STORAGE DEVELOPER CONFERENCE

SDC 23

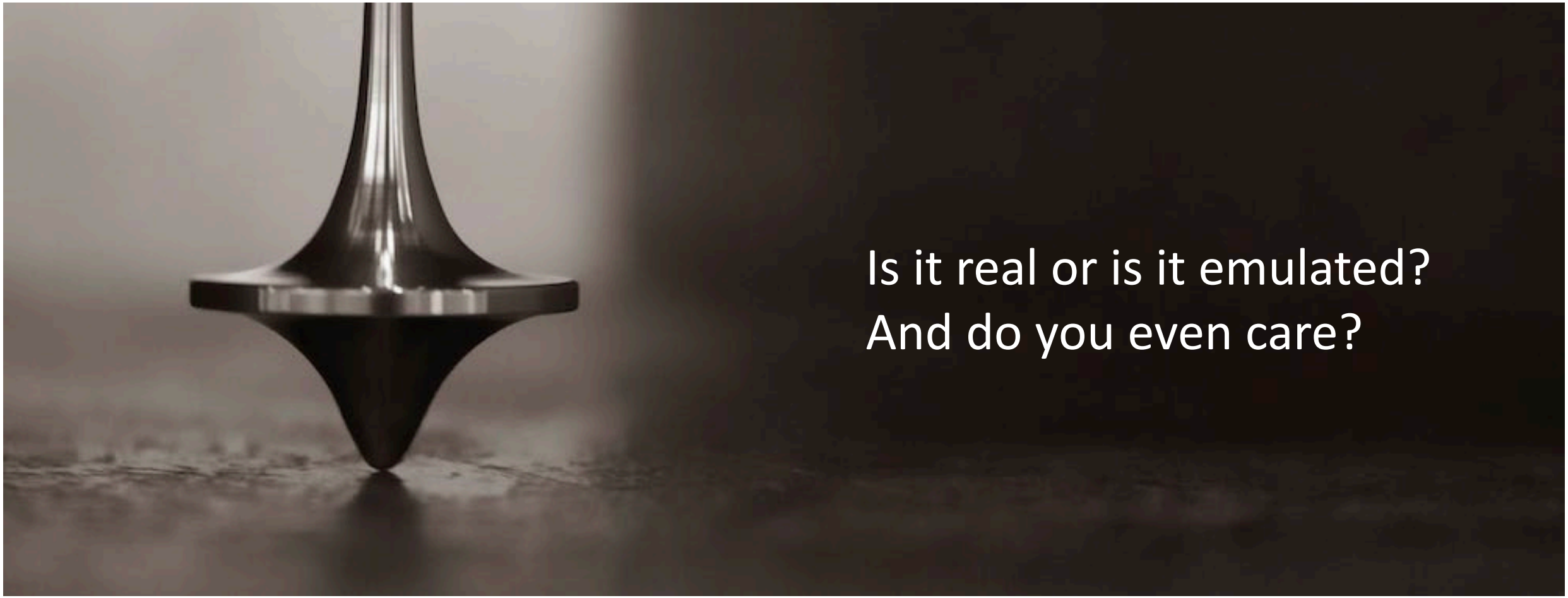
BY Developers FOR Developers

An Emulation Framework for Computational Storage

Stephen Bates and Abhishek Gupta, TESL, Huawei

Emulation=Inception=QEMUception

Emulation=Inception=QEMUception



The Case for Emulation

Hardware is Hard!

The Case for Emulation - I

- Hardware is hard!
  - Chips take a long time to develop.
  - Chips today need firmware, this is buggy.
  - Chips often (always) are broken first time around.
- Look at CXL for example!
  - Spec is at 3.0.
  - Hardware < 1.0 ;-).
- How do software developers develop without hardware?

While other options exist, QEMU is becoming the emulation environment of choice. There are several ways QEMU can provide emulation of hardware. We will review these in this talk!

The Case for Emulation - II

- Hardware is expensive!
- Look at CXL for example!
  - CXL-enabled servers cost a bunch of money, have buggy UEFI code, lack OS support etc.
  - If the system breaks how do you know what is to blame?
    - Bad hardware?
    - Bad firmware?
    - Bad software?
- A software developer in a coffee shop in Lima does not have room in their backpack for a Sapphire Rapids.

While other options exist, QEMU is becoming the emulation environment of choice. There are several ways QEMU can provide emulation of hardware. We will review these in this talk!

SDC 23

The Case for Emulation - III

- Hardware is hard to debug!
- Look at CXL for example ;-).
  - Reboot times are measured in (ten) minutes.
  - When things go wrong early in the boot process there is often little (zero) visibility or debug capability.
  - And then you need to tweak something and reboot again (and again and again).
- Emulation enables full visibility (gdb hardware anyone!?).

While other options exist, QEMU is becoming the emulation environment of choice. There are several ways QEMU can provide emulation of hardware. We will review these in this talk!

The Case for Emulation - IV

- Sometimes you do need actual hardware!
- Sometimes performance is important ;-) (but perhaps not *that* often if you are a developer).
- Sometimes you need to actually sell something ;-).
- But much of the time emulation is just fine.

While other options exist, QEMU is becoming the emulation environment of choice. There are several ways QEMU can provide emulation of hardware. We will review these in this talk!

The Case for Emulation - VI

Is this the real life? Is this just fantasy?
Caught in a landslide, no escape from reality

Perhaps all Freddie was really looking for was an emulation?

Emulation in QEMU

Is it real and do you care?

Emulation in QEMU - I

- QEMU originated in 2003.
- Two types of emulation:
  - Can emulate a CPU with a different Instruction Set Architecture (ISA) to the host (e.g. emulate an arm64 CPU on an Intel server). This **is not** the emulation this talk cares about.
  - Can emulate hardware that is not actually present on the host server. This **is** the emulation this talk cares about.

QEMU supports a range of different emulation modes. This includes the emulation of hardware while using accelerated emulation modes for the CPU (e.g. Xen and KVM).

Emulation in QEMU - II

- This talk focuses on the System Emulation mode of QEMU.
  - CPU is either emulated or KVM/Xen accelerated.
  - Rest of the system seen by the ”Virtual Machine” (VM) can be a mix of real hardware and emulated hardware.
  - Real hardware can be assigned completely to the VM (e.g. via vfio) or para-virtualized.
  - Fake hardware can be emulated by QEMU or another process running on the host.

