ARCOS Group

Universidad Carlos III de Madrid

Lesson 5 (b)

File systems

Operating System Design

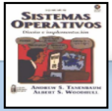
Bachelor in Informatics Engineering

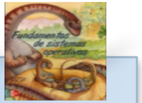
Recommended readings

Base

1. Carretero 2007:
    1. Chapter 9

Additional





1. Tanenbaum 2006(en):
    1. Chap.5
2. Stallings 2005:
    1. Three part
3. Silberschatz 2014:
    1. Chap. 10, 11 & 12

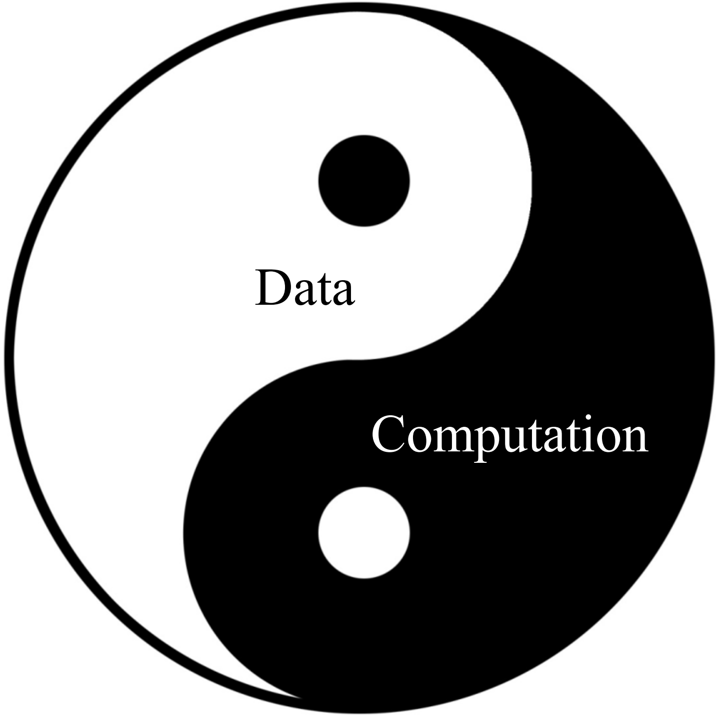
Overview

1. Introduction
2. Main data structures in the secondary memory
3. Main data structures in the main memory
4. Block management
5. Complementary aspects

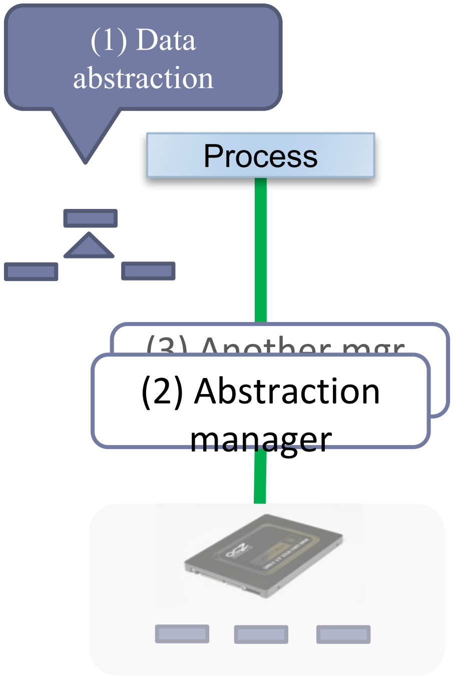
Overview

1. **Introduction**
2. Main data structures in the secondary memory
3. Main data structures in the main memory
4. Block management
5. Complementary aspects

Storage System Scope



Storage System Scope



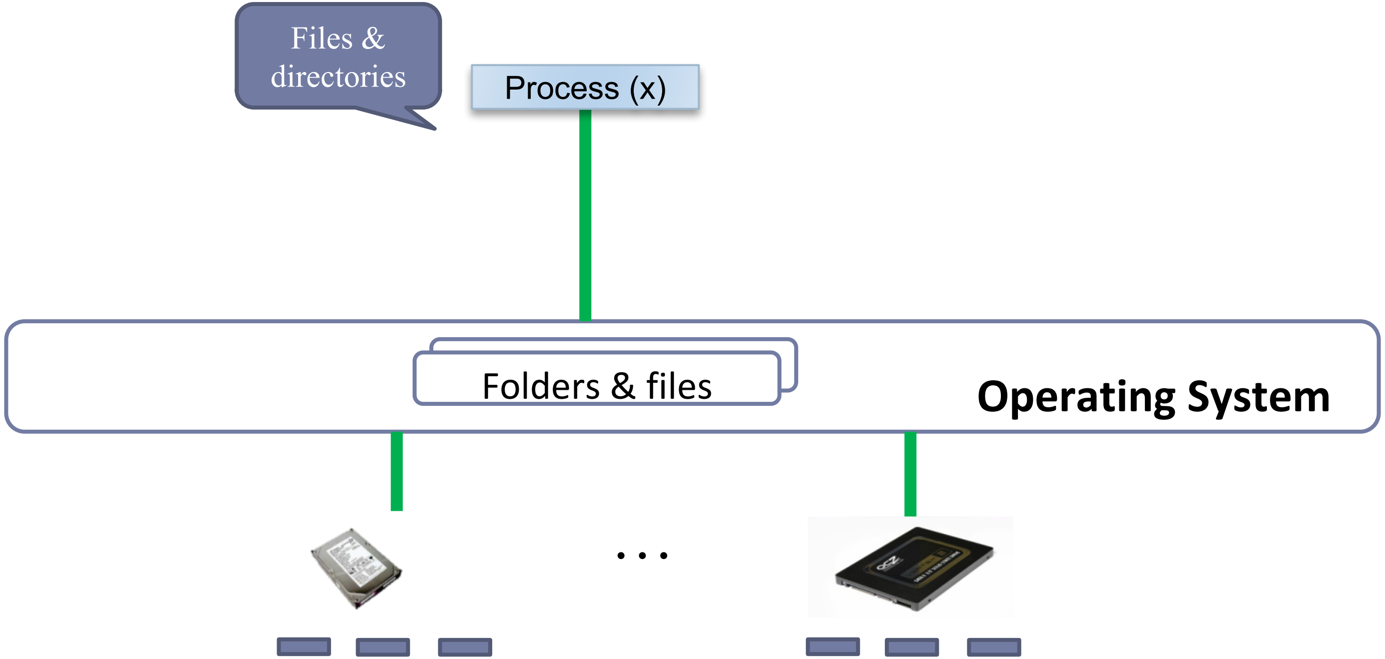
(1) Data abstraction

Process

(3) Another mgr.

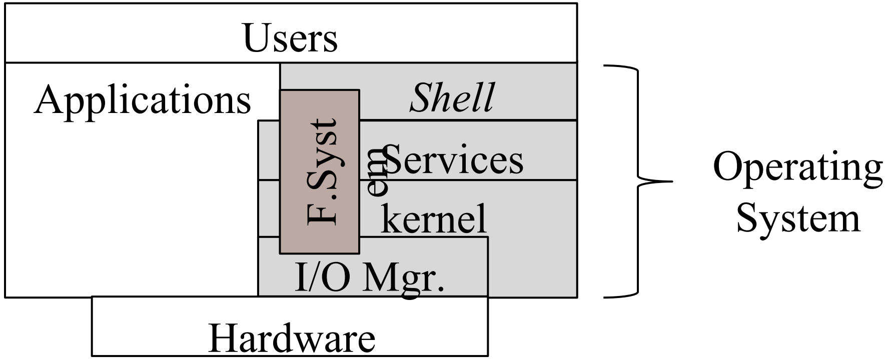
(2) Abstraction manager

(1/2) The O.S. includes a basic and generic abstraction: file system

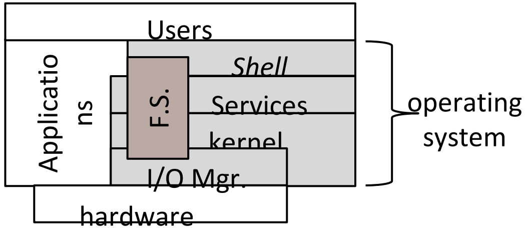


(1/2) File system included

- The Operating System includes some implementations of a basic abstract representation for the storage systems: the file system.



File system Characteristics

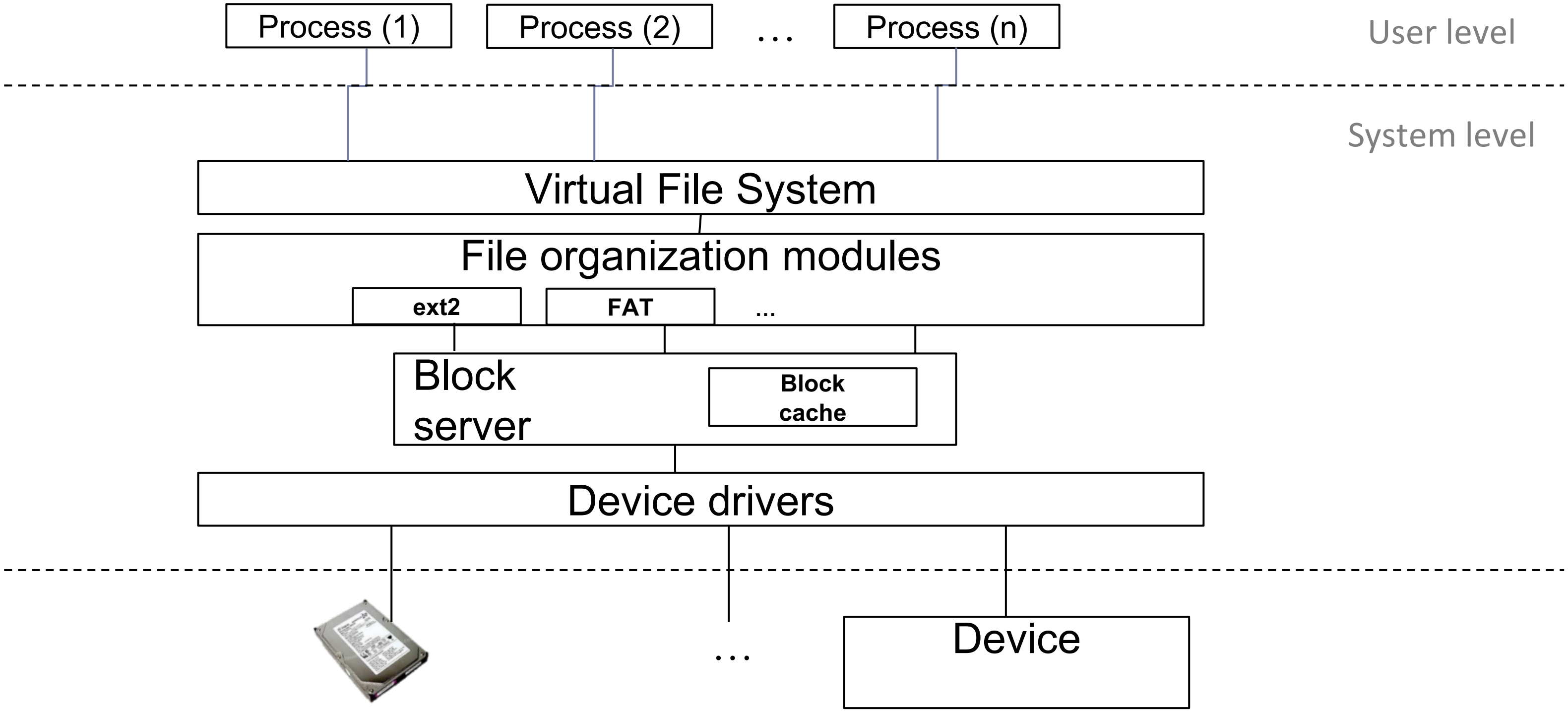


- Facilitate the secondary storage management.
  - Files, directories, etc.

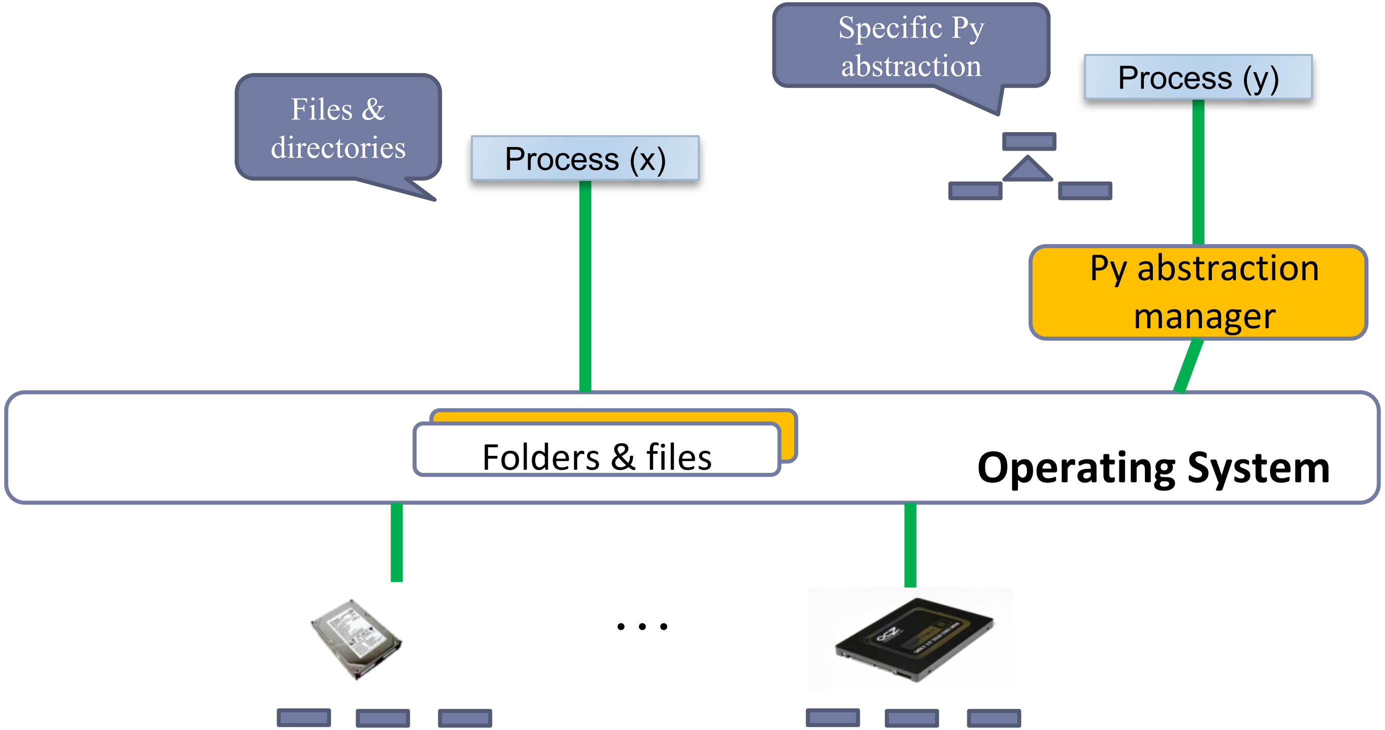
- Independent from the physical device.

- Offer a unified logical view for users and applications.

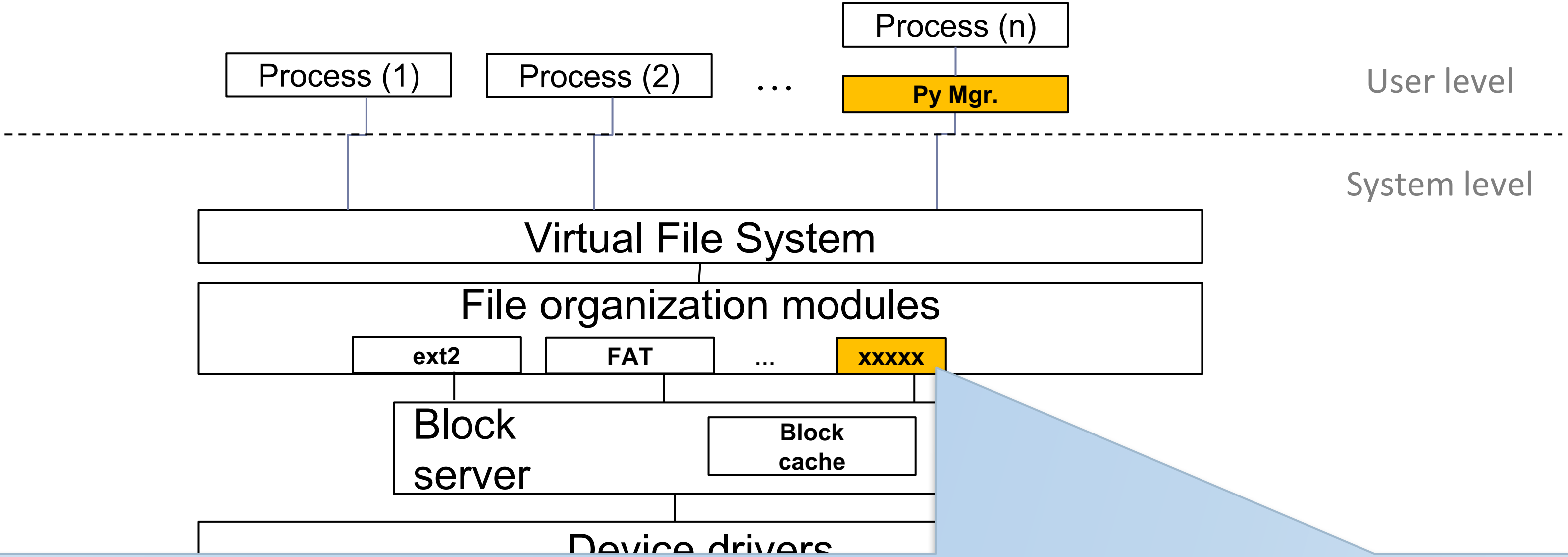
File system Architecture



(2/2) The operating system supports the addition of other abstractions (& mgr.)

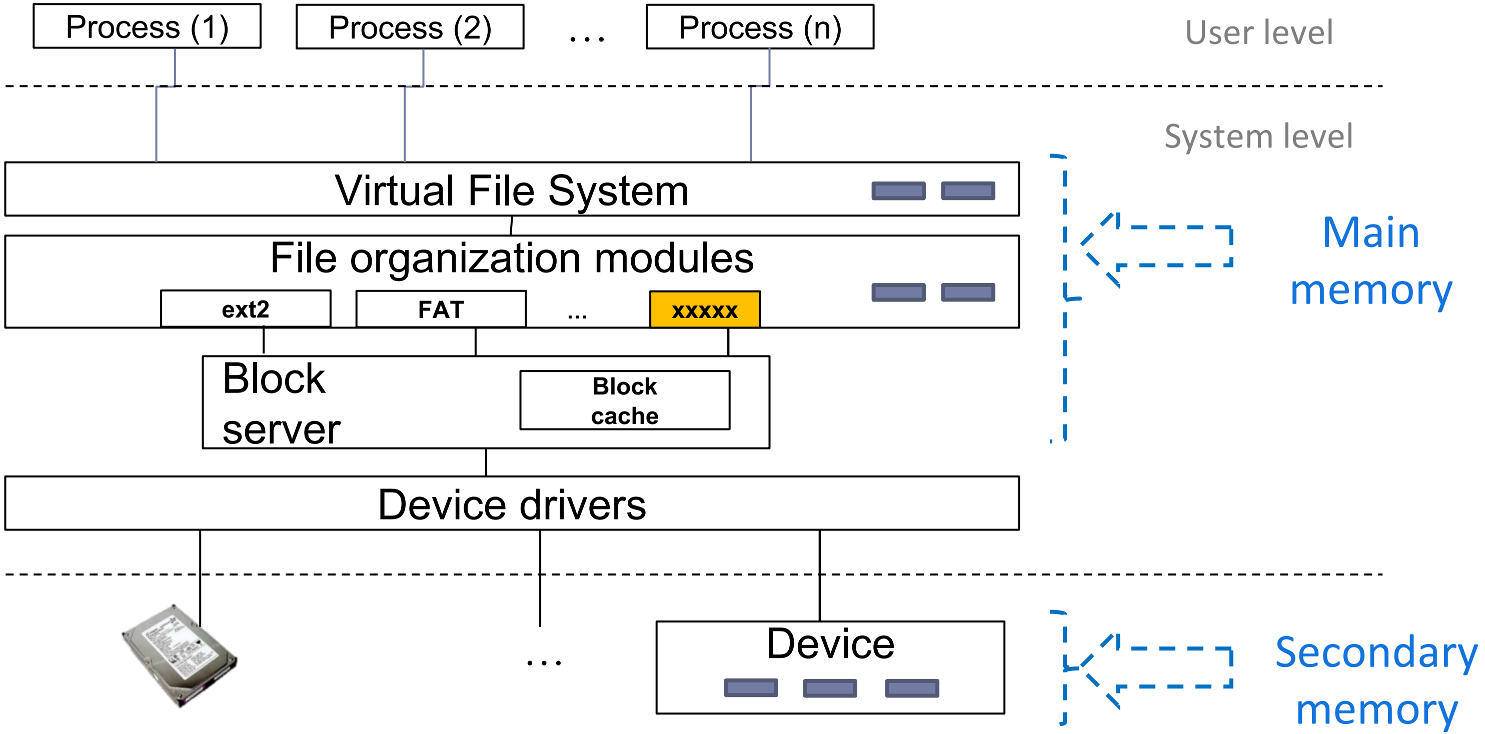


File system Architecture

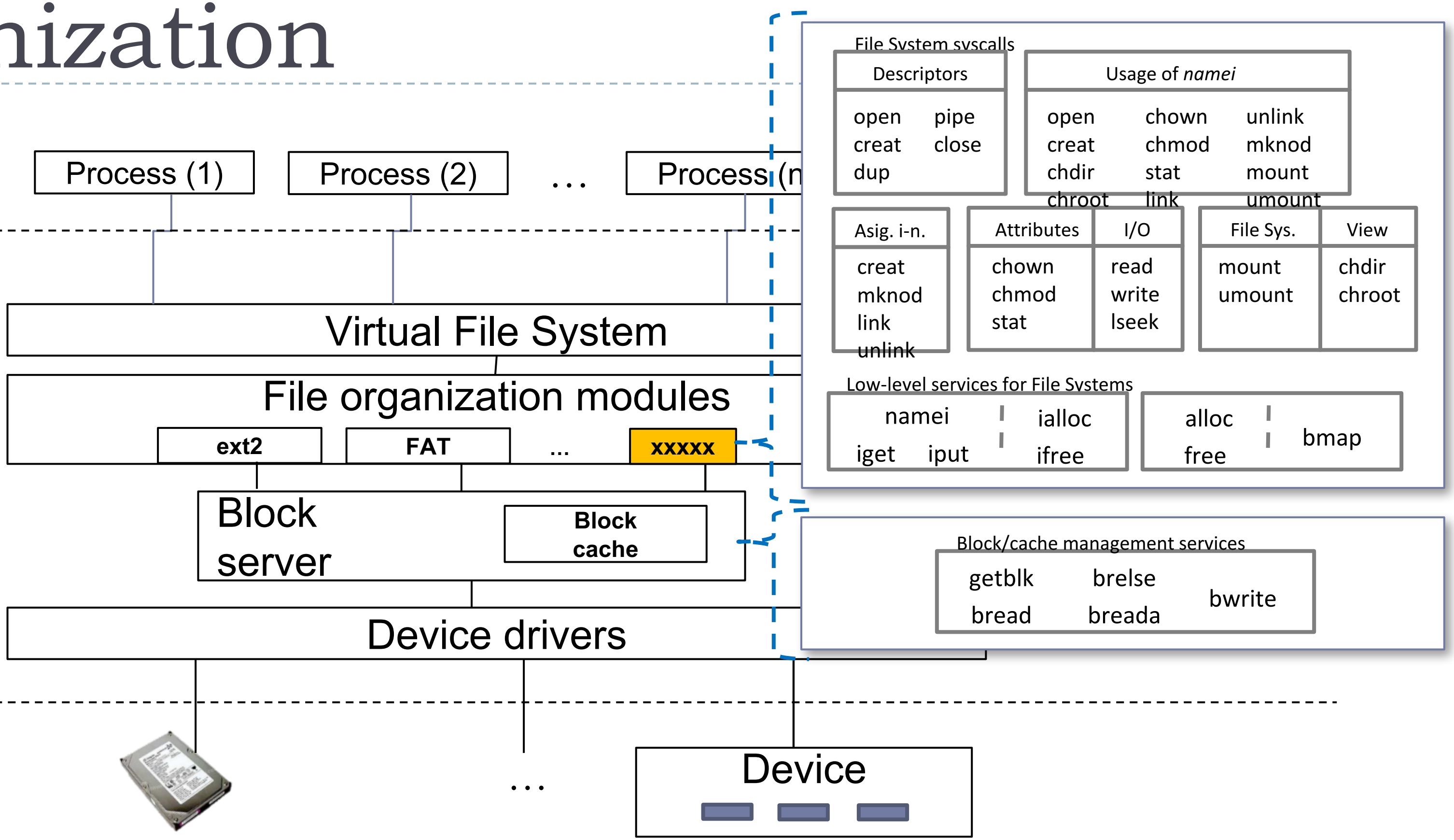


- A new file system implementation could be added.
- Other abstract representations could be implemented using the existing services on the Operating Systems.

(1/2) Management structures



(2/2) Management organization



Process (1)

Process (2)

…

Process (n

Virtual File System

File organization modules

ext2

FAT

…

xxxxx

Block server

Block cache

Device drivers

…

Device

File System syscalls

Descriptors	
open	pipe
creat	close
dup	

Usage of *namei*		
open	chown	unlink
creat	chmod	mknod
chdir	stat	mount
chroot	link	umount

Asig. i-n.	Attributes	I/O	File Sys.	View
creat	chown	read	mount	chdir
mknod	chmod	write	umount	chroot
link	stat	lseek		
unlink				

Low-level services for File Systems

namei | ialloc
iget iput | ifree

alloc | bmap
free

Block/cache management services

getblk brelse bwrite
bread breada

Main requirements
e.g.: Unix-like file system

- Processes have to use a secure interface, without direct access to the kernel data structures.
- Share the file offset position among processes from the same parent that open the file.
- Offer functionality for working with a file/directory in order to update the information that it contains.
- Go back and forth in the file system directory tree.
- Offer persistency of user data, seeking to minimize the impact on the performance and the space needed for the metadata.
- Keep track of the file systems registered in the kernel, and keep track of the mount point of these file systems.

