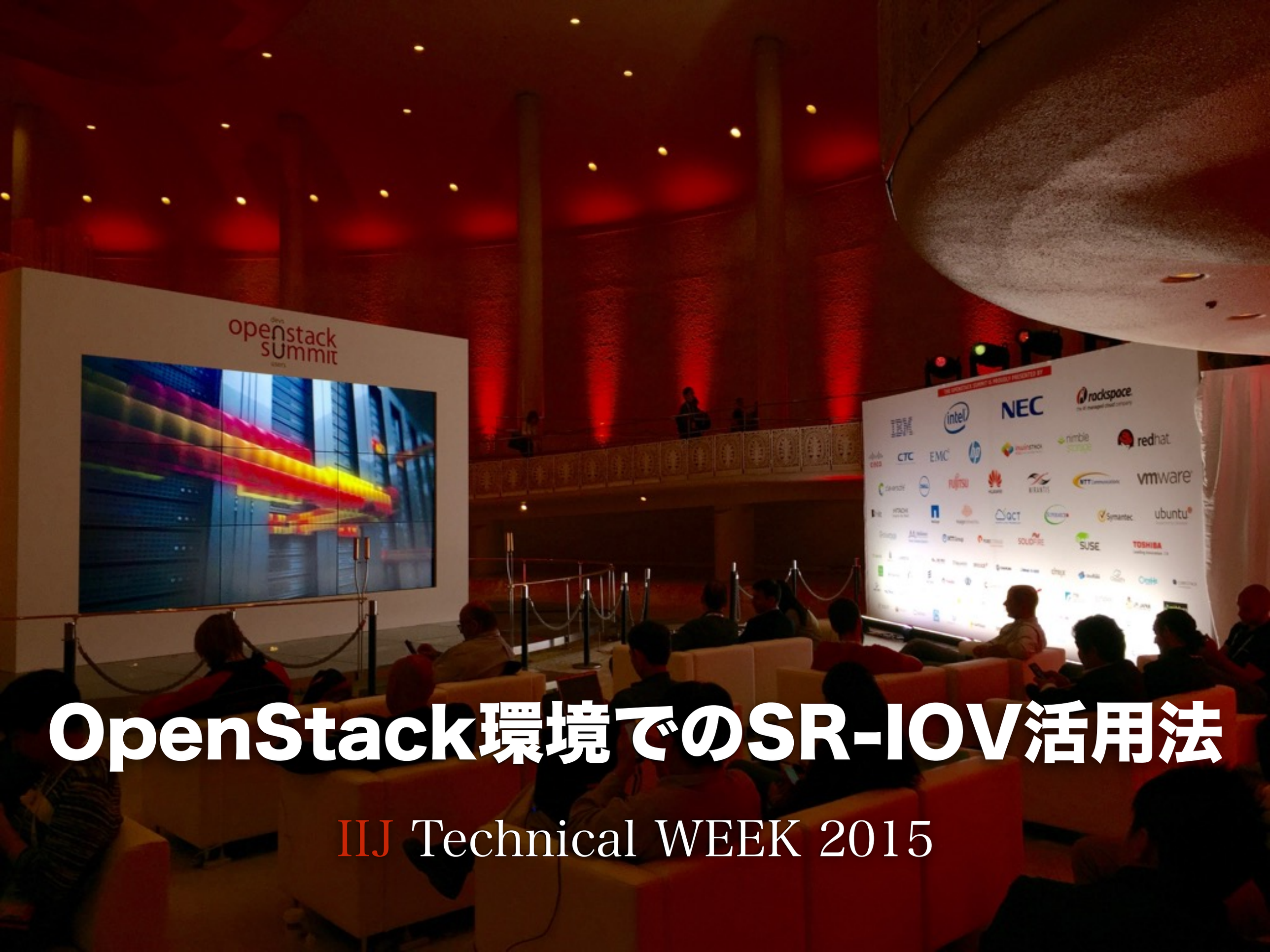


OpenStack環境でのSR-IOV活用法

IIJ Technical WEEK 2015

whoami

齊藤 秀喜(さいとう ひでき) / TwitterID: @saito_hideki

株式会社インターネットイニシアティブ
- http://www.iij.ad.jp/
- プラットフォーム本部
  システム基盤技術部
  シニアエンジニア

日本OpenStackユーザ会
- http://openstack.jp/

トピック

★仮想マシンの高性能化を目指して

- ➡ ネットワーク性能を向上させる手法 - SR-IOV
- ➡ OpenStackとSR-IOV
- ➡ SR-IOVの効果
- ➡ 現状の課題と将来への期待

本セッションでは、OpenStack管理下の仮想マシンの
ネットワーク性能を向上させる手段として
SR-IOVを活用する手法について紹介します

高性能な仮想マシン？

本セッションでは、３つの性能指標のうちネットワーク性能にフォーカスしてお話します

(1) CPU性能/メモリサイズ

> vCPU数とメモリサイズ

(2) ネットワーク性能

> 仮想サーバ単位の帯域上限

(3) ディスク性能

> 最大 IOPS/インスタンス

基本スペック ※1								備考
グレード（品目）	V10	V20	V40	V80	V160	V240w2	V240/V8w3	
料金（日額）※4	133円	266円	466円	733円	1,133円	1,633円	4,266円	
料金（月額）※5	4,000円	8,000円	14,000円	22,000円	34,000円	49,000円	128,000円	
最低利用期間	なし							
CPU性能指標（ICU）※6	1	2	4	8	16	24	24	
メモリ	1GB	2GB	4GB	8GB	16GB	24GB	80GB	
ディスク容量	30G							IP-SAN、RAID6構成
ディスク追加（有償オプション）	100GB、300GB、500GB単位、2領域まで追加可能							
NIC	標準2（グローバル、プライベート） VLANオプション契約で最大3							
インターネットコネクティビティ	付属（共用コネクティビティ）							
仮想サーバ単位の帯域上限（Mbps）	30	50	80	125	200	300	1,000	帯域を保証するものではありません
IPアドレス	グローバル（eth0）はオプションでIPアドレスを追加可能（標準1、追加3で合計4個まで） プライベート（eth1）は標準で設定されるIPv4アドレスのみ利用可能 VLANオプション（eth2）はユーザが自由にアドレスを設定可能（10.0.0.0/8を除く）							
運用監視	Ping監視、Port監視、URL監視を設定可能（初期状態は設定なし）							通知メールアドレスは4つまで

