vDPA Live Migration Downtime Optimizations for VirtIO Net Devices

NetDevConf 0x18

Dragoș Tătulea <dtatulea@nvidia.com> NVIDIA

Eugenio Pérez Martín <eperezma@redhat.com> Red Hat

Si-Wei Liu <si-wei.liu@oracle.com> Oracle

vDPA Live Migration Downtime improvements for net devices

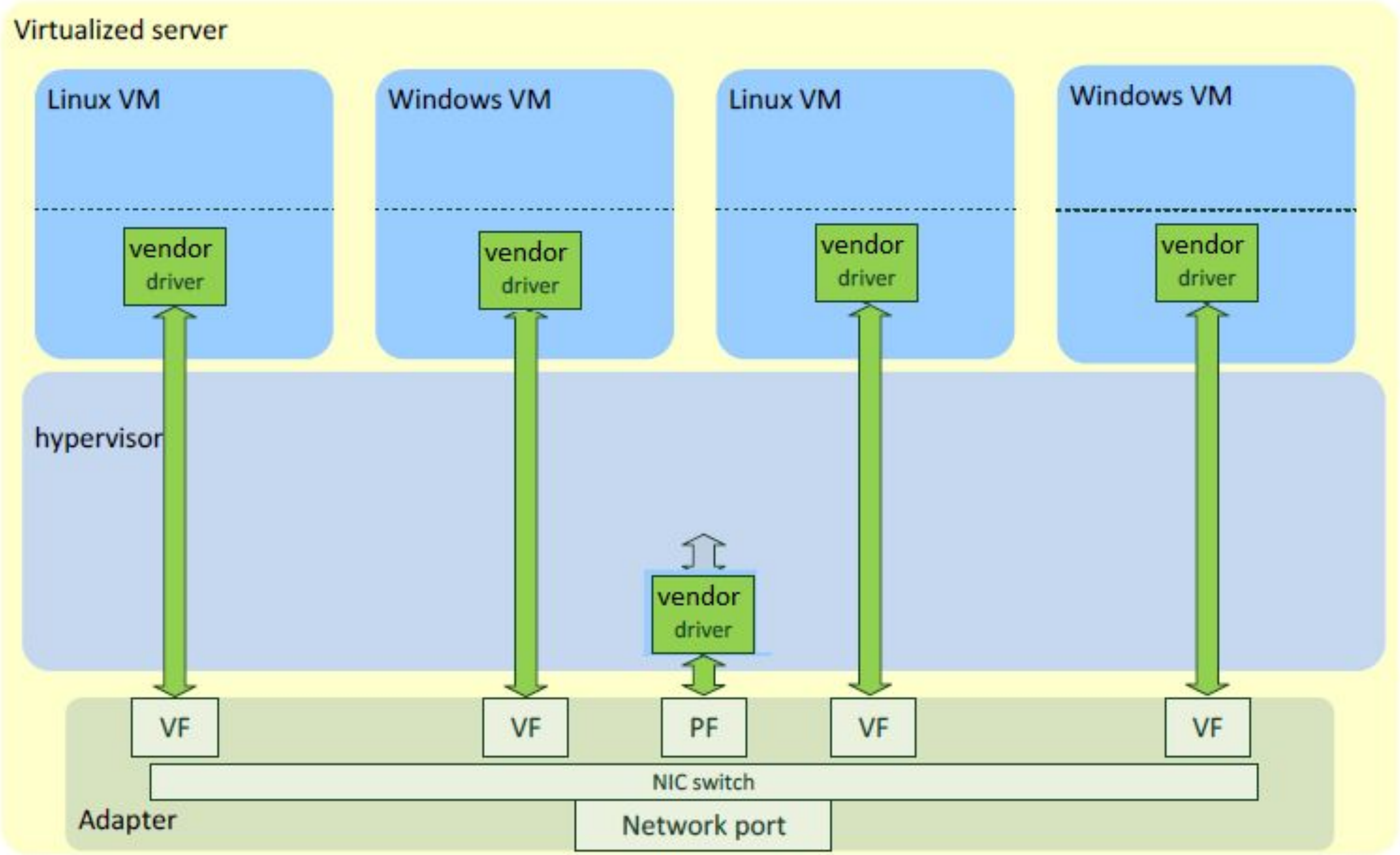
Eugenio Pérez Martín
Sr. Software Engineer <eperezma@redhat.com>

Red Hat

Agenda

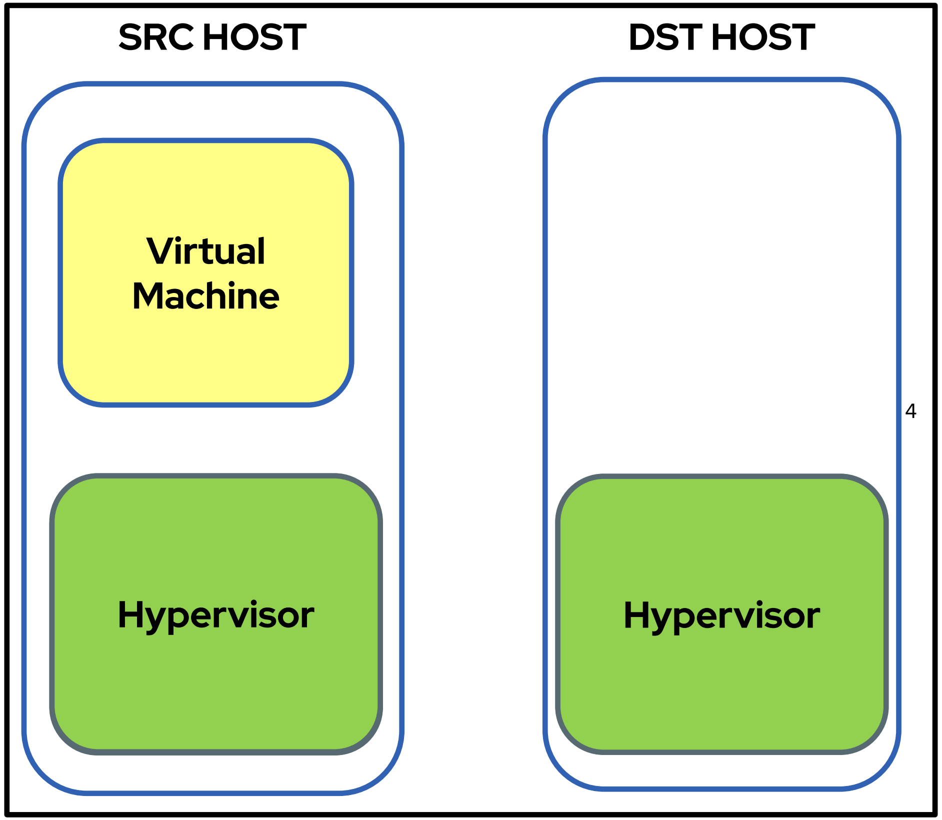
- Basic concepts
    - SR-IOV
    - Live Migration
- Problem: LM with passthrough VF
- Solution: virtio vDPA
- Cross-vendor VM Live Migration Demo
- Shadow virtqueue

Vendor passthrough / SR-IOV



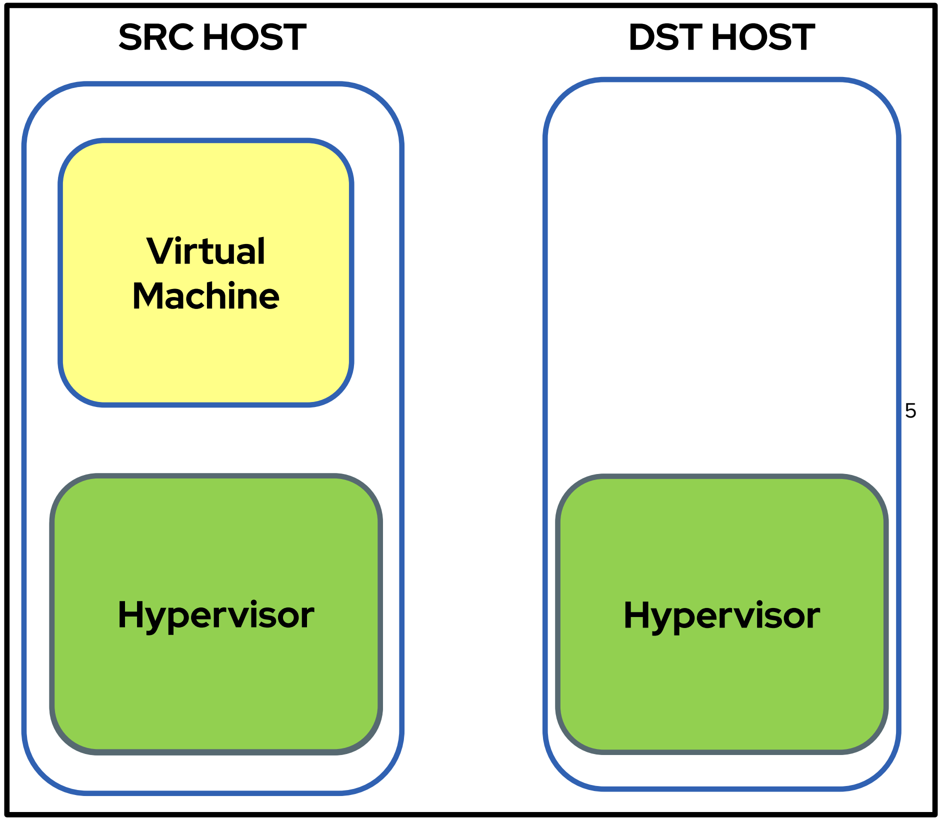
VM Live Migration

- What is Live Migration?
  - Process of moving a VM running on one physical host to another while the guest OS is **running**
  - The guest shouldn't realize the world is changing beneath its feet
  - Useful for load balancing, hardware / software maintenance etc.



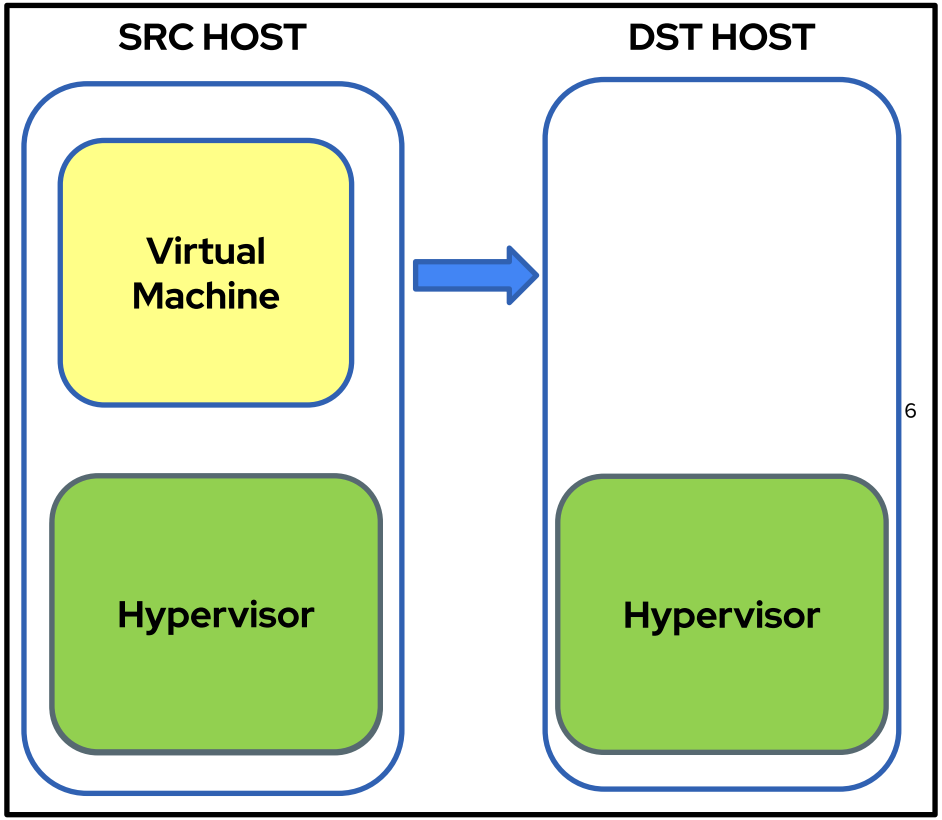
VM Live Migration

- What is Live Migration?
  - Process of moving a VM running on one physical host to another while the guest OS is **running**
  - The guest shouldn't realize the world is changing beneath its feet
  - Useful for load balancing, hardware / software maintenance etc.
- How does it happen?
  - Mark unsent (or modified) RAM as **dirty**



