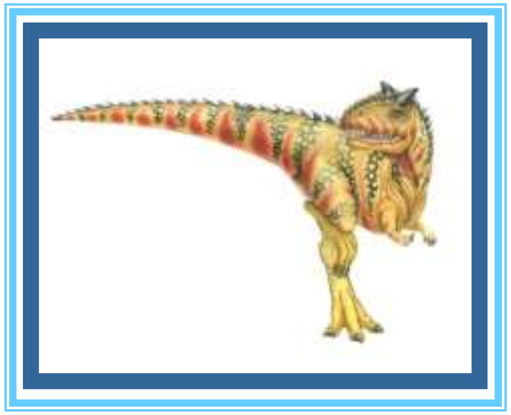
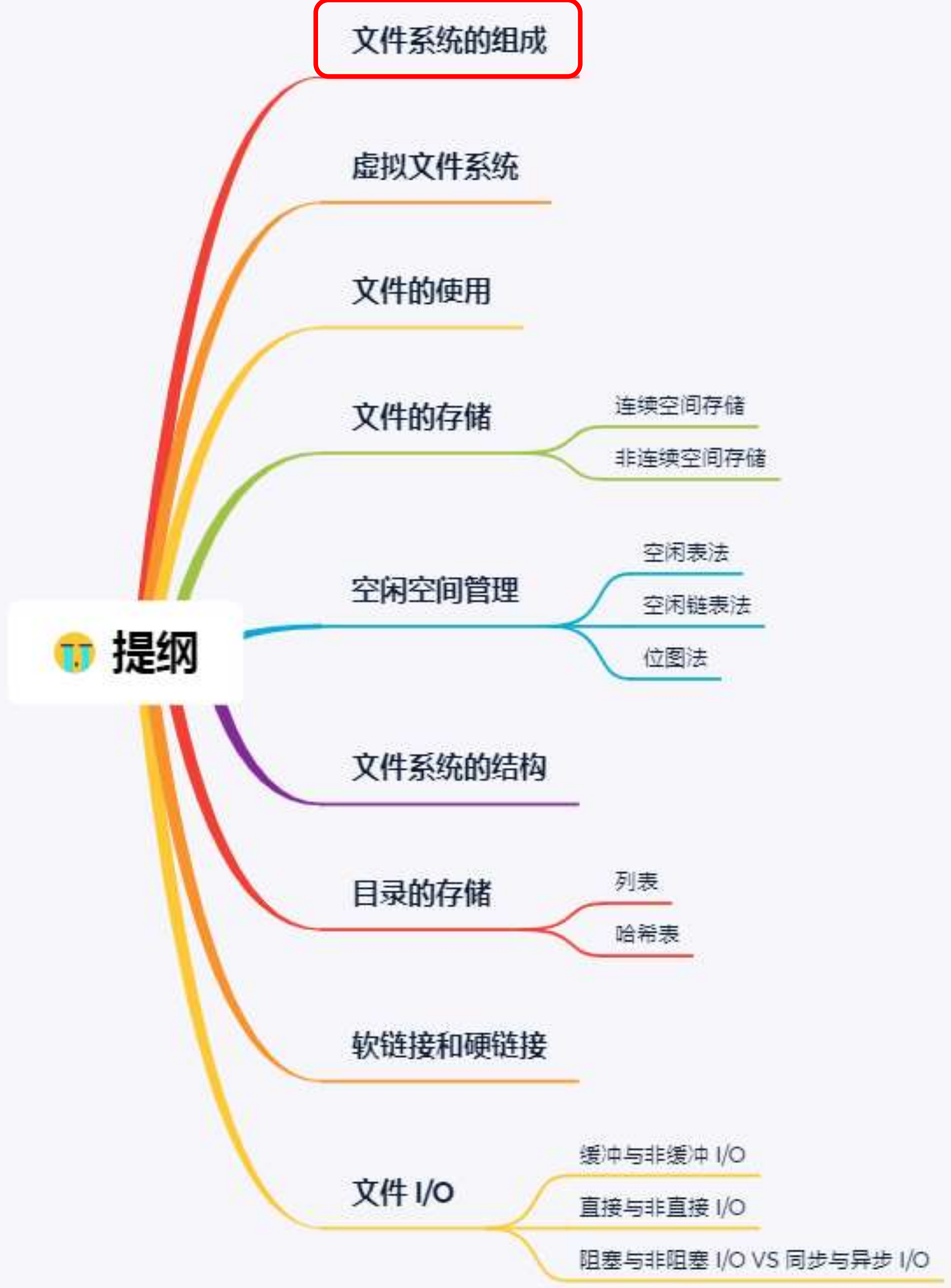
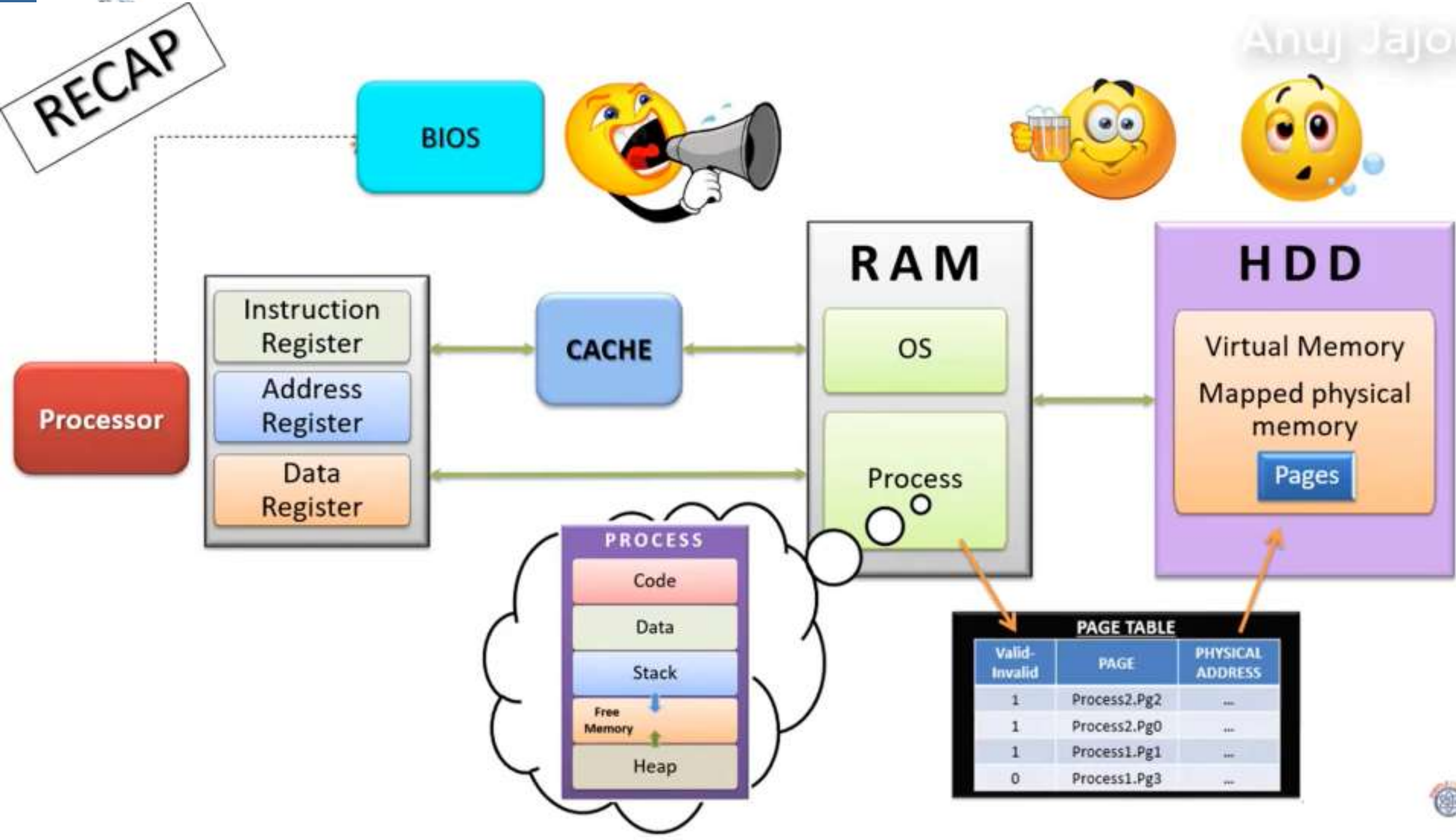
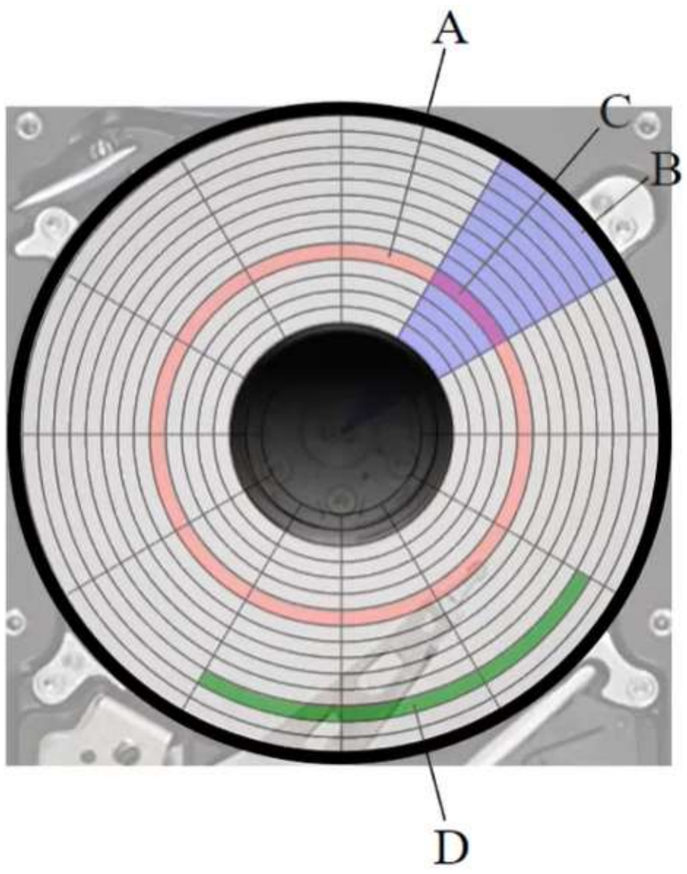
Chapter 11: File-System




提纲
文件系统的组成
虚拟文件系统
文件的使用
文件的存储
连续空间存储
非连续空间存储
空闲空间管理
空闲表法
空闲链表法
位图法
文件系统的结构
目录的存储
列表
哈希表
软链接和硬链接
文件 I/O
缓冲与非缓冲 I/O
直接与非直接 I/O
阻塞与非阻塞 I/O VS 同步与异步 I/O



Valid-Invalid	PAGE	PHYSICAL ADDRESS
1	Process2.Pg2	...
1	Process2.Pg0	...
1	Process1.Pg1	...
0	Process1.Pg3	...



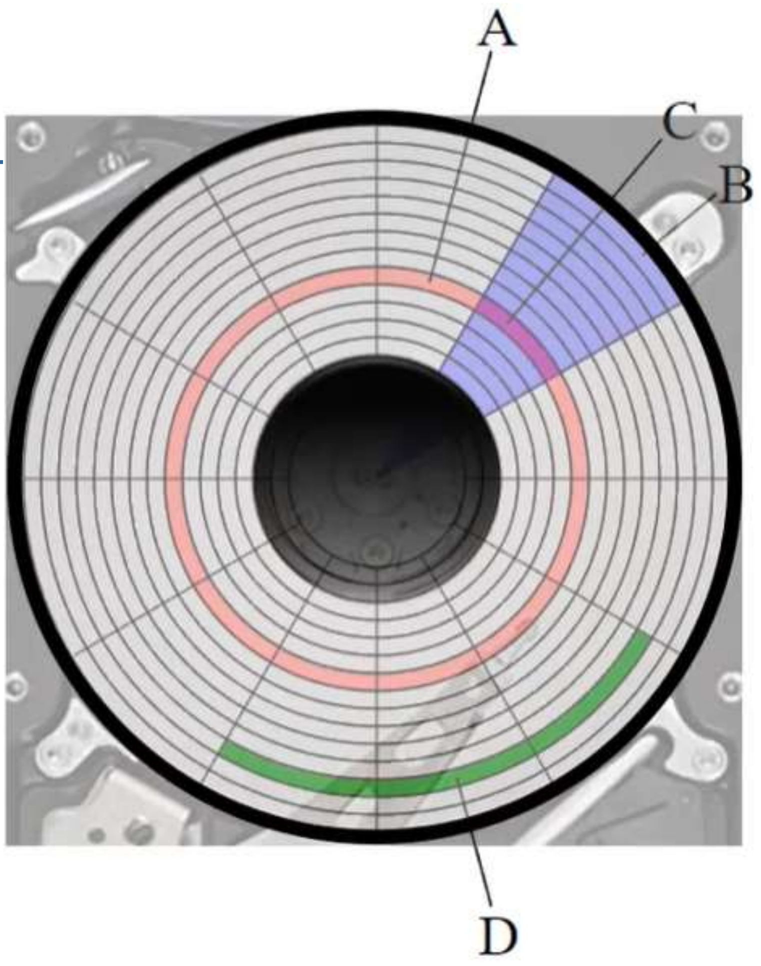
So many users:

User 1: have pdf, docx, ……

User 2: have ……

User n:……

Need to find an efficient way to manage all files of all users

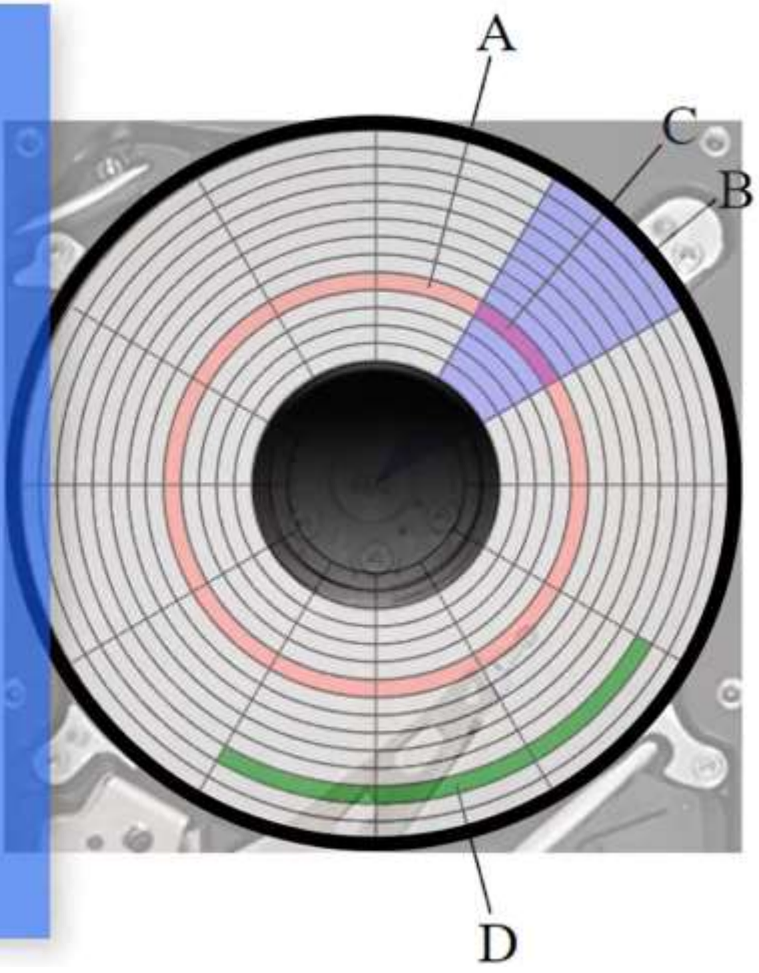


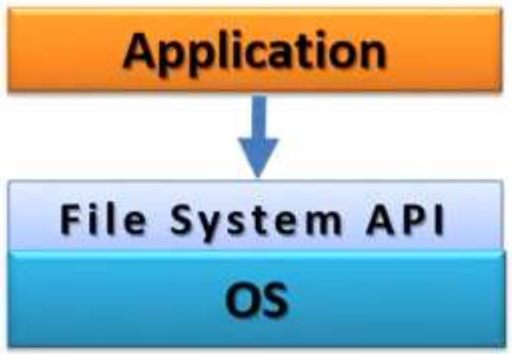
How to prevent the overwriting of one application data by the other applications?

When an application tries to save some data on the hard disk, the addresses of the track sectors cannot expose to the applications, for the security reasons.

File API

create()
open()
read()
write()
append()
seek()
close()
delete()
...





