Data Structures and Algorithms (12)

Instructor: Ming Zhang

Textbook Authors: Ming Zhang, Tengjiao Wang and Haiyan Zhao

Higher Education Press, 2008.6 (the "Eleventh Five-Year" national planning textbook)

https://courses.edx.org/courses/PekingX/04830050x/2T2014/

Chapter 12 Advanced Data Structure

- 12.1 Multidimensional array
- 12.2 Generalized Lists
- 12.3 Storage management
  - Allocation and Reclamation
  - Freelist
  - Dynamic Memory Allocation and Reclamation
  - Failure Policy and Collection of Useless Units
- 12.4 Trie
- 12.5 Improved BST

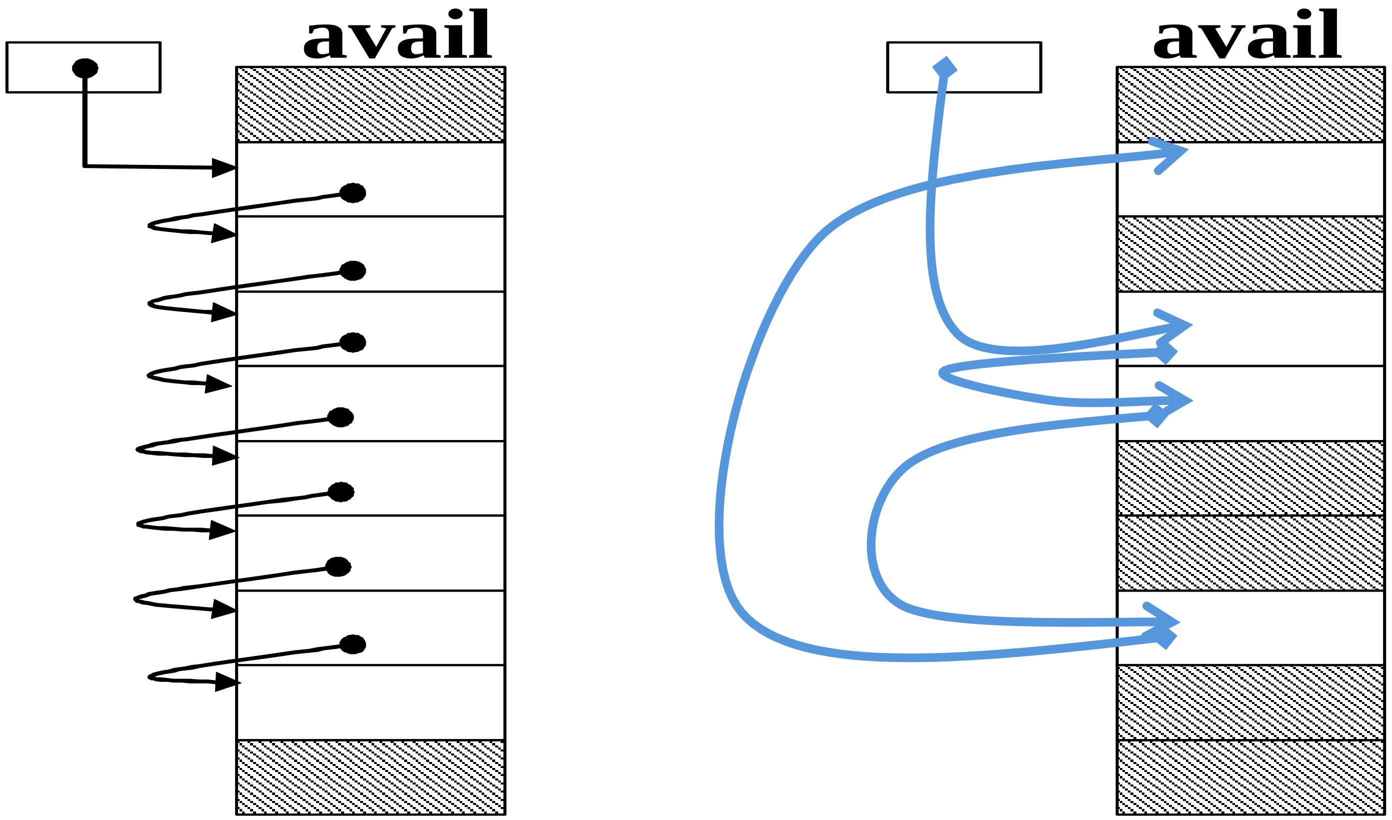
Allocation and Reclamation

- Basic problems in storage management
  - Allocate memory
  - Reclaim "freed" memory
- Fragmentation problem
  - The compression of storage
- Collection of useless units
  - Useless units: memory that can be collected but has not been collected yet
  - Memory leak
    - Programmers forget to delete pointers which will not be used

Freelist

- Consider the memory as an array of changeable number of blocks
  - Some blocks has been allocated
  - Link free blocks together, and form a freelist.
- Memory allocation and reclamation
  - new p: allocate from available space
  - delete p: return the block that p points to to the freelist.
- If there is not enough space, resort to failure policy.

12.3 Storage Management



（1） initial state of the freelist

（2） freelist after the system has run for a while

freelist with nodes of equal length

Function overloading of freelist

```
template <class Elem> class LinkNode{
private:
    static LinkNode  avail;            // head pointer
public:
    Elem value;                        // value of each node
    LinkNode   next;                   // pointer pointing to next node
    LinkNode (const Elem & val, LinkNode   p) ;
    LinkNode (LinkNode   p = NULL) ;   // construction function
    void   operator new (size_t) ;     // redefine new
    void operator delete (void   p) ;  // redefine delete
};
```

12.3 Storage Management

```
//implementation of new
template <class Elem>
void   LinkNode<Elem>::operator new (size_t) {
    if (avail == NULL)           //if the list is empty
       return ::new LinkNode;    //allocate memory using new
    LinkNode<Elem>   temp = avail;
                                 //allocate from available space
list
    avail = avail->next;
    return temp;
}
```

