IOMMU-Assisted Memory Management

Sharing Virtual-Memory Objects with PCIe Devices in the Linux Kernel

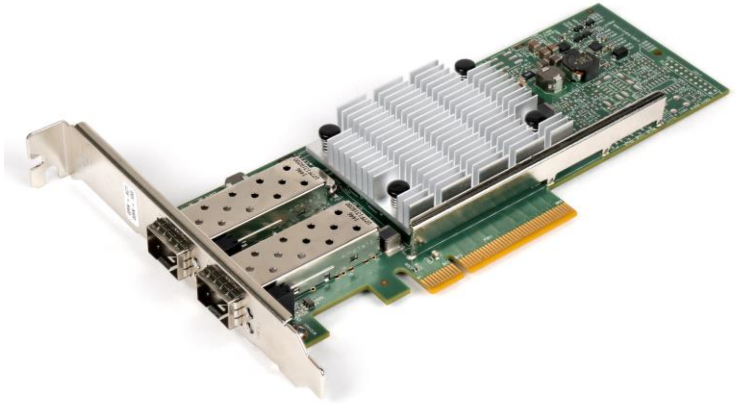
Spring Meeting FGBS

Kenny Albes

14.03.2024

Introduction

- Increasingly heterogeneous systems
  - GPUs, FPGAs, NICs, AI-Accelerators, NVMe SSDs
  - Efficient data transfers via Direct Memory Access (DMA)
- Unrestricted DMA is inherently unsafe
  - Need for memory protection and isolation
    → IOMMU as MMU for external devices
- Memory management/abstractions must account for external devices

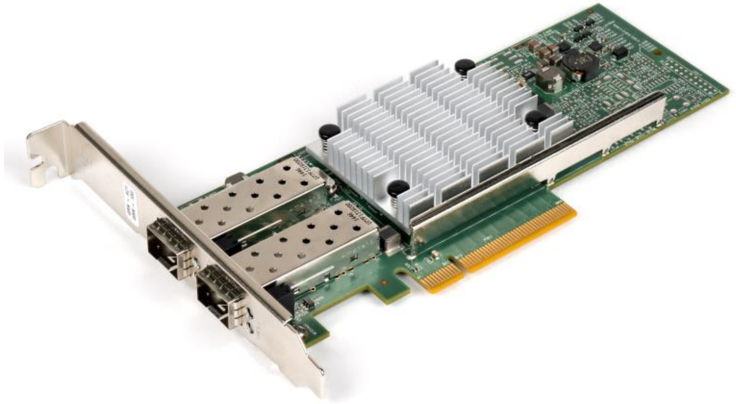


Qlogic NIC [1]

Samsung NVMe SSD [2]

Introduction

- Increasingly heterogeneous systems
  - GPUs, FPGAs, NICs, AI-Accelerators, NVMe SSDs
  - Efficient data transfers via Direct Memory Access (DMA)
- Unrestricted DMA is inherently unsafe
  - Need for memory protection and isolation
    → IOMMU as MMU for external devices
- Memory management/abstractions must account for external devices



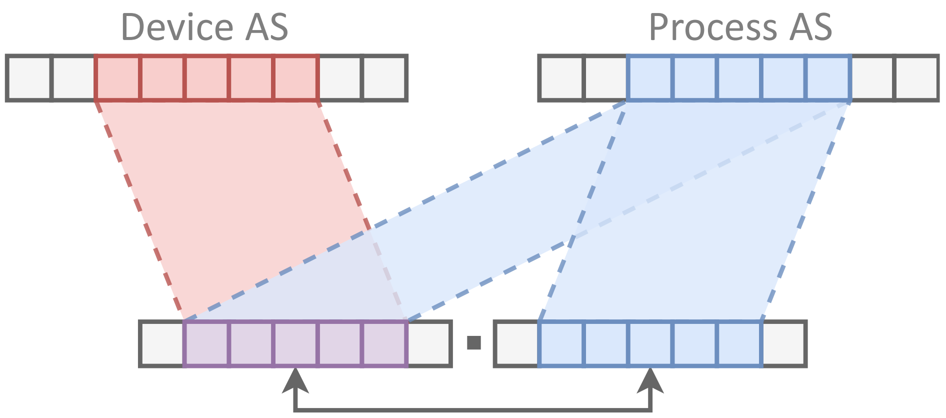
Qlogic NIC [1]

Samsung NVMe SSD [2]

How to allow for efficient data transfers?

Sharing Data With Devices

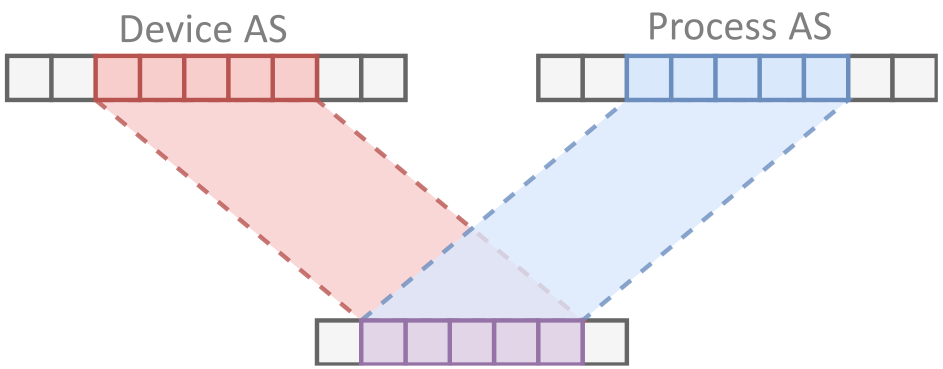
Isolation: *Bounce Buffers*



Data needs to be copied

→ Only feasible for small transfers

Speed: *Zero Copy*



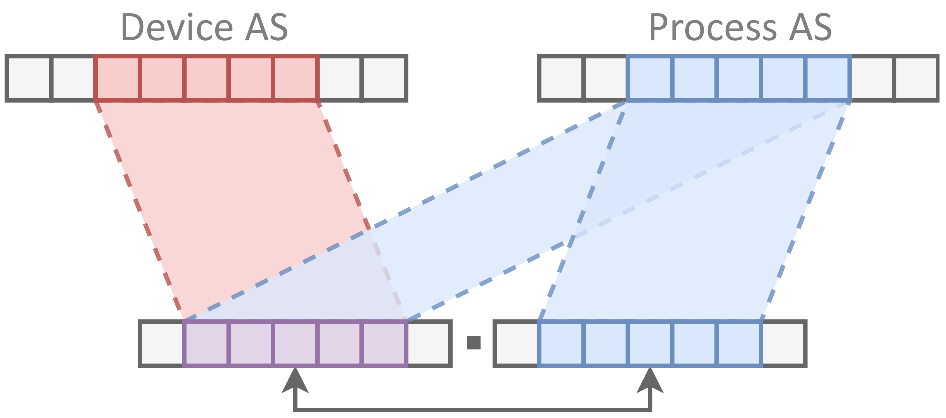
Device is not isolated

Idea: Map shared buffer alternatingly

→ Large management overhead

Sharing Data With Devices

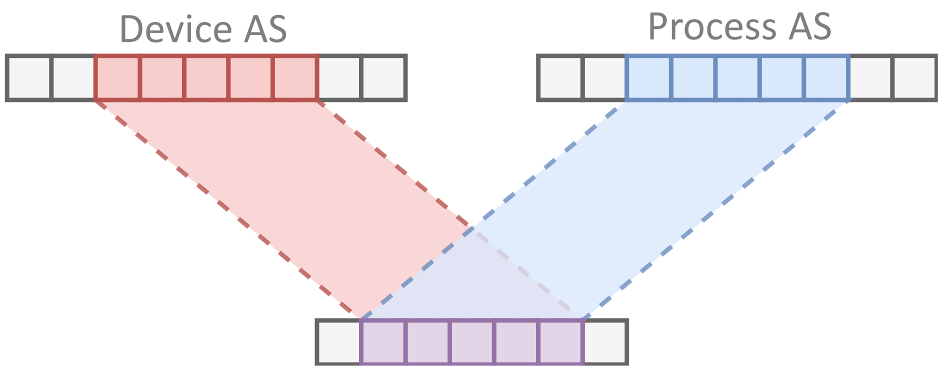
Isolation: *Bounce Buffers*



Data needs to be copied

→ Only feasible for small transfers

Speed: *Zero Copy*



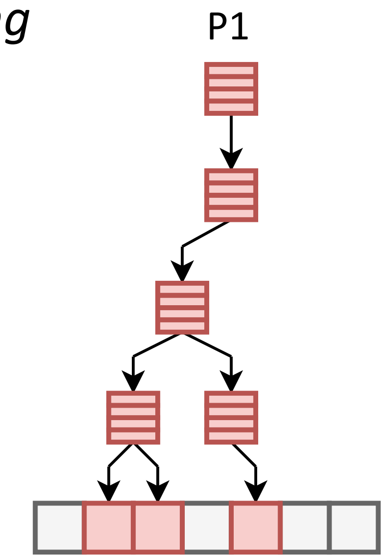
Device is not isolated

Idea: Map shared buffer alternatingly
→ Large management overhead

Can we have both: Isolation and speed?

