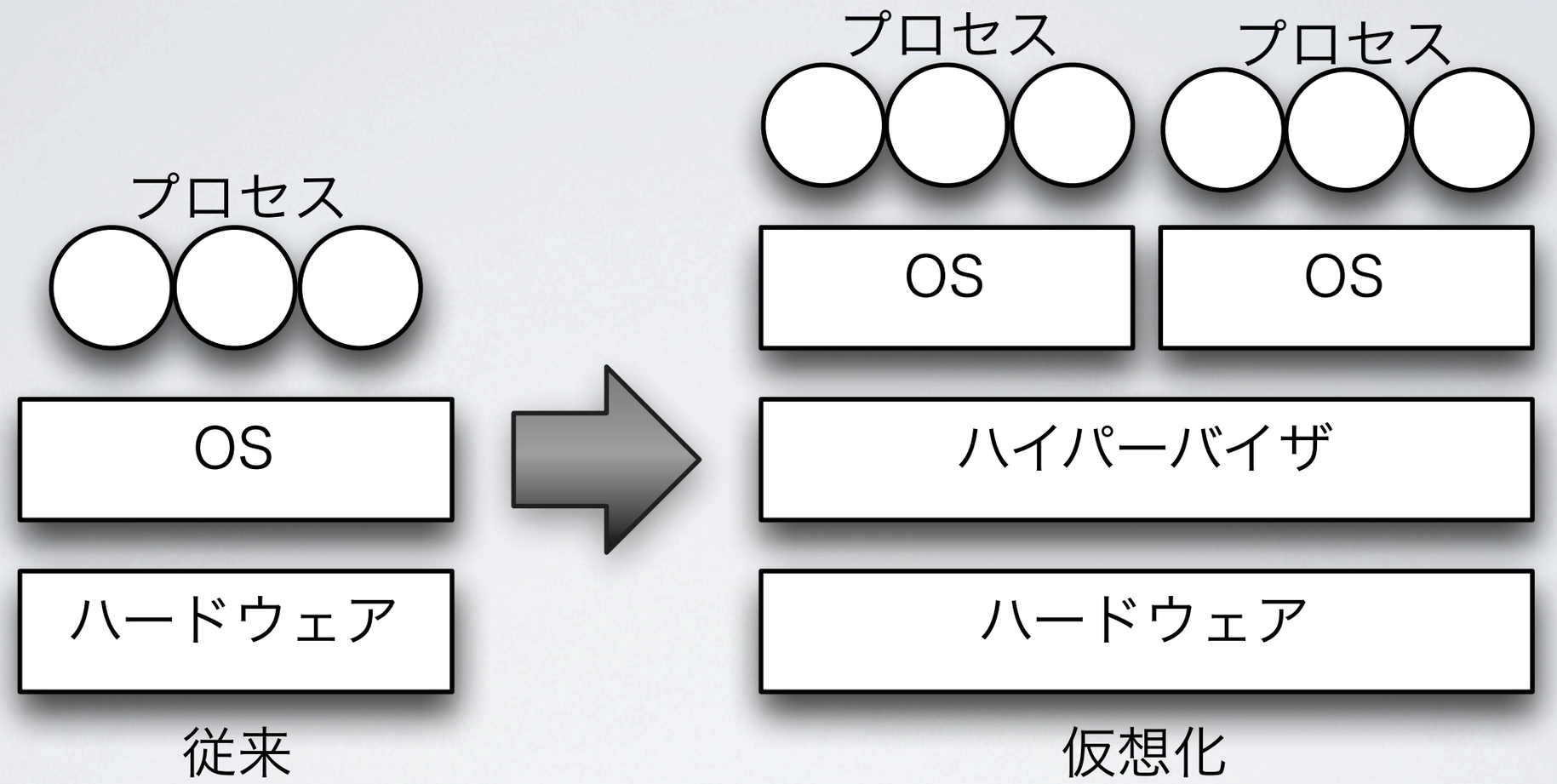
仮想化環境における
パケットフォワーディング

浅田 拓也 @ 東京工科大学

Twitter: @syuu1228

仮想化技術とは



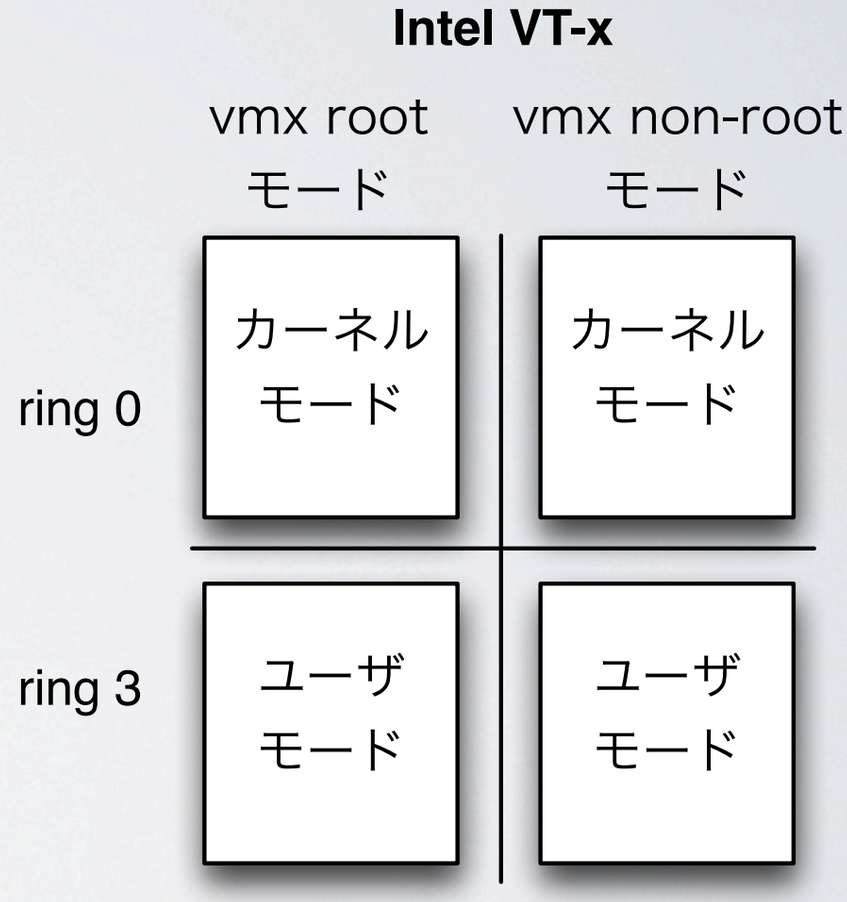
1台のコンピュータ上に複数の仮想的なコンピュータを動作させ、それぞれの上でOS・アプリケーションを実行

