OPERATING SYSTEMS: FILE SYSTEMS

Files, directories and file system

To remember…

Before classes

Class

After class

Prepare the prerequisites.

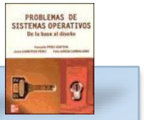
Study the material associated with the **bibliography**: slides alone are not enough.
Please ask questions (especially after study).

Exercising skills:

- Perform all **exercises**.
- Carrying out the **practice notebooks** and **the practical exercises** progressively.

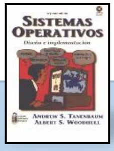
Recommended reading

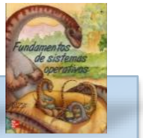
Base



1. Carretero 2020:
   1. Cap. 6
2. Carretero 2007:
   1. Cap. 9.1-9.5,
   2. Cap. 9.8-9.10 & 9.12

Suggested





1. Tanenbaum 2006:
   1. (es) Cap. 6
   2. (en) Cap. 6
2. Stallings 2005:
   1. 12.1-12.8
3. Silberschatz 2006:
   1. 10.3-10.4,
   2. 11.1-11.6 and 13

Contents

□ Introduction

□ File

□ Directory

□ File System

□ Partitions/Volumes

□ Devices

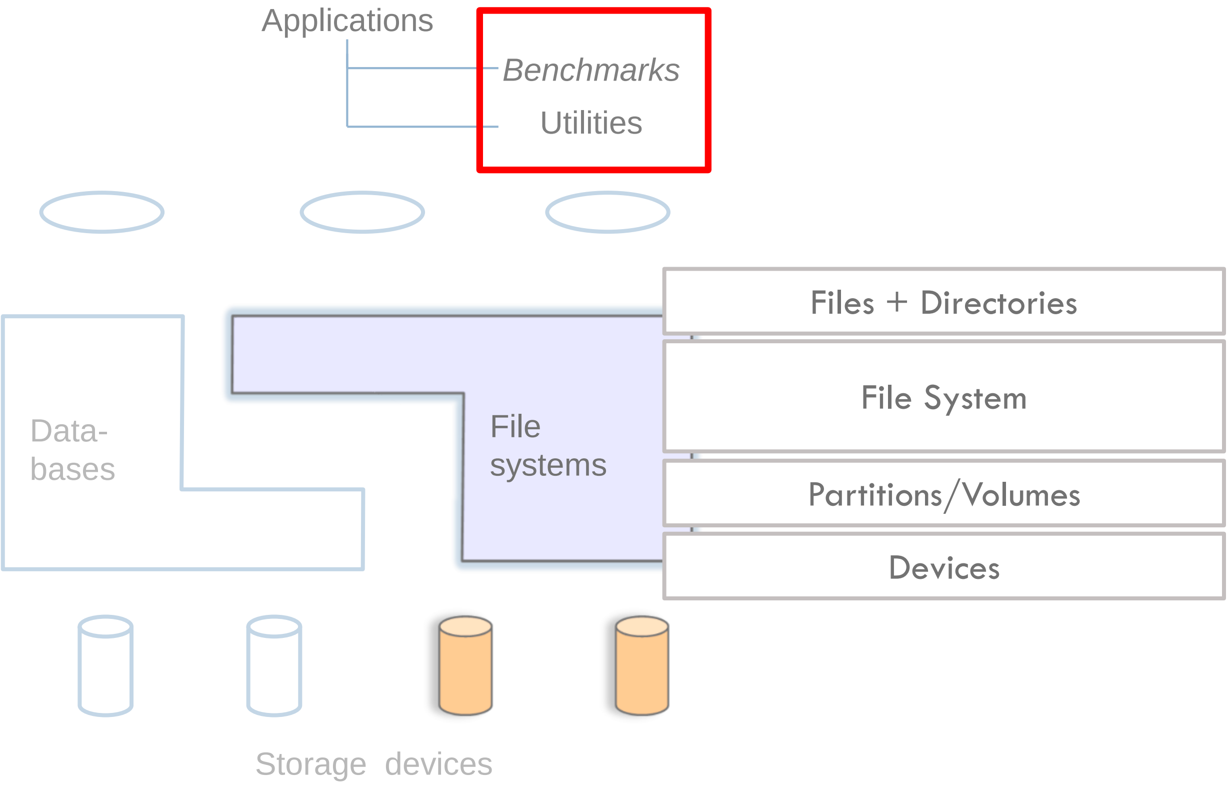
□ System software

□ File System (manager)

Contents

- □ Introduction
- □ File
- □ Directory
- □ File System
- □ Partitions/Volumes
- □ Devices
- □ **System software**
- □ File System (manager)

System software



Benchmarks

- Benchmarks:
  - They allow to measure the performance of the file system (and any dependency on it)
  - Designed to measure different aspects:
    latency, bandwidth,
    number of files processed per unit time, etc.
  - Examples working with metadata: fdtree, mdtest, etc.
  - Examples working with data: iozone, postmark, IOR, etc.

File system consistency

- Software failures may result in inconsistent information (and metadata).
- Solution:
  - Availability of tools to check the file system and repair the errors found.
- Two important aspects to review:
  - Verify that the physical structure of the file system is coherent
  - Verify that the logical structure of the file system is correct.

File system consistency

- Software failures may result in inconsistent information (and metadata).

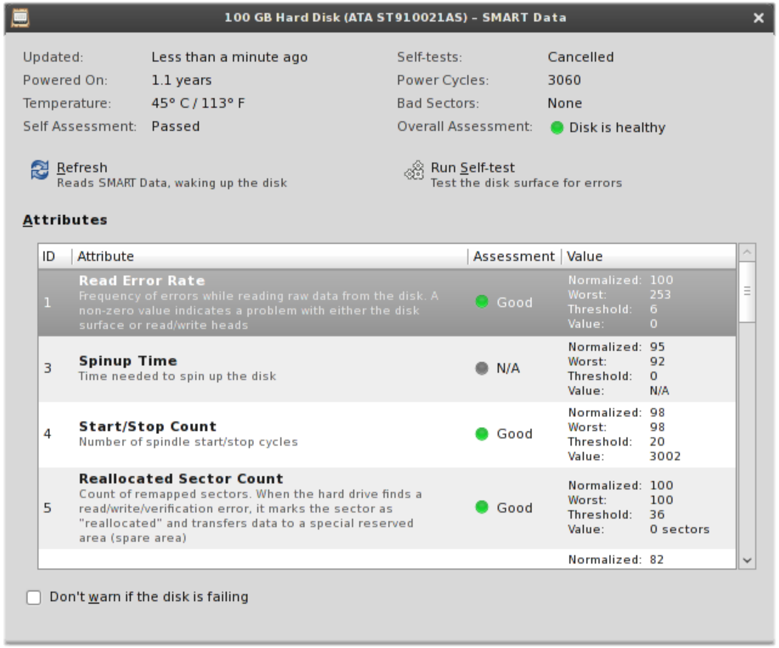
- Solution:
  - Availability of tools to check the file system and repair the errors found.

- Two important aspects to review:
  - Verify that the physical structure of the file system is coherent
  - Verify that the logical structure of the file system is correct.

File system consistency

physical structure

- Controller logic:
  - Disk-controller status tests are performed.
  - E.g.: S.M.A.R.T.



100 GB Hard Disk (ATA ST910021AS) – SMART Data

Updated:	Less than a minute ago	Self-tests:	Cancelled
Powered On:	1.1 years	Power Cycles:	3060
Temperature:	45° C / 113° F	Bad Sectors:	None
Self Assessment:	Passed	Overall Assessment:	● Disk is healthy

Refresh
Reads SMART Data, waking up the disk

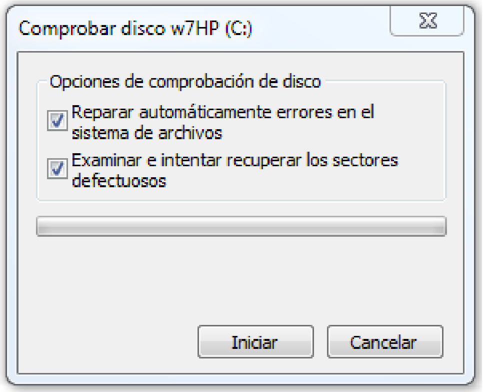
Run Self-test
Test the disk surface for errors

Attributes

ID	Attribute	Assessment	Value
1	**Read Error Rate** Frequency of errors while reading raw data from the disk. A non-zero value indicates a problem with either the disk surface or read/write heads	● Good	Normalized: 100 Worst: 253 Threshold: 6 Value: 0
3	**Spinup Time** Time needed to spin up the disk	● N/A	Normalized: 95 Worst: 92 Threshold: 0 Value: N/A
4	**Start/Stop Count** Number of spindle start/stop cycles	● Good	Normalized: 98 Worst: 98 Threshold: 20 Value: 3002
5	**Reallocated Sector Count** Count of remapped sectors. When the hard drive finds a read/write/verification error, it marks the sector as "reallocated" and transfers data to a special reserved area (spare area)	● Good	Normalized: 100 Worst: 100 Threshold: 36 Value: 0 sectors
			Normalized: 82

☐ Don't warn if the disk is failing

- Disk surface:
  - Reads/writes disk blocks one by one to check for problems on the surface of part of the disk.
  - E.g.: if what is read is different from what is written



Comprobar disco w7HP (C:)

Opciones de comprobación de disco

☑ Reparar automáticamente errores en el sistema de archivos

☑ Examinar e intentar recuperar los sectores defectuosos

File system consistency

- Software failures may result in inconsistent information (and metadata).

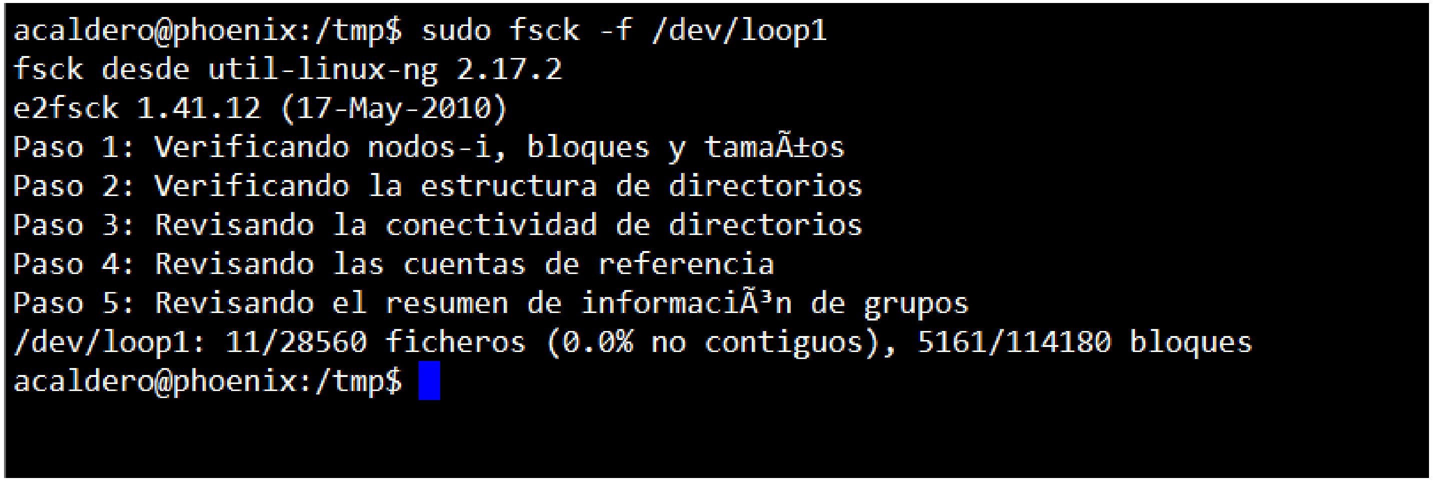
- Solution:
  - Availability of tools to check the file system and repair the errors found.

- Two important aspects to review:
  - Verify that the physical structure of the file system is coherent
  - Verify that the logical structure of the file system is correct.

File system consistency

logical structure

- Disk structures:
  - Check that the data structure on disk is consistent for partition, directories and files
  - E.g.: fsck in Linux, scandisk in Windows



```
acaldero@phoenix:/tmp$ sudo fsck -f /dev/loop1
fsck desde util-linux-ng 2.17.2
e2fsck 1.41.12 (17-May-2010)
Paso 1: Verificando nodos-i, bloques y tamaÃ±os
Paso 2: Verificando la estructura de directorios
Paso 3: Revisando la conectividad de directorios
Paso 4: Revisando las cuentas de referencia
Paso 5: Revisando el resumen de informaciÃ³n de grupos
/dev/loop1: 11/28560 ficheros (0.0% no contiguos), 5161/114180 bloques
acaldero@phoenix:/tmp$
```

File system consistency

logical structure

- File System on disk:
  - Check that the content of the superblock corresponds to the characteristics of the file system.
  - It is checked that the i-node bitmaps correspond to the occupied i-nodes in the file system.
  - Check that the bitmaps of blocks correspond to the blocks assigned to files.
  - Check that no block is assigned to more than one file.
- Directories:
  - The directory system of the file system is checked to see that the same node-i is not assigned to more than one directory.
- Files:
  - The protection and privilege bits are checked.
  - The link counter is checked.

Backup

Where?

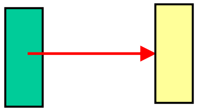
- Place:
  - Distant from the main system
  - Protected from water, fire, etc.
    - Fireproof cabinets

- Medium:
  - Hard disk
    - A: capacity and price, D: fragile
  - Tape
    - A: capacity and price, D: slow

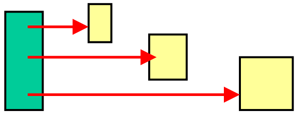
Backup

How?

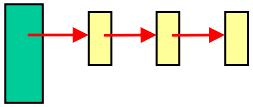
- **Full backup**:
  copy the entire contents of the file system.



- **Differential backup**:
  contains all files that have been changed since the last **full backup**.



- **Incremental backup**:
  contains all files that have been modified since the last **full backup** or **differential backup**

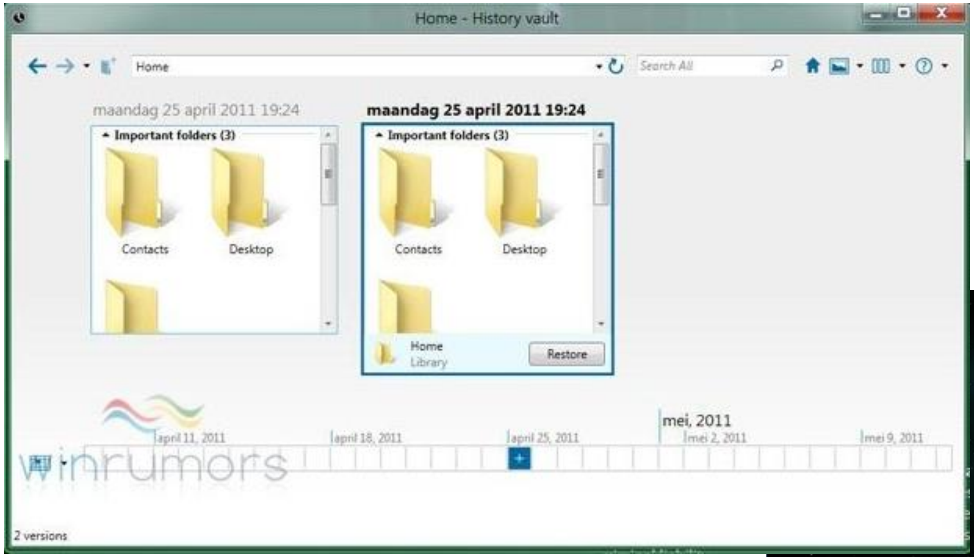


Backup

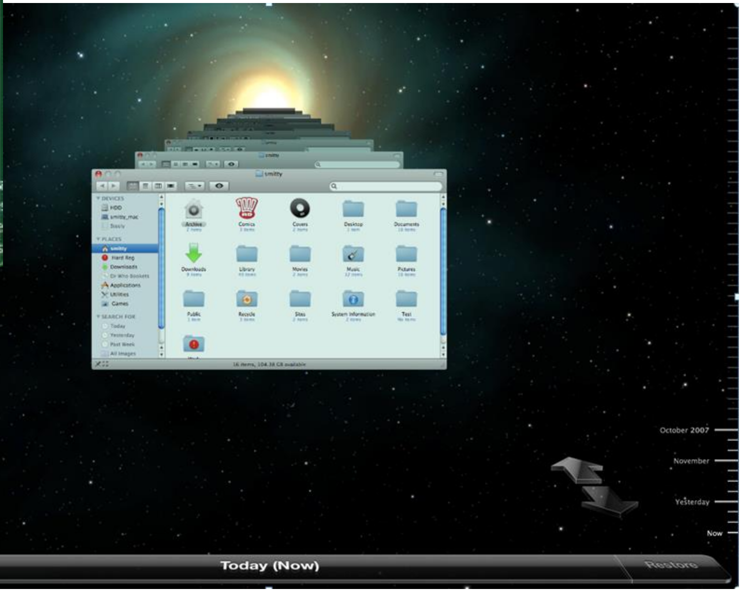
When?

- Off-line:
  - The backup is performed during periods of time when the system data is not in use.
- On-line:
  - The backup is performed while the system is in use.
  - Use of techniques to avoid consistency problems:
    - *Snapshots*
      read-only copy of the file system state.
    - *Copy-on-write*
      writes after snapshot are performed in copies.

Backup copy



http://www.genbeta.com/systems-operativos/primeras-imagenes-de-history-vault-el-time-machine-de-windows-8



