

第五章

设备管理

主要内容

5.1 I/O硬件系统

5.2 I/O软件系统

5.3 磁盘存储器管理

5.4 **UNIX字符块设备管理**

➤ 缓存管理

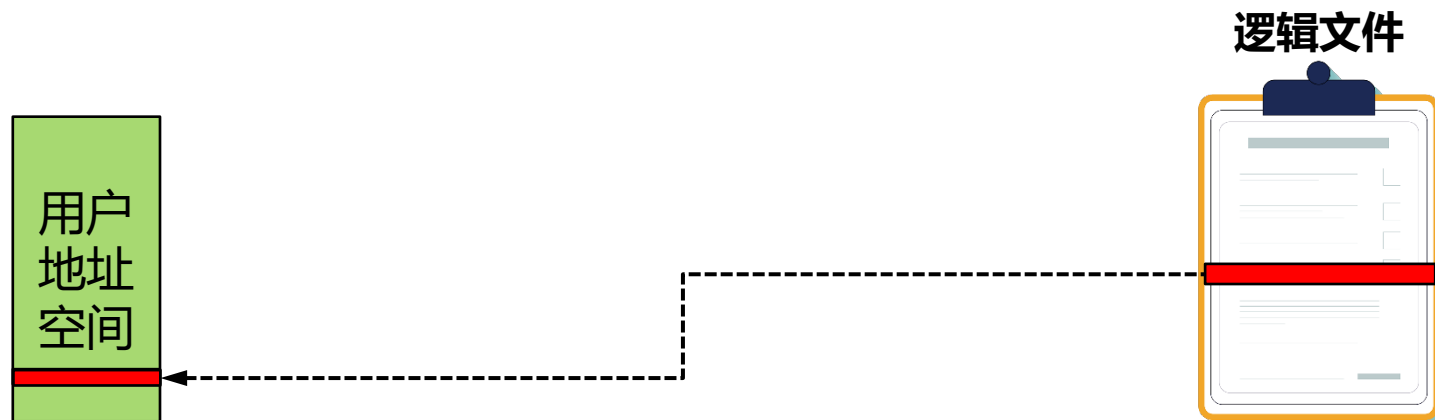
➤ **块设备读写过程**



UNIX块设备读写技术



读
操
作

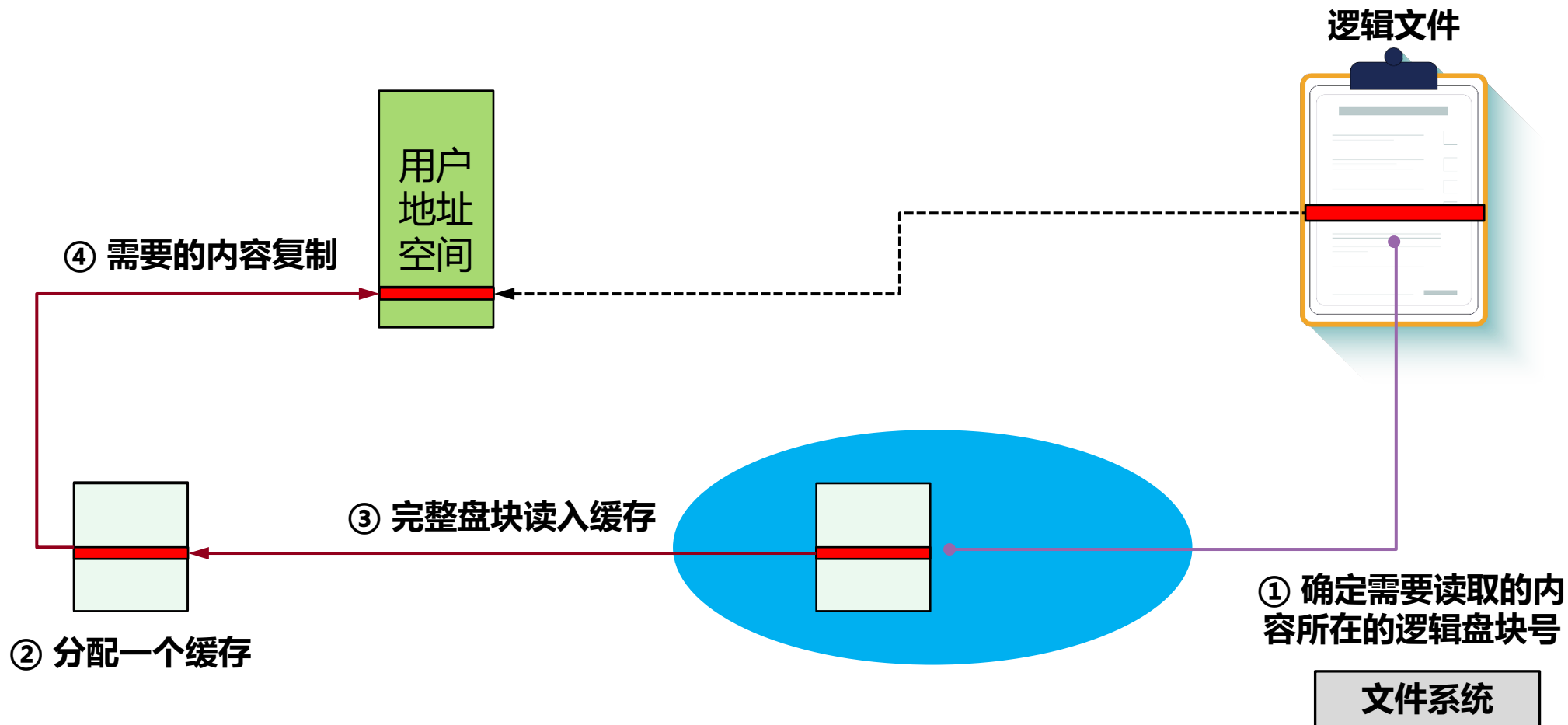




UNIX块设备读写技术



读操作





UNIX块设备读写技术



read() 系统调用时, 通过
文件系统得到dev, blkno

按同步方式将字符块读入内存

BufferManager:: Bread(dev, blkno)

根据dev, blkno 申请缓存
bp = this->GetBlk(dev, blkno)

返回一个在某
一设备队列中
加了B_BUSY的
缓存控制块

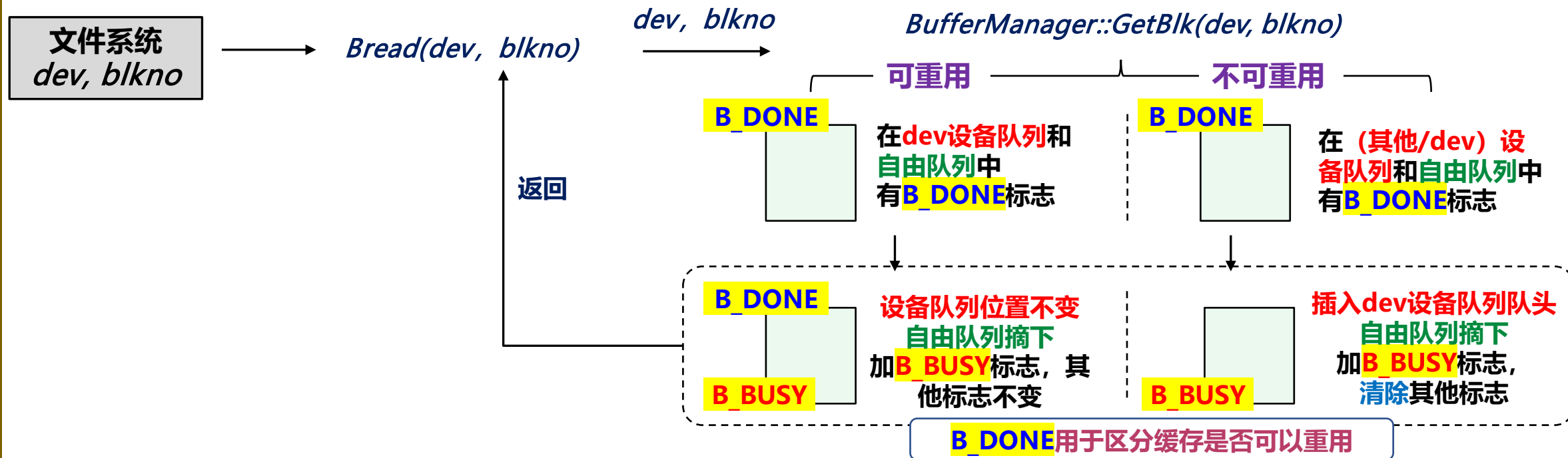
读
操
作



UNIX块设备读写技术



读操作





UNIX块设备读写技术



read() 系统调用时, 通过
文件系统得到dev, blkno

按同步方式将字符块读入内存

BufferManager:: Bread(dev, blkno)

根据dev, blkno 申请缓存
bp = this->GetBlk(dev, blkno)

返回一个在某
一设备队列中
加了B_BUSY的
缓存控制块

Y

含 B_DONE ?

可直接利用!

返回(bp)

读操作

进程没有因
为I/O操作睡
觉, 缓存中
的内容由文
件系统使用
完之后释放
到自由队列
的队尾!

所需信息已在缓存中



UNIX块设备读写技术



读操作

read() 系统调用时, 通过
文件系统得到dev, blkno

按同步方式将字符块读入内存

BufferManager:: Bread(dev, blkno)

根据dev, blkno 申请缓存
`bp = this->GetBlk(dev, blkno)`

返回一个在某
一设备队列中
加了B_BUSY的
缓存控制块

含 B_DONE ?

Y

N

`bp->b_flags |= Buf::B_READ;`
`bp->b_wcount = BufferManager::BUFFER_SIZE;`

将该控制块送入请求队列队尾, 若为队头,
立即启动设备驱动

返回(bp)

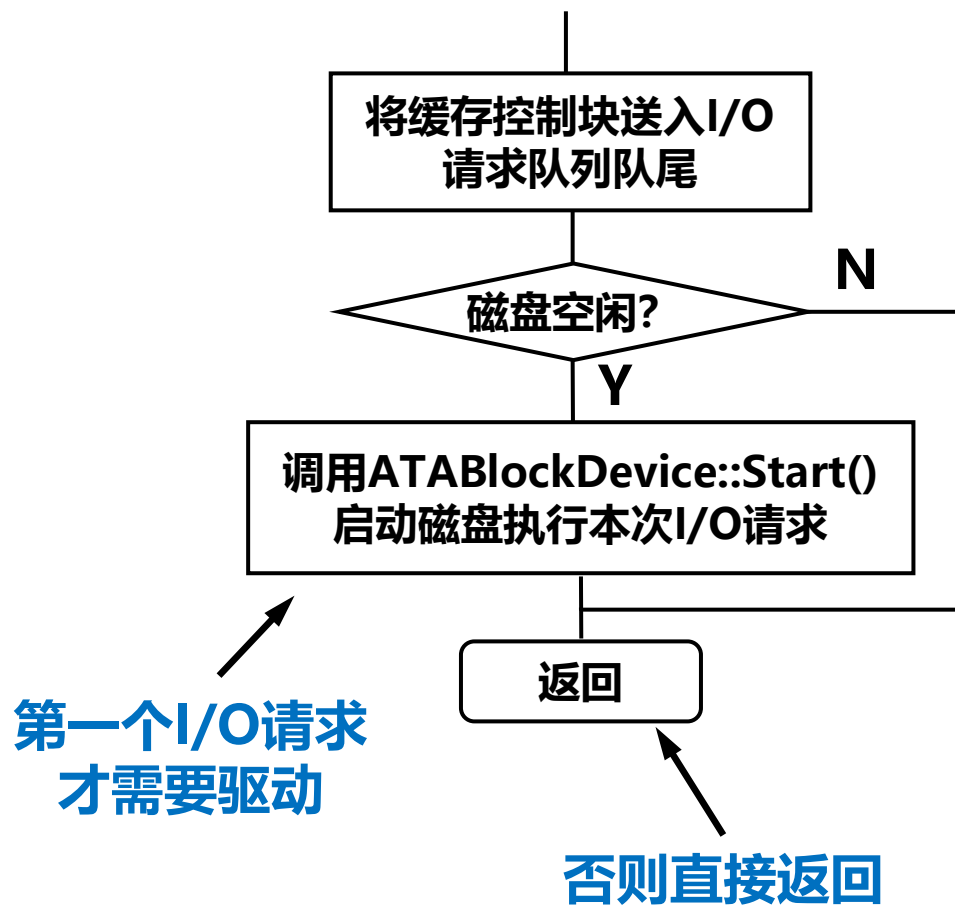
所需信息已在缓存中

可直接利用!

