第六章

文件管理

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

- 常见的文件物理结构
- UNIX文件的物理结构
- UNIX文件的打开结构
- UNIX文件系统的读写操作

6.3 文件存储空间管理

6.4 文件系统的目录管理

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

如果要读取磁盘上的i# inode，需要：

(1) 计算 i# inode所在的盘块号blkno： i / (512/64) + Inode区的起始盘块号;

(2) 调用Bread(dev, blkno)将该盘块的512个字节读入一个缓存;

(3) 从缓存中读取i# inode的64个字节。

磁盘上的Inode节点会导致文件访问效率低

```
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 */
};
```

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)

i node
文件索引节点：文件控制块，FCB

UNIX文件的打开结构

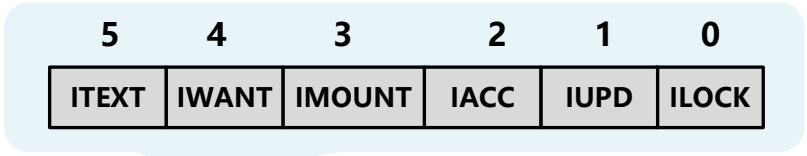
Super Block	inode区	文件数据区

每个文件还有一个内存文件控制块Inode（内存索引节点）

```
class Inode
{
        …;
/* Functions */
public:
        void ReadI();    /* 根据Inode对象中的物理磁盘块索引表，读取文件数据 */
        void WriteI();   /* 根据Inode对象中的物理磁盘块索引表，将数据写入文件 */
        int  Bmap(int lbn);      /* 将文件的逻辑块号转换成对应的物理盘块号 */
        void OpenI(int mode);    /* 打开文件 */
        void CloseI(int mode);   /* 关闭文件 */
        void IUpdate(int time);  /* 更新外存Inode的最后的访问时间、修改时间 */
        void ITrunc();           /* 释放Inode对应文件占用的磁盘块 */
        void Clean();                    /* 清空Inode对象中的数据 */
        void ICopy(Buf* bp, int inumber); /* 将外存Inode信息拷贝到内存Inode中 */
```

UNIX文件的打开结构

每个文件还有一个内存文件控制块Inode（内存索引节点）




```
/* Members */
public:
        unsigned int i_flag;      /* 状态的标志位，定义见enum INodeFlag */
        unsigned int i_mode;      /* 文件工作方式信息 */
        int      i_nlink;         /* 该文件在目录树中不同路径名的数量 */
        short    i_uid;           /* 文件所有者的用户标识数 */
        short    i_gid;           /* 文件所有者的组标识数 */
        int      i_size;          /* 文件大小，字节为单位 */
        int      i_addr[10];      /* 文件逻辑块号和物理块号转换的基本索引表 */
        short    i_dev;           /* 外存inode所在存储设备的设备号 */
        int      i_number;        /* 外存inode区中的编号 */
        int      i_lastr;         /* 存放最近一次读取文件的逻辑块号 */
        int      i_count;         /* 引用计数 */
        static int rablock;       /* 需要预读的物理块号   */
};
```

内存索引节点

UNIX文件的打开结构

每个文件还有一个内存文件控制块Inode（内存索引节点）

```
/* Members */
public:
        unsigned int i_flag;
        unsigned int i_mode;
        int      i_nlink;
        short    i_uid;
        short    i_gid;
        int      i_size;
        int      i_addr[10];
        short    i_dev;
        int      i_number;
        int      i_lastr;
        int      i_count;
        static int rablock;
};
```

```
class DiskInode
{
public:
        unsigned int  d_mode;
        int           d_nlink;
        short         d_uid;
        short         d_gid;
        int           d_size;
        int           d_addr[10];


        int           d_atime;
        int           d_mtime;
};
```

内存索引节点

UNIX文件的打开结构

每个文件还有一个内存文件控制块Inode（内存索引节点）

```
/* Members */
public:
        unsigned int i_flag;
        unsigned int i_mode;
        int      i_nlink;
        short    i_uid;
        short    i_gid;
        int      i_size;
        int      i_addr[10];
        short    i_dev;
        int      i_number;
        int      i_lastr;
        int      i_count;
        static int rablock;
};
```

```
class DiskInode
{
public:
        unsigned int  d_mode;
        int           d_nlink;
        short         d_uid;
        short         d_gid;
        int           d_size;
        int           d_addr[10];

        int           d_atime;
        int           d_mtime;
};
```

内存索引节点

UNIX文件的打开结构

每个文件还有一个内存文件控制块Inode（内存索引节点）

```
/* Members */
public:
        unsigned int i_flag;
        unsigned int i_mode;
        int      i_nlink;
        short    i_uid;
        short    i_gid;
        int      i_size;
        int      i_addr[10];
        short    i_dev;
        int      i_number;
        int      i_lastr;
        int      i_count;
        static int rablock;
};
```

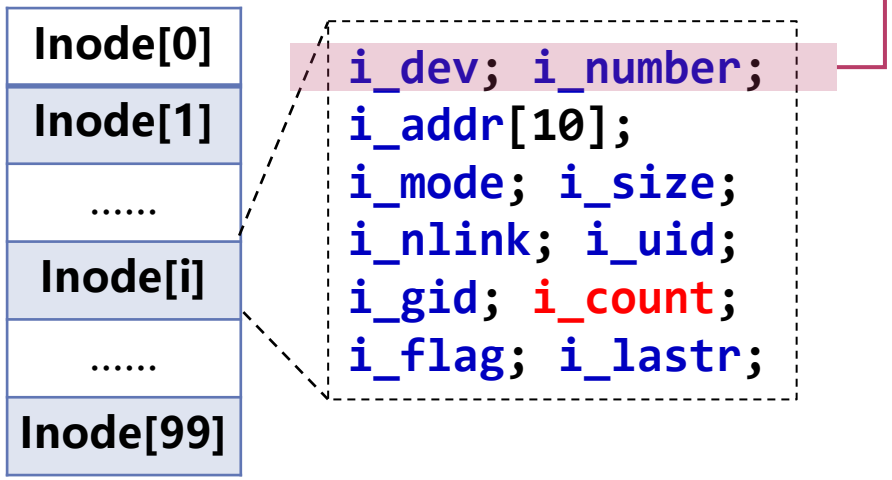
上一次读的块号

```
class DiskInode
{
public:
        unsigned int  d_mode;
        int           d_nlink;
        short         d_uid;
        short         d_gid;
        int           d_size;
        int           d_addr[10];
        int           d_atime;
        int           d_mtime;
};
```

内存索引节点

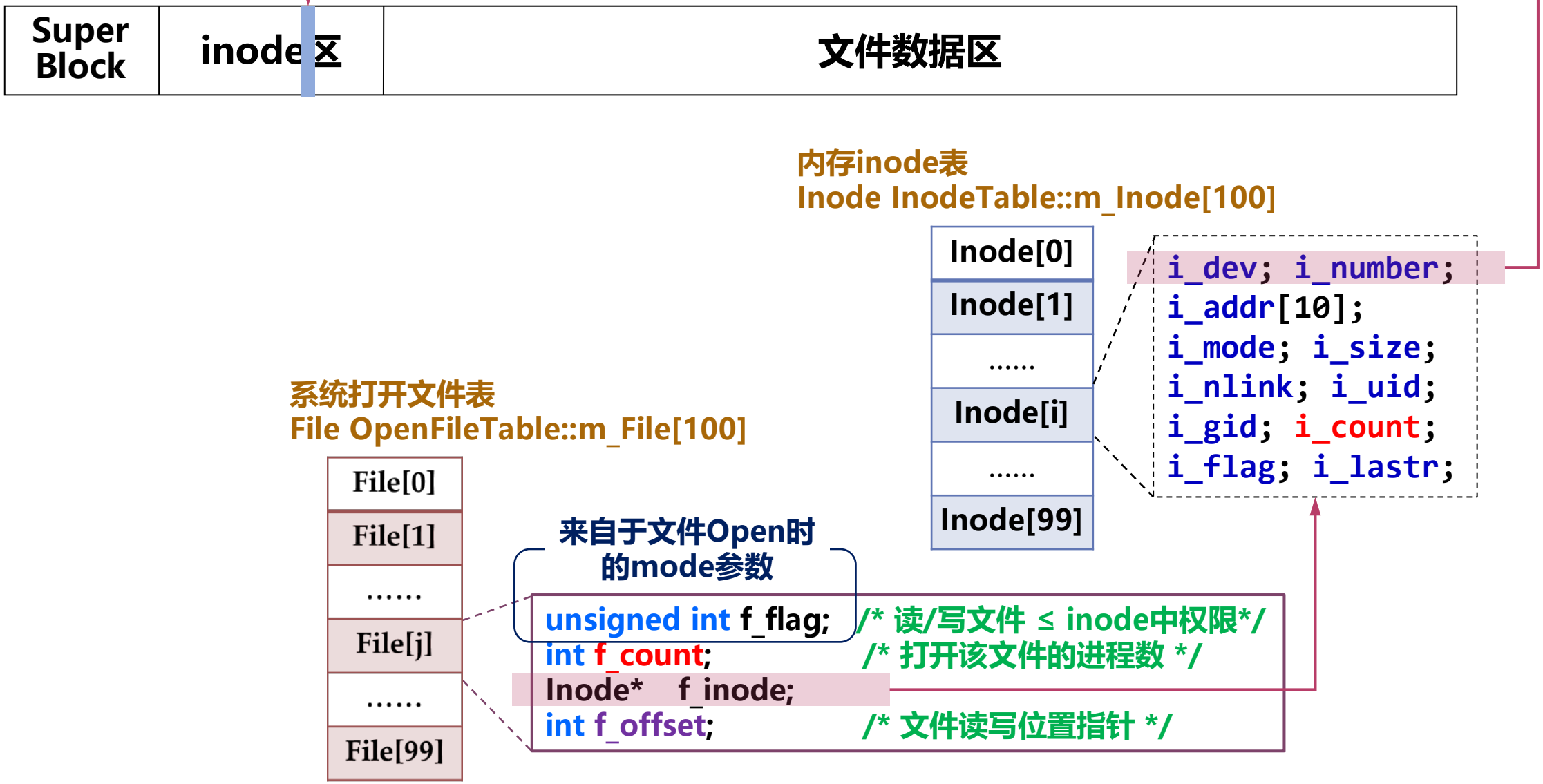
UNIX文件的打开结构

内存inode表
Inode InodeTable::m_Inode[100]



内存索引节点

UNIX文件的打开结构

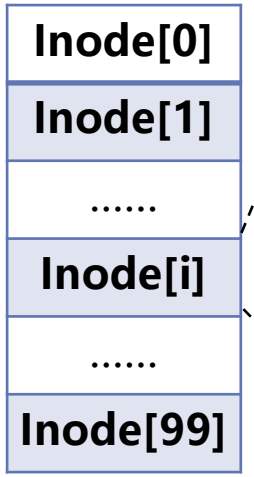


内存索引节点

UNIX文件的打开结构

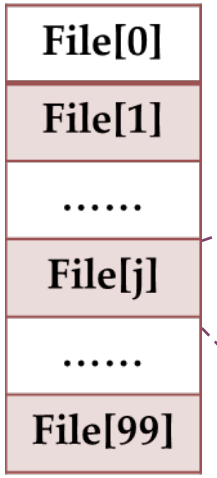
进程打开文件表
OpenFiles u_ofiles;
u_ofiles::ProcessOpenFileTable[15]

内存inode表
Inode InodeTable::m_Inode[100]




```
i_dev; i_number;
i_addr[10];
i_mode; i_size;
i_nlink; i_uid;
i_gid; i_count;
i_flag; i_lastr;
```

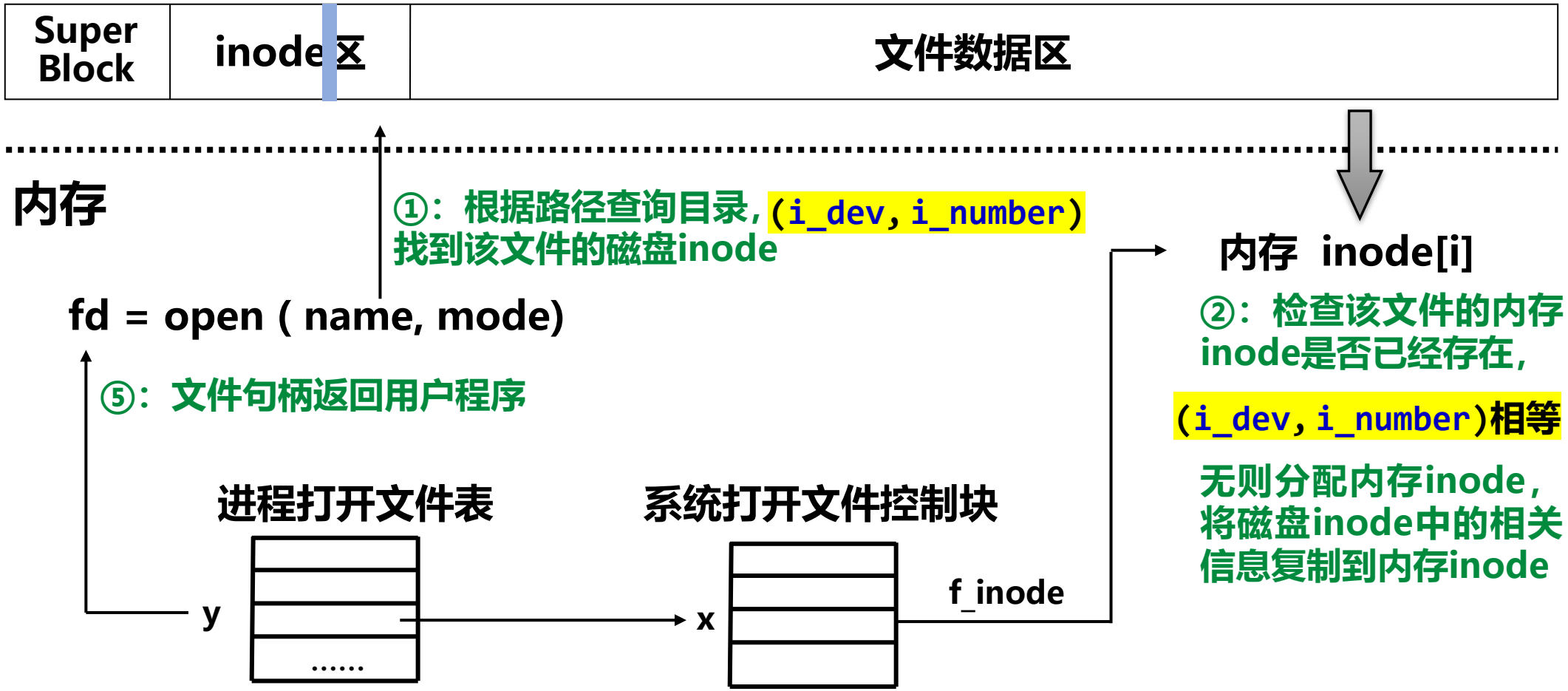
系统打开文件表
File OpenFileTable::m_File[100]



来自于文件Open时的mode参数

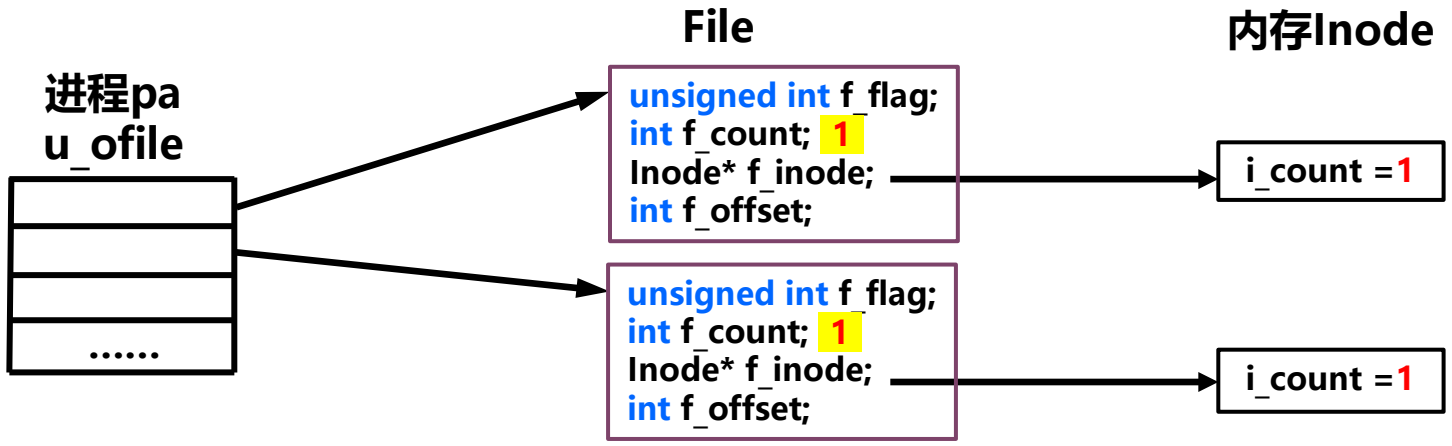
```
unsigned int f_flag;   /* 读/写文件 ≤ inode中权限*/
int f_count;           /* 打开该文件的进程数 */
Inode*    f_inode;
int f_offset;          /* 文件读写位置指针 */
```

