クラウド基盤構築演習

第一部
クラウド基盤を支えるインフラ技術
～ 第7回 サーバ仮想化技術の概要

ver1.1 2012/05/01

目次

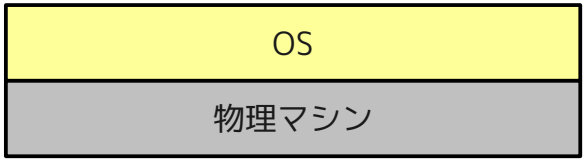
- 仮想化ハイパーバイザの分類
- Xen/KVMのアーキテクチャ
- Linuxの仮想ネットワーク機能
- cgroupsによるリソース制御
- 参考資料: KVMのその他の機能
- 参考資料: LXC（Linuxコンテナ）

仮想化ハイパーバイザの分類

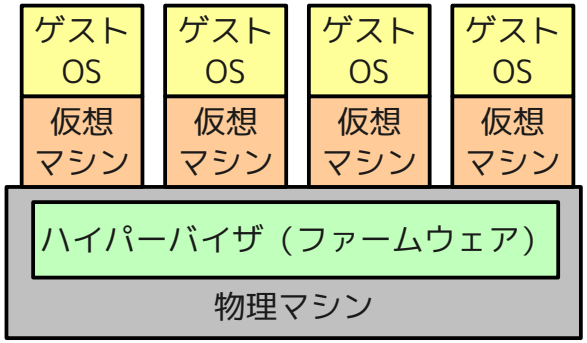
仮想化ハイパーバイザの分類

- 基本的には「物理マシン」と同等の「仮想マシン」を複数作り出す技術です。

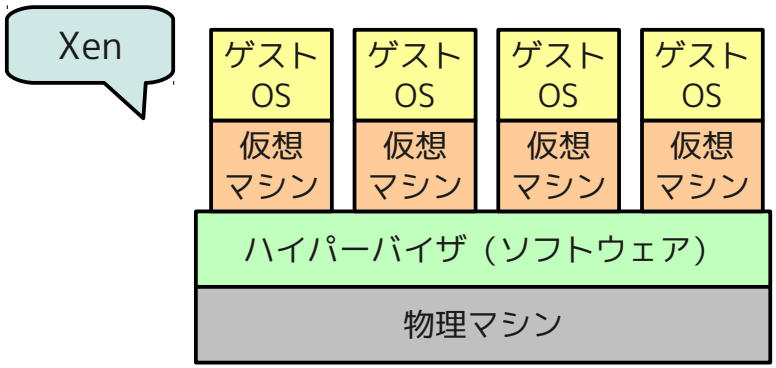
非仮想化環境



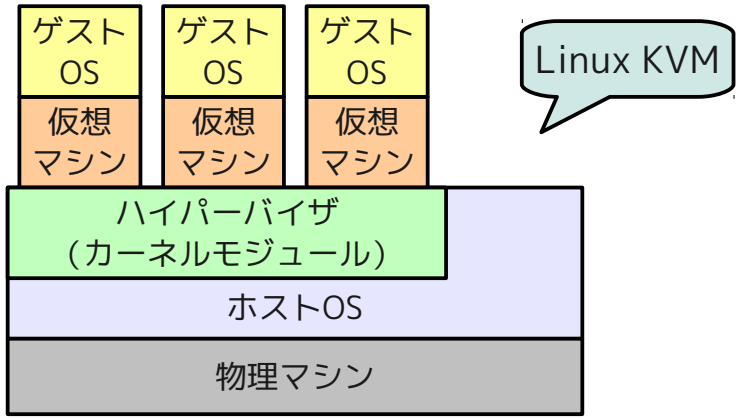
ハードウェアによる仮想化
（物理マシンにハイパーバイザを内蔵）



ソフトウェアによる仮想化
（物理マシン上にハイパーバイザを導入）

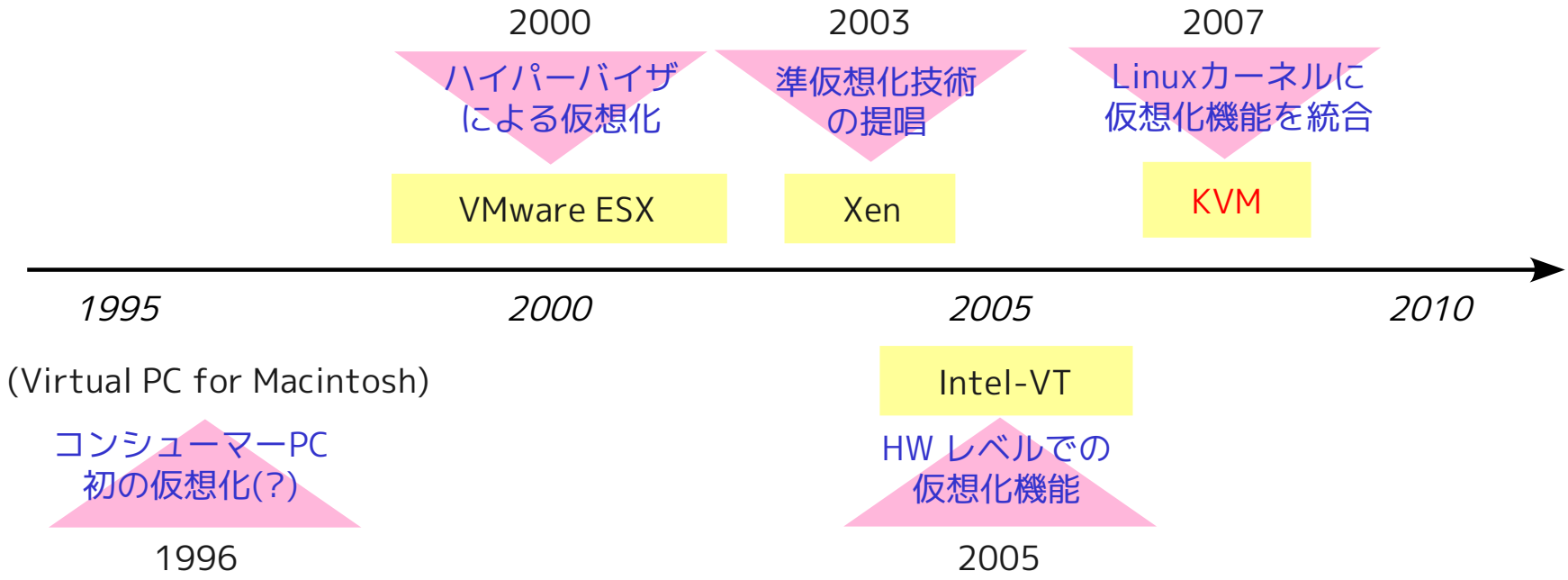


ソフトウェアによる仮想化
（ホストOSにハイパーバイザ機能を追加）



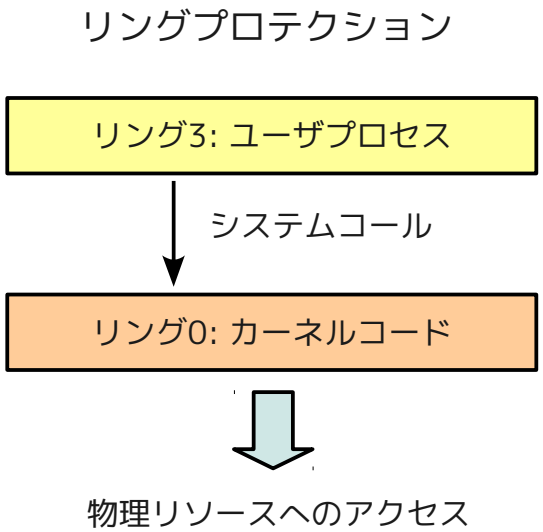
x86サーバ仮想化技術の歴史

- 仮想化支援機能を持つCPUでは、仮想マシン（ゲストOS）を実行するための特権モードを提供することで、センシティブ命令の制御をHWレベルで行います。初期のハイパーバイザでは、仮想化支援機能を前提としないソフトウェア技術が必要でした。
  - VMwareは「Dynamic Binary Translation」を採用しました。センシティブ命令をメモリ上で動的に書き換える技術です。
  - Xenは「準仮想化 (Static Binary Translation)」を提唱しました。センシティブ命令を事前にハイパーバイザコールに書き換えた「準仮想化カーネル」を使用します。
    - 現在のバージョンでは、どちらもCPUの仮想化支援機能を利用可能です。



CPUの特権モードとリングプロテクション

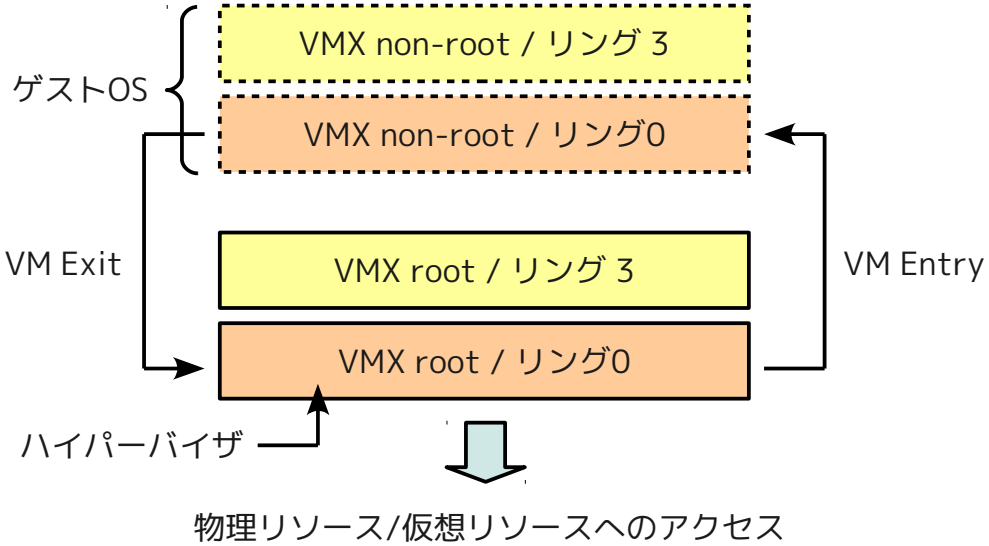
- x86 CPUでは、「リング0～リング3」の4種類の特権モードが提供されます。
  - Linux、WindowsなどのOSでは、リング0とリング3の2種類のモードを使用します。
  - リング0はすべてのCPU命令が実行できるモードで、Linuxカーネルはこのモードで動作します。
  - リング3は物理ハードウェアの操作はできないモードで、ユーザプロセスのコードはこのモードで動作します。ユーザプロセスがハードウェアにアクセスする際は、システムコールによってカーネルコードを呼び出します。



