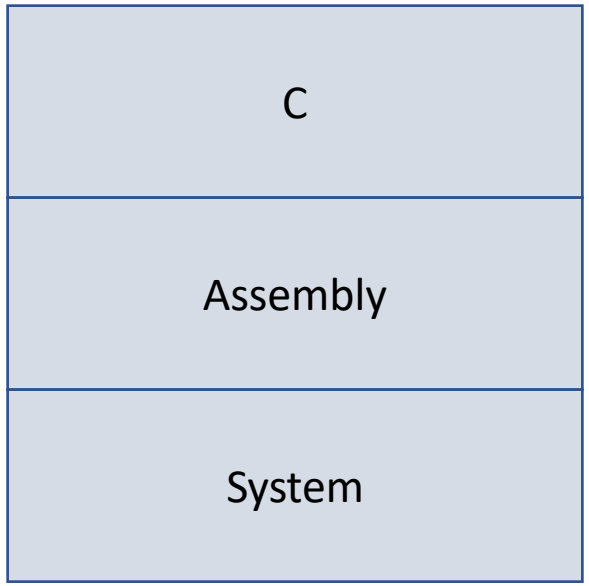
CS 354
Machine Organization and Programming
Lecture 22

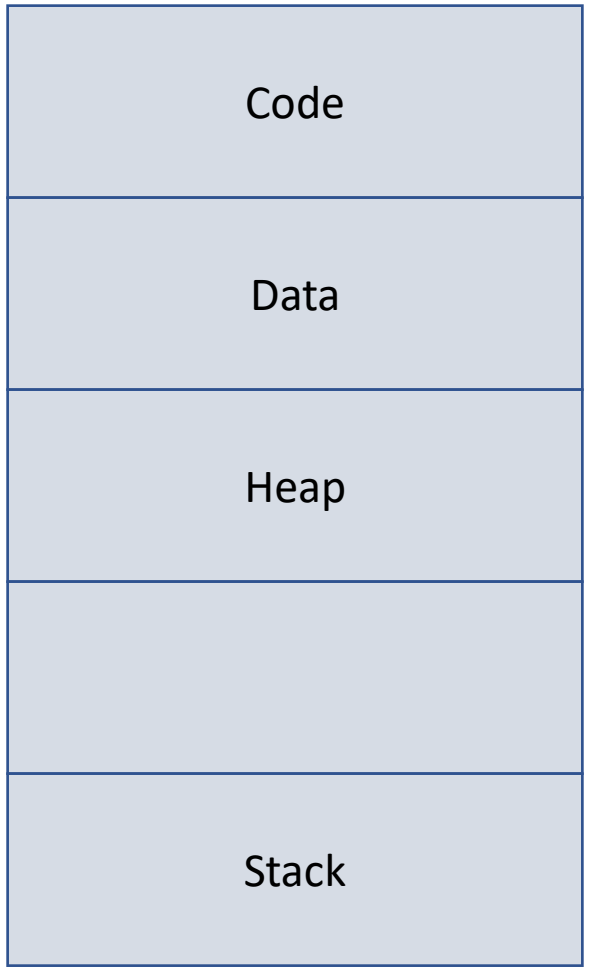
Michael Doescher
Summer 2020

Dynamic Memory Allocation

Course Structure



Address Space



Memory Allocators

Implicit
garbage collection
java / python

Explicit
malloc / free
new delete

malloc

malloc
void * malloc(size_t num_bytes);
int *p = malloc(4*sizeof(int));

free
void free(void *p){...}
free (p);

malloc

malloc
void * malloc(size_t num_bytes);
int *p = malloc(4*sizeof(int));

legacy code
char * malloc(size_t num_bytes);
int * p = (int *)malloc(4*sizeof(int));

free
void free(void *p){...}
free (p);

malloc

malloc

```
void * malloc(size_t num_bytes);
int *p = malloc(4*sizeof(int));
```

legacy code

```
char * malloc(size_t num_bytes);
int * p = (int *)malloc(4*sizeof(int));
```

