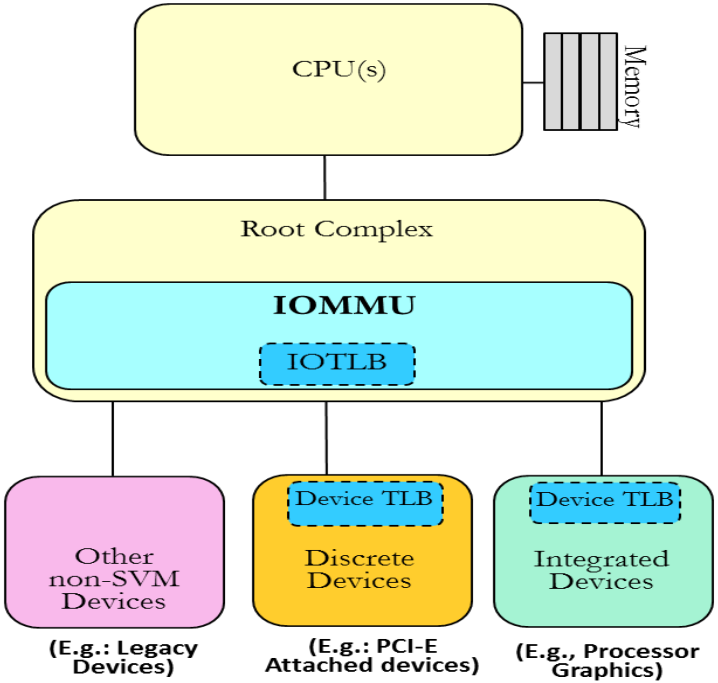
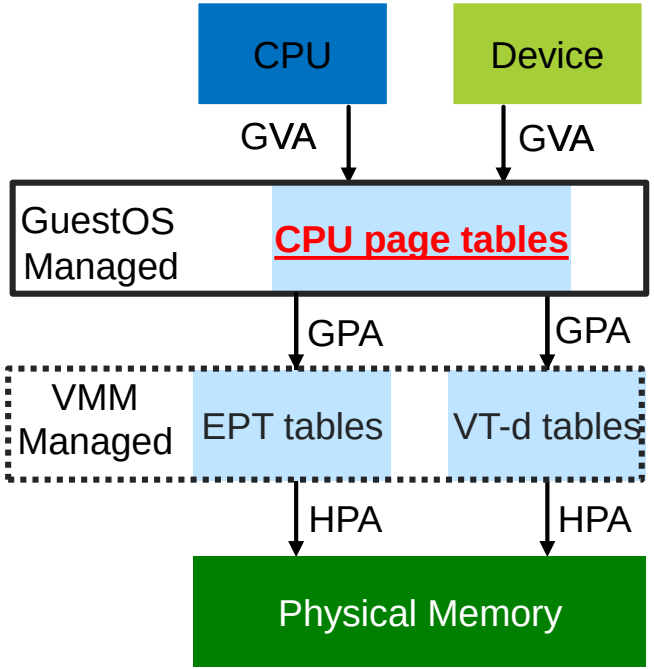
Shared Virtual Memory Virtualization