［参考］仮想サーバのスペック http://www.iij.ad.jp/biz/hosting/spec.html

仮想化基盤のネットワーク

仮想化基盤では、VMM上で仮想スイッチを稼働させ、物理サーバのNICと仮想マシンのNICを中継する実装が一般的

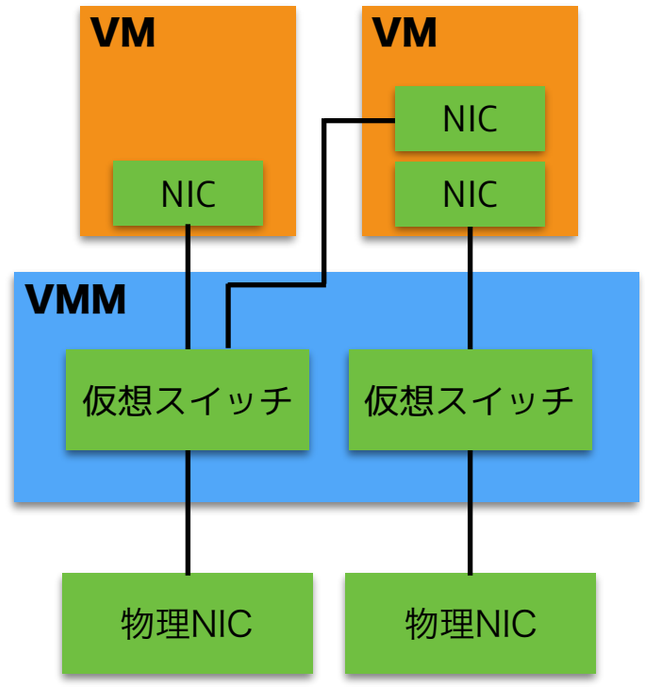
➡ メリット

> 物理NICの数に影響されることなく仮想マシンの収容効率を上げられる

> 仮想スイッチはソフトウェアであるため機能の追加や管理が容易である

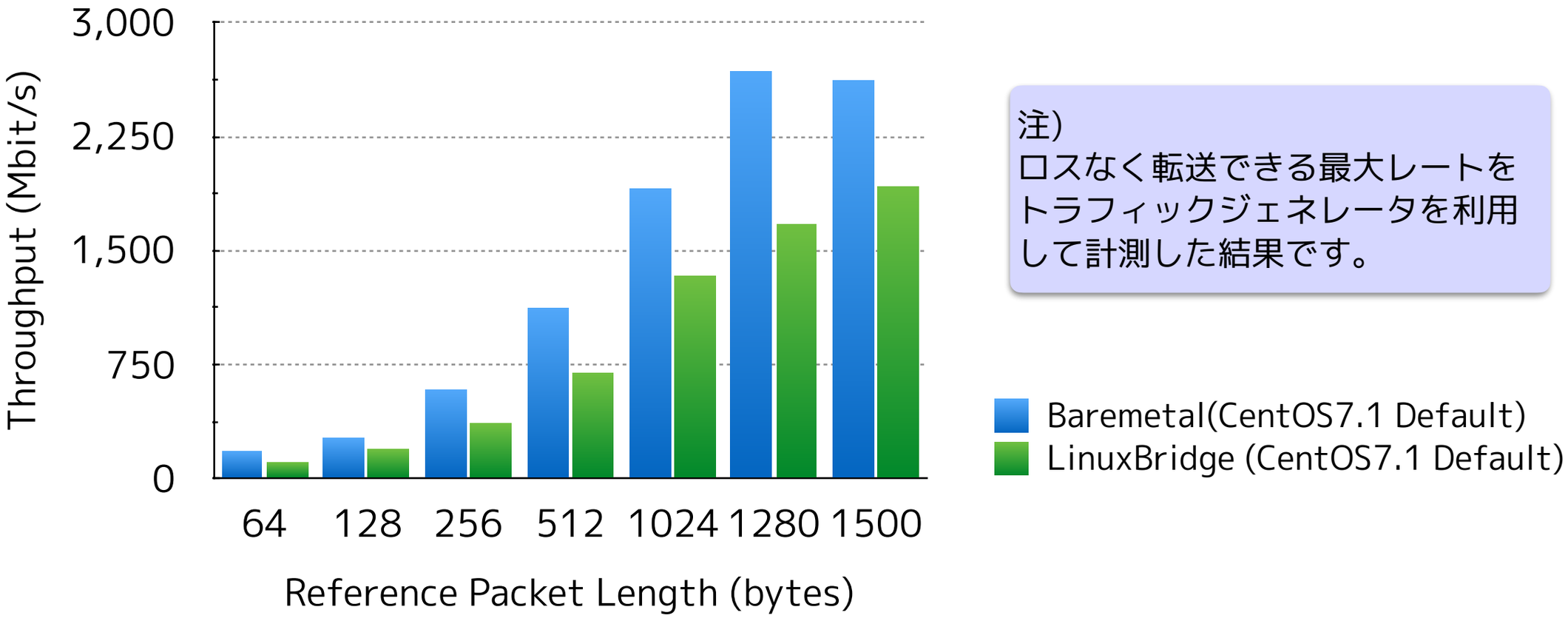
➡ デメリット

> 仮想マシンに中継する通信の量が増加すると、仮想スイッチでの中継処理やバッファコピーにVMMが忙殺されシステムのボトルネックとなる



参考:RFC2544 Throughput

チューニングをしていないデフォルト状態のCentOS 7.1の上でLinuxBridgeを動作させ、パケットのフォワーディング性能を計測した結果は以下の通り



注)
ロスなく転送できる最大レートをトラフィックジェネレータを利用して計測した結果です。

仮想化基盤のネットワーク性能向上

一般的に実現可能な技術を利用して、仮想化基盤でのネットワーク性能を向上させるためのアプローチは大きく2種類

(1) 仮想スイッチをバイパスする技術

> PCIパススルー

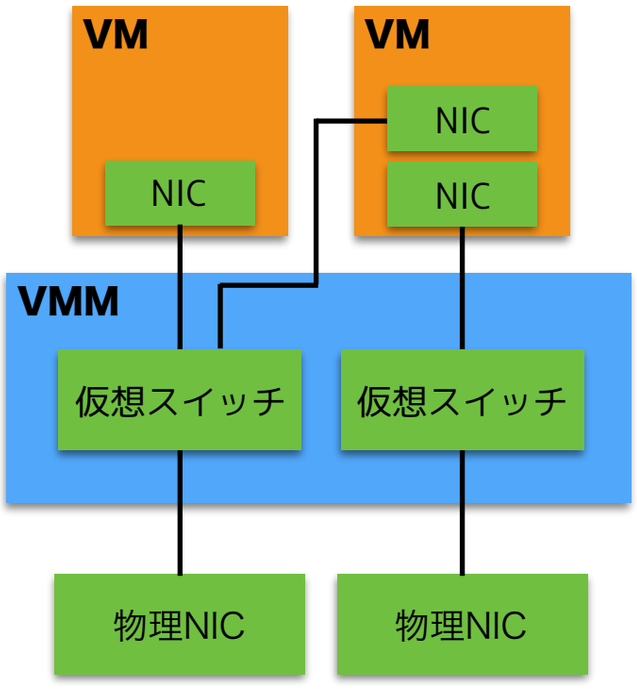
(2) 仮想スイッチを徹底的に速くする技術

> 仮想スイッチ + Intel DPDK

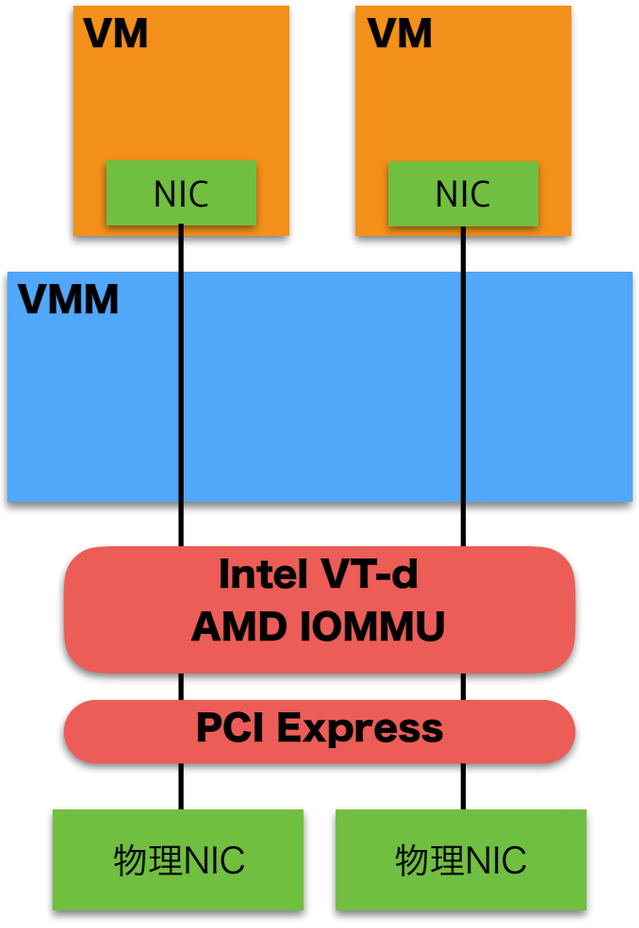
以降では、(1)について少し掘り下げてお話します

仮想スイッチをバイパスする技術

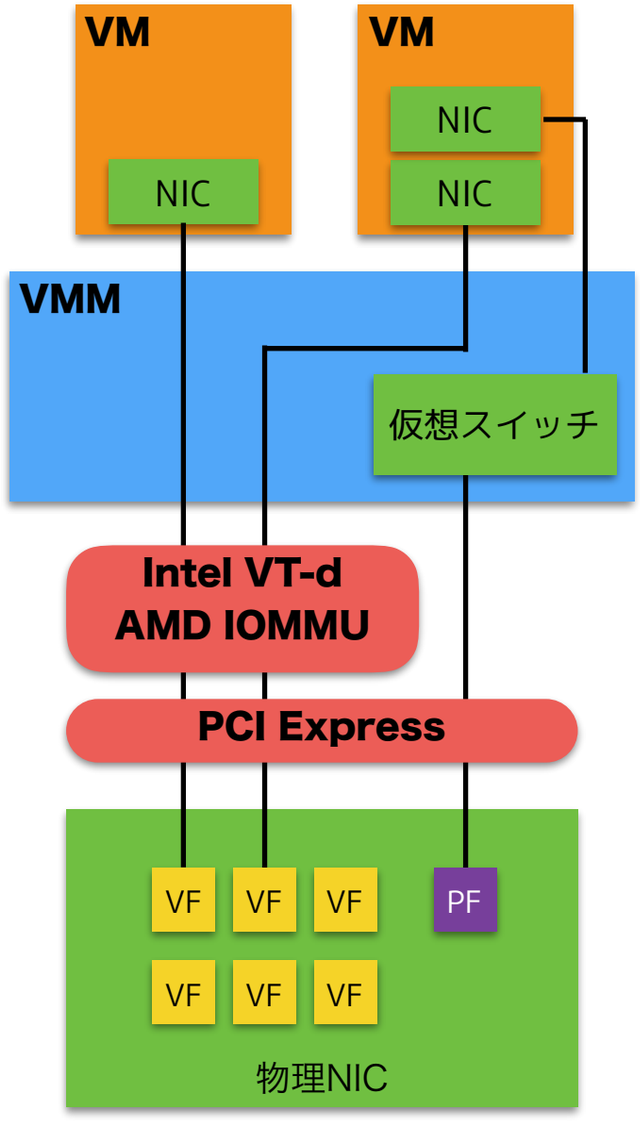
仮想スイッチモデル



PCIパススルーモデル



SR-IOVモデル



Single Root I/O Virtualization

- Single Root I/O Virtualization (SR-IOV)の略
- PCI Special Interest Group(PCI-SIG)が仕様を策定
- PCIデバイスを多重化して仮想マシンにパススルー接続する技術の業界標準

