libvfio-user status update

john.levon@nutanix.com

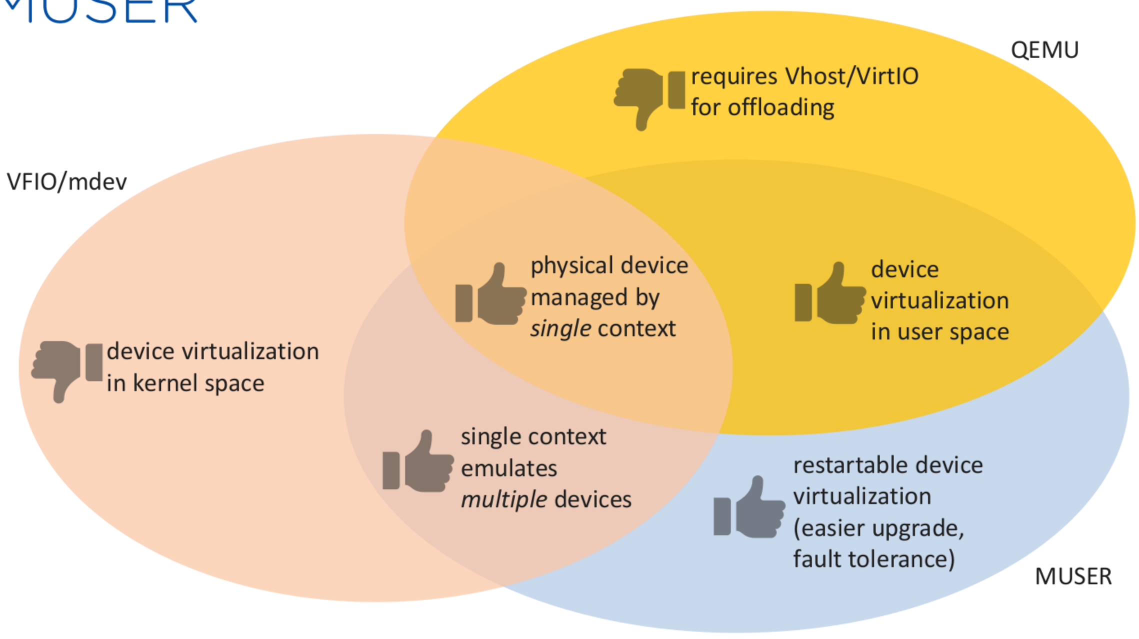
thanos.makatos@nutanix.com

swapnil.ingle@nutanix.com

2021-09-16

KVM Forum 2019

MUSER



QEMU

👎 requires Vhost/VirtIO for offloading

VFIO/mdev

👎 device virtualization in kernel space

👍 physical device managed by *single* context

👍 device virtualization in user space

👍 single context emulates *multiple* devices

👍 restartable device virtualization (easier upgrade, fault tolerance)

MUSER

- "MUSER: mediated userspace device" Thanos Makatos & Swapnil Ingle
- working proof of concept, but had drawbacks
  - kernel module, patch
- much has changed in two years
  - worked with community on better approach

vfio-user

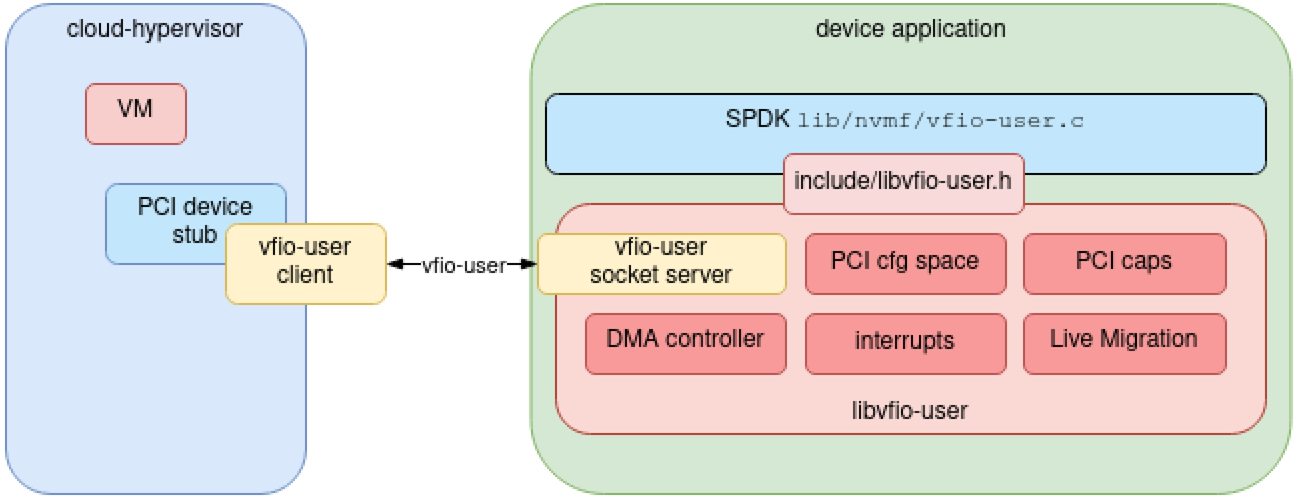
- **`vfio-user`** is a protocol for managing external device servers
    - motivations: performance, security, resilience
- control plane focus
- message protocol over a communication channel
- analogous to, and based on, Linux's **`vfio ioctl()`** interface
- similar to **`vhost-user`**, but not **`virtio`** specific
- VMM agnostic

