device emulation

an introduction

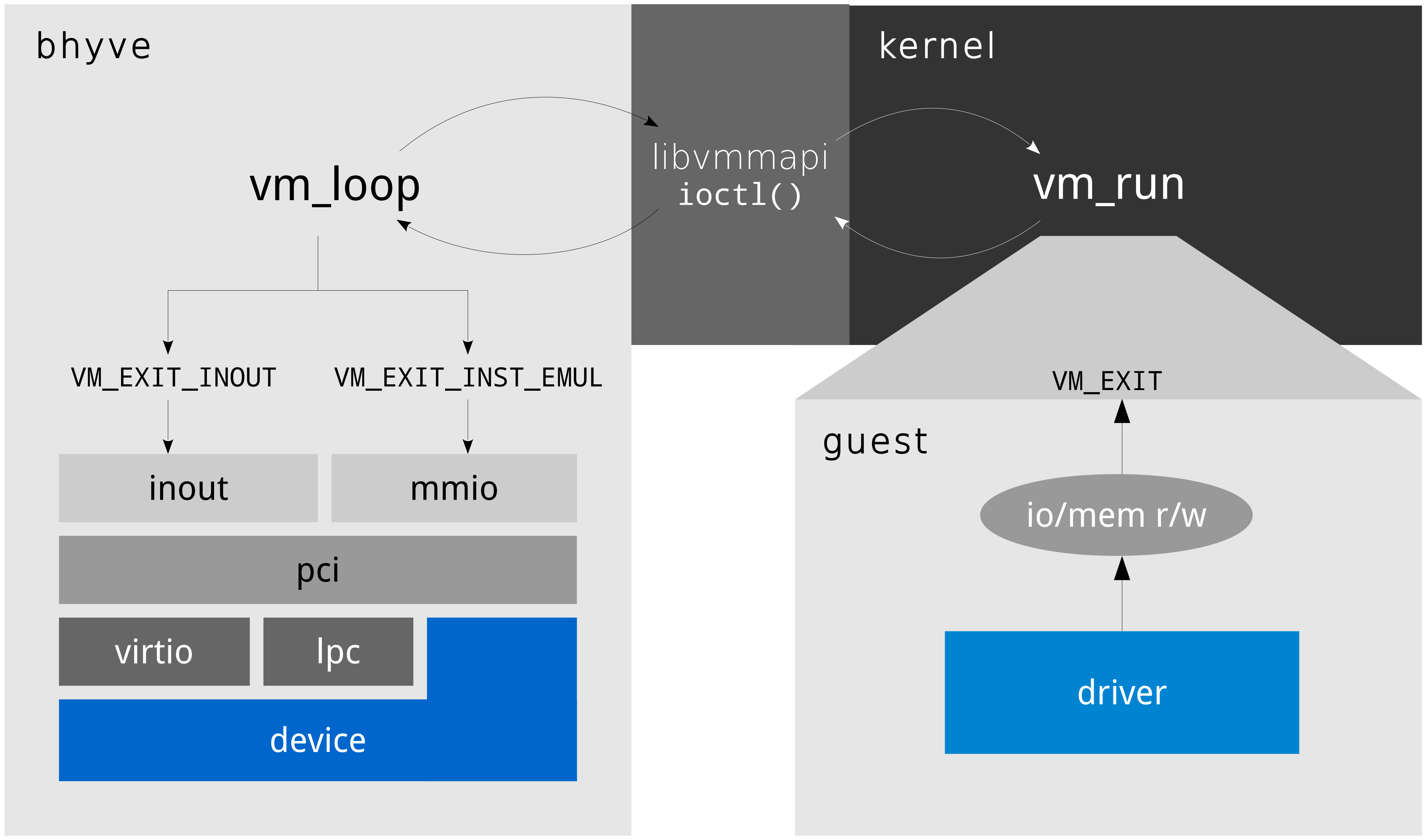
leon dang

nahanni systems

overview

- Most emulated in userspace usr.sbin/bhyve
  - kernel ones in vmm/io/ (PICs and timers)
- ISA-LPC
  - uart, rtc
- PCI
  - virtio
    - block – storage
    - net – tap networking
    - rng – random entropy from /dev/random
  - ahci
  - pass-through

architecture



virtio

- Virtual I/O (virtio) device spec
  - common framework for IO virtualization
  - Linux and FreeBSD built-in support; Windows requires custom drivers
  - transports: PCI, MMIO, channel (e.g. S/390)
  - devices types: network, block, entropy, console, SCSI
  - devices implement virtqueues for data transport
  - http://docs.oasis-open.org/virtio/virtio/v1.0/virtio-v1.0.html

