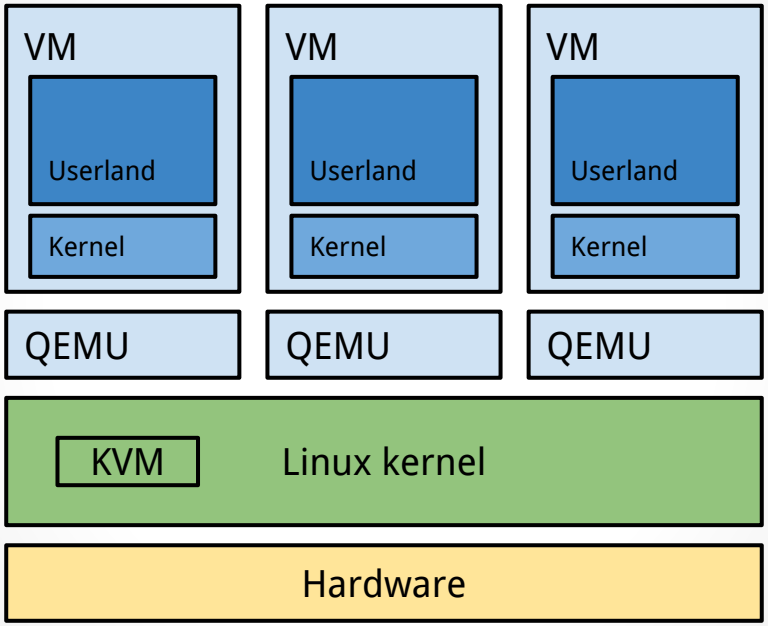
x86: Device Virtualization

Gabriel Laskar <gabriel@lse.epita.fr>

LSE Security System

Hardware Virtualization



HW Virtualization : QEMU/KVM

I/O Virtualization

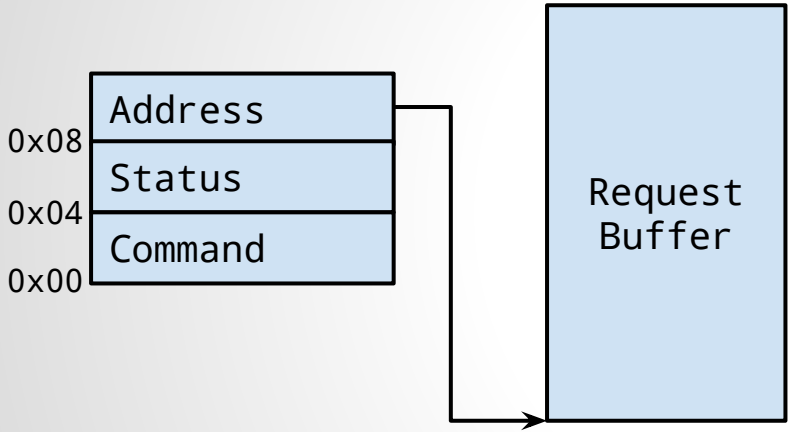
- HW emulation
- Paravirtualized Devices
  - Xen
  - Virtio
  - Other (vmxnet, synthetic devices, …)
- Hardware Pass through
  - Full Device (pci, vga)
  - Protocol (usb, serial)
  - Other way

