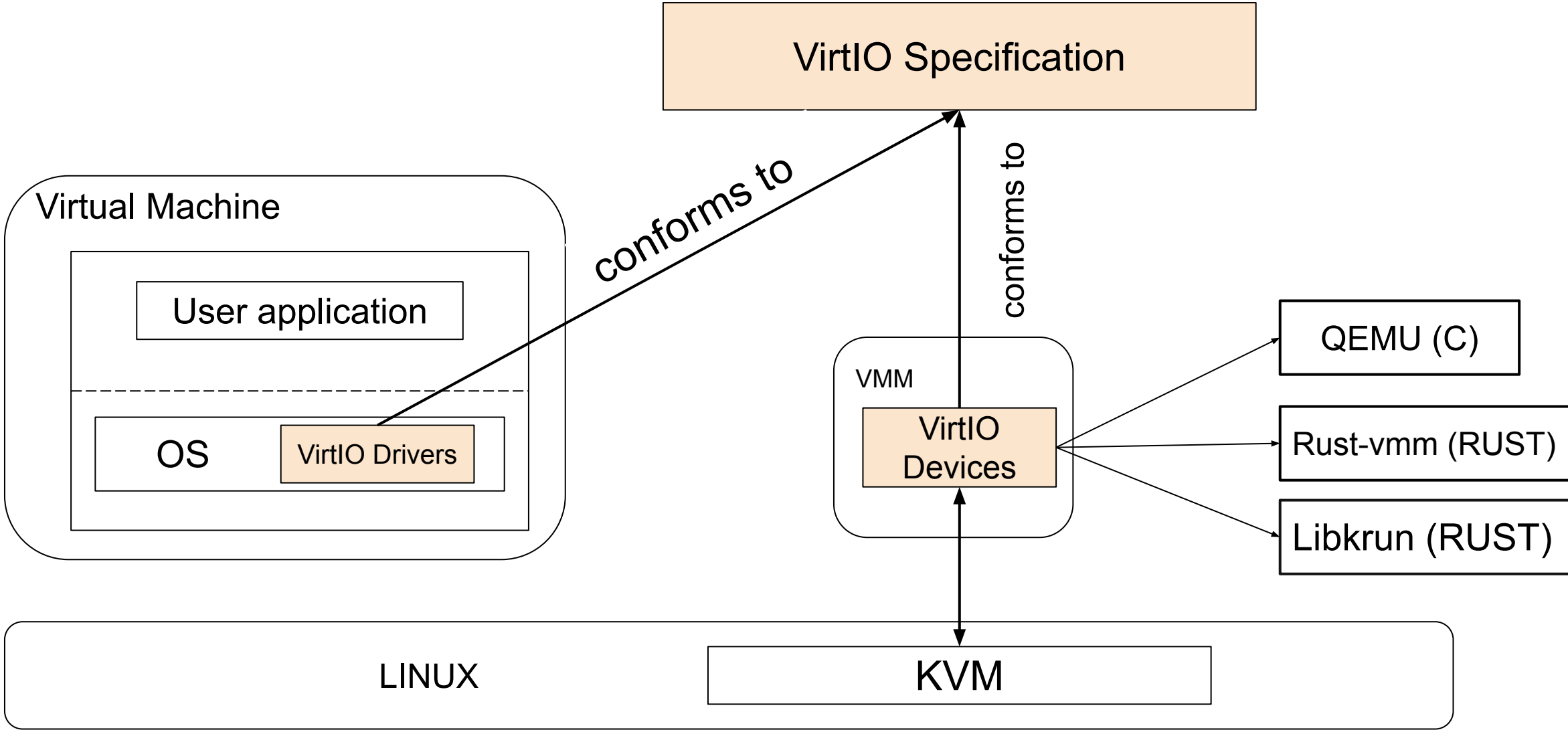
Verifying conformance between VirtIO implementations and the specification