Intel VT-xによる特権モードの拡張

- 仮想化支援機能（Intel VT-x）を持つIntel CPUでは、「VMX rootモード」と「VMX non-rootモード」の区別が追加されてそれぞれに「リング0～リング3」があります。
  - 仮想化ハイパーバイザを使用しない環境では、すべてVMX rootモードで動作します。

- Xenの場合、ハイパーバイザは「VMX root / リング0」で動作して、各ゲストOSは「VMX non-root / リング0、およびリング3」で動作します。
  - 「VMX non-root / リング0」で物理ハードウェアへのアクセスが発生すると、VM Exitが発行されて、CPUは強制的に「VMX root / リング0」のハイパーバイザに処理を移行します。



メモとしてお使いください

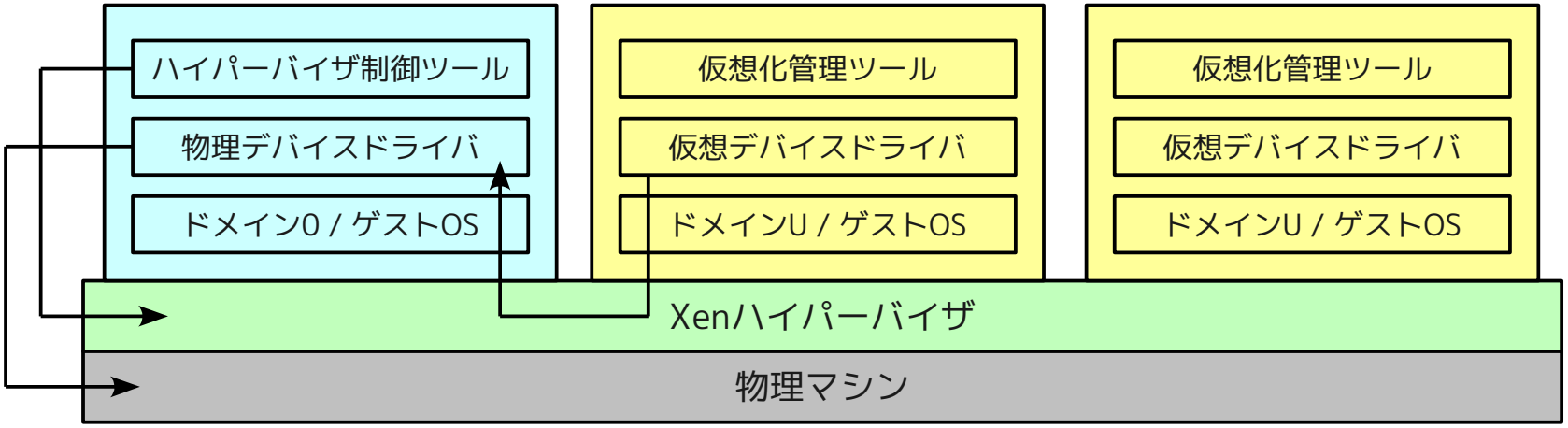
メモとしてお使いください

メモとしてお使いください

Xen/KVMのアーキテクチャ

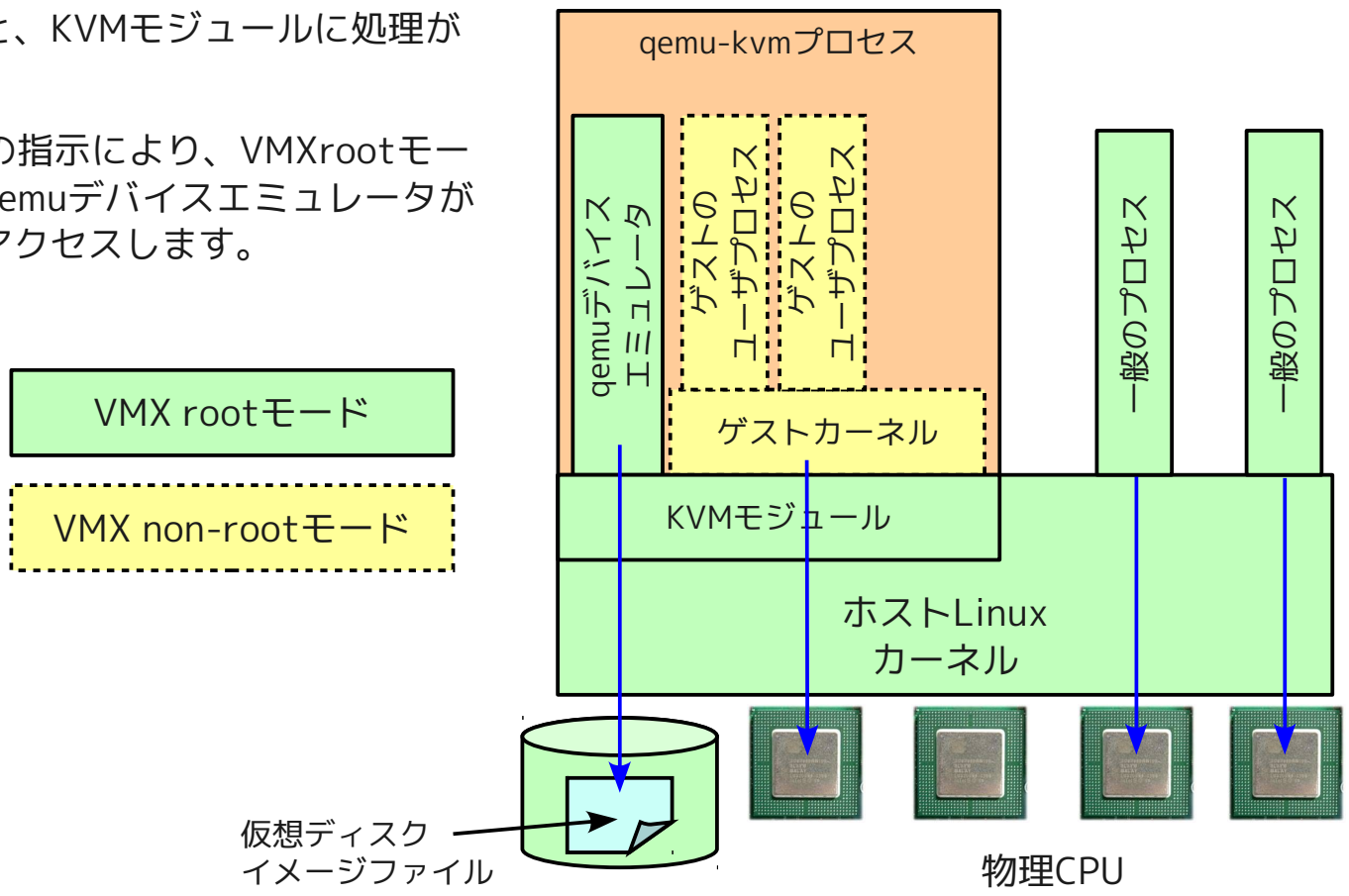
Xenのアーキテクチャ概要

- Xenハイパーバイザを介して複数の仮想マシンを作成して、それぞれで独立したゲストOS を稼動します。
  - 特に「ドメイン0」と呼ばれる特権ドメインのゲストOS 上で、ハイパーバイザ制御ツールと物理デバイスドライバが稼動します。
  - ハイパーバイザ制御ツールから、一般ドメイン（ドメインU）の作成とゲストOSの起動を行います。
  - ドメインUの仮想デバイスドライバは、Xenハイパーバイザを介して、ドメイン0を経由して物理デバイスにアクセスします。



KVMにおける特権モードの利用方法

- KVMでは、仮想マシンはホストLinux上の1プロセスとして実行されます。
  - ゲストOSは、VMX non-rootモードで動作して、デバイスアクセスなどのセンシティブ命令が発行されると、KVMモジュールに処理が移行します。
  - KVMモジュールの指示により、VMXrootモードで動作する、qemuデバイスエミュレータが物理デバイスにアクセスします。

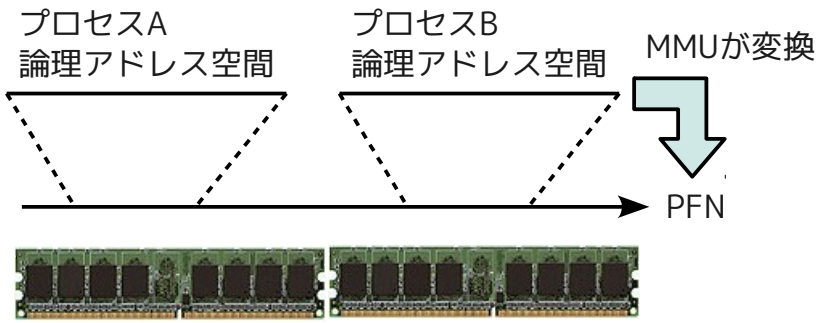


EPTによるメモリアクセスの高速化

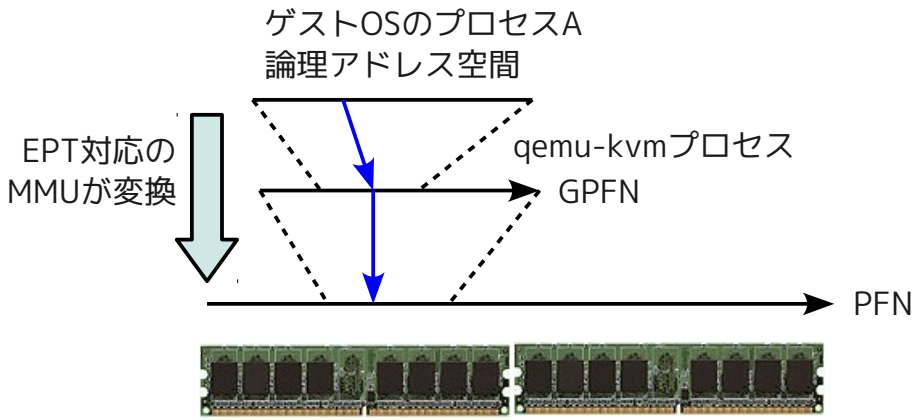
- 非仮想化環境では、ユーザプロセスが認識する仮想アドレスと物理アドレスの変換はCPUに搭載のMMU(Memory Management Unit)がハードウェアレベルで処理します。