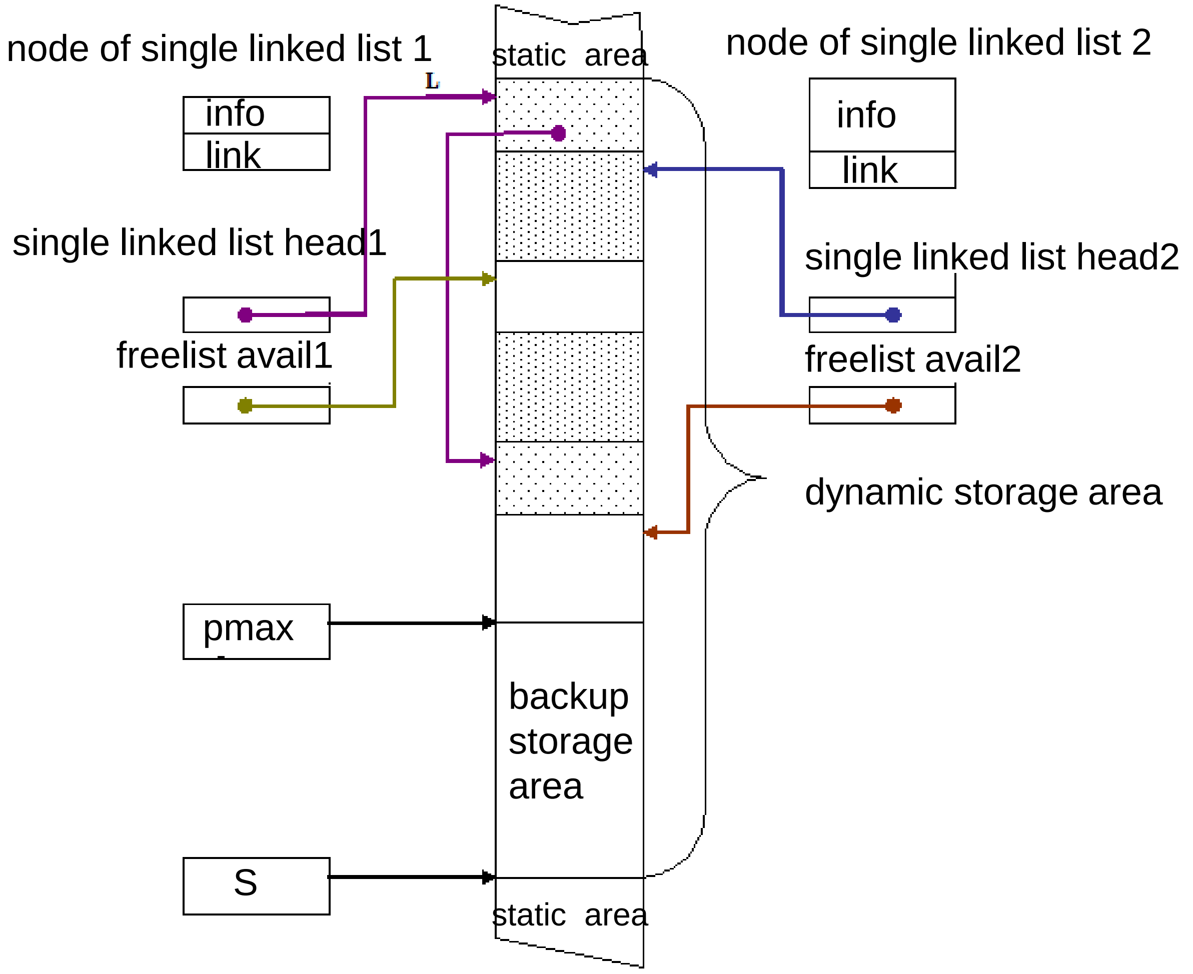
12.3 Storage Management

```
//implementation of delete
template <class Elem>
void LinkNode<Elem>::operator delete (void   p) {
    ( (LinkNode<Elem>   )  p) ->next = avail;
    avail =  (LinkNode<Elem>   ) p;
}
```

Free List: Stack in a Singly-Linked List

- new: deletion in the stack
- delete: insertion in the stack
