MVMUA, 12 January 2016

Eric Farman
IBM, z Systems Virtualization

Agenda

- Options
- Terminology
- Dispatcher / Interrupts
- Device virtualization
- Control Program interfaces
- Guest definition and management

KVM for z Systems for z/VM Lovers

- There are a myriad of KVM introductions out there, so what's the point of this?
  - If you know z/VM (specifically, 6.3.0), then we can draw parallels to KVM
  - If you know KVM (for x86 or Power), then we can draw parallels to z/VM
- Regarding s390x (nay, z Systems), neither hypervisor replaces the other

IBM z Systems Virtualization Options

IBM z Systems now has three strategic virtualization platforms

- IBM Processor Resource/System Manager (PR/SM)
- IBM z/VM
- KVM for IBM z Systems

KVM

KVM for IBM z provides an open source choice for IBM z Systems virtualization for Linux workloads. Best for clients that are not familiar with z/VM and are Linux centric admins.

z/VM

z/Proprietary Server Virtualization that is completely integrated into the full stack. Complete hardware awareness. Supported on all IBM z Systems servers. z/VM will continue to be enhanced to support Linux Workloads.

PR/SM

Divide one physical server into up to 85 logical partitions (LPAR) running a mix of multiple z/OS, z/VM, Linux, KVM for IBM z, Transaction Processing Facility (TPF) and z/VSE instances isolated and secured in parallel. Share resources across LPARs or dedicated to a particular LPAR. Running a mix of multiple z/OS, z/VM, Linux, TPF, KVM for IBM z and z/VSE instances isolated and secured in parallel.

IBM z/VM and KVM for IBM z can co-exist on z Systems

IBM z/VM

World class quality, security, reliability - powerful and versatile

Extreme scalability creates cost savings opportunities

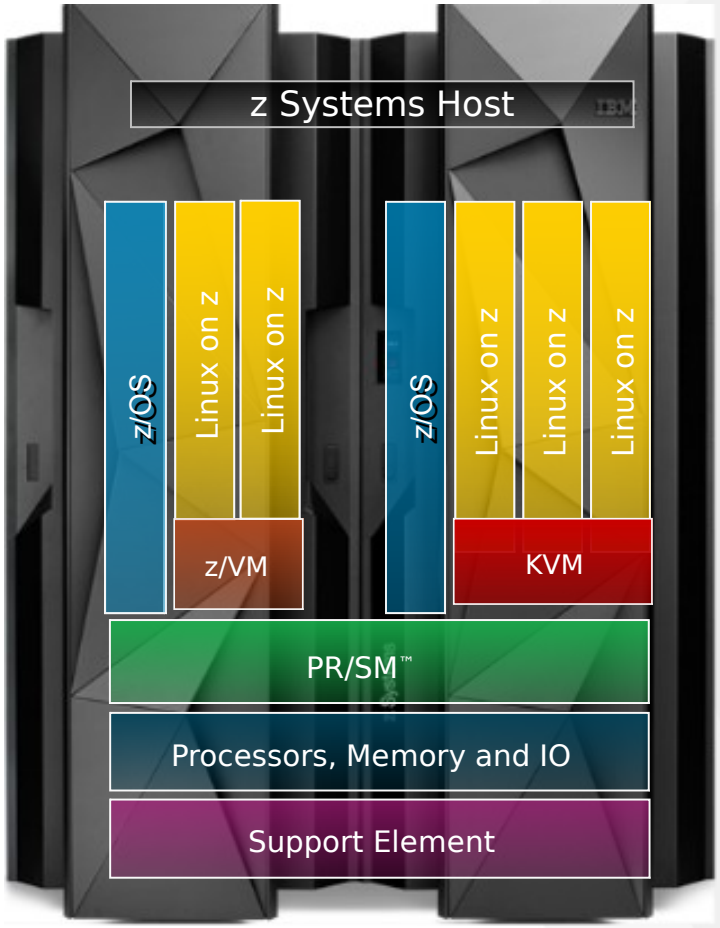
Exploitation of advanced technologies, such as:

- Shared memory (Linux kernel, executables, communications)

