②下面我们采用一种改良后的双向链式结构：

链式结构中每一个块的数据结构如下图所示：

**Block-Data；**

**Next；**

**Prefer；**

**Cross-Next；**

**Block-Data：存储信息，在位示图中表示了分区的序号和该分区是否被使用。**

**Next：顺序指向下一个块。**

**Prefer：顺序指向上一个块。**

**Cross-Next：当顺序结构被破坏时，该指针负责保持链的连通。**

**初始化链表：**

**Idx：4**

**Idx：3**

**Idx：2**

**Idx：1**

**初始状态下块与块之间通过，Next和Prefer指针相连。**

**Idx：n**

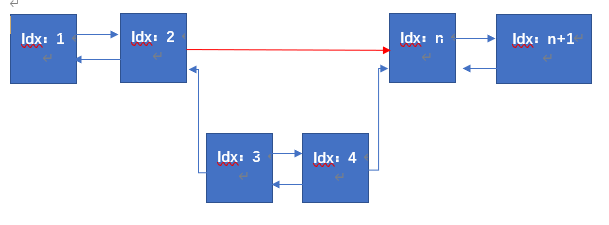
**移除3-4个块：**

**在移除第k到k+n个块时，进行一下操作：**

**①将k-1块的Next指针置None，Cross-Next指针指向k+n+1**

**②将第k+n+1个块的prefer指针置None**

**如此操作之后，就可以将已经被占用的块从顺序结构中移除，在进行遍历时就可以快速跳过被占用的块。同时通过下一个块是否通过Next指针连接还是Cross-Next指针连接判断是否时连续的块。**



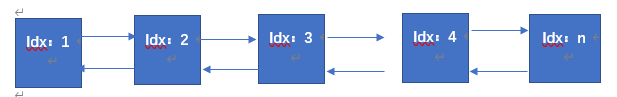
回收3-4号块：

回收3-4号块进行以下操作：

①指针移动到第2块，置第2块的Next指针为第3块的地址，Cross-Next置为None

②指针移动到第5块，置第5块的prefer指针为第4块的地址。

这样就完成了块的回收。



我们发现这样通过这种改良的链式结构的确加快了查找空闲区的速率，但是在进行块的回收时，时间复杂度又变成了O(n)如此看来，这样的链式结构多适用于查找频率大于回收频率的系统。但是我们实际可以再引入一种字典结构存储每一个块的地址，这样在回收块时，通过直接查找字典来代替指针移动，使得时间复杂度降位O（1）。

综上：

①基于位运算的数据结构适用于计算机资源紧张的系统，对查找效率没有要求的系统。

②基于链式的数据结构适用于对查找效率要求高的系统。