MUSER: mediated userspace device

Thanos Makatos / thanos@nutanix.com
Swapnil Ingle / swapnil.ingle@nutanix.com

OCTOBER 2019 | KVM FORUM

Disclaimer

Forward-Looking Statement Disclaimer

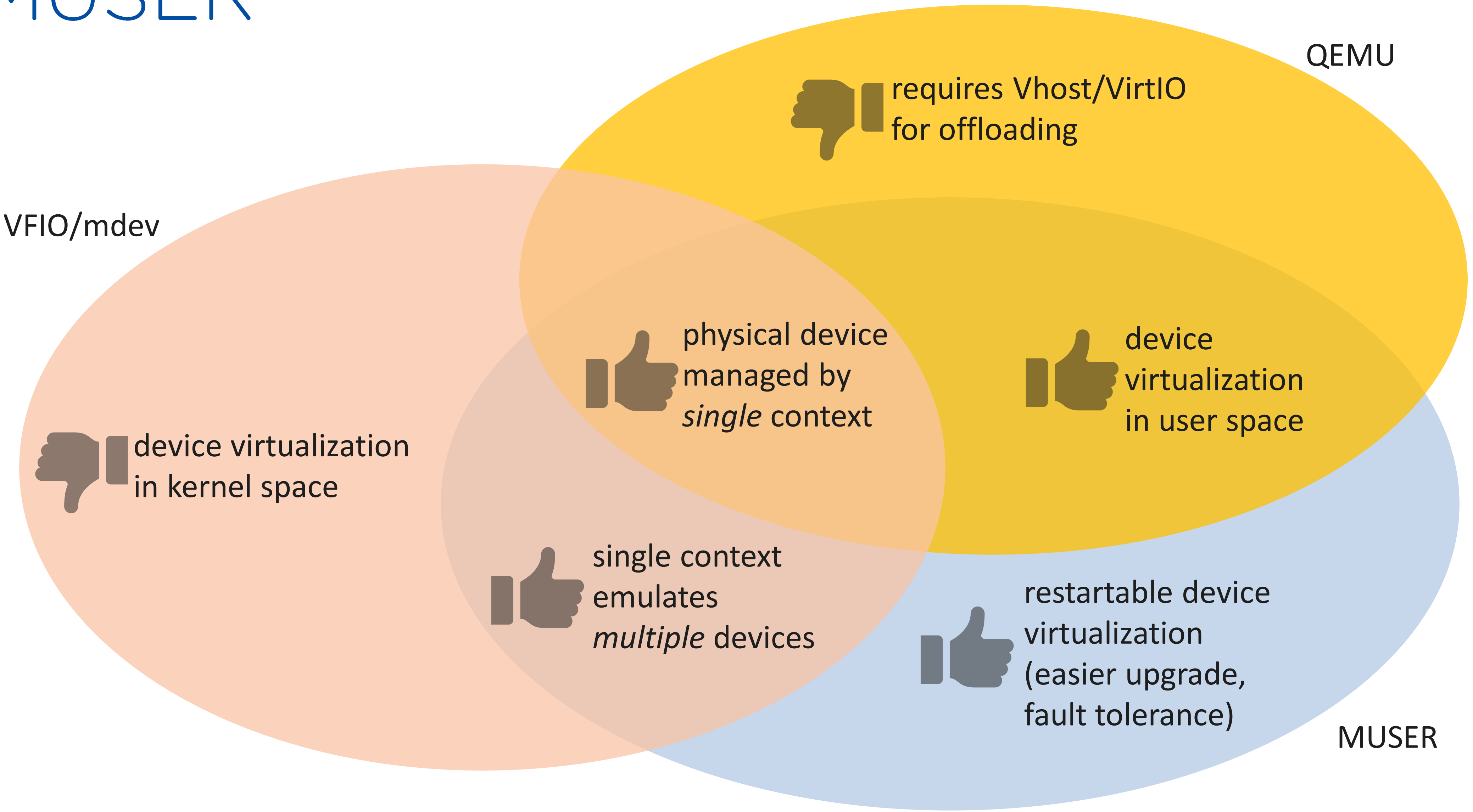
This presentation and the accompanying oral commentary may include express and implied forward-looking statements, including but not limited to statements concerning our business plans and objectives, our financial model targets, new products, services, product features and technology that are under development or in process and the capabilities of such new products, services, product features and technology, our plans to introduce new products, services or product features in the future, the implementation of our products on additional hardware platforms, the integration of newly acquired technology and products, strategic partnerships that are in process, product performance, competitive position, industry environment, and potential market opportunities. These forward-looking statements are not historical facts, and instead are based on our current expectations, estimates, opinions and beliefs. The accuracy of such forward-looking statements depends upon future events, and involves risks, uncertainties and other factors beyond our control that may cause these statements to be inaccurate and cause our actual results, performance or achievements to differ materially and adversely from those anticipated or implied by such statements, including, among others: failure to develop, or unexpected difficulties or delays in developing, new products, services, product features or technology on a timely or cost-effective basis; delays in or lack of customer or market acceptance of our new products, services, product features or technology; the failure of our software to interoperate on different hardware platforms; our ability to successfully integrate newly acquired technology and products; the failure