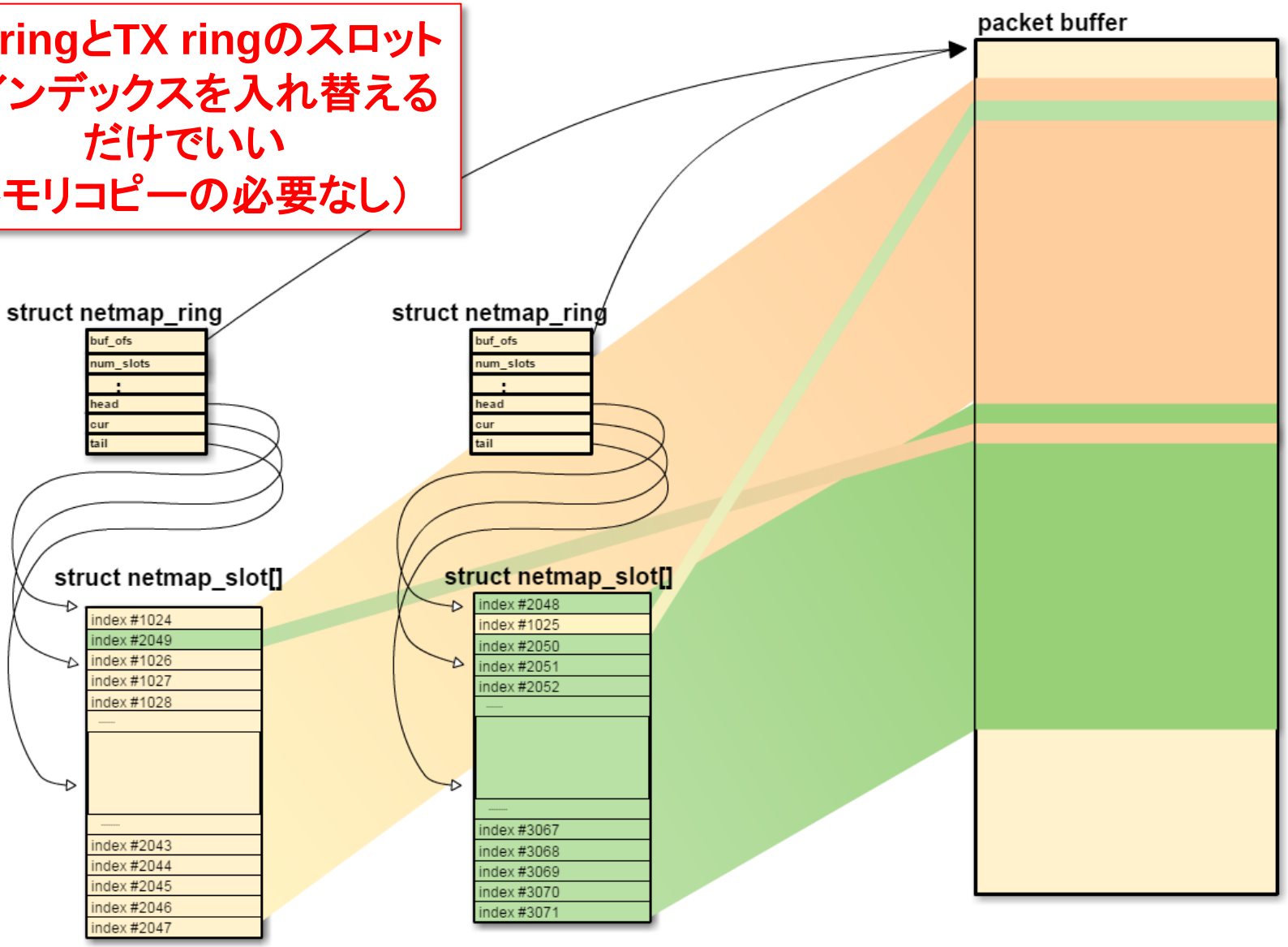
RX ringからTX ringへの転送

**RX ringのスロットを
TX ringのスロットへ
転送したい**



