第六章

文件管理

方　钰

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

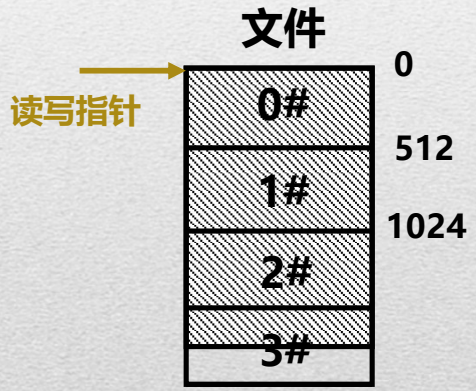
关于文件的读写操作

```
#include <stdio.h>
#include <sys.h>
#include <file.h>
int main1()
{
        char data1[12]="Hello World!";
        char data2[12];
        int fd=0;
        int count = 0;
        fd = open("/usr/Jessy",01);
        count = write(fd, data1, 12);
        count = read(fd, data2, 12);
        close(fd);
        return 1;
}
```

文件的读写操作需要注意的几点：

1. 读写操作之前，必须成功打开文件，建立文件的内存打开结构

2. 每次读写都从读写指针位置开始



打开之后，读写指针在文件起始位置

应用程序（用户态运行）

以读文件为例

```
n = read ( fd, buf, nbytes);
```

```
int read(int fd, char* buf, int nbytes)
{
    int res;
    __asm__ __volatile__ ("int $0x80":"=a"(res):"a"(3),"b"(fd),"c"(buf) ,"d"(nbytes));
    if ( res >= 0 )
        return res;
    return -1;
}
```

4. 执行INT 0x80指令

2. 三个参数分别存入 EBX、ECX、EDX

3. EAX寄存器将带回返回值给res

1. read的系统调用号3存入EAX

```
int SystemCall::Sys_Read()
{
    FileManager& fileMgr = Kernel::Instance().GetFileManager();
    fileMgr.Read();
    return 0;/* GCC likes it ! */
}
```

```
void FileManager::Read()
{
    /* 直接调用Rdwr()函数即可 */
    this->Rdwr(File::FREAD);
}
```

应用程序（用户态运行）

以写文件为例

n = write (fd, buf, nbytes);

```
int write(int fd, char* buf, int nbytes)
{
    int res;
    __asm__ __volatile__ ("int $0x80":"=a"(res):"a"(4),"b"(fd),"c"(buf) ,"d"(nbytes));
    if ( res >= 0 )
        return res;
    return -1;
}
```

4. 执行INT 0x80指令

2. 三个参数分别存入 EBX、ECX、EDX

3. EAX寄存器将带回返回值给res

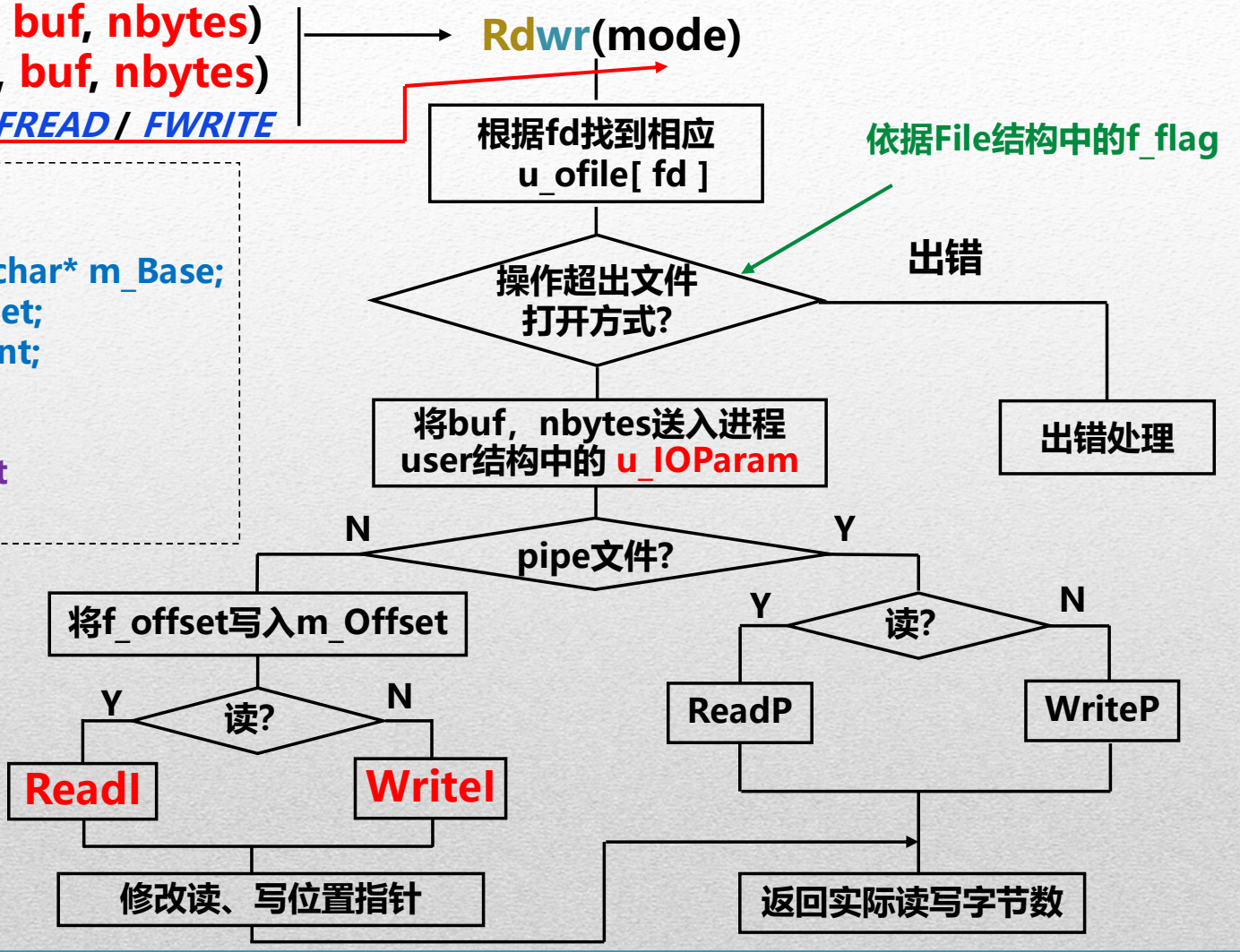
1. write的系统调用号4存入EAX

```
int SystemCall::Sys_Write()
{
    FileManager& fileMgr = Kernel::Instance().GetFileManager();
    fileMgr.Write();
    return 0;/* GCC likes it ! */
}
```

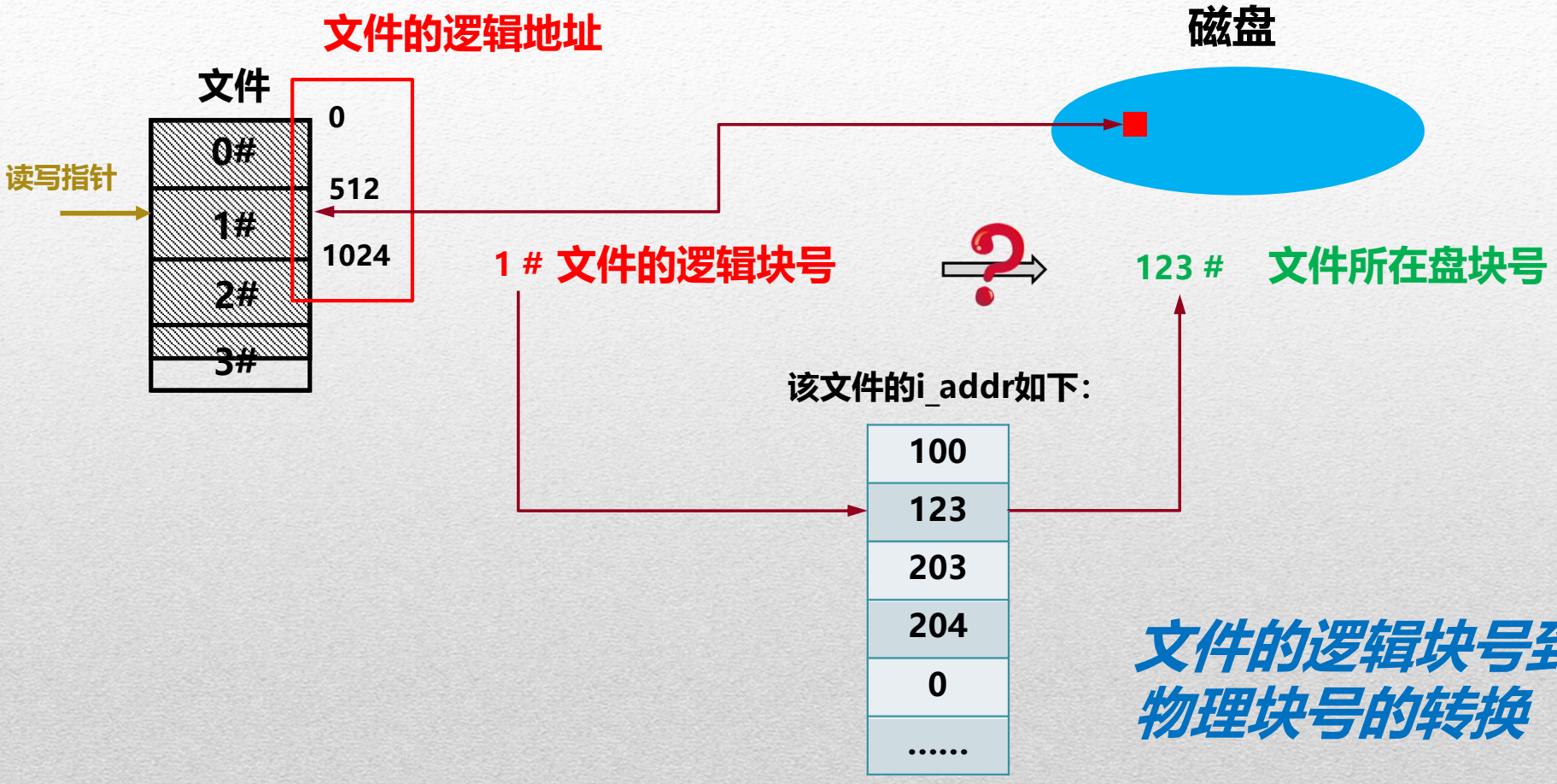
```
void FileManager::Write()
{
    /* 直接调用Rdwr()函数即可 */
    this->Rdwr(File::FWRITE);
}
```

read (fd, buf, nbytes)
write (fd, buf, nbytes)
mode = *FREAD* / *FWRITE*

```
class IOParameter
{
public:   unsigned char* m_Base;
          int m_Offset;
          int m_Count;
};
m_Base ← buf
m_Offset ← f_offset
m_Count ← nbytes
```

