Linux Kernel Internal

단국대학교
컴퓨터학과
2009
백승재
baeksj@dankook.ac.kr
http://embedded.dankook.ac.kr/~baeksj

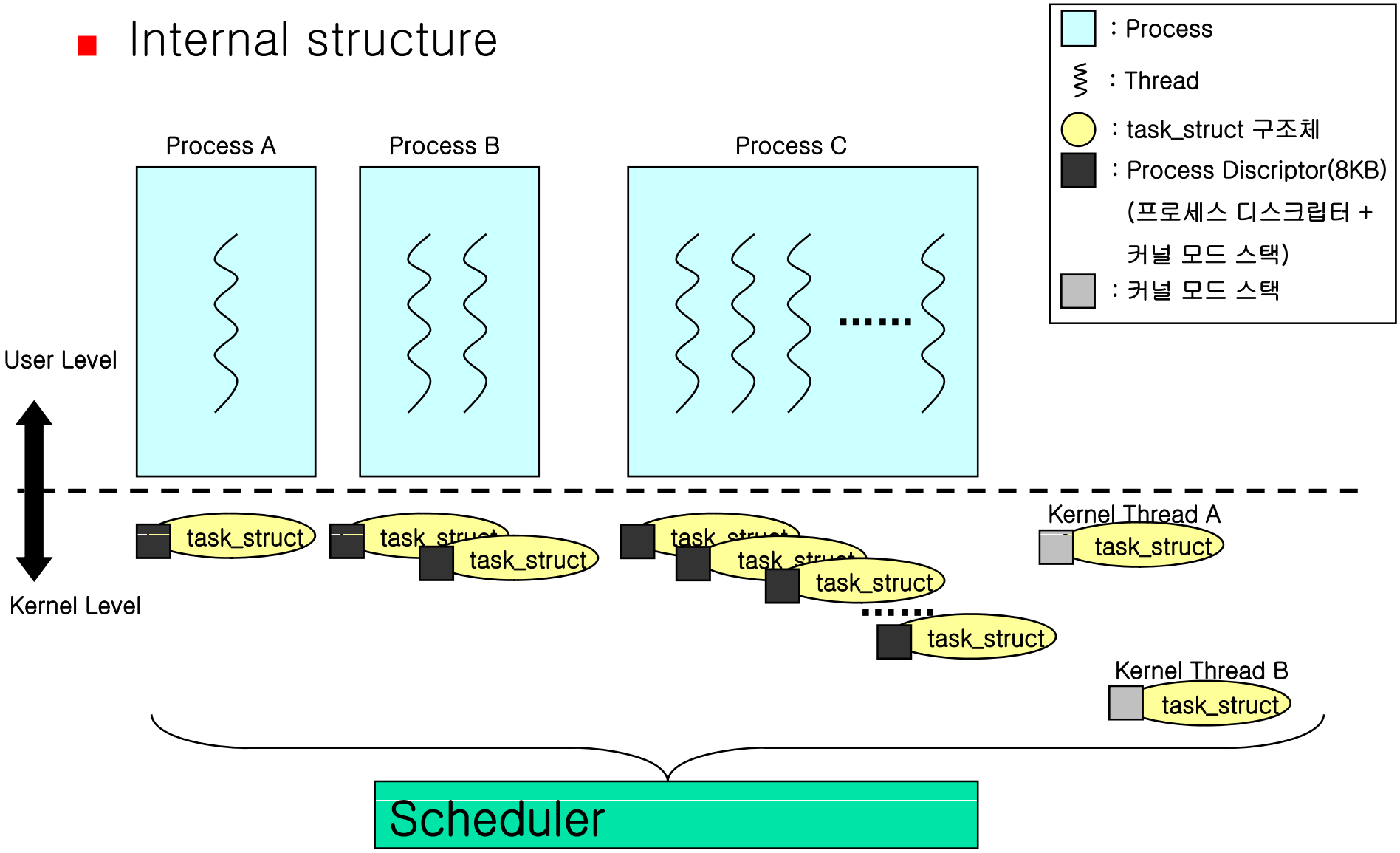
Task, Process and Thread

- Process
  - ✓ 실행 상태에 있는 프로그램의 instance ...
  - ✓ 자원 소유권의 단위
- Thread
  - ✓ 디스패칭의 단위, 실행 흐름 ...
  - ✓ 수행의 단위

- In Linux source code
  - ✓ Process is Task and Thread is also Task

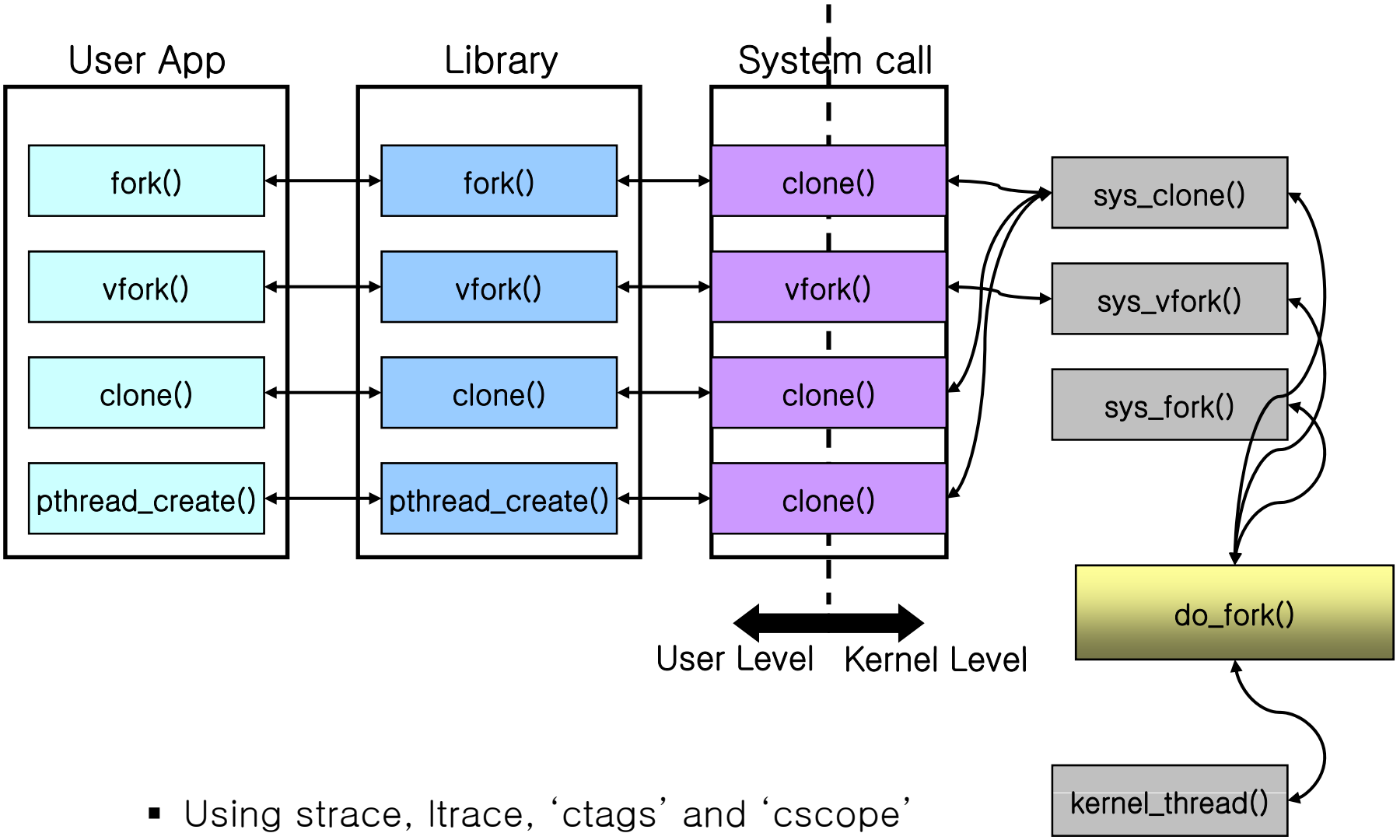
Linux Task Model

- Internal structure



Task 관련 함수 흐름

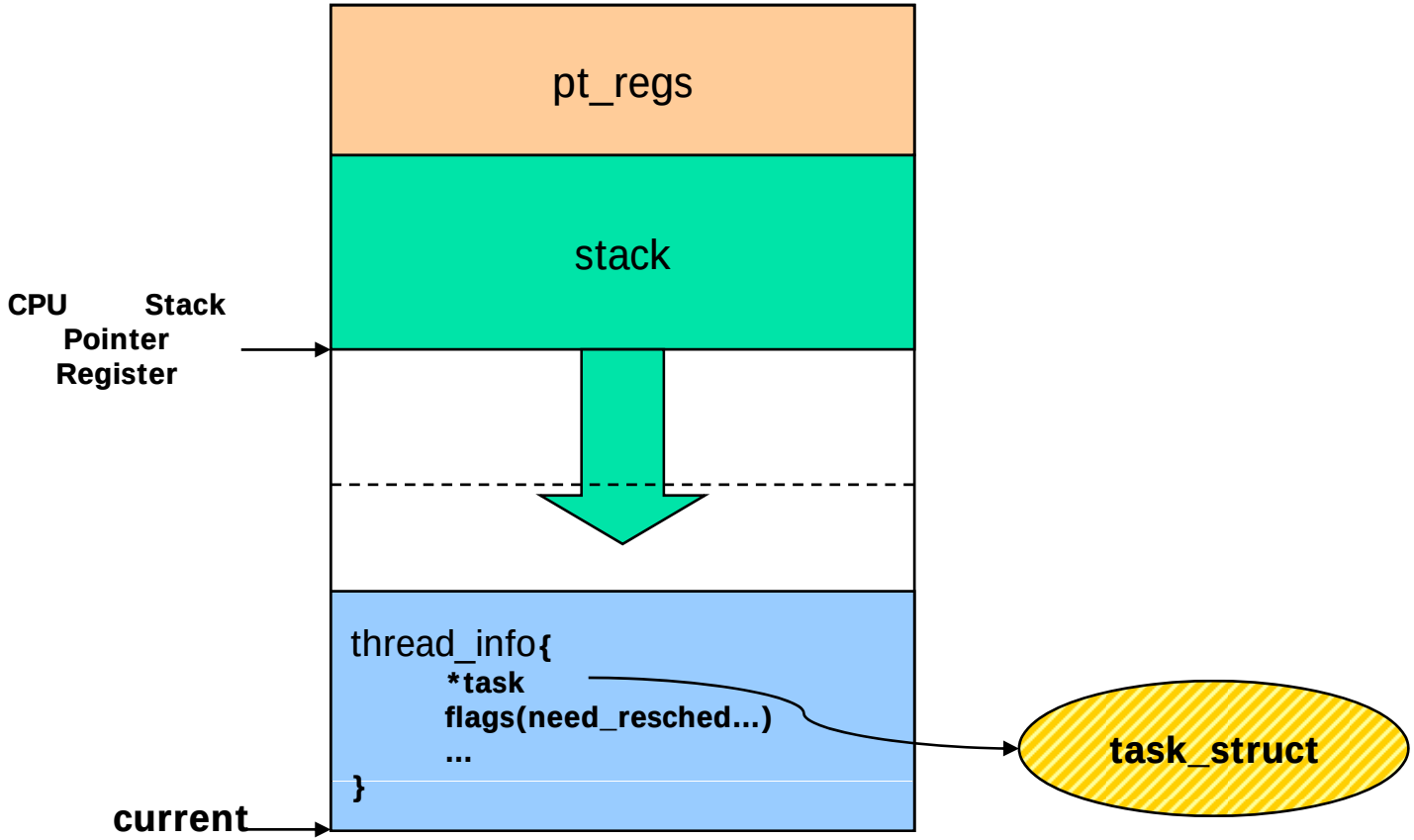
- Flow controls



- Using strace, ltrace, ‘ctags’ and ‘cscope’

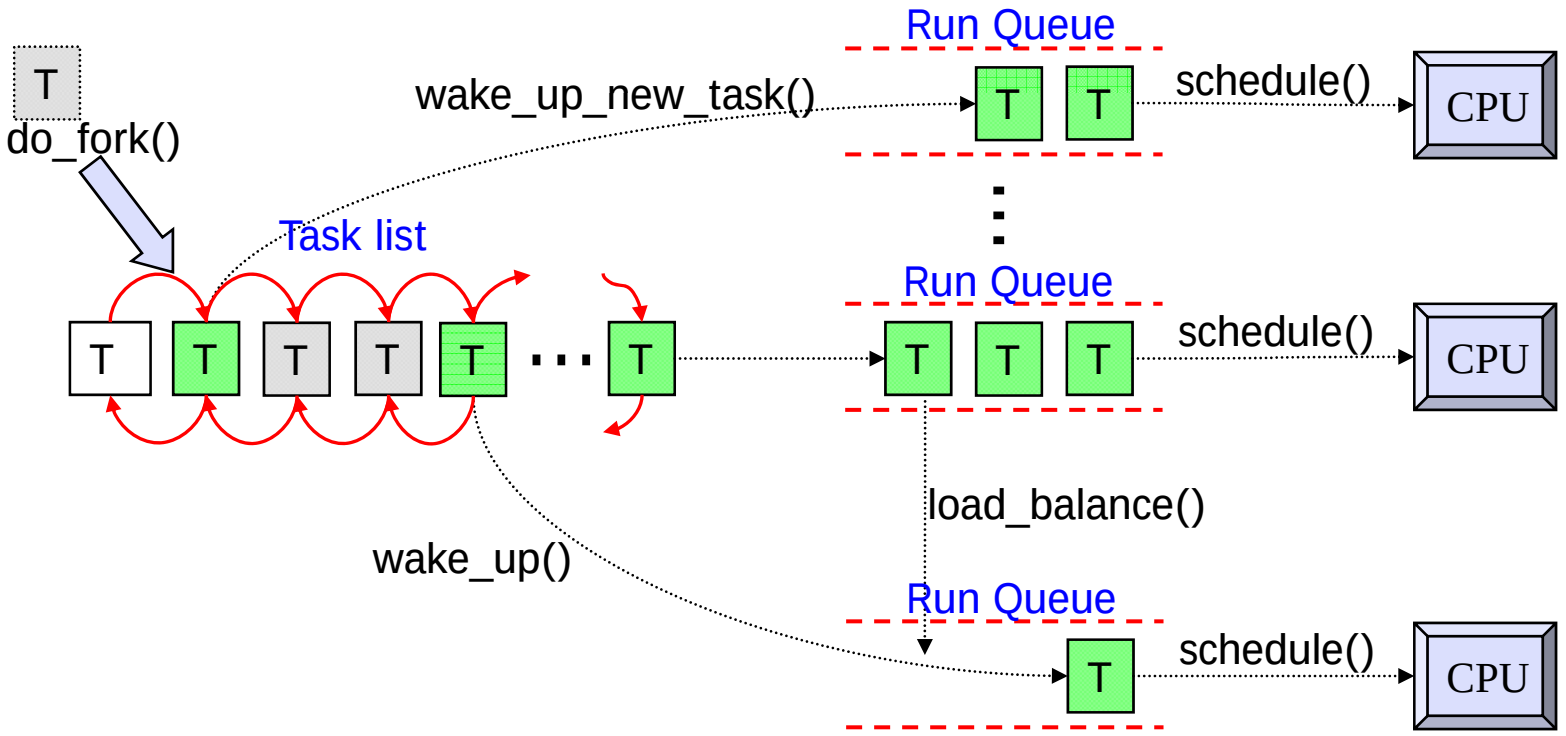
Kernel Stack

- Thread union
  - ✓ 커널은 각 태스크 별로 8KB메모리 할당
  - ✓ thread_info 구조체와 kernel stack
  - ✓ alloc_thread_struct, free_thread_struct