UNIX文件的打开结构



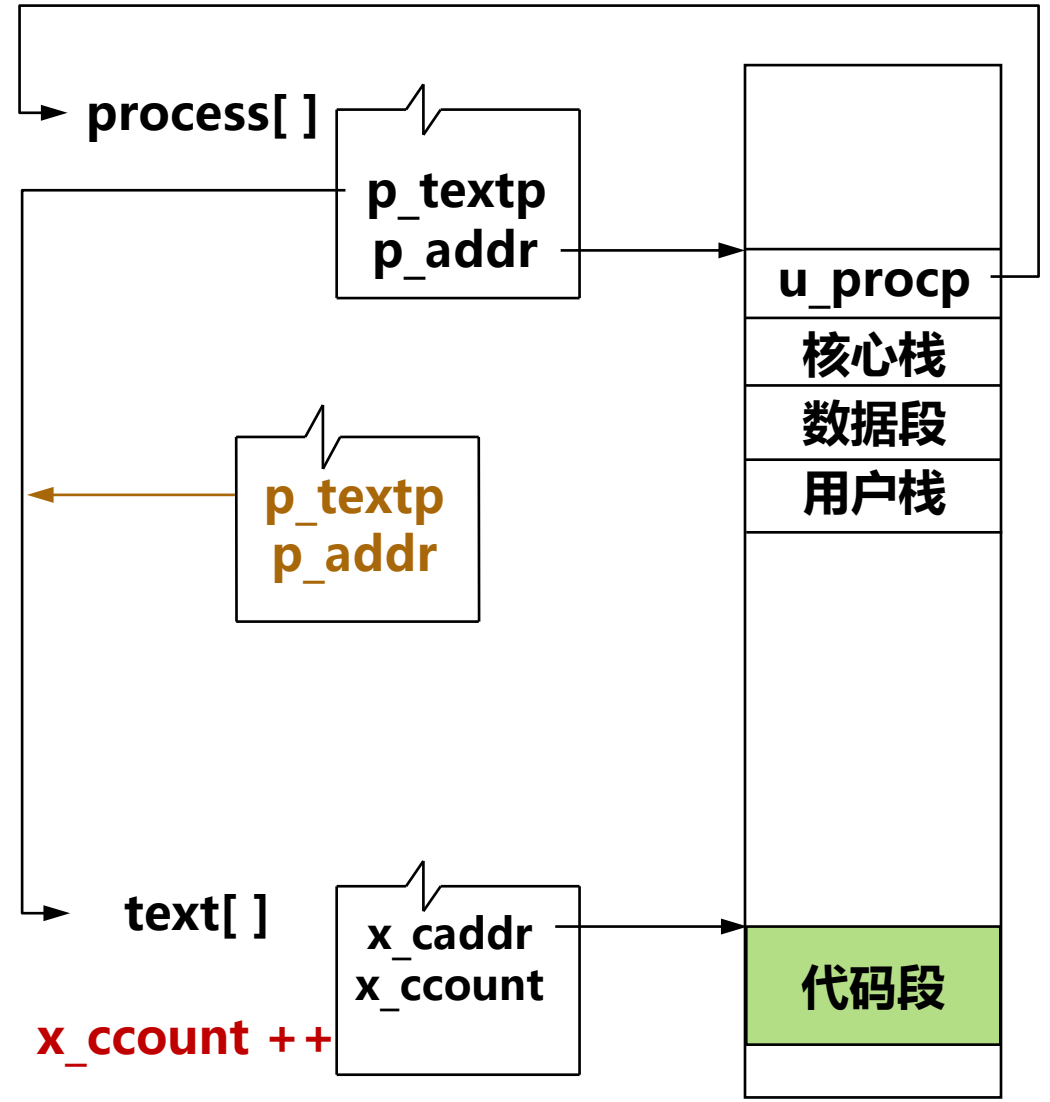
UNIX文件的打开结构

对文件的不同共享方式



进程的创建与终止

创建子进程的关键：
为子进程构造一个能正确上台的图像

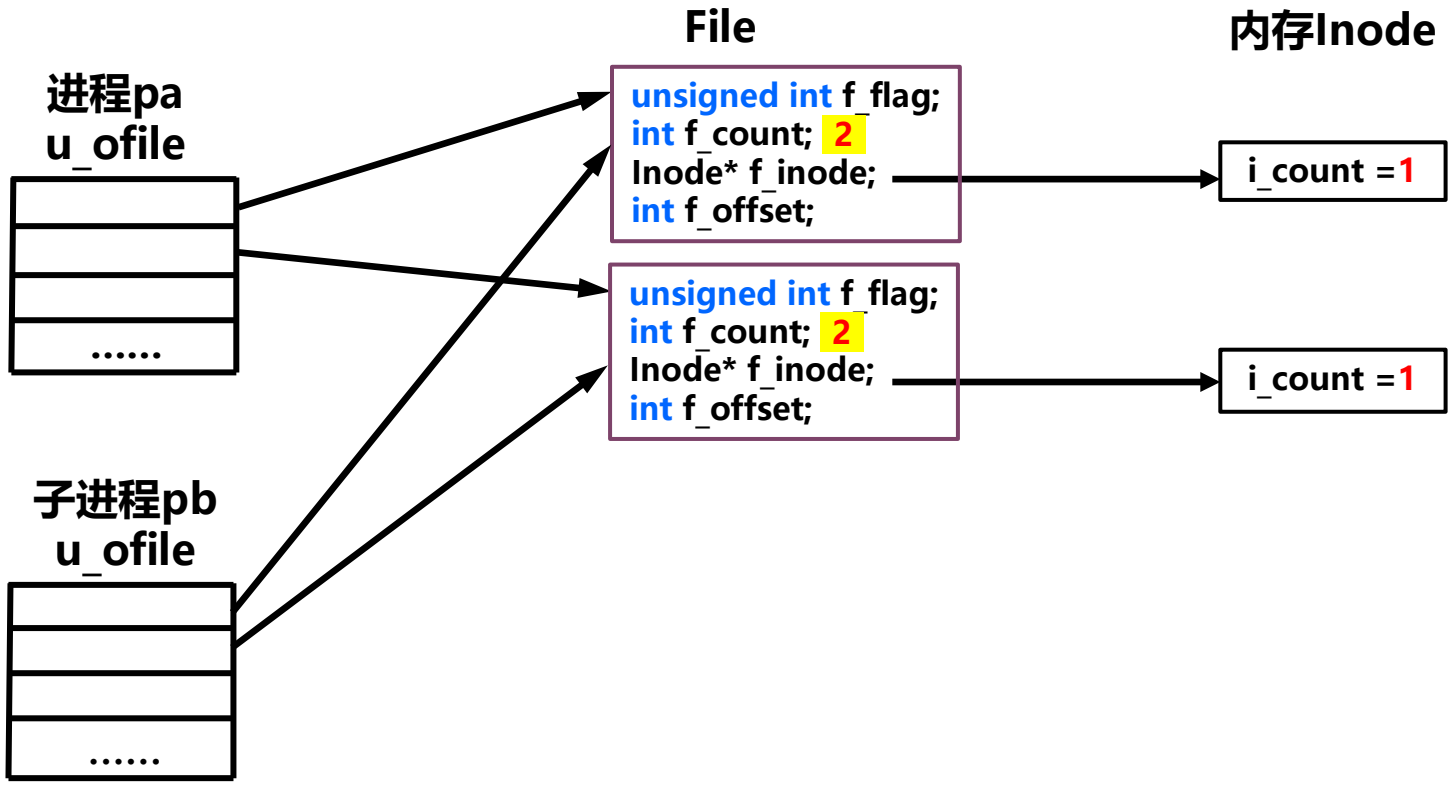


2

各种计数增1：

1. x_count，x_ccount（代码共享）
2. File结构引用数（文件共享）
3. u_cdir 指向的 i_count（目录共享）

UNIX文件的打开结构



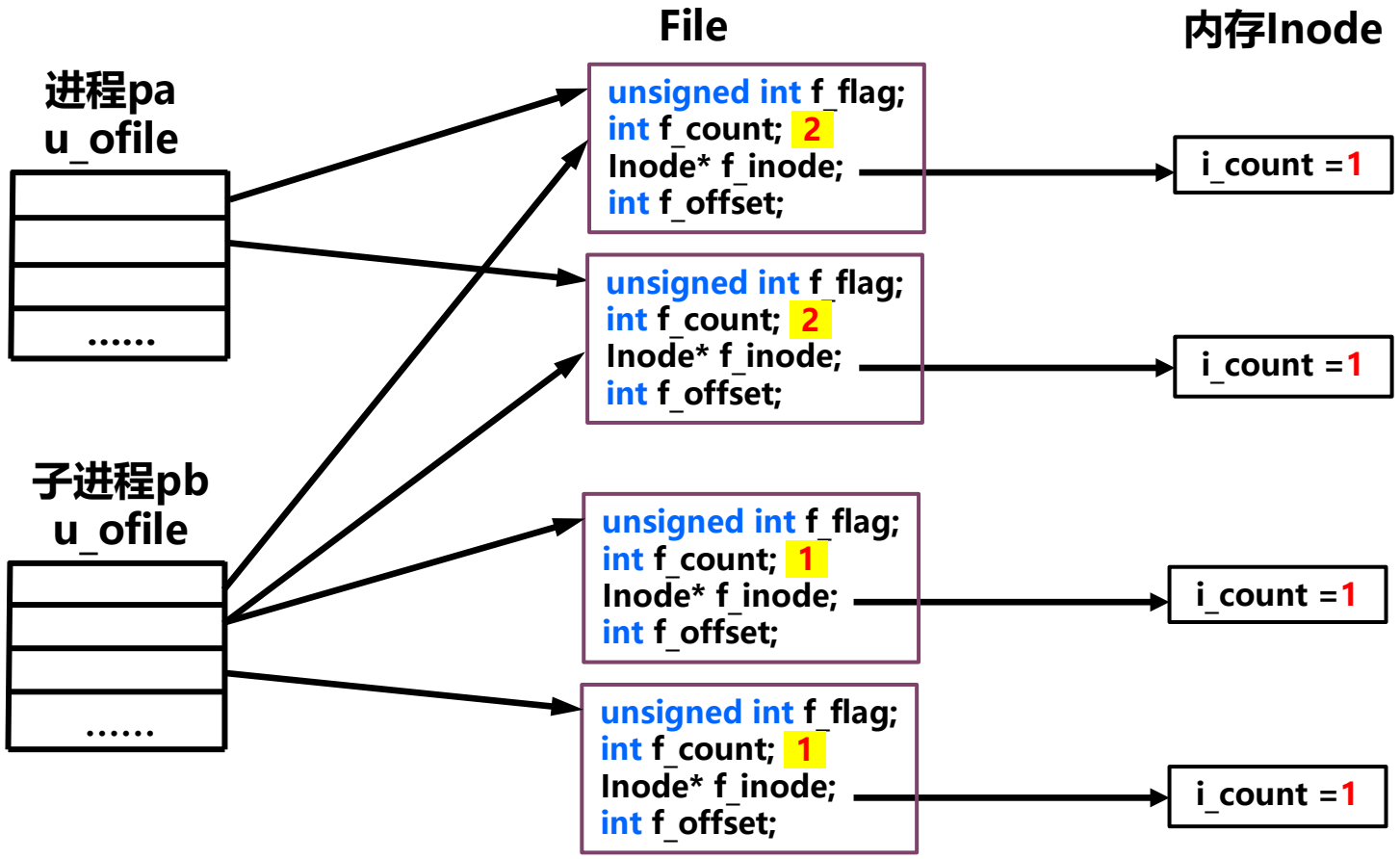
父进程创建子进程：

f_count ++;

