Buckets Presentation

ACU 2025 Team

Copyright

Rules

- You must have downloaded your copy from the Assistants' Intranet <https://intra.forge.epita.fr>.
- This document is strictly personal and must **not** be passed on to someone else.
- Non-compliance with these rules can lead to severe sanctions.

Introduction

What's different from other algorithms ?

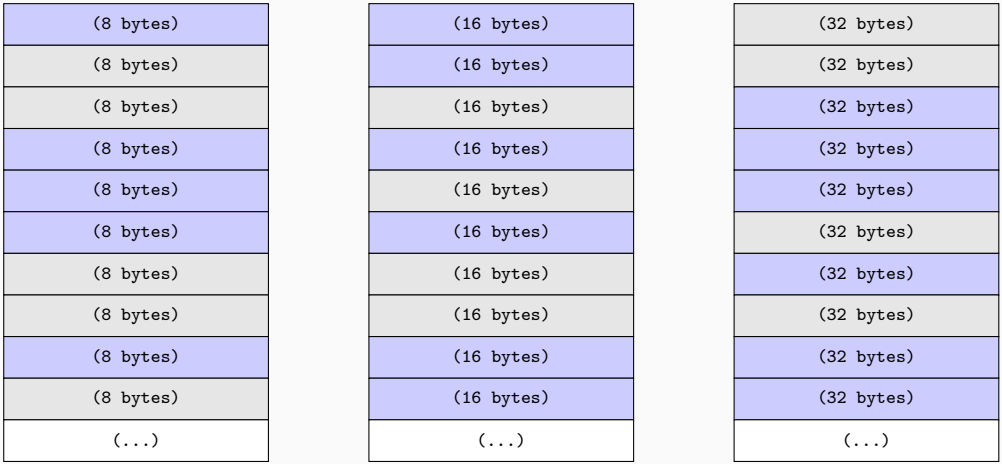
- No fragmentation.
- Very fast allocation time.
- Poor memory efficiency.

Principle

Principle

- The idea is to have a mapped space called a bucket divided into blocks of the same given size.
- The size of these blocks is always a power of 2 (8 bytes, 16 bytes, etc.).
- When the user wants memory, `malloc` chooses the bucket with the smallest block size that can fit the data.
- There is no metadata stored with the data inside the block.
- Therefore, a free-list or any other structure to know which blocks of a bucket are free is mandatory.

Possible layouts



Management

Allocation

- User performs a `malloc` of size `s`.
- Compute the first power of two greater than or equal to `s`: `s2p`.
- Find a bucket with blocks of size `s2p`. If there are none or if they are all full, create a new one.
- Use the free-list to get the address of the first free block of the bucket.
- Return this address and mark it as a used block in the free-list.

Free

- From the given address, you can determine where the start of the page is and find the metadata of the bucket.
- Update the free-list and mark this block as free.
- If the bucket has all its blocks free, you can unmap it.

Metadata

Metadata

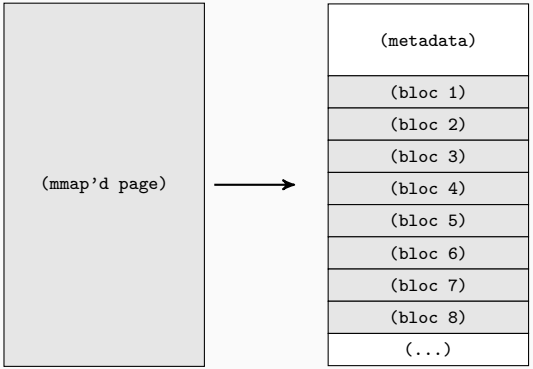
At the beginning of each block you need some metadata, with at least:

- The size of the blocks in this bucket.
- The free-list or a pointer to it.

This metadata structure needs to be put at the start of the memory page. This way, you will be able to find the metadata of a bucket from the address of one of its blocks by searching the start of the page (used for `free`).