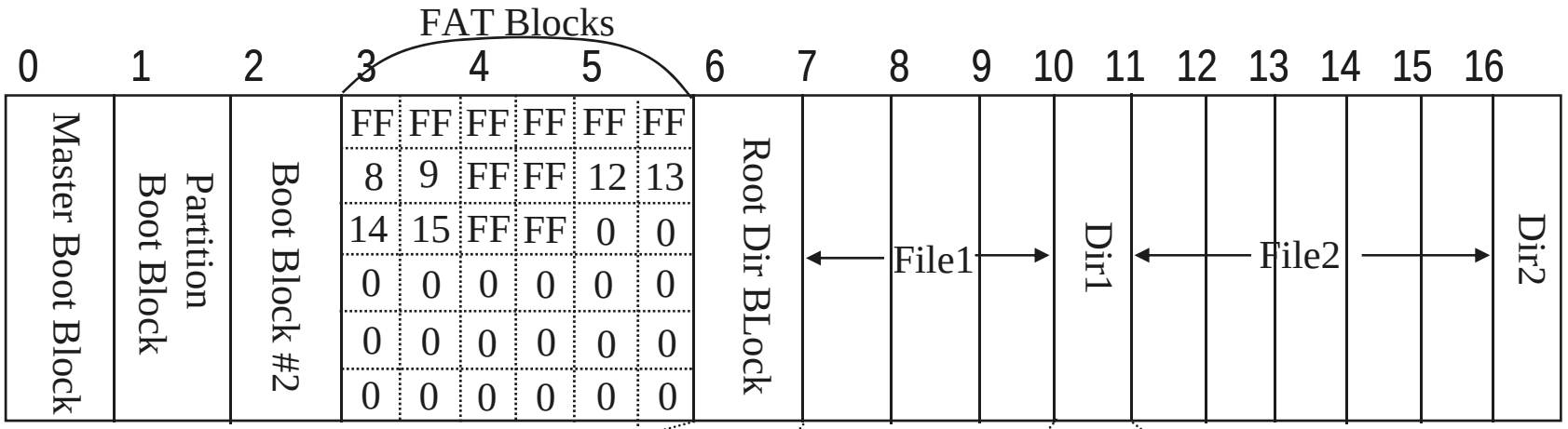


Multi-CPU를 위한 리눅스 특징

- 스케줄링
  - ✓ Run queue per each CPU
  - ✓ Effective priority based O(1) scheduler
    - Priority, affinity, interactivity, …
  - ✓ Load balancing



FAT internal

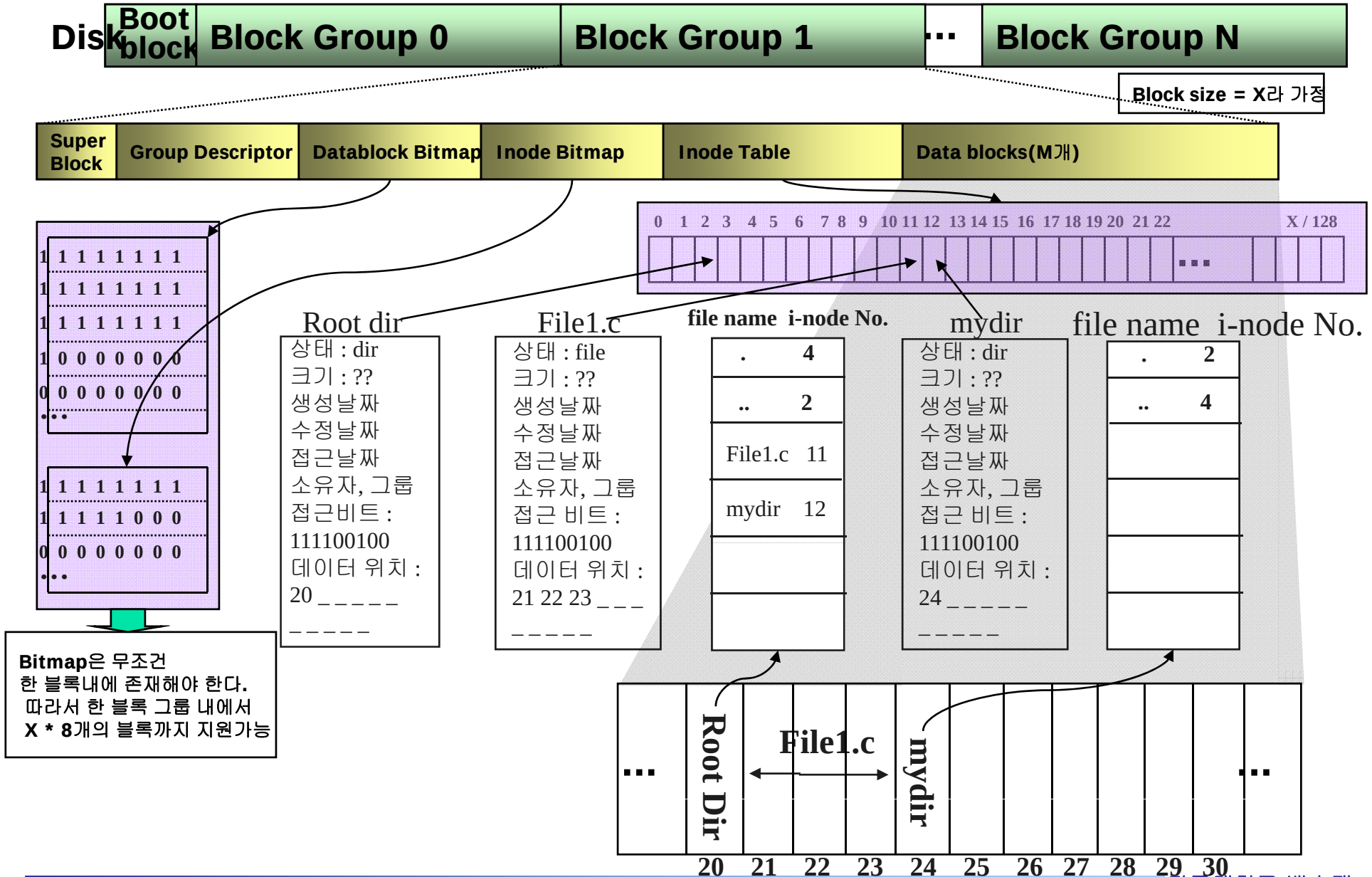


Status	File Name	FirstAddr	Size
F	File1	7	3
D	Dir1	10	1
F	File2	11	5
D	Dir2	16	1

Status	File Name	FirstAddr	Size
D	.	10	?
D	..	6	?
F	x.hwp	17	5
F	my.doc	22	1

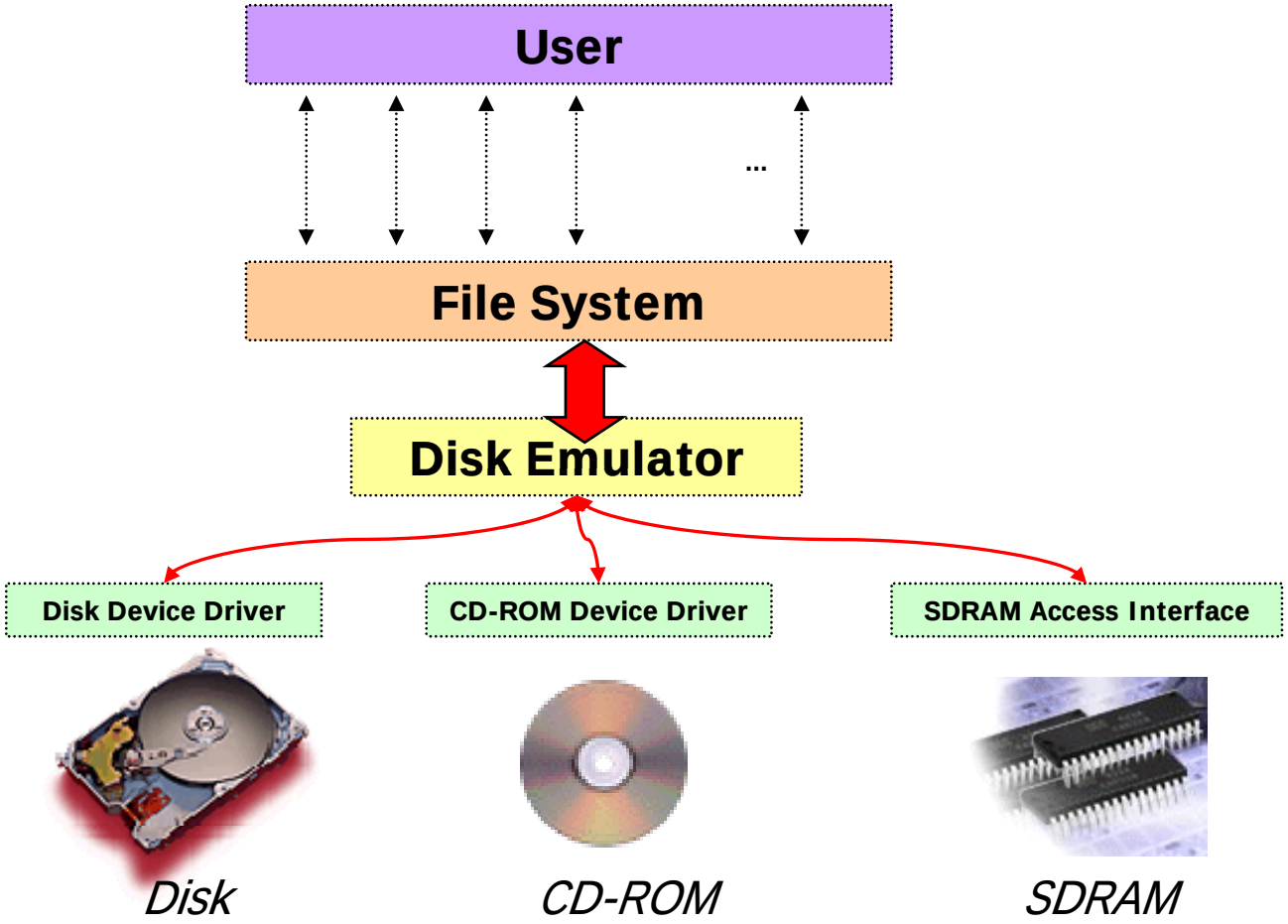
(Source : Dr. dhlee)

Ext2 파일시스템의 디스크에서의 레이아웃



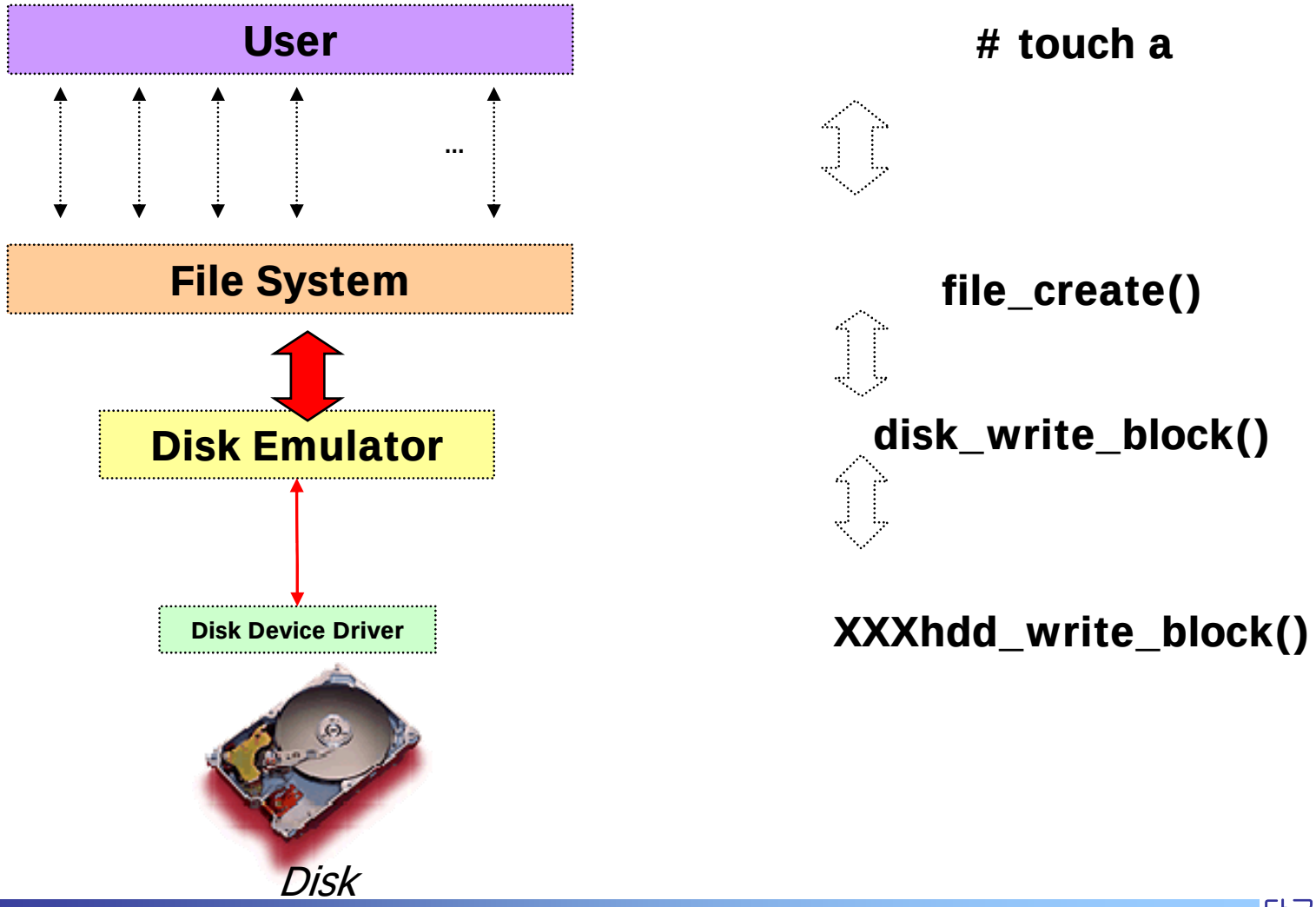
Example-FS제작

- 만약 여러분이 FS제작자라면 만든 FS가 특정 device위에서만 돌도록 만드시겠습니까?