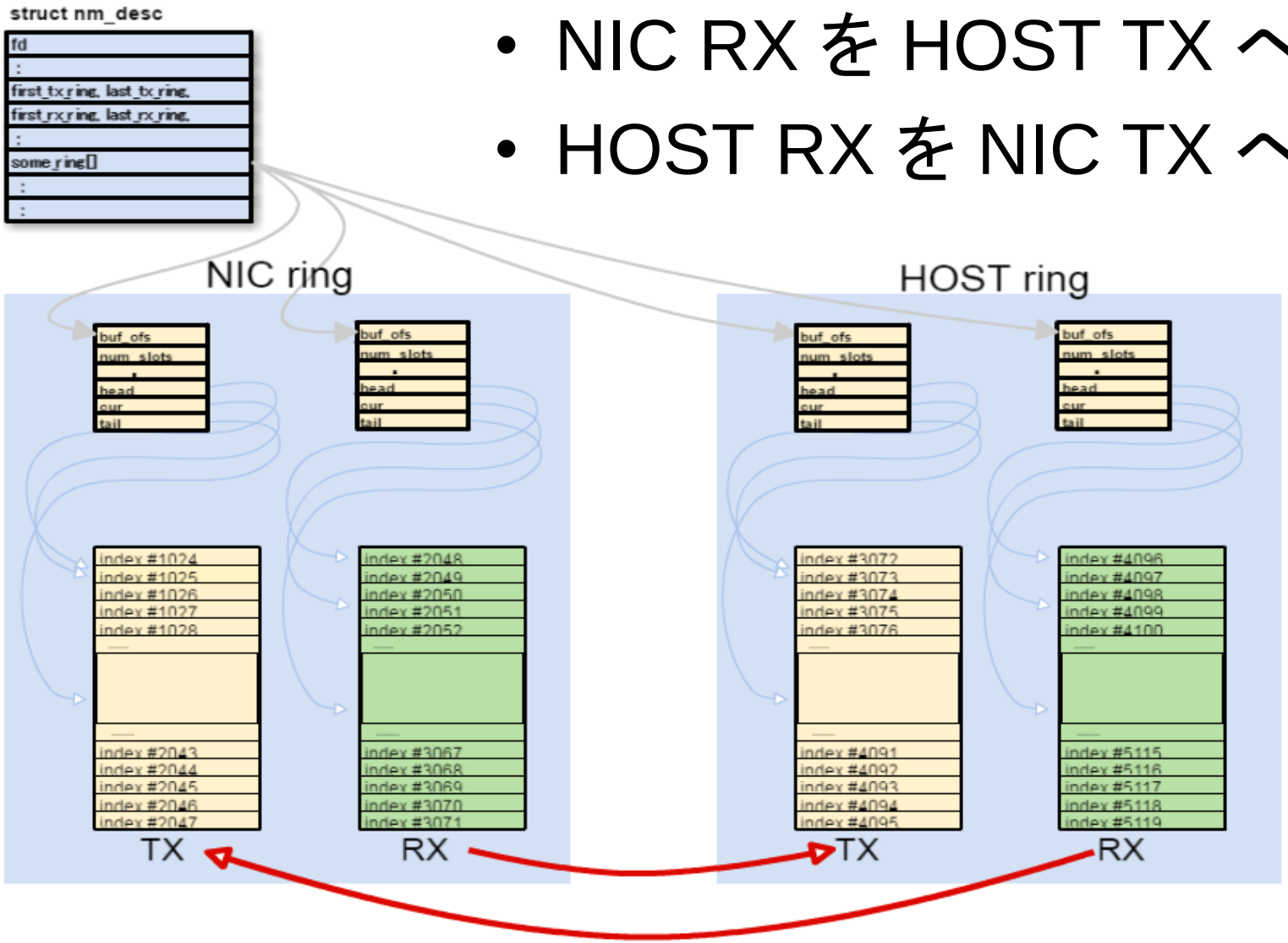
RX ringからTX ringへの転送

RX ringとTX ringのスロットのインデックスを入れ替えるだけでいい(メモリコピーの必要なし)



