Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

IOMMUFD & Generic Page Table Discussion

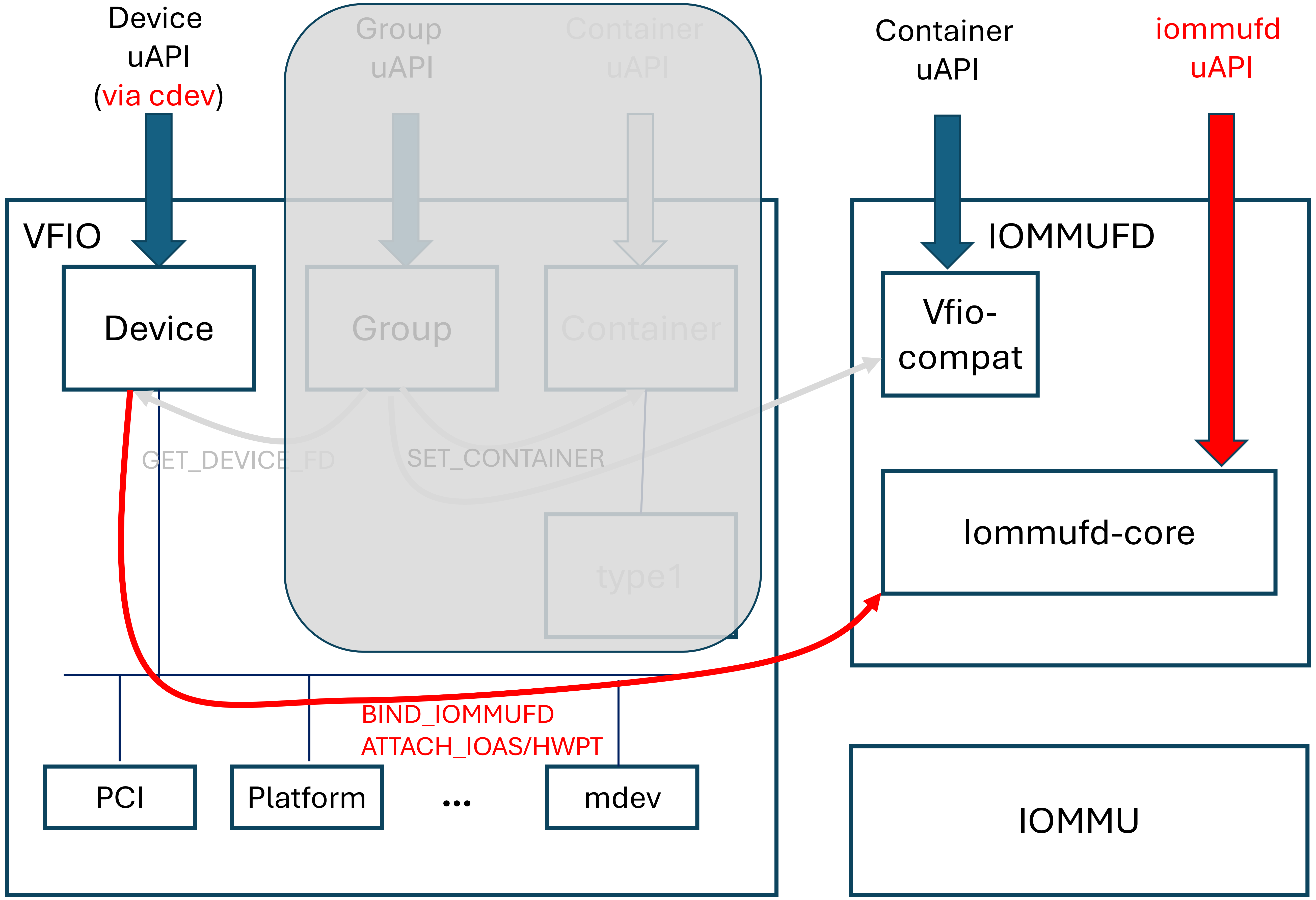
Jason Gunthorpe

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

Session Goals

Overview

- Expose IOMMU HW to user-space control
- Provide advanced IOMMU features
- Provide high performance “kernel bypass” to support VMs
- Support all kernel subsystems equally:
  VDPA, VFIO, uacce, etc



