QEMU Emulated NVMe

Lessons Learned and Future Work

Klaus Jensen <k.jensen@samsung.com>
Samsung Electronics

THE LINUX FOUNDATION

NVMe in **1** Slide

- **N**on-**V**olatile **M**emory **E**xpress
- Designed to exploit the low latency and inherent parallelism of NAND flash memory

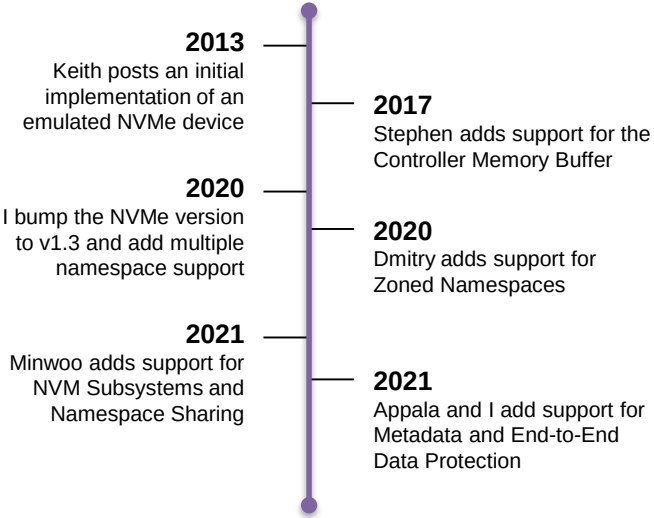
- *Core terminology required for this talk*
  - **Controller** – a PCI Express† function that acts as the interface between a **Host** and an **NVM Subsystem**
  - **Namespace** – a quantity of non-volatile memory, accessed independently from other namespaces, typically by logical block addresses
  - **NVM Subsystem** – A set of one or more **Controllers**, zero or more **Namespaces** and one or more **Ports**

† We will only consider the PCIe-based transport of NVMe

THE LINUX FOUNDATION

Emulated NVMe Timeline

- Initial implementation by Keith did the job for several years
- I was working with the OpenChannel SSD ecosystem in 2018-19 and was using QEMU on a day to day basis
  - Started to add missing mandatory features
  - Started working on multiple namespace support in an effort to upstream OCSSD support *(abandoned)*
  - Became a co-maintainer along with Keith in mid-2020



THE LINUX FOUNDATION

Emulated NVMe Timeline

- Things moved pretty fast
- As we shall see, sometimes a bit too fast

- **Some mistakes were made due to**
  - not knowing about best practices or how to effectively use the available APIs
  - not fully grasping QDev/QOM

THE LINUX FOUNDATION

KVM FORUM

So, lessons…

Speaking of APIs…

- Initially, I had trouble grok'ing QDev vs. QOM
  - Because - It's **not** a 'vs'
  - QDev **builds on** QOM
  - QDev provides an API tuned for setting up user created *devices*
  - **However**, QDev imposes a strict structure (ordered tree) where every alternating level is either a "bus" or a "device"

THE LINUX FOUNDATION

A bit of history

- NVMe device first introduced *(by Keith Busch, 2013)* with single-namespace support

  `-device nvme,drive=DRV`

- I wanted to add support for multiple namespaces, that would have separate block backends and separate `drive`-related parameters (`logical_block_size`, etc.)
  - Lots of helpful comments from the community
  - Ended up adding a new `-device nvme-ns` and plumb it with a QDev bus (it's sort-of how `hw/scsi` does it)

