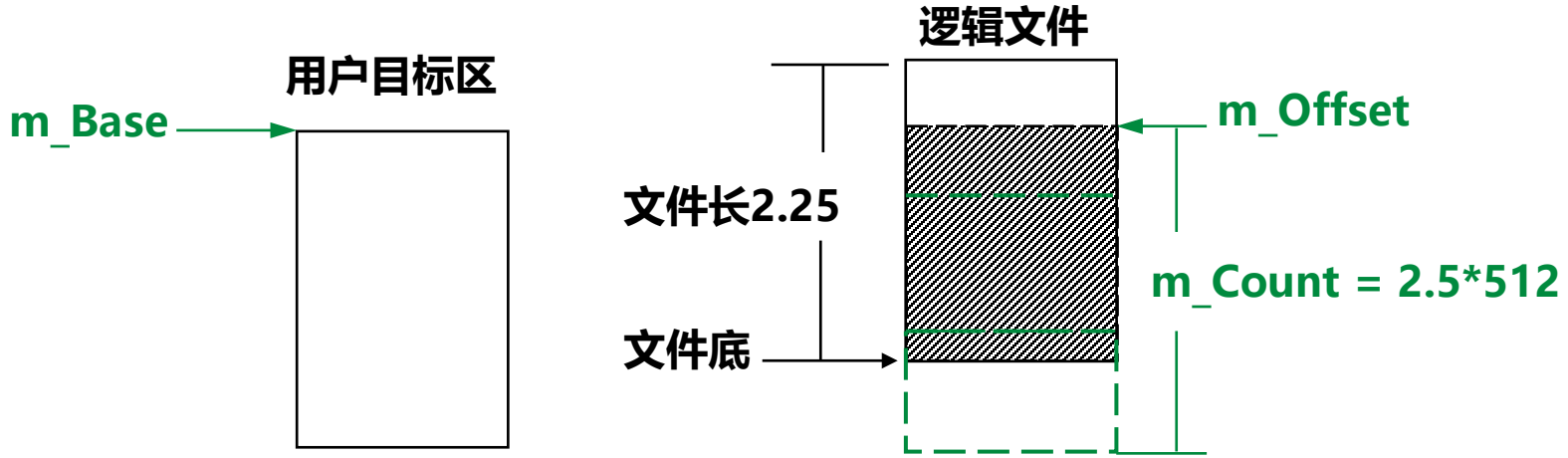
第六章

文件管理

UNIX文件系统的读写操作

读操作

从进程空间写2.5块到文件的过程



写操作超出文件长度怎么办?

