Hunting VirtIO Specification Violations

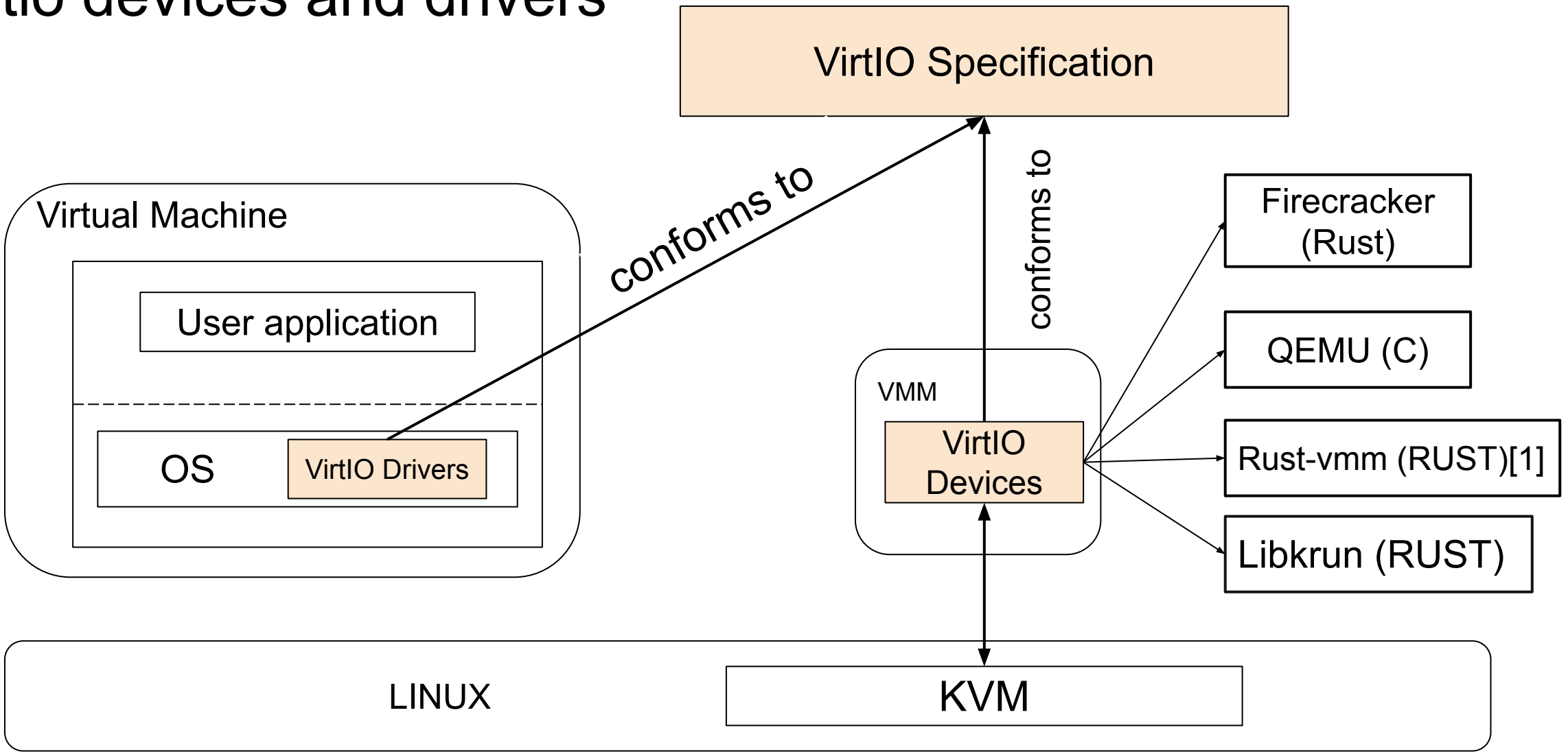
Matias Vara Larsen
mvaralar@redhat.com

Red Hat

Me

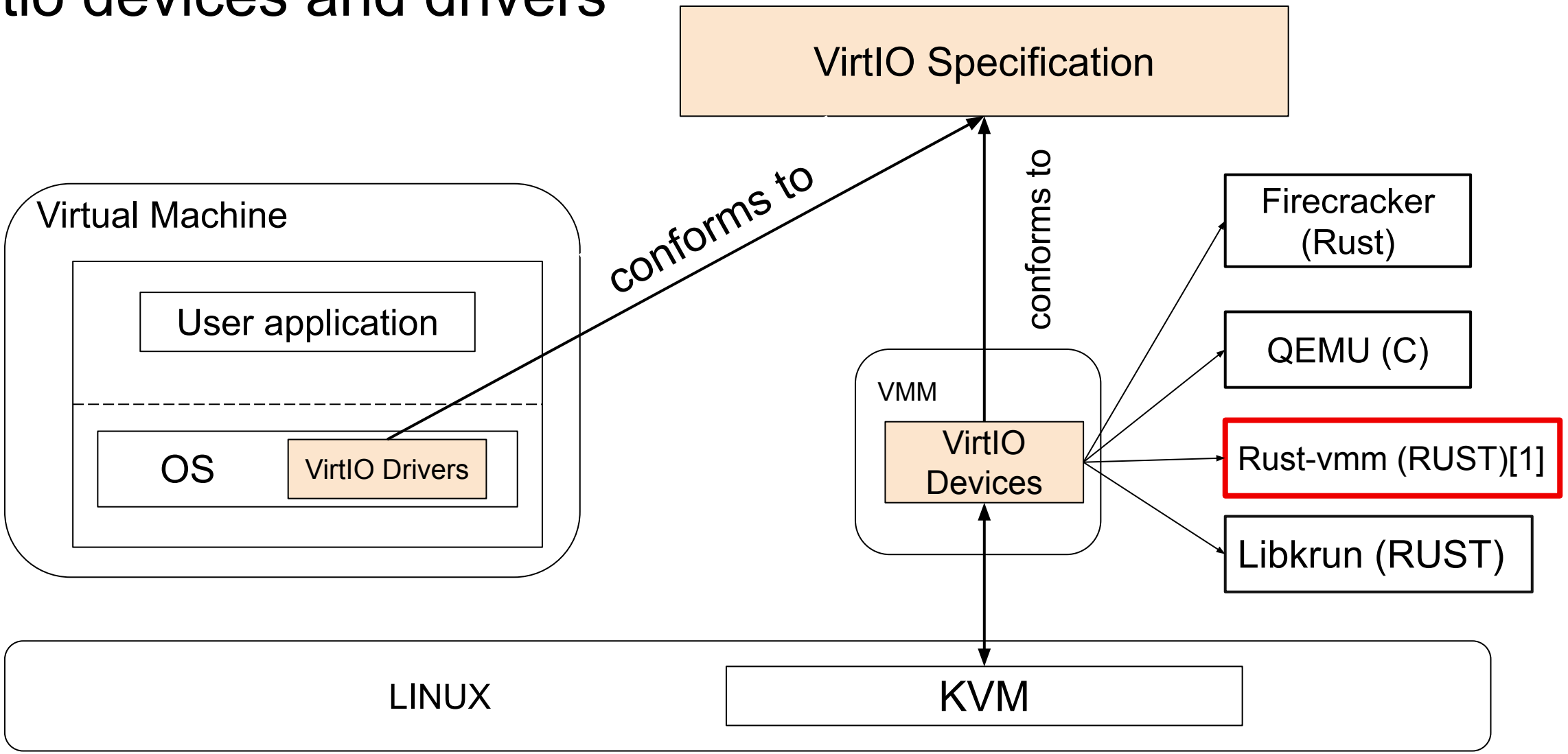
- Software Engineer at Red Hat
- Work on Operating System development, virtualization and formal methods
- Participate on the virtio-comment mailing list as reviewer and co-chair
- https://github.com/MatiasVara

