When a process is to be loaded in memory (RAM) there may be some free spaces in RAM and some preoccupied space. So, we have three basic methods for allocation of space to the incoming process:

First Fit
In the first fit approach is to allocate the first free partition or hole large enough which can accommodate the process.

Best Fit
The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate.

Worst fit
In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit.

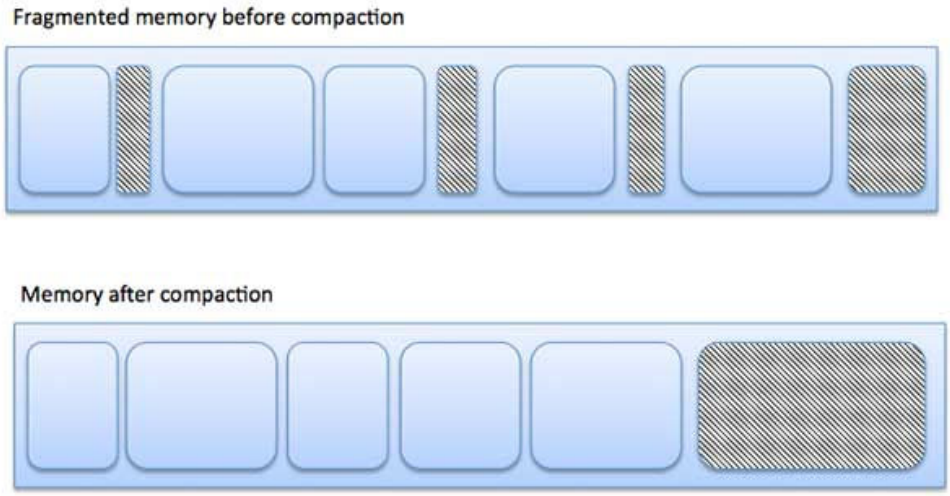
As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. Even though the total free space may be larger than the required size but the small chunks are not together so there is no continuous memory chunk sufficient for the incoming process. This problem is known as (extremal) **Fragmentation**.
Even if a block is allocated to a process another type of fragmentation is possible where the block allocated may be slightly bigger than that needed by the process, thus creating internal fragmentation.

Fragmentation is of two types –

1	**External fragmentation** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
2	**Internal fragmentation** Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

External fragmentation can be reduced by **compaction** or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.



---

garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is erformed automatically. So, java provides better memory management

Main objective of Garbage Collector is to free heap memory by destroying **unreachable objects**

There are many ways for creating objects eligible for garbage collection:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.

The basic process can be described as follows.

Step 1: Marking

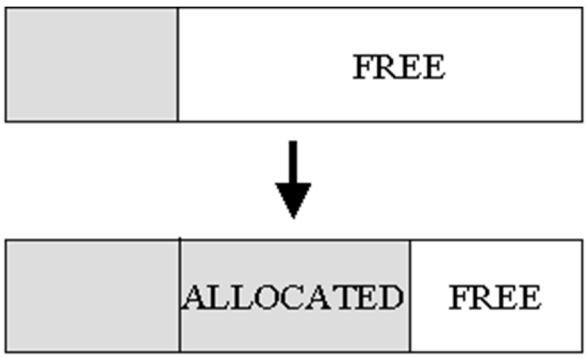
The first step in the process is called marking. This is where the garbage collector identifies which pieces of memory are in use and which are not.

Step 2: Normal Deletion

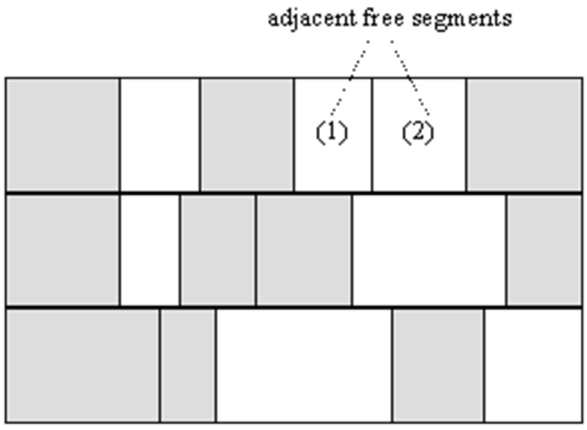
Normal deletion removes unreferenced objects leaving referenced objects and pointers to free space.
To further improve performance, in addition to deleting unreferenced objects, you can also compact the remaining referenced objects. By moving referenced object together, this makes new memory allocation much easier and faster.