- 一方、仮想マシン環境では、2 段階のアドレス変換が必要です。
  - ユーザプロセスの仮想アドレス ⇒ ゲストOSの「仮想」物理アドレス(GPFN) ⇒ 物理アドレス(MPFN)
  - EPT (Extended Page Tables) を利用すると、2段階のアドレス変換をハードウェアレベルで処理します。
    - Xeon 5500番台(Nehalem)以降のCPUがEPTに対応しています。EPT未対応のCPUでは、シャドウページテーブルなどのソフトウェア処理が必要です。

非仮想化環境



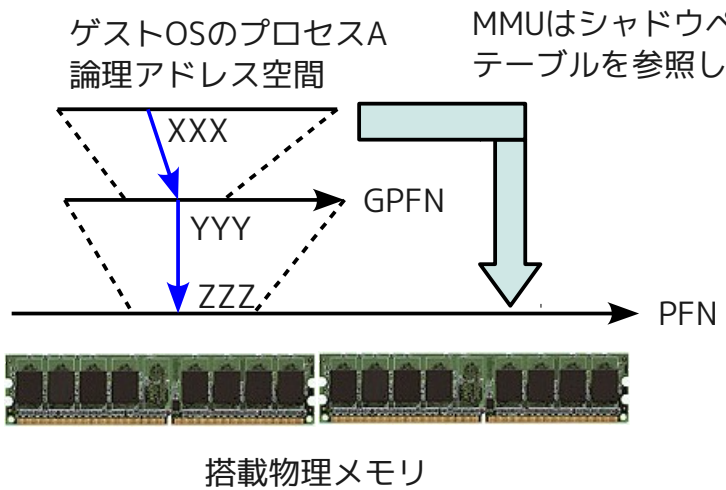
Xen/KVM仮想化環境



シャドウページテーブルの利用

- EPT未対応のCPUでXen/KVMを使用する場合は、ハイパーバイザが管理するシャドウページテーブルを利用してアドレス変換を行います。
  - ゲストOSがプロセスのページテーブル（論理アドレスとGPFNの対応表）を更新すると、それを検知したハイパーバイザは、論理アドレスをPFNに直接変換する「シャドウページテーブル」を更新します。
  - ハイパーバイザは、MMUが、プロセスAのページテーブルではなく、対応するシャドウページテーブルを参照してアドレス変換を行うように操作をします。
    - ハイパーバイザによる余分なソフトウェア処理が発生するので、EPT対応CPUに比べてメモリアクセスのオーバヘッドが大きくなります。

EPT未対応CPUでのXen/KVM仮想化環境



ゲストOS上の
プロセスAのページテーブル

論理アドレス	GPFN
XXX	YYY
・・・	・・・

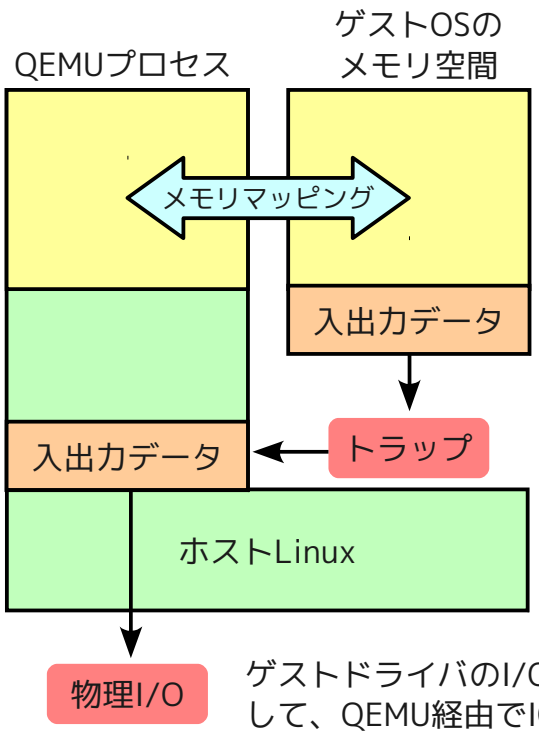
ハイパーバイザが管理する
シャドウページテーブル

論理アドレス	PFN
XXX	ZZZ
・・・	・・・

virtioドライバによるI/Oの高速化

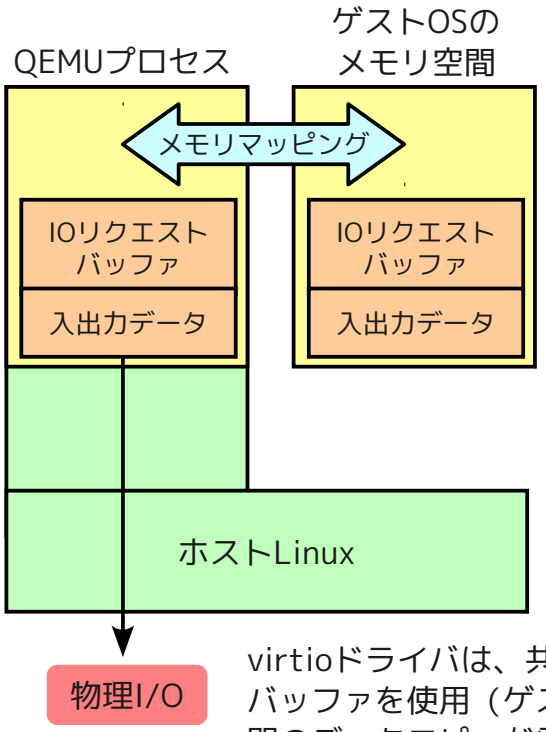
- virtioは、KVM仮想化環境に最適化されたディス/NICのエミュレーション形式です。
  - QEMUが用意したI/Oバッファ領域にvirtioドライバが直接アクセスするため仮想化のオーバヘッドが削減されます。
  - Xenにおける「準仮想化ドライバ」と同等の仕組みになります。

一般のドライバを使用する場合



ゲストドライバのI/O処理をトラップして、QEMU経由でIOを実施

virtioドライバを使用する場合



virtioドライバは、共有メモリ上のバッファを使用（ゲストOSとQEMU間のデータコピーが発生しない）

仮想化APIライブラリ（libvirt）

- libvirtは、複数のハイパーバイザを統一的に操作するAPIライブラリです。
  - C言語、Pythonから使用するためのライブラリを標準で提供します。現在は、Xen/KVM/LXC（Linuxコンテナ）などが対応しています。
  - ゲストOSの起動・停止などの操作以外に、仮想ネットワーク構成、ストレージ管理のためのAPIを提供します。
  - 図は、Python用のライブラリを使用して、KVMのすべての仮想マシンをまとめて起動／停止するスクリプトの例です。

すべての仮想マシンを起動するスクリプト

```
#!/usr/bin/python

import libvirt, time
Conn = libvirt.open( "qemu:///system" )
for name in Conn.listDefinedDomains():
    vm = Conn.lookupByName( name )
    print "Starting " + vm.name()
    vm.create()
    time.sleep( 1 )
```

すべての仮想マシンを停止するスクリプト

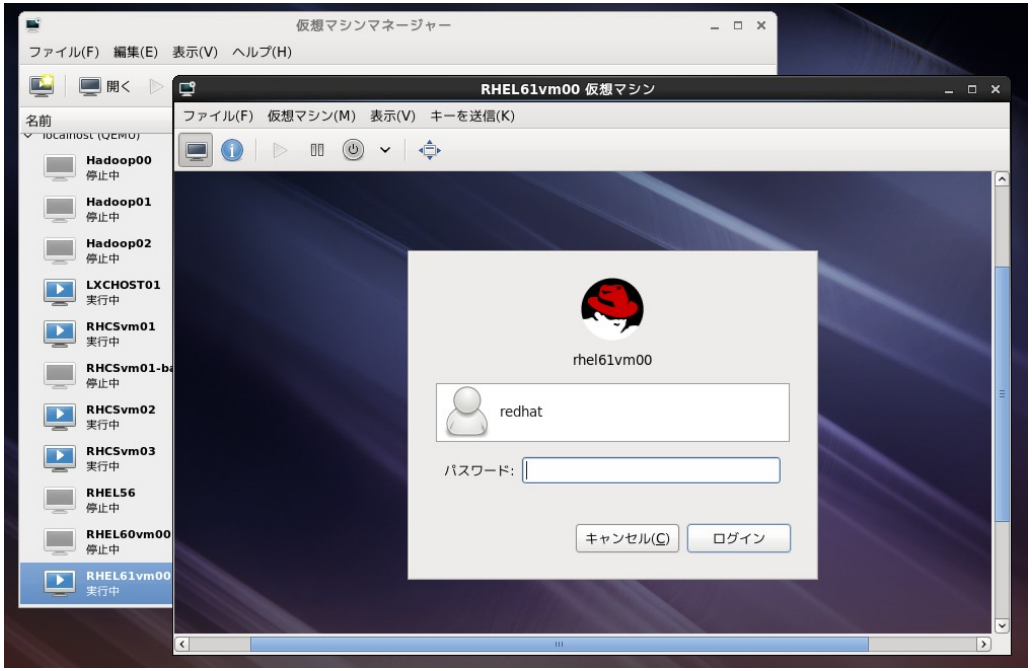
```
#!/usr/bin/python

import libvirt, time
Conn = libvirt.open( "qemu:///system" )
for id in Conn.listDomainsID():
    vm = Conn.lookupByID( id )
    print "Stopping " + vm.name()
    vm.shutdown()
    time.sleep( 1 )
```

- RHEL6では次のようなツールがlibvirtを利用しています。
  - virsh（コマンドライン管理ツール）
  - virt-manager（GUI管理ツール）
  - virt-install（コマンドラインのゲストOSインストールツール）
- Euclyptusなどのクラウド基盤ソフトウェアが内部的にlibvirt経由で仮想化ハイパーバイザを操作する場合もあります。

virt-managerによる仮想化環境の管理

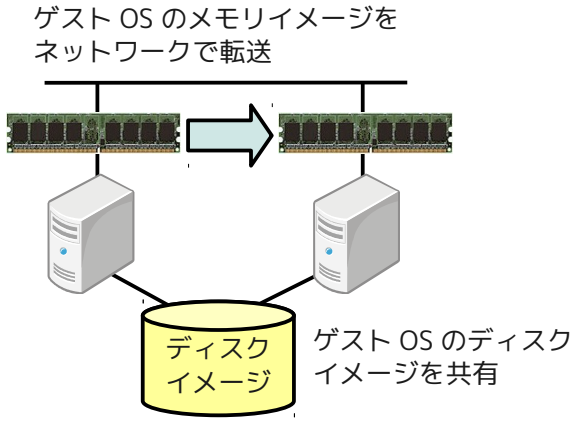
- Virt-managerはRHEL6が標準で提供する仮想化環境の管理ツールです。次のような操作をGUIで行うことができます。
  - 仮想マシンの構成、ゲストOSのインストール
  - 仮想マシンのコンソール表示、起動、停止
  - 仮想ネットワークの構成、仮想ディスク用ストレージの管理
  - など
- ネットワーク経由で複数の仮想化ホストを管理することもできます。