Virtio devices and drivers



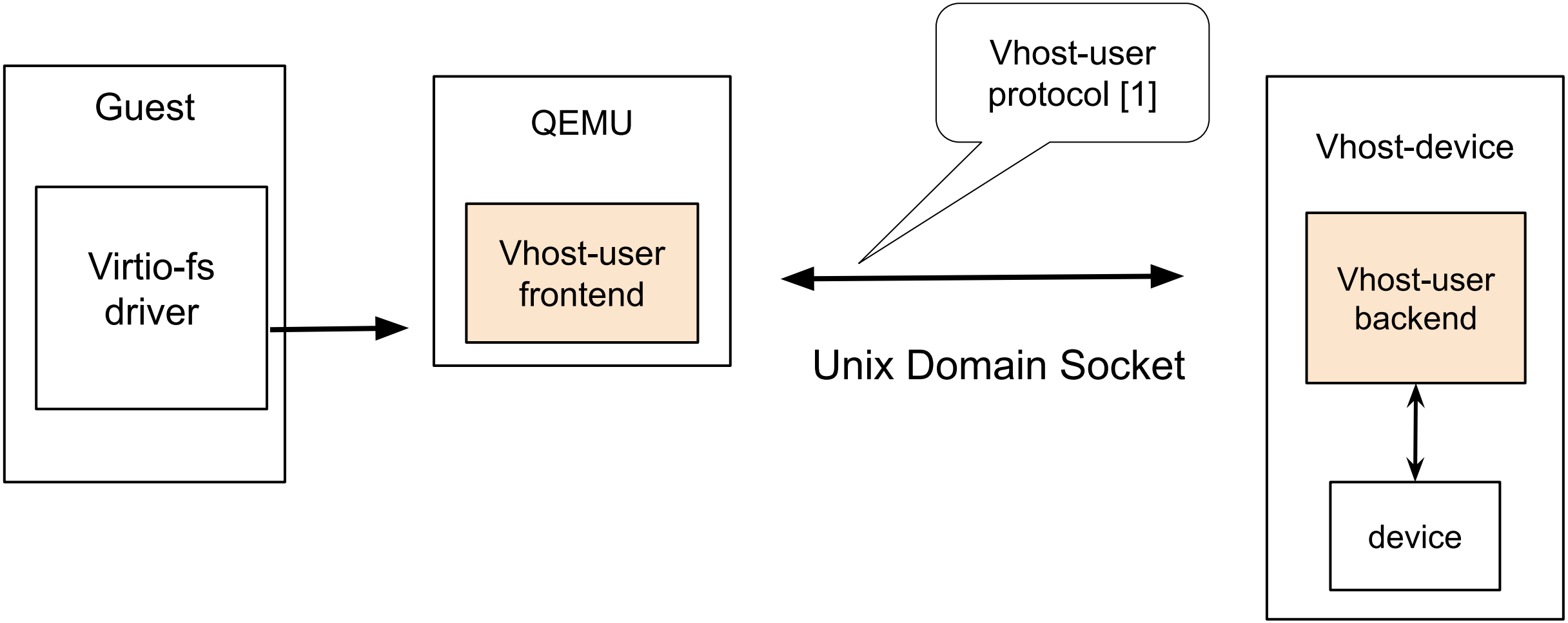
[1] https://github.com/rust-vmm/vhost-device