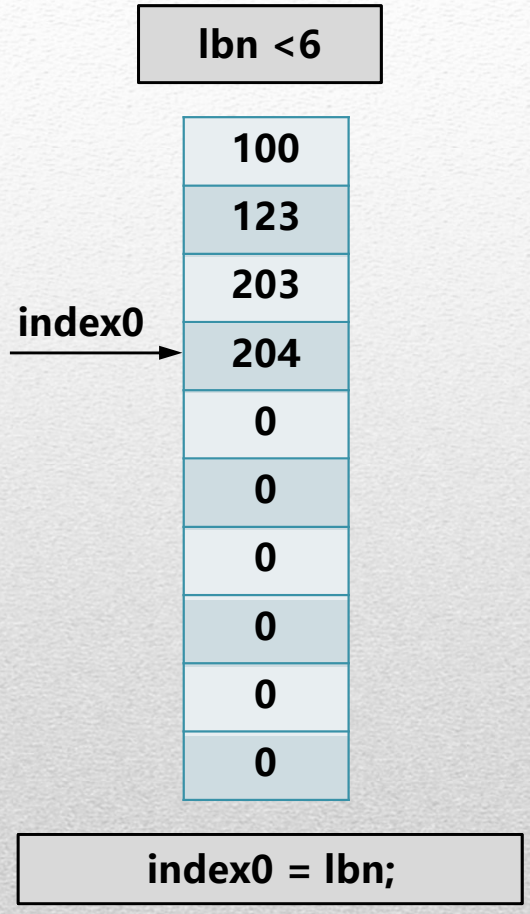


关于文件的读写操作



文件的逻辑块号到物理块号的转换

将文件逻辑块转换成物理块int Inode::Bmap(int lbn)



将文件逻辑块转换成物理块int Inode::Bmap(int lbn)

如果i_addr[index0] = 0，分配新盘块（数据盘块）

i_addr[index0] = 新盘块号

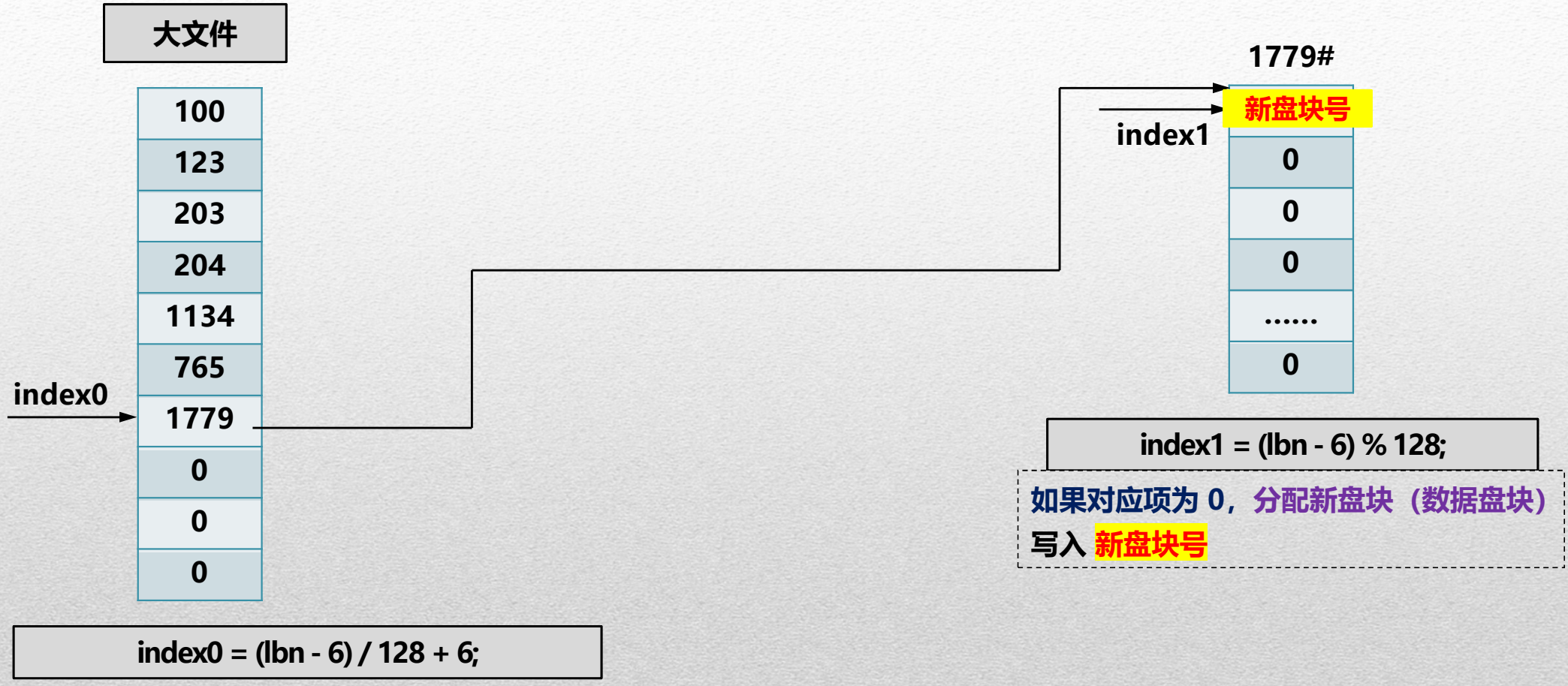
将文件逻辑块转换成物理块int Inode::Bmap(int lbn)

index0 = (lbn - 6) / 128 + 6;

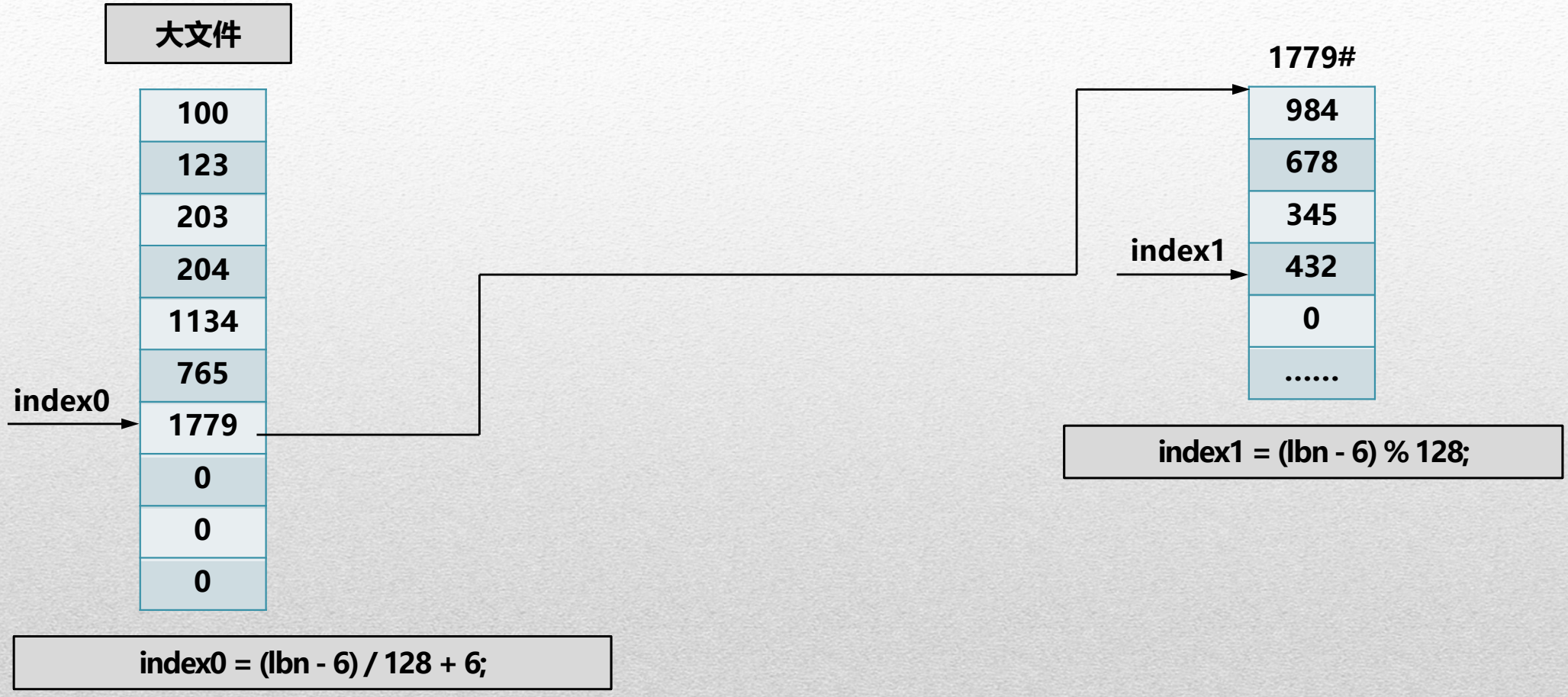
如果i_addr[index0] = 0，分配新盘块（索引盘块）

i_addr[index0] = 新盘块号

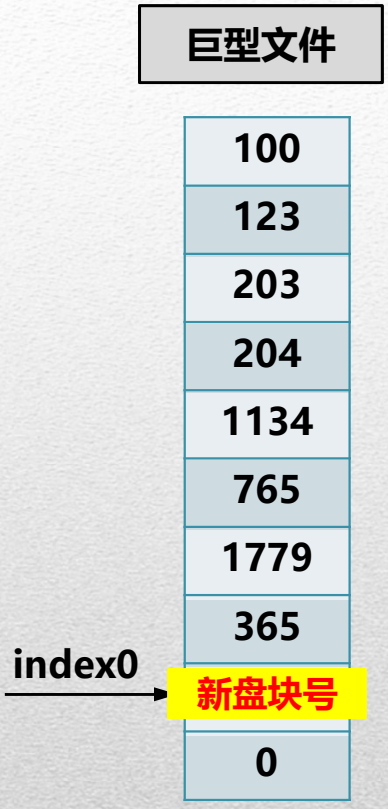
将文件逻辑块转换成物理块int Inode::Bmap(int lbn)



