OpenBSD's `vmd(8)` Hypervisor & Multi-processing — *2 Years Later*

```
fork&
exec&
fork&
exec.
```

Dave Voutila <dv@openbsd.org>, EuroBSDCon 2024

Dave Voutila (dv@)

Vermont 🍁, USA 🇺🇸
(40 mins from Québec 🇨🇦)

Maple (8) & Moxie (3) are featured throughout (and one of their dog friend, Fritz).

What am I going to talk about?

or: why should you stick around and not go grab coffee ☕

- In Tokyo and Ottawa, presented new multi-processing VM model for `vmd(8)`
- Today, we'll look at the lessons learned: *good, the bad, and the ugly!*
  - `vmd(8)` is a good example of "privsep", IPC, and OpenBSD's `imsg`
  - For some definition of *good* 😉
- And, if we're lucky, a glimpse into the future of `vmd(8)`

Multi-process the *what* now? 🤨

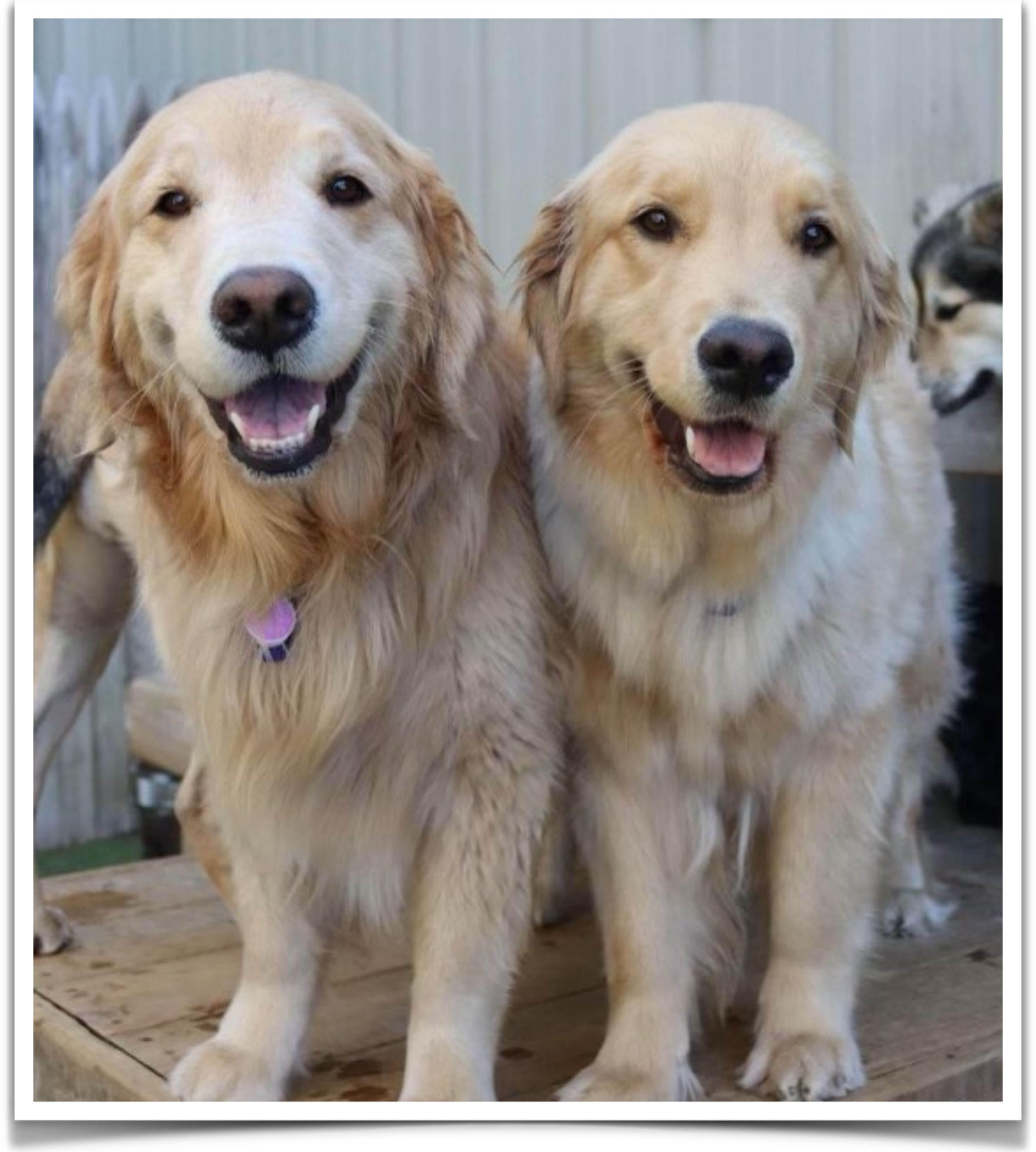
Hypervisors are High Value Targets

Why do you rob a bank? It’s where the money is. 💰