Virtio devices and drivers



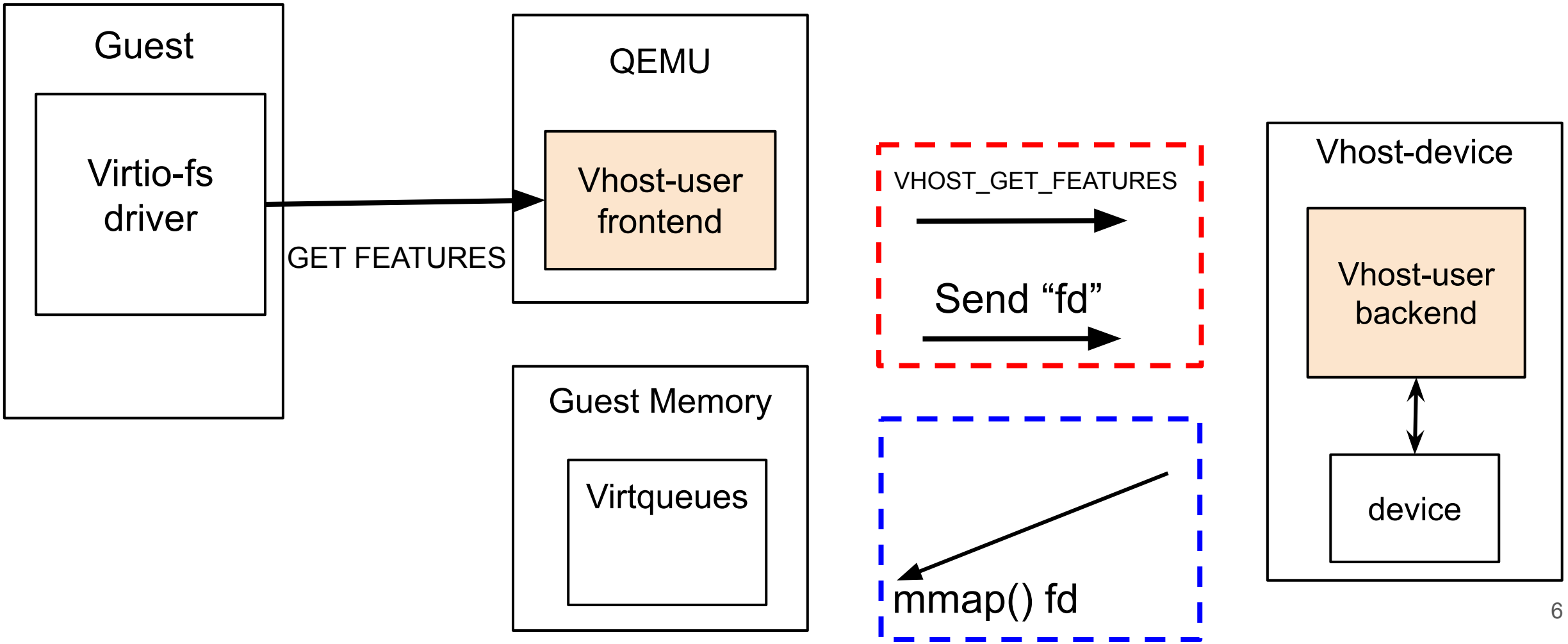
[1] https://github.com/rust-vmm/vhost-device

