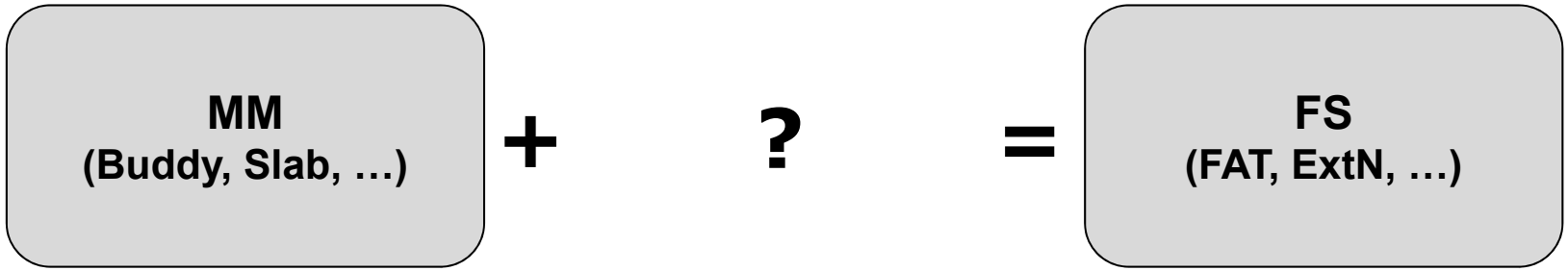
File System and VFS

May, 2016

Seungjae Baek

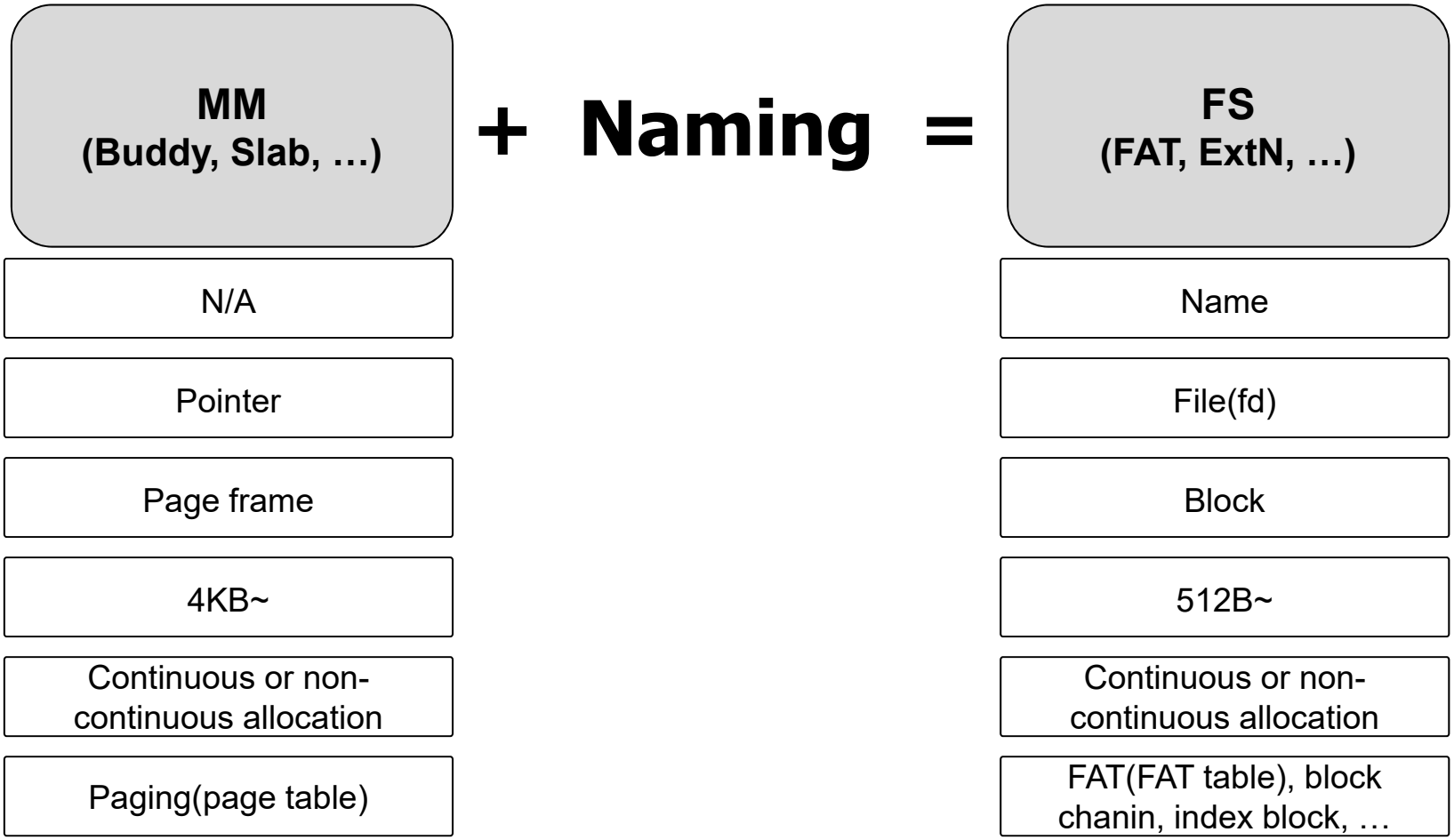
Dept. of software
Dankook University
http://embedded.dankook.ac.kr/~baeksj

File system

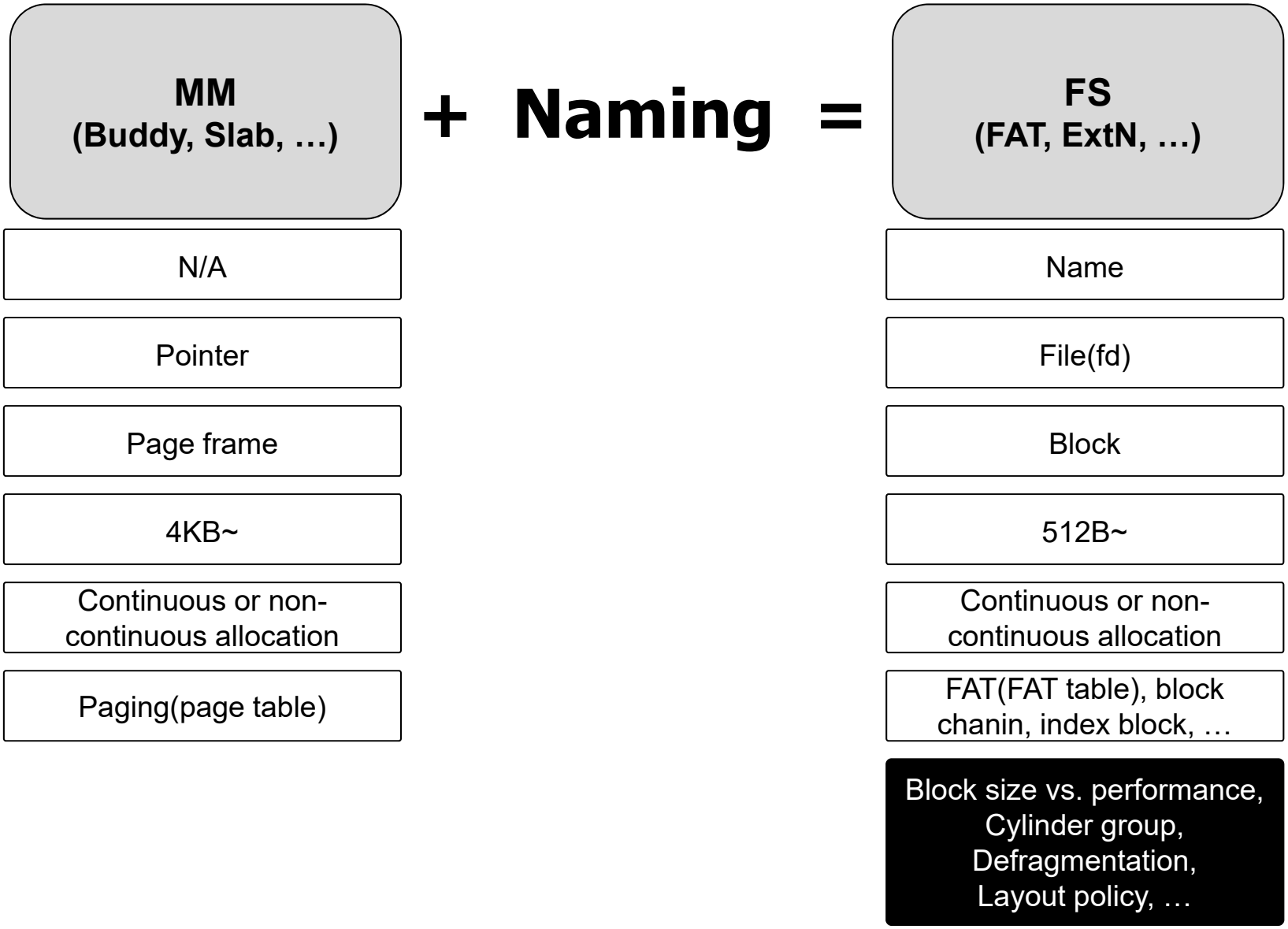


File system

File system


MM
(Buddy, Slab, …)
+
Naming
=
FS
(FAT, ExtN, …)
N/A
Pointer
Page frame
4KB~
Continuous or non-continuous allocation
Paging(page table)
Name
File(fd)
Block
512B~
Continuous or non-continuous allocation
FAT(FAT table), block chanin, index block, …

File system


MM
(Buddy, Slab, …)
+
Naming
=
FS
(FAT, ExtN, …)
N/A
Pointer
Page frame
4KB~
Continuous or non-continuous allocation
Paging(page table)
Name
File(fd)
Block
512B~
Continuous or non-continuous allocation
FAT(FAT table), block chanin, index block, …
Block size vs. performance,
Cylinder group,
Defragmentation,
Layout policy, …

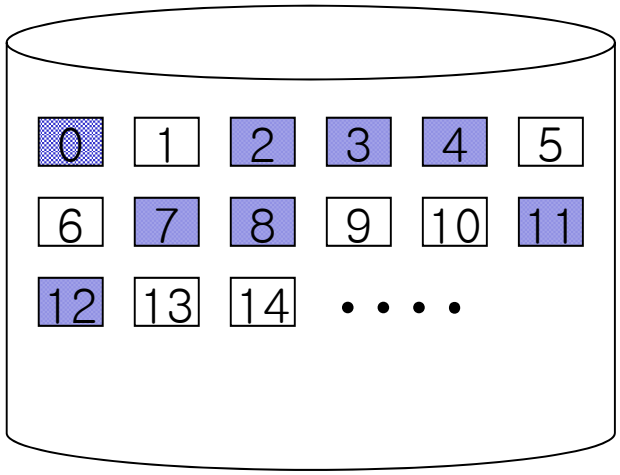
디스크 구조

- 물리적 관점
  - ✓ plotter, arm, head
  - ✓ cylinder, track, sector
  - ✓ 탐색 지간(seek time), 회전 지연 시간(rotational latency), 데이터 전송 시간(transmission time)

- Cylinder group
- Block size vs Performance
- Fragments
- Layout policy
  - ✓ 한 Dir내의 파일
  - ✓ 한 file내의 block

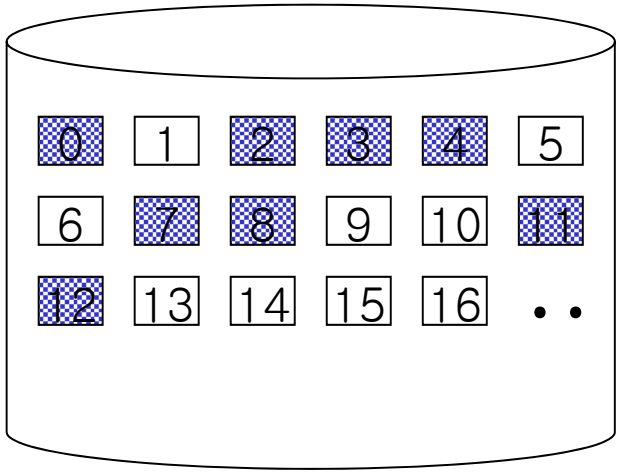
디스크 구조 – 논리적인 관점

- 논리적 관점 (커널이 보는 디스크 구조)
  - ✓ 디스크는 디스크 블록들의 집합 (disk is a collection of disk blocks)
  - ✓ 디스크 블록의 크기는 보통 페이지 프레임의 크기와 같다 (4K, 8K)



디스크 블록 할당 (1/5)

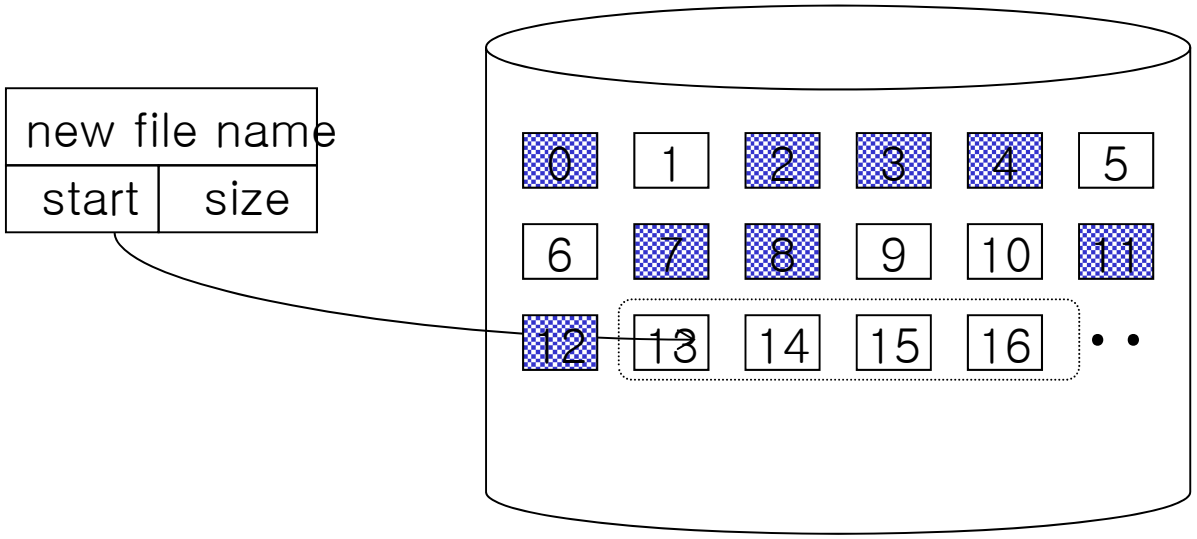
- 시나리오
  - ✓ 새로운 14 K 크기의 파일을 디스크에게 쓰려고 함
  - ✓ 디스크 블록의 크기는 4K로 가정 (따라서 4개의 디스크 블록이 필요)
  - ✓ 아래 그림에서 빗금이 있는 디스크 블록들은 이미 사용 중인 (파일의 데이터를 갖고 있는) 블록이라고 가정
  - ✓ 어떤 디스크 블록들을 할당할 것인가? (disk block allocation)



- 디스크 블록 할당 방법
  - ✓ 연속 할당 방법 (sequential allocation)
  - ✓ 불연속 할당 방법 (non sequential allocation)

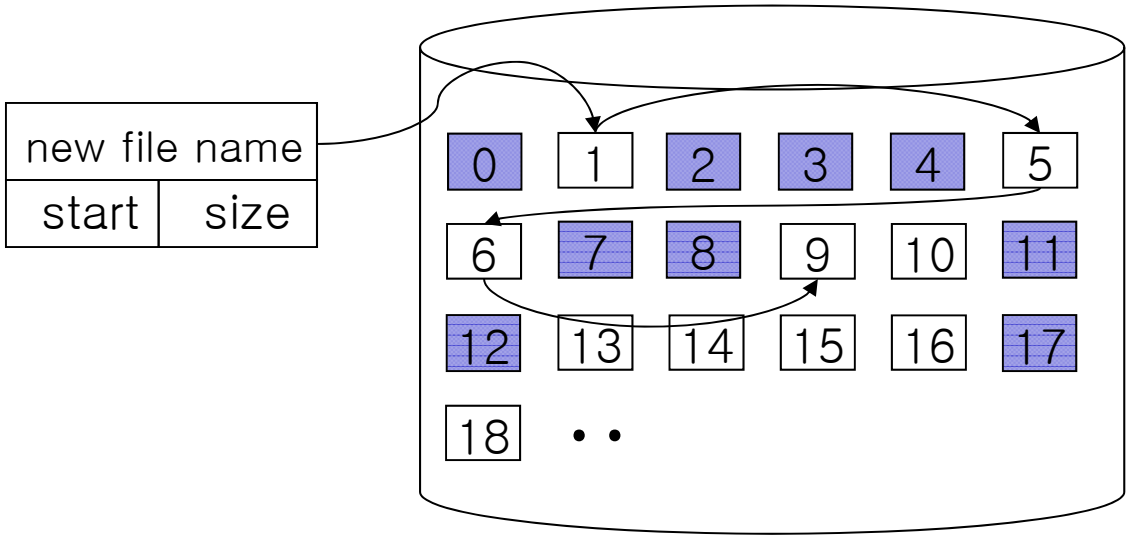
디스크 블록 할당 (2/5)

- 연속 할당 방법



디스크 블록 할당 (3/5)

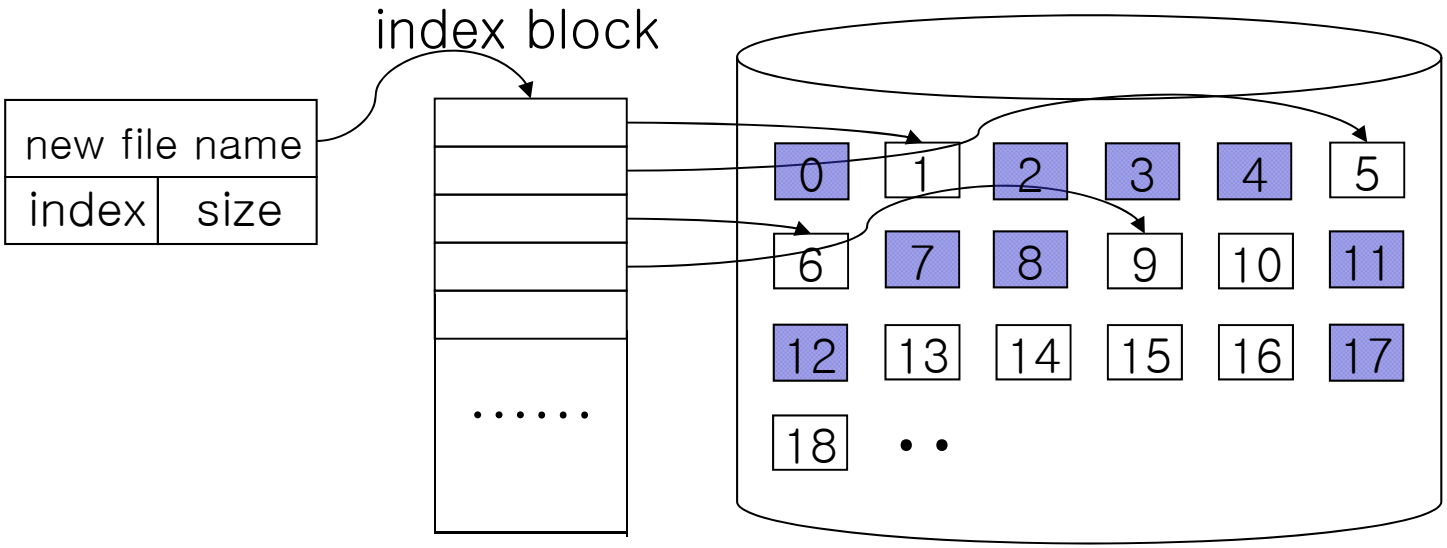
- 불연속 할당 방법 : 블록 체인 (block chain) 방법



✉ 연속 할당과 불연속 할당 방법 간에 장단점을 논해 보자.