While not covered in this talk the topic of assigning real hardware to one or more VMs running on a system is fascinating. See topics like SR-IOV, SIOV, vfio and mediated devices for more information.

Emulation in QEMU - III

- System Emulation
  - A mix of real hardware and fake (emulated) hardware is presented to the VM.
  - The VM has no way of knowing which hardware is “real” and which is “fake”. Perhaps a totem is needed?
  - Since “fake” hardware is actually software we can fully control its behavior by editing the source code!

Some examples of hardware that can be emulated inside QEMU include:

- Storage devices (like NVMe).
- Persistent Memory (NVDIMMs).
- Networking Interface Cards (NICs).
- Peripherals.

Emulation Example in QEMU

KV-Capable NVMe SSDs in a Server Anyone?

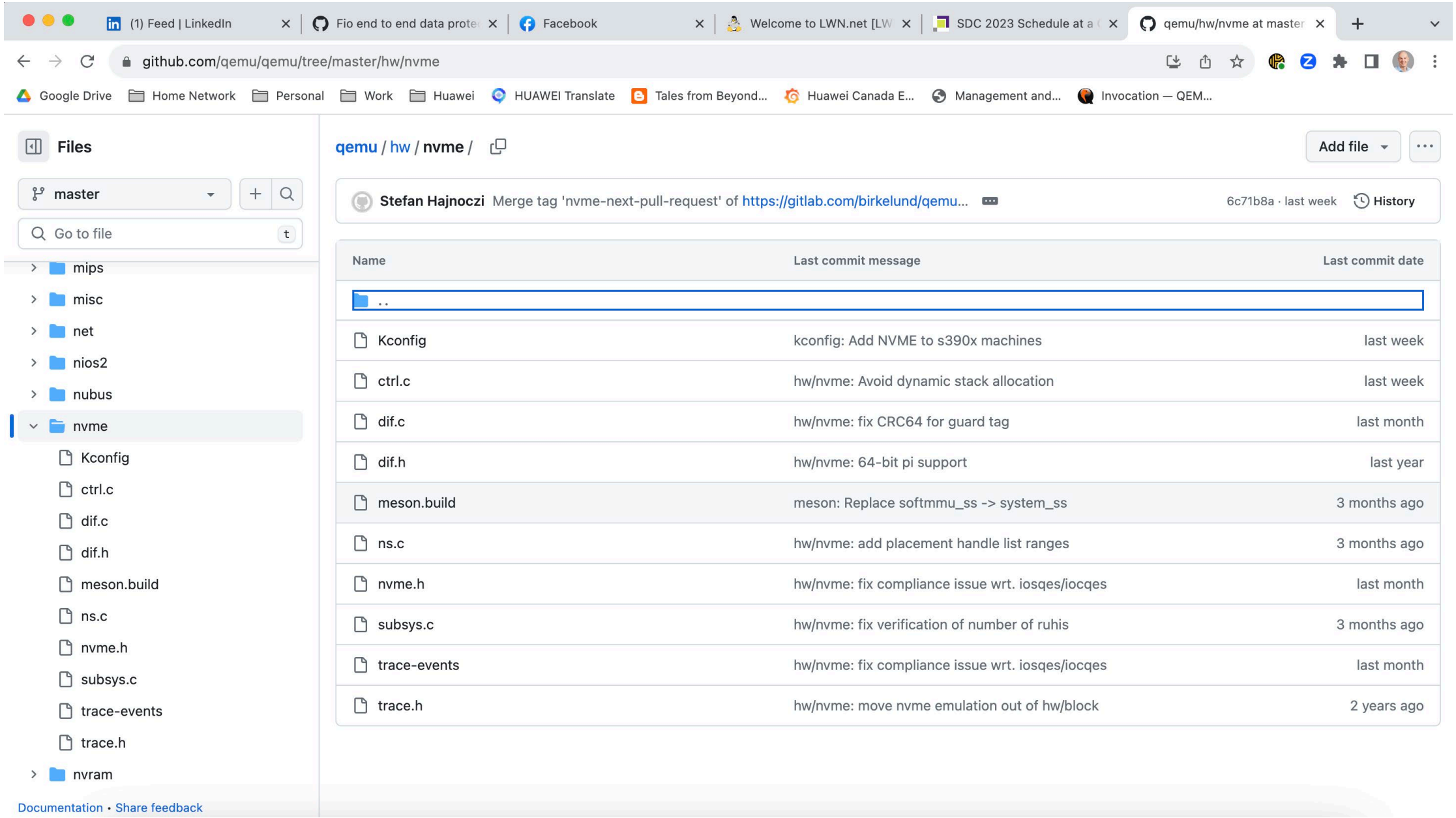
- batesste@bunbeg:~$ lspci
- 00:00.0 Host bridge: Red Hat, Inc. QEMU PCIe Host bridge
- 00:01.0 Ethernet controller: Red Hat, Inc. Virtio network device
- 00:02.0 Display controller: Red Hat, Inc. Virtio GPU (rev 01)
- 00:03.0 Audio device: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) High Definition Audio Controller (rev 01)
- 00:04.0 USB controller: NEC Corporation uPD720200 USB 3.0 Host Controller (rev 03)
- 00:05.0 USB controller: Red Hat, Inc. QEMU XHCI Host Controller (rev 01)
- 00:06.0 SCSI storage controller: Red Hat, Inc. Virtio block device
- 00:07.0 Non-Volatile memory controller: Red Hat, Inc. QEMU NVM Express Controller (rev 02)
- 00:08.0 Communication controller: Red Hat, Inc. Virtio console
- 00:09.0 Unclassified device [0002]: Red Hat, Inc. Virtio filesystem
- 00:0a.0 Unclassified device [00ff]: Red Hat, Inc. Virtio RNG

"I never knew Red Hat made NVMe SSDs!"


```
batesste@bunbeg:~$ sudo nvme list
Node             SN                   Model                                    Namespace Usage                      Format           FW Rev
---------------- -------------------- ---------------------------------------- --------- -------------------------- ---------------- --------
/dev/nvme0n1     5061A8FB-EF70-47BC-B QEMU NVMe Ctrl                           1         68.72  GB /  68.72  GB    512   B +  0 B   8.0.0
```

Of course, this is an emulated NVMe SSD

1. The code running inside the Virtual Machine (VM) has no way of knowing if this NVMe SSD is real or emulated.
2. The code running inside the VM is identical to the code that would talk to a real NVMe SSD. Same kernel driver and same nvme-cli userspace code.
3. We can add or remove features to this NVMe SSD by changing the **source code** of QEMU (see next slide). This means we can explore new NVMe features (either standard or vendor-specifc) before hardware becomes available:
    1. Key-Value capable NVMe SSDs.
    2. FDP capable NVMe SSDs.
4. Now assuming point 2 this means SW developers can write code for new hardware without needing new hardware. Cool!

