第五章

设备管理

方　钰

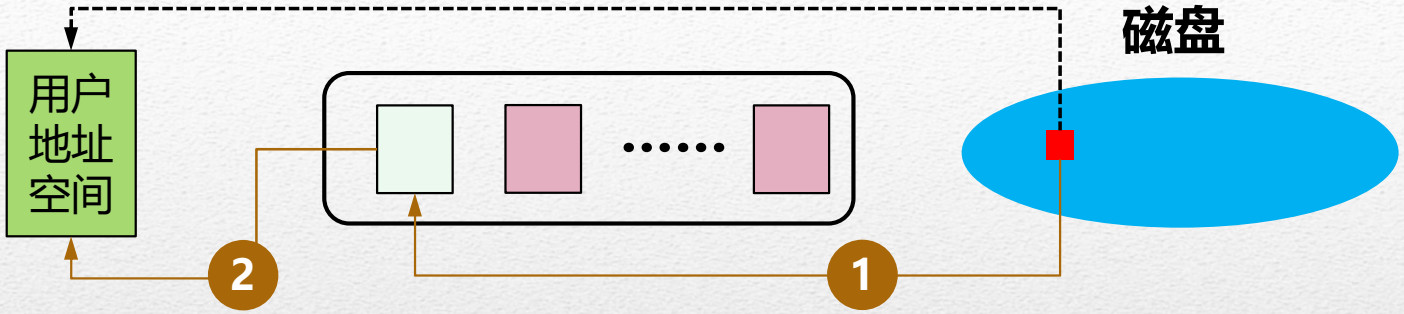
主 要 内 容

5.1 I/O系统

5.2 磁盘存储器管理

5.3 UNIX字符块设备管理

读操作

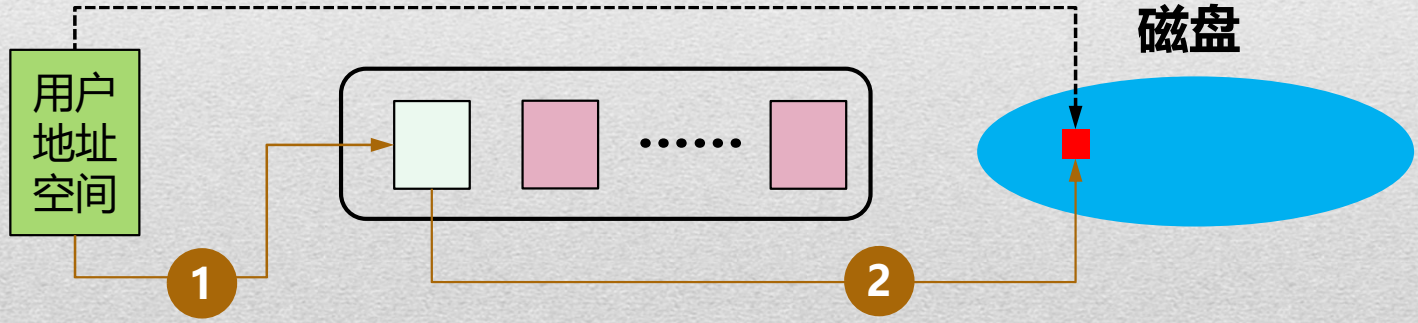


可反复读（重用）

不仅当前进程，

其他进程也可以

写操作

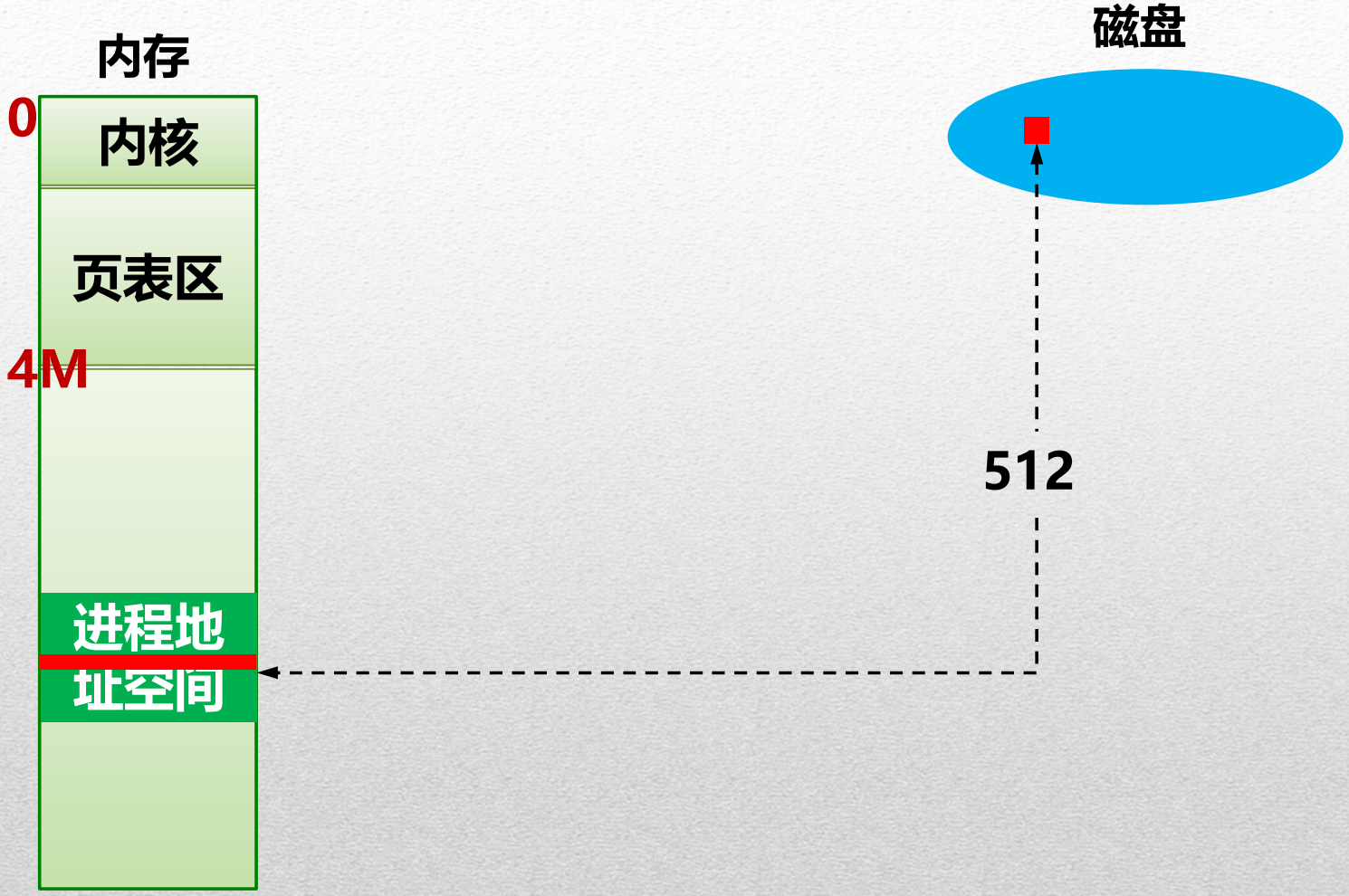


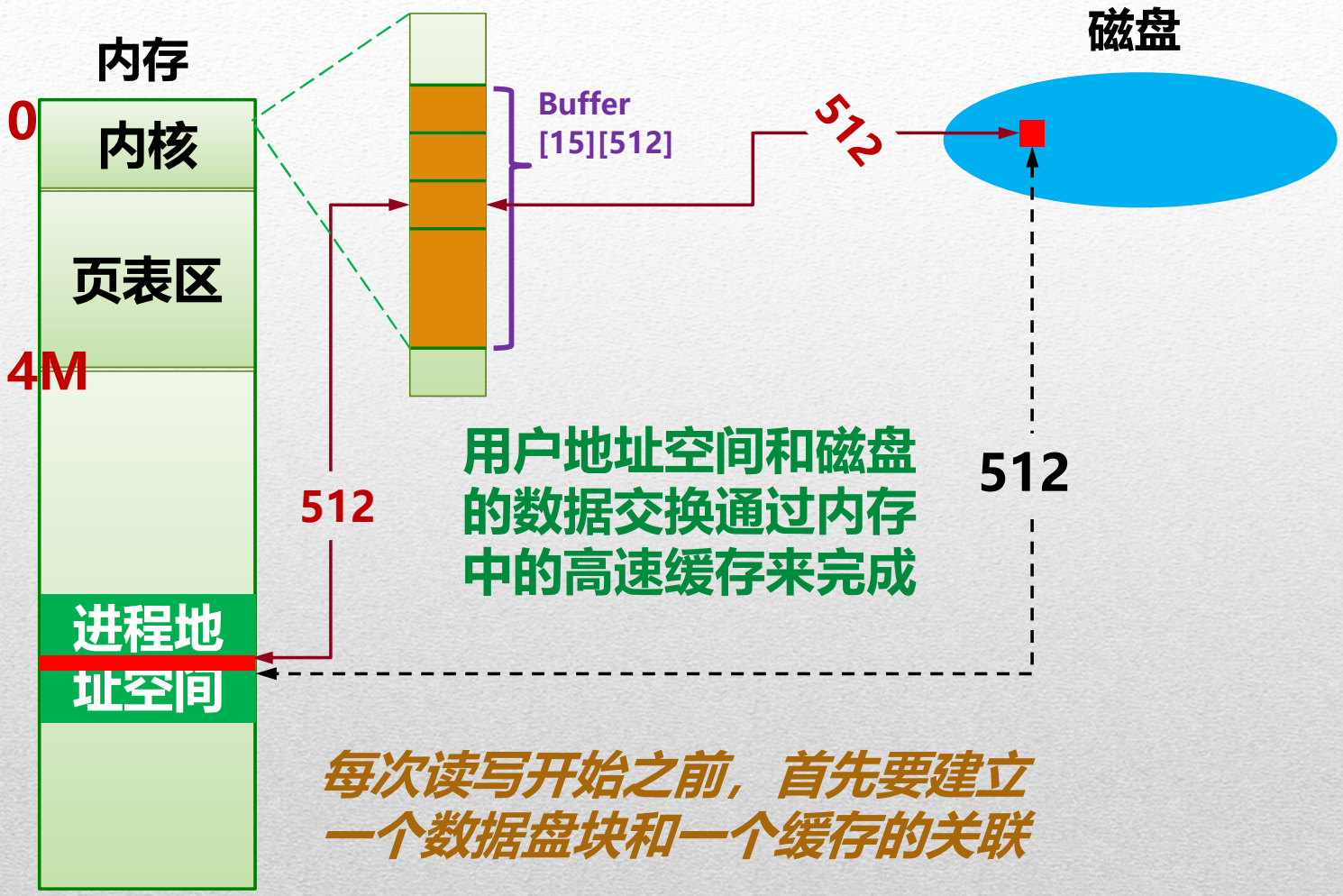
反复写（重用）写满后一次性写回

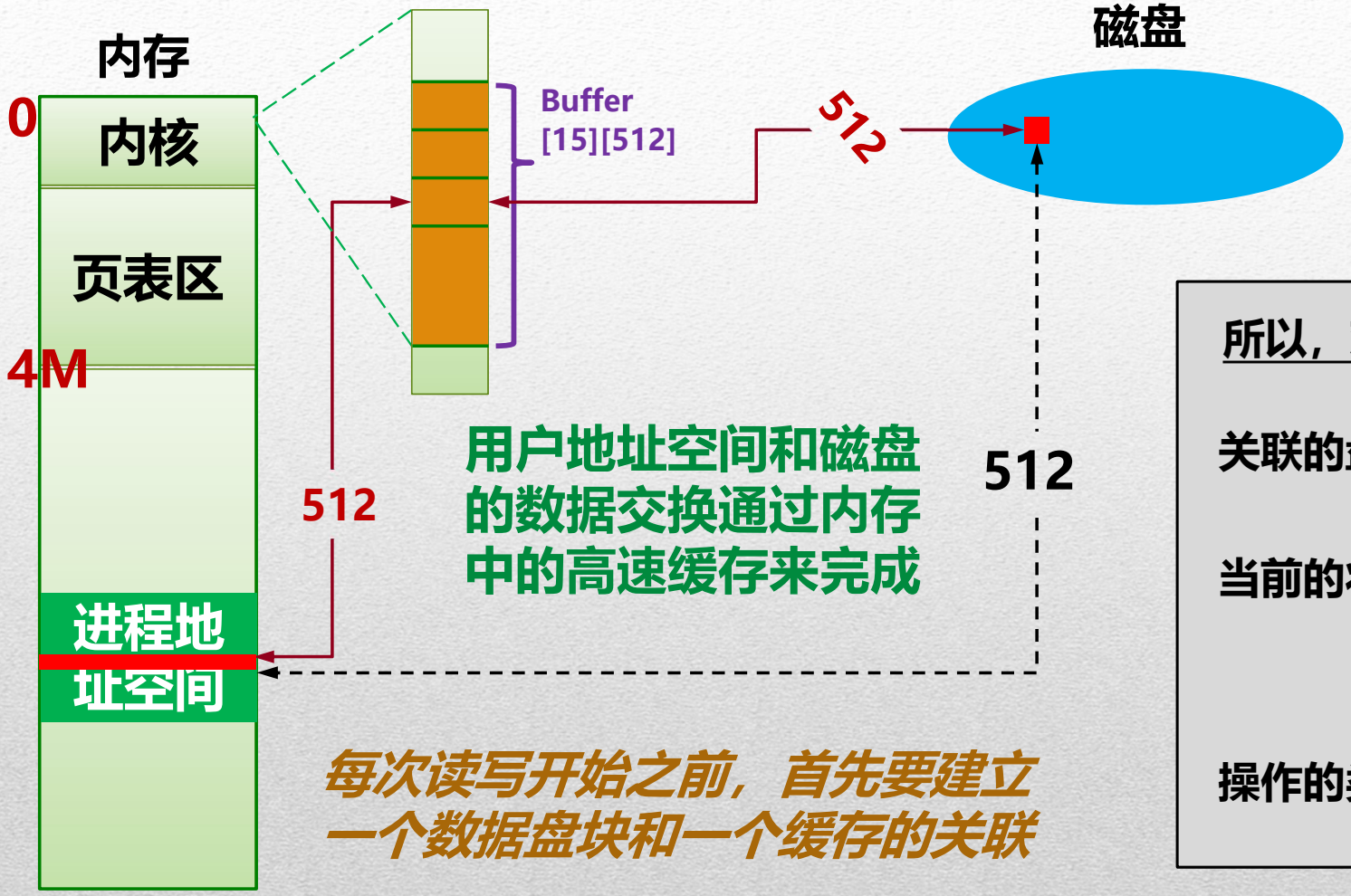
不仅当前进程，

其他进程也可以

特征：顺序，局部性 ⇨ 缓存内容尽量保存时间长一点


内存
0
内核
页表区
4M
进程地
址空间
磁盘
512


内存
0
内核
页表区
4M
进程地址空间
512
Buffer
[15][512]
512
磁盘
512
用户地址空间和磁盘的数据交换通过内存中的高速缓存来完成
每次读写开始之前，首先要建立一个数据盘块和一个缓存的关联



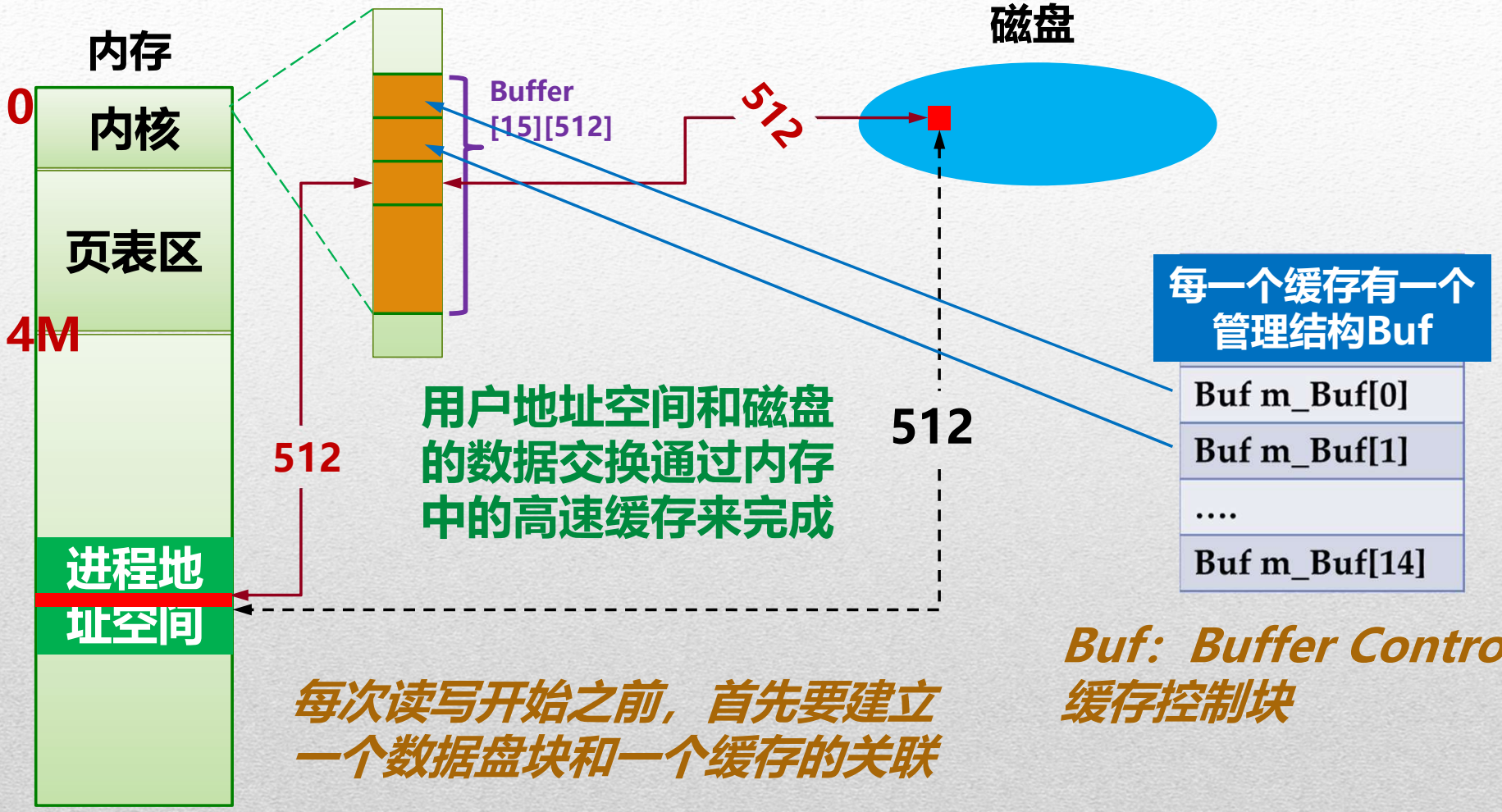
所以，对每一块缓存，需记录：

关联的盘块：　盘块地址

当前的状态：　忙？闲？
　　闲了多久？

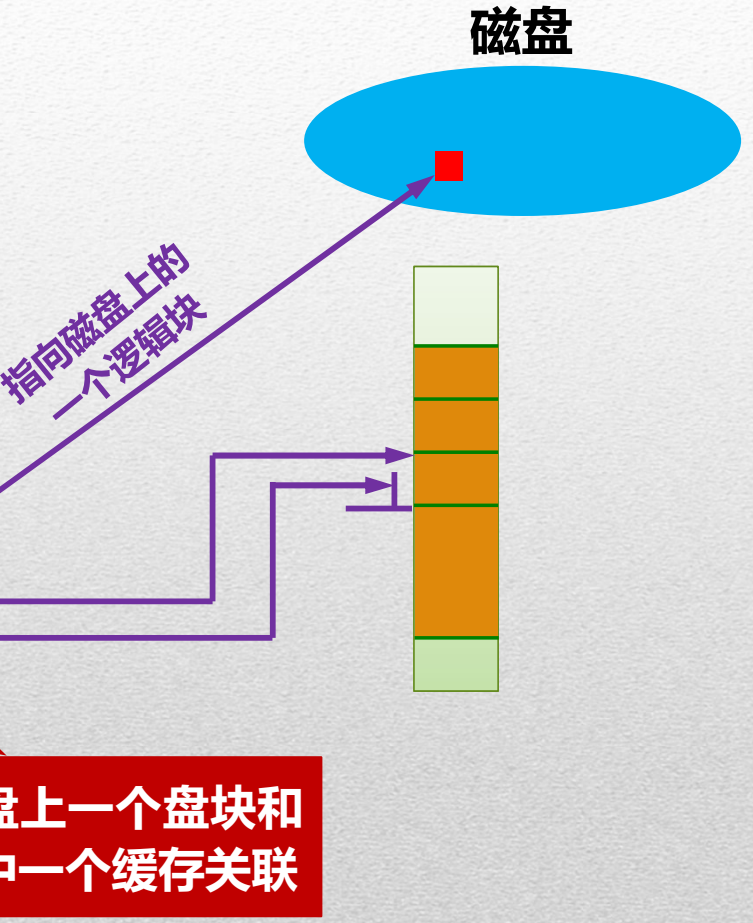
操作的类型：　读？　写？

每次读写开始之前，首先要建立一个数据盘块和一个缓存的关联


内存
0
内核
页表区
4M
进程地址空间
512
Buffer
[15][512]
磁盘
512
512
用户地址空间和磁盘的数据交换通过内存中的高速缓存来完成
每一个缓存有一个管理结构Buf
Buf m_Buf[0]
Buf m_Buf[1]
....
Buf m_Buf[14]
每次读写开始之前，首先要建立一个数据盘块和一个缓存的关联
Buf：Buffer Control Block
缓存控制块

