



KVM PV DEVICES

dor.laor@qumranet.com

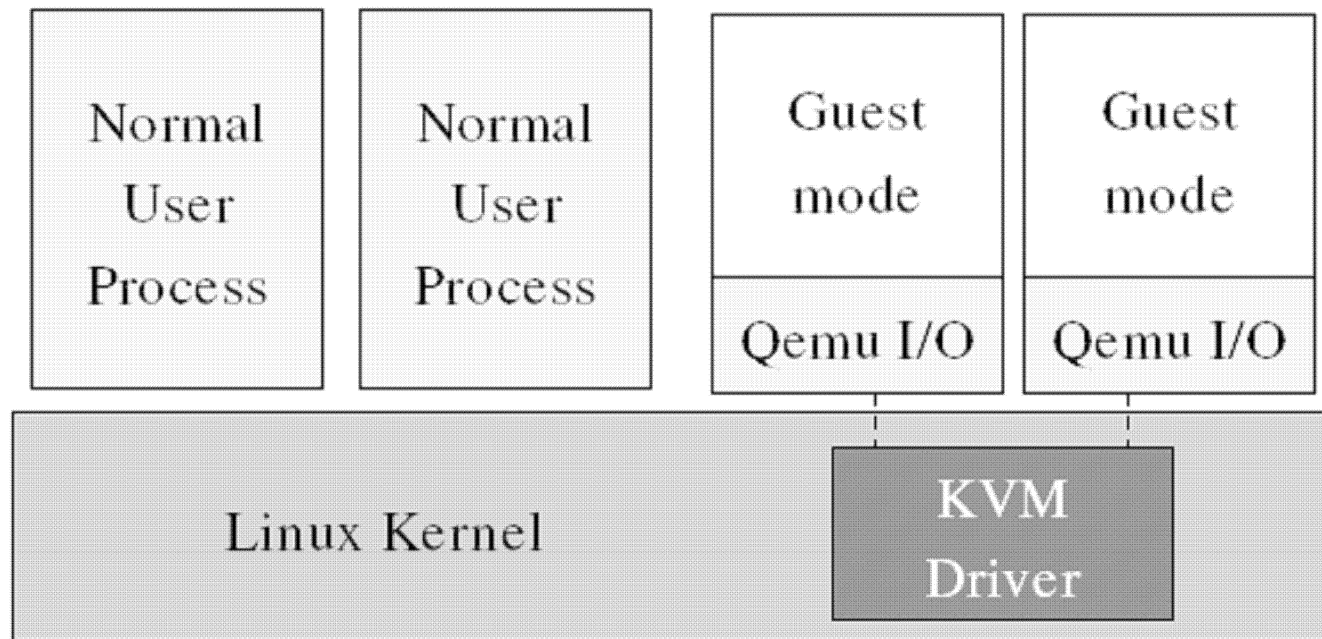


Agenda

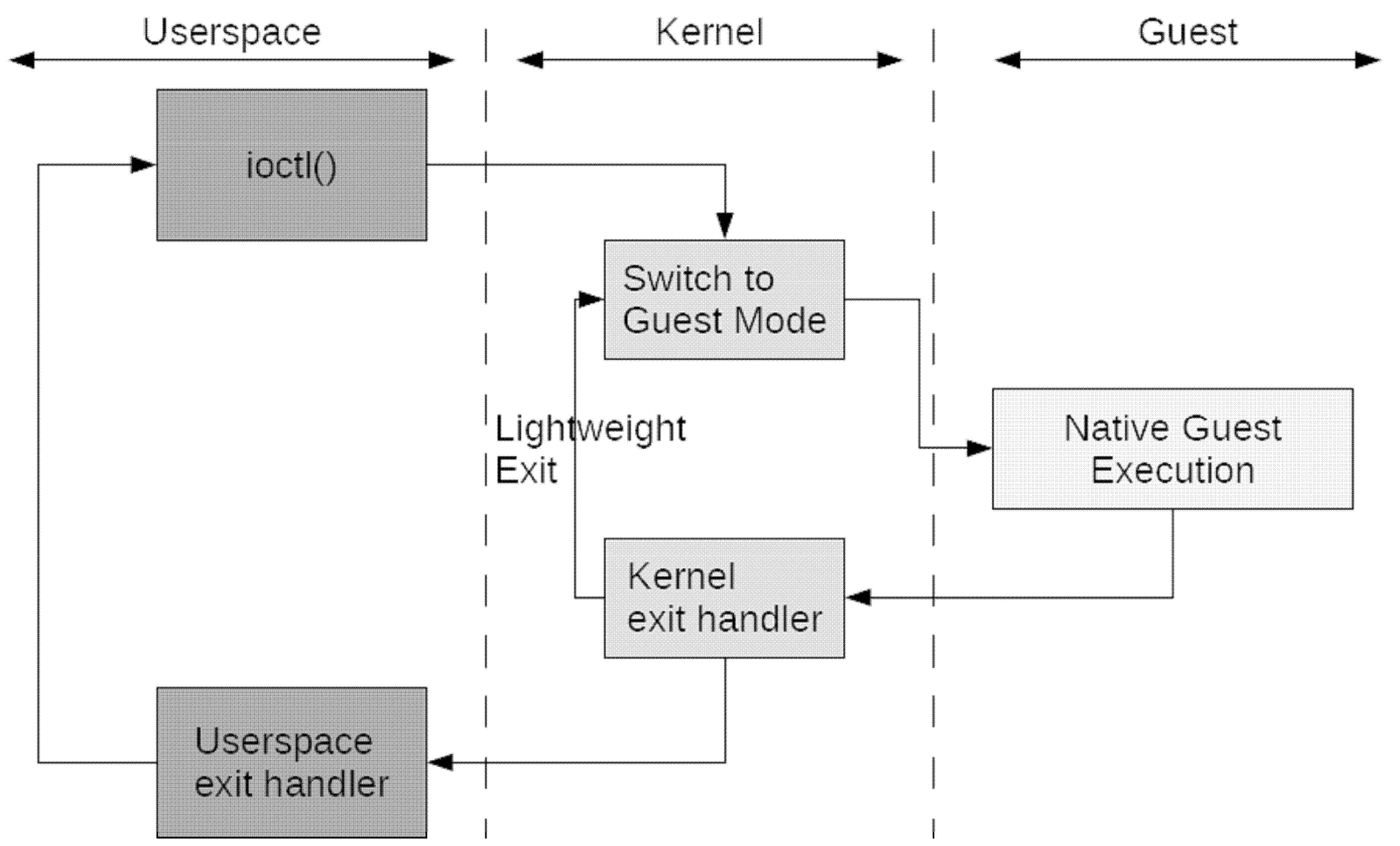
- ◆ Kernel Virtual Machine overview
- ◆ Paravirtualized drivers intro & brief history
- ◆ VirtIO
- ▣ Enhanced VirtIO with KVM support