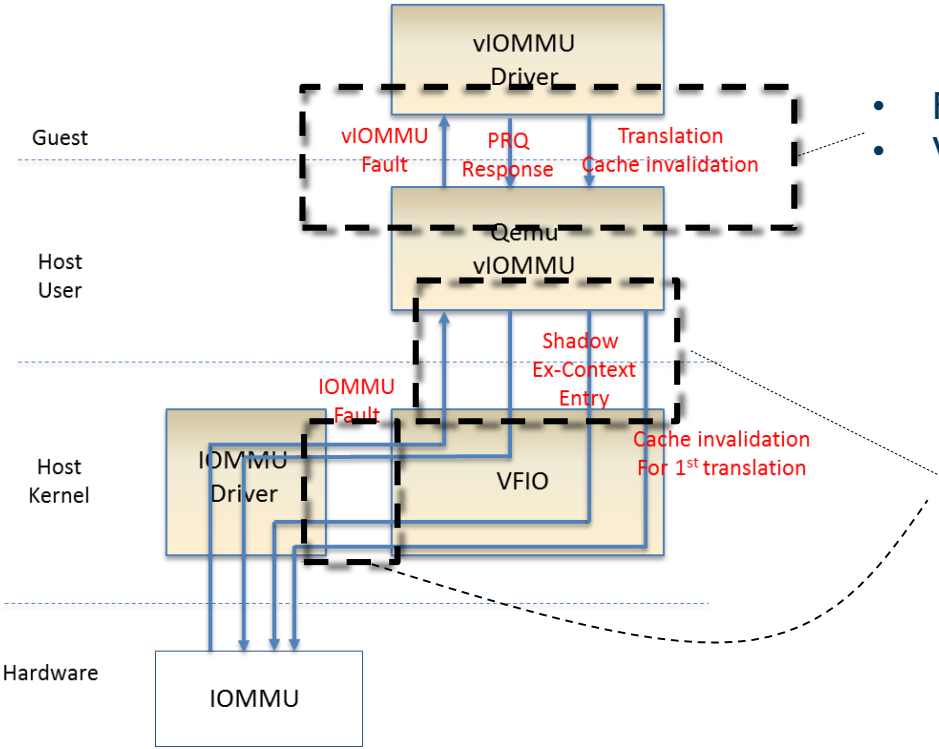
Liu, Yi L
yi.l.liu@intel.com
Raj, Ashok
ashok.raj@intel.com
Pan, Jacob
jacob.jun.pan@intel.com

High-Level View of SVM





SVM Virtualization Arch



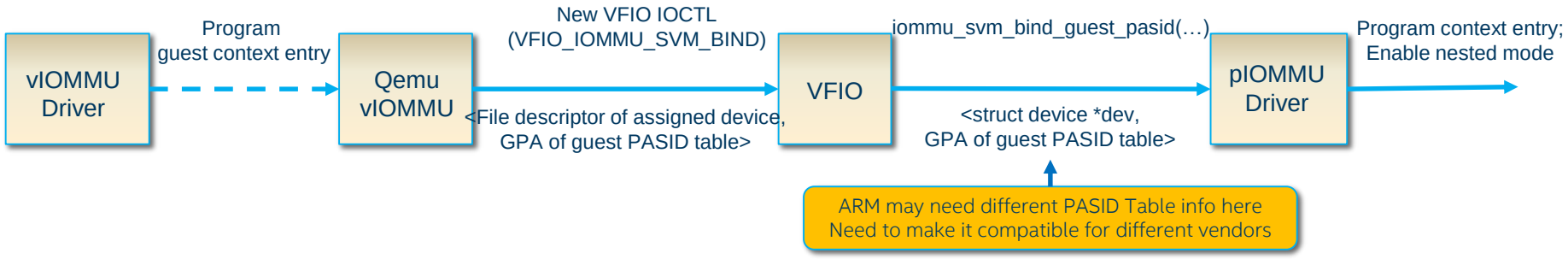
- Full emulated IOMMU interface
- Virtio-iommu interface

Cover four major flows (should be vendor agnostic)

- Bind guest PASID Table to host
- IOMMU-TLB Invalidation Propagation
- Fault reporting
- Page request report/response

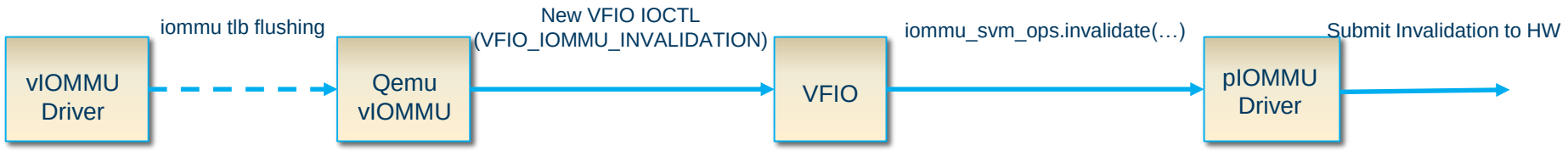
(today's focus)