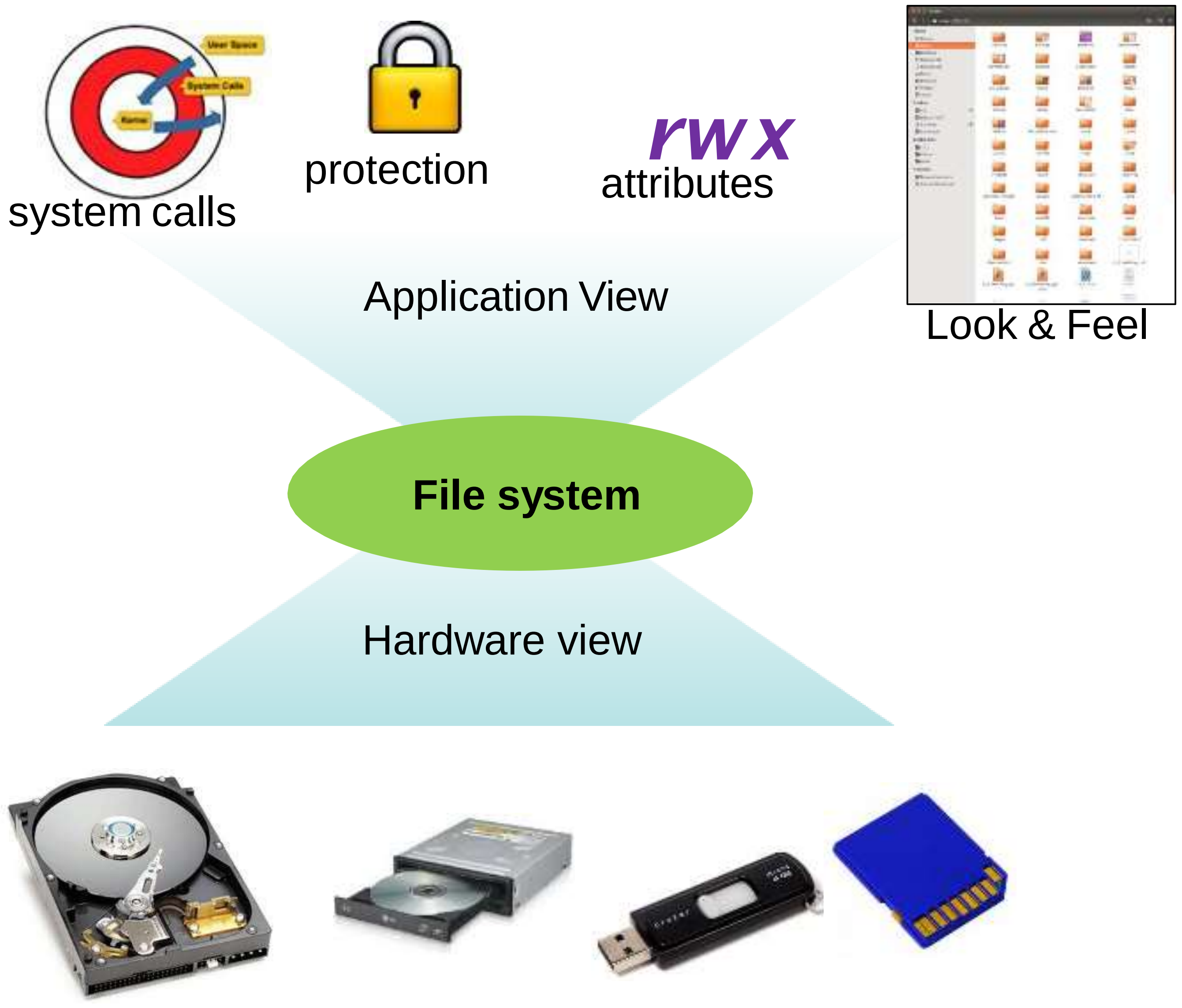
Disk structures:

A. Track
B. Geometrical sector
C. Track sector

So, the OS allows data to be stored on the secondary memory through **file** system

The basic data unit of a file system is a file

Two views of a file system

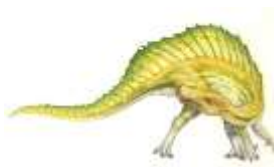


Files

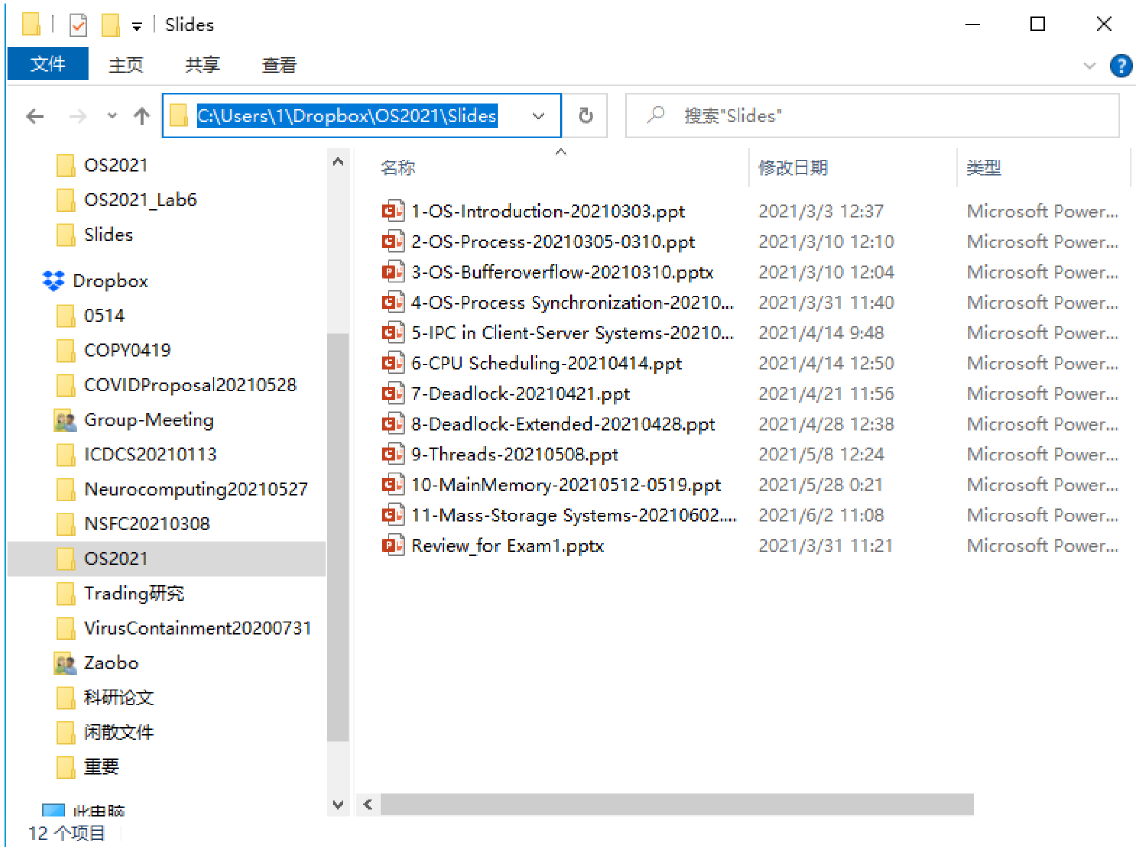
- From a user's perspective,
  - A byte array
  - Persistent across reboots and power failures
- From OS perspective,
  - Secondary (non-volatile) storage device
  - Map bytes as collection of blocks on storage device
  - It is only because the OS is creating the illusion of a file that they exist
- From the computer's perspective,
  - There is only blocks of memory, either allocated or unallocated

Why is file system needed?

- To make the computer system be convenient to use, the OS provides a uniform **logical view** of stored information
- The OS abstracts from the physical properties of its storage devices to define a logical storage unit, the **file**
- Files are mapped by the operating system onto physical devices.



Why is file system needed?



Slides

文件 主页 共享 查看

C:\Users\1\Dropbox\OS2021\Slides

搜索"Slides"

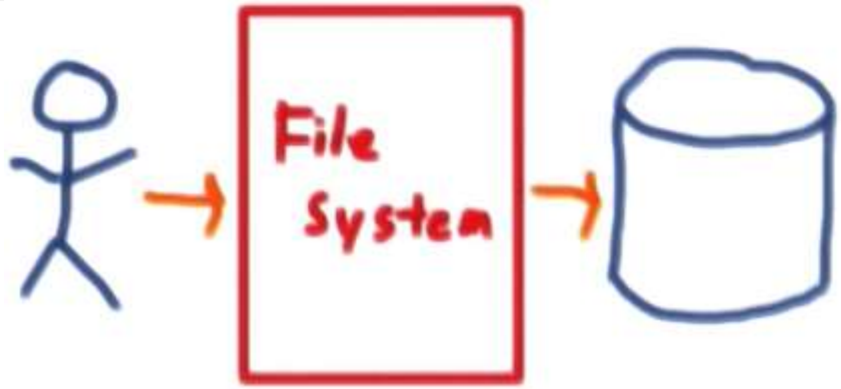
- OS2021
- OS2021_Lab6
- Slides
- Dropbox
- 0514
- COPY0419
- COVIDProposal20210528
- Group-Meeting
- ICDCS20210113
- Neurocomputing20210527
- NSFC20210308
- OS2021
- Trading研究
- VirusContainment20200731
- Zaobo
- 科研论文
- 闲散文件
- 重要
- 此电脑

名称	修改日期	类型
1-OS-Introduction-20210303.ppt	2021/3/3 12:37	Microsoft Power...
2-OS-Process-20210305-0310.ppt	2021/3/10 12:10	Microsoft Power...
3-OS-Bufferoverflow-20210310.pptx	2021/3/10 12:04	Microsoft Power...
4-OS-Process Synchronization-20210...	2021/3/31 11:40	Microsoft Power...
5-IPC in Client-Server Systems-20210...	2021/4/14 9:48	Microsoft Power...
6-CPU Scheduling-20210414.ppt	2021/4/14 12:50	Microsoft Power...
7-Deadlock-20210421.ppt	2021/4/21 11:56	Microsoft Power...
8-Deadlock-Extended-20210428.ppt	2021/4/28 12:38	Microsoft Power...
9-Threads-20210508.ppt	2021/5/8 12:24	Microsoft Power...
10-MainMemory-20210512-0519.ppt	2021/5/28 0:21	Microsoft Power...
11-Mass-Storage Systems-20210602....	2021/6/2 11:08	Microsoft Power...
Review_for Exam1.pptx	2021/3/31 11:21	Microsoft Power...

12 个项目

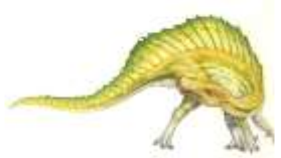
Key Abstractions

1. File
2. Filename
3. Directory Tree

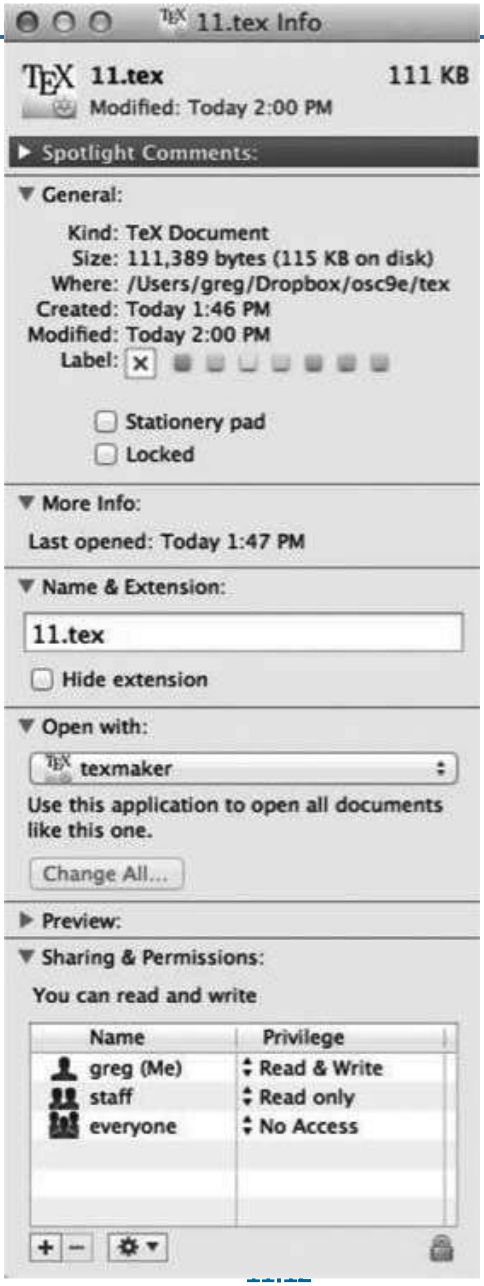


File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – access-control information determines who can do reading, writing, executing, .etc
- **Time, date, and user identification** – data for protection, security, and usage monitoring



File info Window on Mac OS X


11.tex Info

11.tex 111 KB
Modified: Today 2:00 PM

▶ Spotlight Comments:

▼ General:

Kind: TeX Document
Size: 111,389 bytes (115 KB on disk)
Where: /Users/greg/Dropbox/osc9e/tex
Created: Today 1:46 PM
Modified: Today 2:00 PM
Label:

☐ Stationery pad
☐ Locked

▼ More Info:

Last opened: Today 1:47 PM

▼ Name & Extension:

11.tex

☐ Hide extension

▼ Open with:

texmaker

Use this application to open all documents like this one.

Change All...

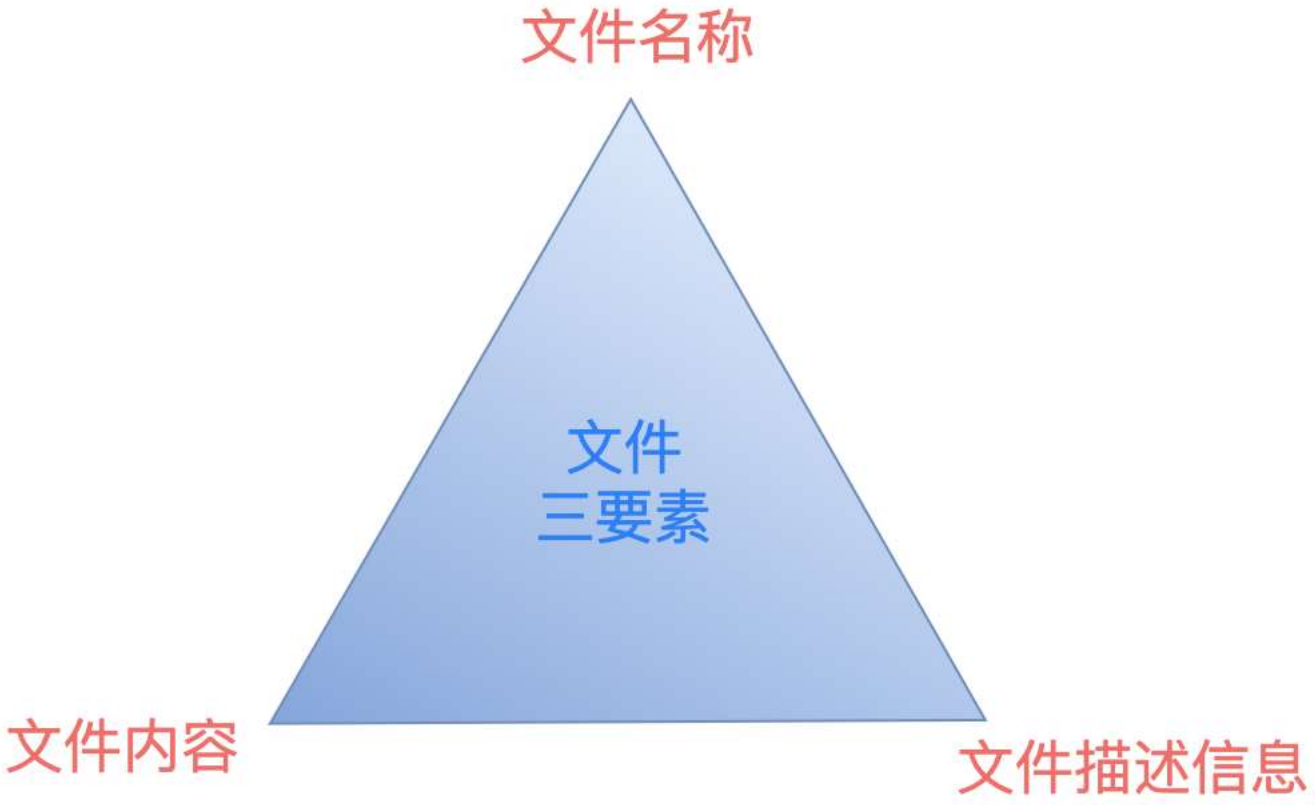
▶ Preview:

▼ Sharing & Permissions:

You can read and write

Name	Privilege
greg (Me)	Read & Write
staff	Read only
everyone	No Access

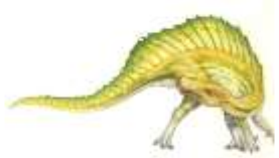
File System of Linux



File System of Linux

- The Linux file system assigns **two** data structures to each file:
  - **index nodes (inode)** and **directory entries**,
  - which are used to record the file's **meta-information** and **directory hierarchy**.