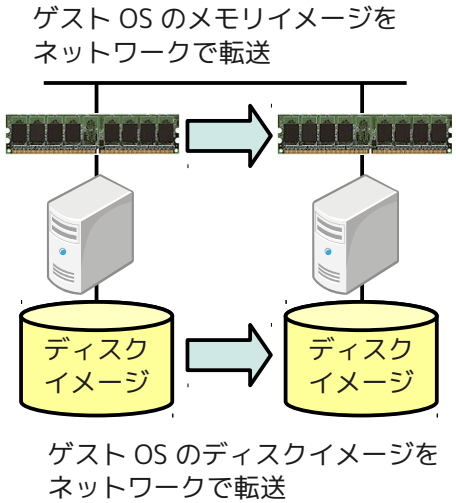
ライブマイグレーションの仕組み

- ライブマイグレーションは、稼動中のゲストOSを、稼動状態を保ったまま異なる物理サーバ上に移動する技術です。
  - ディスクイメージは、共有ディスク装置を利用して、サーバ間で共有します。
  - 稼動中のゲストOSのメモリイメージをネットワーク経由で転送します。メモリイメージの転送後、仮想CPUの状態を転送することで、マイグレーションが完了します。
  - 外部のクライアントから見ると、厳密には10~100msec程度の停止が発生しますが、TCP/IP の再送機能により、通常はネットワーク切断などの問題は発生しません。
- 共有ディスクを使用しない「ストレージ・ライブマイグレーション」も可能です。
  - 稼働中のゲストOSについて、メモリイメージに加えて、仮想ディスクイメージの内容もネットワーク経由で転送します。

ライブマイグレーション



ストレージ・ライブマイグレーション



メモとしてお使いください

メモとしてお使いください

メモとしてお使いください

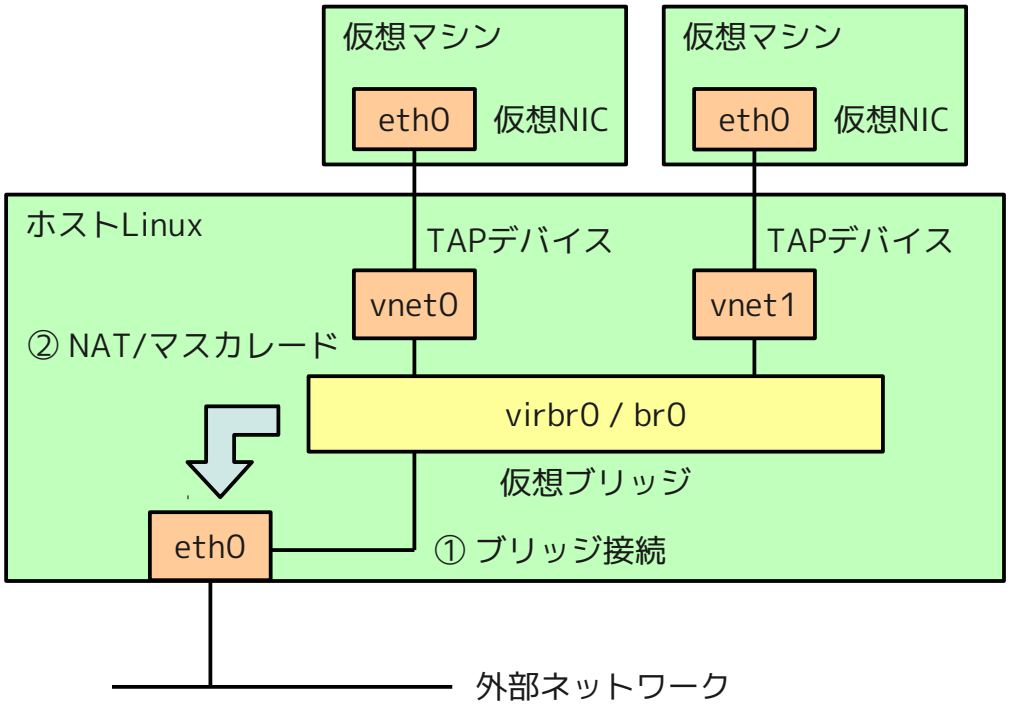
Linuxの仮想ネットワーク機能

KVMの仮想ネットワーク

- KVMのホストLinuxは「TAPデバイス」を経由して仮想マシンの仮想NICとパケットを交換します。TAPデバイスを「仮想ブリッジ」に接続することで、仮想マシン間のプライベートネットワークが構成されます。
  - TAPデバイスは、ユーザプロセスと仮想的なネットワーク通信を行うLinuxの機能です。
  - 仮想ブリッジは、Linux上に仮想的なネットワークスイッチを構成する機能です。

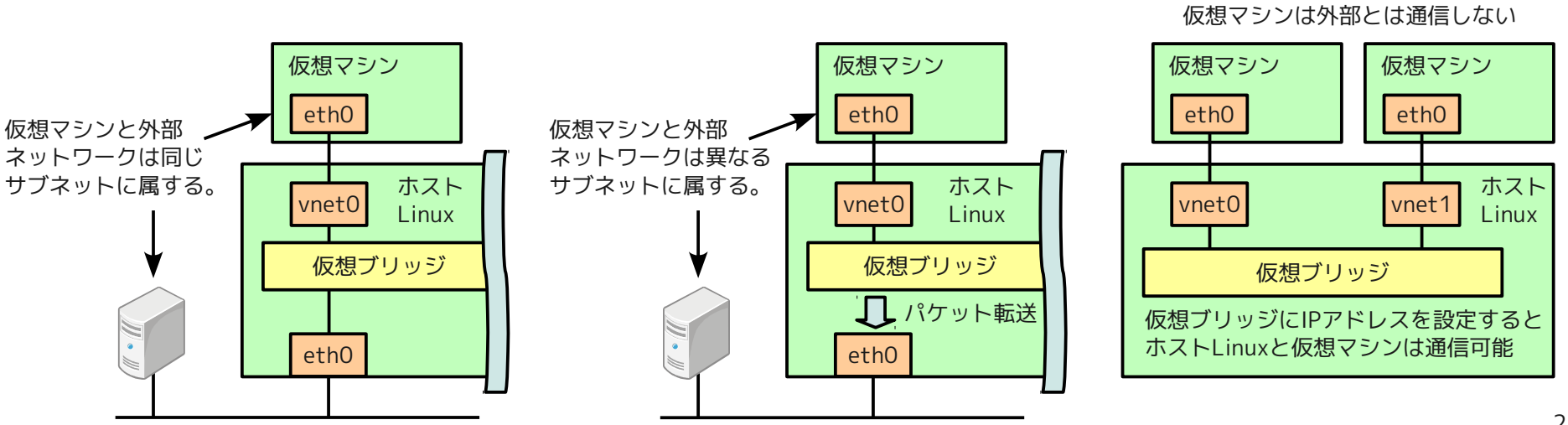
- 外部ネットワークに接続する方法は2種類あります。
  - ホストLinuxの物理NICを仮想ブリッジに接続します。
  - iptablesによるNAT/マスカレードで仮想ブリッジから外部ネットワークにパケット転送します。

- ホストLinuxをドメイン0に読み替えると、Xenでもほぼ同じ仕組みになります。



仮想ネットワークの構成パターン

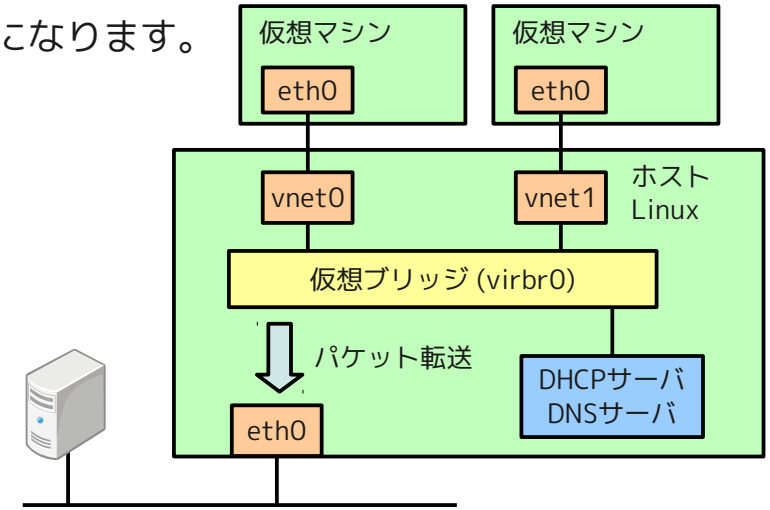
- 複数の仮想ブリッジを使用することで、複数の仮想ネットワークが構成できます。仮想ネットワークの構成パターンには次のようなものがあります。
  - 物理NICをブリッジ接続して外部ネットワークに同一サブネットとして接続
  - NAT/マスカレードを利用して、外部ネットワークとは独立したサブネットとして接続
    - マスカレードを利用場合は、外部ネットワークから仮想ネットワークに向けた接続はできません。
    - 外部ネットワークと相互通信する場合は、外部のIPアドレスと内部のIPアドレスをDNAT/SNATで1対1にひも付ける必要があります。
  - 外部ネットワークには接続しないホストマシン内部のプライベートネットワークを構成
    - 仮想ブリッジにIPアドレスを設定することで、ホストLinux(Xenの場合はドメイン0）と仮想マシンの通信は可能になります。



