

메모리 관리

메모리 관리 기법과 가상 메모리

메모리 관리 기법과 가상 메모리

- 가상 메모리

최초의 컴퓨터부터 우리는
물리적으로 존재하는 메모리보다
더 많은 양의 메모리를 요구했다.



가상 메모리(Virtual Memory) 도입

메모리 관리 기법과 가상 메모리

- 가상 메모리

- 가상 메모리는 실제 시스템에 존재하는 **물리 메모리의 크기와 관계없이** 가상적인 주소공간을 사용자 task에게 제공한다.
- 32bit CPU의 경우 2^{32} (4GB) 크기 만큼,
64bit CPU의 경우 2^{64} (16EB) 크기 만큼 제공한다.
- 해당 크기 만큼의 메모리를 전부 제공하는 것이 아닌 개념적인 공간이며 **실제로는 필요한 만큼의 물리 메모리를 제공한다.**

물리 메모리 관리 자료 구조

메모리 관리 기법과 가상 메모리

- 물리 메모리 관리 자료 구조

- **SMP**

복수 개의 CPU를 가지고 있는 컴퓨터 시스템 중
모든 CPU가 메모리와 입출력 버스를 공유하는 구조

- **NUMA**

복수 개의 CPU를 몇 개의 그룹으로 나눈 뒤
각각의 그룹에 별도의 지역 메모리를 주는 구조

- **UMA**

NUMA와 반대로 모든 CPU가 하나의 메모리를 공유하는 구조

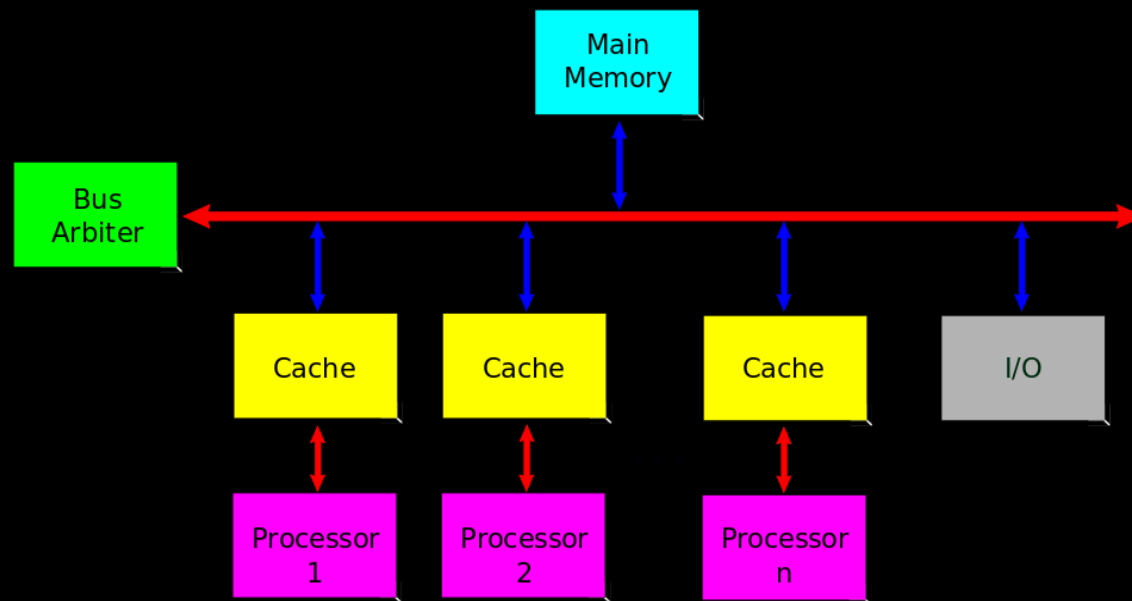
메모리 관리 기법과 가상 메모리

- 물리 메모리 관리 자료 구조

- SMP의 문제점

복수 개의 CPU가 메모리 등의 자원을
공유하기 때문에

병목현상 발생 가능성이 있음.



메모리 관리 기법과 가상 메모리

- 물리 메모리 관리 자료 구조

Solution : **NUMA (Non-Uniform Memory Access)**
CPU들을 몇 개의 그룹으로 분할, 그룹에게 지역메모리 할당



