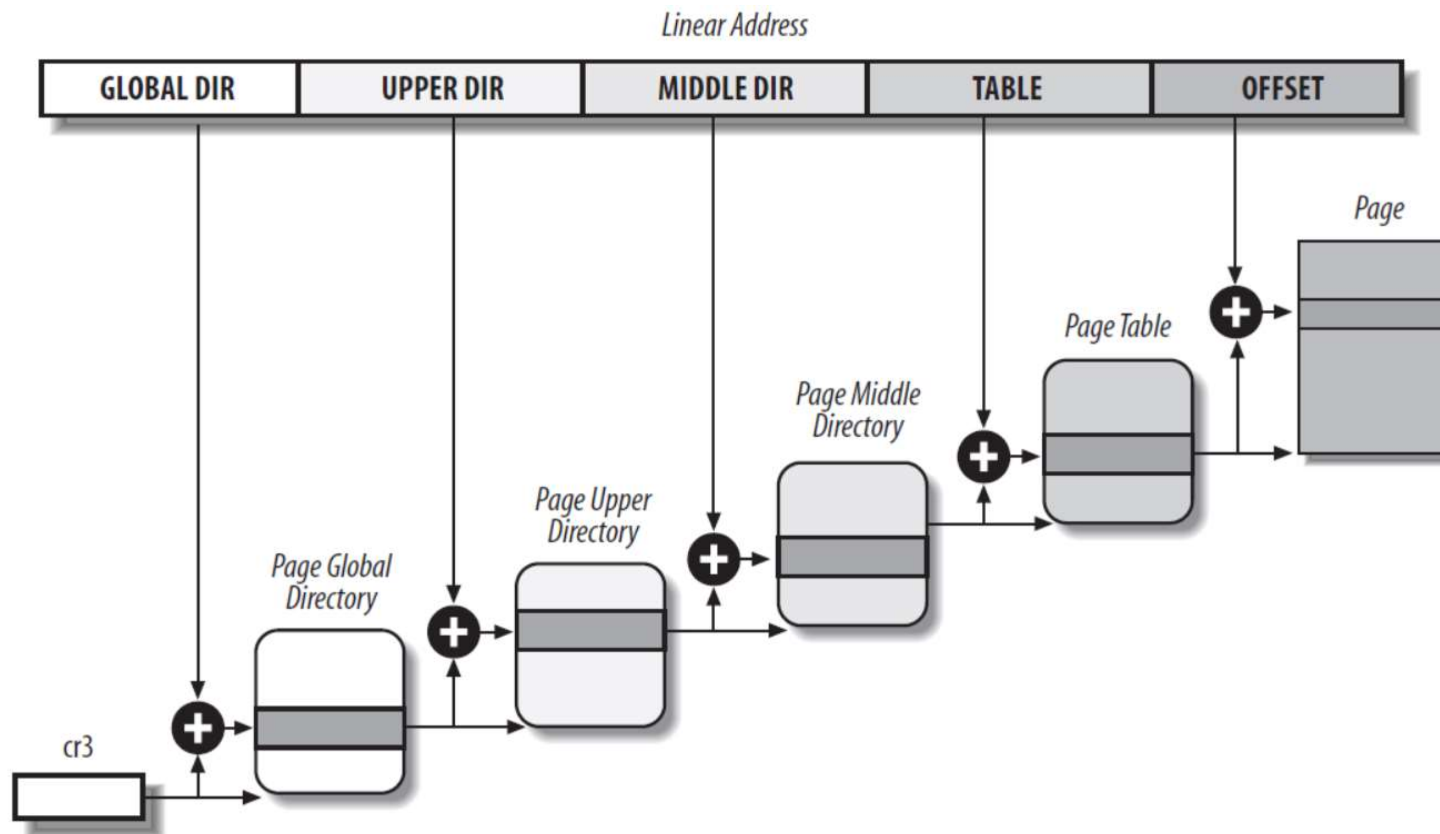
CSE 306 Operating Systems
Linux Memory Management

YoungMin Kwon

SUNY Korea
The State University of New York

Linux Paging Model

- Linux paging model



SUNY Korea

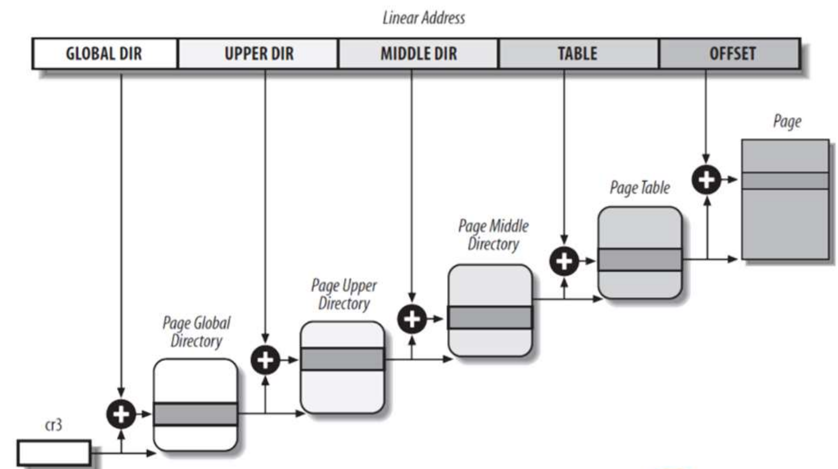
Functions for Page Walk

- Macros and functions for page walk

```
#define pgd_offset_pgd(pgd, address) (pgd + pgd_index((address)))
#define pgd_offset(mm, address) pgd_offset_pgd((mm)->pgd, (address))
```

