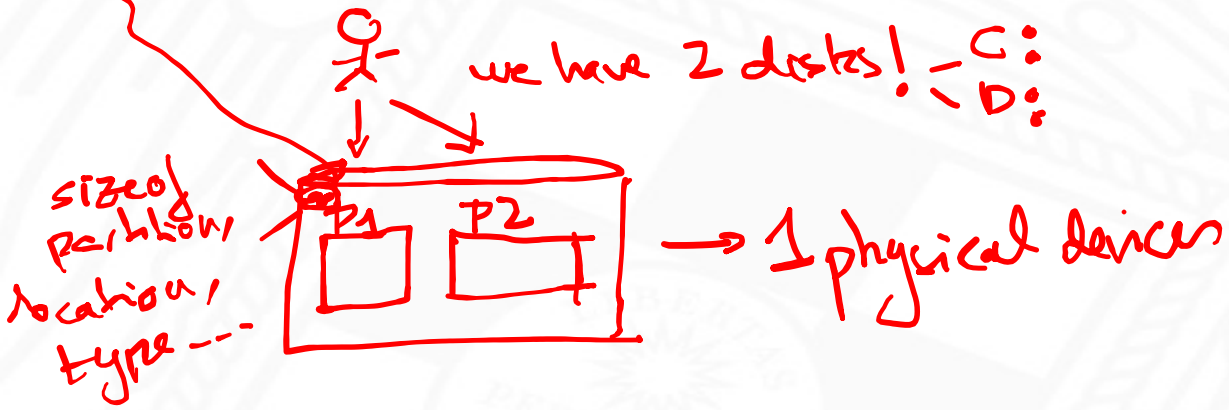
UNIVERSIDAD COMPLUTENSE MADRID

Partitions and filesystem (ESP-IDF)

Partitions

- Partition: region of a storage device that's managed independently from others
  - It is like a separate *virtual* device
- Partition table: structure that stores how the device is partitioned
  - It includes the size, type and location of each partition
  - It is usually stored in a fixed position of the device (and it is usually of fixed size)



Partitions in ESP32

- Offset 0x8000 in Flash
  - Size 3072 bytes
- Checksum (MD5) after partition table to guarantee integrity
  - Also signature after table if *secure boot* activated
- Default partition table:

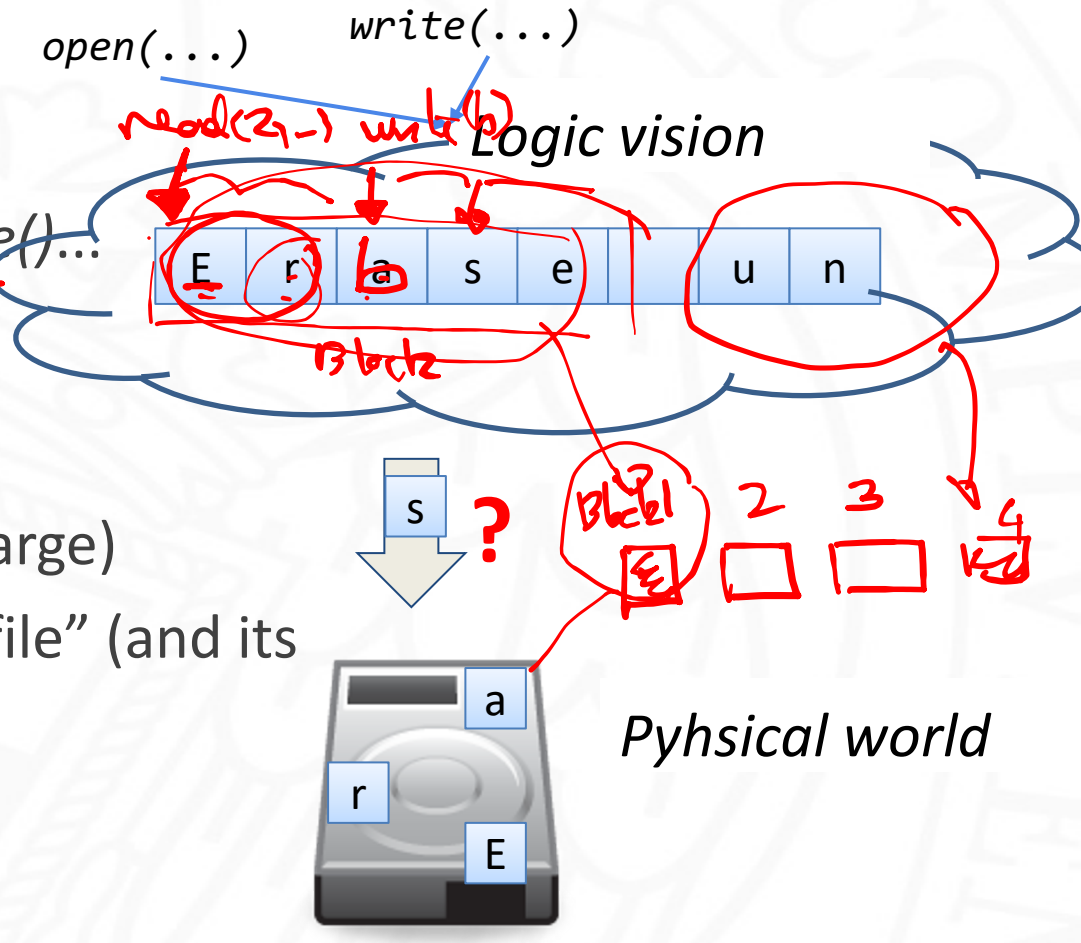
```
# Espressif ESP32 Partition Table
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000, 1M,
```