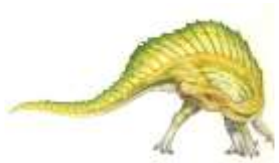
- **Index node (inode)**
  - **meta-information**, such as inode number, file size, access permission, creation time, modification time, location on the disk, etc.
  - Inode is the unique identifier of a file.
  - They correspond to each other one by one and are also stored on hard disks, so inodes also occupy disk space



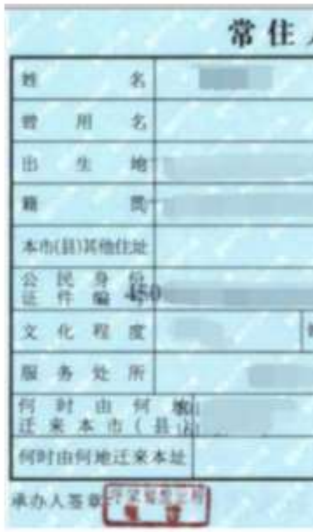
File System of Linux

- The Linux file system assigns **two** data structures to each file:
  - **index nodes (inode)** and **directory entries (dentry)**,
  - which are used to record the file's **meta-information** and **directory hierarchy**.

- **Directory entries (dentry)**
  - Record the name of a file, index node pointers, and hierarchical relationships with other directory entries.
  - When multiple directory entries are associated, a **directory** is formed, but unlike an index node, a directory entry is a data structure maintained by the kernel and is not stored on disk, but cached in memory

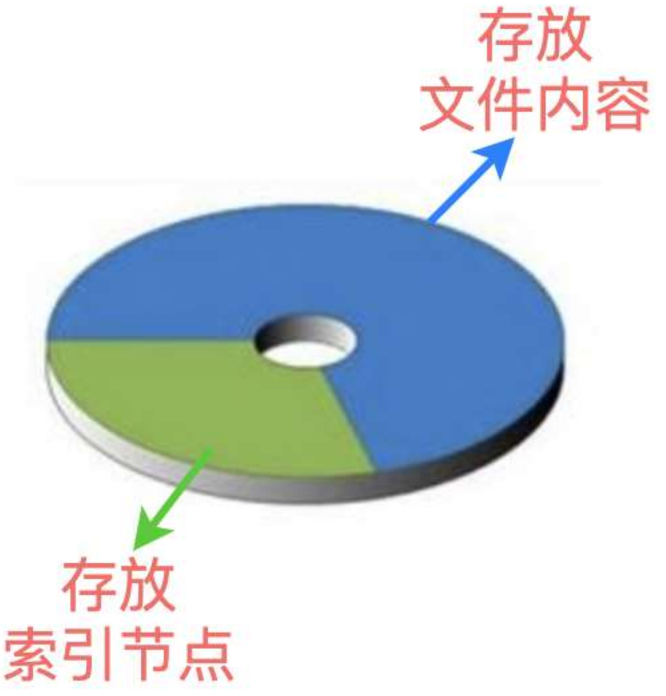


File System of Linux



常住人

姓　　名	[redacted]
曾 用 名	
出 生 地	[redacted]
籍　　贯	[redacted]
本市(县)其他住址	
公民身份证件编号	450[redacted]
文 化 程 度	[redacted]
服 务 处 所	[redacted]
何时由何地迁来本市(县)	[redacted]
何时由何地迁来本址	
承办人签章	

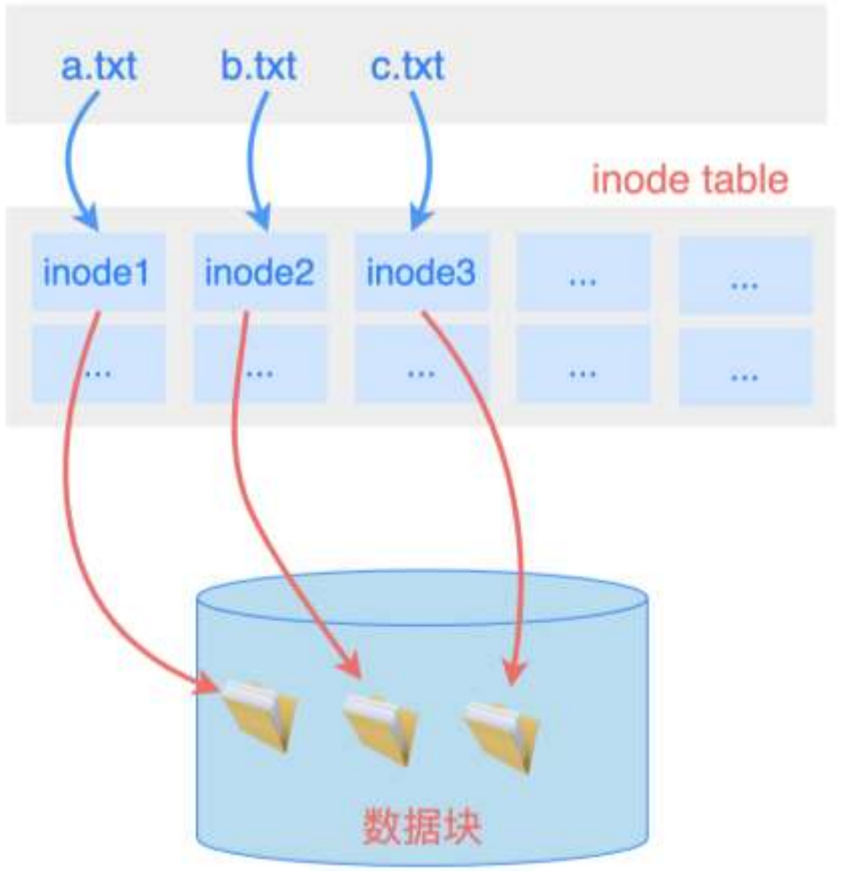


文件索引节点

类似
文件内容

File System of Linux

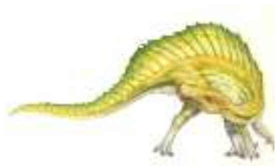
When we call the open file API function, the operating system first finds the inode of the file according to the incoming file path, and then carries out a series of permission check operations, and finally obtains the content of the file from the inode in which blocks (blocks), so that the content of the file can be read and written



Relationship between inode and dentry

- Since **inode** uniquely identifies a file, and **dentry** records the name of the file, the relationship between dentry and inode is **many-to-one**,

- That is, a file can have multiple different names. For example, a hard link is implemented so that inode in multiple directory entries points to the same file



Relationship between dentry and directory

- Note that **directories are also files** and are uniquely identified by inodes

- Unlike ordinary files, which store file data on disk, while directory stores subdirectories or files on disk.

- Are directory entries and directories the same thing?
  - They are not the same thing.
  - A directory is a file that is persisted on disk, and a directory entry is a kernel data structure that is cached in memory.

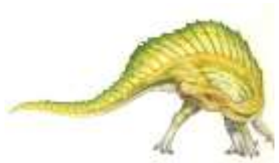
- If the directory is frequently read from the disk, the efficiency will be very low.
- So the kernel caches the read directories in memory using a data structure called **directory entries**
- The next time the kernel reads the same directory, it only needs to read it from memory, which greatly improves the efficiency of the file system

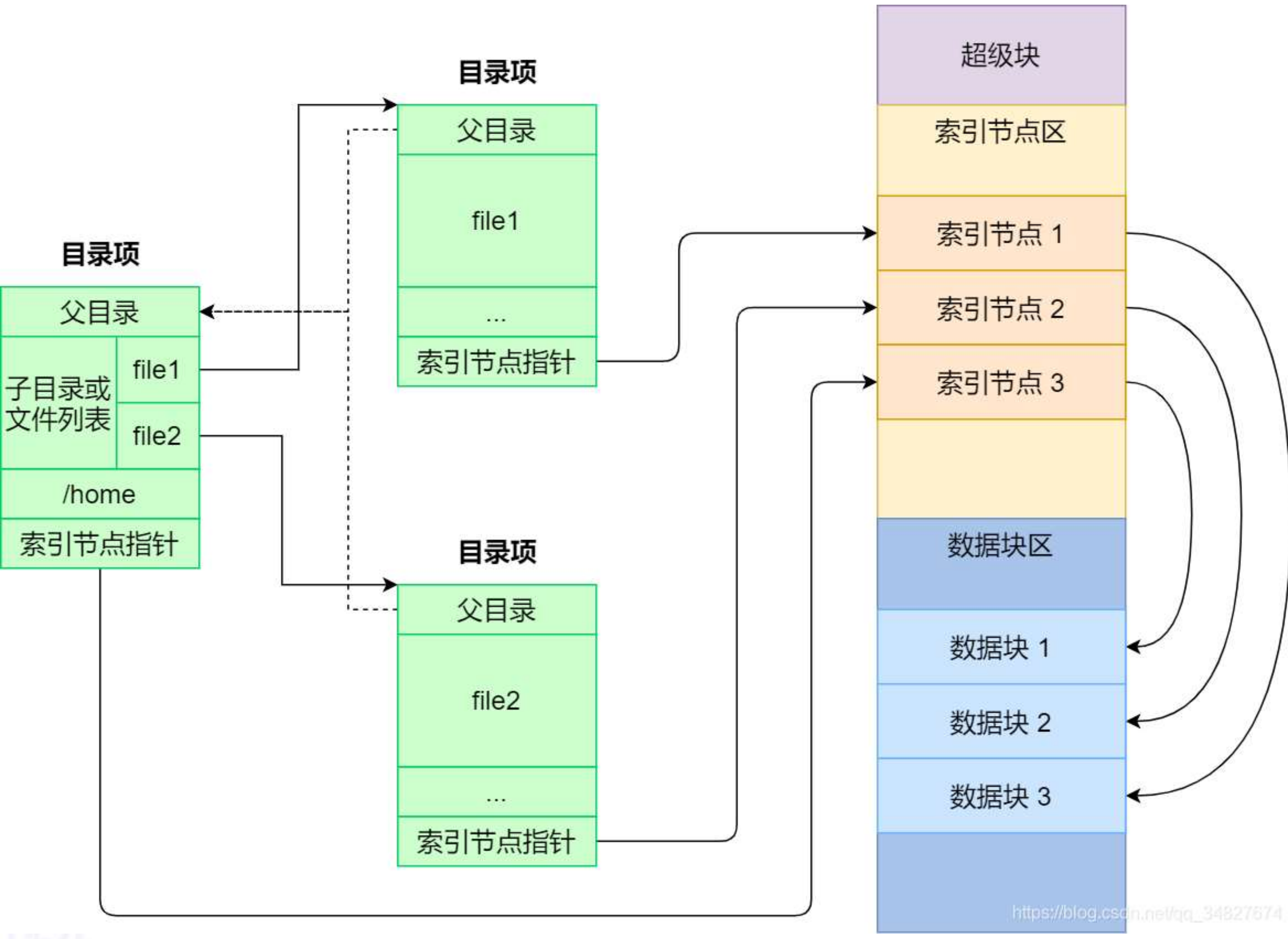
How is file data stored on disk?

- The smallest unit of disk read and write is the **sector**, the size of the sector is only 512B, obviously, if each read and write in such a small unit, the efficiency of the read and write will be very low.

- Therefore, the file system organizes multiple sectors into a **logical block**, the smallest unit of each read and write is a logical block (data block)

- The logical block size in Linux is 4KB, that is, read and write 8 sectors at a time, which will greatly improve the efficiency of disk read and write.




目录项
父目录
子目录或文件列表
file1
file2
/home
索引节点指针
目录项
父目录
file1
...
索引节点指针
目录项
父目录
file2
...
索引节点指针
超级块
索引节点区
索引节点 1
索引节点 2
索引节点 3
数据块区
数据块 1
数据块 2
数据块 3

How is file data stored on disk?

- Inodes are data stored on the hard disk, so in order to speed up file access, inodes are usually loaded into memory.

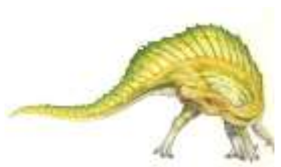
- When the disk is formatted, it is divided into three storage areas
  - The **superblock**, the **inode area**, and the **data block area**.
  - Superblock stores detailed information about the file system, such as the number of blocks, block size, and free blocks.
  - Inode area, used to store inodes;
  - Block area, used to store file or directory data;

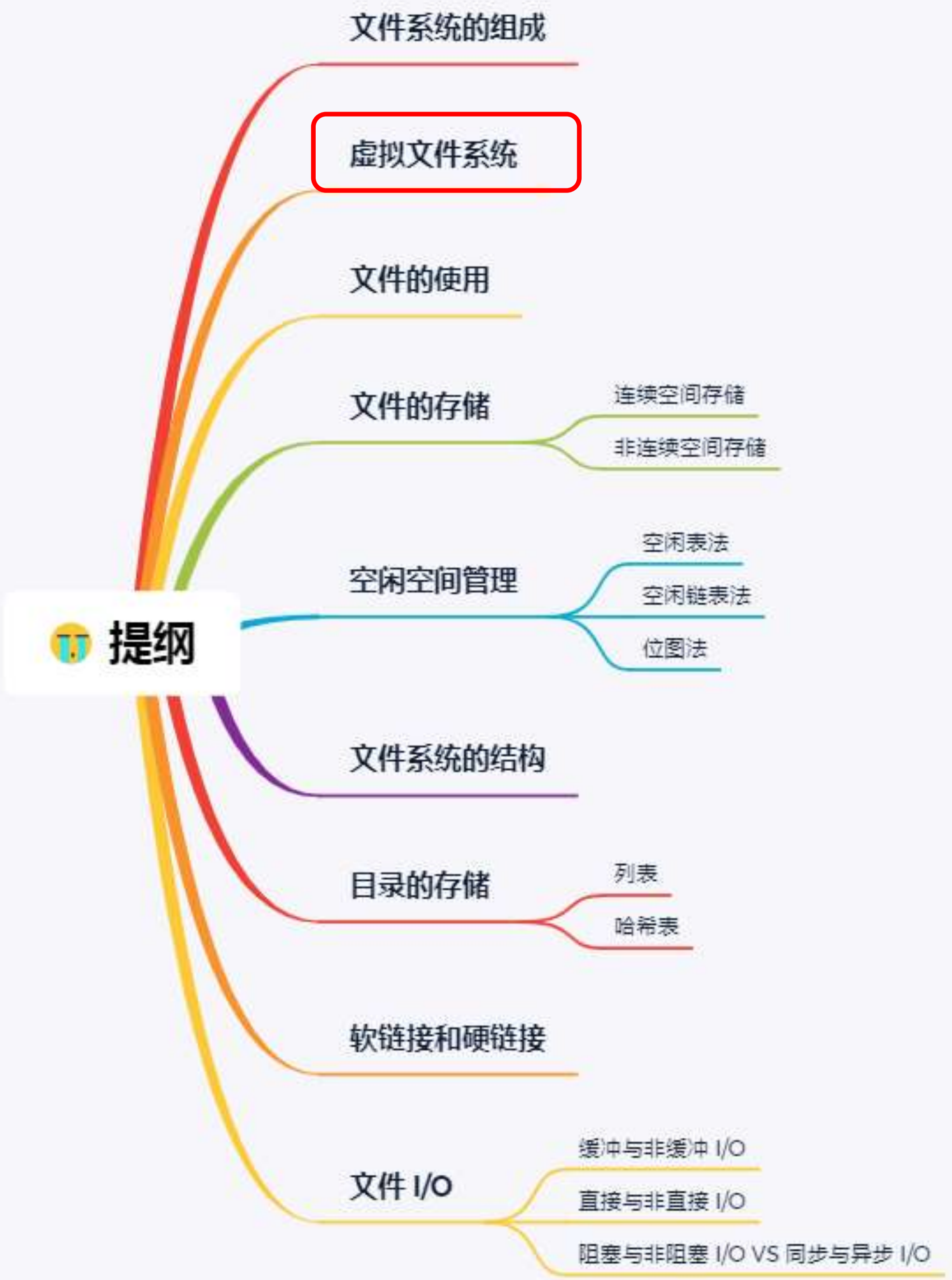
How is file data stored on disk?

- It is impossible to load all of the superblocks and inodes into memory, so the memory will not hold up, so they are only loaded into memory when they need to be used, and the timing of their loading into memory is different:

- Superblock: enters the memory when the file system is mounted.

