数据结构与算法 (十二)

张铭 主讲

采用教材：张铭，王腾蛟，赵海燕 编写
高等教育出版社，2008. 6（“十一五”国家级规划教材)

http://www.jpk.pku.edu.cn/pkujpk/course/sjjg

第十二章 高级数据结构

- 12.1 多维数组
- 12.2 广义表
- 12.3 存储管理
  - 分配与回收
  - 可利用空间表
  - 存储的动态分配和回收
  - 失败处理策略和无用单元回收
- 12.4 Trie 树
- 12.5 改进的二叉搜索树

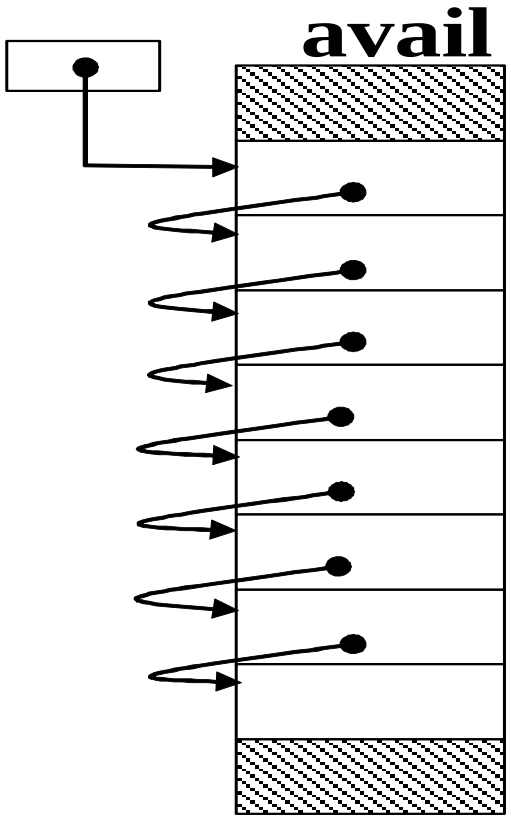
分配与回收

- 内存管理最基本的问题
  - 分配存储空间
  - 回收被“释放”的存储空间
- 碎片问题
  - 存储的压缩
- 无用单元收集
  - 无用单元：可以回收而没有回收的空间
  - 内存泄漏（memory leak）
    - 程序员忘记 delete 已经不再使用的指针

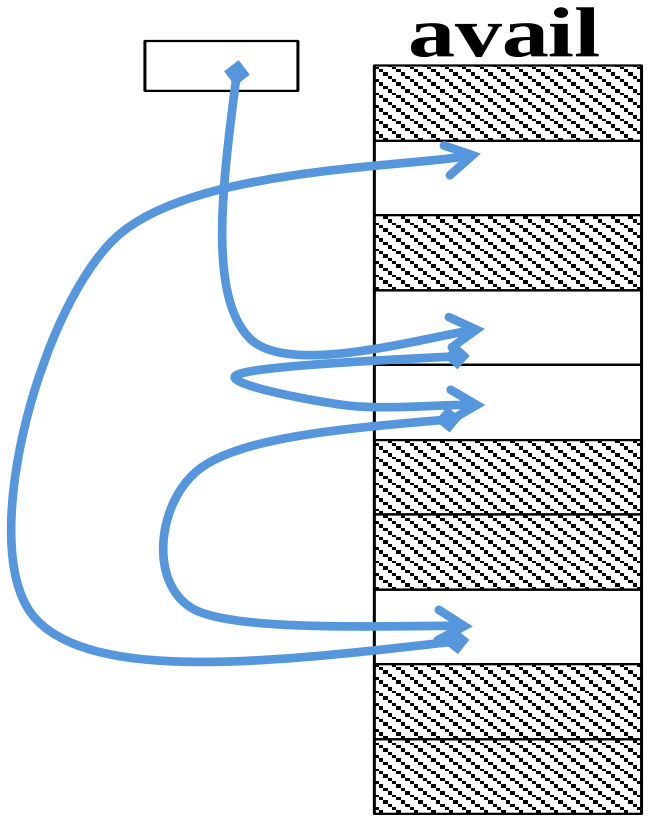
可利用空间表

- 把存储器看成一组定长块组成的数组
  - 一些块是已分配的
  - 链接空闲块，形成可利用空间表 (freelist)
- 存储分配和回收
  - new p 从可利用空间分配
  - delete p 把 p 指向的数据块返回可利用空间表

