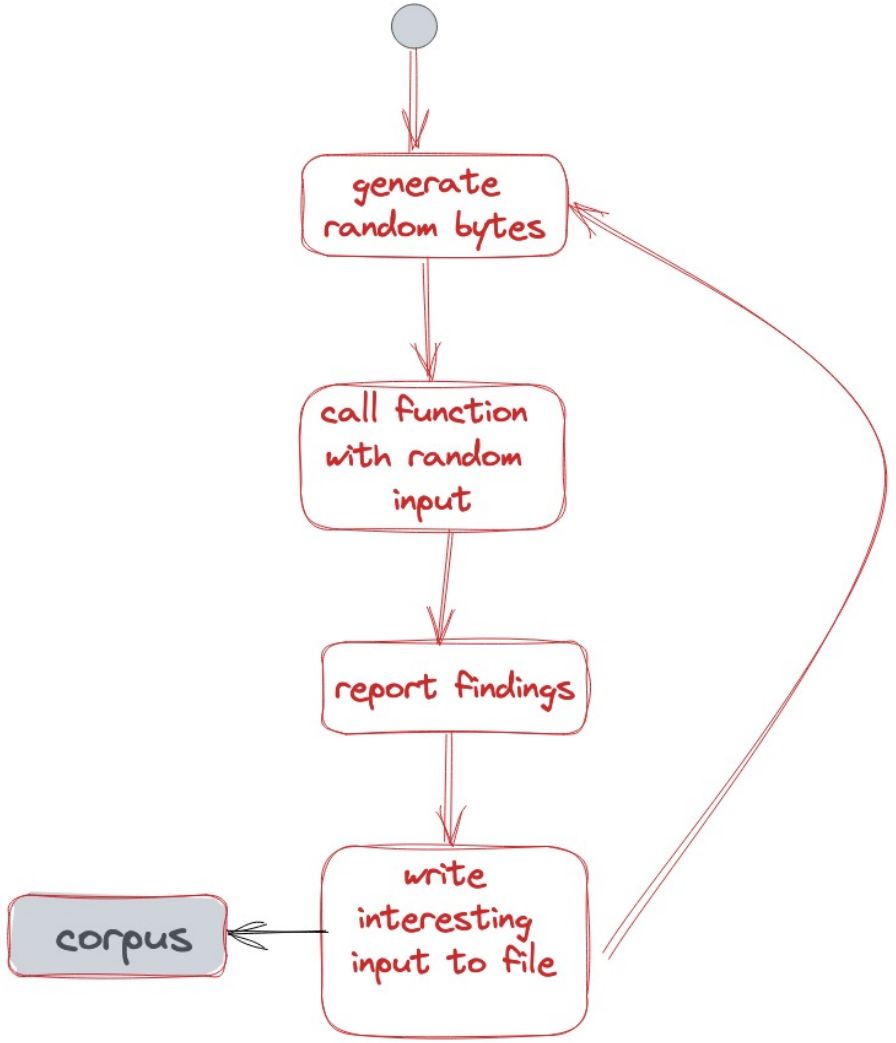
FUZZING DEVICE MODELS IN RUST

Common Pitfalls

- Andreea Florescu, fandree@amazon.com -

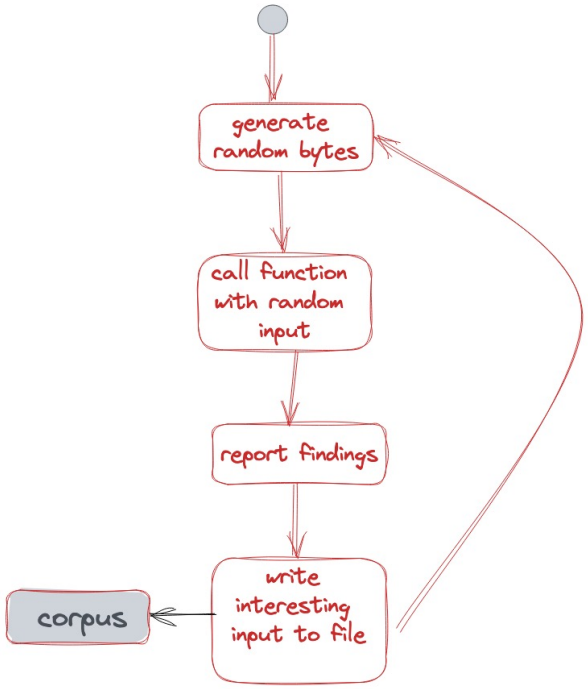
FUZZING - QUICK INTRO

- Automated testing
- Check how system behaves with unexpected input
- Findings: crashes, hangs, timeouts



