第六章

文件管理

方　钰

主 要 内 容

什么是文件?

文件是具有文件名的一组相关信息的集合。

通常，文件由若干个记录组成。

系统或用户可以将一个程序或一组数据命名为一个文件。

操作系统中与管理文件有关的软件和数据统称为**文件管理系统**（简称文件系统）。

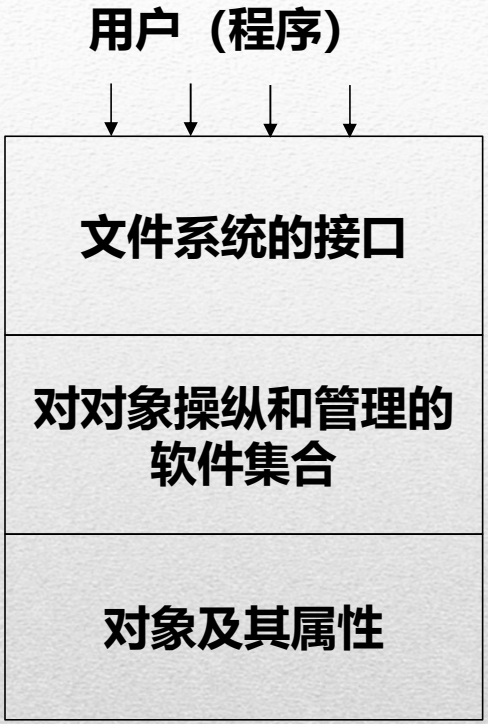
从系统角度，文件系统是对文件的存储空间进行组织、分配，负责文件的存储并对存储的文件进行保护、检索的系统。

从用户角度，文件系统主要实现了对文件的按名存取。

文件系统的功能

1. 按用户要求创建或删除文件；
2. 按用户要求进行文件读写；
3. 用户使用文件符号名实现文件访问，文件的物理组织对用户是透明的；
4. 管理文件存储空间，自动分配，建立文件逻辑结构以及物理结构之间的映照关系；
5. 共享和保密。

文件系统模型



命令接口：用户与文件系统（命令）
程序接口：用户程序与文件系统（系统调用）

文件存储空间的管理、文件目录的管理
文件逻辑地址到物理地址的转换、
文件读写管理、文件的共享与保护等

目录：方便文件的存取和检索（*文件名→文件物理地址*）
文件：　文件管理的直接对象
存储空间：提高利用率和文件存取速度

主 要 内 容

6.1 文件系统概述

6.2 UNIX文件系统接口

6.3 UNIX文件系统的物理结构

6.4 UNIX文件系统的打开结构

6.5 UNIX文件系统的目录管理

6.6 UNIX文件系统的读写操作

文件系统的用户界面

文件主　文件主同组用户　其他用户

最低9位：R W E　R W E　R W E

文件的创建

fd = creat (name, mode)

打开文件标识数　新文件名　新文件的工作方式，包括其文件类型和用户对新文件的访问权限

文件创建成功后，可以使用fd对文件进行存访

例：fd = creat("/usr/Jessy", 0666);

文件的打开和关闭

fd = open (name, mode)

文件标识符　文件名　打开方式

例：fd = open("/usr/Jessy",01);

close (fd)

文件标识符

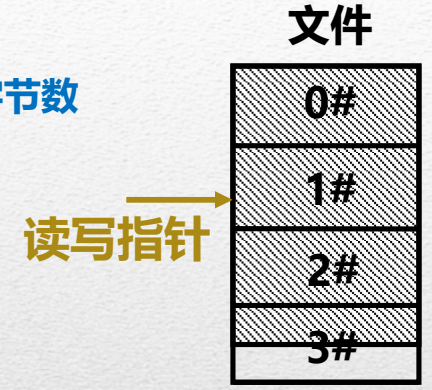
文件系统的用户界面

文件的顺序读写

n = read (fd, buf, nbytes)

- n：实际读取字节数
- fd：文件标识符
- buf：进程地址空间中存放读回数据的首址
- nbytes：读取的字节数

例：count = read(fd, data2, 12);



n = write(fd, buf, nbytes)

- n：实际写入字节数
- fd：文件标识符
- buf：进程地址空间中存放写入文件数据的首址
- nbytes：写入的字节数

例：count = write(fd, data1, 12);

默认：一次读写的起始位置是上次读写结束位置的下一个字节

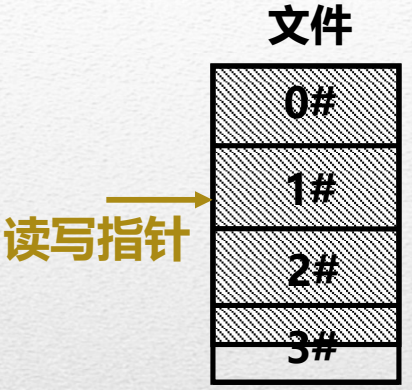
文件系统的用户界面

文件的随机存取

seek (fd, offset, ptrname)

文件标识符　配合调整文件读取位置

例：seek(fd,5,0);



默认：一次读写的起始位置是上次读写结束位置的下一个字节

ptrname = 0：读写指针位置设置为 offset （正）

ptrname = 1：读写指针位置设置为当前位置 + offset（可正可负）

ptrname = 2：读写指针位置设置文件结束位置 + offset（负）

文件系统的用户界面

文件的勾连与取消

link (oldpath, newpath)

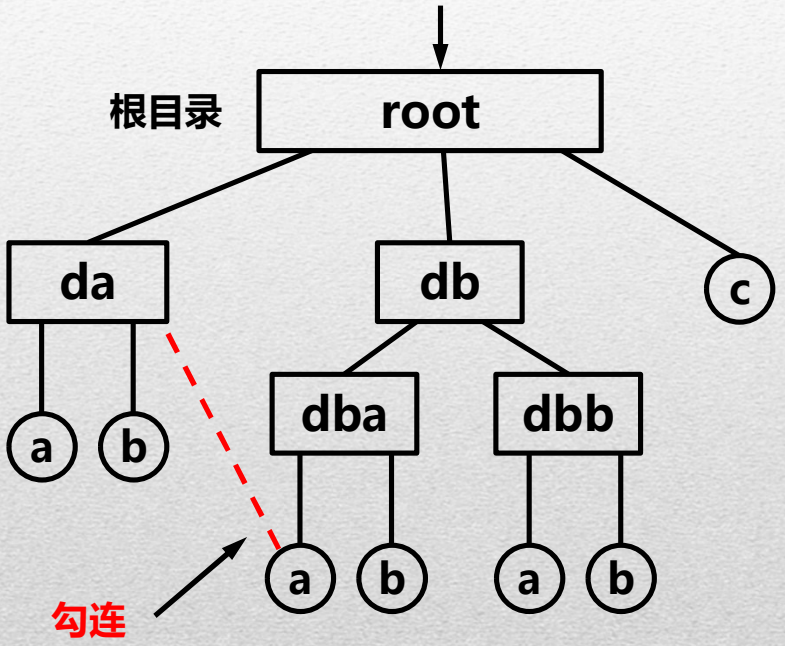
↓

为文件oldpath勾连一个新的路径名newpath

unlink (path)

↓

取消文件的路径名path



主 要 内 容

6.1 文件系统概述

6.2 UNIX文件系统接口

6.3 UNIX文件系统的物理结构

6.4 UNIX文件系统的打开结构

6.5 UNIX文件系统的目录管理

6.6 UNIX文件系统的读写操作

文件的逻辑结构

逻辑结构：从用户角度（操作系统层、应用程序层）观察到的文件的组织形式，是程序可直接处理的数据及其结构。

例：BMP文件

BMP文件的逻辑结构

数据段名称	对应的Windows结构体定义	大小(Byte)
bmp文件头	BITMAPFILEHEADER	14
位图信息头	BITMAPINFOHEADER	40
调色板		由颜色索引数决定
位图数据		由图像尺寸决定

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+a	+b	+c	+d	+e	+f	Dump
0000	42	4d	36	04	01	00	00	00	00	00	36	04	00	00	28	00	BM6.....6...(.
0010	00	00	00	01	00	00	00	01	00	00	01	00	08	00	00	00	.....
0020	00	00	00	00	01	00	00	00	00	00	00	00	00	00	00	01	.....
0030	00	00	00	01	00	00	fe	fa	fd	00	fd	f3	fc	00	f4	f3	.....þúý.ýóü.ôó
0040	fc	00	fc	f2	f4	00	f6	f2	f2	00	fb	f9	f6	00	ea	f3	ü.üòô.öòò.ûùö.êó
0050	f8	00	fb	ee	fa	00	fb	ee	f3	00	f4	ed	f2	00	f4	ea	ø.ûîú.ûîó.ôíò.ôê