- If the default new and delete operations are needed, use **"::new p" and "::delete p".**
  - For example, when a program is finished, return the memory occupied by avail back to the system (free the memory completely)

12.3 Storage Management



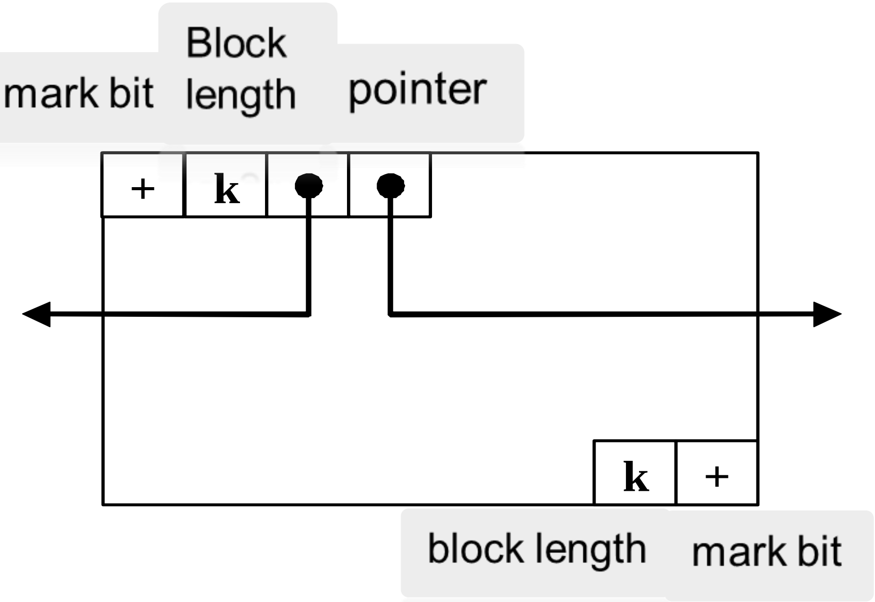
- When pmax is equal to or larger than S, no more memory can be allocated.