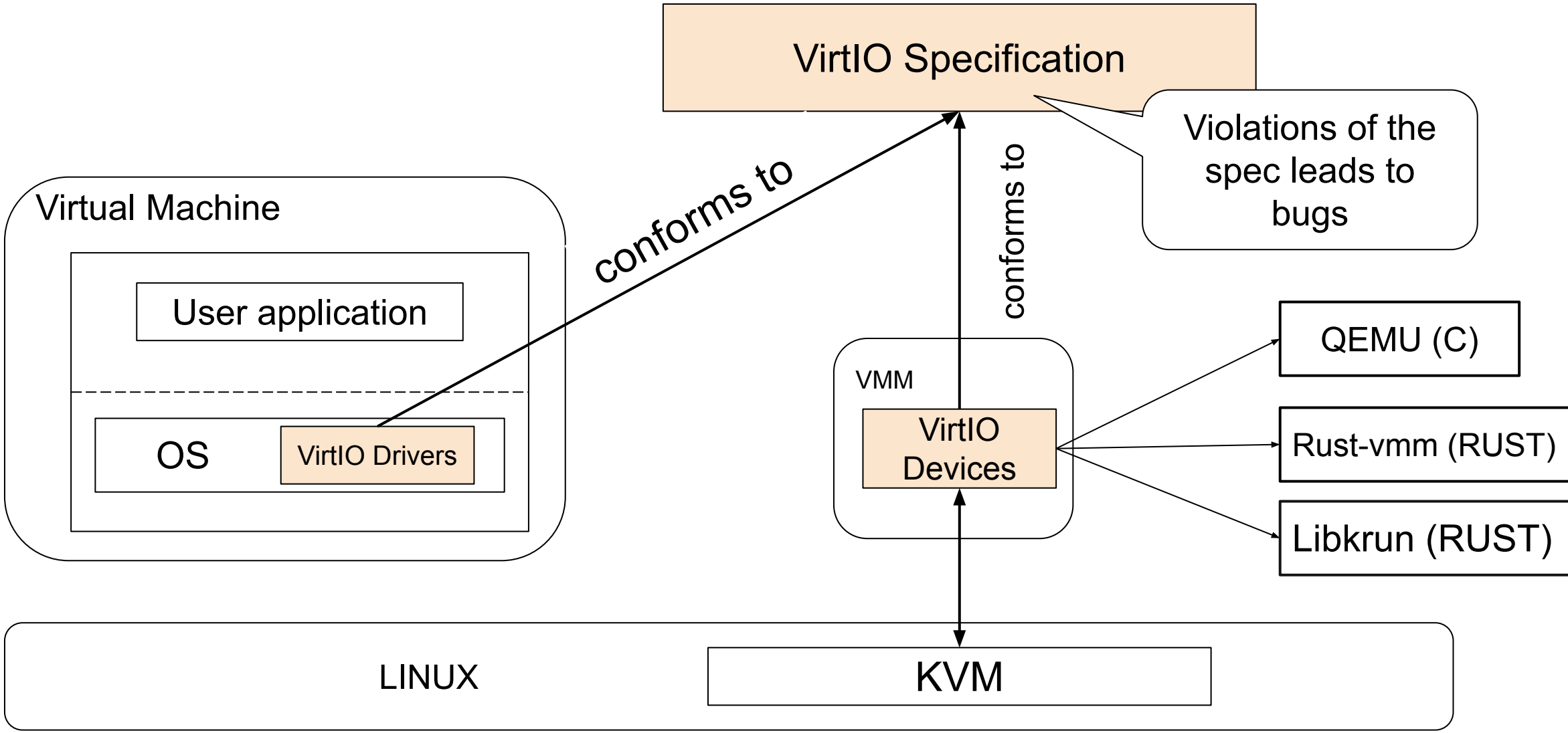
Matias Vara Larsen
mvaralar@redhat.com

Red Hat

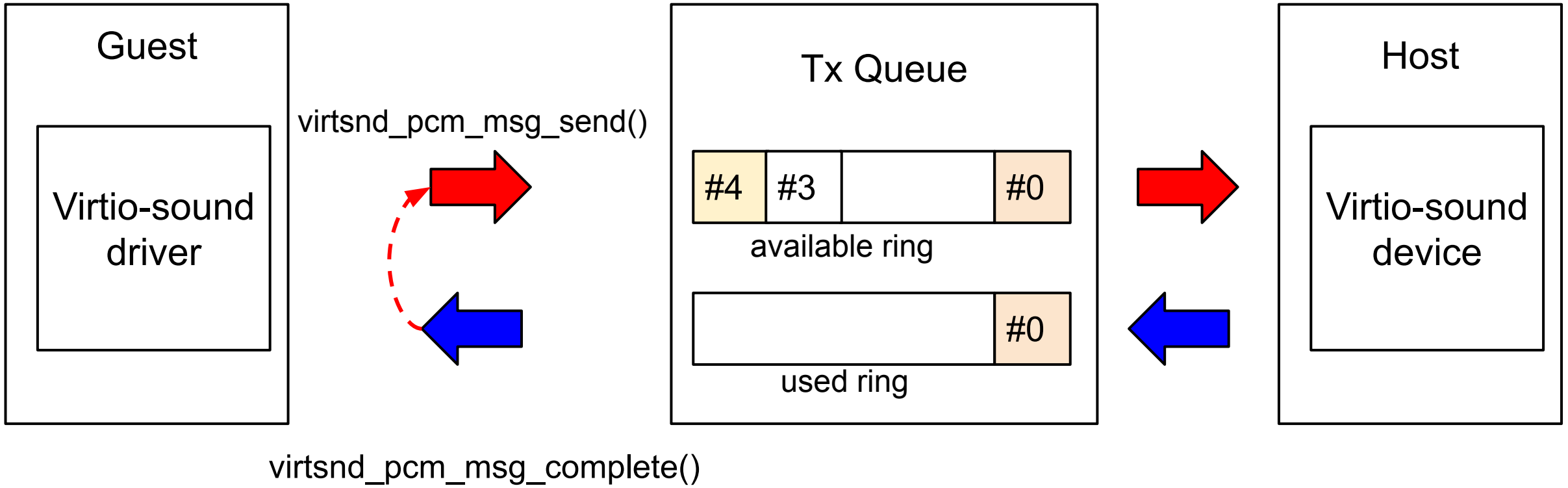
