中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

1924-2024 中山大學 世纪华诞 100th ANNIVERSARY SUN YAT-SEN UNIVERSITY

文件系统I

SSE202/204: 操作系统原理

苏玉鑫

suyx35@mail.sysu.edu.cn

助教：龙玉丹 单诗雯 毛晨希 沈志轩 郑灿峰 胡伟峰

版权信息

- 部分内容来自：上海交通大学并行与分布式系统研究所操作系统课件
  - https://ipads.se.sjtu.edu.cn/courses/os/
- 其它参考资料：
  - 清华大学操作系统公开课
    - https://open.163.com/newview/movie/courseintro?newurl=ME1NSA351
    - 介绍标准内容，适合考研
  - 南京大学计算机软件研究所
    - http://jyywiki.cn/OS/2025/
    - https://space.bilibili.com/202224425/channel/collectiondetail?sid=192498
    - 比较有趣

半程回顾

基本原理

- 操作系统概述
- 硬件结构
- 操作系统结构
- 内存管理
- 进程管理
- 操作系统调度
- 进程间通信
- 同步原语
- 文件系统

通讯

计算

存储

第一层对应关系

半程回顾

基本原理

- 操作系统概述
- 硬件结构
- 操作系统结构
- 内存管理
- 进程管理
- 操作系统调度
- 进程间通信
- 同步原语
- 文件系统

通讯

计算

存储

第二层对应关系

半程回顾

进阶内容

- 文件系统崩溃一致性
- 设备管理
- 系统虚拟化
- 多核与多处理器
- 网络协议栈与系统
- 轻量级虚拟化
- 操作系统安全
- 操作系统调测

通讯

计算

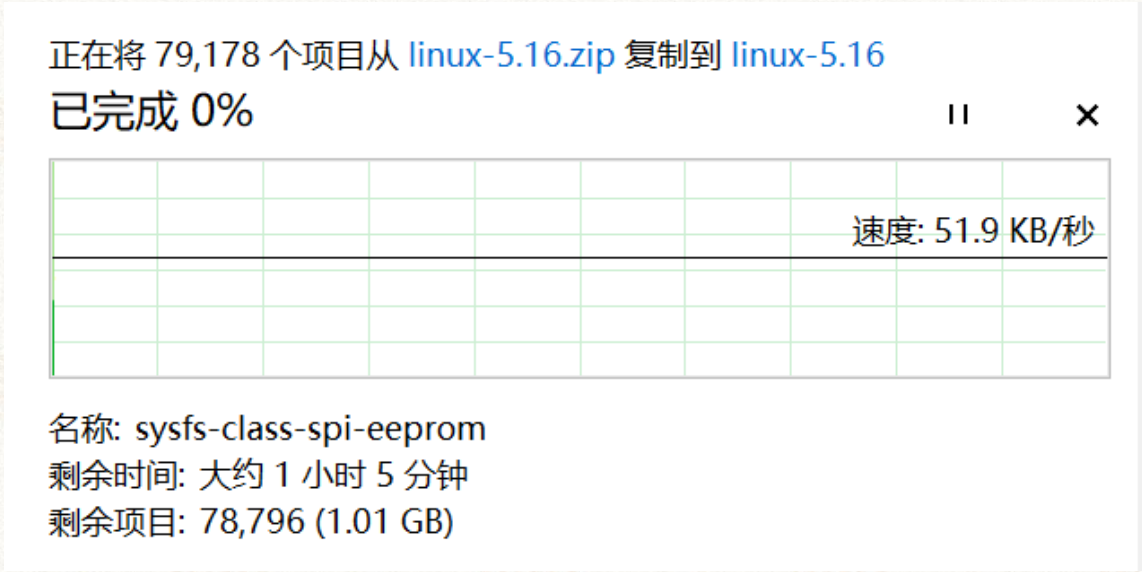
存储

第三层对应关系

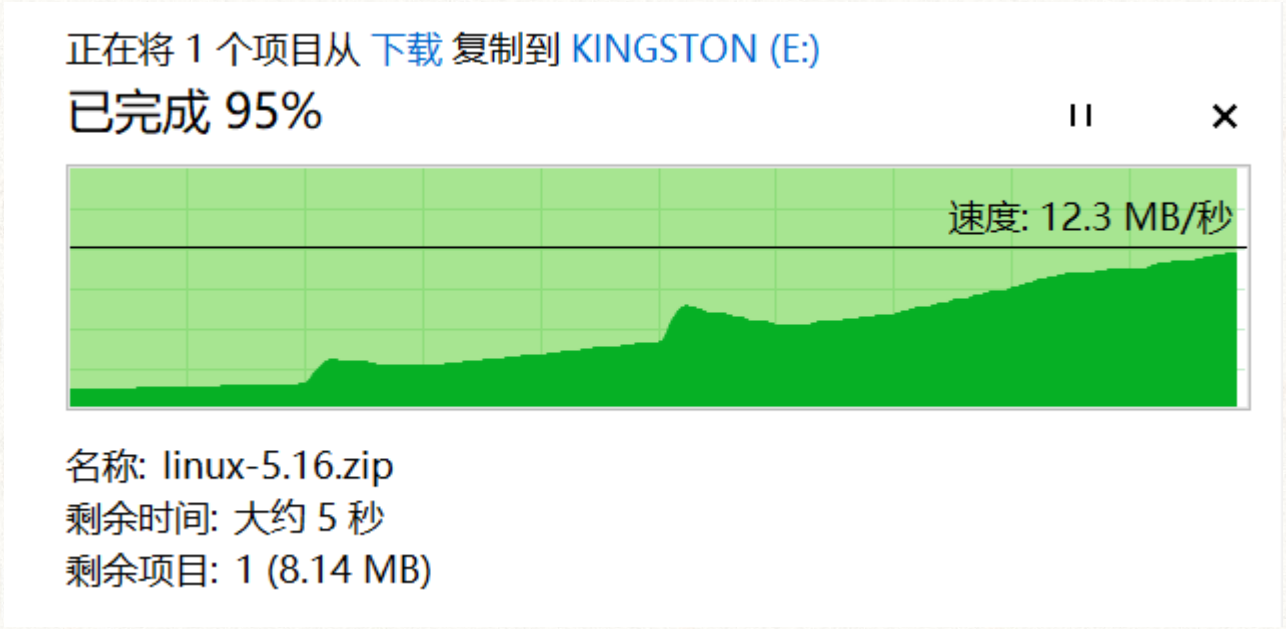
主观题 10分

设置

同样的内容，为什么分开复制与打包复制的速度有很大的差异?



linux内核文件，逐文件复制

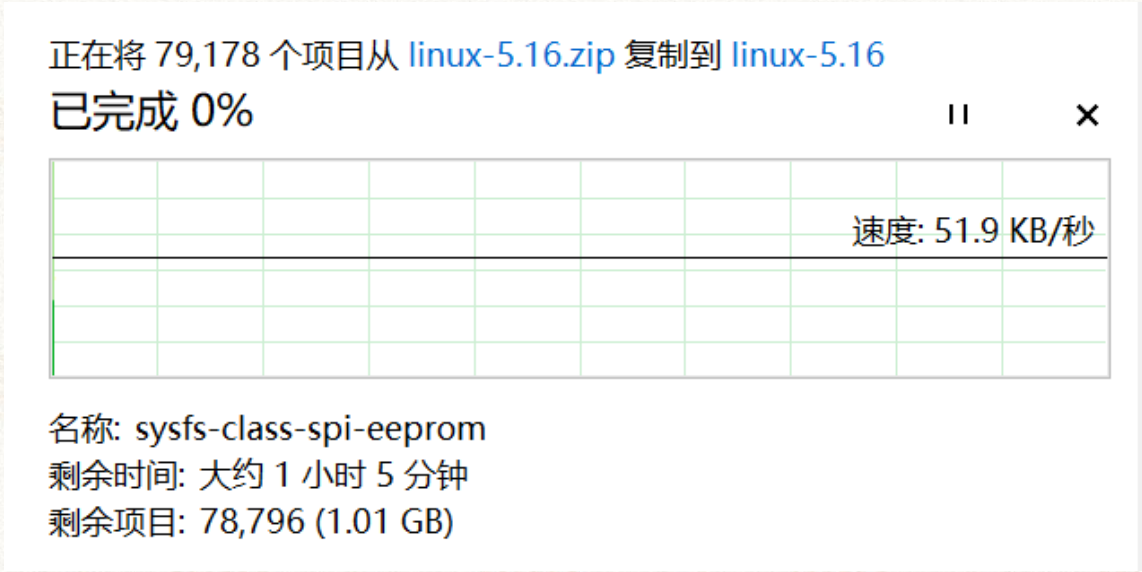


linux内核文件，以压缩包的形式复制

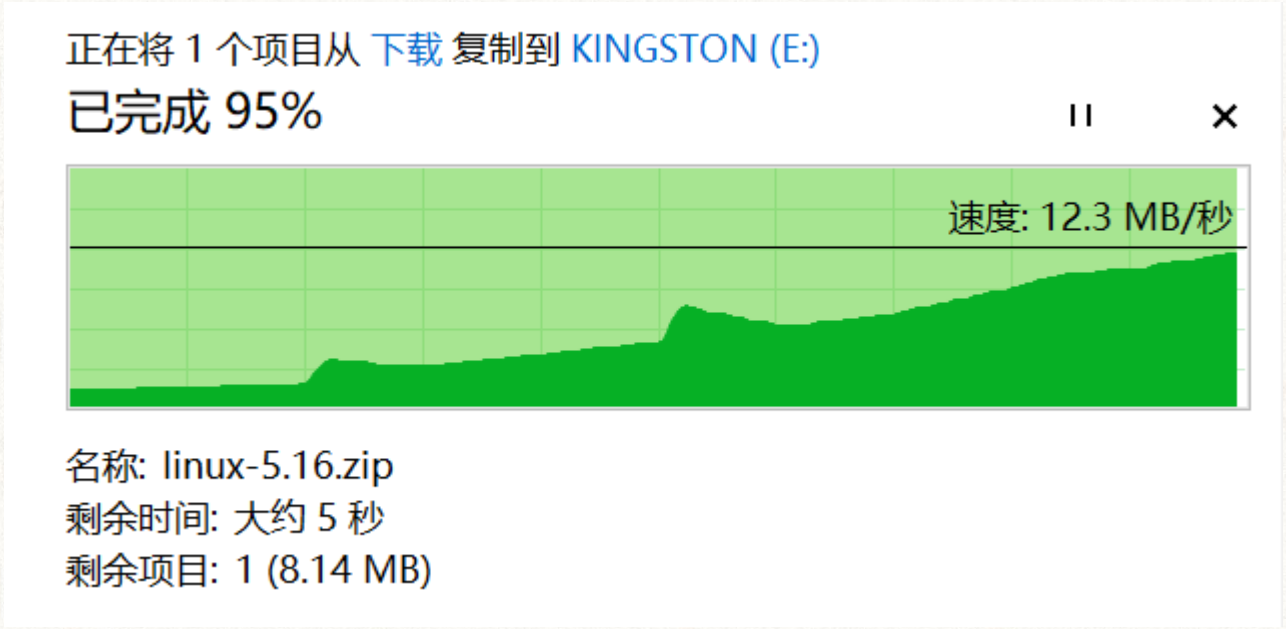
作答

日常使用文件时的怪事

- 同样的内容，分开复制与打包复制的差异



linux内核文件，逐文件复制



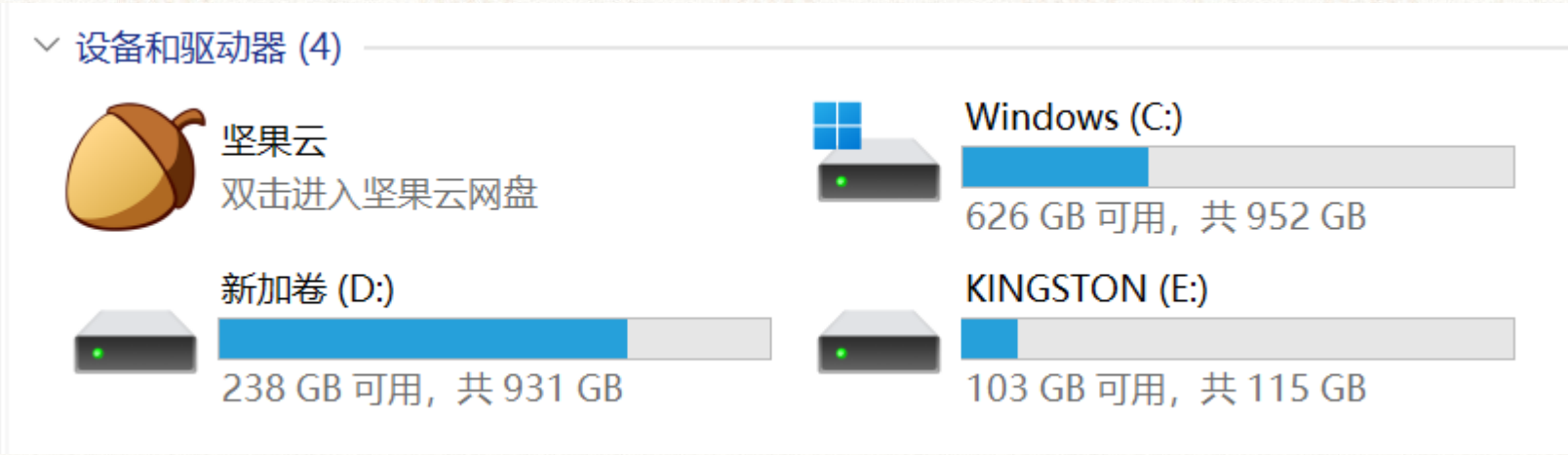
linux内核文件，以压缩包的形式复制

主观题 10分

设置

为什么硬盘、U盘显示的大小和商家宣称的大小不一样?

- 我的C盘D盘是两块硬盘，都是1T大小的
  - 但实际可用的只有952G和931G
- 我的E盘优盘是128G的，也显示只有115G

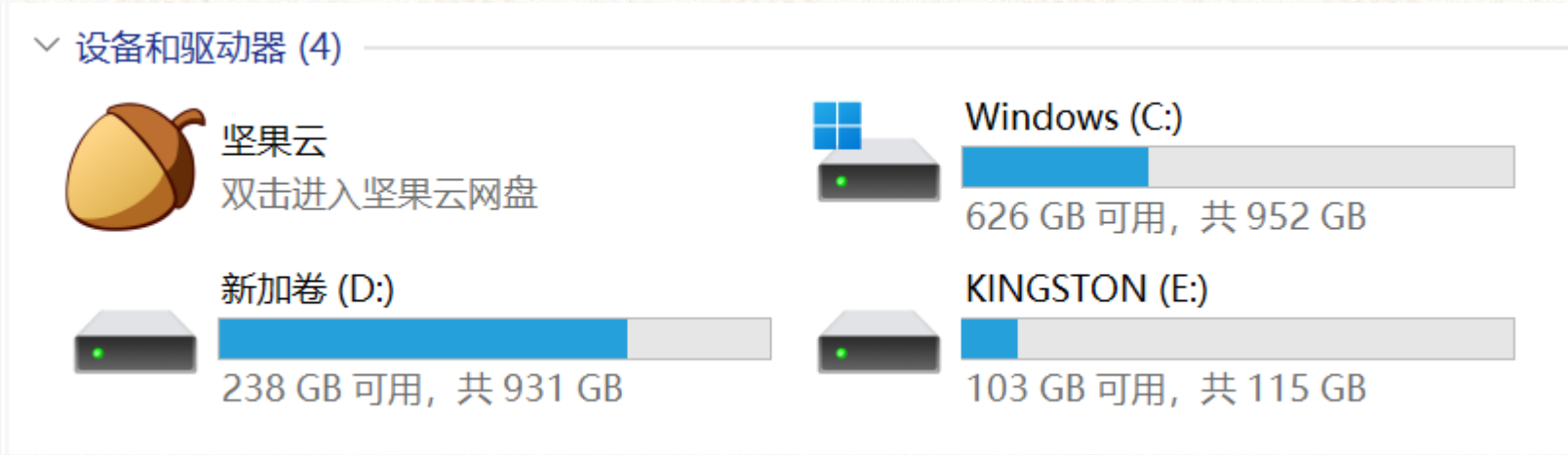


作答

日常使用文件时的怪事

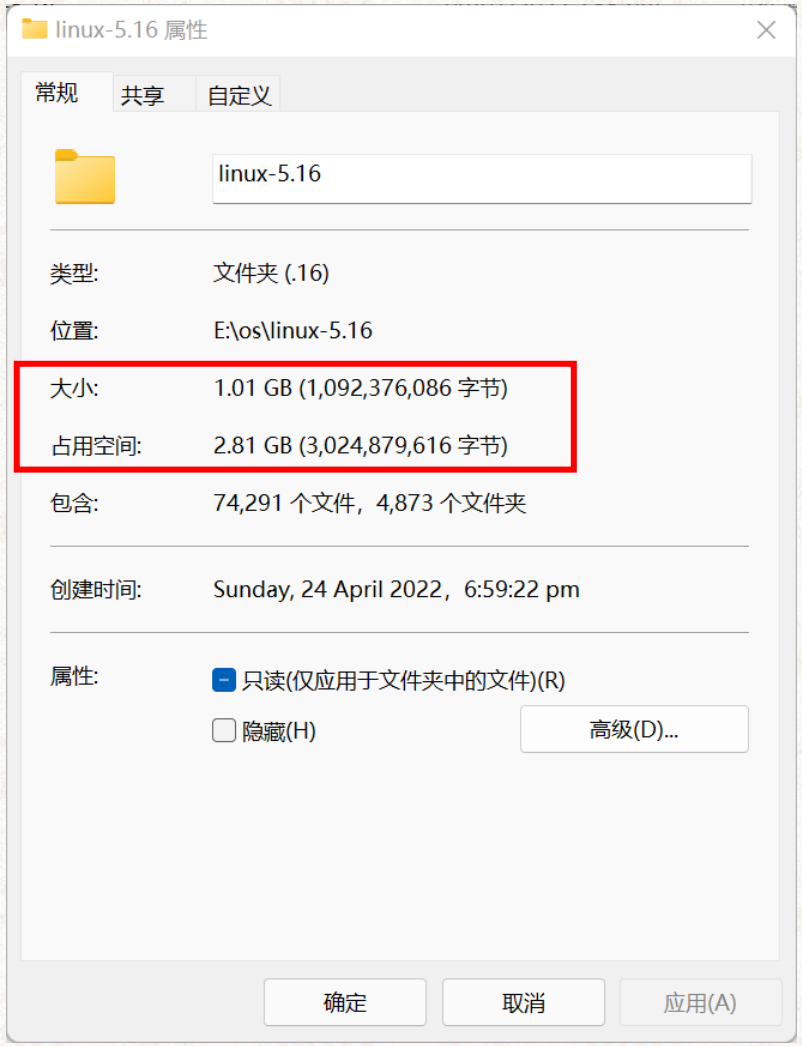
- 我的C盘D盘是两块硬盘，都是1T大小的
- 为什么这显示只有952G和931G？

- 我的E盘优盘是128G的，也显示只有115G

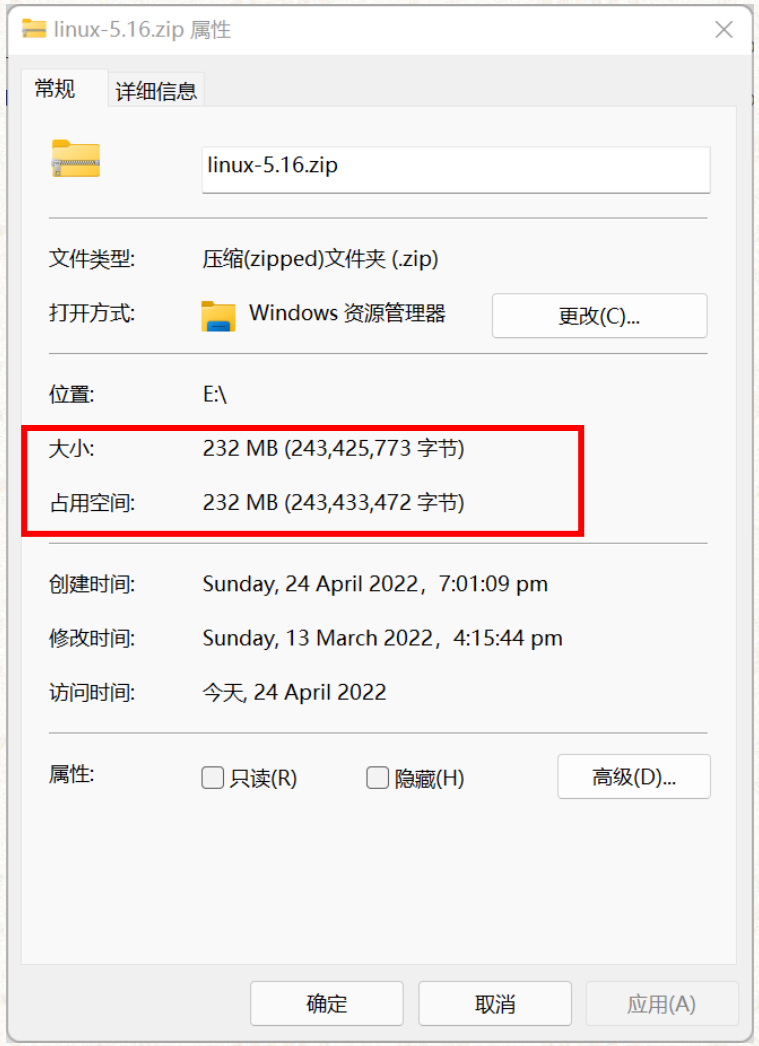


日常使用文件时的怪事

“大小”和“占用空间”为什么是两个概念？



为什么又是相同的？



文件系统大纲

- 块设备
- 基于inode的文件系统
- 基于表的文件系统
  - FAT
  - NTFS
- 使用文件系统
  - 以命令行的形式使用
  - 在代码里使用
    - 基本操作
    - 内存映射
  - 文件系统的高级功能
- 虚拟文件系统
- 用户态文件系统

文件系统大纲

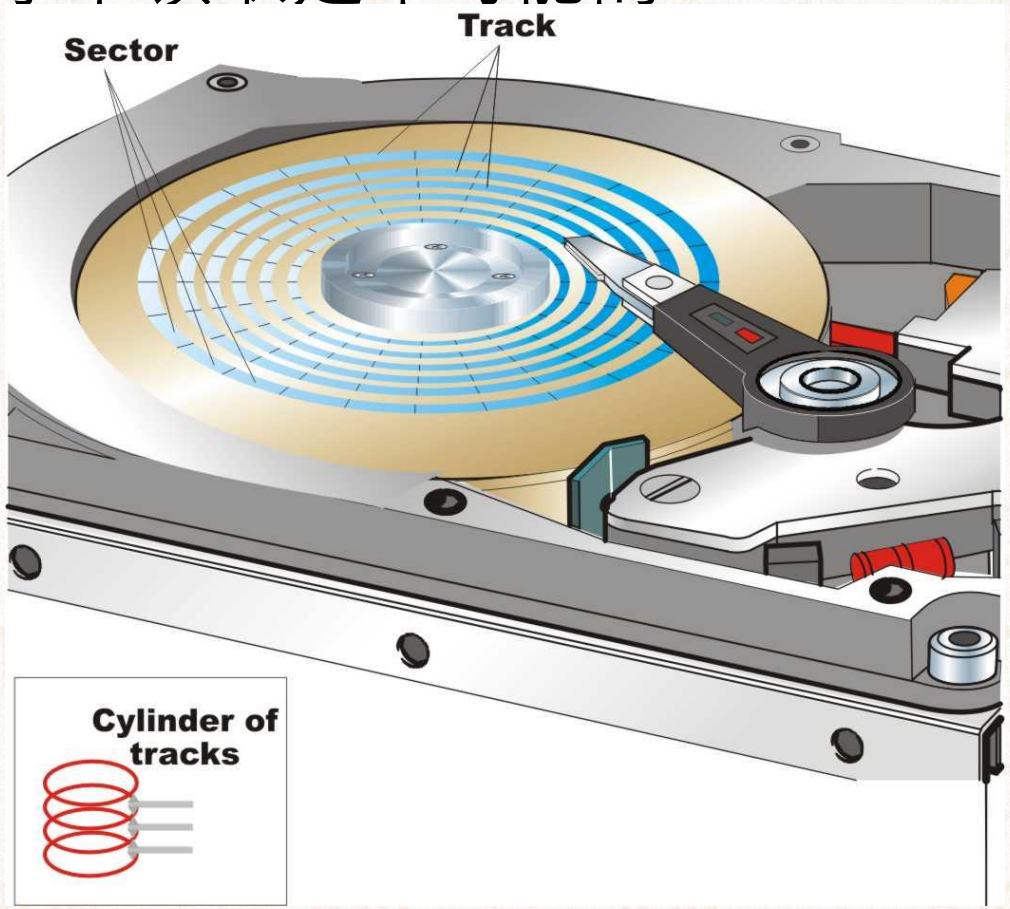
- 块设备
- 基于inode的文件系统
- 基于表的文件系统
  - FAT
  - NTFS
- 使用文件系统
  - 以命令行的形式使用
  - 在代码里使用
    - 基本操作
    - 内存映射
  - 文件系统的高级功能
- 虚拟文件系统
- 用户态文件系统

