
Virtualization Tools: XEN

“The book of XEN:
A Practical Guide For System Administrator”
C. Takemura, Luke S. Crawford



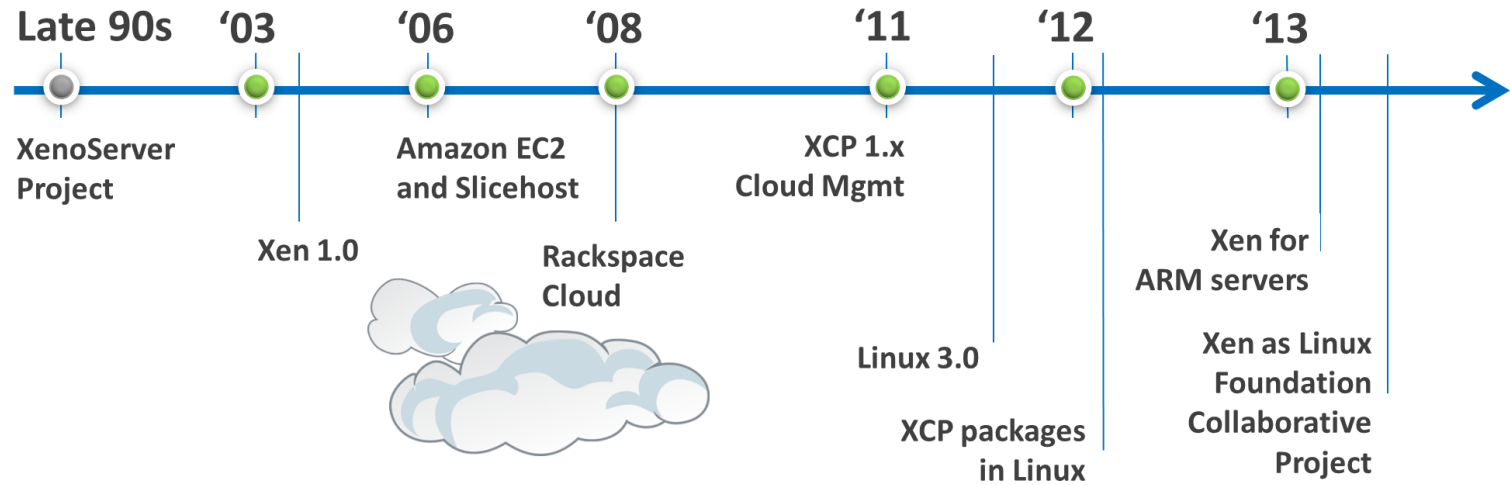
Outline

- Introduction
- High-level overview
- Getting Started
- Provisioning Domus
- Resource Management

Introduction

- **Xen** is a System Level VMM
- Originally developed in the University of Cambridge
 - Initially released year 2003
- Currently multiple “flavors”
 - XenServer, XenApp/XenDesktop: owned by Citrix and targets support and certified appliances for enterprise environments. Both server and Desktop virtualization (VDI and Applications)
 - XenServer core / Xen-source: open source GPLv3. Only Server virtualizations
- Currently available for IA-32, x86_64(amd64) and ARM architectures

Xen history



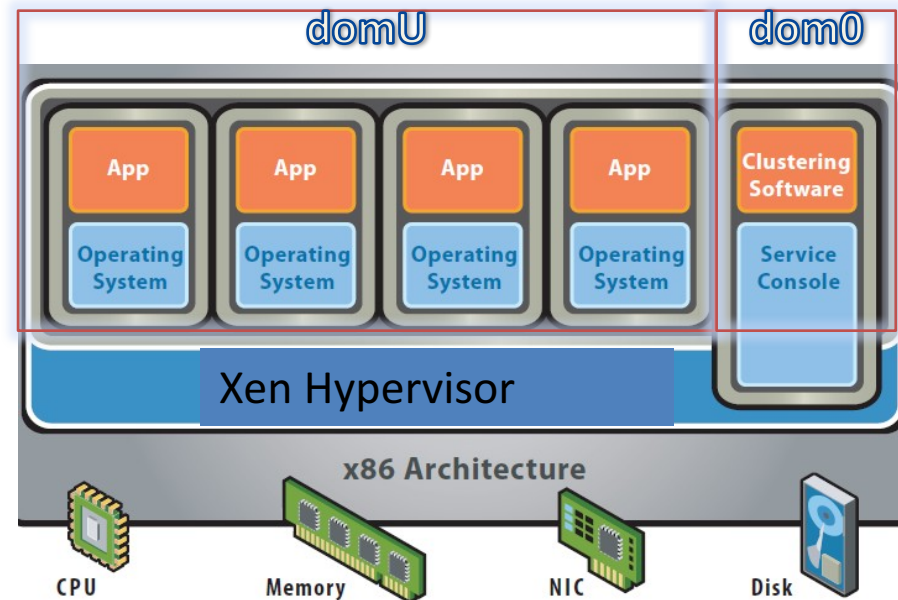
- 1990s-2002: Developed as a research project in Cambridge University(**VMM**)
 - Born as a part of **XenoServer project**, leeded by Ian Pratt and Keir Fraser
- 2003: Public Release(v1.0)
 - The Art of Virtualization → USENIX Operating Systems Design and Implementation conference
 - (<http://www.cl.cam.ac.uk/research/srg/netos/papers/2003-xensosp.pdf>)
- 2004: **XenSource** Ltd. is founded by Ian Pratt from 2.0.5
 - Provide bussines support for Xen

Xen history

- 2005: Red Hat, Novell, and Sun embrace Xen as virtualization layer
 - Creates a large community of developers
- 2006: XenSource releases Xen 3.0
 - Support for acceleration from Intel/AMD processors (Vanderpool/Pacifica)
- 2007: **Citrix** buys XenSource (\$ 500M)
 - Comercial flavor → Citrix XenServer (6.2)
 - Open Source flavor → **Xen.org** XenServer (4.4)
- 2008: Xen.org (Xenproject.org) support for kernel 2.6.18, Solaris 10, FreeBSD ...
- 2010: Xenproject.org releases Xen 4.0
- 2011: Xen Integration
 - Kernel 2.6.37 → 1st xenderized stock kernel
 - Kernel 3.x.xx → Full xen support