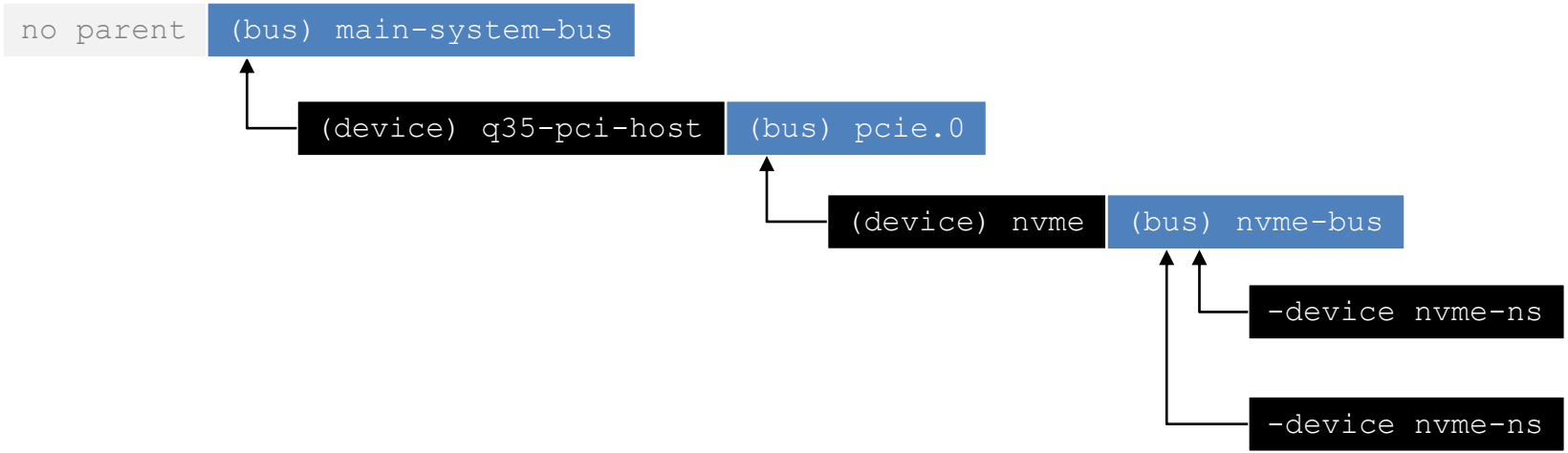
THE LINUX FOUNDATION

QDev Bus-based Plumbing

- The device nicely fits into the QDev tree and introspection just works (`info qtree`)
- If a device is removed, all devices on child busses are recursively unrealized
  - This design made a lot of sense when multiple namespace support was merged *(waaaay back in 2019)*
  - And this *should* be a good thing…
    - … but if we add subsystems and shared namespace functionality to the mix – not so much

THE LINUX FOUNDATION

NVMe Plumbing (pre-v6.0)



THE LINUX FOUNDATION

Shared Namespaces

- A **Shared Namespace** refers to a **Namespace** that may be accessed concurrently by two or more **Controllers** within the same **NVM Subsystem**
  - Quite useful for testing advanced drivers and **multi-path I/O**, so we wanted to support this