父子进程之间可共享一个打开文件及其读写指针。

UNIX文件的打开结构

对文件的不同共享方式



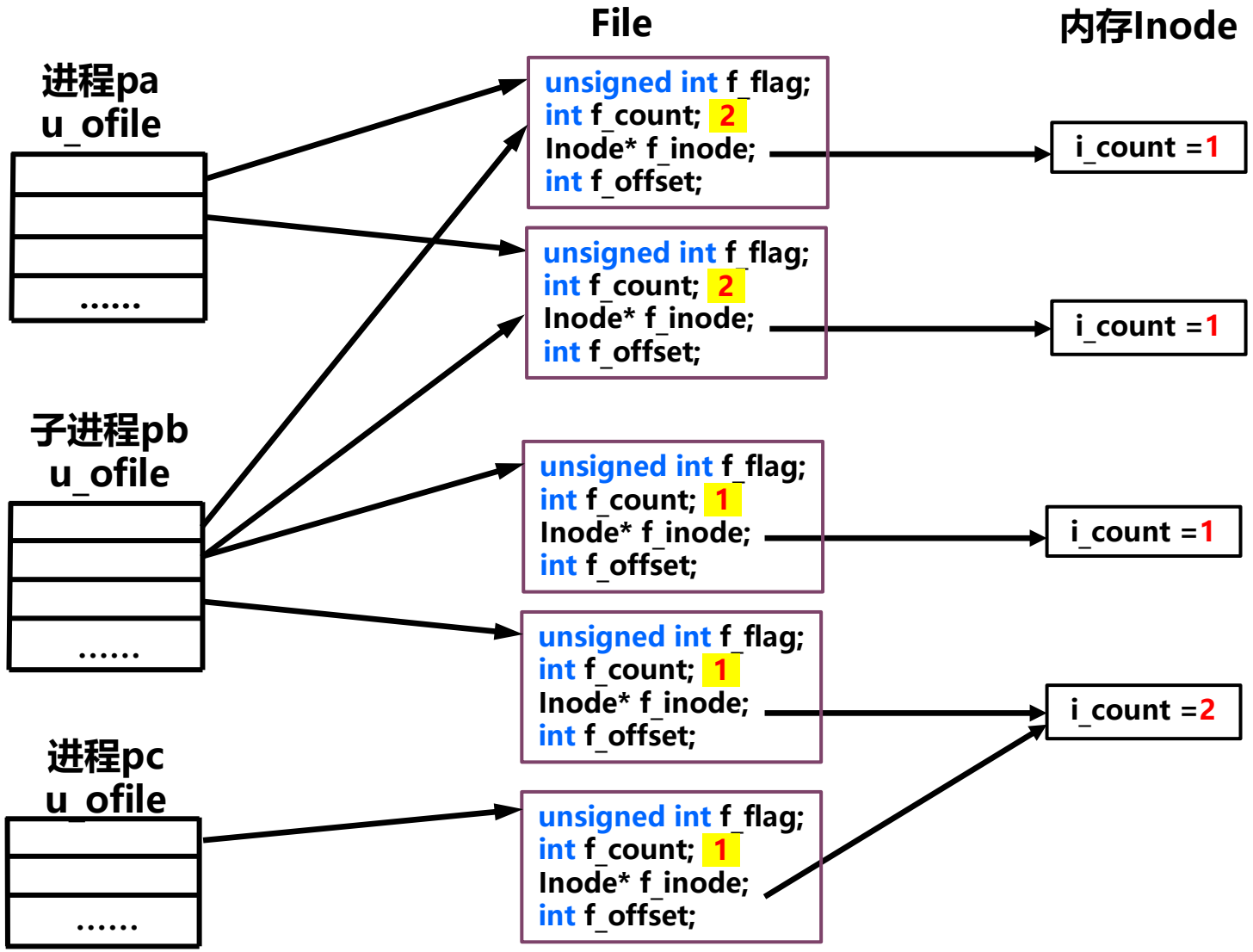
父进程创建子进程：

f_count ++;
父子进程之间可共享一个打开文件及其读写指针。

子进程打开新的文件

UNIX文件的打开结构

对文件的不同共享方式



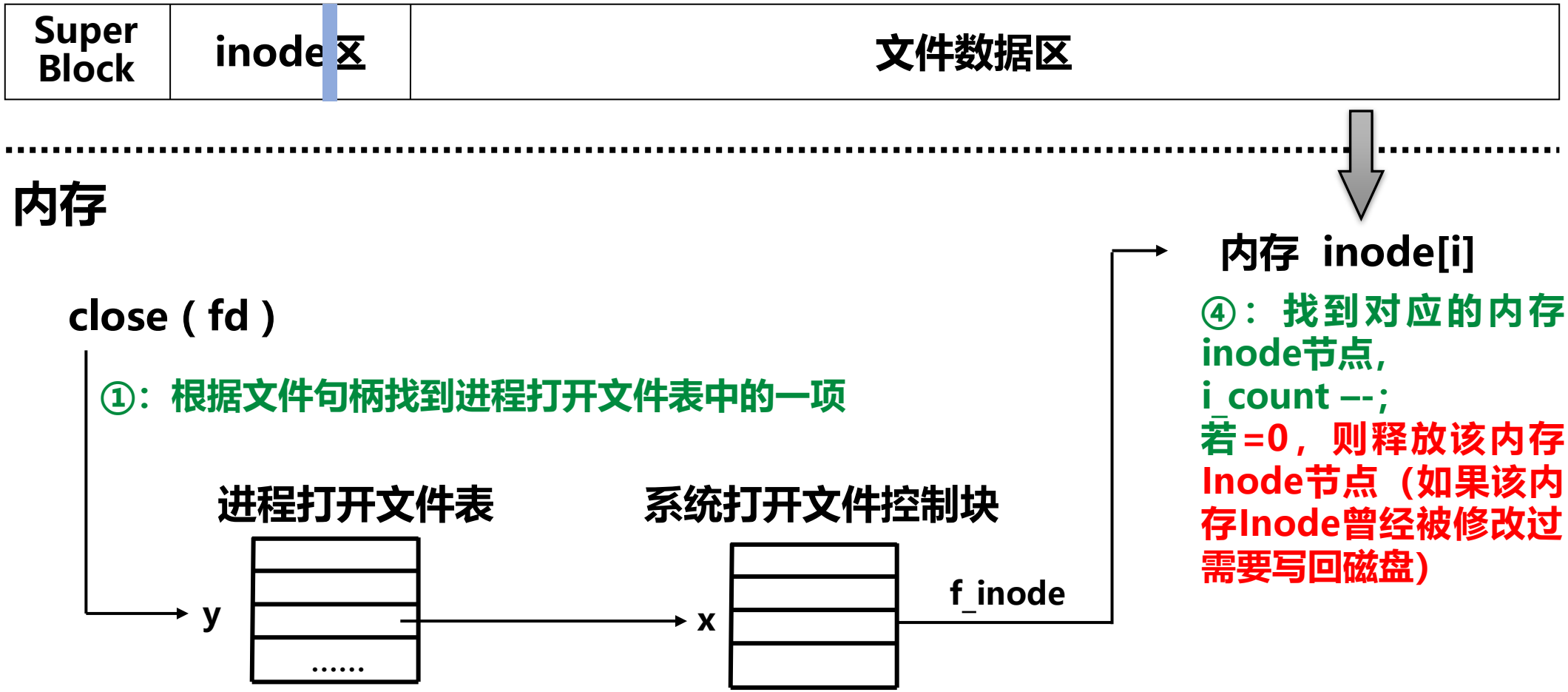
父进程创建子进程：

f_count ++;
父子进程之间可共享一个打开文件及其读写指针。

子进程打开新的文件

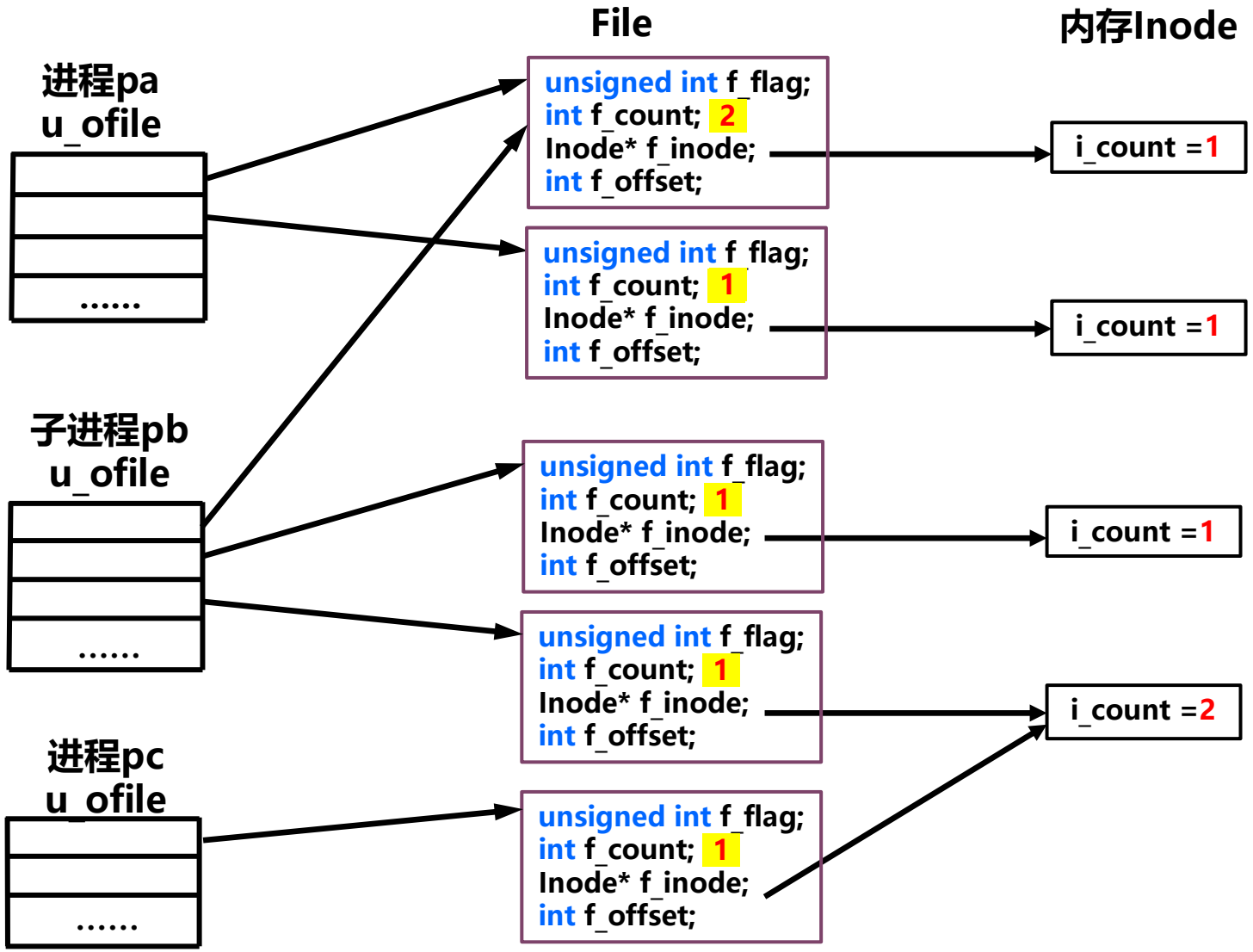
多个进程以不同的读写权限和指针打开同一个磁盘文件

UNIX文件的打开结构



UNIX文件的打开结构

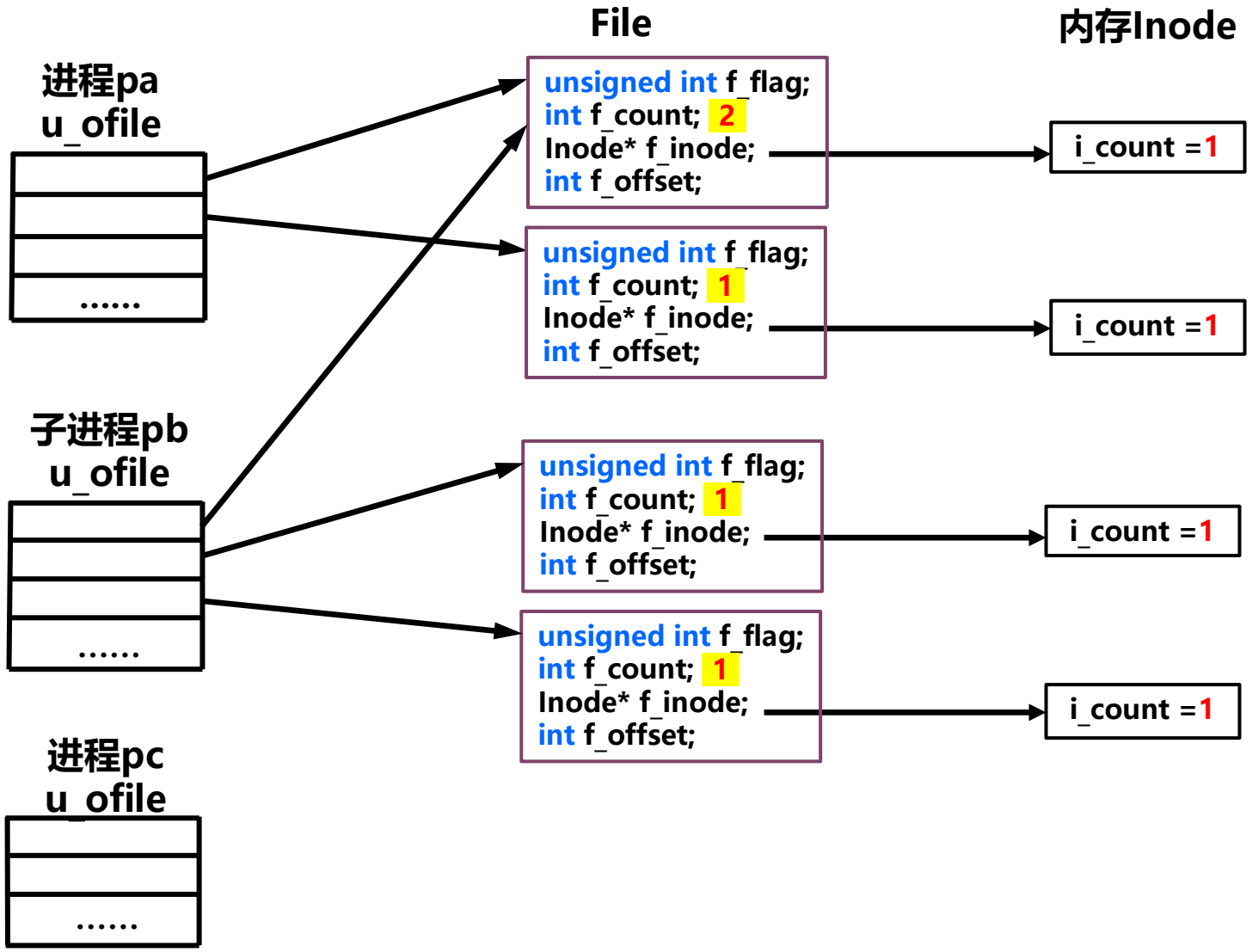
对文件的不同共享方式



子进程关闭与父进程共享的文件，只需要：
f_count --。

UNIX文件的打开结构

对文件的不同共享方式

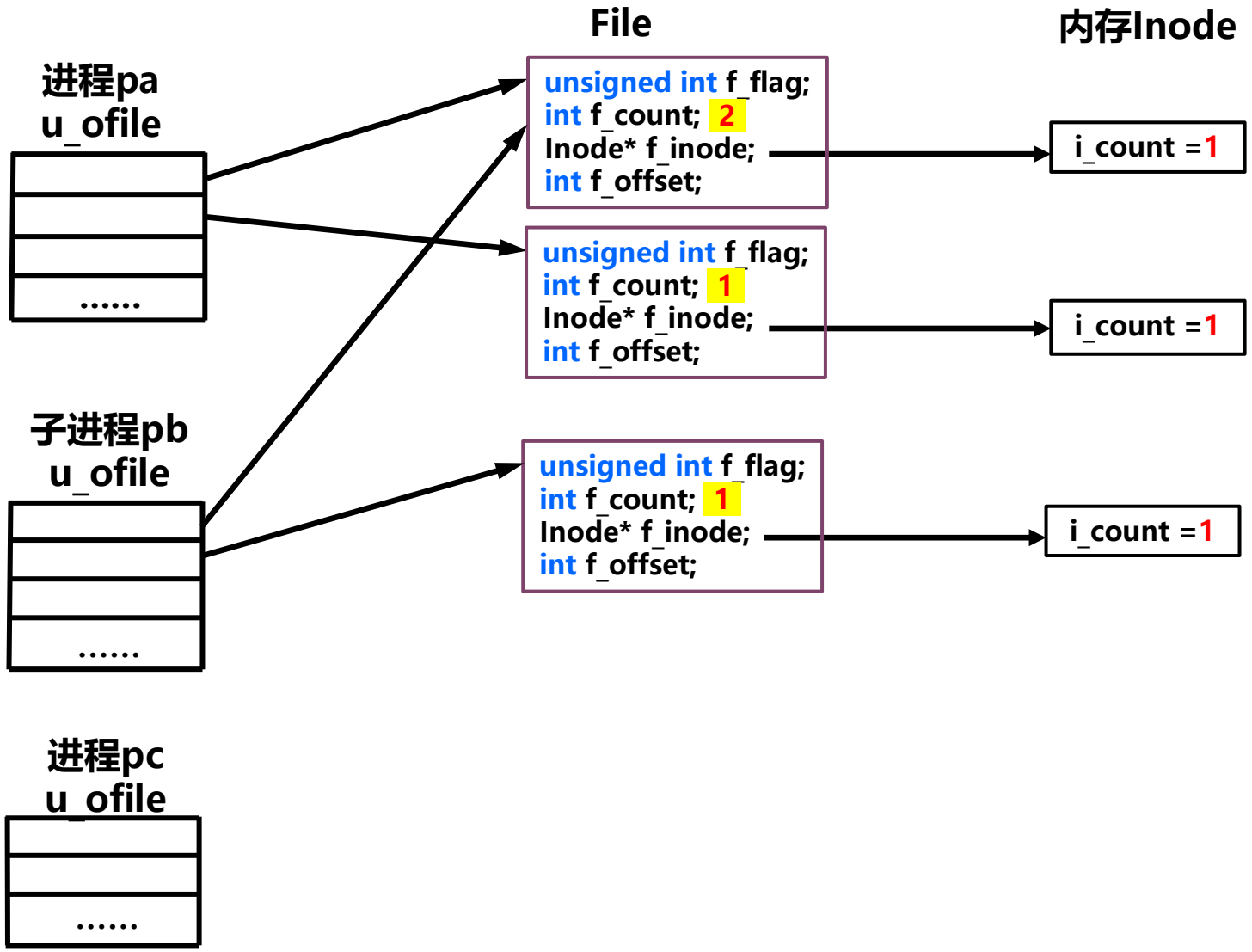


子进程关闭与父进程共享的文件，只需要：
f_count --。

进程关闭文件时，若：
f_count -- 后 = 0，
释放File结构。

UNIX文件的打开结构

对文件的不同共享方式



子进程关闭与父进程共享的文件，只需要：
f_count --。

进程关闭文件时，若：
i_count -- 后 = 0，
释放内存Inode节点。

进程关闭文件时，若：
f_count -- 后 = 0，
释放File结构。

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

- 常见的文件物理结构
- UNIX文件的物理结构
- UNIX文件的打开结构
- UNIX文件系统的读写操作

6.3 文件存储空间管理

6.4 文件系统的目录管理

UNIX文件系统的读写操作

读文件

```
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

1. read的系统调用号3存入EAX

3. EAX寄存器将带回返回值给res

```
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);
}
```

UNIX文件系统的读写操作

写文件

```
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;
}
```

2. 三个参数分别存入EBX、ECX、EDX

1. read的系统调用号4存入EAX

4. 执行INT 0x80指令

3. EAX寄存器将带回返回值给res

```
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);
}
```

UNIX文件系统的读写操作

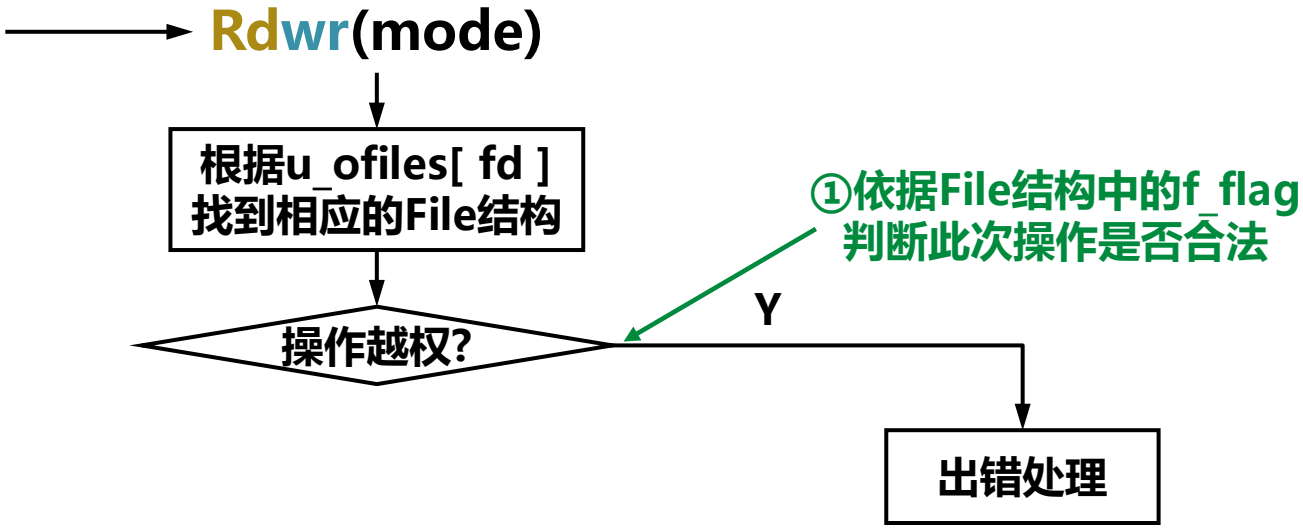
read (fd, buf, nbytes)
write (fd, buf, nbytes)
mode = *FREAD* / *FWRITE*

→ Rdwr(mode)

读写操作的共用流程

UNIX文件系统的读写操作

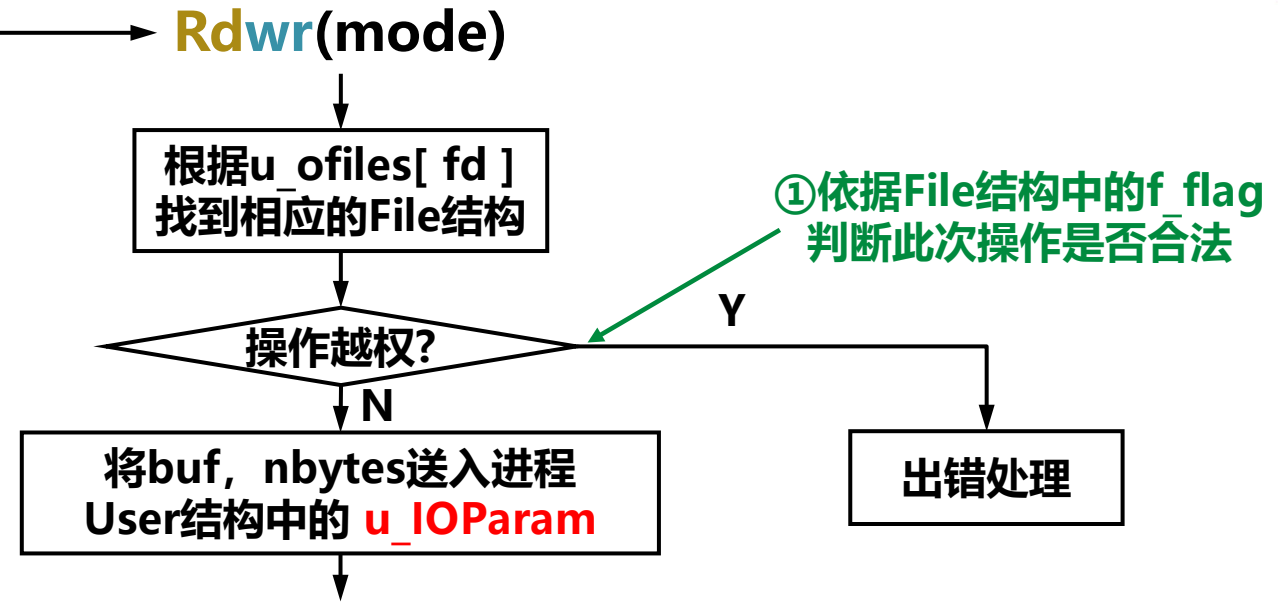
read (fd, buf, nbytes)
write (fd, buf, nbytes)
mode = *FREAD* / *FWRITE*



UNIX文件系统的读写操作

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

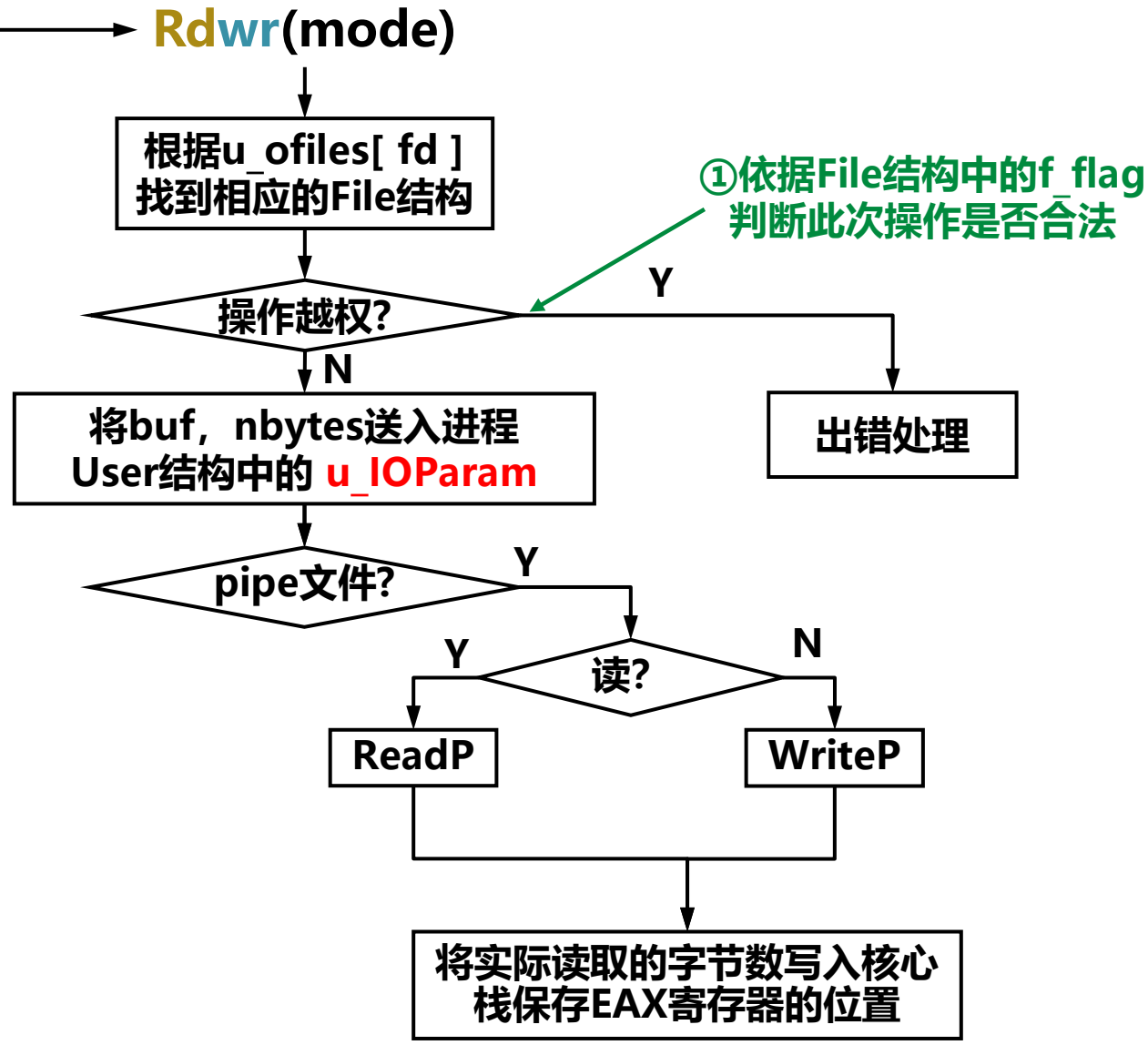
```
class IOParameter
{
public:    unsigned char* m_Base;
           int m_Offset;
           int m_Count;
};
m_Base ← buf
m_Offset
m_Count ← nbytes
```



UNIX文件系统的读写操作

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
m_Count ← nbytes
```

