Multi-process QEMU

Marc-Andre Lureau
Senior Software Engineer, Red Hat, Inc.

Konrad Rzeszutek Wilk
Software Director, Oracle

October 27 2017

Presented with

redhat

Slim down QEMU

Konrad Rzeszutek Wilk

Software Director, Oracle, Inc.

Marc-Andre Lureau

Senior Software Engineer, Red Hat, Inc.

ORACLE

Program Agenda

1. QEMU in virtualization
2. Xen usage of QEMU
3. KVM usage of QEMU
4. Security!

QEMU usage

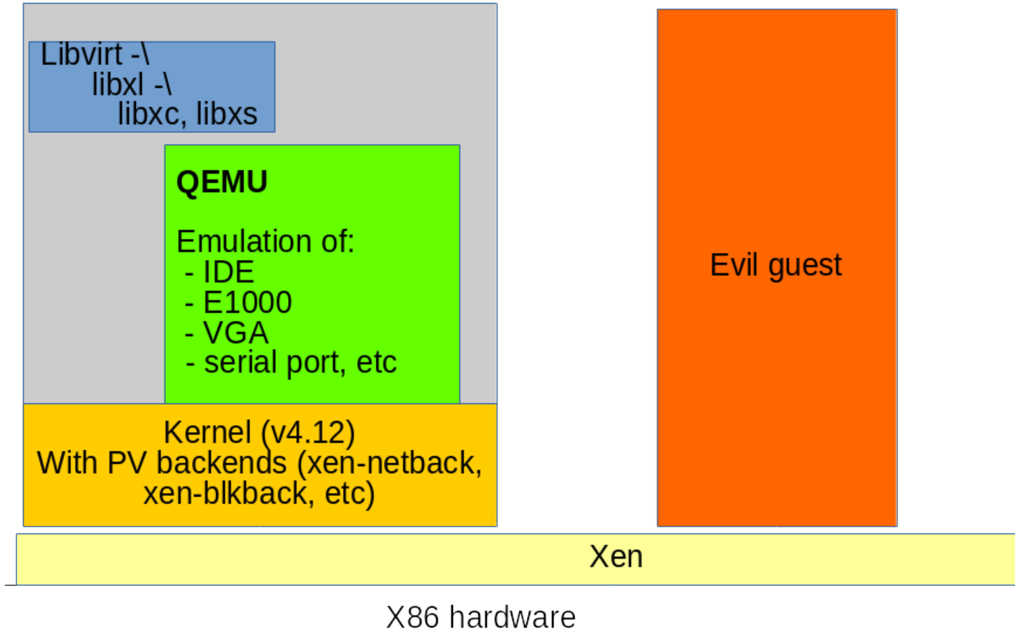
- Both KVM and Xen use QEMU emulation (IDE, e1000)
- None use the binary translation in QEMU.
  - Xen and KVM in the hypervisor code base deal with opcodes:
    - movdqa m128,xmm
    - (traped on MMIO access)
- KVM uses QEMU as control stack (launch/destroy guest) as in privileged operations (access to /dev/kvm).
- Xen uses **only** QEMU emulation (which is why you can't launch guests with QEMU parameters and need to use libvirt or xl).

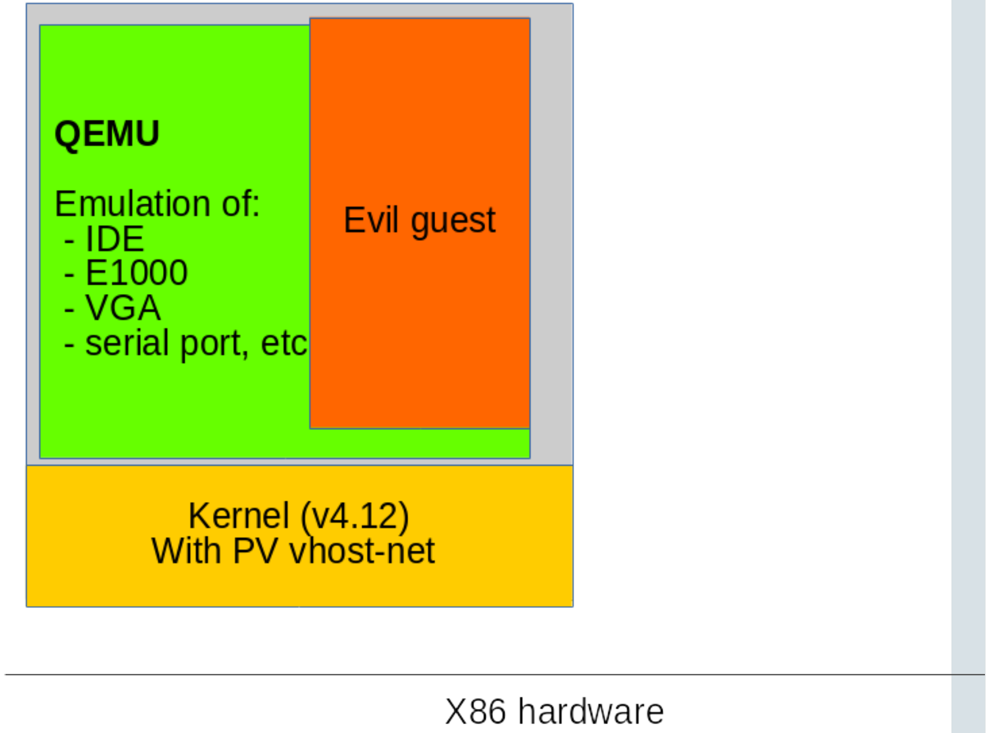
Evil guest attack vectors

- Cloud provides have to deal with risk of customers becoming evil.
- The “customers” have usually four primary attack vectors:
  - Emulation (VENOM – CVE-2015-3456) of floppy driver, VGA, NICs, etc in QEMU.
  - MSRs (x2APIC range gap – CVE-2014-7188) of x2APIC emulation in hypervisor.
  - VMCALL (hypercalls to hypervisor – CVE-2012-3497).
  - Opcode emulations (INVEPT instructions – CVE-2015-0418).
- This talk is about the first: **QEMU** and ways to lessen the impact if it is exploited, or alternatively erect more “jails” around QEMU.

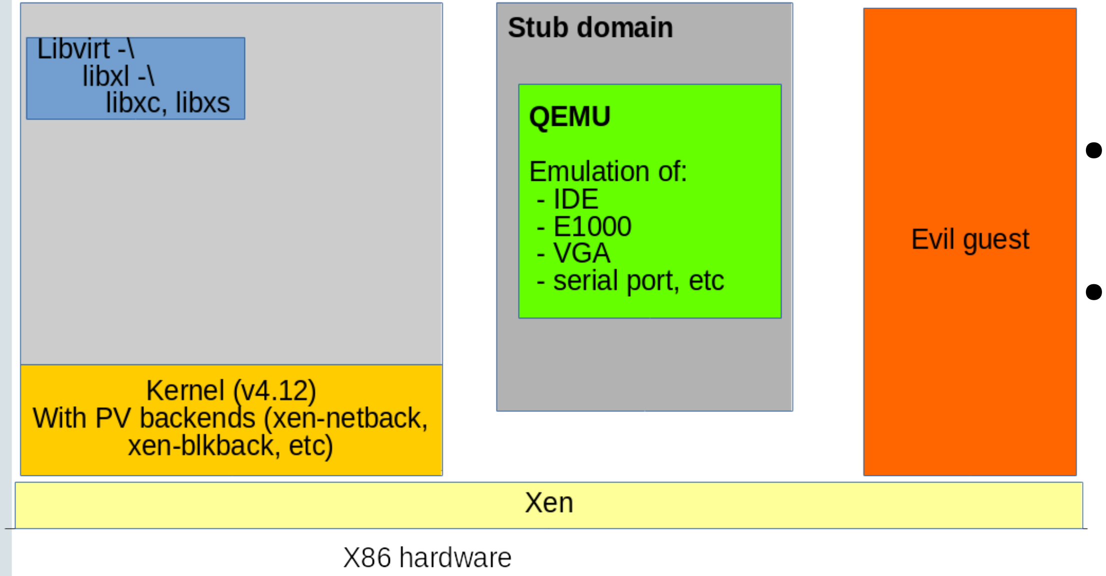
ORACLE

Xen and KVM architecture (usual)