仮想化の手法

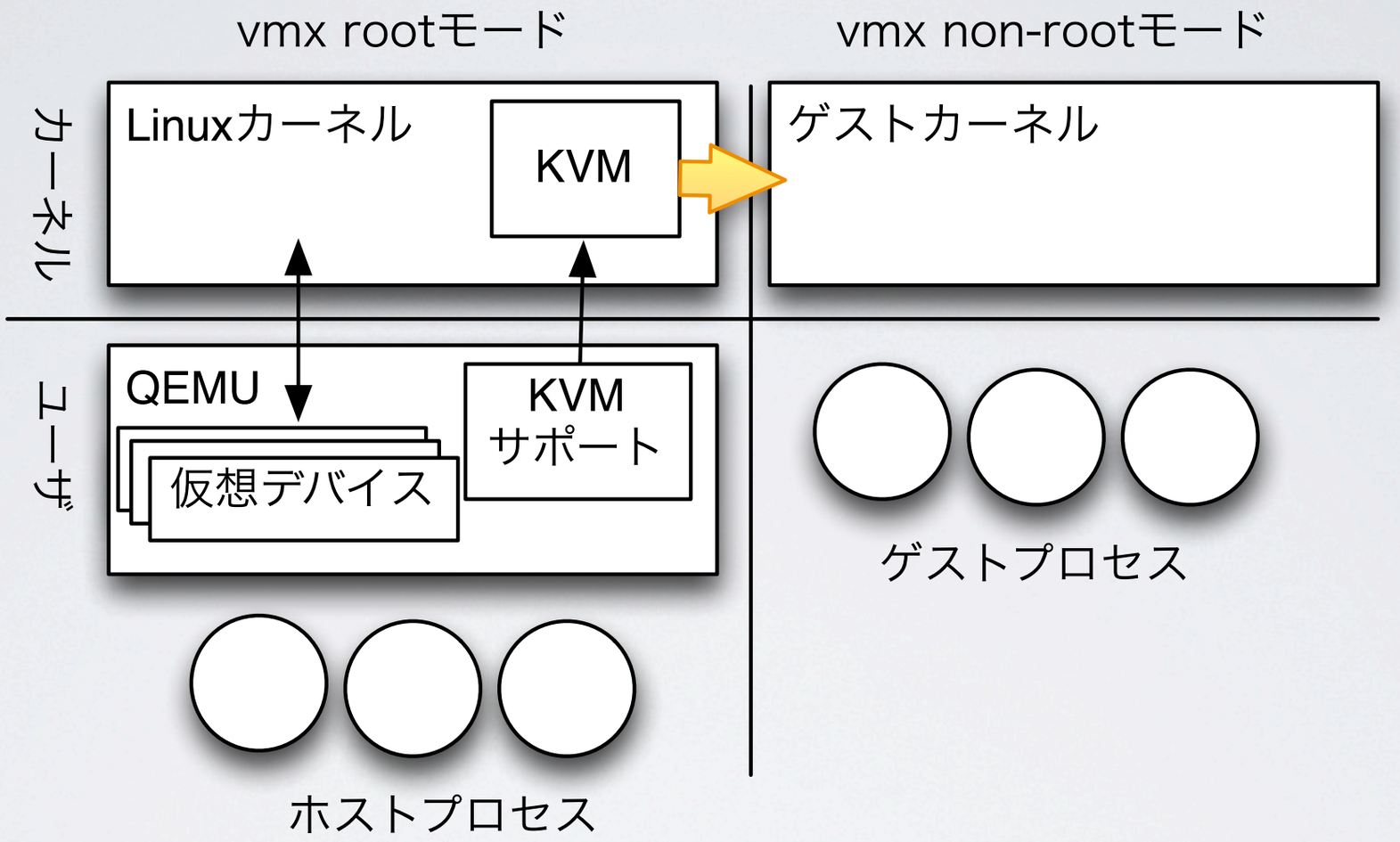
- **エミュレーション（QEMUなど）**
  - 仮想マシンで実行されるプログラムをソフトウェアで実装されたCPUで解釈して実行→遅い
- **ネイティブ仮想化（昔のVMwareなど）**
  - ハードウェアアクセスを伴う命令がきそうになったら回避してソフトウェアで処理しつつ、その他の部分は実CPUで直接実行
    →エミュレータよりずっと速いが、実装／高速化が困難
- **準仮想化（XenのdomUなど）**
  - ハイパーバイザの上で動かせるように、ゲストOSを改造する
    →高速でハードウェア支援も要らないが、ゲストOSの対応が必要（Windowsが動かない！）
- **ハードウェア支援による仮想化（Linux KVM、XenのHVM、最近のVMwareなど）**
  - ハイパーバイザモードとゲストモードの２つのモード切り替え機能をCPUに実装、
    ゲストOSでハイパーバイザの介入が必要な操作を行った時点で自動的にモードが切り替わる
    →ネイティブ仮想化よりも速く、実装が簡単

仮想化支援機構（INTEL VT）

- CPUにハイパーバイザを実行するモードとゲストOSを実行するモードを追加：Intel VT-x
- ハードウェアレベルで仮想化に対応する事により仮想化オーバヘッドを低減し、ハイパーバイザの実装を単純に出来る
- ハードウェア支援出来る範囲を広げつつある
  - メモリ管理の仮想化支援：EPT
  - デバイスIOの仮想化支援：Intel VT-d, Intel VT-c



LINUX KVMの仕組み

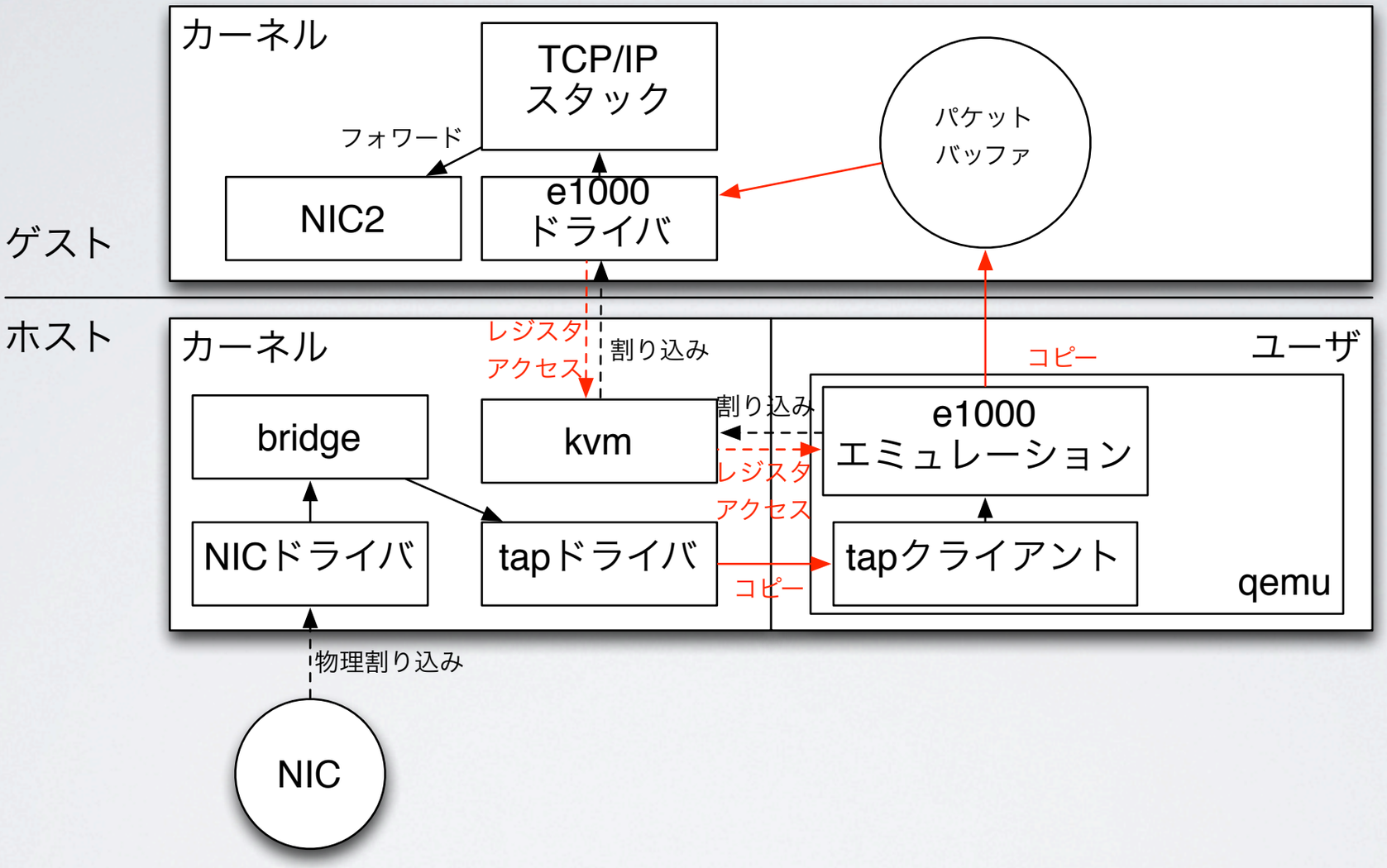


- Linuxカーネルに組み込まれたシンプルなハイパーバイザ（仮想化支援機構のあるCPUを前提）
- CPU仮想化以外の機能はKVM対応のQEMUが担当（仮想デバイス、メモリ確保、BIOS…）
- CPUエミュレーションを行う代わりにゲストへのモード切り替えを依頼するioctlをカーネルへ発行
- ゲストで行われた処理がトラップされるかハードウェア割り込みが起きるまでゲストが実行される
- ゲストからKVMで処理できない内容の操作（仮想デバイスへのアクセスなど）が行われたらQEMUへ制御を渡す