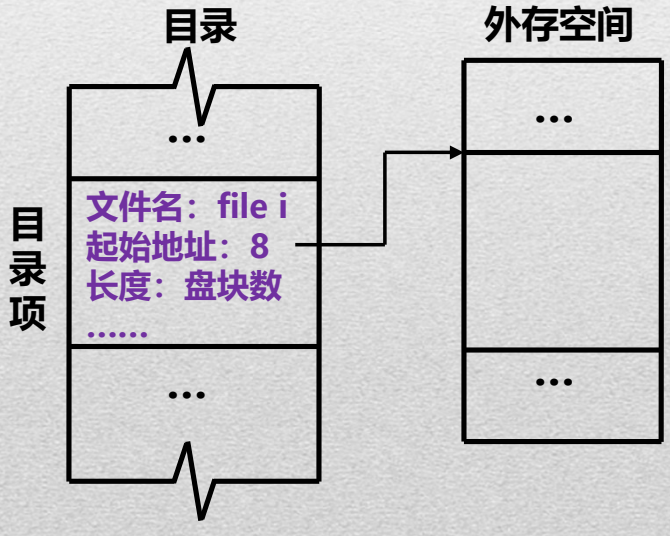
	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+a	+b	+c	+d	+e	+f	Dump
0000	42	4d	36	04	01	00	00	00	00	00	36	04	00	00	28	00	BM6.....6...(.
0010	00	00	00	01	00	00	00	01	00	00	01	00	08	00	00	00	.....
0020	00	00	00	00	01	00	00	00	00	00	00	00	00	00	00	01	.....
0030	00	00	00	01	00	00	fe	fa	fd	00	fd	f3	fc	00	f4	f3	.....þúý.ýóü.ôó
0040	fc	00	fc	f2	f4	00	f6	f2	f2	00	fb	f9	f6	00	ea	f3	ü.üòô.öòò.ûùö.êó
0050	f8	00	fb	ee	fa	00	fb	ee	f3	00	f4	ed	f2	00	f4	ea	ø.ûîú.ûîó.ôíò.ôê

文件的物理结构

物理结构：文件在存储介质上由操作系统如何保存。

连续结构文件：为每个文件分配一组相邻接的盘块。文件存放在连续编号的物理块中。保证了文件中逻辑顺序与占用盘块顺序的一致性。

建立连续文件时，用户给出文件最大长度，系统分配足够的连续外存空间，并在目录项中登记其起始物理地址（起始盘号）及长度（块数）。



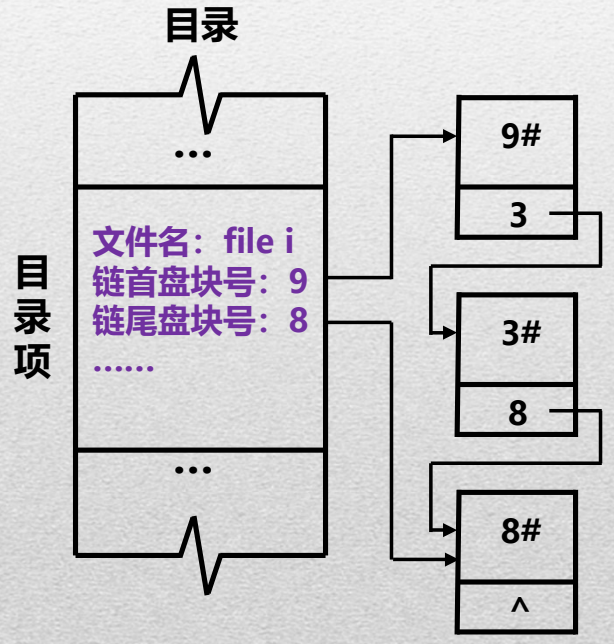
优点：
顺序访问容易且速度快

缺点：
① 要求连续的存储空间（磁盘碎片）
② 创建时需确定文件长度，不利于动态增长

文件的物理结构

物理结构：文件在存储介质上由操作系统如何保存。

链接结构文件：非连续的存储结构（将文件装入到多个离散的盘块中）。通过链接指针，将同属于一个文件的多个离散的盘块链接成一个链表。



优点：

离散分配方式有效利用空间

缺点：

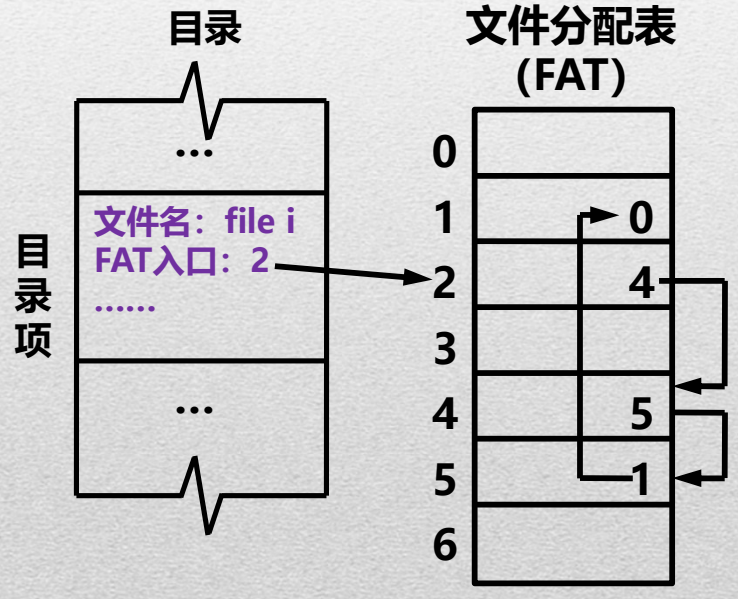
① 适合顺序存取
② 随机存取时有较大难度
③ 可靠性较差（其中一个指针出现问题，其后的文件都将丢失）

隐式链接

文件的物理结构

物理结构：文件在存储介质上由操作系统如何保存。

链接结构文件：非连续的存储结构（将文件装入到多个离散的盘块中）。通过链接指针，将同属于一个文件的多个离散的盘块链接成一个链表。



指针信息显式存放在内存中的一张文件分配表中
(整个磁盘一张)

优点：

① 查找记录的过程在内存进行，显著提高检索速度

② 减少了访问磁盘的次数

缺点：

① 不支持高效的直接存取

② FAT占用较大的存储空间

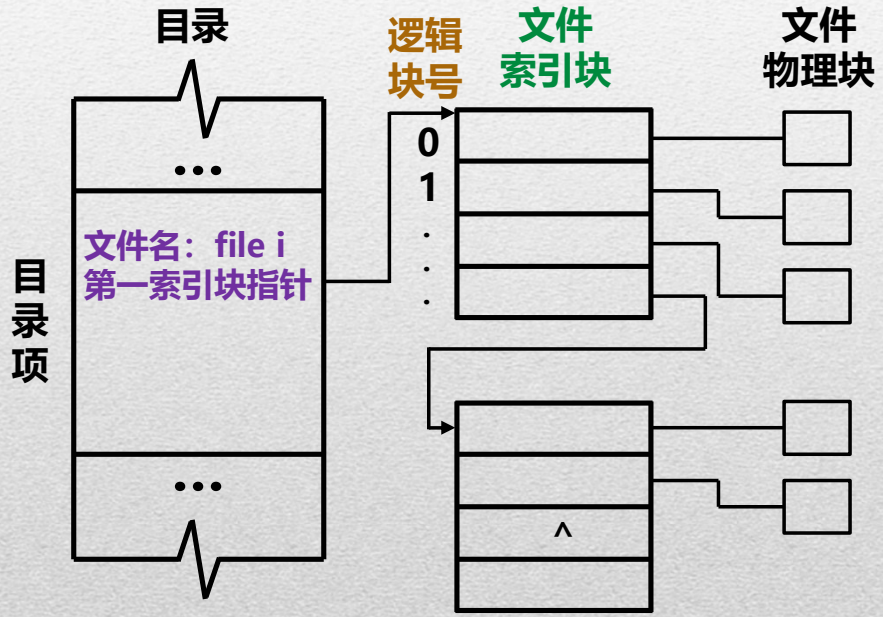
FAT16-FAT32-NTFS都是基于显式链接的文件物理结构

显式链接

文件的物理结构

物理结构：文件在存储介质上由操作系统如何保存。

索引结构文件：每个文件分配一个索引块（表），建立逻辑块号与物理块号的对照表。



优点：

① 可方便地实现随机存取

缺点：

① 先读索引块，才能获得所需的物理块号

② 增删物理块时，必须对索引表中所有后续项做移位操作

③ 索引块占用一定存储空间

当文件太大，索引块太多时，可建立多级索引。

UNIX 采用混合索引分配方式　i node

UNIX文件的物理结构

Super Block	inode区	文件数据区

UNIX文件的物理结构

外存文件控制块区（Inode区，202~1023#盘块）

是否已经分配　1：大型或巨型文件；0：小文件

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IALLOC	文件类型编码		ILARG	ISUID	ISGID	ISVTX	IREAD	IWRITE	IEXEC	IREAD	IWRITE	IEXEC	IREAD	IWRITE	IEXEC

00：普通数据文件　01：字符设备文件
10：目录文件　11：块设备文件

访问权：文件主、　文件主同组用户、其他用户

```
class DiskInode
{
public:
    unsigned int    d_mode;        /* 状态的标志位/
    int             d_nlink;       /* 该文件在目录树中不同路径名的数量 */
    short           d_uid;         /* 文件所有者的用户标识数 */
    short           d_gid;         /* 文件所有者的组标识数 */
    int             d_size;        /* 文件大小，字节为单位 */
    int             d_addr[10];    /* 文件逻辑块号和物理块号转换的混合索引表 */
    int             d_atime;       /* 最后访问时间 */
    int             d_mtime;       /* 最后修改时间 */
};
```