to form, or delays in the formation of, new strategic partnerships and the possibility that we may not receive anticipated results from forming such strategic partnerships; the introduction, or acceleration of adoption of, competing solutions, including public cloud infrastructure; a shift in industry or competitive dynamics or customer demand; and other risks detailed in our Quarterly Report on Form 10-Q for the fiscal quarter ended January 31, 2019, filed with the SEC, filed with the Securities and Exchange Commission. These forward-looking statements speak only as of the date of this presentation and, except as required by law, we assume no obligation to update forward-looking statements to reflect actual results or subsequent events or circumstances. Any future product or roadmap information is intended to outline general product directions, and is not a commitment, promise or legal obligation for Nutanix to deliver any material, code, or functionality. This information should not be used when making a purchasing decision. Further, note that Nutanix has made no determination as to if separate fees will be charged for any future product enhancements or functionality which may ultimately be made available. Nutanix may, in its own discretion, choose to charge separate fees for the delivery of any product enhancements or functionality which are ultimately made available. Certain information contained in this presentation and the accompanying oral commentary may relate to or be based on studies, publications, surveys and other data obtained from third-party sources and our own internal estimates and research. While we believe these third-party studies, publications, surveys and other data are reliable as of the date of this presentation, they have not independently verified, and we make no representation as to the adequacy, fairness, accuracy, or completeness of any information obtained from third-party sources.

