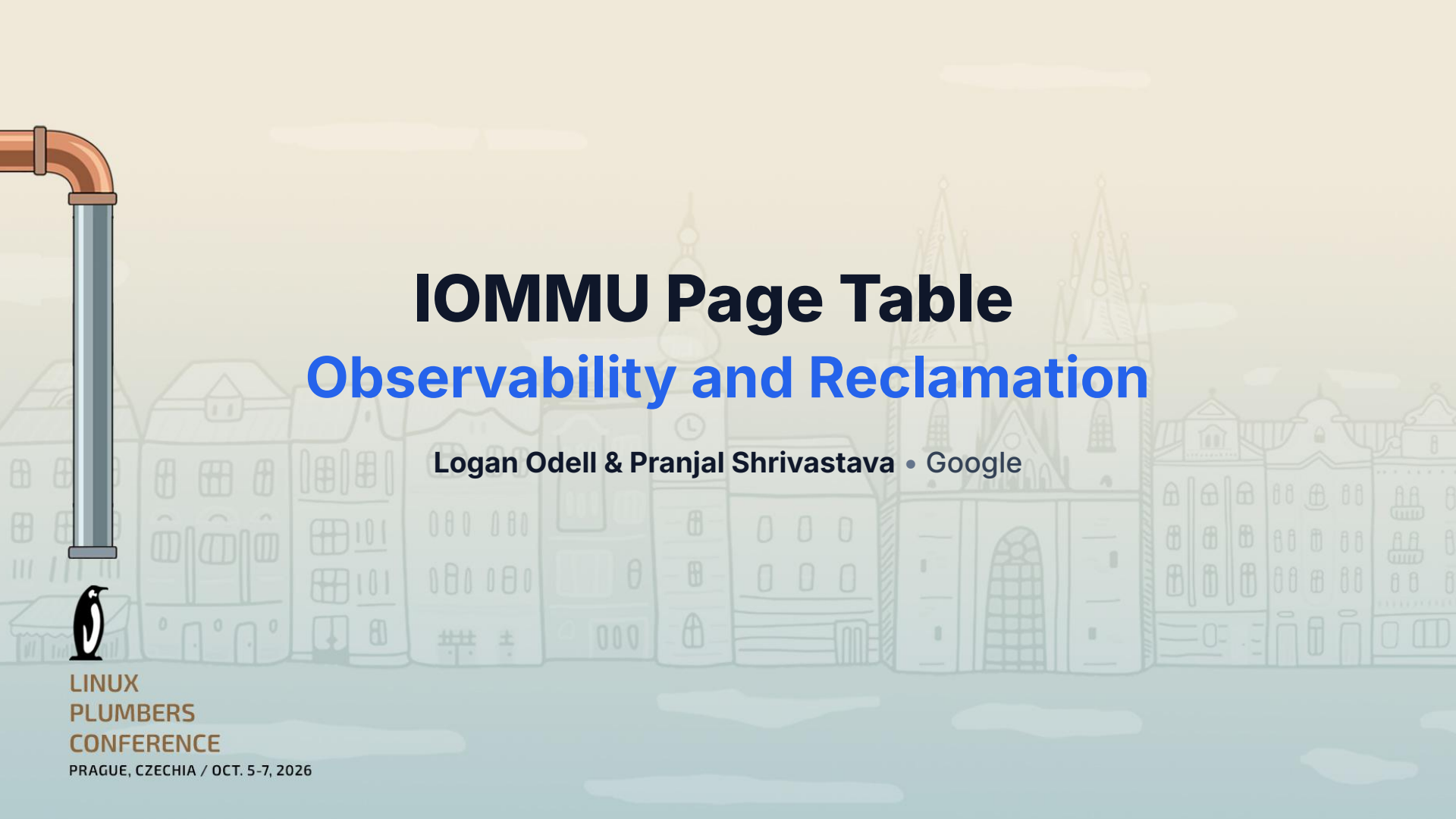




IOMMU Page Table Observability and Reclamation

Logan Odell & Pranjal Shrivastava • Google



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

The Invisible Cost of Sparse IOVAs

What controls IOVA allocation?