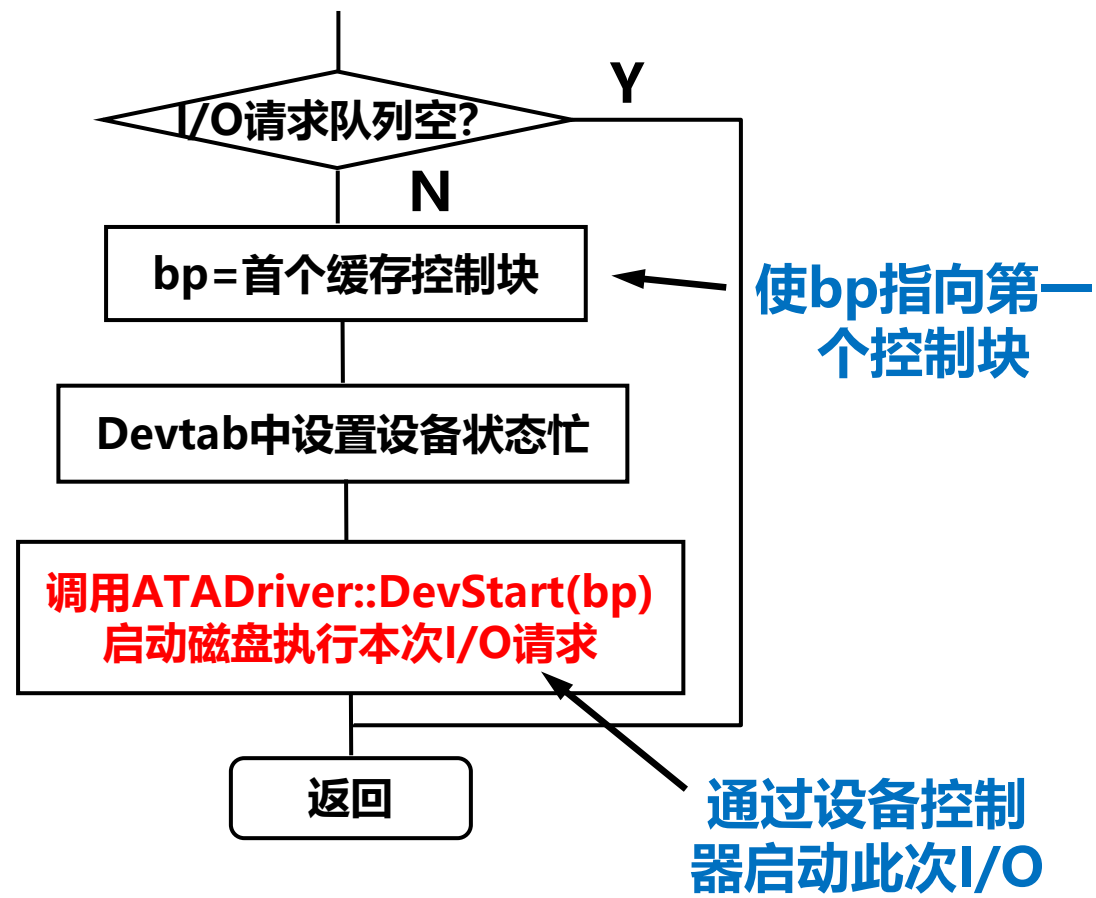


启动设备驱动

ATABlockDevice::Strategy(Buf* bp)



ATABlockDevice::Start()





UNIX块设备读写技术



读操作

read() 系统调用时, 通过
文件系统得到dev, blkno

按同步方式将字符块读入内存

BufferManager:: Bread(dev, blkno)

根据dev, blkno 申请缓存
`bp = this->GetBlk(dev, blkno)`

返回一个在某
一设备队列中
加了B_BUSY的
缓存控制块

Y

含 B_DONE ?

N

`bp->b_flags |= Buf::B_READ;`
`bp->b_wcount = BufferManager::BUFFER_SIZE;`

将该控制块送入请求队列队尾, 若为队头,
立即启动设备驱动

等待读操作结束
`BufferManager:: IOwait (bp)`

`Sleep((unsigned long)bp, ProcessManager::PRIBIO)`

返回(bp)

所需信息已在缓存中

可直接利用!

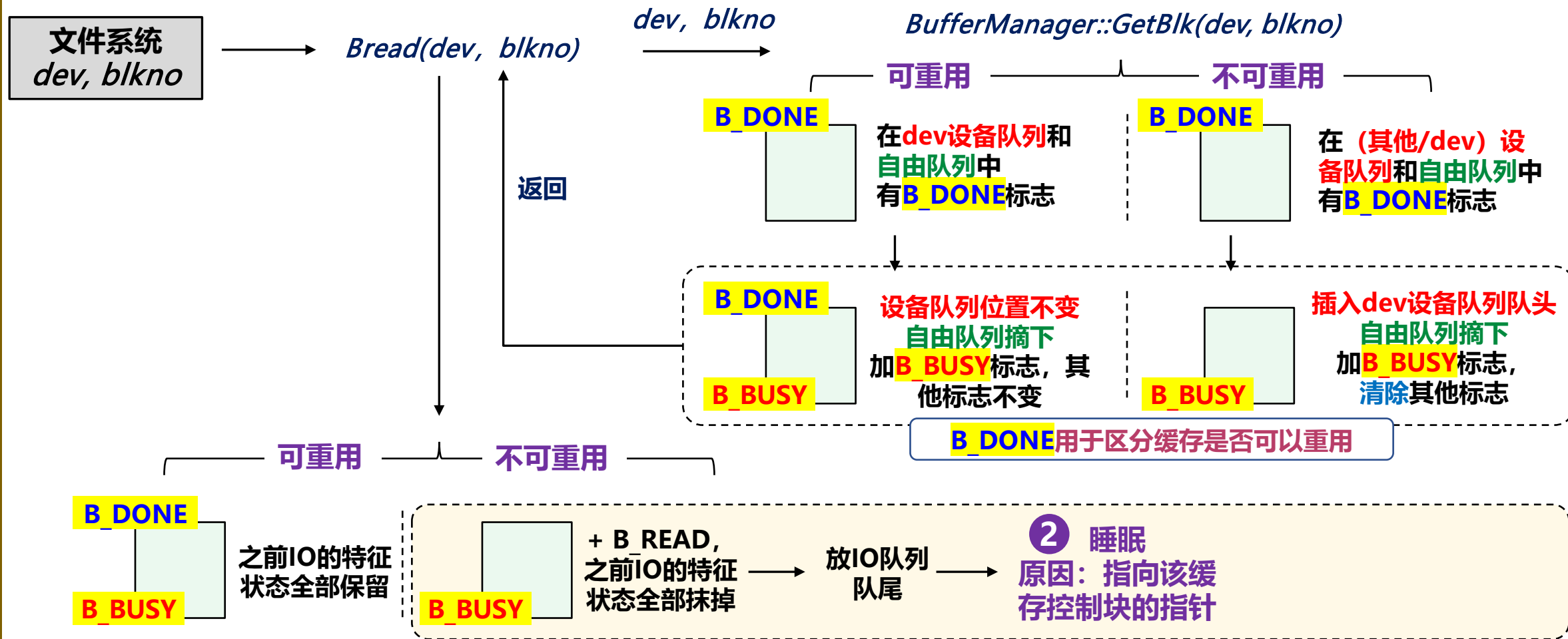
② 睡眠
原因: 指向该缓存控制块的指针



UNIX块设备读写技术



写操作





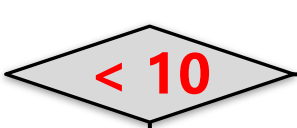
UNIX块设备读写技术



I/O完成, 中断响应ATADriver::ATAHandler()

① 清忙

② 有错:



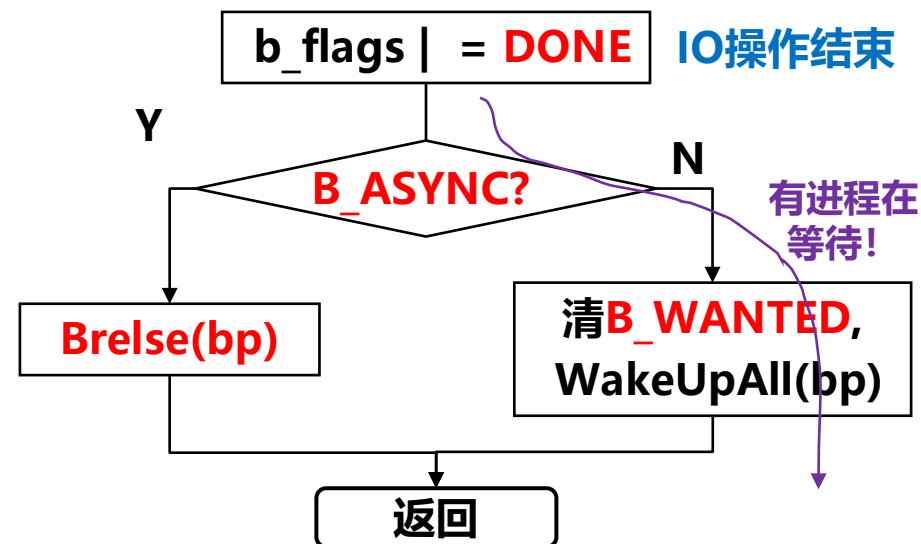
重来一次

b_flag |= B_ERROR

无错: 从I/O请求队列摘下第一个buf

调用 BufferManager:: IOdone(bp)

BufferManager:: IOdone(bp)

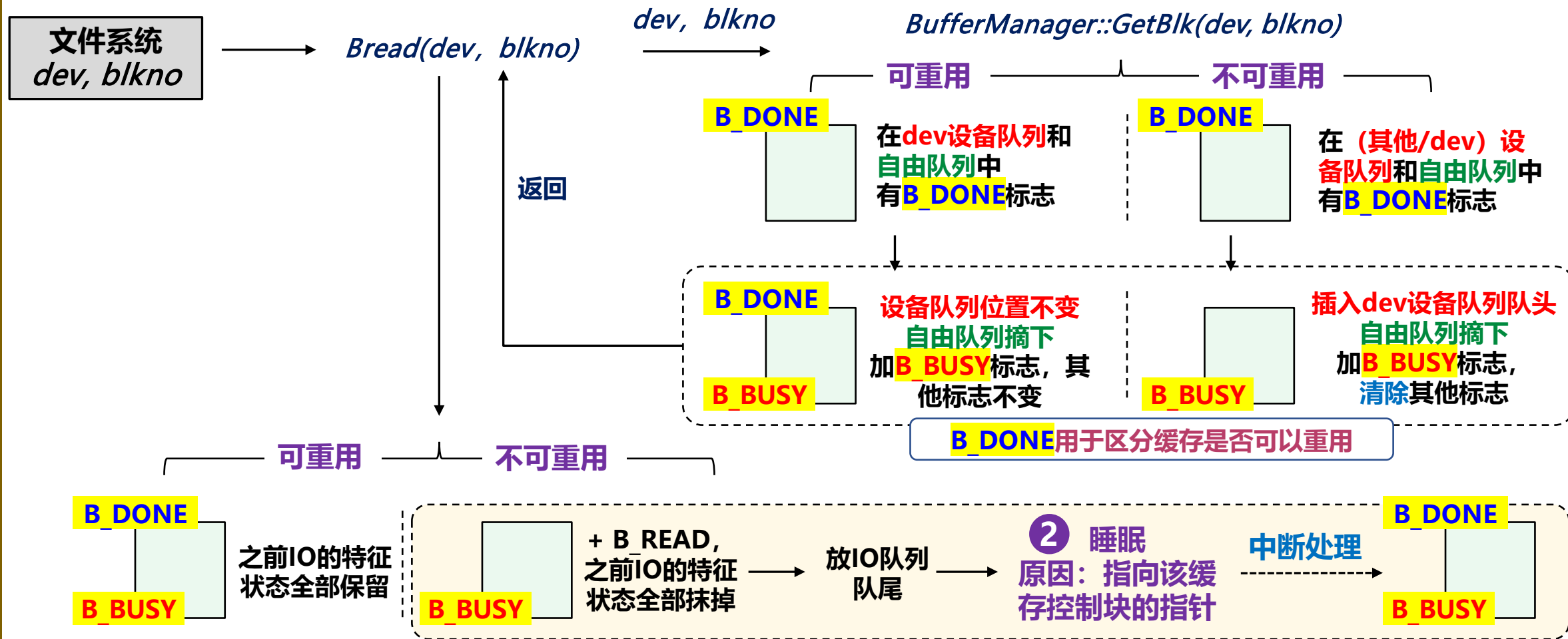




UNIX块设备读写技术



中断处理





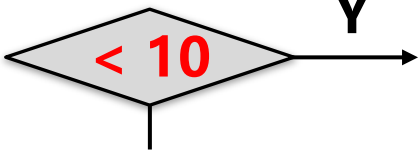
UNIX块设备读写技术



中断处理

I/O完成, 中断响应ATADriver::ATAHandler()

