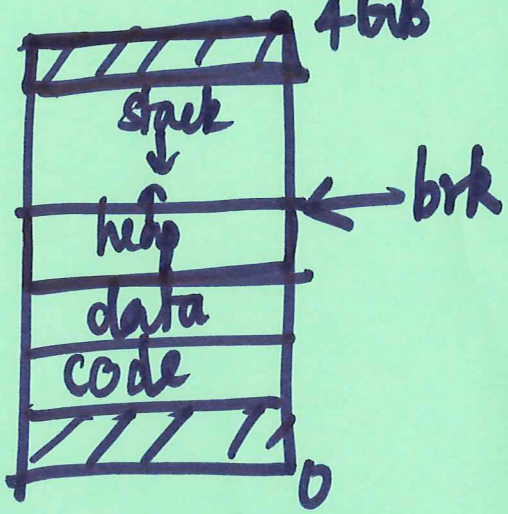
Lecture - 32

Review

Virtual address space

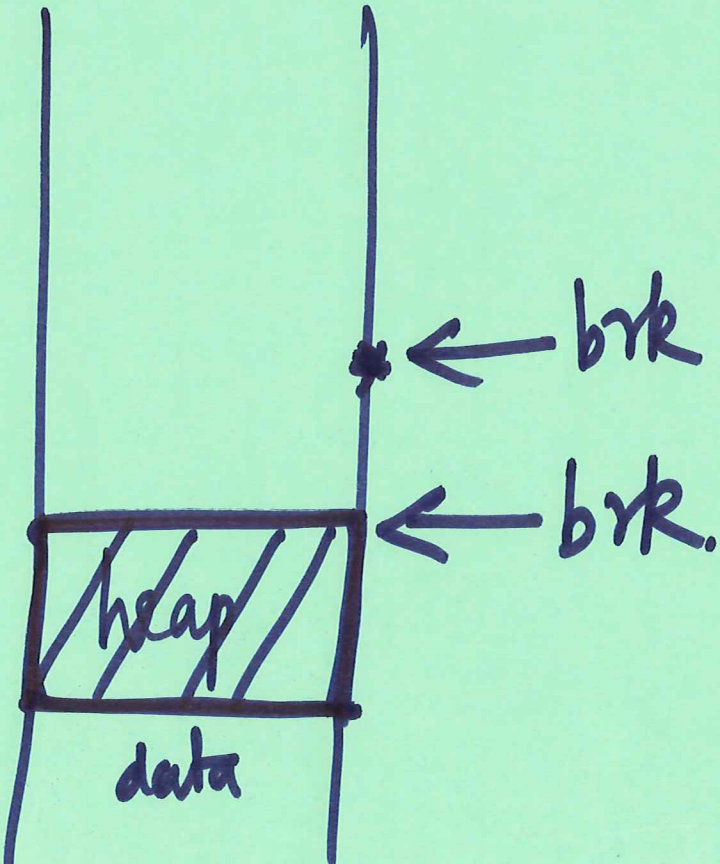
malloc, free,
calloc, realloc.



---

int brk (void *addr);

void * sbrk (int incr); — most used.