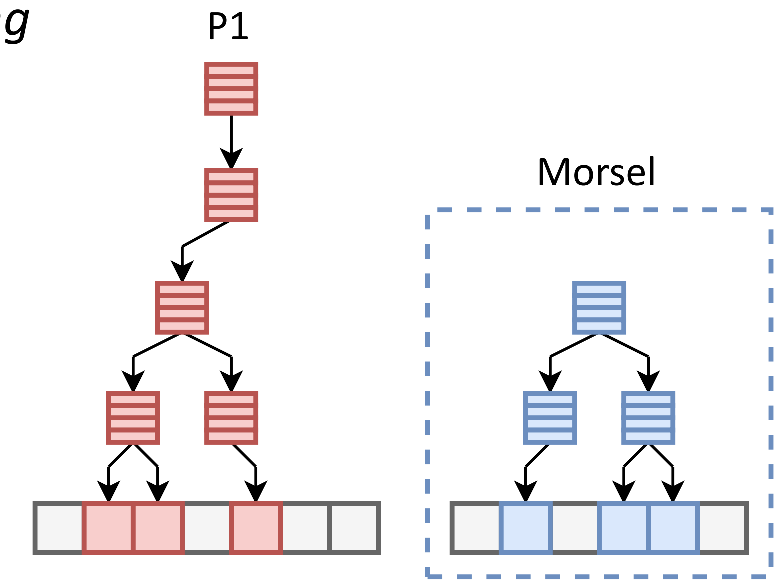
Morsels

- **A page table subtree that acts as a self-contained, sharable memory object** [3]
  - Lifetime detached from processes
  - Shared efficiently between processes by *mounting*



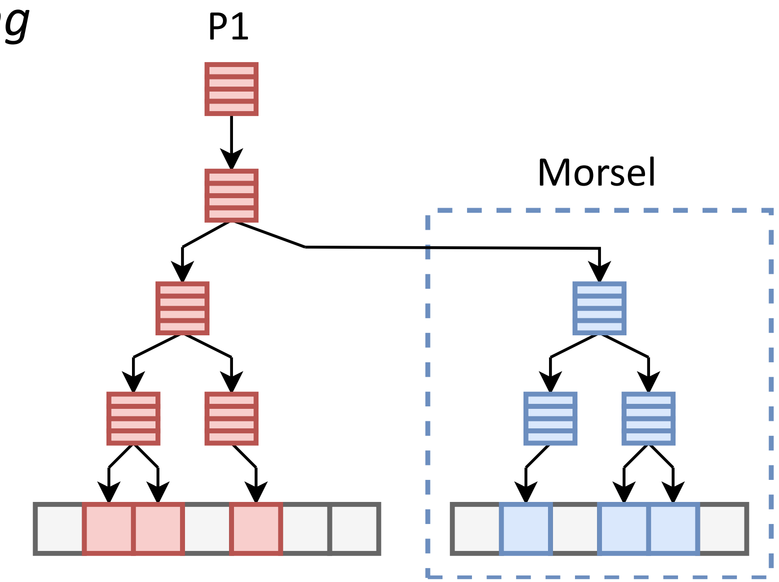
Morsels

- **A page table subtree that acts as a self-contained, sharable memory object** [3]
  - Lifetime detached from processes
  - Shared efficiently between processes by *mounting*



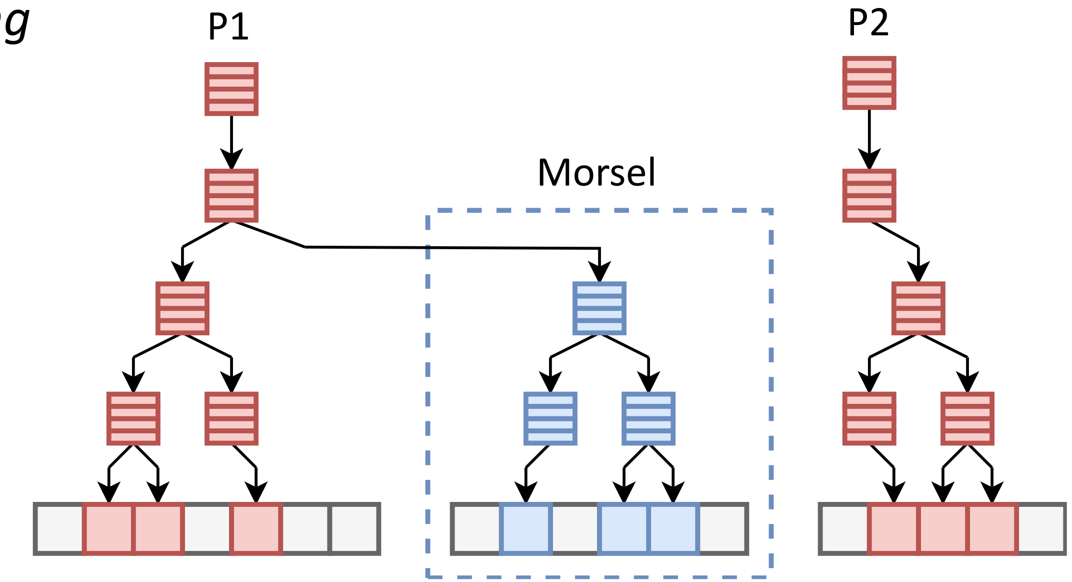
Morsels

- **A page table subtree that acts as a self-contained, sharable memory object** [3]
  - Lifetime detached from processes
  - Shared efficiently between processes by *mounting*



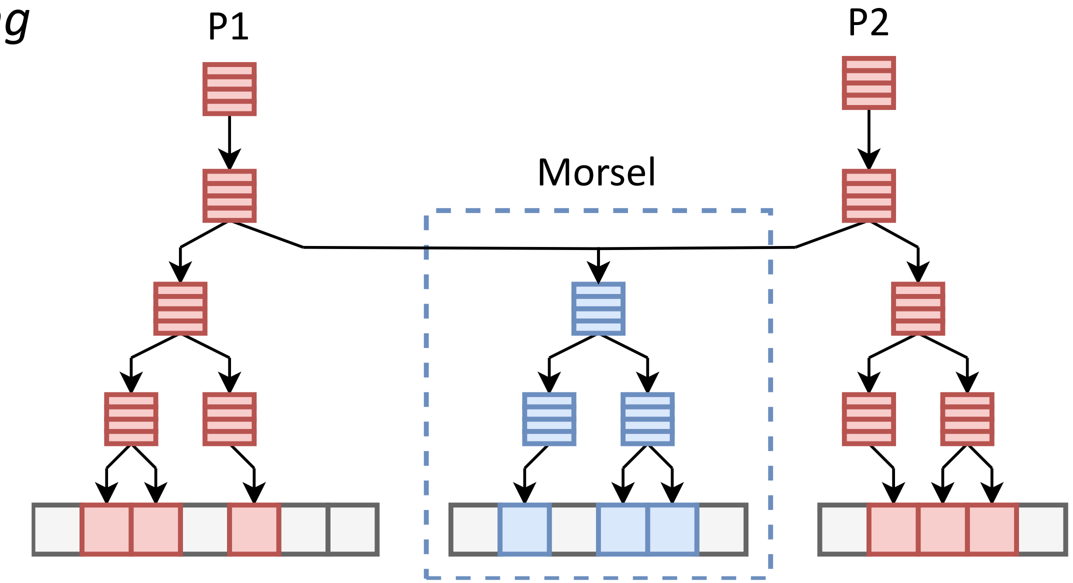
Morsels

- **A page table subtree that acts as a self-contained, sharable memory object** [3]
  - Lifetime detached from processes
  - Shared efficiently between processes by *mounting*



Morsels

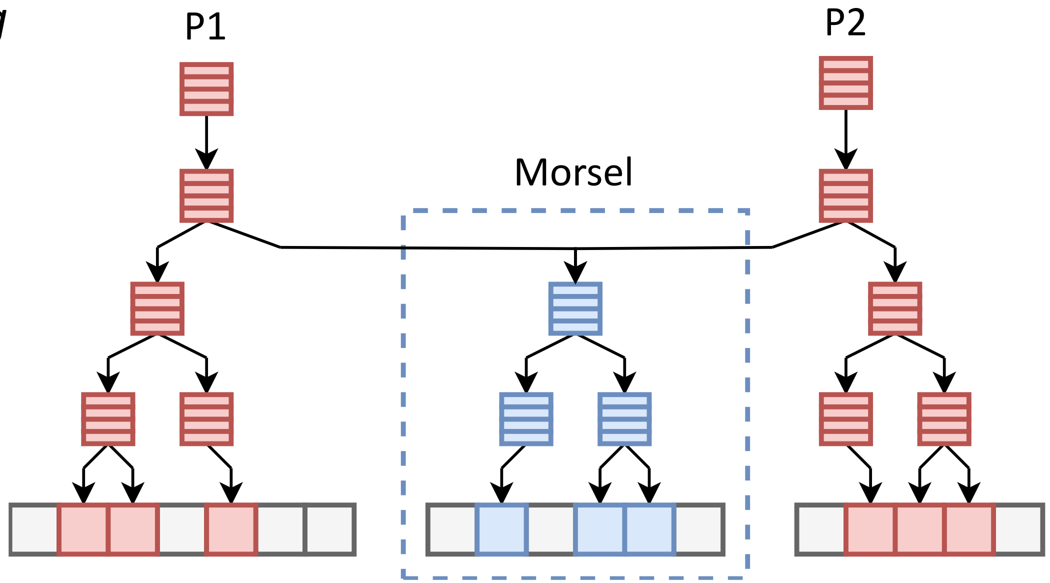
- **A page table subtree that acts as a self-contained, sharable memory object** [3]
  - Lifetime detached from processes
  - Shared efficiently between processes by *mounting*



A Morsel, shared between processes

Morsels

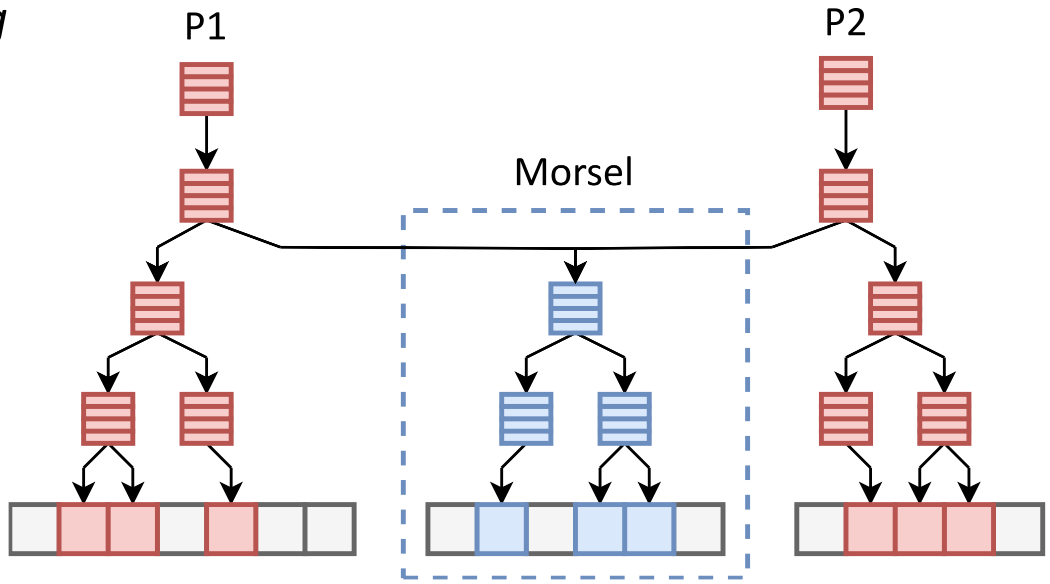
- **A page table subtree that acts as a self-contained, sharable memory object** [3]
  - Lifetime detached from processes
  - Shared efficiently between processes by *mounting*
- Lazily populated through page fault mechanism
- Working implementation for **processes** Linux