Buf结构

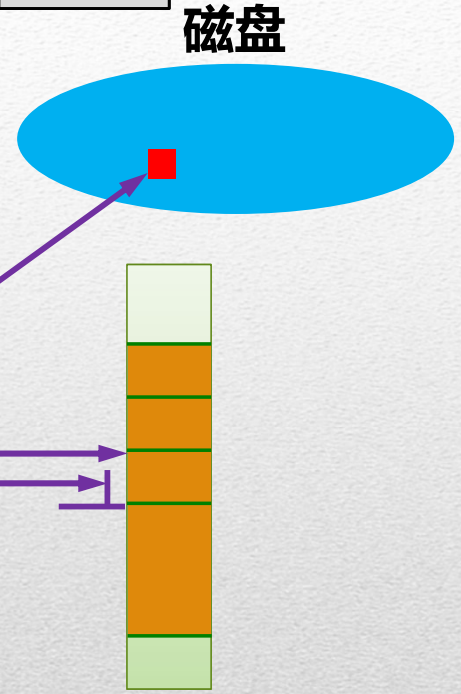
```
class Buf
{
public:
    enum BufFlag                /* b_flags中标志位 */
    {
        B_WRITE = 0x1,          /* 写操作。将缓存中的信息写到硬盘上去 */
        B_READ= 0x2,            /* 读操作。从盘读取信息到缓存中 */
        B_DONE= 0x4,            /* I/O操作结束 */
        B_ERROR= 0x8,           /* I/O因出错而终止 */
        B_BUSY= 0x10,           /* 相应缓存正在使用中 */
        B_WANTED = 0x20,        /* 有进程正在等待使用该buf管理的缓存*/
        B_ASYNC= 0x40,          /* 异步I/O，不需要等待其结束 */
        B_DELWRI = 0x80         /* 延迟写 */
    };
public:
    short b_dev;                /* 高、低8位分别是主、次设备号 */
    int b_blkno;                /* 磁盘逻辑块号 */
    unsigned char* b_addr;      /* 指向该缓存控制块管理的缓冲区首地址 */
    int b_wcount;               /* 需传送的字节数 */
    unsigned int b_flags;       /* 缓存控制块标志位 */
    int b_error;                /* I/O出错时信息 */
    int b_resid;                /* I/O出错时尚未传送的剩余字节数 */
    int padding;                /* 4字节填充，否则强制转换会出错。 */
    Buf* b_forw;                /* 缓存控制块队列勾连指针 */
    Buf* b_back;
    Buf* av_forw;
    Buf* av_back;
};
```



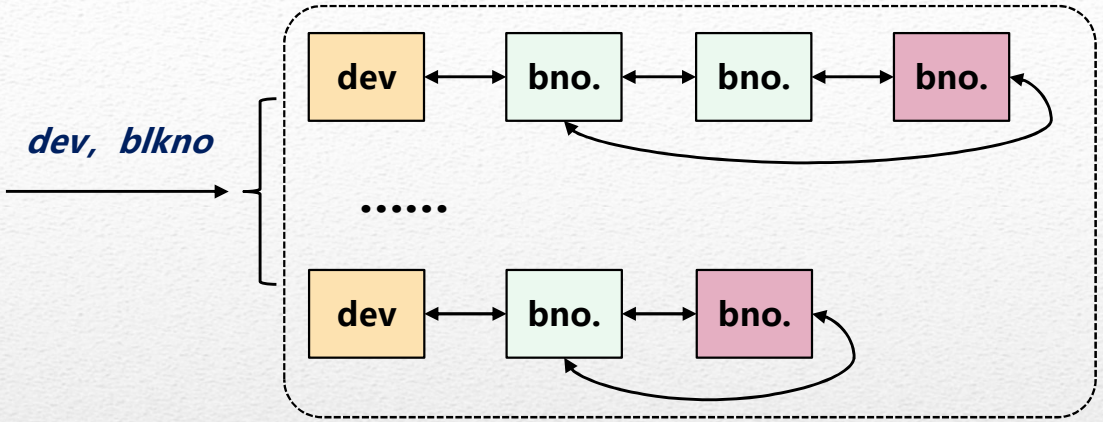
Buf结构

7	6	5	4	3	2	1	0
B_DELWRI	B_ASYNC	B_WANTED	B_BUSY	B_ERROR	B_DONE	B_READ	B_WRITE

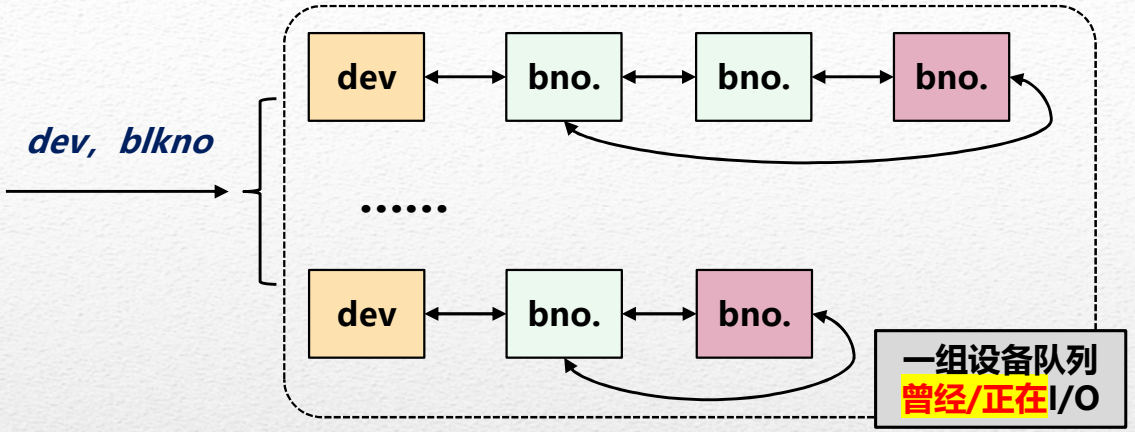
```
class Buf
{
public:
    enum BufFlag                /* b_flags中标志位 */
    {
        B_WRITE = 0x1,          /* 写操作。将缓存中的信息写到硬盘上去 */
        B_READ= 0x2,            /* 读操作。从盘读取信息到缓存中 */
        B_DONE= 0x4,            /* I/O操作结束 */
        B_ERROR= 0x8,           /* I/O因出错而终止 */
        B_BUSY= 0x10,           /* 相应缓存正在使用中 */
        B_WANTED = 0x20,        /* 有进程正在等待使用该buf管理的缓存*/
        B_ASYNC= 0x40,          /* 异步I/O，不需要等待其结束 */
        B_DELWRI = 0x80         /* 延迟写 */
    };
public:
    short b_dev;                /* 高、低8位分别是主、次设备号 */
    int b_blkno;                /* 磁盘逻辑块号 */
    unsigned char* b_addr;      /* 指向该缓存控制块管理的缓冲区首地址 */
    int b_wcount;               /* 需传送的字节数 */
    unsigned int b_flags;       /* 缓存控制块标志位 */
    int b_error;                /* I/O出错时信息 */
    int b_resid;                /* I/O出错时尚未传送的剩余字节数 */
    int padding;                /* 4字节填充，否则强制转换会出错。 */
    Buf* b_forw;                /* 缓存控制块队列勾连指针 */
    Buf* b_back;
    Buf* av_forw;
    Buf* av_back;
};
```



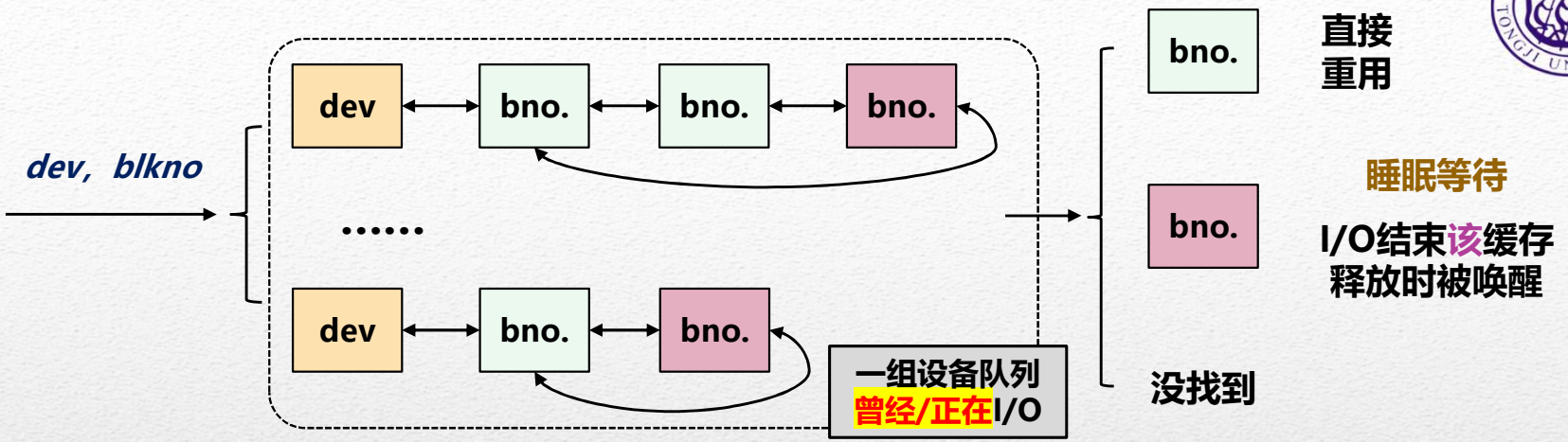
1 可以重用么?



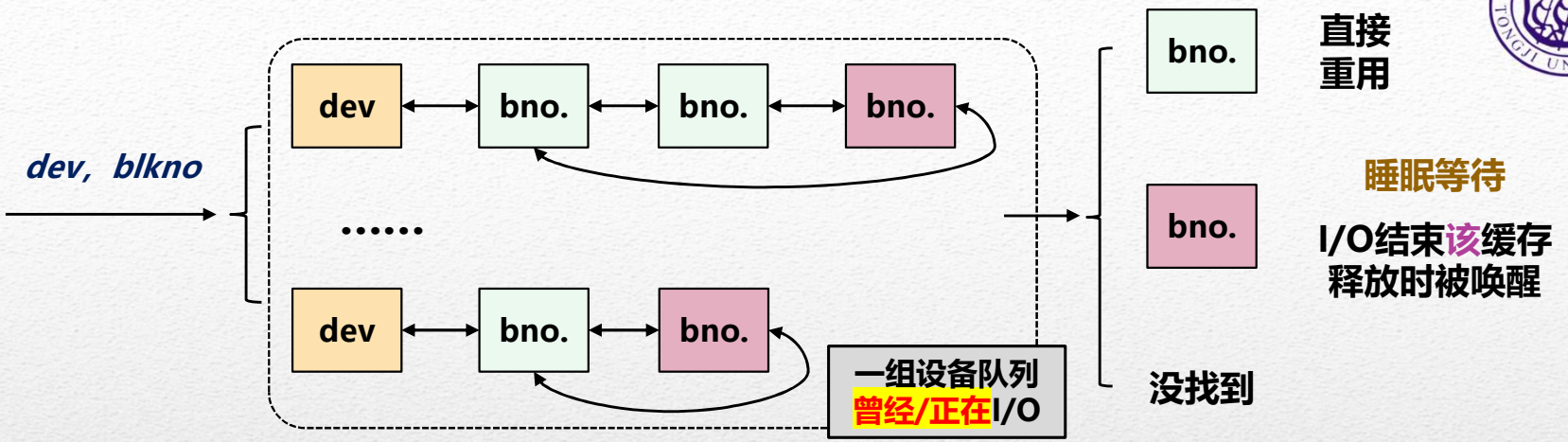
1 可以重用么?



1 可以重用么?

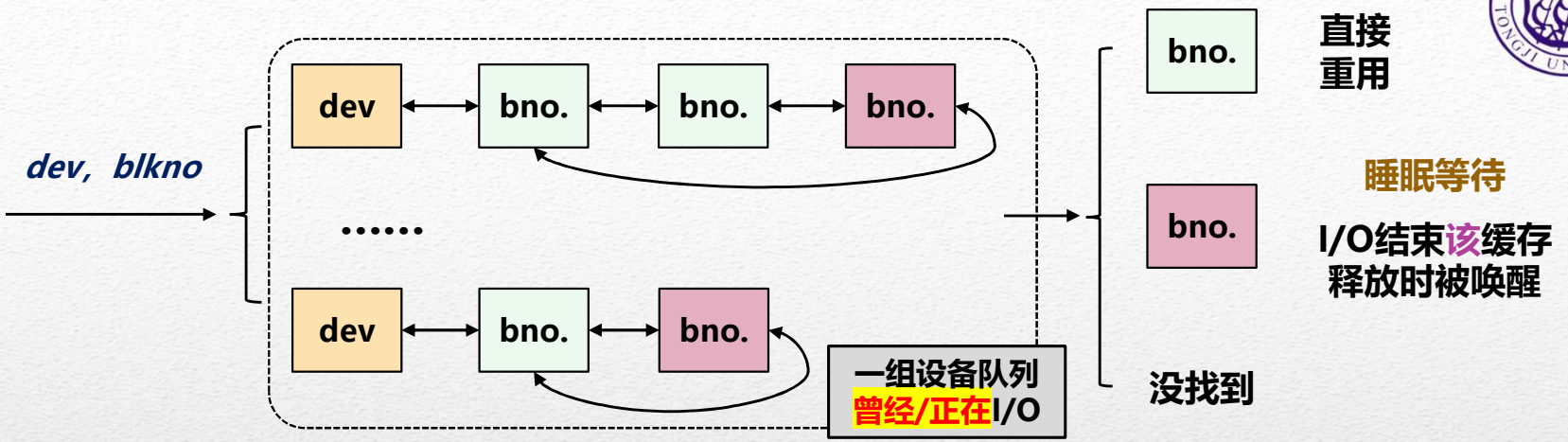

dev, blkno
dev
bno.
bno.
bno.
.....
dev
bno.
bno.
一组设备队列
曾经/正在I/O
bno.
bno.
没找到
直接
重用
睡眠等待
I/O结束该缓存
释放时被唤醒

1 可以重用么?

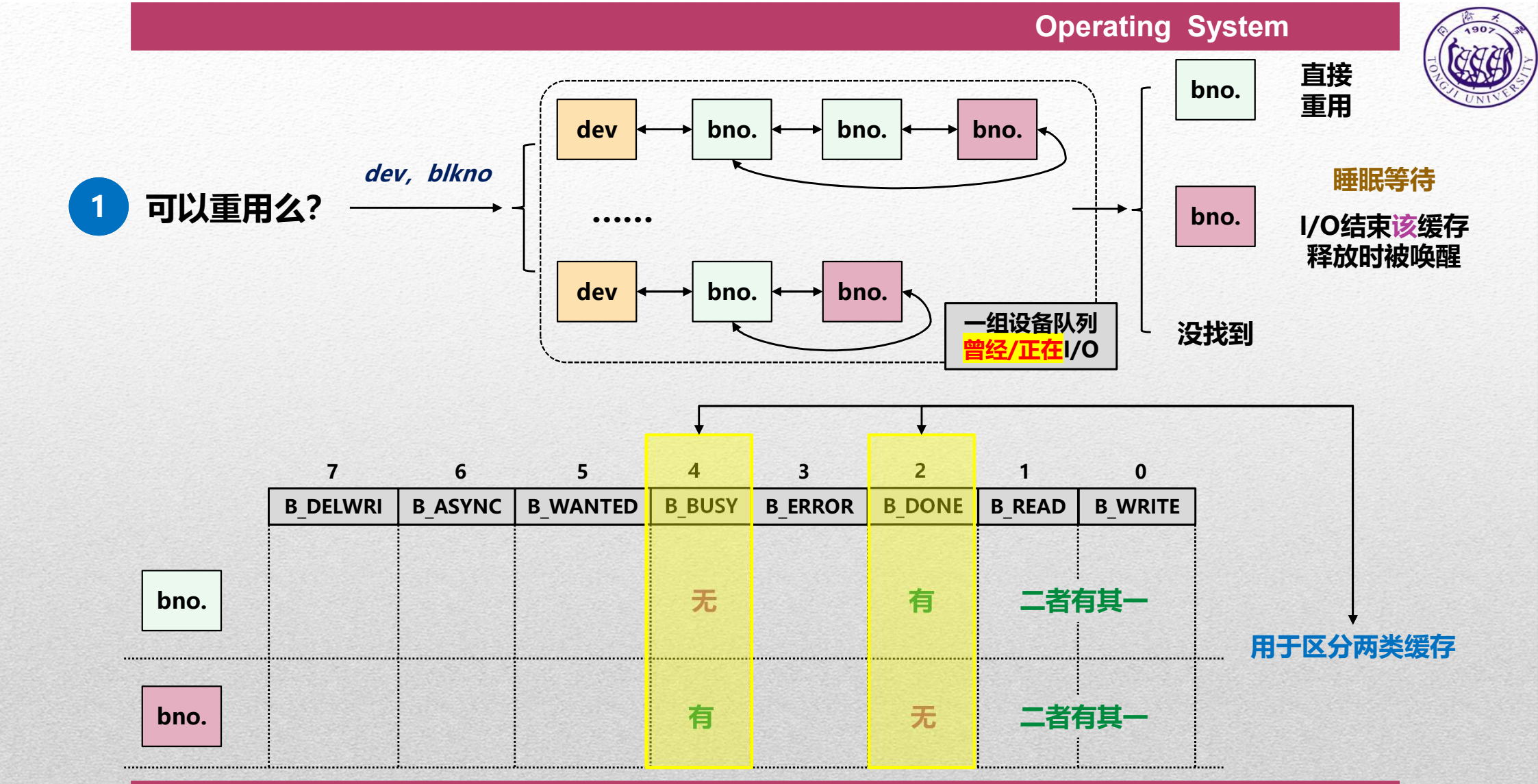


	7 B_DELWRI	6 B_ASYNC	5 B_WANTED	4 B_BUSY	3 B_ERROR	2 B_DONE	1 B_READ	0 B_WRITE
bno.	“延迟写”时有	“异步”时有	“等重用”时有	无	出错时有	有	二者有其一	
bno.	无	“异步”时有	“等重用”时有	有	无	无	二者有其一	

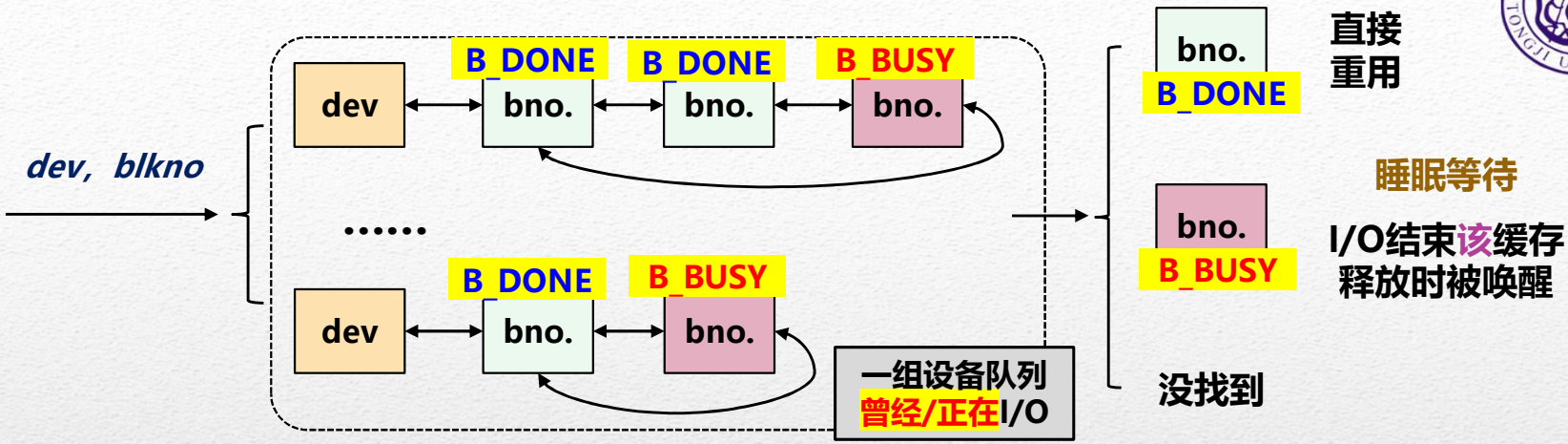
1 可以重用么?

