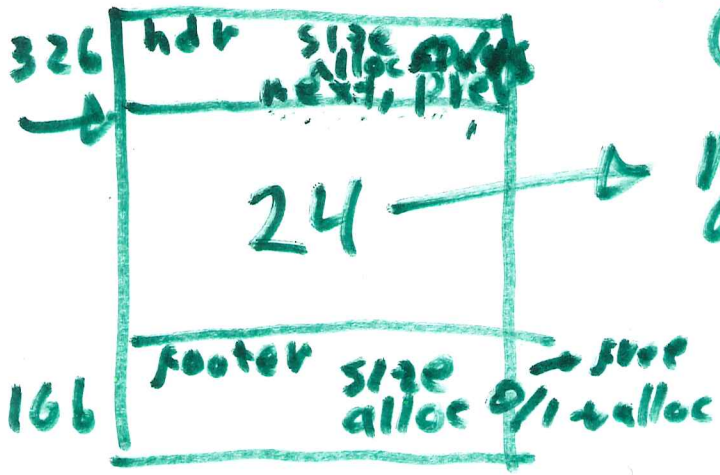
# Lab 1 malloc

```
int MemSize=20;
char *p = Malloc(MemSize);
```

= block




MemalignedSize = (MemSize + 7) & ~7;

Memory size is aligned to next 8byte boundary.

that is because a long or double have to be stored in 8byte boundary.

free(p)

- We need the header to store the size to know how many bytes to return to malloc freelist.

Header (32 bytes)

size - size of block including Header and footer (8 bytes)

alloc flags - 0 = free 1 = alloc 2 = sentinel. (8 bytes)

next, prev - Usage when object is free, to add to free list. (8+8) = 16 bytes

Footer (16 bytes)

Size - Size of block including header + footer (8 bytes)

Alloc flags - 0 = free 1 = alloc 2 = sentinel (8 bytes)

