CS6465: Advanced Operating System Implementation

Lecture 1: Linux Memory Management

Anton Burtsev

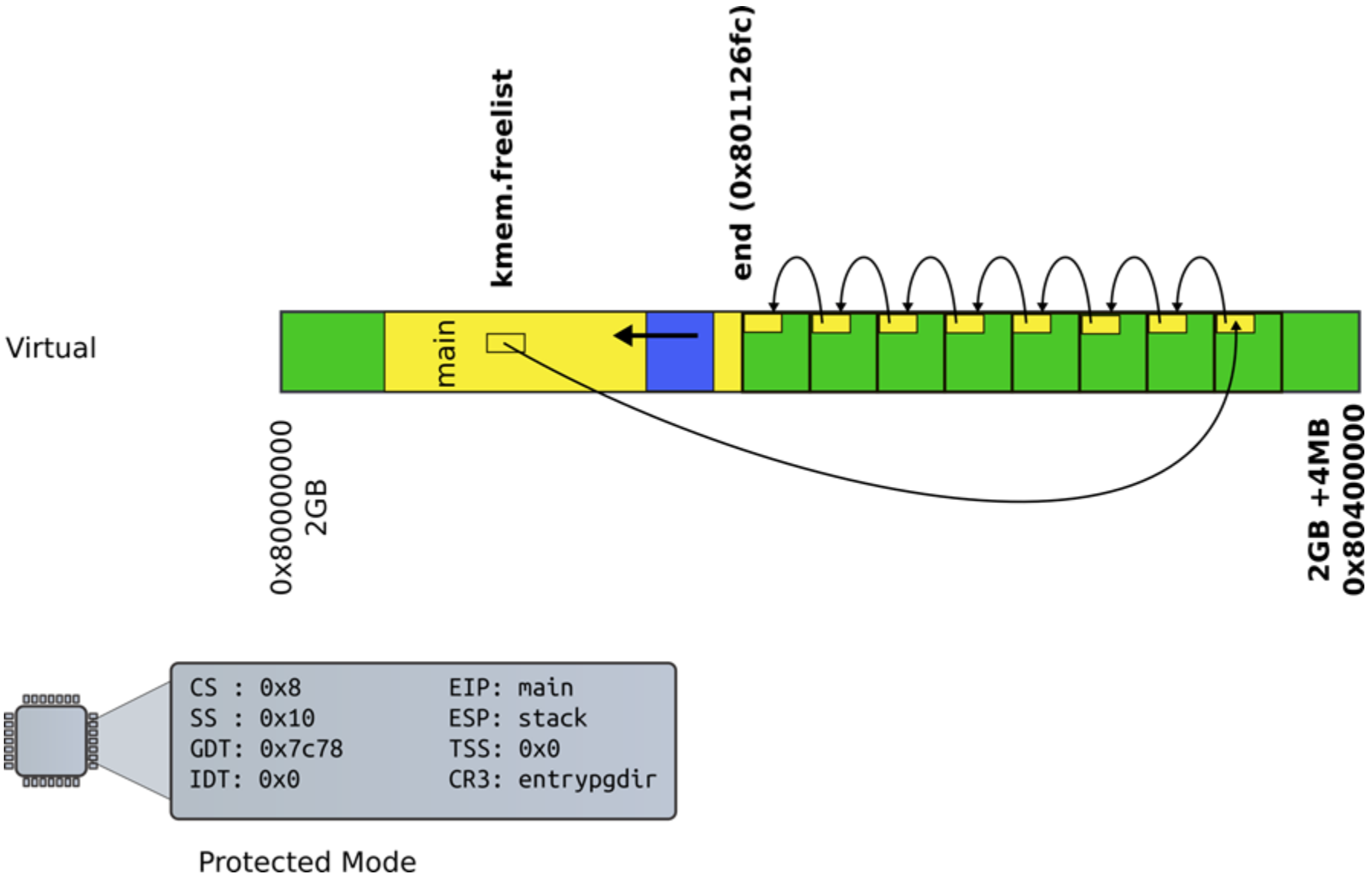
August, 2024

Xv6 Book, Chapter 1. KERNBASE limits the amount of memory a single process can use, which might be irritating on a machine with a full 4 GB of RAM.

Would raising KERNBASE allow a

process to use more memory?

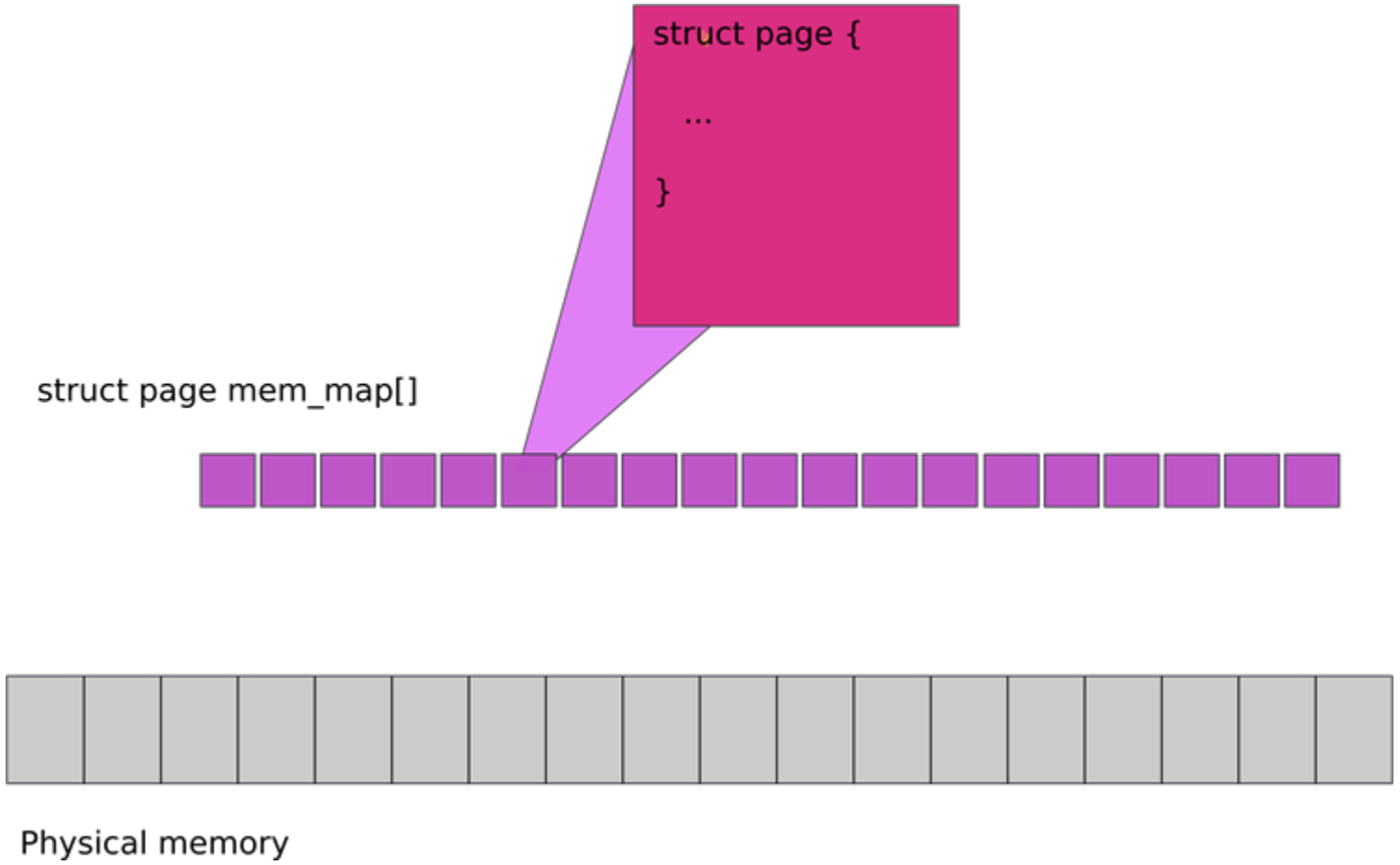
Xv6: physical page allocator



Physical memory

Physical memory

We need a smaller array to describe physical pages, e.g., mem_map[] in Linux



Memory allocation

Simplest memory allocator

- Bitmap of all pages
- Bootmem allocator in Linux
- Allocation searches for an unused page
- Multiple sub-page allocations can be served from the same page by advancing a pointer

- Works ok, but what is the problem?