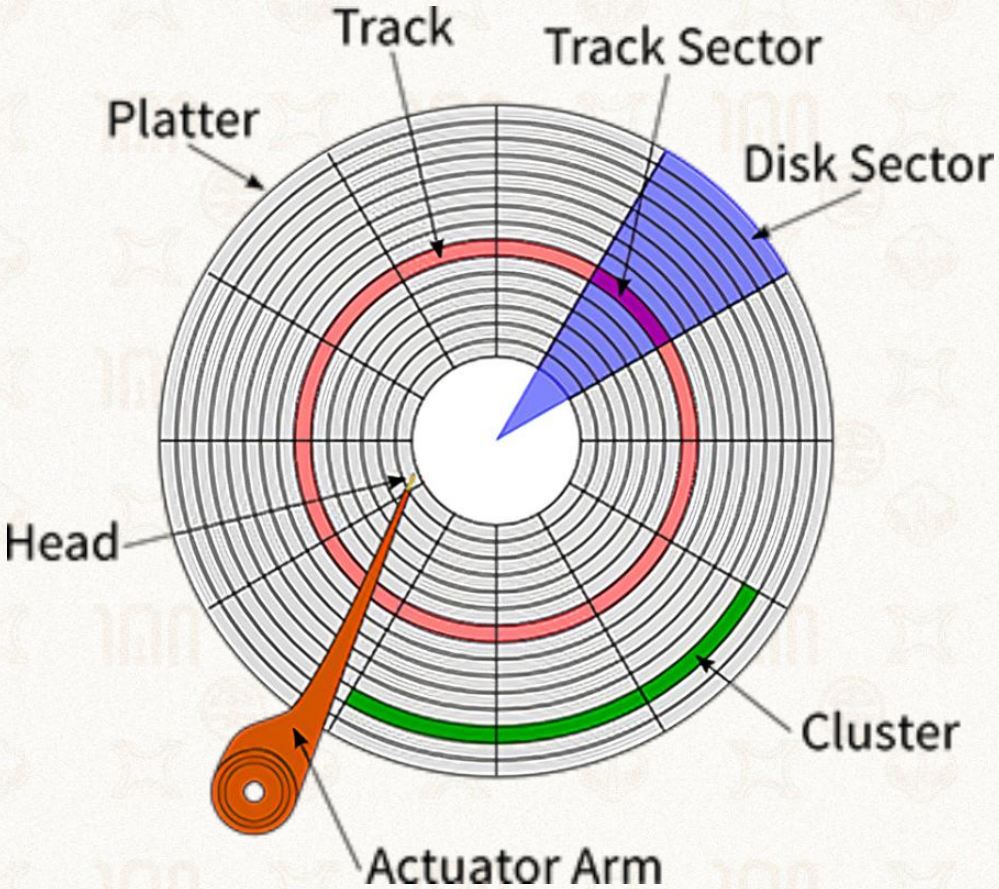
块(block)设备

- 传统机械式硬盘，里面有多个磁盘

块(block)设备

- 磁盘内有多个轨道，探头一次读取一“块”数据
- 单字节读取是不可能的





块(block)设备

磁盘一分钟旋转7200转

西部数据(WD)蓝盘 1TB SATA6Gb/s 7200转64MB 台式机械硬盘(WD10EZEX)

7200转高转速蓝盘，装机存储必备。【点击查看升级至4TB监控硬盘，7*24小时全天值守】查看>

品牌闪购

京东价 ¥ 289.00 降价通知

促销 满送 满4000元即赠热销商品，赠完即止

配送至 广东珠海市香洲区唐家湾镇 有货 支持 可配送海外 | 99元免基础运费 | 180天只换不修

京东物流 次日达 | 预约送货 | 部分收货

由 京东 发货，并提供售后服务. 23:10前下单，预计明天(04月25日)送达

重量 0.515kg

选择颜色

蓝盘 | 日常家用硬盘 | 紫盘 | 安防监控硬盘 | 紫盘 | 轻智能监控盘

紫盘Pro| AI监控硬盘 | 红盘 | NAS存储硬盘 | 红盘Plus| NAS存储盘

红盘Pro | NAS存储盘 | 企业 | 数据中心硬盘 | 企业 | 氦气密封硬盘

黑盘 | 高性能游戏盘 | 金盘 | 企业级存储盘 | 蓝盘 | 2.5"日常存储

黑盘 | 2.5"游戏本盘 | AV系列 | 2.5"监控盘

选择版本

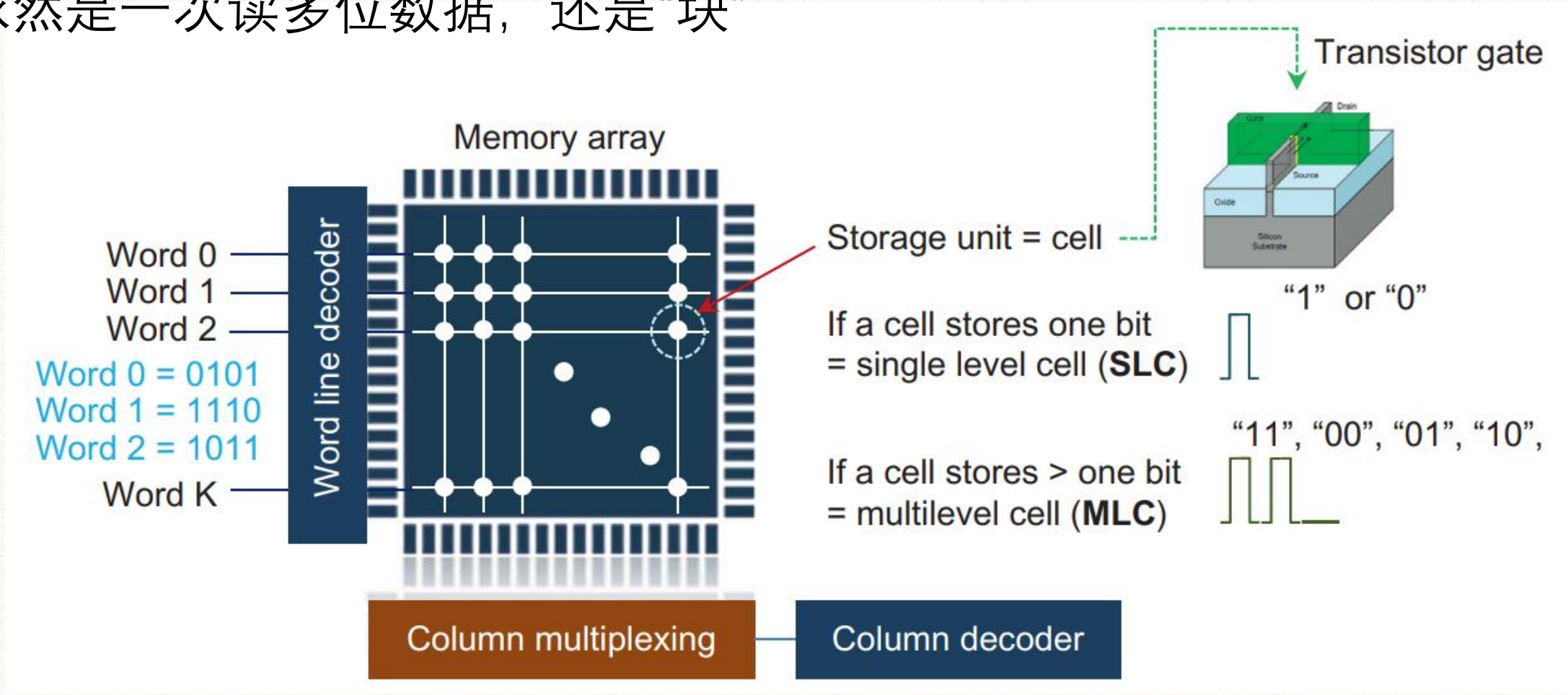
500G | 1TB | 2TB | 2T新 | 3TB | 4TB | 4TBSAS | 5TB | 6TB

块(block)设备：固态硬盘

机械式硬盘　　固态硬盘(Solid State Disk, SSD)

块(block)设备：固态硬盘

- 固态硬盘中闪存(FLASH)的工作原理
- 依然是一次读多位数据，还是“块”

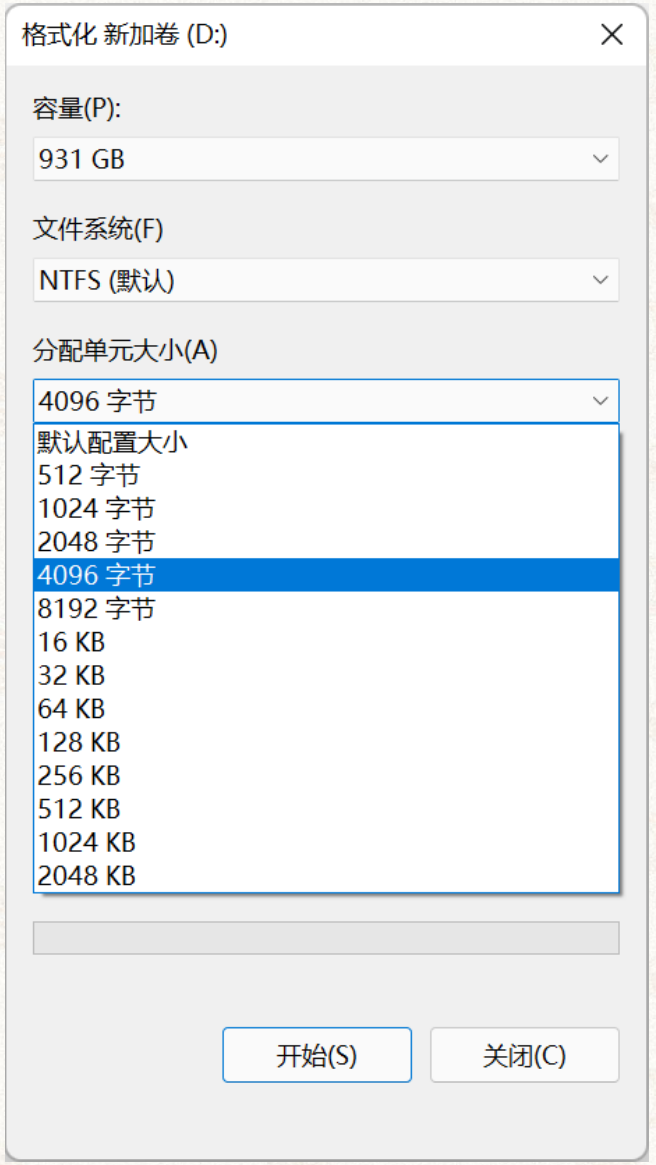


https://www.sciencedirect.com/science/article/pii/B978012801413400012X

块的大小

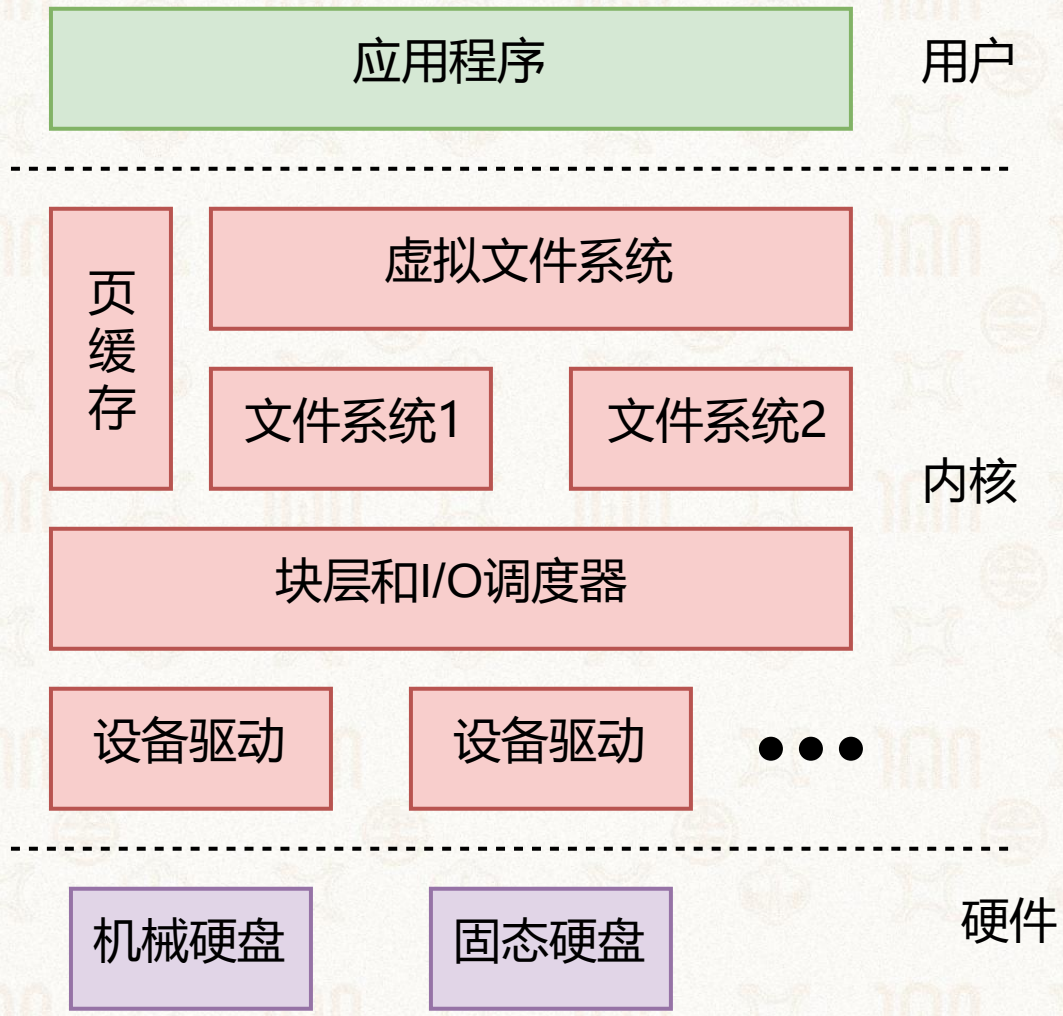
- 一次读写的大小和具体硬件相关，差异很大

- 文件系统关于“存储块”做了一些固定大小的抽象：
  - 一般是512字节或4KB



与块设备匹配的Linux存储软件栈

- 设备驱动
  - 负责与设备进行通讯
- I/O调度
  - 将I/O请求进行合并和调度
- 块层
  - 提供和管理块抽象
- 文件系统
  - 管理存储上的文件系统格式和数据
- 虚拟文件系统
  - 管理多个文件系统，提供统一抽象
- 页缓存：加速读写过程



文件系统大纲

- 块设备
- 基于inode的文件系统
- 基于表的文件系统
  - FAT
  - NTFS
- 使用文件系统
  - 以命令行的形式使用
  - 在代码里使用
    - 基本操作
    - 内存映射
  - 文件系统的高级功能
- 虚拟文件系统
- 用户态文件系统

文件就是有名字的字符序列

- 文本文件当然是字符序列

```
yxsu@lg:~/Desktop$ ls
desktop.ini  stack.cpp
yxsu@lg:~/Desktop$ cat stack.cpp
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

typedef struct Node {
    struct Node *next;
    int value;
} Node;

Node *top = NULL; // 栈顶初始化为NULL
int SLIDE = 1000;
```

cat 命令可以显示文件的内容

文件就是有名字的字符序列

- ppt文件也是

yxsu@lg:~/Desktop$ cat 11-0506-fs-1.pptx | head -n 20
ppt/presentation.xmlPK!猛
Xcppt/_rels/presentation.xml.relsPK!
ppt/slides/slide1.xmlPK!U ppt/slides/slide2.xmlPK![ys9!ppt/slides/slide3.xmlPK!K =7
)ppt/slides/_rels/slide5.xml.relsPK! V| *ppt/slides/slide5.xmlPK! [LR7 Bppt/slides/slide6.xmlPK!" q EJppt/slides
/slide7.xmlPK! {;
Ppt/slides/slide8.xmlPK!c\# 7 Tppt/slides/_rels/slide1.xml.relsPK!K =7 Uppt/slides/_rels/slide2.xml.relsPK!K =7
Vppt/slides/_rels/slide3.xml.relsPK!K =7 Wppt/slides/_rels/slide4.xml.relsPK!K =7
Xppt/slides/_rels/slide6.xml.relsPK!K =7 Yppt/slides/_rels/slide7.xml.relsPK!l m 7
Zppt/slides/_rels/slide8.xml.relsPK! x0 ![ppt/slideLayouts/slideLayout3.xmlPK! z _!;gppt/slideLayouts/slideLayo
ut5.xmlPK! z 2!]mppt/slideMasters/slideMaster1.xmlPK! E v1
! ppt/slideLayouts/slideLayout8.xmlPK!Rb;Q! ppt/slideLayouts/slideLayout2.xmlPK!W L ,Z ppt/slideLayouts/_rels/slide
Layout1.xml.relsPK! ђ 7,z ppt/slideLayouts/_rels/slideLayout3.xml.relsPK! ђ 7, ppt/slideLayouts/_rels/slideLayout4.xml
.relsPK! ђ 7, ppt/slideLayouts/_rels/slideLayout5.xml.relsPK! ђ 7, ppt/slideLayouts/_rels/slideLayout6.xml.relsPK! ђ
7, ppt/slideLayouts/_rels/slideLayout7.xml.relsPK! ђ 7, ppt/slideLayouts/_rels/slideLayout8.xml.relsPK! ђ 7, ppt/s
lideLayouts/_rels/slideLayout2.xml.relsPK! T p ppt/webextensions/taskpanes.xmlPK! 9Γsǫppt/theme/theme3.xmlPK!
X $, ppt/notesMasters/_rels/notesMaster1.xml.relsPK! } $0 ppt/handoutMasters/_rels/handoutMaster1.xml.rels
PK-
! K / docProps/thumbnail.jpegPK!O a\ ppt/theme/theme2.xmlPK *B ppt/webextensions/_rels/taskpanes.xml.relsPK!
q.##J ppt/webextensions/webextension1.xmlPK! ?_G !! ppt/notesMasters/notesMaster1.xmlPK! C %ppt/ha
ndoutMasters/handoutMaster1.xmlPK! R 4? ppt/theme/theme1.xmlPK-

文件就是有名字的字符序列

- pdf文件也不过如此

```
yxsu@lg:~/Desktop$ cat 10-sync-0429.pdf | head -n 10
%PDF-1.7
%
1 0 obj
<</Type/Catalog/Pages 2 0 R/Lang(zh) /StructTreeRoot 266 0 R/Outlines 198 0 R/MarkInfo<</Marked true>>/Metadata 7683 0 R/ViewerPreferences 7684 0 R>>
endobj
2 0 obj
<</Type/Pages/Count 61/Kids[ 3 0 R 23 0 R 40 0 R 42 0 R 46 0 R 65 0 R 67 0 R 69 0 R 71 0 R 73 0 R 82 0 R 84 0 R 86 0 R 88 0 R 90 0 R 92 0 R 94 0 R 101 0 R 103 0 R 105 0 R 107 0 R 109 0 R 111 0 R 113 0 R 115 0 R 117 0 R 119 0 R 121 0 R 123 0 R 125 0 R 127 0 R 129 0 R 131 0 R 133 0 R 135 0 R 137 0 R 139 0 R 141 0 R 143 0 R 145 0 R 147 0 R 149 0 R 151 0 R 153 0 R 155 0 R 157 0 R 159 0 R 161 0 R 163 0 R 165 0 R 167 0 R 169 0 R 171 0 R 173 0 R 175 0 R 177 0 R 179 0 R 181 0 R 183 0 R 185 0 R 187 0 R] >>
endobj
3 0 obj
<</Type/Page/Parent 2 0 R/Resources<</XObject<</Image5 5 0 R/Meta7 7 0 R/Image8 8 0 R>>/ExtGState<</GS6 6 0 R/GS15 15 0 R>>/Font<</F1 10 0 R/F2 16 0 R/F3 18 0 R>>/ProcSet[/PDF/Text/ImageB/ImageC/ImageI] >>/MediaBox[ 0 0 960 540] /Contents 4 0 R/Group<</Type/Group/S/Transparency/CS/DeviceRGB>>/Tabs/S/StructParents 0>>
```

文件就是有名字的字符序列

- PNG图片文件也一样

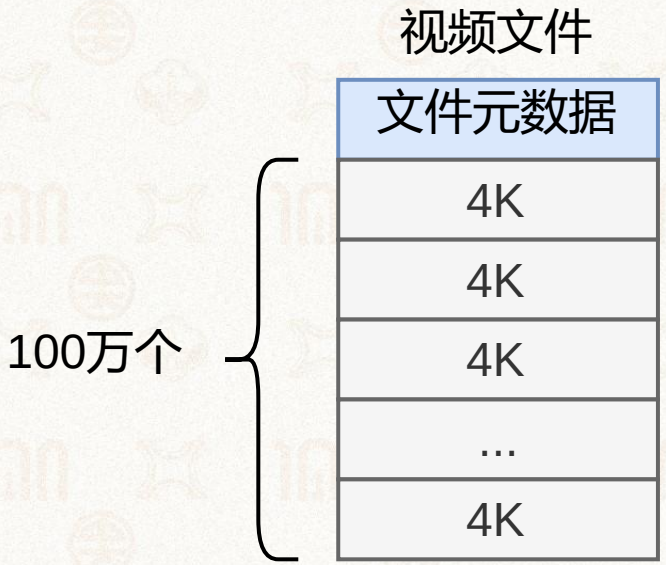
sselogo20220915_0.png

yxsu@lg:~/Desktop$ cat sselogo20220915_0.png | head -n 5

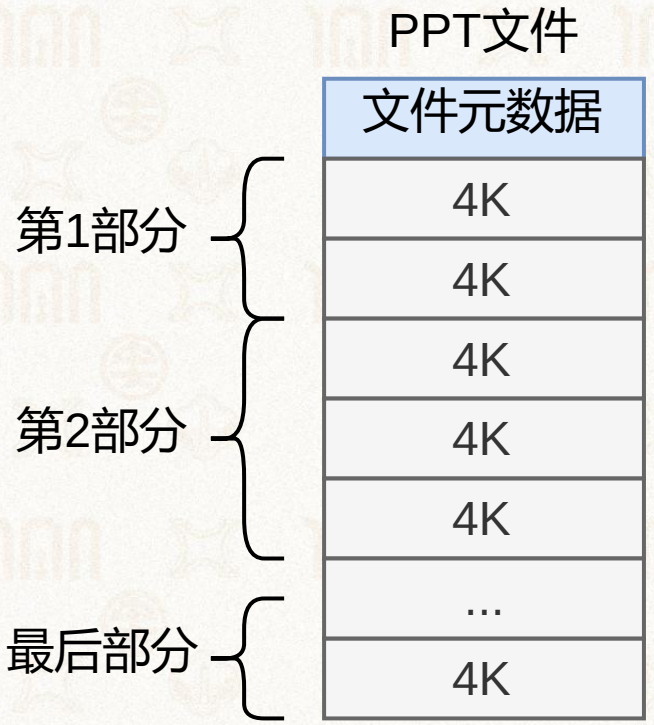
PNG
IHDR
d pHYs.#.#x ?vSiTXtXML:com.adobe.xmp<?xpacket begin="" id="W5M0MpCehiHzreSzNTczkc9d"?> <x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Adobe XMP Core 5.6-c142 79.160924, 2017/07/13-01:06:39 "> <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"> <rdf:Description rdf:about="" xmlns:xmp="http://ns.adobe.com/xap/1.0/" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:photoshop="http://ns.adobe.com/photoshop/1.0/" xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/" xmlns:stEvt="http://ns.adobe.com/xap/1.0/sType/ResourceEvent#" xmp:CreatorTool="Adobe Photoshop CC (Windows)" xmp:CreateDate="2022-09-15T09:55:30+08:00" xmp:ModifyDate="2022-09-15T10:01:15+08:00" xmp:MetadataDate="2022-09-15T10:01:15+08:00" dc:format="image/png" photoshop:ColorMode="3" photoshop:ICCProfile="sRGB IEC61966-2.1" xmpMM:InstanceID="xmp.iid:f1a0535b-4aba-1743-8a07-9292e2246b99" xmpMM:DocumentID="adobe:docid:photoshop:c9208732-1dee-6848-9f79-7f516e6a402f" xmpMM:OriginalDocumentID="xmp.did:1a903096-1e34-404b-b92a-3c116c4bd71e"> <xmpMM:History> <rdf:Seq> <rdf:li stEvt:action="created" stEvt:instanceID="xmp.iid:1a903096-1e34-404b-b92a-3c116c4bd71e" stEvt:when="2022-09-15T09:55:30+08:00" stEvt:softwareAgent="Adobe Photoshop CC (Windows)"/> <rdf:li stEvt:action="converted" stEvt:parameters="from application/vnd.adobe.photoshop to image/png"/> <rdf:li stEvt:action="saved" stEvt:instanceID="xmp.iid:f1a0535b-4aba-1743-8a07-9292e2246b99" stEvt:when="2022-09-15T10:01:15+08:00" stEvt:softwareAgent="Adobe Photoshop CC (Windows)" stEvt:changed="/"/> </rdf:Seq>