Boundary Tag Method.

Assume as given that memory is to be allocated from a large area, in contiguous blocks of varying size, and that no form of compaction or rearrangement of the allocated segments will be used. Now from a continuous FREE chunk the required chunk will be allocated (figure A). With repeated allocation and deallocation(after completion of the process) we could in a scenario as shown in figure b. Now you can see there are two continuous free chunks, may be freed at separate timings when the occupying process completed. But they are now as two separate free chunks of memory and if the next process needed memory more than the individual chunks but less than both in total then too it would be able to get the space as the two chunks are treated as two by the system .



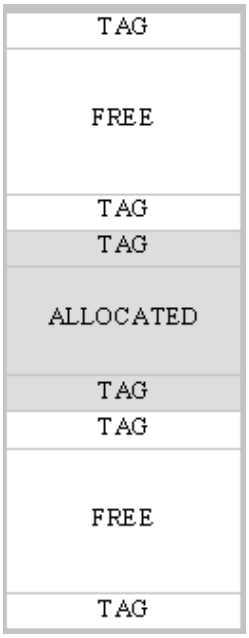
(a) Allocating a segment of contiguous memory



(b) Memory after several allocations and releases

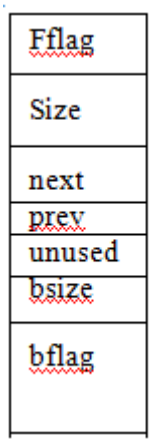
FIGURE C-1. ALLOCATION OF A MEMORY REGION

Now with boundary tag method each block allocated or free has special tags at the start and end of the chunk (hence called boundary tag). Besides the length of the chunk, it also contains a flag to indicate if the chunk is free or allocated. When two such free segments come together the flags at the end of one segment and the start of the next segment both will have flag marked as free and hence would be merged adding the length updated and having boundary tags now at the start of the one block and end of the other block thus effectively merging the free chunks.



After the deallocation of the allocated space

In addition there can be pointers to the next and previous free segment. A sample tag for a free block could look like this



Fflag and bflag will be false for a free block and true for an allocated block.

BINARY BUDDY SYSTEMS

Buddy system is a trade-off between limited process handling ability of static partitioning and complication of dynamic partitioning

This **buddy system** is a memory allocation and management algorithm that manages memory in **power of two increments**. Assume the memory size is 2^N, suppose a size of S is required.

- **If $2^{N-1}<S<=2^N$:** Allocate the whole block
- **Else:** Recursively divide the block equally and test the condition at each time, when it satisfies, allocate the block and get out the loop.

System also keep the record of all the unallocated blocks each and can merge these different size blocks to make one big chunk.

Advantage –

- Easy to implement a buddy system
- Allocates block of correct size
- It is easy to merge adjacent holes
- Fast to allocate memory and de-allocating memory

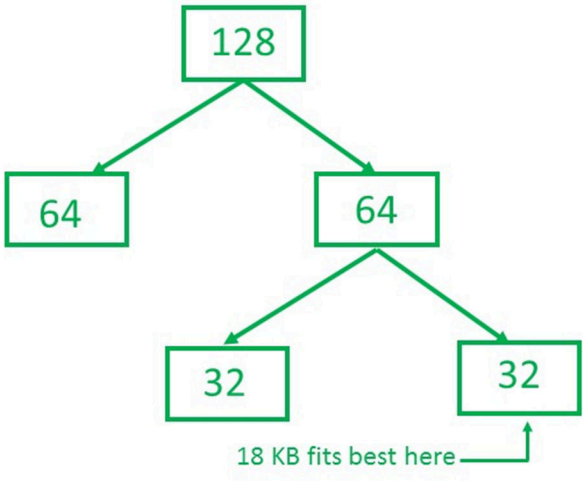
Disadvantage –

- It requires all allocation unit to be powers of two
- It leads to internal fragmentation

Example –

Consider a system having buddy system with physical address space 128 KB. Calculate the size of partition for 18 KB process.