KVMの仮想NIC

- **NICエミュレーション（e1000, rtl8139...）**
  - QEMUを使って実在のNICをエミュレート　・遅い
- **virtio-net（ 準仮想化IO）**
  - IO仮想化フレームワーク「virtio」を用いてパケットの入出力を行う
  - エミュレーションで発生するオーバヘッドを削減
- **vhost-net（in-kernel virtio-net）**
  - カーネル内でIOを完結　・QEMUへモード切替を行うのに伴うオーバヘッドを削減、ゼロコピーの実現
- **ハードウェア支援によるIO仮想化（Intel VT-d & SR-IOV）**
  - ハードウェアがVMに対して仮想NICを提供
  - 最も速いが、ソフト実装よりは制約が多く融通が利かない／ハード対応が必要

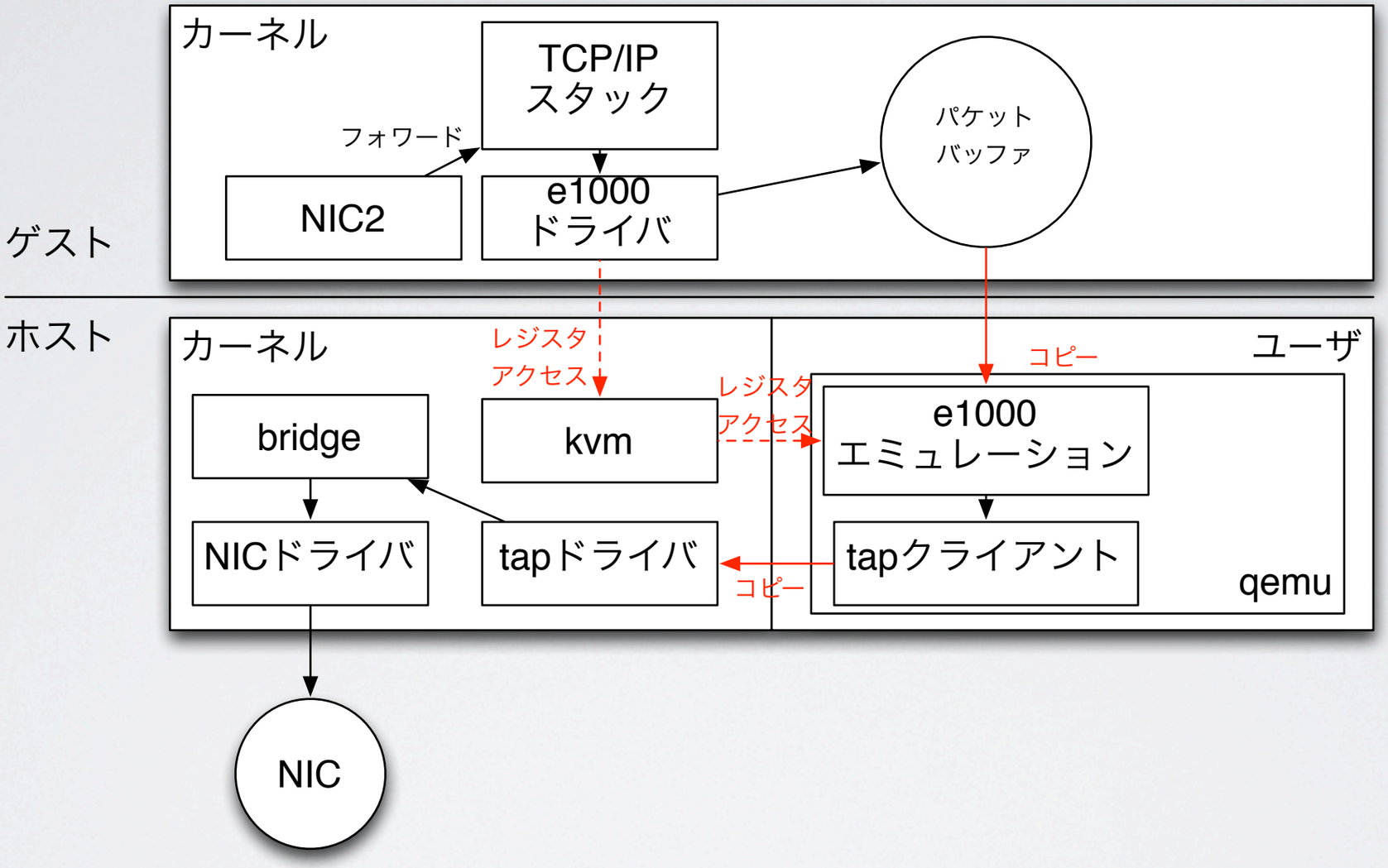