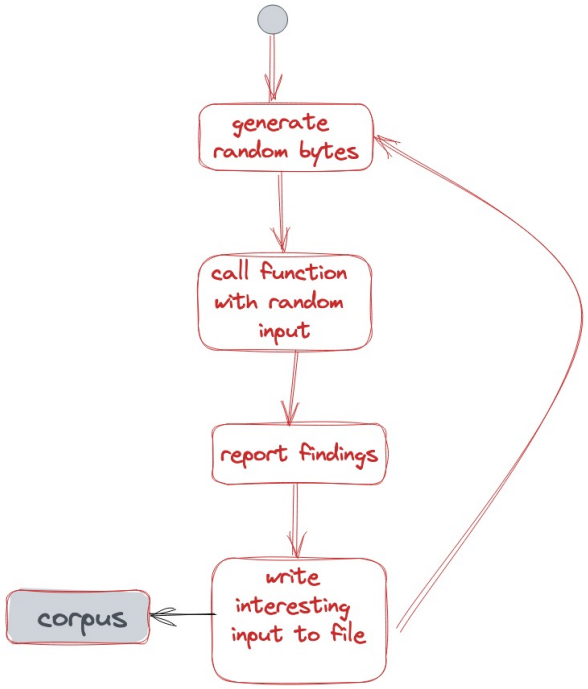
FUZZING - START WITH CORPUS

First Run: Generate Corpus

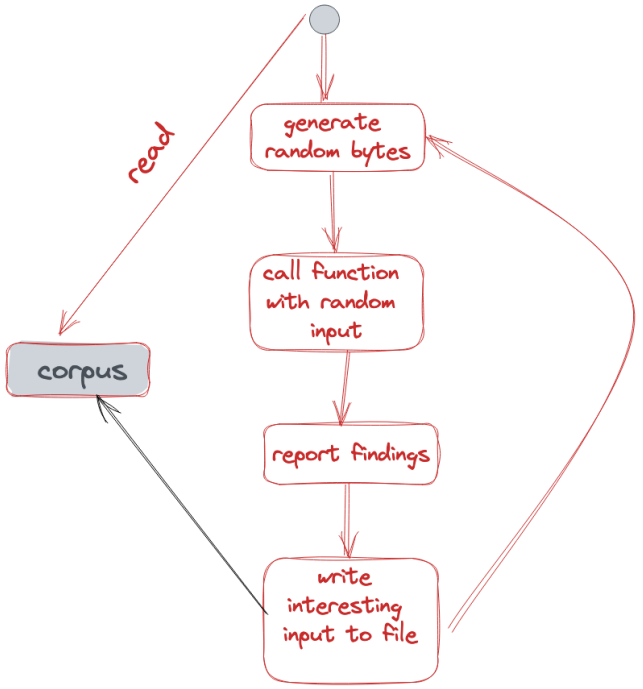


FUZZING - START WITH CORPUS

First Run: Generate Corpus



Next Runs: Extend Corpus



FUZZING IN RUST-VMM/VM-VIRTIO

- Implemented fuzz targets for:
  - Virtio Queue
  - Virtio Queue Serialization
  - Virtio Vsock (Packet)
- Discovered 3 crashes – only 1 triggable by malicious drivers
- Fuzzing runs for 15 minutes on Pull Requests
- Fuzzing at the library level using libfuzzer
- Cloud Hypervisor discovered 1 timeout

P1: RUN TIMEOUT IS TOO LARGE

- Fuzzers have a default timeout for hangs/timeouts (i.e. libfuzzer 20 minutes)
- Instead: adjust timeout to suit your use case

FUZZING AT THE LIBRARY LEVEL

Advantages

- It's EASY to set up
- Can run on almost any host
- Runs in userspace

Disadvantages

- Doesn't cover the whole virtio setup
- Needs mocking for the driver side of communication – complicated
- Can find false positives

EASY IS A GOOD THING!