- Required adding the concept of an **NVM Subsystem** to `hw/nvme`

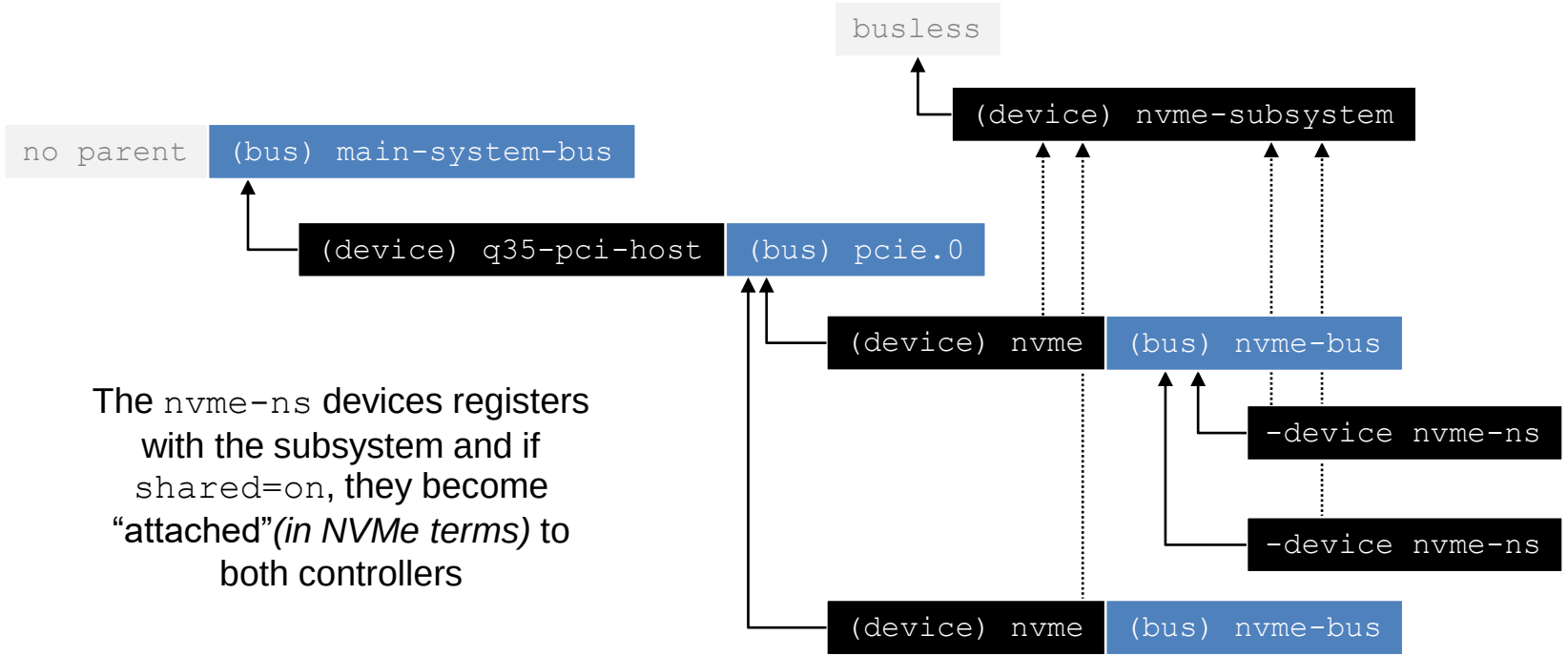
THE LINUX FOUNDATION

Mistake: Just use `-device`

- Because I *(or anyone else interested in `hw/nvme` at the time)* still didn't know any better we merged the subsystem support implemented as a **bus-less** and "**un-rooted**" `-device`
  - Added `subsys` link parameter on controller device to plumb it to a subsystem
  - It followed the design of the `nvme-ns` (namespace) device and it felt like the way to do it

THE LINUX FOUNDATION

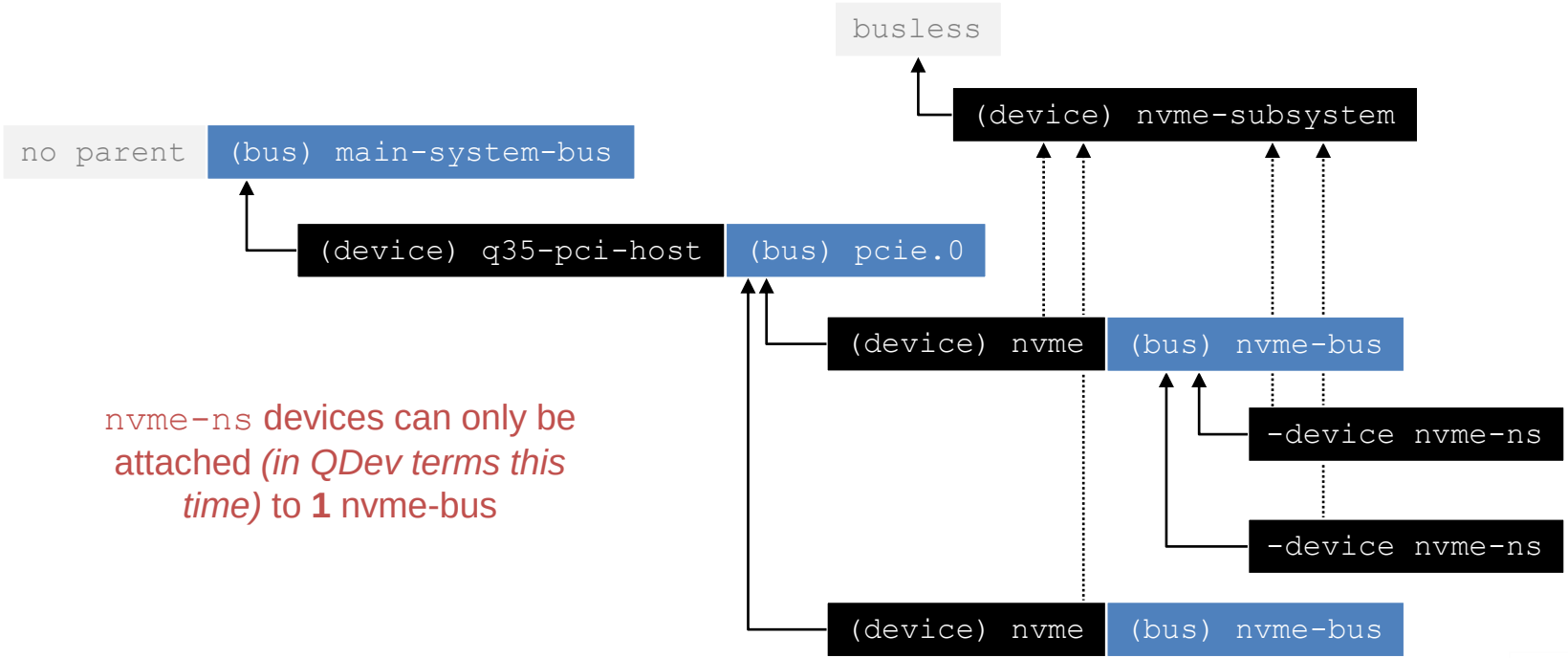
NVMe Plumbing (v6.0)