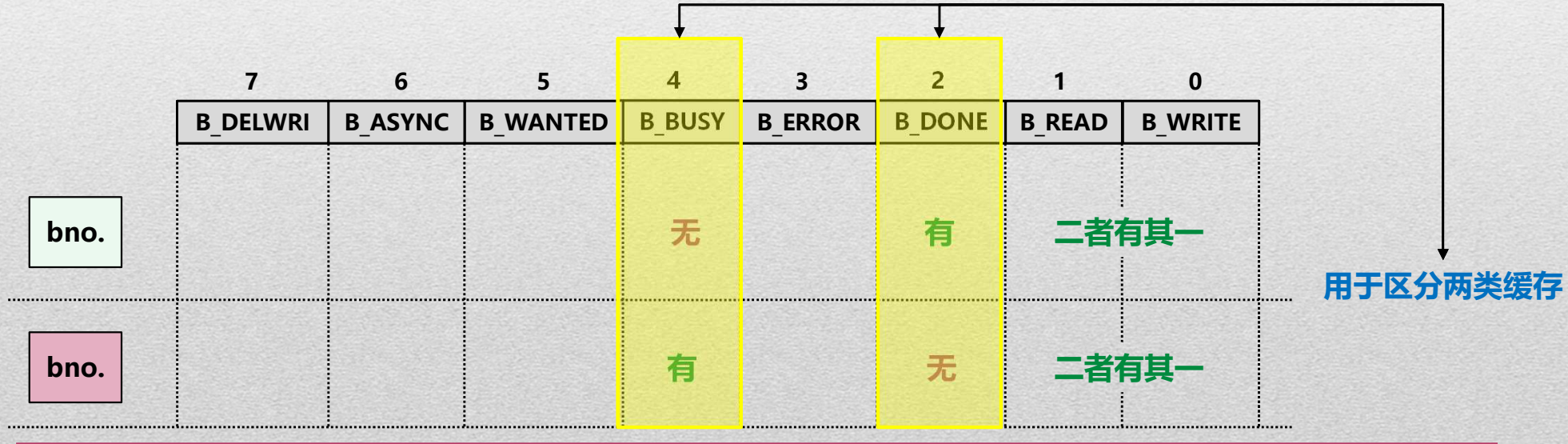


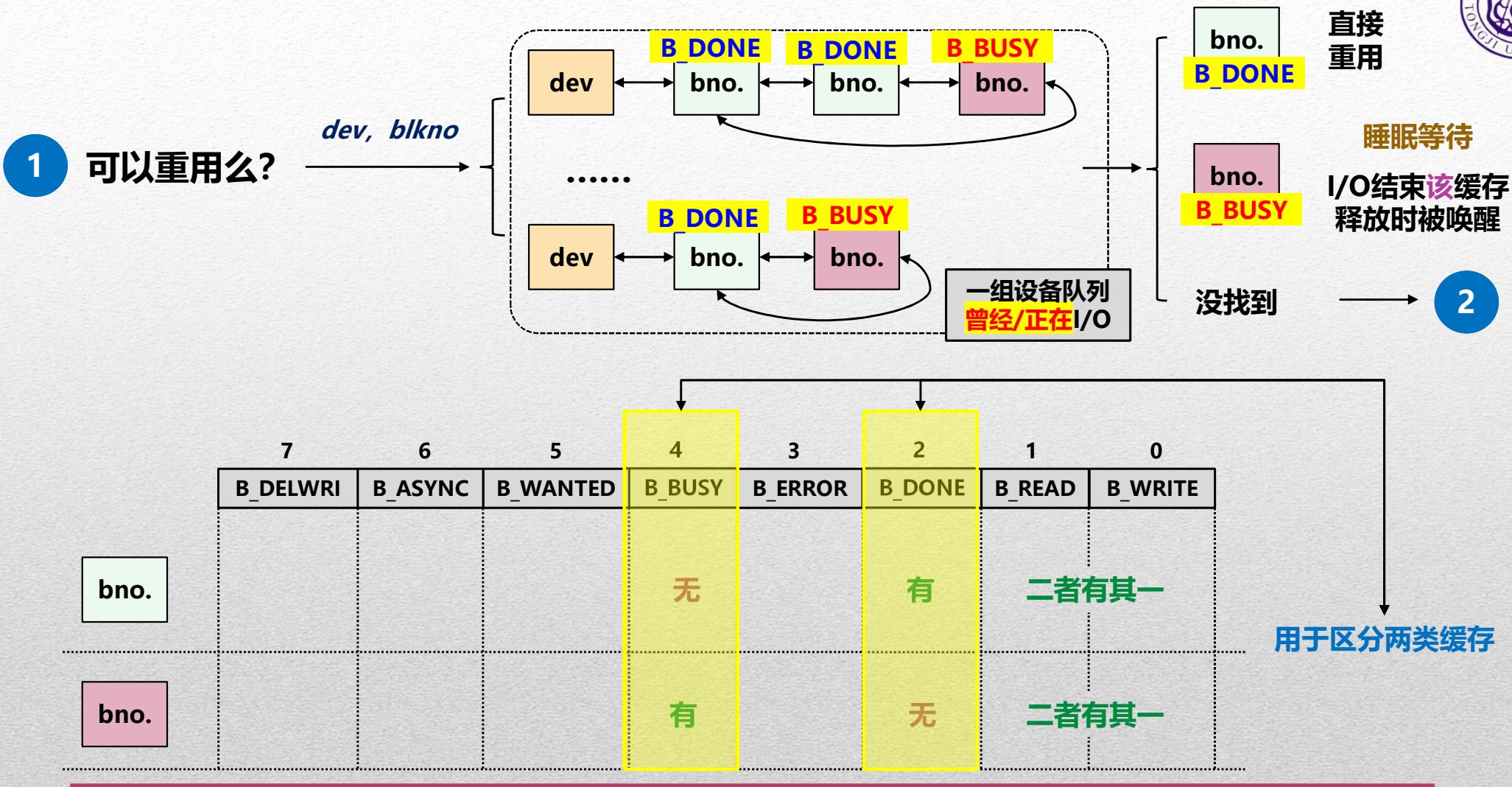
	7	6	5	4	3	2	1	0
	B_DELWRI	B_ASYNC	B_WANTED	B_BUSY	B_ERROR	B_DONE	B_READ	B_WRITE
bno.				无		有	二者有其一	
bno.				有		无	二者有其一	


1
可以重用么?
dev, blkno
dev
bno.
bno.
bno.
.....
dev
bno.
bno.
一组设备队列
曾经/正在I/O
bno.
bno.
没找到
直接
重用
睡眠等待
I/O结束该缓存
释放时被唤醒
7
6
5
4
3
2
1
0
B_DELWRI
B_ASYNC
B_WANTED
B_BUSY
B_ERROR
B_DONE
B_READ
B_WRITE
bno.
无
有
二者有其一
bno.
有
无
二者有其一
用于区分两类缓存

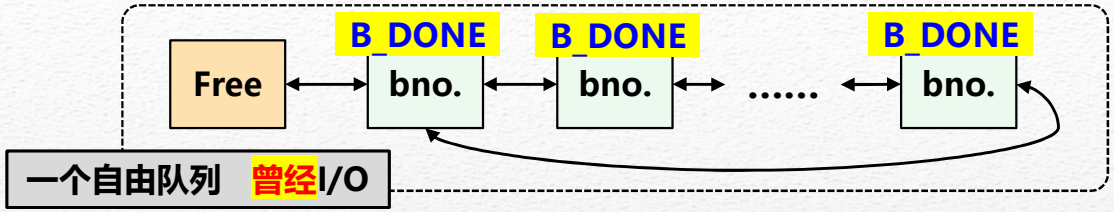
1 可以重用么?


dev, blkno
dev
B_DONE
bno.
B_DONE
bno.
B_BUSY
bno.
.....
dev
B_DONE
bno.
B_BUSY
bno.
一组设备队列
曾经/正在I/O
bno.
B_DONE
直接
重用
睡眠等待
bno.
B_BUSY
I/O结束该缓存
释放时被唤醒
没找到

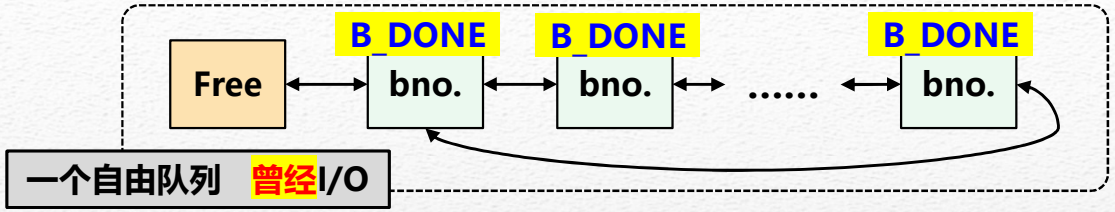

7
6
5
4
3
2
1
0
B_DELWRI
B_ASYNC
B_WANTED
B_BUSY
B_ERROR
B_DONE
B_READ
B_WRITE
bno.
无
有
二者有其一
bno.
有
无
二者有其一
用于区分两类缓存


1
可以重用么?
dev, blkno
dev
B_DONE
bno.
B_DONE
bno.
B_BUSY
bno.
.....
dev
B_DONE
bno.
B_BUSY
bno.
一组设备队列
曾经/正在I/O
bno.
B_DONE
直接
重用
睡眠等待
bno.
B_BUSY
I/O结束该缓存
释放时被唤醒
没找到
2
7
6
5
4
3
2
1
0
B_DELWRI
B_ASYNC
B_WANTED
B_BUSY
B_ERROR
B_DONE
B_READ
B_WRITE
bno.
无
有
二者有其一
bno.
有
无
二者有其一
用于区分两类缓存

2 找可以置换的缓存

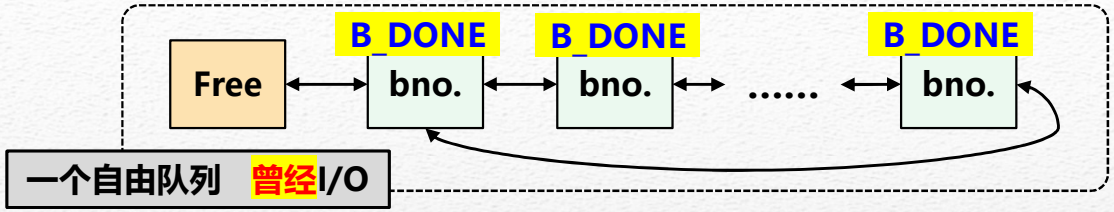


2 找可以置换的缓存



怎么找到最久未使用的缓存?

2 找可以置换的缓存

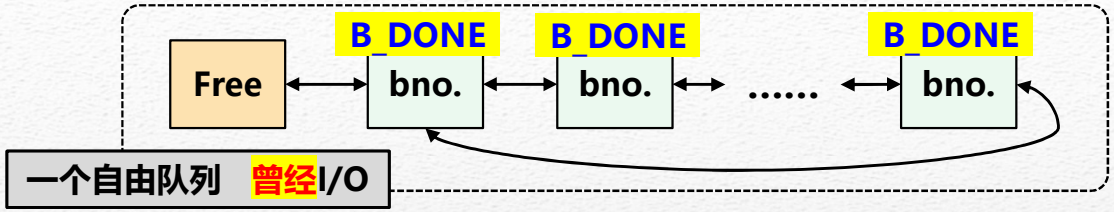


怎么找到最久未使用的缓存?

每次缓存使用（重新分配，重用）完毕，从队尾重新排队

每次分配缓存，从队头开始分配

2 找可以置换的缓存

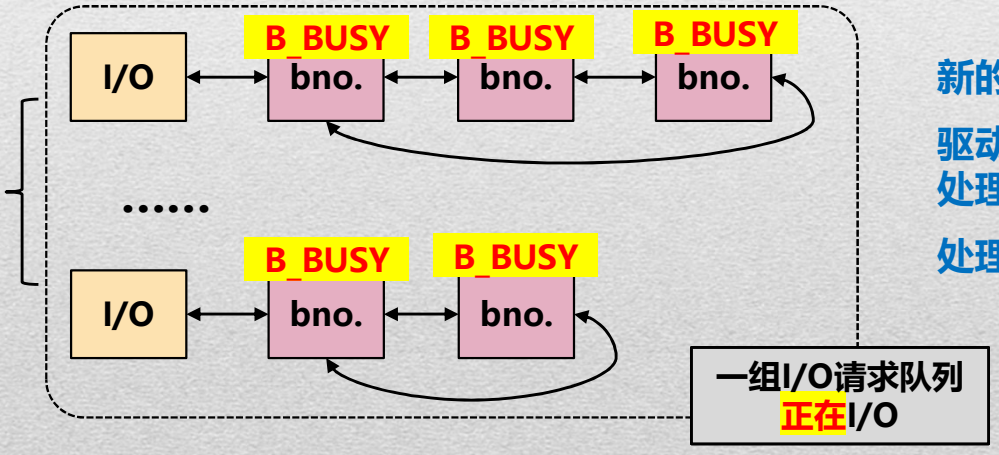


怎么找到最久未使用的缓存?

每次缓存使用（重新分配，重用）完毕，从队尾重新排队

每次分配缓存，从队头开始分配

3 为了I/O操作管理的方便



新的I/O操作挂在队尾

驱动程序从对头开始依次处理

处理完摘下

Buf结构

```
class Buf
{
public:
    enum BufFlag              /* b_flags中标志位
    {
        B_WRITE = 0x1,        /* 写操作。将缓存中
        B_READ= 0x2,          /* 读操作。从盘读取
        B_DONE= 0x4,          /* I/O操作结束 */
        B_ERROR= 0x8,         /* I/O因出错而终止 */
        B_BUSY= 0x10,         /* 相应缓存正在使用中 */
        B_WANTED = 0x20,      /* 有进程正在等待使用该buf管理的缓存*/
        B_ASYNC= 0x40,        /* 异步I/O，不需要等待其结束 */
        B_DELWRI = 0x80       /* 延迟写 */
    };
public:
    short b_dev;              /* 高、低8位分别是主、次设备号 */
    int b_blkno;              /* 磁盘逻辑块号 */
    unsigned char* b_addr;    /* 指向该缓存控制块管理的缓冲区首地址 */
    int b_wcount;             /* 需传送的字节数 */
    unsigned int b_flags;     /* 缓存控制块标志位 */
    int b_error;              /* I/O出错时信息 */
    int b_resid;              /* I/O出错时尚未传送的剩余字节数 */
    int padding;              /* 4字节填充，否则强制转换会出错。 */
    Buf* b_forw;              /* 缓存控制块队列勾连指针 */
    Buf* b_back;
    Buf* av_forw;
    Buf* av_back;
};
```

Buf* b_forw;
Buf* b_back;
Buf* av_forw;
Buf* av_back;

不用于I/O　　曾经/正在I/O

将Buf插入NODEV队列或某一设备队列

曾经I/O　　正在I/O

将Buf插入自由队列或某一I/O请求队列

dev
I/O

每一个设备的设备队列和I/O请求队列共用一个表头

读写时： 先找本设备队列，找到利用。找不到再从自由队首分配。
（*从其他设备队列中摘下*）

读写完毕：从I/O请求队列中摘下，释放到自由队尾。
（*但仍留在原设备队列中*）

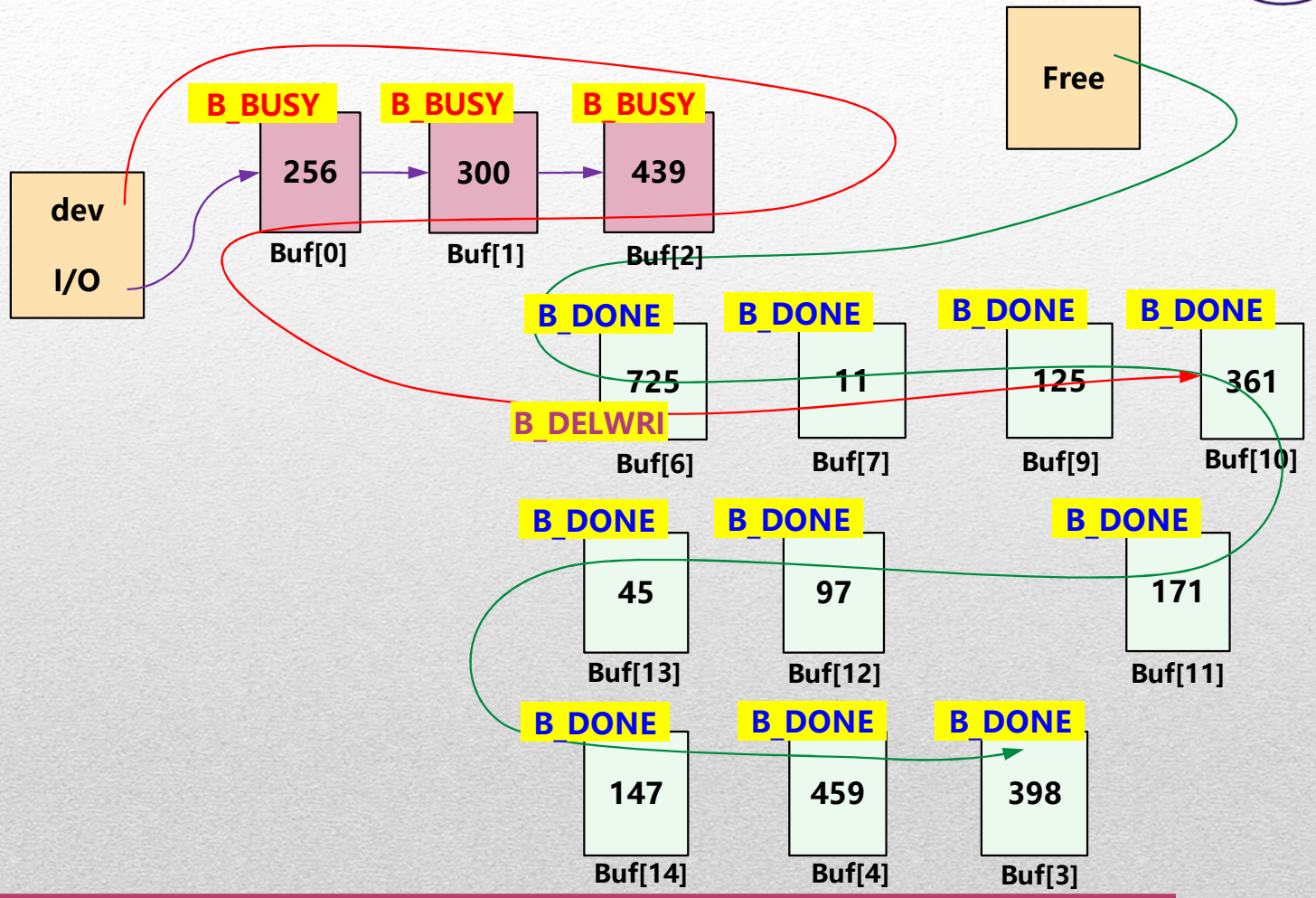
read( ) 系统调用时，通过文件系统得到dev , blkno

BufferManager:: Bread(dev, blkno)