FUZZING AT THE LIBRARY LEVEL

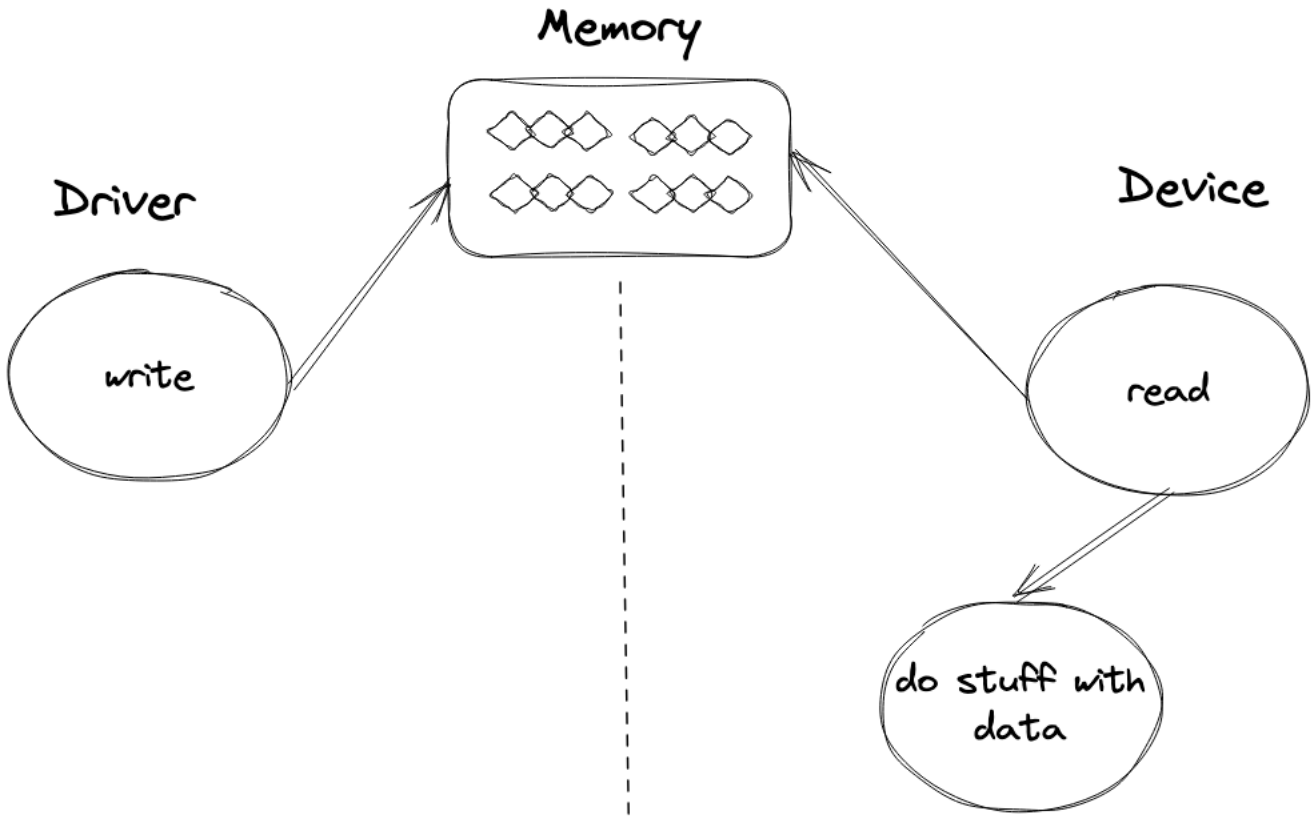
Advantages

- It's EASY to set up
- Can run on almost any host
- Runs in userspace