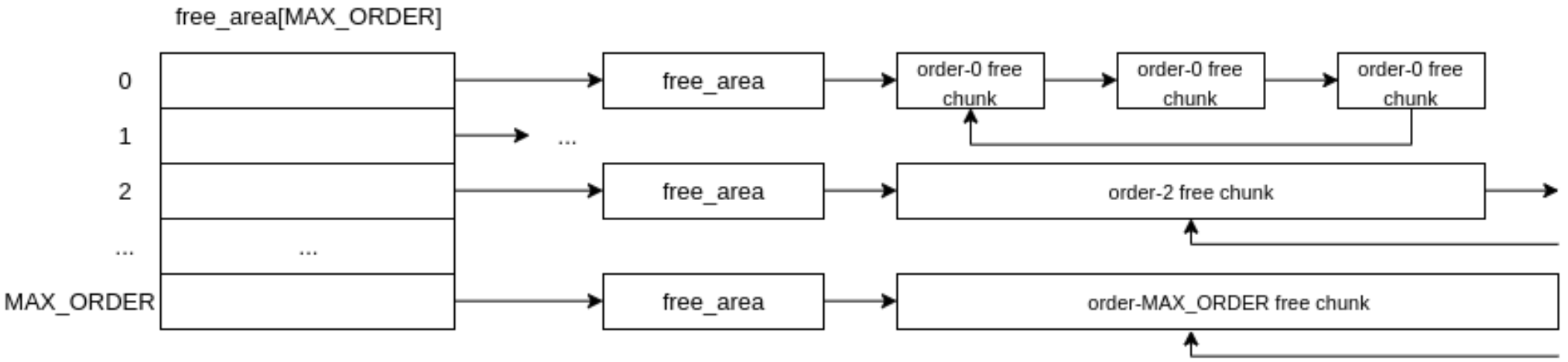
Boot memory allocator

- Bitmap of all pages
- Bootmem allocator in Linux
- Allocation searches for an unused page
- Multiple sub-page allocations can be served from the same page by advancing a pointer

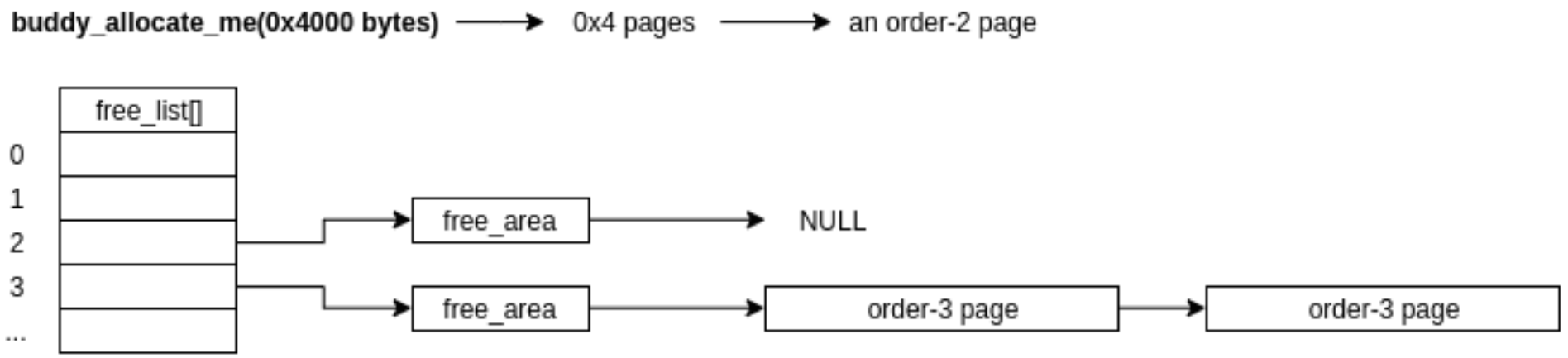
- Works ok, but what is the problem?
- Linear scan of the bitmap
  - Too long

Buddy:
Physical Memory Allocator

Buddy memory allocator



Buddy memory allocator



What's wrong with buddy?

What's wrong with buddy?

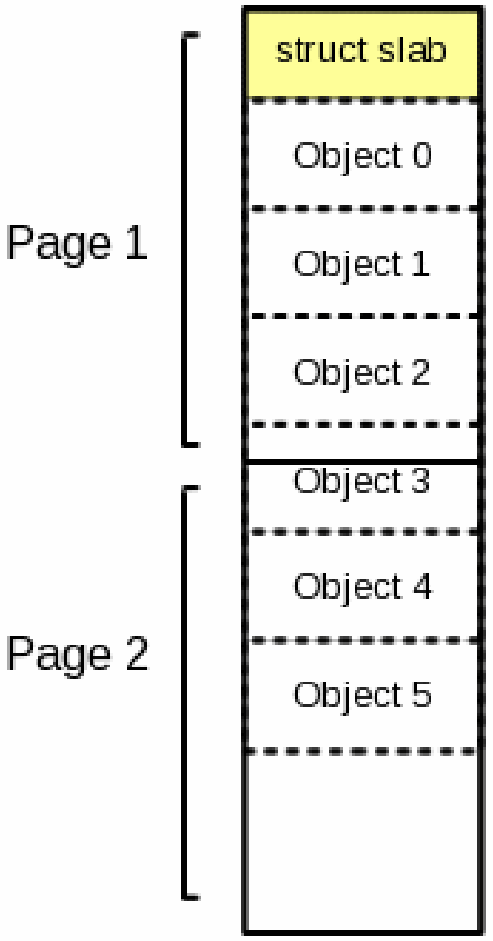
- Buddy allocator is ok for large allocations
  - E.g. 1 page or more
- But what about small allocations?
- Buddy uses the whole page for a 4 bytes allocation
  - Wasteful
- Buddy is still slow for short-lived objects

Slab:

Allocator for object of a fixed size

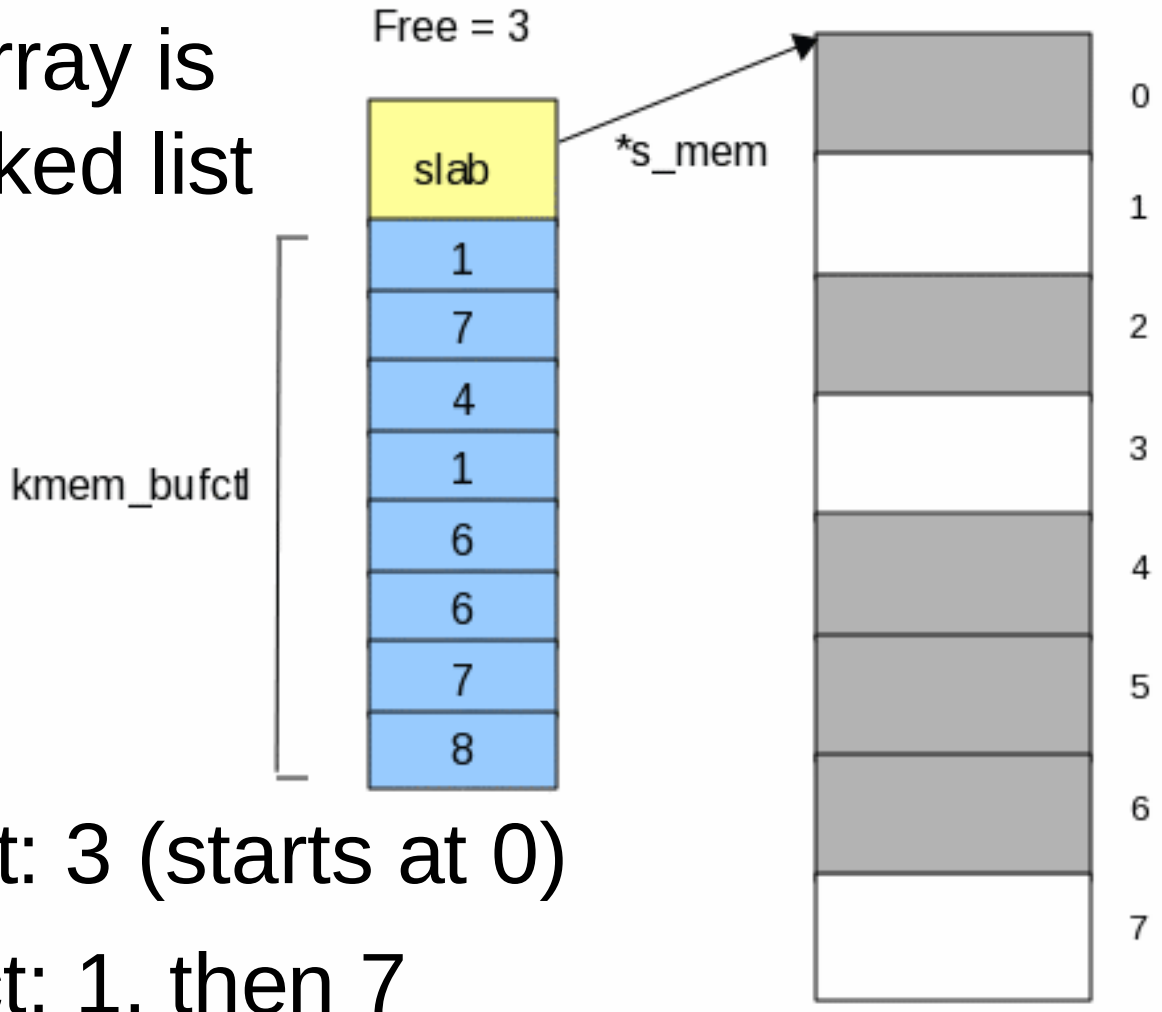
Slab

- A 2 page slab with 6 objects



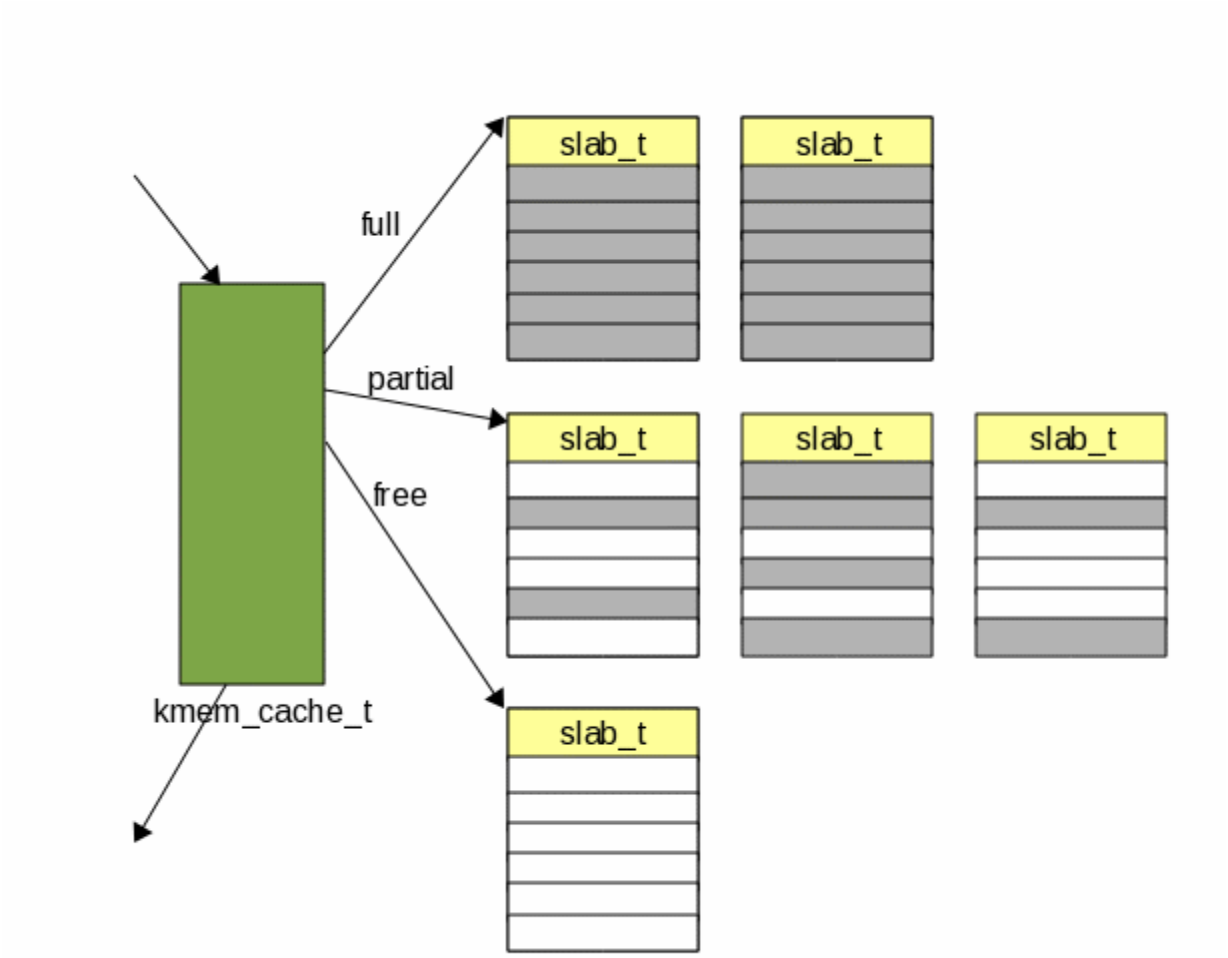
Keeping track of free objects

- kmem_bufctl array is effectively a linked list



- First free object: 3 (starts at 0)
- Next free object: 1, then 7

A cache is formed out of slabs



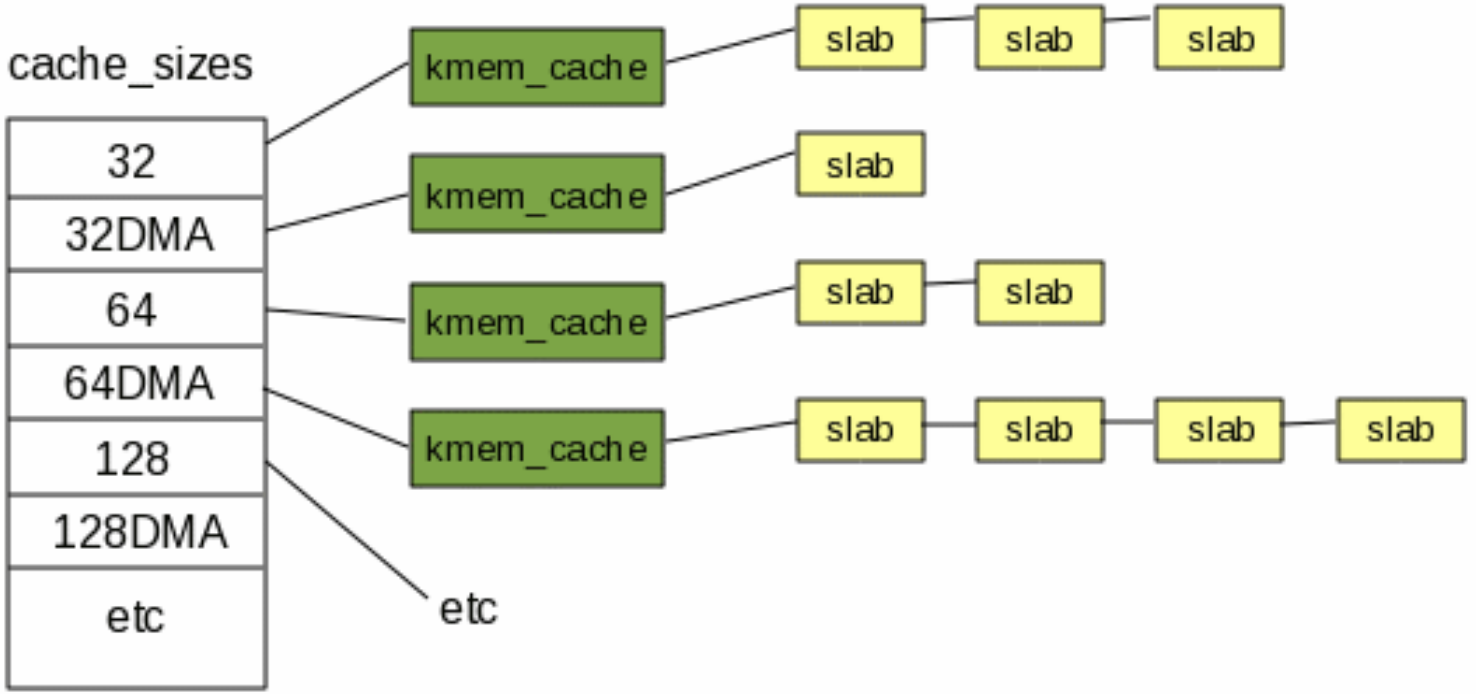
Slab is fine, but what's wrong?

Slab is fine, but what's wrong?

- We can only allocate objects of one size

Kmalloc(): variable size objects

- A table of caches
- Size: 32, 64, 128, etc.



