OPERATING SYSTEMS: FILE SYSTEMS

Files, directories and file system

To remember…

Alejandro Calderón Mateos

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

Alejandro Calderón Mateos

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 y 13

Contents

Alejandro Calderón Mateos

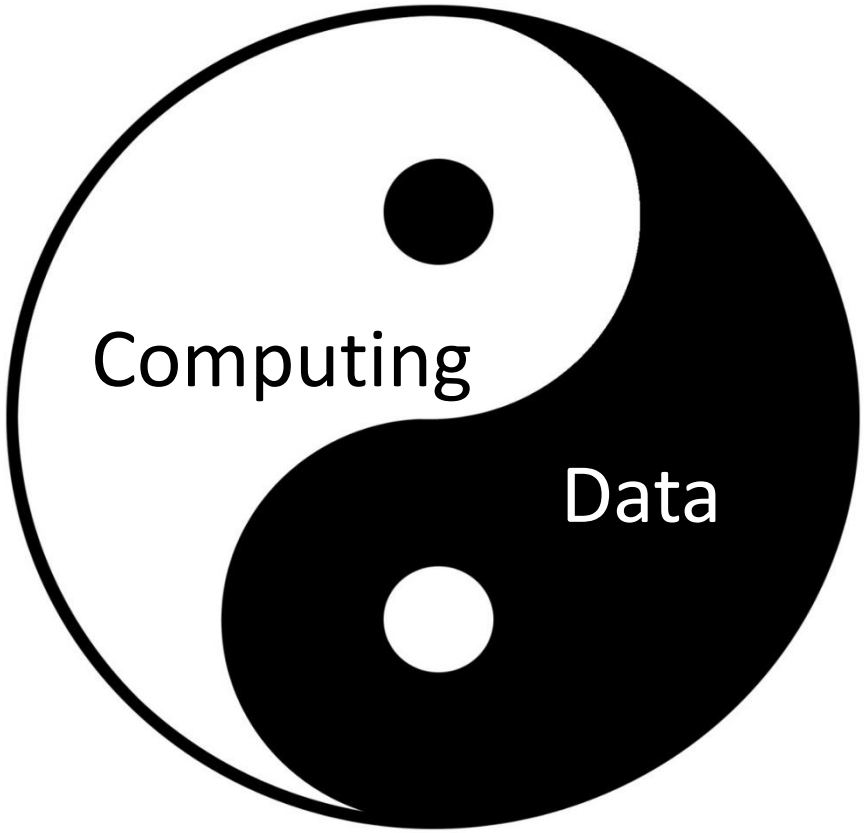
Contents

Alejandro Calderón Mateos

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

General scope

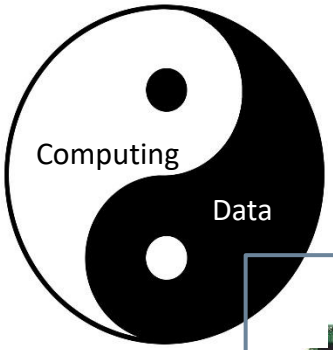
~2021

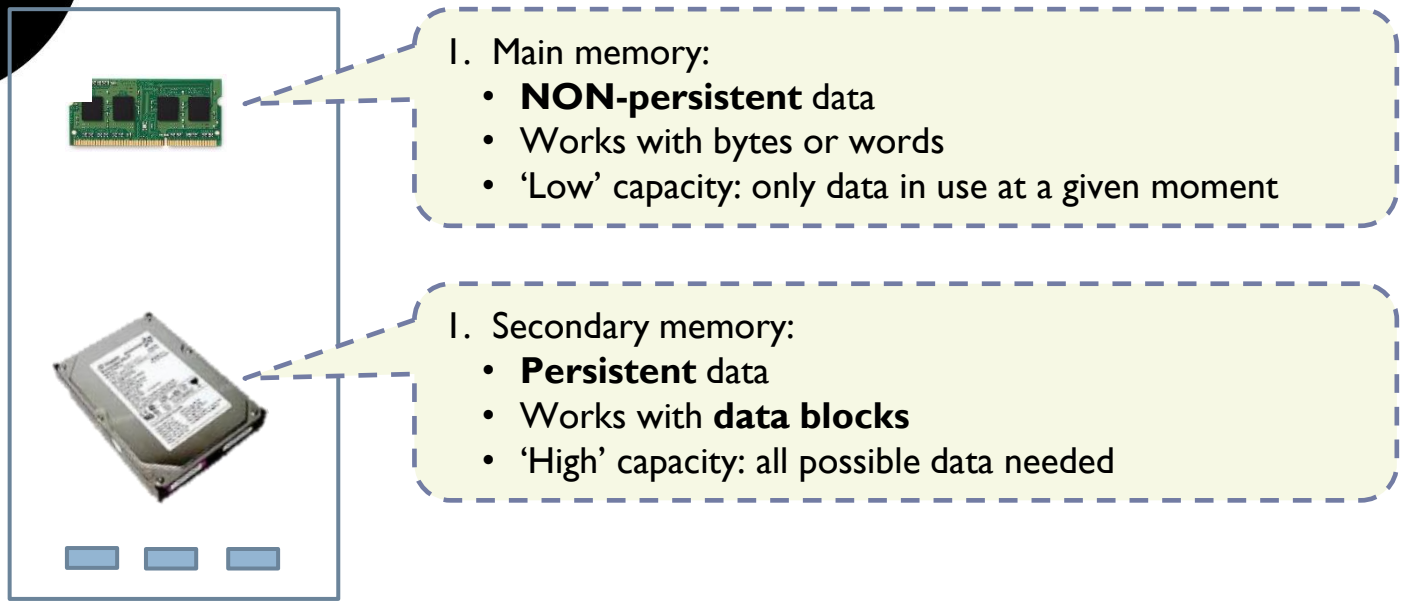


General scope

~2021

Alejandro Calderón Mateos





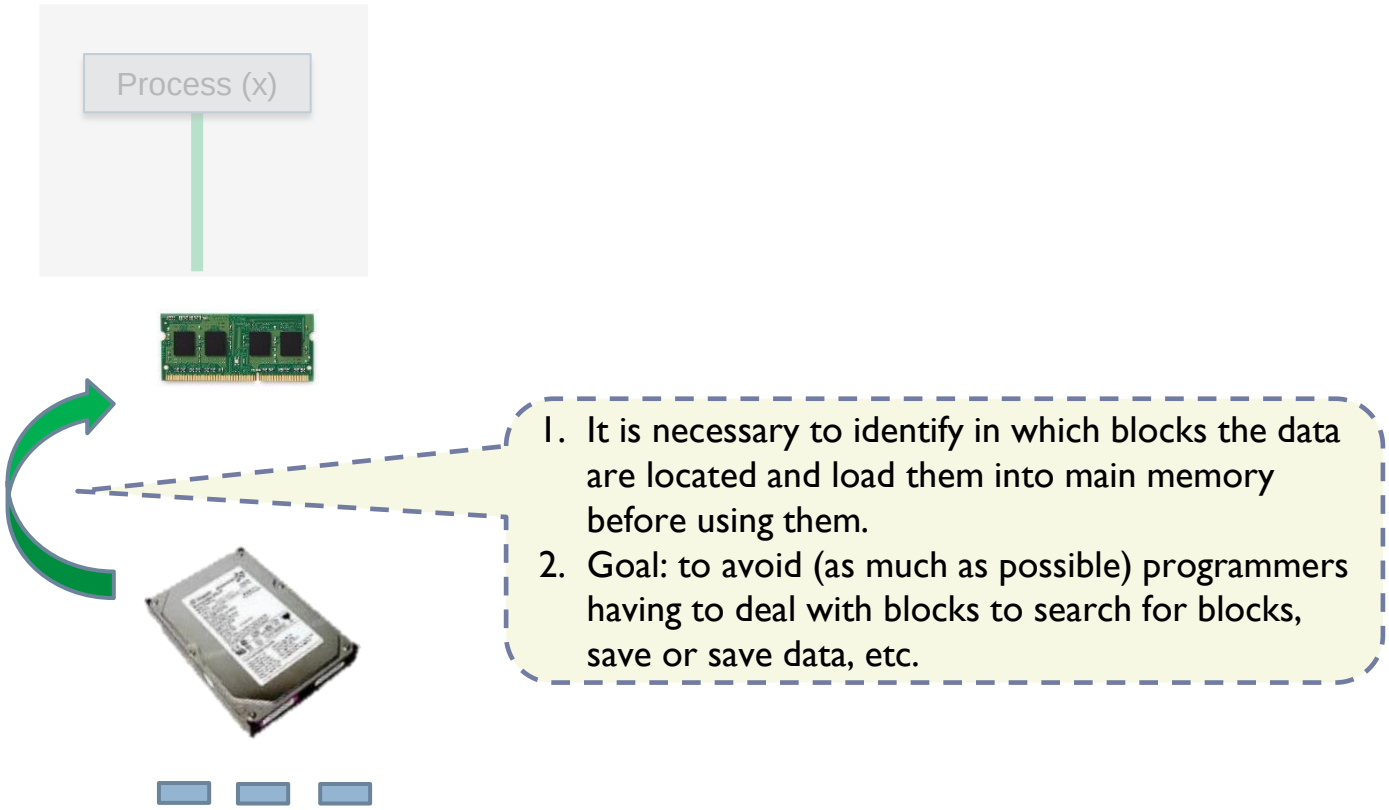
1. Main memory:
   - **NON-persistent** data
   - Works with bytes or words
   - 'Low' capacity: only data in use at a given moment

1. Secondary memory:
   - **Persistent** data
   - Works with **data blocks**
   - 'High' capacity: all possible data needed

General scope

~2021

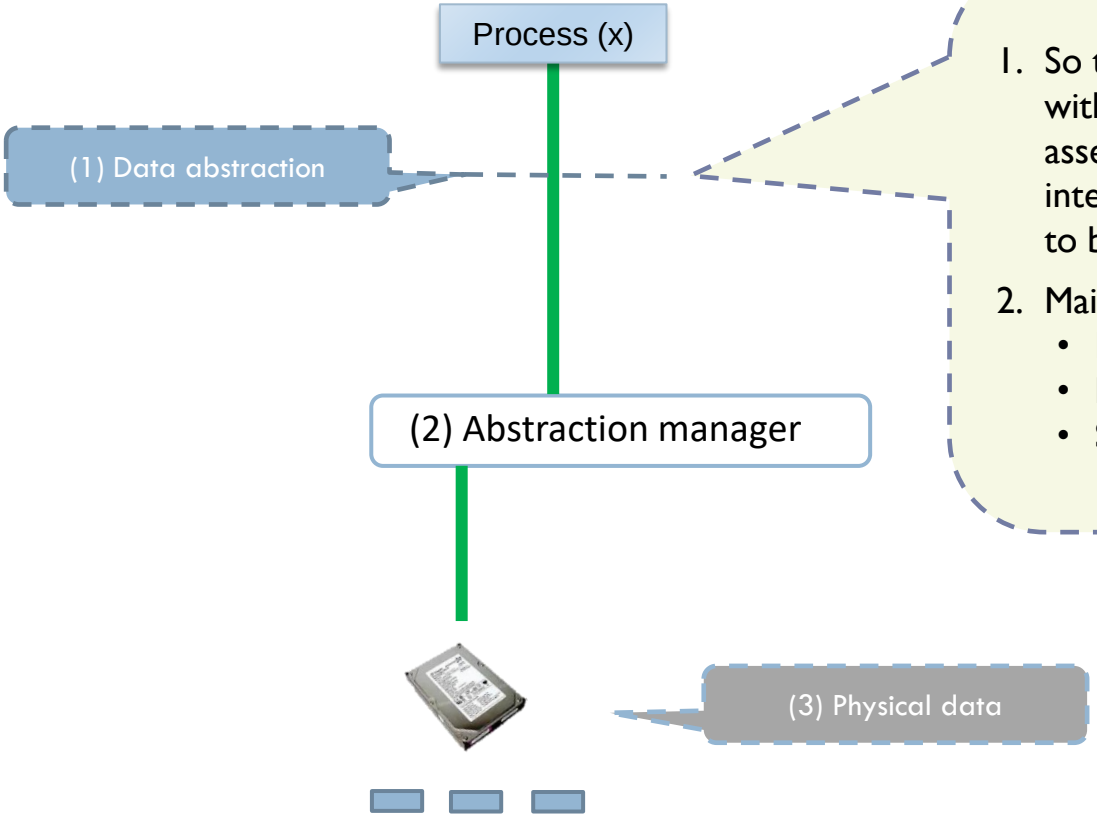
Alejandro Calderón Mateos



General scope

~2021

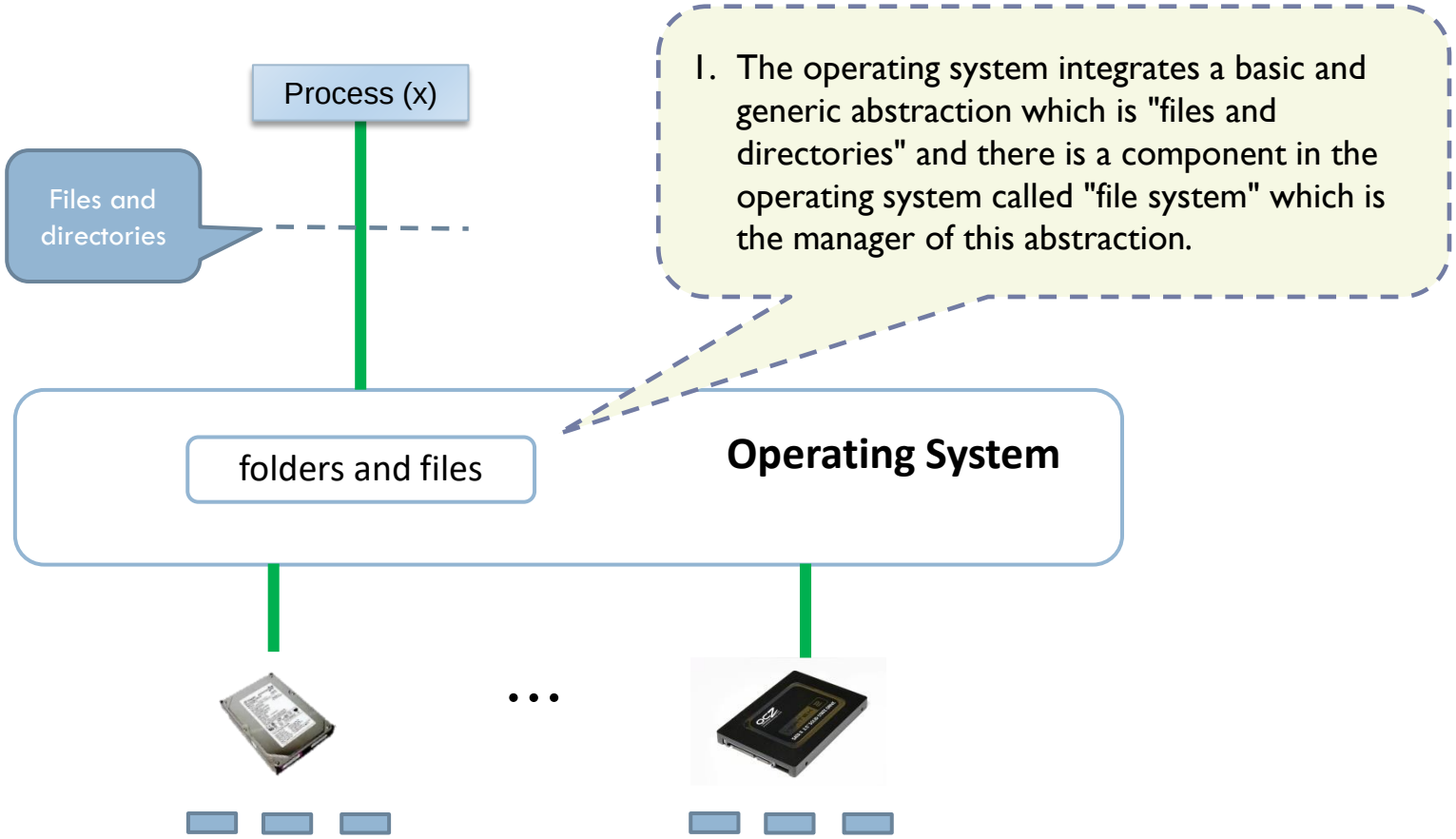
Alejandro Calderón Mateos



1. So that programmers do not have to deal with the problems of dealing with blocks, assembly, etc. it is proposed to use an intermediate data abstraction that translates to blocks.
2. Main objectives of using such abstraction:
   - Be independent of the physical device.
   - Provide a unified logical view.
   - Sufficiently simple but complete.

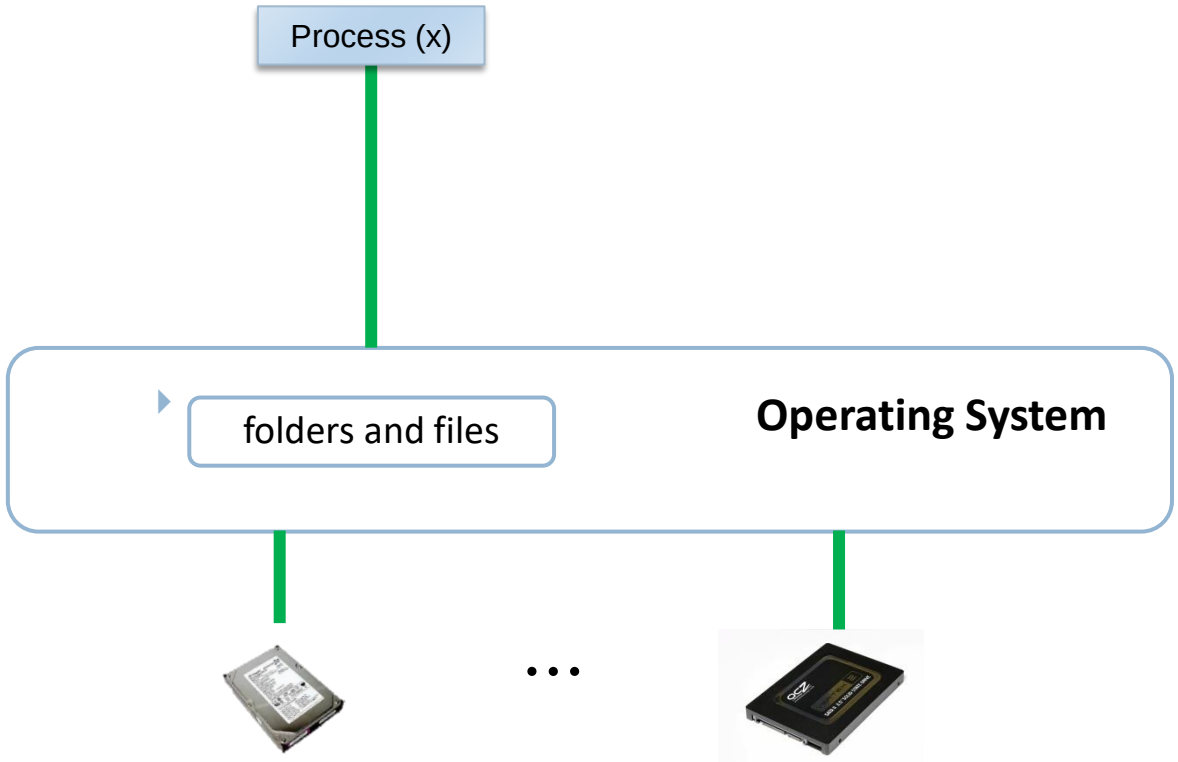
(1/2) The O.S. integrates a basic and generic abstraction: file system

Alejandro Calderón Mateos



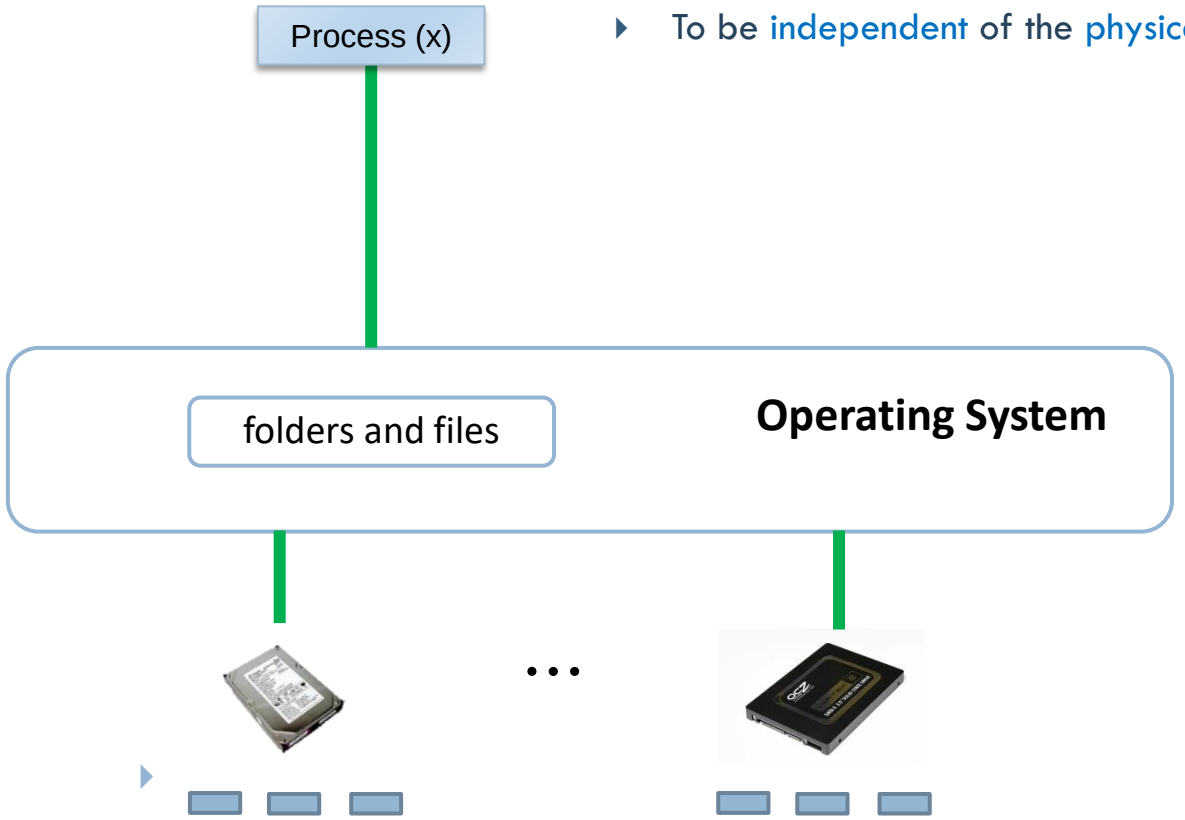
Main features of a file system

- Added functionality to facilitate secondary storage management:

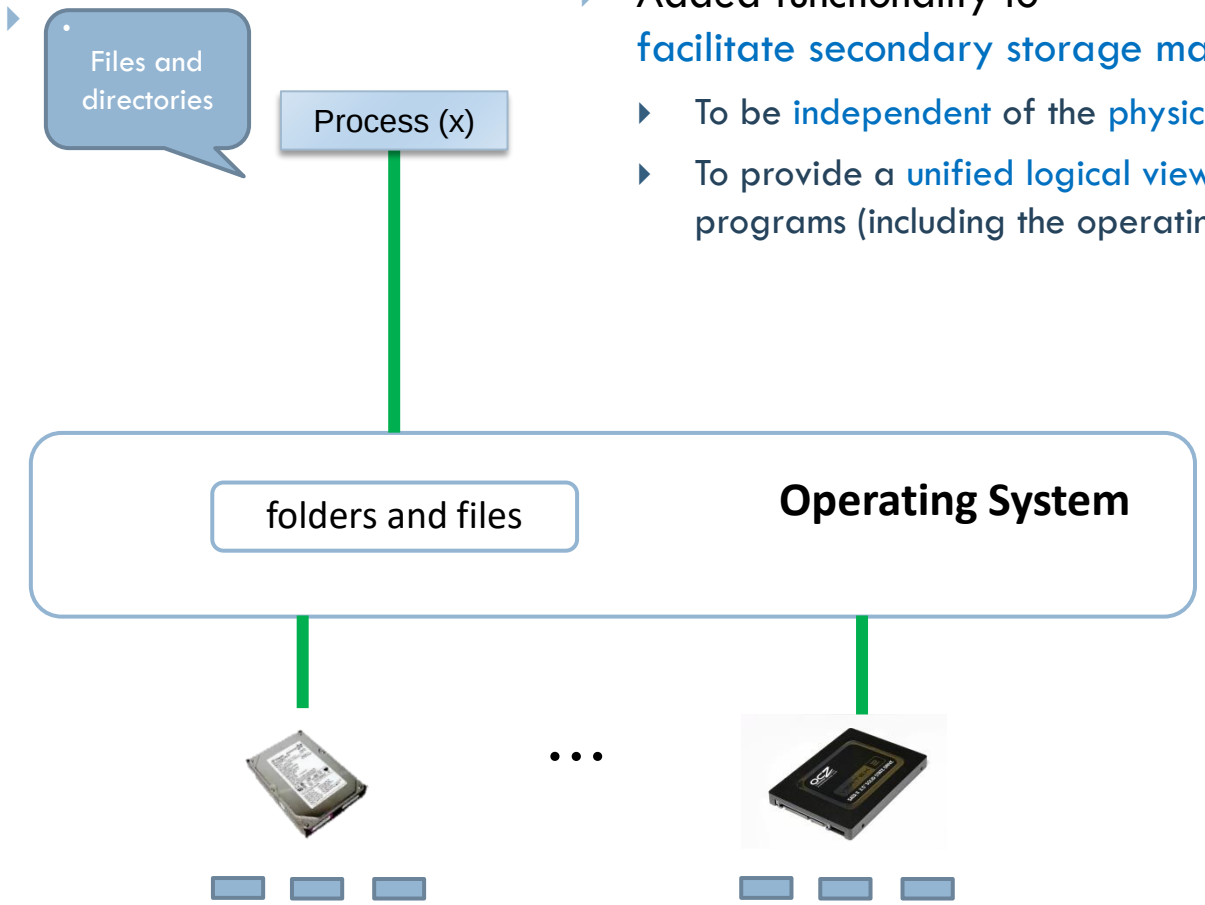


Main features of a file system

- Added functionality to facilitate secondary storage management:
  - To be independent of the physical device.

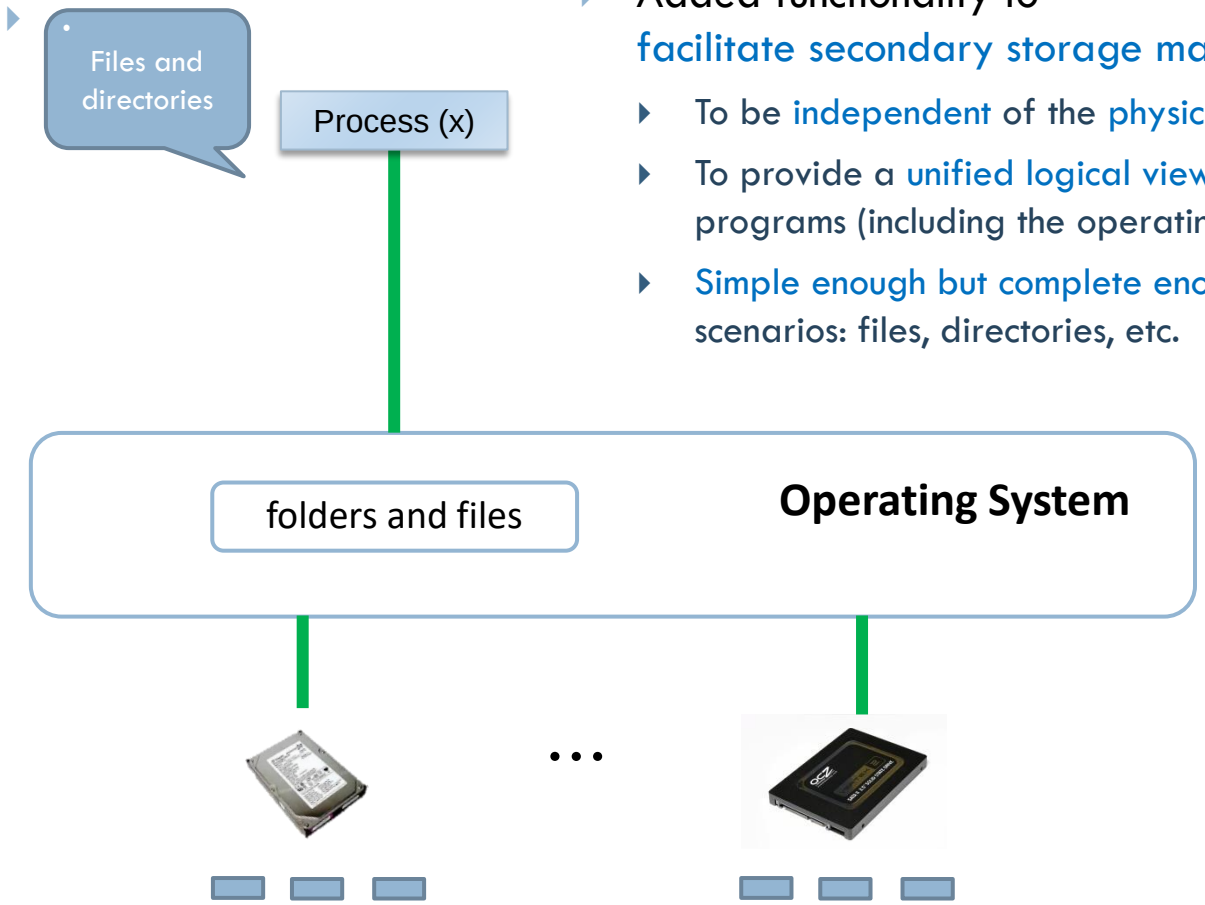


Main features of a file system



- Added functionality to facilitate secondary storage management:
  - To be independent of the physical device.
  - To provide a unified logical view for users and programs (including the operating system itself).

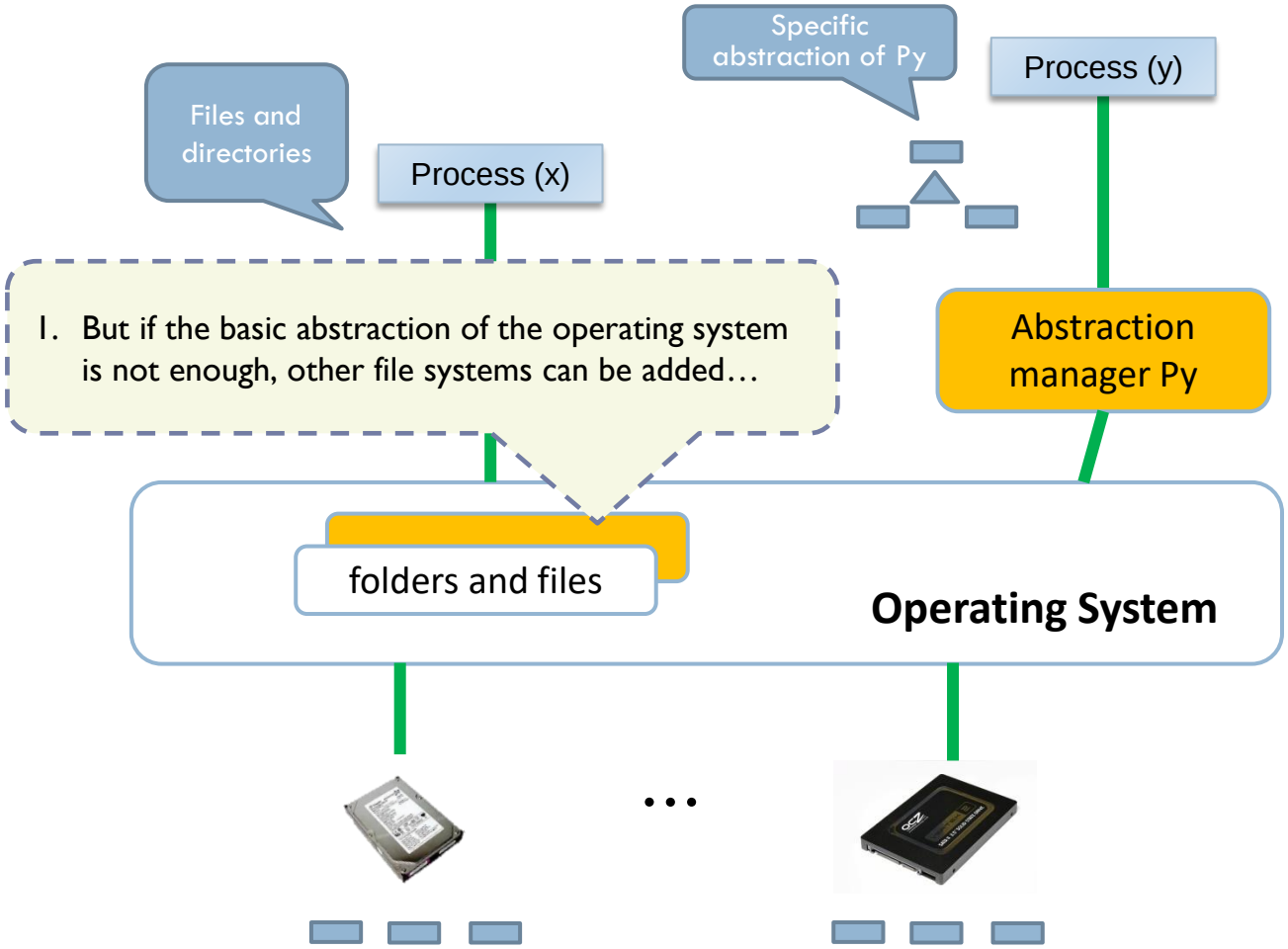
Main features of a file system



- Added functionality to facilitate secondary storage management:
  - To be independent of the physical device.
  - To provide a unified logical view for users and programs (including the operating system itself).
  - Simple enough but complete enough to cover most scenarios: files, directories, etc.

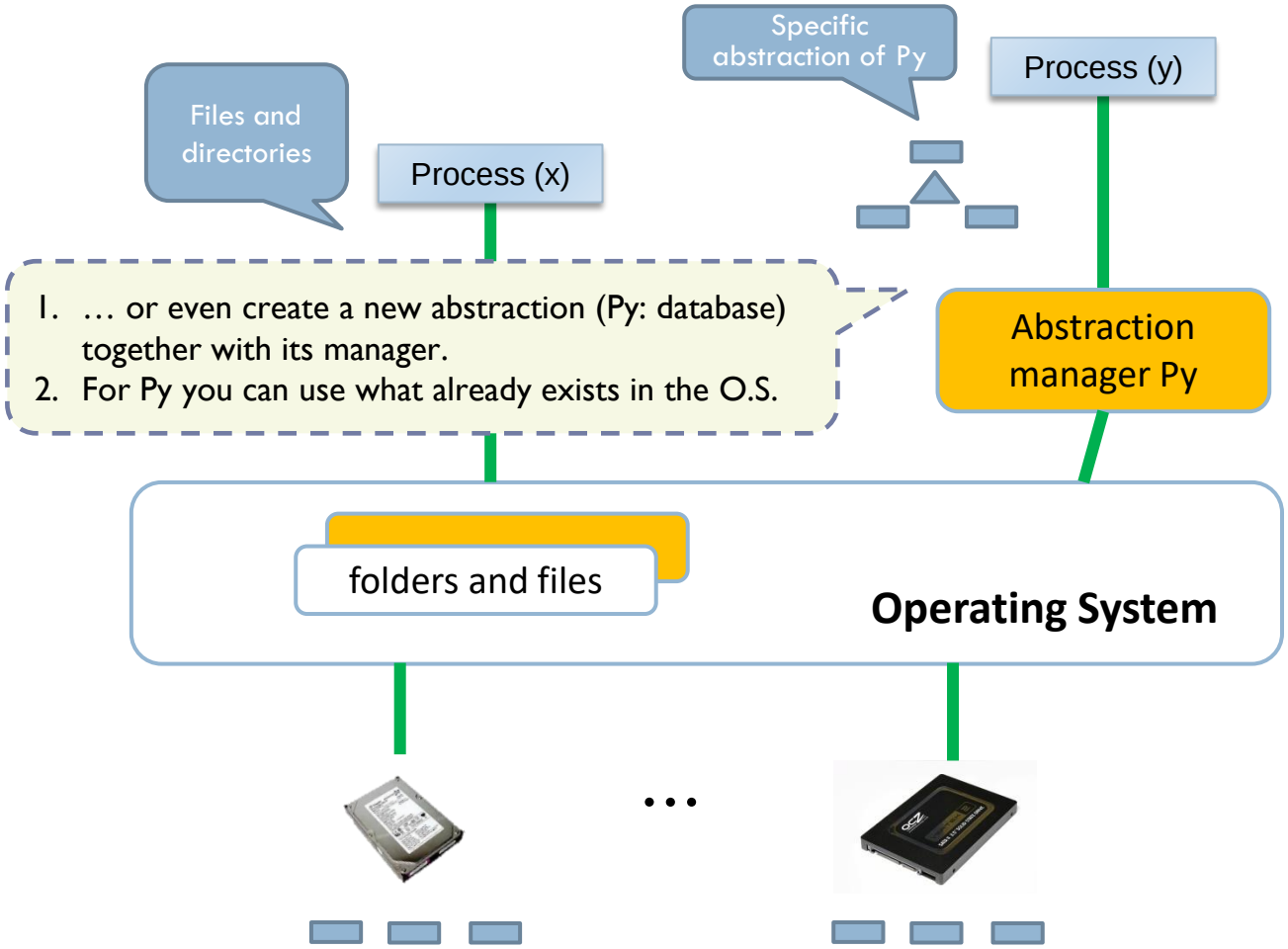
(2/2) The O.S. provides support for building even other storage systems

Alejandro Calderón Mateos



(2/2) The O.S. provides support for building even other storage systems

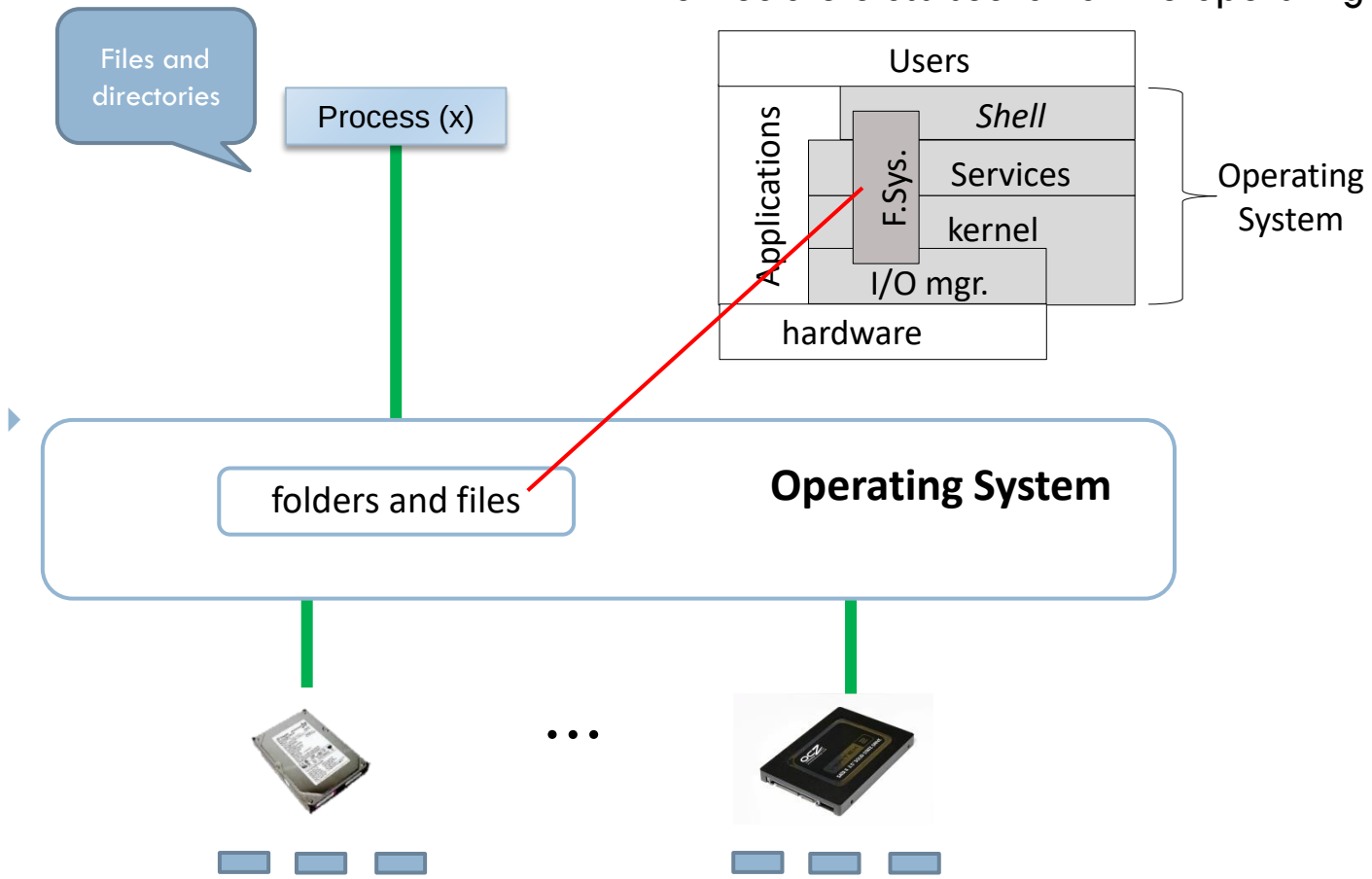
Alejandro Calderón Mateos



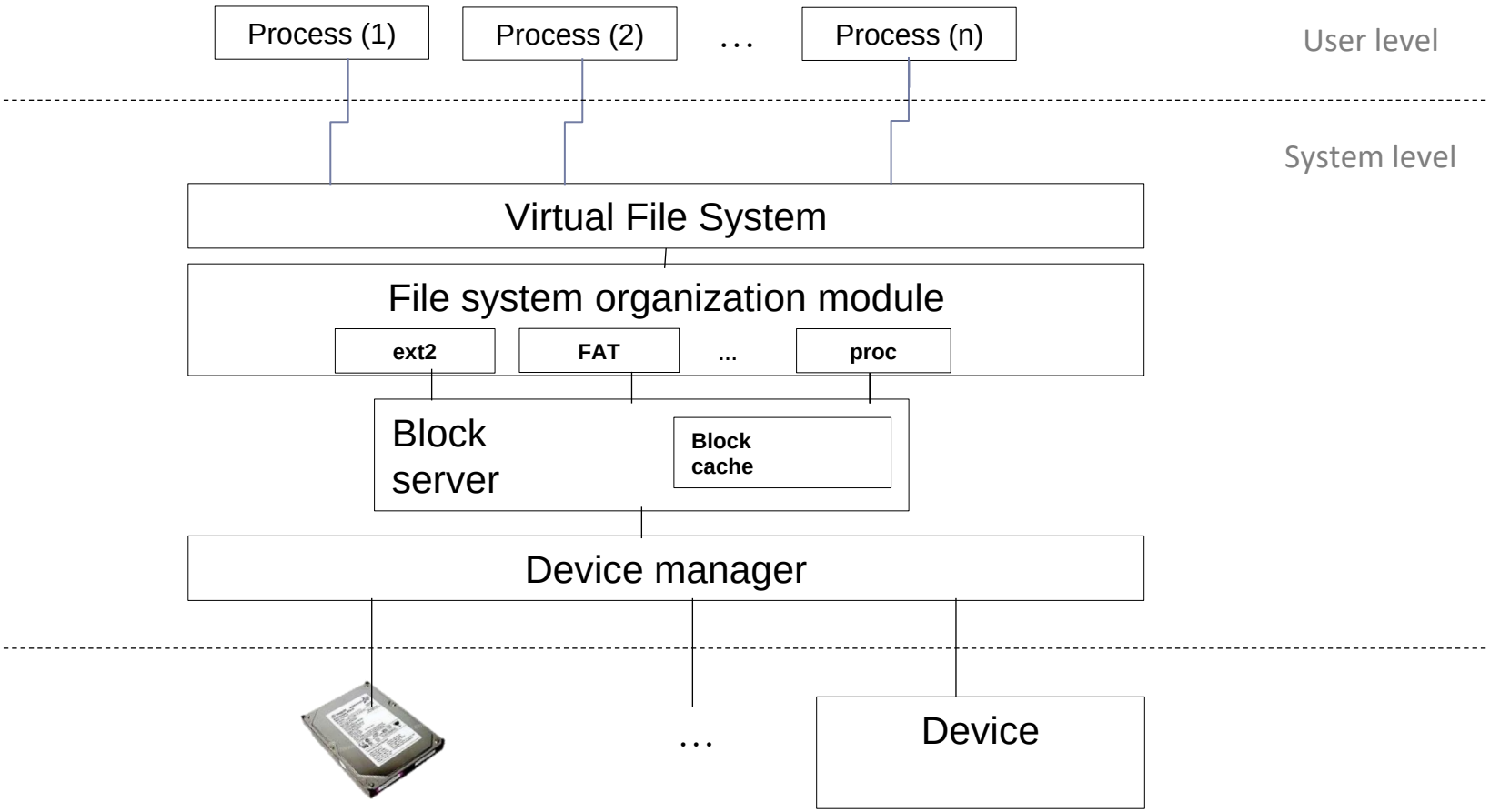
Architecture of file systems

Alejandro Calderón Mateos

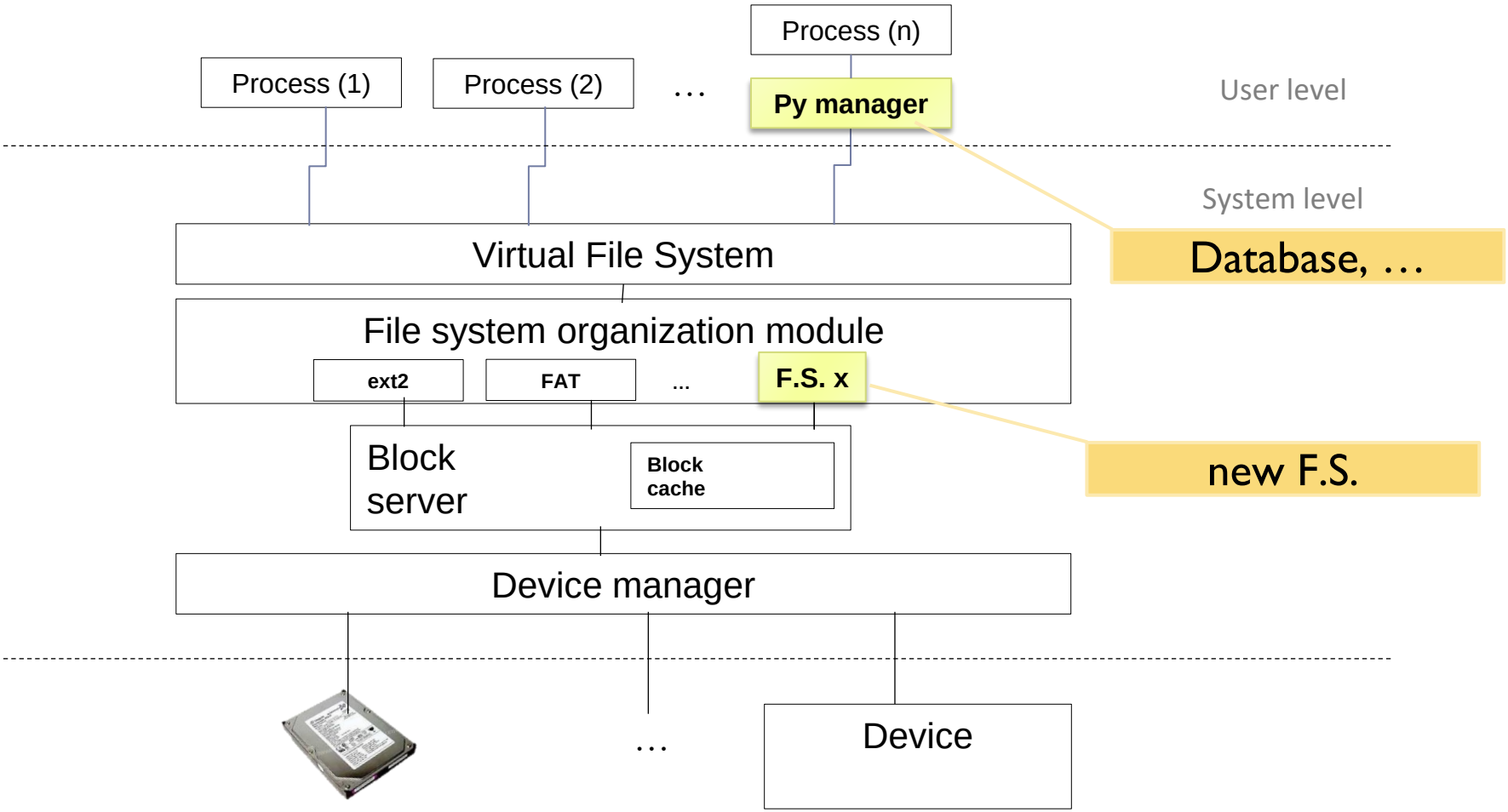
- Architecture cross-section of the operating system:



Architecture of file systems



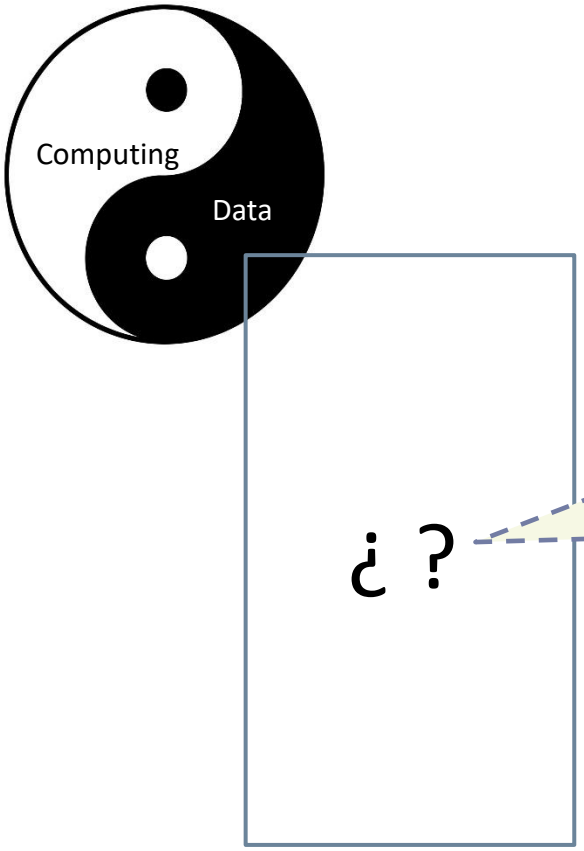
Extensible architecture
with external file systems and file managers



General scope

> 2021

Alejandro Calderón Mateos



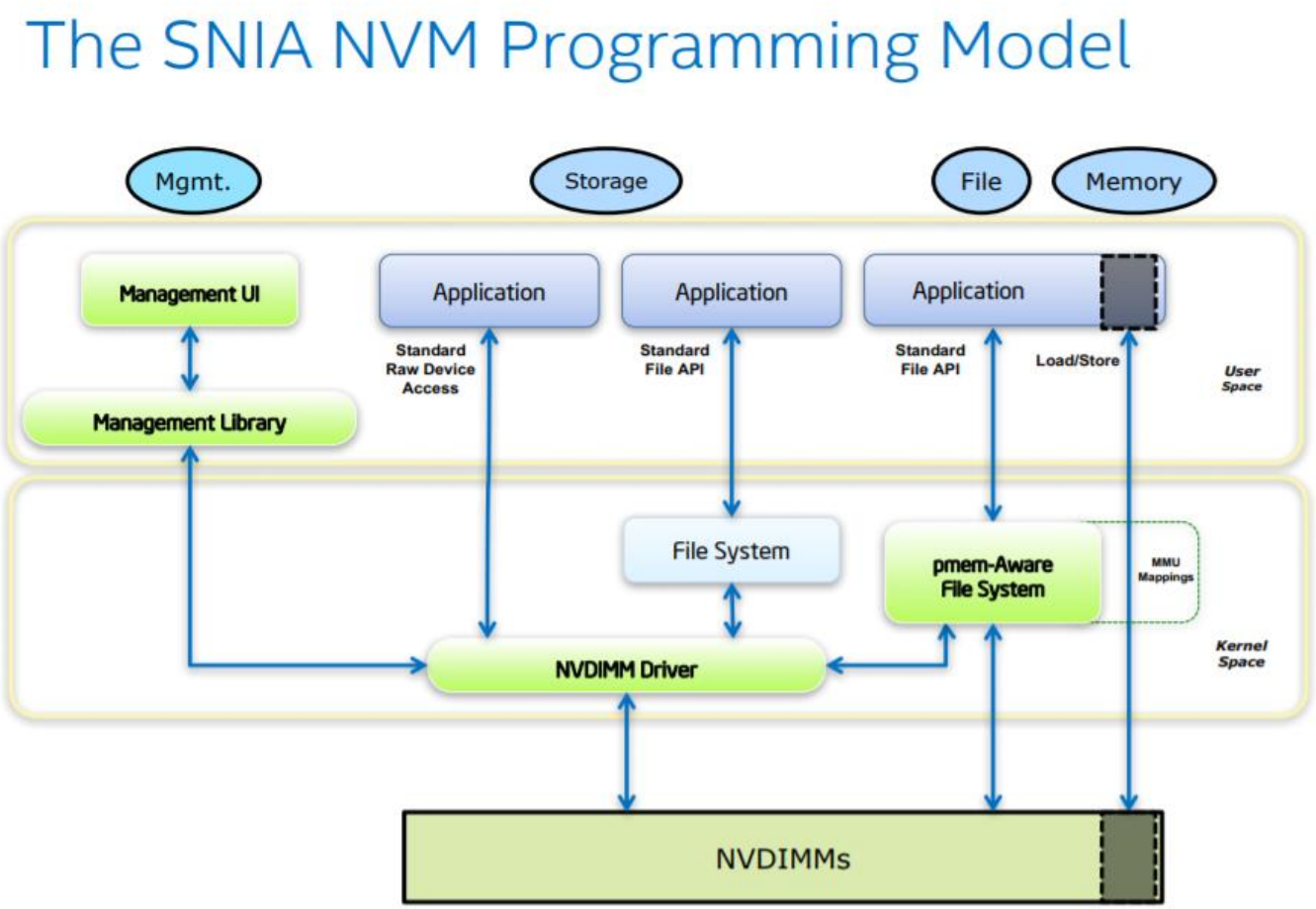
1. New memory (sum of main and secondary memory):
   - **Persistent** data
   - Works with bytes or words, or with blocks
   - 'High' capacity

General scope

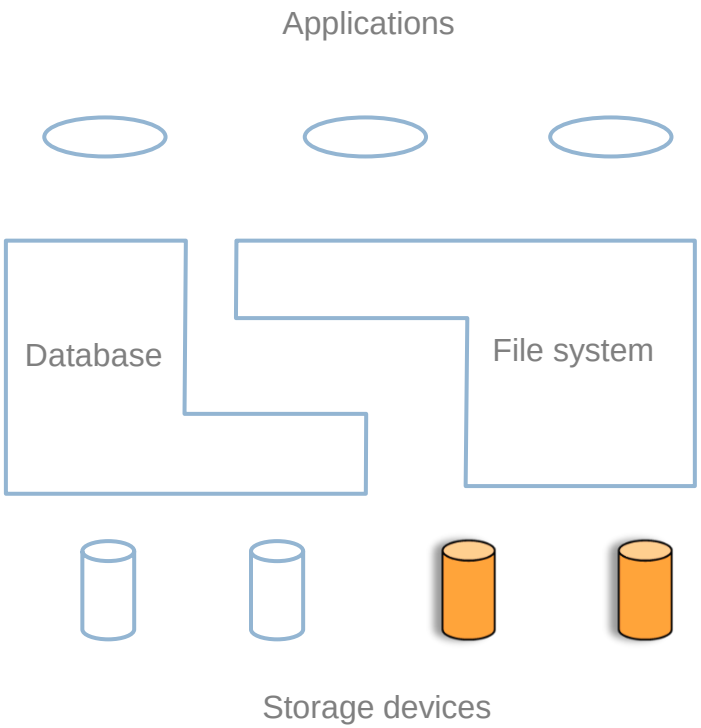
> 2021

https://www.snia.org/tech_activities/standards/curr_standards/npm, http://pmem.io/

Alejandro Calderón Mateos

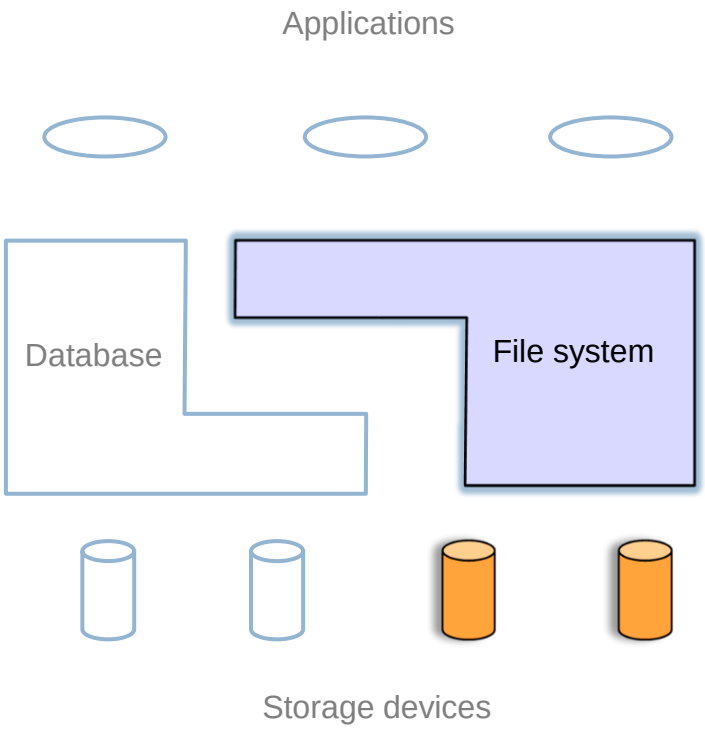


Summary: architecture



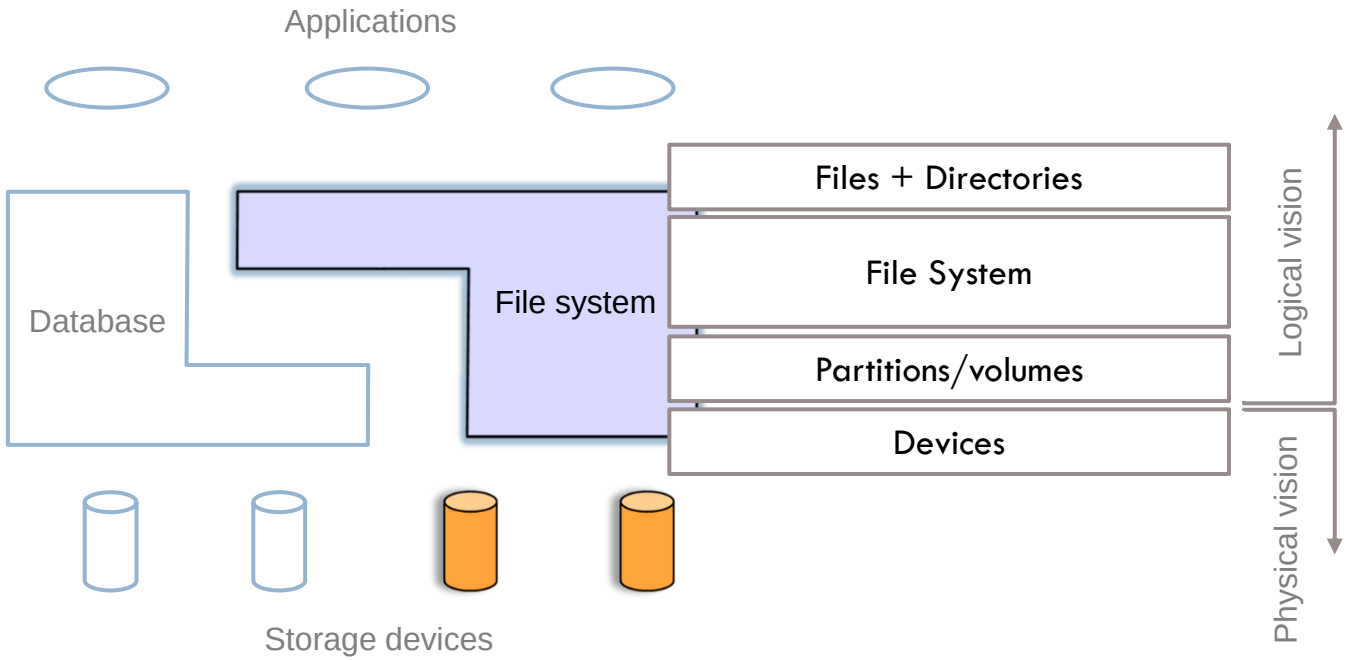
- We have both possibilities in illustration proposed by SNIA:
  - *Storage Networking Industry Association*
  - http://www.snia.org
- Applications access data stored on storage devices using DB and/or file systems.

Summary: architecture



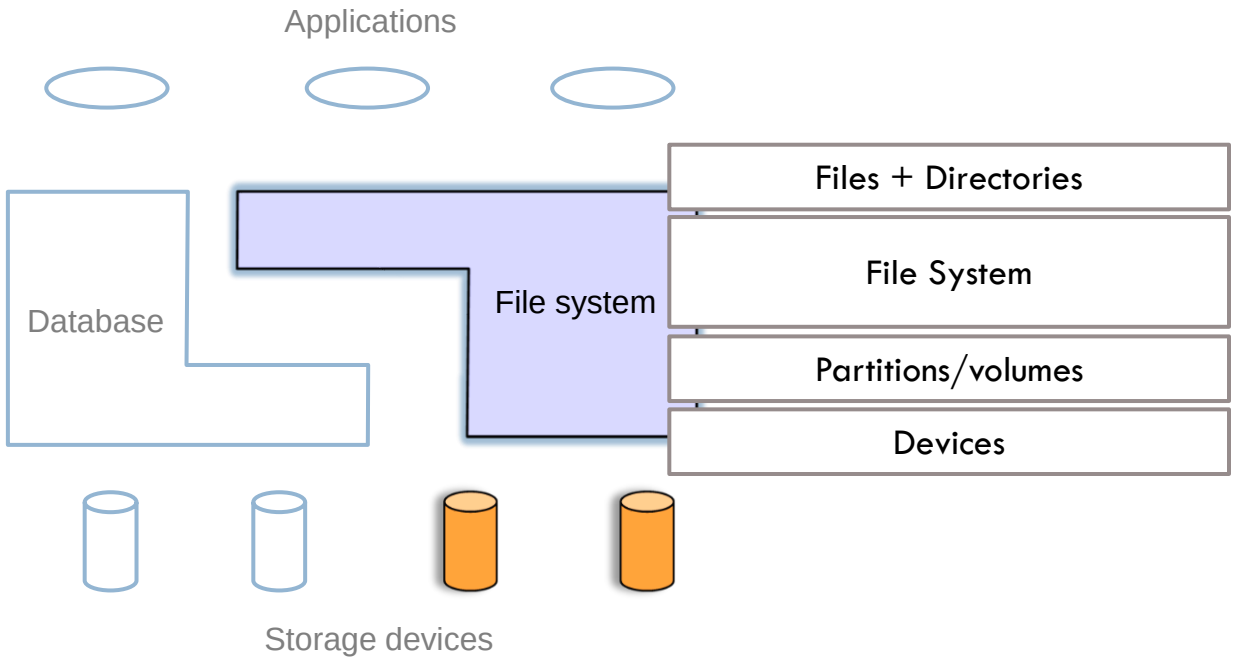
- In this topic we will focus on the management by the O.S. through the file system:
  - Organization
  - Storage
  - Retrieval
  - Name management
  - Implementation of co-utilization semantics
  - Protection

Summary: abstractions



- To be studied: files, directories, file systems, volumes and devices
  - Logical view
  - Physical vision

Summary: abstractions



- Beware of the term "file system" which is used to name both the management software and the data structures on disk (context is important).

Introduction
summary

important

- File System:
  - It is the part of the OS in charge of distributing and organizing the S.M.
  - It provides an abstraction (based on files, directories, etc.) that hides the details of the M.S. organization.
    - Hides details about data storage/distribution on peripherals.
  - Main functions:
    - (1) Organization, (2) Names Management, (3) Storage, (4) Retrieval, (5) Implementation of co-utilization semantics, (6) Protection.
- File System also:
  - It is the software layer between devices and users.
  - Simplifies handling of peripherals by treating them as files
    - Establishes a correspondence between logical devices and files.
    - Facilitates protection and logical vision (as independent of physical details).

Introduction
summary

important

- File System from the user point of view:
  - Permanent storage of information:
    - It does not disappear even if the computer is turned off.
  - Logical abstraction to facilitate information handling:
    - A set of information logically structured according to application criteria.
    - Logical and structured names.
    - They are not tied to the life cycle of a particular application.
    - Abstracting physical storage devices.
  - Access to services offered through an API:
    - They are accessed through operating system calls or utility libraries.
  - It is possible to work with several file systems at the same time in an O.S.:
    - Example: Linux admits at the same time ext4, btfs, fat32, etc.

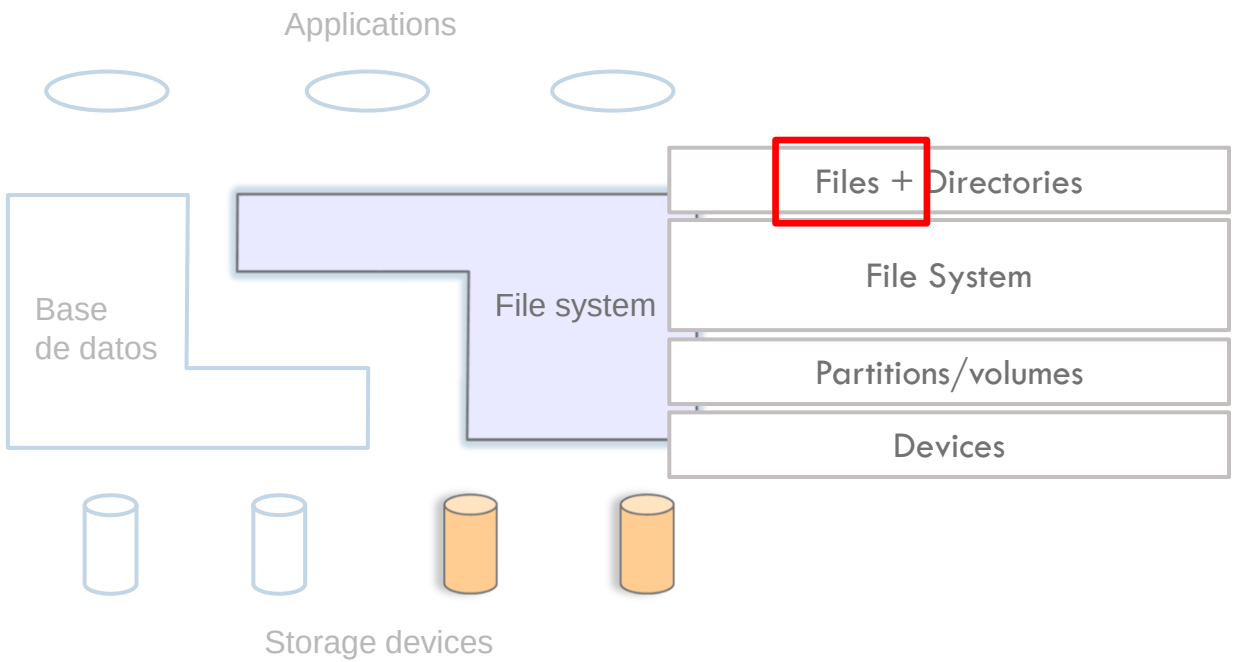
Contents

Alejandro Calderón Mateos

- Introduction
- **File**
  - Metadata
  - Interface
  - Access methods
  - Sharing semantics
- Directory
- File System
- Partitions/Volumes
- Devices
- System software
- File System (manager)

File

Alejandro Calderón Mateos

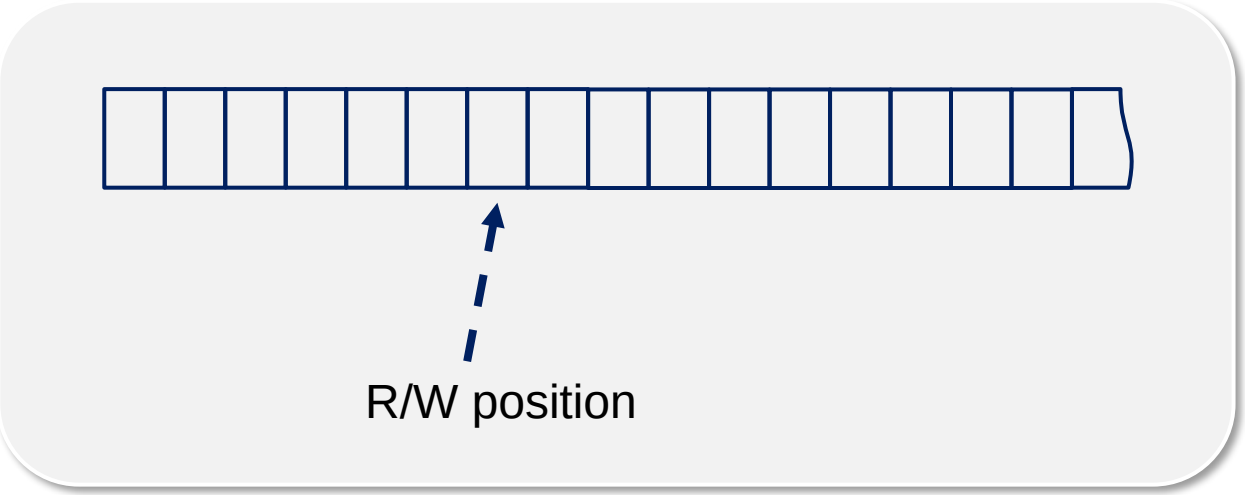


File (logical vision)

important

Alejandro Calderón Mateos

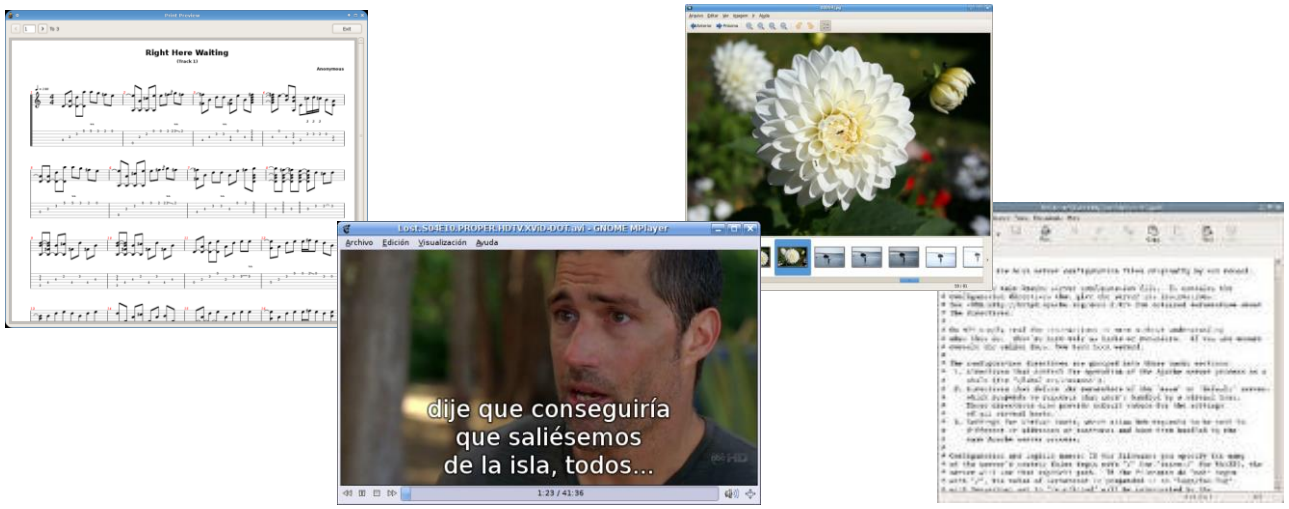
- Set of related information
  that has been defined by its creator.
- The content is usually represented by a sequence or row of bytes (UNIX, POSIX):



File

Alejandro Calderón Mateos

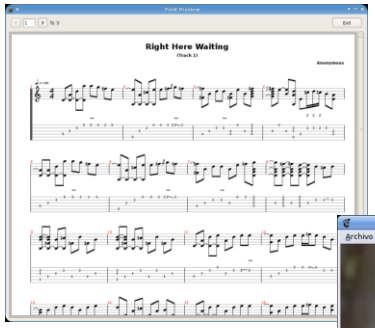
- Different types of information:

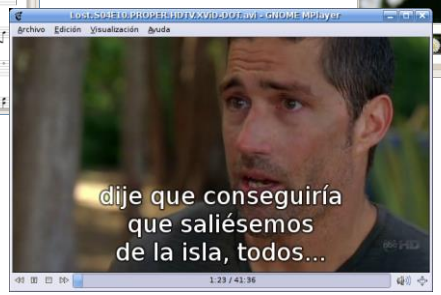


File (structure)

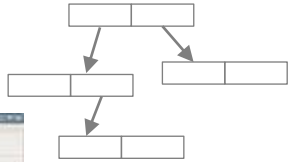
Alejandro Calderón Mateos

- Different types of information structures :

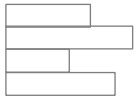




- Complex
  - Formatted (XML, etc.)
  - Relocatable



- Records
  - Fixed length
  - Variable length

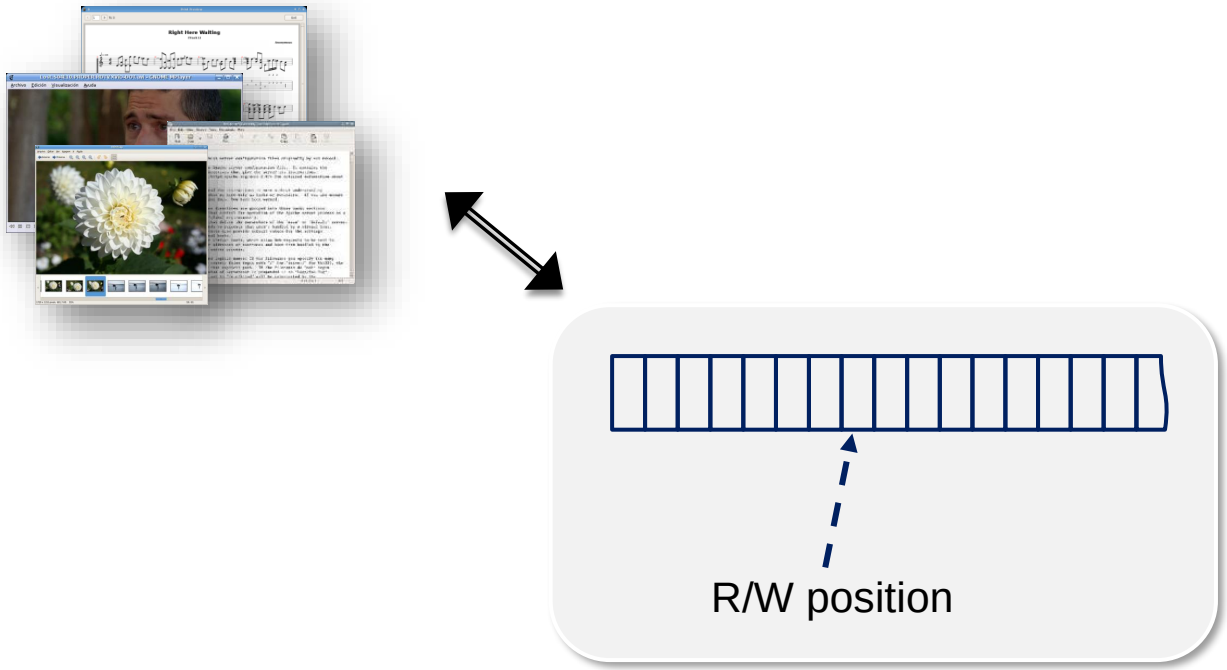


- Sequence of words

File

Alejandro Calderón Mateos

- Applications convert and store as a sequence or row of bytes.



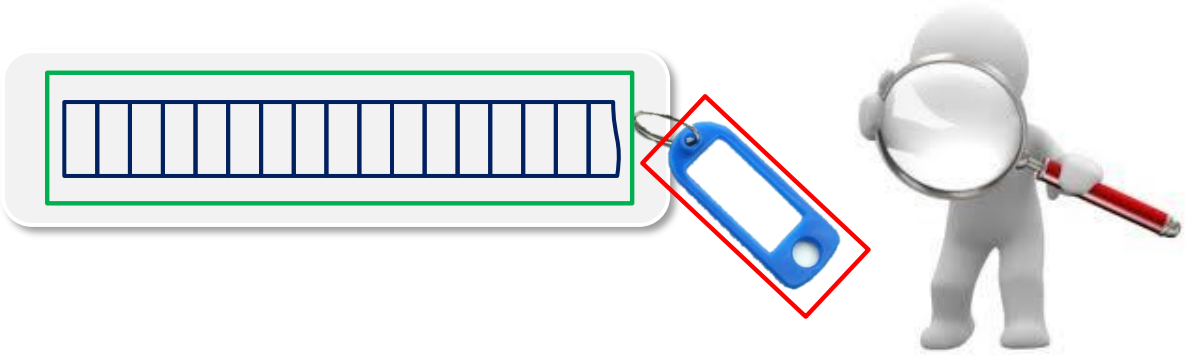
Contents

Alejandro Calderón Mateos

File

Alejandro Calderón Mateos

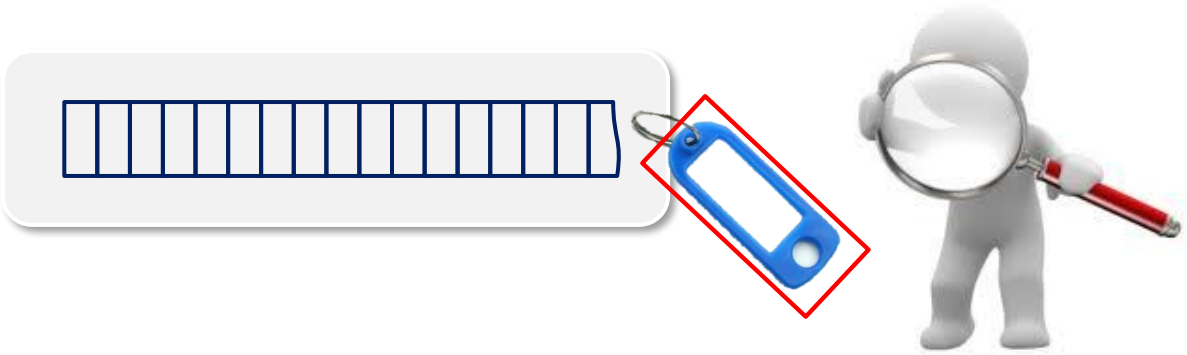
- Information of a file:
  - Data
    - **Information that the file stores.**
  - Metadata
    - **Information about the file.**
    - Different **attributes** of the file (+ information used by the O.S.)
    - File system-dependent.



File

Alejandro Calderón Mateos

- Information of a file:
  - Data
    - **Information that the file stores.**
  - Metadata
    - **Information about the file.**
    - Different **attributes** of the file (+ information used by the O.S.)
    - File system-dependent.



File: attributes

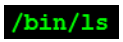
important

- Typical attributes of a file:
  - Name: identifier for users of the file.
  - Identifier: unique file label (numeric) used by the O.S.
  - Type: file type (for systems that require it)
    - E.g.: extension (.exe, .pdf, etc.)
  - Location: identifier that aids in locating the device blocks that belong to the file.
  - Size: current file size (in bytes or disk blocks).
  - Protection: access control and operations which user can do.
  - Temporary information: time of last access, creation, etc. that allows monitoring of file usage.
  - User identification: identifier of the creator, owner of the file, etc.

File name (and extension)

- Strings are used:
  - Allows users to better organize themselves.
  - Users do not remember names such as 00112233.
  - Directories associate name with their internal identifier.
- It is characteristic of each file system:
  - Length of name: fixed (MS-DOS) or variable (UNIX)
  - Case sensitive (Unix) or not (MS-DOS)
    - INMA vs inma
  - Extension required: yes, fixed (MS-DOS) or not (UNIX)

    - .zip -> identifies the file type (and the application to be used)



    - file name -> identifies by content (magic number)

Access control

- Access control lists (ACL):
  - ACL is a list associated with a file that is made up of ACE entries in the form of (user/group, permission).
  - E.g.: NTFS, Solaris UFS, HP-UX HFS, etc.
  - In Linux: `setfacl -d -m g:development:rw /home/devs`
- Permissions:
  - Condensed version used in traditional UNIX.
  - 3 categories: user, group, other.
  - 3 types of access per category: read, write, execute.

Contents

Alejandro Calderón Mateos

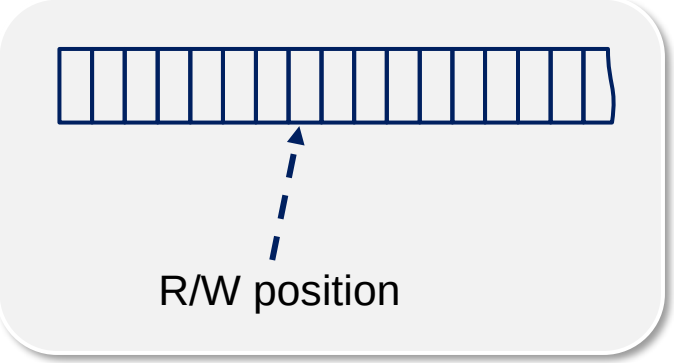
- Introduction
- **File**
  - Metadata
  - **Interface**
  - Access methods
  - Sharing semantics
- Directory
- File System
- Partitions/Volumes
- Devices
- System software
- File System (manager)

File: interface

important

Alejandro Calderón Mateos

- Generic interface to access information:
  - descriptor ← open (name, flags, mode)
  - close (descriptor)
  - read (descriptor, pointer, size)
  - write (descriptor, pointer, size)
  - lseek (descriptor, offset, whence)
  - ioctl (descriptor, command, pointer_to_command_arguments)



OPEN – Opening a file

Service	`#include <sys/types.h>`
	`#include <sys/stat.h>`
	`#include <fcntl.h>`
	
	`int open(char *pathname, int flags[, mode_t mode]);`

Arguments	▫ `pathname` file name (pointer to the first character). ▫ `flags` opening options: ■ `O_RDONLY` Read-only ■ `O_WRONLY` Writing-only ■ `O_RDWR` Reading and writing ■ `O_APPEND` Place the access pointer at the end of the open file ■ `O_CREAT` If it exists it has no effect. If it does not exist, it creates it ■ `O_TRUNC` Truncated if opened for writing ▫ permissions `mode`: ■ `S_I{RWX}{USR|GRP,OTH}` Read, Write, Execute x user, group, others
Return	A file descriptor or -1 in case of error.
Description	File opening (or creation if O_CREAT is used)

CLOSE – Closing a file

Service	`#include <unistd.h>`
	
	`int close(int fd);`

Arguments	`fd` file descriptor.
Return	Return 0 or -1 if error.
Description	The process closes the work session with the file, and the descriptor becomes free.

READ – Reading from file

Service	`#include <sys/types.h>`
	
	`ssize_t read(int fd, void *buf, size_t n_bytes);`

Arguments	▫ `fd` file descriptor ▫ `buf` data storage area ▫ `n_bytes` number of bytes to read
Return	Number of bytes actually read, 0 if end-of-file (EOF) and -1 if error
Description	▫ Attempts to read `n_bytes`. May read less data than requested (e.g., if the end of file is exceeded or interrupted by a signal). ▫ After reading, the file position pointer is updated with the number of bytes actually read.

WRITE – Writing to a file

Service	`#include <sys/types.h>`
	
	`ssize_t` **`write`**`(int fd, void *buf, size_t n_bytes);`

Arguments	▫ `fd` file descriptor ▫ `buf` data area to be written ▫ `n_bytes` number of bytes to write
Return	Number of bytes actually written or -1 if error.
Description	▫ Attempts to write `n_bytes`. May write less data than requested (e.g., if the maximum size of a file is exceeded or is interrupted by a signal). ▫ After writing, the file position pointer is updated with the number of bytes actually written. ▫ If the end of file is exceeded, the file increases in size.

LSEEK – Pointer position movement

Service	`#include <sys/types.h>`
	`#include <unistd.h>`
	
	`off_t lseek(int fd, off_t offset, int whence);`

Arguments	▫ `fd` file descriptor ▫ `offset` offset (in bytes, positive or negative) ▫ `whence` base of the offset
Return	▫ The new position of the pointer or -1 if error. ▫ Example: lseek(fd, 55, SEEK_SET) would return 55 if there is no error.
Description	▫ Modifies the read/write pointer associated to `fd` ▫ The new position is calculated as follows: ▪ `SEEK_SET` -> position = offset ▪ `SEEK_CUR` -> position = current position + offset ▪ `SEEK_END` -> position = file size + offset ▫ Jumping beyond the end is only consolidated if you write.

LINK – Link creation

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

Arguments	▫ `oldpath` name of the existing file to link to. ▫ `newpath` name of link to be created.
Return	Return 0 or -1 if error.
Description	▫ Creates a hard link from an existing entry (file or directory) in the same partition as the linked entry. ▫ Increases the link counter of the file.

UNLINK – Deletion of file

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

Arguments	`path` name of the file
Return	Return 0 if all correct or -1 if error.

Description	▫ Decrements the link counter of the file.
	▫ Si el contador es 0 entonces:
	  ▫ If it is not open, then delete the file and free its resources.
	  ▫ If it is open, closing it will delete it and free its resources.

File: POSIX interface

important

write

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>

int main ( int argc, char *argv[] )
{
 int fd1 ;
 char str1[10] ;
 int nb ;

 fd1 = open ("/tmp/txt1",
             O_CREAT|O_RDWR, S_IRWXU);
 if (-1 == fd1) {
    perror("open:");
    exit(-1);
 }

 strcpy(str1,"hola");
 nb = write (fd1,str1,strlen(str1));
 printf("written bytes = %d\n",nb);

 close (fd1);
 return (0) ;
}
```

read

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>

int main ( int argc, char *argv[] )
{
 int fd1 ;
 char str1[10] ;
 int nb, i ;

 fd1 = open ("/tmp/txt1",O_RDONLY);
 if (-1 == fd1) {
     perror("open:");
     exit(-1);
 }

 i=0;
 do {
      nb = read (fd1,&(str1[i]),1); i++;
 } while (nb != 0) ;
 str1[i] = '\0';
 printf("%s\n",str1);

 close (fd1);
 return (0);
}
```

Contents

Alejandro Calderón Mateos

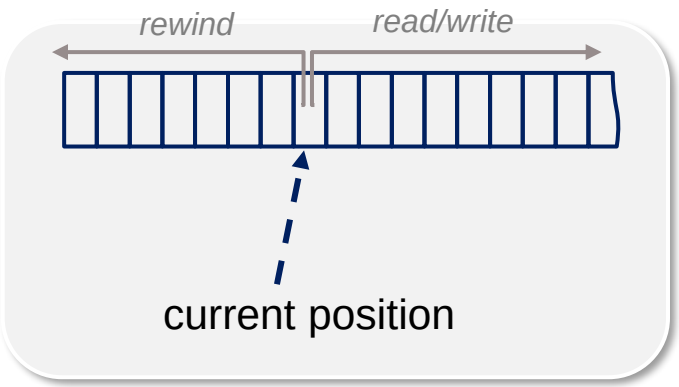
File: access methods

important

Alejandro Calderón Mateos

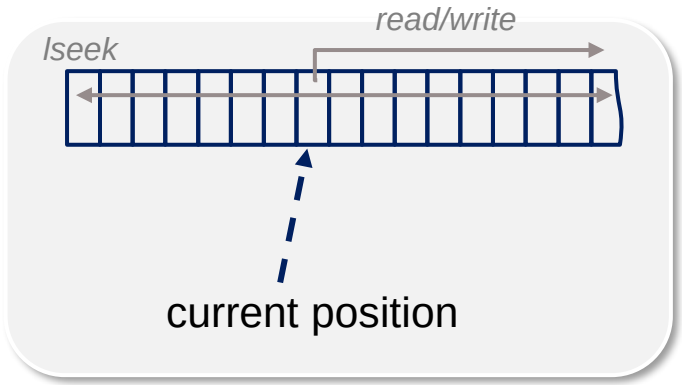
Sequential access:

- Sequential access devices: magnetic tapes.
- It is only possible to rewind to the beginning of the file.



Direct access:

- Random access devices: hard disks.
- It is possible to position (lseek) at any position in the file.
  - It allows other access methods to be built on top of it (e.g., indexing, etc.)



Contents

Alejandro Calderón Mateos

File: Sharing semantics

- Several processes can simultaneously access a file.
- It is required to define a coherence semantics:
  - When are the changes to a file observable by other processes?
- Options:
  - UNIX semantics.
  - Session semantics.
  - Version semantics.
  - Immutable file semantics.

File: Sharing semantics

important

Alejandro Calderón Mateos

Unix Semantics	Session semantics	Version semantics	Immutable semantics
Writes to a file are immediately visible to all processes (and the new R/W pointer).	Writes to a file are not visible to other processes: when closing it is made visible.	Scripts are made on version numbered copies: they are visible when consolidating versions.	If a file is declared shared, it cannot be modified.
Once opened (*open*), the created process family (*fork*) shares its image.	Once the file is closed, the following processes that open the file see the modifications.	Use explicit synchronization for immediate updates.	Until the lock is released, neither name nor content can be modified.
Contention by exclusive access to the single image of the file.	A file can be associated with several images. No contention.	It will have several images and cost to consolidate.	No concurrence.
Ext3, ufs, etc.	AFS (*Andrew File System*)	CODA	HDFS

OPERATING SYSTEMS: FILE SYSTEMS

Files, directories and file system