Bind Guest PASID Table



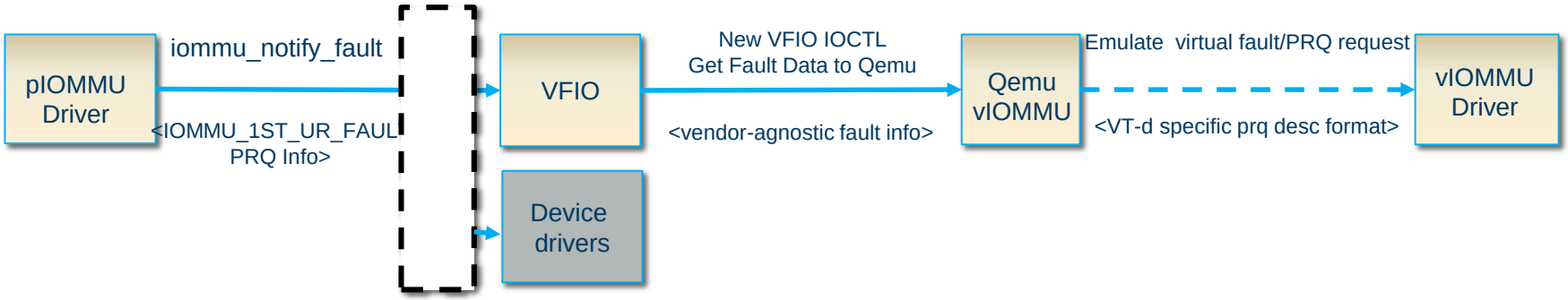
Open: the bind should be per-device or per-iommu-group?

IOMMU-TLB Invalidation Propagation



Open: abstract all the invalidation information to be vendor-agnostic, or have both generic part and vendor-specific part. Need a decision.

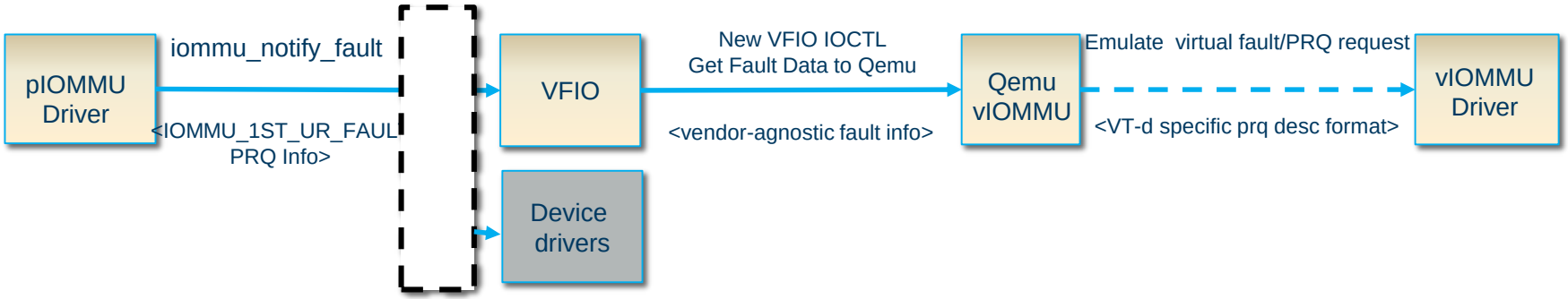
Fault Reporting



Open:

- **Unrecoverable** fault, need to define an abstract layer. Need all affected vendors to jump in.
- Report fault per-device or per-iommu-group?