Vhost-user-devices in rust-vmm

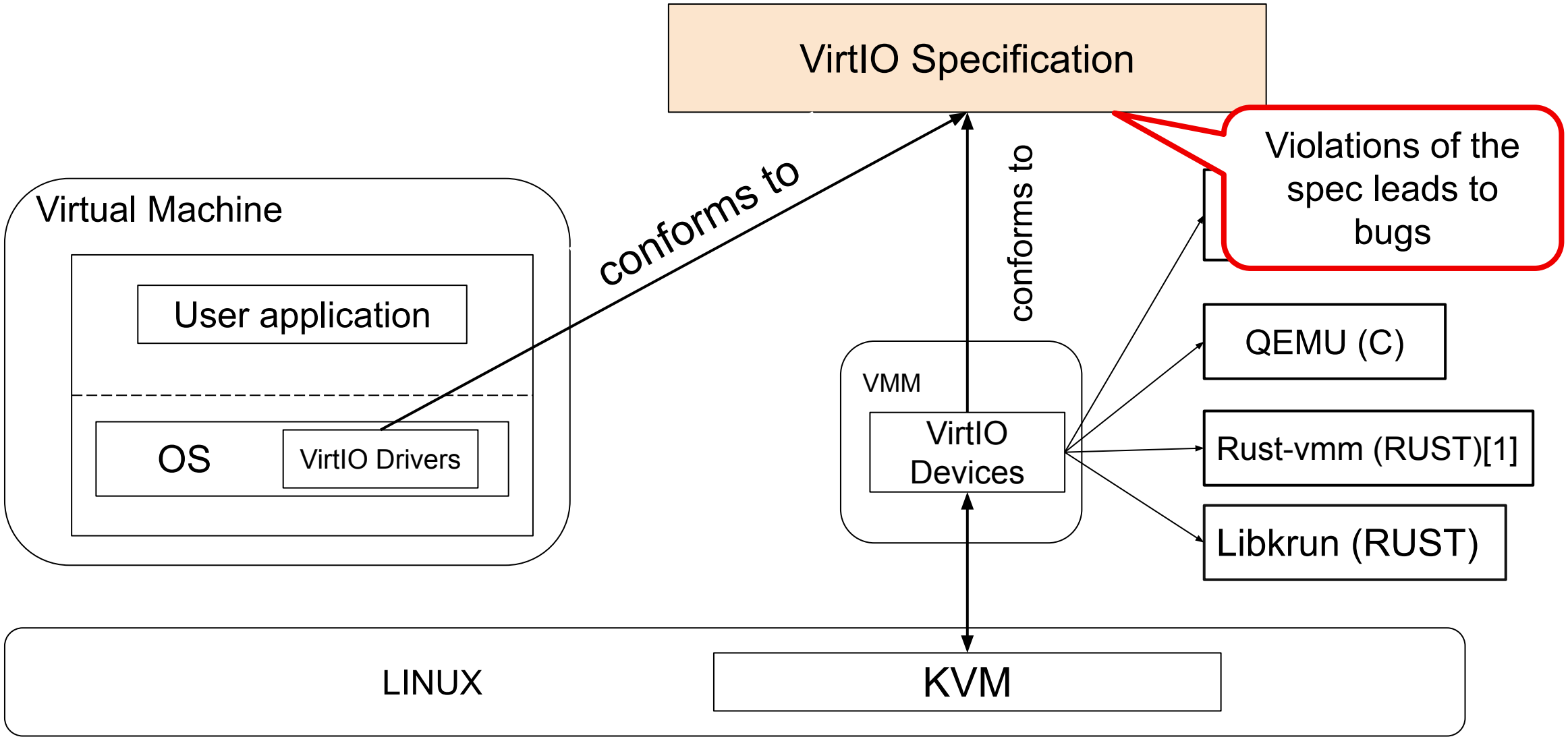


[1] https://qemu-project.gitlab.io/qemu/interop/vhost-user.html

Vhost-user-devices in rust-vmm



Virtio devices and drivers

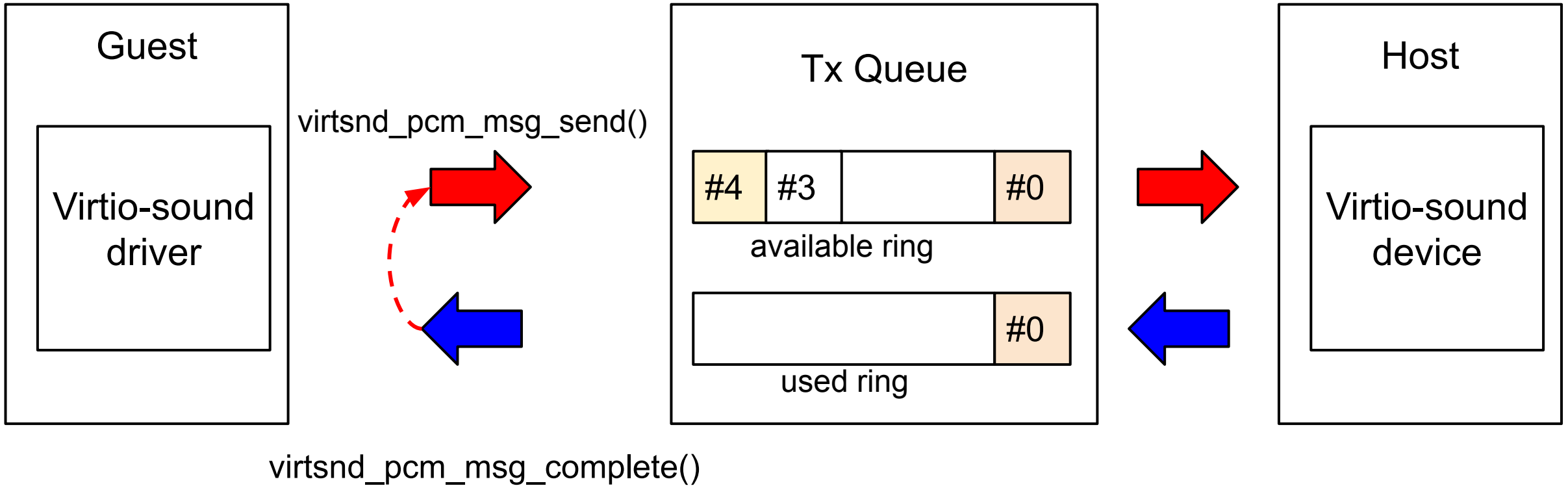


[1] https://github.com/rust-vmm/vhost-device

Examples of violation of the specification

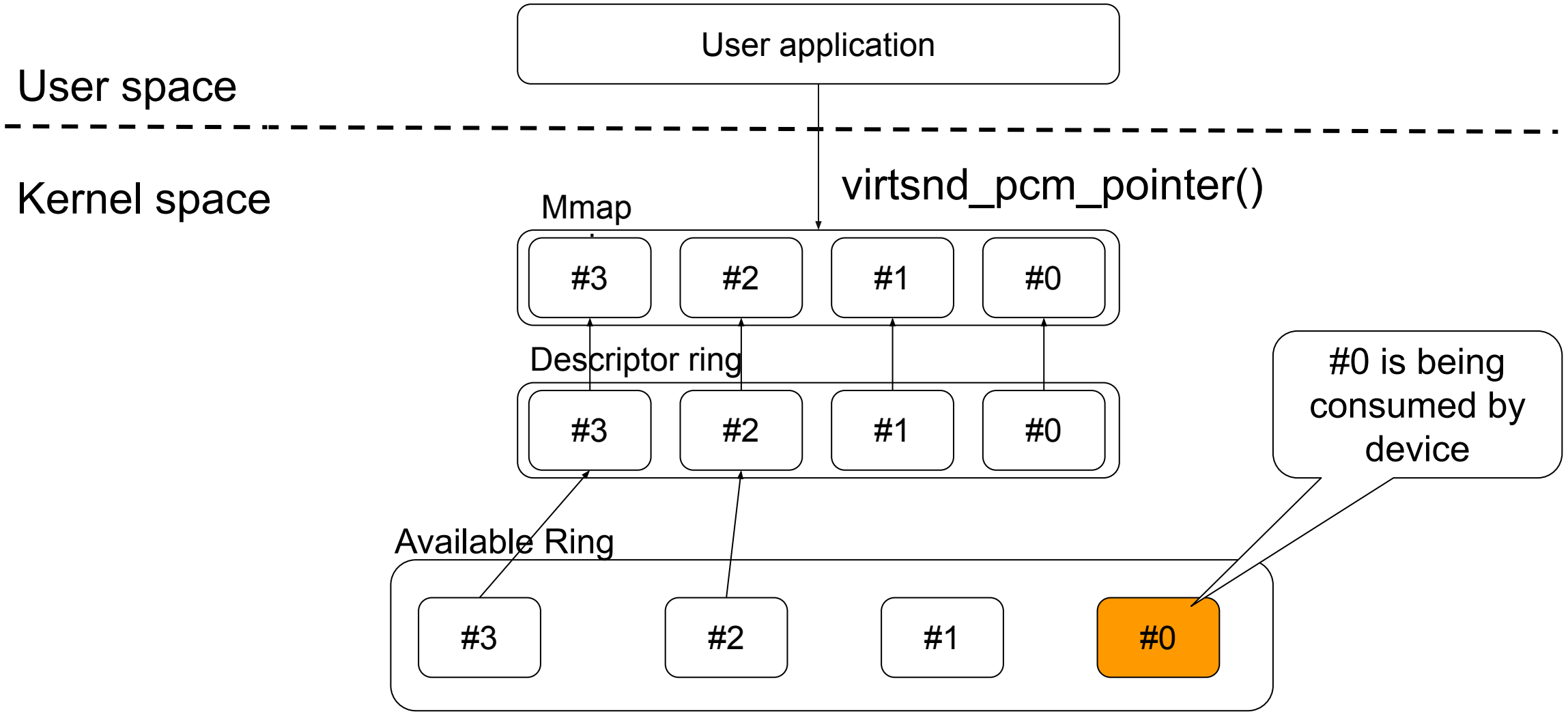
1. Virtio-sound driver
   - See https://lore.kernel.org/all/ZQHPeD0fds9sYzHO@pc-79.home/T/
2. Vhost-user
   - See https://lists.nongnu.org/archive/html/qemu-devel/2023-08/msg05680.html

Violation of the specification in virtio-sound driver [1]