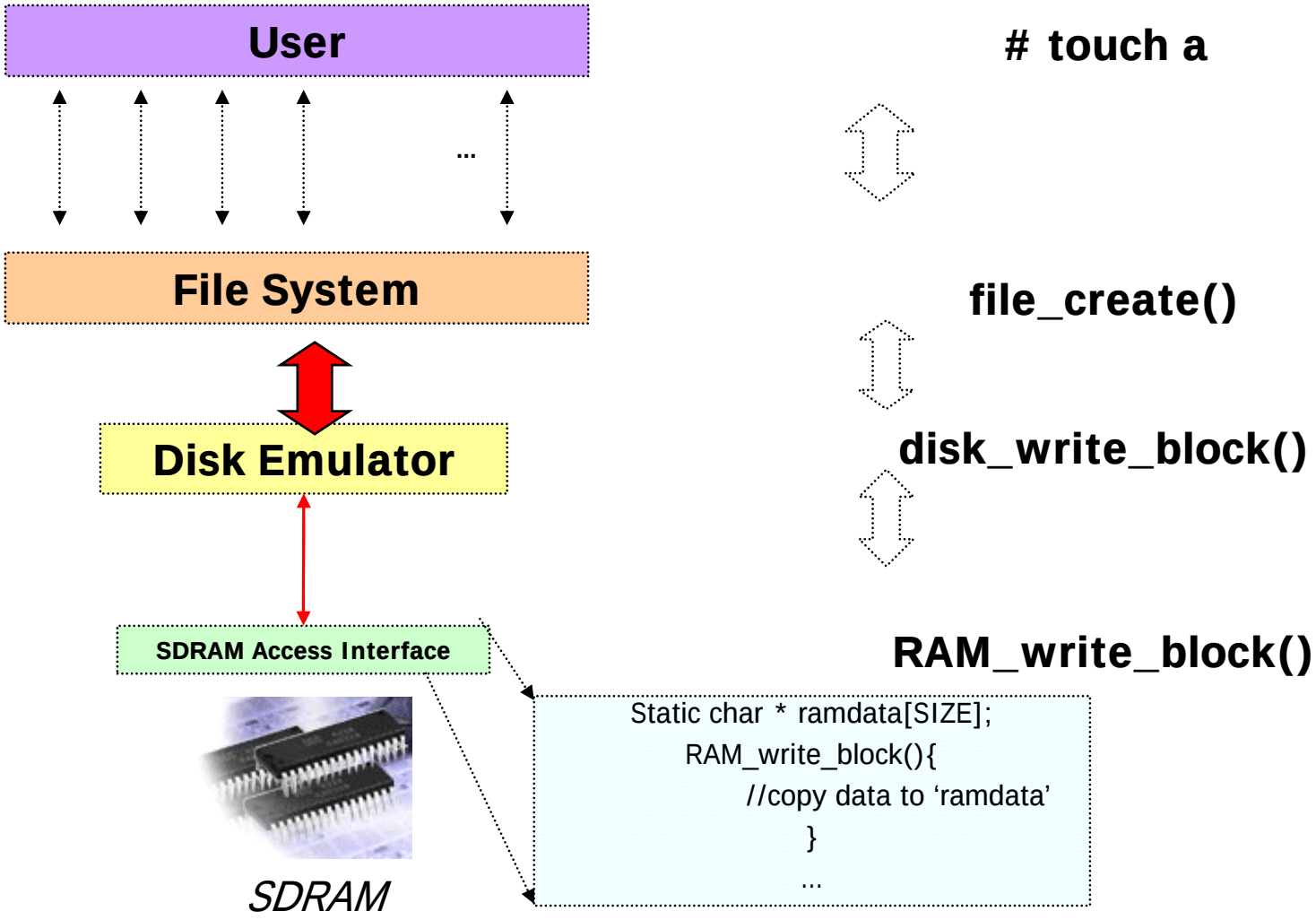
Example-FS제작

✓ 만약 Hard Disk 위 에서 돌리고 싶다면?



Example-FS제작

✓ 만약 RAM 상에서 돌리고 싶다면?



Example-FS제작

- FS함수 구현의 예(ls)

```
int MYFS_dir_read()
{
   int i, sector, offset, files_in_dir;
   DIRENTRY_T  *ep;

   SM_P();
   ep = (DIRENTRY_T   *)buf;
   sector = cur_dir;
   offset = 0;

   disk_read_block(sector, buf);
   files_in_dir = ep->size;

   for (i = 0; i < files_in_dir; i++) {
      disk_read_block(sector, buf);
      ep = (DIRENTRY_T *) (buf+offset);
      if (ep->status == DIR) printk("  %14s/  %d", ep->name, ep->size);
      else printk("  %14s  %d", ep->name, ep->size);  /* ep->status == FILE */
      offset += sizeof(DIRENTRY_T);
      if (offset == 512) {
         sector = FAT[sector];
         offset = 0;
      }
      if (!((i+1) % 4)) printk("₩n");
   }
   printk("₩n");

   SM_V();
   return(FS_SUCCESS);
```

Example-FS제작

- create, read, write …. 의 함수는?

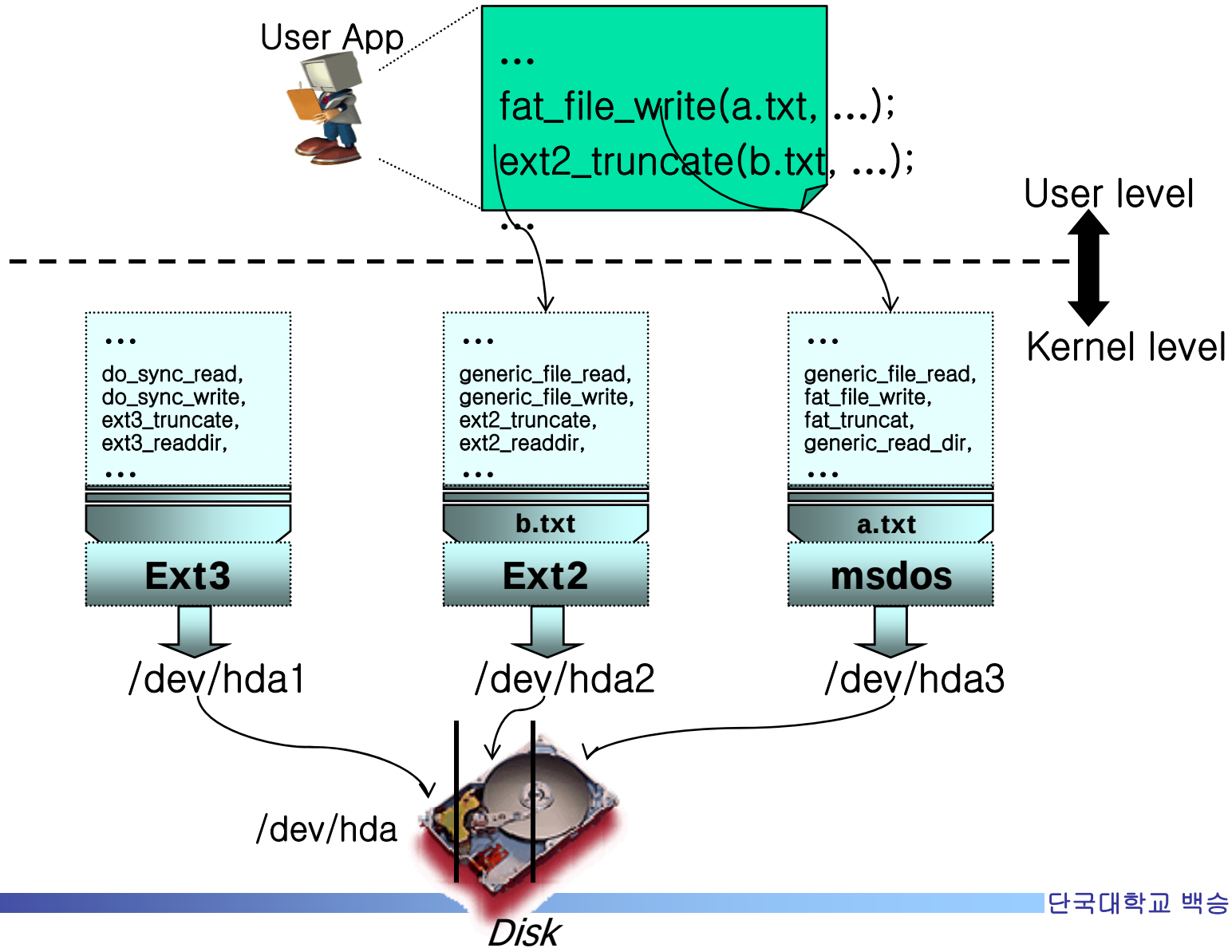
- cp ./a.txt /b.txt

```
Int cp( .. .. )
{
        int a, b, rb;
        char buf[SIZE_OF_BLOCK];
        …
        a = MYFS_file_open(./a.txt);
        b = EXT2_file_open(/b.txt);
        rb = MYFS_file_read(a, buf);
        while( rb ){
                EXT2_file_write(b, rb, buf);
                …
                rb = MYFS_file_read(a, buf);
                …
        }
        …
}
```