12.3 存储管理



（1）初始状态的可利用空间表



（2）系统运行一段时间后的可利用空间表

结点等长的可利用空间表

可利用空间表的函数重载

```
template <class Elem> class LinkNode{
private:
    static LinkNode *avail;                  // 可利用空间表头指针
public:
    Elem value;                              // 结点值
    LinkNode * next;                         // 指向下一结点的指针
    LinkNode (const Elem & val, LinkNode * p) ;
    LinkNode (LinkNode * p = NULL) ;         // 构造函数
    void * operator new (size_t) ;           // 重载new运算符
    void operator delete (void * p) ;        // 重载delete运算符
};
```

12.3 存储管理

```
// 重载new运算符实现
template <class Elem>
void * LinkNode<Elem>::operator new (size_t) {
    if (avail == NULL)                  // 可利用空间表为空
        return ::new LinkNode;          // 利用系统的new分配空间
    LinkNode<Elem> * temp = avail;
                                        // 从可利用空间表中分配
    avail = avail->next;
    return temp;
}
```

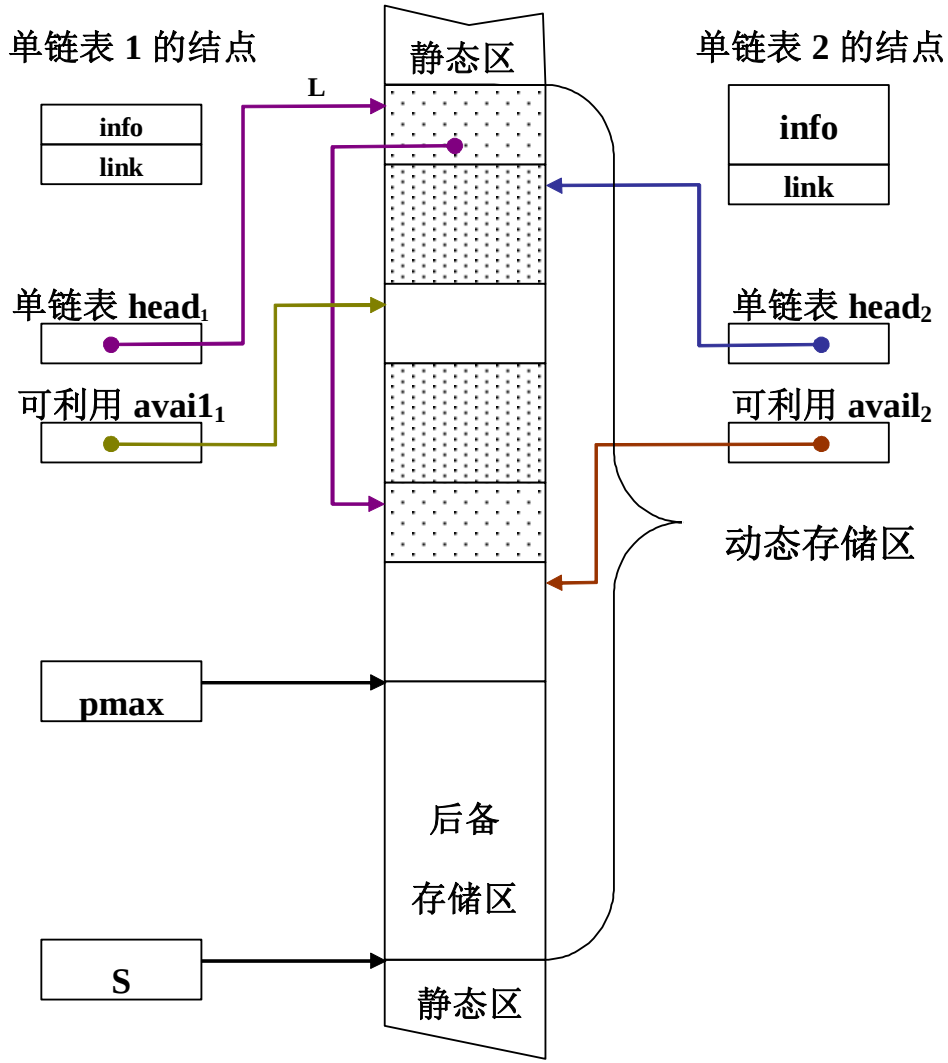
12.3 存储管理

```
// 重载delete运算符实现
template <class Elem>
void LinkNode<Elem>::operator delete (void * p) {
    ( (LinkNode<Elem> *)  p) ->next = avail;
    avail =  (LinkNode<Elem> *) p;
}
```

可利用空间表：单链表栈

- new 即栈的删除操作
- delete 即栈的插入操作
- 直接引用系统的 new 和 delete 操作符，需要强制用 **“::new p”** 和 **“::delete p”**
  - 例如，程序运行完毕时，把 avail 所占用的空间都交还给系统 (真正释放空间)