The `nvme-ns` devices registers with the subsystem and if `shared=on`, they become "attached"*(in NVMe terms)* to both controllers

THE LINUX FOUNDATION

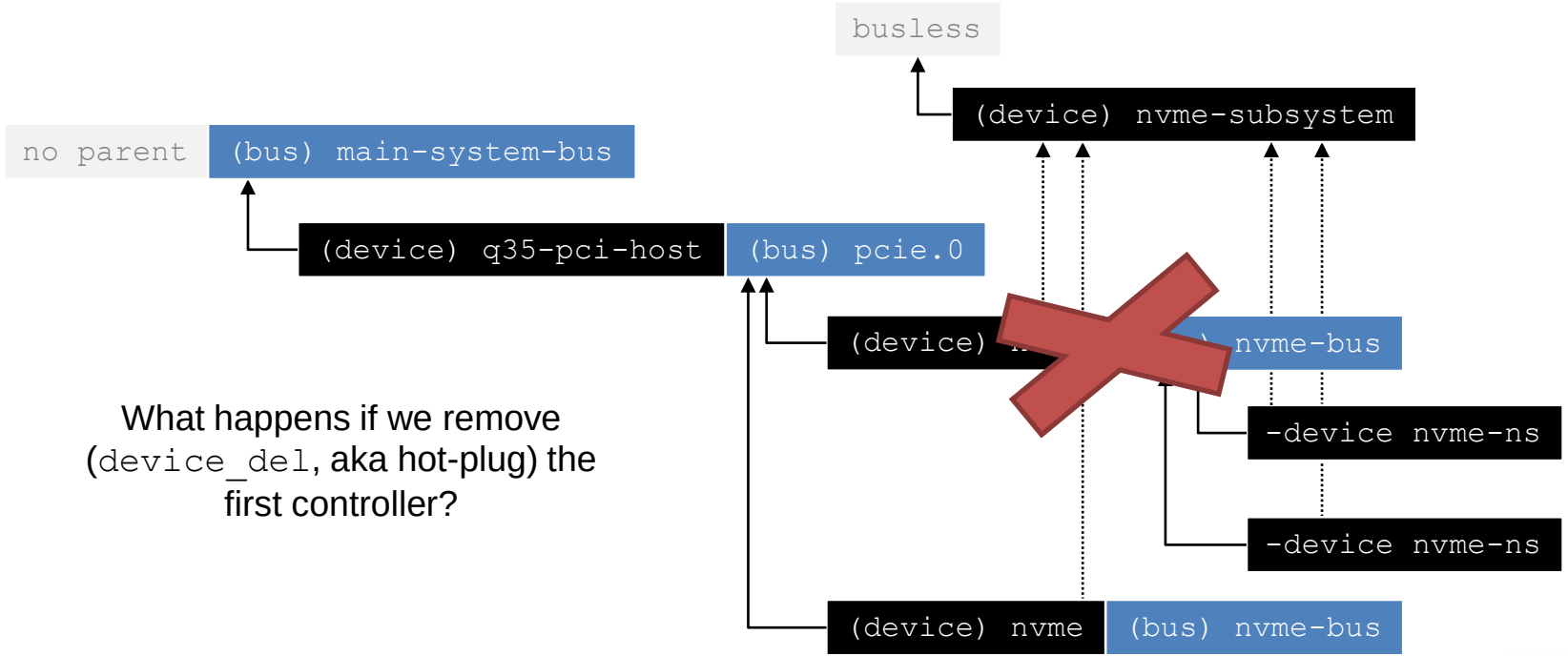
NVMe Plumbing (v6.0)



