

## 6장 메모리 관리

### 페이지 프레임 관리

- typedef struct page {  
    struct page \*next;  
    struct page \*prev;  
    struct inode \*inode;  
    unsigned long offset;  
    struct page \*next\_hash;  
    atomic\_t count;  
    unsigned long flags;  
    struct wait\_queue \*wait;  
    struct page \*\*pprev\_hash;  
    struct buffer\_head \* buffers;  
▪ } mem\_map\_t;

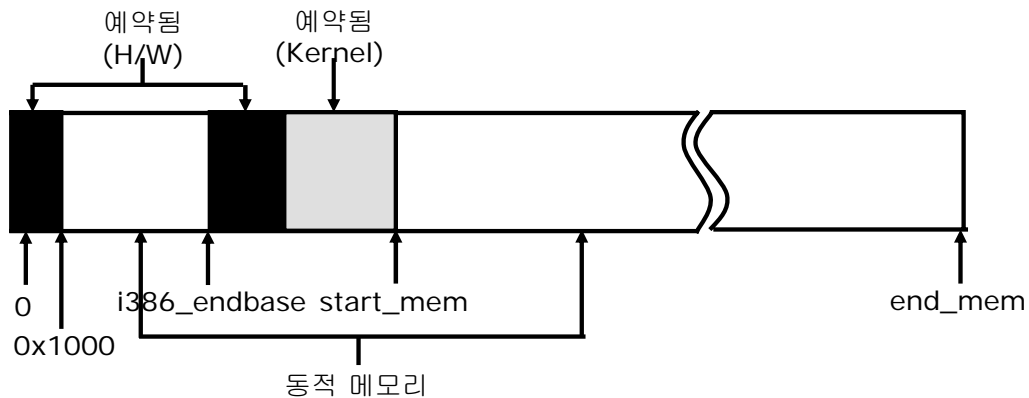
페이지 프레임 정보



descriptor

## ■ mem\_map

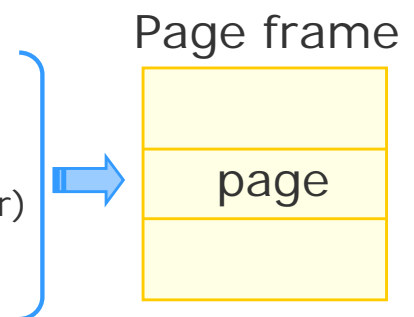
- #define MAB\_NR(addr)  
    (\_\_pa(addr) >> PAGE\_SHIFT)
- free\_area\_init()
- mem\_init()



## 페이지 프레임 요구와 해제

### ■ 페이지 프레임 요구

- \_\_get\_free\_pages(gfp\_mask, order)
- get\_free\_pages(gfp\_mask)
- \_\_get\_dma\_pages(gfp\_mask, order)
- \_\_get\_free\_pages(gfp\_mask | GFP\_DMA, order)
- \_\_get\_free\_page(gfp\_mask)
- \_\_get\_free\_pages(gfp\_mask, 0)



