VM_PFNMAP VMAs and guest_memfd//HugeTLB

For 2025-08-07 guest_memfd bi-weekly upstream call

Contact ackerleytng@google.com if you have questions/suggestions!

Idea overview

- What if we take folios from HugeTLB
- But tell core-mm to assume folio don’t exist with VM_PFNMAP?

About VM_PFNMAP

- “Page-ranges managed without "struct page", just pure PFN”
  - Virtual memory subsystem should not go looking for a struct page [1]
- Even in cases where that structure does exist (such as remappings of real memory), the VM code will pretend that it does not. [1]

[1] https://lwn.net/Articles/162860/

How it would work

- mmap handler
  - Set `VM_PFNMAP` for VMA for the entire VMA range that requests guest_memfd
- Fault handler
  - Fill in process page tables directly (i.e. call `vmf_insert_pfn()`)
  - Return `VM_FAULT_NOPAGE`
- Stage 2 page table fault handler `kvm_gmem_get_pfn()`
  - Compute mapping level based on shareability and return that
  - No page splitting
- Memory can remain `mmap()`-ed like now

Shared to private conversion

- Unmap from process page tables
- Unmap from stage 2 page tables
- No folio merging

With VM_PFNMAP, no more folio restructuring

- No restructuring (split/merge) on conversions
- No more conversion refcounting issues
  - David: Transient refcounting still exists
  - Sean: Will be re-creating refcounting issue if we want to support nested virtualization
- Folios no longer outlive guest_memfd (core-mm assumes no folios)
  - Cleanup can be done on inode release
    - No need to merge folios after guest_memfd closes
    - No guest_memfd page type flag applied temporarily
- No restructuring to race with in memory failure handling
  - (may still need custom handling)
- No more runtime vmemmap restore and optimize
  - No need to reserve memory for overheads when restoring vmemmap optimization
- Runtime restructuring cost goes away

Did not discuss the slides after this one

GUP and IO

- IO is already being offloaded and handled via VFIO
- `vfio_dma_do_map()` can already map VM_PFNMAP memory
  - (`iommufd_ioas_map()` cannot)
- Interoperability: guest_memfd needs a way to force unmapping from IOMMU to ensure devices don’t write to private memory

Limitation: Cannot GUP from VM_PFNMAP VMAs

- GUP use cases?
- What are some GUP use cases that cannot be replaced by other mechanisms?
- Can we re-frame the folio restructuring problem as a guest_memfd-with-VM_PFNMAP-interoperability problem and focus on enlightening other users of guest_memfd?