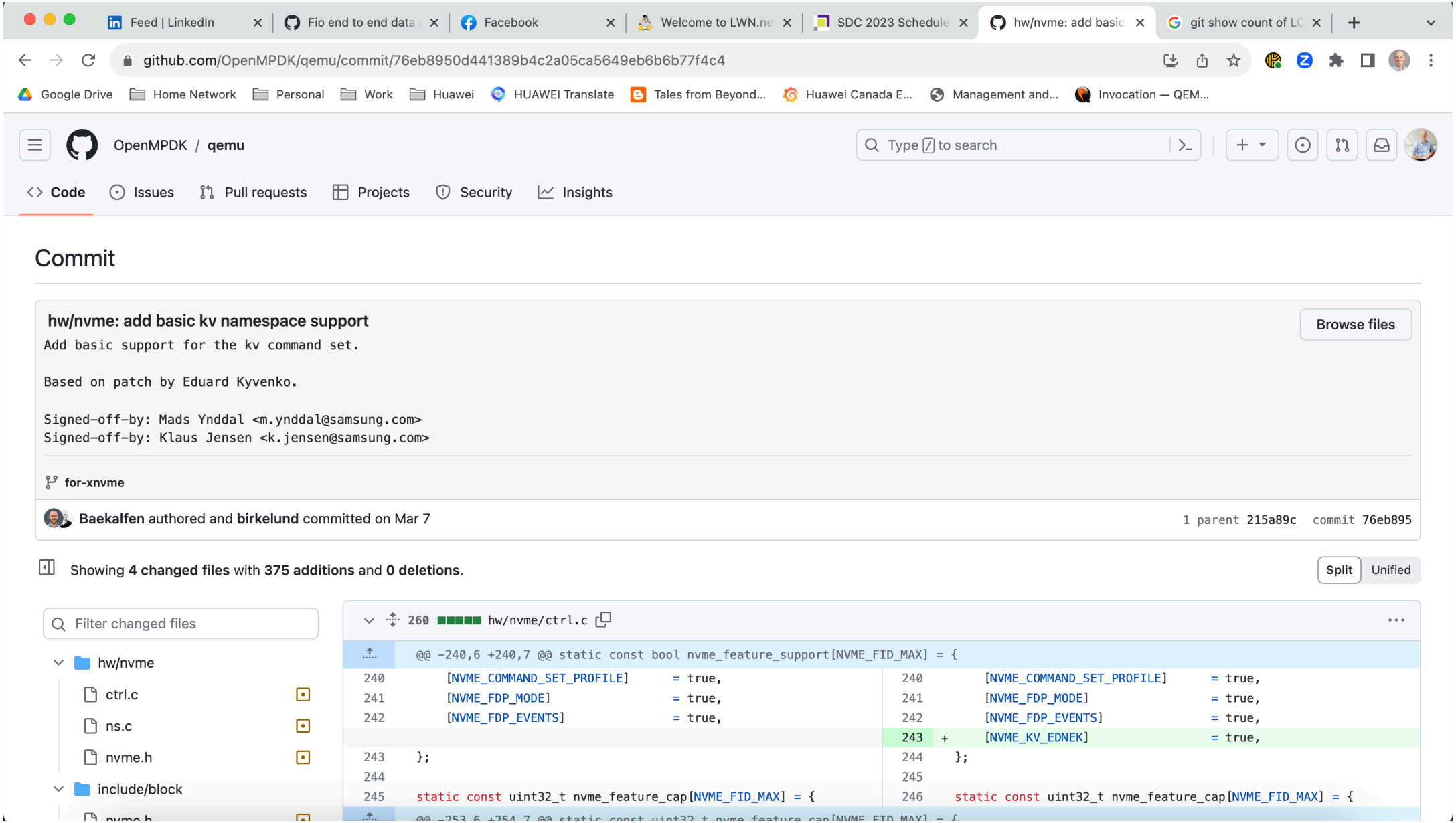


How to build a KV-capable NVMe SSD (the hard way)

1. Build a LBA-capable NVMe SSD.
2. Allocate a bunch of over-committed firmware engineers to the project. Ignore the howls of protest from sales, marketing, management etc.
3. Write firmware that implements the KV command set.
4. Debug firmware that implements the KV command set.
5. Validate your KV-capable NVMe SSD.

How to build a KV-capable NVMe SSD (the easy way)