Intel VT-X/VT-d または AMD IOMMU 拡張をサポートするホストハードウェア

SR-IOV機能を持つPCIデバイス

SR-IOVモデル: 特徴

SR-IOVモデルの特徴をメリット・デメリットで分類すると…

○ 仮想スイッチをバイパスするため、VMMに負荷をかけることなく仮想マシンが外部と通信可能

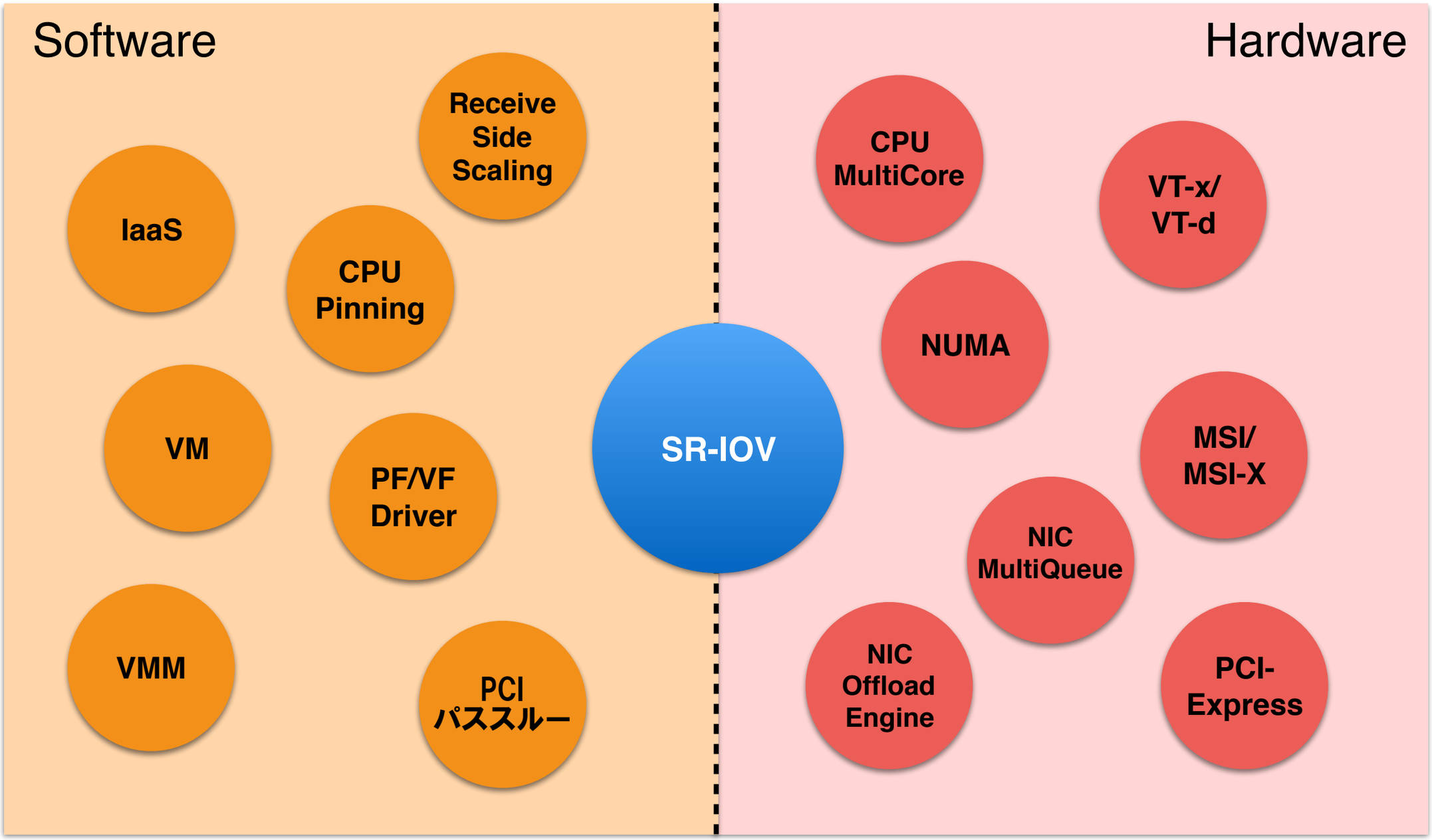
○ 従来のPCIパススルーでは、物理NIC:仮想マシンが1:1の関係だったが、SR-IOVでは1:N (ixgbeでは最大63)。

× PCIバスと仮想マシンの関係を定義・管理する仕組みが必要となる。(人力では無理)

× LiveMigrationのように箱を移動させる際には注意が必要

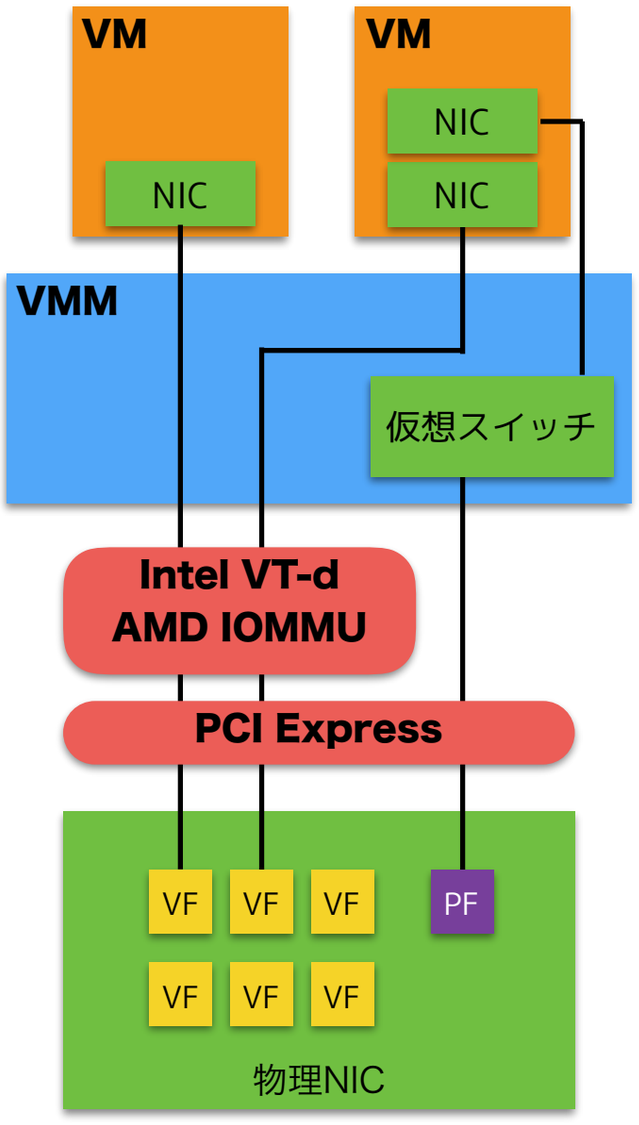
△ 仮想化基盤が提供するセキュリティグループのような、仮想マシンを守るための技術を一部利用できなくなる

SR-IOVモデル: 関連する技術



SR-IOVモデル: キーワード

SR-IOV機能を持つPCIデバイスはPFとVFという2つのファンクションを持っています



➡物理ファンクション(PF)

母艦となるVMMからは、通常のPCIデバイスとして見えている。
デバイスドライバロード時にPFに対してVFを割り当てることによりNICのSR-IOV機能を有効化する。

➡仮想ファンクション(VF)