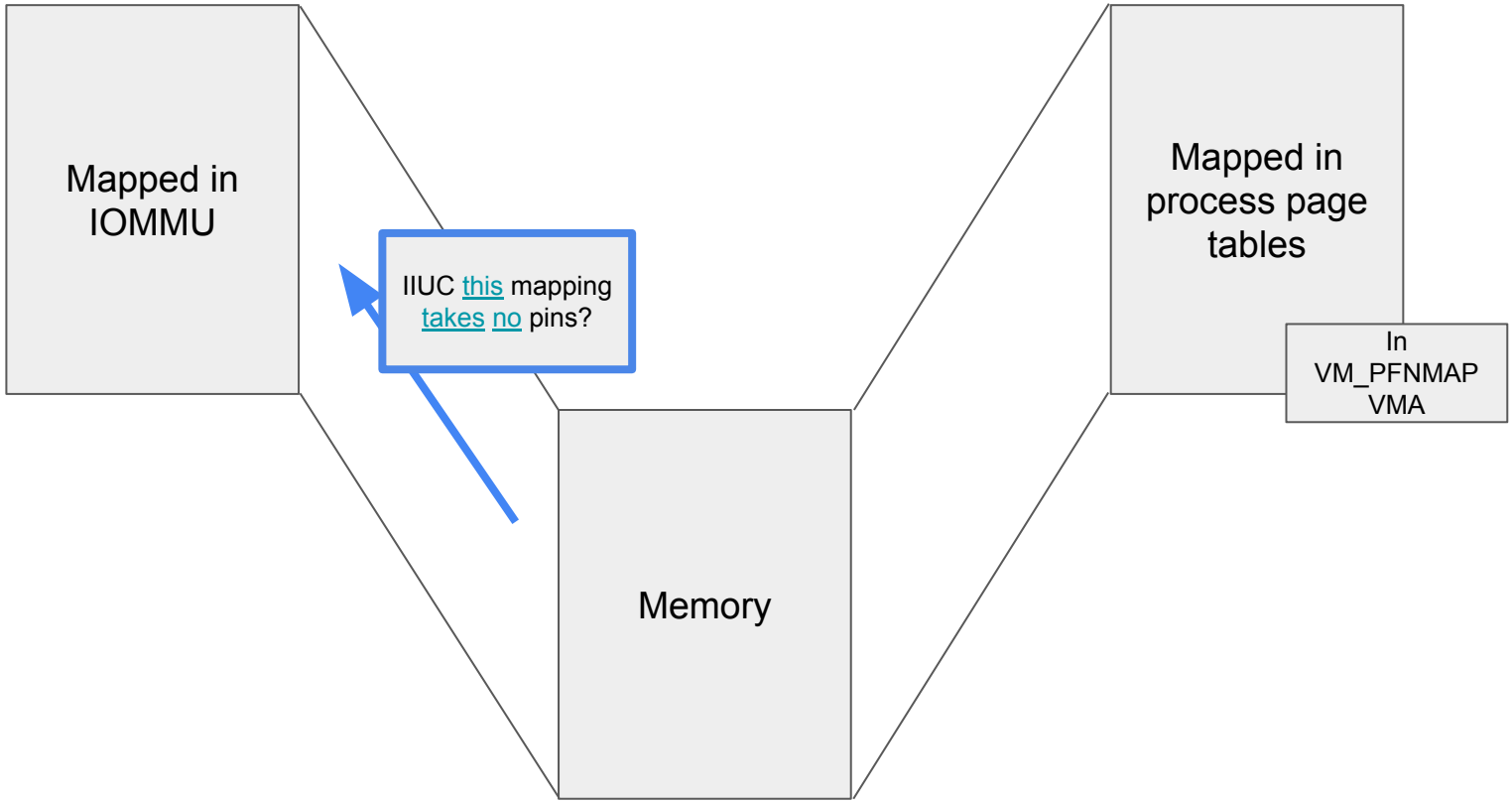
Longer term benefits

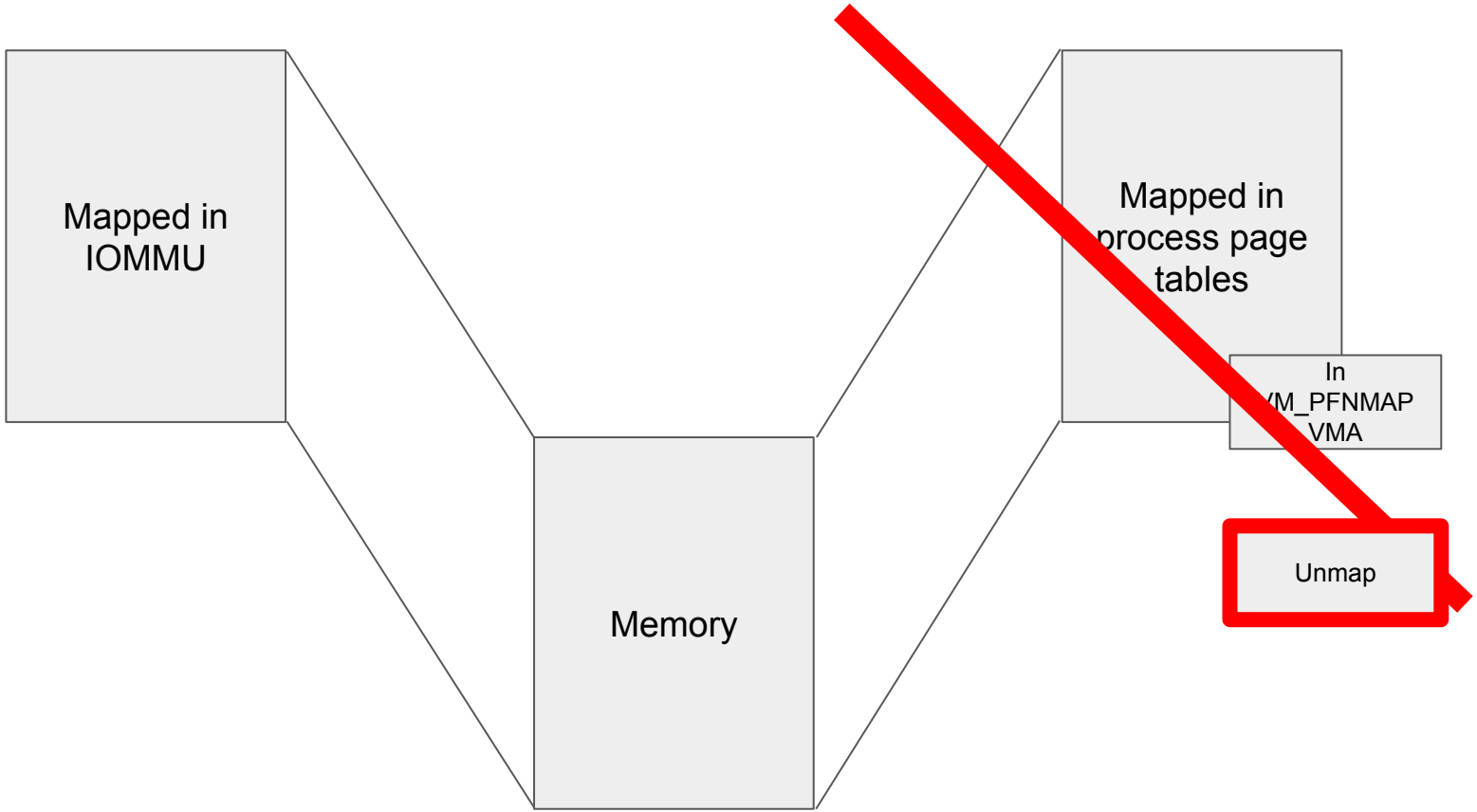
- Brings guest_memfd closer to being a “primary” mmu [1] since it also now manages process page tables
  - Can handle mapping in userspace page tables, even at 1G, without taking dependency on “making HugeTLB less weird” in core-mm
- Easier to move away from `struct folio`s in future