将文件逻辑块转换成物理块int Inode::Bmap(int lbn)



将文件逻辑块转换成物理块int Inode::Bmap(int lbn)

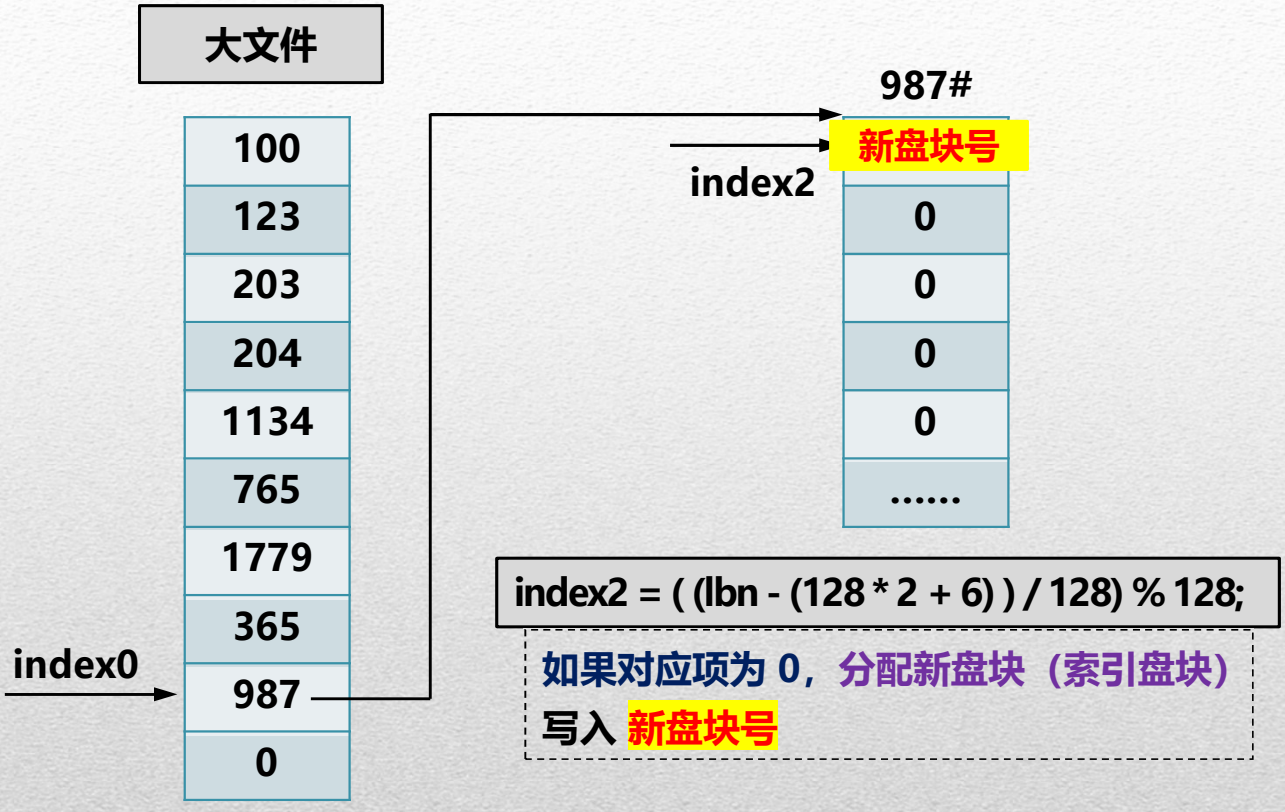


index0 = (lbn – 6- 128 * 2) / (128 * 128) + 8;

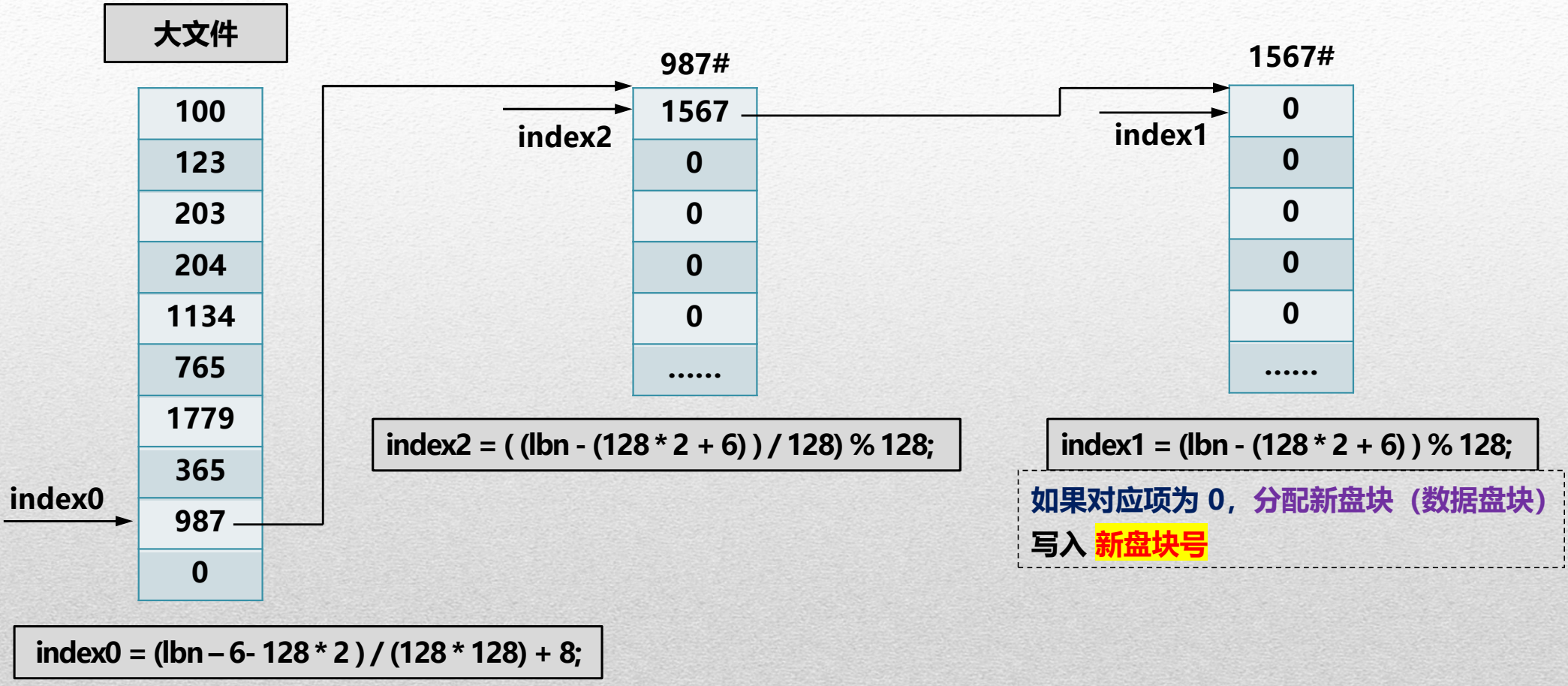
如果i_addr[index0] = 0，分配新盘块（索引盘块）

i_addr[index0] = 新盘块号

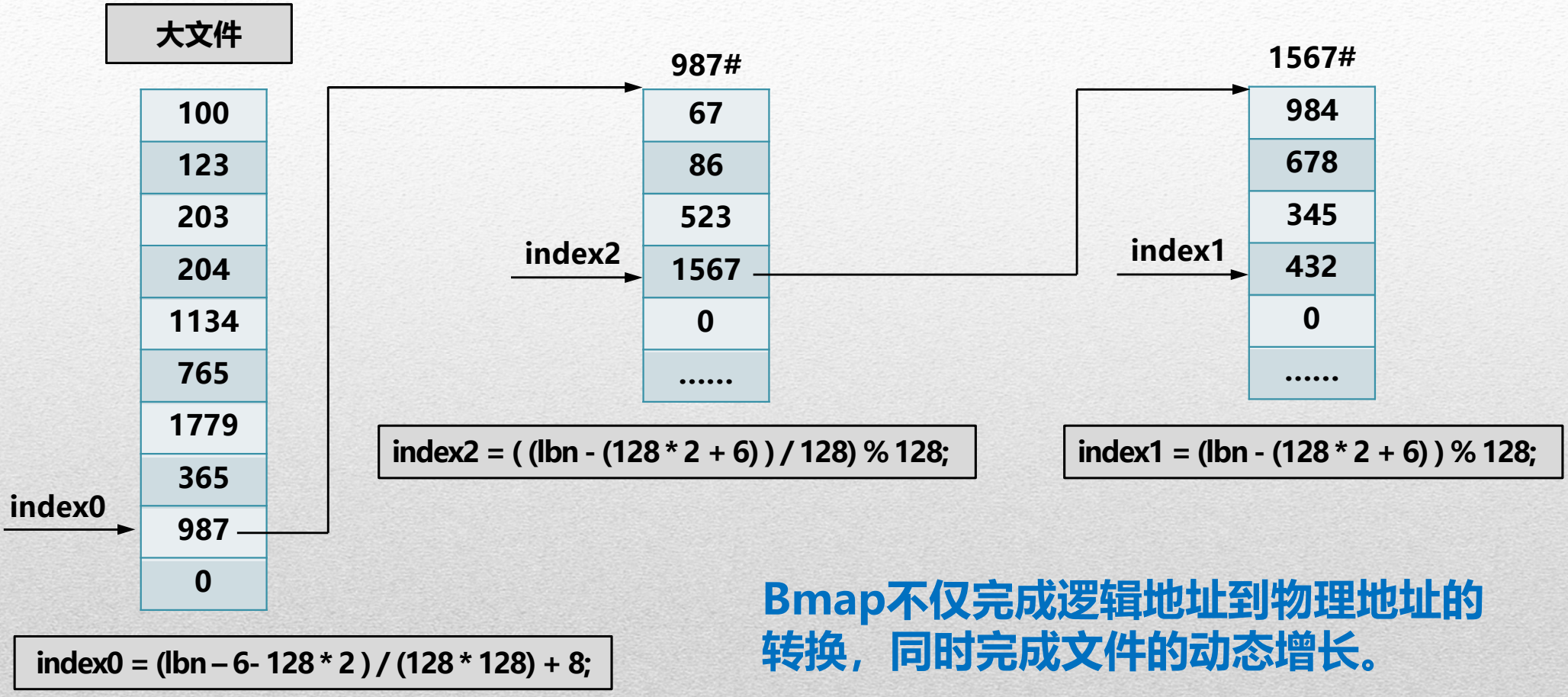
将文件逻辑块转换成物理块int Inode::Bmap(int lbn)



