OPERATING SYSTEMS: FILE SYSTEMS

Files, directories and file system

To remember…

Before classes

Class

After class

Prepare the prerequisites.

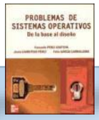
Study the material associated with the **bibliography**: slides alone are not enough.
Please ask questions (especially after study).

Exercising skills:

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

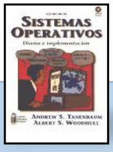
Recommended reading

Base



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

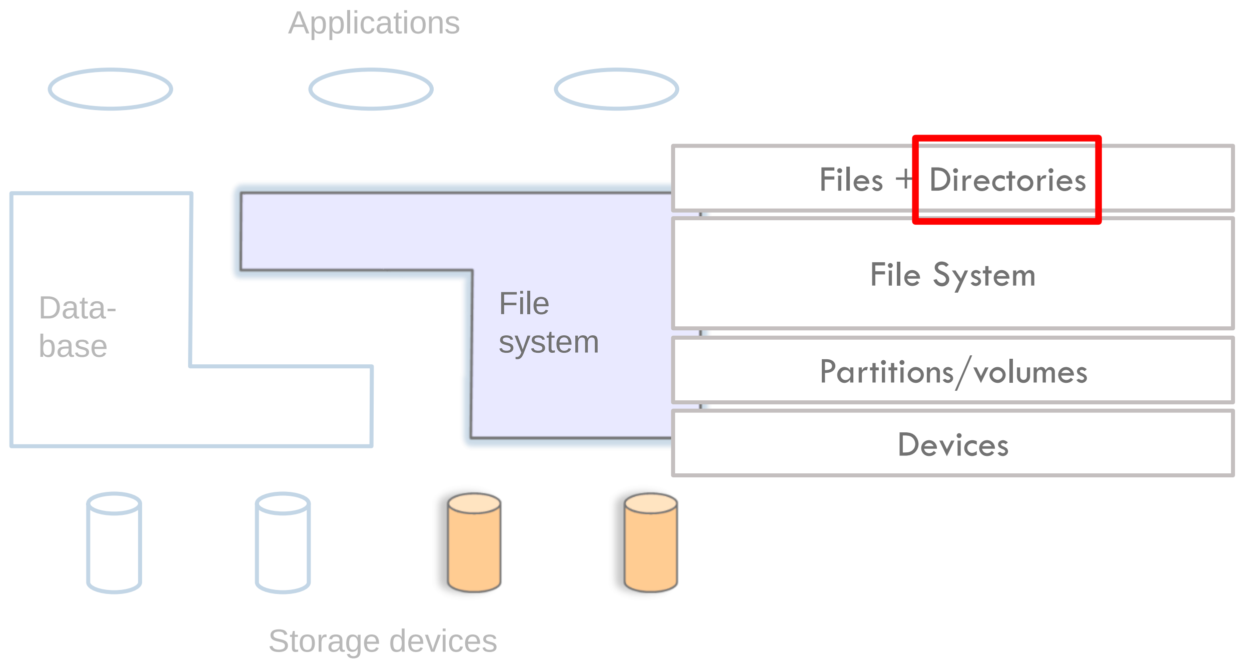
Suggested



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

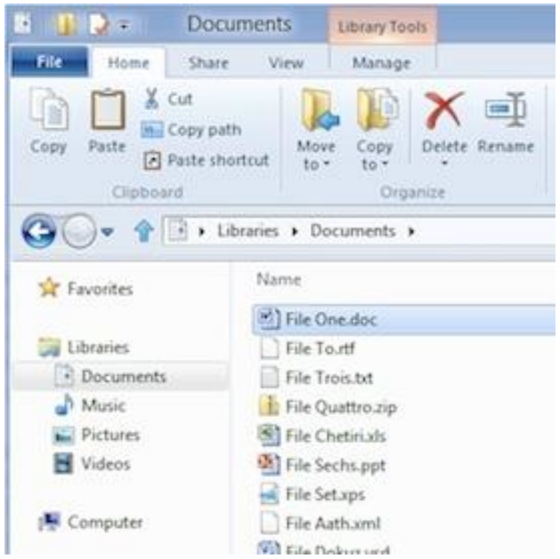
Directory (folder)

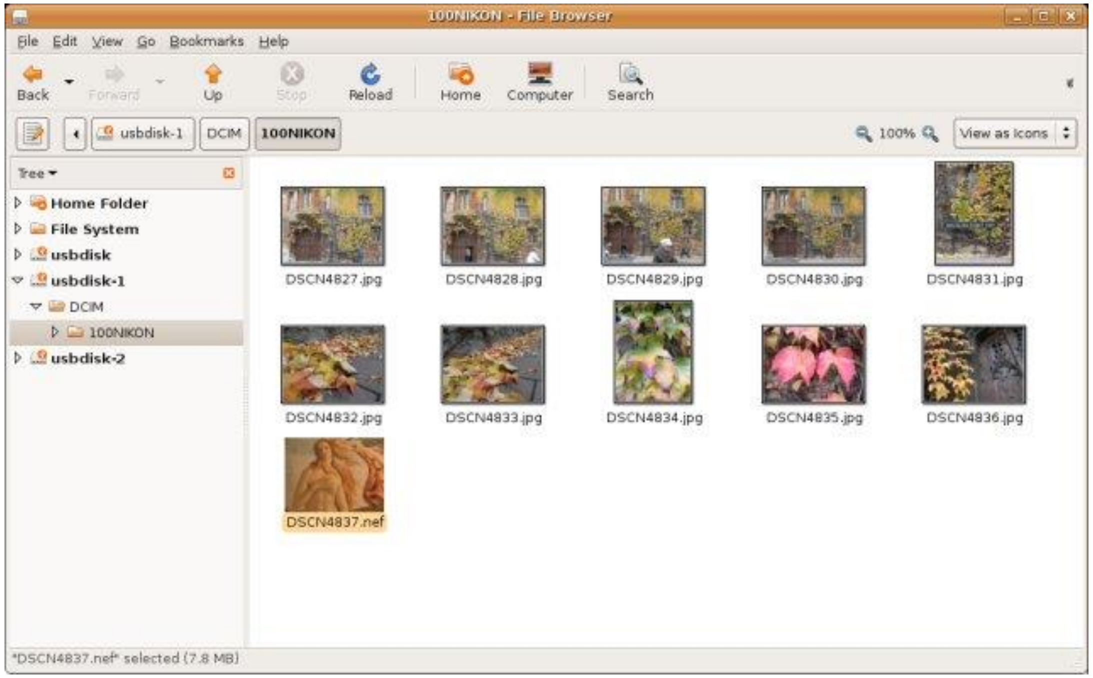


Directory (folder)

important!

- Data structure that allows grouping a set of files according to the user's criteria





important!

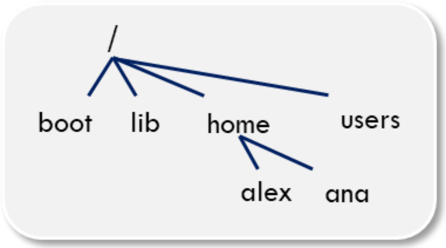
Directory (folders): goals

- Data structure that allows grouping a set of files according to the user's criteria.
  - Search efficiency: locate a file quickly.
  - Grouping: logical grouping of files according to their properties
    - For example: C11 programs, games, etc.
  - Naming: convenient and easy for users:
    - Variable length names.
    - Two users can use the same name for different files.
    - The same files can have different names.
  - Structured: clearly defined operations and hiding
    - C.R.U.D.: mkdir <d>, ls <d>, mv <d> <c>, rmdir <c>
    - cd <d>, cd .., rm <f>, rm –fr <d>
  - Simplicity: the directory entry should be as simple as possible.

important!

Directories: hierarchical names

- Hierarchical names for identification:
  - List of names up to the directory/file.
  - Names are divided by a special character:
    - / in LINUX and \ in Windows

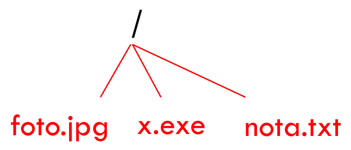


- Special directory names:
  - .   Current directory or working directory (E.g.: cp /home/alex/mail.txt .)
  - . . Parent directory or previous directory (E.g.: ls ..)
  - ~   User home directory in UNIX (E.g.: ls –las ~ ; ls –las $HOME)
  - /   Root directory in UNIX (E.g.: ls –las /)
- Two types of naming used:
  - Absolute or **full name** (starts with the root directory)
    - `/usr/include/stdio.h` (linux)
    - `c:\usr\include\stdio.h` (windows)
  - **Relative name** (it is relative to the current directory, it does not start with root)
    - `stdio.h` assuming that `/usr/include` is the current directory.
    - `../include/stdio.h`

Directories: organization

important!

- Organize and provide information on the structuring of file systems:



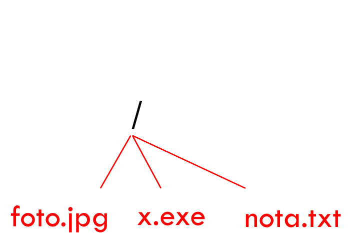
  - Single-level
    - **1** dir. with **n** files
    - **1** file with **1** dir.

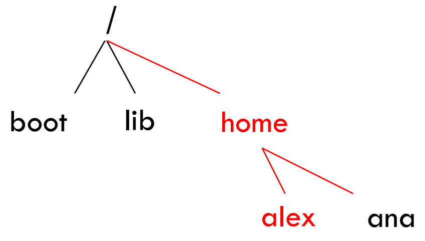
- Single-level:
  - Single directory for all users.
  - [D] high probability of file name matching
- Two-levels:
  - First level with one directory per user.
  - [A] same filename for different users but [D] grouping problems.

Directories: organization

important!

- Organize and provide information on the structuring of file systems:





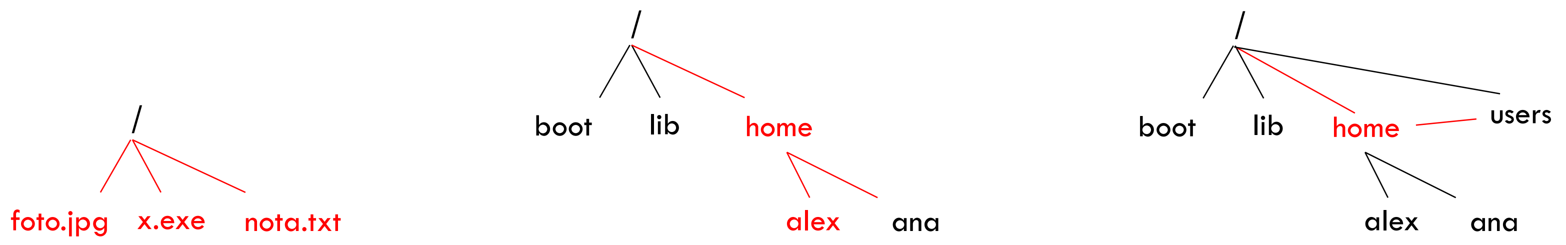
- Single-level
  - **1** dir. with **n** files
  - **1** file with **1** dir.
- Hierarchical (tree)
  - **1** dir. with **n** entries
  - **1** entry with **1** dir.

- Hierarchical or tree:
  - [A] Hierarchy and clustering.
  - [A] Efficient search.
  - Absolute names and relative names (working directory)

Directories: organization

important!

- Organize and provide information on the structuring of file systems:



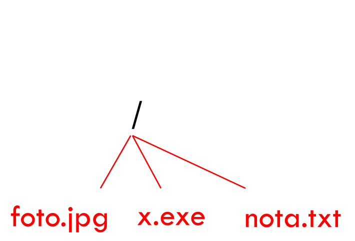
- Single-level
  - 1 dir. with **n** files
  - 1 file with 1 dir.
- Hierarchical (tree)
  - 1 dir. with **n** entries
  - 1 entry with 1 dir.
- A-cyclic tree
  - 1 dir. with **n** entries
  - 1 entry with **n** dir.

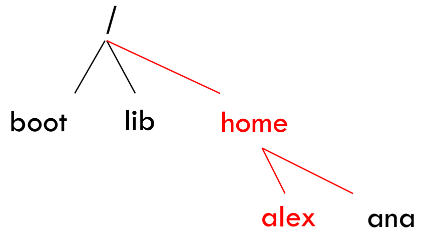
- Acyclic network:
  - Adds to the hierarchical one the possibility of two directories sharing files and/or subdirectories.
  - Use of the link concept.
    - Linux: a) soft/symbolic link and b) hard/physical link.
    - Windows: a) in UI with shortcuts and symlinks and b) junctions (using NTFS reparse points)

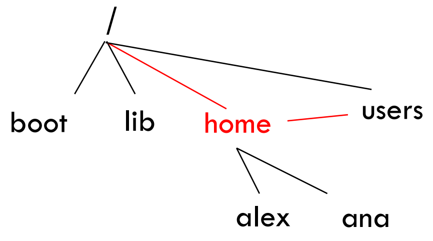
Directories: organization
summary

important!

- Organize and provide information on the structuring of file systems:







- Single-level
  - **1** dir. with **n** files
  - **1** file with **1** dir.

  - Single directory for all users.
  - [I] high probability of name matching.

- Hierarchical (tree)
  - **1** dir. with **n** entries
  - **1** entry with **1** dir.

  - Hierarchy and grouping.
    - Efficient search.
  - Absolute names.
  - Relative names:
    - Working directory

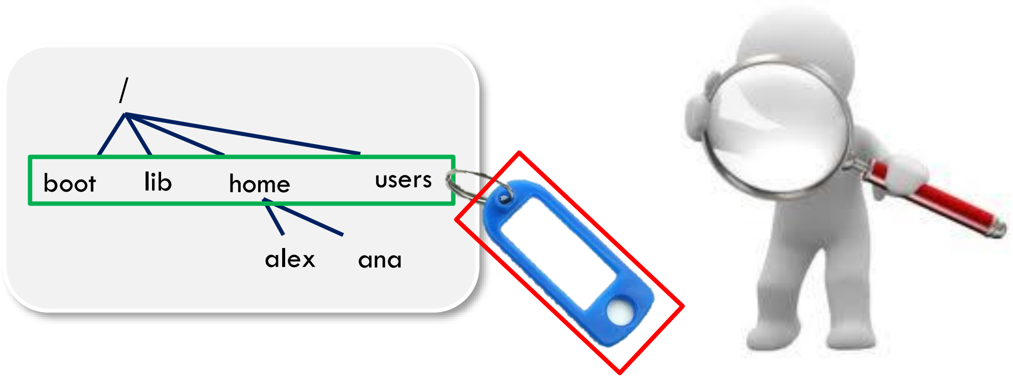
- A-cyclic tree
  - **1** dir. with **n** entries
  - **1** entry with **n** dir.

  - Possible sharing of files and subdirs.
  - Use of the link concept.
    - Important: avoid loops in links.
  - Physical/Hard or soft/symbolic:
    - [I] Physicals within the same file syst.
    - [V] Delete physical decrements counter and only when it reaches 0 it is deleted.
    - [I] Not physical to directory.

Contents

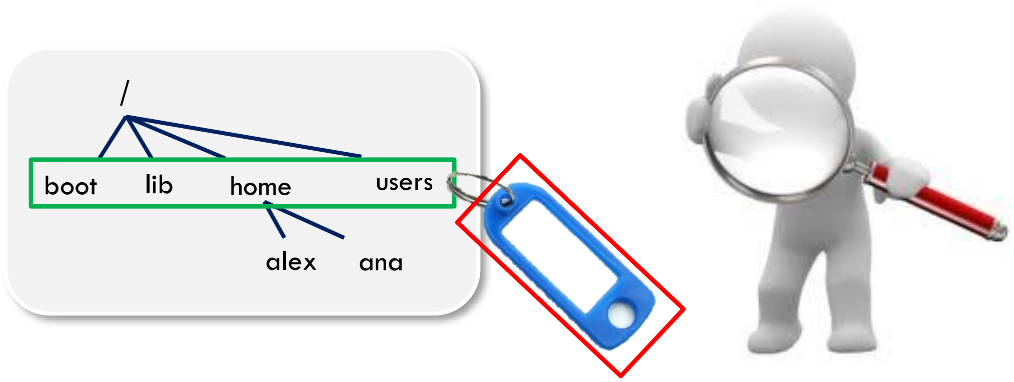
Directory (folders)

- Information of a directory:
  - Data
    - $\overbrace{\text{“special file” whose content is a list of the entries it contains.}}^{\text{file | directory}}$
  - Metadata
    - **Information about the directory itself.**
    - Different **attributes** about the directory (+ information used by the O.S.)



Directory (folders)

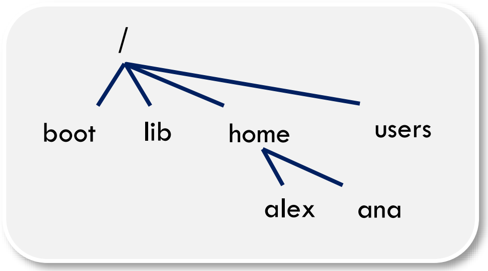
- Information of a directory:
  - Data
    - $\overbrace{\text{“special file” whose content is a list of the entries}}^{\text{file | directory}}$ **it contains.**
  - Metadata
    - **Information about the directory itself.**
    - Different **attributes** about the directory (+ information used by the O.S.)



Directories: attributes

important!

- Typical attributes of a directory:
  - Name: identifier for the users of the directory.
  - Size: number of files in the directory.
  - Protection: control of which user can read, access, etc.
  - Day and time: time stamp of last access, creation, etc. that allows monitoring the use of the directory.
  - User Identification: identifier of the creator, etc.

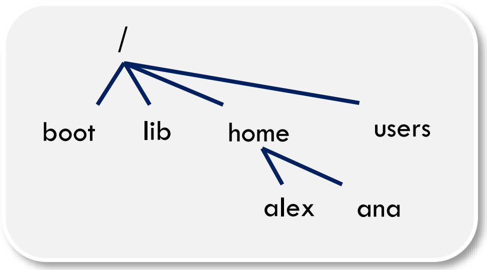


Contents

Important!

Directories: interface

- Generic interface for directory management:
  - mkdir (name, mode)
  - rmdir (name)
  - chdir (name)
  - getcwd (name, name_lenght)
  - descriptor ← opendir (name)
  - closedir (descriptor)
  - structure ← readdir (descriptor)
  - rewindir (descriptor)
  - unlink (name)
  - rename (old_name, new_name)



Example: list entries from /tmp

list /tmp

```c
#include <unistd.h>
#include <sys/types.h>
#include <dirent.h>
#include <stdio.h>

int main ( int argc, char *argv[] )
{
  DIR *dir1 ;
  struct dirent *dp ;
  char name[256] ;
  int ret ;

  ret = chdir ("/tmp/") ;
  if (ret < 0) exit(-1) ;

  getcwd (name, 256);
  printf("%s\n",name);

  dir1 = opendir (name);
  if (NULL == dir1) exit(-1) ;
  while ( (dp = readdir (dir1)) != NULL) {
     printf("%/%s\n", name, dp->d_name);
  }
  closedir (dir1);

  return (0) ;
}
```

Change working directory

Print the current working directory

Open a directory to work with it

Read directory entries and print the name of each entry

Close the working directory

important!

Example: is a file or is a directory?

read from argv[1]

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <stdio.h>

int main ( int argc, char *argv[] )
{
  DIR *dir1 ;
  struct dirent *dp ;
  struct stat s ;

  dir1 = opendir (argv[1]);
  if (NULL == dir1) {
     perror("opendir:");
     return (-1);
  }

  while ( (dp = readdir (dir1)) != NULL) {
      stat(dp->d_name,&s);
      if (S_ISDIR(s.st_mode))
           printf("dir: %s\n",dp->d_name);
      else printf("fch: %s\n",dp->d_name);
  }

  closedir (dir1);
  return (0) ;
}
```

Open a directory to work with it

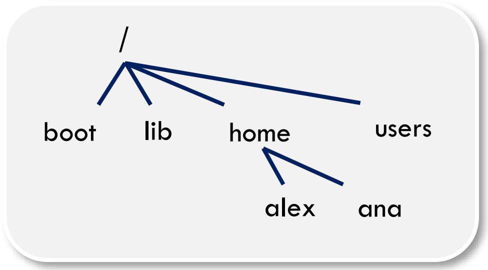
Read entries in the directory...

… for each entry, get the metadata of the entry and print whether it is a file or directory along with the name of the entry

Close the directory you have worked with

Directories: interface

- Generic interface for directory management:
  - mkdir (name, mode)
  - rmdir (name)
  - chdir (name)
  - getcwd (name, name_length)
  - descriptor ← opendir (name)
  - closedir (descriptor)
  - structure ← readdir (descriptor)
  - rewindir (descriptor)
  - unlink (name)
  - rename (old_name, new_name)



OPENDIR – Open a directory

Service	`#include <sys/types.h>`
	`#include <dirent.h>`
	
	`DIR *`**`opendir`**` ( char *dirname );`

Arguments	◻ `dirname` pointer to the directory name
Returns	A pointer to be used in `readdir`(), `closedir`(), etc. or `NULL` if there was an error.
Description	◻ Opens a working session with a directory so that you can work with the directory entries. ◻ It is positioned at the first entry.

READDIR – Reading directory entries

Service	`#include <sys/types.h>`
	`#include <dirent.h>`
	
	`struct dirent *`**`readdir`**` ( DIR *dirp );`

Arguments	▫ `dirp` pointer returned by `opendir()`.
Returns	A pointer to a struct `dirent structure` representing a directory entry or `NULL` if there was an error.
Description	▫ Returns the next directory entry associated with `dirp`. ▫ Advances the pointer to the next entry. ▫ The structure is implementation dependent. It should be assumed that only one member is fetched: `char *d_name`.

REWINDDIR – Position to 1st entry

Service	`#include <sys/types.h>`
	`#include <dirent.h>`
	
	`void *`**`rewinddir`**` ( DIR *dirp );`

Arguments	▫ `dirp` pointer returned by `opendir()`.
Returns	Nothing.
Description	▫ Places the position pointer inside the directory in the first entry.

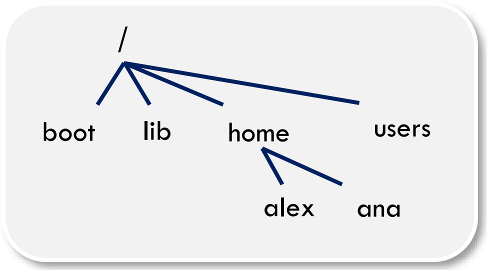
CLOSEDIR – Close a directory

Service	`#include <sys/types.h>`
	`#include <dirent.h>`
	
	`int *`**`closedir`**` ( DIR *dirp );`

Arguments	▫ `dirp` pointer returned by `opendir()`
Returns	Zero if all OK or -1 if error occurred.
Description	▫ Closes the work session with the directory.

Directories: interface

- Generic interface for directory management:
  - mkdir (name, mode)
  - rmdir (name)
  - chdir (name)
  - getcwd (name, name_length)
  - descriptor ← opendir (name)
  - closedir (descriptor)
  - structure ← readdir (descriptor)
  - rewindir (descriptor)
  - unlink (name)
  - rename (old_name, new_name)



MKDIR – Create a directory

Service	`#include <sys/types.h>`
	`#include <dirent.h>`
	
	`int` **`mkdir`** `( const char *name, mode_t mode );`

Arguments	▫ `name` directory name. ▫ `mode` protection bits.
Returns	Zero if all OK or -1 if error occurred.
Description	▫ Creates a directory named `name`. ▫ UID_owner = UID_effective ▫ GID_owner = GID_effective

RMDIR – Deletes a directory

Service	`#include <sys/types.h>`
	`#include <dirent.h>`
	
	`int` **`rmdir`** `( const char *name );`

Arguments	▫ `name` directory name.
Returns	Zero if all OK or -1 if error occurred.
Description	▫ Deletes the directory if it is empty. ▫ If the directory is not empty, it is not deleted.

CHDIR – Change the current directory

Service	`#include <sys/types.h>`
	`#include <dirent.h>`
	
	`int` **`chdir`** `( const char *name );`

Arguments	▫ `name` directory name.
Returns	Zero if all OK or -1 if error occurred.
Description	▫ Modifies the current working directory. ▫ Relative names are formed from the current directory.

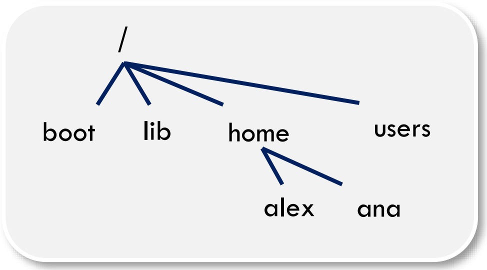
GETCWD – Get current directory

Service	`#include <sys/types.h>`
	`#include <dirent.h>`
	
	`char *`**`getcwd`**` ( char *buf, size_t bufSize );`

Arguments	▫ `buf` pointer to the memory space where to store the name of the current working directory. ▫ `bufMaxSize` size in bytes of the buffer `buf`.
Returns	Pointer to `buf` filled or `NULL` if error.
Description	▫ Gets the name of the current working directory, stores it in `buf` and returns `buf` (where it is stored). ▫ If the name is larger than the size of `buf` then it may truncate the name (due to space limitation).

Directories: interface

- Generic interface for directory management:
  - mkdir (name, mode)
  - rmdir (name)
  - chdir (name)
  - getcwd (name, name_length)
  - descriptor ← opendir (name)
  - closedir (descriptor)
  - structure ← readdir (descriptor)
  - rewindir (descriptor)
  - unlink (name)
  - rename (old_name, new_name)



RENAME – Change the name of a file

Service	`#include <unistd.h>`
	
	`int` **`rename`** `( char *old, char *new );`

Arguments	◻ `old` pointer to the array of characters containing the current file name to change. ◻ `new` pointer to the array of characters containing the new file name.
Returns	Zero or -1 if error.
Description	◻ Renames the current `old` file name to `new` name

UNLINK – Remove directory entry

Service	`#include <unistd.h>`
	
	`int` **`unlink`** `( char *name );`

Arguments	▫ `name` pointer to the array of characters containing the name of the directory entry to try to delete.
Returns	Zero or -1 if error.
Description	▫ Removes the directory entry and decrements the number of links of the corresponding file. ▫ When the number of links equals zero and no process keeps it open, the space occupied by the file is freed and the file is no longer accessible. If any process keeps it open, then it waits until it closes it to free the space.

LINK – Create a directory entry

Service	`#include <unistd.h>`
	
	`int **link** ( const char *current, const char *new );`

Arguments	▫ `current` pointer to the character array containing the name of the existing directory entry to work with. ▫ `new` pointer to the character array containing the name of the new directory entry to link to the `current` one.
Returns	Zero or -1 if error.
Description	▫ Creates a new link, physical or symbolic, for an existing file. ▫ The system does not save what the original link is. ▫ `current` directory name must not be the name of a directory unless: a) you have sufficient privilege and b) your implementation supports the directories link.

SYMLINK – Creation of soft link

Service	`#include <unistd.h>`
	
	`int` **`symlink`** `( const char* oldpath,`
	`                const char* newpath );`

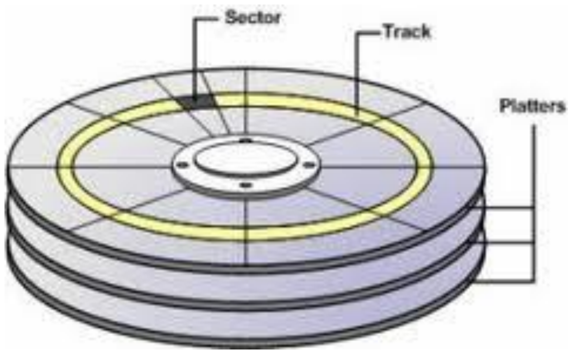
Arguments	◻ `oldpath` name of the existing file to link. ◻ `newpath` name of the soft link to create.
Returns	Returns 0 if all went well or -1 if error.
Description	◻ Creates a soft or symbolic link to an existing entry (file or directory). ◻ It can link entries from another partition, but if it is deleted, access to the contents is lost.

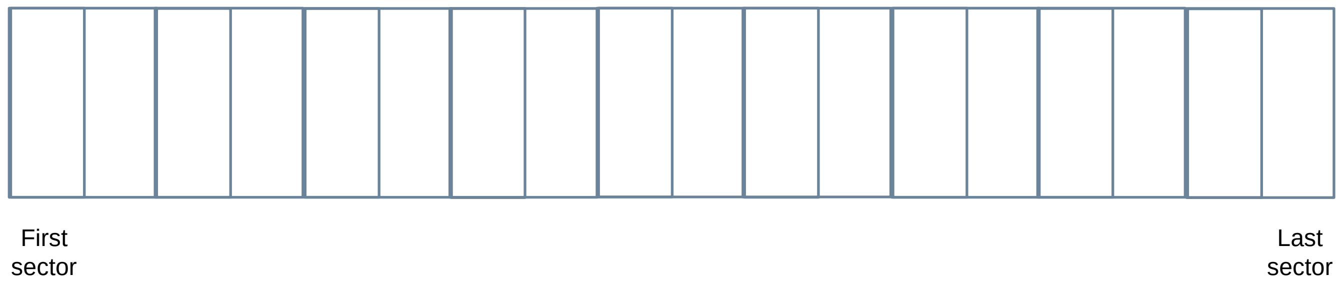
Contents

important!

Sectors

- The storage device is divided into **sectors**, tracks, and cylinders.

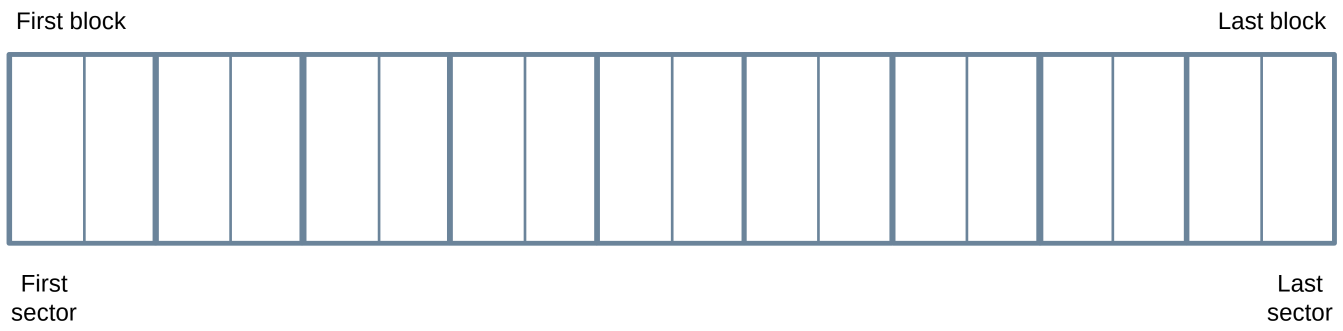




important!

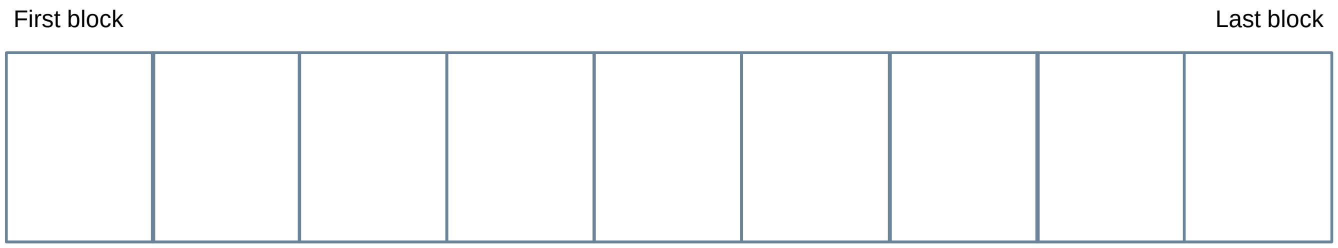
Blocks

- **Block**: *logical grouping of disk sectors (2^n sectors)*
  - It is the minimum transfer unit used by the O.S.
  - Optimize input/output efficiency of devices.
  - Users can define the block size when creating the file system, or use the one offered by default in the O.S.



Blocks

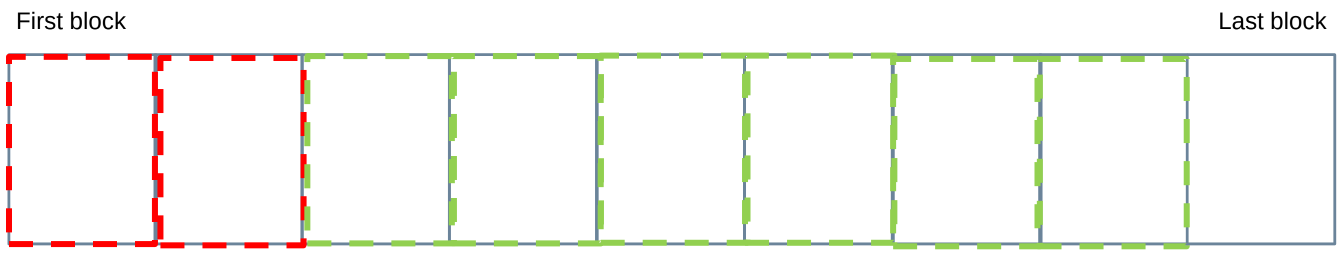
- **Block**: *logical grouping of disk sectors (2^n sectors)*
  - It is the minimum transfer unit used by the O.S.
  - Optimize input/output efficiency of devices.
  - Users can define the block size when creating the file system, or use the one offered by default in the O.S.



important!

File System

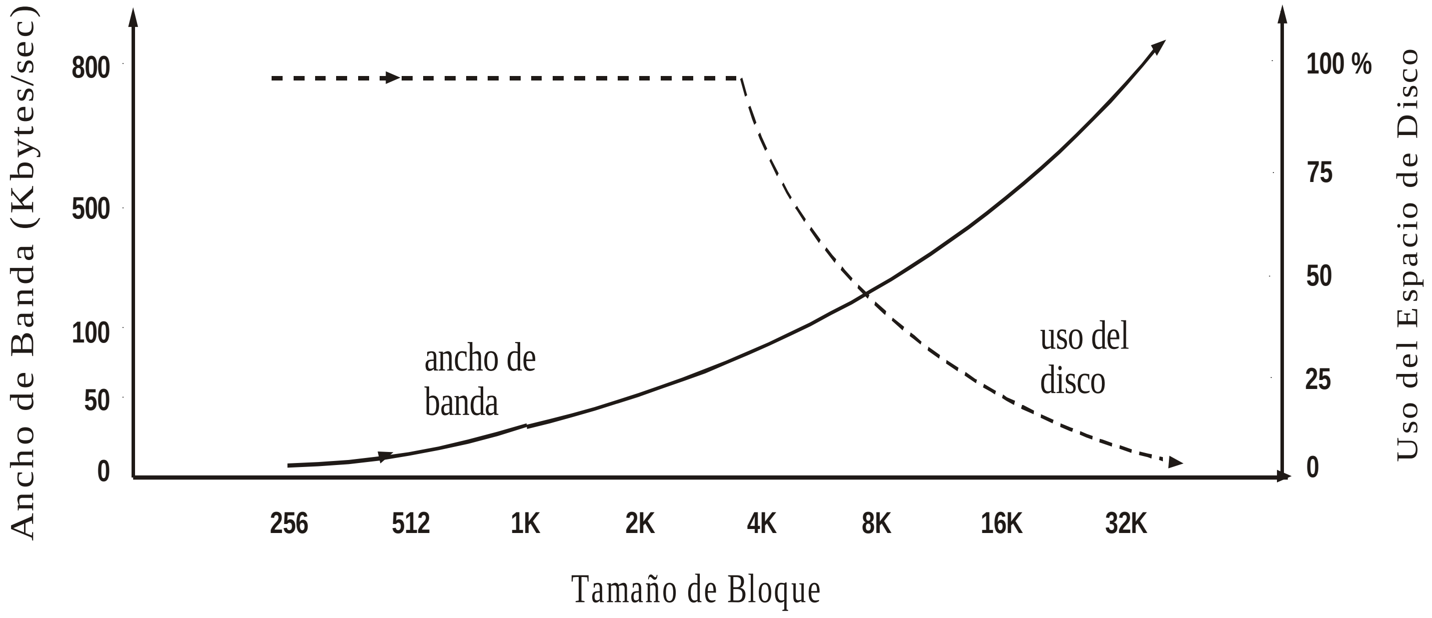
- The file system allows information on storage devices to be organized in a meaningful intelligible to the operating system:
  - Allows you to define an allocation unit (block/grouping).
  - Allows the management of free and occupied space (allocation of disk space to each file).



File System

- The file system allows information on storage devices to be organized in a meaningful intelligible to the operating system:
  - Allows you to define an allocation unit.
    - **Which** allocation unit is used?
  - Allows the management of free and occupied space (allocation of disk space to each file).
    - **What** data structure is used for file allocation?
    - **Is** the maximum space **allocated** on creation or dynamically?
      - **Pre-allocation**: maximum space allocation on creation.
      - **Dynamic**: space allocation as needed.

Block size

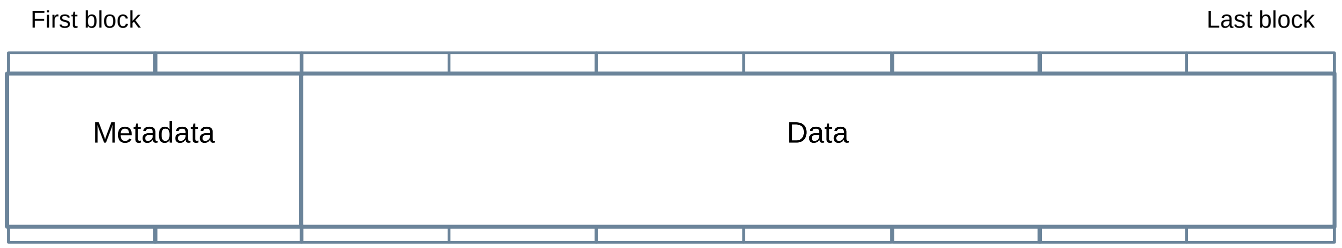


- The choice of block size is important for balancing:
  - Bandwidth: higher number of sectors initially, better bandwidth
  - Disk utilization: fewer sectors, less internal fragmentation

important!

File System

- The file system allows information on storage devices to be organized in a format intelligible to the operating system:
  - It is a *coherent set of meta-information and data.*



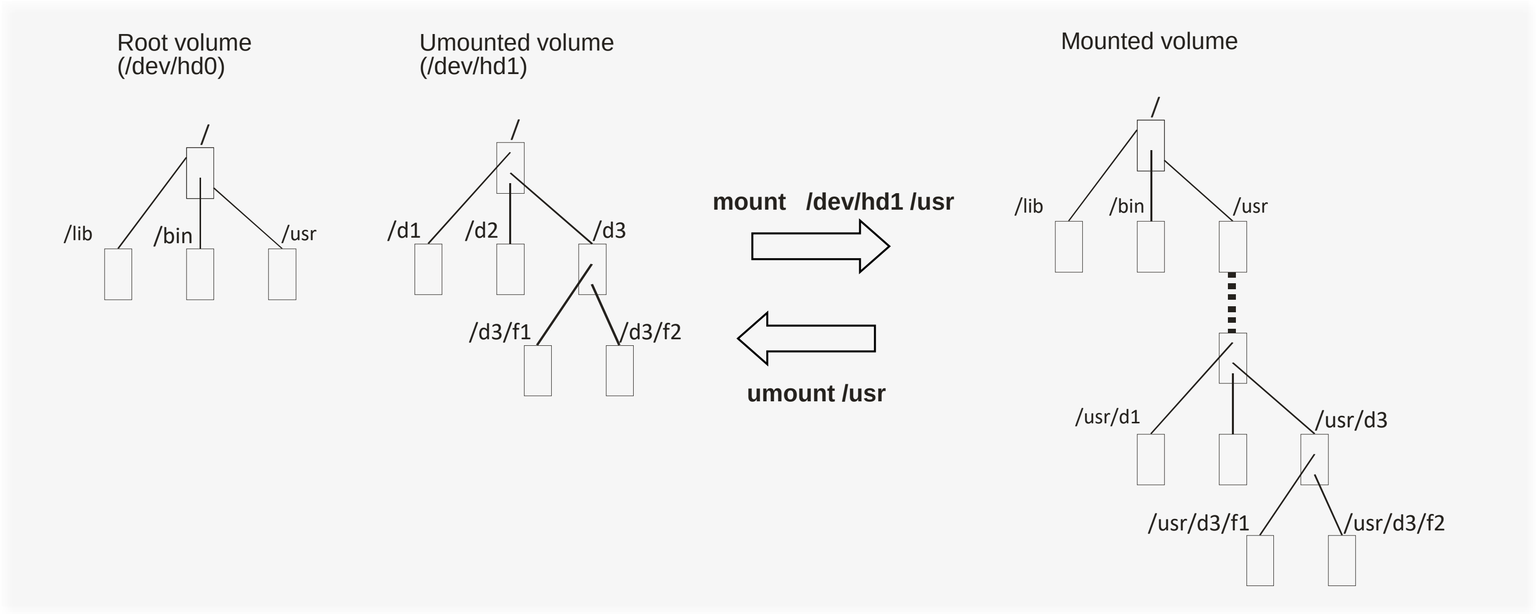
File System: attributes

important!

- Typical attributes of a file system:
  - Sizes used:
    - Number of blocks: number of managed blocks (data + metadata)
    - Block size: block size (in bytes or sectors).
    - Number of entries: number of entries (files and directories) managed.
    - Metadata area size: number of dedicated blocks.
  - Free space management: identification of which block is free.
  - Entry management: for each entry (file or directory) a space is reserved for the metadata describing it:
    - General attributes: dates, permissions, user ID, etc.
    - Attributes for management of used blocks: blocks used by this entry.
  - Reference to the root directory entry: identification of the entry containing the root directory information.

File System: operations

- **File system operations:**
  - Create
  - Mount (in single tree like Unix or in d:, e: drive like ms-dos)
  - Umount



File System

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

- Large number of file systems…

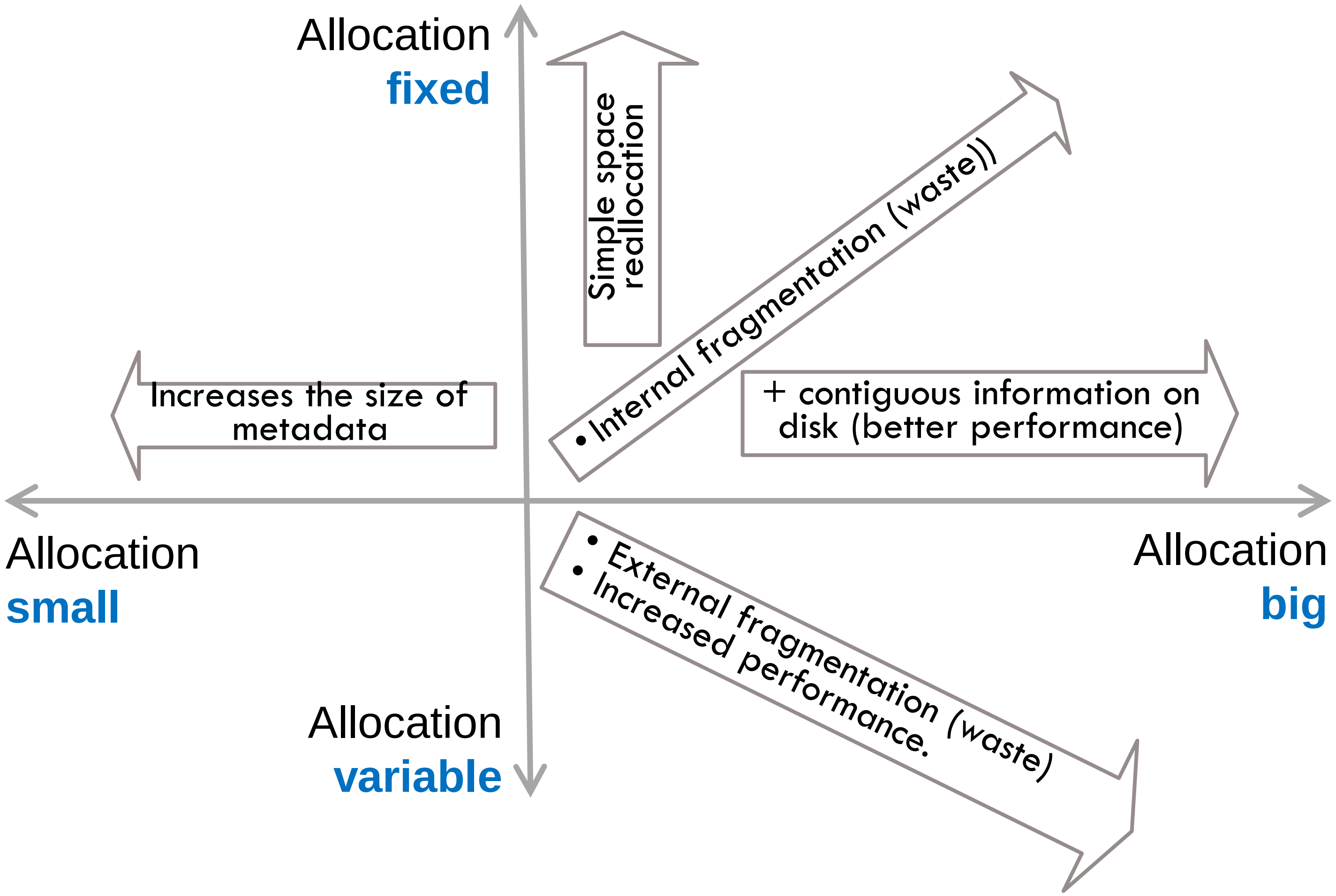
- For storage devices:
  - minix (Minix)
  - ext2 (Linux)
  - ext3 (Linux)
  - ufs (BSD)
  - fat (DOS)
  - vfat (win 95)
  - hpfs (OS/2)
  - hfs (Mac OS)
  - ntfs (win NT/2K/XP)
  - …

- Virtual:
  - procfs (/proc)
  - devFS (/dev)
  - umsdos (Unix sobre DOS)
  - …

- Networking:
  - NFS
  - CODA
  - SMBFS
  - NCPFS (Novell)
  - …

Allocation size

important!

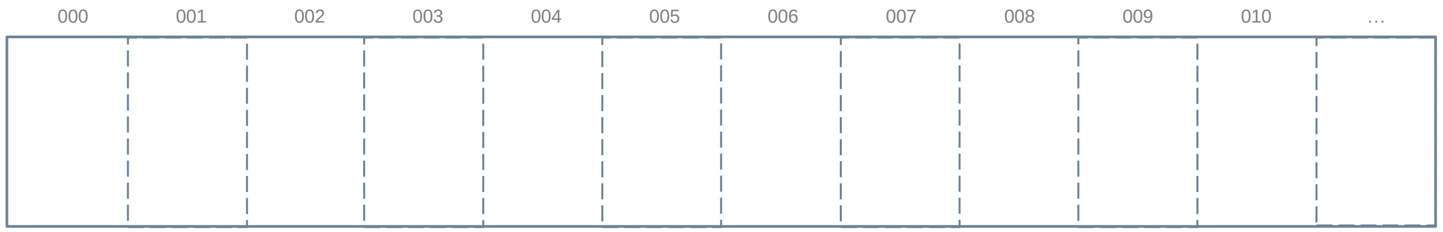

Allocation
fixed
Allocation
small
Allocation
big
Allocation
variable
Simple space reallocation
• Internal fragmentation (waste))
Increases the size of metadata
+ contiguous information on disk (better performance)
• External fragmentation (waste)
• Increased performance.

File System:

representation used in Minix

important!

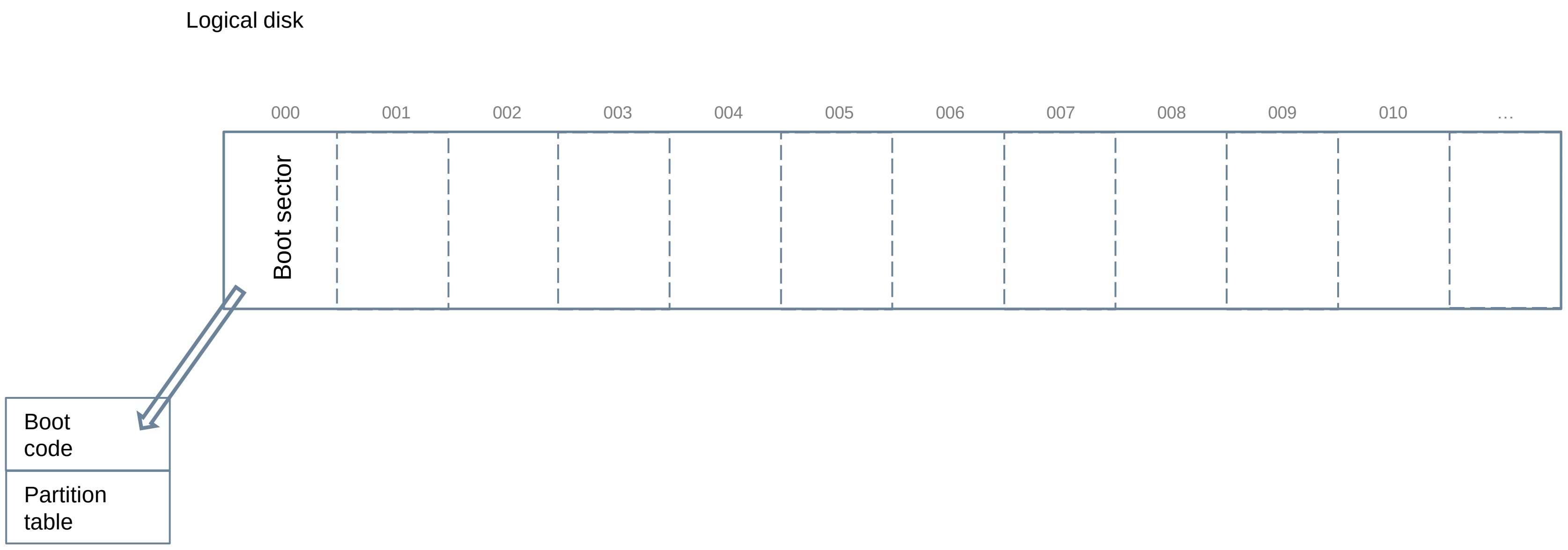
Logical disk



File System:

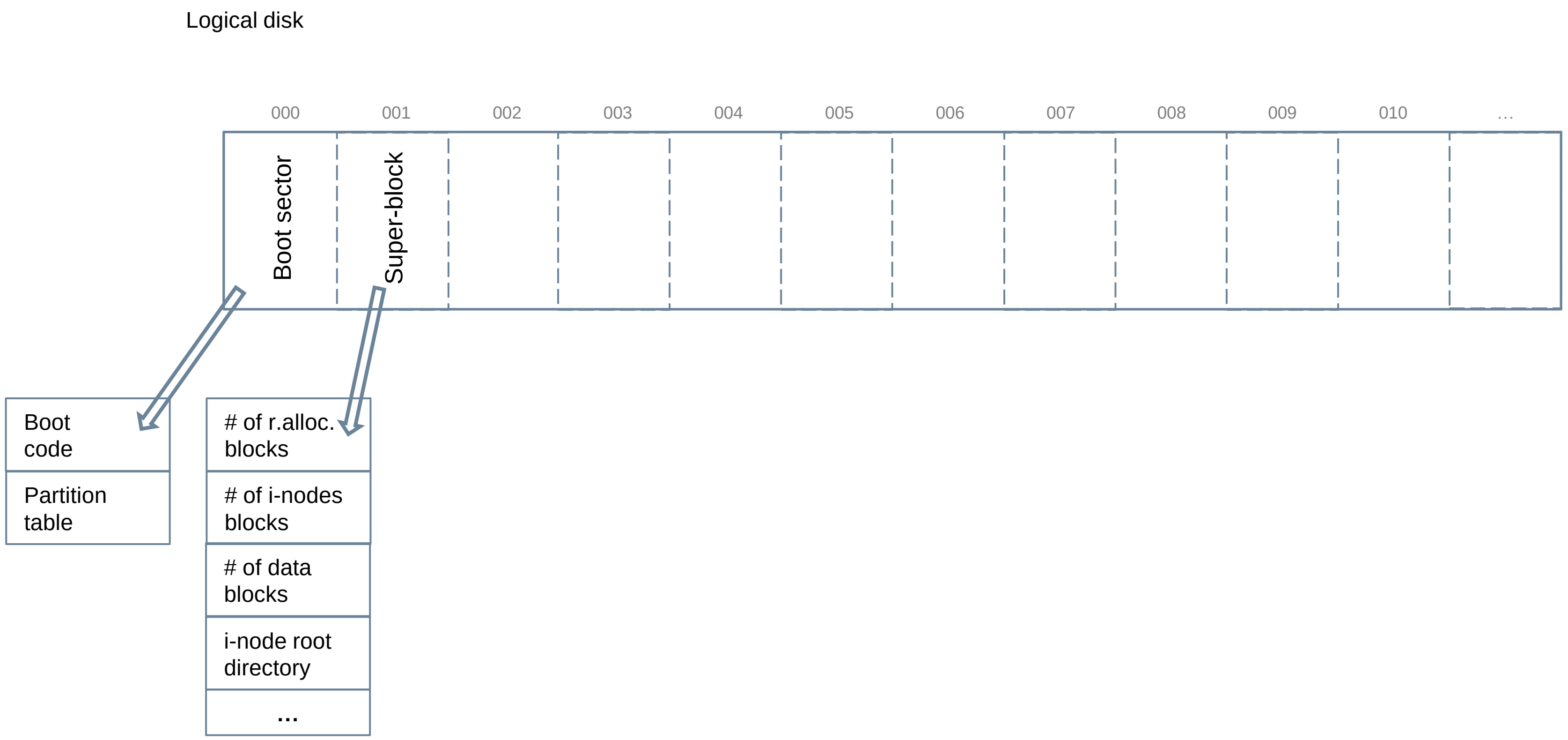
representation used in Minix

important!



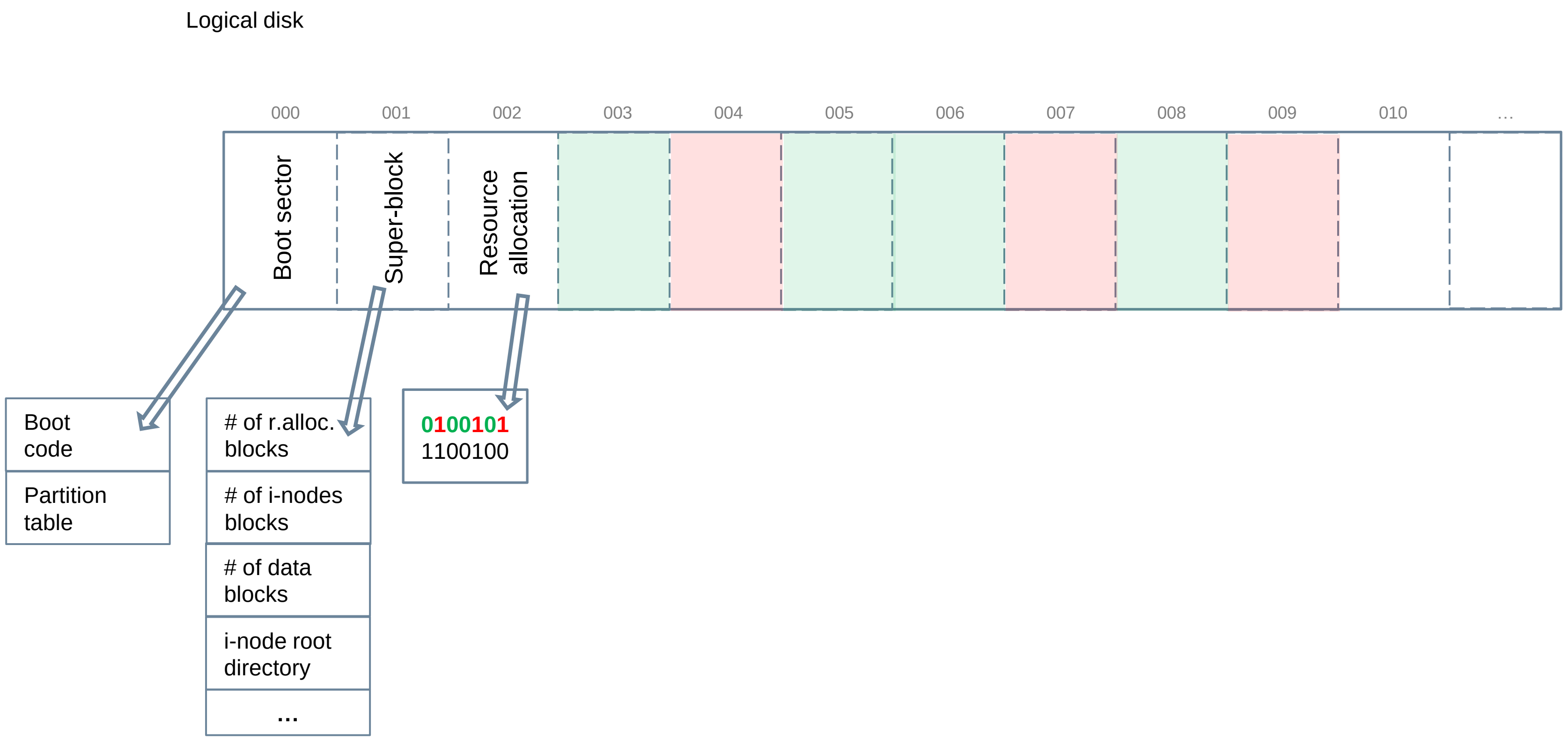
File System:
representation used in Minix

important!



File System:

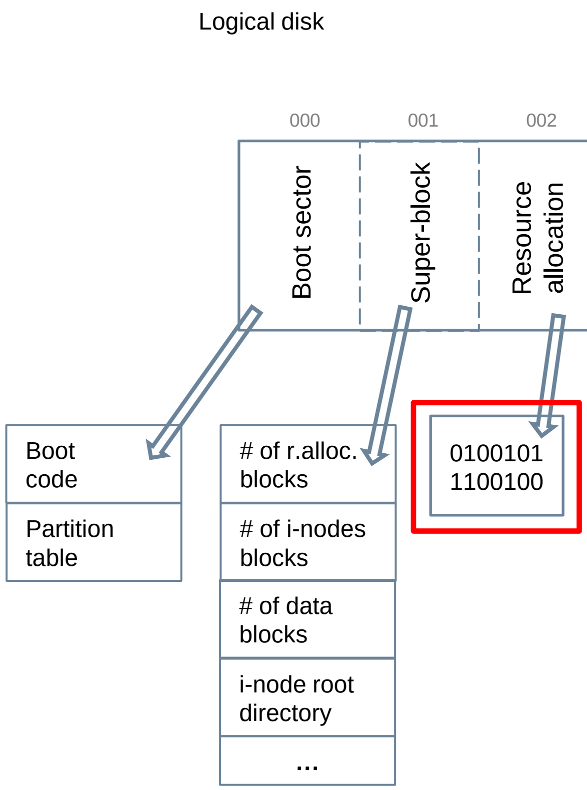
representation used in Minix



File System:

representation used in Minix

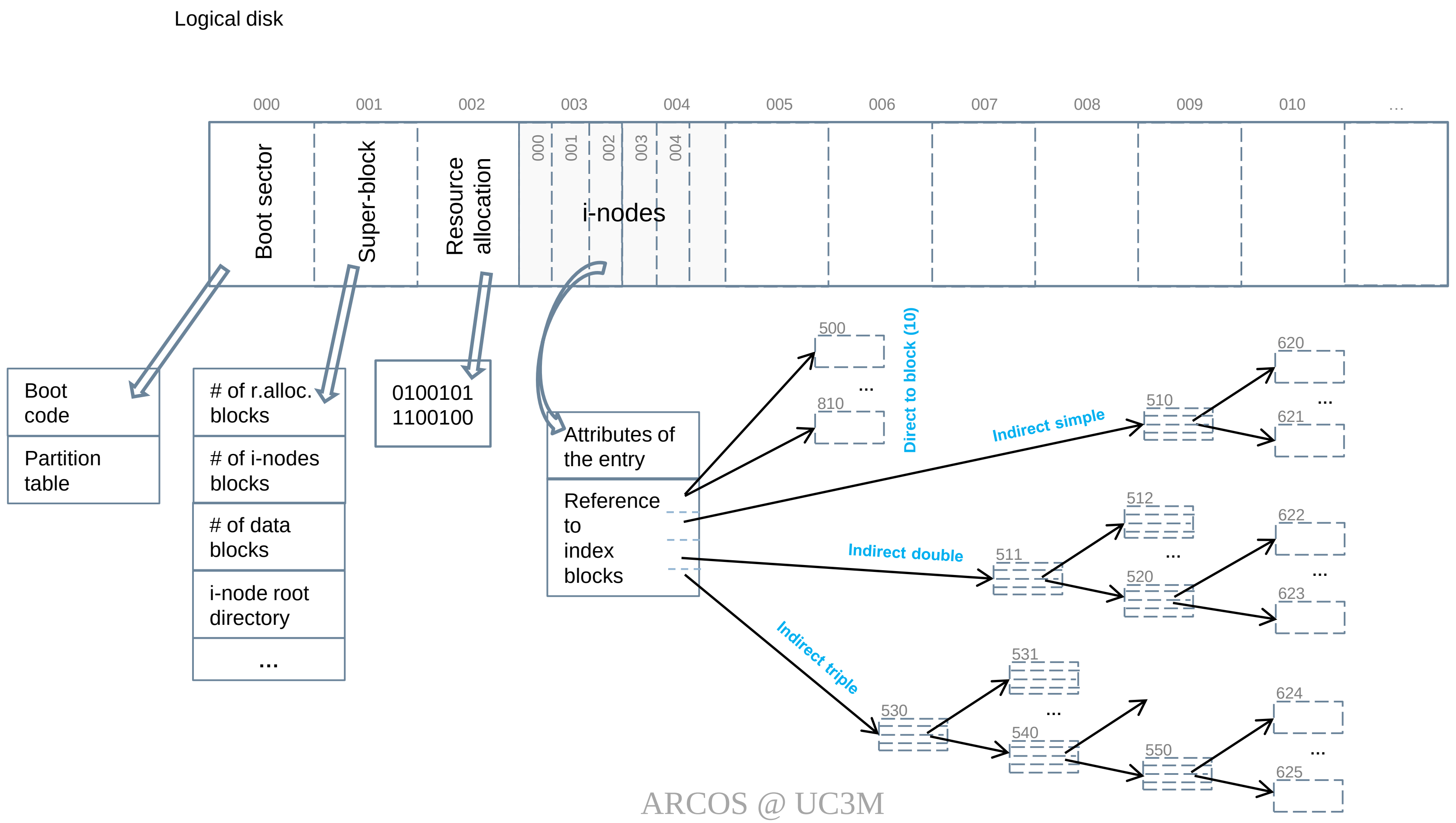
important!



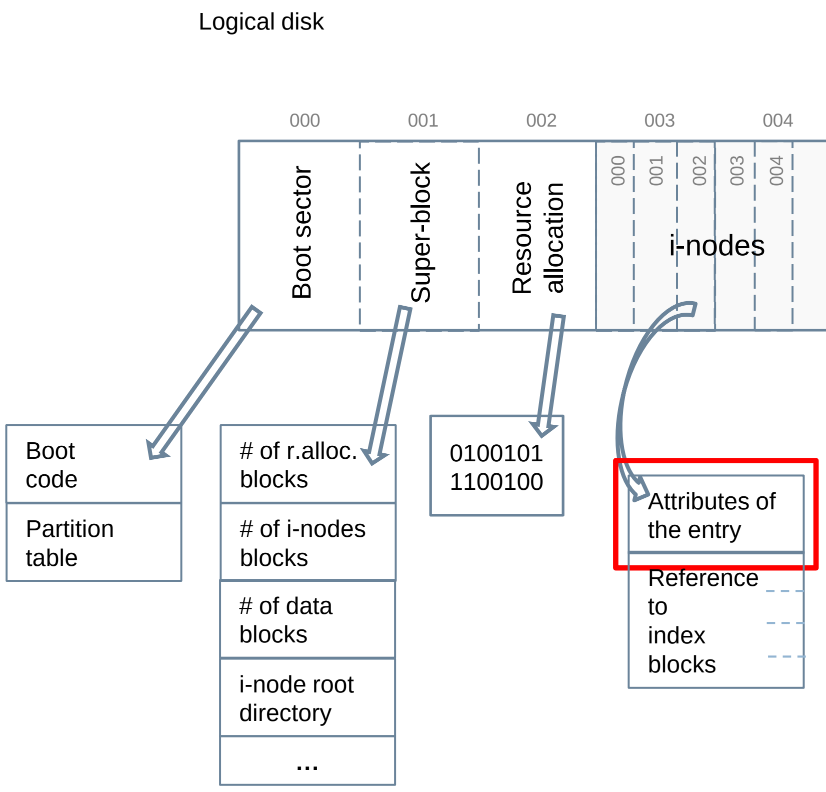
- <span></span>Bitmaps or bit vectors:
  - Vector with one bit per existing resource.
    If free resource then bit with value 1, if busy then value is 0.
    - Easy to implement and simple to use.
    - Efficient if the device is not full or too fragmented.
- Linked list of free resources:
  - Keep linked in a list all available resources by keeping a pointer to the first item in the list.
    - Not efficient except for very full and fragmented devices
- Indexing:
  - Index table of free portions.

File System:
representation used in Minix

important!



File System:
representation used in Minix

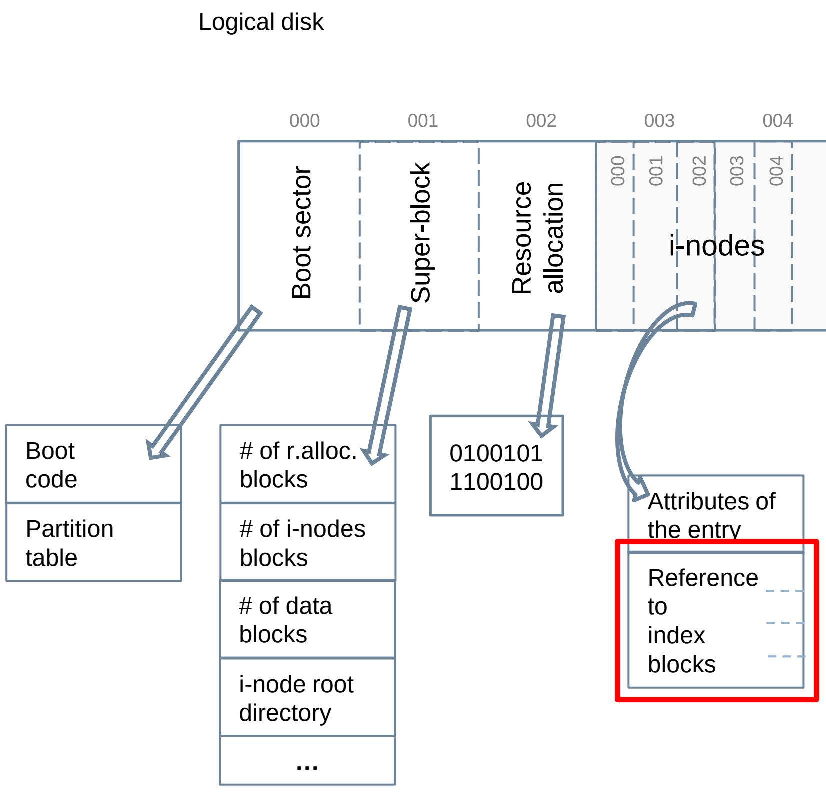


- **Dates and times:**
  - Of creation, modification, access, etc.
- **Size:**
  - In bytes or disk blocks used.
- **User owner and group**
- **Protection:**
  - Attributes, ACL, capabilities, etc.
- **Type of entry** (file, …)
- **Link counter**
- **Etc.**

File System:

representation used in Minix

important!

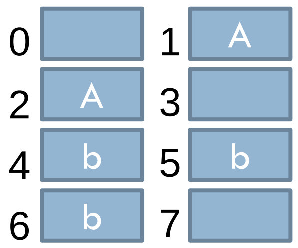


- <span style="color:red">Continuous allocation</span>:
  - A list of consecutive blocks is allocated.
- Discontinuous allocation:
  - The first available block is allocated.
  - Linked/chained mechanism:
    - The next block is indicated at the end of each block.
  - Indexed mechanism:
    - Blocks with indexes of all blocks in the input.

File System:

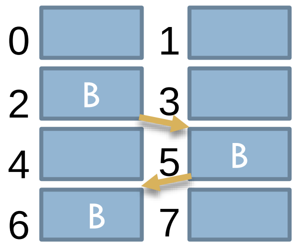
representation used in Minix

important!



F	I	L
A	1	2
B	4	3

- Contiguous allocation:
  - The file blocks are arranged consecutively.
  - Required: first (I) & number of blocks (L)
  - Needs to compact to optimize.



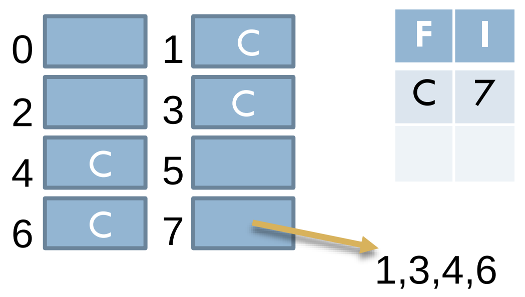
F	I	L
B	2	3

- Linked assignment:
  - Each block contains the reference to the following block (block by block).
  - Required: first (I) & number of blocks (L)
  - Needs to consolidate to optimize.

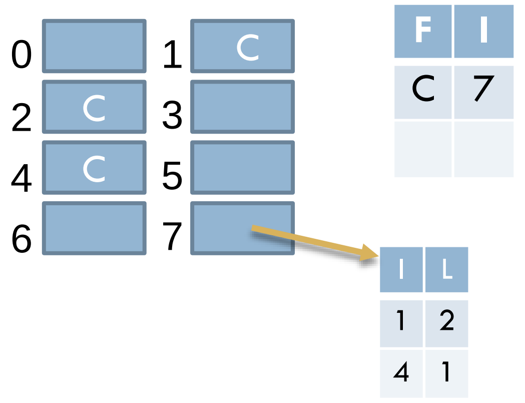
File System:

representation used in Minix

important!



- Indexed allocation (blocks):
  - Blocks are used with references to the blocks that will contain the data.
  - Required: id. of the 1st index block.
  - Defrag.

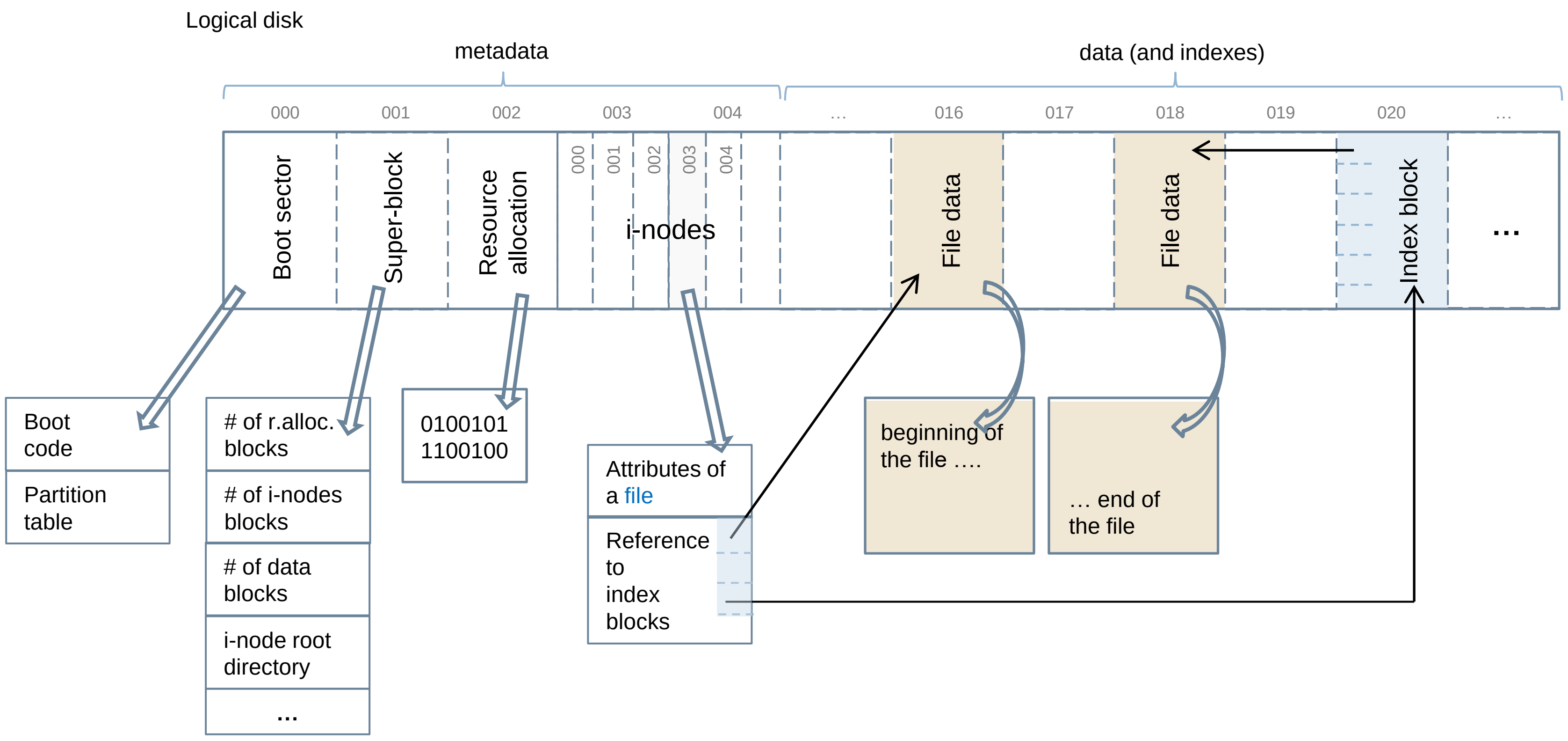


- Indexed allocation (extends):
  - Blocks are used with references at the beginning to the blocks that will contain the data (portions/extends).
  - Required: id. of the 1st index block.
  - Defrag.

File System:

representation used in Minix: files

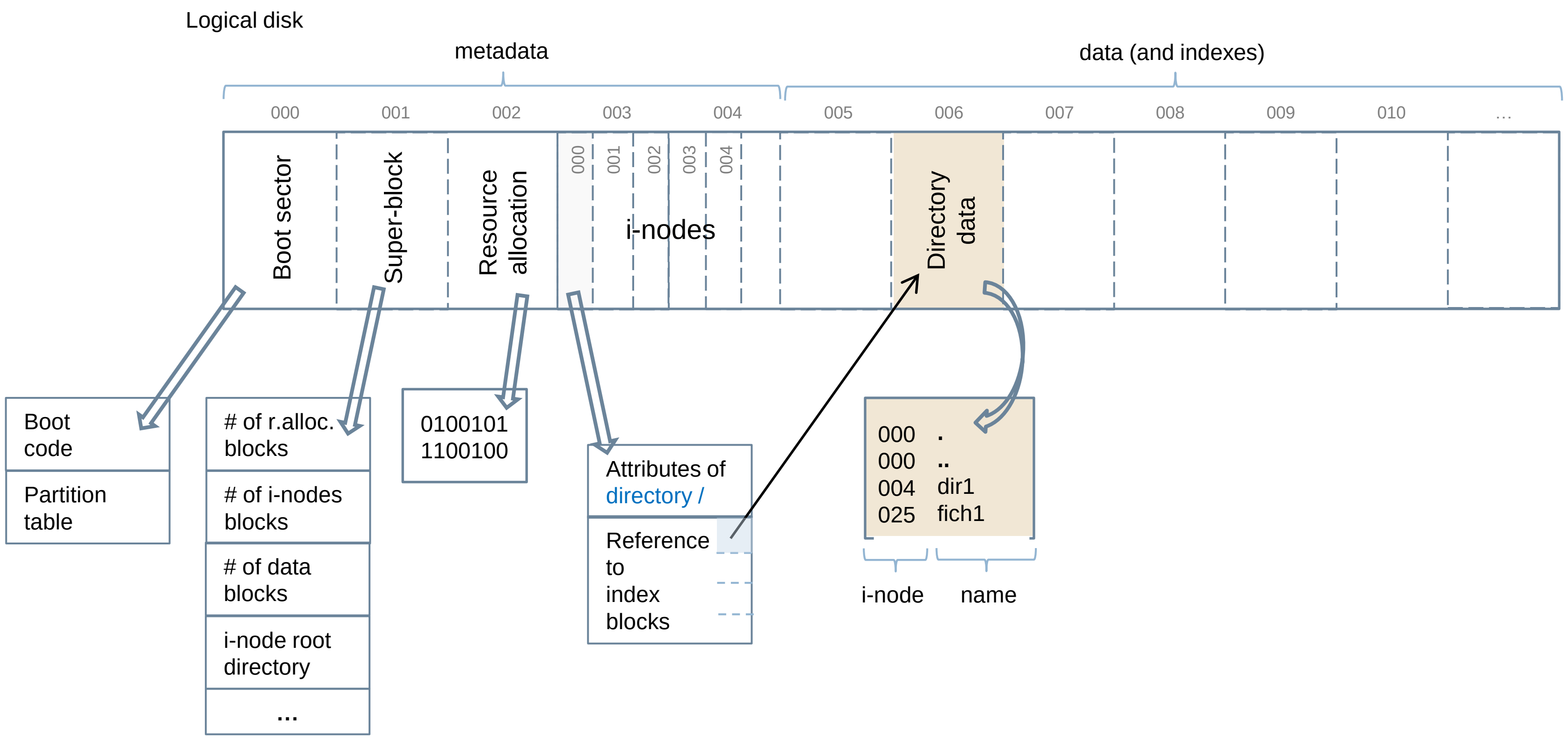
important!



File System:

representation used in Minix: directories

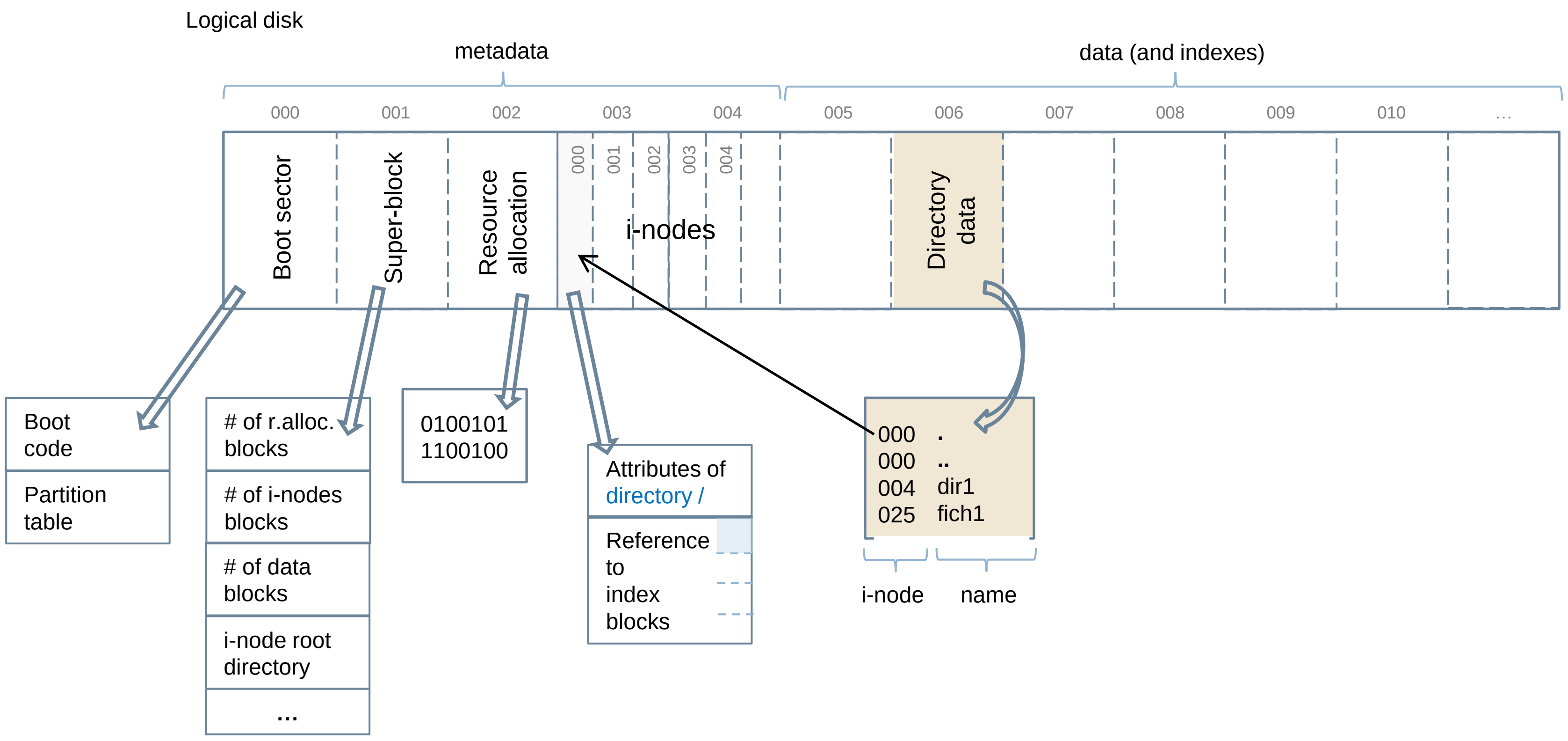
important!



File System:

representation used in Minix: directories

important!

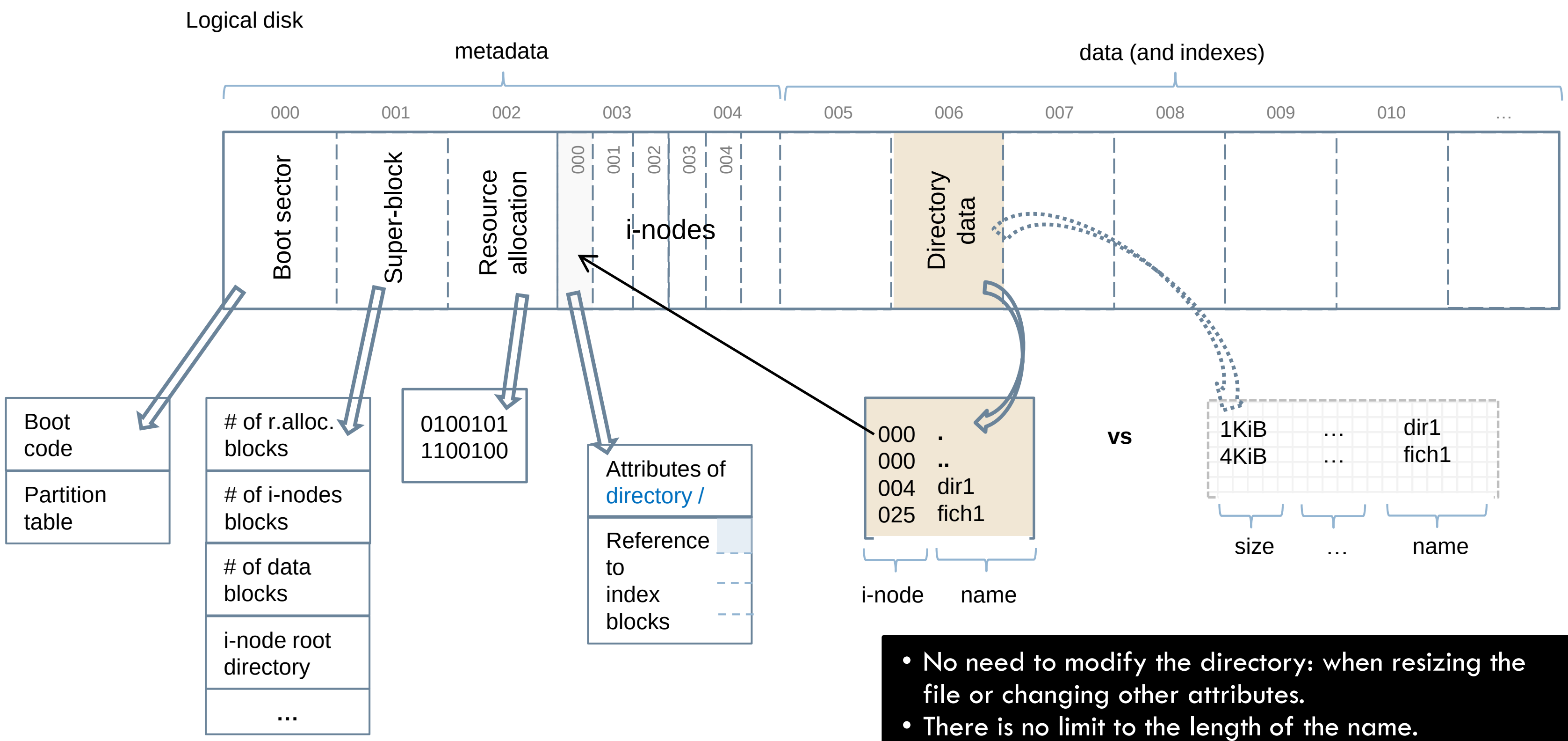


File System:

representation used in Minix: directories

important!

- Directory entry = name + i-node identifier (with all other attributes)

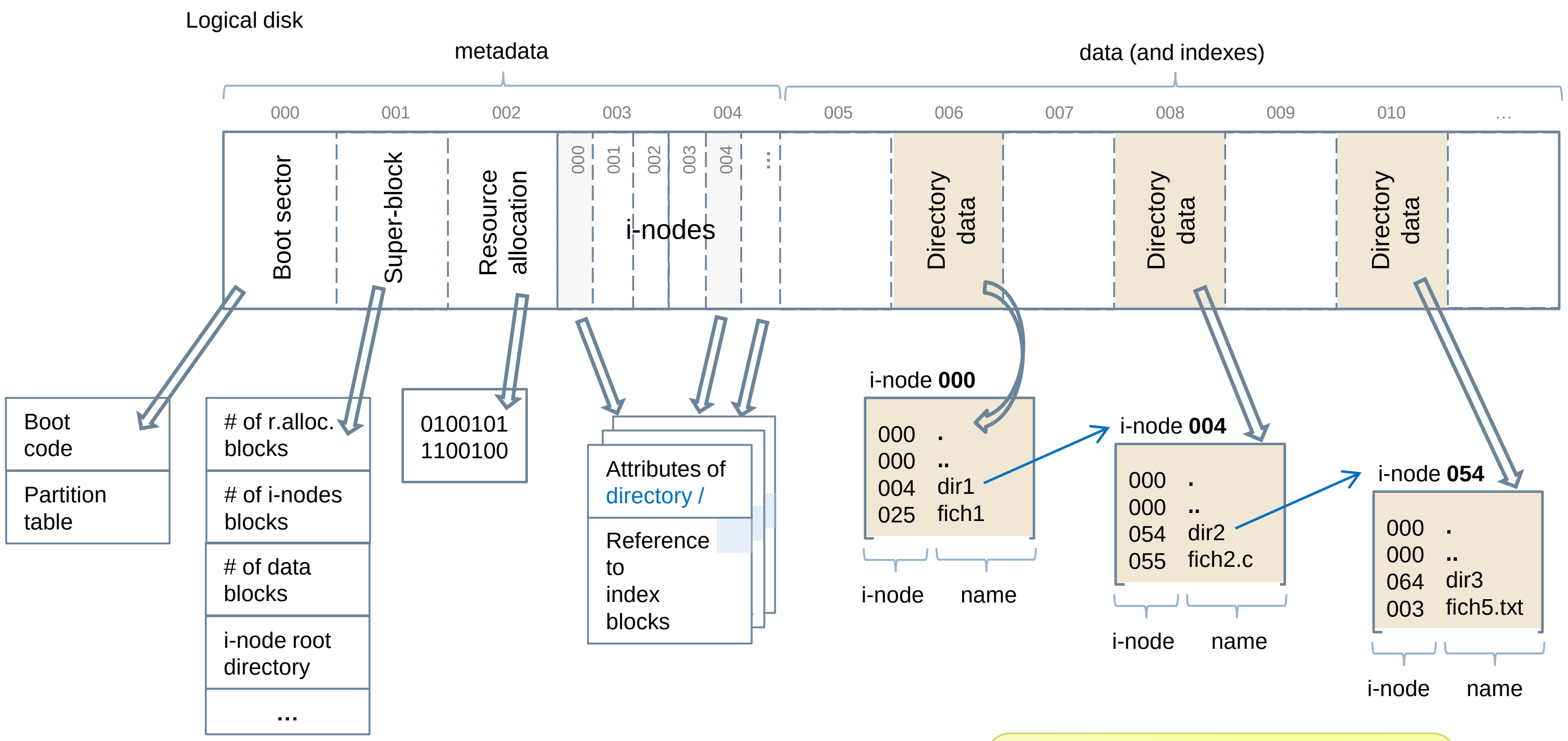


- No need to modify the directory: when resizing the file or changing other attributes.
- There is no limit to the length of the name.
- Easy creation of synonyms for a file (links).
- i-node with file and/or directory attributes

File System:

representation used in Minix: directories

important!



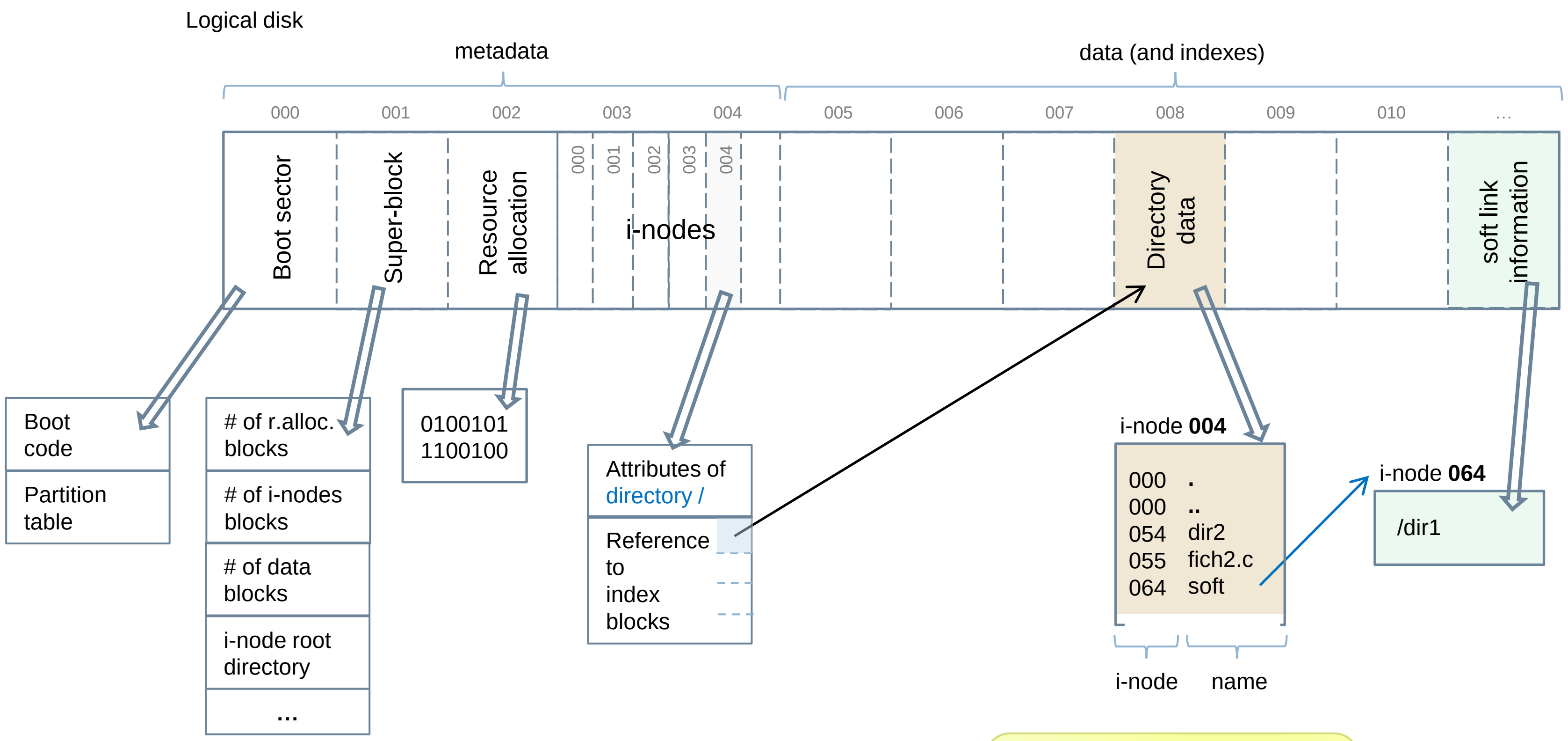
ls –l /dir1/dir2/fich5.txt
- / + dir1 + dir2 + fich5.txt
- 4 i-nodes + 3 data blocks

File System:

representation used in Minix: symbolic link (soft link)

important!

Alejandro Calderón Mateos

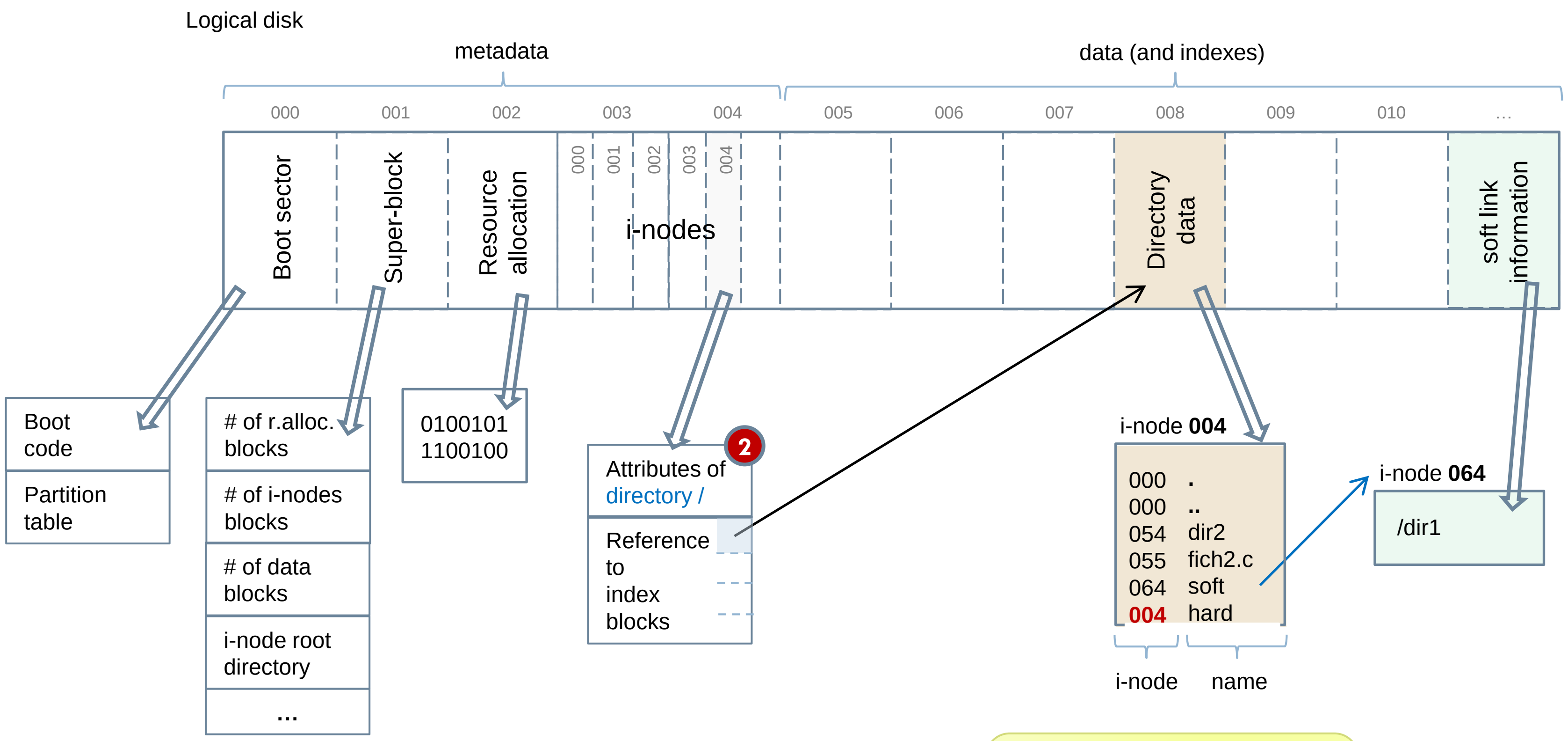


```
ln -s  /dir1  /dir1/soft
```

File System:

representation used in Minix: hard link

important!



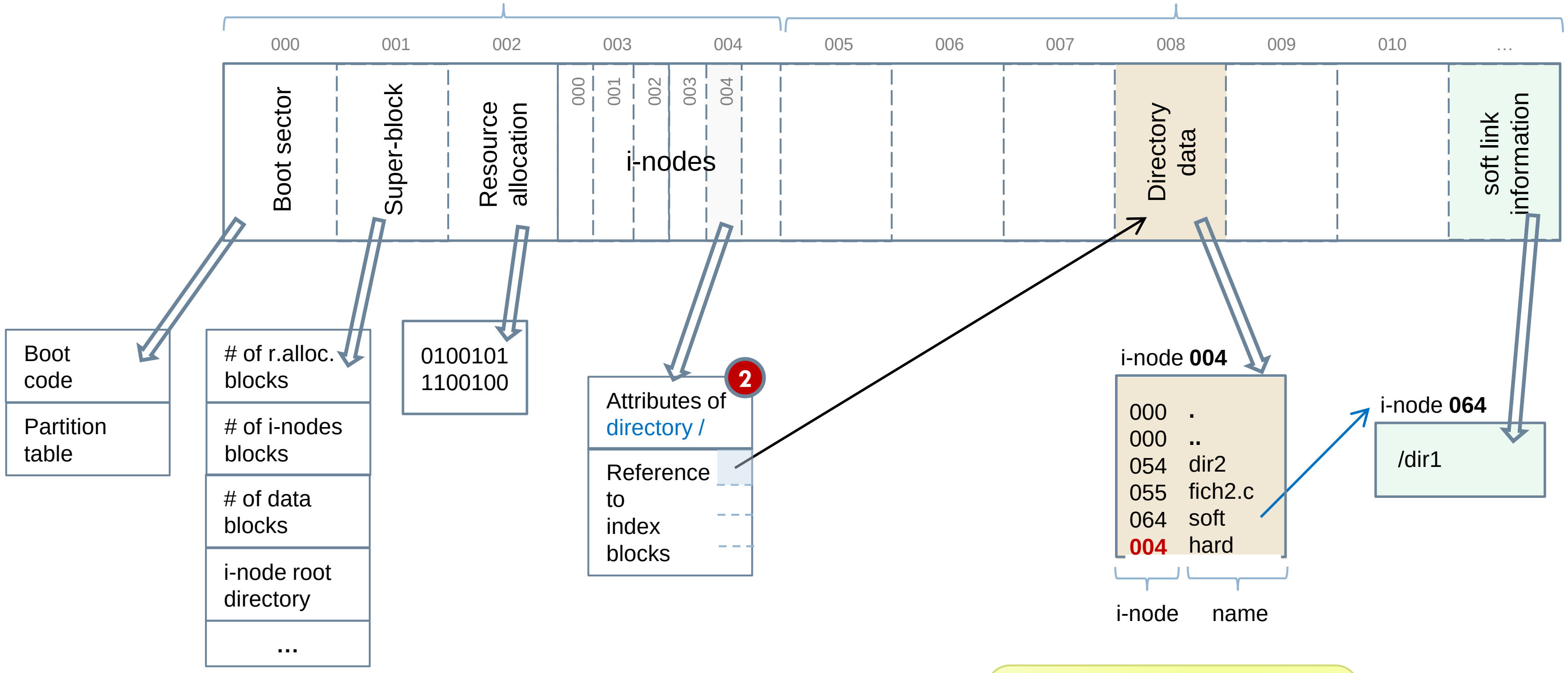
```
ln -s  /dir1  /dir1/soft
ln     /dir1  /dir1/hard
```

File System:

representation used in Minix: hard link

important!

- Physical/hard: new directory entry to an existing i-node (link counter on i-node)
  - Hard/physical links only to other files within the partition
- Symbolic/soft: a new file is created containing the name of the target file/directory.



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

File System:
representation used in FAT: directories

important!

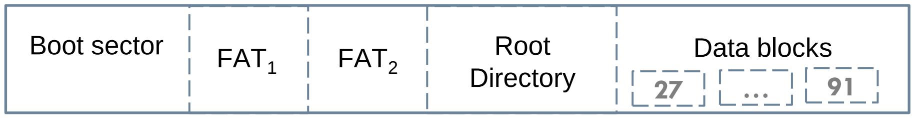
- Directory entry = name + other attributes (used for interleaved and contiguous files)

Root Directory

Name	Attrib.	KB	Grouping
dir1	dir	5	27
fich1.txt		12	45

dir1 Directory

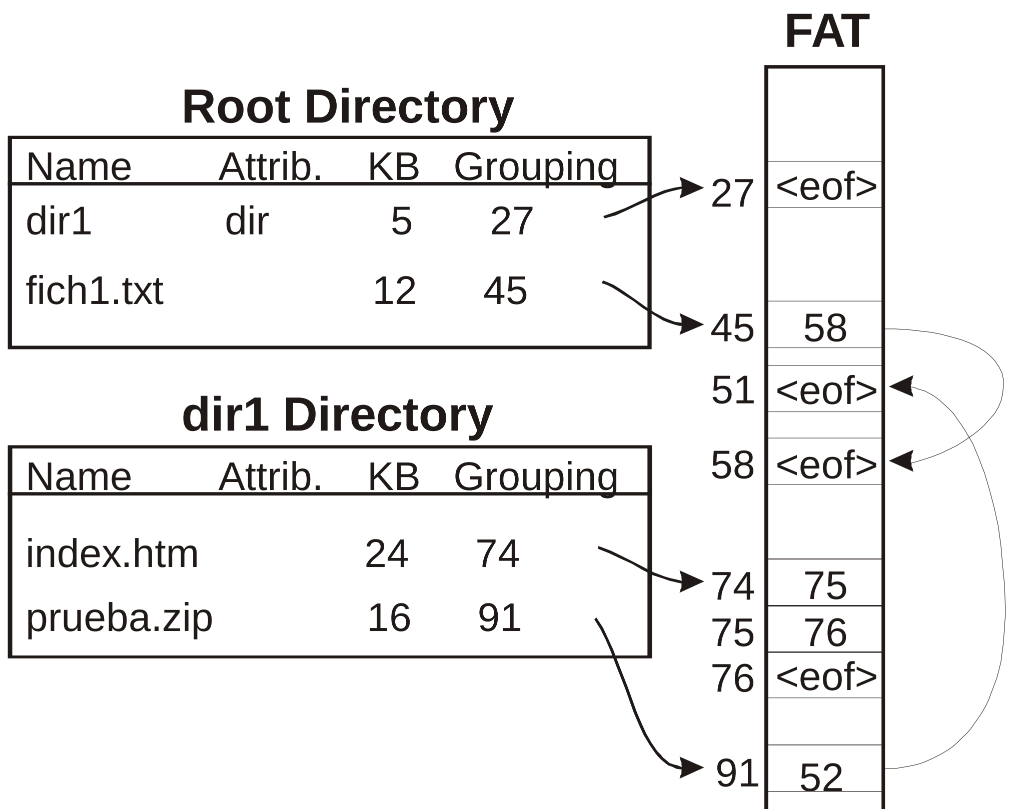
Name	Attrib.	KB	Grouping
index.htm		24	74
prueba.zip		16	91

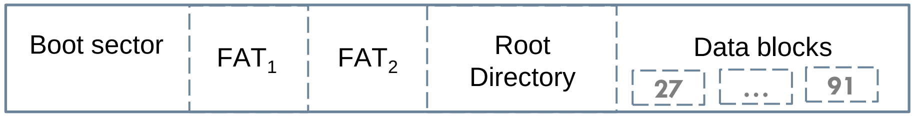


File System:

representation used in FAT: files

important!

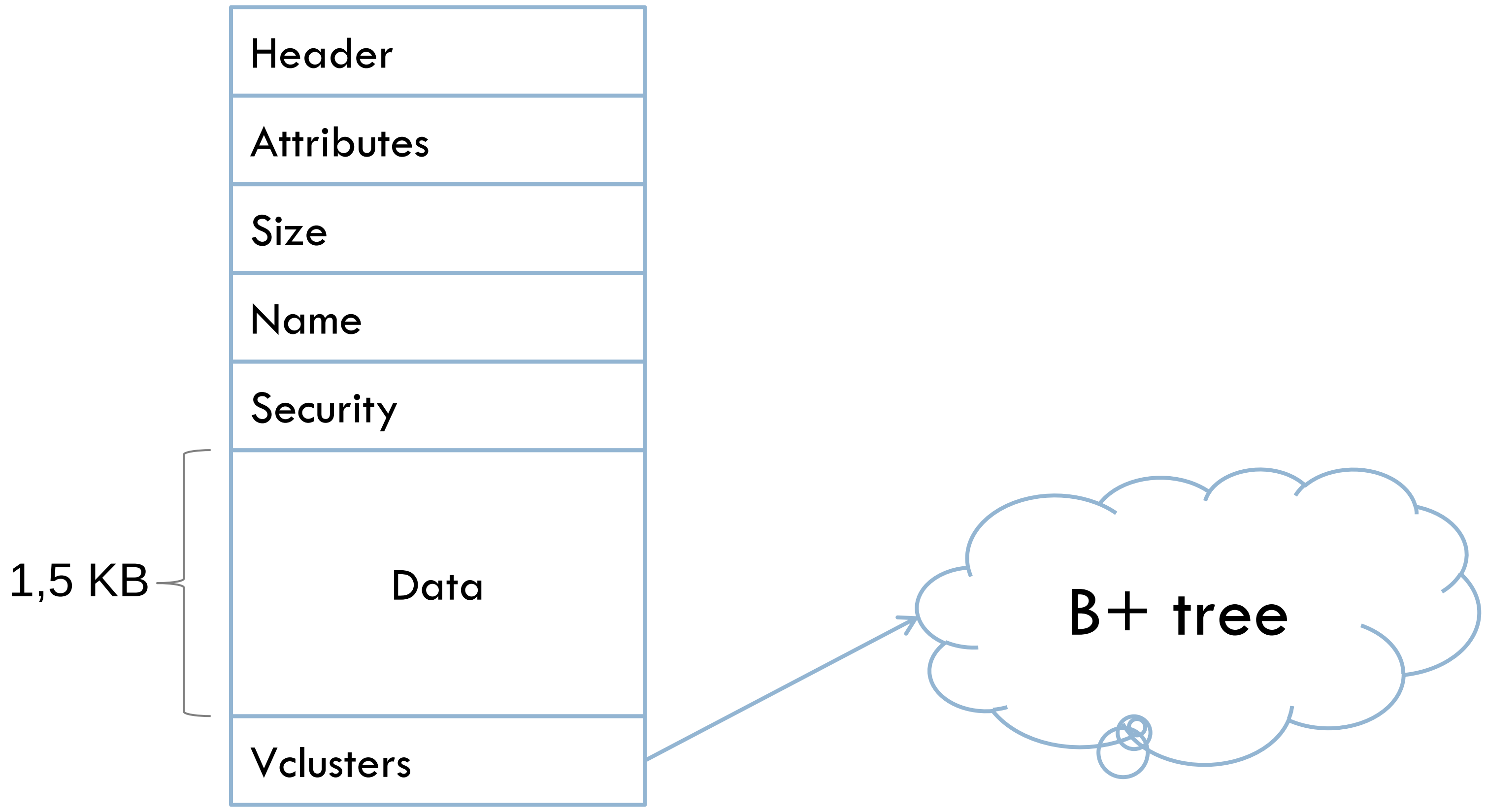




FAT 12 bits (2^{12} blocks)
FAT 16 bits (2^{16} blocks)
FAT 32 bits (2^{32} blocks)

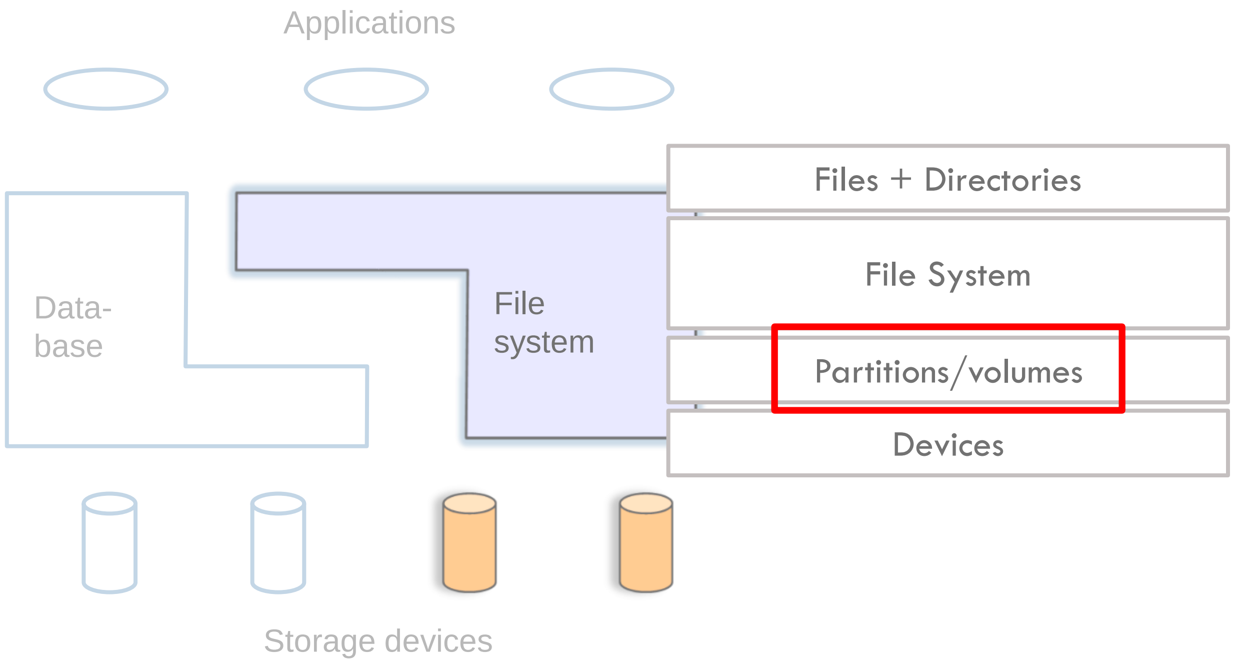
File System:

representation used on NTFS



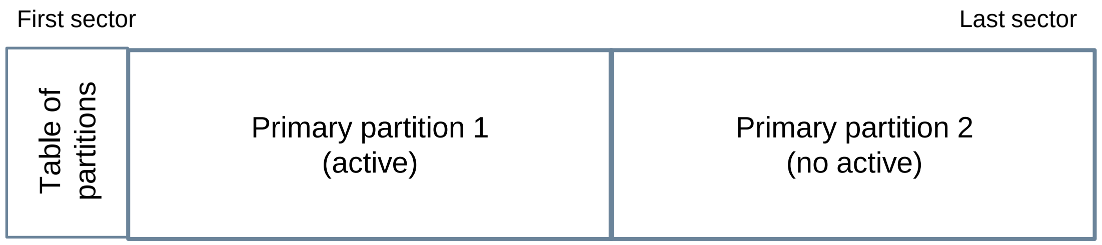
Contents

Partitions/Volumes



Partitions

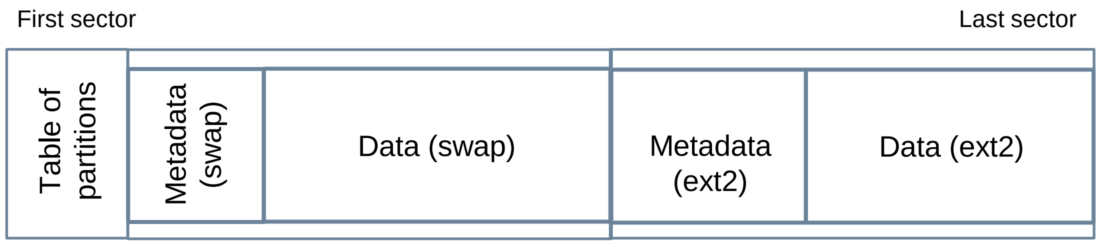
- Container of a file system.



- A **partition** is a *portion of a disk that is given an identity of its own and can be manipulated by the operating system as an independent logical entity*.
- Usually, the partition table is stored at the beginning of the device:
  - Each partition table entry stores the attributes of the associated partition.
  - A device can be divided into one or more partitions (the partition table lists all partitions).

Partitions

- Container of a file system.

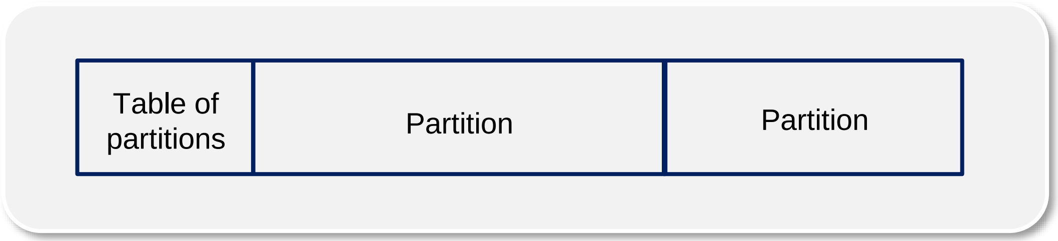


- Once the partitions are created, the operating system must create the data structures of the file systems within those partitions:
  - The boot sector in MS-DOS/DR-DOS
  - On the superblock in Unix
- Commands such as `format` or `mkfs` are provided to the user for this purpose:
  - # mkswap –c /dev/hda1 20800
  - # mkfs -c /dev/hda2 –b 8196 123100

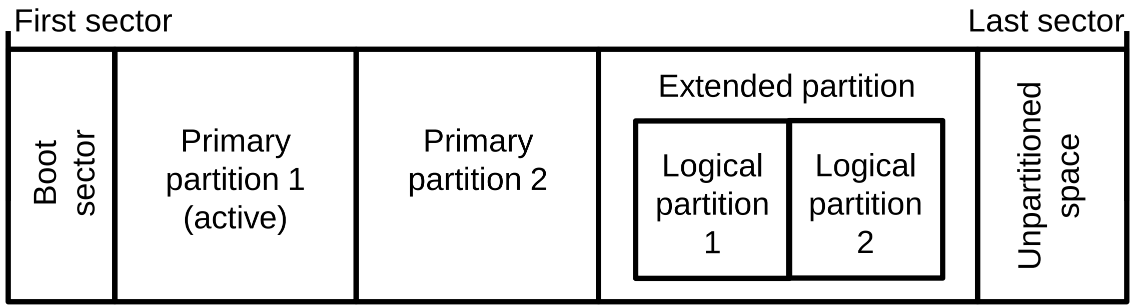
Partitions

important!

- Typical partition attributes:
  - Type: primary, secondary, logical drive, bootable, etc.
  - Size: start and end partition.
  - Hosted system: linux, linux swap, vfat, etc.
  - Identification: partition number (order or UUID).



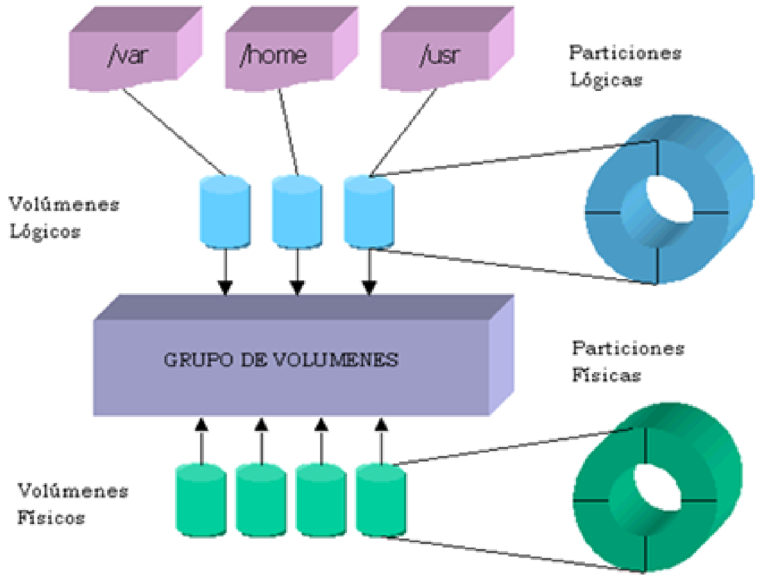
Partitions: traditional partitioning on PC



- Boot sector contains the partition table
- Primary or secondary partition (with logical drives)
- Old and limited:
  - 4 partitions in total (primary + secondary)
  - It is not possible to change the size without losing data

Volumes

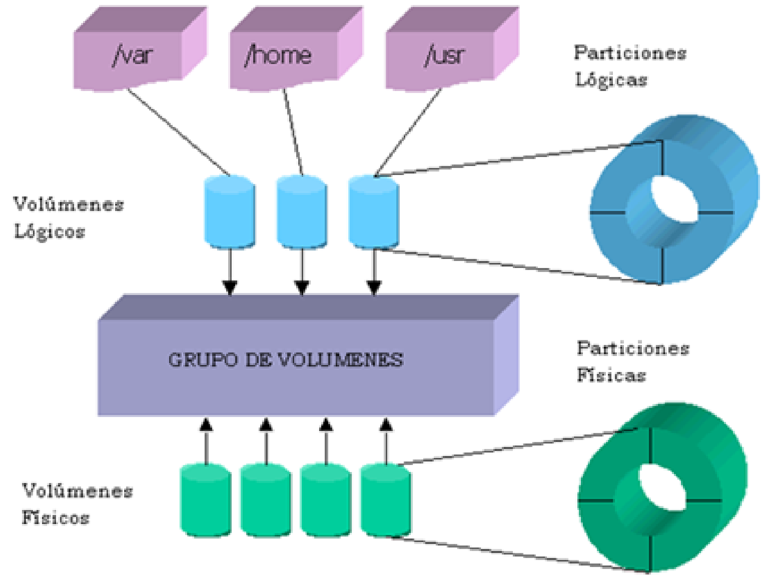
important!



http://www.howtoforge.com/linux_lvm

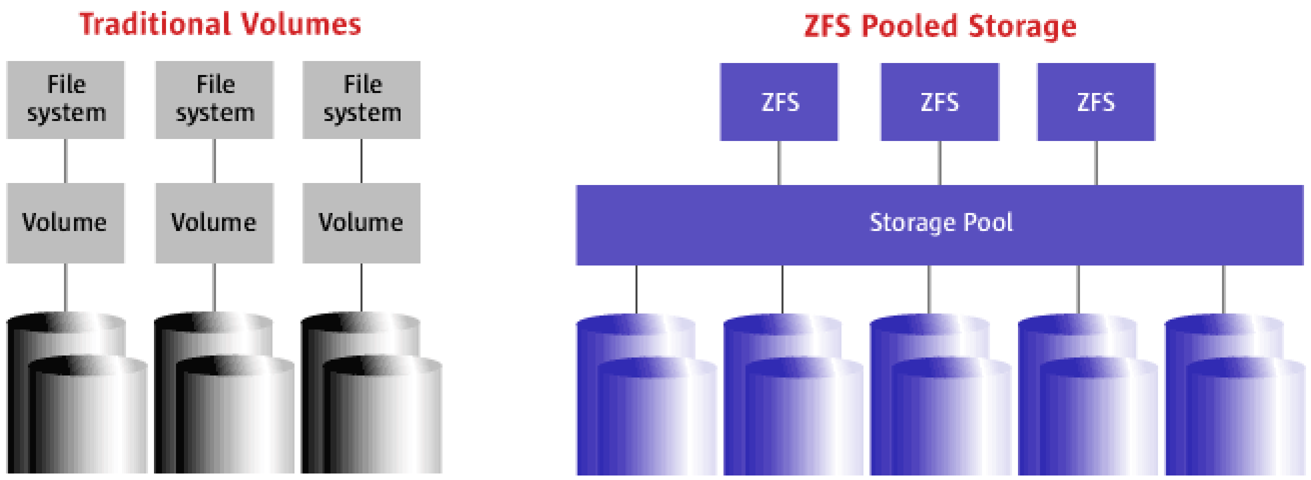
- Logical volumes, over volume group, composed of physical volumes.
  - Logical volume manifests itself similarly to the old partitions
- More modern and flexible:
  - Increased number (+ limit), dynamic change, use of multiple disks, etc.

Volumes



- Create a physical volume, a volume group and a logical volume group:
  - # pvcreate /dev/sdb1
  - # vgcreate vol_infoso /dev/sdb1
  - # lvcreate –L100M –nweb vol_infoso
  - # mkfs –t ext3 /dev/vol_infoso/web
  - # mount /dev/vol_infoso/web /mnt

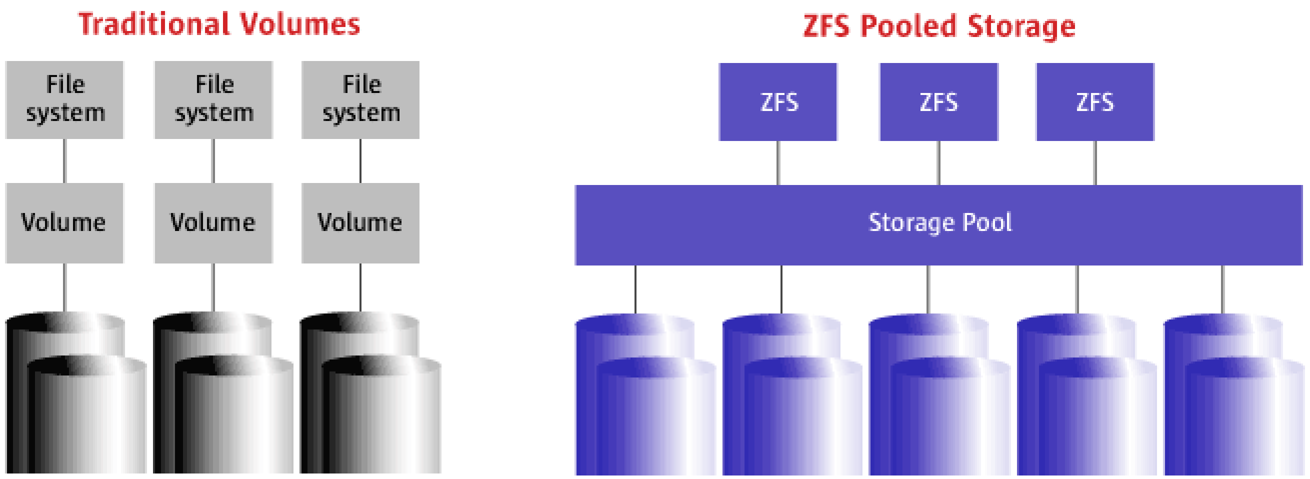
Storage pool (ZFS)



http://hub.opensolaris.org/bin/download/Community+Group+zfs/docs/zfslast.pdf

- Simplification in the use of devices, volumes and file systems through their integration.
- File systems are created on a storage pool composed of physical devices (or parts of them)

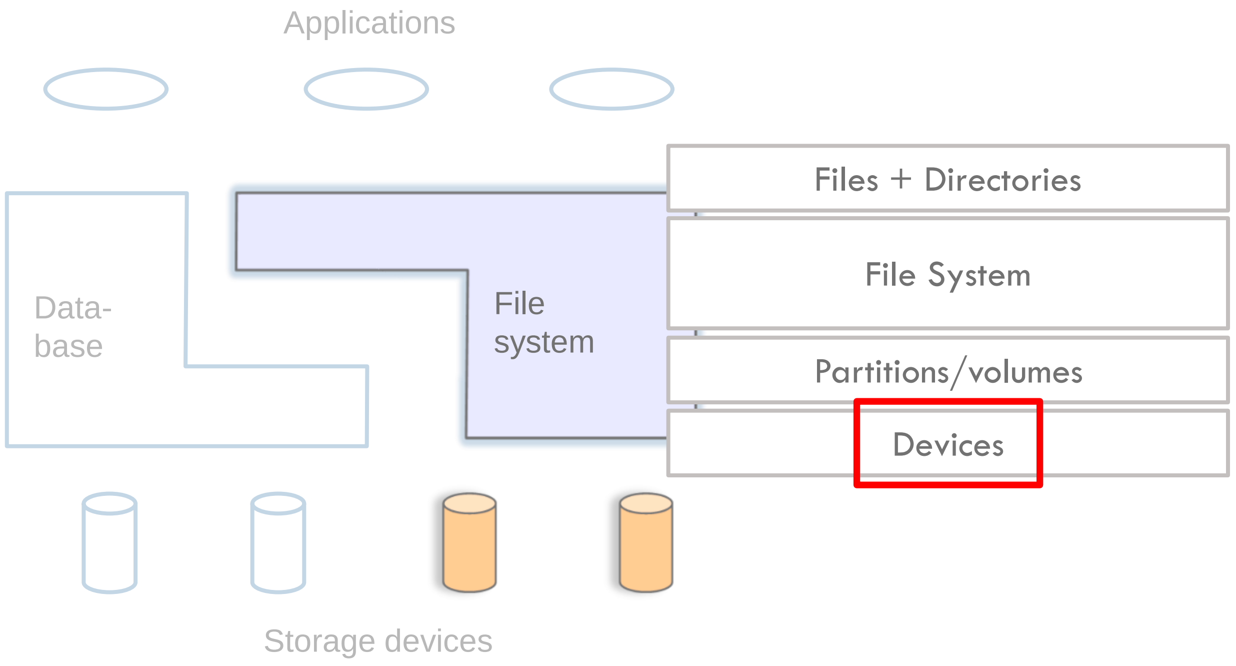
Storage pool (ZFS)



- Creating the pool, a file system and setting options:
  - # zpool create infoso /dev/disk1
  - # zfs create infoso/practicas
  - # zfs set mountpoint=/export/practicas/infoso infoso/practicas
  - # zfs set quota=10g infoso/practicas

Contents

Devices


Applications
Data-
base
File
system
Files + Directories
File System
Partitions/volumes
Devices
Storage devices

Devices

real devices

- Hard disk

- SSD (solid state)

- Optical systems

- Etc.

Devices

real devices

- List the PCI devices:

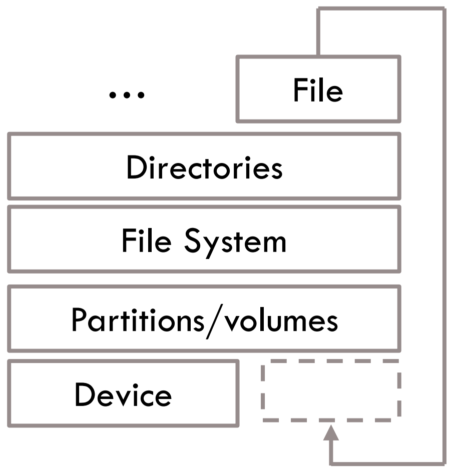
```
acaldero@phoenix:~/infodso/$ lspci
00:00.0 Host bridge: Intel Corporation 82Q35 Express DRAM Controller (rev 02)
00:01.0 PCI bridge: Intel Corporation 82Q35 Express PCI Express Root Port (rev 02)
00:03.0 Communication controller: Intel Corporation 82Q35 Express MEI Controller (rev 02)
00:03.2 IDE interface: Intel Corporation 82Q35 Express PT IDER Controller (rev 02)
00:03.3 Serial controller: Intel Corporation 82Q35 Express Serial KT Controller (rev 02)
...
```

- List the USB devices:

```
acaldero@phoenix:~/infodso/$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
...
Bus 008 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 002: ID 1241:1166 Belkin MI-2150 Trust Mouse
Bus 005 Device 002: ID 0c45:600d Microdia TwinkleCam USB camera
```

Devices

special

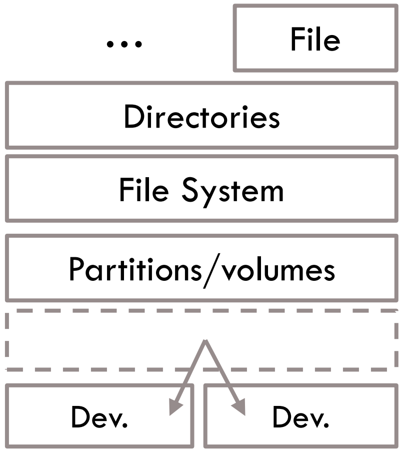


- *loopback* device
  - File as a block device

- Example of a working session:
  1. [1] Use a CD-ROM/DVD image:
     ```
     wget ftp://ftp.rediris.es/sites/releases.ubuntu.com/releases/21.04/ubuntu-21.04-desktop-i386.iso
     ```
  2. Associate the file to the loopback device:
     ```
     sudo losetup /dev/loop1 ubuntu-21.04-desktop-i386.iso
     ```
  3. Mount as a block device (disk):
     ```
     mount  /dev/loop1 /mnt
     ```
  4. Using the /mnt file system
  5. Dismount the device:
     ```
     umount /dev/loop1
     ```
  6. Disassociate the device:
     ```
     losetup -d /dev/loop1
     ```

Devices

special



- *md* device
  - Device of devices

- Example of a working session:
  1. [1] Create the md mirror device:
     ```
     mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/loop1 /dev/loop2
     ```
  2. [1] Create the file system:
     ```
     mkfs -t ext3 /dev/md0
     ```
  3. Mount and unmout the device:
     ```
     mount  /dev/md0 /mnt
     umount /dev/md0
     ```
  4. Stop the md device:
     ```
     mdadm --stop /dev/md0
     ```
  5. Start the md device:
     ```
     mdadm --assemble /dev/md0  /dev/loop1 /dev/loop2
     ```

OPERATING SYSTEMS: FILE SYSTEMS

Files, directories and file system