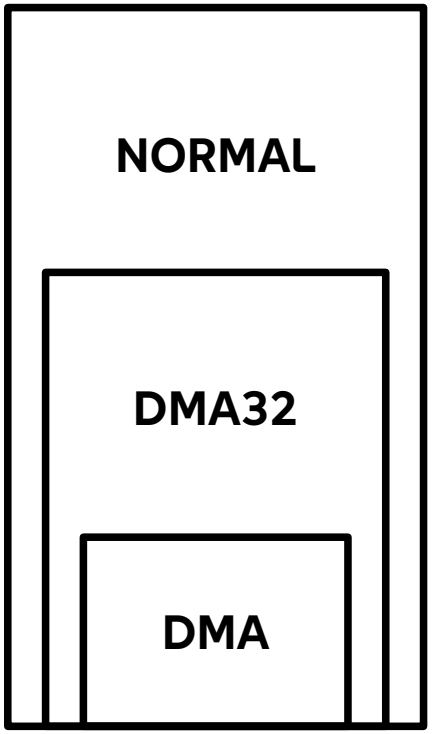
Towards Better Memory Allocation for Device Drivers

Petr Tesarik (SUSE)
<ptesarik@suse.com>

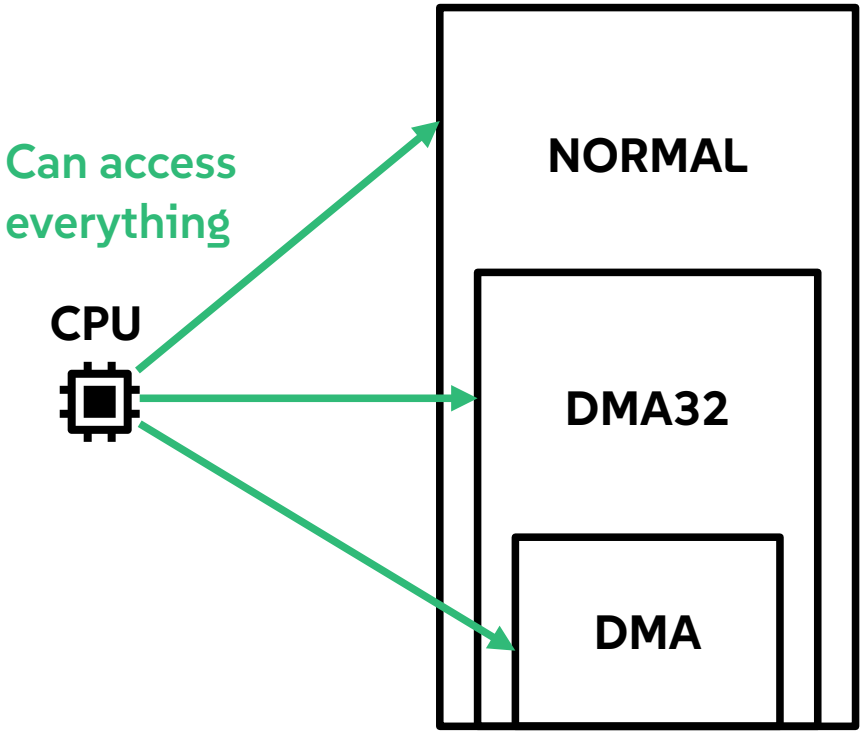
SUSE

Simple Idea

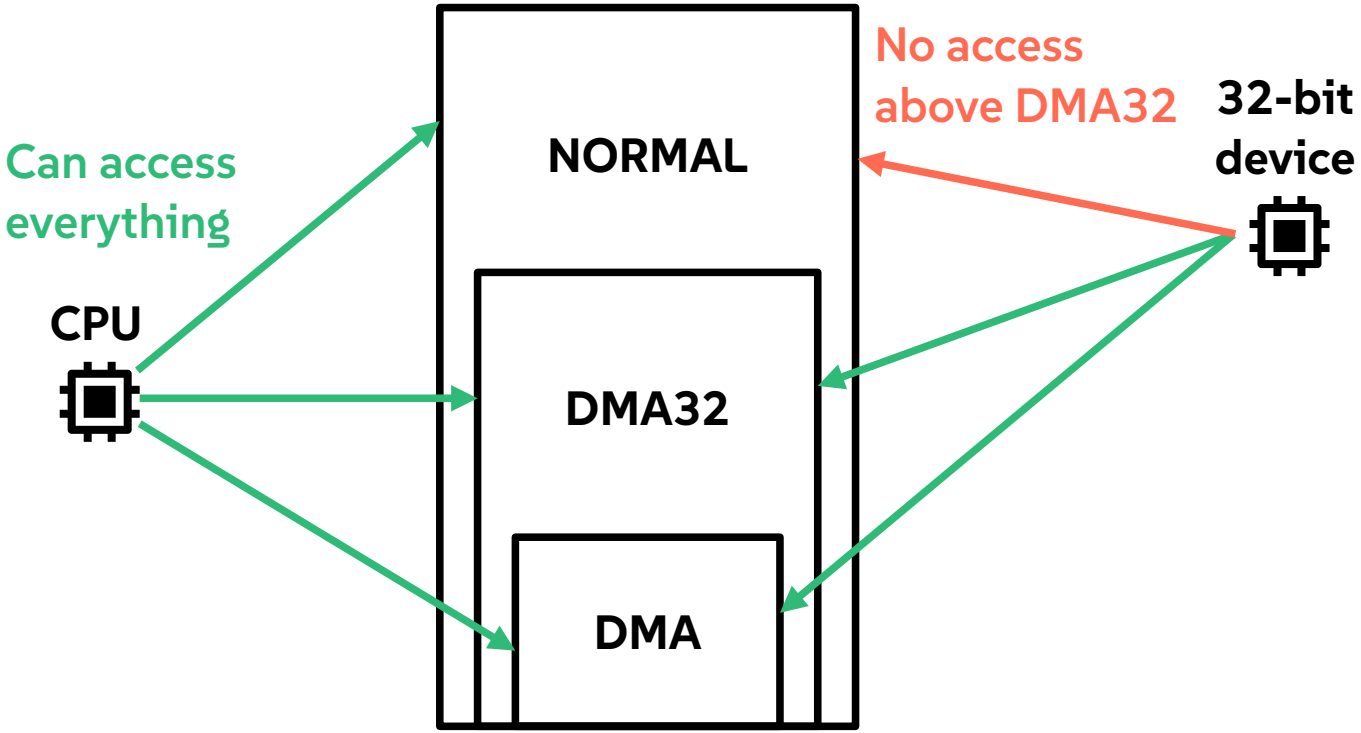
Memory Zones by Physical Address