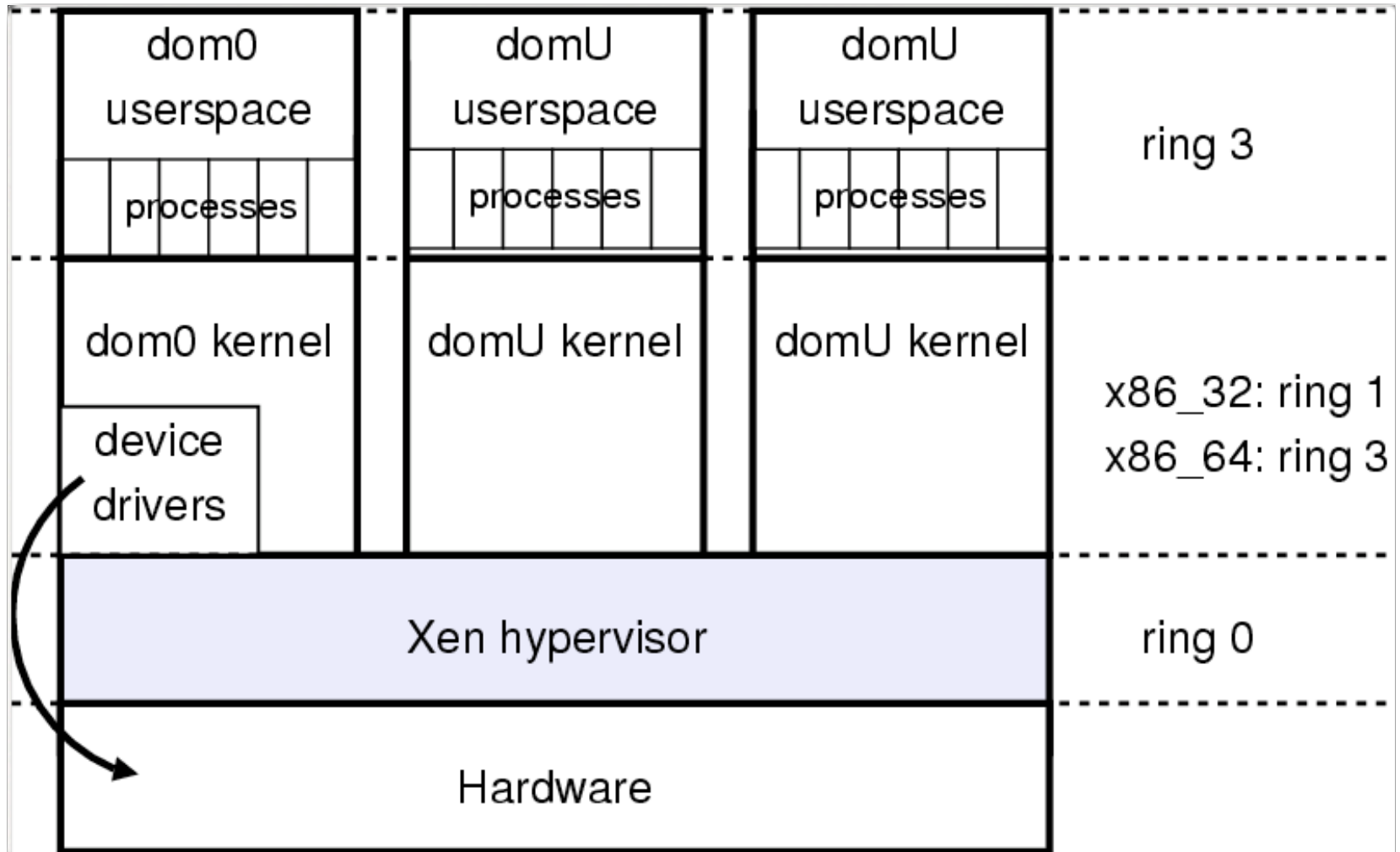
Architecture

- Xen is a native/type-1/bare-metal sVMM
 - Structured in “domains”
- domUs
 - Where the guest OS actually runs
 - Unprivileged access
 - Most OS: Linux, Win, Solaris BSD,
- dom0
 - Management OS
 - Privileged access
 - Linux, Solaris, BSD