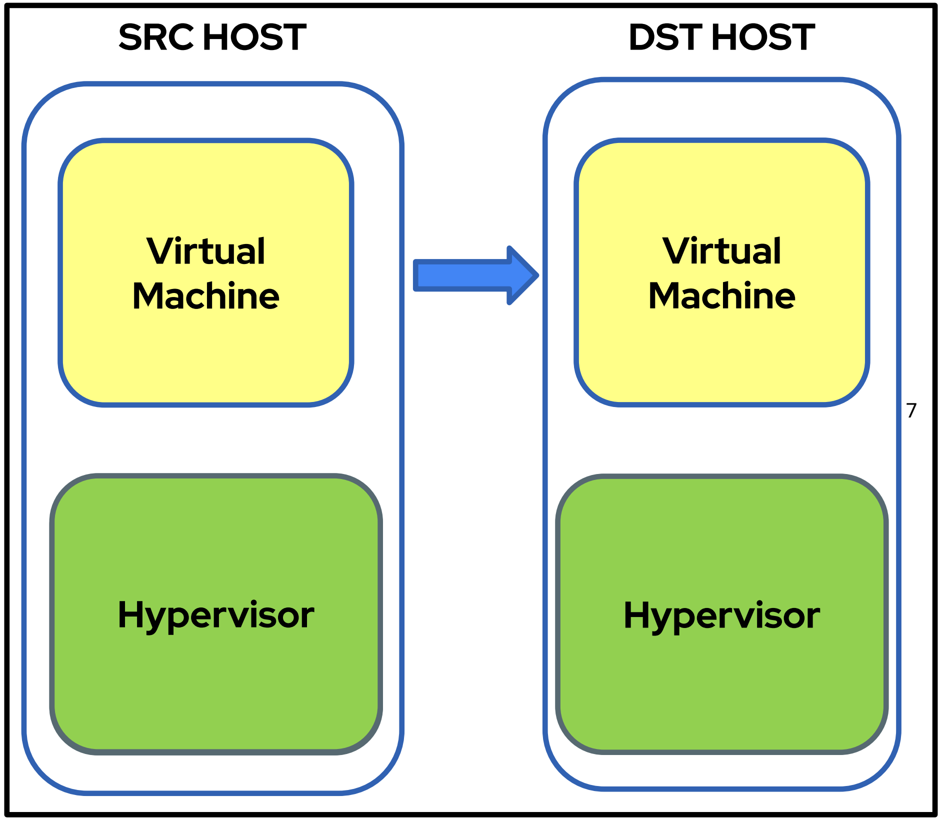
VM Live Migration

- What is Live Migration?
  - Process of moving a VM running on one physical host to another while the guest OS is **running**
  - The guest shouldn't realize the world is changing beneath its feet
  - Useful for load balancing, hardware / software maintenance etc.
- How does it happen?
  - Mark unsent (or modified) RAM as **dirty**
  - Send RAM content to the destination until a **threshold** is reached.



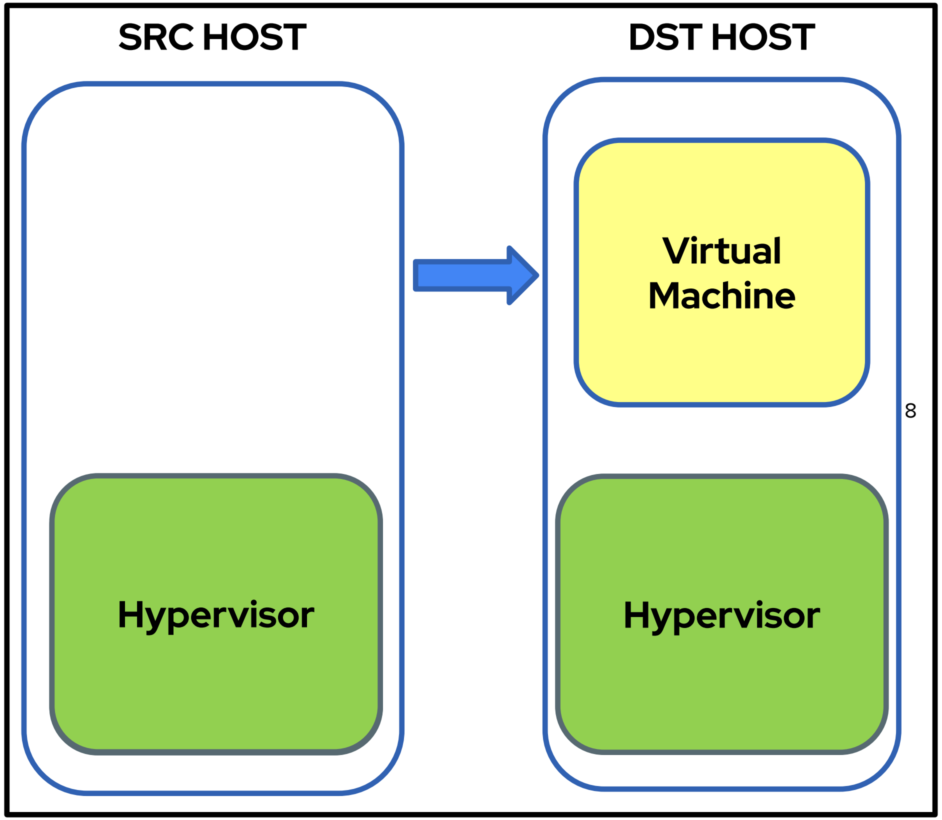
VM Live Migration

- What is Live Migration?
  - Process of moving a VM running on one physical host to another while the guest OS is **running**
  - The guest shouldn't realize the world is changing beneath its feet
  - Useful for load balancing, hardware / software maintenance etc.
- How does it happen?
  - Mark unsent (or modified) RAM as **dirty**
  - Send RAM content to the destination until a **threshold** is reached.
  - Stop guest, transfer **remaining** dirty RAM, device state



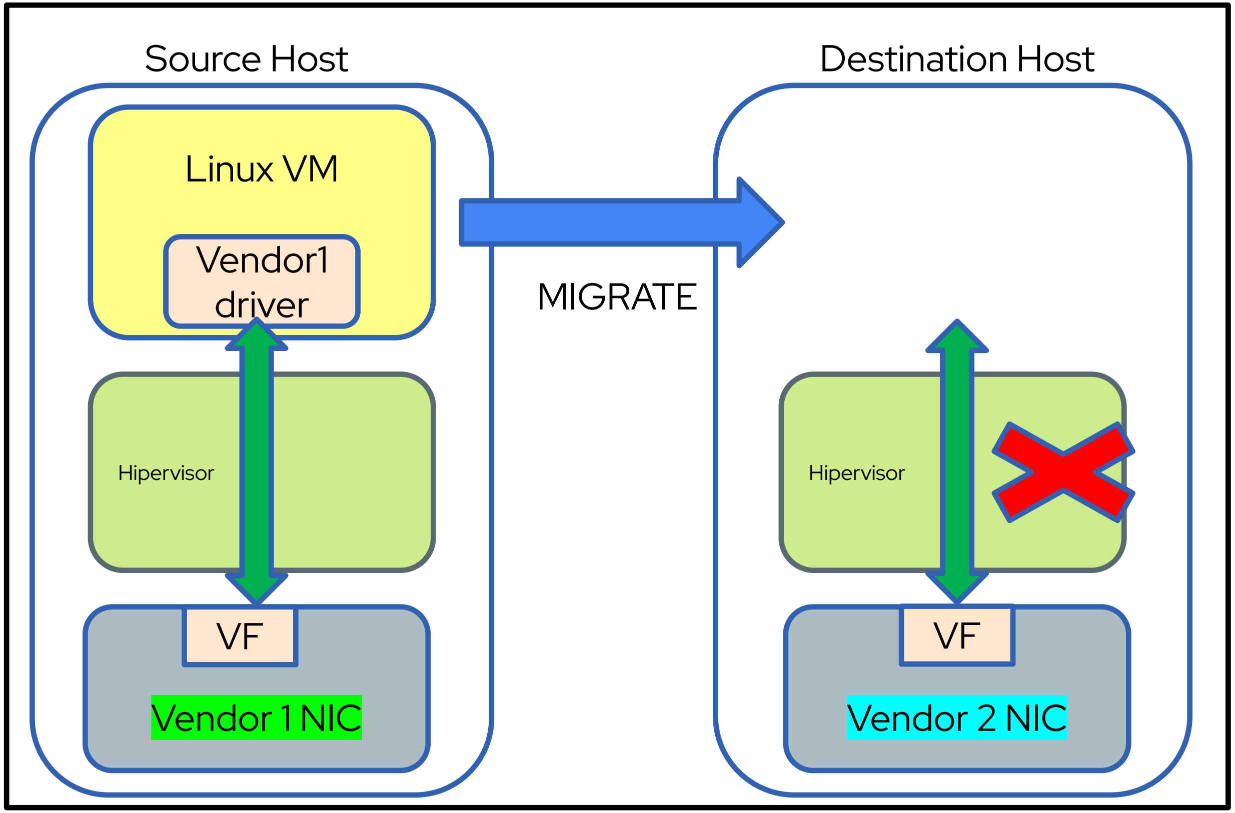
VM Live Migration

- What is Live Migration?
  - Process of moving a VM running on one physical host to another while the guest OS is **running**
  - The guest shouldn't realize the world is changing beneath its feet
  - Useful for load balancing, hardware / software maintenance etc.
- How does it happen?
  - Mark unsent (or modified) RAM as **dirty**
  - Send RAM content to the destination until a **threshold** is reached.
  - Stop guest, transfer **remaining** dirty RAM, device state
  - **Resume** execution on destination



Live Migration: SR-IOV VF Passthrough

- **Requires identical NIC HW** on both source and destination host
  - Tight coupling between the Guest SW and Host HW
  - Vendor's VF driver required in the Guest OS

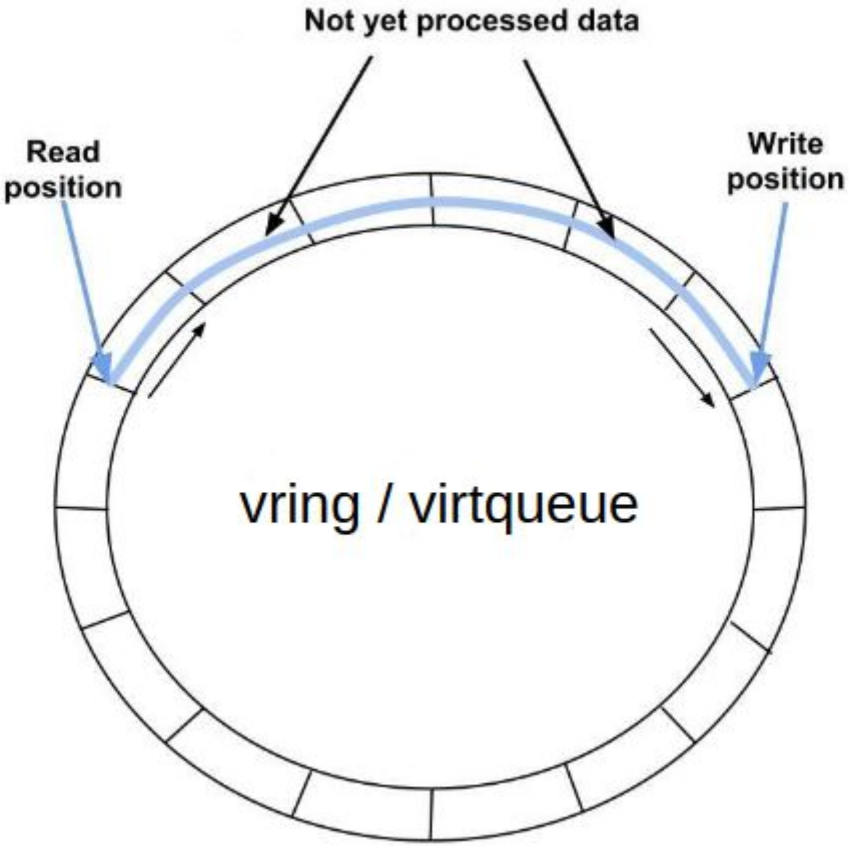


Virtual I/O Device (VIRTIO)

- **Virtio** is a specification that describes virtual devices, drivers and how they interact.

