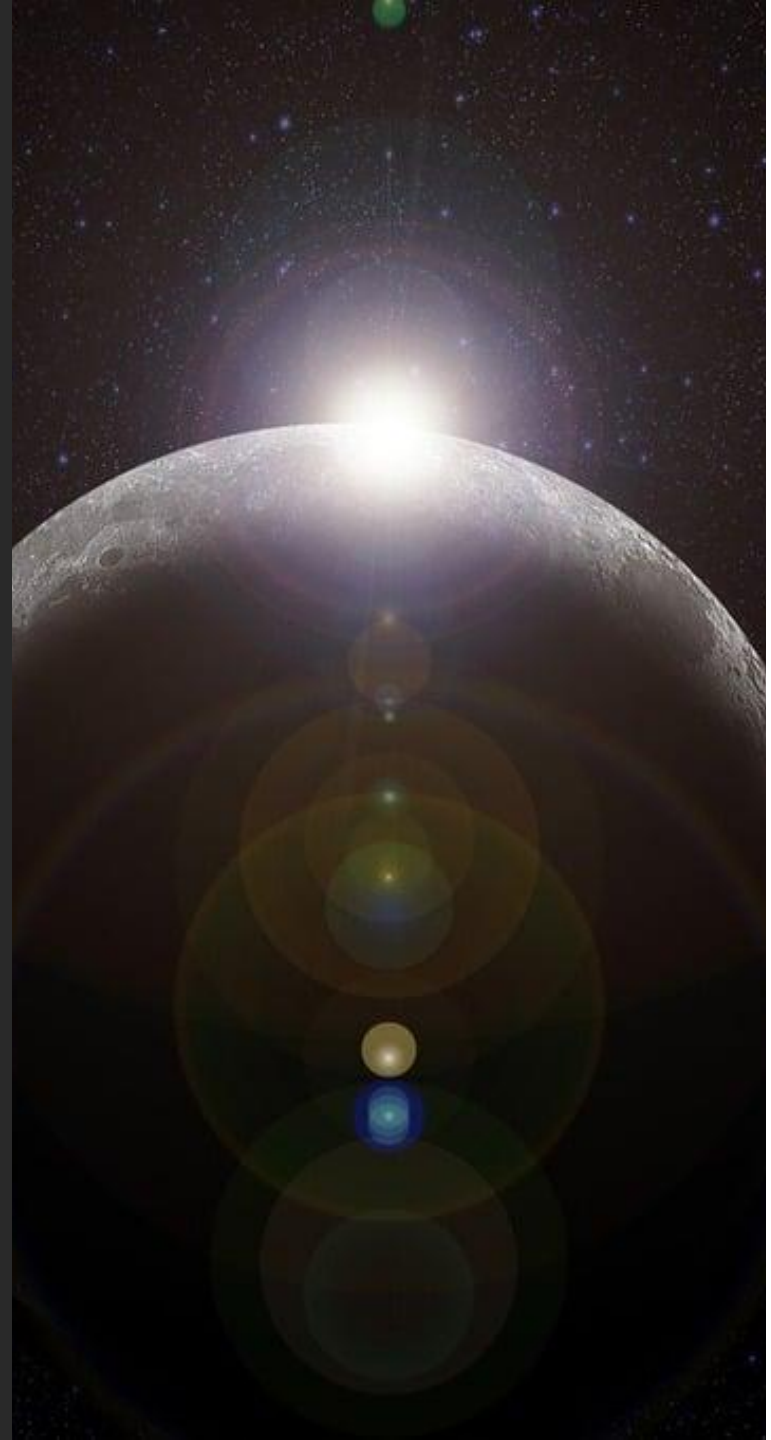


Unleashing VFIO's Potential

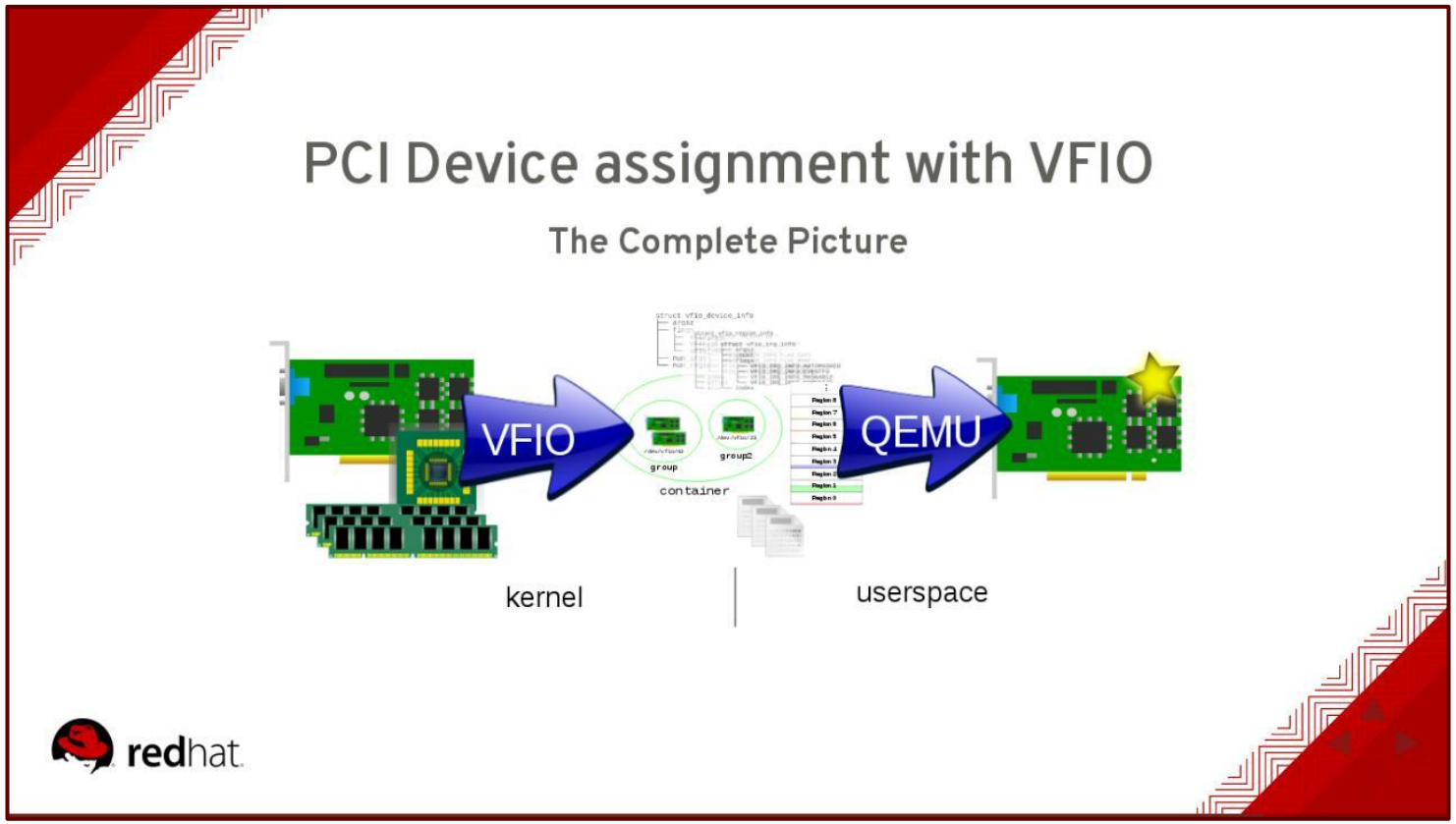
Code refactoring and new frontiers in device virtualization

Alex Williamson
VFIO Maintainer

Red Hat

VFIO Refresher

