Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Part 3: File systems and I/O

Unit 7

Implementing file systems

fSO

Goals and bibliography

- **Goals**
  - To know file system functionality
  - To know the abstraction levels of a file system
  - To understand a file as a secondary storage abstraction and the access ways to its content
  - To analyse the file block allocation techniques
- **Bibliography**
  - Silberschatz, chapters 10 and 11

- **File system architecture**
- File concept
- File block allocation

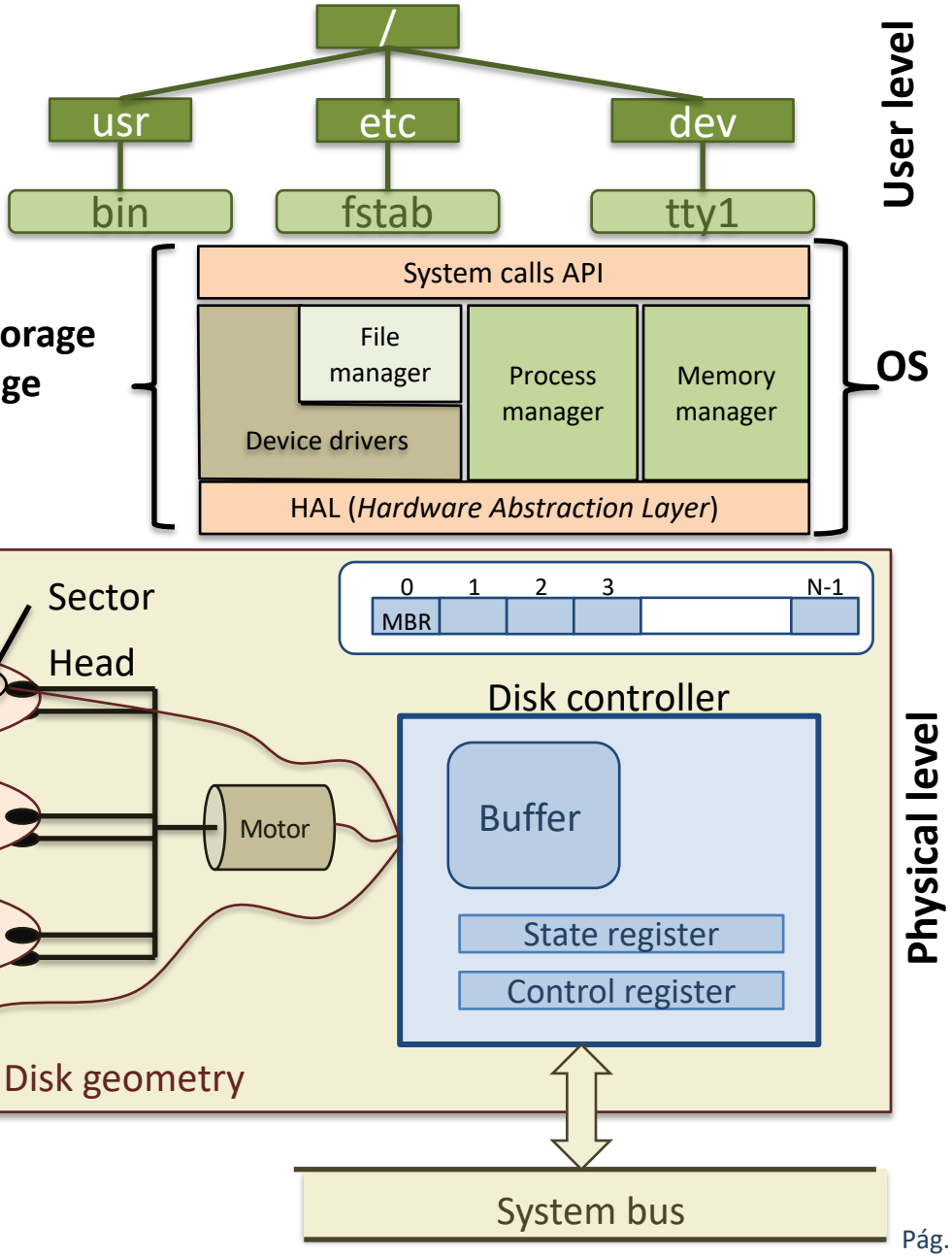
File system architecture

- File system architecture has three aspects:
  1. **How files are organized** from the user point of view
  2. **How files are stored** in secondary storage (commonly disks)
  3. **How file operations** like reading, writing, positioning, etc, are done.

File system architecture

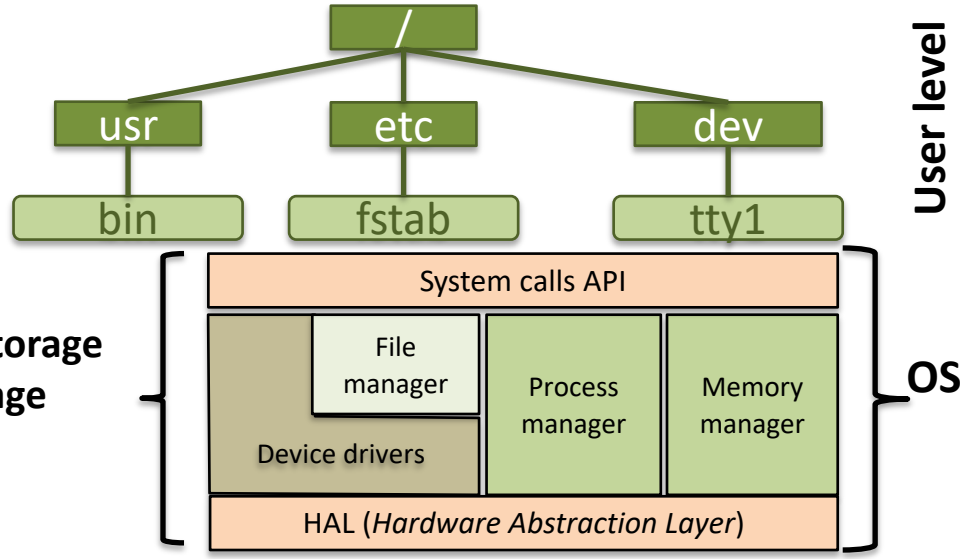
File
OS provided abstraction

The OS **hides the physical properties of the storage system** and provides **a uniform vision of storage devices**