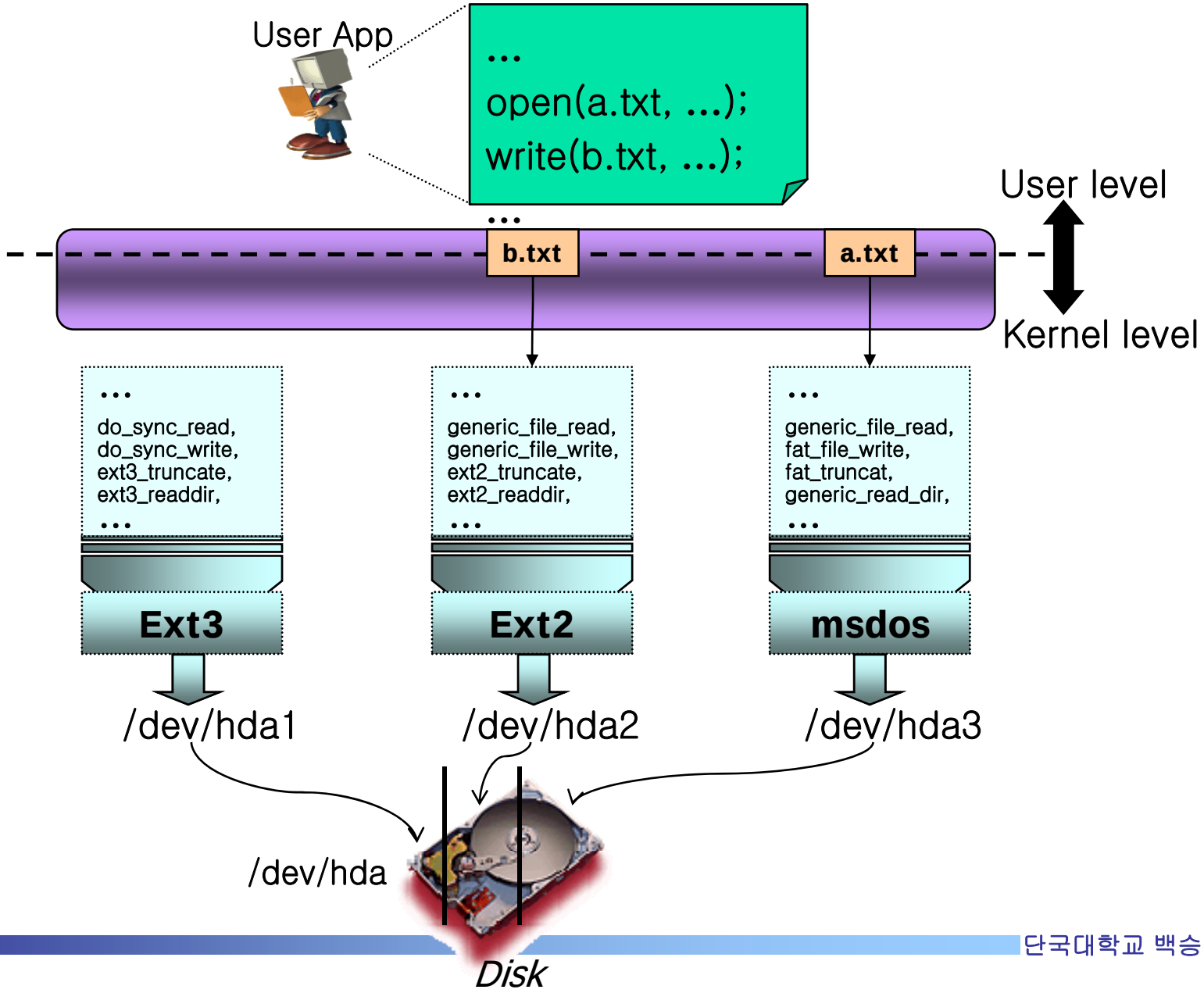
정말 **cp** 프로그램을 이렇게 만들어 줘야 할까**?**

이렇게 **FS** 종속적이지 않도록 하려면**?**

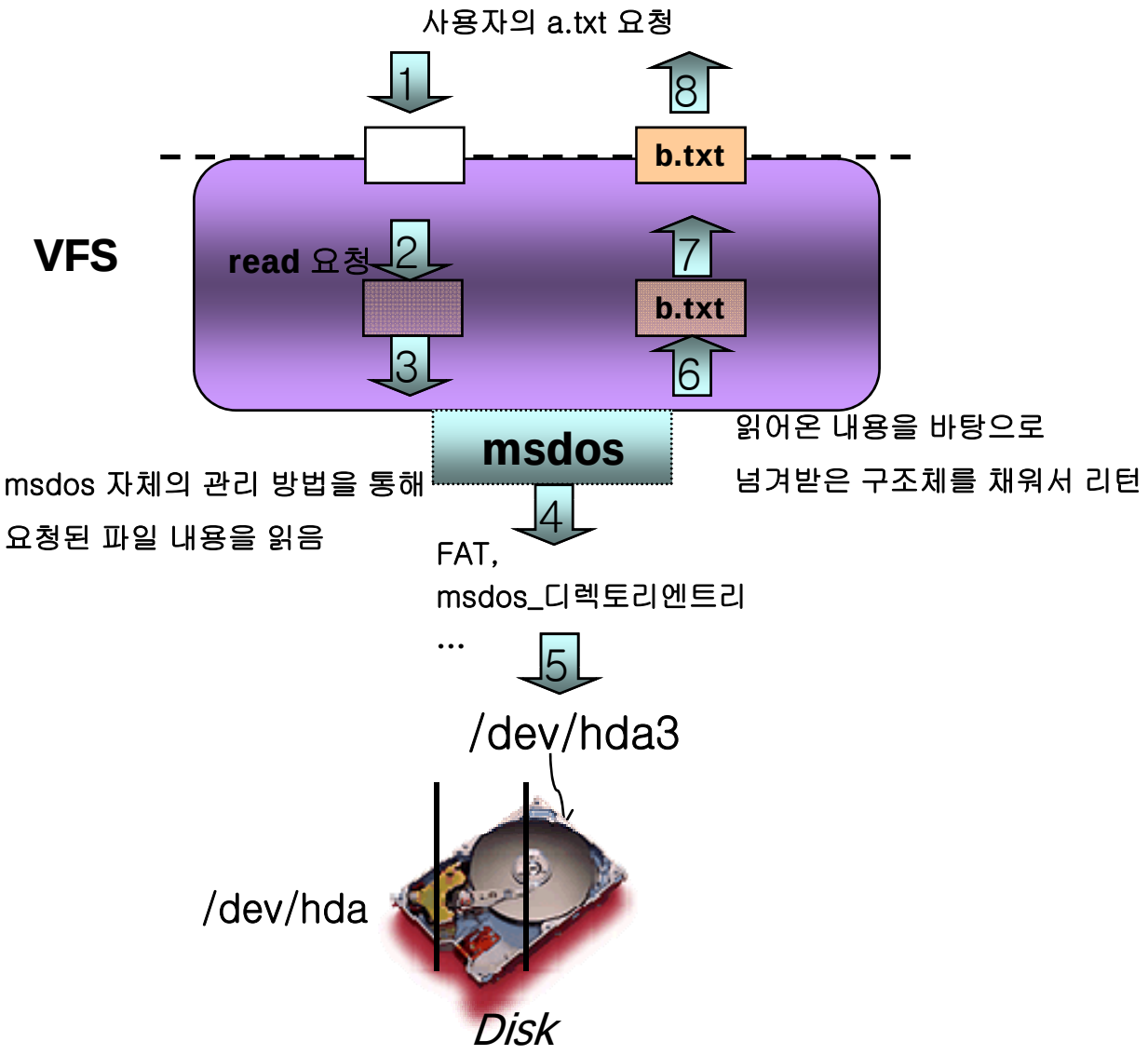
다양한 파일시스템 지원의 문제점



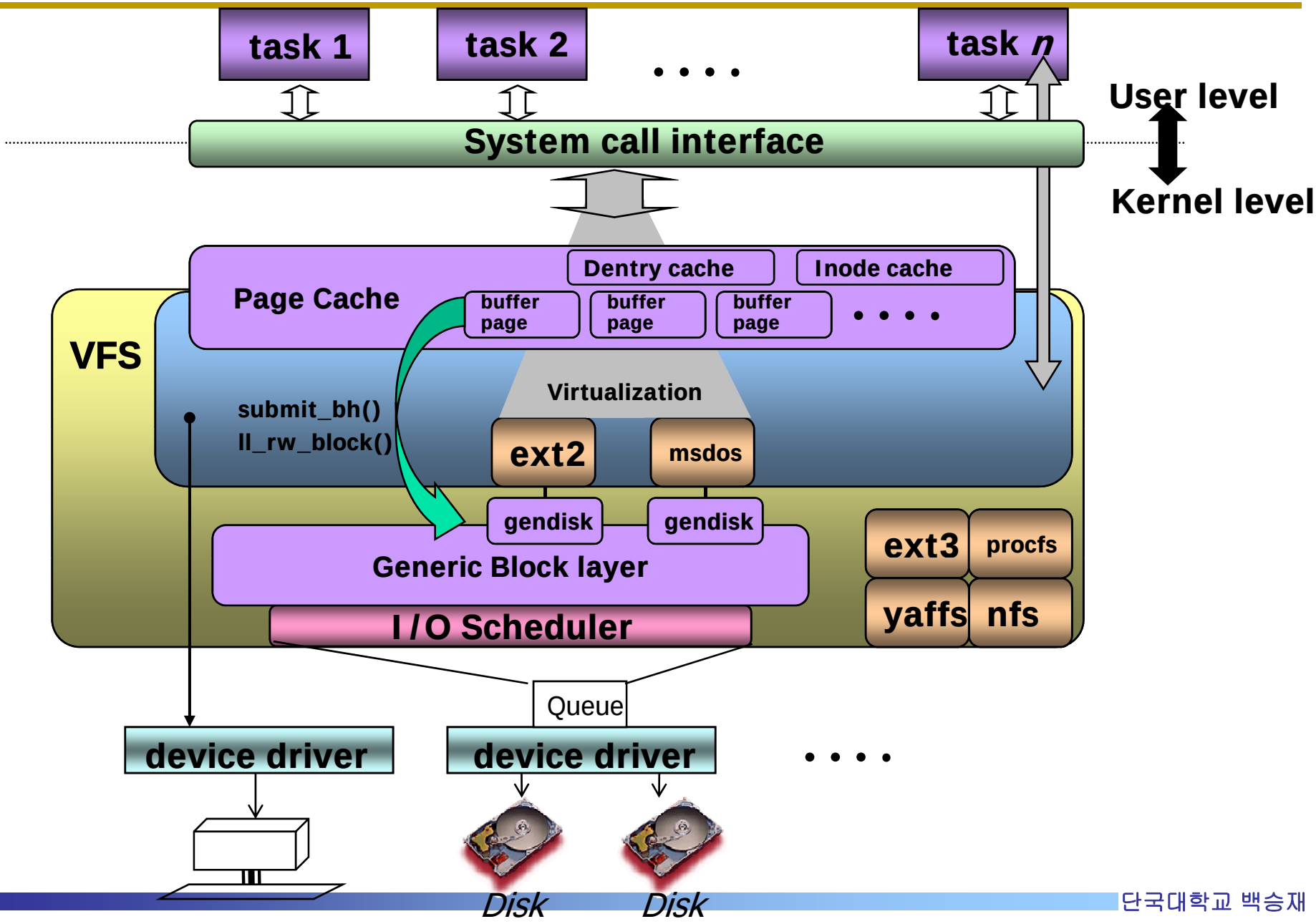
가상적인(Virtual) 계층의 도입



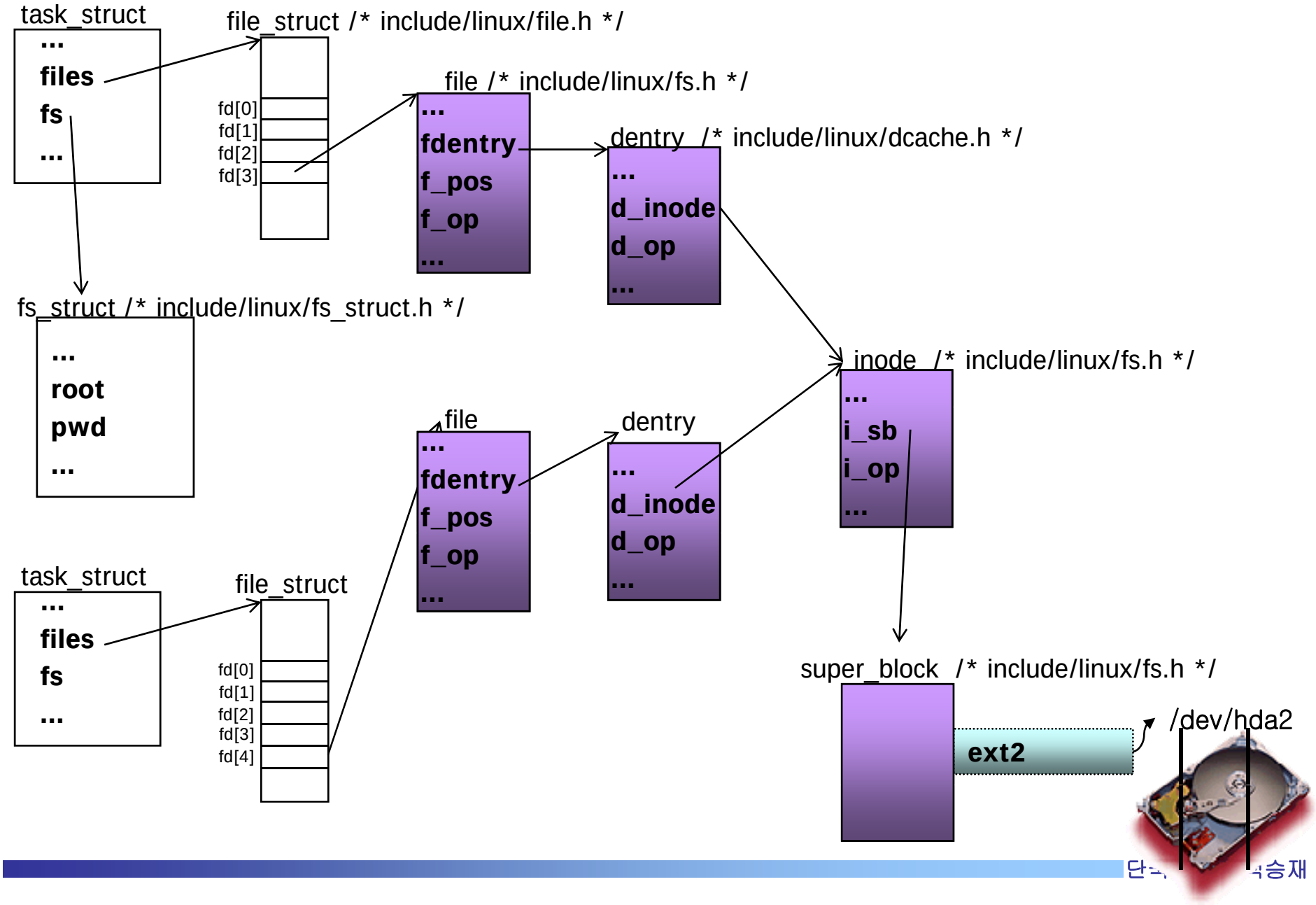
VFS의 동작 원리


사용자의 a.txt 요청
1
8
b.txt
VFS
read 요청
2
7
b.txt
3
6
msdos
msdos 자체의 관리 방법을 통해
요청된 파일 내용을 읽음
읽어온 내용을 바탕으로
넘겨받은 구조체를 채워서 리턴
4
FAT,
msdos_디렉토리엔트리
...
5
/dev/hda3
/dev/hda
Disk

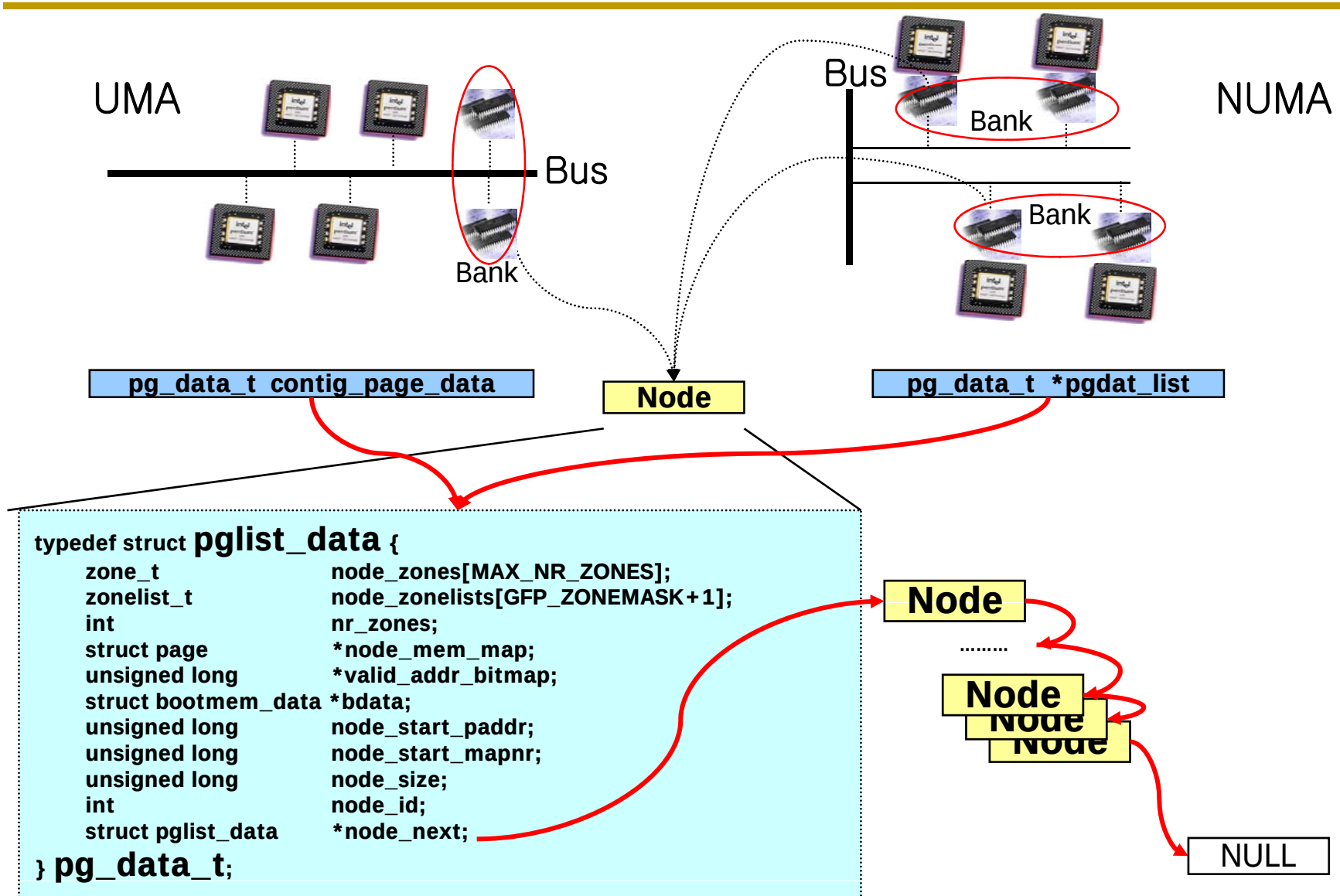
Linux 파일 시스템 구조



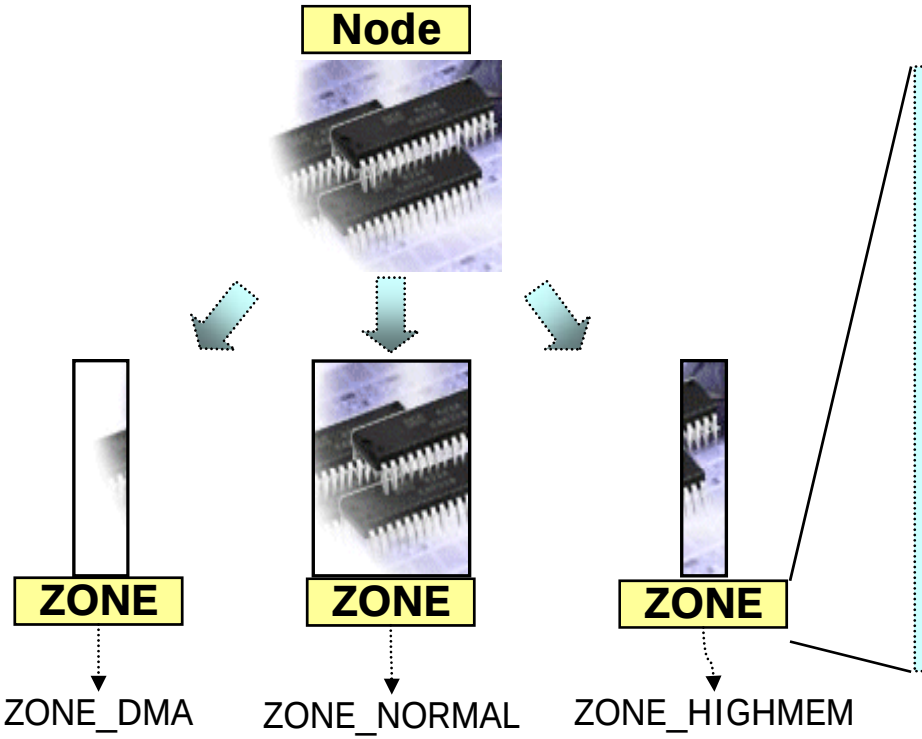
task_struct와 VFS의 객체



Bank and Node



Node and Zone




```
typedef struct zone_struct {
        spinlock_t              lock;
        unsigned long           free_pages;
        unsigned long           pages_min, pages_low, pages_high;
        int                     need_balance;
        free_area_t             free_area[MAX_ORDER];
        wait_queue_head_t       * wait_table;
        unsigned long           wait_table_size;
        unsigned long           wait_table_shift;
        struct pglist_data      *zone_pgdat;
        struct page             *zone_mem_map;
        unsigned long           zone_start_paddr;
        unsigned long           zone_start_mapnr;
        char                    *name;
        unsigned long           size;
} zone_t;
```