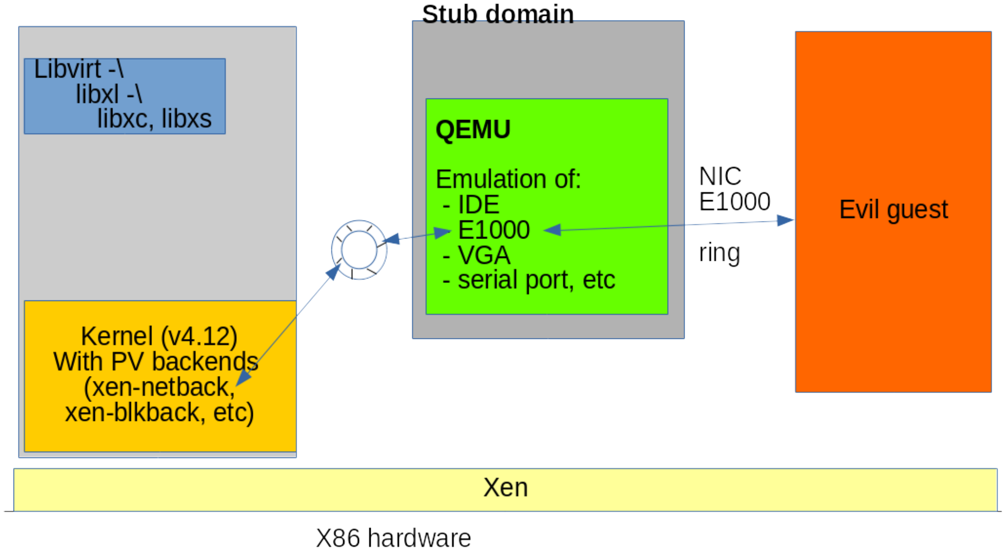


Xen disaggregated architecture



- Move QEMU to be a standalone guest running in ring0 (32MB guest).
- Each stubdomain serves **one** guest.
- Evil guest has to subvert stub domain emulation first, then from there jump to control domain.

Xen disaggregated architecture (network)

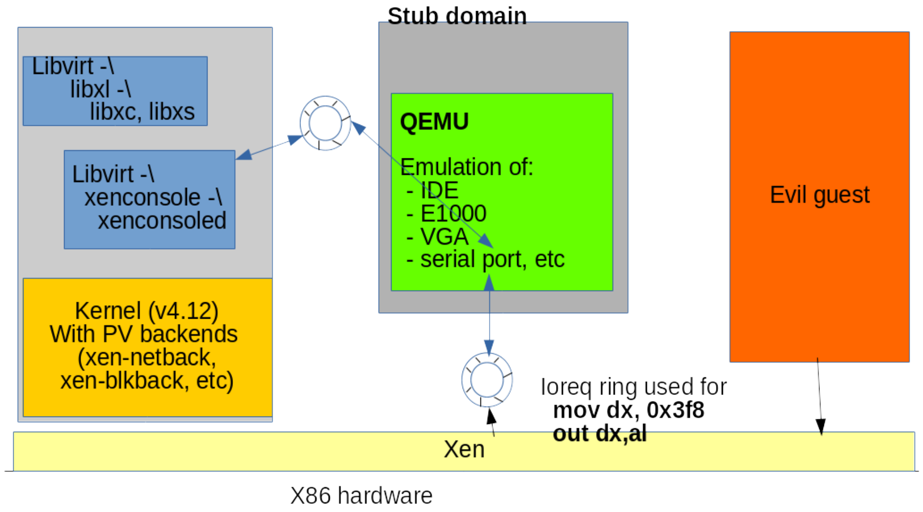


- Evil guest uses e1000 for attack.
- QEMU uses PV frontend driver to send packets to **real** backend
- If evil guest subverts stub domain the next attack is the PV protocol
- CVE-2015-8550: double fetch:

  "Specifically the shared memory between the frontend and backend can be fetched twice (during which time the frontend can alter the contents) possibly leading to arbitrary code execution in backend.
- But protocol **MUCH simpler** than emulated devices.

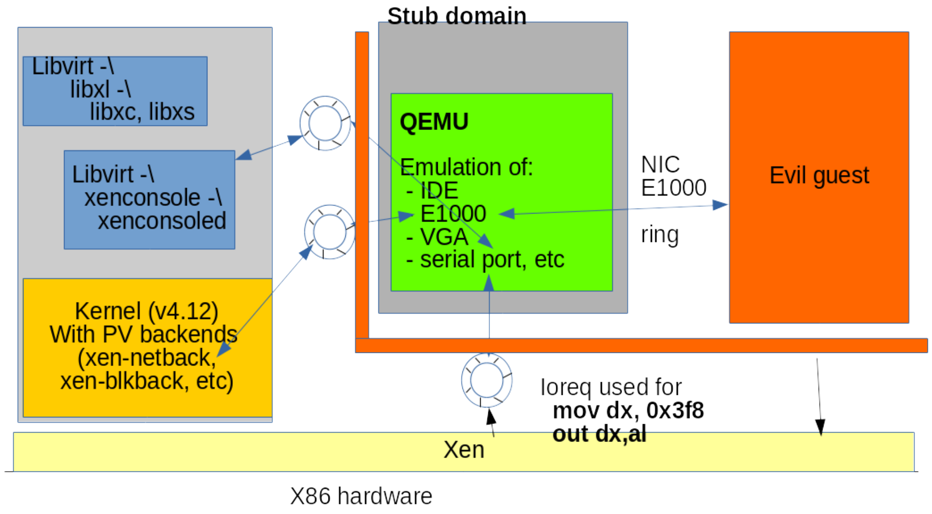
ORACLE

Xen disaggregated architecture (serial)