dnsmasqによるDHCP/DNS機能の提供

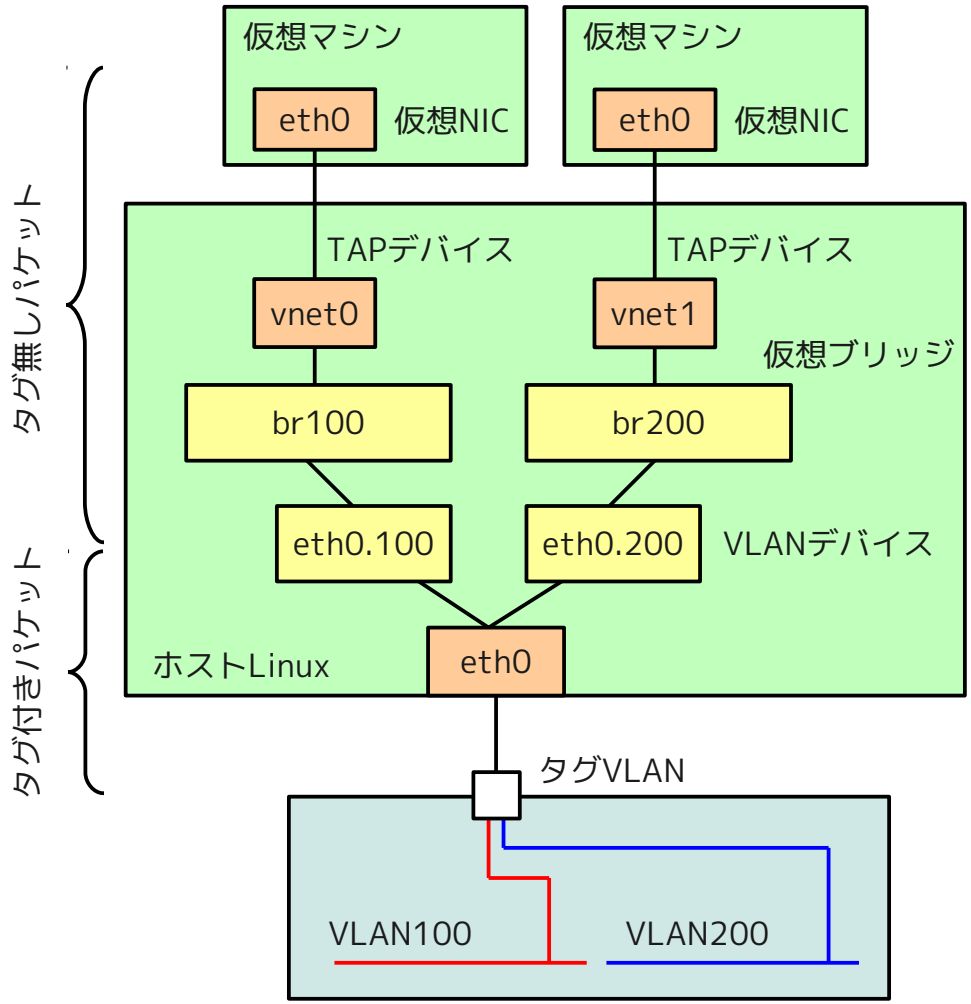
- 物理NICをブリッジ接続しないタイプの仮想ネットワークは、libvirt API（virsh/virt-managerなど）で構成することができます。
  - 物理NICのブリッジ接続を行う場合は、手動で設定ファイルを構成する必要があります。
  - 仮想ブリッジには任意の名称を設定できます。通常は「br0」などを使用します。

- libvirtで構成する仮想ネットワークは、簡易的なDHCP/DNS機能を提供します。
  - これらの機能は、ホストLinuxの「dnsmasqデーモン」が提供します。不要な場合は無効化もできます。DNSサーバは、ホストLinuxの「/etc/hosts」の内容と「/etc/resolv.conf」で指定された外部DNSを参照します。
  - libvirtで構成した仮想ブリッジの名称は「virbrX」になります。



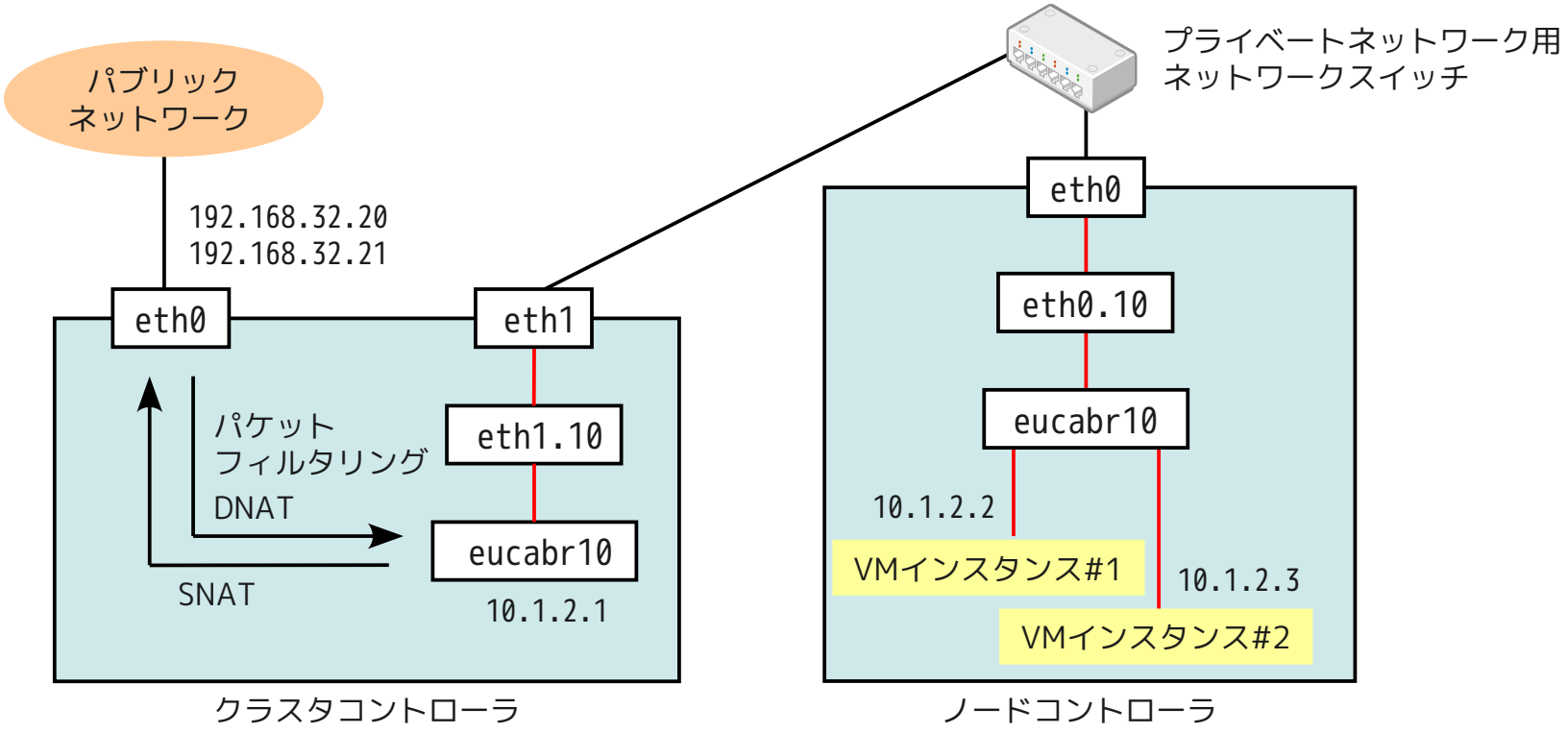
VLANデバイスの組み合わせ

- ホストLinuxでVLANデバイスを構成することにより、仮想マシンを透過的にVLANに接続することができます。
  - VLANタグの処理は、ホストLinux上で行われるので、ゲストOSではVLANを意識した設定は必要ありません。



（参考）Eucalyptusにおけるネットワーク接続

- 仮想マシン（VMインスタンス）に付与されたパブリックIPは、実際にはクラスタコントローラの物理NICにアサインされます。
  - クラスタコントローラがパブリックネットワークから受信したパケットをDNATでプライベートIPに変換して、VMインスタンスに転送します。逆に、VMインスタンスからパブリックネットワークに送信したパケットは、SNATでパブリックIPに変換して送出します。
  - セキュリティグループごとのパケットフィルタリングもこの部分で行います。



メモとしてお使いください

メモとしてお使いください

cgroupsによるリソース制御

Control Groups (cgroups)とは

- Linuxはさまざまなプロセス単位のリソース制御機能を持っていいます。cgroupsは、これらの機能をプロセスのグループ単位で適用するためのインターフェースです。
  - cgroups自体がリソース制御機能を提供するわけではありません。cgroupsのインターフェースを利用するリソース制御機能を「サブシステム」と呼びます。

RHEL6.2で利用できる主要なサブシステムの機能

サブシステム	説明
cpuset	・グループ内のプロセスが使用するCPUコアを指定
cpu	・グループ内のプロセスが使用するCPU時間割合の上限を設定 ・複数グループのプロセスが同一のCPUコアで実行される場合の優先順位（CPU時間配分の割合）を設定
memory	・使用可能メモリの上限（実メモリ、実メモリ+スワップ）を指定
blkio	・物理I/Oの優先順位と帯域を制御
devices	・/dev以下のデバイスファイルへのアクセス制御

cgroupsのグループ管理

- /cgroupにマウントされた特殊ファイルシステムの形式でグループを管理します。ディレクトリのツリー構造がグループのツリー構造に対応します。
  - 「cgconfigサービス」を起動すると「/cgroup」に特殊ファイルシステムがマウントされます。
  - ディレクトリがグループに対応しており、mkdirコマンドでディレクトリを作成すると新しいグループが定義されたことになります。

すべてのグループの親になる「ルートグループ」

```
/cgroup/<サブシステム名>/group01/subgroup01/subsubgroup01
                        |       |
                        |       /subgroup02
                       /group02
```

  - ツリー構造のグループ管理に対応しないサブシステムの場合は、上図の「subgroupXX」以下のディレクトリは作成できません。

- グループに対応するディレクトリの下にある擬似ファイルで設定を行います。
  - 「tasks」ファイルにPIDをecho出力すると、そのプロセスは該当グループに入ります。
    - これ以降は、このプロセスから作成される子プロセスもすべて同じグループに入ります。
  - 「cgroup.procs」ファイルをcatで表示すると、そのグループに含まれるプロセスのPIDを表示します。
  - サブプロセスのパラメータを設定する際は、対応する擬似ファイルに数値をecho出力します。

主要サブシステムのパラメータ(1)

- cpusetサブシステム: グループ内のプロセスを実行するCPUコアを指定

パラメータ	説明
cpuset.cpu	プロセスを実行するCPUコアを指定。「0-2,7」は「0,1,2,7」という意味になる。
cpuseet.cpu_exclusive	1をセットすると、このグループが指定するCPUコアを他のグループでは指定できない。（デフォルトは0）
cpuset.mems	NUMAサーバにおいて、プロセスが利用するメモリの位置を指定。(デフォルトでは値がセットされないので、新しいグループを作成したら、最初はルートグループと同じ値をセットしておく）