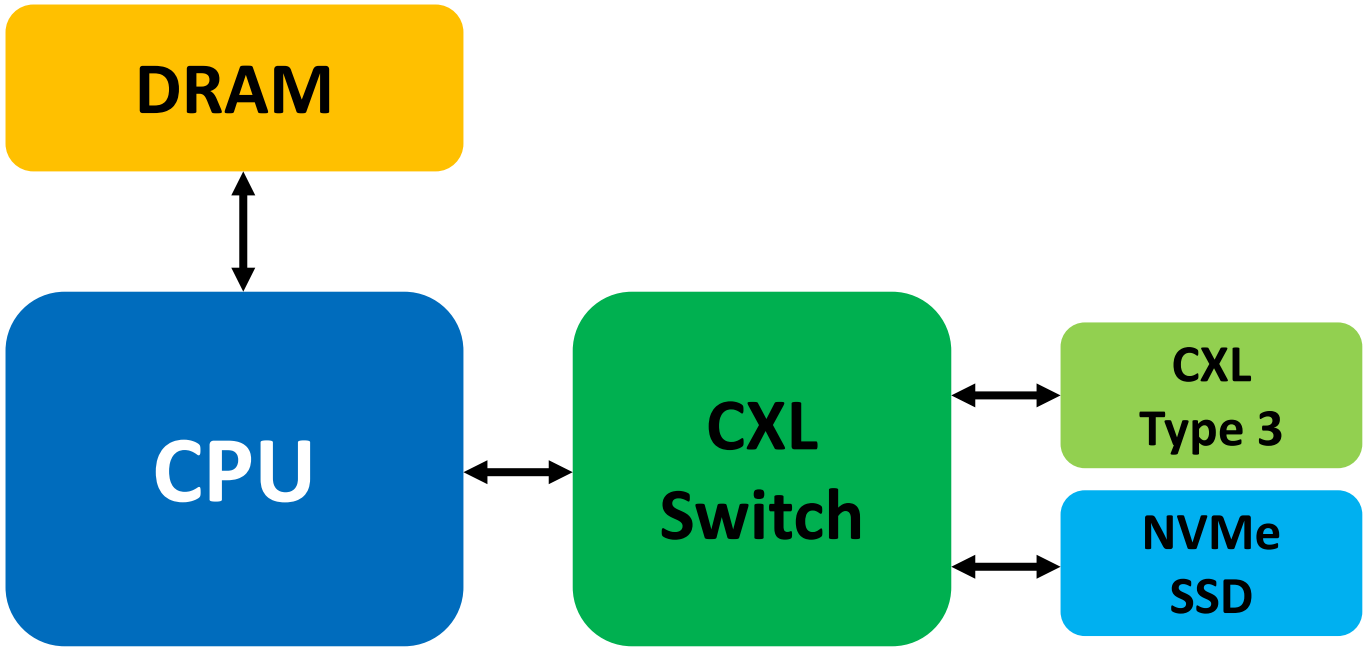
1. git clone git@github.com:qemu/qemu.git
2. cd qemu
3. git checkout –b key-value origin/main
4. Make about 379 LOC changes to the nvme software.
5. git commit –a –m “key-value: Add key-value command set to NVMe emulation model”
6. mkdir build && cd build && ../configure && make –j 32 all && sudo make install
7. Spin up a VM using this new version of qemu-system-<arch>
8. Check out and test your KV-capable NVMe SSD. If you find a problem repeat steps 4-8.
9. Upstream your code!
10. **Oh and Samsung have already done 1-8!!**



Emulation Example in QEMU

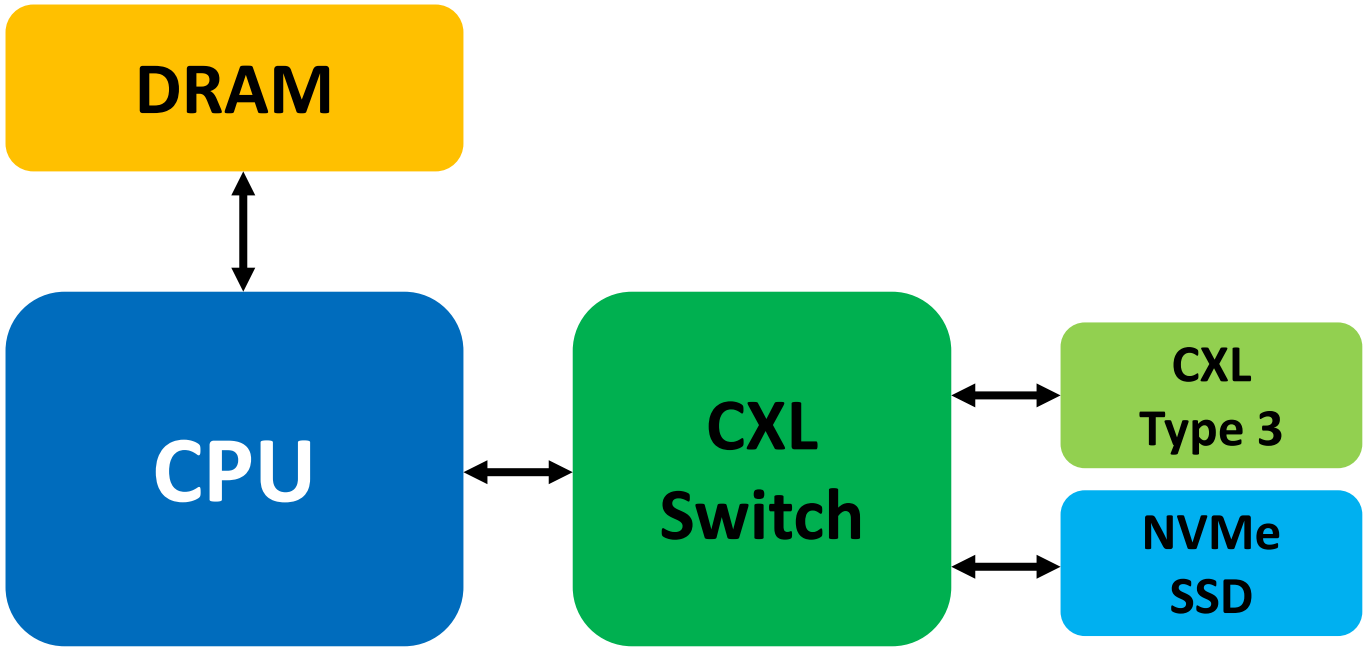
NVMe SSD below a CXL Switch in a Server Anyone?

This is Interesting. But I can’t build it today.



- NVMe/CXL SSD for memory expansion
- P2P DMA between NVMe SSD and Type 3 CXL device for swap/page-cache.
- I would like to have this hardware available so software developers can start working on code for the topics above and use-cases (like Computational Storage and Computational Memory)

This is Interesting. But I can't build it today. Or Can I?