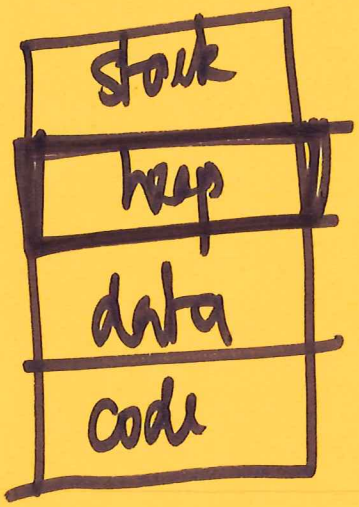
On error, sbrk return

(void *) -1

↓

0x FF FF FF FF

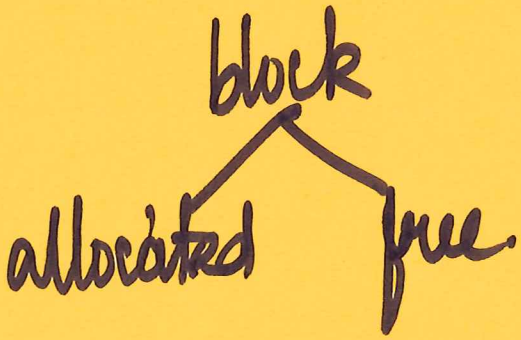
Allocators

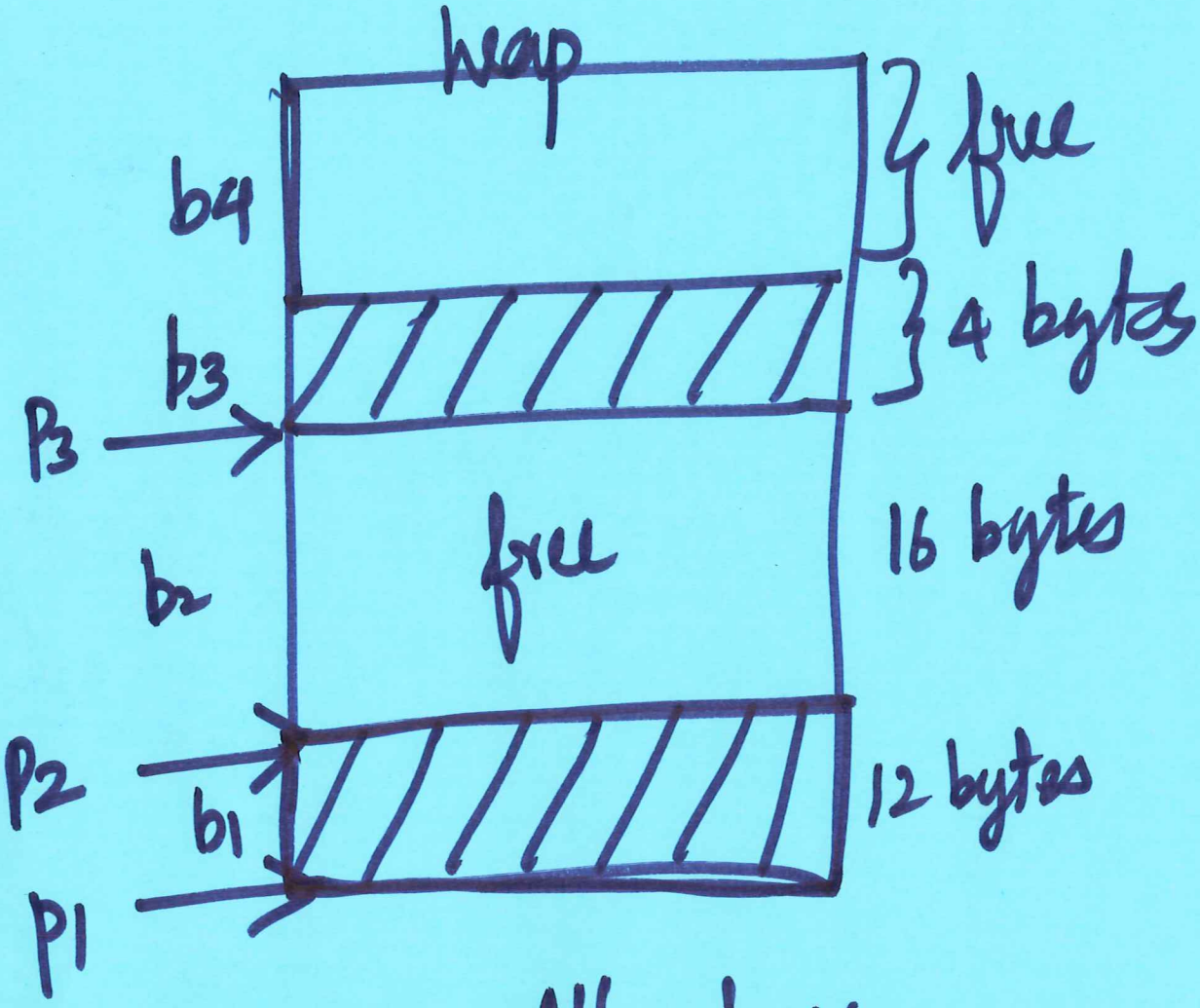


malloc

heap → collection of various-sized blocks

block → contiguous chunk of virtual mem.