Bare-Metal (Hypervisor) Architecture

Architecture

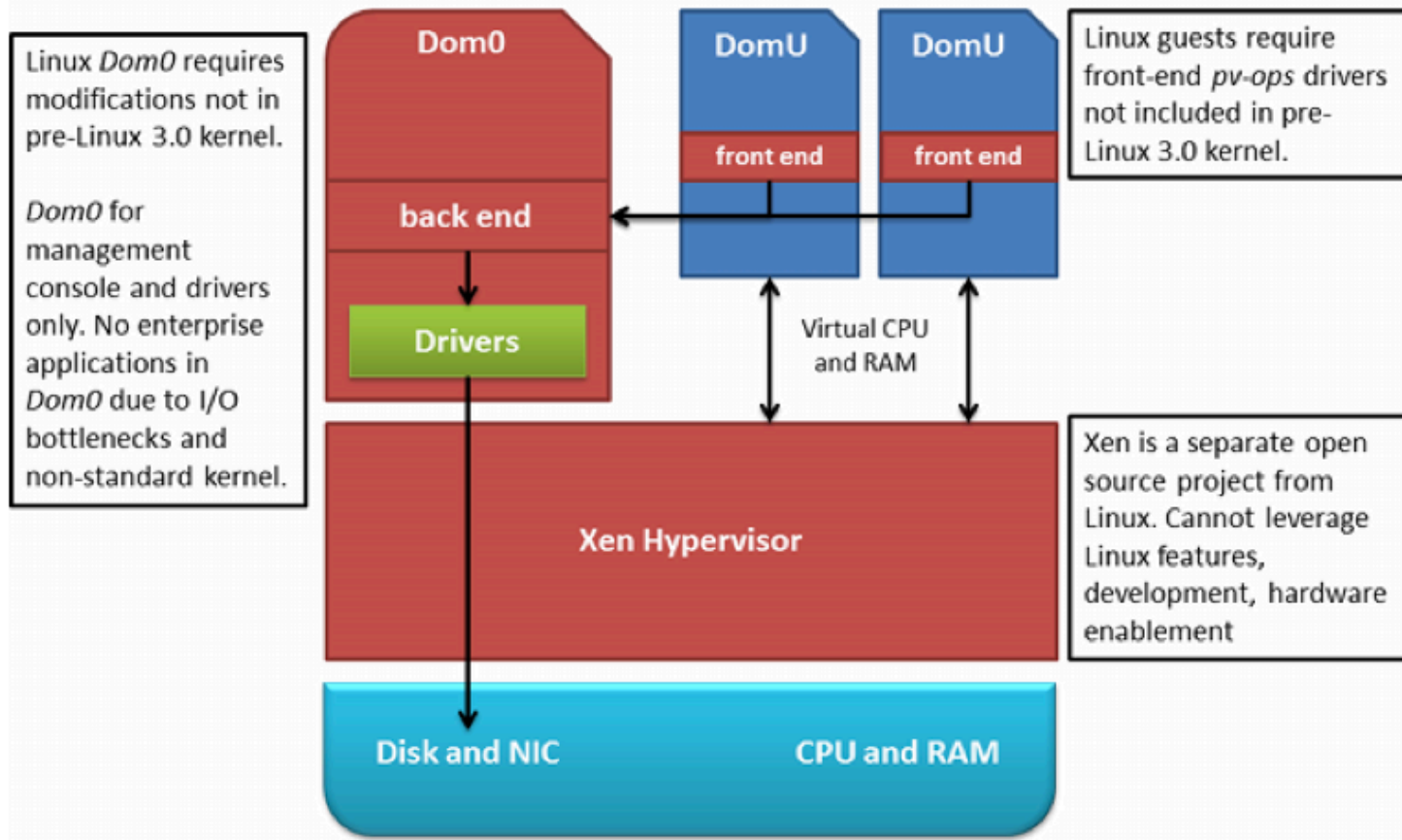


Xen & Linux

- As of 2009, most Linux distributions had included Xen packages to interact with the Xen hypervisor and start additional domains, but because Xen was not accepted into the mainline Linux kernel and installation required several kernel patches, some distros such as Red Hat Enterprise Linux 6 and Ubuntu 8.10 dropped out-of-the-box support for dom0 in subsequent releases
 - Kernel xenderization: a painful and time consuming task
- With the inclusion of the most significant parts of Xen in the Linux 2.6.37 mainline kernel in early 2011, several distributions are again considering dom0 support.
- **Version 3.0** of the Linux kernel supports **dom0** and **domU** in the mainline kernel
 - Everything is ready out-of-the-box: yikes!
- Kernel developers are pushing KVM virtualization
 - Big interest (and money) from IBM and other players: IBM knows a lot about VM!
 - In the medium-term might be predominant architecture?!?!?

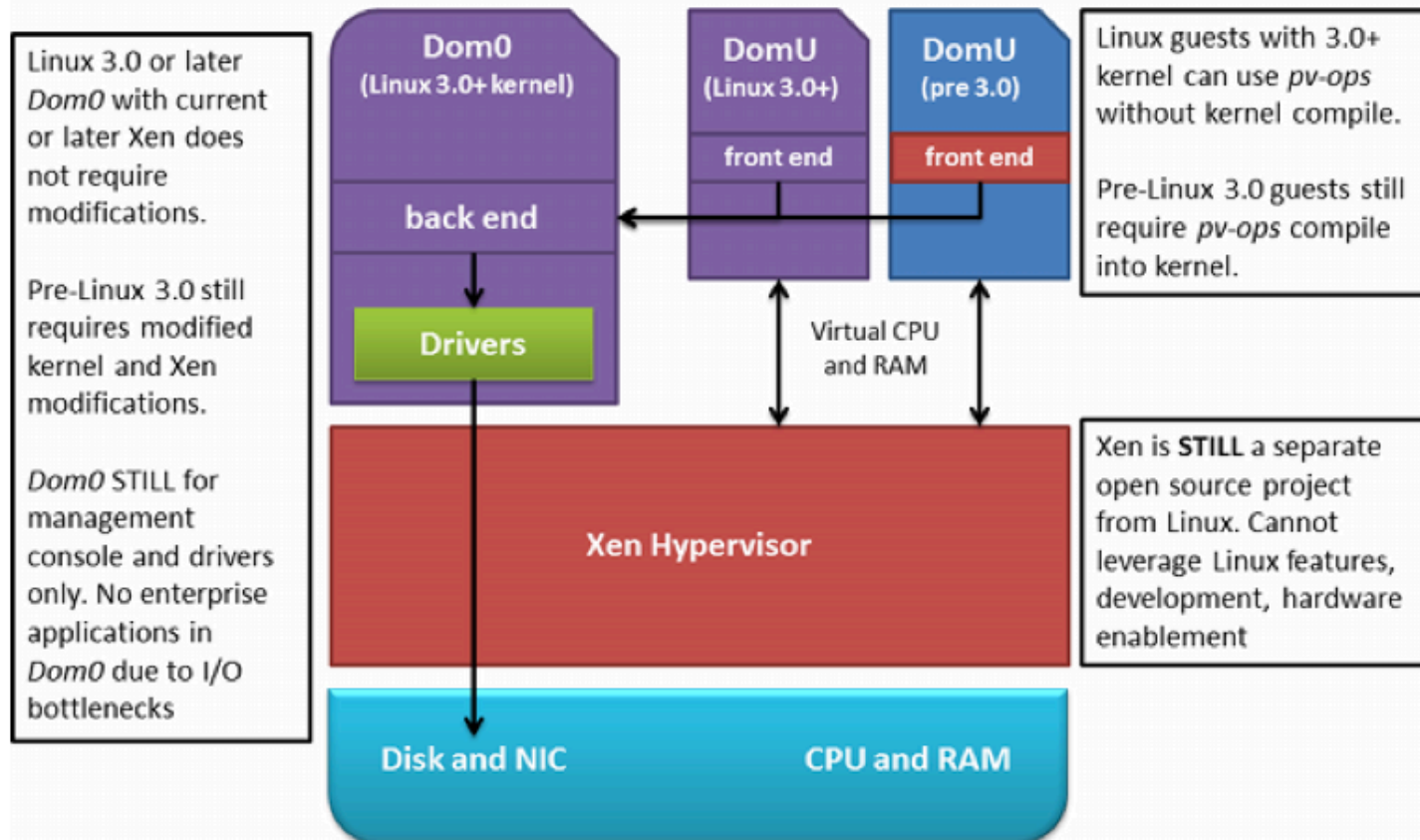
Xen with Kernel 2.6.xx (xx<37)