12.3 存储管理



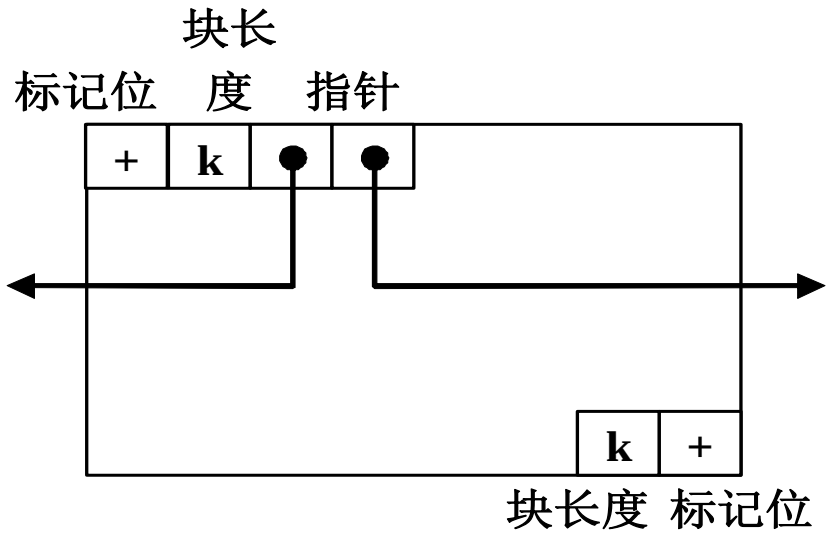
- pmax 值已经达到或超过S值，则不能再分配空间

存储的动态分配和回收

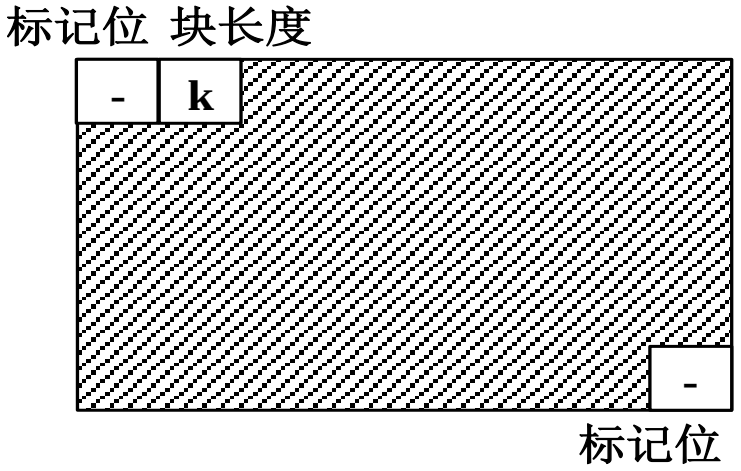
变长可利用块

- 分配
  - 找到其长度大于等于申请长度的结点
  - 从中截取合适的长度
- 回收
  - 考虑刚刚被删除的结点空间能否与邻接合并
  - 以便能满足后来的较大长度结点的分配请求