5 level page table

```
static inline p4d_t *p4d_offset(pgd_t *pgd, unsigned long address);
static inline pud_t *pud_offset(p4d_t *p4d, unsigned long address);
static inline pmd_t *pmd_offset(pud_t *pud, unsigned long address);
static inline pte_t *pte_offset_kernel(pmd_t *pmd, unsigned long address);

static inline int pgd_none(pgd_t pgd);
static inline int p4d_none(p4d_t p4d);
static inline int pud_none(pud_t pud);
static inline int pmd_none(pmd_t pmd);
static inline int pte_none(pte_t pte);
```



SUNY Korea

Page Table Entry

- PTE related functions

```
#define pte_read(pte)        (pte_val(pte) & _PAGE_READ)
#define pte_write(pte)       (pte_val(pte) & _PAGE_WRITE)
#define pte_dirty(pte)       (pte_val(pte) & _PAGE_DIRTY)
#define pte_young(pte)       (pte_val(pte) & _PAGE_ACCESSED)
#define pte_special(pte)     (pte_val(pte) & _PAGE_SPECIAL)
…
```

SUNY Korea

Page Frame Management

- Page descriptor: struct page
  - State information of a page frame
  - All page descriptors are stored in mem_map array

- Some fields of struct page

Type	Name	Description
unsigned long	flags	Array of flags (see Table 8-2). Also encodes the zone number to which the page frame belongs.
atomic_t	_count	Page frame's reference counter.
unsigned long	private	Available to the kernel component that is using the page (for instance, it is a buffer head pointer in case of buffer page; see "Block Buffers and Buffer Heads" in Chapter 15). If the page is free, this field is used by the buddy system (see later in this chapter).
unsigned long	index	Used by several kernel components with different meanings. For instance, it identifies the position of the data stored in the page frame within the page's disk image or within an anonymous region (Chapter 15), or it stores a swapped-out page identifier (Chapter 17).
struct list_head	lru	Contains pointers to the least recently used doubly linked list of pages.

SUNY Korea

Page Descriptor

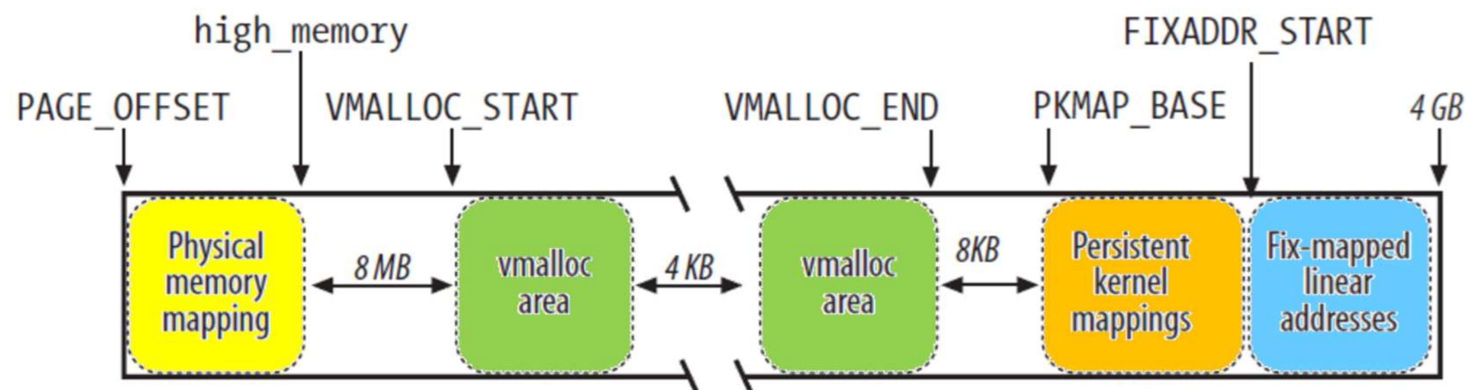
- Page descriptor: status of each page frame

```
struct page {
    unsigned long flags;
    ...
};

page->flags & (1L << PG_locked)     //page is locked
page->flags & (1L << PG_referenced) //page has been recently accessed
page->flags & (1L << PG_dirty)      //page has been modified
page->flags & (1L << PG_slab)       //page frame is included in a slab
…

//get page from pte
#define pte_page(pte)    pfn_to_page(pte_pfn(pte))
#define pte_pfn(pte)     (pte_val(pte) >> PAGE_SHIFT)

//conversion between pfn and page
#define __pfn_to_page(pfn)  (mem_map + pfn)
#define __page_to_pfn(page) ((unsigned long)((page) - mem_map))
```