UMA (Uniform Memory Access)
모든 CPU가 하나의 메모리를 공유

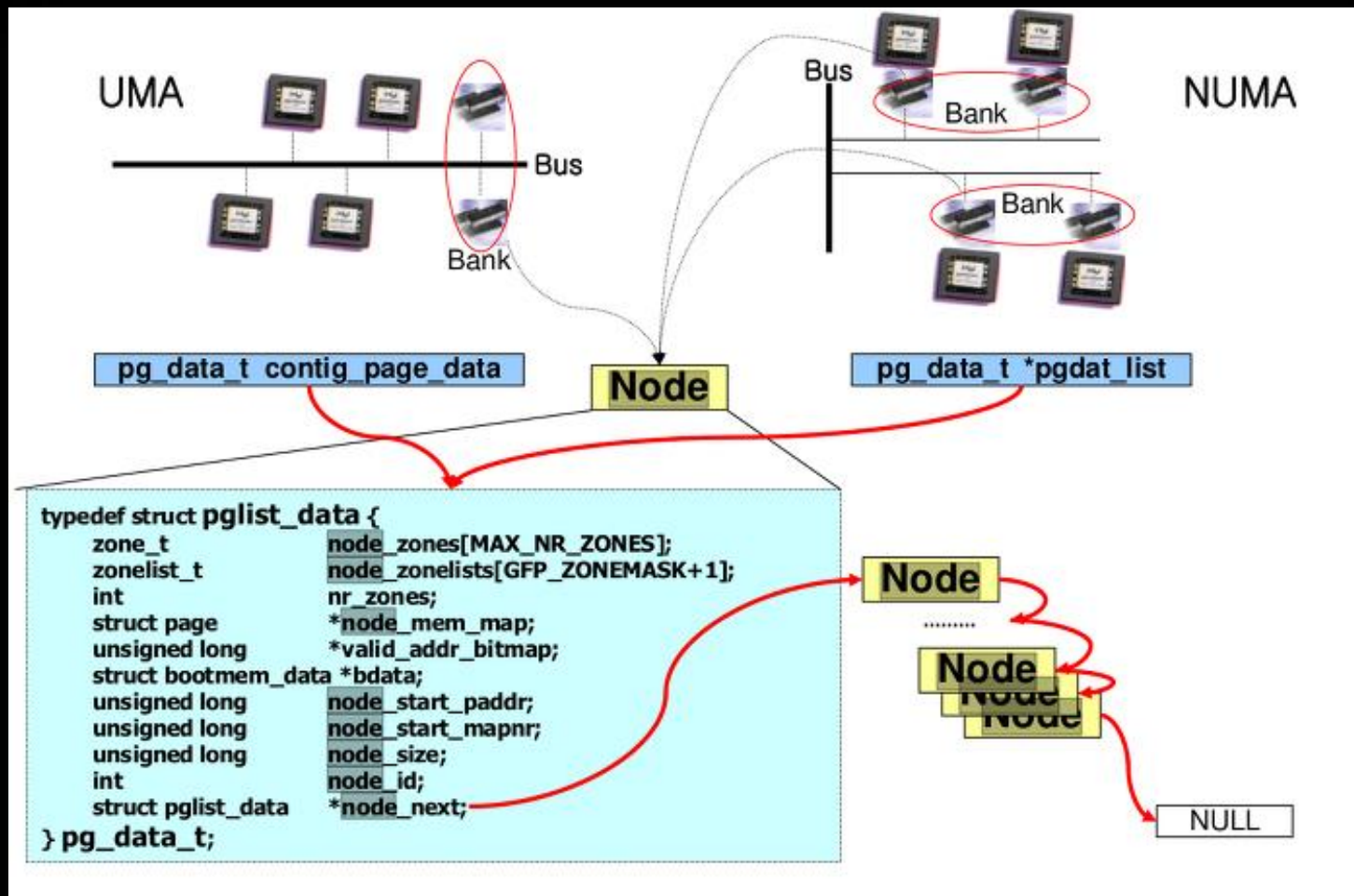
메모리 관리 기법과 가상 메모리

- 물리 메모리 관리 자료 구조

- **Bank**
접근 속도가 같은 메모리의 집합
- **Node**
Bank를 표현하는 구조

메모리 관리 기법과 가상 메모리

- 물리 메모리 관리 자료 구조



```

typedef struct pglist_data {
    struct zone node_zones[MAX_NR_ZONES];
    struct zonelist node_zonelists[MAX_ZONELISTS];
    int nr_zones;
#ifdef CONFIG_FLAT_NODE_MEM_MAP /* means !SPARSEMEM */
    struct page *node_mem_map;
#endif
#ifdef CONFIG_MEMCG
    struct page_cgroup *node_page_cgroup;
#endif
#ifdef CONFIG_NO_BOOTMEM
    struct bootmem_data *bdata;
#endif
#ifdef CONFIG_MEMORY_HOTPLUG

    spinlock_t node_size_lock;
#endif
    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;
    wait_queue_head_t kswapd_wait;
    wait_queue_head_t pfmemalloc_wait;
    struct task_struct *kswapd; /* Protected by
                                mem_hotplug_begin/end() */
    int kswapd_max_order;
    enum zone_type classzone_idx;
#ifdef CONFIG_NUMA_BALANCING
    /* Lock serializing the migrate rate limiting window */
    spinlock_t numabalancing_migrate_lock;

    /* Rate limiting time interval */
    unsigned long numabalancing_migrate_next_window;

    /* Number of pages migrated during the rate limiting time interval */
    unsigned long numabalancing_migrate_nr_pages;
#endif
} pg_data_t;

```

해당 물리 메모리가 속한
메모리 맵의 번지
(Page frame number)

```

typedef struct pglist_data {
    struct zone node_zones[MAX_NR_ZONES];
    struct zonelist node_zonelists[MAX_ZONELISTS];
    int nr_zones;
#ifdef CONFIG_FLAT_NODE_MEM_MAP /* means !SPARSEMEM */
    struct page *node_mem_map;
#endif
#ifdef CONFIG_MEMCG
    struct page_cgroup *node_page_cgroup;
#endif
#ifdef CONFIG_NO_BOOTMEM
    struct bootmem_data *bdata;
#endif
#ifdef CONFIG_MEMORY_HOTPLUG

    spinlock_t node_size_lock;
#endif
    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;
    wait_queue_head_t kswapd_wait;
    wait_queue_head_t pfmemalloc_wait;
    struct task_struct *kswapd; /* Protected by
                                mem_hotplug_begin/end() */
    int kswapd_max_order;
    enum zone_type classzone_idx;
#ifdef CONFIG_NUMA_BALANCING
    /* Lock serializing the migrate rate limiting window */
    spinlock_t numabalancing_migrate_lock;

    /* Rate limiting time interval */
    unsigned long numabalancing_migrate_next_window;

    /* Number of pages migrated during the rate limiting time interval */
    unsigned long numabalancing_migrate_nr_pages;
#endif
} pg_data_t;

```

해당 노드에 속해있는 물리 메모리의
실제 양

```

typedef struct pglist_data {
    struct zone node_zones[MAX_NR_ZONES];
    struct zonelist node_zonelists[MAX_ZONELISTS];
    int nr_zones;
#ifdef CONFIG_FLAT_NODE_MEM_MAP /* means !SPARSEMEM */
    struct page *node_mem_map;
#endif
#ifdef CONFIG_MEMCG
    struct page_cgroup *node_page_cgroup;
#endif
#ifdef CONFIG_NO_BOOTMEM
    struct bootmem_data *bdata;
#endif
#ifdef CONFIG_MEMORY_HOTPLUG

    spinlock_t node_size_lock;
#endif
    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;
    wait_queue_head_t kswapd_wait;
    wait_queue_head_t pfmemalloc_wait;
    struct task_struct *kswapd; /* Protected by
                                mem_hotplug_begin/end() */
    int kswapd_max_order;
    enum zone_type classzone_idx;
#ifdef CONFIG_NUMA_BALANCING
    /* Lock serializing the migrate rate limiting window */
    spinlock_t numabalancing_migrate_lock;

    /* Rate limiting time interval */
    unsigned long numabalancing_migrate_next_window;

    /* Number of pages migrated during the rate limiting time interval */
    unsigned long numabalancing_migrate_nr_pages;
#endif
} pg_data_t;

```

zone 구조체를 담기 위한 배열

```

typedef struct pglist_data {
    struct zone node_zones[MAX_NR_ZONES];
    struct zonelist node_zonelists[MAX_ZONELISTS];
    int nr_zones;
#ifdef CONFIG_FLAT_NODE_MEM_MAP /* means !SPARSEMEM */
    struct page *node_mem_map;
#endif
#ifdef CONFIG_MEMCG
    struct page_cgroup *node_page_cgroup;
#endif
#ifdef CONFIG_NO_BOOTMEM
    struct bootmem_data *bdata;
#endif
#ifdef CONFIG_MEMORY_HOTPLUG

    spinlock_t node_size_lock;
#endif
    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;
    wait_queue_head_t kswapd_wait;
    wait_queue_head_t pfmemalloc_wait;
    struct task_struct *kswapd; /* Protected by
                                mem_hotplug_begin/end() */
    int kswapd_max_order;
    enum zone_type classzone_idx;
#ifdef CONFIG_NUMA_BALANCING
    /* Lock serializing the migrate rate limiting window */
    spinlock_t numabalancing_migrate_lock;

    /* Rate limiting time interval */
    unsigned long numabalancing_migrate_next_window;

    /* Number of pages migrated during the rate limiting time interval */
    unsigned long numabalancing_migrate_nr_pages;
#endif
} pg_data_t;

```

zone의 개수

메모리 관리 기법과 가상 메모리

- 물리 메모리 관리 자료 구조

DMA (Direct Memory Access)