Allocators

explicit

C - malloc, free

implicit

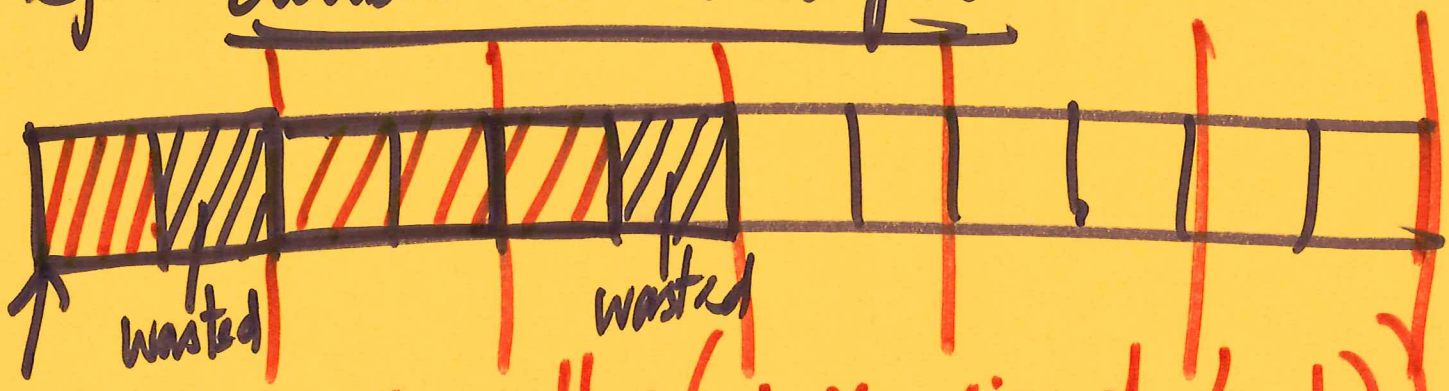
Garbage collection

"new"

free - automatic.

Alignment requirements

eg. double-word aligned



p1 = malloc(1 * sizeof(int));

In P5, align req. is word size.

word aligned.

malloc(1) → give you 4 bytes.

payload: 1 - used

padding: 3 - wasted

Implementation issues:

1. Free block organization.
2. Placement
3. Splitting free blocks.
4. Coalesting.

Requirements

1. handle arbitrary resquests.
2. Respond immediately.
3. Metadata → stored only in the heap.
4. Aligns the blocks.
5. NOT modify allocated blocks.