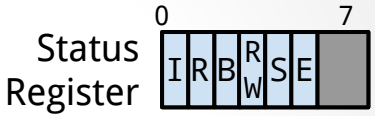
LSE Security System

Devices

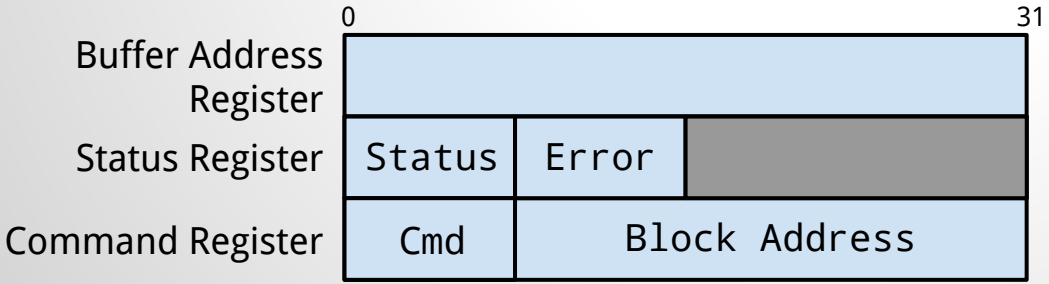
- Registers accessible to CPU :
  - MMIO
  - PIO (in, out)
- Access to Memory (DMA)
- Interrupts (irq, msi)

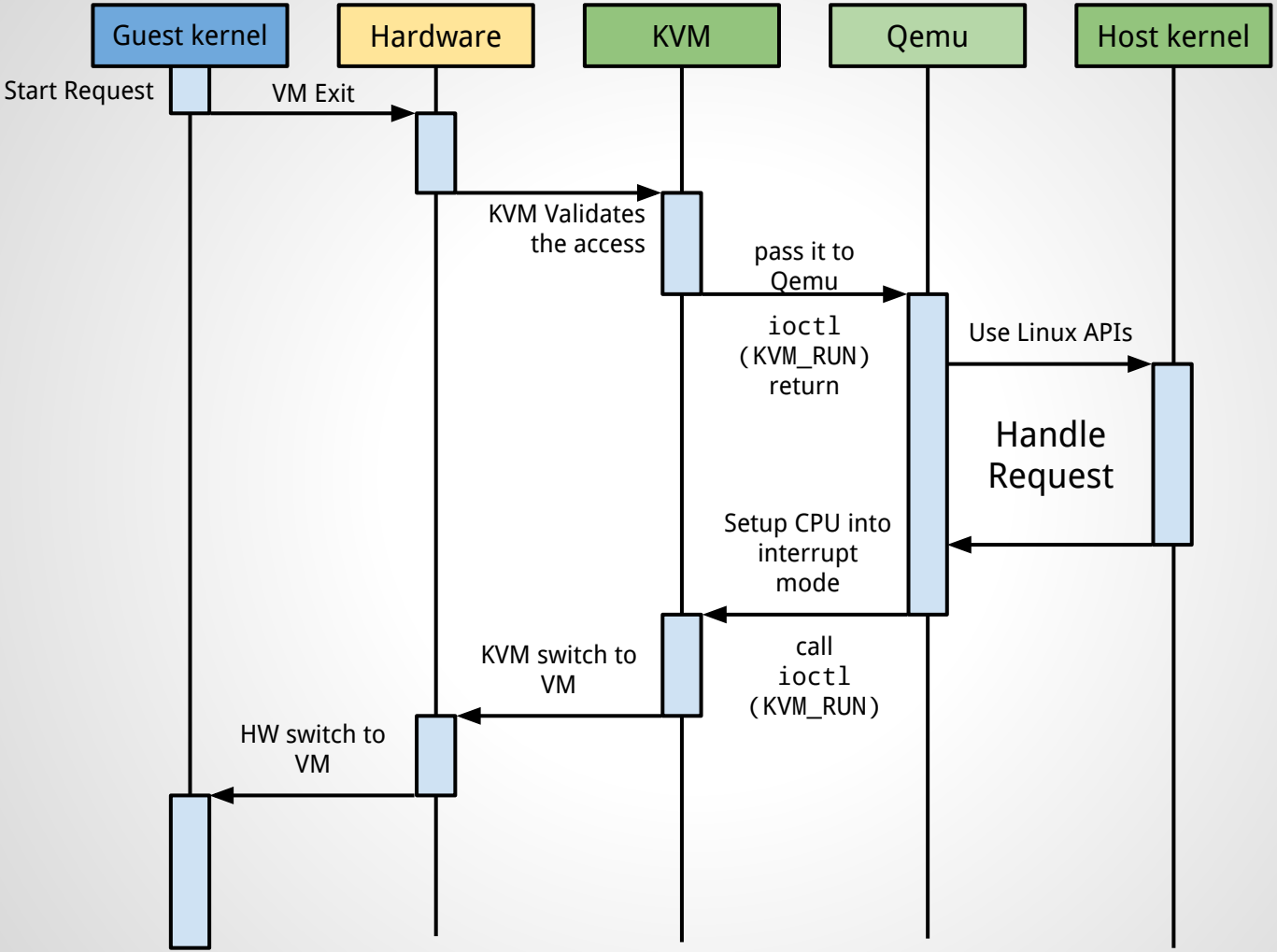
Example of a simple device: a block device