Xen Before Linux 3.0



Xen with Kernel 3.xx

Xen After Linux 3.0

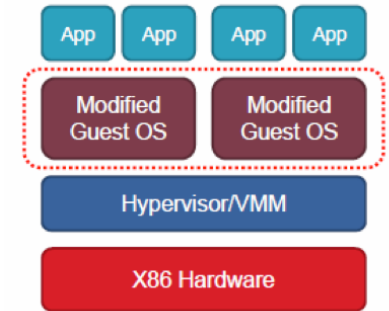


High Level Overview

- Techniques supported
 - Full virtualization (called HVM)
 - Unmodified OS kernel guests
 - OS interventions are emulated through QEMU
 - Para-virtualization (called PV)
 - Adapted OS kernel guests: replace system calls with hyper calls to the hypervisor
 - No binary translation required on x86 32 bits
 - Hybrid techniques are usual
 - And the most efficient in state-of-the art OSes and hardware
 - Can be mixed in the same server
 - Some domU using PV (v.gr. linux) and some domU using HVM (v.gr. windows)
- Initially binary translation is much slower than native execution
 - Xen focused his attention to avoid that through PV
 - With today's hardware, might be better HVM (most binary translation, memory handling overheads are avoided through hardware acceleration in processor)
 - I/O (in many environment) is still better with PV

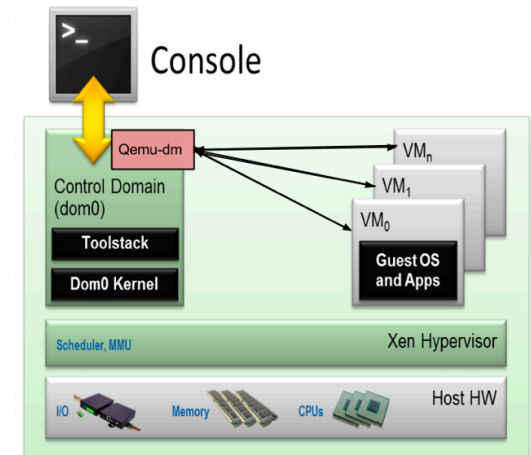
Para-virtualization (PV)

- Was the distinctive characteristic of Xen (in contrast with others)
 - Is the default use mode
 - Don't need any hardware support
 - Emulation/binary translation is NOT required
 - The guest kernel **should** be modified
 - Redirect kernel to the hypervisor through an API
 - Replace system calls with hyper-calls (~50) to the hypervisor
 - Kernel xenderization is the process to redirect system calls (~300) to those hyper-calls
 - The ABI that perceives user level application is the same! No user code recompilation ! Just Guest kernel!
 - The VMM acts like a OS (that handles other Oses): hyper-calls are non-blocking, asynchronous, ...
 - No closed source guestOS support (no windows)
- Performance overheads might be reasonable
 - ~10%
- Hyper-V (Microsoft) also uses this approach



Hardware virtualization Mode (HVM)

- Unmodified OS as guests
 - Requires virtualization extensions (VT-x)
 - **No VMWare “alike” dynamic binary translation**
- Uses a emulator for I/O guest devices
 - Half-way between VM and a Simulator
 - Take insn by insn an emulate its behavior
 - **Each original insn is converted in tens of instructions**, much slower than “native” insn
- QEMU (Quick EMUlator)
 - Separate opensource project (older than Xen) but with strong relation (And with KVM)
 - It has support to emulate a full server (similar to Simics)
 - QEMU provided fully virtualized drivers for HVM
- Virtualization acceleration avoids most QEMU interventions
 - Performance is competitive with PV (almost the same overhead)
 - Memory management is faster in HVM than PV (no shared page tables software management is required)
 - Still paravirtualized driver access is more efficient



Hybrid Mode 1 (PV-on-HVM)

- Replace instruction and Memory management from hypervisor
 - Relies in hardware support (gen3 virt)
- Some device management might be still PV
 - Require xenderized guest kernel
- Still some “pieces” QEMU emulated
 - Interrupts (including timers)
 - Boot
 - Locks

Hybrid mode 2 (PVH)

- No QEMU emulation (at all)
 - Either hardware assisted or Para-virtualized
- Recent support
 - Starts with 4.4
- More info about PVHVM and PV
 - We will revisit later

PV & HVM: Spectrum & Focus

	HVM guest	Classic PV on HVM	Enhanced PV on HVM	PV guest	PVH
Boot sequence	emulated	emulated	emulated	paravirtualized	paravirtualized
Memory	hardware	hardware	hardware	paravirtualized	hardware
Interrupts	emulated	emulated	paravirtualized	paravirtualized	paravirtualized
Timers	emulated	emulated	paravirtualized	paravirtualized	paravirtualized
Spinlocks	emulated	emulated	paravirtualized	paravirtualized	paravirtualized
Disk	emulated	paravirtualized	paravirtualized	paravirtualized	paravirtualized
Network	emulated	paravirtualized	paravirtualized	paravirtualized	paravirtualized
Privileged operations	hardware	hardware	hardware	paravirtualized	hardware

Installing and Booting dom0

- Starting with a conventional OS x86-64 (Debian 7 in our case)
 - Virtualization extensions enabled in BIOS/UEFI `cat /proc/cpuinfo | grep vmx`
 - Compile from source or repository download
 - No changes in stock kernel
- To make it available at GRUB2 , 20_linux_xen in /etc/grub.d
 - In /boot/grub/grub.cfg something like (... is aut. gen. by “update-grub”):

```
submenu "Xen 4.1-amd64" {
menuentry 'Debian GNU/Linux, with Xen 4.1-amd64 and Linux 3.2.0-4-amd64'
--class debian --class gnu-linux --class gnu --class os --class xen {
    insmod part_msdos
    insmod ext2
    set root='(/dev/sda,msdos1)'
    search --no-floppy --fs-uuid --set=root 9d6458e0-5075-456d-8dea-03b82dc2b827
    echo    'Loading Xen 4.1-amd64 ...'
    multiboot /boot/xen-4.1-amd64.gz placeholder allow_unsafe=true
    echo    'Loading Linux 3.2.0-4-amd64 ...'
    module /boot/vmlinuz-3.2.0-4-amd64 root=UUID=BEEF-...-...- ro quiet
    echo    'Loading initial ramdisk ...'
    module /boot/initrd.img-3.2.0-4-amd64
...}
```