Kernel Virtual Machine overview

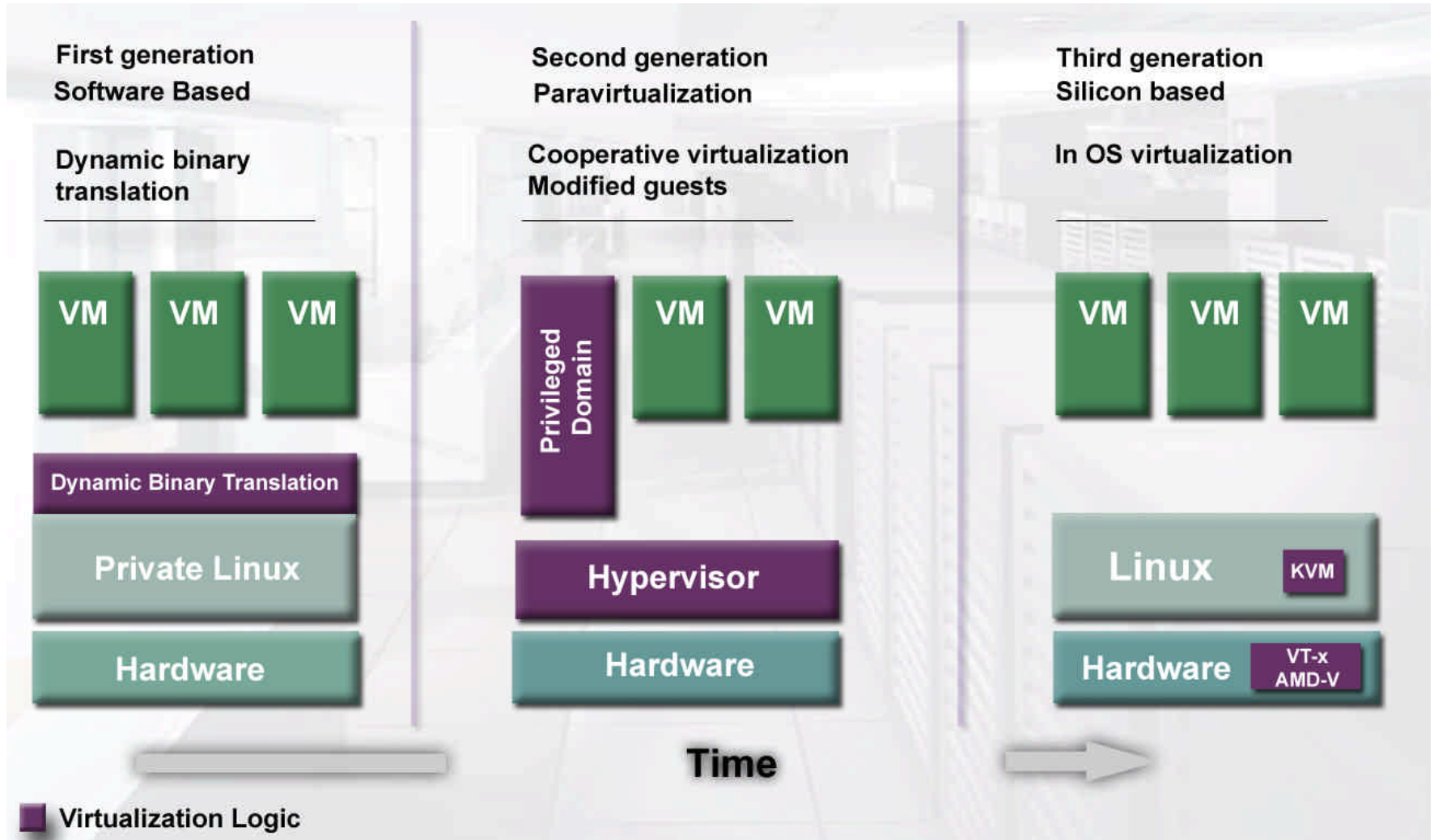
- VM is a regular Linux process.
- KVM handles cpu & memory virtualization
- Qemu virtualizes IO devices