主 要 内 容

6.1 文件系统概述

6.2 文件的逻辑结构与物理结构

6.3 文件存储空间管理

- 文件存储空间管理方法
- UNIX磁盘存储空间管理
- UNIX文件的长度变化

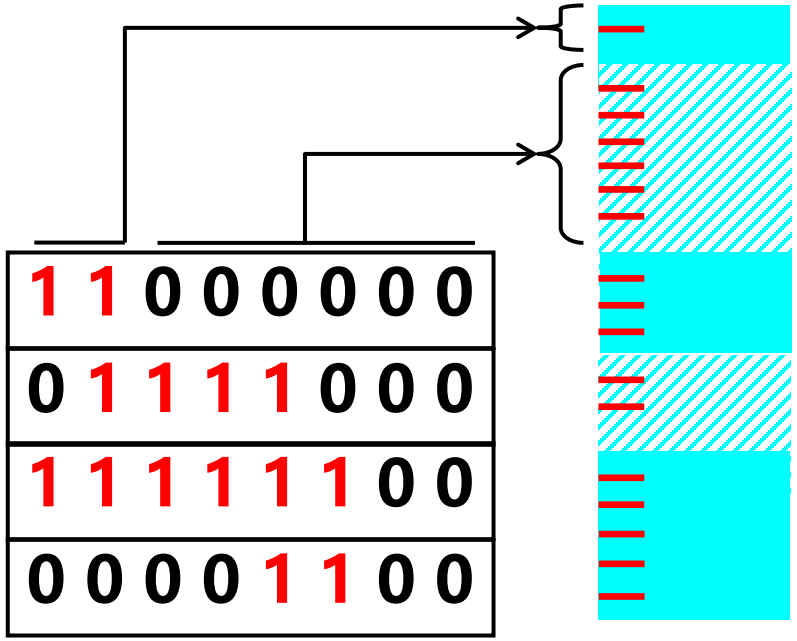
6.4 文件系统的目录管理

文件存储空间管理

盘上空闲区的记录、分配和回收

位图法：利用二进制的一位来表示一个盘块的使用情况

位图法



例如：假定一个盘组共有100个柱面，每个柱面上有8个磁道，每个盘面分成4个扇区。那么，整个磁盘空间共有： 4×8×100＝3200个存储块。

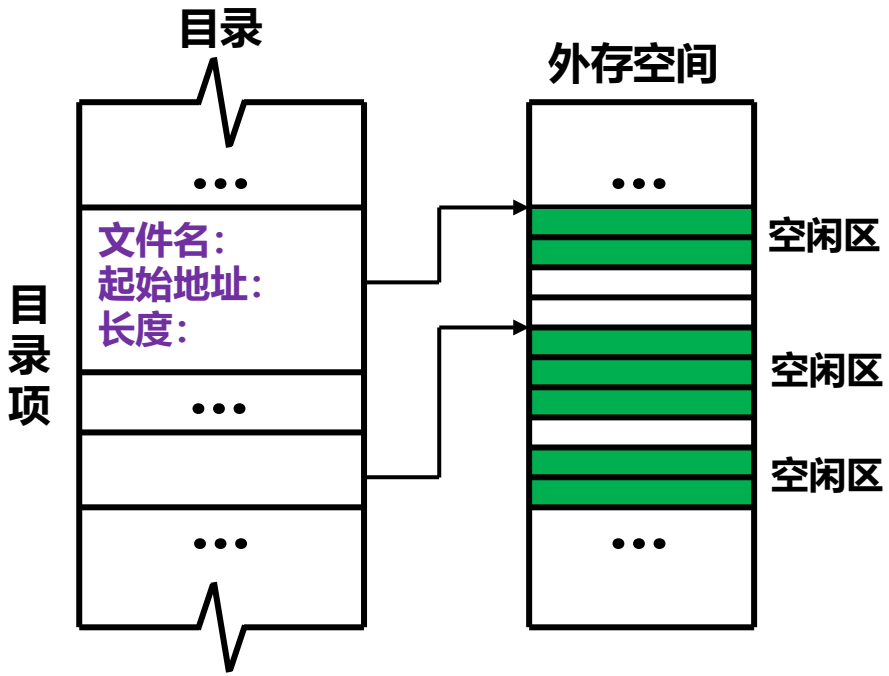
如果用字长为 32位的单元来构造位示图，共需100个字。

文件存储空间管理

盘上空闲区的记录、分配和回收

空白文件：将所有盘空闲区组织成一个或多个空白文件，按连续结构、链接结构等方式组织这些空白文件。

连续文件



空闲文件目录

相邻空闲块（空闲区）组成一空白文件。
系统中空白文件数 = 空闲区数
空白文件参与目录表登记

分配空闲区

首次适应算法/循环首次适应

回收空闲区

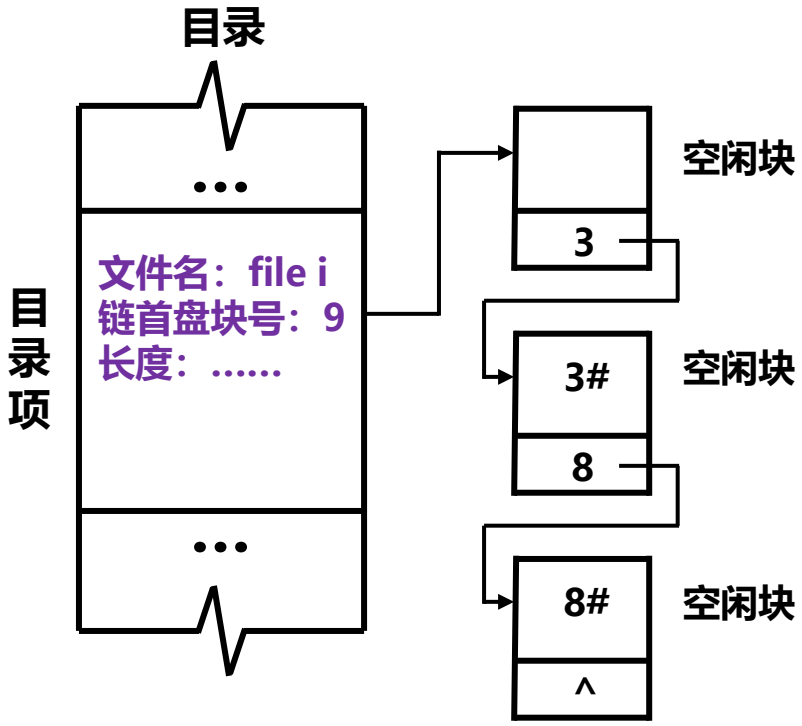
插入一新空白文件目录项；
或：并入相邻的空白文件。

文件存储空间管理

盘上空闲区的记录、分配和回收

空白文件：将所有盘空闲区组织成一个或多个空白文件，按连续结构、链接结构等方式组织这些空白文件。

链接文件



空闲块（区）链

所有空闲块（区）采用隐式链接结构组织成一个空白文件

空白文件参与目录表登记

分配空闲区

在链首处逐块分配

回收空闲区

将回收的空闲块插入队尾

也可用显式链接或索引结构实现

文件存储空间管理

空白文件

盘上空闲区的记录、分配和回收

空白文件：将所有盘空闲区组织成一个或多个空白文件，按连续结构、链接结构等方式组织这些空白文件。

在大型文件系统中，连续文件和链接文件的方式会导致空闲文件目录项太多或空闲链表太长。

UNIX结合上述两种方式的优点，采用了成组链接法，克服了两种方法均有的表太长的缺点。

主 要 内 容

6.1 文件系统概述

6.2 文件的逻辑结构与物理结构

6.3 文件存储空间管理

- 文件存储空间管理方法
- UNIX磁盘存储空间管理
- UNIX文件的长度变化

6.4 文件存储空间管理

6.5 UNIX文件系统

UNIX文件系统的磁盘空间管理

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 */
};
```

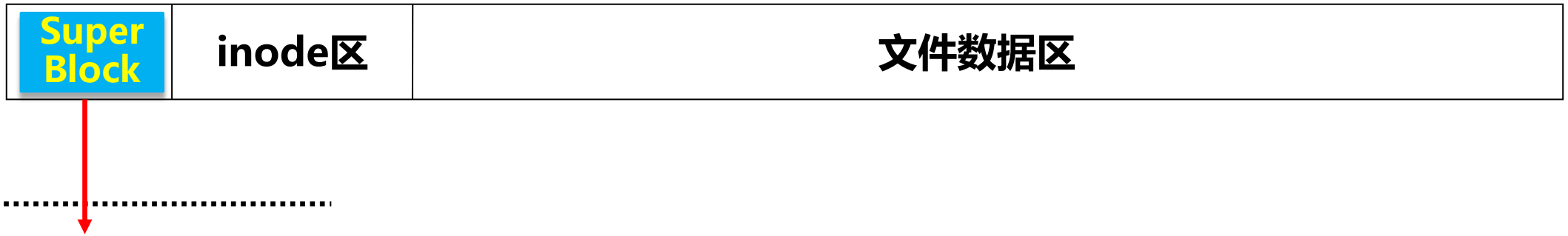
对文件数据区的管理

SuperBlock占用两个盘块，一共1024个字节

UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

对文件数据区的管理

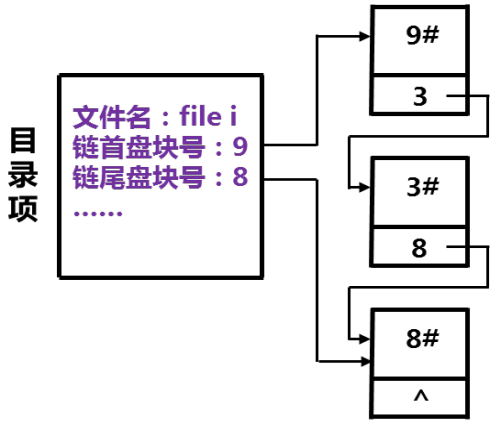


s_fsize ← 盘块总数

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号索引表

s_flock ← 多进程对Block操作的互斥锁



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**。

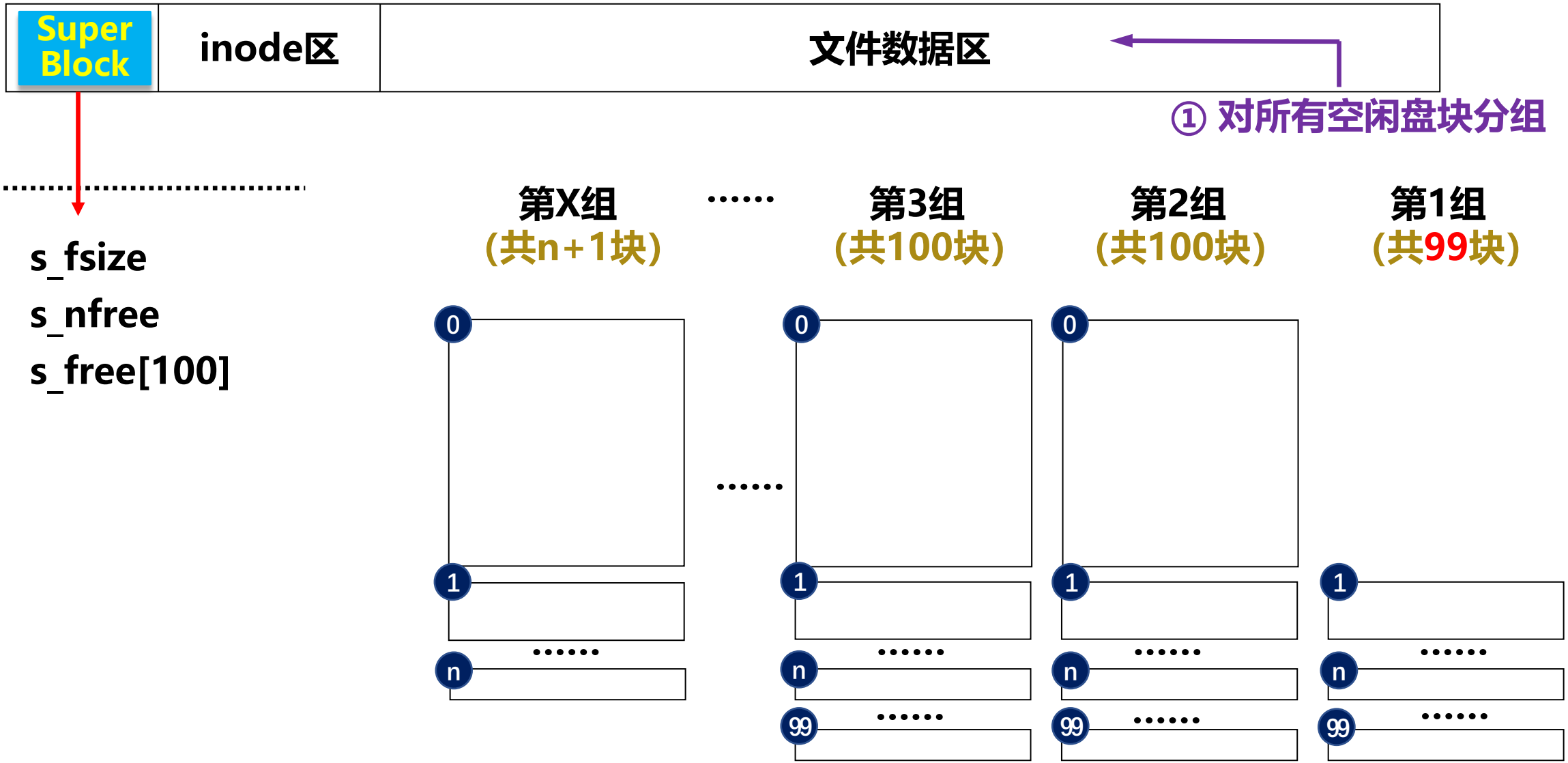
按 “栈的栈” 进行管理：

释放一盘块时，进栈。
分配一盘块时，退栈。

UNIX文件系统的磁盘空间管理

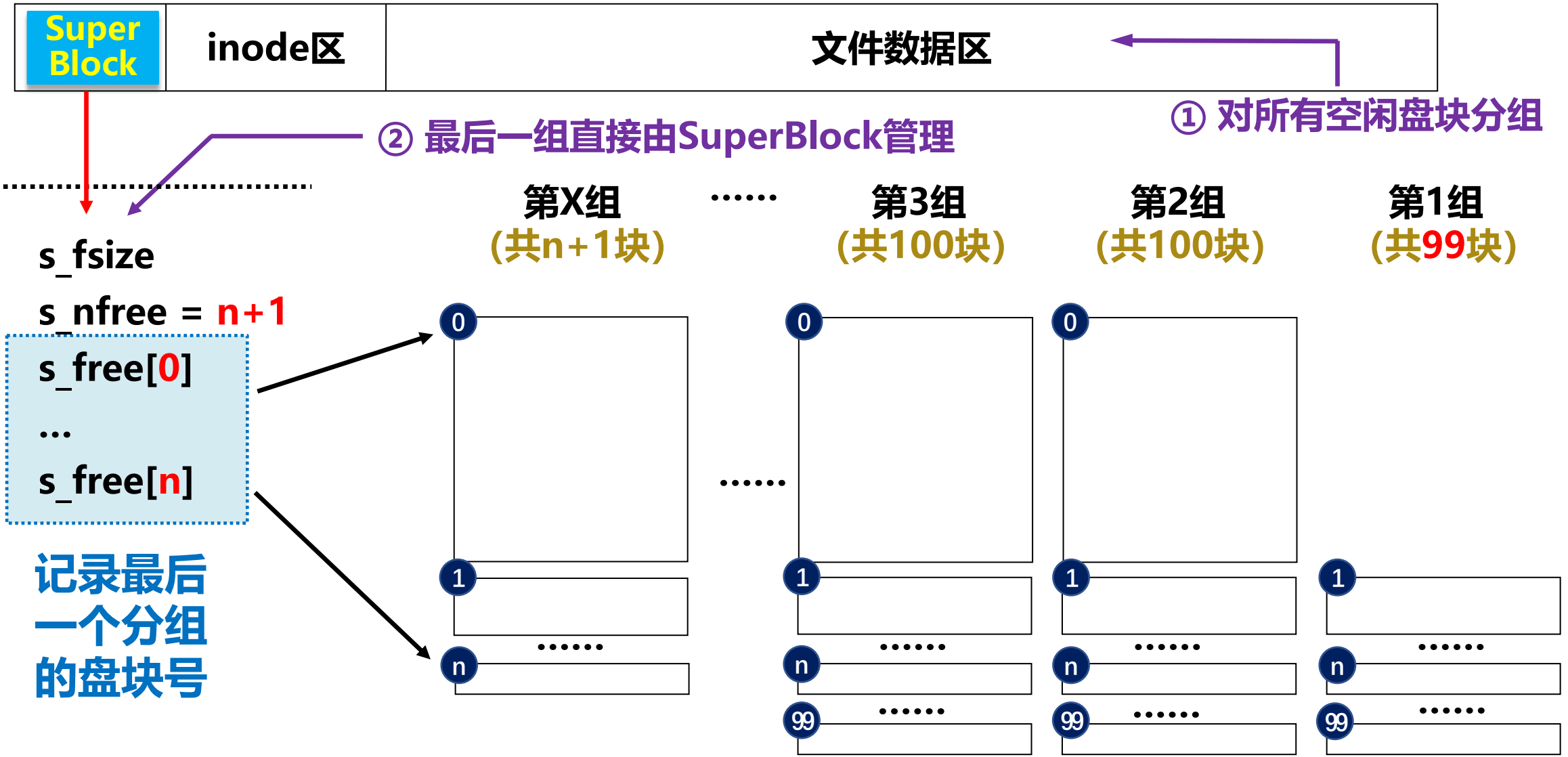
存储资源管理信息块（SuperBlock，200~201#盘块）

对文件数据区的管理



UNIX文件系统的磁盘空间管理

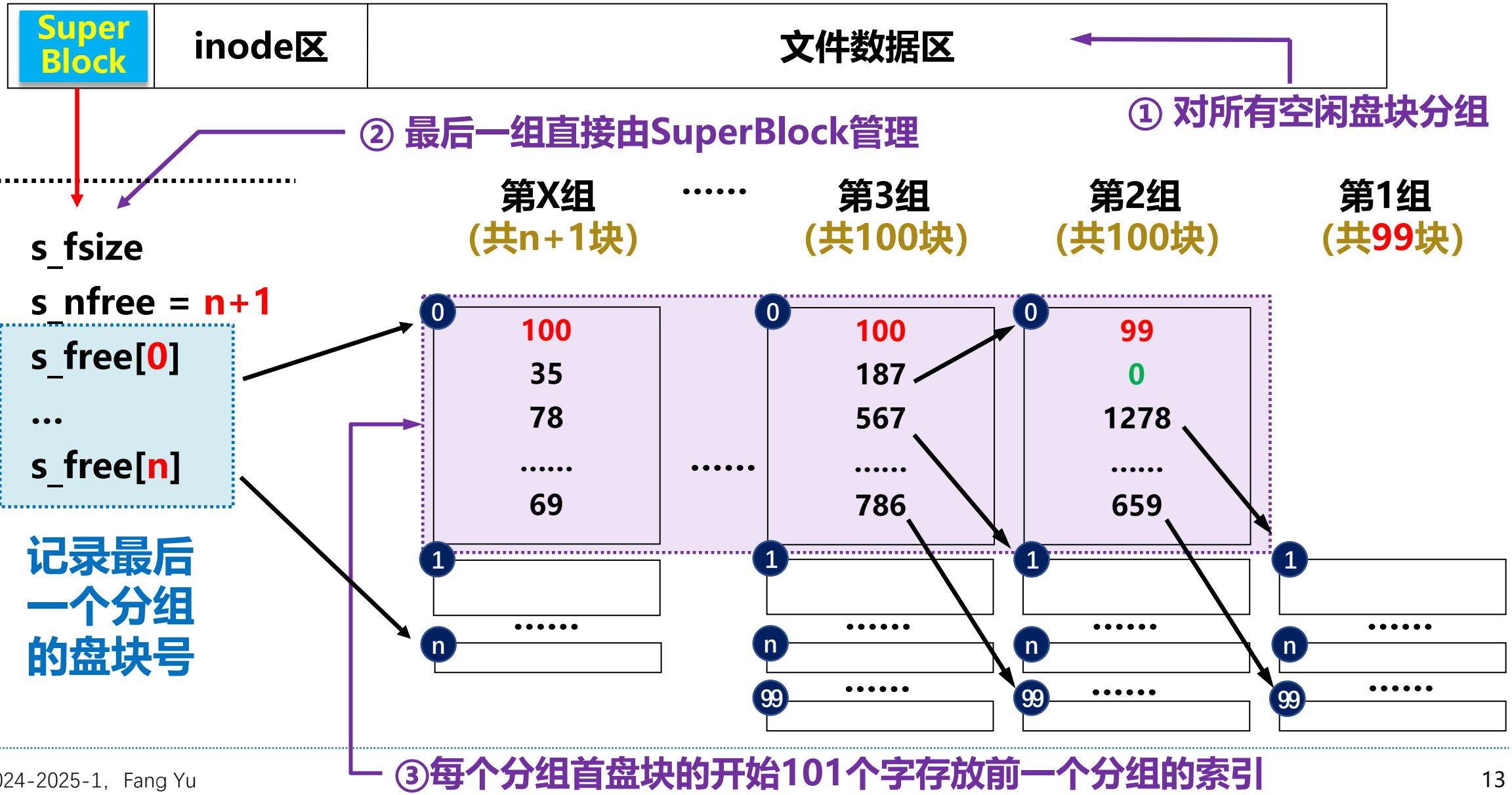
存储资源管理信息块（SuperBlock，200~201#盘块）



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

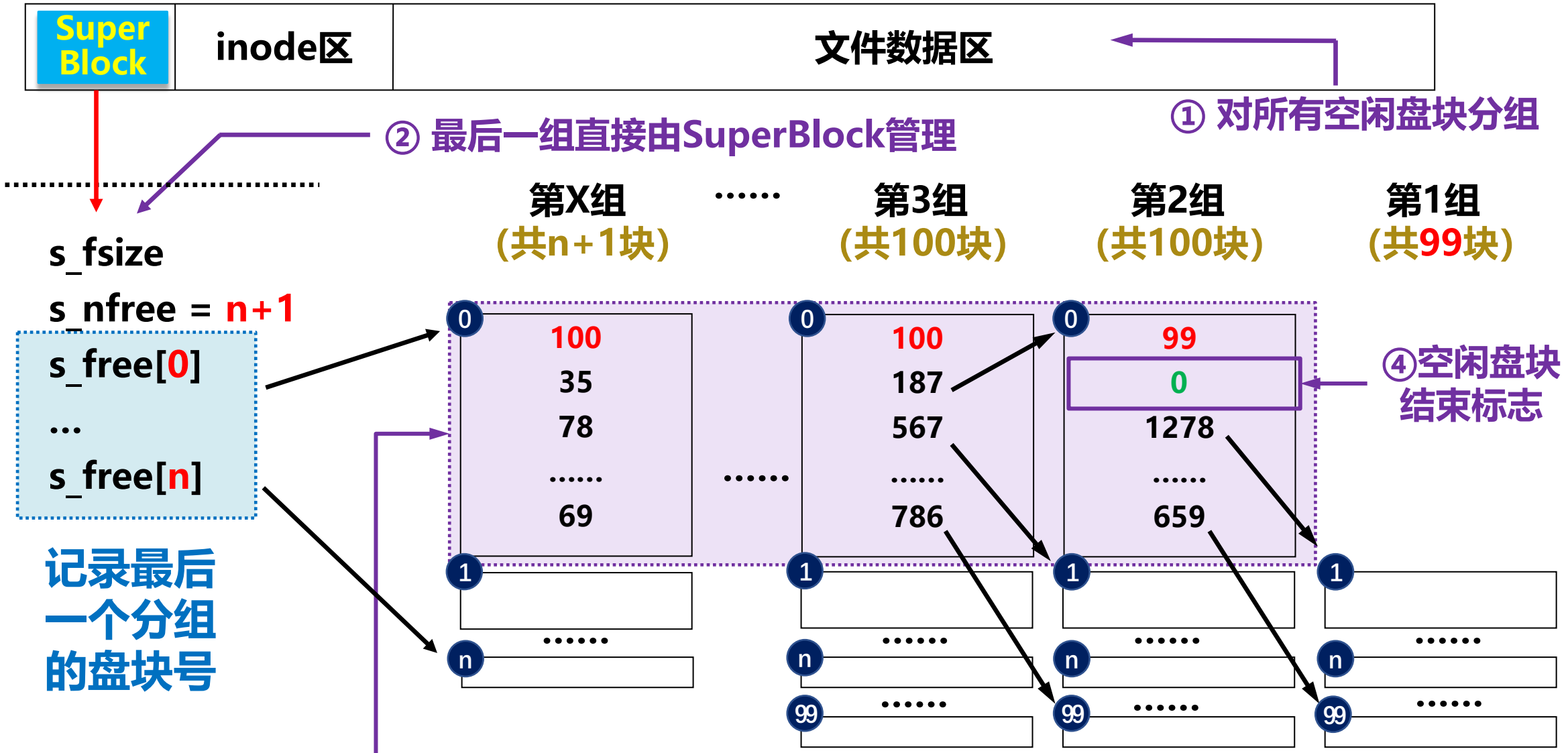
对文件数据区的管理



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

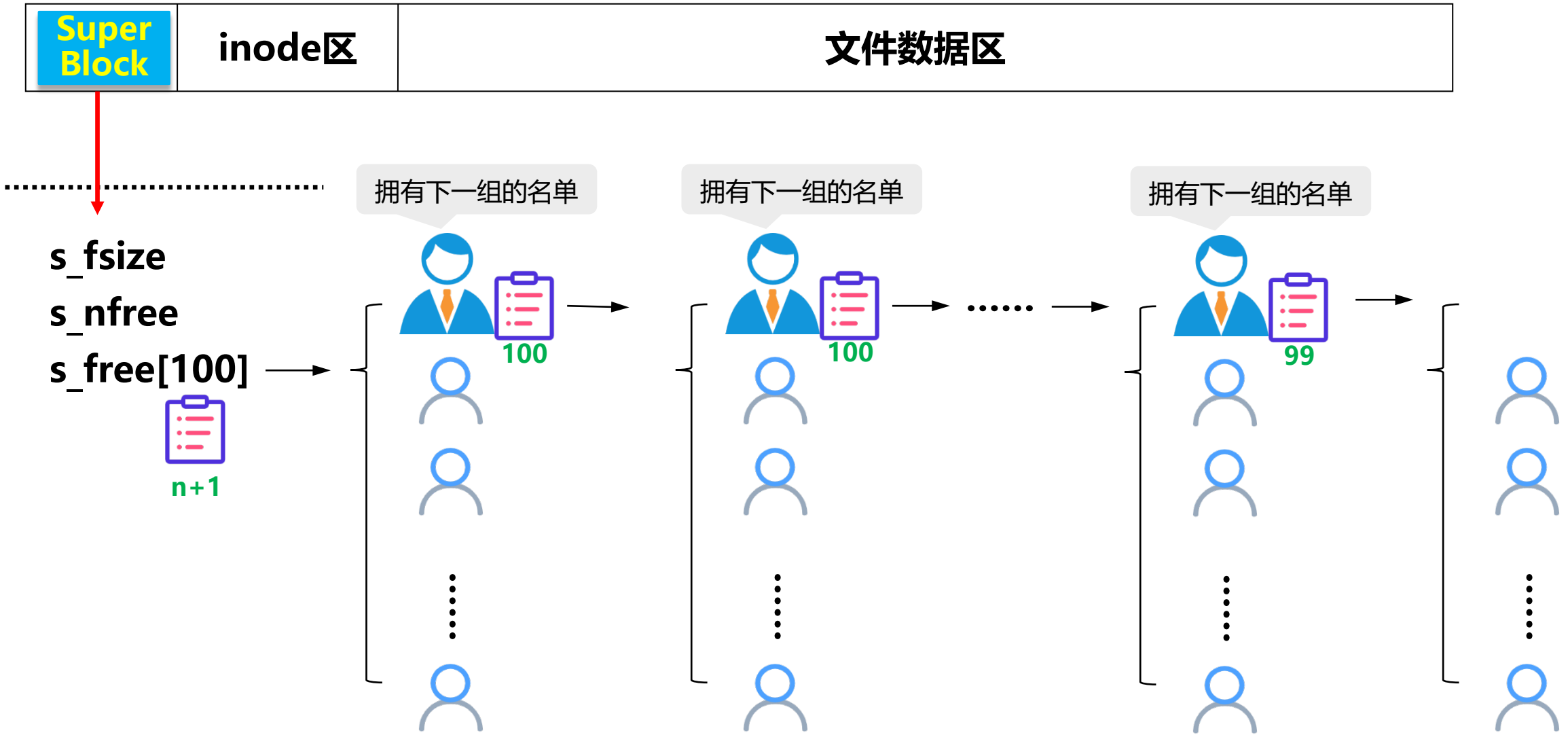
对文件数据区的管理



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

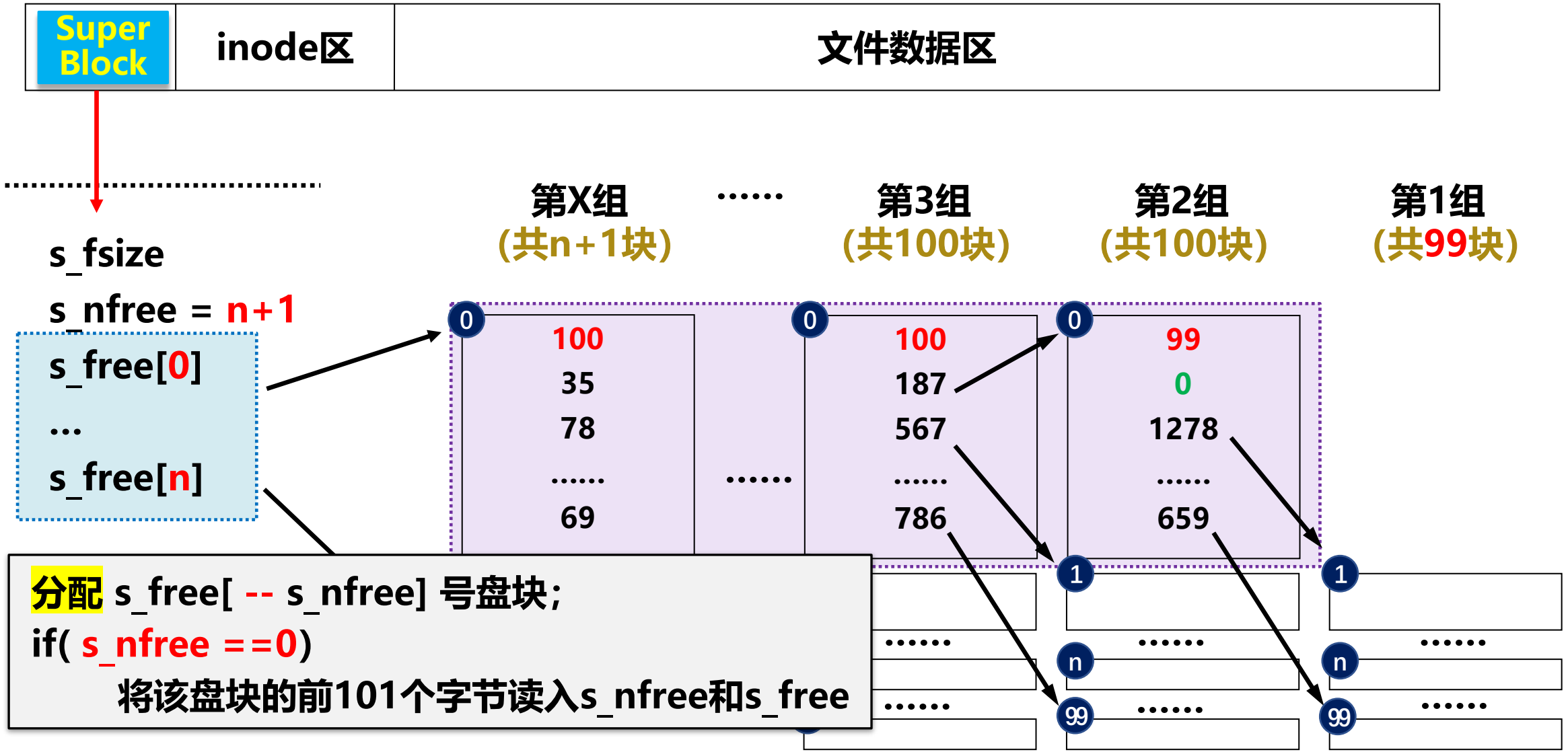
对文件数据区的管理



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

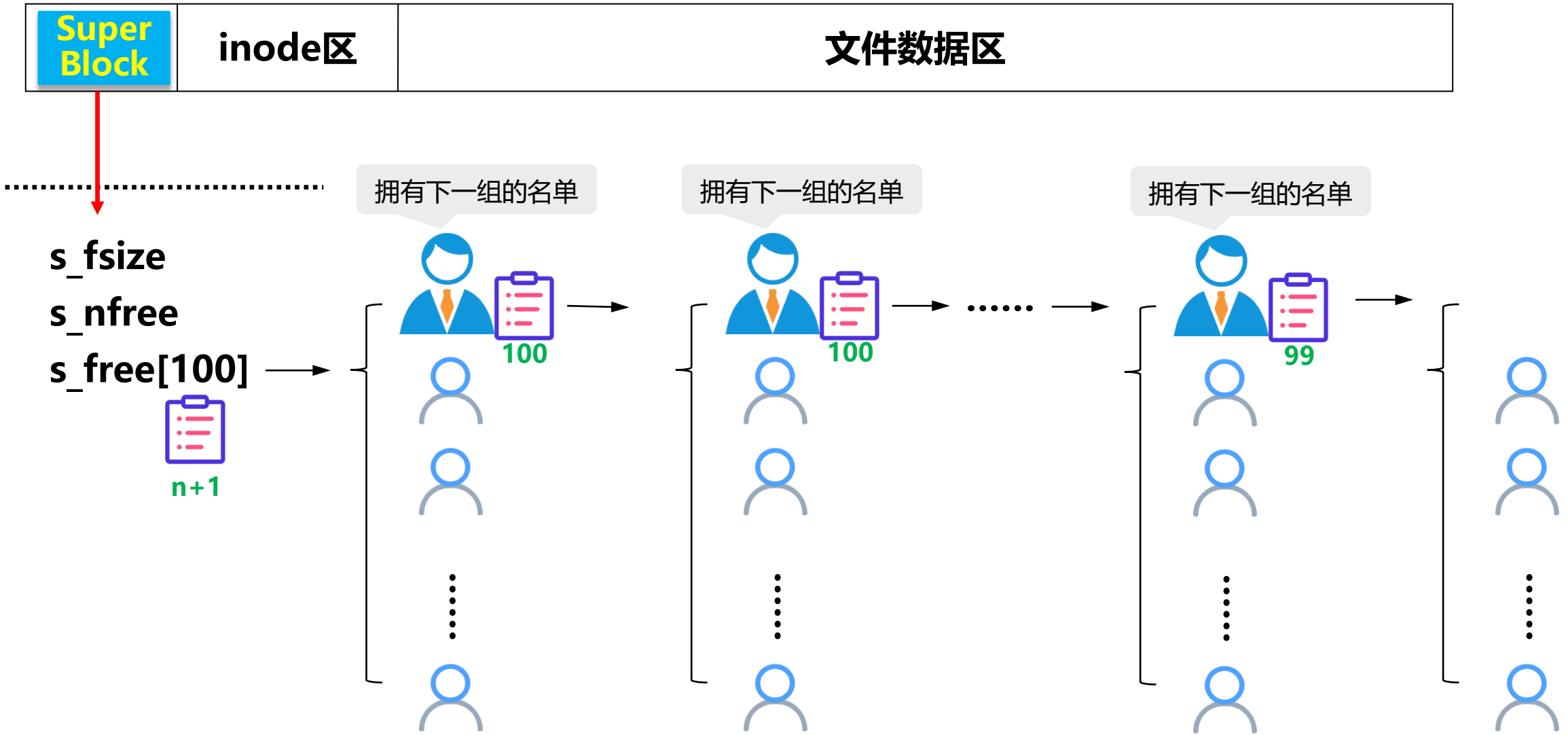
对文件数据区的管理



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

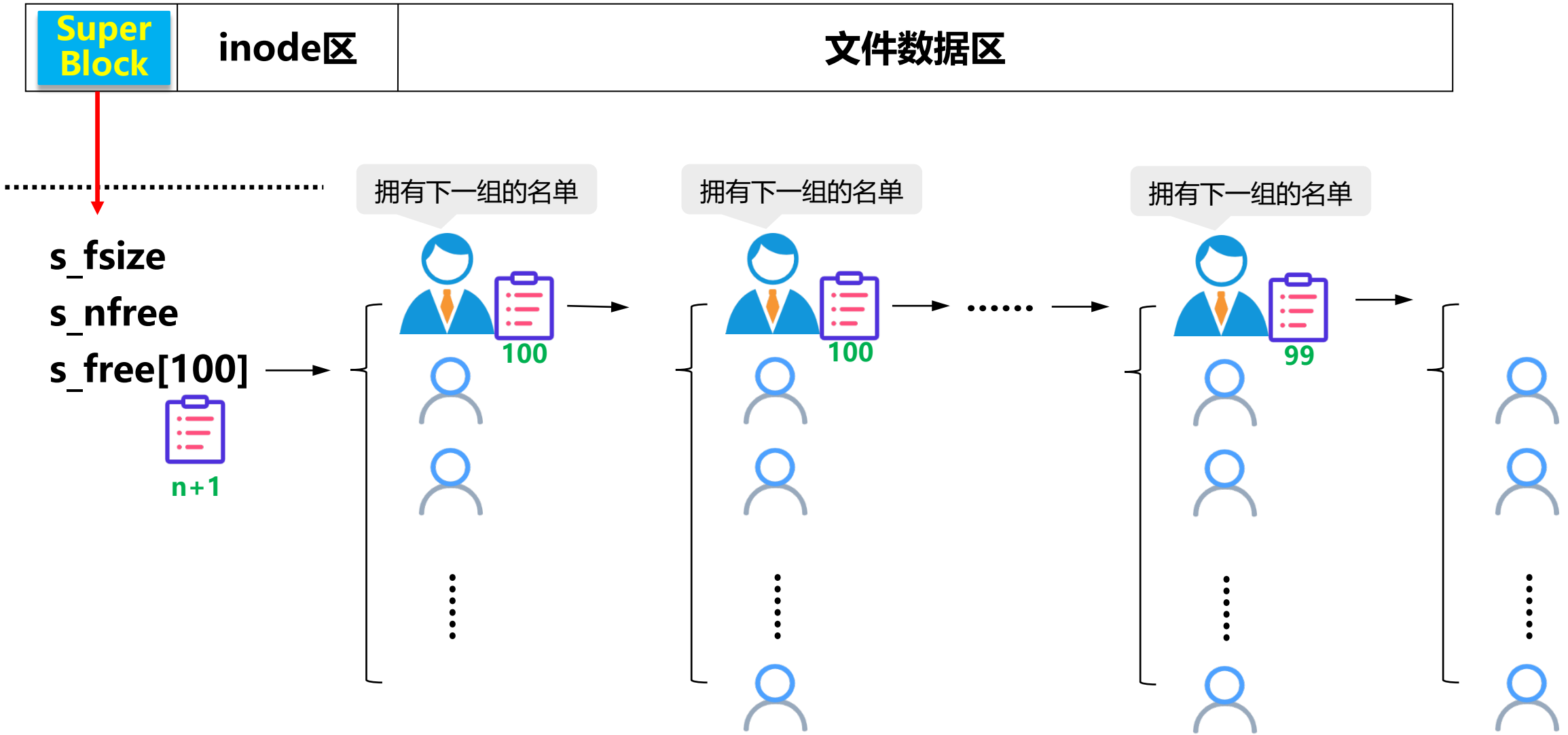
对文件数据区的管理



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

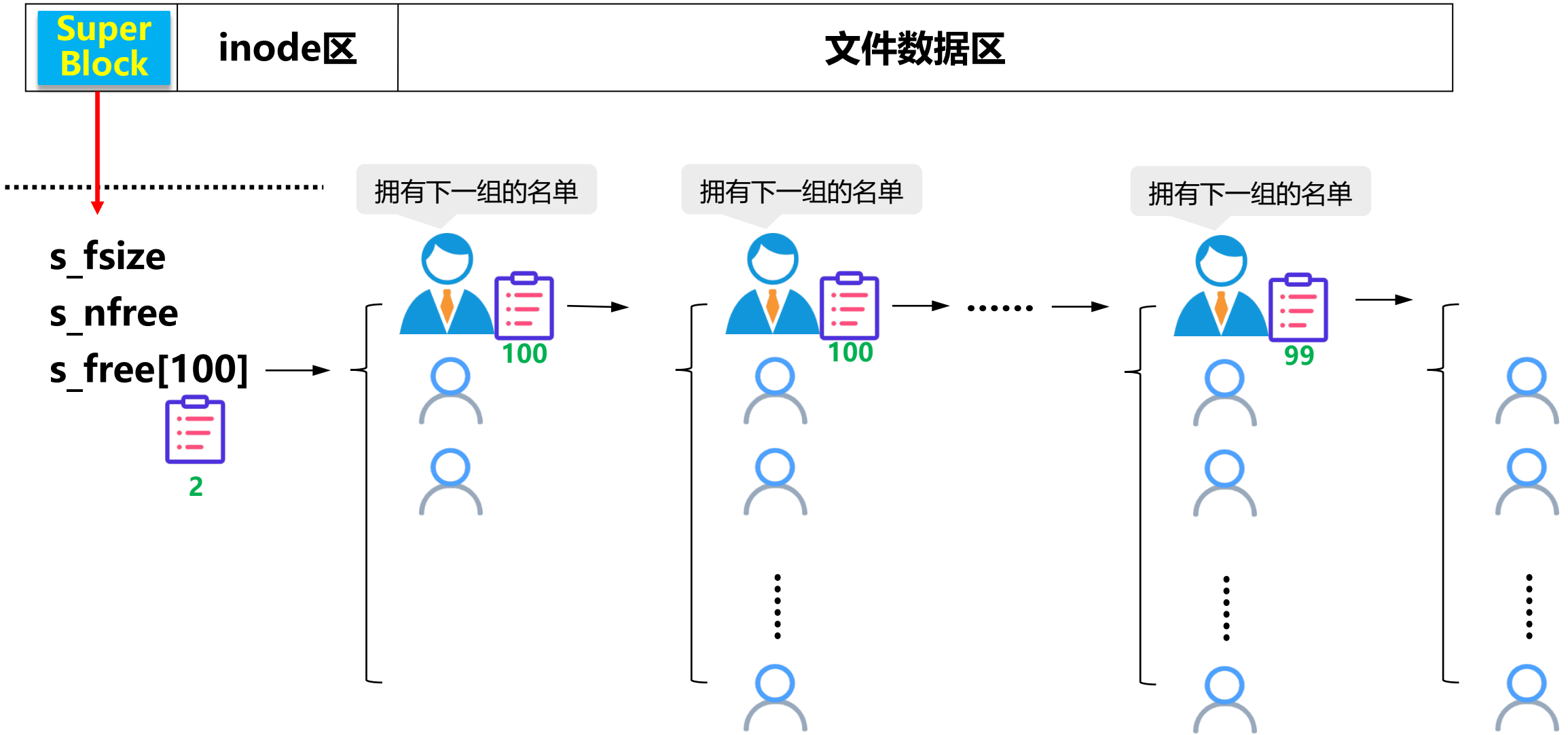
对文件数据区的管理



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

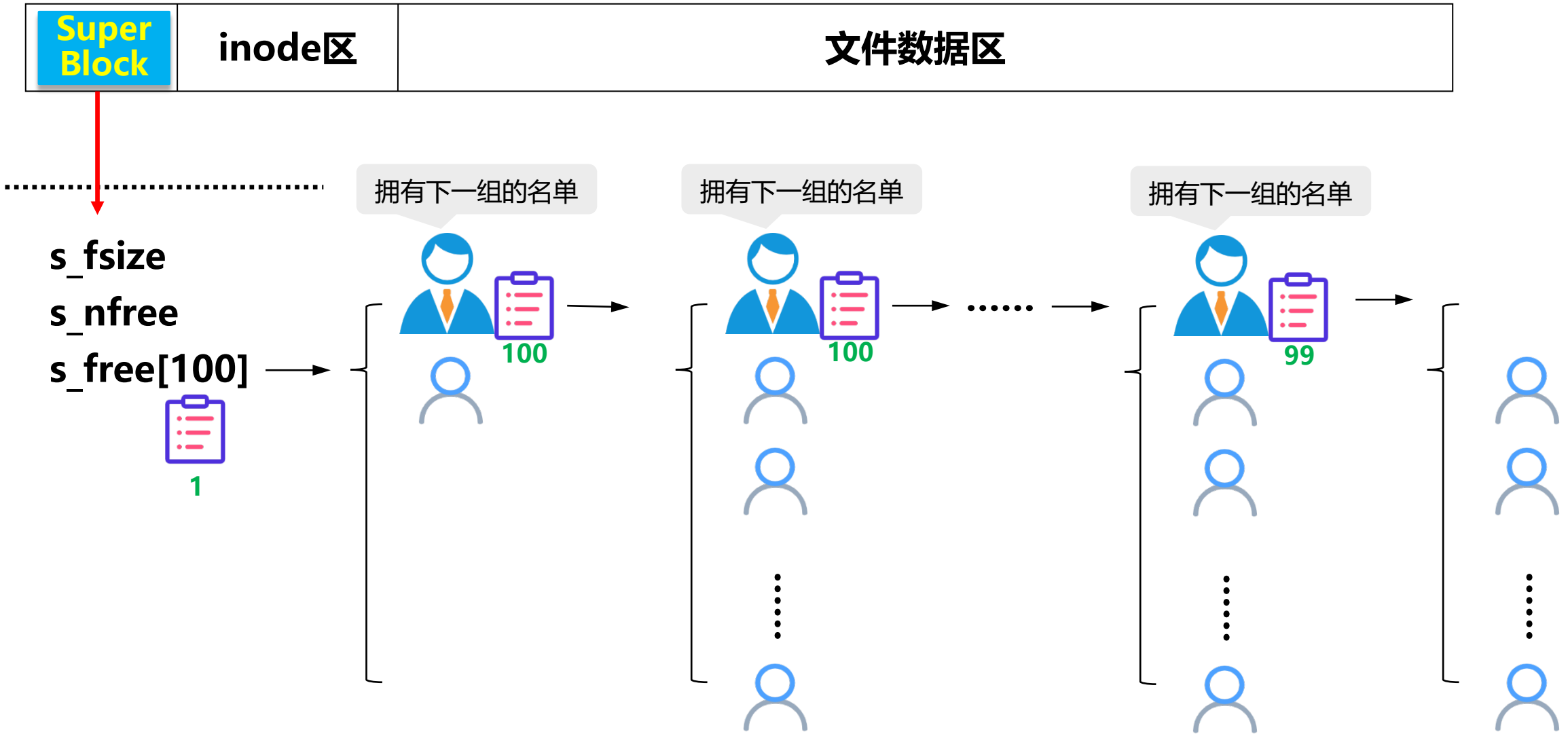
对文件数据区的管理



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

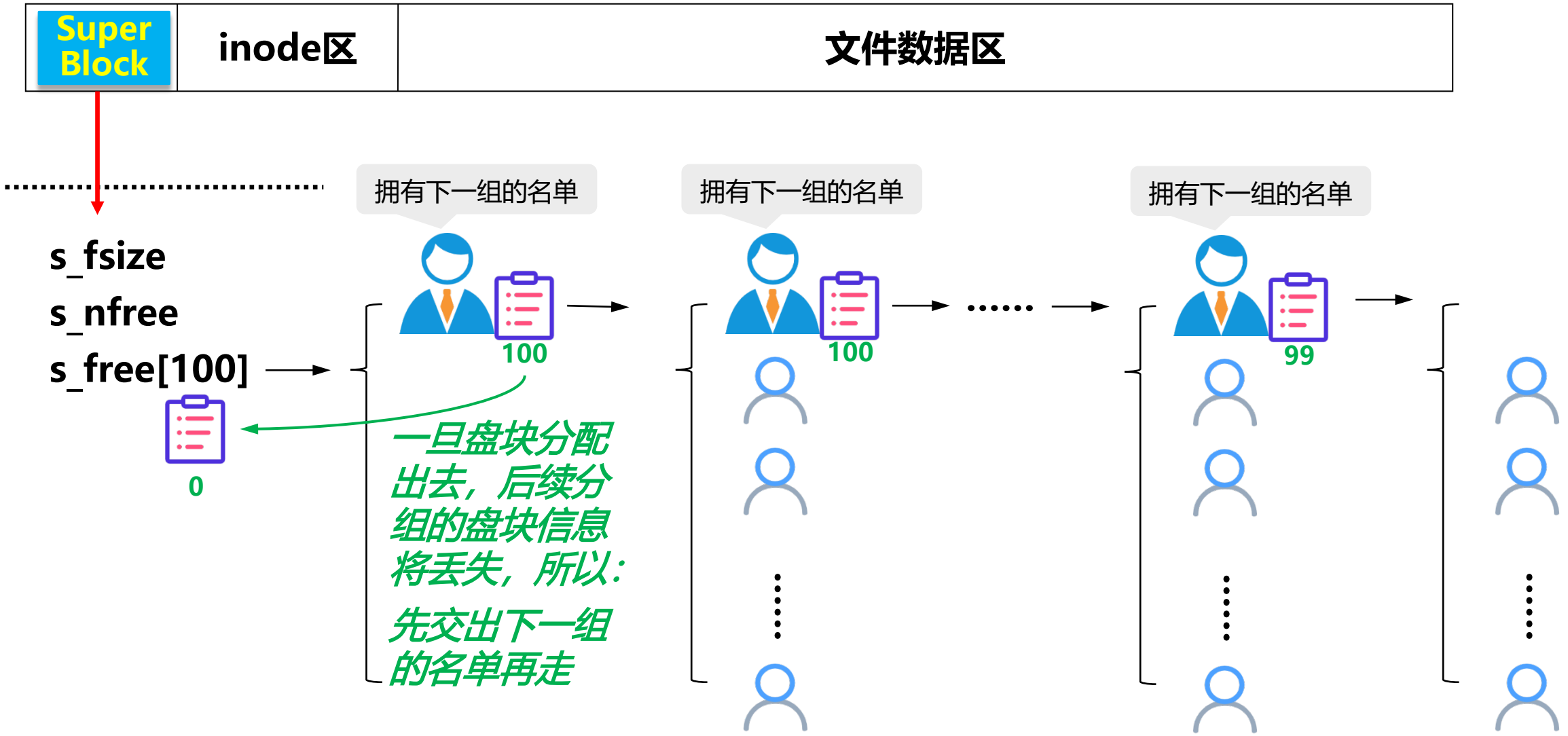
对文件数据区的管理



UNIX文件系统的磁盘空间管理

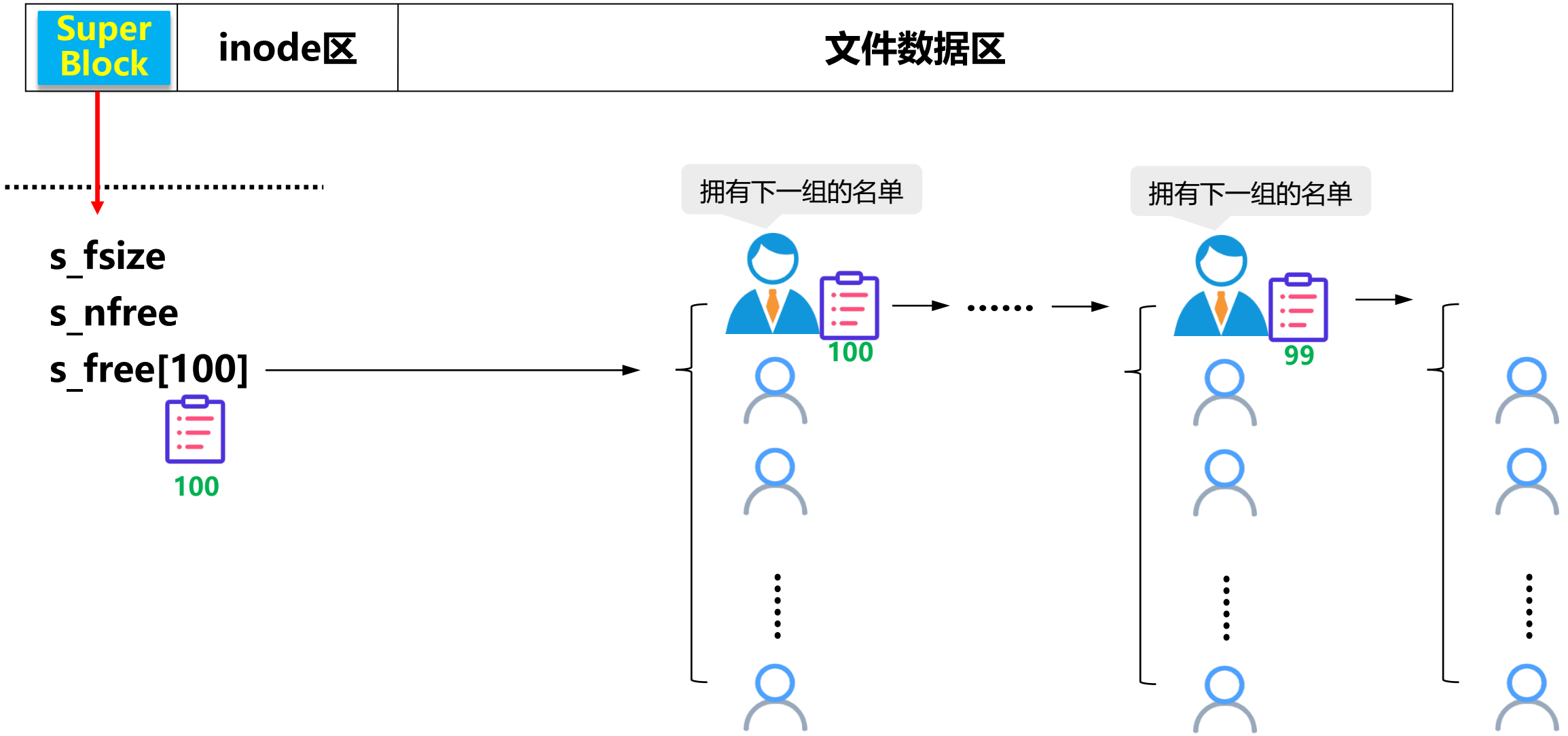
存储资源管理信息块（SuperBlock，200~201#盘块）

对文件数据区的管理



UNIX文件系统的磁盘空间管理

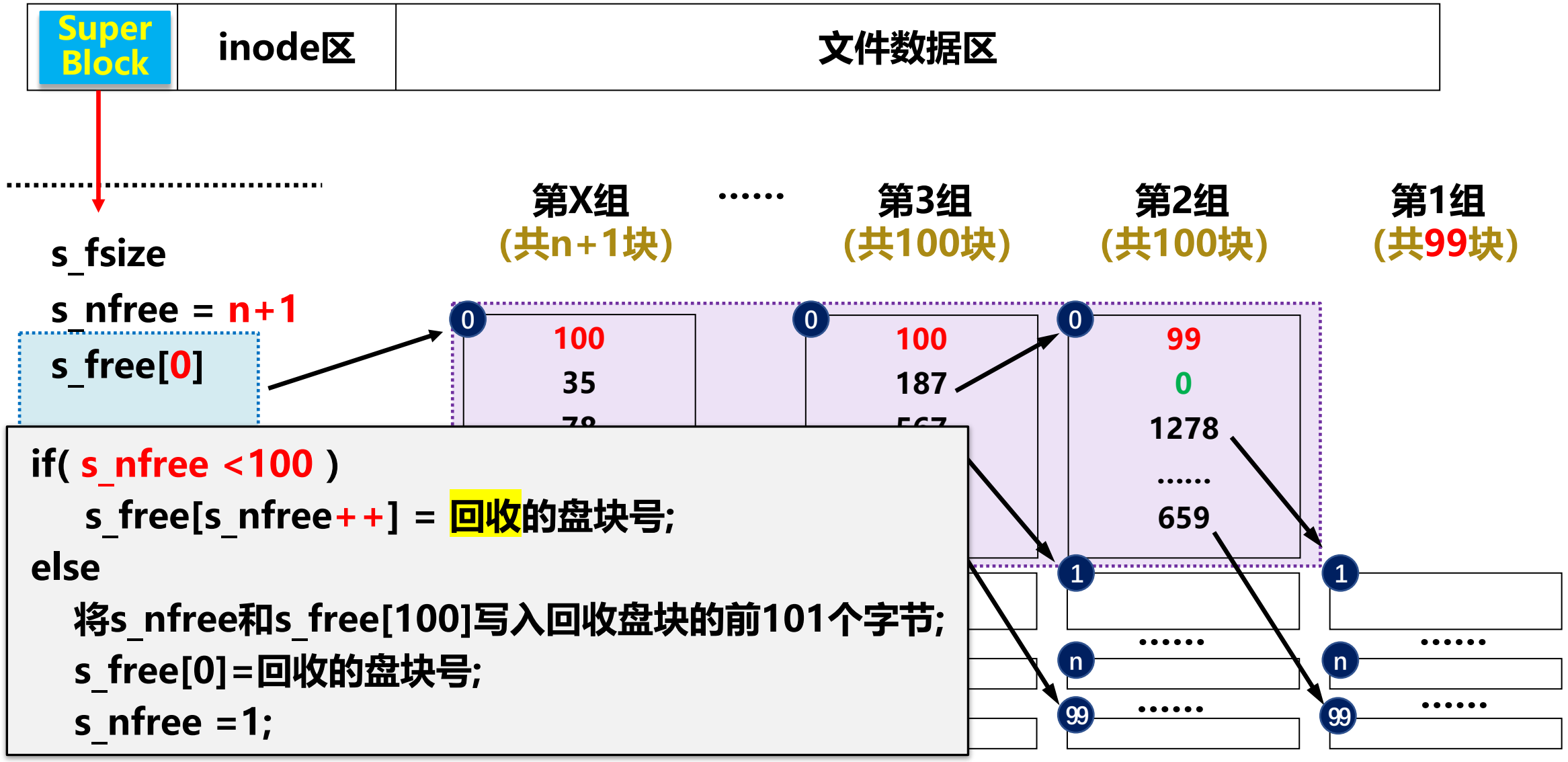
存储资源管理信息块（SuperBlock，200~201#盘块）



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

对文件数据区的管理

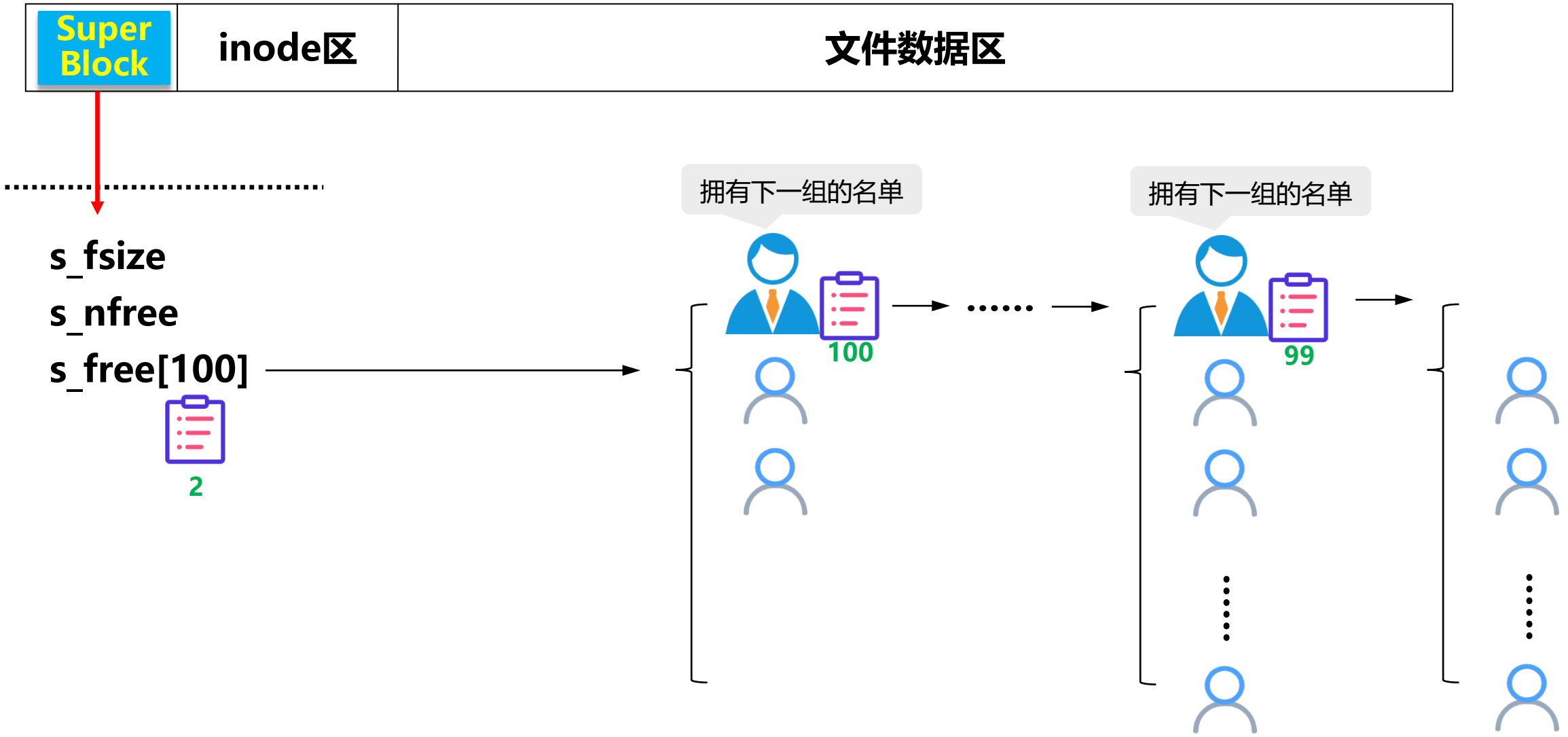