Virtio devices and drivers



Virtio devices and drivers

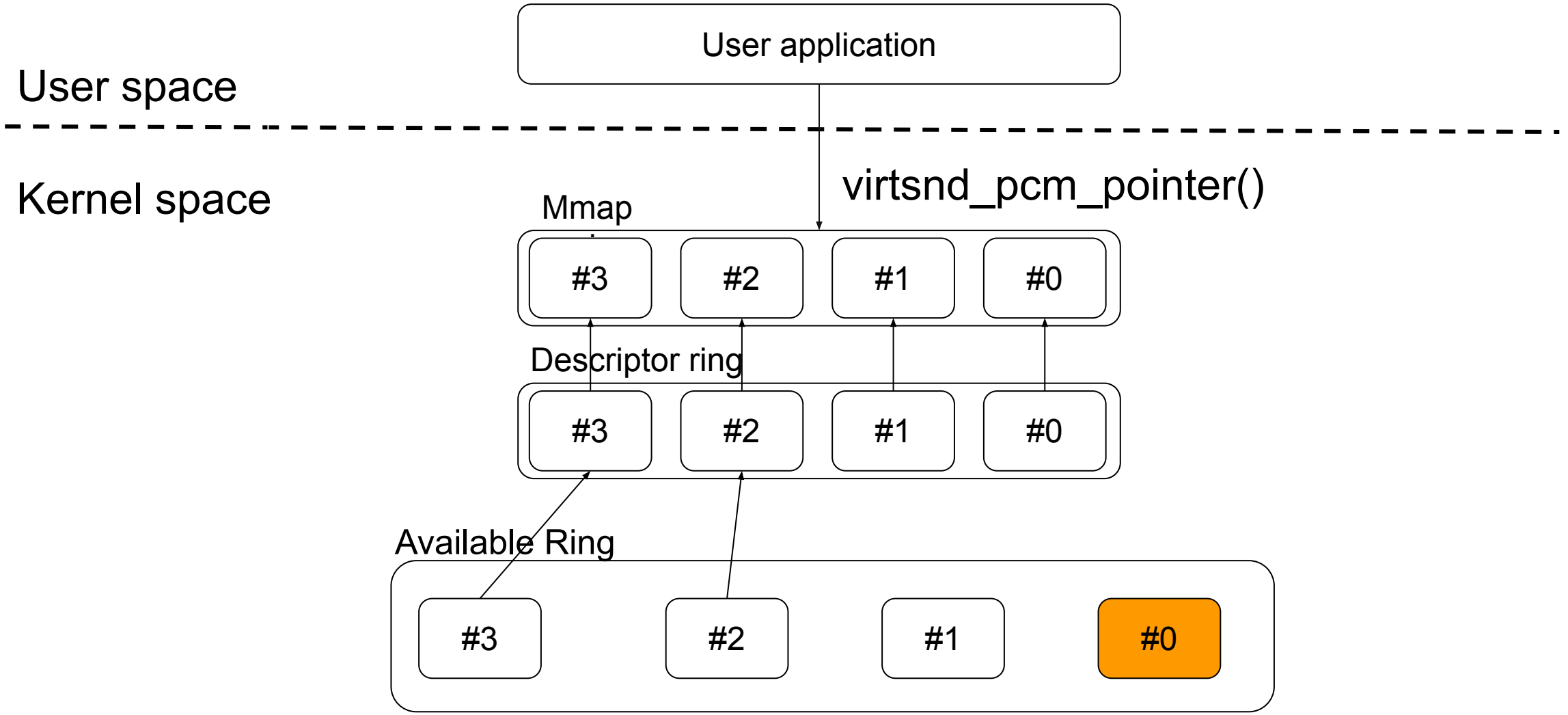


Violation of the specification in virtio-sound driver [1]