http://www.reghardware.com/2007/11/08/review_leopard_pt2/page2.html

Contents

□ Introduction

□ File

□ Directory

□ File System

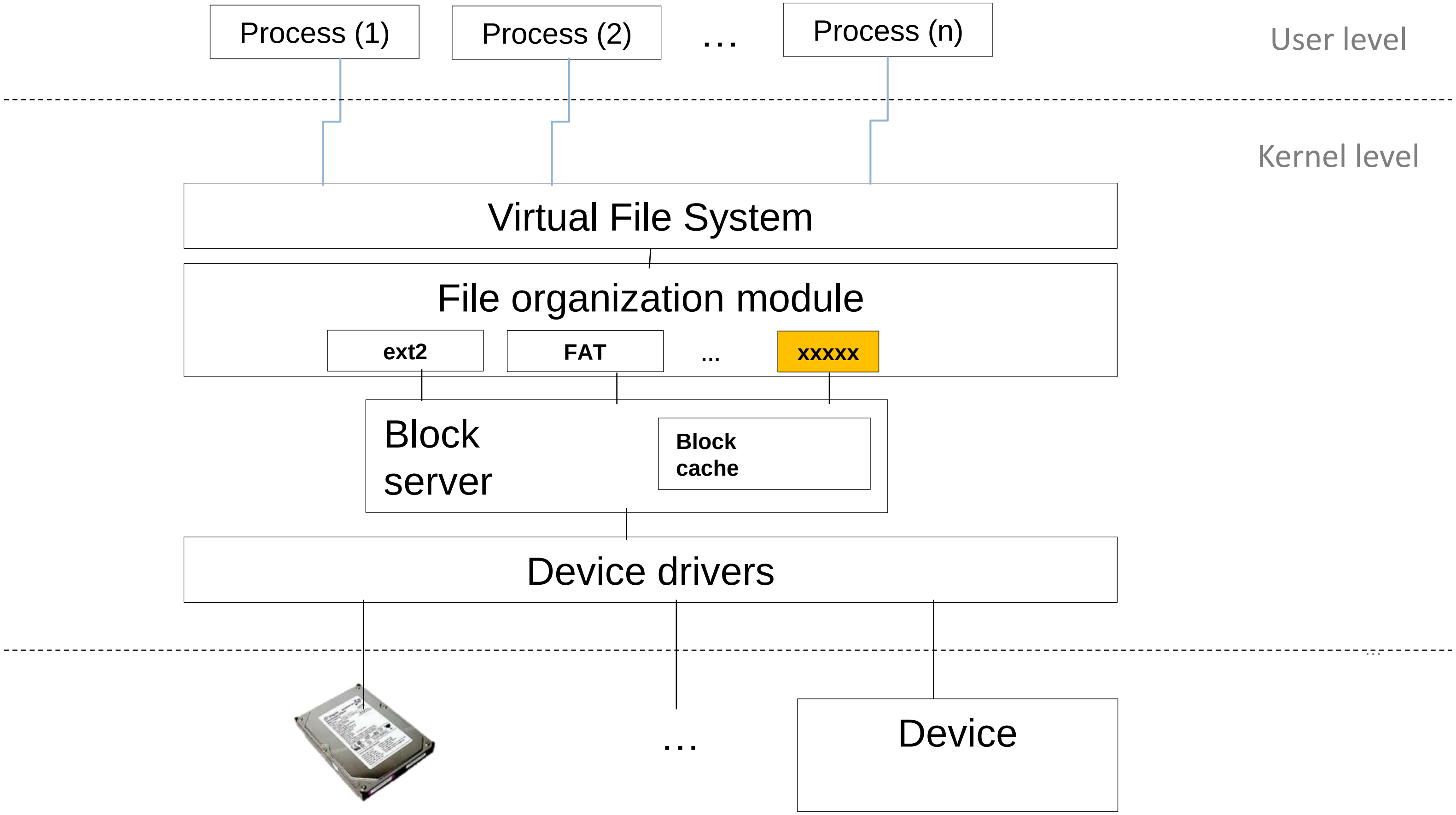
□ Partitions/Volumes

□ Devices

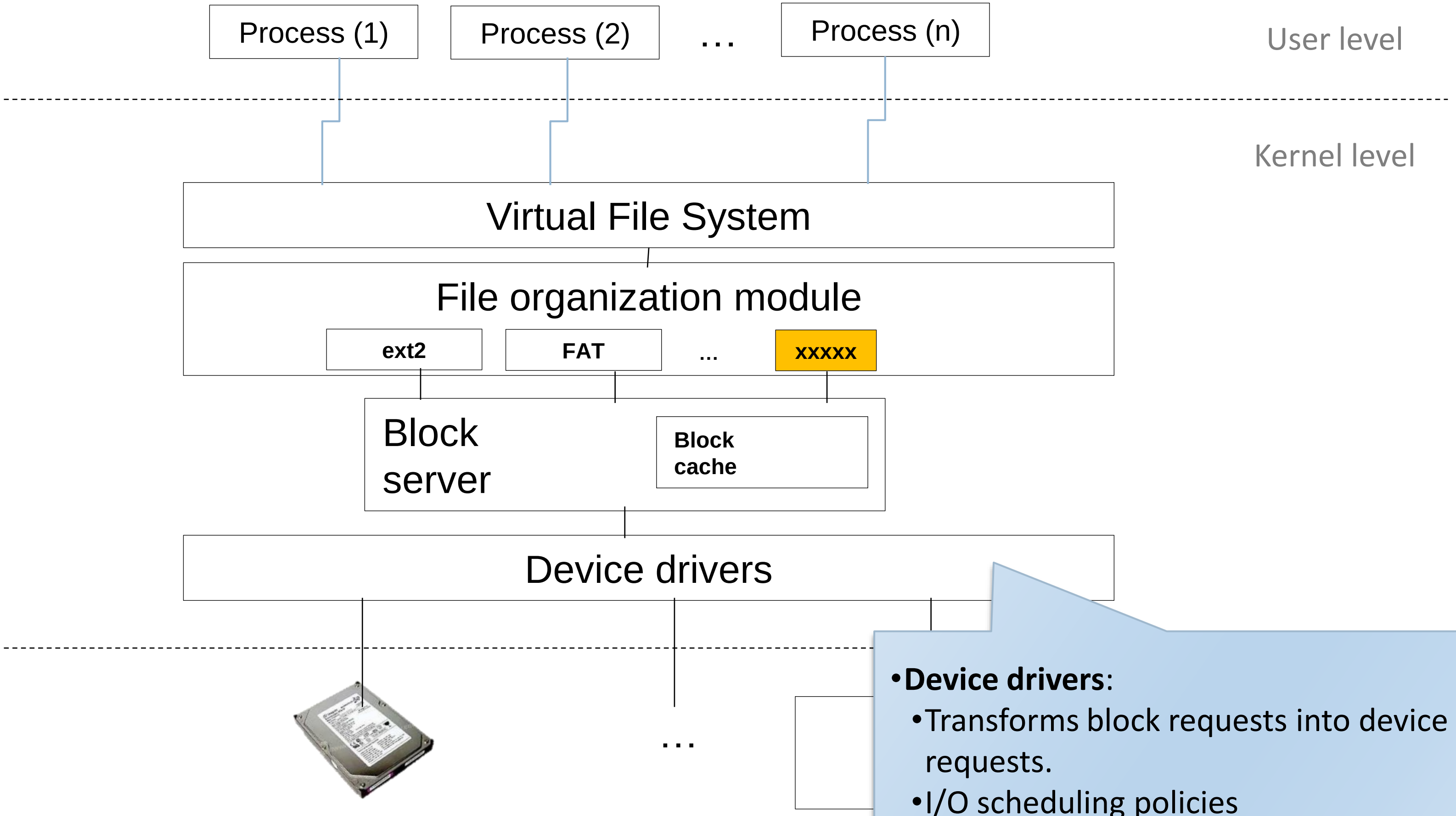
□ System software

□ **File System (manager)**

File management architecture…

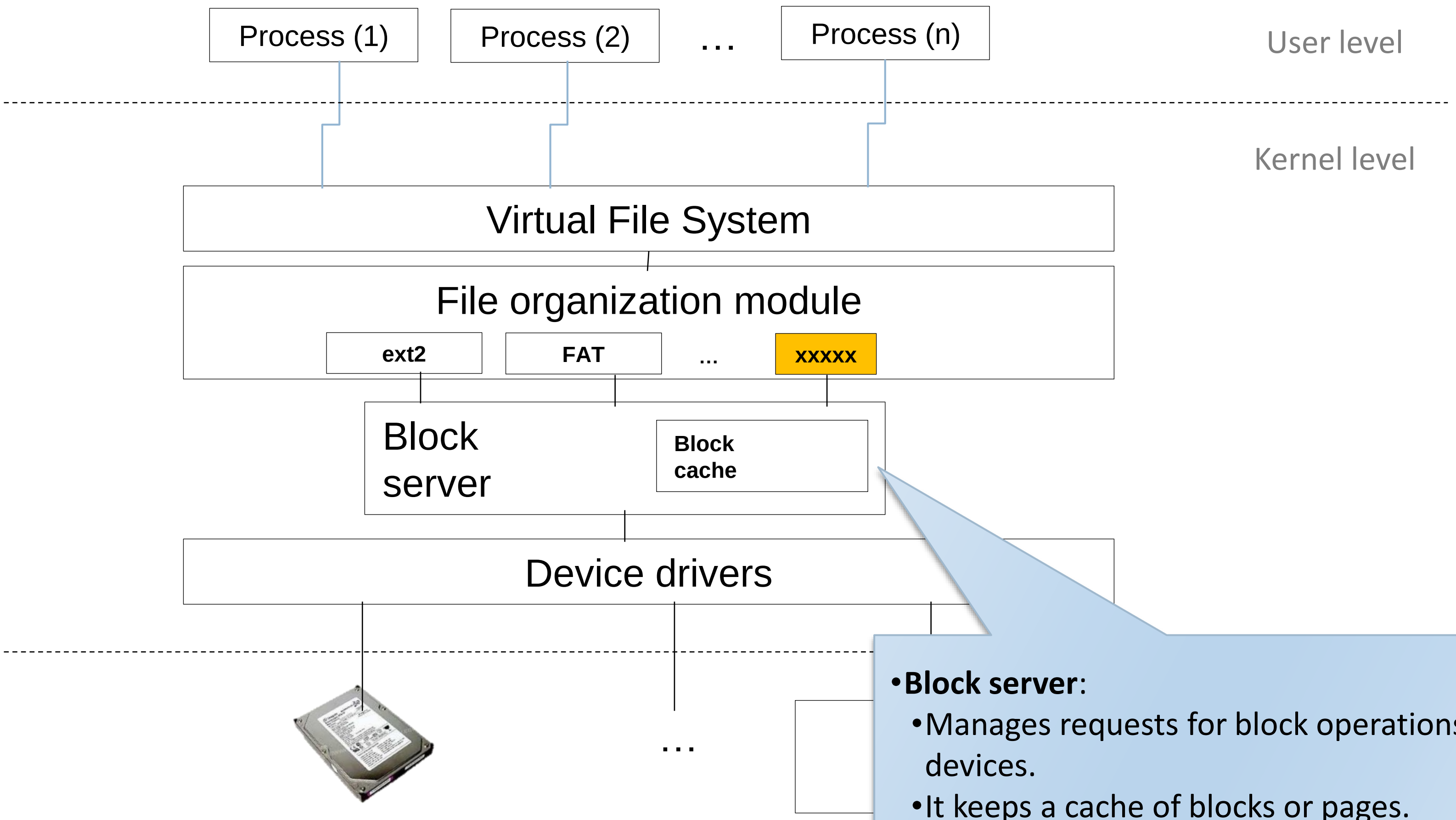


File management architecture…



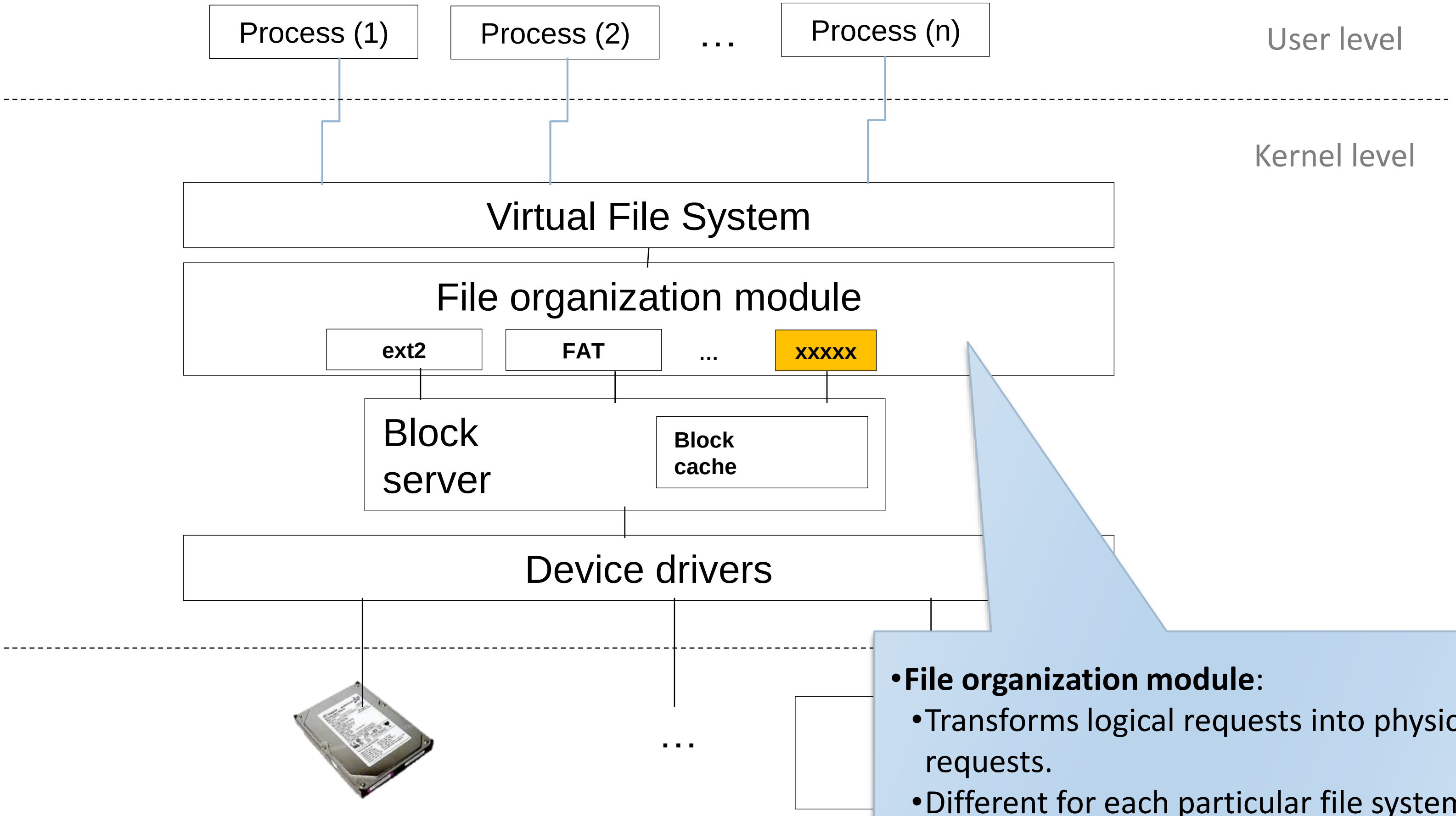
- **Device drivers**:
  - Transforms block requests into device requests.
  - I/O scheduling policies

File management architecture…



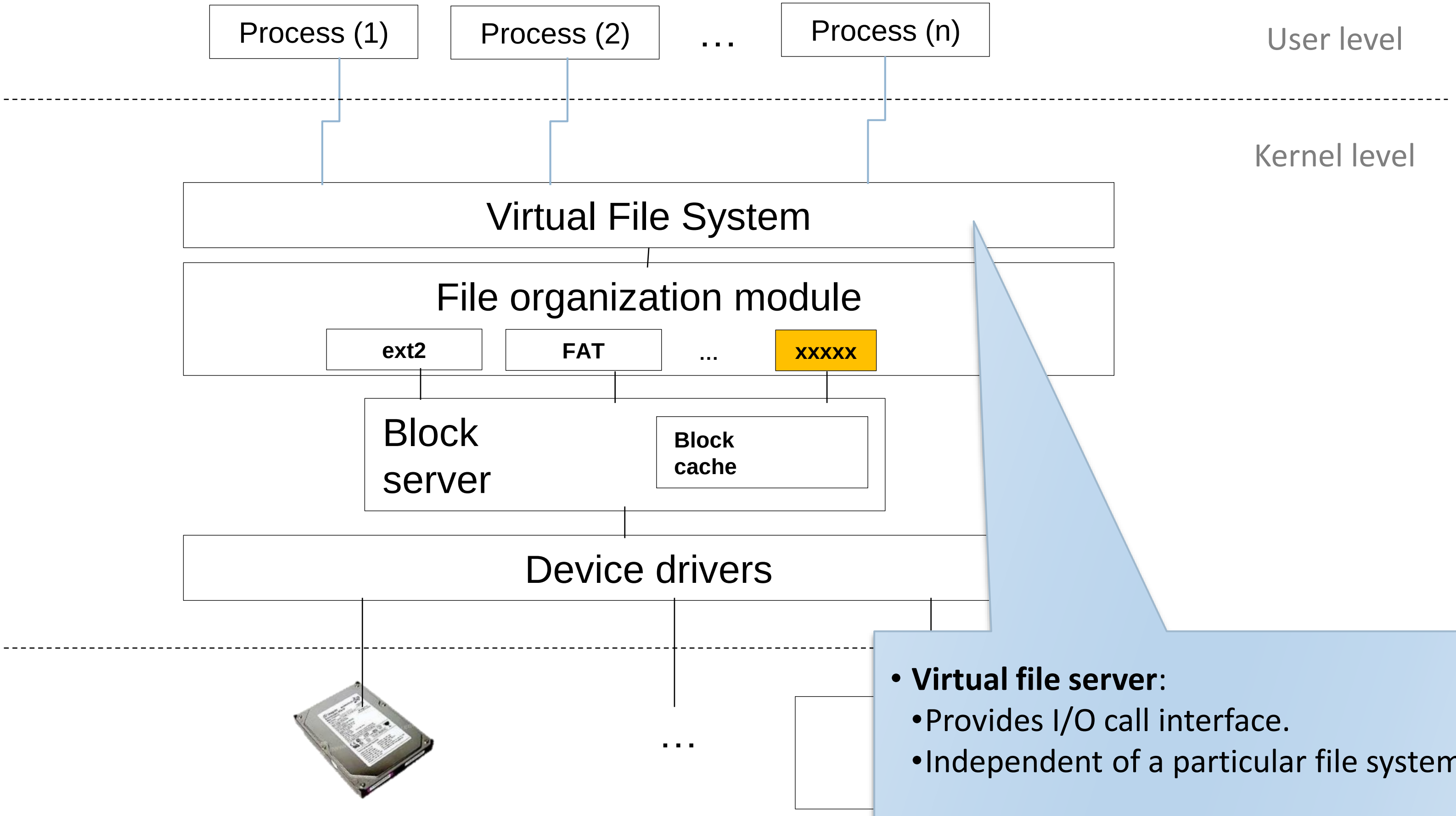
- **Block server**:
  - Manages requests for block operations on devices.
  - It keeps a cache of blocks or pages.

File management architecture…



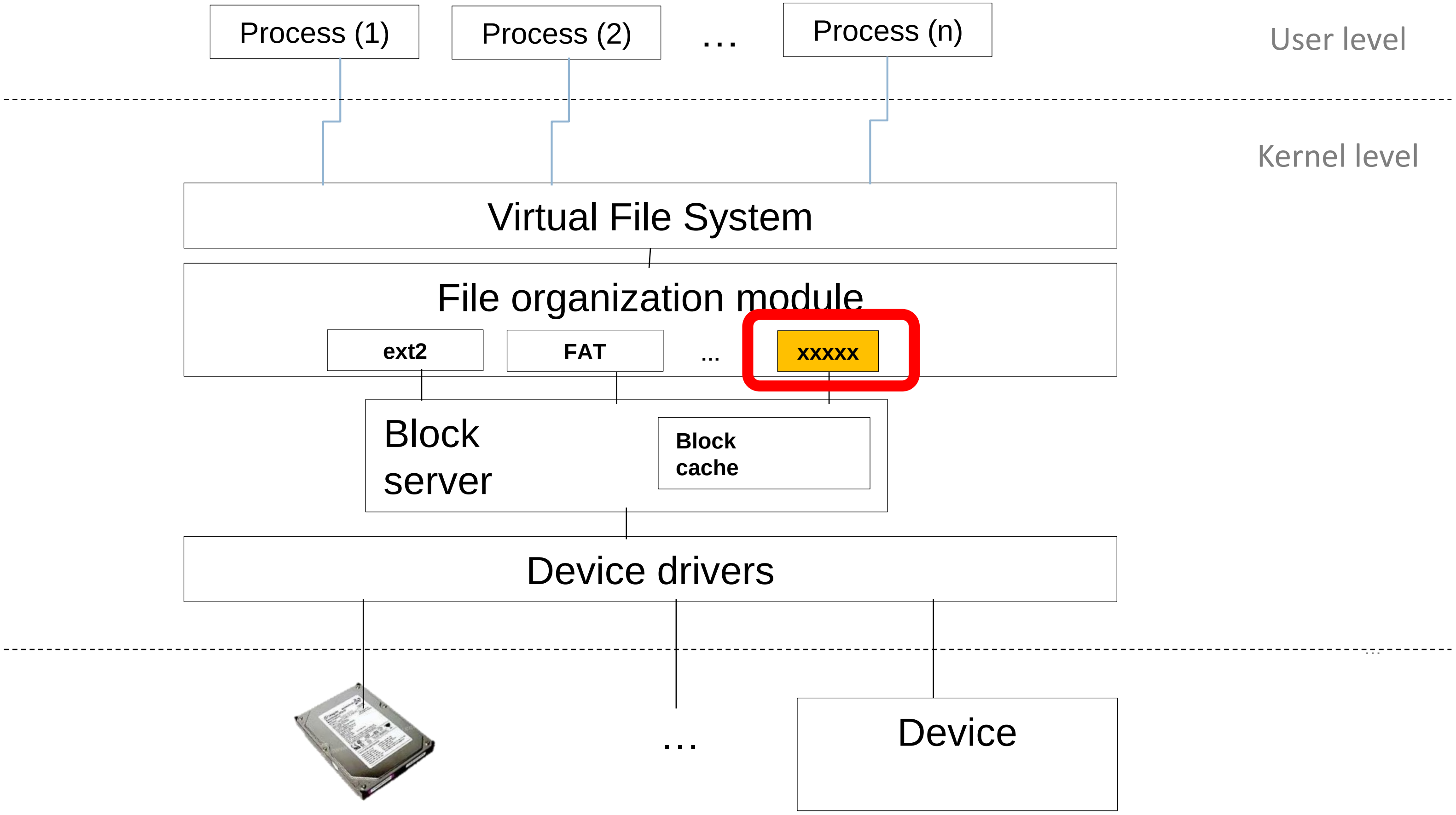
- **File organization module**:
  - Transforms logical requests into physical requests.
  - Different for each particular file system.

