CC WEEK 8

Prepared for: 7th Sem, CE, DDU

Prepared by: Niyati J. Buch

Heap Storage Allocation

- This strategy involves the reserving of a large contiguous block of memory commonly called the **heap**.
- **Heap** is used for allocating space for objects created at run time.

  e.g. nodes of dynamic data structures like linked lists & trees.
- Dynamic memory allocation and deallocation are based on the requirements of the program
  - C : manual: using malloc and free
  - C++: manual: using new and delete
  - Java: semi-automatic: using new and garbage collection
  - Lisp: automatically by runtime system

Memory Manager

- **Heap Memory Manager** manages heap memory by implementing the mechanisms for allocation and deallocation.
- **Goals**
  - space efficiency to minimize fragmentation
  - program efficiency by taking advantage of locality of objects in memory and make the program run faster
  - low overhead by efficient allocation and deallocation
- Heap is maintained either as a **doubly linked list** or as **bins** of free memory chunks.

Allocation and Deallocation

- Initially, the heap is **one large and contiguous** block of memory.

- As **allocation** requests are satisfied, chunks are cut off from this block and given to the program.

- As **deallocations** are made, chunks are returned to the heap and are free to be allocated again (holes).

- After a number of allocations and deallocations, memory becomes **fragmented** and is not contiguous.

Allocation and Deallocation

- Allocation from a fragmented heap may be made either in a **first-fit** or **best-fit** manner.

- After a deallocation, we try to **coalesce** (join together) contiguous holes and make a bigger hole (free chunk)

First-Fit and Best-Fit Allocation Strategies

- The **first-fit** strategy picks the **first** available chunk that satisfies the allocation request.
- The **best-fit** strategy searches and picks the **smallest** (**best**) possible chunk that satisfies the allocation request.

- Both strategies chop off a block of the required size from the chosen chunk, and return it to the program.
- And the rest ***remains*** in the heap.

- **Best-fit** strategy has been shown to reduce fragmentation in practice, better than first-fit strategy.

Next-fit strategy

- **Next-fit strategy** tries to allocate the object in the chunk that has been ***split recently***
  - Tends to improve speed of allocation
  - Tends to improve spatial locality since objects allocated at about the same time tend to have similar reference patterns and life times

    (cache behaviour may be better)

- The overall the speed of the programming increases; when **doubly linked list** approach for storing/managing heaps is used.