CPU에 연결된 Device가 Bus(ISA, EISA, AXI, ...)를 통해

직접적으로 메모리에 r/w 작업하기 위한 공간

→ ZONE_DMA

메모리 관리 기법과 가상 메모리

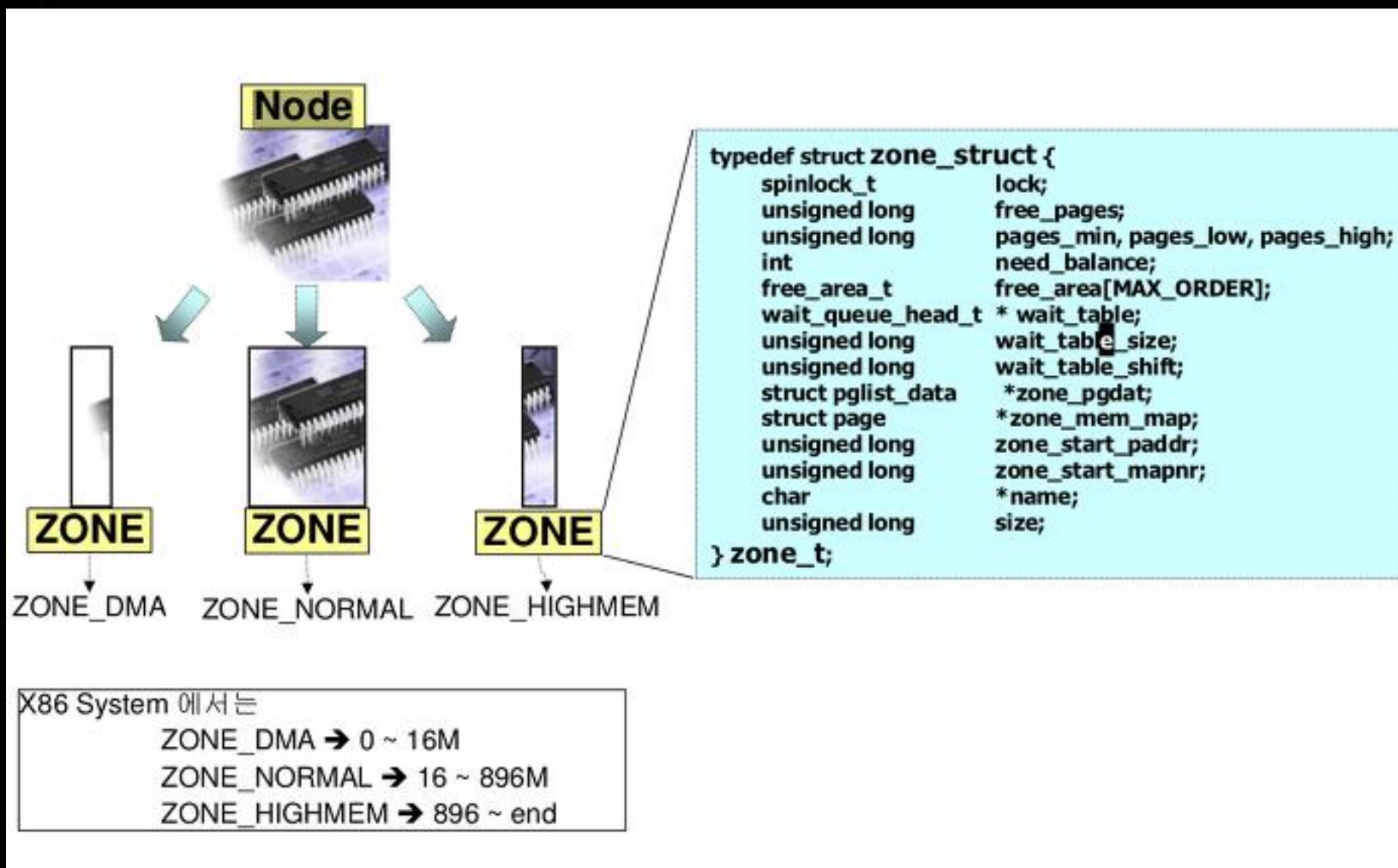
- 물리 메모리 관리 자료 구조

물리 메모리가 아키텍처의 가상 주소에 1:1로 미리 매핑되어 사용할 수 있는 만큼의 영역.