디스크 블록 할당 (4/5)

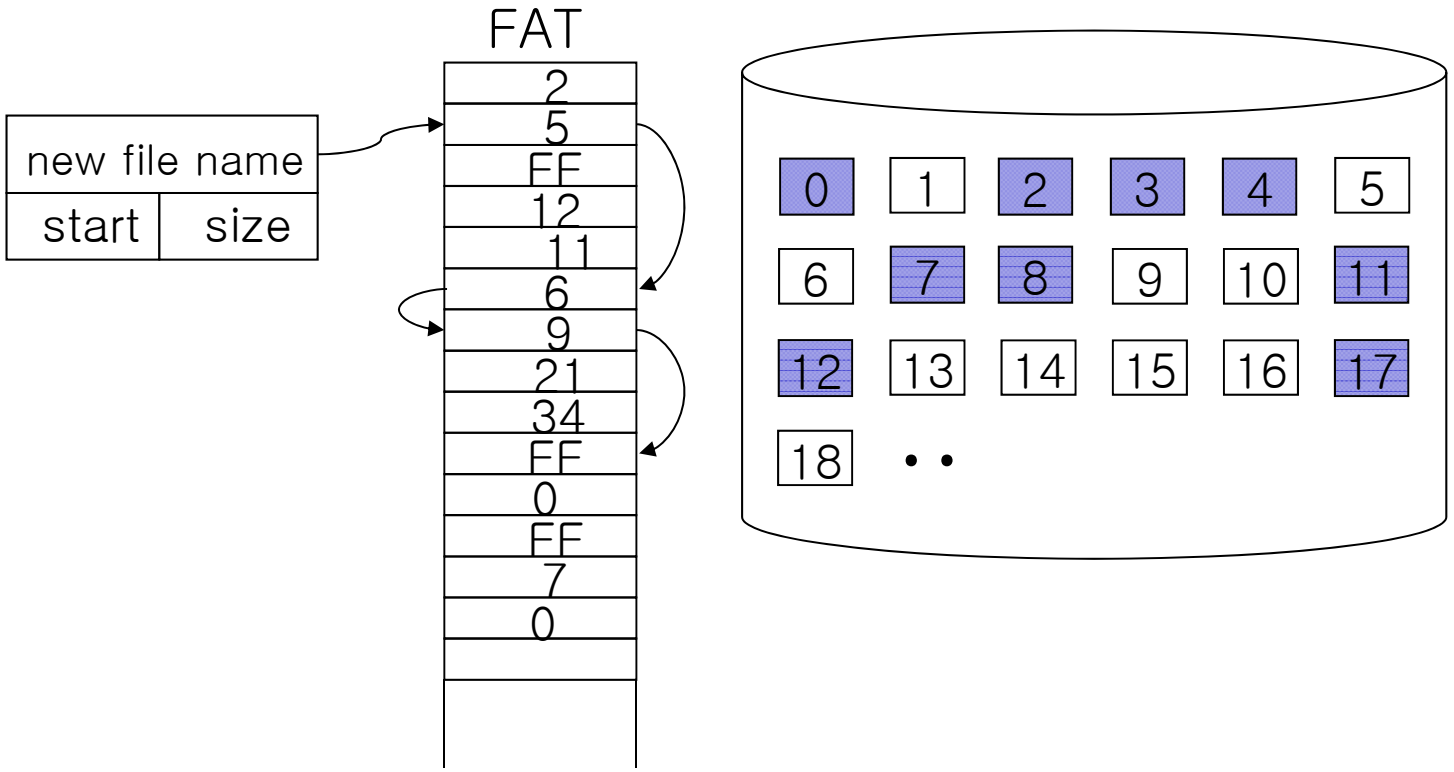
- 불연속 할당 방법 : 인덱스 블록 (index block) 기법



✉ 인덱스 블록이 가득 차면 ?

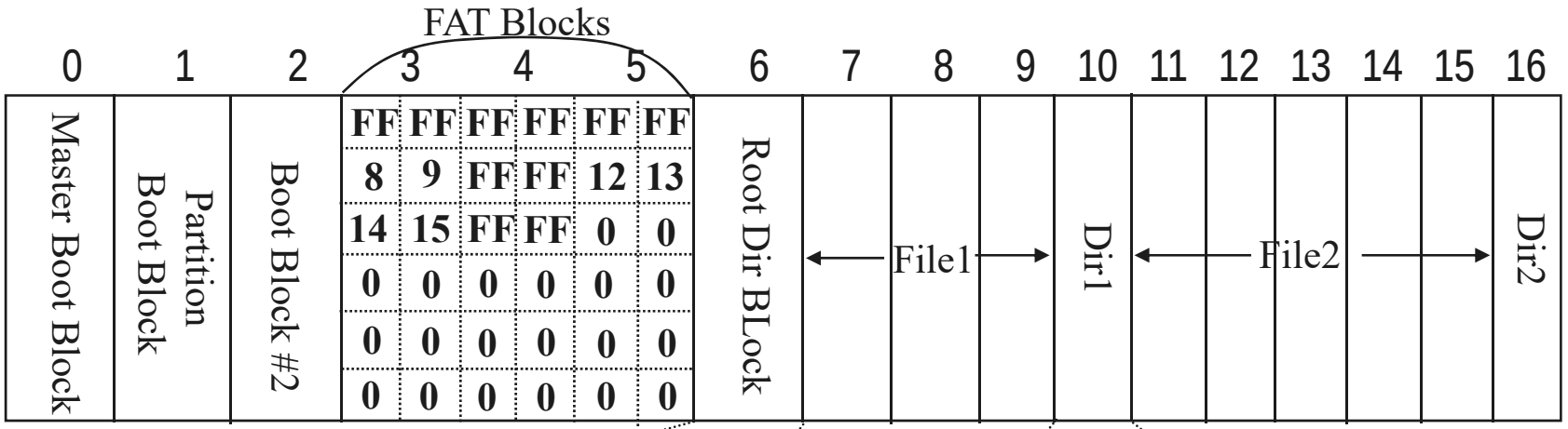
디스크 블록 할당 (5/5)

- 불연속 할당 방법 : FAT (File Allocation Table)



✉ 블록 체인, 인덱스 블록, FAT 간에 장단점을 논해 보자. Linux는 어떤 방법을 사용하고 있는가?

FAT internal



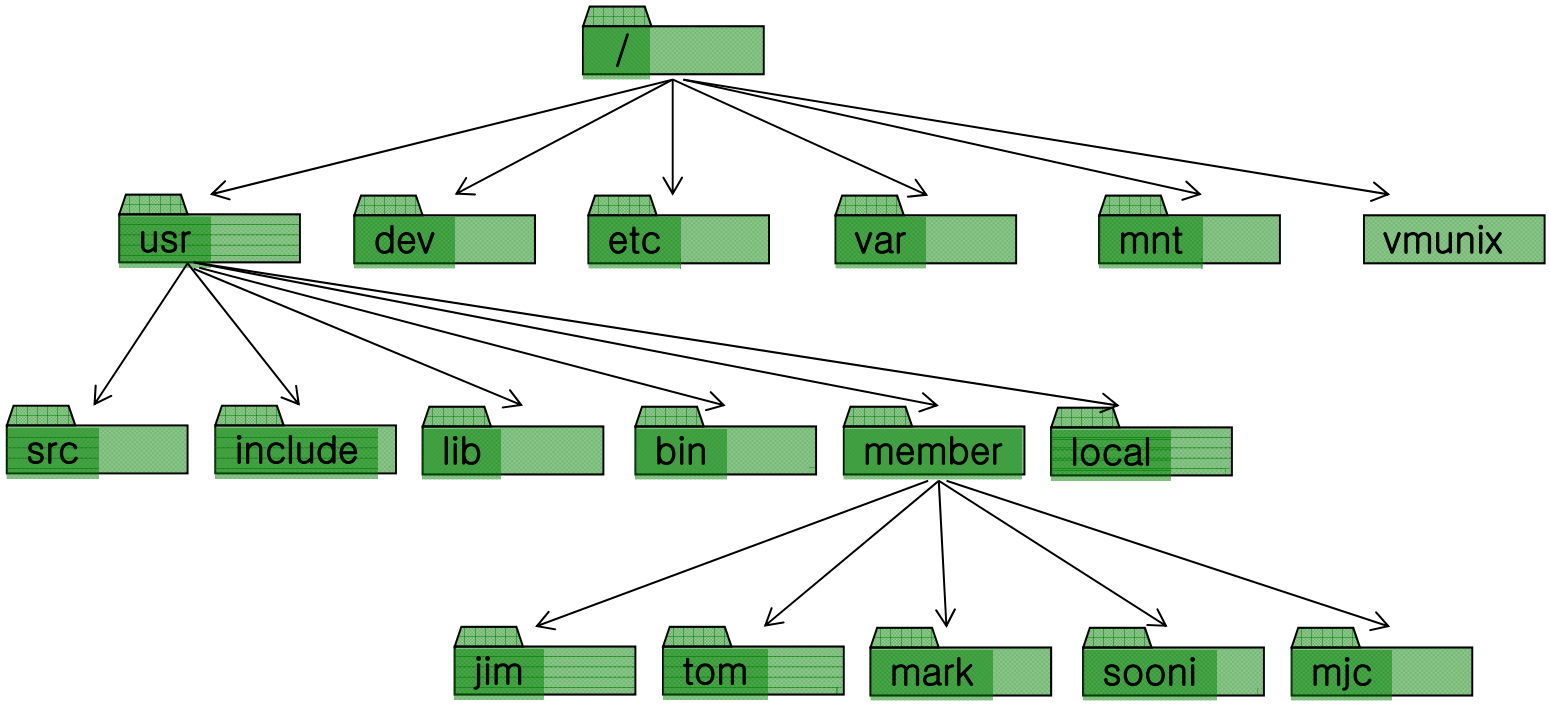
Status	File Name	FirstAddr	Size
F	File1	7	3
D	Dir1	10	1
F	File2	11	5
D	Dir2	16	1

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

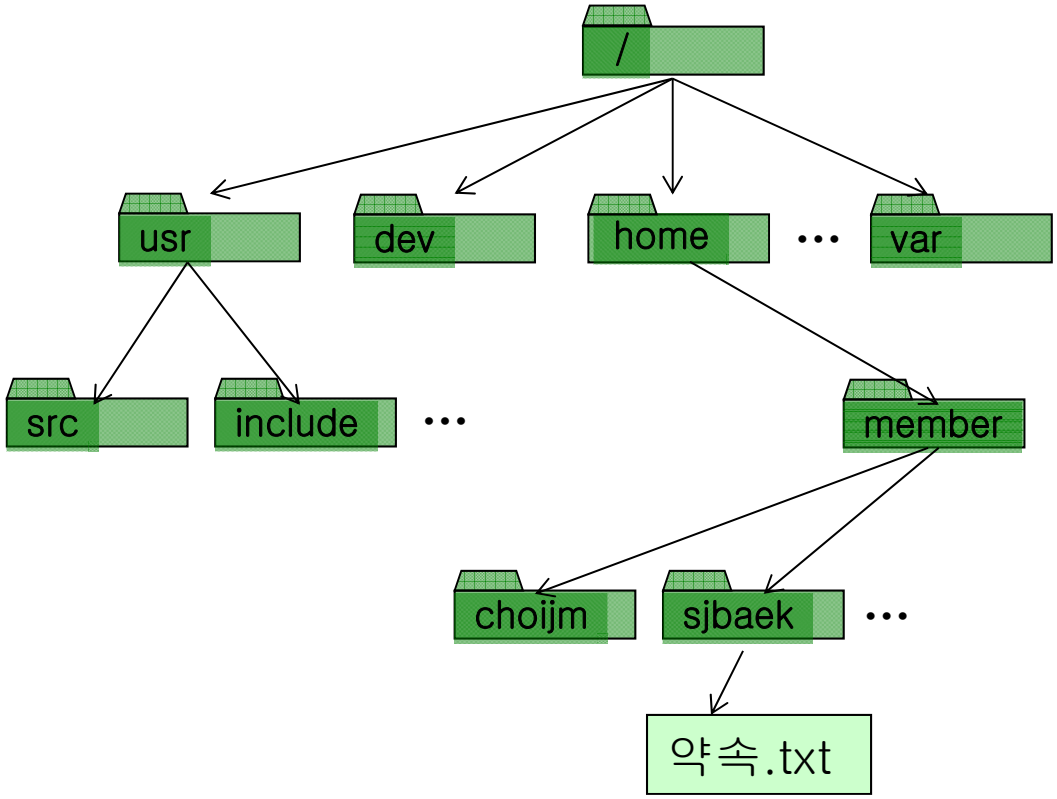
(Source : Dr. dhlee)

디렉토리 구조

- 파일 계층 구조 (hierarchical structure)



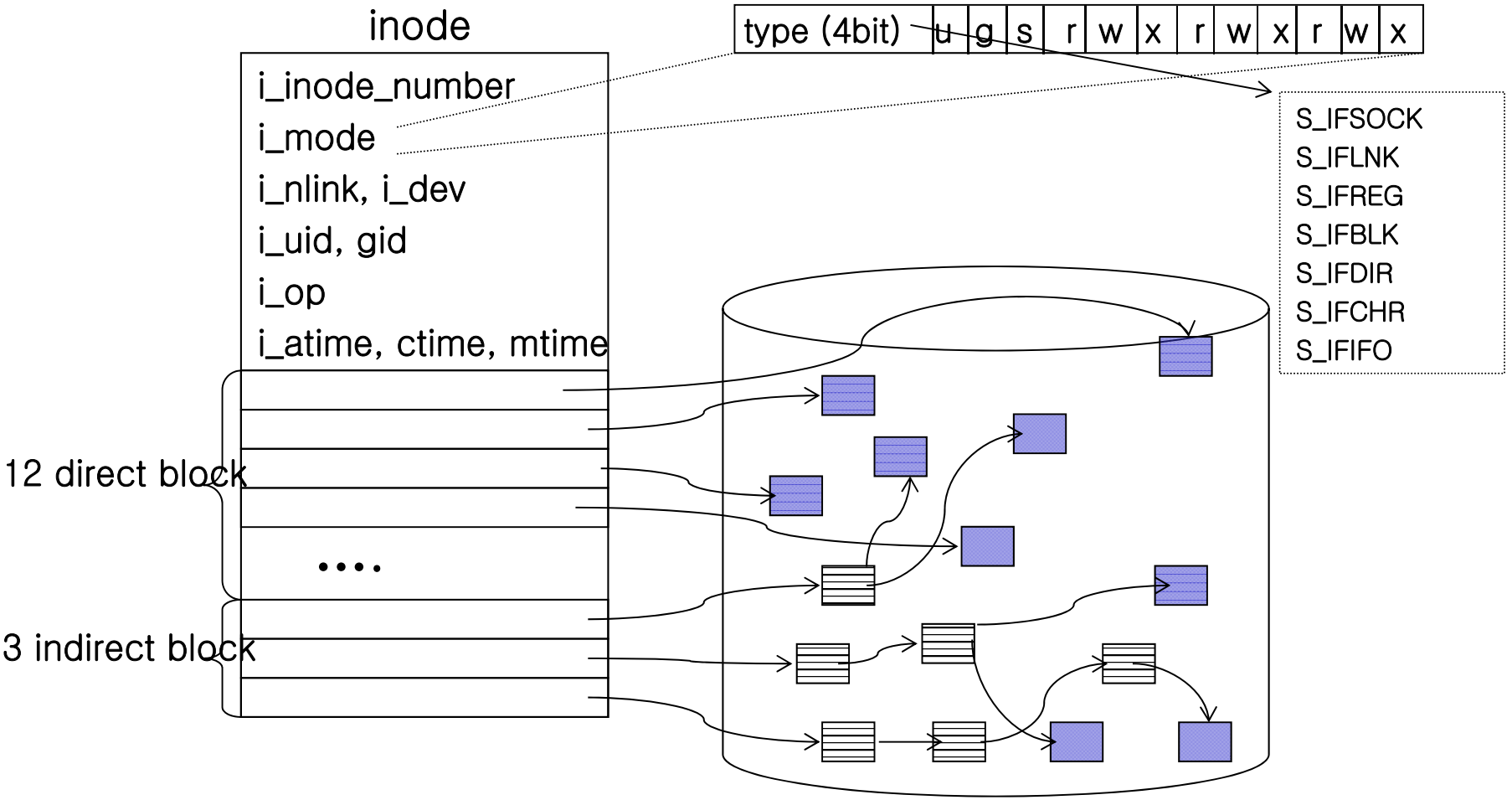
디렉토리 계층구조와 메타데이터의 예(FAT기준)



	Status	File Name	FirstAddr	Size
6	D	usr	7	3
	D	dev	10	1
	D	home	11	5
	D	var	16	11
	...			
11	D	member	20	2
	...			
20	D	choijm	39	3
	D	sjbaek	43	2
	...			
43	F	약속.txt	48	1
	...			