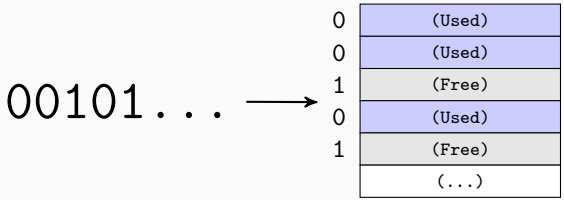
You also need to store those metadata structures in a linked list manner. You can either store all of them in the same linked list or have separate lists for buckets with blocks of the same size.

Metadata


(mmap'd page)
(metadata)
(bloc 1)
(bloc 2)
(bloc 3)
(bloc 4)
(bloc 5)
(bloc 6)
(bloc 7)
(bloc 8)
(...)

Flag method :

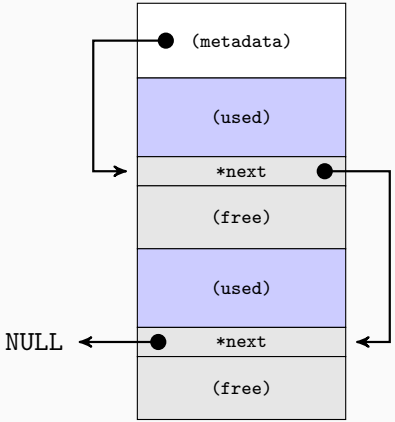
- You store a variable with enough bits in your metadata.
- You use its bits to indicate if a block is used or not.
- The value of the n-th bit represents the state of the n-th block of the bucket.
- You can then easily determine the address of the block knowing n and the size of this bucket's blocks.



Free-list

Internal linked list

- You store the elements of the free list **inside** the free blocks.
- You use the address of the element itself to indicate the address of the free block.
- Therefore, you only need to store the address of the next element of the list inside of your structure
- The metadata stores a pointer to the first element of this free-list.



Example

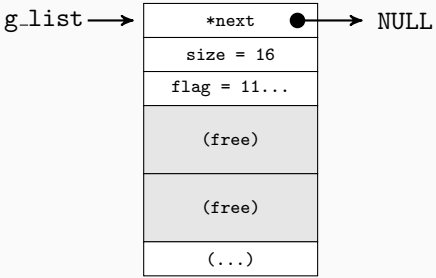
Step 1

```
void *p1 = malloc(15);
```

- The upper power of 2 of 15 is 16.
- We search for a bucket with of size 16.

Step 2

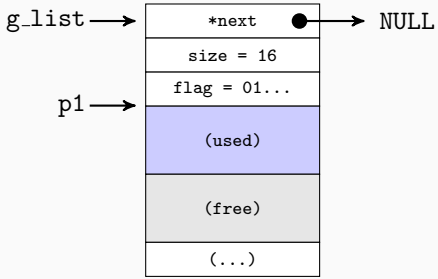
```
void *p1 = malloc(15);
```



- There are none so we create it.
- The new bucket with blocks of size 16 is initialized with all its blocks free.

Step 3

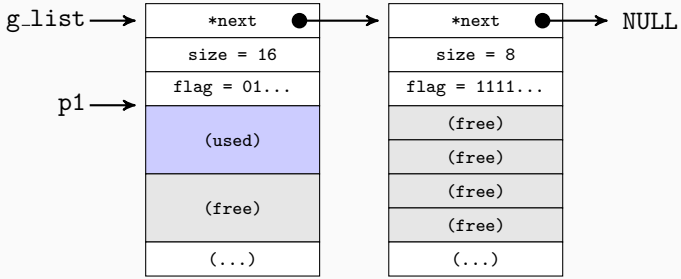
```
void *p1 = malloc(15);
```



- We check the free-list to get the first free block of the bucket.
- We return the address of the block and update the bucket's free-list.