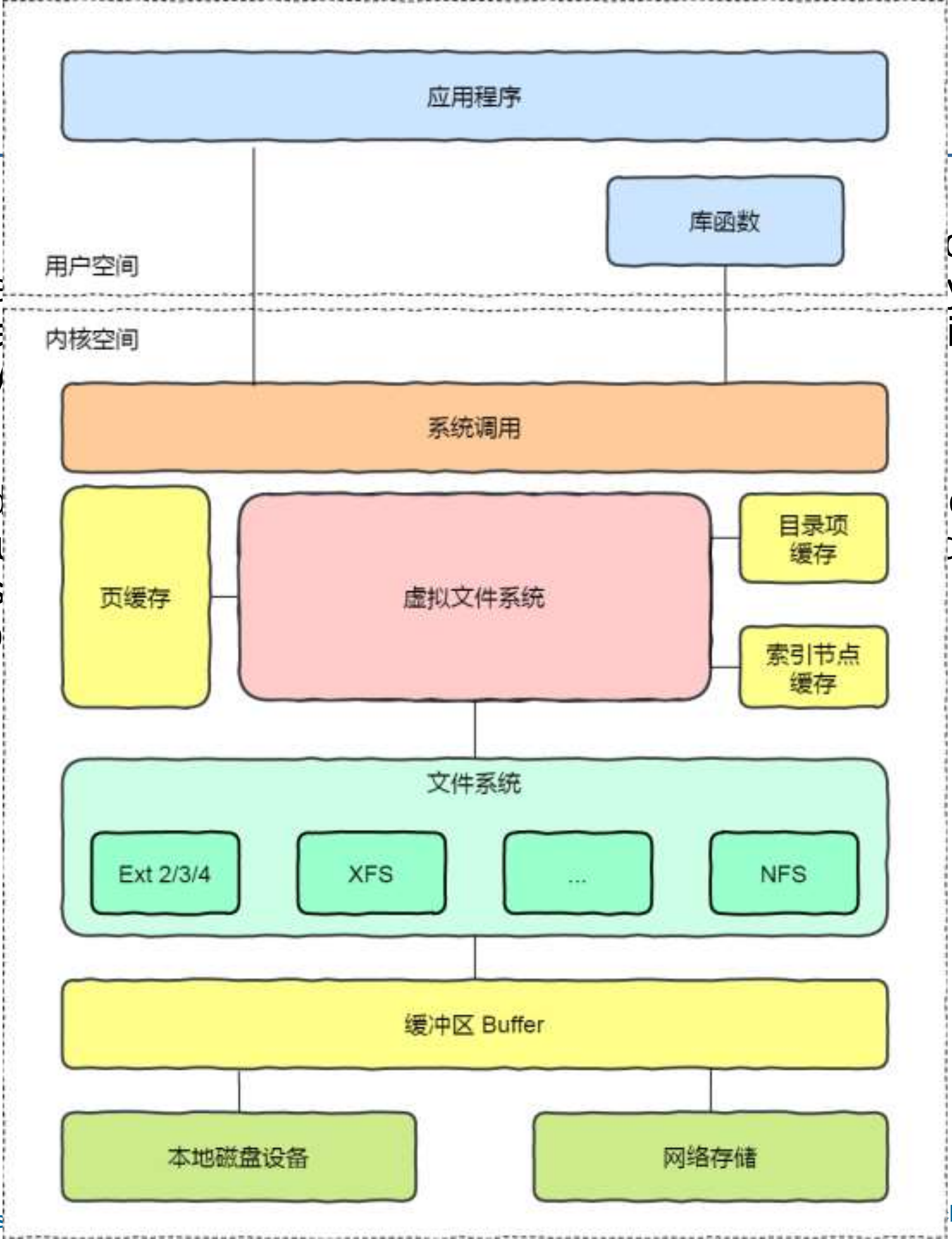
- Inode area: enters memory when a file is accessed;

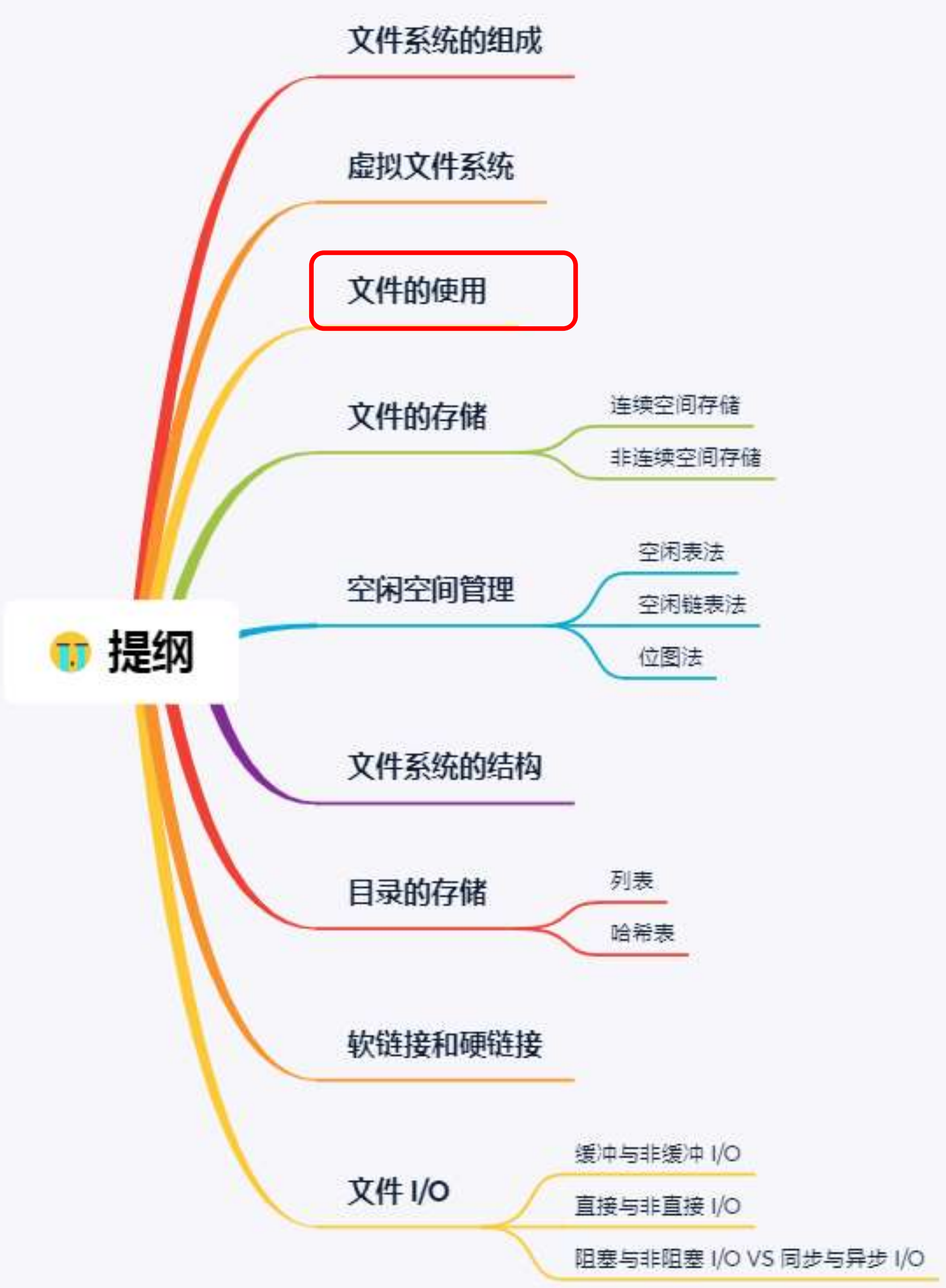



提纲
文件系统的组成
虚拟文件系统
文件的使用
文件的存储
连续空间存储
非连续空间存储
空闲空间管理
空闲表法
空闲链表法
位图法
文件系统的结构
目录的存储
列表
哈希表
软链接和硬链接
文件 I/O
缓冲与非缓冲 I/O
直接与非直接 I/O
阻塞与非阻塞 I/O VS 同步与异步 I/O

- There are ovide a unified inte yer between the user la irtual File System (V

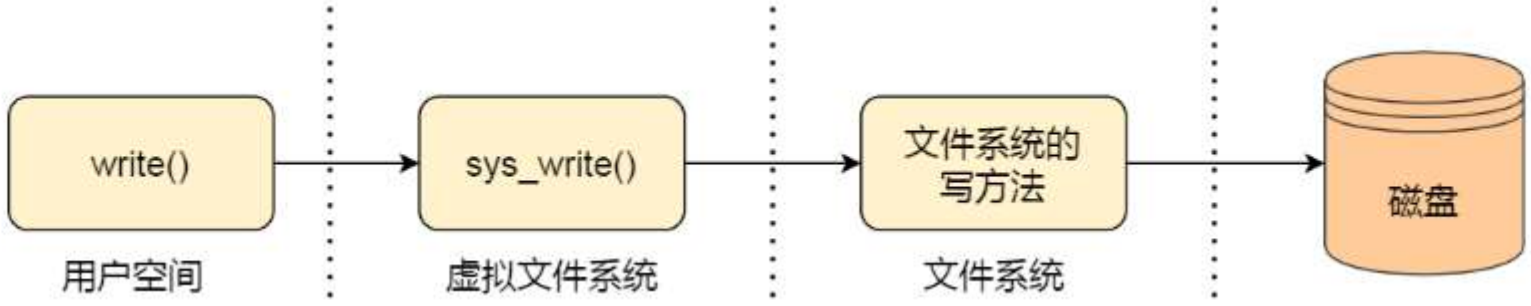
- The VFS d es that all file systems su derstand how the file sys interface provided b

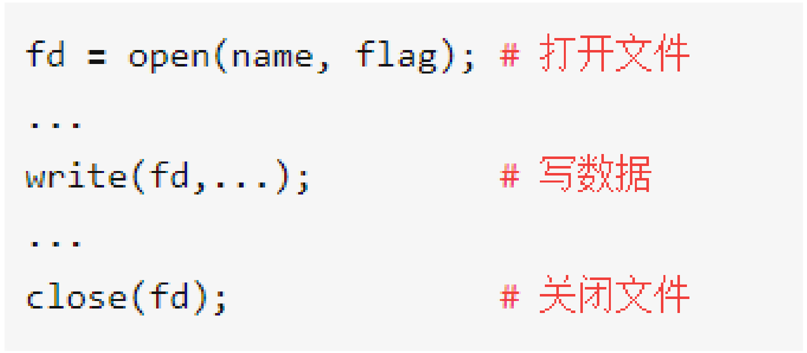



提纲
文件系统的组成
虚拟文件系统
文件的使用
文件的存储
连续空间存储
非连续空间存储
空闲空间管理
空闲表法
空闲链表法
位图法
文件系统的结构
目录的存储
列表
哈希表
软链接和硬链接
文件 I/O
缓冲与非缓冲 I/O
直接与非直接 I/O
阻塞与非阻塞 I/O VS 同步与异步 I/O

Use of file

- From the user's point of view, how do we use the file? First, we need to open a file through a **system call**

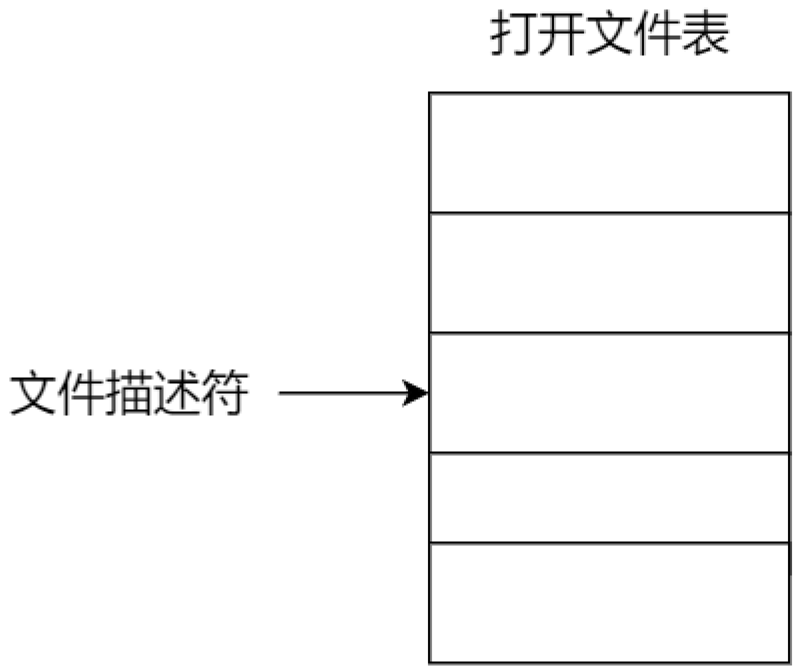




```
fd = open(name, flag); # 打开文件
...
write(fd,...);         # 写数据
...
close(fd);             # 关闭文件
```

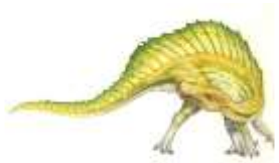
Use of file

- After opening a file, the operating system will track all the files opened by the process,
  - the so-called tracking is that the operating system maintains an open file table for each process, and each item in the file table represents the "file descriptor", so the file descriptor is the identification of the open file.



Use of file

- The OS maintains the status and information of the opened file in the open file table
  - **File pointer**: the system keeps track of the last read/write location as a pointer to the current file location. This pointer is unique to a certain process that opened the file.
  - **File open counter**: tracks the number of open and closed files. When the count is 0, the system closes the file, delete the entry
  - **Disk location**: The vast majority of file operations require the system to modify file data, the information is saved in memory, so that each operation does not read from the disk
  - **Access**: each process needs to have an access mode to open files (create read-only read/write add, etc.), and this information is saved in the process's open file table so that the OS can allow or deny subsequent I/O requests



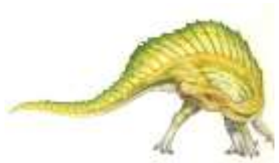
Use of file

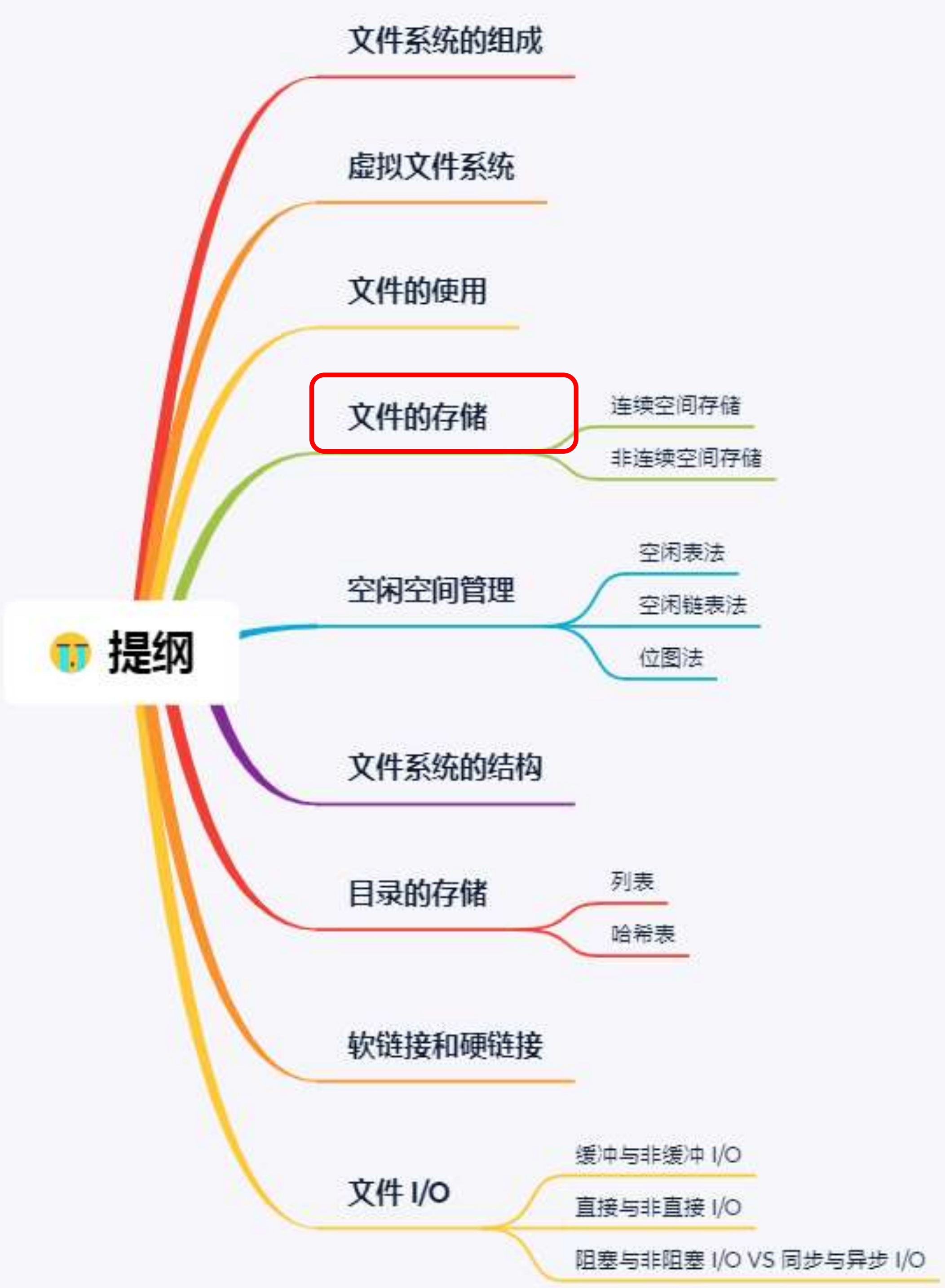
- In the user's view, a file is a persistent data structure, but the operating system doesn't care about any data structure you want to store on disk, and the operating system's view is how the file data corresponds to the disk block.

- Therefore, the user and the operating system to read and write the file is different, the user is used to read and write the file **in bytes**, and the operating system is to read and write the file **in blocks** of data, then shield this difference of work is the file system.

Use of file

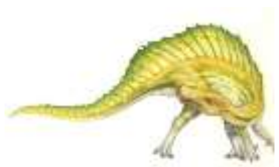
- When a user process reads 1 byte of data from a file, the file system obtains the data block where the byte resides and returns the data part required by the user process corresponding to the data block.
- When a user process writes one-byte data into a file, the file system finds the location of the data block to be written, modifies the corresponding part of the data block, and writes the data block back to the disk.
- Therefore, the basic unit of operation of a file system is **a data block**.