File management architecture…

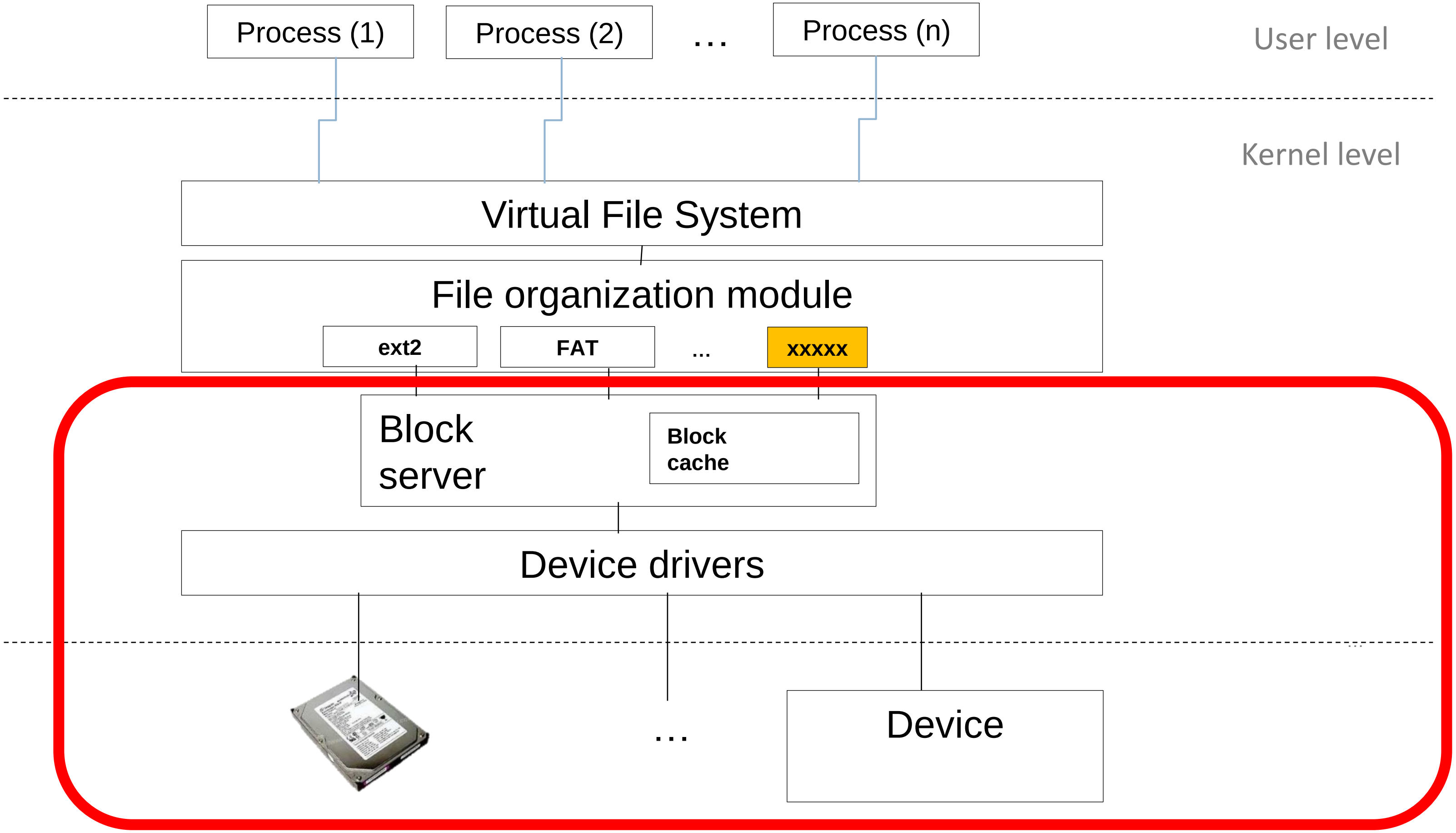


- **Virtual file server**:
  - Provides I/O call interface.
  - Independent of a particular file system.

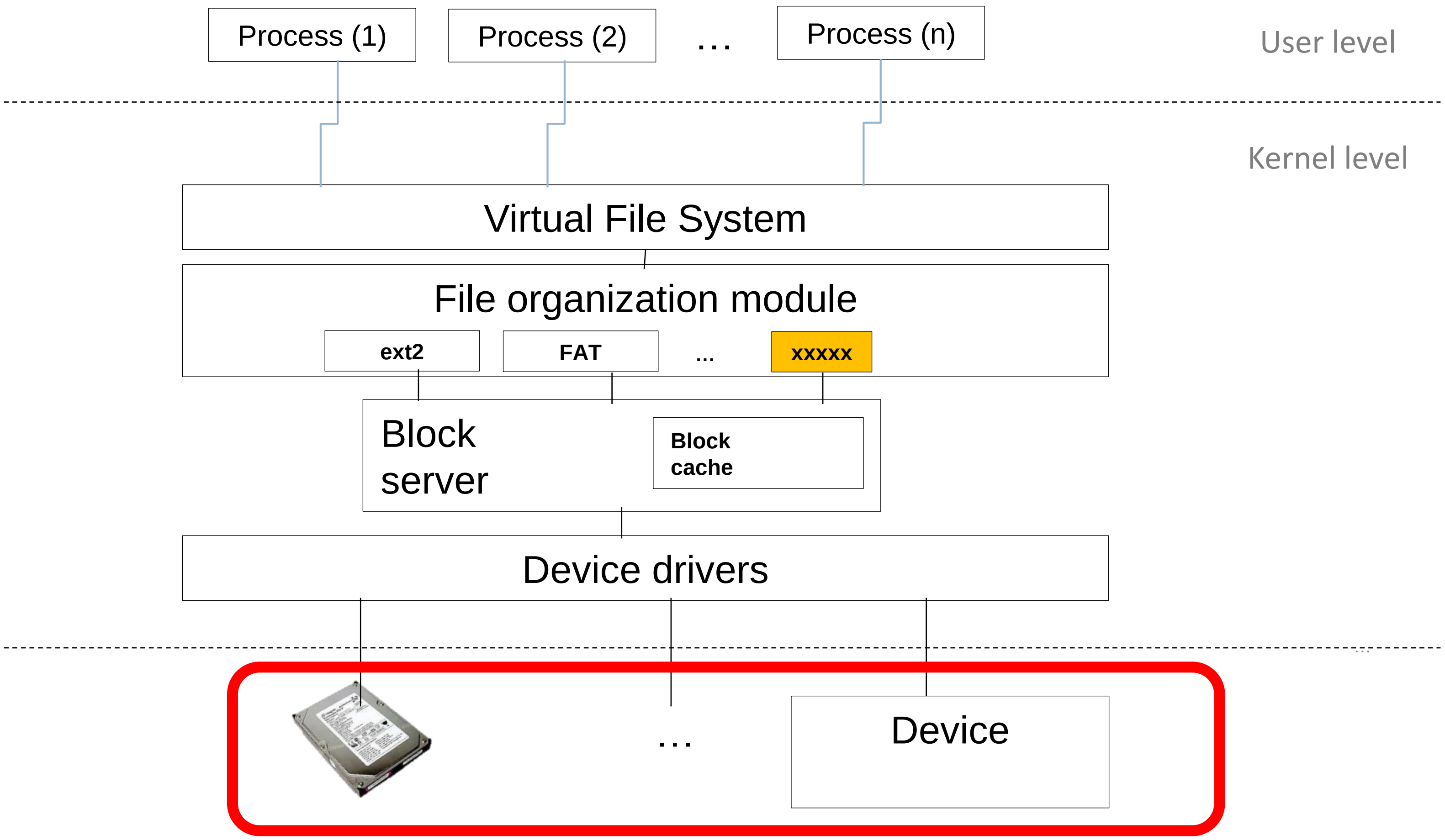
Destination (related to architecture)…
file system design and implementation



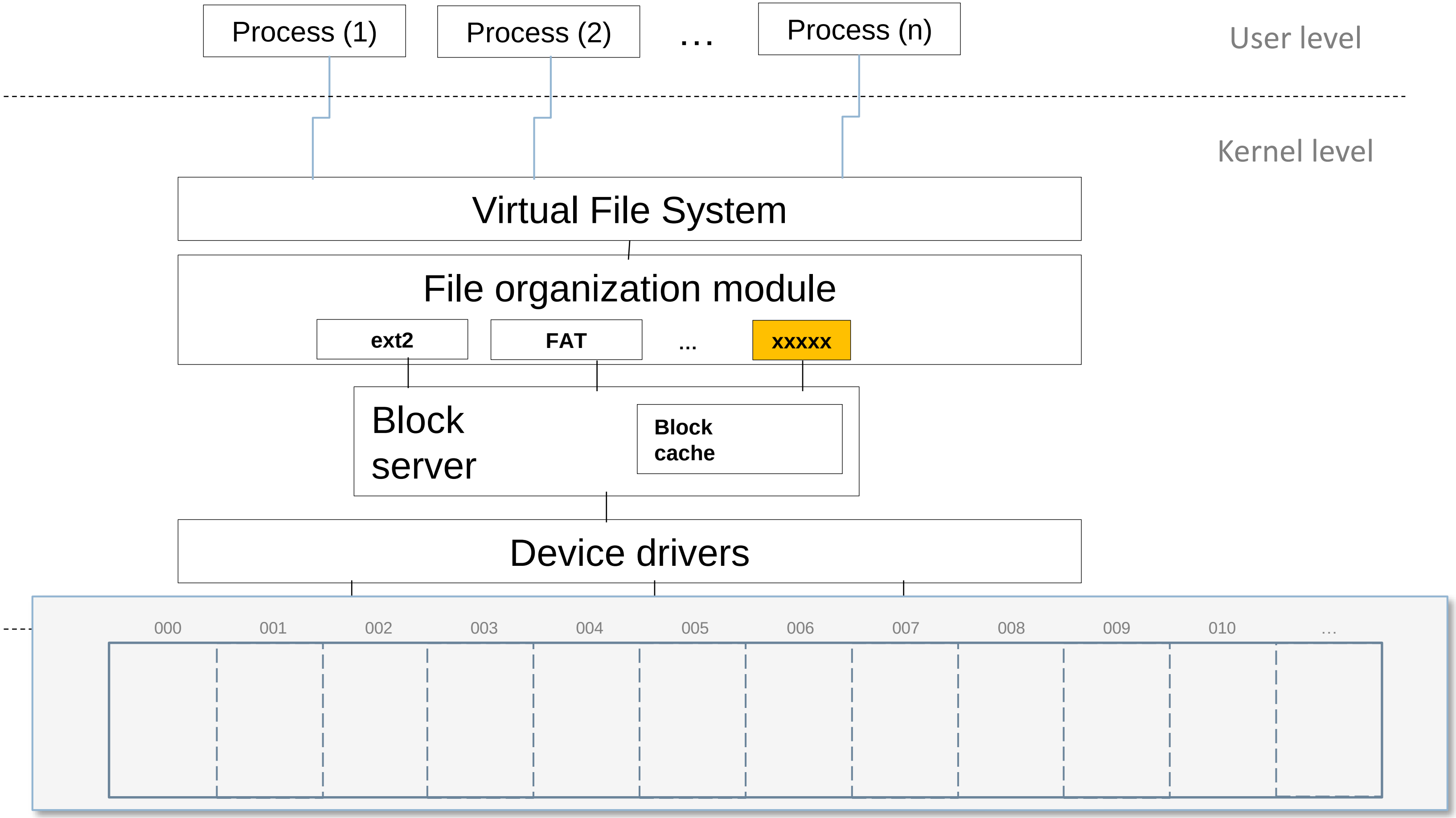
Origin (related to architecture)…
a) disk blocks + b) disk block cache



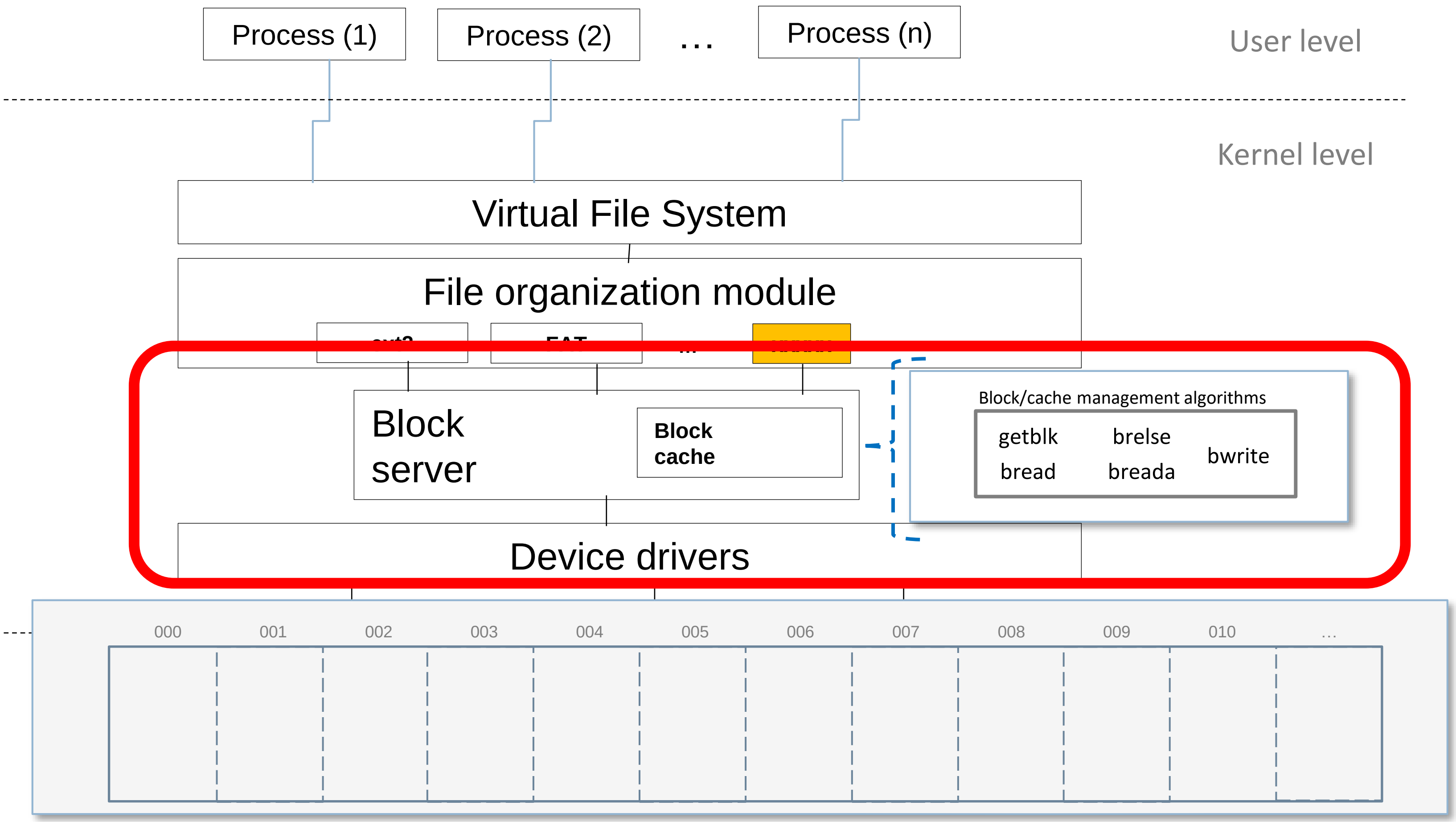
Origin (related to architecture)…
a) disk blocks



Origin (related to architecture)…
a) disk blocks

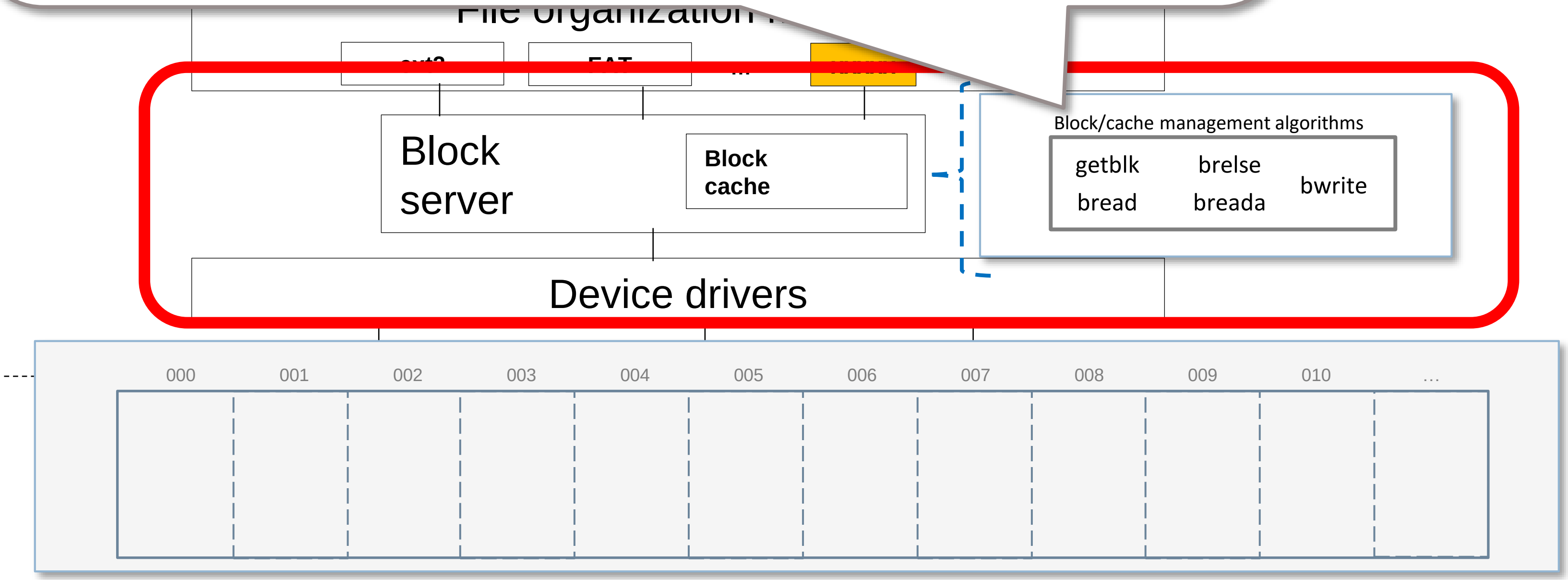


Origin (related to architecture)…
b) disk block cache



Origin (related to architecture)…
b) disk block cache

- **getblk**: searchs/allocates a block in cache (from a given v-node, offset and size).
- **brelse**: releases a block and adds it to the free list.
- **bwrite**: writes a cache block to disk.
- **bread**: reads a block from disk to cache.
- **breada**: reads 1 block (and the next) from disk to cache.



Block server

- It is in charge of:
  - Issue generic commands to read and write blocks to device handlers (using the device-specific routines).
  - Optimize I/O requests.
    - Ej.: block cache.
    - Can be integrated with virtual memory manager.
  - Provide a logical naming for the devices.
    - E.g.: /dev/hda3 (third partition of the first disk)

Block server

- General operation:
  - If the block is in cache
    - Copy content (+ update block usage metadata)
  - If the block is not in cache
    - Read the device block and store it in the cache
    - Copy content (and update metadata)
    - If the block has been written on (dirty)
      - Writing policy
    - If the cache is full, it is necessary to make room for it
      - Replacement policy

Block server

- General operation:

  - **read-ahead**:
    - Read a number of blocks after the required one and cached (improves performance on consecutive accesses)

  - Read the device block and store it in the cache
  - Copy content (and update metadata)
  - If the block has been written on (dirty)
    - Writing policy
  - If the cache is full, it is necessary to make room for it
    - Replacement policy

Block server

- **write-through:**
  - It is written each time the block is modified (– yield, + reliability)
- **write-back:**
  - Data are only written to disk when they are chosen for replacement due to lack of cache space (+ performance, – reliability)
- **delayed-write:**
  - Write to disk the modified data blocks in the cache periodically every certain time (30 seconds in UNIX) (compromise between previous)
- **write-on-close:**
  - When a file is closed, its blocks are dumped to disk..