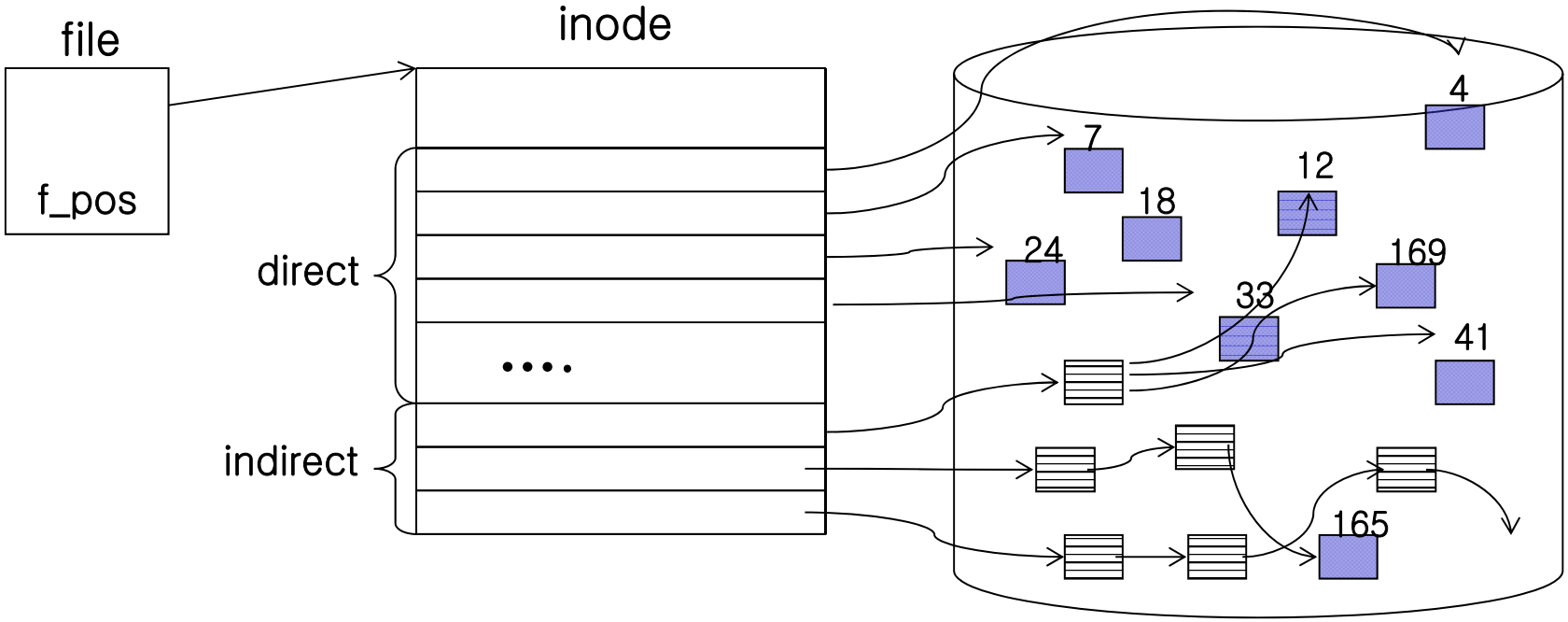
inode 구조 1

- inode /* include/linux/fs.h, ext2_fs_i.h */



inode 구조 2

- 시나리오
  - ✓ 디스크 블록의 크기가 4K로 가정
  - ✓ 파일 옵셋(offset)이 10000 이면 어떤 디스크 블록에 접근하게 되는가? 또한 파일 옵셋이 58000 이라면?



디렉토리 구조 1

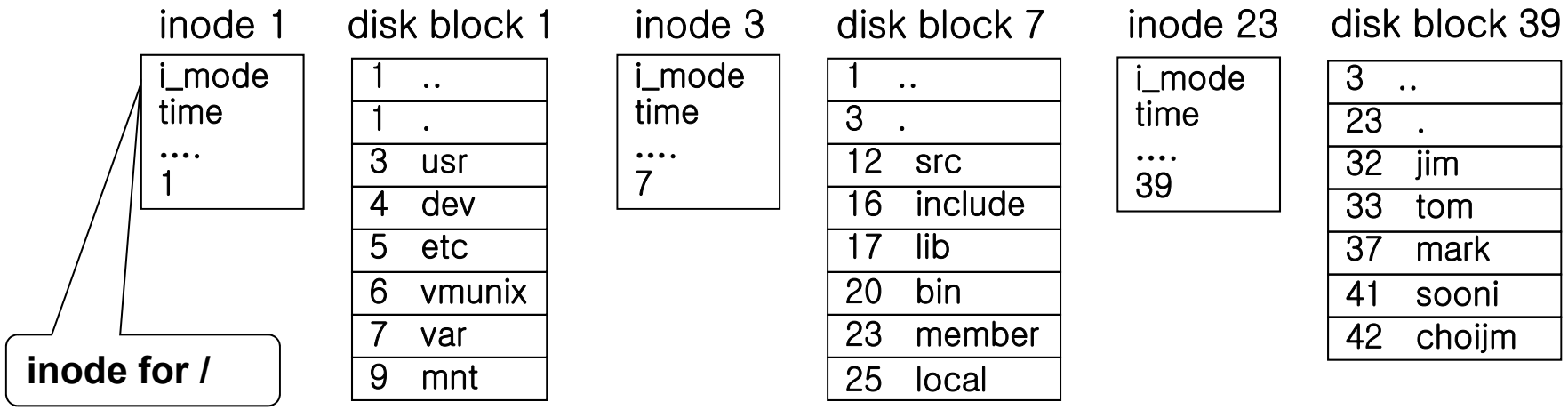
- 디렉터리
  - ✓ 파일 이름과 inode를 연결하는 객체 (namei())
  - ✓ 디렉터리 자체도 파일임
  - ✓ 계층 구조 제공

Linux의 디렉토리 엔트리

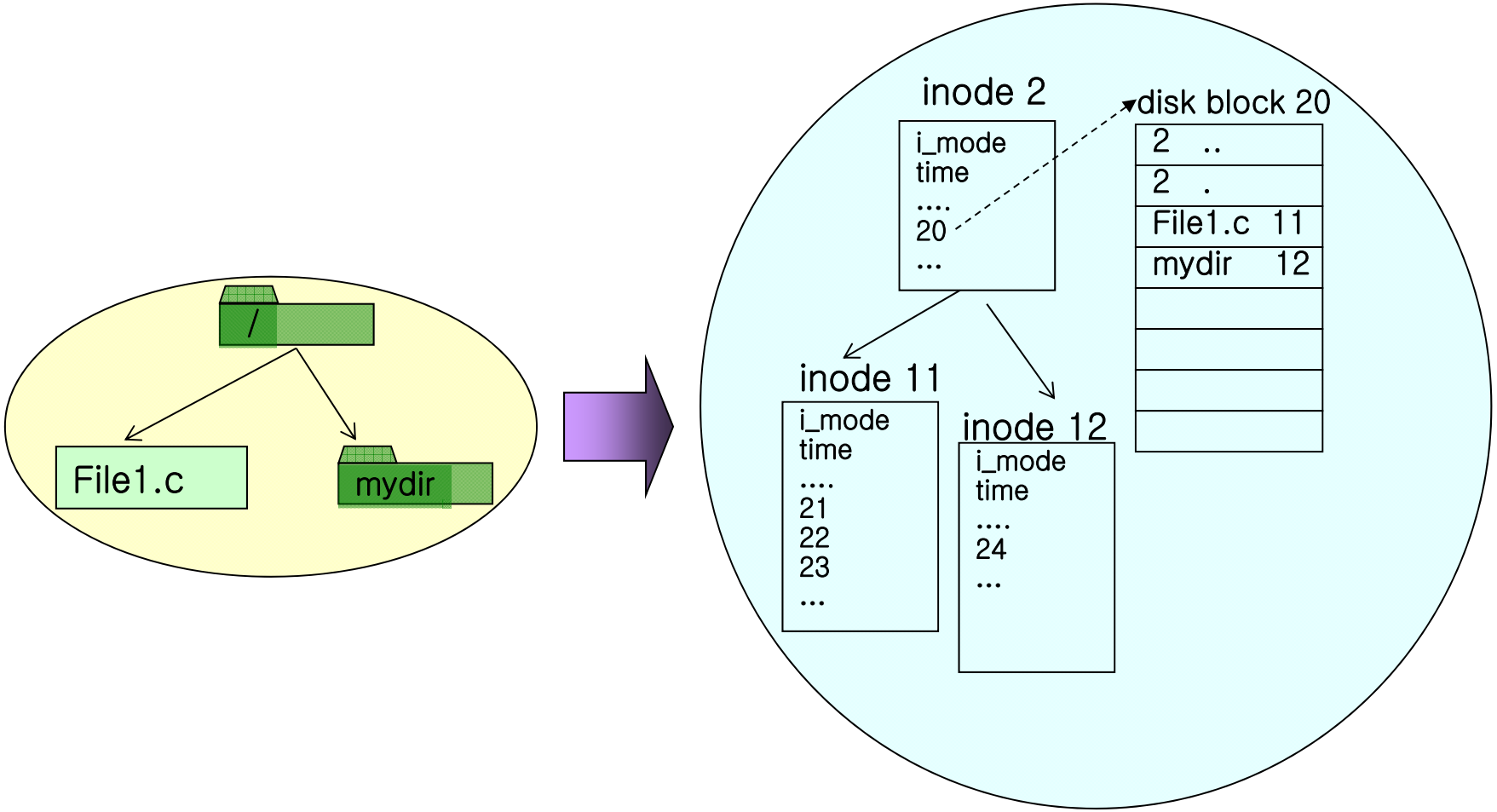
inode number	file name size	file name

directory entry in DOS

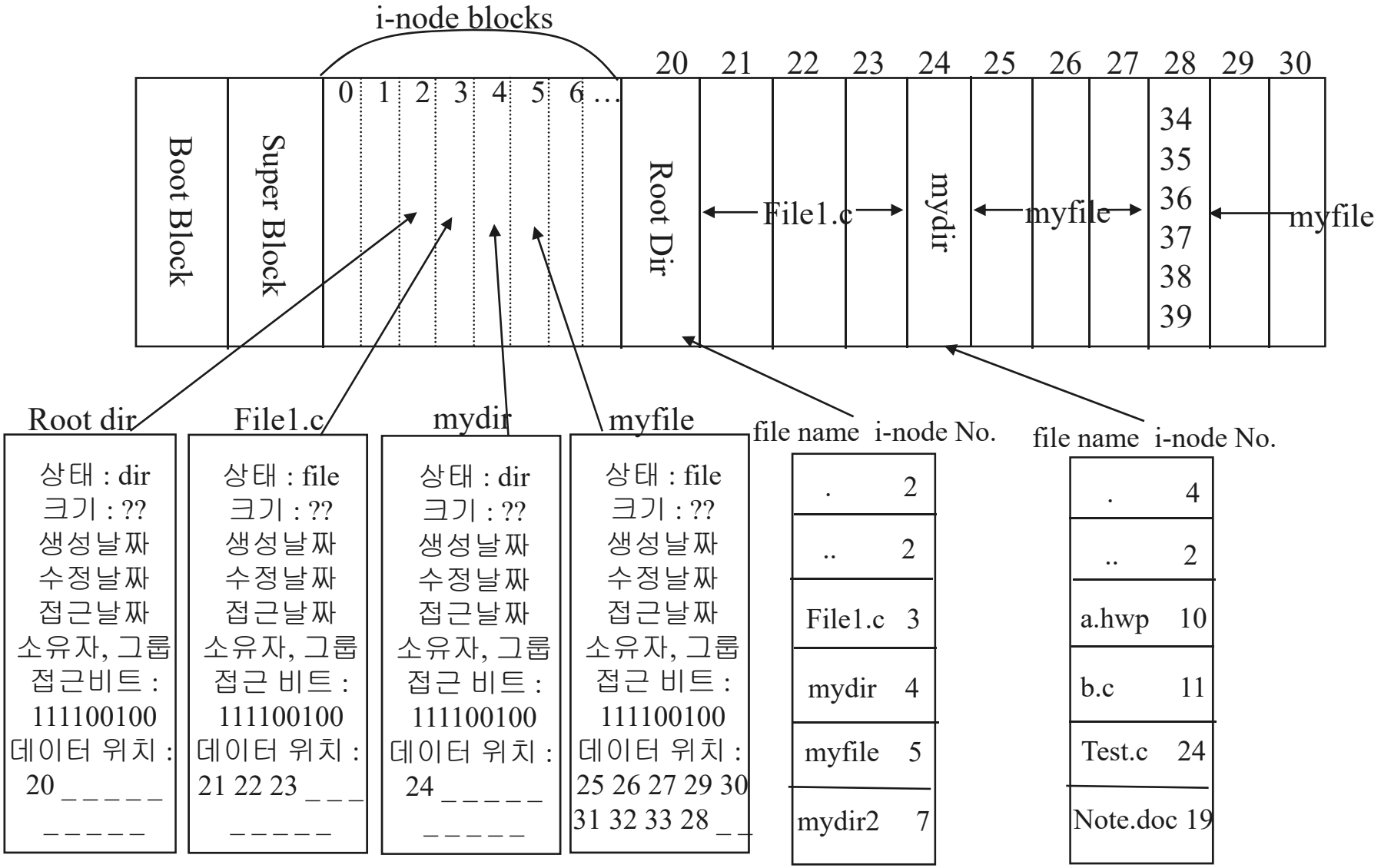
file name	extension	attributes	time	first block number



Ext2의 inode와 파일의 개념적 구조

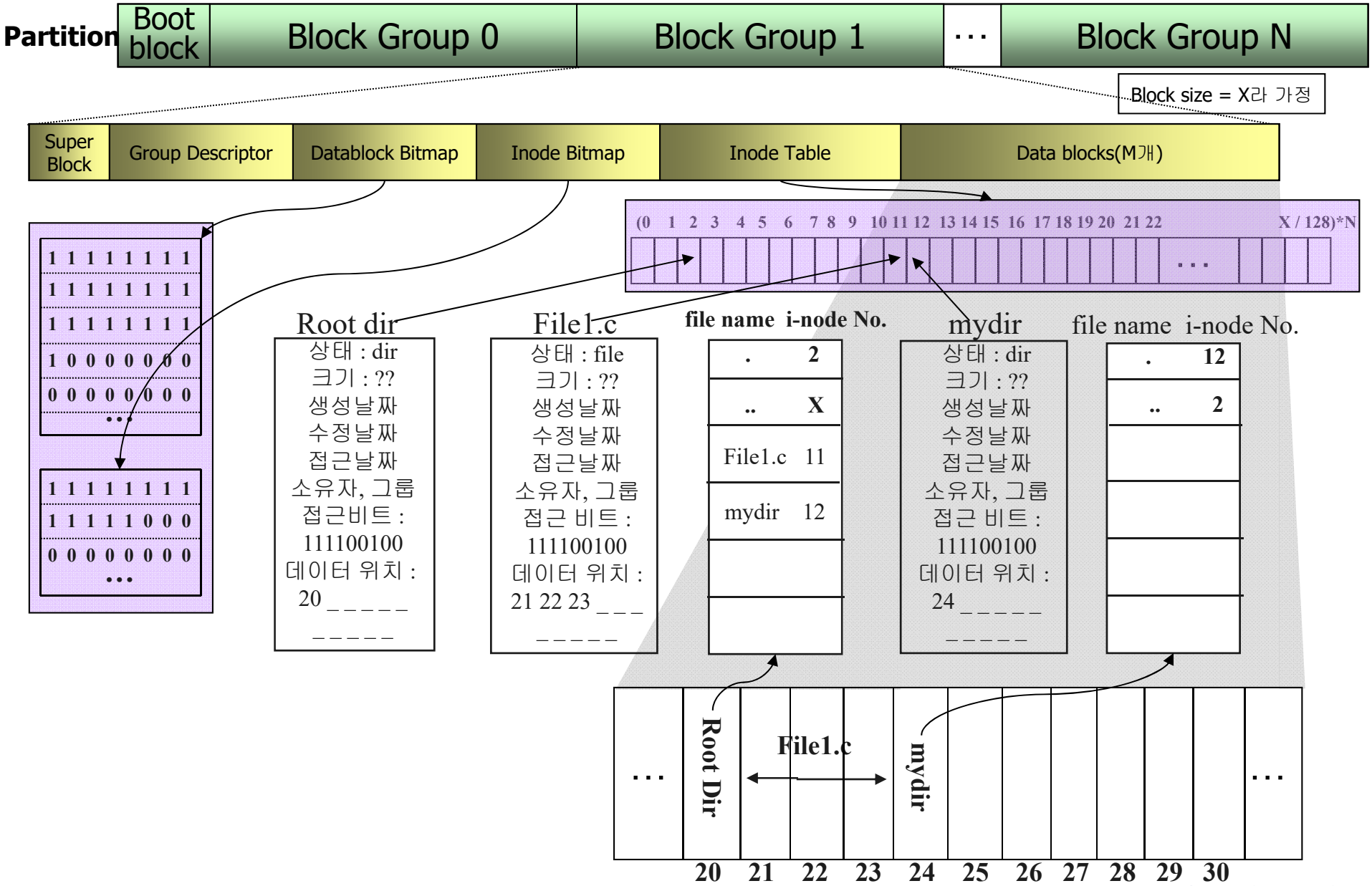


Ext2 internal



(Source : Dr. dhlee)