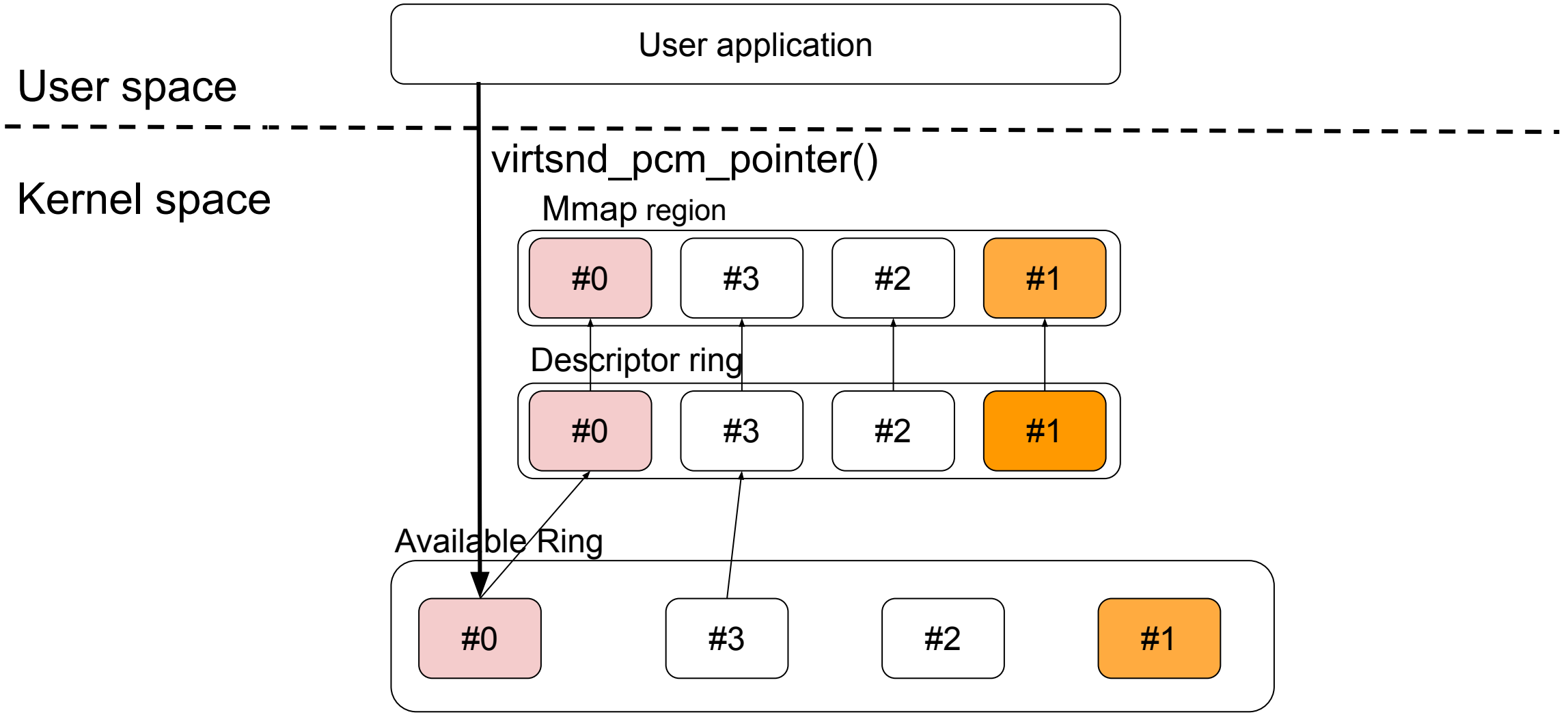


[1] Discusion at https://lore.kernel.org/all/ZQHPeD0fds9sYzHO@pc-79.home/T/

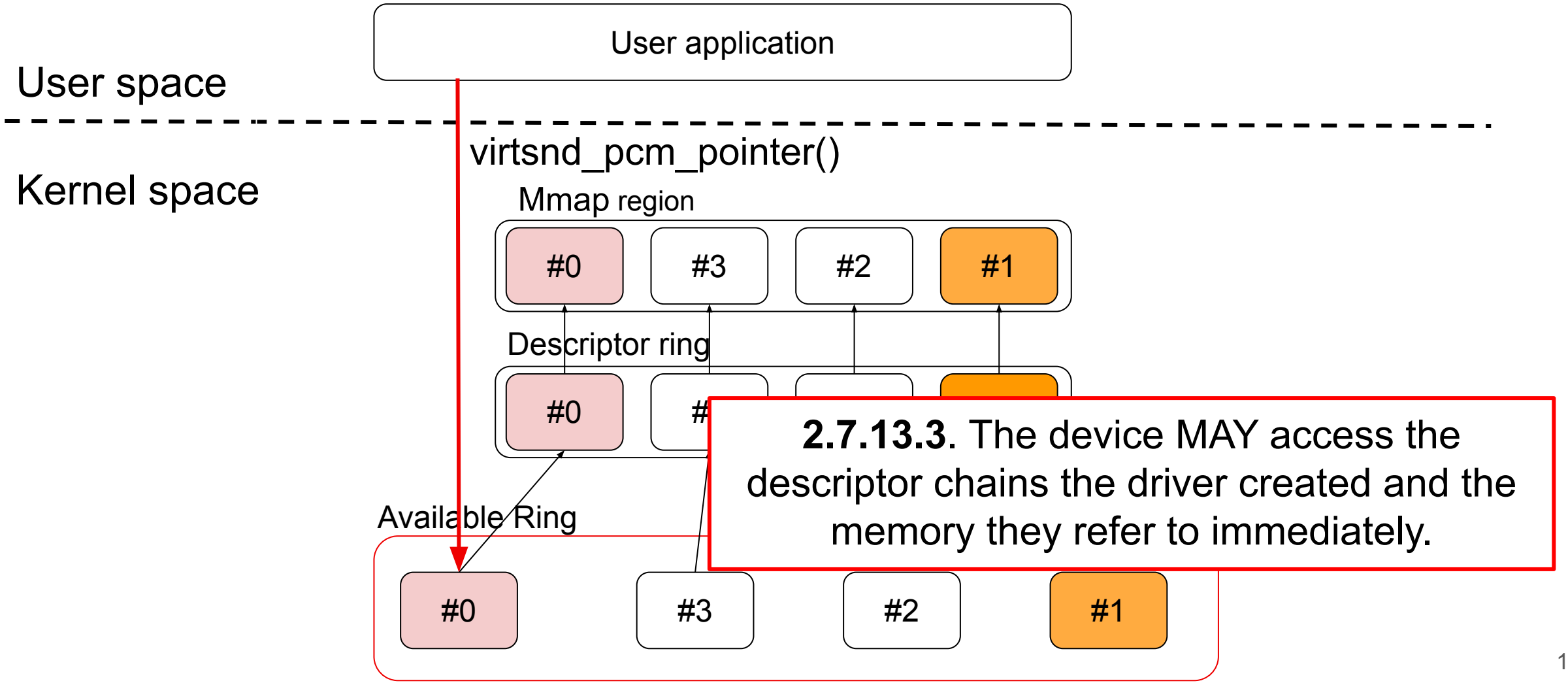
Violation of the specification in virtio-sound driver