Kernel drivers (DMA API)

- IOVA allocator (rbtree / mtree) + per-CPU caches
- Fast reuse keeps allocations tight
- Minimal page-table memory overhead

Userspace* (via VFIO / IOMMUFD)

- The VMM, guest or device driver picks the IOVAs
- Kernel has no say: it simply calls `iommu_map()`
- Arbitrary allocations cause sparse churn

* There are in-kernel users of the IOMMU API as well

The Unmap Problem

- **Leaf invalidation only**
 - Unmap clears the leaf PTEs
 - The tables holding PTEs remain allocated
- **No emptiness check**
 - A table is freed only if a single unmap covers its entire range
- **No recursive unwind**
 - Tables emptied by partial unmappings, and their parents, stay in memory

The Impact: Invisible OOMs

- **Host OOM:** Poorly managed sparse IOVA allocations $\Rightarrow$ OOMs
- **Zeroed directories:** stranded tables hold nothing but zeroed entries
- **Stranded gigabytes:** sequential sparse unmappings leave gigabytes of page tables behind
- **Silent leaks**
 - IOMMU pages counted only as a global total
 - No owner to blame
 - No reclaim



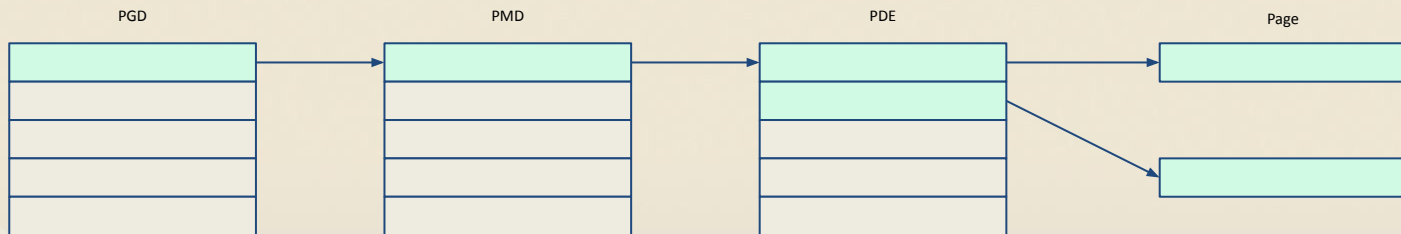
LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Problem Statement

Stranded Memory due to Sparse IOVA Allocations



Tight allocations, unmapping one of the two pages leaves no additional overhead.



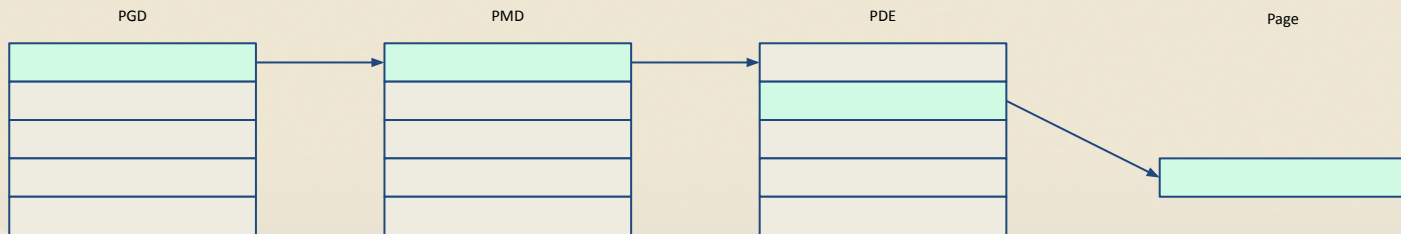
LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Problem Statement

Stranded Memory due to Sparse IOVA Allocations



Tight allocations, unmapping one of the two pages leaves no additional overhead.