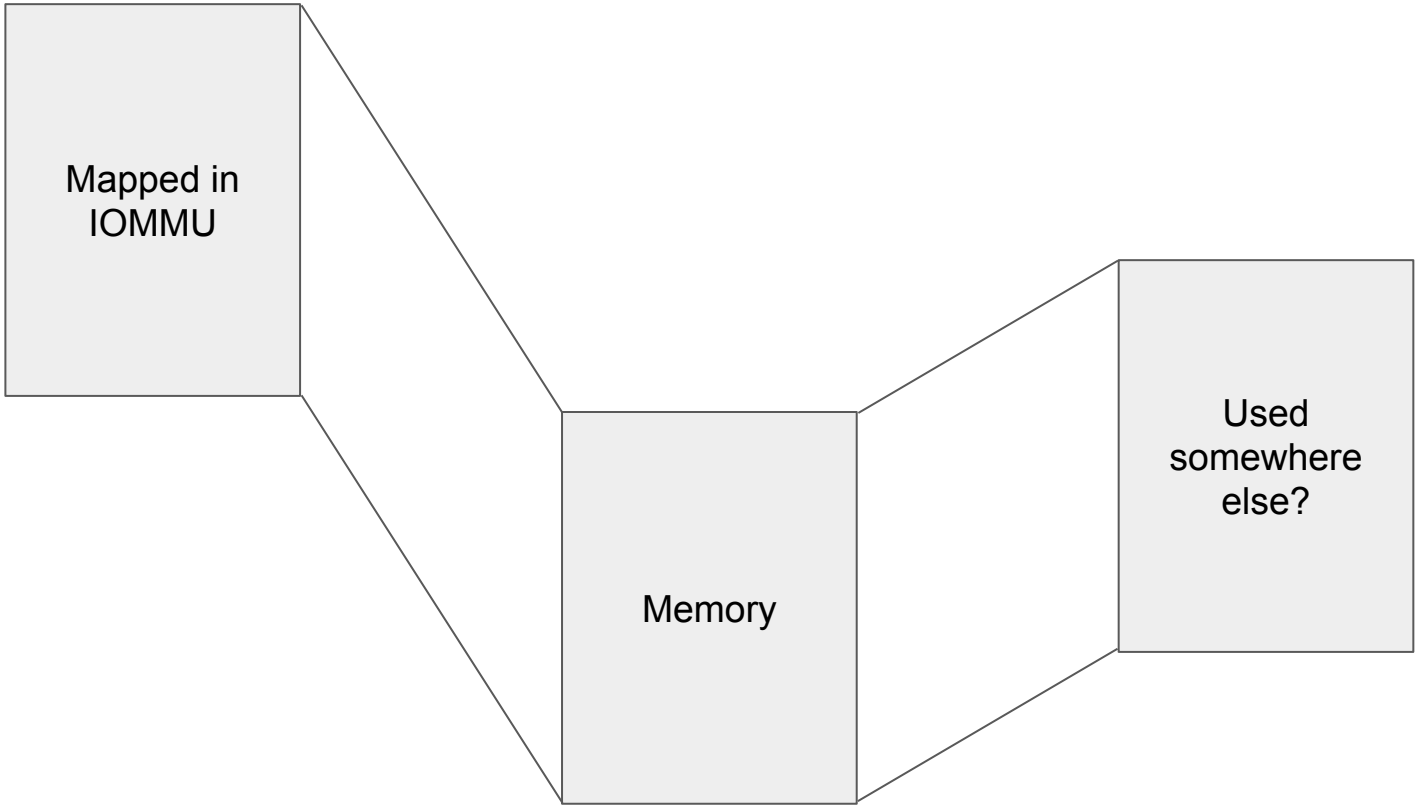
[1] Sean: https://lore.kernel.org/all/aIP-qSnH1jjuykmP@google.com/