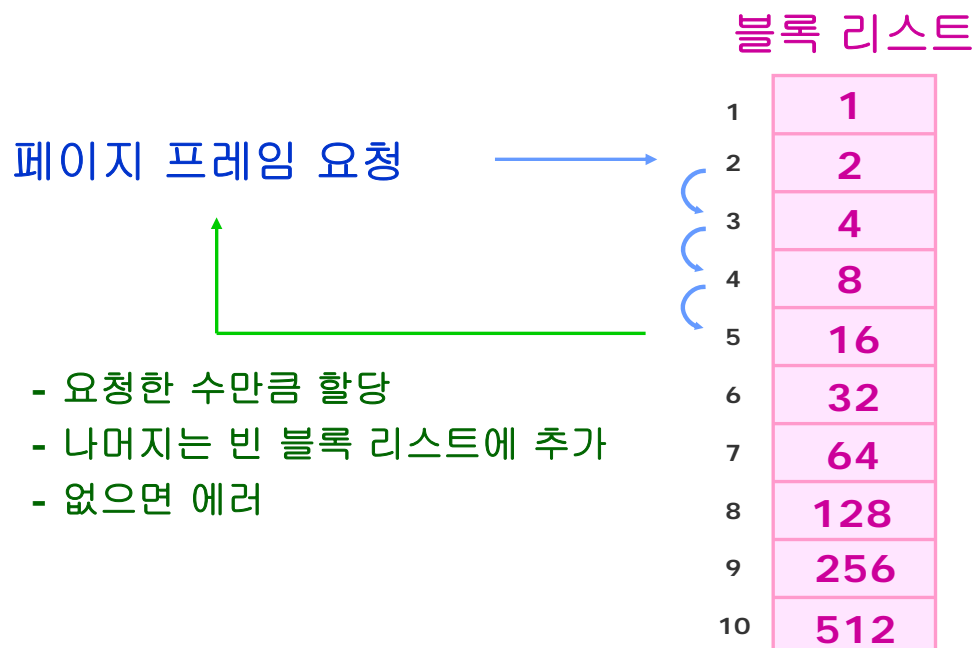
### ■ gfp\_mask

Group Name	__GFP_WAIT	__GFP_IO	Priority
GFP_ATOMIC	0	0	__GFP_HIGH
GFP_BUFFER	1	0	__GFP_LOW
GFP_KERNEL	1	1	__GFP_MED
GFP_NFS	1	1	__GFP_HIGH
GFP_USER	1	1	__GFP_LOW

- 페이지 프레임 해제
  - free\_pages(addr, order)  
count 필드, free\_page\_ok()
  - \_\_free\_page(p)
  - free\_page(addr)
  - ➔ free\_pages(addr, 0)