- If it’s networked, *it’s vulnerable.*
  - In practice, a lot of VMs are networked.
- “It’s ok, I’m running it in a vm.”
- The majority of hypervisor escapes are through emulated devices:
  - CVE-2015-3456 — QEMU floppy disk controller
  - CVE-2015-7504 — QEMU network device
  - CVE-2020-3967 — VMWare EHCI controller
  - OpenBSD 6.8/6.9 — DHCP packet handler stack overflow

Multi-process QEMU

First Type-2 open source hypervisor doing this?



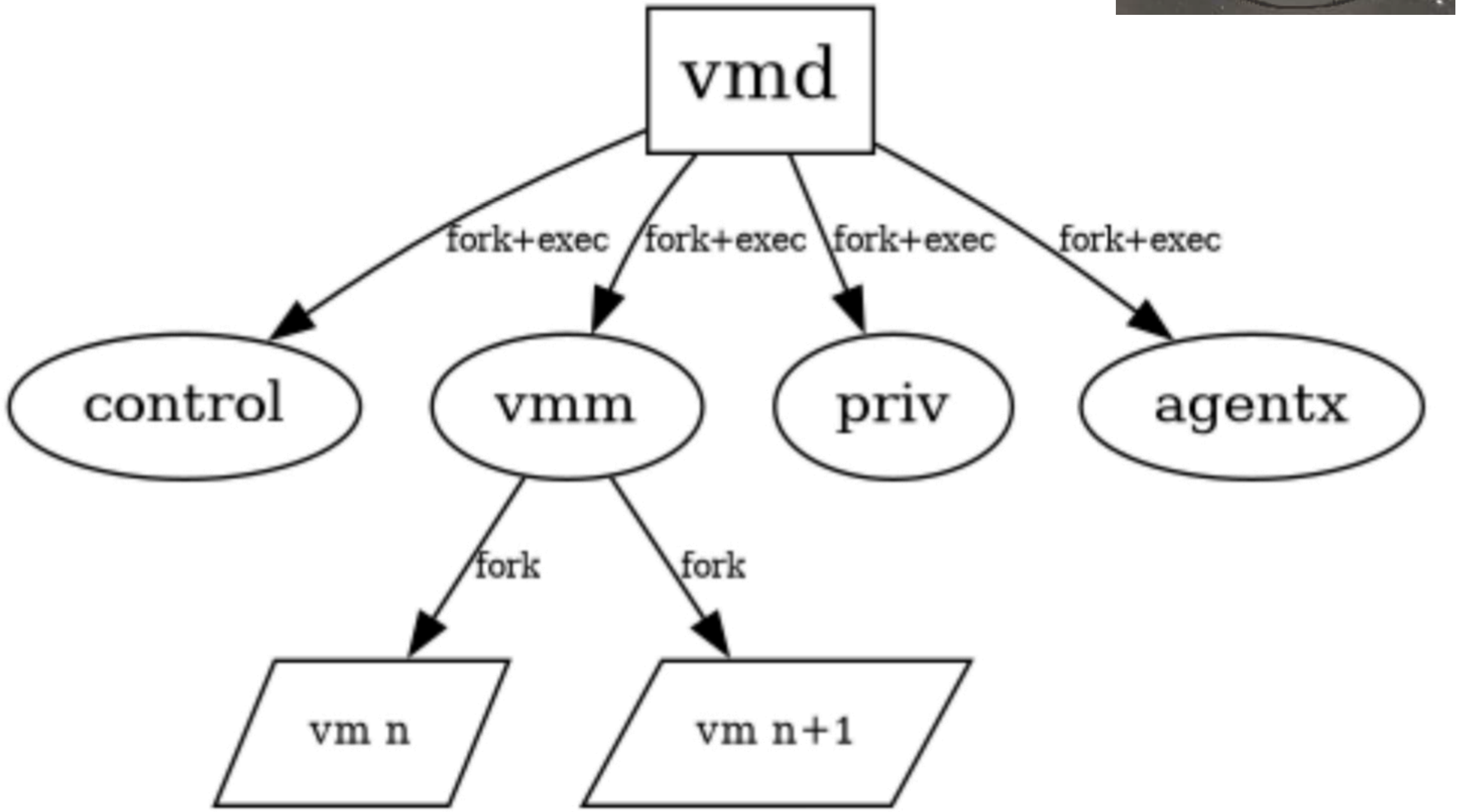
- **Oracle started work in 2017**, landed in QEMU December 2020
  - Elena Ufimtseva, Jag Raman, John G. Johnson
  - https://lists.gnu.org/archive/html/qemu-devel/2020-12/msg00268.html
- **…but, who uses it?**
  - I'd presume Oracle Cloud!
- Documentation is primarily about design, points to a wiki…*last updated in 2020?!*
  - Additional burden placed upon the poor administrators 😩