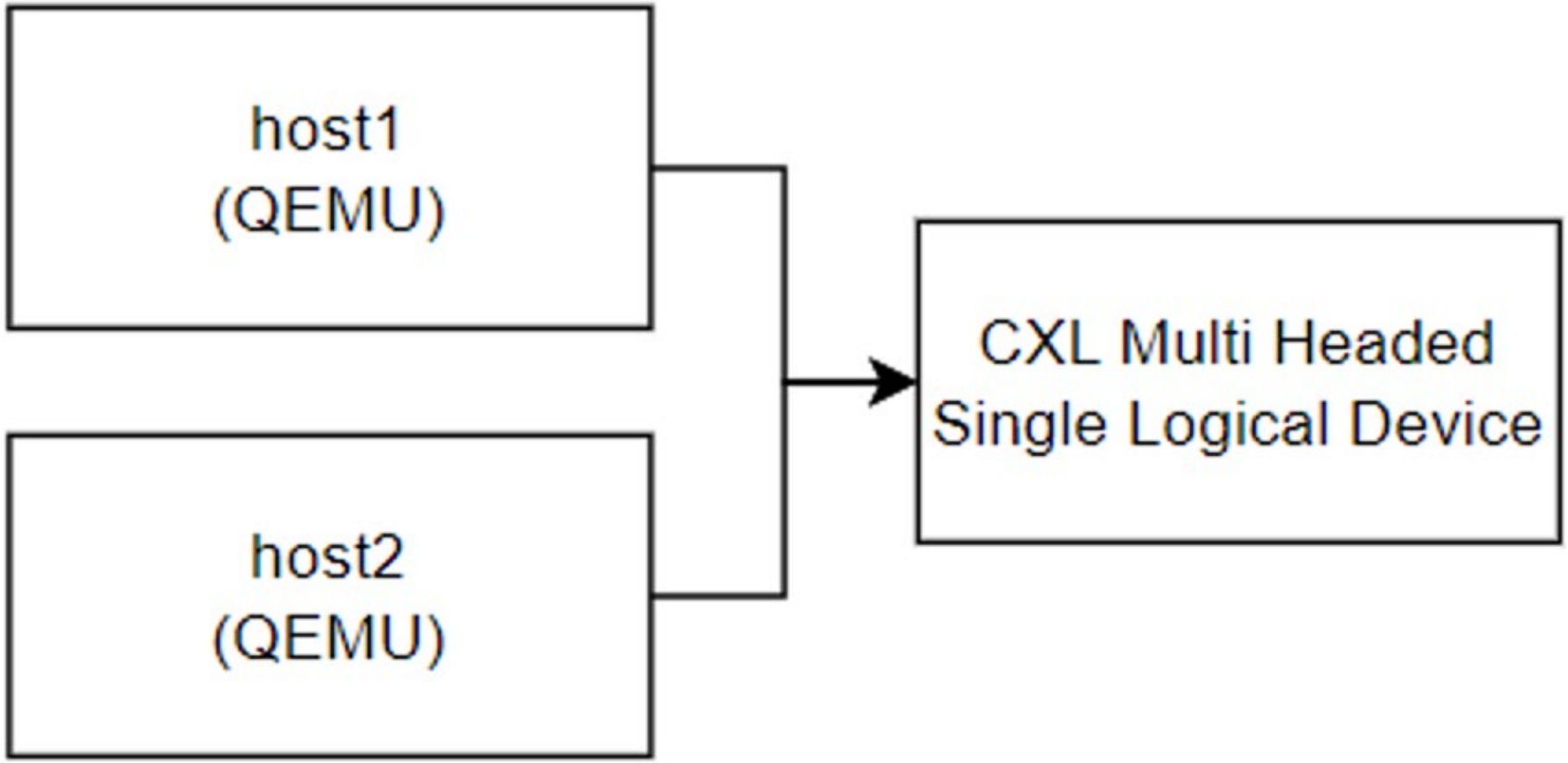
- NVMe/CXL SSD for memory expansion
- P2P DMA between NVMe SSD and Type 3 CXL device for swap/page-cache.
- I would like to have this hardware available so software developers can start working on code for the topics above and use-cases (like Computational Storage and Computational Memory)


```
qemu-system-x86_64 \
   -machine type=q35,hmat=on,nvdimm=on,cxl=on \
   -enable-kvm -cpu host,migratable=no \
   -nographic \
   -serial mon:stdio \
   -m 4G,maxmem=10G \
   -smp 4,sockets=1,maxcpus=4 \
   -numa node,nodeid=0,cpus=0-3,memdev=m0 \
   -drive if=none,file=./images/cxl.qcow2,format=qcow2,id=hd \
   -device virtio-blk-pci,drive=hd \
   -device e1000,netdev=user0 -netdev user,id=user0,hostfwd=tcp::2222-:22 \
   -rtc clock=host \
   -kernel $KERNEL -append "nokaslr norandmaps root=/dev/vda1 console=ttyS0 \
      earlyprintk=serial,ttyS0 ignore_loglevel printk_delay=0" \
   -object memory-backend-ram,id=m0,size=4G \
   -object memory-backend-file,id=cxl-mem1,share=on,mem-path=/tmp/cxltest.raw,size=256M \
   -object memory-backend-file,id=cxl-lsa1,share=on,mem-path=/tmp/lsa.raw,size=256M \
   -device pxb-cxl,bus_nr=52,bus=pcie.0,id=cxl.1 \
   -device cxl-rp,port=0,bus=cxl.1,id=root_port13,chassis=0,slot=2 \
   -device cxl-type3,bus=root_port13,volatile-memdev=cxl-mem1,lsa=cxl-lsa1,id=cxl-pmem0 \
   -M cxl-fmw.0.targets.0=cxl.1,cxl-fmw.0.size=256M
```

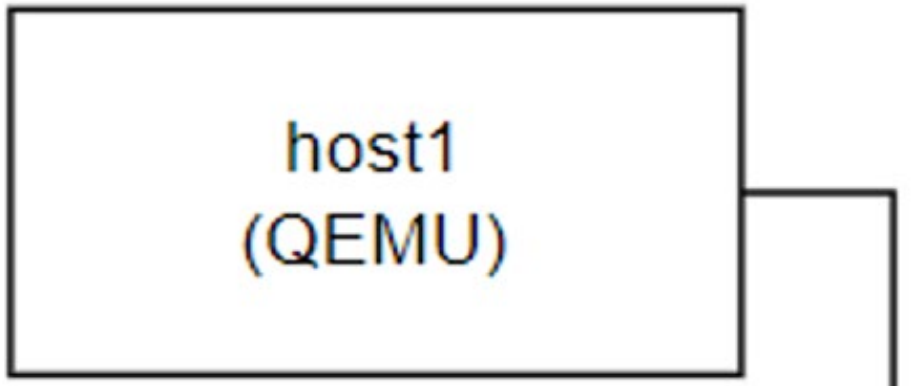

Emulation Example in QEMU

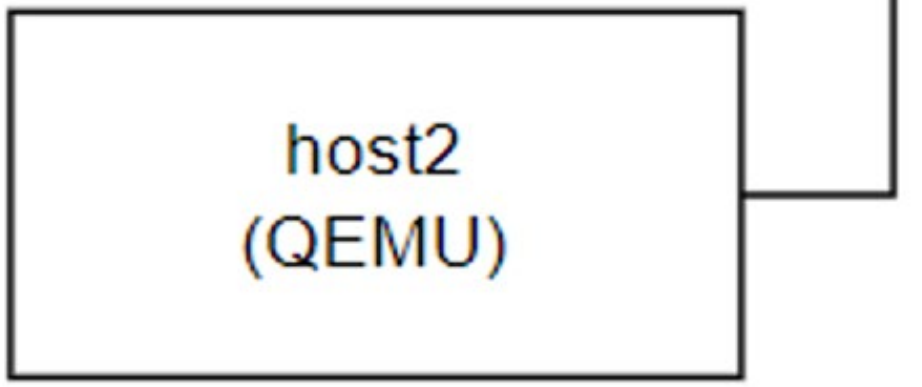
A Rack-Scale Architecture that includes CXL Memory Area Networking (MAN)