Device assignment into virtual machines

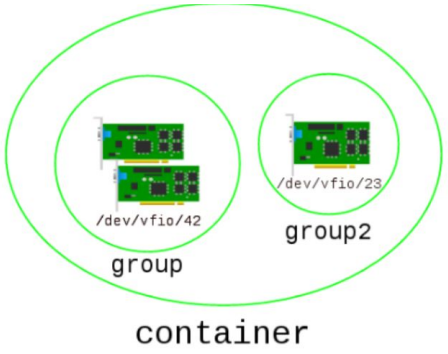


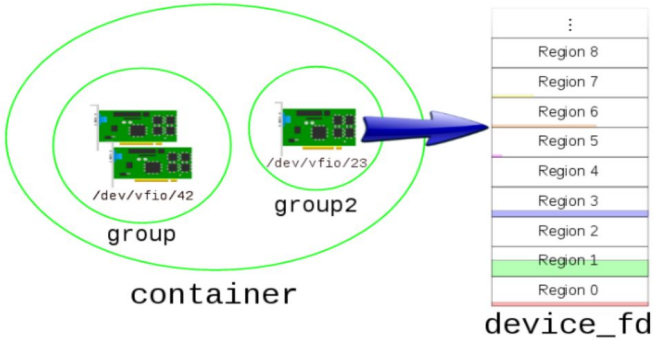
KVM Forum 2016, "An Introduction to PCI Device Assignment with VFIO"
http://www.linux-kvm.org/page/KVM_Forum_2016