Getting the proper storage system for the requirements…

http://en.wikipedia.org/wiki/List_of_file_systems

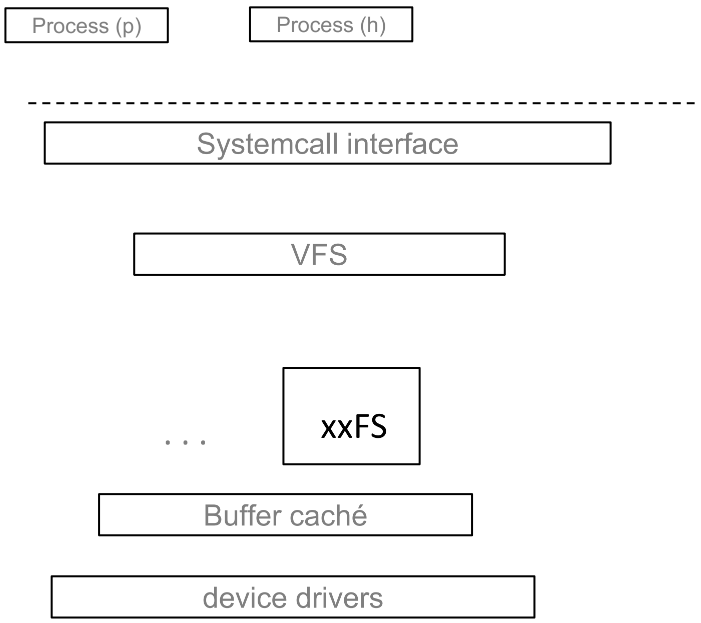
1. To search a file system that satisfies the requirements.

2. To adapt an existing file system in order to satisfy the requirements.

3. To build a file system that satisfies the requirements.

File system organization

main aspects: Linux

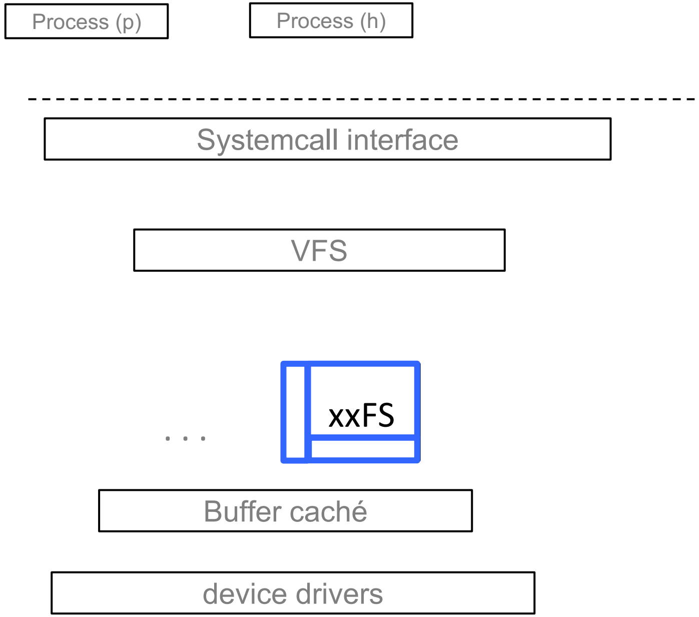


- Layered structure like UNIX.
- Main components:
  - System call interface
  - VFS: *Virtual File System*
  - xxFS: specific file system
  - Buffer caché: block cache
  - device drivers: *drivers*

http://publibn.boulder.ibm.com/doc_link/en_US/a_doc_lib/aixprggd/kernextc/virtual_fsys_over.htm#HDRA29C010F11

File system organization

without *framework*, within kernel. E.g.: simplefs

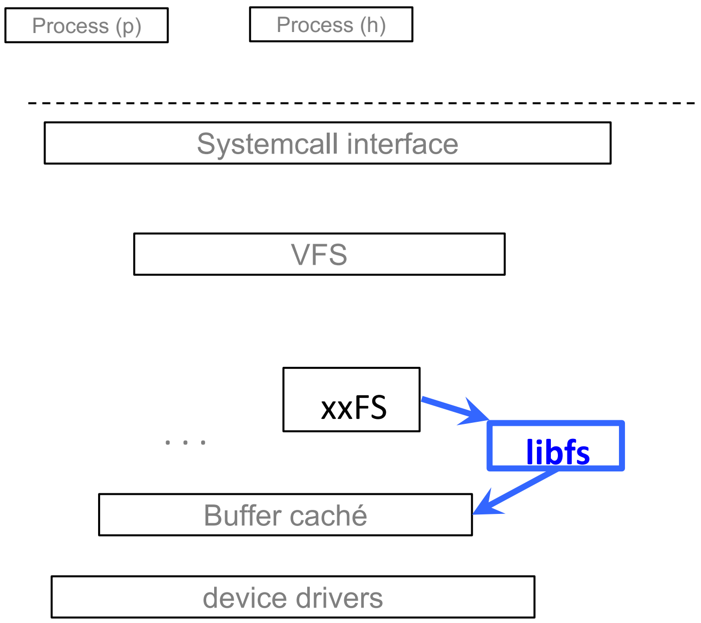


- Interface:
  - **register**: to register the file system
  - …
  - **open**: to open a work session
  - **read**: read data
  - …
  - **namei**: convert from path to i-node
  - **iget**: read a i-node
  - **bmap**: compute an associated offset block
  - …

https://github.com/psankar/simplefs

File system organization

with *framework*, within kernel: libfs

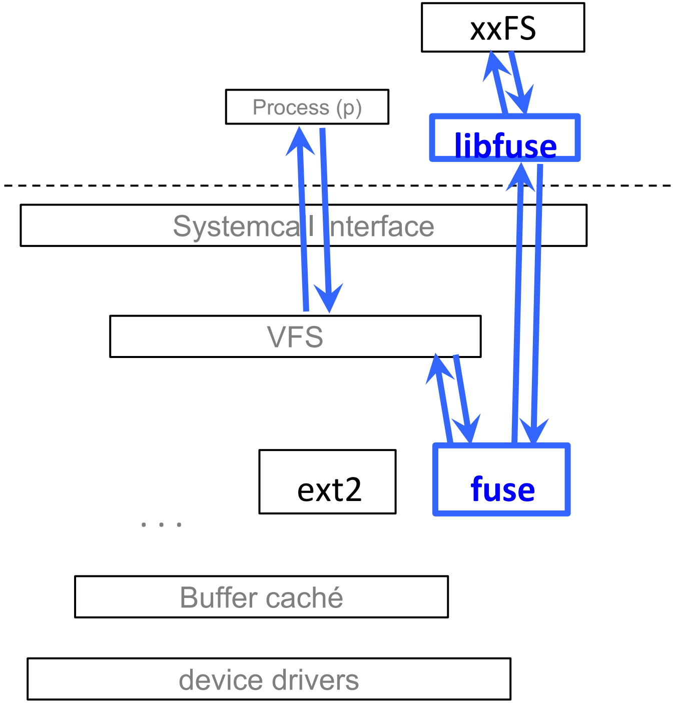


- Interface: libfs
  - **lfs_fill_super**: superblock
  - **lfs_create_file**: file creation
  - **lfs_make_inode**: default i-node
  - **lfs_open**: open a work session
  - **lfs_read_file**: read from file
  - **lfs_write_file**: write to file
  - …

http://lwn.net/Articles/13325/

File system organization

with *framework*, user space: fuse



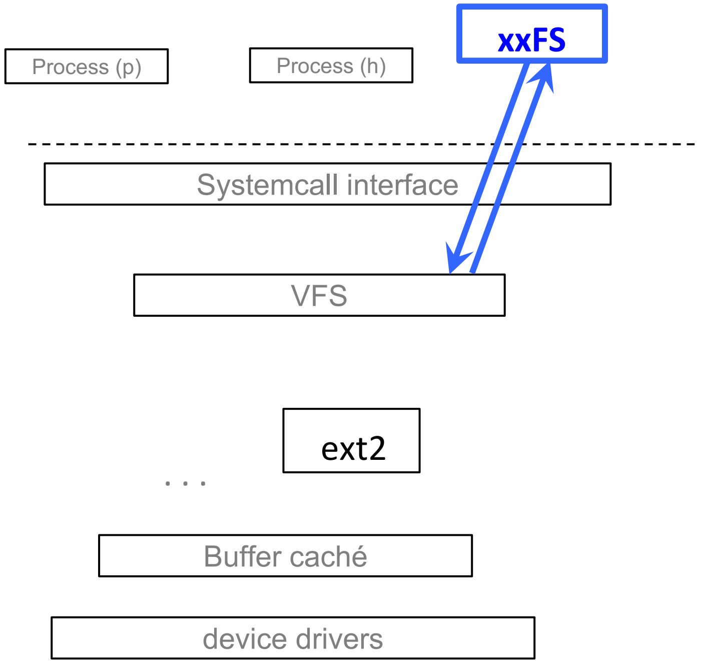
- Interface:
  File system in USer spacE

```
struct fuse_operations {
   …
   int (*open) (const char *, struct fuse_file_info *);
   int (*read) (const char *, char *, size_t, off_t, struct fuse_file_info *);
   int (*write) (const char *, const char *, size_t, off_t,struct fuse_file_info *);
   int (*statfs) (const char *, struct statfs *);
   int (*flush) (const char *, struct fuse_file_info *);
   …
};
```

http://www.ibm.com/developerworks/linux/library/l-fuse/

File system organization

without *framework*, user space. E.g.: mtools

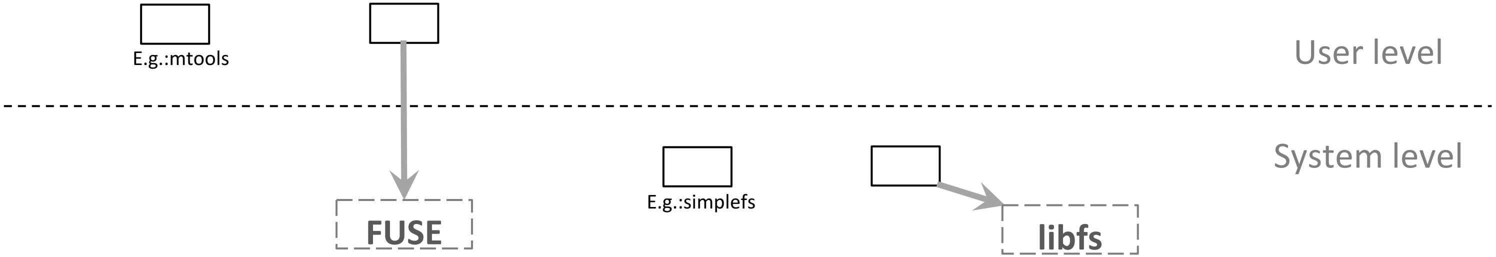


- To implement the file system interface in user space, and as library for other applications:
  - **open**: to open a work session
  - **read**: to read data
  - …
  - **namei**: to convert path into i-node
  - **iget**: read i-node
  - **bmap**: compute the associate block for a given offset
  - …

http://www.gnu.org/software/mtools/manual/mtools.html

Main options for the file system organization

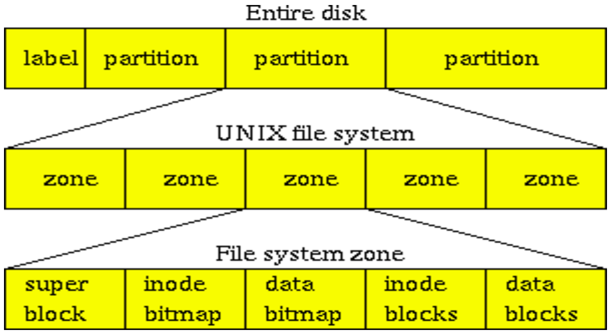
	User space	Kernel space
With *Framework*	FUSE	libfs
Without *Framework*	E.g.: mtools	E.g.: simplefs



Overview

1. Introduction
2. **Main data structures on the secondary memory**
3. Main data structures in the main memory
4. Block management
5. Complementary aspects