But let’s talk about OpenBSD 🐡

vmm(4)/vmd(8)

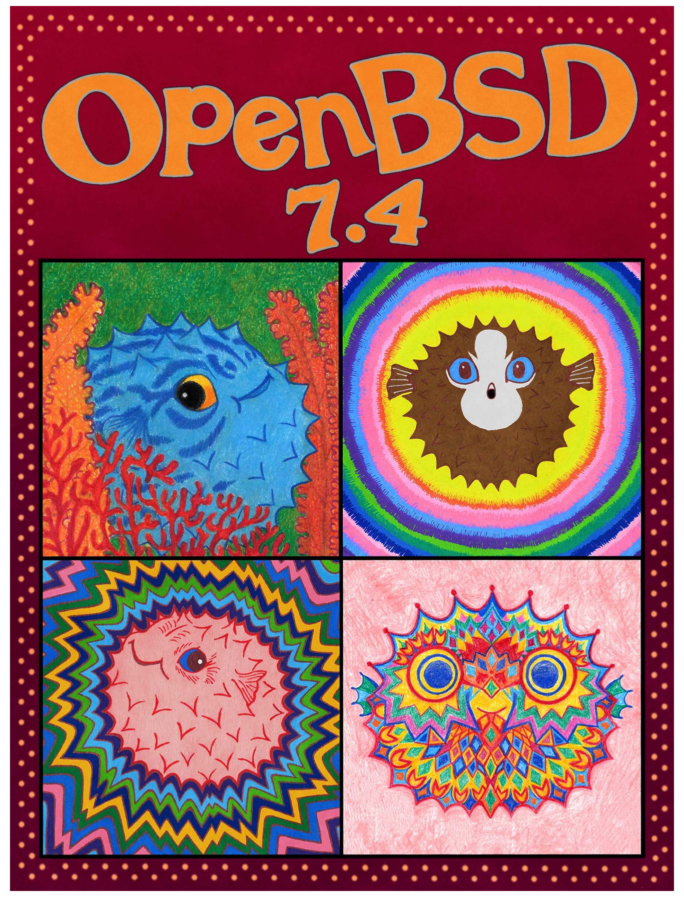
OpenBSD’s native hypervisor — “then” (7.3 and earlier)