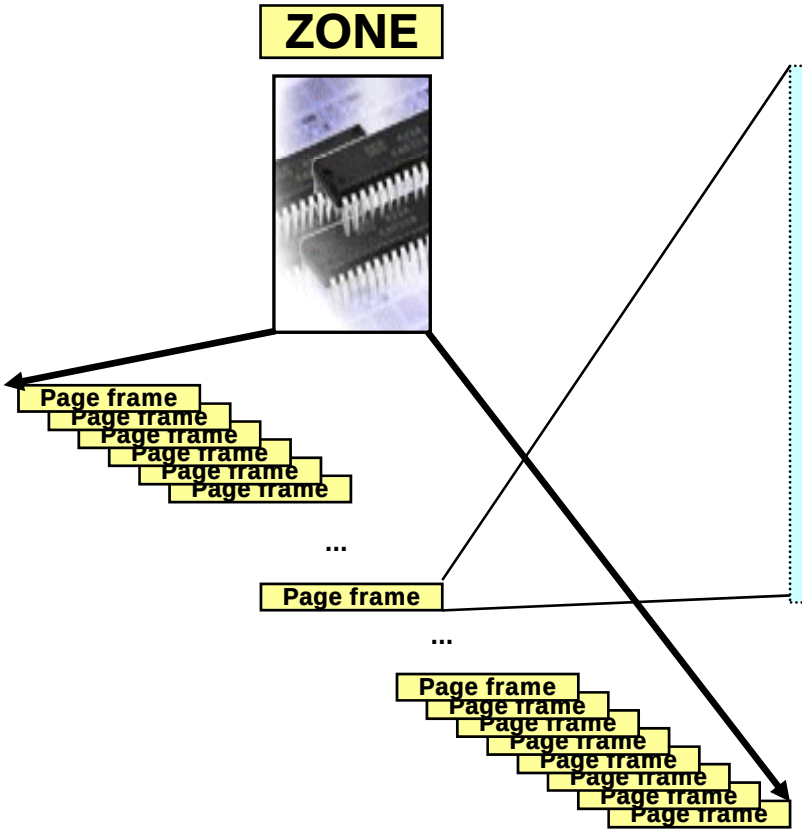
X86 System 에서는

ZONE_DMA ➔ 0 ~ 16M

ZONE_NORMAL ➔ 16 ~ 896M

ZONE_HIGHMEM ➔ 896 ~ end

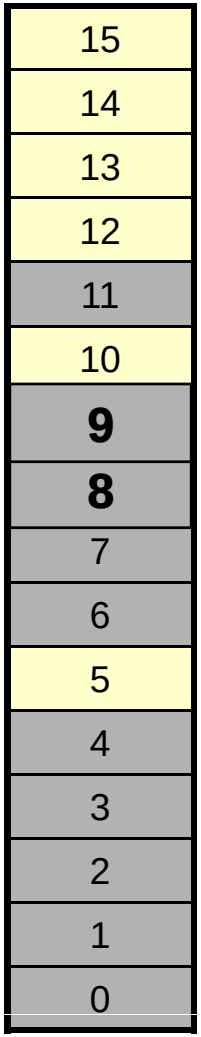
ZONE과 PageFrame



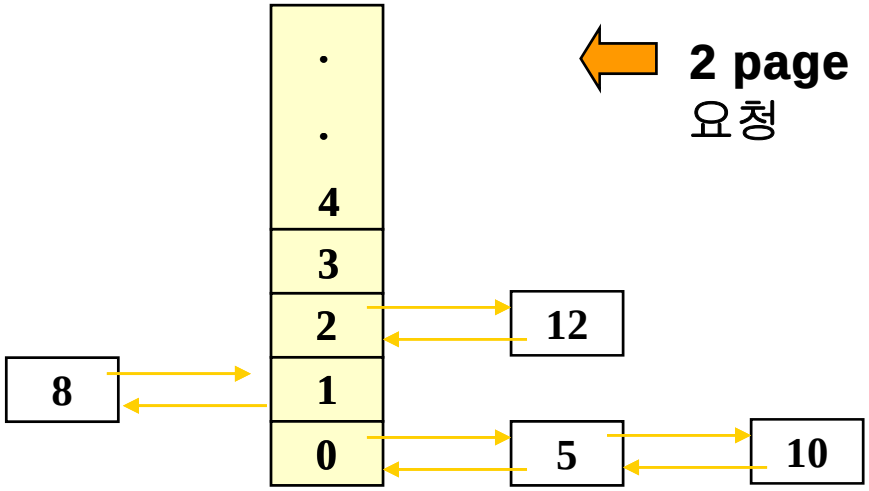

```
typedef struct page {
        struct list_head                list;
        struct address_space  *mapping;
        unsigned long                   index;
        struct page                     *next_hash;
        atomic_t                        count;
        unsigned long                   flags;
        struct list_head                lru;
        struct page                     **pprev_hash;
        struct buffer_head              * buffers;
#if defined(CONFIG_HIGHMEM) || defined(WANT_PAGE_VIRTUAL)
        void                            *virtual;
#endif /* CONFIG_HIGMEM || WANT_PAGE_VIRTUAL */
} mem_map_t;
```

각 NODE의 모든 page 구조체는 보통 ZONE_NORMAL의 시작부분
(혹은 커널이 올라 오기 위해 예약해 놓은 메모리 바로 다음)
위치에 있는 전역 배열인 "**mem_map**"에 유지된다

Memory allocation



Physical Memory

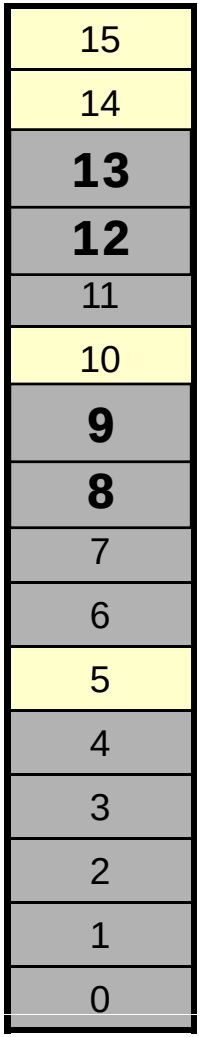


free_area.free_list

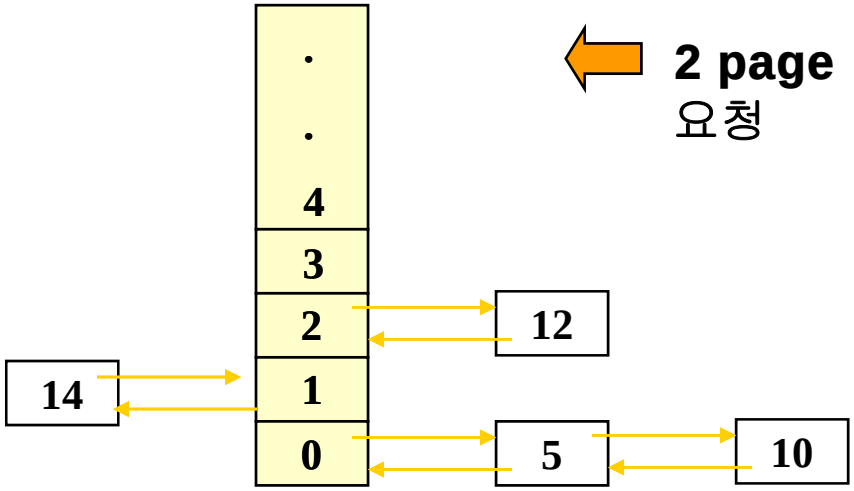
free_area[order].map

Pages	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
order(0)	0		0		1		0		0		1		0		0	
order(1)	0				0				1-> **0**				0			
order(2)	0								1							
order(3)	0															

Memory allocation



Physical Memory



free_area.free_list

2 page 요청

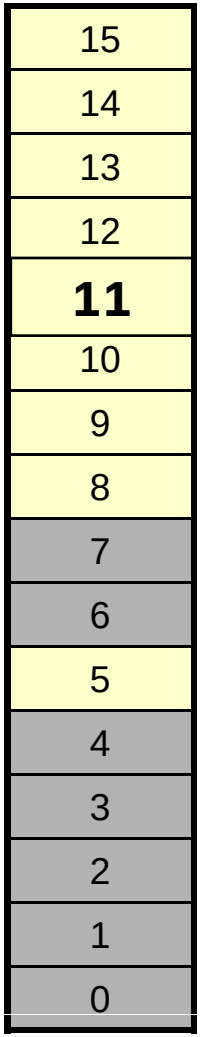
alloc

free

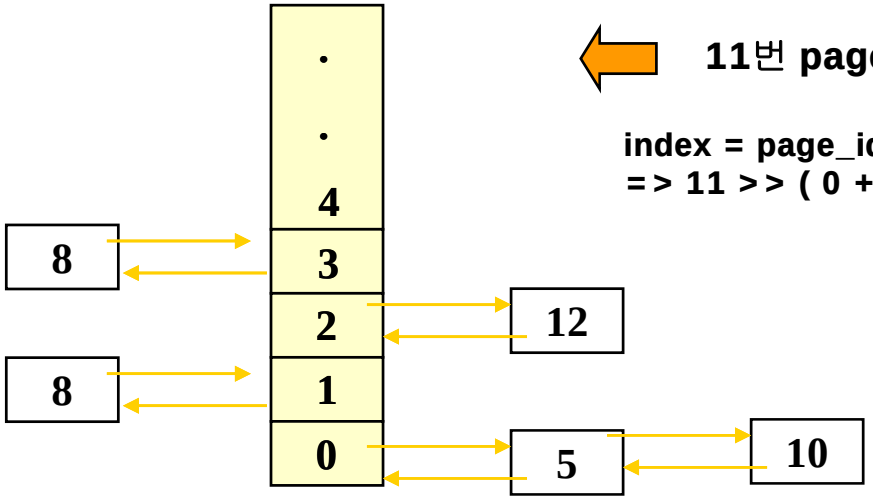
free_area[order].map

Pages	0	1	2	3	4	5	6	7	**8**	**9**	10	11	**12**	**13**	14	15
order(0)	0		0		1		0		0		1		0		0	
order(1)	0				0				0				0 **-> 1**			
order(2)	0								1 **-> 0**							
order(3)	0															