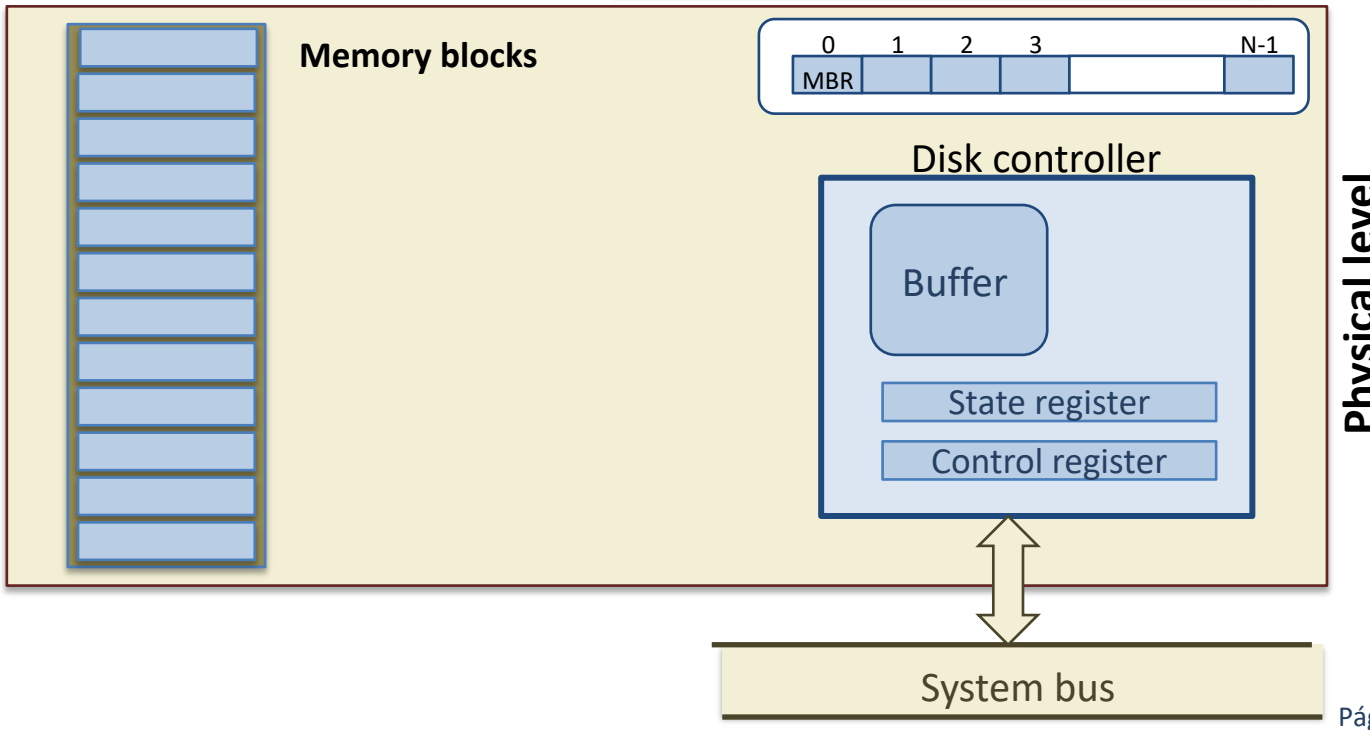
File system architecture

File
OS provided abstraction



The OS **hides the physical properties of the storage system** and provides **a uniform vision of storage devices**

SSD disk



File system architecture

- The **file system** provides mechanisms to:
  - **Persistent storage** of information
    - Information stays in the computer when it is switched off, the most common device nowadays is the hard disk
  - **Access to information**
    - A **user interface** made of:
      - **Files**: logical unit for persistent storage of data
      - **Directories:** mechanism (container) to organize files
- Importance
  - It keeps system critical data
  - It constrains global system performance
  - It is the most visible and used aspect of an OS

Persistent storage is any storage device or system that retains data after power is turned off and indepently of the creator.

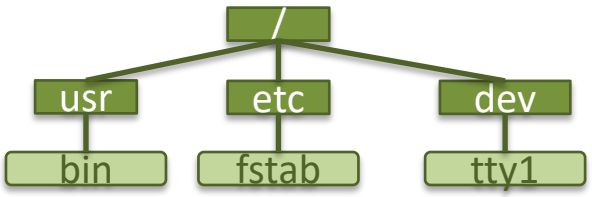
File system architecture

System calls (user library)
It does file and directory management from the programmer sight

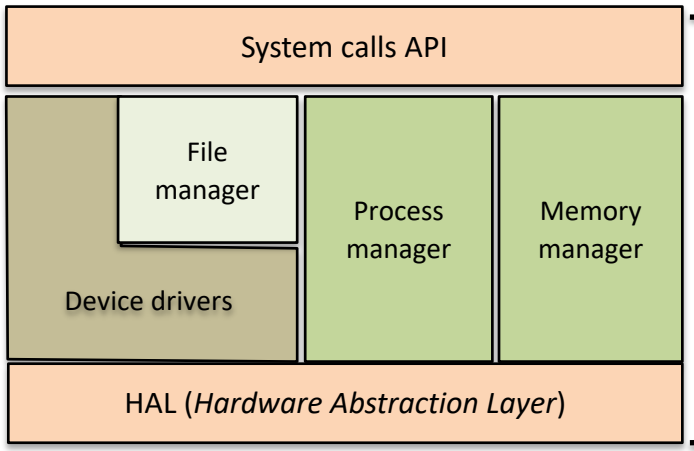
File manager
- It uses a device driver to do information transfer between disk and memory
- It implements mechanisms to provide **coherency**, **security** and **protection**
- **It optimizes** performance
- It creates basic user interface elements: **files** and **directories**

Device driver
- It dialogs with the device controller
- It starts physical operations and processes the end of I/O

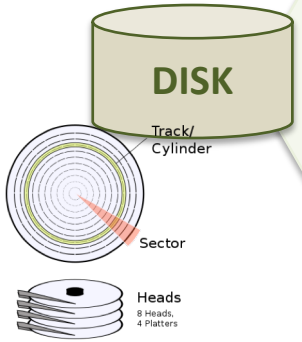
Physical level
Device + controller
- Block device
- Disk geometry
  Disk = Block vector
  Coordinates: CHS

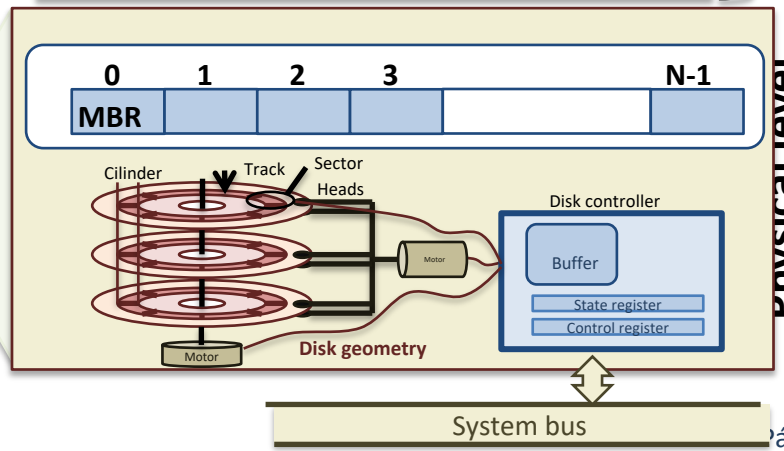


User level



OS





Physical level

- **User View**

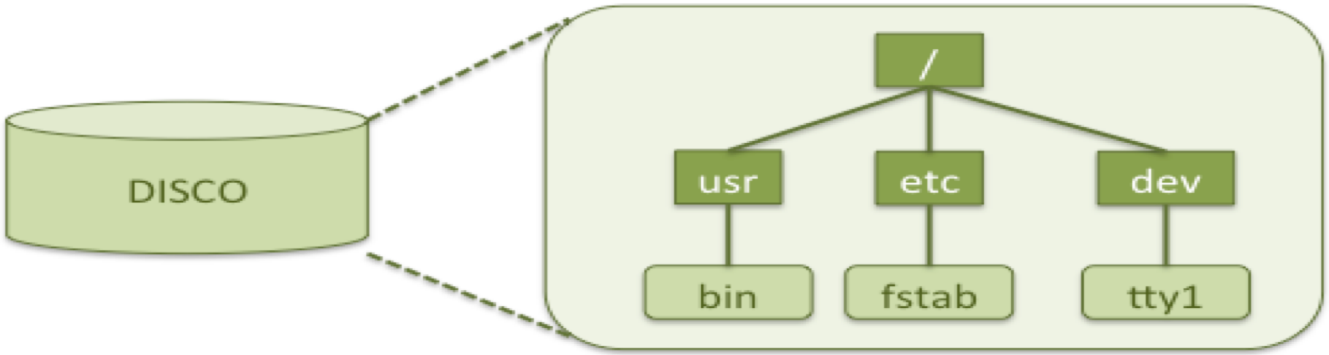
User level
File and directory abstractions