- Originally released with OpenBSD 5.9 (March, 2016) by mlarkin@ & reyk@
- Currently **amd64** only with support for both amd64 and i386 guests (arm64 support “has started”)
- Adopted privilege separation design
  - fork+exec —> chroot(2) & pledge(2)
  - drop from root to _vmd
- Components
  - vmm(4) — in-kernel VM monitor
  - vmd(8) — userland VM daemon
  - vmctl(8) — userland VM control utility

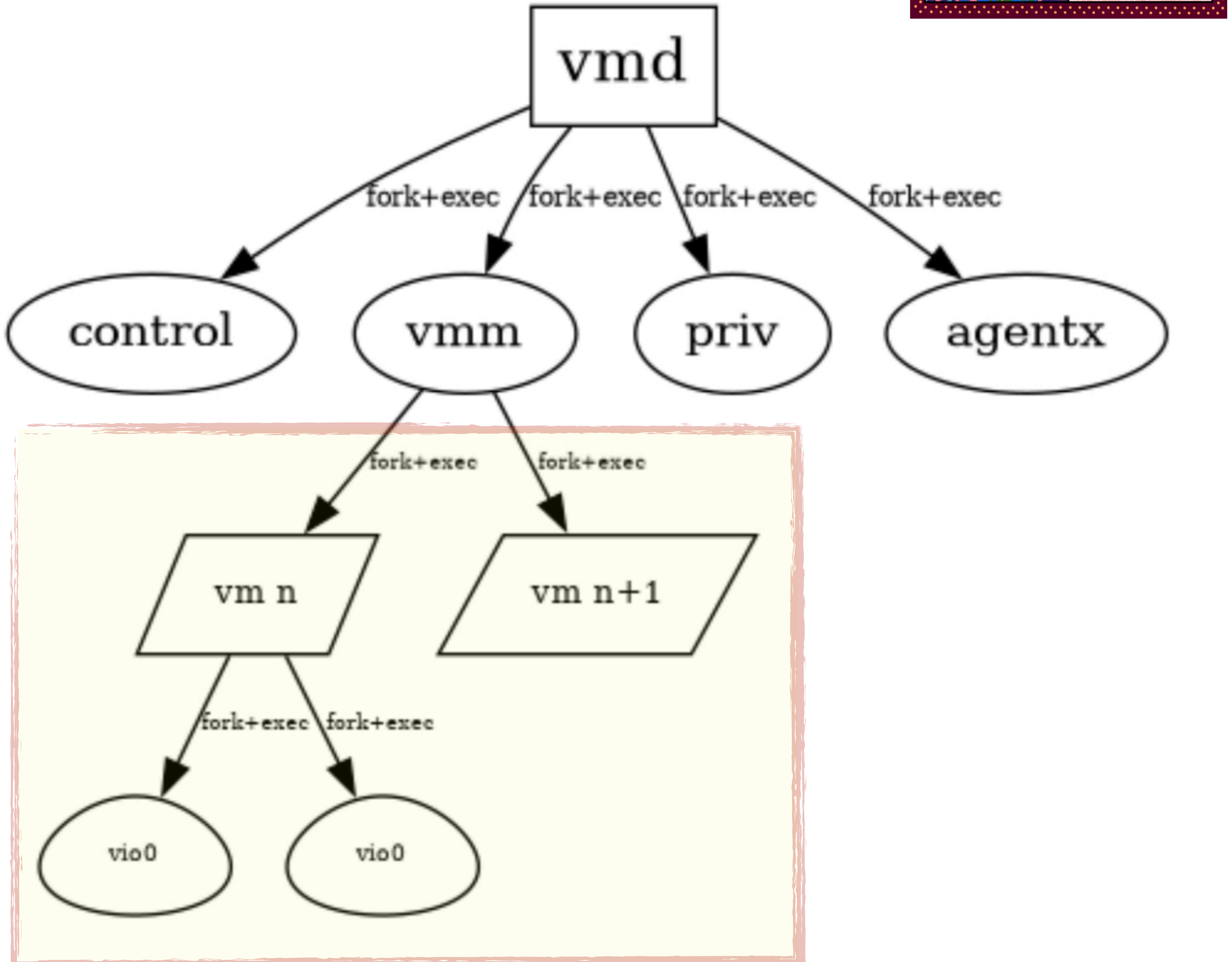


vmm(4)/vmd(8)