Dynamic Memory Allocation and Reclamation

Available blocks with variable lengths

- Allocation
  - Find a block whose length is larger than the requested length.
  - Truncate suitable length from it.
- Reclamation
  - Consider whether the space deleted can be merged with adjacent nodes,
  - So as to satisfy later request of large node.

Data Structure of Free Blocks



mark bit

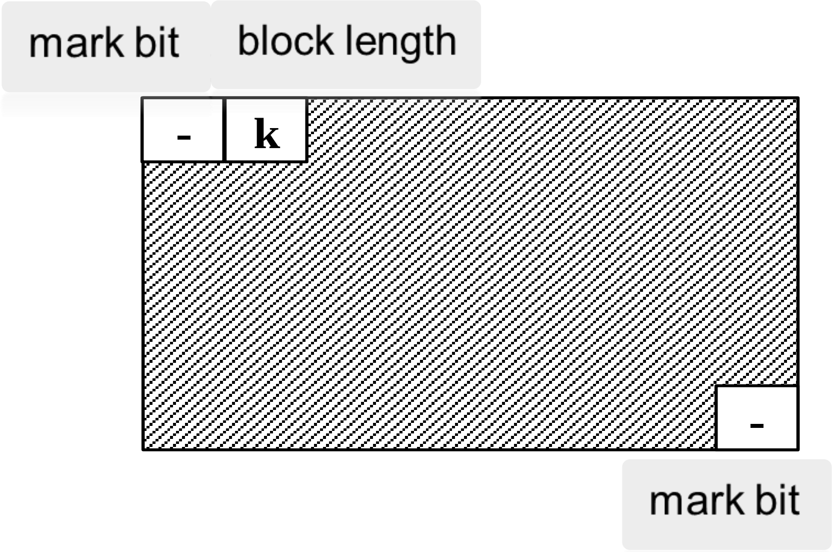
Block length

pointer

block length

mark bit

（a） structure of free block



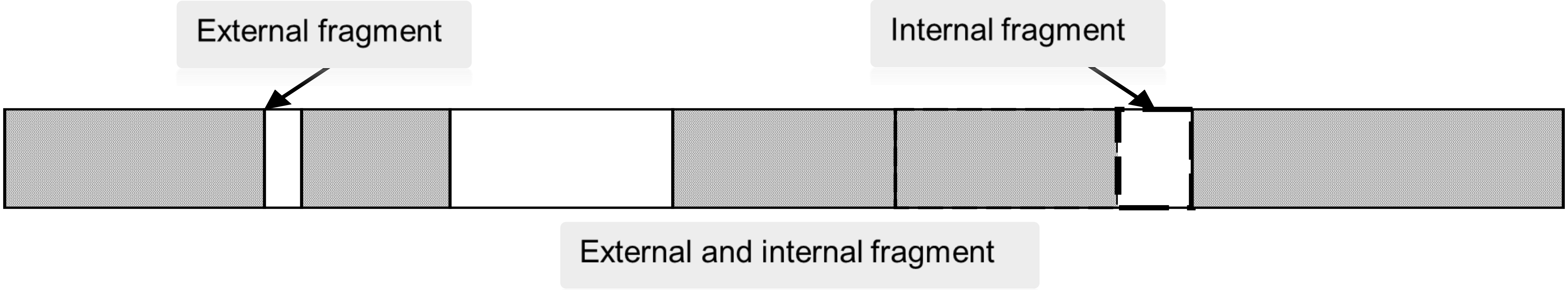
mark bit

block length

mark bit

（b） structure of allocated block

Fragmentation Problem



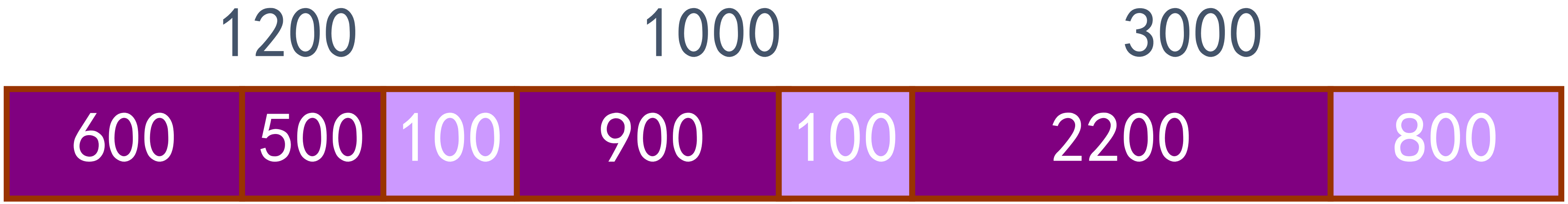
- Internal fragment: space larger than the requested bytes
- External fragment: small free blocks