KVM Execution Model



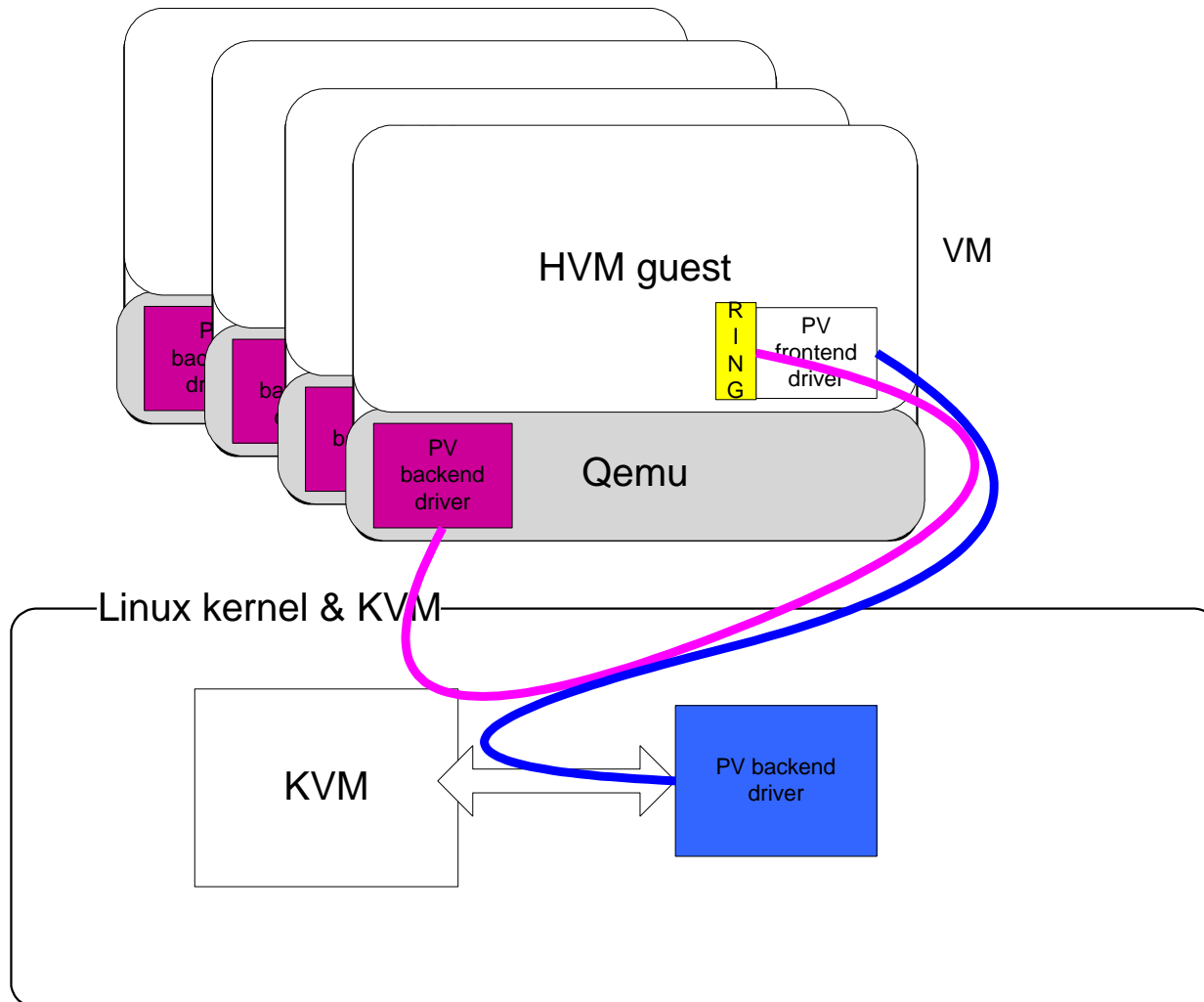
Evolution of x86 virtualization



General & history

- ♦ Fully virtualized devices performs bad
 - ♦ 55 Mbps for RTL
 - ♦ Lots of io-exits per packet
- ♦ Decided to implement a modern e1000
 - ♦ Advantage:
 - ♦ Only Qemu coding
 - ♦ no guest tools involved
 - ♦ Irq coalescing
 - ♦ Only 2-3 io-exits per packet
 - ♦ Can be the base of user-space PV
- ♦ But then came Ingo...