→ ZONE_NORMAL

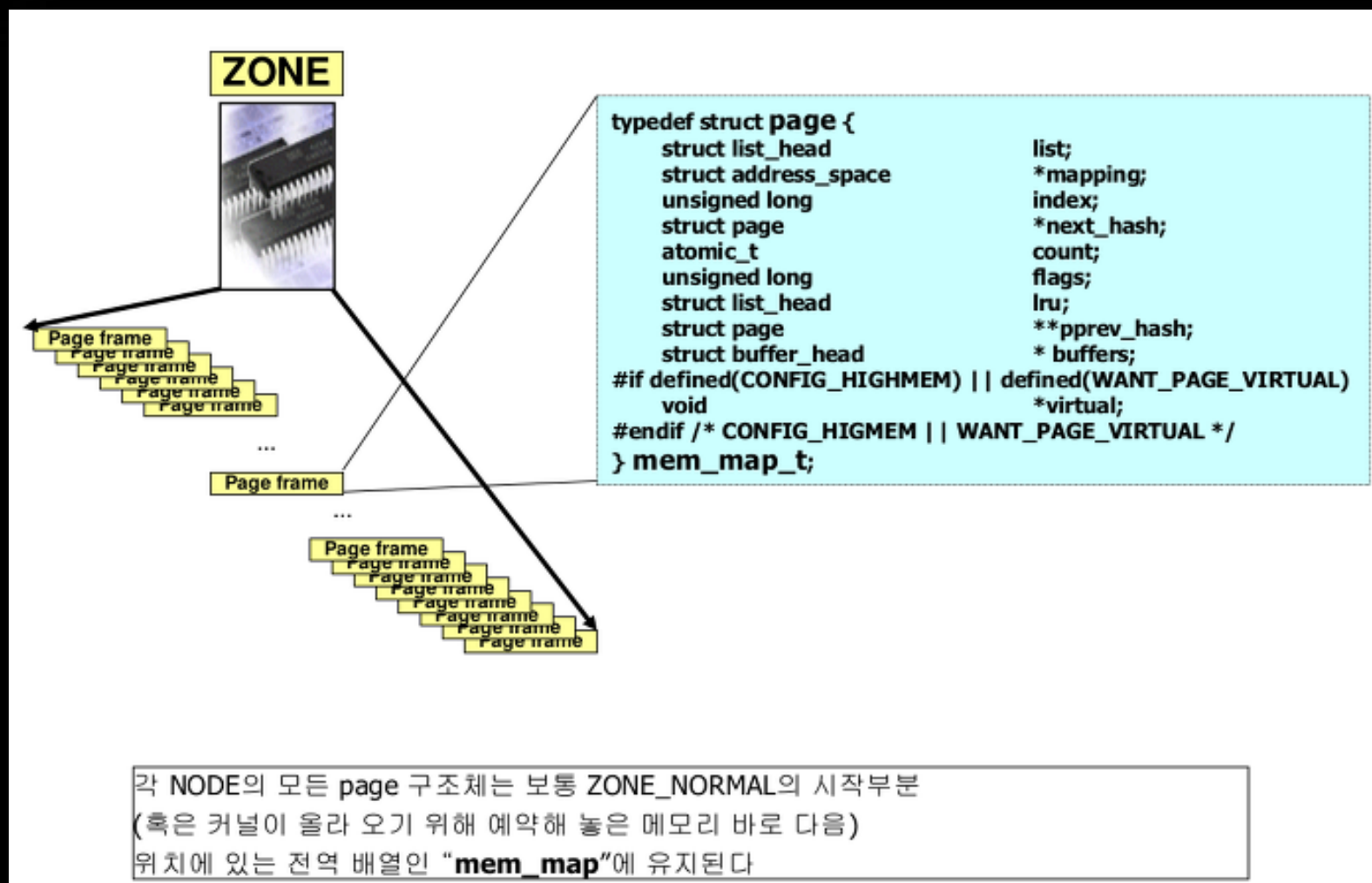
메모리 관리 기법과 가상 메모리

- 물리 메모리 관리 자료 구조



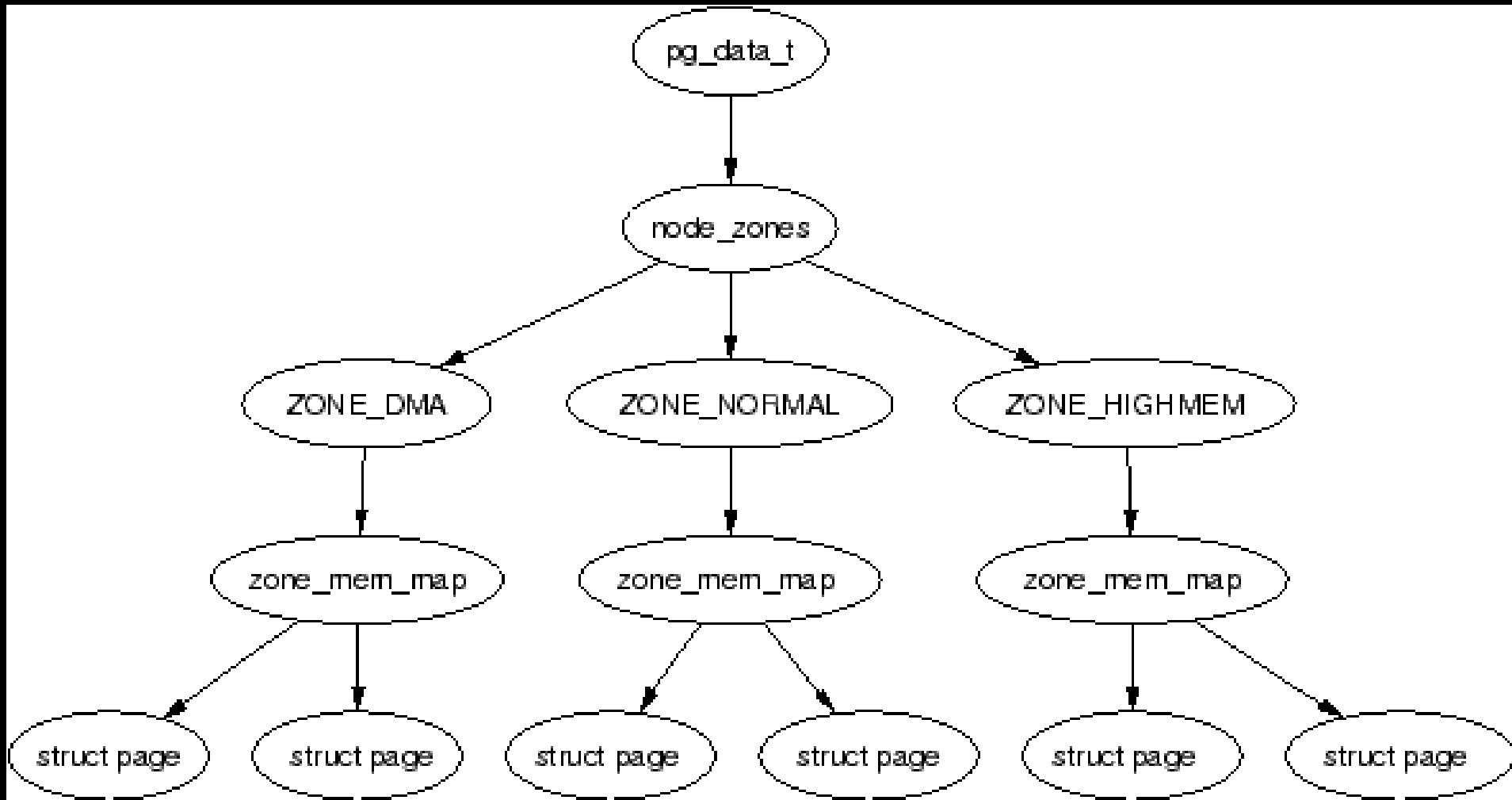
메모리 관리 기법과 가상 메모리

- 물리 메모리 관리 자료 구조



메모리 관리 기법과 가상 메모리

- 물리 메모리 관리 자료 구조




Buddy와 Slab

메모리 관리 기법과 가상 메모리

- Buddy와 Slab → Buddy

```
struct zone {  
    ...  
  
    ZONE_PADDING(_pad1_)  
  
    /* free areas of different sizes */  
    struct free_area    free_area[MAX_ORDER];  
  
    /* zone flags, see below */  
    unsigned long      flags;  
  
    /* Primarily protects free_area */  
    spinlock_t         lock;  
    ...  
};
```

```
struct free_area {  
    struct list_head    free_list;  
    unsigned long       *map;  
};
```



Buddy 할당자는 struct zone의 struct free_area 배열인 free_area[] 를 통해 구축하며 10개의 엔트리가 있으며 0~9 까지의 엔트리가 각 2^n 개의 Page frame을 가진다.

(0 = 4KB, 1 = 8KB, 2 = 16KB, ...)

메모리 관리 기법과 가상 메모리

- Buddy와 Slab → Buddy

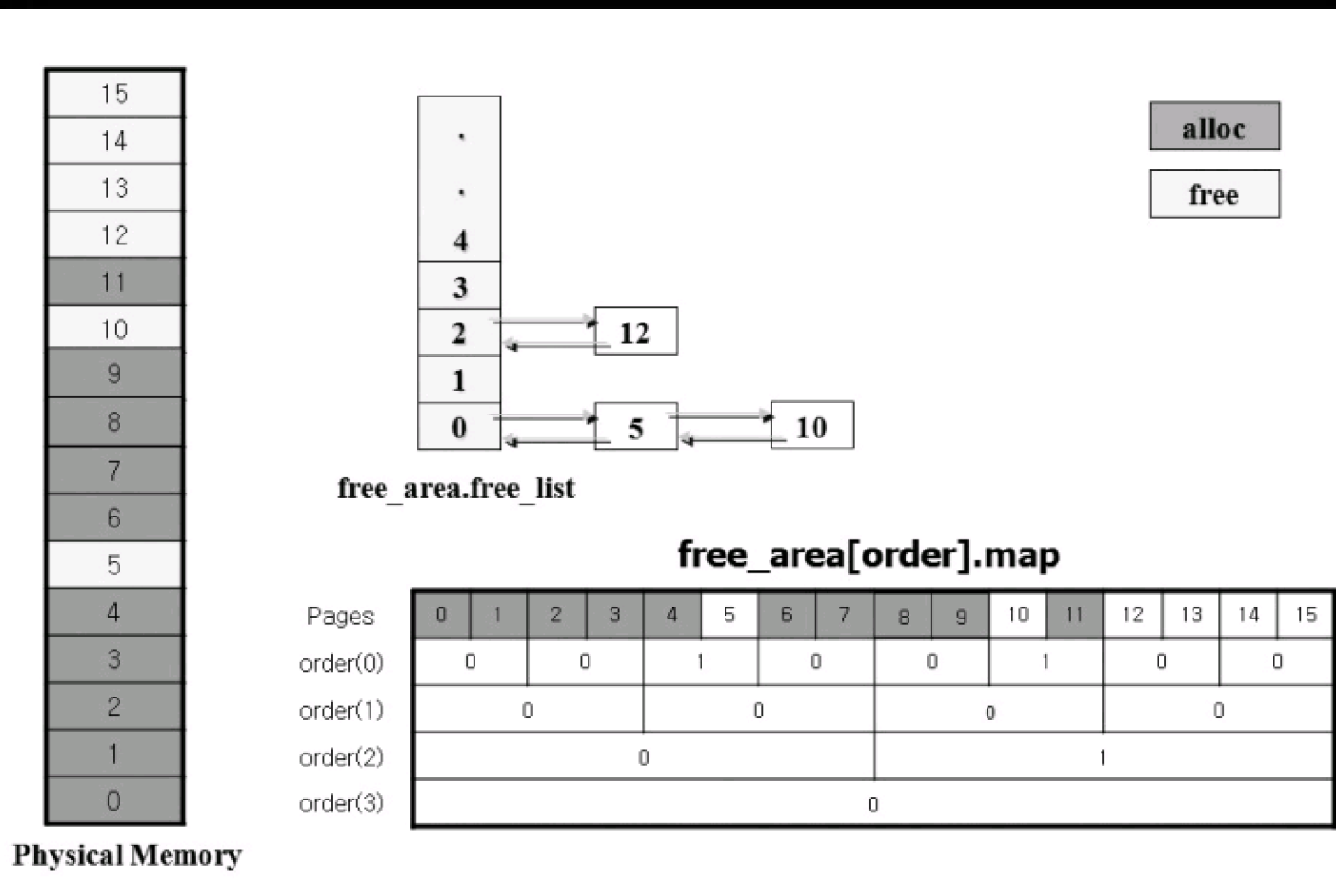
```
struct free_area {  
    struct list_head    free_list;  
    unsigned long      *map;  
};
```

