# Dynamic Memory Management

Tips: Draw the graph of free nodes out.

## Split block or Merge block ?

- Split is good when malloc size is fixed (predictable)
- Merge is good when malloc size is unpredictable

## Jargons:

- Kernel panic: critical system error where the OS detects a fatal internal error it can't recover from, forcing the system to a halt. Similar to "Blue Screen of Death".
- Segmentation fault: happens when accessing invalid memory address
  Example: Dereferencing null pointer, trying to access address that is not in the system
- Memory block's status saved in header:
  The header stores size of the block + 1 bit to store the status
  Example: A block size 112 will have header stores 113, the extra 1 bit determine the status (free / allocated)
  Note: if the header's struct doesn't explicitly stated this, assume there is no status bit.
- Memory leak:
  - Allocated blocks that are not freed properly.
  - Free blocks that are inaccessible (not connected to the free list)
- Coalscing: merging adjacent blocks, happens in either:
  - Allocation: when scanning the free list, adjancent free blocks can be merged to match the required allocatio request.
  - Freeing: merging adjacent free blocks to prevent fragmentation.

  Types of coalsecing for free blocks
  1. Immediate: merge right when freeing blocks.
  2. Deferred: wait until allocation to merge.

  Note: Allocation only use deffeired coalsecing.

## Memory blocks

What is inside a memory block (usually represent an instanciated object) ?

Memory block's structure

| | | |
|---|---|---|
| Header | ---> | Metadata (size, status, pointers) |
| Payload | ---> | Actual data (class properties, object properties) |
| Footer | ---> | Optional, also store some metadata |

How do blocks setup in the heap ?

| Address | Contents | |
|---|---|---|
| 0x1000 | 8 | block 1 |
| 0x1008 | 0x1080 | |
| 0x1010 | [Payload] | |
| 0x1018 | 64 | block 2 |
| 0x1020 | 0xBEEF | |
| 0x1028 | [Payload] | |
| 0x1068 | 8 | block 3 |
| 0x1070 | 0xBEEF | |
| 0x1078 | [Payload] | |
| 0x1080 | 128 | block 4 |
| 0x1088 | 0x1110 | |
| 0x1090 | [Payload] | |
| 0x1110 | 512 | block 5 |
| 0x1118 | 0x1000 | |
| 0x1120 | [Payload] | |

Structure of block:

| | |
|---|---|
| block_size | header |
| next_pointer | |
| class_pointer | payload |
| superclass_pointer | |
| vtable_pointer | |
| ⋮<br>data members<br>⋮ | |

## Malloc and Free methods

- Malloc (size)

  Allocate memory based on size, return the address to the payload (also update the metadata in the header)

  Example: you need to allocate space in the heap for object sized 8 bytes, then your malloc should be:

  ```
  void* ptr = malloc ( 8 + header_size )
  ```

- Free (payload_address)

  Deallocate the memory in the payload_address, return nothing. (ponentially perform coalsecing, by accessing the header's metadata, traverse to the next free block, and merge)

# Malloc Policies:

## First-fit

Choose the first free block that fits.

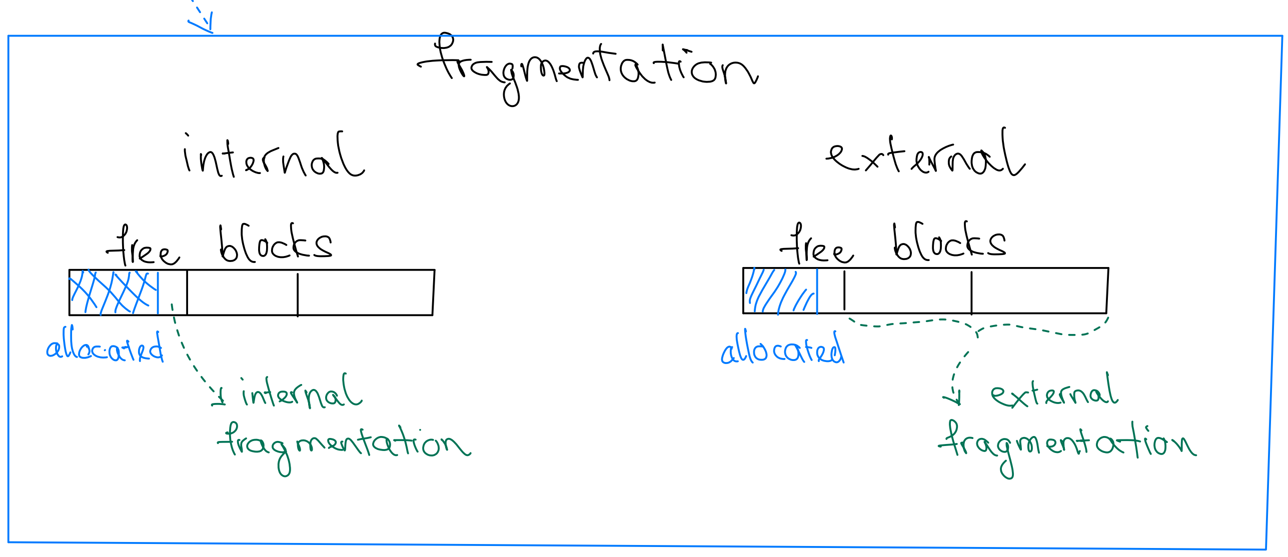
- Pros:
  - Fast. $O(n)$
  - Easy to implement
- Cons:
  - Fragmentation, as smaller free blocks are left scattered throughout the memory.

    → Performance degrade, as harder to find space for larger block.

## Best-fit

Scan all free blocks, choose the smallest block that fits.

- Pros:
  - More efficient use of memory, as it minimize leftover space
  - Less prone to external fragmentation




- Cons:
  - Slower, as require scanning all free blocks
  - Increased internal fragmentation, smaller leftover blocks may not be useable.

## Worst-fit

Scan all free blocks, choose the largest possible.

Pros:

- Aim to leave a larger hole for future requests.

Cons:

- Slow, as it scan through all free blocks.
- Often leads to internal fragmentation, as larger blocks are splitted into unusuable smaller blocks.

## Next-fit: (variation of First-fit)

Just like First-fit, but continue searching from last point instead of starting from the beginning.

- Pros:
  - Faster than First-fit in scenario with many allocations
- Cons:
  - Same as First-fit, it can lead to fragmentation and poor memory use over time.