  - There is another predefined alternative with OTA partitions
  - We can define our own with a CSV file
  - We have to use *menuconfig* to choose the partition table

https://docs.espressif.com/projects/esp-idf/en/v4.1/api-guides/partition-tables.html

Filesystem

- How information is organized inside a partition
  - To create the illusion of files, folders…
- Logic vision of a file
  - Sequence of bytes with a current point
  - We can acces it using an API: *open(), read(), write()…*
- But, where is each byte in the device?
- The filesystem defines that relationship
  - The partition is divied in fixed size blocks (~4KB large)
  - Filesystem has to define the link the “name of a file” (and its path) to the relevant device blocks



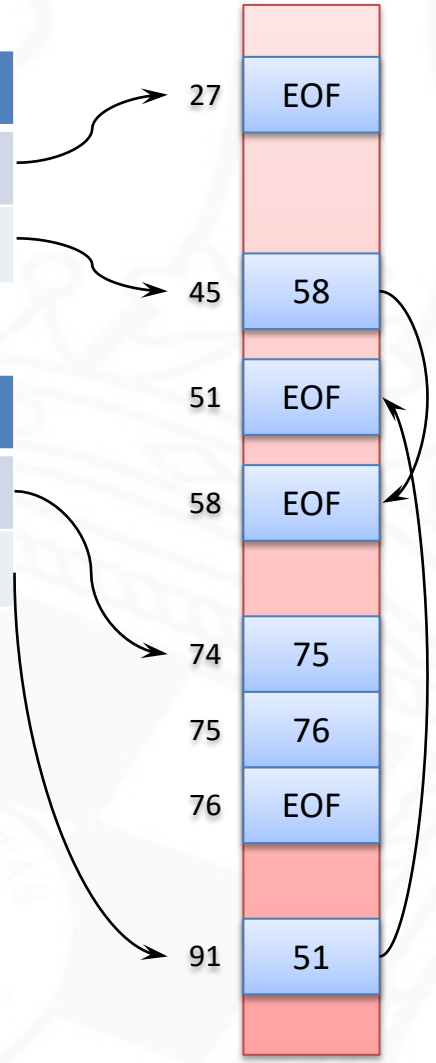
Example filesystem (FAT)