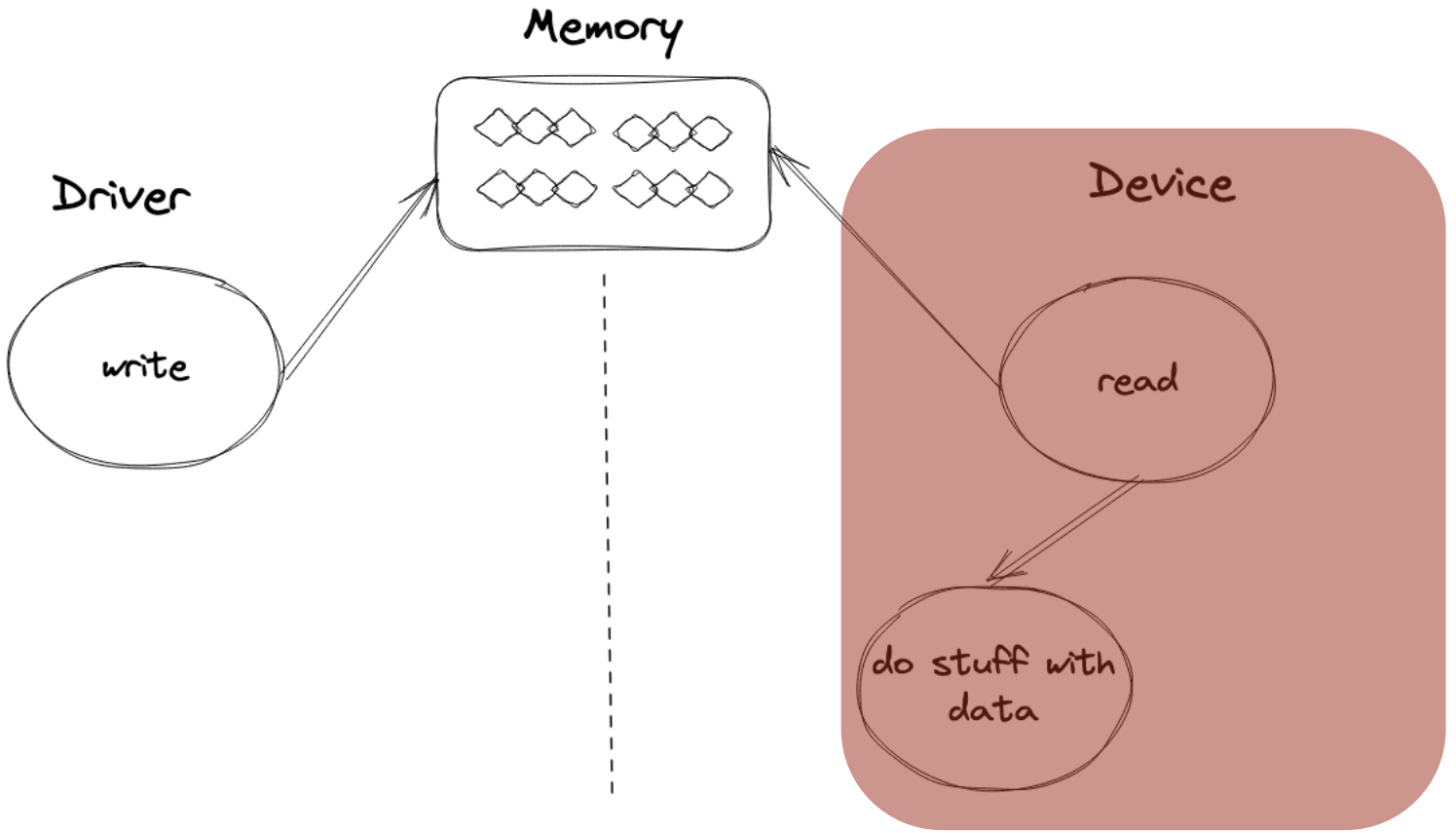
Disadvantages

- Doesn't cover the whole virtio setup
- Needs mocking for the driver side of communication – complicated
- Can find false positives