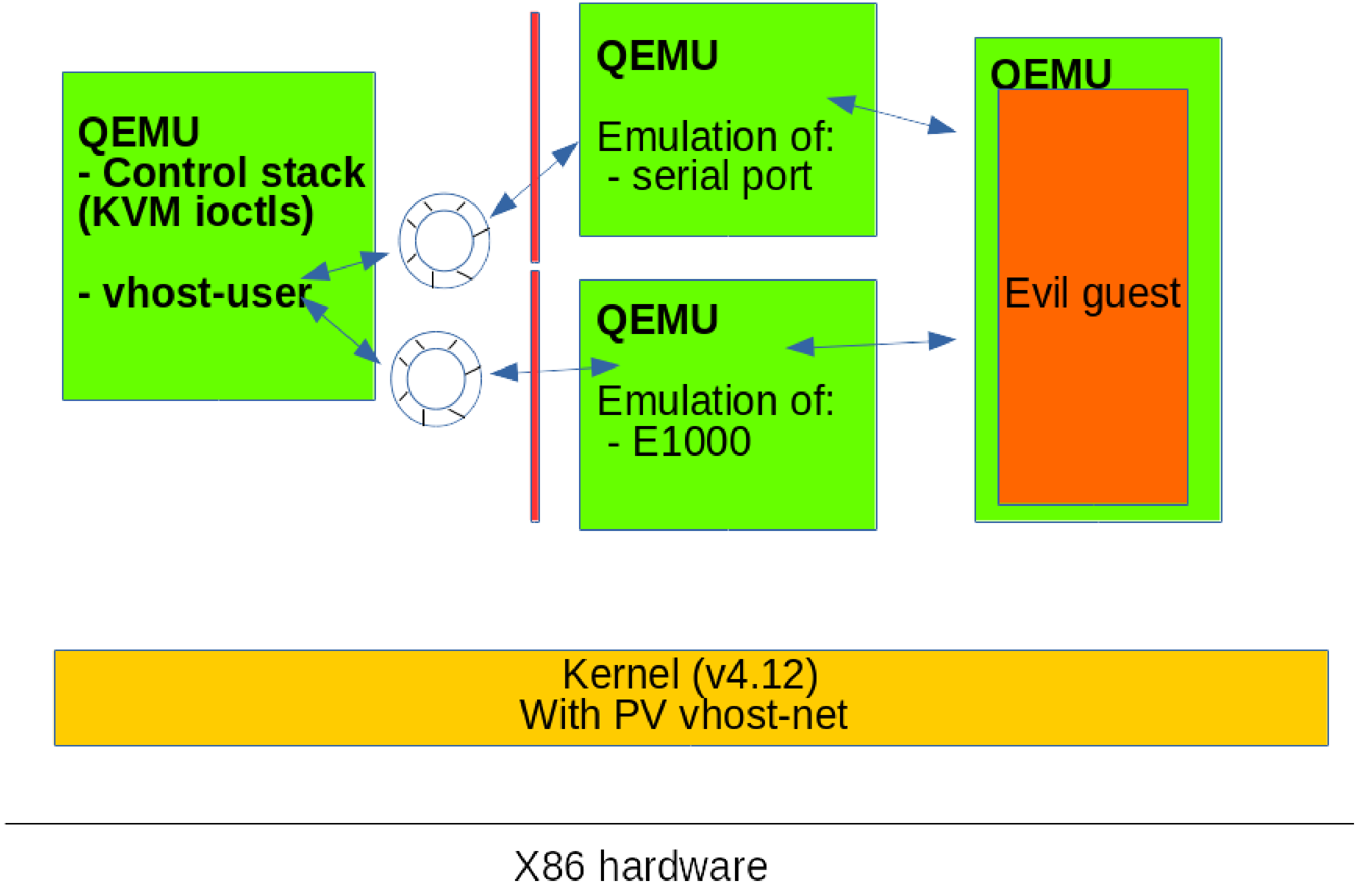
- Privilege opcodes (out/in) always end up in hypervisor.
- A ring between hypervisor and QEMU for device model to process.
- QEMU and xenstored have a PV ring to copy data back/forth.

Xen disaggregated architecture: jail around QEMU



- In effect the barrier between QEMU and control stack is via the PV ring.
- If evil guest exploits stub domain they are the same place as before.
- Attacks left then are via:
  - MSRs
  - Hypervisor hypercalls
  - Opcode emulation
  - (But this presentation is not about those attacks).

Can we do something similar in KVM?



ORACLE®

Can we do something similar in KVM? Is it needed?

- QEMU is used for emulation **and** control stack.
  - If we disaggregate QEMU we can move each component in its own process.
- We have security measures in place:
  - secomp & ebpf (filter the ioctls to /dev/kvm)
  - Containers (chroot jails)
  - Continuing work on improving QEMU security
- Sure, but separating components apart (each running in its own jail) means we can focus security audit on the high-stake parts
- OK, how do we do this?

Integrated Cloud

Applications & Platform Services

ORACLE®

Multi-process QEMU

Marc-Andre Lureau
Senior Software Engineer, Red Hat, Inc.

Konrad Rzeszutek Wilk
Software Director, Oracle

October 27 2017

Multi-threading QEMU

or Ingo might be right.. sort of

Anthony Liguori – aliguori@us.ibm.com

IBM Linux Technology Center

Aug 2010

- **Motivations for QEMU**
- **Requirements for devices**
  - KVM features
- **Various QEMU solutions**
- **Conclusion & QA**

A big binary

```
elmarco@boraha:~$ ls -lhS /bin/ | head -n20
-rwxr-xr-x.   1 root root       33M Aug 16 16:00 dockerd-current
-rwxr-xr-x.   1 root root       17M Sep 15 00:46 emacs-25.3
-rwxr-xr-x.   1 root root       16M Sep  7 16:32 node
-rwxr-xr-x.   1 root root       15M Jun 26 11:51 ocamlopt.byte
-rwxr-xr-x.   1 root root       15M Jul  4 15:33 doxygen
-rwxr-xr-x.   1 root root       13M Aug 16 16:00 docker-current
-rwxr-xr-x.   1 root root       12M Sep  8 21:59 qemu-system-aarch64
-rwxr-xr-x.   1 root root       12M Sep  8 21:59 qemu-system-arm
-rwxr-xr-x.   1 root root       12M Jun 26 11:51 ocaml
-rwxr-xr-x.   1 root root       11M Sep  8 21:59 qemu-system-x86_64
-rwxr-xr-x.   1 root root       11M Sep  8 21:59 qemu-system-i386
-rwxr-xr-x.   1 root root       11M Jun 26 11:51 ocamlc.byte
-rwxr-xr-x.   1 root root       11M Sep  8 21:59 qemu-system-mips64el
-rwxr-xr-x.   1 root root       11M Sep  8 21:59 qemu-system-mips64
-rwxr-xr-x.   1 root root       11M Sep  8 21:59 qemu-system-mipsel
-rwxr-xr-x.   1 root root       11M Sep  8 21:59 qemu-system-mips
-rwxr-xr-x.   1 root root      7.1M Apr 25 17:44 crash
-rwxr-xr-x.   1 root root      6.9M Jun 26 11:51 ocamldoc.opt
-rwxr-xr-x.   1 root root      6.4M Jun 26 11:51 ocamlopt.opt
```

A big project

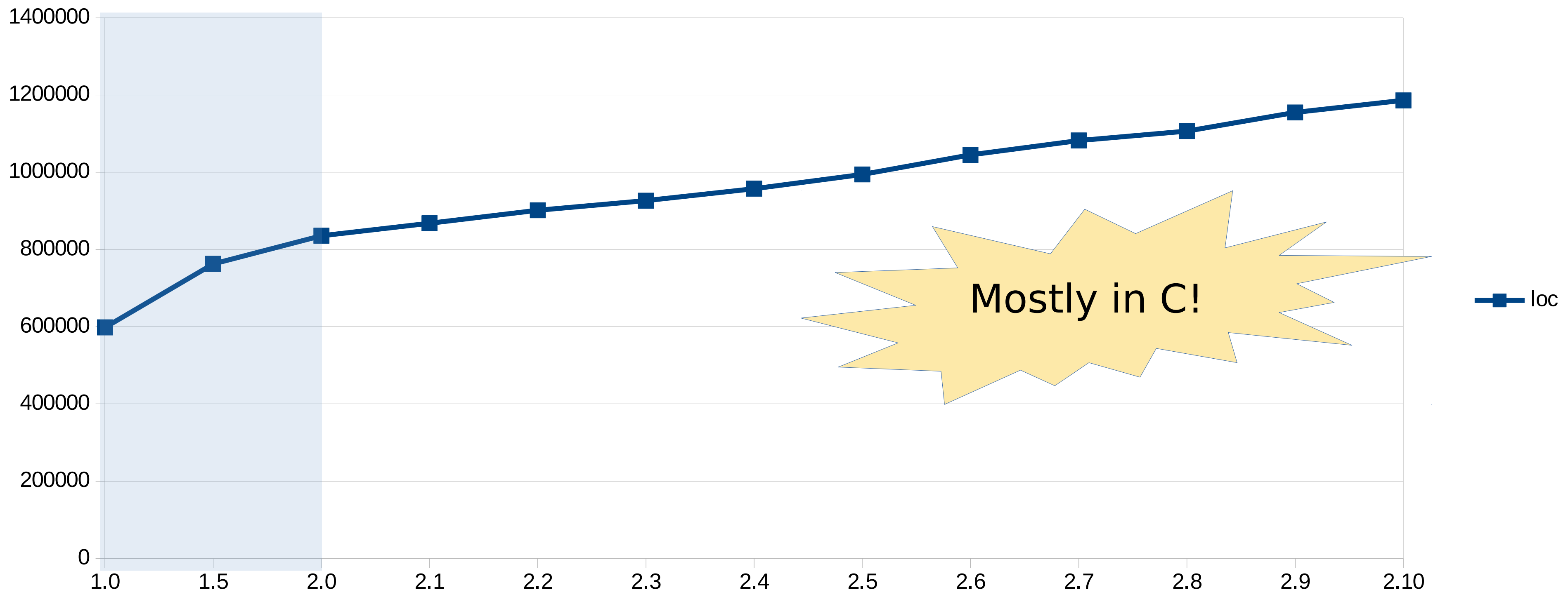
```
$ cloc qemu-2.10
- files: 4 280
- comment: 172 425
- code: 1 186 140

$ cloc kvmtool
- files: 275
- comment: 3 728
- code: 27 844
$ cloc crosvm
- code: 32 159
```

```
$ cloc linux
- files: 49 744
- code: 16 834 046
```