Highly granular control over-resource pool

Provides virtualization for all z Systems operating systems

Integrates with OpenStack



KVM for IBM z

Standardizes configuration and operation of server virtualization

Leverage common Linux administration skills to administer virtualization

Flexibility and agility leveraging the Open Source community

Provides an Open Source virtualization choice

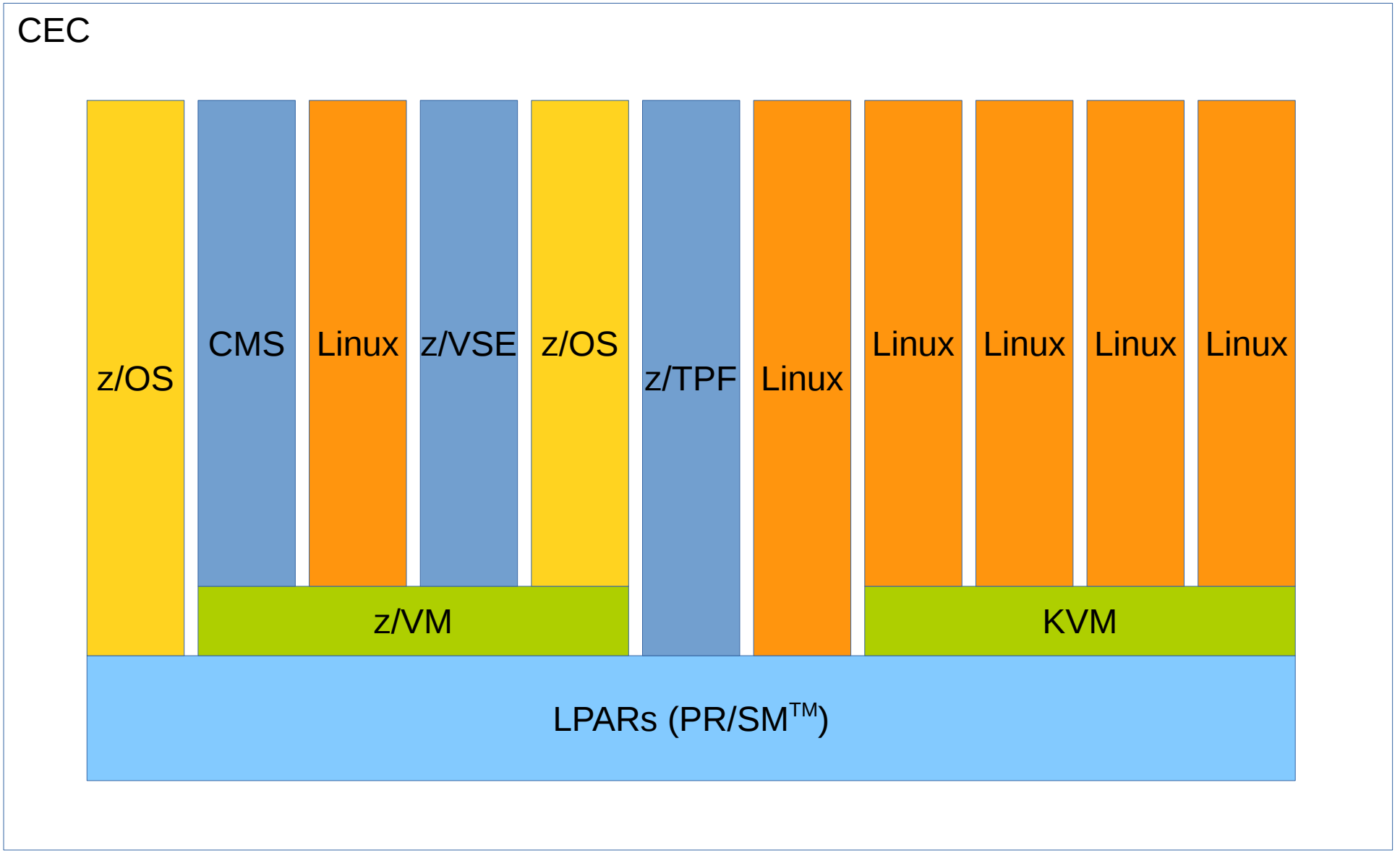
Integrates with OpenStack

Terminology

z Systems	“Open World”
Storage	Memory
DASD (Direct Access Storage Device)	Storage, disk*
CPU, PU, IFL, SAP, ...	CPU, Processor
IPL	Boot
CEC (Central Electronics Complex)	Computer
...	...

* The FCP/SCSI LUN concepts from the open world are used in their natural words in z Systems. Thus, “Disk” = “Disk”, “LUN” = “LUN”, etc. This bullet is merely regarding ECKD DASD devices as “disks”.

Hypervisors



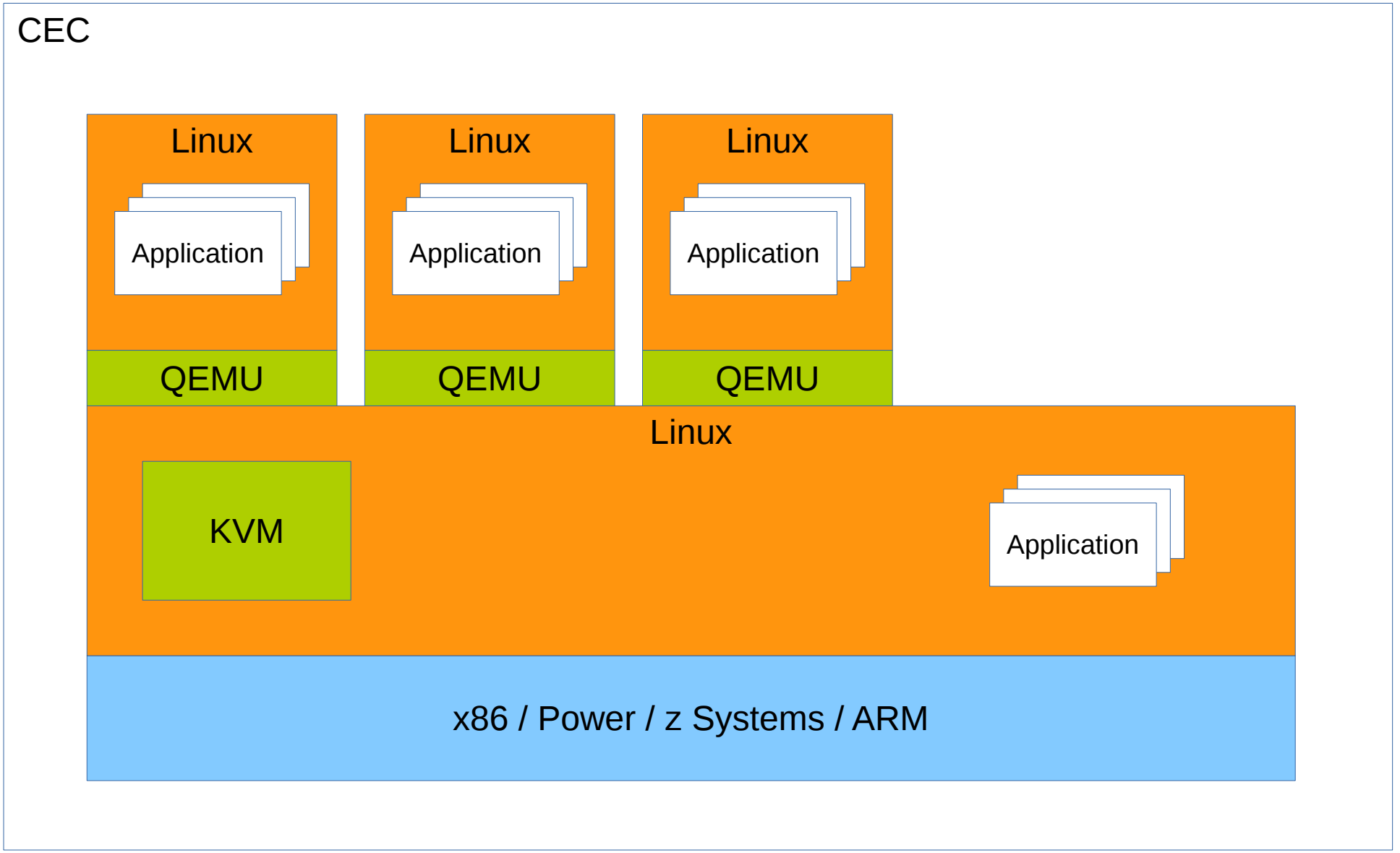
Hypervisor Interfaces

- z/VM
  - Host provides commands to query/change the environment and configuration
  - Can use CMS to provide further functionality
  - Managed by DirMaint, SMAPI, or alternative solutions/exploiters
    - VM:Secure, IBM Wave, xCat, etc.
- KVM
  - Host provides commands to query/change the environment and configuration
  - Host is a full-scale Linux system, with all its existing tools
  - Managed by exploiters of libvirt API
    - virsh (CLI), Openstack, virt-manager, etc.

KVM Hypervisor

- Linux kernel provides base system capabilities
- KVM (Kernel-based Virtual Machine) module provides hypervisor functions
- QEMU (Quick EMUlator) user space application provides device virtualization and emulation
- Libvirt API provides “human-friendly” interface to QEMU processes

KVM Hypervisor



Guest Definition

- z/VM
  - Ultimately, guest is a directory entry that is read during LOGON process
    - Each guest CPU represented by a VMDBK within z/VM
- KVM
  - Ultimately, guest is an invocation of qemu, with various options
    - Each guest represented by a process within Linux

Basic Guest Definition (Example)

- z/VM
  - Directory Entry

```
USER LINUX01 MYPASS 512M 1024M G
MACHINE ESA 2
IPL 190 PARM AUTOCR
CONSOLE 01F 3270 A
SPOOL 00C 2540 READER *
SPOOL 00D 2540 PUNCH A
SPOOL 00E 1403 A
SPECIAL 500 QDIO 3 SYSTEM MYLAN
MDISK 191 3390 012 001 ONEBIT M
MDISK 200 3390 050 100 TWOBIT MR
```