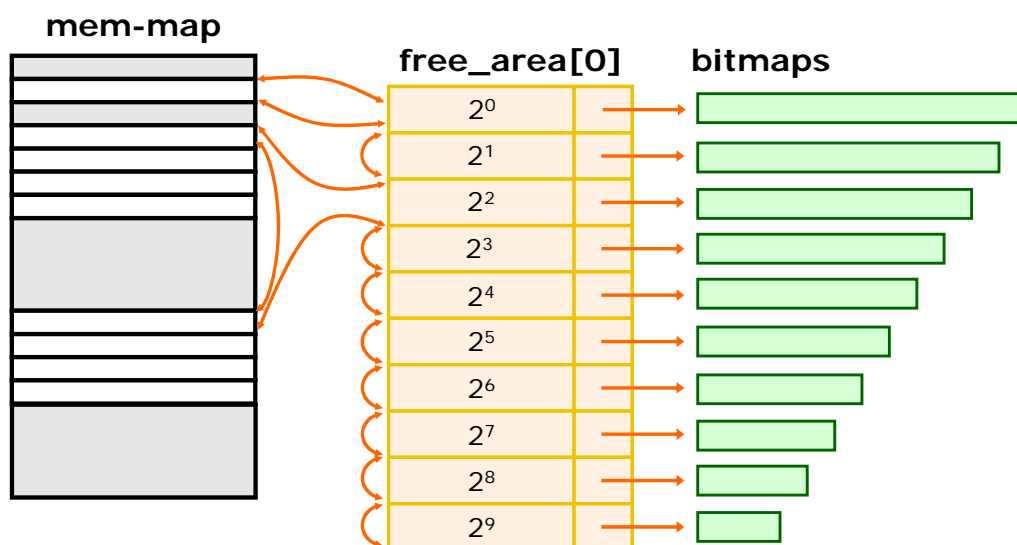
## 버디 시스템 알고리즘

- 페이지 프레임 블록 생성

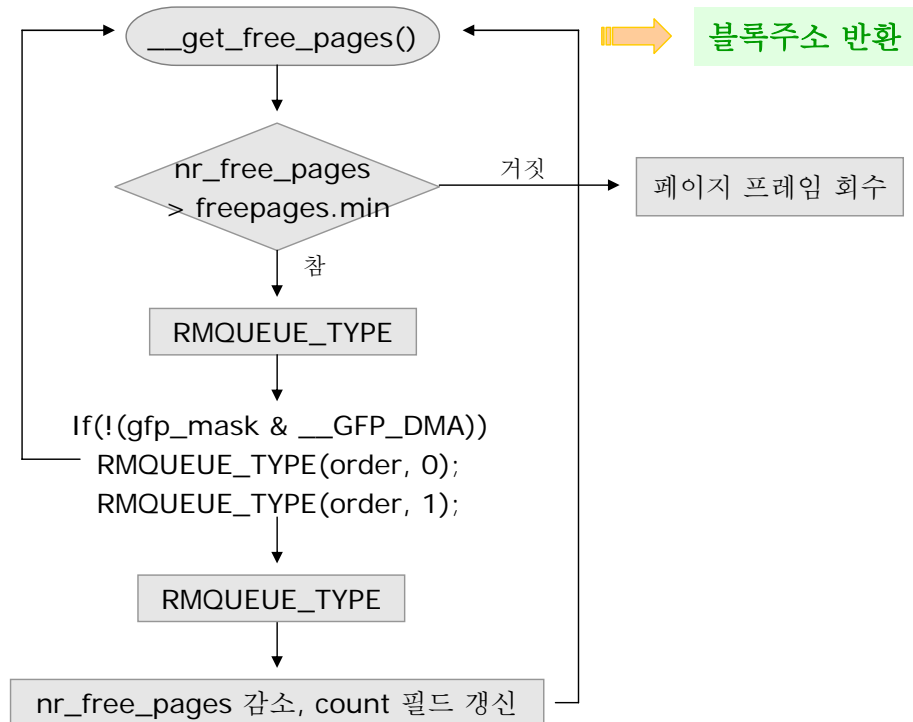


- 외부 단편화 해결 (버디블록의 결합)
- 해제에서 버디 개념이 나옴
- 페이지 프레임 블록 해제 (버디블록)
  - 두 블록이 똑같은 크기  $b$ 를 가지고 있다.
  - 이 두 블록이 연속된 물리적인 주소를 포함
  - 첫째 페이지 프레임 물리주소 :  $2 * b * 2$  의 배수  
(생성의 경우는 크기\*2 의 배수)
  - 해제된 블록 결합 성공, 더 큰 블록 결합 시도