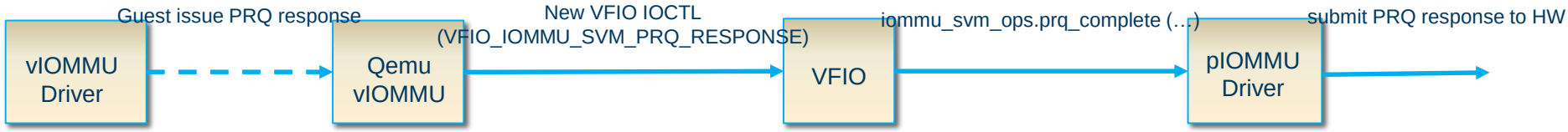
Fault Reporting



Open:

- **Recoverable** fault, has vendor-specific data, like VT-d has “private_data” . How to treat such data? Should it be a reason reason to accept hybrid format?
- Report fault per-device or per-iommu-group

Page Request Response



Open:
How to treat vendor-specific data? E.g. "private_data" in VT-d

Exposing IOMMU Capabilities to User Space

- Qemu may need to impose some limitations upon underlying physical IOMMU capabilities
  - For assigned device only

- Examples
  - Supported Pasid width
  - Address width

- Two methods
  - Thru VFIO ioctl
  - Thru IOMMU sysfs hierarchy

IOMMU Driver Extensions for vSVM

(LPC 2017 VFIO/IOMMU/PCI)

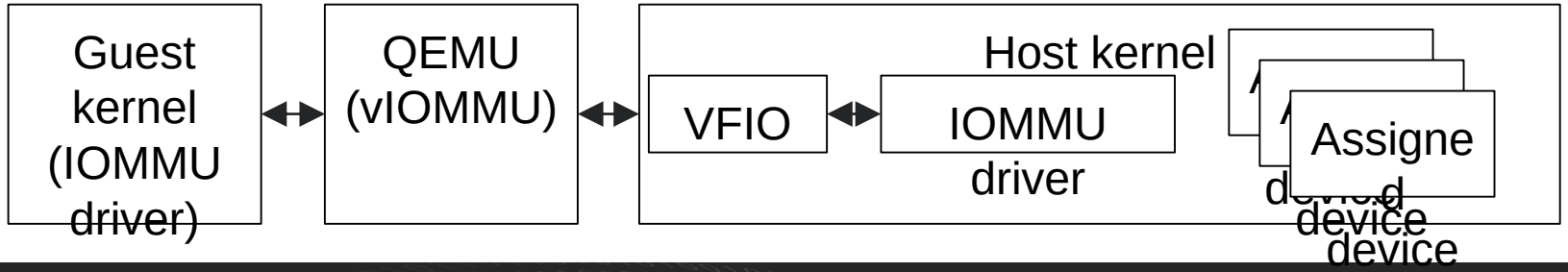
Jacob Pan (jacob.jun.pan@linux.intel.com)

Ashok Raj (ashok.raj@intel.com),

Yi Liu (yi.l.liu@intel.com),

Motivations

- Refer to Yi's talk on "Shared Virtual Memory Virtualization"
- Other in-kernel IOMMU users, e.g. fault reporting, drivers with private user-kernel interface.



Solution requirements

- Accommodate all IOMMU models
- Security: sanitize user requests
- Support both guest use and other in-kernel drivers
- Future proof
- PCI and non-PCI devices
- Support both vIOMMU and pvIOMMU (virtio IOMMU)

New IOMMU APIs

- Bind guest PASID table
- Invalidation pass down: user to invalidate translation structure caches
- Fault reporting and handling (Page request/response, Unrecoverable faults)
- Bind MM*: Allocate and bind PASID to a user task via VFIO (depends on unified IOMMU bind_mm call)

iommu_bind_pasid_table

```
int iommu_bind_pasid_table(struct iommu_domain *domain, struct device *dev,
                           struct pasid_table_info *pasidt_binfo)


/**
 * PASID table data used to bind guest PASID table to the host IOMMU. This will
 * enable guest managed first level page tables.
 * @ptr:        PASID table pointer
 * @size_order:     number of bits supported in the guest PASID table, must be less
 *                  or equal than the host table size.
 */
struct pasid_table_info {
        __u64           ptr;
        __u64           size_order;
};
```

Passdown invalidation

```
int iommu_invalidate(struct iommu_domain *domain,
                     struct device *dev, struct tlb_invalidate_info *inv_info)

struct tlb_invalidate_hdr {
        __u32 version;
        enum iommu_inv_type type;
};
struct tlb_invalidate_info {
        struct tlb_invalidate_hdr     hdr;
        enum iommu_inv_granularity        granularity;
        __u32           flags;
        __u8                    size;
        __u32           pasid;
        __u64           addr;
};
```