VFは、PCIデバイスと仮想マシン間のデータ転送のみを処理する。
SR-IOV機能をもったPCIデバイスは、1つのPFにつき複数のVFを切り出すことが可能。

解説:
SR-IOVを利用する

SR-IOVを利用するまでの流れ

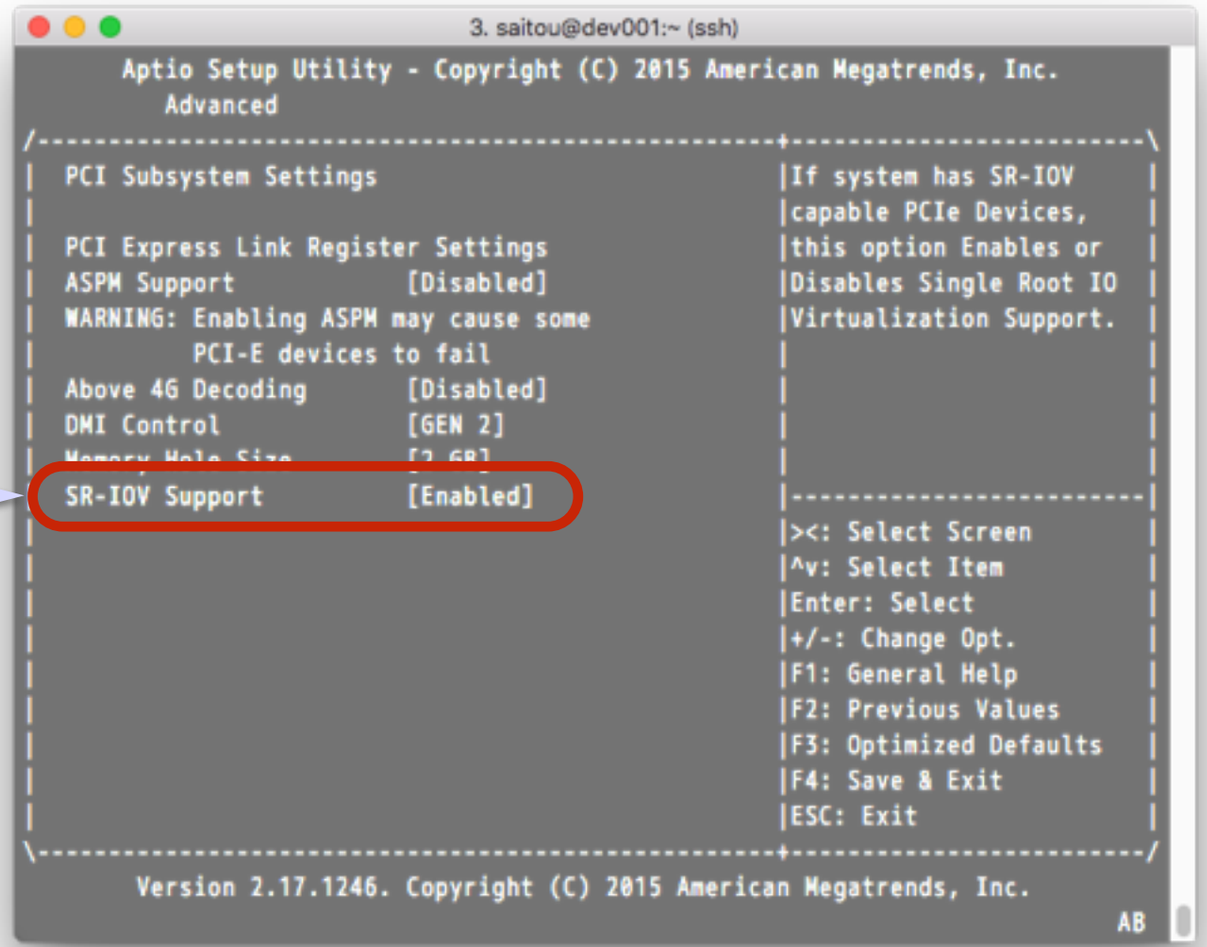
CentOS 7.1をインストールした環境で、Intel X540カードのSR-IOVを有効化して利用する流れは以下の通り

1. BIOSでSR-IOVを有効化
2. 起動時のカーネルパラメータでIOMMUを有効化
3. VFの分割数を指定してixgbeドライバをロード
4. VFに割り当てられるPCIバスを仮想マシンにパススルー接続する

BIOSでSR-IOVを有効化

```
# modprobe ixgbe max_vfs=8
[   54.227092] ixgbe 0000:03:00.0: not enough MMIO resources for SR-IOV
[   54.234257] ixgbe 0000:03:00.0 (unregistered net_device): Failed to enable PCI sriov: -12
[   55.419649] ixgbe 0000:03:00.1: not enough MMIO resources for SR-IOV
[   55.426811] ixgbe 0000:03:00.1 (unregistered net_device): Failed to enable PCI sriov: -12
```

BIOSレベルでのSR-IOV
有効化を忘れずに



IOMMUの有効化とモジュールのロード

- GRUB2設定変更(/etc/sysconfig/grub.conf)
  - IOMMU有効化

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=centos/root rd.lvm.lv=centos/swap console=ttyS1,115200n8
crashkernel=auto rhgb quiet intel_iommu=on"
```

  - ixgbeモジュールのロード設定(VFを15分割/ポート)

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=centos/root rd.lvm.lv=centos/swap console=ttyS1,115200n8
crashkernel=auto rhgb quiet intel_iommu=on ixgbe.max_vfs=15,15”
```

- grub2設定の反映
  - grub2-mkconfigコマンドで設定反映

```
# grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg
```

NICドライバのロード時にVFを有効化

➡分割されたVFはipコマンドで確認可能

```
# ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
[…]
648: eno49: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT qlen 1000
    link/ether 8c:dc:d4:b5:c9:0c brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 1 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 2 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 3 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 4 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 5 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 6 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 7 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 8 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 9 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 10 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 11 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 12 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 13 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
    vf 14 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
```

VFのPCIバスIDでパススルー接続

➡ libvirtのdomain定義ファイルでVFのPCIバスを指定

```
<domain type='kvm' id='2'>
  <name>instance-00000157</name>
  <uuid>c9d9e00d-3608-414e-a71f-1574a262c29a</uuid>
  ...
    <interface type='hostdev' managed='yes'>
      <mac address='fa:16:3e:5a:84:c7'/>
      <driver name='vfio'/>
      <source>
        <address type='pci' domain='0x0000' bus='0x04' slot='0x10' function='0x0'/>
      </source>
      <vlan>
        <tag id='2401'/>
      </vlan>
      <alias name='hostdev0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </interface>
  ...
```

DEMO:
SR-IOV有効化

SR-IOV使用上の注意

- SR-IOVは、VMMが提供する仮想スイッチをバイパスして物理NICが提供するVFと仮想マシンを直接続する
- SR-IOVを利用するには、サーバ/NIC/VMMが対応していなければならない
- 管理者は、NICの物理ポート毎に複数のVFと対応するPCIバスID、MACアドレスを常に管理する必要がある