How much with all dependencies?

Still growing



Many dependencies

- Fedora 26: qemu 2.9.0-5.fc26.x86_64

```
$ readelf -d /usr/bin/qemu-system-x86_64  | grep NEEDED | wc -l
60

$ ldd /usr/bin/qemu-system-x86_64 | wc -l
158
```

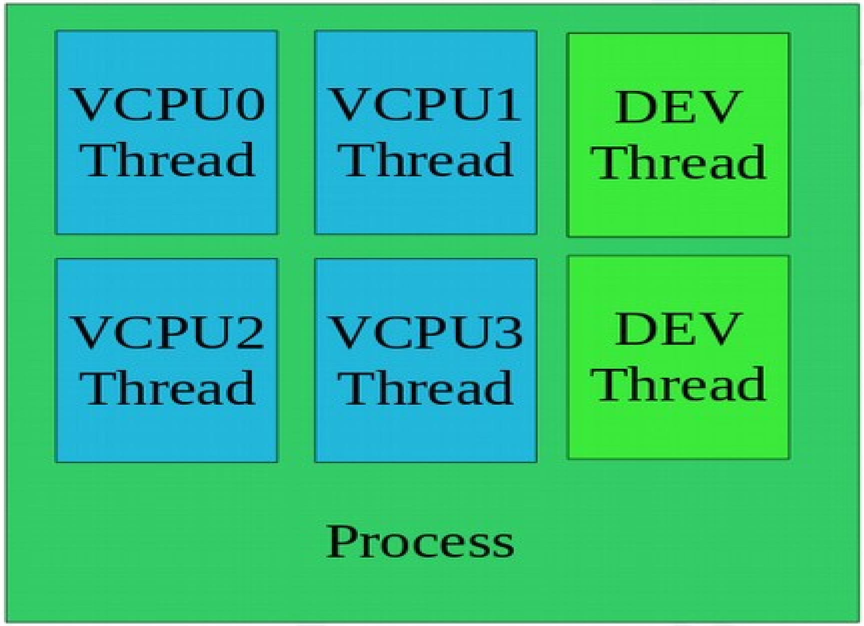
- Kvmtool (with all optional dependencies, gtk3, SDL, vncserver...)

```
$ readelf -d lkvm | grep NEEDED | wc -l
19

$ ldd lkvm | wc -l
83
```


Too big to fail

Ideal KVM Architecture



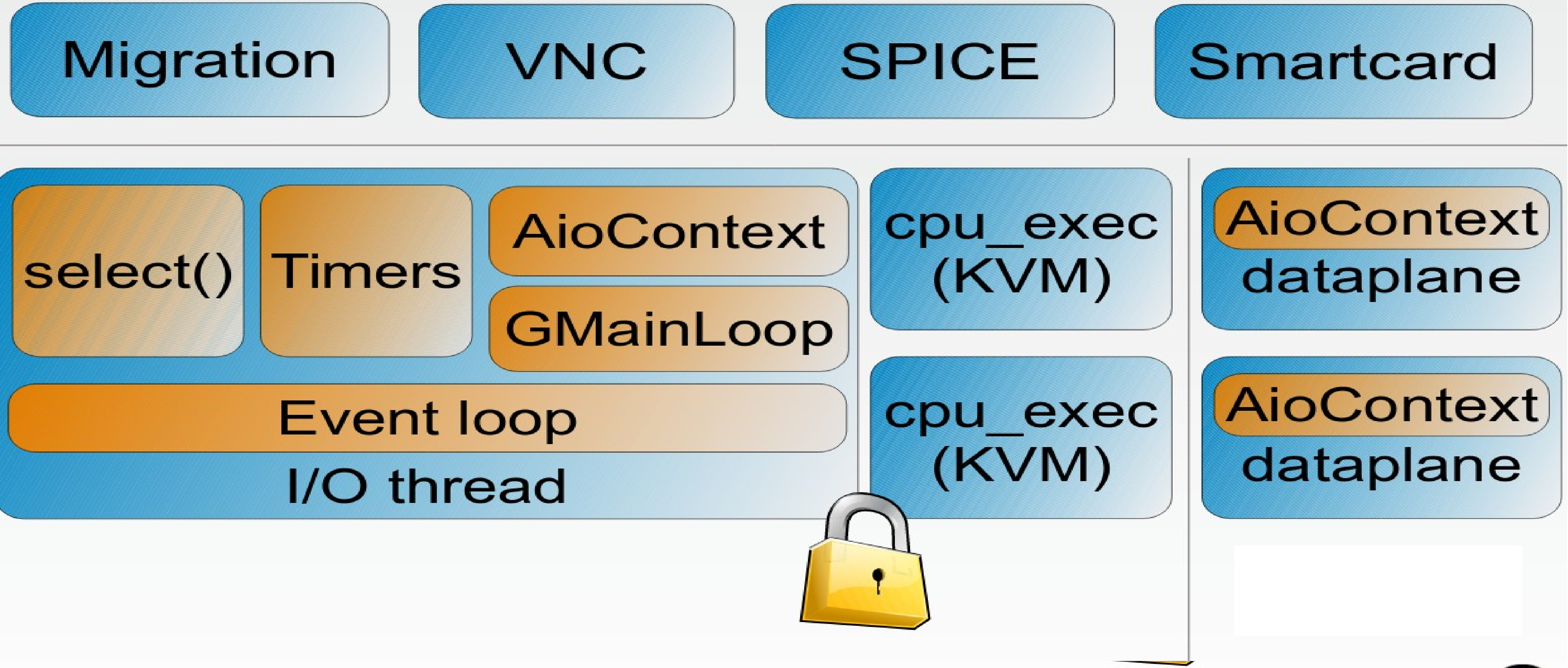
Design

- One thread per-VCPU
- Device models run concurrent in VCPU thread
- Long running operations run in additional device thread