Trademark Disclaimer

Motivation

- QEMU: de facto device emulation
- Performance/efficiency
  - Multiple virtual devices cannot be emulated by single process
  - Polling virtual drivers prohibitively expensive
- Must use vhost-user
  - Datapath offloading protocol designed around VirtIO
  - Not clean for non-VirtIO devices
- Monolithic emulation
  - Single point of failure
  - Harder to upgrade
- VFIO/mdev → requires *kernel* vendor driver
- Is there a better way?

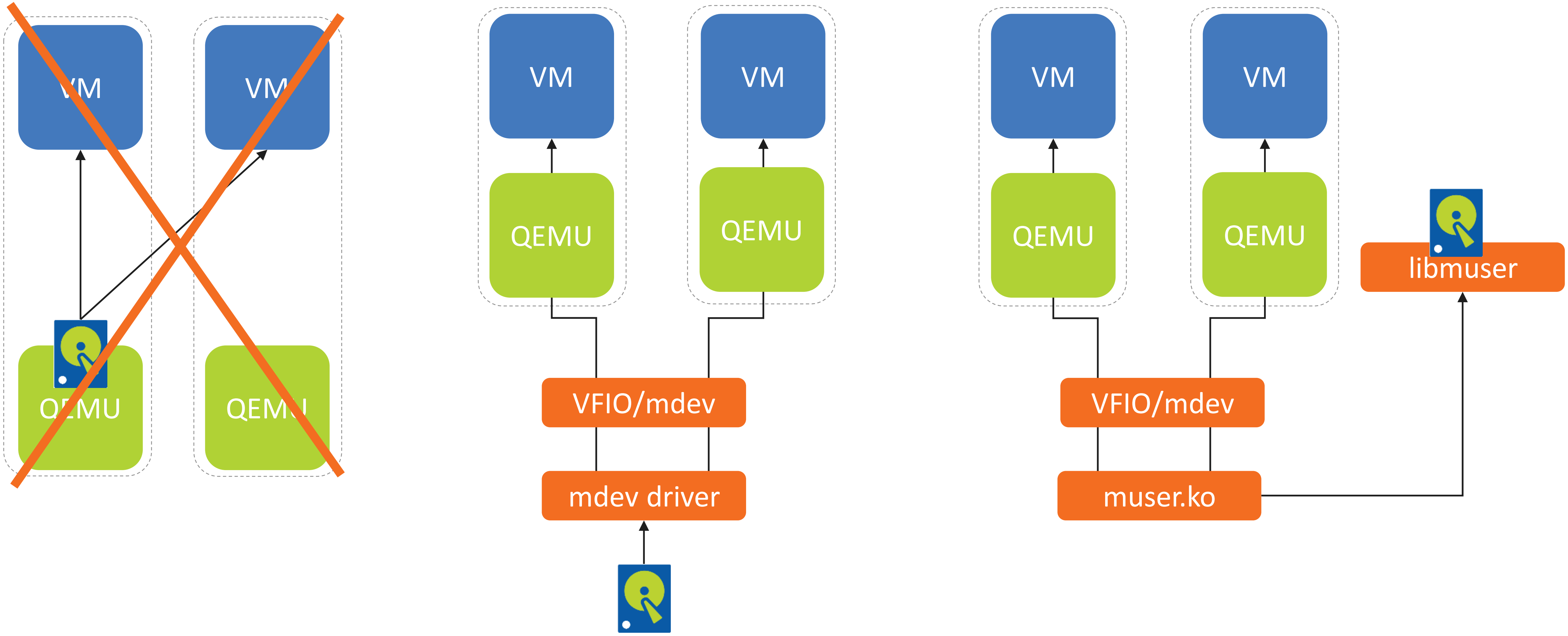
MUSER


QEMU
requires Vhost/VirtIO for offloading
VFIO/mdev
physical device managed by *single* context
device virtualization in user space
device virtualization in kernel space
single context emulates *multiple* devices
restartable device virtualization (easier upgrade, fault tolerance)
MUSER

Background

- VFIO (Virtual Function I/O)
  - Allows secure direct device access to user space
  - Physical device can be passed through to VM in QEMU
- VFIO Mediates Devices (mdev)
  - Virtualizes devices that don’t support SR-IOV
  - Partitions of single device passed through to multiple VMs
    - Simultaneously
    - Securely
  - Kernel vendor driver partitions and mediates
- MUSER leverages VFIO/mdev