A Morsel, shared between processes

Morsels

- **A page table subtree that acts as a self-contained, sharable memory object** [3]
  - Lifetime detached from processes
  - Shared efficiently between processes by *mounting*
- Lazily populated through page fault mechanism
- Working implementation for **processes** Linux



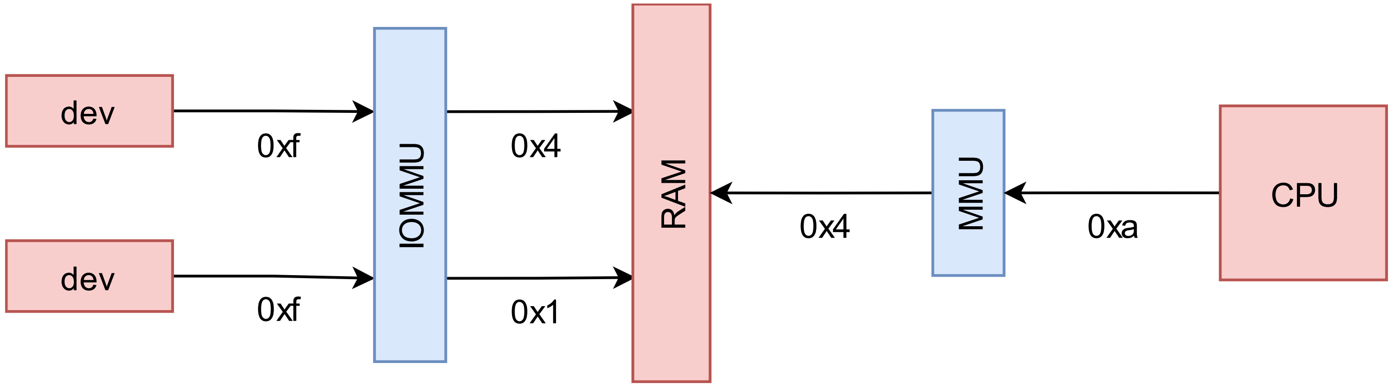
A Morsel, shared between processes

Extend Morsels to DMA-capable devices

On AMD64 (for now)

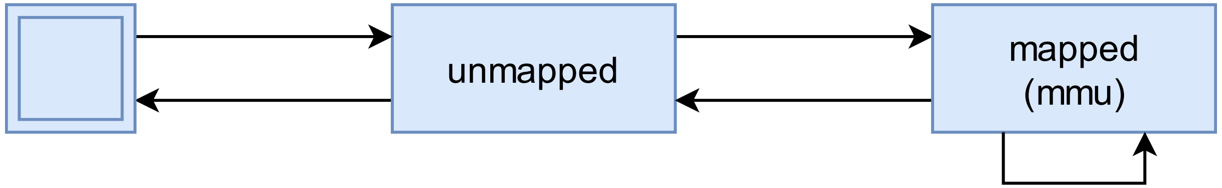
AMD IOMMU (AMD-V)

- AMD's IOMMU specification
  - Similar to an AMD64 MMU
  - Not compatible with Intel's IOMMU
- *IO Page Tables* for each *domain*
- Multiple IOTLBs
- No (efficient) page fault handling



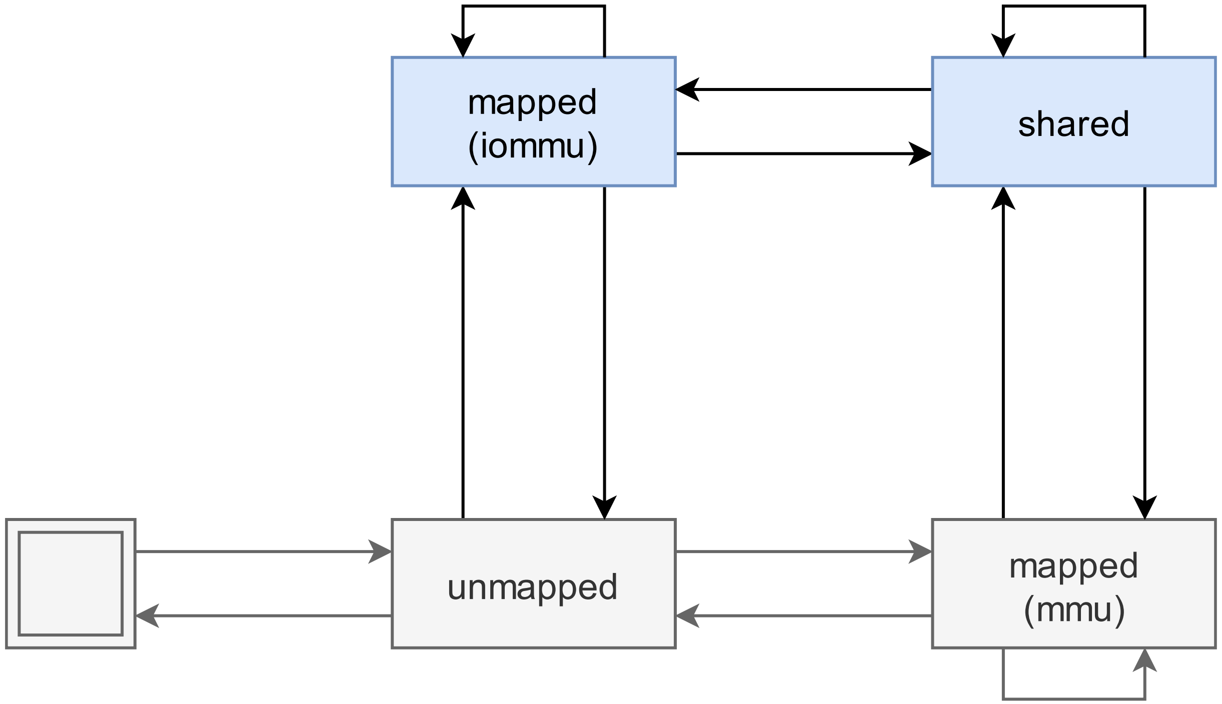
IOMMU in a heterogeneous system

Extending Morsels to the IOMMU



Different states of a Morsel

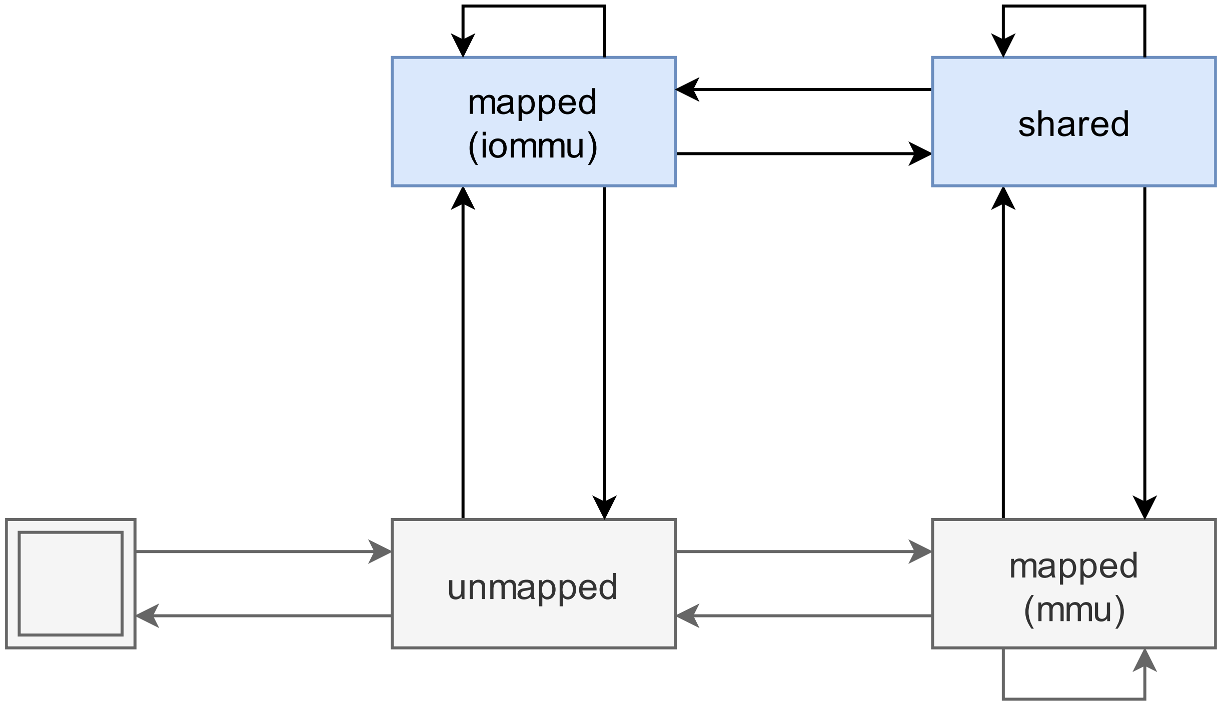
Extending Morsels to the IOMMU



Different states of a Morsel

Extending Morsels to the IOMMU

- New use cases:
  1. Device-Process (static)
  2. Device-Device
  3. **Device-Process (dynamic)**