SUNY Korea

Kernel Memory Layout Overview

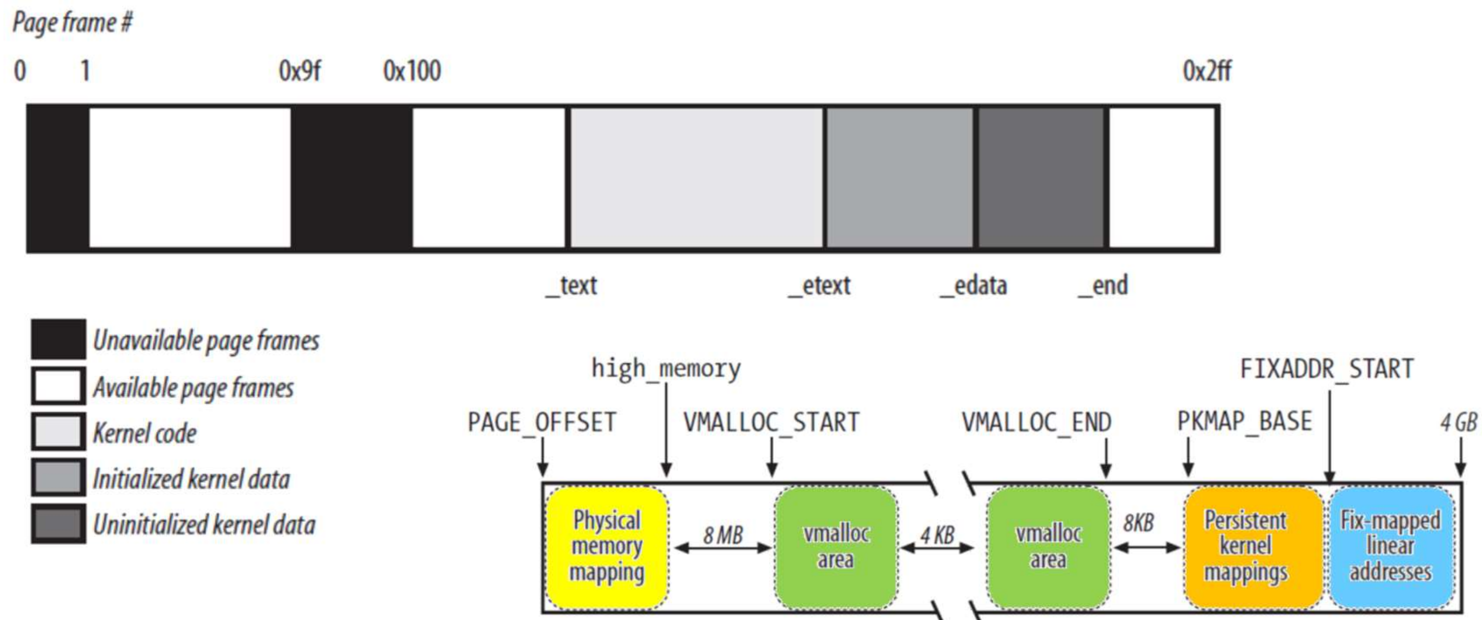
- Kernel memory layout
  - The first 896 MB of the 3rd GB is directly mapped physical memory
    - Kernel code and data
  - The remaining 128 MB of the 3rd GB is high memory
    - To access physical memory larger than linear address space



SUNY Korea

Directly Mapped Physical Memory

- About 3MB of Kernel code, initialized data, and uninitialized data are placed from the 2nd MB
- Virtual memory → Physical memory
  - Beginning of the 3rd GB (PAGE_OFFSET) (VA) → 0 (PA)
    - PA = VA – PAGE_OFFSET



SUNY Korea

Directly Mapped Physical Memory

- Conversion between VA and PA for directly mapped physical memory

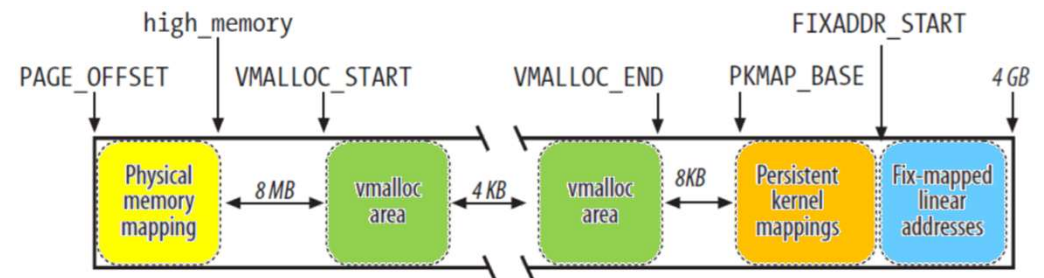
```
#define __va(x) ((void *)((unsigned long) (x)+PAGE_OFFSET))
#define __pa(x) ((unsigned long) (x)-PAGE_OFFSET)

#define virt_to_pfn(kaddr)  (__pa(kaddr) >> PAGE_SHIFT)
#define pfn_to_virt(pfn)    (__va((pfn)  << PAGE_SHIFT))

#define virt_to_page(addr)  pfn_to_page(virt_to_pfn(addr))
#define page_to_virt(page)  pfn_to_virt(page_to_pfn(page))

#define __pfn_to_page(pfn)  (mem_map + pfn)
#define __page_to_pfn(page) ((unsigned long)((page) - mem_map))

static inline unsigned long virt_to_phys(volatile void * address) {
    return __pa(address);
}
```

SUNY Korea

Fix-mapped Linear Address

- Mapping from VA to PA is fixed
  - Similar to directly mapped physical memory
  - But PA = VA – PAGE_OFFSET does not necessarily hold

```
enum fixed_addresses {   //each index maps one page frame
    FIX_HOLE,
    FIX_VSYSCALL,
    FIX_APIC_BASE,
    FIX_IO_APIC_BASE_0,
    ...
    __end_of_fixed_addresses
};
```



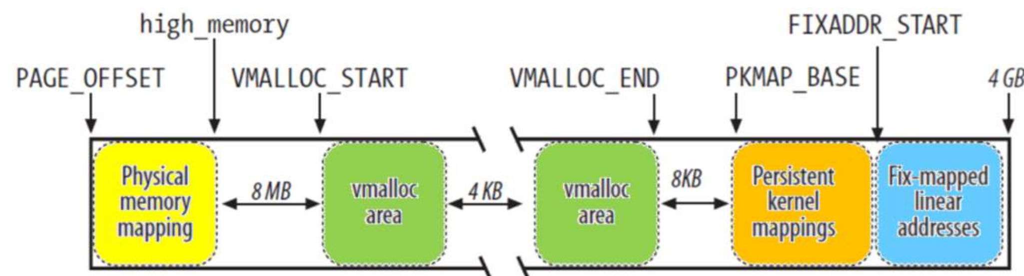

```
#define __fix_to_virt(x)   (FIXADDR_TOP - ((x) << PAGE_SHIFT))
```

SUNY Korea

Persistent Kernel Mapping

- Persistent kernel mapping has
  - Permanent mapping and temporary mapping

- Permanent kernel mapping
  - Long standing mapping of high-memory page frames into the kernel address space
  - Cannot be used in interrupt handlers
  - void* kmap(struct page *page) inserts the page frame address to a page table