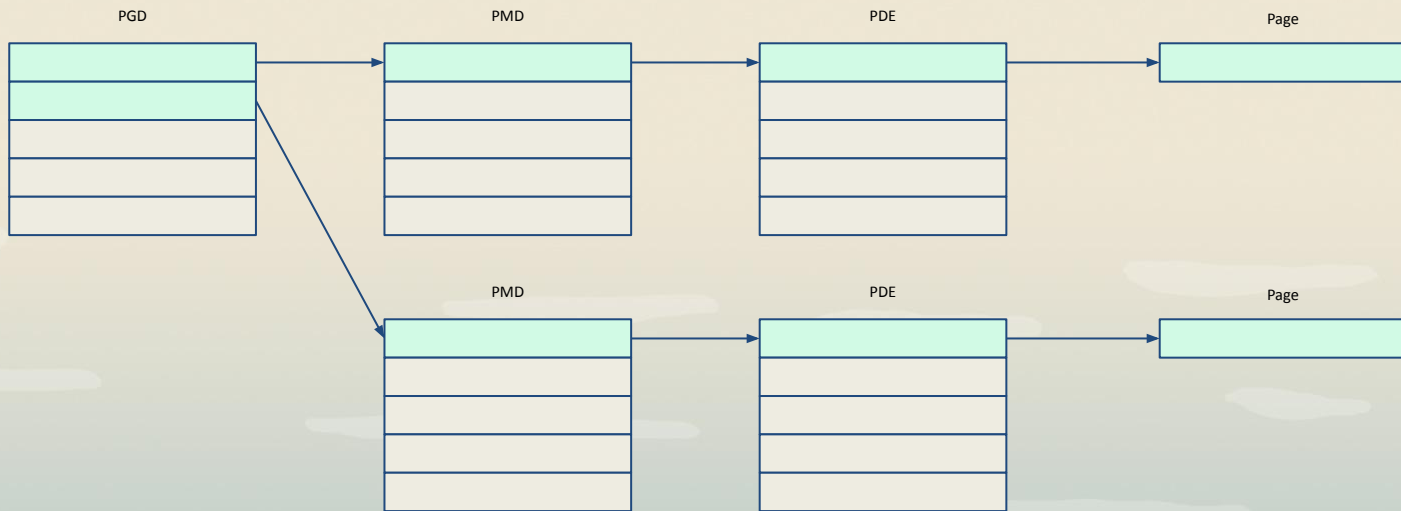
LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Problem Statement

Stranded Memory due to Sparse IOVA Allocations



Sparse allocations, unmapping one of the two pages leaves empty intermediate page tables



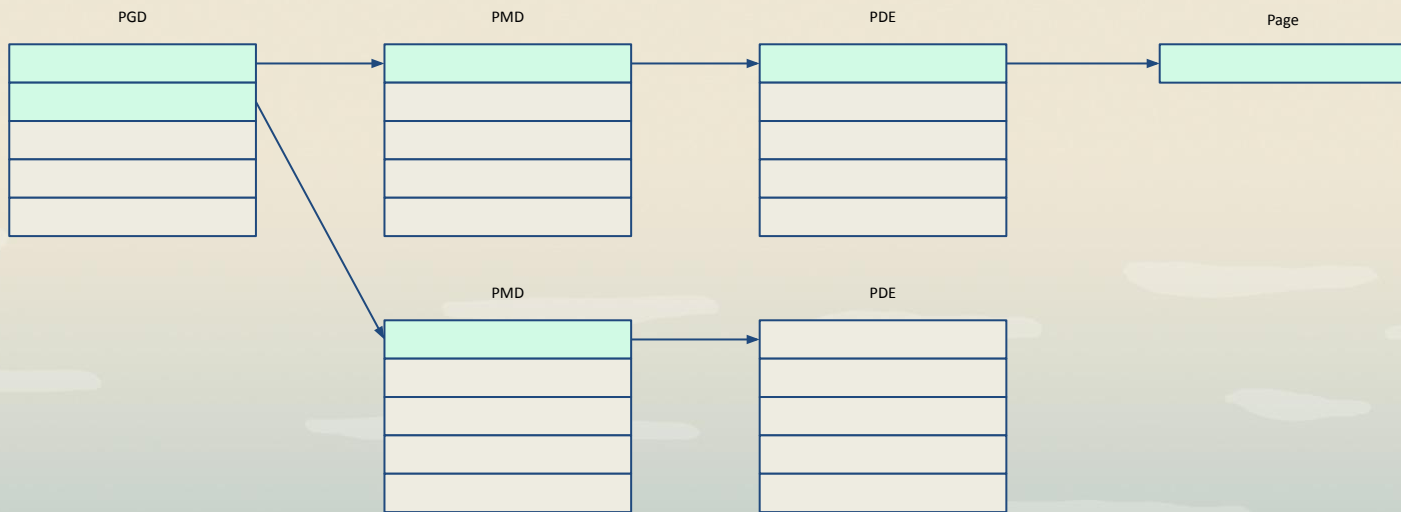
LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Problem Statement