空闲块的数据结构

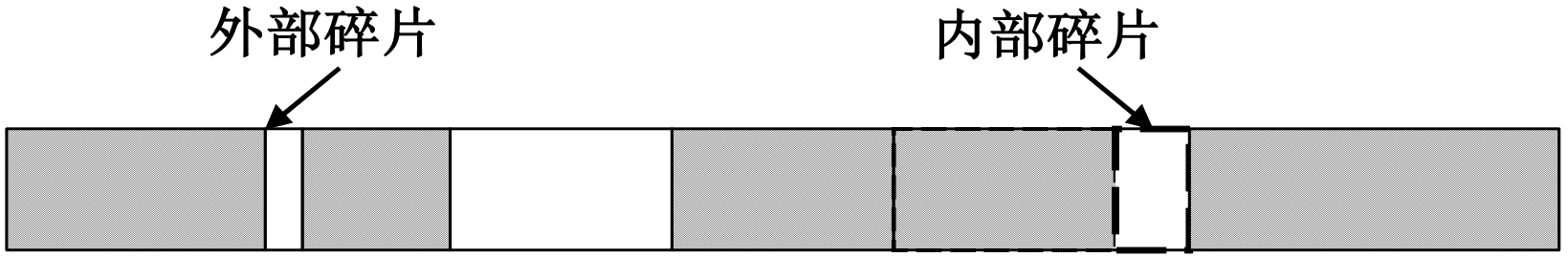


（a）空闲块的结构



（b）已分配块的结构

碎片问题



外部碎片和内部碎片

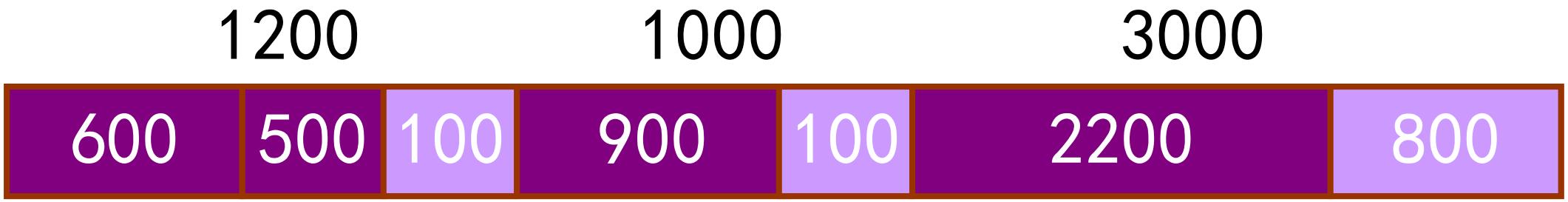
- 内部碎片：多于请求字节数的空间
- 外部碎片：小空闲块

空闲块的顺序适配 (sequential fit)

- 首先适配 (first fit)
- 最佳适配 (best fit)
- 最差适配 (worst fit)