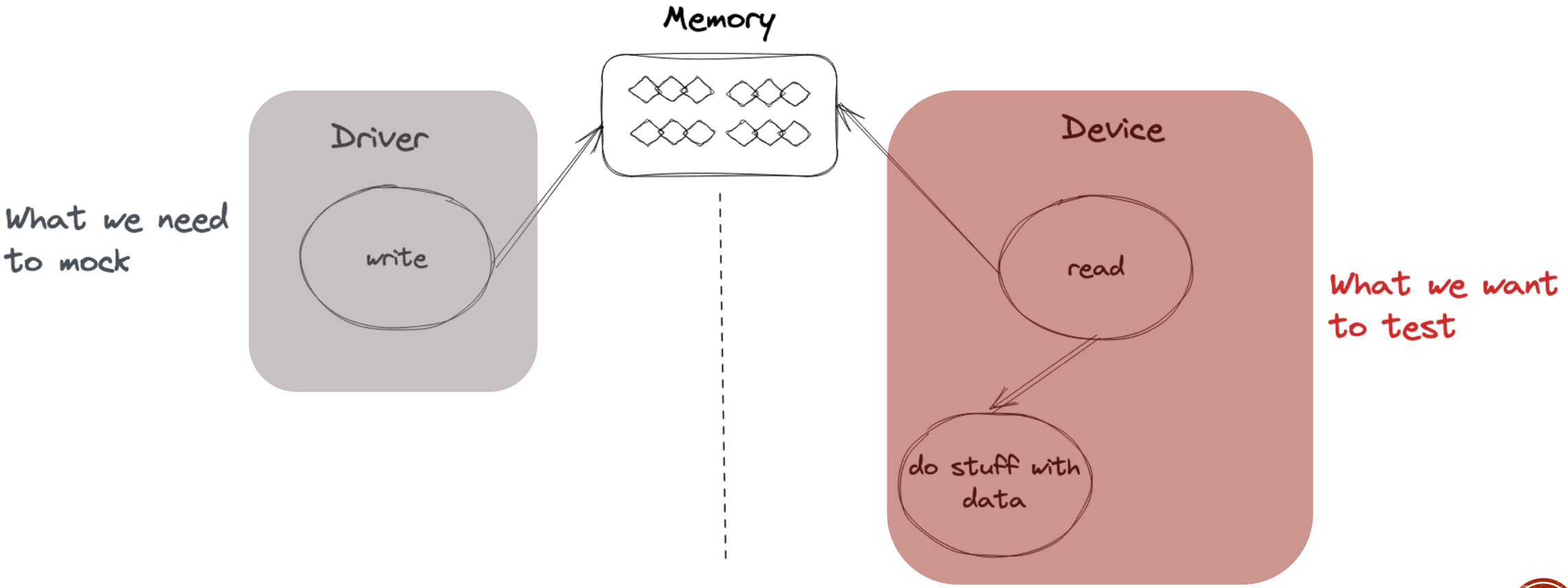
MOCKING THE DRIVER



MOCKING THE DRIVER



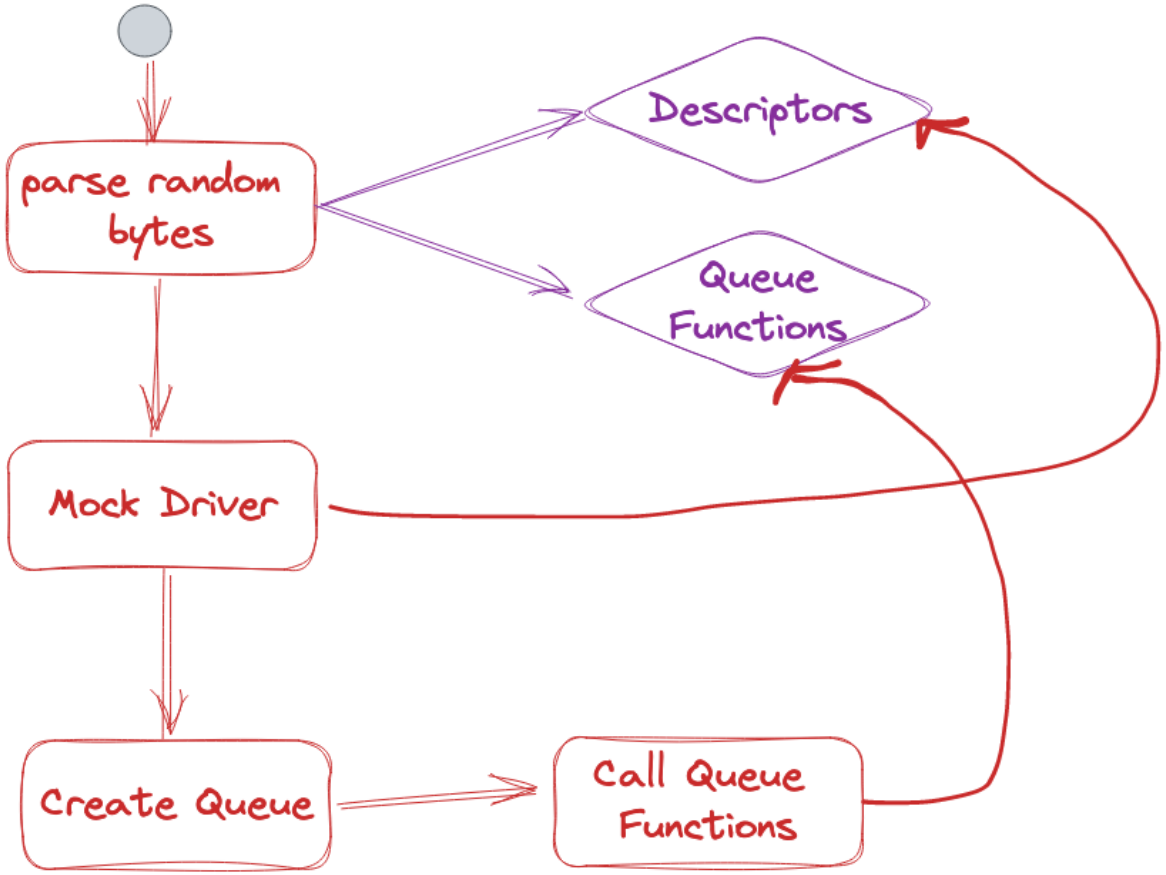
MOCKING THE DRIVER



MOCKING THE DRIVER

- Needed for unit tests as well
- Initial version of a mock interface from the beginning
- Evolve the mock interface as you implement features and devices

FUZZING HIGH LEVEL FLOW



P2: RETROFIT FUZZING WHEN THE PROJECT IS MATURE