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

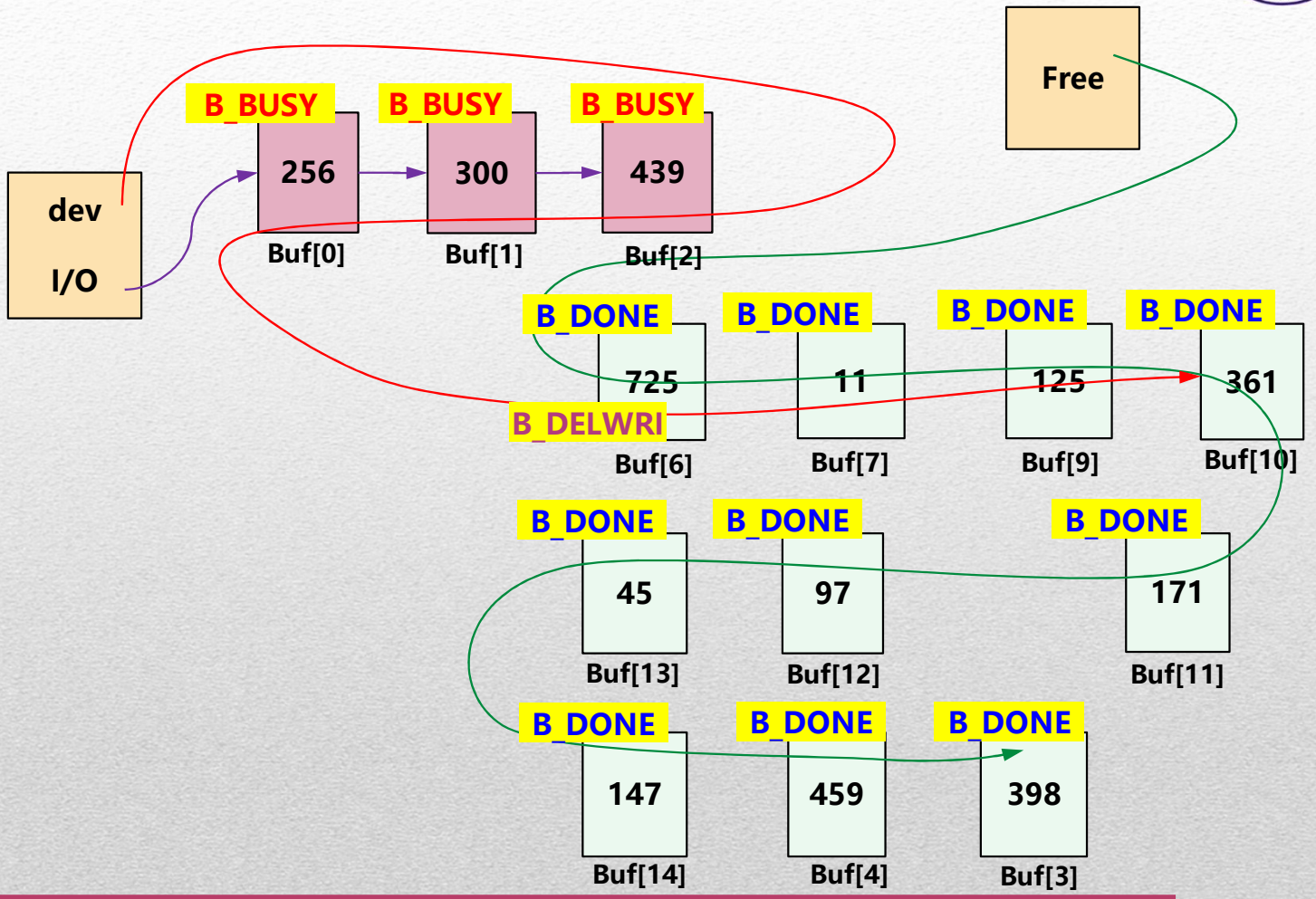
按同步方式将字符块读入内存

BufferManager::GetBlk(dev, blkno)过程



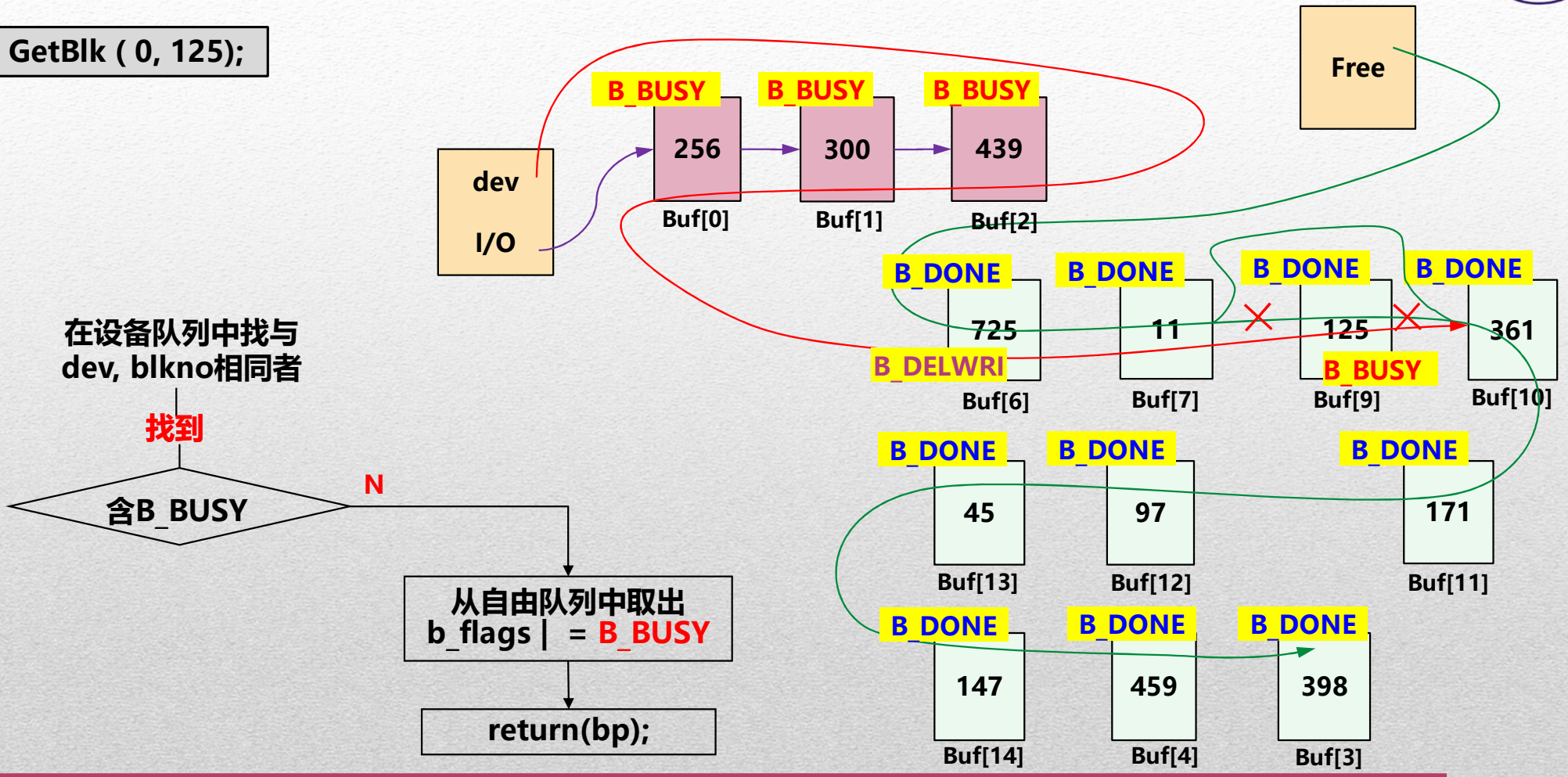
BufferManager::GetBlk(dev, blkno)过程

1. GetBlk (0, 125);



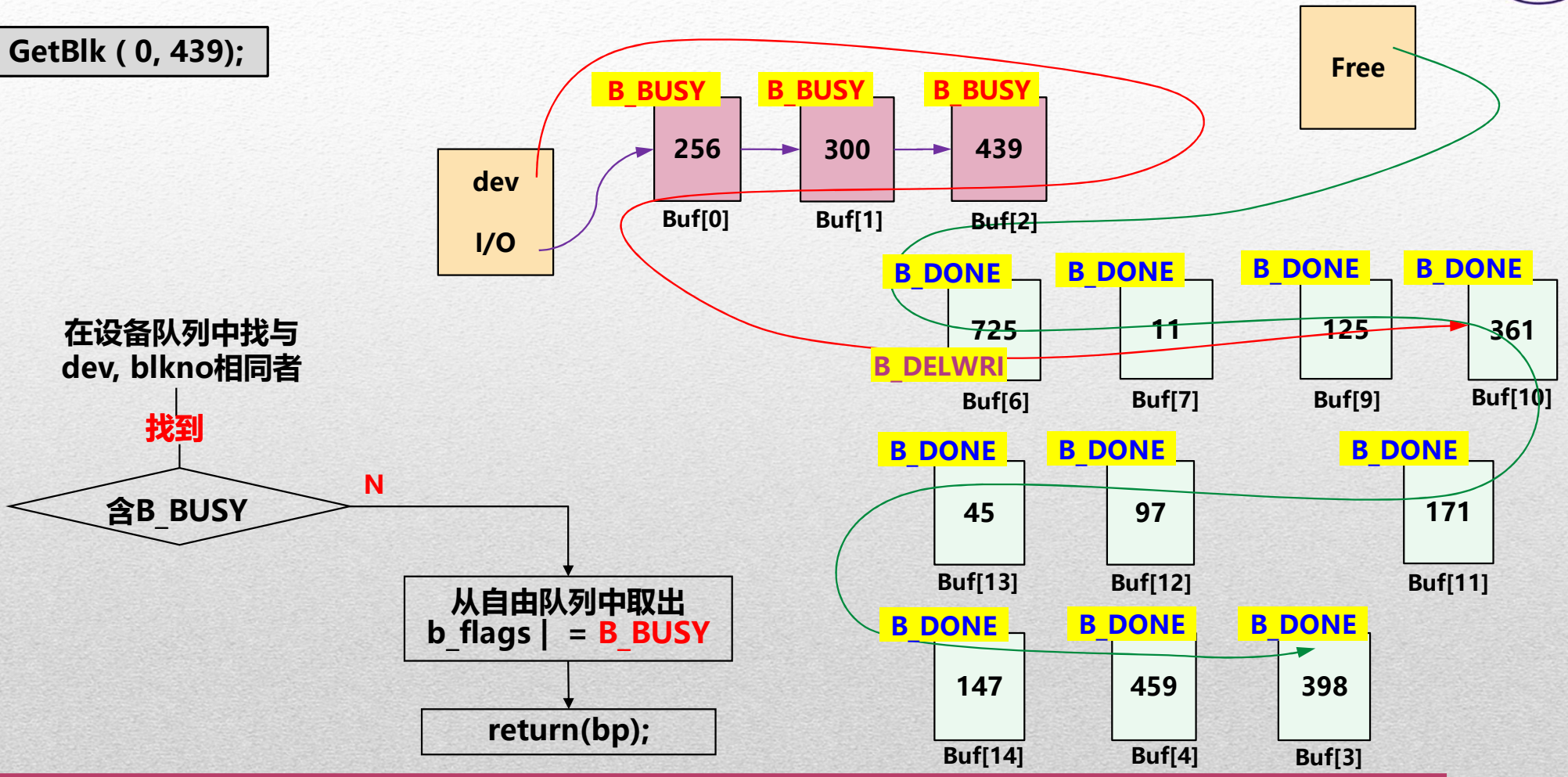
BufferManager::GetBlk(dev, blkno)过程

1. GetBlk (0, 125);



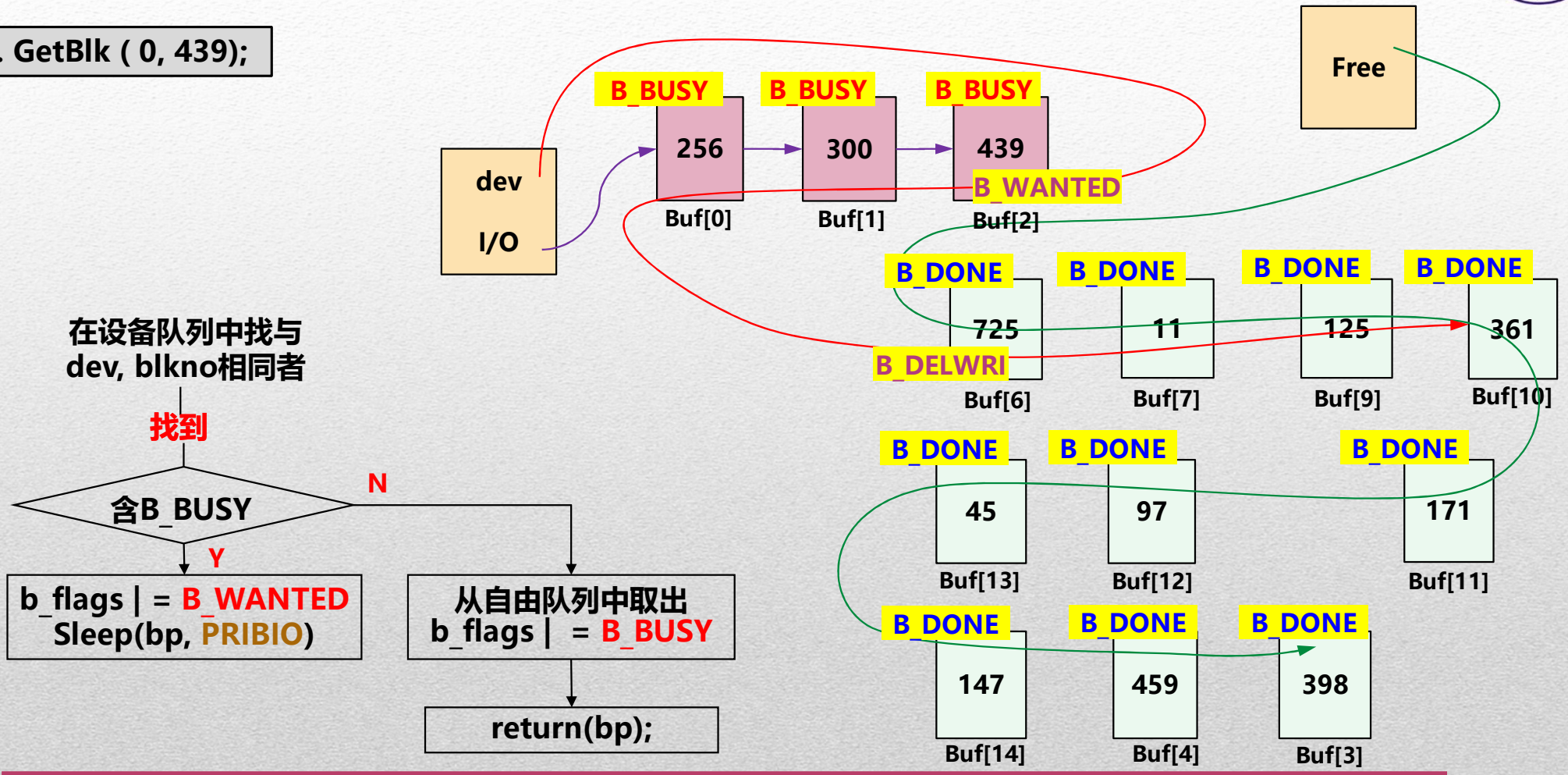
BufferManager::GetBlk(dev, blkno)过程

2. GetBlk (0, 439);



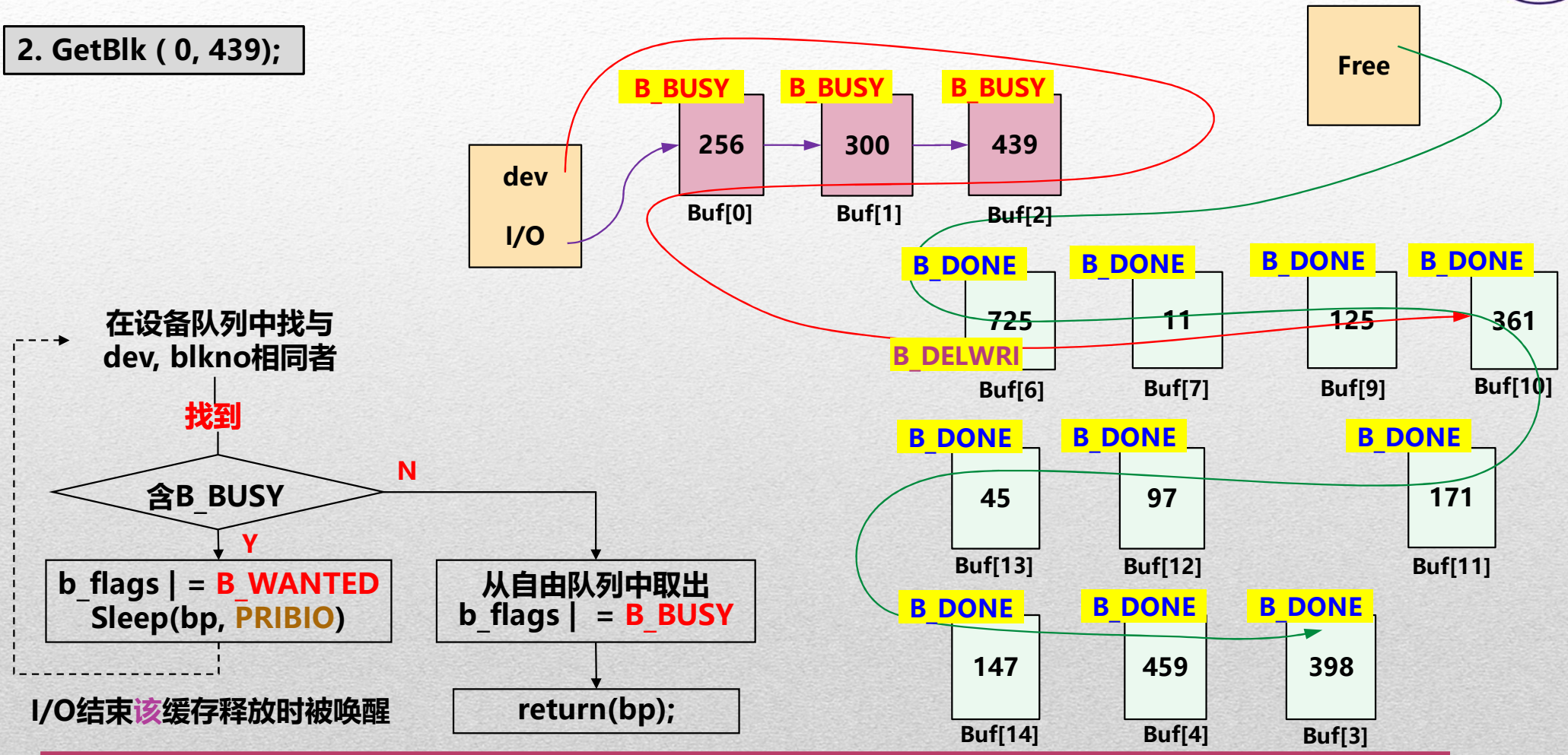
BufferManager::GetBlk(dev, blkno)过程

2. GetBlk (0, 439);



BufferManager::GetBlk(dev, blkno)过程

2. GetBlk (0, 439);



read( ) 系统调用时，通过文件系统得到dev , blkno

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

按同步方式将字符块读入内存

含 B_DONE ?