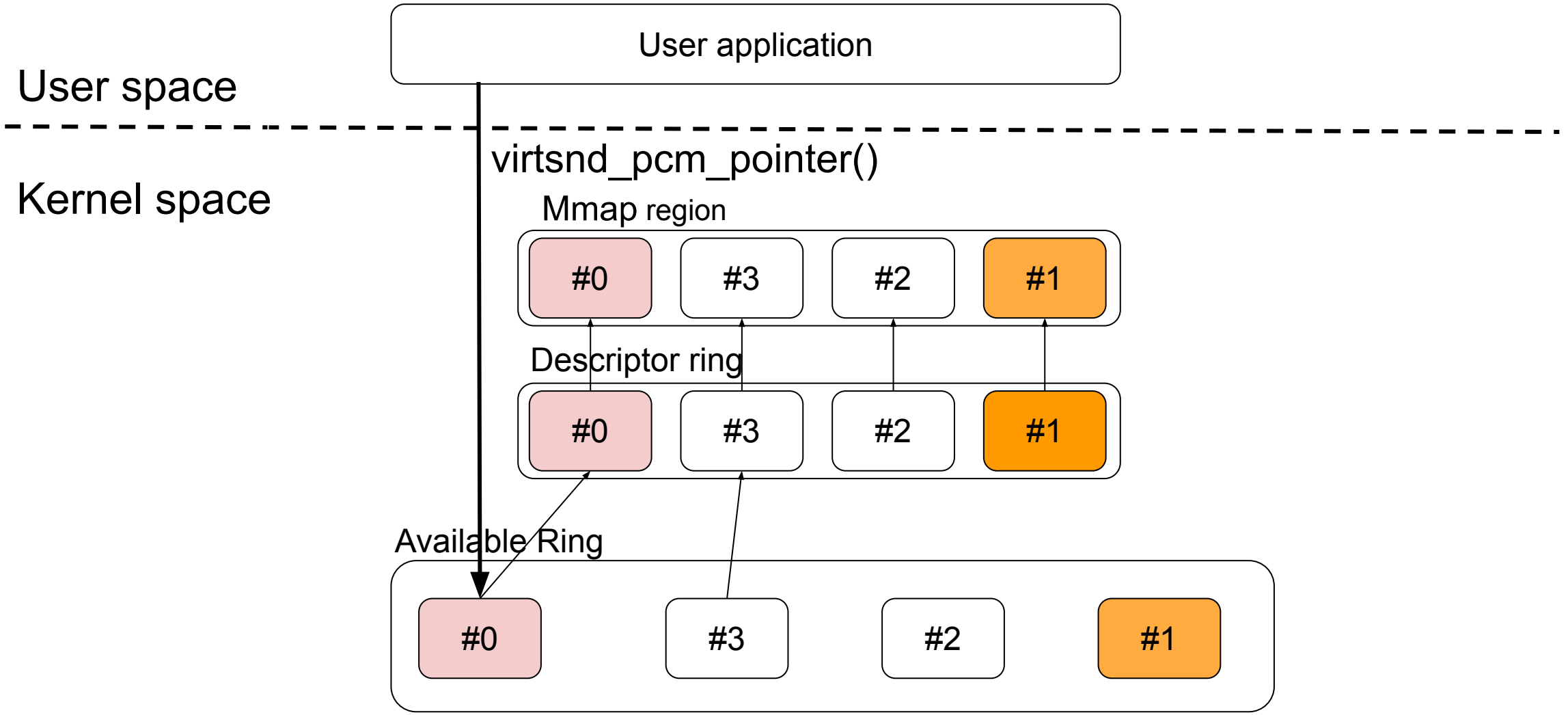


[1] Discusion at https://lore.kernel.org/all/ZQHPeD0fds9sYzHO@pc-79.home/T/