- **free_area.free_list**
Free page frame을 list로 관리
- **free_area.map**
page 상태 bitmap

free_area[n] 에는 free 상태인 연속된 2^n 개의 page frame 들이 free_list를 통해 연결되어 있고, 또한 전체 물리 메모리를 2^n 개의 page frame 단위로 봤을 때의 상태를 map 이라는 bitmap 변수에 저장하고 있음

메모리 관리 기법과 가상 메모리

- Buddy와 Slab \rightarrow Buddy



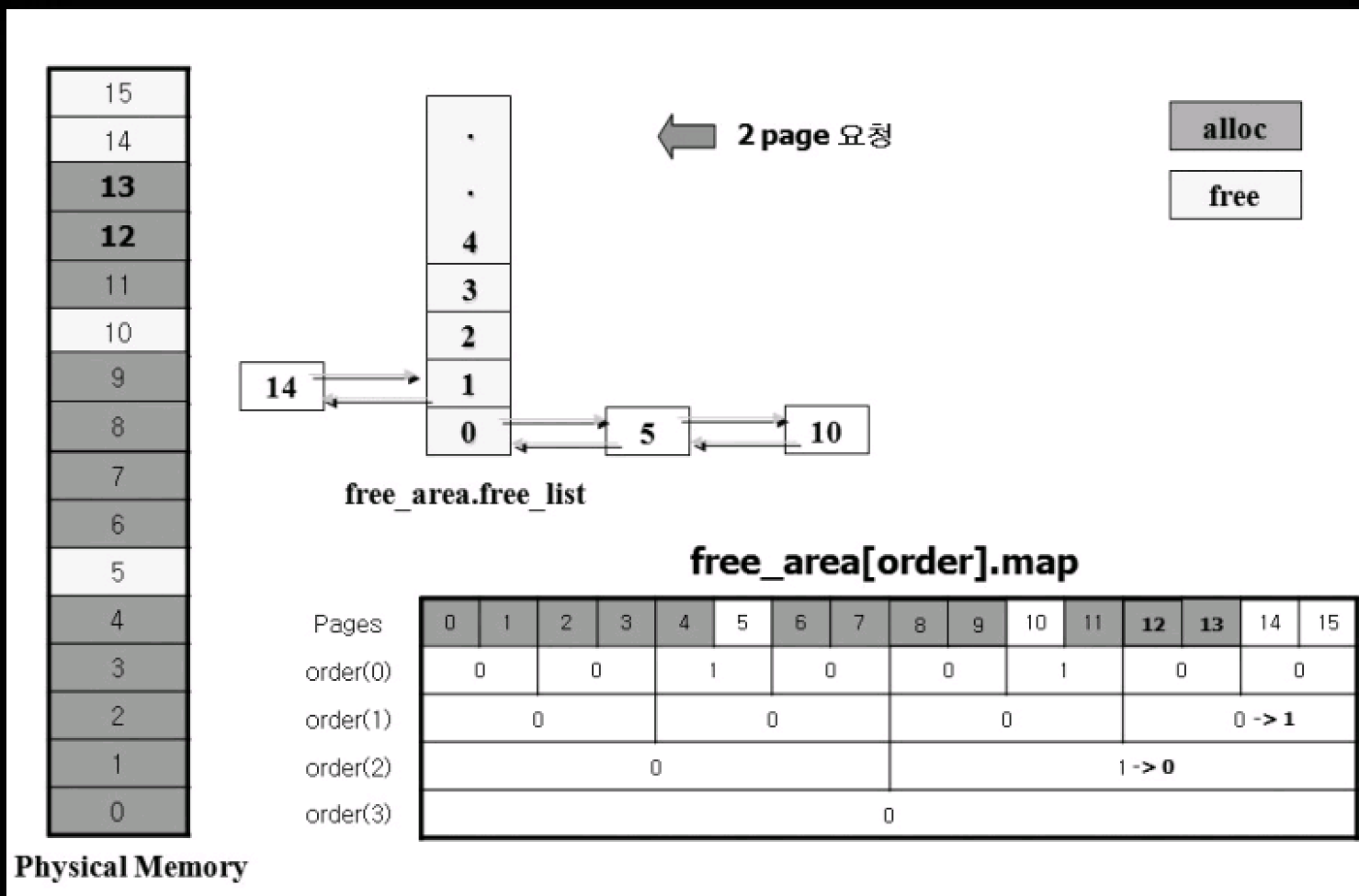
- free_area[n]이 order(n)에 대응 하는 Buddy allocator 동작 예제.

- order(θ)은 한 page frame 단위로 메모리를 관리

- free_area 기준으로 보면
free_area[0].free_list 에는
5번째와 10번째 page frame이,
free_area[2].free_list 에는
12번째 page fram이 list로 연결됨

메모리 관리 기법과 가상 메모리

- Buddy와 Slab → Buddy



- 2 페이지를 할당 요청 받아
free_area[2].free_list에 있는
12번째 page frame이 빠져나간
상태

- 각 order(n)의 bitmap은
 2^n 개의 page frame 기준으로
할당 가능 여부를 통해 설정됨

- 역으로 해제 요청을 받아도
이와 똑같이 작동

메모리 관리 기법과 가상 메모리

- Buddy와 Slab → Buddy

```
struct free_area {  
    struct list_head    free_list;  
    unsigned long       *map;  
};
```



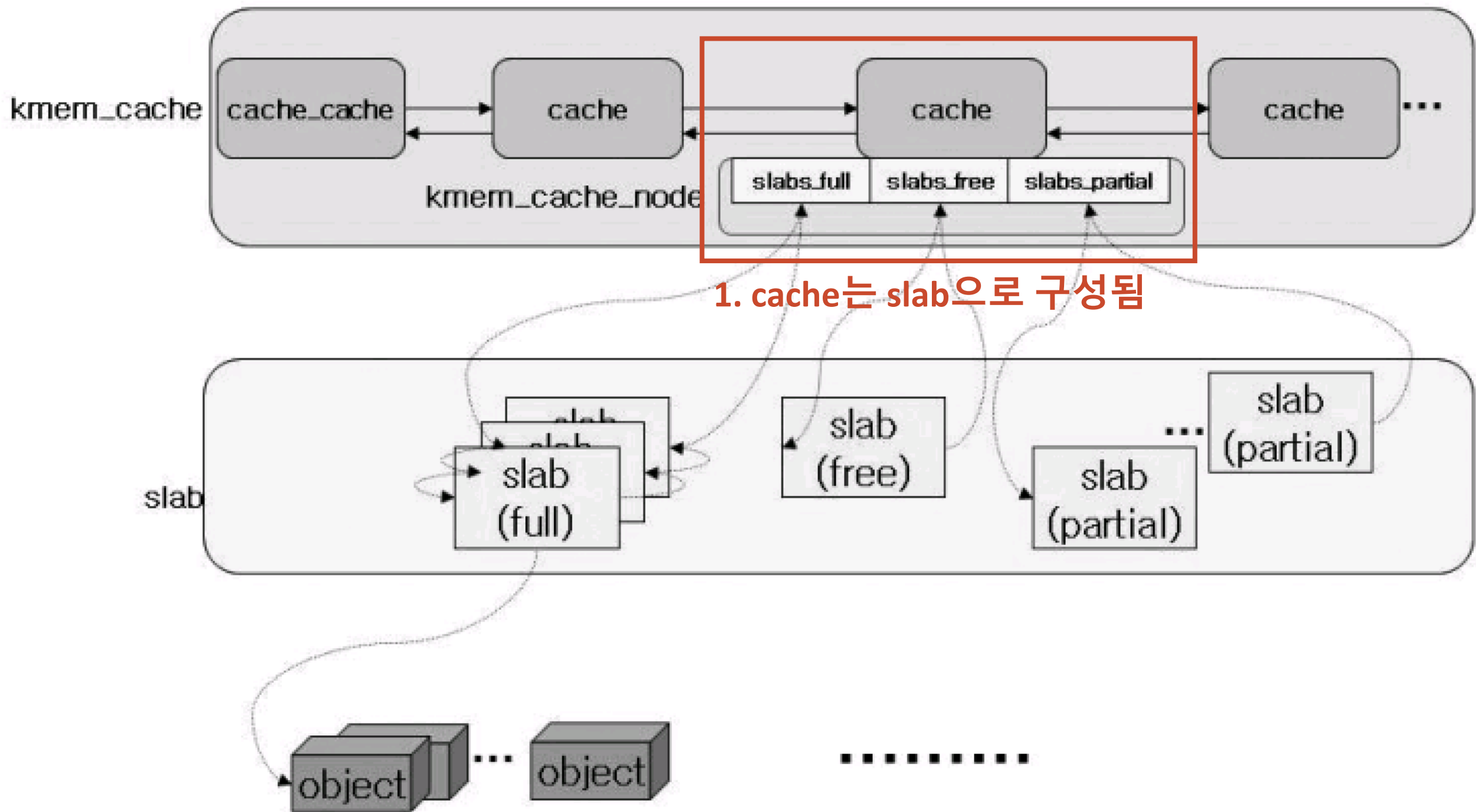
```
struct free_area {  
    struct list_head    free_list[MIGRATE_TYPES];  
    unsigned long       nr_free;  
};
```