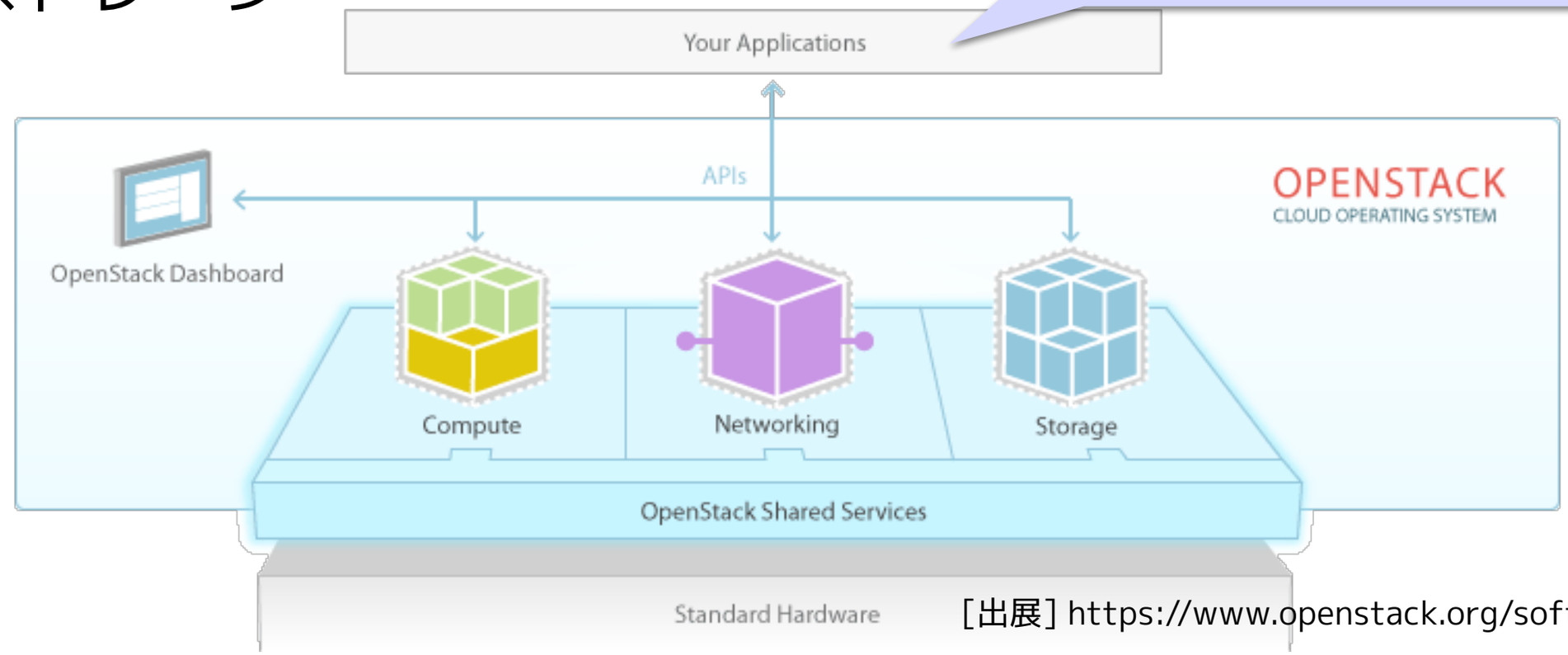
活用:
OpenStackとSR-IOV

OpenStackについて

ITに必要な資源管理機能をソフトウェア制御し、必要なときに必要なだけ利用者に提供するCloudOSです。

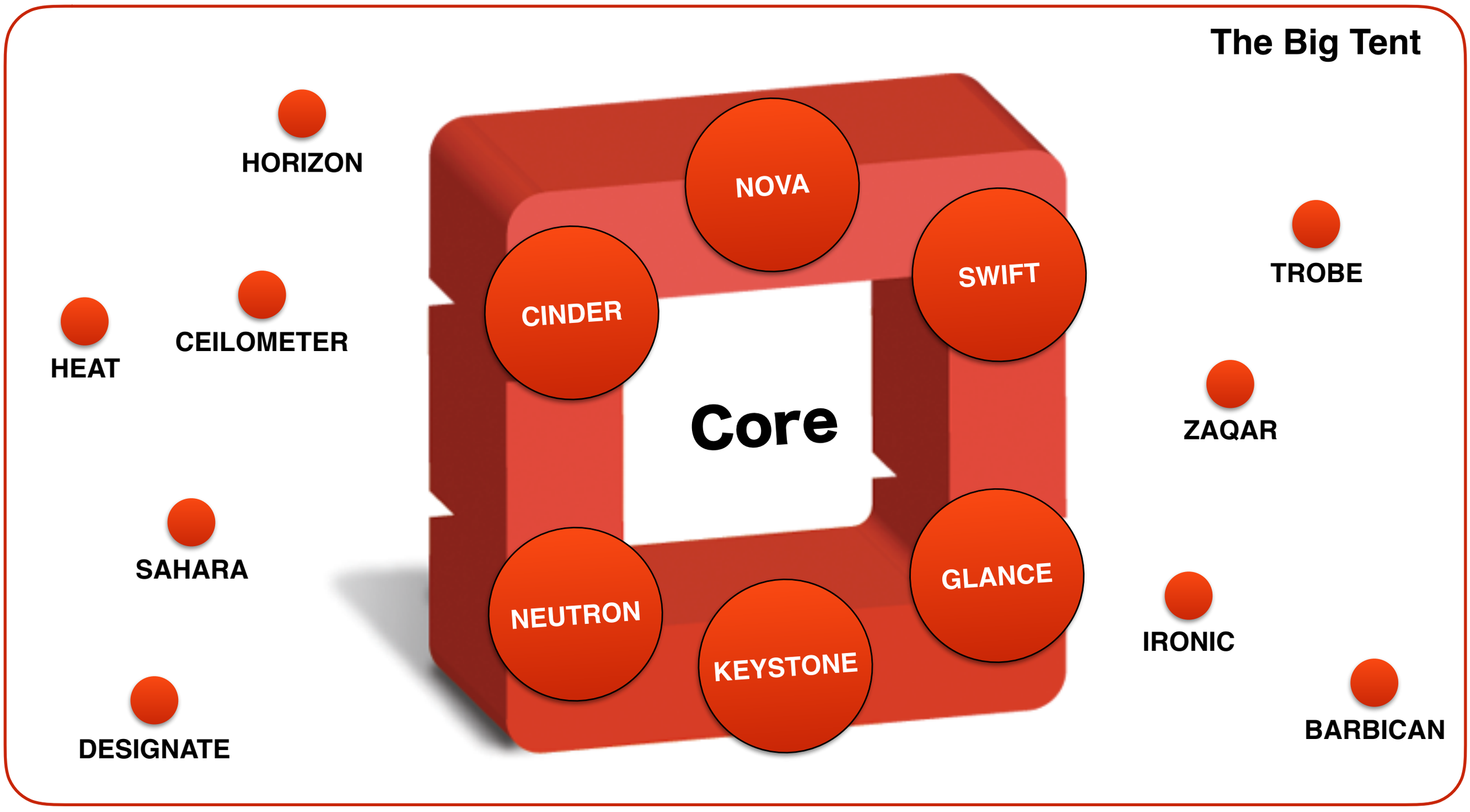
> 計算機
> ネットワーク
> ストレージ

あなたが思い描くインフラを自在に構成することができる仕組みを提供(コンセプト)



[出展] https://www.openstack.org/software/

OpenStackを構成するプロジェクト群



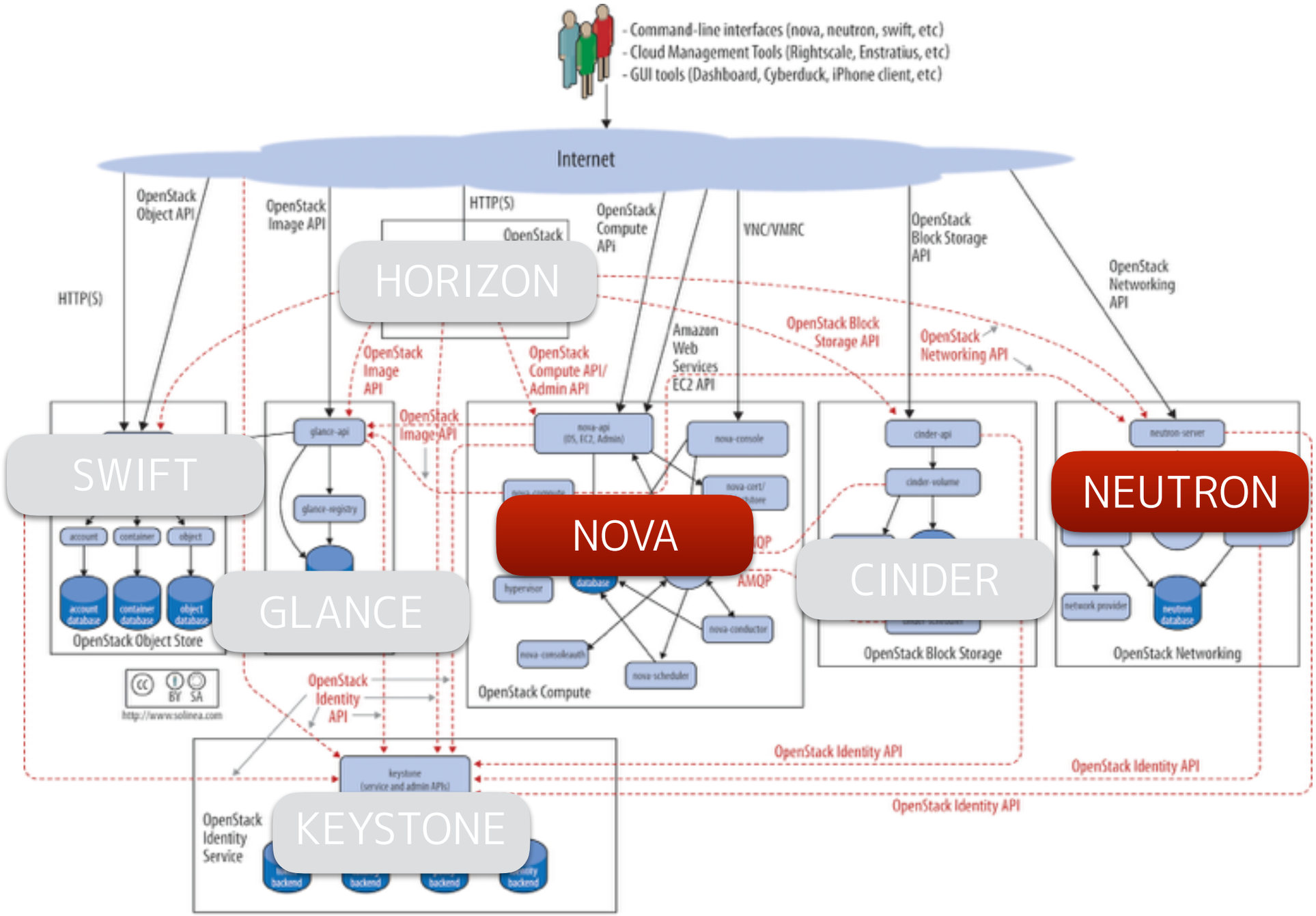
コアプロジェクト

- **NOVA**: 仮想マシンインスタンスの管理サービス
- **SWIFT**: オブジェクトストレージサービス
- **GLANCE**: 仮想マシン用OSイメージ管理サービス
- **KEYSTONE**: ユーザ認証/権限の管理サービス
- **NEUTRON**: IaaS基盤のネットワーク管理サービス
- **CINDER**: 仮想マシン用ブロックストレージサービス