- Instead: keep fuzzing in the back of your head, think about how mock objects can be reused

P3: CRASH ON INVALID INPUT

- Instead: return error to be processed at higher level

STRUCTURE AWARE FUZZING

STRUCTURE AWARE FUZZING

Without

```
fuzz_target!(|data: &[u8]| {
  // Interpret data as bytes
  // sense for the library
  // fuzz.
}
```

With

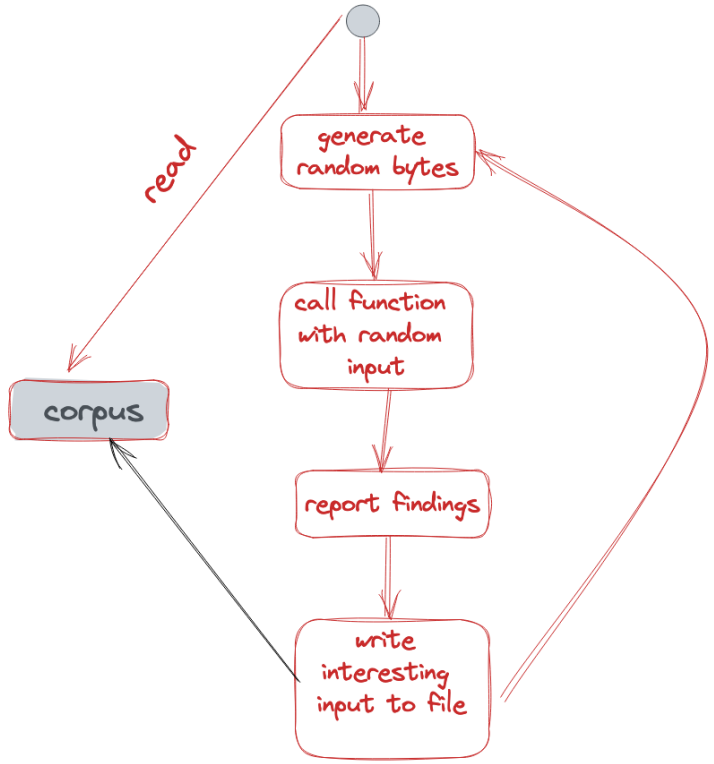
```
fuzz_target!(|color: Rgb| {
  // Data already parsed as
  // what you want to fuzz.
}
```