User libraries (to operate with files)
API with system calls related to files and directories

File operations:
- **Open/close**
- **Read/write**
- **Seek** within a file

Directory operations:
- **Create/remove**
- **Rename**
- Searching
- Navigating through the file system

Hierarchical view
Files and directories have a tree organization



- **Open file call**
  - System call that allows accessing file content
    - Processes must open a file before reading or writing on it
    - The **calling process receives a file handler** that will reference the file in future reading or writing operations
  - It establishes the file access mode (read and/or write) and the initial position on the file of the access pointer
    - For instance, when accessing a file in writing or reading mode
      - Locating the access pointer at the beginning (0) to overwrite the whole file content
      - Locating the access pointer at the end (filesize – 1) to add new content to the file
    - Upcomming accesses will start at the access pointer location left in the last access and will update it
  - File access permissions are checked
    - Open call will fail if the mode specified is inconsistent with the file permissions

File system architecture

The UNIX ''less is better'' philosophy imposed a few simple rules on files:

- One type of file is a list of other files; this type of file is called a directory.
- All input and output is done using files. Disks, tapes, displays, devices and input/output are all manipulated using the same function calls.
- A file is an ordered sequence of bytes. Semantic is provided by the program that reads or writes the data.
- One type of file is a list of other files; this type of file is called a **directory**.

File system architecture

- **File operations**
  - The OS provides a set of system calls to work with files
    - **creat**
      - It requires free space on disk and to create a file
    - **open**
      - Required operation before reading or writing
    - **read**
      - It requires a file identifier and a reading location pointer
    - **write**
      - It requires a file identifier, data to write and a writing location pointer
    - **lseek**
      - It sets file reading and writing pointers
    - **close**
      - It frees OS internal structures that support file access
    - **remove**
      - It frees disk space allocated to a file and removes the corresponding directory entry

The primitive file operations all operate on file descriptors. A file descriptor is a small, non-negative integer used to identify an open file.

- **Open:** Required operation before reading or writing

int **open**(const char *path, int oflag, ...);

O_RDONLY Open for reading only.
O_WRONLY Open for writing only.
O_RDWR Open for reading and writing.

O_APPEND Set the file offset to the end-of-file prior to each write.
O_CREAT If the file does not exist, allow it to be created. T
O_EXCL This flag may be used only if O_CREAT. It causes the call open to fail if the file already exists.
O_TRUNC This flag should be used only on ordinary files opened for writing. It causes the file to be truncated to zero length

```
int fd;
…
fd = open("/home/user/so_user",O_WRONLY | O_CREAT |
O_TRUNC);
```

- **read:** reads *nbyte* bytes from the file open on *fildes* into buffer *buff*
- ssize_t read(int fildes, void ***buf**, size_t nbyte);

returns the number of bytes read. The value will be smaller than *nbyte* if the file has fewer bytes immediately available for reading. If there is an error, a value of -1 is returned.

```
int fd;
int nb;
char *buff;
…
fd = open("/home/user/so_user", O_WRONLY | O_CREAT |
O_TRUNC);
nb = read(fd, buff, 32);
```

- File system architecture
- **File concept**
- File block allocation

File concept

- **A file is:**
  - An abstract data type
  - An interrelated information collection established by its author
  - The required element to write information in secondary storage

```c
#include <stdio.h>

main() {
  int x; /*variable entera*/
  int y;/*variable entera*/
  int *px; /* puntero a entero*/
  x=5;
  px=&x;/*px=direccion de x*/
  y=*px;/* y contiene el dato apuntado por px*/

  printf("x=%d\n",x);
  printf("y=%d\n",y);
  printf("*px=%d\n",*px);
 printf("px = %p\n", px);

}
```

Content of a C course code file

File concept

- **File** = Attributes + Data

METADATA
Attributes
required to manage the file system

DATA
File content, like for instance text, binary code, etc

- **File attributes**
  - The change from system to system
    - Type
    - Size
    - Protection info
    - Owner
    - Creation date and time
- **File data**
  - The OS sees the file content as a byte vector, it is up to the application to give meaning to them
  - A file can store different information types: program source, text data, binary code, graphics, sound, etc
  - The file data can have a certain structure set by the file type (i.e. wav files, jpeg files, etc)
  - An executable file is a file made up by a sequence of code sections that the OS is able to load and execute

File list showing file attributes

```
gandreu@shell-labs:~/sisop/FSO/ejemplosC$ ls -lhi
total 136K
11928522 -rw-r--r-- 1 gandreu disca-upvnet  470 sep 20  2013 aritmetica_punteros.c
11930469 -rw-r--r-- 1 gandreu disca-upvnet  453 sep 26  2011 aritmetica_punteros.c~
11930470 -rwxr-xr-x 1 gandreu disca-upvnet 8,8K sep 26  2011 cir
11928236 -rw-r--r-- 1 gandreu disca-upvnet  193 sep 22  2011 circulo.c
11930472 -rwxr-xr-x 1 gandreu disca-upvnet 8,9K sep 26  2011 cua
11928246 -rwxr-xr-x 1 gandreu disca-upvnet 8,3K sep 16 16:41 cuad
11928433 -rwxr-xr-x 1 gandreu disca-upvnet 8,9K sep 20  2013 cuadrado
11928435 -rw-r--r-- 1 gandreu disca-upvnet  214 sep 22  2011 cuadrado.c
11928418 -rw-r--r-- 1 gandreu disca-upvnet  193 sep 22  2011 cuadrado.c~
11928437 -rwxr-xr-x 1 gandreu disca-upvnet 8,9K sep 22  2011 cuadro
11933192 -rw-r--r-- 1 gandreu disca-upvnet   80 sep 10  2013 error
11930471 -rw-r--r-- 1 gandreu disca-upvnet  579 sep 26  2011 hipotenusa.c
11930468 -rw-r--r-- 1 gandreu disca-upvnet  453 sep 26  2011 hipotenusa.c~
11927803 -rwxr-xr-x 1 gandreu disca-upvnet  424 jun 27  2014 hola.c
11928409 -rwxr-xr-x 1 gandreu disca-upvnet  241 jun 20  2014 hola.c~
11930473 -rwxr-xr-x 1 gandreu disca-upvnet 8,8K sep 26  2011 punt
11928436 -rwxr-xr-x 1 gandreu disca-upvnet 8,8K sep 22  2011 puntero
11928438 -rw-r--r-- 1 gandreu disca-upvnet  315 sep 22  2011 punteros.c
11928434 -rw-r--r-- 1 gandreu disca-upvnet  214 sep 22  2011 punteros.c~
11928243 -rw-r--r-- 1 gandreu disca-upvnet  525 sep 22  2011 variables.c
```

```c
#include <stdio.h>

main() {
  int x; /*variable entera*/
  int y;/*variable entera*/
  int *px; /* puntero a entero*/
  x=5;
  px=&x;/*px=direccion de x*/
  y=*px;/* y contiene el dato apuntado por px*/

  printf("x=%d\n",x);
  printf("y=%d\n",y);
  printf("*px=%d\n",*px);
 printf("px = %p\n", px);

}
```