vfio-user message types

Name	Purpose	Direction
VFIO_USER_VERSION	Lifecycle	client -> server
VFIO_USER_DEVICE_RESET	Lifecycle	client -> server
VFIO_USER_DEVICE_GET_INFO	Lifecycle	client -> server
VFIO_USER_DEVICE_GET_IRQ_INFO	IRQs	client -> server
VFIO_USER_DEVICE_SET_IRQS	IRQs	client -> server
VFIO_USER_DEVICE_GET_REGION_INFO	Device mem	client -> server
VFIO_USER_DEVICE_GET_REGION_IO_FDS	Device mem	client -> server
VFIO_USER_REGION_READ/WRITE	Device mem	client -> server
VFIO_USER_DMA_MAP/UNMAP	Guest mem	client -> server
VFIO_USER_DMA_READ/WRITE	Guest mem	server -> client
VFIO_USER_DIRTY_PAGES	Guest mem	client -> server

libvfio-user

- C library with two roles
  - **`vfio-user`** socket server
  - PCI device emulation wrapper
- API for sync and async/non-blocking mode

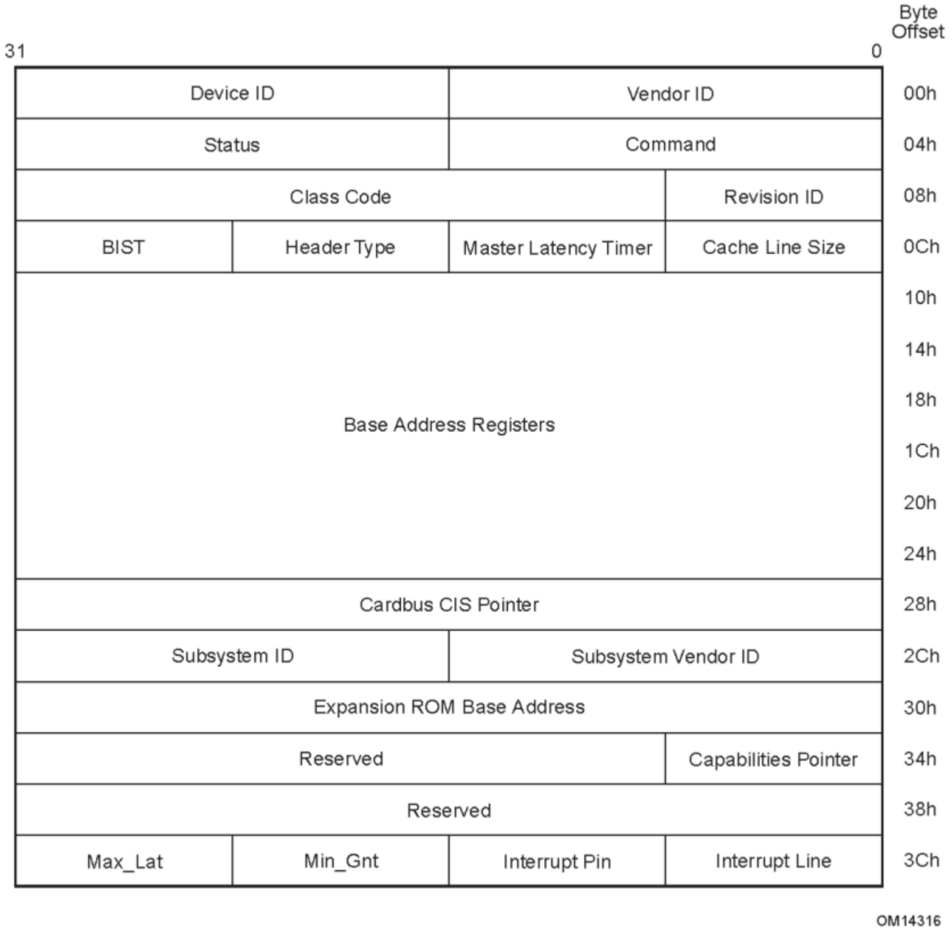


libvfio-user "Hello World"

```c
int main()
{
        vfu_ctx_t *ctx = vfu_create_ctx(VFU_TRANS_SOCK, "/tmp.sock", 0, NULL,
                                        VFU_DEV_TYPE_PCI);

        vfu_pci_init(ctx, VFU_PCI_TYPE_EXPRESS, PCI_HEADER_TYPE_NORMAL, 0);
        vfu_pci_set_id(ctx, 0x494f, 0x0dc8, 0x0, 0x0);

        vfu_setup_region(ctx, VFU_PCI_DEV_BAR2_REGION_IDX, 0x100,
                         bar2_access, VFU_REGION_FLAG_RW, NULL, 0, -1, 0);

        vfu_setup_device_nr_irqs(ctx, VFU_DEV_INTX_IRQ, 1);

        vfu_setup_device_dma(ctx, dma_register, dma_unregister);

        vfu_realize_ctx(ctx);

        /* accept() on the socket */
        vfu_attach_ctx(ctx);

        do {
                err = vfu_run_ctx(ctx);
        } while (err != -1);
}
```

```c
static ssize_t
bar2_access(vfu_ctx_t *ctx, char *buf, size_t count, loff_t off, bool is_write)
{
        if (off == DEV_REG_OFF_CTRL) {
                ...
        }
        ...
}
```

```c
typedef struct vfu_dma_info {
    struct iovec iova;
    void *vaddr;
    struct iovec mapping;
    size_t page_size;
    uint32_t prot;
} vfu_dma_info_t;

/* handle new guest memory mapping */
static void
dma_register(vfu_ctx_t *ctx, vfu_dma_info_t *info)
{
        ...
}
```