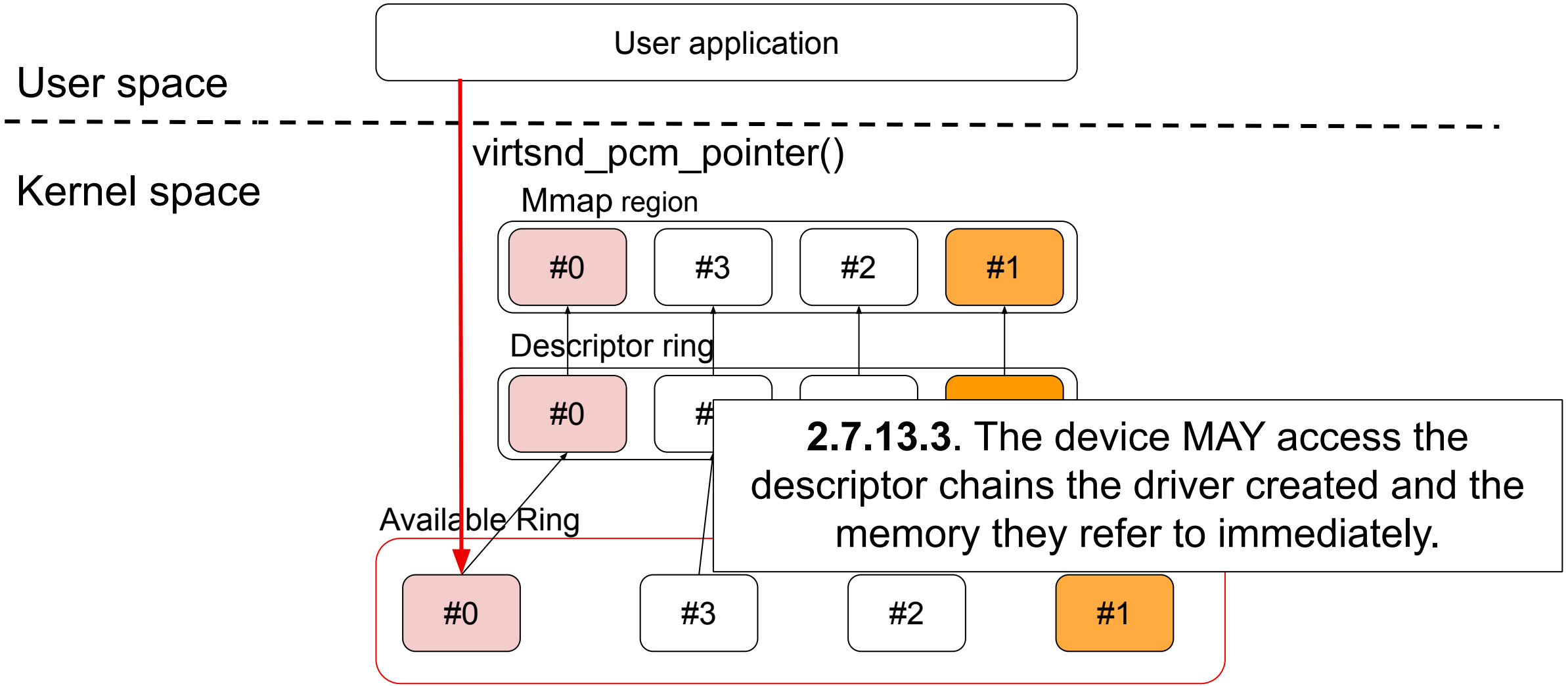
Violation of the specification in virtio-sound driver