Sequential Fit

Allocation of free blocks

- Common sequential fit algorithms
  - first fit
  - best fit
  - worst fit

Sequential Fit

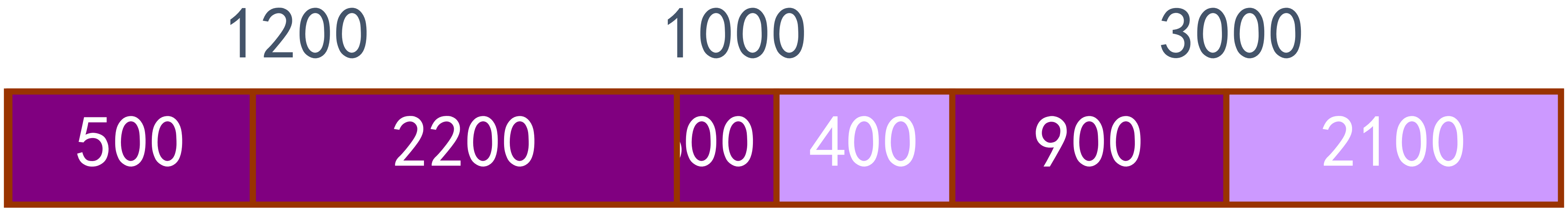


- 3 Blocks 1200，1000，3000
  request sequence: 600, 500, 900, 2200

- first fit：

Sequential Fit

- best fit

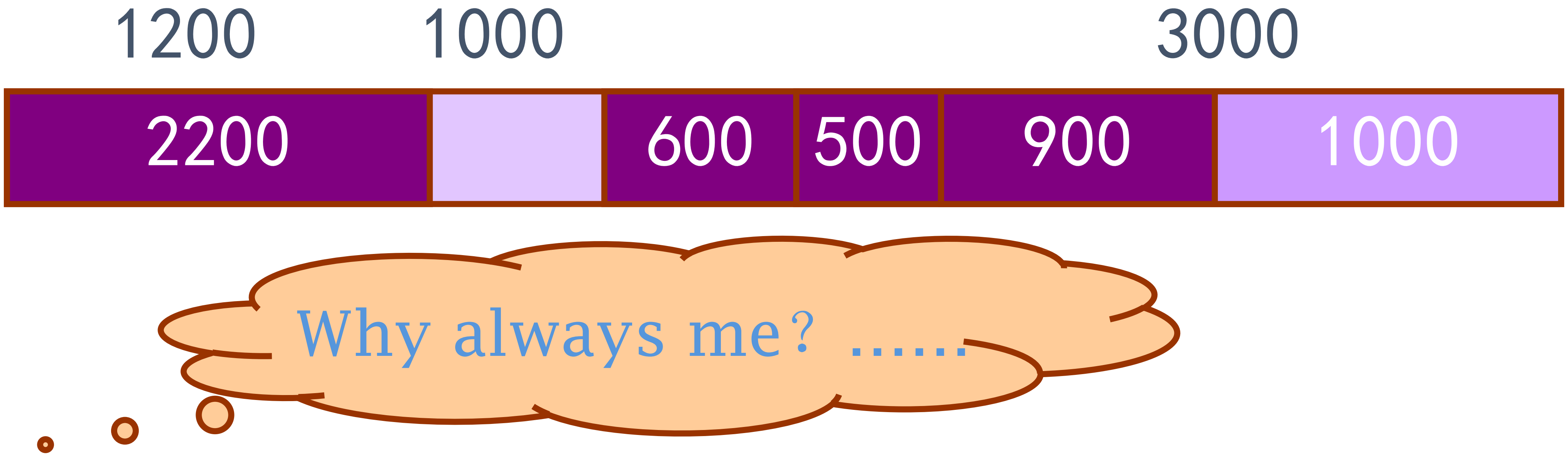

1200 1000 3000

500 | 2200 | 00 | 400 | 900 | 2100

request sequence: 600, 500, 900, 2200

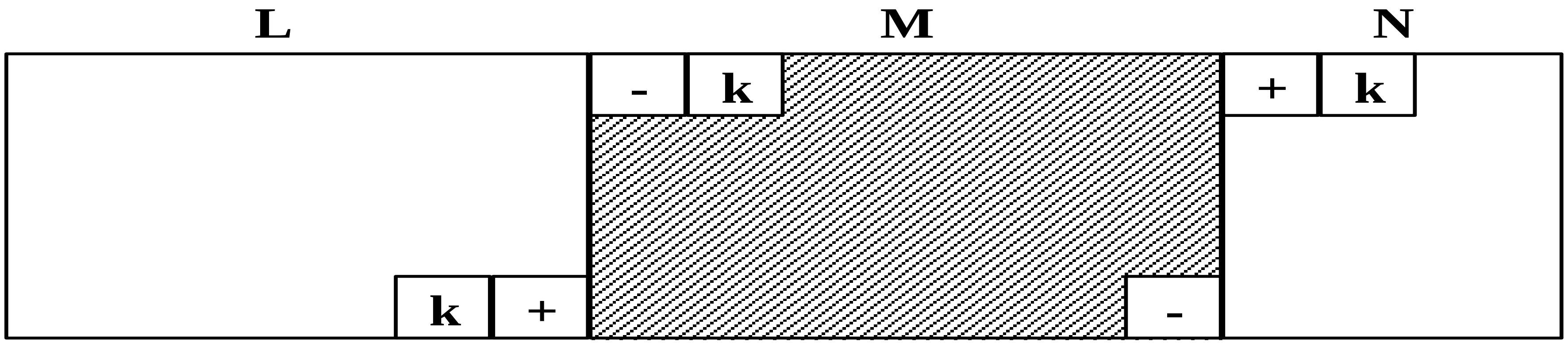
Sequential Fit

- worst fit



request sequence: 600, 500, 900, 2200

Reclamation: merge adjacent blocks



allocate block M back to the freelist

Fitting Strategy Selection

- Need to take the following user request into account
  - Importance of allocation and reclamation efficiency.
  - Variation range of the length of allocated memory
  - Frequency of allocation and reclamation
- In practice, fist fit is **the most commonly used.**
  - Quicker allocation and reclamation.
  - Support random memory requests.

Hard to decide which one is the best in general.

Failure Policy and Collection of Useless Units

- If a memory request cannot be satisfied because of insufficient memory, the memory manager has two options:
    - do nothing, and return failure info;
    - follow failure policy to satisfy requests.