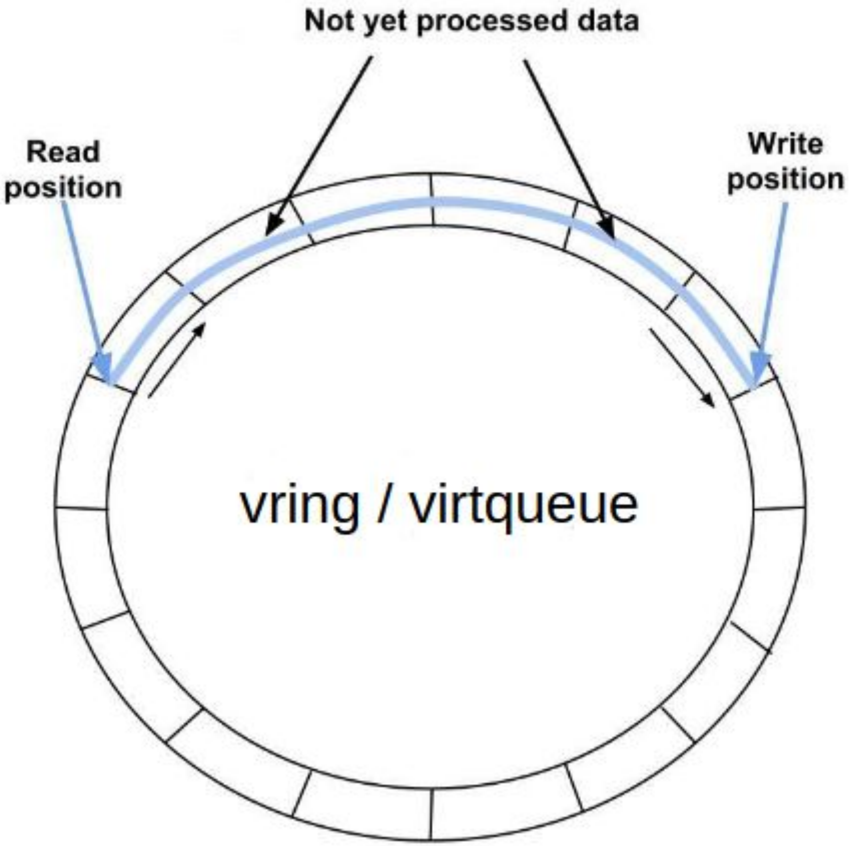
Virtual I/O Device (VIRTIO)

- **Virtio** is a specification that describes virtual devices, drivers and how they interact.
  - Data plane
    - **Virtqueues**, implemented with vrings: ring of buffers descriptors
    - Transfers the actual data

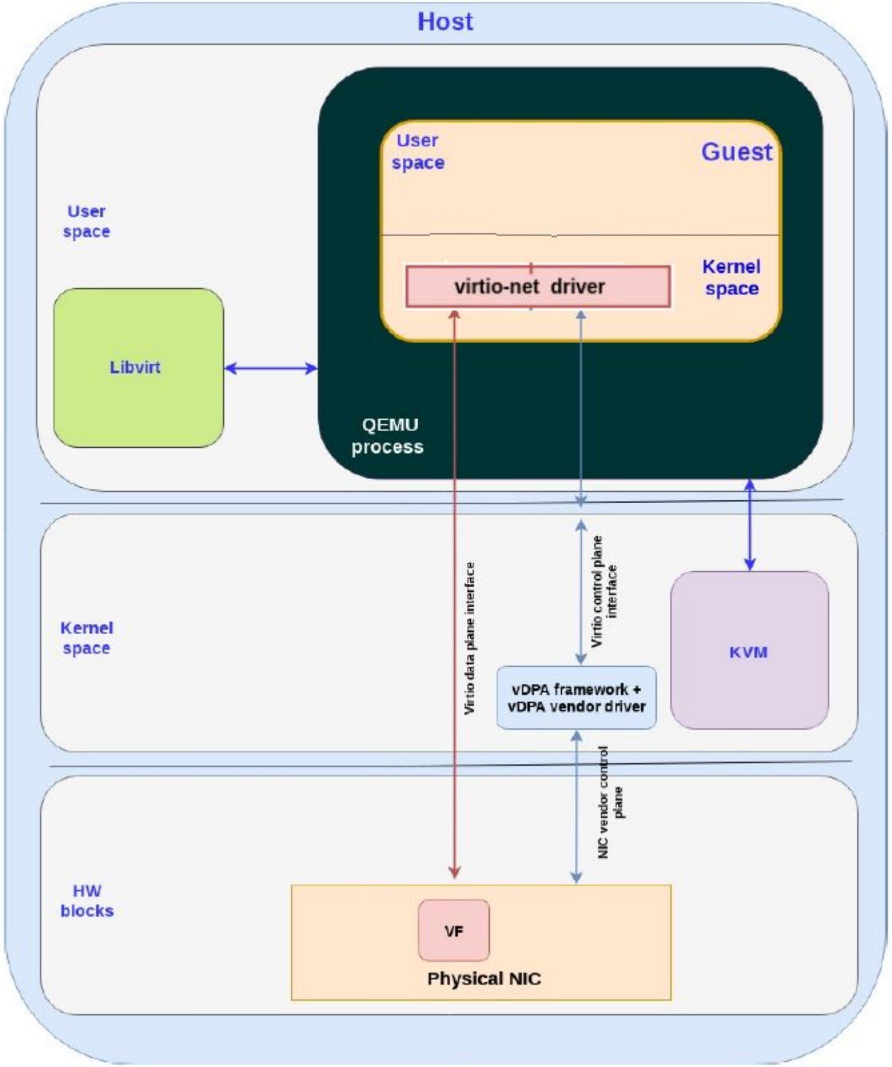


Virtual I/O Device (VIRTIO)

- **Virtio** is a specification that describes virtual devices, drivers and how they interact.
  - Data plane
    - **Virtqueues**, implemented with vrings: ring of buffers descriptors
    - Transfers the actual data
  - Control plane
    - Manages the data plane
    - Feature negotiation, shared memory configuration...

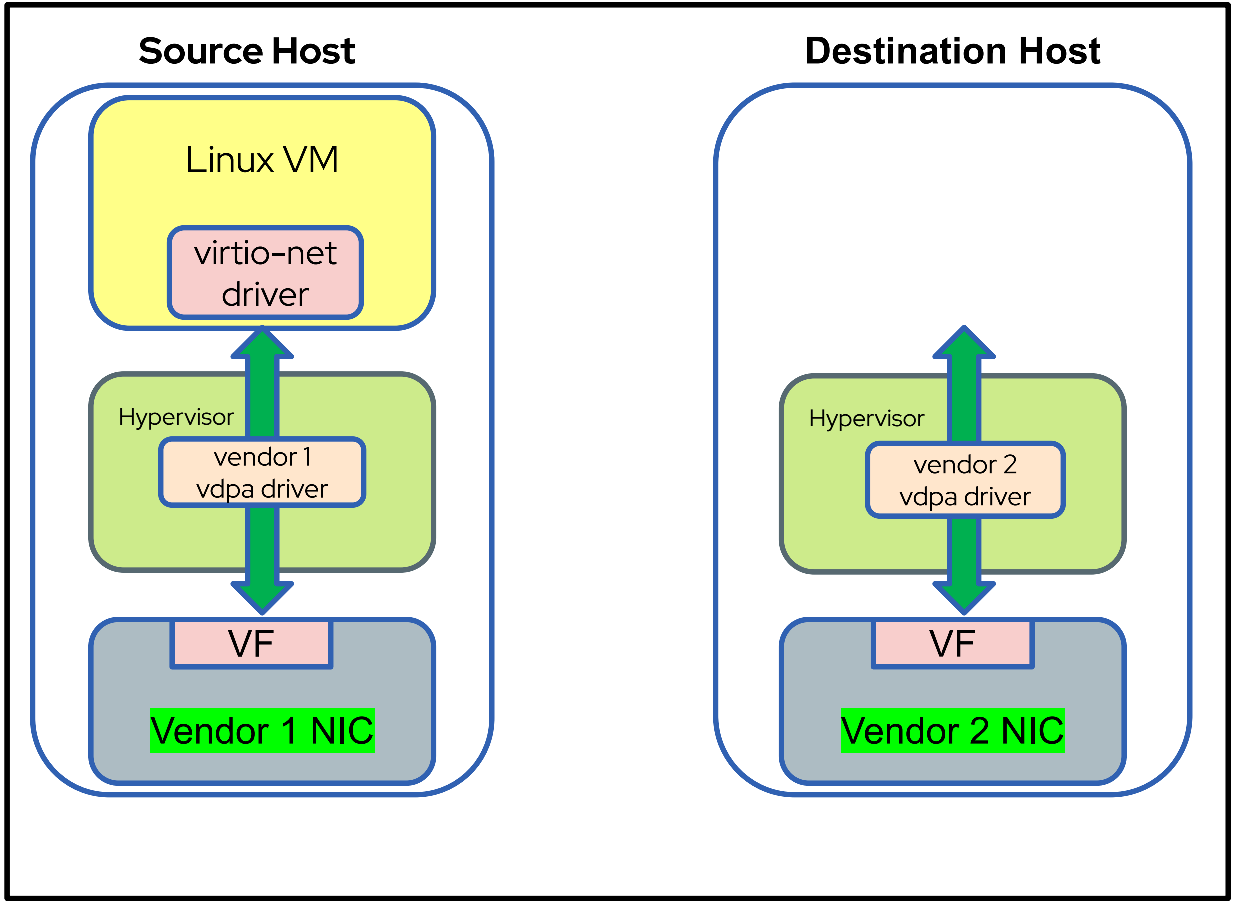


vDPA


Host
User space
Guest
User space
virtio-net driver
Kernel space
Libvirt
QEMU process
Kernel space
Virtio data plane interface
Virtio control plane interface
vDPA framework + vDPA vendor driver
KVM
NIC vendor control plane
HW blocks
VF
Physical NIC