example virtio device

- basic pci device
  - virtio vendor ID 0x1AF4, dev-id 0x1000-0x103F
- virtio random number generator
  - `usr.sbin/bhyve/pci_virtio_rnd.c`
  - guest rng driver requests 32-bit number to replenish its random pool
  - FreeBSD /dev/random non-blocking
    - Yarrow PRNG
    - in-progress: Fortuna

device (virtio) registration

- declare virtio constants and handlers

```
static struct virtio_consts vtrnd_vi_consts = {
        "vtrnd",                        /* our name */
        1,                              /* we support 1 virtqueue */
        0,                              /* config reg size */
        pci_vtrnd_reset,                /* reset */
        pci_vtrnd_notify,               /* device-wide qnotify */
        NULL,                           /* read virtio config */
        NULL,                           /* write virtio config */
        0,                              /* our capabilities */
};
```

- virtio:
  - qnotify handler called on new message from guest to read/write

device (pci) registration

- register handlers
  - use vi_pci_{read|write} for virtio

```
struct pci_devemu pci_de_vrnd = {
    .pe_emu =       "virtio-rnd",    /* bhyve cmd line  */
    .pe_init =      pci_vtrnd_init, /* device init */
    .pe_barwrite = vi_pci_write,    /* virtio framework */
    .pe_barread =  vi_pci_read      /* virtio framework */
};
PCI_EMUL_SET(pci_de_vrnd);
```

device initialization

- .pe_init
  - init device
  - virtio: link virtio constants to softc
  - register PCI config space, intr, io

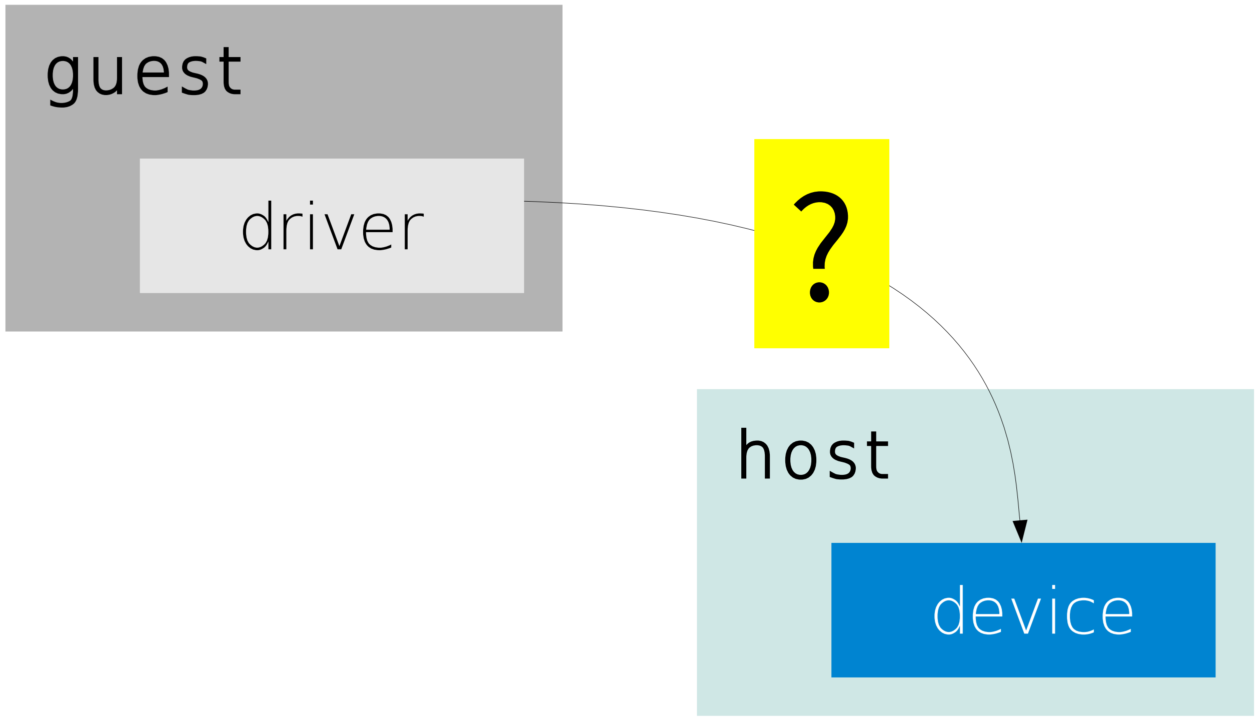
```
vi_softc_linkup(&sc->vrsc_vs, &vtrnd_vi_consts, sc, pi, &sc->vrsc_vq);

pci_set_cfgdata16(pi, PCIR_DEVICE, VIRTIO_DEV_RANDOM);
pci_set_cfgdata16(pi, PCIR_VENDOR, VIRTIO_VENDOR);
pci_set_cfgdata8(pi, PCIR_CLASS, PCIC_CRYPTO);
pci_set_cfgdata16(pi, PCIR_SUBDEV_0, VIRTIO_TYPE_ENTROPY);

if (vi_intr_init(&sc->vrsc_vs, 1, fbsdrun_virtio_msix()))
       return (1);

vi_set_io_bar(&sc->vrsc_vs, 0);
```