- KVM
  - QEMU command

```
/usr/bin/qemu-system-s390x
 -name LINUX01
 -machine s390-ccw-virtio,accel=kvm,usb=off
 -m 512
 -smp 2,sockets=2,cores=1,threads=1
 -drive file=/dev/disk/by-path/ccw-0.0.0e1e-part1,
  if=none,id=virtio-disk0,format=raw,cache=none
 -device virtio-blk-ccw,scsi=off,
  drive=virtio-disk0,id=disk0,bootindex=1
 -netdev tap,fd=15,id=hostnet0
 -device virtio-net-ccw,netdev=hostnet0,id=net0,
  mac=10:ba:11:be:15:ef
 -chardev pty,id=charconsole0
 -device virtconsole,chardev=charconsole0,id=console0
```

Basic Guest Definition (Example)

- z/VM
  - Directory Entry

```
USER LINUX01 MYPASS 512M 1024M G
MACHINE ESA 2
IPL 190 PARM AUTOCR
CONSOLE 01F 3270 A
SPOOL 00C 2540 READER *
SPOOL 00D 2540 PUNCH A
SPOOL 00E 1403 A
SPECIAL 500 QDIO 3 SYSTEM MYLAN
MDISK 191 3390 012 001 ONEBIT M
MDISK 200 3390 050 100 TWOBIT MR
```

- KVM
  - XML (abbreviated)

```
<domain type='kvm'>
  <name>LINUX01</name>
  <memory unit='MB'>512</memory>
  <vcpu>2</vcpu>
  <os>
    <type arch='s390x' machine='s390-ccw-virtio'>
      hvm
    </type>
  </os>
  <devices>
    <console type='pty'>
      <target type='sclp'/>
    </console>
    <interface type='direct'>
      <mac address='10:ba:11:be:15:ef'/>
      <source dev='eth1' mode='bridge'/>
    </interface>
    <disk type='block' device='disk'>
      <driver name='qemu' type='raw' cache='none'/>
      <source dev='/dev/disk/by-path/ccw-0.0.0e1e-part1
    </disk>
  </devices>
</domain>
```

Virtualization Basics

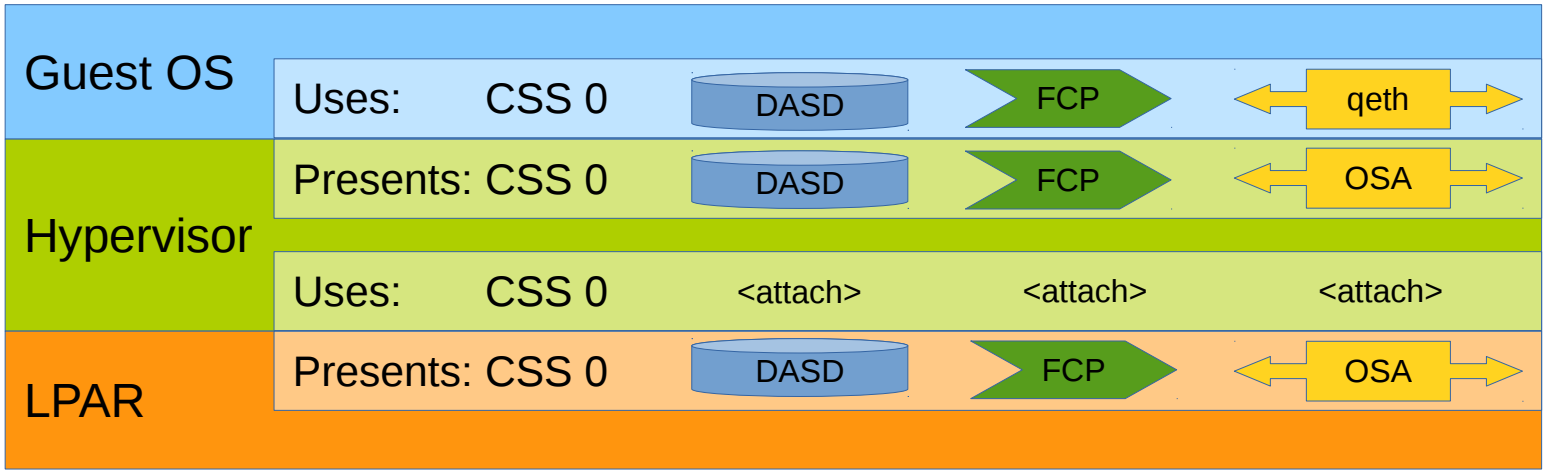
- z/VM
  - Replicates the z/Architecture Principles of Operation
  - Permits overcommitment of real hardware
  - Provides both emulation and passthrough access to I/O devices
- KVM
  - Replicates the z/Architecture Principles of Operation
  - Permits overcommitment of real hardware
  - Provides virtio-based access to I/O devices

Device Virtualization

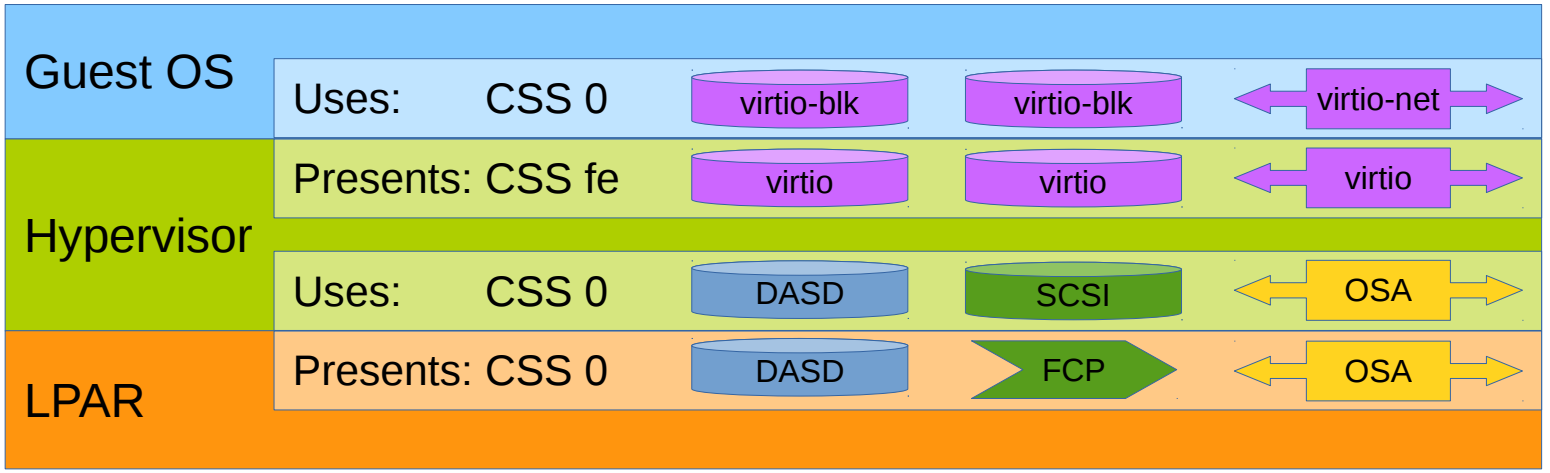
- z/VM
  - Trap instructions to emulate I/O
    - Architecture provides certain instructions to paravirtualize I/O
  - Utilizes hardware features to pass through devices, letting guest drive I/O without hypervisor exit
    - Initial setup still requires host involvement
- KVM
  - Trap instructions to emulate I/O
    - Architecture provides certain instructions to paravirtualize I/O