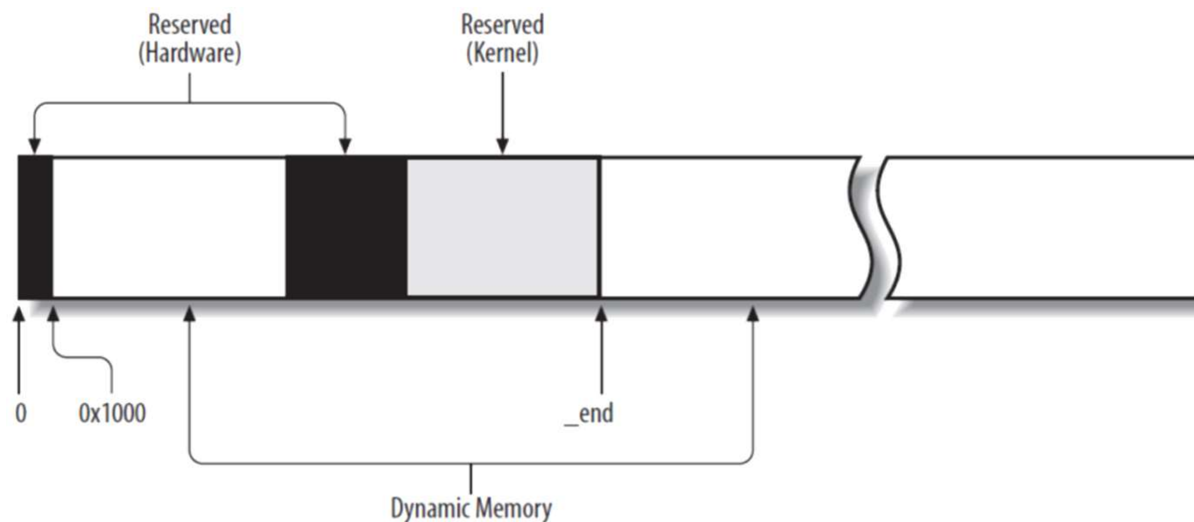
Persistent Kernel Mapping

- Temporary kernel mapping
  - Every page table in high memory can be mapped through a window
    - 13 windows are defined like KM_BOUNCE_READ, KM_USER0, KM_PTE0,…
    - The same window should never be used by two kernel paths at the same time.
  - void* kmap_atomic(struct page *page, enum km_type type) establishes a mapping through type window

SUNY Korea
The State University of New York

Dynamic Memory

- Dynamic Memory
  - Some portion of RAM is assigned to the kernel to store kernel code and kernel data structures
  - Dynamic memory: the remaining part of the RAM



SUNY Korea

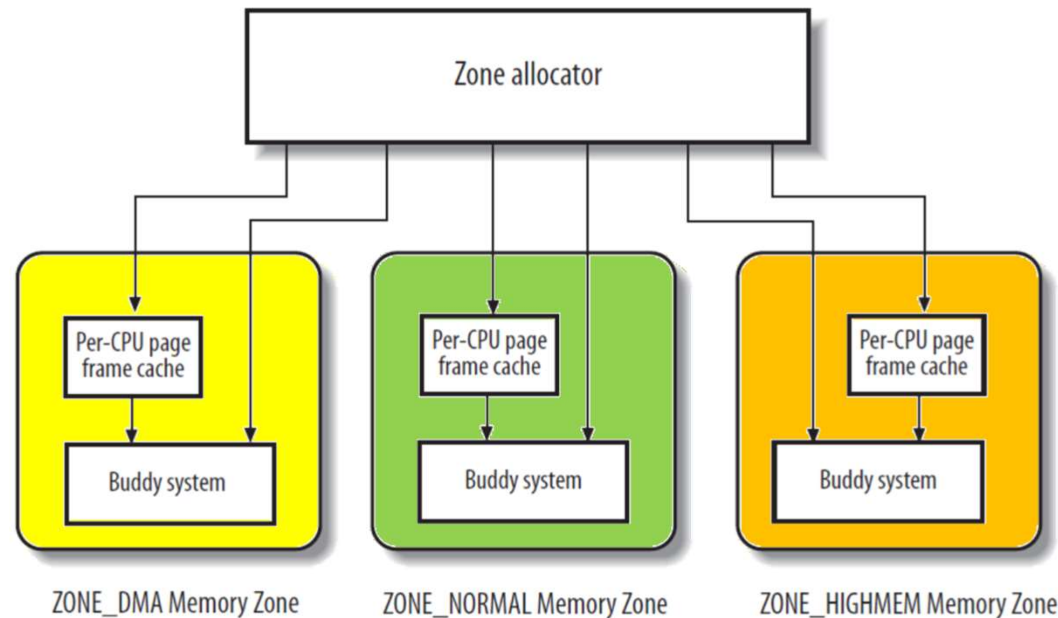
Memory Zones

- x86 hardware constraints in page frames
  - DMA processor for old ISA buses can address only the first 16 MB of RAM
  - In 32-bit computers with large RAM, physical memory address space (64GB) > virtual memory address space (4GB)

- 3 zones to cope with the limitations
  - ZONE_DMA: page frames below 16 MB
  - ZONE_NORMAL: 16 MB ~ 896 MB
  - ZONE_HIGHMEM: page frames above 896 MB (empty in 64bit architectures)

SUNY Korea

Zoned Page Frame Allocator



- Zone allocator
  - Searches for a memory zone that includes a group of contiguous page frames that can satisfy the request
  - Inside each zone, Buddy system algorithm is used

SUNY Korea

Zoned Page Frame Allocator

- Requesting and releasing pages frames
  - struct page* alloc_pages(gfp_mask, order): request 2^{order} contiguous page frames. Returns the address of the descriptor of the first allocated page frame
  - unsigned long __get_free_pages(gfp_mask, order): similar to alloc_pages, but returns the linear address of the first allocated page
  - __free_pages(page, order): decreases the count of the page descriptor; if count becomes 0, it assumes that 2^{order} contiguous frames from the page is no longer used
  - free_pages(addr, order): similar to __free_pages, but takes the linear address of the first page frame to be released
  - Methods involving linear addresses does not work in high memory area