PV driver architecture



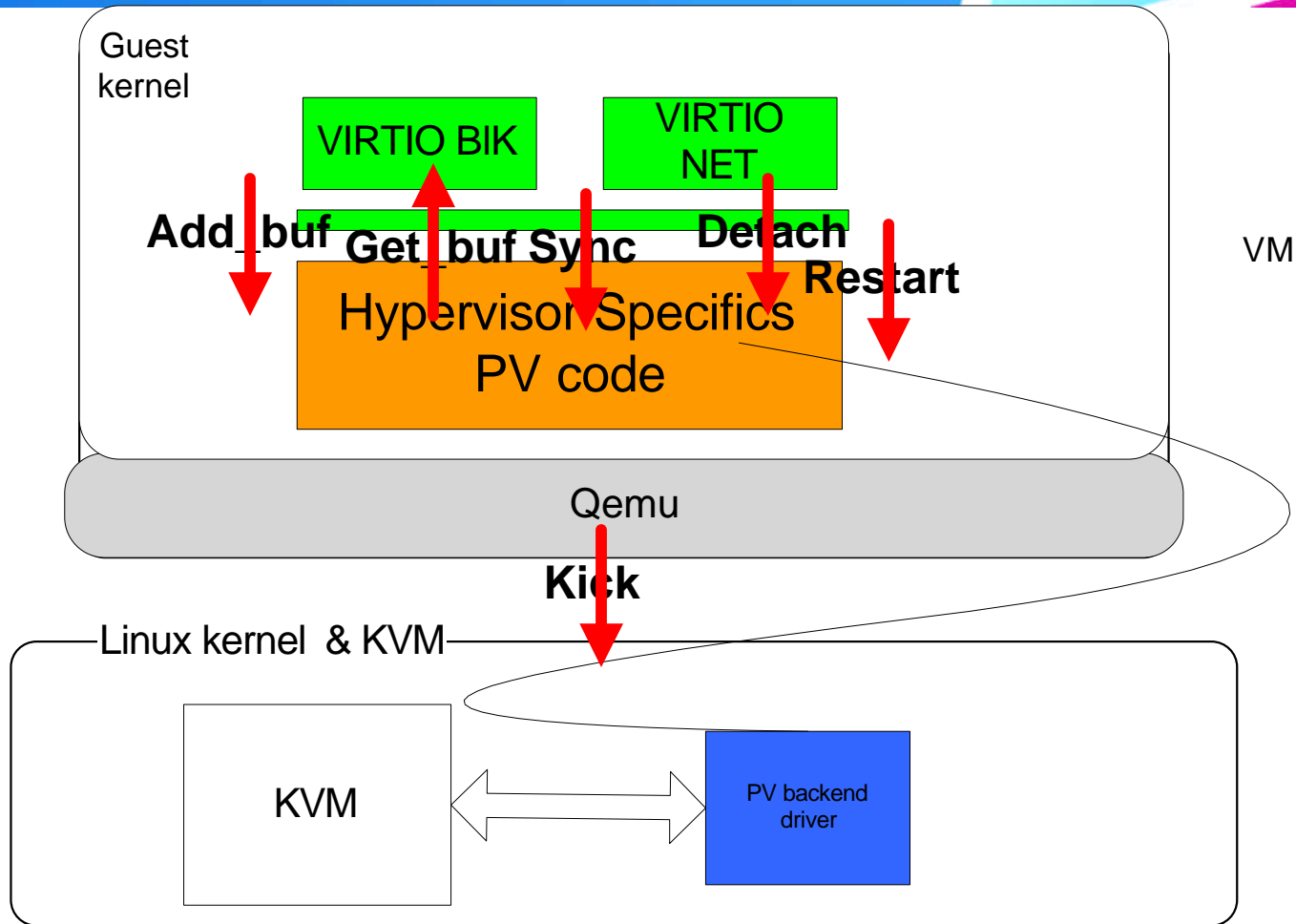
General & history

- ♦ V0 - leveraging Ingo Molnar's PV code
 - ♦ Make loadable module
 - ♦ Add HVM support
 - ♦ Add NAPI
 - ♦ Add memory barriers and improved ring
 - ♦ Keep running after performance & stability
 - ♦ Merge to the kernel?
- ♦ Alternatives
 - ♦ Xen
 - ♦ Polished drivers
 - ♦ Xen specific
 - ♦ VirtIO was just published.

VirtIO

- ♦ An API for virtual I/O
 - ♦ Implements network & block driver logic
 - ♦ Written by Rusty Russell
- ♦ Motivation
 - ♦ Many hypervisors of all types
 - ♦ Hard to tune and maintain each one
 - ♦ Code reuse – The KVM way ;)
- ♦ Implementations
 - ♦ Lguest
 - ♦ KVM
 - ♦ Possible (Xen, UML, Qemu, VMware?..)