- cpuサブシステム: CPUコアの実行時間配分を指定

パラメータ	説明
cpu.shares	CPUコアの占有率を2～262144の数値で指定。 複数グループのプロセスが同一のCPUコアを使用する場合、この値に比例したCPU コア使用時間が割り当てられる。（デフォルト値は1024）
cpu.cfs_period_us cpu.cfs_quota_us	CPU使用時間割合の上限を設定。「cpu.cfs_period_us」(μs)中の「cpu.cfs_quota_us」(μs)まで、このグループのプセスはCPUを使用できる。 マルチコア環境では「cpu.cfs_quota_us」は「cpu.cfs_period_us」を越えて設定可能。（4コア環境で「cpu.cfs_period_us=250000, cpu.cfs_quota_us=500000」とした場合、このグループは最大2コア分のCPU性能が得られる。）

主要サブシステムのパラメータ(2)

- memoryサブシステム: メモリ使用量の上限を指定

パラメータ	説明
memory.limit_in_bytes	該当グループが利用可能な物理メモリの上限をバイト単位で指定。
memory.memsw.limit_in_bytes	該当グループが利用可能な「物理メモリ+スワップ領域」の上限をバイト単位で指定。
memory.use_hierarchy	1をセットするとグループのメモリ使用量として、サブグループのプロセスのメモリ使用量も加えられる。（デフォルトは0）

- blkioサブシステム: ディスクI/Oの優先順位を指定

パラメータ	説明
blkio.weight	全ブロックデバイスに共通の優先順位を100～1000の値で指定。 この値に比例したI/O処理時間が割り当てられる。
blkio.weight_device	特定デバイスに対する優先順位を「Major:Minor 指定値」の形式で指定 （例「8:0 500」）※パーティション単位での指定はできない。
blkio.throttle.read_bps_device blkio.throttle.write_bps_device	特定のデバイスに対するアクセス速度の上限をBytes/Sec単位で指定。0を指定すると制限を解除 。 （例「8:0 1048576」）※パーティション単位での指定はできない。
blkio.throttle.read_iops_device blkio.throttle.write_iops_device	特定のデバイスに対するアクセス速度の上限をIOPS単位で指定。0を指定すると制限を解除 。 （例「8:0 2048」）※パーティション単位での指定はできない。

仮想マシンに対するcgroupsの操作

- libvirt API（virt-manager/virshコマンド）から仮想マシンを起動すると、該当の仮想マシンに対するcgroupsの設定を行うためのディレクトリが自動的に作成されます。
  - 具体的には「/cgroup/<サブシステム>/libvirt/qemu/<VM名>」になります。
  - このディレクトリ内のパラメータに値を設定することで、該当の仮想マシンに対する設定が可能です。

```
# ls -lF /cgroup/cpu/libvirt/qemu
合計 0
drwxr-xr-x. 4 root root 0  1月 13 20:11 2012 RHEL62vm01/
drwxr-xr-x. 3 root root 0  1月 13 20:15 2012 RHEL62vm02/
-r--r--r--. 1 root root 0 12月 26 11:55 2011 cgroup.procs
-rw-r--r--. 1 root root 0 12月 26 11:55 2011 cpu.cfs_period_us
-rw-r--r--. 1 root root 0 12月 26 11:55 2011 cpu.cfs_quota_us
-rw-r--r--. 1 root root 0 12月 26 11:55 2011 cpu.rt_period_us
-rw-r--r--. 1 root root 0 12月 26 11:55 2011 cpu.rt_runtime_us
-rw-r--r--. 1 root root 0 12月 26 11:55 2011 cpu.shares
-r--r--r--. 1 root root 0 12月 26 11:55 2011 cpu.stat
-rw-r--r--. 1 root root 0 12月 26 11:55 2011 notify_on_release
-rw-r--r--. 1 root root 0 12月 26 11:55 2011 tasks
```

cgroupsによるリソース制御の例

- 2個の仮想マシンのディスクI/O帯域を10MB/Secと50MB/Secに制限して、各仮想マシンでI/O負荷をかける例です。

```
# echo "8:0 50000000" > /cgroup/blkio/libvirt/qemu/RHEL61vm01/blkio.throttle.write_bps_device
# echo "8:0 10000000" > /cgroup/blkio/libvirt/qemu/RHEL61vm02/blkio.throttle.write_bps_device

# virt-top -3 -d 1
virt-top 16:18:06 - x86_64 4/4CPU 3101MHz 7751MB
22 domains, 3 active, 3 running, 0 sleeping, 0 paused, 19 inactive D:0 O:0 X:0
CPU: 2.4%  Mem: 4096 MB (ゲストによる 4096 MB)

   ID S RDBY WRBY RDRQ WRRQ DOMAIN       DEVICE
   14 R    0  48M    0   96 RHEL61vm01   vda
   15 R    0  10M    0   22 RHEL61vm02   vda
```

- 2個の仮想マシンのCPU使用時間を0.5コアと2.0コアに制限して、各仮想マシンでCPU負荷をかける例です。

```
#  echo "500000" > /cgroup/cpu/libvirt/qemu/RHEL61vm01/cpu.cfs_period_us
#  echo "250000" > /cgroup/cpu/libvirt/qemu/RHEL61vm01/cpu.cfs_quota_us
#  echo "500000" > /cgroup/cpu/libvirt/qemu/RHEL61vm02/cpu.cfs_period_us
#  echo "1000000" > /cgroup/cpu/libvirt/qemu/RHEL61vm02/cpu.cfs_quota_us

# top
top - 16:30:36 up 19 days,  4:35,  5 users,  load average: 3.45, 3.57, 5.48
Tasks: 297 total,   6 running, 291 sleeping,   0 stopped,   0 zombie
Cpu(s): 65.5%us,  1.9%sy,  0.0%ni, 32.6%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   7937076k total,  6186888k used,  1750188k free,   381924k buffers
Swap:  2097144k total,     3260k used,  2093884k free,  2410048k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  780 qemu      20   0 1377m 1.0g 3696 R 192.7 13.2   9:51.11 qemu-kvm
  741 qemu      20   0 1376m 1.0g 3692 R 50.8 13.4  10:36.94 qemu-kvm
```

libcgroupによるcgroupsの管理

- libcgroupパッケージが提供するコマンド群でcgroupsを管理することもできます。
  - 次は、「cg*コマンド」による管理の例です。

```
# cgcreate -g cpuset:group00
# cgset -r cpuset.cpus=0 group00
# cgset -r cpuset.mems=0 group00

# cgcreate -g cpuset:group01
# cgset -r cpuset.cpus=1 group01
# cgset -r cpuset.mems=0 group01

# cgget -r cpuset.cpus -r cpuset.mems group00 group01
group00:
cpuset.cpus: 0
cpuset.mems: 0

group01:
cpuset.cpus: 1
cpuset.mems: 0

# cgexec -g cpuset:group00 ./loop.sh &
[1] 2930

# cgclassify -g cpuset:group01 2930
```

cpusetサブシステムにグループgroup00を作成して、パラメータを設定（cpuset.memsは必須）

cpusetサブシステムにグループgroup01を作成して、パラメータを設定（cpuset.memsは必須）

設定値を確認

グループを指定してコマンドを実行

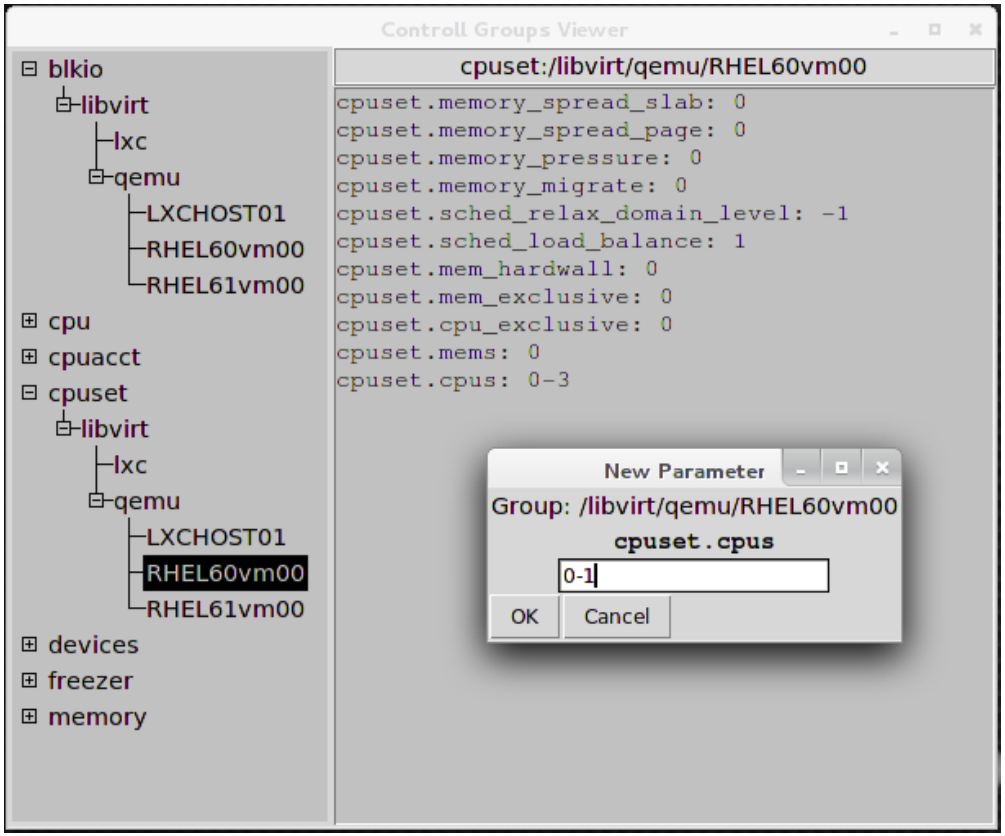
実行中のプロセスのグループを変更

  - cgredサービスを利用するとプロセスを起動したユーザ／グループによって、自動的にグループを割り当てることもできます。「/etc/cgroups.conf」に設定を記載します。

（参考）cgroups管理のGUIツールの例

- 下図は、Pythonで作成したcgroups管理ツールの例です。
  - ソースコードは下記を参照してください。

    http://d.hatena.ne.jp/enakai00/20110729/1311912971



メモとしてお使いください

メモとしてお使いください

メモとしてお使いください

参考資料: KVMのその他の機能

仮想CPUのモデル選択

- 物理CPUが持つ命令セットのフラグをマスクして、古い世代の互換CPUとしてゲストOSに見せる機能です。
  - 物理CPUが持っていないフラグが見えるようになることはありません。
  - デフォルト設定は、少し古い世代のCPUに相当します。
- 「Copy host CPU configuration」を押すと、物理CPUと同じモデルが選択されます。
  - 暗号化や画像処理などでCPU固有の機能を利用するアプリケーションでは、物理CPUと同じモデルにした方がパフォーマンスが向上する場合があります。