Ext2 파일시스템의 디스크에서의 레이아웃



Example(1/19)-FS제작

- 만약 여러분이 FS제작자라면 어떤 구조로 FS를 제작할 것인가?
  - ✓ FAT
  - ✓ Ext2
  - ✓ LFS
  - ✓ …

Example(2/19)-FS제작

- 만약 여러분이 FS제작자라면 파일을 관리하기 위한 메타 데이터 자료구조를 어떻게 정의할 것인가?
  - ✓ 이름, 크기, 사이즈, 생성시간, data블록의 위치, …

```
typedef struct {
        unsigned int        blockno;
        unsigned short      offset;
}__attribute__ ((packed)) ET_T;

typedef struct {
        unsigned char       status;
        time_t      date;
        unsigned long       size;
        char        name[FILE_NAME];
        union{
                ET_T        index[5];
                char        data[30];
        }__attribute__ ((packed)) uarea;
        unsigned int        next_blkno;
        char        reserved[4];
        char        attr;
}__attribute__ ((packed)) DIRENTRY_T;

typedef struct {
        unsigned char       status;
        char        reserved[5];
        ET_T        index[9];
        unsigned int        next_blkno;
} __attribute__ ((packed)) IDXDIRENTRY_T;
```

nebFS

Example(3/19)-FS제작

```
struct ext2_dir_entry {
  __le32  inode;        /* Inode number */
  __le16  rec_len;      /* Directory entry length */
  __le16  name_len;     /* Name length */
  char    name[];       /* File name, up to EXT2_NAME_LEN */
};
```

```
struct ext2_inode {
    __u16           i_mode;       /* File mode */
    __u16           i_uid;        /* Low 16 bits of Owner Uid */
    __u32           i_size;       /* Size in bytes */
    __u32           i_atime;      /* Access time */
    __u32           i_ctime;      /* Creation time */
    __u32           i_mtime;      /* Modification time */
    __u32           i_dtime;      /* Deletion Time */
    __u16           i_gid;        /* Low 16 bits of Group Id */
    __u16           i_links_count;              /* Links count */
    __u32           i_blocks;     /* Blocks count */
    __u32           i_flags;      /* File flags */
    union {
        struct {    __u32  l_i_reserved1;   } linux1;
        struct {    __u32  h_i_translator;  } hurd1;
        struct {    __u32  m_i_reserved1; } masix1;
    } osd1;                       /* OS dependent 1 */
    __u32           i_block[EXT2_N_BLOCKS];
                    /* Pointers to blocks */
    __u32           i_generation;/* File version (for NFS) */
    __u32           i_file_acl;   /* File ACL */
    __u32           i_dir_acl;    /* Directory ACL */
    __u32           i_faddr;      /* Fragment address */
    ........
    ......
}
```

ext2

Example(4/19)-FS제작

```
struct msdos_dir_entry {
    __s8    name[8],ext[3];     /* name and extension */
    __u8    attr;               /* attribute bits */
    __u8    lcase;              /* Case for base and extension */
    __u8    ctime_ms;           /* Creation time, milliseconds */
    __u16   ctime;              /* Creation time */
    __u16   cdate;              /* Creation date */
    __u16   adate;              /* Last access date */
    __u16               starthi;/* High 16 bits of cluster in FAT32 */
    __u16   time,date,start;    /* time, date and first cluster */
    __u32   size;               /* file size (in bytes) */
};
```

Msdos FS

Example(5/19)-FS제작

```
typedef struct
{
    yaffs_ObjectType type;
    int   parentObjectId;
    __u16 sum__NoLongerUsed;  // checksum of name. Calc this off the name to prevent inconsistencies
    char  name[YAFFS_MAX_NAME_LENGTH + 1];
    __u32 st_mode;  // protection // Thes following apply to directories, files, symlinks - not hard links
#ifdef CONFIG_YAFFS_WINCE
    __u32 notForWinCE[5];
#else
    __u32 st_uid;   // user ID of owner
    __u32 st_gid;    // group ID of owner
    __u32 st_atime; // time of last access
    __u32 st_mtime; // time of last modification
    __u32 st_ctime; // time of last change
#endif
    int fileSize; // File size  applies to files only
    int  equivalentObjectId; // Equivalent object id applies to hard links only.
    char alias[YAFFS_MAX_ALIAS_LENGTH + 1]; // Alias is for symlinks only.
    __u32 st_rdev;  // device stuff for block and char devices (maj/min)
#ifdef CONFIG_YAFFS_WINCE
    __u32 win_ctime[2];          __u32 win_atime[2]; __u32 win_mtime[2];__u32 roomToGrow[6];
#else
    __u32 rootToGrow[12];
#endif
} yaffs_ObjectHeader;
```

Example(6/19)-FS제작

- 만약 여러분이 FS제작자라면 파일시스템에 어떤 연산을 구현할 것인가? 즉, FS외부로 어떤 interface를 뽑아줄 것인가?
  - ✓ touch ➔ MYFS_file_create()
  - ✓ rm ➔ MYFS_file_delete()
  - ✓ cp ➔ MYFS_file_rename()
  - ✓ cat ➔ MYFS_file_read()
  - ✓ echo >> ➔ MYFS_file_write()
  - ✓ mkdir ➔ MYFS_dir_create()
  - ✓ rmdir ➔ MYFS_dir_delete()
  - ✓ ls ➔ MYFS_dir_read()
  - ✓ …

Example(7/19)-FS제작

- 다양한 유형의 파일 지원
  - ✓ 정규 파일, 디렉토리, 특수 장치 파일, 링크, 파이프, 소켓, ..
- 파일 시스템 관련 시스템 호출
  - ✓ open, close
  - ✓ read, write
  - ✓ lseek
  - ✓ dup
  - ✓ link
  - ✓ pipe, mkfifo
  - ✓ mkdir, readdir
  - ✓ mknod
  - ✓ stat
  - ✓ mount
  - ✓ sync, fsck
- POSIX interface ?

Example(8/19)-FS제작

```
static struct file_operations neb_dir_operations = {
        readdir:    neb_readdir,
};
static struct inode_operations
neb_dir_inode_operations = {
        create:     neb_create,
        lookup:     neb_lookup,
        unlink:     neb_unlink,
        mkdir:      neb_mkdir,
        rmdir:      neb_rmdir,
        rename:     neb_rename,
        permission:  neb_permission,
        symlink:    neb_symlink,
        mknod:      neb_mknod,
};
static struct file_operations neb_file_operations = {
        read:       neb_file_read,
        write:      neb_file_write,
        mmap:       generic_file_mmap,
};
static struct inode_operations
neb_file_inode_operations = {
        lookup:     neb_lookup,
        permission:  neb_permission,
        truncate:   neb_truncate,
};
```

nebFS

Example(9/19)-FS제작


```
struct file_operations ext2_dir_operations = {
        read:       generic_read_dir,
        readdir:    ext2_readdir,
        ioctl:      ext2_ioctl,
        fsync:      ext2_sync_file,
};
struct file_operations ext2_file_operations = {
        llseek:     generic_file_llseek,
        read:       generic_file_read,
        write:      generic_file_write,
        ioctl:      ext2_ioctl,
        mmap:       generic_file_mmap,
        open:       generic_file_open,
        release:    ext2_release_file,
        fsync:      ext2_sync_file,
};
static struct super_operations ext2_sops = {
        read_inode:         ext2_read_inode,
        write_inode:        ext2_write_inode,
        put_inode:          ext2_put_inode,
        delete_inode:       ext2_delete_inode,
        put_super:          ext2_put_super,
        write_super:        ext2_write_super,
        statfs:             ext2_statfs,
        remount_fs:         ext2_remount,
};
```

ext2

Example(10/19)-FS제작


```
struct inode_operations
msdos_dir_inode_operations = {
        create:   msdos_create,
        lookup:   msdos_lookup,
        unlink:   msdos_unlink,
        mkdir:    msdos_mkdir,
        rmdir:    msdos_rmdir,
        rename:   msdos_rename,
        setattr:  fat_notify_change,
};
```

Msdos FS

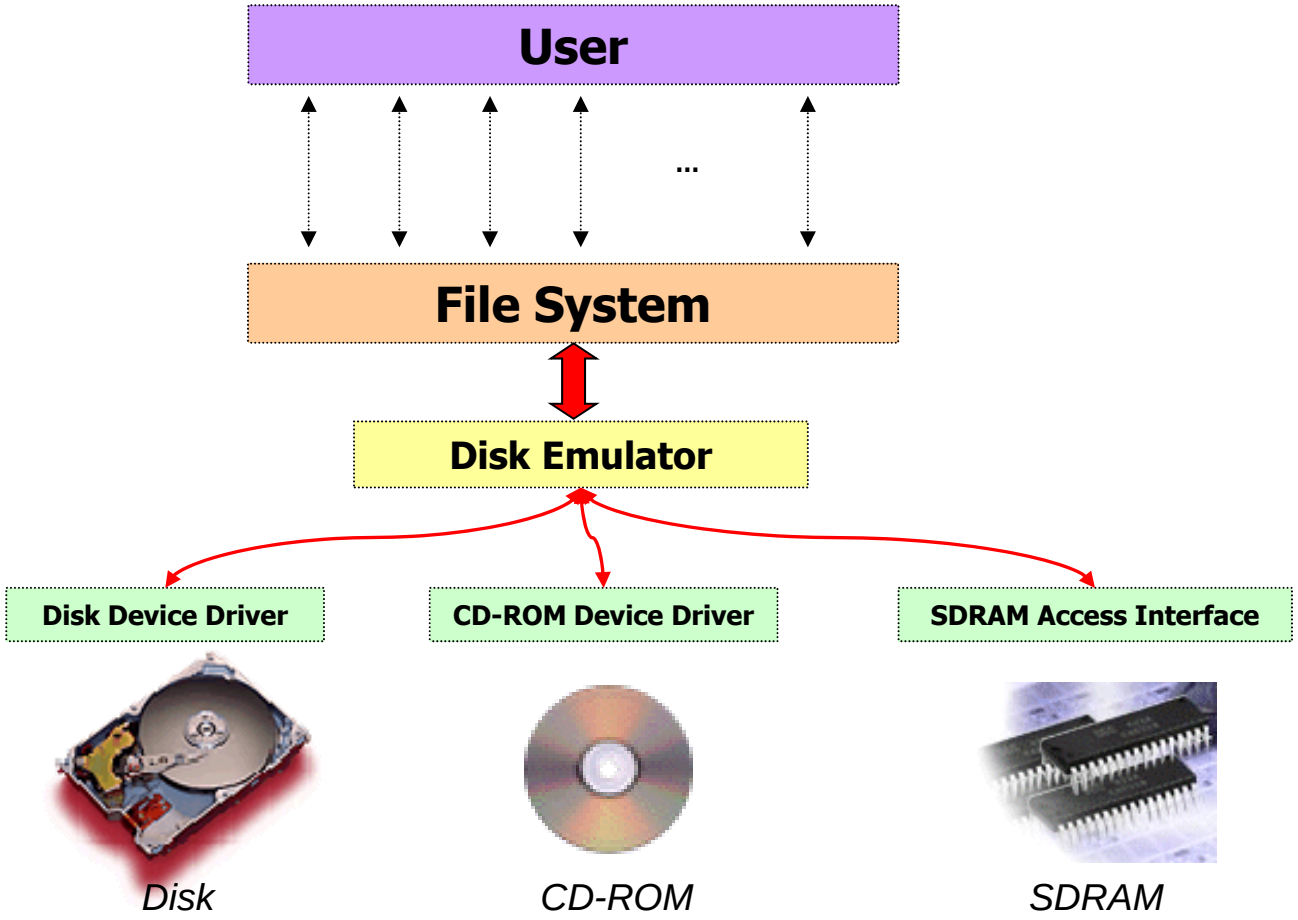
Example(11/19)-FS제작

```
static struct address_space_operations
yaffs_file_address_operations = {
        readpage:   yaffs_readpage,
        prepare_write:      yaffs_prepare_write,
        commit_write:       yaffs_commit_write
};
static struct file_operations yaffs_file_operations = {
#ifdef CONFIG_YAFFS_USE_GENERIC_RW
        read:       generic_file_read,
        write:      generic_file_write,
#else
        read:       yaffs_file_read,
        write:      yaffs_file_write,
#endif
        mmap:       generic_file_mmap,
        flush:      yaffs_file_flush,
        fsync:      yaffs_sync_object,
};
static struct inode_operations
yaffs_file_inode_operations = {
        setattr:    yaffs_setattr,
};
struct inode_operations
yaffs_symlink_inode_operations =
{
        readlink:   yaffs_readlink,
        follow_link:yaffs_follow_link,
        setattr:    yaffs_setattr
};
```

```
static struct inode_operations
yaffs_dir_inode_operations = {
        create:     yaffs_create,
        lookup:     yaffs_lookup,
        link:       yaffs_link,
        unlink:     yaffs_unlink,
        symlink:    yaffs_symlink,
        mkdir:      yaffs_mkdir,
        rmdir:      yaffs_unlink,
        mknod:      yaffs_mknod,
        rename:     yaffs_rename,
        setattr:    yaffs_setattr,
};
static struct file_operations yaffs_dir_operations = {
        read:       generic_read_dir,
        readdir:    yaffs_readdir,
        fsync:      yaffs_sync_object,
};
static struct super_operations yaffs_super_ops = {
        statfs:     yaffs_statfs,
        read_inode: yaffs_read_inode,
        put_inode:  yaffs_put_inode,
        put_super:  yaffs_put_super,
        delete_inode: yaffs_delete_inode,
        clear_inode: yaffs_clear_inode,
};
```

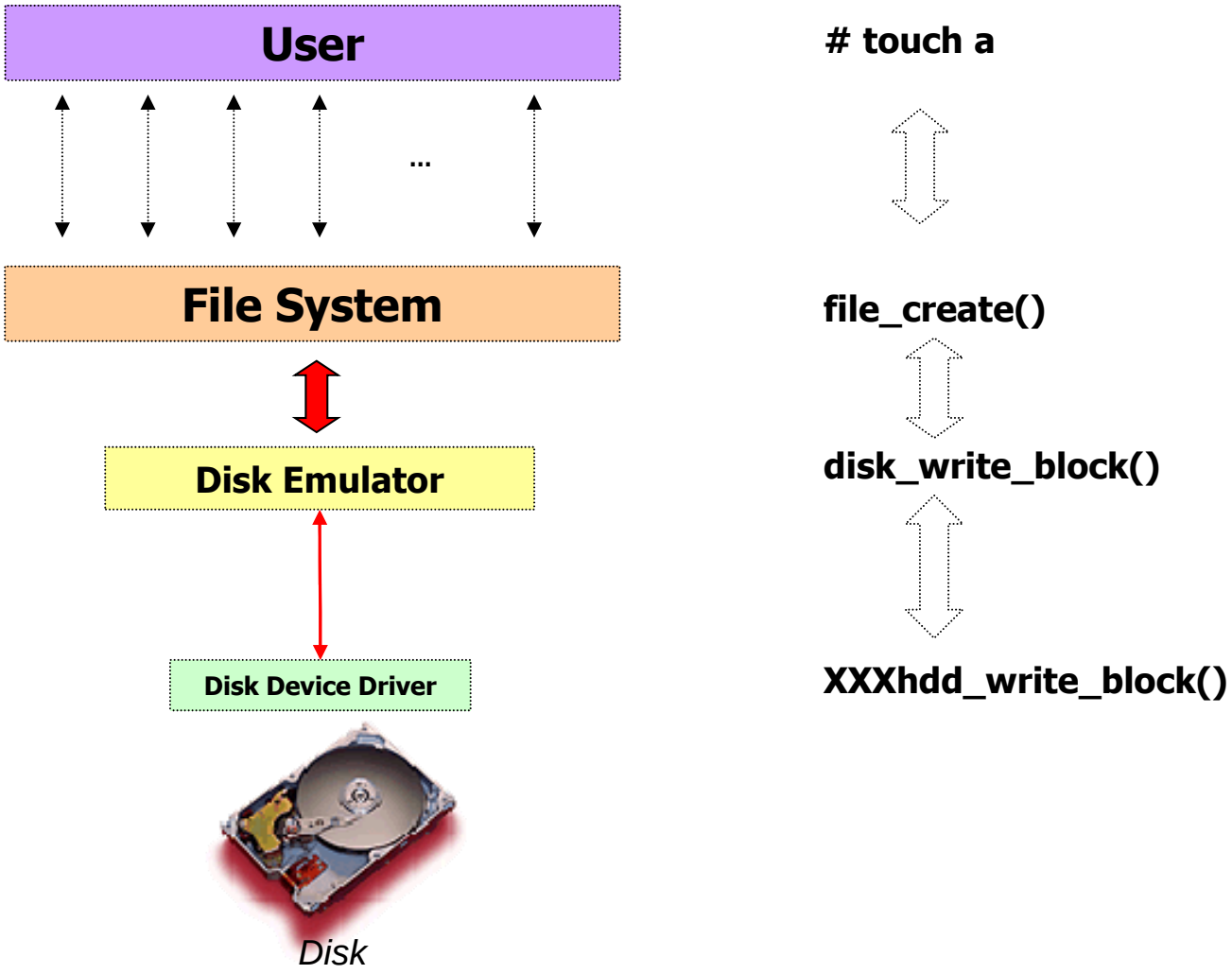
Example(12/19)-FS제작

- 만약 여러분이 FS제작자라면 만든 FS가 특정 device위에서만 돌도록 만드시겠습니까?



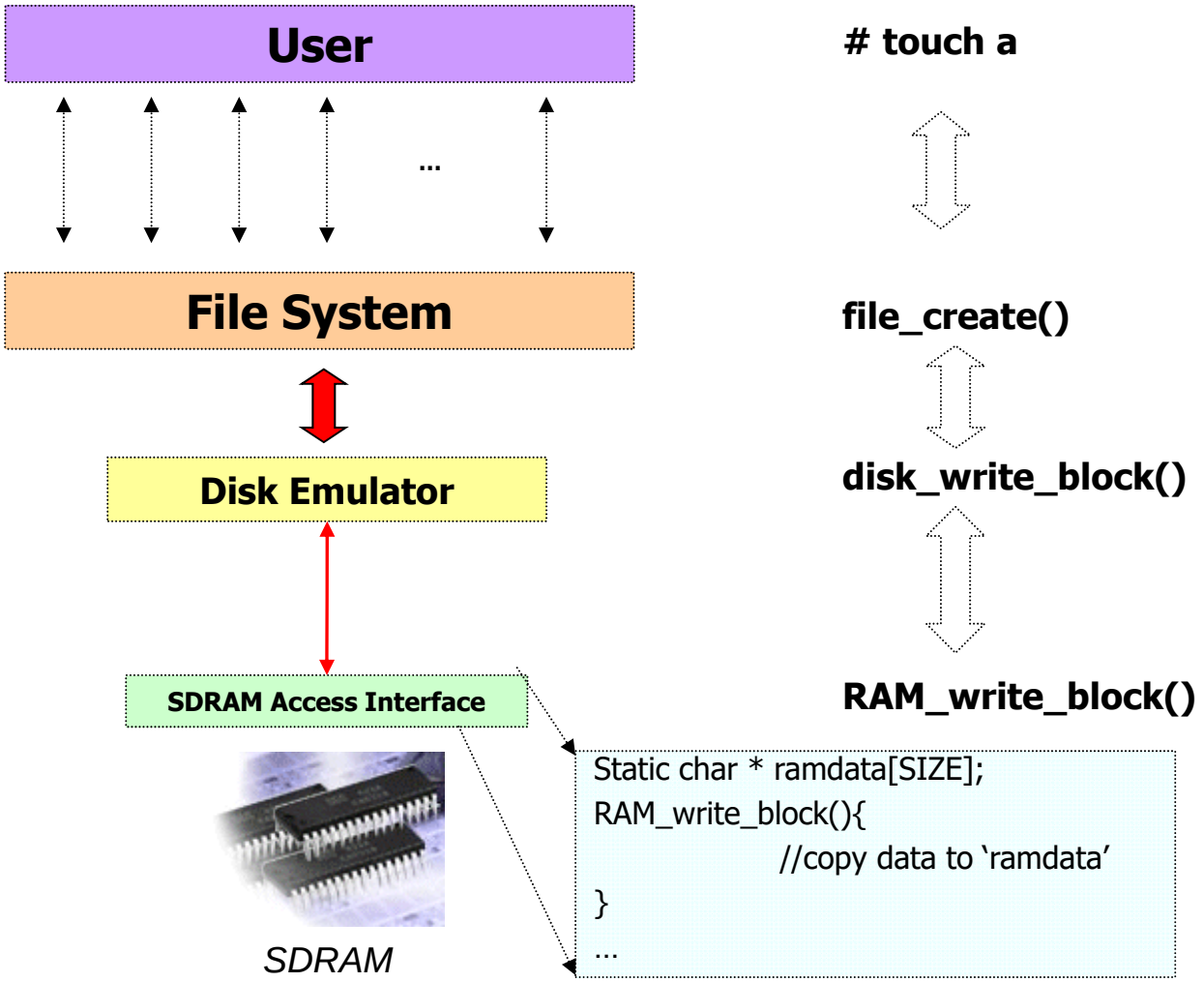
Example(13/19)-FS제작

✓ 만약 Hard Disk 위 에서 돌리고 싶다면?