Memory de-allocation



Physical Memory



free_area.free_list

11번 page 해제 요청

index = page_idx >> (order + 1)
=> 11 >> (0 + 1) = 5

alloc

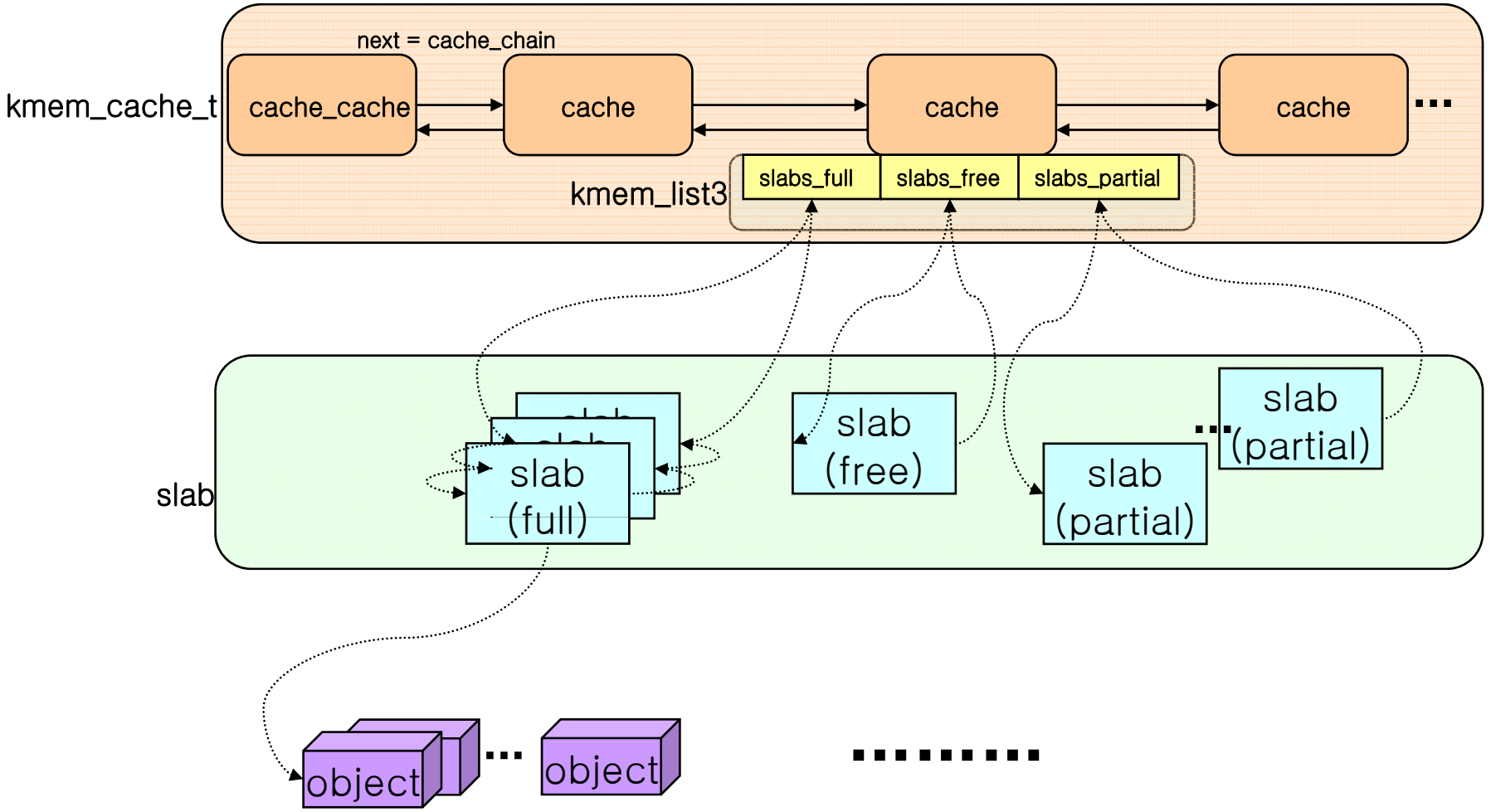
free

index >>= 1

free_area[order].map

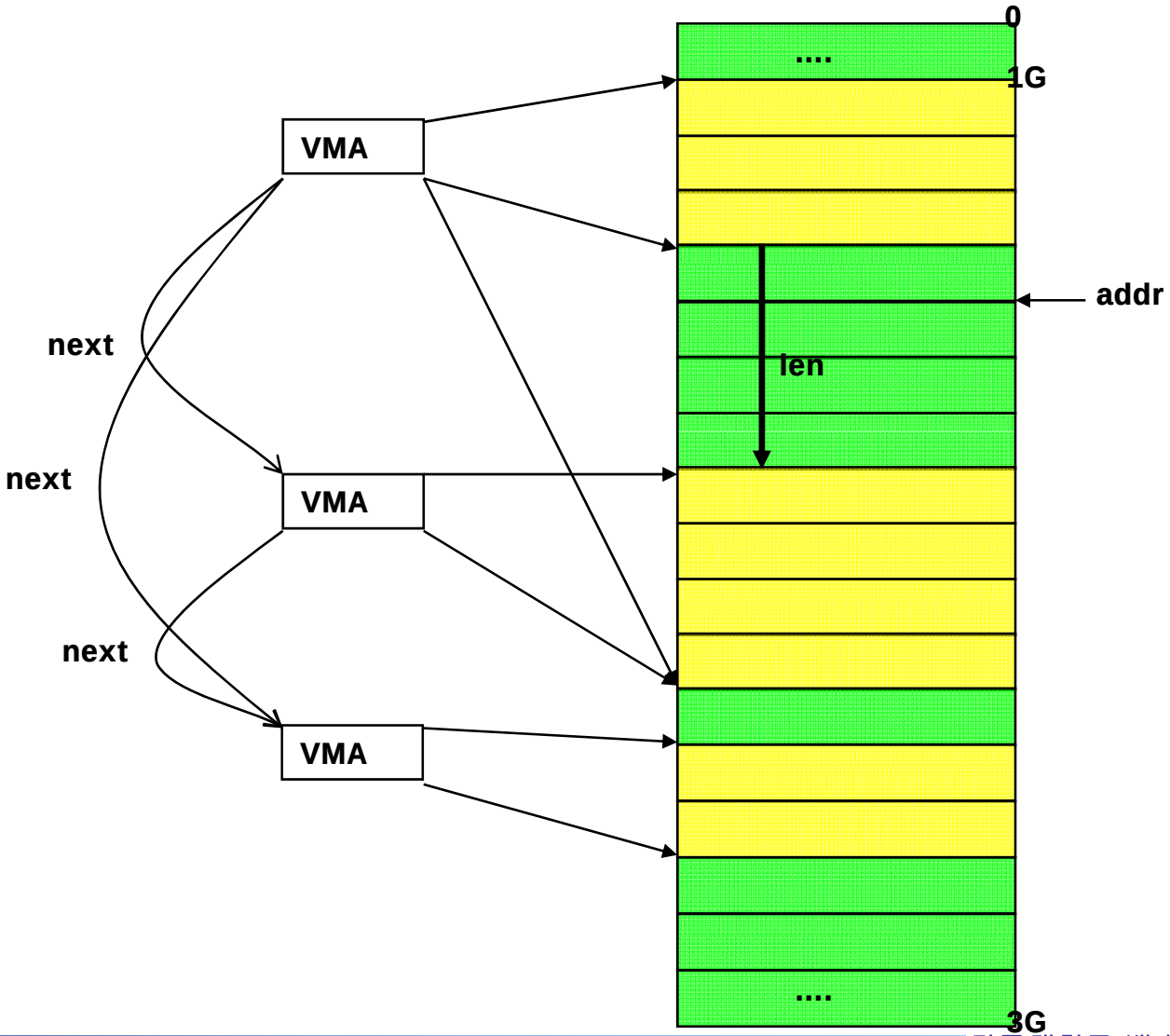
Pages	0	1	2	3	4	5	6	7	8	9	10	**11**	12	13	14	15
order(0)	0		0		1		0		0		1 **-> 0**		0		0	
order(1)	0				0				1 **-> 0**				0			
order(2)	0								1 **-> 0**							
order(3)	0 **-> 1**															

Slab Allocator 구조

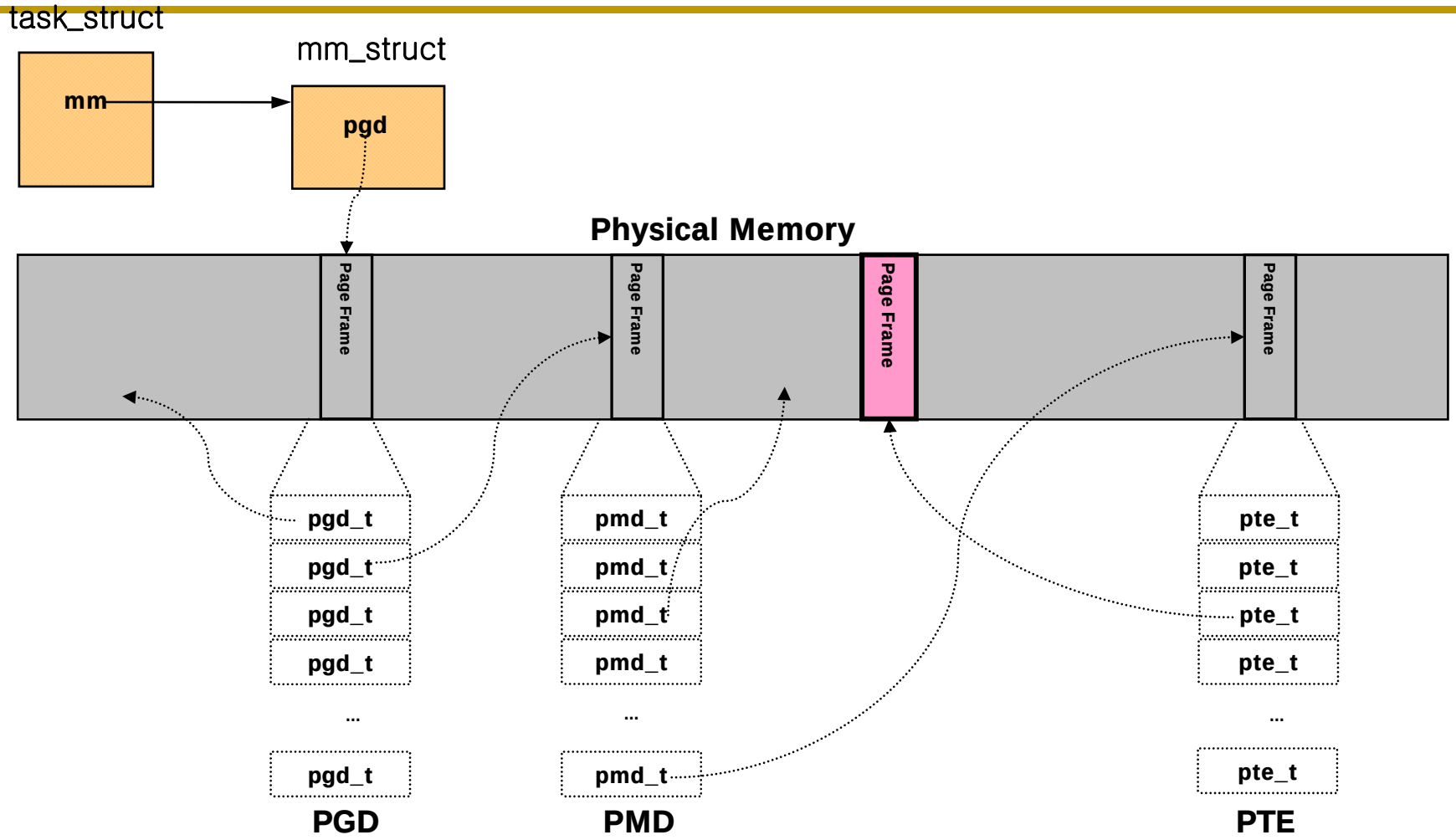


Merging contiguous region

- do_mmap_pgoff
  - ✓ merge-able



Linux의 PGD, PMD, PTE

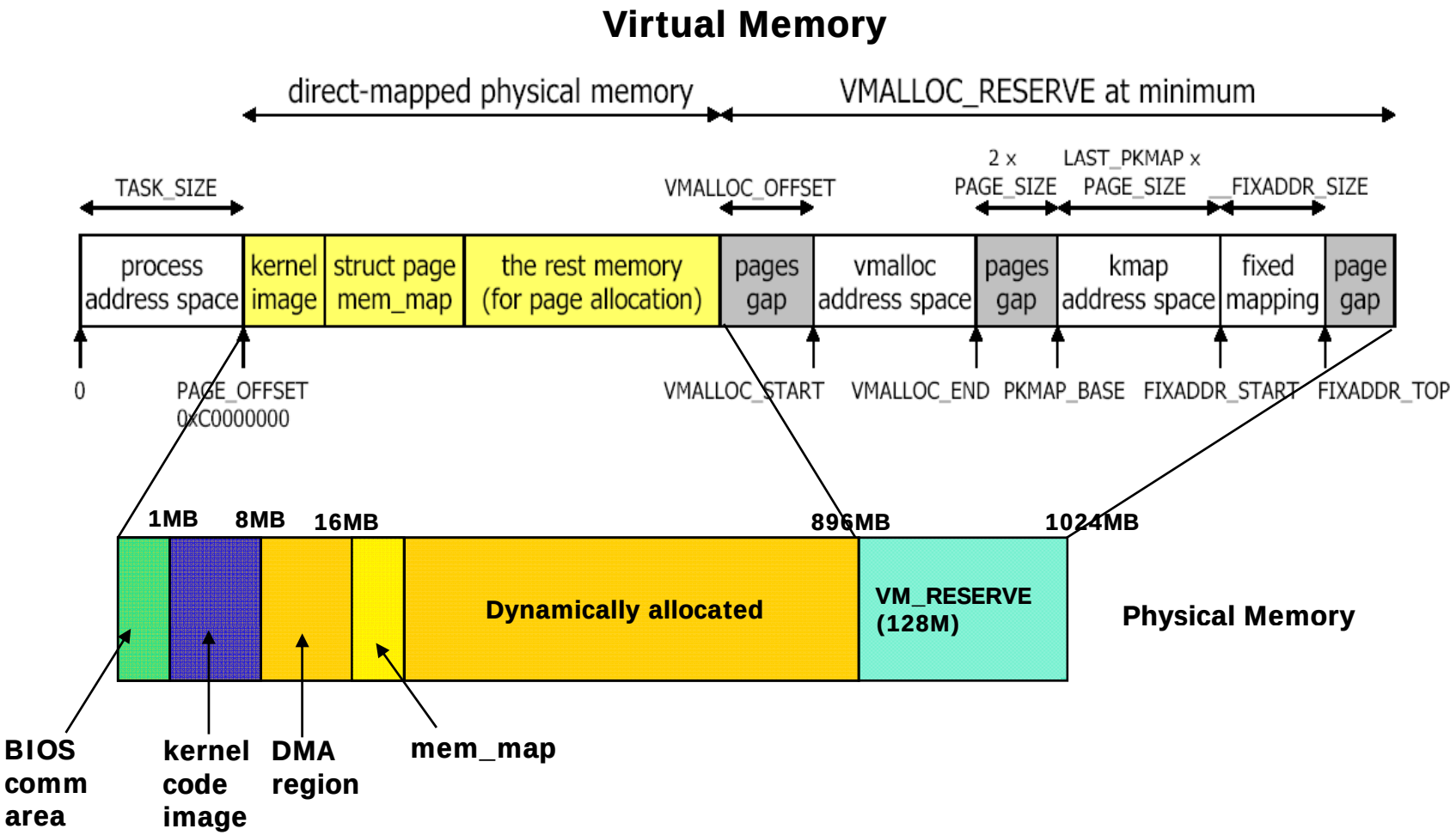


Swap out된 경우에는**?**

Swap entry가 **PTE**에 저장되어 있음
Fault 맞으면 이 **swap entry**를 이용해 **do_swap_page()** 함수가 **page**를 처리해 줌

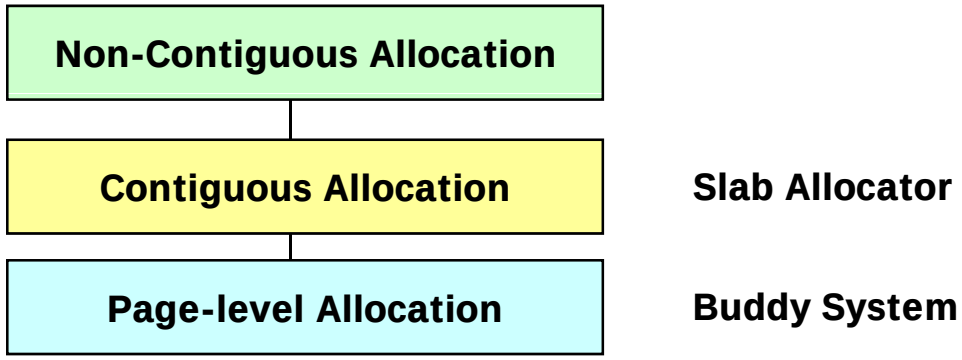
커널의 가상 주소 사용(32bit 처리기 기준)

- Kernel address space(32bit)