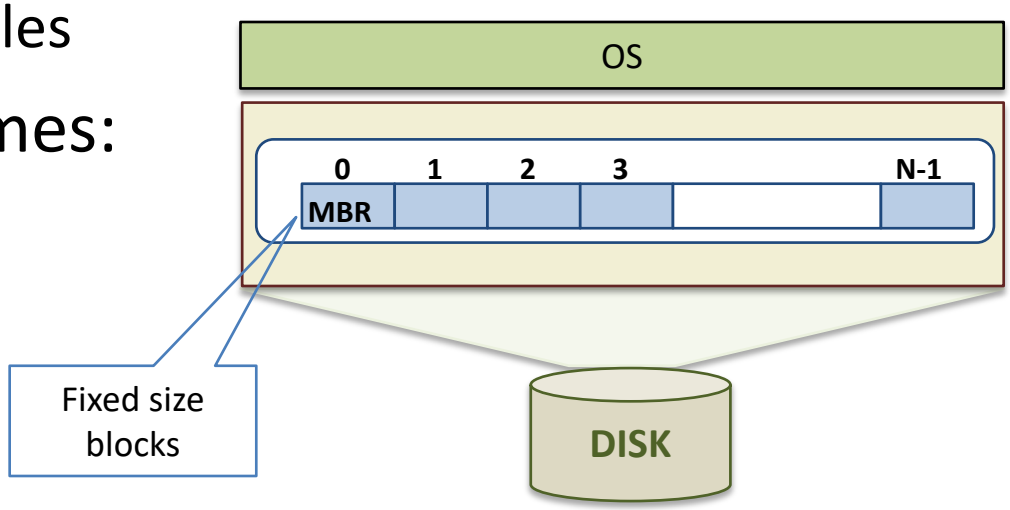
File content

- **Access methods to file data**
  - There are three access modes to file information:
    - **Sequential**
      - Information is accessed (reading or writing) in order
      - In every read/write operation the location pointer is implicitly updated
    - **Direct**
      - The file is made up of logical registers
      - In every operation an argument indicates the working register
    - **Memory mapping**
      - The file is allocated in a logical memory range of one or several processes
      - In this way file read/write ops are transformed into main memory read/write ops
      - The OS is in charge of updating information into disk

- File system architecture
- File concept
- **File block allocation**

File block allocation

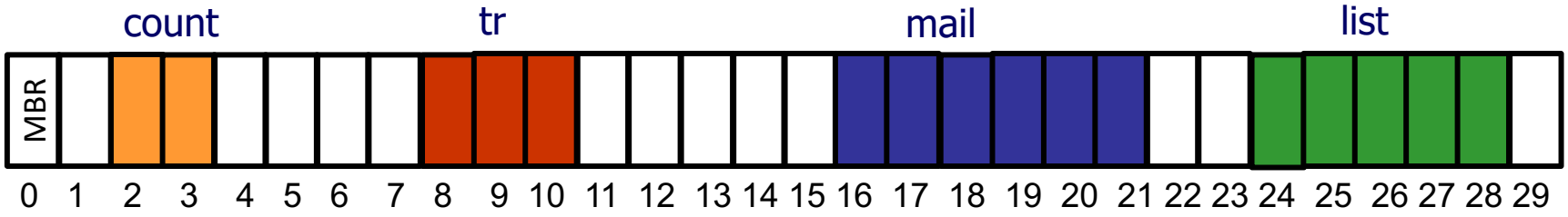
- ¿How to allocate disk space to files?
  - Modern OSs view hard disks as a numbered set of fixed size byte blocks
    - Common sizes are between 512 Bytes y 4 KB (i.e. 1KB)
  - It requires:
    - Efficient use of disk space
    - Fast access to files
  - Allocation schemes:
    - Contiguous
    - Linked
    - Indexed



File block allocation

- **Contiguous allocation**
  - A file is allocated as a set of consecutive disk blocks
  - It is defined for every file as the first allocated block address and the file length in blocks

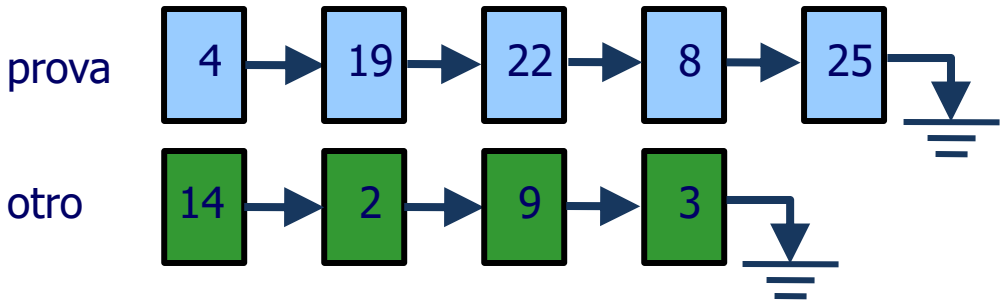
Directory		
File	**Start**	**Length**
count	2	2
tr	8	3
mail	16	6
list	24	5

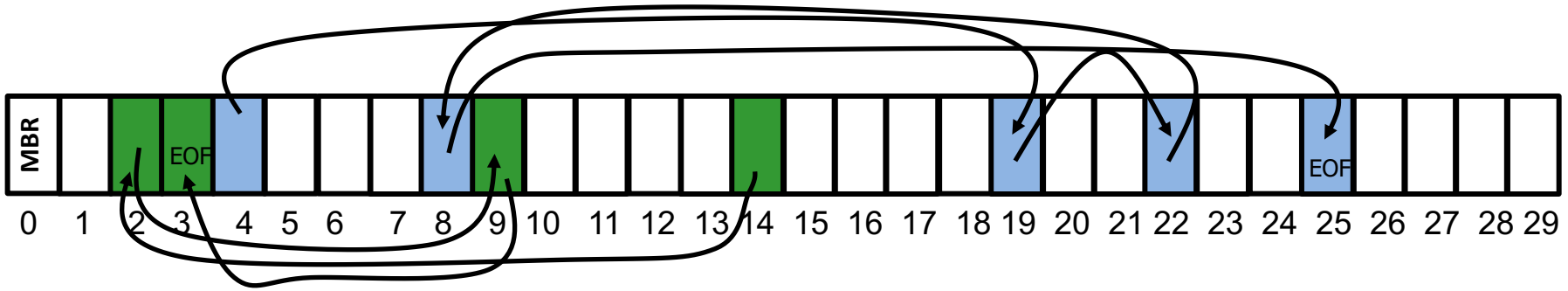


File block allocation

- **Linked allocation**
  - File allocated blocks do not need to be contiguous, then every block is linked to the next by means of a pointer