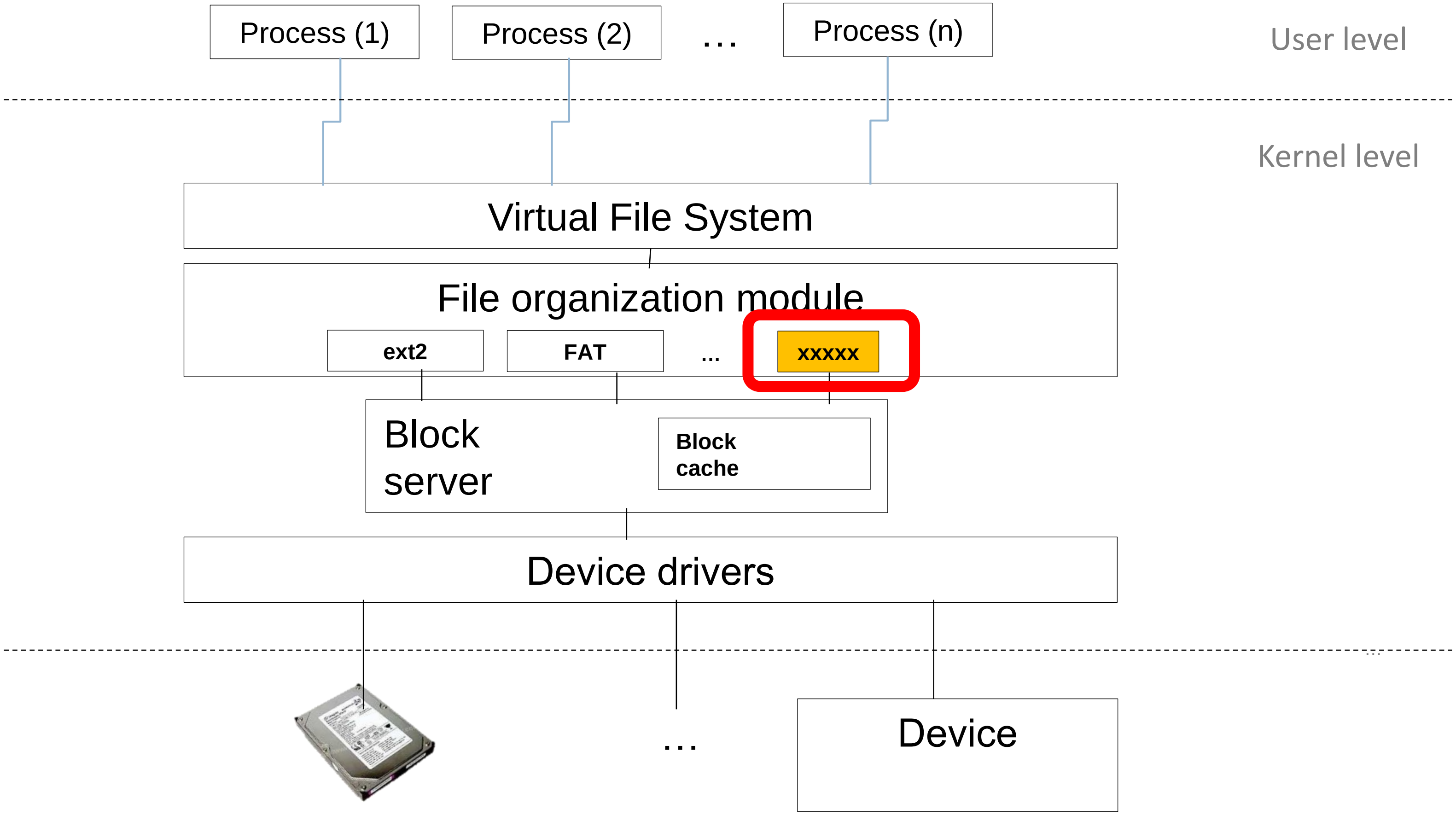
- If the been written on (dirty)
  - Writing policy
- If the cache is full, it is necessary to make room for it
  - Replacement policy

Block server

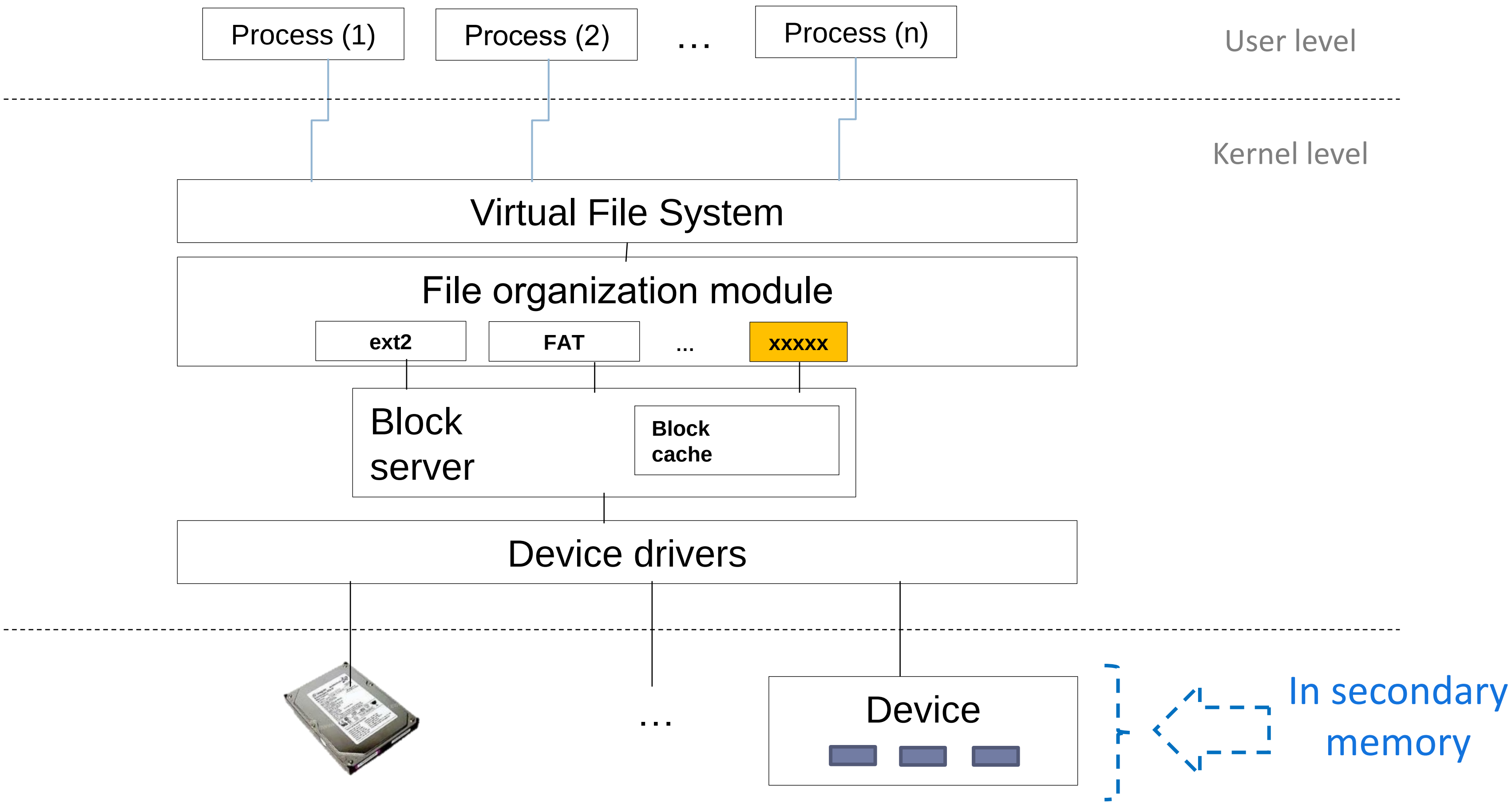
- General operation:
  - If the block is in cache
    - Copy content (+ update block usage metadata)
  - If the block is not in cache
    - Read the device block and store it in the cache
    - If the [illegible] full, it is necessary to make room for it
      - Replacement policy

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

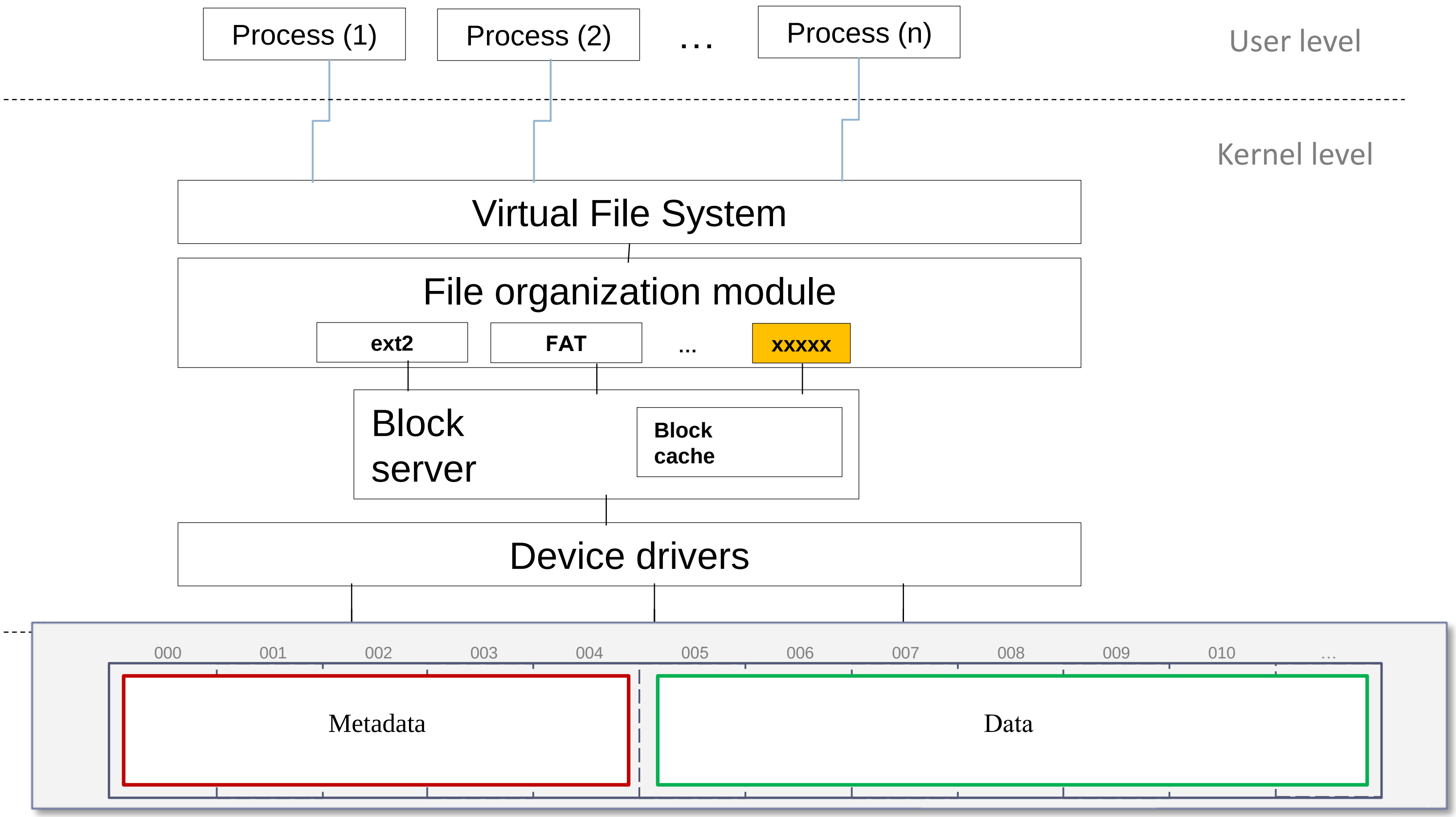
Destination (related to architecture)…
file system design and implementation



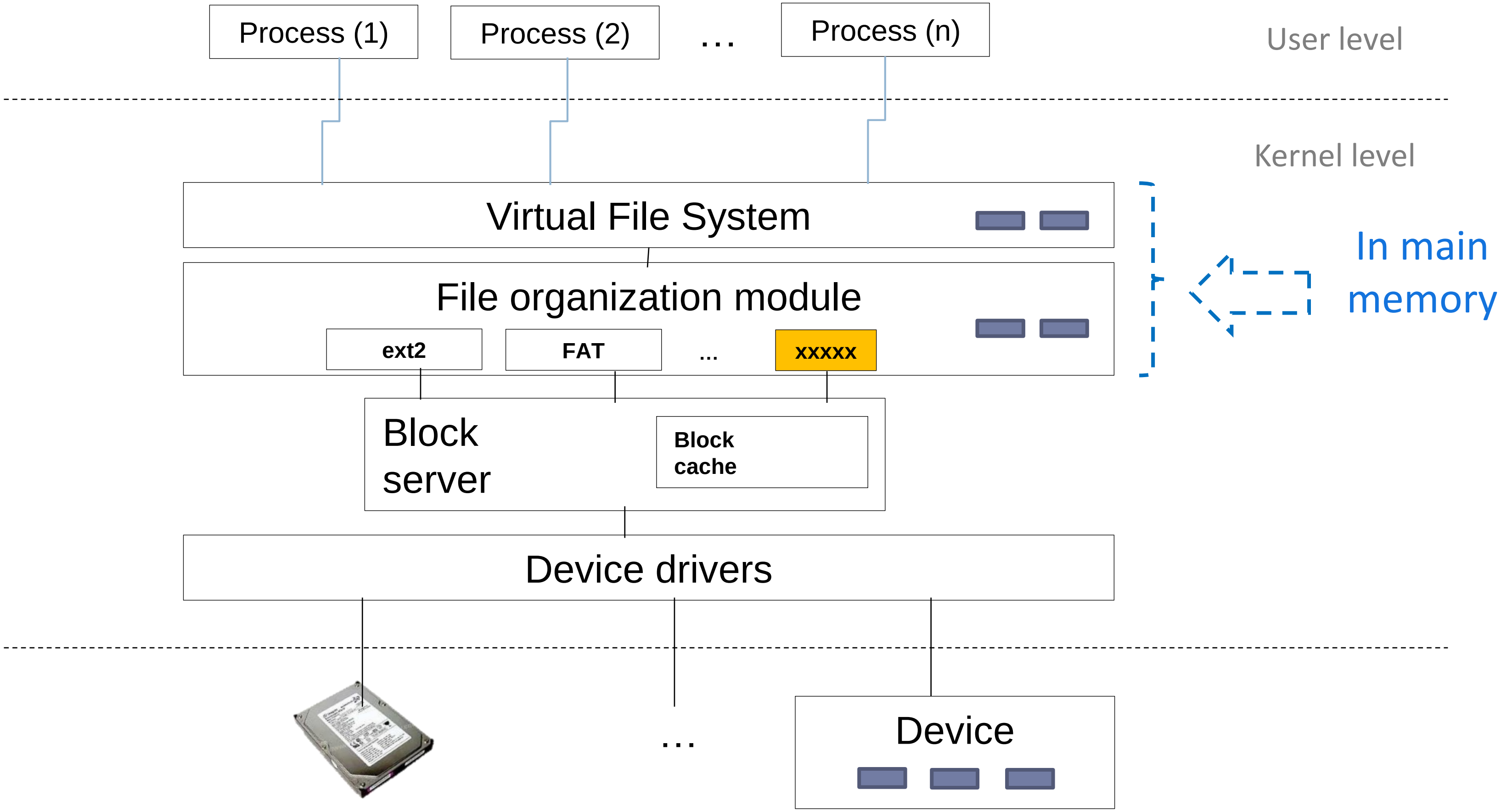
Aspects to be design (related to architecture)… (1) Data structures on disk…



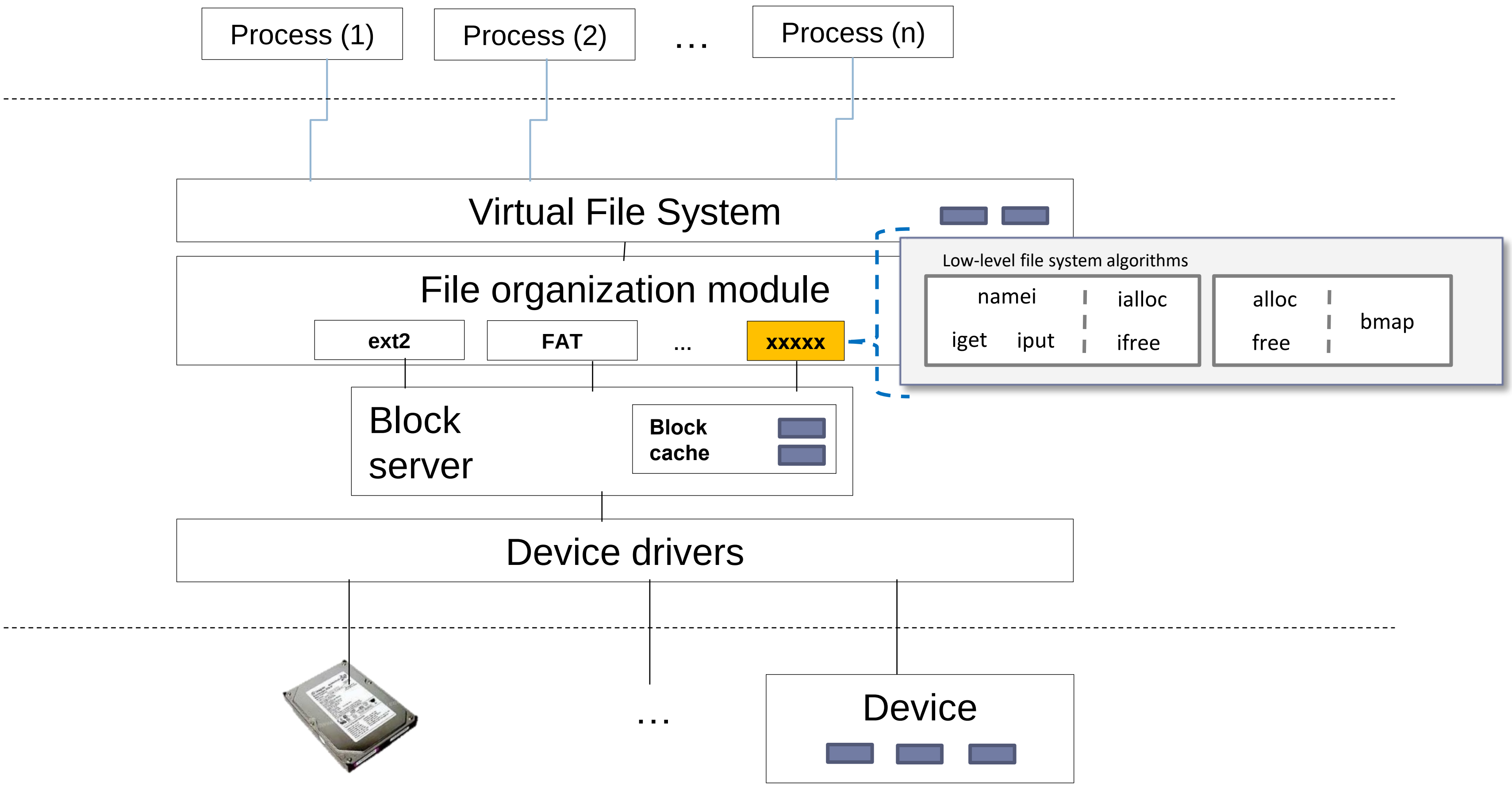
Aspects to be design (related to architecture)…
(1) Data structures on disk…



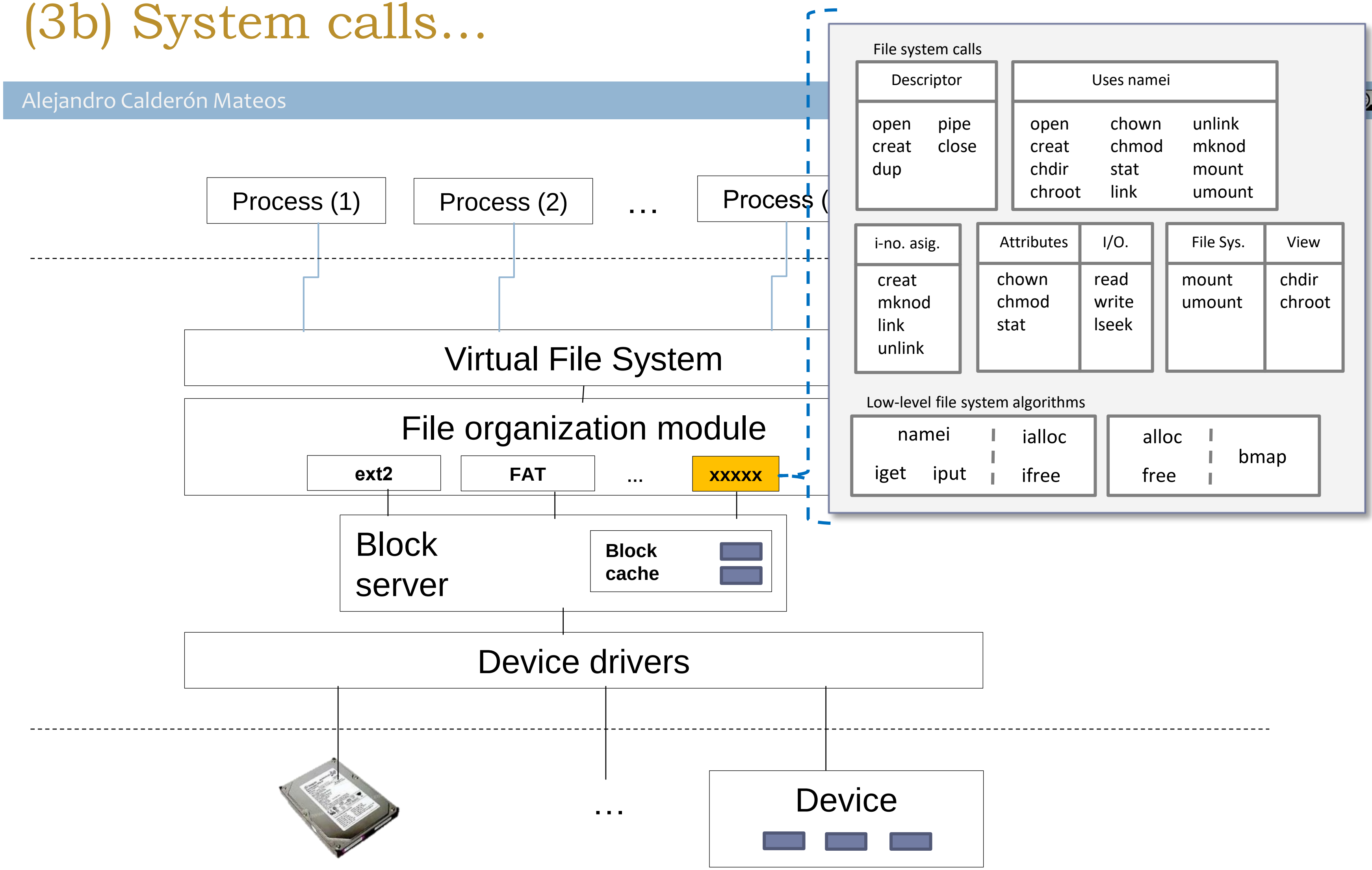
Aspects to be design (related to architecture)…
(2) Data structures in memory…