```
if( s_nfree <100 )
    s_free[s_nfree++] = 回收的盘块号;
else
    将s_nfree和s_free[100]写入回收盘块的前101个字节;
    s_free[0]=回收的盘块号;
    s_nfree =1;
```

UNIX文件系统的磁盘空间管理

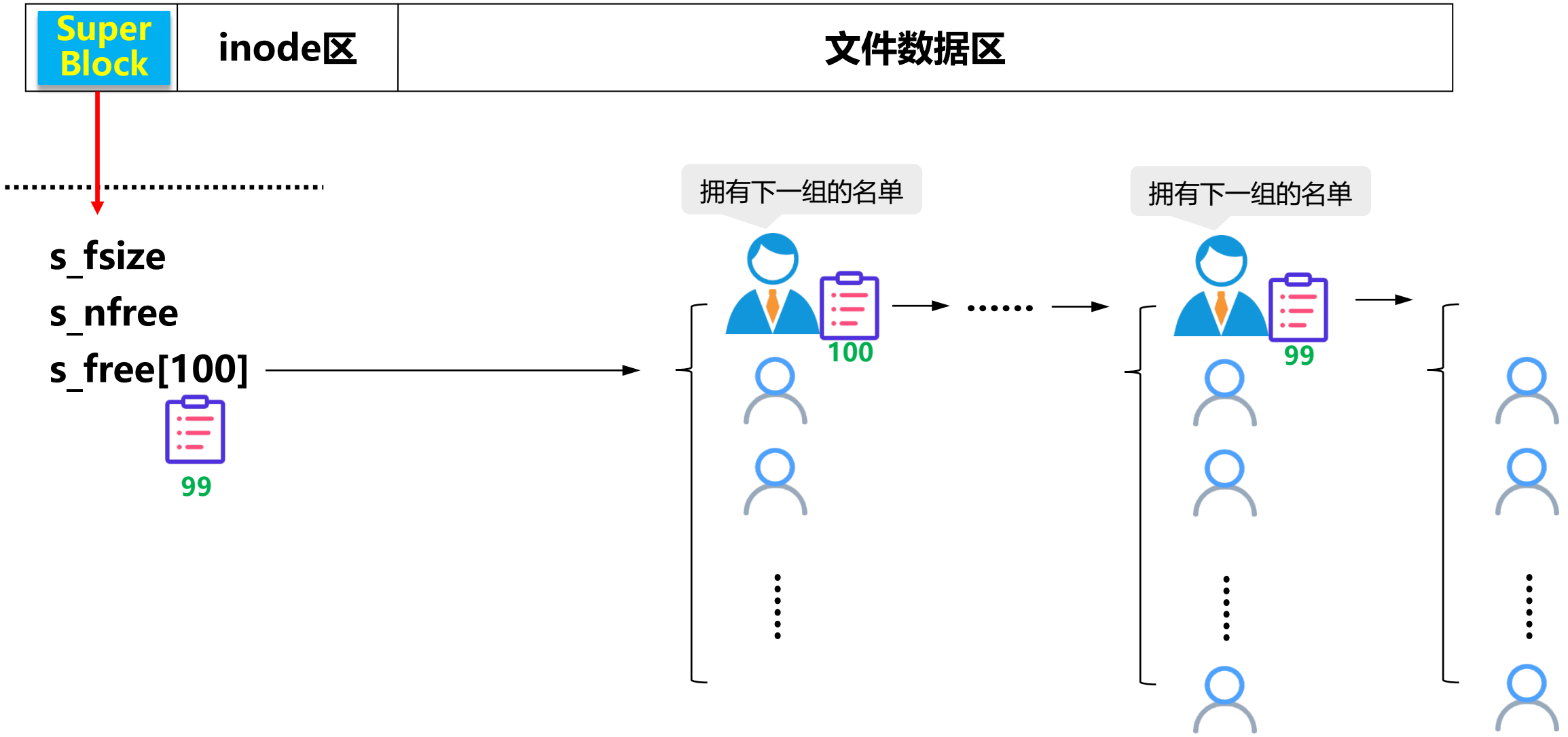
存储资源管理信息块（SuperBlock，200~201#盘块）

对文件数据区的管理



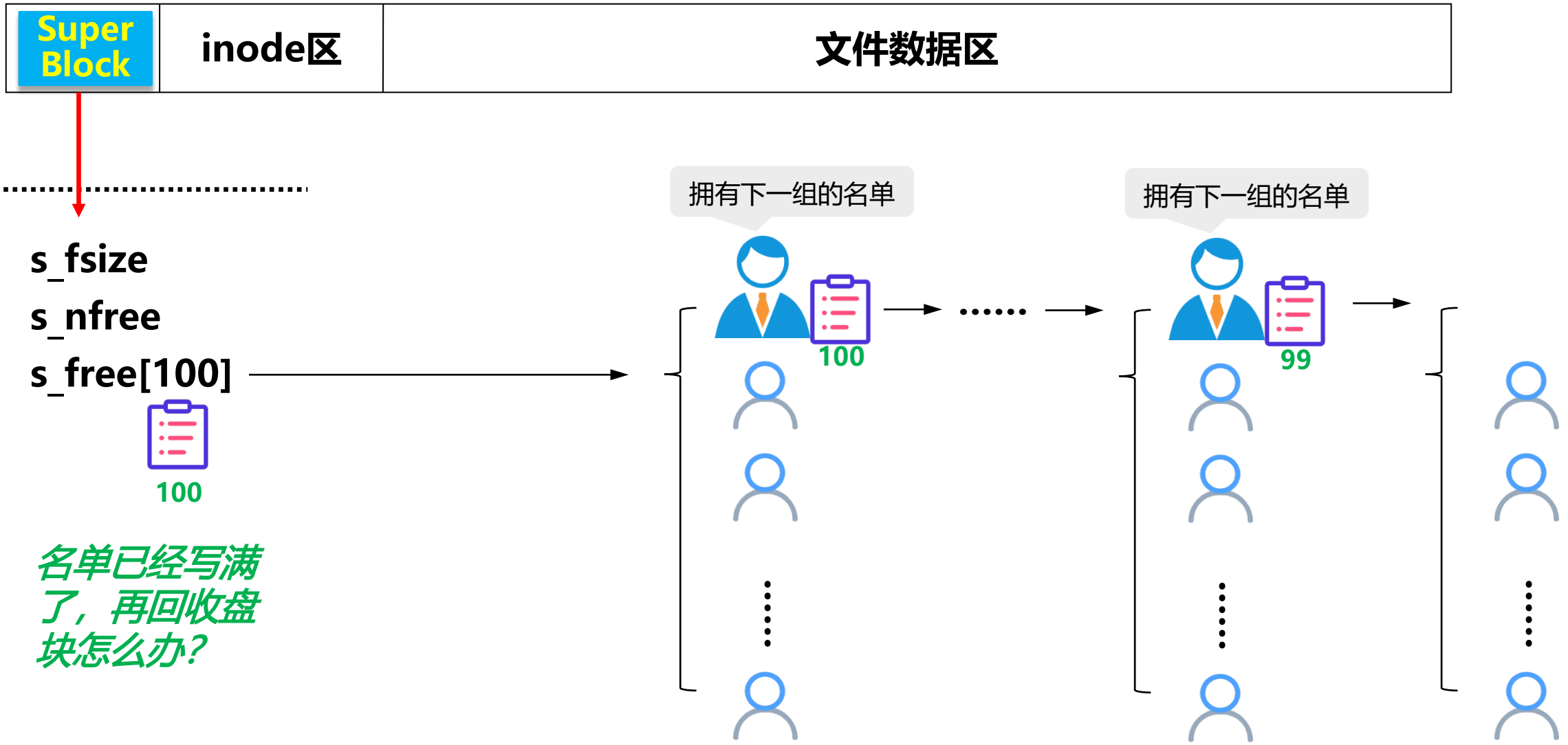
UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

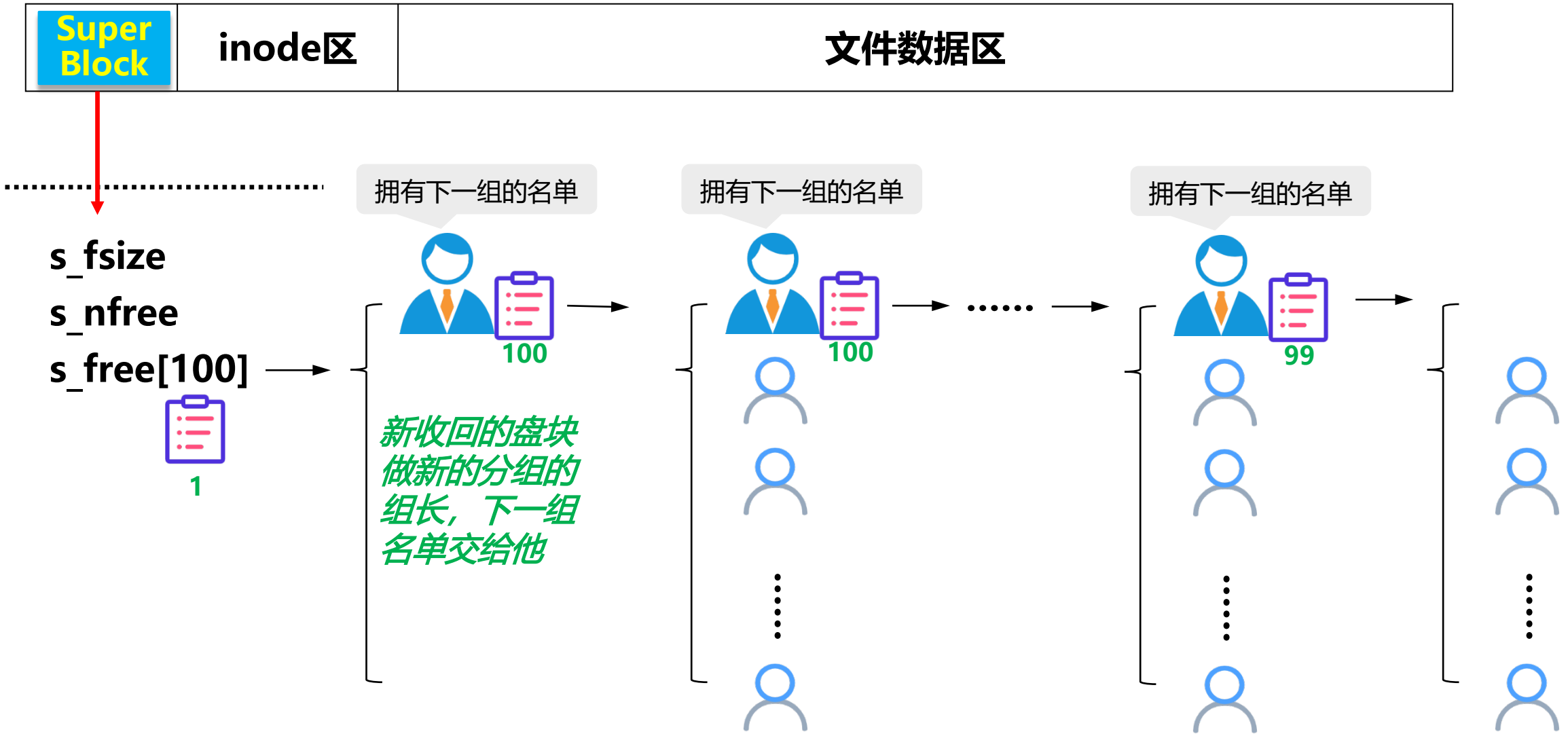


名单已经写满了，再回收盘块怎么办?

UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

对文件数据区的管理



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

SuperBlock

s_nfree :	2
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

SuperBlock

s_nfree :	1
s_free[0]:	2000
s_free[97]:	/
.....	
s_free[98]:	/
s_free[99]:	/

继续分配

SuperBlock

s_nfree :	1
s_free[0]:	2000
.....	
s_free[97]:	/
s_free[98]:	/
s_free[99]:	/

2000号盘块中记录下一个分组的盘块号

100
3000
.....
3750
3751
3752

继续分配

2000号盘块分配之前，先将前101个字读入SuperBlock

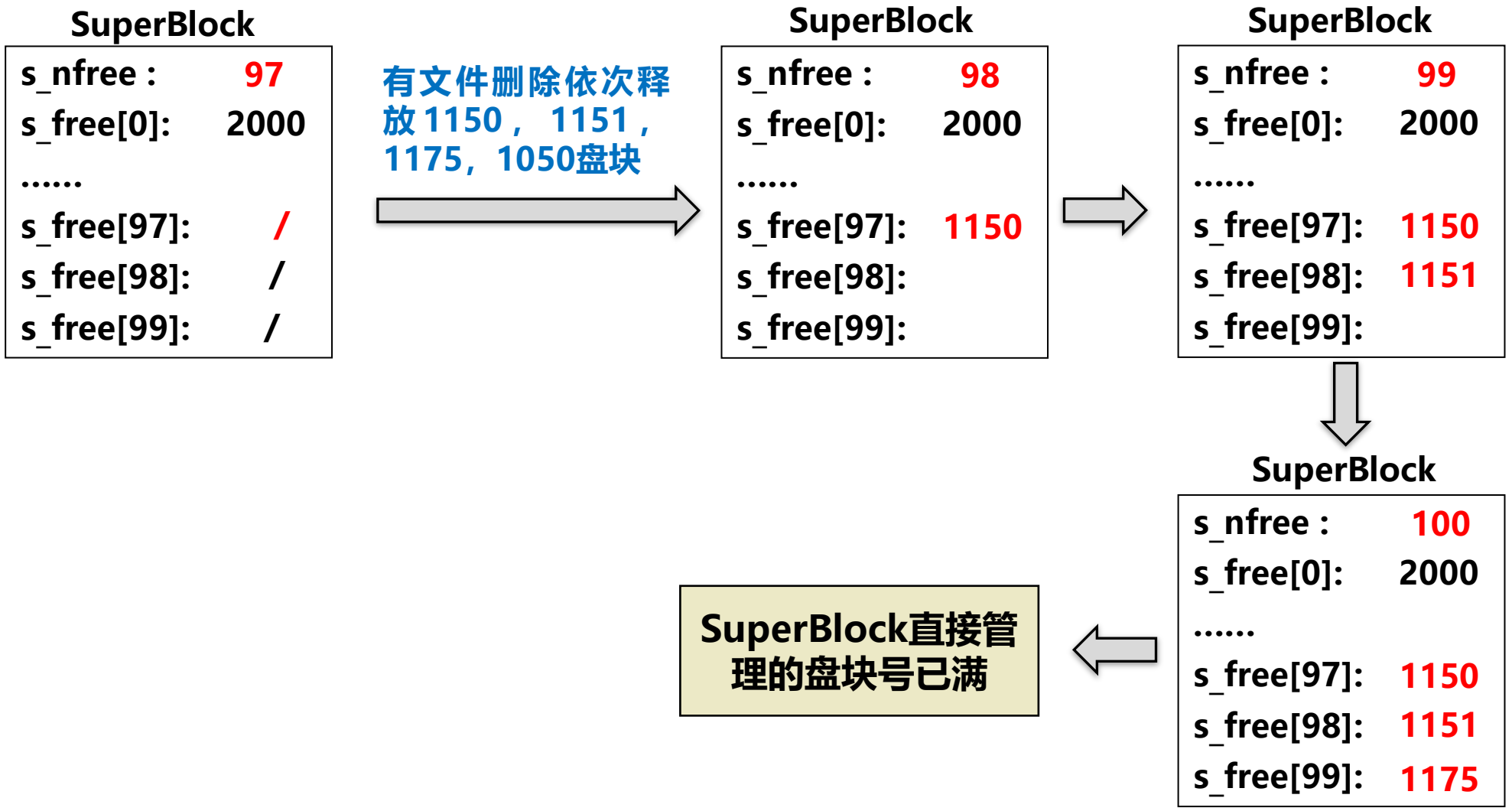
SuperBlock

s_nfree :	100
s_free[0]:	3000
.....	
s_free[97]:	3750
s_free[98]:	3751
s_free[99]:	3752

UNIX文件系统的磁盘空间管理

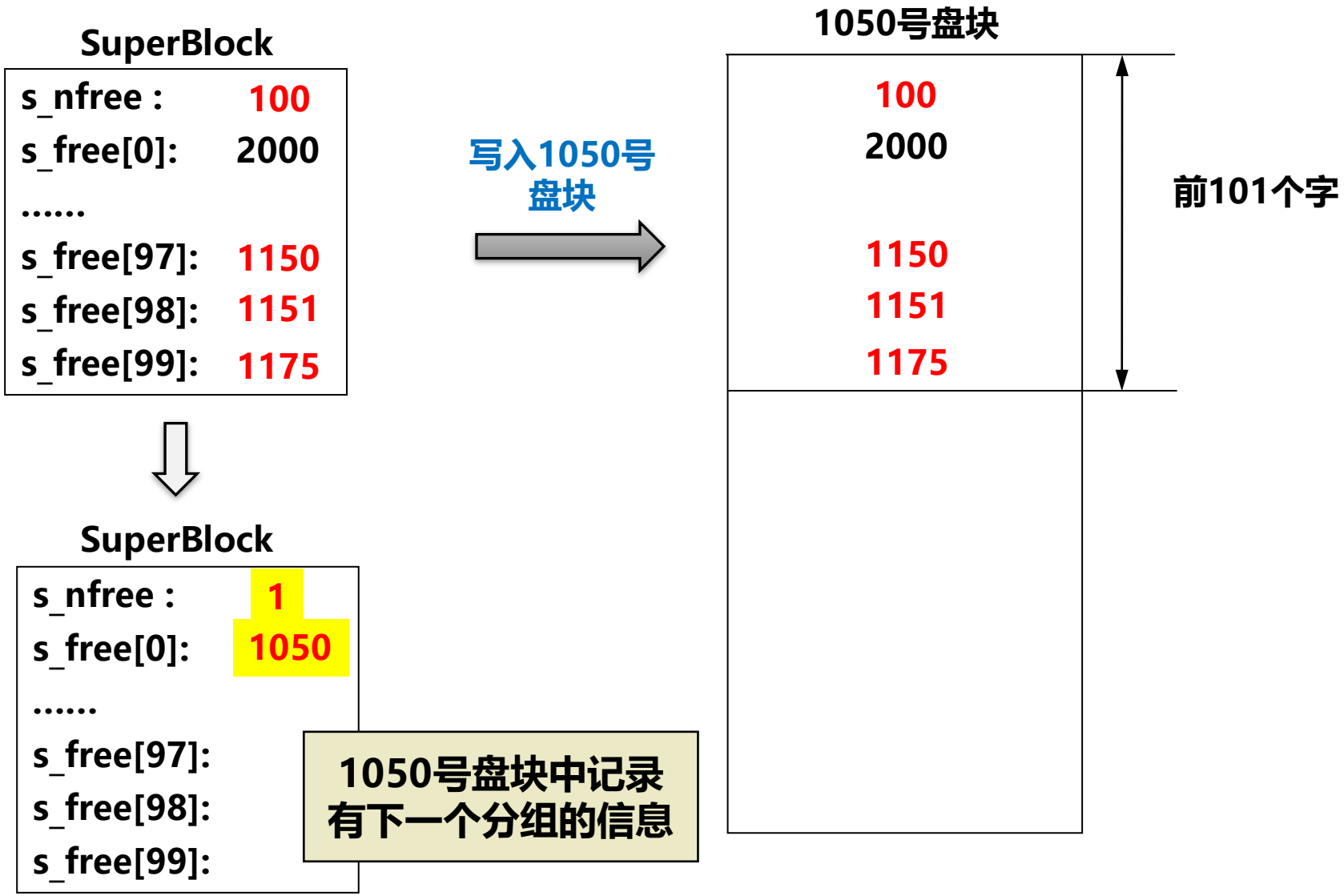
存储资源管理信息块（SuperBlock，200~201#盘块）

对文件数据区的管理



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）



UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）