Violation of the specification in virtio-sound driver



Violation of the specification in virtio-sound driver



2.7.13.3. The device MAY access the descriptor chains the driver created and the memory they refer to immediately.

Violation of the specification in virtio-sound driver

[15.611647] virtsnd_pcm_msg_send: adding buffer #1 to avail

[15.611854] virtsnd_pcm_msg_send: adding buffer #2 to avail

[15.612037] virtsnd_pcm_msg_send: adding buffer #3 to avail

[15.612240] virtsnd_pcm_msg_send: adding buffer #4 to avail

[15.612758] virtsnd_pcm_pointer: return pointer to buffer #1

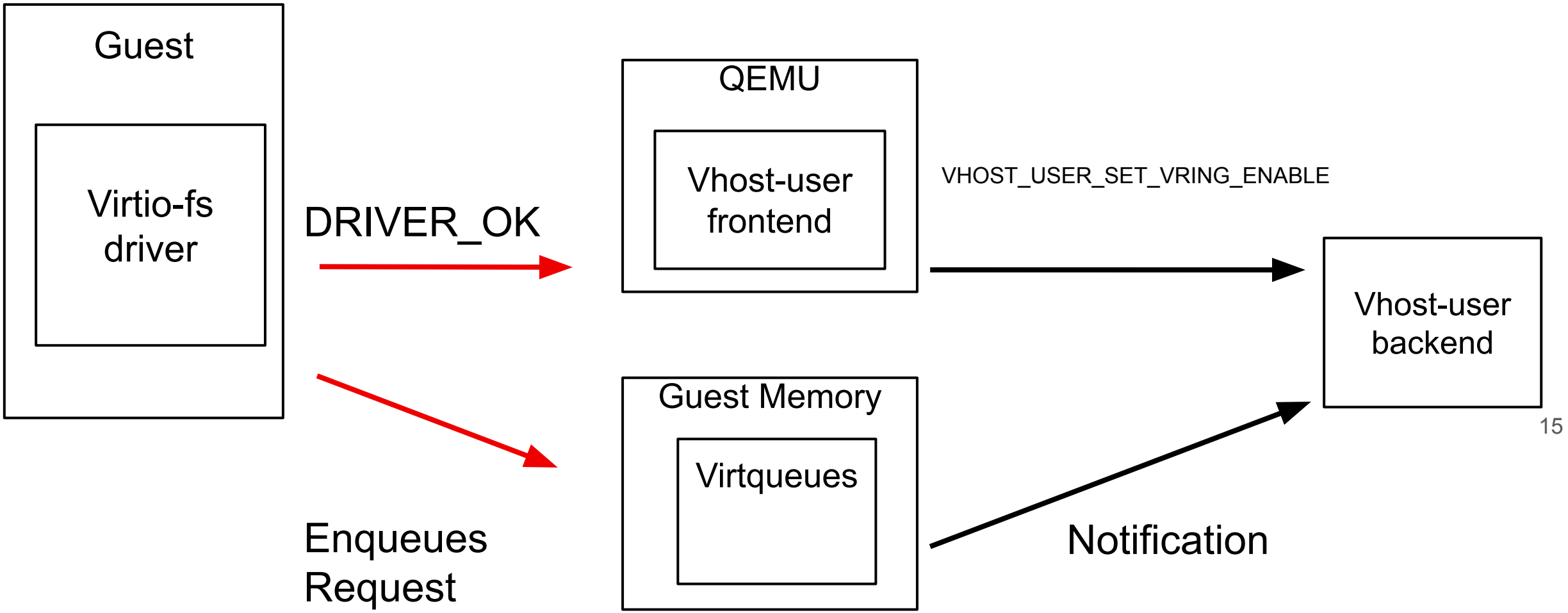
[15.713371] virtsnd_pcm_msg_complete: interruption, buffer #1 at consume

How can we detect this violation of the partial order?

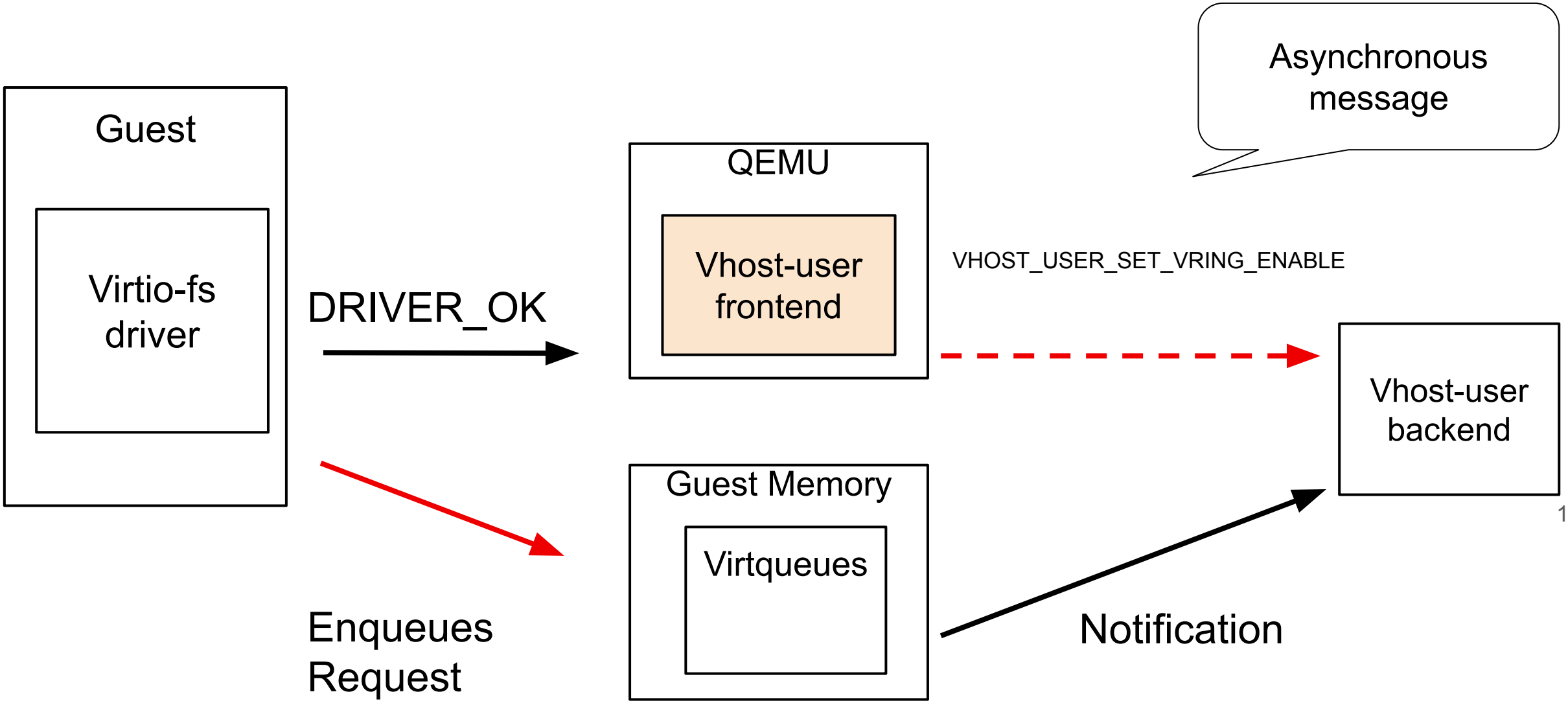
Examples of violation of the specification