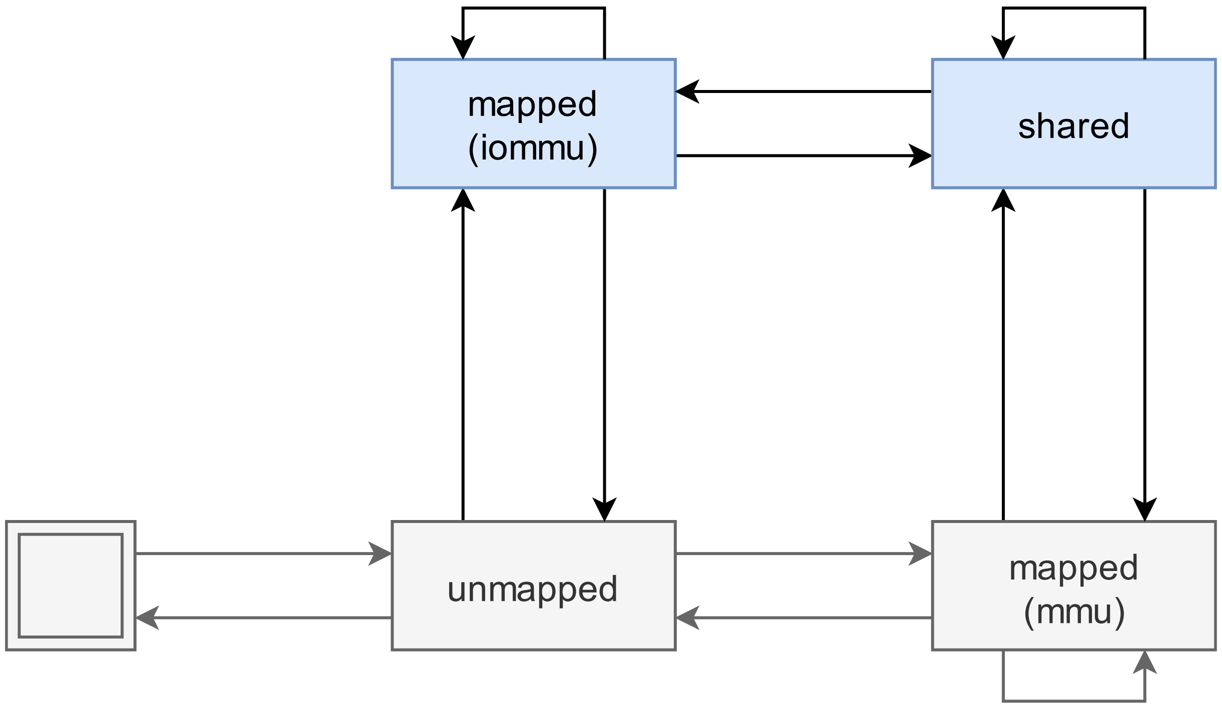


Different states of a Morsel

Extending Morsels to the IOMMU

- New use cases:
  1. Device-Process (static)
  2. Device-Device
  3. **Device-Process (dynamic)**

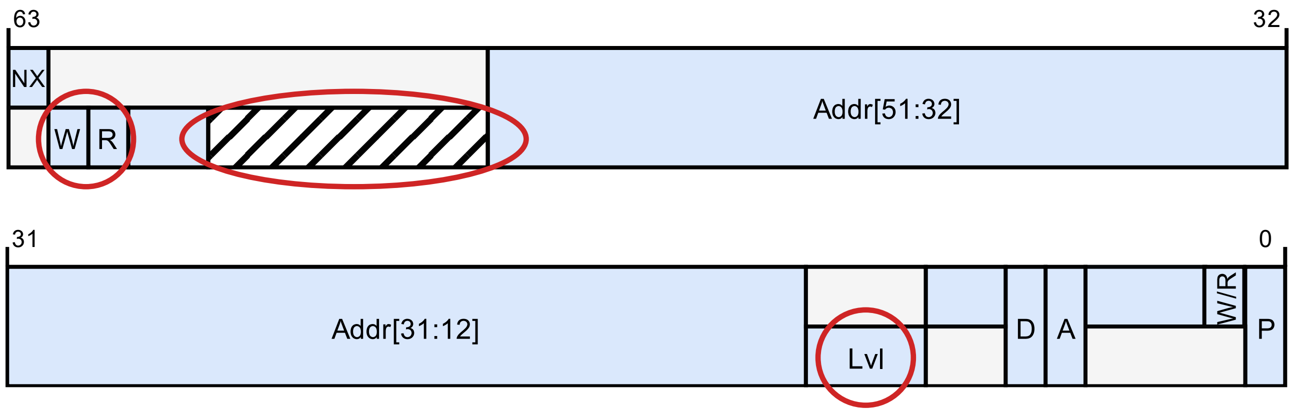
- Requirements:
  1. Morsels are visible to devices
  2. Pinning



Different states of a Morsel

Extending Morsels – Sharing Page Tables

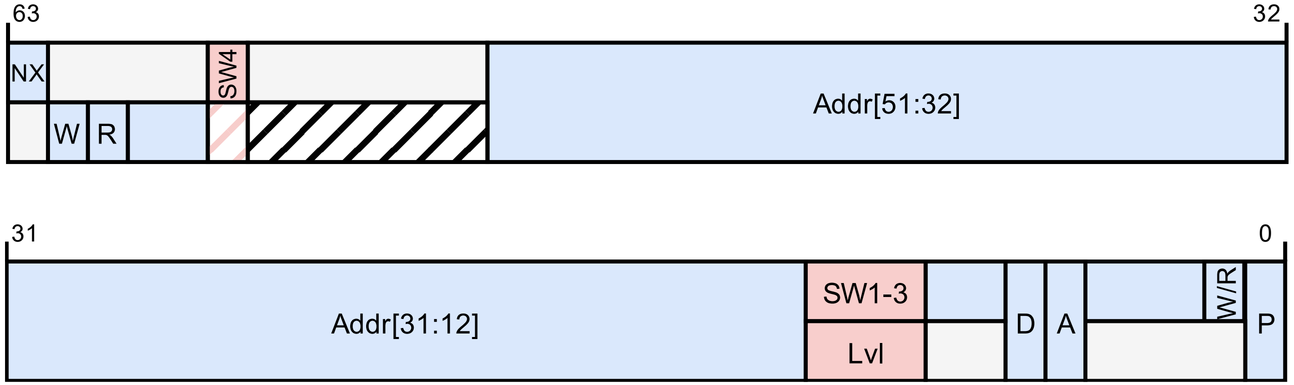
- Sharing tables between MMU and IOMMU is possible (for AMD)
  - Compatible formats
  - (IO)TLBs must be synchronized manually (when unmapping)
- Minor changes required



Comparison between PTEs of an AMD64 MMU (top) and an AMD IOMMU (bottom)

Extending Morsels – Sharing Page Tables

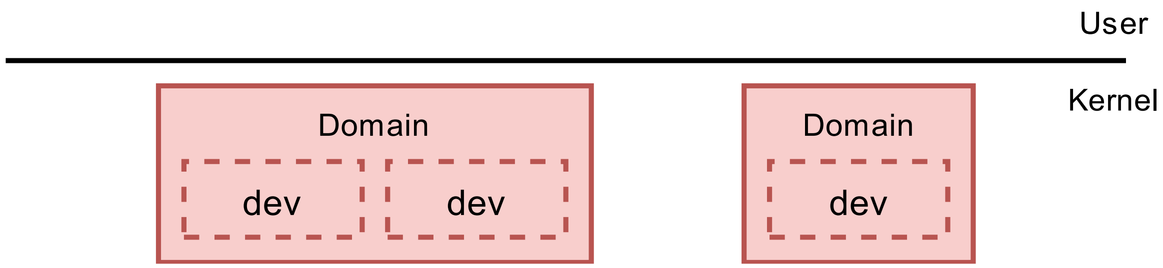
- Sharing tables between MMU and IOMMU is possible (for AMD)
  - Compatible formats
  - (IO)TLBs must be synchronized manually (when unmapping)
- Minor changes required
- *Possible conflicts*
  - Bits ignored by the MMU may be used by SW
  - Linux uses 4 of these



Comparison between PTEs of an AMD64 MMU (top) and an AMD IOMMU (bottom)

Implementation for Linux 6.1

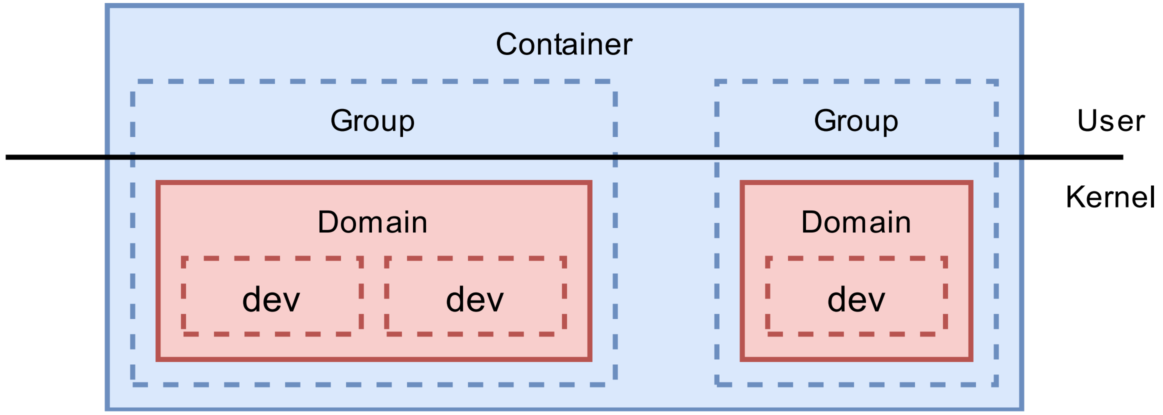
- IOMMU driver
  - Kernel-only
  - API defines operations on domains
- Extended driver API to allow sharing of page tables
  - Implemented for AMD-IOMMU



IOMMU subsystem in Linux 6.1

Implementation for Linux 6.1

- IOMMU driver
  - Kernel-only
  - API defines operations on domains
- Extended driver API to allow sharing of page tables
  - Implemented for AMD-IOMMU
- *Virtual Function I/O (VFIO)*
  - Current IOMMU user space API
  - Container manages lifetime of its mappings
- Extended VFIO to handle Morsels
  - Benefit from existing lifetime management