Example(14/19)-FS제작

✓ 만약 RAM 상에서 돌리고 싶다면?



Example(15/19)-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(16/19)-FS제작

- create, read, write …. 의 함수는?

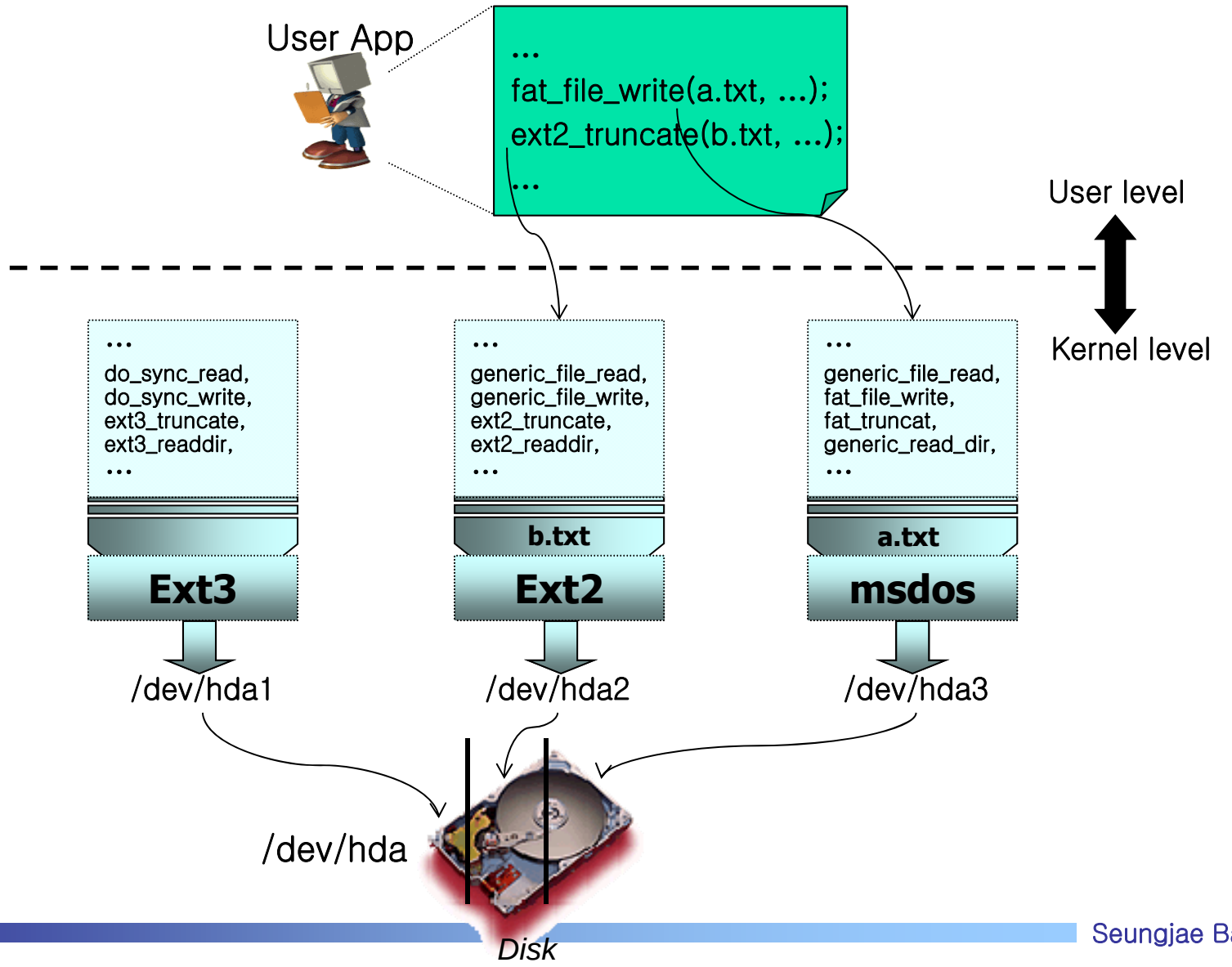
- cp ./a.txt /b.txt

```
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);
                …
        }
        …
}
```

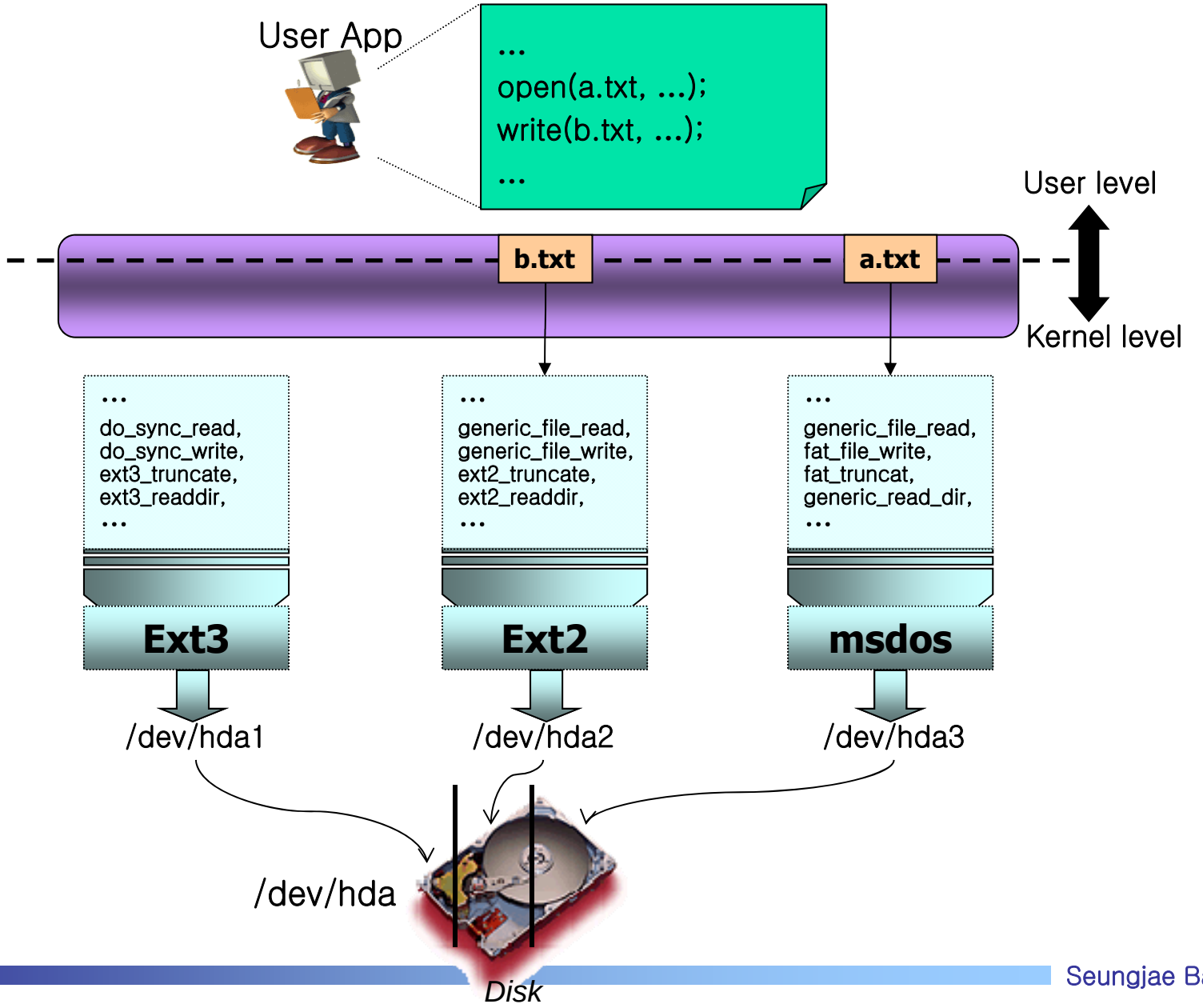
☞정말 **cp** 프로그램을 이렇게 만들어 줘야 할까**?**

☞이렇게 **FS** 종속적이지 않도록 하려면**?**

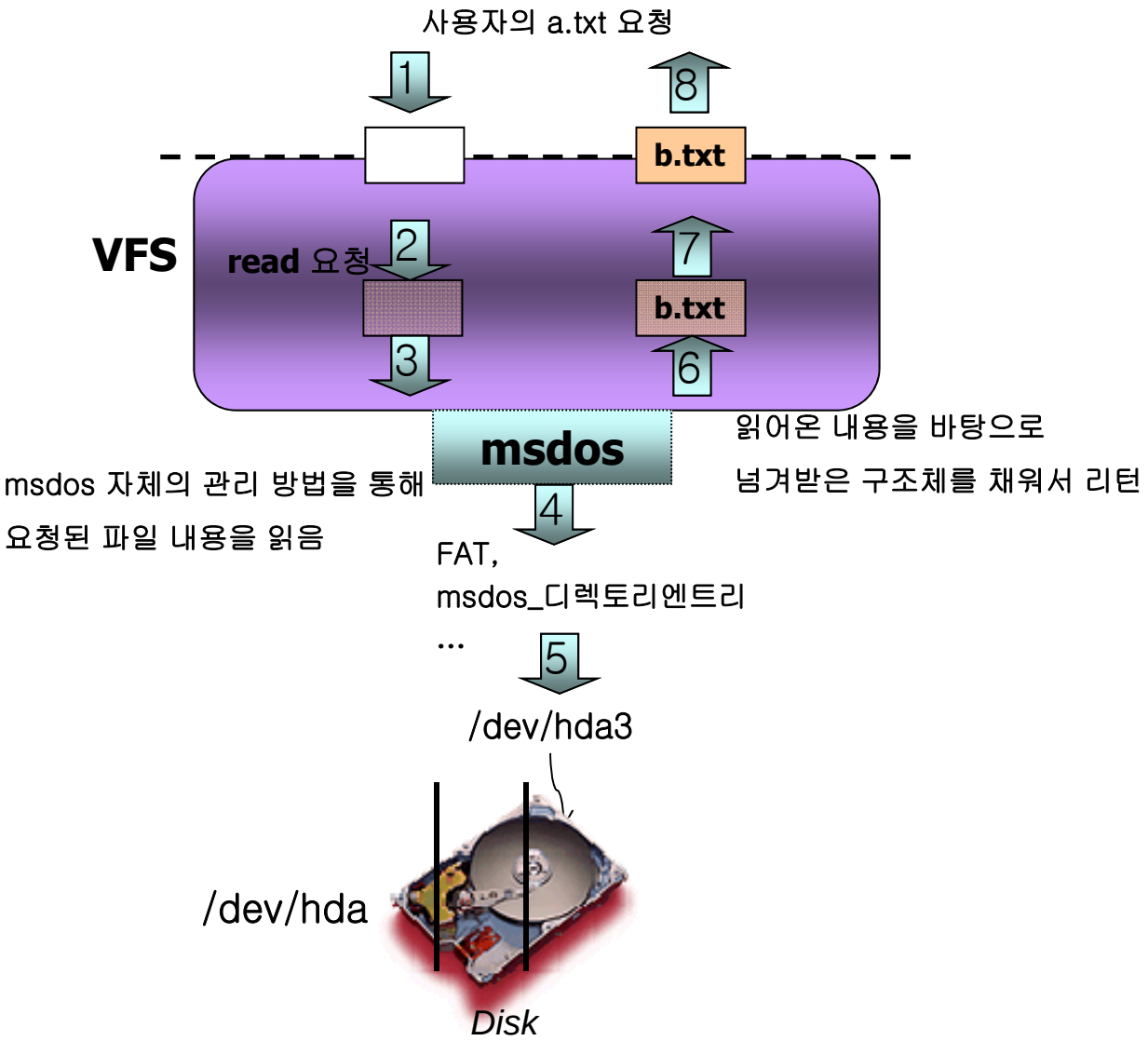
다양한 파일시스템 지원의 문제점



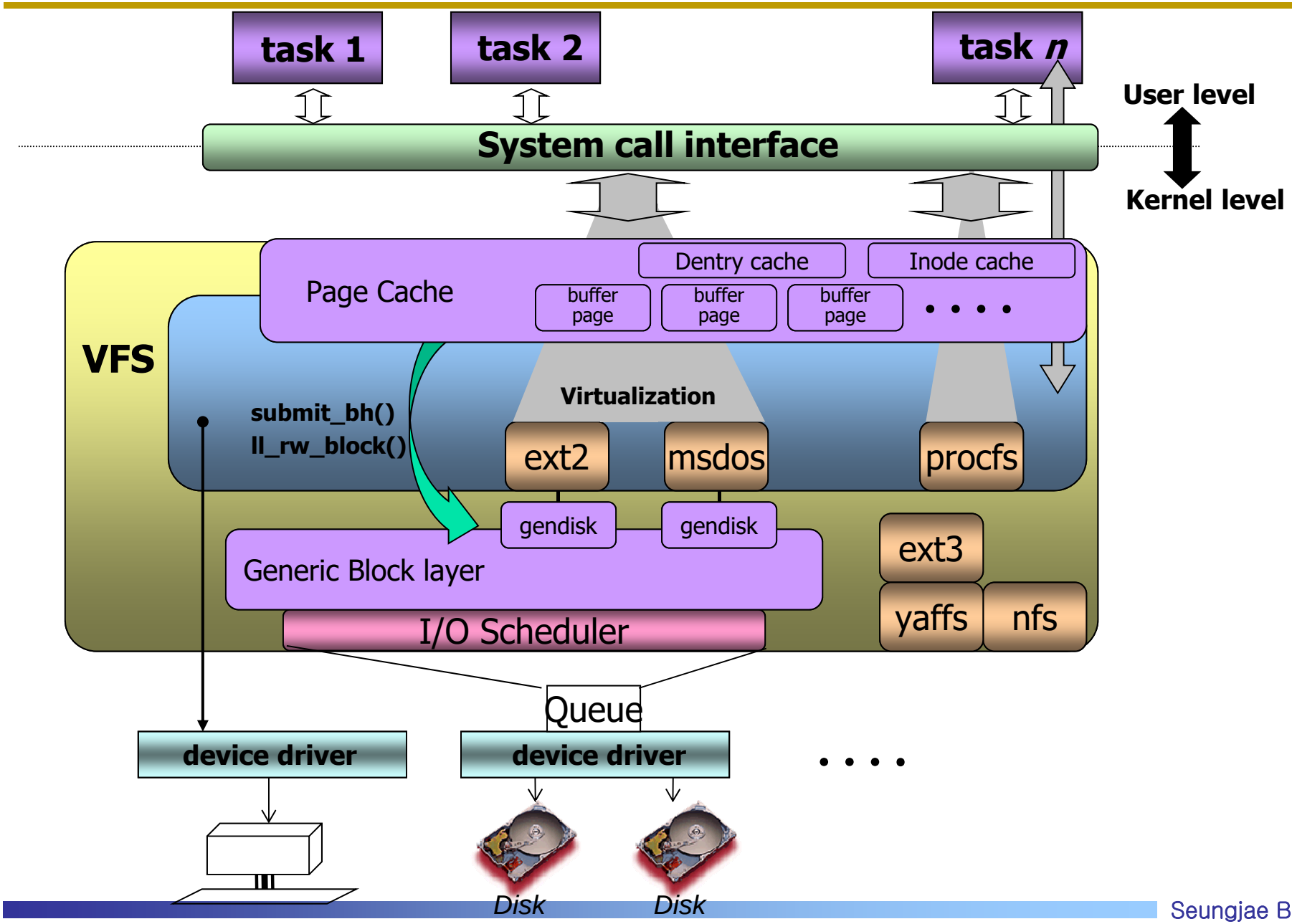
가상적인(Virtual) 계층의 도입



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



VFS와 네 가지 객체

- **VFS의** 객체 유형
  - ✓ 슈퍼블록 객체
    - 마운트 된 FS을 나타내는 정보 저장
    - (디스크의) 파일 시스템 제어 블록(filesystem control block)에 대응
  - ✓ 아이노드 객체
    - 특정 파일을 나타내는 정보 저장
    - (디스크의) 파일 제어 블록(file control block)에 대응
    - inode 번호로 FS 내에서 유일하게 식별됨
  - ✓ 파일 객체
    - 프로세스와 연관되어 Open되어있는 파일과 관련된 정보를 저장
    - **각 process 가 file에 접근하는 동안 커널 메모리에만 존재**
  - ✓ 디엔트리 객체
    - 디렉토리 항목과 이에 대응하는 파일의 연결에 대한 정보를 저장
    - 독자적 방법으로 디스크에 저장함
- 디엔트리 캐시
  - ✓ 가장 최근에 사용한 디엔트리 객체를 캐시