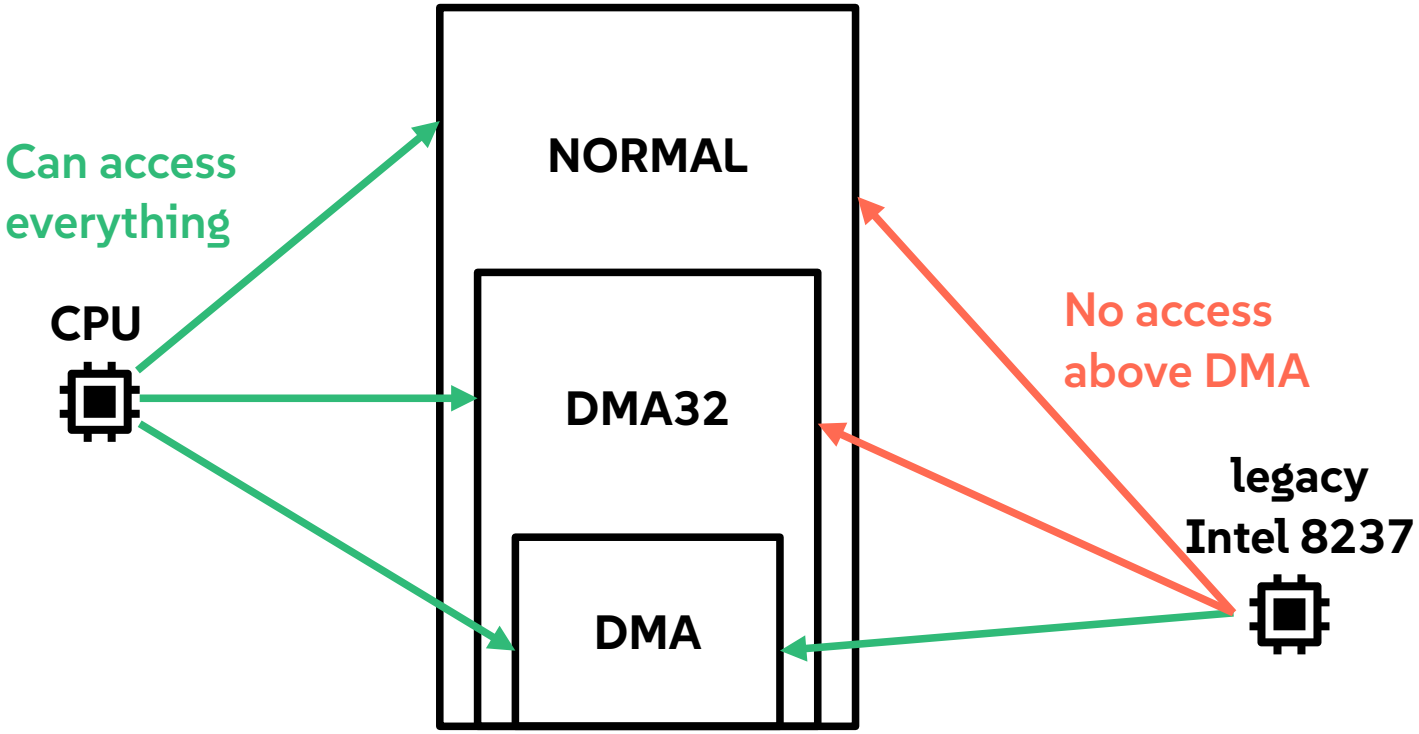
Memory Zones by Physical Address



Memory Zones by Physical Address

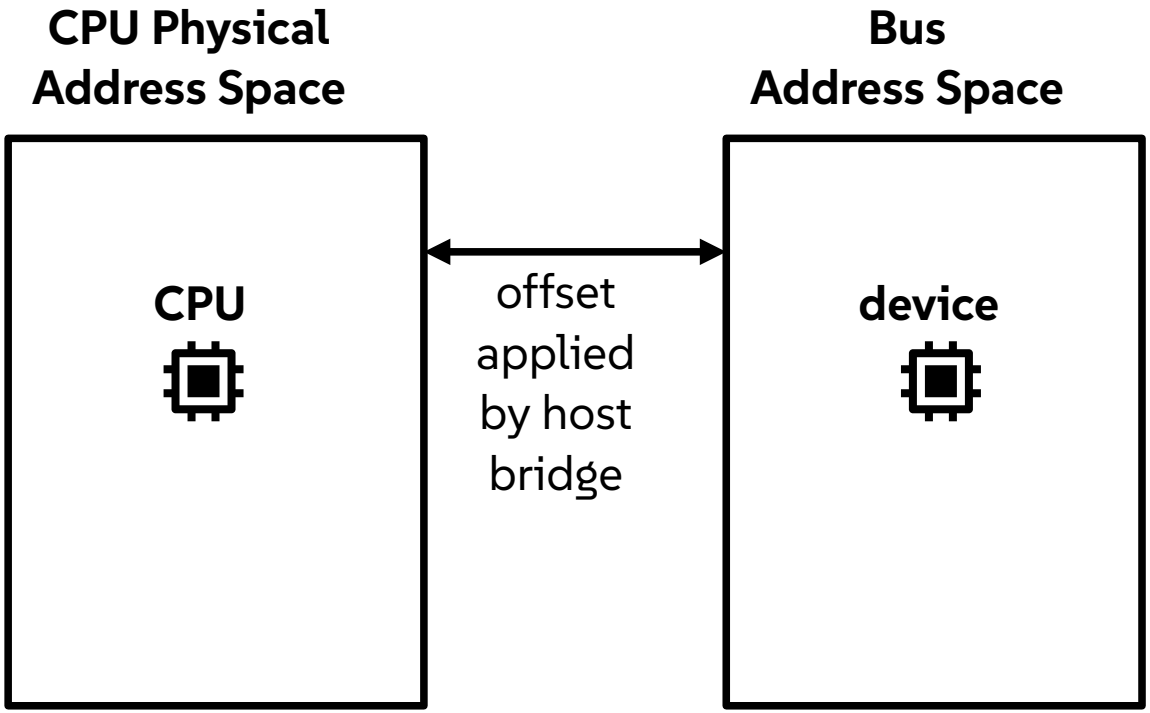