free

```
void free(void *p){...}
free (p);
```

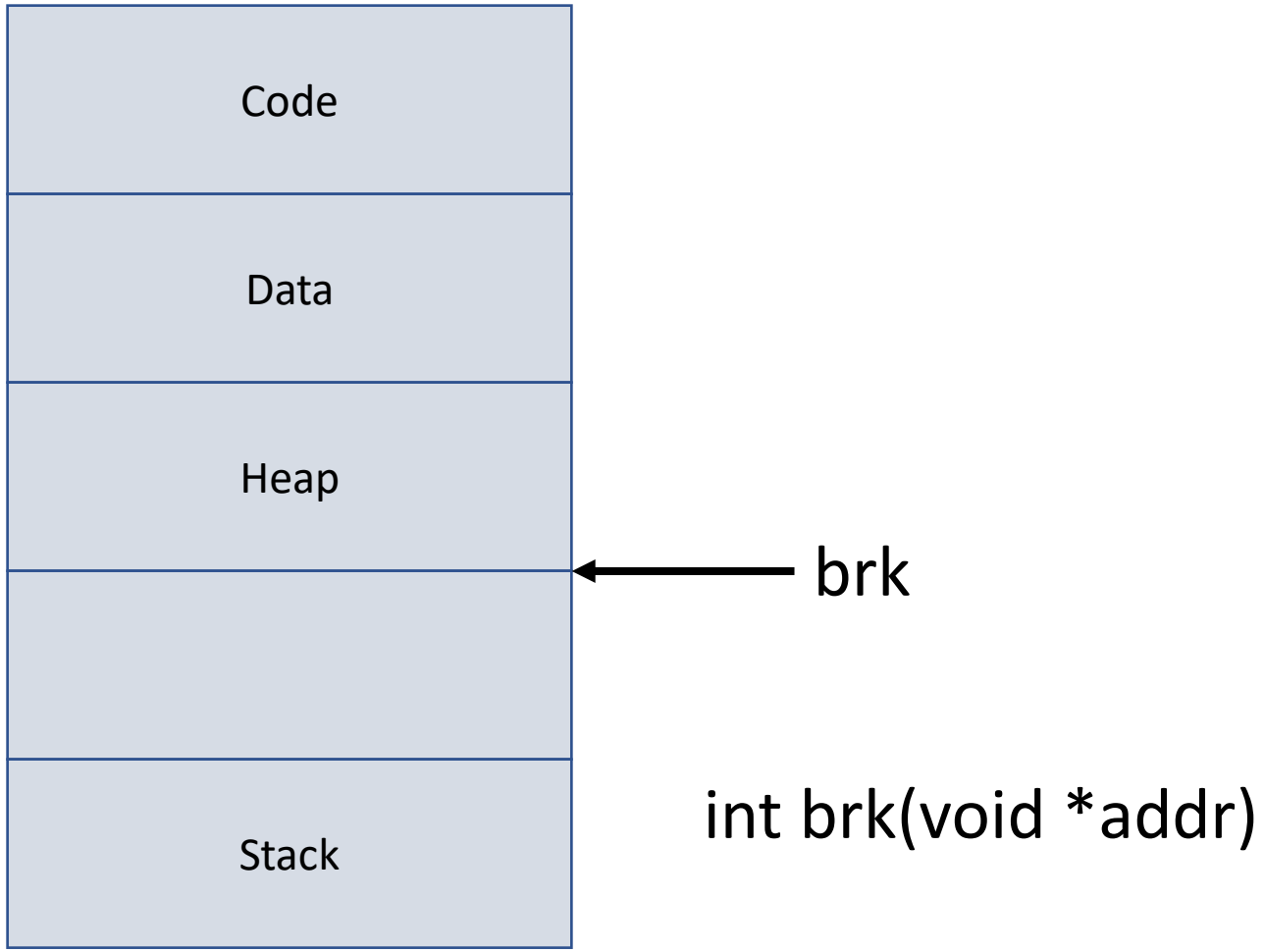
How do we know how many bytes to deallocate?