launching bhyve

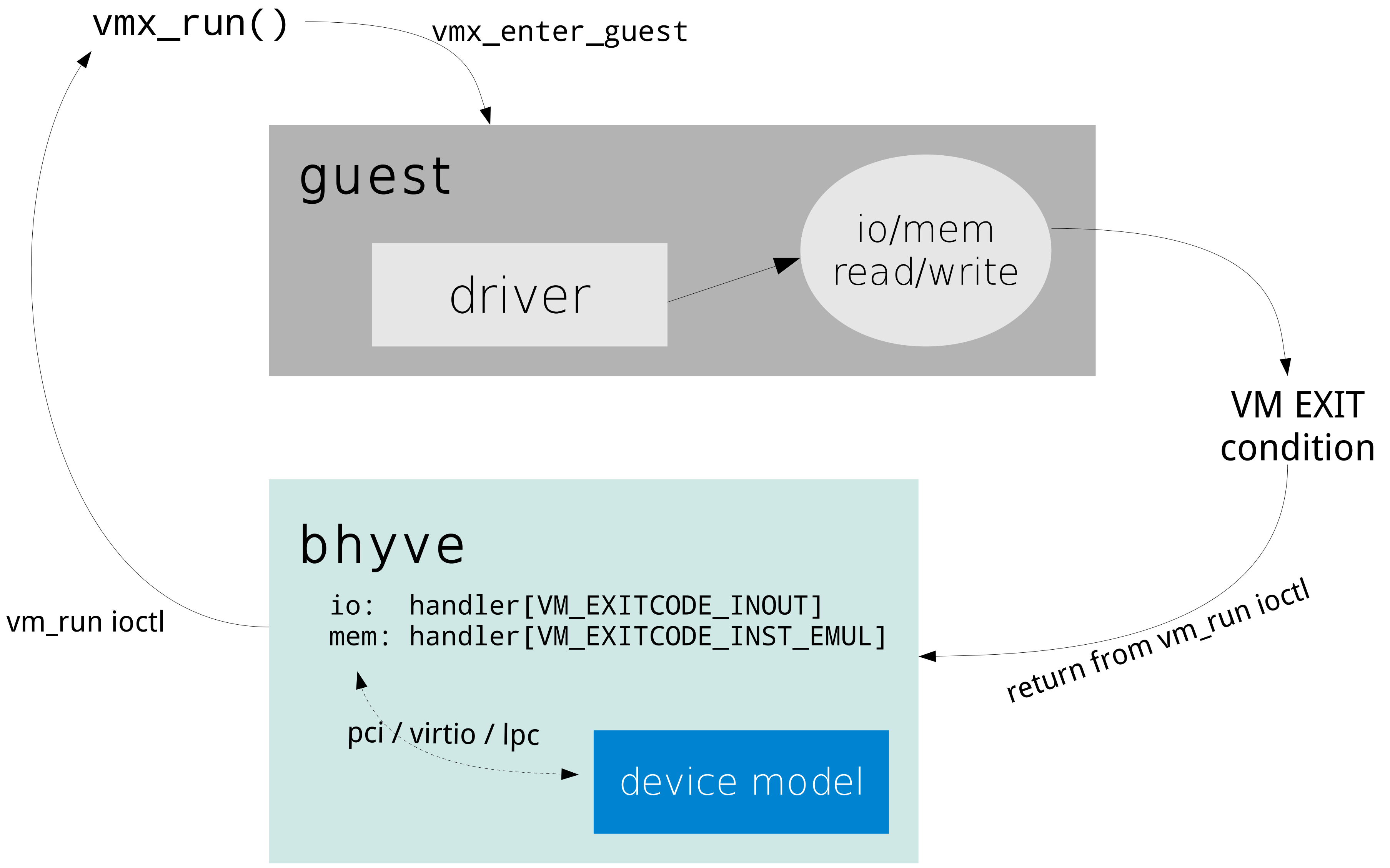
```
$ bhyve … -s <slot>,emulation,{conf} … vmname
```