## ■ 버디 시스템 자료구조

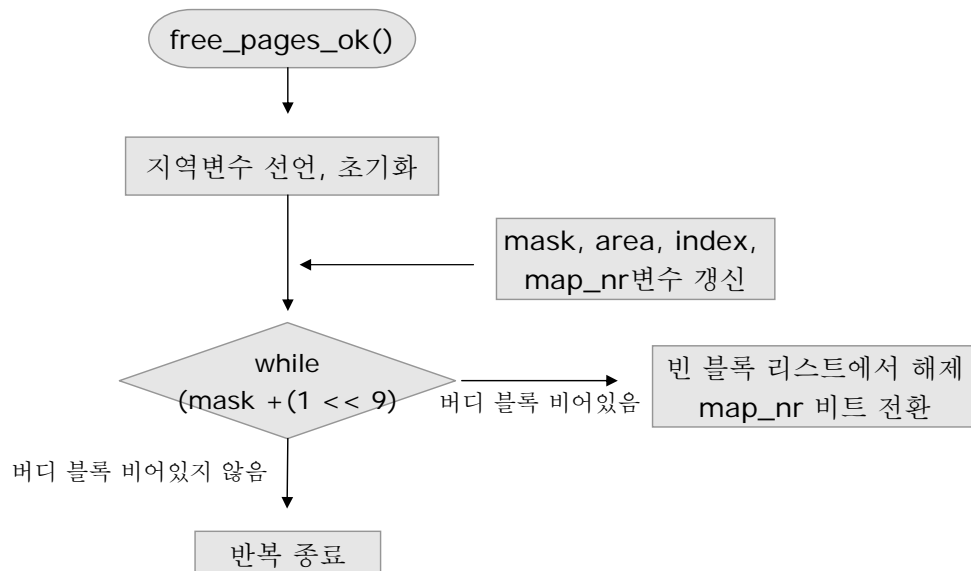


## ■ 블록 할당



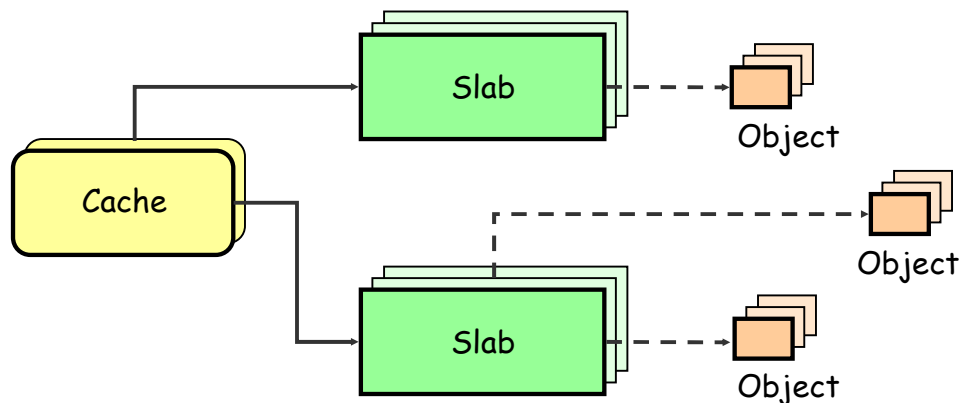
## ■ 블록 해제

- map\_nr, order, type



## 슬랩 할당자

- 목적 : 버디 시스템 할당호출 횟수 감소
- 내부단편화 해결
- 하드웨어 캐시 성능 증가



## 캐시 디스크립터

- struct kmem\_cache\_s 테이블로 표시

```
210.110.145.185 - CRT
File Edit View Options Transfer Script Window Help
struct kmem_cache_s {
    kmem_slab_t      *c_freep;
    unsigned long    c_flags;
    unsigned long    c_offset;
    unsigned long    c_num;

    unsigned long    c_magic;
    unsigned long    c_inuse;
    kmem_slab_t      *c_firstp;
    kmem_slab_t      *c_lastp;

    spinlock_t       c_spinlock;
    unsigned long    c_growing;
    unsigned long    c_dflage;
    size_t           c_orig_size;
    unsigned long    c_gfporder;
    void (*c_ctor)(void *, kmem_cache_t *, unsigned long);
    void (*c_dtor)(void *, kmem_cache_t *, unsigned long);
    unsigned long    c_align;
    size_t           c_colour;
    size_t           c_colour_next;
    unsigned long    c_failures;
    const char       *c_name;
    struct kmem_cache_s *c_nextp;
    kmem_cache_t      *c_index_cachep;

    /* SLAB_STATS */
    unsigned long    c_num_active;
    unsigned long    c_num_allocations;
    unsigned long    c_high_mark;
    unsigned long    c_grown;
    unsigned long    c_reaped;
    atomic_t         c_errors;
}

#endif /* SLAB_STATS */
38
```

```
210.110.145.185 - CRT
File Edit View Options Transfer Script Window Help
struct kmem_cache_s {
    struct list_head slabs;
    struct list_head *firstnotfull;
    unsigned int     objsize;
    unsigned int     flags;
    unsigned int     num;
    spinlock_t       spinlock;
    unsigned int     batchcount;
    unsigned int     sfporder;
    unsigned int     sfpflags;
    size_t           colour;
    unsigned int     colour_off;
    unsigned int     colour_next;
    kmem_cache_t      *slabp_cache;
    unsigned int     growing;
    unsigned int     dflags;

    void (*ctor)(void *, kmem_cache_t *, unsigned long);
    void (*dtor)(void *, kmem_cache_t *, unsigned long);

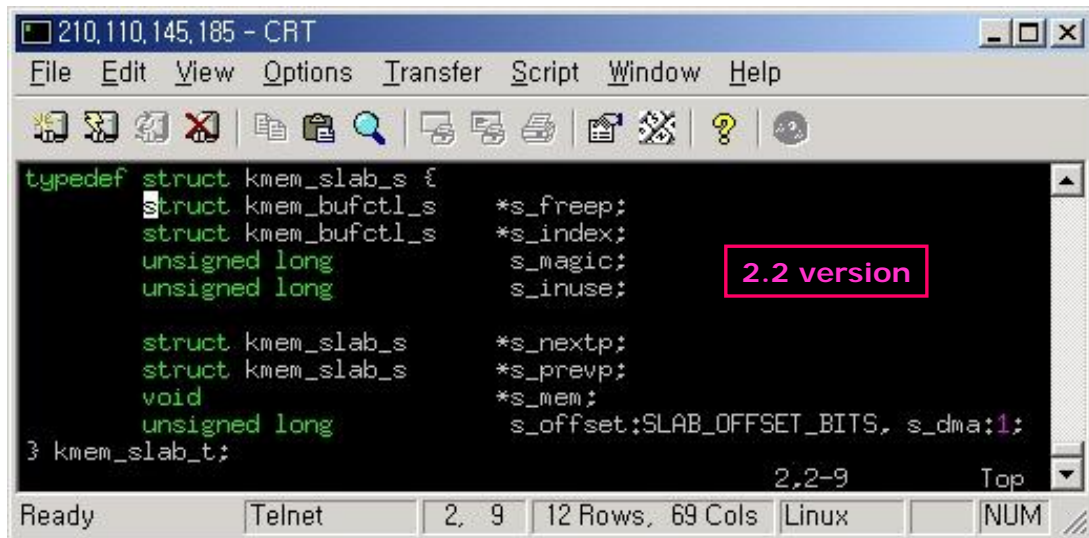
    unsigned long    failures;
    char             name[CACHE_NAMELEN];
    struct list_head next;
    #ifdef CONFIG_SMP
    cpucache_t       *cpudata[NR_CPUS];
    #endif
    /* IF STATS */
    unsigned long    num_active;
    unsigned long    num_allocations;
    unsigned long    high_mark;
    unsigned long    grown;
    unsigned long    reaped;
    unsigned long    errors;
    #ifdef CONFIG_SMP
    atomic_t         allochit;
    atomic_t         allocmiss;
    atomic_t         freehit;
    atomic_t         freemiss;
    #endif
}

#endif
39
```

<mm/slab.c>

## 슬랩 디스크립터

- struct kmem\_slab\_s 형 descriptor가짐  
- 2.4 version에는 없음.