The smallest object is

MemSize = 1 byte → 8 bytes aligned sizes

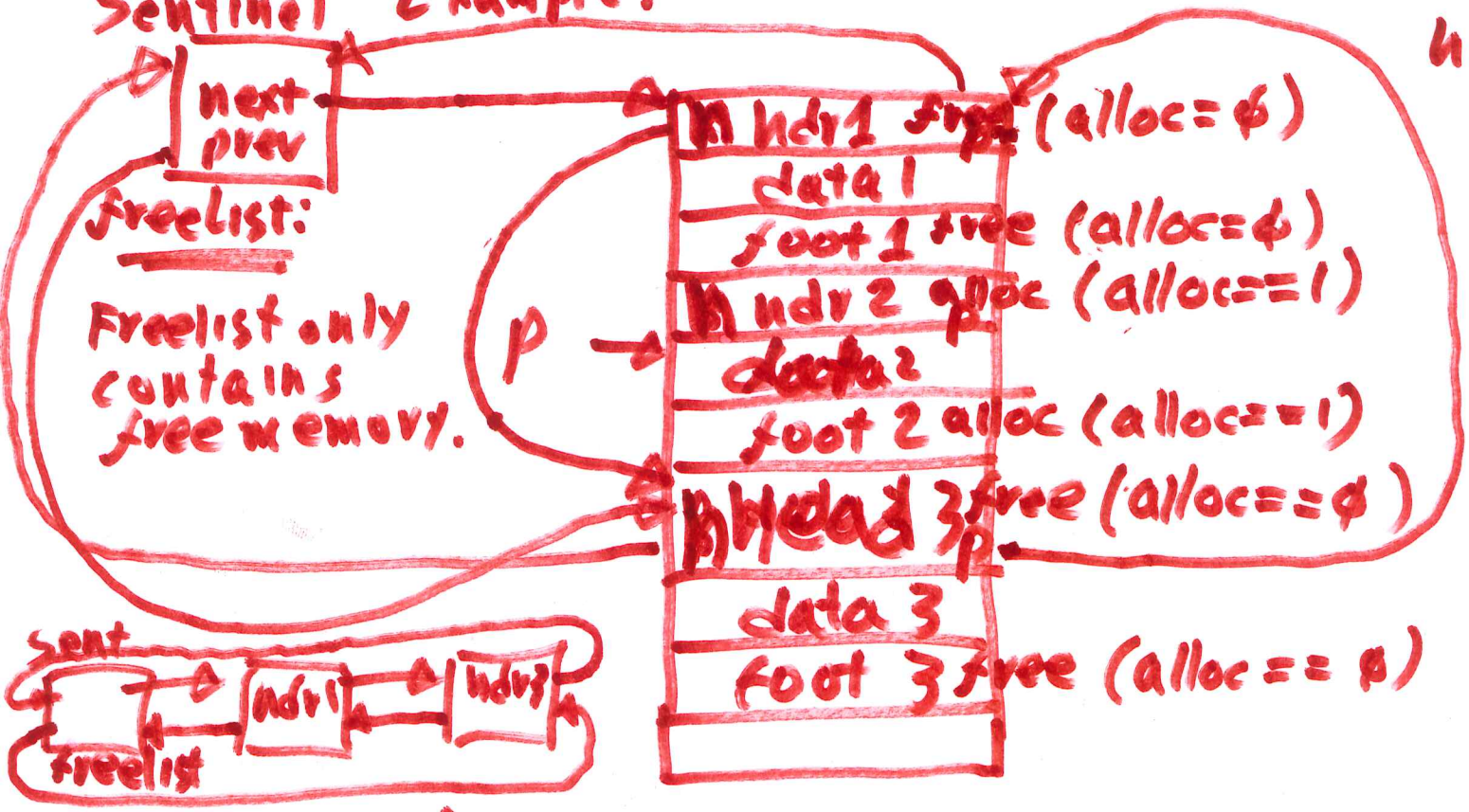
Header 32 bytes → 32

Footer 16 bytes + 16

56 bytes

The footer is used to coalesce blocks when they are freed.

Sentinel Example:




hdr & footer are also called "boundary tags"

We have two separate data structures
- Boundary tags
- Double linked list.

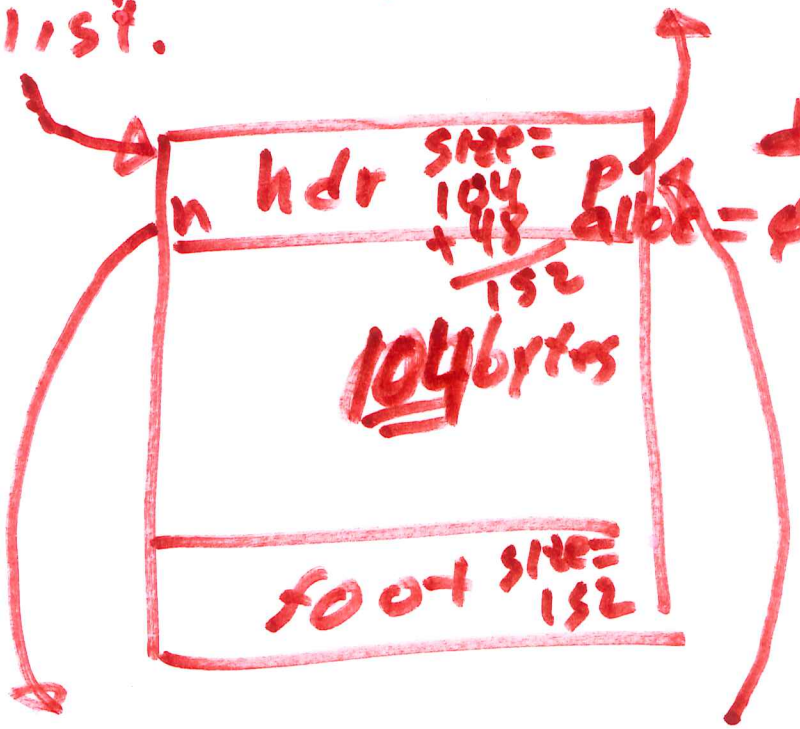
free(p)