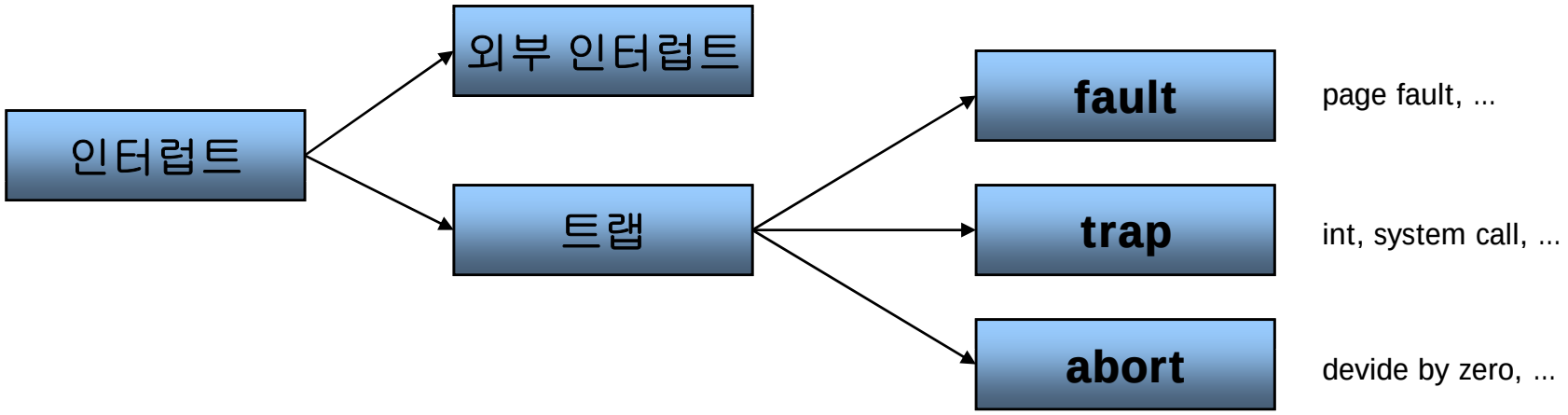


Linux 커널 메모리 관리 함수

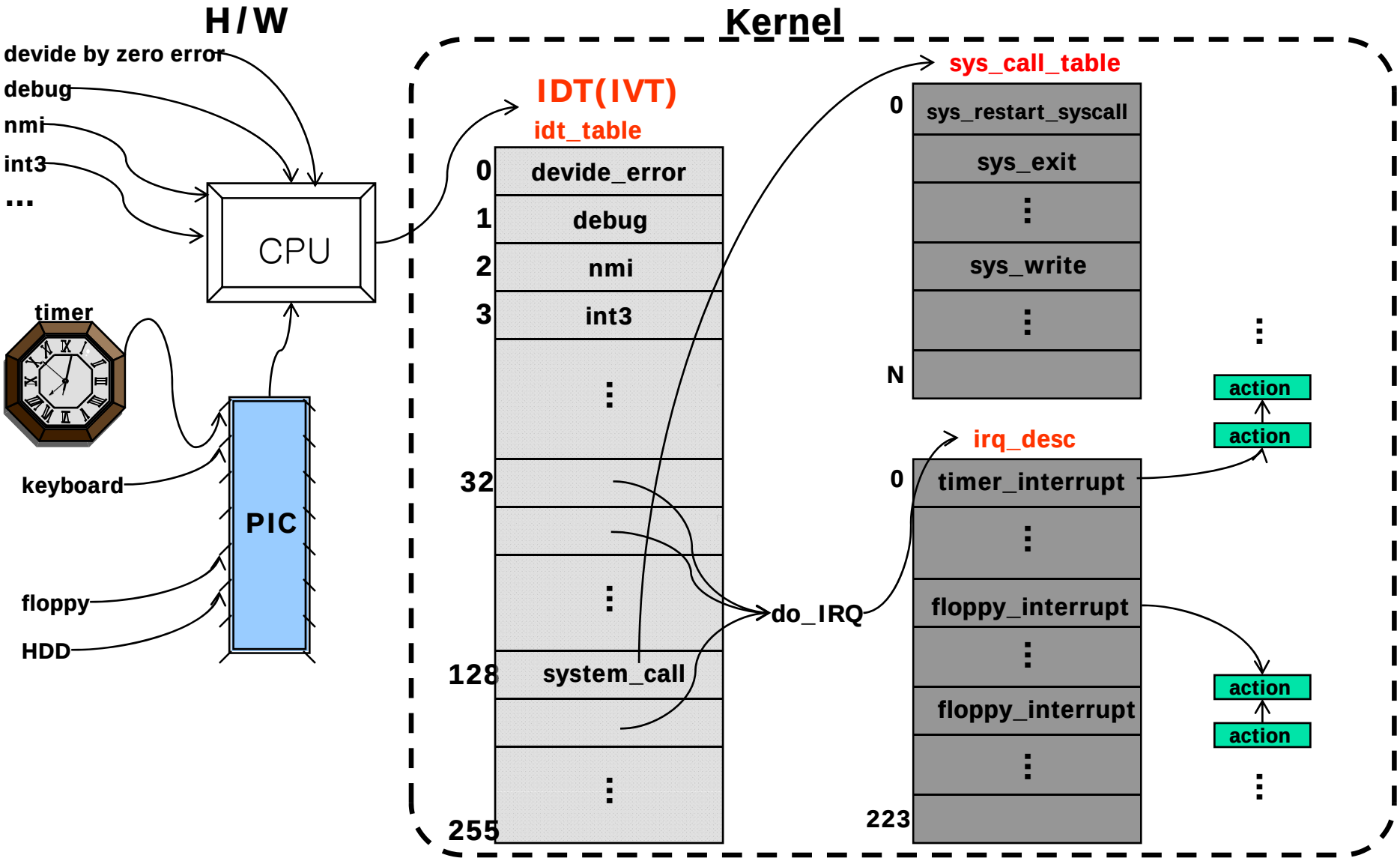
- Interface
  - ✓ struct page * alloc_pages(unsigned int gfp_mask, unsigned int order)
    - 2^order 크기의 연속된 물리적 페이지들을 할당 후, 첫 페이지의 page 구조체 포인터를 리턴
  - ✓ void * page_address(struct page *page)
    - 매개변수로 넘긴 물리적 페이지가 현재 포함돼 있는 논리적 주소에 대한 포인터 반환
  - ✓ Wrapping Function
    - Page-level allocator : get_free_page()
      - 요청된 만큼의 연속된 메모리를 할당
    - Kernel-level allocator
      - 물리적으로 **연속**하며, 128KB이하 임의 길의의 메모리 할당 : kmalloc()
      - 물리적으로 **비 연속적인** 메모리 할당 : vmalloc()



인터럽트의 분류


인터럽트
외부 인터럽트
트랩
fault
trap
abort
page fault, ...
int, system call, ...
devide by zero, ...

인터럽트와 트랩의 처리


H/W
devide by zero error
debug
nmi
int3
...
CPU
timer
keyboard
PIC
floppy
HDD
Kernel
IDT(IVT)
idt_table
0
devide_error
1
debug
2
nmi
3
int3
32
128
system_call
255
do_IRQ
sys_call_table
0
sys_restart_syscall
sys_exit
sys_write
N
irq_desc
0
timer_interrupt
floppy_interrupt
floppy_interrupt
223
action
action
action
action

system_call 함수

```
ENTRY(system_call)
        pushl %eax                      # save orig_eax
1       SAVE_ALL
2       GET_CURRENT(%ebx)
        testb $0x02,tsk_ptrace(%ebx)    # PT_TRACESYS
3       jne tracesys
        cmpl $(NR_syscalls),%eax
4       jae badsys
        call *SYMBOL_NAME(sys_call_table)(,%eax,4)
5       movl %eax,EAX(%esp)             # save the return
```

```
#define SAVE_ALL \
        cld; \
        pushl %es; \
        pushl %ds; \
        pushl %eax; \
        pushl %ebp; \
        pushl %edi; \
        pushl %esi; \
        pushl %edx; \
        pushl %ecx; \
        pushl %ebx; \
        movl $(__KERNEL_DS),%edx; \
        movl %edx,%ds; \
        movl %edx,%es;
```

넘어온 매개 변수를 커널 모드 스택에 저장

```
badsys:
        movl $-ENOSYS,EAX(%esp)
        jmp ret_from_sys_call
```

1. 제어 유닛이 자동으로 저장한 eflags, cs, eip, ss, esp제외한 모든 reg를 스택에 저장
2. ebx reg에 current P의 디스크립터를 저장
3. current의 ptrace필드에 PT_TRACESYS플래그가 들어 있는지, 즉 디버거가 프로그램의 시스템콜 호출을 추적 중인지 검사 ➔ syscall_trace()를 처음, 마지막 두번 호출하게 됨
4. 올바른 syscall번호인지 검사. 잘못된 번호이면 바로 종료
5. dispatch table의 각 엔트리는 4바이트이므로, 시스템 콜 번호에 4를 곱한 후 + sys_call_table 시작주소를 더해서 ➔ 서비스 루틴의 ptr얻어와서 호출함 . 호출 종료되면 리턴값을 저장한 스택(사용자 모드에서의 eax)에 저장해놓고, syscall핸들러를 종료하는 ret_from_sys_call로 점프

모듈 프로그래밍 예 (1/3) - 2.6

- 2.6 커널을 위한 hello_module

```
#include <linux/kernel.h>
#include <linux/module.h>

int hello_module_init(void)
{
	printk(KERN_EMERG "Hello Module~! I'm in Kernel₩n");
	return 0;
}

void hello_module_cleanup(void)
{
	printk("<0>Bye Module~!₩n");
}

module_init(hello_module_init);
module_exit(hello_module_cleanup);

MODULE_LICENSE("GPL");
```

모듈 프로그래밍 예 (2/3) – 2.6

- 2.6 커널에서 모듈 컴파일을 위한 Makefile

```
O_TARGET        := hello_module.ko
obj-m           := hello_module.o

KERNEL_DIR      := /lib/modules/$(shell uname -r)/build
MODULE_DIR      := /lib/modules/$(shell uname -r)/kernel/hello_module
PWD             := $(shell pwd)

default :
        $(MAKE)  -C  $(KERNEL_DIR)  SUBDIRS=$(PWD)  modules
install :
        mkdir  -p  $(MODULE_DIR)
        cp  -f  $(O_TARGET)  $(MODULE_DIR)
clean :
        $(MAKE)  -C  $(KERNEL_DIR)  SUBDIRS=$(PWD)  clean
```