NIC ring ←→ HOST ring

- NIC RX を HOST TX へ
- HOST RX を NIC TX へ



netmap_slotのswap

https://github.com/ryo/netmap_sample/ … 02_nic2host

indexだけを入れ替える

```
/* swap buf_idx */
tmp = txring->slot[cur].buf_idx;
txring->slot[cur].buf_idx = rxring->slot[src].buf_idx;
rxring->slot[src].buf_idx = tmp;
```

sizeは上書きでok

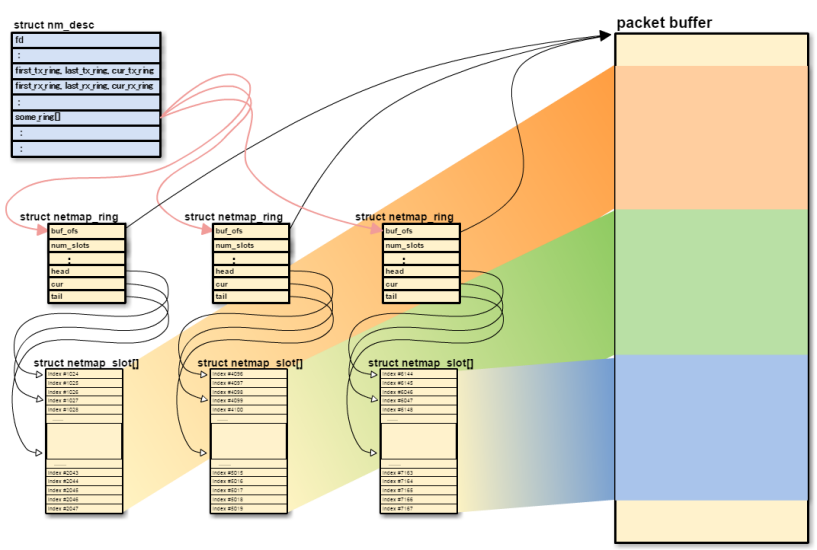
```
/* set len */
txring->slot[cur].len = rxring->slot[src].len;
```

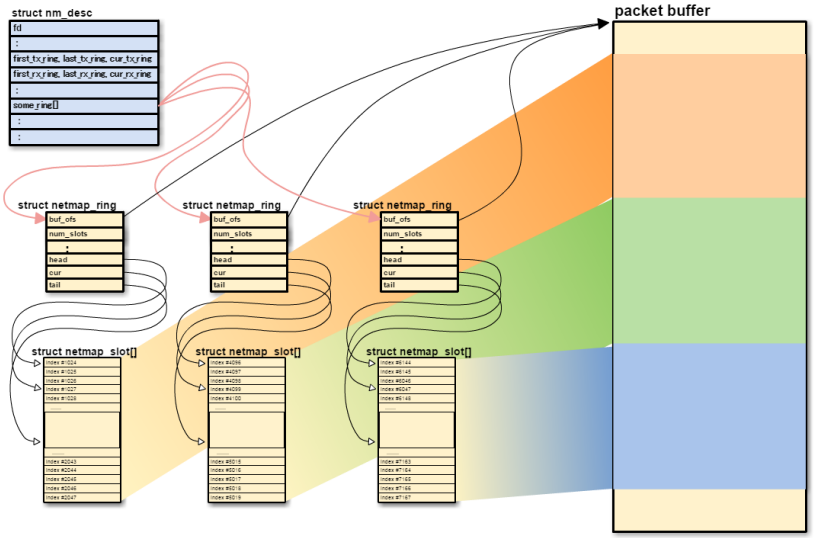
indexを書き換えた場合に必要

```
/* update flags */
txring->slot[cur].flags |= NS_BUF_CHANGED;
rxring->slot[src].flags |= NS_BUF_CHANGED;
```

NICからNICへの転送(複数NIC)