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

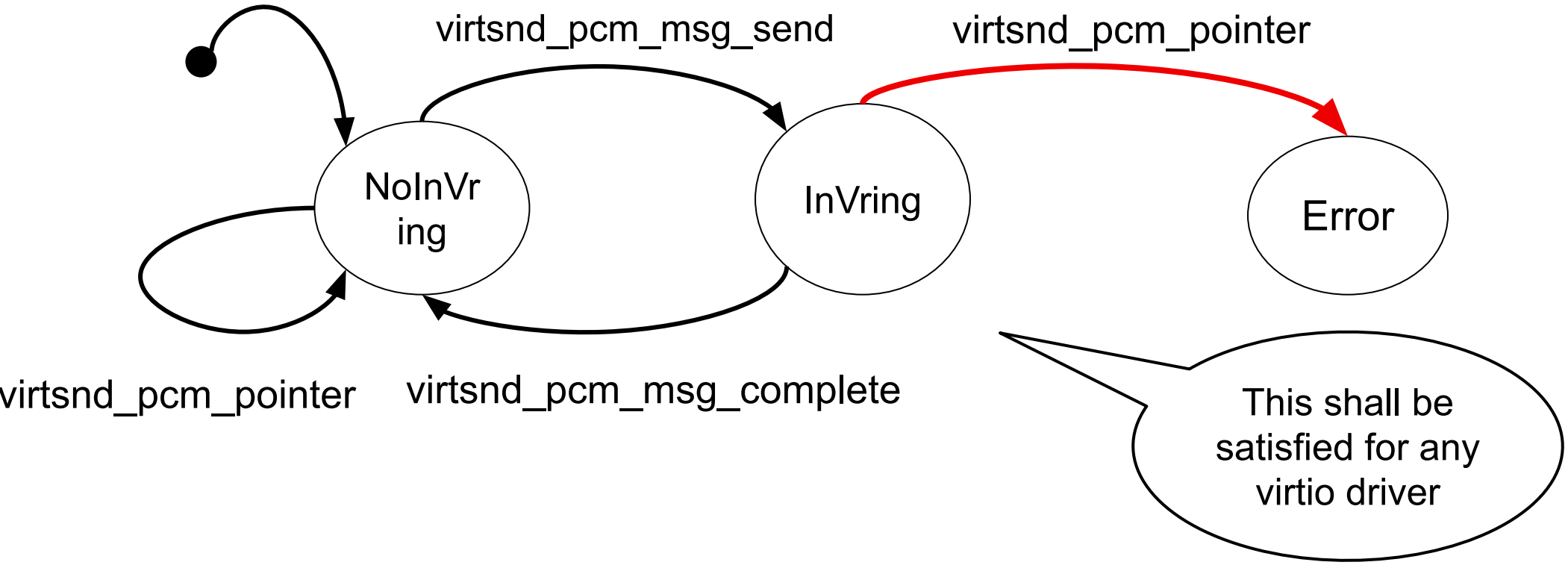
How can we detect this violation of the partial order before?

