# A Hyper-V Based pvIOMMU Built on Generic I/O Page Table

Yu Zhang zhangyu1@linux.microsoft.com
Easwar Hariharan easwar.hariharan@linux.microsoft.com
Jacob Pan jacob.pan@linux.microsoft.com


# Target

- Enable endpoint device in Hyper-V Linux guest to fully leverage IOMMU capabilities e.g.,
  - In Kernel DMA protection.
  - User-space drivers based on VFIO/IOMMUFD.
  - vSVA & vPRI.
- Integrate seamlessly with the latest IOMMU core framework.
- Minimize the use of vendor/architecture specific code.
- Maximize performance by fully utilizing the underlying IOMMU HW features.

# Design choices

- Para-virtualization vs. emulation
- Nested vs. map/unmap trapping
  - ✓ Performance consideration
    - Nested avoids the need to trap for map operations and allows lazy IOTLB flush for unmaps.
    - Yet nested may require more page table walking during DMA translation.
  - ✓ Page table memory accounting is straightforward in nested, avoiding the need of stage-2 table for each guest IOVA domain.
  - ✓ Ecosystem has matured
    - Nested translation supported by most IOMMU hardware.
    - Generic I/O page table
- One pvIOMMU instance per guest for now.

**Conclusion: A pvIOMMU offering stage-1 translation based on generic I/O PT**

# Design choices

- Hypercall based enumeration (presence, capability, device scope etc.)
  - ✓ Avoid the need for spec extensions and virtual firmware modifications.
  - ✓ Common IOMMU feature set, no vendor specific capabilities.

Conclusion: Detect pvIOMMU capabilities and endpoint device membership by hypercalls.