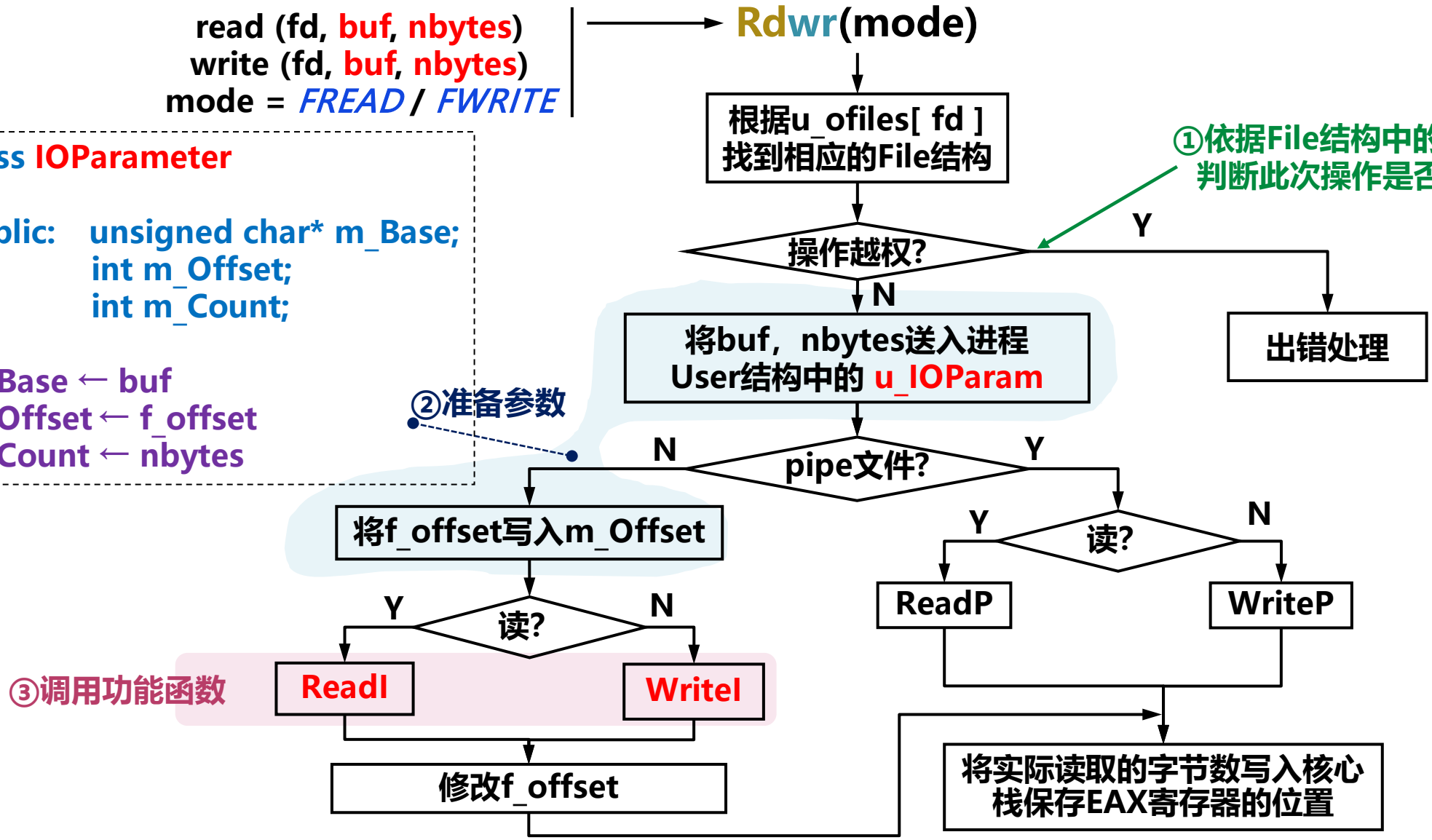


UNIX文件系统的读写操作

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



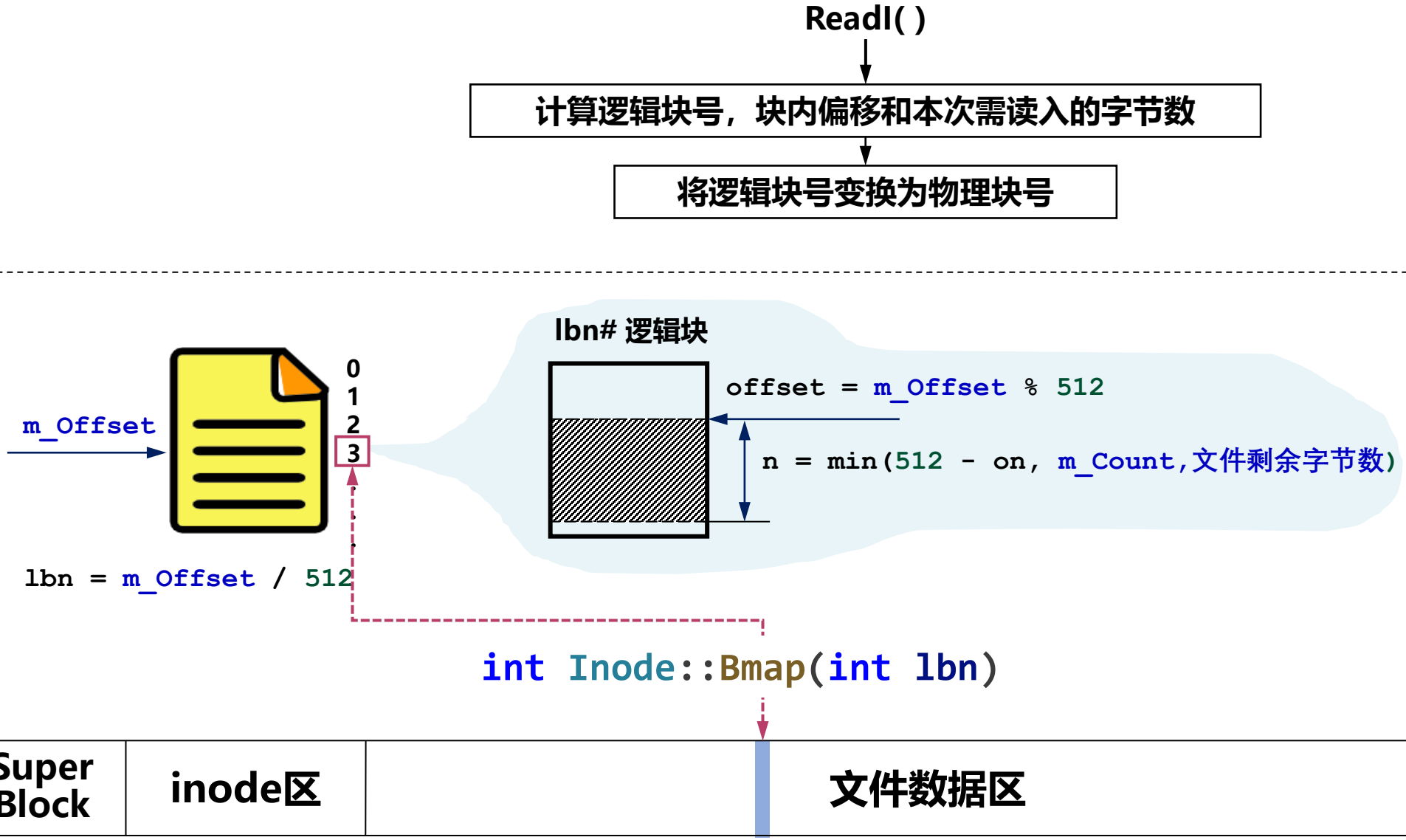
UNIX文件的打开结构

每个文件还有一个内存文件控制块Inode（内存索引节点）

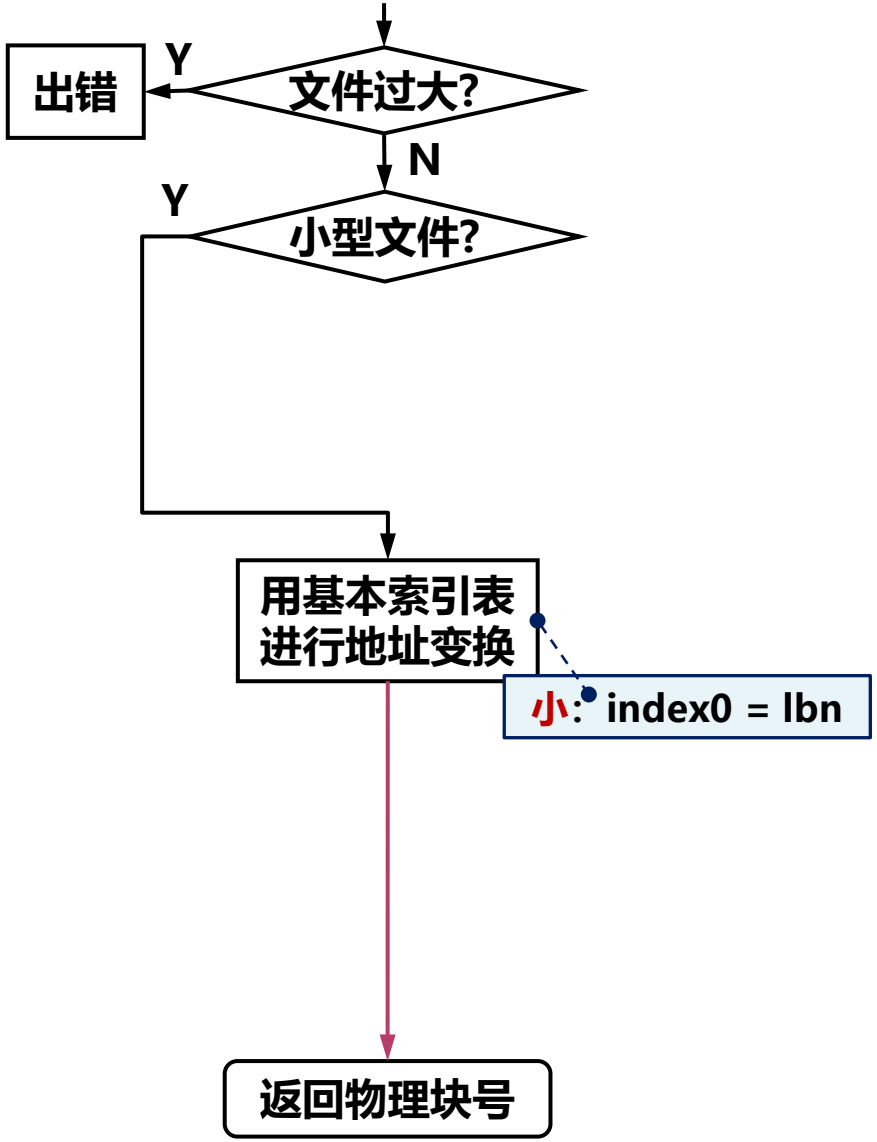
```
class Inode
{
        …;
/* Functions */
public:
        void ReadI();    /* 根据Inode对象中的物理磁盘块索引表，读取文件数据 */
        void WriteI();   /* 根据Inode对象中的物理磁盘块索引表，将数据写入文件 */
        int  Bmap(int lbn);      /* 将文件的逻辑块号转换成对应的物理盘块号 */
        void OpenI(int mode);    /* 打开文件 */
        void CloseI(int mode);   /* 关闭文件 */
        void IUpdate(int time);  /* 更新外存Inode的最后的访问时间、修改时间 */
        void ITrunc();           /* 释放Inode对应文件占用的磁盘块 */
        void Clean();                    /* 清空Inode对象中的数据 */
        void ICopy(Buf* bp, int inumber); /* 将外存Inode信息拷贝到内存Inode中 */
```