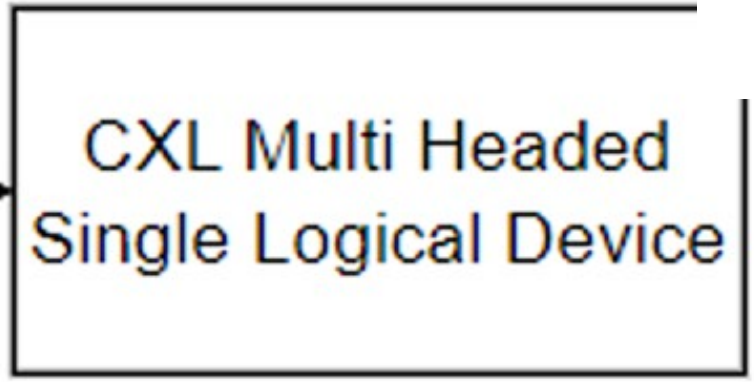
This is Interesting. But I can’t build it today.



This is Interesting. But I can’t build it today. Or Can I?







This is Interesting. But I can’t build it today. Or Can I?

https://memverge.com/cxl-qemuemulating-cxl-shared-memory-devices-in-qemu/

Emulation Example outside QEMU

A NVMe Computational Storage Drive (CSD using QEMU, SPDK and vfio-user)

Emulation outside of QEMU - I

- Up until now we have looked at emulations that are based on software that resides in the QEMU git tree.
- This is great but not all devices deserve to be upstream,.
- This is great but other code software applications exist.
- Can do emulation in a separate process to QEMU and connect it too QEMU via something like a socket?

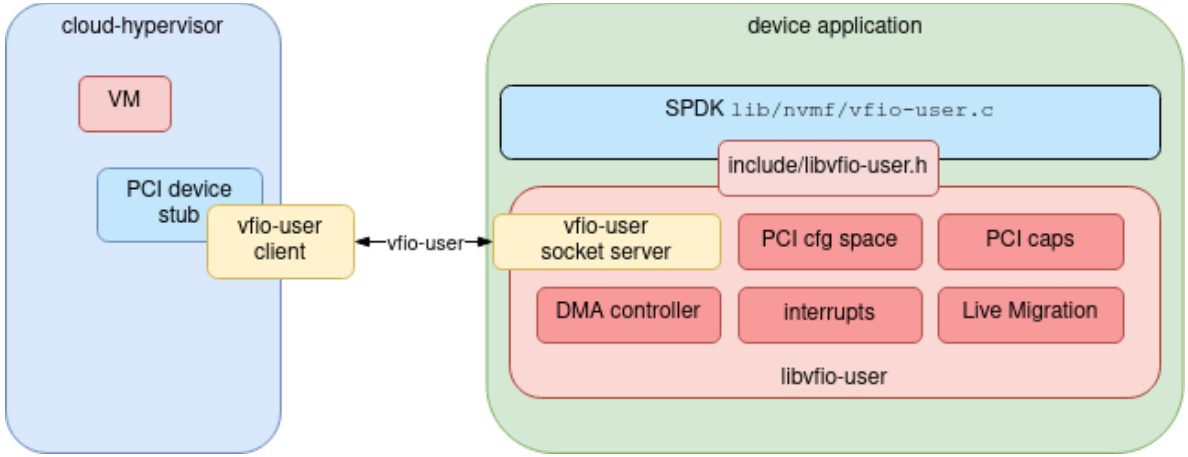
QEMU could leverage emulation code running in a separate process to the main qemu system emulation process.

Emulation outside of QEMU - II

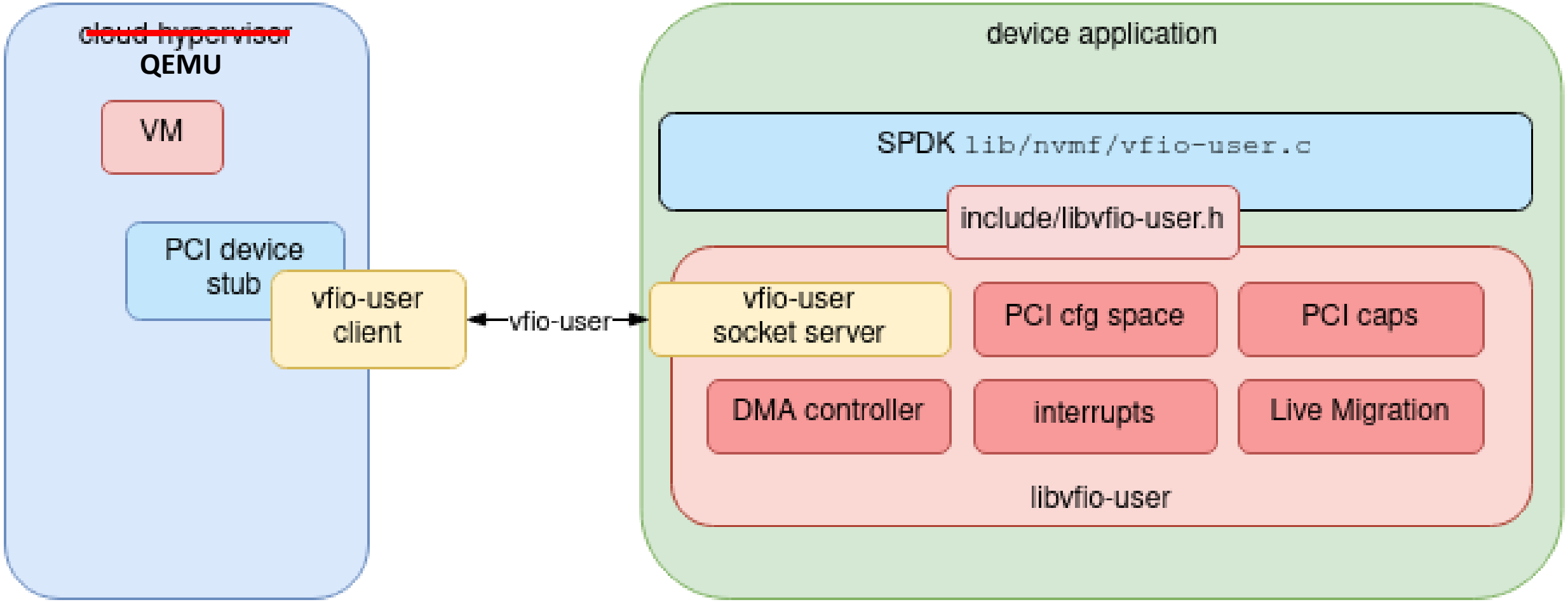
- A team at Nutanix have proposed vfio-user. A mechanism that allows PCIe devices to be emulated in a separate process to QEMU.
- vfio-user has a specification and consists of a server (the emulated device) and the client (QEMU or some other VMM).
- SPDK has been updated to support being an NVMe emulated device.