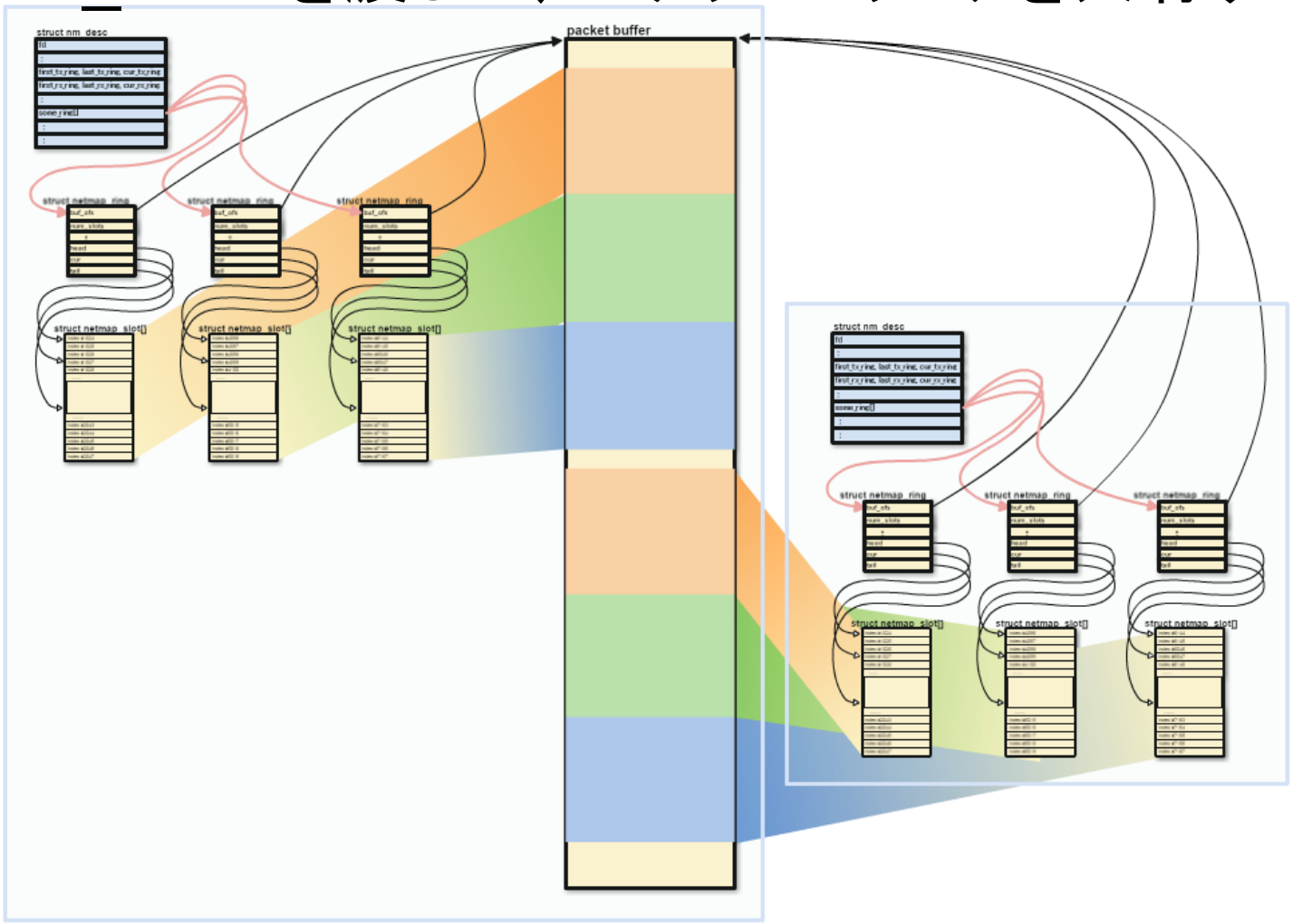
- nm_open() で複数インターフェイスを開くと…
  - packet bufferが独立してしまう
    → ring buffer swapできない？！





NICからNICへの転送(複数NIC)