```
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

由于SuperBlock非常重要，且访问频率高，系统初启时，会通过两个缓存将SuperBlock读入内存，在内存构建副本。——文件系统的挂载

SuperBlock占用两个盘块，一共1024个字节

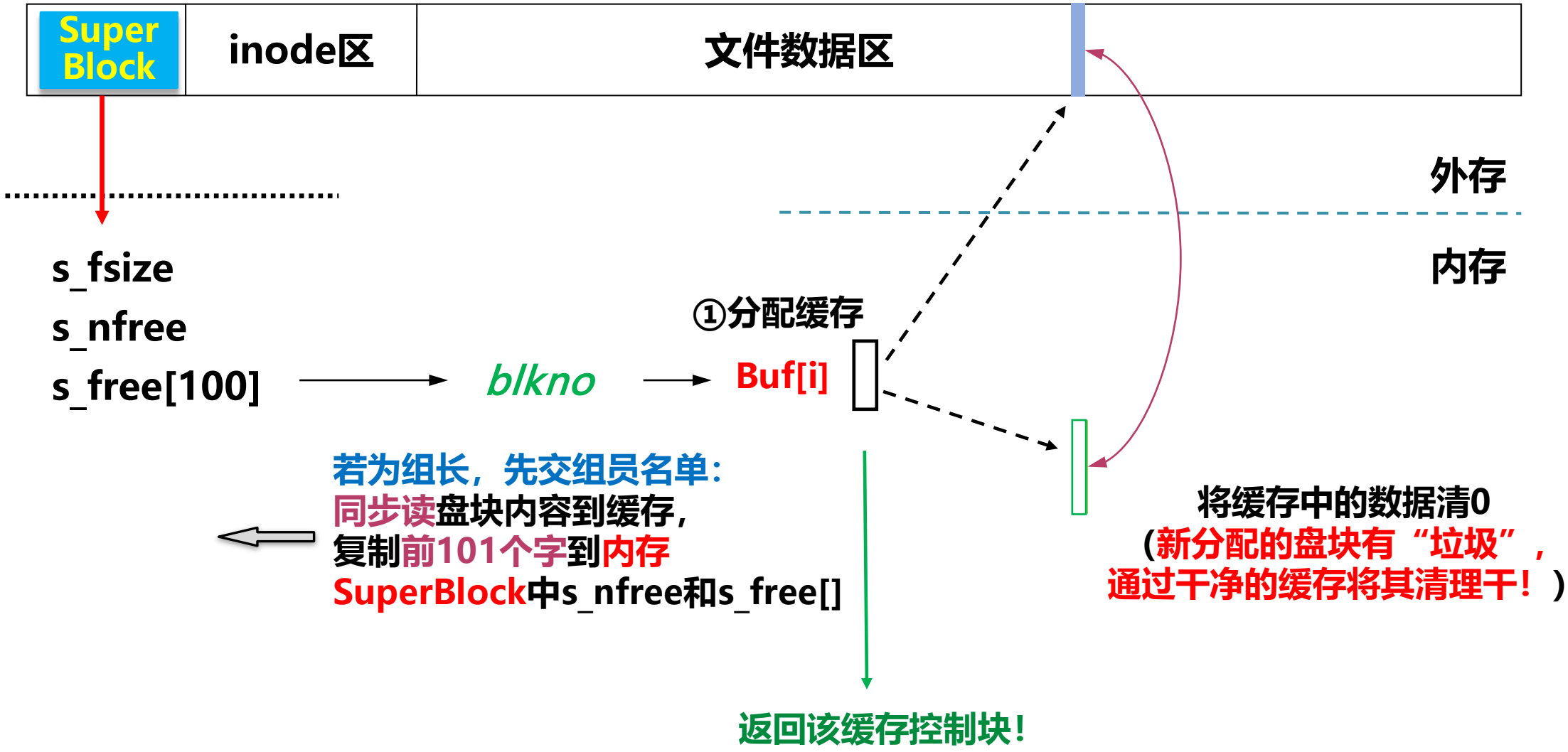
如果SuperBlock的内存被修改过，文件系统卸载时，需写回磁盘。

磁盘分布

UNIX文件系统的磁盘空间管理

存储资源管理信息块（SuperBlock，200~201#盘块）

盘块分配过程



主 要 内 容

6.1 文件系统概述

6.2 文件的逻辑结构与物理结构

6.3 文件存储空间管理

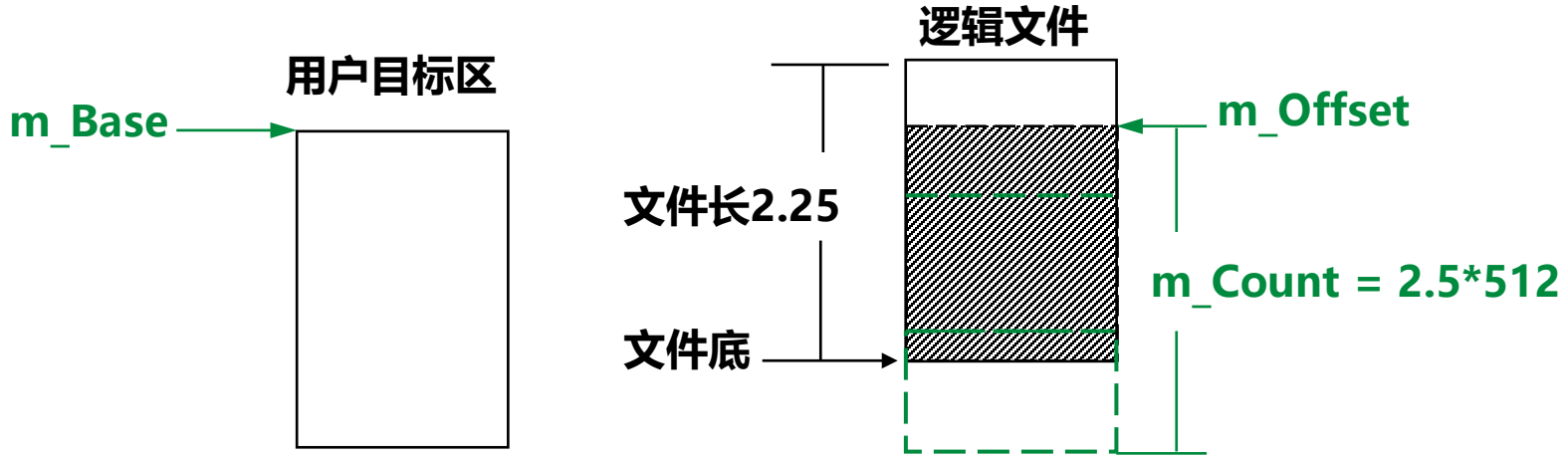
- 文件存储空间管理方法
- UNIX磁盘存储空间管理
- UNIX文件的长度变化

6.4 文件系统的目录管理

UNIX文件系统的读写操作

读操作

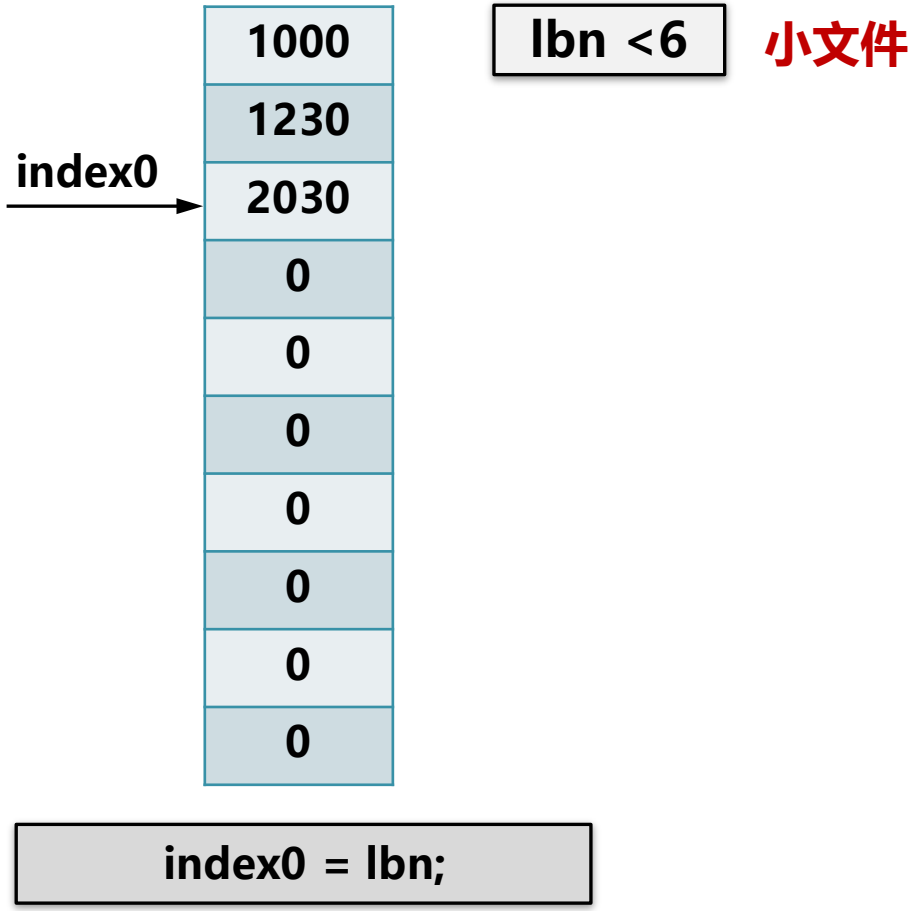
从进程空间写2.5块到文件的过程



写操作超出文件长度怎么办?