Aspects to be design (related to architecture)… (3a) Management of disk/memory structures …



Aspects to be design (related to architecture)…
(3b) System calls…



File system calls

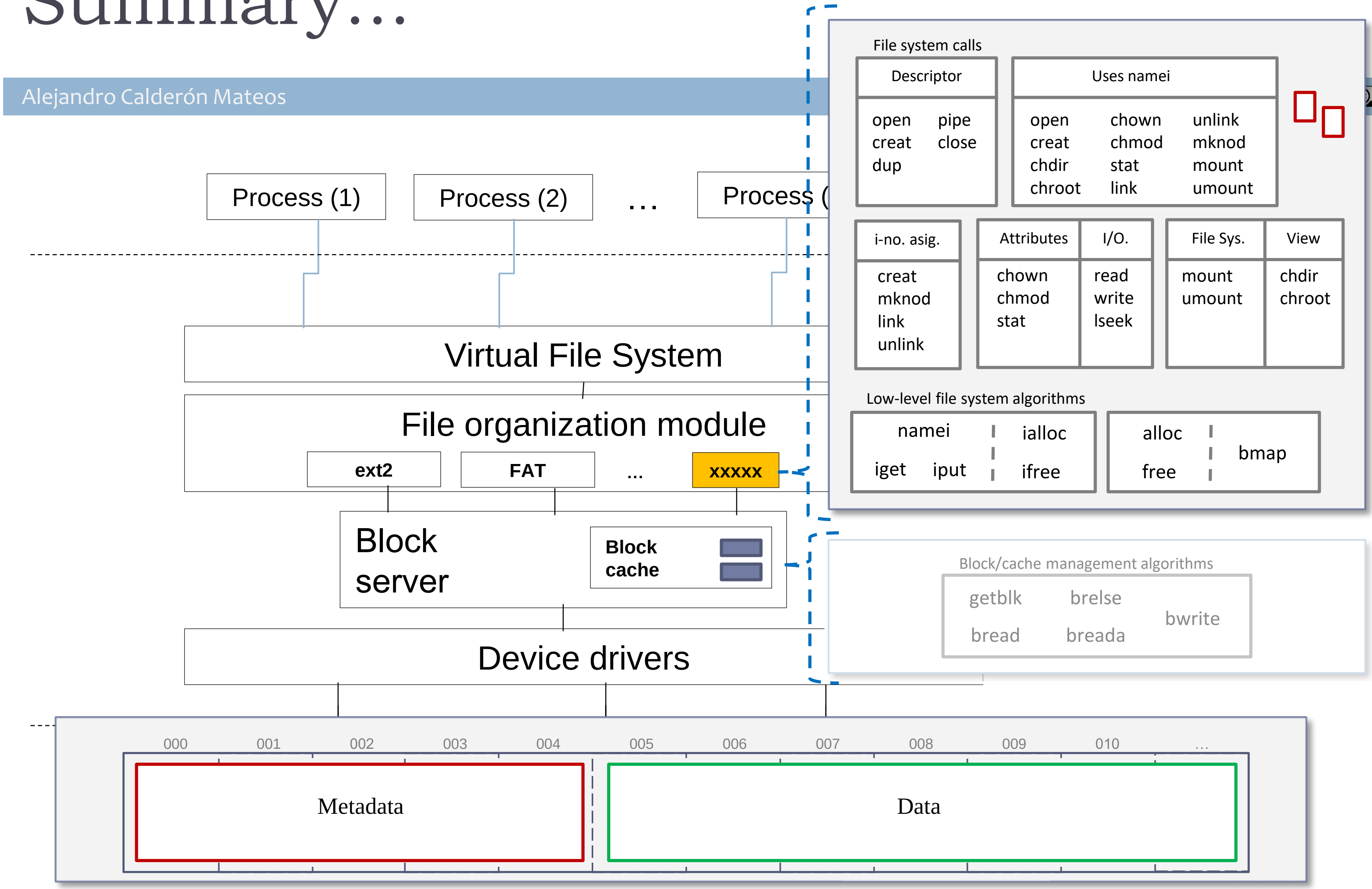
Descriptor		Uses namei		
open creat dup	pipe close	open creat chdir chroot	chown chmod stat link	unlink mknod mount umount

i-no. asig.	Attributes	I/O.	File Sys.	View
creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

Low-level file system algorithms

namei iget iput	ialloc ifree	alloc free	bmap

Summary…



File system calls

Descriptor	
open creat dup	pipe close

Uses namei		
open creat chdir chroot	chown chmod stat link	unlink mknod mount umount

i-no. asig.	Attributes	I/O.	File Sys.	View
creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

Low-level file system algorithms

namei		ialloc	alloc	bmap
iget	iput	ifree	free	

Block/cache management algorithms

getblk	brelse	bwrite
bread	breada	

Simplified summary…

File system calls

Descriptor	Uses namei	i-no. asig.	Attributes	I/O.	File Sys.	View
open pipe creat close dup	open chown unlink creat chmod mknod chdir stat mount chroot link umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

xx

Low-level file system algorithms

namei iget iput	ialloc ifree	alloc free	bmap

Block/cache management algorithms

getblk	brelse	bread	breada	bwrite

d-entries

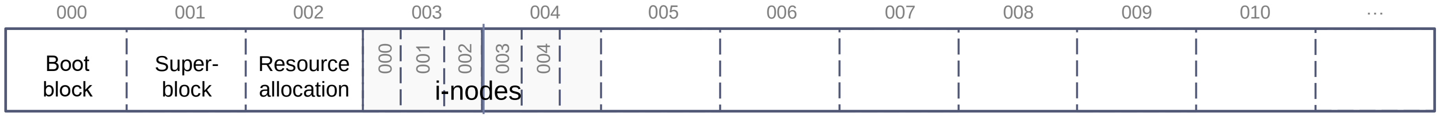
mounted

file r/w pointers

open files

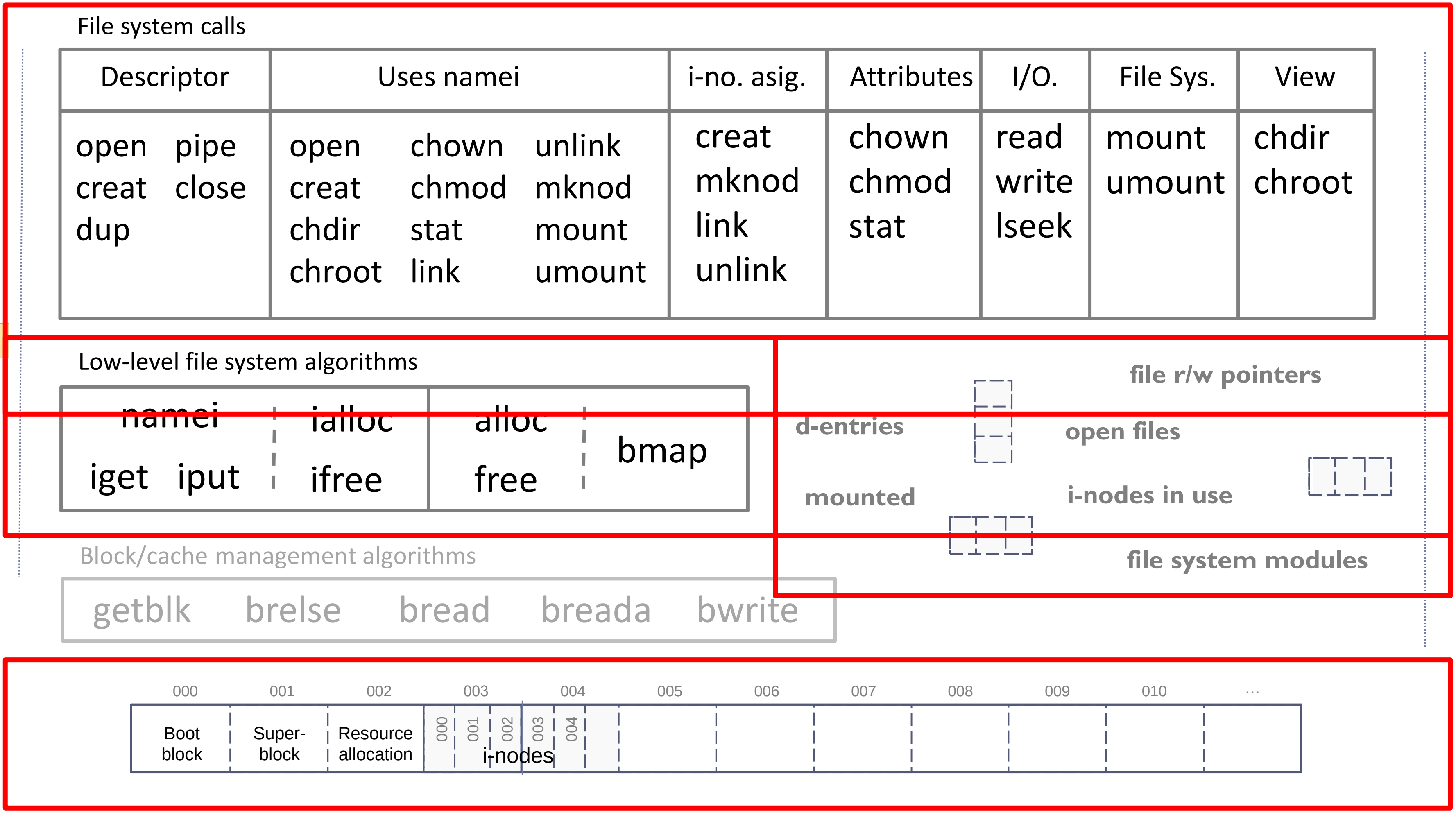
i-nodes in use

file system modules



Elements to be analyzed (1, 2, 3a y 3b)

xx



File system calls

Descriptor	Uses namei	i-no. asig.	Attributes	I/O.	File Sys.	View
open pipe creat close dup	open chown unlink creat chmod mknod chdir stat mount chroot link umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

Low-level file system algorithms

namei
iget iput

ialloc
ifree

alloc
free

bmap

Block/cache management algorithms

getblk brelse bread breada bwrite

file r/w pointers

d-entries

open files

mounted

i-nodes in use

file system modules

(1) Data structures on disk...

XX

File system calls

Descriptor	Uses namei	i-no. asig.	Attributes	I/O.	File Sys.	View
open pipe creat close dup	open chown unlink creat chmod mknod chdir stat mount chroot link umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

Low-level file system algorithms

namei iget iput	ialloc ifree	alloc free	bmap

d-entries

mounted

file r/w pointers

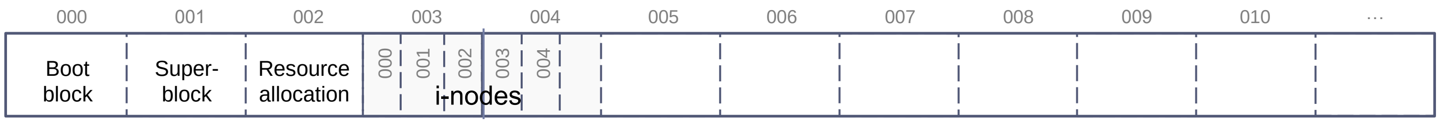
open files

i-nodes in use

file system modules

Block/cache management algorithms

getblk	brelse	bread	breada	bwrite

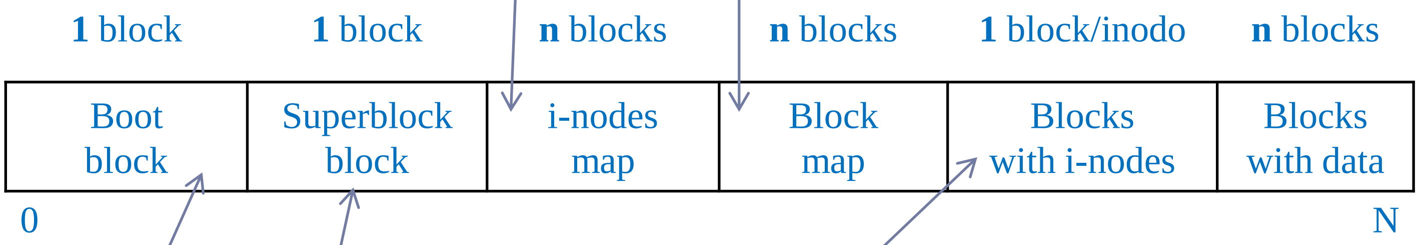


Example of disk organization

https://github.com/acaldero/nanofs

char **imap**[numInodes] ;   /* **1**00…0 (used: imap[x]=1 | free: imap[x]=0)

char **bmap**[numBlocksData] ;   /* 000…0 (used: bmap[x]=1 | free: bmap[x]=0)



000000…000

```
typedef struct {
    unsigned int type;                    /*  T_FILE o T_DIRECTORY */
    char nombre[200];                     /* Name of the associated file/directory */
    unsigned int inodosContents[200];     /* type==dir: list of directory inodes */
    unsigned int tamanyo;                 /* Current file size in bytes */
    unsigned int blockDirecto;            /* Index of the direct block */
    unsigned int blockIndirecto;          /* Index of the indirect block */
    char padding[PADDING_INODO];           /* Padding field to fill one block */
} TypeDiskInode;
```

```
typedef struct {
    unsigned int numMagico;              /* Superblock magic number: 0x000D5500 */
    unsigned int numBlocksInodeMap;      /* Number of inodes map blocks */
    unsigned int numBlocksDataMap;       /* Number of data    map blocks */
    unsigned int numInodes;              /* Number of inodes in the device */
    unsigned int firstInode;             /* Index of the first block with inodes */
    unsigned int numBlocksData;          /* Number of data blocks in the device */
    unsigned int primerBloqueData;       /* Index of the first data block */
    unsigned int tamDevice;              /* Total device size (in bytes) */
    char padding[PADDING_SB];            /* Padding field to fill one block) */
} TypeSuperblock;
```

(2) Data structures on memory…

xx

File system calls

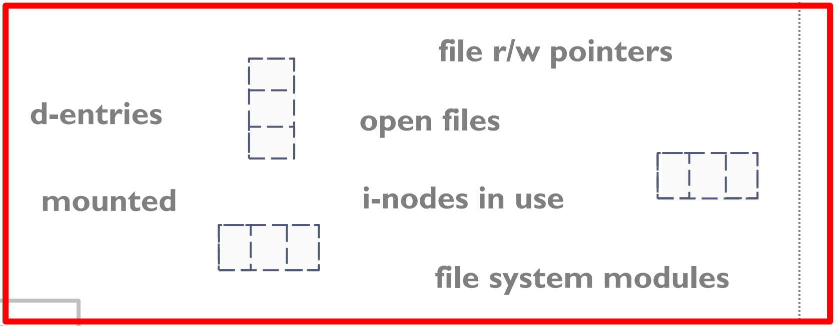
Descriptor	Uses namei			i-no. asig.	Attributes	I/O.	File Sys.	View
open pipe creat close dup	open creat chdir chroot	chown chmod stat link	unlink mknod mount umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