Directory	
File	Start
prova	4
otro	14





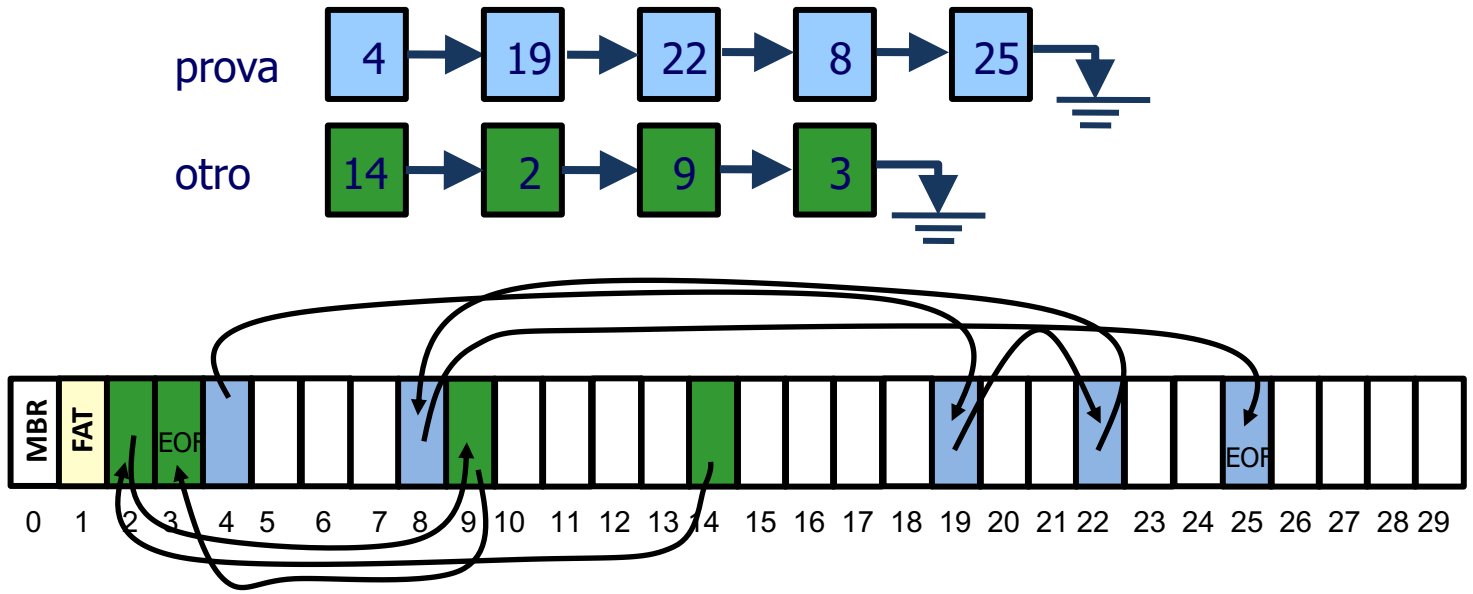
File block allocation

- **FAT** – variation of linked allocation

0	reservado
1	reservado
2	9
3	Eof
4	19
5	-
6	-
7	4
8	25
9	3
10	-
11	-
12	-
13	-
14	2
15	-
16	-
17	-
18	-
19	22
20	-
21	-
22	8
23	-
24	-
25	Eof
26	-
27	-
28	-
29	-

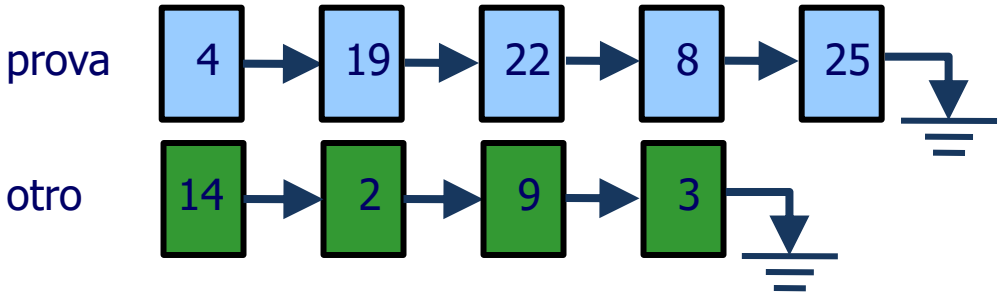
- Pointers are not inside the disk blocks but in a disk dedicated area (File Allocation Table)
- EOF marks the list end

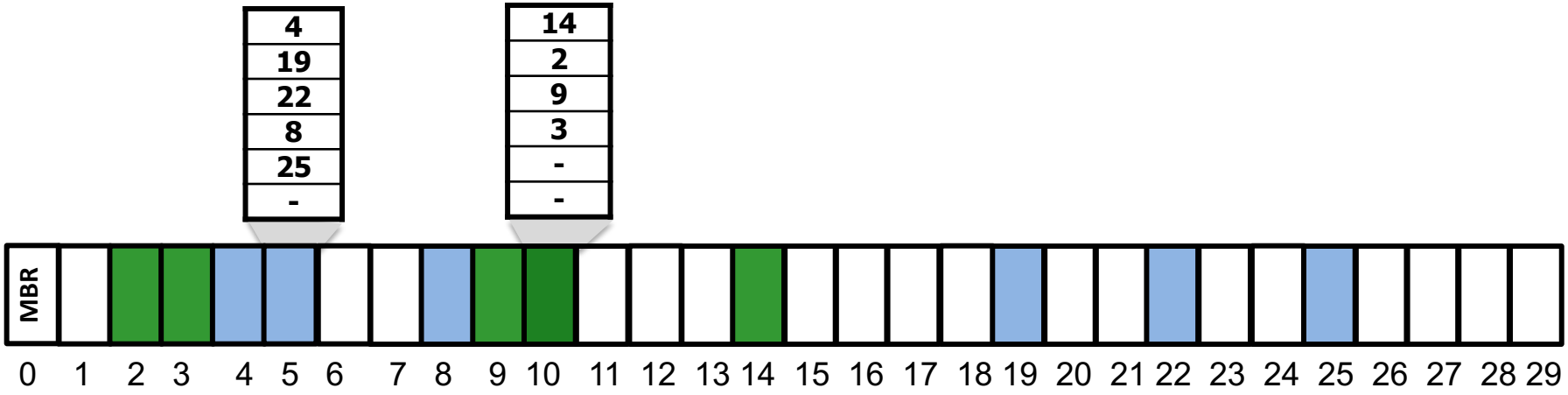
Directory	
File	Start
prova	4
otro	14



- **Indexed allocation**
  - A block could be index block or data block, a index block contains pointers to data blocks