顺序适配

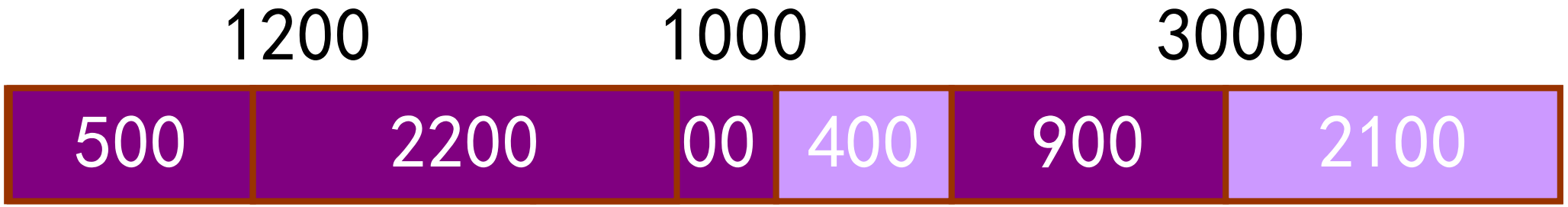
- 首先适配：



- 问题: 三个块 1200，1000，3000
  请求序列：600，500，900，2200

顺序适配

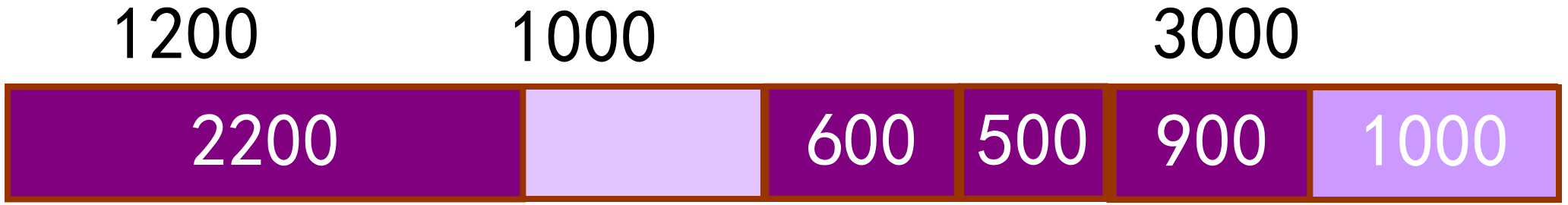
- 最佳适配



请求序列：600，500，900，2200

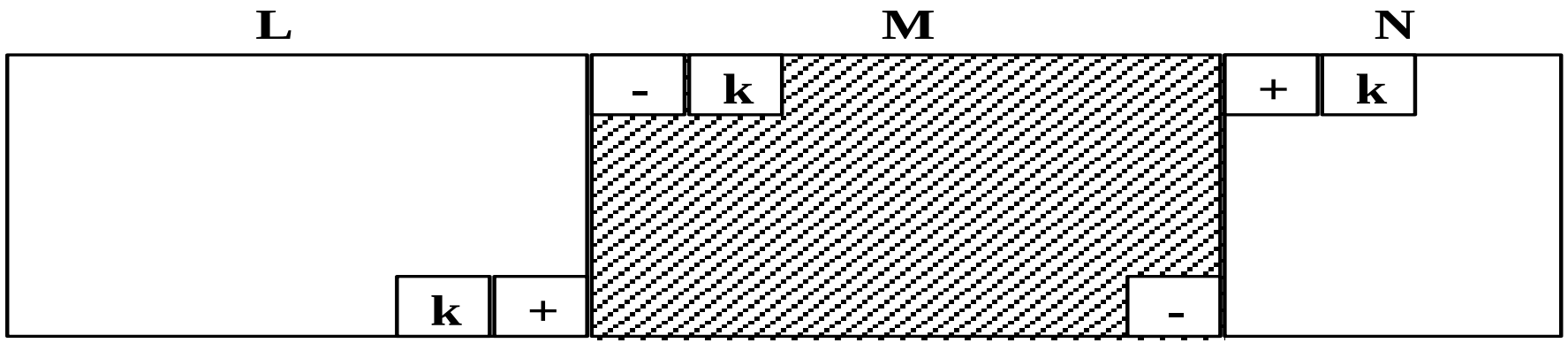
顺序适配

- 最差适配



请求序列：600，500，900，2200

回收：考虑合并相邻块



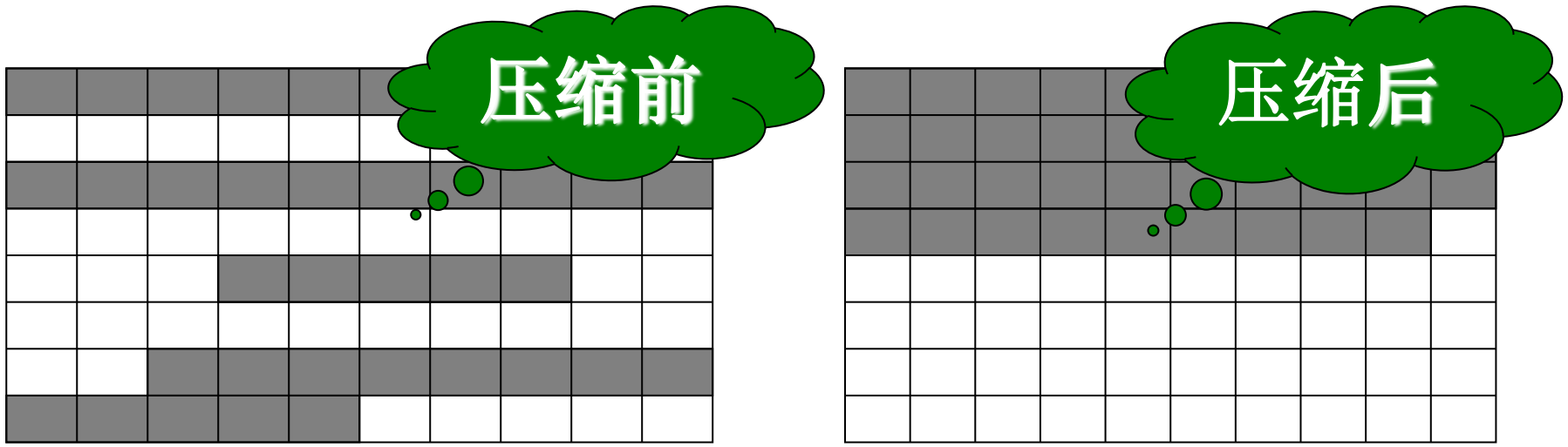
把块 M 释放回可利用空间表

失败处理策略和无用单元回收

- 如果遇到因内存不足而无法满足一个存储请求，存储管理器可以有两种行为
  - 一是什么都不做，直接返回一个系统错误信息
  - 二是使用失败处理策略 (failure policy) 来满足请求

存储压缩（compact）

- 在可利用空间不够分配或在进行无用单元的收集时进行“存储压缩”



无用单元收集

- 无用单元收集：最彻底的失败处理策略
  - 普查内存，标记把那些不属于任何链的结点
  - 将它们收集到可利用空间表中
  - 回收过程通常还可与存储压缩一起进行

思考

- 比较首先适配、最佳适配和最差适配的特点
  - 哪种适配方案更容易产生外部碎片？
  - 哪种方案总体最优？
- 怎样有效地组织空闲块，使得分配时查找到合适块的效率更高？
  - 所有空闲块都组织在一个线性表中
  - 不同规模的空闲块组织在不同的链表中
  - 以空闲块的大小为 key 组织在平衡的 BST 中

数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

张铭，王腾蛟，赵海燕
高等教育出版社，2008. 6。“十一五”国家级规划教材