提纲
文件系统的组成
虚拟文件系统
文件的使用
文件的存储
连续空间存储
非连续空间存储
空闲空间管理
空闲表法
空闲链表法
位图法
文件系统的结构
目录的存储
列表
哈希表
软链接和硬链接
文件 I/O
缓冲与非缓冲 I/O
直接与非直接 I/O
阻塞与非阻塞 I/O VS 同步与异步 I/O

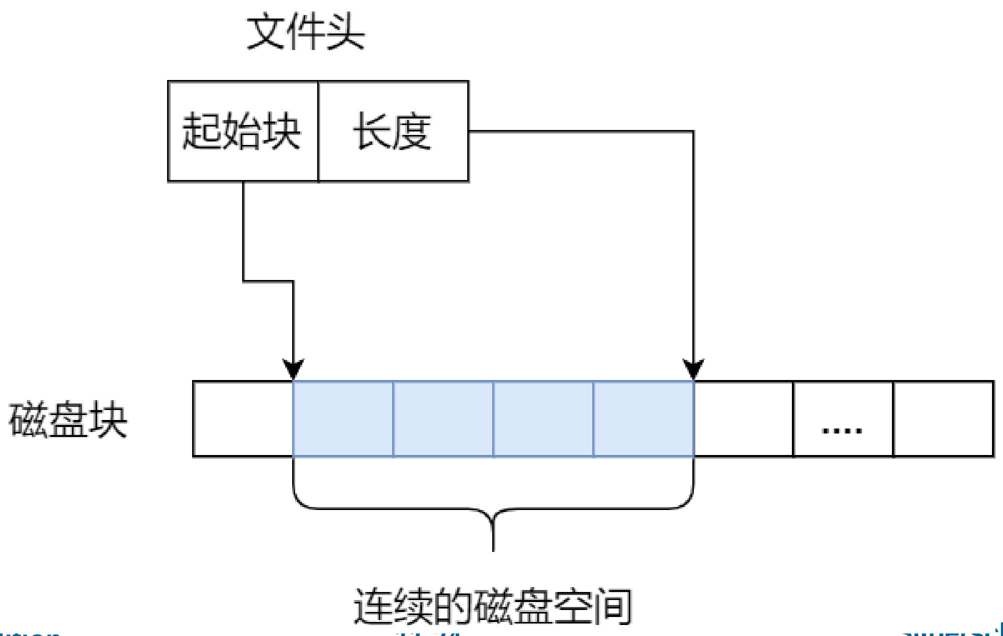
Physical Storage Allocation

- There are two main ways a file is physically stored in memory:
  - Contiguous Storage
  - Non-contiguous Storage
    - Indexed Storage
    - Linked list



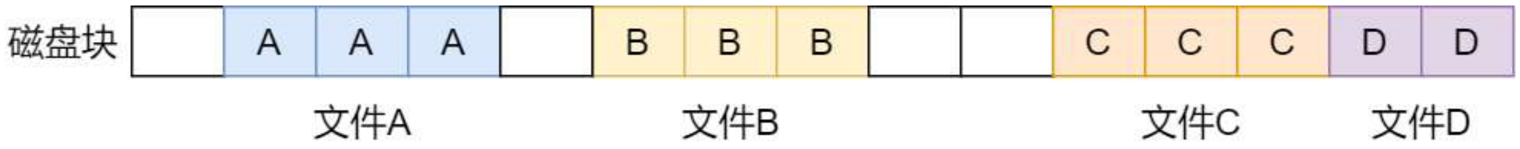
Contiguous Storage

- Contiguous Storage
  - Files are stored in "contiguous" physical space on the disk.
  - The data of the file is closely connected, and the reading and writing efficiency is high, because the entire file can be read at one disk seek
  - The premise is that the size of a file must be known first, so that the file system can find a continuous space on the disk to allocate to the file according to the size of the file



Contiguous Storage

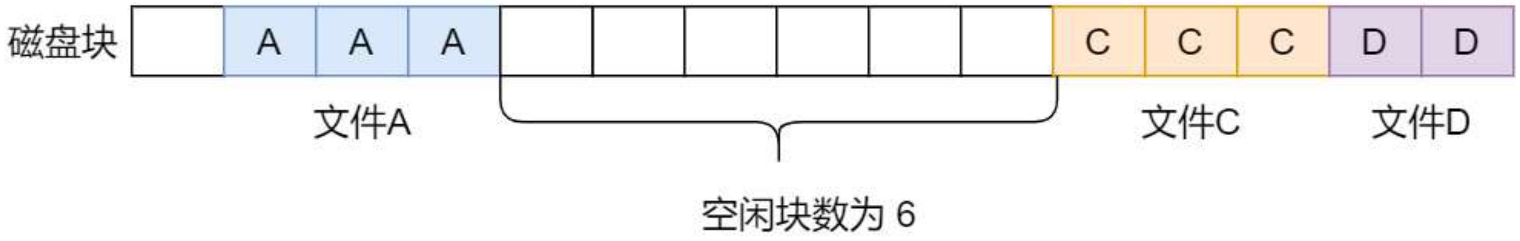
- Although the continuous storage mode has high read and write efficiency, it has the defects of "**disk fragmentation**" and "**file length is not easy to expand**"



假设文件 B 被删除了，此时最大空闲块数是 6

如果新来的文件所需的块数 <= 6，这时新的文件就可以被正常存放

如果新来的文件所需的块数 > 6，这时新的文件就无法存放到磁盘



磁盘碎片的缺陷

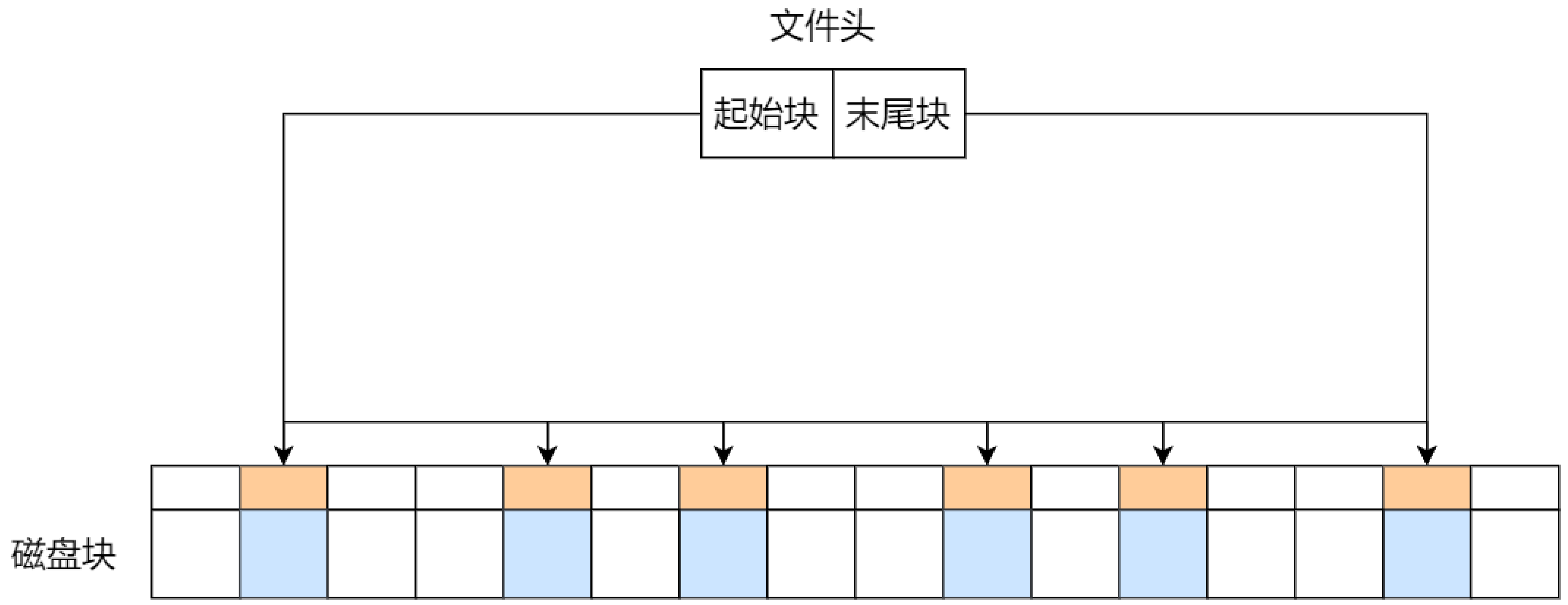
Contiguous Storage

- Another defect is that the file length expansion is not convenient, for example, file A in the figure above to expand, need more disk space, the only way is to move the way, also said earlier, this way is very low efficiency.

Non-contiguous Storage

- Also called **linked-storage**
- Non-contiguous Storage means that records of a file are stored wherever there is free space.
- The file manager will try to put as much of it together as possible, but there will be other part spread out over the disk.
- This means there is no easy way to determine the exact location of a record in a file.

Non-contiguous Storage

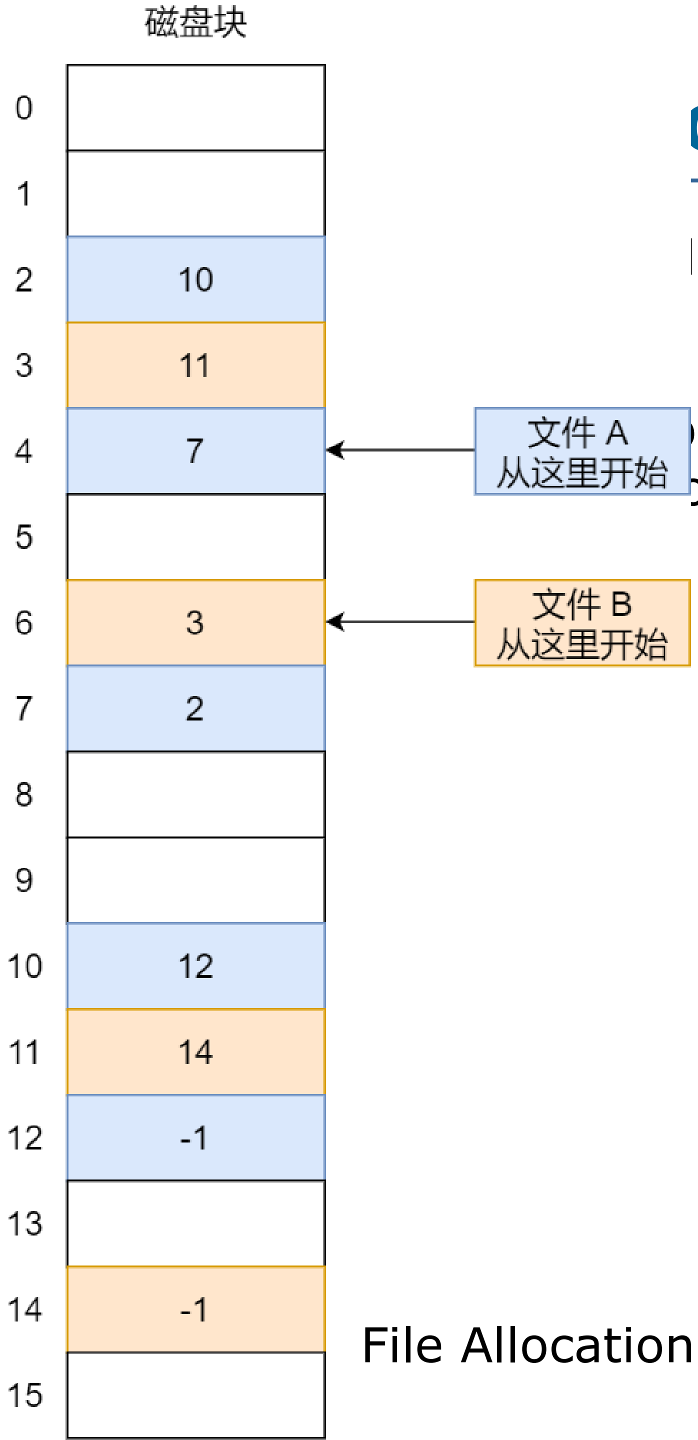


The stability is poor. When the pointer in the linked list is lost or damaged due to software or hardware errors during the running of the system, resulting in the loss of file data.

ous Storage

Solutic lisk block and put it in a table in memoı

The im licit link", which refers to the pointeı of the file, explicitly stored in a linked



File Allocation Table，FAT

Non-contiguous Storage

Because the process of finding records is carried out **in memory**, it not only significantly improves the retrieval speed, but also greatly reduces the number of times the disk is accessed.

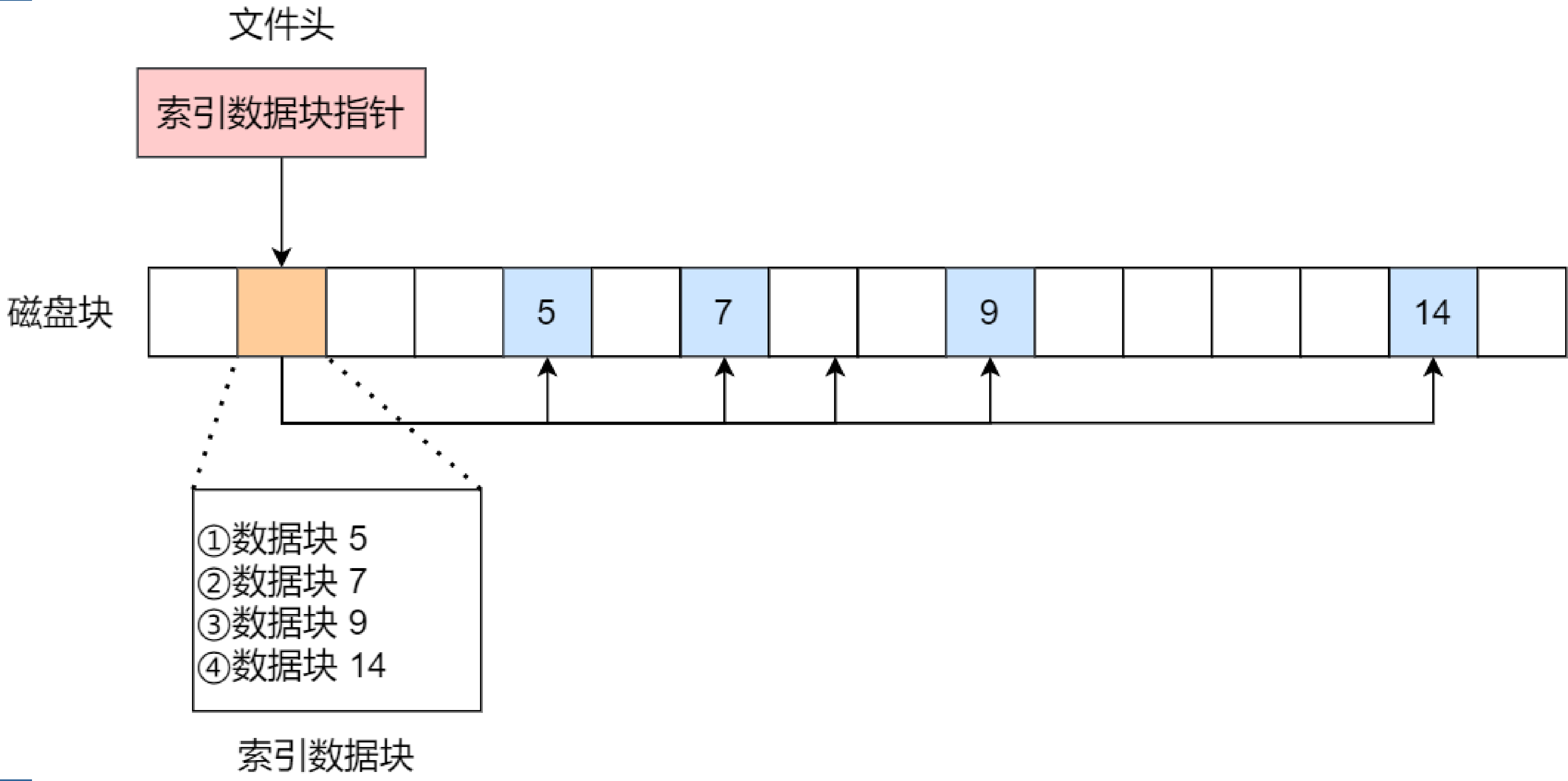
But it is precisely the relationship in which the entire table is stored in memory that its main disadvantage is that **it does not apply to large disks.**

Indexed Storage

- The linked list solves the problem of continuously allocated disk fragmentation and dynamic file expansion, but it does not support **direct access** effectively

- Indexed storage means that as well as the records in the file, an index block is created, with pointers to each individual file.