STRUCTURE AWARE FUZZING

- Implemented with Arbitrary
- Significantly reduces the code you need for parsing
- LOC: 270 vs 738

STRUCTURE AWARE FUZZING - PROBLEMS

- Not reproducible
- Reads introduce randomness
  - read_corpus(write_corpus()) – not idempotent
- Cannot use it with a custom corpus



P4: RELY ON INCREMENTAL IMPROVEMENTS

- Instead: check that the tools you want to use have support for future extensions or allow sufficient time for changes

THE CURIOS CASE OF AN OVERFLOW NOT FOUND BY FUZZING...

THE BUG: EXPLAINED

```
463       -         let data_desc = desc_chain.next().ok_or(Error::DescriptorChainTooShort)?;
      463 +         // Starting from Linux 6.2 the virtio-vsock driver can use a single descriptor for
          both
      464 +         // header and data.
      465 +         let data_slice = if chain_head.len() >= PKT_HEADER_SIZE as u32 + pkt.len() {
```

- Addition can overflow
- Bug found during code review

ANALYSIS AND IMPROVEMENTS

- With fuzzing: ~40 minutes to find the bug
- Added an optional fuzz session that runs for 24 hours:
  - Needs to be started by one of the maintainers
  - Should be started only when needed (changes impacting device implementation)

P5: NOT RUNNING FUZZING LONG ENOUGH

- Instead: figure out a way to include fuzzing for extended period of times without disrupting development

CODE COVERAGE FOR FUZZING

COVERAGE IN RUST

- Llvm-cov
- In rust: only line coverage

COVERAGE REPORT

- Coverage for queue.rs file

Fuzzing Config	Missed Regions	Coverage
15 minutes	34	81.82%

COVERAGE REPORT

- Coverage for queue.rs file

Fuzzing Config	Missed Regions	Coverage
15 minutes	34	81.82%

😀

COVERAGE REPORT

- Coverage for queue.rs file

Fuzzing Config	Missed Regions	Coverage
15 minutes	34	81.82%
15 minutes + minimal corpus		

😀

COVERAGE REPORT

- Coverage for queue.rs file

Fuzzing Config	Missed Regions	Coverage
15 minutes	34	81.82%
15 minutes + minimal corpus	34	81.82%

COVERAGE REPORT

- Coverage for queue.rs file

Fuzzing Config	Missed Regions	Coverage
15 minutes	34	81.82%
15 minutes + minimal corpus	34	81.82%
2 weeks		

COVERAGE REPORT

- Coverage for queue.rs file

Fuzzing Config	Missed Regions	Coverage
15 minutes	34	81.82%
15 minutes + minimal corpus	34	81.82%
2 weeks	30	83.96%

COVERAGE REPORT

- Coverage for queue.rs file

Fuzzing Config	Missed Regions	Coverage	
15 minutes	34	81.82%	😀
15 minutes + minimal corpus	34	81.82%	😀
2 weeks	30	83.96%	☹

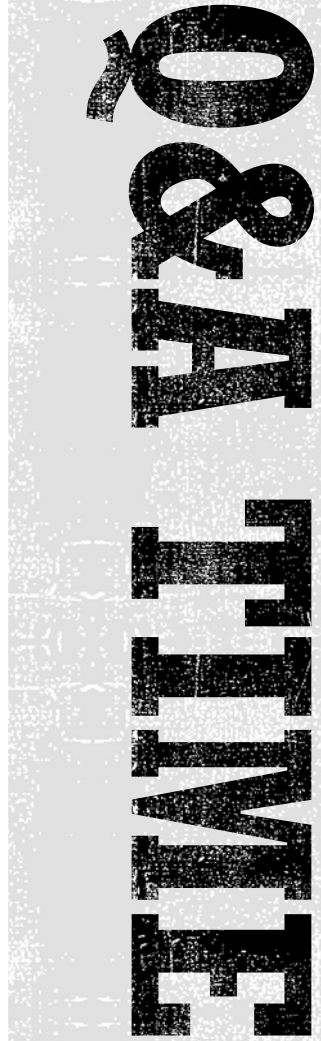
P6: USING COVERAGE TO DECIDE WHEN TO STOP FUZZING

- Instead: use coverage to understand how to extend fuzz targets

Summary

- P1: Run Timeout is too large
- P2: Retrofit Fuzzing when the project is mature
- P3: Crash on invalid input
- P4: Rely on incremental improvements
- P5: Not running fuzzing long enough
- P6: Using coverage to decide when to stop fuzzing

Fuzzing does not need to be hard to be useful.