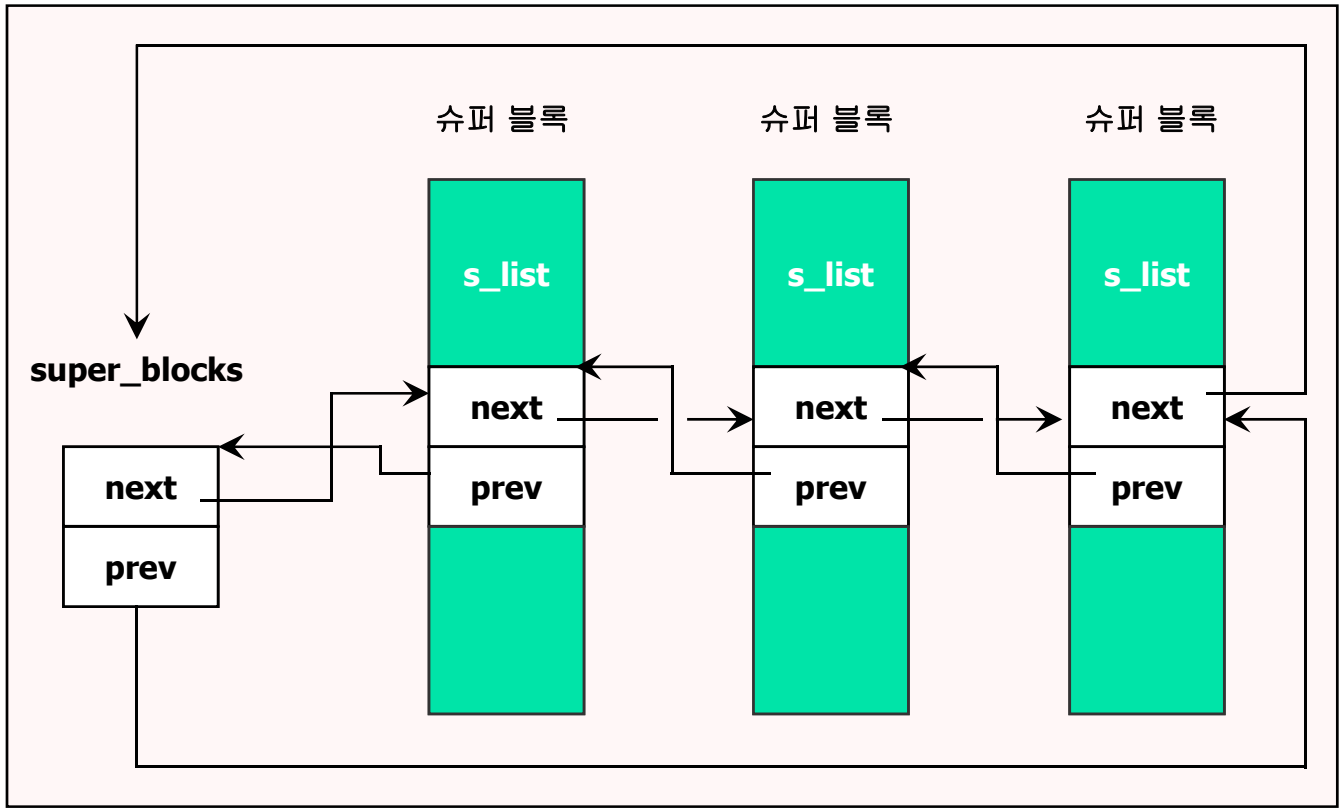
Super block 객체

- VFS 자료 구조
  - ✓ 슈퍼블록 객체(struct super_block, <linux/fs.>)

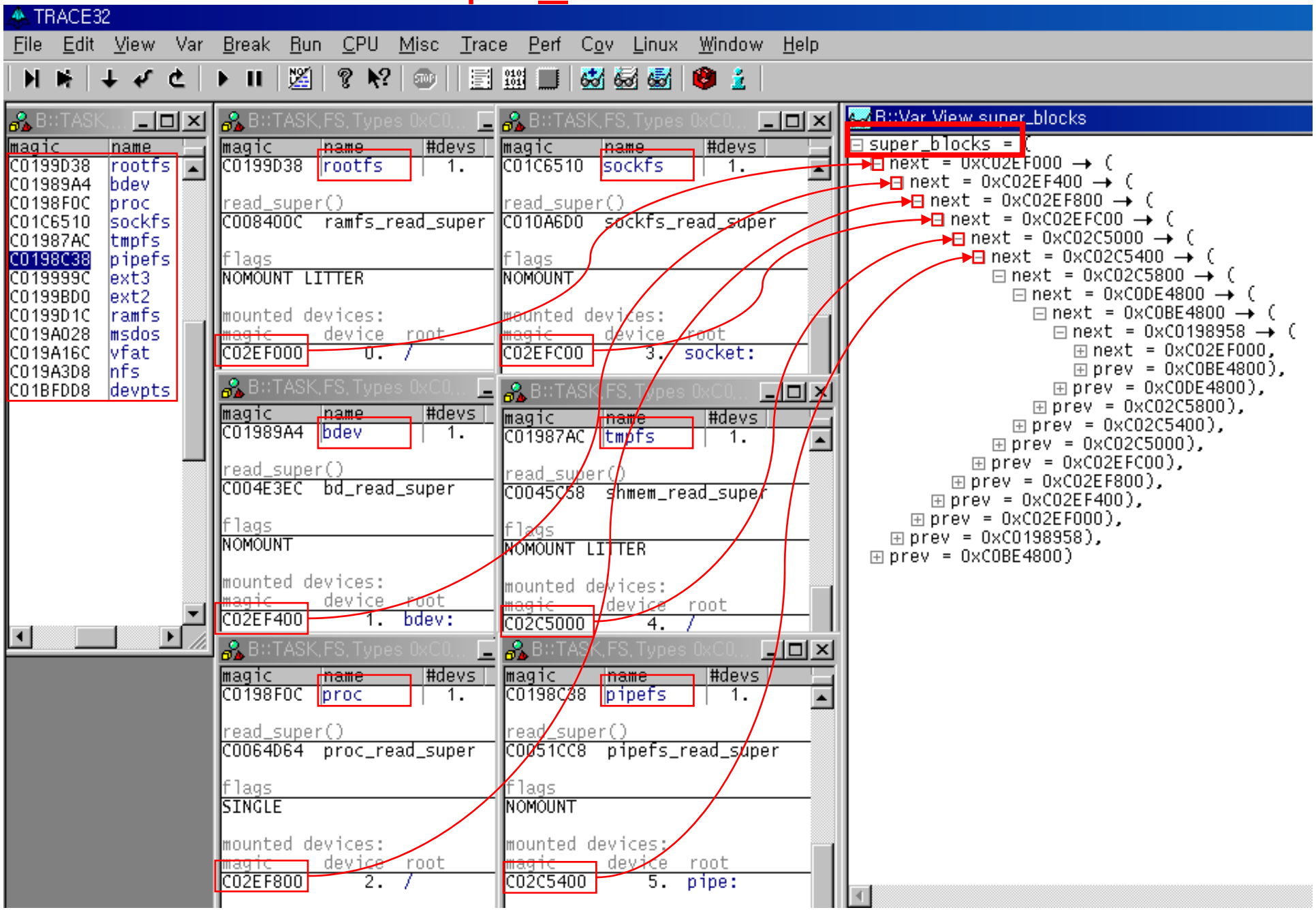
Type	Field	Description
struct list_head	s_list	Pointers for superblock list
kdev_t	s_dev	Device identifier
unsigned long	s_blocksize	Block size in bytes
unsigned char	s_blocksize_bits	Block size in number of bits
unsigned char	s_dirt	Modified (dirty) flag
unsigned long long	s_maxbytes	Maximum size of the files
struct file_system_type *	s_type	Filesystem type
struct super_operations *	s_op	Superblock methods
struct dquot_operations *	dq_op	Disk quota methods
unsigned long	s_flags	Mount flags
unsigned long	s_magic	Filesystem magic number
struct dentry *	s_root	Dentry object of mount directory
struct rw_semaphore	s_umount	Semaphore used for unmounting
struct semaphore	s_lock	Superblock semaphore
int	s_count	Reference counter
atomic_t	s_active	Secondary reference counter
struct list_head	s_dirty	List of modified inodes
struct list_head	s_locked_inodes	List of inodes involved in I/O
struct list_head	s_files	List of file objects assigned to the superblock
struct block_device *	s_bdev	Pointer to the block device driver descriptor
struct list_head	s_instances	Pointers for a list of superblock objects of a given filesystem type
struct quota_mount_options	s_dquot	Options for disk quota
union	u	Specific filesystem information

Super block 객체 관리

- 모든 superblock object는 circular doubly linked list로 연결
- list field
  - ✓ 처음과 끝은 super_blocks 변수의 `s_list`의 `next`와 `prev`에 저장



super_blocks list 확인



Super block 객체와 super operations

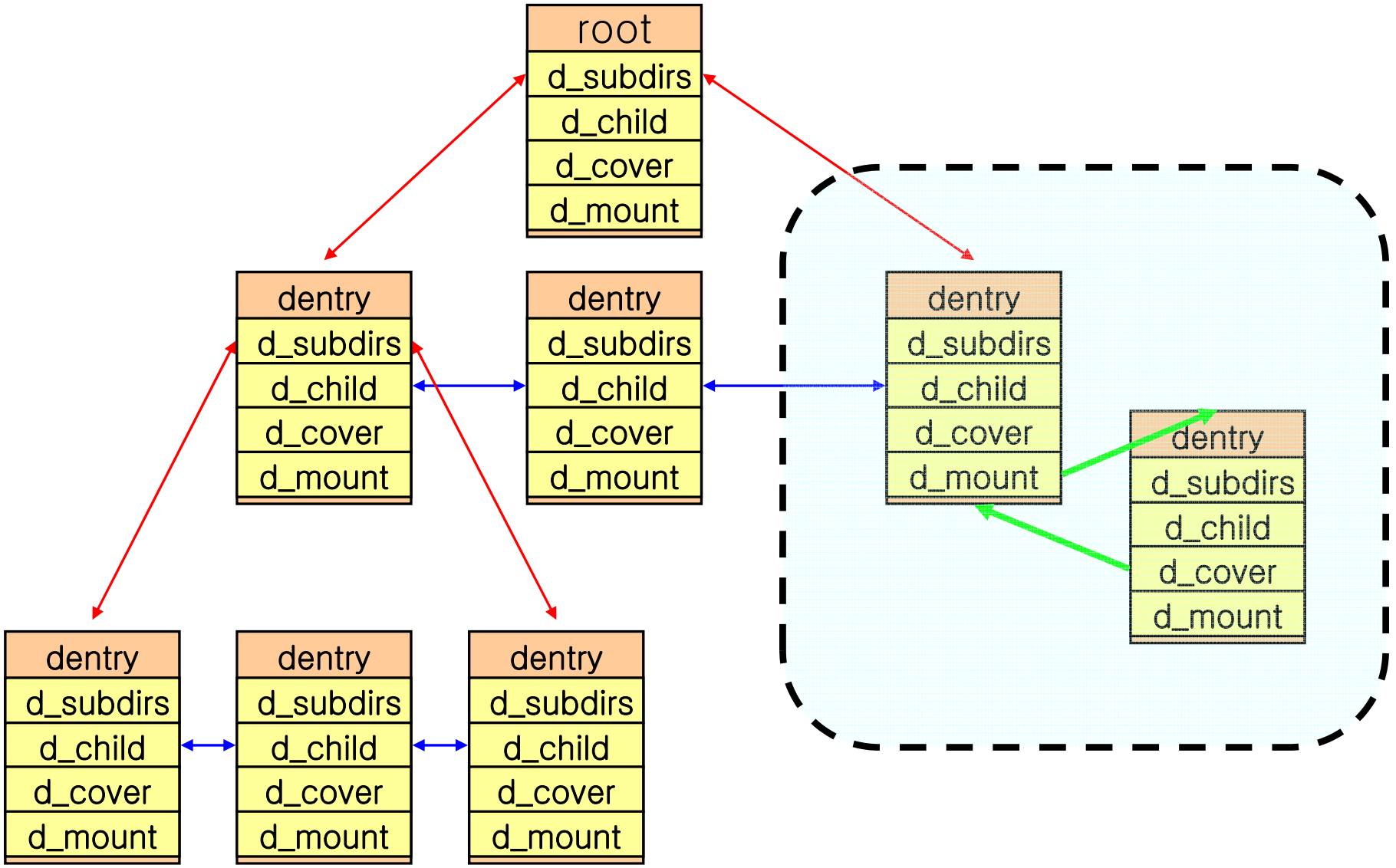

```
struct super_operations {
        struct inode *(*alloc_inode)(struct super_block *sb);
        void (*destroy_inode)(struct inode *);

        void (*read_inode) (struct inode *);

        void (*read_inode2) (struct inode *, void *) ;
        void (*dirty_inode) (struct inode *);
        void (*write_inode) (struct inode *, int);
        void (*put_inode) (struct inode *);
        void (*delete_inode) (struct inode *);
        void (*put_super) (struct super_block *);
        void (*write_super) (struct super_block *);
        int (*sync_fs) (struct super_block *);
        void (*write_super_lockfs) (struct super_block *);
        void (*unlockfs) (struct super_block *);
        int (*statfs) (struct super_block *, struct statfs *);
        int (*remount_fs) (struct super_block *, int *, char *);
        void (*clear_inode) (struct inode *);
        void (*umount_begin) (struct super_block *);

        struct dentry * (*fh_to_dentry)(struct super_block *sb, __u32 *fh, int len, int fhtype, int parent);
        int (*dentry_to_fh)(struct dentry *, __u32 *fh, int *lenp, int need_parent);
        int (*show_options)(struct seq_file *, struct vfsmount *);
};
```

VFS' mounted FS system structure



Example – 간단한 FS실습(1/2)


```
[root@embeddedDell root]# dd if=/dev/zero of=./img bs=1k count=8192
8192+0개의 레코드를 입력하였습니다
8192+0개의 레코드를 출력하였습니다
[root@embeddedDell root]# mke2fs ./img
mke2fs 1.32 (09-Nov-2002)
./img is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
2048 inodes, 8192 blocks
409 blocks (4.99%) reserved for the super user
First data block=1
1 block group
8192 blocks per group, 8192 fragments per group
2048 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 21 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@embeddedDell root]#
```

Example – 간단한 FS실습(2/2)


```
[root@embeddedDell root]# mkdir mnt
[root@embeddedDell root]# mount -t ext2 -o loop ./img mnt
[root@embeddedDell root]#
[root@embeddedDell root]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda2              38G  7.5G   29G  21% /
/dev/hda1              99M   13M   82M  13% /boot
/dev/hdb2              73G   65G  4.4G  94% /home
none                  251M     0  251M   0% /dev/shm
/root/img             7.8M   13K  7.4M   1% /root/mnt
[root@embeddedDell root]# cd mnt
[root@embeddedDell mnt]# ls -alh
합계 17K
drwxr-xr-x    3 root     root         1.0K  5월 30 12:06 .
drwxr-x---   51 root     root         4.0K  5월 30 12:07 ..
drwx------    2 root     root          12K  5월 30 12:06 lost+found
[root@embeddedDell mnt]# touch a
[root@embeddedDell mnt]# ls
a  lost+found
[root@embeddedDell mnt]#
```

- Block Size를 2048 혹은, 4096으로 만들 수 있는가?

Inode 객체

✓ 아이노드 객체(struct inode <linux/fs.h>)

Type	Field	Description
struct list_head	i_hash	Pointers for the hash list
struct list_head	i_list	Pointers for the inode list
struct list_head	i_dentry	Pointers for the dentry list
struct list_head	i_dirty_buffers	Pointers for the modified buffers list
struct list_head	i_dirty_data_buffers	Pointers for the modified data buffers list
unsigned long	i_ino	inode number
unsigned int	i_count	Usage counter
kdev_t	i_dev	Device identifier
umode_t	i_mode	File type and access rights
nlink_t	i_nlink	Number of hard links
uid_t	i_uid	Owner identifier
gid_t	i_gid	Group identifier
kdev_t	i_rdev	Real device identifier
off_t	i_size	File length in bytes
time_t	i_atime	Time of last file access
time_t	i_mtime	Time of last file write
time_t	i_ctime	Time of last inode change
unsigned int	i_blkbits	Block size in number of bits
unsigned long	i_blksize	Block size in bytes
unsigned long	i_blocks	Number of blocks of the file
unsigned long	i_version	Version number, automatically incremented after each use
struct semaphore	i_sem	inode semaphore
struct semaphore	i_zombie	Secondary inode semaphore used when removing or renaming the inode

Inode 객체

struct inode_operations *	i_op	inode operations
struct file_operations *	i_fop	Default file operations
struct super_block *	i_sb	Pointer to superblock object
wait_queue_head_t	i_wait	inode wait queue
struct file_lock *	i_flock	Pointer to file lock list
struct address_space *	i_mapping	Pointer to an address_space object (see Chapter 14)
struct address_space	i_data	address_space object for block device file
struct dquot **	i_dquot	inode disk quotas
struct list_head	i_devices	Pointers of a list of block device file inodes (see Chapter 13)
struct pipe_inode_info *	i_pipe	Used if the file is a pipe (see Chapter 19)
struct block_device *	i_bdev	Pointer to the block device driver
struct char_device *	i_cdev	Pointer to the character device driver
unsigned long	i_dnotify_mask	Bit mask of directory notify events
struct dnotify_struct *	i_dnotify	Used for directory notifications
unsigned long	i_state	inode state flags
unsigned int	i_flags	Filesystem mount flags
unsigned char	i_sock	Nonzero if file is a socket
atomic_t	i_writecount	Usage counter for writing processes
unsigned int	i_attr_flags	File creation flags
__u32	i_generation	inode version number (used by some filesystems)
union	u	Specific filesystem information

Inode 객체와 inode operatoins

- inode관련 연산은 inode_operations구조체에 담아 i_op필드에

```
struct inode_operations {
        int (*create) (struct inode *,struct dentry *,int);
        struct dentry * (*lookup) (struct inode *,struct dentry *);
        int (*link) (struct dentry *,struct inode *,struct dentry *);
        int (*unlink) (struct inode *,struct dentry *);
        int (*symlink) (struct inode *,struct dentry *,const char *);
        int (*mkdir) (struct inode *,struct dentry *,int);
        int (*rmdir) (struct inode *,struct dentry *);
        int (*mknod) (struct inode *,struct dentry *,int,int);
        int (*rename) (struct inode *, struct dentry *,
                        struct inode *, struct dentry *);
        int (*readlink) (struct dentry *, char *,int);
        int (*follow_link) (struct dentry *, struct nameidata *);
        void (*truncate) (struct inode *);
        int (*permission) (struct inode *, int);
        int (*revalidate) (struct dentry *);
        int (*setattr) (struct dentry *, struct iattr *);
        int (*getattr) (struct dentry *, struct iattr *);
        int (*setxattr) (struct dentry *, const char *, void *, size_t, int);
        ssize_t (*getxattr) (struct dentry *, const char *, void *, size_t);
        ssize_t (*listxattr) (struct dentry *, char *, size_t);
        int (*removexattr) (struct dentry *, const char *);
};
```