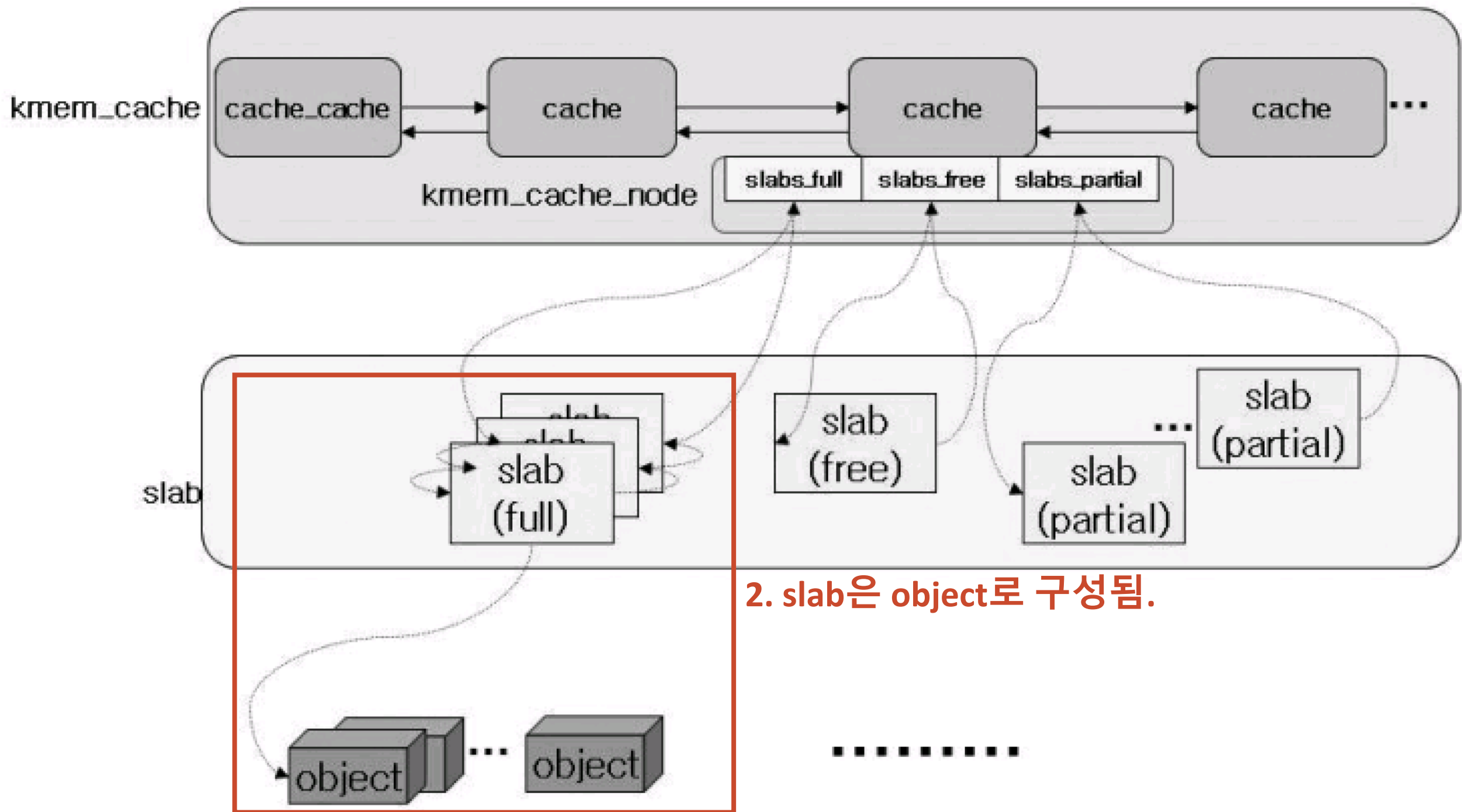
- 한 page frame 을 할당/해제를 반복 하게 될 때 발생하는 **오버헤드를 줄이기 위해서**
커널 버전 2.6.19부터 변경된 Buddy allocator 구조
- free_area 구조체의 비트맵 포인터가 nr_free 변수로 변경
- nr_free 변수는 자신이 관리하는 zone 내에서 비사용 중인 page frame의 개수