IOMMUFD Features

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

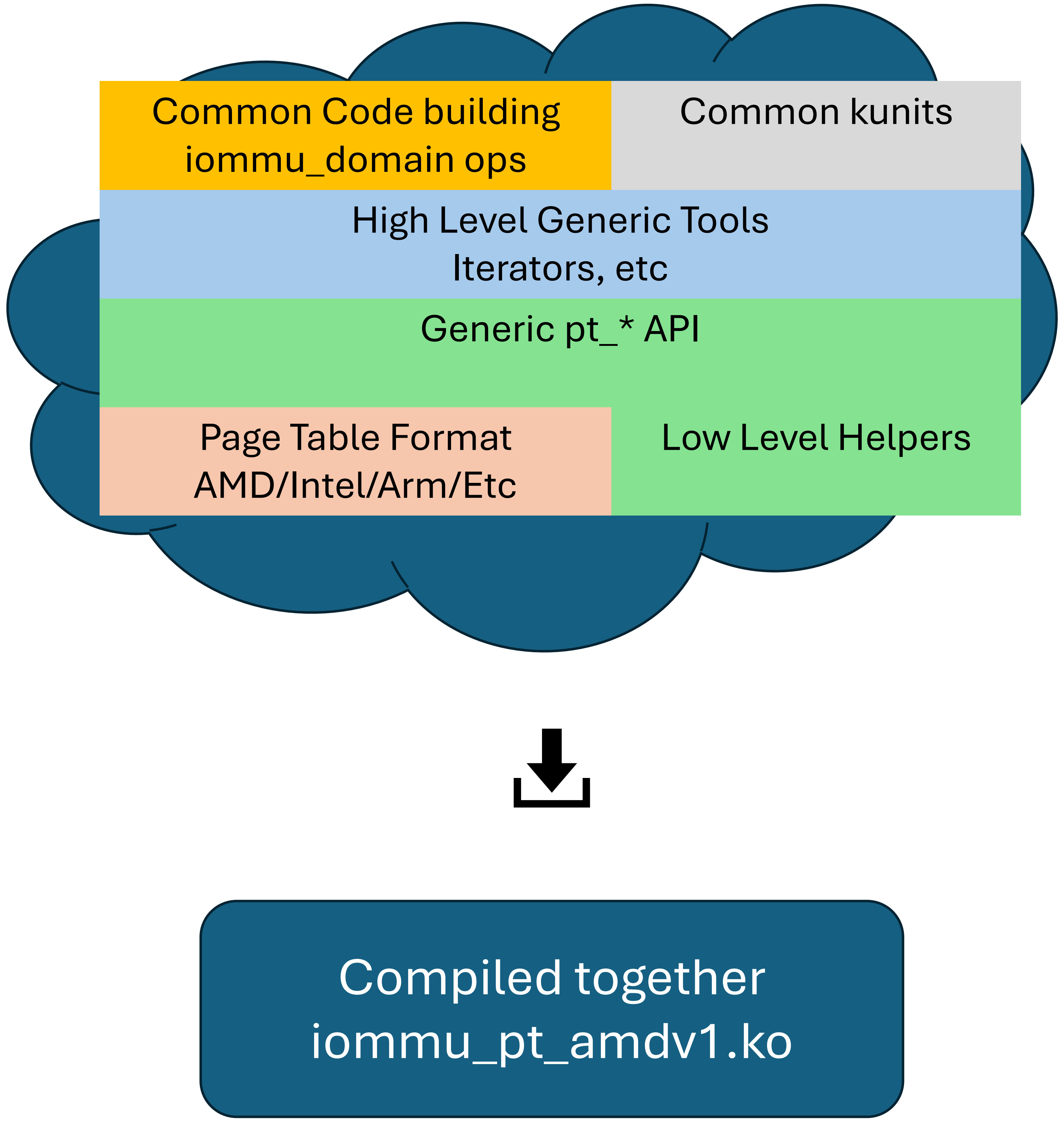
Driver Modernization

1. Global Static ‘never fail’ BLOCKED and IDENTITY domains
2. Map before Attach for PAGING domains
3. attach_dev() failure does not change HW
4. Hitless IDENTITY->DMA/PAGING->IDENTITY for active IOVA
5. PAGING->BLOCKED->PAGING is !IDENTITY, even under failure
6. PAGING domains attached to a PASID
7. Set IDENTITY/BLOCKED/PAGING on RID while PASID in use
8. SVA domains attached to a PASID
9. SVA domains use new core infrastructure
10. Hitless change between domains when possible

Generic Page Table

- Project to bring a mm like API around many different page table formats
- Conceivably useful for mm/kvm, focusing on iommu