Y

所需信息已在缓存中

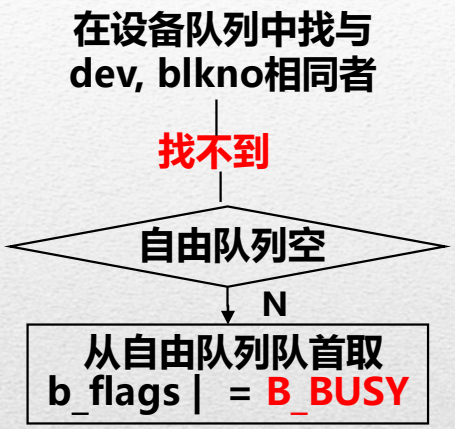
可直接利用！

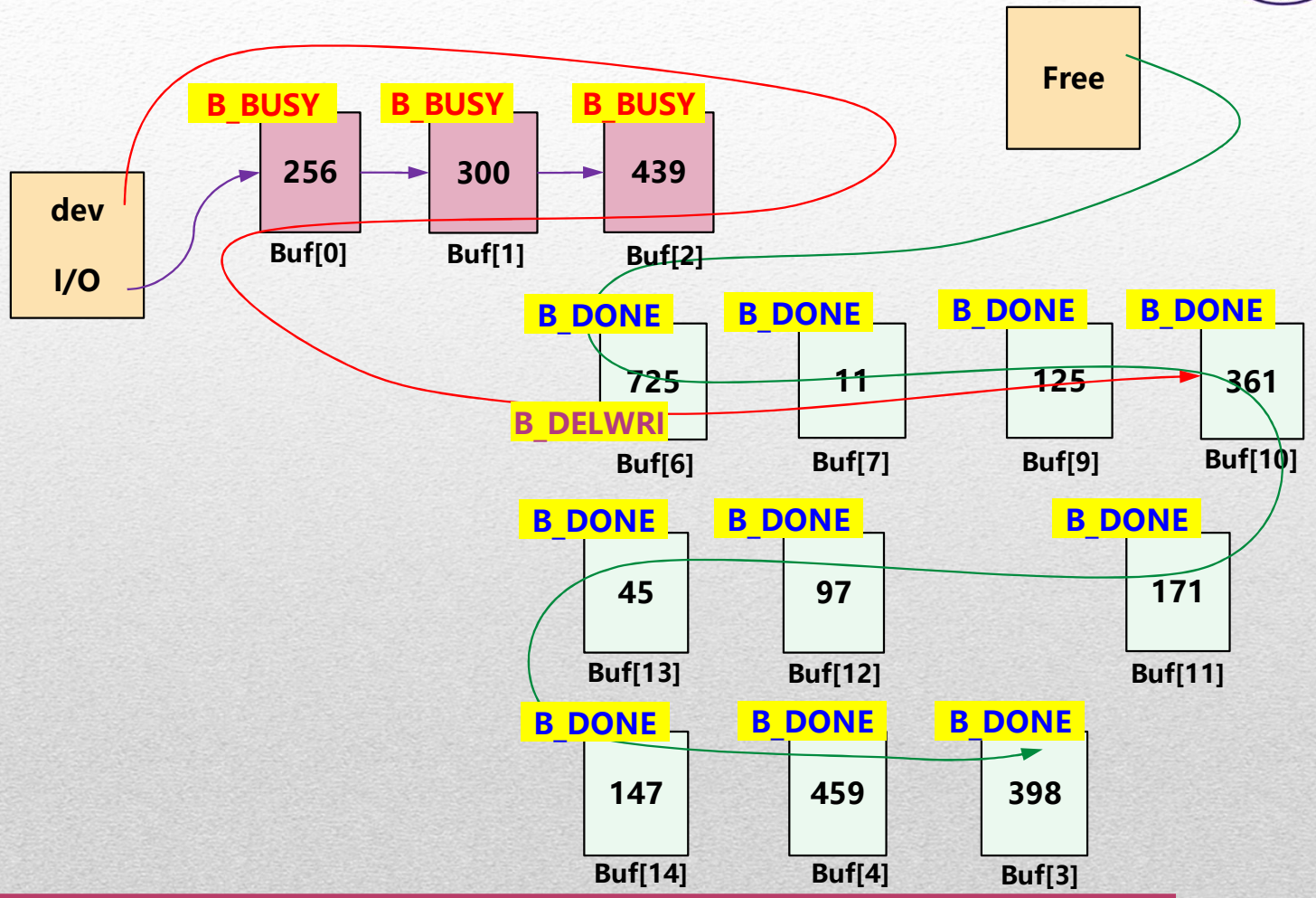
进程没有因为I/O操作睡觉，缓存中的内容由文件系统使用完之后释放到自由队列的队尾!

返回(bp)

BufferManager::GetBlk(dev, blkno)过程

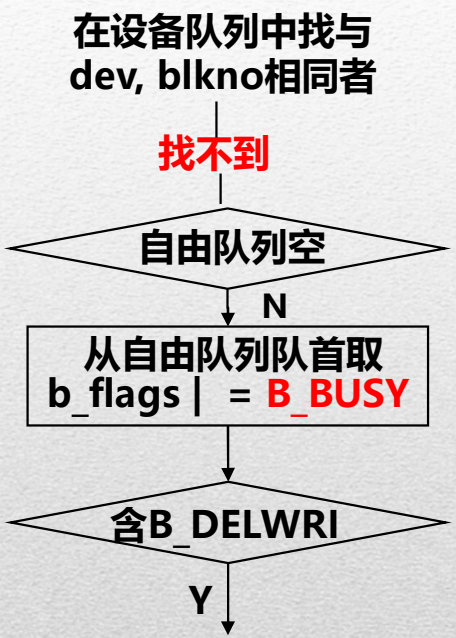
3. GetBlk (0, 100);

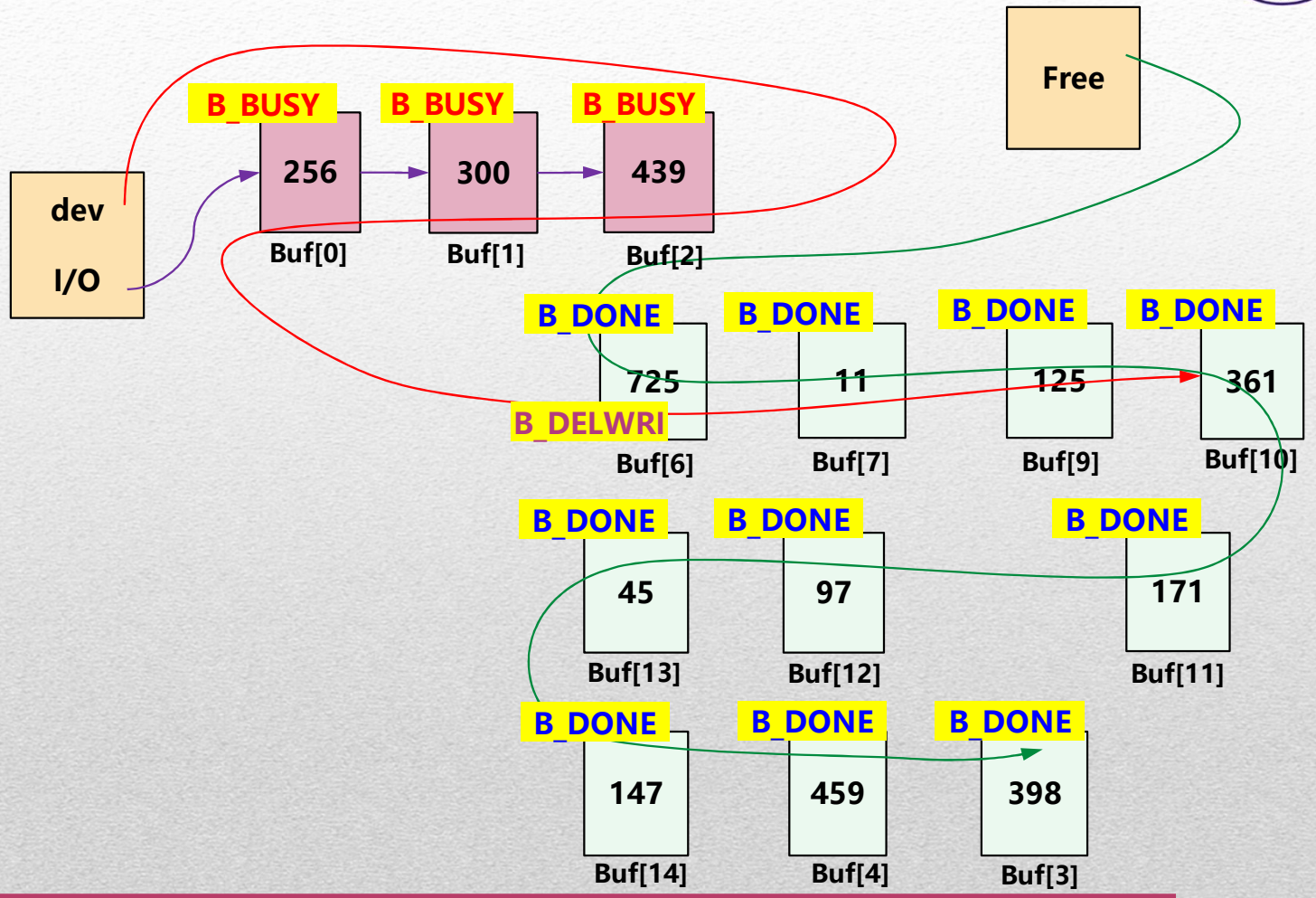




BufferManager::GetBlk(dev, blkno)过程

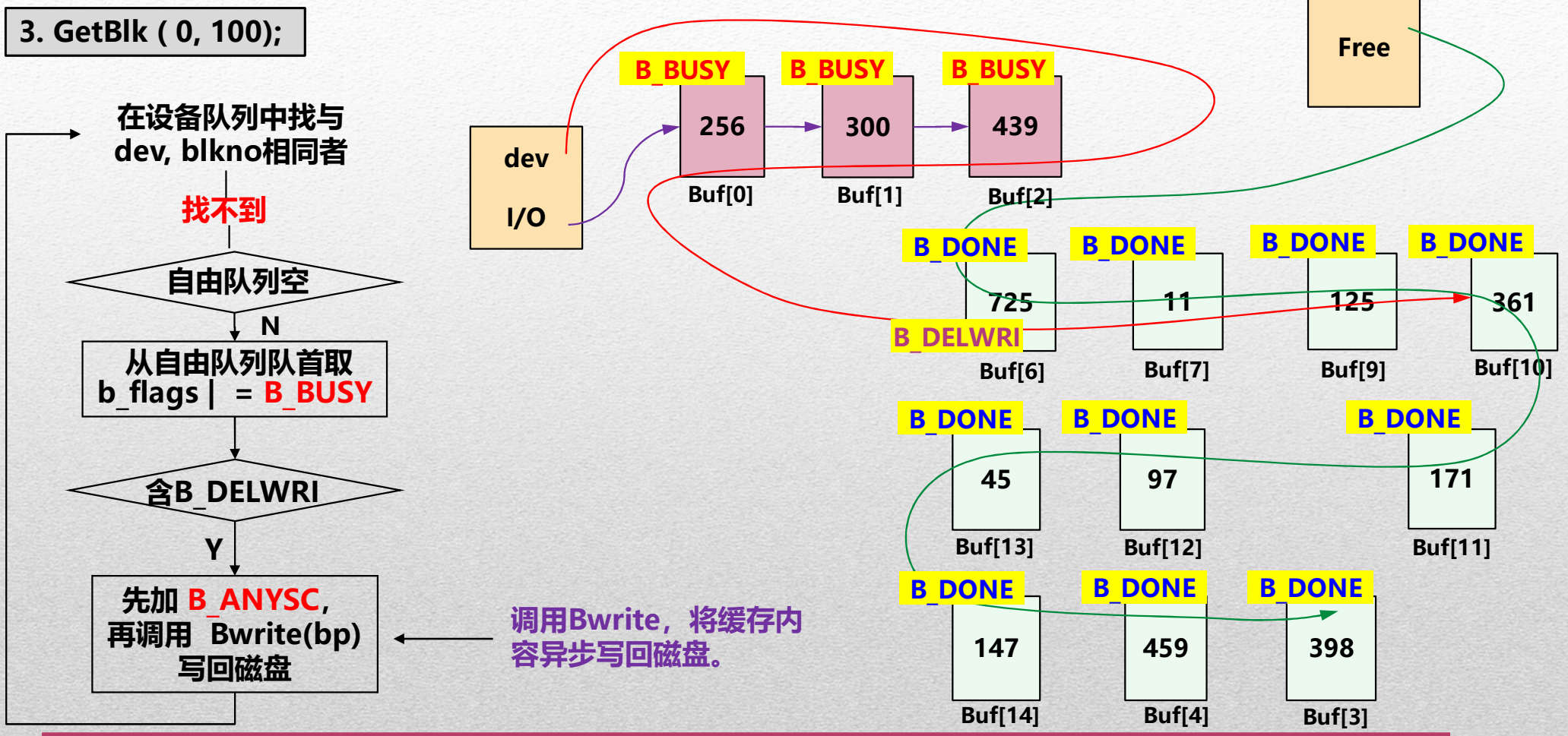
3. GetBlk (0, 100);





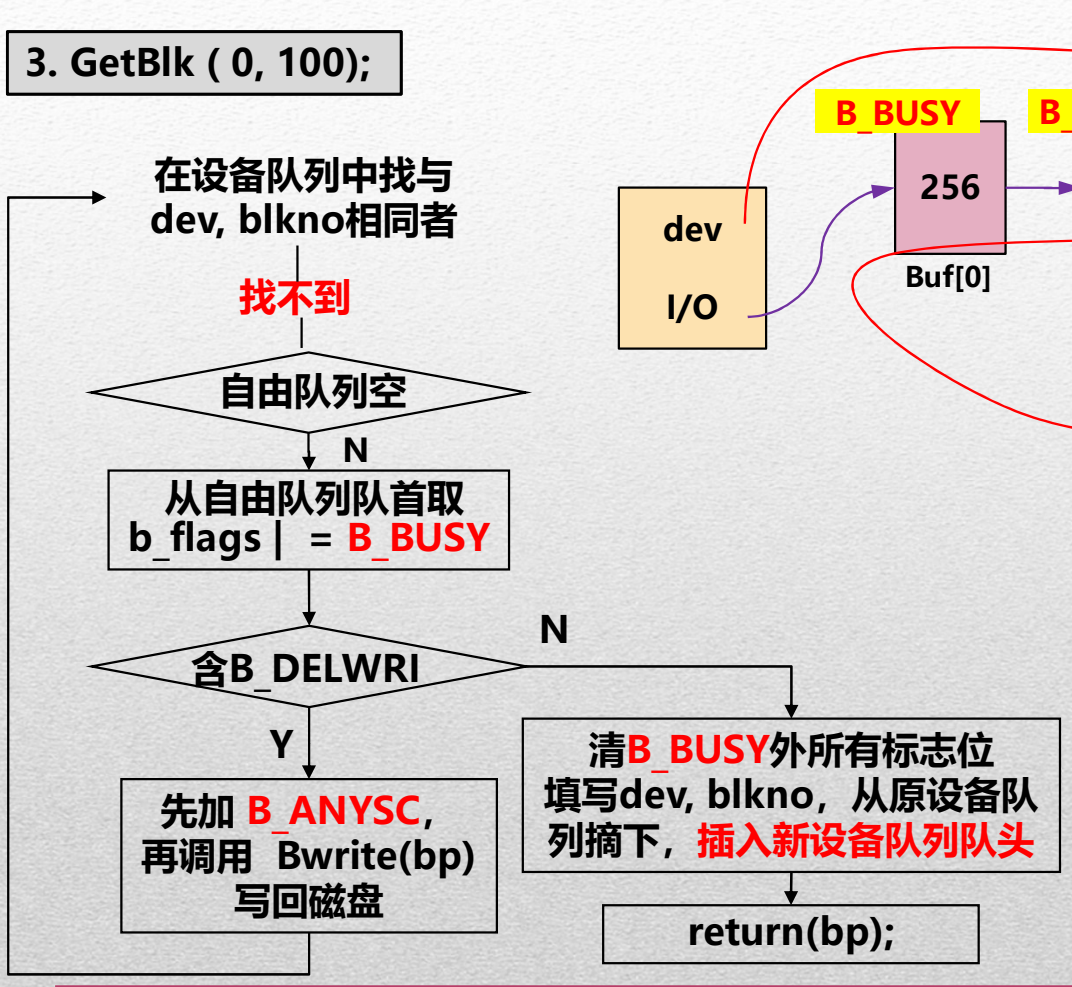
BufferManager::GetBlk(dev, blkno)过程

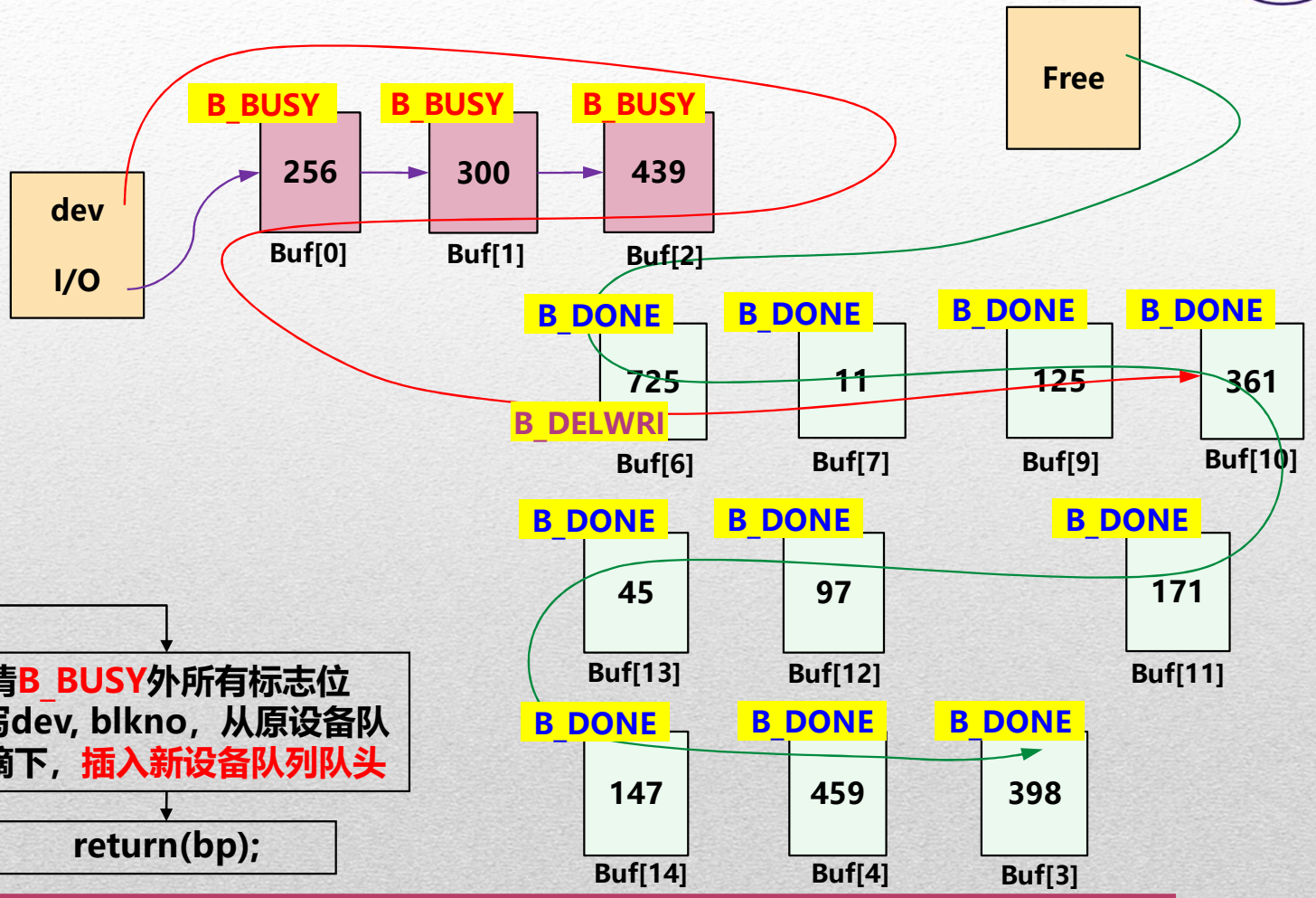
3. GetBlk (0, 100);