https://github.com/nutanix/libvfio-user

https://lists.gnu.org/archive/html/qemu-devel/2020-11/msg02458.html

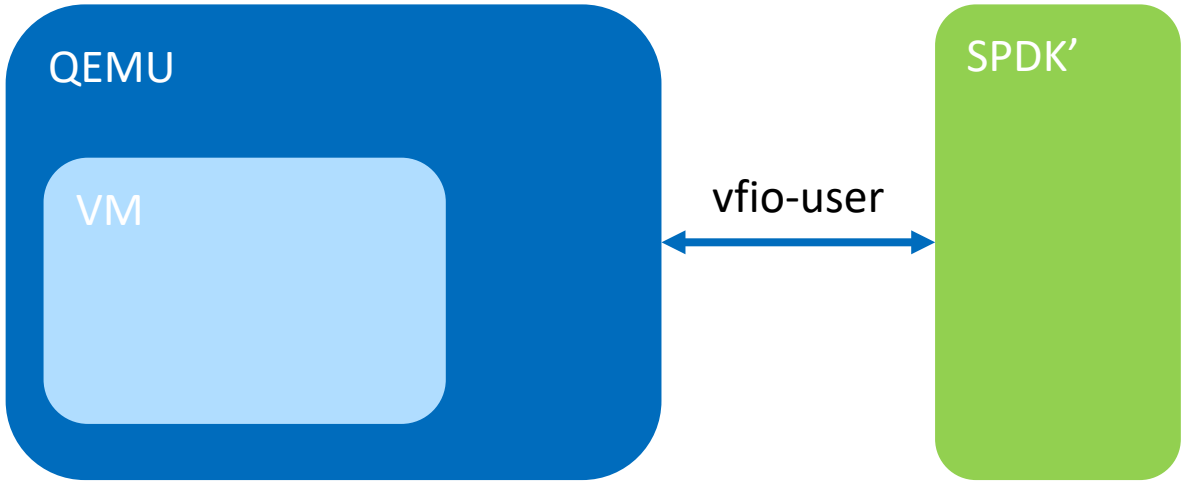


Emulation outside of QEMU - III



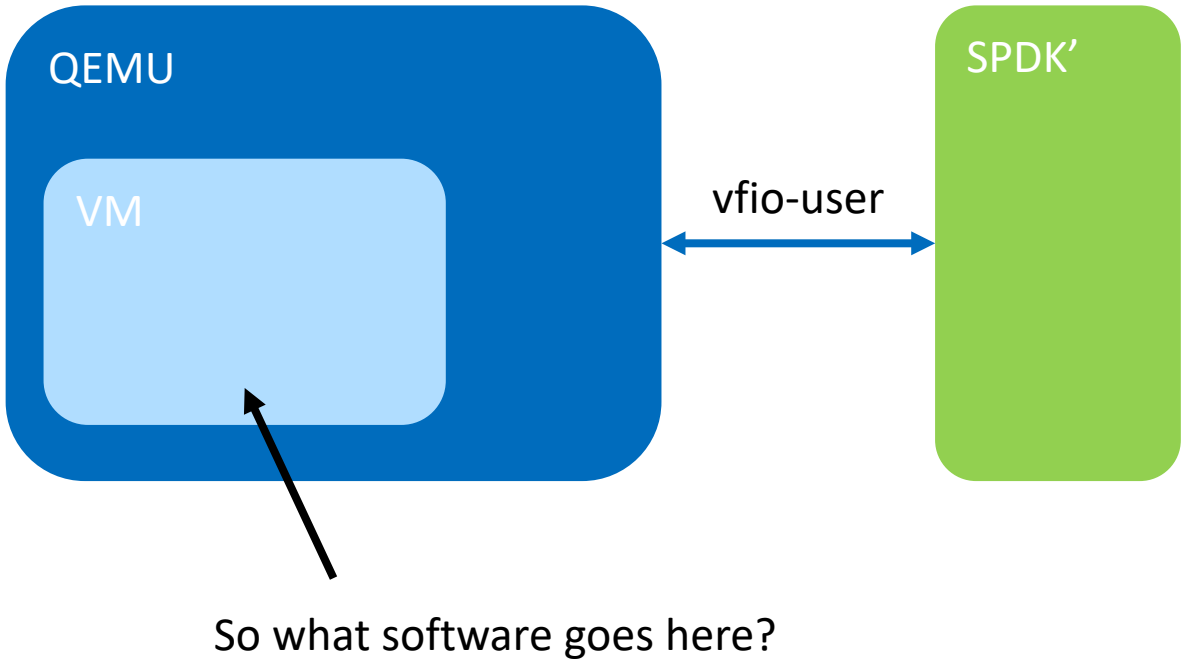
Emulation outside of QEMU - IV

- We can modify SPDK to support the upcoming TP 4091 and TP 4131 command sets that pertain to computational storage.
- We can then use QEMU and vfio-user to expose this version of SPDK as a NVMe CSD.
- We can then install software inside the VM to allow the VM and it's applications to leverage the emulated CSD.



