Ryo Shimizu

<ryo@netbsd.org>

AsiaBSDCon 2023 WIP

What is NVMM?

NVMM - NetBSD Virtual Machine Monitor

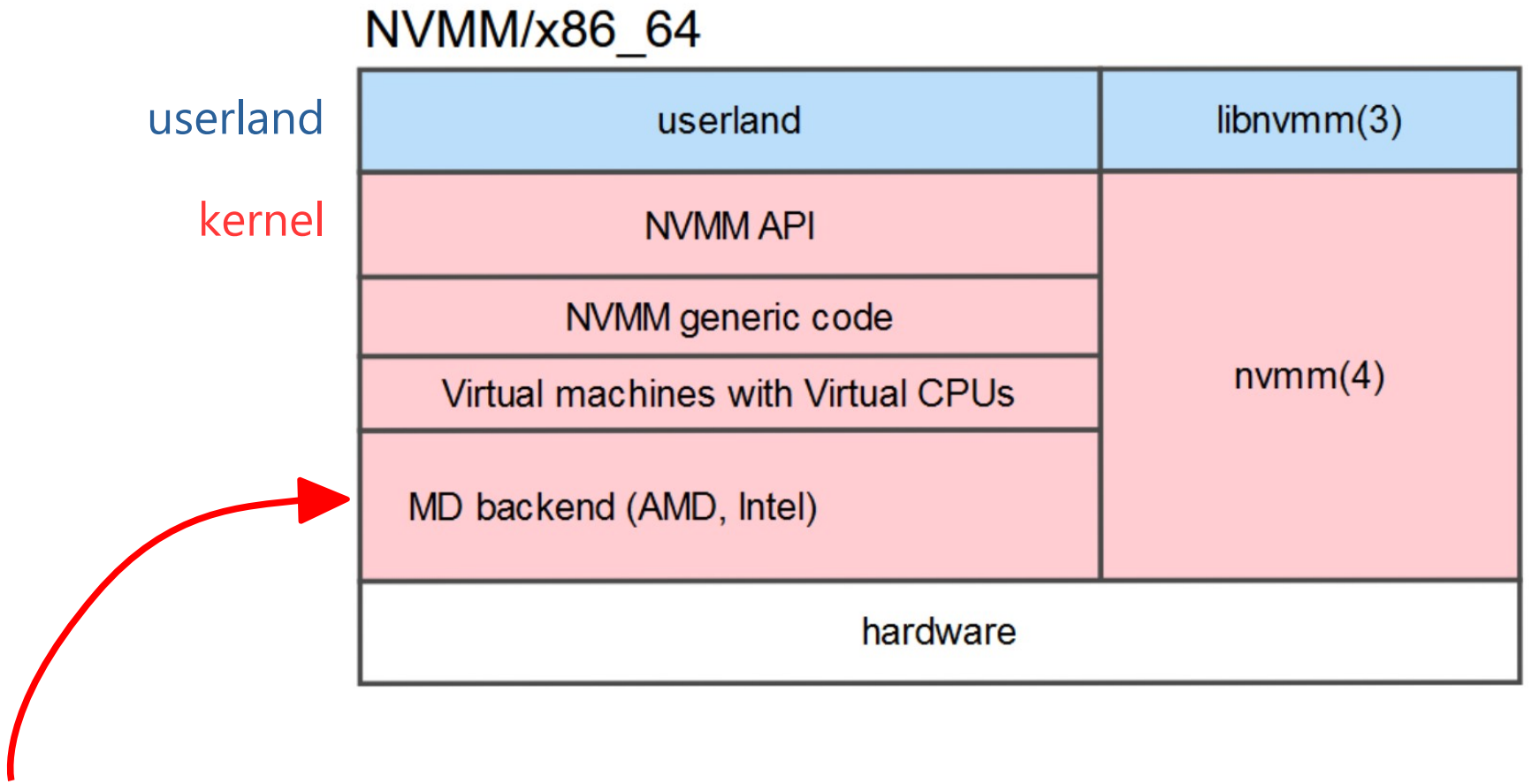
Currently only works on x86_64.
I'm porting nvmm to netbsd/aarch64 ARMv8.0

Q: Why ARMv8.0? Why not use ARMv8.1 VHE?
(VHE:Virtualization host extension)