MUSER

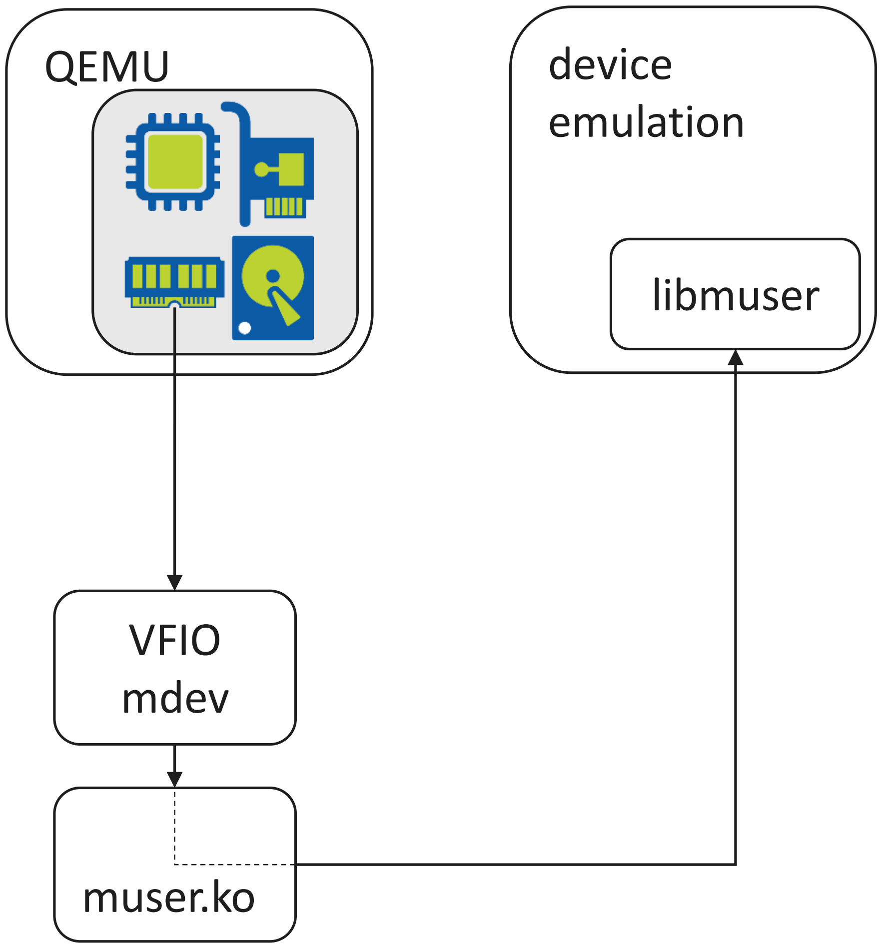

VM
VM
QEMU
QEMU
VM
VM
QEMU
QEMU
VFIO/mdev
mdev driver
VM
VM
QEMU
QEMU
libmuser
VFIO/mdev
muser.ko

MUSER

- Framework for implementing PCI device emulation in user space
  - Implemented as a VFIO mediated device (muser.ko)
  - Forwards ops to user space (libmuser)
  - https://github.com/nutanix/muser
- What does MUSER offer?
  - Single process can emulate multiple devices
  - Complexity hidden (DMA & mem mgmt, IRQs, PCI config space)
- Device emulation app links with libmuser and specifies:
  - Device/vendor ID, PCI regions, #IRQs, PCI caps
  - Callbacks for PCI regions/caps/mmap

MUSER: in a nutshell

- muser.ko
  - Registers with mdev, provides read/write/mmap callbacks
  - Creates char dev for comm. with libmuser
  - Forwards callbacks to libmuser via char dev
- device emulation (libmuser)
  - Opens char dev
  - Sends PCI device configuration
  - Waits for commands and executes user –provided callbacks
- muser.ko/libmuser communication
  - custom ioctl interface
  - libmuser synchronously waits for commands