Indexed Storage



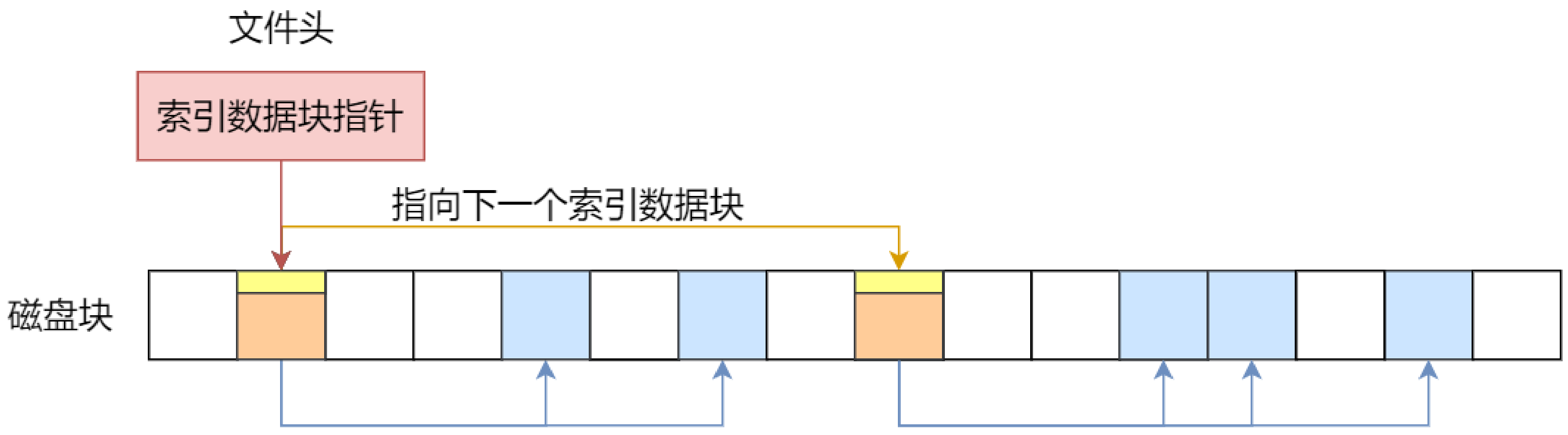
①数据块 5
②数据块 7
③数据块 9
④数据块 14

Indexed Storage

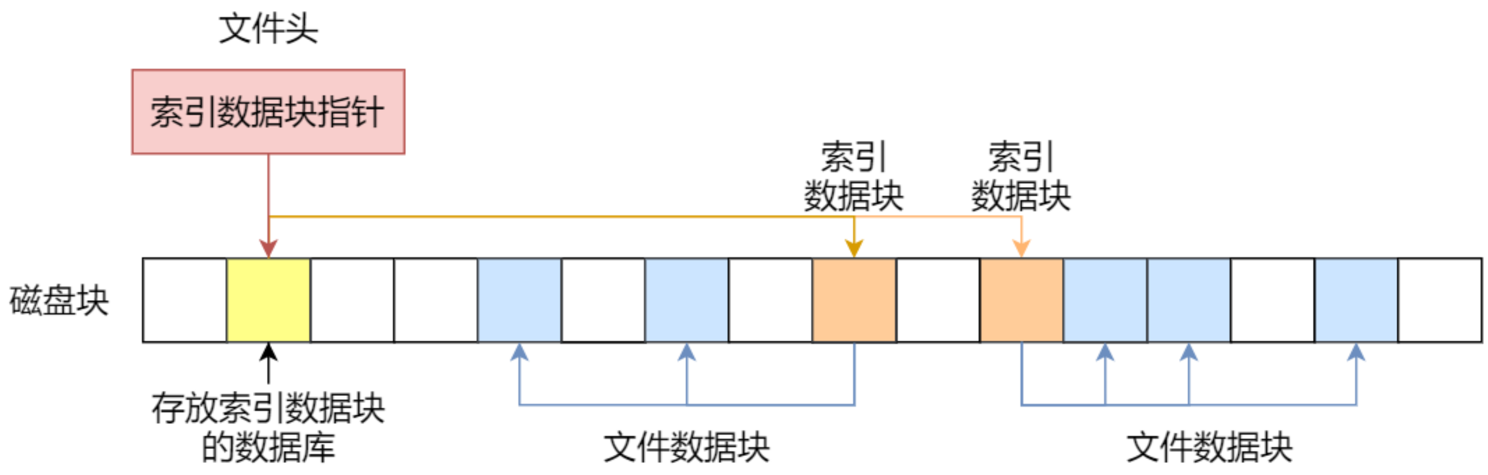
It is convenient to create, enlarge and shrink files; - There will be no fragment problem; - Supports sequential read and write and random read and write

Since the index data is also stored in the disk, if the file is small, it can be stored only one block, but still need to allocate an additional block to store the index data, so one of the defects is the cost of storing the index.

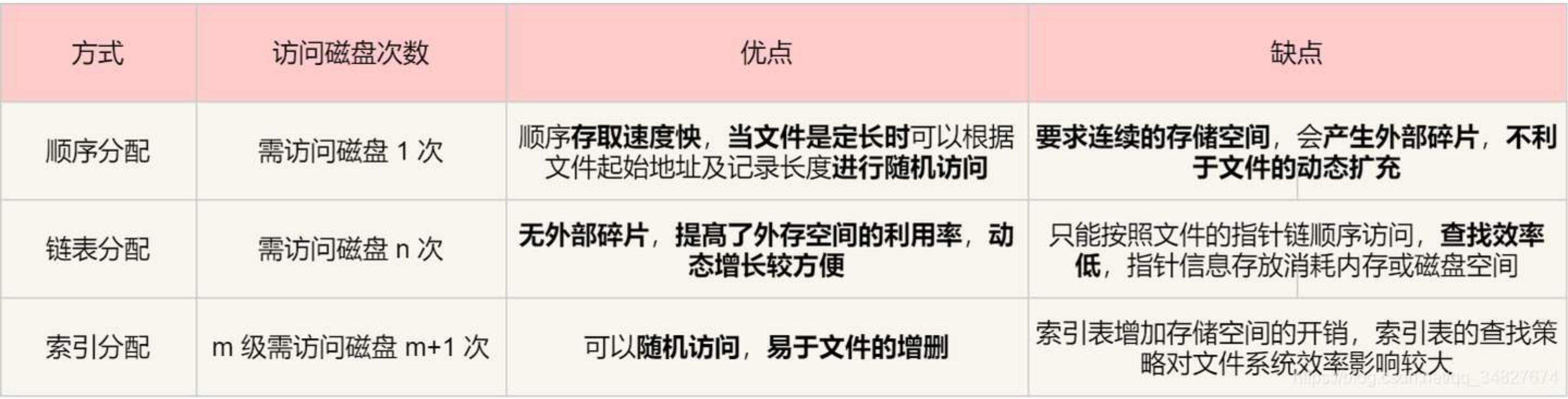
If the file is large, so large that an index data block can not put the index information, then how to deal with the storage of large files?



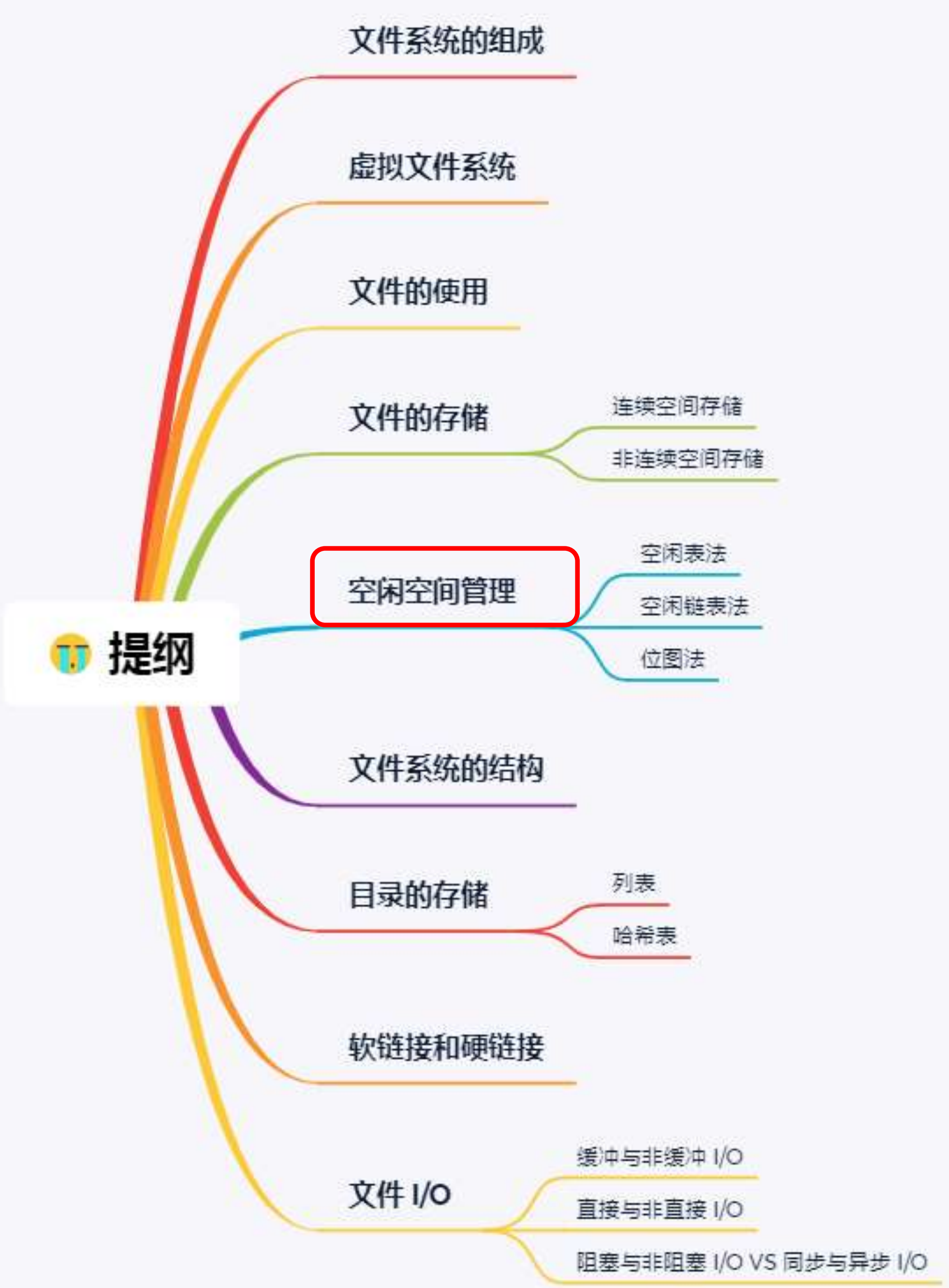
链表 + 索引



多级索引块

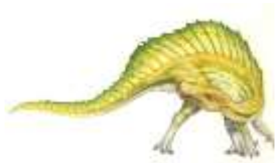


方式	访问磁盘次数	优点	缺点
顺序分配	需访问磁盘 1 次	顺序**存取速度快**，**当文件是定长时**可以根据文件起始地址及记录长度**进行随机访问**	**要求连续的存储空间**，会**产生外部碎片**，**不利于文件的动态扩充**
链表分配	需访问磁盘 n 次	**无外部碎片**，**提高了外存空间的利用率**，**动态增长较方便**	只能按照文件的指针链顺序访问，**查找效率低**，指针信息存放消耗内存或磁盘空间
索引分配	m 级需访问磁盘 m+1 次	可以**随机访问**，**易于文件的增删**	索引表增加存储空间的开销，索引表的查找策略对文件系统效率影响较大


提纲
文件系统的组成
虚拟文件系统
文件的使用
文件的存储
连续空间存储
非连续空间存储
空闲空间管理
空闲表法
空闲链表法
位图法
文件系统的结构
目录的存储
列表
哈希表
软链接和硬链接
文件 I/O
缓冲与非缓冲 I/O
直接与非直接 I/O
阻塞与非阻塞 I/O VS 同步与异步 I/O

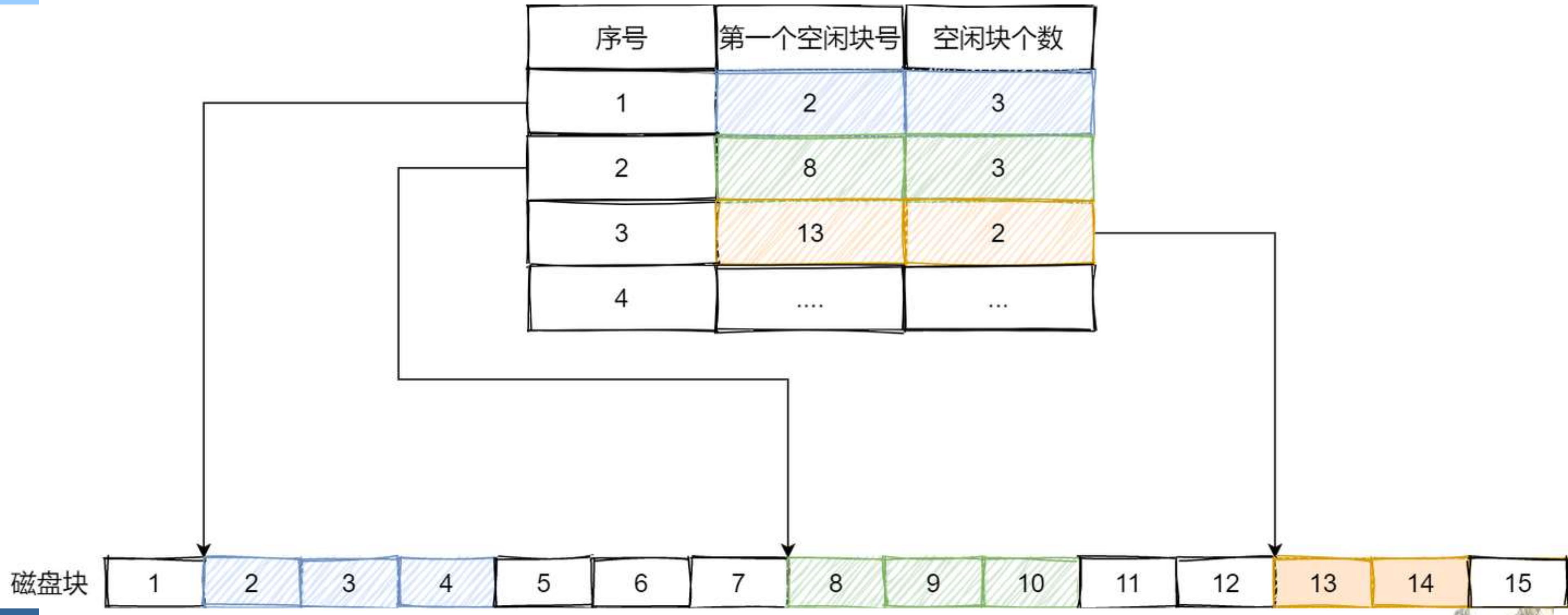
Free space management

- The next question is, if I want to save a data block, where should I put it on the hard disk? Do we need to scan all the blocks and find an empty place to put them?
- Methods
  - Free table method
  - Free linked list method
  - Bitmap method



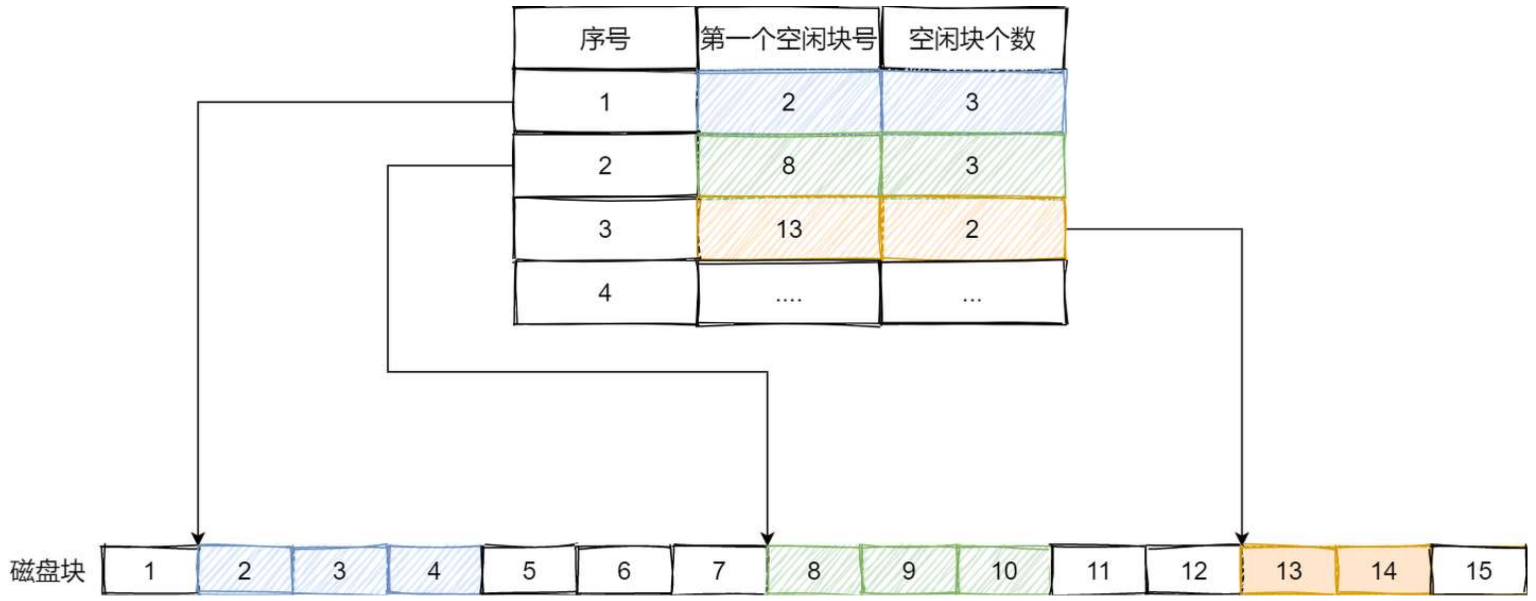
Free table method

- The free table method is to **create a table for all the free space**, including the first block number of the free area and the number of blocks in the free area.
- Note that this method is consecutively allocated



序号	第一个空闲块号	空闲块个数
1	2	3
2	8	3
3	13	2
4	....	...