How big was the allocated block of memory

realloc

calloc

History



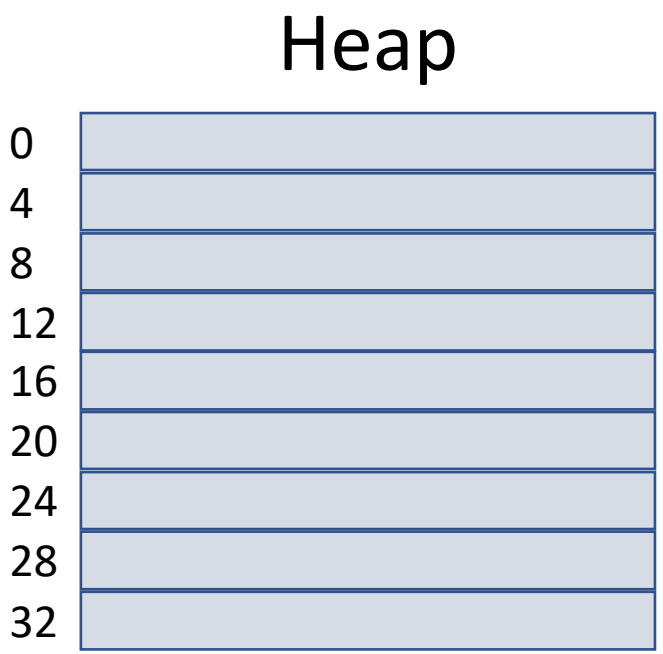
History

brk

void * sbrk(int increment)

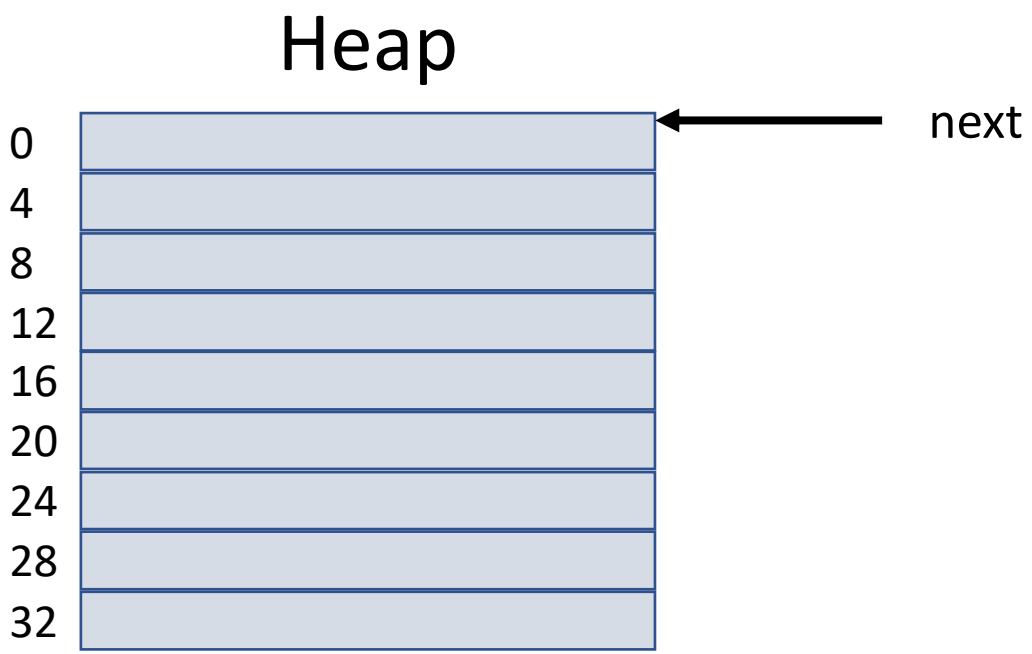
Super Simple Memory Allocator: Alignment Requirement