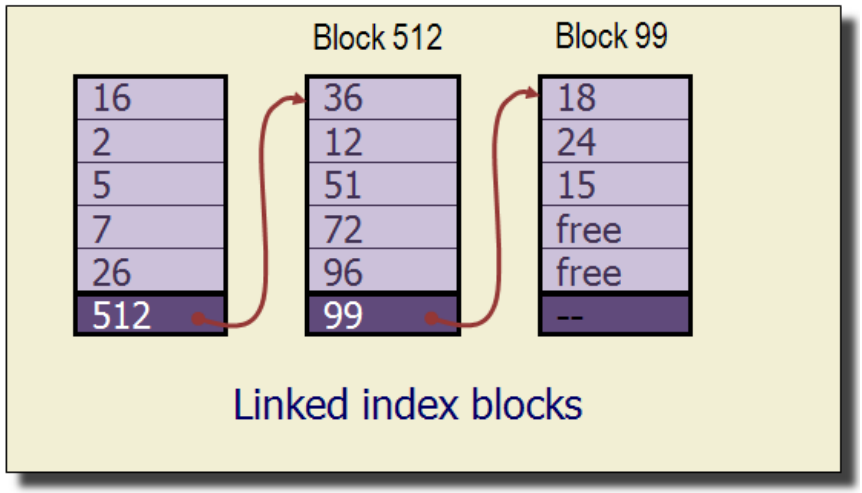
Directory	
File	Start
prova	5
otro	10

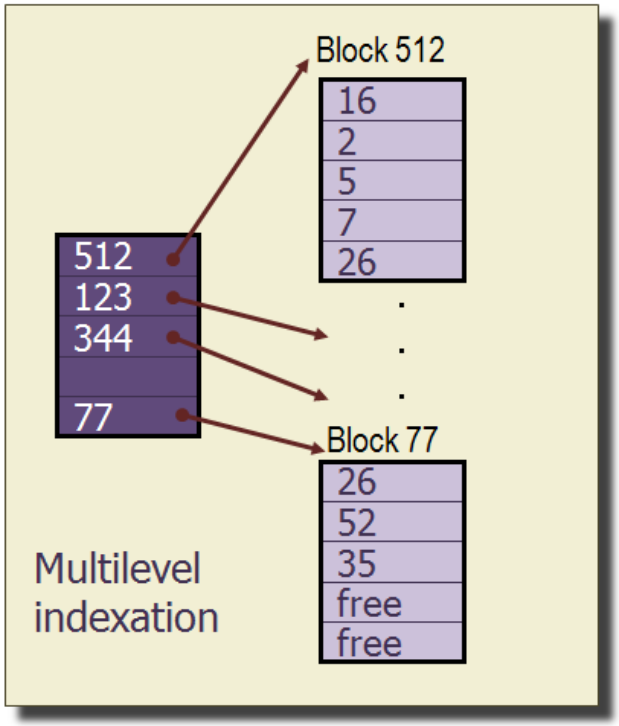




File block allocation

- **Multilevel indexed allocation**
  - It is a variation of indexed allocation
  - Motivation
    - Supporting big files requires several index blocks
  - Solution
    - A pointer can point to a data block or to another index block





- Allocation types analysis

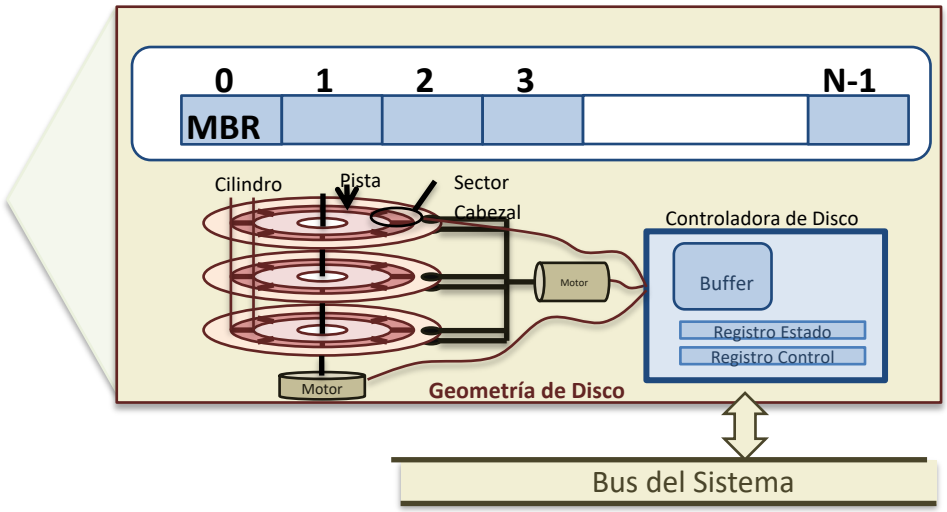
	Advantages	Disadvantages
Contiguous	It is the more efficient It supports sequential and direct access Stable access speed Perfect for read only devices (CD, DVD, etc)	Complex space management (i.e. finding the best gap, relocation due to file growth, etc) It suffers from external fragmentation (compatation required from time to time)
Linked	It doesn't contrain file growing	It doesn't support direct access It is litte robust against failures
FAT	If FAT is copied in memory then it supports direct acces It makes easy free space management	It FAT doesn't fit in main memory then it lacks from any advantage -> only useful for low capacity devices It is little robust against failures
Indexed	It supports sequential and direct access	It constrains file growing (index block size)
Multilevel Indexed	It doesn't contrain file growing	To locate a block several disk accesses may be required

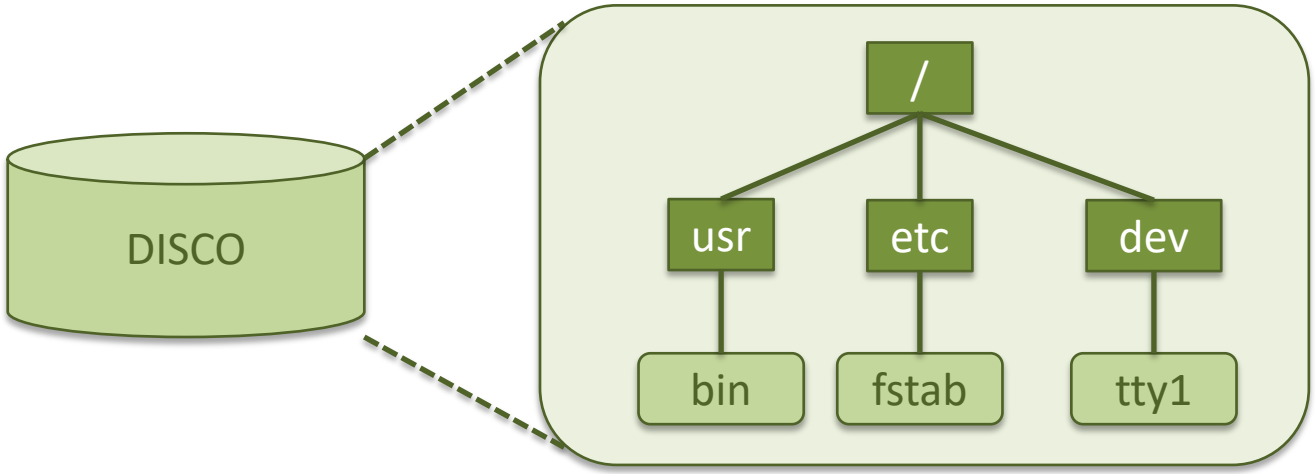
NOTE. In every case there is **internal fragmentation**, half of last block is wasted in average

Summary

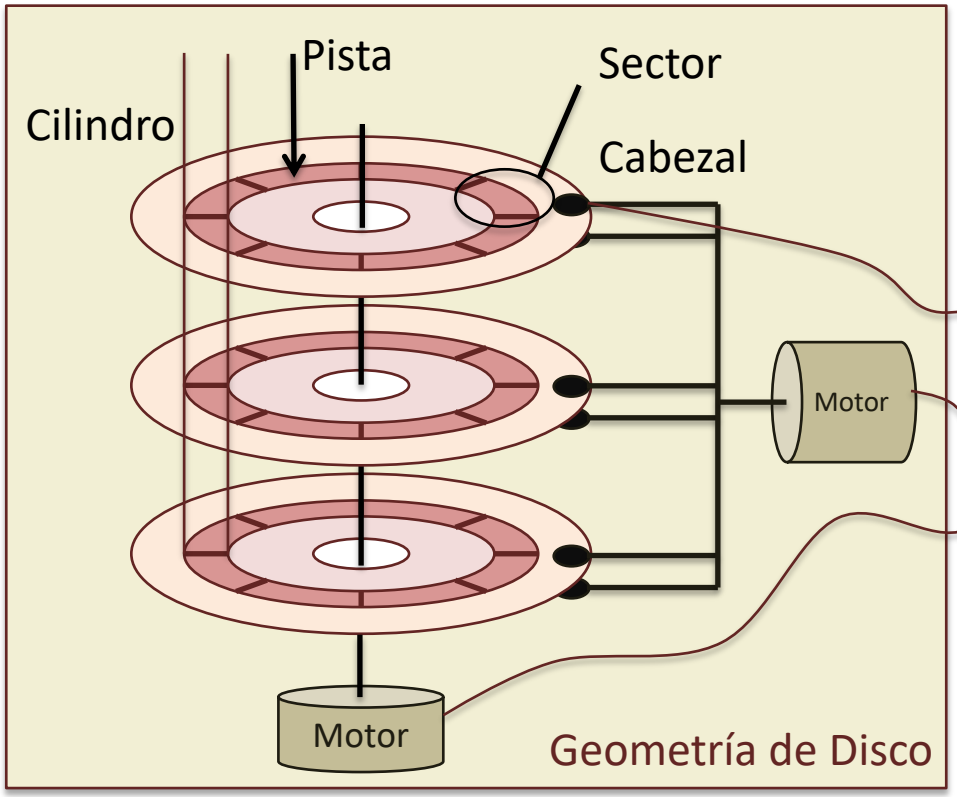
- File system view have been presented
- We have analysed:
  - How files are organized from the user point of view
  - How files are organised in secondary storage (commonly disks)
  - How file are used from the user point of view