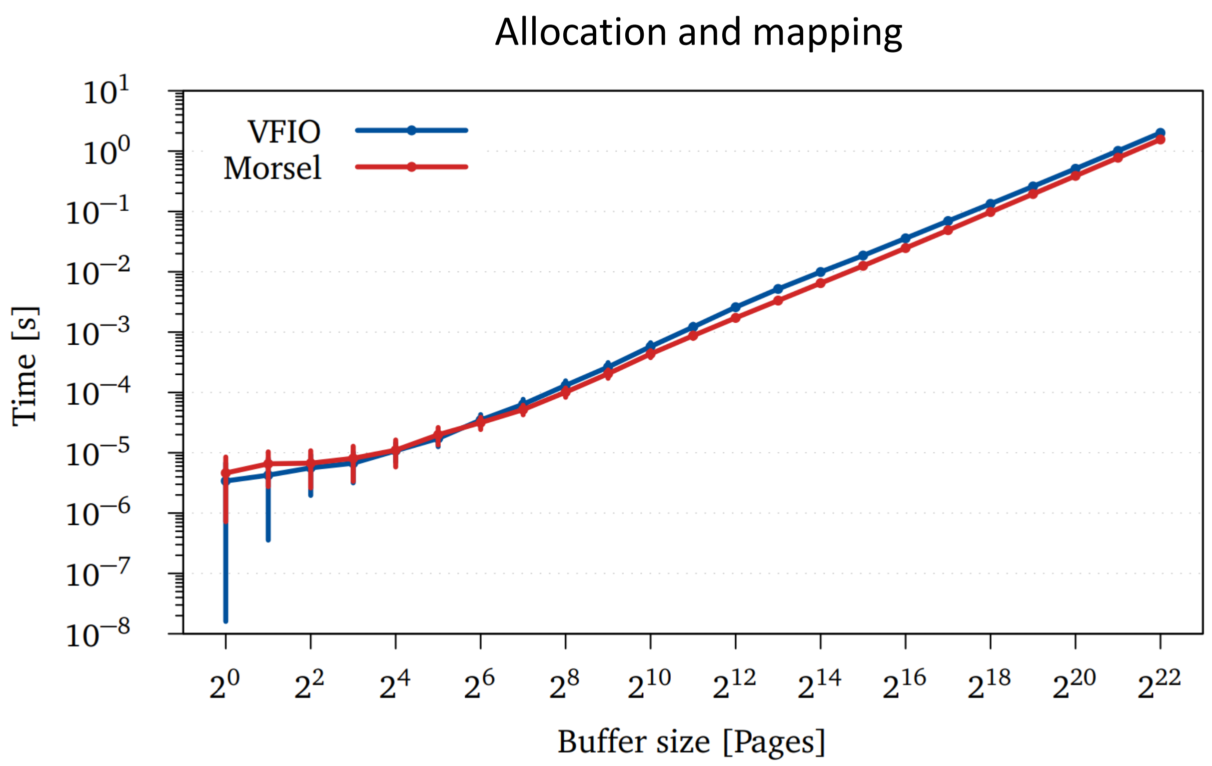
IOMMU subsystem in Linux 6.1

Evaluation

- Measurements taken on recent desktop system
- Mapping
- Unmapping
- Prototypical NVMe driver

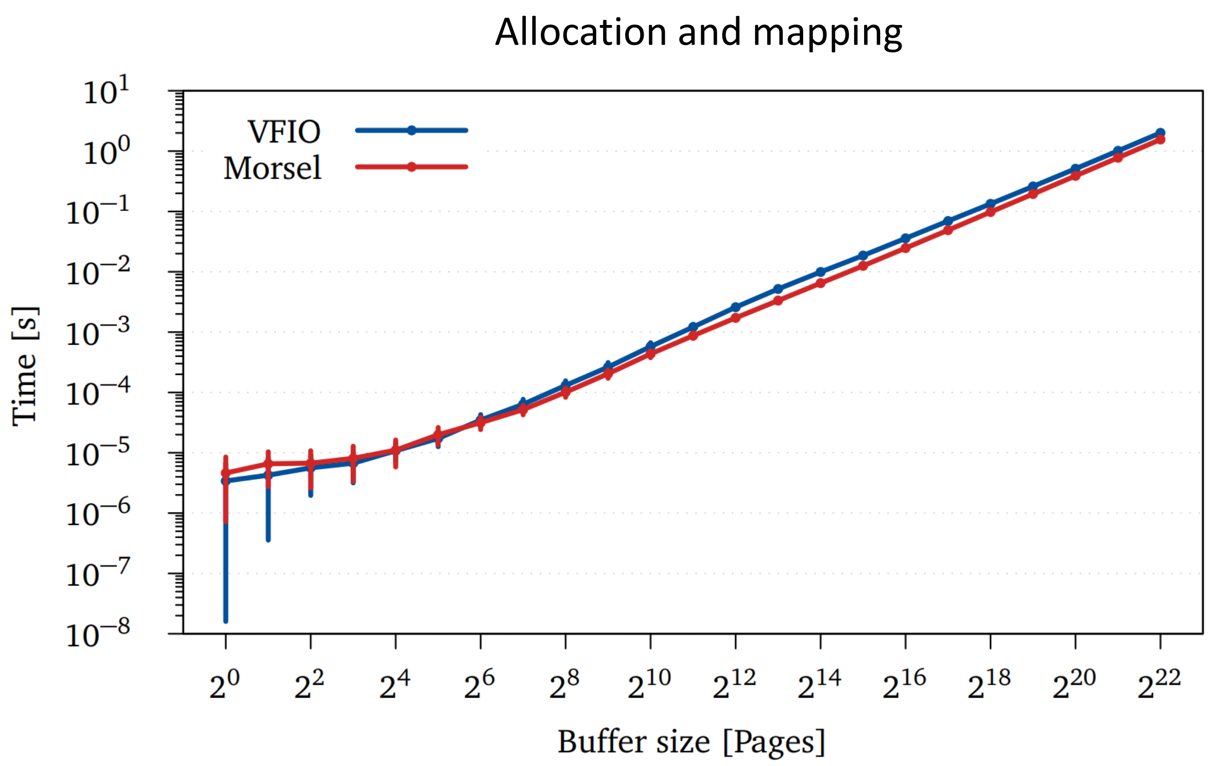
CPU	RAM	NVMe SSD
AMD Ryzen 7 PRO 5750G 8 Cores / 16 Threads @ 3.8 GHz	ADATA DDR4 32 GB @ 3200 MHz	Samsung 970 EVO Plus 1 TB 3.5 GB/s seq. read