Memory Zones by Physical Address

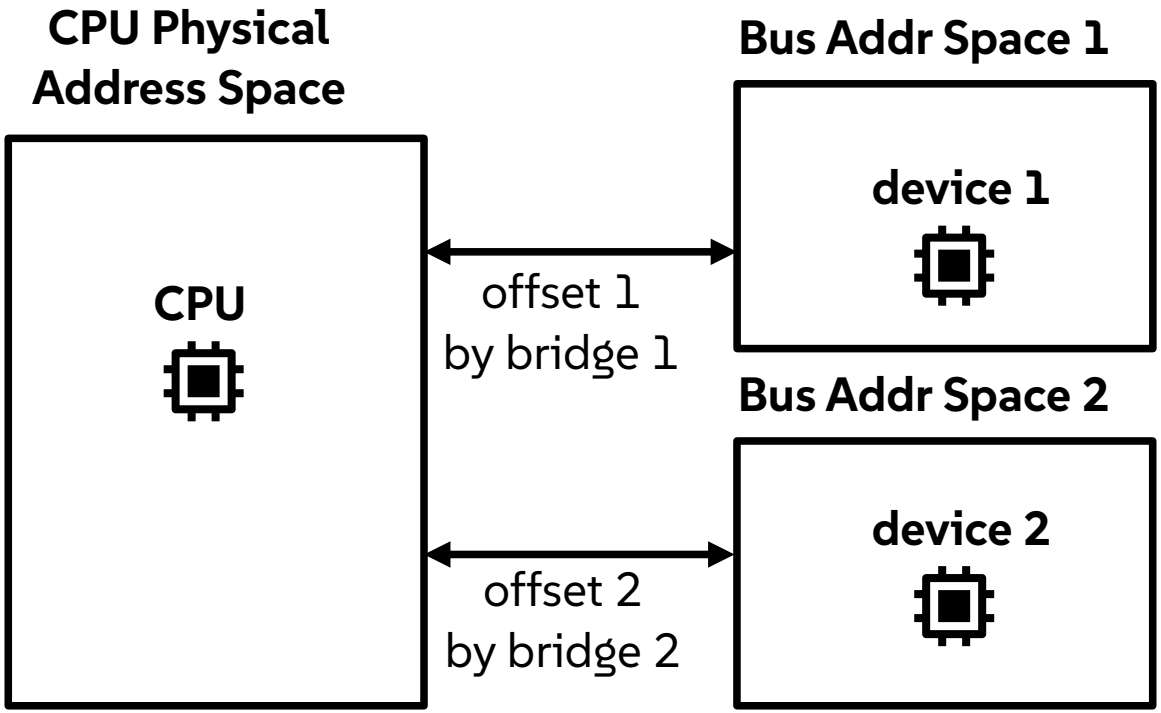


Reality Strikes Back

Different Address Spaces



It Gets Worse...