Proposal

1. Defining with less ambiguity the specification, i.e., using a formal language
2. Building runtime observers to detect violations
3. Extending the VirtIO specification with a formal specification that everyone can “include”

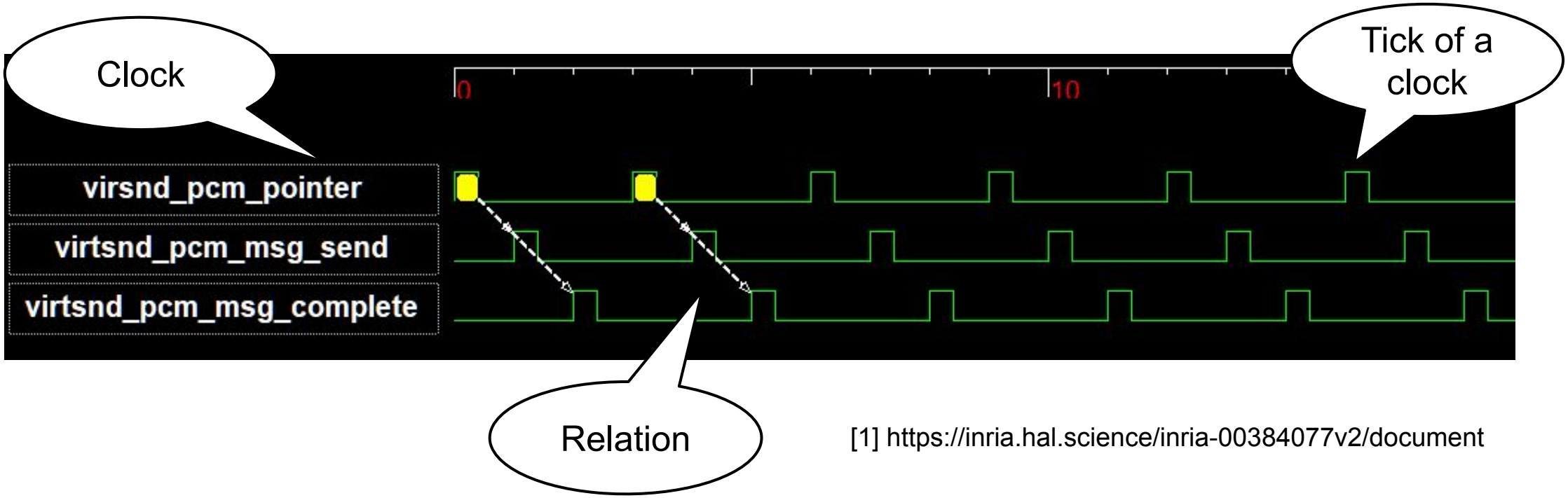
Defining without ambiguity the specification