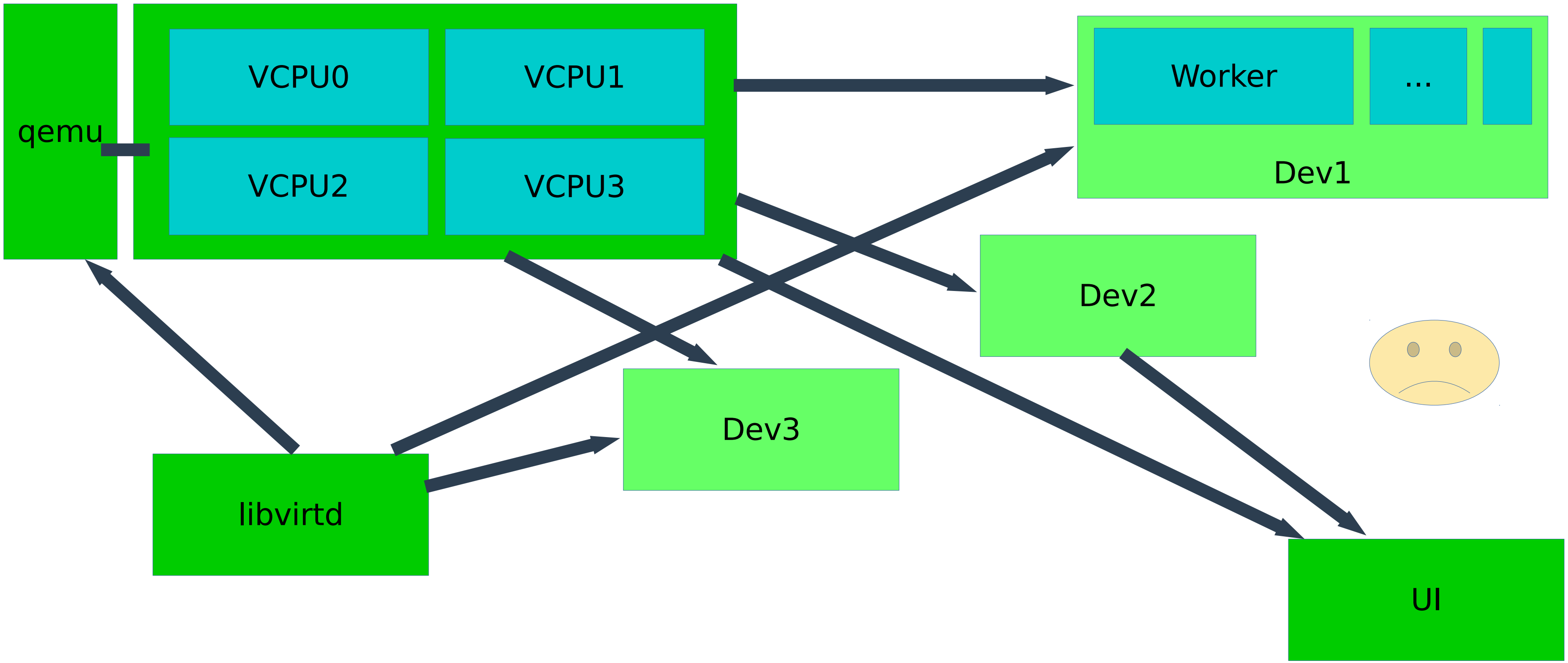
Goals

- Maximize CPU affinity
- Minimize PIO/MMIO latency

QEMU architecture (now)



Ideal architecture ?



Why not?

- The monolithic vs microkernel/services debate
- Difficult to manage
- Difficult to debug
- Difficult to test (test matrix)
- Performance?

Why seperate processes?

- **Modularity**
  - clear interface separation = less conflicts/bql concerns
  - smaller qemu, less dependencies
  - allowing alternative implementations, “crazy” ideas
  - separate projects, different release cycles…
- **Isolation (+iommu) & crash robustness**
- **Better sandboxing (seccomp/ns)**
- **Easier monitoring/tweaking (memory, cpu etc)**

Sandboxing for dummies

Change user id

Regular DAC/MAC check

Add/drop capabilites(7)

Subset of root privileges (if needed)

Namespaces(7)

Own view/access of the system (uid/pid/ns/net/ipc..)

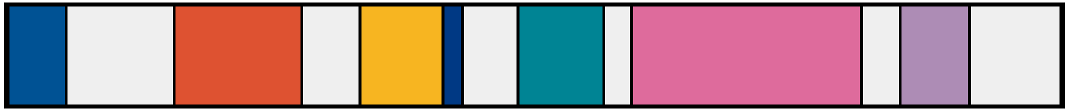
Seccomp()/bpf

Filter syscalls

Libvirt, minijail, systemd, flatpack...

A word about memory fragmentation

All devices & workloads in a single process can lead to more fragmentation.



Using subprocesses may help to partition the load and more easily reclaim the space.

How? various strategies

- **Fork-only strategy (crosvm)**
  - Code in same binary
  - No version combinations, less modularity
  - Device setup and teardown can be hardcoded in parent

- **Exec a helper or device process**
  - Can allow arbitrary implementations
  - IPC require greater level of stability
  - Nicer if IPC allows various kind of devices

Managing the processes

- **Qemu**
  - Not a great idea to fork from qemu (VM space, safety)
  - Slirp & migration can do it…
  - Could exec() from an helper process instead?

- **Outside, libvirt or other:**
  - Not suitable for command line users
  - Natural fit for libvirt etc

How? various device needs

- **HW description & bus registration**
- **Communication mechanism:**
  - Io / Mmio regions & rw events, Irqs
  - Memory map (& iommu)
  - Or at higher level of abstraction (USB etc)
- **acpi / device-tree manipulation (& fw_cfg)**
- **Device state & migration**
- **Dirty regions tracking, post-copy...**
- **Object hierarchy / introspection**

KVM <-> device emulation

Direct memory access

Or VM exit:

```
run = mmap(cpufd,..)
ioctl(cpufd, KVM_RUN)
run→exit_reason == KVM_EXIT_IO/MMIO
run→io/mmio_addr mapping
BQL!
MemoryRegionOps.read/write()
```

← ioctl(vmfd, KVM_IRQ_LINE, irq_level)

KVM nifty ioctl

KVM_IOEVENTFD

This ioctl attaches or detaches an ioeventfd to a legal pio/mmio address within the guest. A guest write in the registered address will signal the provided event instead of triggering an exit.

KVM_IRQFD

Allows setting an eventfd to directly trigger a guest interrupt.

Ioeventfd vs MemoryRegionOps

```
struct kvm_ioeventfd {
        __u64 datamatch;
        __u64 addr;        /* legal pio/mmio address */
        __u32 len;         /* 0, 1, 2, 4, or 8 bytes    */
        __s32 fd;
        __u32 flags;
        __u8  pad[36];
};
```

Write only, coalesced events, not a range API

Extend it to support ranges - IOEVENTFD_FLAG_RANGE?

Then KVM_GET_IOEVENTS (similarity with AIO)

For traditional sync devices

IPC qemu → helper (necessary for TCG)

Introduce a KVM user device?

```
devfd = ioctl(vmfd, KVM_CREATE_DEVICE_USER)
reg = {
  .group = KVM_DEV_USER_GROUP,
  .attr = KVM_DEV_USER_SET_MEMORY_REGION,
  .addr = (struct) { .slot = 0,
                     .addr = 0x3f8,
                     .flags = PIO,
                     .eventfd = efd }
}
ioctl(devfd, KVM_SET_DEVICE_ATTR, &reg)
poll(efd)
ioctl(devfd, KVM_GET_DEVICE_CPU_EXITS, &exits)
ioctl(devfd, KVM_SET_DEVICE_CPU_EXITS, &exits)
```

Migration

In qemu stream vs out of stream

- Handled by qemu or not
- Security aspect

Share VMState infrastructure with helper?