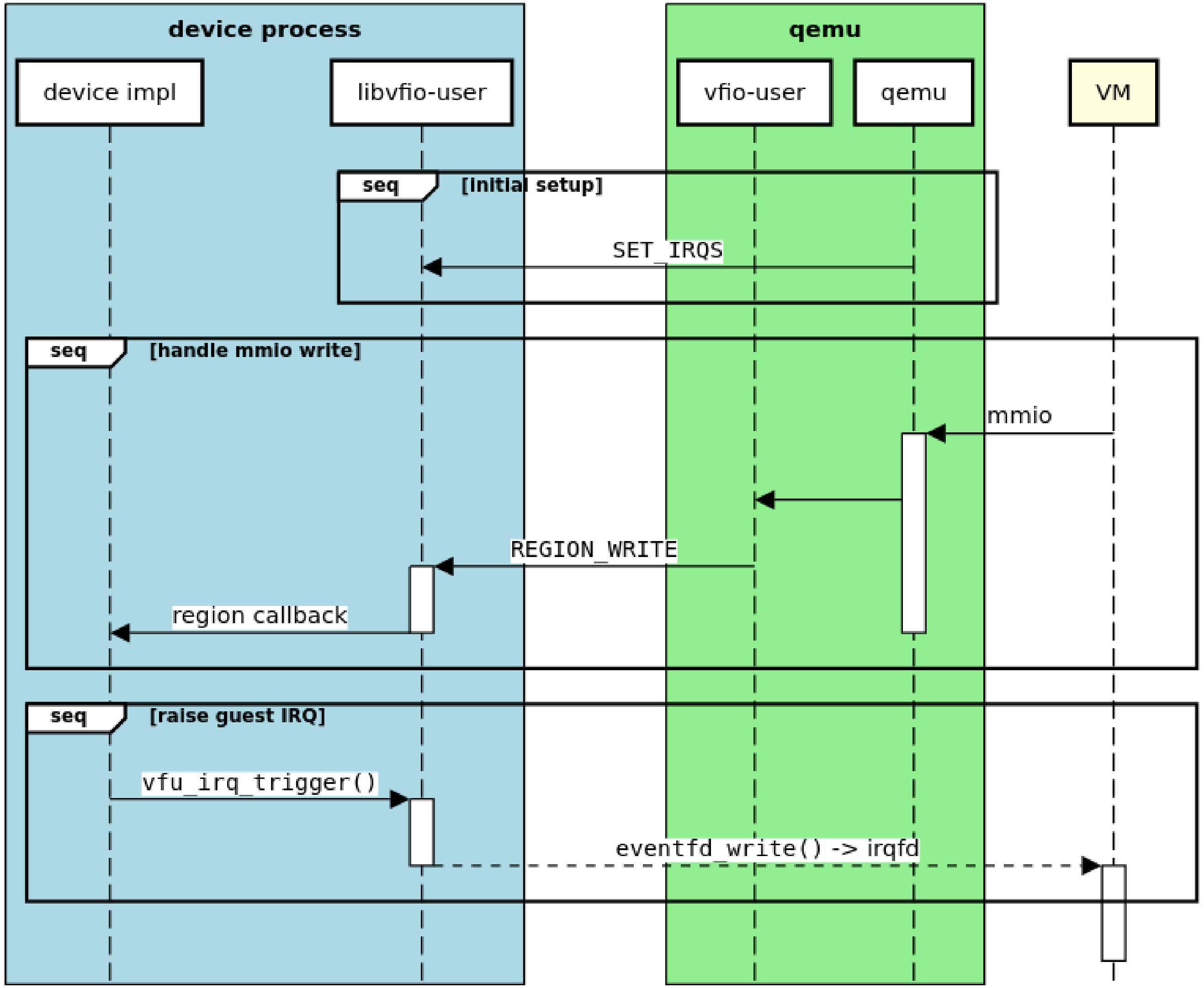
31			0	Byte Offset
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Master Latency Timer	Cache Line Size	0Ch
Base Address Registers				10h
				14h
				18h
				1Ch
				20h
				24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved			Capabilities Pointer	34h
Reserved				38h
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	3Ch