如何用“块”来存储这些字符序列

- 如果一个视频文件大小为4G，假设块大小为4K，需要1M个块

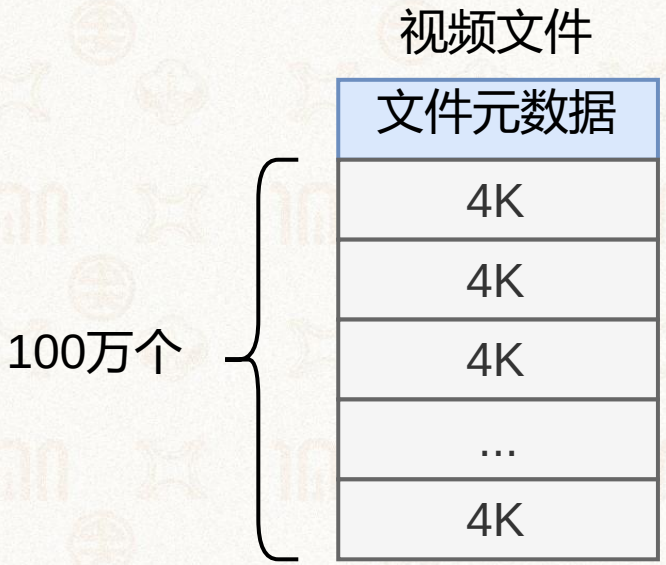


- 有些文件会经常修改

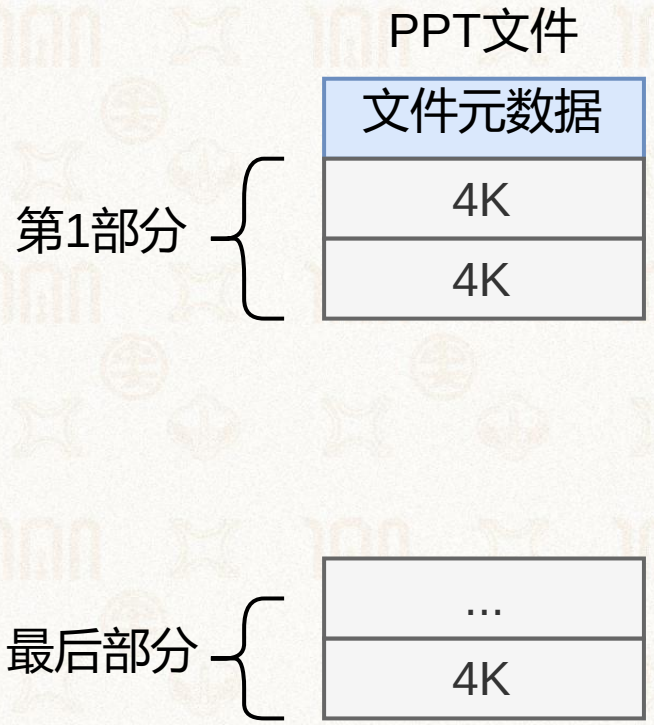


如何用“块”来存储这些字符序列

- 如果一个视频文件大小为4G，假设块大小为4K，需要1M个块

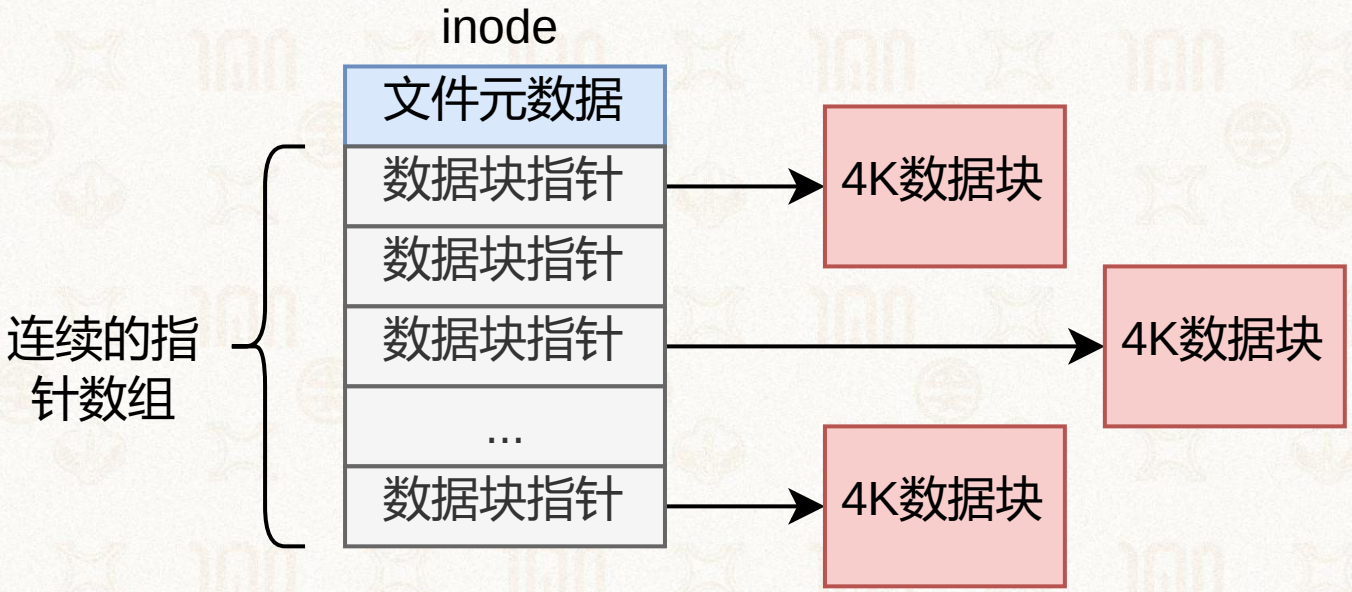


- 文件经常修改，怎么灵活处理？



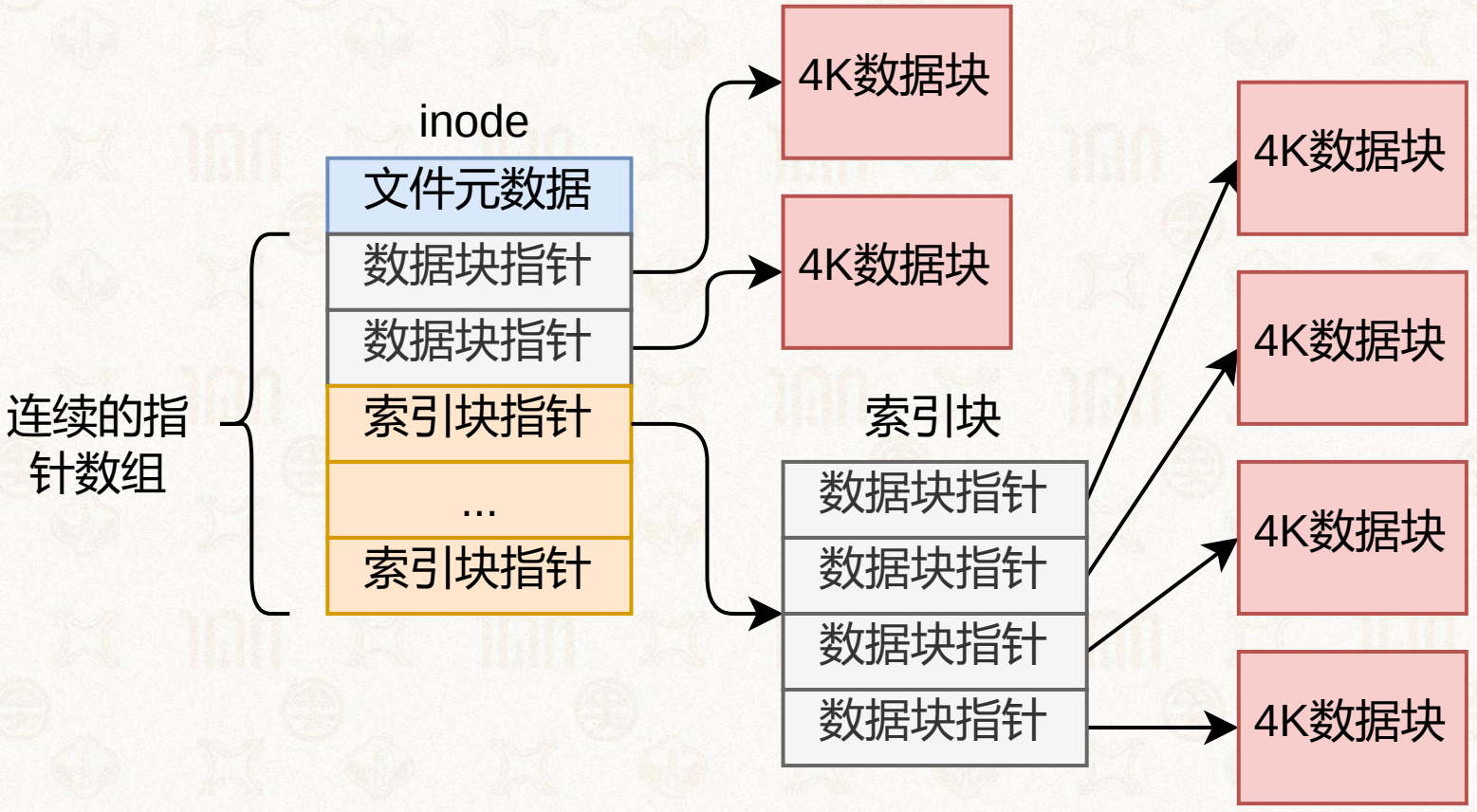
inode: 常规文件的数据索引方式

- 4G大小的文件：
  - 用100万个指针数组，指针长度8字节，数组共8M大小



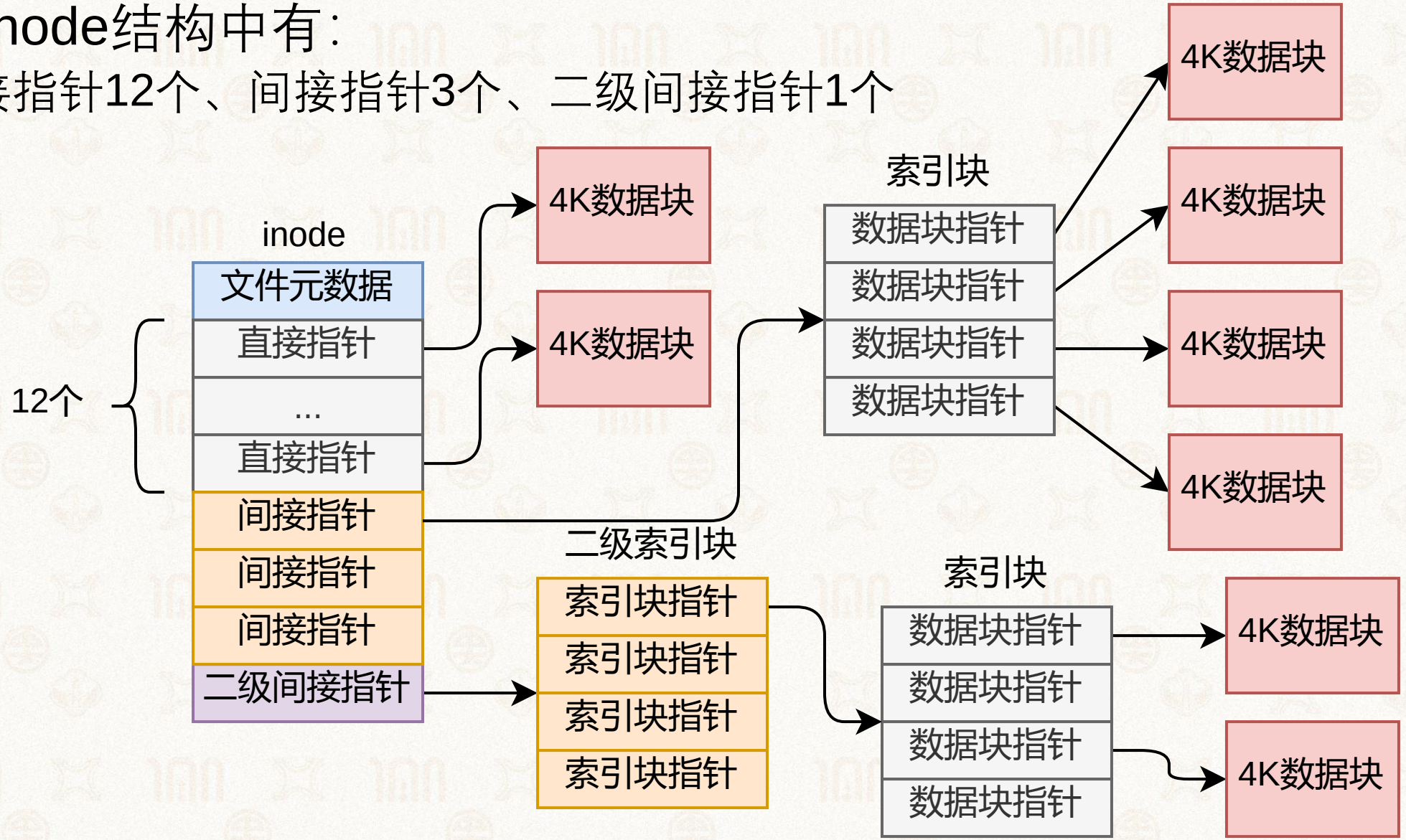
inode: 常规文件的数据索引方式

- 和多级页表类似，用分级减少组织管理代价



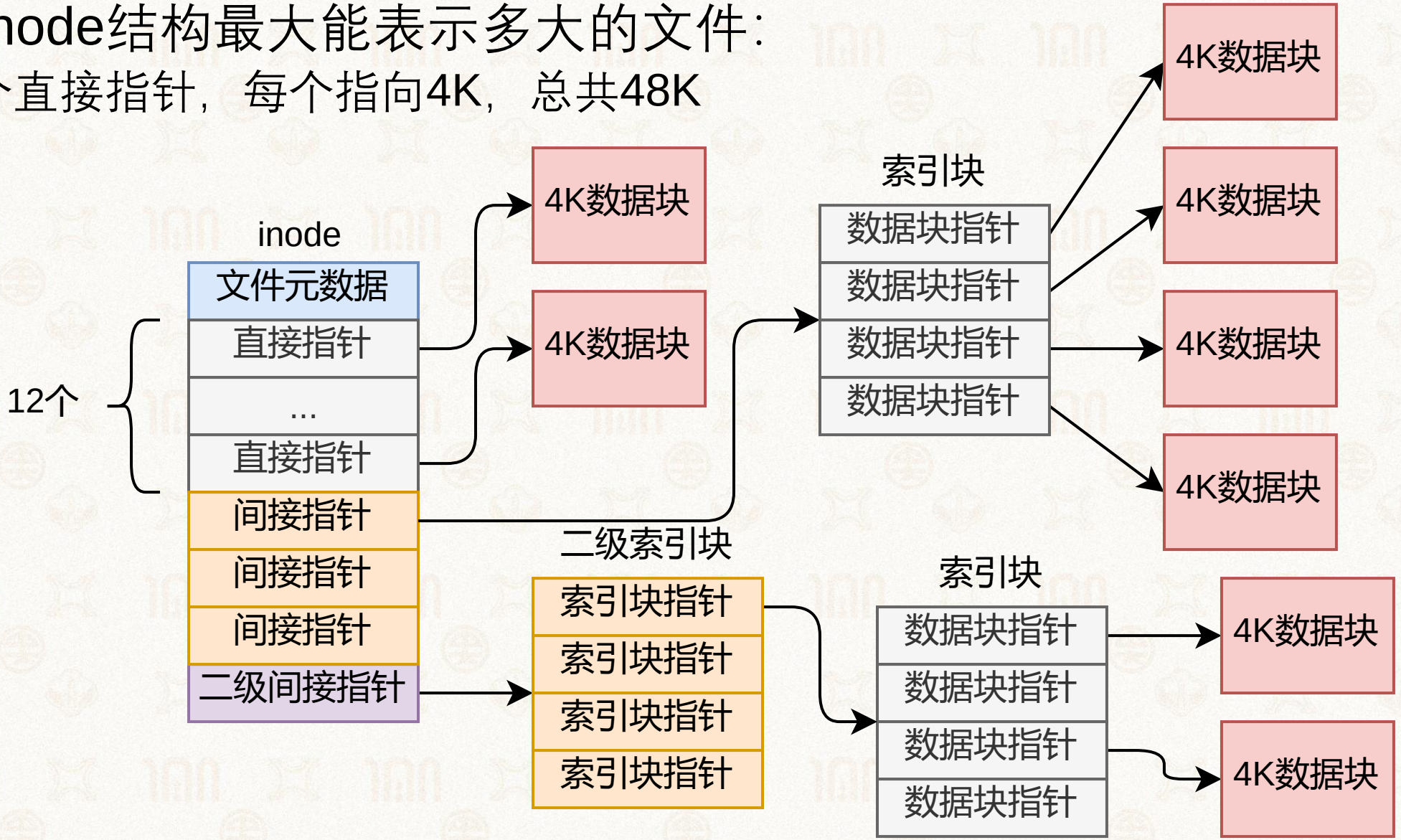
inode: 常规文件的数据索引方式

- 假设inode结构中有：
  - 直接指针12个、间接指针3个、二级间接指针1个



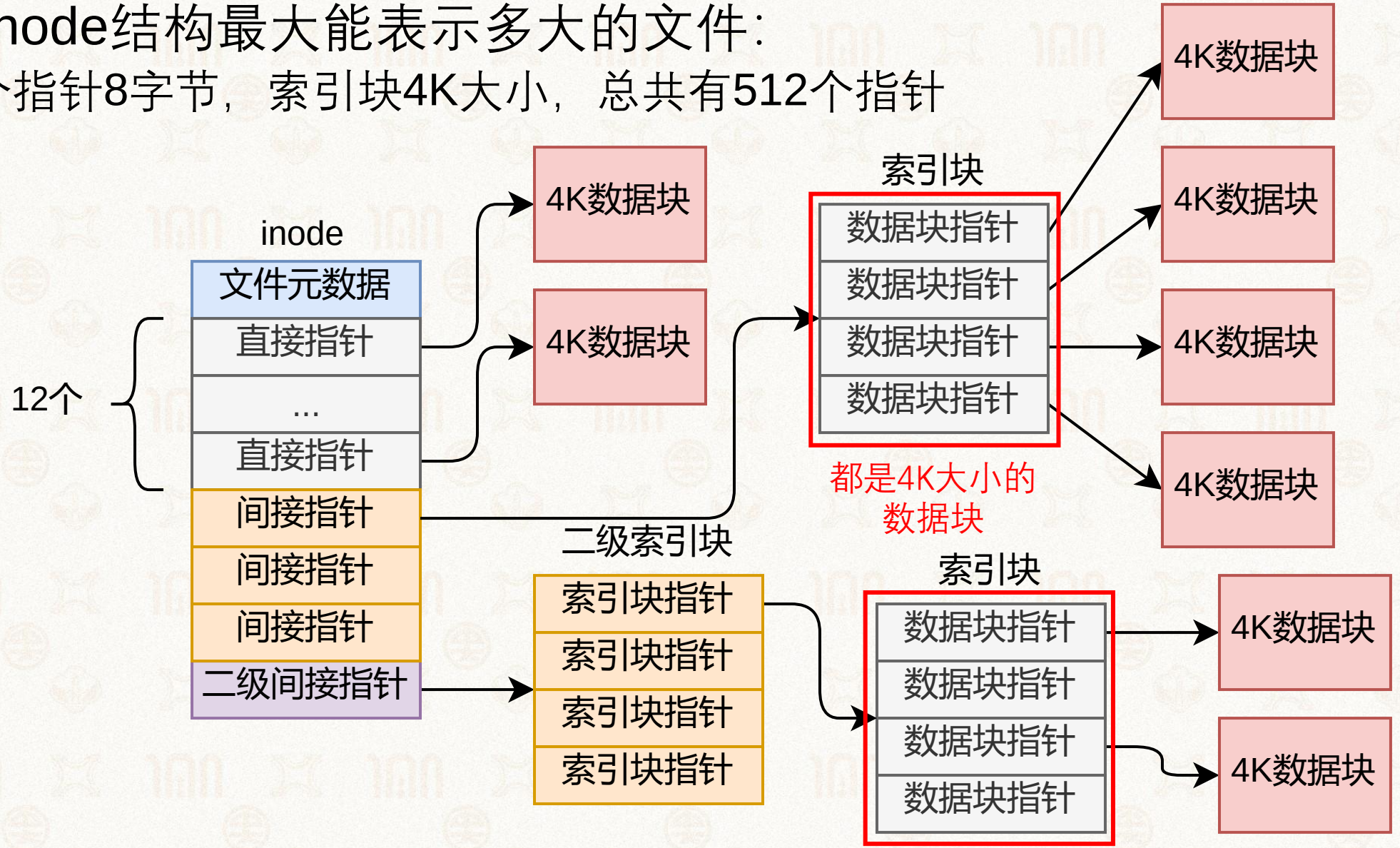
inode: 常规文件的数据索引方式

- 一个inode结构最大能表示多大的文件：
  - 12个直接指针，每个指向4K，总共48K



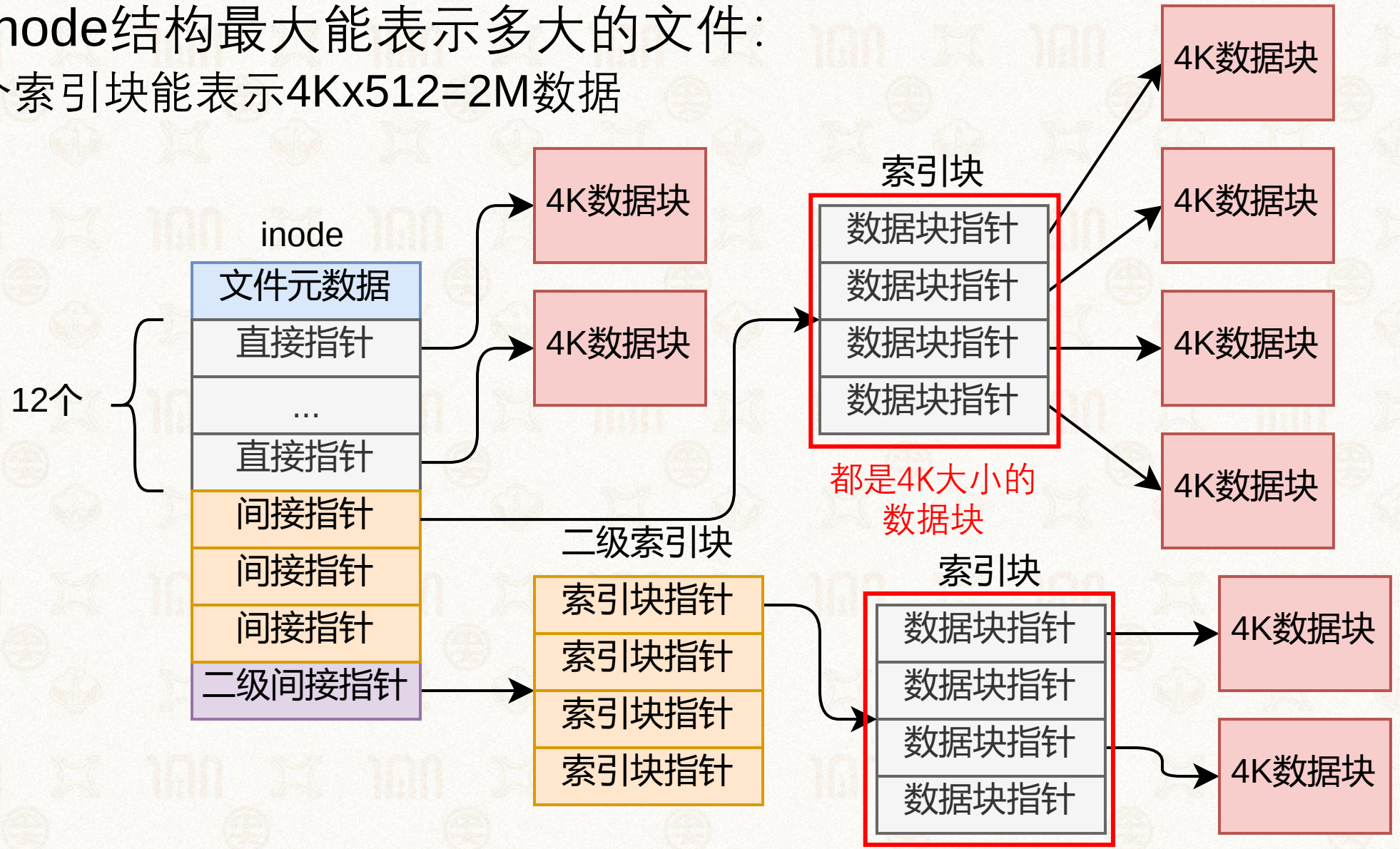
inode: 常规文件的数据索引方式

- 一个inode结构最大能表示多大的文件：
  - 每个指针8字节，索引块4K大小，总共有512个指针



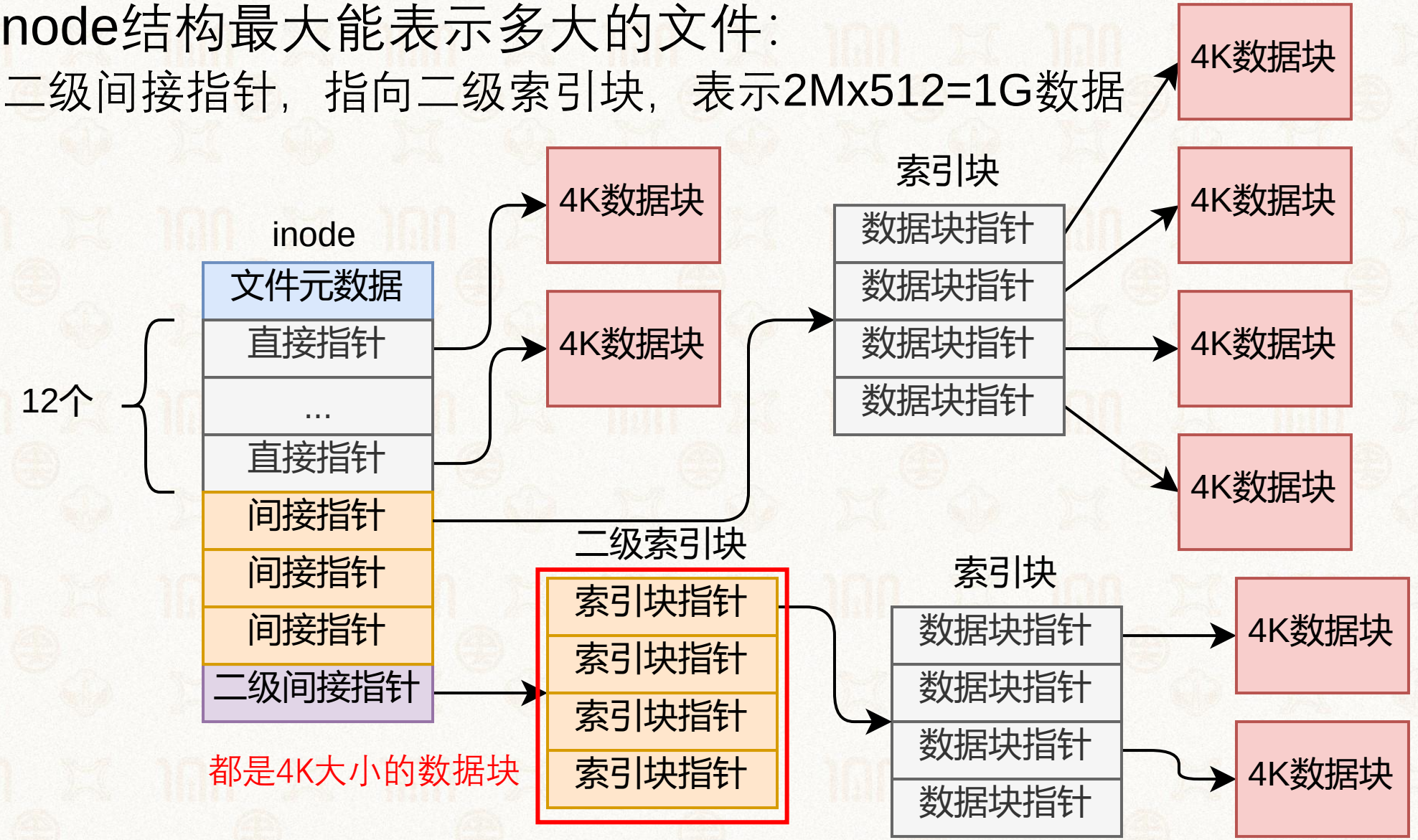
inode: 常规文件的数据索引方式

- 一个inode结构最大能表示多大的文件:
  - 一个索引块能表示4Kx512=2M数据



inode: 常规文件的数据索引方式

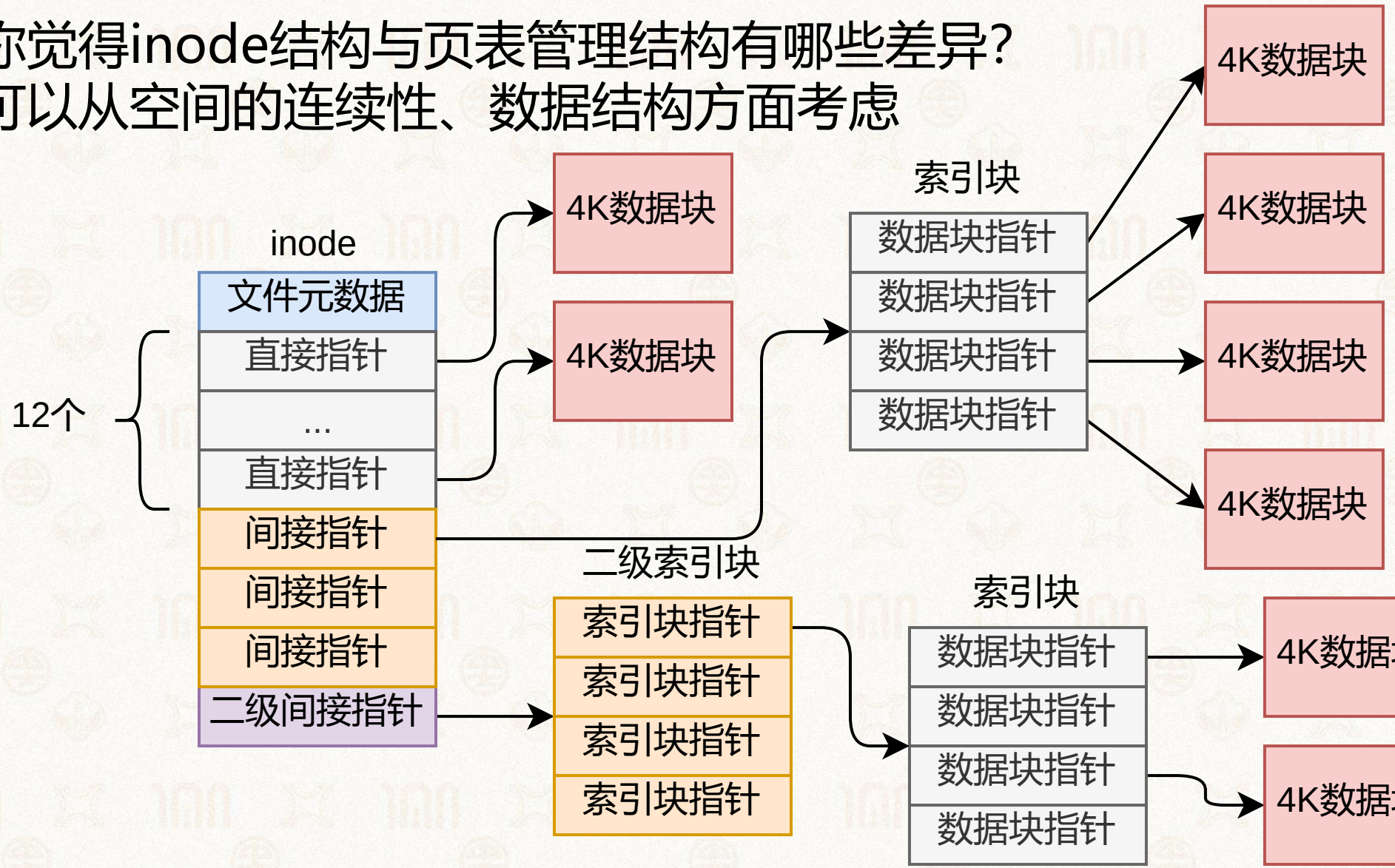
- 一个inode结构最大能表示多大的文件：
  - 1个二级间接指针，指向二级索引块，表示2Mx512=1G数据



主观题 10分

设置

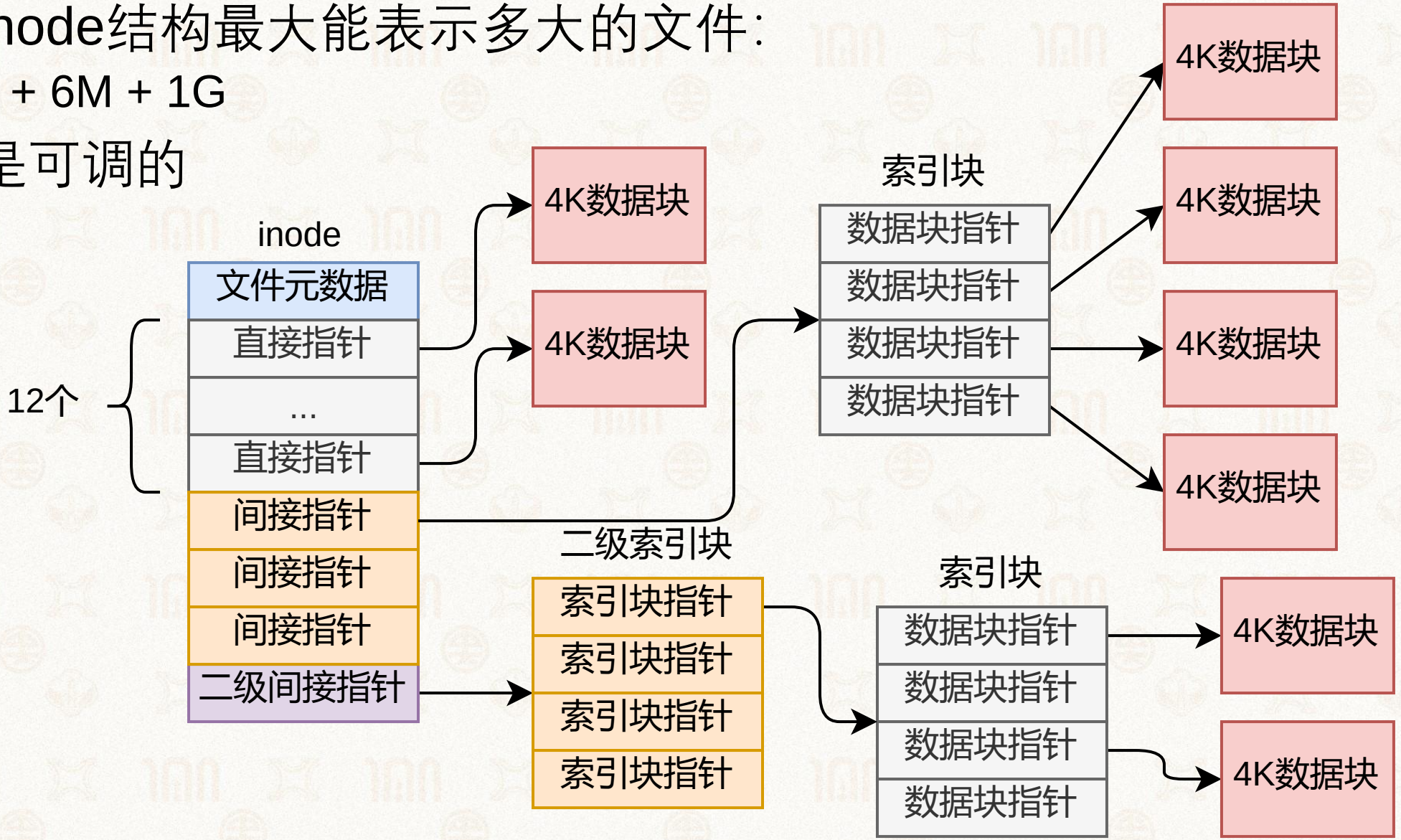
你觉得inode结构与页表管理结构有哪些差异?
可以从空间的连续性、数据结构方面考虑