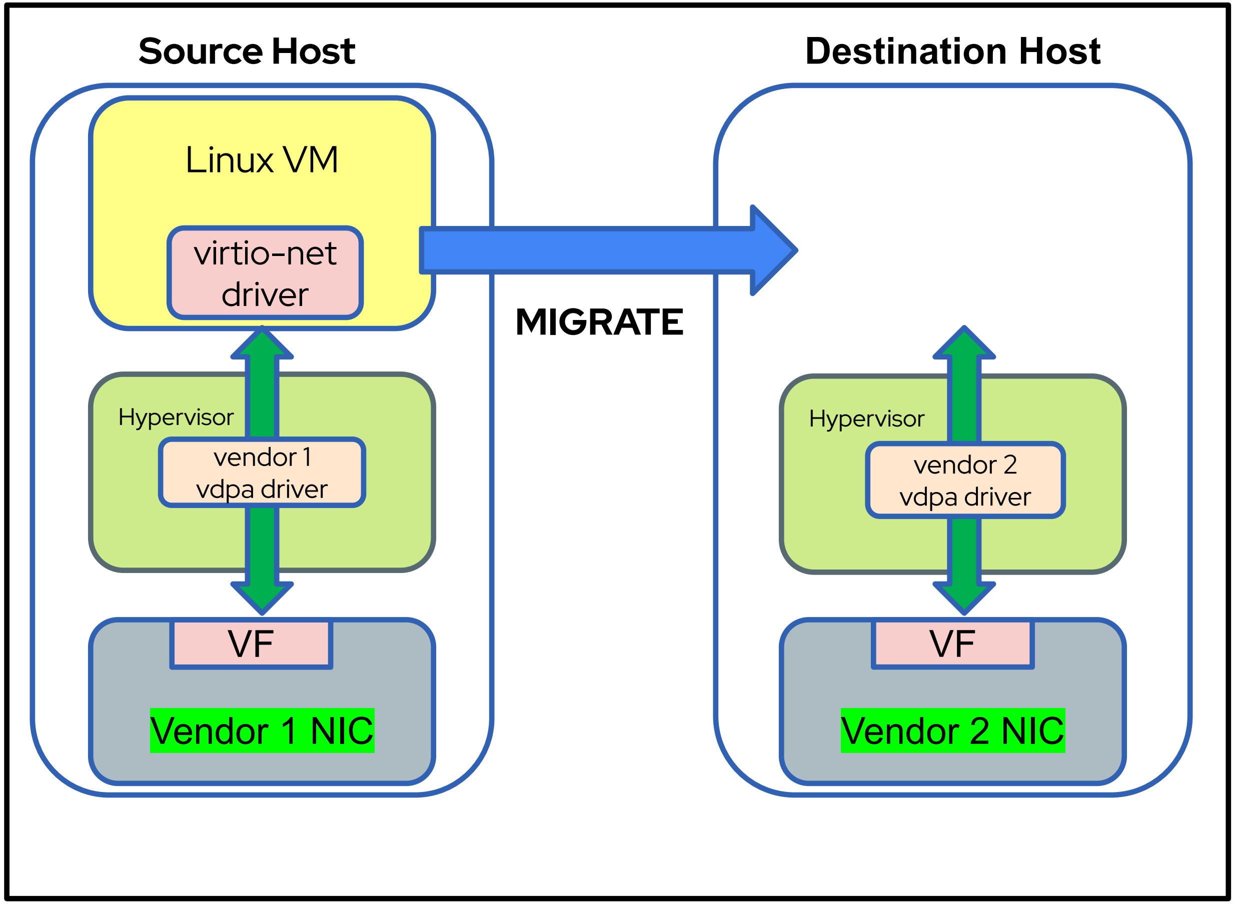
Live Migration with vDPA

- Live migration is **transparent**
  - Guest always talk with virtio-net device, irrespective of actual vendor HW
  - Hypervisor **doesn't require guest's collaboration**.



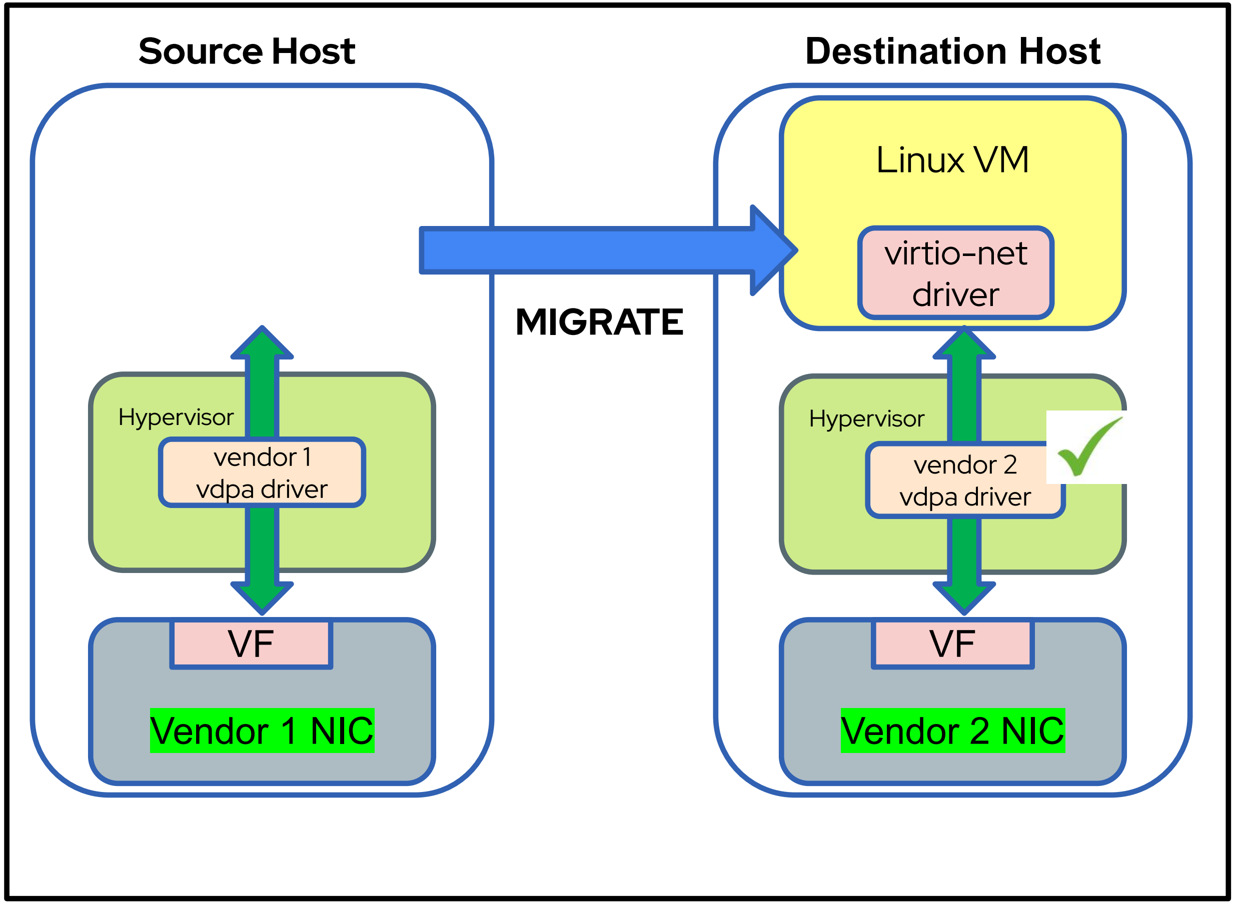
Live Migration with vDPA

- Live migration is **transparent**
  - Guest always talk with virtio-net device, irrespective of actual vendor HW
  - Hypervisor **doesn't require guest's collaboration**.



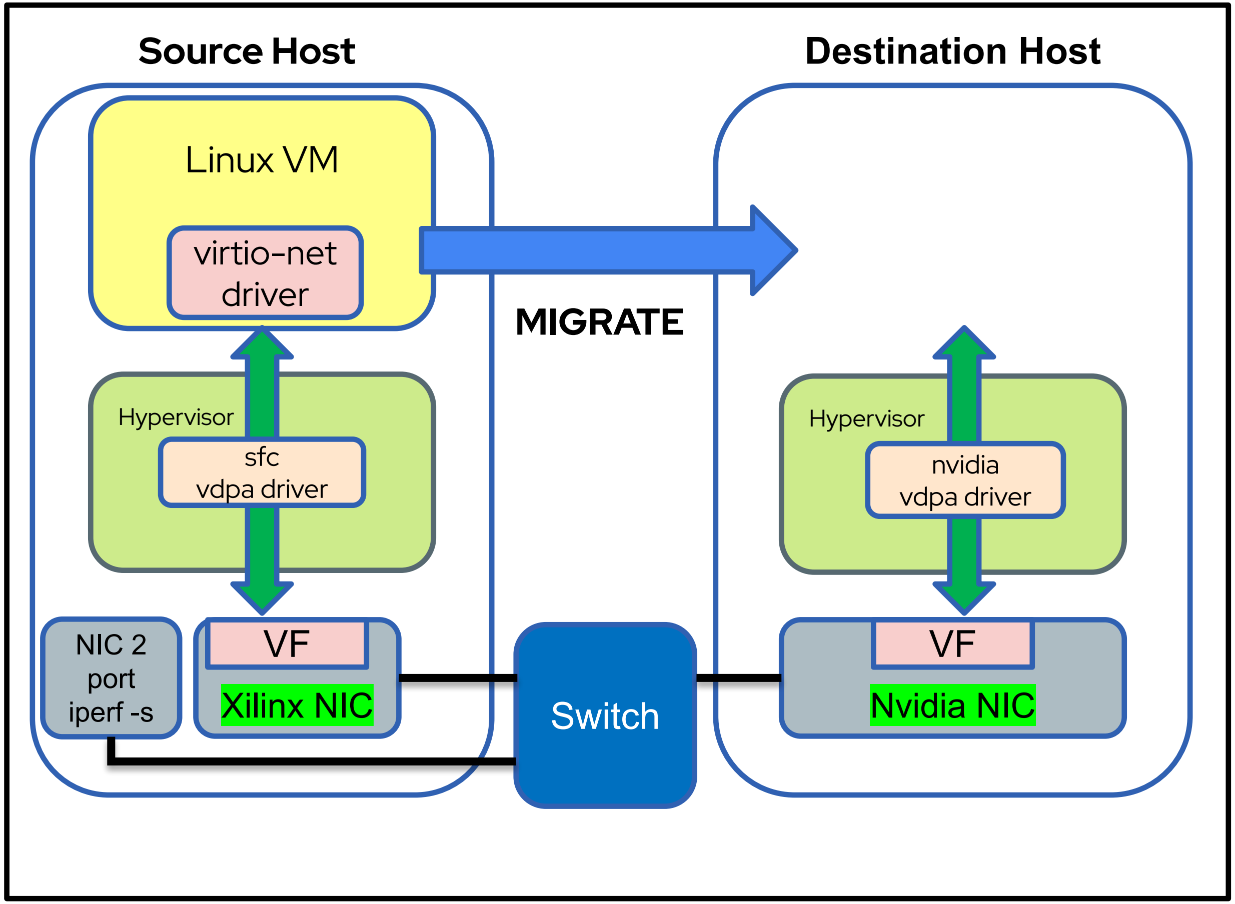
Live Migration with vDPA

- Live migration is **transparent**
  - Guest always talk with virtio-net device, irrespective of actual vendor HW
  - Hypervisor **doesn't require guest's collaboration**.



Demo scenario

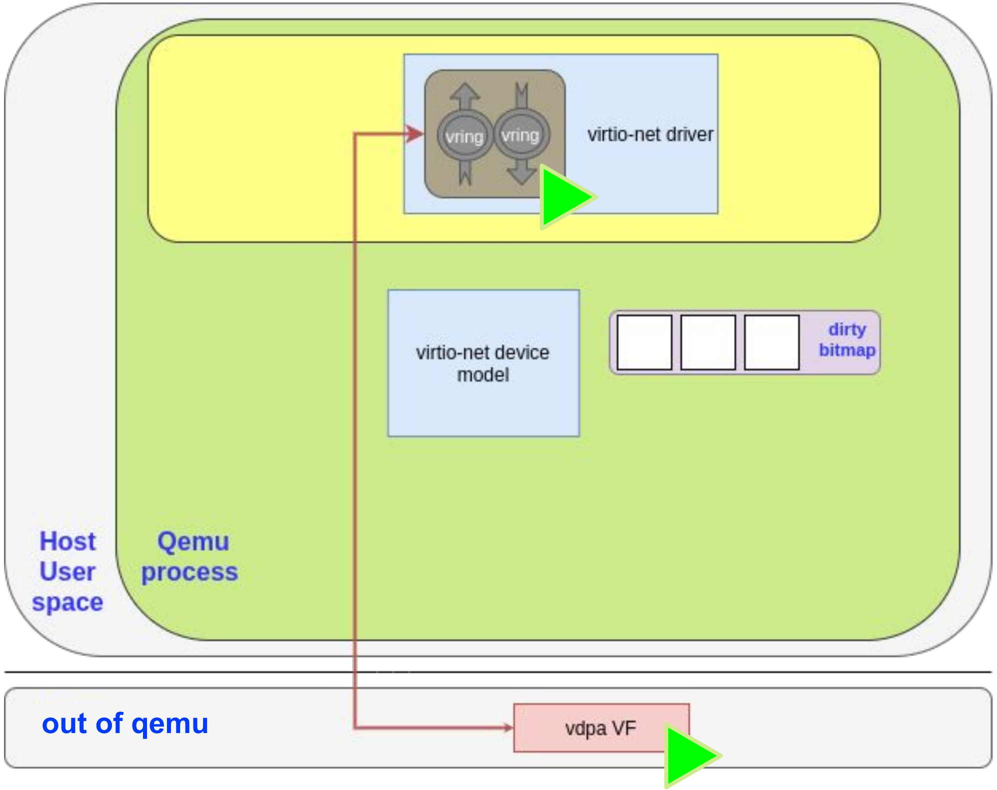
- Source host (dell750-28) has two interesting NICs
  - **AMD Xilinx SN1022**
  - **Mellanox ConnectX 6 (running iperf server)**
- Destination host (dell750-23) has single interesting NIC
  - **Nvidia ConnectX 6**
- These NIC ports are connected via Switch