Dom0 Access

- After reboot, inherits all the “characteristics” of the base system and the administrative task are performed accordingly
 - Software
 - User configuration
 - Devices
 - ...
- Dom0 kernel runs on top of Xen hypervisor
 - Pre 3.0 kernel might require patching (and recompilation)
 - Post 3.0 kernels has already the Xen hyper-calls
 - Note that vanilla Linux boot is still available
- How to tell if are we virtualized or not?
 - `dmesg | grep Xen`
 - `xm dmesg`
- We should perform all the administrative task from dom0
 - V.gr. Install domU
 - Has all the services, privileges and tools required to perform such task
 - Good rule of thumb is use a minimal system

Dom0 Services and tools

- To handle domU the basic tool is **xm**
 - Uses “xen” service to interact with the hypervisor

```
root@aos:~# xm list
```

Name	ID	Mem	VCPUs	State	Time (s)
Domain-0	0	1895	2	r-----	85.4

- Allows to start, suspend, checkpoint and migrate any domU
- Xend also configs dom0
 - Dom0 processor & memory, networking, remote administration, storage, devices,...
 - Configured from `/etc/xen/xend-config.sxp`
 - Changes will require restart the service (not reboot!)

Create a new domU

■ Multiple approaches

- Manually

- dd, mount, cdbootstrap (debian), rinse(CentOS), etc...

- Red-hat virtinst: virt-install

- Powerful but complex (enterprise oriented)

- Xen-tools

- Easy and most suitable if environment is semi-“closed”
- Uses /etc/xen-tools/xen-tools.conf template
 - Defines memory, processor, networking, storage, etc...
 - Automatically install a Debian 7 in a disk image

```
xen-create-image --hostname=prueba --ip 172.16.30.130 --netmask  
255.255.255.0 --gateway 172.16.30.129 --dir /xen/ --force
```

domU booting

- This will create the domU config file in /etc/xen/prueba.cfg

```
kernel      = '/boot/vmlinuz-3.2.0-4-amd64'
ramdisk     = '/boot/initrd.img-3.2.0-4-amd64'
vcpus       = '1'
memory      = '128'
root        = '/dev/xvda2 ro'
disk        = [
                'file:/xen//domains/prueba/disk.img,xvda2,w',
                'file:/xen//domains/prueba/swap.img,xvda1,w',
            ]
name        = 'prueba'
vif         = [ 'ip=172.16.30.130 ,mac=00:16:3E:6C:82:F6' ]

#on_poweroff = 'destroy'
on_reboot   = 'restart'
on_crash    = 'restart'
```

- Configure “xen” networking as bridge (simplest approach)
 - (network-script network-bridge)
- We can boot the domU
 - `xm create prueba.cfg`
- Once booted is listed in the dom0

```
root@aos:~# xm list
```

Name	ID	Mem	VCPUs	State	Time (s)
Domain-0	0	1894	2	r-----	501.9
prueba	3	128	1	-b----	4.3

Access the domU (U=3)

- Xen console

```
root@aos:~# xm console 3
```

- Alternatively we can boot and connect

```
root@aos:~# xm create -c prueba.cfg
```

- To exit console “CTR+5”

- Yes... the root password is in /var/log/xen-tools/prueba.log :)
- Change eth0 to dhcp

- Dom0 operations

- Suspend: `xm suspend 3`
- Power off: `xm shutdown 3` (`xm destroy eq. hard-reset!`)
- Most operations requires a “managed domain”

Managed DomU

- Previous example is unmanaged (volatile domain)
- Add a machine configuration to “xenstore” (persistent domain)

- Xenstore can keep much more things than domU
- Requires python-lxml

```
root@aos:~# xm new prueba.cfg
root@aos:~# xm list
```

Name	ID	Mem	VCPUs	State	Time (s)
Domain-0	0	1894	2	r-----	520.1
prueba		128	1		0.0

- Full access to the domain

```
root@aos:~# xm start prueba
root@aos:~# xm suspend prueba
root@aos:~# xm resume prueba
```

...

```
root@aos:~# xm migrate prueba <other host>
```

- Now changes in “prueba.cfg” will be ignored (delete/new or xm)
- To inspect managed domain parameters

```
root@aos:~# xm list prueba -l > prueba.py
```

- To change parameters in production use “xm” but here use delete/edit config/new

DomU Provisioning

- It's time to get our hands dirty
 - Up to now, simple approach (PV + automated tools)
 - In practice, not always feasible
 - Guest constraints (v.gr., custom kernel?, Windows?)
 - Performance optimizations (v.gr. HVM?)
 - ...scripts are not bullet proof
- Three main parts
 - Prepare installation
 - Monolithic image
 - LV (LVM)
 - Tune virtual machine resources
 - CPU, memory, network, admin connections
 - Adjust booting
 - Kernel selection

Installation

- LVM is the weapon of choice in a production environment (networked block level device: FC, iSCSI, etc...)
 - We will do it later
- By now just monolithic images
 - Similar to the desktop virtualization solutions
 - Via CD
 - booting the VM over the CD or using QUEMU)
 - Bootstrap

```
dd if=/dev/zero of=manual.img bs=1G count=4
mkfs.ext4 manual.img
mount manual.img /mnt/
debootstrap --arch amd64 wheezy /mnt/ http://ftp.es.debian.org/debian/
rinse --distribution centos-6 --arch amd64 --directory /mnt/
#debootstrap for debian and ubuntu / rinse for centos/fedora/opensuse
#perhaps other steps are required (v.gr. fstab, modules, etc...)
```

Adjust the installation


- Normally debootstrap installs a minimal set of tools to boot the system
- Lets assume that we want apache and our dom0 kernel can handle the distribution used in the domU
- Chroot (poor's man "VM")

```
cp /etc/resolv.conf /mnt/etc/  
mount -t proc proc /mnt/proc/  
mount --rbind /sys /mnt/sys/  
mount --rbind /dev /mnt/dev/  
chroot /mnt/  
apt-get update  
apt-get install apache2  
exit; umount in reverse order,...
```