- 2つめの nm_open() に、親として1つめの nm_desc を渡して、パケットバッファを共有する

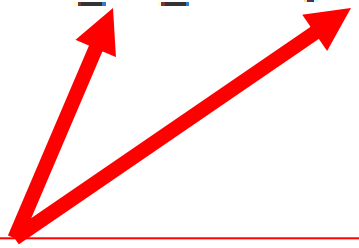


nm_desc1 = nm_open(..., NULL) → nm_open(..., nm_desc1)

NICからNICへの転送

https://github.com/ryo/netmap_sample/ … 03_nic2nic

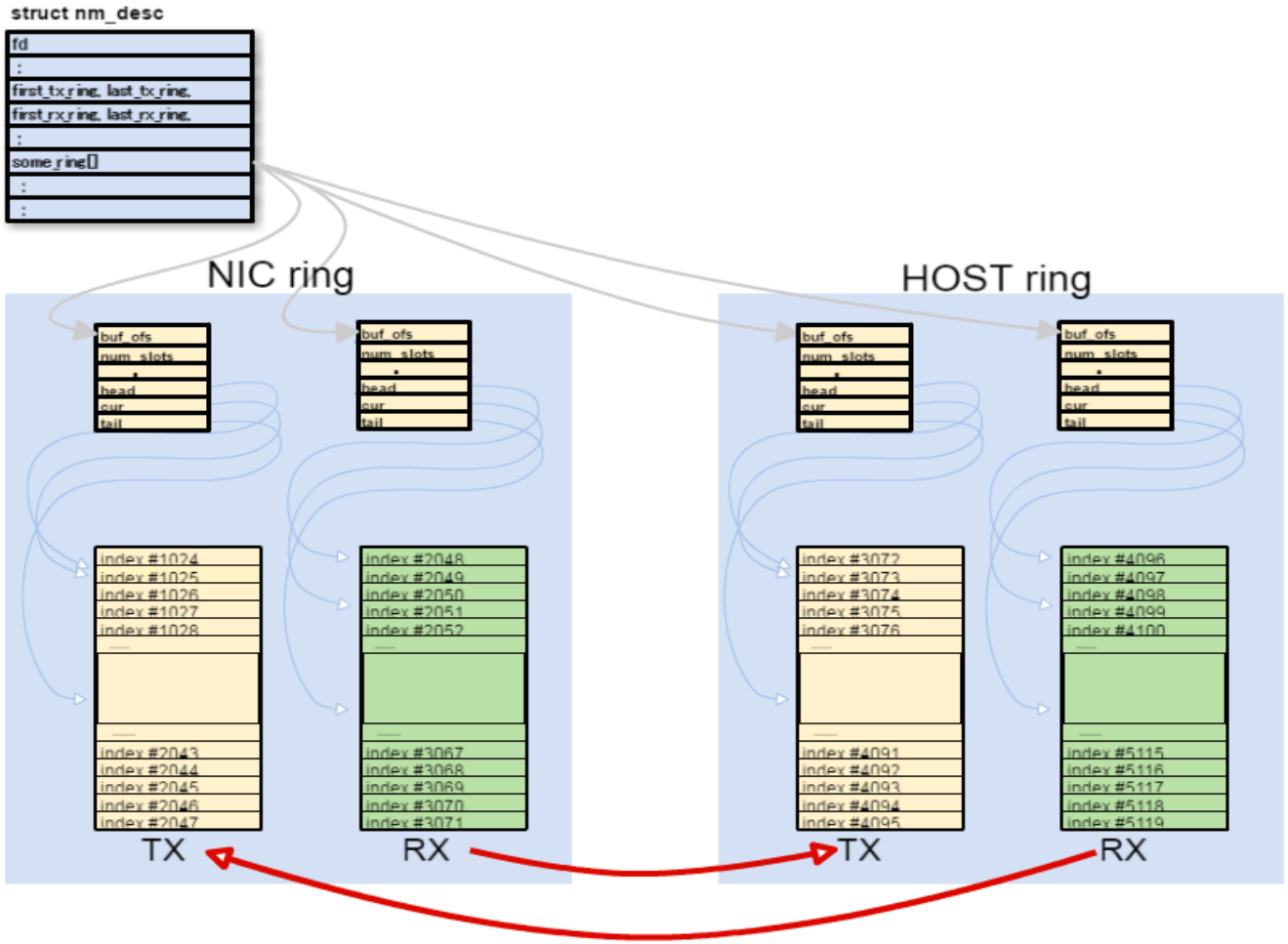
```
fprintf(stderr, "in advance, 'ifconfig igb2 promisc' and "
                            "'ifconfig igb3 promisc' for bridge\n");

nm_desc1 = nm_open("netmap:igb2", NULL, 0, NULL);
nm_desc2 = nm_open("netmap:igb4", NULL, NM_OPEN_NO_MMAP, nm_desc1);
```