UNIX文件系统的静态结构

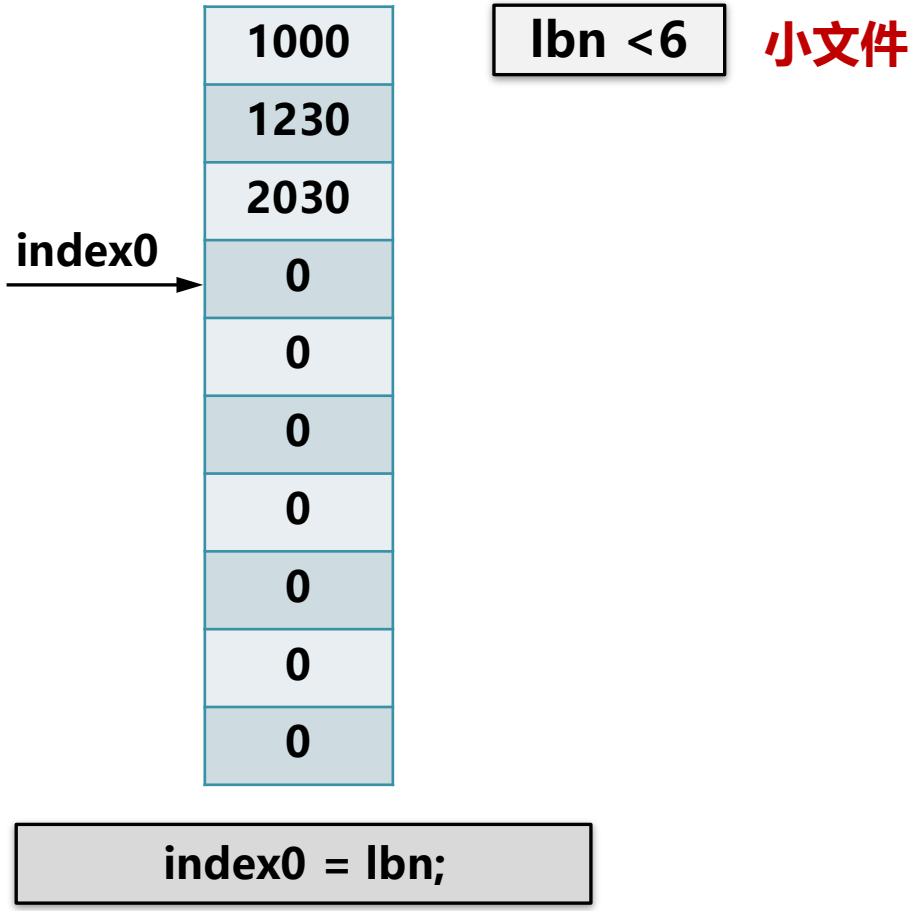

```
int Inode::Bmap(int lbn)
```



文件增长，但没有超出当前逻辑块

UNIX文件系统的静态结构

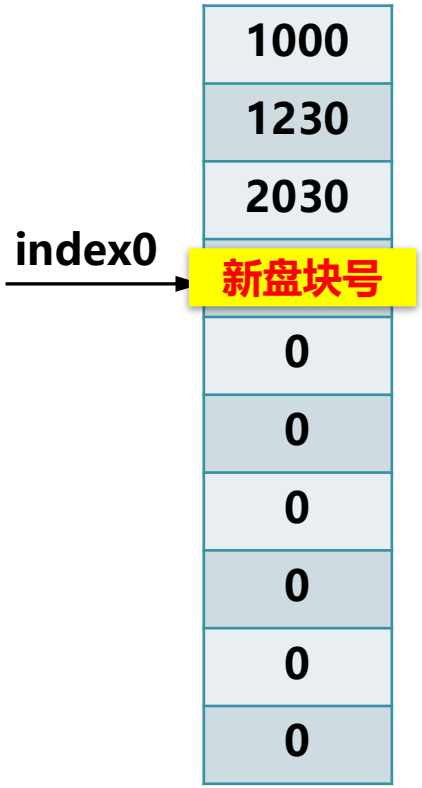

```
int Inode::Bmap(int lbn)
```



文件继续增长，超出当前逻辑块

UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```

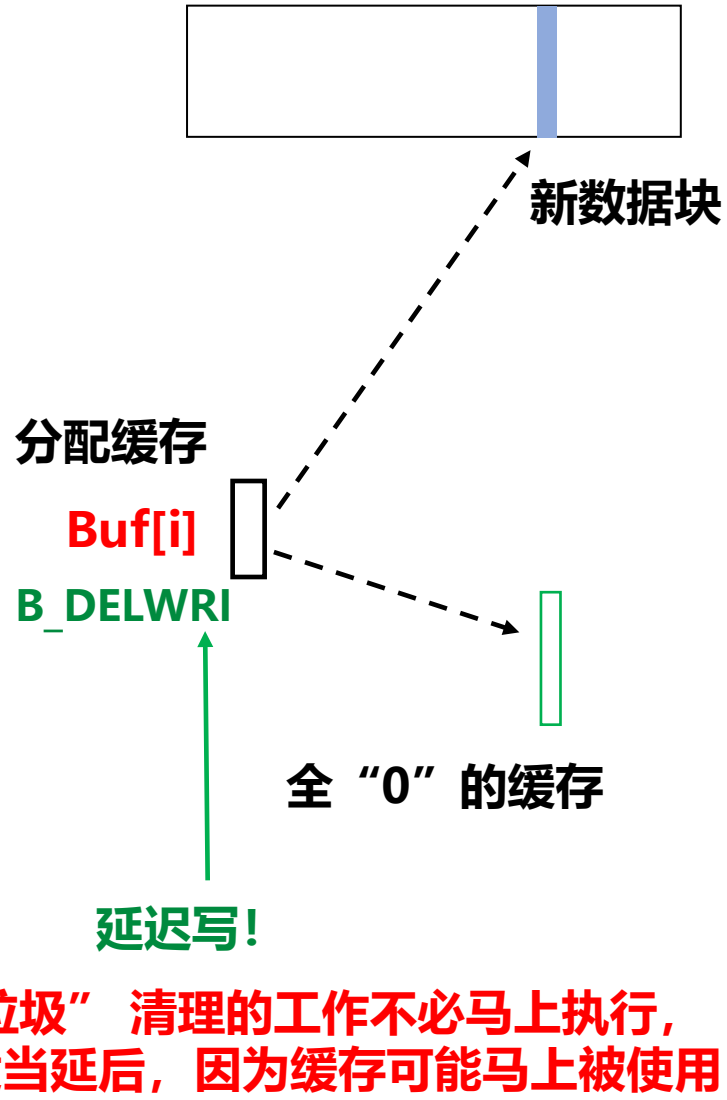


lbn <6 小文件

如果某次写操作时：index0 = lbn;
i_addr[index0]= 0 ->
此次写操作将导致文件变大

index0 = lbn;

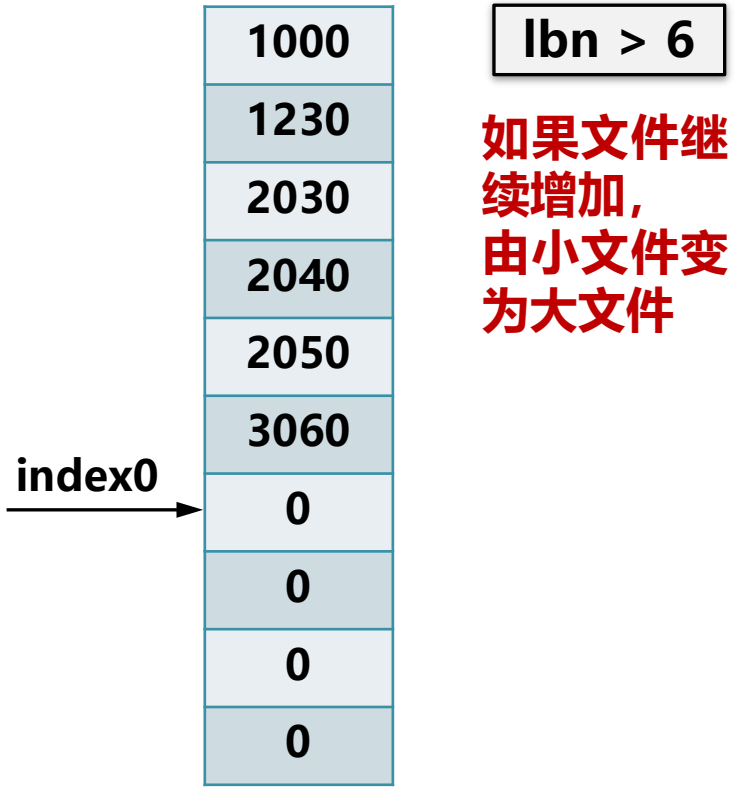
如果i_addr[index0] = 0，
分配新的数据盘块，延迟写
i_addr[index0] = 新数据盘块号