将文件逻辑块转换成物理块int Inode::Bmap(int lbn)



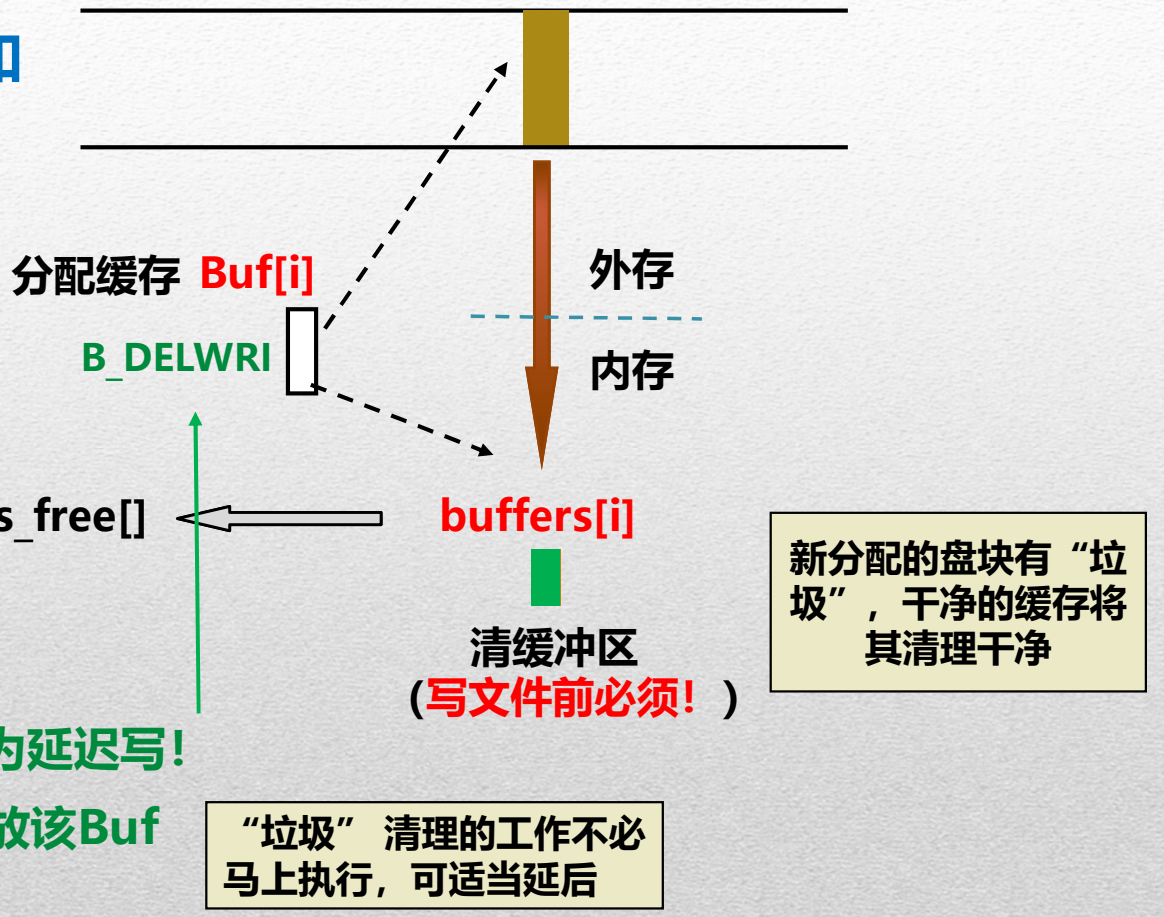
将文件逻辑块转换成物理块int Inode::Bmap(int lbn)



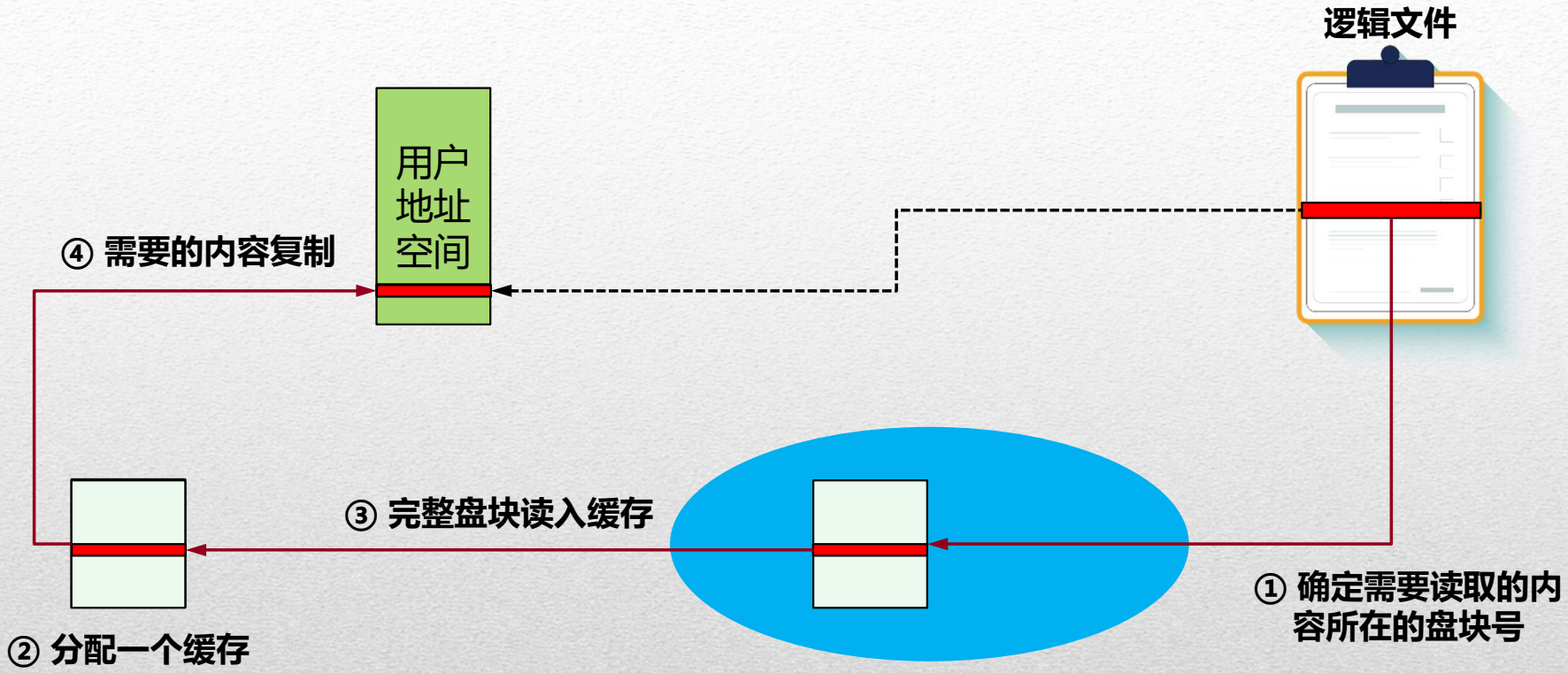
Bmap不仅完成逻辑地址到物理地址的转换，同时完成文件的动态增长。

将文件逻辑块转换成物理块int Inode::Bmap(int lbn)

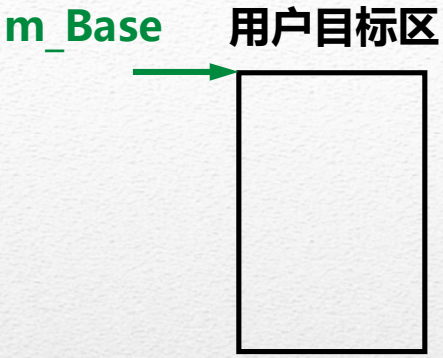
到内存SuperBlock中的s_nfree和s_free[]中查询一个新的盘块号