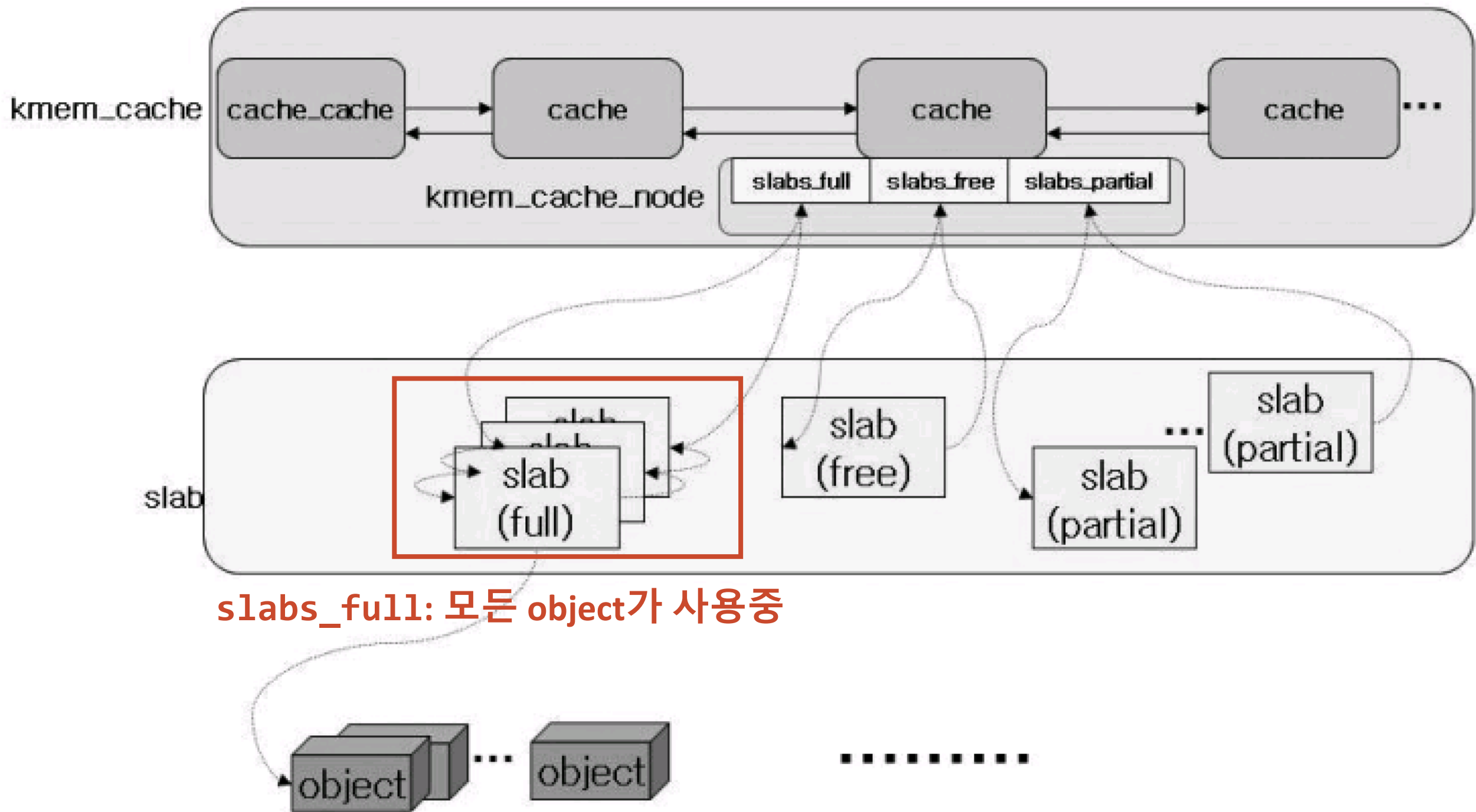
메모리 관리 기법과 가상 메모리

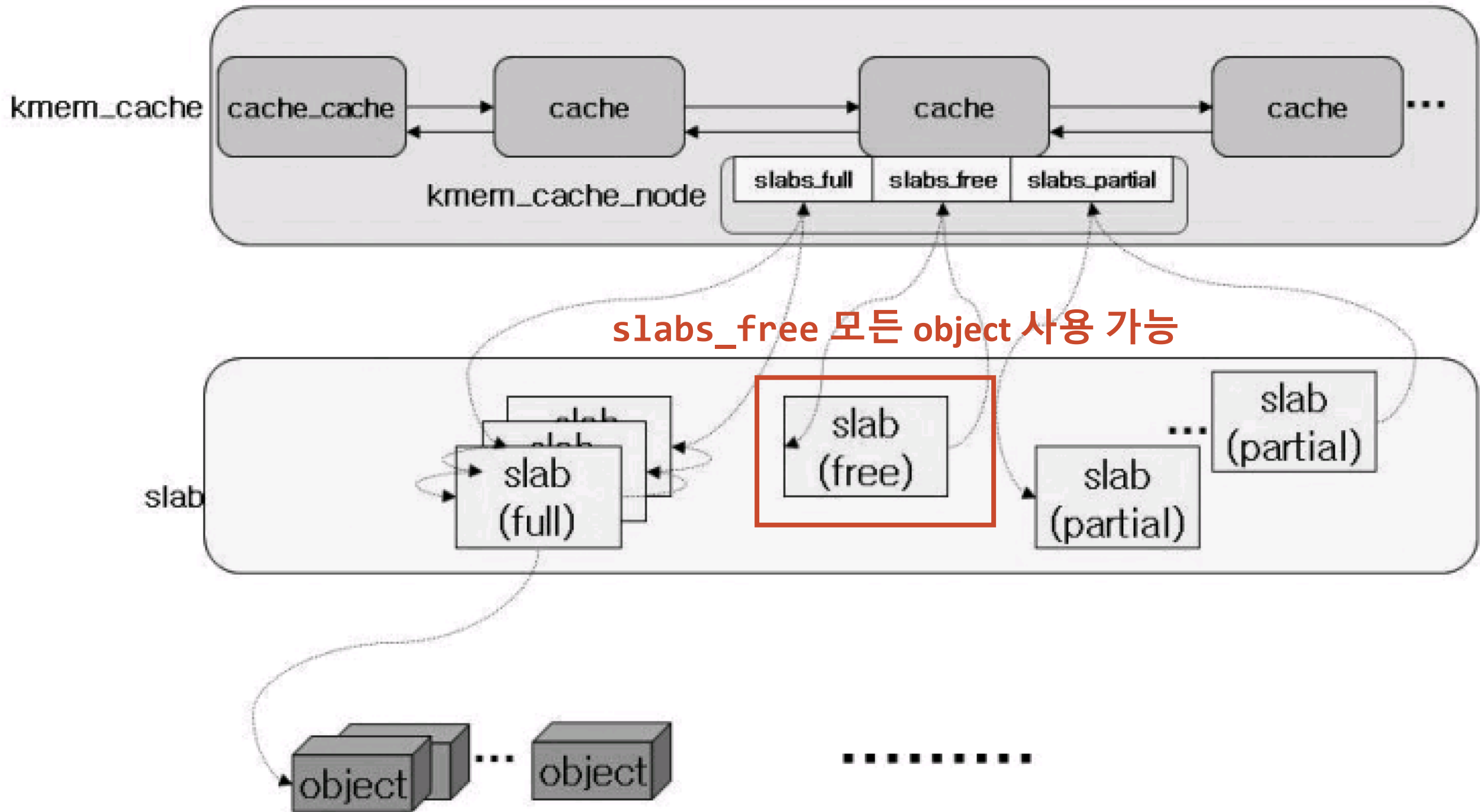
- Buddy와 Slab → Slab

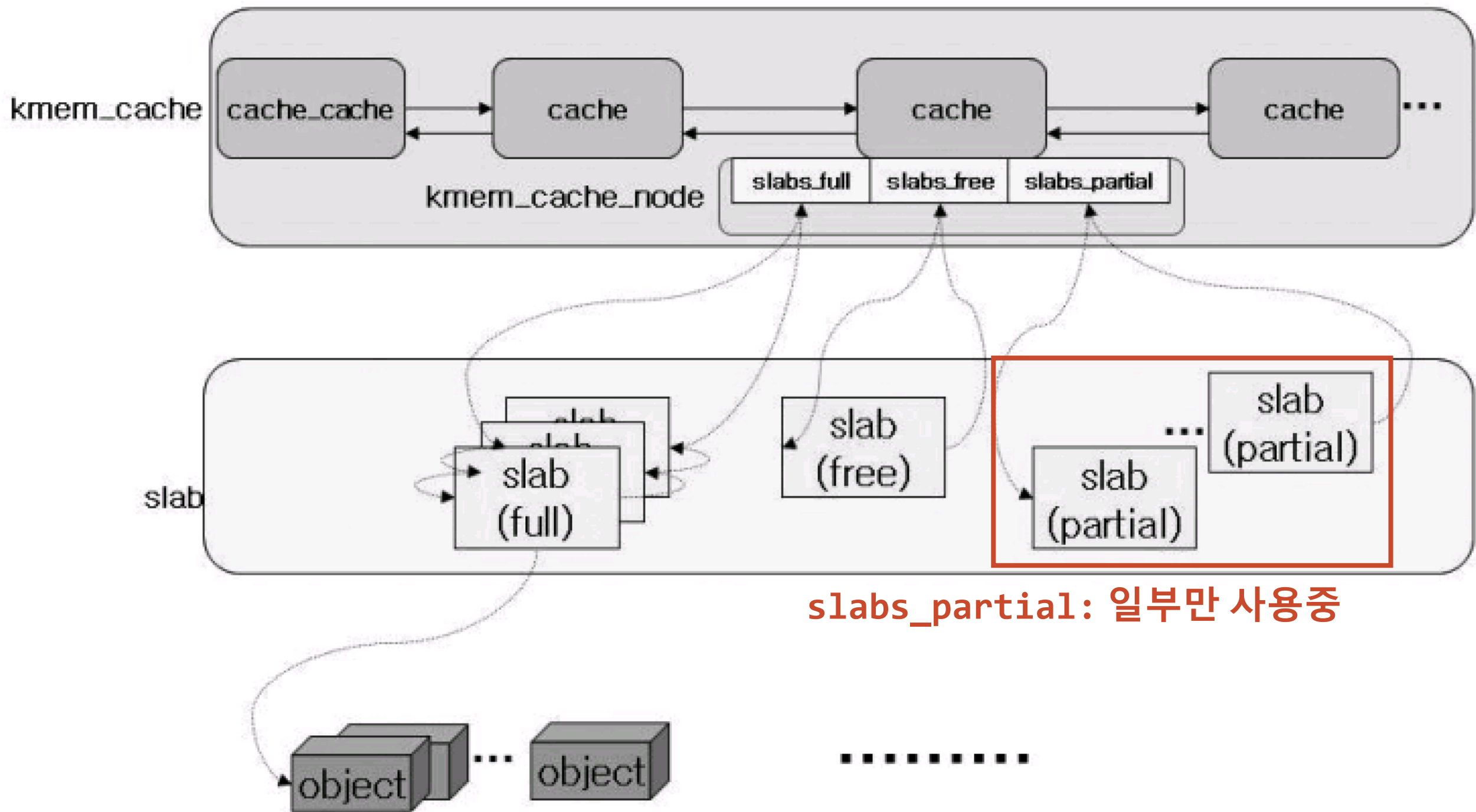
- 리눅스에서는 메모리 할당의 최소 크기는 Page frame 크기인 4KB이다. 만약 사용자가 4KB 보다 작은 크기를 할당 요청할 경우, 4KB를 할당하면 내부 단편화가 발생하기 때문에 Slab allocator를 사용.
- 내부 단편화 최소화를 위해 자주 할당되는 사이즈인 2의 승수 크기의 cache를 128KB 크기까지 유지함.