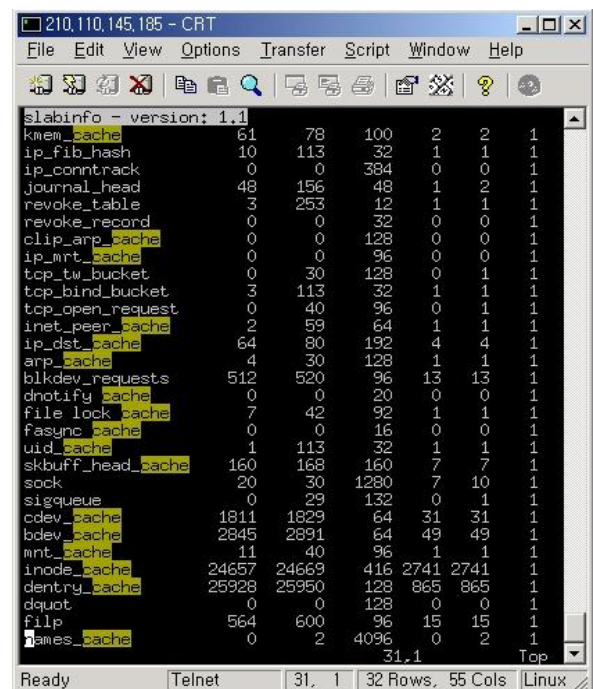
```
210,110,145,185 - CRT
File Edit View Options Transfer Script Window Help
typedef struct kmem_slab_s {
    struct kmem_bufctl_s    *s_freep;
    struct kmem_bufctl_s    *s_index;
    unsigned long           s_magic;
    unsigned long           s_inuse;

    struct kmem_slab_s      *s_nextp;
    struct kmem_slab_s      *s_prevp;
    void                    *s_mem;
    unsigned long           s_offset:SLAB_OFFSET_BITS, s_dma:1;
} kmem_slab_t;
2,2-9 Top
Ready Telnet 2, 9 12 Rows, 69 Cols Linux NUM
```

<mm/slab.c>

## 일반 캐시 특수 캐시

- 일반캐시  
- 슬랩 할당자에서 필요  
- kmem\_cache\_init(),  
kmem\_cache\_sizes\_init())
- 특수캐시  
- 커널의 나머지 부분에서 필요  
- kmem\_cache\_create()



slabinfo - version: 1.1						
kmem_cache	61	78	100	2	2	1
ip_fib_hash	10	113	32	1	1	1
ip_conntrack	0	0	384	0	0	1
journal_head	48	156	48	1	2	1
revoke_table	3	253	12	1	1	1
revoke_record	0	0	32	0	0	1
clip_arp_cache	0	0	128	0	0	1
ip_mrt_cache	0	0	96	0	0	1
tcp_tw_bucket	0	30	128	0	1	1
tcp_bind_bucket	3	113	32	1	1	1
tcp_open_request	0	40	96	0	1	1
inet_peer_cache	2	59	64	1	1	1
ip_dst_cache	64	80	192	4	4	1
arp_cache	4	30	128	1	1	1
blkdev_requests	512	520	96	13	13	1
dnotify_cache	0	0	20	0	0	1
file_lock_cache	7	42	92	1	1	1
fasync_cache	0	0	16	0	0	1
uid_cache	1	113	32	1	1	1
skbuff_head_cache	160	168	160	7	7	1
sock	20	30	1280	7	10	1
sigqueue	0	29	132	0	1	1
cdev_cache	1811	1829	64	31	31	1
bdev_cache	2845	2891	64	49	49	1
mnt_cache	11	40	96	1	1	1
inode_cache	24657	24669	416	2741	2741	1
dentry_cache	25928	25950	128	865	865	1
dquot	0	0	128	0	0	1
filp	564	600	96	15	15	1
ames_cache	0	2	4096	0	2	1
				31,1		Top