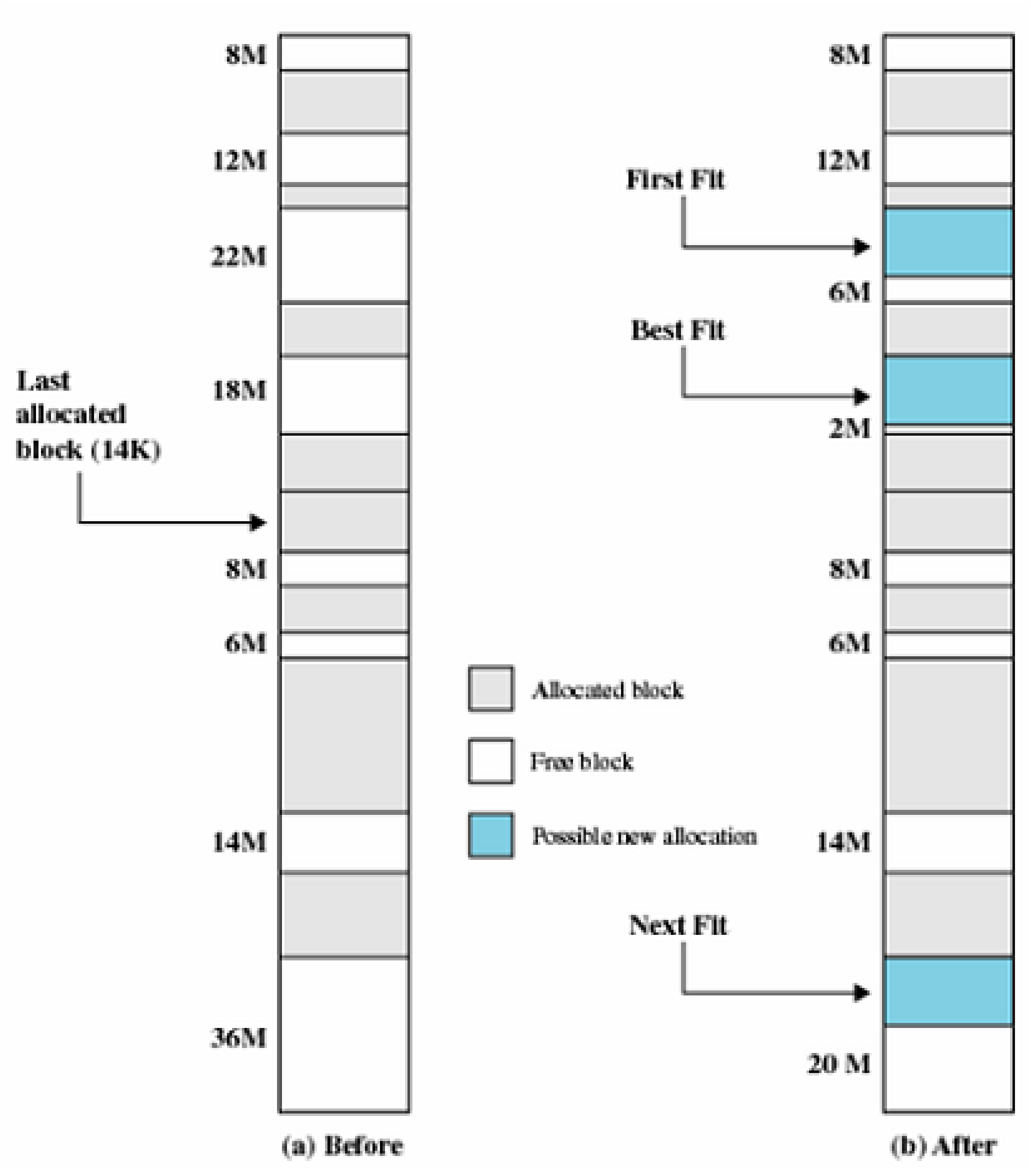
Summary

- **Best-Fit**:
  - Closest in size to the request .

- **First-Fit**:
  - Scans the main memory from the beginning and first available block that is large enough .

- **Next-Fit**:
  - Scans the memory from the location of last placement and chooses next available block that is large enough.