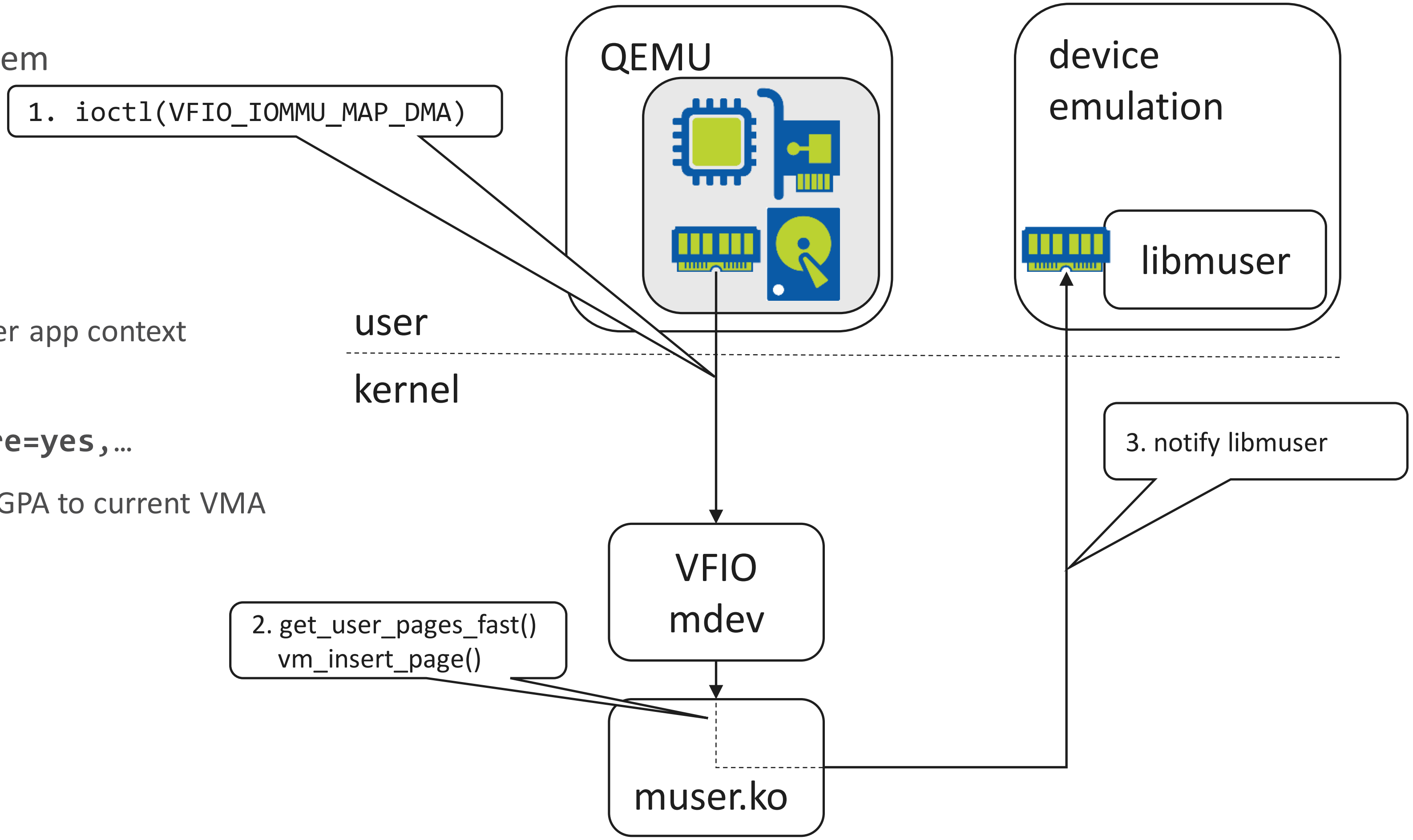


MUSER internals

DMA, mmap, IRQs

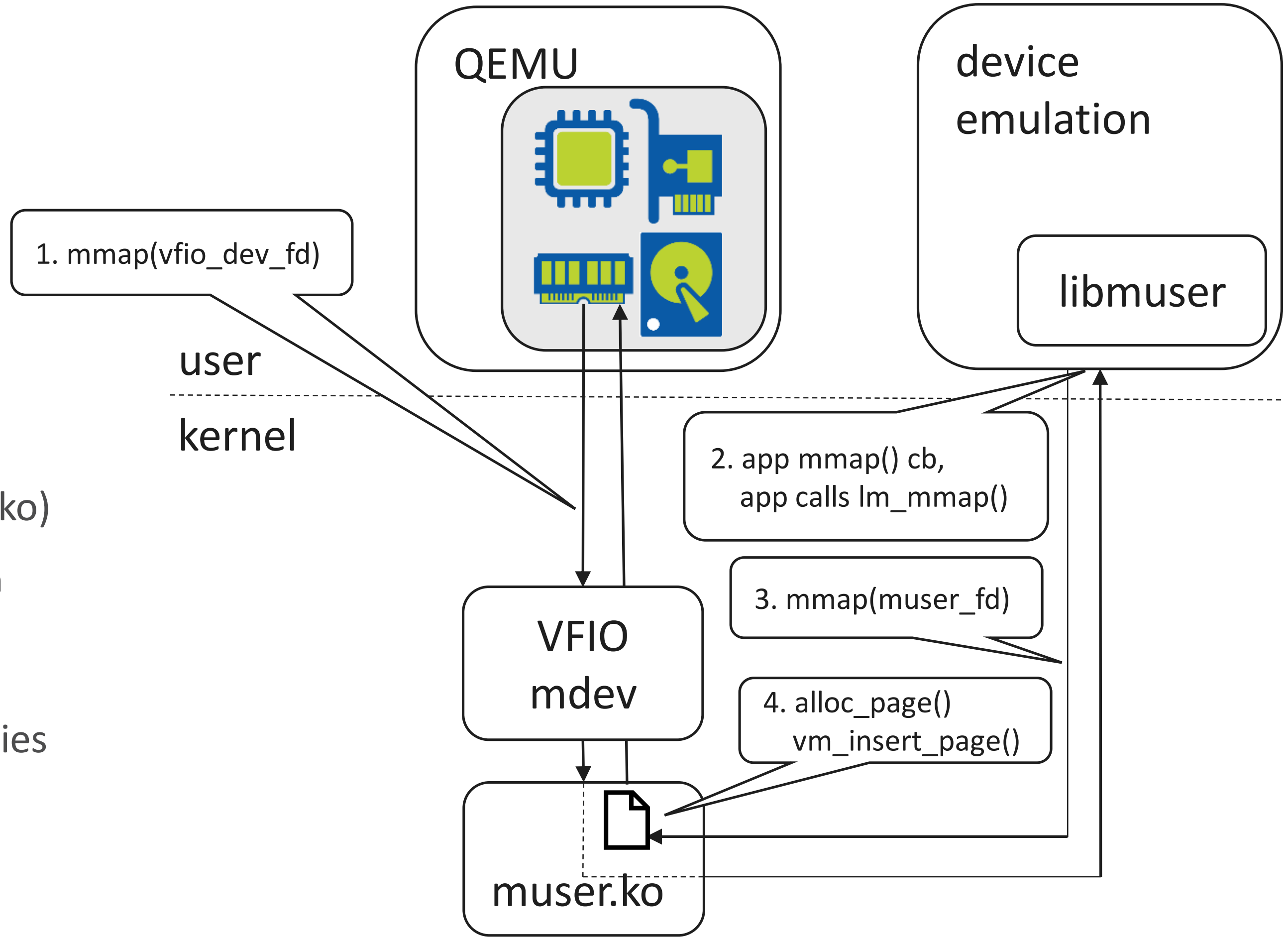
MUSER: DMA map

- Device needs to DMA data from/to guest mem
  - Trivial to do with actual HW
  - QEMU registers guest memory to VFIO
- DMA region registration in MUSER
  - muser.ko injects guest memory into libmuser app context
  - QEMU guest memory *must* be shared:
    `-object memory-backend-file,`**`share=yes`**`,…`
  - libmuser provides functions for translating GPA to current VMA (pin/unpin in HW terms)



MUSER: mapping device memory

- Optional, useful for high performance
  - Sparse maps supported
- QEMU calls mmap to VFIO fd
- muser.ko notifies libmuser
- libmuser calls back to app, app *must* use `lm_mmap()`
- `lm_mmap()` allocs device memory (mmaps into muser.ko)
  - Requesting libmuser to mmap() simplifies implementation
  - mmap()'ing device done infrequently
- Device memory lives in muser.ko, not freed if libmuser dies