- Instead of blobs
- Make it a library, IPC hook for saving/loading to/from stream
- Unlikely to be accepted as standard in external projects

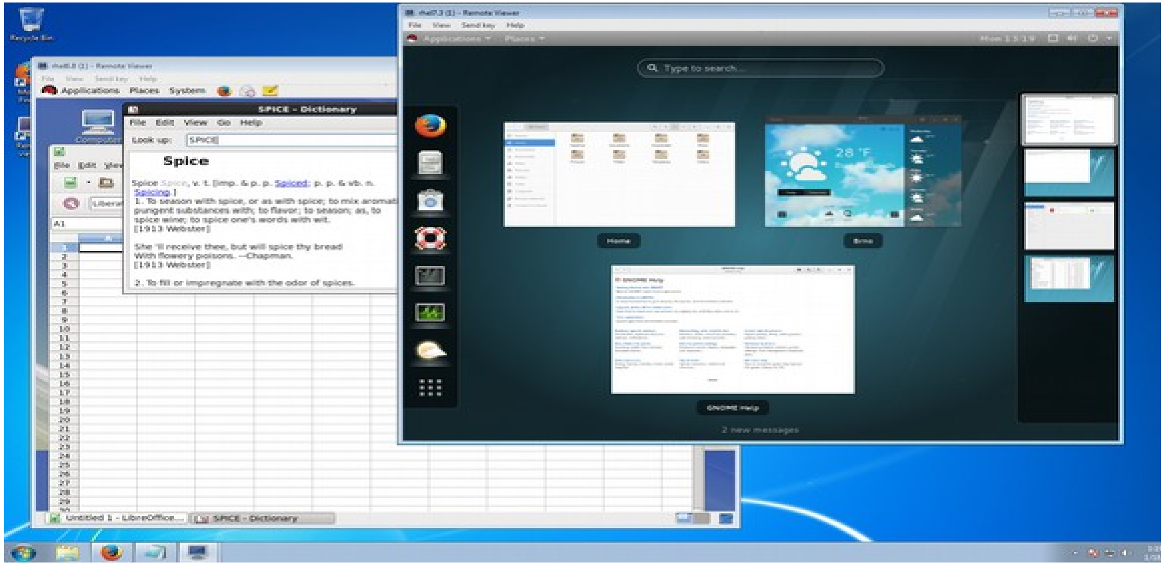
Mostly non-existent today, with rare exceptions

And today?

- VNC / Spice
- Block devices
- usbredir / cacard
- ipmi-bmc-extern
- TPM emulation
- ~~ivshmem~~ device
- vhost, vhost-user
- VFIO/mdev

VNC & Spice

- UI in remote process
- Resume session
- Migration
- VT & monitor?



What about?

QEMU to start a graphical client instead?

Remove GTK/SDL/VTE/audio code from qemu?

Block devices

```
$ qemu-nbd -k nbd.sock vm.qcow2
$ qemu -drive driver=nbd,
  server.path=nbd.sock,server.type=unix
```

(other protocols exist: iSCSI, NBD, SSH, Sheepdog, gluster, http/ftp..)

Block devices

Would performance be good enough for general case?

Could use shared memory, to avoid extra copy, opportunistic polling…

Usbredir

```
$ usbredirserver -p 2001 <vendorid>:<prodid>

$ qemu <ehci-uhci> …
-chardev socket,port=2001,id=chr
-device usb-redir,chardev=chr
```

✓migrate

✗**migrate**

USB Devices

QEMU emulation of USB devices in standalone process using usbredir API?

Cacard

```
$ qemu … -device usb-ccid
-chardev socket,server,port=2001,id=chr
-device ccid-card-passthru,chardev=chr

$ vscclient <host> 2001
```

✓migrate ✗**migrate**

Ipmi-bmc-extern

```
$ ipmilan -c conf-file -f cmd-file -s statedir

$ qemu … -device ipmi-bmc-extern,chardev=chr
-chardev socket,id=chr,host=localhost,port=…
-device isa-ipmi-bt,bmc=bmc0,irq=0
```

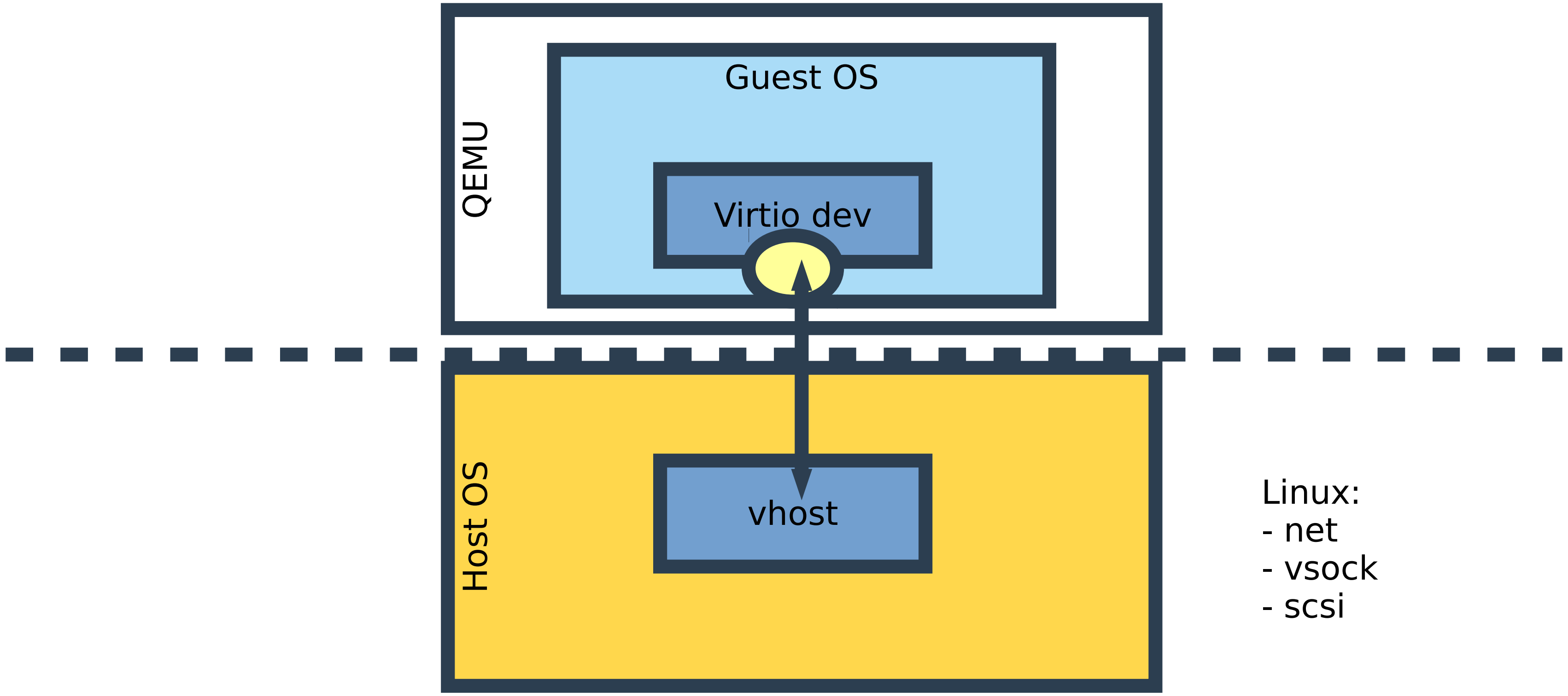
TPM emulator

```
$ swtpm socket --tpmstate dir=/tmp/myvtpm --ctrl
type=unixio,path=/tmp/ctrl
```

```
$ qemu … -tpmdev emulator,id=tpm0,chardev=chr
-chardev socket,id=chr,path=/tmp/ctrl
-device tpm-tis,tpmdev=tpm-tpm0,id=tpm0
```