Prepare the VM config file

```
kernel      = '/boot/<EXISTING KERNEL IN dom0>'
ramdisk     = '/boot/<EXISTING INITRD IN dom0>'
vcpus       = '<DON'T OVERCOMMIT>?'
memory      = '<DON'T OVERCOMMIT>?'
root        = '/dev/<domU_diskdevice> ro'
extra       = '<Any parameter for domU kernel>'
disk        = ['file:/xen//<path_to_img/lv_dev>'
file>,<domU_diskdevice>,w', ...]
name        = 'my_name'
vif         = [ 'ip=<IP_MAN> ,mac=<DIFF>' ]

on_poweroff = 'destroy' #Desired behavior if Dom0
on_reboot   = 'restart'  #
on_crash    = 'restart'  #
```



Initial steps

- Usually there are problems at beginning
 - Format, console, etc...
 - Don't manage the domain until everything is ok (use `xm create` not `xm new`)
 - ~~• Alternatively use `xm create` and do the modification via py file (`xm list <name> -l`)~~
 - ~~• Annoying for manual mode~~
 - ~~• Interesting for scripting~~
- System last “touches”
 - Init will not start a hypervisor compatible tty
 - Not ready with debootstrap
 - Fix it adding to `inittab` in `domU`
 - `1:2345:respawn:/sbin/getty 38400 hvc0`
 - Root account is disabled
 - Beware with concurrent modifications in `img` file and `vm` (disk buffers!)
 - **NEVER BOOT TWO domU OVER THE SAME IMG!!!!**
 - Enable network
 - `echo "iface eth0 inet dhcp" >> /etc/network/interfaces`
 - ...

Boot

- Xen by default uses “pyGrub”
 - Limited to kernels available in dom0
 - Amazon EC2 uses this approach
 - Zillion of kernels and configurations available
- We can give the kernel control to the domU: use pvgrub (chained bootloaders)
 - Allows to define the kernel, and boot parameters inside the domU system disks
 - Not available in Debian xen4.1 pkg
 - Compile Xen from source code
 - Download it from:
 - www.atc.unican.es/%7evpuente/SVS/pv-grub-x86_64.gz

PVGRUB

- DomU config file should replace kernel by pc-grub-file

```
kernel= '/xen/pv-grub-x86_64.gz'      #IN dom0
extra= ' (hd0)/boot/grub/pvgrub.cfg'  #IN domU
#ramdisk = ###Comment (not needed)
#root /dev/xvda1    ###Comment (not needed)
...
```

- Easiest road to update the pvgrub.cfg
 - Prepare the image booting with pygrub
 - Install the desired kernel
 - (v.gr. apt-get install linux-image-3.4-amd64 from backports)
 - Pvgrub works with “GRUB1”. Configure it manually

```
#!/boot/grub/pvgrub.cfg on domU
title "custom kernel 2.6"
    root (hd0)
    kernel /boot/vmlinuz-3.4 root=/dev/xvda1 quiet ro
    intird /boot/initrd-3.4
title "custom kernel 3.2"
    root (hd0)
    kernel /boot/vmlinuz-3.2 root=/dev/xvda1 quiet ro
    intird /boot/initrd-3.2
```

Storage in Xen

- File based (“file:/path_to_img/,domU_device,w”)
 - The basic approach
 - Partition per file or portioned single file

```
dd if=/dev/zero of=/xen/parted.img bs=1M count=2M
xm block-attach 0 file:/xen/parted.img /dev/xvda1 w 0
kpartx /dev/xvda1
xm block-detach 0 /dev/xvda1
```
 - Xen allows to attach a detach dynamically storage devices to any domX via “block-attach”, “block-detach”, “block-list”,
- Physical device based (“phy:/dev/device_file, domU_device, w”)
 - Better performance
 - To achieve flexibility (and performance too), use LVM
 - Create a VG with separate LV per domU
 - To achieve reliability add RAID bellow VG
- Networked block devices (“iscsi:...”, “npiv:...”)
- Tap driver (user space mapped files)
- Live migration and storage
 - Live migration requires “networked” storage
 - From NAS (v.gr. NFS) to SAN (v.gr FC)

Disk QoS

- Multiple domU from different clients might be sharing a server (a.k.a Virtual Private Servers or VPS)
 - Shared resource usage has to be fair and Storage is one of the most important (QoS is defined by the Service Level Agreement or SLA)
 - If the provider don't keep SLA, client might ask for a "refund"
- Capacity "fair use" is straightforward
- Bandwidth isn't direct
 - A VPS doing a backup can interfere with other VPS in the same system (multi-tenancy)
 - Limit I/O using standard Linux Tools
 - `ionice -p <PID> -c 2 -n <prio>`
 - Similar to "nice" but for I/O system. Actually tampers the disk scheduling queue
 - » Requires CFQ scheduler (`cat /sys/block/[sh]d[a-z]*/queue/scheduler | grep cfq`)
 - Prio varies from 0 (higher prio) to 7 (lower priority)
 - Identify the process accessing the device/image file and tune accordingly

```
root@aos:/etc/xen# ps -ef| grep xvda
root      5432      2  0 02:41 ?           00:00:00 [blkback.5.xvda1]
root      5827      2  0 02:56 ?           00:00:00 [blkback.6.xvda2]
root      5828      2  0 02:56 ?           00:00:00 [blkback.6.xvda1]
root@aos:/etc/xen# ionice -p 5432 -c 2 -n 0; ionice -p 5827 -c 2 -n 7
```


A better approach for Storage

- Use LVM (preferably on top of RAID)
 - A logical volume has better performance
 - Supports natively snapshots
 - Multiple machine can share a common base (linked clones)
 - Can be handled like the img (via dd)
- Example of utility: snapshots
 - (1 step) Transferring a raw “img” to a logical volume

```
pvcreate /dev/sdb /dev/sdc
vgcreate vol1 /dev/sdb /dev/sdc
lvcreate -L4Gi -n base vol1
dd if=/xen/domains/manual/manual.img of=/dev/vol1/base
```
 - (2 step) Create a snapshot LVM per VM (500Mi available)

```
lvcreate -L500Mi -s -n vmdisk1 /dev/vol1/base #FOR DOMU 1
lvcreate -L500Mi -s -n vmdisk2 /dev/vol1/base #FOR DOMU 2
vgdisplay (4GB for base and 1GB for the base) /lvdisplay vmdisk1
dd if=/dev/vol1/vmdisk1 | gzip >vmdisk1.img.gz
dd if=/dev/vol1/vmdisk1 | gzip | ssh anotherhost gzip -dc | dd of=/file/remote/machine
```
- **Caveat: if a snapshot it is overflowed, it will be dropped!**
 - Becomes corrupt!
 - Best policy is to assign the same capacity to all volumes
- There are better solutions with better performance/flexibility
 - ZFS and btrfs