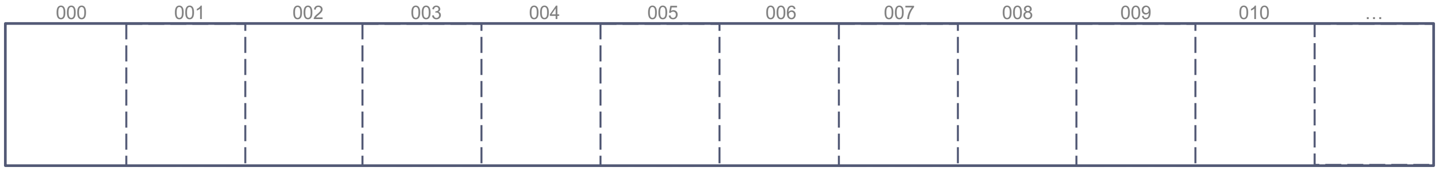
File system Structures



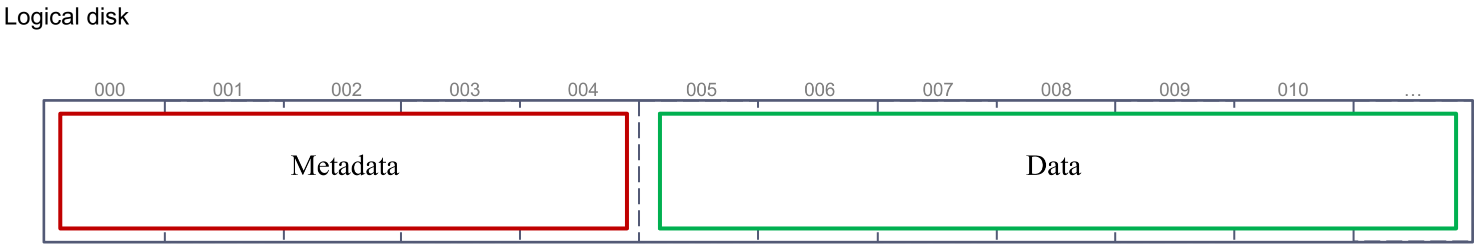
- UNIX/Linux
- FAT

File system:
Unix-like representation

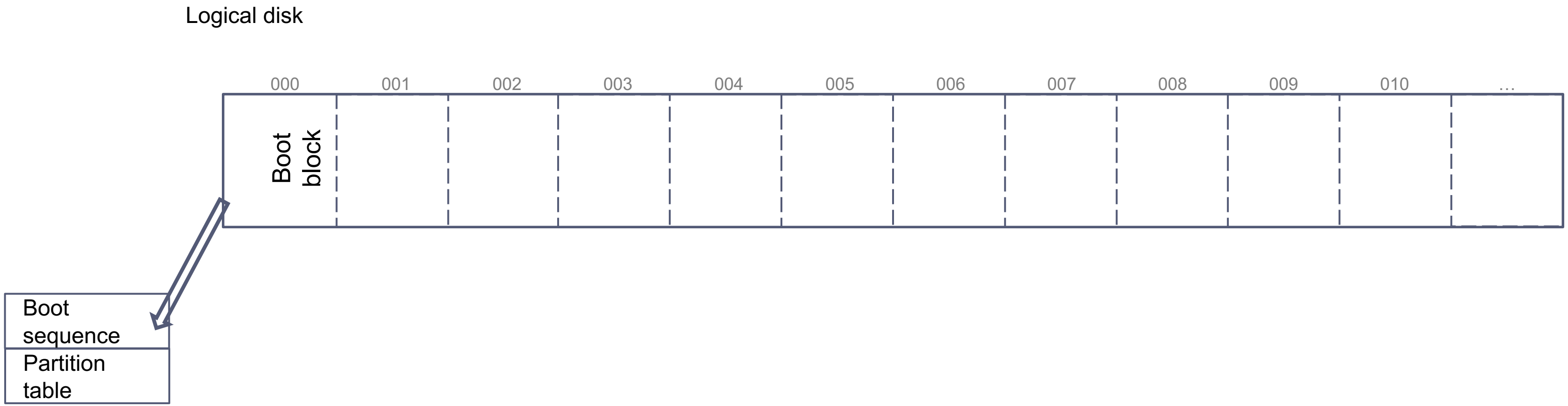
Logical disk



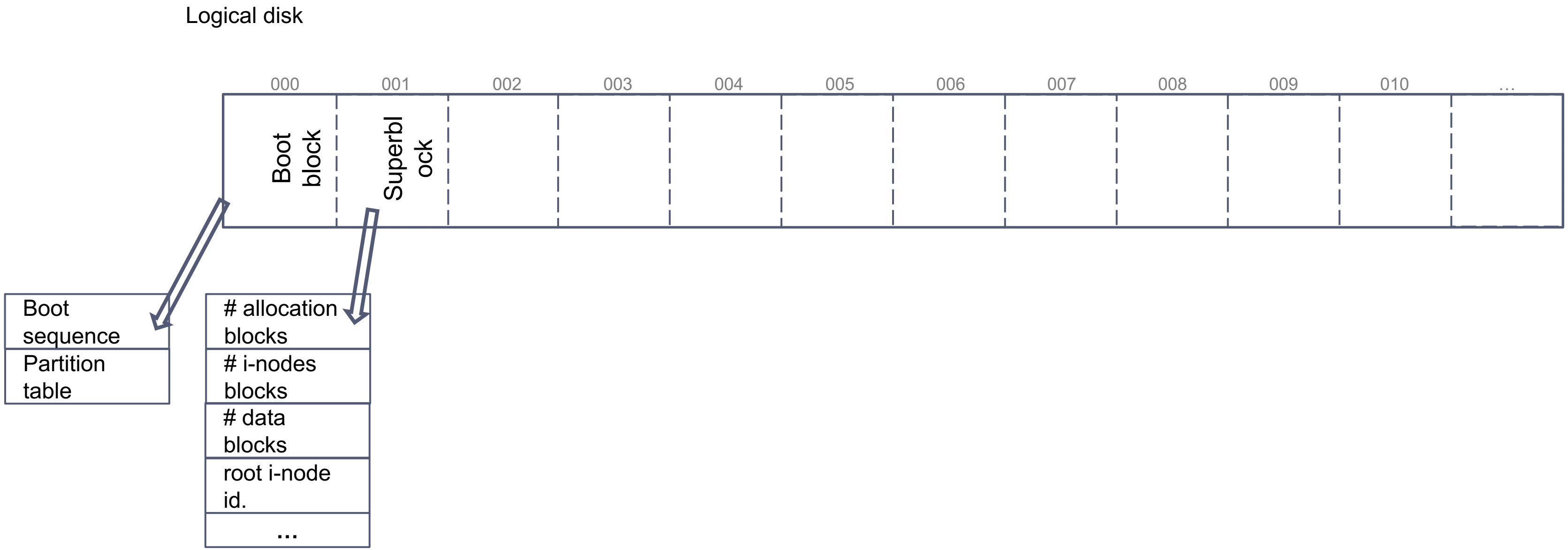
File system:
Unix-like representation



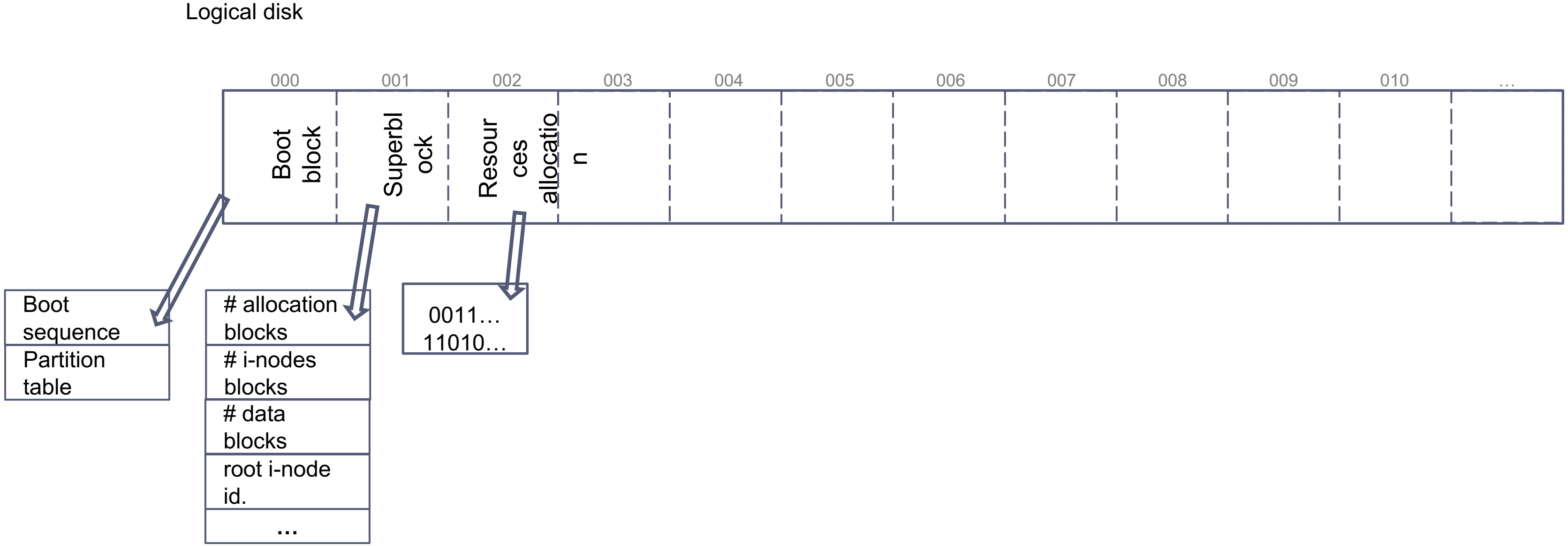
File system:
Unix-like representation



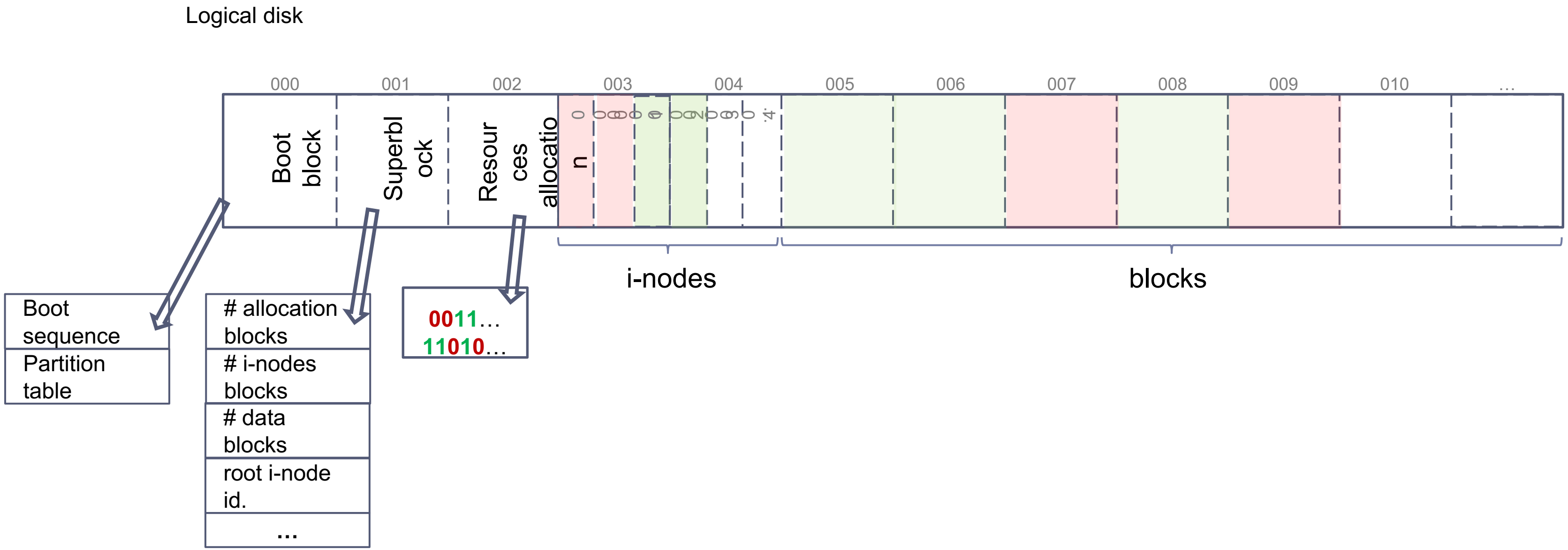
File system: Unix-like representation



File system:
Unix-like representation



File system:
Unix-like representation



Logical disk

000 001 002 003 004 005 006 007 008 009 010 …

Boot block

Superblock

Resources allocation

i-nodes

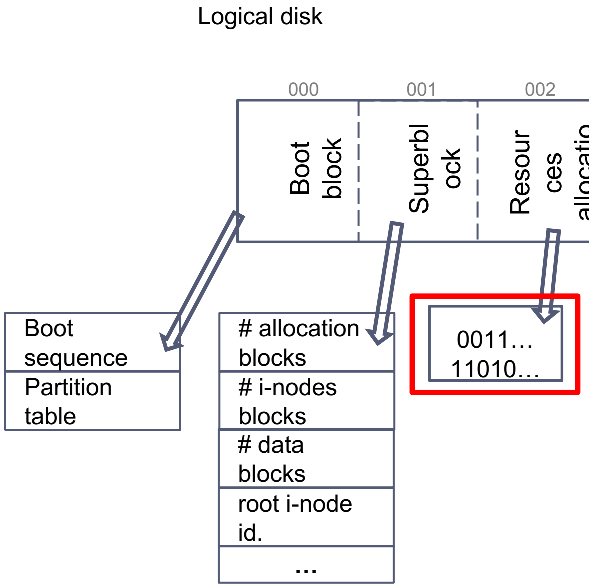
blocks

Boot sequence
Partition table

# allocation blocks
# i-nodes blocks
# data blocks
root i-node id.
…

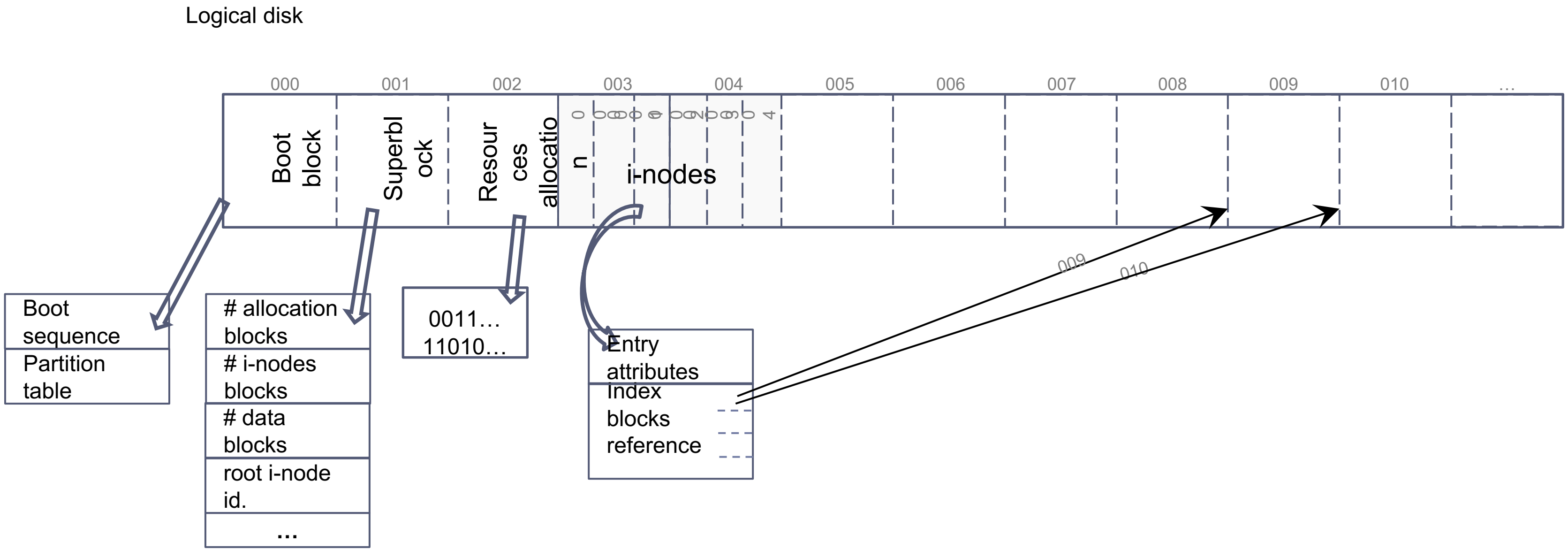
0011…
11010…

File system: Unix-like representation

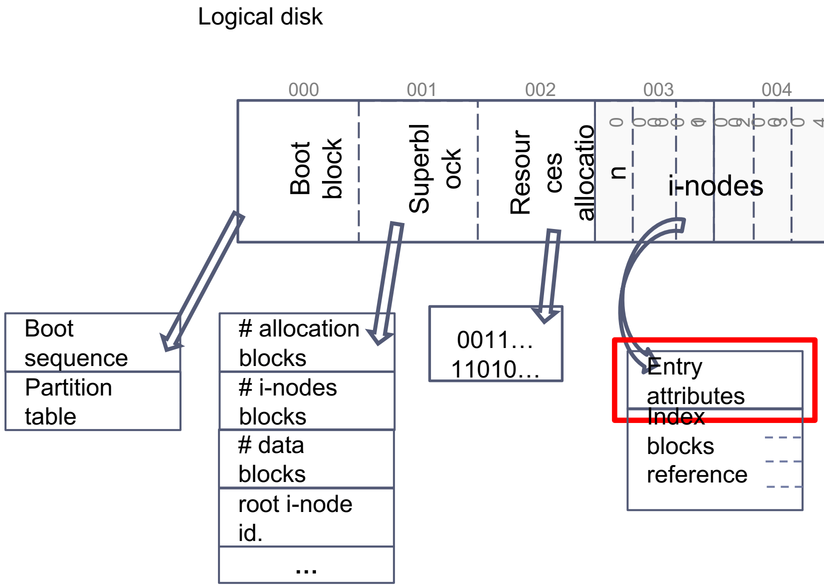


- Bit maps or bit vectors:
  - A vector with a bit per existing resource. If a resource is free then the bit value is 1, otherwise is 0.
    - Easy to implement and simple to be used.
    - Efficient if the device is not full or very fragmented.
- Free resources list:
  - It keeps a linked list all the available resources and to use a pointer to the first element in the list.
    - No very efficient method, unless for full devices or very fragmented devices.
- Indexing:
  - Index table with the identification of the free resources.

File system:
Unix-like representation

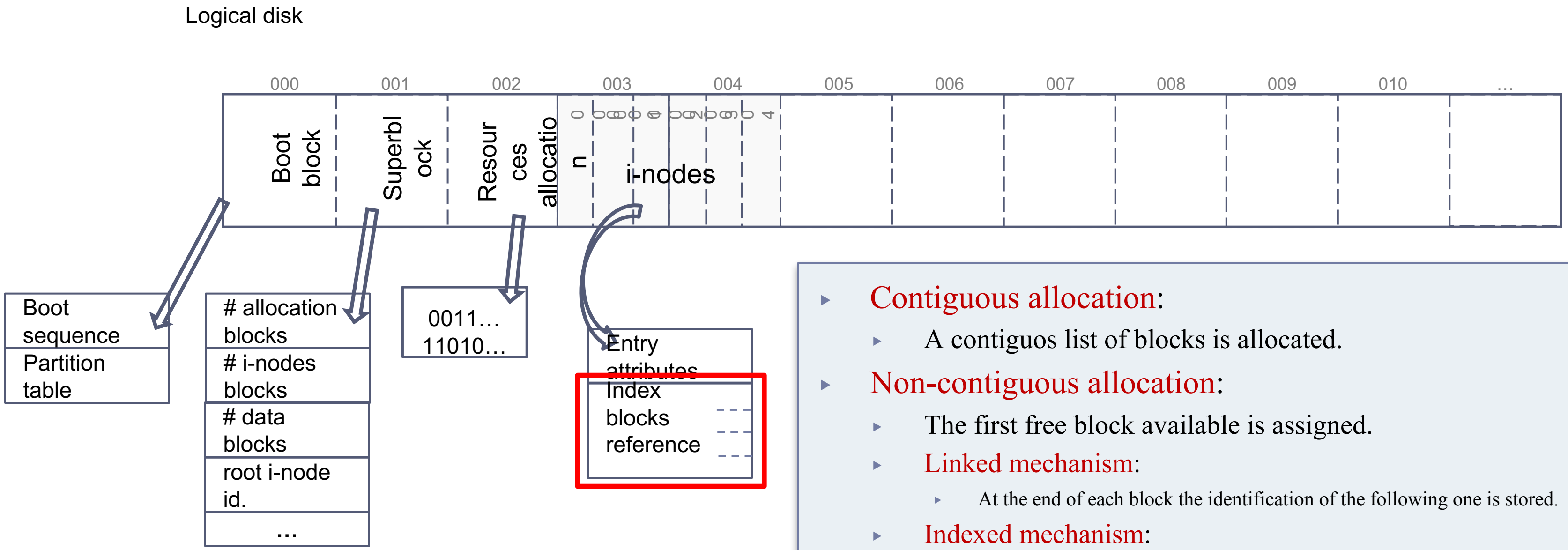


File system:
Unix-like representation



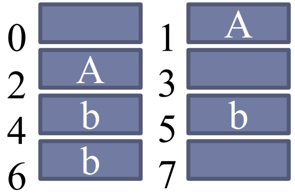
- Timestamp:
  - Creation, modification, last access, etc.
- Size:
  - Bytes or disk blocks used.
- Owner and protection:
  - Attributes, ACL, capacities, etc.
- Kind of file, link counter, etc.

File system:
Unix-like representation



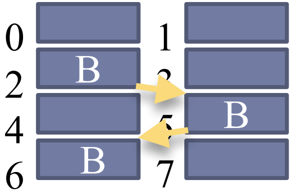
- Contiguous allocation:
  - A contiguos list of blocks is allocated.
- Non-contiguous allocation:
  - The first free block available is assigned.
  - Linked mechanism:
    - At the end of each block the identification of the following one is stored.
  - Indexed mechanism:
    - Blocks with the references of all file blocks.

File systems:
resources allocation alternatives



F	I	L
A	1	2
B	4	3

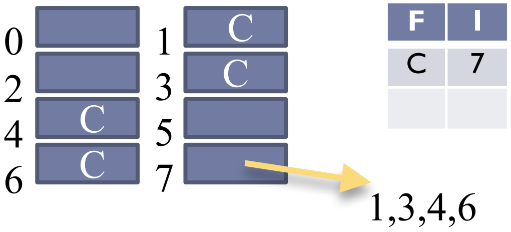
- **Contiguous allocation:**
  - The blocks of the files are contiguous.
  - It needs: first (I) and # of blocks (L)
  - To pack.



F	I	L
B	2	3

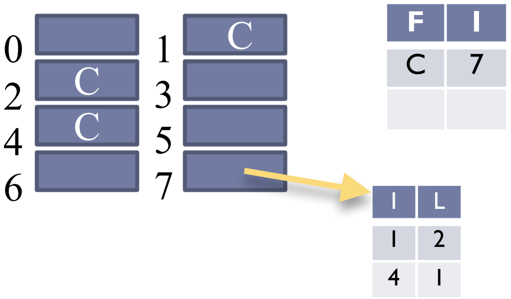
- **Non-contiguous allocation :**
  - Each block has the reference of the following one.
  - It needs: first (I) and # of blocks (L)
  - To defrag.

File systems:
resources allocation alternatives



F	I
C	7

- Indexed allocation (blocks):
  - Some blocks are used to store the reference list of file data blocks.
  - It needs: id. Of the first index block.
  - To defrag.

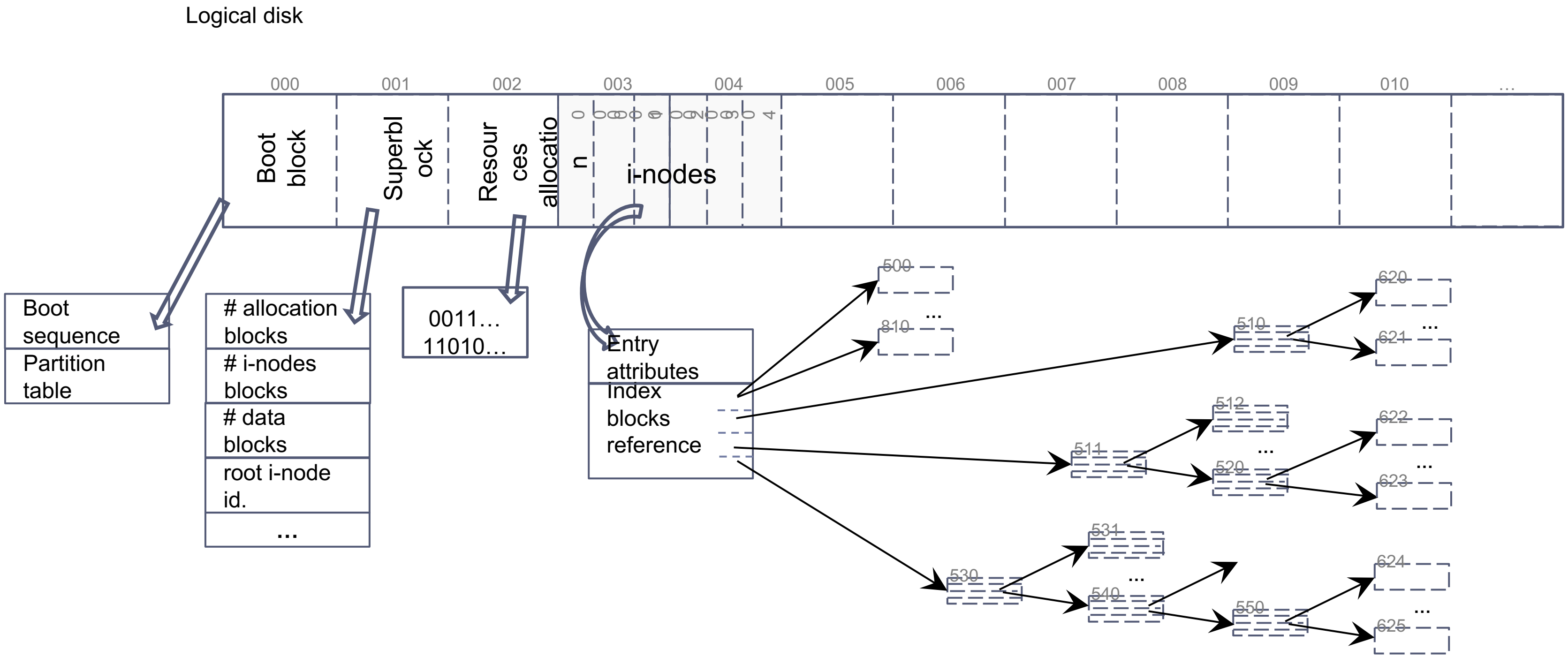


F	I
C	7

I	L
1	2
4	1

- Indexed allocation (extends):
  - Some blocks are used to store the reference list of continuous file data blocks sequences.
  - It needs: id. of the first index block.
  - To defrag.

File system:
Unix-like representation



Logical disk

Boot block

Superblock

Resources allocation

i-nodes

Boot sequence
Partition table

allocation blocks
i-nodes blocks
data blocks
root i-node id.
...

0011…
11010…

Entry attributes
Index blocks reference

How elements are represented

- Files

- Directories

- Links

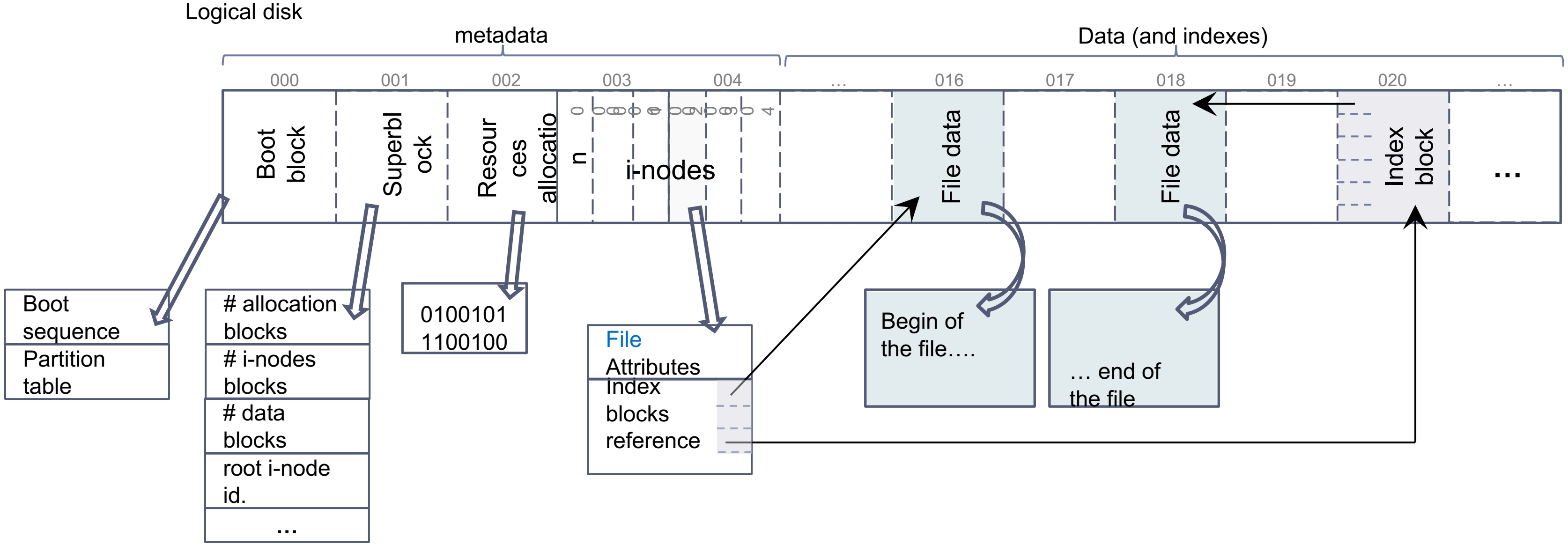
How elements are represented

- Files

- Directories

- Links

File system:
Unix-like representation: files

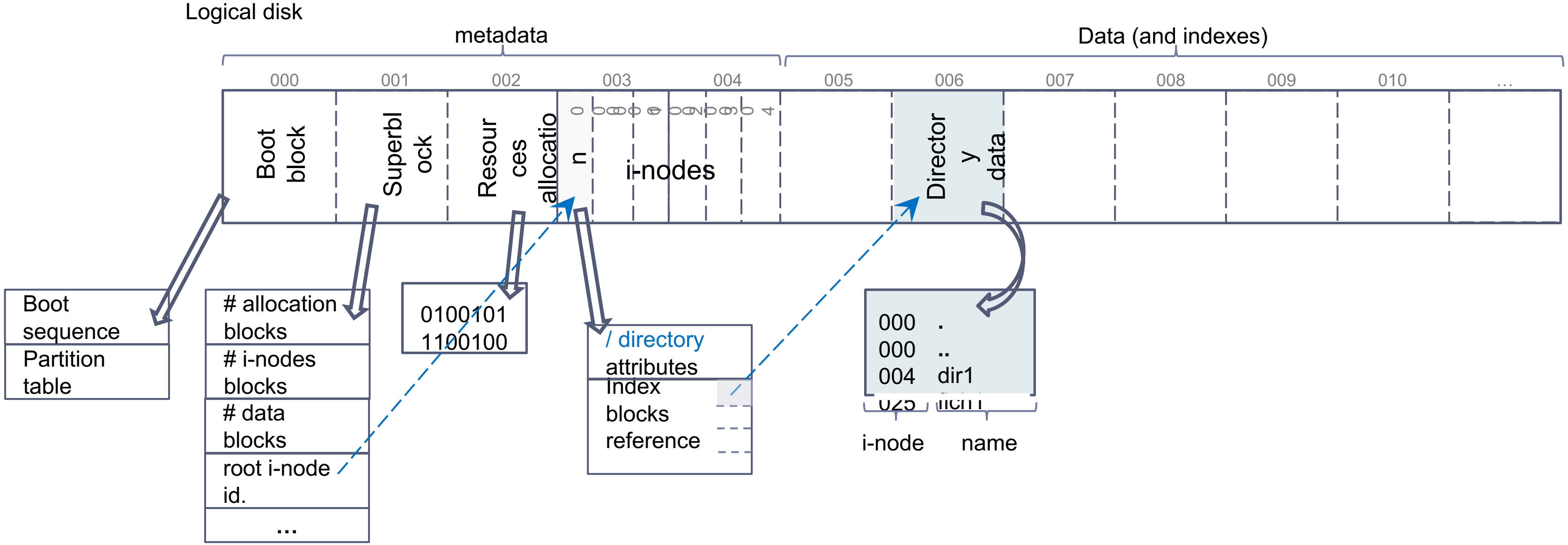


How elements are represented

- Files
- Directories
- Links

File system:

Unix-like representation: directories



Boot sequence

Partition table

# allocation blocks

# i-nodes blocks

# data blocks

root i-node id.

...

0100101
1100100

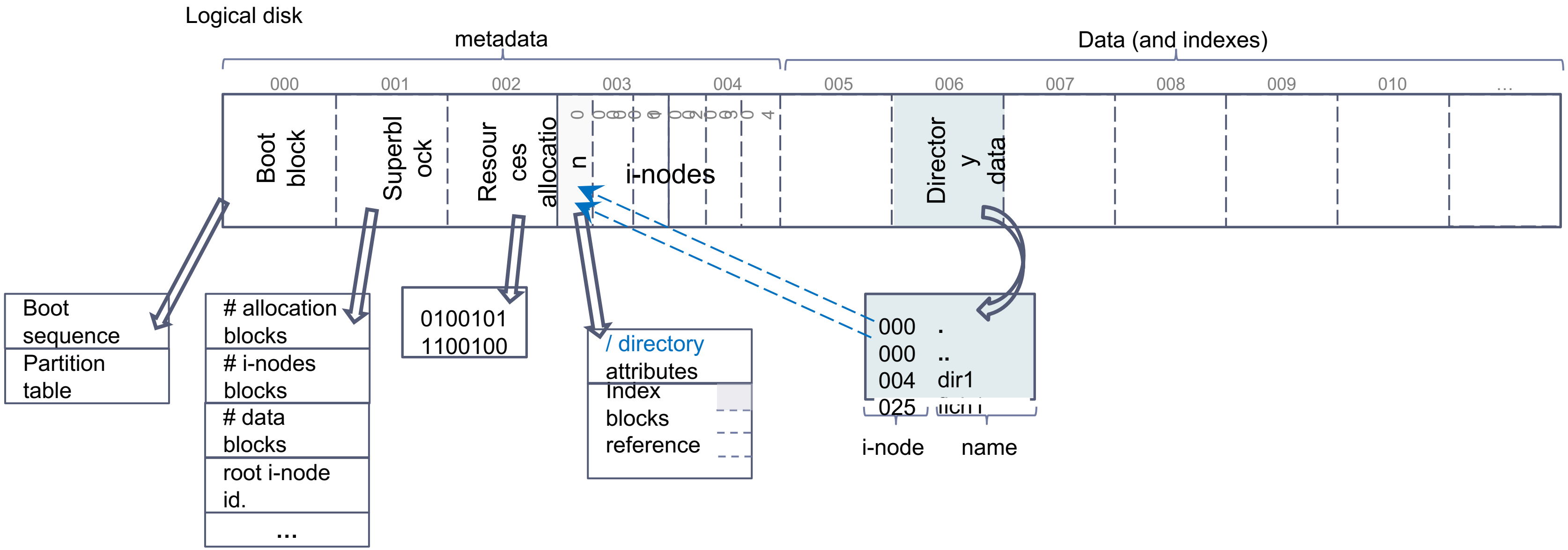
/ directory

attributes

Index blocks reference

i-node	name
000	.
000	..
004	dir1
025	fich1

File system:
Unix-like representation: directories



Logical disk

metadata

Data (and indexes)

000 001 002 003 004 005 006 007 008 009 010 …

Boot block

Superblock

Resources allocation

i-nodes

Directory data

Boot sequence

Partition table

# allocation blocks

# i-nodes blocks

# data blocks

root i-node id.

...