① 清忙

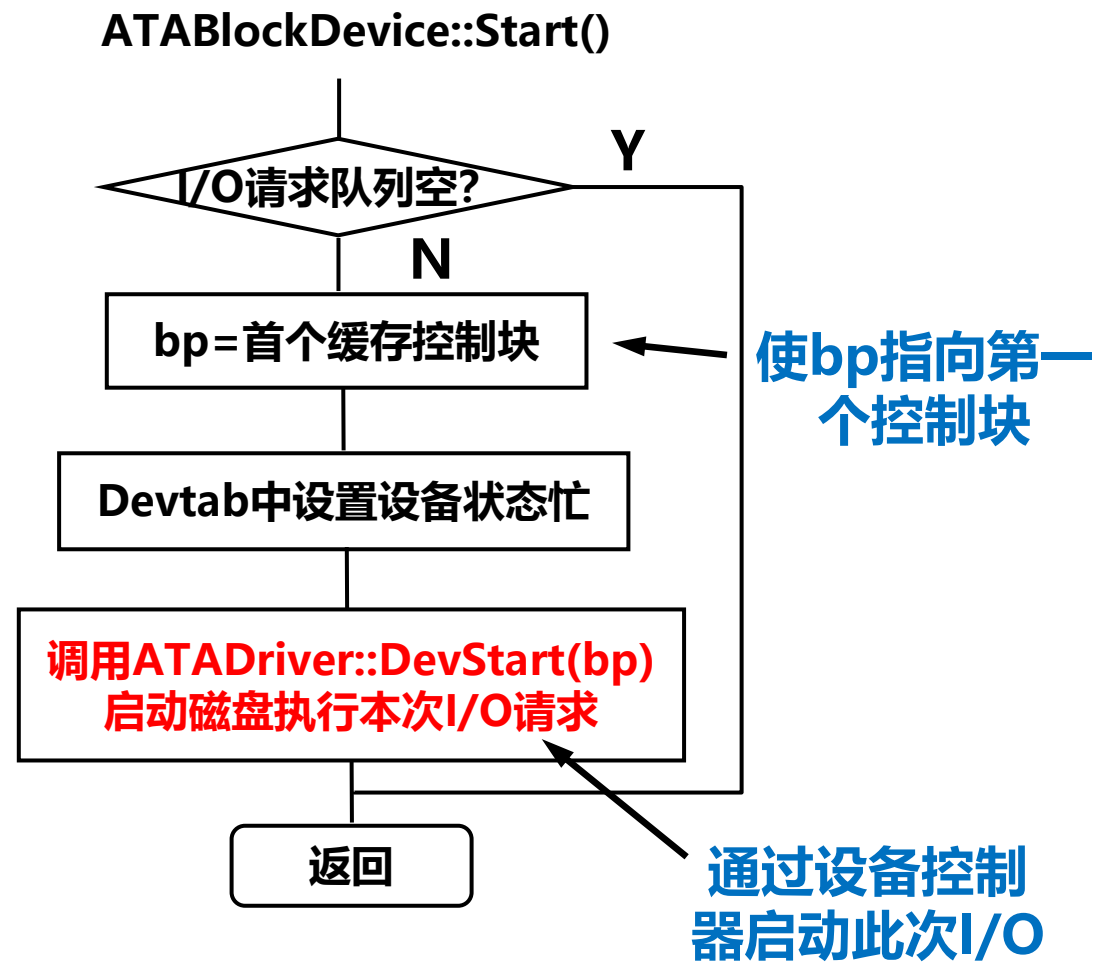
② 有错:  重来一次
b_flag |= B_ERROR

无错: 从I/O请求队列摘下第一个buf

调用 BufferManager:: IOdone(bp)

③ 启动I/O请求队列中的下一个请求

中断处理中实现了IO操作连续不断地处理





UNIX块设备读写技术



读操作

read() 系统调用时, 通过
文件系统得到 dev, blkno

按同步方式将字符块读入内存

BufferManager:: Bread(dev, blkno)

根据 dev, blkno 申请缓存
`bp = this->GetBlk(dev, blkno)`

返回一个在某
一设备队列中
加了 B_BUSY 的
缓存控制块

Y

含 B_DONE ?

N

`bp->b_flags |= Buf::B_READ;`
`bp->b_wcount = BufferManager::BUFFER_SIZE;`

将该控制块送入请求队列队尾, 若为队头,
立即启动设备驱动

等待读操作结束
`BufferManager::IOwait(bp)`

`Sleep((unsigned long)bp, ProcessManager::PRIBIO)`

唤醒后重新上台, 返回文件系统

返回(bp)

所需信息已在缓存中

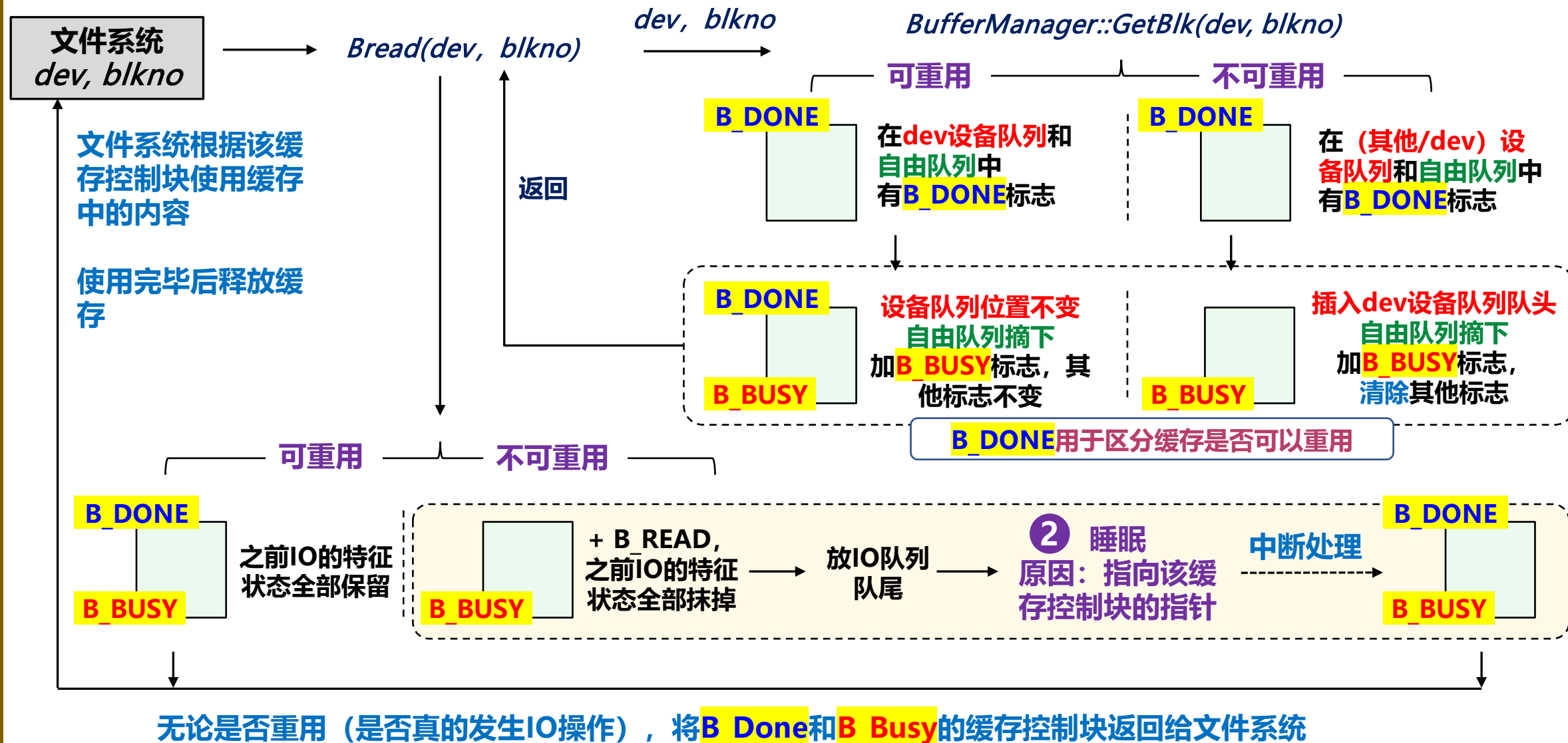
可直接利用!



UNIX块设备读写技术



读操作





UNIX块设备读写技术



读操作

文件系统

```
class BufferManager {
```

```
.....;
```

```
public:
```

②

```
void Initialize();
```

```
Buf* GetBlk( short dev, int blkno);
```

```
void Brelse(Buf* bp);
```

```
void IOWait(Buf* bp);
```

```
void IODone(Buf* bp);
```

```
Buf* Bread ( short dev, int blkno);
```

①

```
void Bwrite(Buf* bp);
```

```
void Bdwrite(Buf* bp);
```

```
void Bawrite(Buf* bp);
```

```
.....;
```

```
}
```

```
class ATADriver
```

```
{
```

```
public:
```

```
/* 磁盘中断设备处理子程序 */
```

```
static void ATAHandler(struct pt_regs* reg, struct pt_context* context);
```

```
/* 设置磁盘寄存器, 启动磁盘进行I/O操作 */
```

```
static void DevStart(struct Buf* bp);
```

```
.....;
```

```
};
```

```
class ATABlockDevice : public BlockDevice
```

```
{
```

```
public:
```

```
.....;
```

```
int Open(short dev, int mode);
```

```
int Close(short dev, int mode);
```

```
int Strategy(Buf* bp);
```

```
void Start();
```