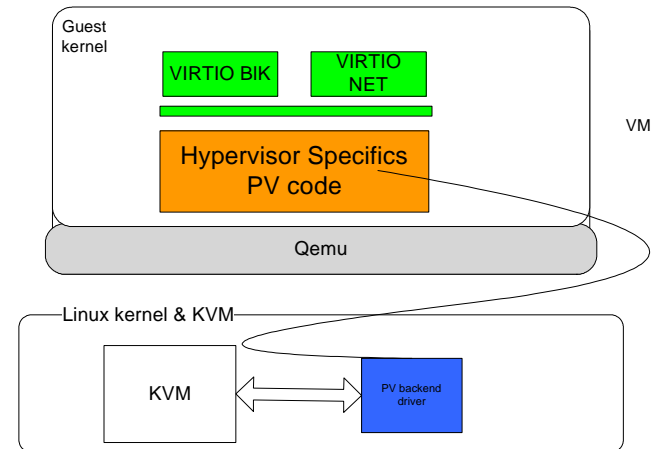
VirtIO



- ◆ scatterlist sg[] for skb/blk_req data

VirtIO – Hypervisor specifics

- ♦ The front end logic is implemented by VirtIO
- ♦ The backend needs
 - ♦ Probing & Bus services
 - ♦ Enumeration
 - ♦ Irq
 - ♦ Parameters (mac,..)
 - ♦ Shared memory with remote side
 - ♦ Hypercalls
 - ♦ Host driver/userspace backend



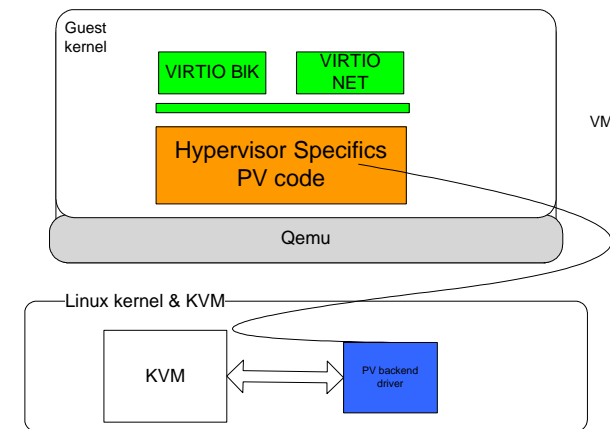
Enhanced VirtIO – Reuse the reuser

♦ Motivation

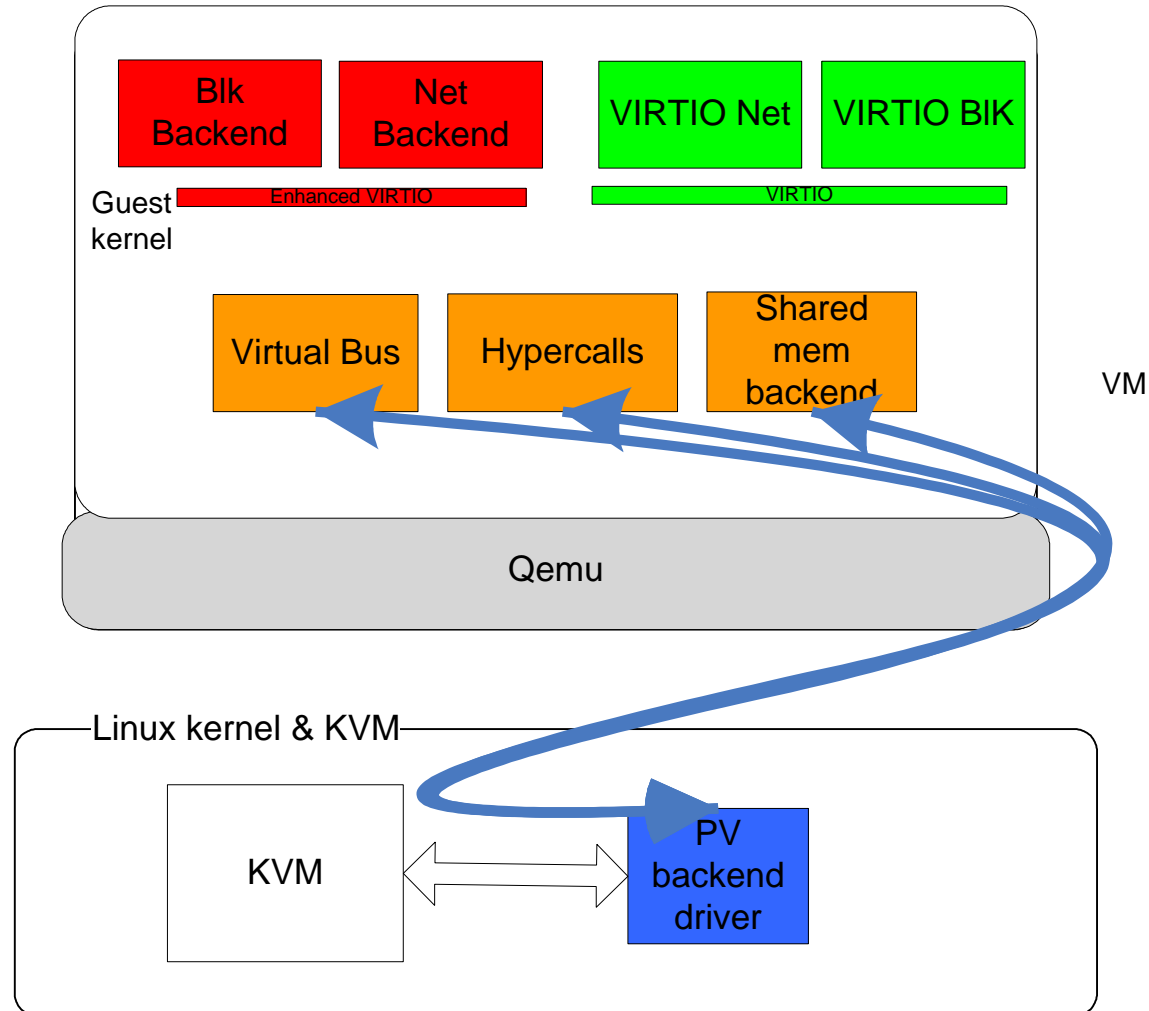
- ♦ Increase re-use
- ♦ Allow operation with various bus types
- ♦ Make new devices code smallest