Example: Allocation of 16 MB block using three placement algorithms



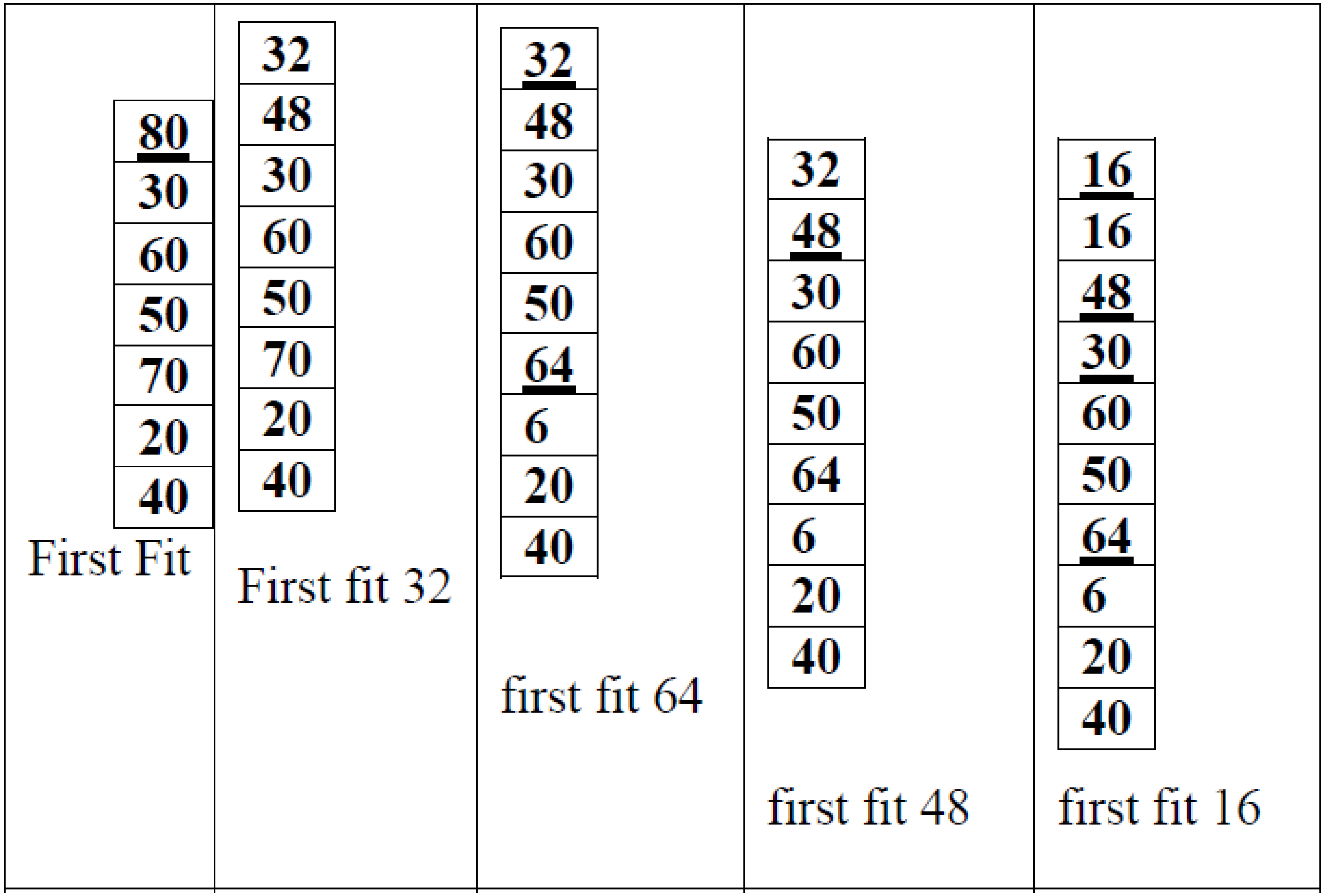
Example

(ex. From Aho-Ullman book)

- Suppose the heap consists of **seven chunks**, starting at address 0. The sizes of the chunks, in order, are **80, 30, 60, 50, 70, 20, 40 bytes**.
- When we place an object in a chunk, we put it at the high end if there is enough space remaining to form a smaller chunk (so that the smaller chunk can easily remain on the linked list of free space).
- However, we cannot tolerate chunks of fewer that 8 bytes, so if an object is almost as large as the selected chunk, we give it the entire chunk and place the object at the low end of the chunk.
- If we request space for objects of the following sizes: **32, 64, 48, 16,** in that order, what does the free space list look like after satisfying the requests, if the method of selecting chunks is

  (a) First Fit. (b) Best Fit.