“垃圾” 清理的工作不必马上执行，
可适当延后，因为缓存可能马上被使用

UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```



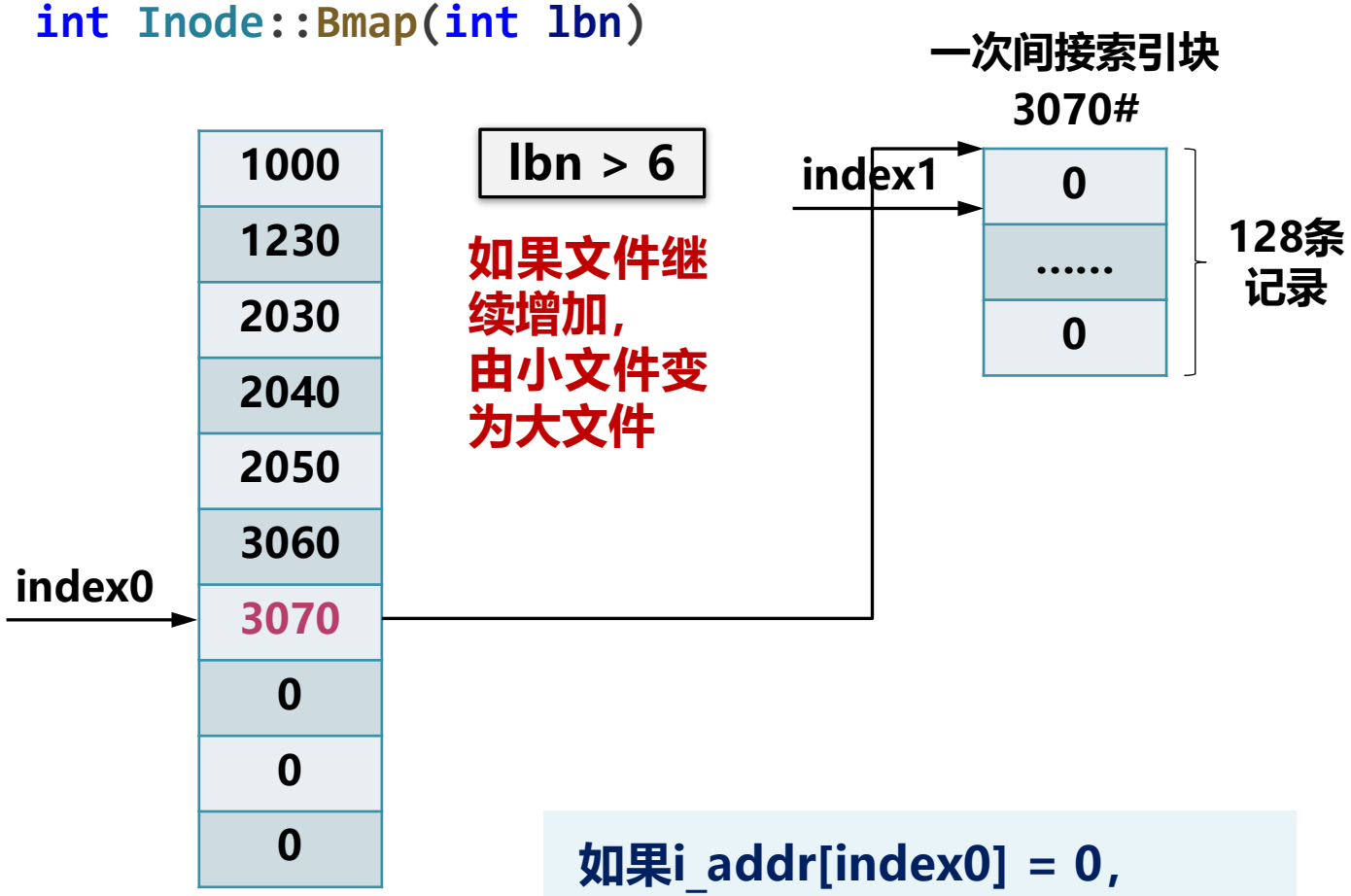
lbn > 6

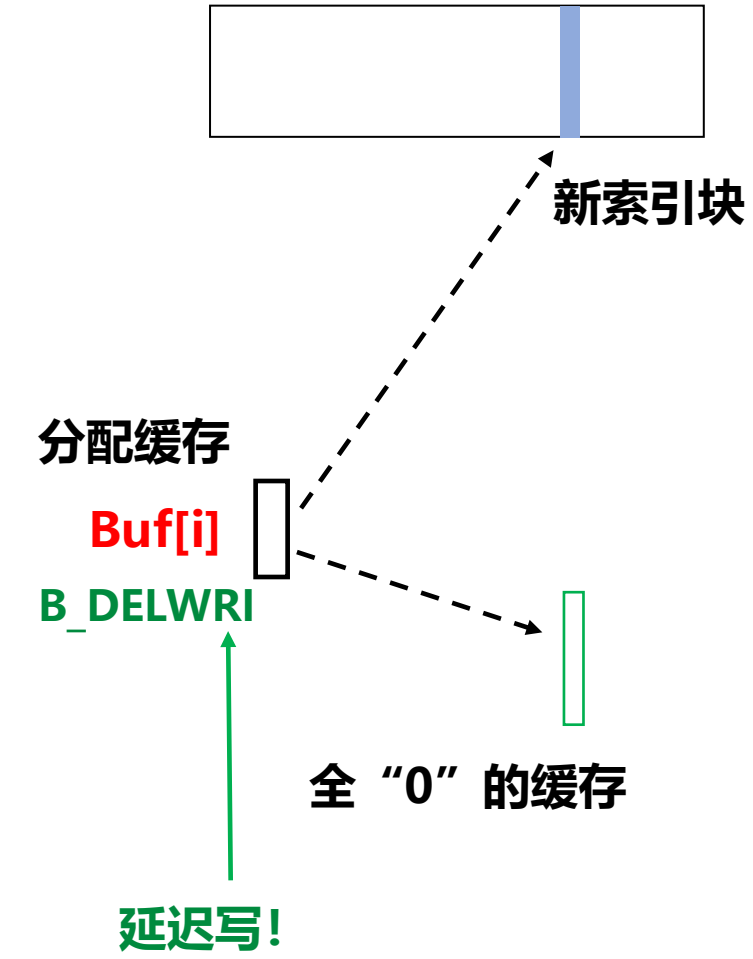
如果文件继续增加，由小文件变为大文件

文件继续增长，需要新的一级索引块

UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```



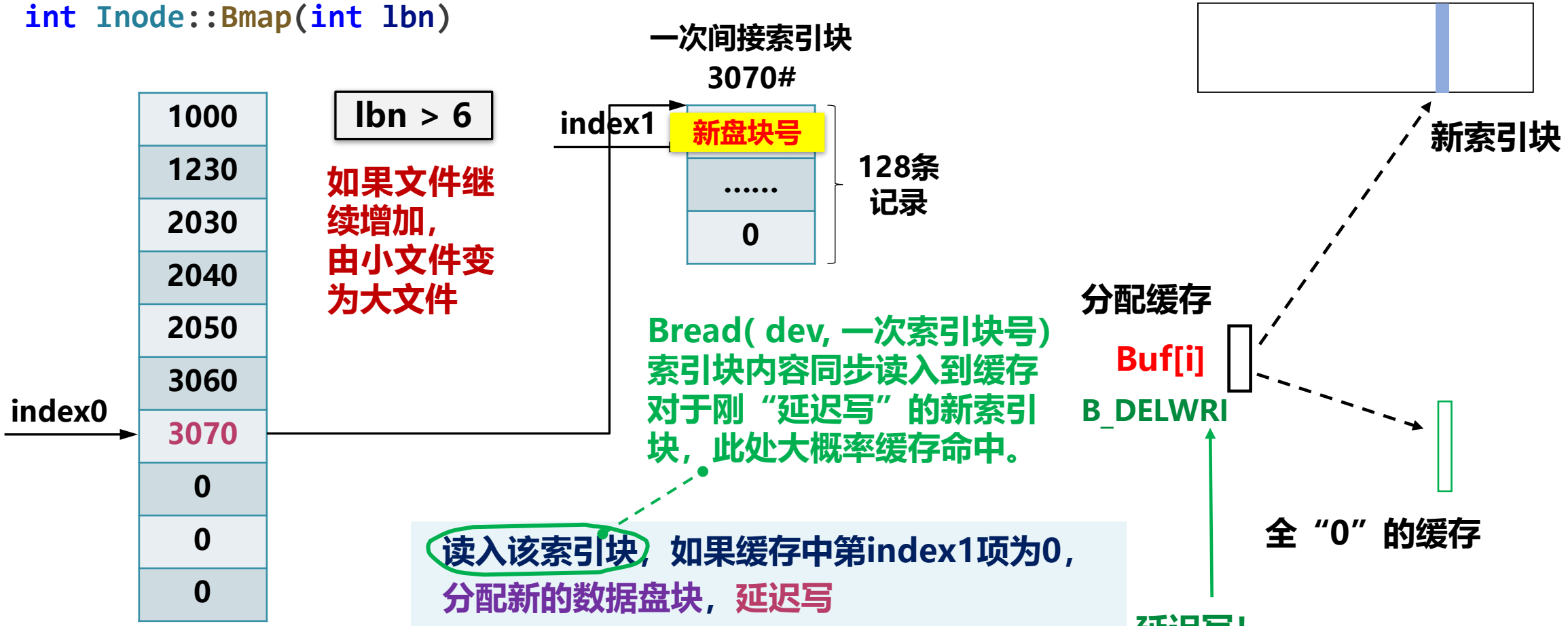


如果i_addr[index0] = 0，
分配新的索引盘块，延迟写
i_addr[index0] = 新索引盘块号

“垃圾” 清理的工作不必马上执行，
可适当延后，因为缓存可能马上被使用

UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```



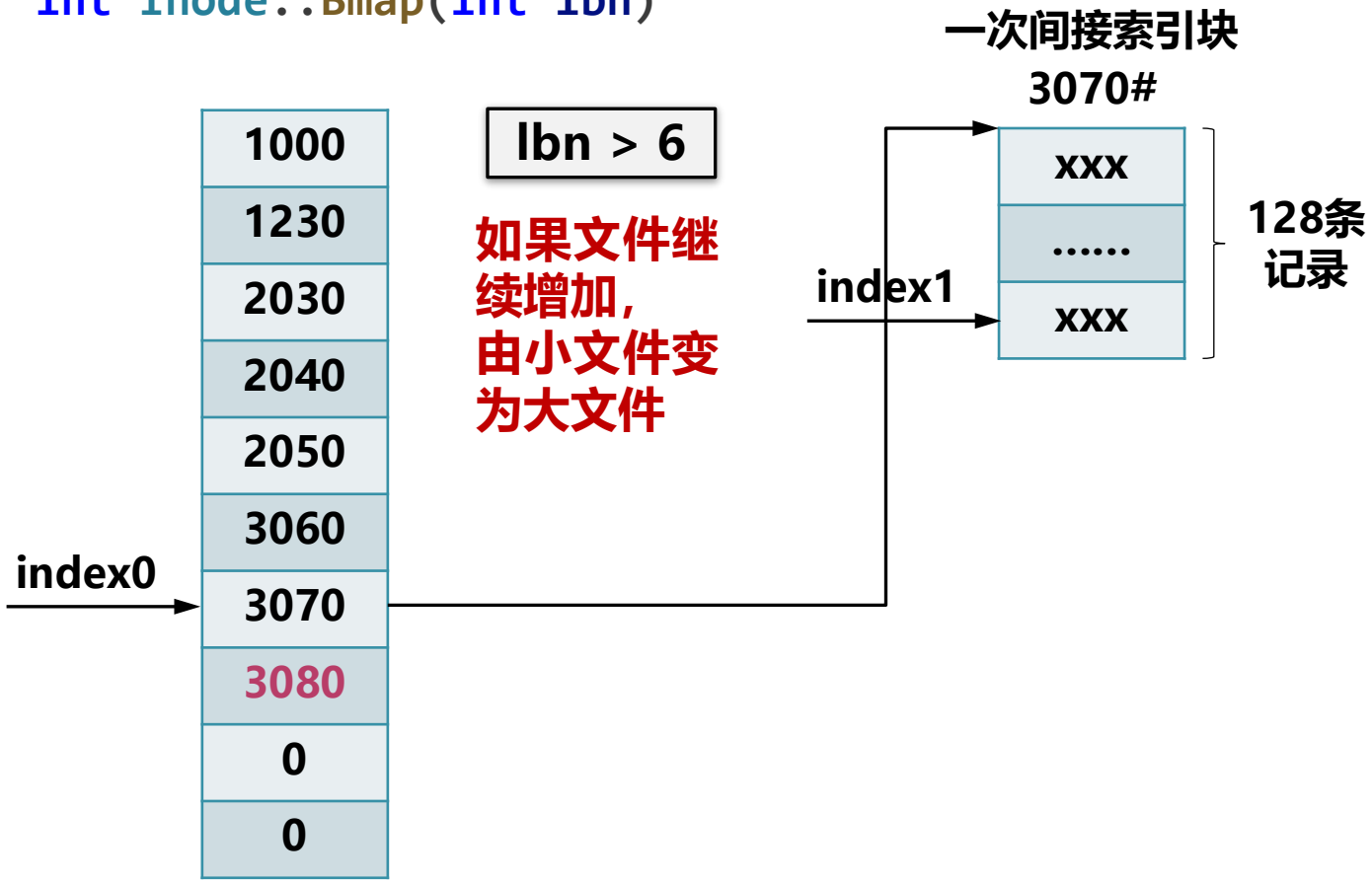
读入该索引块，如果缓存中第index1项为0，分配新的数据盘块，延迟写

缓存中第index1项 = 新数据盘块号

延迟写

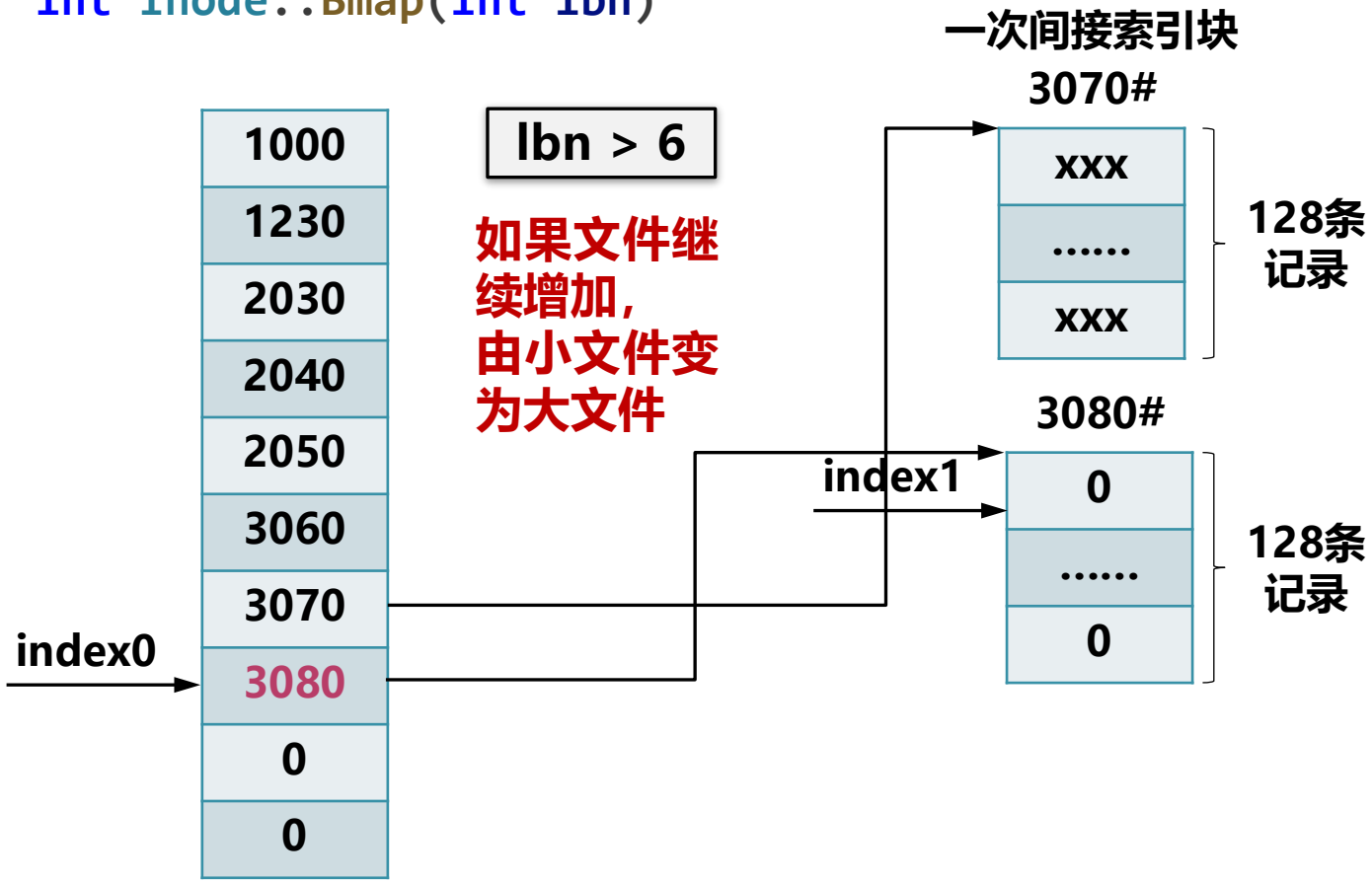
UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```



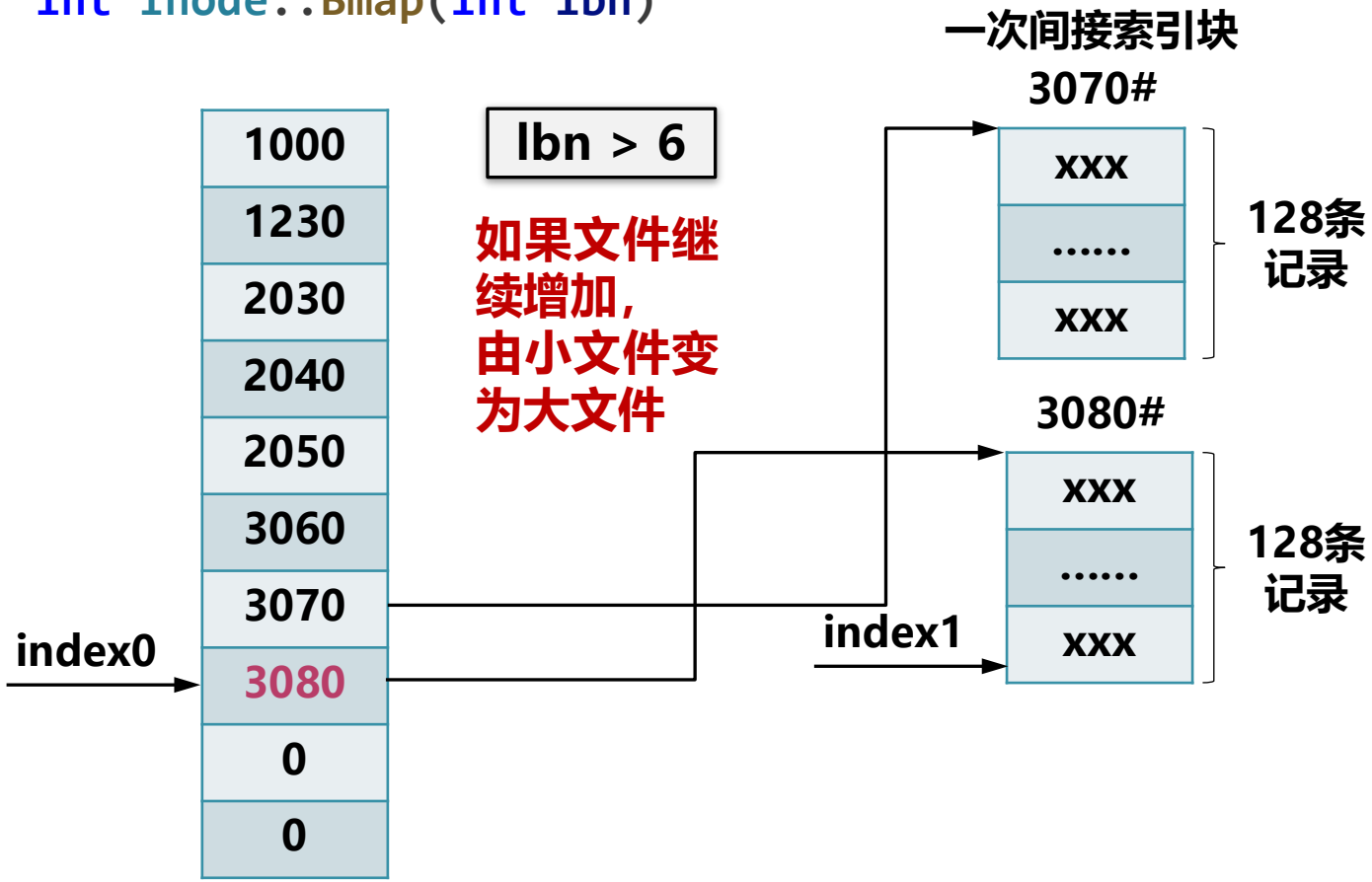
UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```



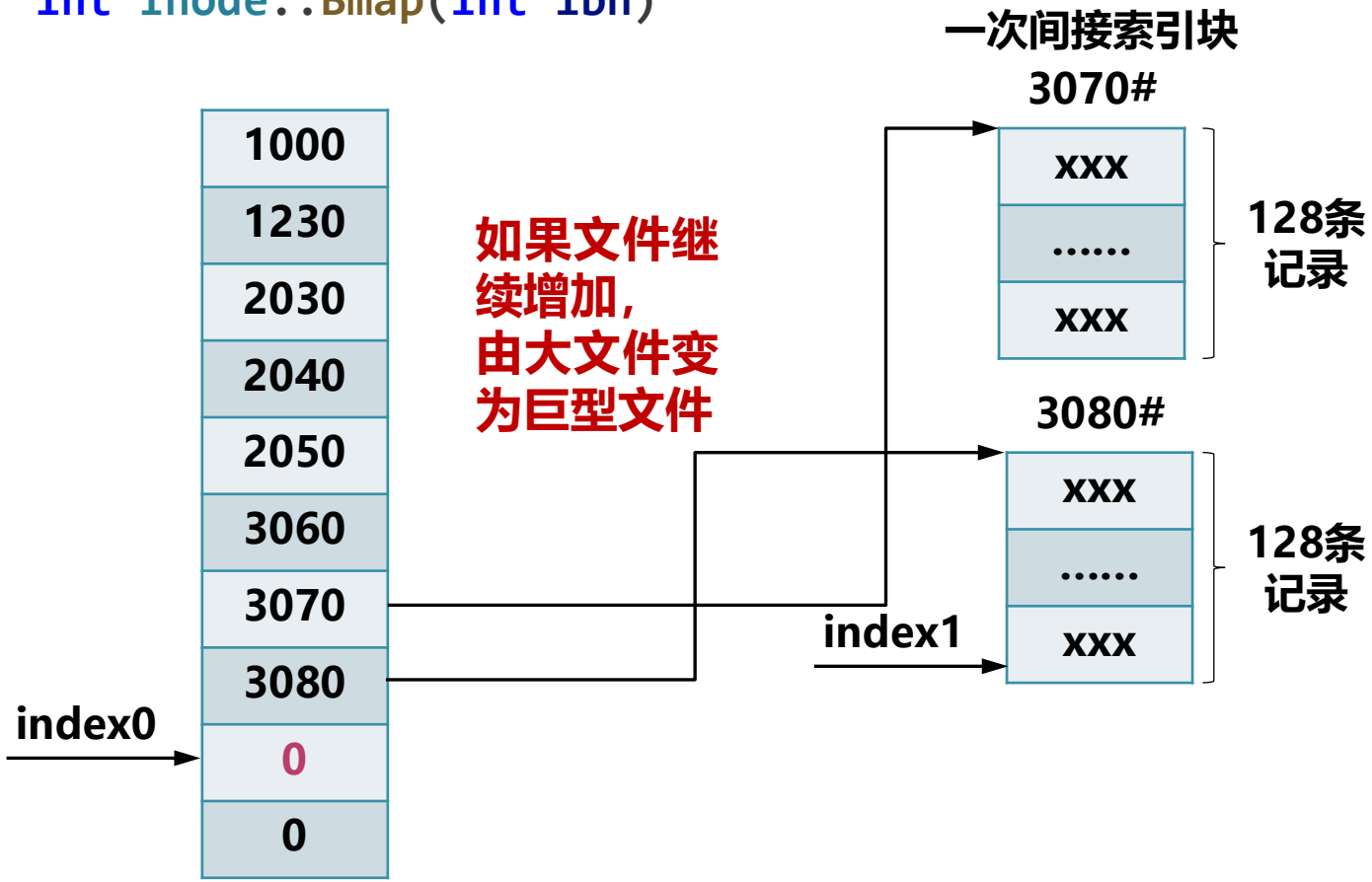
UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```



UNIX文件系统的静态结构

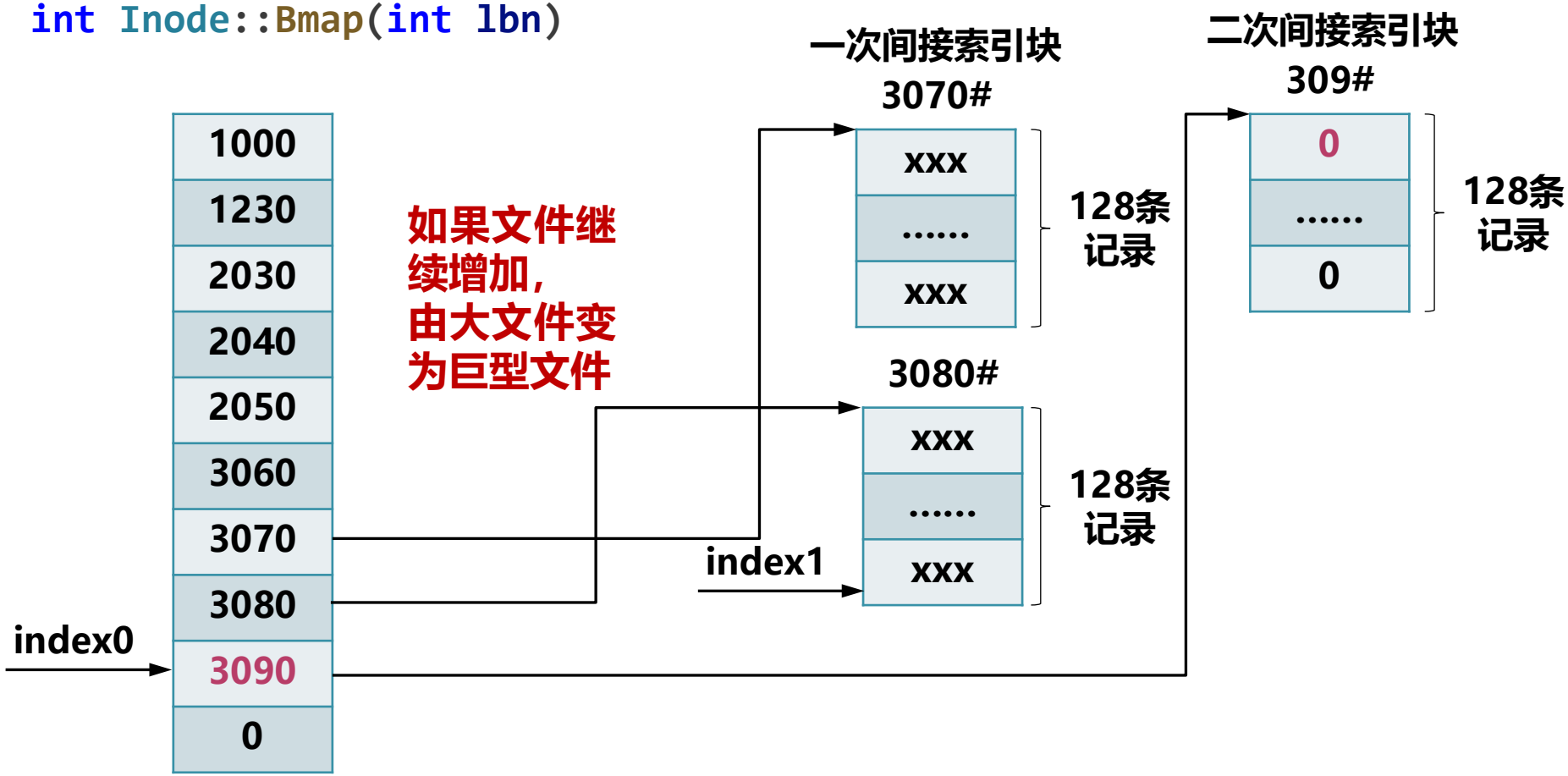

```
int Inode::Bmap(int lbn)
```



文件继续增长，需要新的二级索引块

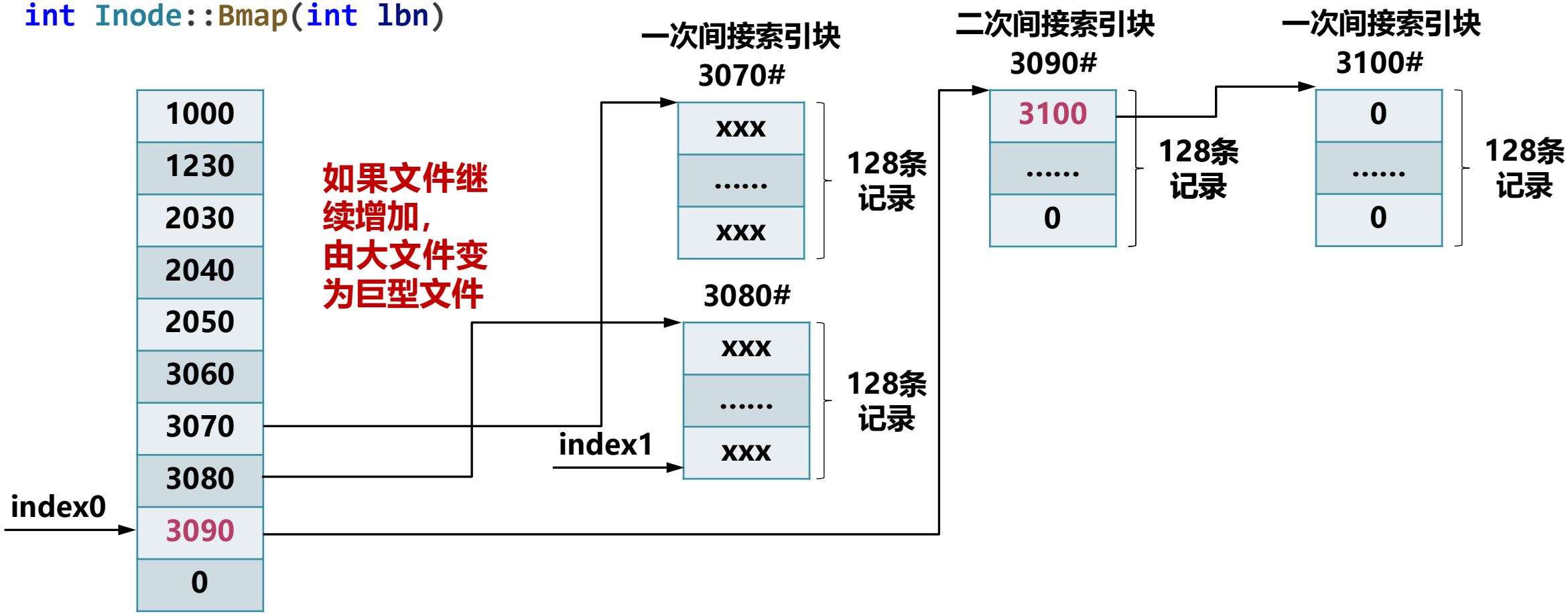
UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```



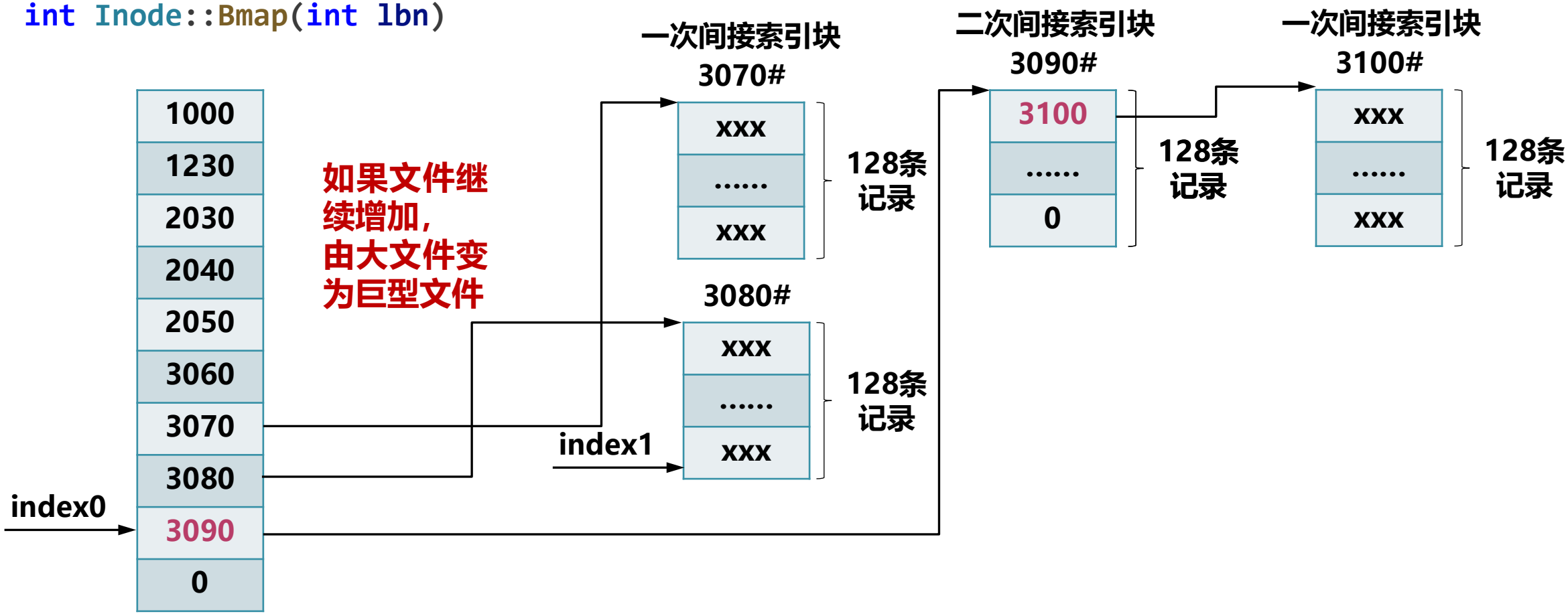
UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```



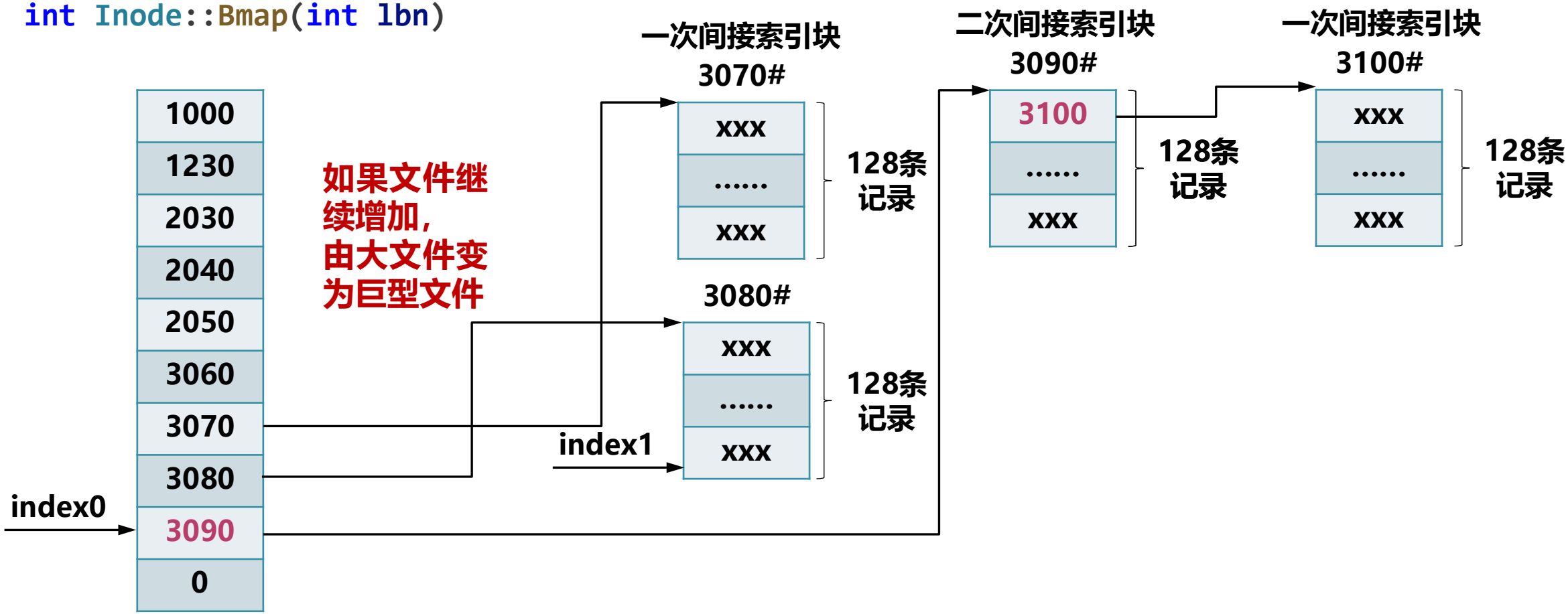
UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```