E1000 受信処理



- 仮想レジスタへのアクセスのたびにゲストモードの実行を中断し、カーネルからQEMUに切り替えてエミュレーションを行う

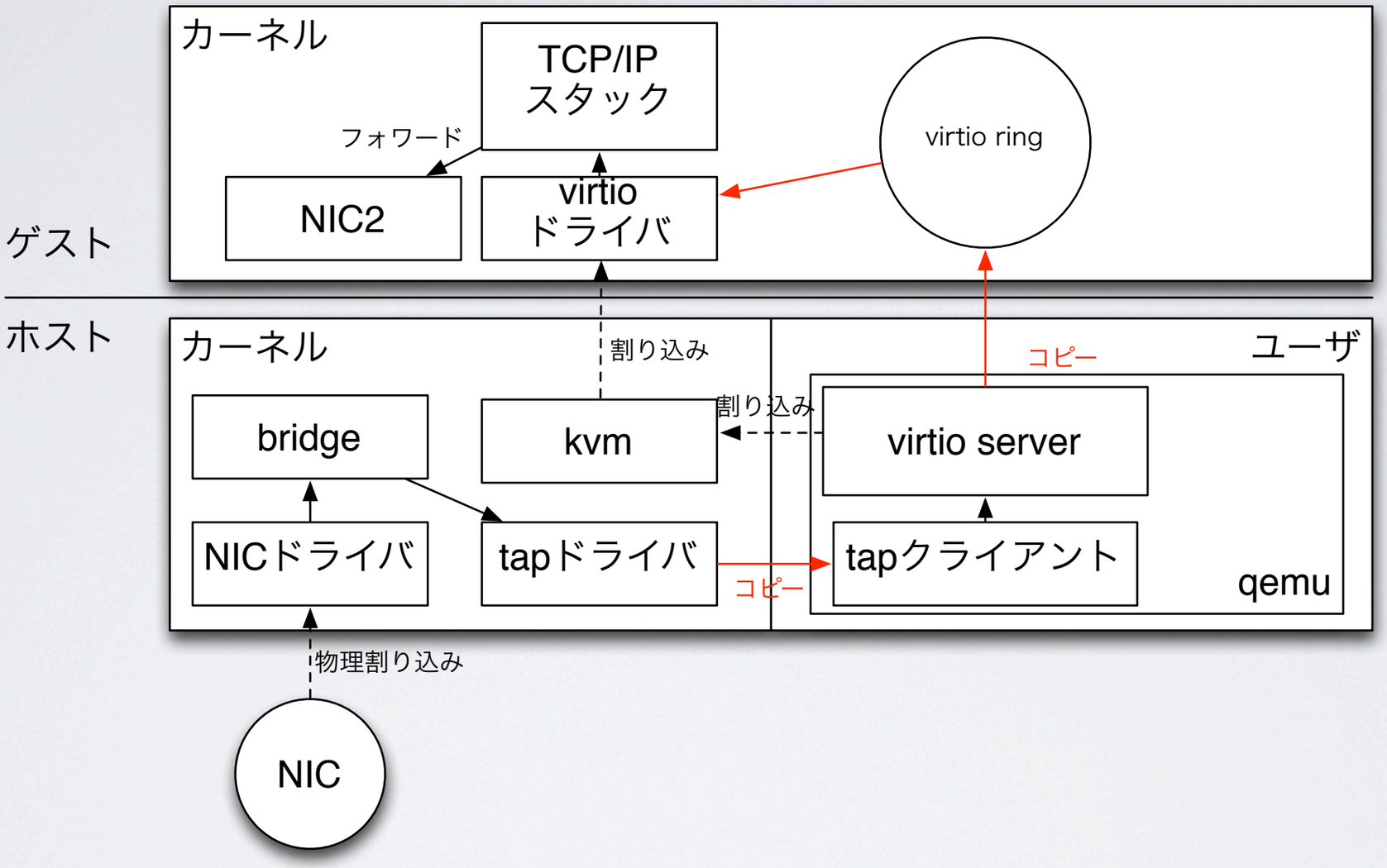
- TAPからQEMU、QEMUからバッファへのデータコピーが発生

E1000 送信処理



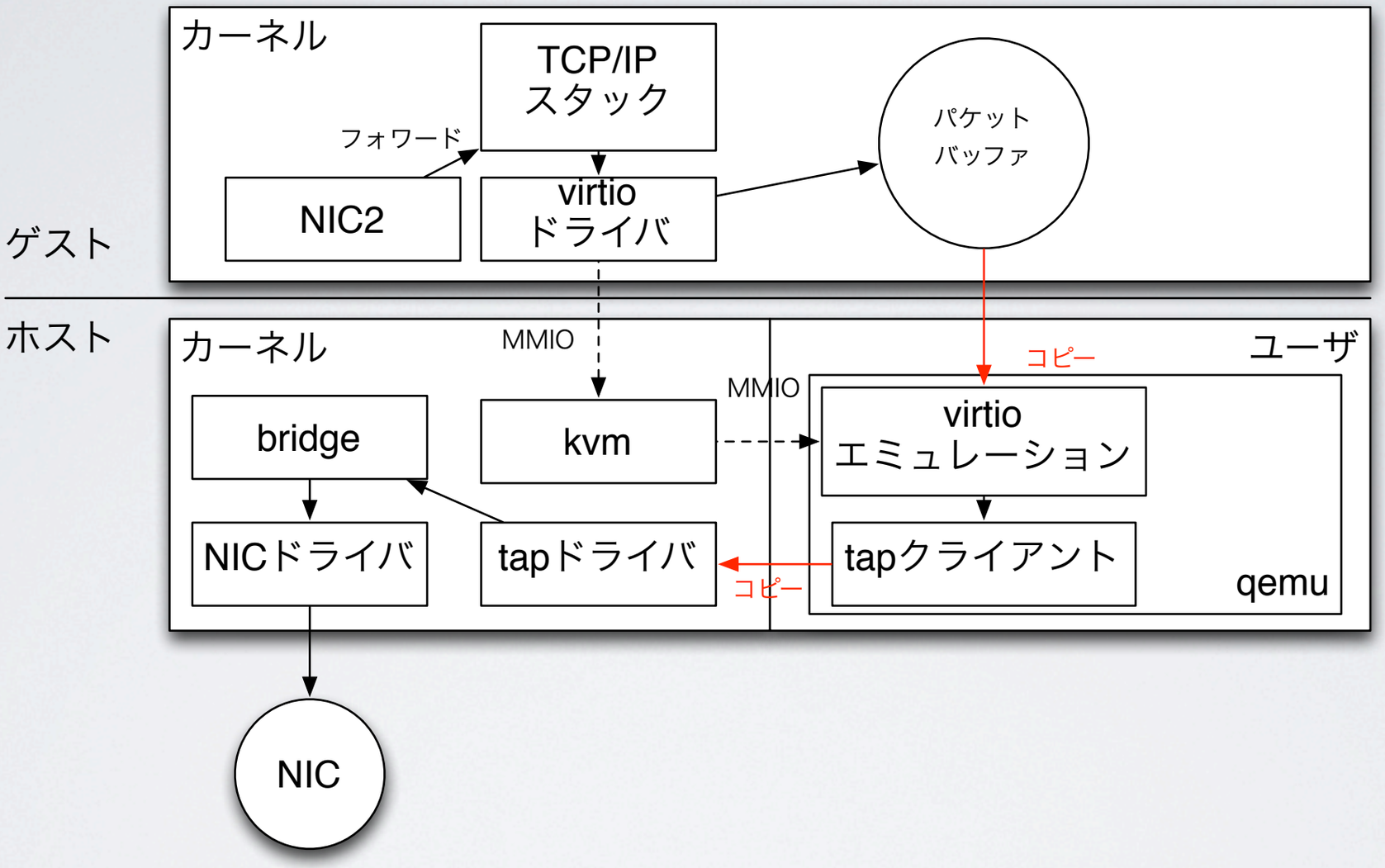
- オーバヘッドがある部分は受信処理とほぼ同じ

VIRTIO-NET 受信処理



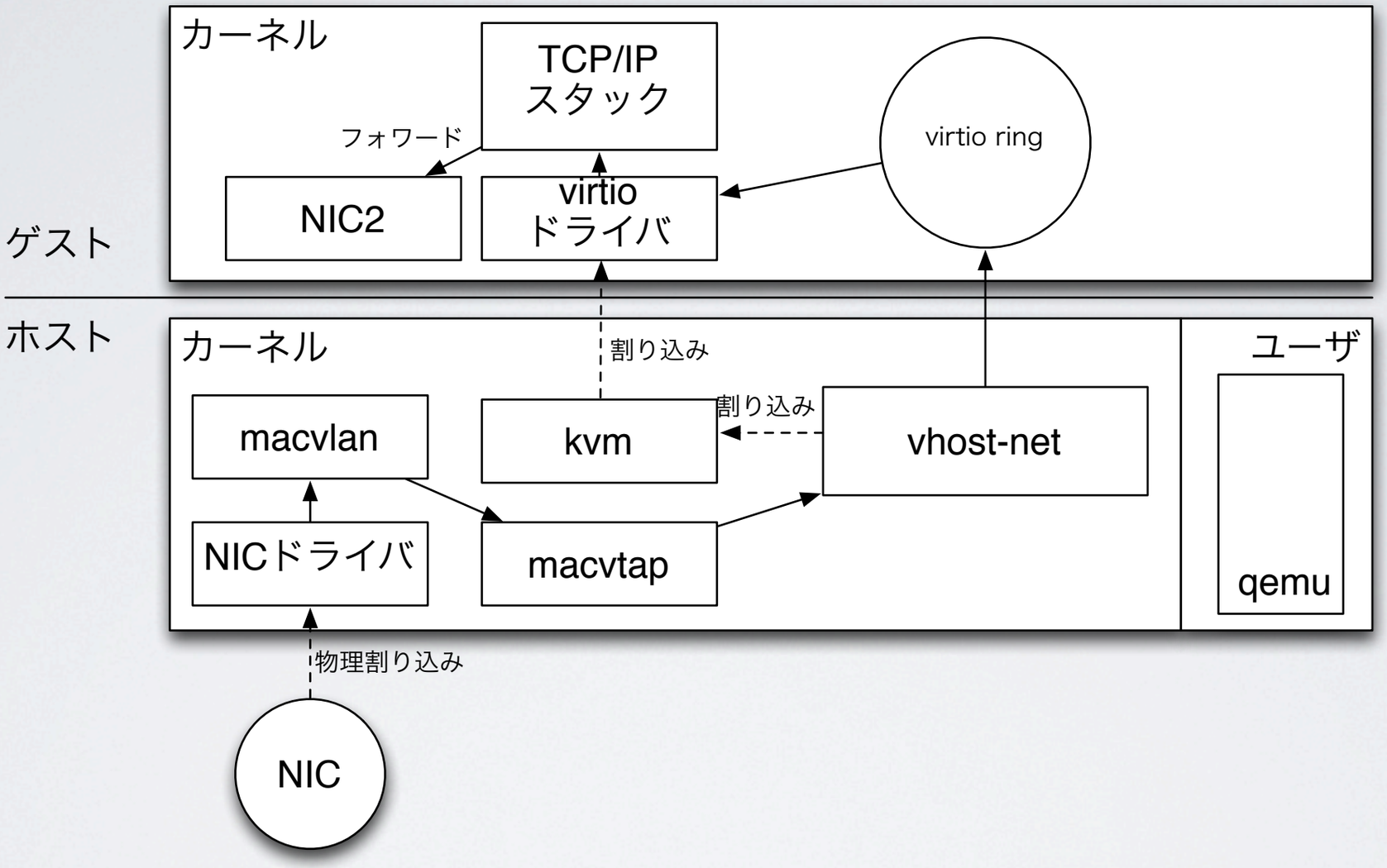
- 仮想レジスタは存在せず、virtio ring上の情報を見ながらパケットを取り出す為、ハイパーバイザへの切り替え回数が減る
- データコピーが発生するのは同様