`nvme-ns` devices can only be attached *(in QDev terms this time)* to **1** nvme-bus

THE LINUX FOUNDATION

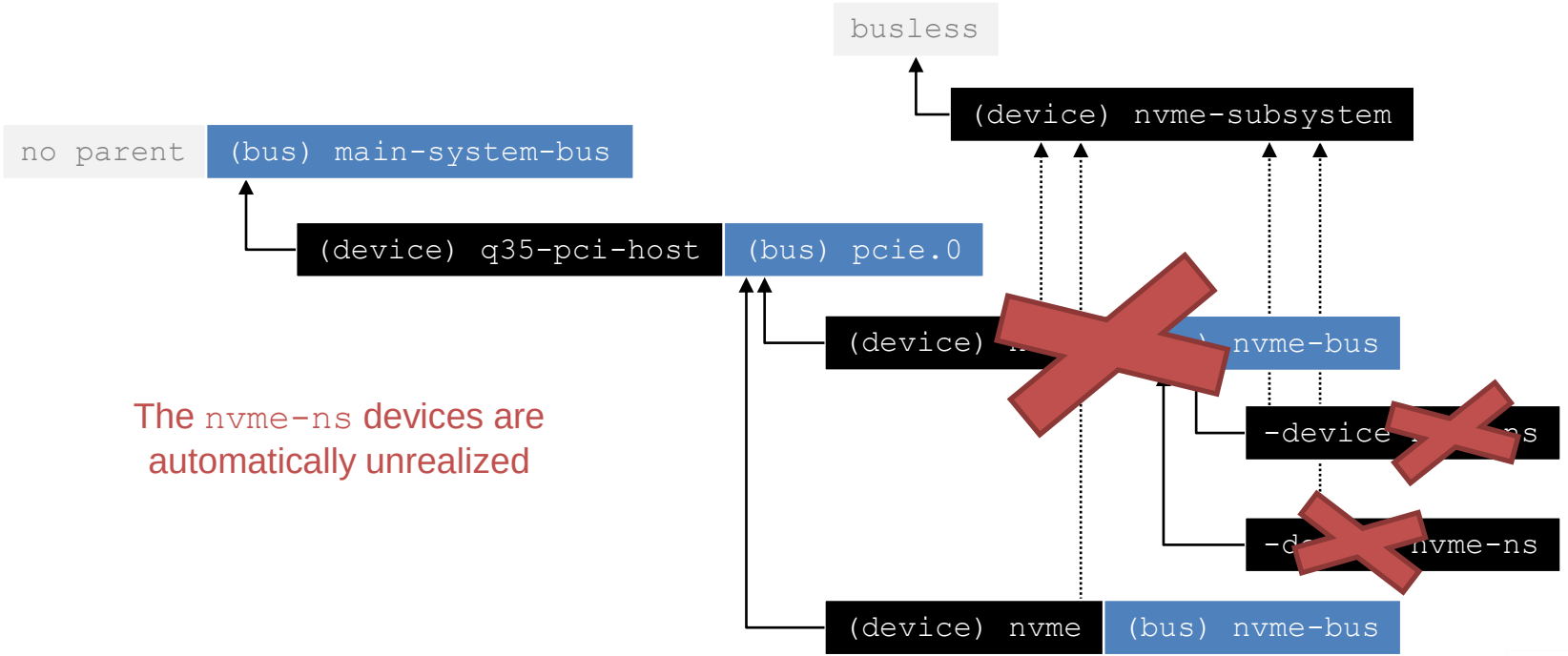
NVMe Plumbing (v6.0)



What happens if we remove (`device_del`, aka hot-plug) the first controller?