Networking in Xen

■ Easiest approach: Bridged Networking

- Requires domU to have real IP numbers

- xend devices

- Creates a hub (virtual interface) to connect the domU

```
# in xend config
# dom0: ----- bridge -> real eth0 -> the network
#                               |
# domU: fake eth0 -> vifN.0 -+
#
# use (also /etc/xen/scripts/network-bridge {start|stop})
#
(network-script network-bridge)
```

- Ifconfig list 2 devices

- peth0: It's the actual real network
 - eth0: is the virtual hub in dom0 (it is a bridge)

- domU connectors

- Ifconfig list the domU devices

- vifxx.yy (xx is bridge id)

- In domU appears like “ethyy,...”

How to inspect the bridge?

```
root@aos:~# /etc/xen/scripts/network-bridge status
```

```
=====
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 00:0c:29:86:5b:1f brd ff:ff:ff:ff:ff:ff
    inet 172.16.30.130/24 brd 172.16.30.255 scope global eth0
    inet6 fe80::20c:29ff:fe86:5b1f/64 scope link
        valid_lft forever preferred_lft forever
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 00:0c:29:86:5b:1f brd ff:ff:ff:ff:ff:ff
    inet 172.16.30.130/24 brd 172.16.30.255 scope global eth0
    inet6 fe80::20c:29ff:fe86:5b1f/64 scope link
        valid_lft forever preferred_lft forever
```

bridge name	bridge id	STP	enabled	interfaces
eth0	8000.000c29865b1f	no	peth0 vif1.0 vif2.0	

```
default via 172.16.30.2 dev eth0
```

```
172.16.30.0/24 dev eth0  proto kernel  scope link  src 172.16.30.130
```

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	172.16.30.2	0.0.0.0	UG	0	0	0	eth0
172.16.30.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0

More about networks

- A dom0 can have multiple bridges (v.gr. Some domU might use one physical interface or might don't require real world access)

```
#in xend-config
(network-script my-custom-script)

#!/bin/sh
#My custom script in /etc/xen/scripts/
set -e
# First arg is the operation.
OP=$1
shift
script=/etc/xen/scripts/network-bridge
case ${OP} in
start)
    $script start vifnum=2 bridge=xenbr2 netdev=dummy0
    $script start vifnum=1 bridge=xenbr1 netdev=eth1
    $script start vifnum=0 bridge=xenbr0 netdev=eth0
    ;;

stop)
    $script stop vifnum=2 bridge=xenbr2 netdev=dummy0
    $script stop vifnum=1 bridge=xenbr1 netdev=eth1
    $script stop vifnum=0 bridge=xenbr0 netdev=eth0
    ;;

status)
    $script status vifnum=2 bridge=xenbr2 netdev=dummy0
    $script status vifnum=1 bridge=xenbr1 netdev=eth1
    $script status vifnum=0 bridge=xenbr0 netdev=eth0
    ;;

*)
    echo 'Unknown command: ' ${OP}
    echo 'Valid commands are: start, stop, status'
    exit 1

esac
```

- If there is no public IP use network-NAT
- If we need to do route traffic (i.e. firewall some traffic) use network-router

Networking in a closed environment

- If no public IP is available for domU, use NAT
 - Via Xen scripts
 - Via bridge+iptables+route: allows to mimic functionality of bridged (i.e., pseudo-VLANs)
- Via Xen scripts uses simplest approach: create a virtual interface per domU and use physical interface as a gateway (via route) and enabling ipv4 forwarding
 1. Enable NAT in xend-config.sxp

```
(network-script network-nat)
(vif-script vif-route)
```
 2. Configure explicitly the ip in domU config

```
vif=[ 'ip=10.0.0.2' ]
```
 3. Configure domU network interface statically

```
iface eth0 inet static
address 10.0.0.2
netmask 255.255.255.0
network 10.0.0.0
broadcast 10.0.0.255
gateway 10.0.0.1
```

domU networking

- Just add the entry in config file
 - `vif= ['mac=aa:1:2:3:4', 'bridge=xenbr0, ip="10.0.0.1"]`
 - IP is only relevant to xend
- In most cases (no multiple bridges or routes, and dhcp server in the network
 - `vif =[""]`
- In large scale environments basic tools
 - `brctl` (bridge control)
 - `iptables` (firewall)
 - `routed` (routing)
 - `tc` (QoS)

CPUs

- Xen schedules CPUs usage across all domains like regular OSes does with process
 - We can inspect scheduler with:
`xm dmesg | grep sched`
 - We can change scheduler appending to xen kernel the desired scheduler (credit by default)
 - Others (4.6+): real-time, hard real-time, credit 2, ...
- vCPU and CPU affinity
 - vCPU are “virtual” CPU visible by the guest kernel
 - Sets at domain config file with
`vcpu=#`
 - To inspect CPU assigned
`xm vcpu-list <domain>`
 - We can dynamically reduce the number of vCPUs (hot plug-CPU)
`xm vcpu-set <domain> #CPUs`
 - We can pin a vCPU to a physical CPU
`xm vcpu-pin <domain> <vCPU> <CPU>`

```
root@aos:/etc/xen# xm vcpu-list
```

Name	ID	VCPU	CPU	State	Time(s)	CPU	Affin
Domain-0	0	0	1	-b-	250.6	any	cpu
Domain-0	0	1	0	r--	73.9	any	cpu
manual	5	0	1	-b-	33.3	any	cpu
prueba	6	0	0	-b-	30.3	any	cpu
prueba	6	1	1	-b-	30.3	any	cpu

```
root@aos:/etc/xen# xm vcpu-pin prueba all 1
```

```
root@aos:/etc/xen# xm vcpu-pin manual all 1
```

```
root@aos:/etc/xen# xm vcpu-list
```