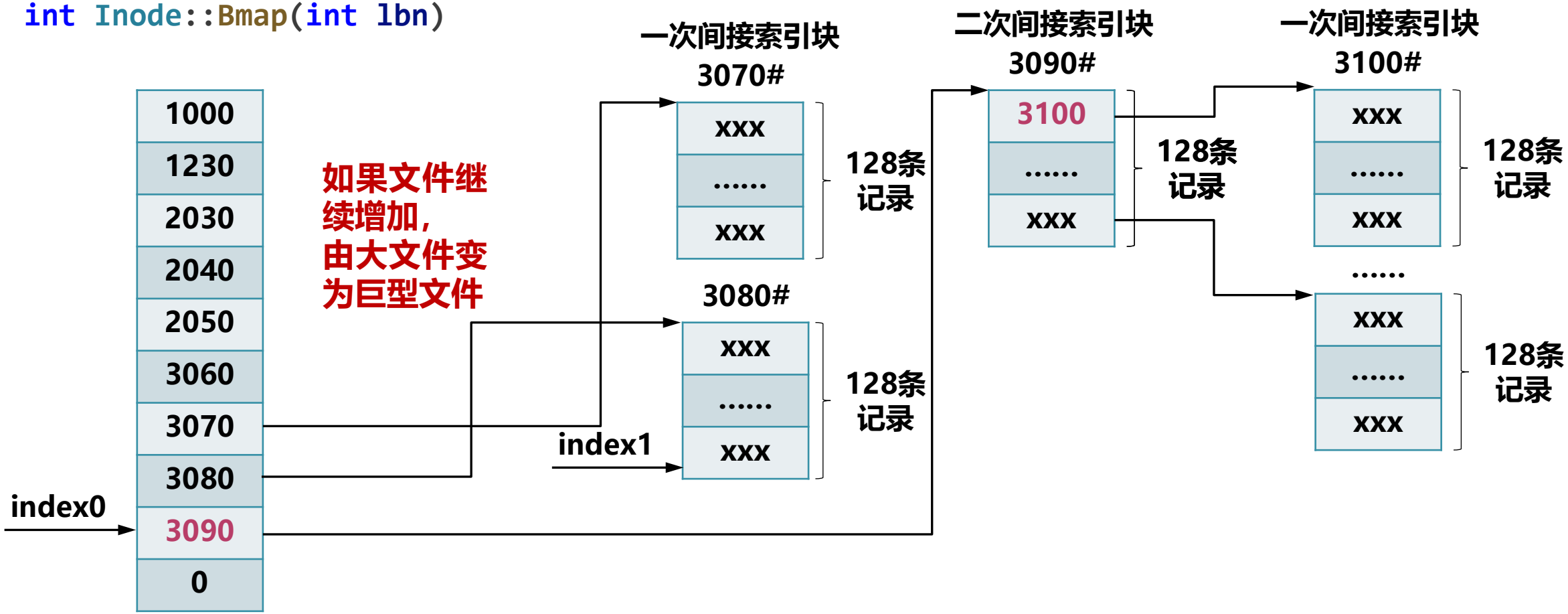
UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```



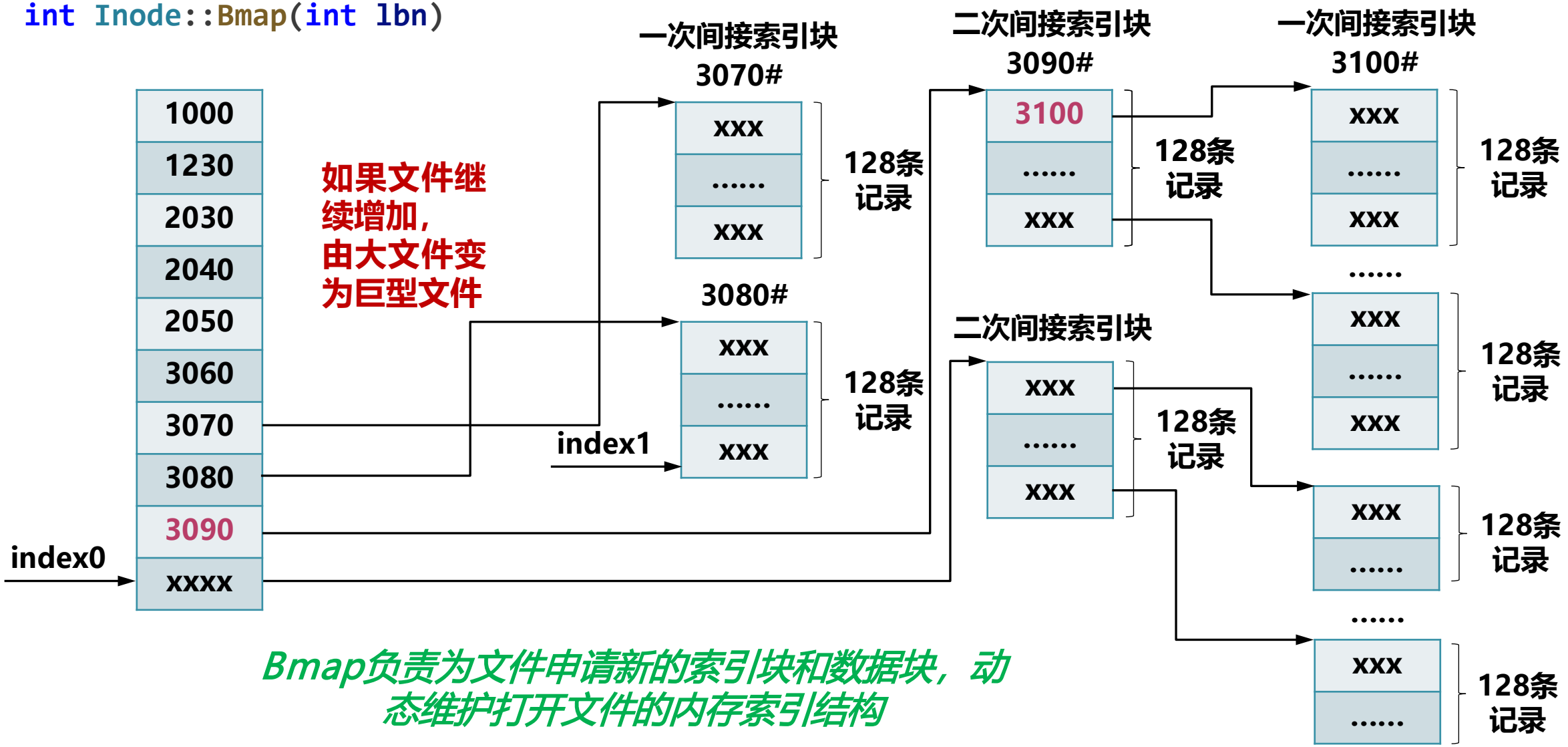
UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```



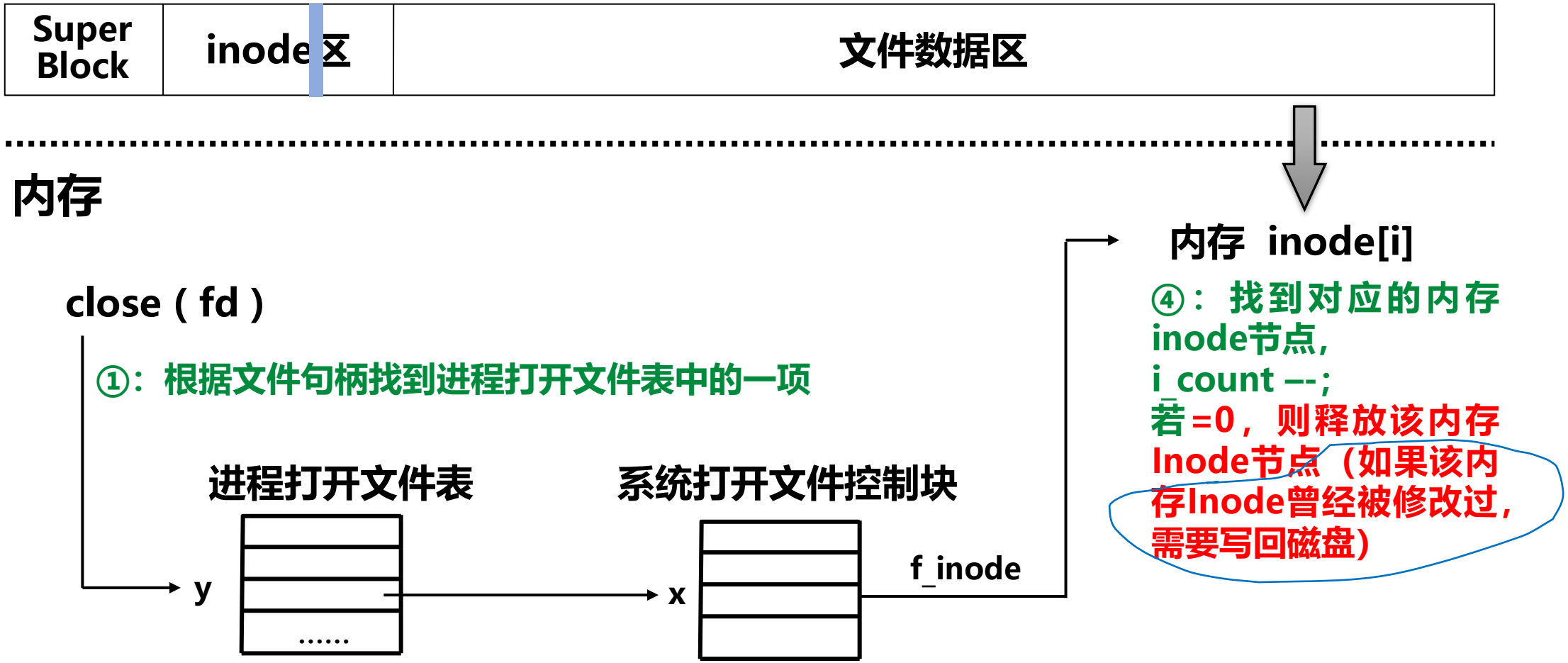
UNIX文件系统的静态结构


```
int Inode::Bmap(int lbn)
```

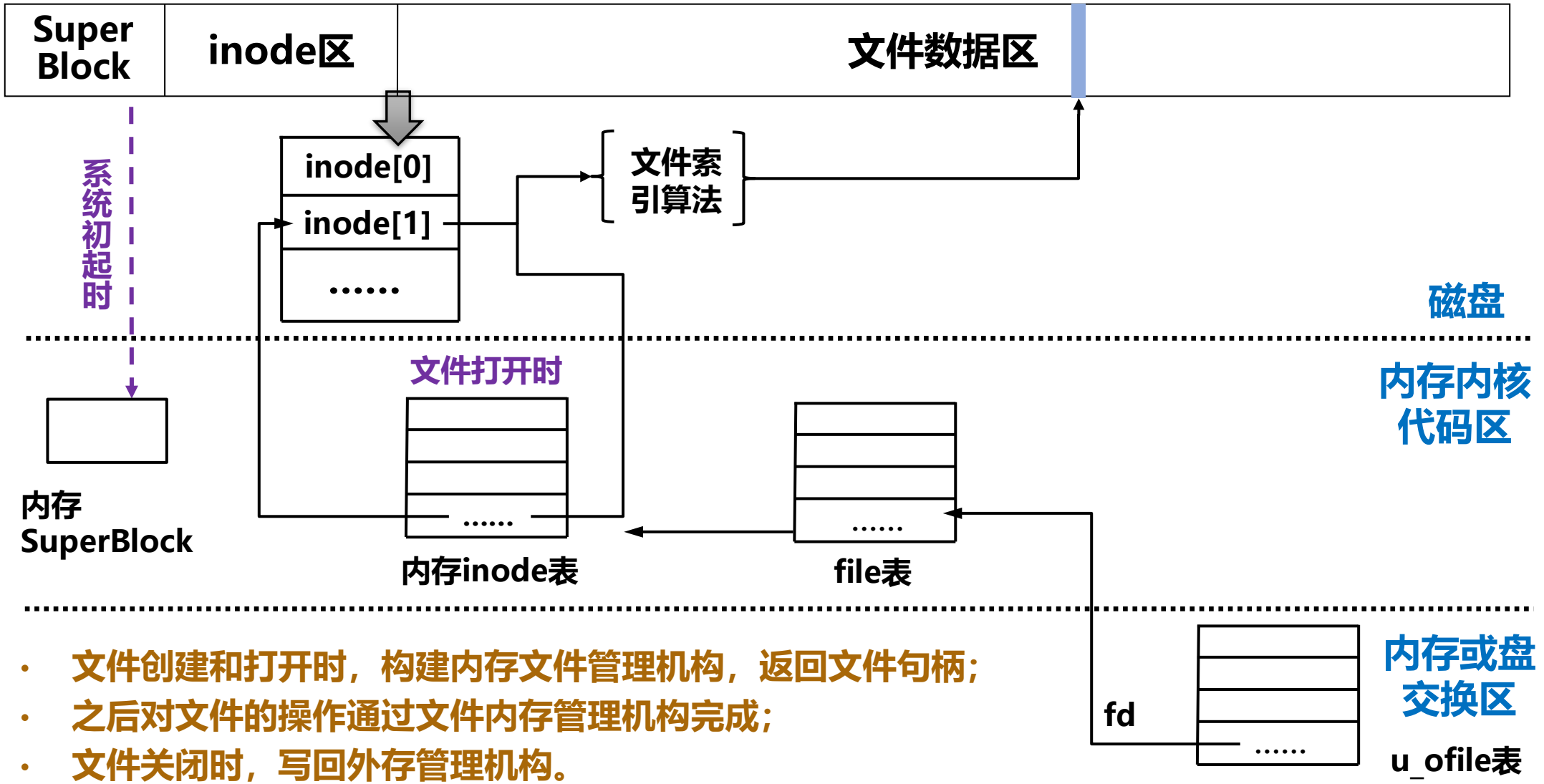


Bmap负责为文件申请新的索引块和数据块，动态维护打开文件的内存索引结构

UNIX文件的打开结构



UNIX文件系统相关的数据结构



- 文件创建和打开时，构建内存文件管理机构，返回文件句柄；
- 之后对文件的操作通过文件内存管理机构完成；
- 文件关闭时，写回外存管理机构。

UNIX文件索引节点

外存文件控制块区
(Inode区，202~1023#盘块)

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?

例题

Unix V6++系统，磁盘数据块 512 字节。文件的d_addr 数组管理着6个直接块，2个一次索引块和2个二次索引块。

现运行进程 PA 成功打开磁盘文件 example。已知该文件长 400K字节。请问：

（1）访问偏移量是100的字节需要读入__1__个磁盘块；

直接索引，无需读入索引块，直接读数据块

（2）访问偏移量是10K的字节需要读入__2__个磁盘块；

一次间接索引，需读入1个索引块，再读入一个数据块

（3）访问偏移量是132K的字节需要读入__3__个磁盘块；随后，马上访问偏移量是 132K+1的字节需要读入 __1__个磁盘块。

二次间接索引，需读入2个索引块，再读入一个数据块

访问偏移量是 132K+1字节索引块缓存命中

注意，本题读入指的是磁盘 IO，所有文件数据缓存不命中。

例题

假如UNIX V6++系统刚启动完成，系统中只有用户进程pa执行如下代码。
文件Jerry大小为5200字节。

```
main()
{
    int count = 0;
    int fd = open(“/usr/ast/Jerry”, O_RDWR); //以读写方式打开文件
    char data[1000];
    seek(fd, 4500, 0);
    count = read(fd, data, 1000);
    write(fd, data, count);
    ……;
}
```

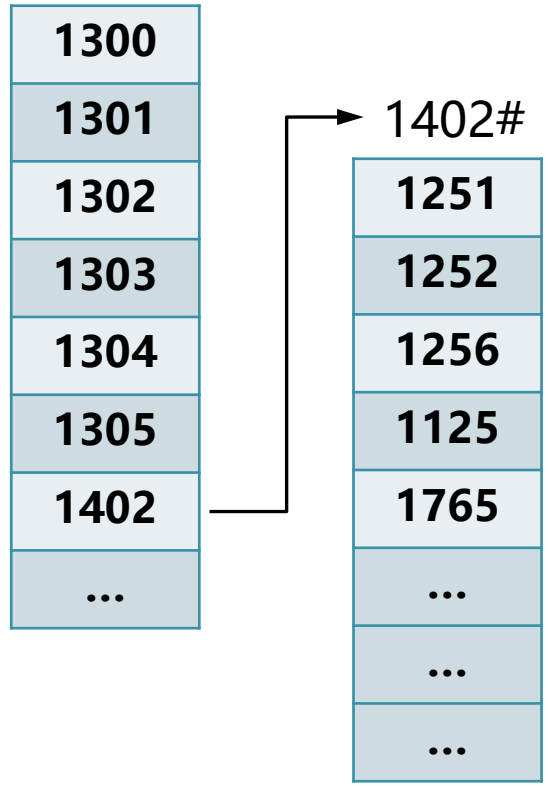
例题

假如UNIX V6++系统刚启动完成，系统中只有用户进程pa执行如下代码。文件Jerry大小为5200字节。

```
main()
{
    int count = 0;
    int fd = open(“/usr/ast/Jerry”, O__RDWR); //以读写方式打开文件
    char data[1000];
    seek(fd, 4500, 0);
    count = read(fd, data, 1000);
    write(fd, data, count);
    ……;
}
```

请回答下列问题：

（1）绘制文件的索引结构



5200 / 512 = 10

索引共需11个数据块，1个索引块

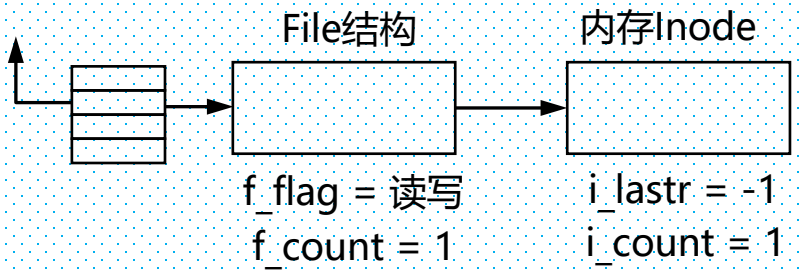
例题

假如UNIX V6++系统刚启动完成，系统中只有用户进程pa执行如下代码。
文件Jerry大小为5200字节。

```
main()
{
    int count = 0;
    int fd = open(“/usr/ast/Jerry”, O_RDWR); //以读写方式打开文件
    char data[1000];
    seek(fd, 4500, 0);
    count = read(fd, data, 1000);
    write(fd, data, count);
    ……;
}
```

请回答下列问题：
（2）简述open系统调用的过程

1. 目录检索/usr/ast/Jerry，获得Jessy文件的磁盘Inode号；
2. 确认该Inode的内存副本是否存在（否）
3. 申请一个空白的内存Inode，将磁盘Inode读入（计算该Inode所在盘块号，同步读入缓存），并复制；
4. 申请内存File结构；
5. 申请u_ofiles数组中的空白项；
6. 返回文件句柄。



例题

假如UNIX V6++系统刚启动完成，系统中只有用户进程pa执行如下代码。
文件Jerry大小为5200字节。

```
main()
{
    int count = 0;
    int fd = open(“/usr/ast/Jerry”, O_RDWR); //以读写方式打开文件
    char data[1000];
    seek(fd, 4500, 0);
    count = read(fd, data, 1000);
    write(fd, data, count);
    ……;
}
```

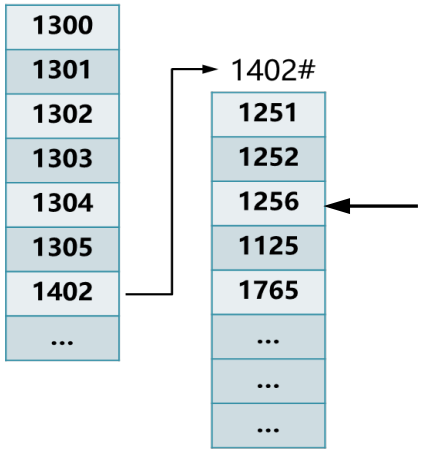
请回答下列问题：

（3）read 操作对文件实施的是顺序读写还是随机读写？ 随机读写 。为什么？因为使用seek修改了文件读写指针的位置 。read 操作结束后的返回值为 700 ， 此时文件读写指针所在的位置为： 5200 。

例题

假如UNIX V6++系统刚启动完成，系统中只有用户进程pa执行如下代码。文件Jerry大小为5200字节。

```
main()
{
    int count = 0;
    int fd = open(“/usr/ast/Jerry”, O_RDWR); //以读写方式打开文件
    char data[1000];
    seek(fd, 4500, 0);
    count = read(fd, data, 1000);
    write(fd, data, count);
    ……;
}
```