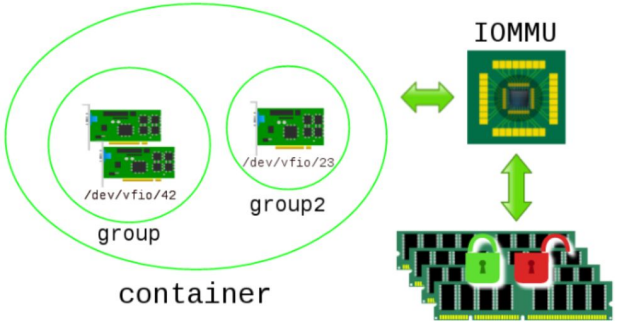
Red Hat

The VFIO Ecosystem

"Legacy" VFIO







Containers, Groups, and Devices

- Groups: one or more devices
- Container: shared IOMMU context
- Device access in protected context

Devices

- Regions: segments of device fd
- Access to device resources
- Interrupts provided via eventfd*

DMA

- VFIO IOMMU backends
- Configured through container

* Not pictured

Red Hat

Agenda:

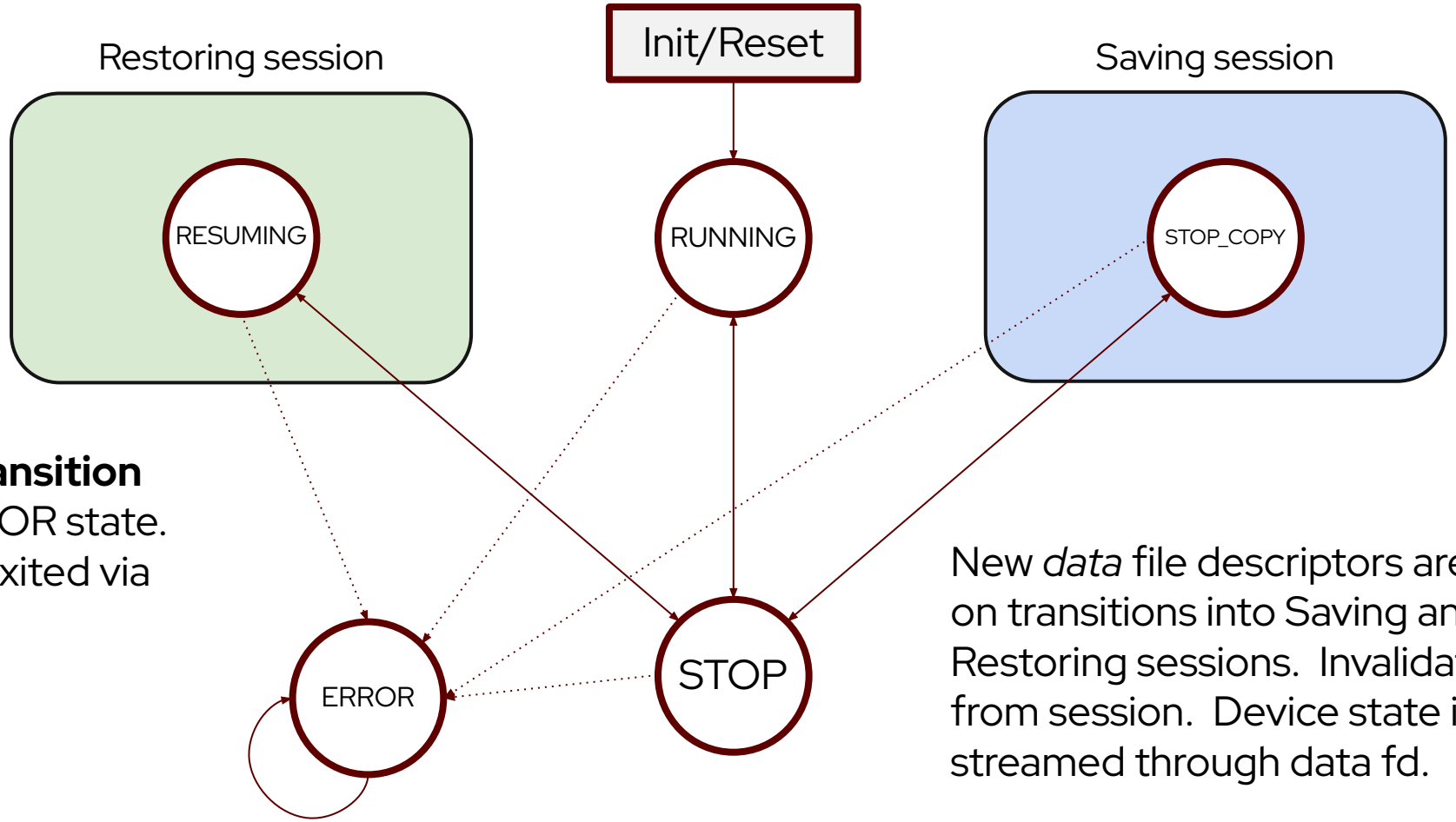
- Live Migration
- Variant Drivers
- IOMMUFD