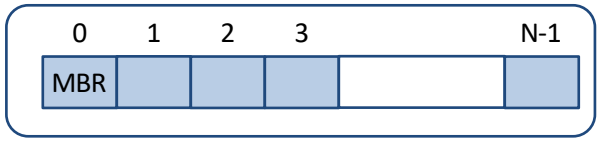
Anexo- figuras

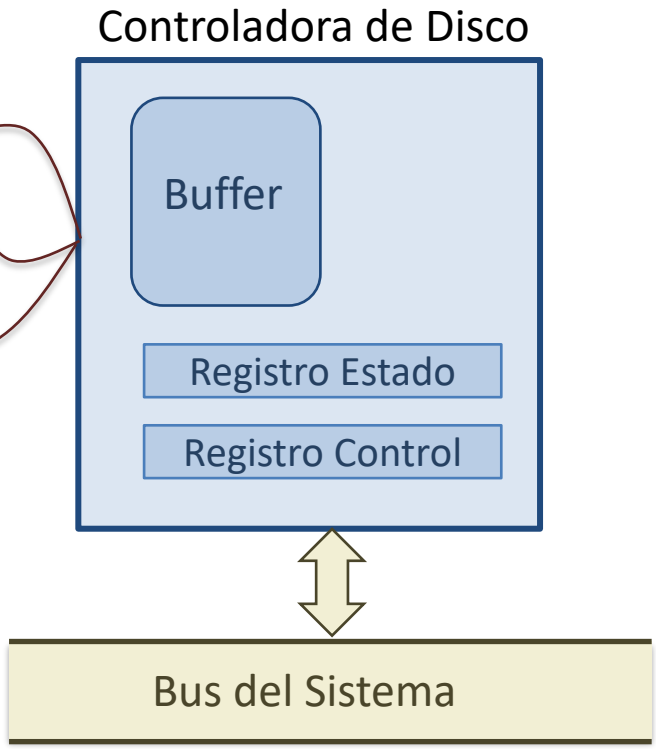




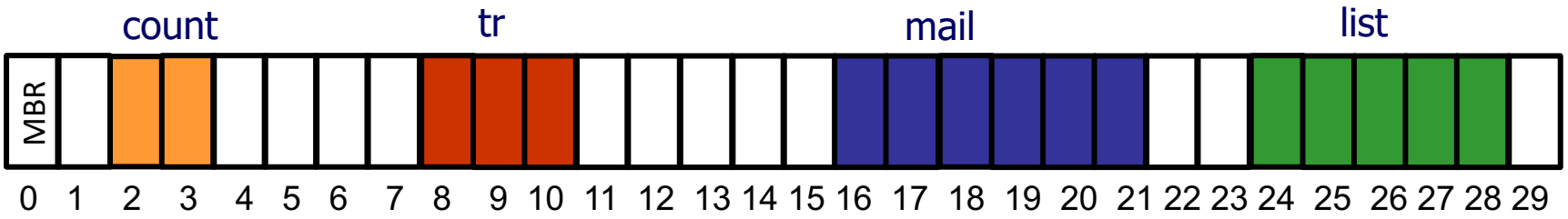
- Disco

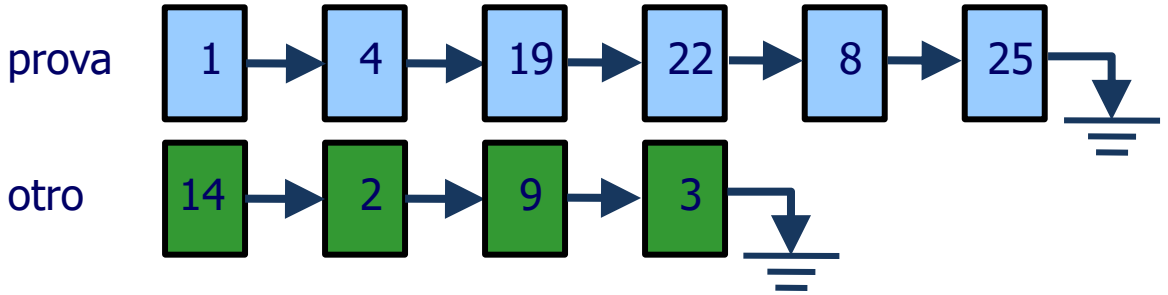

Pista
Sector
Cilindro
Cabezal
Motor
Motor
Geometría de Disco

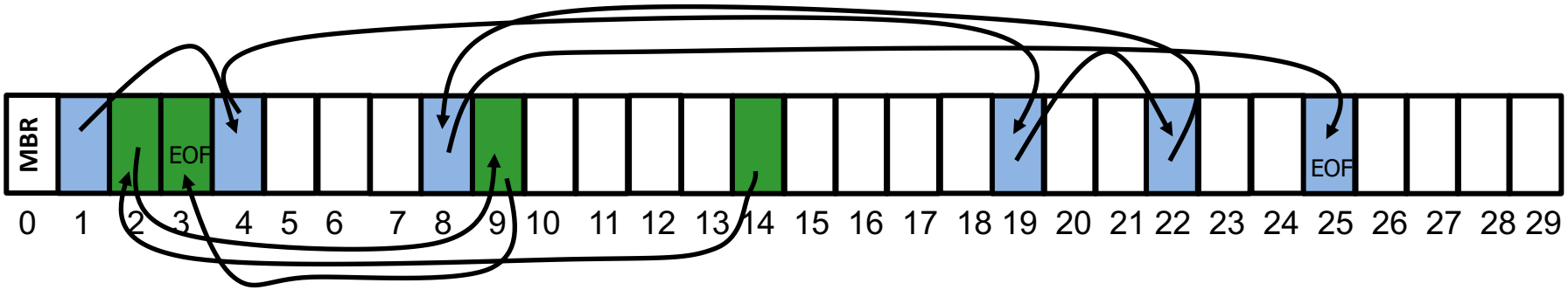

0
1
2
3