♦ Components

- ♦ Shared memory code
 - ♦ With per hypervisor I/O hypercalls
- ♦ Bus (pci, virtual bus)
- ♦ Host backend



Enhanced VirtIO



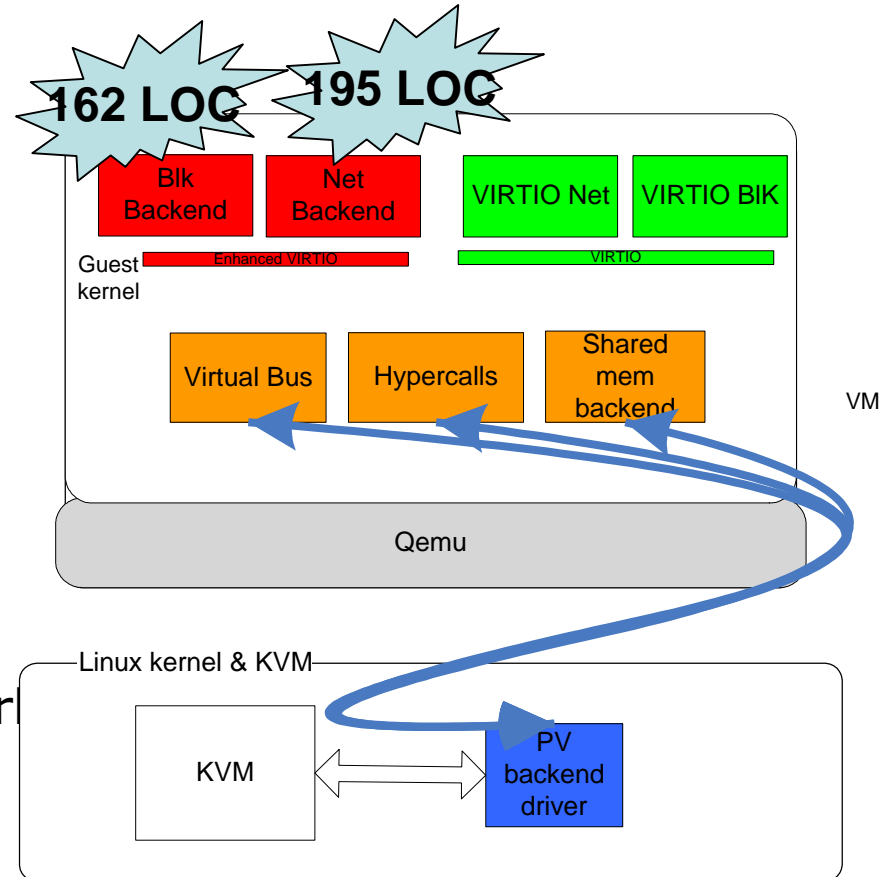
Enhanced VirtIO

- ◆ Status:

- ◆ Interface was enhanced
- ◆ KVM support
- ◆ PCI like configuration space added
- ◆ Actually we got re-used...

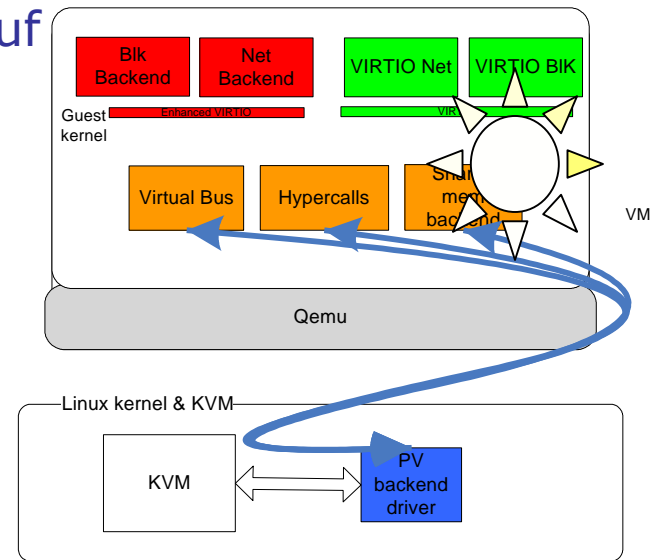
- ◆ Result:

- ◆ Makes backend driver tiny
- ◆ 620 Mbps throughput for network
 - ◆ HVM Linux guest
 - ◆ Before optimization
 - ◆ Userspace backend driver