1. By using Finite Automata



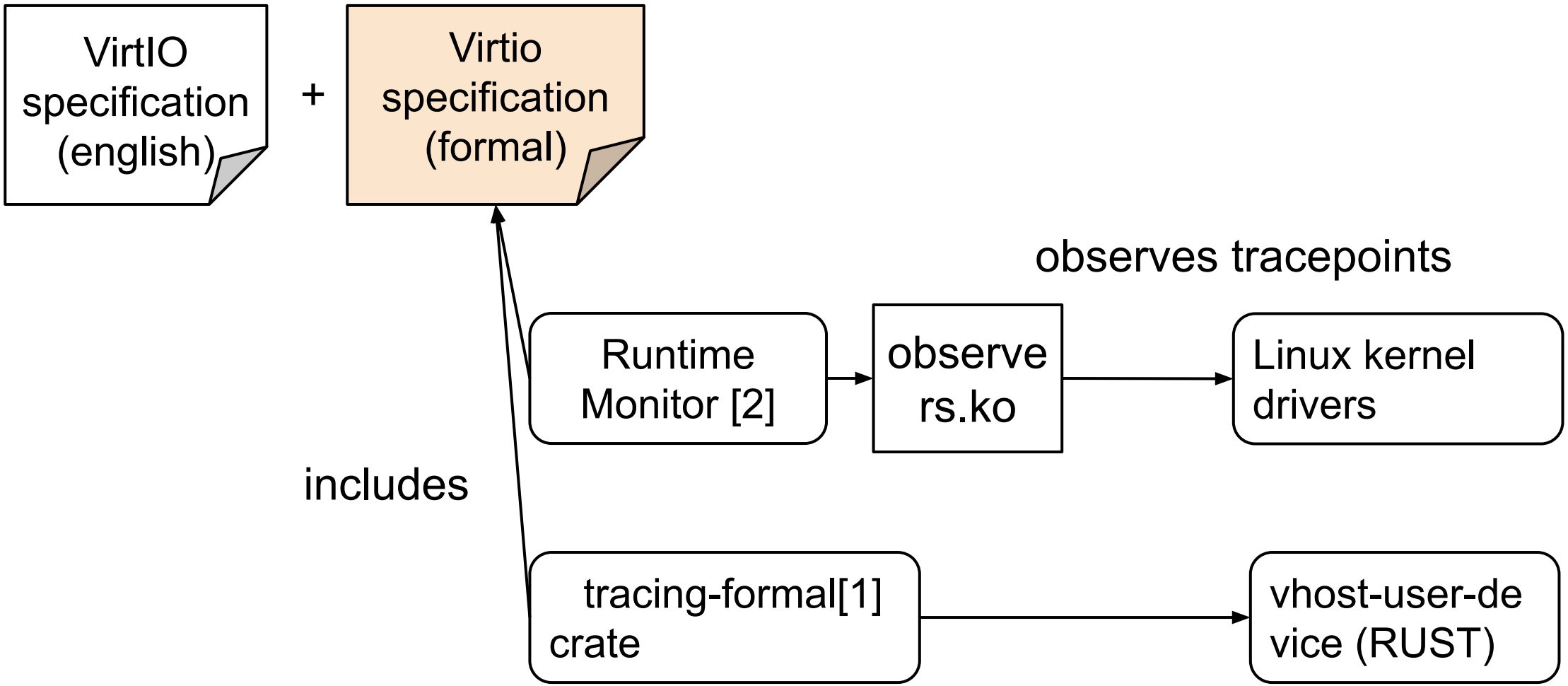
Defining without ambiguity the specification

1. By using Finite Automata or the Clock Constraint Specification Language [1]
   a. virtsnd_pcm_msg_send() **alternatesWith** virtsnd_pcm_msg_complete()
   b. virtsnd_pcm_pointer() **alternatesWith** virtsnd_pcm_msg_send()
   c. virtsnd_pcm_pointer() **alternatesWith** virtsnd_pcm_msg_complete()



[1] https://inria.hal.science/inria-00384077v2/document

Using the formal definition to build observers



[2] https://docs.kernel.org/trace/rv/runtime-verification.html

[1] https://github.com/MatiasVara/tracing-formal

Using the formal definition to build observers

```rust
#[instrument(fields(event = "do_first"))]
fn first() {
    println!("do first()");
}

#[instrument(fields(event = "do_second"))]
fn second() {
    println!("do second()");
}

fn main() {
    let alternates: Alternates = Alternates::new("do_first", "do_second");
    let subscriber = TracingFormal::new(vec![alternates]);

    tracing::subscriber::set_global_default(subscriber).expect("Failed to set subscriber");

    first();
    second();
    second(); // this is a violation in the partial order
}
```

1. Annotation on function definition

2. Constraint instantiation

3. Tracing subscriber instantiation

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