作答

inode: 常规文件的数据索引方式

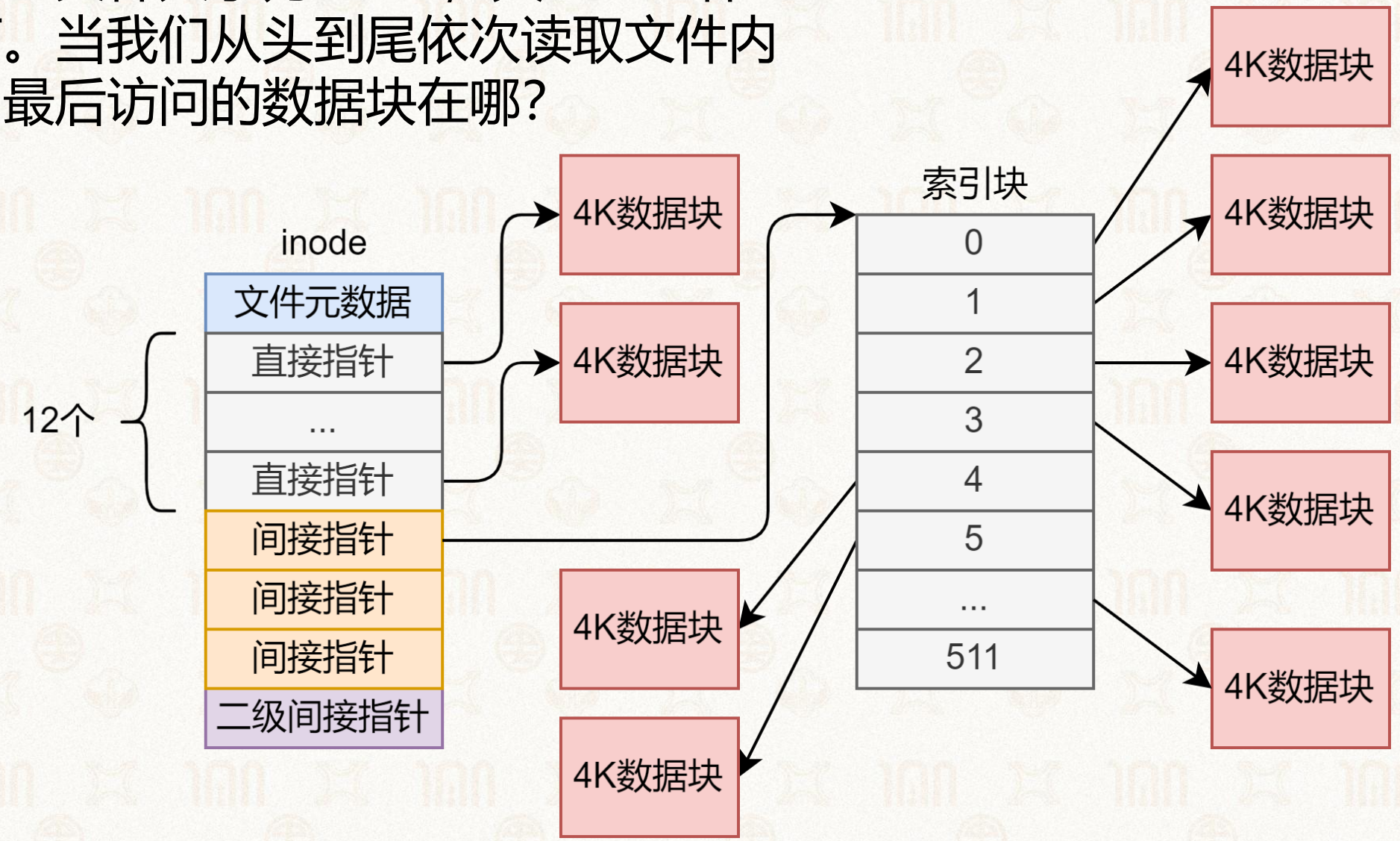
- 一个inode结构最大能表示多大的文件:
  - 48K + 6M + 1G
- 比例是可调的



单选题 1分

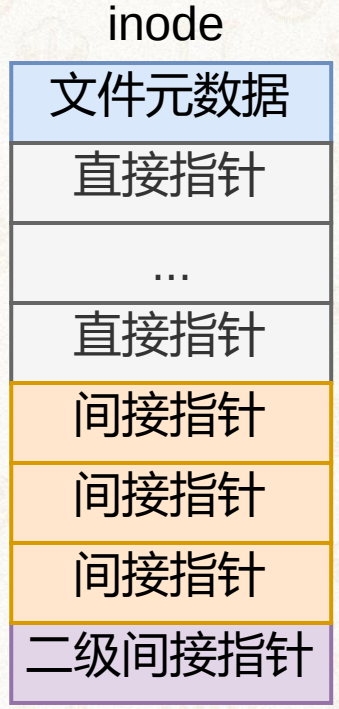
此题未设置答案，请点击右侧设置按钮

假设某一文件大小为62KB，其inode结构如下。当我们从头到尾依次读取文件内容时，最后访问的数据块在哪？



inode: 常规文件的数据索引方式

- 一个inode节点有多大：
  - 16个指针，每个8字节，共128字节
  - 实际是256字节
  - 剩下的128字节：元数据



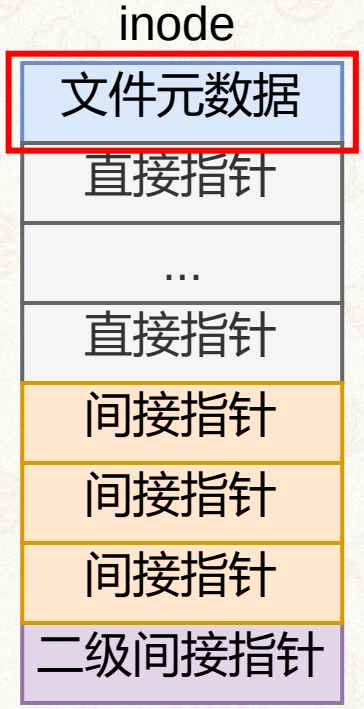

```
yxsu@Dell-T6401:/dev$ sudo dumpe2fs -h /dev/sda | grep Inode
dumpe2fs 1.45.5 (07-Jan-2020)
Inode count:              366256128
Inodes per group:         2048
Inode blocks per group:   128
Inode size:               256
```

inode: 常规文件的数据索引方式

- 元数据：
  - 文件类型
  - 文件大小
  - 文件权限
  - 拥有者(uid, gid)
  - 创建、修改、访问时间

这里的”块”是指硬盘里扇区的个数，单扇区一般为512字节

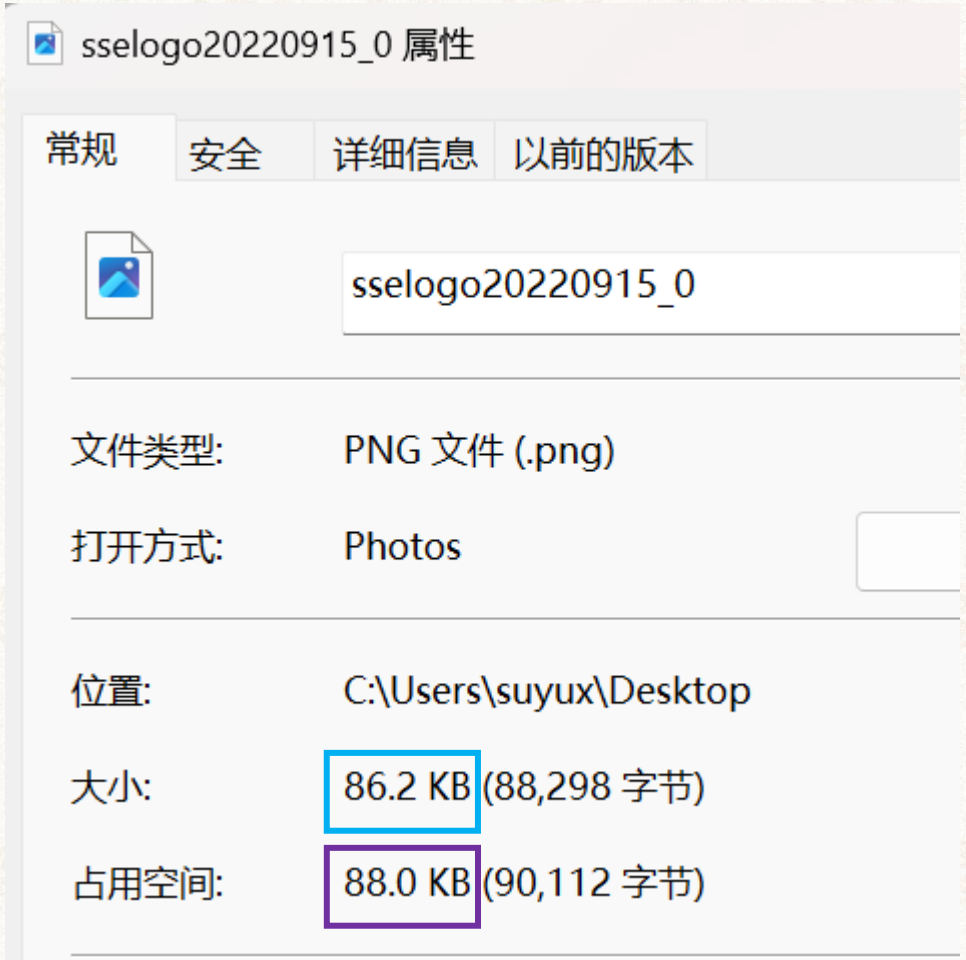
```
yxsu@Dell-T6401:~/os$ stat demo.cpp
  文件：demo.cpp
  大小：605           块：8          IO 块：4096   普通文件
设备：812h/2066d      Inode：20449913    硬链接：1
权限：(0664/-rw-rw-r--)  Uid：( 1002/    yxsu)   Gid：( 1002/    yxsu)
最近访问：2022-04-05 10:16:14.406914366 +0800
最近更改：2022-03-09 09:32:40.793590932 +0800
最近改动：2022-03-09 09:32:40.793590932 +0800
创建时间：-
```