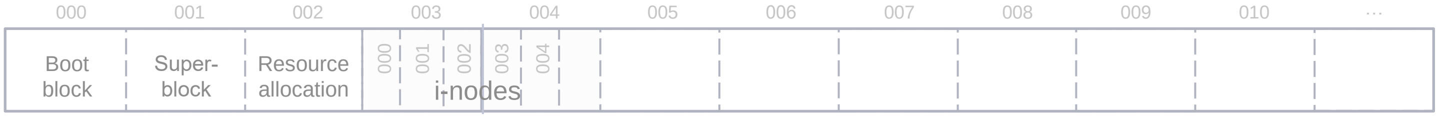
Low-level file system algorithms

namei iget iput	ialloc ifree	alloc free	bmap

Block/cache management algorithms

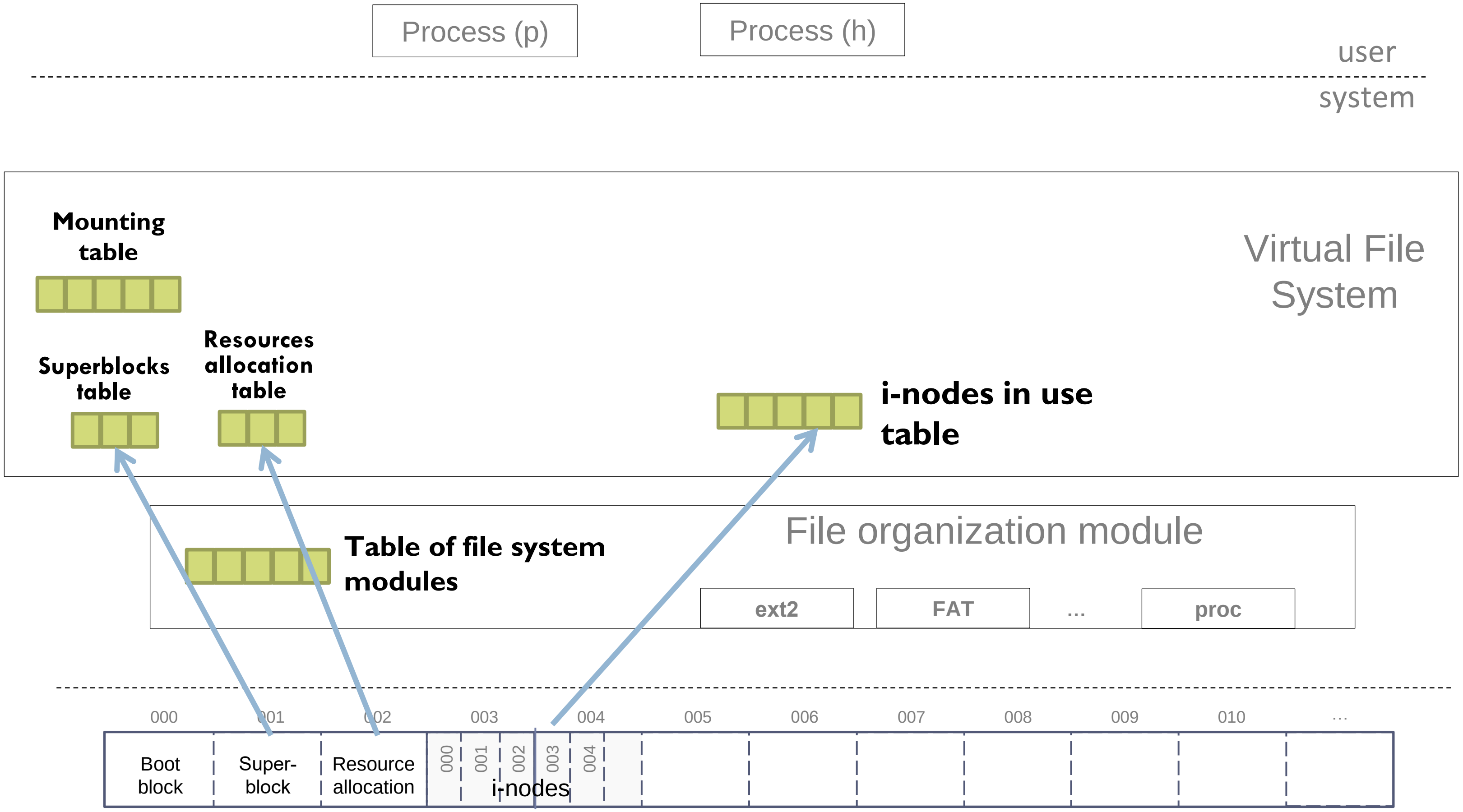
getblk brelse bread breada bwrite





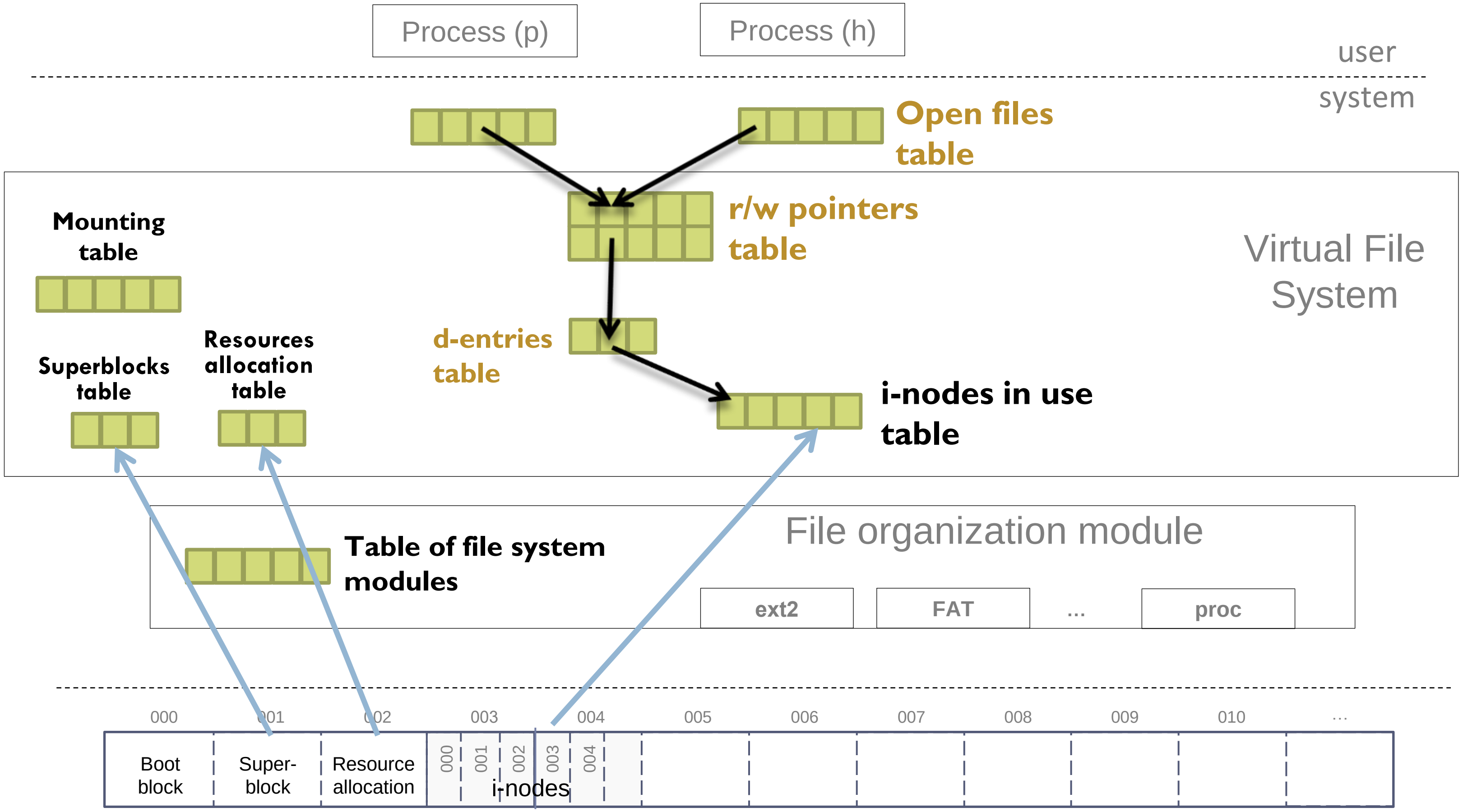
Main management structures

main metadata on disk…



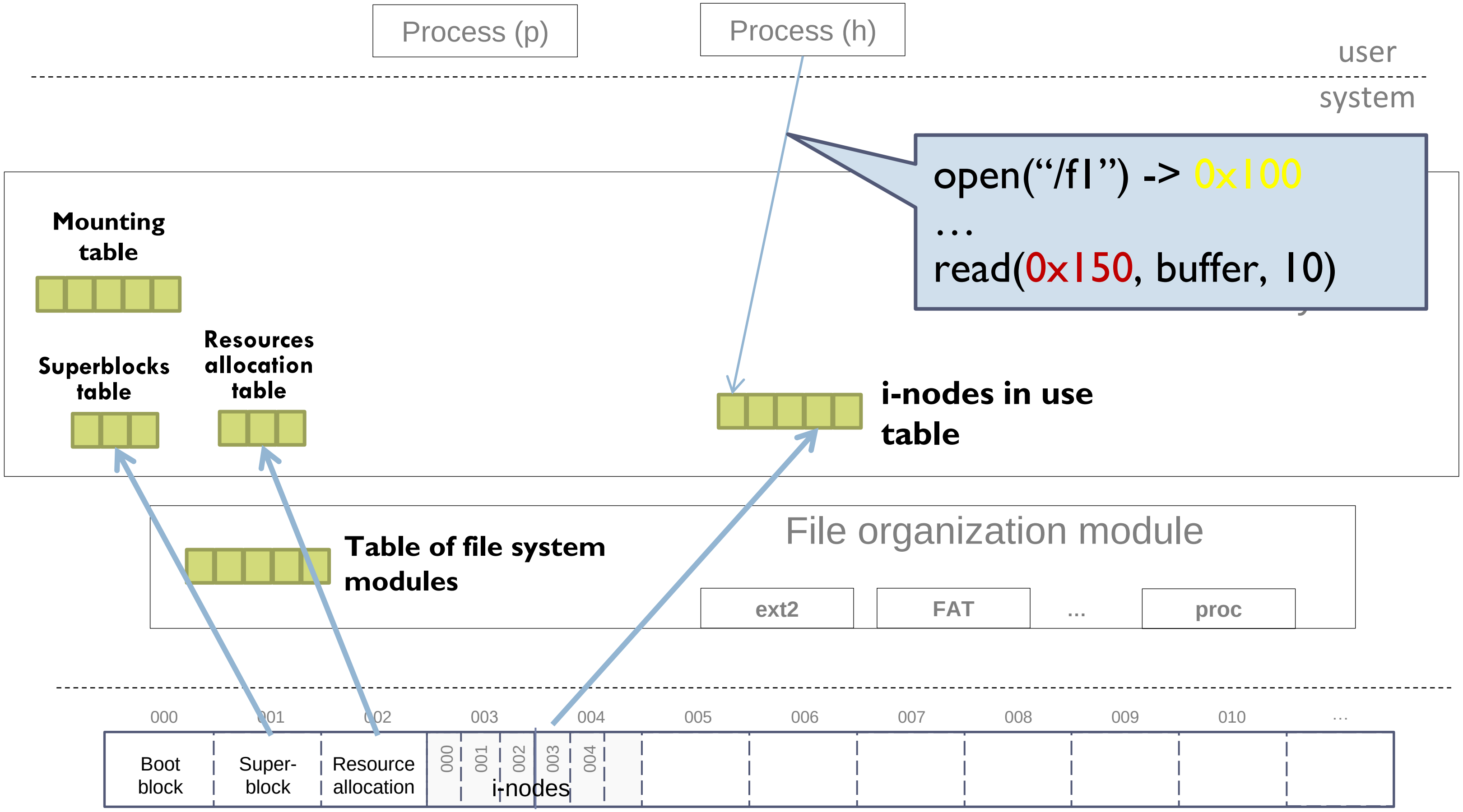
Main management structures

main metadata on disk… + 3



Main management structures

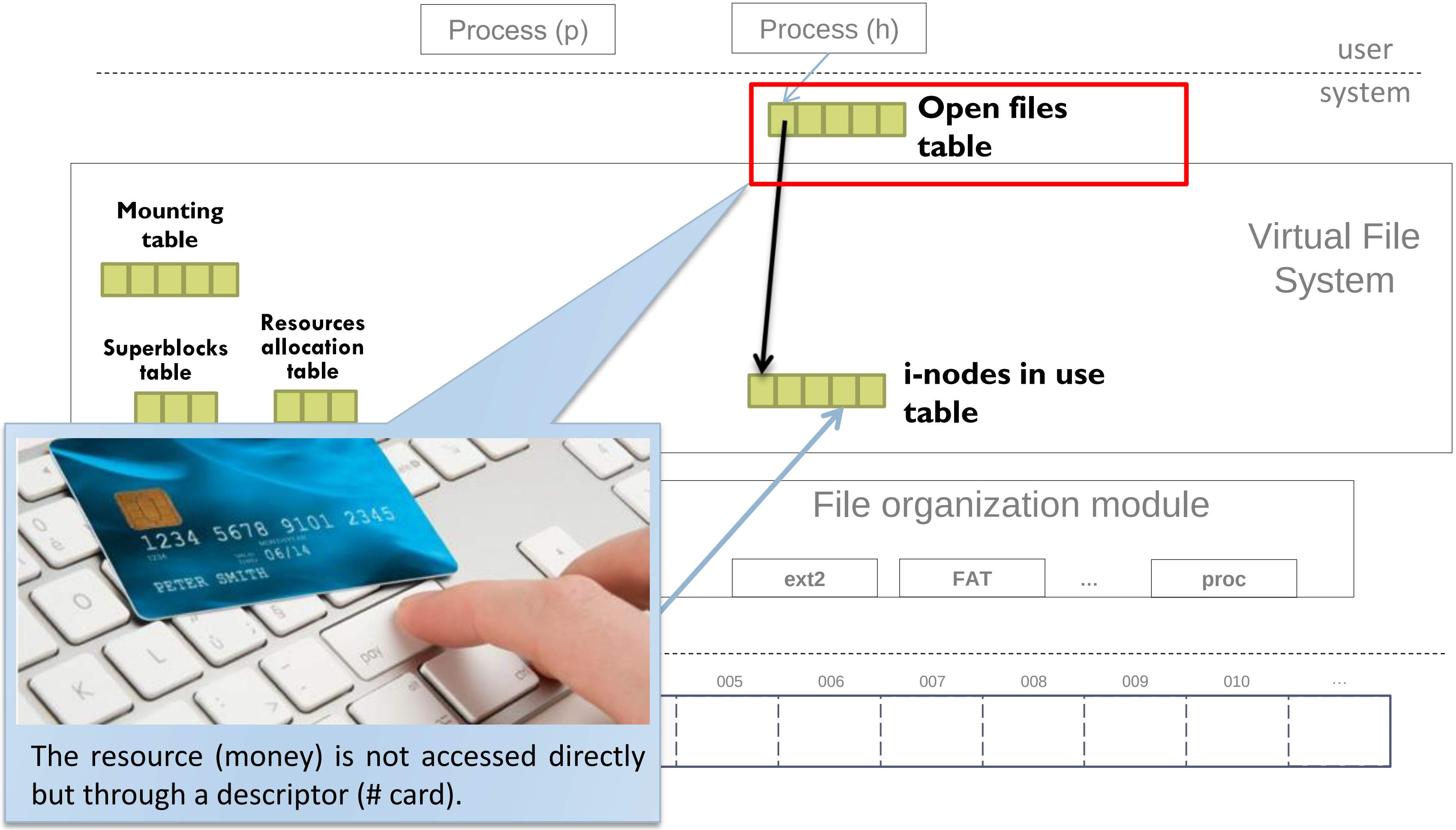
secure API interface?



open("/f1") -> 0x100
…
read(0x150, buffer, 10)

Main management structures

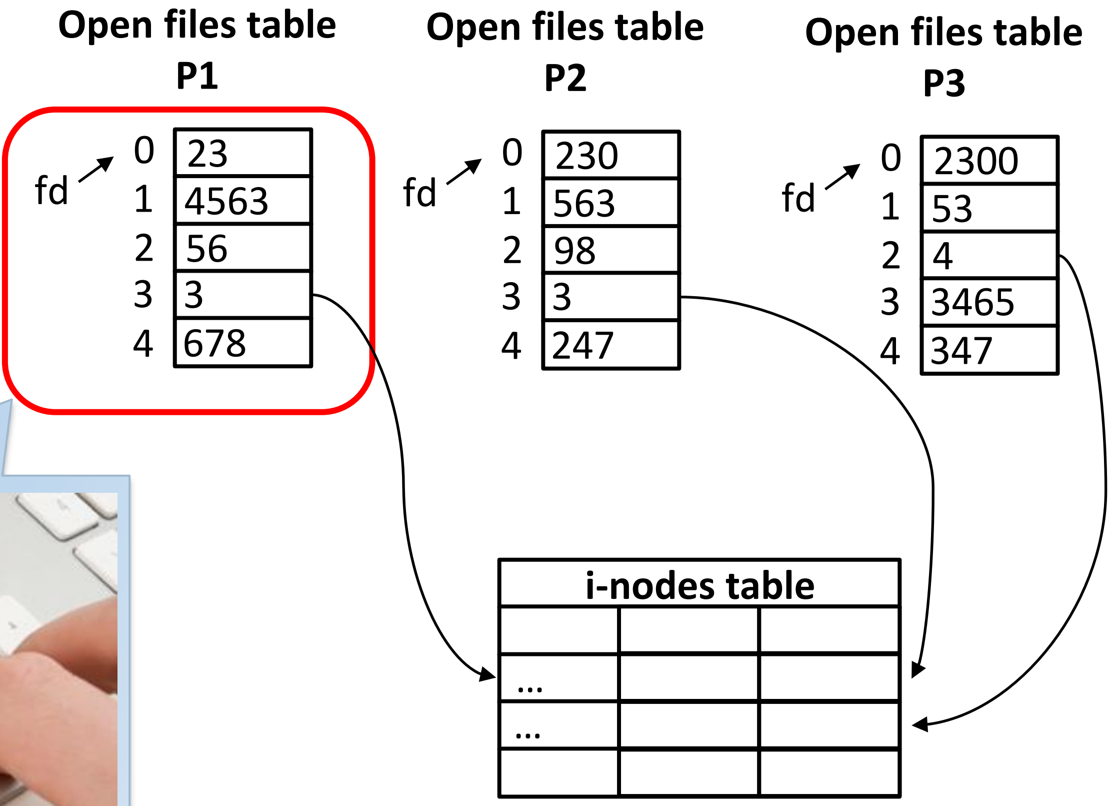
open files table: secure interface



The resource (money) is not accessed directly but through a descriptor (# card).

Main management structures

open files table: secure interface



Open files table P1

fd	
0	23
1	4563
2	56
3	3
4	678

Open files table P2

fd	
0	230
1	563
2	98
3	3
4	247

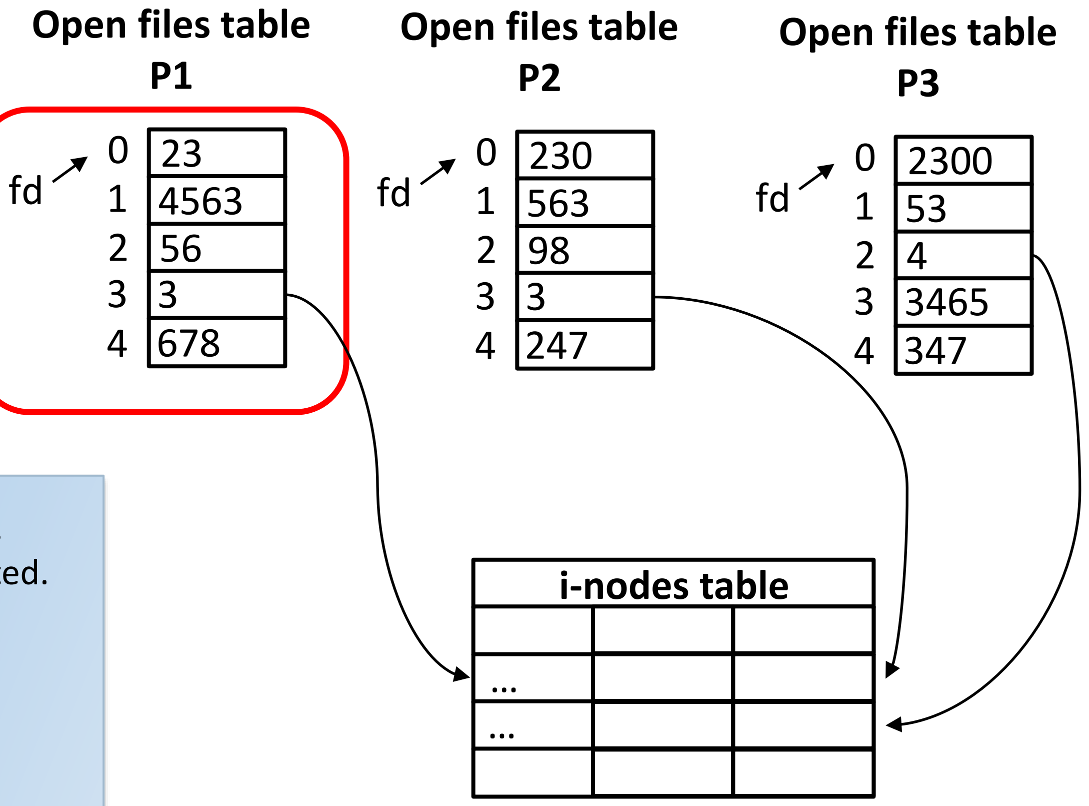
Open files table P3

fd	
0	2300
1	53
2	4
3	3465
4	347

The resource (money) is not accessed directly but through a descriptor (# card).