④

③

⑥

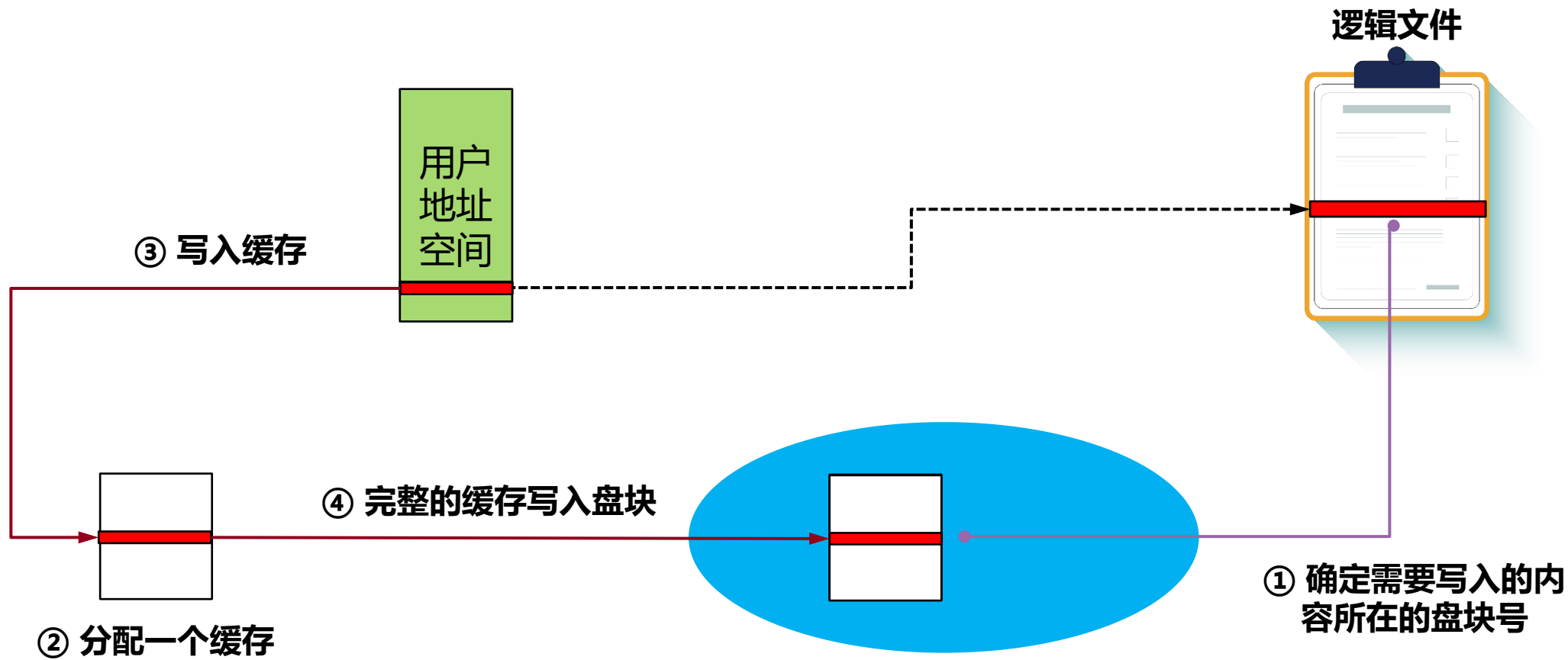
⑤



UNIX块设备读写技术



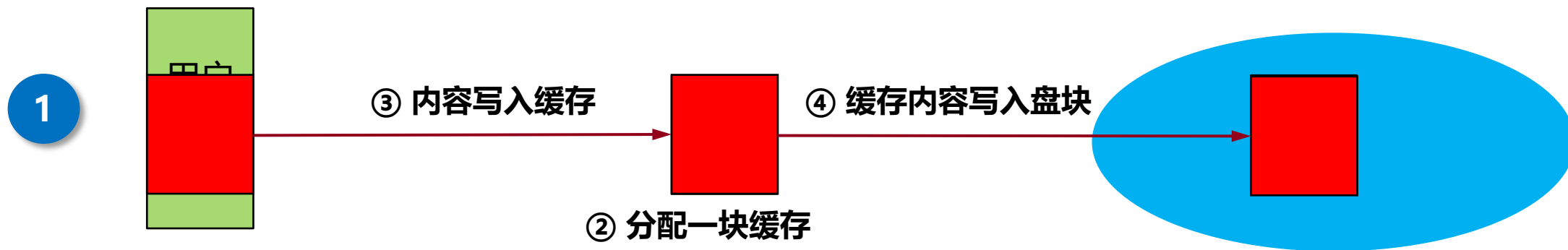
写操作



当写入部分不足一个盘块时，文件原有信息丢失



根据写入位置的不同分情况处理

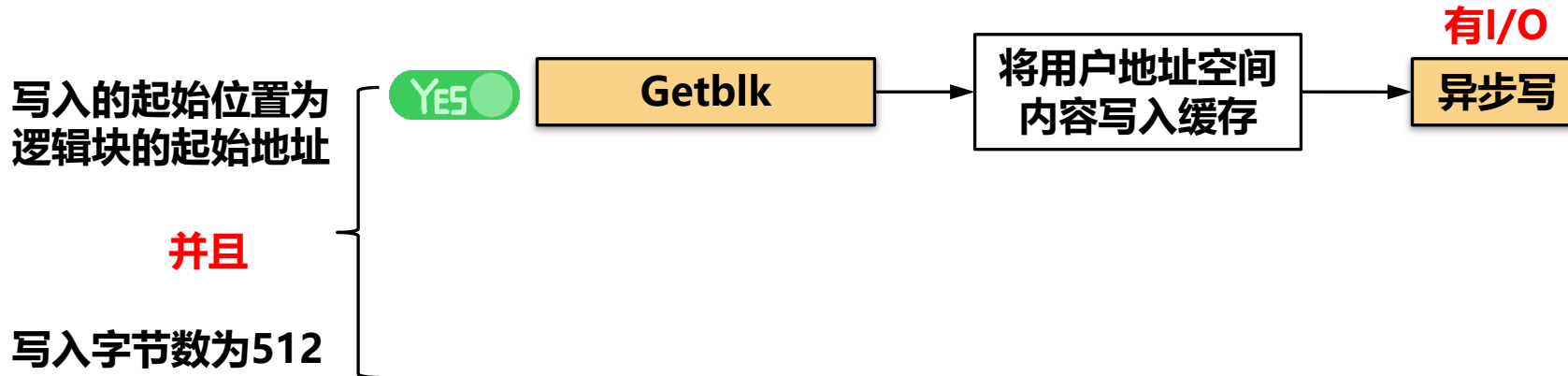
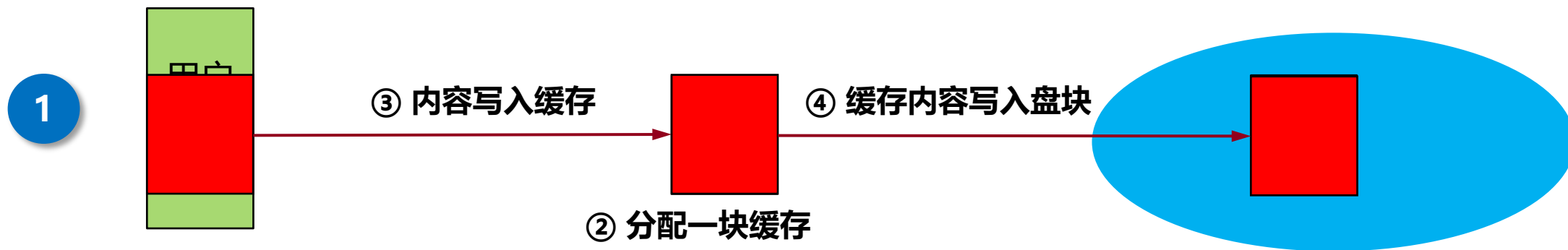




UNIX块设备读写技术



根据写入位置的不同分情况处理

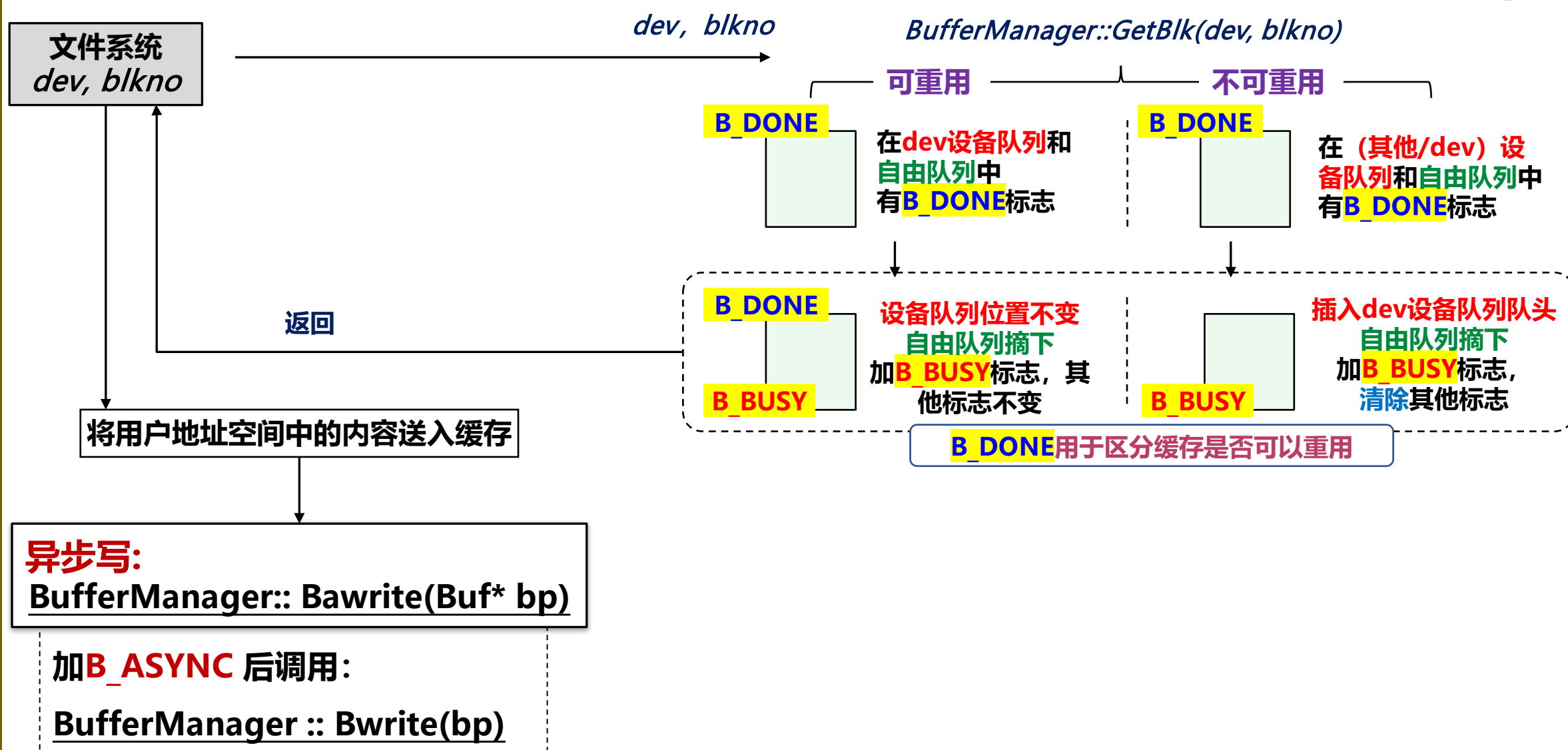




UNIX块设备读写技术



写操作





UNIX块设备读写技术



异步写:

BufferManager:: Bawrite(Buf* bp)

加**B_ASYNC** 后调用:

BufferManager :: Bwrite(bp)

BufferManager:: Bwrite(Buf* bp)

清bp->b_flags中: **B_READ,**
B_DONE, B_DELWRI, B_ERROR

bp->b_wcount = 256

启动设备驱动,
将该请求块送入请求队列

Y
异步写?

N

等待写操作结束
IOWait (bp)

Brelse(bp)

返回(bp)

写
操
作



UNIX块设备读写技术



异步写:

BufferManager:: Bawrite(Buf* bp)

加**B_ASYNC** 后调用:

BufferManager :: Bwrite(bp)

同步IO, 进程睡眠等待, 未来中断处理
中被唤醒后, 再由Bwrite释放缓存

这里与Bread不同, 因为写操作结束后,
文件系统不再需要缓存中的数据



BufferManager:: Bawrite(dev, blkno)

BufferManager:: Bwrite(Buf* bp)

清bp->b_flags中: **B_READ,**
B_DONE, B_DELWRI, B_ERROR

bp->b_wcount = 256

启动设备驱动,
将该请求块送入请求队列

Y
异步写?

N

等待写操作结束
IOWait (bp)

Brelse(bp)

返回(bp)



UNIX块设备读写技术



异步写:

BufferManager:: Bawrite(Buf* bp)

加**B_ASYNC** 后调用:

BufferManager :: Bwrite(bp)

BufferManager:: Bwrite(Buf* bp)

清bp->b_flags中: **B_READ**,
B_DONE, **B_DELWRI**, **B_ERROR**

bp->b_wcount = 256

启动设备驱动,
将该请求块送入请求队列

异步写?

等待写操作结束
IOWait (bp)

Brelse(bp)

返回(bp)

BufferManager:: IOdone(bp)

b_flags | = **DONE**

B_ASYNC?

没有进
程在等待!

Brelse(bp)

有进程在
等待!

清**B_WANTED**,
WakeUpAll(bp)

返回

异步IO, 进程没有
睡眠等待, 未来中
断处理中释放缓存

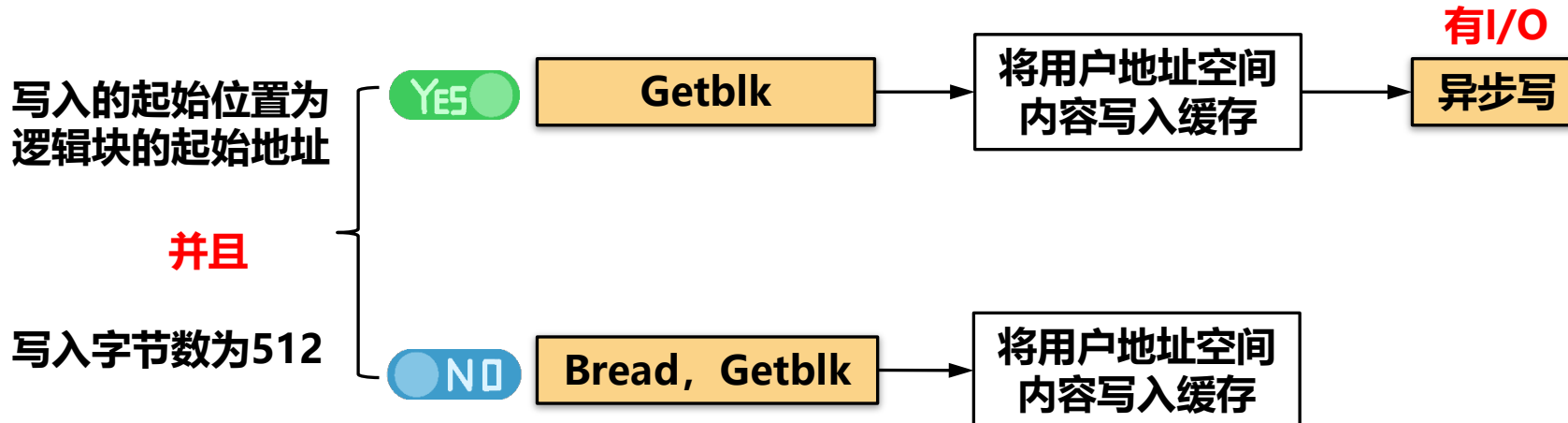
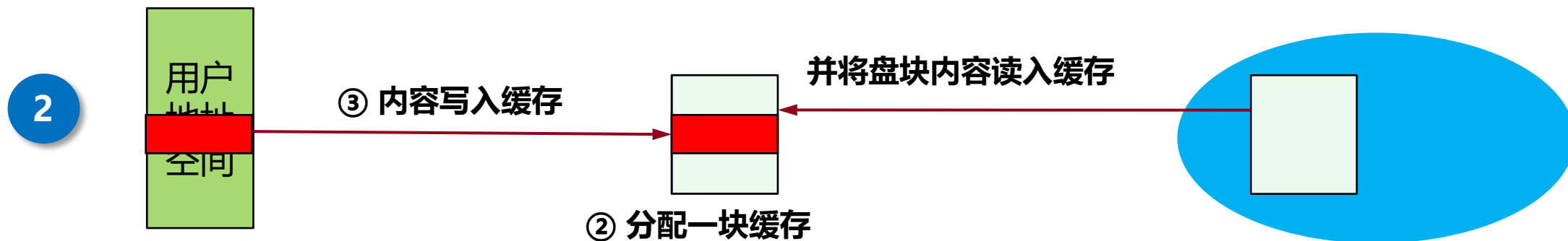
写
操
作