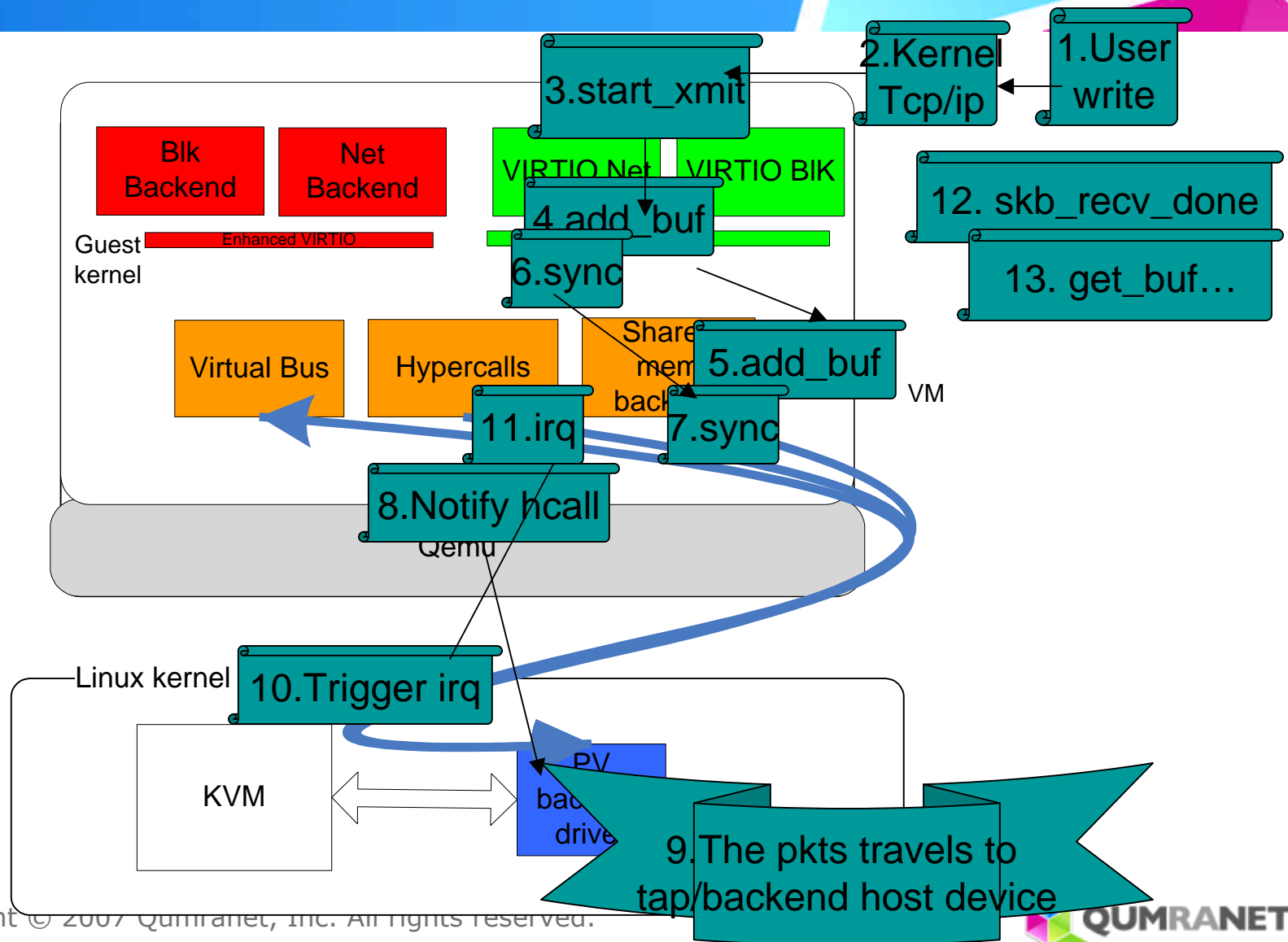


Enhanced VirtIO – shared memory backend

- ♦ VirtIO backend
 - ♦ Implements VirtIO interface
 - ♦ Callbacks to hypervisor and
- ♦ Code consists of
 - ♦ `add_buf`, `get_buf`, `restart`, `detach_buf`
 - ♦ Only shared memory logic needed
 - ♦ `sync`
 - ♦ Ring logic
 - ♦ **IO pending hypercall**
 - ♦ `be_virtqueue_interrupt` handler