2 Goals

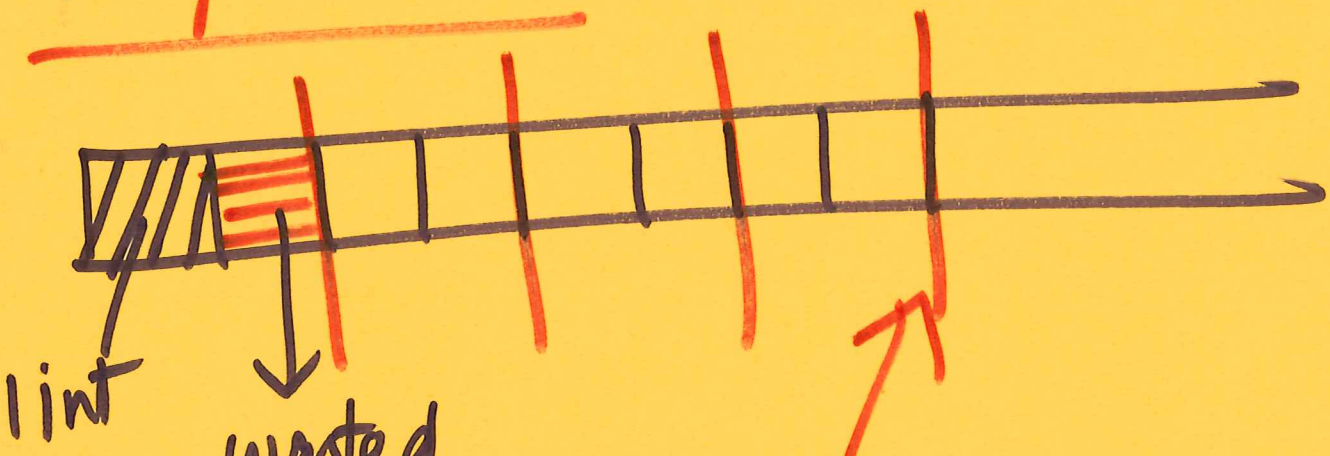
1. Maximize throughput

1000 mallocs + 1000 frees in 1 sec.

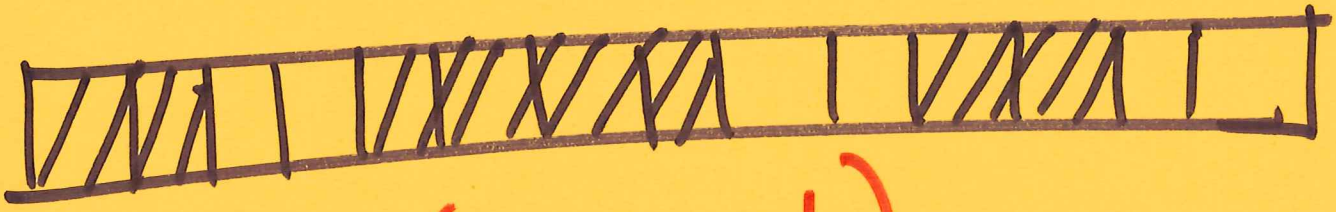
$\Rightarrow$ throughput of 2000 op/s.