UNIX文件系统的读写操作

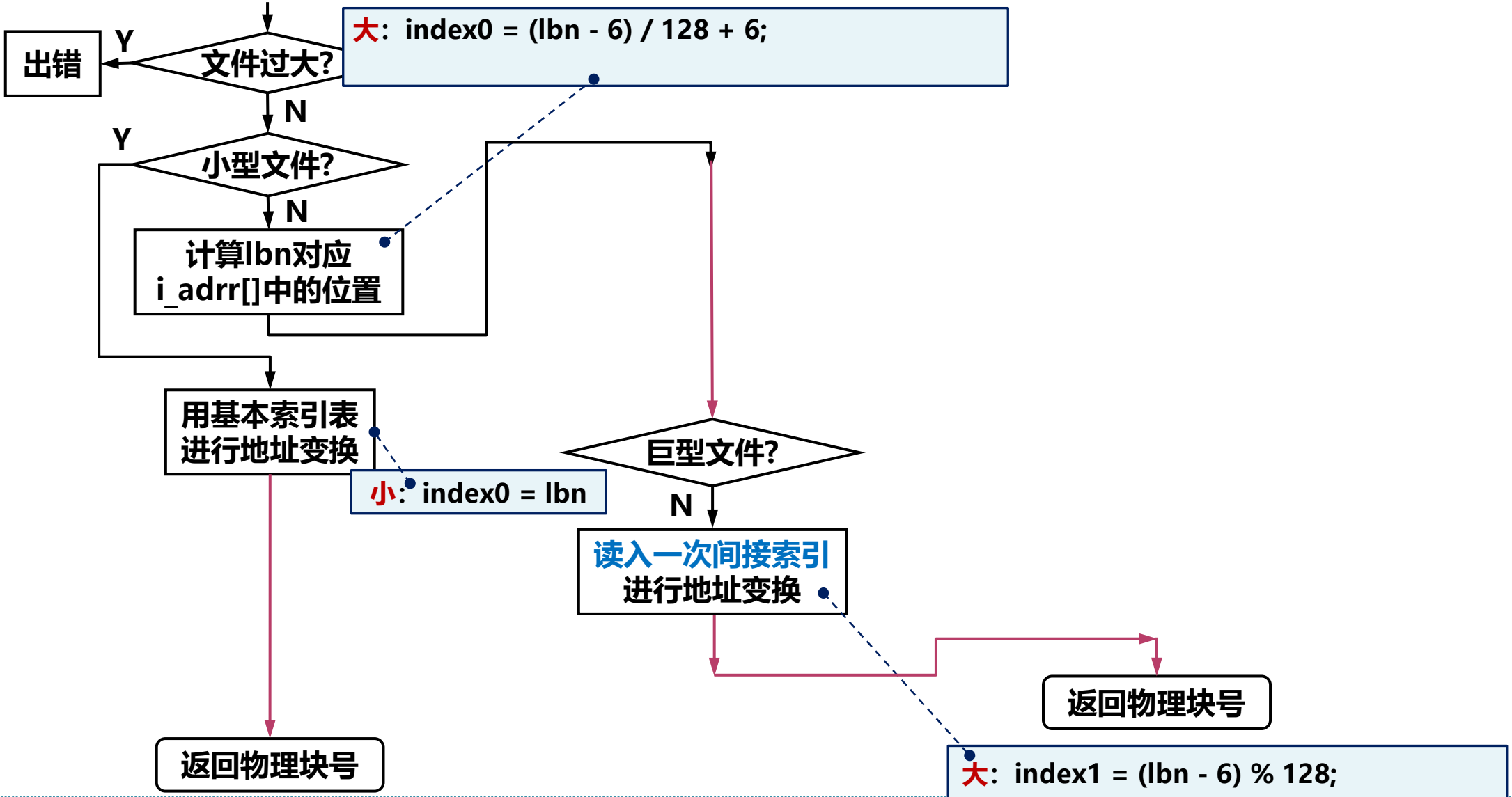
读操作



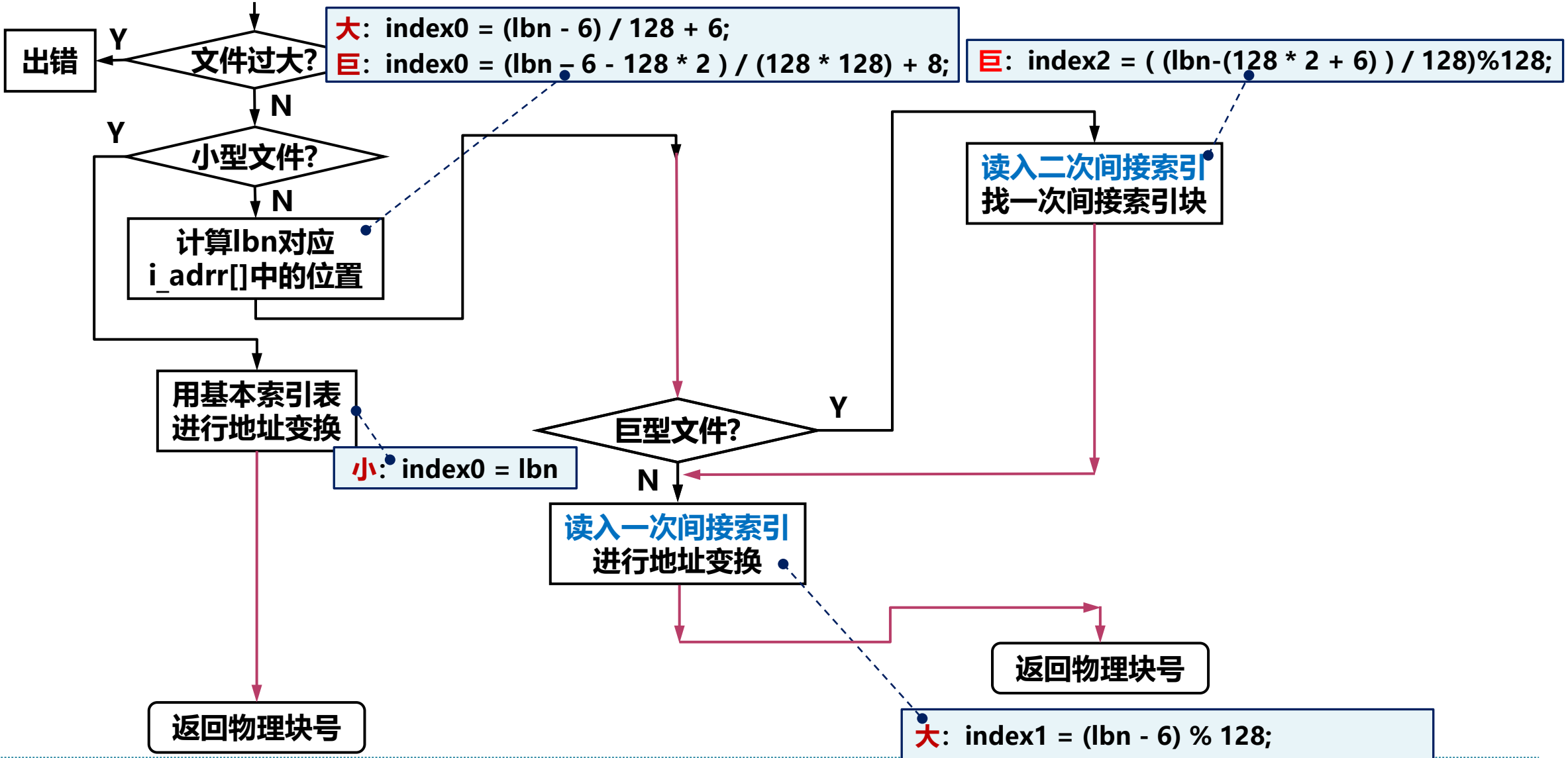
UNIX文件系统的静态结构



UNIX文件系统的静态结构



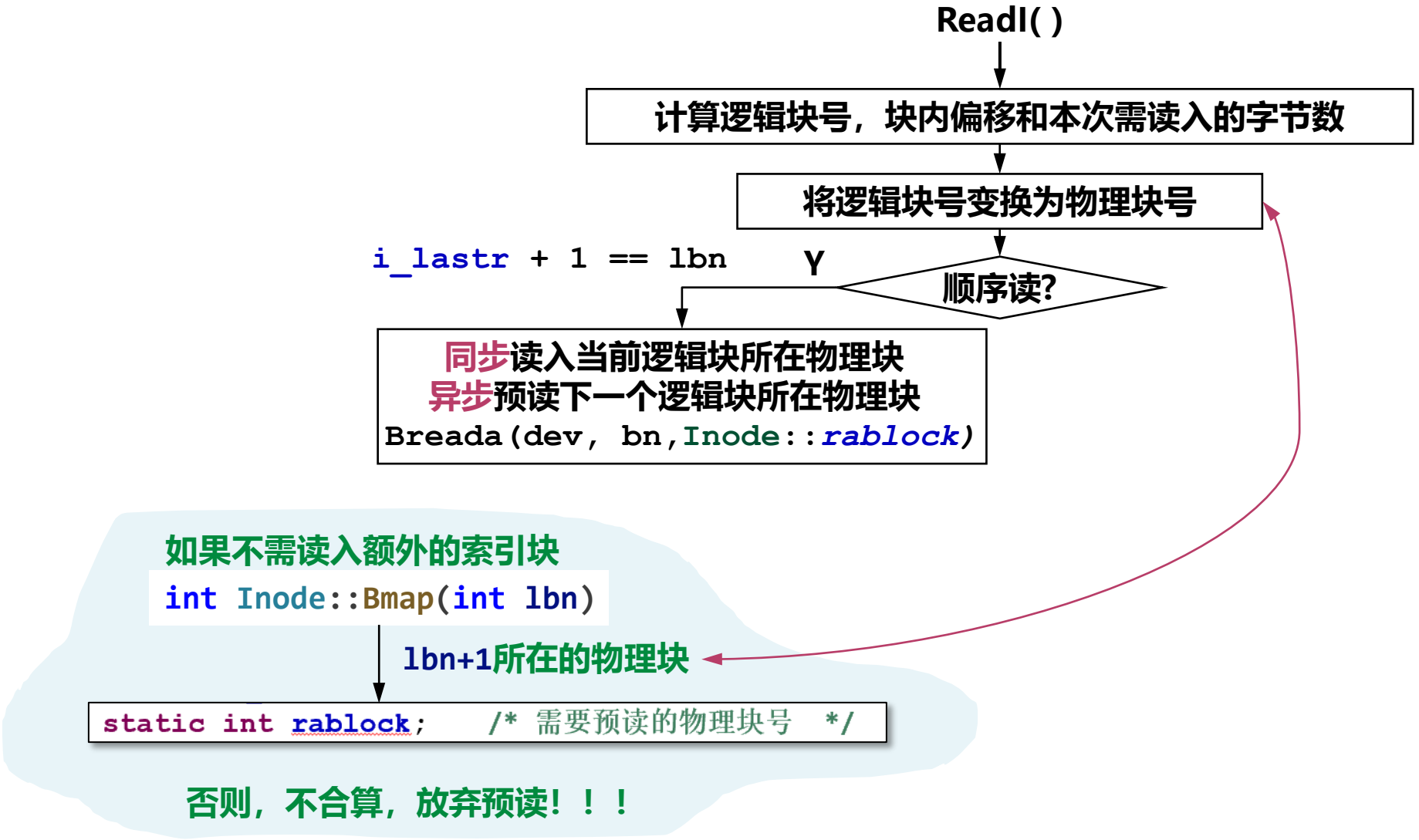
UNIX文件系统的静态结构



索引结构

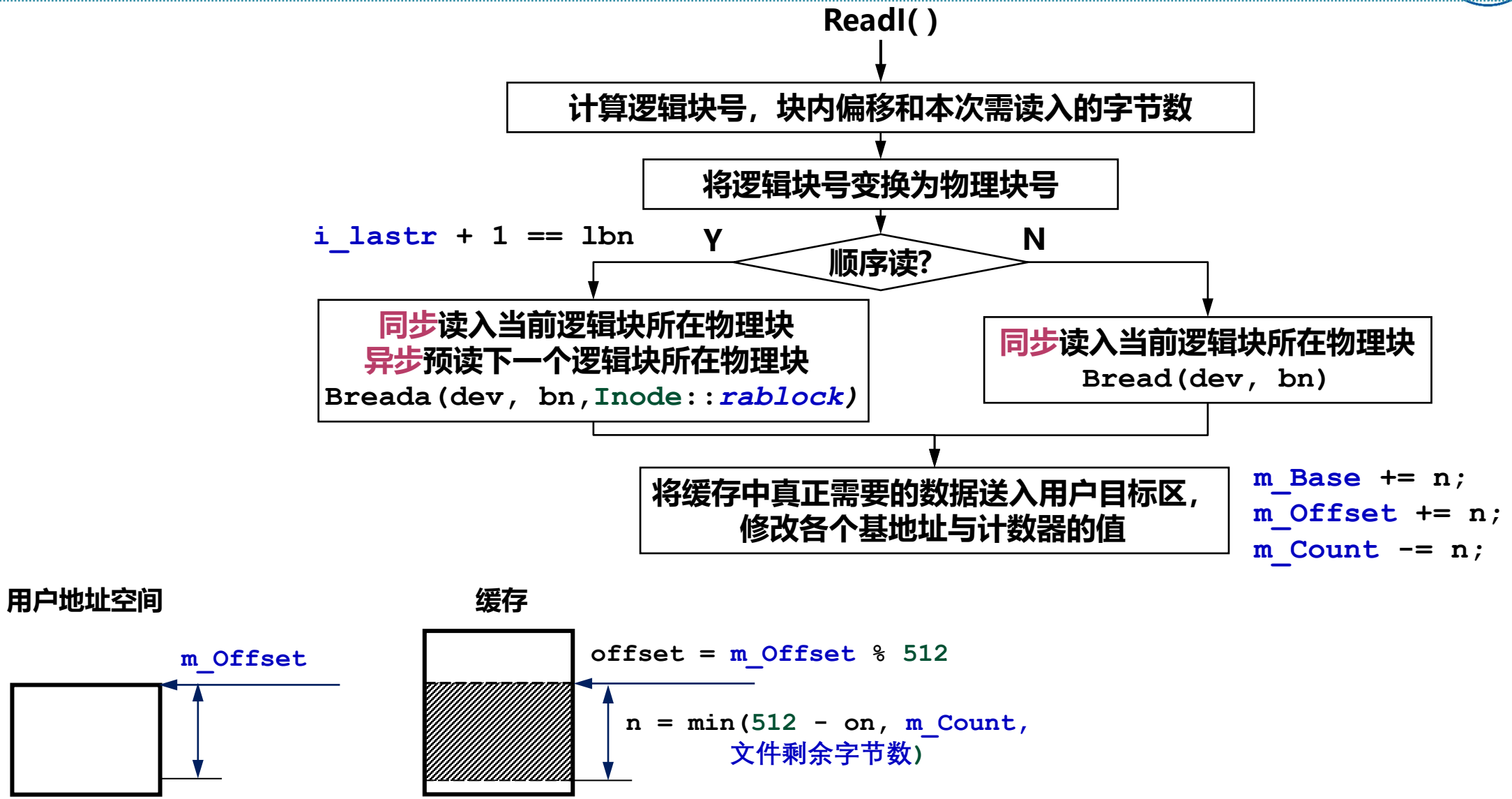
UNIX文件系统的读写操作

读操作



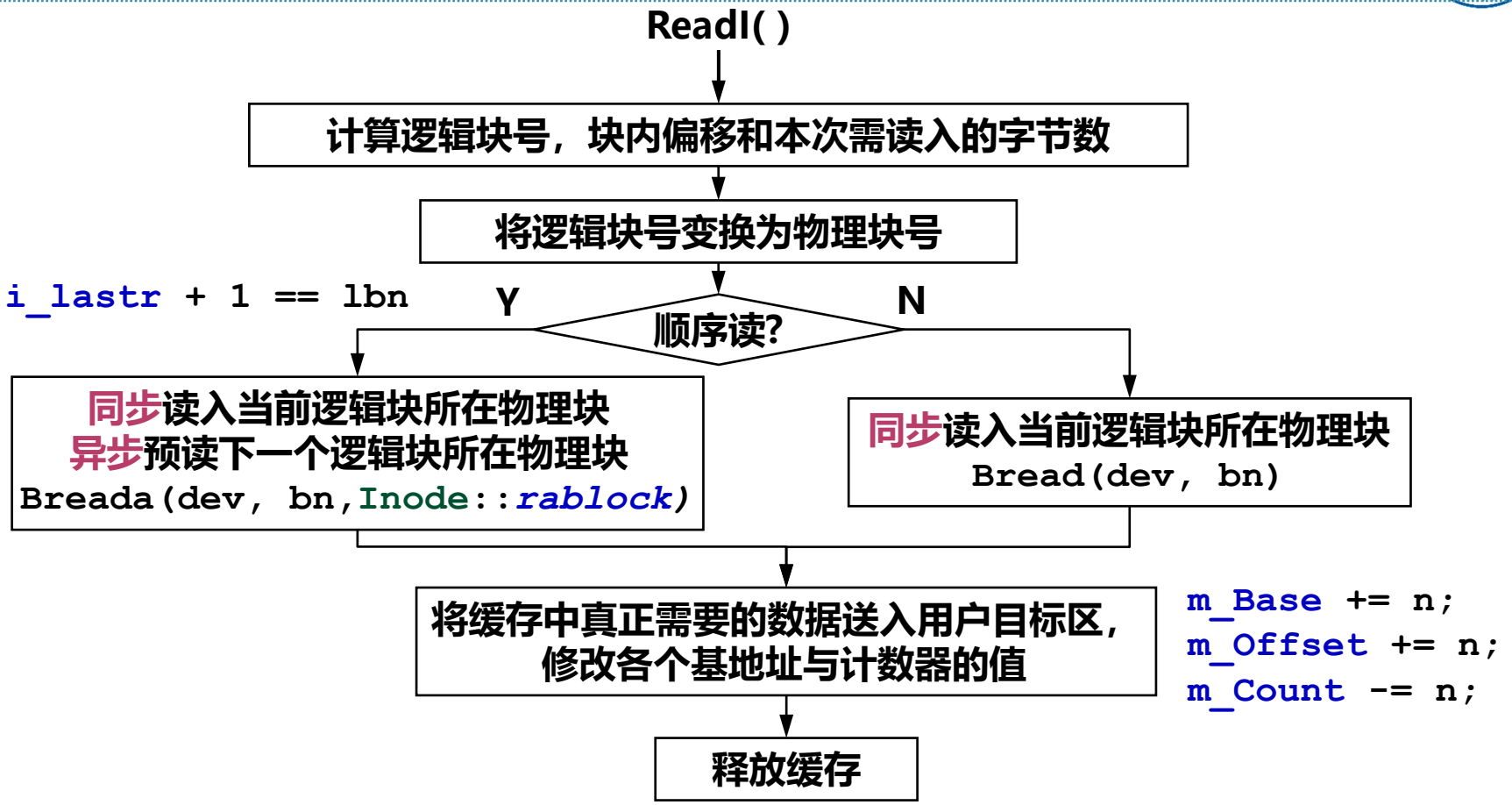
UNIX文件系统的读写操作

读操作



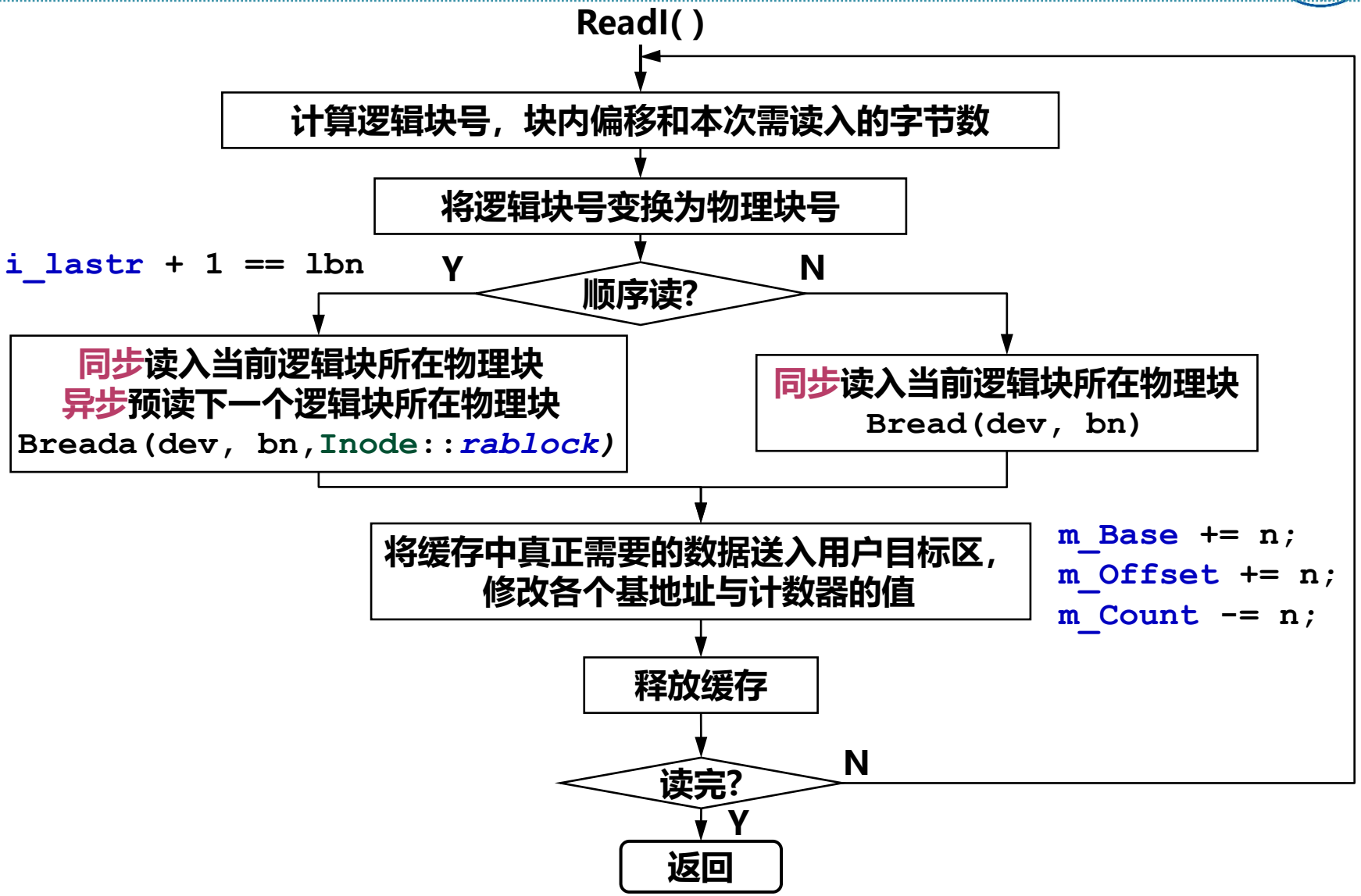
UNIX文件系统的读写操作

读操作