日常使用文件时的怪事

- 现在是不是明白了：
- “大小”和“占用空间”为什么是两个概念？

```
yxsu@lg:~/Desktop$ stat sselogo20220915_0.png
  File: sselogo20220915_0.png
  Size: 88298           Blocks: 176        IO Block: 4096   regular file
Device: 53h/83d Inode: 5348024557517749  Links: 1
Access: (0777/-rwxrwxrwx)  Uid: ( 1000/    yxsu)   Gid: ( 1000/    yxsu)
Access: 2024-05-06 01:31:45.075709600 +0800
Modify: 2024-05-06 01:30:11.717151000 +0800
Change: 2024-05-06 01:30:20.564434200 +0800
 Birth: -
```



inode: 文件名与目录文件

- 一个文件对应一个inode，用inode编号即可找到文件
  - inode编号是给操作系统看的，而文件名是给人看的

```
yxsu@Dell-T6401:~/os$ ls -i process/
21105362 a.out                21105368 fork_demo          21105367 fork_readfile.c
21105366 myecho               21105365 test.txt           21105373 vfork_demo.c
21105364 execve_demo          21105363 fork_demo.c        21105355 hello-name
21105369 myecho.c             21105380 ucontext_demo.c    21105371 waitpid_demo.c
21105370 execve_demo.c        21105372 fork_readfile      21105361 hello-name.c
21105378 pthread_yield_demo.c 21105377 vfork_demo
```

- 所以inode的元数据里没有文件名

- 文件名是存在目录里的

inode: 文件名与目录文件

- 目录也是一个文件

```
yxsu@Dell-T6401:~$ stat os
  文件：os
  大小：4096          块：8          IO 块：4096   目录
设备：812h/2066d      Inode：20449904    硬链接：4
权限：(0775/drwxrwxr-x)  Uid：( 1002/   yxsu)   Gid：( 1002/   yxsu)
最近访问：2022-04-24 21:27:45.629216327 +0800
最近更改：2022-04-24 21:27:45.517214858 +0800
最近改动：2022-04-24 21:27:45.517214858 +0800
创建时间：-
```

- 套娃：目录名是存在哪里的？
  - 存在父目录里
  - …
  - 存在根目录：/

```
yxsu@Dell-T6401:~$ tree os
os
├── 9-midterm-preview-0413.pptx
├── 9-sync-0411.pdf
├── 实验2提交地址.txt
├── a.out
├── demo.cpp
├── fs
│   ├── copyfile
│   ├── copyfile2.c
│   └── copyfile.c
├── process
│   ├── a.out
│   ├── execve_demo
│   ├── execve_demo.c
│   ├── fork_demo
│   ├── fork_demo.c
│   ├── fork_readfile
│   ├── fork_readfile.c
│   ├── hello-name
│   ├── hello-name.c
│   ├── myecho
│   ├── myecho.c
│   ├── pthread_yield_demo.c
│   ├── test.txt
│   ├── ucontext_demo.c
│   ├── vfork_demo
│   ├── vfork_demo.c
│   └── waitpid_demo.c
└── sselogo_duiqi_new.png

2 directories, 26 files
```

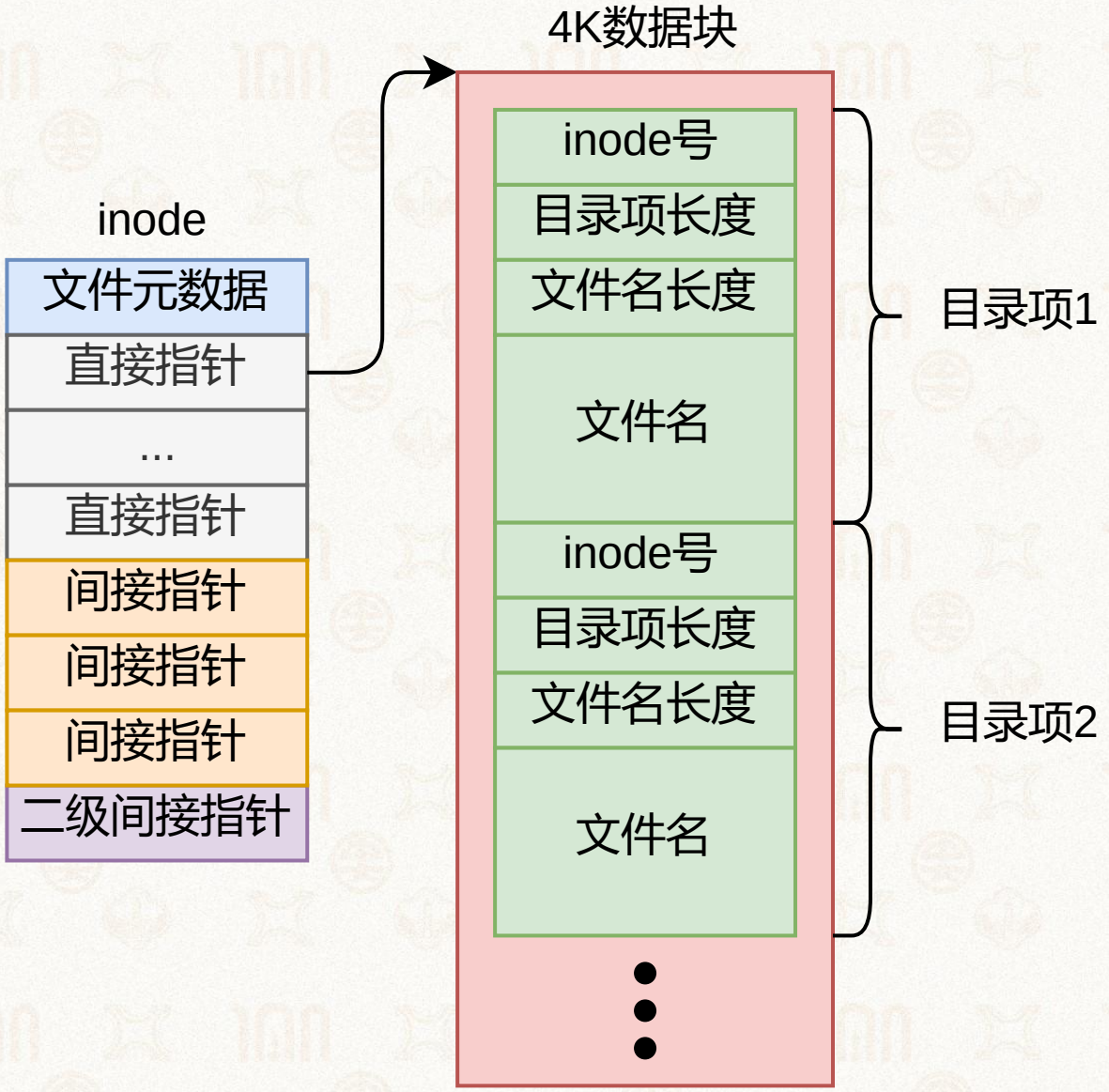
查看‘os’目录的结构： tree os

inode: 文件名与目录文件

- 目录项的结构

- 目录中查找文件：
  - 逐个遍历文件名

- 删除文件：
  - 把inode号置为0
  - 如果连续两个都为0，可以合并

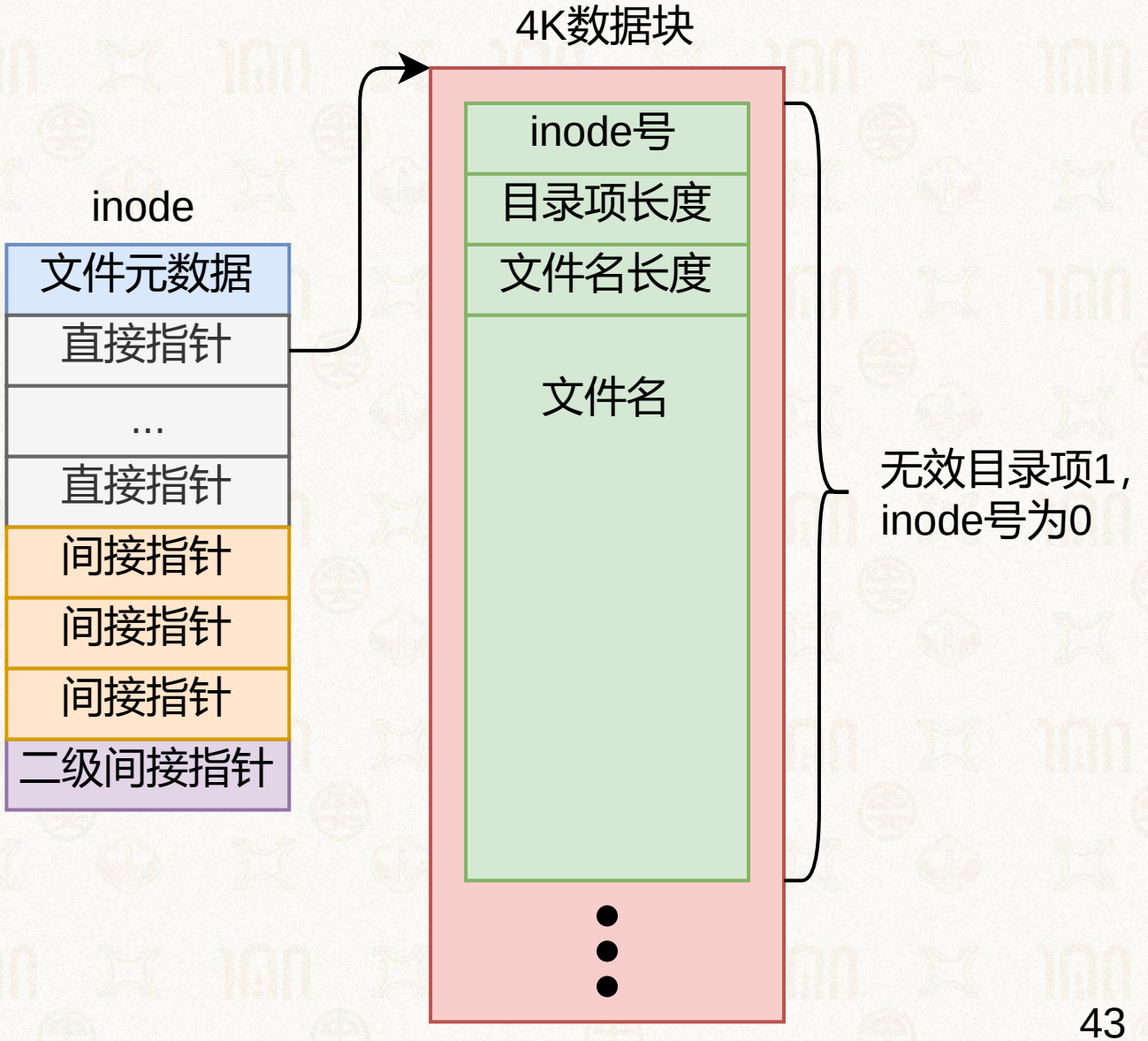


inode: 文件名与目录文件

- 目录项的结构

- 目录中查找文件：
  - 逐个遍历文件名

- 删除文件：
  - 把inode号置为0
  - 如果连续两个都为0，可以合并



单选题 1分

设置

假设文件系统中有两个目录。其中一个保存 10 个电影文件（movie1.mp4 ~ movie-10.mp4），另一个保存了 100 个笔记文件（note-1.md ~ note-100.md），哪个目录占的存储空间更大？

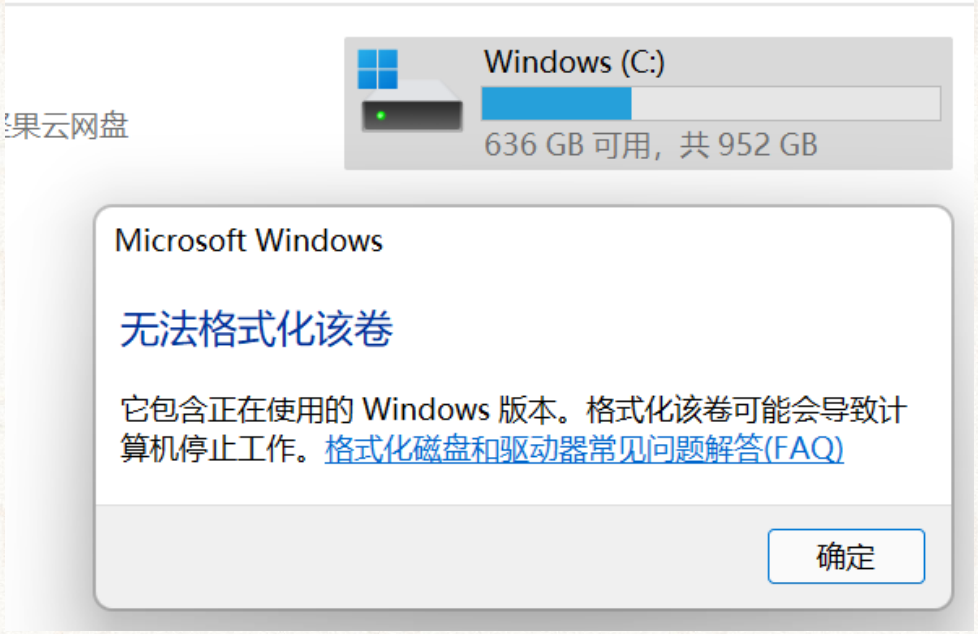
10 个电影文件的目录

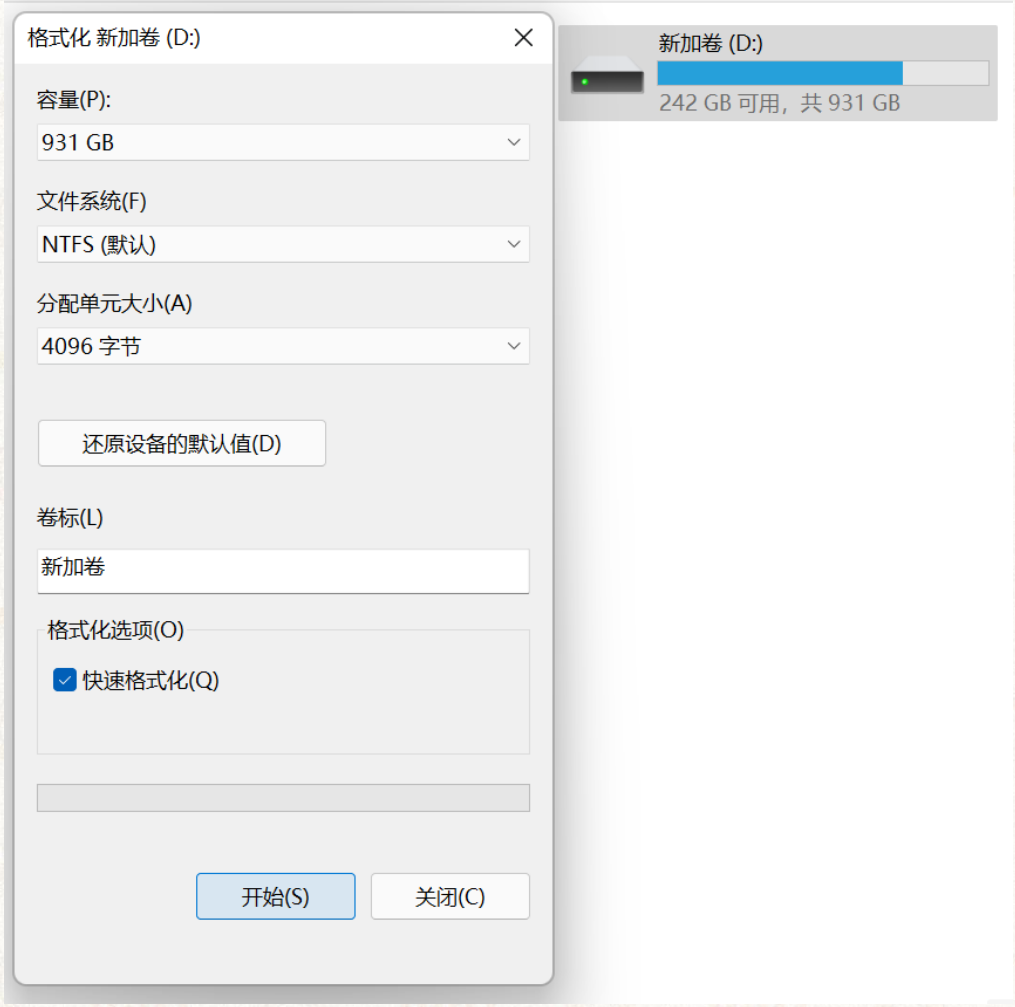
100 个笔记文件的目录

提交

文件系统的存储布局

- 整套文件系统在硬盘格式化时创建存储布局





基于inode文件系统的存储布局

- 在整块硬盘上统一分配inode节点信息、数据块信息

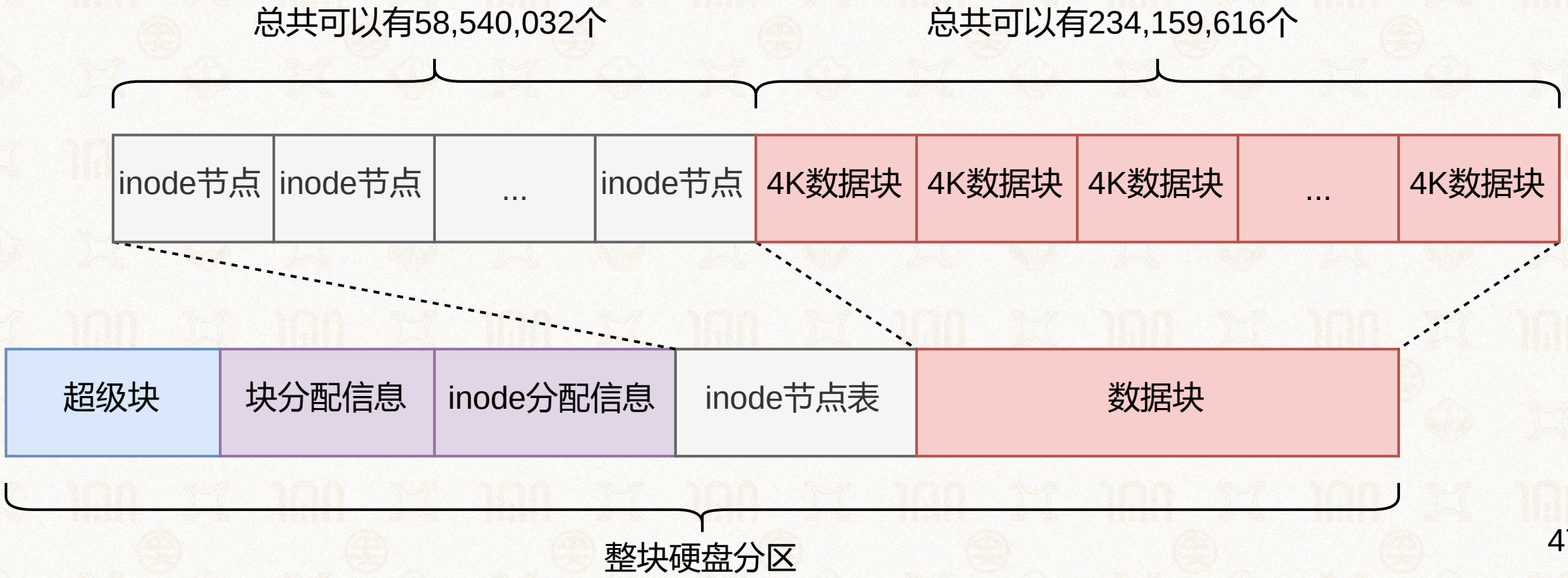
- 超级块(super block) 作为文件系统的元数据，存储基本信息
  - 魔法数字(magic number): 识别文件系统类型
  - inode节点数量
    - 决定最多能存多少个文件
  - 管理数据空间大小

```
yxsu@Dell-T6401:/dev$ sudo dumpe2fs -h /dev/sdb2
dumpe2fs 1.45.5 (07-Jan-2020)
Filesystem volume name:   <none>
Last mounted on:          /
Filesystem UUID:          f559efda-9c55-414b-9310-160297c78503
Filesystem magic number:  0xEF53
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              58540032
Block count:              234159616
Reserved block count:     11707980
Free blocks:              204605872
Free inodes:              57440401
First block:              0
Block size:               4096
Fragment size:            4096
Flex block group size:    16
Filesystem created:       Fri Dec  3 22:04:39 2021
Last mount time:          Wed Apr 20 10:14:30 2022
Last write time:          Wed Apr 20 10:14:28 2022
```

基于inode文件系统的存储布局

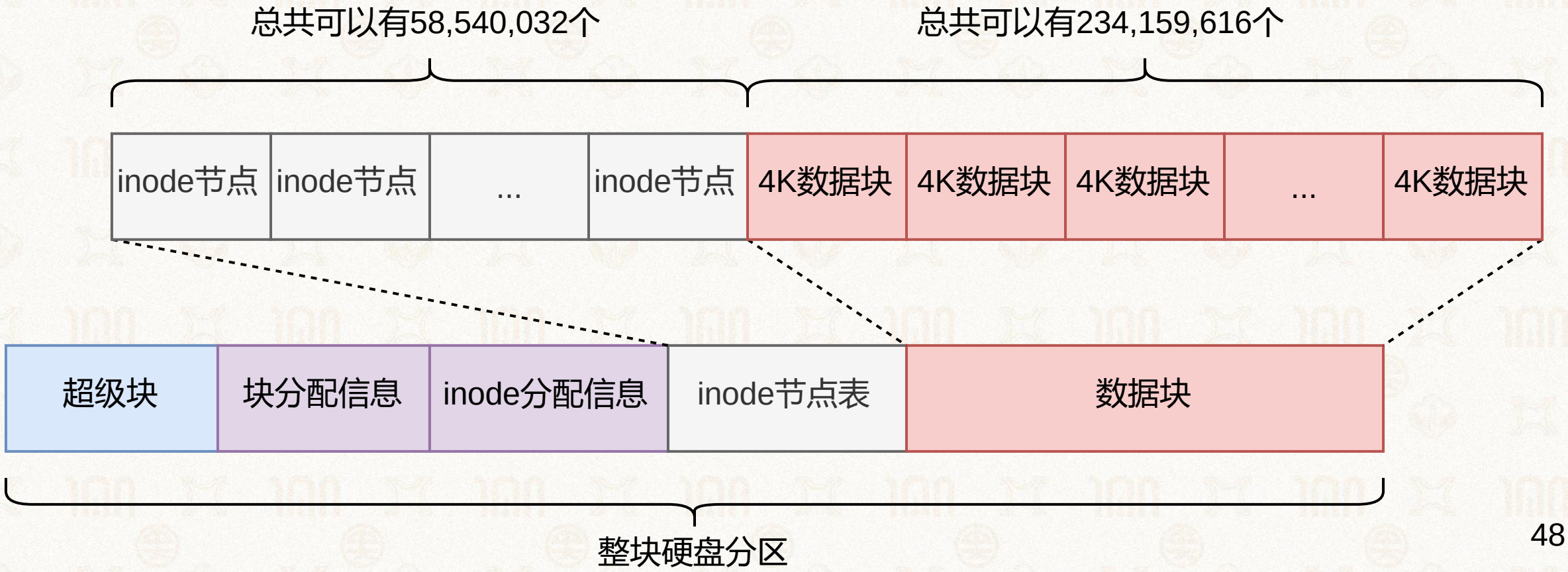
- 以服务器上“sdb2”的分区为例：

```
yxsu@Dell-T6401:/dev$ sudo dumpe2fs -h /dev/sdb2 | grep count
dumpe2fs 1.45.5 (07-Jan-2020)
Inode count:              58540032
Block count:              234159616
Reserved block count:     11707980
```