Check if object before p is free.
If free then coalesce it.

p = malloc(memSize)

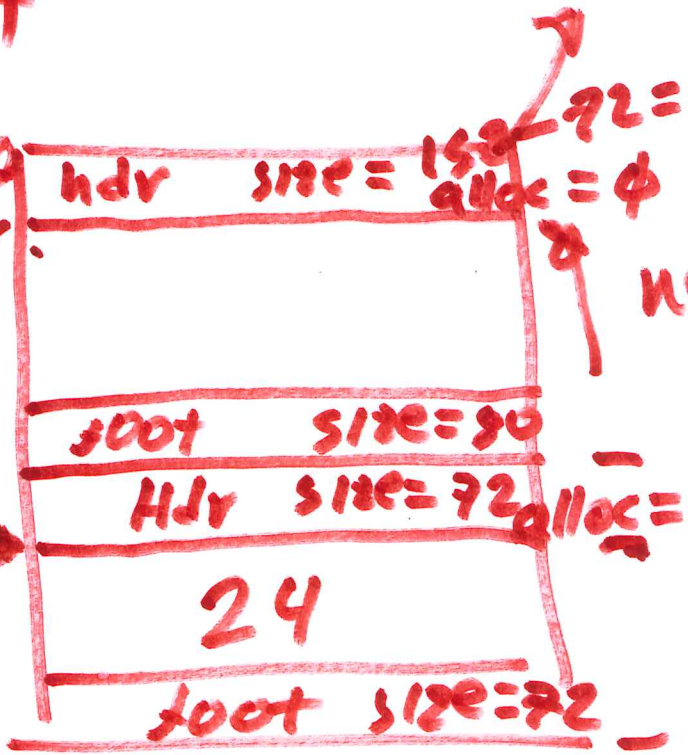
- find first block large enough to satisfy request.
- Try split object. If reminder is smaller than smallest usable object i.e. sizeof(Header) + sizeof(Footer) + 8, then don't split and return whole obj. else split object return reminder to free list.

p = malloc(20)