1. ~~Virtio-sound driver~~
    a. See https://lore.kernel.org/all/ZQHPeD0fds9sYzHO@pc-79.home/T/
2. Vhost-user
    a. See https://lists.nongnu.org/archive/html/qemu-devel/2023-08/msg05680.html

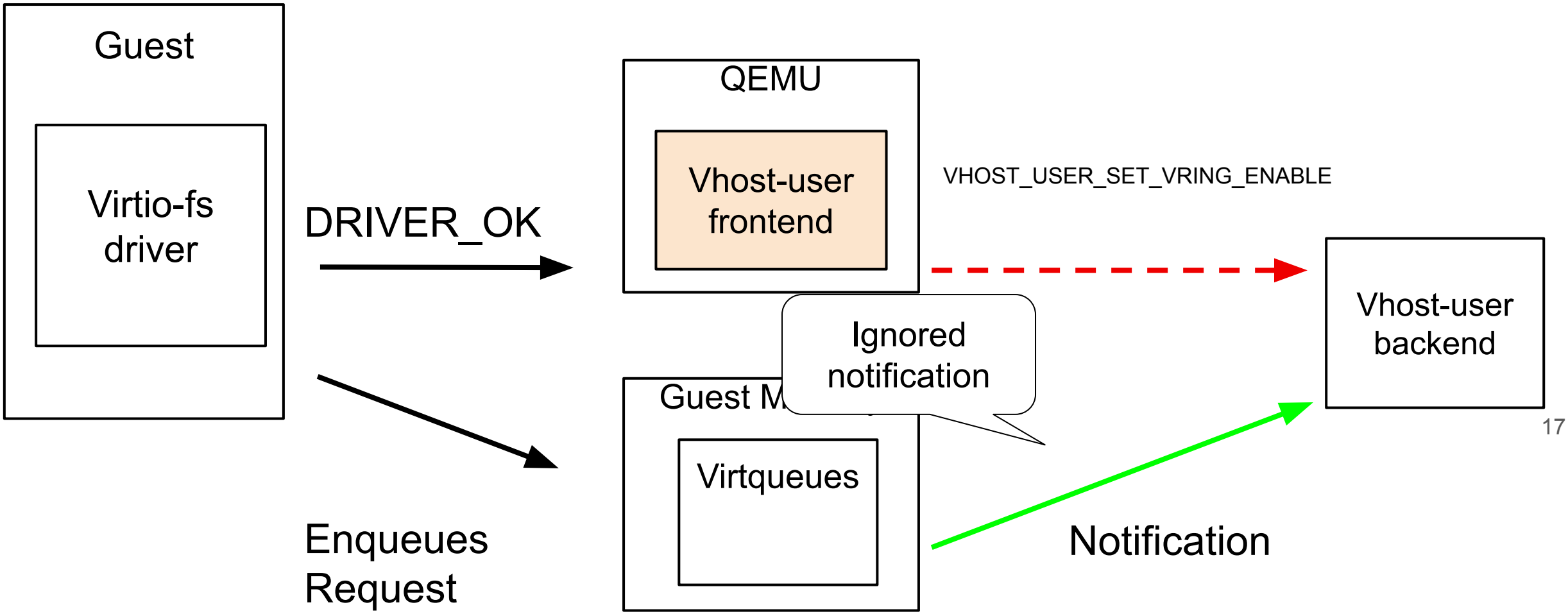
Violation of the specification in vhost-user