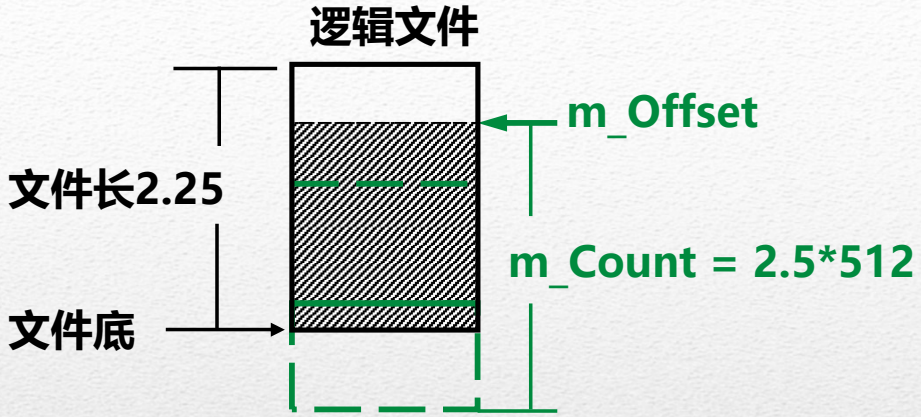
读操作


逻辑文件
用户
地址
空间
④ 需要的内容复制
③ 完整盘块读入缓存
① 确定需要读取的内
容所在的盘块号
② 分配一个缓存

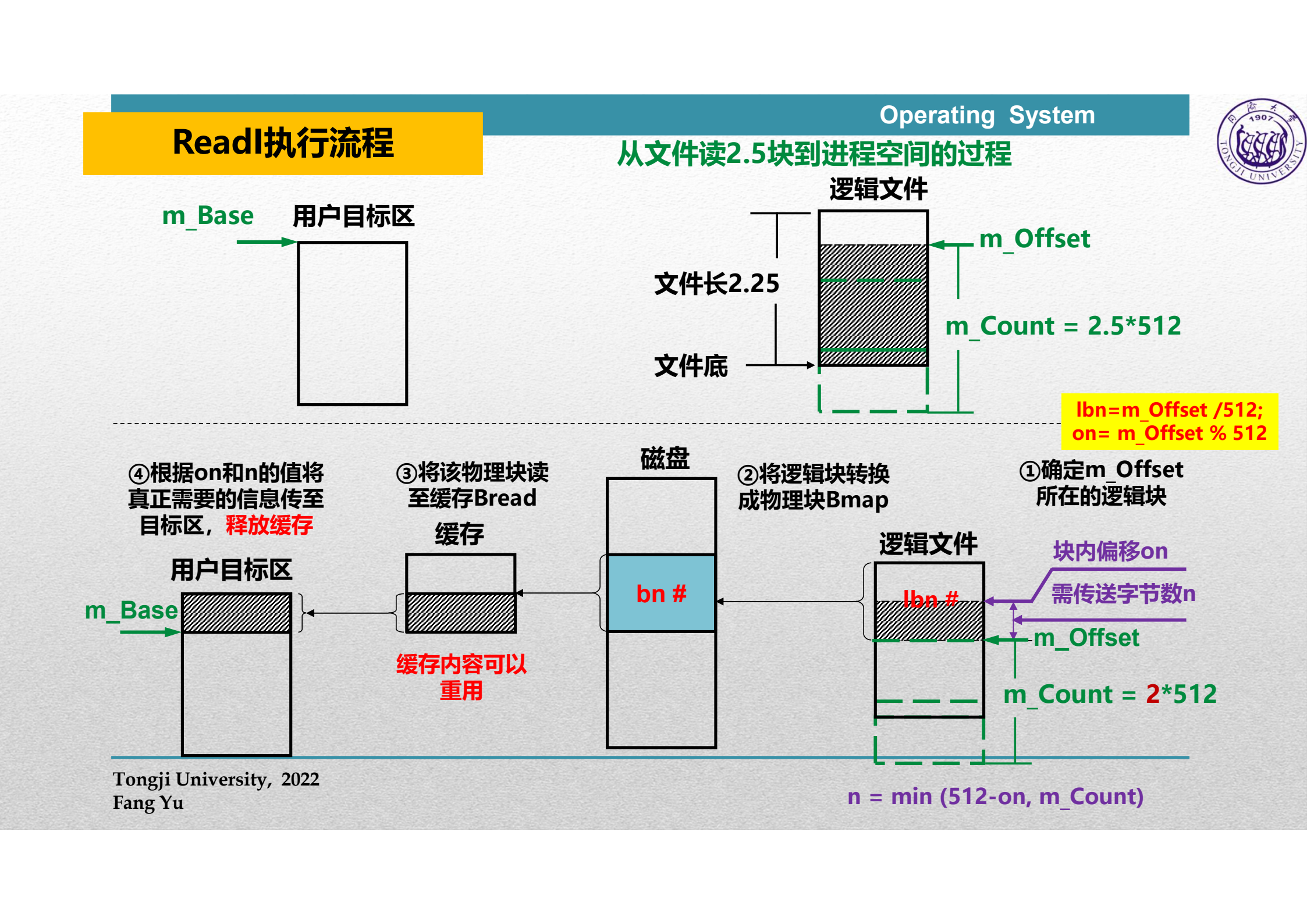
ReadI执行流程



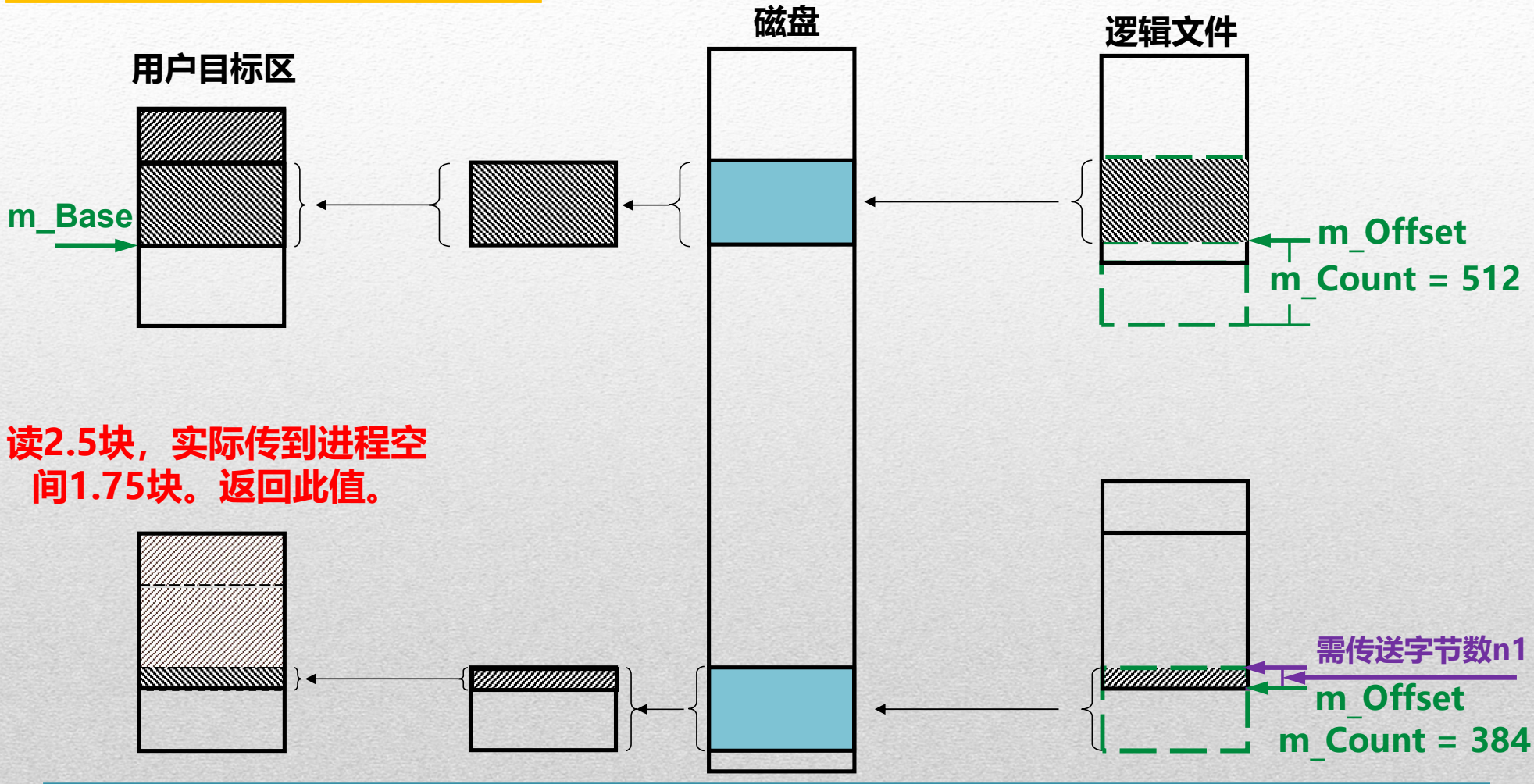
从文件读2.5块到进程空间的过程



ReadI执行流程

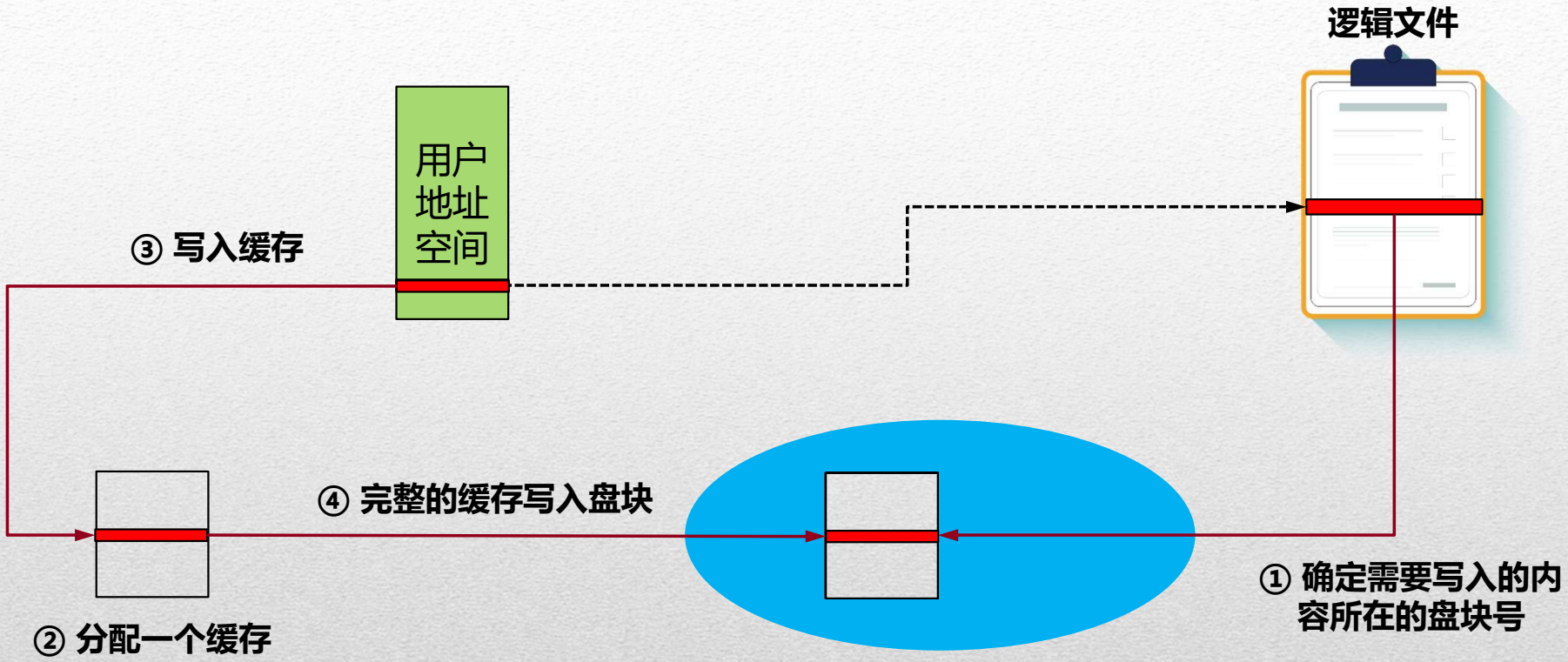

从文件读2.5块到进程空间的过程
逻辑文件
m_Base
用户目标区
m_Offset
文件长2.25
m_Count = 2.5*512
文件底
lbn=m_Offset /512;
on= m_Offset % 512
④根据on和n的值将真正需要的信息传至目标区，释放缓存
③将该物理块读至缓存Bread
磁盘
②将逻辑块转换成物理块Bmap
①确定m_Offset所在的逻辑块
缓存
逻辑文件
用户目标区
块内偏移on
需传送字节数n
m_Base
bn #
lbn #
m_Offset
缓存内容可以重用
m_Count = 2*512
n = min (512-on, m_Count)