Red Hat

VFIO Live Migration

- VFIO protocol developed to support device live migration
- Version 1 developed as a region based protocol
  - Data read and written through defined offset of device fd
  - No restrictions on device state transitions
  - Never adopted by in-kernel drivers
  - Considered too complicated, **removed**
- Version 2 developed as a streaming protocol
  - New file descriptor generated for migration session
  - Finite state machine defining fixed arcs through states

VFIO Live Migration Minimal State Machine

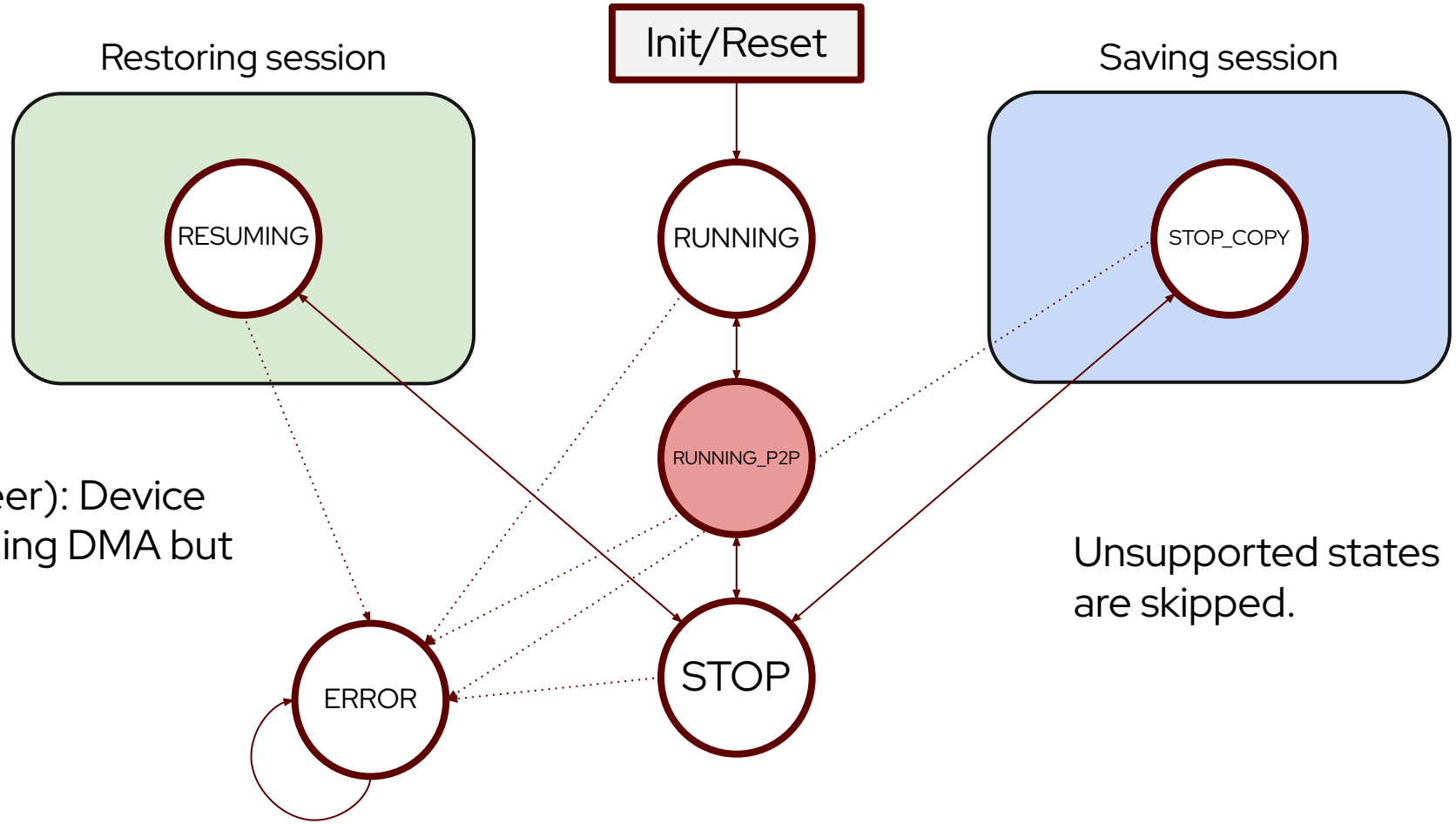


Any fault during **transition** may enter the ERROR state. Error state is only exited via device reset.