← object that needs to be split

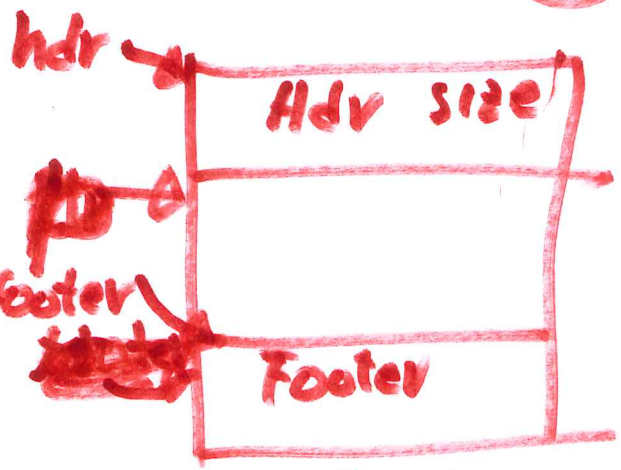
After Split

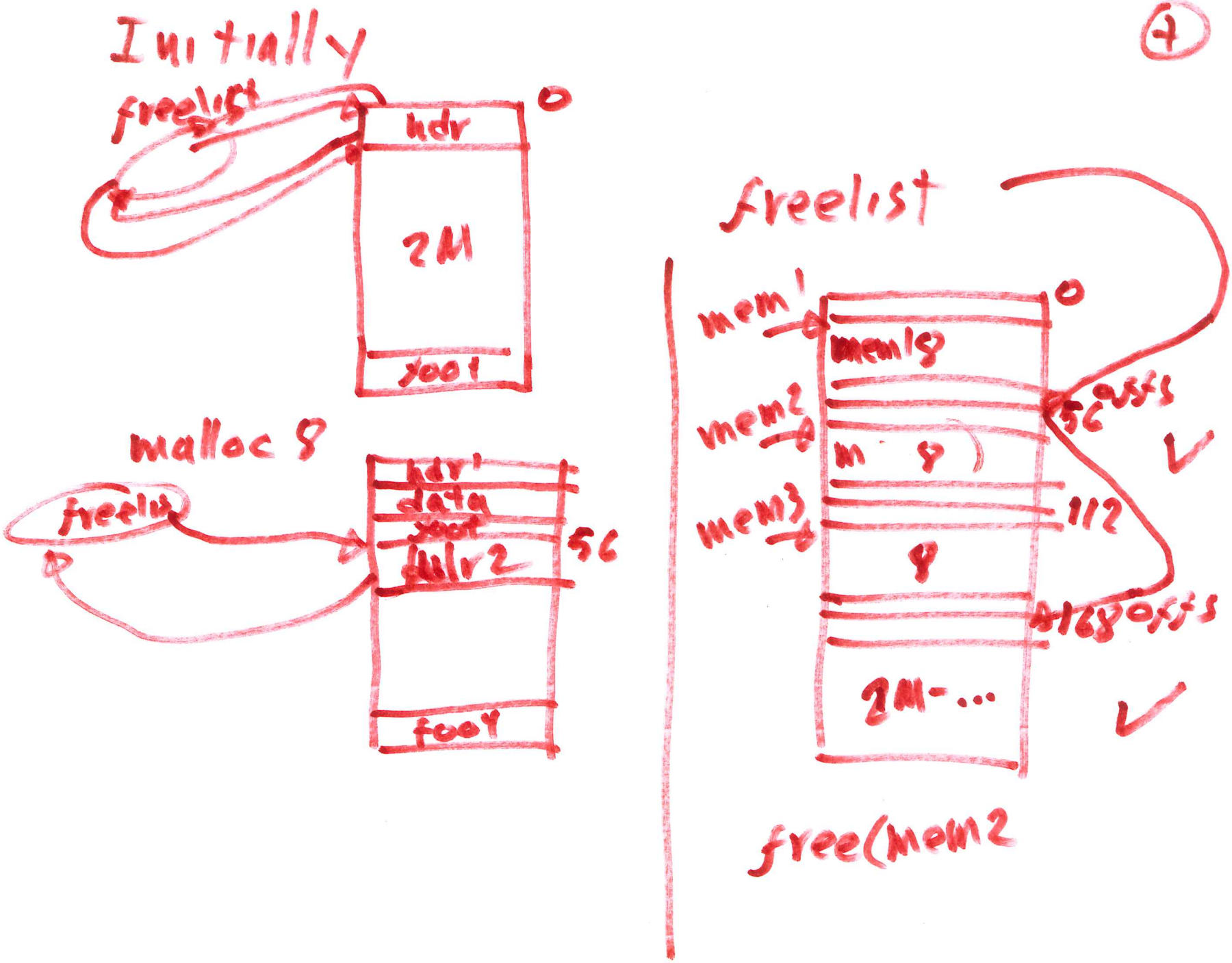



MemAligned =
24
+ Header 32
+ Foot 16
---
72

```
free(void *ptr) {
  Header * hdr =
    (Header*)((char*)p
      - sizeof(Header))

  Footer * footer =
    (Footer*)((char*)hdr
      + hdr->size - sizeof(Footer))
```




Initially
freelist
hdr
0
2M
foot
malloc 8
freelist
hdr
data
foot
hdr2
56
foot
freelist
mem1
mem2
mem3
0
mem1 8
56 bytes
V
m 8
112
8
168 bytes
2M-...
V
free(mem2