THE LINUX FOUNDATION

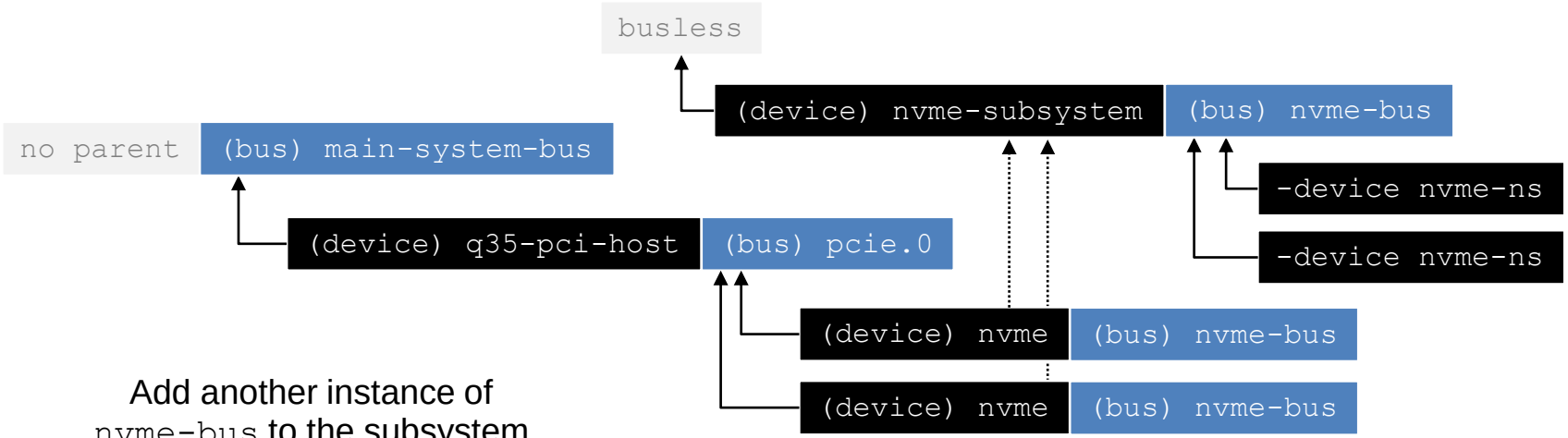
NVMe Plumbing (v6.0)



The `nvme-ns` devices are automatically unrealized

THE LINUX FOUNDATION

NVMe Plumbing (the “fix”)



Add another instance of `nvme-bus` to the subsystem device and re-parent the `nvme-ns` devices if a subsystem is configured

THE LINUX FOUNDATION

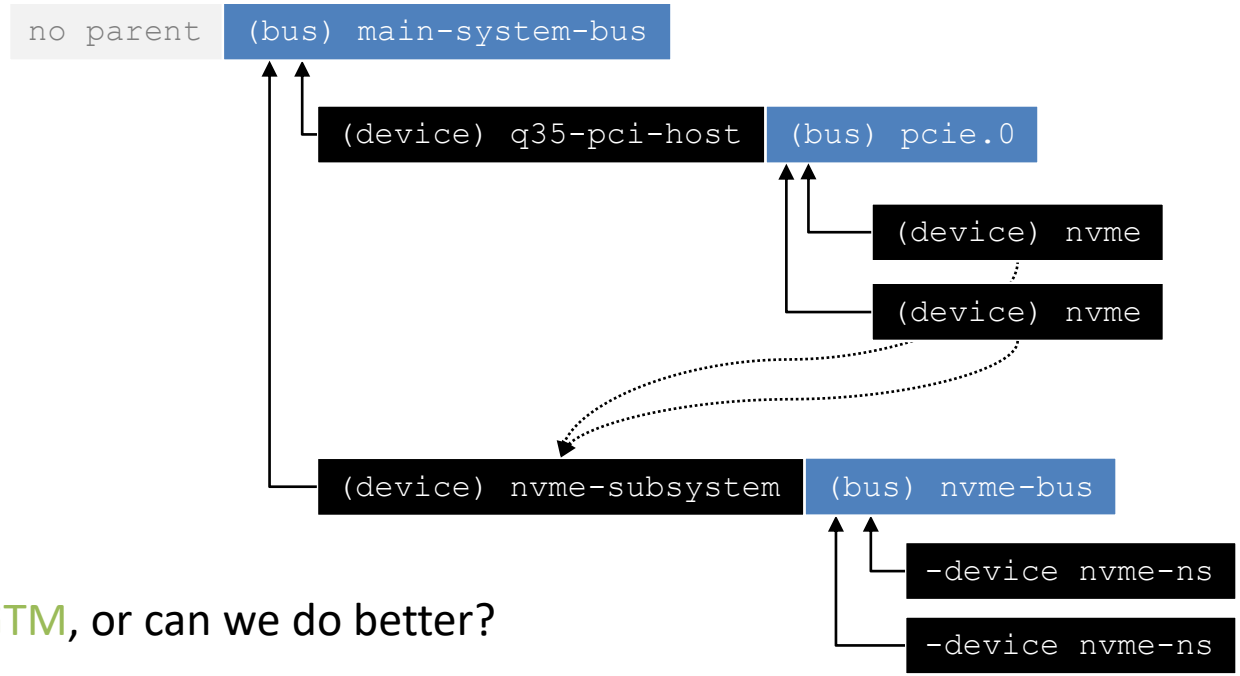
How do we fix this properly?