Device Virtualization – Enumeration

- z/VM



- KVM



Device Virtualization – Enumeration

- z/VM
  - Devices enumerated via channel subsystem
  - Data transfer
    - Start Subchannel and CCW
    - Queued Direct I/O
      - Initial setup via start subchannel
  - Several device types can use the same transport scheme
- KVM
  - Devices enumerated via channel subsystem
  - Data transfer
    - No SSCH / CCW
    - virtio
      - Initial setup via start subchannel
  - Several device types can use the same transport scheme

Device Virtualization – Data transfer

- z/VM
  - Subchannel-based I/O
    - Represented by Channel Command Words (CCWs)
  - QDIO-based I/O
    - Hardware assists avoid hypervisor exits
    - Implements Adapter Interrupt architecture
- KVM
  - Virtio transport-based I/O
    - Structural similarities to QDIO
    - No hardware assists
      - Latency optimizations (dataplane, eventfd, vhost) exist in KVM
    - Implements Adapter Interrupt architecture

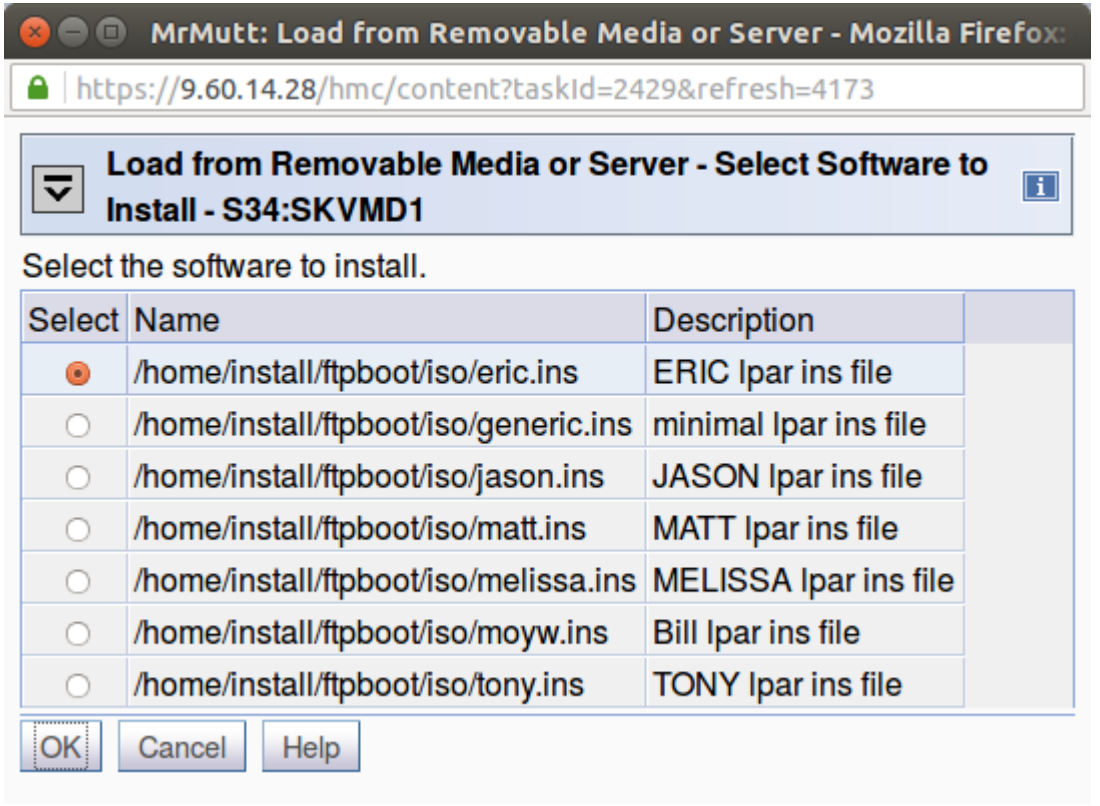
Device Virtualization – Disks

- z/VM
  - Dedicate full volume
    - Also FCP subchannels for access to SAN fabric
  - Minidisks carve a device into smaller/shared components
  - Fully virtual disks in memory
  - Shared filesystem (SFS)
- KVM
  - Dedicate full volume, partition, or LVM
    - Uses virtio drivers, not native configuration
  - Image files residing on host filesystem
    - Can be on NFS or cluster filesystem for migration

Host Installation

Host Installation

- Installable via FTP or HMC-DVD
- Copy “iso” directory to a R/W location, to update generic.prm file
  - HMC reads contents of this directory



Host Installation

```
# ls /home/install/ftpboot/iso/
eric.ins     images     matt.ins     moyw.ins  repodata  TRANS.TBL
generic.ins  jason.ins  melissa.ins  Packages  tony.ins
# cat /home/install/ftpboot/iso/eric.ins
* ERIC lpar ins file
images/kernel.img 0x00000000
images/initrd.img 0x02000000
images/eric.prm 0x00010480
images/initrd.addrsize 0x00010408

# cat /home/install/ftpboot/iso/images/generic.prm
ro ramdisk_size=40000 rd.dasd=autodetect systemd.show_status=0
#
# cat /home/install/ftpboot/iso/images/eric.prm
inst.repo=ftp://install:install@9.60.29.66/ftpboot/iso
rd.zfcp=0.0.1c98,0x500507630603c763,0x4022403300000000
rd.zfcp=0.0.1ca8,0x500507630608c763,0x4022403300000000
rd.znet=qeth,0.0.4300,0.0.4301,0.0.4302,layer2=1
ip=9.60.18.175::9.60.18.129:255.255.255.128:frolic:enccw0.0.4300:n
one systemd.show_status=0
```

Guest Installation

Guest Installation

- Copy initrd and cd.ikr from installation media to /var/lib/libvirt/images on the host system
- Create a qcow2 image file for guest filesystem

```
# cp -p /mnt/SLES12/boot/s390x/initrd /var/lib/libvirt/images/sles12-initrd.boot
# cp -p /mnt/SLES12/boot/s390x/cd.ikr /var/lib/libvirt/images/sles12-kernel.boot
#
# qemu-img create -f qcow2 /var/lib/libvirt/images/sles12_qcow2.img 20G
Formatting '/var/lib/libvirt/images/sles12_qcow2.img', fmt=qcow2 size=21474836480
 encryption=off cluster_size=65536 lazy_refcounts=off refcount_bits=16
# qemu-img info /var/lib/libvirt/images/sles12_qcow2.img
image: /var/lib/libvirt/images/sles12_qcow2.img
file format: qcow2
virtual size: 20G (21474836480 bytes)
disk size: 196K
cluster_size: 65536
Format specific information:
    compat: 1.1
    lazy refcounts: false
    refcount bits: 16
    corrupt: false
```