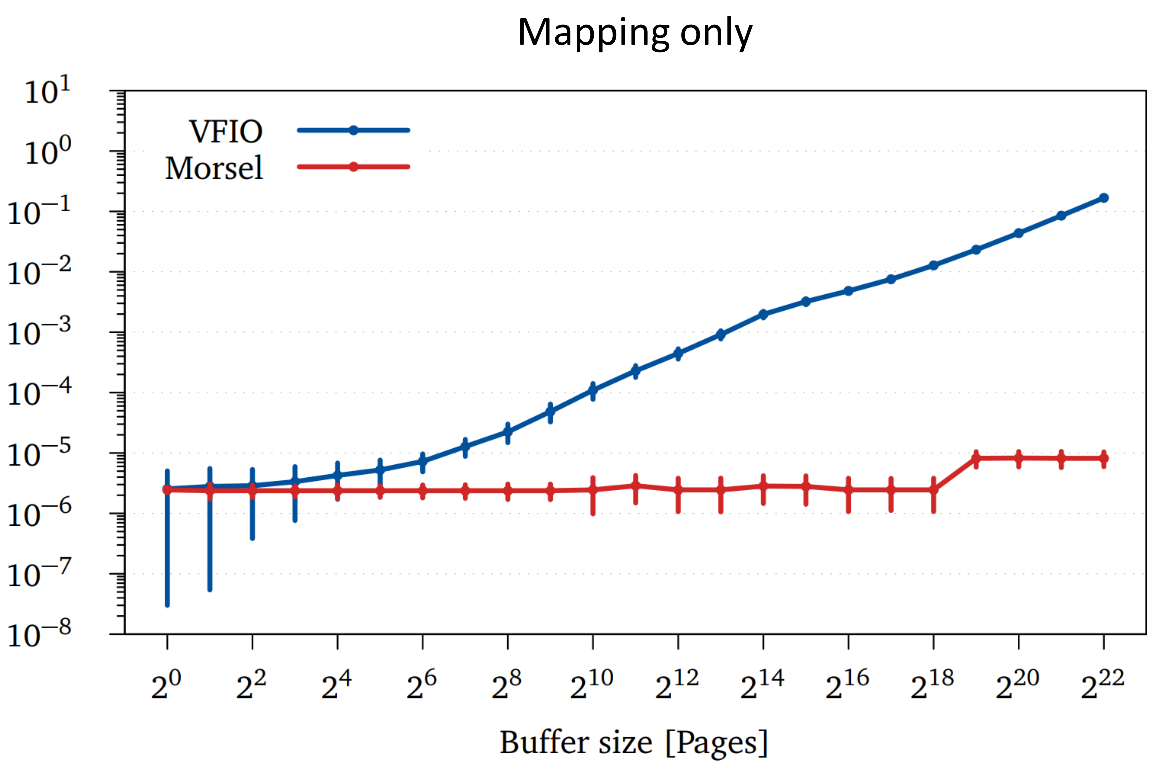
Evaluation – Mapping



Evaluation – Mapping

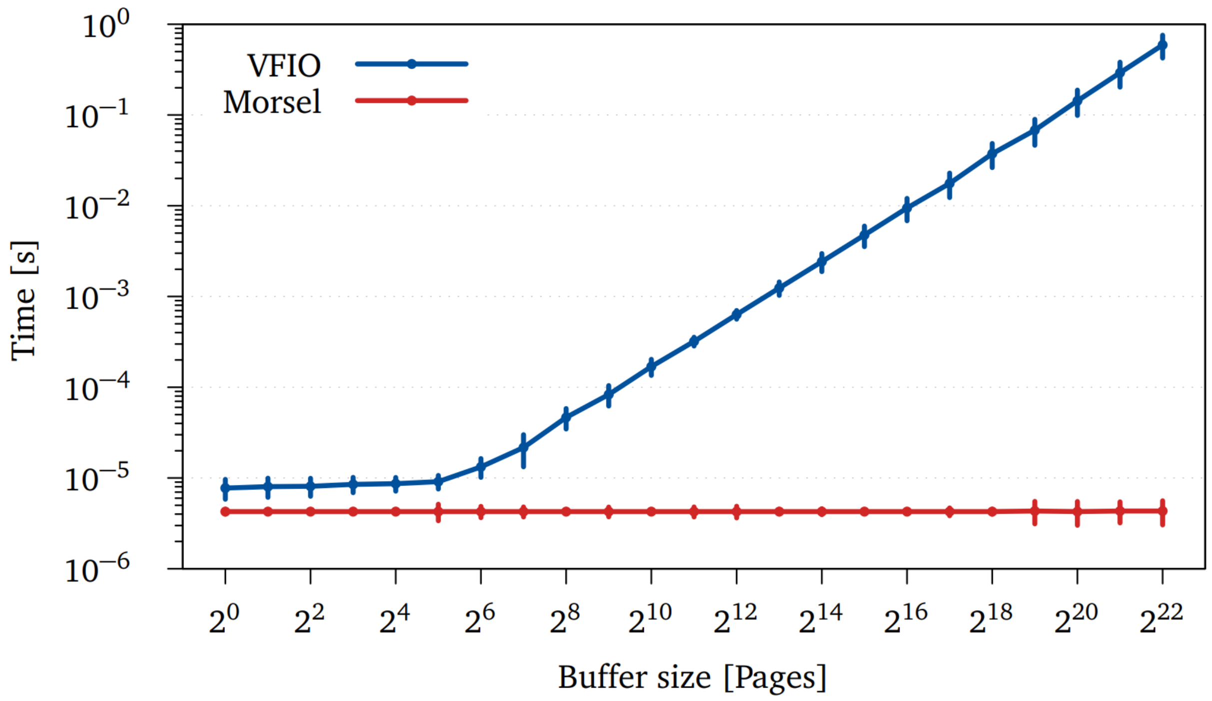
- Morsels compared to VFIO Buffers
  - VFIO: Runtime proportional to buffer size (~75% pinning, 25% table management)
  - Morsel: **Orders of magnitude faster** for large sizes due to constant runtime





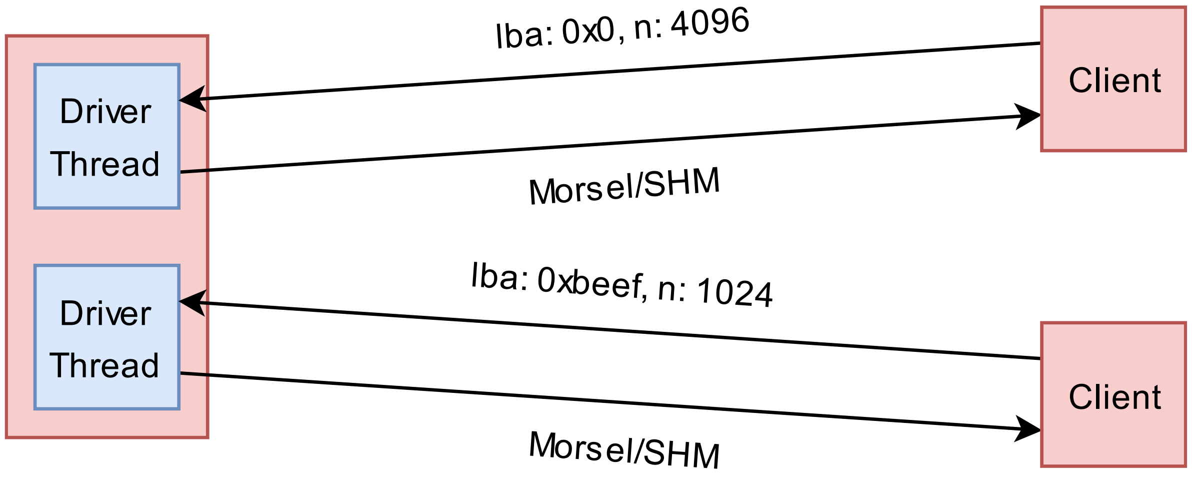
Evaluation – Unmapping

- Morsels compared to VFIO Buffers
  - VFIO: Runtime proportional to buffer size (due to unpinning)
  - Morsel: **Constant runtime**. IOTLB invalidation accounts for ~90%



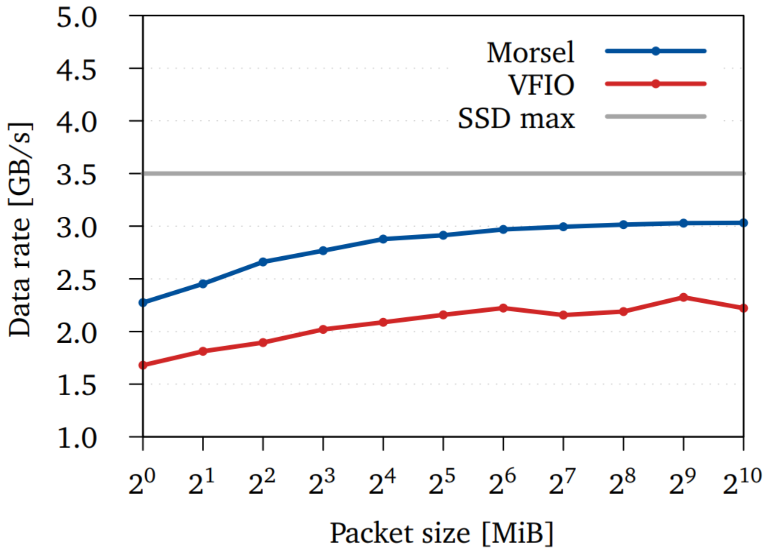
Evaluation – NVMe Driver

- User space NVMe driver
  - Driver process has exclusive SSD access → Isolation
  - Client request data
  - Driver reads data into Morsel and sends back reference
- Benchmark: Client-side data rate
  - Morsel vs. VFIO and Posix-SHM
  - Workload: Clients calculate checksum
  - Measurements for one and two clients



Communication between driver and clients

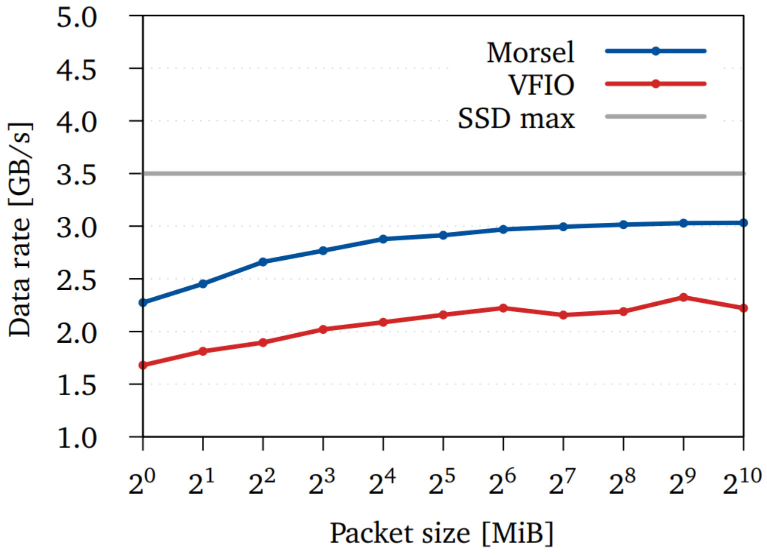
Evaluation – NVMe Driver 1 Client



With data transfer

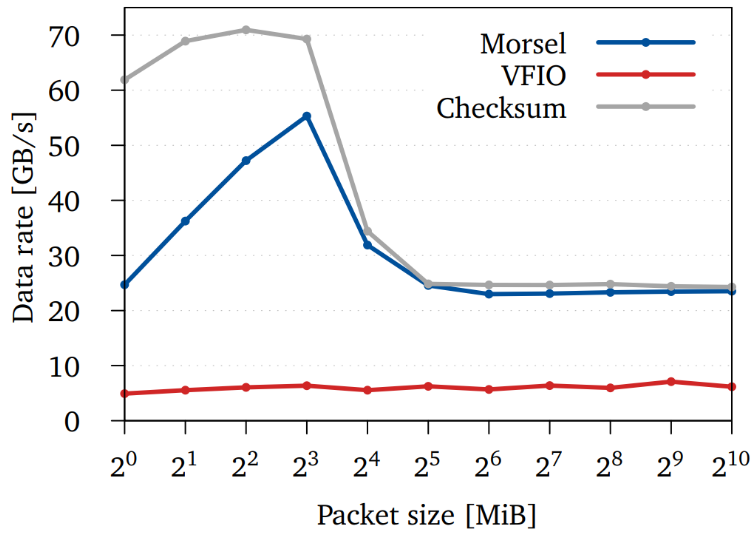
- Morsel: ~3 GB/s max
  - Does not reach SSD max, since packets are not prefetched
- VFIO: ~2.3 GB/s max
  - Overhead from memory management and additional `mmap()`

Evaluation – NVMe Driver 1 Client



With data transfer

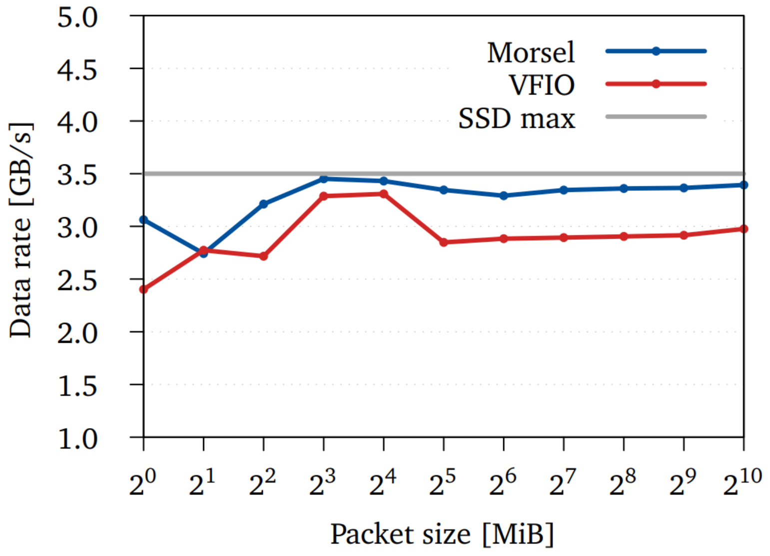
- Morsel: ~3 GB/s max
  - Does not reach SSD max, since packets are not prefetched
- VFIO: ~2.3 GB/s max
  - Overhead from memory management and additional `mmap()`