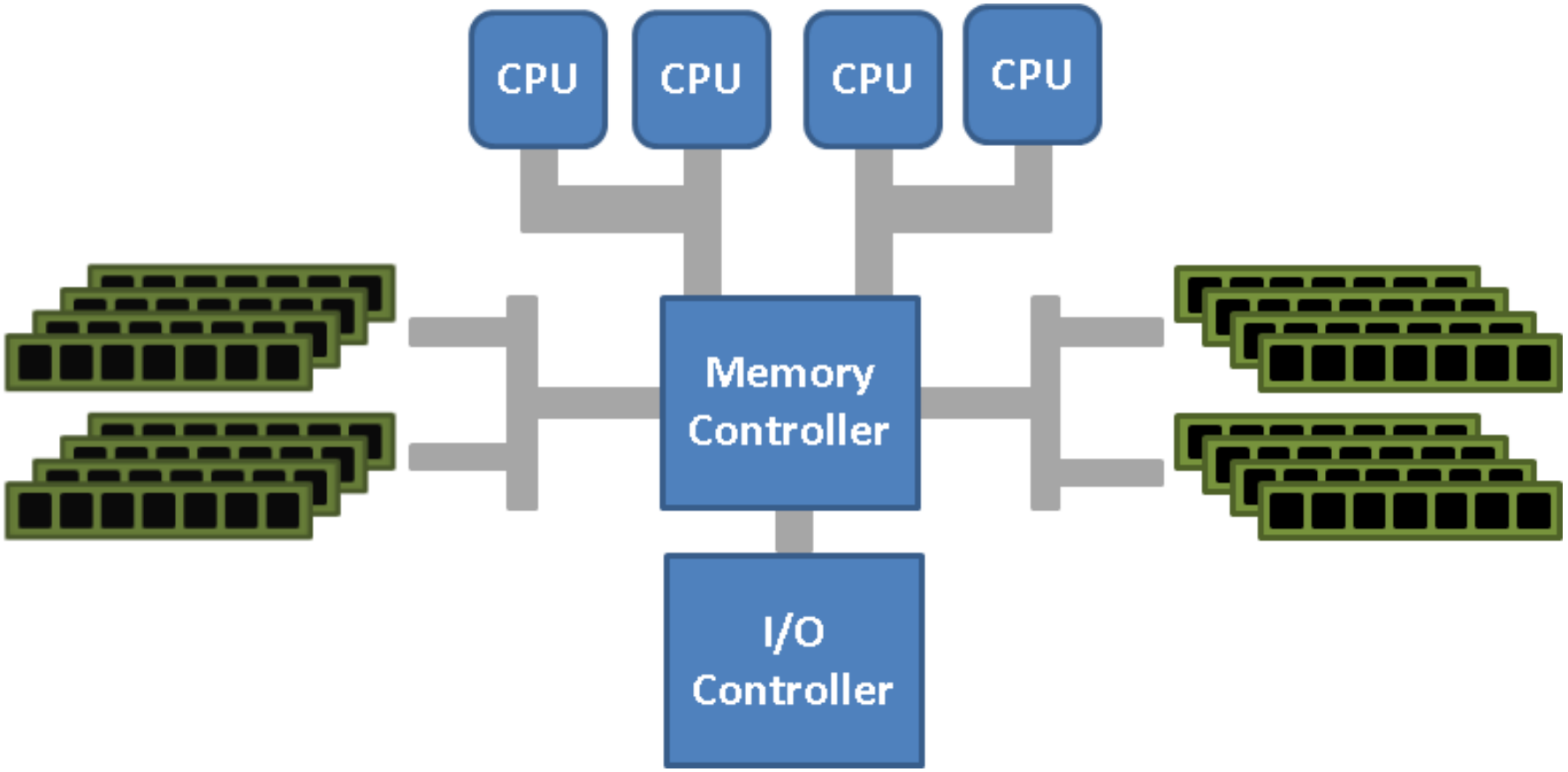
NUMA

Non-uniform memory access

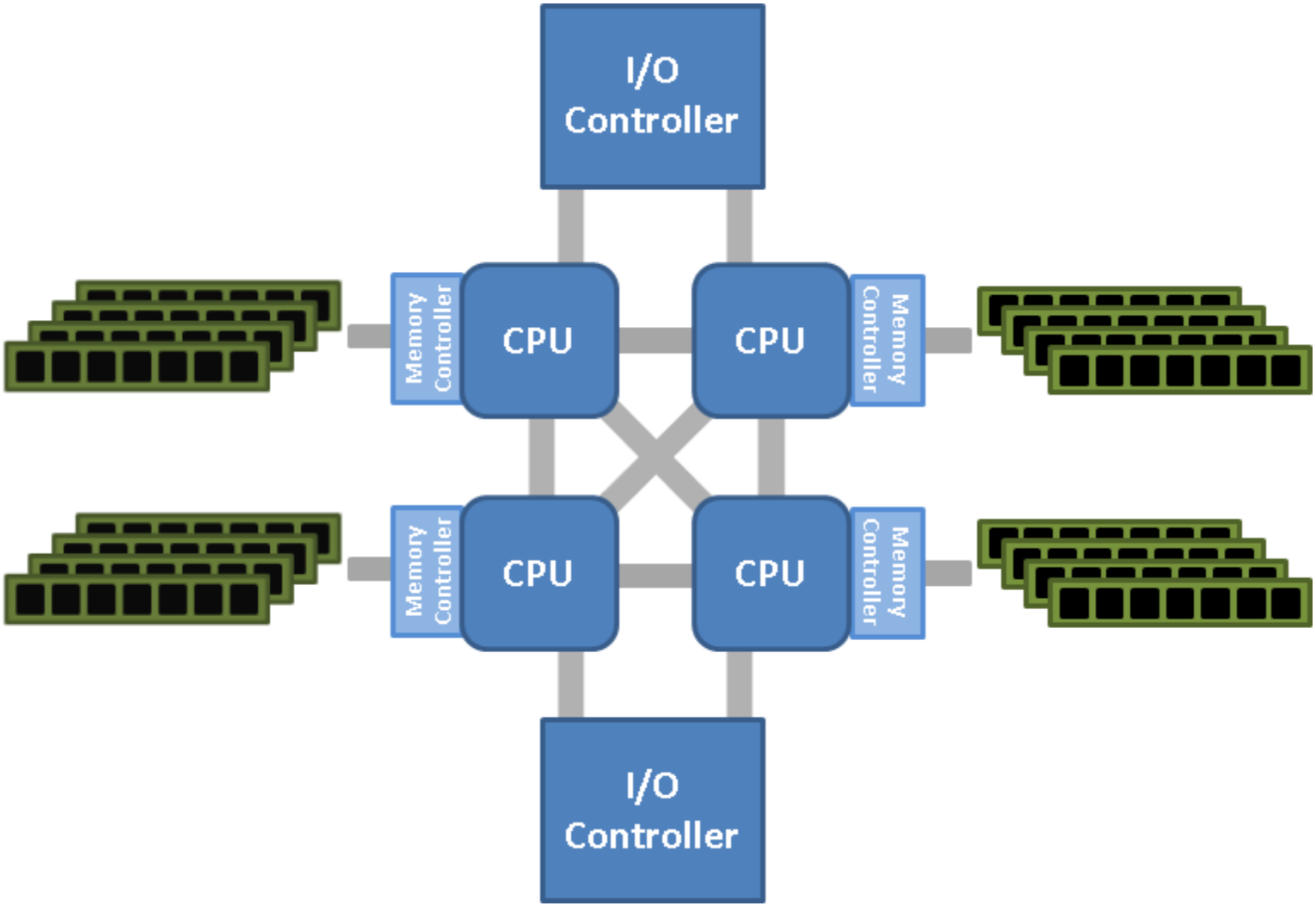
Uniform and non-uniform memory access

- Parts of memory can be faster than others

Uniform memory access (UMA)

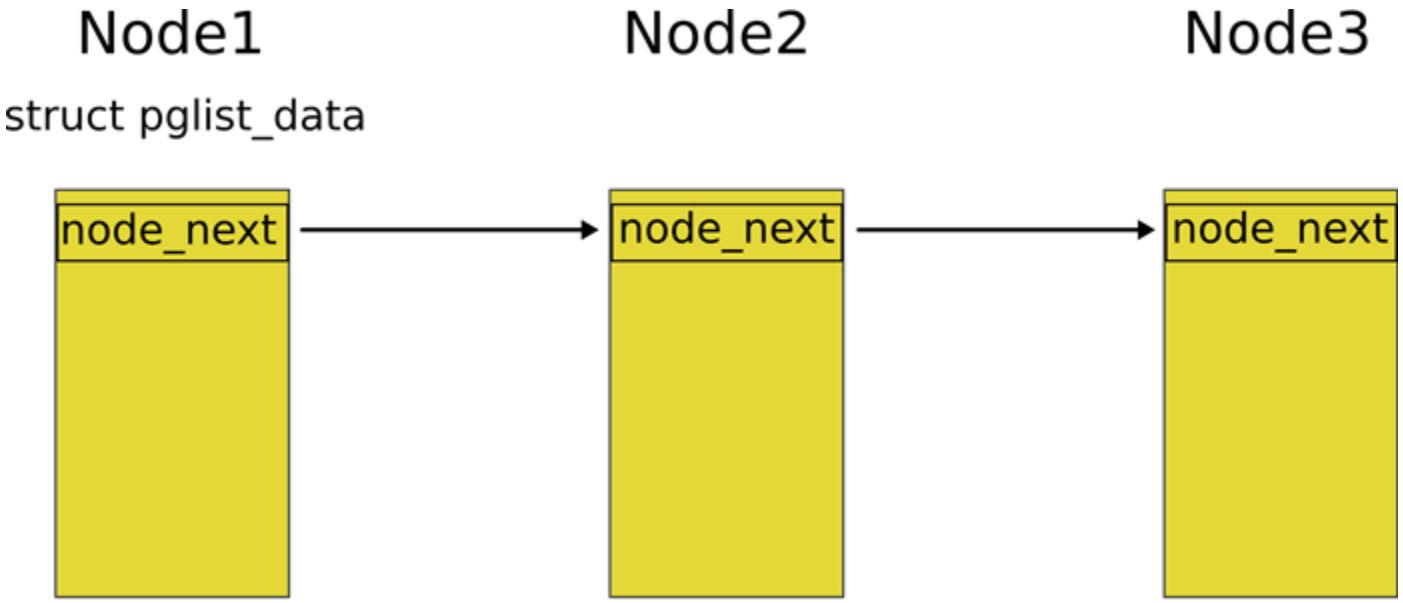


Nonuniform memory access (NUMA)



Nodes

- Attempt to allocate memory from the current node
- Fall back to the next node in list
  - If ran out of local memory