Step 4

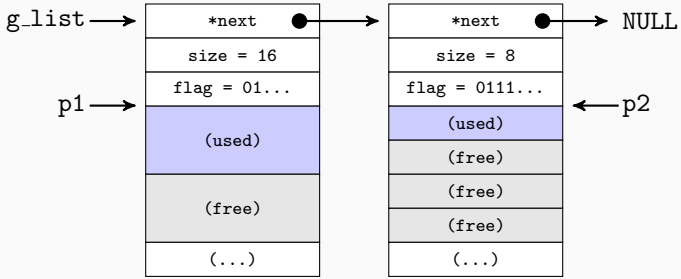
```
void *p1 = malloc(15);
void *p2 = malloc(6);
```



- The upper power of 2 of 6 is 8.
- We search for a bucket of size 8.
- There are none so we create it and add it to our linked list of buckets.

Step 5

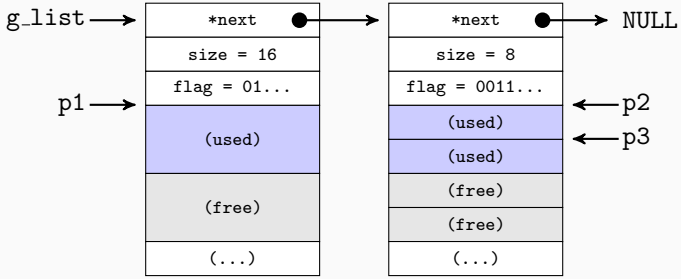
```
void *p1 = malloc(15);
void *p2 = malloc(6);
```



- We check the free-list to get the first free block of the bucket.
- We return the address of the block and update the bucket's free-list.

Step 6

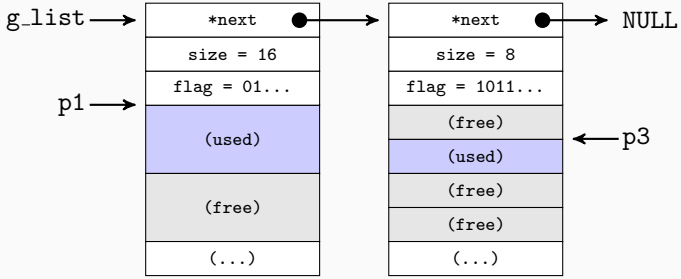
```
void *p1 = malloc(15);
void *p2 = malloc(6);
void *p3 = malloc(8);
```



- The upper power of 2 of 8 is 8.
- We check the free-list to get the first free block of the bucket.
- We return the address of the block and update the bucket's free-list.

Step 7

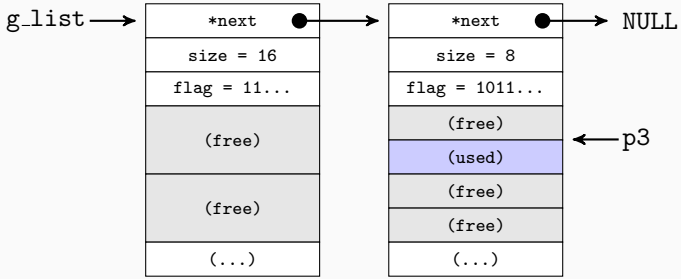
```
void *p1 = malloc(15);
void *p2 = malloc(6);
void *p3 = malloc(8);
free(p2);
```



- From the address given, we determine the start of the page to find the metadata of the bucket.
- We update the free-list to mark the block as free.

Step 8

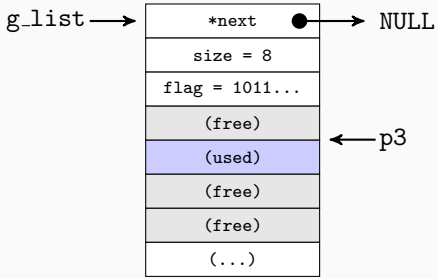
```
void *p1 = malloc(15);
void *p2 = malloc(6);
void *p3 = malloc(8);
free(p2);
free(p1)
```



- We do the same thing here but for another bucket.

Step 9

```
void *p1 = malloc(15);
void *p2 = malloc(6);
void *p3 = malloc(8);
free(p2);
free(p1)
```



- The bucket now only contains free blocks, we can unmap it.
- We update our linked list of buckets.

Conclusion

Conclusion

- Buckets with all blocks of the same size.
- Very fast allocation and no fragmentation but not very memory efficient.
- Free-list is mandatory.