NM_OPEN_NO_MMAP と、
親として nm_desc1 を指定することにより、
2つの netmap ディスクリプタで
パケットバッファが共有される

応用編

- 簡易firewall



ここでTXに渡す/渡さないを取捨選択する

簡易firewall

https://github.com/ryo/netmap_sample/ … 04_firewall

```
/* last ring is host ring */
is_hostring = (i == nm_desc->last_rx_ring);
rxring = NETMAP_RXRING(nm_desc->nifp, i);

cur = rxring->cur;

for (n = nm_ring_space(rxring); n > 0; n--, cur = nm_ring_next(rxring, cur)) {


        /* test packet. block when return !0 */
        if (packetfilter(is_hostring,
                        NETMAP_BUF(rxring,rxring->slot[cur].buf_idx),
                        rxring->slot[cur].len)) {
                printf("BLOCK:\n");
                hexdump(NETMAP_BUF(rxring, rxring->slot[cur].buf_idx),
                        rxring->slot[cur].len, "  ", 0);
                continue;       /* discard */
        }
        swapto(!is_hostring, &rxring->slot[cur]);

}
rxring->head = rxring->cur = cur;
```

簡易firewall

https://github.com/ryo/netmap_sample/ … 04_firewall

```
static int
packetfilter(int dir, void *buf, unsigned int len)
{
        char *payload;
        struct ether_header *ether;
        struct ip *ip;
        struct tcphdr *tcp;
        struct udphdr *udp;
        int i;

        ether = (struct ether_header *)buf;
        ip = (struct ip *)(ether + 1);
        payload = (char *)ip + ip->ip_hl * 4;

        if (ip->ip_v == IPVERSION) {
```

パケットはether header含む

ether headerから辿っていく

簡易firewall

https://github.com/ryo/netmap_sample/ … 04_firewall

フィルタルールの定義

```
struct filterrule {
        int dir;
        int proto;
        struct in_addr srcaddr, srcaddr_mask;
        struct in_addr dstaddr, dstaddr_mask;
        int srcport;
        int dstport;
} filterrule[] = {
        {
                .dir          = 0,              /* 0:in, 1:out */
                .proto        = IPPROTO_TCP,
                .srcaddr      = { 0x0a000000 }, /* 10.0.0.0 */
                .srcaddr_mask = { 0xff000000 }, /* 255.0.0.0 */
                .dstaddr      = { 0x00000000 }, /* 0.0.0.0 */
                .dstaddr_mask = { 0x00000000 }, /* 0.0.0.0 */
                .srcport      = -1,             /* any */
                .dstport      = 80
        },
```

簡易firewall

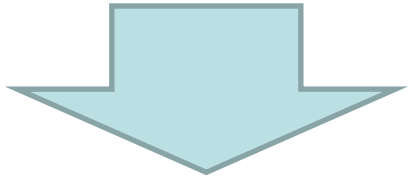
https://github.com/ryo/netmap_sample/ … 04_firewall

```
if (ip->ip_v == IPVERSION) {
        for (i = 0; i < nitems(filterrule); i++) {
                if (filterrule[i].dir != dir)
                        continue;

                /* test protocol */
                if ((filterrule[i].proto != -1) && (filterrule[i].proto != ip->ip_p))
                        continue;

                /* test src addr */
                if ((filterrule[i].srcaddr.s_addr & filterrule[i].srcaddr_mask.s_addr) !=
                    (ntohl(ip->ip_src.s_addr) & filterrule[i].srcaddr_mask.s_addr))
                        continue;

                /* test dst addr */
                if ((filterrule[i].dstaddr.s_addr & filterrule[i].dstaddr_mask.s_addr) !=
                    (ntohl(ip->ip_dst.s_addr) & filterrule[i].dstaddr_mask.s_addr))
                        continue;

                /* test src/dst port */
                if (ip->ip_p == IPPROTO_TCP) {
                        tcp = (struct tcphdr *)payload;
```