SUNY Korea

Zoned Page Frame Allocator

- Some flags for gfp_mask

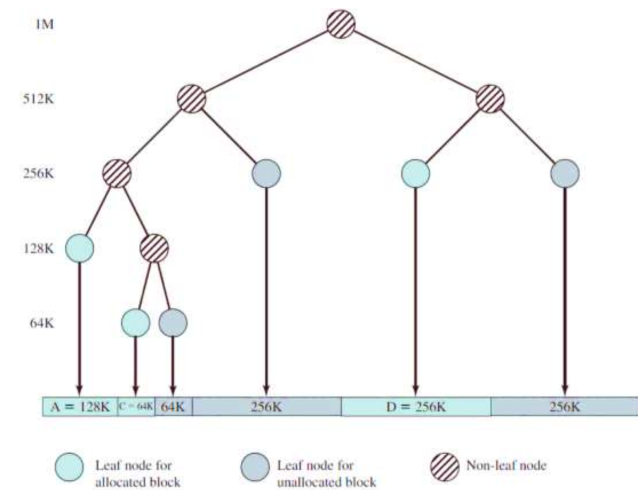
Flag	Description
__GFP_DMA	The page frame must belong to the ZONE_DMA memory zone. Equivalent to GFP_DMA.
__GFP_HIGHMEM	The page frame may belong to the ZONE_HIGHMEM memory zone.
__GFP_WAIT	The kernel is allowed to block the current process waiting for free page frames.
__GFP_HIGH	The kernel is allowed to access the pool of reserved page frames.
__GFP_IO	The kernel is allowed to perform I/O transfers on low memory pages in order to free page frames.
__GFP_FS	If clear, the kernel is not allowed to perform filesystem-dependent operations.

Group name	Corresponding flags
GFP_ATOMIC	__GFP_HIGH
GFP_NOIO	__GFP_WAIT
GFP_NOFS	__GFP_WAIT | __GFP_IO
GFP_KERNEL	__GFP_WAIT | __GFP_IO | __GFP_FS
GFP_USER	__GFP_WAIT | __GFP_IO | __GFP_FS
GFP_HIGHUSER	__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_HIGHMEM

SUNY Korea

Buddy System

- Buddy system algorithm
  - 11 different size groups:
    1, 2, 4, …, 1024 pages
  - struct zone has struct free_area free_area[11] array
  - struct free_area has nr_free and free_list fields
  - private fields of struct page stores the order of the block



SUNY Korea

Buddy System Algorithm (Allocation)

```
static struct page *__rmqueue(struct zone *zone, unsigned int order) {
    …
    struct free_area *area;
    unsigned int current_order;
    for (current_order = order; current_order < 11; ++current_order) {

        area = &zone->free_area[current_order];

        if (!list_empty(&area->free_list))
            goto block_found;
    }
    return NULL;
    …
```

- Look for a contiguous free block that can satisfy the request

SUNY Korea

Buddy System Algorithm (Allocation)

```
block_found:
    page = list_entry(area->free_list.next, struct page, lru);
    list_del(&page->lru);   //remove the block from the list

    ClearPagePrivate(page); //clear flag: not using the private field
    page->private = 0;

    area->nr_free--;        //decrease the # of free blocks
    zone->free_pages -= 1UL << order; //reduce the # of free pages
```

- If a block is found, remove it from the free_list of the area and decrease its nr_free
- Decrease free_pages count of zone by 2^{order}

Buddy System Algorithm (Allocation)

```
size = 1 << curr_order;
while (curr_order > order) {
    area--;                       //area ← &free_area[current_order - 1]
    curr_order--;
    size >>= 1;                   //size = size / 2
    buddy = page + size;          //start page of the buddy block

    /* insert buddy as first element in the list */
    list_add(&buddy->lru, &area->free_list); //add buddy to free_list
    area->nr_free++;              //area has one more free block

    buddy->private = curr_order; //free block size at the first page
    SetPagePrivate(buddy);        //set flag: using the private field
}
return page;
```

- If the block found is larger than the request, split the blocks and add the remaining (buddy) to the corresponding free list

SUNY Korea

Buddy System Algorithm (Free)

```
static inline void
__free_pages_bulk(struct page *page, struct zone *zone, unsigned int order)
{
…
    struct page  *base = zone->zone_mem_map; //first page of the zone
    unsigned long buddy_idx, page_idx = page - base;
    struct page  *buddy, *coalesced;

    //increase the free page size of the zone
    int order_size = 1 << order;
    zone->free_pages += order_size;
…
```

- Compute the page index page_idx starting from the first page of the zone
- Increase the free page count of the zone

Buddy System Algorithm (Free)

```
while (order < 10) {
    buddy_idx = page_idx ^ (1 << order); //page_idx +/- order_size
    buddy = base + buddy_idx;            //page of the buddy
    if (!page_is_buddy(buddy, order))    //break if we cannot coalesce
        break;

    list_del(&buddy->lru); //remove buddy from the free_list[] of free_area
    zone->free_area[order].nr_free--;

    ClearPagePrivate(buddy); //not using the private field
    buddy->private = 0;

    page_idx &= buddy_idx;   //the smaller of page_idx and buddy_idx
    order++;
}
```

- Coalesce with buddies

Buddy System Algorithm (Free)

```
coalesced = base + page_idx; //find the first page of coalesced block

coalesced->private = order;  //update its private field with the order
SetPagePrivate(coalesced);

/* insert the first page in the free_list of the corresponding free_area*/
list_add(&coalesced->lru, &zone->free_area[order].free_list);
zone->free_area[order].nr_free++;
```

- Update the first page of the coalesced free block
- Insert the block in the free_list of the free_area

SUNY Korea

Memory Area Management

- Memory area
  - A sequence of memory cells having contiguous physical address and an arbitrary length
- Buddy system algorithm is good for relatively large memory requests
  - How about few tens or hundreds of bytes?