0100101
1100100

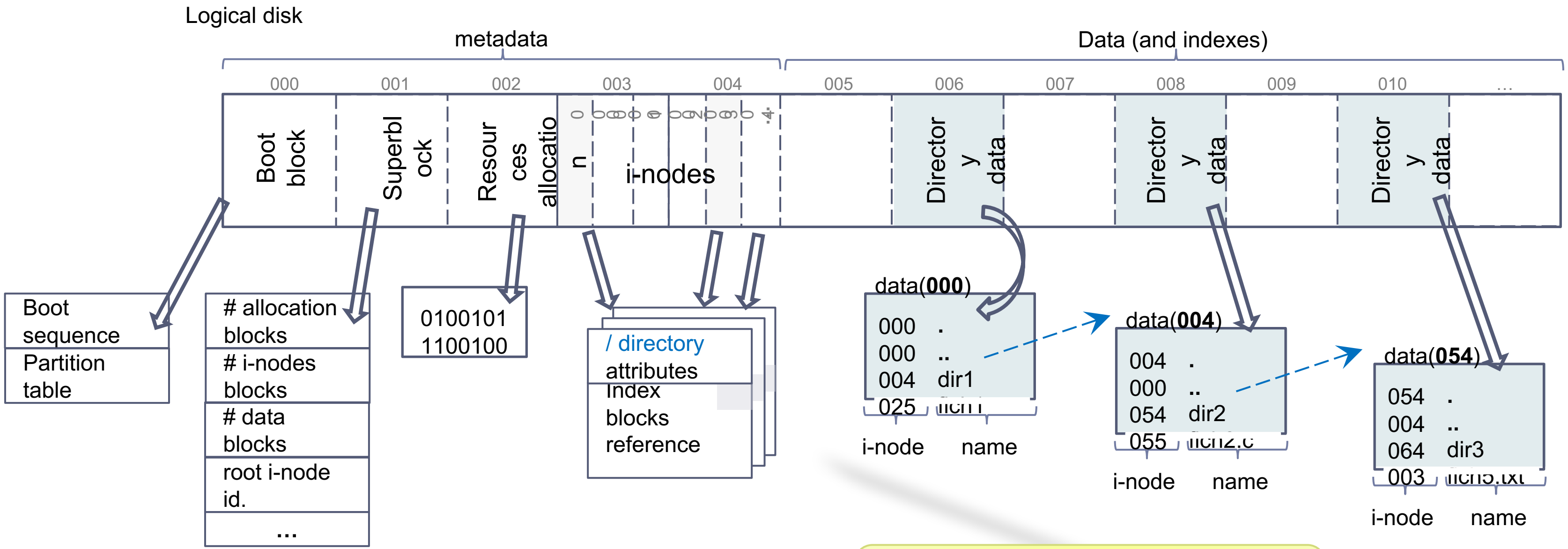
/ directory

attributes

Index blocks reference

i-node	name
000	.
000	..
004	dir1
025	fich1

File system:
Unix-like representation: directories



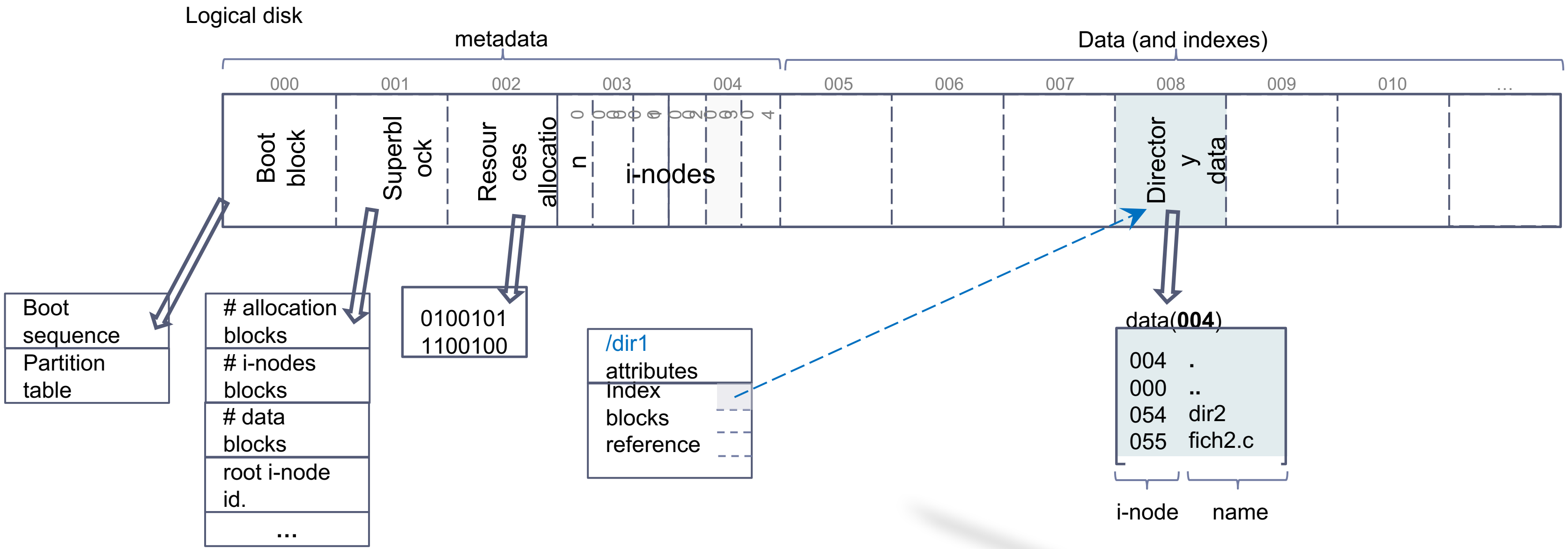
ls –l /dir1/dir2/fich5.txt

- / + dir1 + dir2 + fich5.txt
- 4 i-nodes + 3 data blocks

How elements are represented

- Files
- Directories
- Links

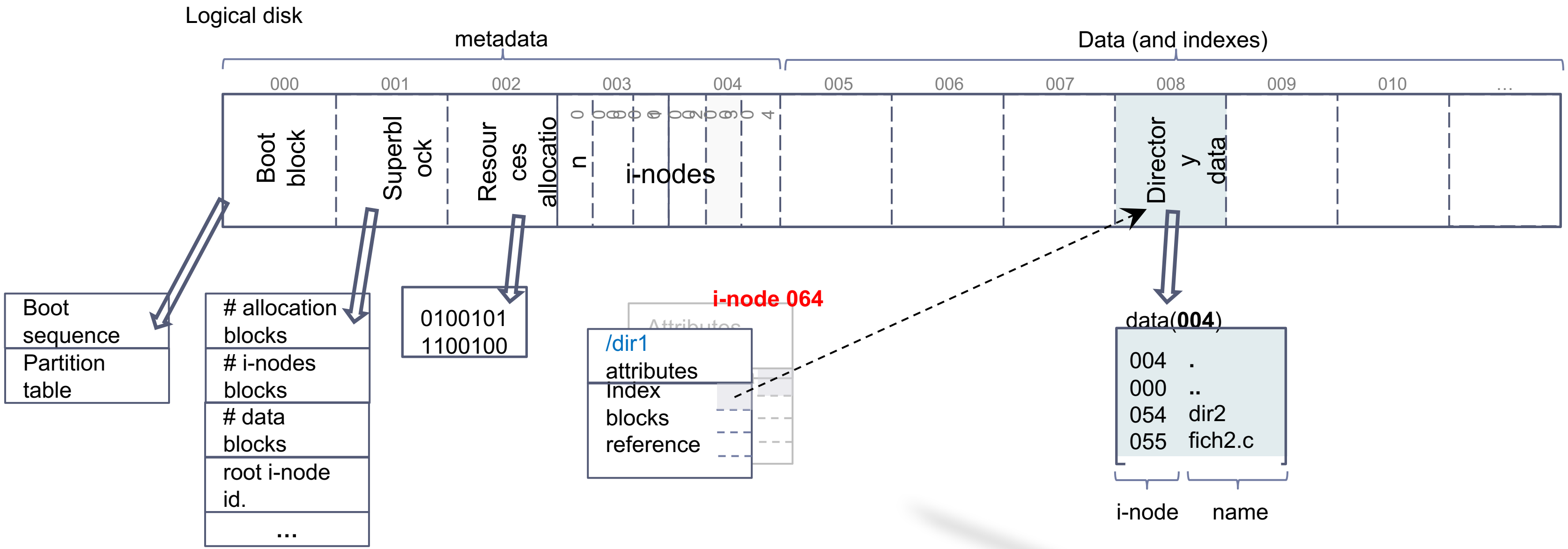
File system:
Unix-like representation: Symbolic link (soft link)



```
ln -s  /dir1  /dir1/soft
```

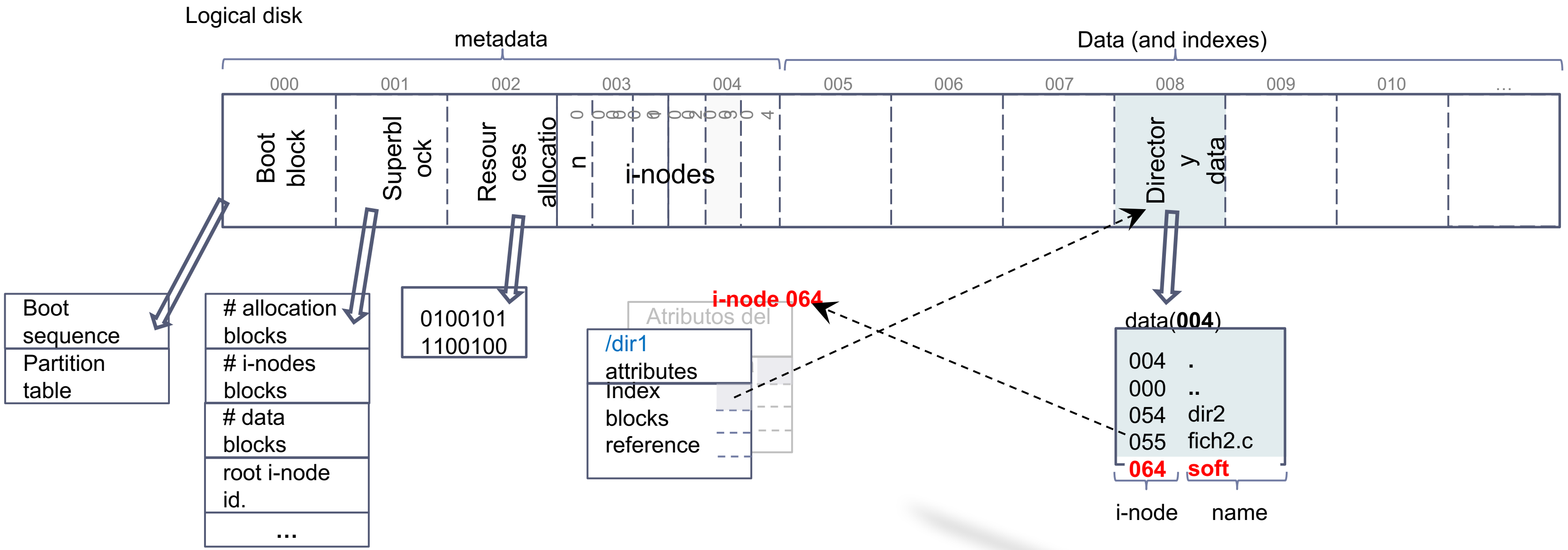
File system:

Unix-like representation: Symbolic link (soft link)



```
ln -s  /dir1  /dir1/soft
```

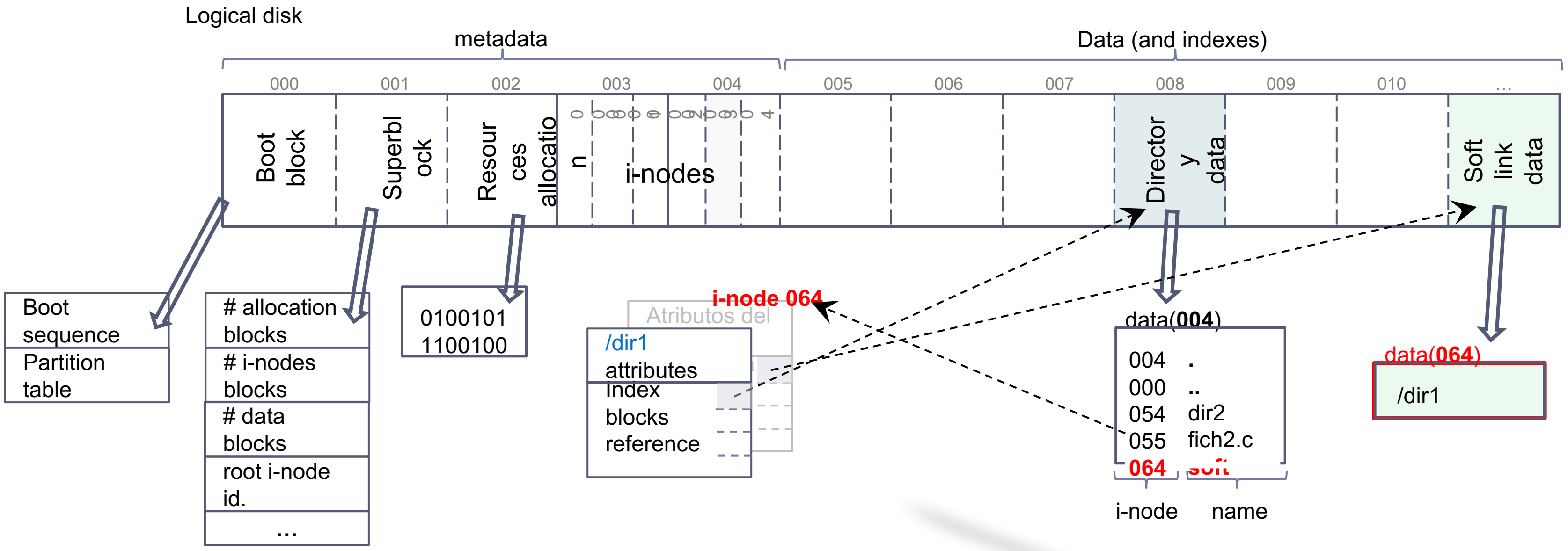
File system:
Unix-like representation: Symbolic link (soft link)



```
ln -s  /dir1  /dir1/soft
```

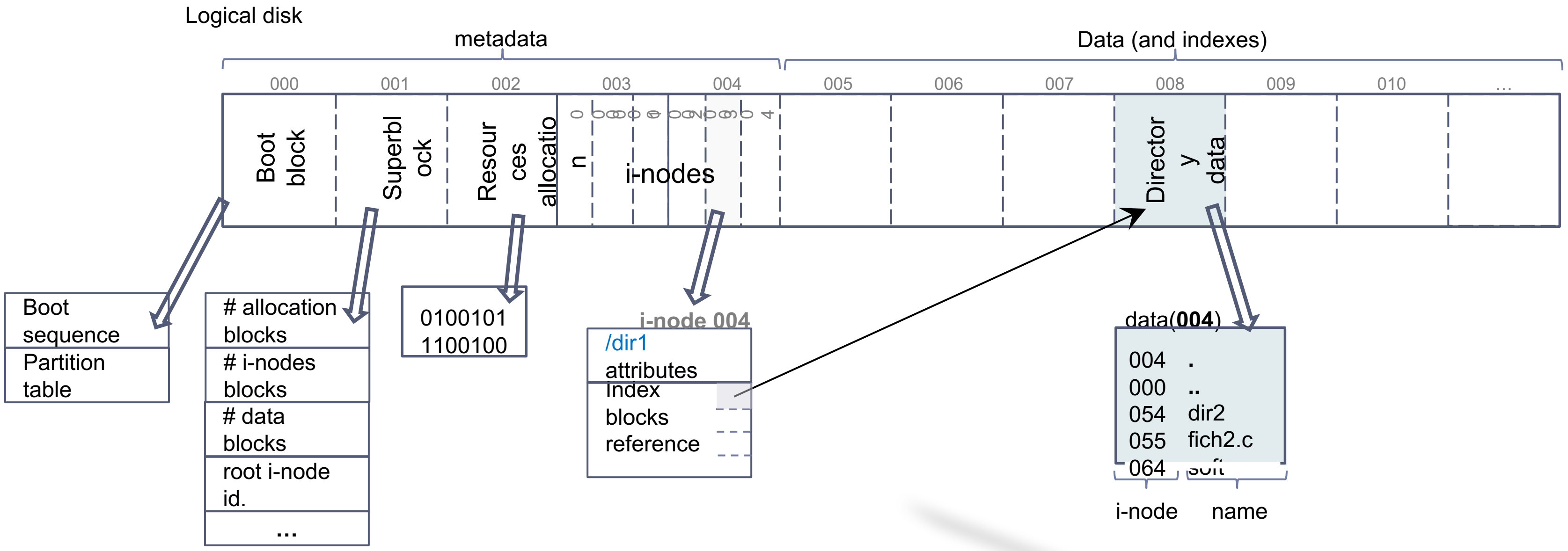
File system:

Unix-like representation: Symbolic link (soft link)



```
ln -s  /dir1  /dir1/soft
```

File system:
Unix-like representation: hard link



Logical disk

metadata

Data (and indexes)

000 001 002 003 004 005 006 007 008 009 010 …

Boot block

Superblock

Resources allocation

i-nodes

Directory data

Boot sequence
Partition table

# allocation blocks
# i-nodes blocks
# data blocks
root i-node id.
...

0100101
1100100

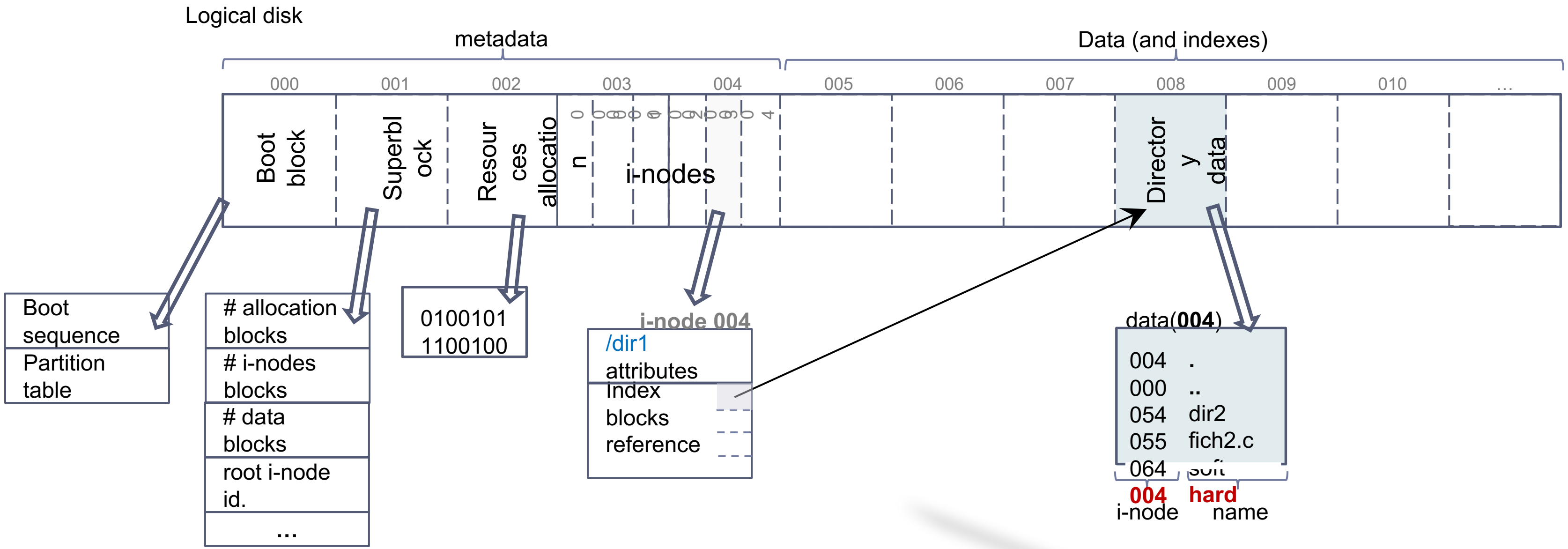
i-node 004
/dir1
attributes
Index blocks reference

data(**004**)

i-node	name
004	.
000	..
054	dir2
055	fich2.c
064	soft

```
ln    /dir1  /dir1/hard
```

File system:
Unix-like representation: hard link



Logical disk

metadata

Data (and indexes)

000 001 002 003 004 005 006 007 008 009 010 ...

Boot block | Superblock | Resources allocation | i-nodes | Directory data

Boot sequence
Partition table

# allocation blocks
# i-nodes blocks
# data blocks
root i-node id.
...

0100101
1100100

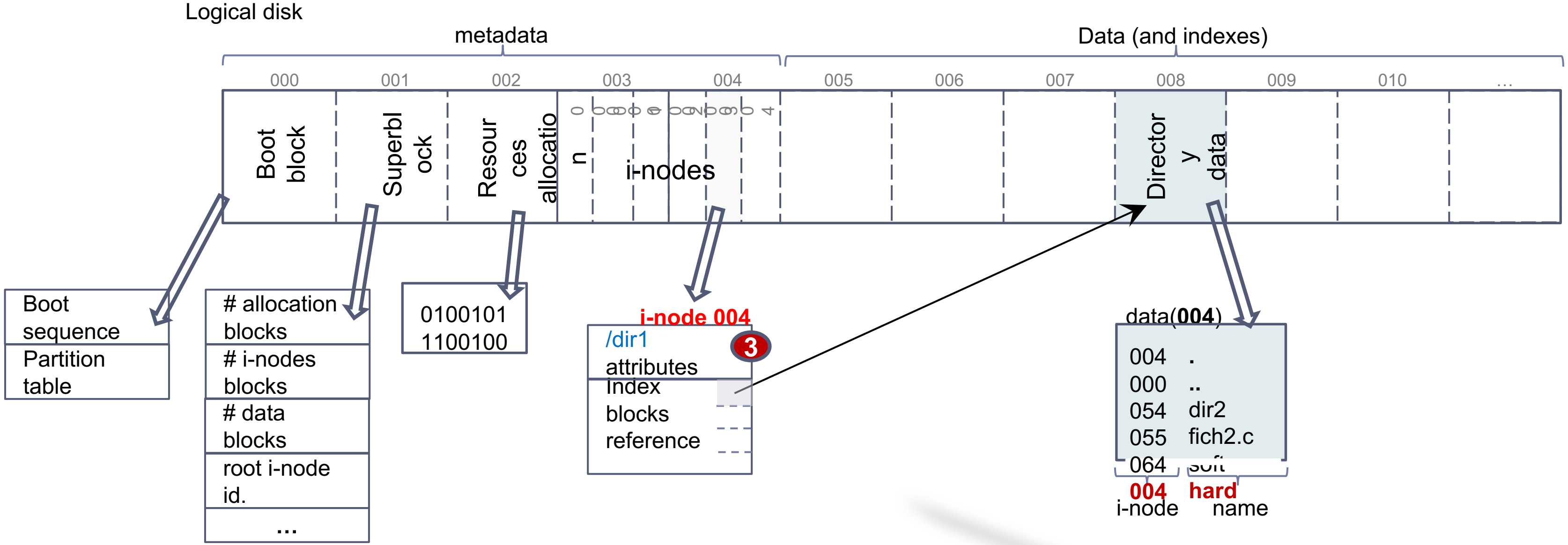
i-node 004
/dir1
attributes
Index blocks reference

data(**004**)

i-node	name
004	.
000	..
054	dir2
055	fich2.c
064	soft
004	**hard**

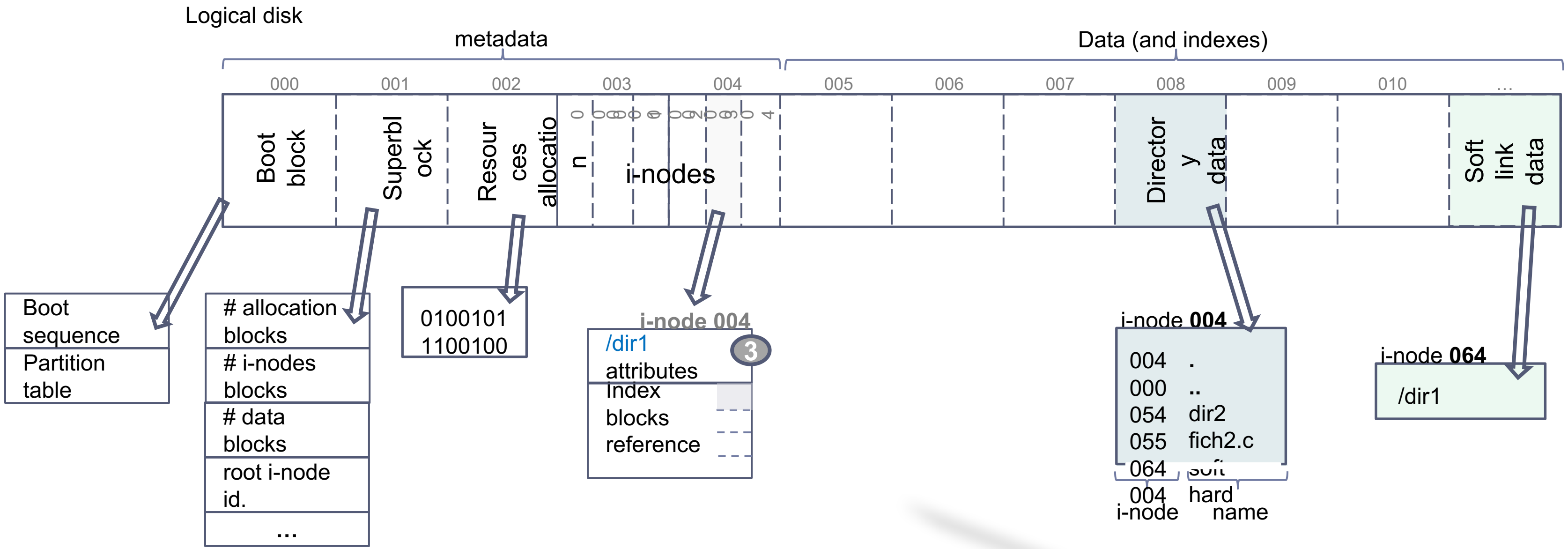
```
ln   /dir1  /dir1/hard
```

File system:
Unix-like representation: hard link



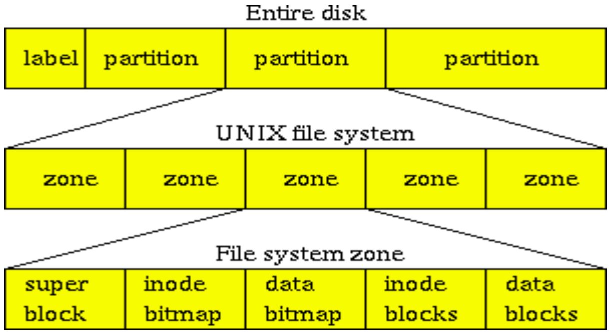
```
ln    /dir1  /dir1/hard
```

File system:
hard link vs soft link



```
ln -s  /dir1  /dir1/soft
ln     /dir1  /dir1/hard
```

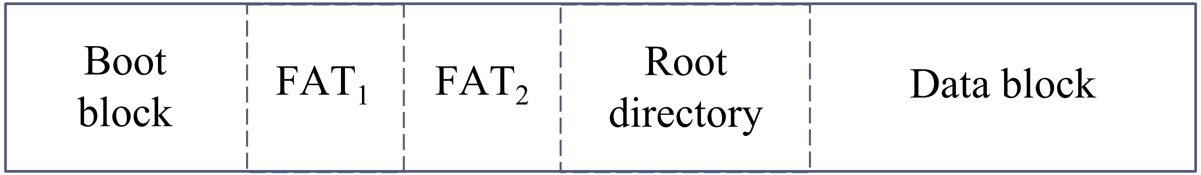
File system structures



- UNIX/Linux
- FAT

File sytem structures:

FAT



Boot block

FAT_1

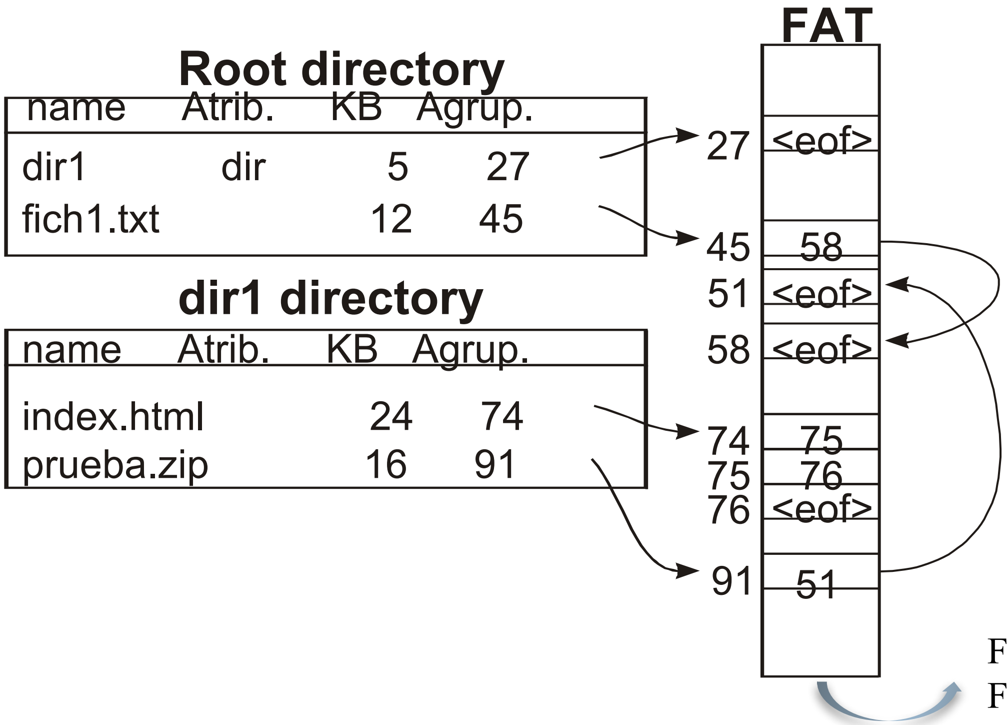
FAT_2

Root directory

Data block

Files and directories representation:

FAT



Root directory

name	Atrib.	KB	Agrup.
dir1	dir	5	27
fich1.txt		12	45

dir1 directory

name	Atrib.	KB	Agrup.
index.html		24	74
prueba.zip		16	91

FAT

27	<eof>
45	58
51	<eof>
58	<eof>
74	75
75	76
76	<eof>
91	51

FAT 12 bits (2^{12} blocks)

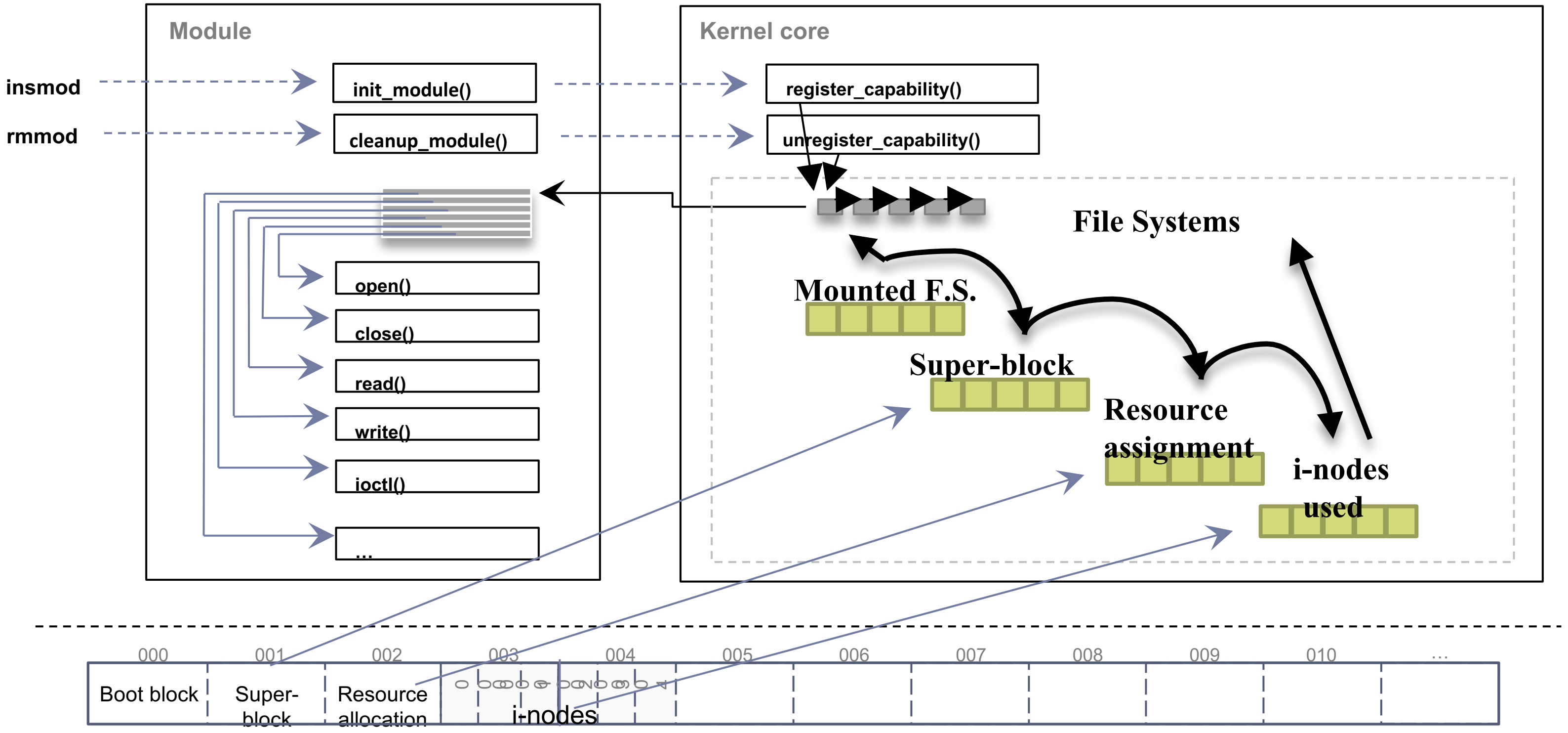
FAT 16 bits (2^{16} blocks)

FAT 32 bits (2^{32} blocks)

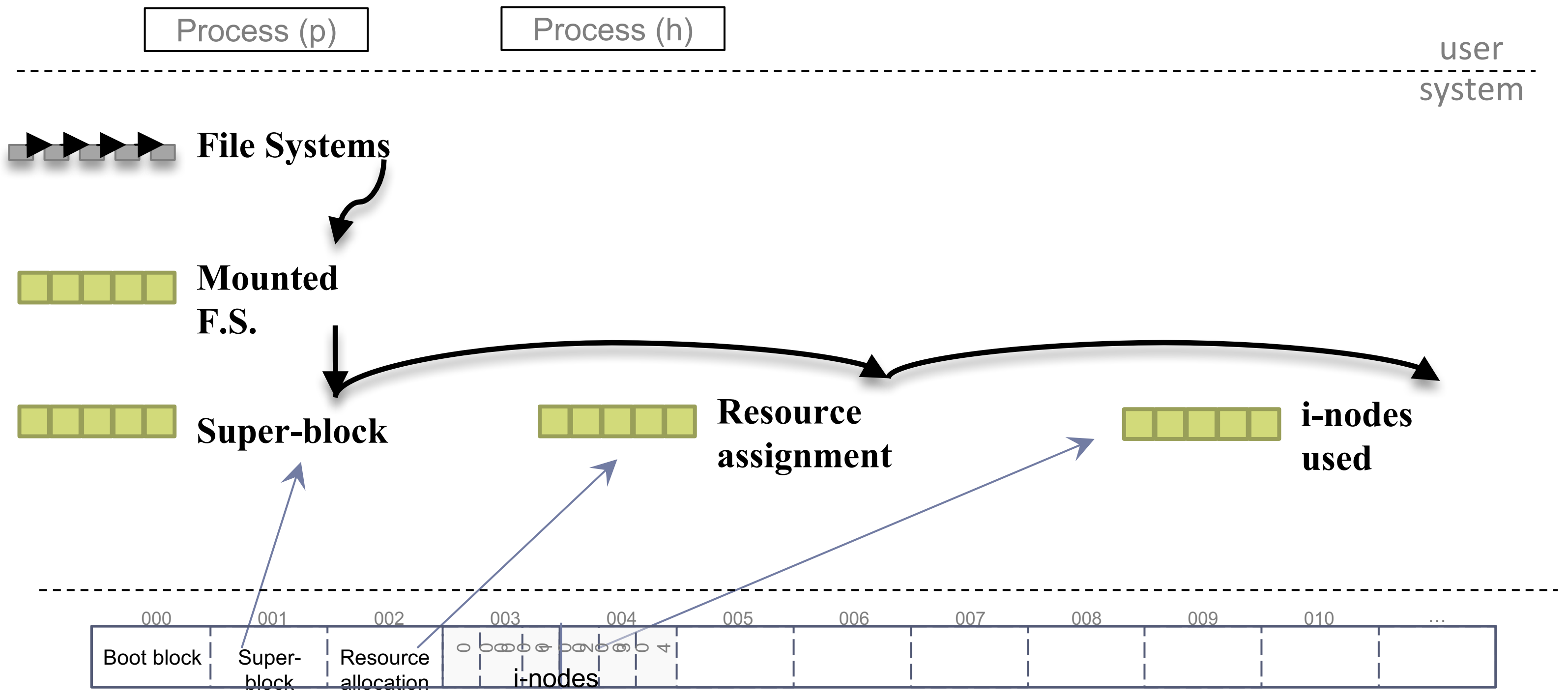
Overview

1. Introduction
2. Main data structures on the secondary memory
3. **Main data structures in the main memory**
4. Block management
5. Complementary aspects

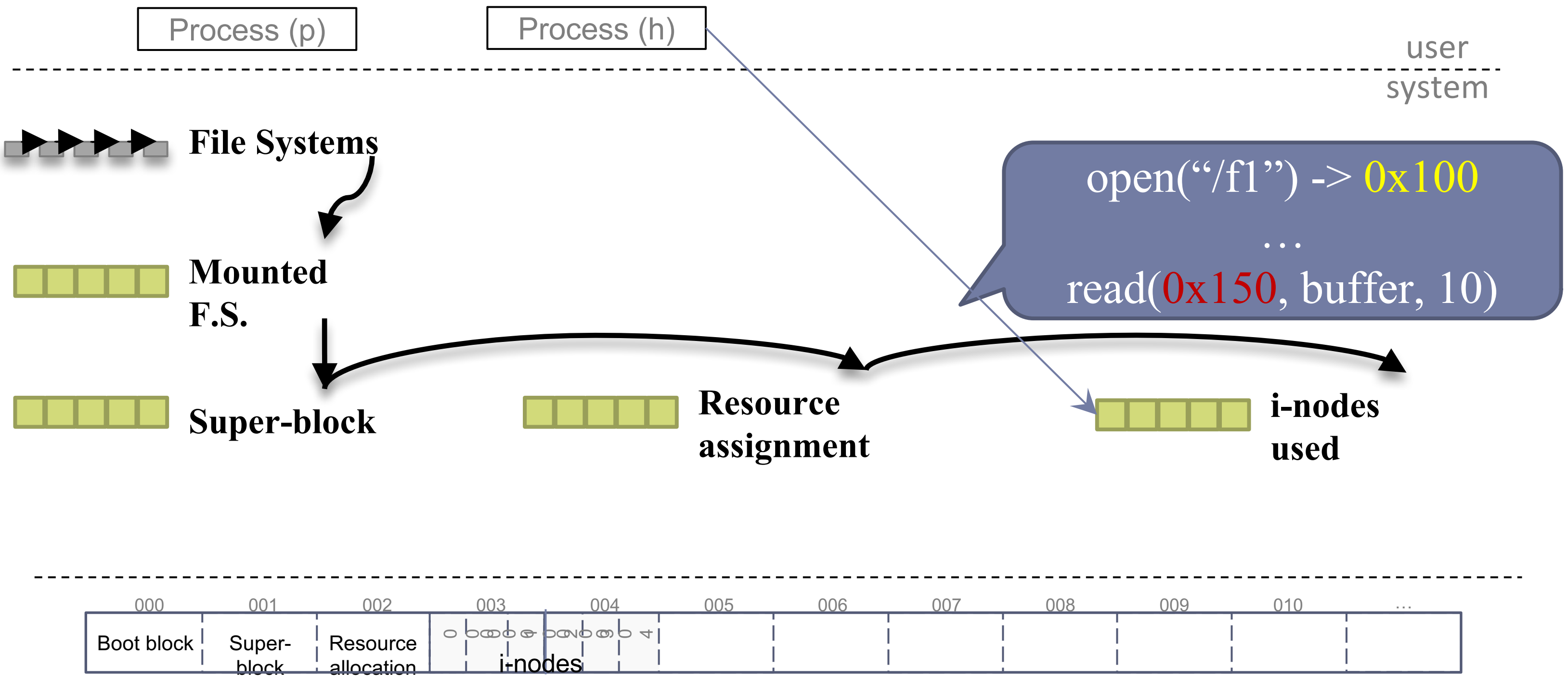
Initial design…



Initial design…



Initial design…



Process (p)

Process (h)

user

system

File Systems

Mounted F.S.

Super-block

Resource assignment

i-nodes used

open("/f1") -> 0x100

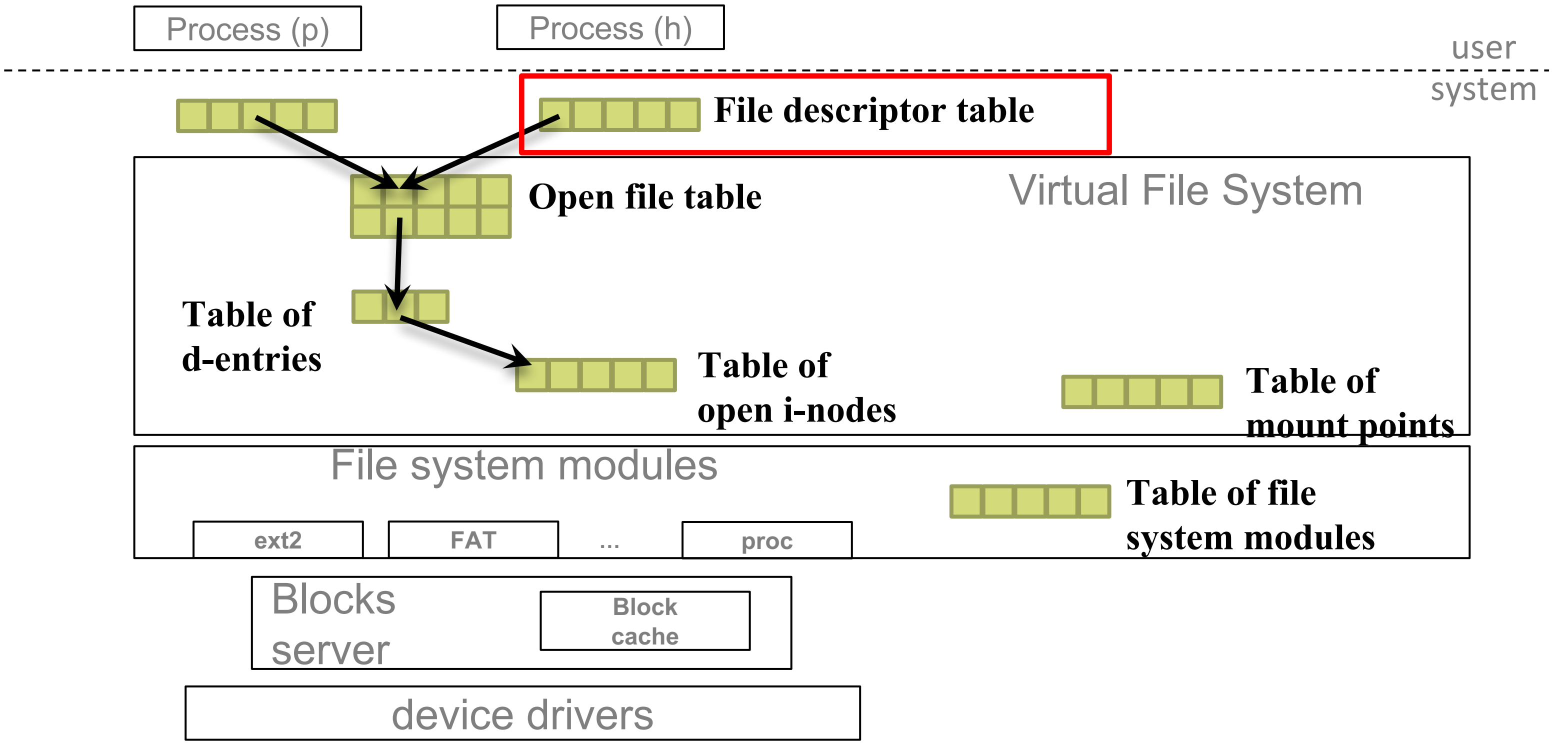
…

read(0x150, buffer, 10)

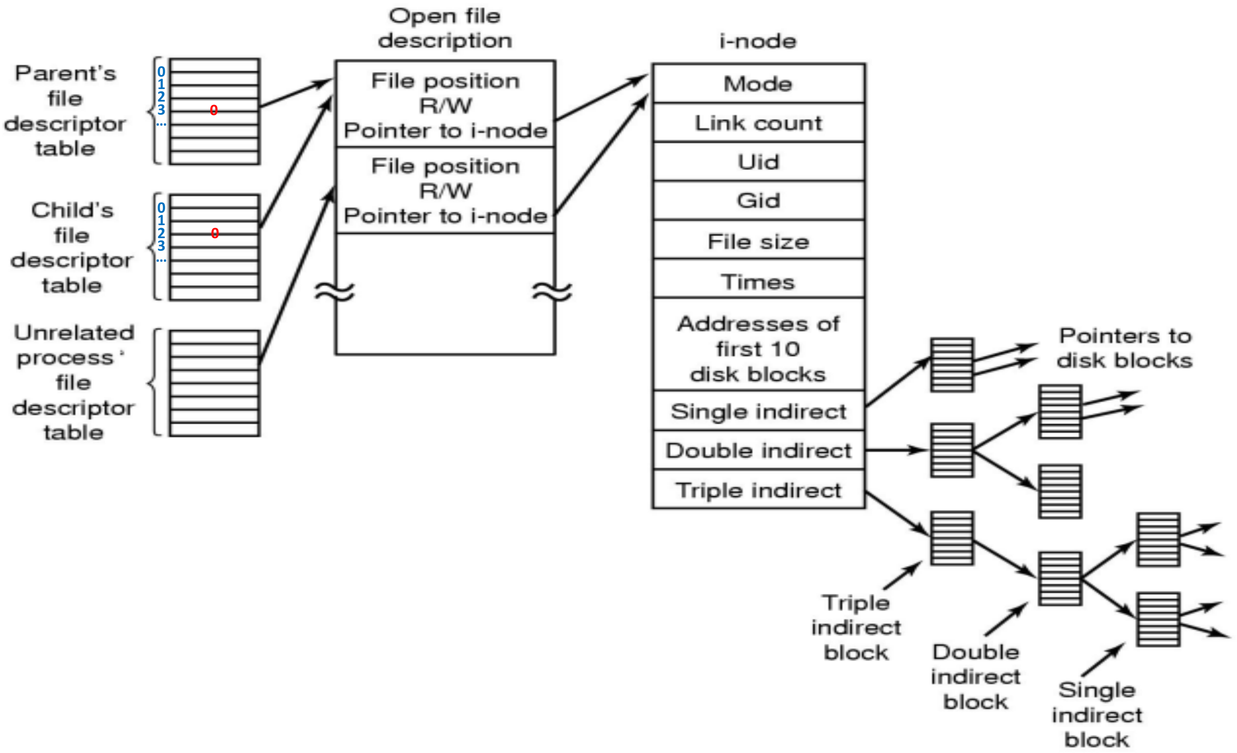
Main goals
(for a Unix-like file system)

- Processes have to use a secure interface, without direct access to the kernel data structures.
- Share the file offset position among processes from the same parent that open the file.
- Offer functionality for working with a file/directory in order to update the information that it contains.
- Go back and forth in the file system directory tree.
- Offer persistency of user data, seeking to minimize the impact on the performance and the space needed for the metadata.
- Keep track of the file systems registered in the kernel, and keep track of the mount point of these file systems.

Main management structures



Main management structures



Main management structures

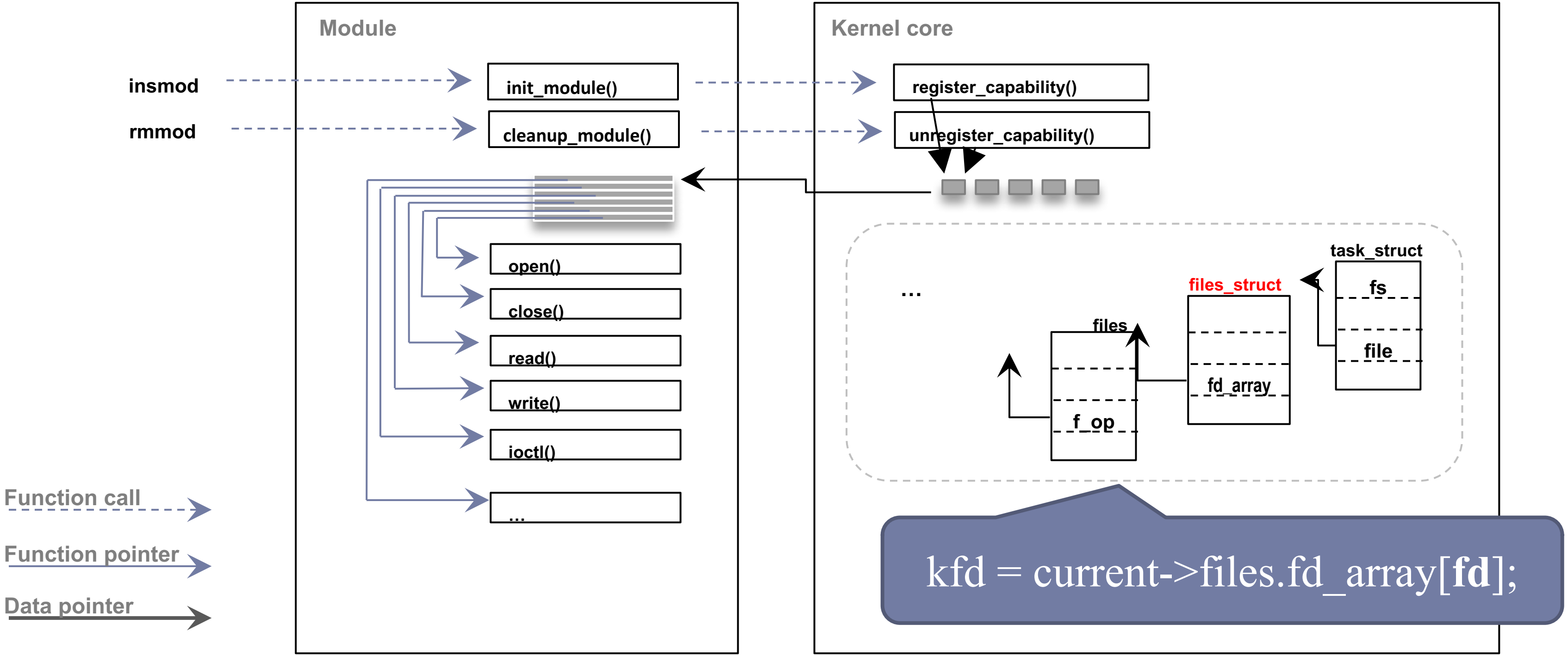
File descriptor table: Linux

```
struct fs_struct {
    atomic_t    count;              /* structure's usage count */
    spinlock_t  file_lock;          /* lock protecting this structure */
    int         max_fds;            /* maximum number of file objects */
    int         max_fdset;          /* maximum number of file descriptors */
    int         next_fd;            /* next file descriptor number */
    struct file **fd;               /* array of all file objects */
    fd_set      *close_on_exec;     /* file descriptors to close on exec() */
    fd_set      *open_fds;          /* pointer to open file descriptors */
    fd_set       close_on_exec_init; /* initial files to close on exec() */
    fd_set       open_fds_init;     /* initial set of file descriptors */
    struct file  *fd_array[NR_OPEN_DEFAULT]; /* array of all file objects */
};
```

http://www.makelinux.net/books/lkd2/ch12lev1sec10

Main management structures

Descriptors table (open files): Linux



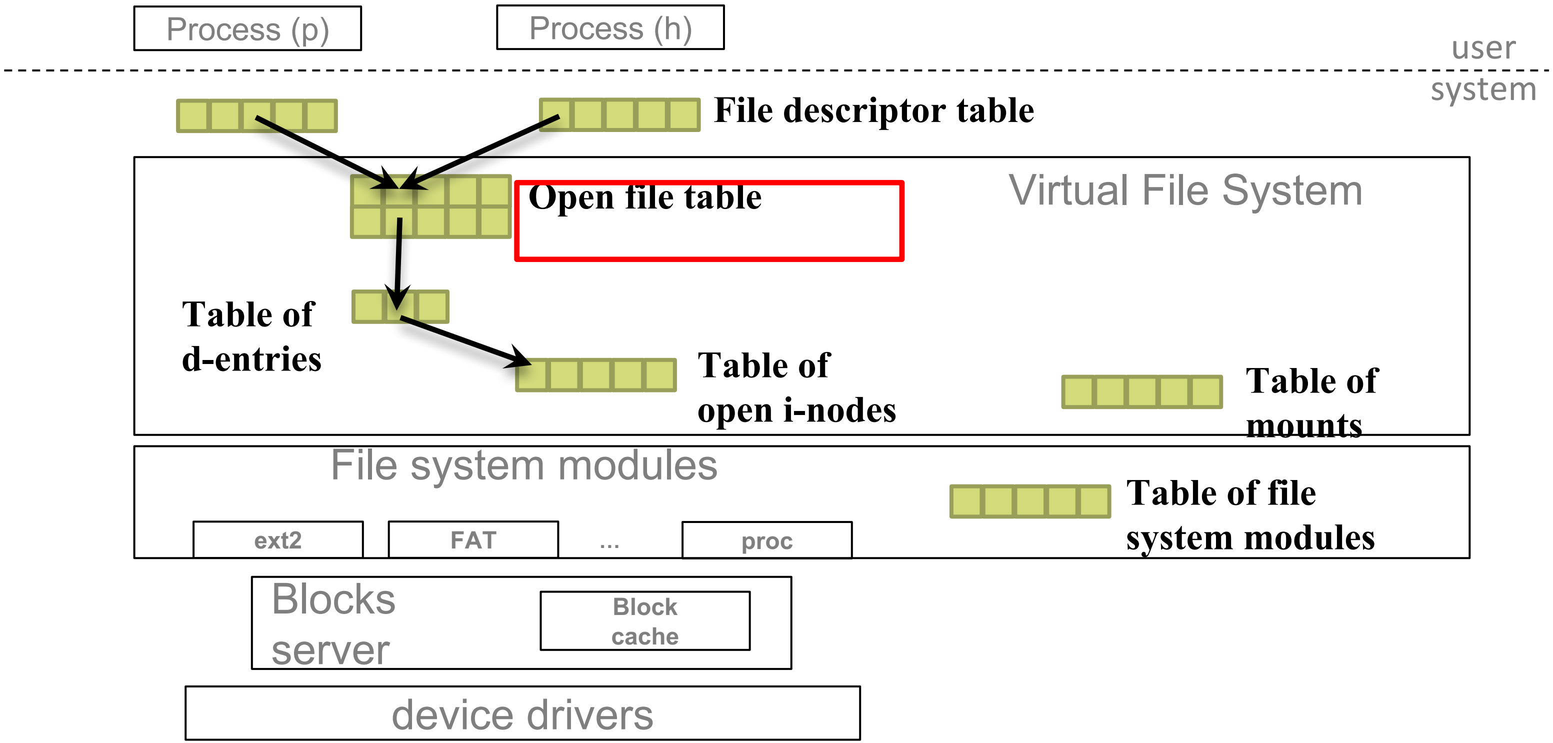
kfd = current->files.fd_array[**fd**];

http://www.xml.com/ldd/chapter/book/ch02.html

Main goals
(for a Unix-like file system)

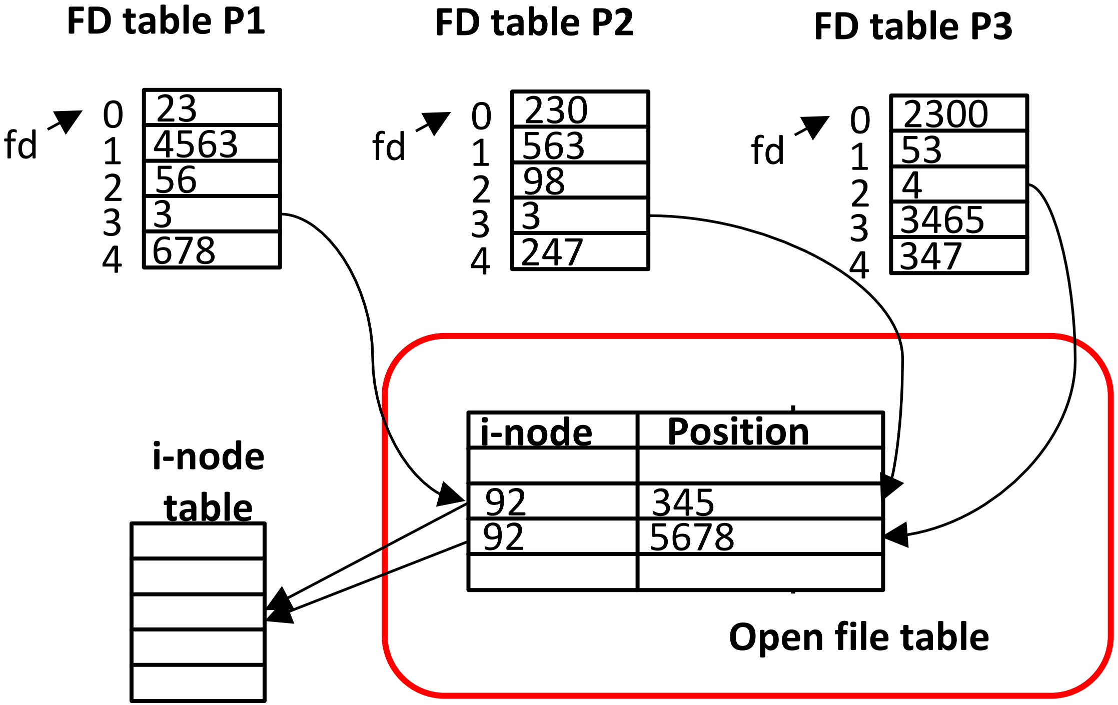
- The processes have to use a secure interface, without direct access to the kernel representation.
- To share the file offset among process from the same parent that open the file.
- To have a working session with the file/directory in order to update the information that it contains.
- Go back and forth in the file system directory tree.
- Offer persistency of user data, seeking to minimize the impact on the performance and the space needed for the metadata.
- Keep track of the file system registered in the kernel, and keep track of the mount points of these file systems.