Main management structures

open files table: secure interface



Open files table P1

fd	
0	23
1	4563
2	56
3	3
4	678

Open files table P2

fd	
0	230
1	563
2	98
3	3
4	247

Open files table P3

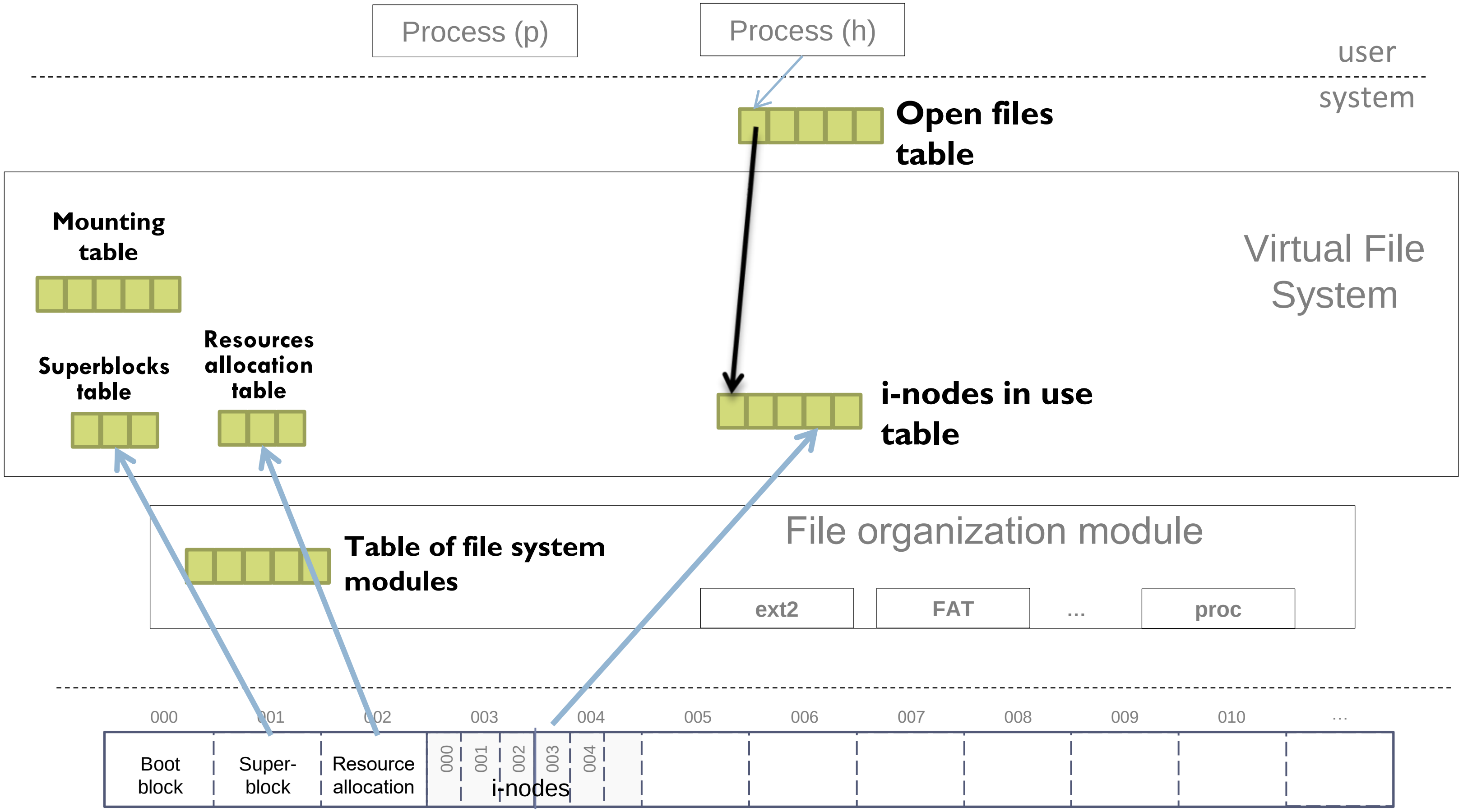
fd	
0	2300
1	53
2	4
3	3465
4	347

i-nodes table

…		
…		

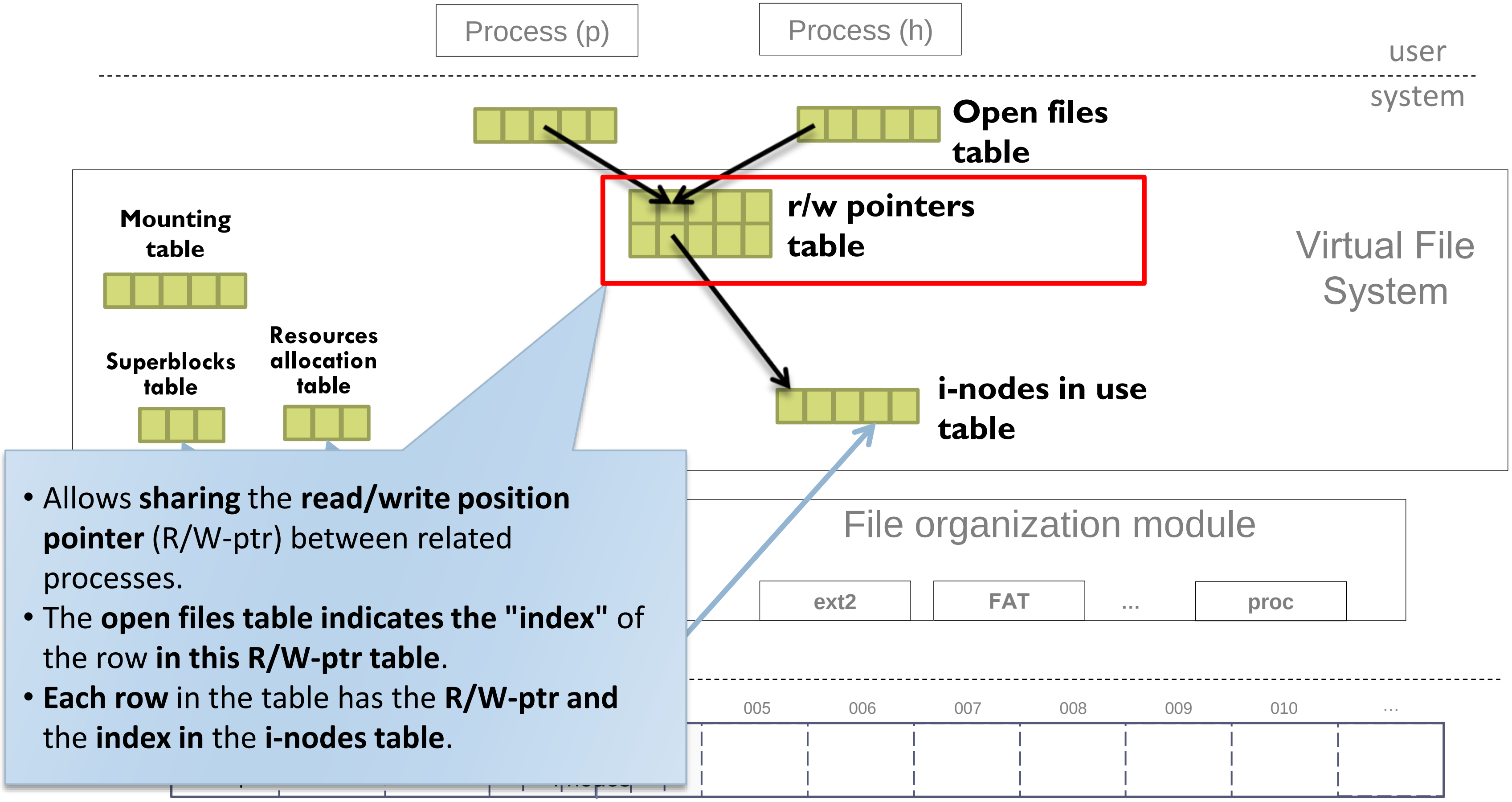
- Table **included in the BCP of the process**.
  - When fork() is performed, it is duplicated.
- Table with **one entry per open file**.
  - 0, 1 and 2 used by default.
- **Number of rows limits** the **maximum number of open files per process**.
- The table **is filled in orderly fashion**:
  - open/creat/dup: search first free entry.
  - close: marks entry as free.

Main management structures
open files table: secure interface



Main management structures

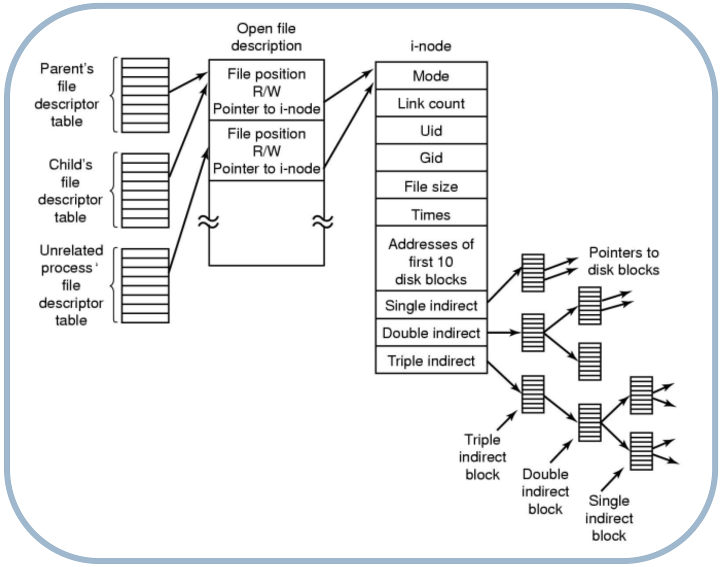
table of file r/w pointers: sharing r/w ptr.

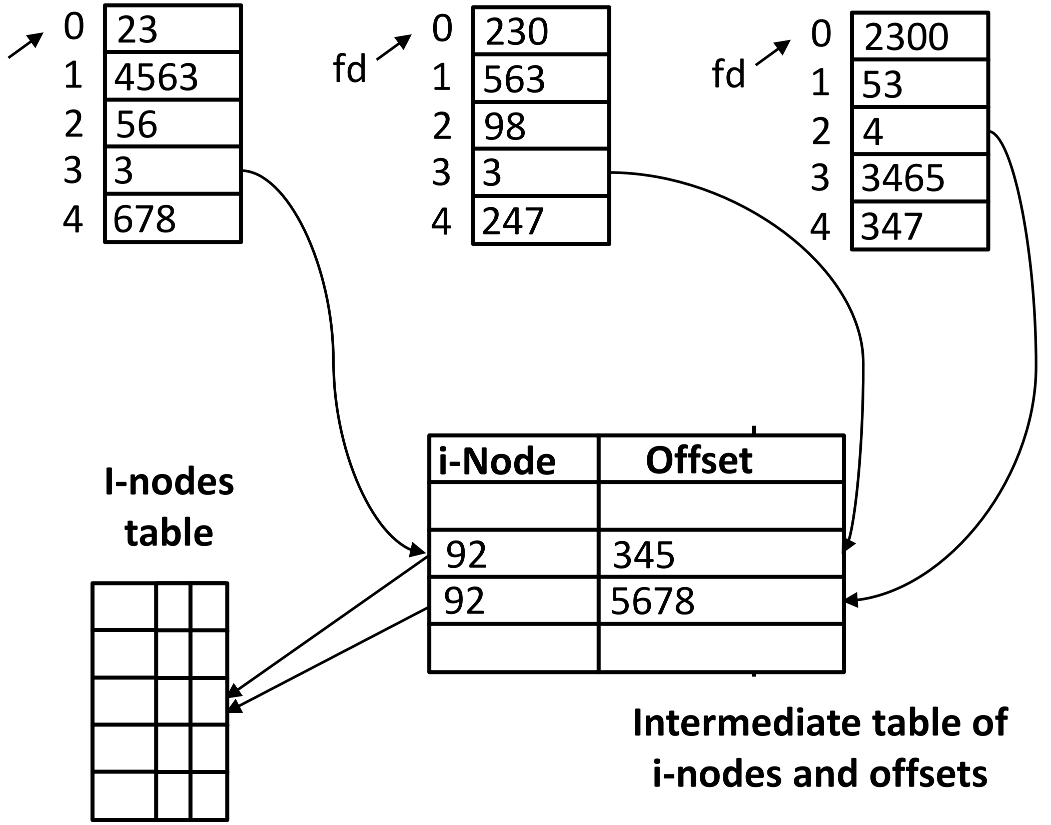


- Allows **sharing** the **read/write position pointer** (R/W-ptr) between related processes.
- The **open files table indicates the "index"** of the row **in this R/W-ptr table**.
- **Each row** in the table has the **R/W-ptr and** the **index in** the **i-nodes table**.

Main management structures

table of file r/w pointers: sharing r/w ptr.





Open files table P1

fd

0	23
1	4563
2	56
3	3
4	678

Open files table P2

fd

0	230
1	563
2	98
3	3
4	247

Open files table P3

fd

0	2300
1	53
2	4
3	3465
4	347

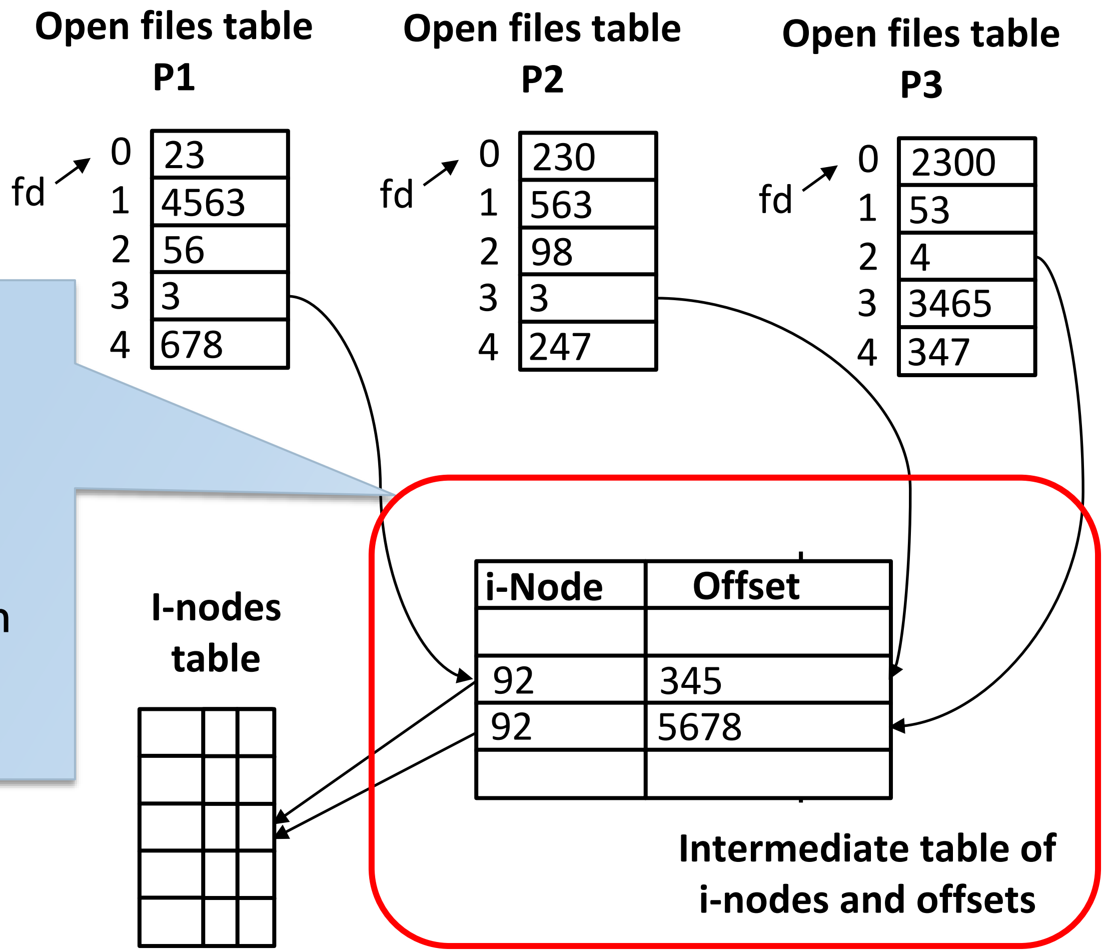
I-nodes table

i-Node	Offset
92	345
92	5678

Intermediate table of i-nodes and offsets

Main management structures
table of file r/w pointers: sharing r/w ptr.

- FILP table (FILe Pointer)
- Between the descriptor table and (usually) the i-node table.
- Saves (mainly) the file position pointer.



Open files table P1

0	23
1	4563
2	56
3	3
4	678

Open files table P2

0	230
1	563
2	98
3	3
4	247

Open files table P3

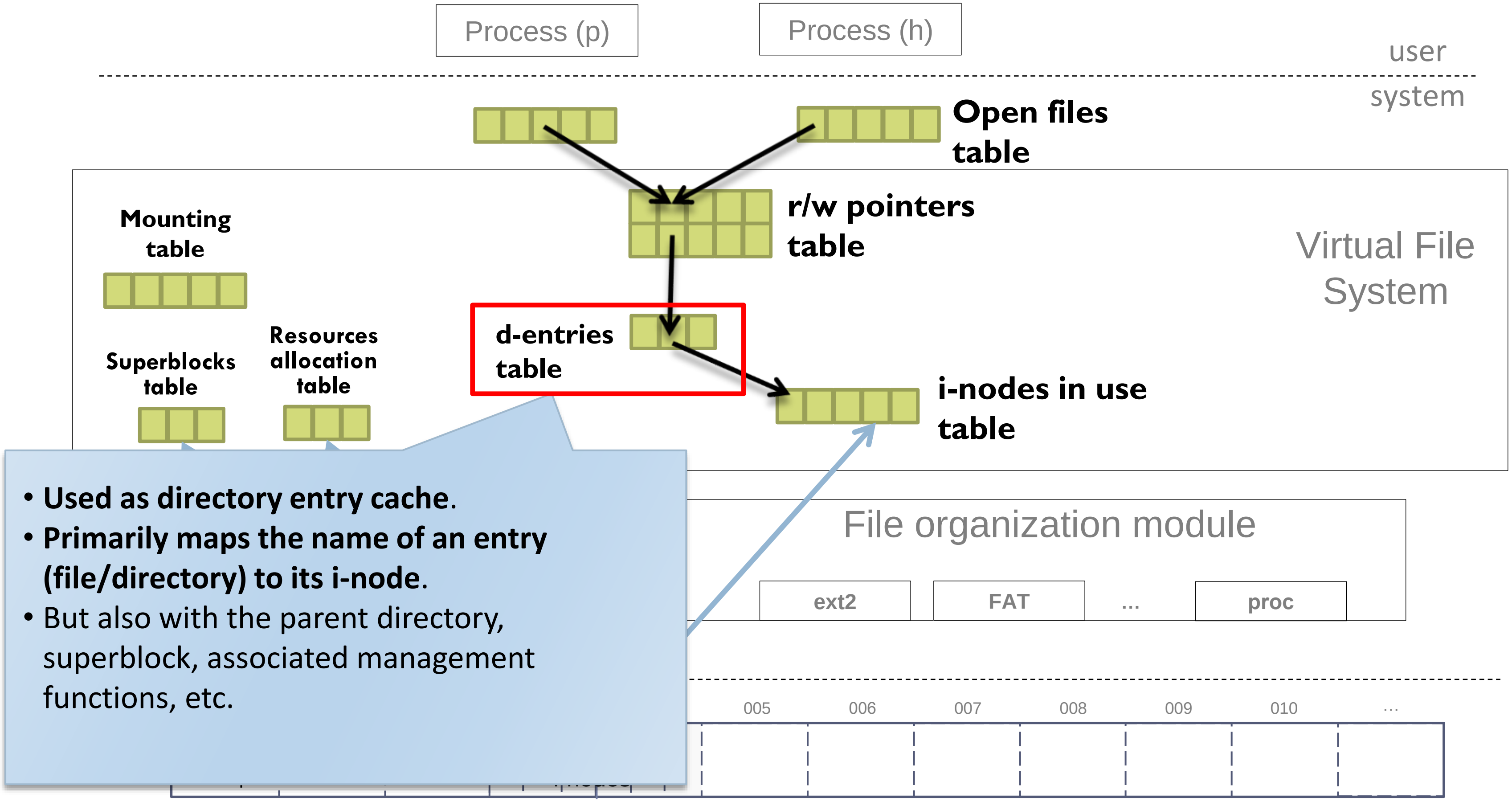
0	2300
1	53
2	4
3	3465
4	347

i-Node	Offset
92	345
92	5678

Intermediate table of i-nodes and offsets

Main management structures

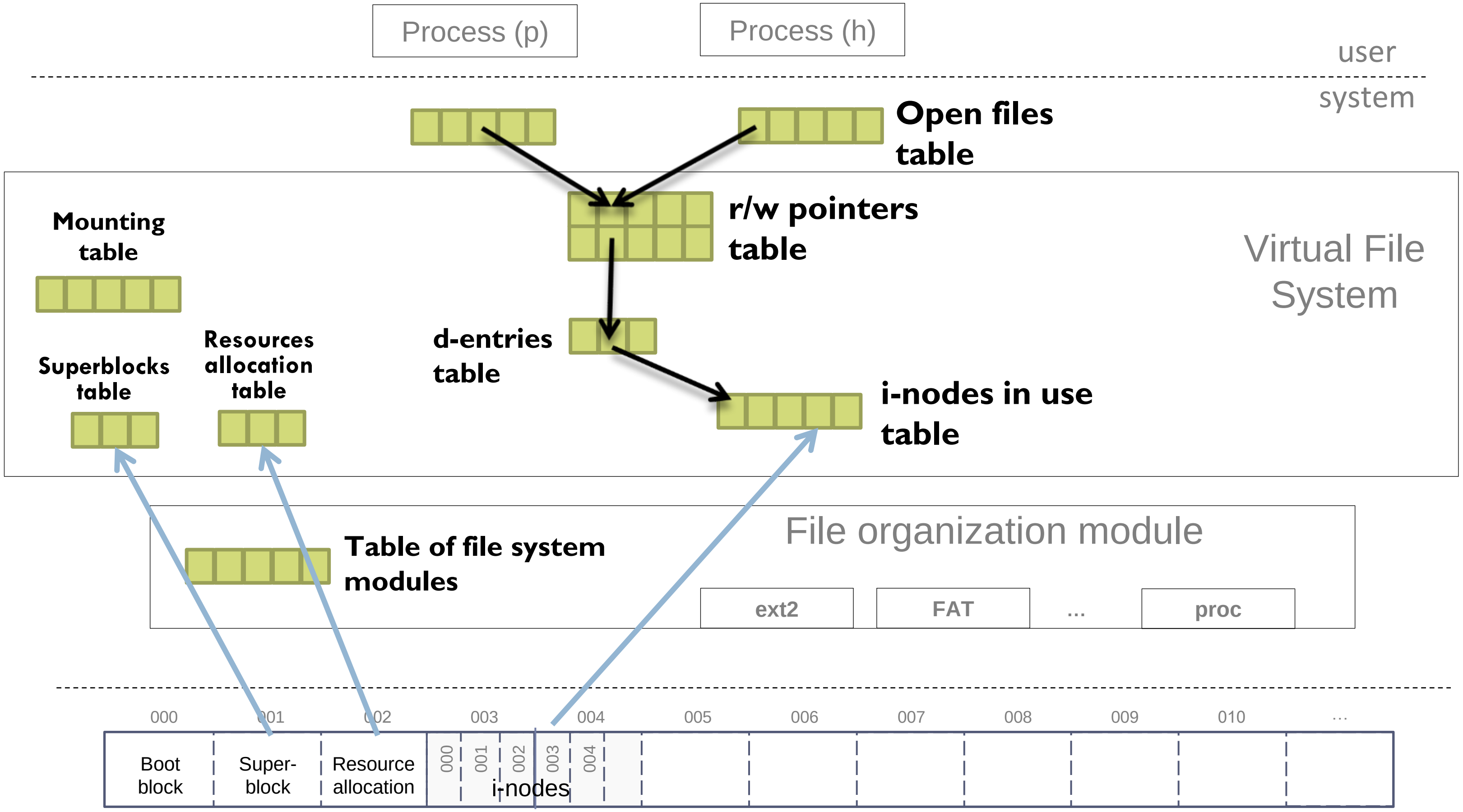
d-entries table: working with directories



- **Used as directory entry cache.**
- **Primarily maps the name of an entry (file/directory) to its i-node.**
- But also with the parent directory, superblock, associated management functions, etc.