New *data* file descriptors are created on transitions into Saving and Restoring sessions. Invalidated on exit from session. Device state is streamed through data fd.

Red Hat

VFIO Live Migration Minimal State Machine + P2P

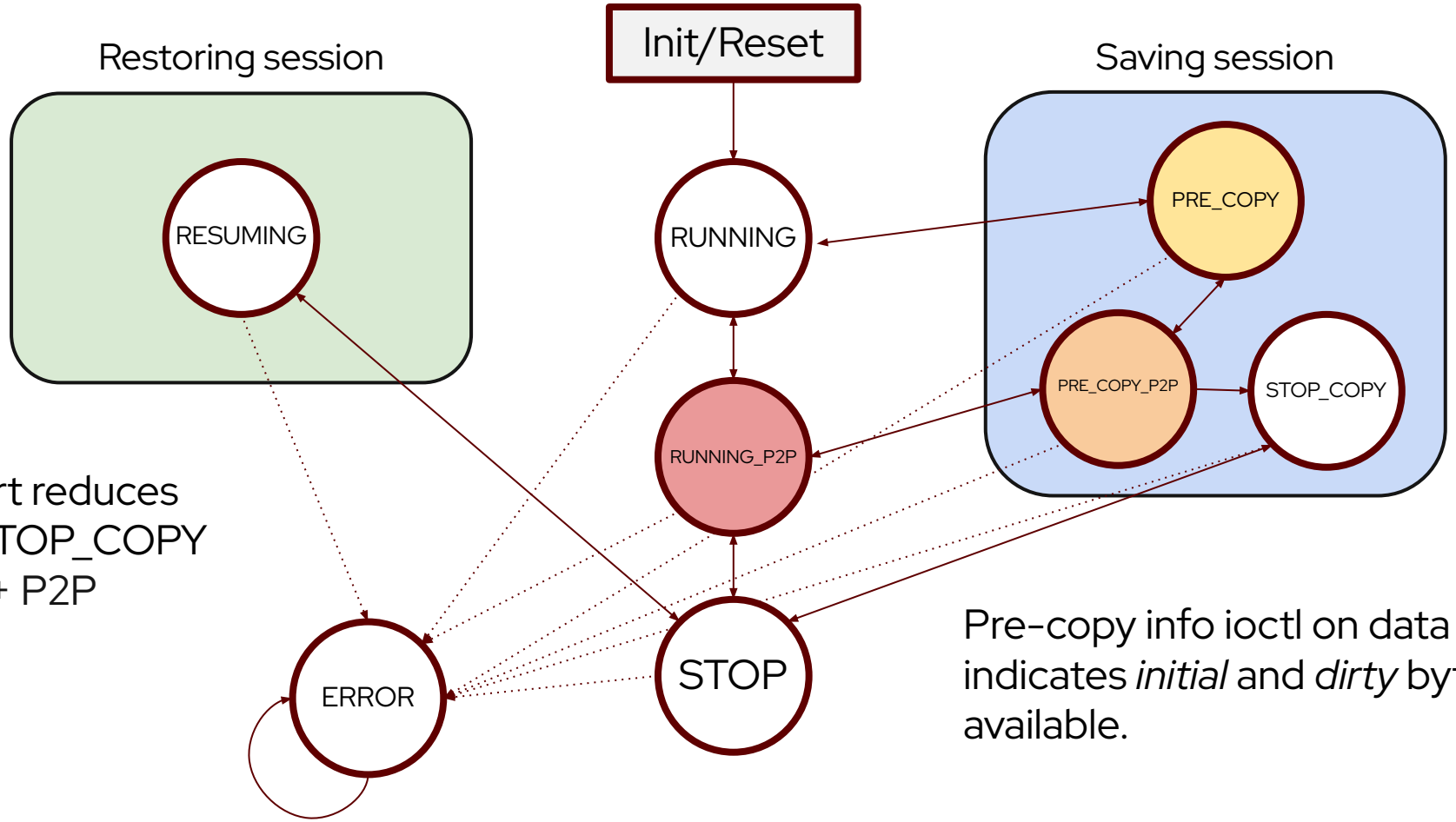


P2P (Peer-to-Peer): Device must accept incoming DMA but may not initiate.

Unsupported states are skipped.