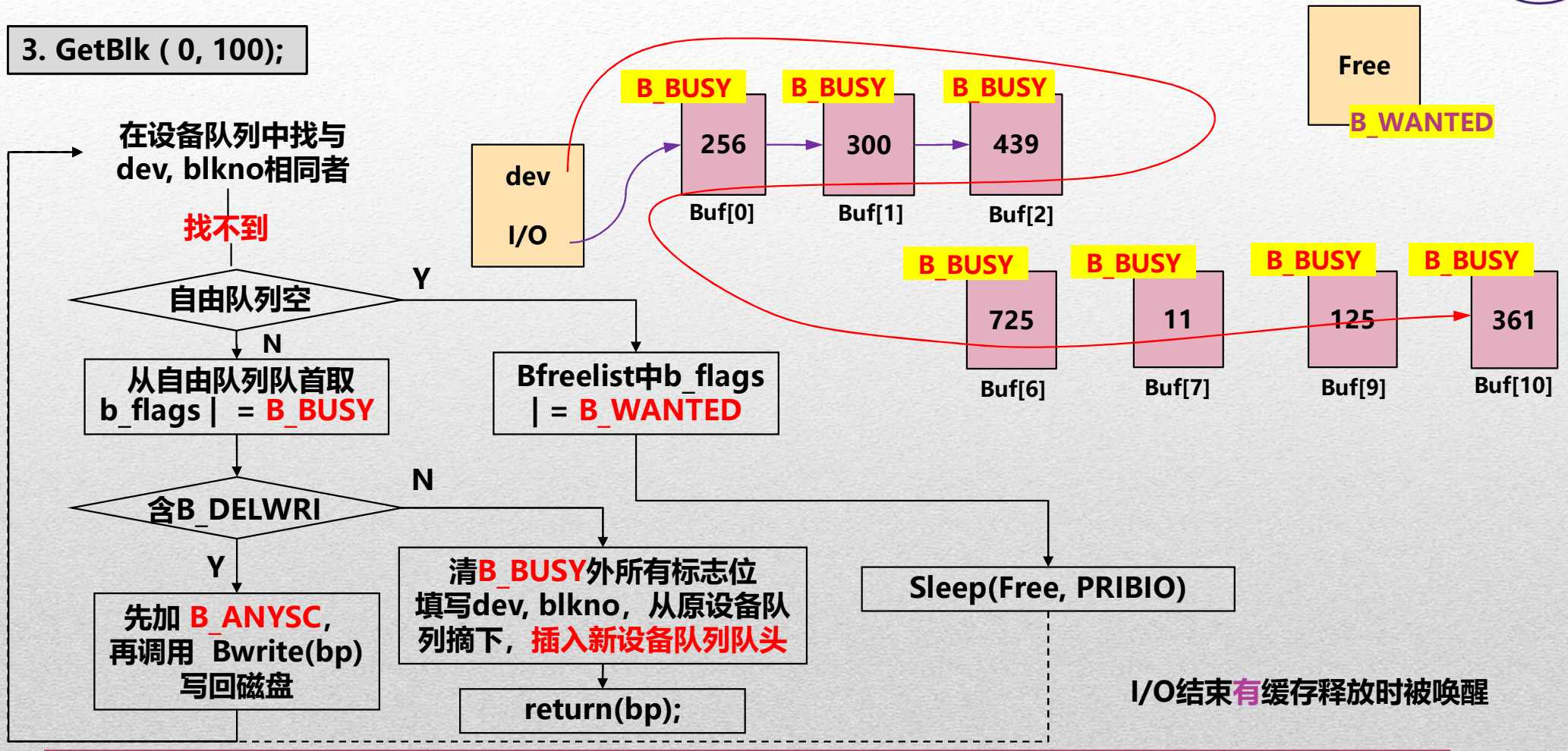
BufferManager::GetBlk(dev, blkno)过程

3. GetBlk (0, 100);

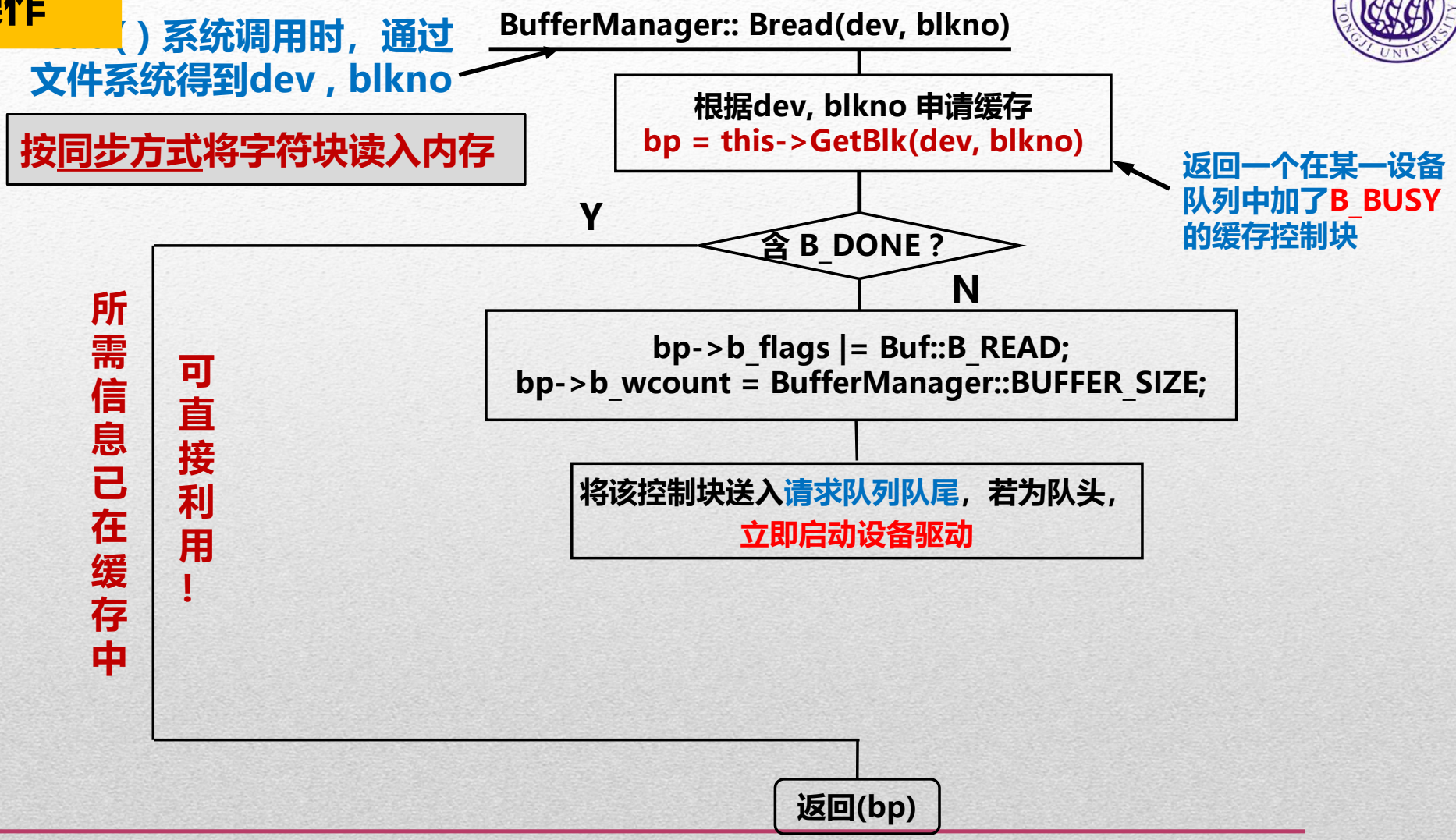




BufferManager::GetBlk(dev, blkno)过程



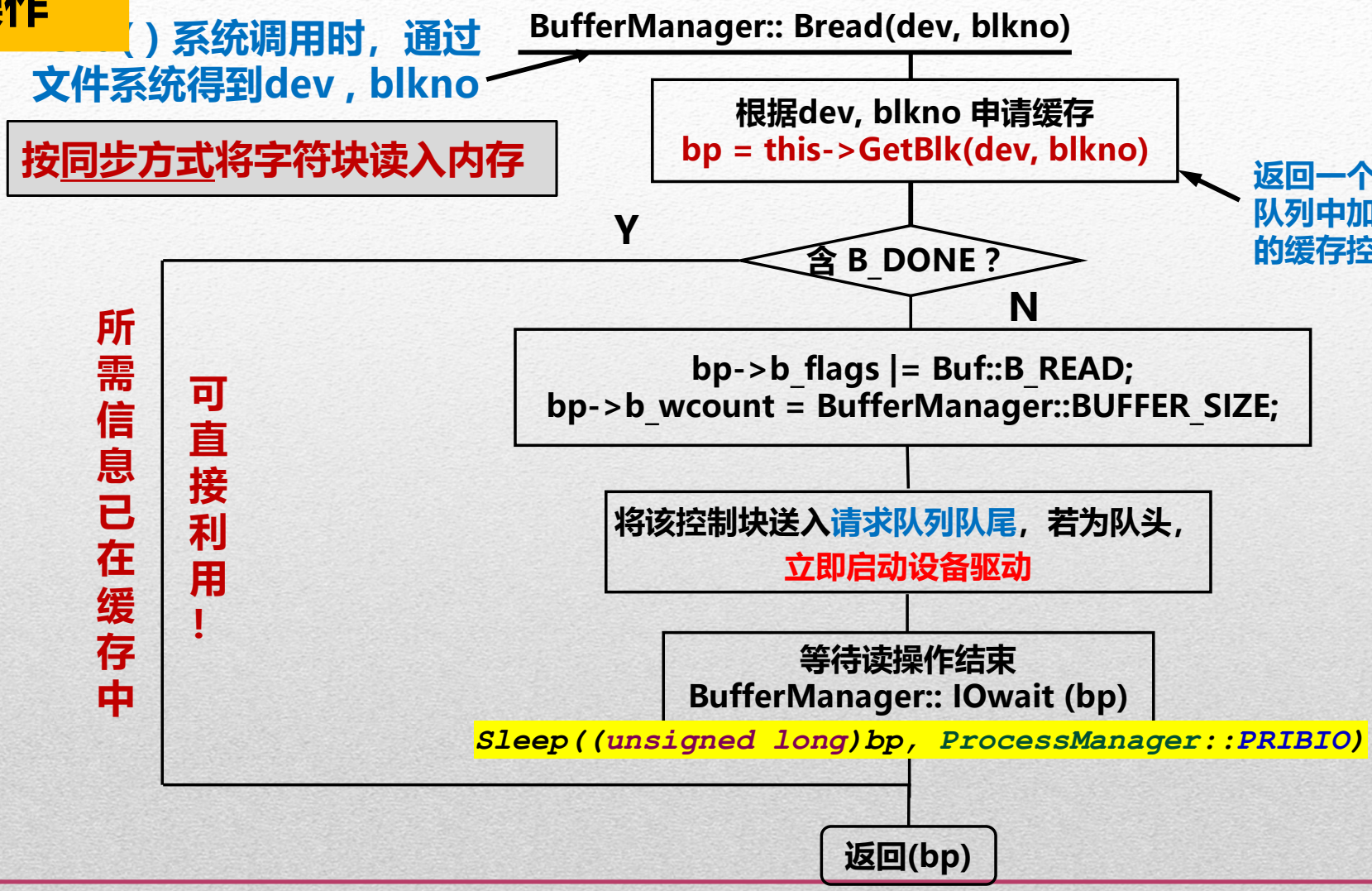
读操作


( ) 系统调用时，通过
文件系统得到dev，blkno
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
按同步方式将字符块读入内存
Y
含 B_DONE ?
N
所需信息已在缓存中
可直接利用！
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)

读操作

( ) 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

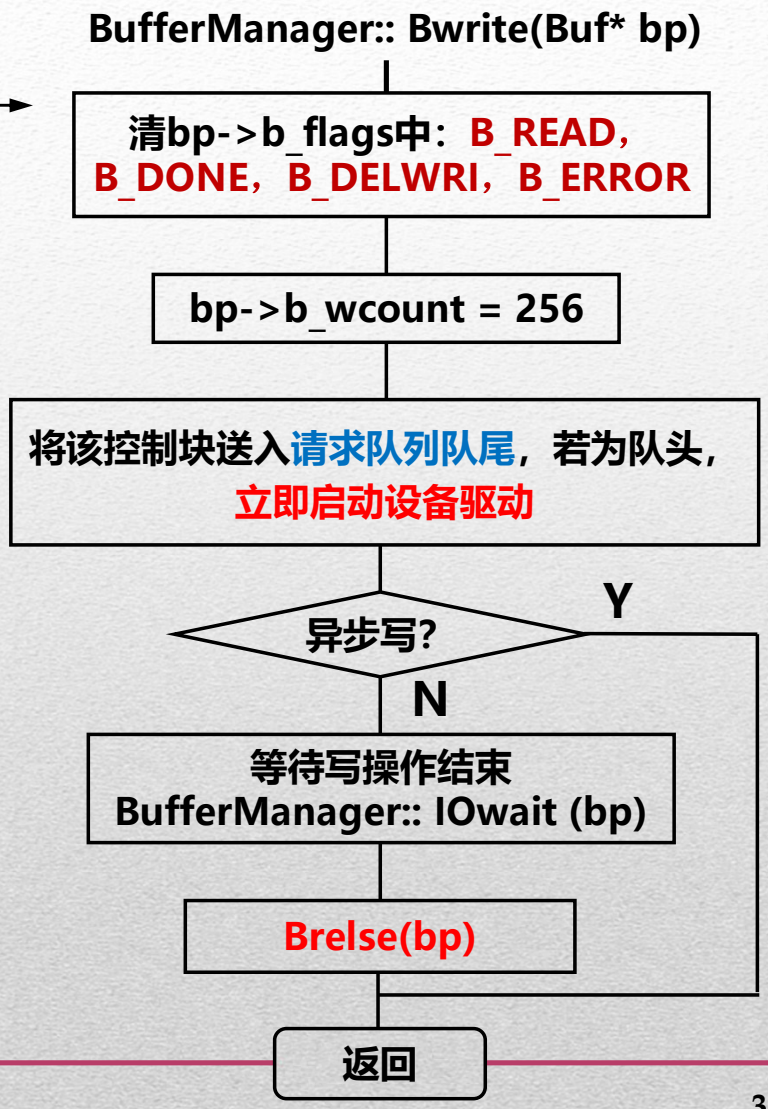


写操作

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:

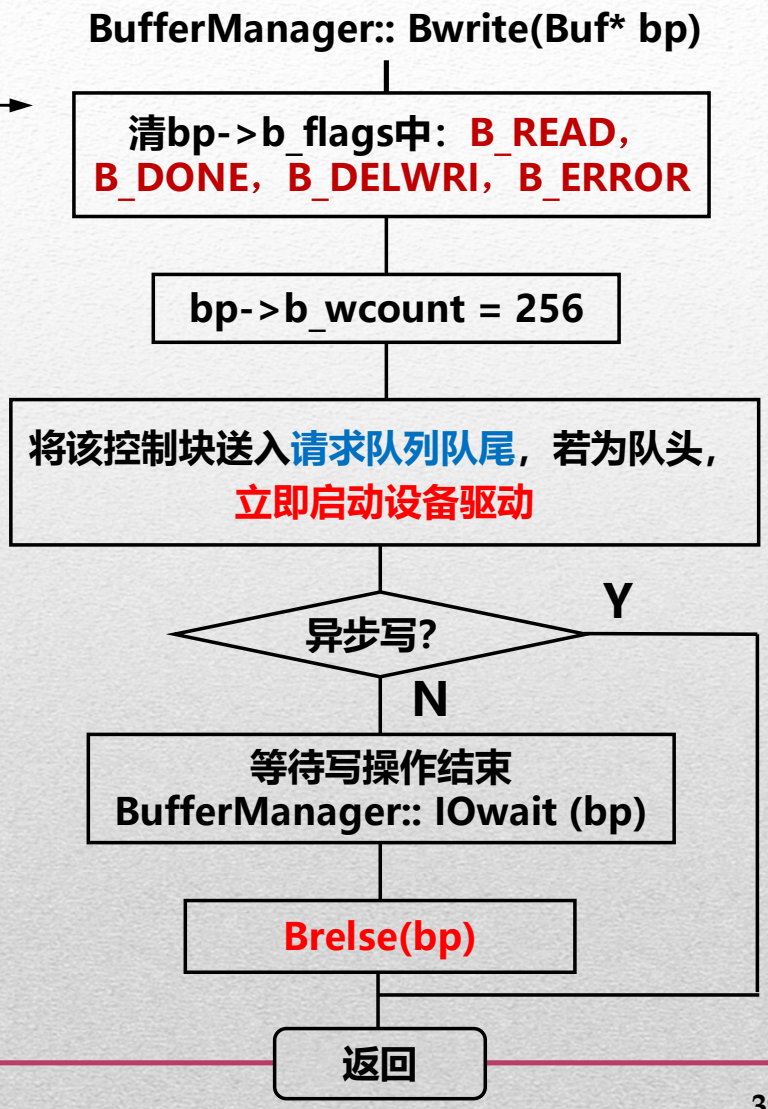
BufferManager :: Bwrite(bp)

写文件不足512字节时:

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)