2. Max. memory utilization.

Fragmentation

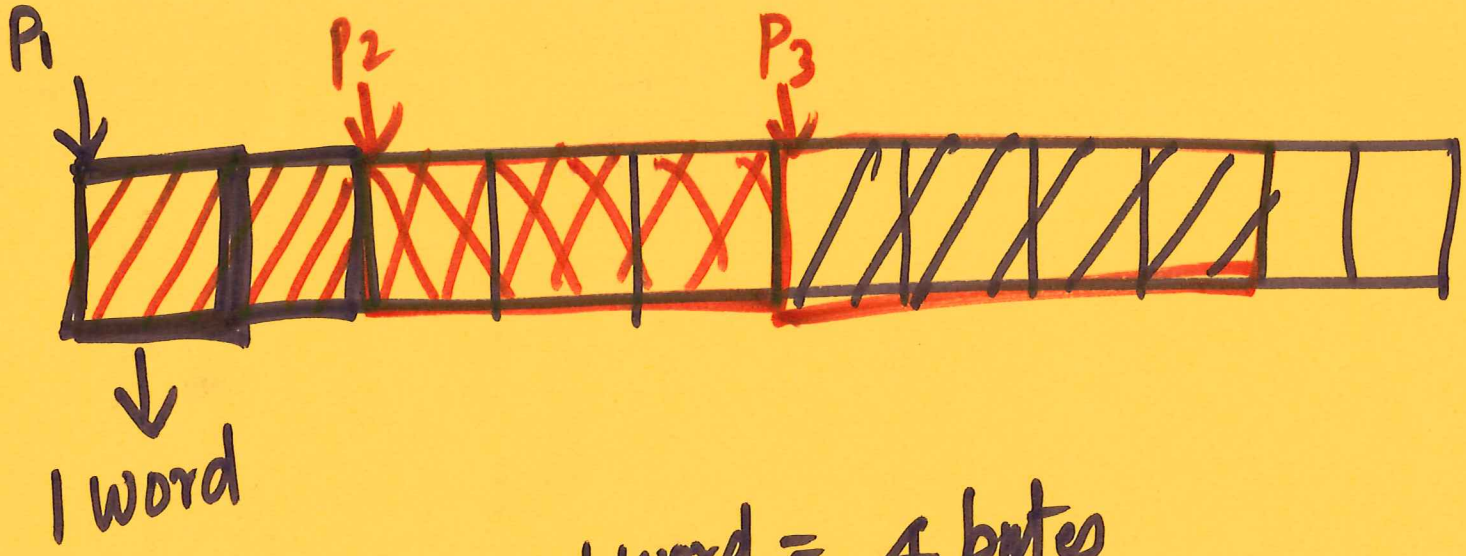


1. internal frag



malloc (6 * int)

2. external frag



1 word = 4 bytes

```
p1 = malloc(2 * sizeof(int));
p2 = malloc(3 * sizeof(int));
p3 = malloc(4 * 4);
free(p2);
```

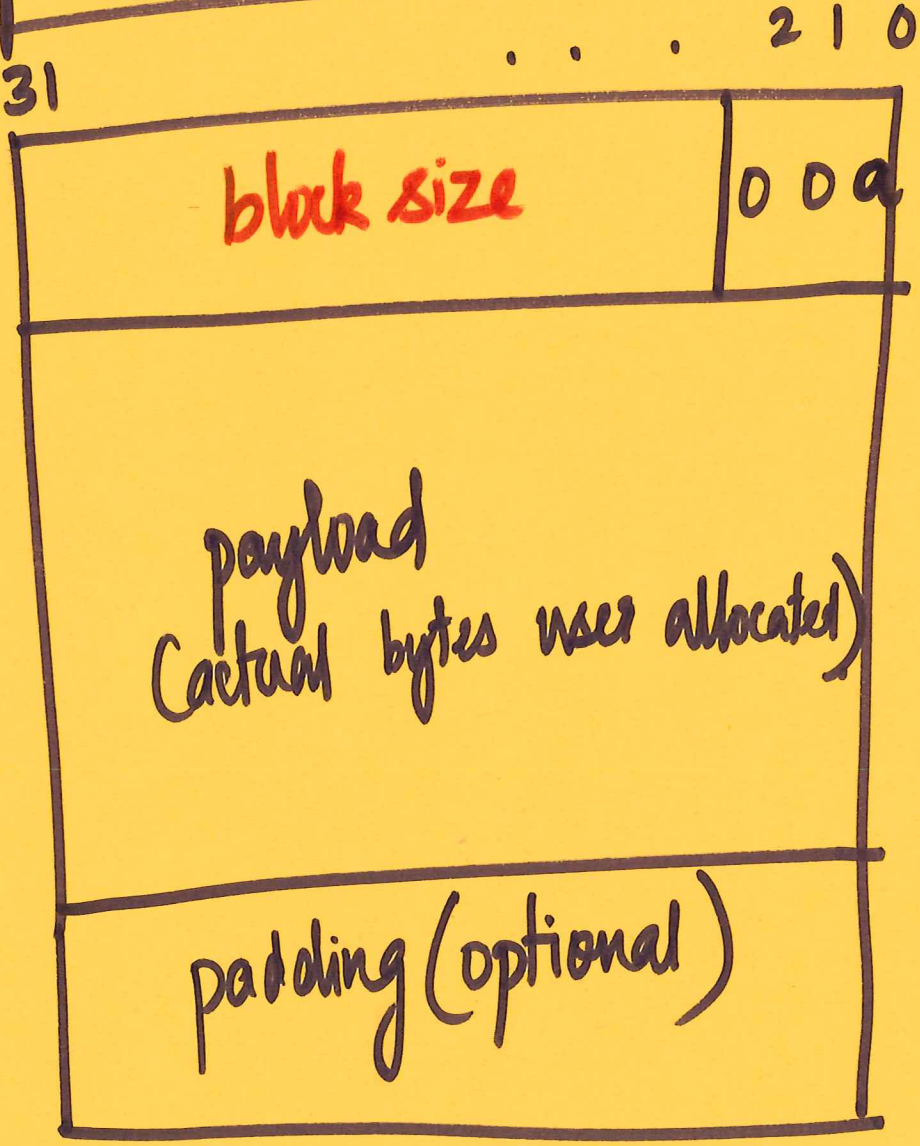

```
P4 = malloc(2 ints);
```

Free block organization

implicit free lists

double word aligned.