N-1
MBR


Controladora de Disco
Buffer
Registro Estado
Registro Control
Bus del Sistema

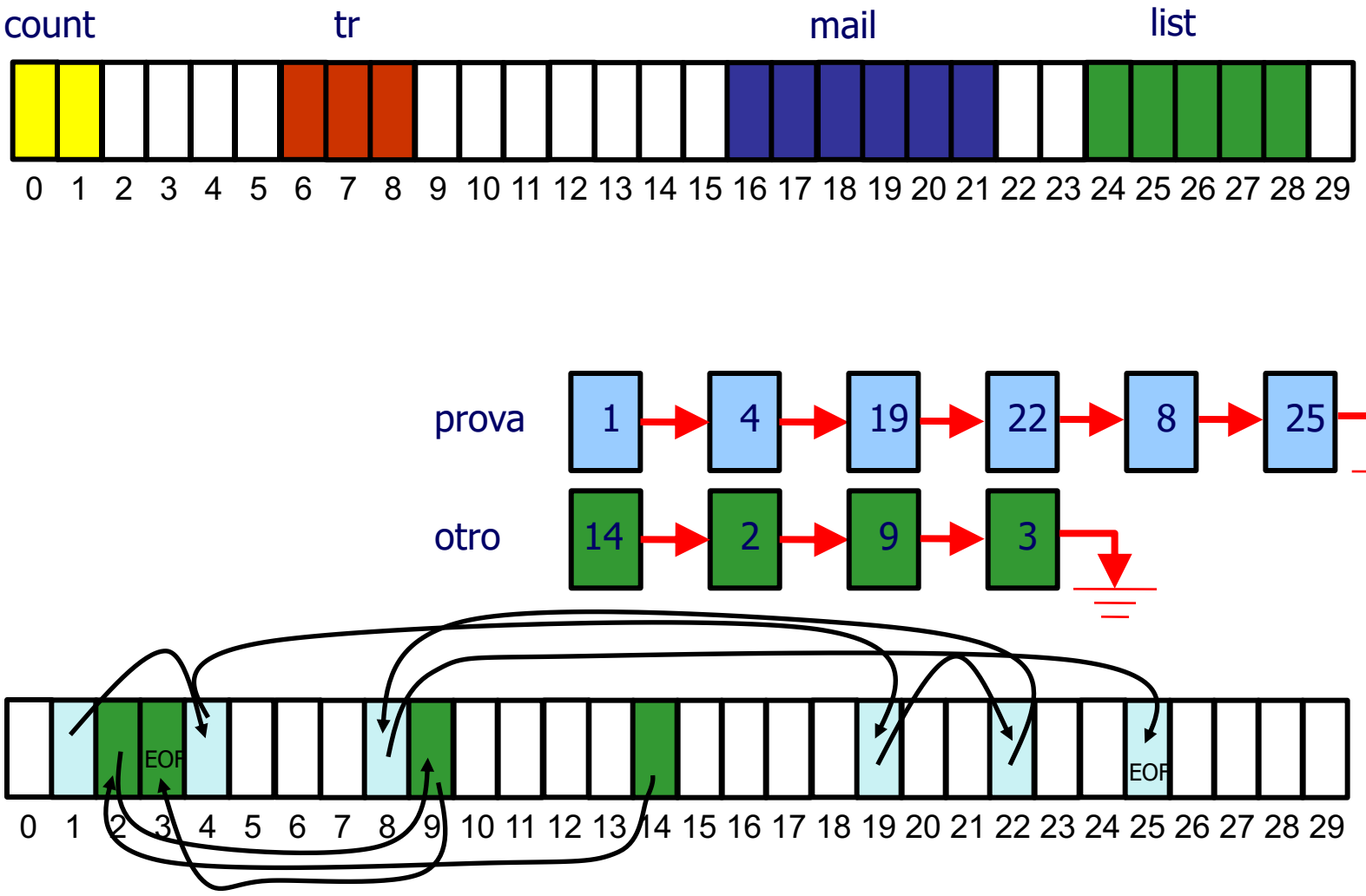
Figuras Asignación

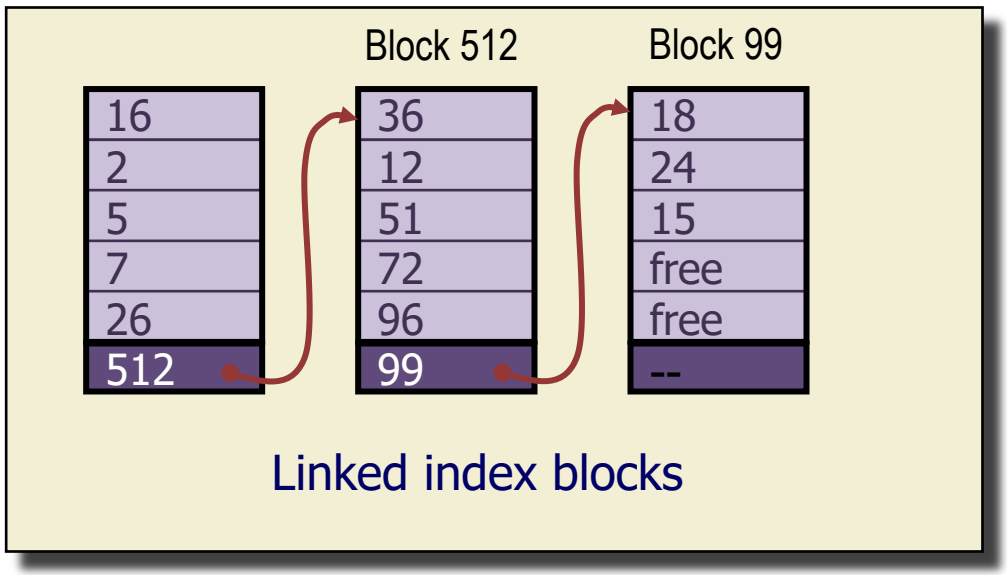




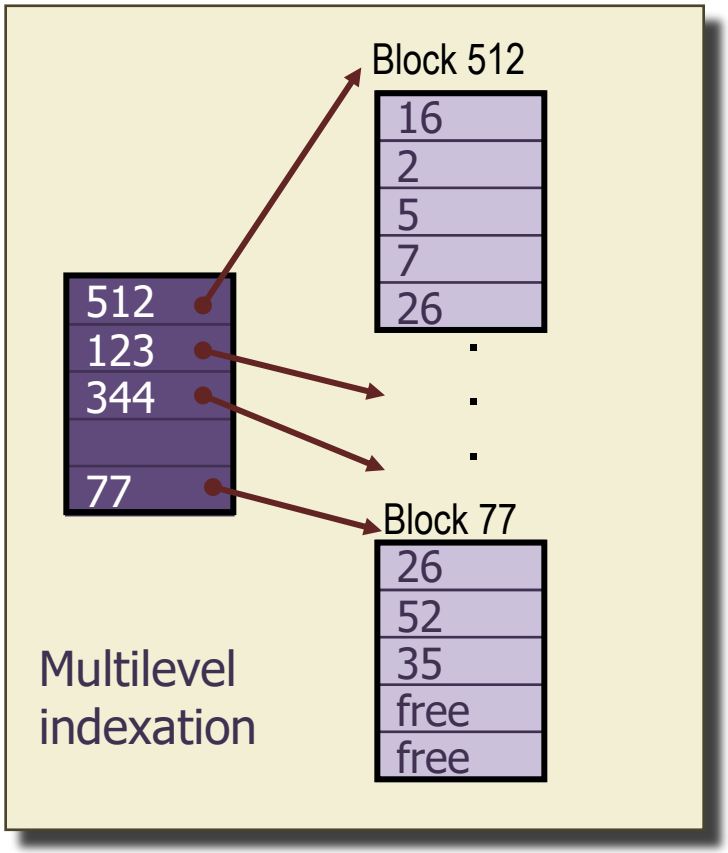


Figuras Asignación





Linked index blocks



Multilevel indexation