MUSER: interrupts

- QEMU passes IRQ fd to VFIO → muser.ko installs fd into libmuser
- Device emulation can trigger interrupts by calling lm_irq_trigger()
- INTx and MSI/X supported
- muser.ko handles gory details
  - `VFIO_DEVICE_SET_IRQS`
  - `VFIO_IRQ_SET_XXX`

MUSER: libmuser API (simplified)

```
typedef struct  { /* more stuff … */
    uint32_t            flags;
    uint32_t            size;
    lm_region_access_t  *fn;
    lm_map_region_t     *map; /* optional */
} lm_reg_info_t;

typedef struct {
    uint32_t            irq_count[LM_DEV_NUM_IRQS];
    lm_reg_info_t       reg_info[LM_DEV_NUM_REGS];
    lm_pci_hdr_id_t     id;
    lm_pci_hdr_ss_t     ss;
    lm_pci_hdr_cc_t     cc;
} lm_pci_info_t;

typedef struct {
    uint8_t id;
    size_t size;
    lm_cap_access_t *fn;
} lm_cap_t;

typedef struct { /* more stuff … */
    lm_pci_info_t   pci_info;
    int (*reset)    (void *pvt); /* optional */
    lm_cap_t        caps[LM_MAX_CAPS]; /* optional */
} lm_dev_info_t;
```

```
typedef ssize_t (lm_region_access_t) (void *pvt, char *buf,
    size_t count, loff_t offset, bool is_write);

typedef unsigned long (lm_map_region_t) (void *pvt,
    unsigned long off, unsigned long len);

typedef ssize_t (lm_cap_access_t)(void *pvt, uint8_t id,
    char *buf, size_t count, loff_t offset, bool is_write);

lm_ctx_t *lm_ctx_create(lm_dev_info_t *dev_info);

int lm_ctx_drive(lm_ctx_t *ctx);

void lm_ctx_destroy(lm_ctx_t *ctx);

int lm_irq_trigger(lm_ctx_t *ctx, uint32_t subindex);

int lm_addr_to_sg(lm_ctx_t *ctx, dma_addr_t dma_addr,
    uint32_t len, dma_sg_t *sg, int max_sg);

int lm_map_sg(lm_ctx_t *ctx, int prot, const dma_sg_t *sg,
    struct iovec *iov, int cnt);

void lm_unmap_sg(lm_ctx_t *ctx, dma_sg_t *sg,
    struct iovec *iov, int cnt);
```

MUSER: libmuser API

- app specifies minimum PCI dev characteristics:

```
lm_dev_info_t dev_info = {
    .pci_info = {
        .id = {.vid = 0x8086, .did = 0x1234},
        .reg_info[LM_BAR0_REG_IDX] = {
            .flags = LM_REG_FLAG_RW,
            .size = 0x40,
            .fn = &bar0
        },
        .irq_count[LM_DEV_INTX_IRQ_IDX] = 1
    }
}
```

- app specifies dev behavior by providing region callbacks:
  - One callback per region (9 in total, unused left blank)
  - VM accessing a particular region (e.g. reading BAR 0) results in registered callback getting called:

```
ssize_t bar0(void *pvt, char *buf, size_t count,
    loff_t offset, bool is_write) { … return count; }
```

  - MUSER handles standard PCI header (first 64 bytes)

31	16	15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Lat. Timer	Cache Line S.	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			Cap. Pointer	34h
Reserved				38h
Max Lat.	Min Gnt.	Interrupt Pin	Interrupt Line	3Ch

MUSER: future work

- Live migration
- Restartable libmuser
- Multithreaded libmuser
- Poll for commands from muser.ko
- libmuser can hide even more PCI complexity
- Language bindings (e.g. Python, JavaScript)
- Provide more examples
- Trap mem writes when mmap is used

https://github.com/nutanix/muser

Thanos Makatos / thanos@nutanix.com
Swapnil Ingle / swapnil.ingle@nutanix.com