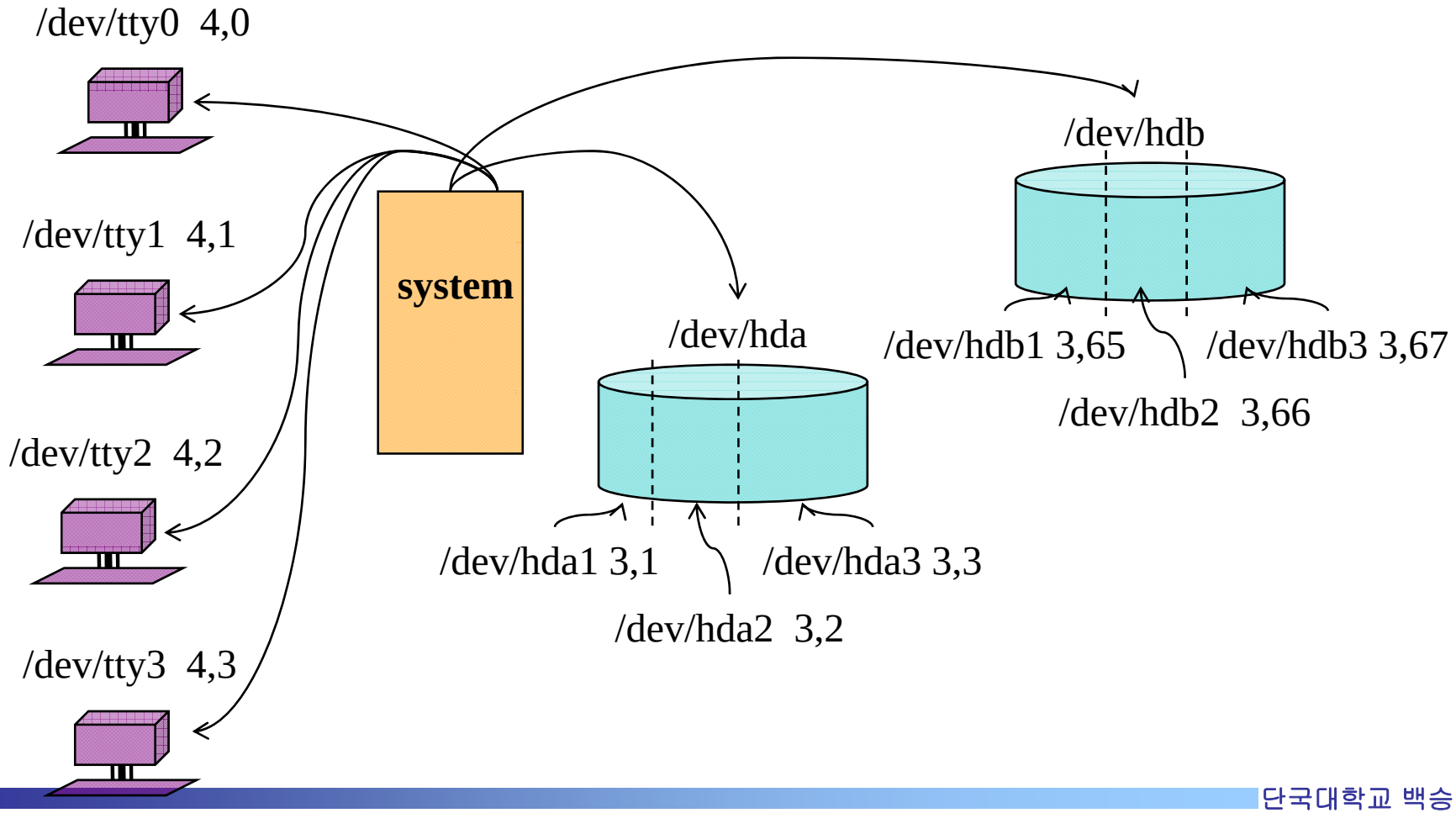
모듈 프로그래밍 예 (3/3) – 2.6

- 2.6 커널에서 모듈 테스트를 위한 명령어 수행

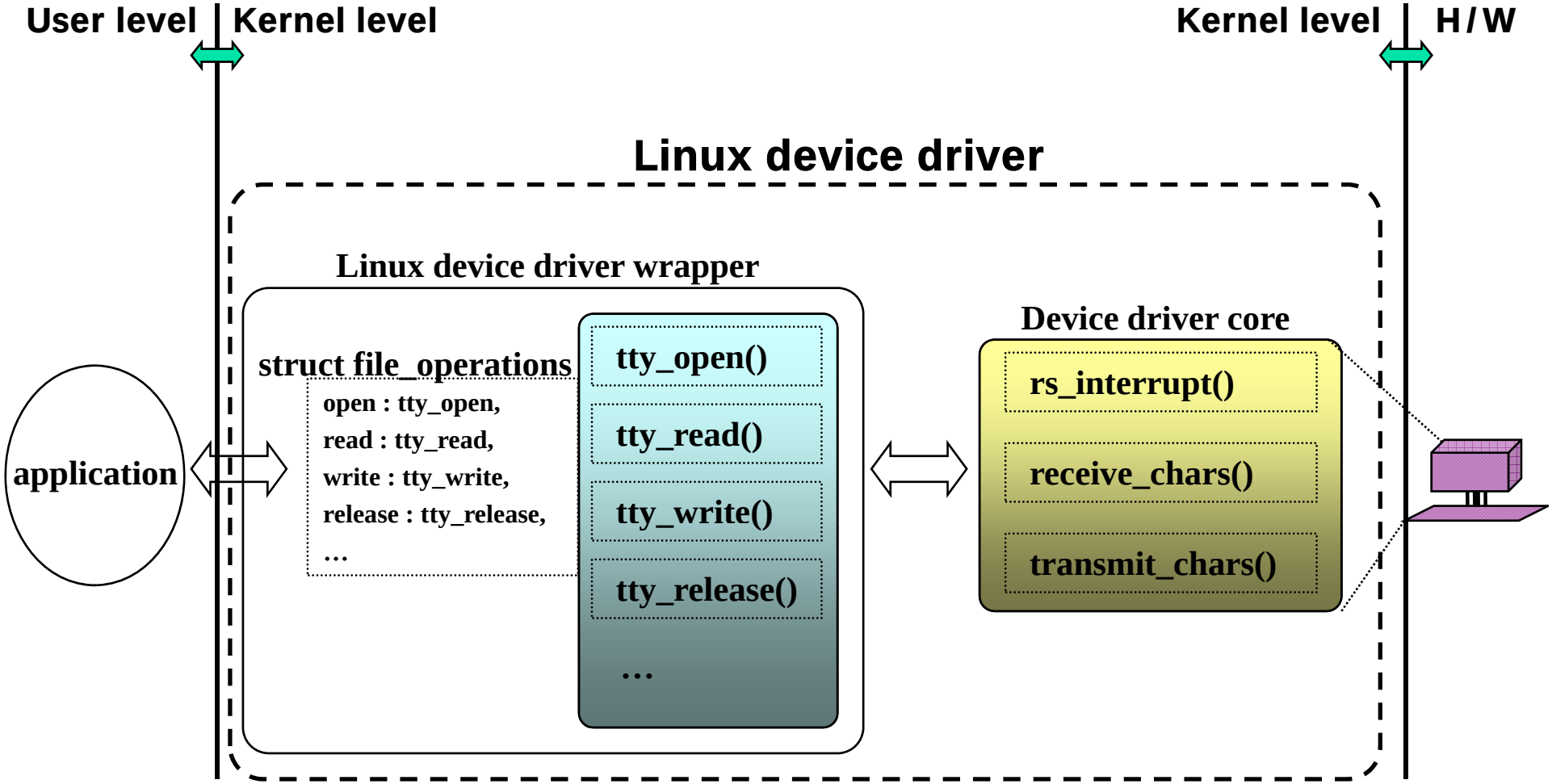
```
$ vi Makefile
$ vi hello_module.c
$ ls
Makefile        hello_module.c
$ make
$ ls
Makefile        hello_module.c        hello_module.ko        ……
$ insmod hello_module.ko
Hello Module~!  I'm in Kernel
$ rmmod hello_module
Bye Module~!
$ make install
…
$ ls  –l  /lib/modules/2.6.18/kernel/hello_module
hello_module.ko
$ make clean
…
$ ls
Makefile        hello_module.c
```

장치 파일 (1/3)

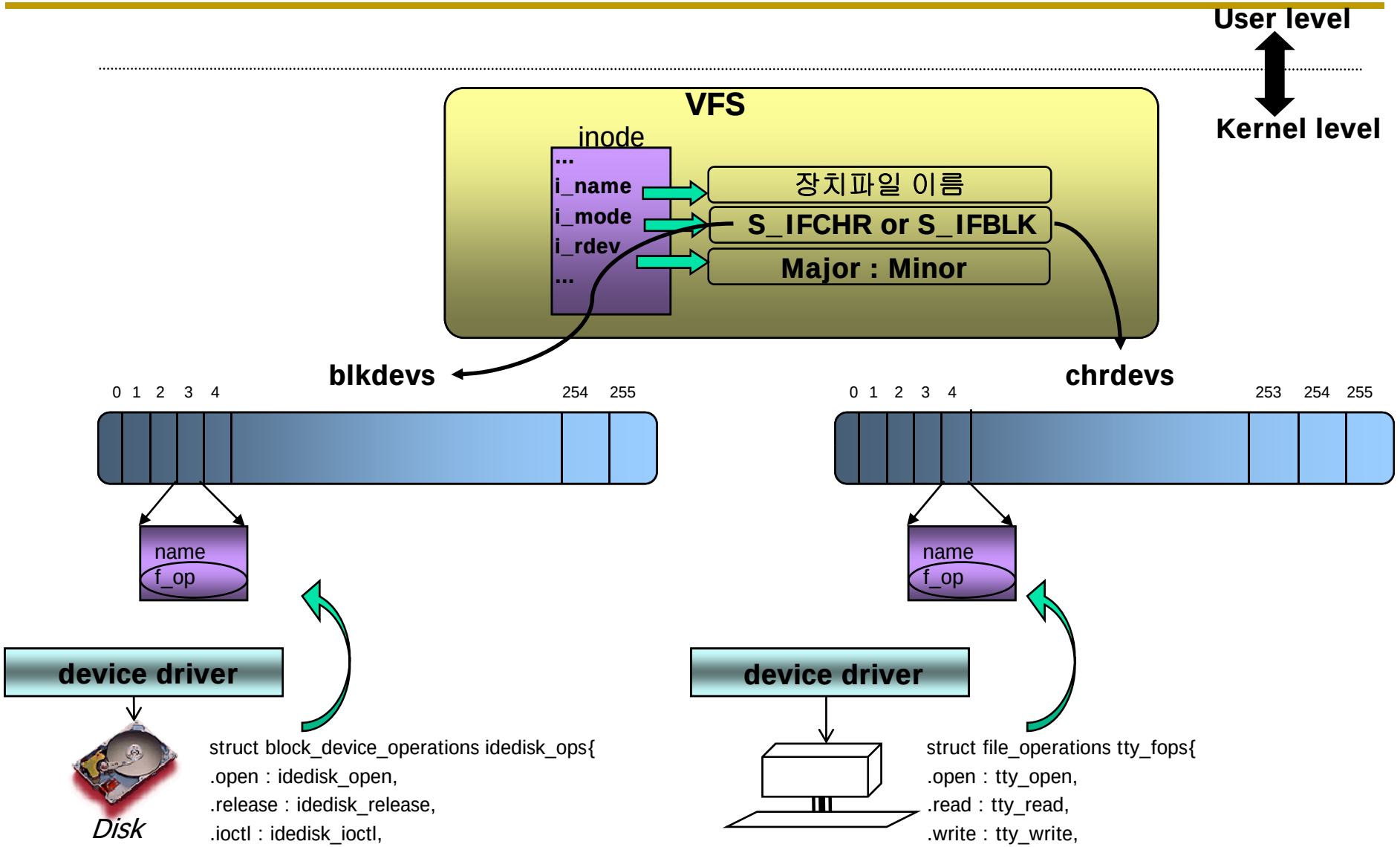
- 주 번호와 부 번호
  - ✓ 주 번호 : 디바이스 유형 (device type)
  - ✓ 부 번호 : 디바이스 단위 (device unit)



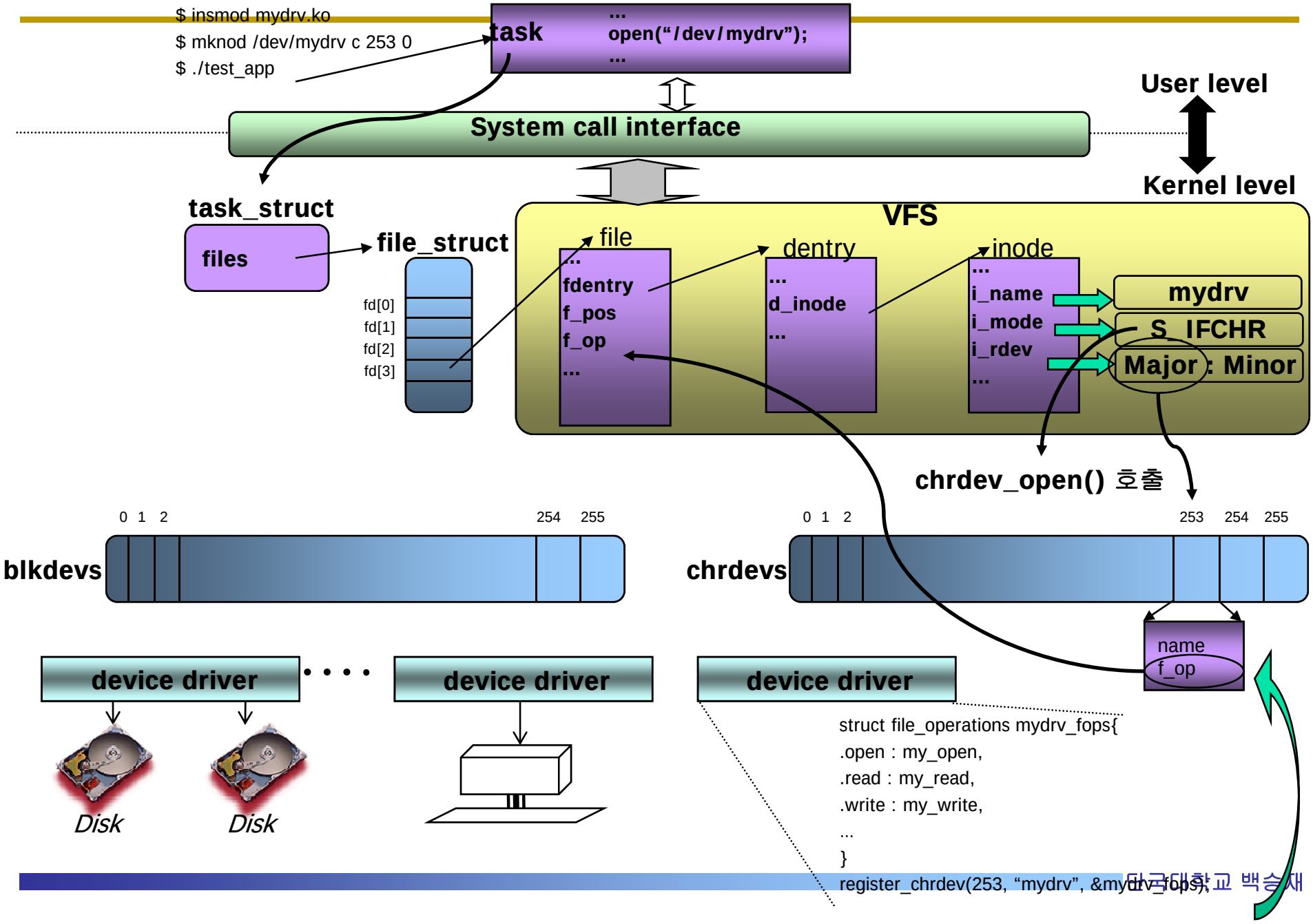
디바이스 드라이버 코어와 래퍼의 분리된 구조



디바이스 드라이버 관리 구조



디바이스 드라이버와 장치파일 그리고 VFS



모듈 프로그램 : Character Device Driver - 2.6

- 2.6 커널에서 character device driver

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/fs.h>
#include <asm/uaccess.h>

#define DEVICE_NAME    "mydrv"
#define MYDRV_MAX_LENGTH      4096
#define MIN(a, b) (((a) < (b)) ? (a) : (b))
static int MYDRV_MAJOR;
static char * mydrv_data;
static int mydrv_read_offset, mydrv_write_offset;

static int mydrv_open(struct inode *inode, struct file *file)
{
      if( MAJOR(inode->i_rdev) != MYDRV_MAJOR )
            return -1;
      return 0;
}

static int mydrv_release(struct inode * inode, struct file *file)
{
      if( MAJOR(inode->i_rdev) != MYDRV_MAJOR )
            return -1;
      return 0;
}

static int mydrv_ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg)
{
      printk("<0> mydrv_ioctl is invoked₩n");
      return 0;
}
```

모듈 프로그램 : Character Device Driver - 2.6

```
static ssize_t mydrv_read(struct file *file, char *buf, size_t count, loff_t *ppos)
{
	if( (buf == NULL) || (count < 0 ) )
		return -EINVAL;
	if( (mydrv_write_offset - mydrv_read_offset) <= 0 )
		return 0;
	count = MIN( (mydrv_write_offset - mydrv_read_offset), count );
	if( copy_to_user(buf, mydrv_data + mydrv_read_offset, count) )
		return -EFAULT;
	mydrv_read_offset += count;
	return count;
}

static ssize_t mydrv_write(struct file *file, const char *buf, size_t count, loff_t *ppos)
{
	if( (buf==NULL) || (count<0) )
		return -EINVAL;
	if( count+mydrv_write_offset >= MYDRV_MAX_LENGTH) {
		/* driver space is too small */
		return 0;
	}
	if( copy_from_user(mydrv_data + mydrv_write_offset, buf, count) )
		return -EFAULT;
	mydrv_write_offset += count;
	return count;
}
```

모듈 프로그램 : Character Device Driver - 2.6

```
struct file_operations mydrv_fops = {
	.owner = THIS_MODULE,
	.read = mydrv_read,
	.write = mydrv_write,
	.ioctl = mydrv_ioctl,
	.open = mydrv_open,
	.release = mydrv_release,
};

int mydrv_init(void)
{
	if( (MYDRV_MAJOR = register_chrdev(0, DEVICE_NAME, &mydrv_fops)) < 0 ){
		printk("<0> can't be registered₩n");
		return MYDRV_MAJOR;
	}
	printk("<0> major NO = %d₩n", MYDRV_MAJOR);
	if( (mydrv_data = (char *) kmalloc(MYDRV_MAX_LENGTH * sizeof(char), GFP_KERNEL)) == NULL ){
		unregister_chrdev(MYDRV_MAJOR, DEVICE_NAME);
		return -ENOMEM;
	}
	mydrv_read_offset = mydrv_write_offset = 0;
	return 0;
}

void mydrv_cleanup(void)
{
	kfree(mydrv_data);
	unregister_chrdev(MYDRV_MAJOR, DEVICE_NAME);
}

module_init(mydrv_init);
module_exit(mydrv_cleanup);
```

모듈 프로그램 : Character Device Driver – 2.6

- 2.6 커널에서 character device driver 모듈을 컴파일하기 위한 Makefile

```
MY_TARGET      := chr_test.ko
obj-m          := chr_test.o

KERNEL_DIR     := /lib/modules/$(shell uname -r)/build
MODULE_DIR     := /lib/modules/$(shell uname -r)/kernel/chr_test
PWD            := $(shell pwd)

default :
	$(MAKE) -C $(KERNEL_DIR) SUBDIRS=$(PWD) modules
install :
	mkdir -p $(MODULE_DIR)
	cp -f $(MY_TARGET) $(MODULE_DIR)
clean :
	$(MAKE) -C $(KERNEL_DIR) SUBDIRS=$(PWD) clean
```

모듈 프로그램 : Character Device Driver – 2.6

- 2.6 커널에서 character device driver 테스트를 위한 명령어 수행

```
$ make
$ insmod  chr_test.ko
Major NO = 252
$mknod /dev/mydrv c 252 0
$echo “test string” >> /dev/mydrv
$cat /dev/mydrv
test string
$...
```