Guest Installation (Example)

```
# cat lmb_guest.xml
<domain type='kvm'>
  <name>lmb_guest</name>
  <memory unit='MiB'>1024</memory>
  <vcpu>1</vcpu>
  <os>
    <type arch='s390x' machine='s390-ccw-virtio'>hvm</type>
    <kernel>/var/lib/libvirt/images/sles12-kernel.boot</kernel>
    <initrd>/var/lib/libvirt/images/sles12-initrd.boot</initrd>
  </os>
  <devices>
    <emulator>/usr/bin/qemu-system-s390x</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none' io='native'/>
      <source file='/var/lib/libvirt/images/sles12_qcow2.img'/>
      <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0002'/>
    </disk>
...
```

Guest Definition

Guest Definition (Example)

```
# cat lmb_guest.xml
<domain type='kvm'>
  <name>lmb_guest</name>
  <memory unit='MiB'>1024</memory>
  <vcpu placement='static' current='1'>4</vcpu>
  <os>
    <type arch='s390x' machine='s390-ccw-virtio'>hvm</type>
    <kernel>/boot/vmlinux-4.4.0-rc5</kernel>
    <initrd>/boot/initramfs-4.4.0-rc5.img</initrd>
    <cmdline>selinux=0 root=/dev/vda</cmdline>
  </os>
  <devices>
    <emulator>/usr/bin/qemu-system-s390x</emulator>
    <disk type='block' device='disk'>
      <driver name ='qemu' type='raw' cache='none'/>
      <source dev='/dev/disk/by-path/ccw-0.0.0e1e-part1'/>
      <target dev='vda' bus='virtio'/>
      <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0e1e'/>
    </disk>
...
```

Guest Definition (Example) Cont'd

```
...
    <interface type='direct'>
      <mac address='10:ba:11:be:15:ef'/>
      <source dev='enccw0.0.4300' mode='bridge'/>
      <model type='virtio'/>
    </interface>
    <console type='pty'>
      <target type='virtio'/>
    </console>
  </devices>
</domain>
```

Guest Definition

```
# virsh define lmb_guest.xml
Domain lmb_guest defined from lmb_guest.xml

# virsh list --all
 Id    Name                           State
----------------------------------------------------
 -     lmb_guest                      shut off
```

Guest Definition (Example)

```
# virsh dumpxml lmb_guest
<domain type='kvm'>
  <name>lmb_guest</name>
  <uuid>e748ac35-7f3b-4294-9cbd-88576b41e369</uuid>
  <memory unit='KiB'>1048576</memory>
  <currentMemory unit='KiB'>1048576</currentMemory>
  <vcpu placement='static' current='1'>4</vcpu>
  <iothreads>1</iothreads>
  <os>
    <type arch='s390x' machine='s390-ccw-virtio-2.5'>hvm</type>
    <kernel>/boot/vmlinux-4.2.0</kernel>
    <initrd>/boot/initramfs-4.2.0.img</initrd>
    <cmdline>selinux=0 root=/dev/vda</cmdline>
    <boot dev='hd'/>
  </os>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
...
```

Guest Definition (Example) Cont'd

```
...
  <devices>
    <emulator>/usr/bin/qemu-system-s390x</emulator>
    <disk type='block' device='disk'>
      <driver name='qemu' type='raw' cache='none'/>
      <source dev='/dev/disk/by-path/ccw-0.0.0e1e-part1'/>
      <target dev='vda' bus='virtio'/>
      <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0e1e'/>
    </disk>
    <controller type='virtio-serial' index='0'>
      <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0001'/>
    </controller>
    <interface type='direct'>
      <mac address='10:ba:11:be:15:ef'/>
      <source dev='enccw0.0.4300' mode='bridge'/>
      <model type='virtio'/>
      <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0000'/>
    </interface>
    <console type='pty'>
      <target type='virtio' port='0'/>
    </console>
    <memballoon model='virtio'>
      <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0002'/>
    </memballoon>
  </devices>
</domain>
```

Guest management

```
# virsh list --all
 Id    Name                           State
----------------------------------------------------
 1     lmb_guest                      shut off

# virsh start lmb_guest
Domain lmb_guest started

# virsh list --all
 Id    Name                           State
----------------------------------------------------
 2     lmb_guest                      running

# virsh console lmb_guest
Connected to domain lmb_guest
Escape character is ^]

[root@kvmguest ~]#
```

Guest Management

Guest management

- From guest perspective, all devices have control unit type 3832 of varying models
  - Model -01 is a virtio-net device
  - Model -02 is a virtio-blk device
  - ...
- Device type/model are both zero
- Single virtual channel path, of ID x00

```
[root@kvmguest ~]# lscss
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.0001 0.0.0000  0000/00 3832/03 yes  80  80  ff   00000000 00000000
0.0.0e1e 0.0.0001  0000/00 3832/02 yes  80  80  ff   00000000 00000000
0.0.0000 0.0.0002  0000/00 3832/01 yes  80  80  ff   00000000 00000000
0.0.0002 0.0.0003  0000/00 3832/05 yes  80  80  ff   00000000 00000000
```

Guest management

```
# virsh dominfo lmb_guest
Id:             2
Name:           lmb_guest
UUID:           e748ac35-7f3b-4294-9cbd-88576b41e369
OS Type:        hvm
State:          running
CPU(s):         1
CPU time:       7.6s
Max memory:     1048576 KiB
Used memory:    1048576 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: none
Security DOI:   0
```

Guest management

```
# /usr/bin/virsh dominfo lmb_guest | grep memory
Max memory:     1048576 KiB
Used memory:    1048576 KiB

# /usr/bin/virsh setmem lmb_guest 512M

# /usr/bin/virsh dominfo lmb_guest | grep memory
Max memory:     1048576 KiB
Used memory:    524288 KiB
```

Guest management

```
# chccwdev -e 1c98
Setting device 0.0.1c98 online
Done
# echo 0x04022401100000000 > /sys/bus/ccw/devices/0.0.1c98/0x500507630603c763/unit_add
# lsscsi
[0:0:15:1074872354]disk    IBM      2107900          .217  /dev/sda

# cat disk.xml
    <hostdev mode='subsystem' type='scsi'>
      <source>
        <adapter name='scsi_host0'/>
        <address bus='0' target='15' unit='1074872354'/>
      </source>
      <address type='drive' controller='0' bus='0' target='0' unit='16383'/>
    </hostdev>
# /usr/bin/virsh attach-device lmb_guest disk.xml
Device attached successfully
```

```
scsi 0:0:0:32767: Direct-Access     IBM      2107900          .217 PQ: 0 ANSI: 5
sd 0:0:0:32767: Attached scsi generic sg0 type 0
sd 0:0:0:32767: [sda] 16777216 512-byte logical blocks: (8.58 GB/8.00 GiB)
sd 0:0:0:32767: [sda] Write Protect is off
sd 0:0:0:32767: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO...
 sda: sda1
sd 0:0:0:32767: [sda] Attached SCSI disk
# lsscsi
[0:0:0:32767]disk    IBM      2107900          .217  /dev/sda
```