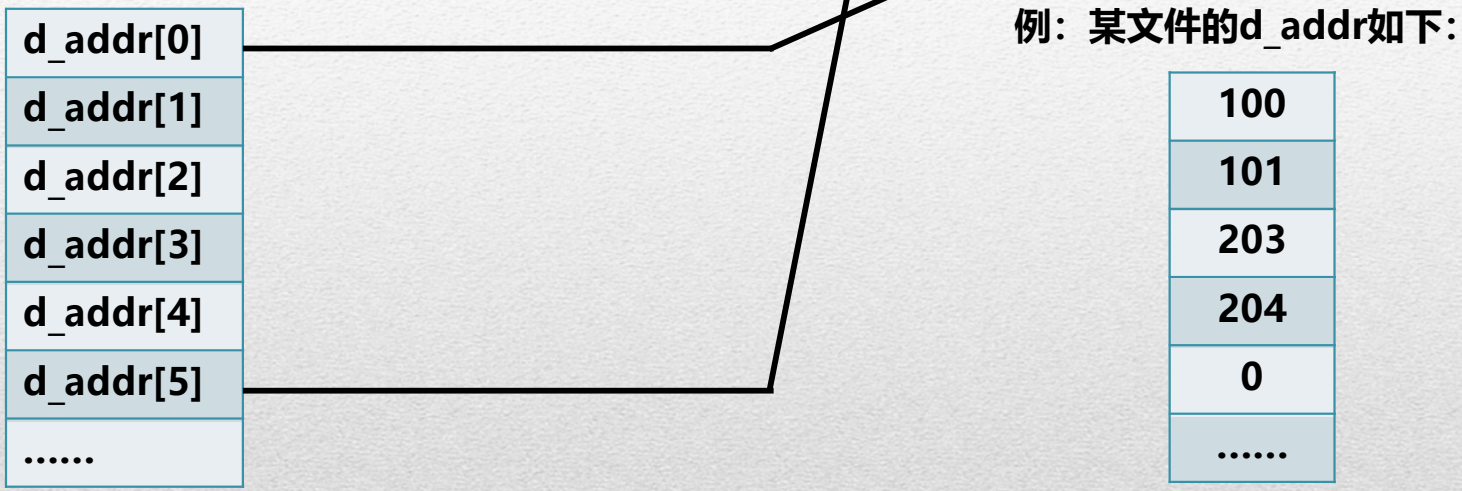
i node

文件索引节点：文件控制块，FCB

每个文件在Inode区有一个外存文件控制块DiskInode（外存索引节点，64个字节）

UNIX文件的物理结构

外存文件控制块区（Inode区，202~1023#盘块）



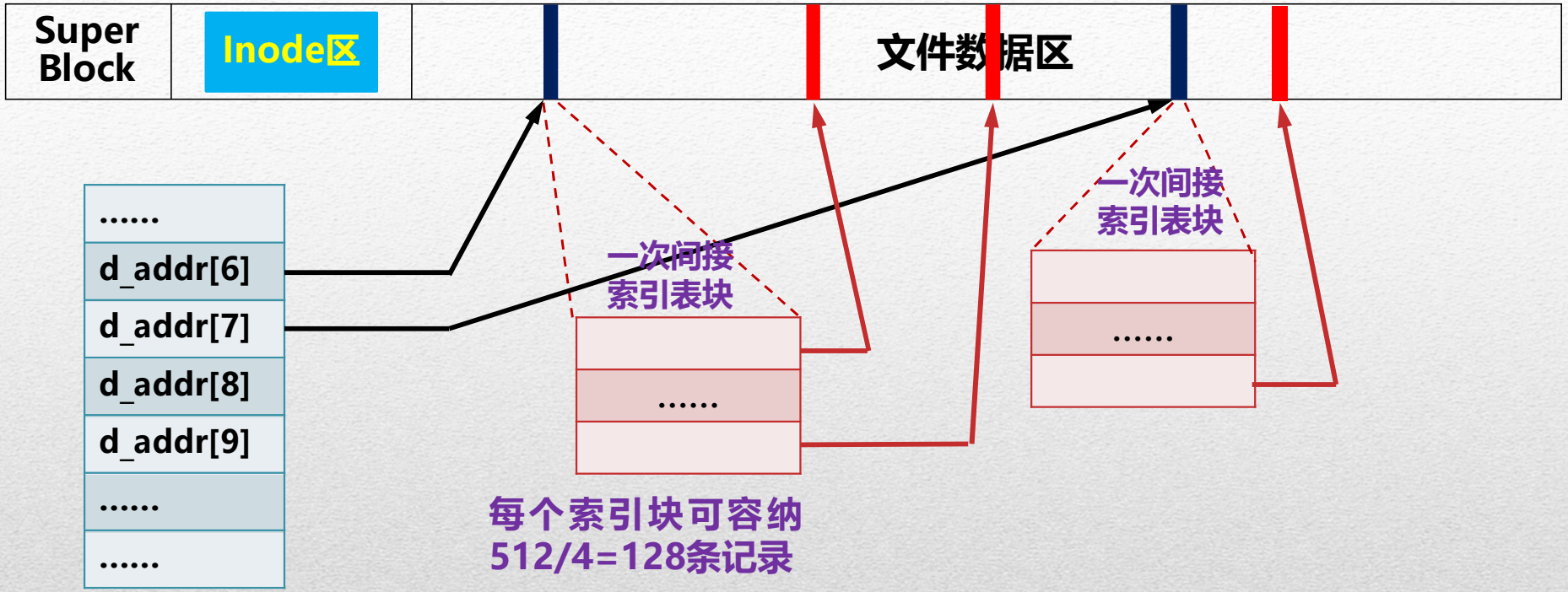
小型文件：0～6盘块（文件最大6×512 = 3K）

直接地址

逻辑块号n对应的物理块号存放在d_addr[n]中。

UNIX文件的物理结构

外存文件控制块区（Inode区，202~1023#盘块）

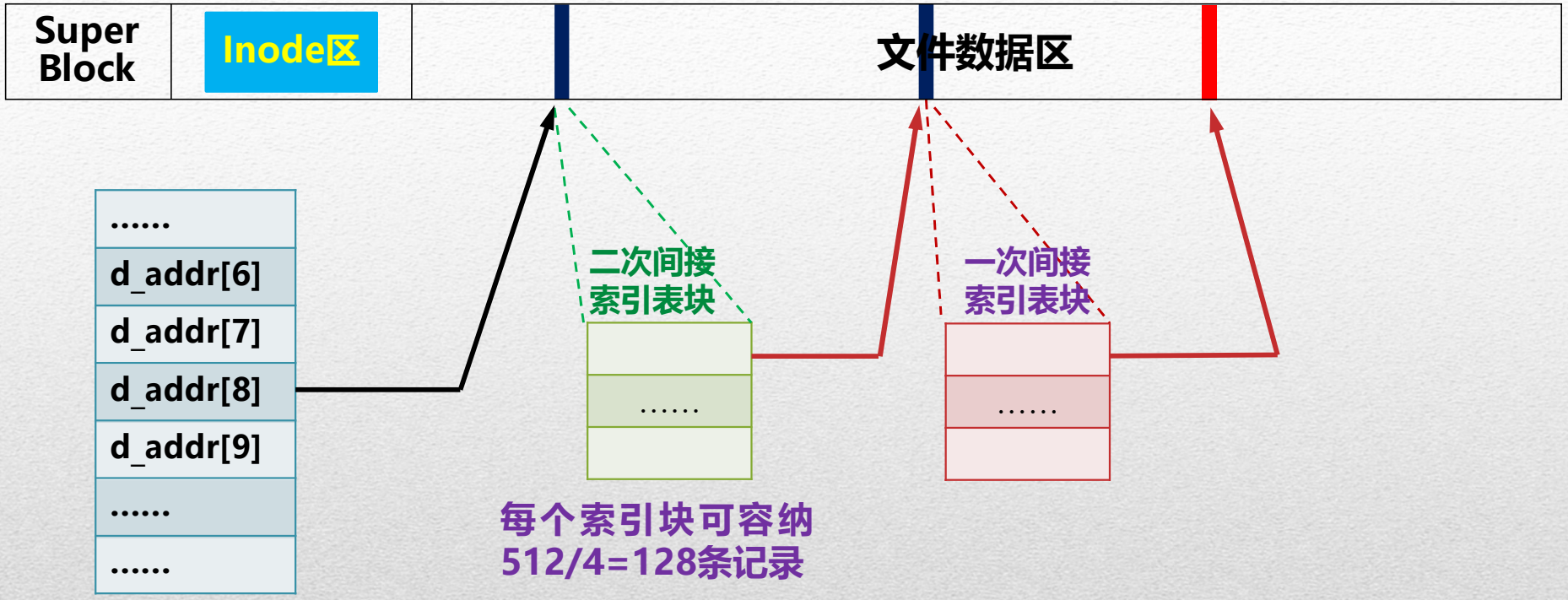


大型文件：7 ~（6+ 128 * 2）盘块

一次间接地址

UNIX文件的物理结构

外存文件控制块区（Inode区，202~1023#盘块）



巨型文件：（128 * 2 + 7）~
（6 + 128 * 2 + 128 * 128 * 2 ）

二次间接地址

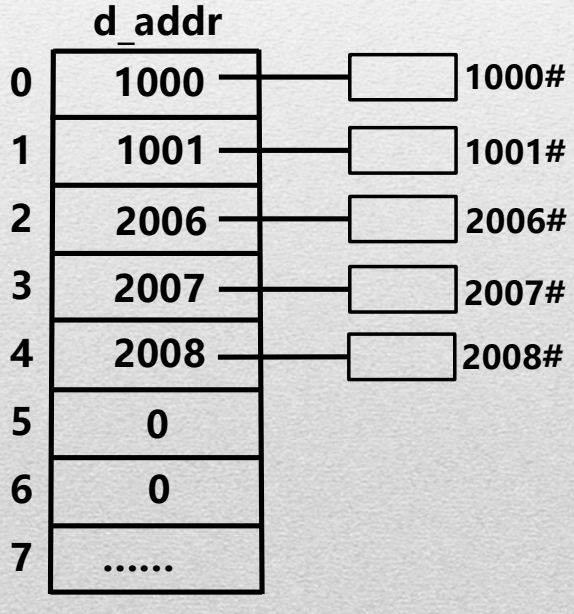
假如现在有三个文件，其大小分别为2248字节、65100字节和2M字节，分别画出d_addr[10]的结构示意图