Figure 7-5: Type 0 Configuration Space Header

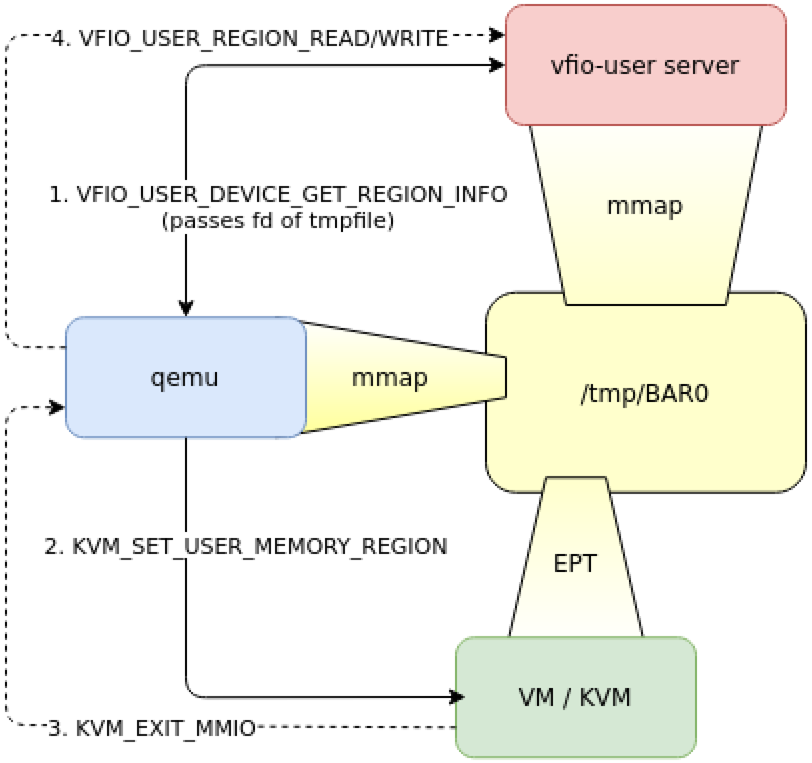
PCI device support

- each context corresponds to a PCI endpoint
- library handles (most) standard config space accesses
  - or, fully delegated (for **`MPQEMU`**)
- other accesses are handled via application callbacks
  - e.g. access to BAR regions
- user can register (extended) capabilities
  - vendor caps via callbacks

