KVM Architecture Overview

Virtualization on Linux x86-64

Stefan Hajnoczi

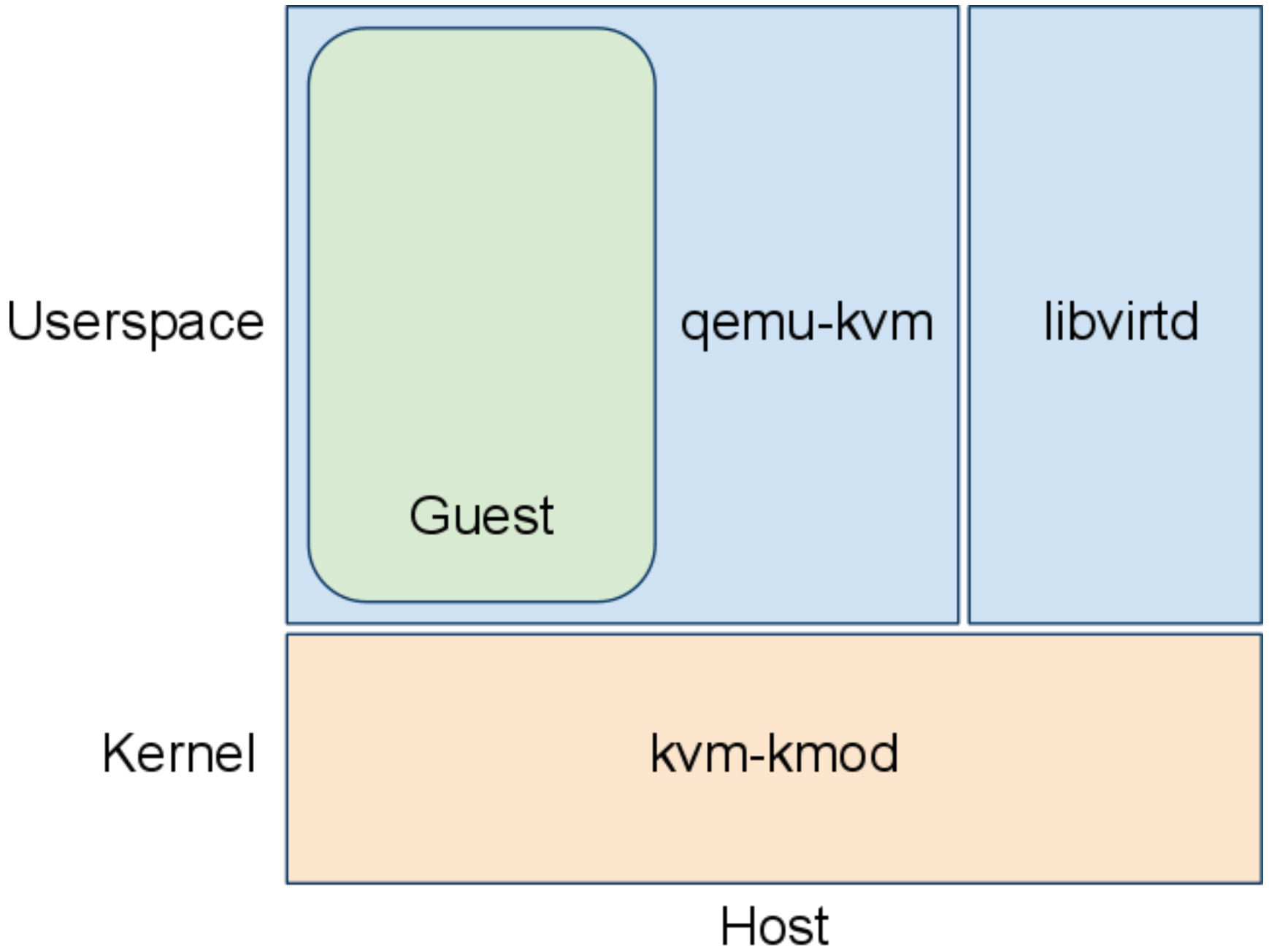
Use cases

- Server consolidation
- Virtual desktop infrastructure
- Compute cloud
- Embedded
- End-user virtualization