I/O完成，中断响应

① 清忙

② 有错：

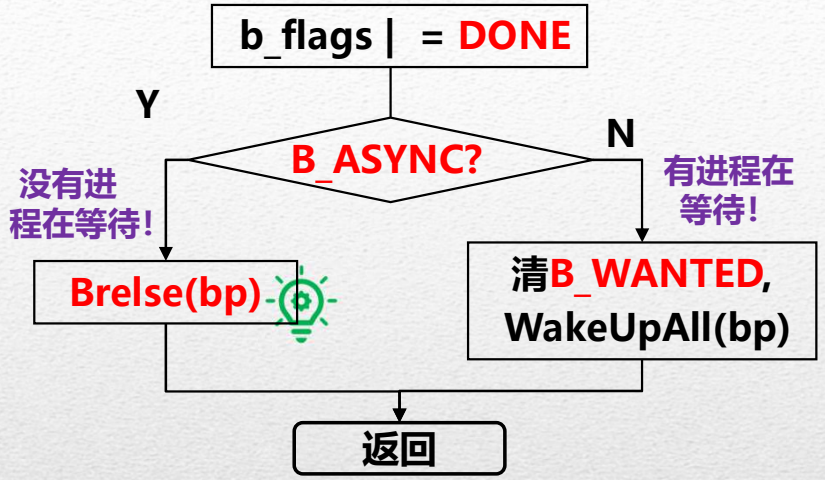
重来一次

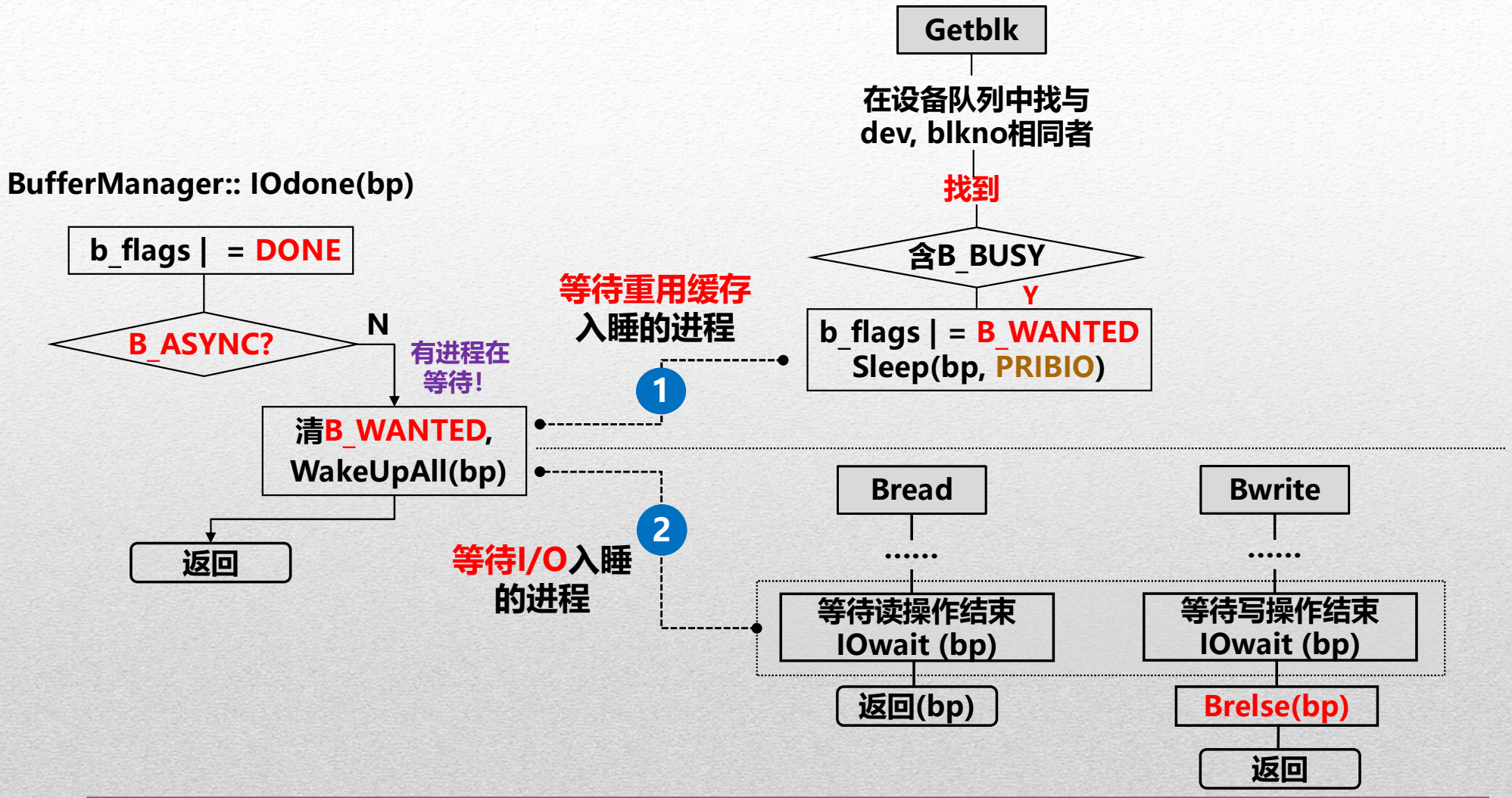
b_flag |= B_ERROR

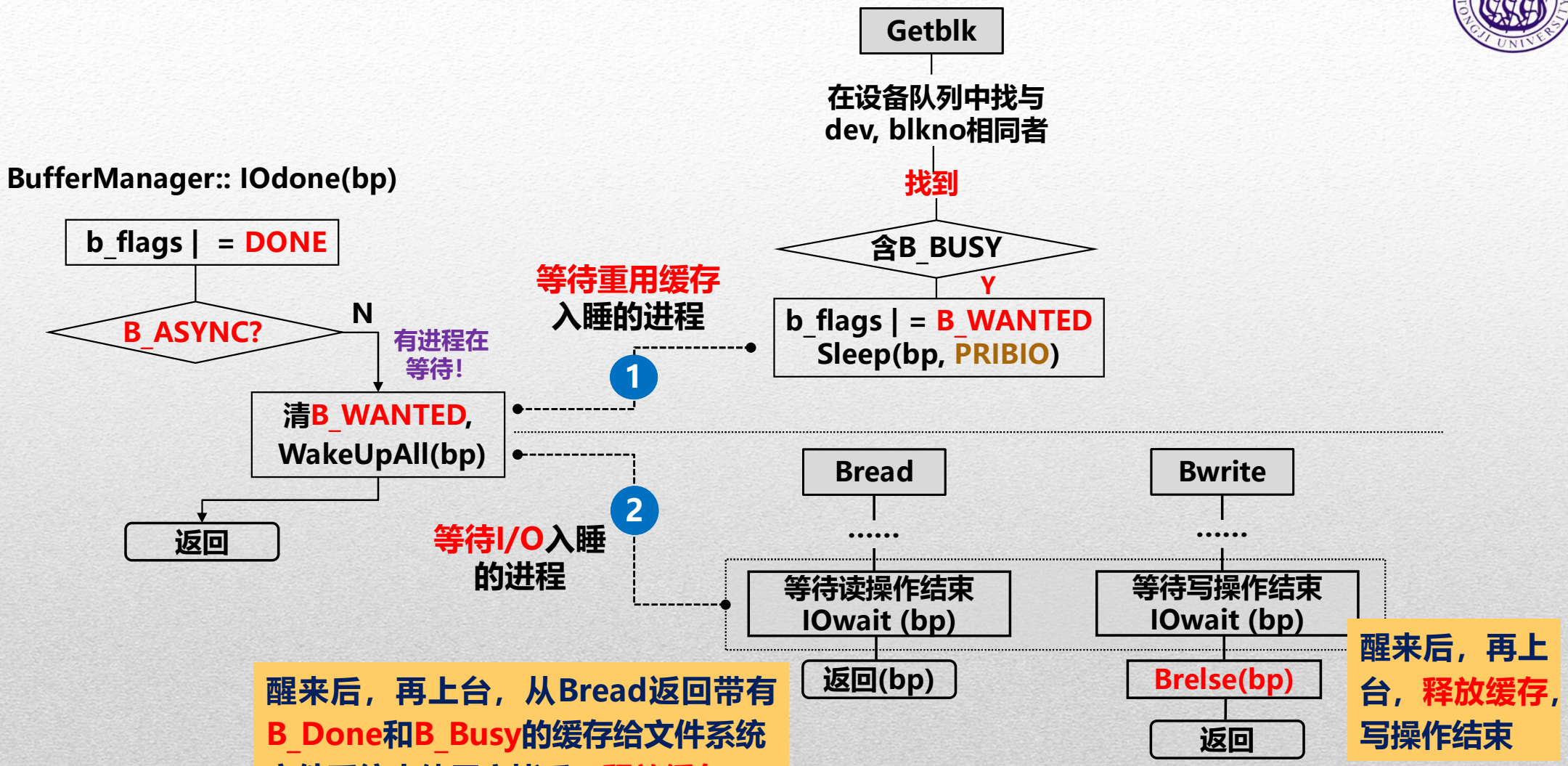
无错： 从I/O请求队列摘下第一个buf

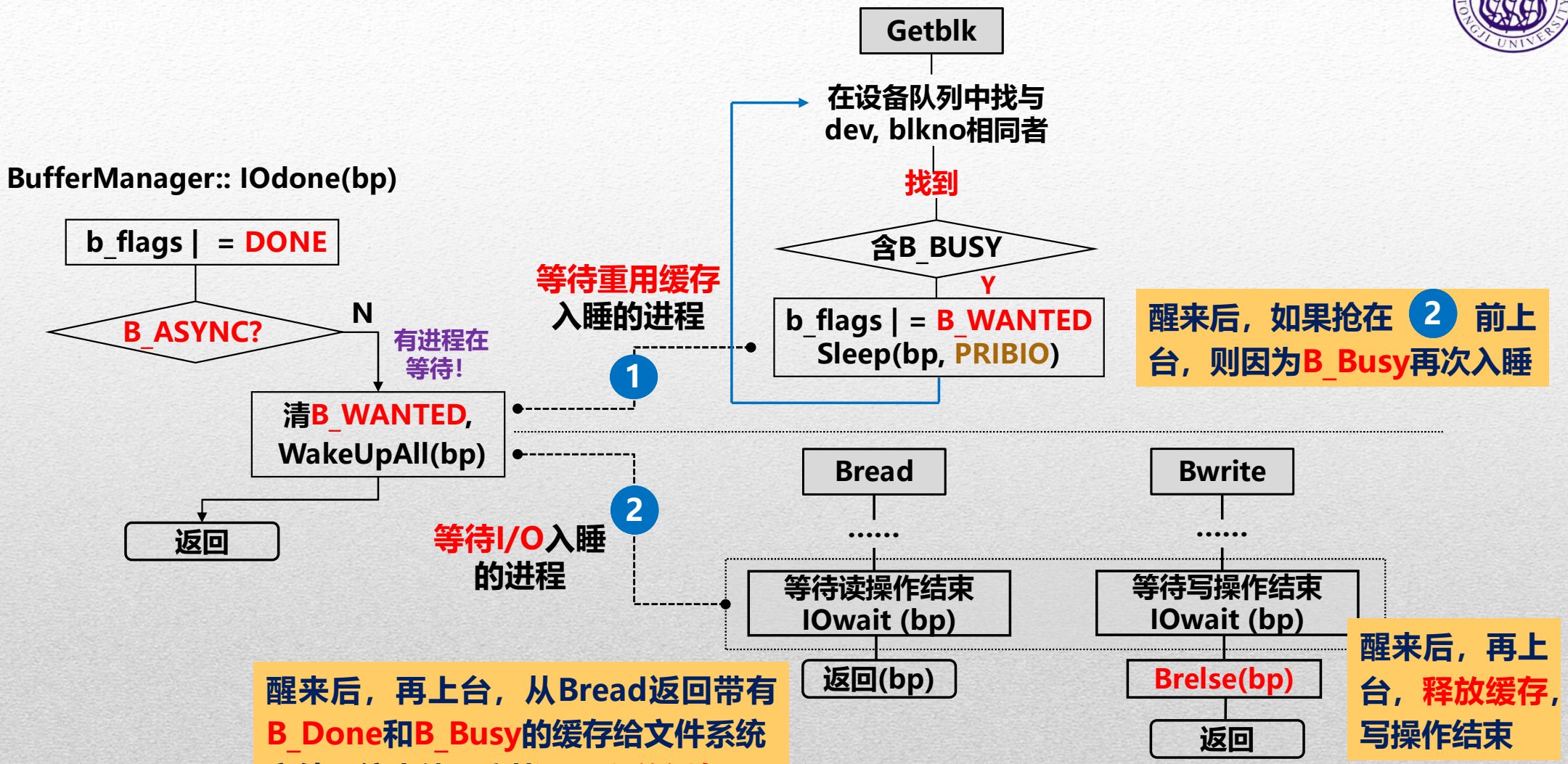
调用 BufferManager:: IOdone(bp)

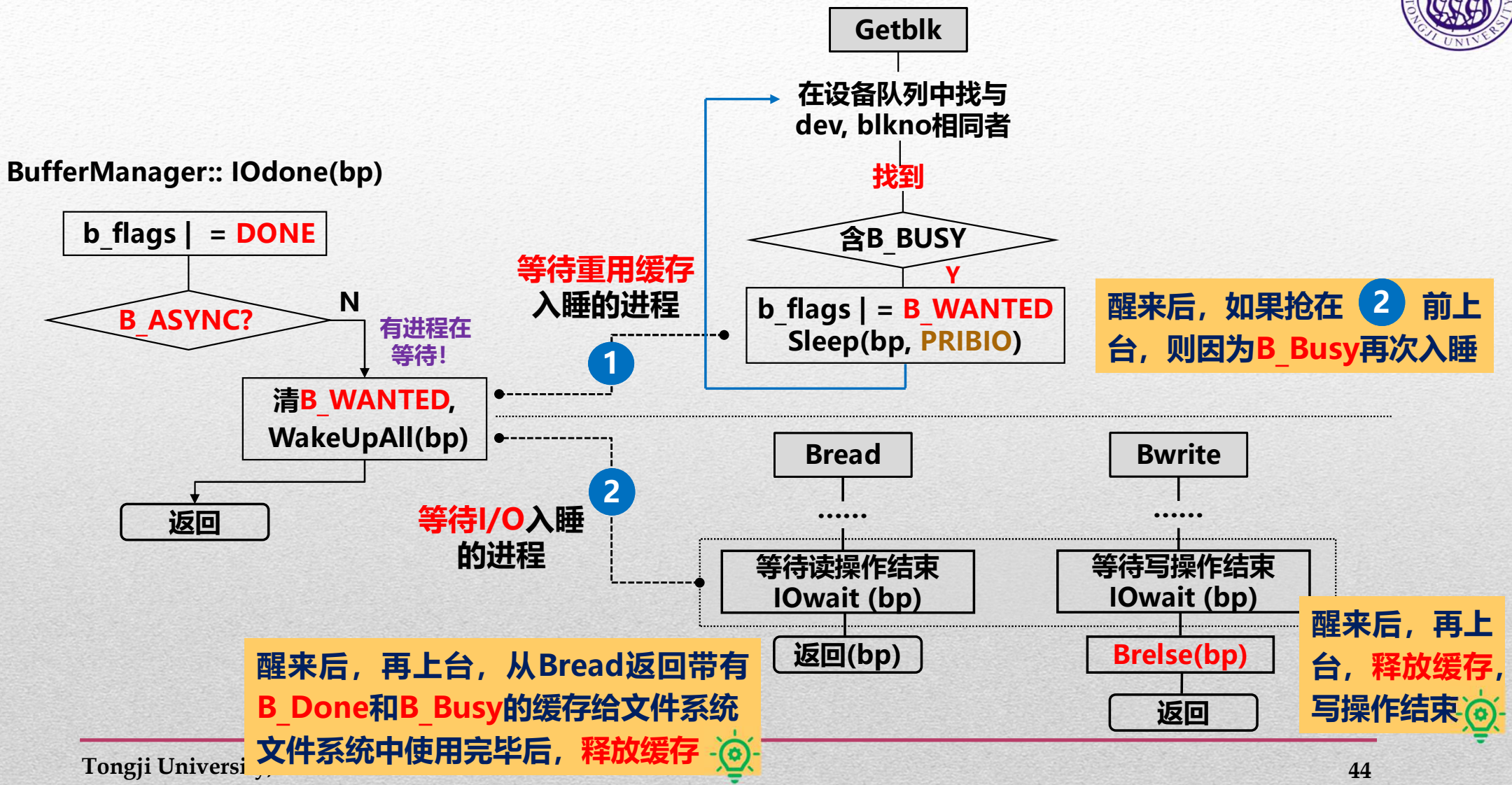
BufferManager:: IOdone(bp)



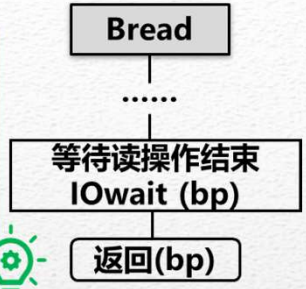

Getblk
在设备队列中找与
dev, blkno相同者
找到
含B_BUSY
Y
b_flags | = B_WANTED
Sleep(bp, PRIBIO)
等待重用缓存
入睡的进程
1
BufferManager:: IOdone(bp)
b_flags | = DONE
B_ASYNC?
N
有进程在
等待!
清B_WANTED,
WakeUpAll(bp)
返回
2
等待I/O入睡
的进程
Bread
.....
等待读操作结束
IOwait (bp)
返回(bp)
Bwrite
.....
等待写操作结束
IOwait (bp)
Brelse(bp)
返回


Getblk
在设备队列中找与
dev, blkno相同者
找到
含B_BUSY
Y
b_flags | = B_WANTED
Sleep(bp, PRIBIO)
等待重用缓存
入睡的进程
1
BufferManager:: IOdone(bp)
b_flags | = DONE
B_ASYNC?
N
有进程在
等待!
清B_WANTED,
WakeUpAll(bp)
返回
2
等待I/O入睡
的进程
Bread
.....
等待读操作结束
IOwait (bp)
返回(bp)
Bwrite
.....
等待写操作结束
IOwait (bp)
Brelse(bp)
返回
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

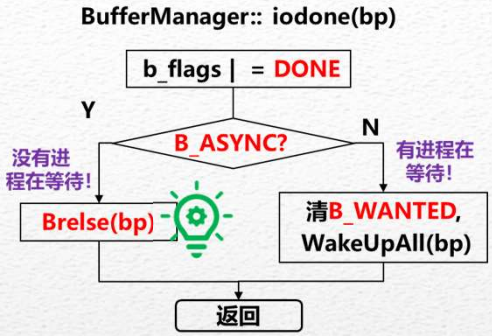

Getblk
在设备队列中找与
dev, blkno相同者
找到
含B_BUSY
Y
b_flags | = B_WANTED
Sleep(bp, PRIBIO)
等待重用缓存
入睡的进程
1
醒来后，如果抢在 2 前上台，则因为B_Busy再次入睡
BufferManager:: IOdone(bp)
b_flags | = DONE
B_ASYNC?
N
有进程在等待!
清B_WANTED,
WakeUpAll(bp)
返回
2
等待I/O入睡
的进程
Bread
.....
等待读操作结束
IOwait (bp)
返回(bp)
Bwrite
.....
等待写操作结束
IOwait (bp)
Brelse(bp)
返回
醒来后，再上台，释放缓存，写操作结束
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
找到
含B_BUSY
Y
b_flags | = B_WANTED
Sleep(bp, PRIBIO)
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
BufferManager:: IOdone(bp)
b_flags | = DONE
B_ASYNC?
N
有进程在等待!
清B_WANTED,
WakeUpAll(bp)
返回
2
等待I/O入睡
的进程
Bread
.....
等待读操作结束
IOwait (bp)
返回(bp)
Bwrite
.....
等待写操作结束
IOwait (bp)
Brelse(bp)
返回
醒来后，再上台，释放缓存，写操作结束
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，从Bread返回带有B_Done和B_Busy的缓存给文件系统，使用完毕后，释放缓存