IRQ handling

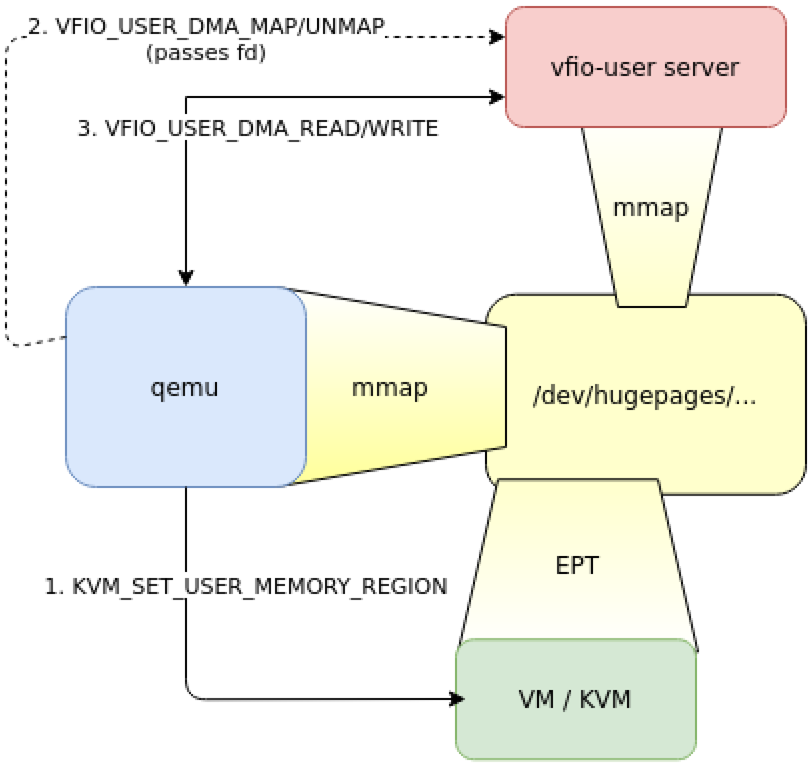


Device region access: MMIO and DMA


4. VFIO_USER_REGION_READ/WRITE
vfio-user server
1. VFIO_USER_DEVICE_GET_REGION_INFO
(passes fd of tmpfile)
mmap
qemu
mmap
/tmp/BAR0
2. KVM_SET_USER_MEMORY_REGION
EPT
VM / KVM
3. KVM_EXIT_MMIO

- **`VFIO_USER_DEVICE_GET_REGION_INFO`**
  - info on device regions (e.g. BARs)
  - can be (partially) mapped directly into VM
- **`VFIO_USER_REGION_READ/WRITE`**
  - read/write via **`vfio-user`** message