Without data transfer

- Morsel: ~23 GB/s
  - >16 MiB bottlenecked by checksum (memory bound)
  - Smaller packets profit from L_3 cache (up to 55 GB/s)
- VFIO: ~7 GB/s max

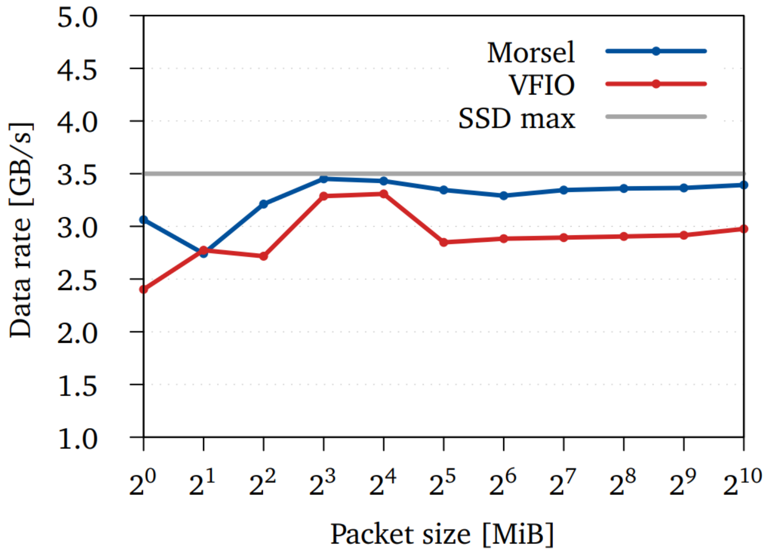
Evaluation – NVMe Driver 2 Clients



With data transfer

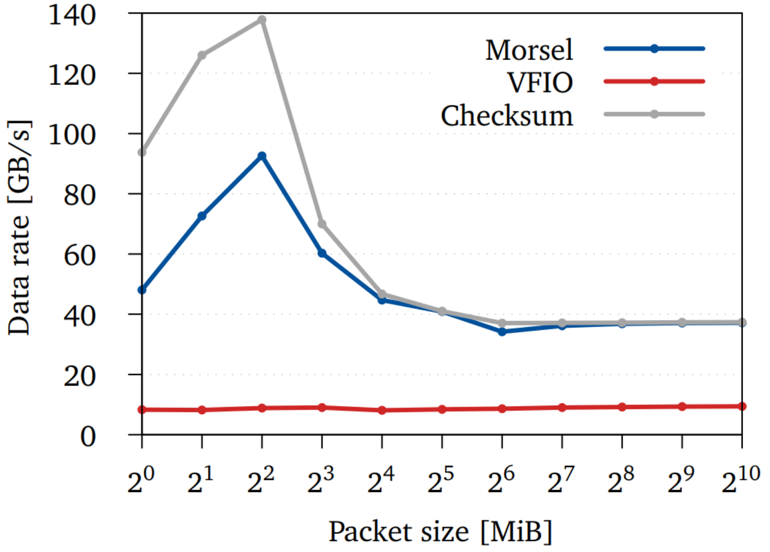
- Morsel: At SSD max
- VFIO: ~3 GB/s
  - Does not reach SSD limit due to lock contention around VFIO container

Evaluation – NVMe Driver 2 Clients



With data transfer

- Morsel: At SSD max
- VFIO: ~3 GB/s
  - Does not reach SSD limit due to lock contention around VFIO container



Without data transfer

- Morsel: Speedup x1.6-2.0
  - >16 MiB bottlenecked by checksum (memory bound)
  - Smaller packets profit from L_3 cache (up to 95 GB/s)
- VFIO: <10 GB/s
  - Scales poorly

Summary

- Modern memory management must account for external devices
  - Tradeoff: Speed vs. Isolation
- Extended Morsels to the IOMMU
  - Efficient memory sharing with devices
  - (Modified) page table subtrees are directly shared between MMU and IOMMU
  - Implementation for Linux 6.1
- Evaluated performance characteristics
  - (Un)mapping on IOMMU in constant time
    → High throughput while enforcing isolation (~23 GB/s in NVMe bench)
  - More flexible than VFIO buffers: Don't need to be mapped on the MMU-side first

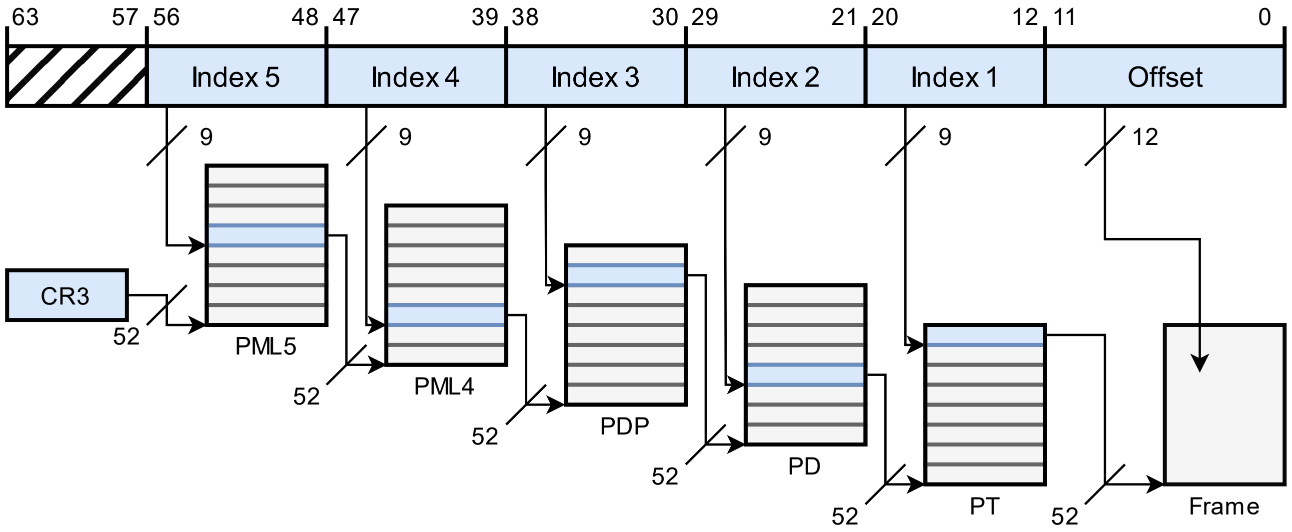
Summary

- Modern memory management must account for external devices
  - Tradeoff: Speed vs. Isolation
- Extended Morsels to the IOMMU
  - Efficient memory sharing with devices
  - (Modified) page table subtrees are directly shared between MMU and IOMMU
  - Implementation for Linux 6.1
- Evaluated performance characteristics
  - (Un)mapping on IOMMU in constant time
    → High throughput while enforcing isolation (~23 GB/s in NVMe bench)
  - More flexible than VFIO buffers: Don’t need to be mapped on the MMU-side first
- Future work
  - Support for Intel IOMMUs
  - Evaluating ARM SMMUv3 support
  - Driver for an accelerator (FPGA?) → Morsel-Pipeline

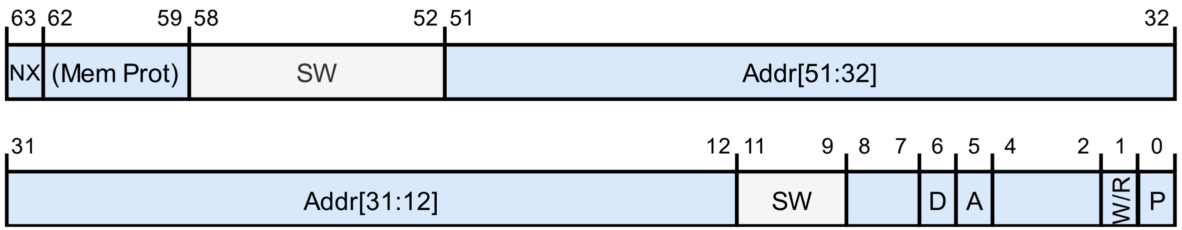
Backup Slides

AMD64 Long Mode Paging

- 4KiB page (frames)
  - Optional *Huge Pages* of 2MiB und 1GiB
- 48/57 Bit virtual → 52 Bit physical
  - 4 or 5 level paging
- Access rights at each level (r/w/x)
  - Restrictions are propagated downwards
- Missing entries/access rights result in a page fault
- Translation happens on Memory Management Unit (MMU)
  - Previous results will be cached inside *Transaction Lookaside Buffer (TLB)*



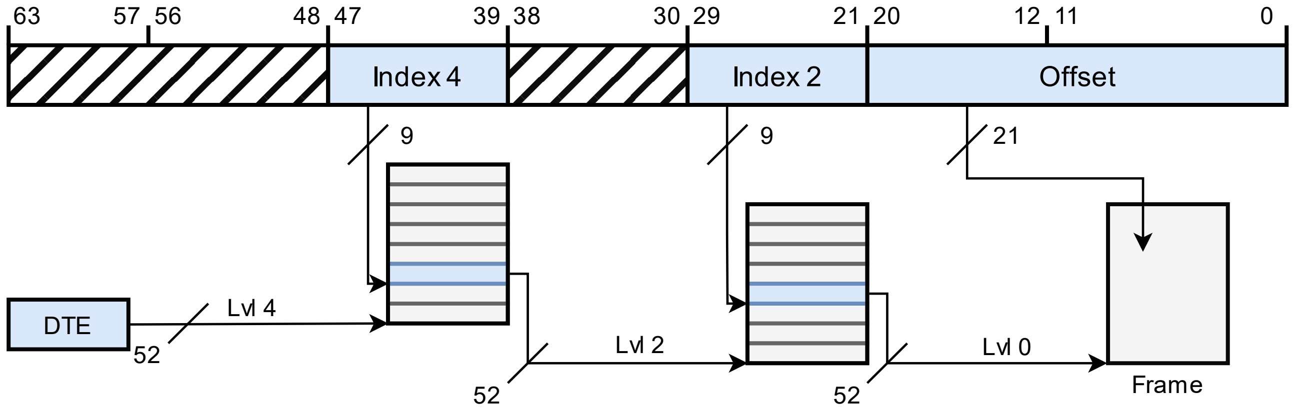
5 level address translation



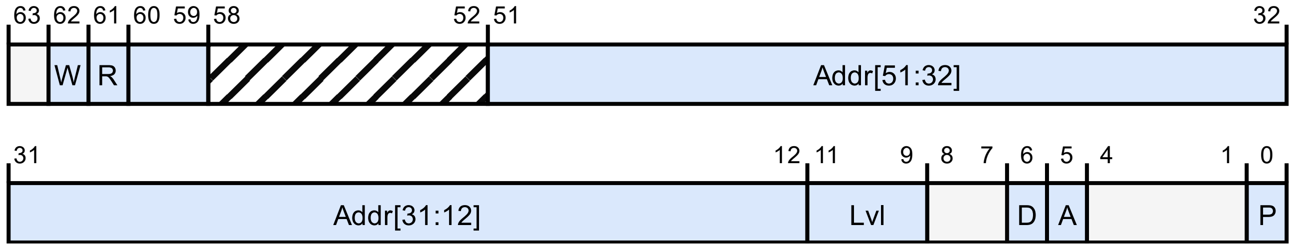
Simplified page table entry

AMD IOMMU (AMD-V) – Address translation

- 6-Level-Paging → 64bit address space
- *Level Bits* encode level of the next table
  - Skipping levels
  - Huge pages by terminating early (lvl=0)
- Separate r/w rights → *write-only*
  - Restrictions are propagated downwards
- Format compatible with MMU