WHY COVERAGE WAS 81.82%?

- Functions not called on purpose:
  - Iterators over descriptor chains -> the data needs to be interpreted by devices anyway
- Missed to call 1 function: desc_table

Fuzzing Config	Missed Regions	Coverage
15 minutes	34	81.82%
15 minutes + minimal corpus	34	81.82%
2 weeks	30	83.96%
15 mins + missing funcs	26	86.1%

WHY COVERAGE WAS 81.82%? (2)

- Most of the missed coverage regions are macros that are printing errors
- logger is not initialized when running fuzzing

```
------------------
| <virtio_queue::queue::Queue as virtio_queue::QueueT>::is_valid::<vm_memory::mmap::GuestMemoryMmap>::{closure#0}:
|  317|  55.6k|            .map_or(true, |v| !mem.address_in_range(v))
------------------
| Unexecuted instantiation: <virtio_queue::queue::Queue as virtio_queue::QueueT>::is_valid::<_>::{closure#0}
------------------
318|       |        {
319|  13.5k|            error!(
320|     0|                "virtio queue descriptor table goes out of bounds: start:0x{:08x} size:0x{:08x}",
321|     0|                desc_table.raw_value(),
322|       |                desc_table_size
323|       |            );
324|  13.5k|            false
325|  45.2k|        } else if avail_ring
326|  45.2k|            .checked_add(avail_ring_size)
327|  45.2k|            .map_or(true, |v| !mem.address_in_range(v))
                                        ^39.3k
------------------
| <virtio_queue::queue::Queue as virtio_queue::QueueT>::is_valid::<vm_memory::mmap::GuestMemoryMmap>::{closure#1}:
|  327|  39.3k|            .map_or(true, |v| !mem.address_in_range(v))
------------------
| Unexecuted instantiation: <virtio_queue::queue::Queue as virtio_queue::QueueT>::is_valid::<_>::{closure#1}
------------------
328|       |        {
329|  16.7k|            error!(
330|     0|                "virtio queue available ring goes out of bounds: start:0x{:08x} size:0x{:08x}",
331|     0|                avail_ring.raw_value(),
332|       |                avail_ring_size
333|       |            );
334|  16.7k|            false
335|  28.5k|        } else if used_ring
336|  28.5k|            .checked_add(used_ring_size)
337|  28.5k|            .map_or(true, |v| !mem.address_in_range(v))
                                        ^26.9k
------------------
| <virtio_queue::queue::Queue as virtio_queue::QueueT>::is_valid::<vm_memory::mmap::GuestMemoryMmap>::{closure#2}:
|  337|  26.9k|            .map_or(true, |v| !mem.address_in_range(v))
------------------
| Unexecuted instantiation: <virtio_queue::queue::Queue as virtio_queue::QueueT>::is_valid::<_>::{closure#2}
------------------
338|       |        {
339|  6.70k|            error!(
340|     0|                "virtio queue used ring goes out of bounds: start:0x{:08x} size:0x{:08x}",
341|     0|                used_ring.raw_value(),
342|       |                used_ring_size
343|       |            );
344|  6.70k|            false
345|       |        } else {
346|  21.8k|            true
347|       |        }
348|  91.1k|    }
------------------
```

FUZZING FINDINGS

- Index out of bounds access in the Virtio Mock implementation
  - https://github.com/rust-vmm/vm-virtio/pull/162/commits/e42fe6b3165aceec7183e206874d5970a6e591f7
- Panic when using wrong ordering in functions called by VMM:
  - https://github.com/rust-vmm/vm-virtio/pull/174
  - Also, excluded the invalid ordering in fuzzer
- Division by 0 in descriptor chains iterator:
  - https://github.com/rust-vmm/vm-virtio/pull/173