VIRTIO-NET 送信処理



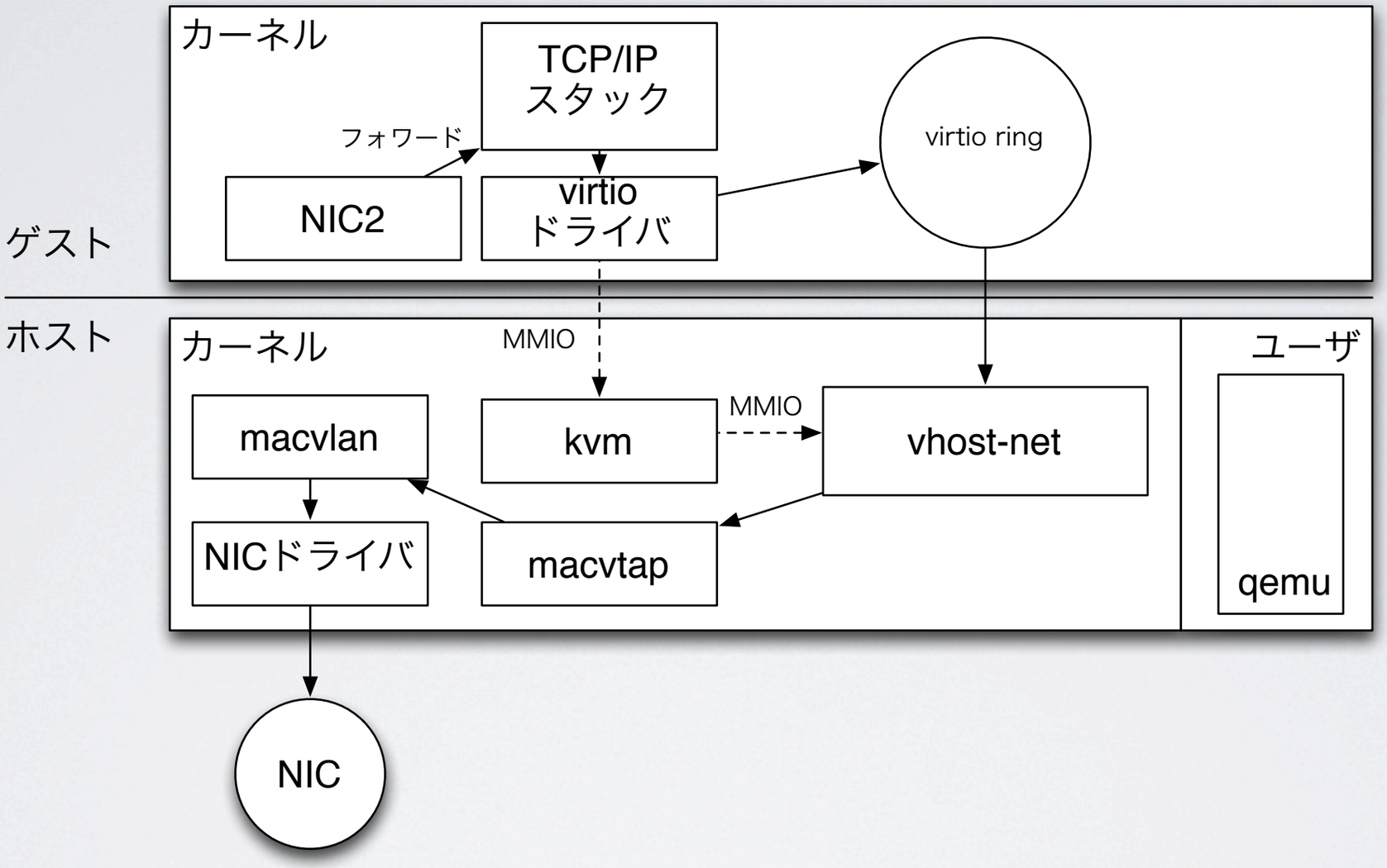
- 送信完了通知の為に 1 度だけMMIO（仮想レジスタアクセスと同じ）

VHOST-NET 受信処理



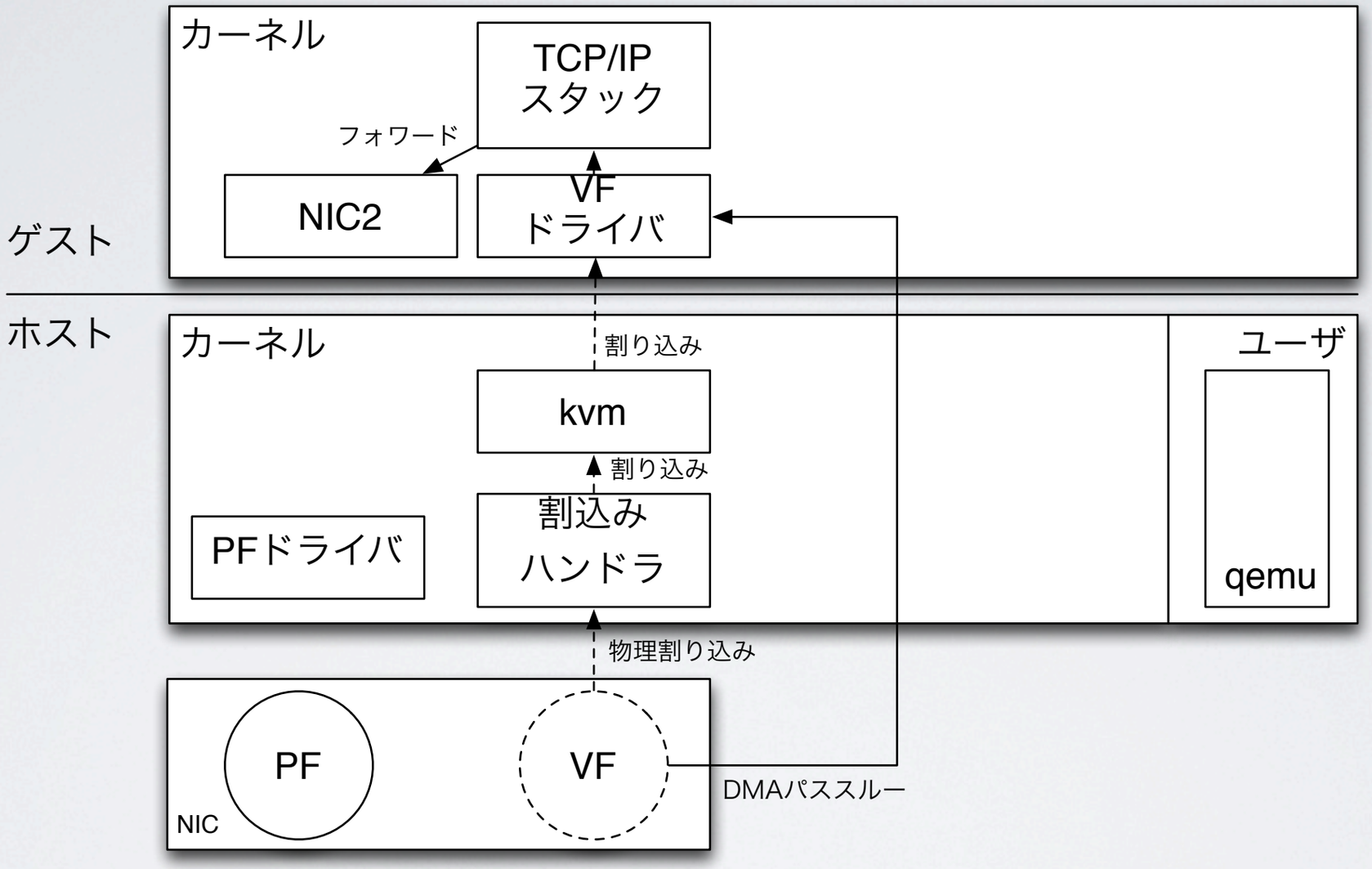
- virtio server機能をカーネルに実装した為、QEMUへの切り替えオーバヘッドが削減された
- bridge/tapの代わりにmacvlan/macvtap（後述）を用いればゼロコピーが実現