OpenBSD’s native hypervisor — “now” (7.4 - current)



- Proper re-exec by vmm process to give each VM their own address space layout, `pledge(2)`s, and files
  - Borrowed approach from OpenSSH to deal with the fact `vmm` process uses `chroot(2)` & `unveil(2)`
- Emulated VirtIO devices are `fork+exec`’d from the VM process



The Good 🍀 🌈 🥰

Security! But at what cost?

What about the user/admin experience? Does it change?

- OpenBSD 7.3 and earlier

```
# rcctl -f start vmd

# vmctl start -Lc -d disk.qcow2 -m 8g guest
```

- OpenBSD-current

```
# rcctl -f start vmd

# vmctl start -Lc -d disk.qcow2 -m 8g guest
```

Security! But at what cost?
What about the user/admin experience? Does it change?
• OpenBSD 7.3 and earlier
# rcctl -f start vmd
# vmctl start -Lc -d disk.qcow2 -m 8g guest
• OpenBSD-current
# rcctl -f start vmd
# vmctl start -Lc -d disk.qcow2 -m 8g guest

Security! But at what cost?

What about the user/admin experience? Does it change?

- OpenBSD 7.3 and earlier

```
# rcctl -f start vmd

# vmctl start -Lc -d disk.qcow2 -m 8g guest
```

- OpenBSD-current

```
# rcctl -f start vmd

# vmctl start -Lc -d disk.qcow2 -m 8g guest
```

Vectorized IO and Zero-copy

Multi-process VirtIO makes things easier to hack on (1/2)

- For **raw** disks, the vioblk device can now use `p{read,write}v(2)`
  - Simpler code reading/writing from the guest buffers
  - This was a **net-negative** diff! (~80 lines shorter)
- Lower average host CPU utilization under io load
  - Guests with more advanced VirtIO usage benefit the most **cough*linux*cough**
- Adapted to VirtIO **network** device emulation as well

Full(ish)-Duplex VirtIO Networking!

Multi-process VirtIO makes things easier to hack on (2/2)

- Original vionet device had a major flaw: one side could starve the other
- **3 event-loops/threads**: main/control, transmit (tx), receive (rx)
- Uses `pipe(2)`’s as channels between threads
- Simplifies packet injection for `vmd(8)`’s internal DHCP service
  - “local” interfaces in `vmd(8)` intercept DHCP requests on tx-side, pass to rx-side via passing a pointer via a `pipe(2)`
- Reduced average latency, better CPU utilization

The Bad 🌧️ 💩

An IPC Headache

Pain is really just a deviation from you current baseline.

- **Synchronous** Channel
  - Bootstrapping device config post-`execvp(2)`
  - VirtIO PCI register reads need to block vcpu
- **Asynchronous** Channel
  - Lifecycle events (vm pause/resume, shutdown)
  - Assert/Deassert IRQ
  - Set host MAC address (vionet)