Main management structures



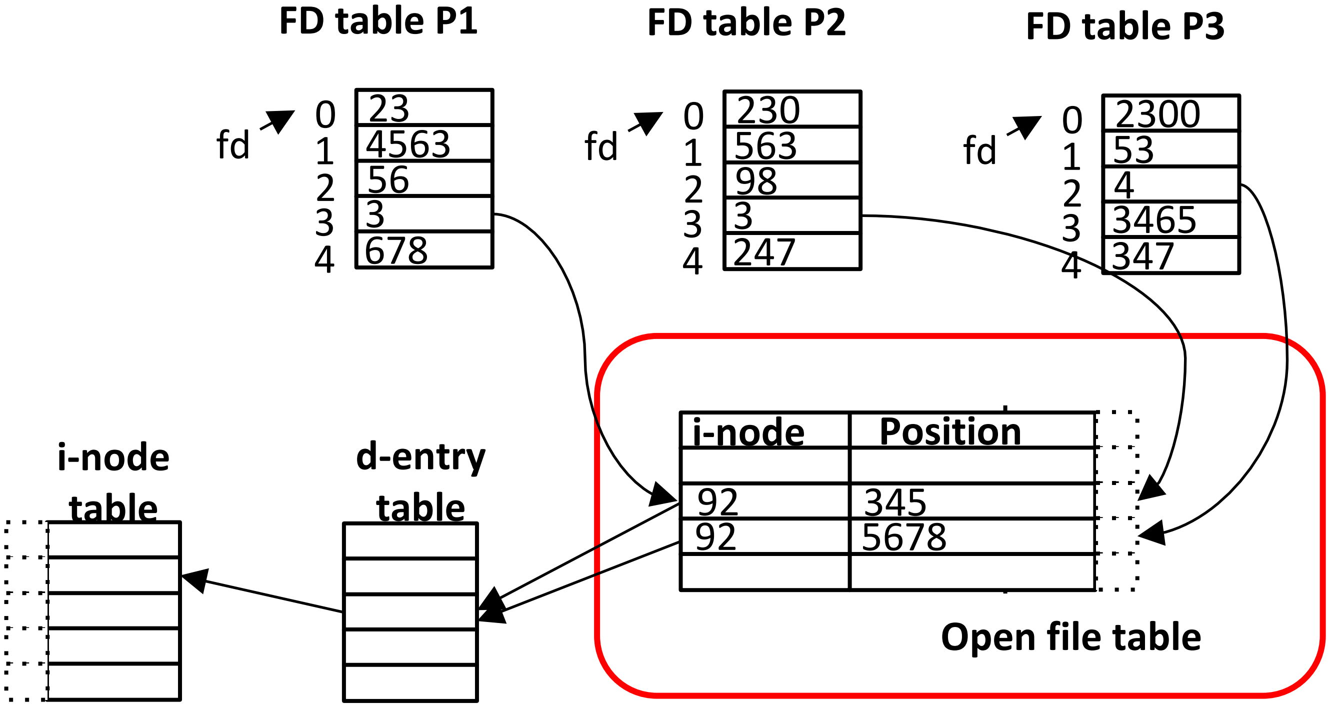
Main management structures

Seek pointers table



Main management structures

Seek pointers table: Linux



Main management structures

File table: Linux

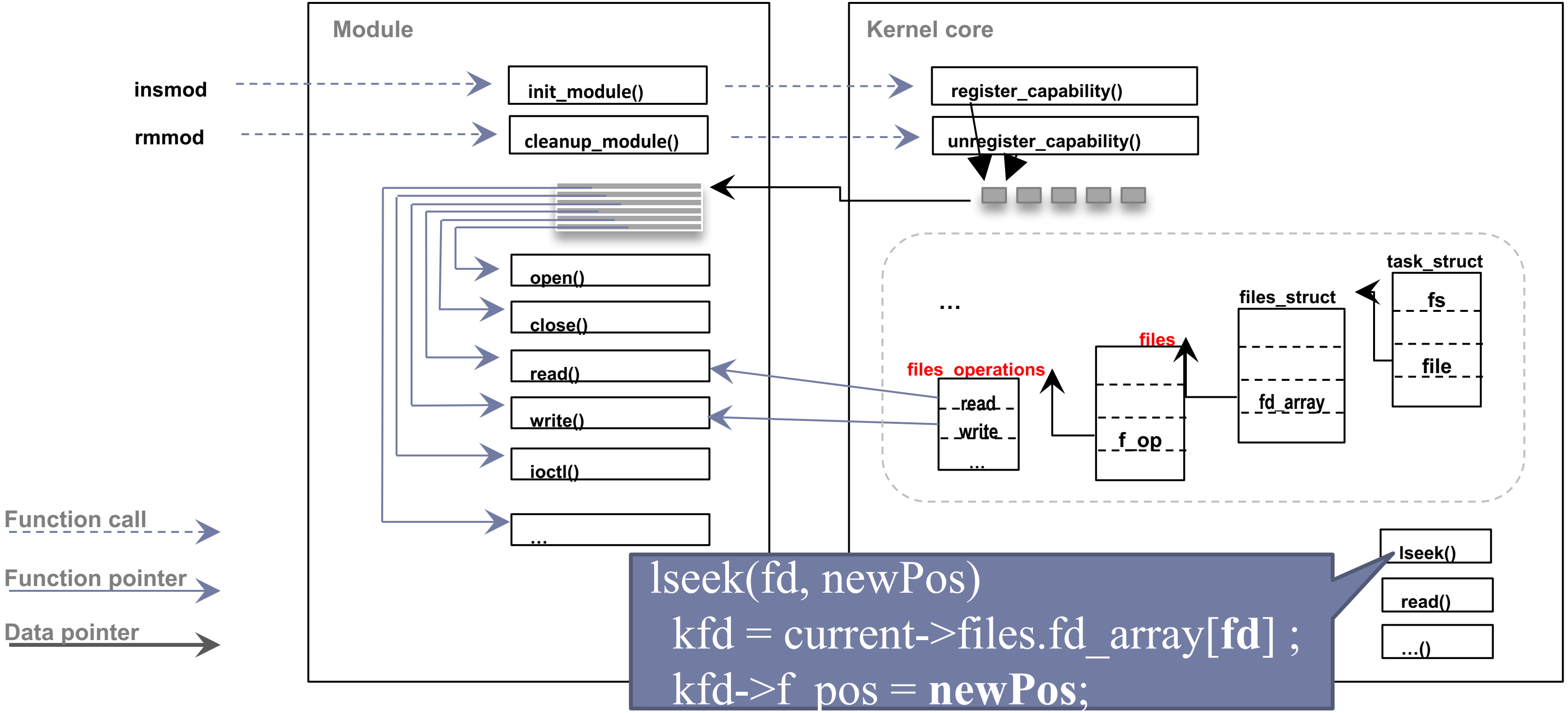
```
struct file {
        struct dentry             *f_dentry;
        struct vfsmount           *f_vfsmnt;
        struct file_operations    *f_op;
        mode_t                    f_mode;
        loff_t                    f_pos;
        struct fown_struct        f_owner;
        unsigned int              f_uid, f_gid;
        unsigned long             f_version;
        …
} ;
```

```
struct file_operations  {
    int      (*open)    (struct inode *, struct file *);
    ssize_t (*read)     (struct file *, char *, size_t, loff_t *);
    ssize_t (*write)    (struct file *, const char *, size_t, loff_t *);
    loff_t   (*llseek)  (struct file *, loff_t, int);
    int      (*ioctl)   (struct inode *, struct file *,
                         unsigned int, ulong);
    int      (*readdir) (struct file *, void *, filldir_t);
    int      (*mmap)    (struct file *, struct vm_area_struct *);
    …
};
```

http://www.win.tue.nl/~aeb/linux/lk/lk-8.html

Main management structures

File table: Linux

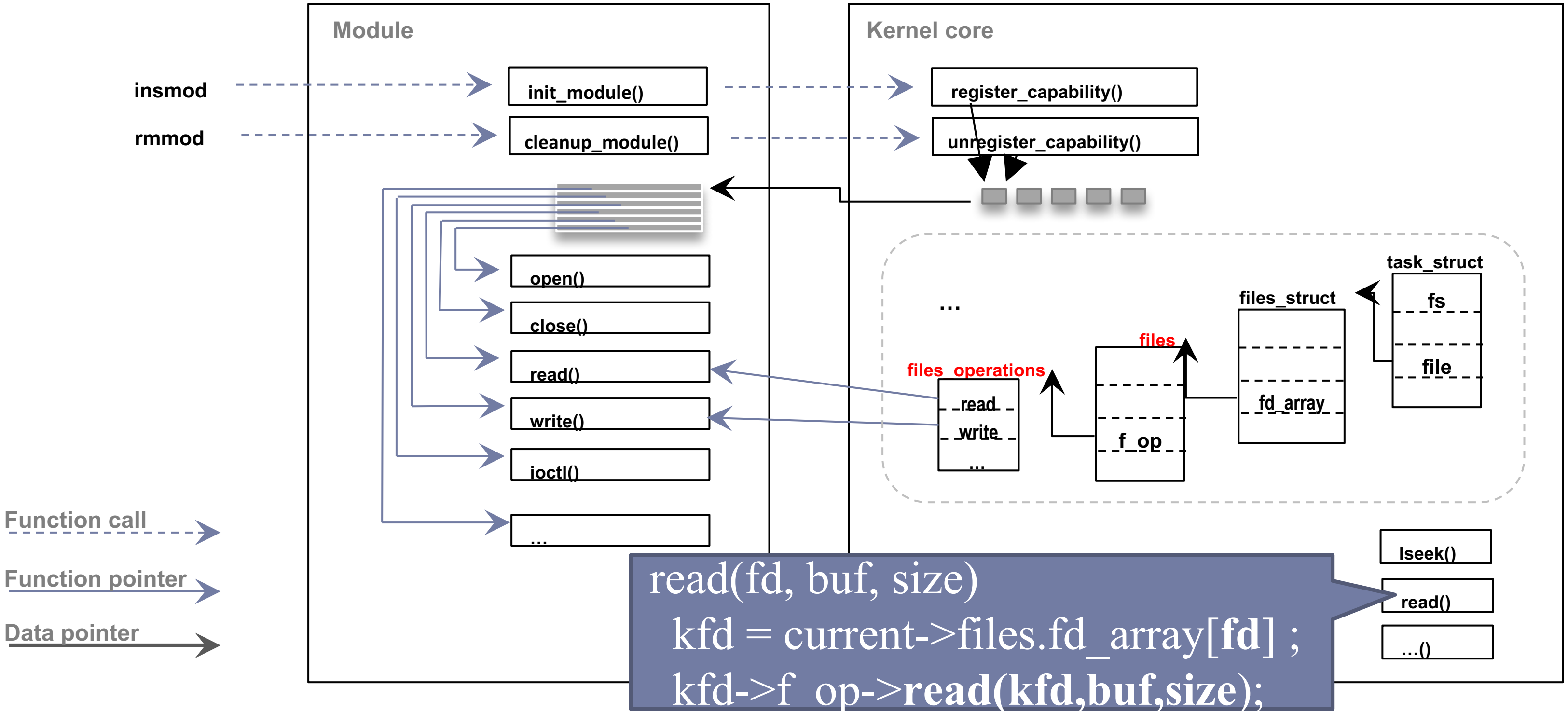


```
lseek(fd, newPos)
  kfd = current->files.fd_array[fd] ;
  kfd->f_pos = newPos;
```

http://www.xml.com/ldd/chapter/book/ch02.html

Main management structures

File table: Linux



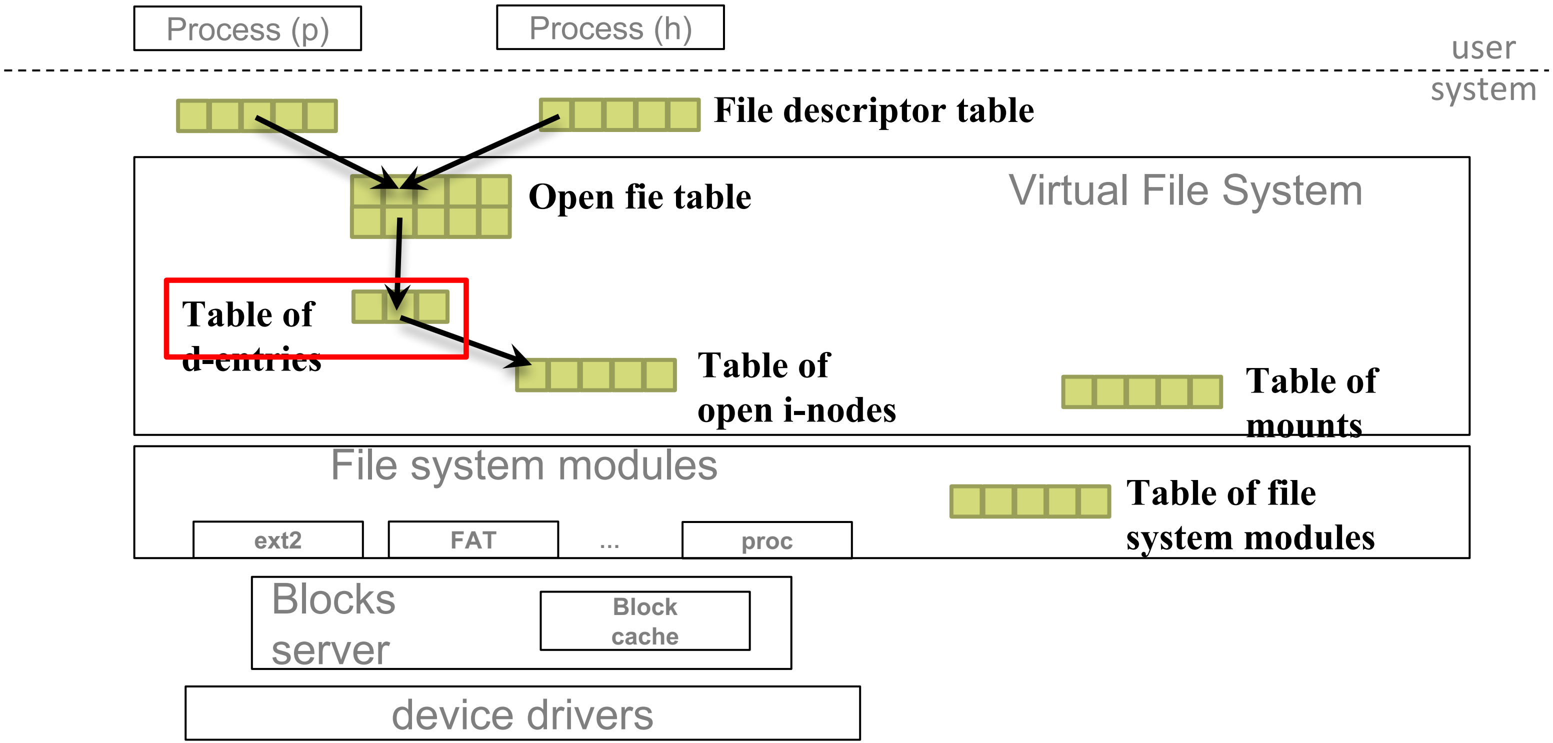
```
read(fd, buf, size)
  kfd = current->files.fd_array[fd] ;
  kfd->f_op->read(kfd,buf,size);
```

http://www.xml.com/ldd/chapter/book/ch02.html

Main goals
(for a Unix-like file system)

- The processes have to use a secure interface,
  without direct access to the kernel representation.
- To share the file offset among process from the same parent that open the file.
- To have a working session with the file/directory in order to update the information that it contains.
- Go back and forth in the file system directory tree.
- Offer persistency of user data, seeking to minimize the impact on the performance and the space needed for the metadata.
- Keep track of the file system registered in the kernel, and keep track of the mount points of these file systems.

Main management structures



Main management structures

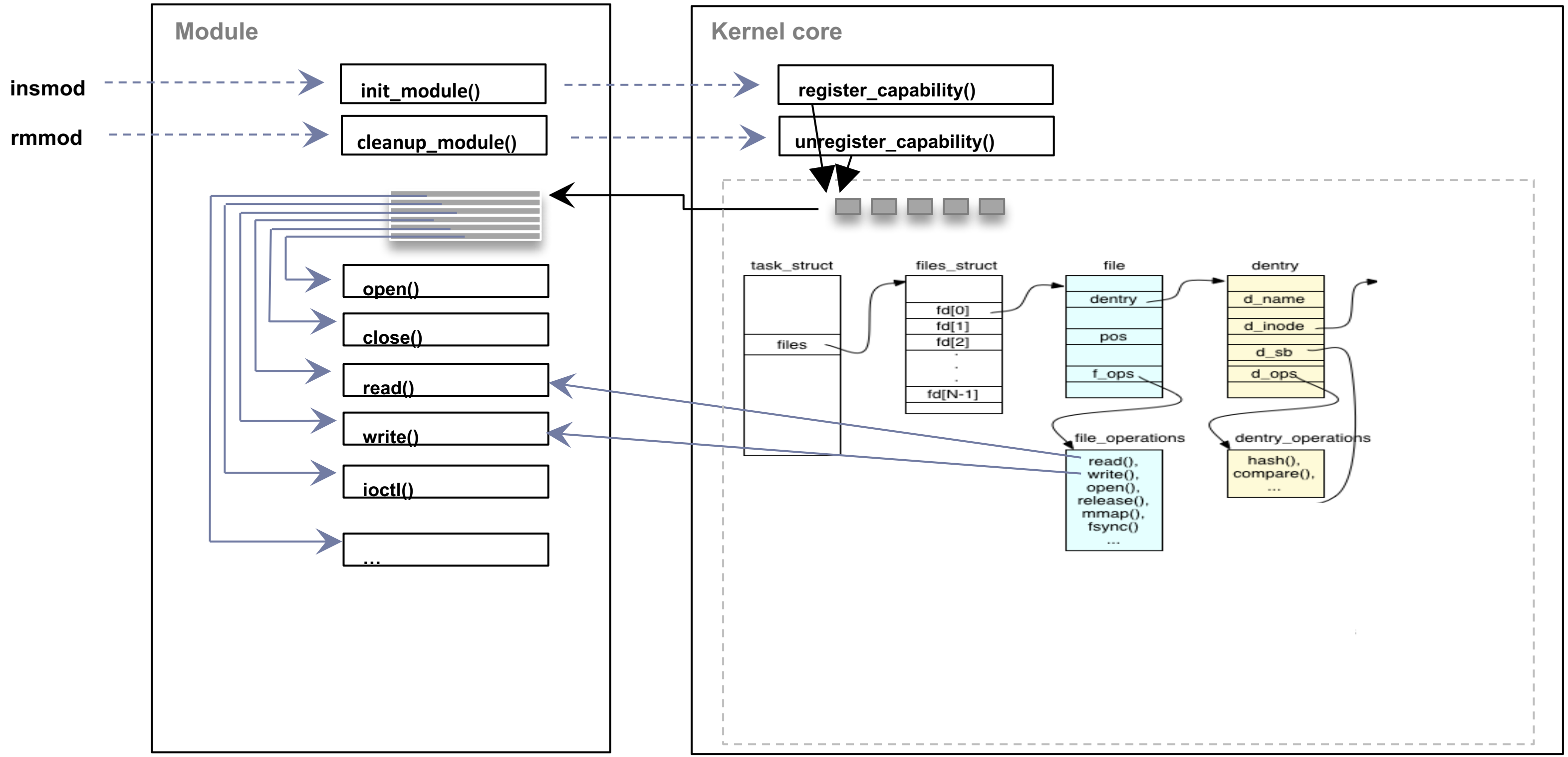
Table of d-entries (directory entries): Linux

```
struct dentry {
        struct  inode            *d_inode;
        struct  dentry           *d_parent;
        struct  qstr              d_name;
        struct  dentry_operations  *d_op;
        struct  super_block         *d_sb;
        struct  list_head          d_subdirs;
        …
}
```

```
struct dentry_operations {
    int (*d_revalidate) (struct dentry *, int);
    int (*d_hash)        (struct dentry *,  struct qstr *);
    int (*d_compare)  (struct dentry *, struct qstr *,
                                    struct qstr *);
    int (*d_delete)      (struct dentry *);
    void (*d_release)  (struct dentry *);
    void (*d_iput)        (struct dentry *,
                                    struct inode *);
}
```

Main management structures

Table of d-entries (directory entries): Linux

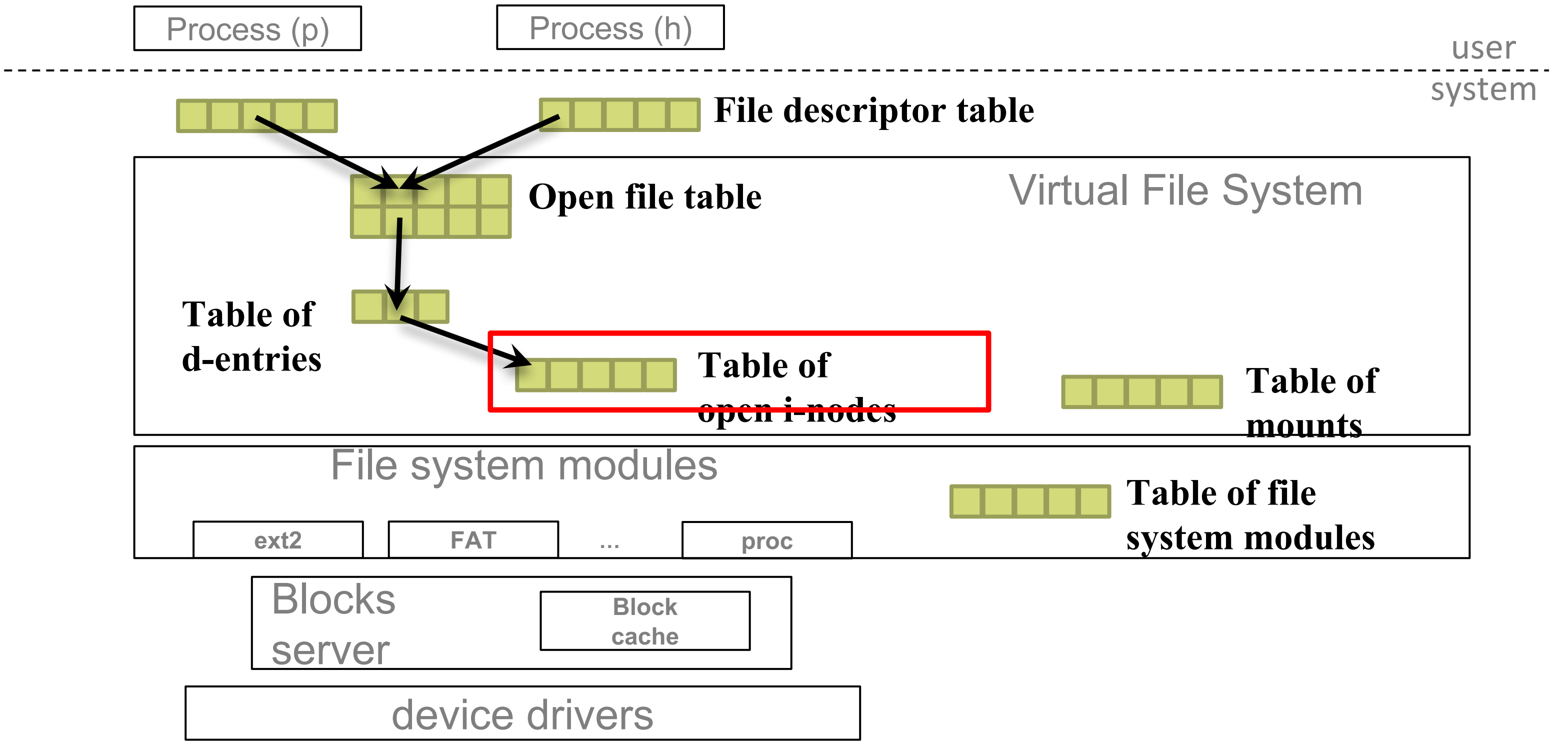


http://www.coins.tsukuba.ac.jp/~yas/classes/os2-2012/2013-02-19/index.html

Main goals
(for a Unix-like file system)

- The processes have to use a secure interface, without direct access to the kernel representation.
- To share the file offset among process from the same parent that open the file.
- To have a working session with the file/directory in order to update the information that it contains
- Go back and forth in the file system directory tree.
- Offer persistency of user data, seeking to minimize the impact on the performance and the space needed for the metadata.
- Keep track of the file system registered in the kernel, and keep track of the mount points of these file systems.