일반, 특수캐시 이름 확인 → <proc/slabinfo>

## 버디 시스템의 슬랩 할당자

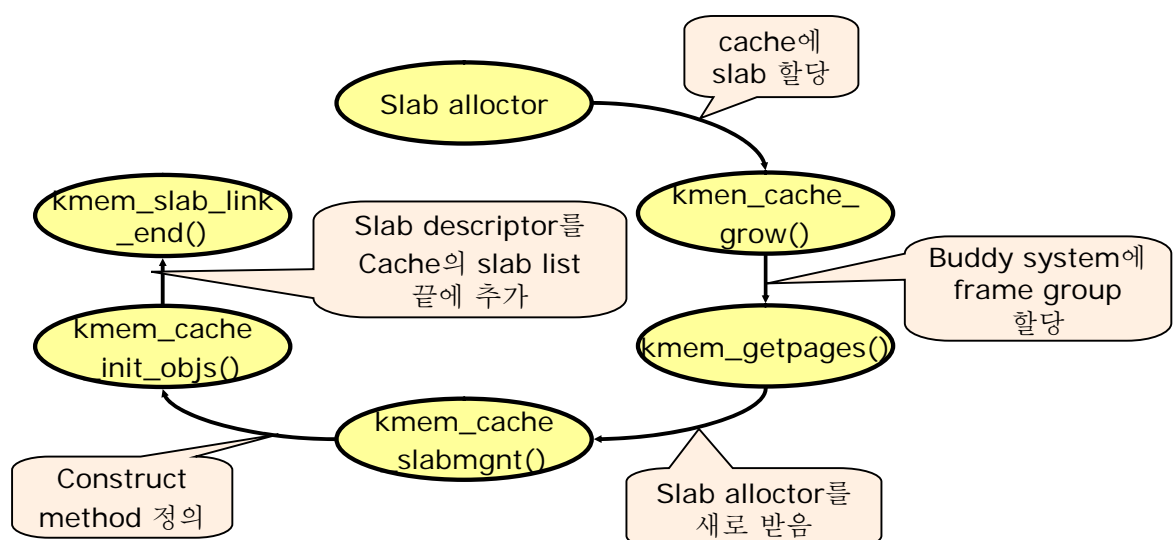
- 연속된 빈 페이지 프레임 할당
  - kmem\_getpages()
- 할당한 페이지 프레임 해제
  - kmem\_freepages()

```
free_pages((unsigned long)addr, cache->c_gfporder);
```

페이지프레임의 물리주소

## 캐시에 슬랩 할당

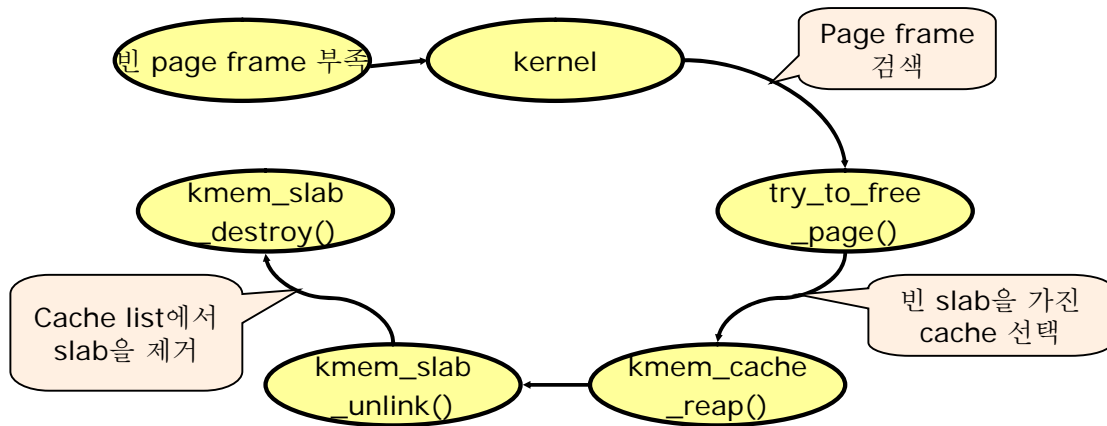
- cache에 slab을 할당하는 경우
  - 새 object를 할당해 달라는 요청이 들어왔을 때
  - cache에 빈 object가 없을 때





## 캐시에서 슬랩 제거

- Slab을 해제하는 경우
  - 버디 시스템이 새로 페이지 프레임group을 할당해 달라는 요구를 처리 할 수 없을 때
  - slab과, slab안의 모든 object가 비어있을 때