Stranded Memory due to Sparse IOVA Allocations



Sparse allocations, unmapping one of the two pages leaves empty intermediate page tables



LINUX
PLUMBERS CONFERENCE

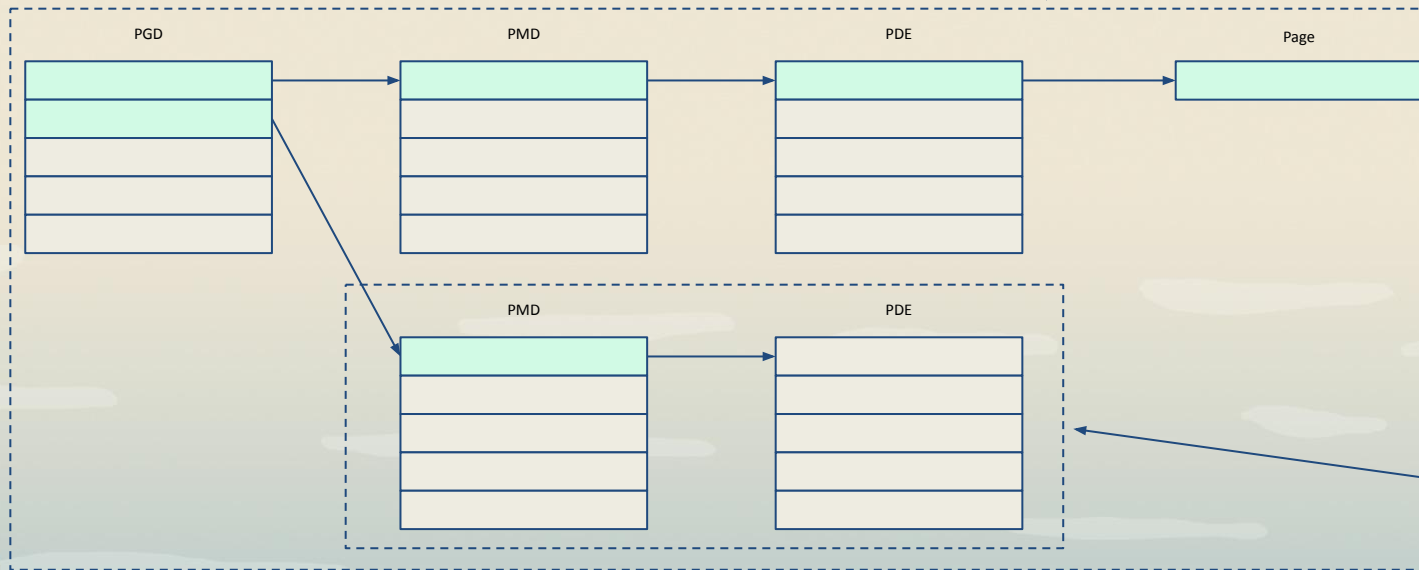
PRAGUE, CZECHIA / OCT. 5-7, 2026



Problem Statement

Stranded Memory due to Sparse IOVA Allocations

1. Who's using this memory?



2. How much of this memory is out there?

Sparse allocations, unmapping one of the two pages leaves empty intermediate page tables