- Implement a hot-plug handler to “fail-over” the namespaces to another controller?

- If we started from scratch, we could
  - make the `nvme-subsys` device a *system bus device* that exposes an `nvme-bus`
  - keep the `nvme-ns` devices as-is and they would attach to the `nvme-bus` created by the subsystem instead
  - remove the `nvme-bus` that the `nvme` controller device used to create

THE LINUX FOUNDATION

How do we fix this properly?



LGTM, or can we do better?

THE LINUX FOUNDATION

How do we fix this properly?

- Potential issues with the system bus approach
  - Backwards compatibility issues…
    - Requires adding new devices and deprecating existing ones
  - The `nvme` controller device must still be attached under the PCI bus, so it must backlink itself out of the tree to reach the subsystem device anyway

Retains the idea of subsystems and namespaces being considered *devices*

THE LINUX FOUNDATION

How do we fix this properly?

- `hw/scsi` also separates the controller from the drives and exploits QDev buses
  - It uses `-device` for both controllers and drives, so something must be right here?
  - But, the one-parent restriction have also impeded the addition of multipath I/O in `hw/scsi`

    Hannes Reinecke attempted this back in 2017 and noted that he would constantly run into the restrictions of the ordered tree

- *No other QEMU subsystem seems to support “shared” block devices like `hw/nvme` does*

THE LINUX FOUNDATION

Rethinking the model

- ***What if…*** Subsystems and Namespaces were **not** modeled as *devices*
  - Neither subsystems or namespaces expose virtual hardware (e.g. memory, IRQs,…) to the guest
  - Conceptually, a subsystem is the parent of controllers, but a subsystem is not a PCI device *(how would this fit into the qtree?)*
  - Namespace may be associated with *(i.e., children of)* multiple controllers *(but in QDev, devices can only have one parent)*
  - Fundamentally, they are *concepts* in a device model and happen to benefit design-wise from being independent devices

- User creatable objects (`-object`) might be more appropriate

THE LINUX FOUNDATION

Rethinking the model

- No existing devices use `-object`'s like this
  - Use of `-object memory-backend-{file,memfd,ram}` comes close (i.e NVMe PMR support)

- I gave it a shot

THE LINUX FOUNDATION

The `hw/nvme` “devpocalypse”

- RFC patch series posted late August
  - `13 files changed, 2519 insertions(+), 1164 deletions(-)`
  - Ouch… considering `hw/nvme` is ~8500 LOC in total

- Major refactoring of the `hw/nvme` subsystem
  - Introduces NVM Subsystems and Namespaces as user creatable objects

THE LINUX FOUNDATION

The `hw/nvme` “devpocalypse”

- Series goals
  - Introduce a new experimental controller device (`x-nvme-ctrl`)
  - Introduce new experimental user creatable objects
    - `x-nvme-ns-{nvm,zoned}`
    - `x-nvme-subsystem`

THE LINUX FOUNDATION

The `hw/nvme` “devpocalypse”

- Series goals *(continued)*
  - Exploit the QEMU Object Model
    - The `x-nvme-ns` abstract object provides the base implementation of NVMe namespace types
    - The `x-nvme-ns-zoned` derives from the `x-nvme-ns-nvm` object

THE LINUX FOUNDATION

The `hw/nvme` “devpocalypse”

- Series goals *(continued)*
  - Retain backwards compatibility by keeping the existing devices around
    - Uses the new object code internally, **no code duplication**
    - Deprecate the subsystem and namespace devices as the experimental objects stabilizes