物理CPU（第2世代Core i5）

```
# cat /proc/cpuinfo | grep flags | head -1
flags       : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr
sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts rep_good xtopology nonstop_tsc
aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm sse4_1 sse4_2 x2apic
popcnt aes xsave avx lahf_lm ida arat epb xsaveopt pln pts tpr_shadow vnmi flexpriority ept vpid
```

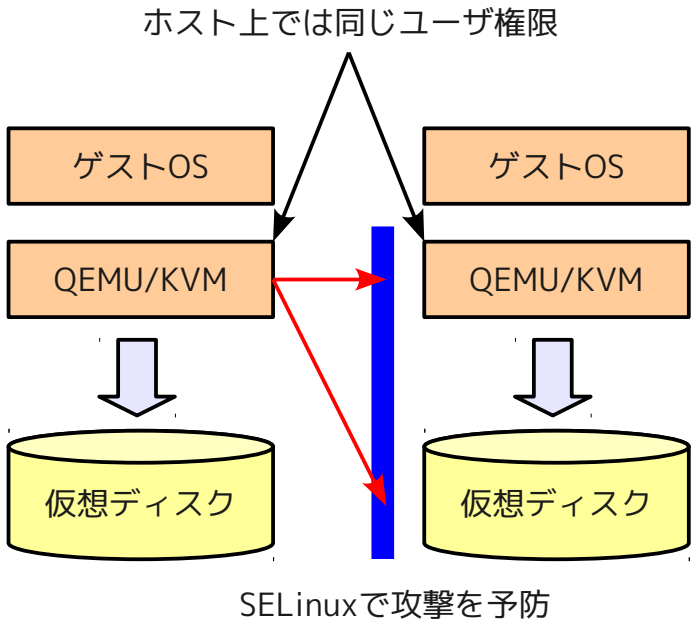
ゲストOS（デフォルト設定）

```
# cat /proc/cpuinfo | grep flags | head -1
flags       : fpu de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pse36 clflush mmx fxsr sse sse2 syscall
nx lm rep_good unfair_spinlock pni cx16 hypervisor lahf_lm
```

sVirtによるセキュリティ保護

- RHEL6では、SELinuxの「targetedポリシー」を適用すると、sVirtによる仮想マシン間のアクセス保護が行われます。
  - RHEL6ではデフォルトでtargetedポリシーが適用されます。特別な設定は不要です。

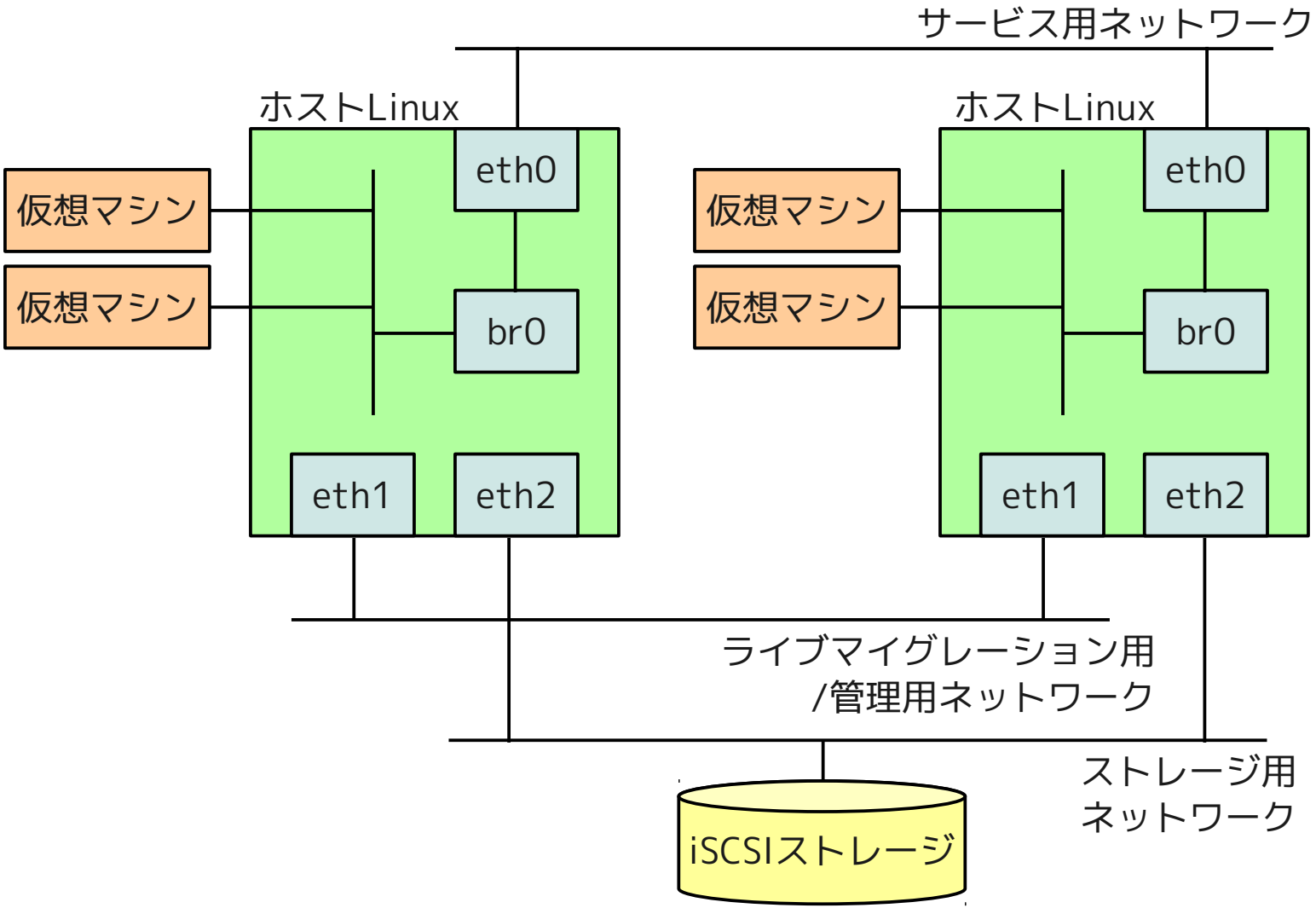
- QEMU/KVMの実装に未知のセキュリティ脆弱性があった場合、あるゲストOSから、他のゲストOSのファイルアクセスや他のゲストOSプロセスの操作を行われる可能性があります。sVirtでは、仮想マシンプロセスごとに異なる権限(MCSのカテゴリラベル）を設定することで、このような問題を防止します。



ライブマイグレーションに必要な構成

- ライブマイグレーションを行う場合は、共有ディスク装置を使用します。
  - NFSを共有マウントする方法が簡単ですが、ディスクI/O性能が必要な場合は、FC（ファイバーチャネル）接続やiSCSI接続による共有ディスク装置を使用します。
  - 共有ディスク装置上に仮想ディスクイメージファイルを配置する場合は、GFS2などの共有ファイルシステムが必要です。
  - 共有LUN上のLVM論理ボリュームを使用する場合は、CLVM（クラスタLVM）が必要です。
- マイグレーション時のネットワーク転送がサービス通信に影響を与えないために、サービスネットワークとは別のマイグレーション用ネットワークを用意することをお勧めします。
- マイグレーションするホストは同一モデルの物理CPUを使用します。
  - 異なる世代の物理CPUを使用する場合は、仮想CPUの機能フラグが一致するように、仮想CPUのタイプを調整します。
  - Intel CPUとAMD CPUの間のライブマイグレーションなどは失敗する可能性があります。

ライブマイグレーションの構成例



メモとしてお使いください

メモとしてお使いください

メモとしてお使いください

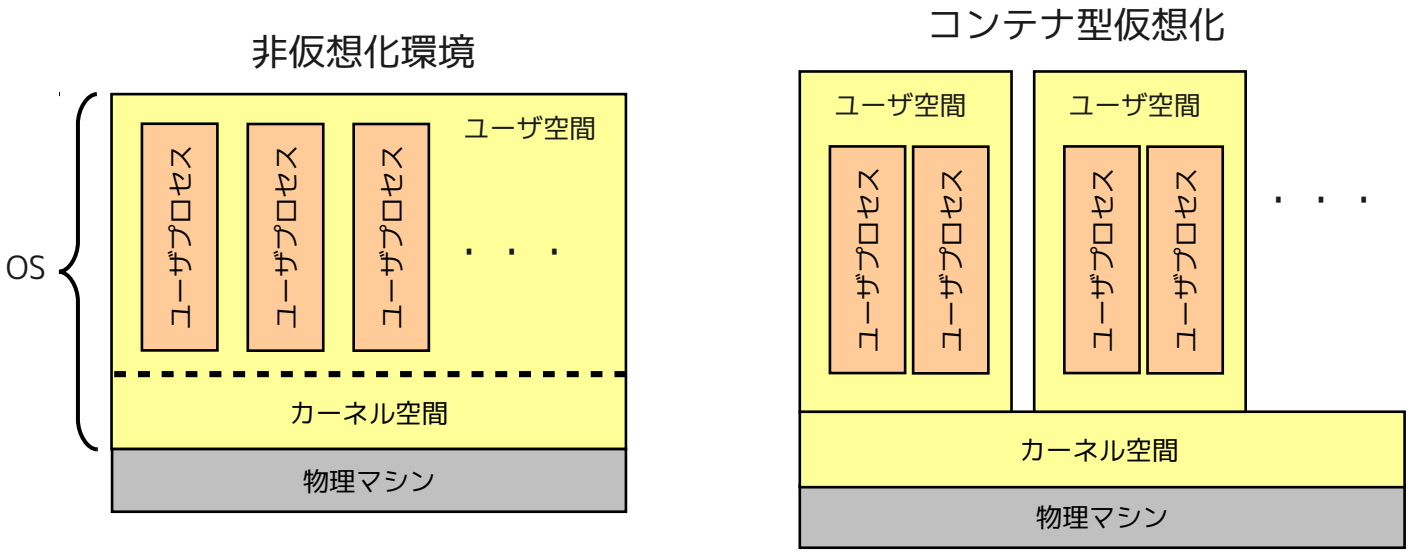
参考資料: LXC（Linuxコンテナ）

コンテナ型仮想化とは

- OSの内部は、物理リソースを管理するカーネル空間とユーザプロセス（アプリケーション）を実行するユーザ空間に分かれます。コンテナ型仮想化は、カーネルの機能で複数のユーザ空間を作り出すことにより、ユーザプロセスから見えるリソースを分割します。
  - プロセスリスト、ホストネーム、OSユーザ、ネットワーク、ファイルシステム、コンソールなど