Guest migration

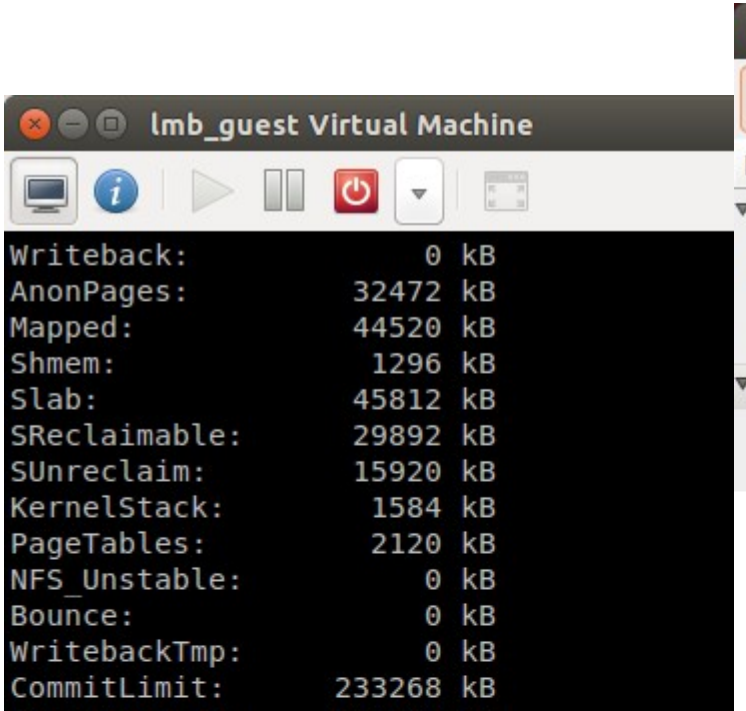
- Source and target systems must have access to the same or equivalent resources
  - Use consistent naming on both systems
    - e.g., /dev/disk/by-id/... instead of /dev/dasd...
  - Image files should be on shared storage (e.g., nfs)
- Performed in two phases

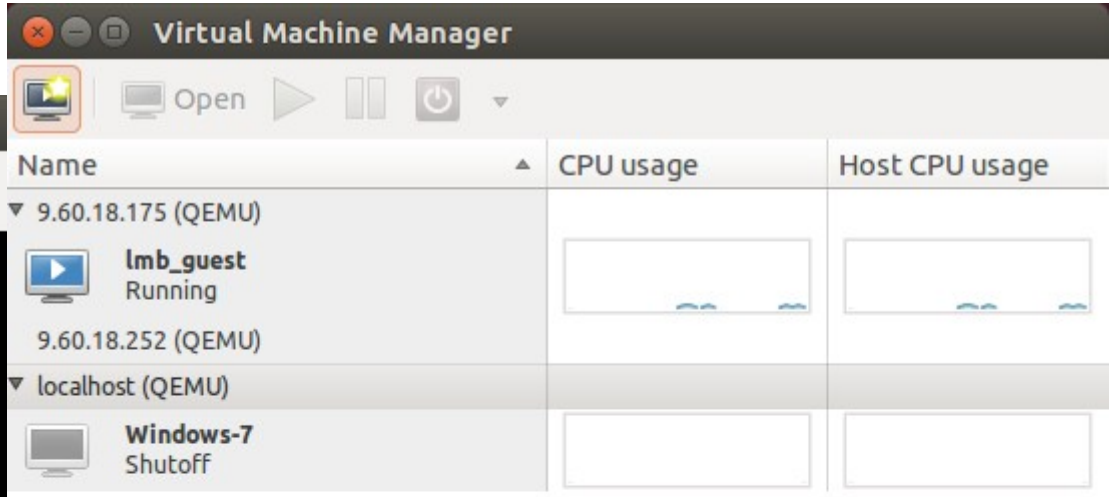
```
# virsh migrate --live --verbose --tunnelled –p2p lmb_guest qemu+ssh://9.60.18.252/system
Migration: [100 %]
```

KVM Management

KVM Management

- virt-manager
  - Simple GUI interface, likely included in your favorite Linux distribution, can connect to multiple hosts to create, start/stop, change, manage, clone guests
  - Connects via ssh
  - Provides console access





KVM Management

- OpenStack
  - Code enabling KVM for IBM z Systems is upstream (kilo)
  - OpenStack Support Matrix
- IBM Cloud Manager
  - Support for KVM for IBM z Systems as a compute node added to 4.3.0.3 (GA'd September 2015)

Debugging

Host Debugging

- z/VM
  - CP Trace Table
    - Internal interface maintained by nucleus of its activities (calls, mallocs, etc.)
    - Extracted via TRSAVE or system dump
  - CP TRSOURCE
    - Command to record host data structures
- KVM
  - Kernel provides many tracepoints (debuginfo)
    - Controlled via virtual filesystem
  - Additional tools exist to profile tracepoints
    - blktrace, perf sched, etc.

Guest Debugging

- z/VM
  - CP TRACE
    - Trap on data, addresses, memory writes, instruction
    - Can show/modify memory/registers
      - display, store
    - Can search memory
      - LOCATEVM
    - Can modify execution
      - SKIP, PASS, STOP
- KVM
  - gdb
    - Trap on data, addresses, memory writes, source line
    - Can show/modify memory/registers
      - info, print, set
    - Can search memory
      - find
    - Can modify execution
      - skip, continue, step

Guest Debugging

- z/VM
  - TRACE is major front end
  - Based on Program-Event-Recording
- KVM
  - qemu provides a gdb server (option -s), which can be connected to
    - target remote localhost:1234
  - Software breakpoints alter the guest
    - Visible to guest, but fast
  - Hardware breakpoints use Program-Event-Recording
    - Invisible, but can slow down

KVM Product Information and Education

KVM Product Information

- Available via ShopZ:
  - Product: 5648-KVM
  - S&S: 5648-KVS
- Hardware
  - zBC12/zEC12 or LinuxONE Rockhopper
    - OSA: H49525.013 D15F Bundle 45a
  - z13 or LinuxONE Emperor
    - OSA: N98805.010 D22H Bundle 20a
- Storage
  - DS8K, XIV, SVC, V7K, Flash Systems
- Guest
  - SUSE SLES12 SP1

KVM Education

- KVM Forum 2015
  - Jens Freimann: Pushing the limits: 1000 guests and beyond
  - David Hildenbrand: Guest operating system debugging
- KVM for IBM z Systems
- KVM Knowledge Center
- Upstream:
  - KVM: http://www.linux-kvm.org/
  - QEMU: http://www.qemu.org/
  - Libvirt: http://www.libvirt.org/

IBM

Fin

Question, comments, concerns?