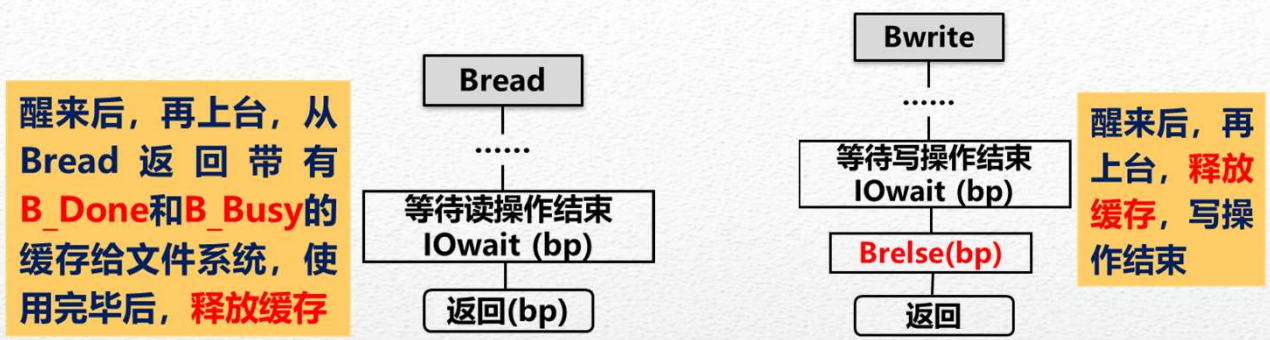


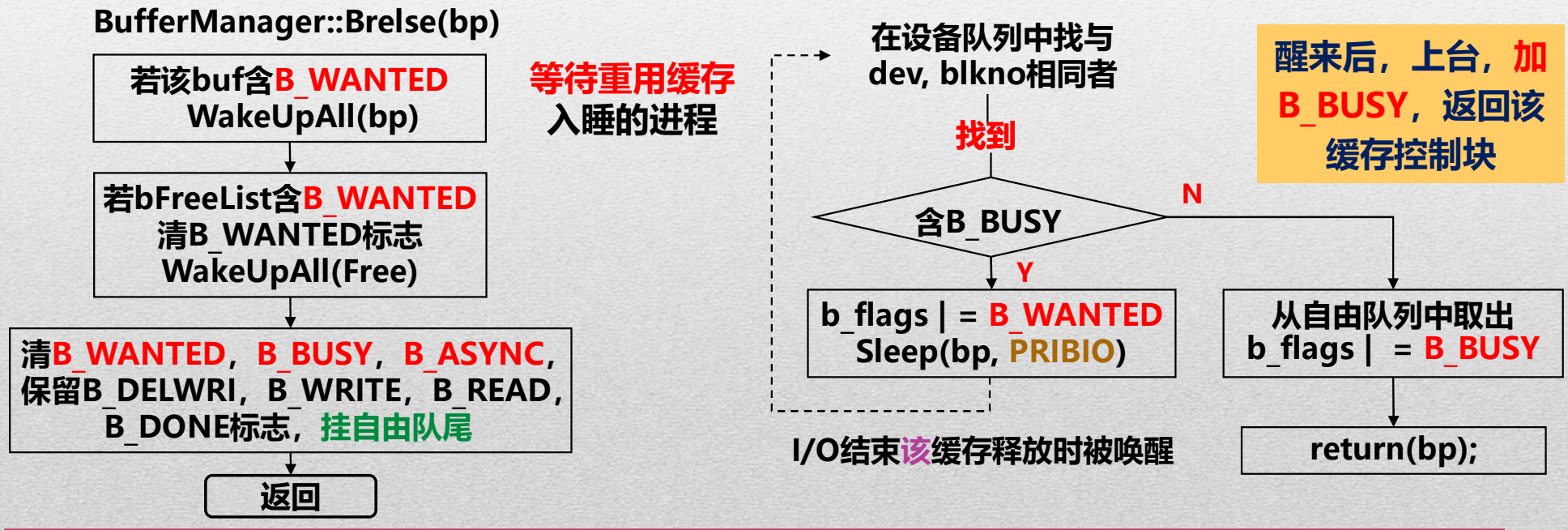
醒来后，再上台，释放缓存，写操作结束



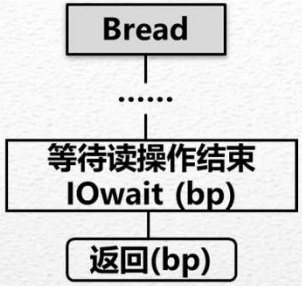
BufferManager::Brelse(bp)

等待重用缓存入睡的进程

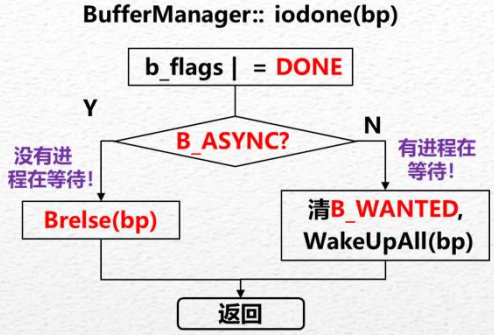

醒来后，再上台，从Bread返回带有B_Done和B_Busy的缓存给文件系统，使用完毕后，释放缓存
Bread
.....
等待读操作结束
IOwait (bp)
返回(bp)
Bwrite
.....
等待写操作结束
IOwait (bp)
Brelse(bp)
返回
醒来后，再上台，释放缓存，写操作结束
BufferManager:: iodone(bp)
b_flags | = DONE
B_ASYNC?
Y
N
没有进程在等待!
有进程在等待!
Brelse(bp)
清B_WANTED, WakeUpAll(bp)
返回


BufferManager::Brelse(bp)
若该buf含B_WANTED
WakeUpAll(bp)
若bFreeList含B_WANTED
清B_WANTED标志
WakeUpAll(Free)
清B_WANTED，B_BUSY，B_ASYNC，保留B_DELWRI，B_WRITE，B_READ，B_DONE标志，挂自由队尾
返回
等待重用缓存入睡的进程
在设备队列中找与dev, blkno相同者
找到
含B_BUSY
N
Y
b_flags | = B_WANTED
Sleep(bp, PRIBIO)
I/O结束该缓存释放时被唤醒
从自由队列中取出
b_flags | = B_BUSY
return(bp);
醒来后，上台，加B_BUSY，返回该缓存控制块

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，使用完毕后，释放缓存

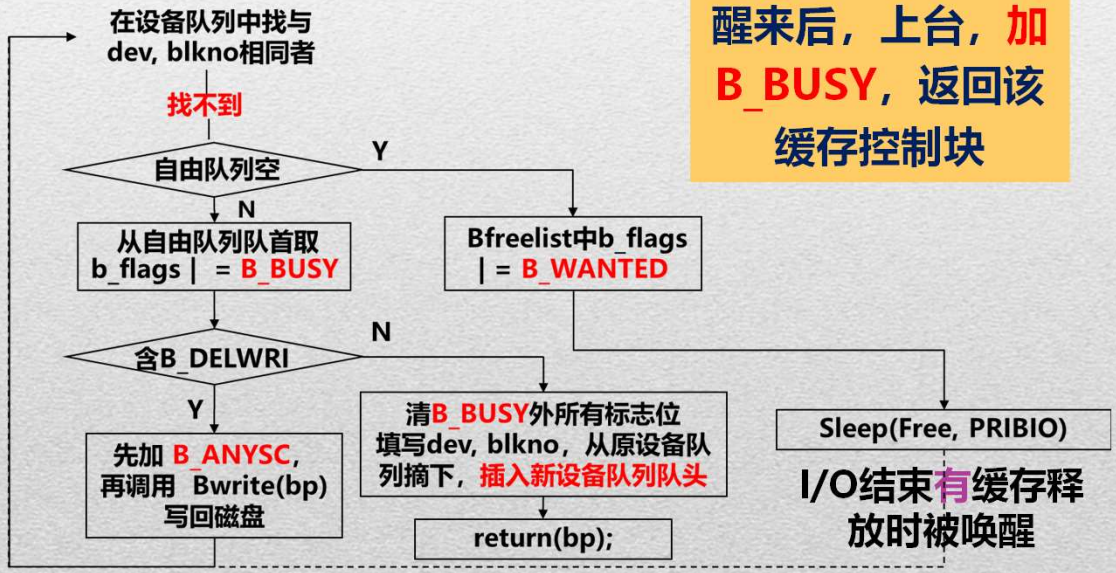


醒来后，再上台，释放缓存，写操作结束

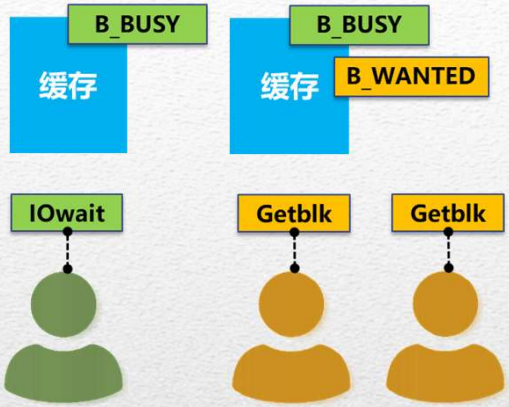


BufferManager::Brelse(bp)

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



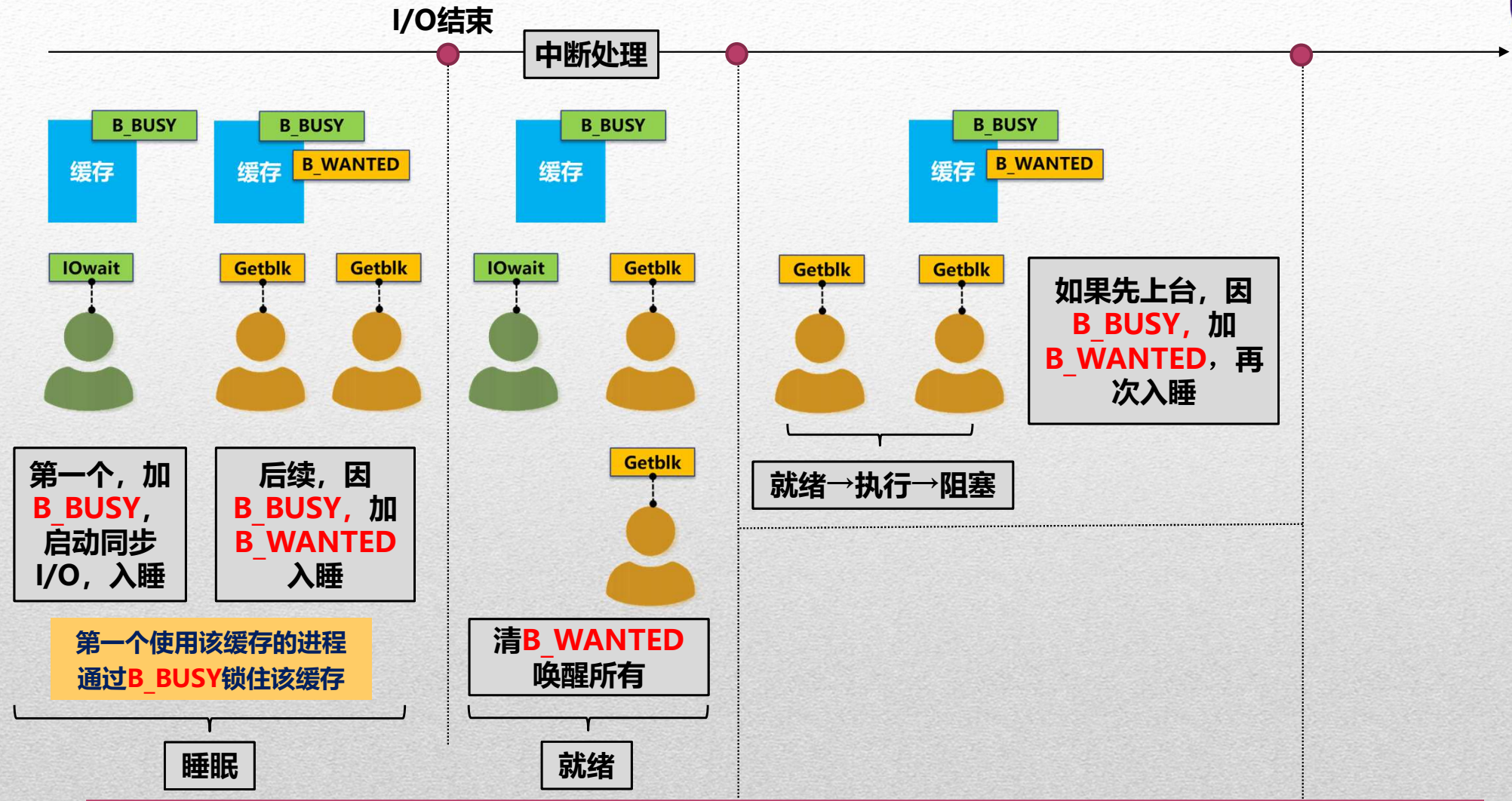
第一个，加B_BUSY，启动同步I/O，入睡

后续，因B_BUSY，加B_WANTED入睡

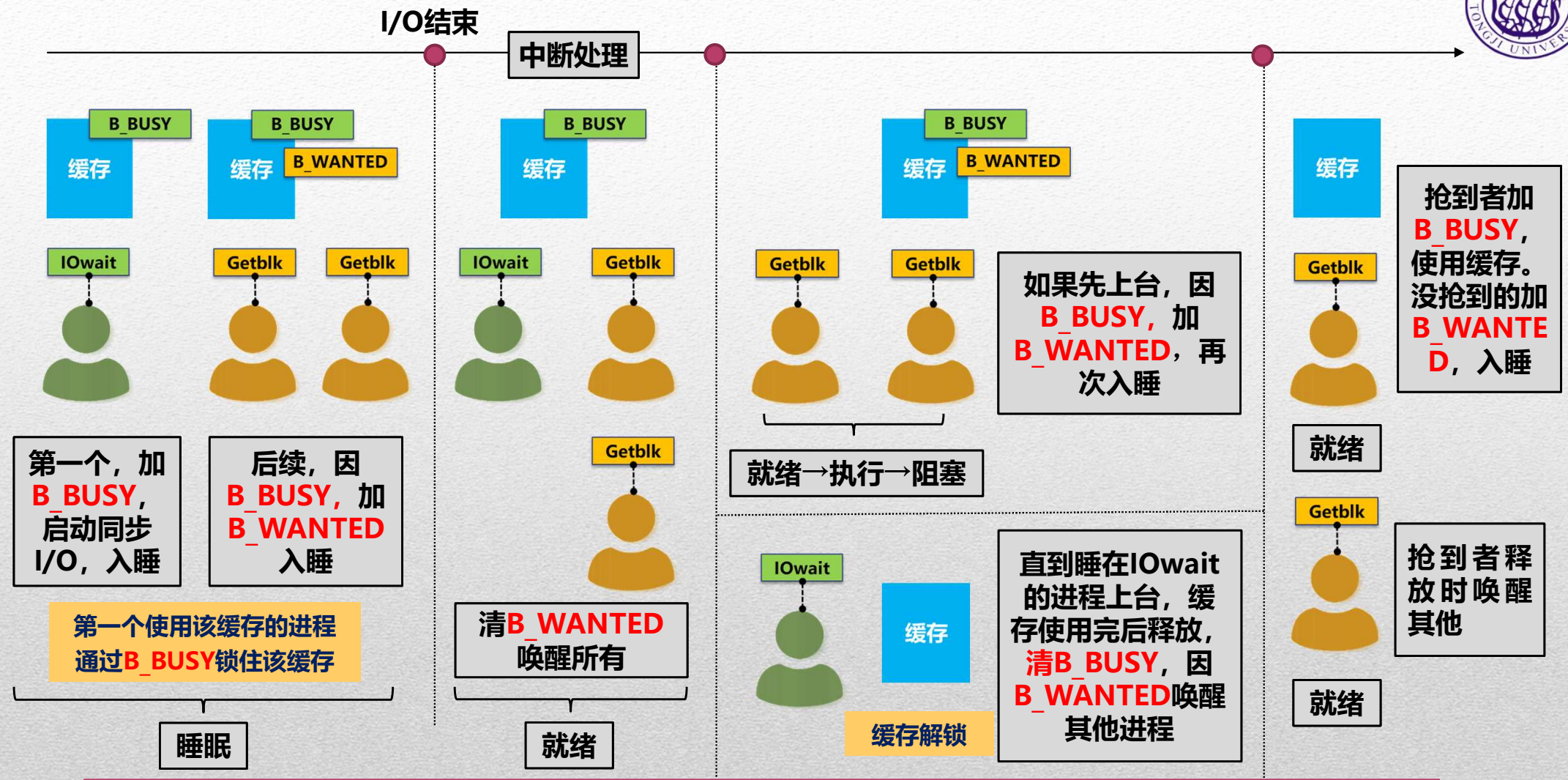
第一个使用该缓存的进程通过B_BUSY锁住该缓存

睡眠


I/O结束
中断处理
B_BUSY
缓存
B_BUSY
缓存
B_WANTED
B_BUSY
缓存
IOwait
Getblk
Getblk
IOwait
Getblk
Getblk
第一个，加B_BUSY，启动同步I/O，入睡
后续，因B_BUSY，加B_WANTED入睡
第一个使用该缓存的进程通过B_BUSY锁住该缓存
清B_WANTED唤醒所有
睡眠
就绪


I/O结束
中断处理
B_BUSY
缓存
B_BUSY
缓存
B_WANTED
B_BUSY
缓存
B_BUSY
缓存
B_WANTED
IOwait
Getblk
Getblk
IOwait
Getblk
Getblk
Getblk
Getblk
如果先上台，因B_BUSY，加B_WANTED，再次入睡
第一个，加B_BUSY，启动同步I/O，入睡
后续，因B_BUSY，加B_WANTED入睡
就绪→执行→阻塞
Getblk
第一个使用该缓存的进程通过B_BUSY锁住该缓存
清B_WANTED唤醒所有
睡眠
就绪


I/O结束
中断处理
B_BUSY
缓存
B_BUSY
缓存
B_WANTED
B_BUSY
缓存
B_BUSY
缓存
B_WANTED
IOwait
Getblk
Getblk
IOwait
Getblk
Getblk
Getblk
Getblk
如果先上台，因B_BUSY，加B_WANTED，再次入睡
就绪→执行→阻塞
第一个，加B_BUSY，启动同步I/O，入睡
后续，因B_BUSY，加B_WANTED入睡
第一个使用该缓存的进程通过B_BUSY锁住该缓存
清B_WANTED唤醒所有
IOwait
缓存
直到睡在IOwait的进程上台，缓存使用完后释放，清B_BUSY，因B_WANTED唤醒其他进程
缓存解锁
睡眠
就绪


I/O结束
中断处理
B_BUSY
缓存
B_BUSY
缓存
B_WANTED
B_BUSY
缓存
B_BUSY
缓存
B_WANTED
缓存
IOwait
Getblk
Getblk
IOwait
Getblk
Getblk
Getblk
Getblk
Getblk
Getblk
Getblk
第一个，加B_BUSY，启动同步I/O，入睡
后续，因B_BUSY，加B_WANTED入睡
第一个使用该缓存的进程通过B_BUSY锁住该缓存
睡眠
清B_WANTED唤醒所有
就绪
就绪→执行→阻塞
如果先上台，因B_BUSY，加B_WANTED，再次入睡
IOwait
缓存
缓存解锁
直到睡在IOwait的进程上台，缓存使用完后释放，清B_BUSY，因B_WANTED唤醒其他进程
就绪
抢到者加B_BUSY，使用缓存。没抢到的加B_WANTED，入睡
就绪
抢到者释放时唤醒其他

I/O完成，中断响应

① 清忙

② 有错：< 10 —Y→ 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 启动请求队列中的下一个请求

BufferManager:: iodone(bp)

本节小结：

1　熟悉UNIX V6++的缓存管理方法

2　掌握UNIX V6++的缓存读写技术

E09：设备管理