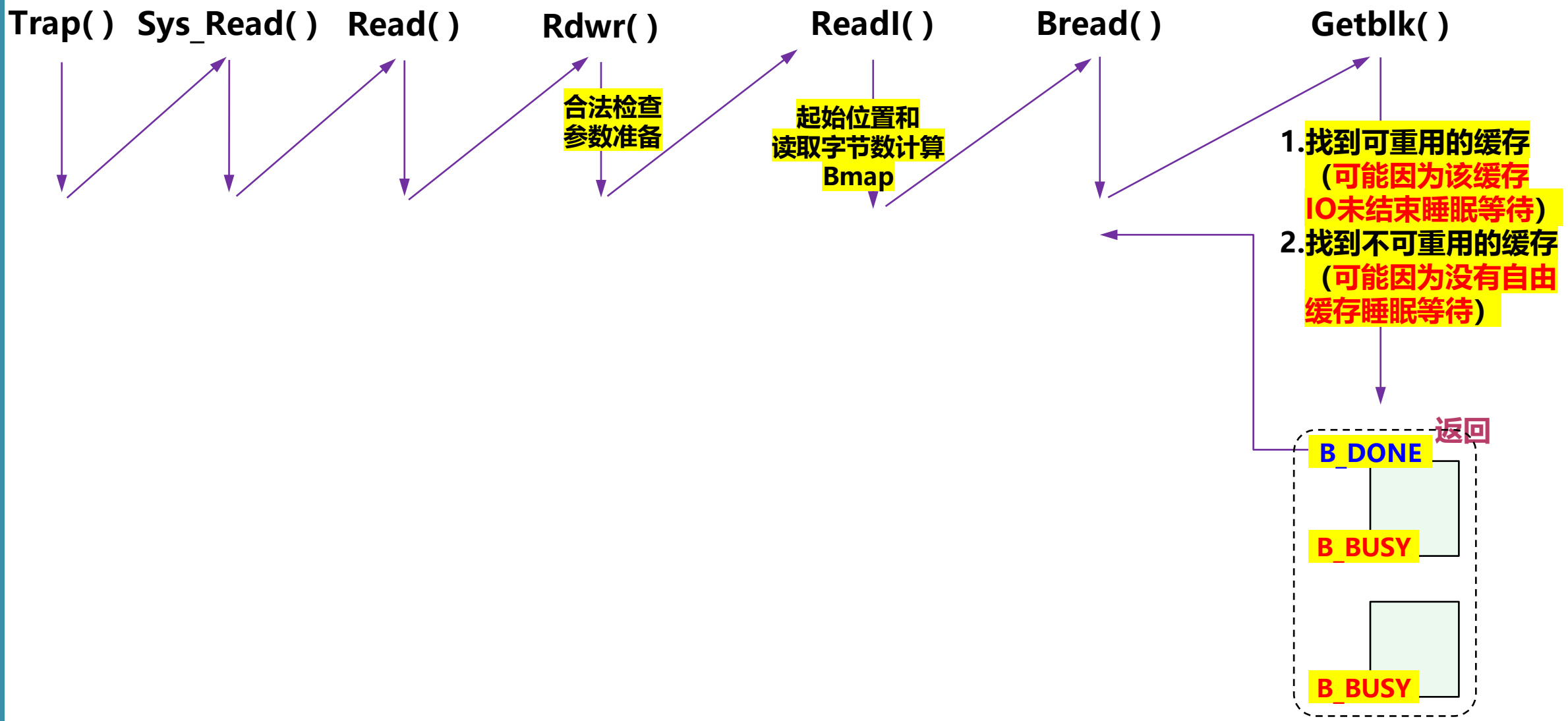
UNIX文件系统的读写操作

读操作



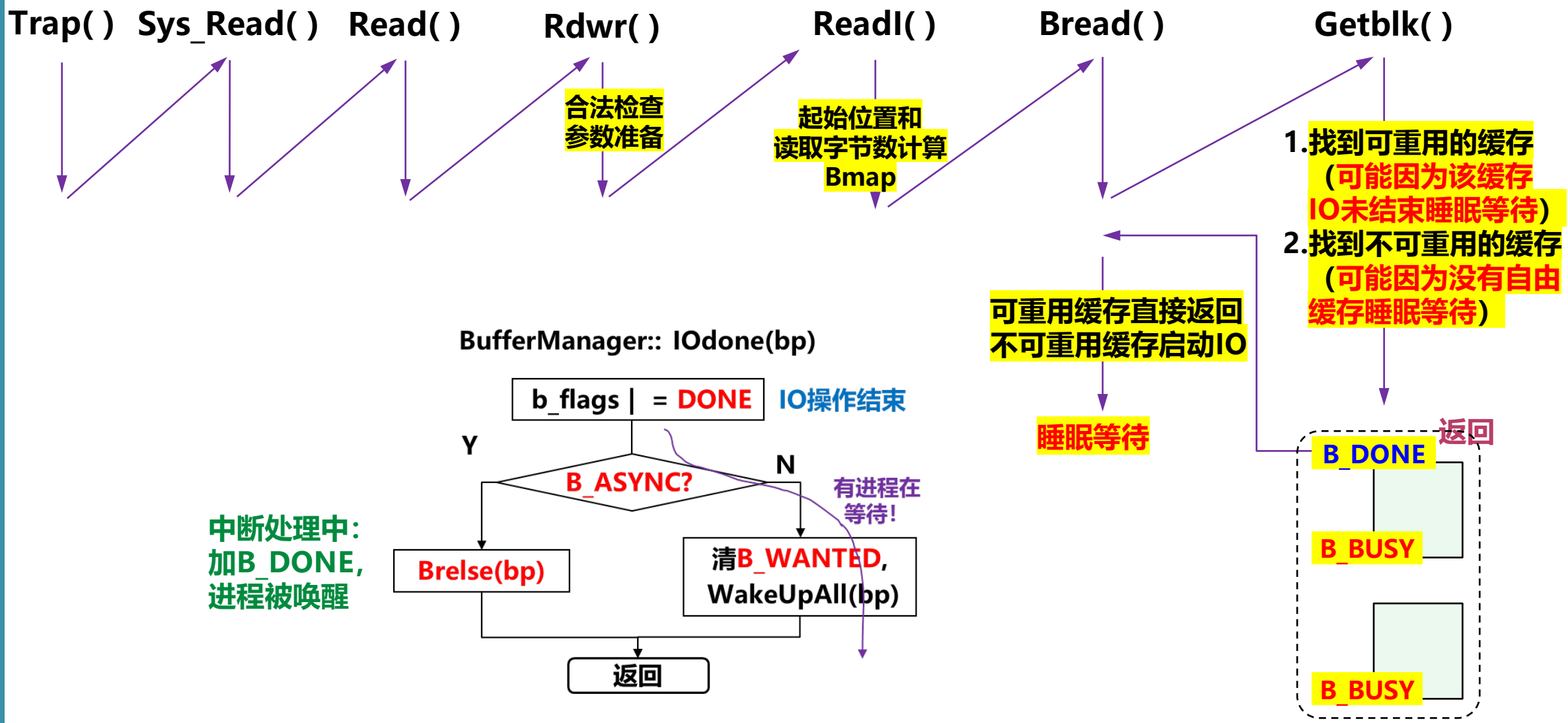
UNIX文件系统的读写操作

读操作



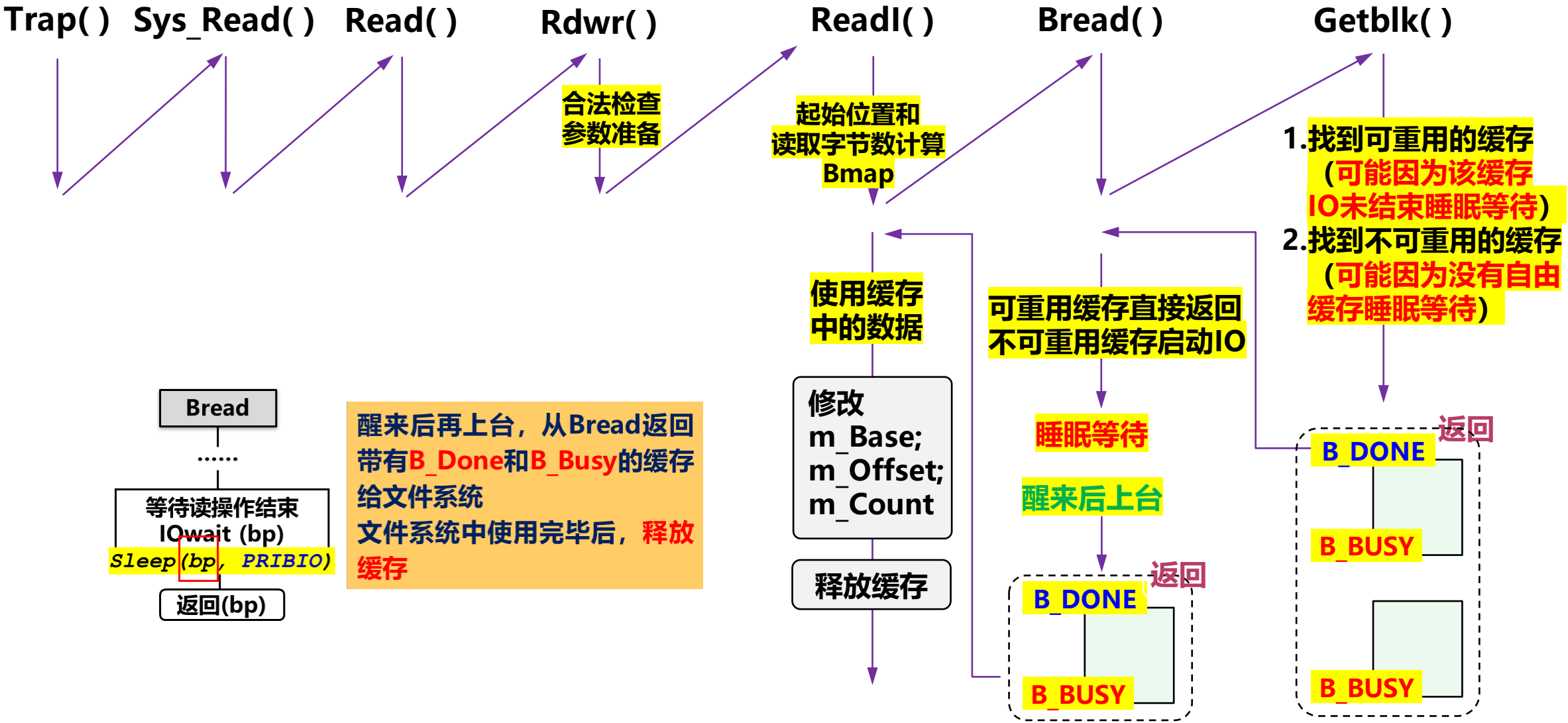
UNIX文件系统的读写操作

读操作



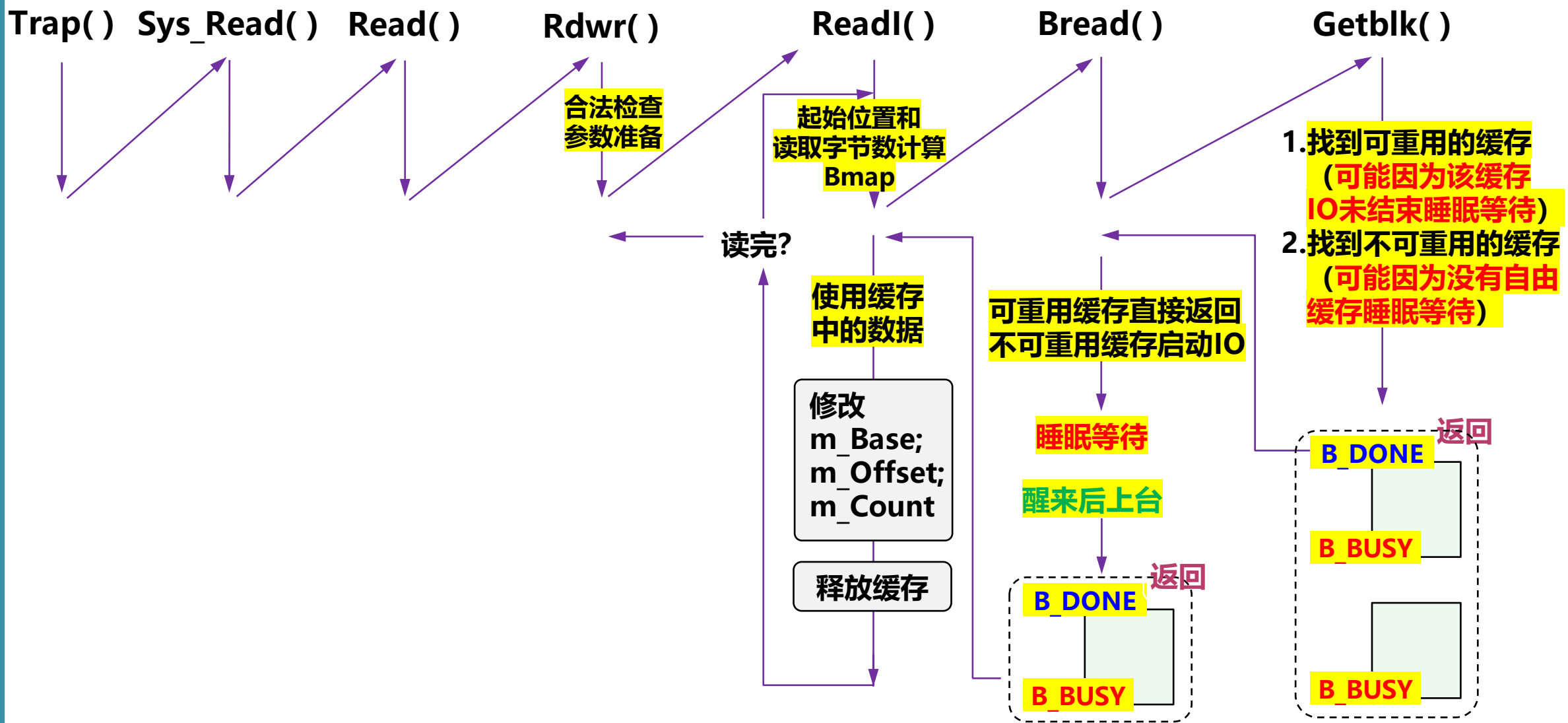
UNIX文件系统的读写操作

读操作



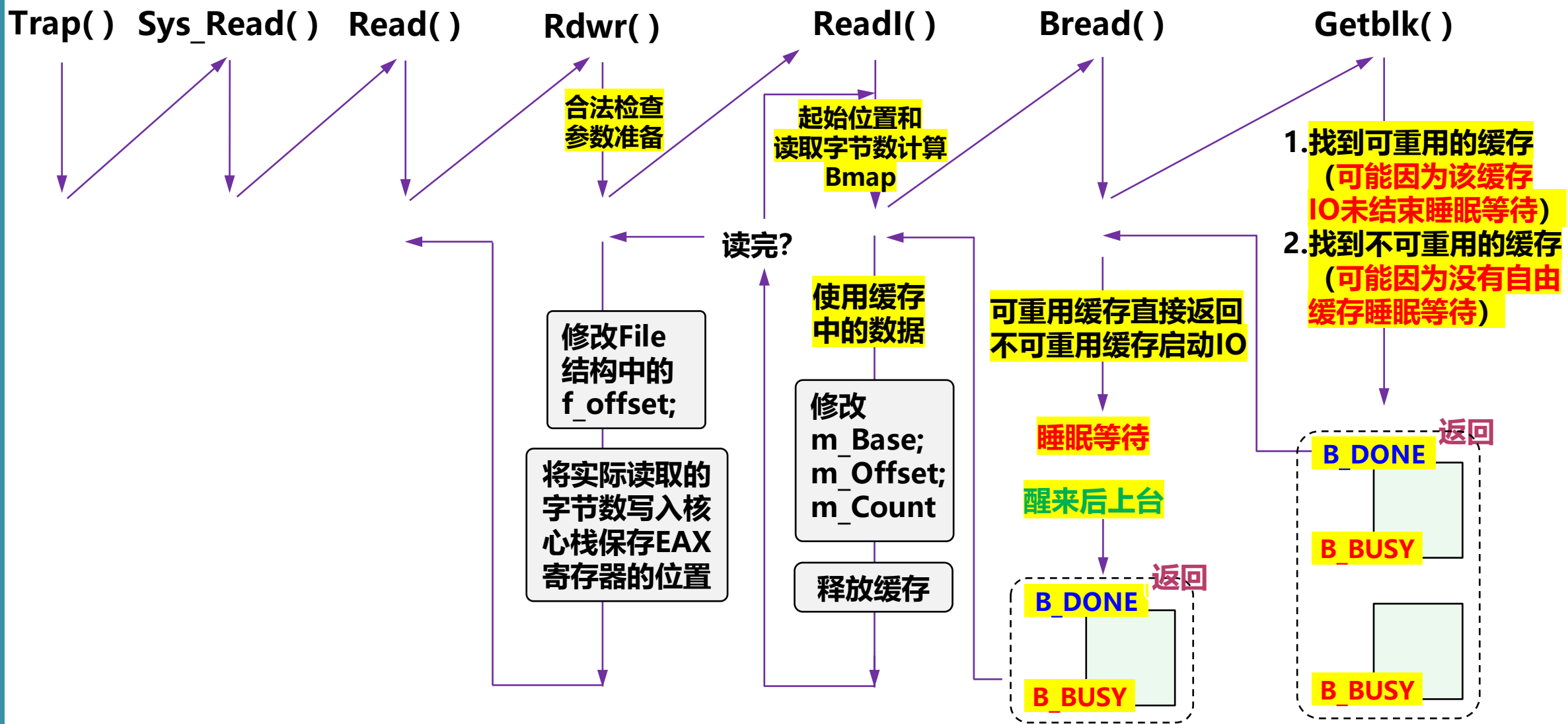
UNIX文件系统的读写操作

读操作

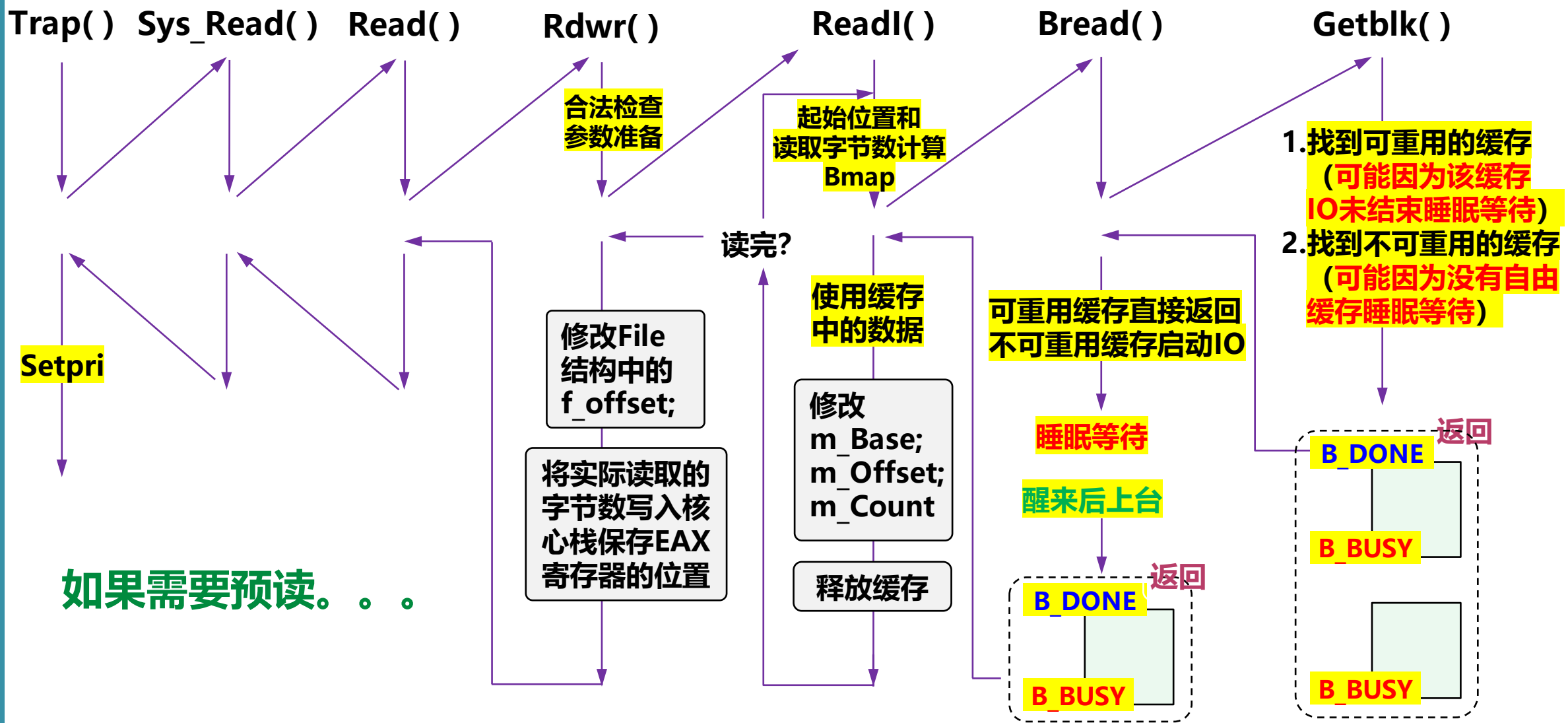


UNIX文件系统的读写操作

读操作



UNIX文件系统的读写操作

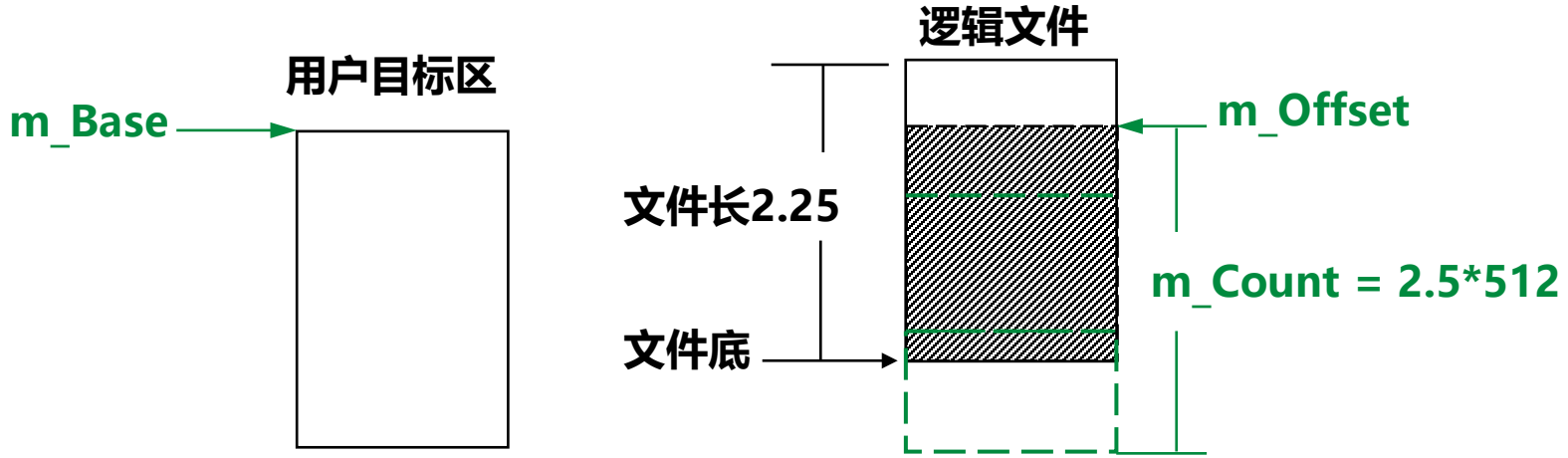