2248/512 = 4　2248%512 = 200

所以2248个字节共占5个字符块，

前4个字符块为满块，

第5字符块占用200个字节。



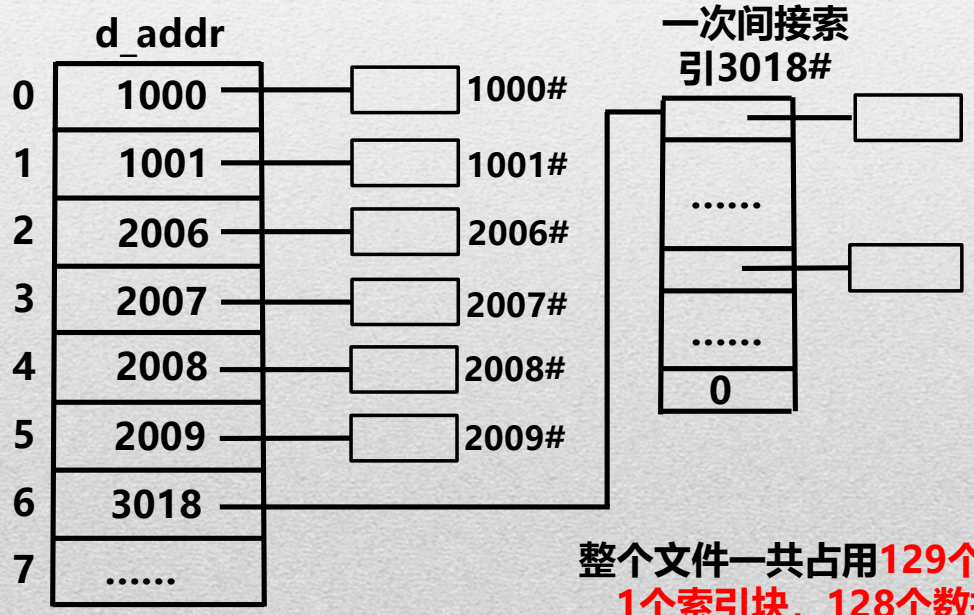
文件的逻辑块号lbn对应的物理块号

在直接索引中的入口：index0 = lbn;
d_addr[index0]= lbn对应的盘块号

例：lbn=3，index0 = 3，即：d_addr中的入口为3
d_addr[3] = 3号逻辑块对应的盘块号

假如现在有三个文件，其大小分别为2248字节、65100字节和2M字节，分别画出d_addr[10]的结构示意图

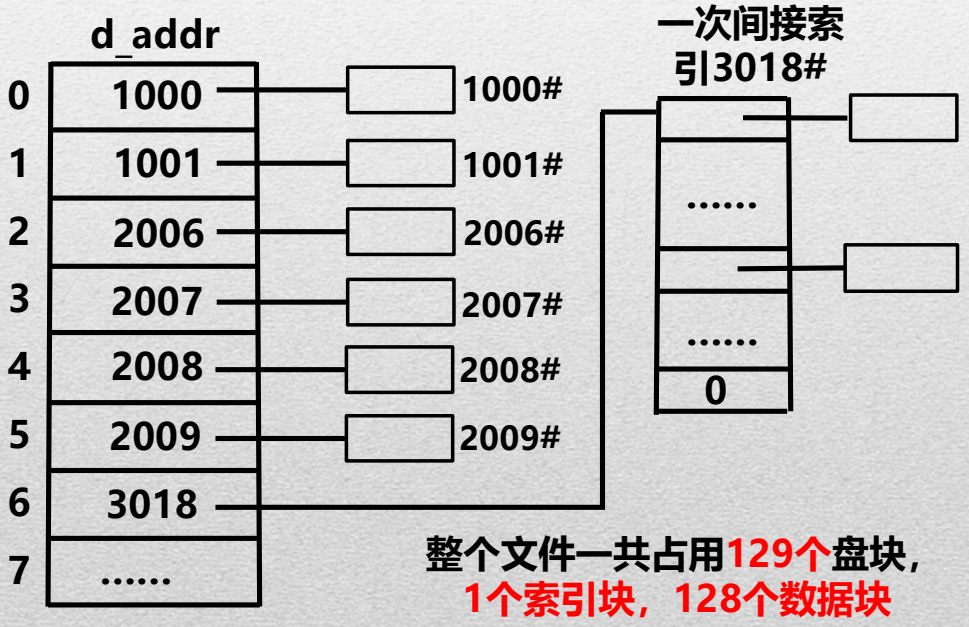
65100/512 = 127　65100%512 = 76
所以65100个字节共占128个字符块，
前127个字符块为满块，
第128字符块占用76个字节。



整个文件一共占用129个盘块，
1个索引块，128个数据块

假如现在有三个文件，其大小分别为2248字节、65100字节和2M字节，分别画出d_addr[10]的结构示意图

65100/512 = 127　65100%512 = 76
所以65100个字节共占128个字符块，
前127个字符块为满块，
第128字符块占用76个字节。



整个文件一共占用129个盘块，
1个索引块，128个数据块

文件的逻辑块号lbn对应的物理块号

在直接索引中的入口：index0 = (lbn -6) /128 +6;

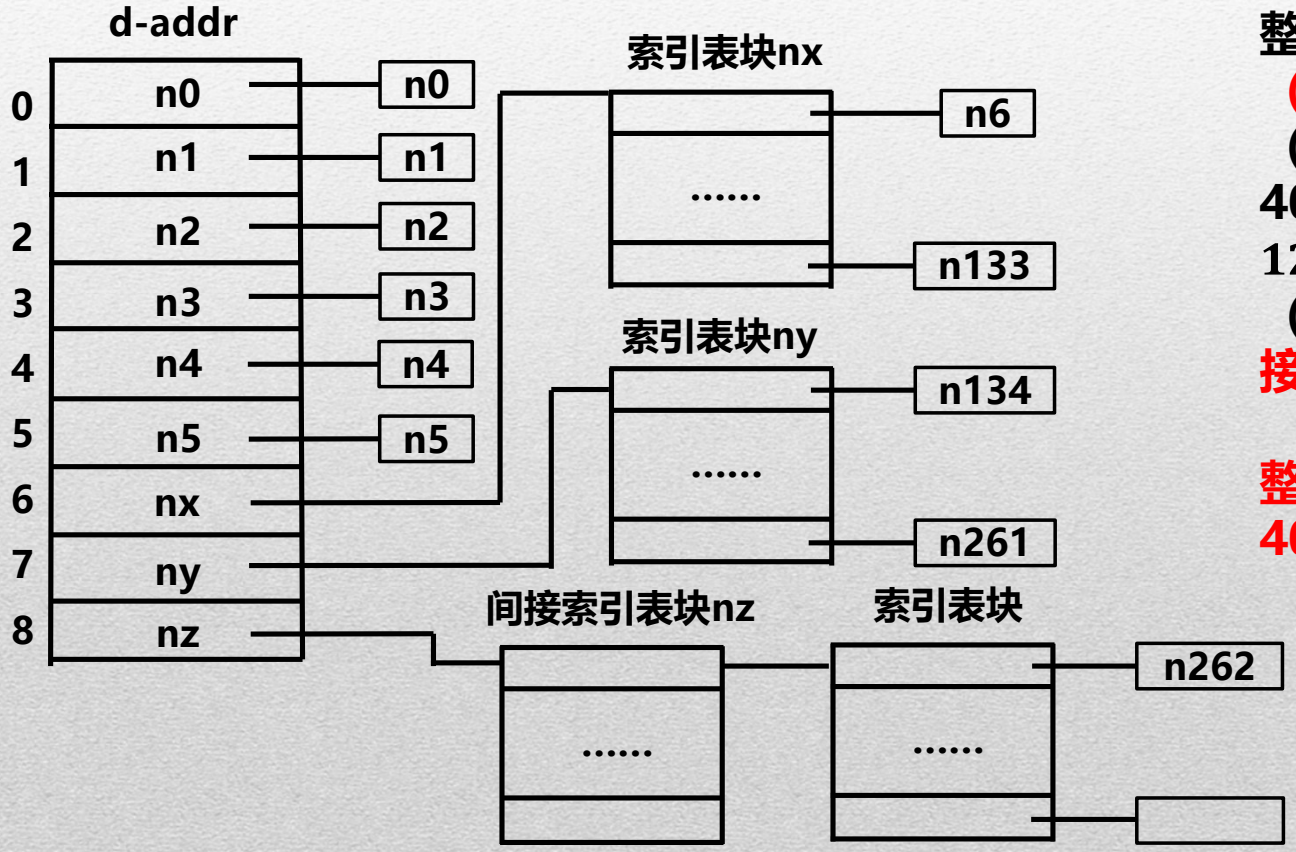
例：lbn=173 index0 = (173 - 6) / 128 + 6 = 7,
即：在d_addr中的中入口为7
d_addr[7] = 一次间接索引块所在的盘块号

在一次间接索引中的入口：
index1 = (lbn -6) % 128 ;

例：lbn=173 index1 = (173 - 6) %128 = 39,
即：在一次间接索引块中的入口为39
该位置保存173号逻辑块对应的盘块号

假如现在有三个文件，其大小分别为2248字节、65100字节和2M字节，分别画出d_addr[10]的结构示意图

2M/512 = 4096 所以2M个字节共占4096个字符块，为巨型文件。



整个文件一共占用的盘块数：

（1）4096个数据块；

（2）前6块为直接地址，后4090个数据块需要⌈4090/128⌉=32个索引块；

（3）后30个索引块需要1个间接索引块。

整个文件共占用：

4096+32+1=4129个物理盘块

假如现在有三个文件，其大小分别为2248字节、65100字节和2M字节，分别画出d_addr[10]的结构示意图

2M/512 = 4096 所以2M个字节共占4096个字符块，为巨型文件。

文件的逻辑块号lbn对应的物理块号

在直接索引中的入口：index0 = (lbn – 6- 128 * 2) / (128 * 128) + 8;