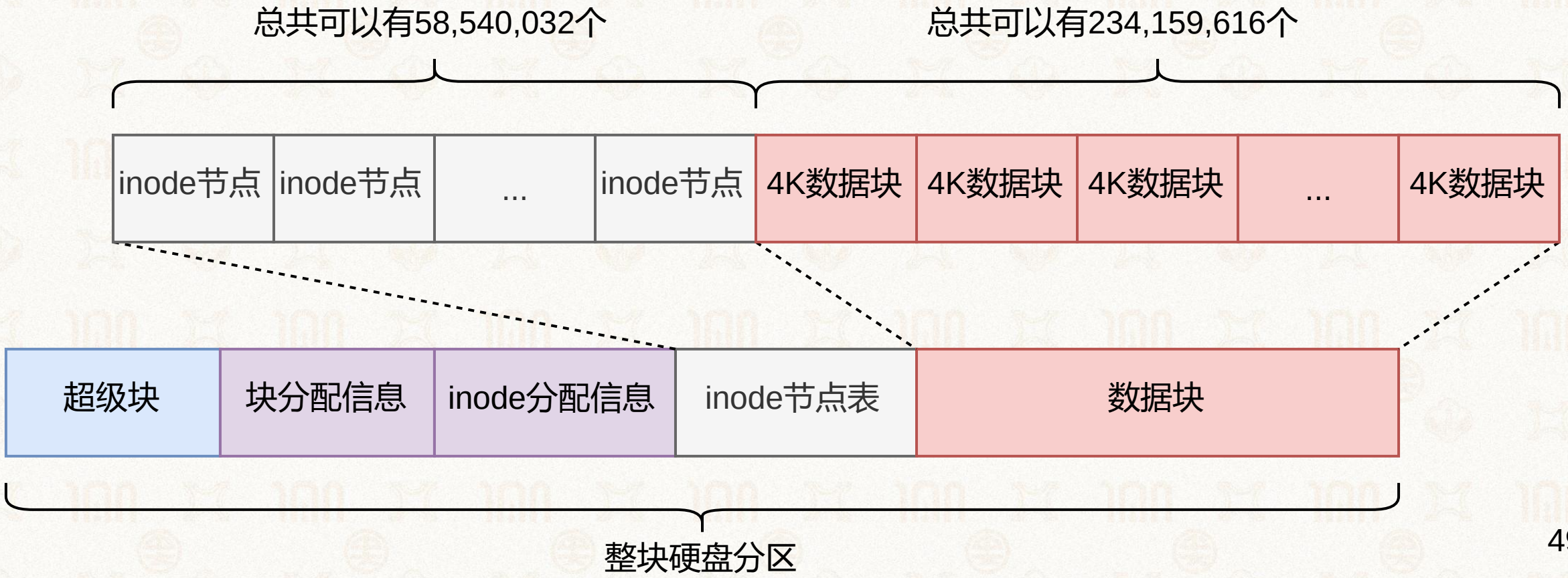
基于inode文件系统的存储布局

- 数据块能表示的空间有多大?
- 4K x 234,159,616 = 937G



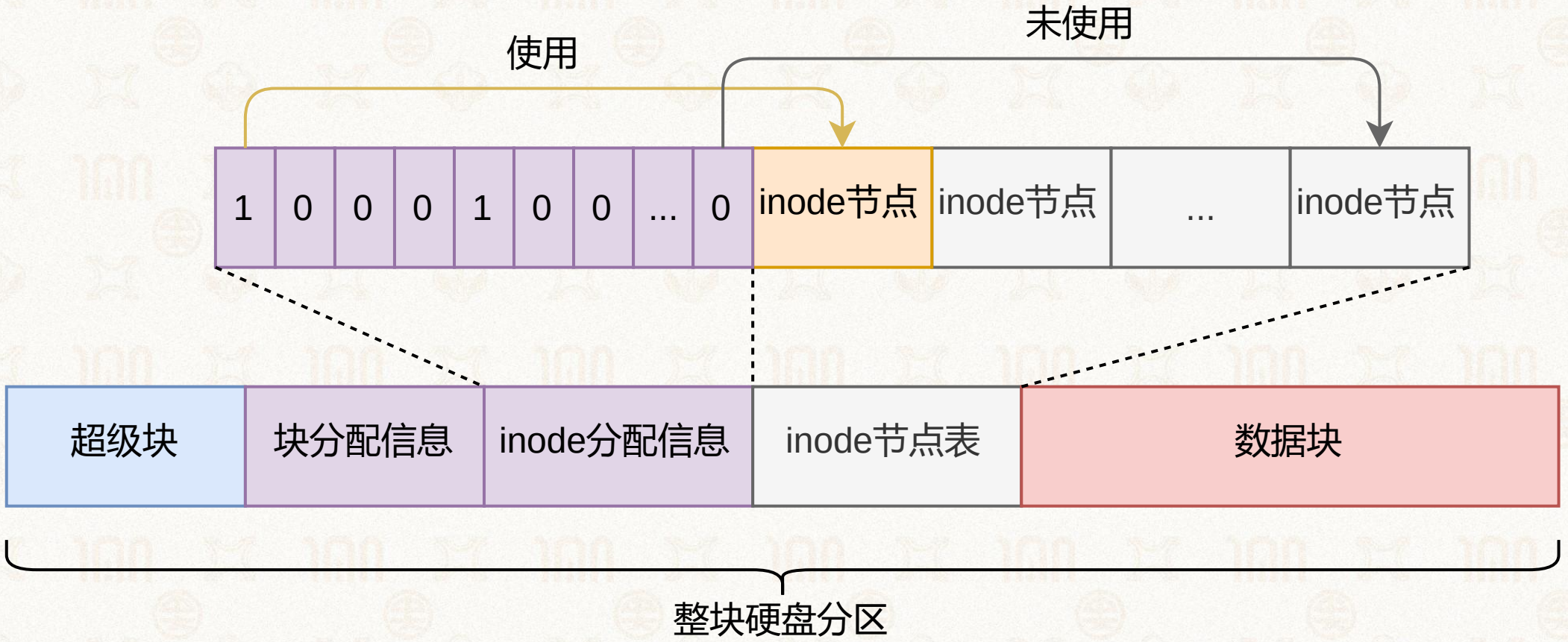
基于inode文件系统的存储布局

- inode节点总共占多少空间?
- 256B x 58,540,032 = 15G



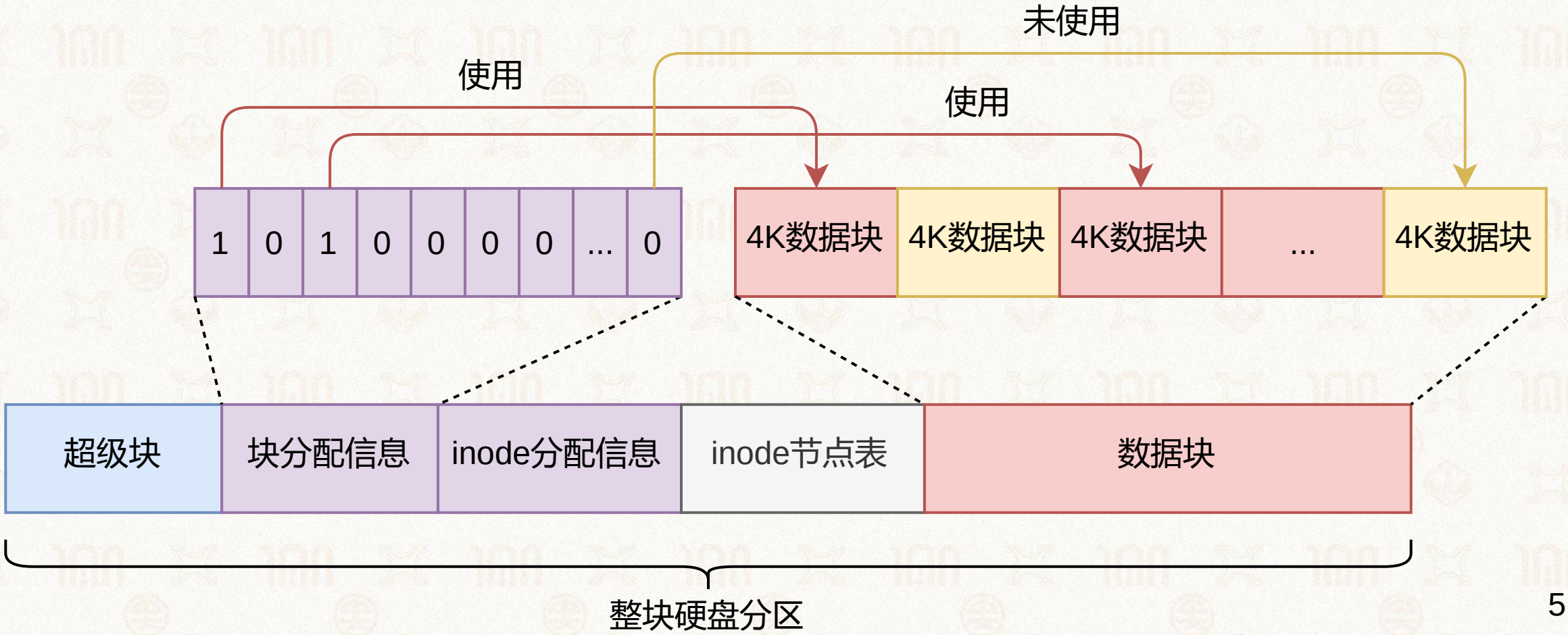
基于inode文件系统的存储布局

- inode分配信息：用位图表示对应的节点是否使用(1表示使用)
- inode分配信息要占多少空间：
- 58,540,032 / 8 = 7.3M



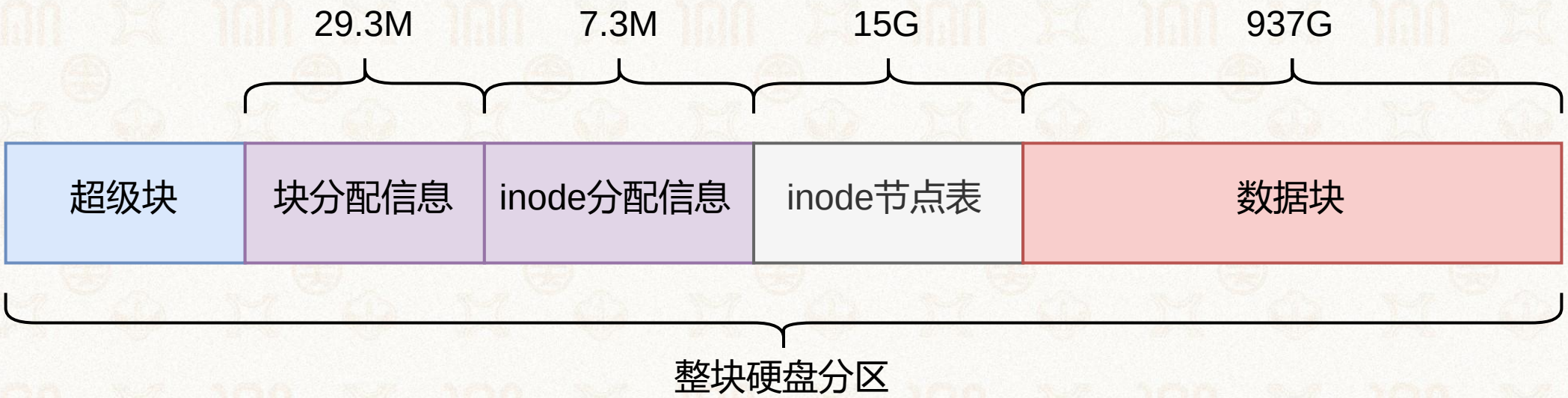
基于inode文件系统的存储布局

- 块分配信息：用位图表示对应的节点是否使用(1表示使用)
- 块分配信息要占多少空间：
- 234,159,616 / 8 = 29.3M



基于inode文件系统的存储布局

- 谁偷了你的硬盘空间?
- 文件系统总空间为：952G，但只能存937G有效数据
- 这样布局的优点是灵活高效，浪费点空间是值得的
- 针对不同大小的硬盘，每次格式化自动推算出这些布局数据

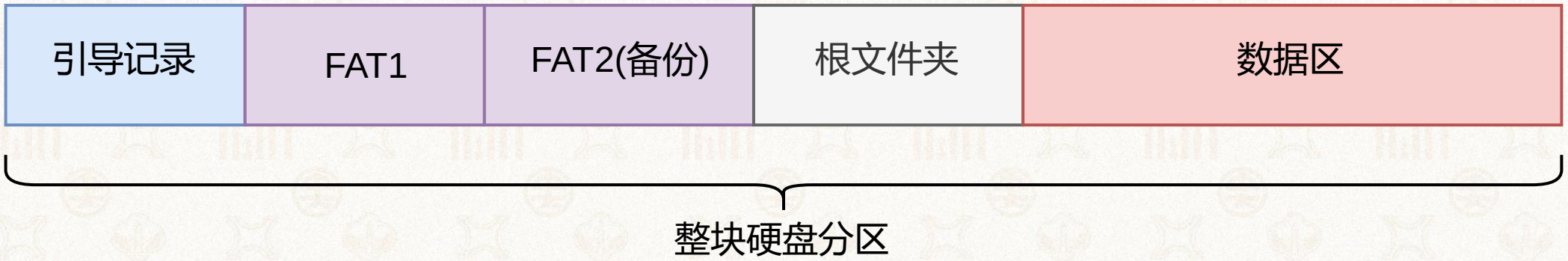


文件系统大纲

- 块设备
- 基于inode的文件系统
- 基于表的文件系统
  - FAT
  - NTFS
- 使用文件系统
  - 以命令行的形式使用
  - 在代码里使用
    - 基本操作
    - 内存映射
  - 文件系统的高级功能
- 虚拟文件系统
- 用户态文件系统

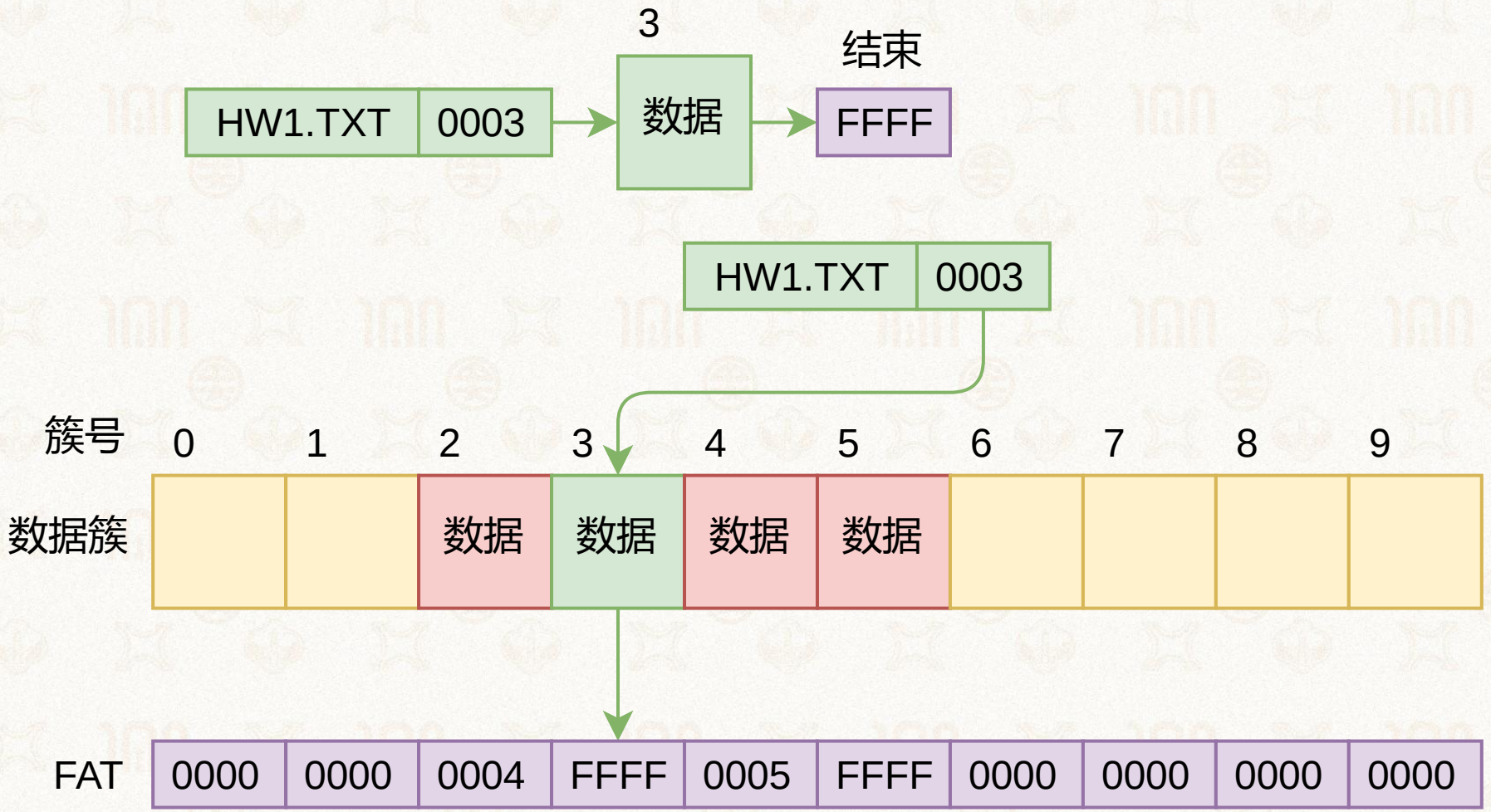
FAT：文件分配表(File Allocation Table)

- Windows系统从1985年起使用至今
- 引导记录区与超级块相似
- 数据基本块的单位为“簇”为单位，与inode中的块相同
- FAT是簇序号的数组，表示下一个簇
  - 文件由簇的链表形式组织