Root folder

Name	Atrib.	KB	Block
pep_dir	dir	5	27
fiche1.txt		12	45

Folder pep_dir

Name	Atrib.	KB	Block
carta1.wp	R	24	74
prue.zip		16	91



Logic-to-physic translation with FAT

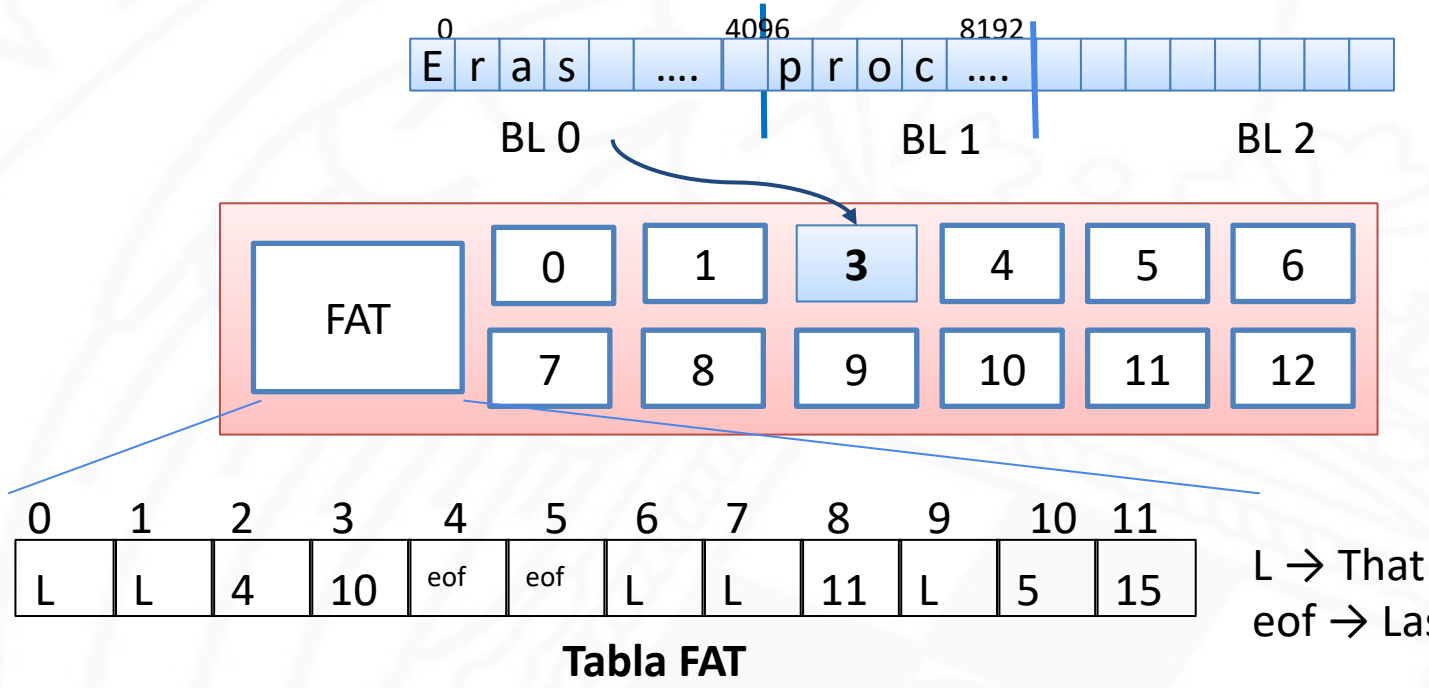
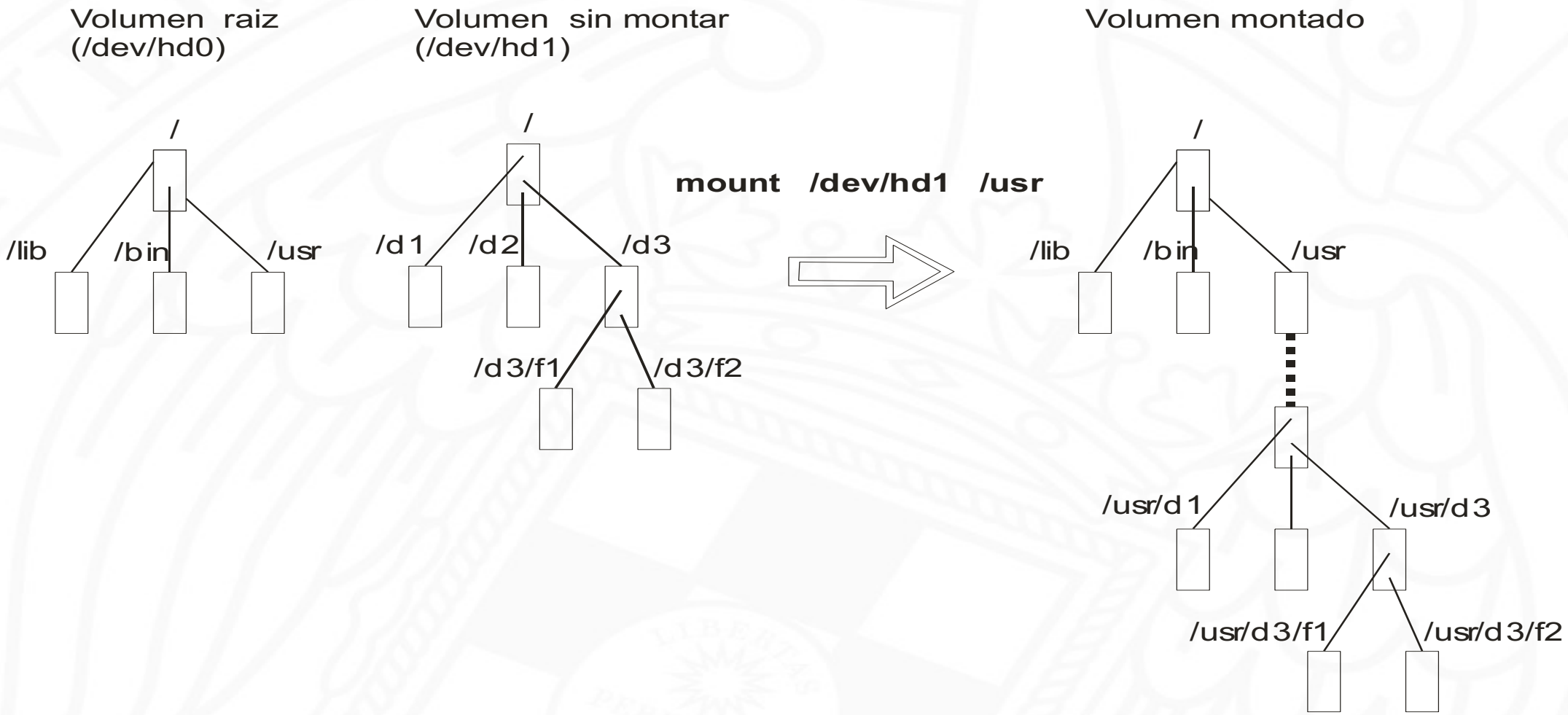


Tabla FAT

BL 0 → BF 3
BL 1 → BF 10
BL 2 → BF 5

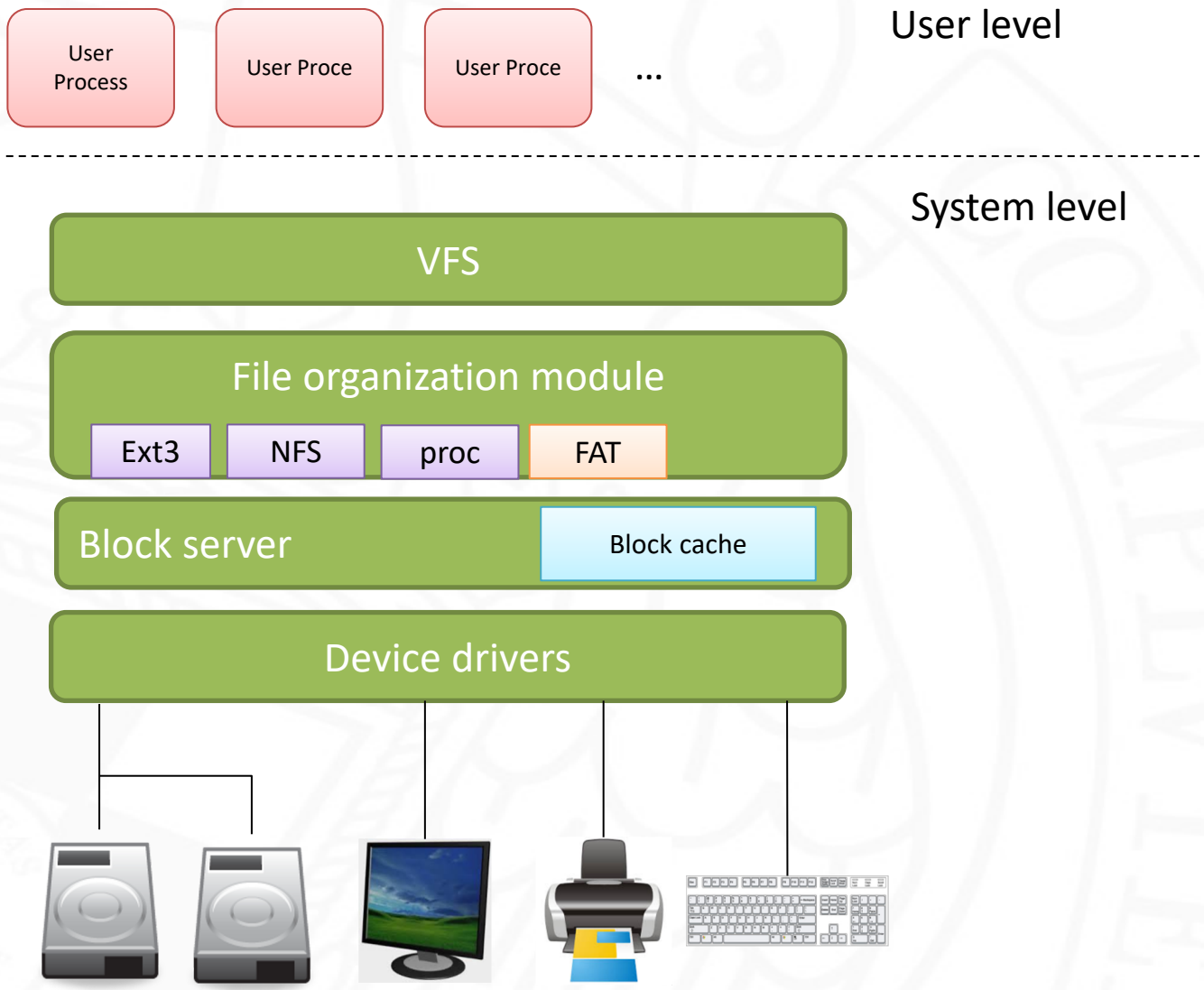
L → That block is **free** (0,1…)
eof → Last block of a file