VHOST-NET 送信処理



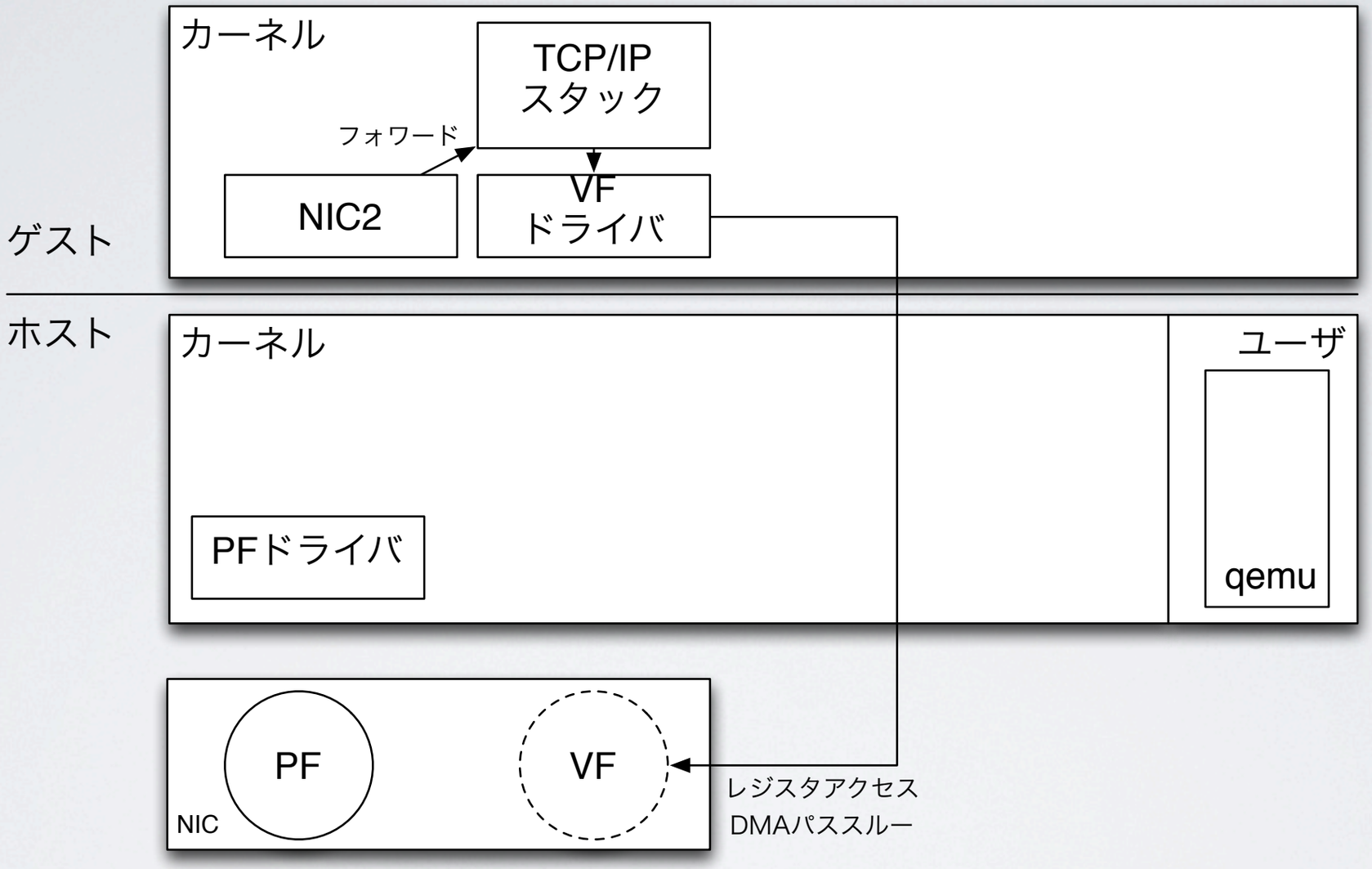
- オーバヘッドの削減箇所、動作原理は受信と同じ

INTEL VT-D & SR-IOV 受信処理



- VFはゲストへパススルーされ、直接IOが行える
- 割込みだけは仮想化出来ていないのでKVMを通じて転送される
- 最も性能が高いがハード対応が必要／ゲストのパケットをフィルター・改変する事は難しい

INTEL VT-D & SR-IOV 送信処理



- 送信完了割込みが発生する所まではKVMが介在せず、ゲストモードで動き続ける

スイッチ・TAPの実装

- bridge + tap
  スタンダードな実装

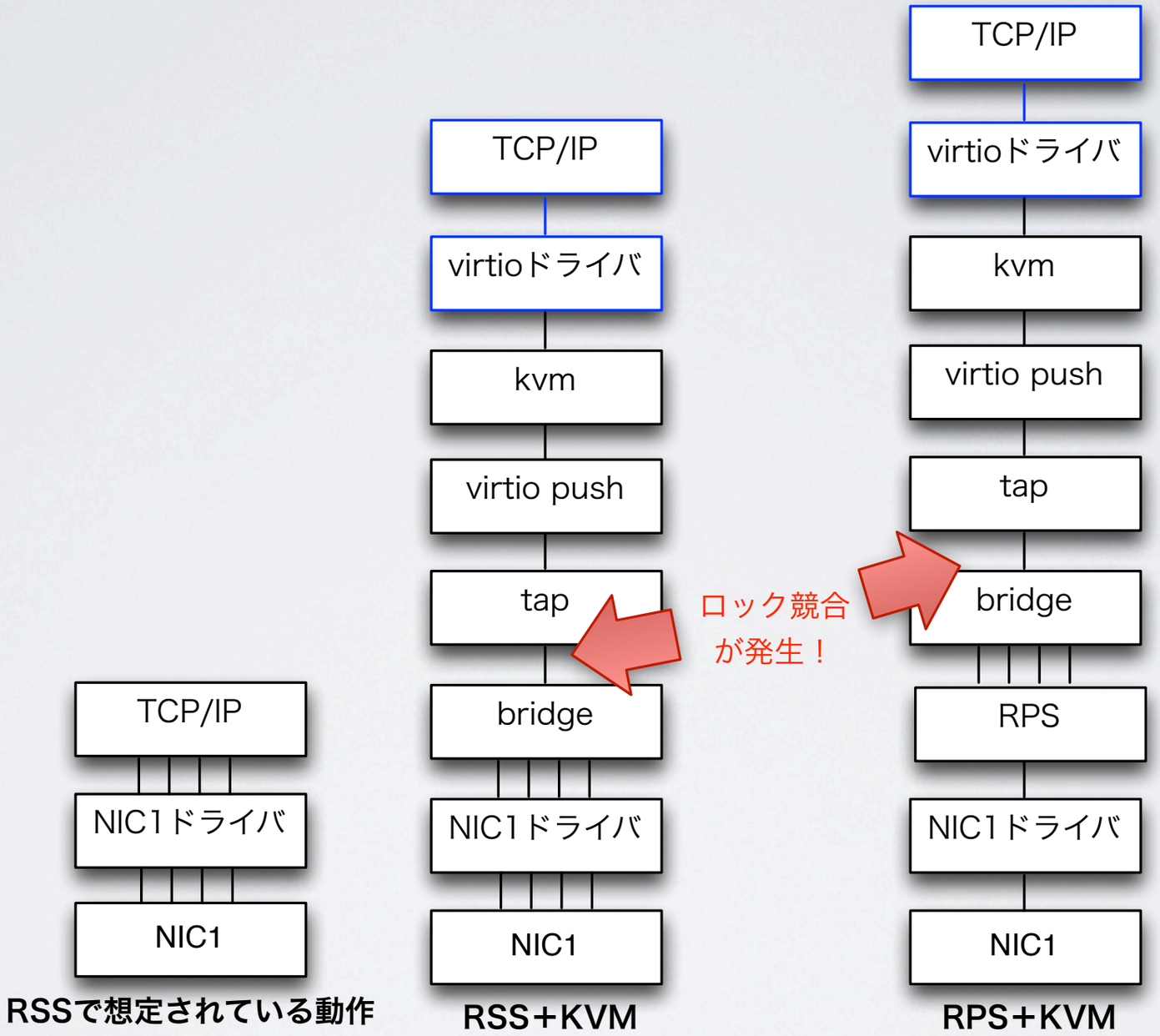
- macvlan + macvtap
  仮想マシンのNICに特化、bridgeを迂回する事で高速化している
  vhost-netと組み合わせる事でzerocopyを実現