High-level Message Flow

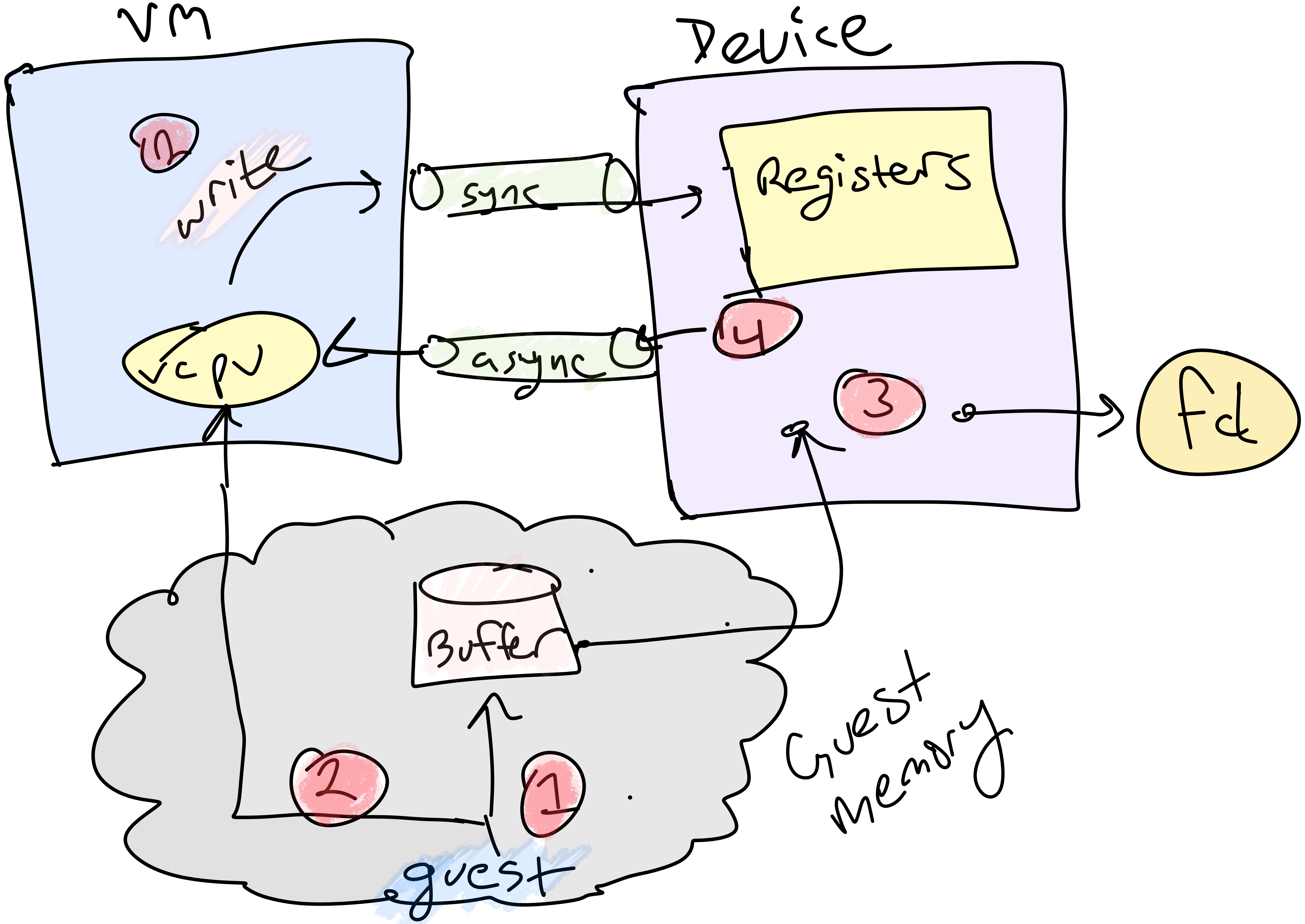
Sorry for my artwork 🧑‍🎨

1. Guest fills buffers, updates virtqueues, etc.

2. Guest writes to Device register via IO instructions *(note: not using mmio yet)* causing VM exit

3. Device is notified it can process data. Performs `write(2)`

4. Device kicks guest via vcpu interrupt to notify buffers are processed



The Ugly 🤡 🫣

Multi-process means shared memory

Sort of simple on the surface…handled via an `ioctl(2)`

- …a new `vmm(4) ioctl(2)` appears! (`VMM_IOC_SHAREMEM`)
- If and only if:
  - You have an open fd to `/dev/vmm`
  - You know all the `vm_mem_ranges` for a given vm id
  - You have the `vmm` and `proc` promises (in a pledged program)
- …it will create a shared UVM anonymous mapping into your process's virtual address space