ReadI执行流程



如果文件在本块结束，则需传送的字节数为：
n1 = min (n，文件剩余字节数)

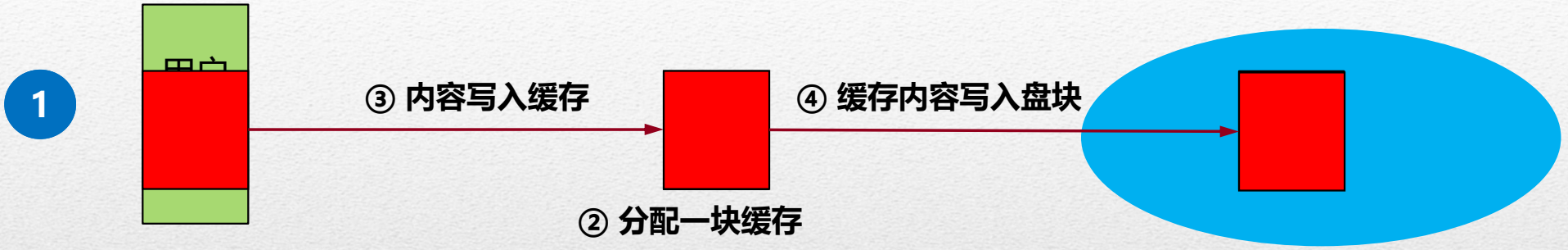
写操作



当写入部分不足一个盘块时，文件原有信息丢失

写操作

根据写入位置的不同分情况处理



写入的起始位置为
逻辑块的起始地址

+

写入字节数为512

YES

Getblk

将用户地址空间
内容写入缓存

有I/O

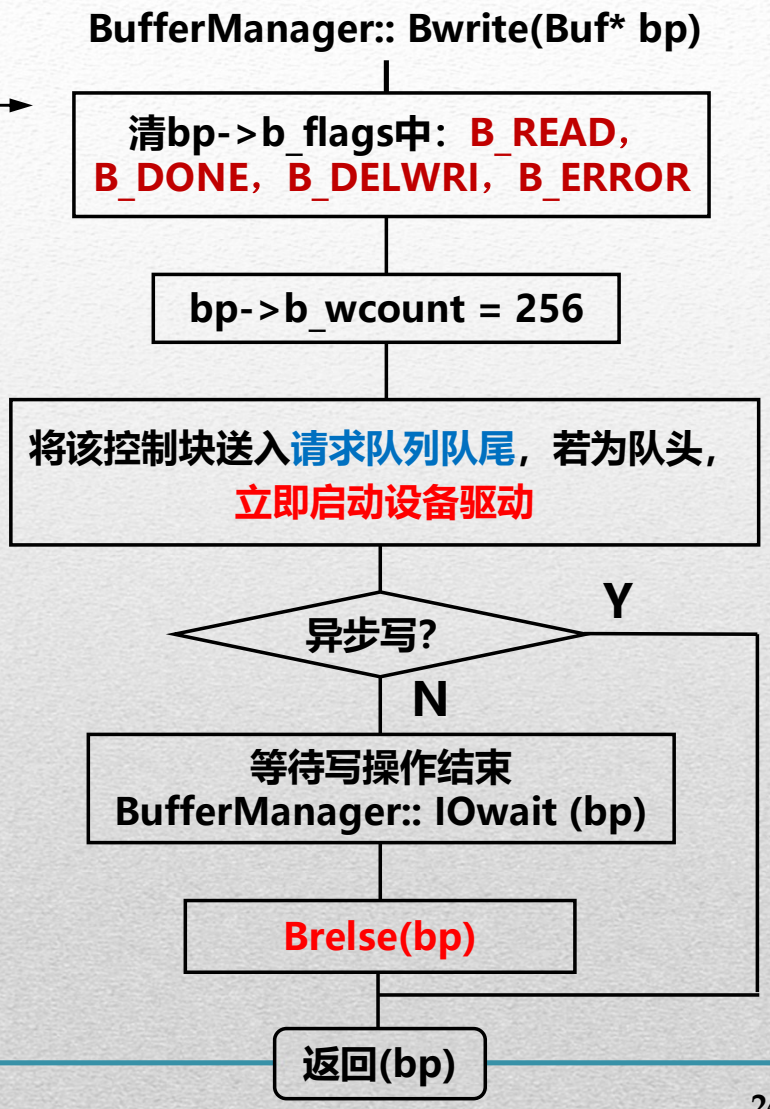
异步写 Bawrite(Buf* bp)

写操作

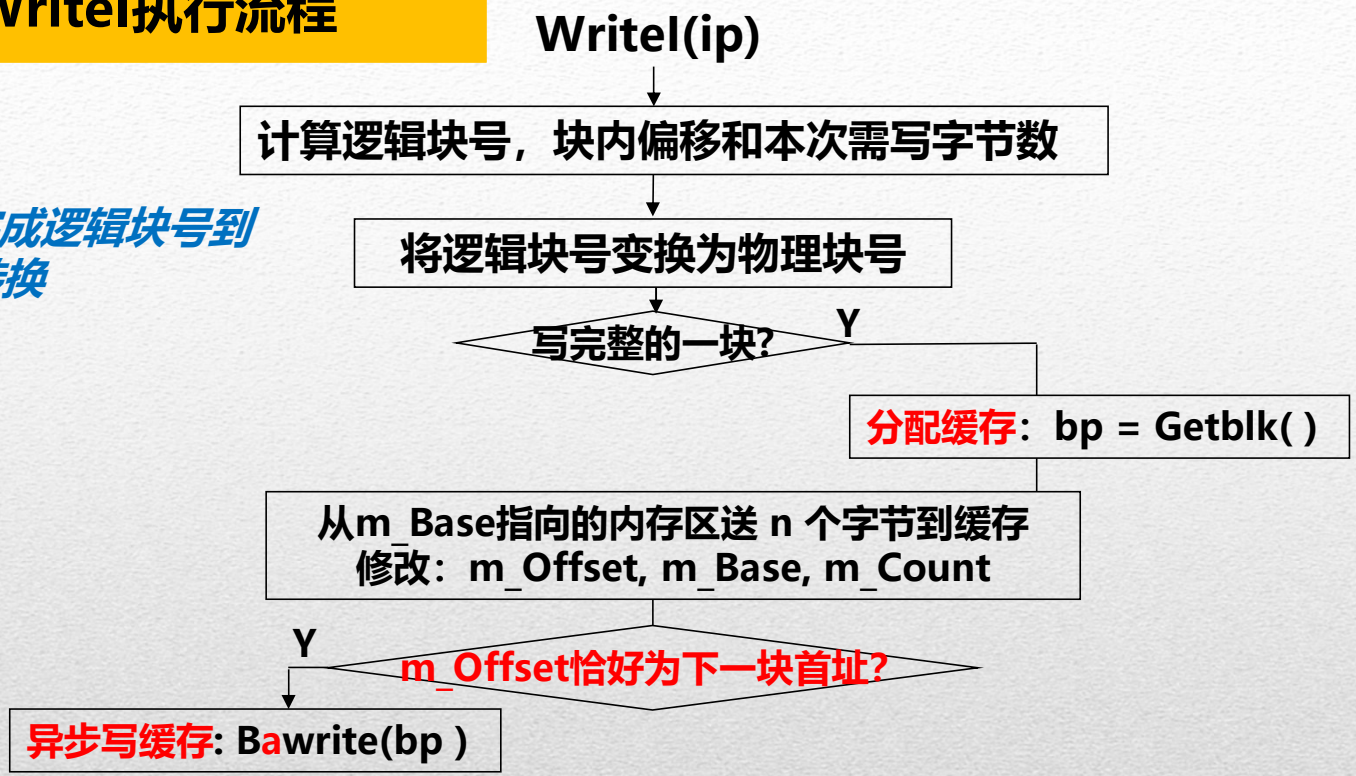
大多数写操作为异步写:
BufferManager:: Bawrite(Buf* bp)

加B_ASYNC 后调用:

BufferManager :: Bwrite(bp)



WriteI执行流程

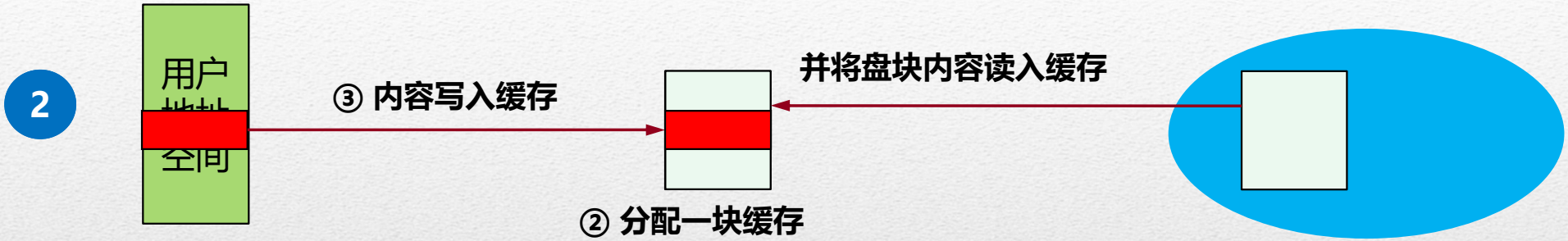


利用Bmap完成逻辑块号到物理块号的转换

本块已经写满

写操作

根据写入位置的不同分情况处理



写入的起始位置为
逻辑块的起始地址

+

写入字节数为512

YES → Getblk → 将用户地址空间内容写入缓存 → 异步写 Bawrite(Buf* bp)　　有I/O

NO → Bread（Getblk） → 将用户地址空间内容写入缓存 → 延迟写 Bdwrite(bp)　　没有I/O

写操作

大多数写操作为异步写:
BufferManager:: Bawrite(Buf* bp)

加B_ASYNC 后调用:

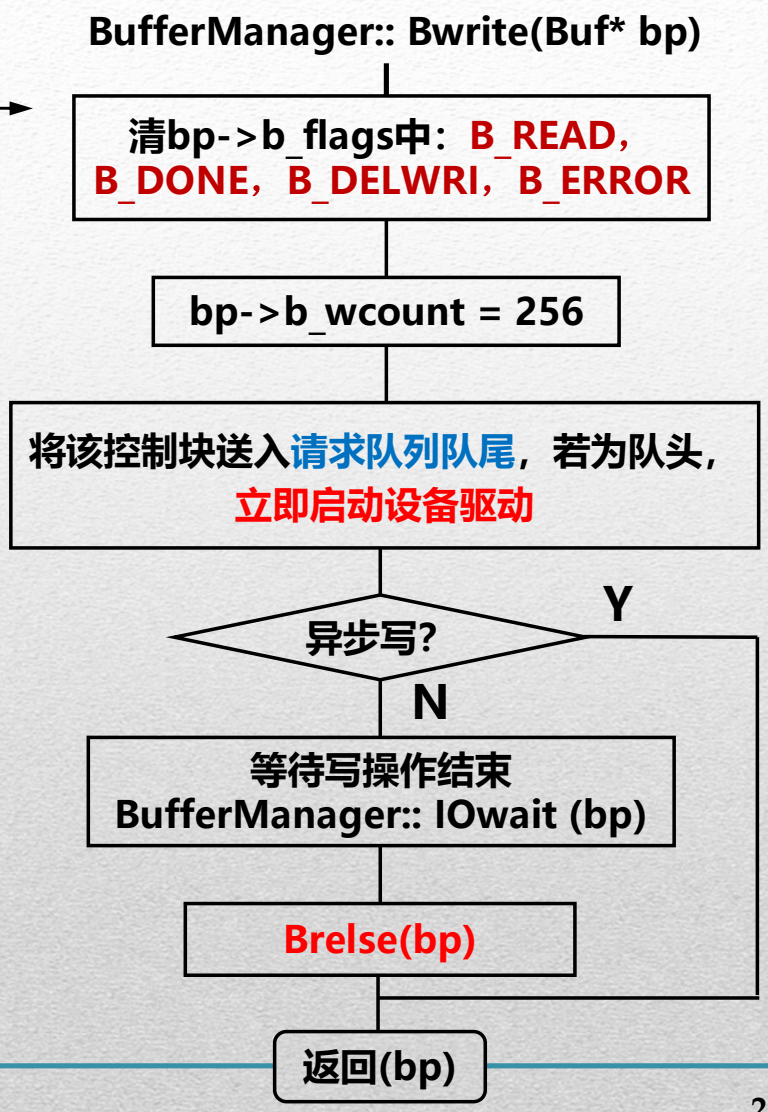
BufferManager :: Bwrite(bp)

写文件不足512字节时延迟写:

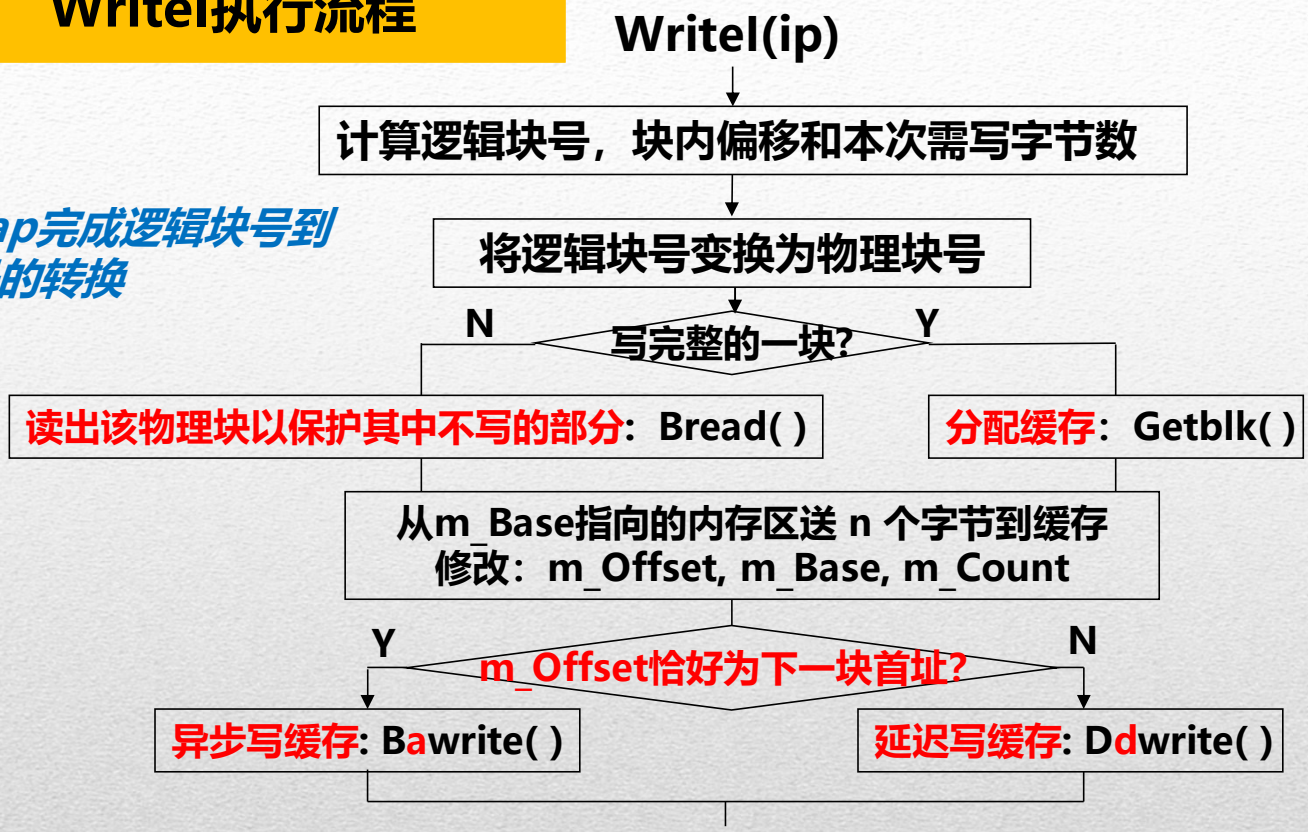
添b_wcount后，调用:

BufferManager :: Bdwrite(bp)

加 B_DONE，B_DELWRI 后直接释放到自由队尾！ (二次机会)



Writei执行流程

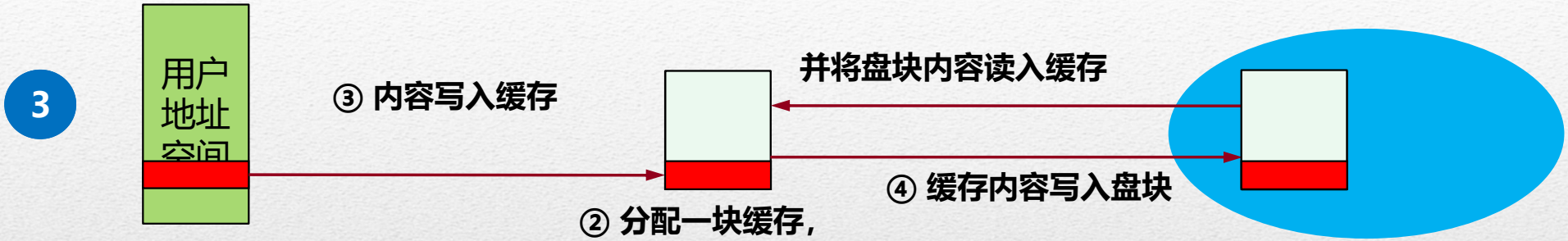


利用Bmap完成逻辑块号到物理块号的转换

本块还能再写?