I : Initialize device
R : Ready to receive command
B : Busy
RW : Read/Write (clear if read, set if write)
S : Start Command
E : Error

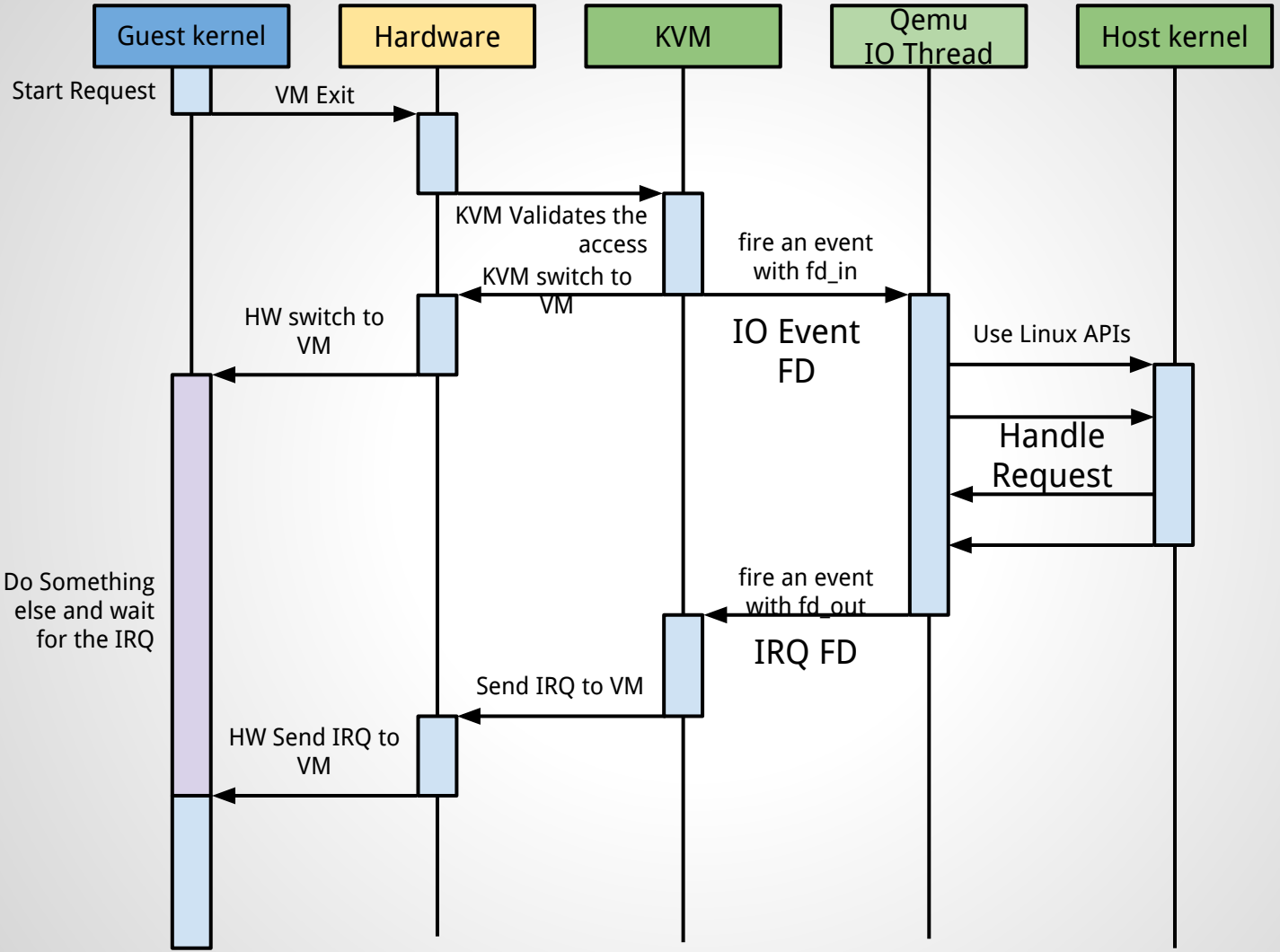



Guest kernel
Hardware
KVM
Qemu
Host kernel
Start Request
VM Exit
KVM Validates the access
pass it to Qemu
ioctl (KVM_RUN) return
Use Linux APIs
Handle Request
Setup CPU into interrupt mode
call ioctl (KVM_RUN)
KVM switch to VM
HW switch to VM

MMIO, PIO : How fast ?

- For each mmio access, there is an exit
- We have to assert the read/write, and process the command
- Can’t be asynchronous, ie : we can’t do that with the vcpu guest running
- What solutions do we have ?

LSE Security System


Guest kernel
Hardware
KVM
Qemu IO Thread
Host kernel
Start Request
VM Exit
KVM Validates the access
KVM switch to VM
fire an event with fd_in
IO Event FD
HW switch to VM
Use Linux APIs
Handle Request
Do Something else and wait for the IRQ
fire an event with fd_out
IRQ FD
Send IRQ to VM
HW Send IRQ to VM

How can we solve the IO Problem ?

- KVM_IOEVENTFD
  - Attach an ioeventfd to a pio/mmio guest address
  - When guest write into this address, it fire an event instead of an exit
- KVM_IRQFD
  - Allow setting an eventfd that will trigger a guest interrupt
- With eventfd and irqfd, we can offload io traffic into another thread, and just listen/fire event through fds.

LSE Security System

Example : handling device

```