- OpenvSwitch + tap
  ソフトウェアベースのOpenFlow Switchを実現
  柔軟なL2制御

マルチコアと
仮想化ネットワークIO

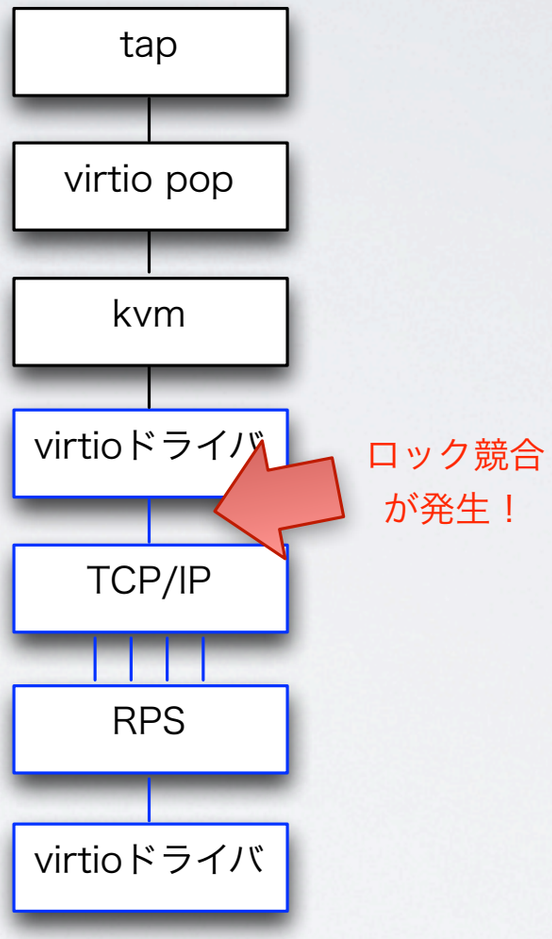
- 実は問題が多い
  - RPS/RSSで性能悪化
  - 割込み先CPUとvCPU実行中CPUの不一致による性能悪化
  - 割込み先CPUとvCPU実行中CPUのNUMAノード不一致による性能悪化
  - マルチvCPUの活用

RPS/RSSで性能悪化(1)



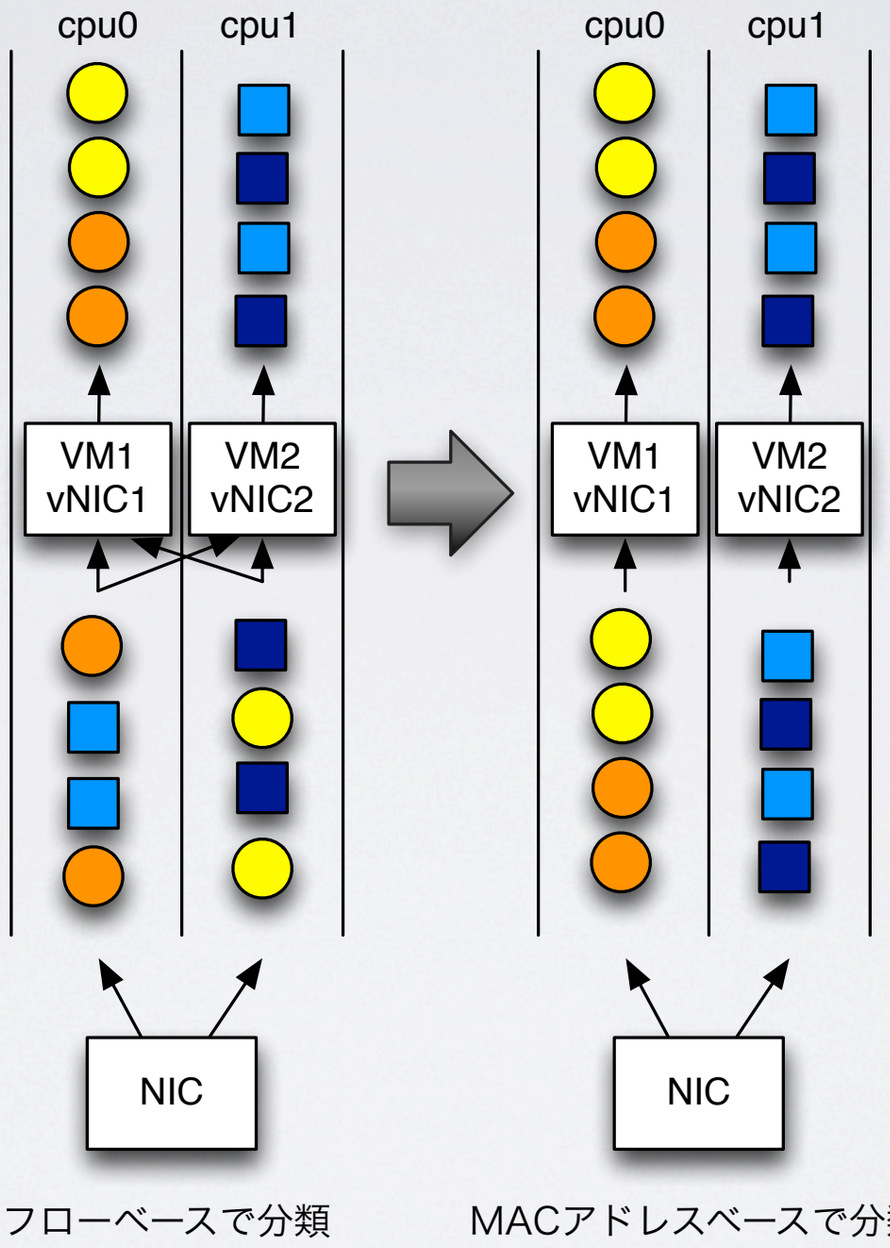
- RSSやRPSを用いて下位レイヤから並列にパケットを送っても、シングルキューなパスがあるのでロック競合が発生し、かえって性能が下がる

RPS/RSSで性能悪化(2)



- ゲストでRPSを用いる場合も同様に、 シングルキューなパスがあるのでロック競合が発生し、かえって性能が下がる

RPS/RSSで性能悪化(3)



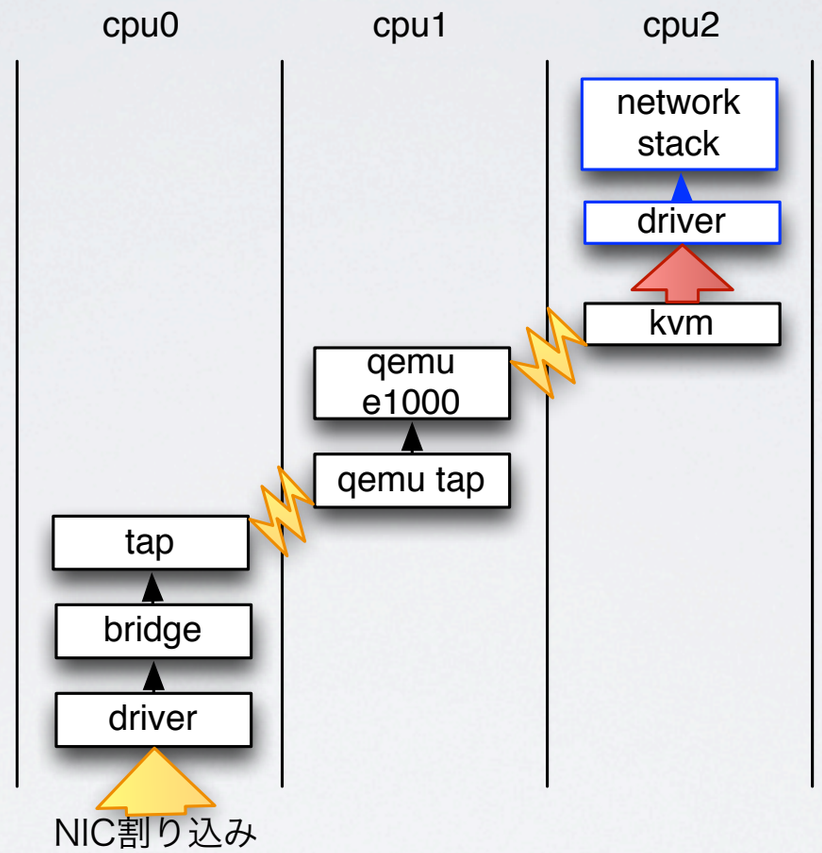
フローベースで分類　　MACアドレスベースで分類

- RSSはフローベースでコア分散を行うので割り込み先が入り乱れるような結果になり、性能が出ない
- MACアドレスベースで分類すると改善出来る→SR-IOV対応NIC