Name	ID	VCPU	CPU	State	Time(s)	CPU	Affin
Domain-0	0	0	0	-b-	251.3	any	cpu
Domain-0	0	1	1	r--	74.5	any	cpu
manual	5	0	1	-b-	33.4	1	
prueba	6	0	1	-b-	30.4	1	
prueba	6	1	1	-b-	30.3	1	

```
root@aos:/etc/xen# xm vcpu-set 0 1
```

```
root@aos:/etc/xen# xm vcpu-list
```

Name	ID	VCPU	CPU	State	Time(s)	CPU	Affin
Domain-0	0	0	0	r--	252.7	any	cpu
Domain-0	0	1	-	--p	75.0	any	cpu
manual	5	0	1	-b-	33.5	1	
prueba	6	0	1	-b-	30.4	1	
prueba	6	0	1	-b-	30.4	1	

Reduce on-line dom0 num CPU to 1
check with htop the number of CPUs
after and before

QoS with CPUs

- Usually #vCPU>#CPU
 - On average vCPU are most of the time idle, then it's a great way to save costs
- But, what happens if usage of vCPU is greater than CPU?
 - SLA might be different from domain to domain
 - We need to prioritize important "vCPU" (usually involves interactive work) over unimportant "vCPU" (usually involves batch work)
- Domain niceness
 - Controlled with "`xm sched-credit`"
 - Weight
 - How many CPU slots receive the domain in comparison with others
 - V.gr. A domain with weight 256 will receive half CPU slots than another with 512
 - Cap
 - Absolute maximum that a domain can receive
 - Usually controlled with the #CPU assigned (cap=0)
 - Otherwise can constrain #CPU (e.g. 50 is half #CPUs)
 - Useful for singular scenarios

```
root@aos:/etc/xen# xm sched-credit
Name                               ID Weight  Cap
Domain-0                           0    256    0
manual                             5    256    0
prueba                             6    256    0
root@aos:/etc/xen# xm sched-credit -d manual -w 512
root@aos:/etc/xen# xm sched-credit
Name                               ID Weight  Cap
Domain-0                           0    256    0
manual                             5    512    0
prueba                             6    256    0
```


Memory

- One of the critical resources
 - If the Hypervisor enters in trashing can be disastrous
 - Dom0 out of memory too
- Static setup DomU
 - `"memory="` value in config
- Static setup in Dom0
 - Controlled through grub or `xen-configure.sxd` or boot
 - Seems better configure it at boot (at least max memory)
 - `/etc/default/grub`
 - `GRUB_CMDLINE_XEN_DEFAULT="dom0_mem=512M,max:1024M"`
 - Update-grub will update xen entry in `/boot/grub/grub.cfg` (don't modify grub.cfg!!!)
 - `multiboot /boot/xen-4.1-amd64.gz placeholder dom0_mem=512M,max:1024M`
 - Also can limit CPU, network, ... in dom0
 - Is the most convenient way to configure it
- Dynamic modification (hot-plug) and memory ballooning
 - `xm mem-set domain value<max`
 - Kernel guest has to handle dynamically allocated memory
 - Kernel guest sets for max at boot time. **Increase max will require reboot to make it visible**
 - DomU has a "ballooning" driver that "request/inflate" and "return/deflate" to hypervisor

Memory Oversubscription (memory ballooning)

- Like in CPU, we might want to oversubscribe memory
 - $\sum \text{domU}_{\text{MaxMem}} > \text{HW}_{\text{Mem}}$
- Memory ballooning could allow to increase or decrease the amount of memory allocated by the domU dynamically
- Dom0 has memory ballooning enabled by default
 - `(enable-dom0-ballooning yes)` in `xend-config.sxp`
 - A good policy is, at least, guarantee a minimum memory and reserve an exclusive cpu for dom0

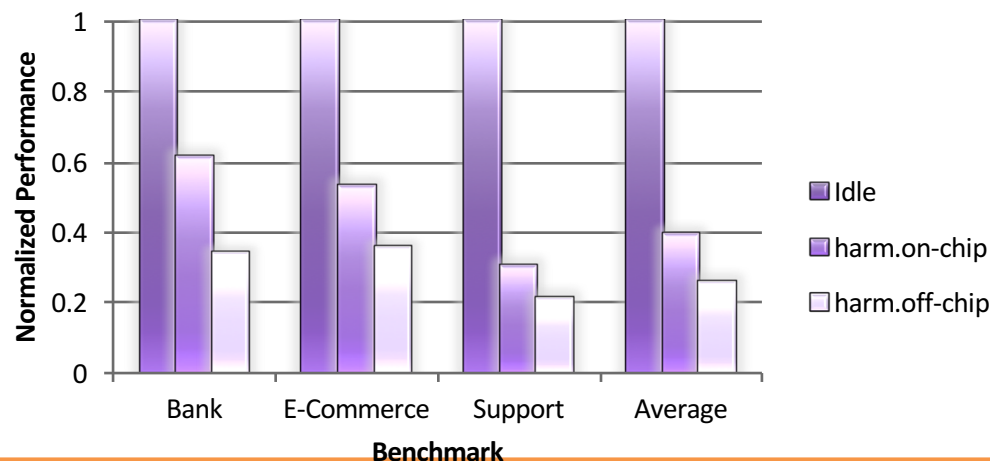
Memory QoS

■ Capacity

- Guarantee that $\sum \text{domU}_{\text{UsedMem}} \leq \text{HW}_{\text{MaxMem}}$
- A page fault in hypervisor and not in the guest could have a catastrophic results for performance

■ Little margin caches and off chip bandwidth

- One misbehaving domU can trash others



Intel Memory Hierarchy QoS Technologies

(Cache Monitoring Technology)

- Introduced in Intel Xeon E5 v3 (Broadwell), 2013
- Three main parts
 - Cache Management Technology (CMT)
 - Cache Allocation Technology (CAT) (Only in a subset of Xeon E5 v3)
 - Memory Bandwidth Monitoring (MBM) (???)
- How works
 - The hardware exposes to the VMM
 - The LLC utilization and memory bandwidth per VM (CMT, BMT)
 - And the knobs to control cache utilization (CAT)
- Xen implements the logic to use this since 4.5-4.6
 - More information
http://wiki.xenproject.org/wiki/Intel_Platform_QoS_Technologies