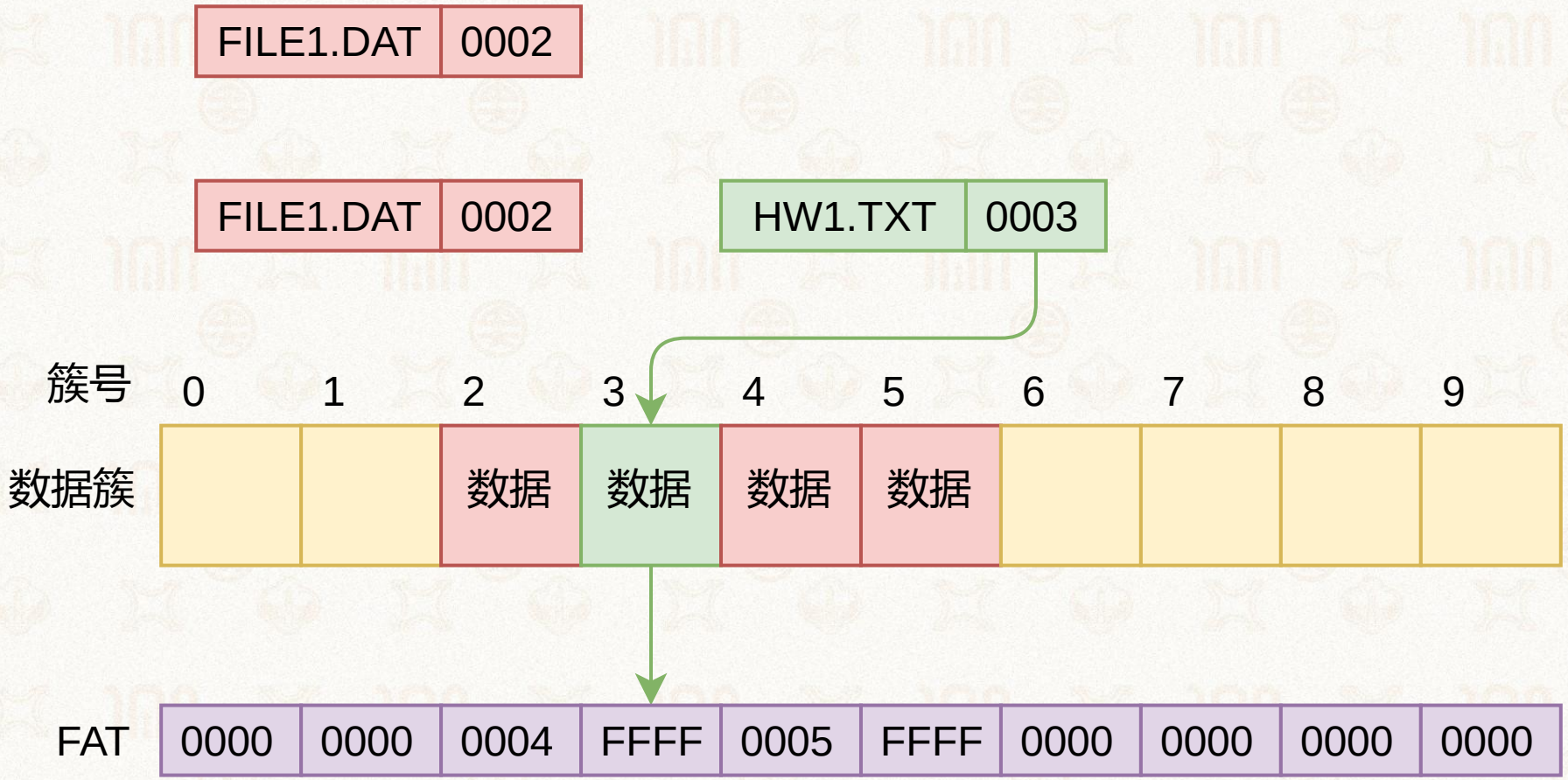
FAT：文件的链表组织形式

- 目录项中存有文件名和起始簇号



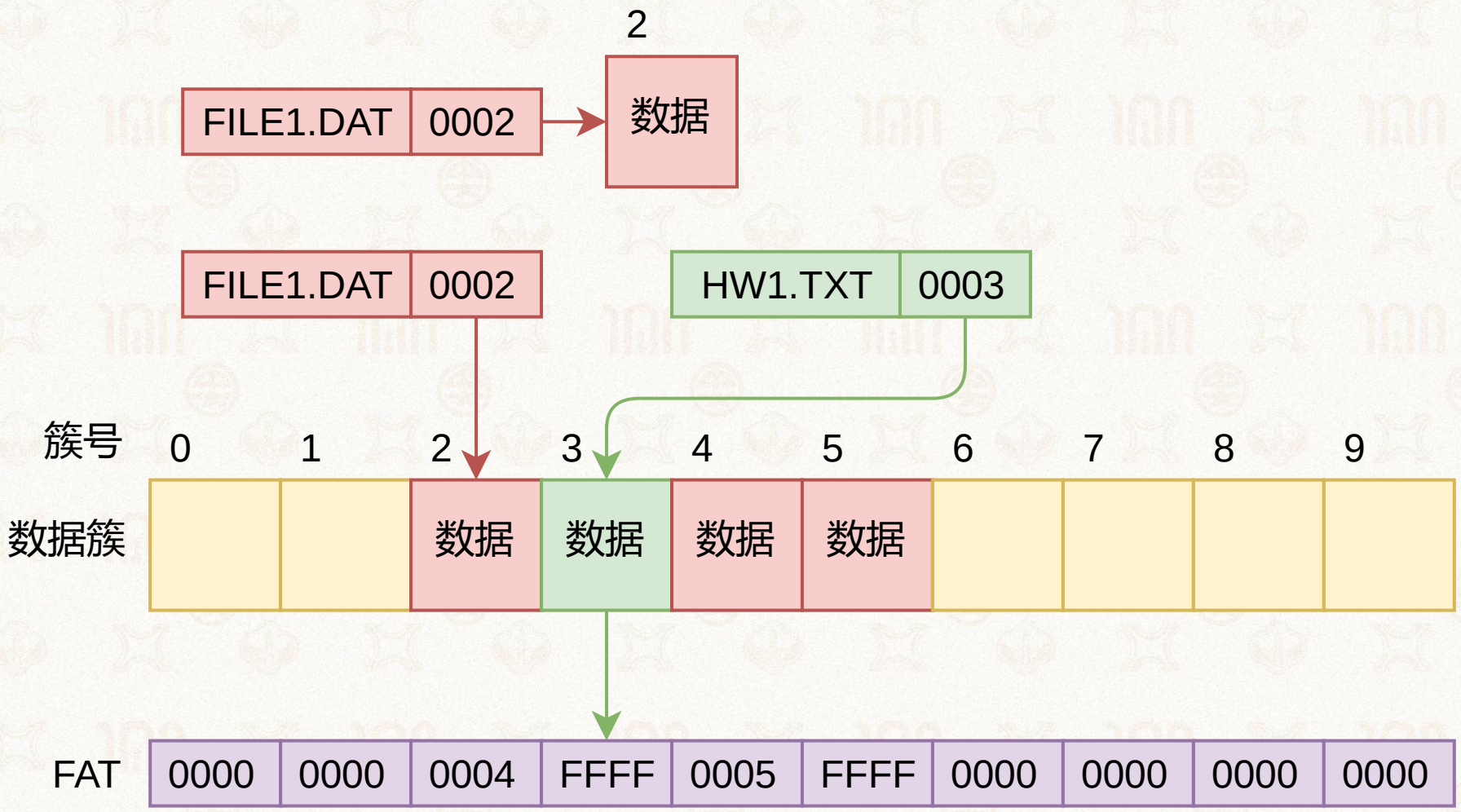
FAT：文件的链表组织形式

- 目录项中存有文件名和起始簇号



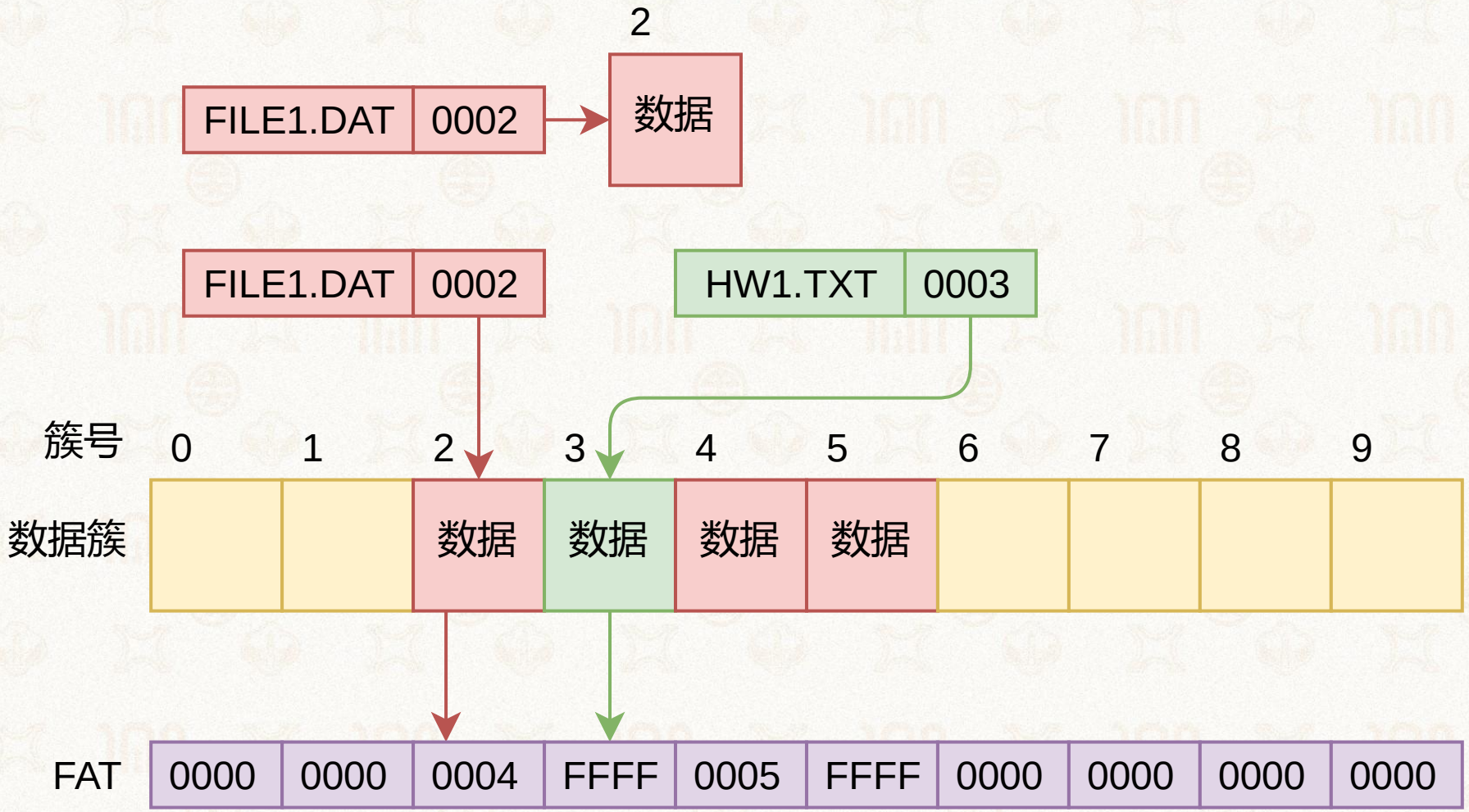
FAT：文件的链表组织形式

- 读起始数据簇



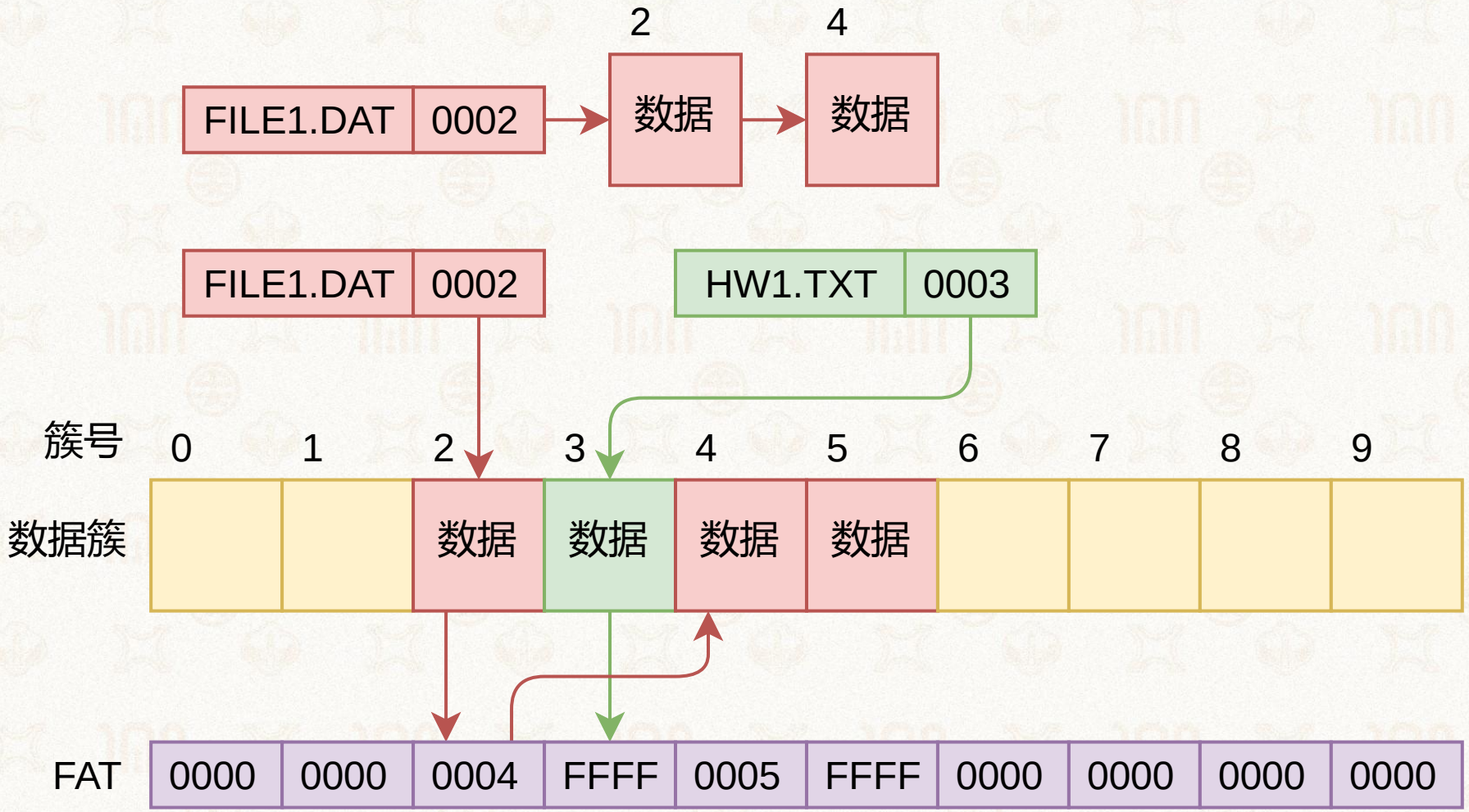
FAT：文件的链表组织形式

- 读FAT表中对应的项



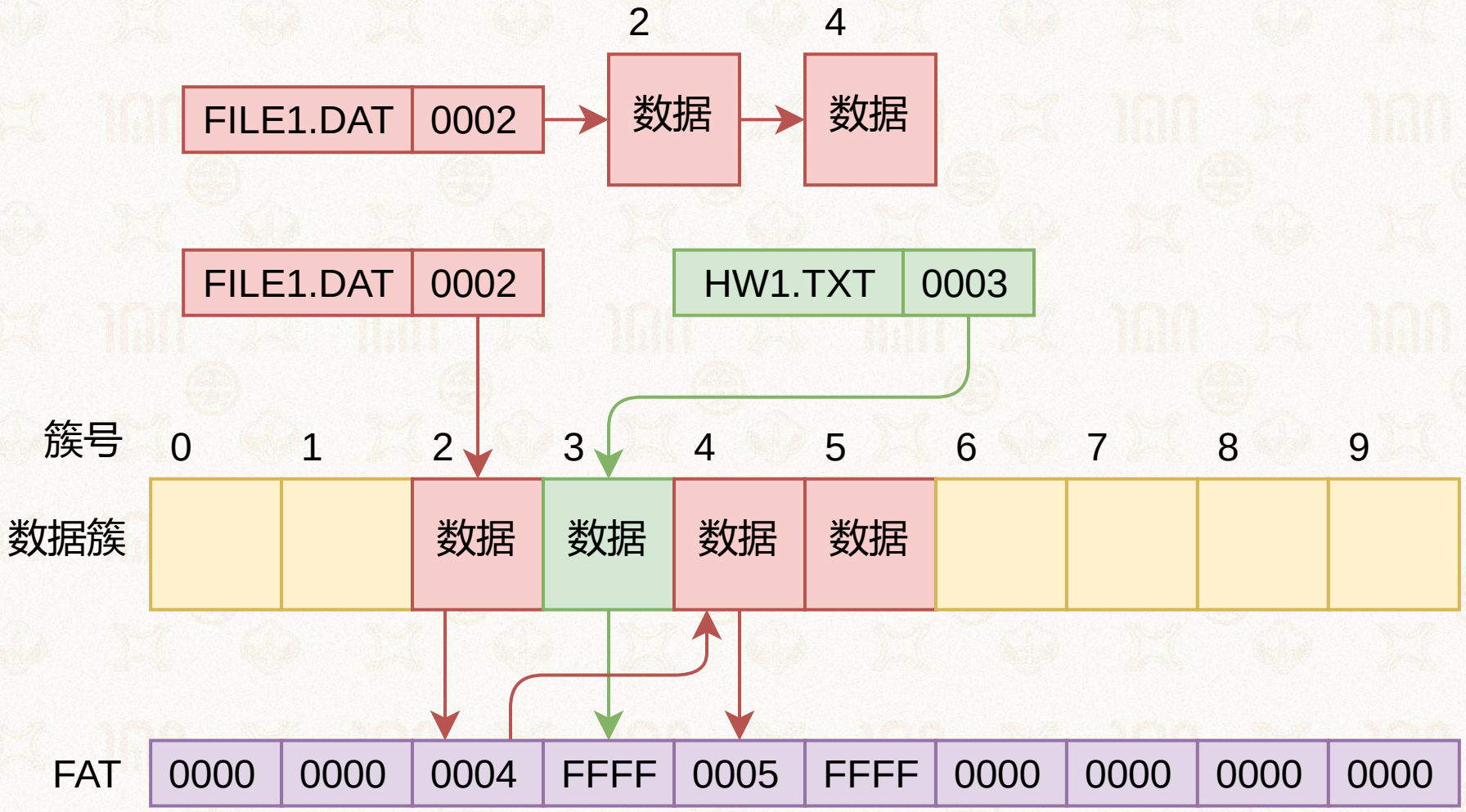
FAT：文件的链表组织形式

- 读FAT表中对应的项，得知下一个簇号是0004



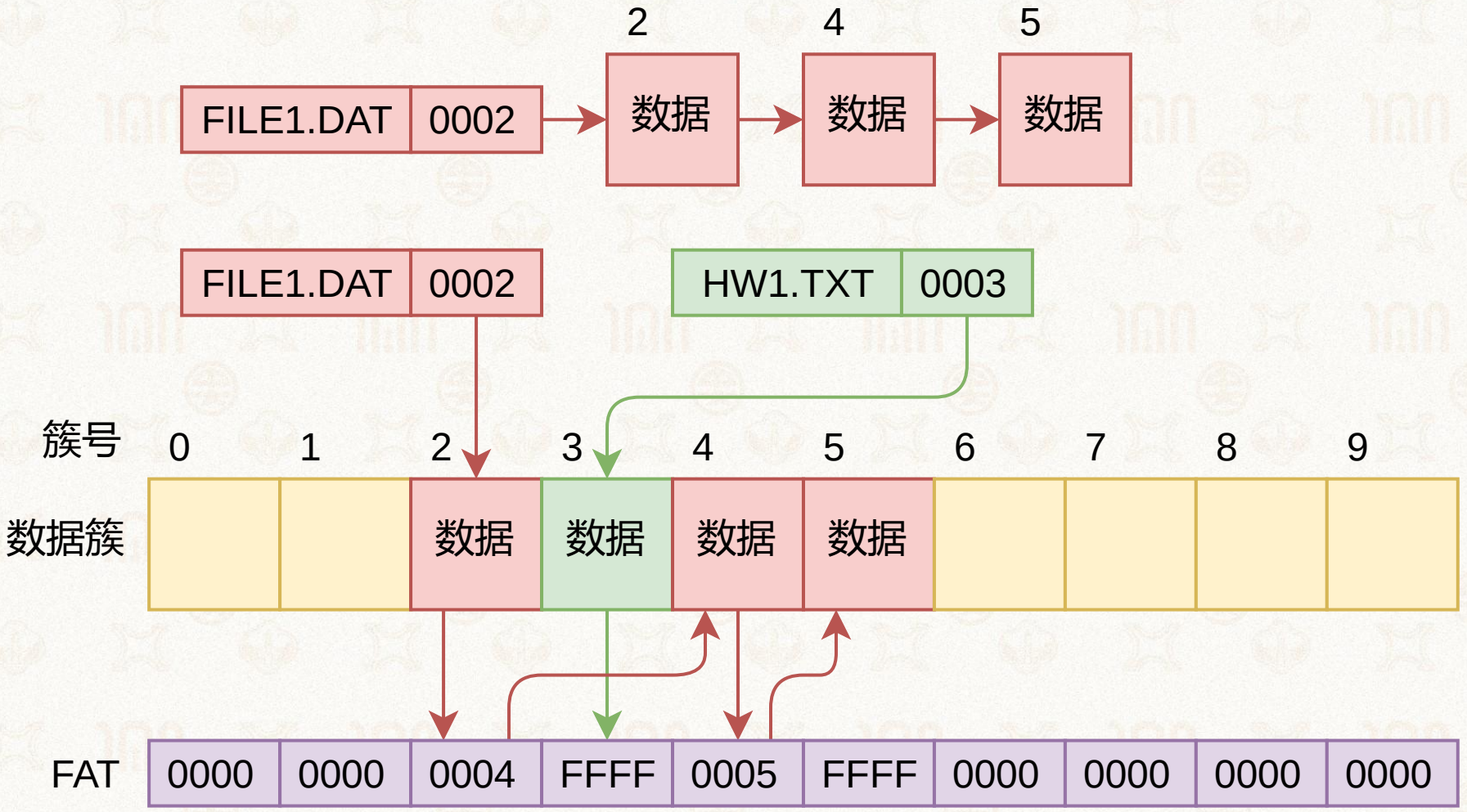
FAT：文件的链表组织形式

- 读FAT表中对应的项



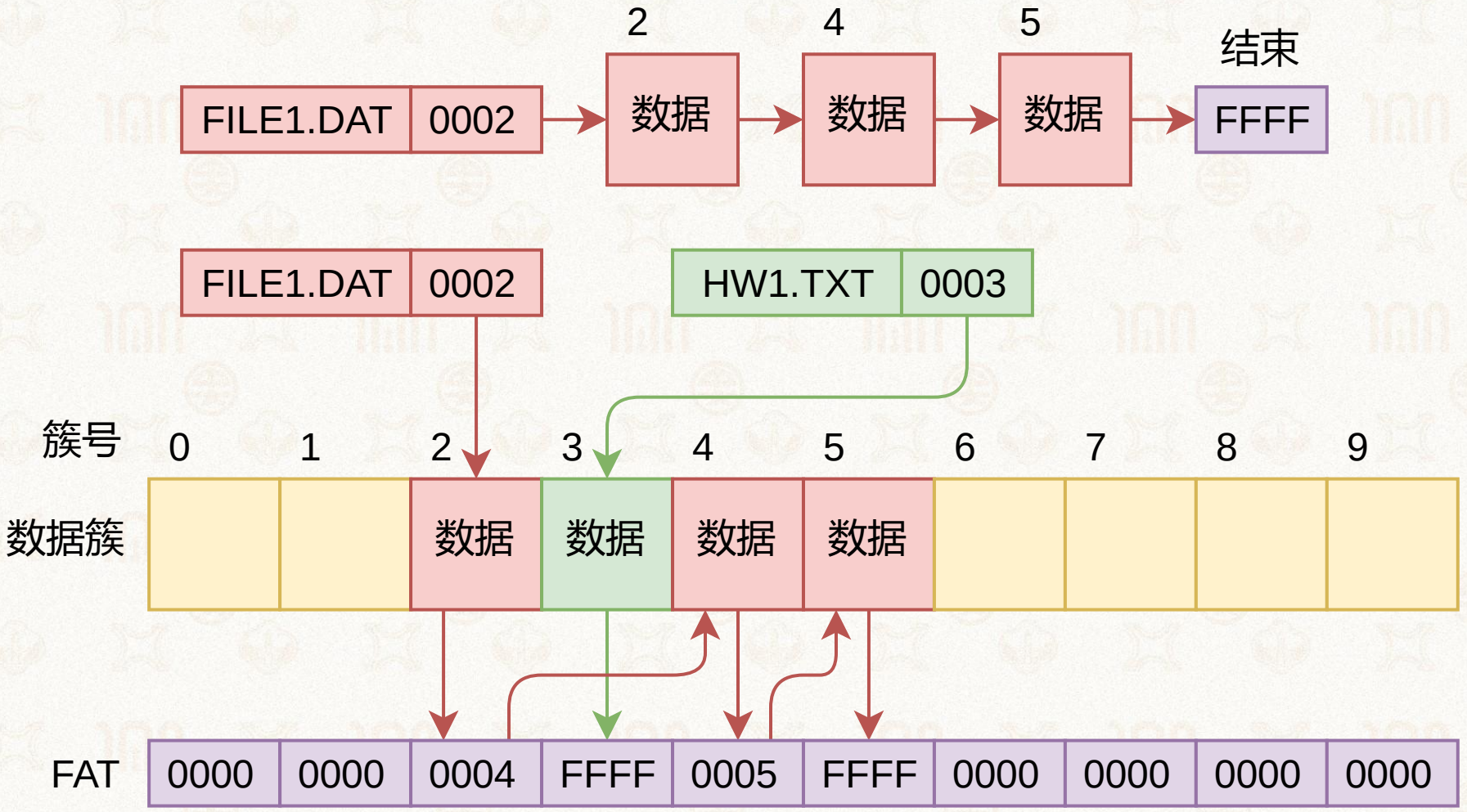
FAT：文件的链表组织形式

- 读FAT表中对应的项，得知下一个簇号是0005



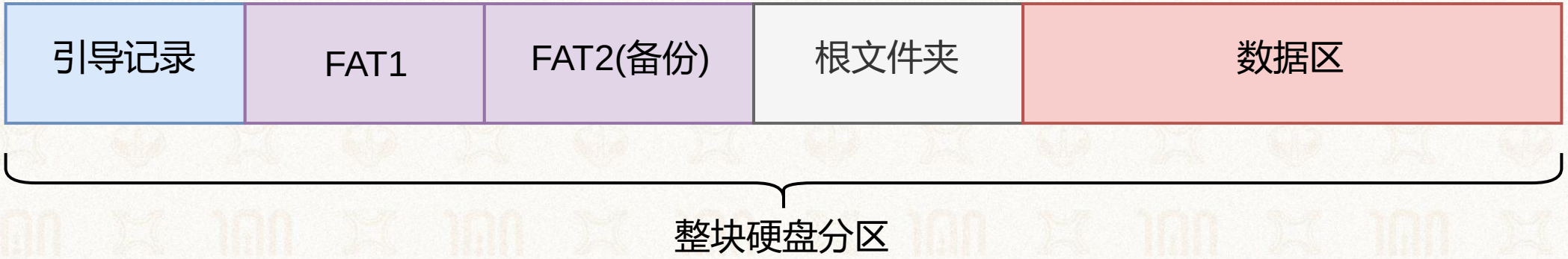
FAT：文件的链表组织形式

- 读FAT表中对应的项，发现该结束了(FFFF表示结尾)



FAT：文件的链表组织形式

- 实现简单，应用在多种设备中
- 链表容易断，所以又创建了一模一样的FAT2当备份
- 遍历链表，文件越大，读写越慢
- FAT32中，“文件大小”元数据用32位整数表示，因此文件最大为$2^{32} = 4G$

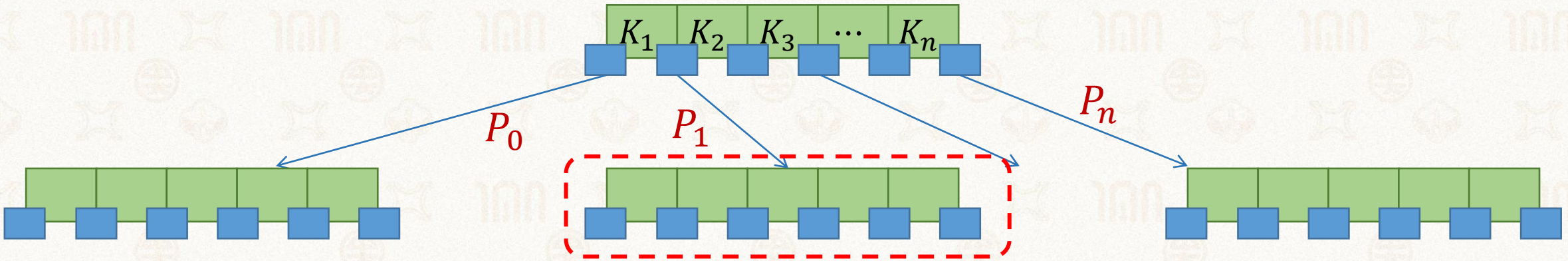


exFAT：U盘常用格式

- 与FAT32并不兼容
- 使用位图加快空间分配
- Unicode保存长文件名
- 目录查找文件时使用哈希对比
- 允许4GB以上文件(文件大小用8字节)
- 使用校验码保证元数据完整性