读操作

UNIX文件系统的读写操作

读操作

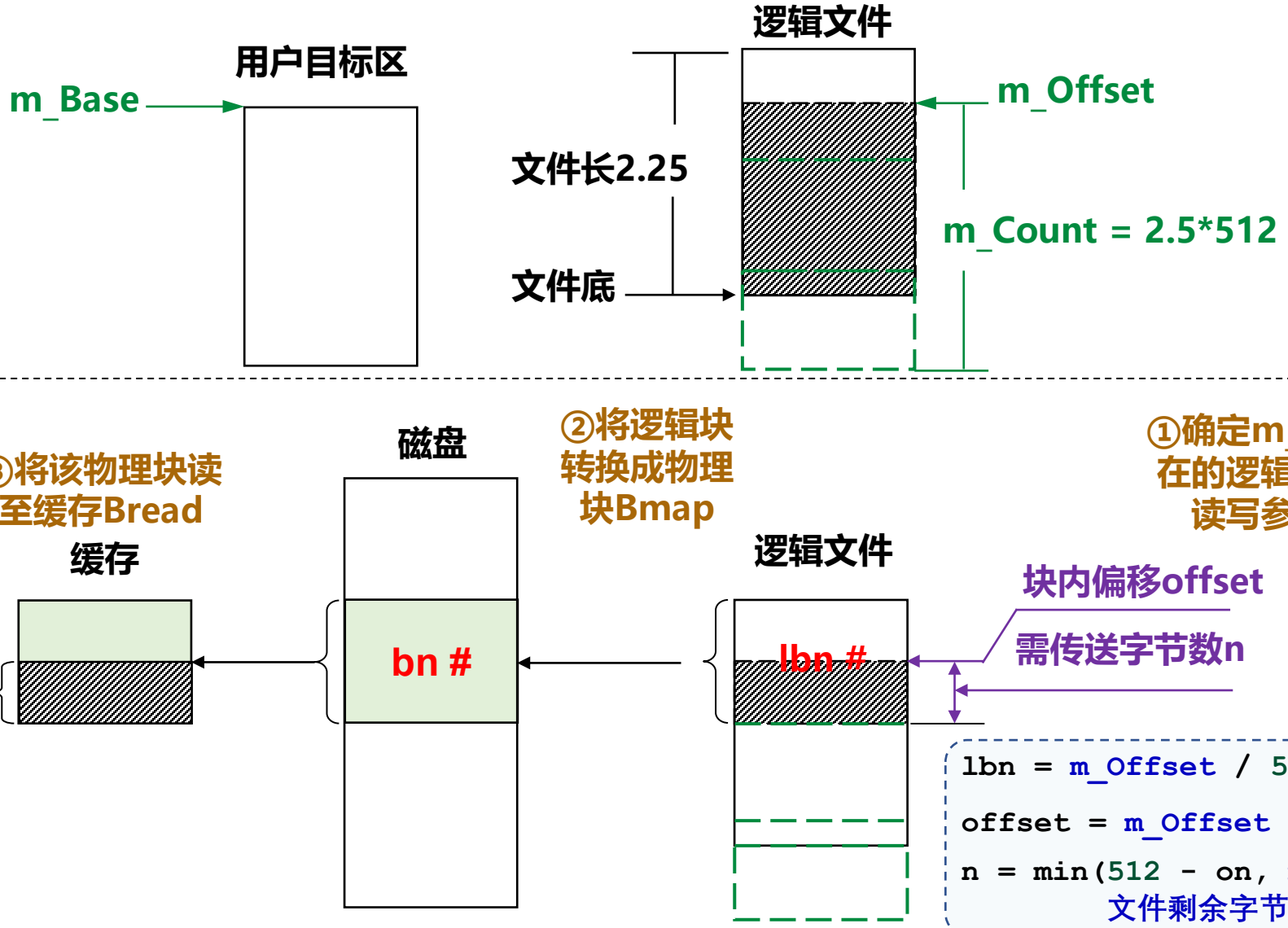
从文件读2.5块到进程空间的过程



UNIX文件系统的读写操作

读操作

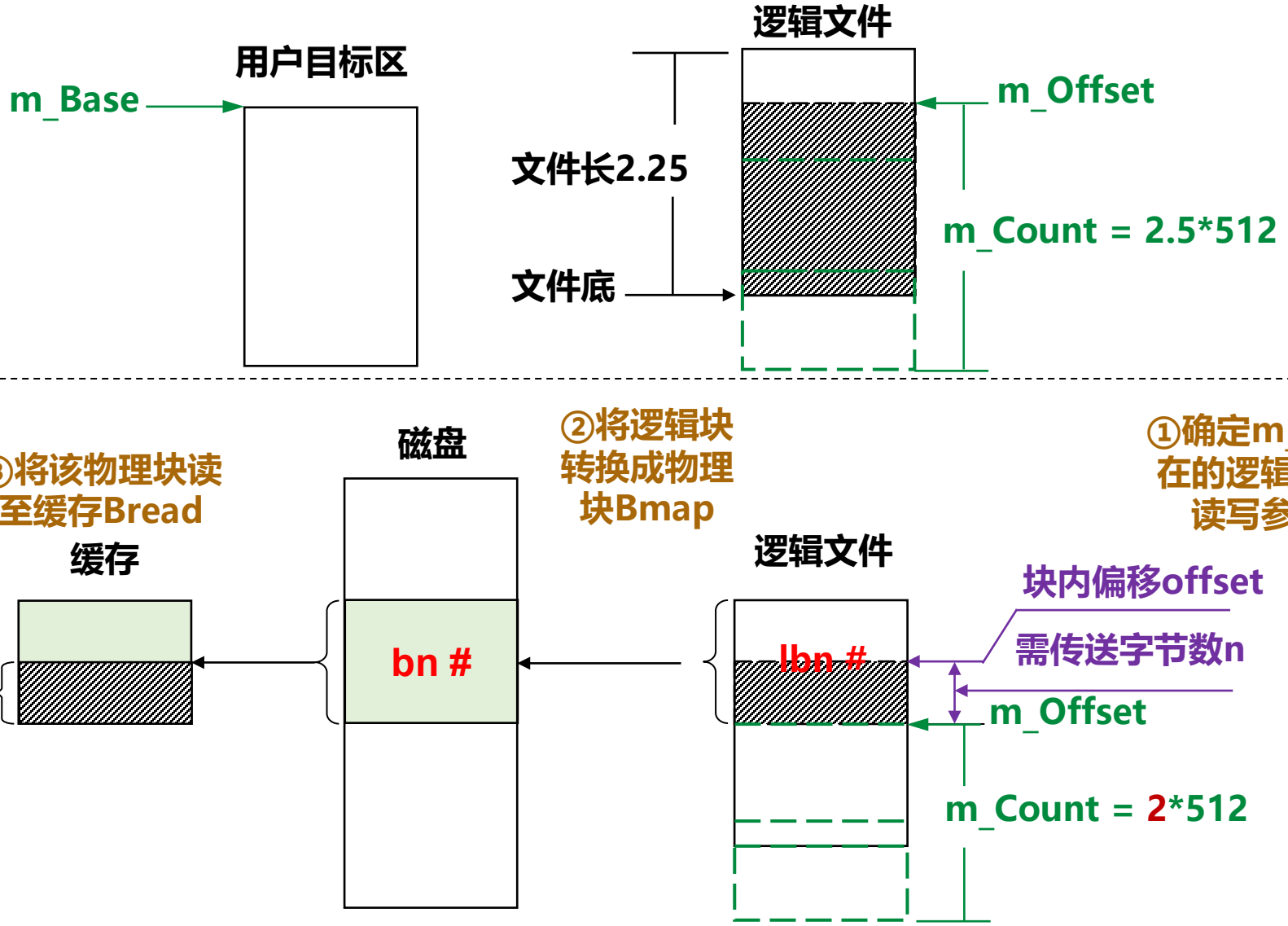
从文件读2.5块到进程空间的过程



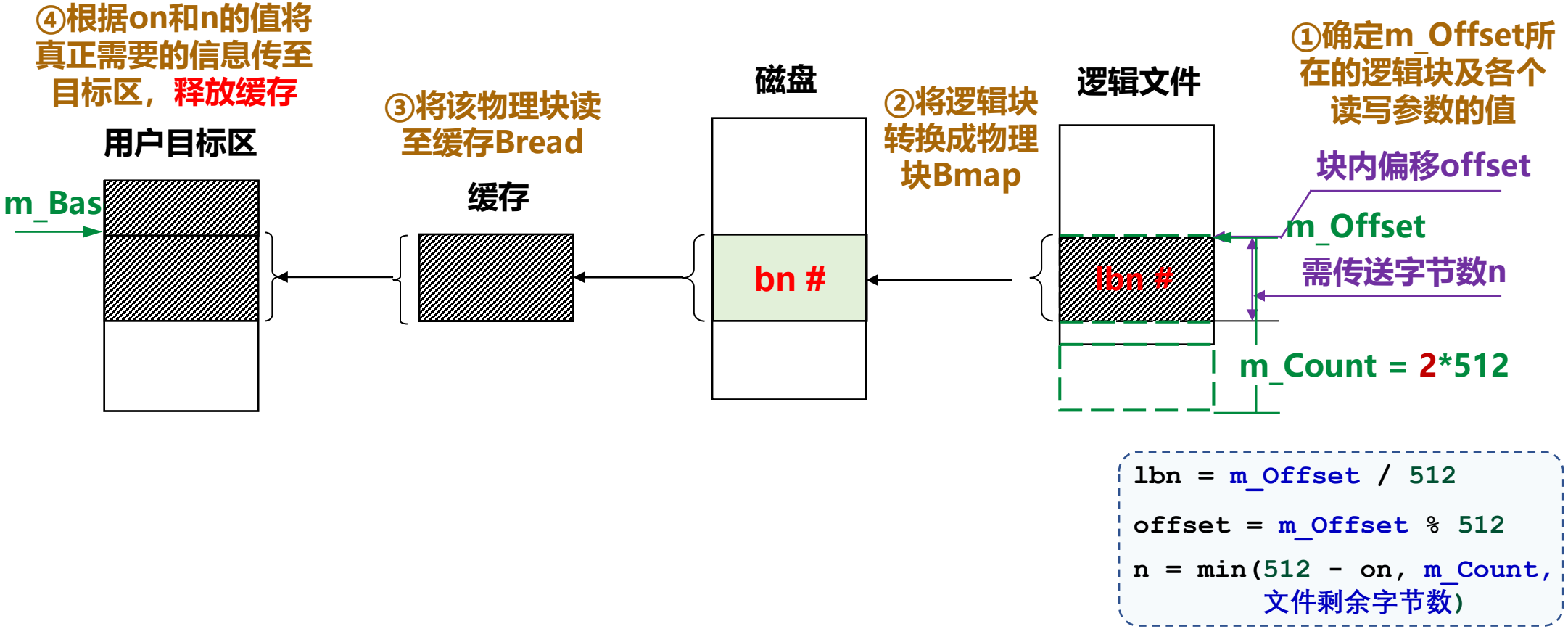
UNIX文件系统的读写操作

读操作

从文件读2.5块到进程空间的过程



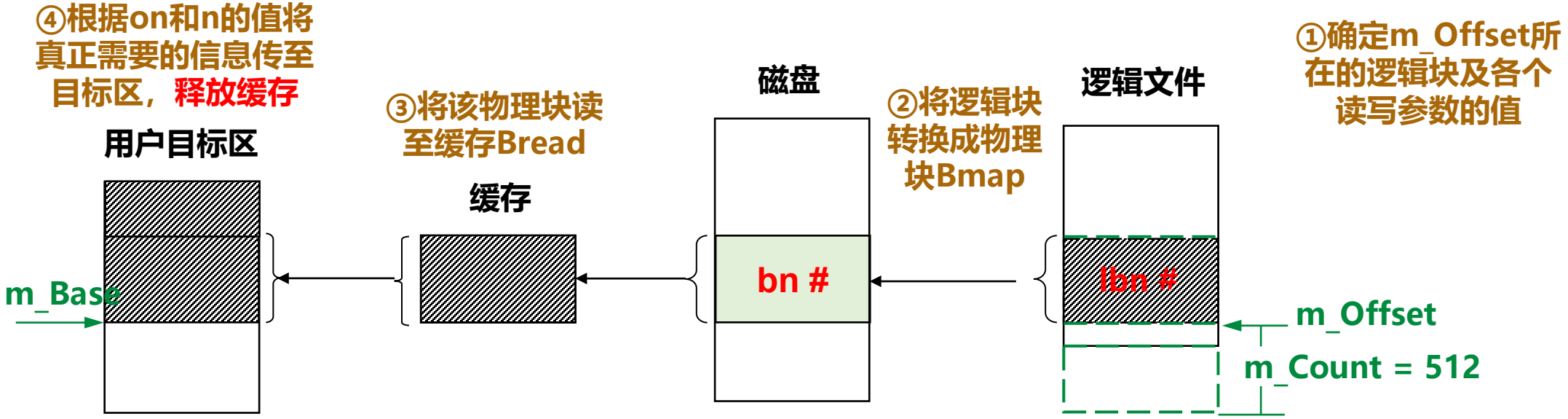
UNIX文件系统的读写操作




```
lbn = m_Offset / 512
offset = m_Offset % 512
n = min(512 - on, m_Count,
        文件剩余字节数)
```