Free table method

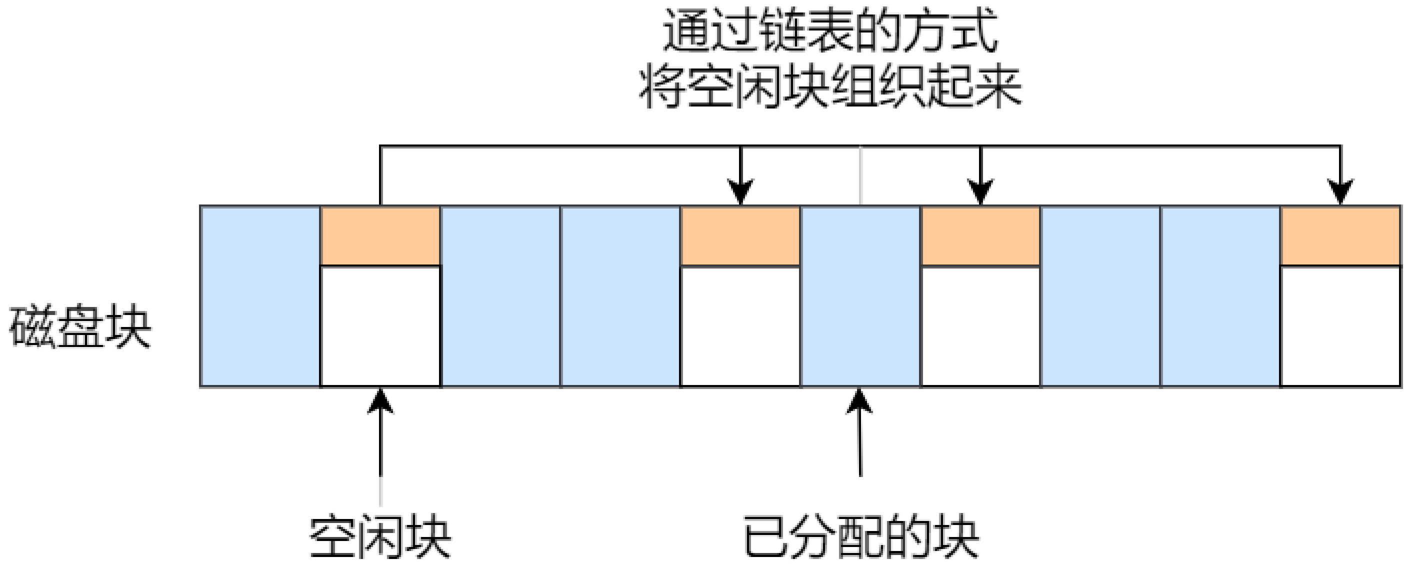


序号	第一个空闲块号	空闲块个数
1	2	3
2	8	3
3	13	2
4	....	...

This method has a good performance only when there is a small amount of free area.

Because, if the storage space has a large number of small free areas, the free table becomes large, so the query efficiency will be very low. In addition, this allocation technique is suitable for creating continuous files.

Free linked list method

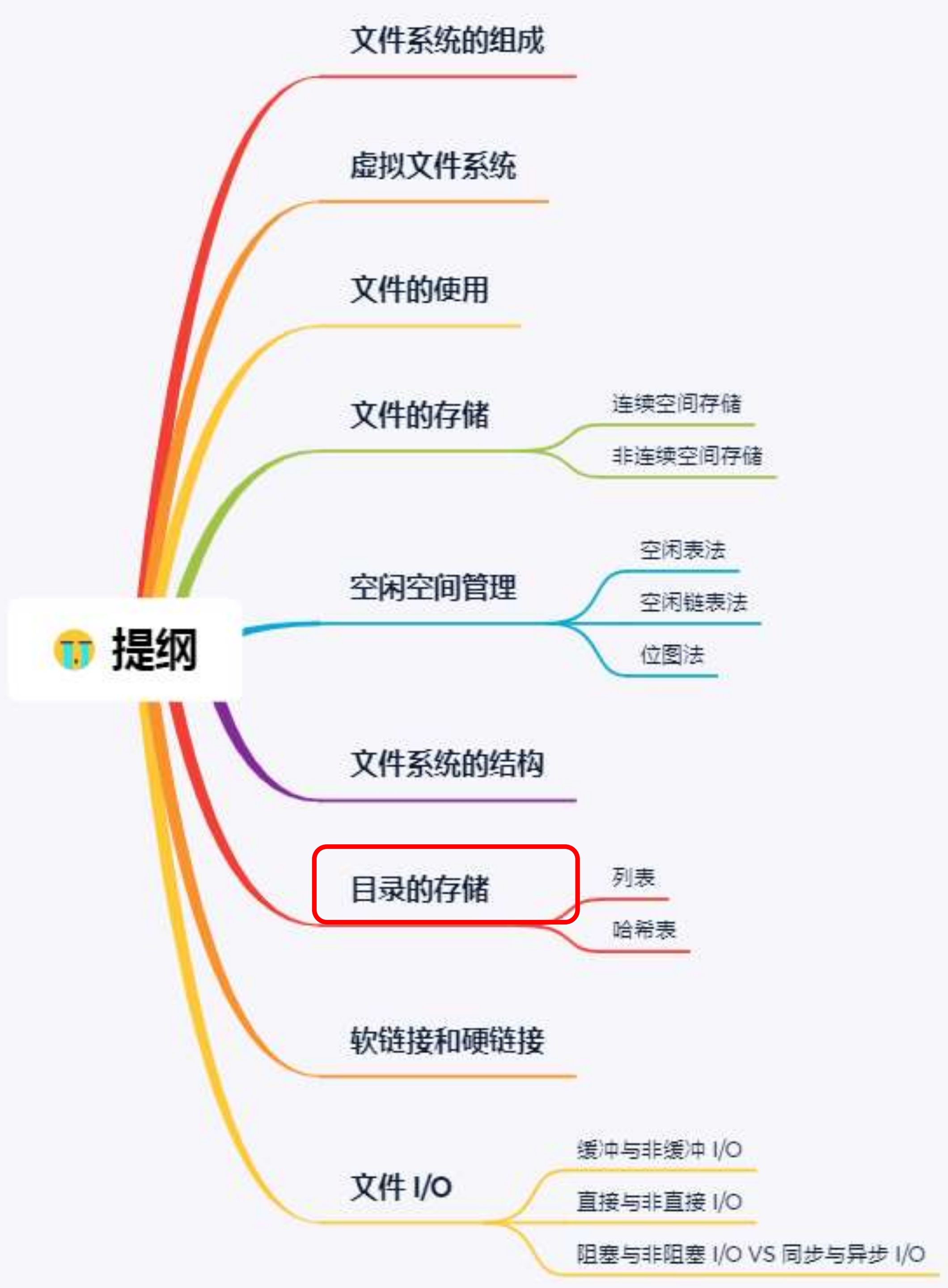


Bitmap method

A bitmap uses a binary bit to represent the usage of a disk block, and all disk blocks on the disk have a binary bit corresponding to it

If the value is 0, the corresponding disk block is free. If the value is 1, the corresponding disk block is allocated

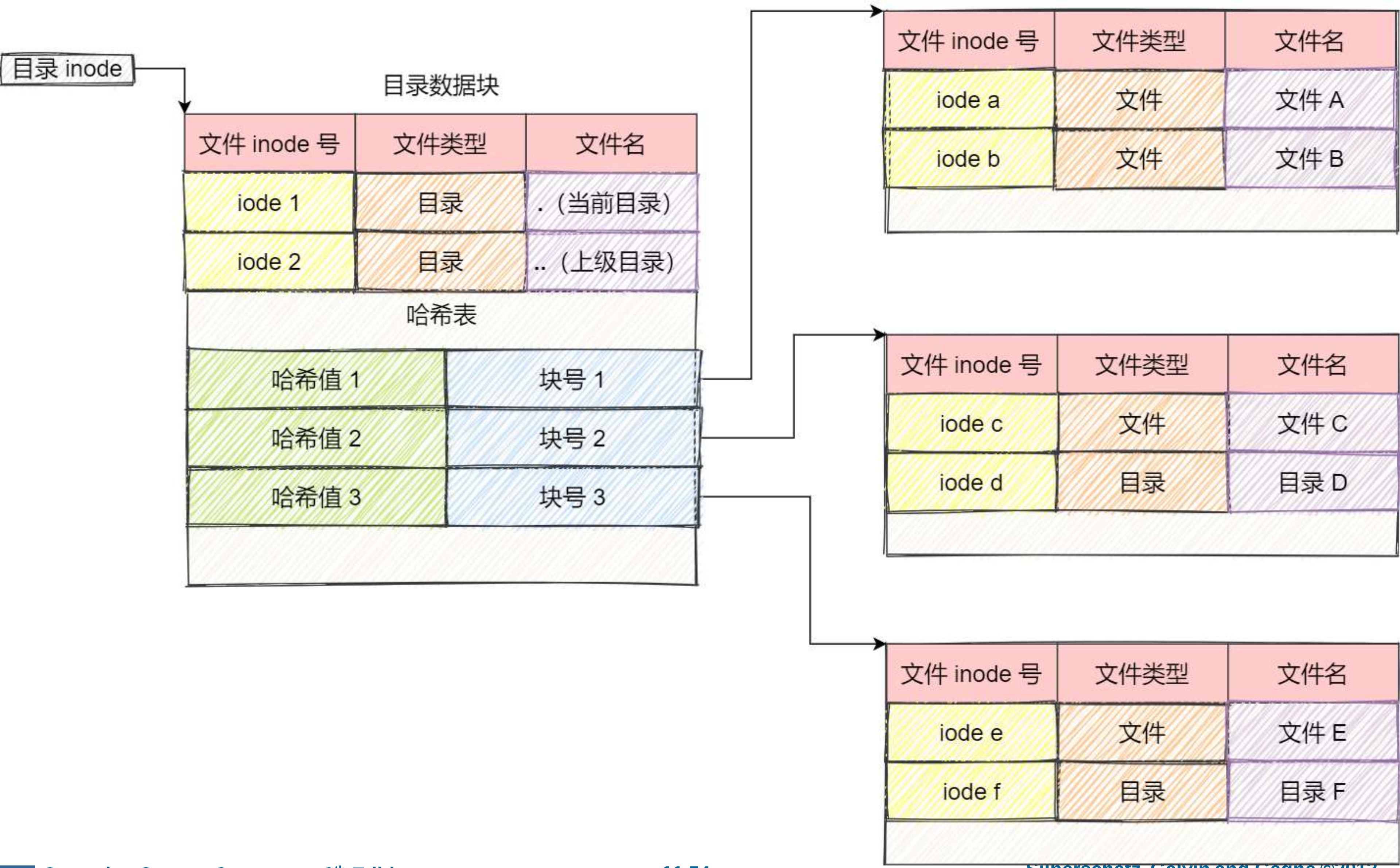
111111001111111000111011011111111100111 ...

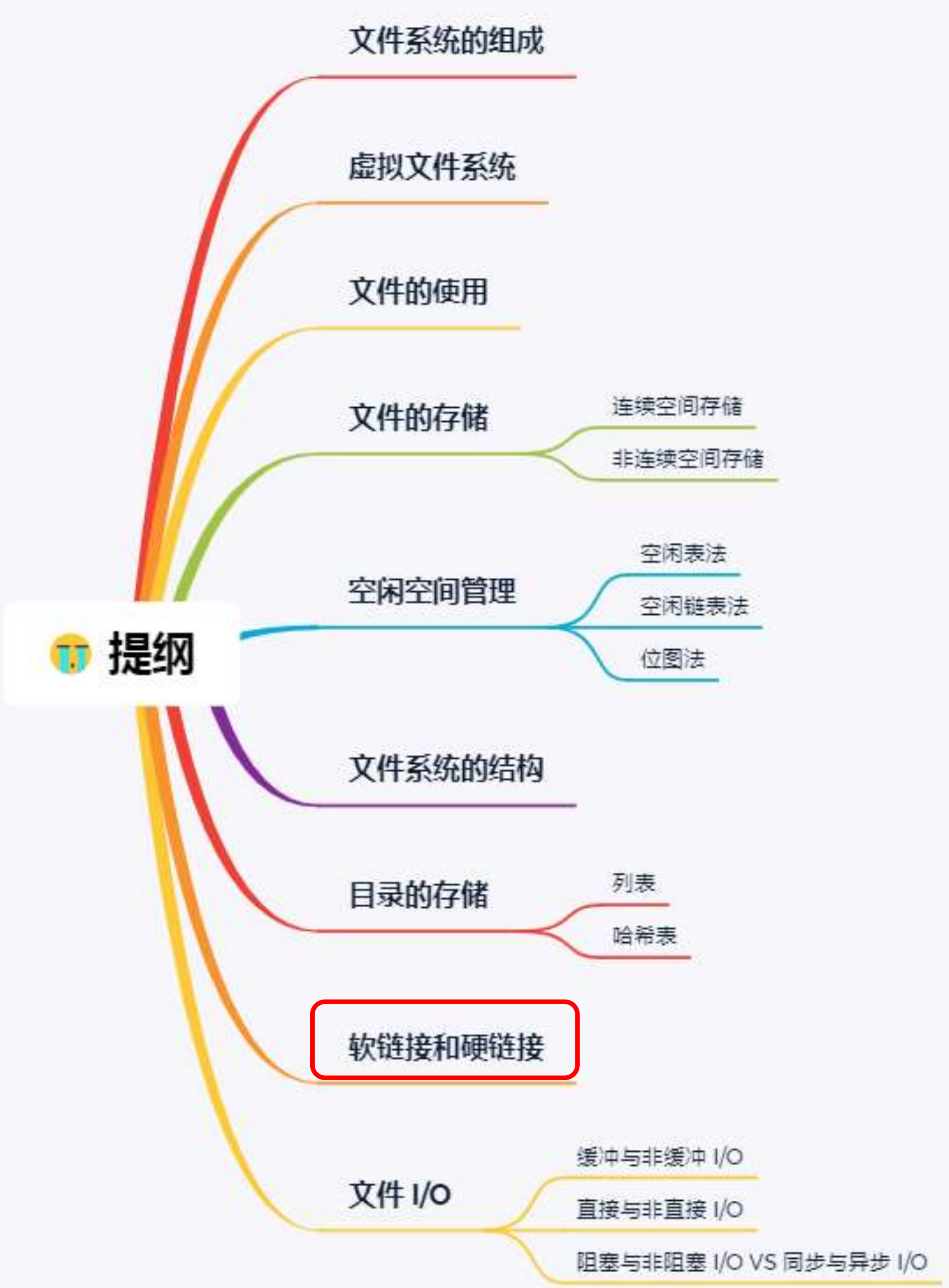

提纲
文件系统的组成
虚拟文件系统
文件的使用
文件的存储
连续空间存储
非连续空间存储
空闲空间管理
空闲表法
空闲链表法
位图法
文件系统的结构
目录的存储
列表
哈希表
软链接和硬链接
文件 I/O
缓冲与非缓冲 I/O
直接与非直接 I/O
阻塞与非阻塞 I/O VS 同步与异步 I/O

Storage of Directory

- We have known how a common file is stored, but there is also a special file, **directory**, how is it saved?
- The block of an ordinary file stores the file data, while the block of a directory file stores the file information of a directory item by item
- In the block of directory files, the simplest saving format is the list, which is to list the file information (such as file name, file inode, file type, etc.) in the directory item by item
- Each item in the list represents the name of the file in that directory and the corresponding inode, through which you can find the real file.