Eric Farman
IBM z Systems Virtualization
Email: farman@us.ibm.com

BACKUP

Virtualization Basics

- Both z/VM and KVM
  - Use “Start Interpretive Execution” (SIE) instruction to “run” virtual processors
    - SIE uses a control block that describes the virtual processor state
    - SIE uses the Dynamic Address Translation (DAT) tables for the virtual machine
  - Get control back for various reasons
    - (Host) page fault
    - I/O
    - Privileged instruction (including service calls, like DIAG)
  - Can kick another processor “out of SIE”

Interruptibility / Concurrency

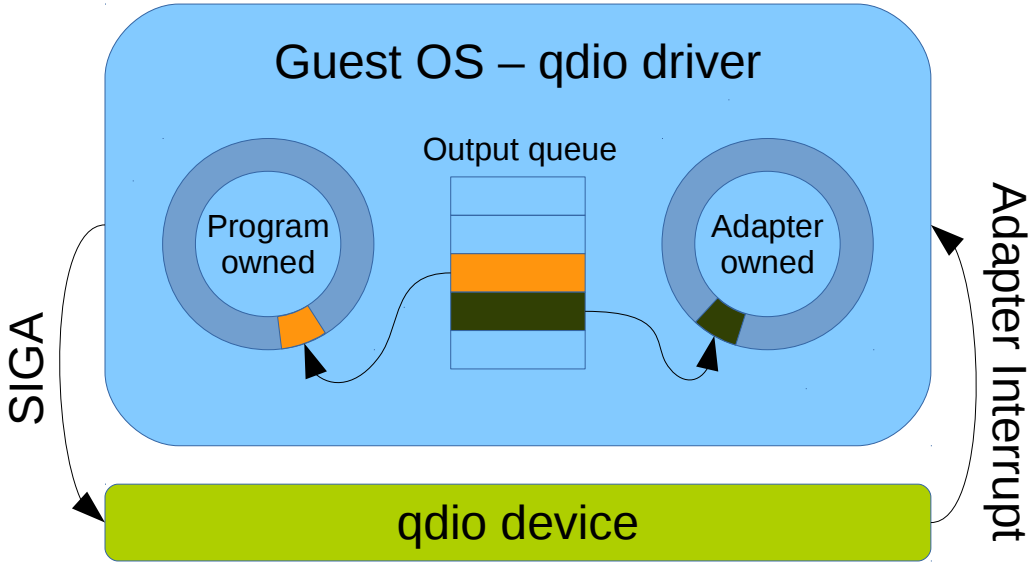
- z/VM
  - Some tasks require a master CPU
  - Some core parts of z/VM run with interrupts disabled
    - Eases data handling
  - Optimized for efficiency
- KVM
  - No master CPU concept
  - Runs almost always with interrupts enabled
    - Explicit/implicit pre-emption
    - Code needs to be safe against concurrency effects
  - Optimized for low latency

Dispatcher

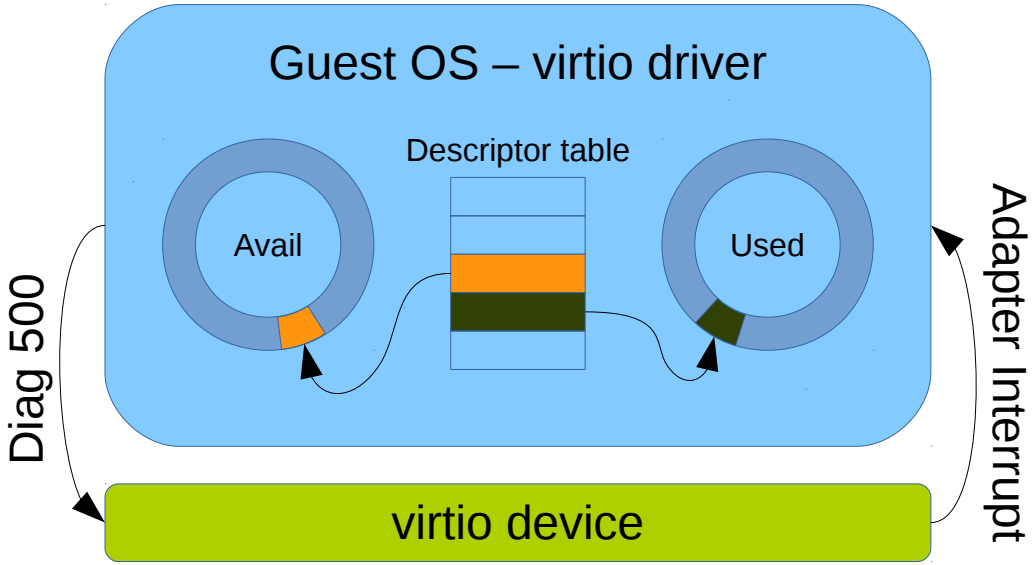
- z/VM
  - Scheduler determines priorities based on shares and other factors
  - Dispatcher runs a virtual processor on a logical processor
  - Uses a central dispatch list
    - Queue of runnable VMDBKs kept in order by urgency
    - Tries to preserve VMDBK homes when assigning to dispatch vectors
    - Tries to keep VMDBKs of a guest topologically close
  - Single entity (dispatcher) is good for central decisions
- KVM
  - Scheduler handles prioritization and dispatching of processes
  - Uses one runqueue per CPU
    - Holds task structs for each context (process/thread) ordered by priority/fairness
    - Processes are pulled to or migrated off a runqueue, but they always belong to one
    - The more hierarchies a migration travels the more expensive it is considered to be
    - Tries to group related processes to simplify IPC
  - Spread entity (scheduler) is good for scaling