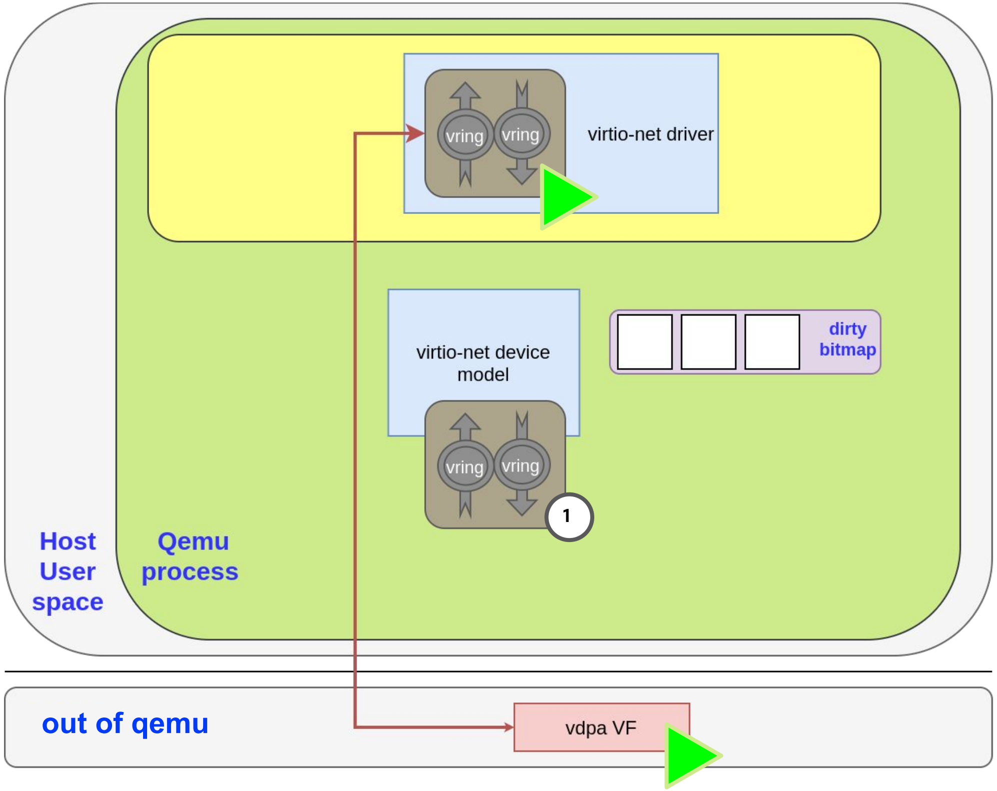
DEMO

https://www.youtube.com/watch?v=ocpwyiBkBBc

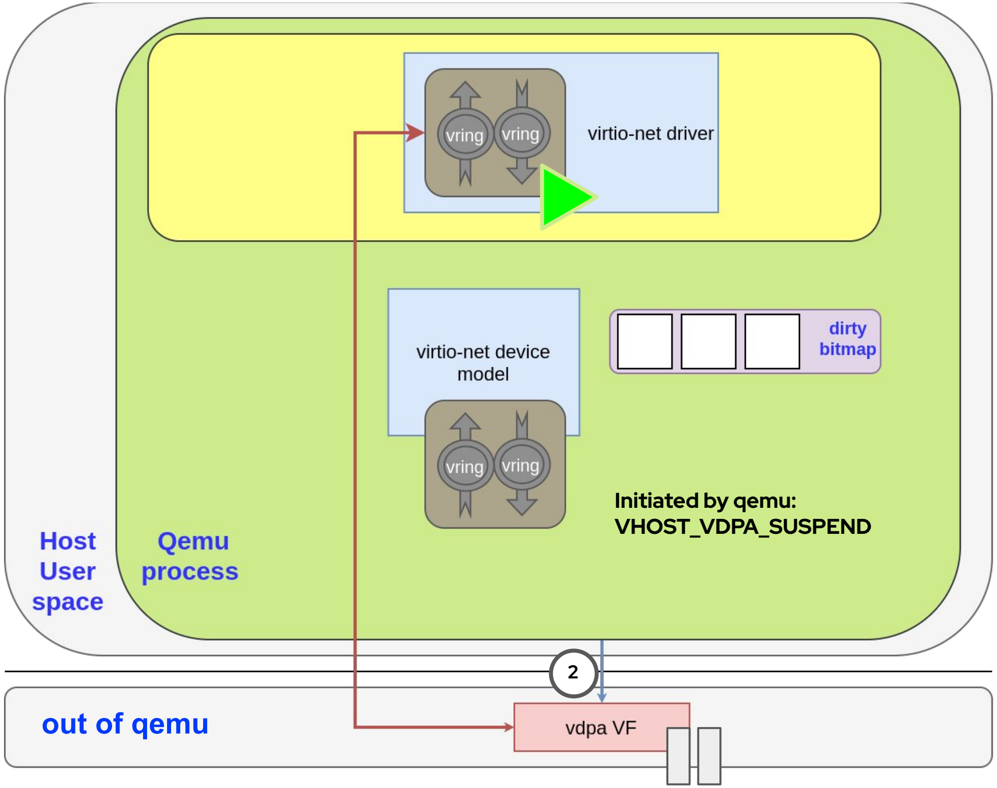
Shadow virtqueue: Regular operation



Shadow virtqueue: Regular operation

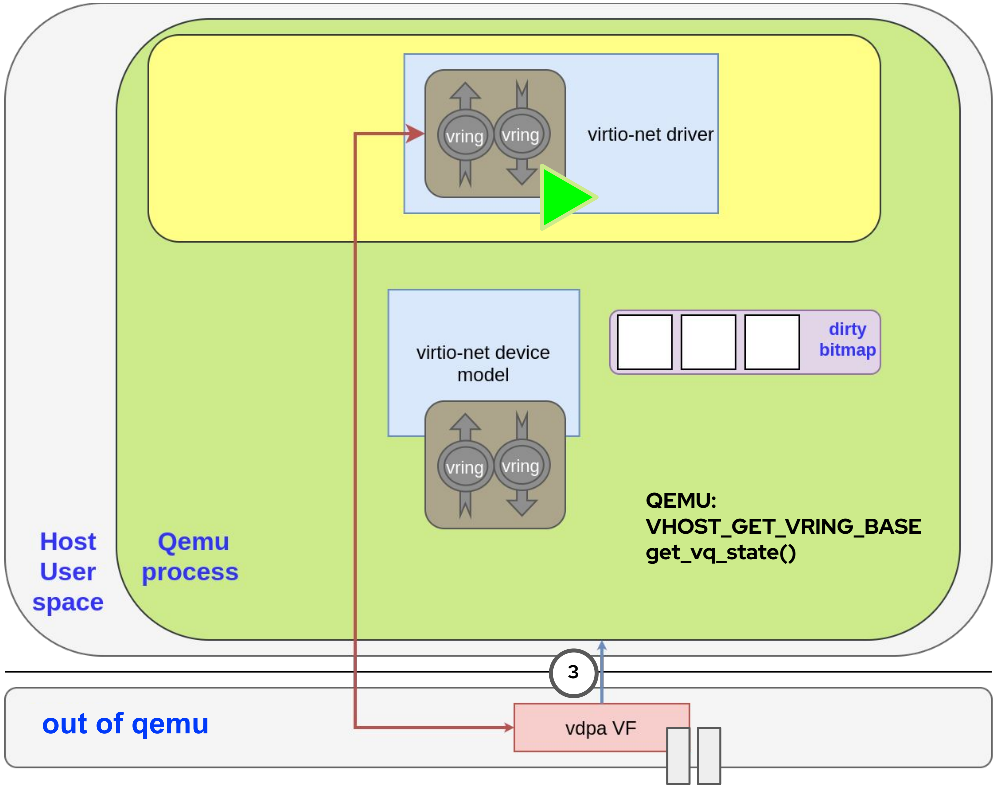


Shadow virtqueue: Regular operation

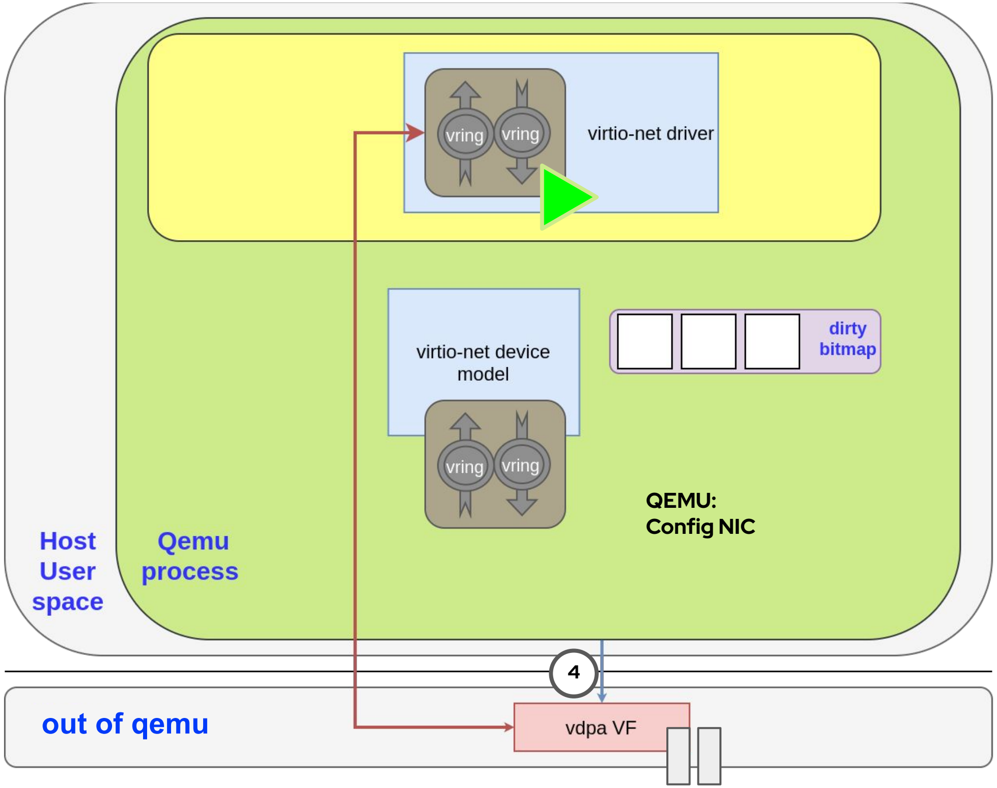


Initiated by qemu:
VHOST_VDPA_SUSPEND

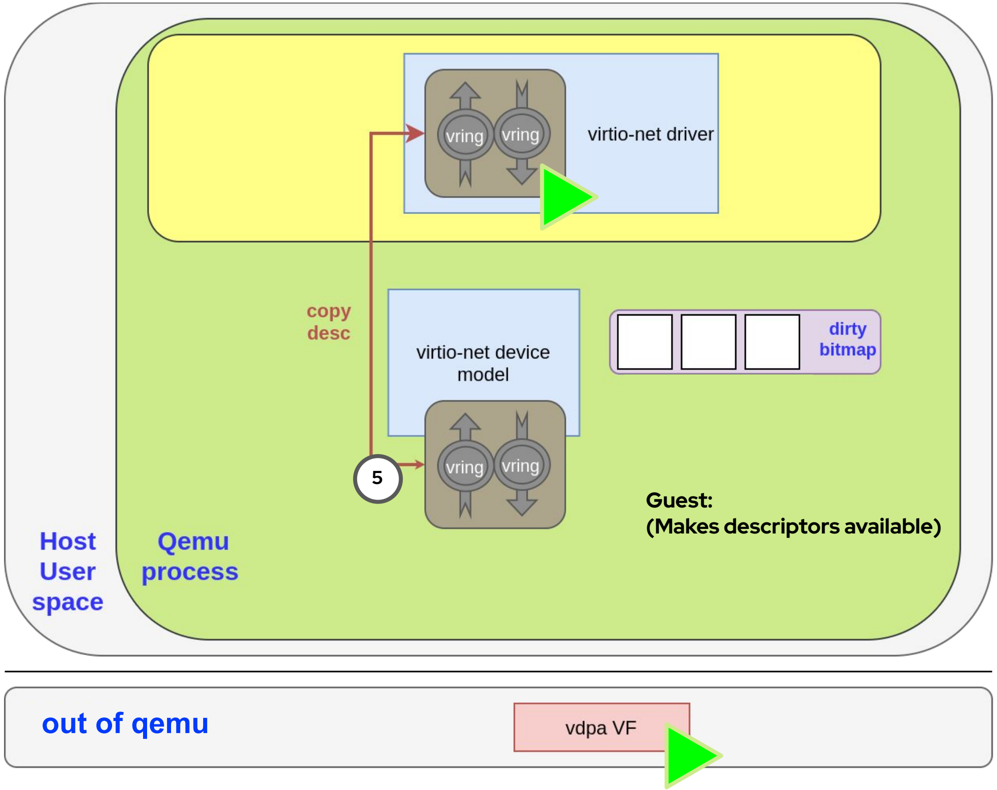
Shadow virtqueue: Regular operation



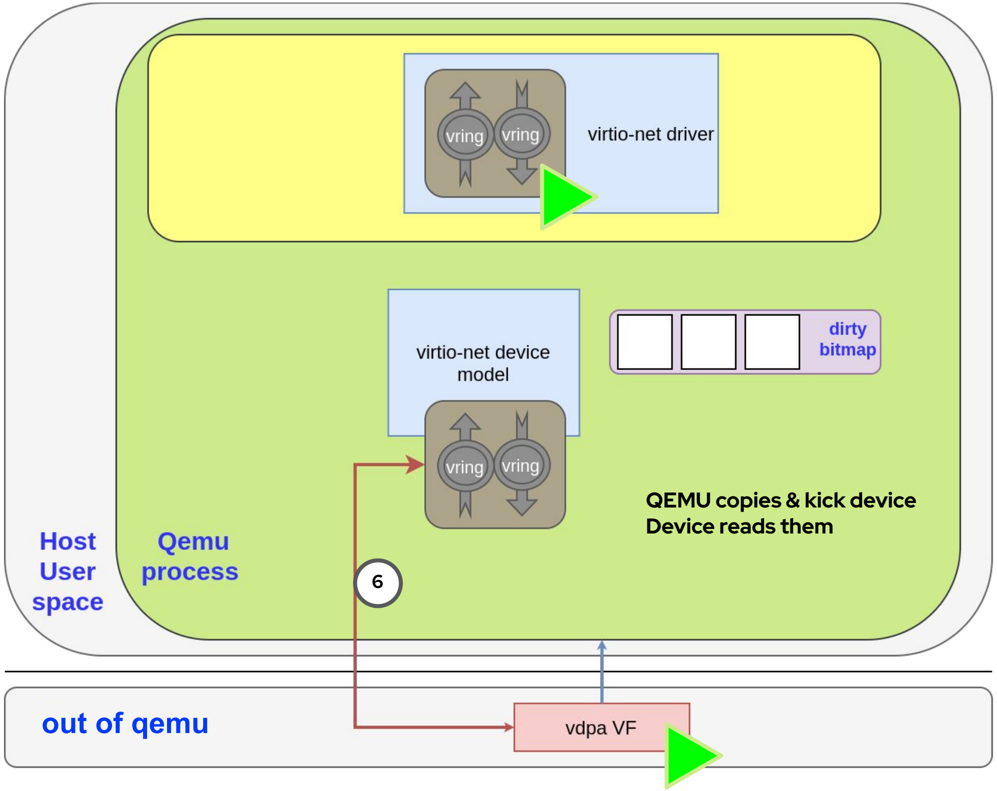
Shadow virtqueue: Regular operation



Shadow virtqueue: Regular operation

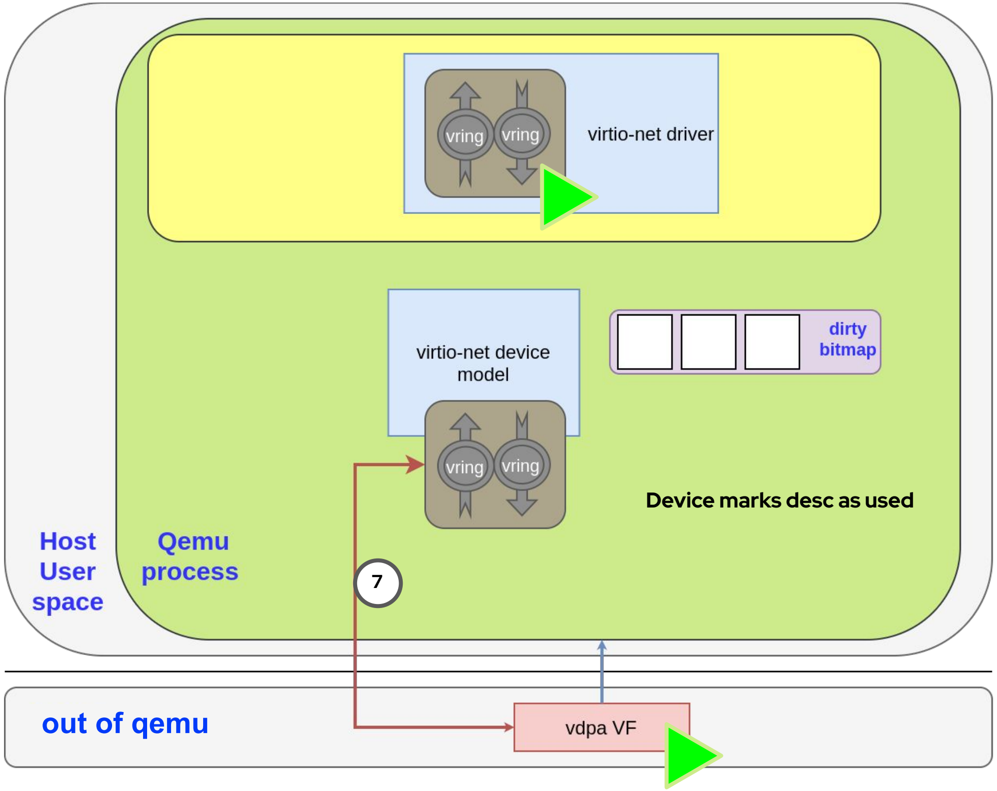


Shadow virtqueue: Regular operation

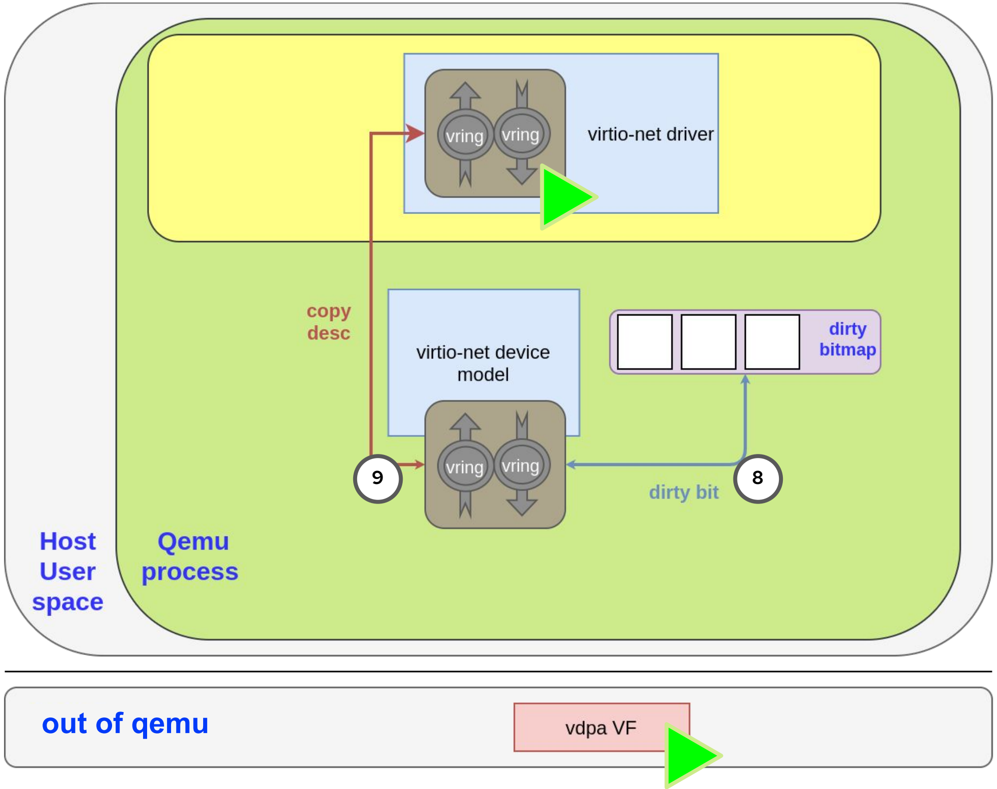


QEMU copies & kick device
Device reads them

Shadow virtqueue: Regular operation



Shadow virtqueue: Regular operation



ORACLE

vDPA Live Migration Downtime improvements for net devices

Netdev 0x18

Si-Wei Liu

<si-wei.liu@oracle.com>

Oracle Corporation

Extend vDPA Infra to Cloud Scale

- Scale
  - Hundred of Virtual Functions per card - VM use case
  - Thousand of Scalable Functions per card - container use case
  - Could support VM with high density of vDPA vNICs
  - VM could go up to a couple of TBs in memory size and 100+ of vCPU cores

- Performance
  - Should exceed para-virtualized vhost-kernel backend
  - Should be comparable to SR-IOV passthrough: H/W offload required
  - Micro-benchmarks: bandwidth, packet rate, latency, host cpu utilization

- Live Migration & Hypervisor (QEMU) Live Update
  - Should keep 50% - 70% of I/O performance during live migration
  - Target sub-second latency (a few hundred milliseconds of blackout time) per VM