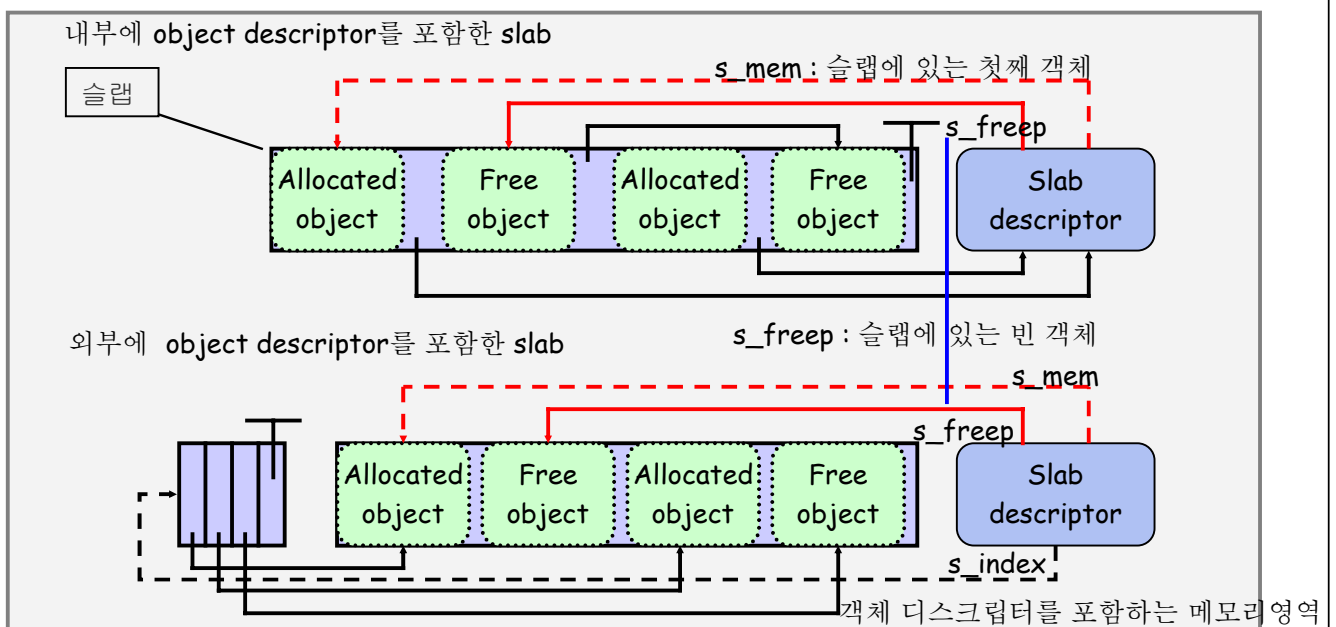
## 메모리 영역 관리



## ■ 개 요

- 자신이 사용할 동적 메모리를 커널이 어떻게 할당하는지 설명한다.
- 물리적으로 연속된 메모리 영역을 다루는 기법
  1. 객체 디스크립터
- 불연속적인 메모리 영역을 다루는 기법

## 슬랩 디스크립터와 객체 디스크립터 사이의 관계



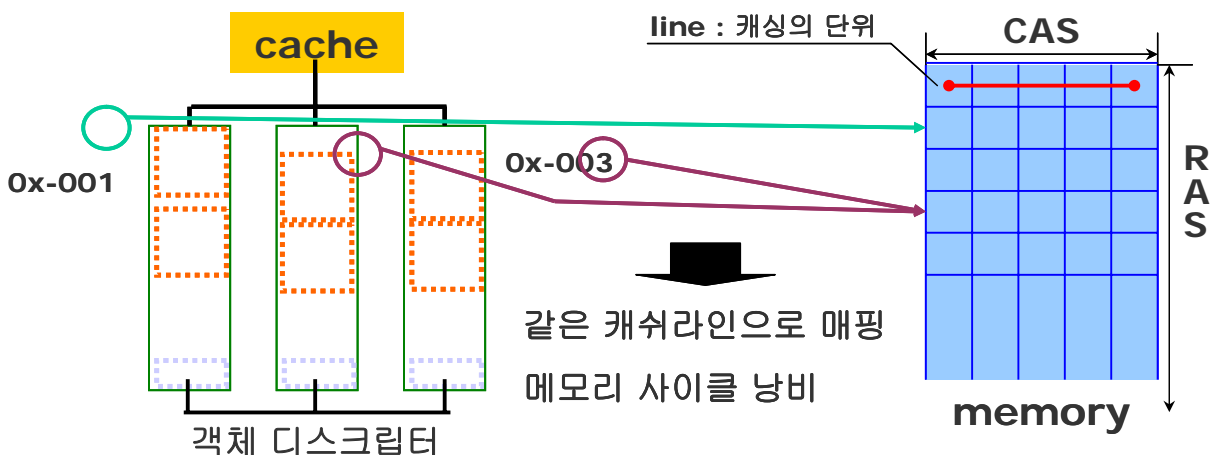
- 슬랩의 객체 디스크립터를 슬랩 외부와 슬랩 내부에 저장할 수 있다.
- 슬랩 할당자가 객체의 크기가 512보다 크거나 512의 배수라면 외부에 저장하고, 512보다 작거나 배수가 아니라면 내부에 저장

## 메모리에서의 객체 정렬

- 슬랩 할당자에서 사용할 수 있는 최대 정렬 계수는 4096, 즉 페이지 프레임의 크기다.
- 이는 객체를 객체의 물리 주소나 선형 주소를 참조하여 정렬할 수 있다는 것을 말한다.
- **BYTE\_PER\_WORD** 매크로가 지정한 워드 크기에 따라 정렬
  - 물리 주소가 내부 메모리의 버스의 폭에 정렬될 때 빨리 접근.
- 캐시에 들어가는 객체가 **CPU의 1차 캐시**에서 정렬
  - 객체를 **L1\_CACHE\_BYTES**의 배수가 되도록, 즉 캐시 라인 시작에 위치하도록 한다.
  - 객체 크기를 **L1\_CACHE\_BYTES** 계수에 따라 올림한다. 이는 한 객체가 절대 두 캐시 라인에 걸쳐 있지 않는다.