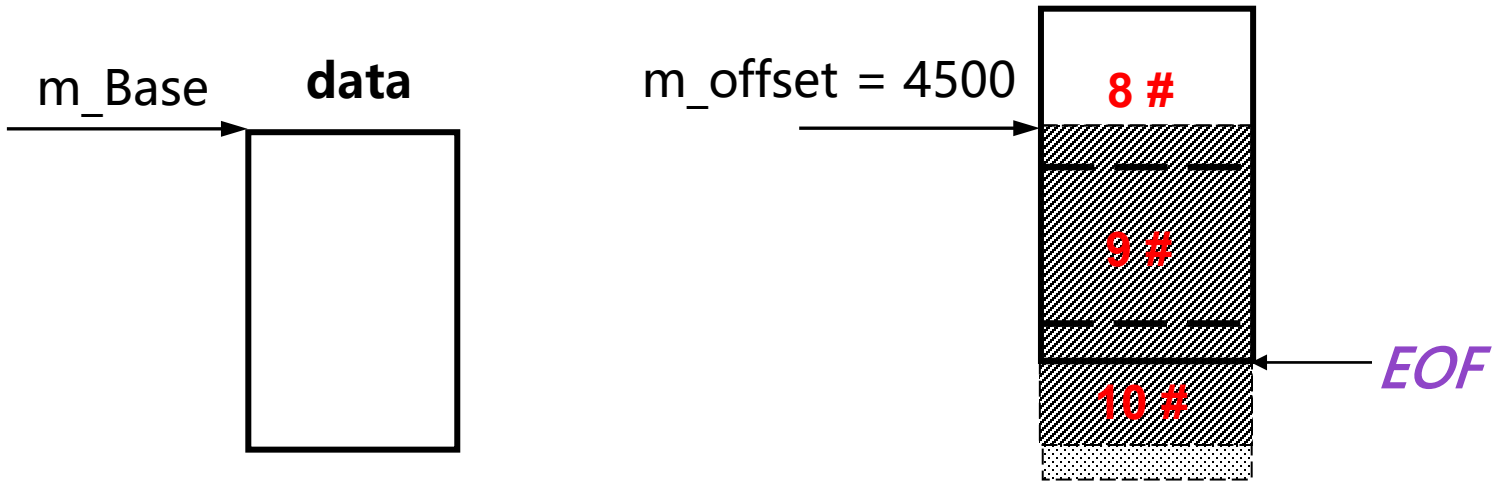


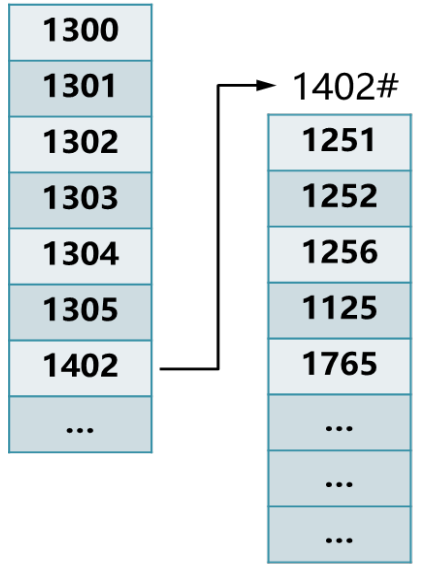
请回答下列问题：

（4）简述read 操作的读入过程

例题

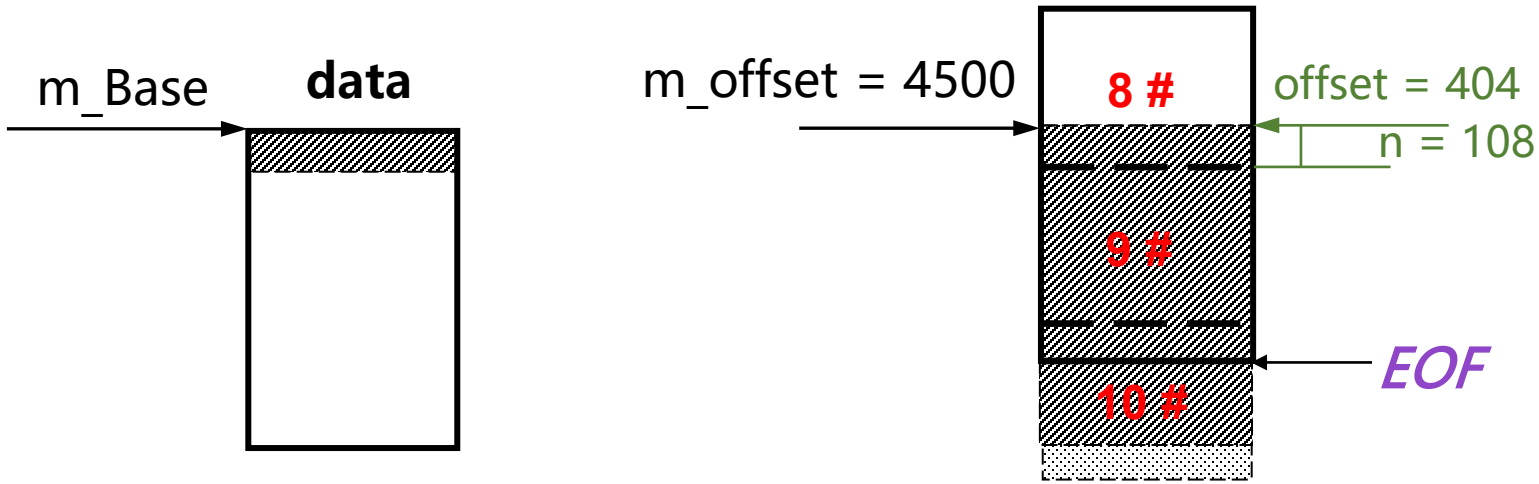
初始状态：m_count = 1000；m_offset = 4500；m_Base = data[0];

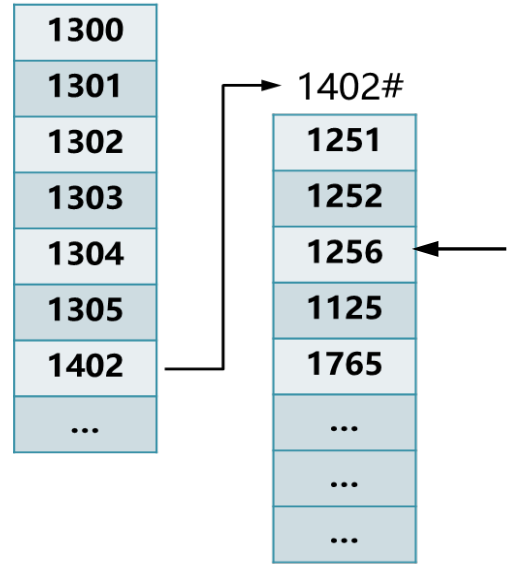




例题

初始状态：m_count = 1000；m_offset = 4500；m_Base = data[0];



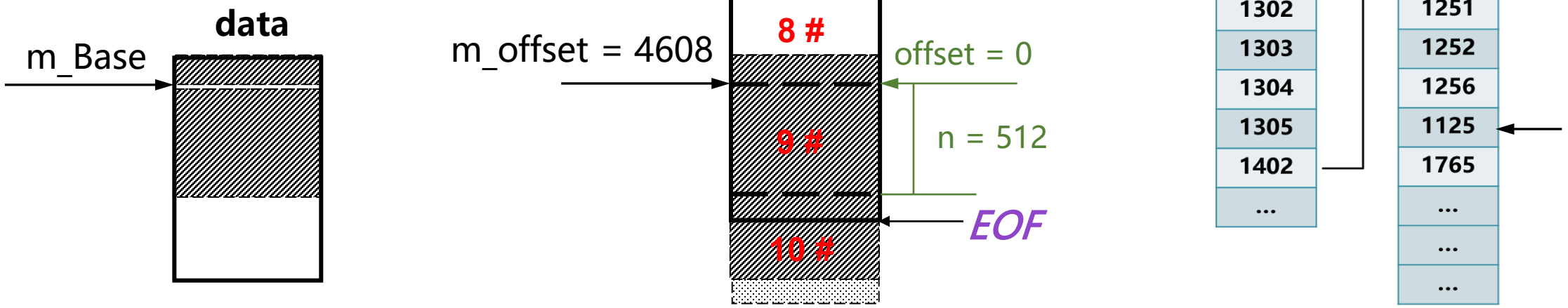


读第一块：

1. lbn = 4500 / 512 = 8，offset = 4500 % 512 = 404，n = min (512 - 404, 1000, 5200 - 4500)= 108；
2. 调用Bmap，同步读入1402#盘块（缓存不命中），得到物理块号1256，释放缓存；
3. 同步读入1256#盘块到缓存（缓存不命中），从该缓存的第404个字节开始，复制108个字节到data数组的前108个字节；释放缓存；
4. m_count = 1000-108 = 892，m_offset = 4500 + 108 = 4608，m_Base = data[108];

例题

m_count = 892；m_offset = 4608；m_Base = data[108];

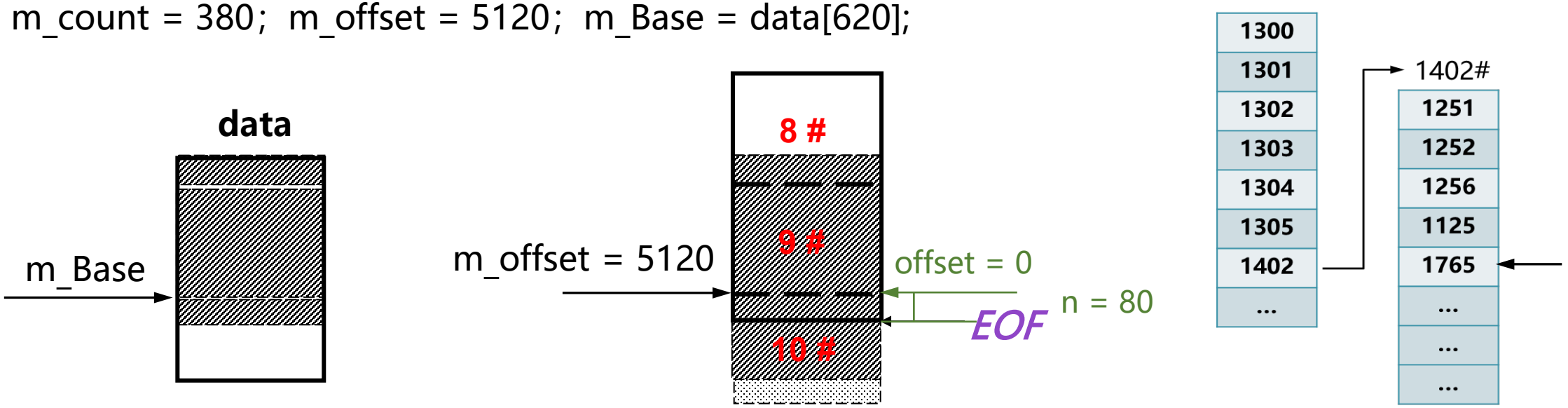


读第二块:

1. lbn = 4608 / 512 = 9，offset = 4608 % 512 = 0， n = min (512 - 0, 1000, 5200 - 4608)= 512;
2. 调用Bmap，同步读入1402#盘块（缓存命中），返回物理块号1125，释放缓存;
3. 同步读入1125#盘块到缓存（缓存不命中）， 从该缓存的第0个字节开始，复制512个字节到data数组的108 ~ 619字节;
   ① 预读：异步读1765块（Bmap中已经将1765写入rablock）；
4. m_count = 892 -512 = 380，m_offset = 4608 + 512 = 5120， m_Base = data[620]

例题

m_count = 380；m_offset = 5120；m_Base = data[620];



读第三块:

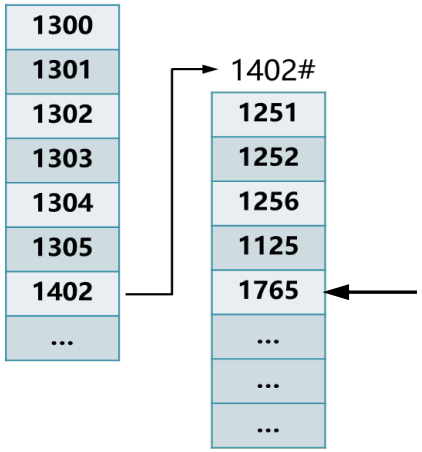
1. lbn = 5120 / 512 = 10，offset = 5120 % 512 = 0，n = min (512 - 0, 380, 5200-5120)= 80;
2. 调用Bmap，同步读入1402#盘块（缓存命中），返回物理块号1765，释放缓存;
3. 同步读入1765#盘块到缓存（因为预读，缓存命中，如果预读尚未完成，睡在Getblk中）， 从该缓存的第0个字节开始，复制80个字节到data数组的620 ~ 699字节；释放缓存
4. m_count = 380 - 80 = 300，m_offset = 5120 + 80 = 5200， m_Base = data[700]

返回1000 – 300 = 700， 修改f_offset = m_offset = 5200

例题

假如UNIX V6++系统刚启动完成，系统中只有用户进程pa执行如下代码。文件Jerry大小为5200字节。

```
main()
{
    int count = 0;
    int fd = open(“/usr/ast/Jerry”, O_RDWR); //以读写方式打开文件
    char data[1000];
    seek(fd, 4500, 0);
    count = read(fd, data, 1000);
    write(fd, data, count);
    ……;
}
```

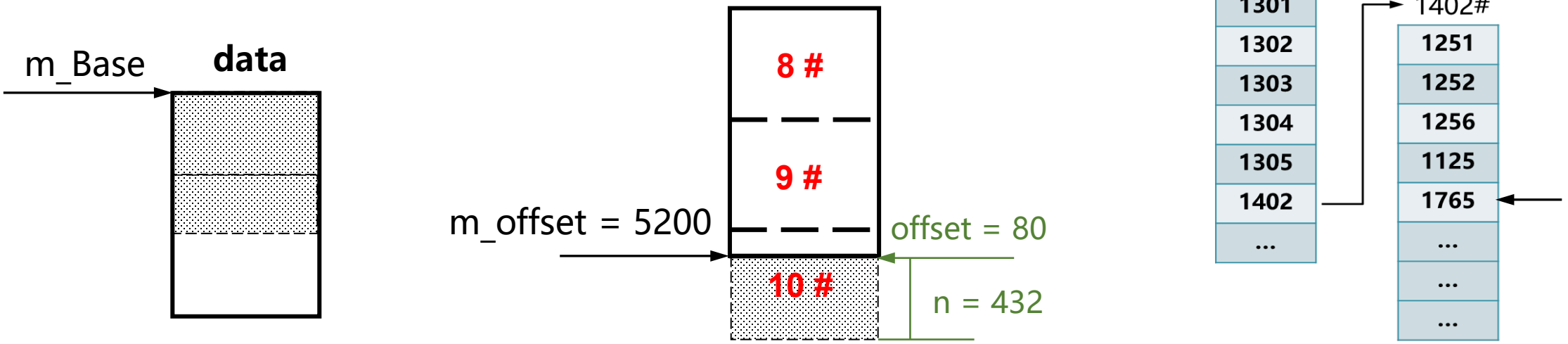


请回答下列问题：

（5）简述write操作的写入过程

例题

初始状态：m_count = 700；m_offset = 5200；m_Base = data[0];

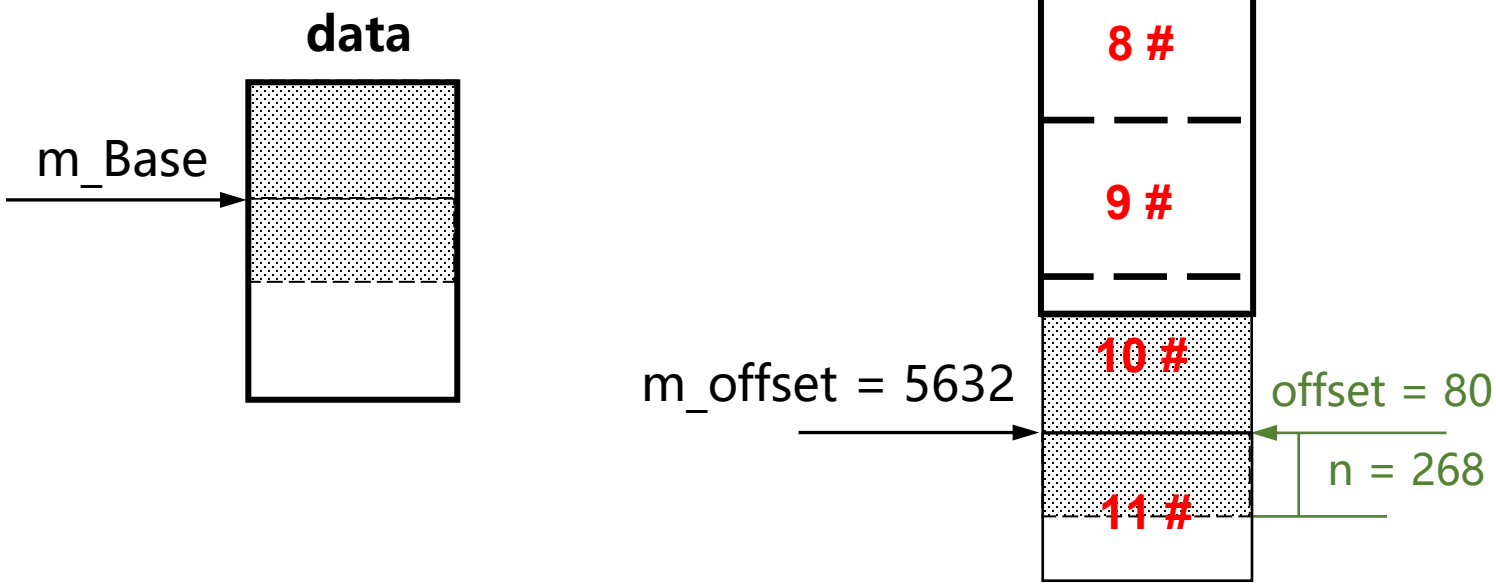


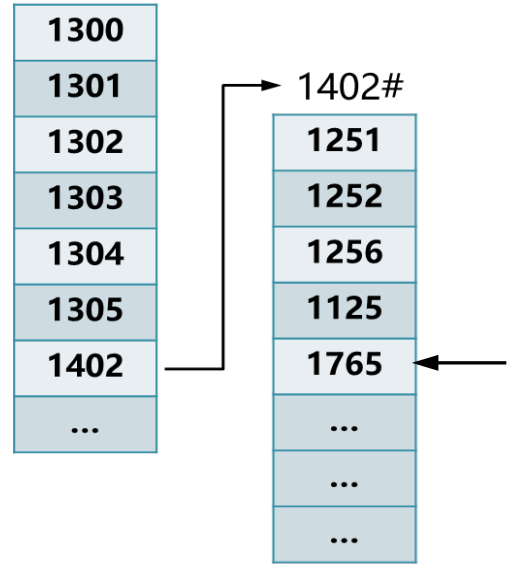
写第一块：

1. lbn = 5200 / 512 = 10，offset = 5200 % 512 = 80，n = min (512 - 80, 700)= 432；
2. 调用Bmap，同步读入1402#盘块（缓存命中），返回物理块号1765，释放缓存；
3. 因为432<512，同步读入1765#盘块到缓存（缓存命中），将data数组的前432个字节写入该缓存的后432个字节；缓存写满，异步写磁盘；
4. m_count = 700 - 432 = 268，m_offset = 5200 + 432 = 5632，m_Base = data[432]；

例题

m_count = 268；m_offset = 5632；m_Base = data[432];



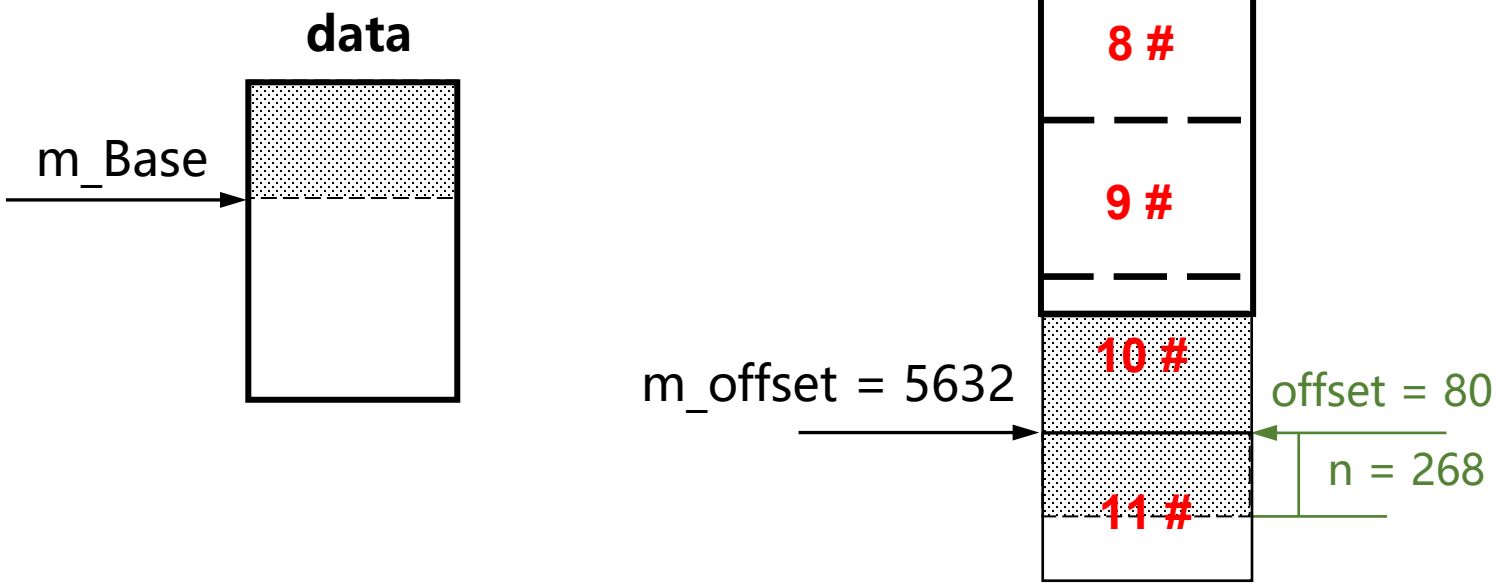


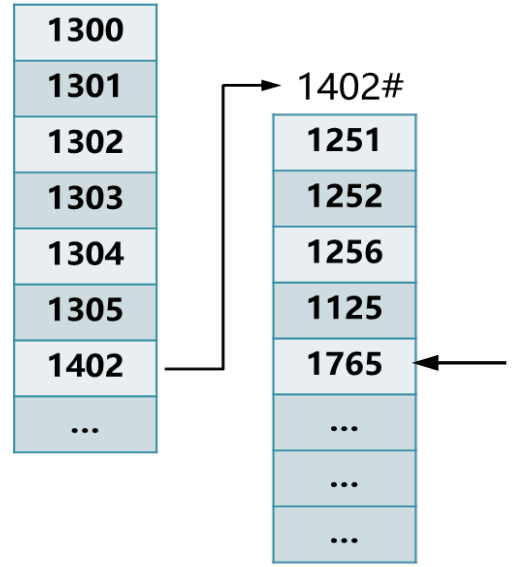
写第二块:

1. lbn = 5632 / 512 = 11，offset = 5632 % 512 = 0，n = min (512 - 0, 268)= 268；
2. 调用Bmap，同步读入1402#盘块（缓存命中），对应物理盘块号为0；
   ① 分配新数据盘块，为该盘块分配缓存，缓存清干净，延迟写，释放缓存；
   ② 将新分配数据盘块的盘块号登记在1402#盘块的缓存中，延迟写，释放缓存；

例题

m_count = 268；m_offset = 5632；m_Base = data[432];





写第二块:

3. 因为268<512，同步读新盘块到缓存（缓存命中），将data数组的后268个字节写入该缓存的前268个字节；
   缓存未写满，延迟写，释放缓存；
4. m_count = 268 - 268 = 0，m_offset = 5632 + 268 = 5900；

返回700 – 700 = 0， 修改f_offset = m_offset = 590
修改i_size = 5900

例题

假如UNIX V6++系统刚启动完成，系统中只有用户进程pa执行如下代码。
文件Jerry大小为5200字节。

```
main()
{
    int count = 0;
    int fd = open(“/usr/ast/Jerry”, O__RDWR); //以读写方式打开文件
    char data[1000];
    seek(fd, 4500, 0);
    count = read(fd, data, 1000);
    write(fd, data, count);
    ……;
}
```

请回答下列问题：

（6）write 操作结束后，该文件的内存索引节点是否会发生变化？ 会

如果会，变化的具体内容包括： i_addr 数组，i_size。

本节小结

1　了解文件系统磁盘空间管理的一般方法

2　掌握UNIX磁盘存储空间的成组链接管理

阅读教材：262页 ~ 278页

E19：文件管理（UNIX文件系统的读写操作）