写操作

根据写入位置的不同分情况处理



写入的起始位置为逻辑块的起始地址

+

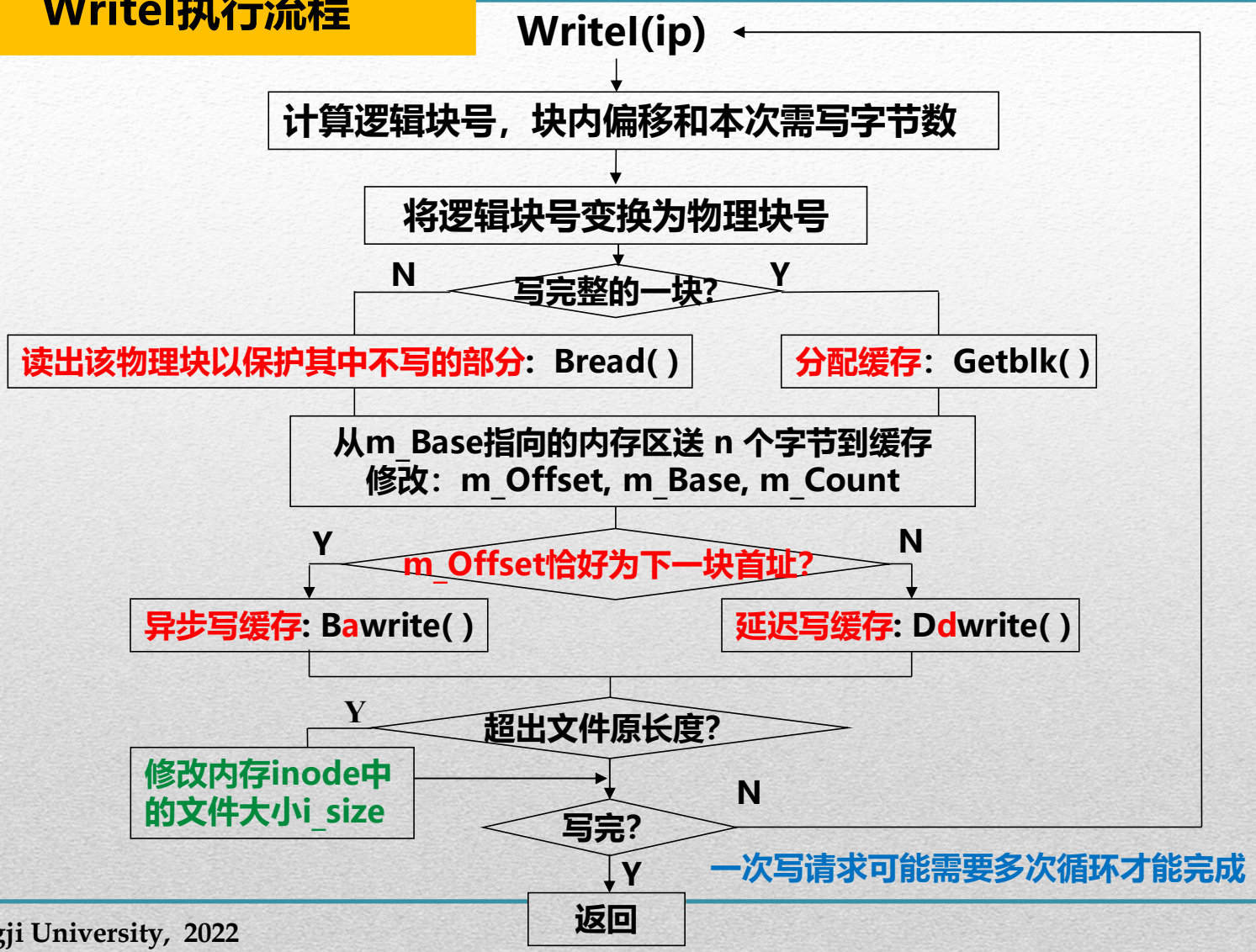
写入字节数为512

YES → Getblk → 将用户地址空间内容写入缓存 → 异步写 Bawrite(Buf* bp)（有I/O）

结束位置为下一个逻辑块起始位置

NO → Bread（Getblk） → 将用户地址空间内容写入缓存 → 延迟写 Bdwrite(bp)（没有I/O）

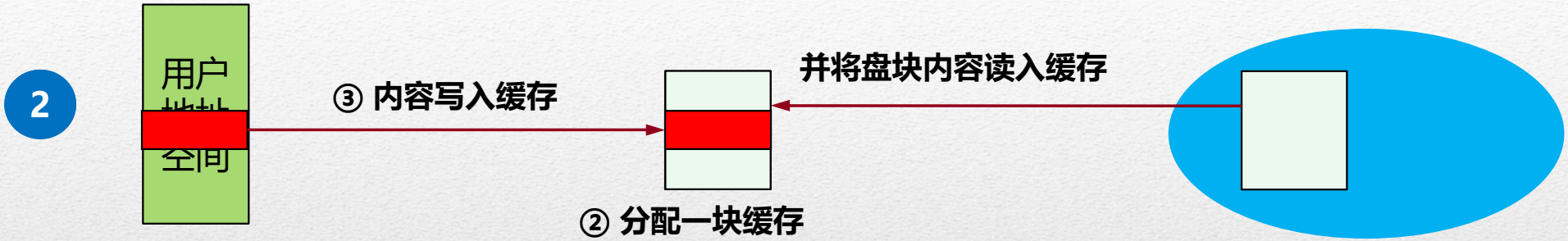
WriteI执行流程



一次写请求可能需要多次循环才能完成

写操作

根据写入位置的不同分情况处理



写入的起始位置为
逻辑块的起始地址

+

写入字节数为512

YES

Getblk

将用户地址空间
内容写入缓存

有I/O

异步写 Bawrite(Buf* bp)

Getblk中摘下自由队列队头有
B_DELWRITE标志的缓存

结束位置为下一个
逻辑块起始位置

没有I/O

NO

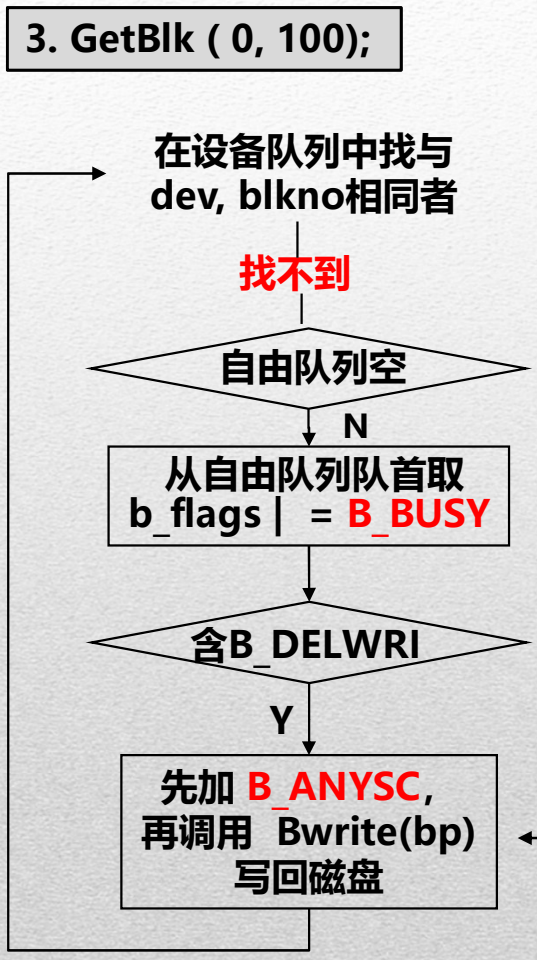
Bread（Getblk）

将用户地址空间
内容写入缓存

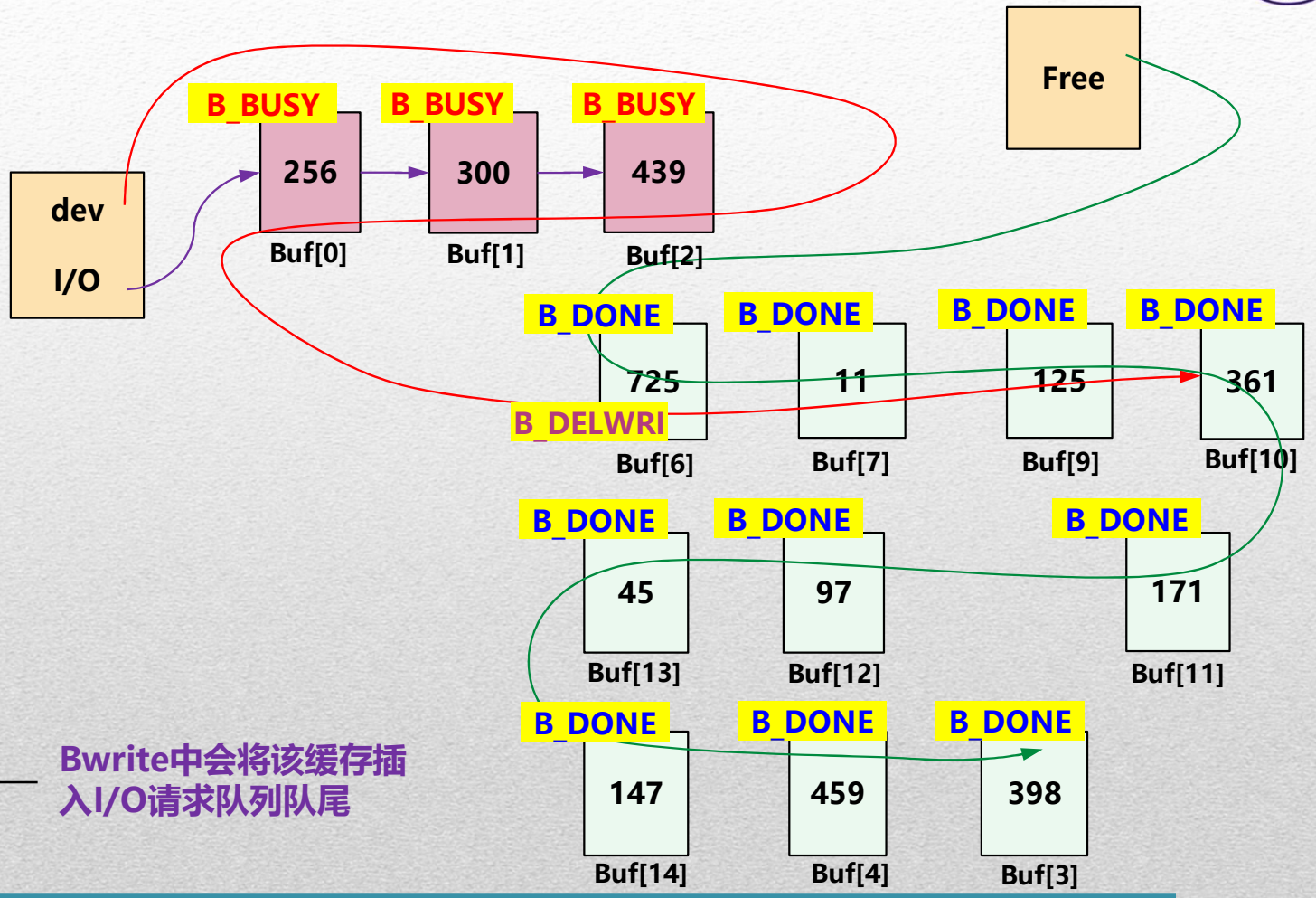
延迟写 Bdwrite(bp)

BufferManager::GetBlk(dev, blkno)过程

3. GetBlk (0, 100);



Bwrite中会将该缓存插入I/O请求队列队尾



本节小结：

1 UNIX文件系统的读写操作

2 UNIX的管道文件

E10：UNIX V6++文件系统的实施

期末考试范围：

UNIX操作系统：

(1) UNIX中断和系统调用
(2) UNIX进程控制
(3) UNIX设备管理
(4) UNIX文件系统

一般操作系统：

(1) 设备管理基本概念
(2) 文件系统基本概念

期末考试范围：

UNIX操作系统：

(1) UNIX中断和系统调用
(2) UNIX进程控制
(3) UNIX设备管理
(4) UNIX文件系统

一般操作系统：

(1) 设备管理基本概念
(2) 文件系统基本概念

S14 ~ S19，E06 ~ E08，P04，P05，U03

UNIX文件系统与设备管理：

1. 缓存的分配与回收
2. 文件系统静态结构：
   - 磁盘空间的划分及存储空间管理
   - 文件的静态结构（Inode节点结构与索引结构）
3. 文件的内存打开结构及打开过程
4. 目录结构
   - 目录文件与树状目录结构之间的对应关系
   - 目录搜索过程及文件的创建、删除对目录项的修改
5. read与write系统调用的执行过程（结合缓存的分配与回收）

一般操作系统：

1. 磁盘管理
   - 柱面/磁道/扇区的概念
   - 磁盘的读写原理
   - 磁盘调度算法
2. 文件系统
   - 文件的逻辑结构与物理结构
   - 顺序文件、链接文件、索引文件的特点与优缺点