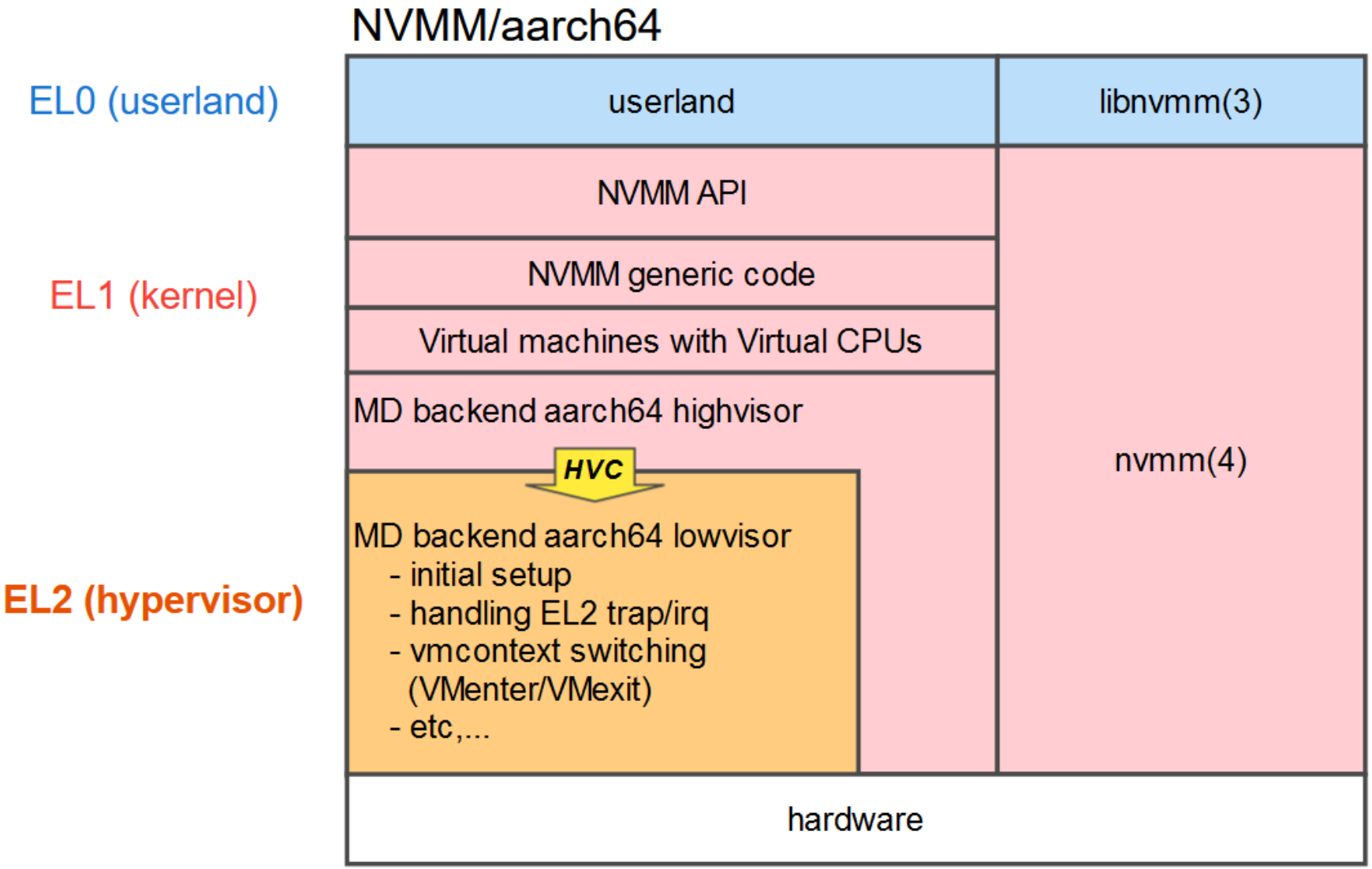
A: Because everyone's favorite raspberry Pi is ARMv8.0 :-)

The number of ARMv8.x devices has been increasing there recently, but there are still many ARMv8.0 devices.

What should I make?



I need to create an MD backend for aarch64

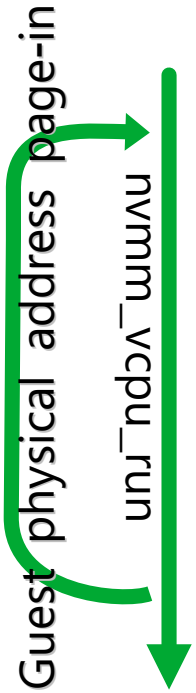


- **Separate EL1 highvisor and EL2 lowvisor in the same way that linux's kvm/arm64 does.**
- **Switches to EL2 by HVC instruction with context switching parameters.**
- **It is important to note that EL2 cannot directly access variables in EL1 kernel virtual address. because EL2 is running in Physical Address. the VA of a parameter would be convert to PA on EL1, and passes it to EL2.**
- **If ARMv8.1 VHE is used, It should be more simple and fast implementation because it can run the netbsd kernel with EL2.**

And one more thing to do - pmap(9)

- The nvmm process has two virtual memory maps (struct vm_map). One is used for normal user processes, while the other is used for the Guest Physical Addresses (GPA) of the virtual machine that nvmm creates.
- When a guest virtual machine attempts to access unmapped guest physical memory, nvmm uses the GPA virtual memory map to perform a page-in operation using the uvm(9) and pmap(9) functions.
- Since aarch64 uses the stage2 MMU to translate the GPA to the Host Physical Address, it's necessary for pmap(9) to support stage2 translation tables. This is because the page table entry descriptor formats for the stage1 and stage2 tables in the aarch64 MMU are slightly different.

Basic Operation




```
netbsd userland(EL0): nvmm_vcpu_run(3) -> nvmm ioctl(2)
netbsd kernel(EL1): nvmm aarch64 highvisor: HVC -> EL2
netbsd kernel(EL2): nvmm aarch64 lowvisor: VM context switching (aka. VMEnter)
guest VM (EL0,EL1): running VM…
                  :
guest VM (EL0,EL1): IRQ or TRAP -> EL2
netbsd kernel(EL2): nvmm aarch64 lowvisor: VM context switching (aka. VMExit)
netbsd kernel(EL2): nvmm_aarch64 lowvisor: ERET -> EL1
netbsd kernel(EL1): nvmm aarch64 highvisor: nvmm ioctl(2) returns
netbsd userland(EL0): nvmm_vcpu_run(3) returned
```

How is the progress?

<DEMO>

- It would be about 20%?
- Hopefully, it will be completed by the end of this year. (?)
- I'm working on the interrupt part now.
- It is still unstable on MULTIPROCESSOR due to bodge TLB/CPU cache handling :-(
- My goal (for now) is to make it work stably on a uniprocessor environment.

Conclusion

- Im working on porting nvmm/aarch64.
- working on ARMv8.0.
- working to the extent that the netbsd kernel boots.

Any question?