UNIX块设备读写技术



根据写入位置的不同分情况处理

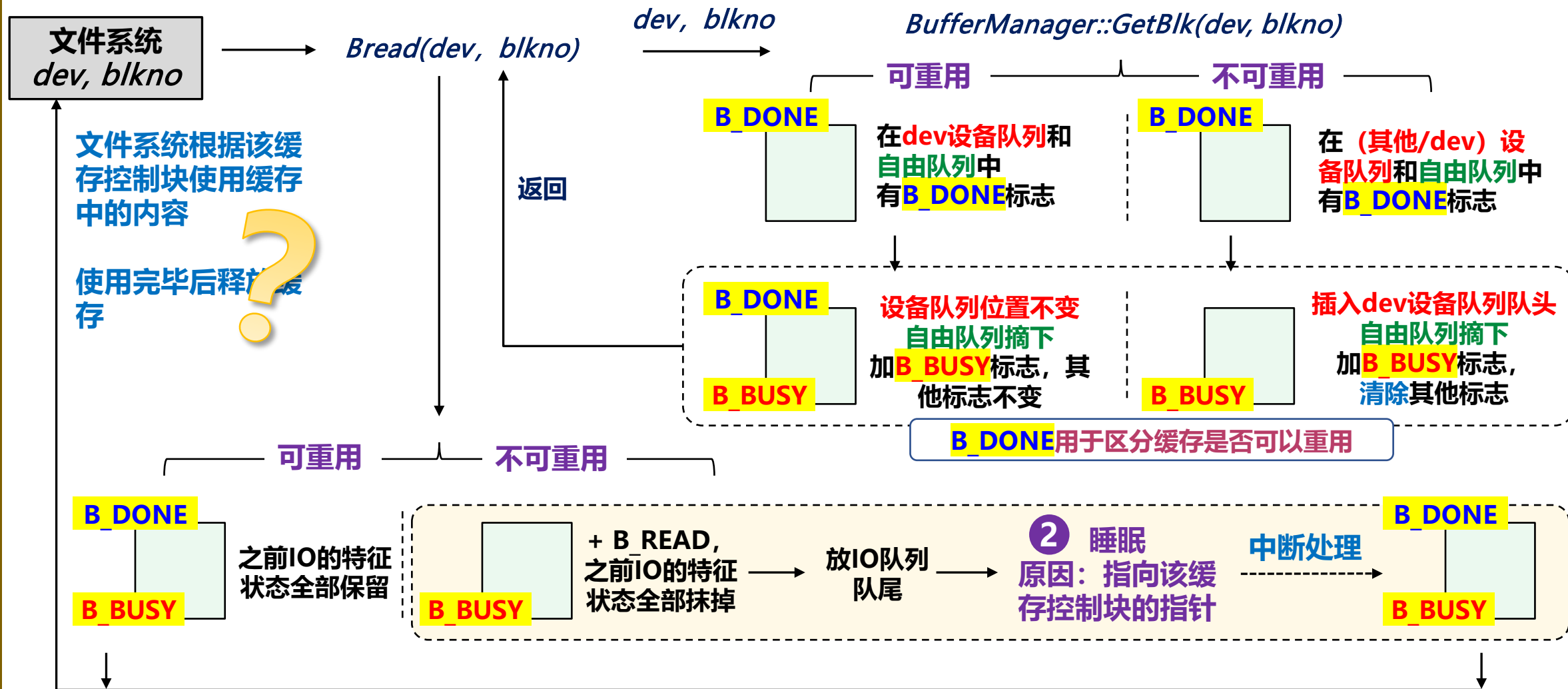




UNIX块设备读写技术



写操作

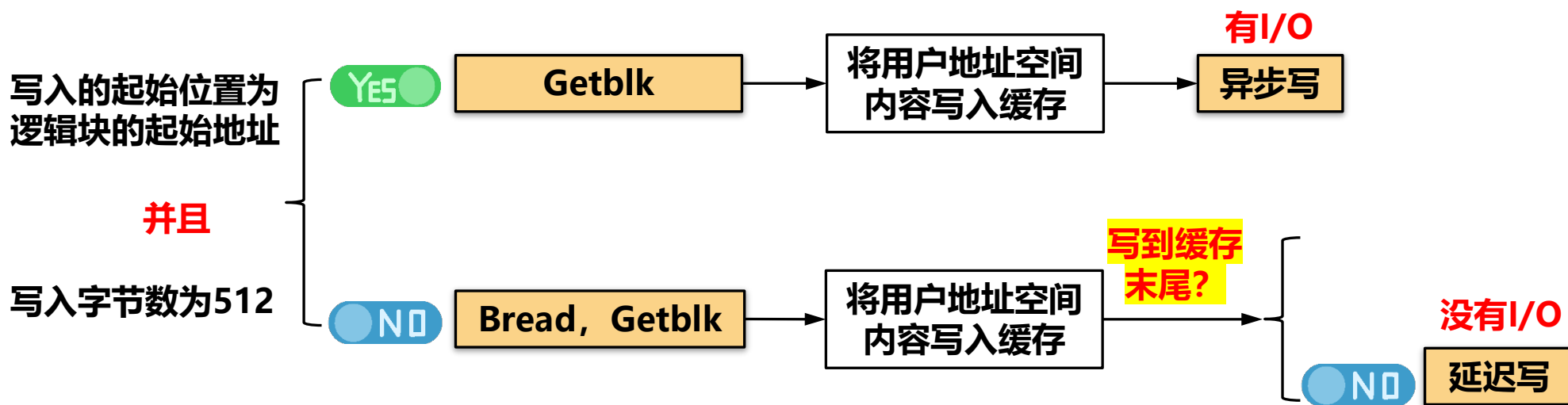
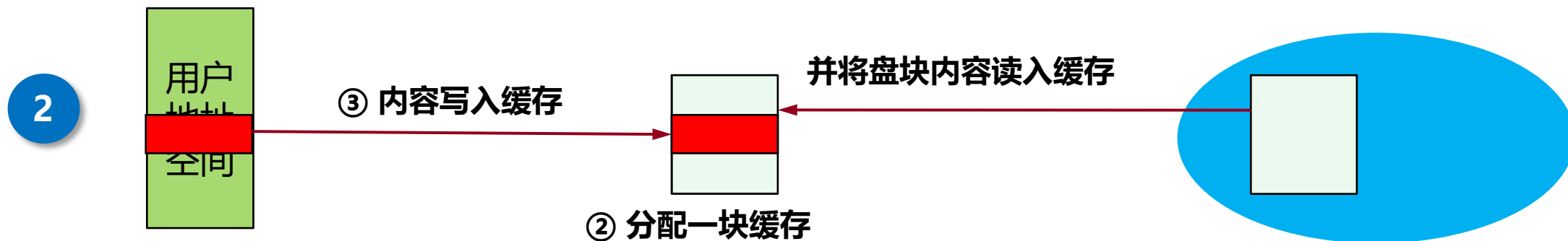




UNIX块设备读写技术



根据写入位置的不同分情况处理





UNIX块设备读写技术



写操作

异步写:

BufferManager:: Bawrite(Buf* bp)

加**B_ASYNC** 后调用:

BufferManager :: Bwrite(bp)

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)

BufferManager:: Bwrite(Buf* bp)

清bp->b_flags中: **B_READ,**
B_DONE, B_DELWRI, B_ERROR

bp->b_wcount = 256

启动设备驱动,
将该请求块送入请求队列

Y
异步写?

N

等待写操作结束
IOWait (bp)

Brelse(bp)

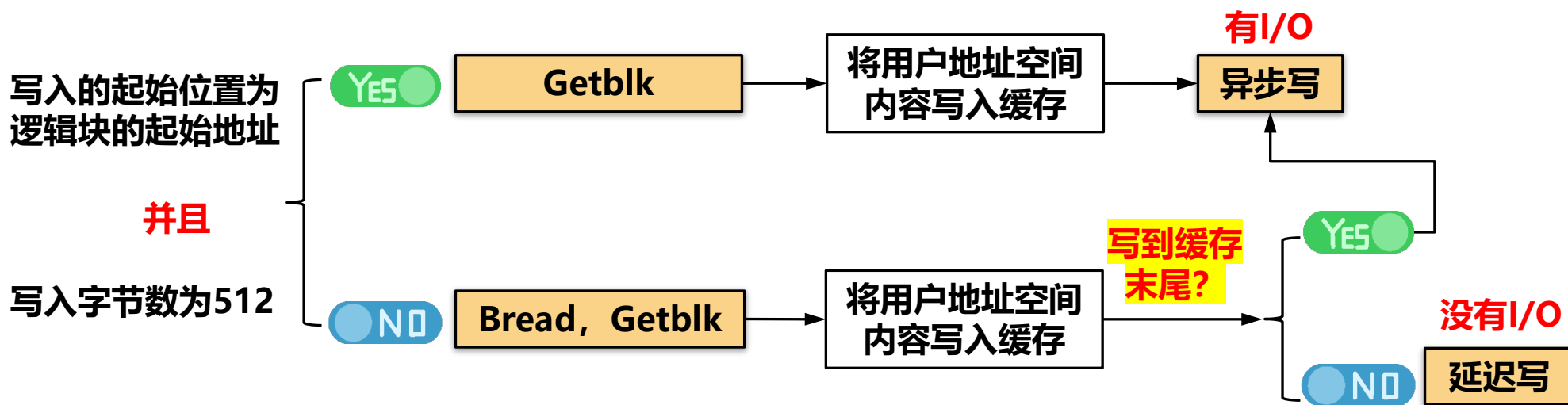
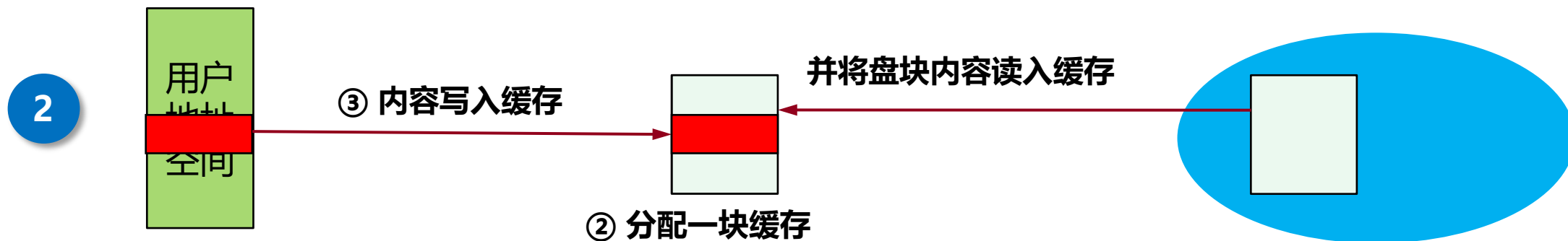
返回(bp)



UNIX块设备读写技术

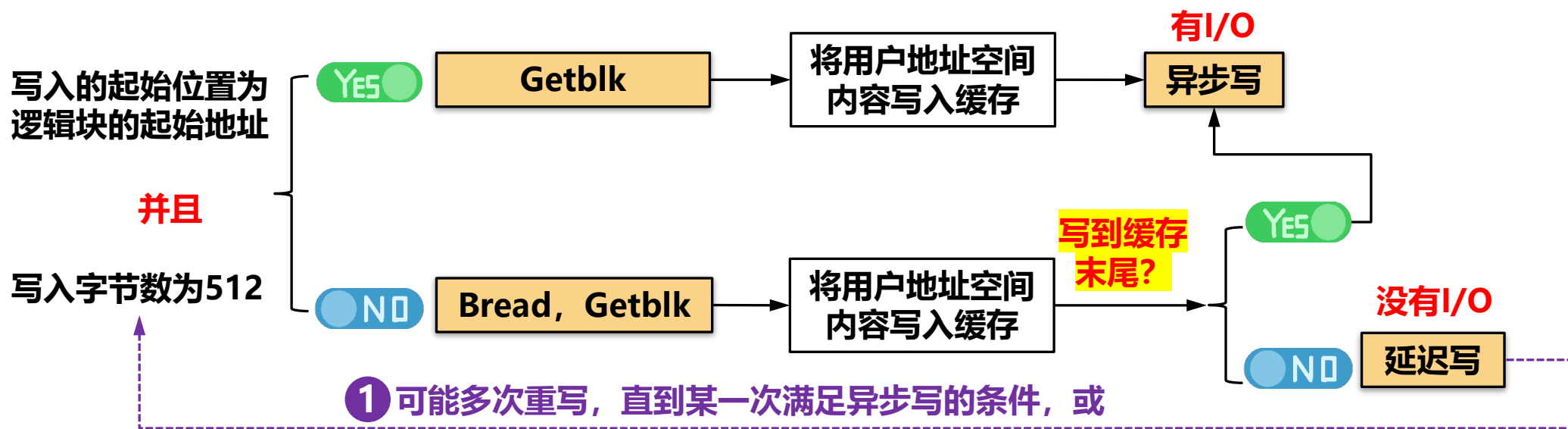
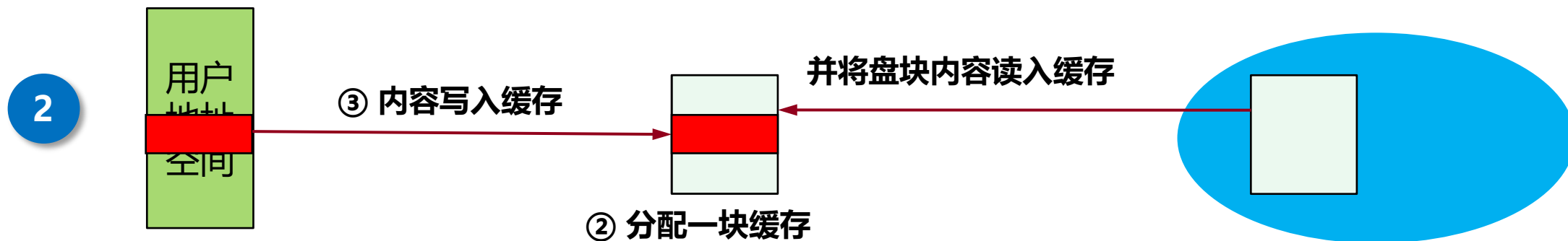