Opens:

- Data types, generic vs. model specific data
- Data layout: mandatory header + optional data with version specific generic data

Fault reporting

iommu_register_fault_notifier(struct device *dev, struct notifier_block *nb)
int iommu_fault_notifier_call_chain(struct iommu_fault_event *event)

```
struct device {
        struct iommu_group {
+       struct atomic_notifier_head fault_notifier;
…
        }
}
struct iommu_fault_event {
        struct device *dev;
        struct page_request_msg msg; /* Modeled aftr PCI ATS spec */
        enum iommu_fault_type type;
        enum iommu_fault_reason reason;
};
```

Opens:

- Per group notifier (current patchset) vs. per device callback

Page response

int iommu_page_response(struct iommu_domain *domain, struct device *dev,

struct page_response_msg *msg);

- PCI ATS spec., vendor specific needs?

```
struct page_response_msg {
        u64 paddr;
        u32 pasid;
        u32 rid:16;
        u32 did:16;
        u32 resp_code:4;
        u32 last_req:1;
        u32 pasid_present:1;
        u32 page_req_group_id : 9;
        u32 prot;
        enum page_response_type type;
        u32 prs_data;
};
```

Q/A

BACKUP

Invalidation data

```
/**
 * Translation cache invalidation header that contains mandatory meta data.
 * @version:      info format version, expecting future extesions
 * @type:         type of translation cache to be invalidated
 */
struct tlb_invalidate_hdr {
        __u32 version;
#define TLB_INV_HDR_VERSION_1 1
        enum iommu_inv_type type;
};

/**
 * Translation cache invalidation information, contains generic IOMMU
 * data which can be parsed based on model ID by model specific drivers.
 *
 * @granularity: requested invalidation granularity, type dependent
 * @size_order: 2^size of 4K pages, 0 for 4k, 9 for 2MB, etc.
 * @pasid:              processor address space ID value per PCI spec.
 * @addr:               page address to be invalidated
 * @flags       IOMMU_INVALIDATE_NO_PASID: targets non-pasid mappings,
 *                                         @pasid is not valid
 *              IOMMU_INVALIDATE_ADDR_LEAF: indicating that the pIOMMU
```

Fault event data

```
/**
 * Generic page request information based on PCI ATS and PASID spec.
 * @paddr: tells the offending page address
 * @pasid: contains process address space ID, used in shared virtual memory(SVM)
 * @rid: requestor ID
 * @page_req_group_id: page request group index
 * @last_req: last request in a page request group
 * @pasid_valid: indicates if the PRQ has a valid PASID
 * @prot: page access protection flag, e.g. IOMMU_READ, IOMMU_WRITE
 * @private_data: uniquely identify device-specific private data for an
 *                individual page request
 *
 */
struct page_request_msg {
        u64 paddr;
        u32 pasid;
        u32 rid:16;
        u32 page_req_group_id : 9;
        u32 last_req : 1;
        u32 pasid_valid : 1;
        u32 prot;
        u32 private_data;
```

Fault event data (cont'd)

```
/**
 * Generic fault event notification data, used by all IOMMU models
 *
 * - PCI and non-PCI devices
 * - Recoverable faults (e.g. page request) & un-recoverable faults
 * - DMA remapping and IRQ remapping faults
 *
 * @dev The device which faults are reported by IOMMU
 * @msg generic page fault message
 * @flags contains fault type, etc.
 * @reason additional reasons if relevant outside IOMMU driver, in most cases
 *         there is no need to report IOMMU fault to a guest or an in-kernel
 *         driver.
 */
struct iommu_fault_event {
            struct device *dev;
            struct page_request_msg msg;
            enum iommu_fault_type type;
            enum iommu_fault_reason reason;
};
```