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



SECTION 01 • OBSERVABILITY

Bridging the Blind Spot

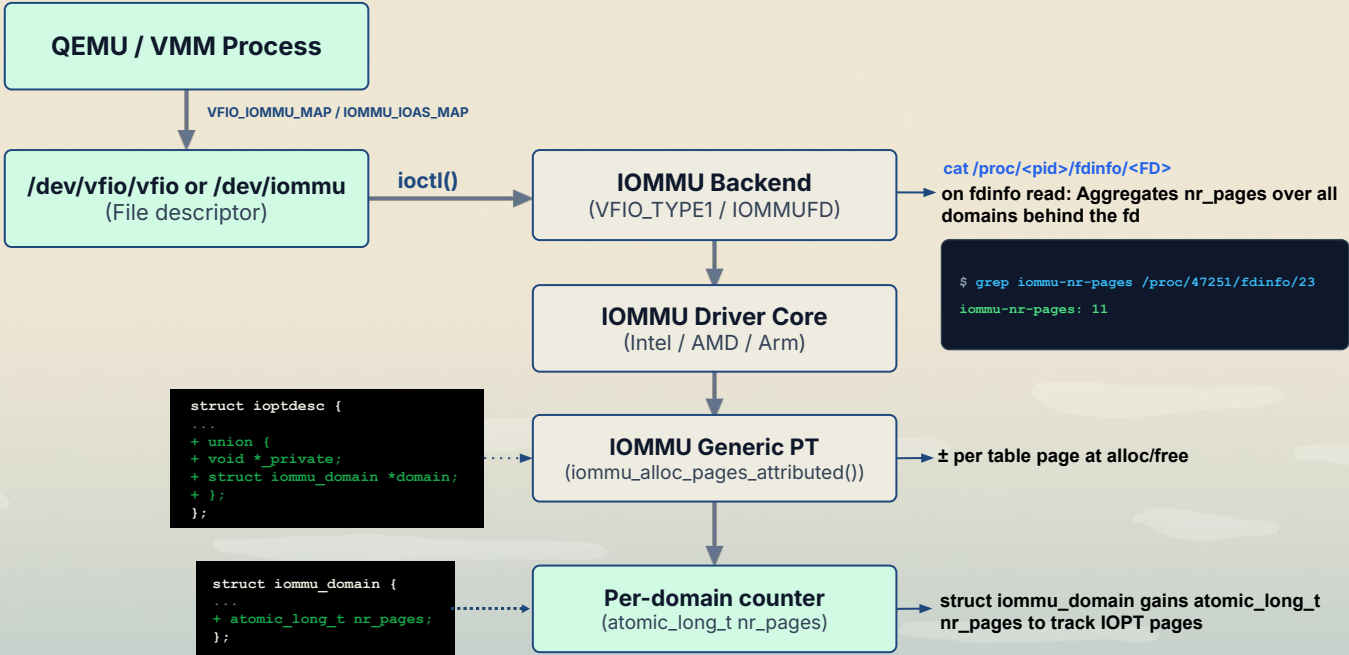
The Path to IOPT Observability



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

IOPT Observability Framework



SECTION 02 • RECLAMATION

Is Observing Enough?

Designing an IOPT Shrinker



Requirements for IOPT Reclamation

⚡ Minimal Fast Path Penalty

Map/unmap stay $O(1)$ per entry. No reverse maps, tree walks, or synchronous reclaim in the hot path.

⬆ SMP Cache Line Scalability

Minimize cache-line bouncing on shared upper directories (L1/L2) when concurrent map/unmap target independent IOVA ranges.

🔒 Lock-free Hot Path

Prune empty directories without domain-wide locks on map/unmap. No UAF or races against concurrent map/unmap or IOMMU walk caches.

📐 Arch & Geometry Agnostic

Avoid tight coupling with hardware translation layouts. Interface remains generic, supporting arbitrary map sizes, leaf levels, and directory geometries.



IOMMU Page Table Shrinker Architecture

RFC: lore.kernel.org/all/20261001230219.818128-1-praan@google.com/

Fast Path: Eager Refcounting

- **Atomic Refcount for Leaf directory pages:**
 - Refcount ONLY leaf-directory pages during a page table walk
 - Overload the native page→_refcount (aliased via the **ioptdesc**)
- **Deferred Free:** If the L4 refcount drops to exactly 1. Log the page and its IOVA into a per-domain Xarray.
- **Minimal penalty:** The hot path simply updates refcount and moves a pointer in an xarray infrequently.