オプションサービス

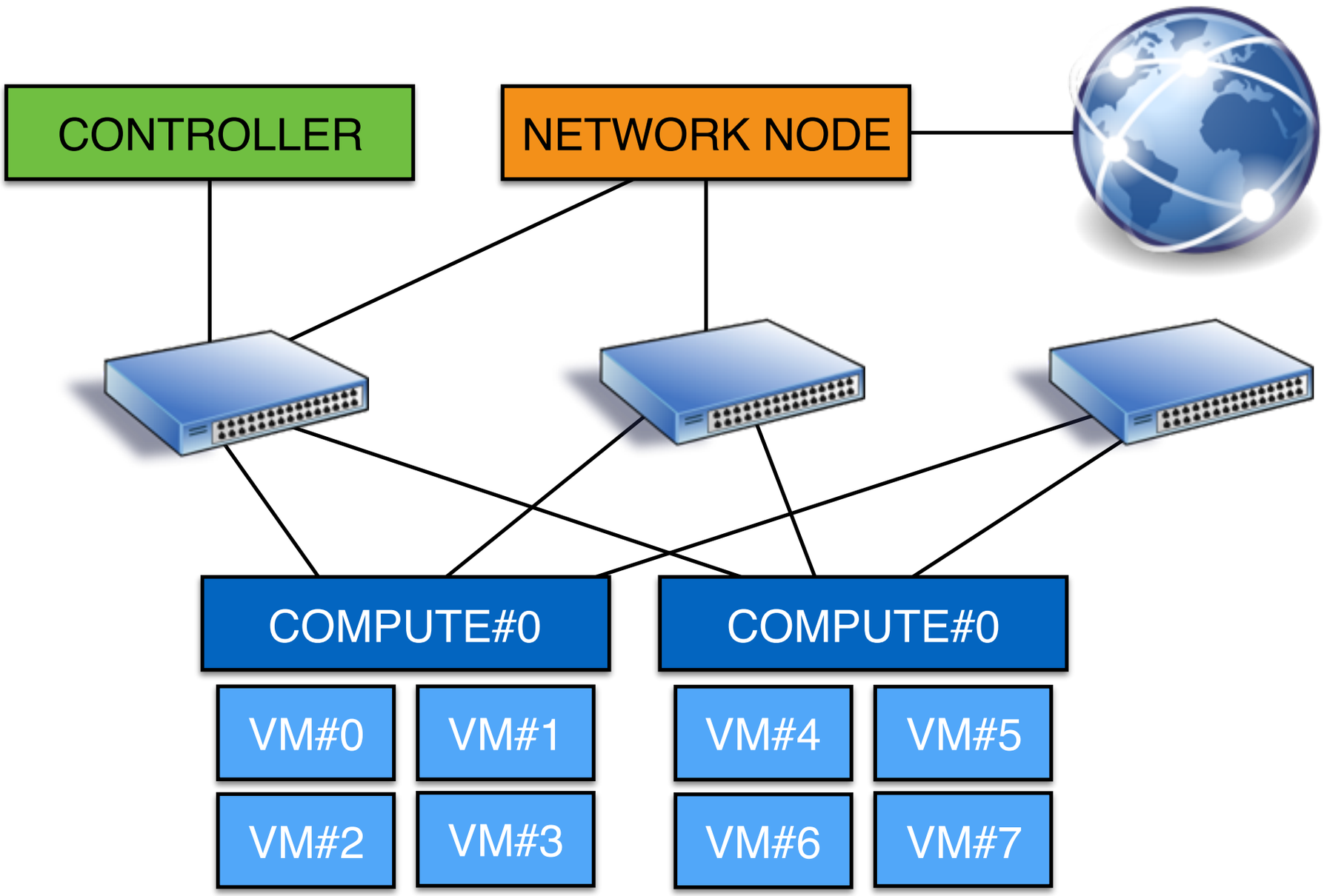
- **HORIZON**: Webベースのダッシュボード
- **HEAT**: システムレベルのオーケストレーションサービス
- **CEILOMETER**: 仮想リソースのメータリングサービス
- **TROVE**: データベースサービス
- **IRONIC**: ベアメタルサーバのデプロイ
- **BARBICAN**: 暗号化に必要な鍵管理サービス
- **DESIGNATE**: DNSサービス
- **SAHARA**: 分散データ処理サービス
- **ZAQAR**: メッセージキューサービス

アーキテクチャ

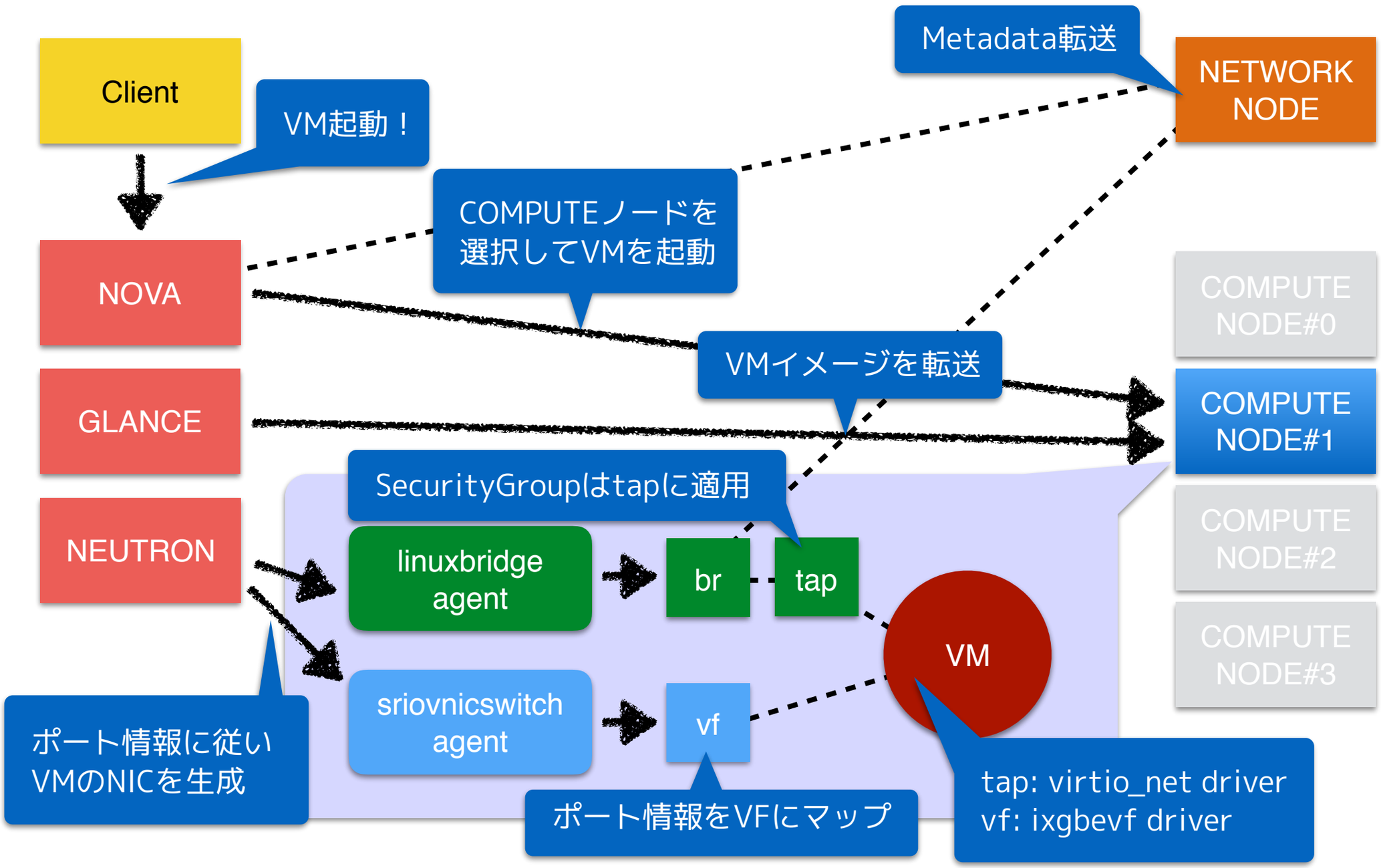


[出展] http://docs.openstack.org/openstack-ops/content/architecture.html

今回紹介するシステム全体構成



SRIOVと仮想マシンの起動(概念図)



DEMO: OpenStackとSR-IOV

CONTROLLERの設定(抜粋)

➡ NOVA

> /etc/nova/nova.confスケジューラにフィルタを追加

```
scheduler_available_filters = nova.scheduler.filters.all_filters
scheduler_default_filters = RetryFilter,AvailabilityZoneFilter,RamFilter,ComputeFilter,
ComputeCapabilitiesFilter, ImagePropertiesFilter, PciPassthroughFilter
```

PCIバスのvendor_id,product_idでVMを起動するVMMを選定

➡ NEUTRON

> /etc/neutron/plugins/ml2/ml2_conf.ini でSR-IOV関連設定を追加

```
[ml2]
mechanism_drivers = linuxbridge,sriovnicswitch
tenant_network_types = vlan
type_drivers = vlan

[ml2_sriov]
agent_required = True
supported_pci_vendor_devs = 8086:10ed
```

sriovnicswitchはl3/dhcp/metadata agent機能をサポートしない。そこでlinuxbridge agentと併用する(OVS agentでも可)

COMPUTEの構成(Intel X540の場合)