What Address is OK for a Device?

```
bool dma_coherent_ok(struct device *dev, phys_addr_t phys, size_t size)
{
        // Translate a CPU physical address to a bus address:
        dma_addr_t dma_addr = phys_to_dma_direct(dev, phys);

        // Does the CPU address even map to a bus address?
        if (dma_addr == DMA_MAPPING_ERROR)
                return false;

        // Good, we have a bus address. Now check if it is within limits:
        return dma_addr + size - 1 <=
                min_not_zero(dev->coherent_dma_mask, dev->bus_dma_limit);
}
```

How Buffers are Allocated

```
static struct page *__dma_direct_alloc_pages(struct device *dev, size_t size, gfp_t gfp, bool allow_highmem)
{
        int node = dev_to_node(dev);
        struct page *page = NULL;
        u64 phys_limit;

        // Use a restricted pool if possible:
        if (is_swiotlb_for_alloc(dev))
                return dma_direct_alloc_swiotlb(dev, size);

        // Try to allocate from CMA, hoping for the best:
        gfp |= dma_direct_optimal_gfp_mask(dev, &phys_limit);
        page = dma_alloc_contiguous(dev, size, gfp);

        // If CMA is in fact not suitable for this device, free the pages again and continue as if there was no CMA:
        if (page) {
                if (!dma_coherent_ok(dev, page_to_phys(page), size) ||
                    (!allow_highmem && PageHighMem(page))) {
                        dma_free_contiguous(dev, page, size);
                        page = NULL;
                }
        }
…
```

How Buffers are Allocated (cont'd)

```
…
        // Allocate from the zoned buddy allocator:
again:
        if (!page)
                page = alloc_pages_node(node, gfp, get_order(size));
        if (page && !dma_coherent_ok(dev, page_to_phys(page), size)) {
                // Bad luck? Free the freshly allocated pages!
                dma_free_contiguous(dev, page, size);
                page = NULL;

                // Try DMA32 if available and looks like it might help:
                if (IS_ENABLED(CONFIG_ZONE_DMA32) && phys_limit < DMA_BIT_MASK(64) && !(gfp & (GFP_DMA32 | GFP_DMA))) {
                        gfp |= GFP_DMA32;
                        goto again;
                }

                // Try DMA if available and we haven't tried yet:
                if (IS_ENABLED(CONFIG_ZONE_DMA) && !(gfp & GFP_DMA)) {
                        gfp = (gfp & ~GFP_DMA32) | GFP_DMA;
                        goto again;
                }
        }
        return page;
}
```

Memory Encryption Requirements

- SEV always requires DMA to unencrypted addresses.
- SME requires DMA to unencrypted addresses if the device does not support DMA to addresses that include the encryption mask.
- Decrypting a memory range may block, so atomic allocations use coherent pools:
  - One coherent pool per zone, allocated from CMA or with the zoned buddy allocator
  - Similar allocate/check/free loop over possible zones

SUSE

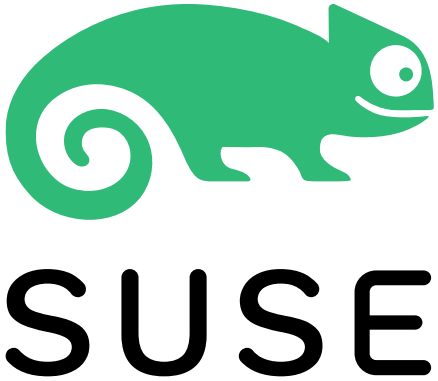
Can We Do Better?

What's Good and Bad About Zones

- Good
  - Dedicated pools used only for atomic allocations
  - Memory reclaim when low watermark is reached
  - Pools automatically refilled during memory reclaim
- Bad
  - Based on global CPU physical address limits, but each device is constrained within its own bus address space
  - Initialized at boot time (before all device constraints are known)
  - No more than 3 zones can be used

Ideas for a Replacement

- Dynamically created allocation groups
  - Mapping of device constraints to physical addresses is known at device initialization time
  - Each group defines an emergency pool of pages for atomic allocations
  - Multiple devices can share a single group (more efficient than per-device pools)
  - Initial “direct reclaim” when a new group is created
- Integrated with the buddy allocator
  - Groups below high watermark are refilled while pages are walked during memory reclaim
- Supersede coherent pools
  - Memory decrypted during reclaim



Questions?

Talk to me now!

Or write to me:

ptesarik@suse.com


SUSE