SUNY Korea

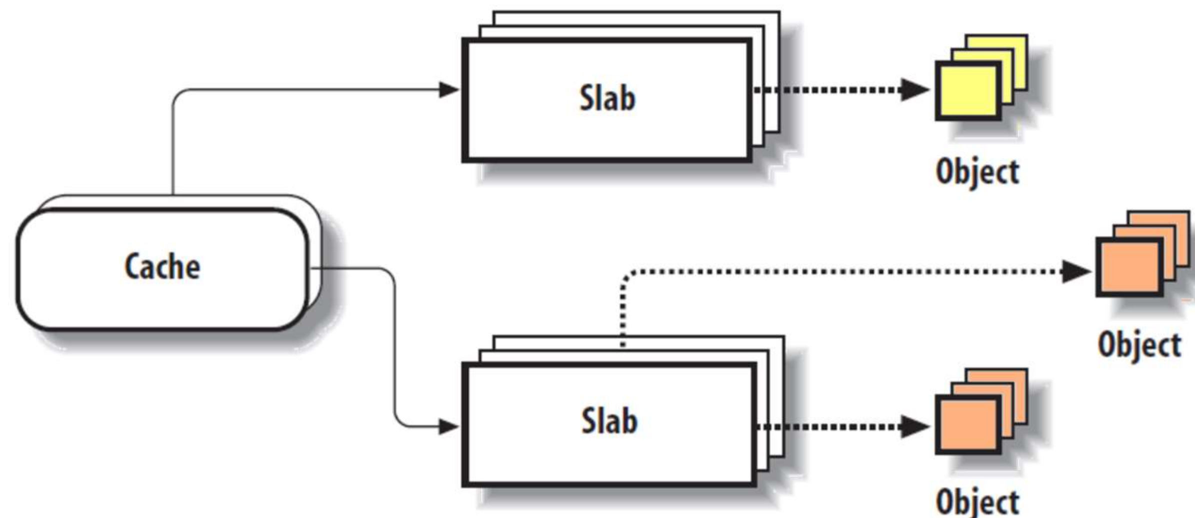
The Slab Allocator

- Premises
  - Views memory as objects
    - To avoid initializing objects repeatedly, objects are stored in memory when they are freed
  - Kernel functions tend to request memory area of the same type repeatedly
    - E.g. task_struct, fs_struct, file_struct, …
  - Frequent request of a particular size
    - Handled efficiently by creating a set of special objects of the right size

SUNY Korea

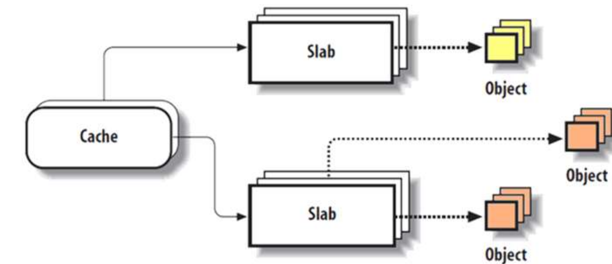
The Slab Allocator

- Slab allocator groups objects into caches
  - Each cache is a store of objects of the same type
  - Each slab consists of one or more contiguous page frames for both allocated and free objects



SUNY Korea

Slab Allocator: Cache Descriptor



- Some fields of kmem_cache_t

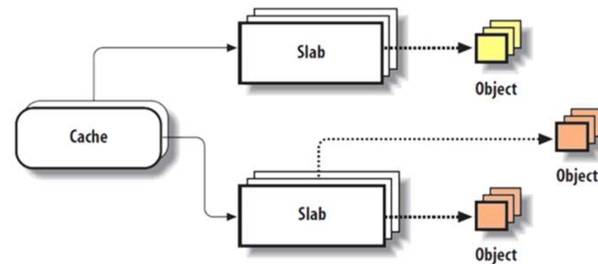
Type	Name	Description
struct array_cache * []	array	Per-CPU array of pointers to local caches of free objects (see the section "Local Caches of Free Slab Objects" later in this chapter).
struct kmem_list3	lists	See next table.
unsigned int	objsize	Size of the objects included in the cache.
unsigned int	flags	Set of flags that describes permanent properties of the cache.
unsigned int	gfporder	Logarithm of the number of contiguous page frames included in a single slab.
unsigned int	gfpflags	Set of flags passed to the buddy system function when allocating page frames.
kmem_cache_t *	slabp_cache	Pointer to the general slab cache containing the slab descriptors (NULL if internal slab descriptors are used; see next section).
struct list_head	next	Pointers for the doubly linked list of cache descriptors.

SUNY Korea

Slab Allocator: Cache Descriptor

- Some fields of the kmem_list3 structure

Type	Name	Description
struct list_head	slabs_partial	Doubly linked circular list of slab descriptors with both free and non-free objects
struct list_head	slabs_full	Doubly linked circular list of slab descriptors with no free objects
struct list_head	slabs_free	Doubly linked circular list of slab descriptors with free objects only
unsigned long	free_objects	Number of free objects in the cache