pglist_data represents a node

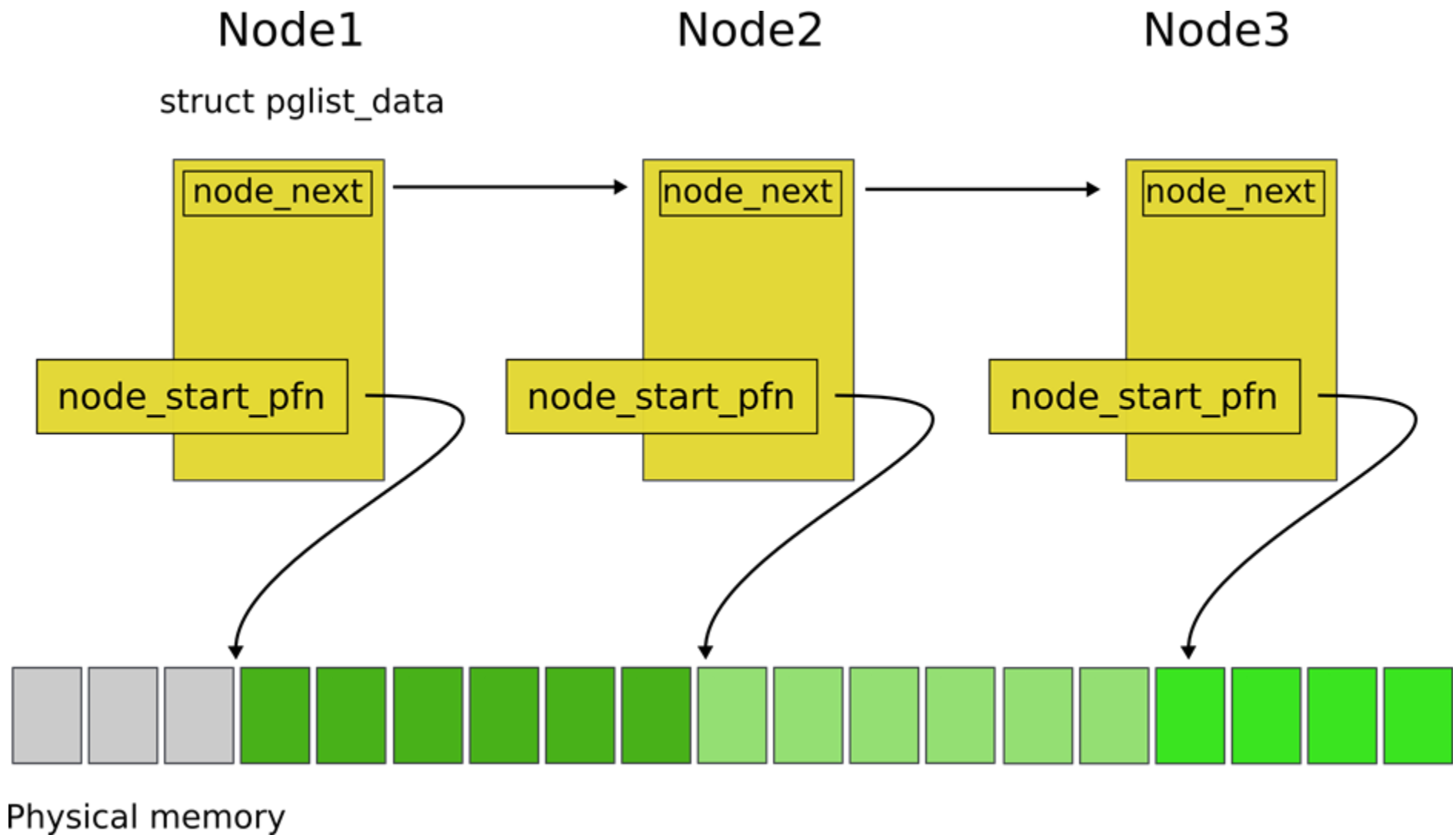
<mmzone.h>

```
typedef struct pglist_data {
        struct zone node_zones[MAX_NR_ZONES];
        struct zonelist node_zonelists[MAX_ZONELISTS];
        int nr_zones;
        struct page *node_mem_map;
        struct bootmem_data *bdata;

        unsigned long node_start_pfn;
        unsigned long node_present_pages; /* total number of physical pages */
        unsigned long node_spanned_pages; /* total size of physical page
                                             range, including holes */
        int node_id;
        struct pglist_data *pgdat_next;
        wait_queue_head_t kswapd_wait;
        struct task_struct *kswapd;
        int kswapd_max_order;
} pg_data_t;
```

https://elixir.bootlin.com/linux/v6.10.6/source/include/linux/mmzone.h#L1277

Nodes

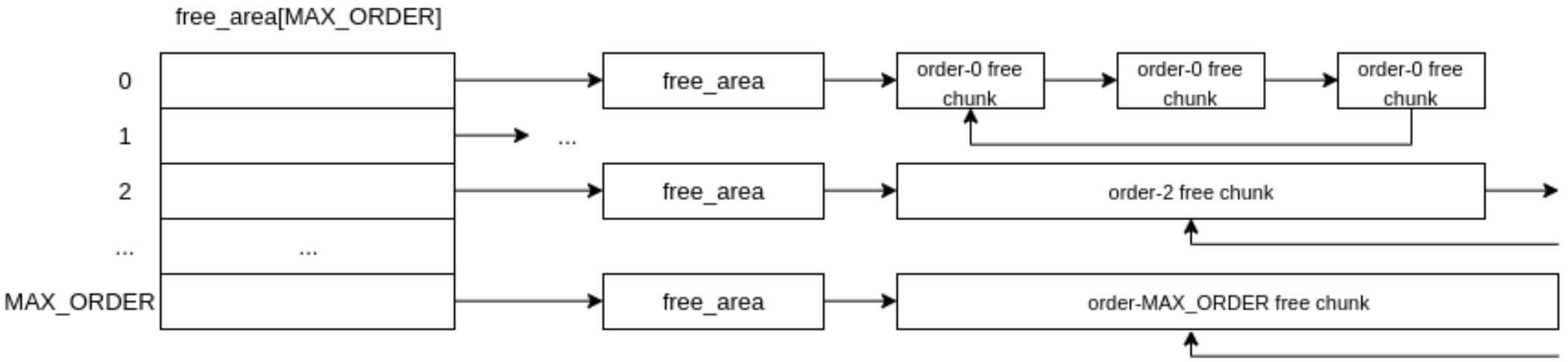

Node1
Node2
Node3
struct pglist_data
node_next
node_next
node_next
node_start_pfn
node_start_pfn
node_start_pfn
Physical memory

zone represents a zone

```
struct zone {
    ...
    /* free areas of different sizes */
    struct free_area free_area[NR_PAGE_ORDERS];
    ...
}
```

https://elixir.bootlin.com/linux/v6.10.6/source/include/linux/mmzone.h#L824

Remember buddy?