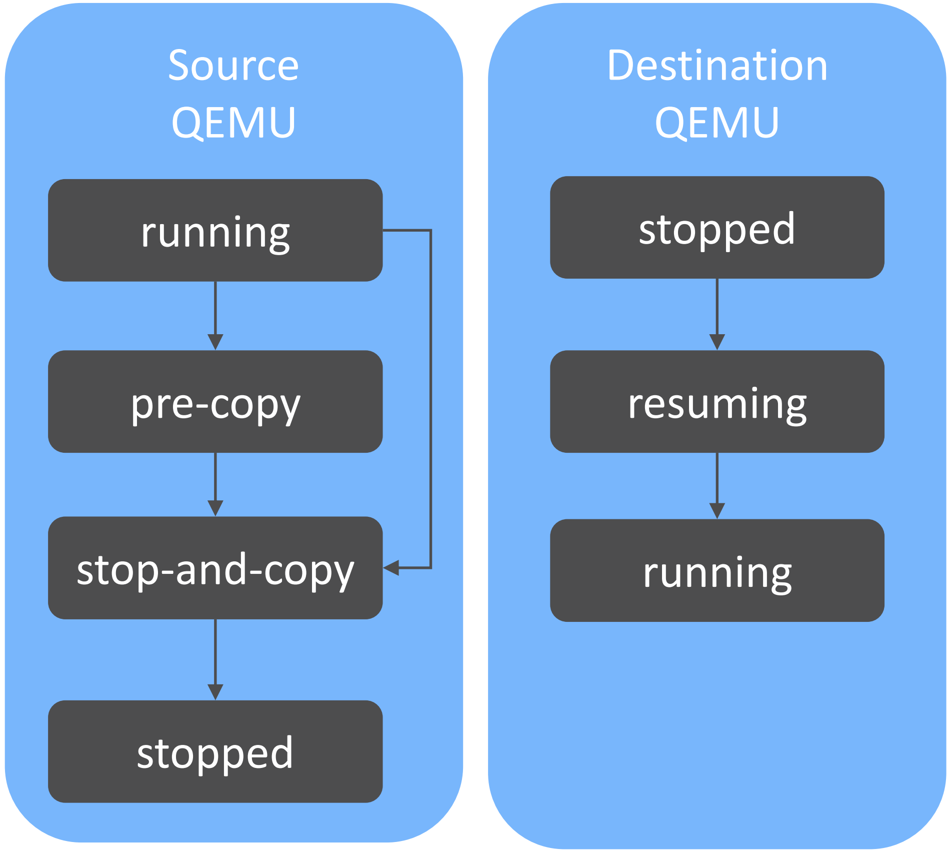
Guest memory: Device DMA


2. VFIO_USER_DMA_MAP/UNMAP
(passes fd)
vfio-user server
3. VFIO_USER_DMA_READ/WRITE
mmap
qemu
mmap
/dev/hugepages/...
EPT
1. KVM_SET_USER_MEMORY_REGION
VM / KVM

- **`vfu_addr_to_sg(gpa, count, ...)`**
  - convert guest PA to scatter-gather list array
- **`vfu_dma_read/write(sg, ...)`**
  - read/write via **`vfio-user`** message (so a misnomer!)
- **`vfu_map_sg(sg, &iov, &iovcnt, ...)` `vfu_unmap_sg()`**
  - provide direct-mapped access
  - dirty page tracking

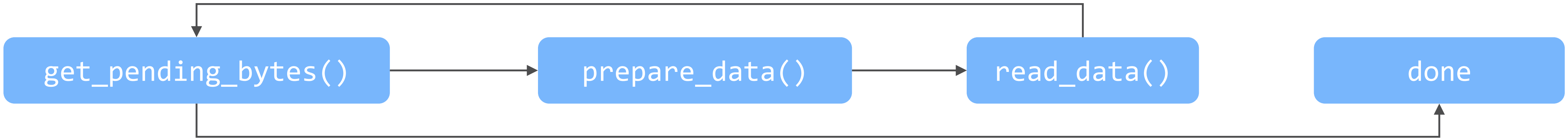
libvfio-user: Migration states

- Uses VFIO migration sub-protocol
    - Migration region with special regs.
    - libvfio-user migration API
- `transition`: device transitions between states
    - running/ stopped
    - pre-copy / stop-and-copy
    - resuming



libvfio-user: copying migration data

- Migrating *from*
  - `get_pending_bytes`: QEMU asks how much data need migrating
  - `prepare/read_data`: QEMU tells device to prep. migr. data, reads migr. data



- Migrating *to*
  - `prepare_data`: device tells QEMU where to write migr. data
  - `write_data/data_written`: guest writes migr. data to device

Demo

- GPIO sample from original MUSER presentation
  - Simple device with external pin
  - Pin can be either zero or one and can be read by host driver
  - Now using latest libvfio-user
- Live migrate from C implementation to Rust implementation

Future Work

- Stability (1.0 API/ABI)
- Ioeventfd & ioregionfd support: cut QEMU out of the loop for port/MMIO accesses
- vIOMMU support
- DMA controller / API improvements
- Better PCI support (caps, non-endpoint)
- Multi-threading
- Restartable device emulation
  - Fault tolerance
  - Seamless upgrade
- Other transports, device types
- H/W device mediation / SR-IOV

libvfio-user

- GitHub: https://github.com/nutanix/libvfio-user
  - BSD licensed
  - Contributions are welcome
- Mailing list: libvfio-user-devel@nongnu.org
- Slack: https://libvfio-user.slack.com

NUTANIX™

Thank you