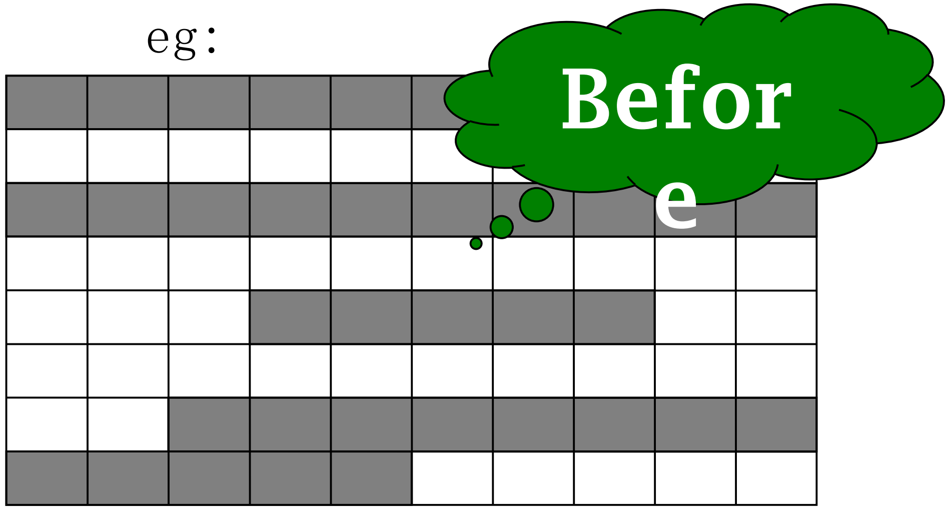
Compaction

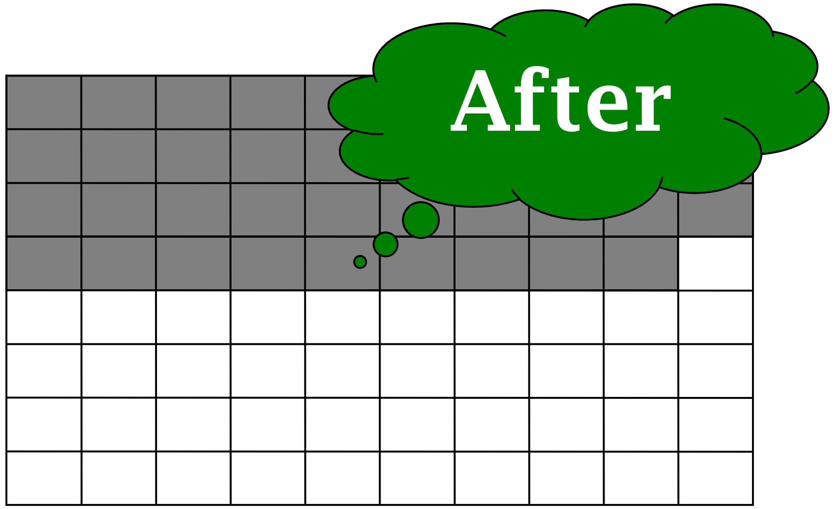
- Collect all the fragments together
  - Generate a larger free block.
  - Used when there are a lot of fragments.
- Handler makes the address relative
  - Secondary indirect reference to the storage location.
  - Only have to change handlers to move blocks.
    - No need to change applications.

Two Types of Compaction

- Perform a compact once a block is freed.
- Perform a compact when there is not enough memory or when collecting useless units.

eg:





Collecting Useless Units

- Collecting useless units: the most complete failure policy.
  - Search the whole memory, and label those nodes not belonging to any link.
  - Collect them to the freelist.
  - The collection and compaction processes usually can perform at the same time.

Data Structures and Algorithms

Thanks

the National Elaborate Course (Only available for IPs in China)
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/
Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (awarded as the "Eleventh Five-Year" national planning textbook)