根据写入位置的不同分情况处理





根据写入位置的不同分情况处理

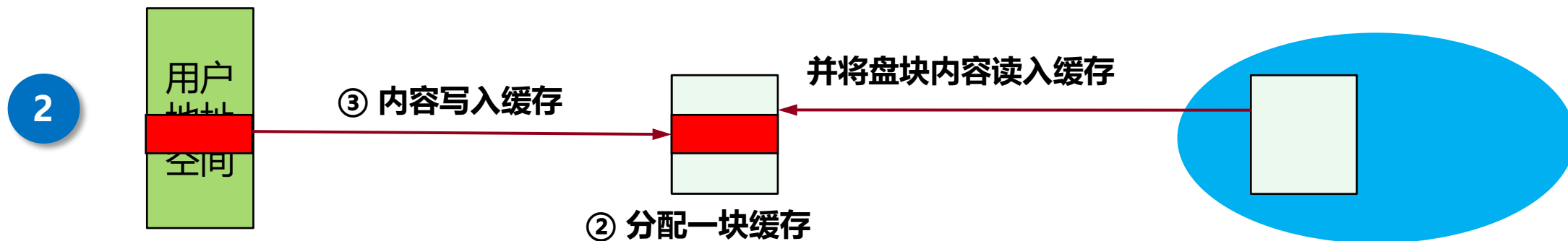




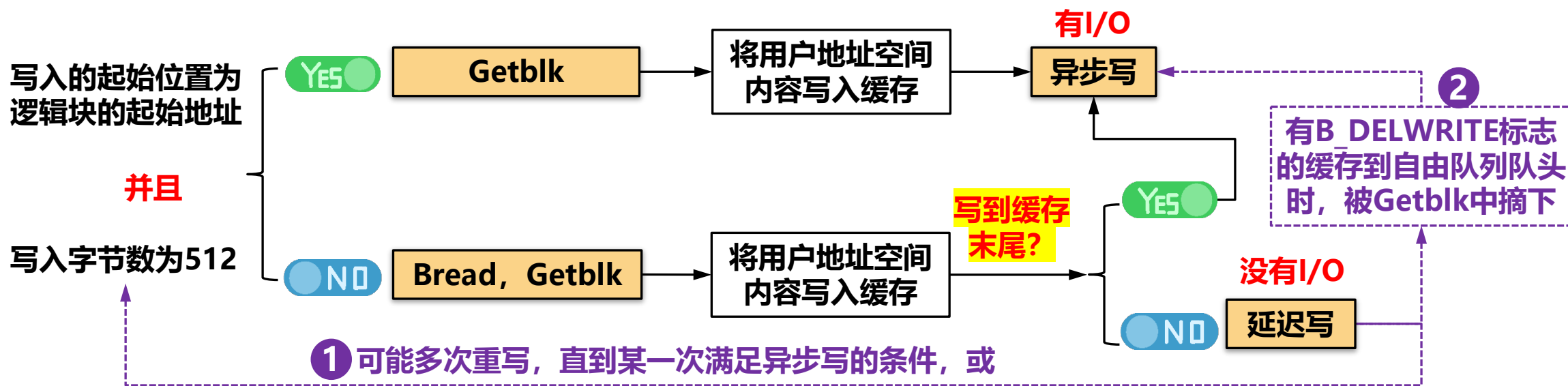
UNIX块设备读写技术



根据写入位置的不同分情况处理



写操作



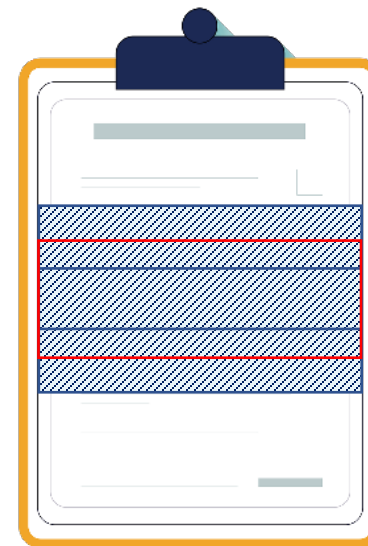


根据写入位置的不同分情况处理

内存空间

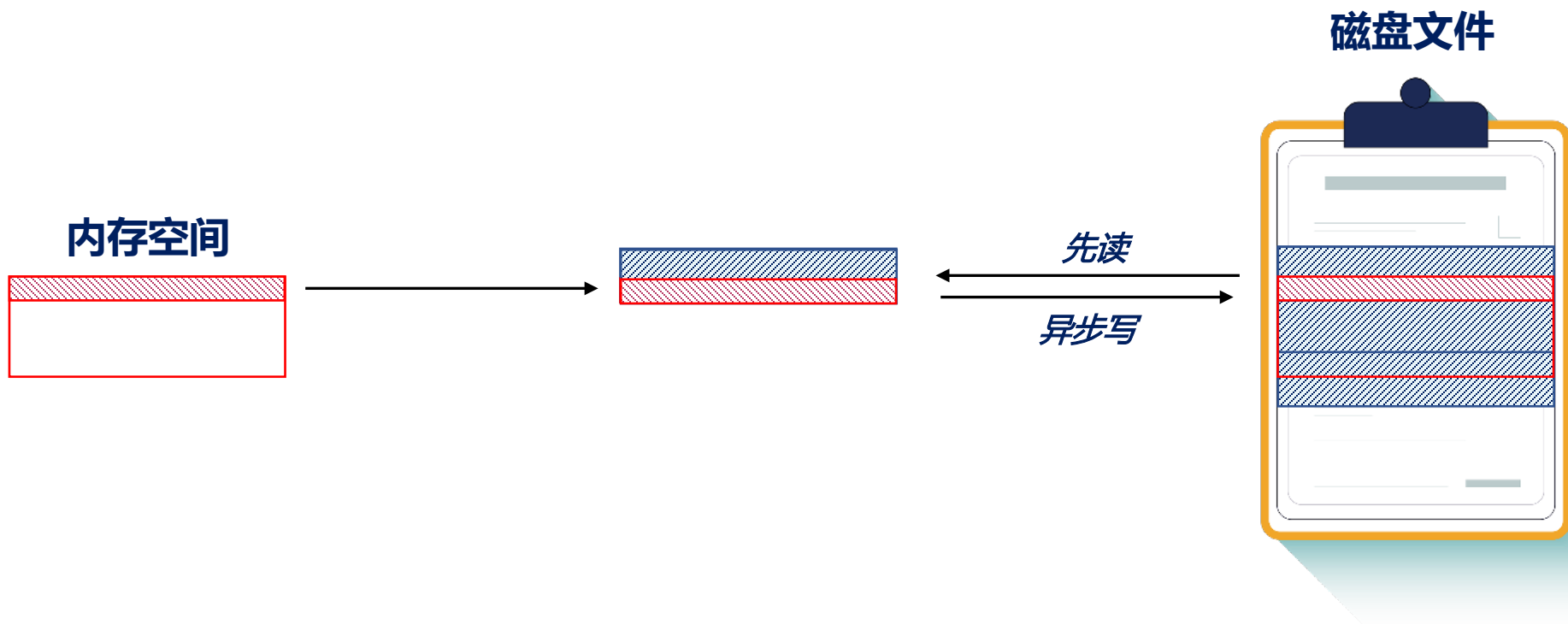


磁盘文件



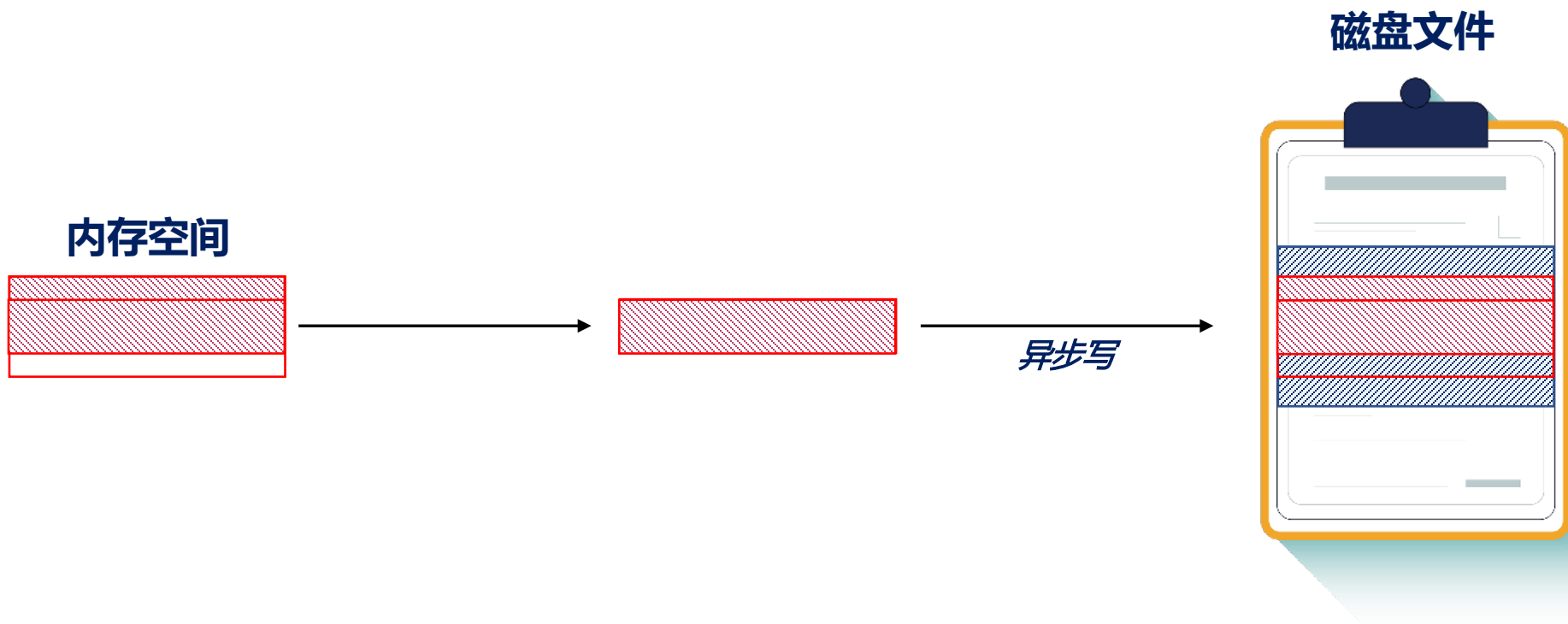


根据写入位置的不同分情况处理





根据写入位置的不同分情况处理

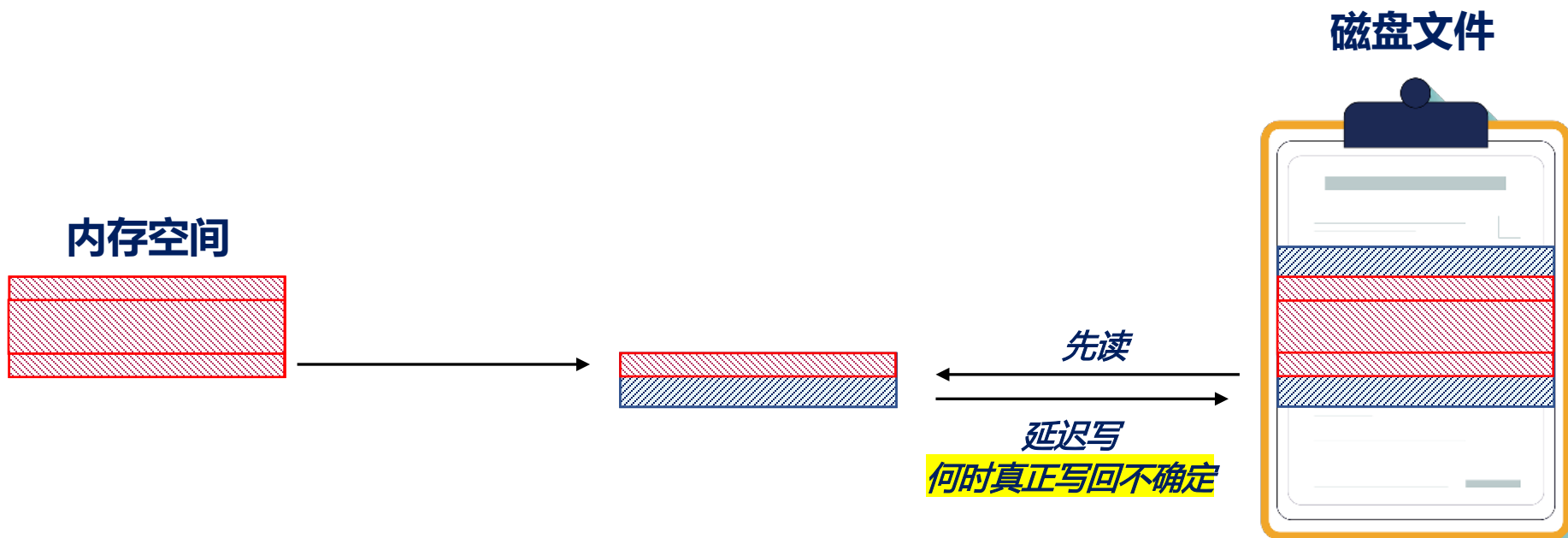




UNIX块设备读写技术



根据写入位置的不同分情况处理





UNIX块设备读写技术



关于缓存的竞争使用

BufferManager:: IOdone(bp)