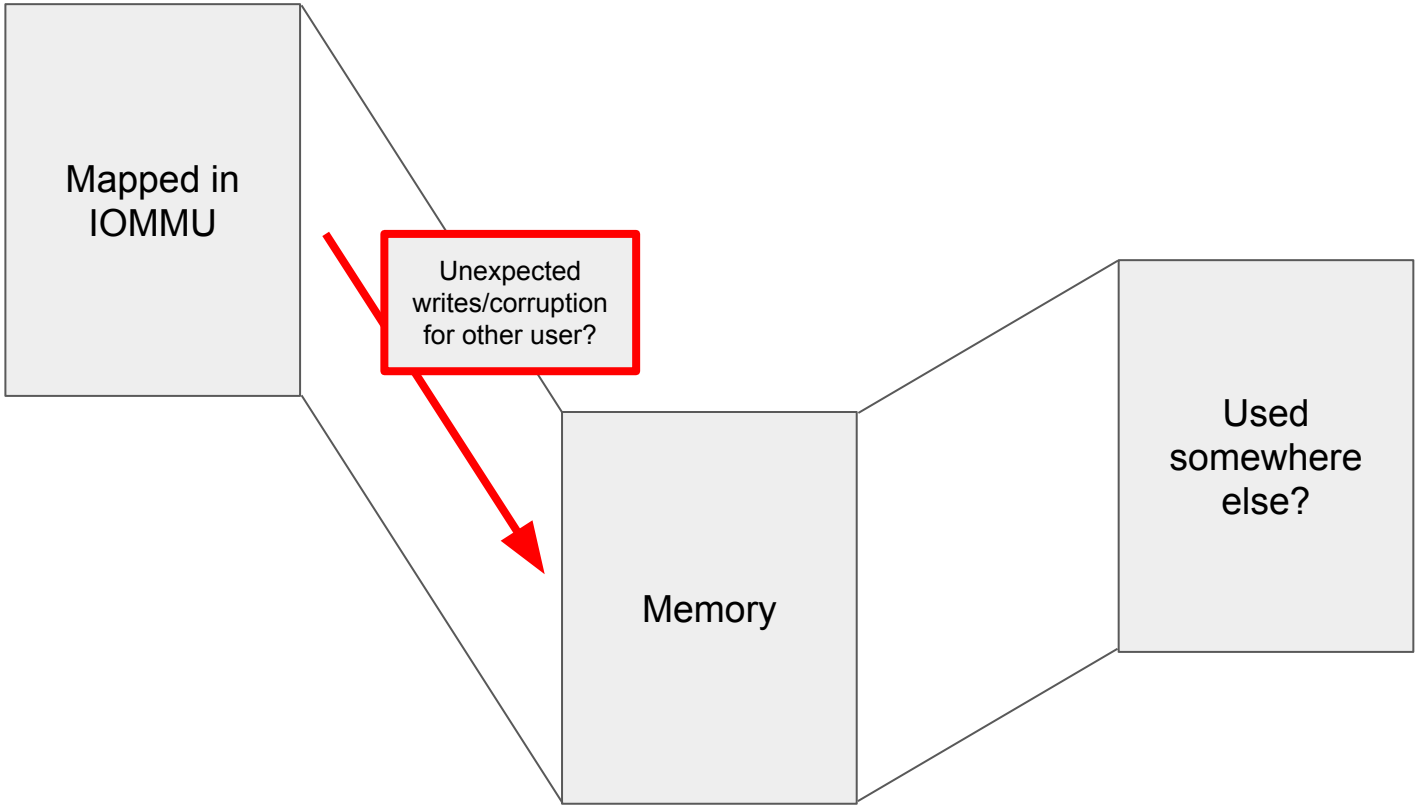
Thank you!

Possible Data Corruption in VFIO?


Mapped in IOMMU
IIUC this mapping takes no pins?
Memory
Mapped in process page tables
In VM_PFNMAP VMA


Mapped in IOMMU
Memory
Mapped in process page tables
In VM_PFNMAP VMA
Unmap


Mapped in IOMMU
Memory
Used somewhere else?


Mapped in IOMMU
Unexpected writes/corruption for other user?
Memory
Used somewhere else?

Smallest changes tested to work

I could boot a VM with Fuad's kvmtool instructions

- Add this in guest_memfd's mmap handler

```
vm_flags_set(vma, VM_PFNMAP);
```

- Change this in guest_memfd's fault handler

```
// vmf->page = folio_file_page(folio, vmf->pgoff);
vmf_insert_pfn(vmf->vma, vmf->address, folio_file_pfn(folio, vmf->pgoff));
ret = VM_FAULT_NOPAGE;
```