  - Goal is to have per-device teardown & startup cost to be < 100 milliseconds!

vDPA Live Migration - Overview of Challenges

- vDPA hardware device assisted dirty tracking?
  - Hardware resource constraints: scalability bottleneck
  - Highly contentious with host vCPU dirtying thread
  - No intrinsic throttling on DMA, indirect throttling via hypervisor software as mitigation
  - Performance optimization could be complex and vendor device dependent

- IOMMUFD dirty tracking
  - IOMMU dirty tracking only available in newer platforms

- Safe Harbor: Software Mediation via Shadow VirtQueue (SVQ)
  - Implementation originated from software virtio-based backend
  - Slow on real hardware device, could use some improvements
  - Profiling on hardware backend: most costly part is on memory pinning (and mapping)
  - Varied sources of latency on hardware device startup or teardown affecting downtime
  - Some are vendor device specific: on-chip iommu mapping, virtqueue creation and setup
  - Some are generally related to virtio spec conformance or vhost(-vdpa) plumbing

Shadow VQ - Performance Potentials

- Move hardware slow path out of downtime!
  - Device reset (slow) -> Suspend and Resume (relatively fast)
  - SVQ translation cost -> Dedicated address space for SVQ descriptors
  - DRIVER_OK setup cost at dest -> Move it ahead to device initialization? Iterative migration?
  - uAPI: vhost-vdpa backend features
  - Multiple CVQ cmd ioctls to restore device state -> Batch and streamline with io_uring?
  - Multiple vhost-vdpa devices -> Parallelize migration with multi-threaded per-device teardown
  - Participation and feedback from hardware vendors are more than welcomed!