等待重用缓存入睡的进程

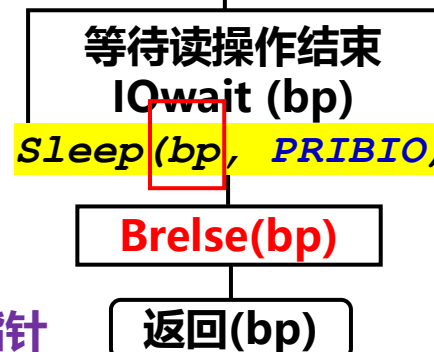
1



Bread



Bwrite



睡眠原因: 指向该缓存控制块的指针



UNIX块设备读写技术



关于缓存的竞争使用

BufferManager:: IOdone(bp)



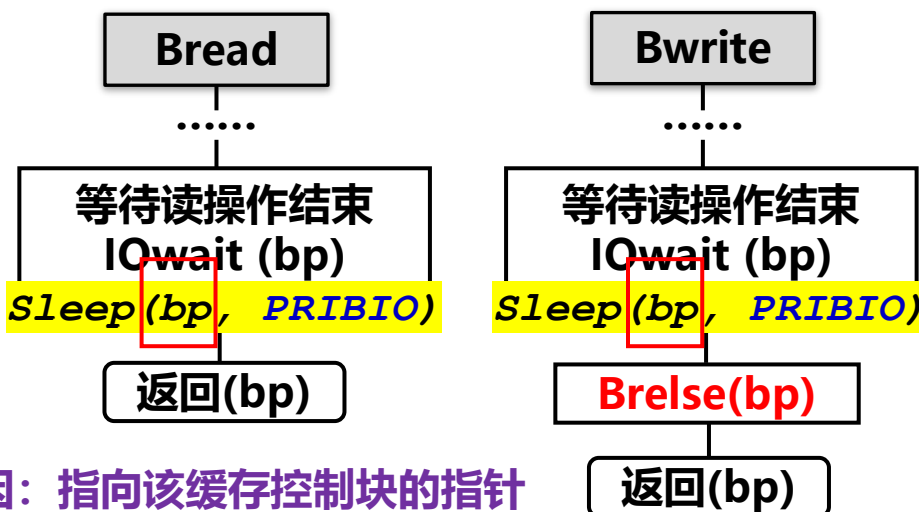
等待重用缓存入睡的进程

1



醒来后, 如果抢在2前上台, 则因为B_Busy再次入睡

睡眠原因: 指向该缓存控制块的指针



睡眠原因: 指向该缓存控制块的指针



UNIX块设备读写技术



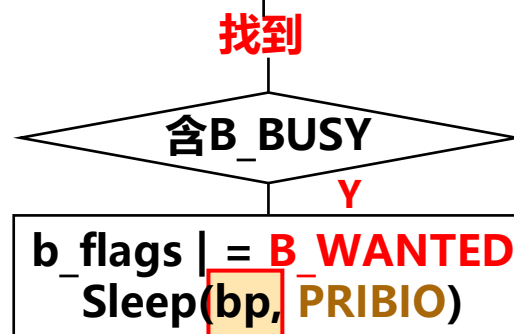
关于缓存的竞争使用

BufferManager:: IOdone(bp)



醒来后再上台，从Bread返回带有B_Done和B_Busy的缓存给文件系统
文件系统中使用完毕后，释放缓存

Getblk
在设备队列中找与dev, blkno相同者



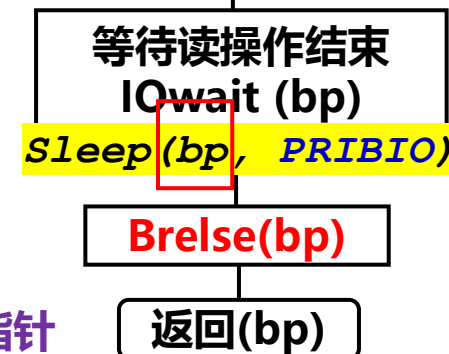
醒来后，如果抢在2前上台，则因为B_Busy再次入睡

睡眠原因：指向该缓存控制块的指针

Bread



Bwrite



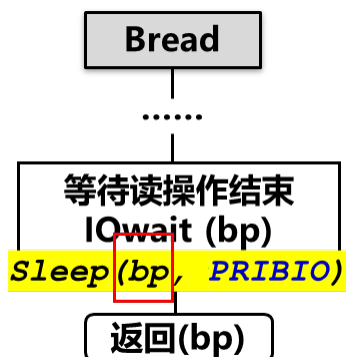
醒来后，再上台，释放缓存，写操作结束



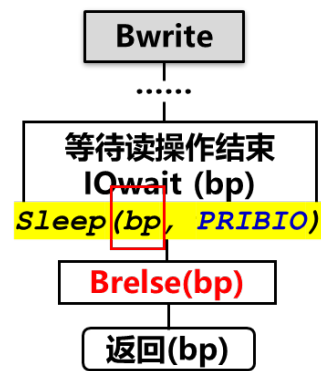
UNIX块设备读写技术



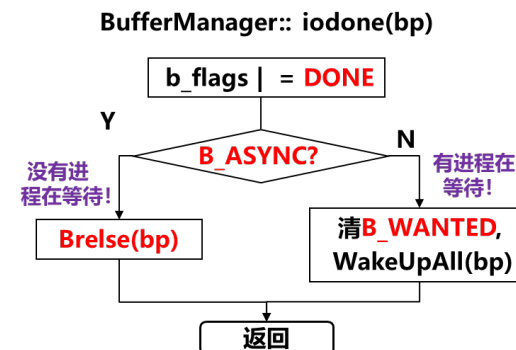
关于缓存的竞争使用



醒来后再上台，从Bread返回带有B_Done和B_Busy的缓存给文件系统
文件系统中使用完毕后，释放缓存



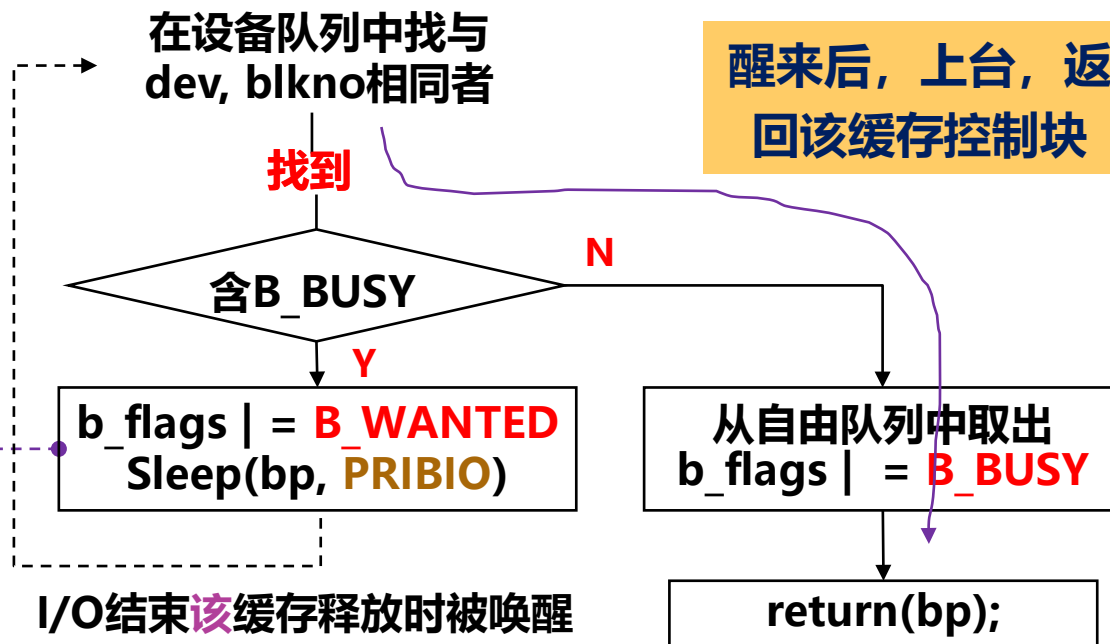
醒来后，再上台，释放缓存，写操作结束



BufferManager::Brelse(bp)



等待重用缓存入睡的进程



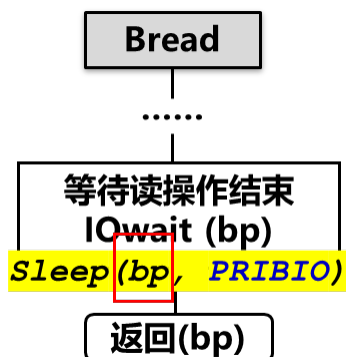
醒来后，上台，返回该缓存控制块



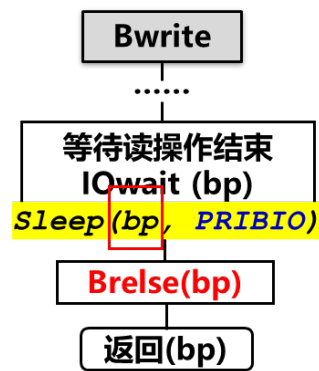
UNIX块设备读写技术



关于缓存的竞争使用



醒来后再上台，从Bread返回带有**B_Done**和**B_Busy**的缓存给文件系统
文件系统中使用完毕后，**释放缓存**



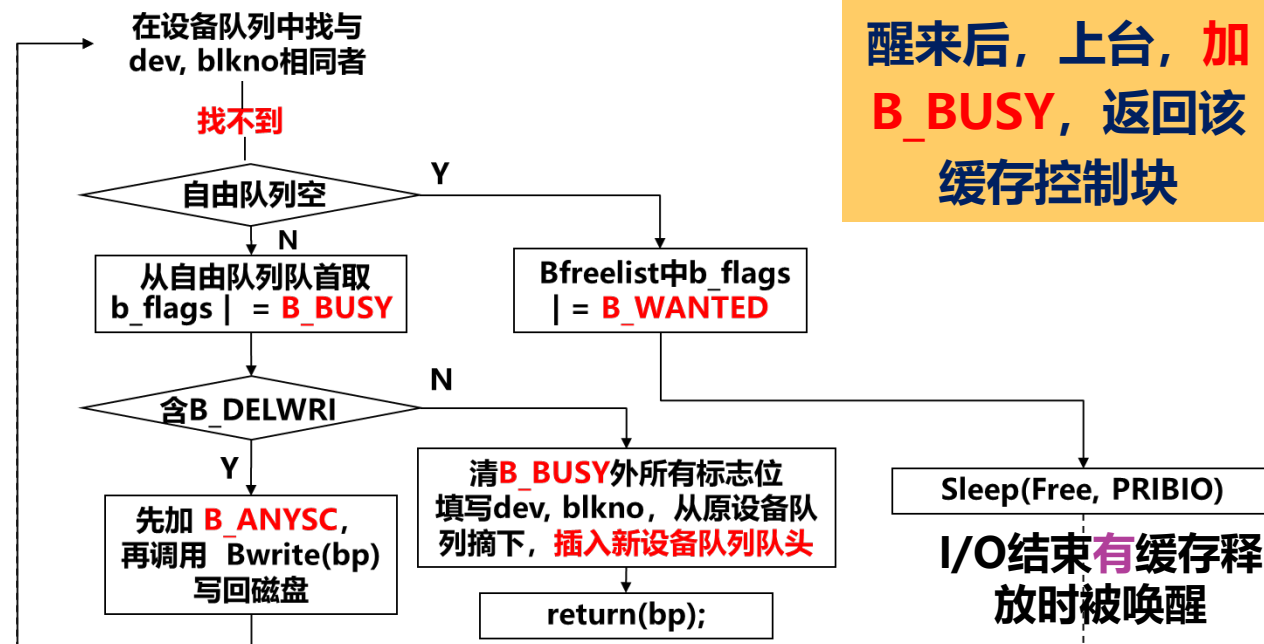
醒来后，再上台，释放缓存，写操作结束



BufferManager::Brelse(bp)