    ⇒ コンテナごとに異なる内容が見えます。
  - CPU、メモリ、デバイス（/dev/*）

    ⇒ 物理マシン全体の情報が見えるが、cgroupsでコンテナごとに利用範囲を制限します。



RHEL6でのLXCの利用方法

- RHEL6では、デフォルトカーネルでLXCが利用可能です。
  - RHEL6.2では、Technology Previewのためレッドハットの正式サポート対象外です。

- コンテナを操作するツールは、sourceforge（http://lxc.sourceforge.net/）から「LXCツール」をダウンロードして使用します。RHEL6.2では、libvirt API（virt-manager/virshコマンド）でコンテナを操作することもできます。
  - 利用手順の例は下記を参照してください。
    - 「RHEL6.0 で LXC (Linux コンテナ）」http://d.hatena.ne.jp/enakai00/20110529/1306658627
    - 「RHEL6.2のlibvirtからLinuxコンテナを利用」http://d.hatena.ne.jp/enakai00/20120104/1325653622

LXCのリソース分割 (1) ～ プロセステーブル

- 各コンテナのプロセスは、同じLinuxカーネルで実行されます。
  - Linux以外のOSをコンテナで実行することはできません。
- プロセステーブルはコンテナごとに独立しているので、他のコンテナのプロセスは見えません。
  - コンテナ外部にあたるホストLinux上では全てのプロセスが見えます。
  - ホストLinux上のプロセステーブルでは、コンテナに属するプロセスには、コンテナ名ラベルが付与されます。

ホストLinuxのプロセステーブル　（コンテナ内部ではそのコンテナに属するプロセスしか見えない）

```
# lxc-ps -ef
CONTAINER  UID        PID  PPID  C STIME TTY          TIME CMD
           root         1     0  0 Jun03 ?        00:00:00 /sbin/init
           root         2     0  0 Jun03 ?        00:00:00 [kthreadd]
...
           root     10258     1  0 10:30 ?        00:00:00 lxc-start -n 005 -d
005        root     10266 10258  0 10:30 ?        00:00:00 /sbin/init
           root     10268     1  0 10:30 ?        00:00:00 lxc-start -n 006 -d
006        root     10278 10268  0 10:30 ?        00:00:00 /sbin/init
006        root     10801 10278  0 10:30 ?        00:00:00 /sbin/rsyslogd -c 4
...
005        root     10989 10266  0 10:30 ?        00:00:00 crond
005        root     11027 10266  0 10:30 pts/4    00:00:00 /sbin/agetty /dev/console 38400 vt100-nav
...
           root     14452 12337  0 14:20 pts/3    00:00:00 lxc-start -n 003
003        root     14455 14452  0 14:20 ?        00:00:00 /sbin/init
003        root     14626 14455  0 14:20 ?        00:00:00 /sbin/rsyslogd -c 4
```

LXCのリソース分割 (2) ～ ファイルシステム

- コンテナごとに特定ディレクトリにchrootします。
  - コンテナごとのルートファイルシステムを任意のディレクトリにマウントしておきます。
- コンテナで特定デーモンだけを動かす場合
  - デーモンの稼働に必要なファイルだけを用意します。
- コンテナでゲストOSをまるごと動かす場合
  - ゲストOSのルートファイルシステムをまるごとマウントします。
  - KVMのゲストOSイメージファイルをループマウントして利用するなどの方法も可能です。

```
# ls -l /lxc/rootfs/
合計 28
drwxr-xr-x   2 root root 4096  5月 28 13:53 2011 001
drwxr-xr-x   2 root root 4096  5月 28 15:13 2011 002
dr-xr-xr-x. 23 root root 4096  5月 30 12:41 2011 003
dr-xr-xr-x. 24 root root 4096  5月 31 13:13 2011 004
dr-xr-xr-x. 24 root root 4096  5月 31 13:13 2011 005
dr-xr-xr-x. 24 root root 4096  5月 31 13:13 2011 006
dr-xr-xr-x. 23 root root 4096  5月 30 12:41 2011 007

# ls -l /lxc/rootfs/003
合計 104
dr-xr-xr-x.  2 root root  4096  5月 25 20:24 2011 bin
drwxr-xr-x.  2 root root  4096  5月 25 20:22 2011 boot
drwxr-xr-x.  2 root root  4096  7月 14 20:45 2010 cgroup
drwxr-xr-x.  2 root root  4096  5月 25 20:22 2011 dev
drwxr-xr-x. 57 root root  4096  6月  4 14:20 2011 etc
drwxr-xr-x.  3 root root  4096  5月 29 05:35 2011 home
dr-xr-xr-x.  8 root root  4096  5月 25 20:24 2011 lib
dr-xr-xr-x.  7 root root 12288  5月 25 20:24 2011 lib64
drwx------. 12 root root 16384  5月 25 20:22 2011 lost+found
drwxr-xr-x.  2 root root  4096 12月  4 22:33 2009 media
drwxr-xr-x.  2 root root  4096 12月  4 22:33 2009 mnt
drwxr-xr-x.  5 root root  4096  5月 31 11:48 2011 opt
drwxr-xr-x.  2 root root  4096  5月 25 20:22 2011 proc
dr-xr-x---.  5 root root  4096  5月 31 13:01 2011 root
dr-xr-xr-x.  2 root root  4096  5月 25 20:25 2011 sbin
drwxr-xr-x.  2 root root  4096  5月 25 20:23 2011 selinux
drwxr-xr-x.  2 root root  4096 12月  4 22:33 2009 srv
drwxr-xr-x.  2 root root  4096  5月 25 20:22 2011 sys
drwxrwxrwt.  2 root root  4096  6月  4 14:18 2011 tmp
drwxr-xr-x. 14 root root  4096  5月 29 05:33 2011 usr
drwxr-xr-x. 17 root root  4096  5月 25 20:23 2011 var
```

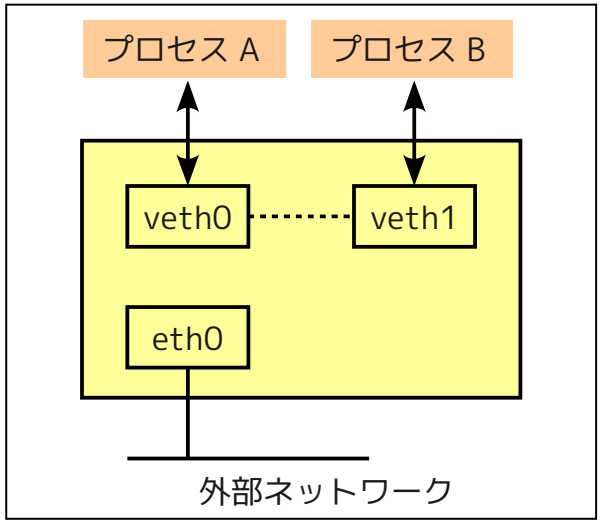
コンテナごとのルートファイルシステムをマウント

LXCのリソース分割 (3) ～ ネットワーク

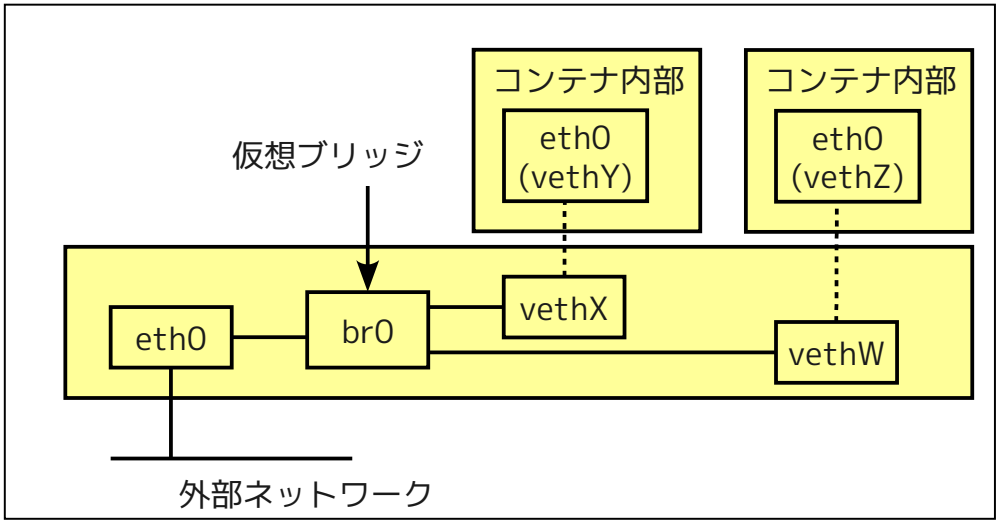
- 各コンテナの仮想NIC（veth）をホストOSの物理NICにブリッジ接続します。
  - vethはLinuxカーネルが標準で提供する機能です。2つのvethのペアを（クロスケーブルで直結したかのように）接続して、プロセス間の擬似的なネットワーク通信を実現します。

- LXCではvethペアの一方をコンテナ内部に見せて、もう一方を仮想ブリッジに接続することで、コンテナ内部と外部ネットワークを接続します。
  - コンテナ内部ではデバイス名をeth0に書き換えています。
  - KVMの仮想ネットワークと似ていますが、KVMではTAPデバイスを使用する点が異なります。

vethの基本的なイメージ



LXCでの実際のvethの使い方



LXC のリソース分割 (4) ～ CPU/メモリ/デバイス

- CPU/メモリ/デバイスは、各コンテナから物理サーバ全体のリソースが見えます。
  - Control Groups (cgroups)を利用して、各コンテナで実際に利用できるCPU/メモリ/デバイスを設定します。

- 次のような設定が可能です。
  - コンテナが使用するCPUコア（cpuset.cpus）

    ⇒ CPUコアの番号を指定。
  - CPUの優先順位 （cpu.shares）

    ⇒ 2～262144を指定。値に比例した実行時間が与えられる。
  - メモリ使用量の上限 （memory. limit_in_bytes, memory. memsw.limit_in_bytes）

    ⇒ 物理メモリ、および、物理メモリ+Swapの上限を指定。
  - I/Oの優先順位 （blkio.weight）

    ⇒ 100～1000 を指定。値に比例したI/O処理時間が与えられる。
  - アクセスを許可するデバイス （devices.deny, devices.allow）

    ⇒ Major, Minor番号でアクセスを許可するデバイスを指定。

メモとしてお使いください