- specify free slot for pci device
  - bus:slot:function
  - slot:function
  - slot
- lpc uses device name and options, e.g.
  - `-l com1,stdio`
- refer to bhyve(8)

device i/o

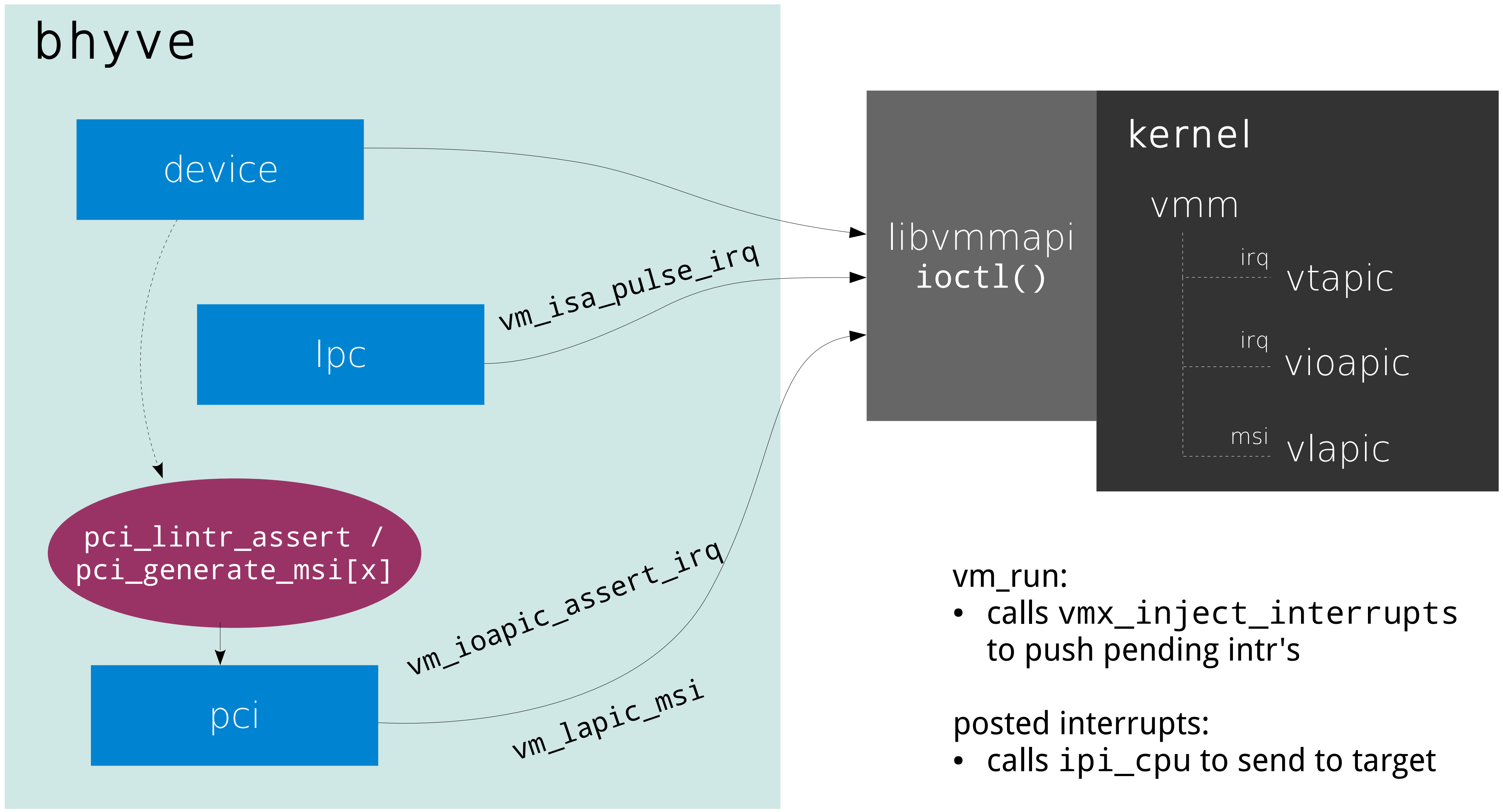


device i/o



vmx_run()
vmx_enter_guest
guest
driver
io/mem read/write
VM EXIT condition
return from vm_run ioctl
bhyve
io: handler[VM_EXITCODE_INOUT]
mem: handler[VM_EXITCODE_INST_EMUL]
pci / virtio / lpc
device model
vm_run ioctl

device interrupts



vm_run:

- calls `vmx_inject_interrupts` to push pending intr's

posted interrupts:

- calls `ipi_cpu` to send to target

accessing guest memory

- `paddr_guest2host()` helper function
  - access to memory mapped regions
  - device can directly access returned address

in-progress & future

- in progress
  - at-keyboard
  - vga
  - watchdog
  - e1000
- future?
  - other virtio: scsi, serial, gpu
  - audio, usb, ethernets