Main management structures



Main management structures

Table of i-nodes: Linux

```
struct inode {
        unsigned long       i_ino;
        umode_t             i_mode;
        uid_t               i_uid;
        gid_t               i_gid;
        kdev_t              i_rdev;
        loff_t              i_size;
        struct timespec     i_atime;
        struct timespec     i_ctime;
        struct timespec     i_mtime;
        struct super_block        *i_sb;
        struct inode_operations  *i_op;
        struct address_space      *i_mapping;
        struct list_head           i_dentry;
        ...
} ;
```

http://www.win.tue.nl/~aeb/linux/lk/lk-8.html

Main management structures

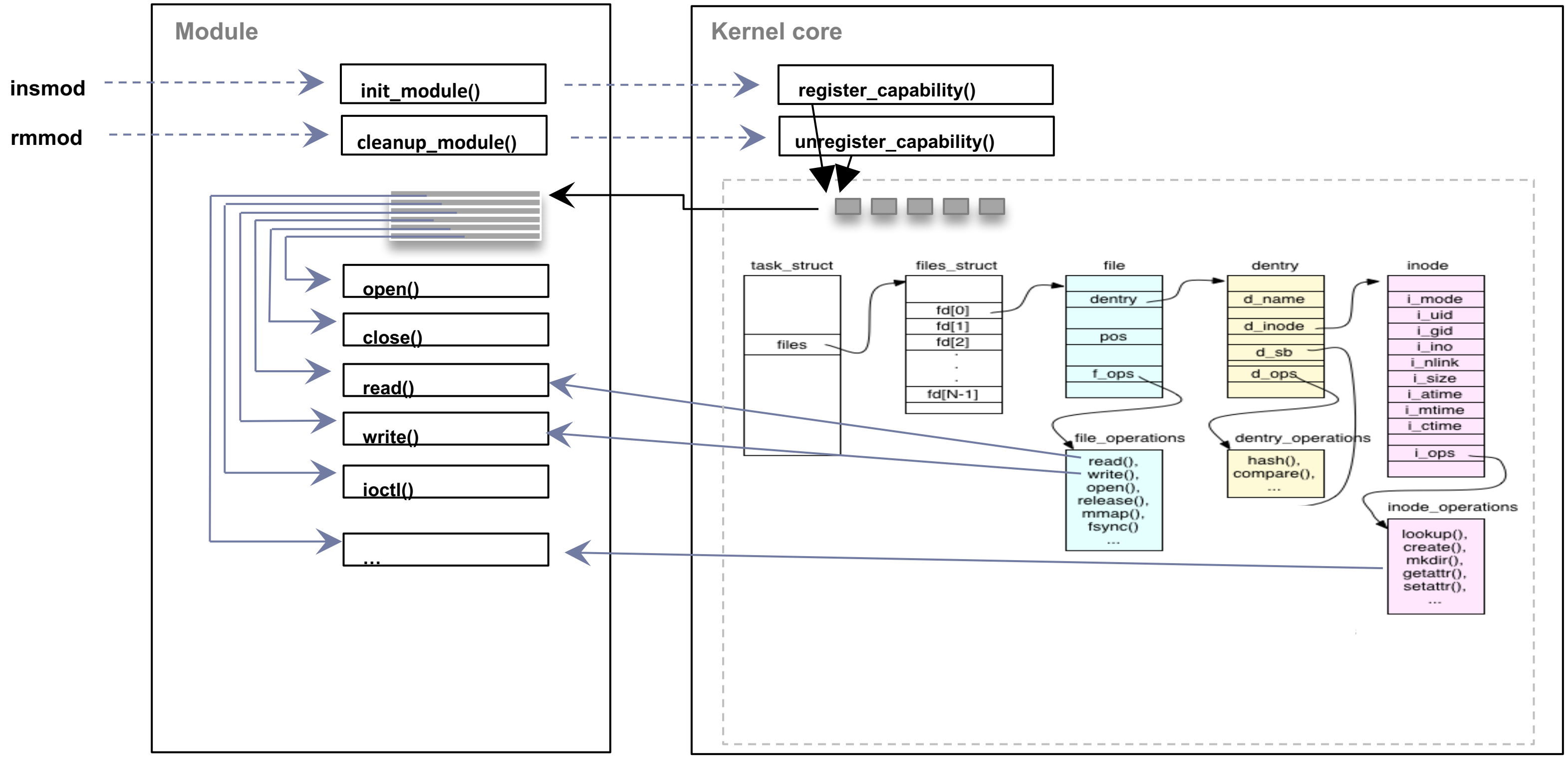
Table of i-nodes: Linux

```
struct inode_operations {
    int (*create) (struct inode *,
                   struct dentry *, int);
    int (*unlink) (struct inode *,
                   struct dentry *);
    int (*mkdir) (struct inode *,
                  struct dentry *, int);
    int (*rmdir) (struct inode *,
                  struct dentry *);
    int (*mknod) (struct inode *,
                  struct dentry *,
                  int, dev_t);
    int (*rename) (struct inode *,
                   struct dentry *,
                   struct inode *,
                   struct dentry *);
    void (*truncate) (struct inode *);
    struct dentry * (*lookup) (struct inode *,
                               struct dentry *);
    int (*permission) (struct inode *, int);
    int (*setattr) (struct dentry *,
                    struct iattr *);
    int (*getattr) (struct vfsmount *mnt,
                    struct dentry *,
                    struct kstat *);
    int (*setxattr) (struct dentry *,
                     const char *,
                     const 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 *);
    int     (*link) (struct dentry *,
                     struct inode *,
                     struct dentry *);
    int (*symlink) (struct inode *,
                    struct dentry *,
                    const char *);
    int (*readlink) (struct dentry *,
                     char *, int);
    int (*follow_link) (struct dentry *,
                        struct nameidata *);
};
```

http://www.win.tue.nl/~aeb/linux/lk/lk-8.html

Main management structures

Table of i-nodes: Linux

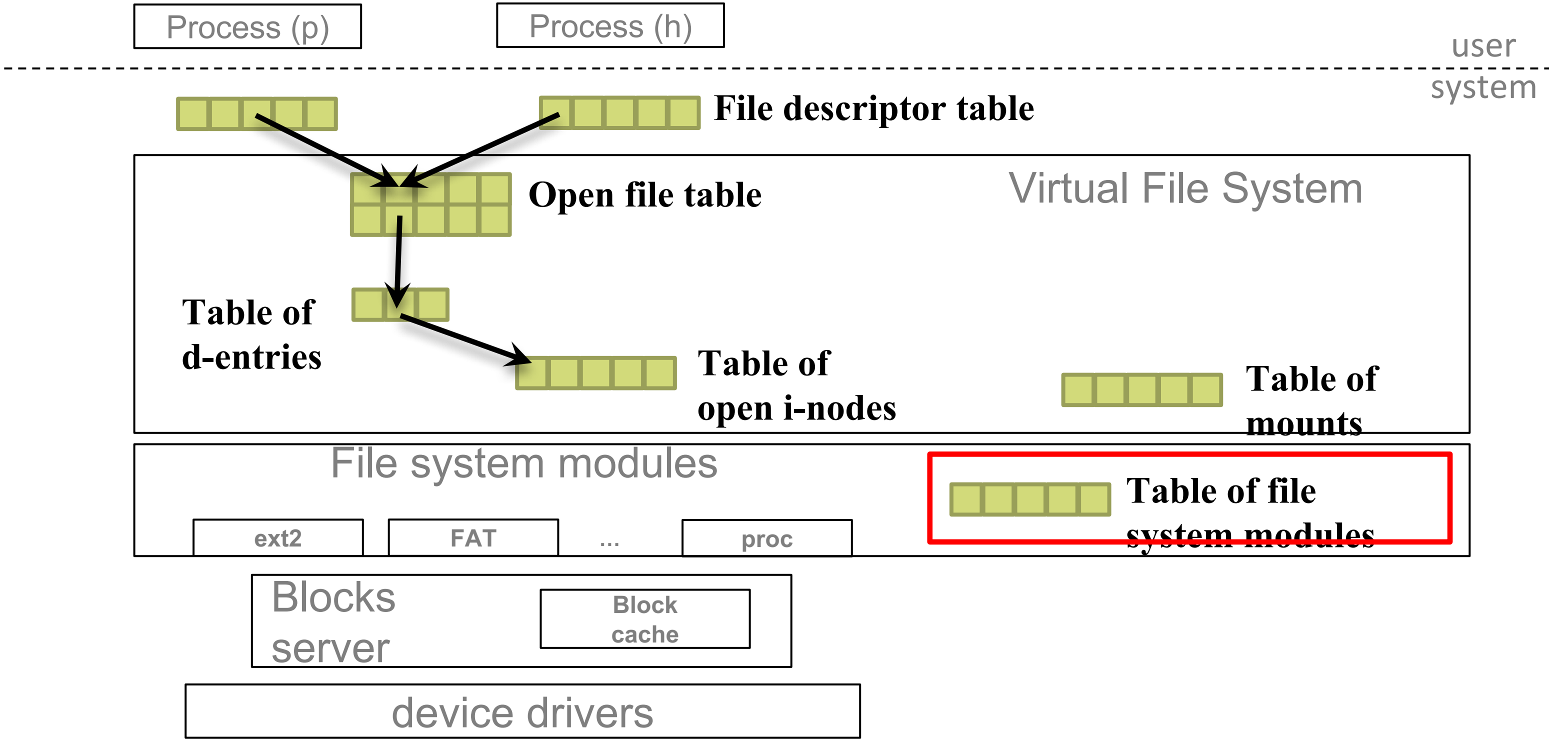


http://www.coins.tsukuba.ac.jp/~yas/classes/os2-2012/2013-02-19/index.html

Main goals
(for a Unix-like file system)

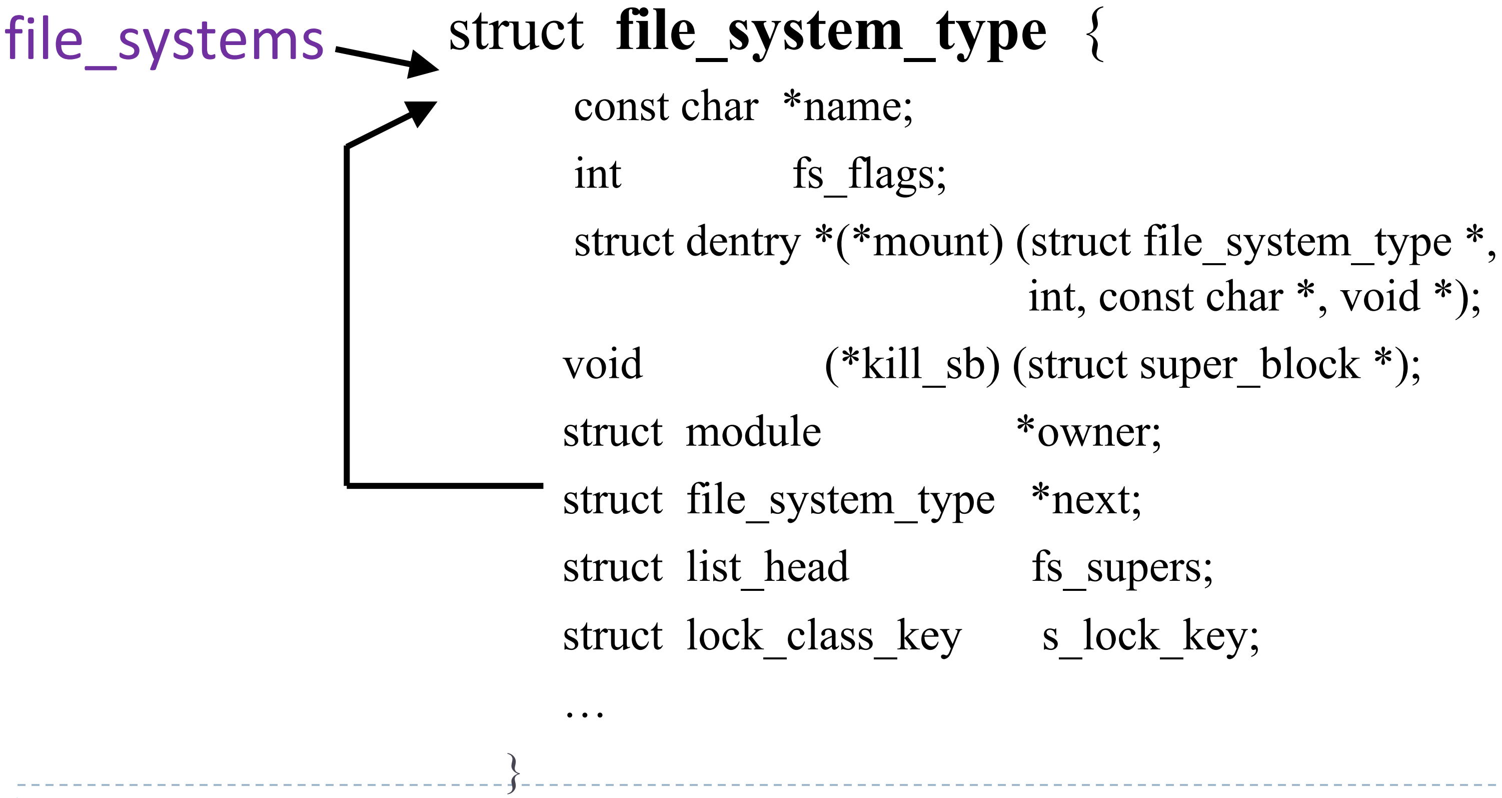
- The processes have to use a secure interface, without direct access to the kernel representation.
- To share the file offset among process from the same parent that open the file.
- To have a working session with the file/directory in order to update the information that it contains.
- Go back and forth in the file system directory tree.
- Offer persistency of user data, seeking to minimize the impact on the performance and the space needed for the metadata.
- Keep track of the file systems registered in the kernel, and keep track of the mount points of these file systems.

Main management structures



Main management structures

File system table: Linux



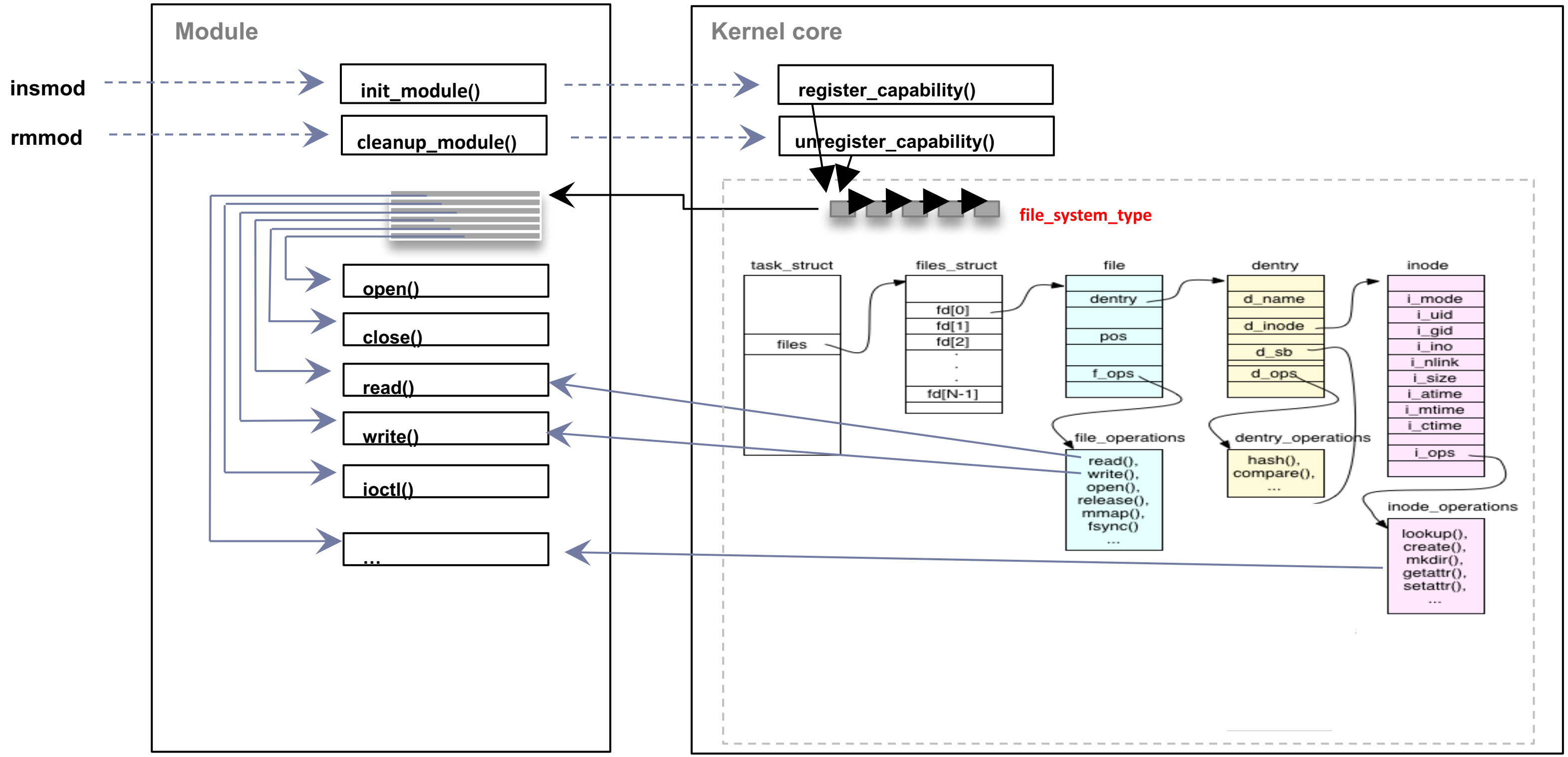
file_systems

```
struct  file_system_type  {
        const char  *name;
        int               fs_flags;
        struct dentry *(*mount) (struct file_system_type *,
                                          int, const char *, void *);
       void                (*kill_sb) (struct super_block *);
       struct  module                  *owner;
       struct  file_system_type   *next;
       struct  list_head                fs_supers;
       struct  lock_class_key       s_lock_key;
       …
}
```

http://www.ibm.com/developerworks/library/l-linux-filesystem/

Main management structures

File system table: Linux



http://www.coins.tsukuba.ac.jp/~yas/classes/os2-2012/2013-02-19/index.html

Main management structures

Table of mounts: Linux

current->namespace->list

```
struct vfsmount {
        struct vfsmount   *mnt_parent;  /* fs we are mounted on */
        struct dentry       *mnt_mountpoint;  /* dentry of mountpoint */
        struct dentry       *mnt_root;     /* root of the mounted tree */
        struct super_block *mnt_sb;      /* pointer to superblock */
        struct list_head     mnt_hash;
        struct list_head     mnt_mounts;  /* list of children, anchored here */
        struct list_head     mnt_child;      /* and going through their mnt_child */
        struct list_head     mnt_list;
        atomic_t               mnt_count;
        int                       mnt_flags;
        char                    *mnt_devname;  /* Device name, e.g. /dev/hda1 */
};
```

http://www.win.tue.nl/~aeb/linux/lk/lk-8.html

Main management structures

Superblock table: Linux

current->namespace->list->mnt_sb

```
struct super_block {
    dev_t                      s_dev;
    unsigned long               s_blocksize;
    struct file_system_type   *s_type;
    struct super_operations   *s_op;
    struct dentry             *s_root;
    …
} ;
```

http://www.win.tue.nl/~aeb/linux/lk/lk-8.html

Main management structures

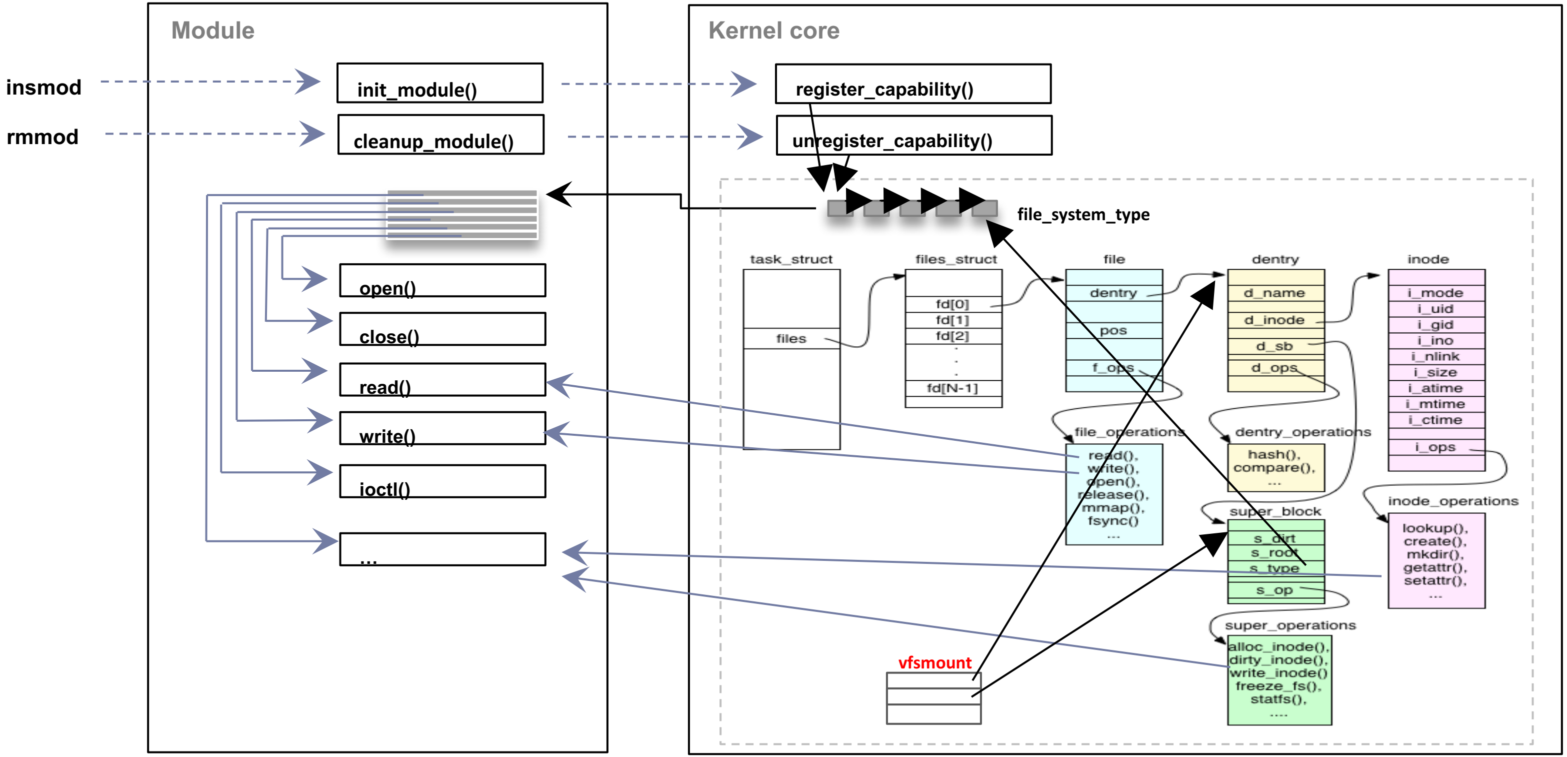
Superblock table: Linux

```
struct super_operations  {
   struct inode *(*alloc_inode)(struct super_block *sb);
   void (*destroy_inode)(struct inode *);
   void (*read_inode) (struct inode *);
   void (*dirty_inode) (struct inode *);
   void (*write_inode) (struct inode *, int);
   void (*put_inode) (struct inode *);
   void (*drop_inode) (struct inode *);
   void (*delete_inode) (struct inode *);
   void (*clear_inode) (struct inode *);
   void (*put_super) (struct super_block *);
   void (*write_super) (struct super_block *);
   int   (*sync_fs)(struct super_block *sb, int wait);
   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 (*umount_begin) (struct super_block *);
   int (*show_options)(struct seq_file *, struct vfsmount *);
};
```