SUNY Korea

Slab Allocator: Slab Descriptor

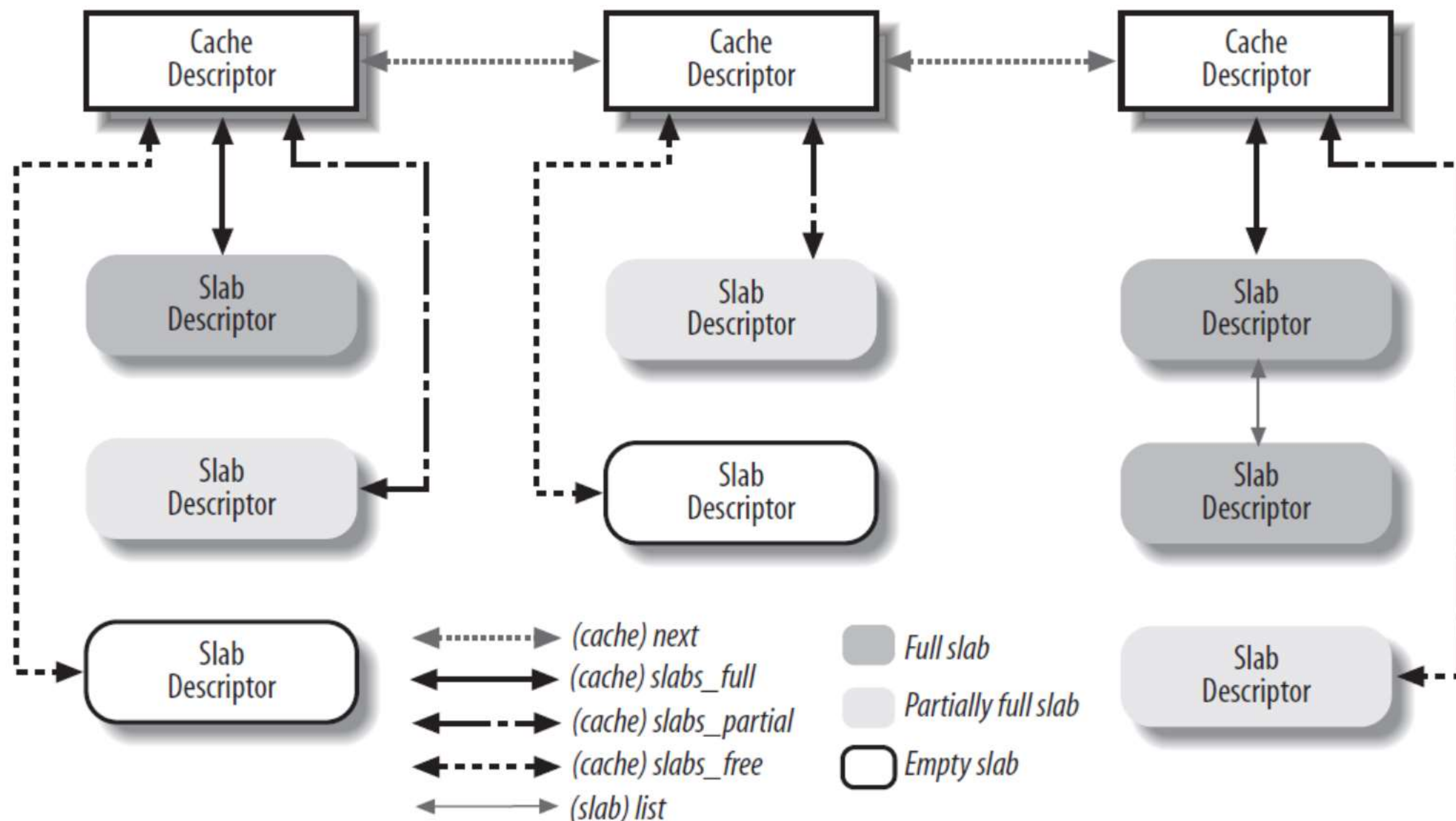
- Some fields of the slab type

Type	Name	Description
struct list_head	list	Pointers for one of the three doubly linked list of slab descriptors (either the slabs_full, slabs_partial, or slabs_free list in the kmem_list3 structure of the cache descriptor)
unsigned long	colouroff	Offset of the first object in the slab (see the section "Slab Coloring" later in this chapter)
void *	s_mem	Address of first object (either allocated or free) in the slab
unsigned int	inuse	Number of objects in the slab that are currently used (not free)
unsigned int	free	Index of next free object in the slab, or BUFCTL_END if there are no free objects left (see the section "Object Descriptor" later in this chapter)