Solution –

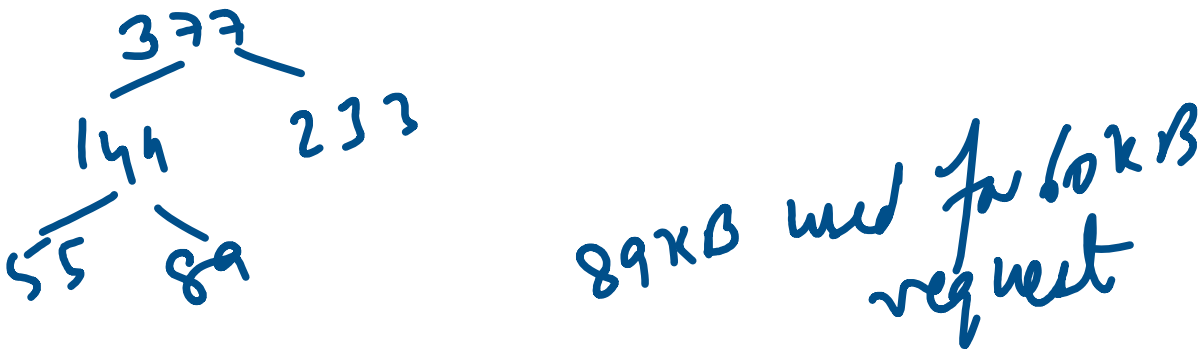


So, size of partition for 18 KB process = 32 KB. It divides by 2, till possible to get minimum block to fit 18 KB.

FIBONACCI BUDDY SYSTEMS

It is similar to the binary buddy system except the partitioning is not done on factors of 2 but rather the Fibonacci sequence is used (1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2582,.....)

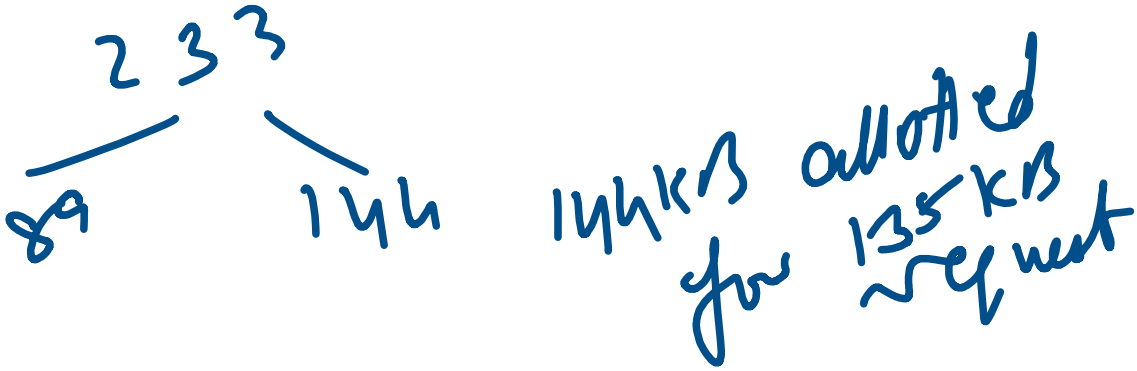
Lets take an example: We need to allocate 60KB from block of 377KB



Now the memory is like

55KB FREE	89KB ALLOCATED	233KB FREE

Now lets say 135KB is demanded



Now the memory is like

55KB FREE	89KB ALLOCATED	89KB ALLOCATED	144KB FREE