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



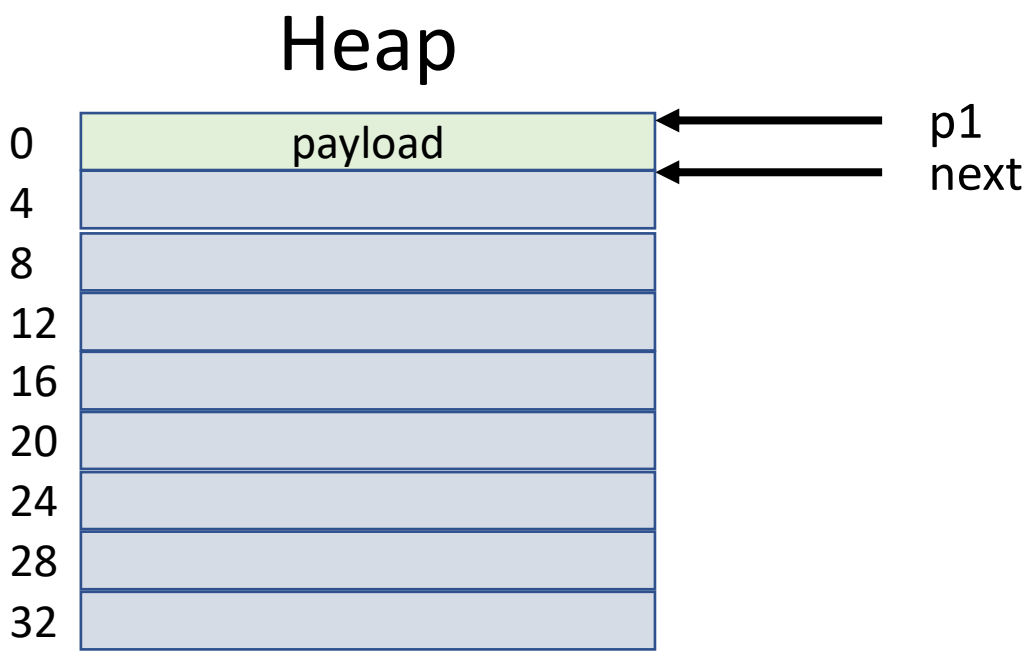
Super Simple Memory Allocator: Pointer to Next Free Byte

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



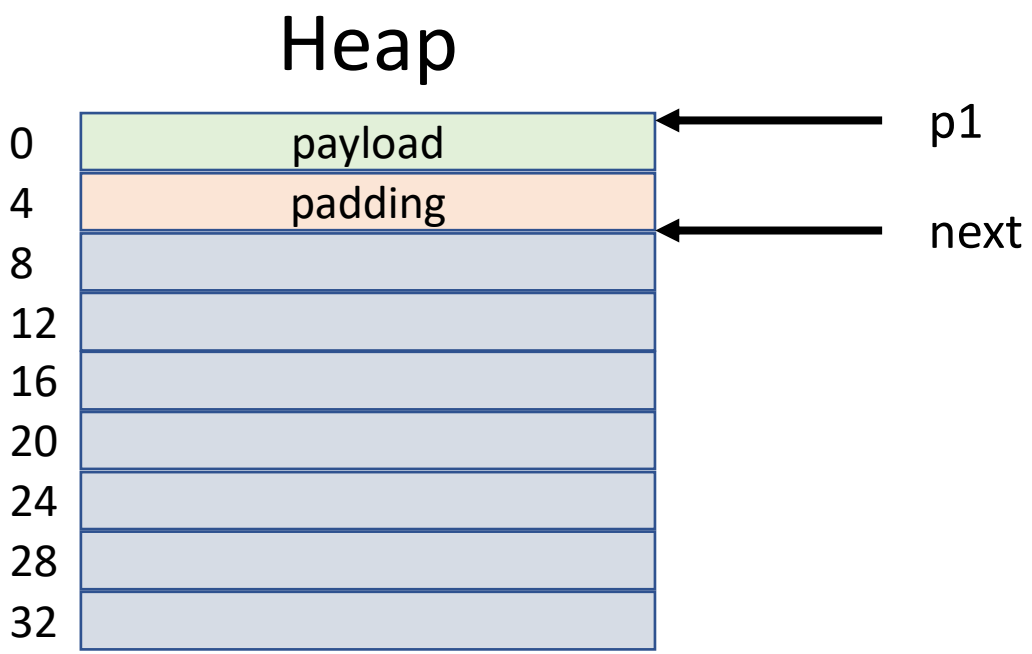
Super Simple Memory Allocator: Alignment Requirement