- Further improve Shadow VQ datapath performance
  - only forwards descriptor metadata rather than copy over memory buffers
  - bandwidth throughput: could use multi-threaded SVQ
  - lower down PCIe transaction and cache utilization: packed ring

- Could be used to emulate other ring layout using virtio v1.0 spec compliant device
  - legacy v0.9.5 device emulation due to lack of IOMMU platform feature

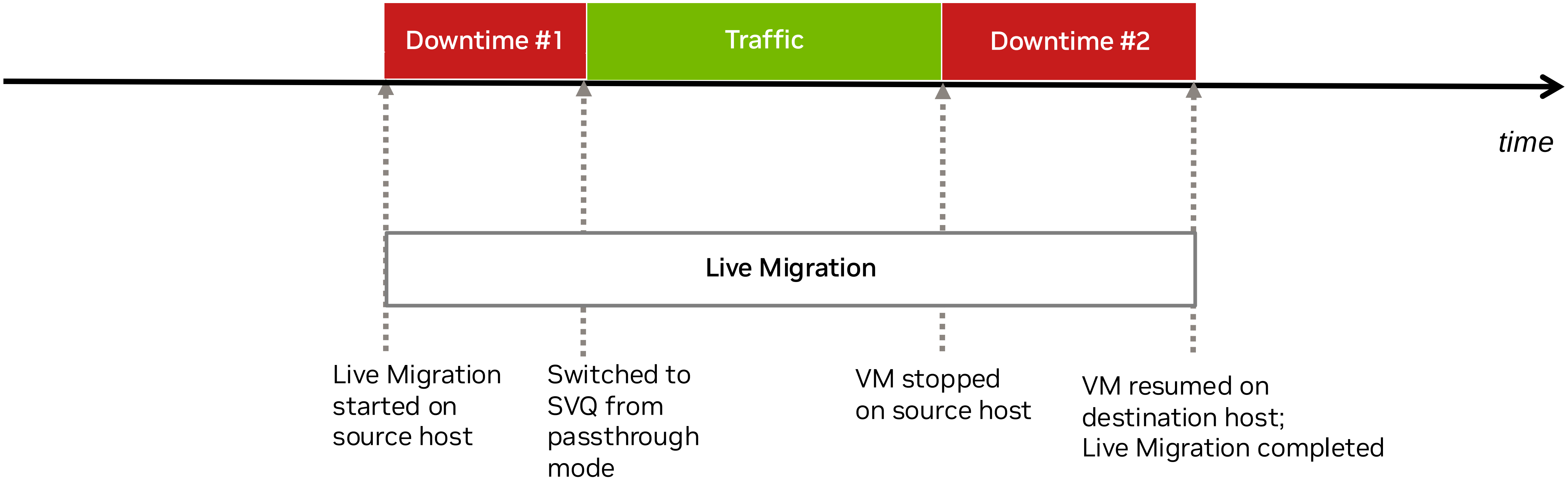
vDPA Live Migration Downtime Optimizations for VirtIO Net Devices

Dragoș Tătulea (NVIDIA)
<dtatulea@nvidia.com>

NetDevConf 0x18

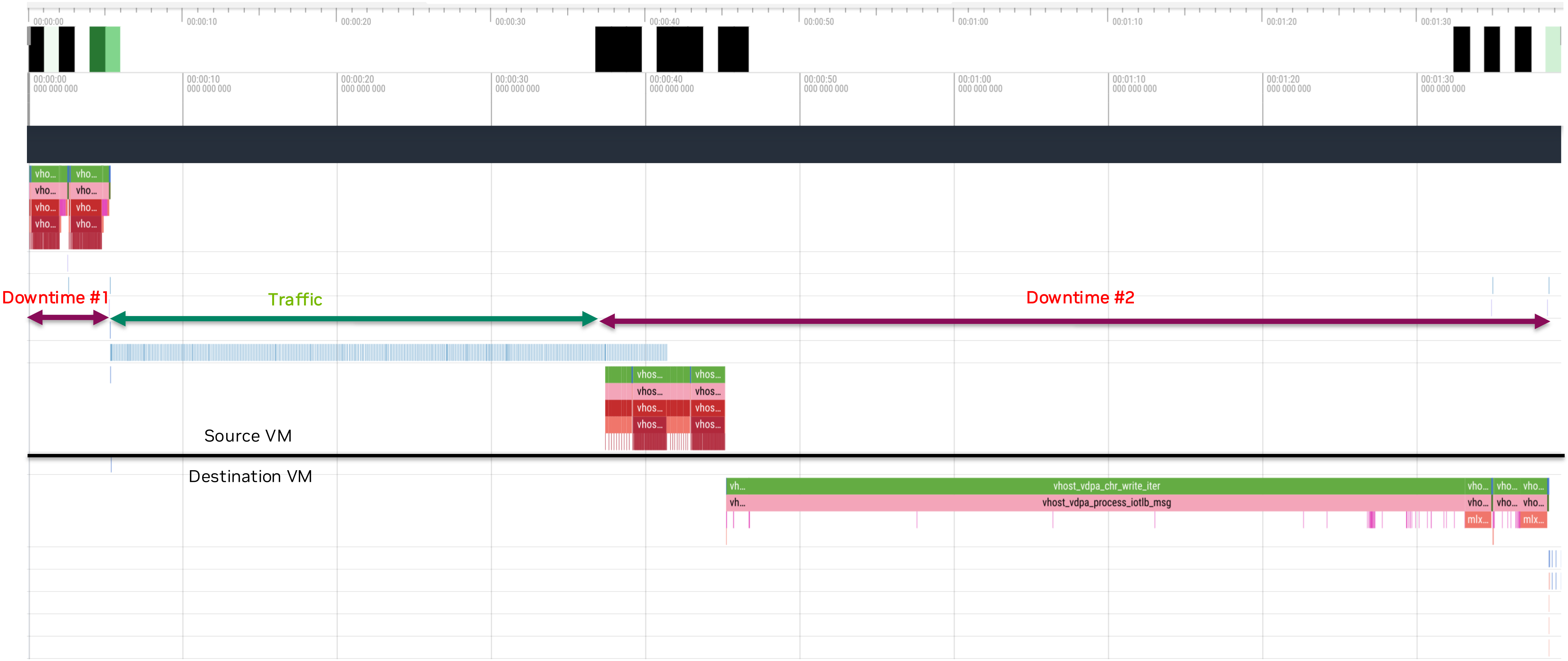
Live Migration Downtime

With Shadow VQ on Hardware vDPA


Downtime #1
Traffic
Downtime #2
time
Live Migration
Live Migration started on source host
Switched to SVQ from passthrough mode
VM stopped on source host
VM resumed on destination host; Live Migration completed

Live Migration Downtime

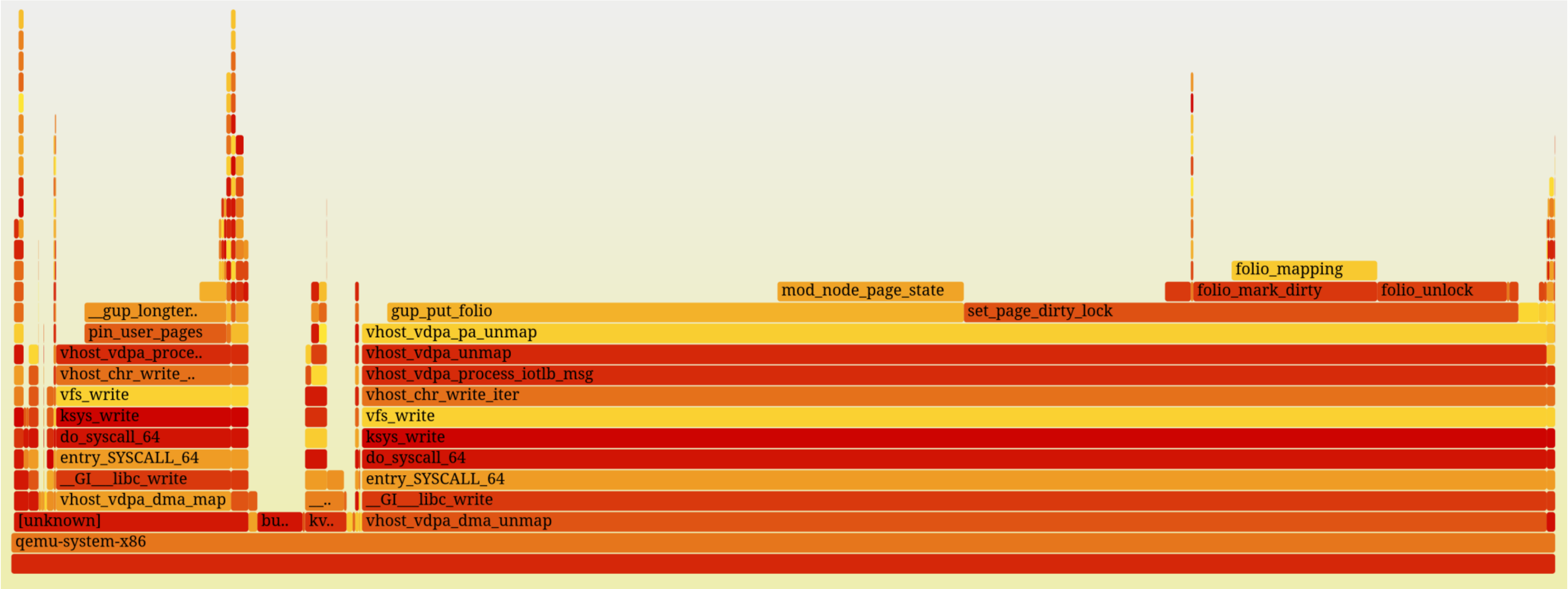
... a closer look



Downtime Breakdown

Downtime #1

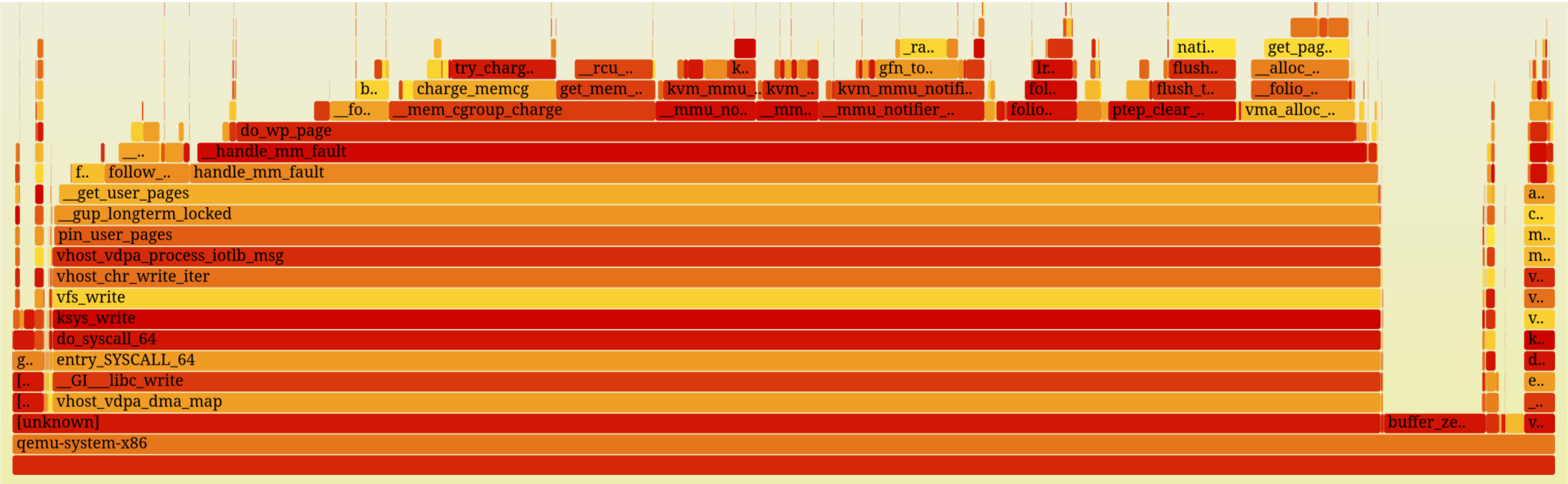
- Shared memory mapping for both guest memory and Shadow VQ
- Unnecessary memory mappings/unmappings
- Cost of tearing down and setting up hardware virtqueues



Downtime Breakdown

Downtime #2

- Page pinning cost at device startup
- Create and set up hardware virtqueues



Downtime Breakdown

Device considerations

- Expensive operations for mlx5_vdpa device
  - Memory mapping/unmapping
    - On chip IOMMU
    - Relative to map size
  - Virtqueue resource creation/deletion
    - Number of virtual queues

Path to a Lower Downtime

1. Move operations out of downtime
2. Reduce operations in downtime
3. Make operations faster

Early page pinning at device initialization

Qemu

- Page pinning done on destination after source device stops -> downtime
- Send guest memory layout to destination during active period of live migration
- Keep migration state on source until mappings on destination are done.
- Mapping done on a separate thread to not block QMP.
- [PATCH for 9.0 00/12] Map memory at destination .load_setup in vDPA-net migration

Descriptor group for SVQ ASID

Qemu, Kernel, Hardware

- Virtqueue descriptors in own mapping
  - Only descriptors -> much smaller maps
  - Buffers in still in default map
- New API for configuring descriptor virtqueues map in new ASID
- Merged in mainline v6.7
  - mlx5_vdpa: [PATCH vhost v4 00/16] vdpa: Add support for vq descriptor mappings
  - vdpa core: [PATCH RFC v2 0/3] vdpa: dedicated descriptor table group
- Qemu:
  - [PATCH 00/40] vdpa-net: improve migration downtime through descriptor ASID and persistent IOTLB
  - Based on page pinning series.