Red Hat

VFIO Live Migration Minimal State Machine + P2P + Pre-Copy



Pre-Copy support reduces downtime during STOP_COPY

+ = Pre-Copy + P2P

Pre-copy info ioctl on data fd indicates *initial* and *dirty* bytes available.

Red Hat

VFIO Live Migration Dirty Page Tracking

- Dirty page tracking required, otherwise all memory within device AddressSpace must be assumed perpetually dirty from DMA
- Guest memory can be **rapidly** dirtied by device DMA
  - PCIe 6.0 x16: 128GB/s
- VFIO Device feature ioctls allow device-level dirty page tracking
  - Requires devel-level support exposed by variant driver
  - QEMU specifies IOVA range(s) for tracking
  - Reported as bitmap and merged with CPU dirty bitmap

VFIO Live Migration Status

- In-kernel support for:
  - NVIDIA (née Mellanox) mlx5 (v5.18)
  - HiSilicon ACC (v5.18)
  - AMD PDS (v6.6)
  - Intel QAT (v6.10)
- Out-of-tree kernel support for:
  - NVIDIA vGPU
- QEMU support for v2 migration: v8.0
  - Dirty Tracking: v8.0
  - Pre-Copy: v8.1
  - P2P: v8.2

Where does all this device specific code live?

The vfio-pci module is refactored to enable device specific functionality:

- A separate vfio-pci-core module provides default implementations
- The vfio-pci module becomes the default *variant* driver
- Device specific variant drivers provide new functionality
  - Migration: mlx5-vfio-pci, hisi-acc-vfio-pci, pds-vfio-pci, qat-vfio-pci
  - BAR manipulation: nvgrace-gpu-vfio-pci
  - Emulated virtio-legacy support: virtio-vfio-pci

Selecting a vfio-pci variant driver

```
$ grep vfio_pci: /lib/modules/`uname -r`/modules.alias

alias vfio_pci:v*d*sv*sd*bc*sc*i* vfio_pci

alias vfio_pci:v000015B3d0000101Esv*sd*bc*sc*i* mlx5_vfio_pci

alias vfio_pci:v00001DD8d00001003sv*sd*bc*sc*i* pds_vfio_pci

…
```

- `vfio_pci` alias is added for drivers
  - Denotes a vfio-pci compatible PCI driver
- Userspace picks the driver requiring the fewest wildcards
- Supported with `<hostdev managed="yes">`in libvirt since v10.0

vfio-pci variant vs mdev

- VFIO Mediated Devices (mdev) is still available
  - More aligned with “software defined” assignable devices
- mdev continues to be used for *legacy* vGPU solutions
  - Also `ccw` and `ap` devices on s390x
- vfio-pci variant drivers expected to align better where assignable device has IOMMU support
  - mdev dropped concept of an IOMMU backing device (v5.16)
- Transition from mdev entails hurdles for users
  - Different methods for configuring devices
  - No mdevctl for vfio-pci variant drivers

IOMMUFD

- Intended to provide a shared subsystem for mapping devices and memory through the IOMMU from userspace
- Support for advanced IOMMU features
  - page faults, error reporting, nested paging, etc...
- Initial use cases targeted to support VFIO and VDPA
- Intends to **replace** vfio IOMMU backends, ex. vfio-iommu-type1
  - Currently not accepting new functionality directly into type1
- Provides a **device** level interface vs vfio **group** level interface
  - IOMMU group constraints are still enforced
  - IOMMU group is not a fundamental object of the API

Flow comparison

- The VFIO API has a model of **containers** and **groups**, where the user sets a container for the group, thereby establishing the IOMMU context for the group, after which **devices** are made available through the group
- The IOMMUFD API has a model where **devices** are bound to an **iommufd*** instance, allowing an IO Address Space (**IOAS**) to be defined within the instance, and devices are attached to an IOAS, making the device fully accessible

* Note uppercase *IOMMUFD* describes the subsystem while lowercase *iommufd* describes an instance

Introducing the VFIO character device (*cdev*)

- New device access model for use with IOMMUFD
- Device file descriptor is directly opened by user
  - **vfio-dev** attribute in sysfs provides device file association
- Physical access is restricted until iommufd bind operation
- Runtime exclusion relative to VFIO legacy group API
- VFIO cdev support added in Linux v6.6

IOMMUFD Notable Features

- IOMMUFD can track pinned pages across multiple IOAS within the same iommufd, solving duplicate locked page accounting of type1
- Direct access to VFIO cdev + multiple IOAS per iommufd enables a QEMU model that supports passing file descriptors via SCM_RIGHTS
- IOMMUFD provides a VFIO compatibility mode by linking device files
  - Intended to ease removal of VFIO IOMMU backends
- Provides an interface for IOMMU-based dirty page tracking
  - Ubiquitous support where available in system IOMMU hardware

IOMMUFD Status

- IOMMUFD will **eventually** replace VFIO IOMMU backends
  - Deprecation process has not officially begun
  - Compatibility interfaces should make this transparent
  - IOMMUFD currently has a temporary feature gap for DMA mapping device memory, ie. no peer-to-peer DMA support
- Kernel support added in v6.2
- QEMU support added in v9.0
  - New `iommufd` object, vfio-pci device `iommufd=` parameter
- libvirt support in progress

IOMMUFD Status

IOMMUFD Features

Feature	Version	
IO Page Table dirty tracking	v6.7	Merged
User IO Page Table	v6.7	Merged
User IO Page Table Invalidation	V6.8	Merged
Fault delivery to user space	V6.11	Merged
PASID Support	v4	In Progress
VIOMMU Kernel Support	v2	In Progress
IOMMU_IOAS_CHANGE_PROCESS	RFC	In Progress
Memfd/guestmemfd backing store	N/A	In Progress
Consolidated Page Table	RFC	In Progress
SIOV Support	RFC	In Progress
VDPA Integration	RFC	In Progress
Confidential Compute TDISP	N/A	
ARM ITS Direct routing	N/A	
~~Share KVM page table with IOMMU~~	~~N/A~~	

LINUX PLUMBERS CONFERENCE Vienna, Austria / Sept. 18-20, 2024

https://lpc.events/event/18/contributions/1789/attachments/1460/3100/LPC2024_iommufd.pdf

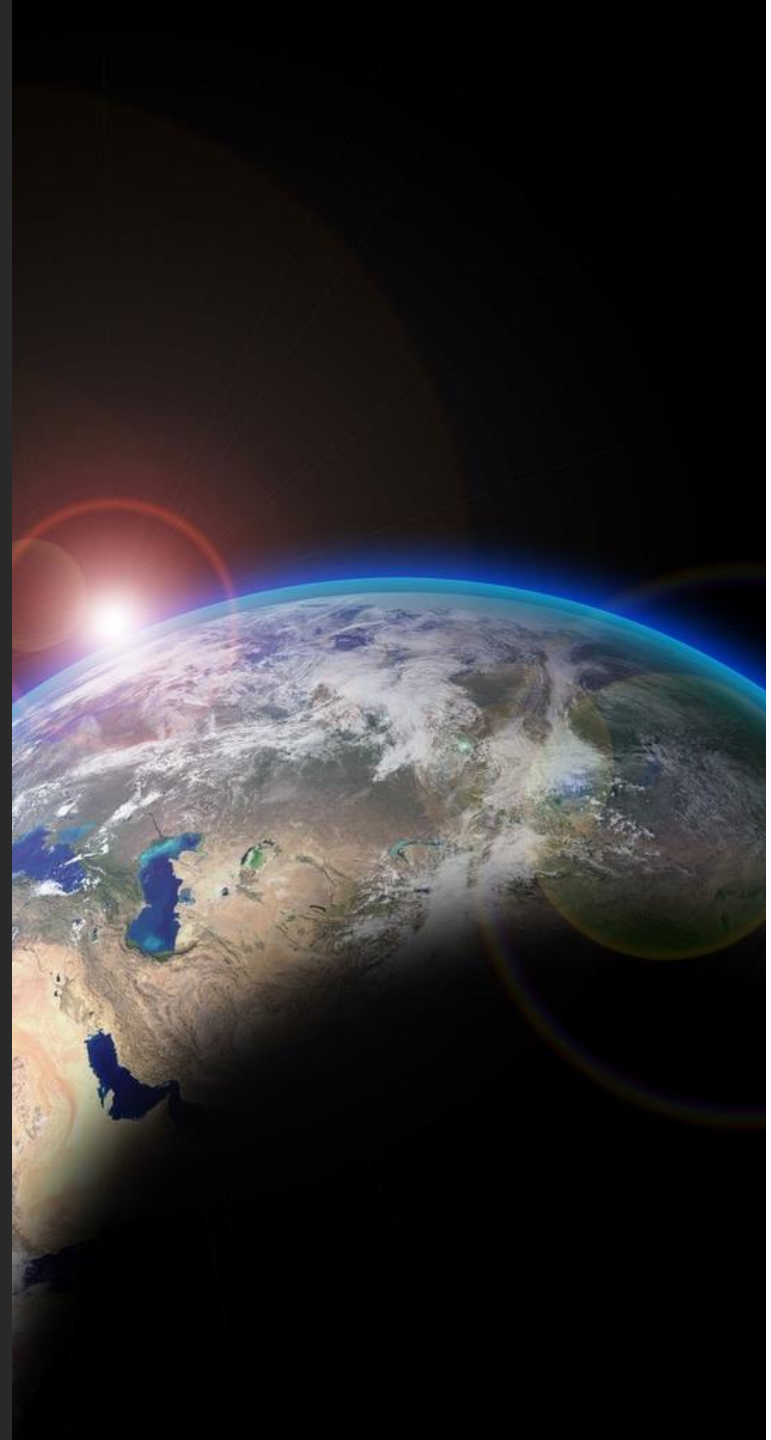
Red Hat

Ongoing work...

- Building PCI config space in the VMM
- Exporting MMIO via dma-buf (also for IOMMUFD P2P mappings?)
- QEMU multi-fd migration
- IOMMUFD nested page tables, generic page tables, faults to userspace, process address spaces, ...

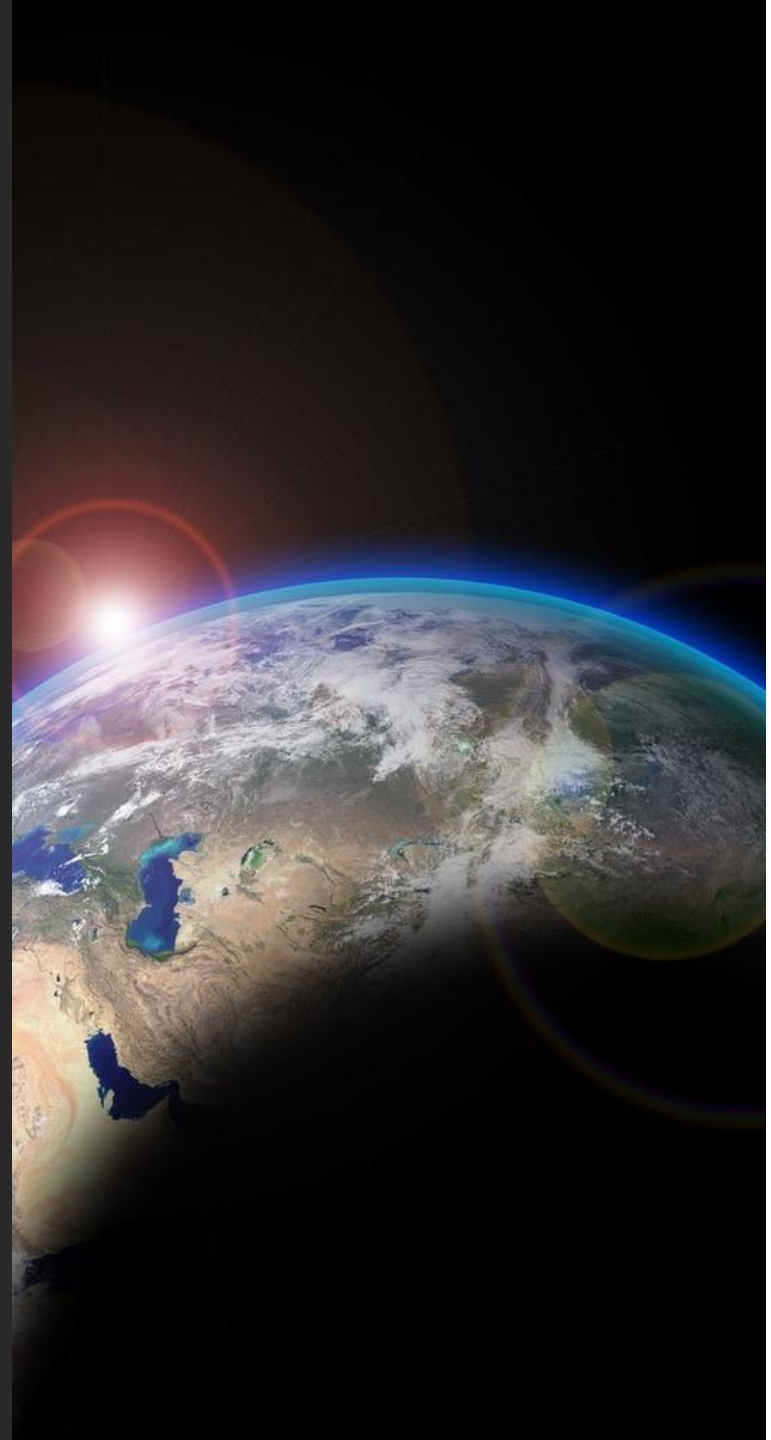
And a new recruit!

- Cédric Le Goater is now the primary QEMU VFIO maintainer



Questions?

Red Hat



Thanks!

Red Hat