- Multi compilation approach
- Single code providing iommu_domain ops



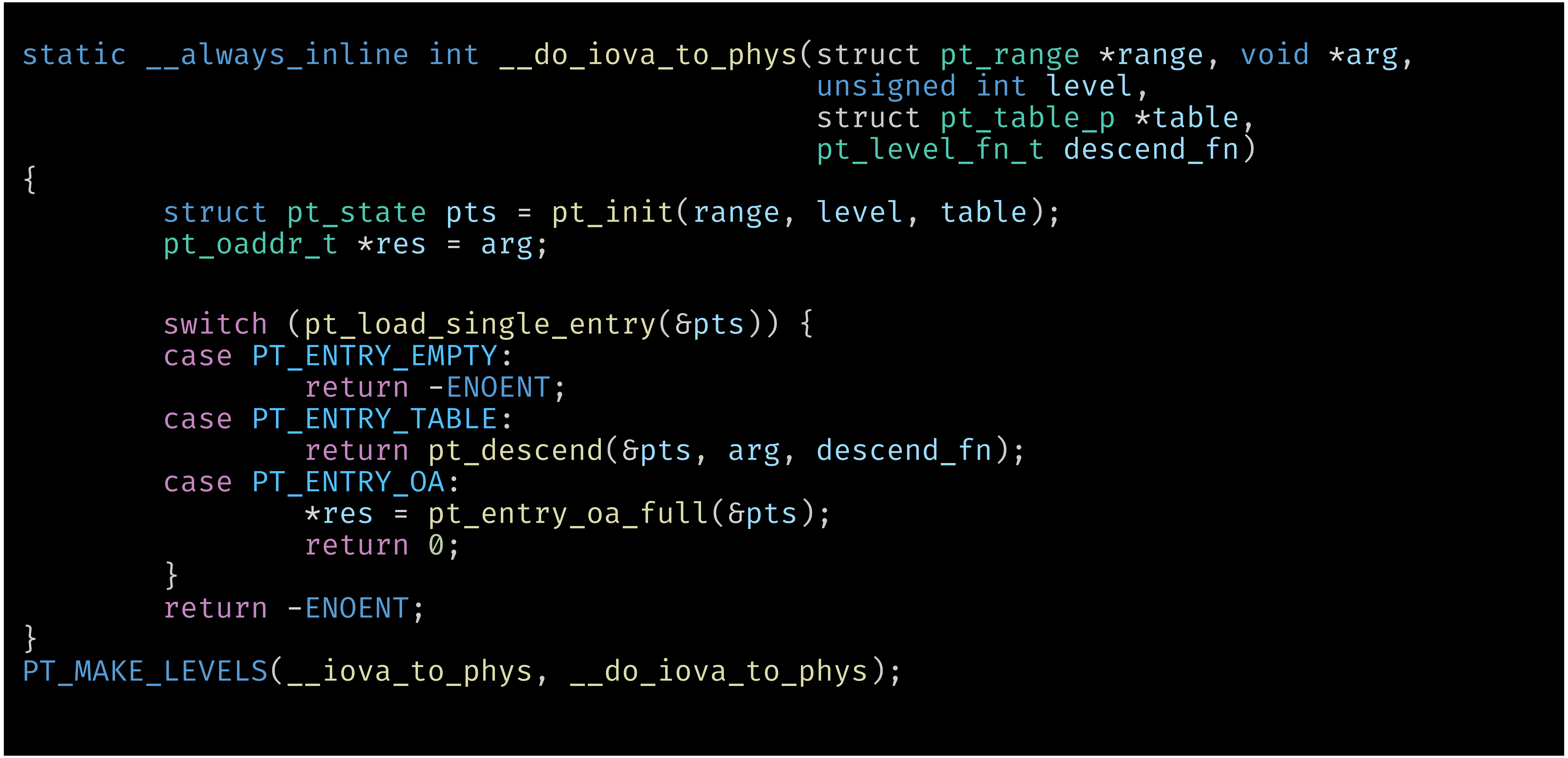
Common Code building iommu_domain ops	Common kunits
High Level Generic Tools Iterators, etc	
Generic pt_* API	
Page Table Format AMD/Intel/Arm/Etc	Low Level Helpers

Compiled together
iommu_pt_amdv1.ko

General Features

- 2 to 6 levels
- Levels can be different sizes
- Top level can change at runtime (expand address space)
- Leaf pages at any level (huge pages)
- “Contiguous page” features
- Both fully inlined and recursive function options
- Fully arch independent – no assumption of PAGE_SIZE
- Kunit test ensures identical API semantics across formats

Simple Example



```
static __always_inline int __do_iova_to_phys(struct pt_range *range, void *arg,
                                             unsigned int level,
                                             struct pt_table_p *table,
                                             pt_level_fn_t descend_fn)
{
        struct pt_state pts = pt_init(range, level, table);
        pt_oaddr_t *res = arg;

        switch (pt_load_single_entry(&pts)) {
        case PT_ENTRY_EMPTY:
                return -ENOENT;
        case PT_ENTRY_TABLE:
                return pt_descend(&pts, arg, descend_fn);
        case PT_ENTRY_OA:
                *res = pt_entry_oa_full(&pts);
                return 0;
        }
        return -ENOENT;
}
PT_MAKE_LEVELS(__iova_to_phys, __do_iova_to_phys);
```

IOMMU Enhancements

With common code build more complex page table operations

1. Batching map from bio_vec, sg, iommufd’s batch
   Avoid walk-per-page overheads
2. Unmap and read-back
   Avoid having to do iova_to_phys() prior to unmap, and the resulting walk per PAGE_SIZE
3. Cut a hole in a large page to put a small page (iopt_cut_iova)
   Use cases in new VMM scenarios, similar to VFIO type 1.0 capability
4. Reduce page size for finer dirty tracking granularity
   Increase page size to restore if there is a failed migration
5. Faster dirty bit readback
6. More aggressive freeing of table memory to avoid waste

Incremental Plan

- RFC “maximum” implementation.
  All formats, all functionality, some new functions
- Stripped down AMD only core logic and AMD driver implementation
- New functionality
- More drivers