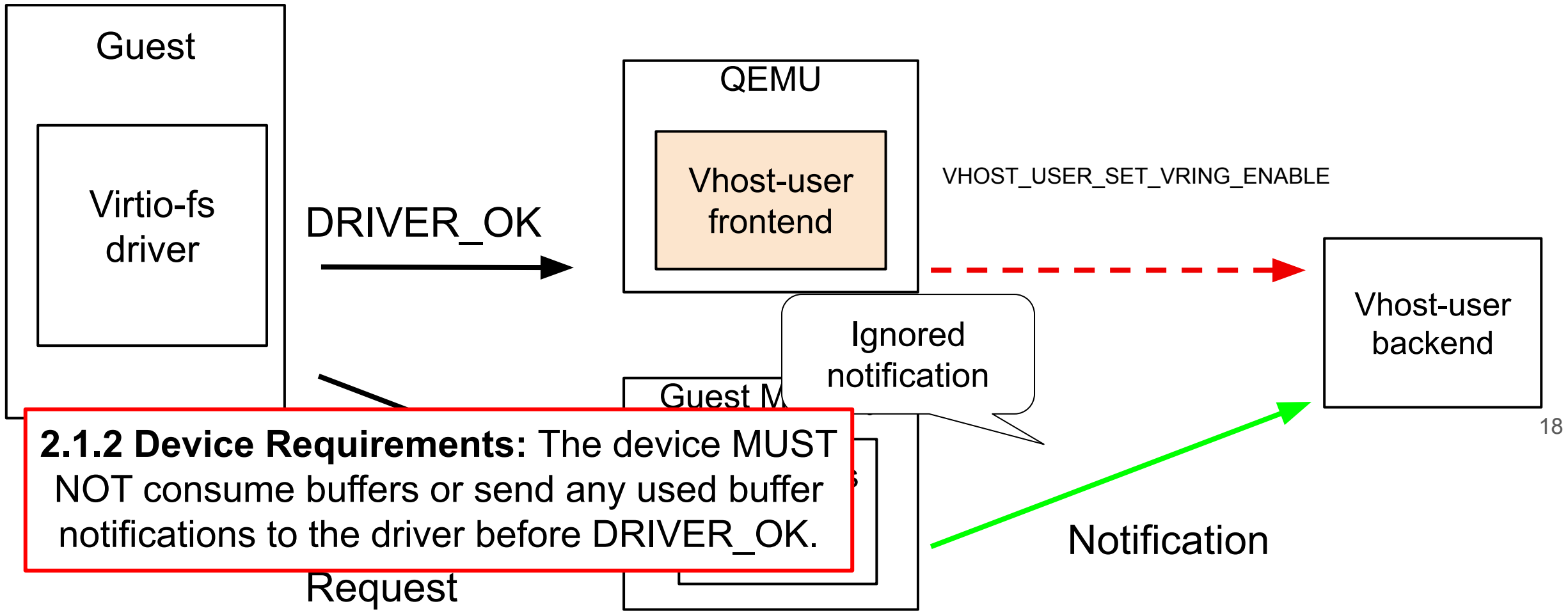
Violation of the specification in vhost-user



Violation of the specification in vhost-user



Violation of the specification in vhost-user



Violation of the specification in vhost-user

INVALID TRACE

[15.612037] handled event

[15.611647] set_vring_enable

[15.611854] set_vring_enable

VALID TRACE

[15.612037] set_vring_enable

[15.611647] set_vring_enable

[15.611854] handled event

How can we detect this violation of the partial order?

Tools to check conformance of the implementation

- Kani [1]
- Tracing-formal [2]

[1] https://github.com/model-checking/kani/
[2] https://github.com/MatiasVara/tracing-formal

Proof for the notification suppression mechanism

```rust
#[kani::proof]
fn verify_spec_2_7_7_2() {
    let ProofContext(mut queue, mem) = kani::any();
    let num_added_old = queue.num_added.0;
    let needs_notification = queue.needs_notification(&mem);

    if !queue.event_idx_enabled {
        assert!(needs_notification.unwrap());
    } else {
        if Wrapping(queue.used_event(&mem, Ordering::Relaxed).unwrap())
            == std::num::Wrapping(queue.next_used - Wrapping(1))
            && num_added_old > 0
        {
            assert!(needs_notification.unwrap());

            kani::cover!();
        }
```

See PR at https://github.com/rust-vmm/vm-virtio/pull/324

Proof for the notification suppression mechanism

```
#[kani::proof]
fn verify_spec_2_7_7_2() {
    let ProofContext(mut queue, mem) = kani::any();
    let num_added_old = queue.num_added.0;
    let needs_notification = queue.needs_notification(&mem);

    if !queue.event_idx_
        assert!(needs_no
    } else {
        if Wrapping(queu
            == std::num:
            && num_added
        {
            assert!(need

            kani::cover!();
        }
```

```
fn needs_notification<M: GuestMemory>(&mut self, mem: &M)
-> Result<bool, Error> {
...
        return Ok(used_idx - used_event - Wrapping(1) <
    used_idx - old);
}
```

See PR at https://github.com/rust-vmm/vm-virtio/pull/324

Proof for the notification suppression mechanism

```
#[kani::proof]
fn verify_spec_2_7_7_2() {
    let ProofContext(mut queue, mem) = kani::any();
    let num_added_old = queue.num_added.0;
    let needs_notification = queue.needs_notification(&mem);

    if !queue.event_idx_
        assert!(needs_no
    } else {
        if Wrapping(queu
            == std::num:
            && num_added
        {
            assert!(need

            kani::cover!();
        }
```

```
fn needs_notification<M: GuestMemory>(&mut self, mem: &M)
-> Result<bool, Error> {
...
        return Ok(used_idx - used_event - Wrapping(1) <
    used_idx - old);
        return Ok( used_event + Wrapping(1) > old);
}
```

See PR at https://github.com/rust-vmm/vm-virtio/pull/324

Tools to check conformance of the implementation

- ~~Kani [1]~~
- Tracing-formal [2]

[1] https://github.com/model-checking/kani/
[2] https://github.com/MatiasVara/tracing-formal

Instrumenting traces with tracing-formal

```
fn set_vring_enable(&mut self, index: u32, enable:
bool) -> VhostUserResult<()> {
        . . .
        vring.set_enabled(enable);
        event!(Level::INFO, event =
"set_vring_enable”);
        Ok(())
```

```
   fn handle_event(
        &self,
        vrings: &[VringRwLock],
 -> IoResult<()> {
        let vring = &vrings
            .get(device_event as usize)
            event!(Level::INFO, event = "handle_event"),
```

Instrumenting traces with tracing-formal

```
fn set_vring_enable(&mut self, index: u32, enable:
bool) -> VhostUserResult<()> {
        . . .
        vring.set_enabled(enable);
        event!(Level::INFO, event =
"set_vring_enable”);
        Ok(())
```

```
   fn handle_event(
        &self,
        vrings: &[VringRwLock],
 -> IoResult<()> {
        let vring = &vrings
            .get(device_event as usize)
            event!(Level::INFO, event = "handle_event"),
```

```
fn main() {
    let alternates: Alternates =
Alternates::new("handle_event",
"set_vring_enable");

    let subscriber =
TracingFormal::new(vec![alterna
tes]);

tracing::subscriber::set_global
_default(subscriber).expect("Fa
iled to set subscriber");
```

Questions? Suggestions?

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make
Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat