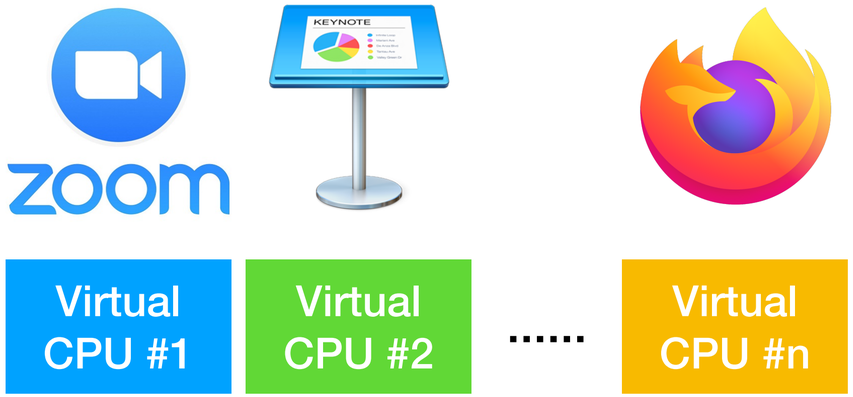
# Virtualization and File Systems

# What are we studying in OS?

## Short answer:
## one-to-many virtualization

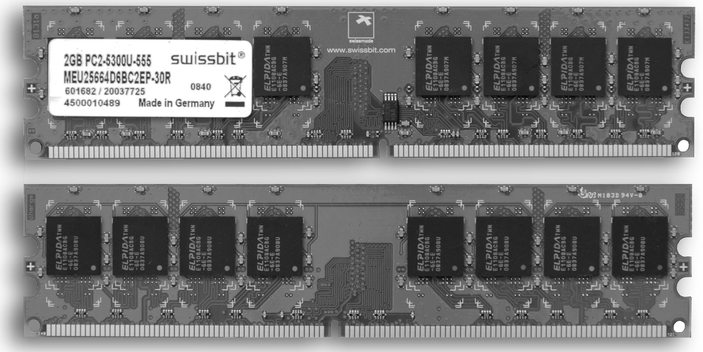


# Virtualizing CPU




Virtualize


One physical CPU
to many virtual CPU




# Virtualizing Memory

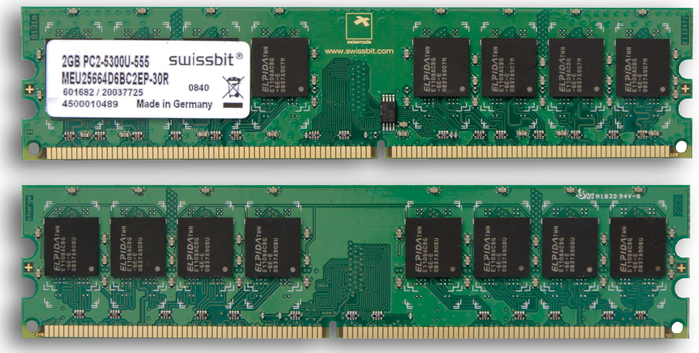
One physical Memory



Virtual memory address space #1


Virtual memory address space #2

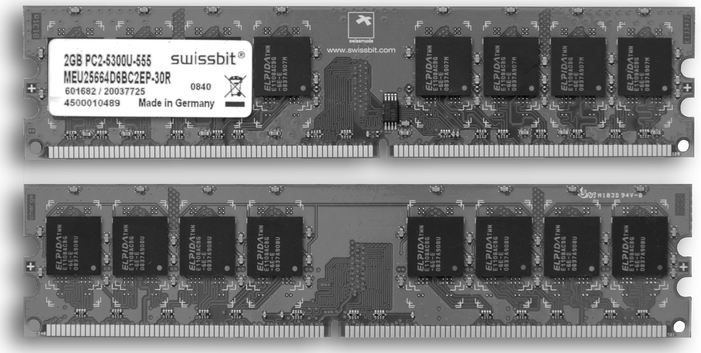
Virtualize




to many virtual Memory

# Virtualizing I/O

One physical I/O device to many virtual I/O device






Virtualized device #1

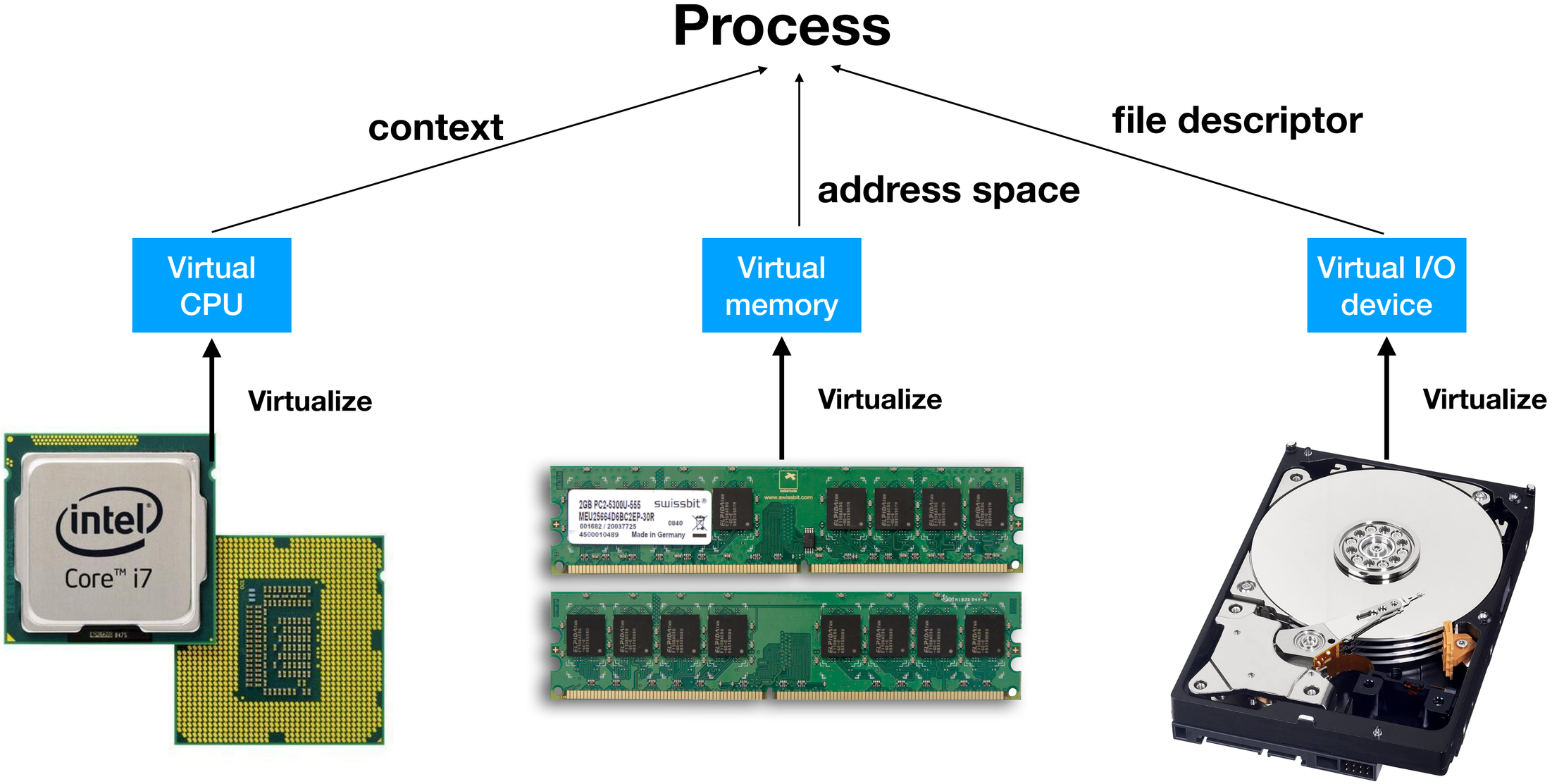
Virtualized device #2

Virtualize

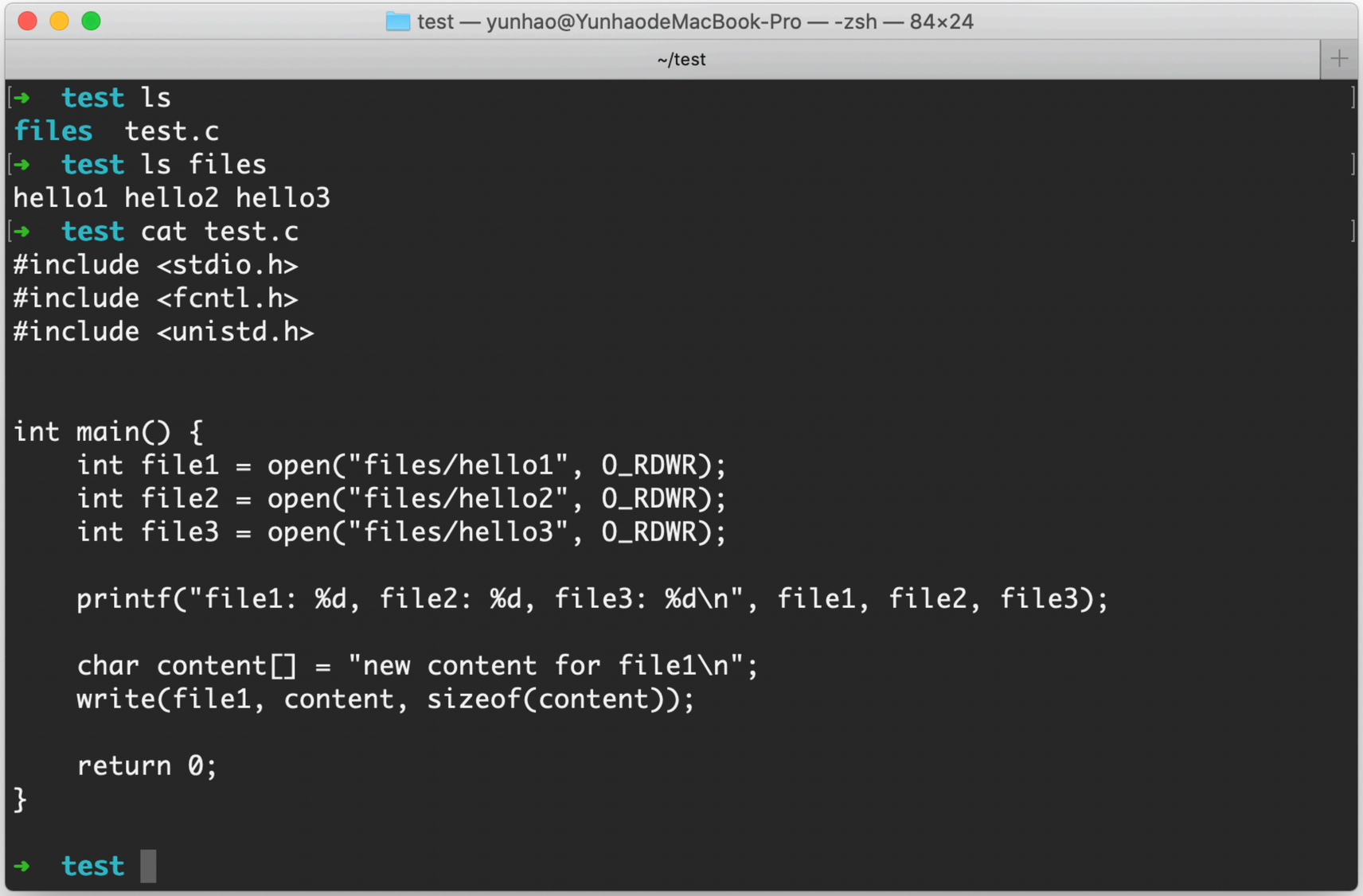
# What are we studying in OS?

- One-to-many virtualization
  - One physical CPU to many virtual CPUs: each user application runs on its own virtual CPU.
  - One physical memory to many virtual memory spaces: each user application runs with its own virtual memory.
  - One physical I/O device to many virtual I/O devices: each user application uses its own files / network connections.
- A process is a “virtual machine” with a virtual CPU, a piece of virtual memory and some virtual devices.

# Big picture: One-to-many Virtualization



# Understanding File Descriptors



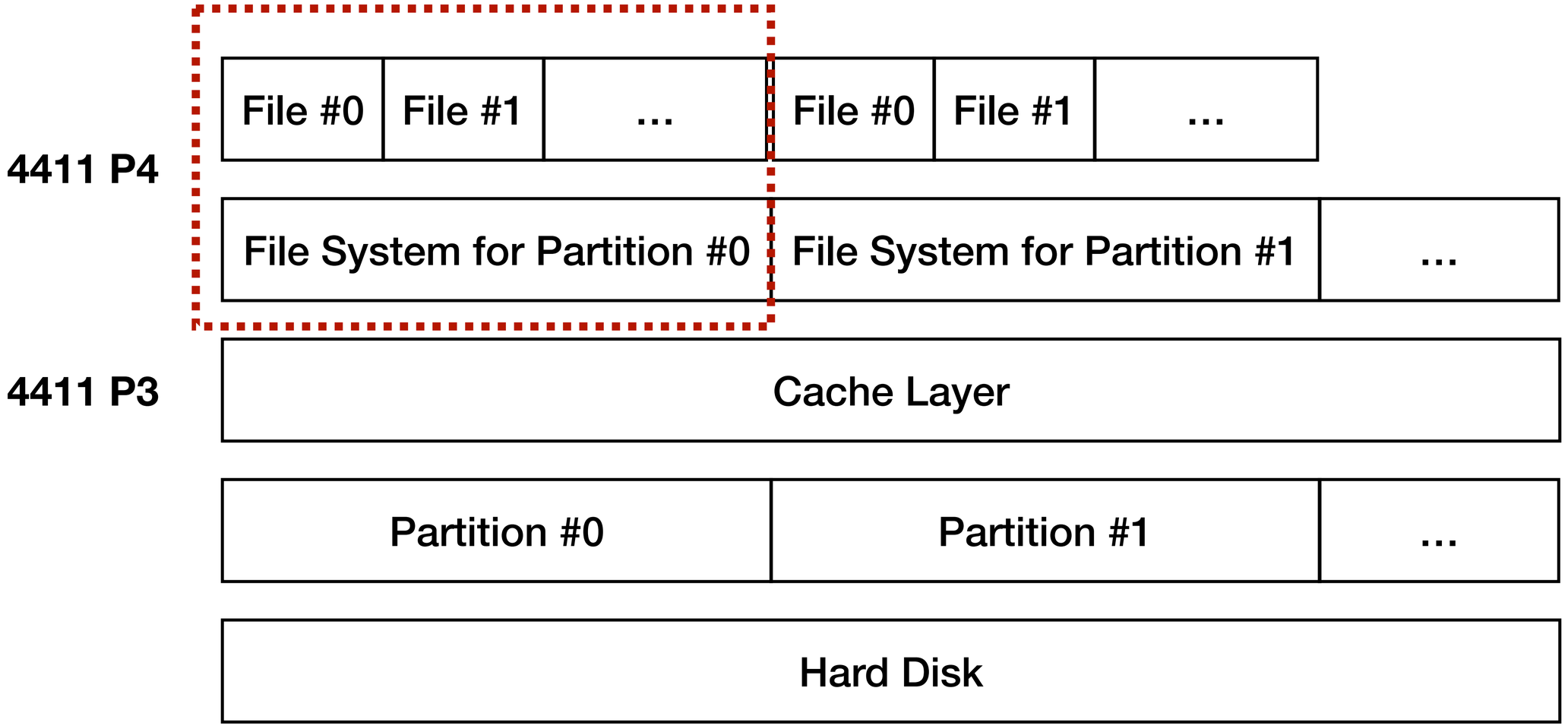
# Other Types of Virtualization

- One-to-many virtualization
- Many-to-one virtualization
  - e.g., RAID
- A-to-B virtualization
  - e.g., run a Linux on top of Windows using Virtual Box
- Virtualization has been the key technology empowering cloud computing.

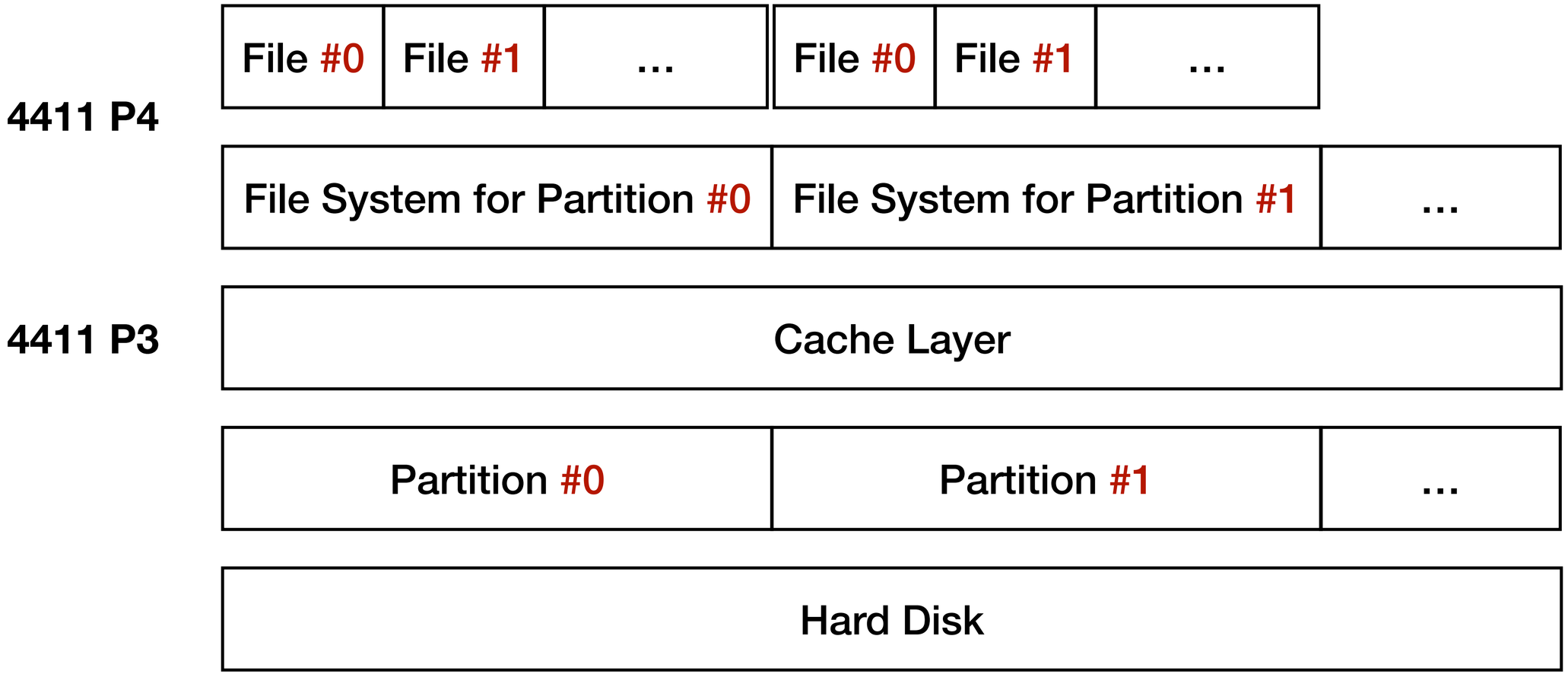
# File Systems in EGOS

One-to-many virtualization for disks:
virtualizing one storage device to many files

# Layering Design



# Inode Number for One-to-many



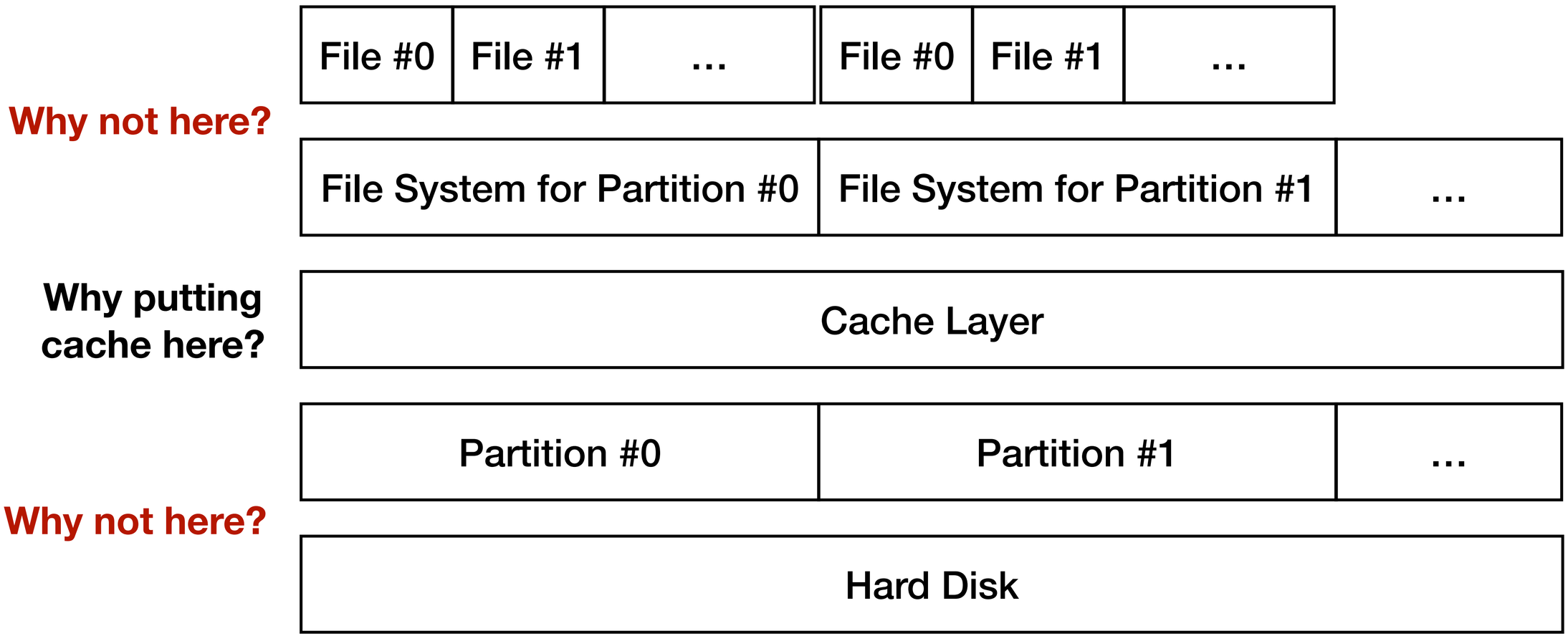
# Power of Layering: Unified Interface

```
typedef struct block_store {
    void *state;
    int (*getninodes)(struct block_store *this_bs);
    int (*getsize)(struct block_store *this_bs, unsigned int ino);
    int (*setsize)(struct block_store *this_bs, unsigned int ino, block_no newsize);
    int (*read)(struct block_store *this_bs, unsigned int ino, block_no offset, block_t *block);
    int (*write)(struct block_store *this_bs, unsigned int ino, block_no offset, block_t *block);
    void (*release)(struct block_store *this_bs);
    int (*sync)(struct block_store *this_bs, unsigned int ino);
} block_store_t;
```

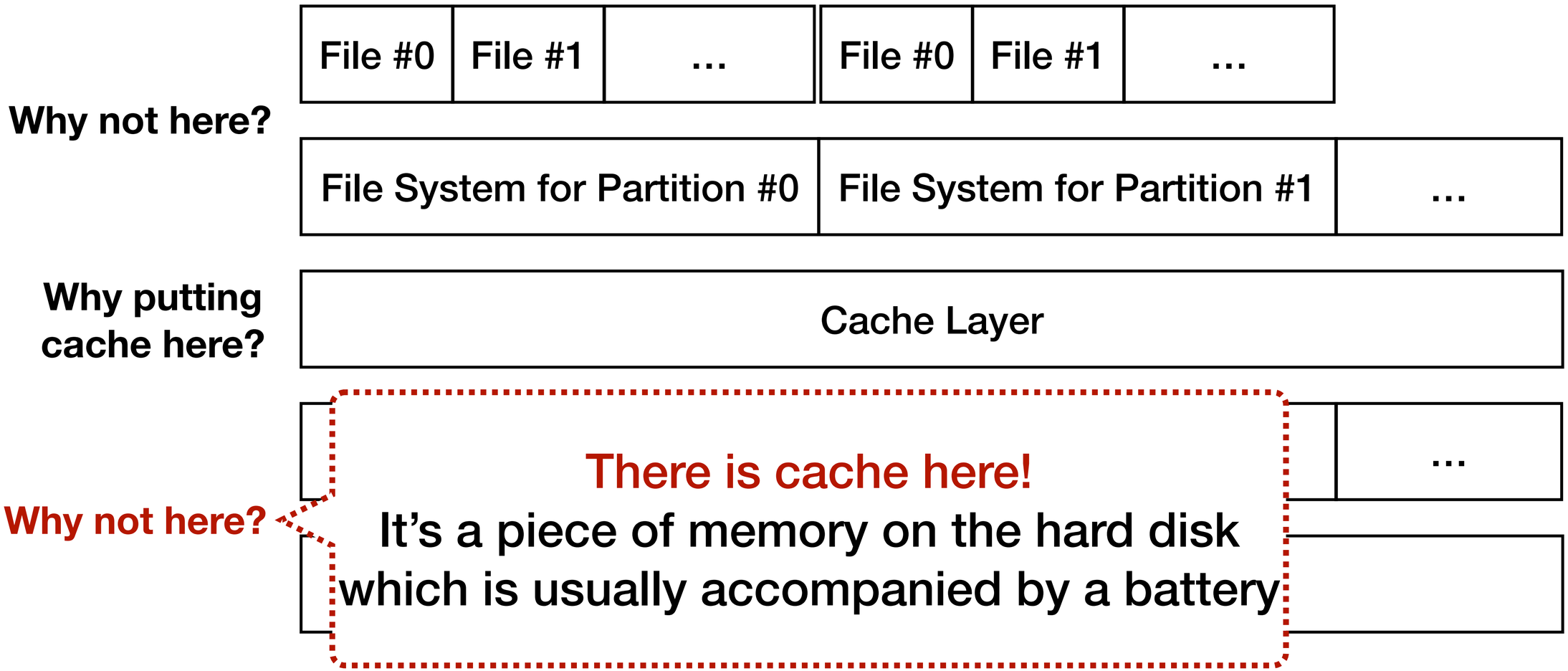
**src/h/egos/block_store.h**

**Read the `block_init` function in src/apps/blocksvr.c**

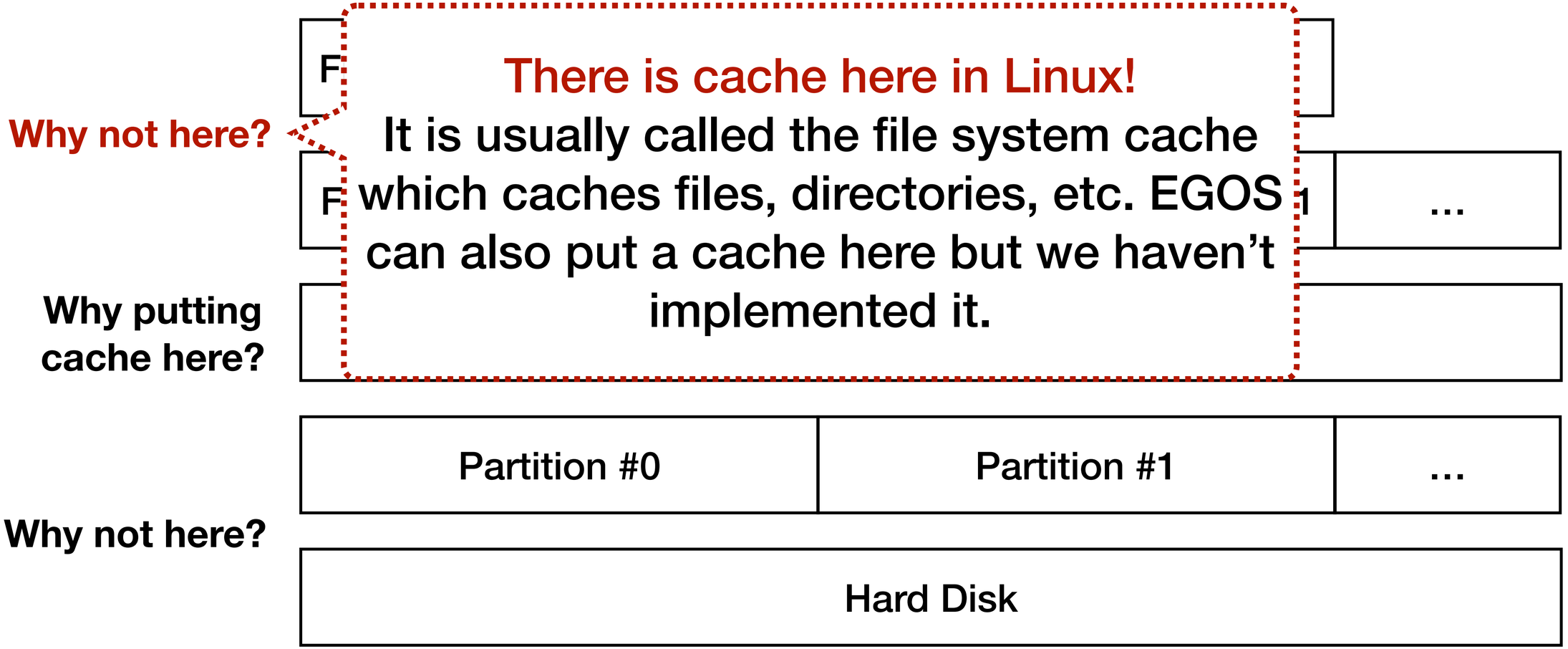
# Question: Where to put the cache layer?



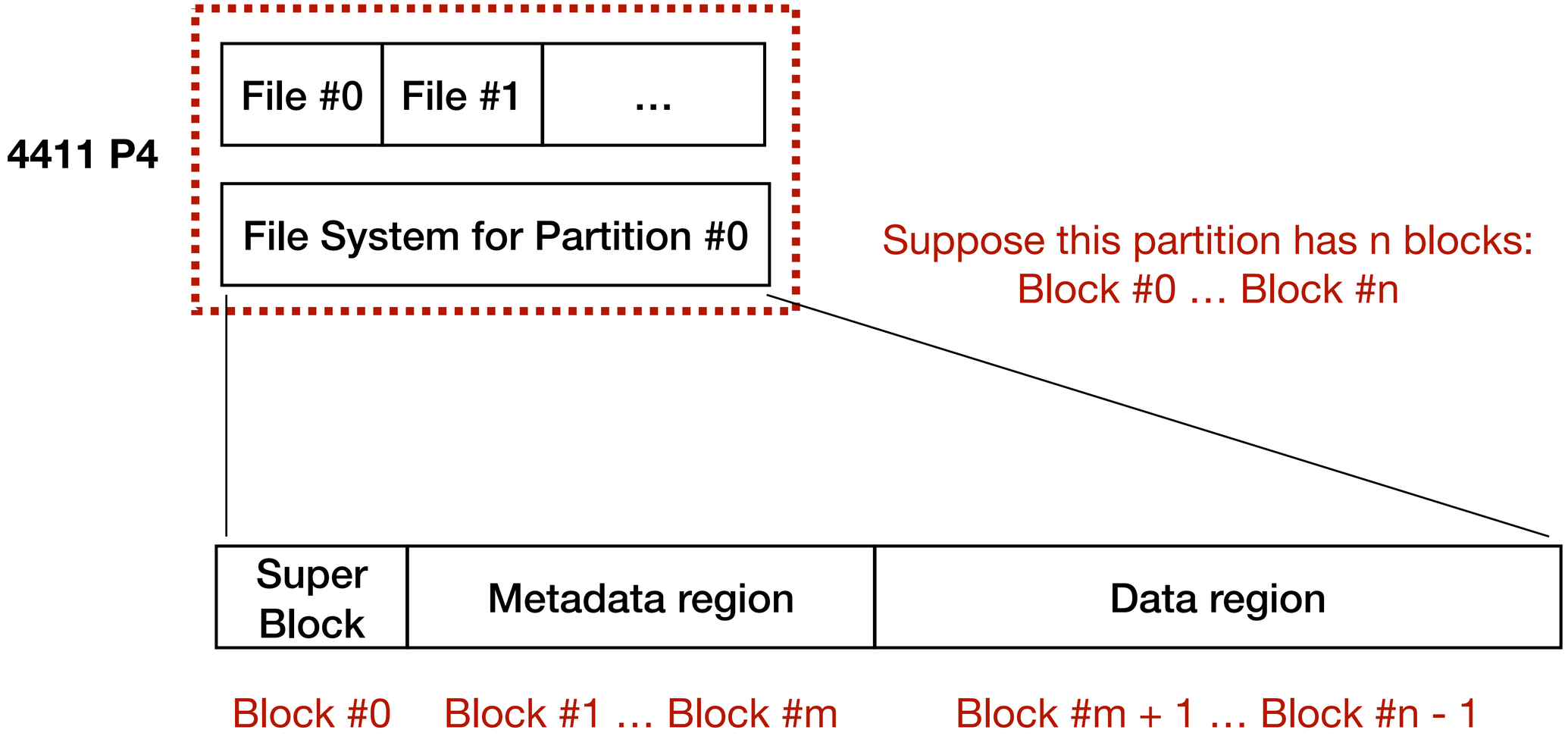
# Disk Hardware Cache



# File System Cache

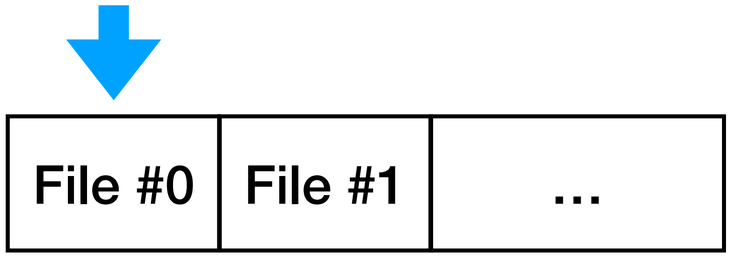


# First Look into File Systems: Layout



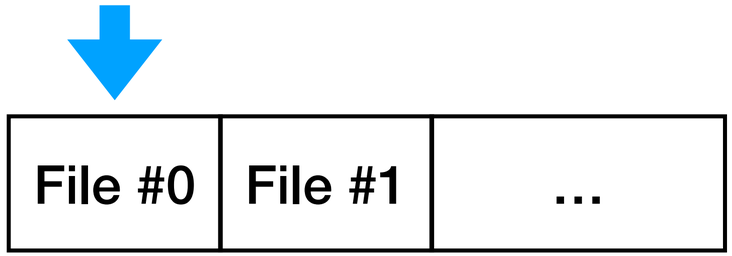
# General Flow of Read

1 `Read (ino, offset)`

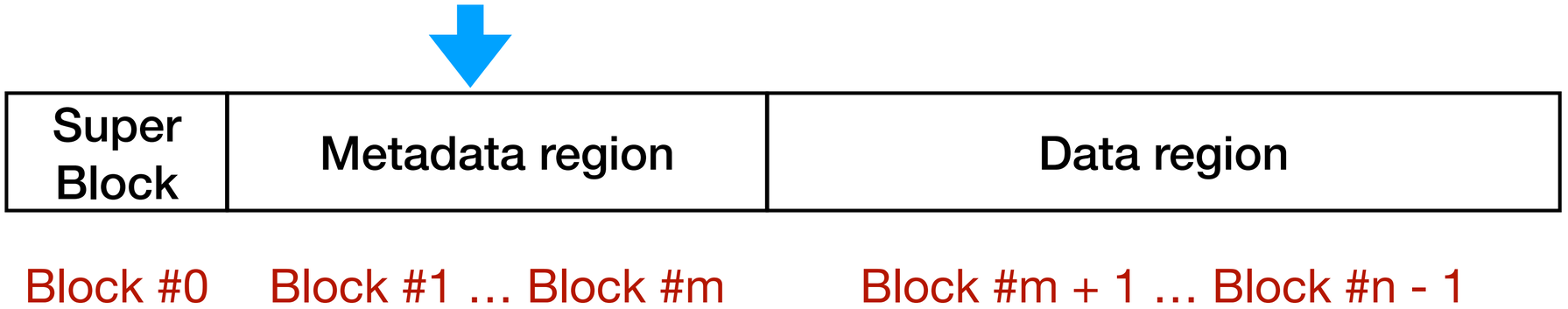



# General Flow of Read

1 `Read (ino, offset)`

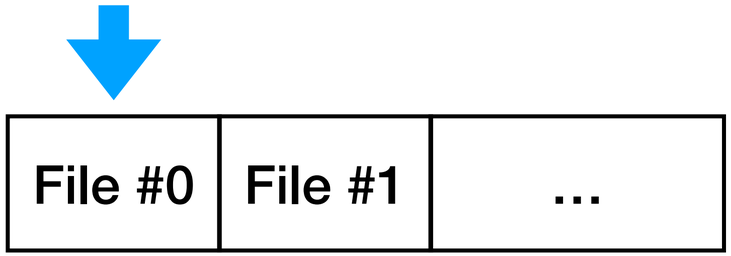



2 `Read the metadata of ino`



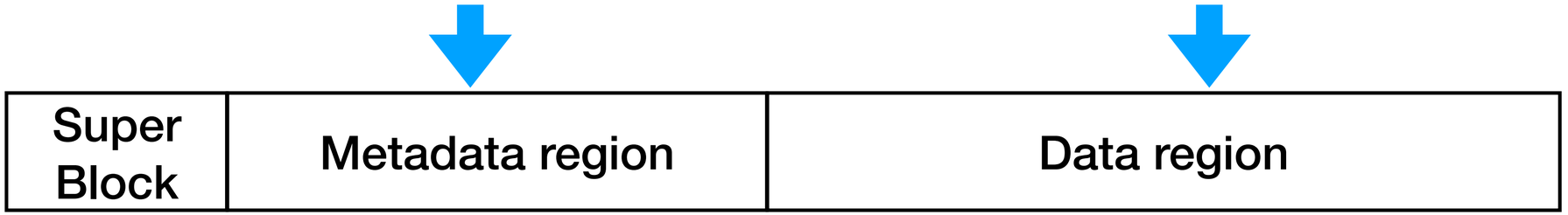
# General Flow of Read

1 Read (ino, offset)




2 Read the metadata of ino

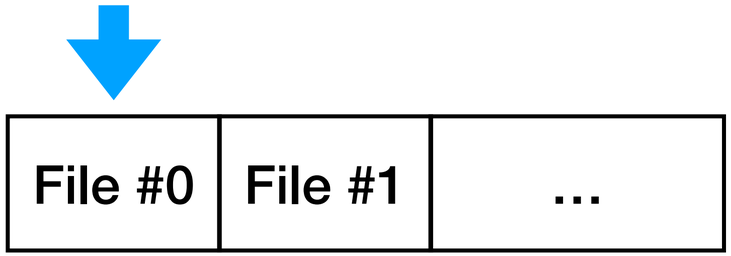
3 Read the data of ino




Block #0 Block #1 … Block #m Block #m + 1 … Block #n - 1

# Key Challenge: Maintain Metadata

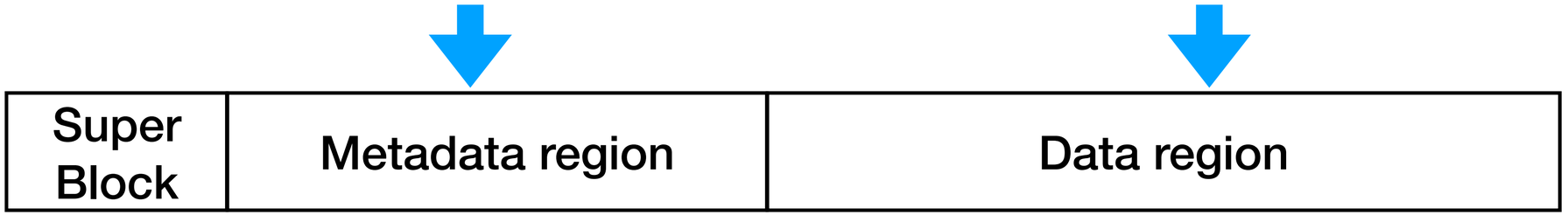
1 Read (ino, offset)




Different file systems maintain the metadata in different ways and in P5, you will implement and maintain a simple way using linked lists.

2 Read the metadata of ino

3 Read the data of ino




Block #0 Block #1 … Block #m Block #m + 1 … Block #n - 1

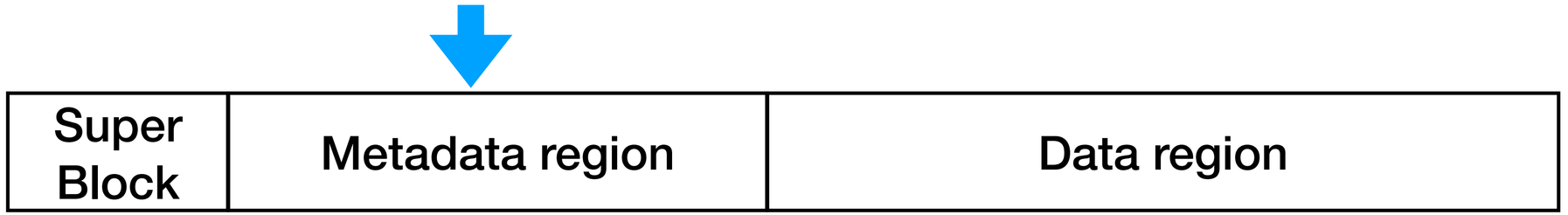
# Maintain a Linked-list in Memory

- This is similar to what you did in P0.
  - Allocate memory and then simply read/write.
  - But how to maintain such a data structure on disk?

# 3 steps to maintain a Linked-list on disk

- Step1: allocate a buffer in memory and read a disk block into the buffer.

```
Read & update the metadata of ino
```

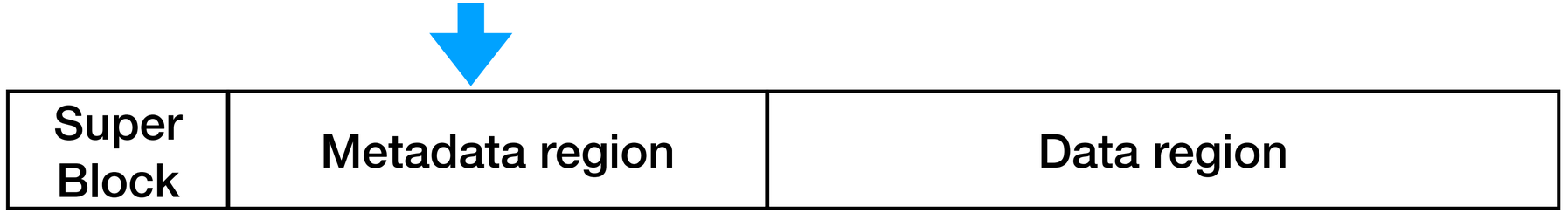



```
char block[BLOCK_SIZE];
(*cs->below->read)(cs->below, ino, offset, block);
```

# 3 steps to maintain a Linked-list on disk

- Step2: read/write the data structure in this memory buffer.

```
Read & update the metadata of ino
```

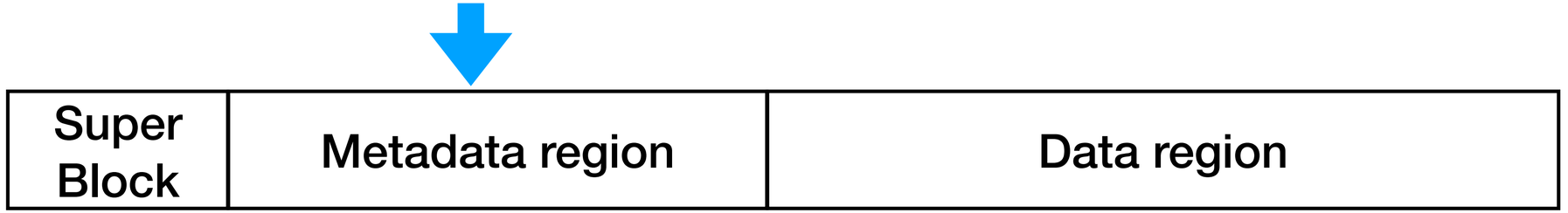



```
// example to increase the size of one file by one block
// variable block was defined in previous slide
struct fatdisk_inodeblock* inode_block = block;
inode_block->inodes[idx].nblocks += 1;
```

# 3 steps to maintain a Linked-list on disk

- Step3: write the memory buffer back to disk.

```
Read & update the metadata of ino
```




```
(*cs->below->write)(cs->below, ino, offset, block);
```

# Take-aways

- OS studies virtualization of CPU, memory and I/O, leading to the concepts of context, virtual memory and file.
- File systems adopt a layering design: each layer has a specific purpose.
- The general layout of a file system contains super block, metadata region and data region. (P5 handout splits the metadata region into two regions)
- There are 3 steps to maintain the data structures in a file system: read from disk to memory; read/write memory; write back to disk.

# Homework

- P5 is due on Dec 11. Implement the FAT file system.
- Read the `block_init` function in src/apps/blocksvr.c
- Next lecture on Dec. 2: Makefile and testing (P4).