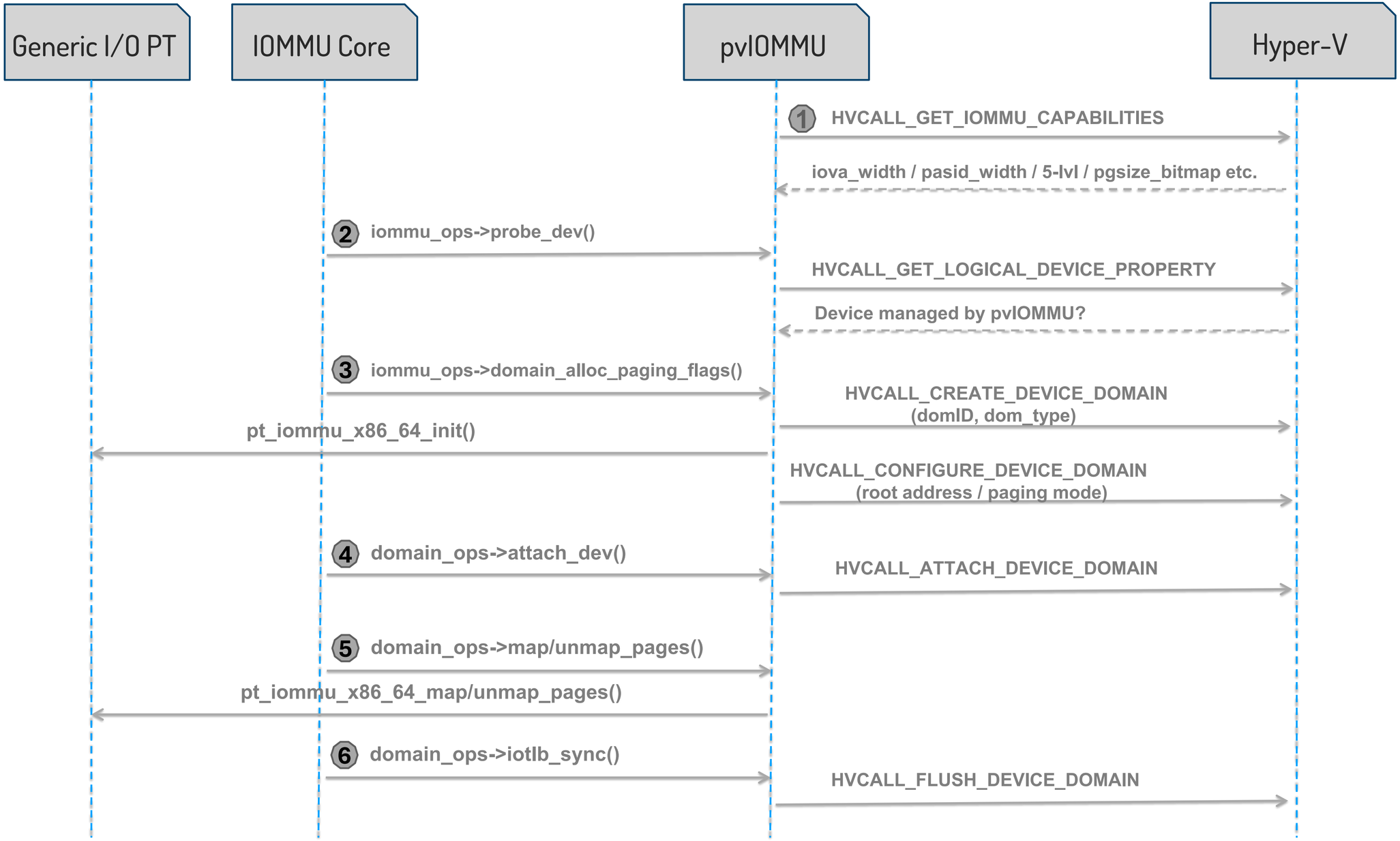
```
struct hv_output_get_iommu_capabilities {
        u32 size;
        u16 reserved;
        u8  max_iova_width;
        u8  max_pasid_width;

#define HV_IOMMU_CAP_PRESENT (1ULL << 0)
#define HV_IOMMU_CAP_S2 (1ULL << 1)
#define HV_IOMMU_CAP_S1 (1ULL << 2)
#define HV_IOMMU_CAP_S1_5LVL (1ULL << 3)
#define HV_IOMMU_CAP_PASID (1ULL << 4)
#define HV_IOMMU_CAP_ATS (1ULL << 5)
#define HV_IOMMU_CAP_PRI (1ULL << 6)

        u64 iommu_cap;
        u64 pgsize_bitmap;
} __packed;
```

```
struct hv_output_get_logical_device_property {
#define HV_DEVICE_IOMMU_ENABLED (1ULL << 0)
        u64 device_iommu;
        u64 reserved;
} __packed;
```

# pvIOMMU workflow



# Hyper-V IOMMU incremental planning

| Features | Status |
| --- | --- |
| Guest IOVA | - pvIOMMU enumeration<br>- Hypercall based device property detection<br>- DMA/DMA-FQ/Identity/Blocking domain supported<br>- Dmatest passed for Intel DSA devices<br>- RFC at https://lkml.org/lkml/2025/12/9/160 |
| Guest SVA (without IO page fault) | - PoC finished<br>- Idxd-Config DSA test passed for Intel DSA devices |
| IO page fault | - Upcoming in 2026 |

# Opens for discussion

- The PV IOMMU driver aims for architecture and vendor neutrality, yet it must manage guest-owned translation structures that are inherently architecture-specific.

- How to leverage HW acceleration features (e.g., to speed up the IOTLB flushing).

| Guest owned/managed | Solutions |
| --- | --- |
| S1 IOPT | Generic I/O PT |
| PASID table | PASID lib? * (not merged) |
| Hardware acceleration (CMDQV etc.) | Extract vendor code? |

*https://lore.kernel.org/linux-iommu/20210115121342.15093-3-vivek.gautam@arm.com/

# Opens for discussion (cont'd)

- What is the plan and demand for Virtio-IOMMU
  - Page table - based translation (Virtio-iommu specification with page table extensions)
  - CMDQ hardware acceleration