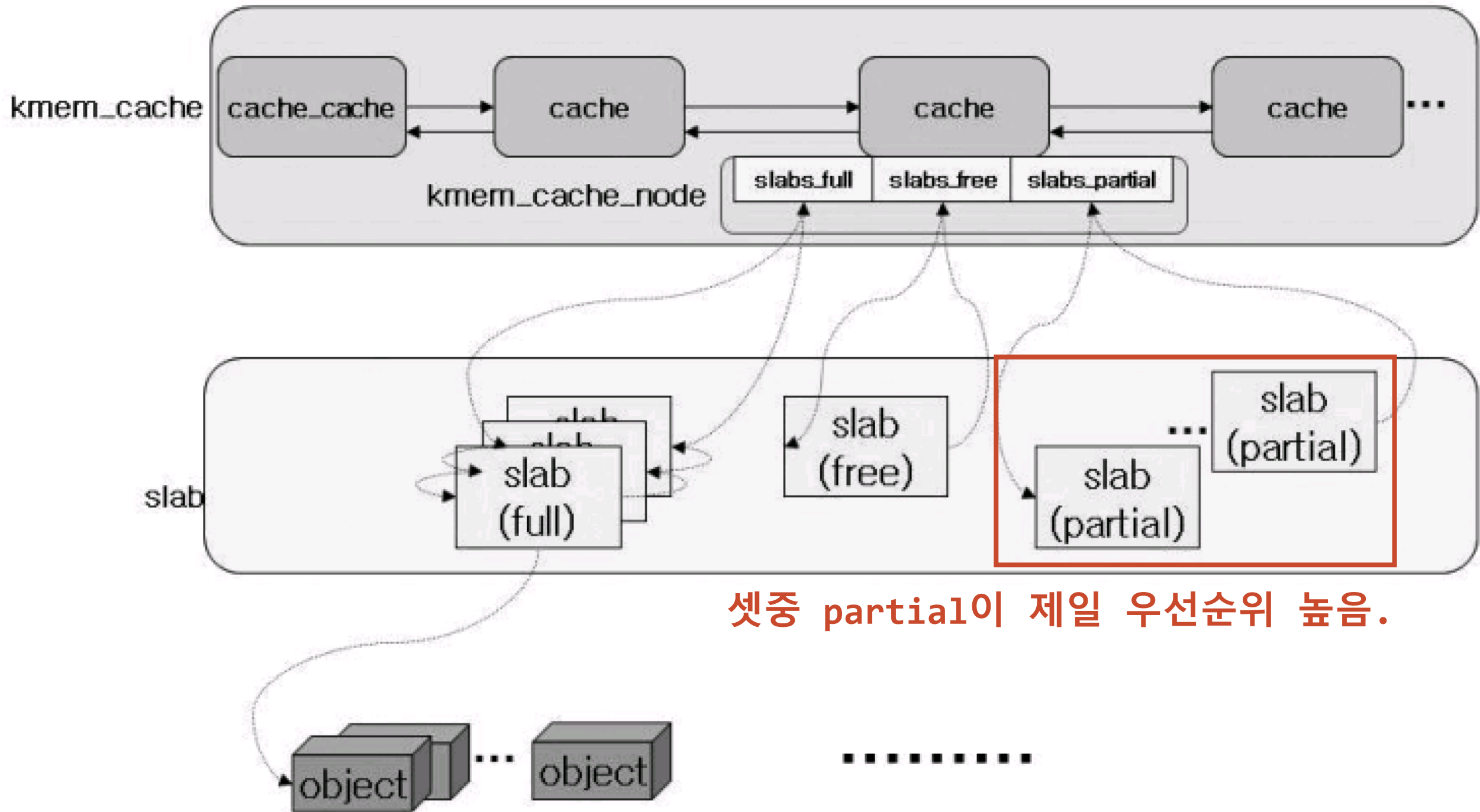













```
include/linux/slab_def.h

struct kmem_cache {
    struct array_cache __percpu *cpu_cache;

    unsigned int batchcount;
    unsigned int limit;
    unsigned int shared;

    unsigned int size;
    struct reciprocal_value reciprocal_buffer_size;

    slab_flags_t flags;
    unsigned int num;
    ...
    struct kmem_cache_node *node[MAX_NUMNODES];
};
```

```
include/linux/slab_def.h

struct kmem_cache {
    struct array_cache __percpu *cpu_cache;

    unsigned int batchcount;
    unsigned int limit;
    unsigned int shared;

    unsigned int size;
    struct reciprocal_value reciprocal_buffer_size;

    slab_flags_t flags;
    unsigned int num;
    ...
    struct kmem_cache_node *node[MAX_NUMNODES];
};
```

mm/slab.h

```
struct kmem_cache_node {  
    spinlock_t list_lock;
```

```
#ifdef CONFIG_SLAB
```

```
    struct list_head slabs_partial;
```

```
    struct list_head slabs_full;
```

```
    struct list_head slabs_free;
```

```
    unsigned long total_slabs;
```

```
    unsigned long free_slabs;
```

```
    unsigned long free_objects;
```

```
    unsigned int free_limit;
```

```
    unsigned int colour_next;
```

```
    struct array_cache *shared;
```

```
    struct alien_cache **alien;
```

```
    unsigned long next_reap;
```

```
    int free_touched;
```

```
#endif
```

```
...
```

```
# cat /proc/slabinfo
```

#name	<active_objs>	<num_objs>	<objsize>	<objperslab>
...				
kmalloc-512	16000	16000	512	64
kmalloc-256	3840	3840	256	64
kmalloc-192	1218	1218	192	42
kmalloc-128	1088	1088	128	64
kmalloc-96	3402	3402	96	42
kmalloc-64	8768	8768	64	64
kmalloc-32	9472	9472	32	128
kmalloc-16	12800	12800	16	256
kmalloc-8	13312	13312	8	512
...				