File 객체

✓ 파일 객체(struct file <linux/fs.h>)

Type	Field	Description
struct list_head	f_list	Pointers for generic file object list
struct dentry *	f_dentry	dentry object associated with the file
struct vfsmount *	f_vfsmnt	Mounted filesystem containing the file
struct file_operations *	f_op	Pointer to file operation table
atomic_t	f_count	File object's usage counter
unsigned int	f_flags	Flags specified when opening the file
mode_t	f_mode	Process access mode
loff_t	f_pos	Current file offset (file pointer)
unsigned long	f_reada	Read-ahead flag
unsigned long	f_ramax	Maximum number of pages to be read-ahead
unsigned long	f_raend	File pointer after last read-ahead
unsigned long	f_ralen	Number of read-ahead bytes
unsigned long	f_rawin	Number of read-ahead pages
struct fown_struct	f_owner	Data for asynchronous I/O via signals
unsigned int	f_uid	User's UID
unsigned int	f_gid	User's GID
int	f_error	Error code for network write operation
unsigned long	f_version	Version number, automatically incremented after each use
void *	private_data	Needed for tty driver
struct kiobuf *	f_iobuf	Descriptor for direct access buffer (see Section 15.2)
long	f_iobuf_lock	Lock for direct I/O transfer

File 객체와 file operations

- 파일에 대한 연산을 file_operations 구조체에 담아 f_op에

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
};
```

Dentry 객체

- ✓ 디엔트리 객체(struct dentry <linux/dcache.h>)
  - 디렉토리 엔트리를 메모리로 읽은 뒤, VFS가 dentry 구조체의 디엔트리 객체로 변환
  - P가 탐색하는 경로명의 모든 구성 요소에 대해 디 엔트리 객체 생성

Type	Field	Description
atomic_t	d_count	Dentry object usage counter
unsigned int	d_flags	Dentry flags
struct inode *	d_inode	Inode associated with filename
struct dentry *	d_parent	Dentry object of parent directory
struct list_head	d_hash	Pointers for list in hash table entry
struct list_head	d_lru	Pointers for unused list
struct list_head	d_child	Pointers for the list of dentry objects included in parent directory
struct list_head	d_subdirs	For directories, list of dentry objects of subdirectories
struct list_head	d_alias	List of associated inodes (alias)
int	d_mounted	Flag set to 1 if and only if the dentry is the mount point for a filesystem
struct qstr	d_name	Filename
unsigned long	d_time	Used by d_revalidate method
struct dentry_operations*	d_op	Dentry methods
struct super_block *	d_sb	Superblock object of the file
unsigned long	d_vfs_flags	Dentry cache flags
void *	d_fsdata	Filesystem-dependent data
unsigned char *	d_iname	Space for short filename

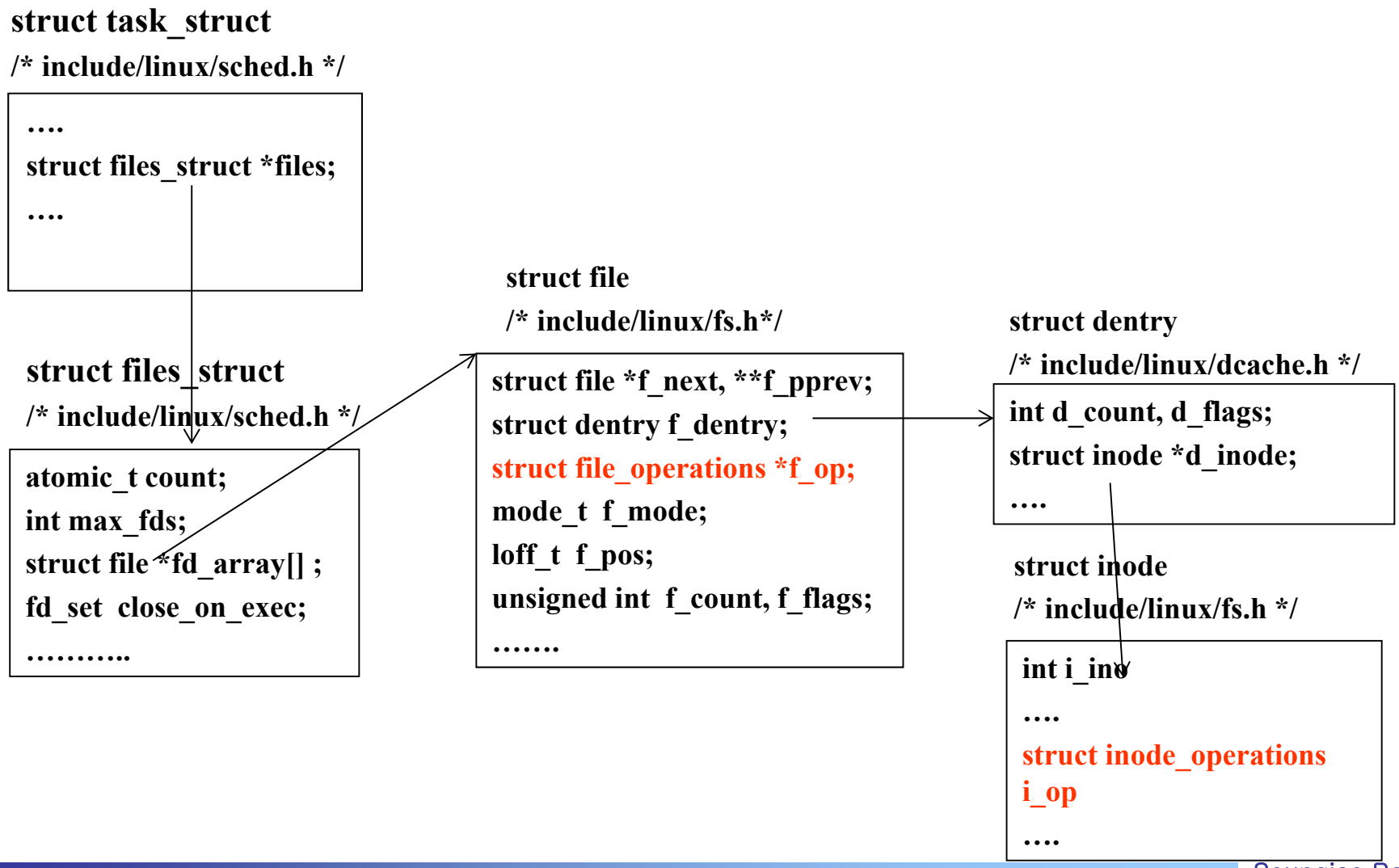
Dentry 객체와 dentry operations

- dentry 객체에 대한 연산을 dentry_operations구조체에 담아 d_op 필드에, 대개 NULL임

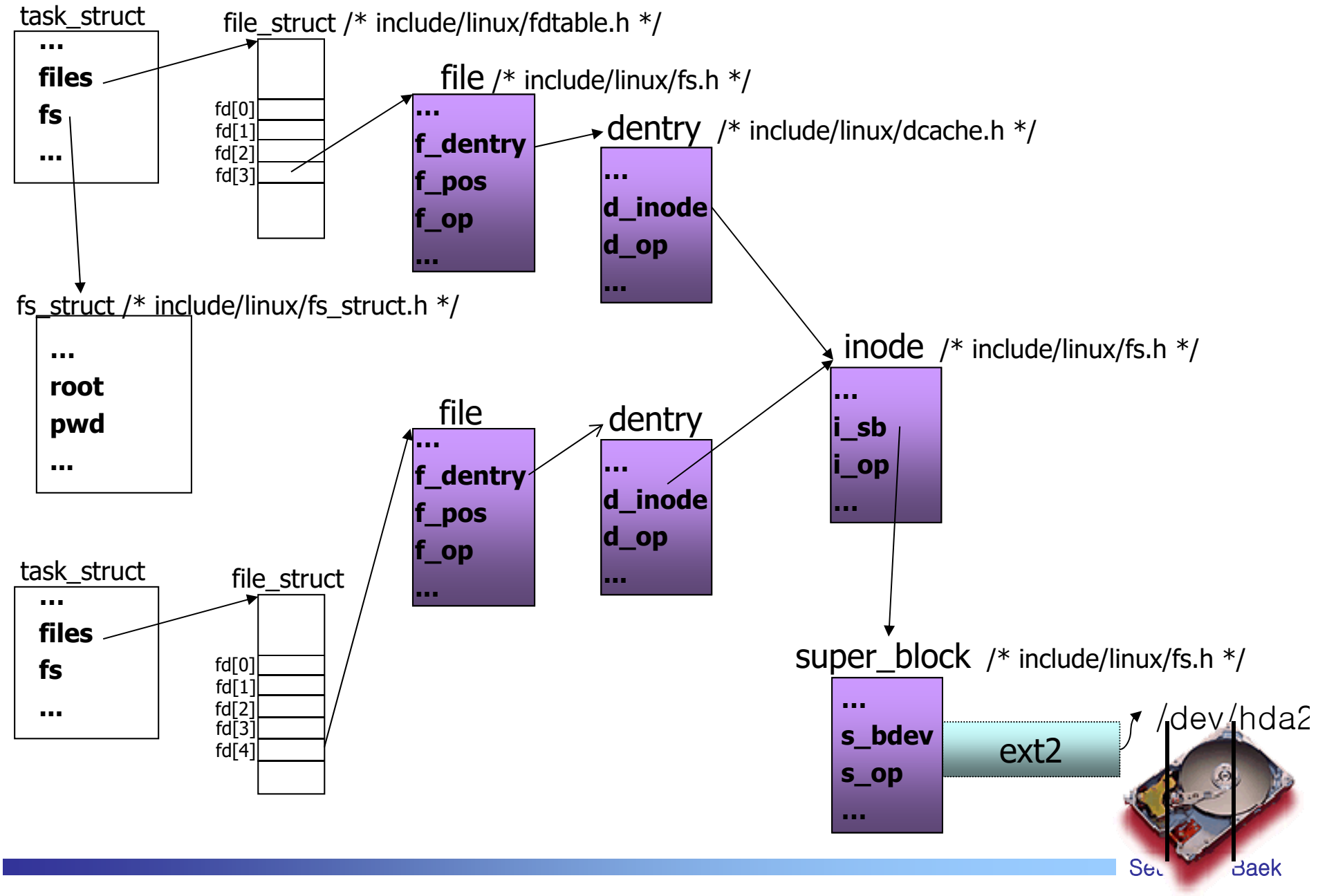
```
struct dentry {
        atomic_t d_count;
        unsigned int d_flags;
        struct inode  * d_inode;        /* Where the name belongs to - NULL is negative */
        struct dentry * d_parent;       /* parent directory */
        struct list_head d_hash;        /* lookup hash list */
        struct list_head d_lru;         /* d_count = 0 LRU list */
        struct list_head d_child;       /* child of parent list */
        struct list_head d_subdirs;     /* our children */
        struct list_head d_alias;       /* inode alias list */
        int d_mounted;
        struct qstr d_name;
        unsigned long d_time;           /* used by d_revalidate */
        struct dentry_operations  *d_op;
        struct super_block * d_sb;      /* The root of the dentry tree */
        unsigned long d_vfs_flags;
        void * d_fsdata;                /* fs-specific data */
        unsigned char d_iname[DNAME_INLINE_LEN]; /* small names */
};
```

Linux의 파일 연산 구현 분석 (1/5)

- 파일 연산 : file_operations
- inode 연산 : inode_operations

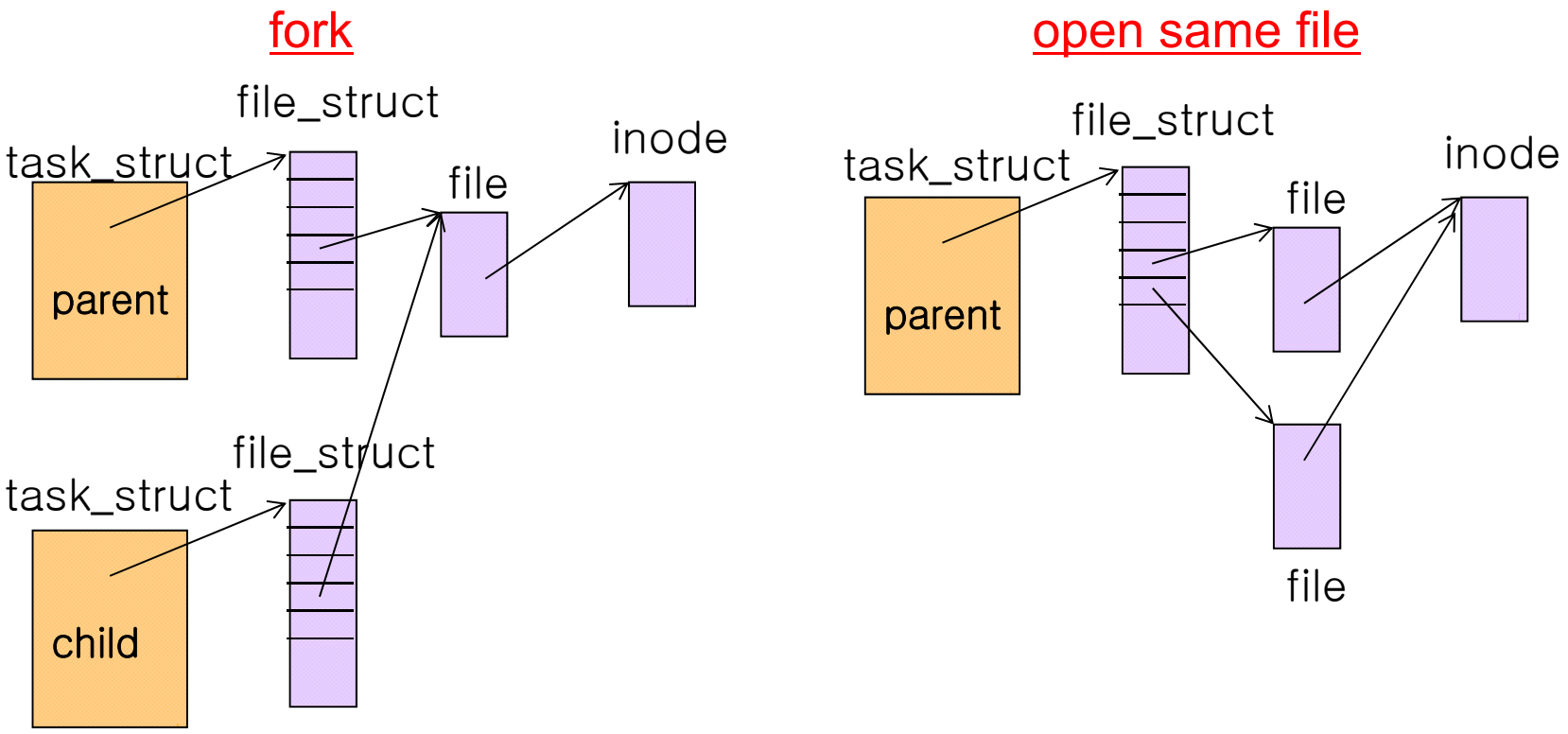


task_struct와 VFS의 객체



파일 인터페이스

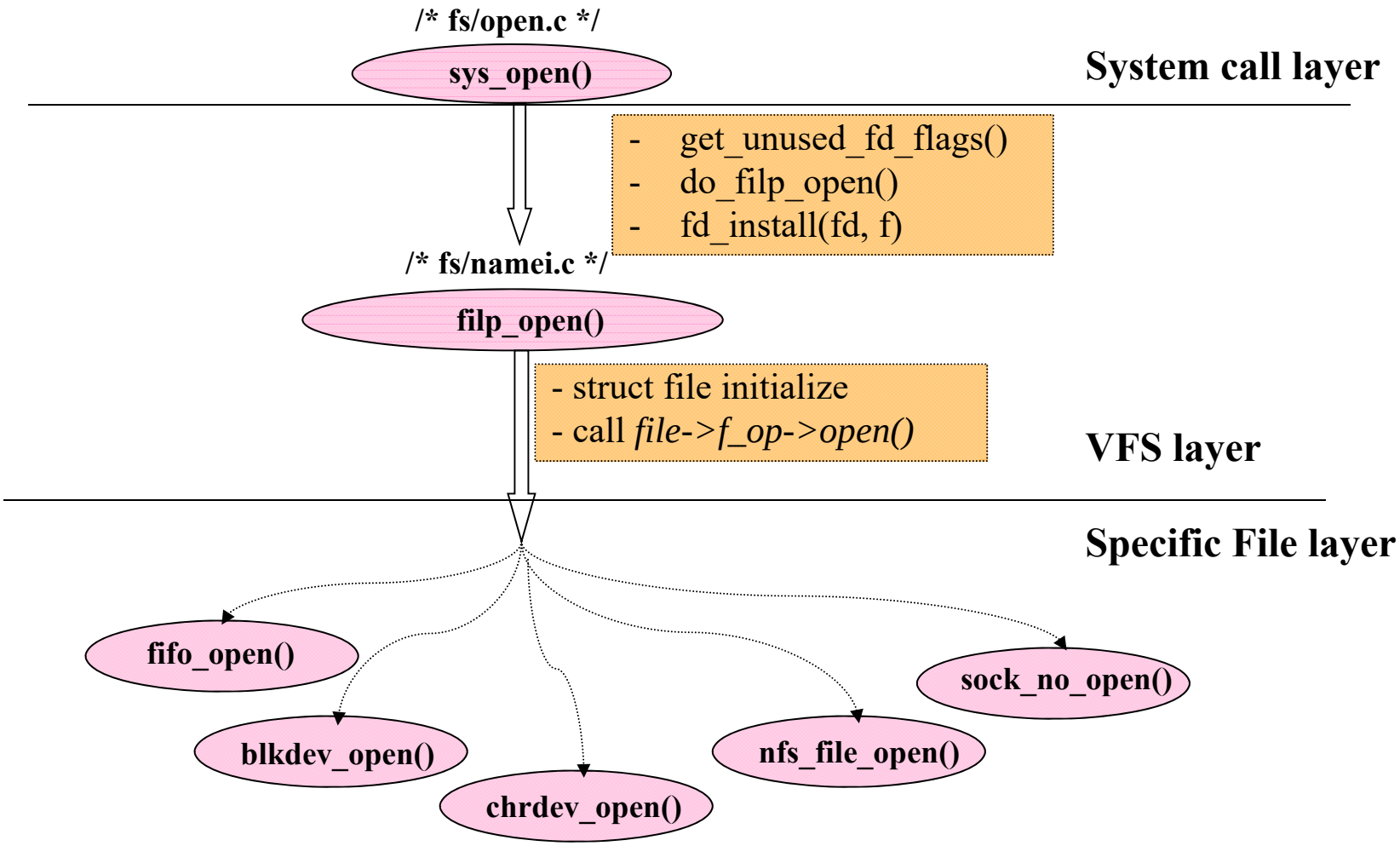
- 커널 자료 구조 관계



- how about dup?

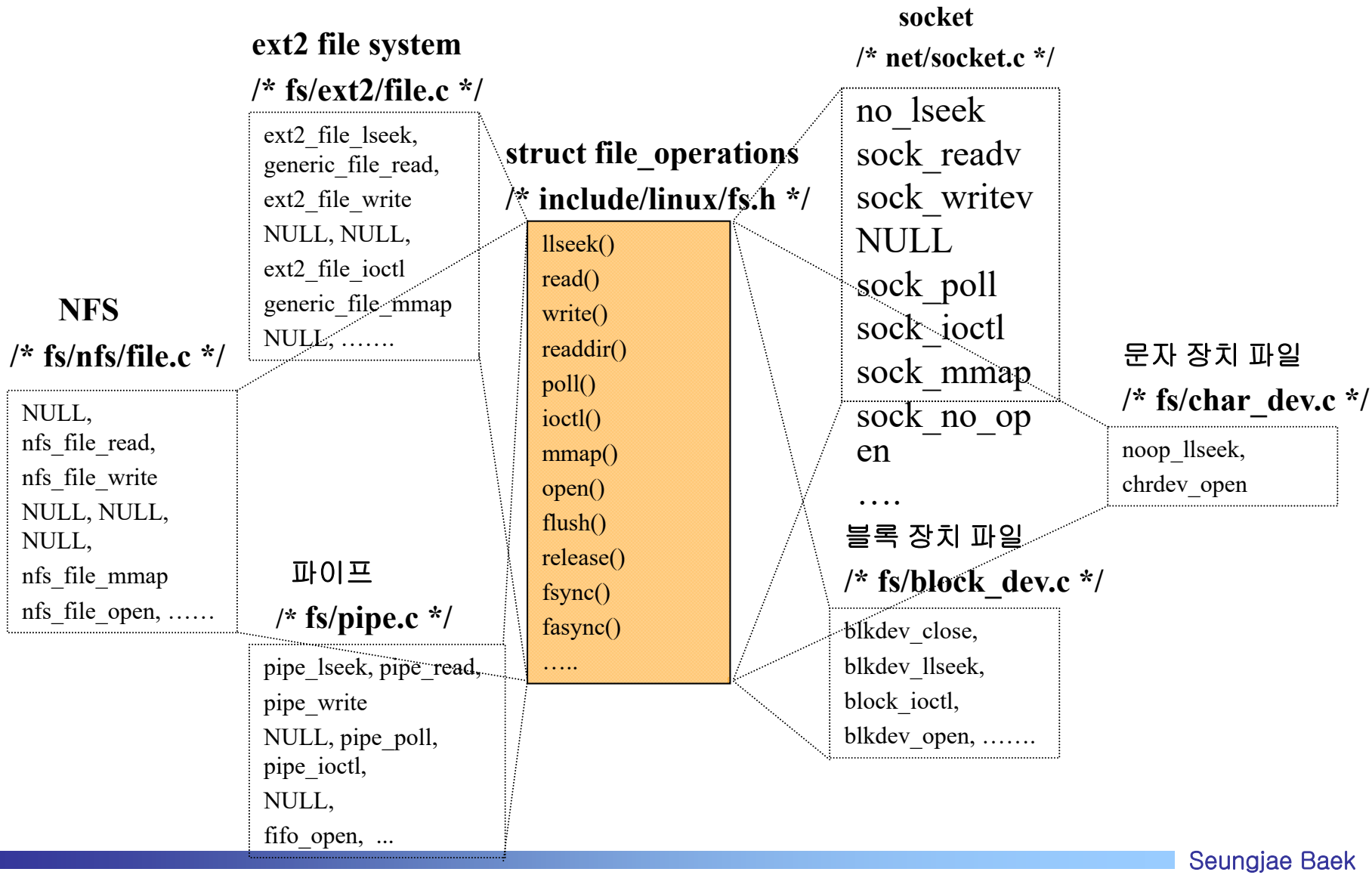
Linux의 파일 연산 구현 분석 (2/5)

- open 시스템 호출 분석



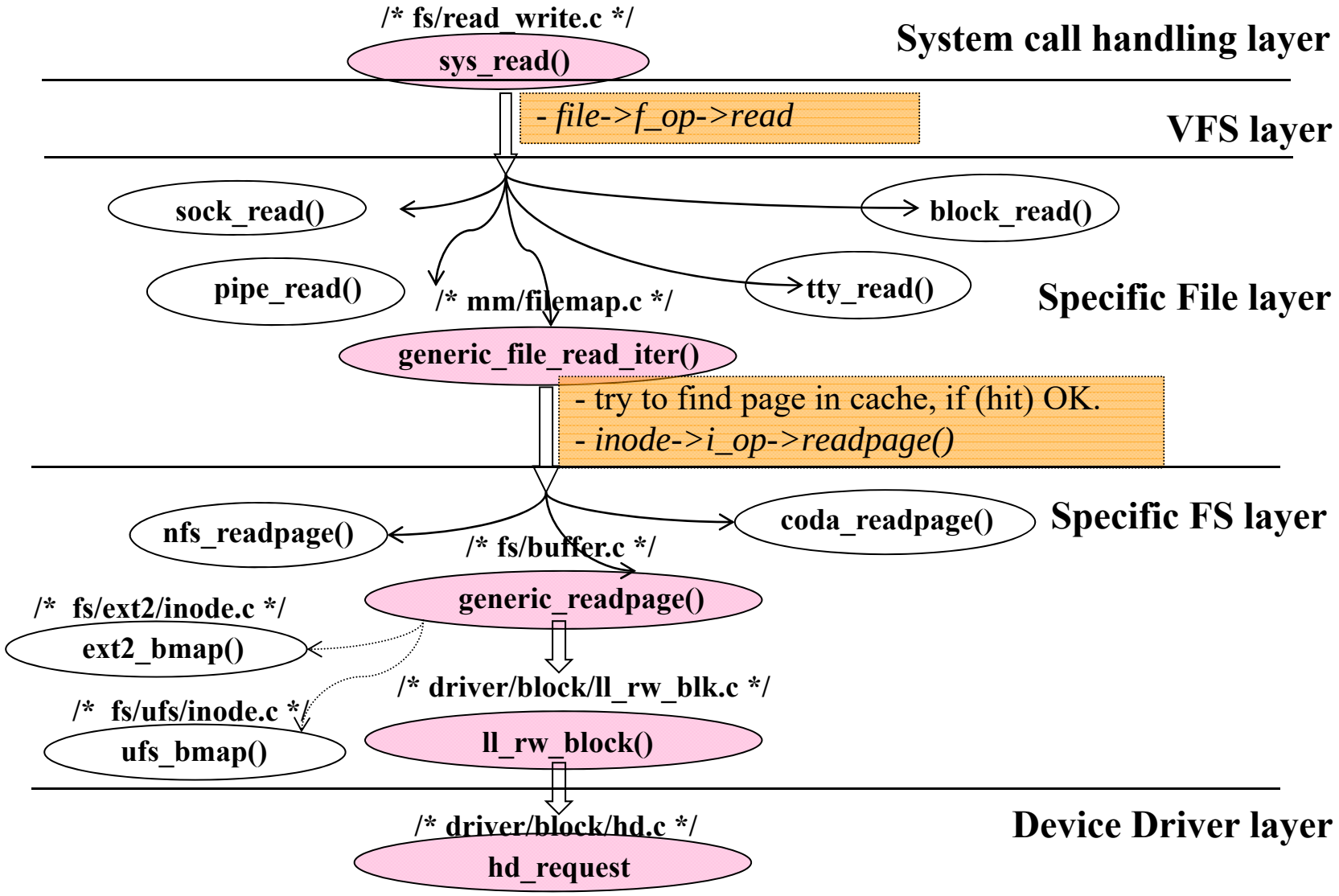
Linux의 파일 연산 구현 분석 (3/5)

- struct file_operations 구조 : 다양한 파일 유형을 일관된 인터페이스로 지원



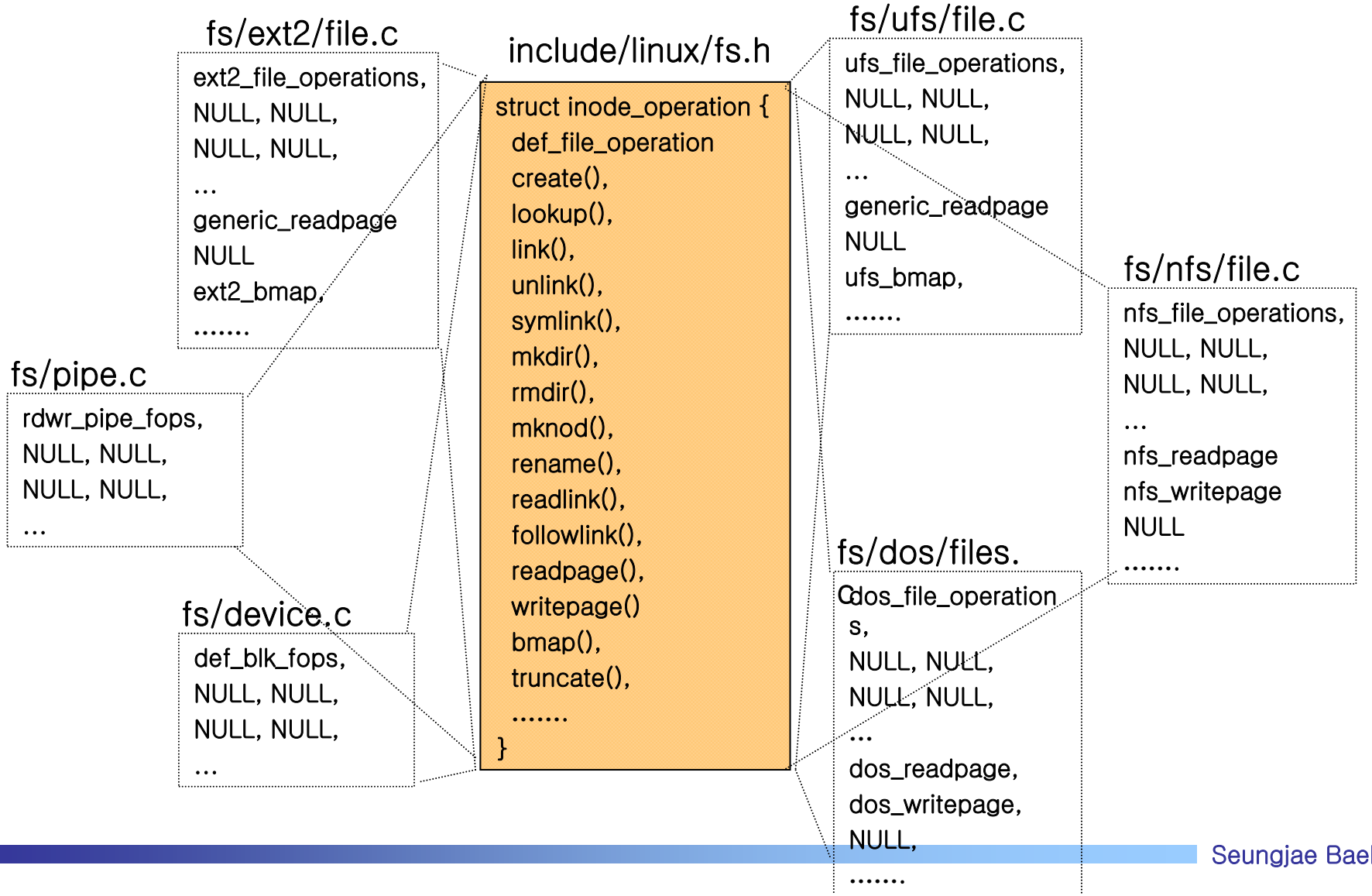
Linux의 파일 연산 구현 분석 (4/5)

- open 시스템 호출 분석



Linux의 파일 연산 구현 분석 (5/5)

- struct inode_operations 구조 : 다양한 파일 시스템을 일관된 인터페이스로 지원



파일시스템 등록

- register_filesystem()