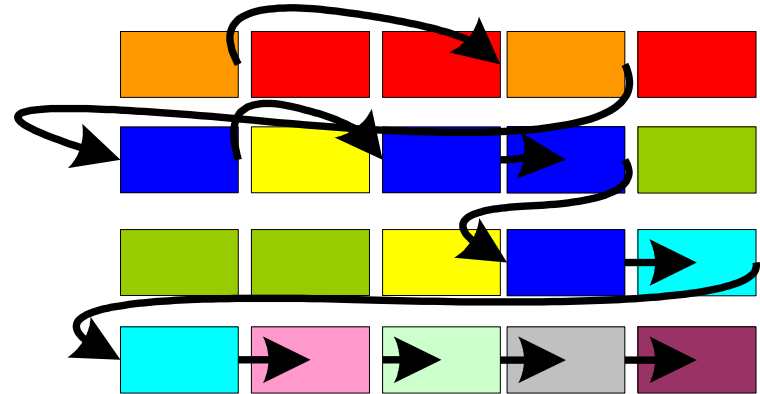


Enhanced VirtIO – Day in a life of packet



Enhanced VirtIO – shared memory details

- ◆ Based on Iguest
- ◆ 1-1 shared memory
- ◆ Data structure
 - ◆ Page of **descriptors** for rx, tx.
 - ◆ **Available** pointers page controlled by guest
 - ◆ **Used** pointers page controlled by host
- ◆ SG list is currently internal to descriptors
 - ◆ Descriptors are chained by next pointer



Enhanced VirtIO – network backend driver

- ◆ Implements `kvm_virtnet_probe` for pci bus

- ◆ Creates tx,rx `be_new_virtqueue`

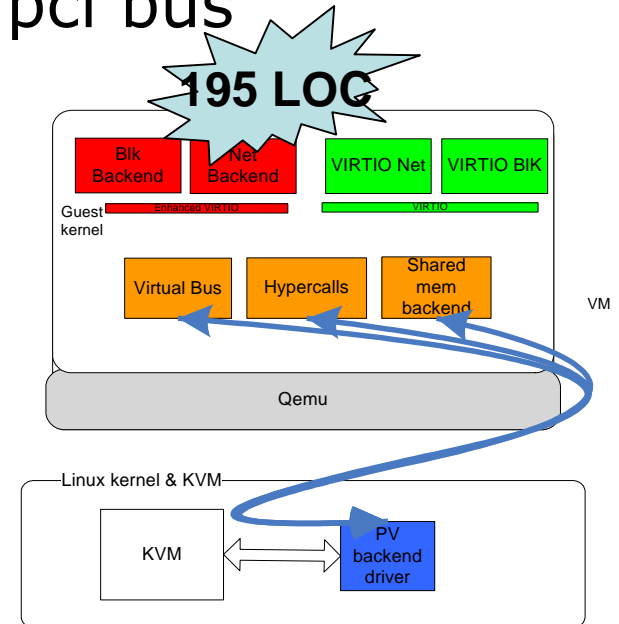
- ◆ **Probes** virtnet

- ◆ **Request_irq**

- ◆ `Irq#` taken from bus

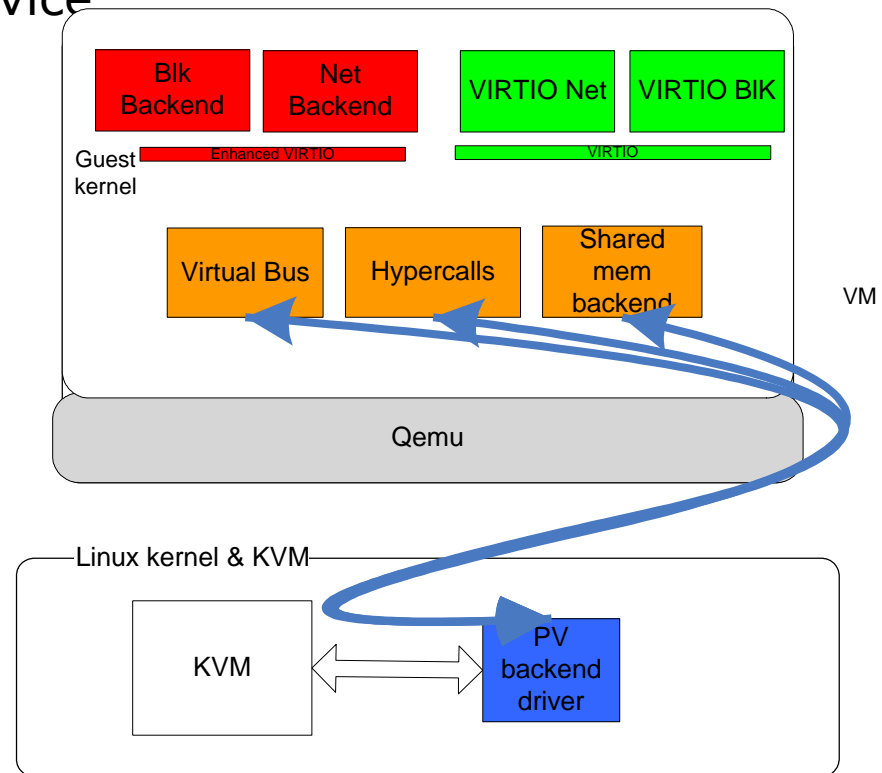
- ◆ **Register hypercall** - shared memory pfn

- ◆ Device key for enumeration taken from bus



Further work

- ♦ Basic
 - ♦ Add readv/writev handlers to Qemu
 - ♦ Complete the user-space block device
 - ♦ Complete migration support
- ♦ Advanced (also simple)
 - ♦ Optimize and stabilize
 - ♦ Add host back end drivers
 - ♦ Add virtual bus
 - ♦ HVM improvements
 - ♦ Test with PV kernel



Thank you ;)