Heap block structure



a = 1 ⇒ allocated

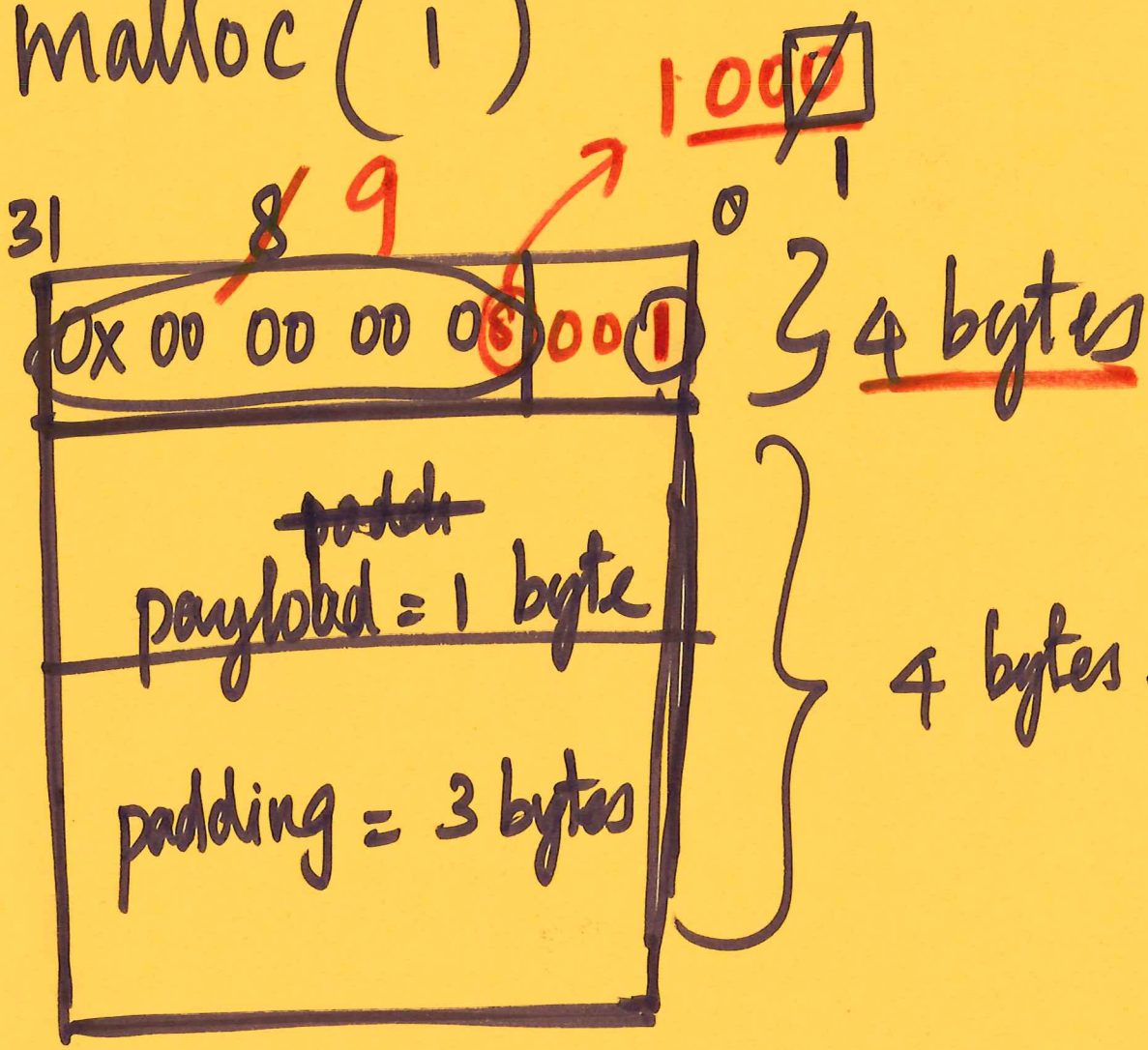
a = 0 ⇒ free.