Main management structures

summary of the main data structures in memory

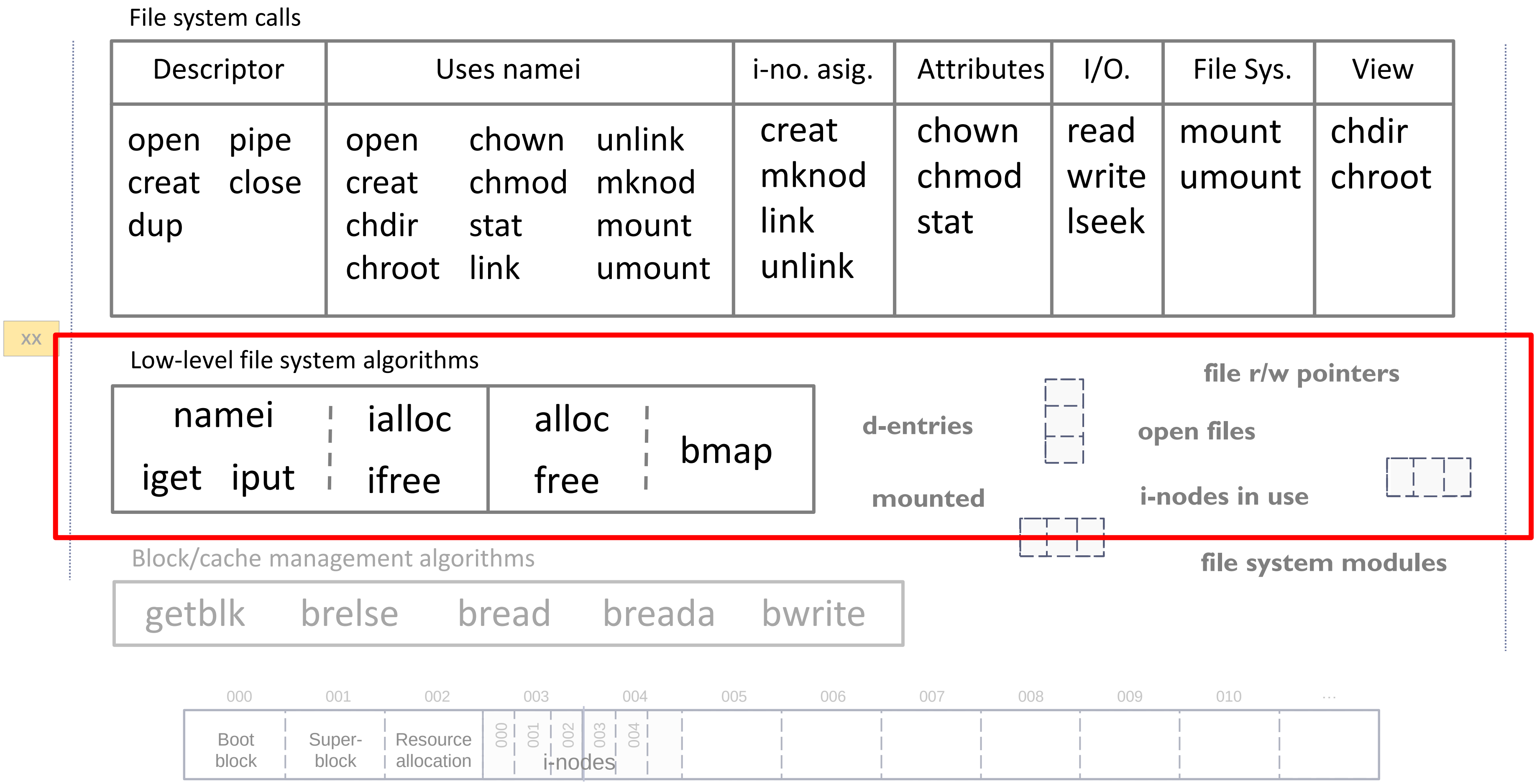


Example of memory organization…

https://github.com/acaldero/nanofs

```
// Information read from the disk
TypeSuperblock sblocks [1] ;
char imap [numInodo] ;
char bmap [numBlocksData] ;
TypeDiskInode inodos [numInodo] ;

// Extra support information
struct {
    int posicion;
    int abierto;
} inodos_x [numInodo] ;

…
```

(3a) Management of disk/memory structures …



File system calls

Descriptor	Uses namei	i-no. asig.	Attributes	I/O.	File Sys.	View
open pipe creat close dup	open chown unlink creat chmod mknod chdir stat mount chroot link umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

Low-level file system algorithms

namei iget iput	ialloc ifree	alloc free	bmap

d-entries

mounted

file r/w pointers

open files

i-nodes in use

file system modules

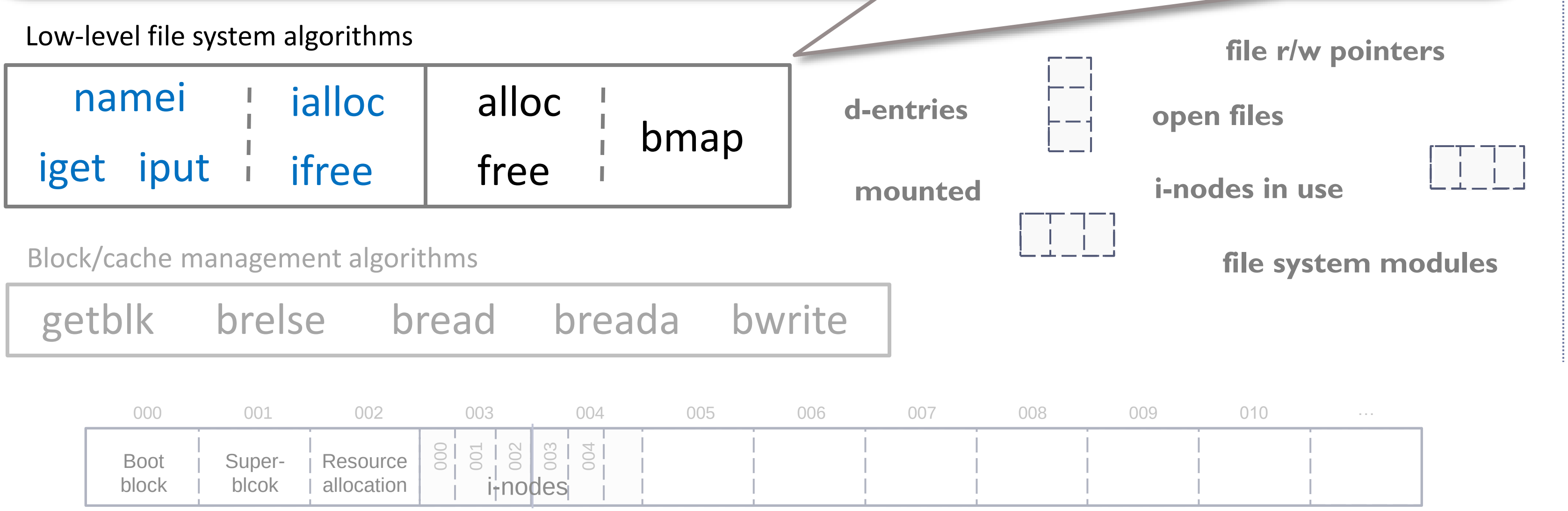
Block/cache management algorithms

getblk brelse bread breada bwrite

Example of management routines

i-nodes

- **namei**: converts a path to the associated i-node.
- **iget**: returns an i-node from the i-node table and if not present, reads it from secondary memory, adds it to the i-node table and returns it.
- **iput**: releases an i-node from the i-node table, and if necessary, updates it in secondary memory.
- **ialloc**: allocates an i-node to a file.
- **ifree**: releases an i-node previously assigned to a file.

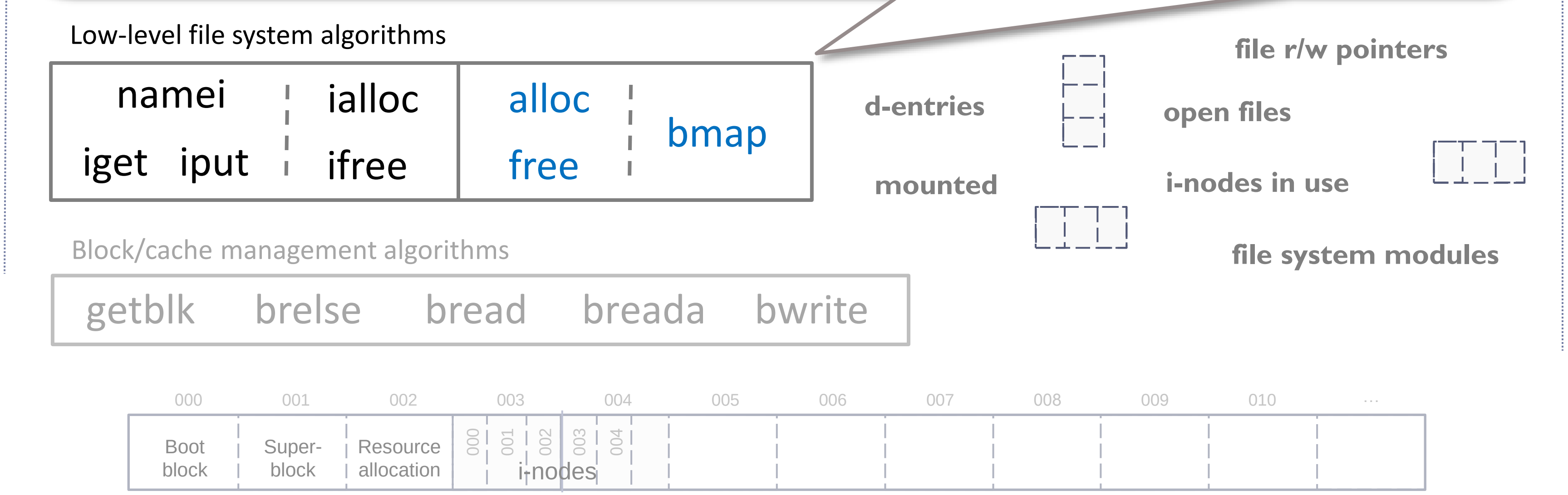


Example of management routines

blocks

http://www.buet.ac.bd/iict/iictcourses/ict6005/lecture9.ppt

- **bmap**: calculates the disk block associated with a file offset.
  Translates logical addresses (file offset) to physical addresses (disk block).
- **alloc**: allocates a block to a file.
- **free**: releases a block previously assigned to a file.



Low-level file system algorithms

namei

iget iput

ialloc

ifree

alloc

free

bmap

d-entries

mounted

file r/w pointers

open files

i-nodes in use

file system modules

Block/cache management algorithms

getblk brelse bread breada bwrite

Example: ialloc and alloc

https://github.com/acaldero/nanofs

```
int ialloc ( void )
{
   // buscar un i-nodo libre
   for (int=0; i<sblocks[0].numInodes; i++)
   {
       if (imap[i] == 0) {
          // inodo ocupado ahora
          imap[i] = 1;
          // valores por defecto en el i-nodo
          memset(&(inodos[i]),0,
                    sizeof(TypeDiskInode));
          // devolver identificador de i-nodo
          return i;
       }
   }

   return -1;
}
```

```
int alloc ( void )
{
   char b[BLOCK_SIZE];

   for (int=0; i<sblocks[0].numBlocksData; i++)
   {
       if (bmap[i] == 0) {
          // block ocupado ahora
          bmap[i] = 1;
          // valores por defecto en el block
          memset(b, 0, BLOCK_SIZE);
          bwrite(DISK, sblocks[0].primerBloqueData + i, b);
          // devolver identificador del block
          return i;
       }
   }
   return -1;
}
```

Example: ifree and free

https://github.com/acaldero/nanofs

```
int ifree ( int inodo_id )
{
    // comprobar validez de inodo_id
    if (inodo_id > sblocks[0].numInodes)
        return -1;

    // liberar i-nodo
    imap[inodo_id] = 0;

    return -1;
}
```

```
int free ( int block_id )
{
    // comprobar validez de block_id
    if (block_id > sblocks[0].numBlocksData)
        return -1;

    // liberar block
    bmap[block_id] = 0;

    return -1;
}
```

Example: namei and bmap

https://github.com/acaldero/nanofs

http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix_unit4.pdf

```
int namei ( char *fname )
{
   // buscar i-nodo con nombre <fname>
   for (int=0; i<sblocks[0].numInodes; i++)
   {
         if (! strcmp(inodos[i].nombre, fname))
               return i;
   }

    return -1;
}
```

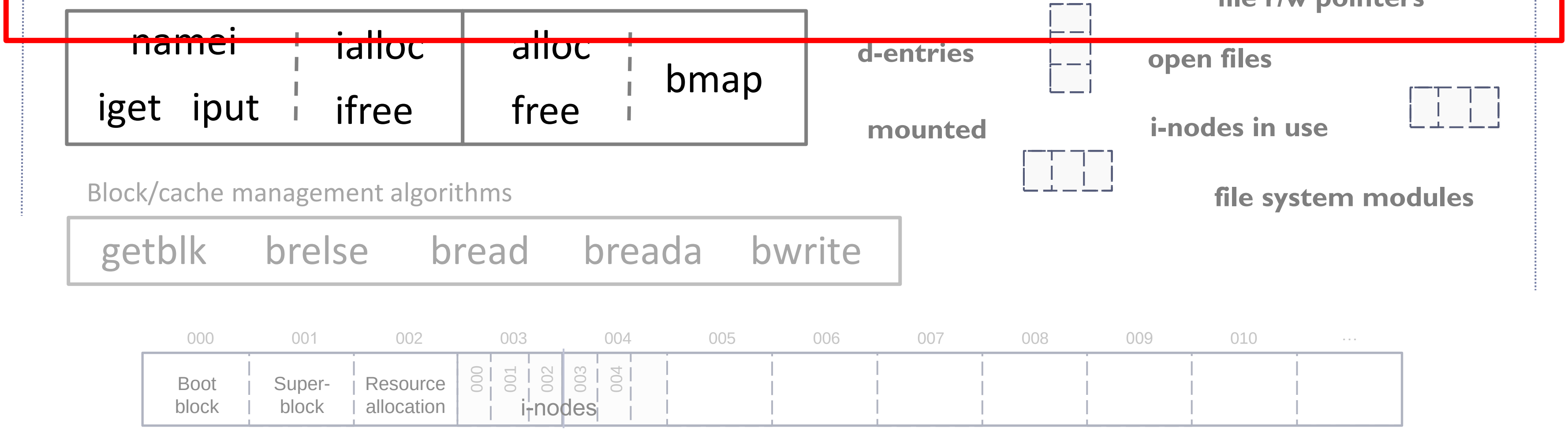
```
int bmap ( int inodo_id, int offset )
{
    int b[BLOCK_SIZE/4];

    // comprobar validez de inodo_id
    if (inodo_id > sblocks[0].numInodes)
        return -1;

    // block de datos asociado
    if (offset < BLOCK_SIZE)
        return inodos[inodo_id].blockDirecto;
    if (offset < BLOCK_SIZE*BLOCK_SIZE/4) {
         bread(DISK, sblocks[0].primerBloqueData +
                                inodos[inodo_id].blockIndirecto, b);
         offset = (offset – BLOCK_SIZE) / BLOCK_SIZE;
         return b[offset] ;
    }

    return -1;
}
```

(3b) System calls…

File system calls

Descriptor	Uses namei	i-no. asig.	Attributes	I/O.	File Sys.	View
open pipe creat close dup	open chown unlink creat chmod mknod chdir stat mount chroot link umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

Low-level file system algorithms



Example

sys. calls

- **open**: locates the i-node associated with the path of the file, …
- **read**: locates the data block, read data block, …
- **write**: locate the data block, write data block, …
- …

http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix_unit4.pdf

File system calls

Descriptor	Uses namei	i-no. asig.	Attributes	I/O.	File Sys.	View
open pipe creat close dup	open chown unlink creat chmod mknod chdir stat mount chroot link umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

Low-level file system algorithms

namei	ialloc	alloc	bmap
iget iput	ifree	free	

d-entries

mounted

file r/w pointers

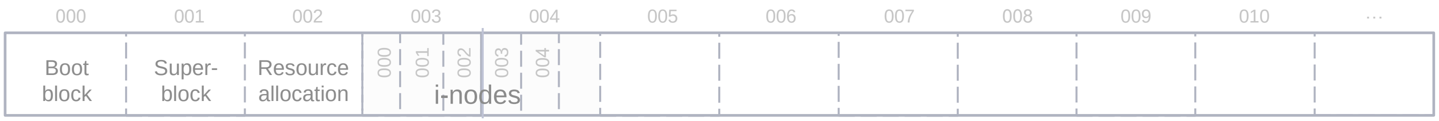
open files

i-nodes in use

file system modules

Block/cache management algorithms

getblk	brelse	bread	breada	bwrite



Example: mount

https://github.com/acaldero/nanofs

```
int mount ( void )
{
   // leer block 0 de disco en sblocks[0]
   bread(DISK, 1, &(sblocks[0]) );

   // leer los blocks para el mapa de i-nodes
   for (int=0; i<sblocks[0].numBlocksInodeMap; i++)
         bread(DISK, 2+i, ((char *)imap + i*BLOCK_SIZE) ;

   // leer los blocks para el mapa de blocks de datos
   for (int=0; i<sblocks[0].numBlocksDataMap; i++)
        bread(DISK, 2+i+sblocks[0].numBlocksInodeMap, ((char *)bmap + i*BLOCK_SIZE);

   // leer los i-nodes a memoria
   for (int=0; i<(sblocks[0].numInodes*sizeof(TypeDiskInode)/BLOCK_SIZE); i++)
        bread(DISK, i+sblocks[0].firstInode, ((char *)inodos + i*BLOCK_SIZE);

   return 1;
}
```

Example: umount
https://github.com/acaldero/nanofs

```
int umount ( void )
{
    // escribir block 0 de sblocks[0] a disco
    bwrite(DISK, 1, &(sblocks[0]) );

    // escribir los blocks para el mapa de i-nodes
    for (int=0; i<sblocks[0].numBlocksInodeMap; i++)
          bwrite(DISK, 2+i, ((char *)imap + i*BLOCK_SIZE) ;

    // escribir los blocks para el mapa de blocks de datos
    for (int=0; i<sblocks[0].numBlocksDataMap; i++)
         bwrite(DISK, 2+i+sblocks[0].numBlocksInodeMap, ((char *)bmap + i*BLOCK_SIZE);

    // escribir los i-nodes a disco
    for (int=0; i<(sblocks[0].numInodes*sizeof(TypeDiskInode)/BLOCK_SIZE); i++)
         bwrite(DISK, i+sblocks[0].firstInode, ((char *)inodos + i*BLOCK_SIZE);

    return 1;
}
```

Example: mkfs

https://github.com/acaldero/nanofs

```
int mkfs ( void )
{
    // inicializar a los valores por defecto del superblock, mapas e i-nodes
    sblocks[0].numMagico = 1234; // ayuda a comprobar que se haya creado por nuestro mkfs
    sblocks[0].numInodes = numInodo;
    …
    for (int=0; i<sblocks[0].numInodes; i++)
           imap[i] = 0; // free
    for (int=0; i<sblocks[0].numBlocksData; i++)
           bmap[i] = 0; // free
    for (int=0; i<sblocks[0].numInodes; i++)
          memset(&(inodos[i]), 0, sizeof(TypeDiskInode) );

    // to write the default file system into disk
    umount();

    return 1;
}
```

Example: open and close

https://github.com/acaldero/nanofs

http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix_unit4.pdf

```
int open ( char *nombre )
{
    int inodo_id ;

    inodo_id = namei(nombre) ;
    if (inodo_id < 0)
        return inodo_id ;


    inodos_x[inodo_id].posicion = 0;
    inodos_x[inodo_id].abierto   = 1;

    return inodo_id;
}
```

```
int close ( int fd )
{




    if (fd < 0)
        return fd ;

    inodos_x[fd].posicion = 0;
    inodos_x[fd].abierto   = 0;

    return 1;
}
```

Example: creat and unlink

https://github.com/acaldero/nanofs

http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix_unit4.pdf

```
int creat ( char *nombre )
{
    int b_id, inodo_id ;

    inodo_id = ialloc() ;
    if (inodo_id < 0) { return inodo_id ; }
    b_id = alloc();
    if (b_id < 0) { ifree(inodo_id); return b_id ; }

    inodos[inodo_id].type = 1 ; // FICHERO
    strcpy(inodos[inodo_id].nombre, nombre);
    inodos[inodo_id].blockDirecto = b_id ;
    inodos_x[inodo_id].posicion = 0;
    inodos_x[inodo_id].abierto   = 1;

    return 1;
}
```

```
int unlink ( char * nombre )
{
     int inodo_id ;

     inodo_id = namei(nombre) ;
     if (inodo_id < 0)
         return inodo_id ;

     free(inodos[inodo_id].blockDirecto);
     memset(&(inodos[inodo_id]),
                    0,
                    sizeof(TypeDiskInode));
     ifree(inodo_id) ;

    return 1;
}
```

Example: read and write

https://github.com/acaldero/nanofs

http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix_unit4.pdf

```
int read ( int fd, char *buffer, int size )
{
   char b[BLOCK_SIZE] ;
   int b_id ;

   if (inodos_x[fd].posicion+size > inodos[fd].size)
       size = inodos[fd].size - inodos_x[fd].posicion;
   if (size =< 0)
       return 0;

   b_id = bmap(fd, inodos_x[fd].posicion);
   bread(DISK,
            sblocks[0].primerBloqueData+b_id, b);
   memmove(buffer,
                  b+inodos_x[fd].posicion, size);
   inodos_x[fd].posicion += size;

   return size;
}
```

```
int write ( int fd, char *buffer, int size )
{
   char b[BLOCK_SIZE] ;
   int b_id ;

   if (inodos_x[fd].posicion+size > BLOCK_SIZE)
       size = BLOCK_SIZE - inodos_x[fd].posicion;
   if (size =< 0)
       return 0;

   b_id = bmap(fd, inodos_x[fd].posicion);
   bread(DISK, sblocks[0].primerBloqueData+b_id, b);
   memmove(b+inodos_x[fd].posicion,
                   buffer, size);
   bwrite(DISK, sblocks[0].primerBloqueData+b_id, b);
   inodos_x[fd].posicion += size;

   return size;
}
```

SISTEMAS OPERATIVOS: SISTEMAS DE FICHEROS

Files, directorios y system de ficheros