设置

一颗m阶B树(Balanced tree of order m)是一颗高度平衡的m叉查找树。那么红框范围内的所有关键字都 [填空1] 。



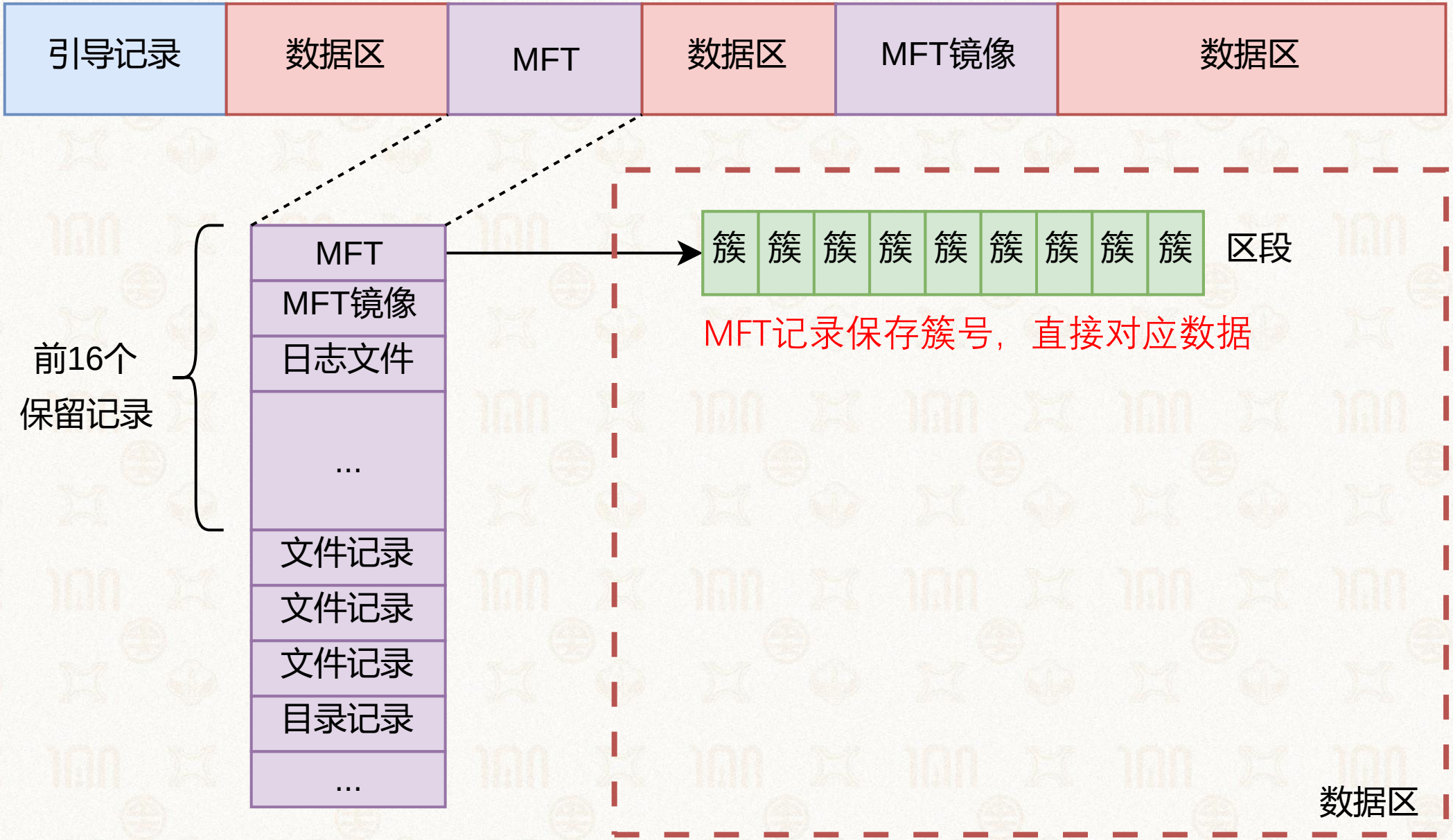
作答

NTFS: New Technology File System

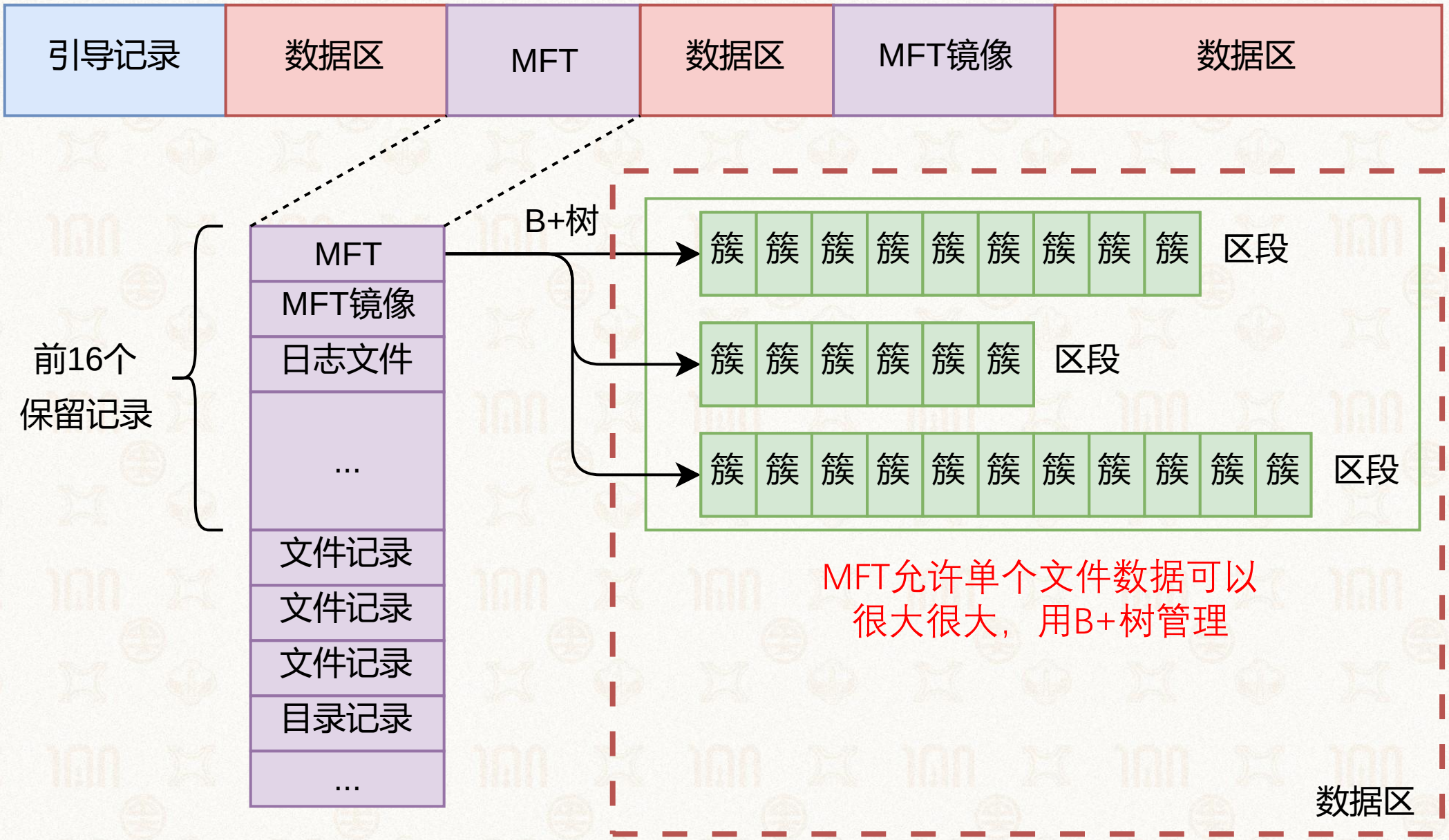
前16个保留记录均为元数据，记录基本信息

- 现代Windows系统中广泛使用的文件系统
- 核心结构主文件表(Master File Table, MFT)

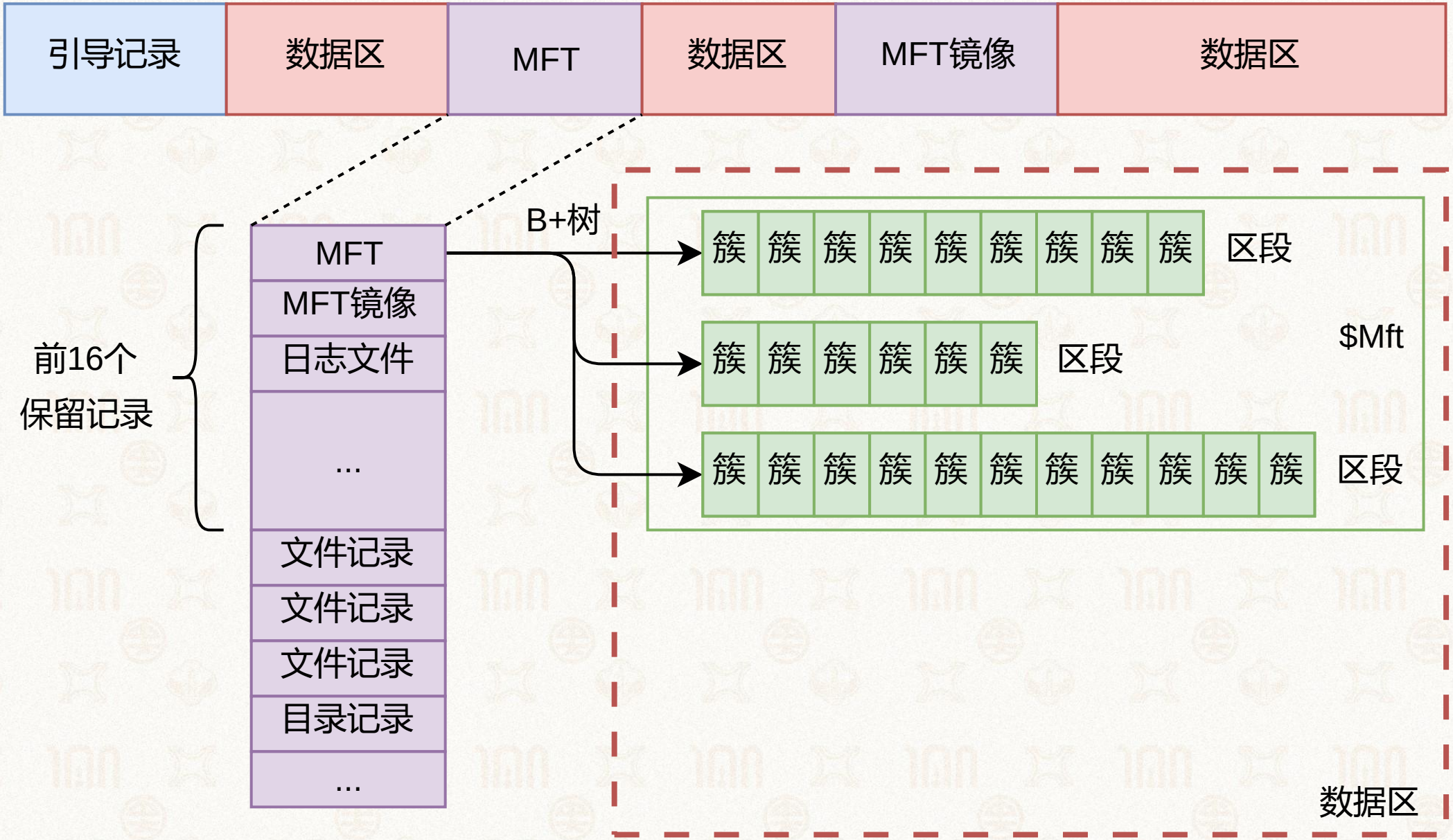
NTFS存储布局


引导记录
数据区
MFT
数据区
MFT镜像
数据区
MFT
MFT镜像
日志文件
...
文件记录
文件记录
文件记录
目录记录
...
前16个
保留记录
簇 簇 簇 簇 簇 簇 簇 簇 簇
区段
MFT记录保存簇号，直接对应数据
数据区

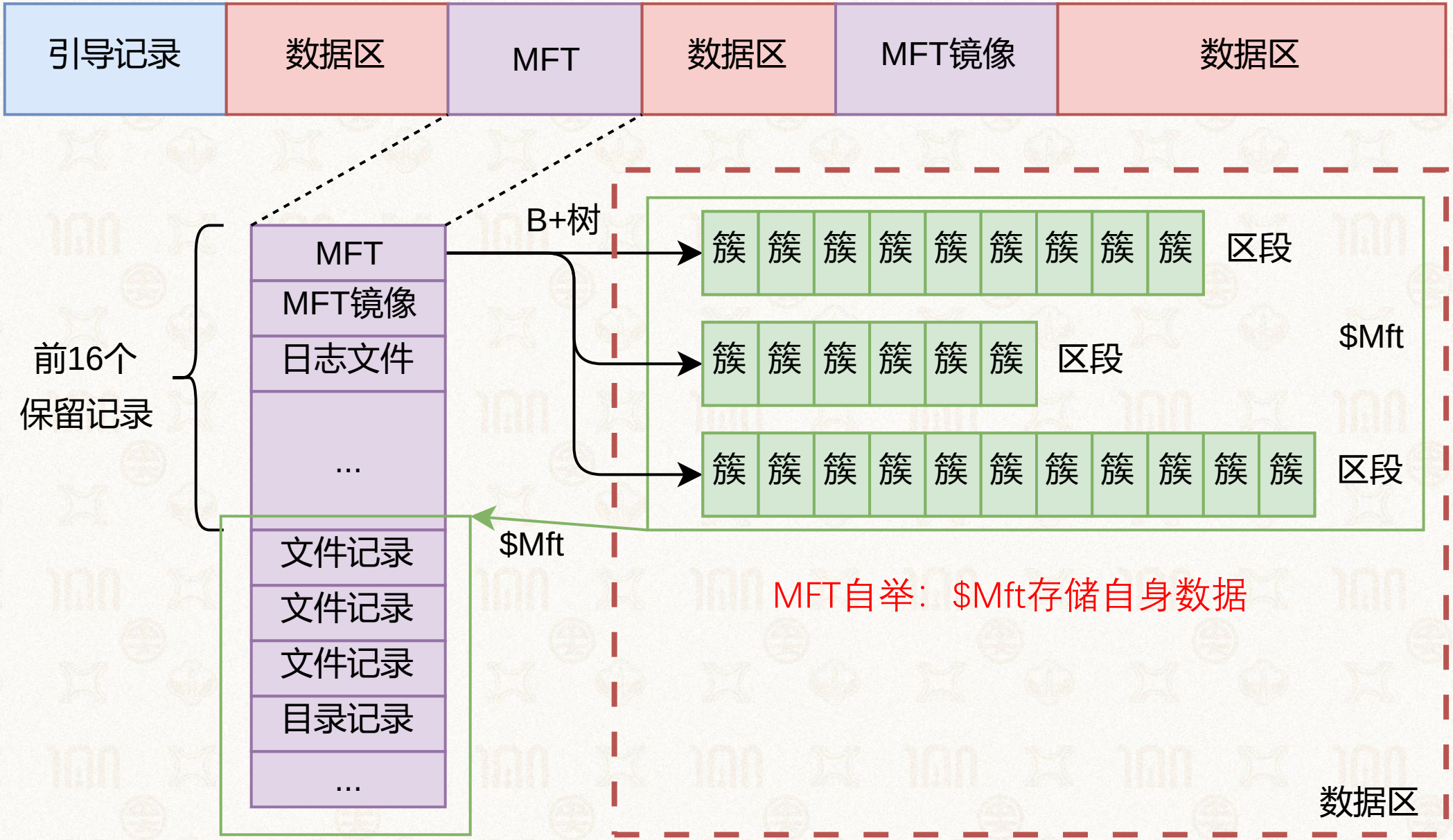
NTFS存储布局


引导记录
数据区
MFT
数据区
MFT镜像
数据区
前16个
保留记录
MFT
MFT镜像
日志文件
...
文件记录
文件记录
文件记录
目录记录
...
B+树
簇 簇 簇 簇 簇 簇 簇 簇 簇
区段
簇 簇 簇 簇 簇 簇
区段
簇 簇 簇 簇 簇 簇 簇 簇 簇 簇 簇
区段
MFT允许单个文件数据可以
很大很大，用B+树管理
数据区

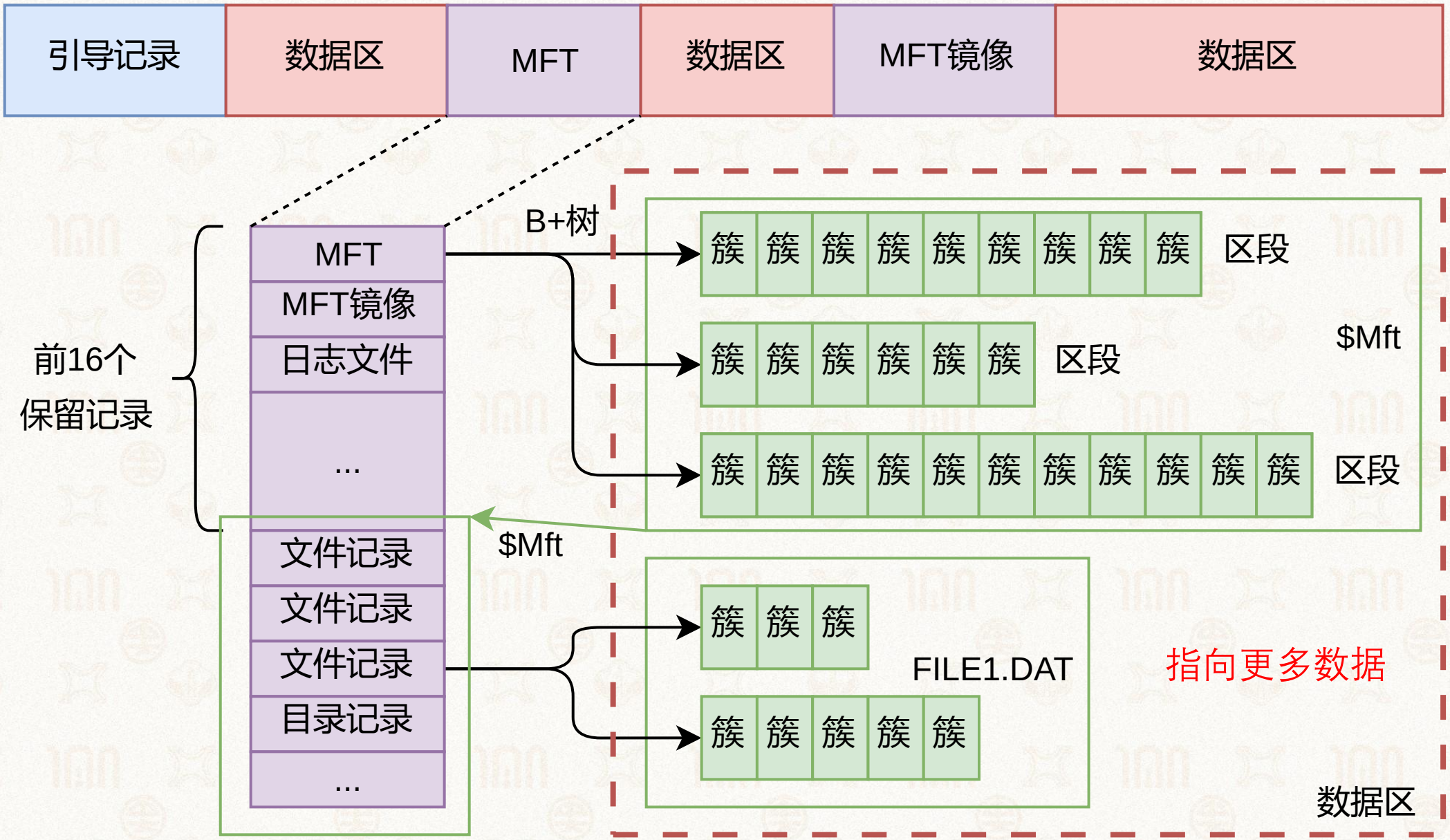
NTFS存储布局


引导记录
数据区
MFT
数据区
MFT镜像
数据区
前16个
保留记录
MFT
MFT镜像
日志文件
...
文件记录
文件记录
文件记录
目录记录
...
B+树
簇
区段
$Mft
数据区

NTFS存储布局


引导记录
数据区
MFT
数据区
MFT镜像
数据区
前16个
保留记录
MFT
MFT镜像
日志文件
...
文件记录
文件记录
文件记录
目录记录
...
B+树
$Mft
簇 簇 簇 簇 簇 簇 簇 簇 簇
区段
簇 簇 簇 簇 簇 簇
区段
簇 簇 簇 簇 簇 簇 簇 簇 簇 簇 簇
区段
$Mft
MFT自举：$Mft存储自身数据
数据区

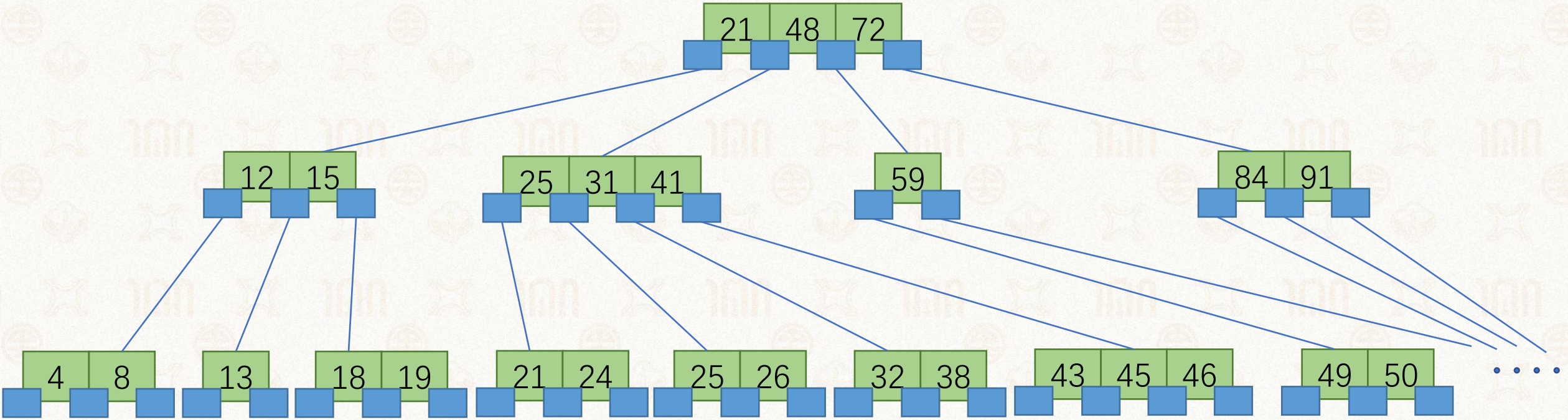
NTFS存储布局


引导记录
数据区
MFT
数据区
MFT镜像
数据区
前16个
保留记录
MFT
MFT镜像
日志文件
...
文件记录
文件记录
文件记录
目录记录
...
B+树
$Mft
簇
区段
$Mft
FILE1.DAT
指向更多数据
数据区

主观题 10分

设置

为什么文件系统喜欢用B+树或B树？
（B+树的关键字只在叶子节点，其它特征与B树相同）



作答

谢谢

微信：suyuxin
钉钉：苏玉鑫
B站：https://space.bilibili.com/502854403
软工集市课程专区：https://ssemarket.cn/new/course
匿名提问箱：https://suask.me/ask-teacher/106/苏玉鑫