等待自由缓存入睡的进程



醒来后，上台，加**B_BUSY**，返回该缓存控制块



UNIX块设备读写技术



关于缓存的竞争使用

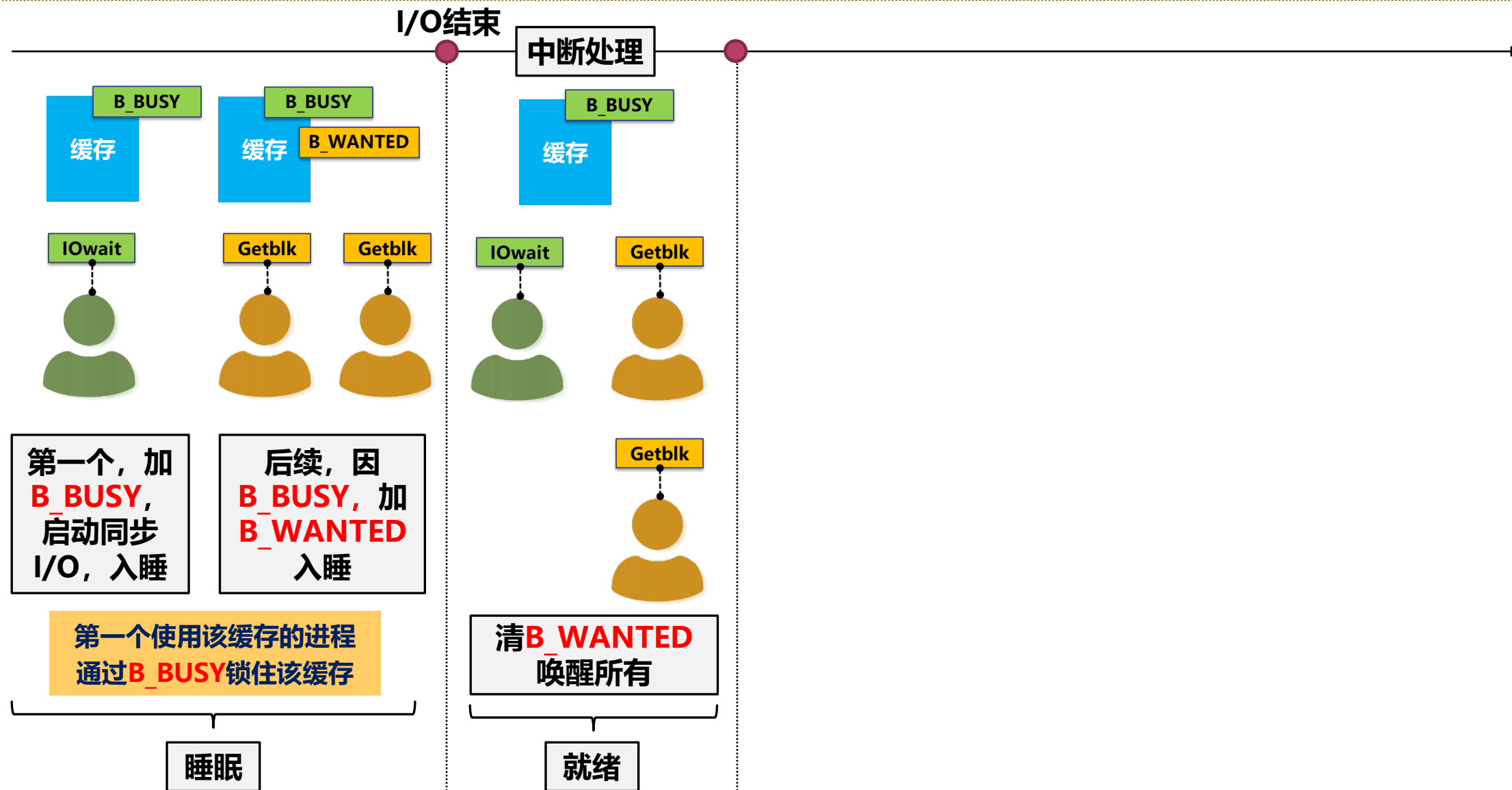




UNIX块设备读写技术



关于缓存的竞争使用





UNIX块设备读写技术



关于缓存的竞争使用

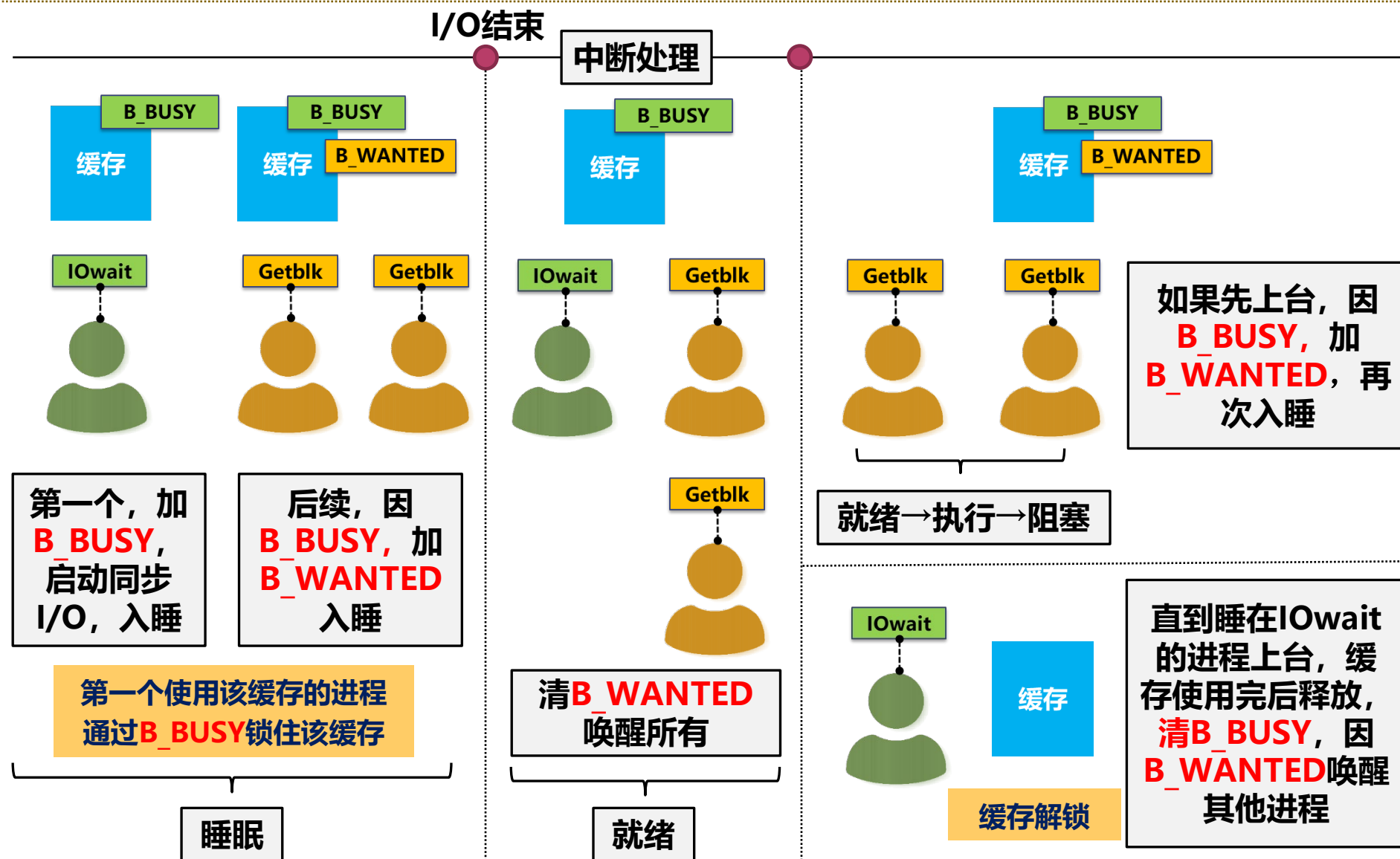


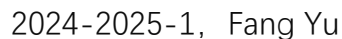


UNIX块设备读写技术



关于缓存的竞争使用







1. 磁盘调度算法（减少寻道时间）
2. 磁盘高速缓存（重复读，延迟写，**预读**）



如果用户进程在顺序读文件时，能够将用户即将要读到的东西预先读到缓存里，当用户真正需要的时候，就可以重用该缓存，减少IO。



1. 磁盘调度算法（减少寻道时间）
2. 磁盘高速缓存（重复读，延迟写，**预读**）



Q: 如何判断?

A: 上次读的第 i 块，这次读 $i+1$ 块，下次很可能读 $i+2$ 块。

如果用户进程在**顺序读**文件时，能够将用户即将要读到的东西**预先读**到缓存里，当用户**真正需要**的时候，就可以重用该缓存，减少IO。

Q: 不是用户发起的，如何在用户进程不知道的情况下，尽量少打扰，完成读操作？

A: 采用异步IO方式。

如何判断预测失败？

如果预测失败怎么办？

如何将预测失败的代价降到最小？

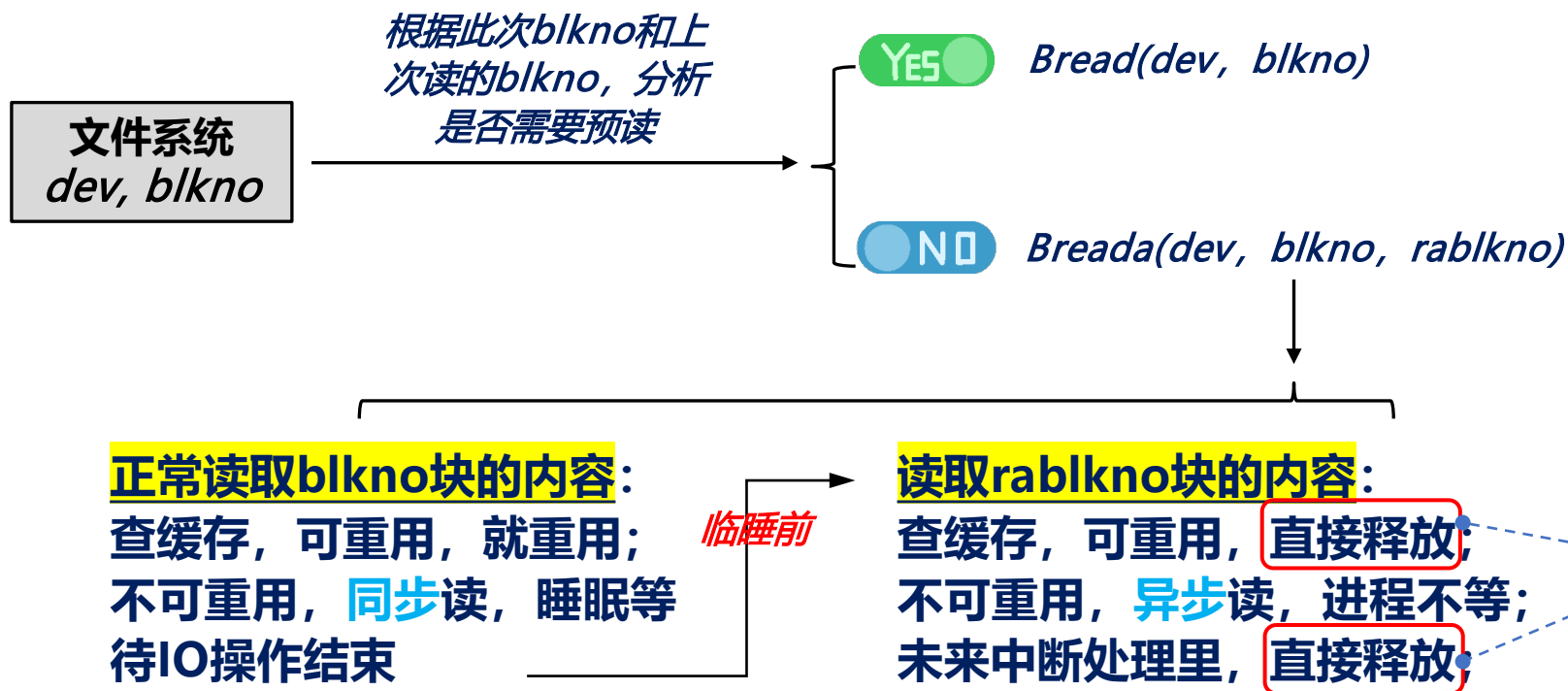


磁盘存储器管理



提高磁盘读写效率的方法

1. 磁盘调度算法（减少寻道时间）
2. 磁盘高速缓存（重复读，延迟写，**预读**）



预读块缓在自由缓存队列的时间里, 进程如果需要 (**预测对**), 则可以随时重用;

预读块到达自由缓存队列队头, 如果进程都没有用过 (**预测失败**), 则由GetBlk分配给其他盘块。



1. 在UNIX V6++中，试说明缓存控制块Buf 有无可能，在什么样的条件下出现下列情况：

(1) 同时处在自由Buf 和一个设备Buf 队列中；

IO操作完成，缓存已释放：有B_DONE，无B_BUSY

(2) 同时处在某一设备Buf 队列和I/O 请求队列中；

IO操作正在进行中，无B_DONE，有B_BUSY

(3) 同时处在自由Buf 和NODEV Buf 队列中；系统初启完成初始化

(4) 只处在某一设备Buf 队列中；

Getblk中找到可以重用的缓存，将其从自由队列中摘下，返回给文件系统，但是仍然保持在原设备队列中：有B_DONE，有B_BUSY

(5) 只处在自由Buf 队列中； (6) 只处在某一设备的I/O 请求队列中；

(7) 只处在NODEV Buf 队列中；

(8) 同时出现在自由Buf、某一设备的Buf 队列和I/O 请求队列中；

(9) 同时出现在一类设备的Buf 队列、另一类设备的I/O 请求队列中；



本节小结



1 掌握UNIX如何利用高速缓存实现块设备读写

阅读讲义：242页 ~ 259页



E16：设备管理（UNIX的缓存管理与读写技术）