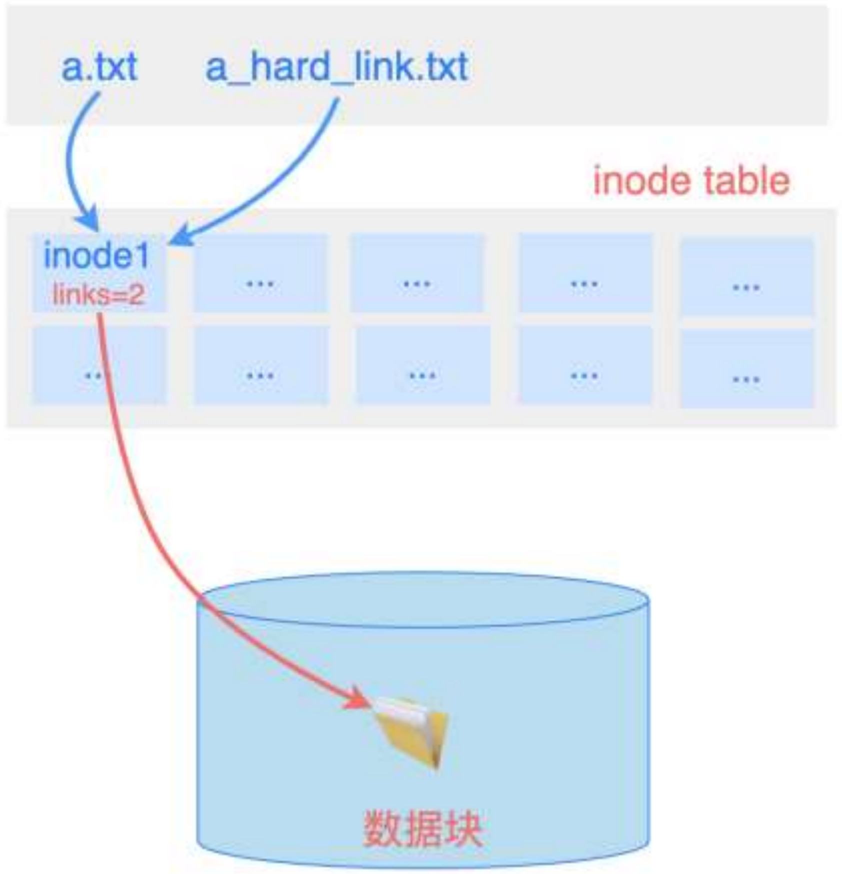
Storage of Directory




提纲
文件系统的组成
虚拟文件系统
文件的使用
文件的存储
连续空间存储
非连续空间存储
空闲空间管理
空闲表法
空闲链表法
位图法
文件系统的结构
目录的存储
列表
哈希表
软链接和硬链接
文件 I/O
缓冲与非缓冲 I/O
直接与非直接 I/O
阻塞与非阻塞 I/O VS 同步与异步 I/O

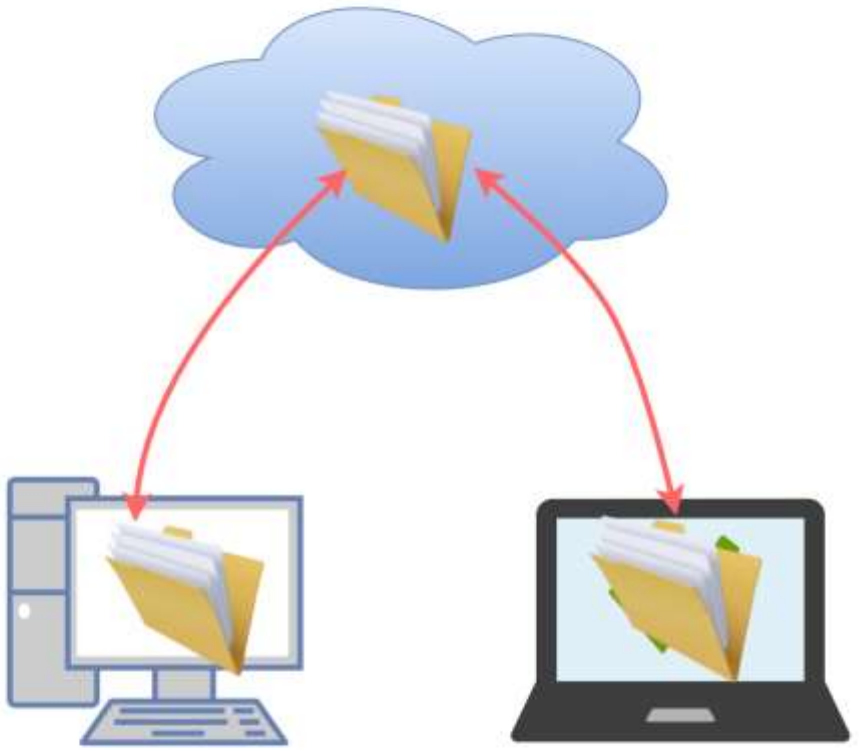
Hard Link

- A **hard link** acts as a copy (mirrored) of the selected file. It accesses the data available in the original file.
  - If the earlier selected file is deleted, the hard link to the file will still contain the data of that file.



Hard Link

- Based on hard links, users can use different file names to access the same file, and all operations end up modifying the same file.
- If only from the user's point of view, it looks like we are working with different files, but these files have automatic synchronization.
- This behavior is somewhat similar to web disk:

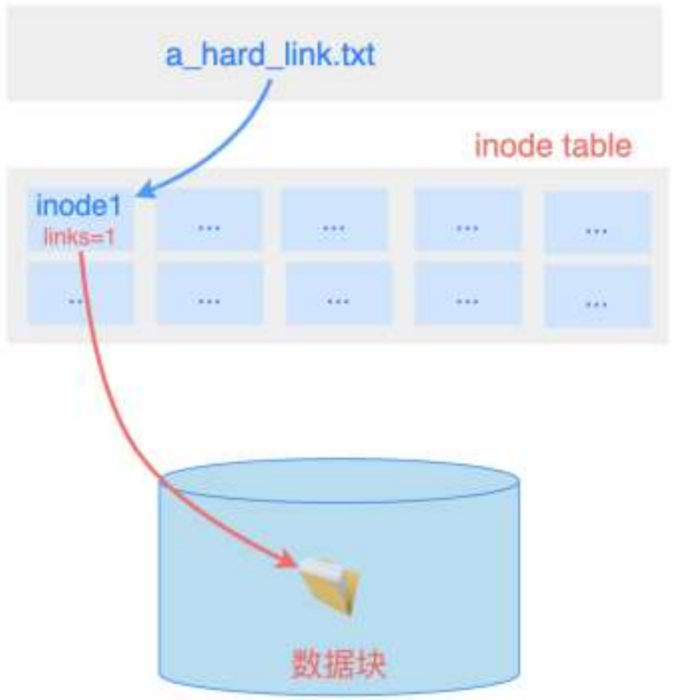


Hard Link

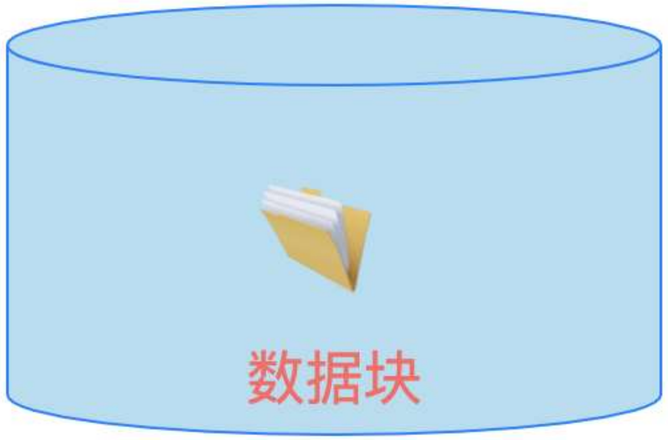
- I have a file hello.txt in the cloud, and then I have two computers A and B, and these two computers will create a local image of the file hello.txt in the cloud, as if the file is on their own hard drive.
- When I manipulate hello.txt on computer A, the file with the same name in computer B is automatically updated.
- Therefore, from a behavioral point of view, hard linking is equivalent to: file copy + automatic synchronization.

Hard Link

- After executing the $ln a.txt a_hard_link.txt command, the value of links in the inode node corresponding to the file is 2.
- If we delete a.txt, the operating system will subtract 1 from the links value of the inode corresponding to the file, the result is 1, the operating system finds that it is not 0, so it will not delete the inode.
- If we delete a_hard_link.txt, the operating system again reduces inde.links by 1, finds that the value becomes 0, deletes the inode, and the file no longer exists



inode table
...
...
...
...
...
...
...
...
...

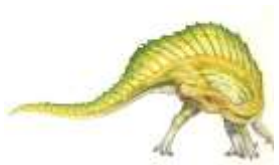

数据块

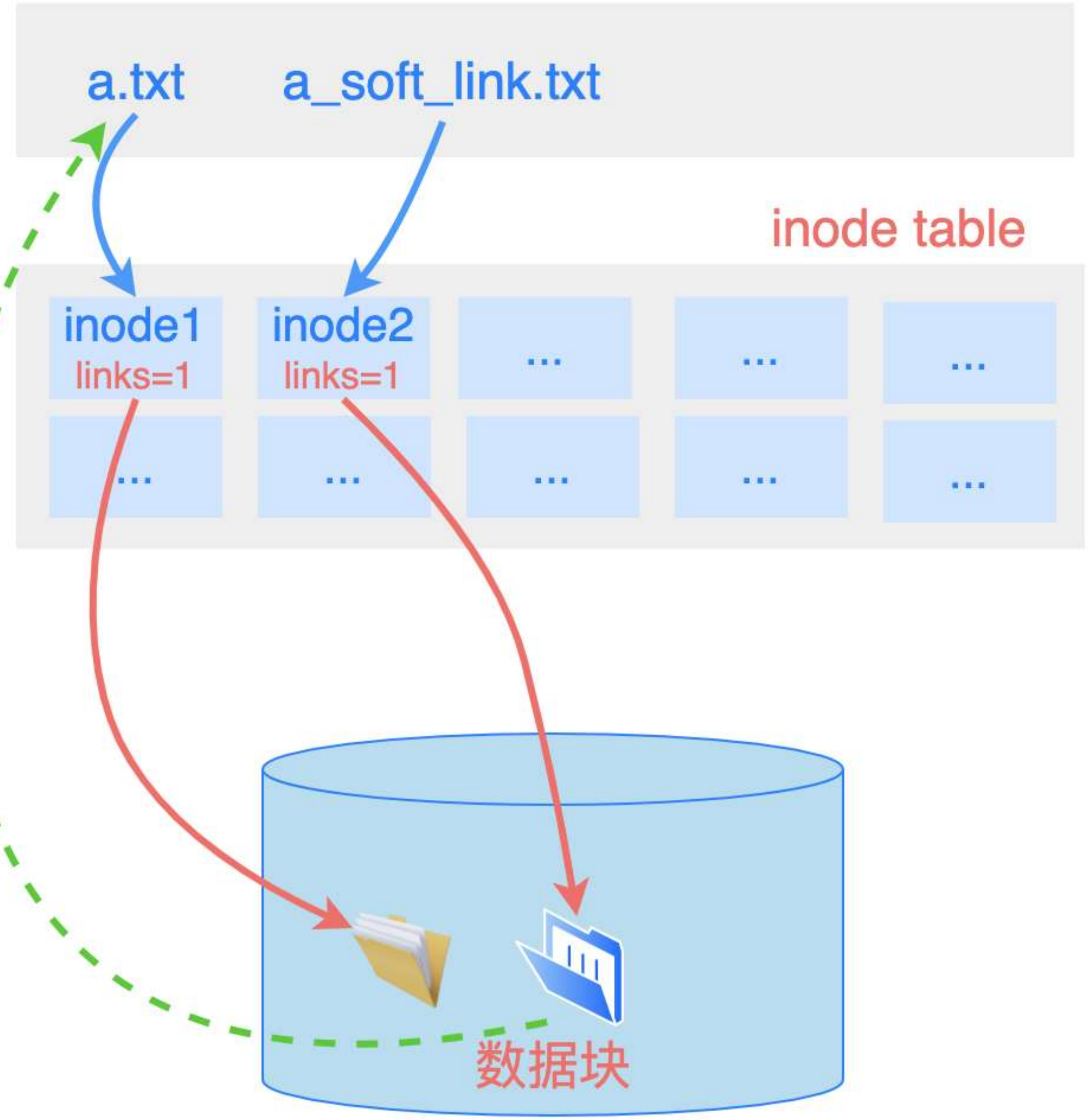
Hard Link

- Hard links have 2 limitations:
  - Users are not allowed to create hard links to directories, that is, users can not, the operating system can (think of each directory. And...) ;
  - A hard link can be created only for files in the same file system, that is, not across file systems.

Soft Link

- A soft link (also known as Symbolic link) acts as a pointer or a reference to the file name.
- It does not access the data available in the original file.
- If the earlier file is deleted, the soft link will be pointing to a file that does not exist anymore.

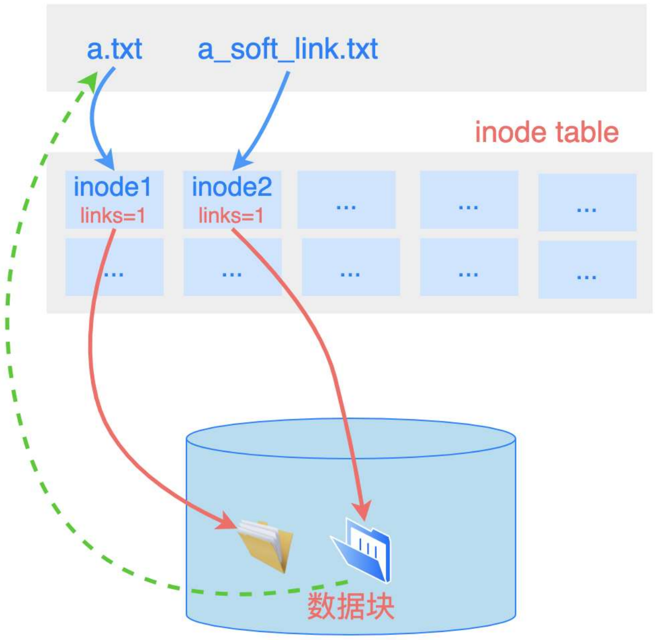




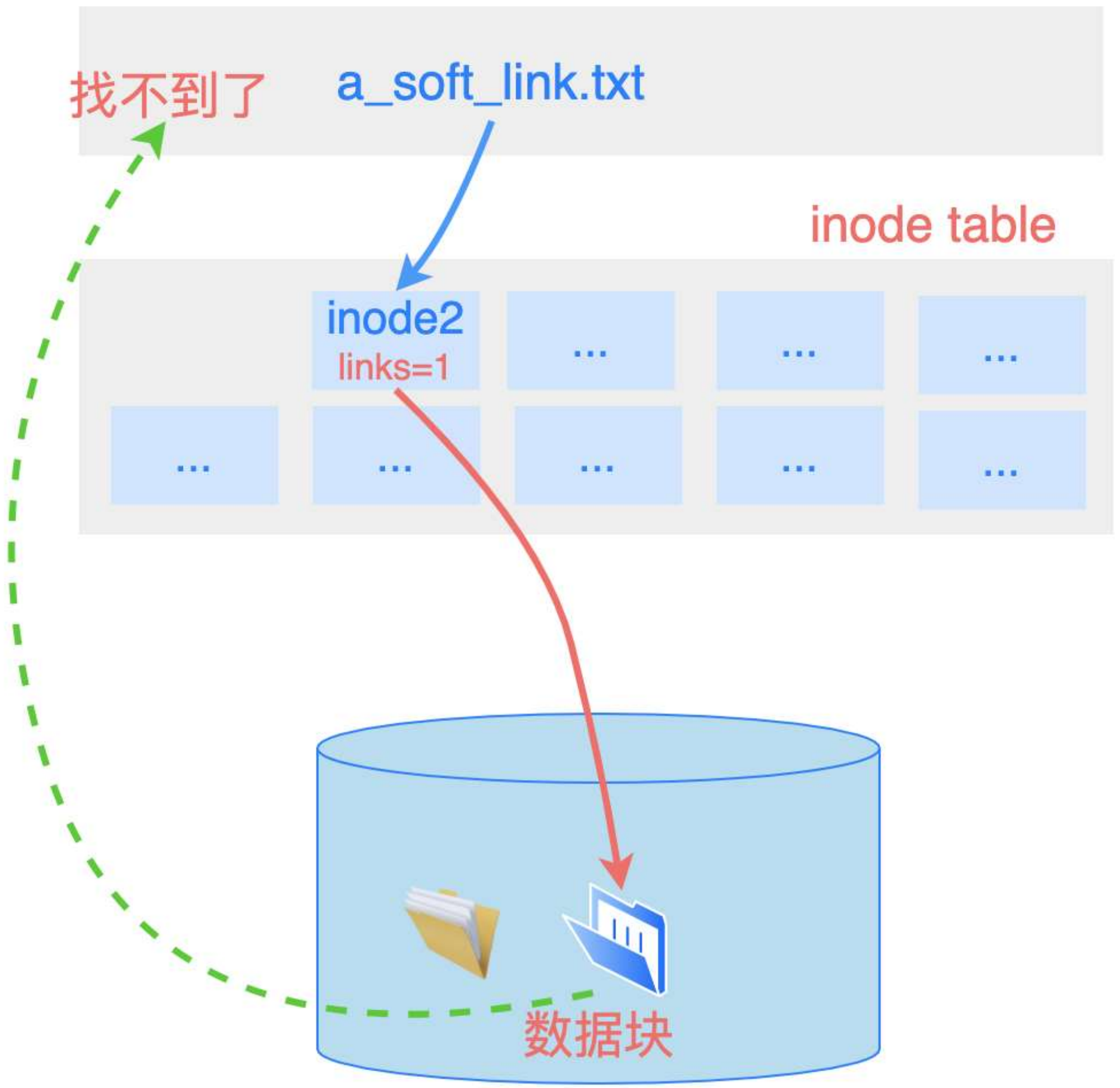
- A s
- The ne of the
- Inst ng syst
- For t to poir

Soft Link

- The green dotted line in the figure represents the file path in the soft link file.
- Because the soft link file stores only the path string of the target file, it can represent a file in any file system, or a directory
- When we open the file soft link a_soft_link.txt, the operating system discovers from the inode data structure corresponding to a_soft_link.txt: this is a soft link file.
- Based on the path information, the operating system finds the inode node of a.xT and operates on the final destination file.



- If w
  equ
- In t
  pat
  cor
- The
- Wh



End of Chapter 11