➡ キーポイント

SR-IOVのVF情報は、CMDBではなく
設定ファイル(nova.conf)で管理する。

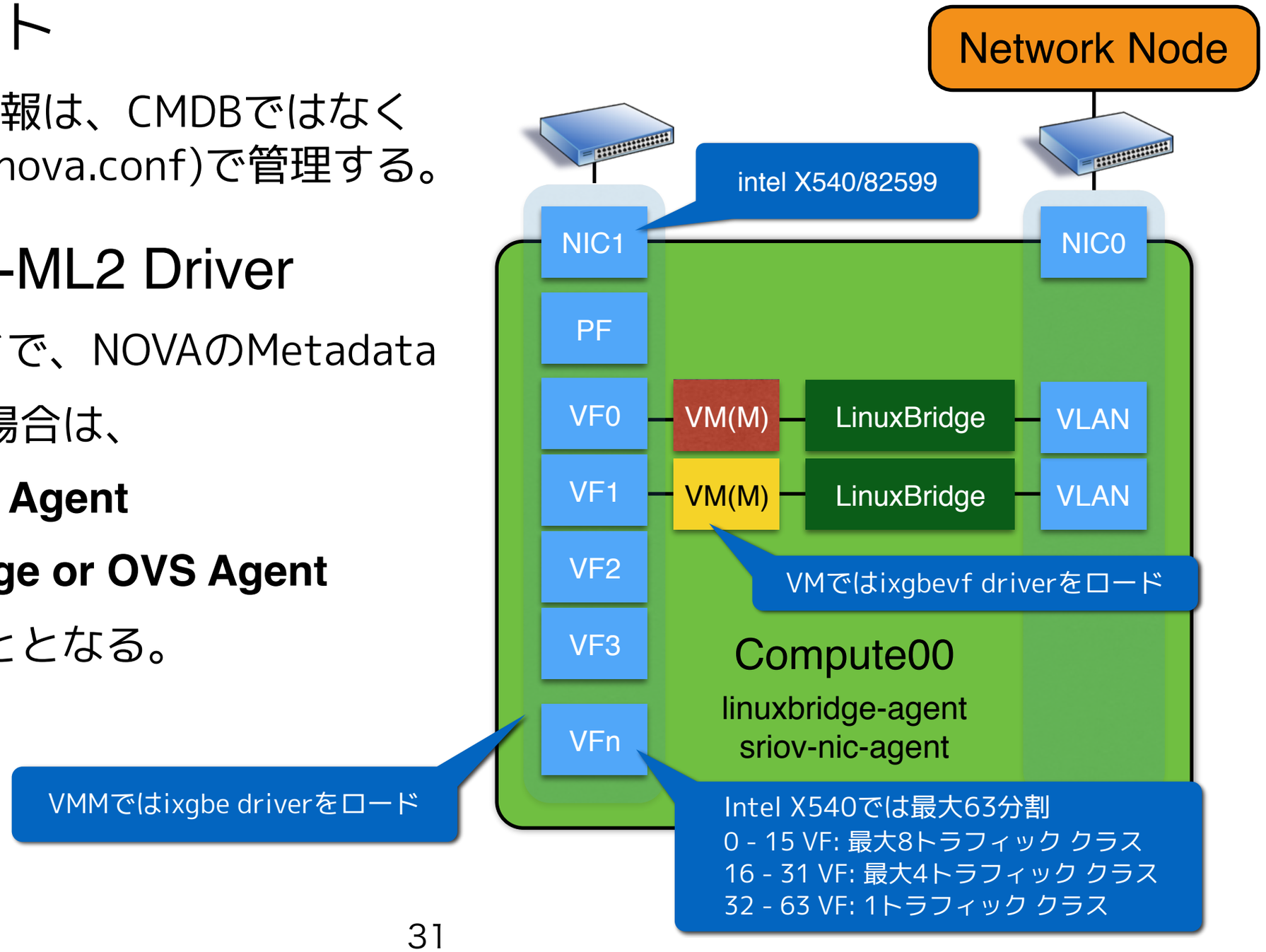
➡ NEUTRON-ML2 Driver

cloud-initなどで、NOVAのMetadata
を利用したい場合は、

(A) SRIOVNIC Agent

(B) LinuxBridge or OVS Agent

を併用することとなる。



COMPUTEの設定(抜粋)

➡ NOVA

> /etc/nova/nova.confスケジューラにフィルタを追加

```
pci_passthrough_whitelist = {"devname": "eno49", "physical_network":  “sriovnet0"}
```

whitelistでPCIパススルー可能なデバイスを指定

➡ NEUTRON

> /etc/neutron/plugins/ml2/ml2_conf.ini でSR-IOV関連設定を追加

```
[ml2]
mechanism_drivers = linuxbridge,sriovnicswitch
tenant_network_types = vlan
type_drivers = vlan
[linux_bridge]
physical_interface_mappings = physnet0:eno3
[sriov_nic]
physical_device_mappings = sriovnet0:eno49
[ml2_sriov]
agent_required = True
supported_pci_vendor_devs =8086:10ed
[ml2_type_vlan]
network_vlan_ranges = physnet0:2000:2400,sriovnet0:2401:2432
```

VMはphysnet0のeno3からmetadataを取得

sriovnet0にはeno49のVFが割り当られる

physnet0,sriovnet0 それぞれにVLANIDのレンジを割り当てる

仮想マシンの起動

➡ 仮想マシンのNICポート作成

> LinuxBridgeにtapするポートを作成する

```
$ neutron port-create ${NETWORK0} --name vm_nic0 --security-group ${SECGROUP}
```

> SR-IOVのVF用ポートを作成する(vnic-typeはdirect)

```
$ neutron port-create ${NETWORK1} --name vm_nic1 --binding:vnic-type direct
```

> 仮想マシンを起動する

```
$ nova boot --flavor m1.standard --image CentOS-7-x86_64-GenericCloud \
--availability-zone rack0 --key-name dev001 --security-groups ${SECGROUP} \
--nic port-id=${vm_nic0のUUID} --nic port-id=${vm_nic1のUUID} vm00
```

OpenStack & SR-IOV 使用上の注意

- PF/VFドライバの問題
  - まだ枯れていないため、思わぬ不具合や機能要件を満たさない場合も…
- sriovnicswitch agentからはmetadataを取得できない

  (A) LinuxBridge Agent または Open vSwitch Agent経由で取得

  (B) 他の手段で初期設定を投入する
- Security Groupが適用されない
  - 仮想マシン側でiptables等の設定と管理を行う仕組みが必要
- PCIパススルー情報はCOMPUTEノードの設定に直書き
  - 現状ではOpenStackの仕様となっている
  - Mitaka Design Summitでも重要な課題としてあげられていた