Page response

```
/**
 * Generic page response information based on PCI ATS and PASID spec.
 * @paddr: servicing page address
 * @pasid: contains process address space ID, used in shared virtual memory(SVM)
 * @rid: requestor ID
 * @last_req: last request in a page request group
 * @resp_code: response code
 * @page_req_group_id: page request group index
 * @prot: page access protection flag, e.g. IOMMU_READ, IOMMU_WRITE
 * @type: group or stream response
 * @prs_data: uniquely identify device-specific page request data
 */
struct page_response_msg {
        u64 paddr;
        u32 pasid;
        u32 rid:16;
        u32 resp_code:4;
        u32 last_req:1;
        u32 pasid_present:1;
#define IOMMU_PAGE_RESP_SUCCESS  0
#define IOMMU_PAGE_RESP_INVALID  1
#define IOMMU_PAGE_RESP_FAILURE  0xF
        u32 page_req_group_id : 9;
```

Thank you!

Backups

Shared Virtual Memory

Shared Virtual Memory (SVM) is a VT-d feature that allows sharing application address space with the I/O device. The feature works with the PCI sig Process Address Space ID (PASID). SVM has the following benefits:

- Programmer gets a consistent view of memory across host application and device
- Efficient access to data, avoiding pining or copying overheads
- Memory over-commit via demand paging for both CPU and device access to memory

Currently, IGD is SVM capable device. In future SVM will be used for other IPs in server platforms to enable direct workload submission from user space driver.

Bind Guest PASID Table Detail

- IOMMU API Example:

```
static int iommu_bind_pasid_table(struct iommu_domain *domain,
                    struct device *dev, struct pasid_table_info *pasidt_binfo)
struct pasid_table_info {
        __u64   ptr;    /* PASID table ptr */
        __u64   size;   /* PASID table size*/
        __u32   model;  /* magic number */
};
```

Invalidation Propagation Detail

- IOMMU API Example:

  static int iommu_do_invalidate(struct iommu_domain *domain,
  struct device *dev, struct tlb_invalidate_info *inv_info)

- Previous discussion major on hjybrid format which has generic part and vendro=-specific part for the definition of struct tlb_invalidate_info.

- Latest discussion in community:

  https://patchwork.kernel.org/patch/9701023/

Fault Reporting(Cont.)

```
/*
 * Generic fault event notification data, used by all IOMMU architectures.
 *
 * - PCI and non-PCI devices
 * - Recoverable faults (e.g. page request) & un-recoverable faults
 * - DMA remapping and IRQ remapping faults
 *
 * @dev The device which faults are reported by IOMMU
 * @addr tells the offending address
 * @pasid contains process address space ID, used in shared virtual  memory (SVM)
 * @prot page access protection flag, e.g. IOMMU_READ, IOMMU_WRITE
 * @flags contains fault type, etc.
 * @length tells the size of the buf
 * @buf contains any raw or arch specific data
 *
 */
struct iommu_fault_event {
        struct device *dev;
        __u64 addr;
        __u32 pasid;
        __u32 prot;
        __u32 flags;
#define IOMMU_FAULT_PAGE_REQ        BIT(0)
#define IOMMU_FAULT_UNRECOV         BIT(1)
#define IOMMU_FAULT_IRQ_REMAP       BIT(2)
#define IOMMU_FAULT_INVAL     BIT(3)
        __u32 length;
        __u8  buf[];
};
```

Page Request Response (Cont.)

- Similar parameter generalization required as for recoverable fault reporting (to be refined)
- Parameters in ATS spec
  - pSID
  - Addr
  - PASID
  - Response Code: response code defined by ATS spec
  - Page group index
- Should be general:
  - PASID Present: indicates whether PASID is present
- Parameter being VT-d specific:
  - Type: for VTd, may have two, Page Group Response, Page Stream Response
  - Private Data: included in VTd prq_desc

Patchset in Community

- vSVM Qemu patch
  https://www.spinics.net/lists/kvm/msg148798.html

- vSVM IOMMU patch:
  https://lists.gnu.org/archive/html/qemu-devel/2017-04/msg04946.html

- Fault report patch:
  https://www.spinics.net/lists/kvm/msg145372.html
  https://lists.gnu.org/archive/html/qemu-devel/2017-02/msg04145.html

- IOMMU patch with fault report
  https://lwn.net/Articles/726606/