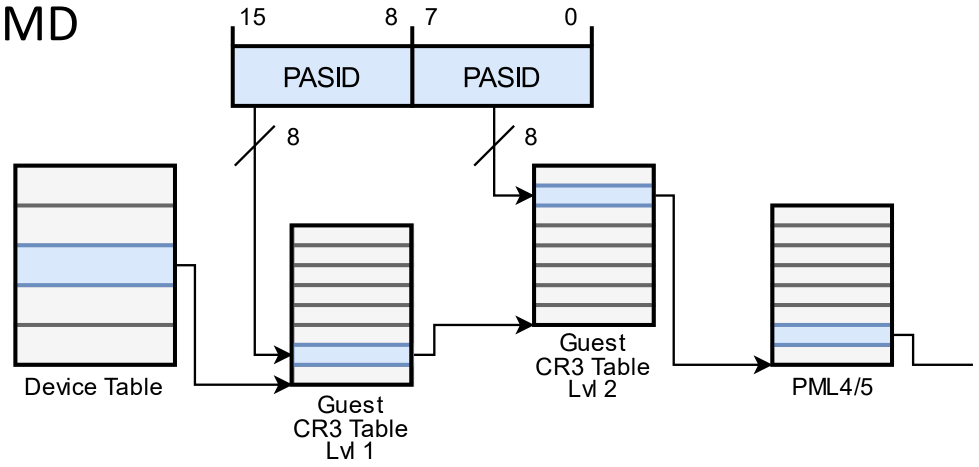
Address translation (example)



Simplified page table entry

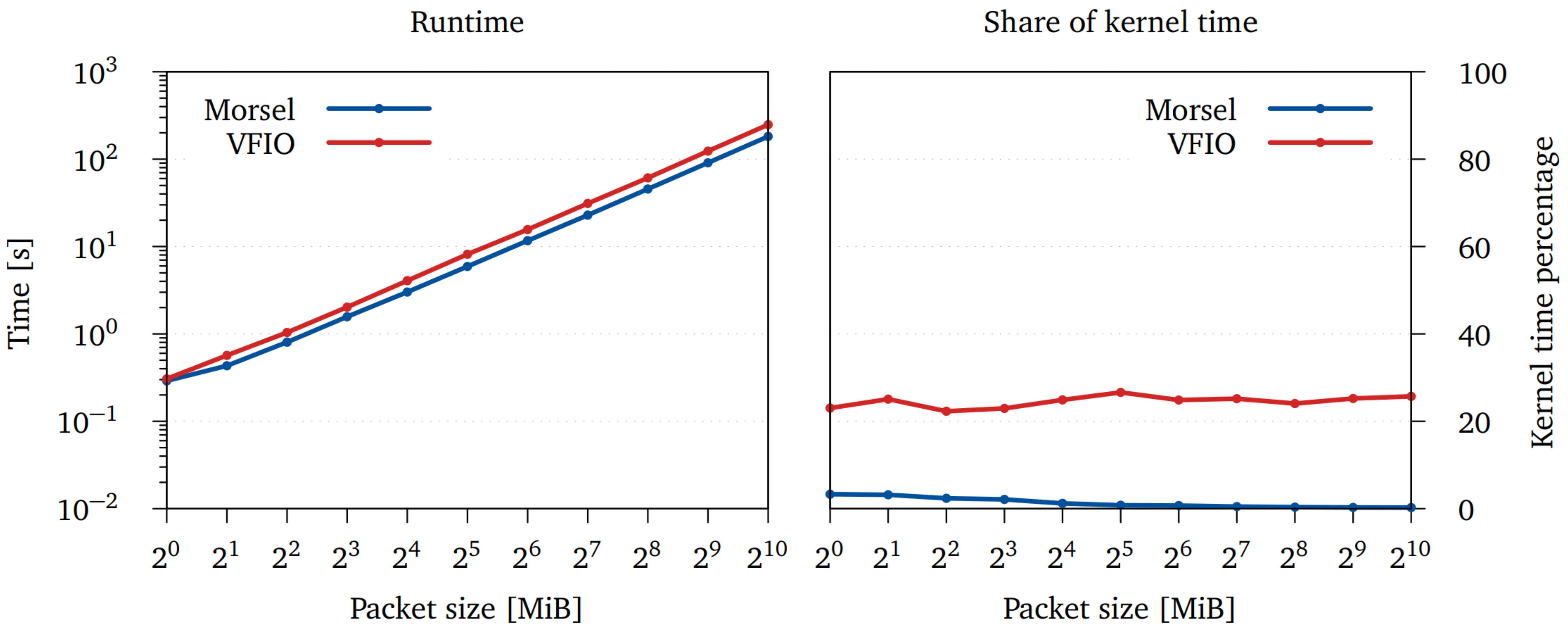
Extending Morsels – Alternative Implementations

- Exploiting the guest translation
  - Sharing morsel tables as guest tables
  - Guest tables use MMU format for both Intel and AMD
  - Features often not available
  - Address translation is more costly
- Separate page tables
  - One set of page tables per (IO)MMU
  - Supports incompatible table formats
  - May exploit special hardware capabilities
  - Added overhead in page fault handler
  - Adds extra state to Morsel

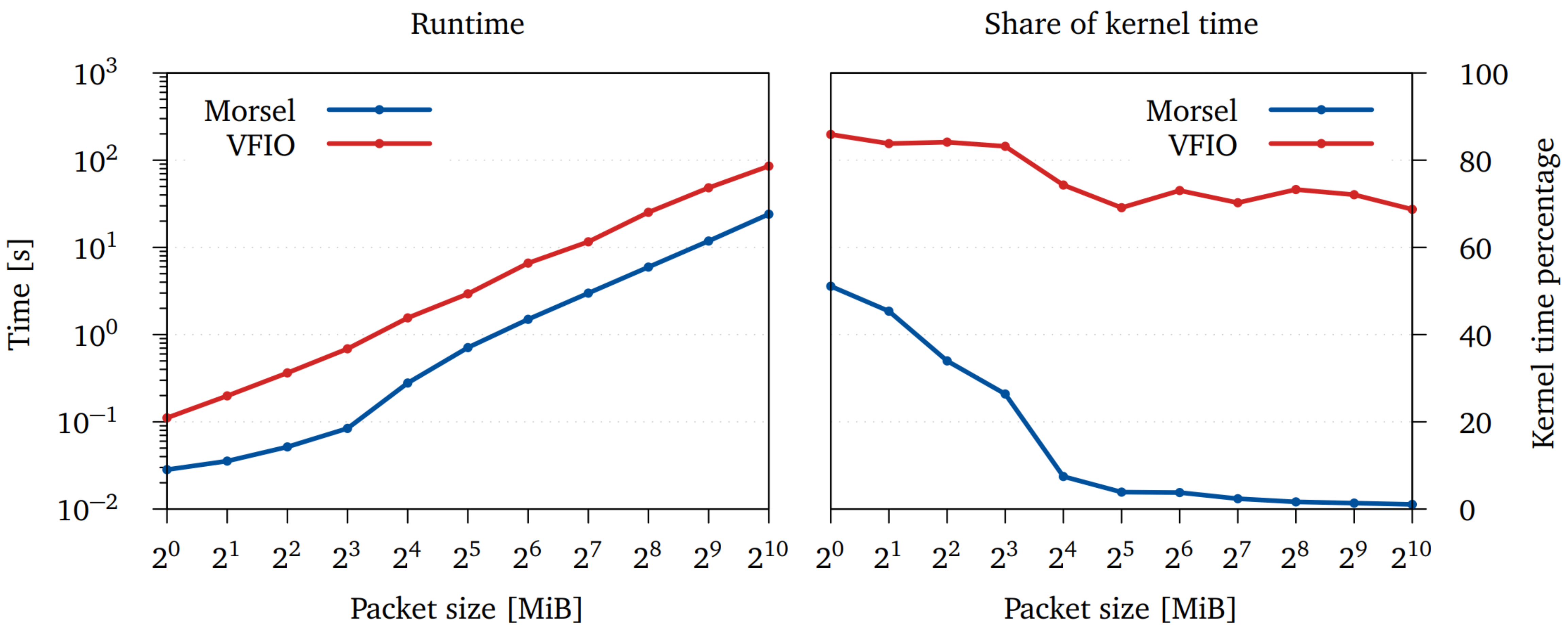


Extra indirections when using guest translation

Evaluation – NVMe Driver Kernel Times (Data Transfer)



Evaluation – NVMe Driver Kernel Times



Sources

[1] Author: D-Kuru/Wikimedia Commons, Licence: CC-BY-SA-4.0

[2] Author: Dmitry Nosachev/Wikimedia Commons, Licence: CC-BY-SA-4.0

[3] Alexander Halbuer et. al. „Morsels: Explicit Virtual Memory Objects“. In: Proceedings of the 1st Workshop on Disruptive Memory Systems. DIMES ’23. New York, NY, USA: Association for Computing Machinery, 2023, P. 52–59. ISBN: 9798400703003. DOI: 10.1145/3609308.3625267. URL: https://doi.org/10.1145/3609308.3625267.