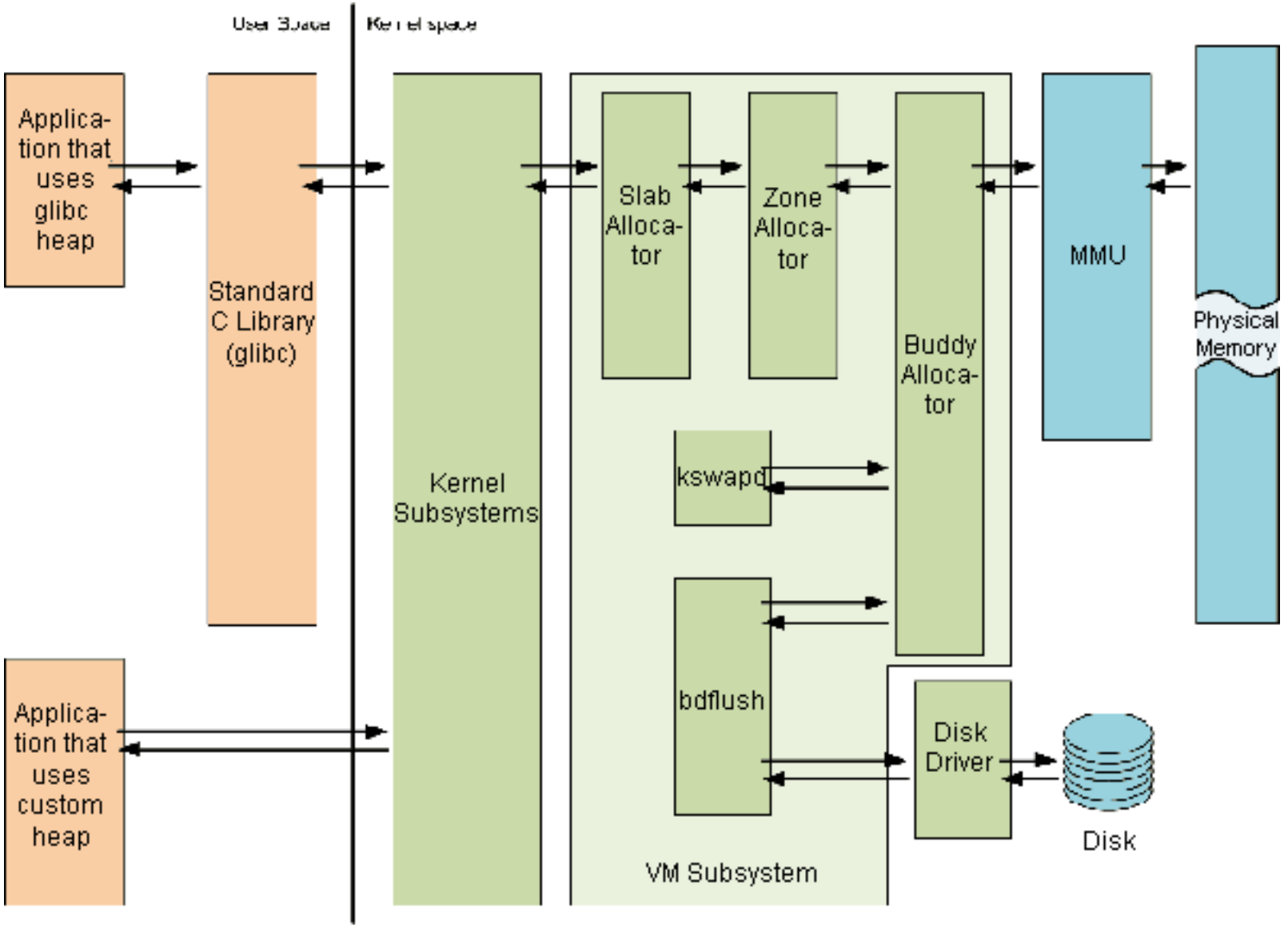


zone represents a zone

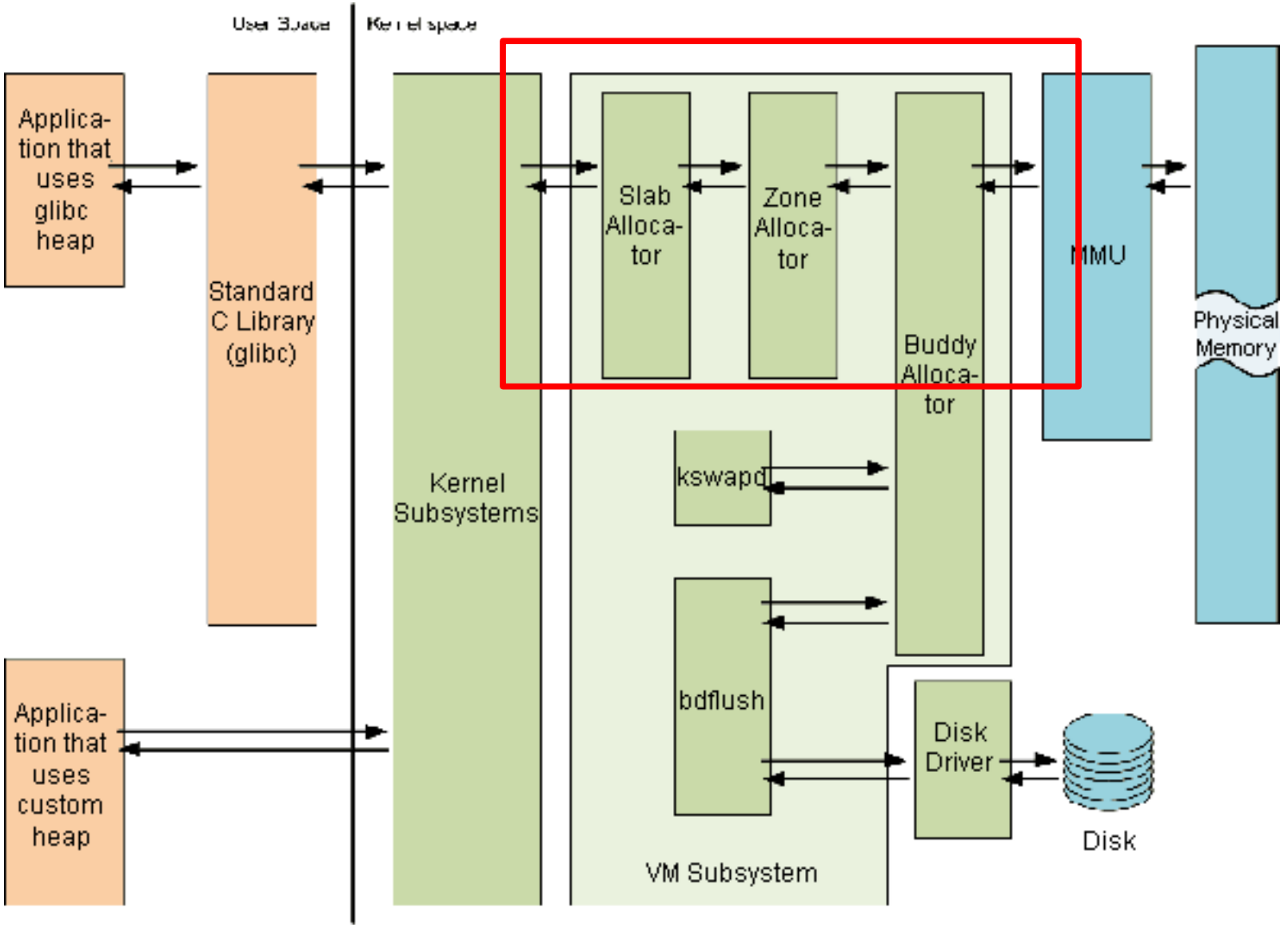
```
struct free_area {
    struct list_head free_list[MIGRATE_TYPES];
    unsigned long nr_free;
};
```

https://elixir.bootlin.com/linux/v6.10.6/source/include/linux/mmzone.h#L117

Linux memory management



Linux memory management



Recap: understanding allocators

How do we recover slab metadata on free()?

How do we recover slab metadata on free()?

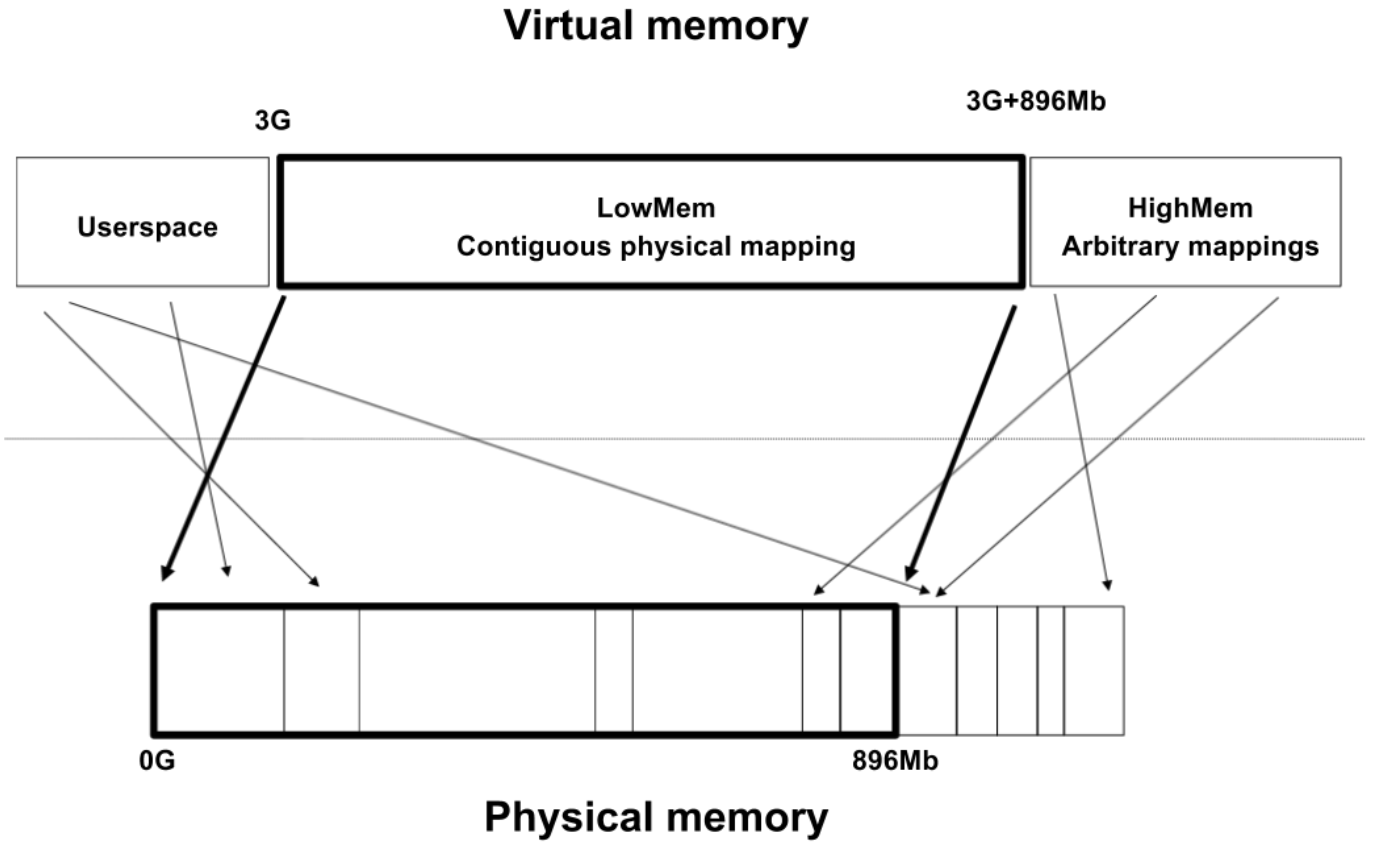
mm/slab.c

```
void kfree(const void *objp)
{
        kmem_cache_t *c;
        unsigned long flags;

        if (unlikely(ZERO_OR_NULL_PTR(objp)))
                return;
        c = virt_to_cache(objp));
        __cache_free(c, (void*)objp);
}
```

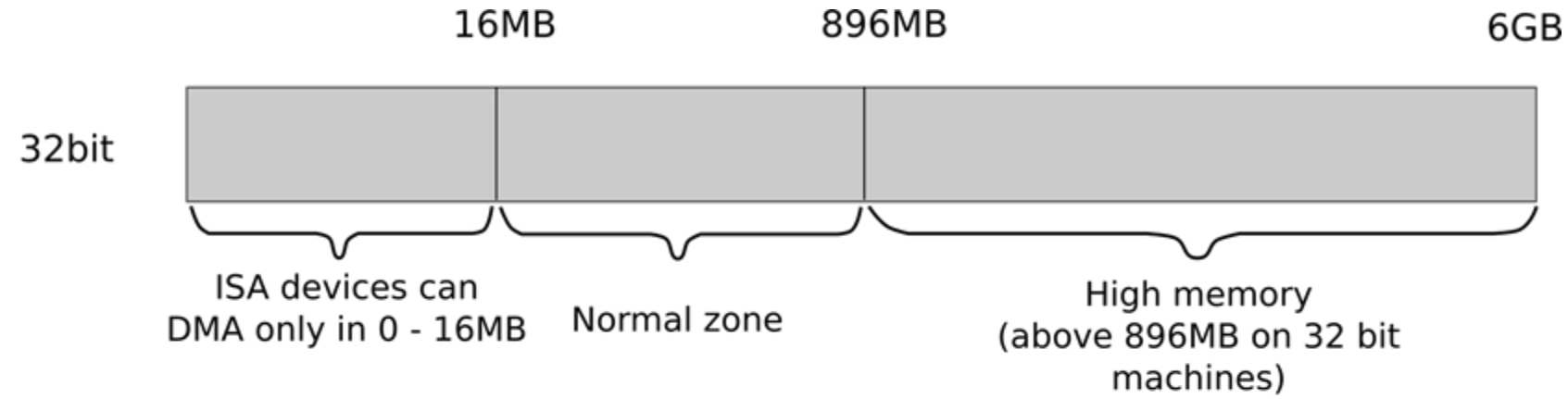
https://elixir.bootlin.com/linux/v2.6.39.4/source/mm/slab.c#L3819