http://www.win.tue.nl/~aeb/linux/lk/lk-8.html

Main management structures

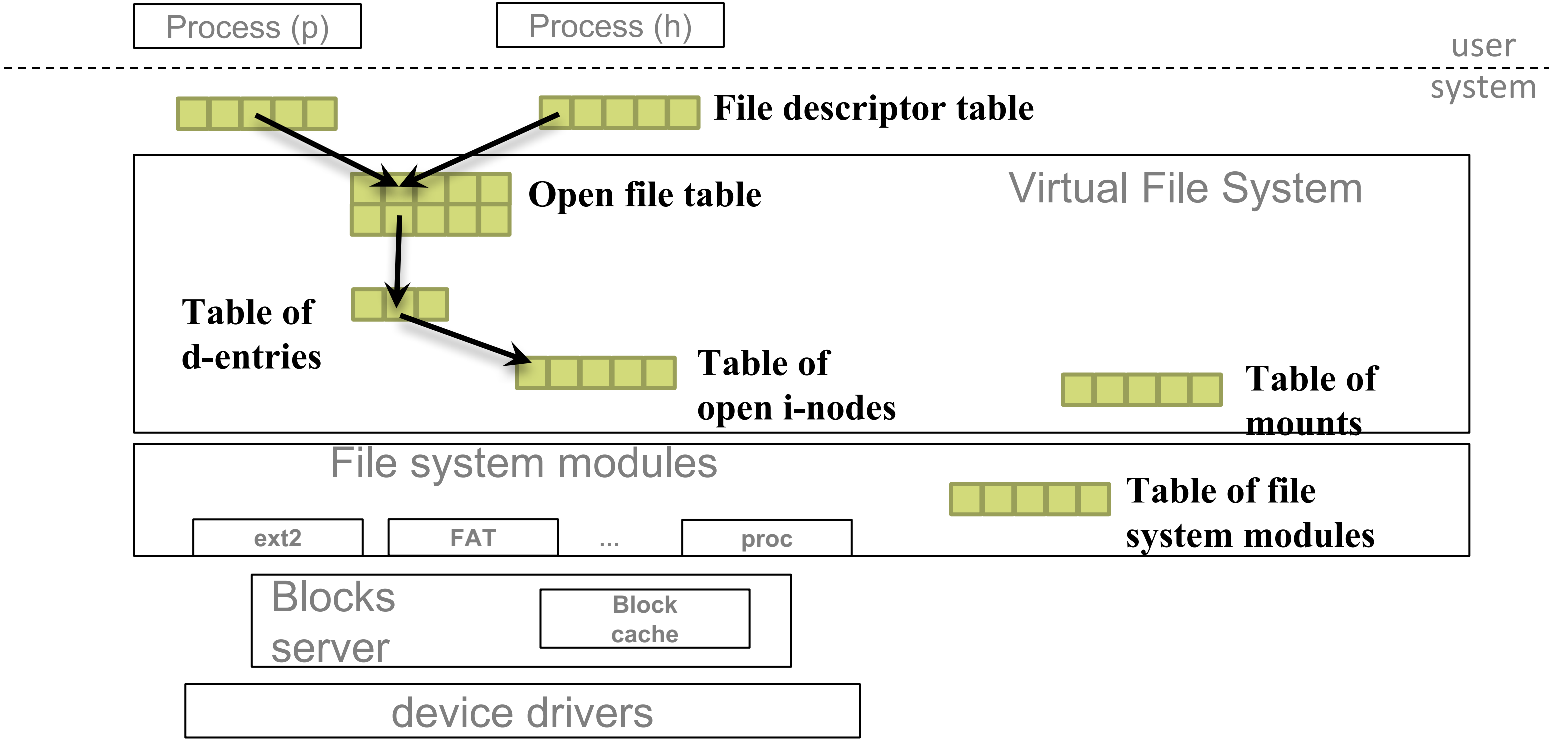
Table of mounts: Linux



http://www.coins.tsukuba.ac.jp/~yas/classes/os2-2012/2013-02-19/index.html

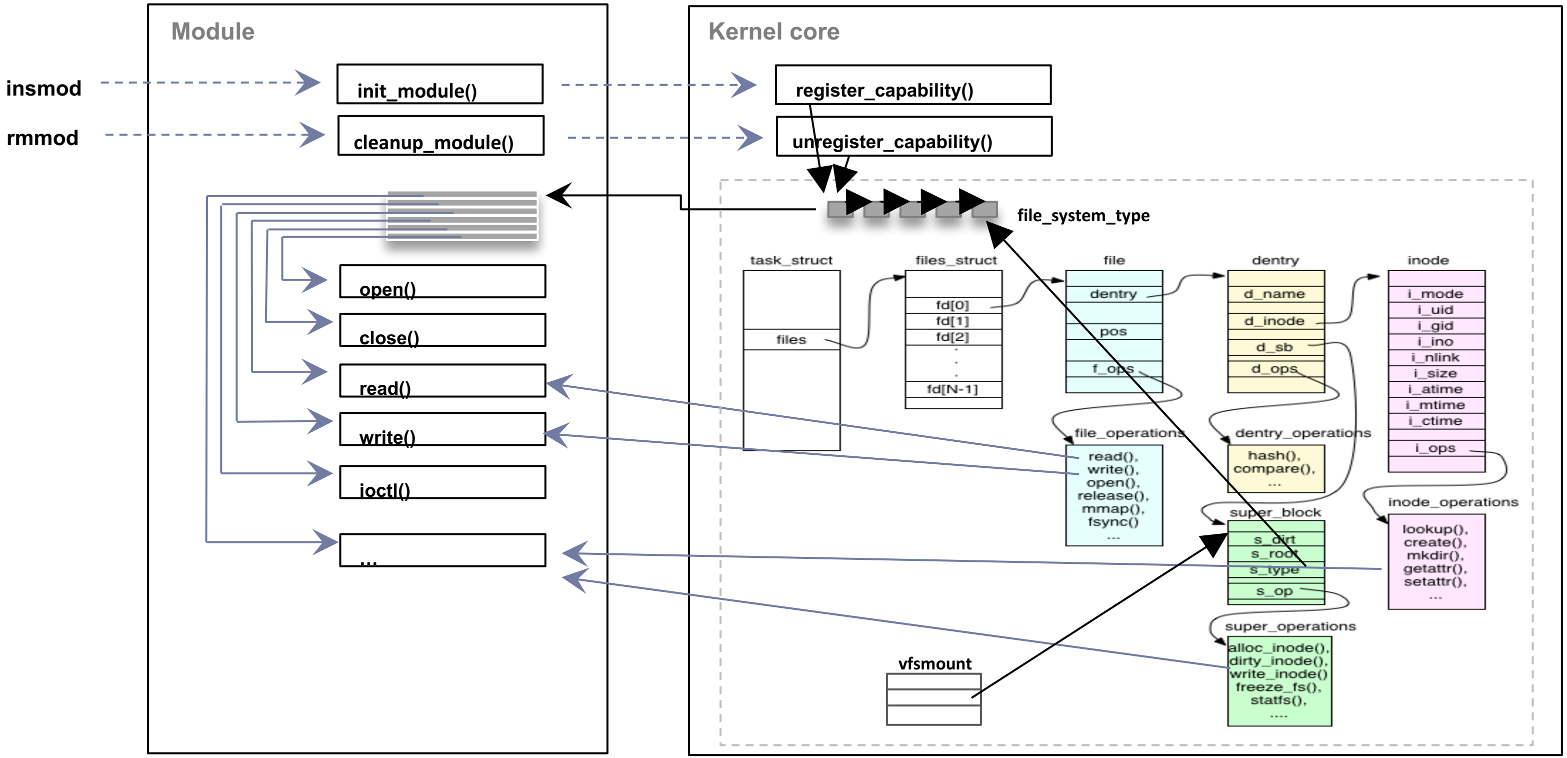
Main management structures

summary



Main management structures

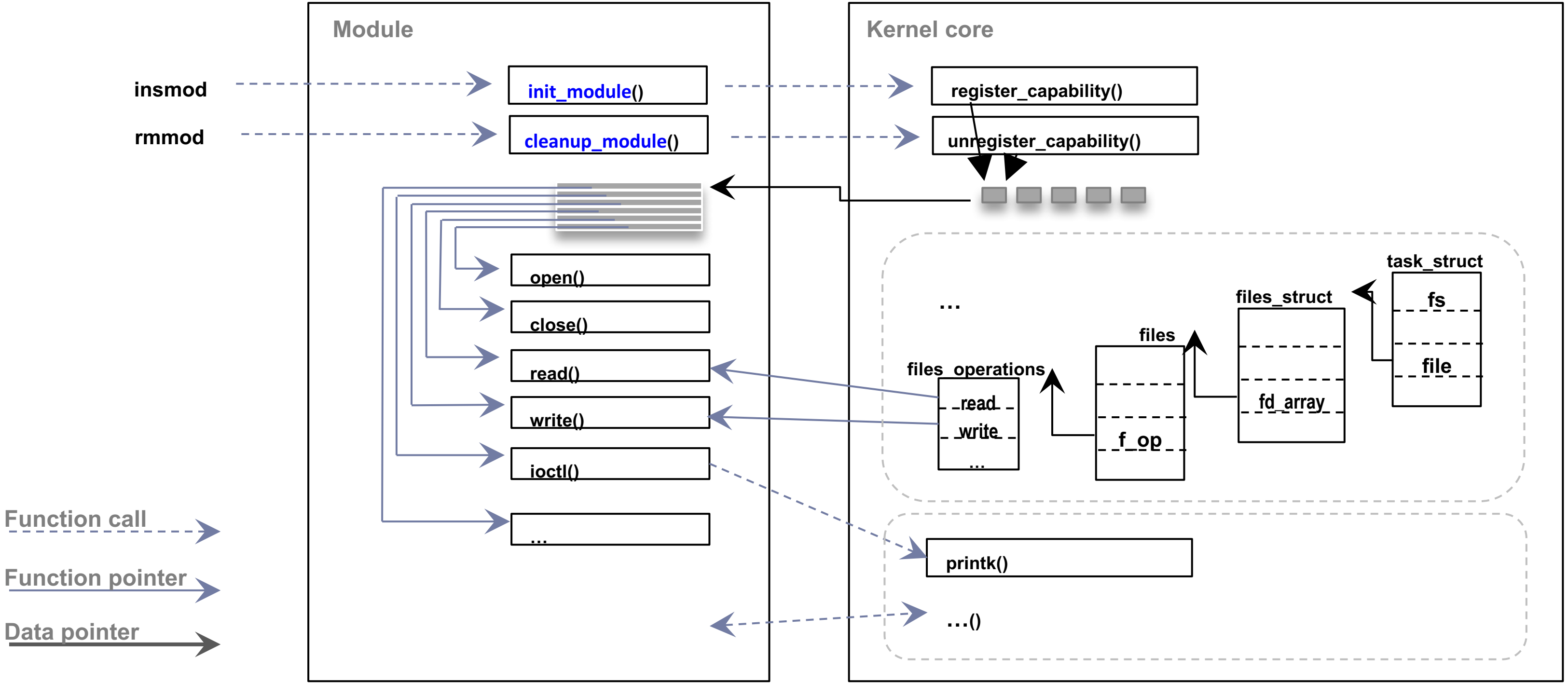
summary



http://www.coins.tsukuba.ac.jp/~yas/classes/os2-2012/2013-02-19/index.html

Main management structures

summary (usage)



http://www.xml.com/ldd/chapter/book/ch02.html

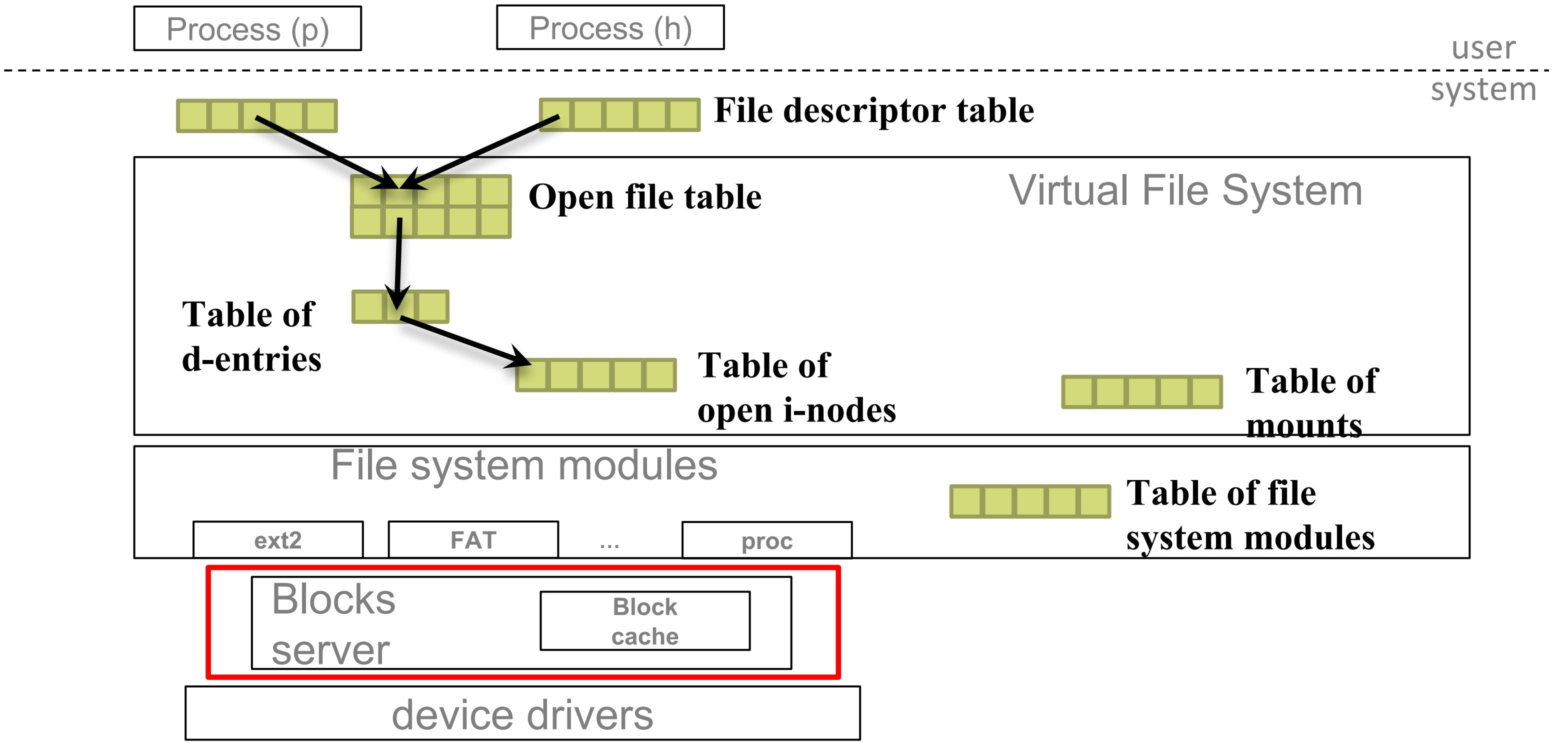
Main goals
(for a Unix-like file system)

- ✓ The processes have to use a secure interface, without direct access to the kernel representation.
- ✓ To share the file offset among process from the same parent that open the file.
- ✓ To have a working session with the file/directory in order to update the information that it contains.
- ✓ Go back and forth in the file system directory tree.
- ✓ Offer persistency of user data, seeking to minimize the impact on the performance and the space needed for the metadata.
- ✓ Keep track of the file system registered in the kernel, and keep track of the mount points of these file systems.

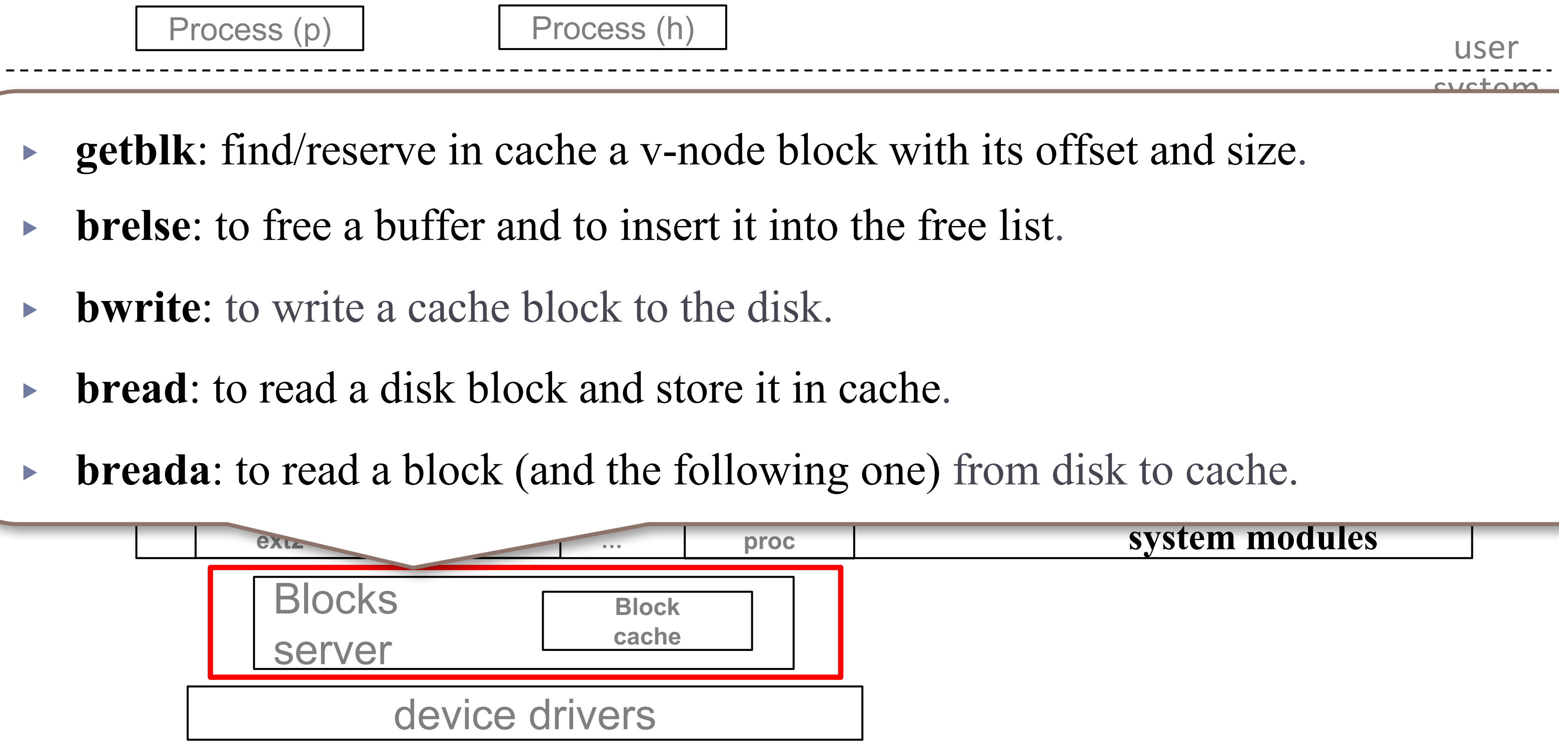
Overview

1. Introduction
2. Main data structures on the secondary memory
3. Main data structures in the main memory
4. **Block management**
5. Complementary aspects

Main management structures



Main management structures



- **getblk**: find/reserve in cache a v-node block with its offset and size.
- **brelse**: to free a buffer and to insert it into the free list.
- **bwrite**: to write a cache block to the disk.
- **bread**: to read a disk block and store it in cache.
- **breada**: to read a block (and the following one) from disk to cache.

Block server

- It is responsible for:
  - Issuing commands to read and write device drivers blocks (by using the specific device routines)
  - Optimizing the I/O requests.
    - E.g.: Block cache.
  - Offering a logical device namespace.
    - E.g.: /dev/hda3 (third partition of the first disk)

Block server

- General behavior:
  - If the block is in the cache
    - Copy the content (and update the block usage metadata)
  - If it is not in the cache
    - Read the block from the device and store it in cache
    - Copy the content (and to update the block metadata)
    - If the block has been modified (*dirty*)
      - □ Cache write policy
    - If the cache is full, it is necessary get some free slots
      - □ Cache replacement policy

Block server

- General behavior:
  - If the block is in the cache

○ **Read-ahead**:

  ○ Read the following blocks into the cache (in order to improve the performance on sequential accesses)

    - To read the block from the device and store it in cache
    - To copy the content (and to update the block metadata)
    - If the block has been modified (*dirty*)
      - □ Cache write policy
    - If the cache is full, it is necessary get some free slots
      - □ Cache replacement policy

Block server

- General behavior:

- **write-through**:
  - Each time a block is modified it is also flushed to disk (lower performance)
- **write-back**:
  - The blocks are flushed to disk only when the block has to be evicted from the cache and it was dirty (better performance but reliability problems)
- **delayed-write**:
  - The modified blocks are saved to disk periodically (e.g., every 30 seconds in Unix) (trade-off for the former options)
- **write-on-close**:
  - When the file descriptor is closed, all file blocks are flushed to disk.

  - If the bloc een modified (*dirty*)
    - □ Cache write policy
  - If the cache is full, it is necessary get some free slots
    - □ Cache replacement policy

Block server

- General behavior:
  - If the block is in the cache
    - To copy the content (and to update the block usage metadata)
  - If it is not in the cache
    - To read the block from the device into the cache
    - If the cache is full, it is necessary get some free slots
      - □ Cache replacement policy

- ○ **FIFO** *(First in First Out)*
- ○ **Clock algorithm** *(Second opportunity)*
- ○ **MRU** *(Most Recently Used)*
- ○ **LRU** *(Least Recently Used)*

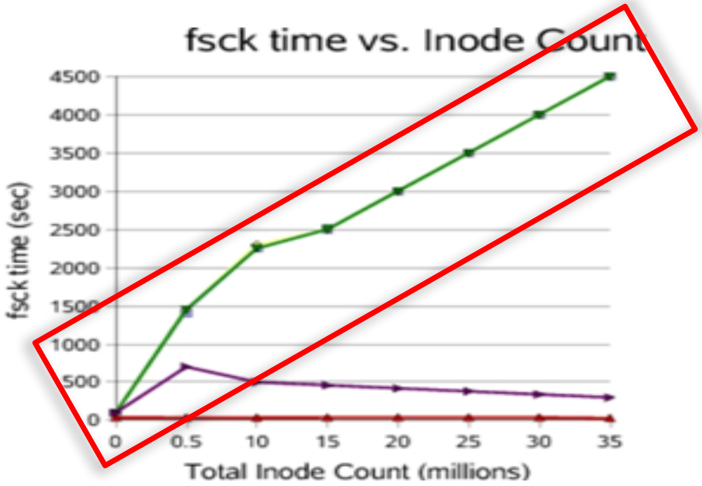
Overview

1. Introduction
2. Main data structures on the secondary memory
3. Main data structures in the main memory
4. Block management
5. **Complementary aspects**

Advanced features

- Journaling
- Snapshots
- Dynamic file system expansion

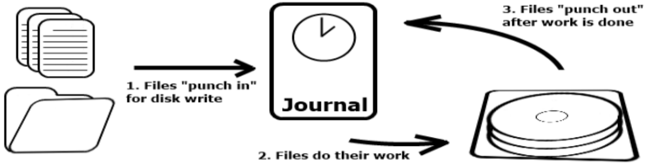
Without *Journaling*

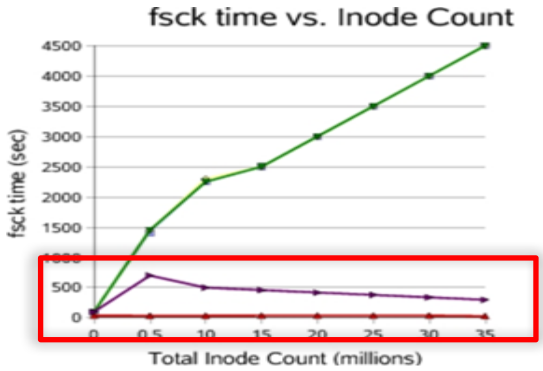


- If the computer is shut down abruptly, the file system might remain be inconsistent.
- In order to repair the file system, all metadata has to be reviewed:
  - The required time depends of the file system size (all the metadata has to be reviewed, the more metadata to be reviewed the more time is needed).

Storage Networks Explained: Basics and Application of Fibre Channel SAN, NAS… (Página 139)

With *Journaling*


1. Files "punch in" for disk write
Journal
2. Files do their work
3. Files "punch out" after work is done



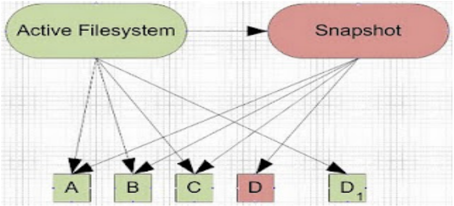
- The file system writes the changes in a log before changing the file.
- If the computer is shut down abruptly, the file system checks has to review the log for the pending changes, and do these changes (commit):
  - The time needed depends of the number of pending changes in the log, and does not depend on the file system size.
  - From hours to seconds…

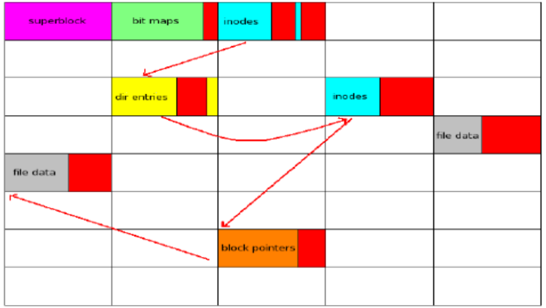
http://www.howtogeek.com/howto/33552/htg-explains-which-linux-file-system-should-you-choose/

Advanced features

- Journaling
- Snapshots
- Dynamic file system expansion

Snapshot





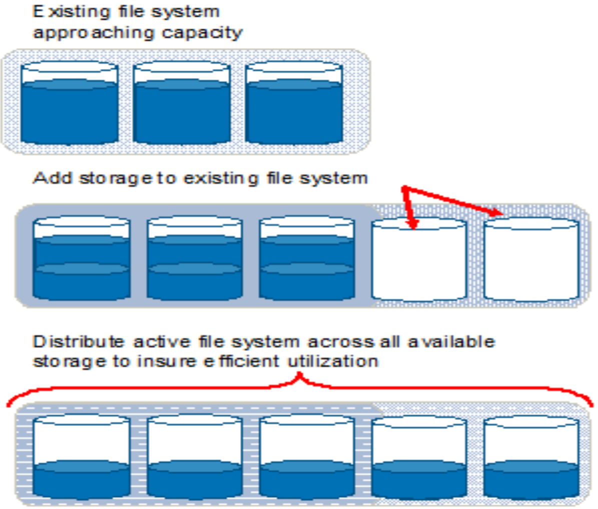
- A Snapshot represents the state of the file system at a point of time:
  - In a few seconds is done.
  - It is possible to access to all the file system snapshots on this disk.
- E.g.: system updates, backups, etc.

Storage Networks Explained: Basics and Application of Fibre Channel SAN, NAS… (Página 139)
http://lwn.net/Articles/342892/, http://sannasanalyst.blogspot.com.es/2010_06_01_archive.html

Advanced features

- Journaling
- Snapshots
- Dynamic file system expansion

Dynamic file system expansion



Existing file system approaching capacity
Add storage to existing file system
Distribute active file system across all available storage to insure efficient utilization

- It is important to design the file system in a way that it could be resized (add more space, remove space, etc.) without losing information.
  - Dynamic and flexible structures
  - Metadata is distributed along the disk

http://www.storagenetworks.com/documents/productdatasheets/x9000_nas_proposal.htm

ARCOS Group
Universidad Carlos III de Madrid

Lesson 5 (b)
File systems

Operating System Design
Bachelor in Informatics Engineering