Device Virtualization – Data transfer

- z/VM



- KVM

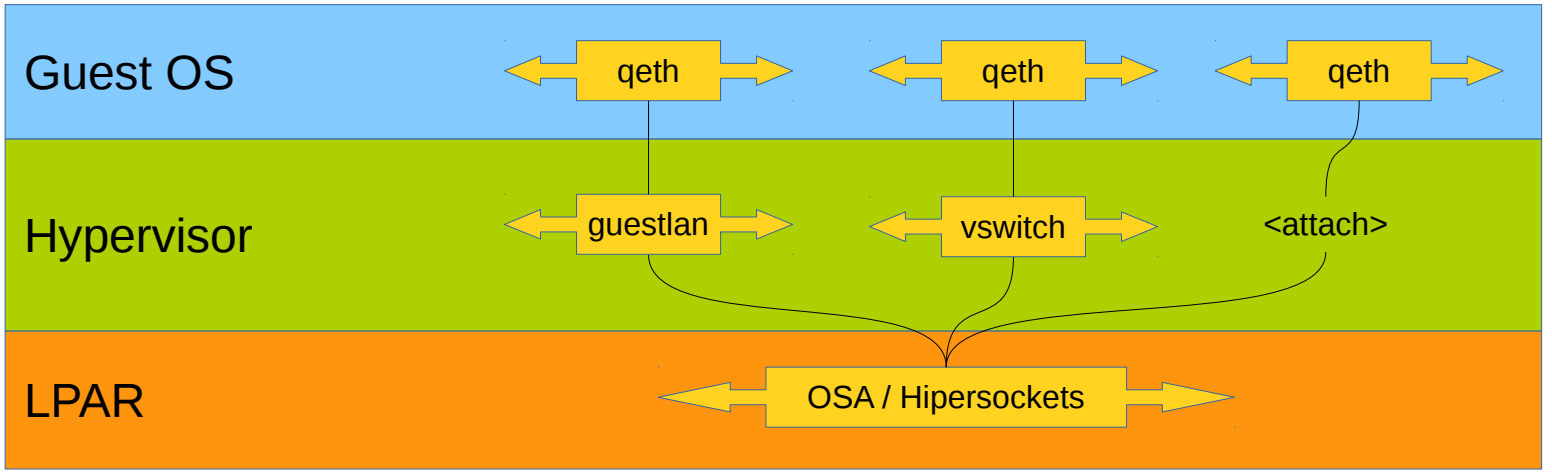


Device Virtualization – Networking

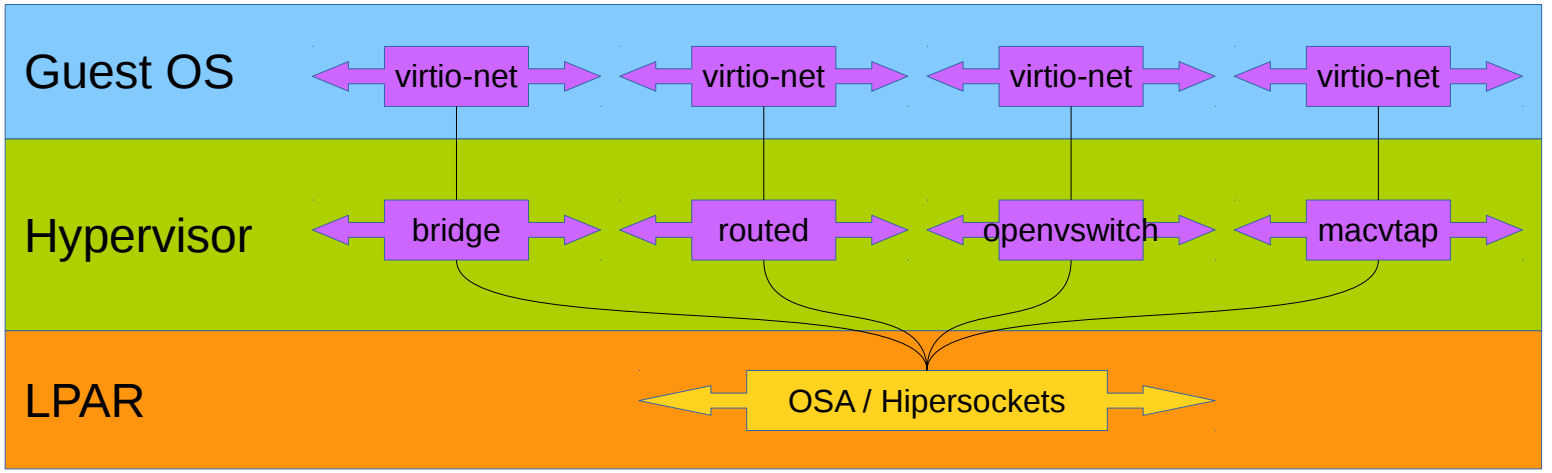
- z/VM
  - Host devices can be
    - Dedicated to virtual machine
    - Connected to guest LAN
    - Connected to VSWITCH
- KVM
  - Host devices can be associated with a virtual machine via
    - macvtap
    - openvswitch
    - bridge

Device Virtualization – Networking

- z/VM



- KVM



Paging

Paging

- z/VM
  - Block* paging to disk
    - Evacuation is performed in groups, to keep IOPs down
    - Pages needed back from disk are also read in groups
  - Maximum of 256 devices of 64GiB each
- KVM
  - Swapping is done a per-page basis
    - Page-out are done in so-called “page clusters”
      - Assumes pages that age together can be read together as well
      - Avoid IOPs
    - Page-in reads
  - Maximum of 30 devices of ~32YiB each

* Prior to z/VM 6.3, demand paging existed between central and expanded storage

Paging (examples)

- z/VM – 64 x 16GiB devices

```
vmcp "query alloc page"
                EXTENT      EXTENT  TOTAL  PAGES    HIGH     %
VOLID  RDEV      START         END  PAGES IN USE    PAGE USED
------ ---- ---------- ---------- ------ ------ ------- ----
102964 488F          8   16777214 16384K     29      63    1%
[...]
102901 4850          8   16777214 16384K     24      60    1%
                                    ------ ------          ----
SUMMARY                             1024M    628            1%
USABLE                              1024M    628            1%
```

Per-disk usage info

- KVM/Linux – 16 x 256GiB devices

```
cat /proc/swaps
Filename                        Type            Size            Used    Priority
/dev/dm16                       partition       268434428       0       100
[...]
/dev/dm29                       partition       268434428       0       100
# cat /proc/meminfo | grep -i swap
SwapCached:                0 kB
SwapTotal:       4288659392 kB
SwapFree:        4288659392 kB
```

Totals in the ~4TiB range

Terminology – Tricky parts

z Systems	“Open World”
(Page) frame	Page
Page Table	Page Table
FRMTE	Page struct (often just called “page”)
...	...

- Some things use the same names, but can mean different things depending on context
  - This is especially true in the following pages regarding memory management

Paging a page

- z/VM
  - Page table holds address resolution
  - For resident pages, there will also be a FRMTE which maps real memory usage and holds extra information
  - Non-resident pages have no FRMTE
    - From the PTE, z/VM can reach serialization information via PGSTE/ASATE
- KVM
  - Page table holds address resolution
    - Also HW bits like “validity”
    - Also OS bits used to flag swap device
  - A page can be
    - In memory (anonymous)
    - On backing storage (swapped out)
    - On both (swap cache)

Paging – Available list

- z/VM
  - Free frames are on the available list
  - Holds certain categories
    - <2G, >2G, single or contiguous
    - Allocations are served from here
- KVM
  - Free pages are on the free list
  - Such a list is member of a zone
    - Zone DMA covers the 31-bit range (<2G)
    - Zone Normal is >2G
    - Structured in “orders” for contiguity
      - Order 0 = 2^0 contiguous pages
      - Order 1 = 2^1 contiguous pages

Paging – Page-out selection

- z/VM
  - Demand scan pushes frames from
    - Owned valid frames, to
    - Owned “invalid but resident” frames, to
    - Global aging list, to
    - The available list
- KVM
  - Reclaim code pushes pages from
    - The active list, to
    - The inactive list, to
    - The free list

Paging – Scans and disk position

- z/VM
  - Optional pre-writing allows eviction in case of demand
    - If not enough, demand scan kicks in
  - A page almost always goes back to the same DASD slot
  - A page not chagned since last read from DASD is almost never re-written
- KVM
  - Prewriting via watermarks once free pages drop below high watermark
    - Below the low watermark allocating processes contribute time (direct reclaim)
  - A page “owns” its swap slot, which never changes
  - There is no swap re-write until a page is made dirty
  - Reclaim work at the end of the inactive list, until it freed enough pages
    - Anon pages get swap slots assigned and written asynchronously
    - Dirty pages are written
    - Clean pages are discarded