Mounting filesystems



Virtual File System

- Abstraction layer of the OS to offer a common API to work with files
  - It hides the existance of several filesystems
  - From the path of the mounting point it determines the type of filesystem (FAT, ext2, ext4...)
  - And selects the right operation for that filesystem
    - read() → read_fat()



ESP-IDF. Partitions and filesystems

- ESP-IDF supports FAT y SPIFFS
- ESP-IDF allows to define partitions of type *data* and subtype *fat* or *spiffs*
  - CSV file examples in
    - https://github.com/espressif/esp-idf/tree/master/components/partition_table
  - Documentation
    - https://docs.espressif.com/projects/esp-idf/en/v4.1/api-guides/partition-tables.html

```
# Name,     Type,     SubType,     Offset,       Size,          Flags
nvs,        data,     nvs,         0x9000,       0x6000,
phy_init,   data,     phy,         0xf000,       0x1000,
factory,    app,      factory,     0x10000,      1M,
storage,    data,     fat,                 ,     1M,
```

.csv file has to be located at the root of the project

menuconfig allows to show the name of the CSV file

If using PlatformIO we must include *board_build.partitions = name.csv* in `platformio.ini`

Using *flash* command we write both the table and the app

ESP-IDF. Mounting filesystems

- To use a FAT filsesystem we need to
  - Register FAT in the VFS
  - Mount the filesystem
  - https://docs.espressif.com/projects/esp-idf/en/v4.1/api-reference/storage/fatfs.html
- It is convenient to uses the *Wear levelling* component
  - It distributes write operations to increase FAT lifetime
  - Integrated in the FAT file sytem
  - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/storage/wear-levelling.html
  - https://github.com/espressif/esp-idf/tree/release/v4.1/examples/storage/wear_levelling
  - There is a function that registers and mounts a FAT system:
    - `esp_vfs_fat_spiflash_mount()`

ESP-IDF. FAT Example

```
const char* char *base_path = "/spiflash";
static wl_handle_t s_wl_handle = WL_INVALID_HANDLE;
const esp_vfs_fat_mount_config_t mount_config = {
   .max_files = 4,
   .format_if_mount_failed = true,
   .allocation_unit_size = CONFIG_WL_SECTOR_SIZE
};
void app_main(void) {
...
  esp_err_t err = esp_vfs_fat_spiflash_mount(base_path, "storage", &mount_config, &s_wl_handle);
....

    FILE *f = fopen("/spiflash/hello.txt", "wb");
    fprintf(f, "written using ESP-IDF %s\n", esp_get_idf_version());
    fclose(f);
    ...
    esp_vfs_fat_spiflash_unmount(base_path, s_wl_handle);
```