读操作

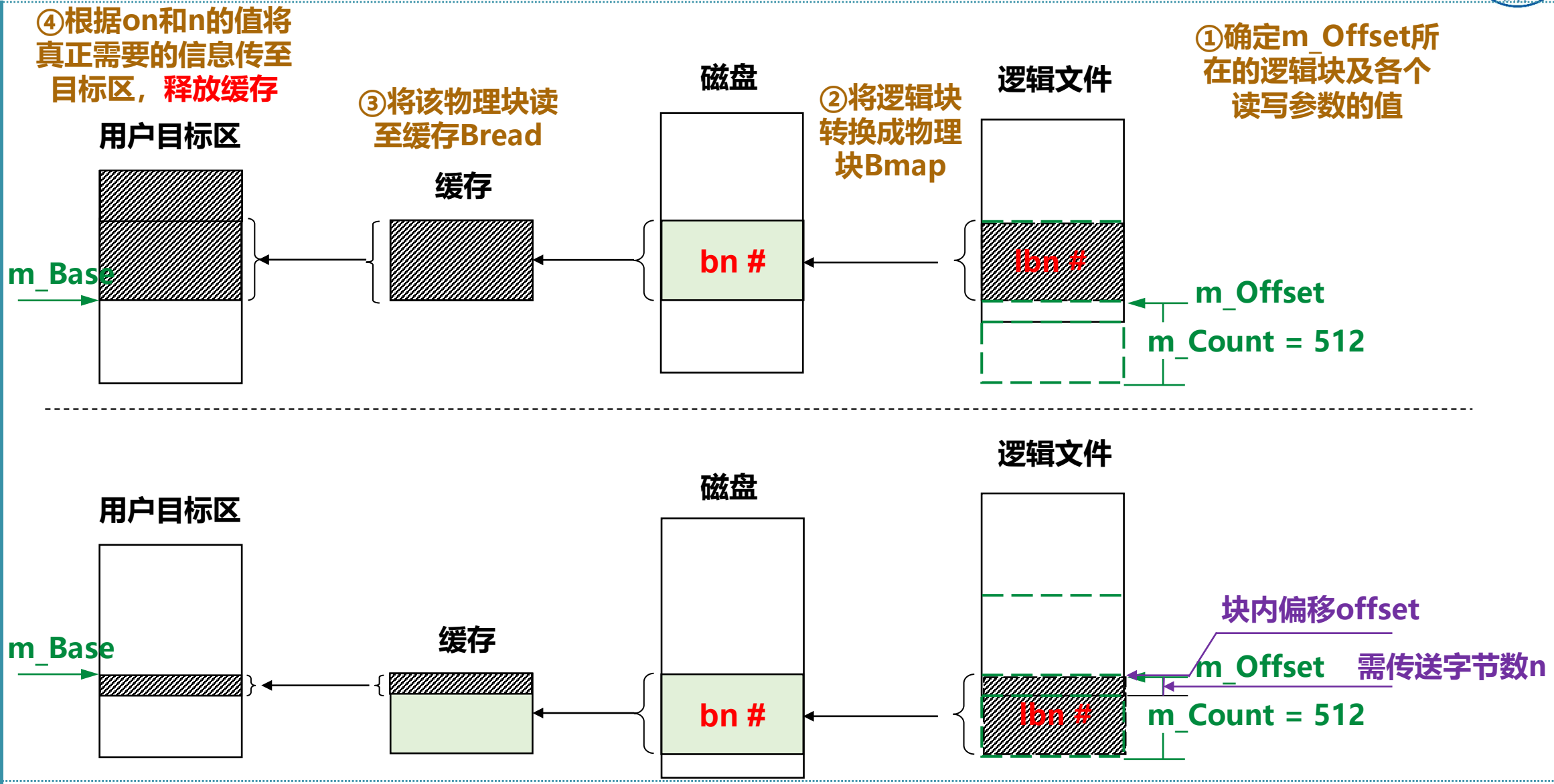
UNIX文件系统的读写操作



读操作

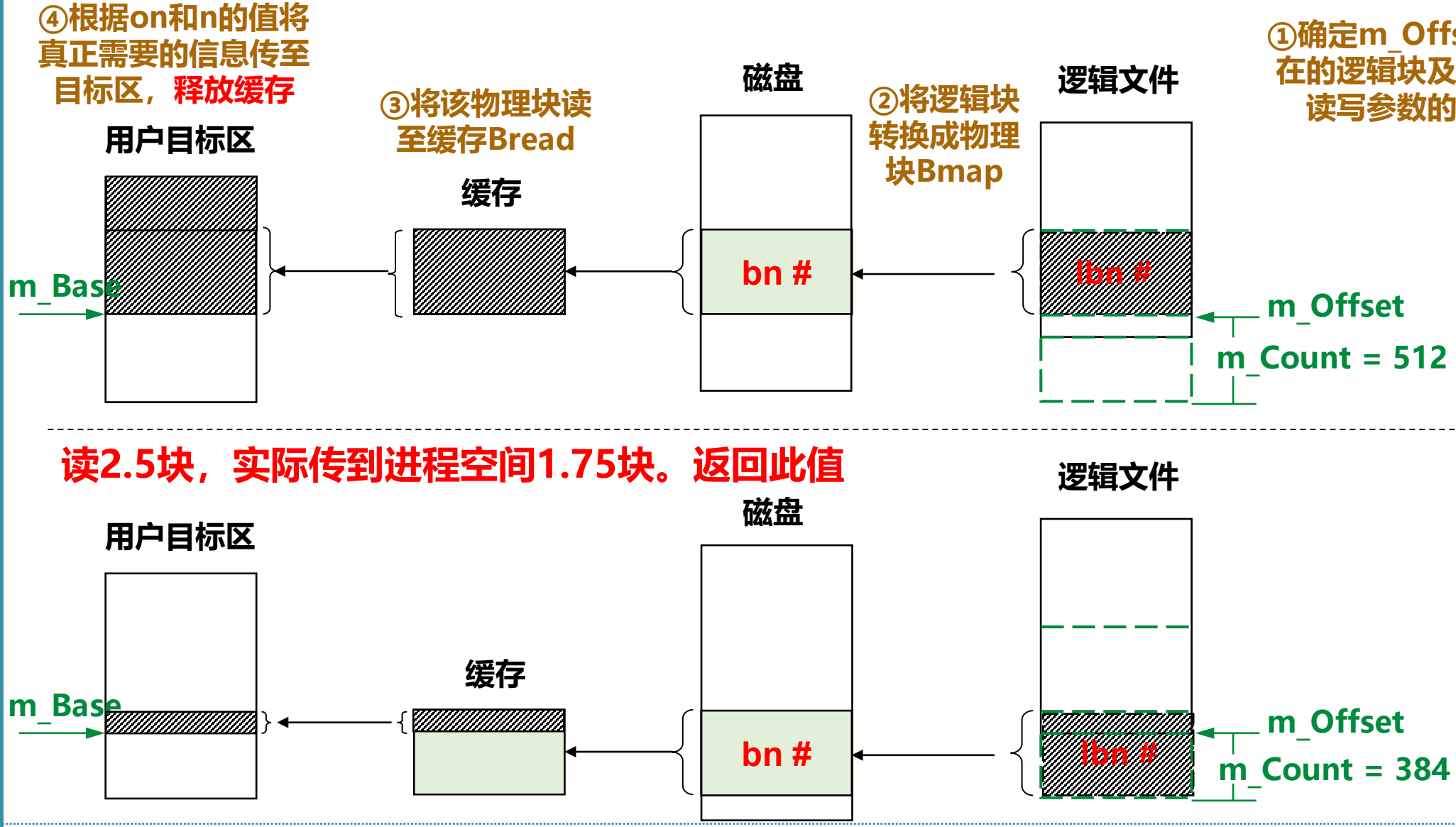
UNIX文件系统的读写操作

读操作



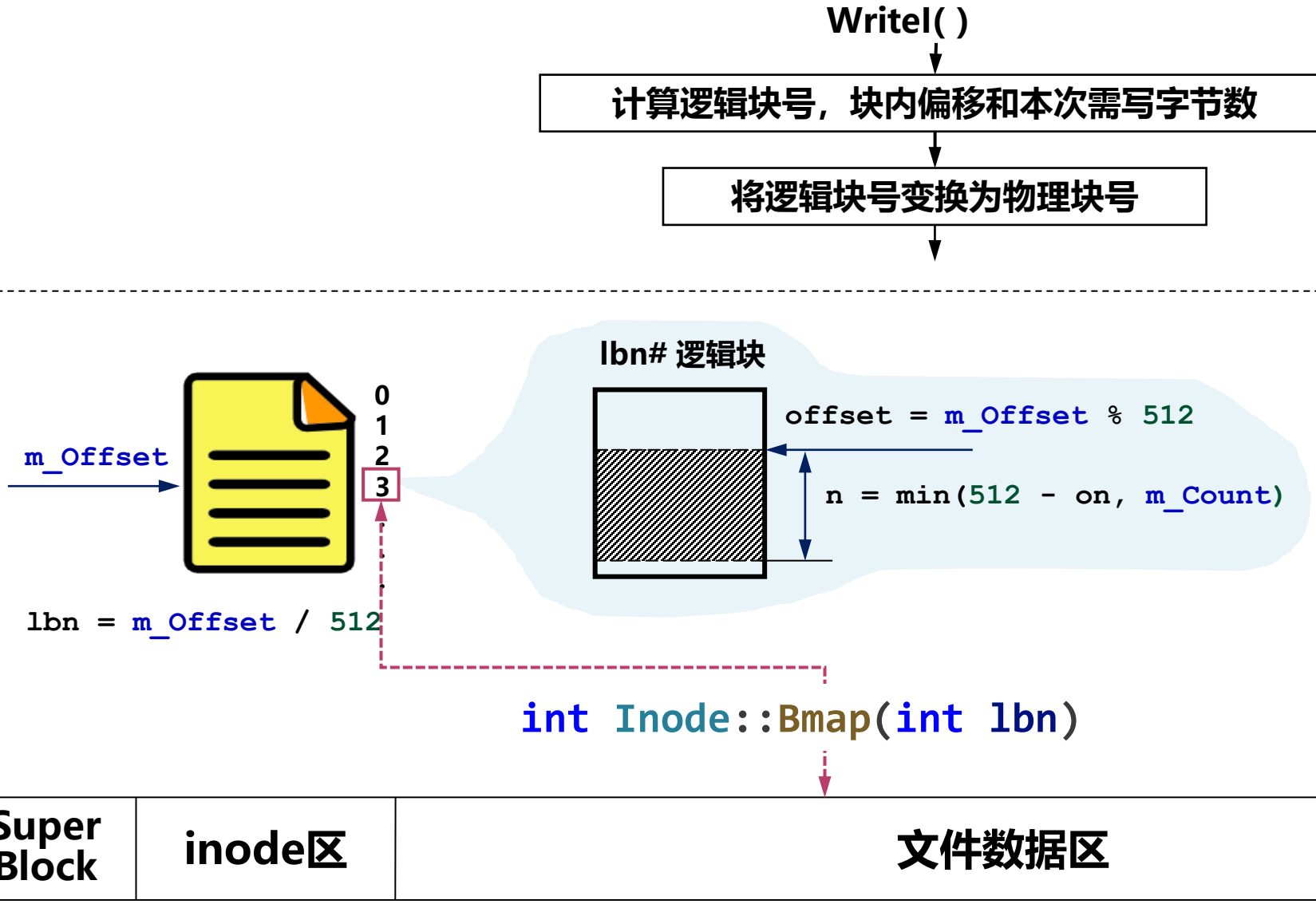
UNIX文件系统的读写操作

读操作



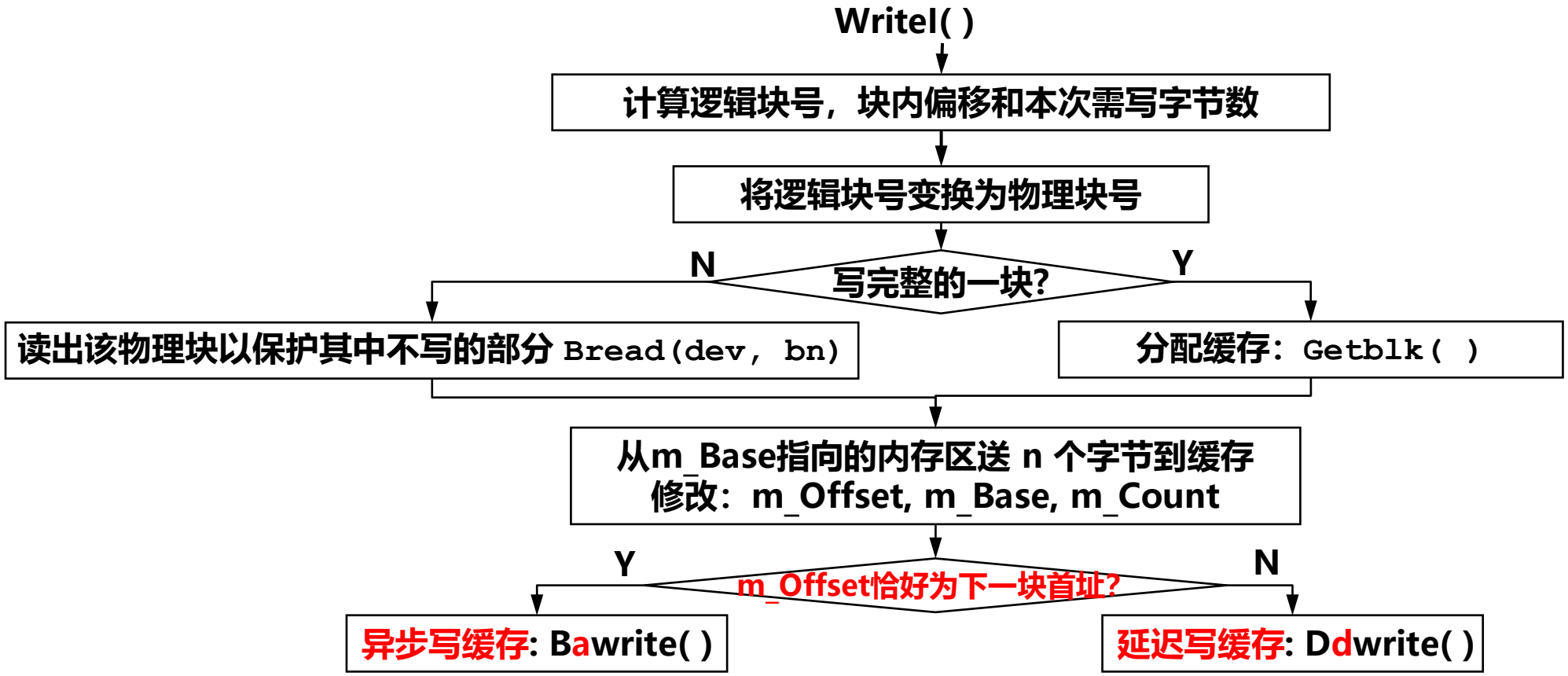
UNIX文件系统的读写操作

写操作



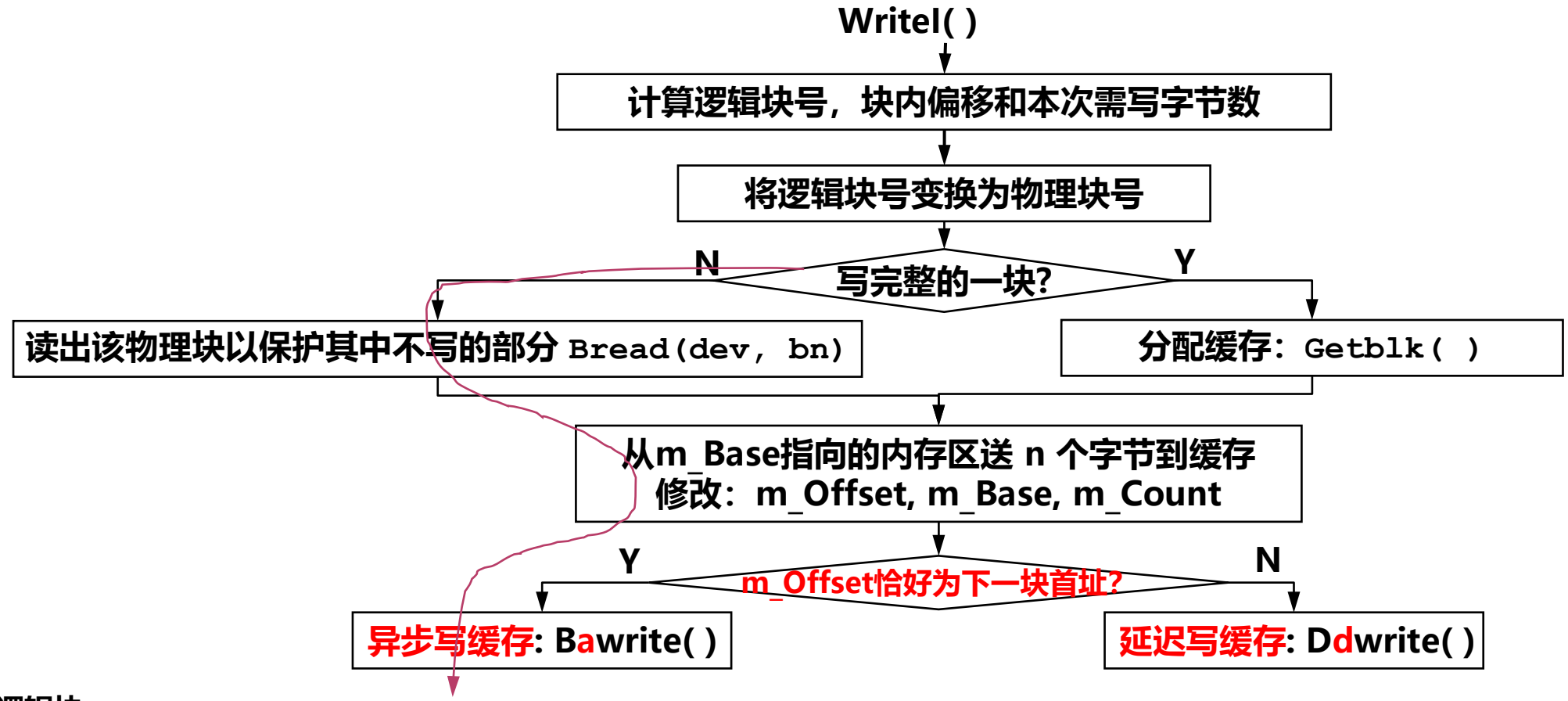
UNIX文件系统的读写操作

写操作

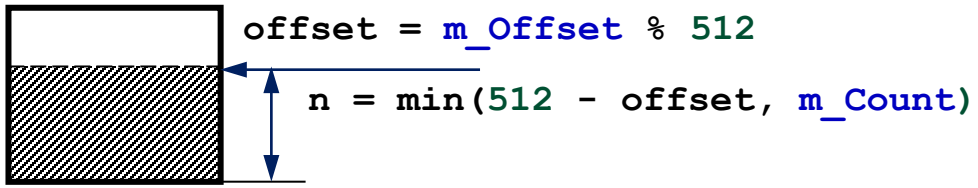


UNIX文件系统的读写操作

写操作

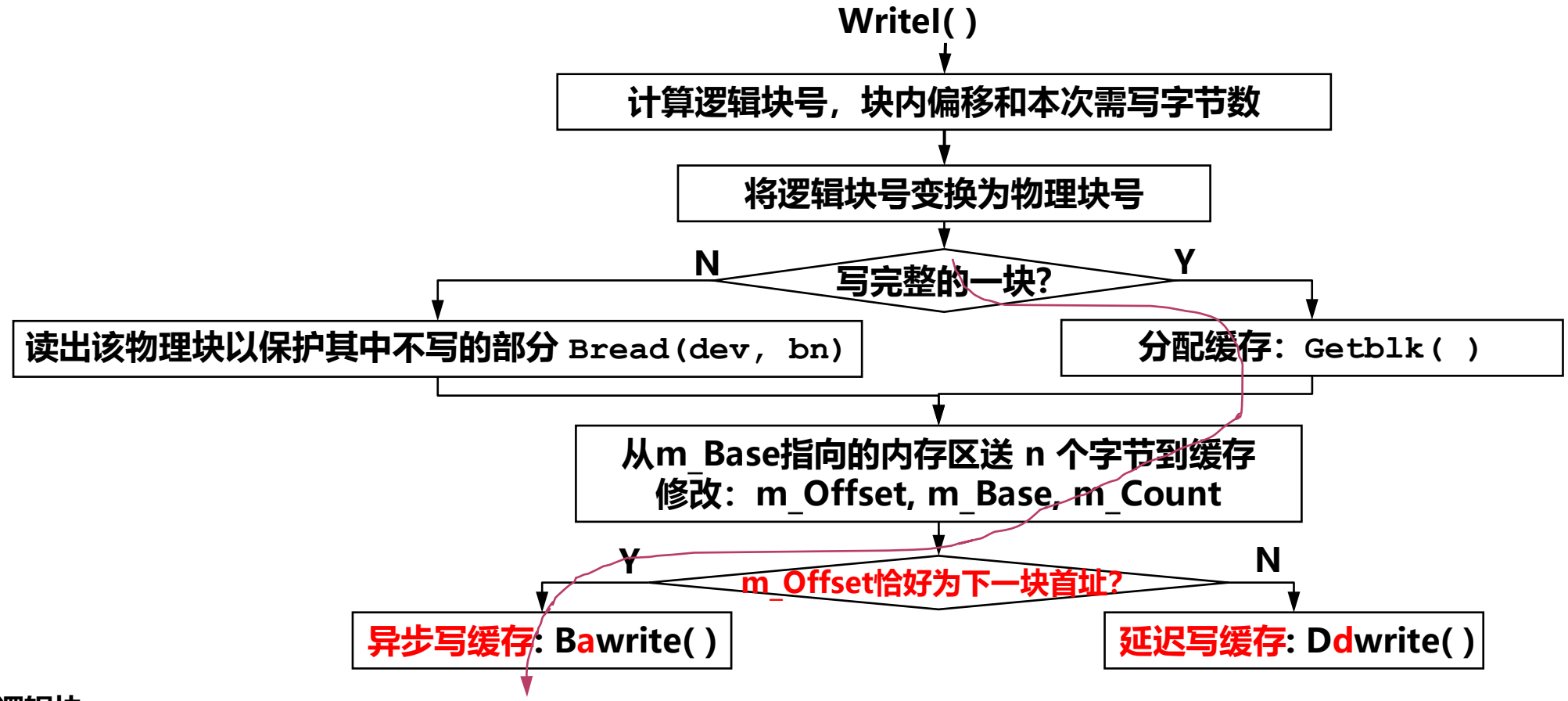


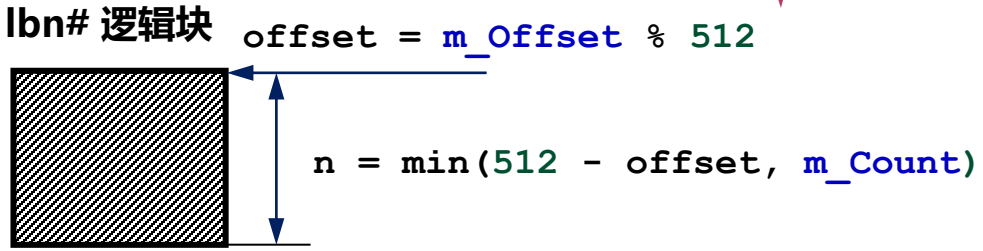
lbn# 逻辑块



UNIX文件系统的读写操作

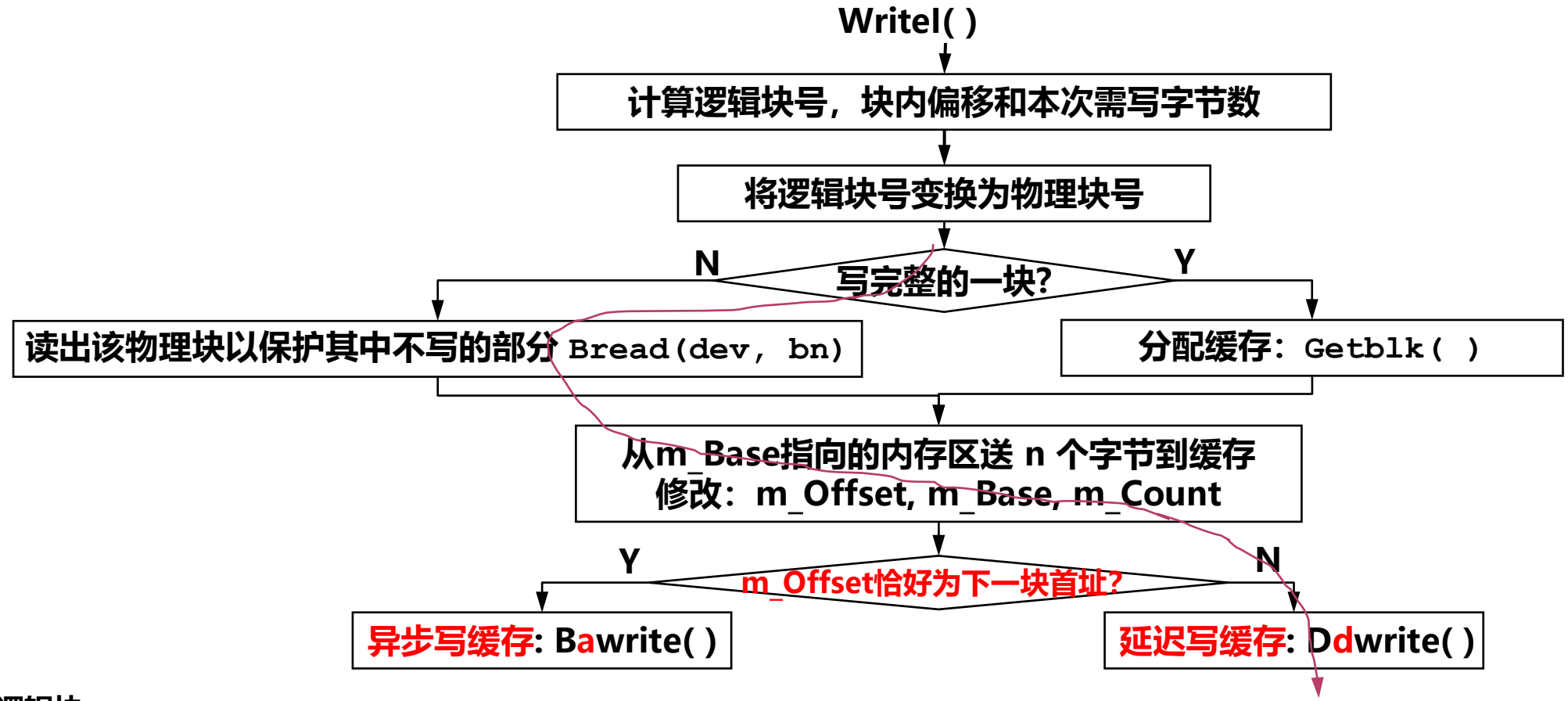
写操作



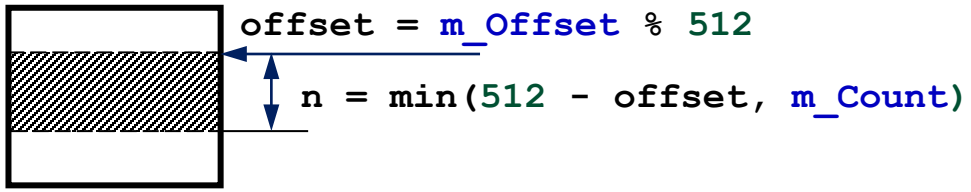


UNIX文件系统的读写操作

写操作

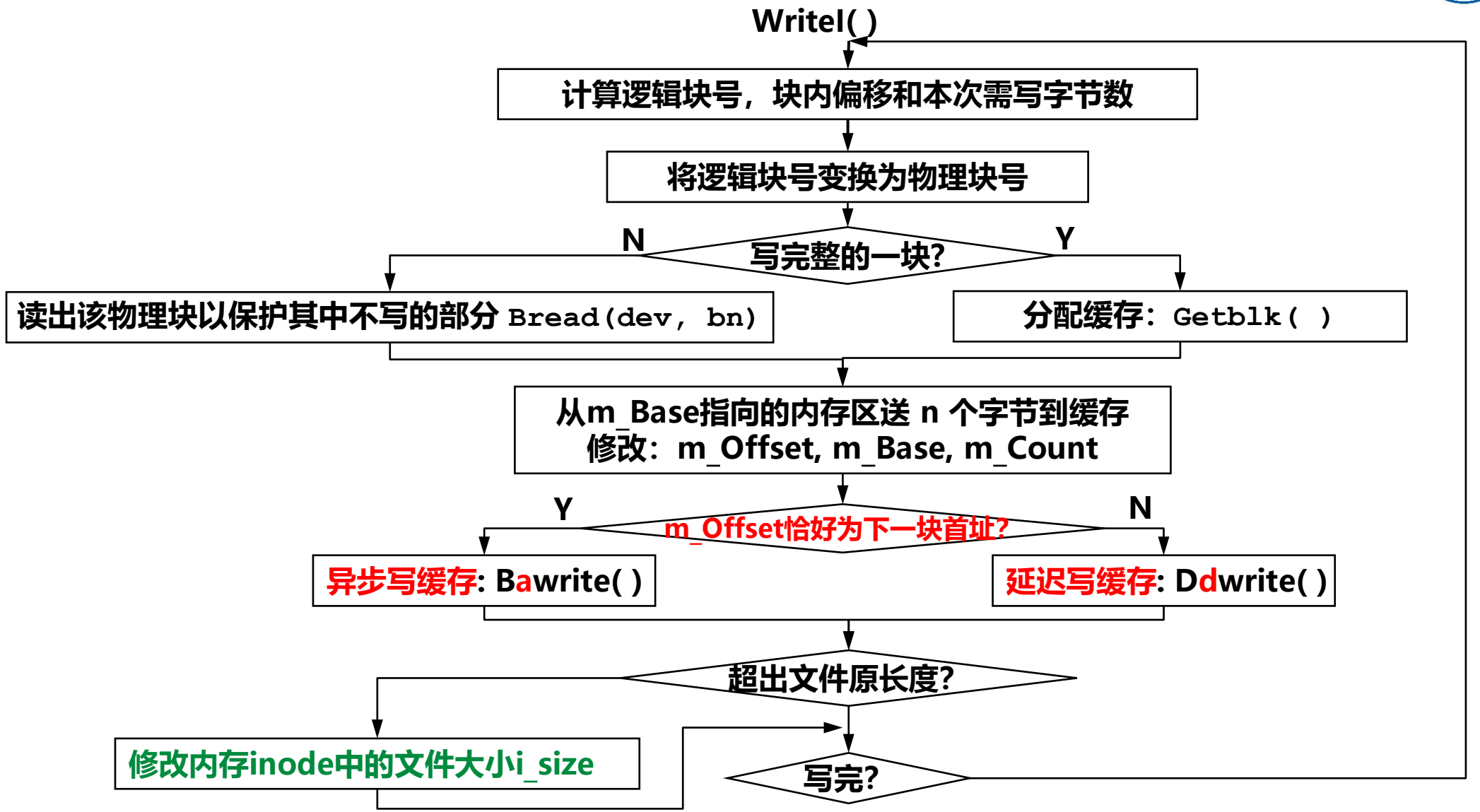


lbn# 逻辑块



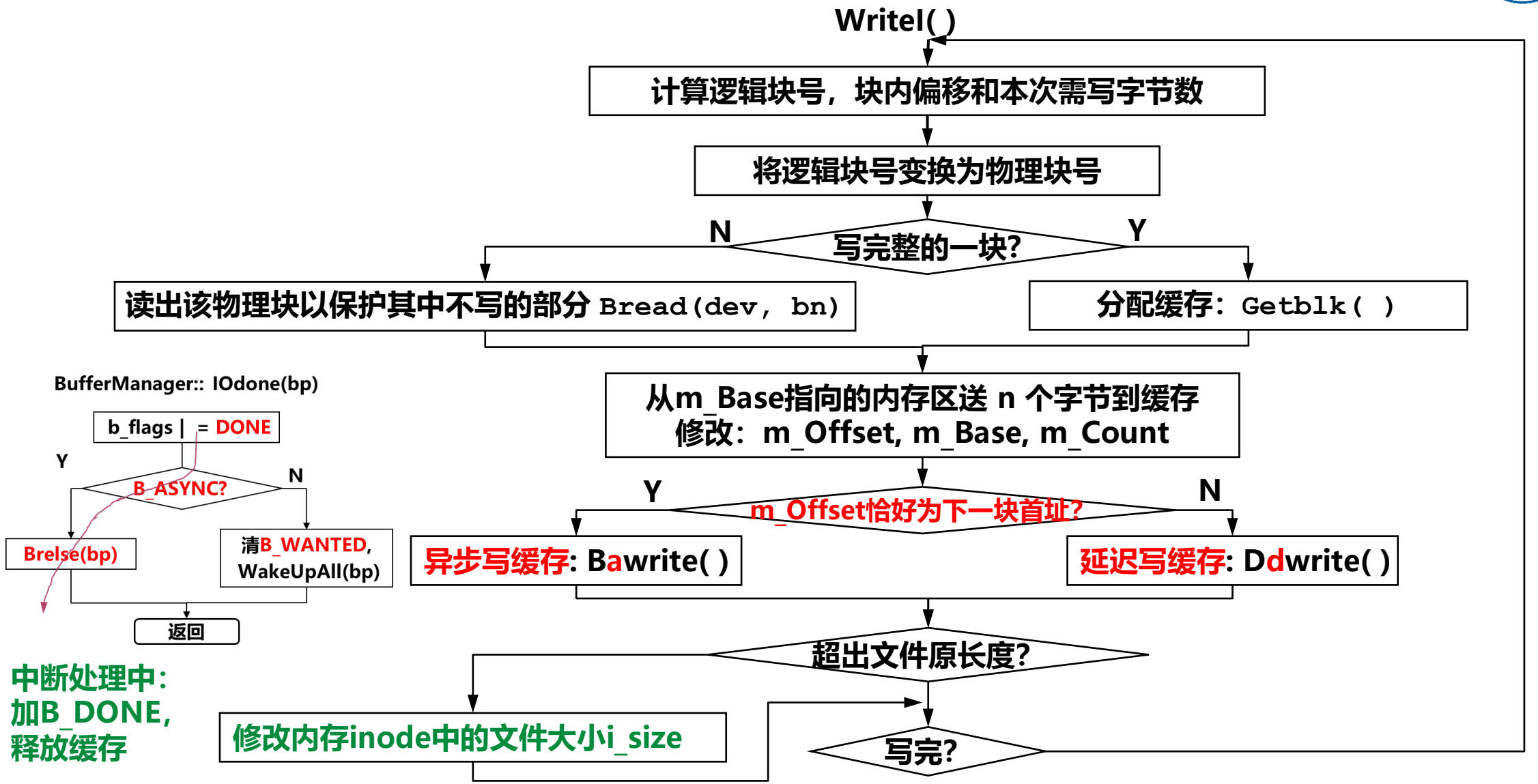
UNIX文件系统的读写操作

写操作



UNIX文件系统的读写操作

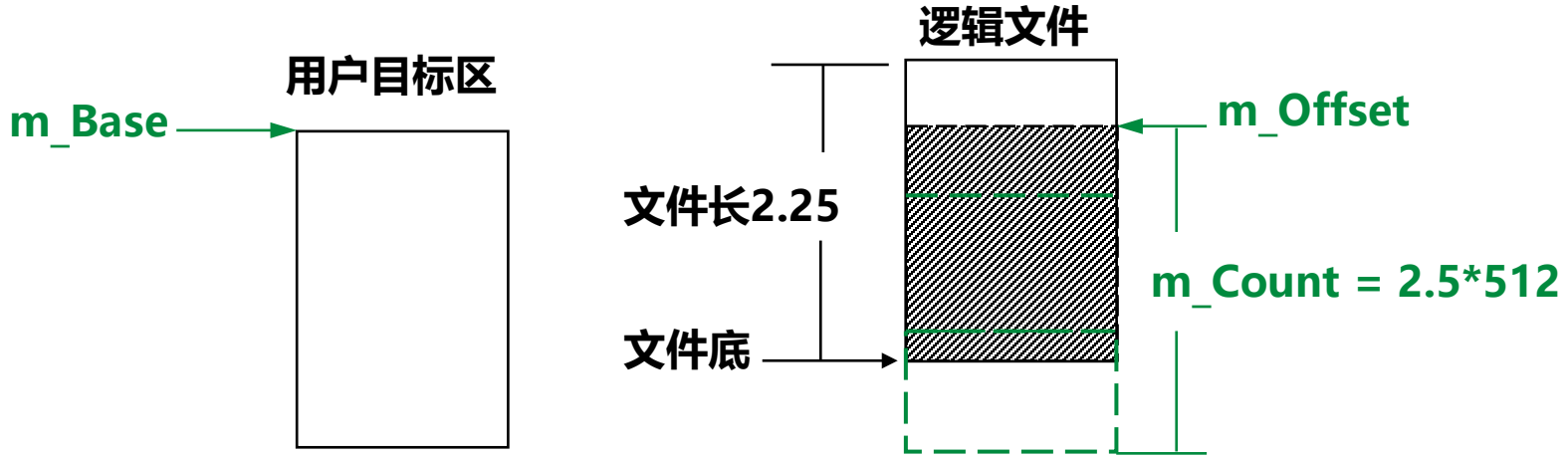
写操作



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?

本节小结

1　了解文件的逻辑结构与物理结构

2　掌握几种文件物理结构的特征和优缺点

3　熟悉UNIX文件系统的基本结构

阅读教材：262页 ~ 278页

E18：文件管理（UNIX文件系统的打开结构）