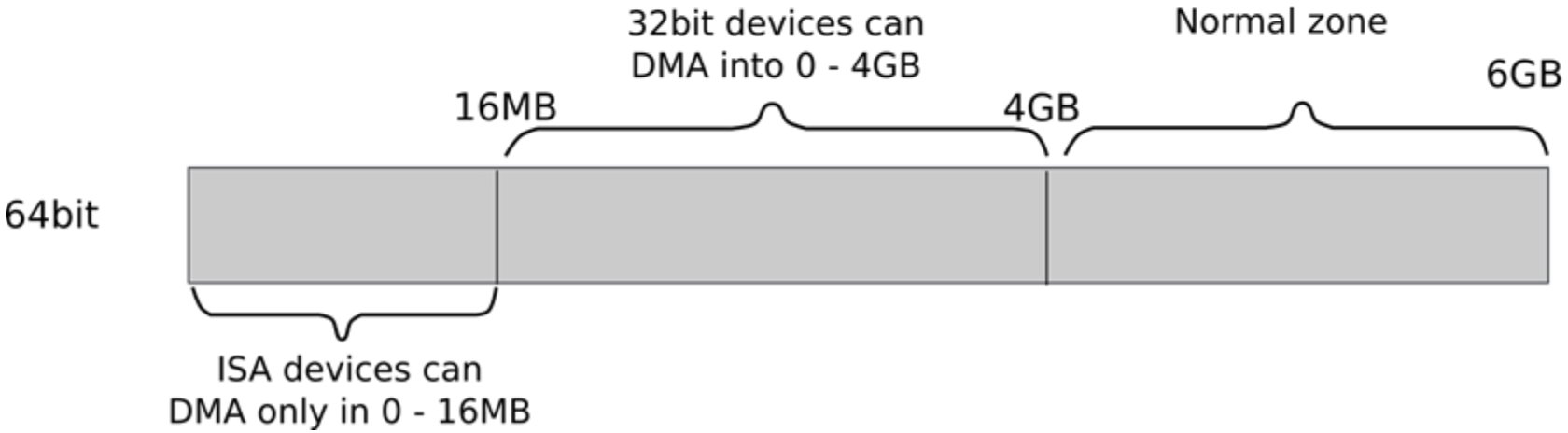
Memory map on x86_32



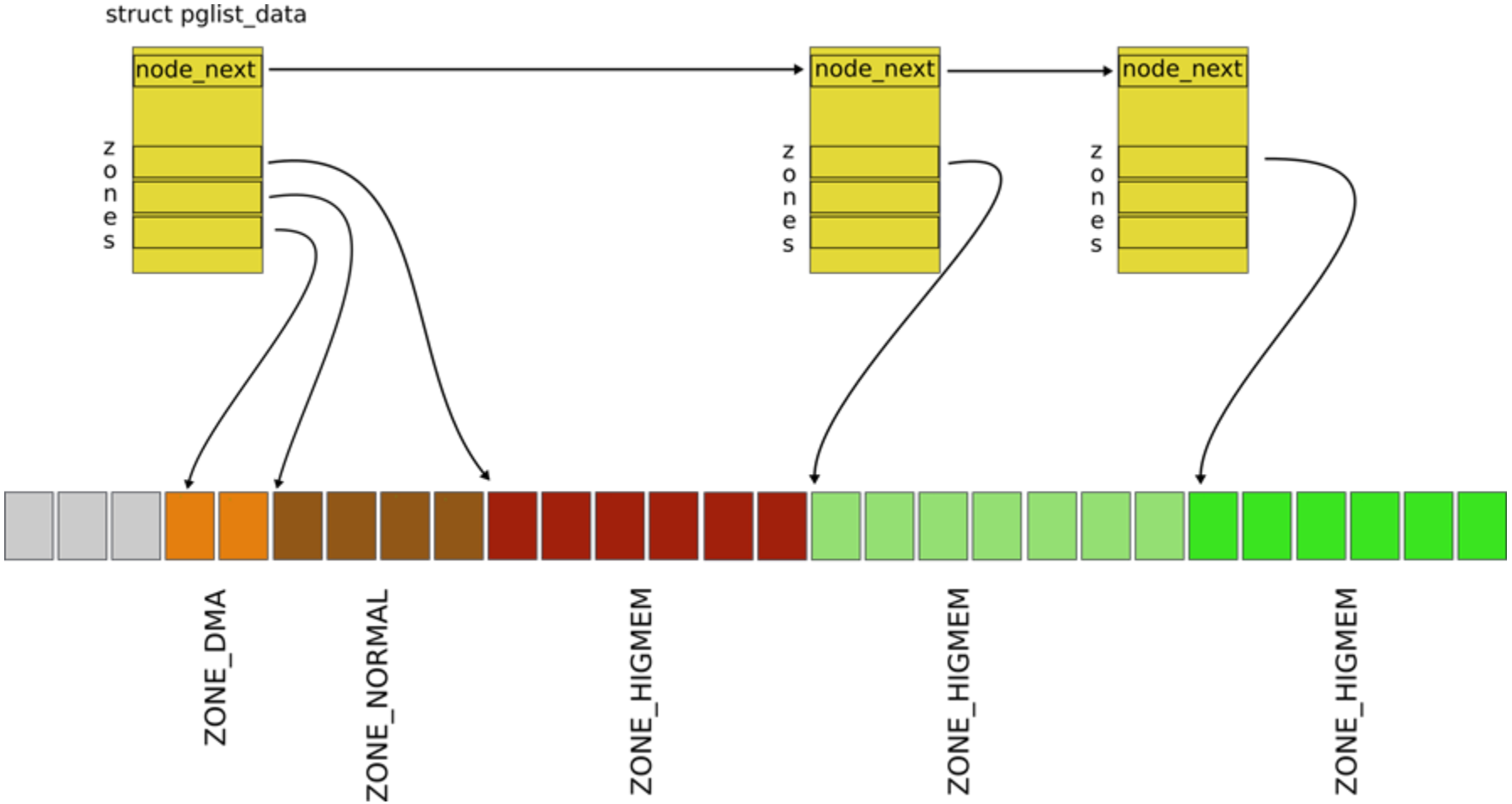
https://www.kernel.org/doc/Documentation/x86/x86_64/mm.txt

Zones


16MB
896MB
6GB
32bit
ISA devices can
DMA only in 0 - 16MB
Normal zone
High memory
(above 896MB on 32 bit
machines)


32bit devices can
DMA into 0 - 4GB
Normal zone
6GB
16MB
4GB
64bit
ISA devices can
DMA only in 0 - 16MB

Zones


struct pglist_data
node_next
zones
node_next
zones
node_next
zones
ZONE_DMA
ZONE_NORMAL
ZONE_HIGMEM
ZONE_HIGMEM
ZONE_HIGMEM

External metadata

- Slabs for large objects
- Don’t want to waste an entire region for just metadata

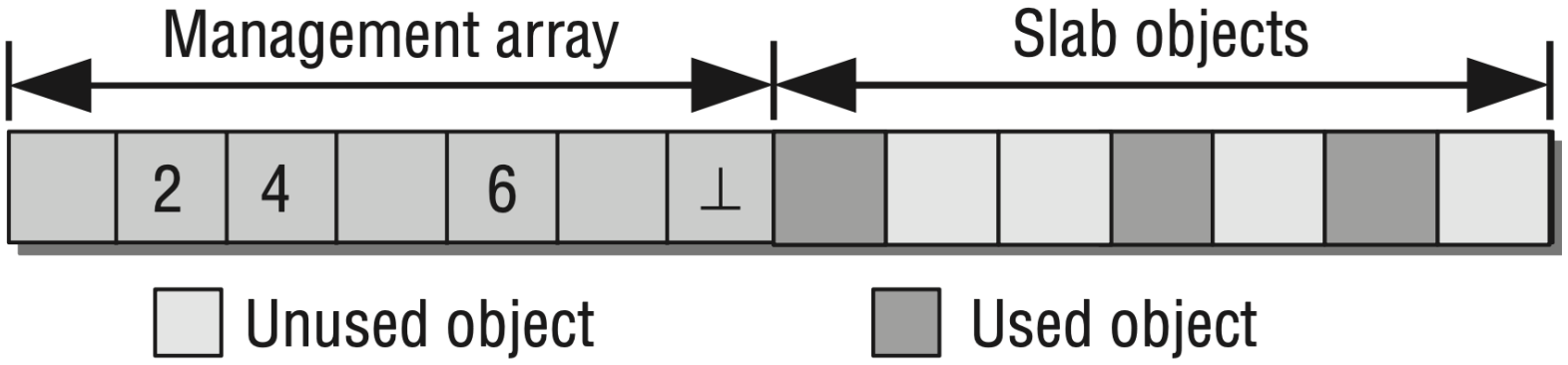


Figure 3-47: Management of the free objects in a slab.