Reclamation: Deferred MM Shrinker

- **Async Trigger** via the core MM Shrinker API.
- **State Claim (cmpxchg to 0):**
 - Pulls an empty directory page from the Xarray
 - Sets refcount = 0 **atomic_cmpxchg(&refcount, 1, 0)**.
- **Leaf Severance**
 - **cmpxchg** on the L3 parent slot to write 0x0
 - Severs the dead branch from the live page table tree.
- **RCU Synchronization**
 - Waits for any transient threads via **synchronize_srcu()**
- **Refcount restore:** Restores page's refcount=1 before freeing it to buddy.





Recount-based State Machine & Race Mitigation

STATE 1: ACTIVE DIRECTORY

Recount > 1

Directory contains active, valid leaf PTEs. Incremented on map, decremented on unmap.

STATE 2: STRANDED CANDIDATE

Recount == 1

All leaf entries are zeroed (invalidated). The L4 page is logged into the per-domain Xarray.

STATE 3: LOGICALLY DEAD

Recount == 0

The Shrinker transitions the recount to 0 via `atomic_cmpxchg` and severs the parent pointer.

Scenario A: The Map-thread Wins

- 1 Recount Increment**
The Map thread executes `atomic_inc_not_zero(&recount)`. Recount is 1, so increment succeeds (**Recount → 2**).
- 2 PTE Written**
The Map thread successfully writes the new leaf PTE under `srcu_read_lock()`.
- 3 Shrinker Fails**
The Shrinker attempts `atomic_cmpxchg(&recount, 1, 0)`. CAS fails because recount is now 2.
- 4 Aborted Severance**
Shrinker cleanly aborts, erases page from per-domain Xarray, and leaves active L4 branch intact.

Scenario B: The Shrinker Wins

- 1 State Claim**
Shrinker executes `atomic_cmpxchg(&recount, 1, 0)`. CAS succeeds, claiming page and locking recount at 0 (Dead).
- 2 Mapper Blocked**
Map thread attempts `atomic_inc_not_zero(&recount)`. Call fails immediately because recount is 0.
- 3 Clean Retreat**
Map thread drops the SRCU read lock and retries; once the parent slot is cleared, it allocates a fresh L4 page.
- 4 Grace Period & Free**
Shrinker severs parent pointer, invokes `synchronize_srcu()` to wait for readers, and frees old page.

Key Principle: Atomic `cmpxchg()` determines the winner of the race, while `SRCU` makes freeing the dead directory safe for concurrent walkers.

Why Refcount Only L4 / Leaf Directory Pages?

Eliminating the Cache-Line Tax

Prevents contention and serialization on shared upper-level page table lines:

- **Cache-Line Bouncing:** Eliminates contention on shared L1/L2 descriptors across physical cores/NUMAs.
- **Localized Updates:** Restricts refcount updates to the leaf directory's own cache line.
- **High Yield:** >99.8% of table pages in a fully populated tree are L4 directories.

Level	Span per Table	Global Page Footprint
L1 (Root)	256TB	1 Page (512^0)
L2 Dir	512GB	512 Pages (512^1)
L3 Dir	1GB	262,144 Pages (512^2)
L4 Leaf	2MB	134,217,728 Pages (512^3)



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Plumber's Discussion

1. Observability vs. Shrinker

Do we need observability if we have shrinkers?

2. Reclaim Strategy

Free on-demand like shrinker or proactively?

3. Shrinker vs. Work Deferral

Is registering a shrinker optimal, or should we consider alternative deferral tools?

4. Struct Page Re-purpose

Is re-purposing **struct page** fields acceptable for IOPT tracking?

5. Leaf Directories with Child Tables

Count child tables at install time, or strictly restrict refcounting to L4?

6. DMA Domain Support

- Should we support these? Maps/unmaps can run in IRQ contexts.
- Wouldn't the kernel ever want to reclaim DMA domain IOPTs?

7. Support for All Unmanaged Domains?

VFIO_TYPE1 v/s IOMMUFD v.s in-kernel IOMMU API user

8. IOTLB Cache Invalidation

Use **flush_iotlb_all()** before freeing, or range-based paging structure flushes?

9. SRCU in Reclaim Context

Defer free (**call_srcu** / work item) when running inside direct reclaim?

10. Page Accounting for User-Space

Account page table memory directly to cgroups or user process footprints?