Solution: First Fit



80
30
60
50
70
20
40

First Fit

32
48
30
60
50
70
20
40

First fit 32

32
48
30
60
50
64
6
20
40

first fit 64

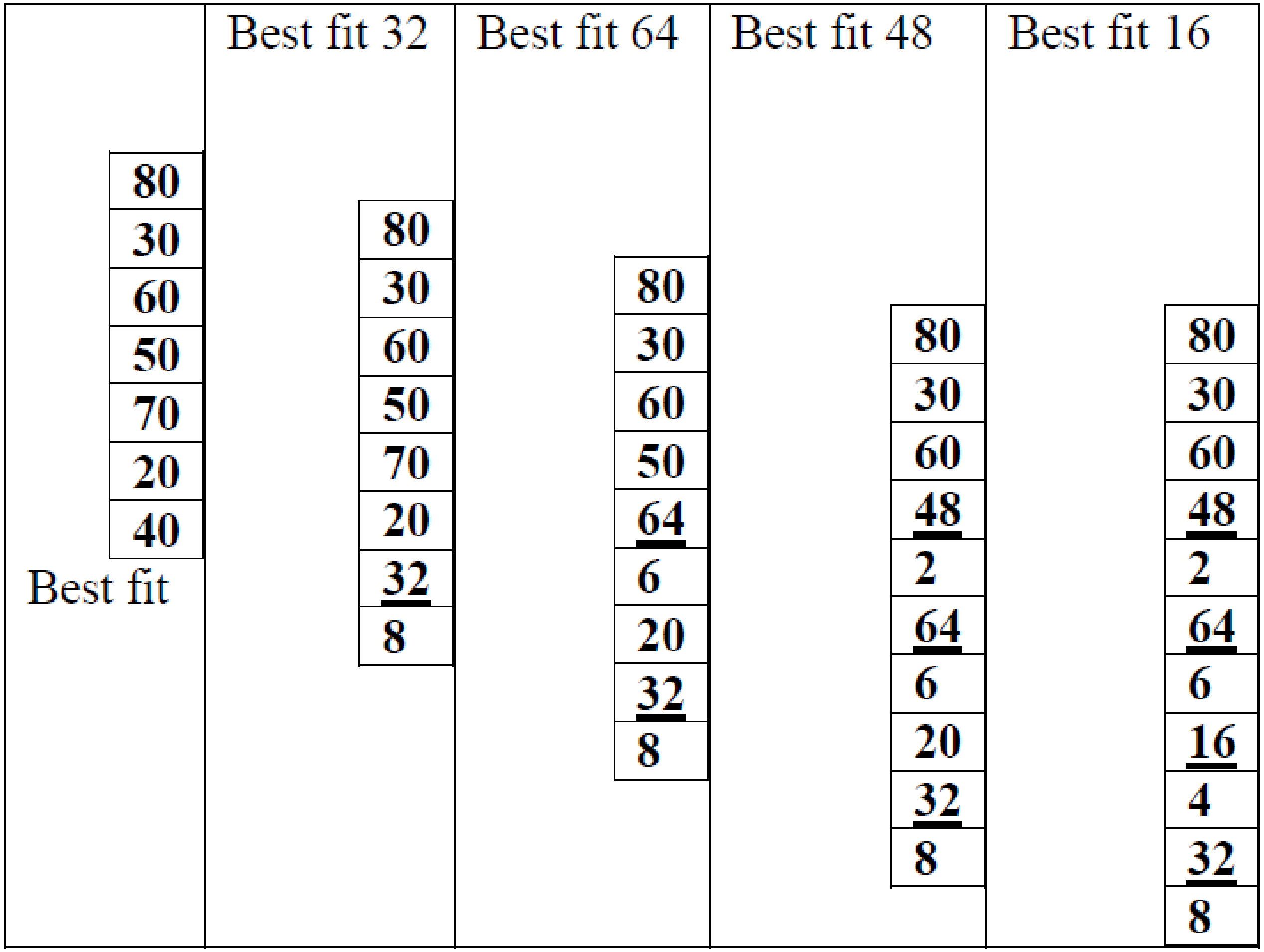
32
48
30
60
50
64
6
20
40

first fit 48

16
16
48
30
60
50
64
6
20
40

first fit 16

Solution: Best Fit



Best fit

80
30
60
50
70
20
40

Best fit 32

80
30
60
50
70
20
<u>32</u>
8

Best fit 64

80
30
60
50
<u>64</u>
6
20
<u>32</u>
8

Best fit 48

80
30
60
<u>48</u>
2
<u>64</u>
6
20
<u>32</u>
8

Best fit 16

80
30
60
<u>48</u>
2
<u>64</u>
6
<u>16</u>
4
<u>32</u>
8