例：lbn= 1730。Index0 = (lbn – 6- 128 * 2) / (128 * 128) +8 = 8，即：在d_addr中的入口为8，d_addr[8] = 二次间接索引块所在的盘块号

在二次间接索引中的入口：index2 = ((lbn - (128 * 2 + 6)) / 128) % 128;;

index2 = ((1730 - (128 * 2 + 6)) /128) %128 = 11，即：在二次间接索引块中的入口为11，该位置保存一次间接索引块所在的盘块号

在一次间接索引中的入口：index1 = (lbn - (128 * 2 + 6)) % 128;;

index2 = (1730 - (128 * 2 + 6)) %128 = 60，即：在二次间接索引块中的入口为60，该位置保存一次间接索引块所在的盘块号

UNIX文件系统的静态结构

存储资源管理信息块（SuperBlock，200~201#盘块）

Super Block	inode区	文件数据区

```
class SuperBlock
{
/* Functions */
public:
SuperBlock(); /* Constructors */
~SuperBlock(); /* Destructors */
/* Members */
public:

int   s_fsize;        /* 盘块总数 */
int   s_nfree;        /* 直接管理的空闲盘块数量 */
int   s_free[100];    /* 直接管理的空闲盘块索引表 */
int   s_flock;        /* 封锁空闲盘块索引表标志 */

int   s_isize;        /* 外存Inode区占用的盘块数 */
int   s_ninode;       /* 直接管理的空闲外存Inode数量 */
int   s_inode[100];   /* 直接管理的空闲外存Inode索引表 */
int   s_ilock;        /* 封锁空闲Inode表标志 */

int   s_fmod;         /* 内存中super block副本被修改标志，意味着需要更新外存对应的Super Block */
int   s_ronly;        /* 本文件系统只能读出 */
int   s_time;         /* 最近一次更新时间 */
int   padding[47];    /* 填充使SuperBlock块大小等于1024字节，占据2个扇区 */
};
```

对文件数据区的管理

对INODE区的管理

SuperBlock占用两个扇区，一共1024个字节

UNIX文件系统的静态结构

存储资源管理信息块（SuperBlock，200~201#盘块）

1. SuperBlock对空闲inode的管理

- s_isize; ← inode区占用的块数
- s_ninode ← SuperBlock直接管理的空闲inode个数 ≤100
- s_inode[100] ← SuperBlock直接管理的空闲inode索引表
- s_ilock; ← 临界区锁

UNIX文件系统的静态结构

存储资源管理信息块（SuperBlock，200~201#盘块）

1. SuperBlock对空闲inode的管理

s_isize;
s_ninode
s_inode[100]
s_ilock;

任何时候只管理s_ninode个空闲inode

按栈的方式使用

释放一个inode时，进栈：

```
if( s_ninode <100 )
        s_inode[s_ninode++]
                = 该inode号
else
        不采取任何措施
```

分配一个inode时，退栈：

```
if( s_ninode > 0 )
        分配 s_inode[--s_ninode]
else
        重新在inode区
        搜索100个空闲inode
```