フィルタルールを順番にテスト

NICのマルチキュー

- CPU coreはたくさんある
- パケット処理は1 core（1プロセス）で動く
- 1つのcoreのCPU使用量だけ増え、
  他のcoreは処理空き
- パケットの処理を分散させられないか？



- NICに複数のキューを持たせればいい
  →NIC マルチキュー
  (RSS:Receive Side Scaling)

NICのマルチキューとnetmap

- Intel等のNICが対応している。
  キューの数は4、8、…
- netmapでは、キューの数の分のringが見える。（デバイスドライバにもよる）
- netmapでキューを独立に操作するには…？
  - nm_openで指定するデバイス名の後ろに数字を付ける

    “netmap:igb0” ・・・NICのハードウェアTX/RX ringを開く

    “netmap:igb0^” ・・・NICに対応するホストリングのTX/RXを開く

    “netmap:igb0”* ・・・ハードウェアリングとホストリング両方を開く

    ***“netmap:igb0-1”*・・・NICの1番目のハードウェアリングだけを開く**

マルチプロセッサにおけるチューニング

https://github.com/ryo/netmap_sample/ … 05_multiqueue

```
snprintf(buf, sizeof(buf), "netmap:%s", argv[1]);
nm_desc = nm_open(buf, NULL, 0, NULL);
nic_ring_num = nm_desc->nifp->ni_rx_rings;
nm_close(nm_desc);
```

一度 nm_open してRINGの数を調べる

RINGの数だけthreadを作る

```
for (i = 0; i < nic_ring_num; i++) {
        threadwork[i].no = i;

        snprintf(buf, sizeof(buf), "netmap:%s^", argv[1]);
        threadwork[i].nm_desc_host = nm_open(buf, NULL, 0, NULL);

        snprintf(buf, sizeof(buf), "netmap:%s-%d", argv[1], i);
        threadwork[i].nm_desc_nic = nm_open(buf, NULL,
            NM_OPEN_NO_MMAP, threadwork[i].nm_desc_host);

        pthread_create(&threadwork[i].thread, NULL,
            pthread_main, &threadwork[i]);
}
```

TIPS (あるいはバッドノウハウ)

- netmap対応ドライバが必要
  - 現状 em(4), igb(4), ixgbe(4), lem(4), re(4) のみ対応
- bridge等の動作をさせるには、別途promiscuousモードにするなどの操作が必要
  - プログラムで面倒みましょう
- ハードウェアオフローディングと相性が悪い
  - ifconfig –txcsum –tso 等しておく

Appendix

今回解説しなかったこと

- vale - a Virtual Local Ethernet と呼ばれる仮想イーサネットスイッチを持っている。“vale:foo” 等で作成・使用可能。
- netmap pipeと呼ばれるnetmap ringを使った1対1 pipe機能を持つ。
- bpfのように観測だけ行う MONITOR モードも持っている(NR_MONITOR_{TX,RX}

まとめ

- netmap_user.h の nm_open を使えば簡単に扱える
- netmap ring からパケットバッファへはインデックスを介して間接アクセスされる。それにより、メモリコピーせずにスロットのswapで高速に転送が可能
- 複数のNICを扱う場合は、nm_openに NM_OPEN_NO_MMAP を指定して親子関係を作ってパケットバッファを共有させる
- 高速化するには、ハードウェアRINGを別々にnm_openして、マルチキューマルチコア化