void handle_device(void *device, int eventfd, int irqfd)
{
     struct pollfd input_queue = {
          .fd = eventfd,
          .events = POLLIN;
     };

     for (;;) {
          int ret = poll(input_queue, 1, timeout);

          if (ret > 0) {
               uint64_t event_value;
               read(eventfd, &event_value, sizeof(event_value));
               uint64_t res = do_something(device, event_value);
               write(irqfd, &res, sizeof(res));
          }
     }
}
```

Virtio

- Abstraction layer for virtualized devices
  - multiple device types: PCI, MMIO
  - Split in 2 parts: Configuration Space, Queues
- Multiple Devices supported:
  - block
  - network
  - channels (for serial line, 9p, or custom data channel)
  - scsi devices
  - RNG
  - Ballooning
  - GPU and Input devices (not completely merged yet)

LSE Security System

Direct Pass-Through: VFIO, VT-D and IOMMU

- In order to pass-through a device, we must have some kind of IOMMU support
- VFIO is the linux API allowing to use it
- VFIO allows to use PCI (or other) devices in userland

Notes for the project

- Create a VM that can boot stos without `stos.grub`
  - Boot directly into PM mode
  - Load the kernel/stos:
    - kernel/stos
    - initramfs
    - commandline
    - modules
    - memory map
  - pass the boot param structure
  - Devices:
    - 8250
    - 8259a
    - 8254
- All your code should be working in your VM

LSE Security System