UNIX文件系统的静态结构

存储资源管理信息块（SuperBlock，200~201#盘块）

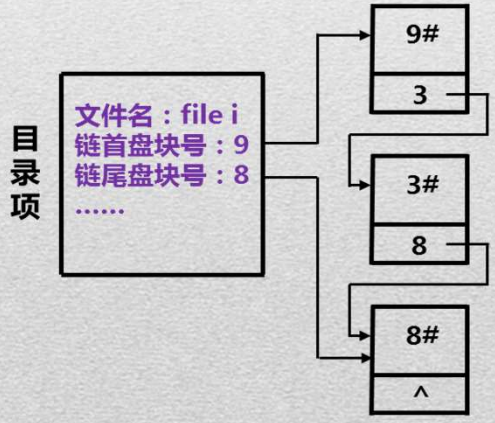
Super Block	inode区	文件数据区

2. SuperBlock对空闲数据块的管理

s_fsize ← 盘块总数

s_nfree ← SuperBlock直接管理的空闲数据块数 ≤100

s_free[100] ← SuperBlock直接管理的空闲盘块索引表



采用隐式链接来管理空闲盘块

为了防止链表太长，UNIX采用了成组链接法。

按 “栈的栈” 进行管理：

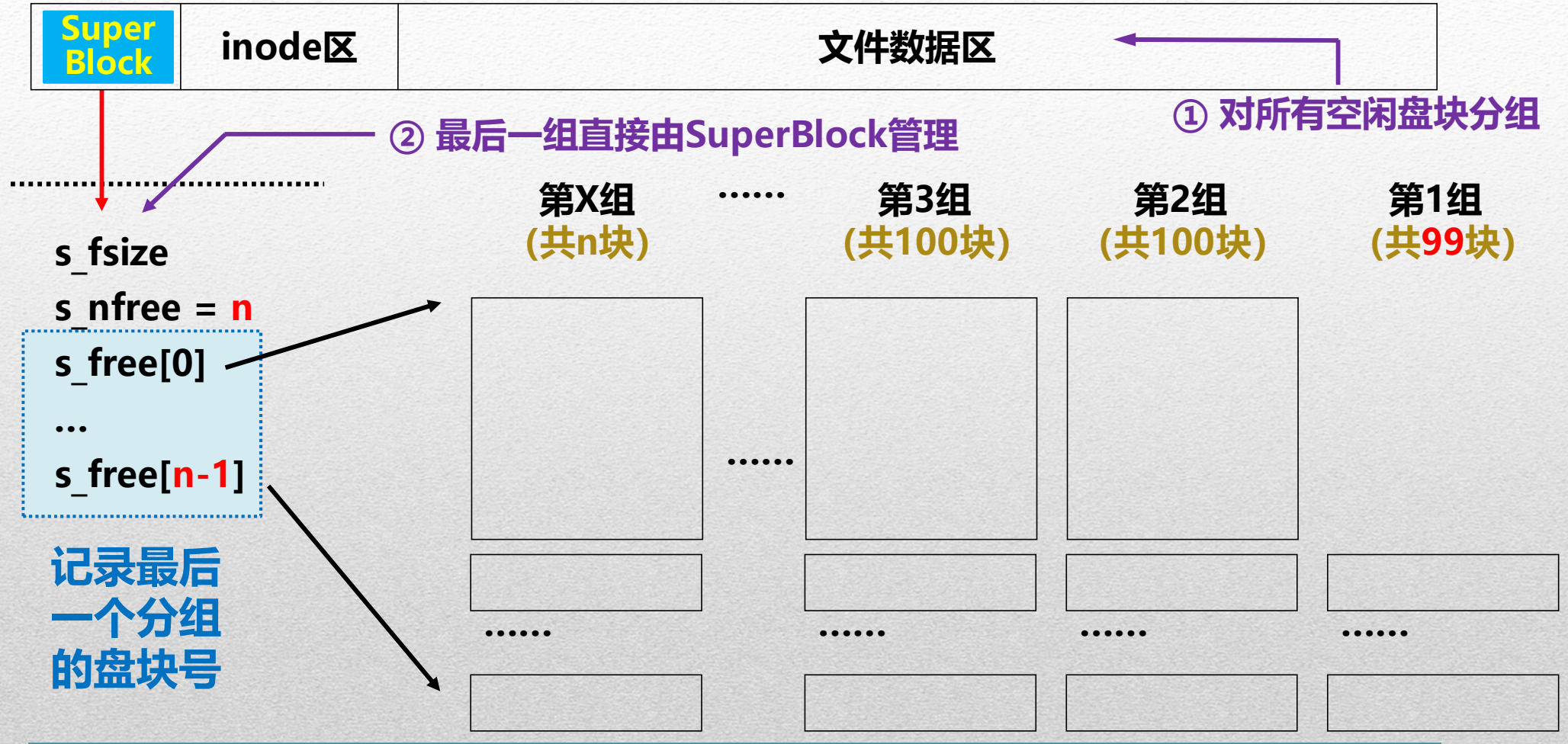
释放一盘块时，进栈。
分配一盘块时，退栈。

UNIX文件系统的静态结构

存储资源管理信息块（SuperBlock，200~201#盘块）

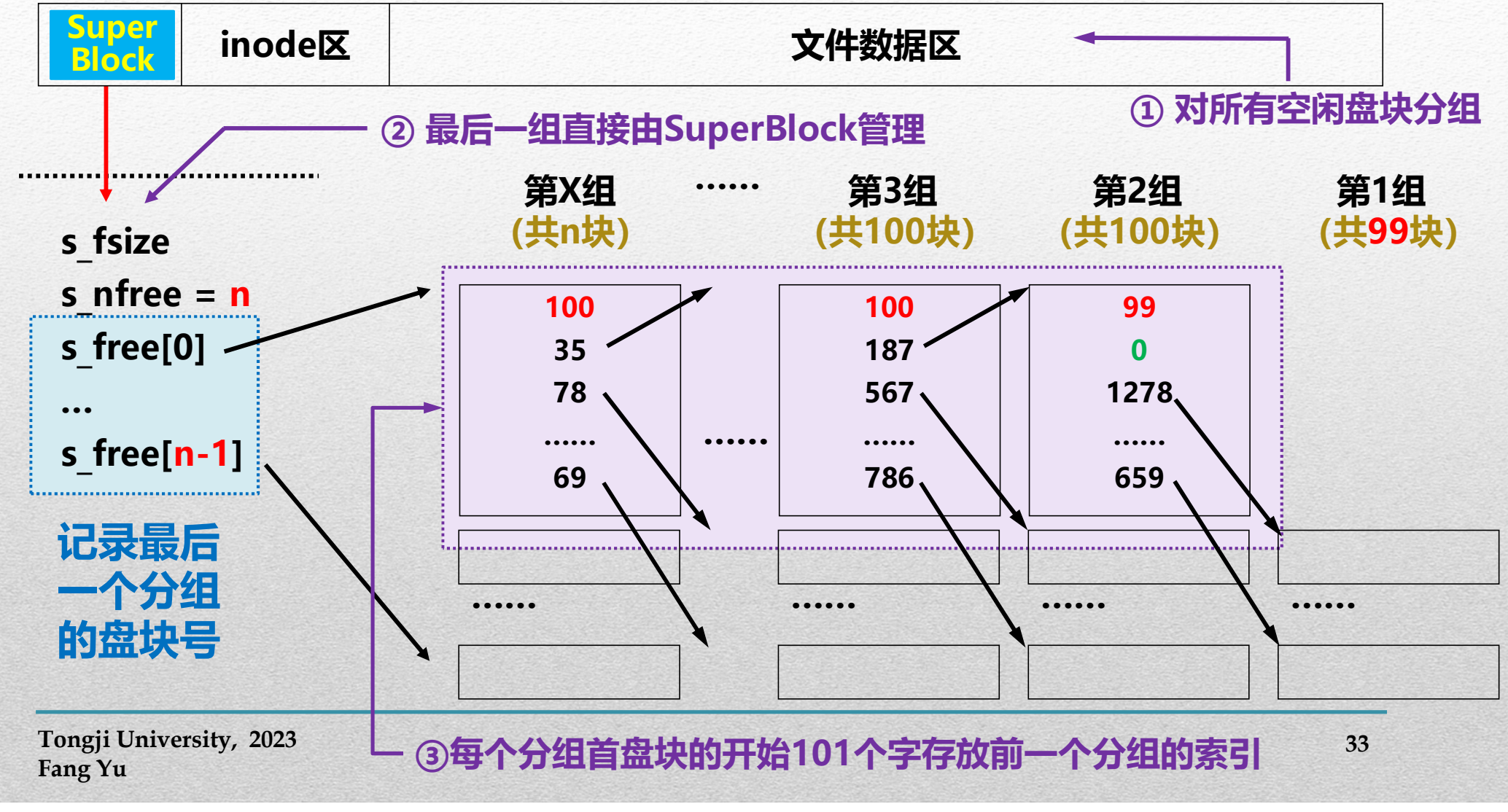
UNIX文件系统的静态结构

存储资源管理信息块（SuperBlock，200~201#盘块）



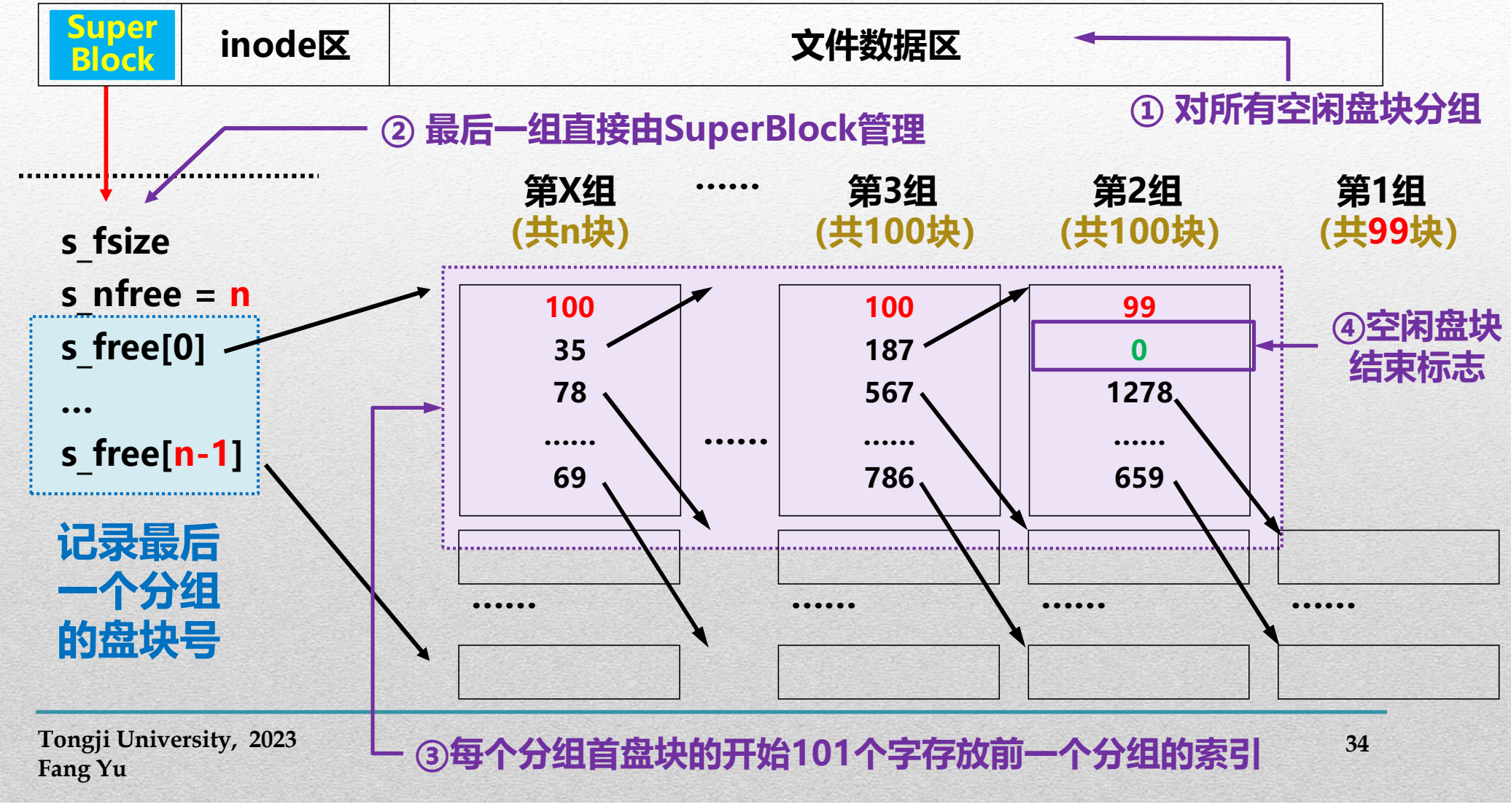
UNIX文件系统的静态结构

存储资源管理信息块（SuperBlock，200~201#盘块）



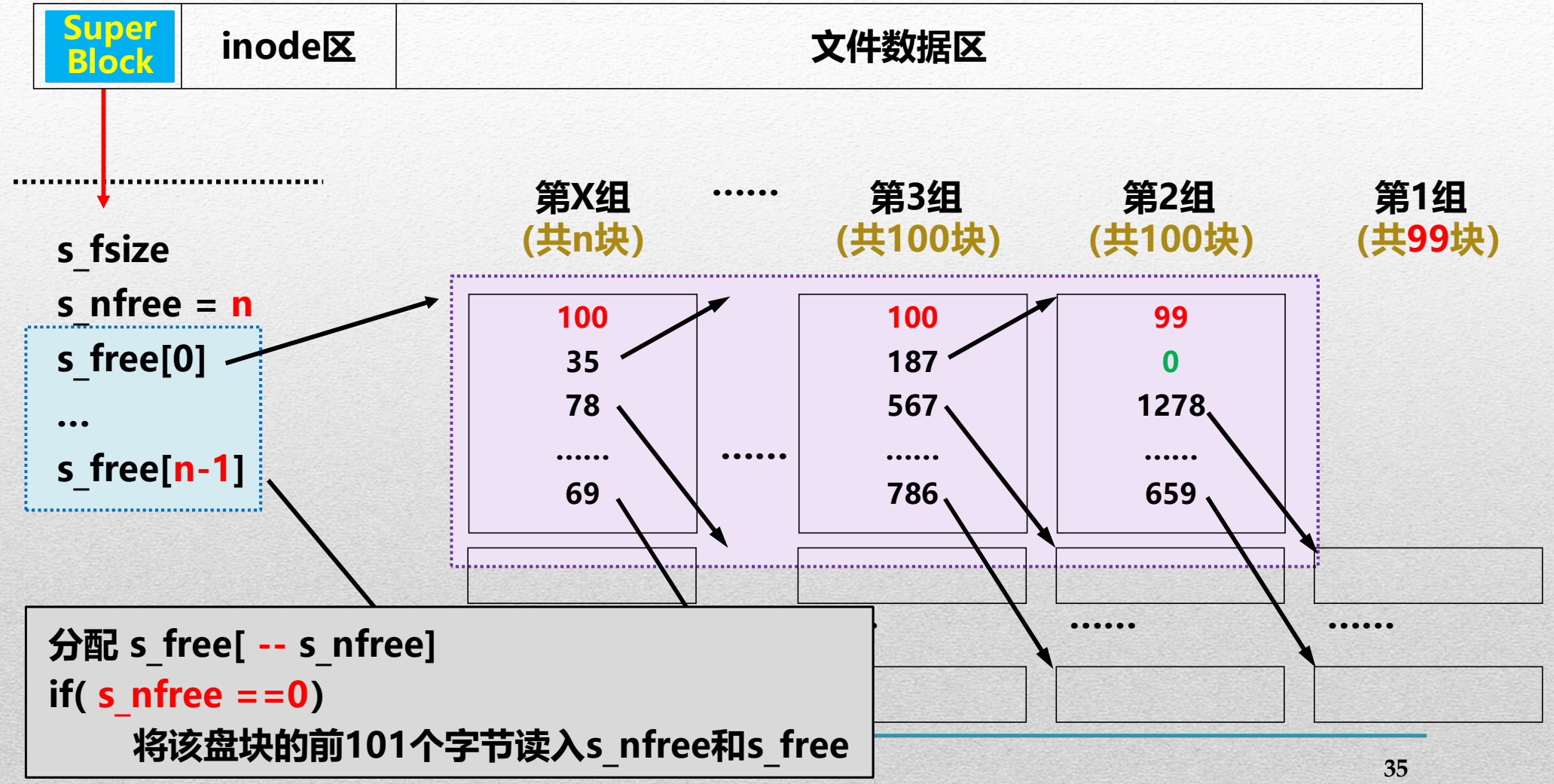
UNIX文件系统的静态结构

存储资源管理信息块（SuperBlock，200~201#盘块）



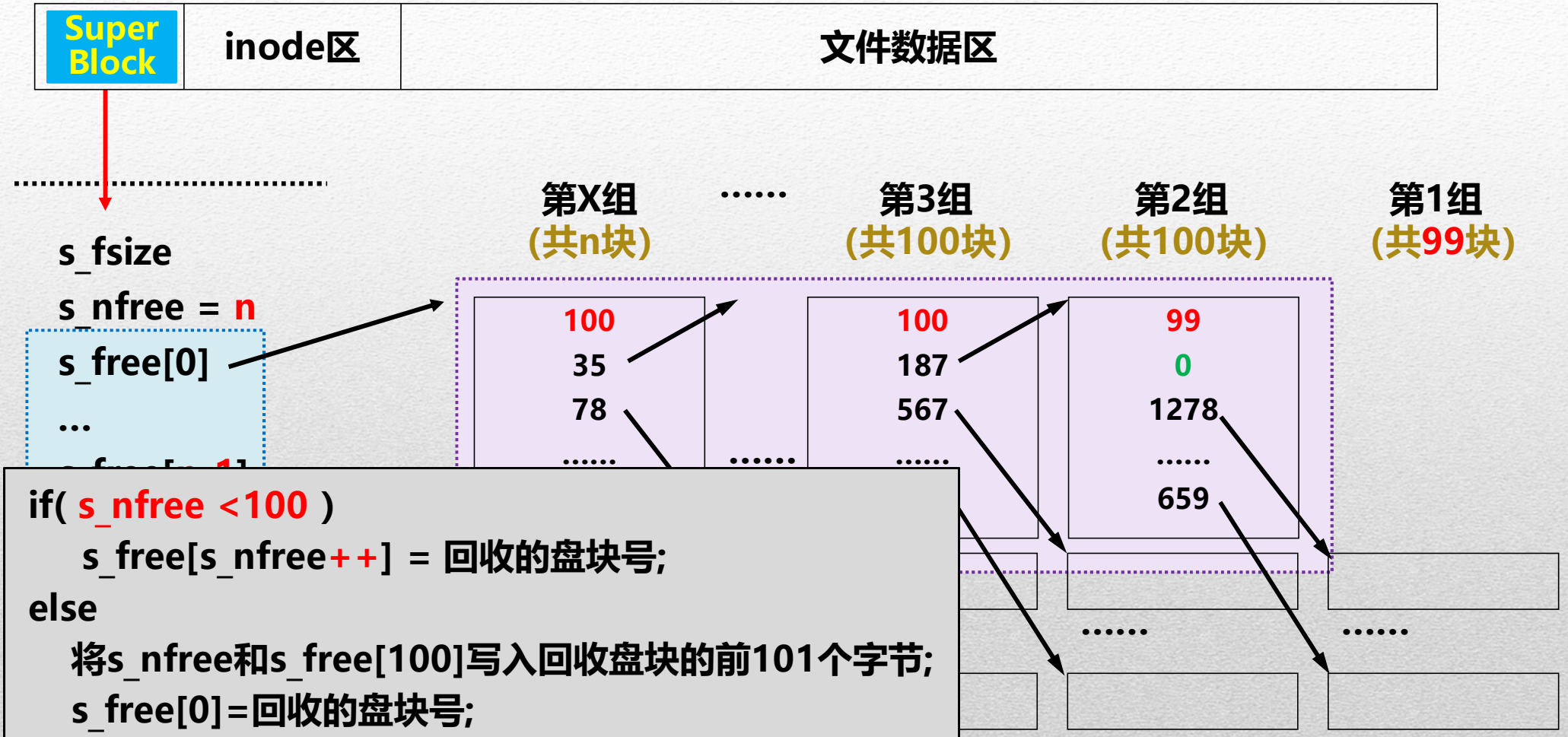
UNIX文件系统的静态结构

存储资源管理信息块（SuperBlock，200~201#盘块）

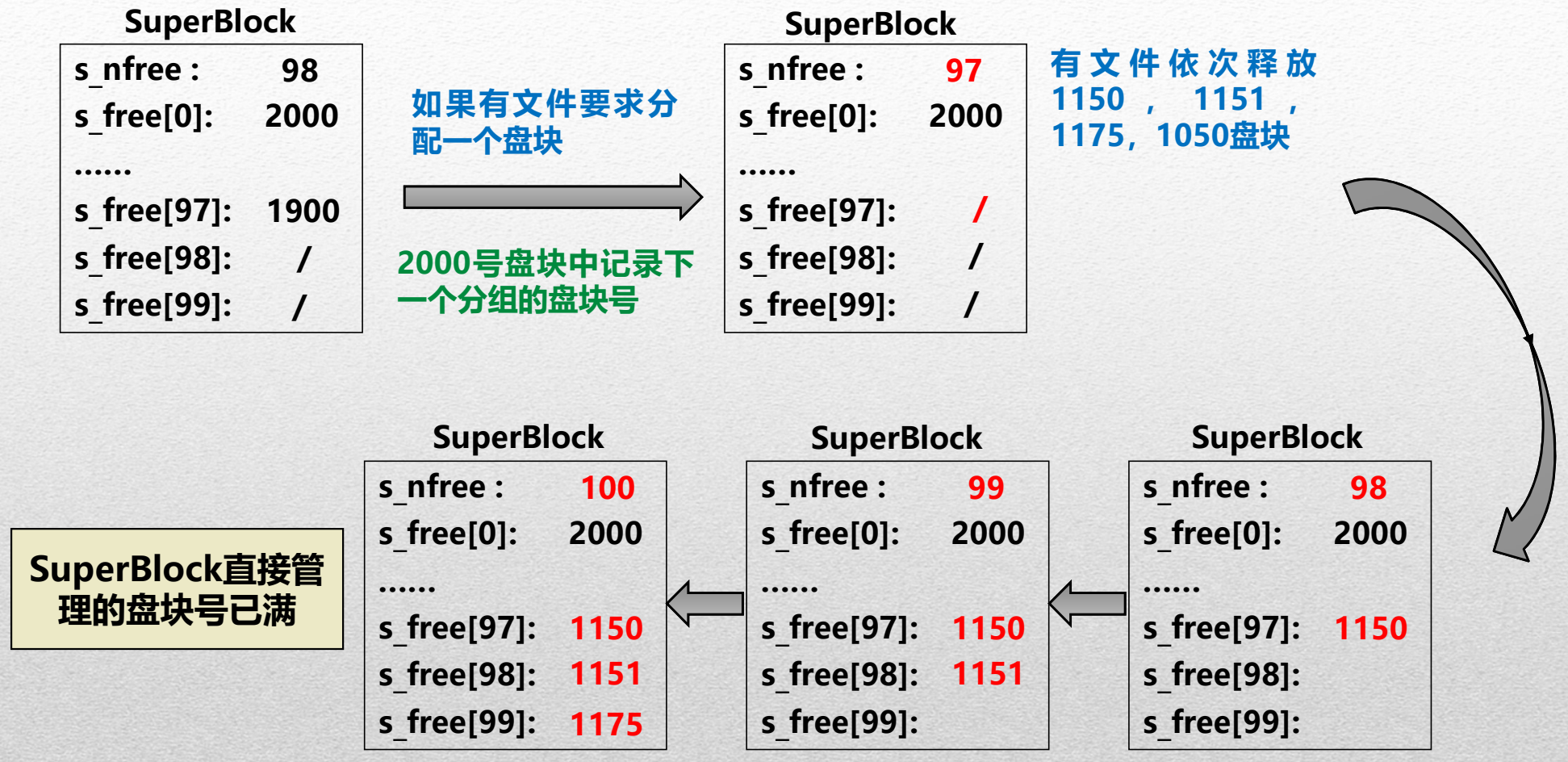