0x F1 23 45 7**8**

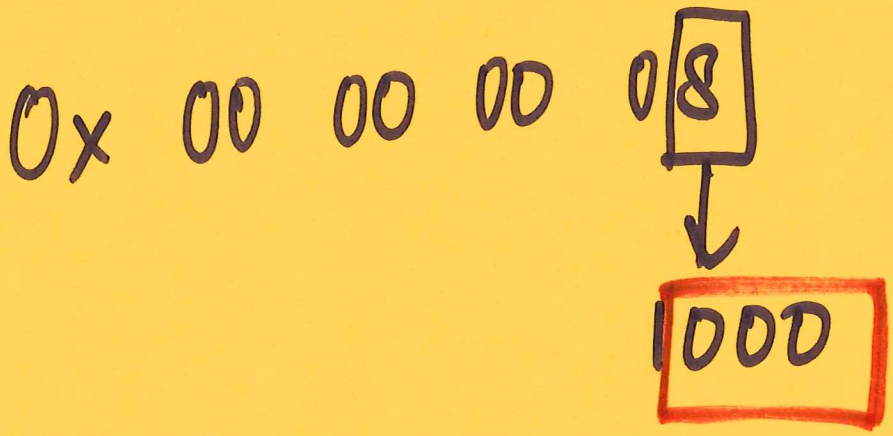
0x F1 23 45 7**0**

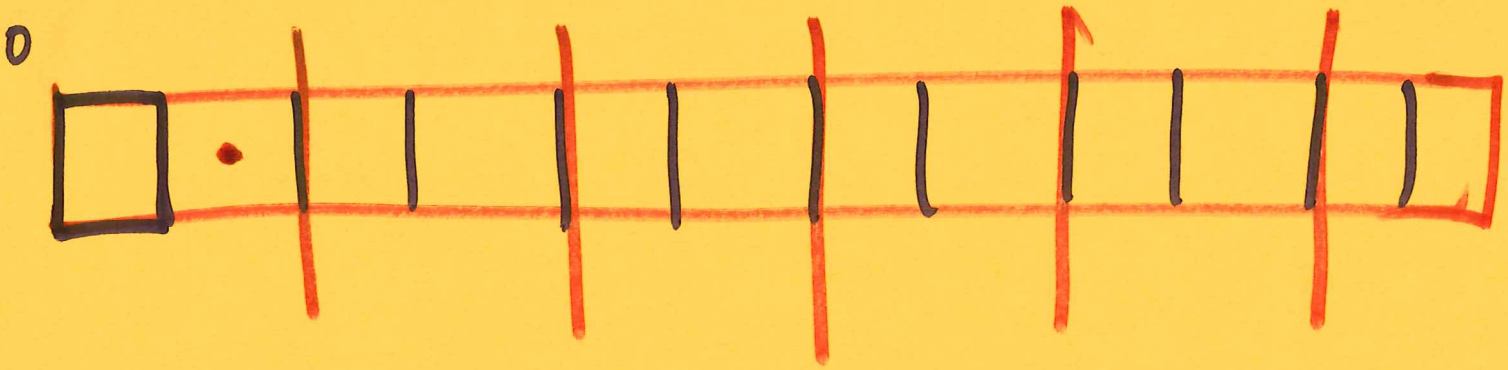
blocksize = header + payload + padding

malloc(1)



4 + 1 + 3 = 8 bytes





malloc(2)