SUNY Korea

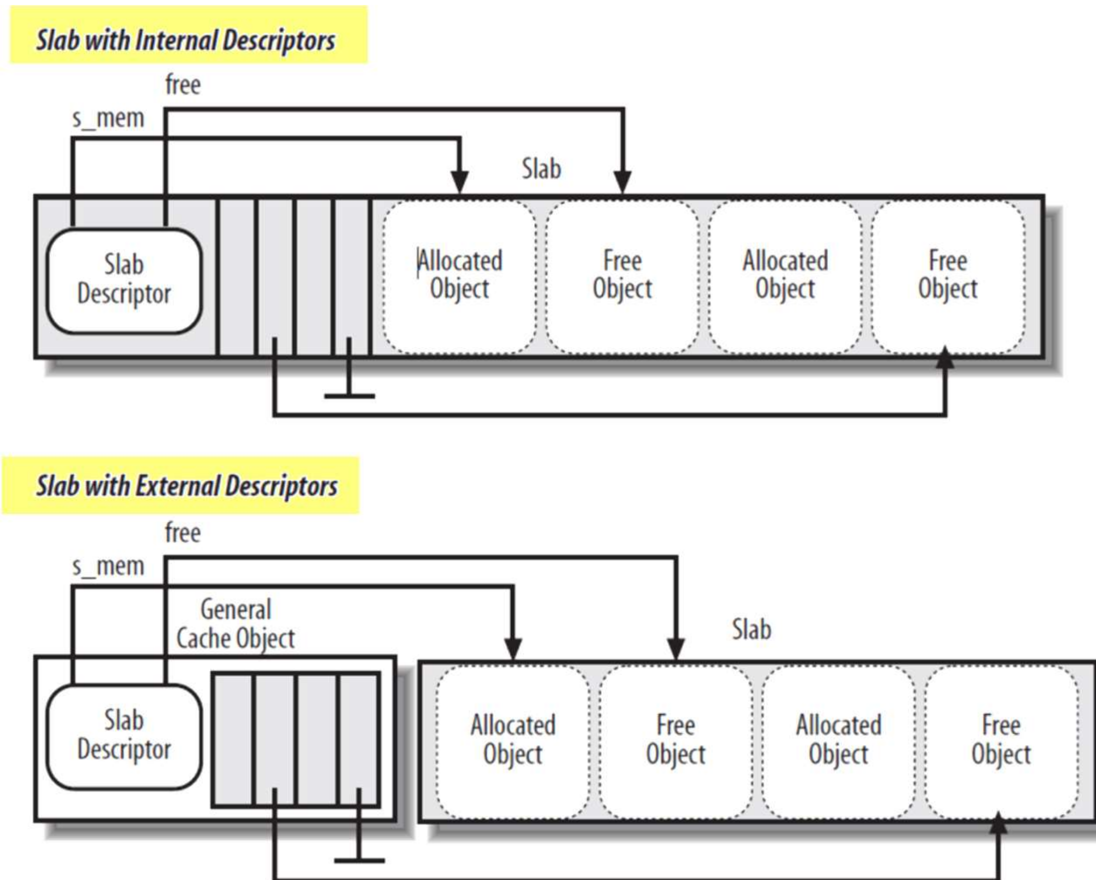
Slab Allocator: Slab Descriptor

- Relationship among cache and slab descriptors



SUNY Korea

Slab and Object Descriptors



- Right after slab descriptors are object descriptors
  - Each object descriptor is an index of the next free object

SUNY Korea

Creating a New Slab

```
void *kmem_getpages(kmem_cache_t *cachep, int flags) {
    struct page *page;
    int i;

    flags |= cachep->gfpflags;
    page = alloc_pages(flags, cachep->gfporder);
    if (!page)
        return NULL;

    i = (1 << cache->gfporder);
    if (cachep->flags & SLAB_RECLAIM_ACCOUNT)
        atomic_add(i, &slab_reclaim_pages);

    while (i-- >= 0)
        SetPageSlab(page+i);

    return page_address(page);
}
```

SUNY Korea

Releasing a Slab

```
void kmem_freepages(kmem_cache_t *cachep, void *addr) {
    unsigned long i = (1 << cachep->gfporder);
    struct page *page = virt_to_page(addr);

    if (current->reclaim_state)
        current->reclaim_state->reclaimed_slab += i;

    while (i--)
        ClearPageSlab(page++);

    free_pages((unsigned long)addr, cachep->gfporder);

    if (cachep->flags & SLAB_RECLAIM_ACCOUNT)
        atomic_sub(1 << cachep->gfporder, &slab_reclaim_pages);
}
```

SUNY Korea

Allocating/Freeing a Slab Object

```
//allocate a slab object
void *kmem_cache_alloc(kmem_cache_t *cachep, int flags);

//free a slab object
void kmem_cache_free(kmem_cache_t *cachep, void *objp);
```

SUNY Korea
The State University of New York

Allocating General Purpose Objects

```
void *kmalloc(size_t size, int flags) {
    struct cache_sizes *csizep = malloc_sizes;
    kmem_cache_t * cachep;

    for (; csizep->cs_size; csizep++) {
        if (size > csizep->cs_size)
            continue;

        if (flags & __GFP_DMA)
            cachep = csizep->cs_dmacachep;
        else
            cachep = csizep->cs_cachep;

        return kmem_cache_alloc(cachep, flags);
    }
    return NULL;
}
```

- malloc_sizes points to 13 geometrically distributed sizes (32, 64, …, 131,702)

SUNY Korea
The State University of New York

Freeing General Purpose Objects

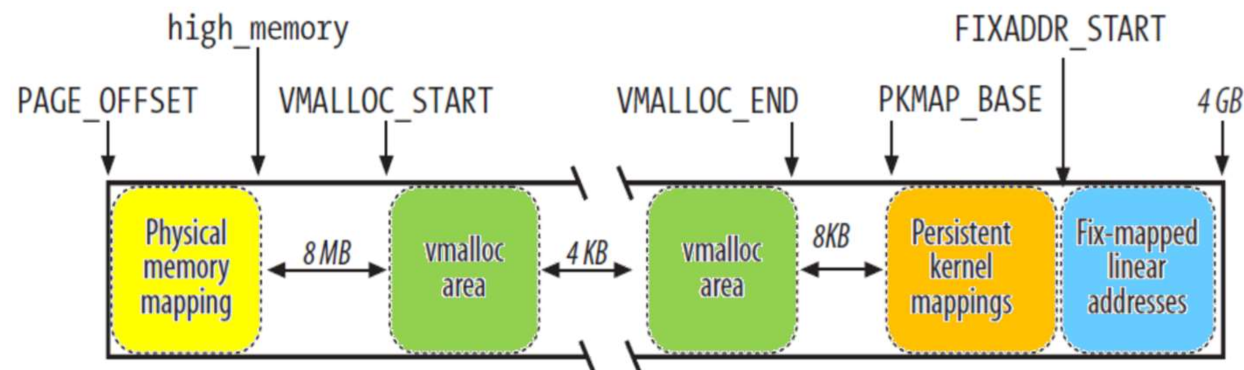
```
void kfree(const void *objp) {
    kmem_cache_t * c;
    unsigned long flags;
    if (!objp)
        return;

    local_irq_save(flags);
    c = (kmem_cache_t *)(virt_to_page(objp)->lru.next);

    kmem_cache_free(c, (void *)objp);
    local_irq_restore(flags);
}
```

lru.next of page descriptors points to its cache descriptor

Noncontiguous Memory Area

- Vmalloc/vfree
  - Allocate/free physically noncontiguous memory
  - Using page tables, they are virtually contiguous

```
void *vmalloc(unsigned long size);

void vfree(const void *addr);
```



SUNY Korea