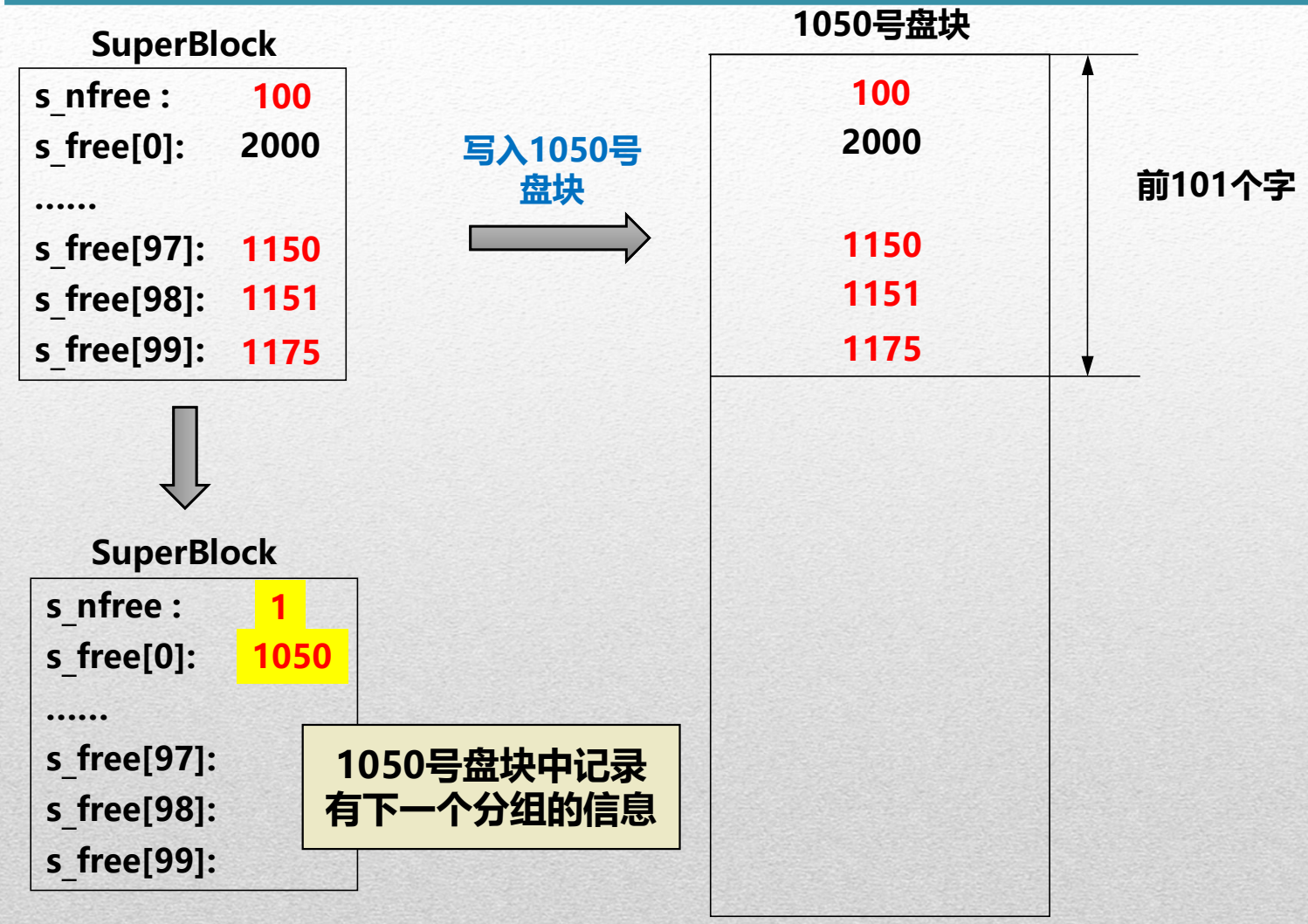
UNIX文件系统的静态结构

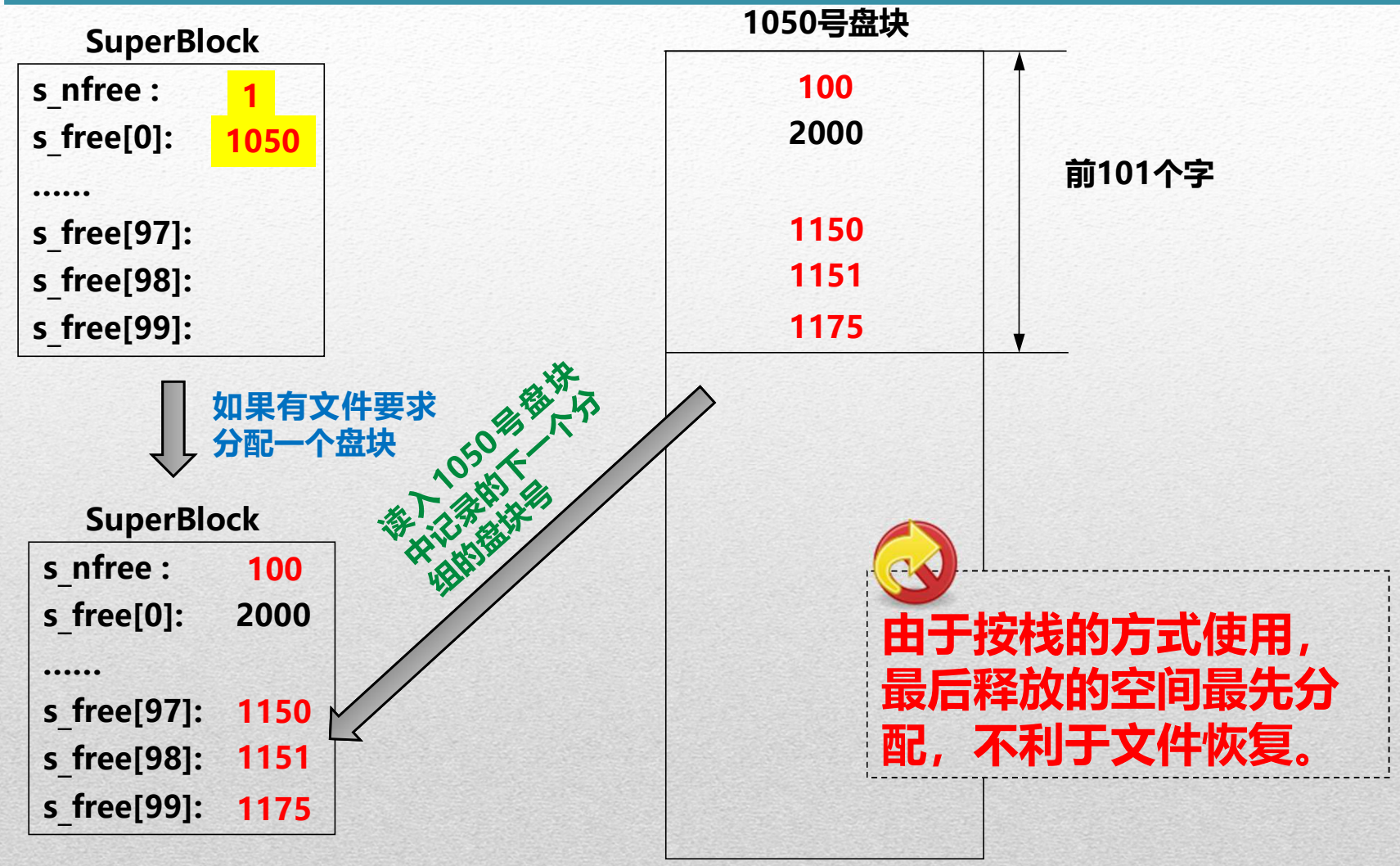
存储资源管理信息块（SuperBlock，200~201#盘块）




```
if( s_nfree <100 )
    s_free[s_nfree++] = 回收的盘块号;
else
    将s_nfree和s_free[100]写入回收盘块的前101个字节;
    s_free[0]=回收的盘块号;
    s_nfree =1;
```


SuperBlock
s_nfree : 98
s_free[0]: 2000
.....
s_free[97]: 1900
s_free[98]: /
s_free[99]: /
如果有文件要求分配一个盘块
2000号盘块中记录下一个分组的盘块号
SuperBlock
s_nfree : 97
s_free[0]: 2000
.....
s_free[97]: /
s_free[98]: /
s_free[99]: /
有文件依次释放1150，1151，1175，1050盘块
SuperBlock
s_nfree : 98
s_free[0]: 2000
.....
s_free[97]: 1150
s_free[98]:
s_free[99]:
SuperBlock
s_nfree : 99
s_free[0]: 2000
.....
s_free[97]: 1150
s_free[98]: 1151
s_free[99]:
SuperBlock
s_nfree : 100
s_free[0]: 2000
.....
s_free[97]: 1150
s_free[98]: 1151
s_free[99]: 1175