Decouple map flush from device reset

Kernel

- vDPA device reset
  - Reset device state
  - Reset mapping
- Map reset is not always necessary
- New API:
  - .reset(): does not reset map
  - .reset_map(): resets only map
  - .compat_reset(): old behaviour
- Merged in mainline v6.7: [PATCH v5 0/7] vdpa: decouple reset of iotlb mapping from device reset

Resumable virtqueues

Qemu, kernel, hardware

SUSPEND + RESET (pre v6.8)

- SUSPEND -> .suspend()

- GET_VRING_BASE -> .get_vq_state()
- RESET -> vdpa_reset() slow operation
- Change ASID for SVQ descriptors
- Restore device states
  - .set_config()
  - .set_vring_addr()
  - .set_vq_state()
  - .set_vq_ready()
  - .set_status(DRIVER_OK) slow operation

SUSPEND + RESUME (v6.8)

- SUSPEND -> .suspend()

- Change ASID for SVQ descriptors
- RESUME -> .resume()

- Merged in mainline v6.8: [PATCH vhost v5 0/8] vdpa/mlx5: Add support for resumable vqs
- Qemu Resumable VQs PoC - Upcoming

Pre-create Virtqueues

Device side optimization

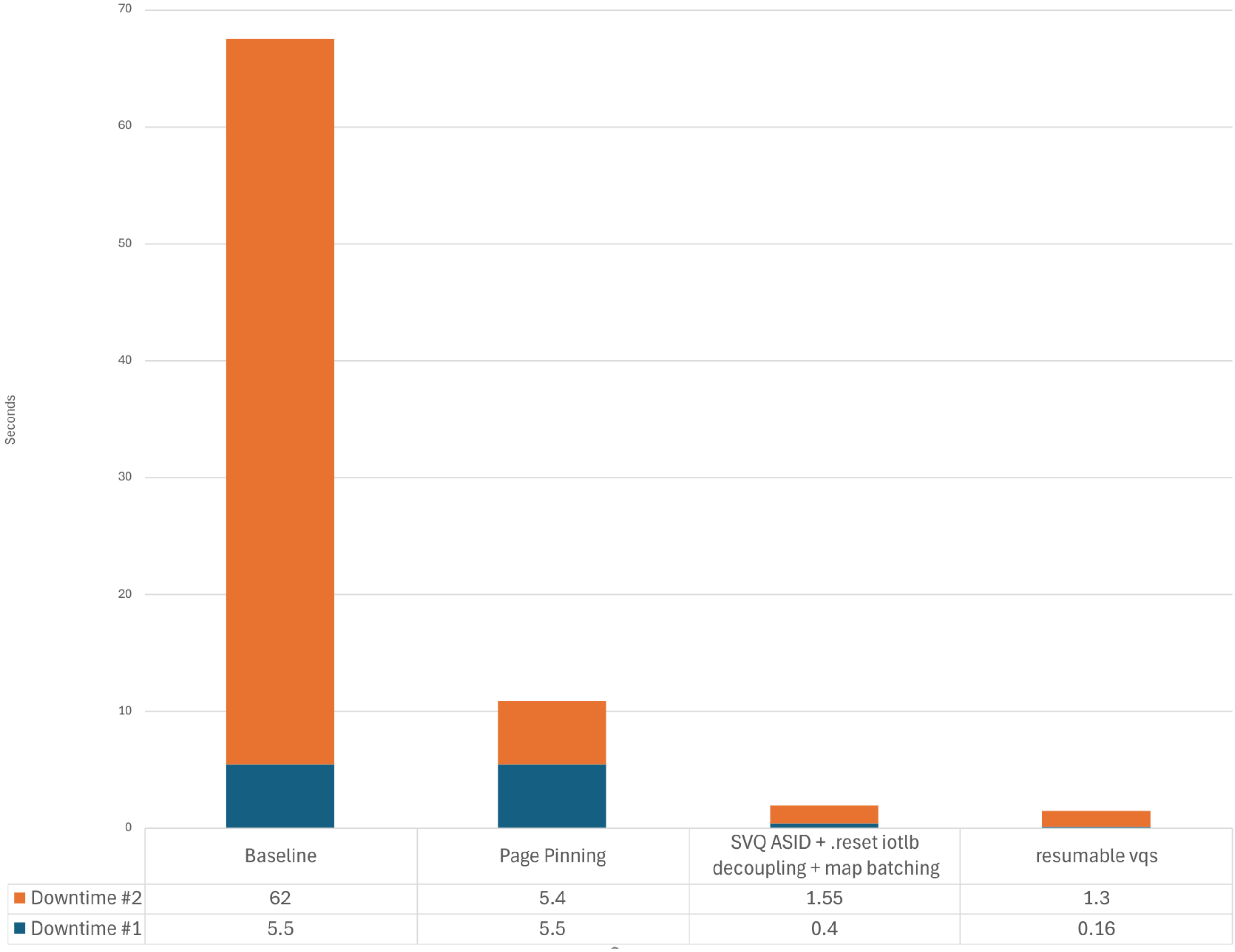
- Previously:
  - All hardware virtqueue resources created on device start (status DRIVER_OK)
  - For many devices with many virtqueues, this adds up.

- Now:
  - Initialize device with default state
  - Virtqueue configuration:
    - Fast, tracked on driver side.
    - Apply configuration to hardware

- Slow path: non default queue size
- [PATCH vhost v2 00/24] vdpa/mlx5: Pre-create HW VQs to reduce LM downtime
- Possible improvement: configurable default queue size

Downtime Reduction Overview

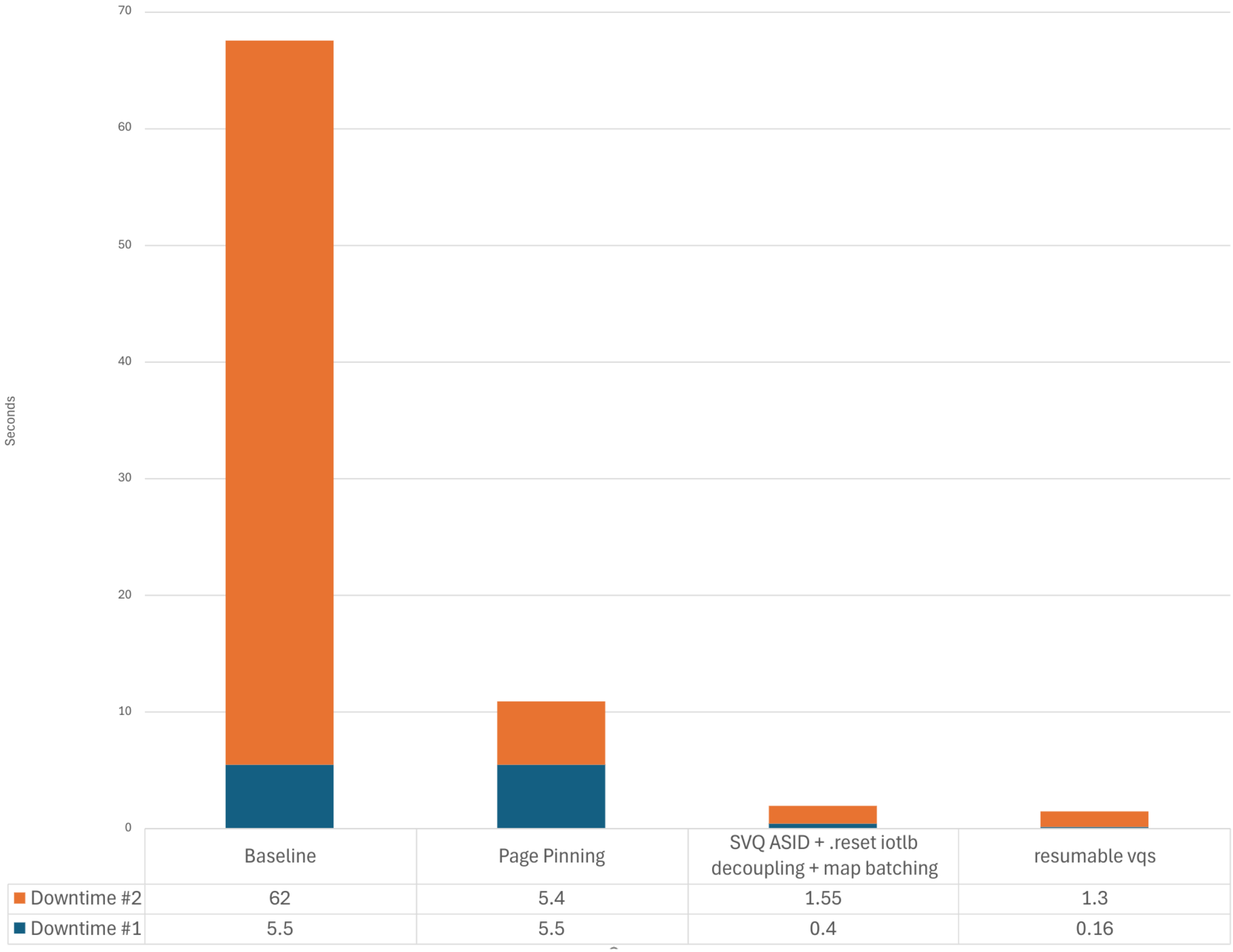
- Benchmark VM:
  - 128 GB RAM
  - 8 CPUs
  - 2 vDPA net devices, each with 4 data virtqueues
- Downtime measurements with *mig_mon* tool
- No hugepages



	Baseline	Page Pinning	SVQ ASID + .reset iotlb decoupling + map batching	resumable vqs
🟧 Downtime #2	62	5.4	1.55	1.3
🟦 Downtime #1	5.5	5.5	0.4	0.16

Downtime Reduction Overview

- Benchmark VM:
  - 128 GB RAM
  - 8 CPUs
  - 2 vDPA net devices, each with 4 data virtqueues
- Downtime measurements with *mig_mon* tool
- No hugepages
- VQ precreation: ~ 300 ms / device reduction
  - 256 GB RAM VM, 64 vCPUs, 4 devices x 32 virtqueues



	Baseline	Page Pinning	SVQ ASID + .reset iotlb decoupling + map batching	resumable vqs
🟧 Downtime #2	62	5.4	1.55	1.3
🟦 Downtime #1	5.5	5.5	0.4	0.16

Upcoming Improvements

Generic

- Scaling
  - Parallel device operations
  - Parallel VQ operations (device level)

- Move work out of downtime #2
  - Map memory ahead of time
    - [PATCH 0/6] Move memory listener register to vhost_vdpa_init
  - Device configuration before downtime
    - [RFC PATCH 0/5] virtio-net: Introduce LM early load

Thank you

Questions?