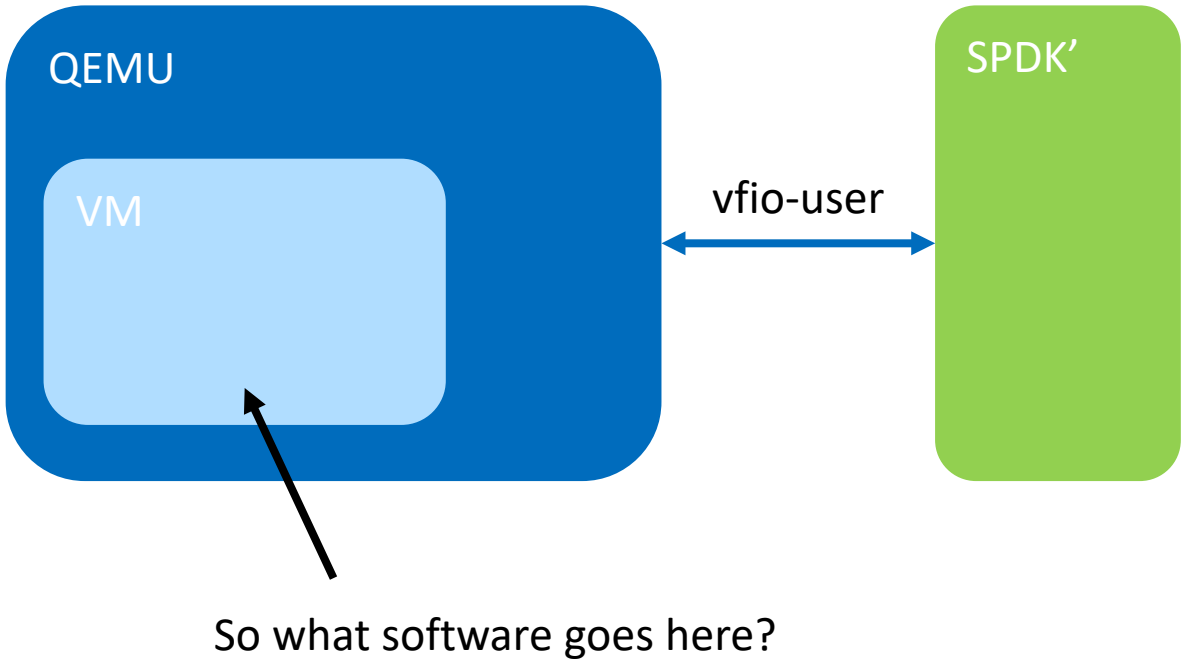
Emulation outside of QEMU - IV

- We can modify SPDK to support the upcoming TP 4091 and TP 4131 command sets that pertain to computational storage.
- We can then use QEMU and vfio-user to expose this version of SPDK as a NVMe CSD.
- We can then install software inside the VM to allow the VM and it's applications to leverage the emulated CSD.



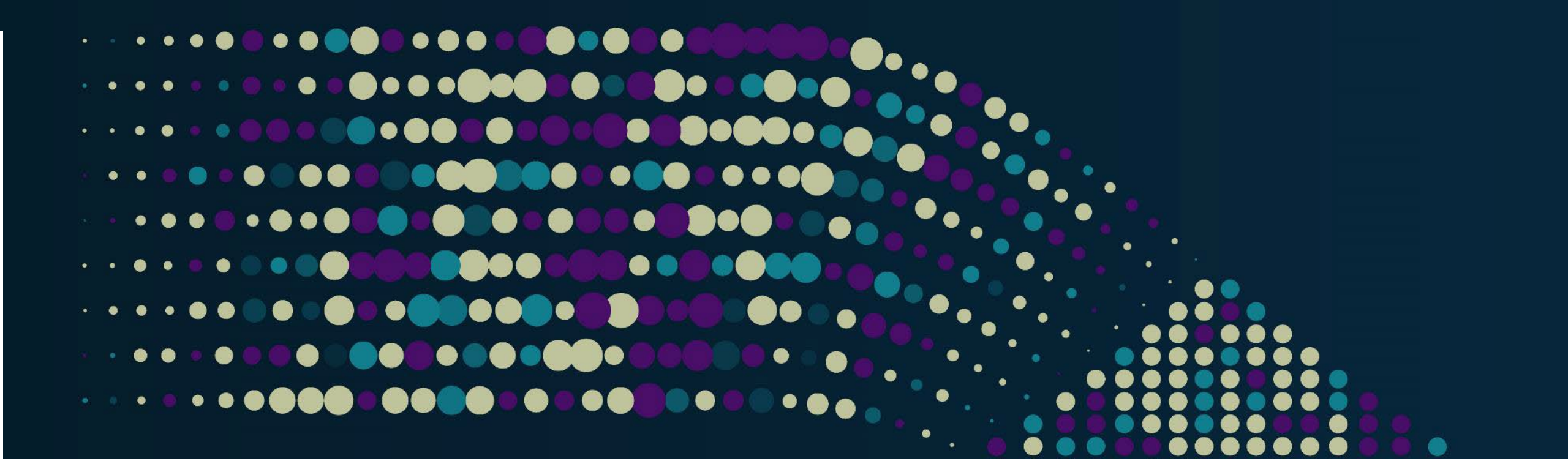
Emulation outside of QEMU - IV

- **Linux Kernel**: Can provide a path to NVMe namespaces (off all types) via io_uring_passthru.
- **xNVMe**: Can provide access to NVMe commands from new and emerging command sets.
- **SNIA's Comp. Storage API**: Can tie xNVMe to applications.



Conclusions

- Real hardware is HARD. And most of the time your software developers don't need it.
- QEMU is on a roll. A hugely successful hypervisor and system emulator. Get to know it!
- Emulation can be done inside the QEMU source tree. And this can be ISA emulation and/or hardware device emulation.
- Emulation of PCIe devices can be done outside the QEMU source tree via vfio-user. Keeps QEMU source tree clean.
- All the parts we need for CSD/CSP/CSA emulation are coming. Hopefully software devs can run with this to tie these devices to applications!



Please take a moment to rate this session.

Your feedback is important to us.