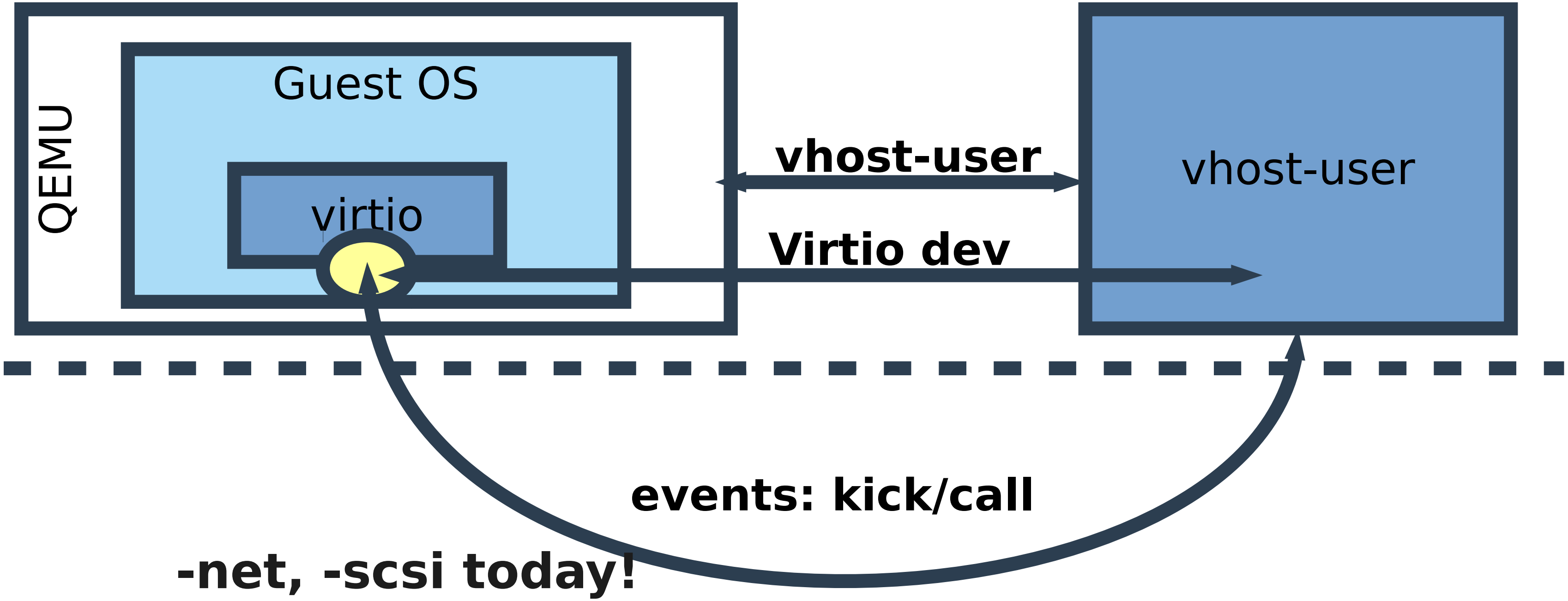
✓ migrate soon

Vhost overview



Linux:
- net
- vsock
- scsi

vhost-user



-net, -scsi today!

-blk, -gpu, -input, -crypto coming!

Vhost(-user) in a nutshell

Memory listener to have RAM flat view

SET_MEM_TABLE

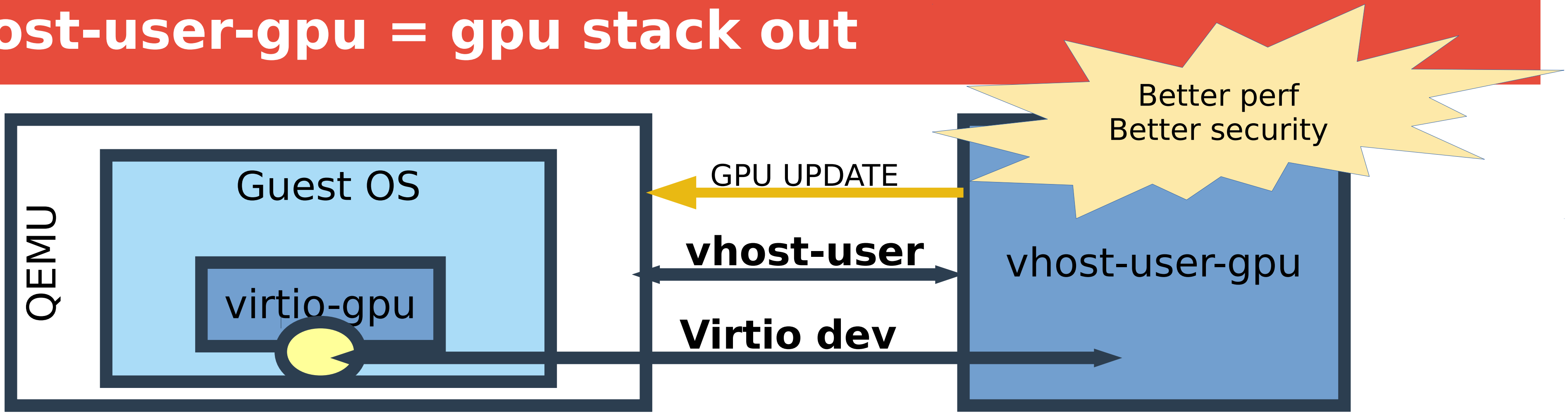
(Fd)	Guest Address	User Address	Size
34	0xA000	0xf2bc0000	0x40000000
...			

SET_VRING_ADDR, SET_VRING_NUM

Index	Desc Address	Used Address	Avail Address
0	0xf2bc1000	0xf2bc2000	0xf2bc3000

SET_VRING_KICK(fd), SET_VRING_CALL(fd)

vhost-user-gpu = gpu stack out




```
-object vhost-user-backend,id=vug,cmd="./vhost-user-gpu"
-device virtio-vga,virgl=true,vhost-user=vug
```

GPU socket commands:
- SCANOUT
- UPDATE
- GL SCANOUT
- GL UPDATE (+)
- CURSOR UPDATE

Could be handled outside of QEMU
(spice or client)

Benefits of virgl out of process?