```
int register_filesystem(struct file_system_type * fs)
{
        int res = 0;
        struct file_system_type ** p;

        if (!fs)
                return -EINVAL;
        if (fs->next)
                return -EBUSY;
        INIT_LIST_HEAD(&fs->fs_supers);
        write_lock(&file_systems_lock);
        p = find_filesystem(fs->name);
        if (*p)
                res = -EBUSY;
        else
                *p = fs;
        write_unlock(&file_systems_lock);
        return res;
}
```

간단하다. file_system_type 구조체 하나 채워주고 호출하면 끝

파일시스템 등록

이때 file_system_type 구조체의 내용은 아래와 같다. read_super 필드는 슈퍼 블록을 읽는 함수를 지정해 주는 필드이다

```
struct file_system_type {
        const char *name;
        int fs_flags;
        struct super_block *(*read_super) (struct super_block *, void *, int);
        struct module *owner;
        struct file_system_type * next;
        struct list_head fs_supers;
};
```

앞장에서 find_filesystem()함수는 현재 시스템에 올라와 있는 모든 FS를 찾아봐서 중복된 놈을 올리는게 아닌지 확인하는 역할을 한다

```
static struct file_system_type **find_filesystem(const char *name)
{
        struct file_system_type **p;
        for (p=&file_systems; *p; p=&(*p)->next)
                if (strcmp((*p)->name,name) == 0)
                        break;
        return p;
}
```

파일시스템 등록

구조체는 직접 채워도 되지만, 보통 모두 DECLARE_FSTYPE() 매크로를 사용한다. 쉽고 간단하다.

```
#define DECLARE_FSTYPE(var,type,read,flags) \
struct file_system_type var = { \
        name:           type, \
        read_super:     read, \
        fs_flags:       flags, \
        owner:          THIS_MODULE, \
};

#define DECLARE_FSTYPE_DEV(var,type,read) \
        DECLARE_FSTYPE(var,type,read,FS_REQUIRES_DEV)
```

Example(17/19)-FS제작

- FS를 Module형태로 제작 한다면?

```
/* NEB_super.c */

.....

static DECLARE_FSTYPE(neb_type, "neb", neb_read_super, FS_SINGLE);
static int __init init_neb(void)
{
        return register_filesystem(&neb_type);
}
int init_module(void)
{
        dprintk("<0>%s:%s(%d)₩n",__FILE__, __FUNCTION__, __LINE__);
        return init_neb();
}

.....
```

Example(18/19)-FS제작

- User가 mount명령을 내리면 ?

```
static struct super_block *neb_read_super(struct super_block *sb, void *data, int silent)
{
        dprintk("<0>%s:%s(%d)₩n",__FILE__, __FUNCTION__, __LINE__);

        struct inode *root_inode;
        int vol, err;
        dev_t dev;
        neb_file_id_t fid;
        unsigned long tmp;

        vol = MINOR(sb->s_dev) & 0x7;
        dev = MKDEV(MAJOR(sb->s_dev), MINOR(sb->s_dev)-vol);

        if ((err = _neb_mount_vol(dev, vol-1)) < 0) {
                printk("<0> %s: _neb_mount_vol failed: %d₩n", __FUNCTION__, err);
                goto neb_sb_err1;
        }

        // read_super의 인자로 넘어온 super block 포인터 sb에 file system specific 한 정보를 기입
        if ((err = _neb_read_super(dev, vol-1, &sb->u.neb_sb)) < 0) {
                printk("<0> %s: _neb_read_super failed: %d₩n", __FUNCTION__, err);
                goto neb_sb_err1;
        }
```

Example(19/19)-FS제작


```
        // 나머지 sb 의 정보를 채움
        sb->s_blocksize = sb->u.neb_sb.cluster_size;
        sb->s_blocksize_bits = 0;
        tmp = sb->s_blocksize;
        while (tmp > 1) {
                sb->s_blocksize_bits++;
                tmp >>= 1;
        }
        sb->s_maxbytes = 0xFFFFFFFF;    // maximum file size
        sb->s_magic = NEB_MAGIC;   sb->s_op = &neb_ops;          sb->s_dirt = 0;
        // 지역변수 fid에다가 root directory의 inode 정보를 기입
        fid.dev = dev;          fid.vol = vol-1;
        fid.parent_first_block = sb->u.neb_sb.root_cluster;
        fid.dir_first_block = sb->u.neb_sb.root_cluster;
        strcpy(fid.filename, ".");
        if ((err = _neb_lookup(&fid)) < 0) {
                printk("<0>%s: _neb_lookup failed: %d₩n", __FUNCTION__, err);
                goto neb_sb_err1;   }
        // root_inode 에 root dir 을 위한 inode 할당
        root_inode = iget4(sb, sb->u.neb_sb.root_cluster, (find_inode_t) _neb_find_actor,
(void *) &fid);

        if (!root_inode)        goto neb_sb_err1;
        // root_inode 를 위한 dentry할당
        if (!(sb->s_root = d_alloc_root(root_inode)))        goto neb_sb_err2;
        return sb;
}
```

Example(1/3)- Lock 실험

- Process A에서 fcntl()이나 flock()함수를 사용하여 파일 a.txt에 lock을 건다
- Process A가 fork()하여 Process A'를 생성한다
- Process A'가 파일 a.txt에 접근 하려 하면?

- Process A가 파일 a.txt에 lock을 잡고 있는 동안, 창을 하나 더 띄워서 vi 편집기로 a.txt를 편집하려고 하면?

Example(2/3)- Lock 실험

- Advisory lock(POSIX : fcntl(), BSD : `flock()`)
  - OS가 특정 프로세스에 의해 Locking된 파일에 대하여 정확한 자료를 유지하되, 다른 프로세스에 의해 Locking된 파일에 대하여 쓰기를 막지 않는 것을 말함
  - 프로세스는 advisory lock을 무시하고 적절한 권한이 있다면 Lock된 파일에 쓰기를 할 수 있음
- Mandatory lock(System V Release 3 : `lockf()`)
  - 호출 프로세스가 접근하려는 파일에 대해 lock을 위반하지 않는가를 검사하기 위해 kernel로 하여금 open, read, write를 검사하는 방식