SuperBlock直接管理的盘块号已满


SuperBlock
s_nfree : 100
s_free[0]: 2000
.....
s_free[97]: 1150
s_free[98]: 1151
s_free[99]: 1175
写入1050号盘块
1050号盘块
100
2000
1150
1151
1175
前101个字
SuperBlock
s_nfree : 1
s_free[0]: 1050
.....
s_free[97]:
s_free[98]:
s_free[99]:
1050号盘块中记录有下一个分组的信息


SuperBlock
s_nfree : 1
s_free[0]: 1050
.....
s_free[97]:
s_free[98]:
s_free[99]:
如果有文件要求
分配一个盘块
SuperBlock
s_nfree : 100
s_free[0]: 2000
.....
s_free[97]: 1150
s_free[98]: 1151
s_free[99]: 1175
读入1050号盘块
中记录的下一个分
组的盘块号
1050号盘块
100
2000
1150
1151
1175
前101个字
由于按栈的方式使用，
最后释放的空间最先分
配，不利于文件恢复。

UNIX文件系统磁盘组织结构

Super Block	inode区	文件数据区

inode区有多少个inode?

inode和磁盘文件一一对应，所以有多少个inode就能创建多少个磁盘文件。

inode用完了，文件数据区还有空?

inode还有，文件数据区没空了?

本节小结：

1　UNIX文件系统的磁盘组织结构与SuperBlock

2　UNIX文件的物理结构与索引节点

E09：UNIX V6++的SuperBlock与Inode