External metadata

- Slabs for large objects
- Don’t want to waste an entire region for just metadata

- `page->list.next` points to the management structure of the cache in which the page resides.
- `page->list.prev` points to the management structure of the slab on which the page is held.

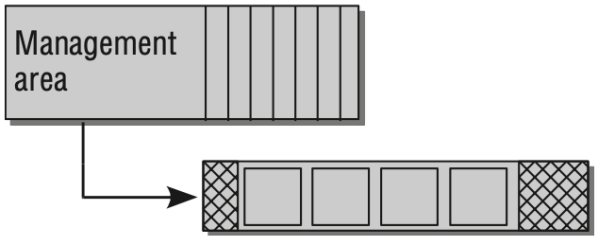


Figure 3-48: Slab with external (*off-slab*) slab header.

Compound Pages

- First page is a head page
  - Others are called tail pages
  - PG_compound bit is set

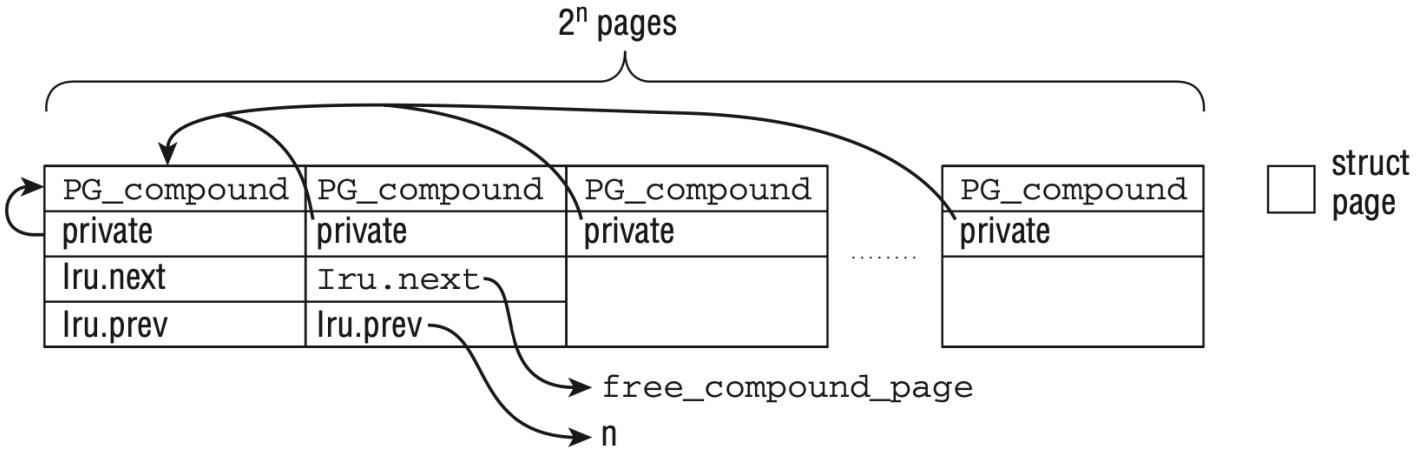


Figure 3-33: Higher-order allocations generate compound pages in which the individual pages are linked.

How do we recover the size of the region on buddy free()?

vmalloc()

<vmalloc.h>

```
struct vm_struct {
        struct vm_struct        *next;
        void                    *addr;
        unsigned long           size;
        unsigned long           flags;
        struct page             **pages;
        unsigned int            nr_pages;
        unsigned long           phys_addr;
};
```

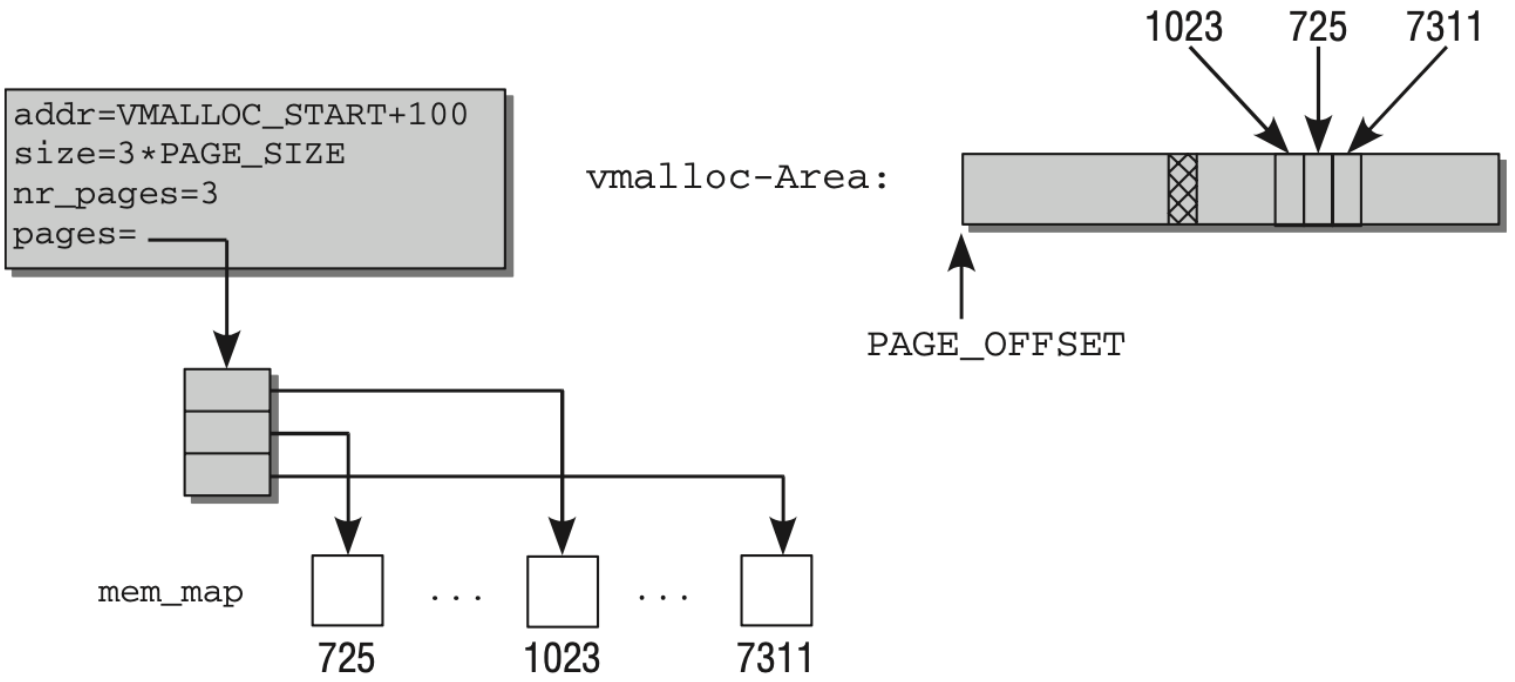


Figure 3-38: Mapping physical pages into the `vmalloc` area.

https://elixir.bootlin.com/linux/v2.6.39.4/source/include/linux/vmalloc.h#L26

Vmalloc area

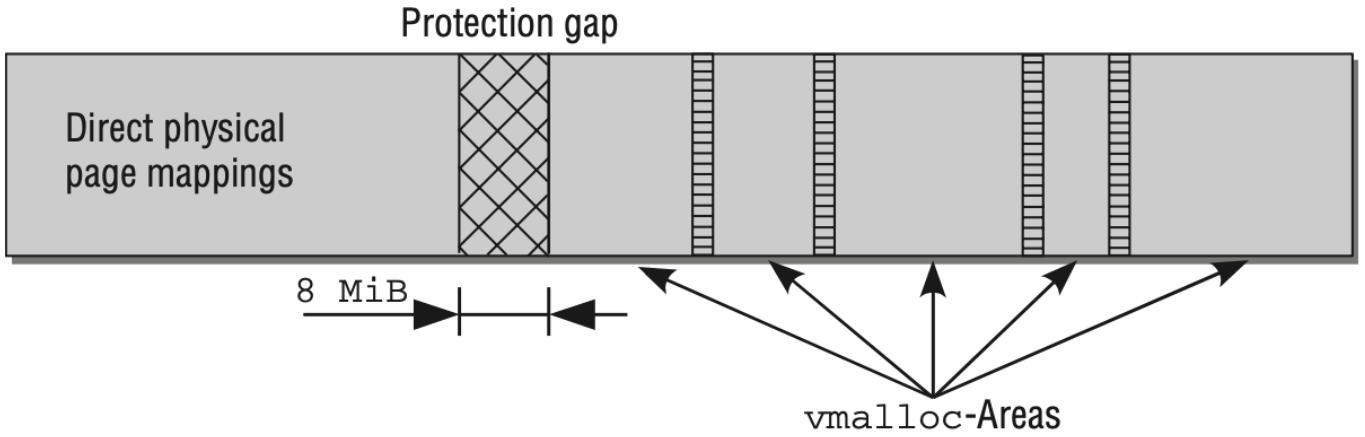


Figure 3-37: `vmalloc` area in the kernel's virtual address space on IA-32 systems.

Thank you!