Multi-process means shared memory

Shared memory leads to chasing UVM ghosts 👻

- Multiple processes sharing UVM mappings really puts pressure on OpenBSD’s UVM & pmap layers
    - Been chasing a corruption for ~2 years now!
    - As we unlock more of the kernel, more fireworks happen
- Intel EPT pmaps are still a WIP 🚧
    - I’ve floated some diffs, but won’t make 7.6 release
    - Intel always makes things interesting

Looking forward 🔭

Future Work & Research

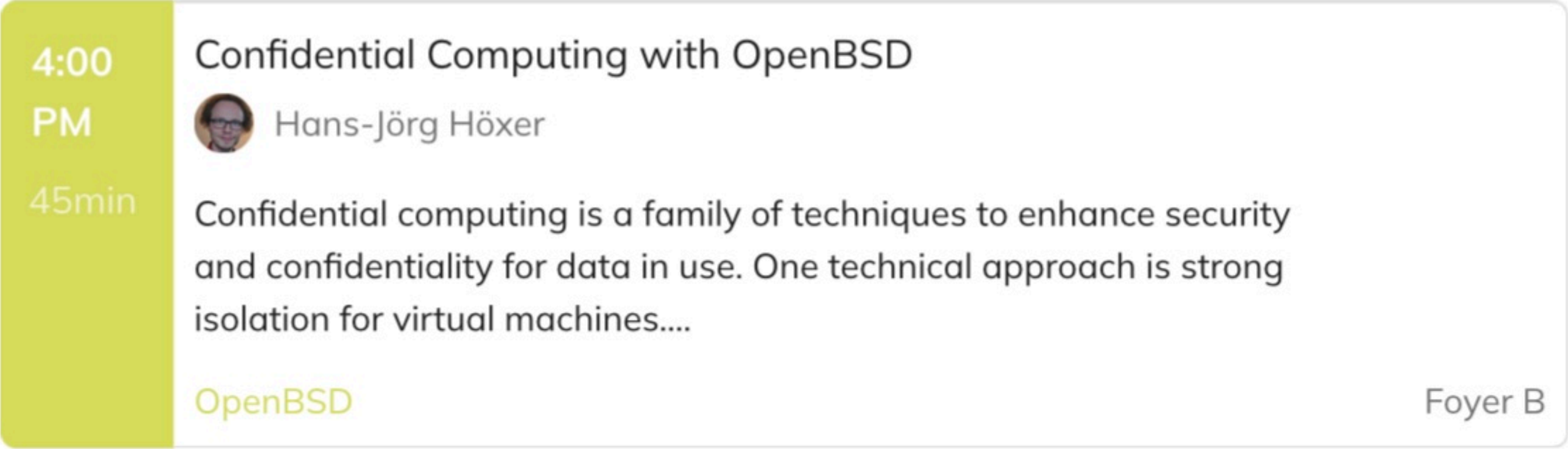
Plans for the next hackathon? 📋

- SMP-ification at some point
    - honestly…not the most interesting thing to me!
- `arm64` — have the hardware, don't have the time 🌅
- `ipcgen(1)` — my current thought experiment on simplifying vmd's most confusing part… the ipc plumbing
    - IDL for defining IPC message flows and fd-passing
        - file descriptor passing is major pain when needing to pass a variable number of them…like qcow2 images!
    - Thought is to push imsg and event loop code behind code generator
    - Could make it easier to contribute and improve quality

Confidential Computing with `vmd(8)`

Bringing AMD's SEV to OpenBSD's guest vms.

- Check out Hans-Jörg Höxer's talk tomorrow (Sunday)!



4:00 PM

45min

Confidential Computing with OpenBSD

Hans-Jörg Höxer

Confidential computing is a family of techniques to enhance security and confidentiality for data in use. One technical approach is strong isolation for virtual machines....

OpenBSD

Foyer B

Thanks!

See you next year, maybe?