THE LINUX FOUNDATION

The `hw/nvme` “devpocalypse”

- Perks of introducing brand new models
  - Easy clean-up of some confusing device parameters
    - **Controller**
      - `msix_qsize` → `max-intr-vectors`
      - `aer_max_queued` → `max-aer-retention`
      - Remove unofficially deprecated `num_queues`, `use-intel-id`
    - **Namespace**
      - Fix Simple Copy related parameters that should have been defined in bytes and not LBAs
      - Remove the `eui64-default` compatibility parameter
      - `detached`, `shared` → `attach-to`
    - **Subsystem**
      - `nqn` → `subnqn`

THE LINUX FOUNDATION

The `hw/nvme` “devpocalypse”

- Set up a subsystem

```
-object x-nvme-subsystem,id=subsys-1
```

THE LINUX FOUNDATION

The `hw/nvme` “devpocalypse”

- Adding controllers

```
-device x-nvme-ctrl,id=ctrl-1,subsys=subsys-1
```

THE LINUX FOUNDATION

The hw/nvme “devpocalypse”

- Adding namespaces and attach to specific controllers

```
-object x-nvme-ns-nvm,id=ns-nvm-1,subsys=subsys1, \
    attached-to=ctrl-1, \
    attached-to=ctrl-3

-object x-nvme-ns-zoned,id=ns-zoned-1,subsys=subsys1, \
    attached-to=all
```

THE LINUX FOUNDATION

-device vs -object

- Properties are more verbose to define

```
DEFINE_PROP_UUID("uuid", NvmeNamespace, uuid)

char *get_uuid(Object *obj, Error **errp) {
  NvmeNamespace *ns = NVME_NAMESPACE(obj);
  char *str = g_malloc(UUI_FMT_LEN + 1);
  qemu_uuid_unparse(&ns->uuid, str);
  return str;
}

void set_uuid(Object *obj, const char *v, Error **errp) {
  NvmeNamespace *ns = NVME_NAMESPACE(obj);

  if (qemu_uuid_parse(v, &ns->uuid) < 0) {
    error_setg(errp, "invalid UUID");
  }
}

object_property_add_str(oc, get_uuid, set_uuid);
```

THE LINUX FOUNDATION

`-device` vs `-object`

- No “realize” phase as in QDev
  - Use a “machine done notifier” to emulate this
- If you do not use object composition (`object_initialize_child`), **you** are responsible for cleaning up

THE LINUX FOUNDATION

Lesson Learned

- Consider all your options when deciding on your model
  - Should my device be split into individual parts?
  - Is this part really a `-device` or an `-object`?
    - Aka, does it **behave** like a device? (i.e. expose virtual hardware or memory regions?)
    - *… does it quack?*
  - The flexibility of a user creatable object might be just what you are looking for…
    - *… if you can get by without the luxury of the QDev APIs!*

THE LINUX FOUNDATION

KVM FORUM

Future Work

Future Work

- Get rid of the `QEMUSGList/QEMUIOVector` duality
  - The device code deals with both
    - `QEMUSGList`'s for use with the DMA-helpers (controller/host transfers)
    - and with `QEMUIOVector`'s for controller-only transfers (CMB, PMR, Verify, Copy, etc.)
  - Consider open-coding DMA-mapping and transfer while processing command payloads (PRPs/SGLs) incrementally
    - Removes the need for the temporary `QEMUSGList` data structure
    - Allow incremental T10 Data Integrity calculations

THE LINUX FOUNDATION

Future Work

- Polling NVMe drivers relies on continuously reading the completion queue head for a change in the *phase bit* instead of waiting for an interrupt
  - Profiling using QEMU is not easy since there are a lot of various places that introduce latency in the emulated device
  - Limit these latencies
    - In-memory and no-op I/O
    - Use `iothread` for queue processing

THE LINUX FOUNDATION

Future Work

- Para-virtualization features in NVMe
  - Shadow Doorbell and EventIdx buffers
    - The host may provide two separate memory buffers that mirror the controller doorbell registers
    - Reduces number of vmexits by reducing MMIO
  - Some existing patches floating around from when this was a Google Vendor Extension

- Can the emulated device *possibly* be made faster (latency-wise) than available hardware?
  - Or, how low can we go? Good enough for profiling?
  - Is this a hopeless endeavor?

THE LINUX FOUNDATION

KVM FORUM