RPS/RSSで性能悪化の対策

- 仮想マシンのホストではRSS, RPSを無効化するべき
- VMが常に複数台走っているような環境では、RSS, RPSのようにフローベースでキュー先を決定するのではなくMACアドレスベースで決定すべき
  - SR-IOV対応NICはこれをハードウェアレベルで実現

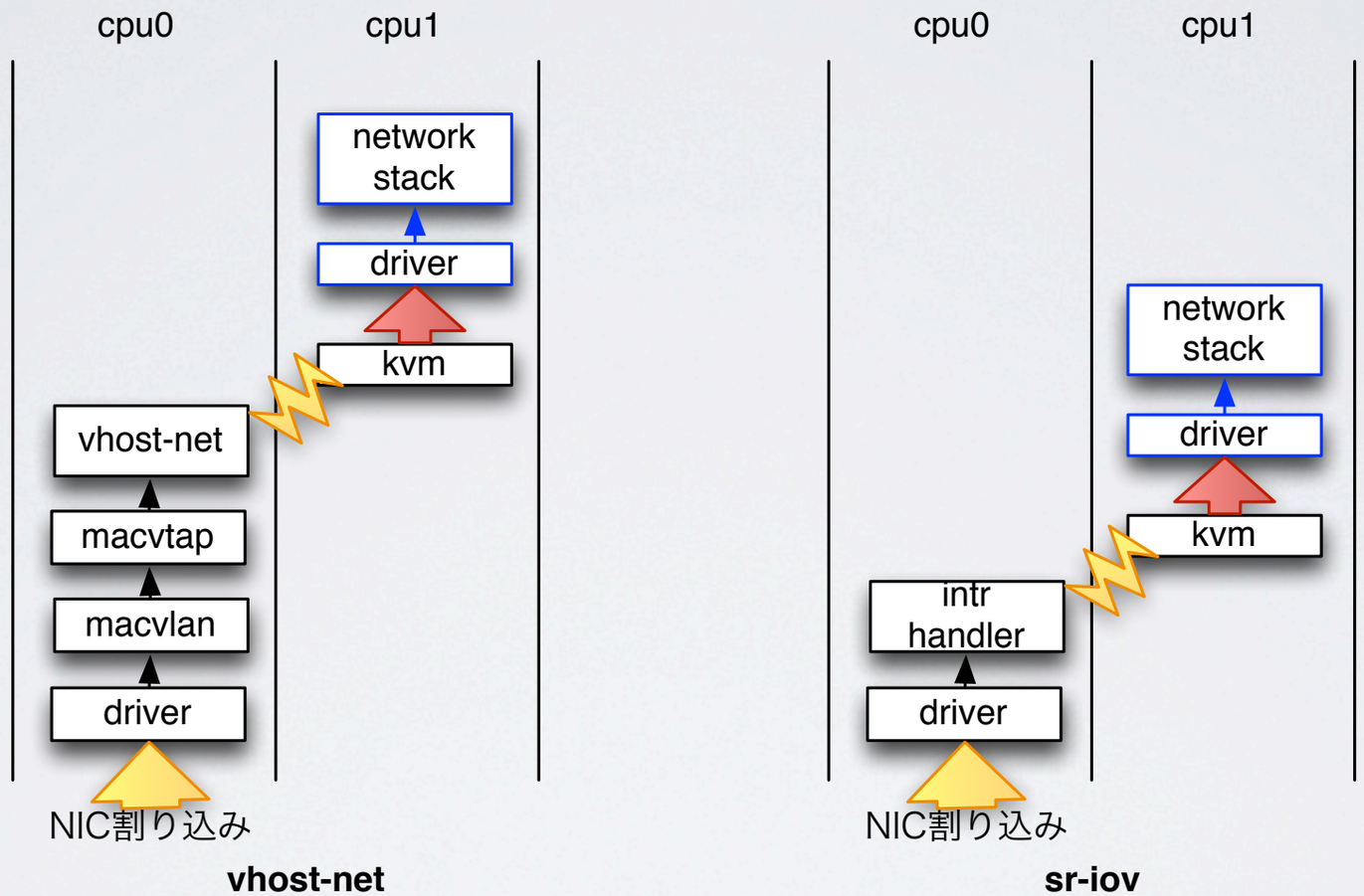
割込み先CPUとVCPU実行中CPUの不一致(1)



e1000 emulation

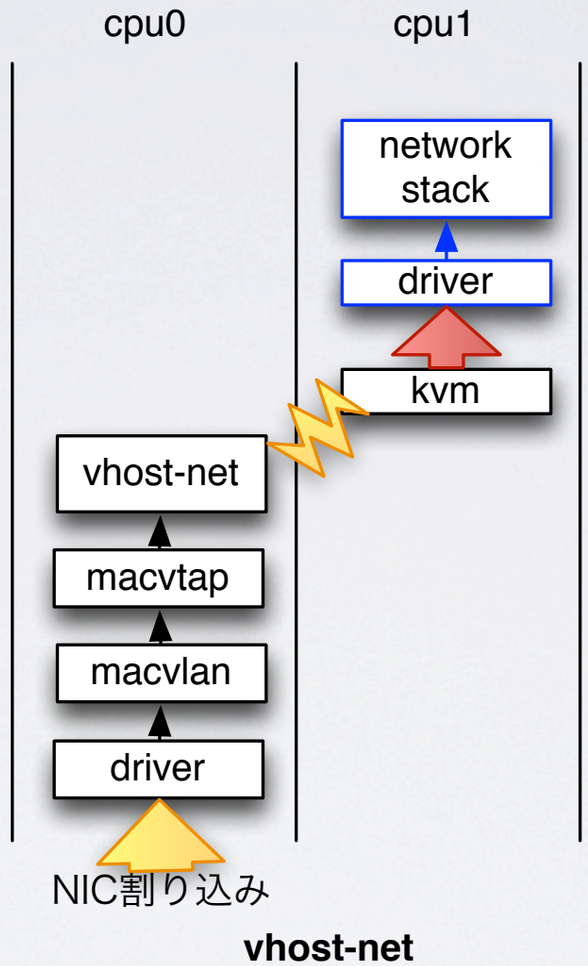
- e1000エミュレーションではQEMU IO threadとvCPU threadとNIC割込み先がバラバラのCPUになった場合、３つのCPUにまたがって処理が行われる
- RSS／RPSが有効になっていると更に増え、最悪４つのCPUにまたがって処理が行われる

割込み先CPUとVCPU実行中CPUの不一致(2)



- vhost-net, SR-IOVではNIC割込み先がvCPU threadと異なる場合、２つのCPUにまたがって処理が行われる

割込み先CPUとVCPU実行中CPUのNUMAノード不一致による性能悪化



- cpu0がnode0、cpu1がnode1に属しドライバが初期化時にnode0からバッファを確保したと仮定すると、cpu1(node1)上のvCPUでパケット受信処理を行うとアクセスコストが高くなってしまう

CPU不一致による性能悪化の対策

- 割込みをpinning

  echo 1 > /proc/irq/<num>/smp_affinity

- 割込み先と同じCPUか同じnodeへvCPUをpinning

  <vcpu cpuset='1'>1</vcpu>

マルチVCPUの活用

- virtio-netでのマルチキュー／RSS対応
  未だ開発段階で十分な性能を引き出すに至っていない
  http://www.linux-kvm.org/page/Multiqueue

- SR-IOVでのマルチキュー／RSS対応
  Intel 82599ではSR-IOVのVF上で最大4キューのRSSに対応しているが、最新のLinuxのドライバでもこの機能は実装されておらず、シングルキューのNICとしてしか動作しない

- RPS
  仮想マシン上でも動作するが、パケットフォワーディング用途だと送信側NICがシングルキューなのがボトルネックになる

まとめ

- 仮想NICはSR-IOVが使える場合はSR-IOVを、
  それ以外ではvhost-netを使うのが最も性能が良い

- マルチコア環境におけるNIC→vCPUへの割込みルーティングは自動的に最適化されない為、手動でチューニングを行う必要がある

- マルチvCPUを活用したパケットフォワーディングは、開発段階であり今すぐ使える状態ではない