- avoids blocking qemu main loop

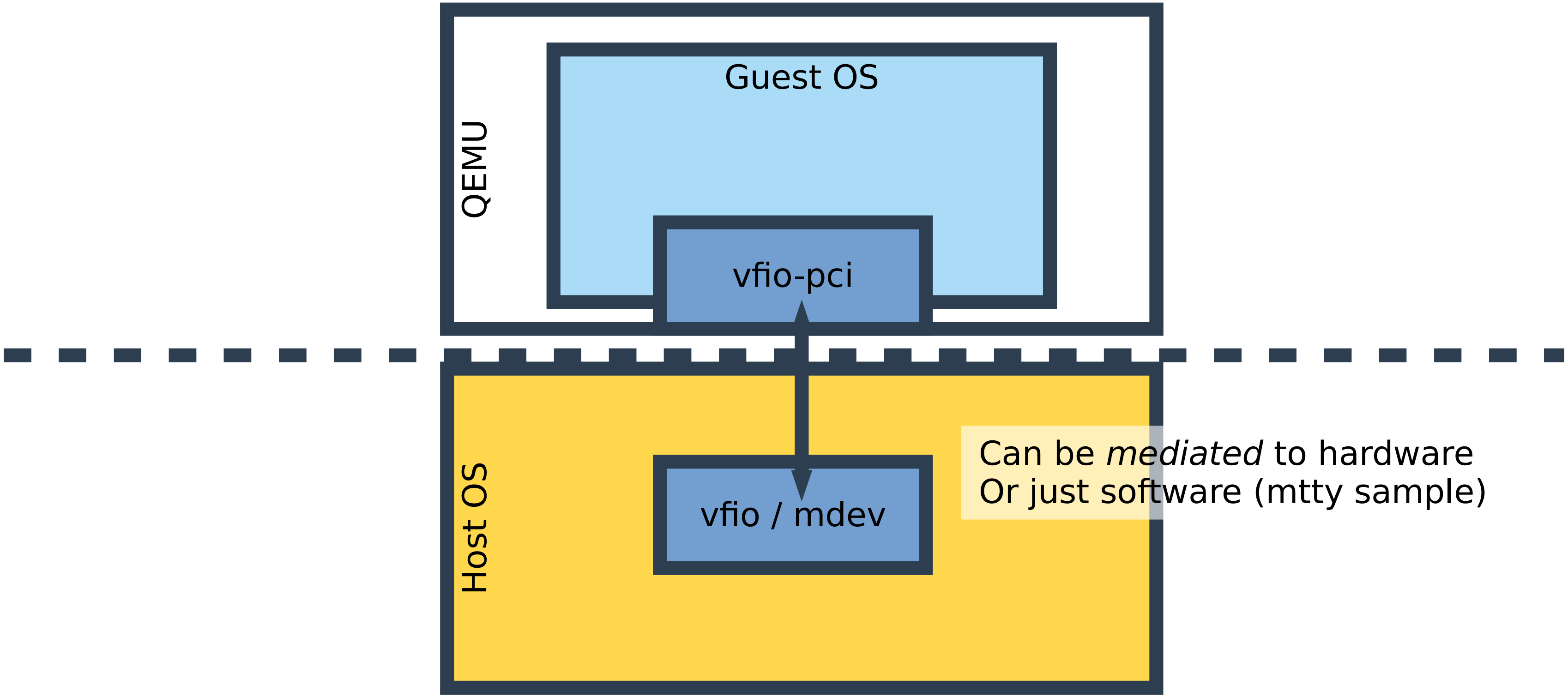
Shaders may take long to compile

- virgl needs to do polling (GL queries & fences)

- virgl crash (various crash/leaks fixed)

- GL isn’t a very safe API (size/buffer mismatch – ARB_robustness is an extension)

Mdev / vfio overview



Can be *mediated* to hardware
Or just software (mtty sample)

VFIO in userspace?

Implement PCI devices in userspace with a VFIO-user?

Conclusion

- Qemu is mostly monolithic & big today
- Strategies to run separate processes exist, but provide different interfaces & integration levels
- Use vhost-user for virtio devices
- Many ideas for a multi-process future

Questions

QEMU

STOP STOP STOP STOP STOP STOP STOP

STOP
STOP
✓ **migrate**

✗ **migrate**

Virtio device → vhost-user device

Check ioeventfd support

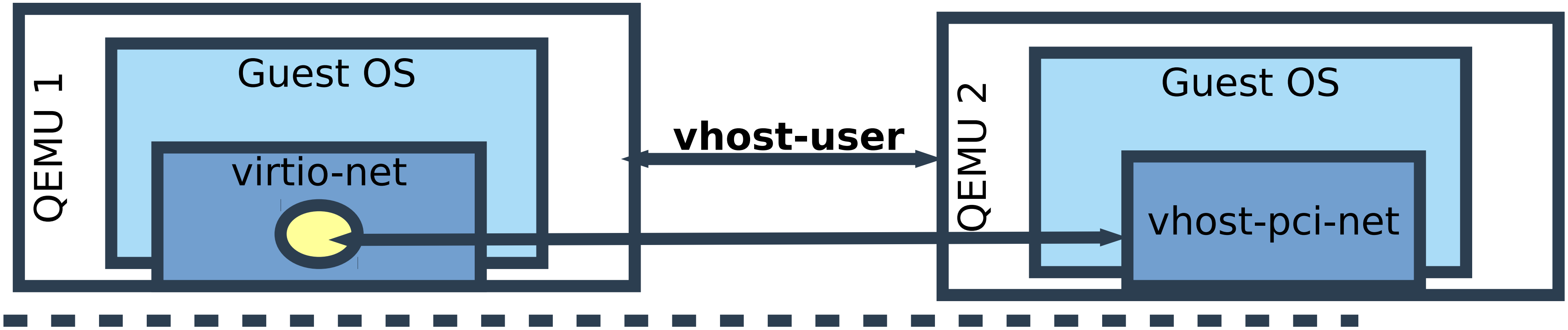
vhost_dev.vqs = g_new(vhost_queues, N)

vhost_dev_init(vhost, chr, TYPE_USER, timeout)

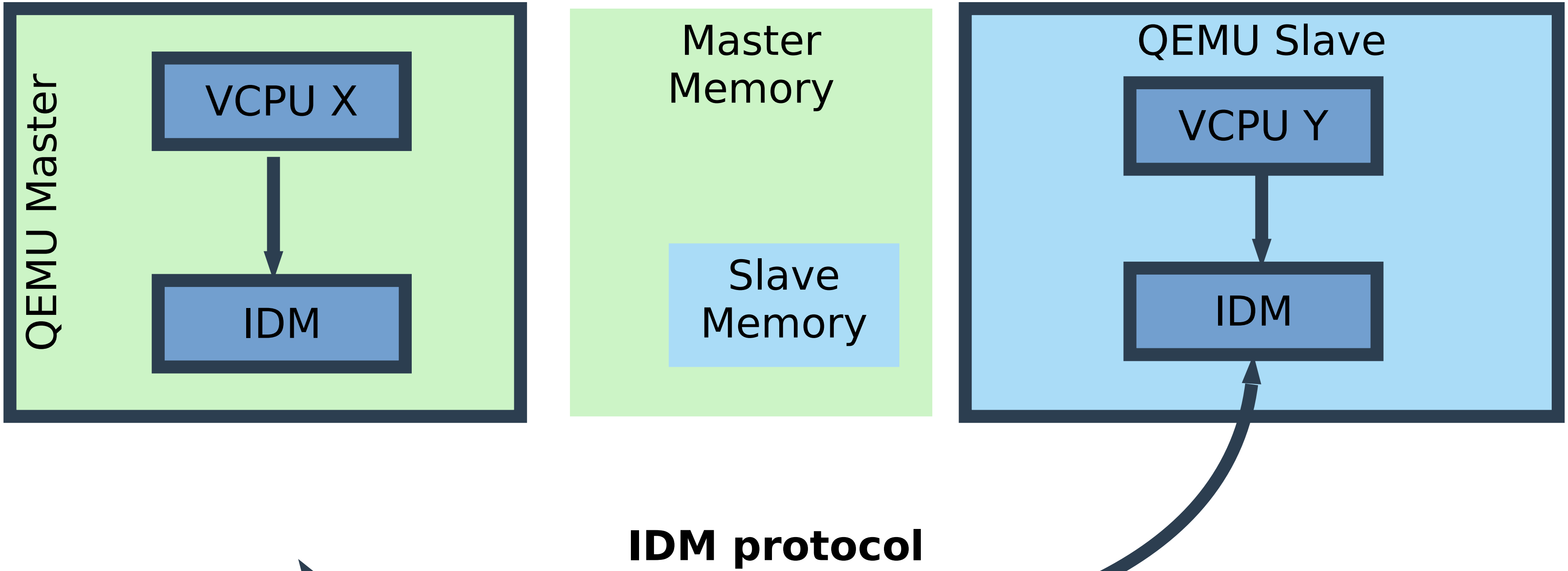
VirtioDeviceClass.set_status() & reset():

- vhost_dev_enable_notifiers()
- VirtioBus parent: set_guest_notifiers()
- Set dev.acked_features = virtio.guest_features
- vhost_dev_start()
- vhost_virtqueue_mask() forall queues

Vhost-pci WIP
(Inter-VM communication)



Heterogeneous QEMU



"[RFC PATCH 0/8] Towards an Heterogeneous QEMU" C. Pinto Sept 2015
& virtio-sdm & also xilinx remote-proc