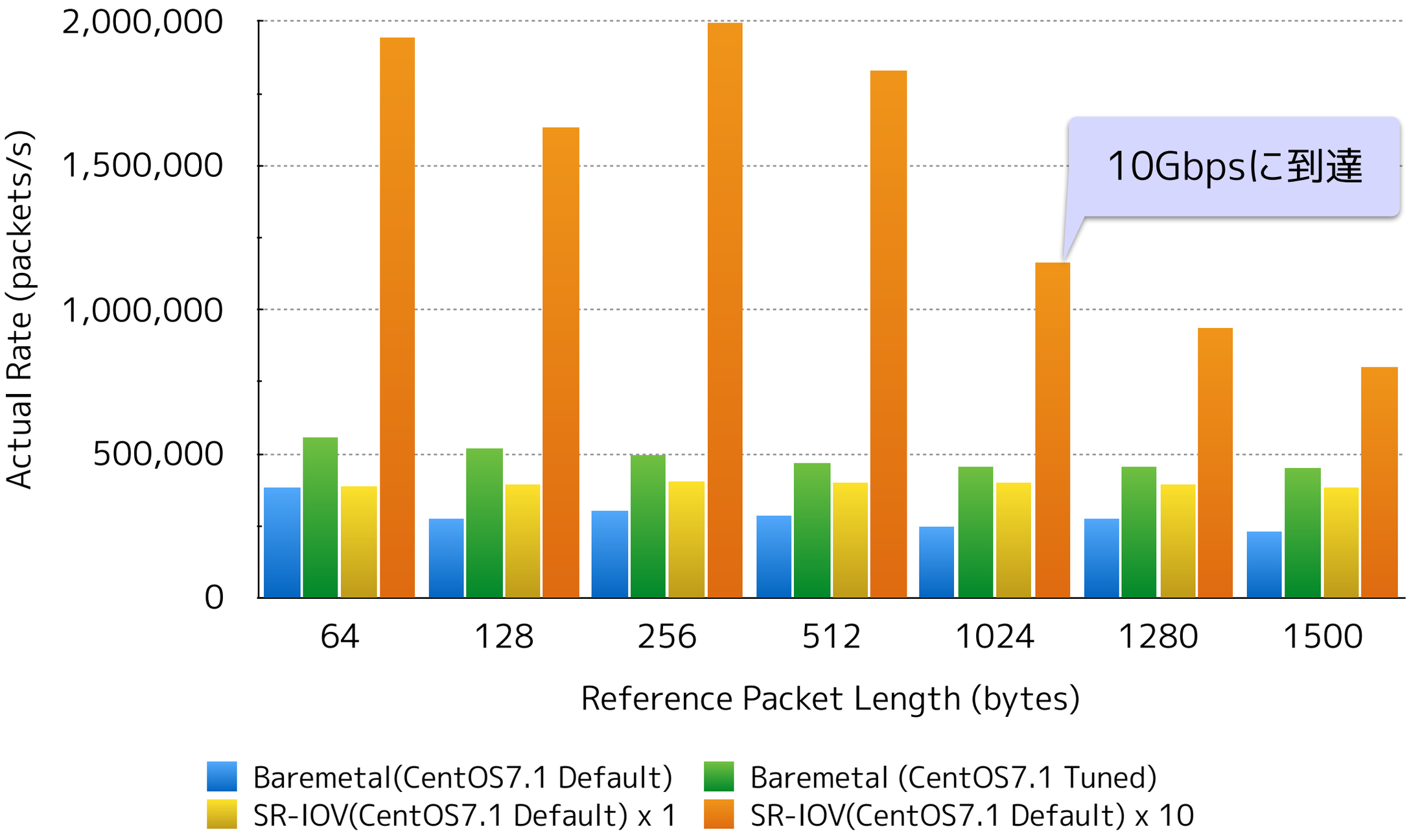
SR-IOV利用の効果

IPフォワーディング性能

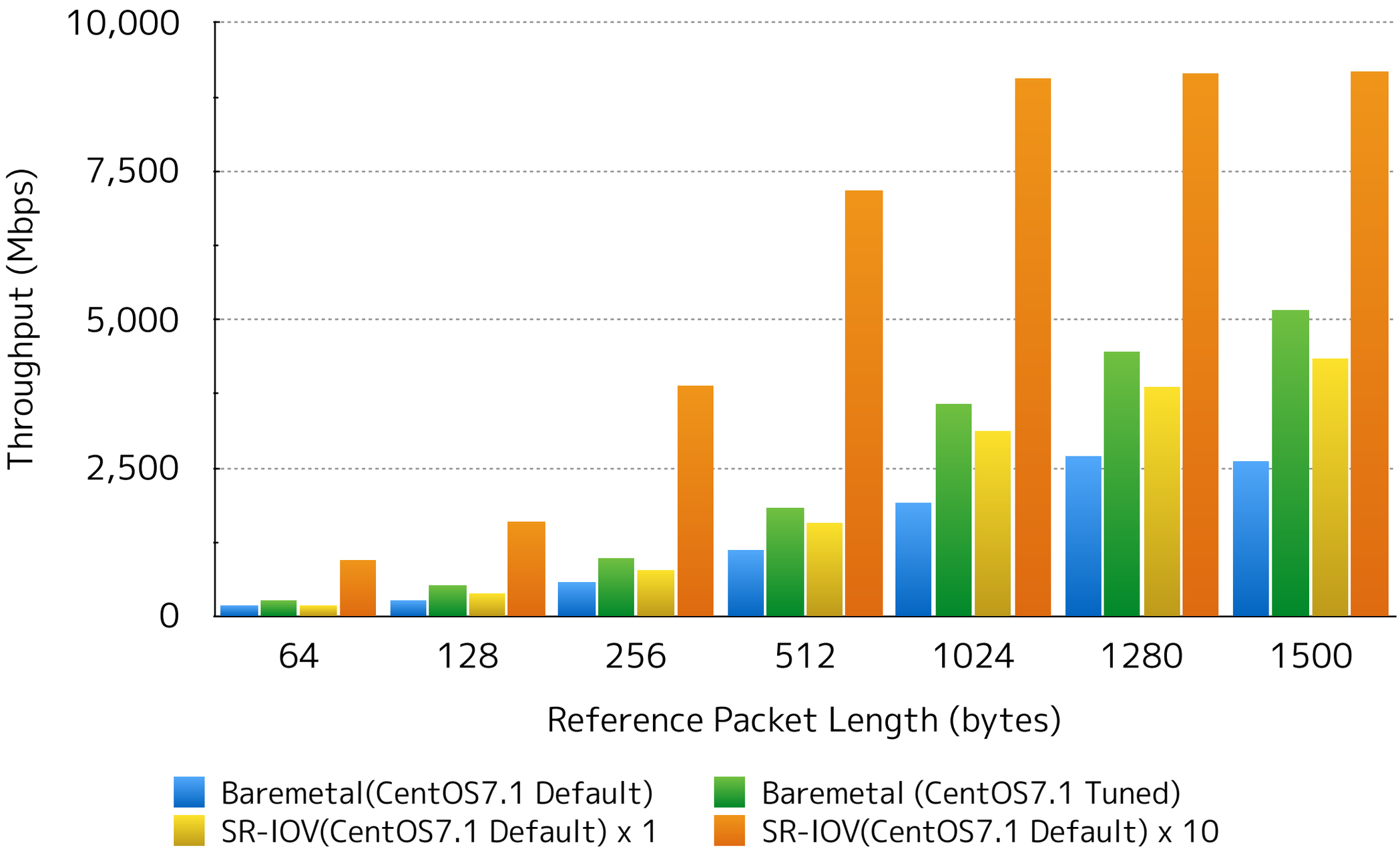
トラフィックジェネレータを利用し、4パターンで性能を計測(RFC2544準拠)

1. ベアメタルサーバ(CentOS 7.1)
2. ベアメタルサーバ(CentOS 7.1) チューニング済み
3. 仮想マシン 1VM/HV (CentOS 7.1 + SR-IOV)
4. 仮想マシン 10VM/HV (CentOS 7.1 + SR-IOV)

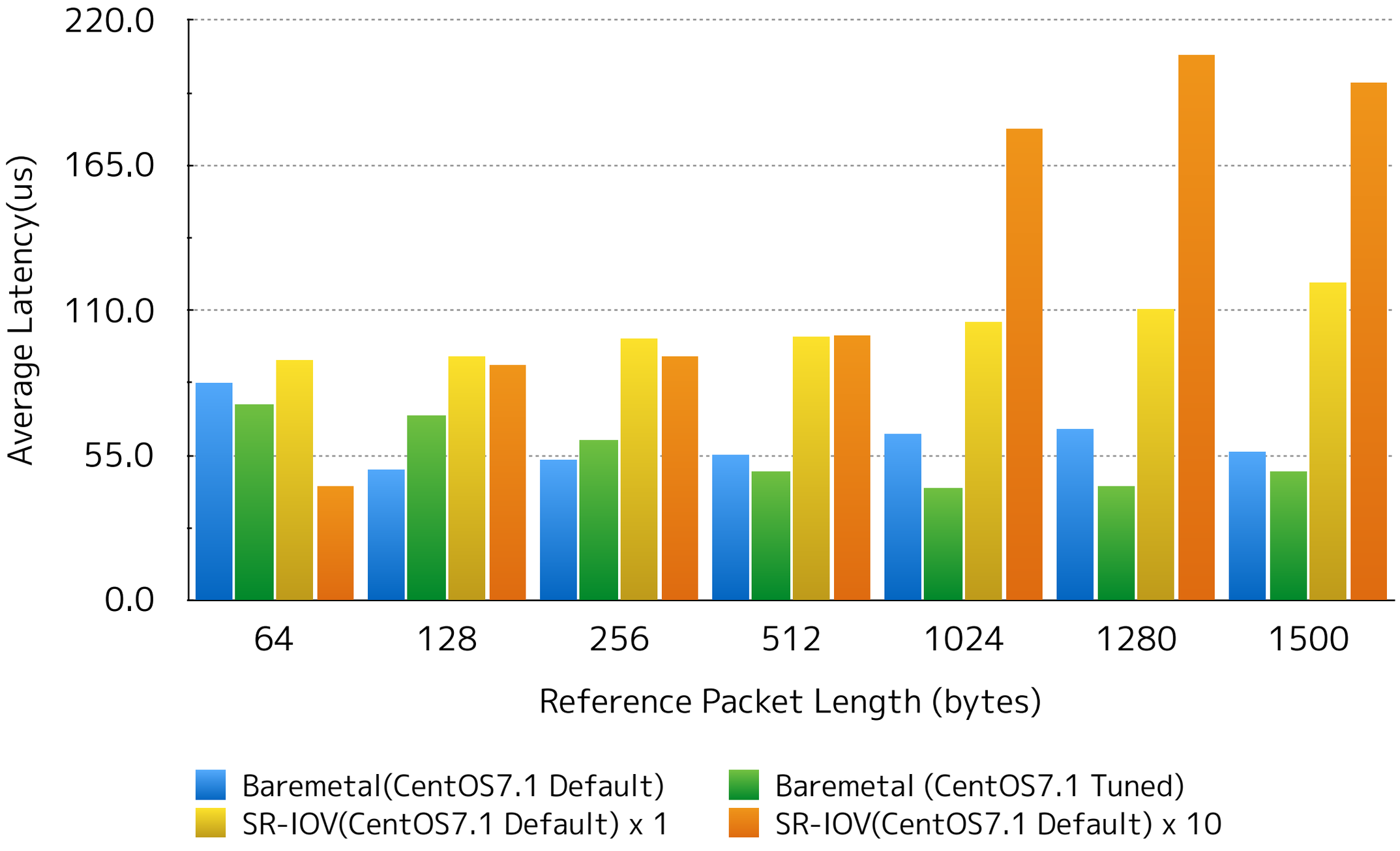
RFC2544 PPS


2,000,000
1,500,000
1,000,000
500,000
0
Actual Rate (packets/s)
10Gbpsに到達
64
128
256
512
1024
1280
1500
Reference Packet Length (bytes)
Baremetal(CentOS7.1 Default)
Baremetal (CentOS7.1 Tuned)
SR-IOV(CentOS7.1 Default) x 1
SR-IOV(CentOS7.1 Default) x 10

RFC2544 Throughput



RFC2544 Latency(us)



まとめ

- 仮想マシンのネットワーク性能を向上させる手法
  - SR-IOVの利点と欠点
  - SR-IOVの活用方法
    - ✓ SR-IOVを利用するための設定
    - ✓ SR-IOVの欠点である管理の複雑さを、OpenStackを利用して解消する

SR-IOVにより仮想マシンのネットワーク性能は飛躍的に向上します。商用製品では早くからサポートされていましたが、OpenStackでもJunoでサポートされた後、日々改善されており、実用レベルまでもう少しという状況まできています。

ご静聴ありがとうございました