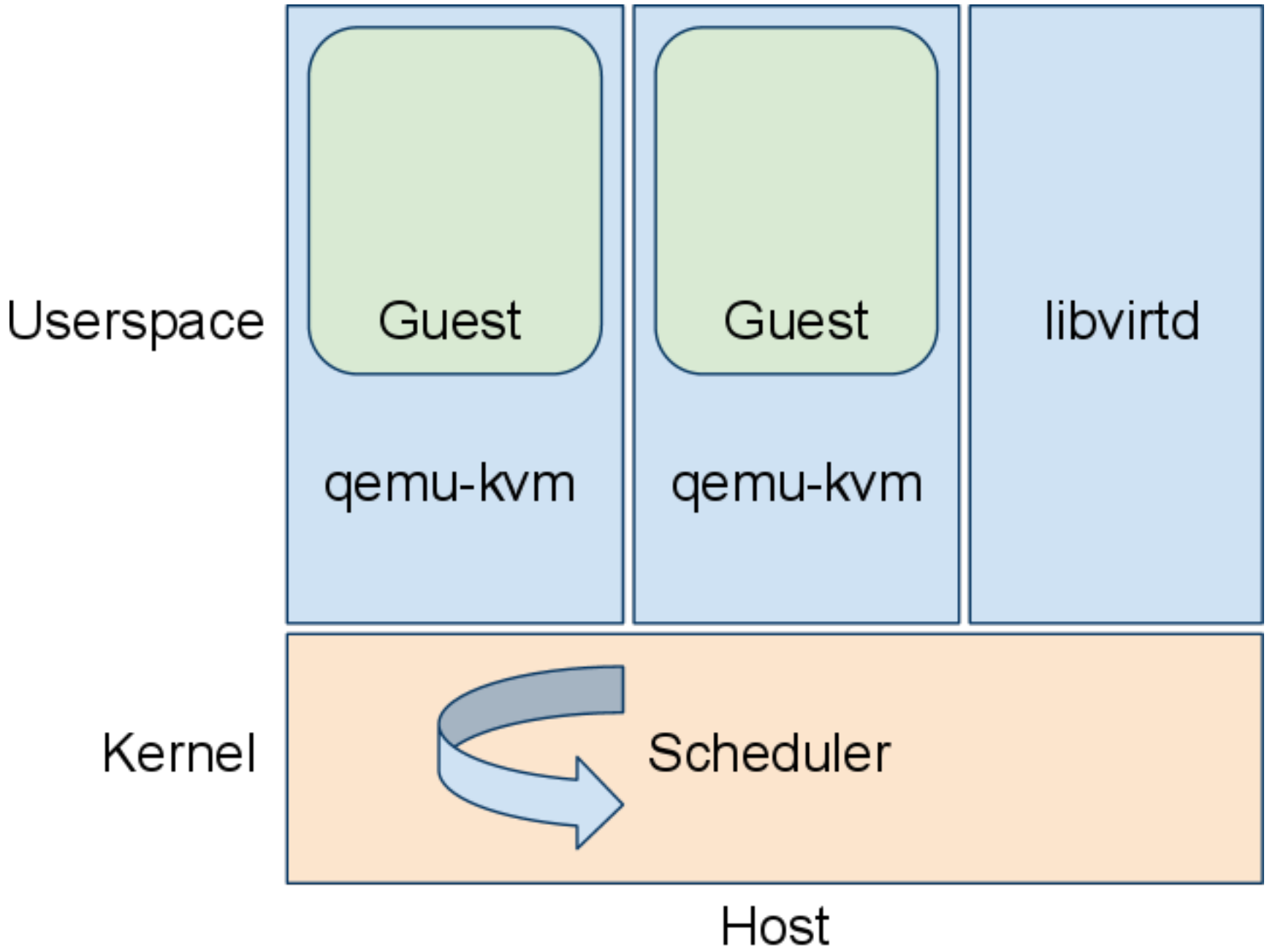
Features

- i386 and x86_64 UP and SMP guests
- Runs Linux, Windows, and many other OSes
- PCI pass-through
- Optional paravirtualized I/O
- Live migration including block migration
- Snapshot save/resume
- Guest swapping and memory dedup (KSM)

KVM is not monolithic



Linux scheduler and resource controls



KVM kernel module

http://linux-kvm.org/

Part of mainline Linux kernel tree.

Provides common interface for Intel VMX and AMD SVM hardware assist.

Contains emulation for instructions and CPU modes not supported by hardware assist.

Handles performance critical parts of timers and interrupts via in-kernel I/O emulation.

qemu-kvm userspace

http://linux-kvm.org/

Usually shipped as "kvm" or "qemu-kvm" package.

Command-line program to run a VM.

Responsibilities:

1. Set up VM and I/O devices
2. Execute guest code via KVM kernel module
3. I/O emulation and live migration

libvirt management stack

http://libvirt.org/

User-visible tool virsh does VM management.

Virtualization API for Xen, KVM, VMware ESX, others.

Each host runs libvirtd to manage VMs, storage, and networking.

Provides secure remote management.

Architecture internals and examples

Internals:

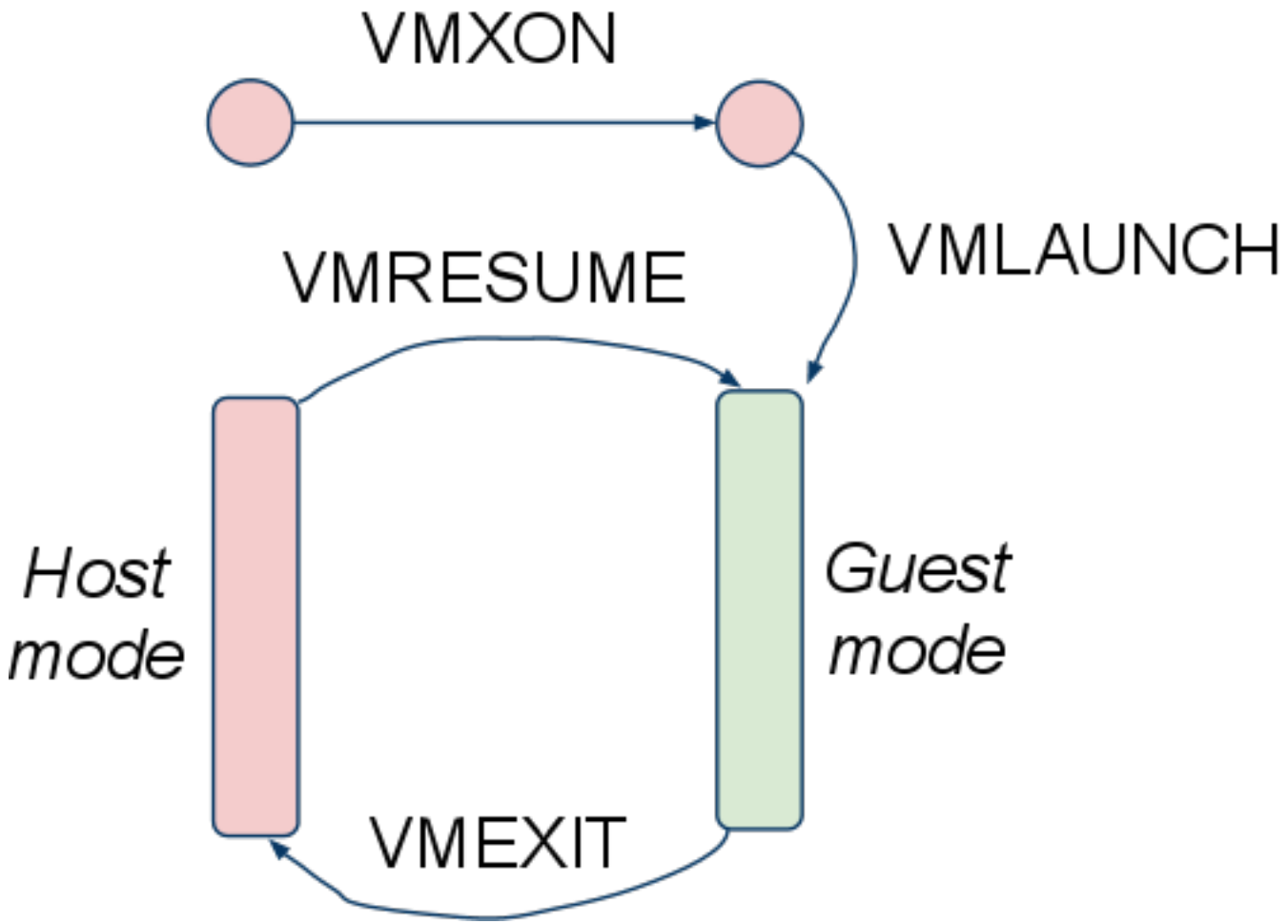
- How is the CPU virtualized?
- How is memory virtualized?
- How does qemu-kvm use the kvm kernel module?

Examples:

- qemu-kvm command-line
- libvirt XML file
- virt-manager GUI

CPU virtualization with VMX (Intel)

Enables real Virtual Machine Monitors (VMM), reduces need for binary translation and emulation.



Memory virtualization with EPT (Intel)

Extended Page Tables add a level of address translation for guest physical memory.

EPT supports read/write/execute access bits, can be used to generate faults for physical memory accesses.

Avoid TLB flushes with virtual processor ID tagged address spaces.

KVM kernel module usage

The qemu-kvm process does something like this:

```
open("/dev/kvm")
ioctl(KVM_CREATE_VM)
ioctl(KVM_CREATE_VCPU)
for (;;) {
     ioctl(KVM_RUN)
     switch (exit_reason) {
     case KVM_EXIT_IO:  /* ... */
     case KVM_EXIT_HLT: /* ... */
     }
}
```

qemu-kvm command-line example

```
/usr/bin/kvm -M pc-0.12 -m 1024
    -smp 1,sockets=1,cores=1,threads=1
    -mon chardev=monitor,mode=readline
    -drive file=fedora13.img,if=none,id=drive-
virtio-disk0,boot=on,format=raw
    -device virtio-blk-pci,bus=pci.0,addr=0x4,
drive=drive-virtio-disk0,id=virtio-disk0
    -device virtio-net-pci,vlan=0,id=net0,
     mac=52:54:00:f5:7a:c9,bus=pci.0,addr=0x5
    -net tap,fd=46,vlan=0,name=hostnet0
```

This is not the complete line, I have shortened it...

libvirt XML file example

```
<domain type='kvm'>
  <name>fedora13</name>
  <uuid>6170299a-f9b6-41f0-7eda-
d3feadc8d5f1</uuid>
  <memory>1048576</memory>
  <currentMemory>1048576</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='x86_64' machine='pc-
0.12'>hvm</type>
    <boot dev='hd'/>
  </os>
```

...but luckily GUIs like virt-manager handle the XML!

virt-manager VM configuration