## 슬랩 컬러링

- 똑같은 크기의 객체는 캐시에서 똑같은 오프셋에 저장되는 경향이 있다.
- 슬랩의 빈 영역 일부를 맨 끝에서 맨 앞으로 옮기는 방법



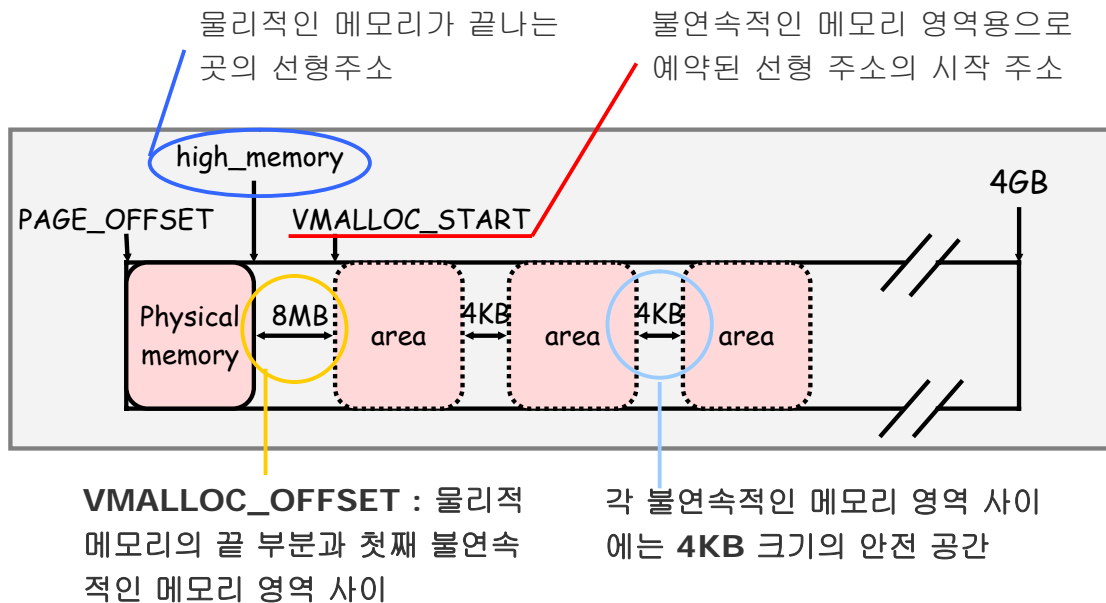
- \* **RAS(Row Address Strobe)** : 번지 지정을 위해 사용하는 신호
- \* **CAS(Column Address Strobe)**

- 객체 할당
  - kmem\_cache\_alloc()
- 객체 해제
  - kmem\_cache\_free()
- 일반 목적 객체
  - 드물게 일어나는 메모리 영역 요구에 사용
  - 객체 크기가 최소 32부터 최대 131072바이트까지 기하학적으로 분포된 일반 캐시 그룹으로 처리
  - kcalloc(size\_t size, int flags) 사용

## 불연속적인 메모리 영역 관리

- 연속적인 메모리 관리
  - 메모리 영역을 연속된 페이지 프레임 집합으로 매핑
  - 캐시를 잘 활용하여 평균메모리 접근 시간을 줄이는 방법
- 장점
  - 메모리 영역에 대한 요청이 드물다면 외부 단편화 문제를 피함
- 단점
  - 커널 페이지 테이블을 다뤄야 한다.
- 리눅스에서는 드물게 불연속적인 메모리 영역을 사용

## 불연속적인 메모리 영역의 선형 주소



- 불연속적인 메모리 영역 디스크립터
  - struct\_vm\_struct 자료형 디스크립터
  - next 필드를 이용하여 간단한 리스트에 들어간다.
  - get\_vm\_area() 함수를 이용
- 불연속적인 메모리 영역 할당
  - vmalloc() 함수는 커널에 불연속적인 메모리 영역 할당
- 불연속적인 메모리 영역 해제
  - vfree() 함수를 이용하여 불연속적인 메모리 영역 해제
  - 영역 자체는 vmfree\_area\_pages(), 디스크립터는 kfree() 이용

## 객체 할당, 해제 예