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



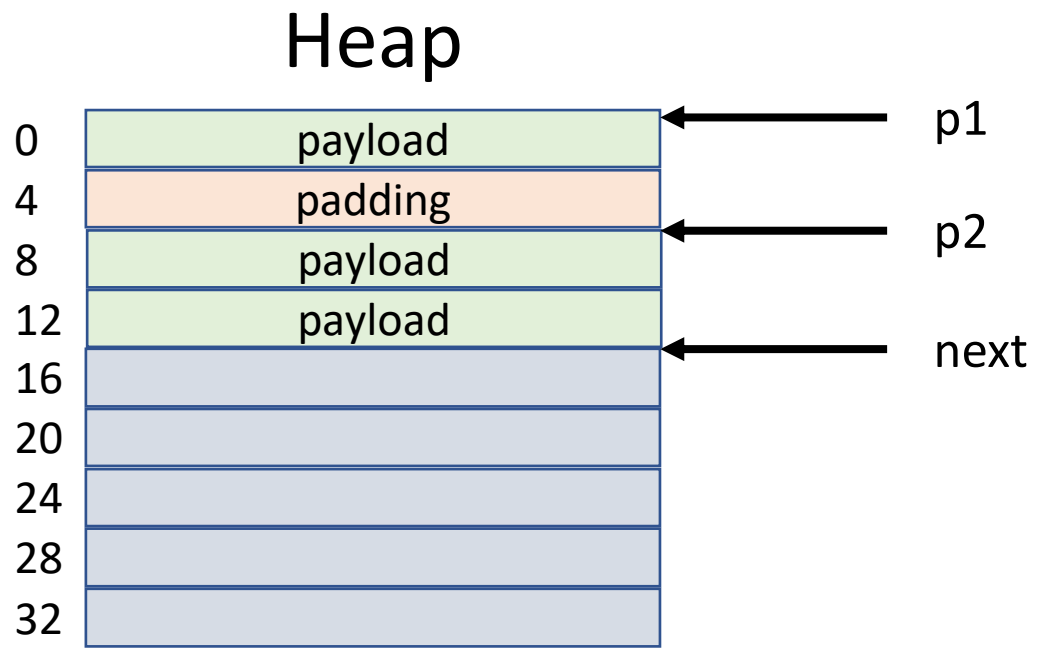
Super Simple Memory Allocator: Alignment Requirement

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



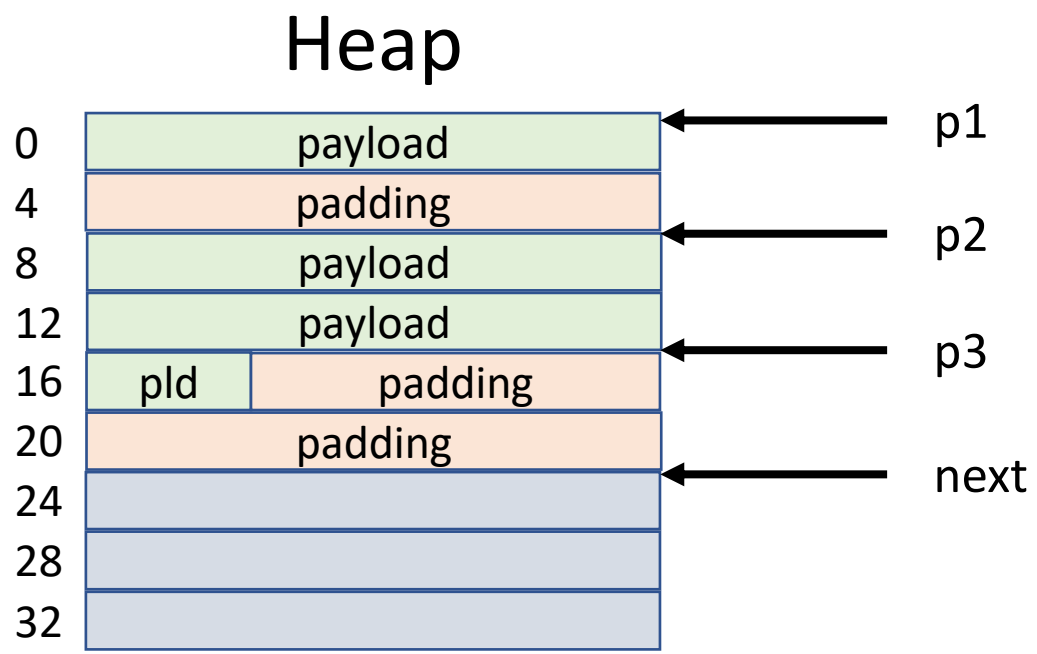
Super Simple Memory Allocator: Alignment Requirement

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



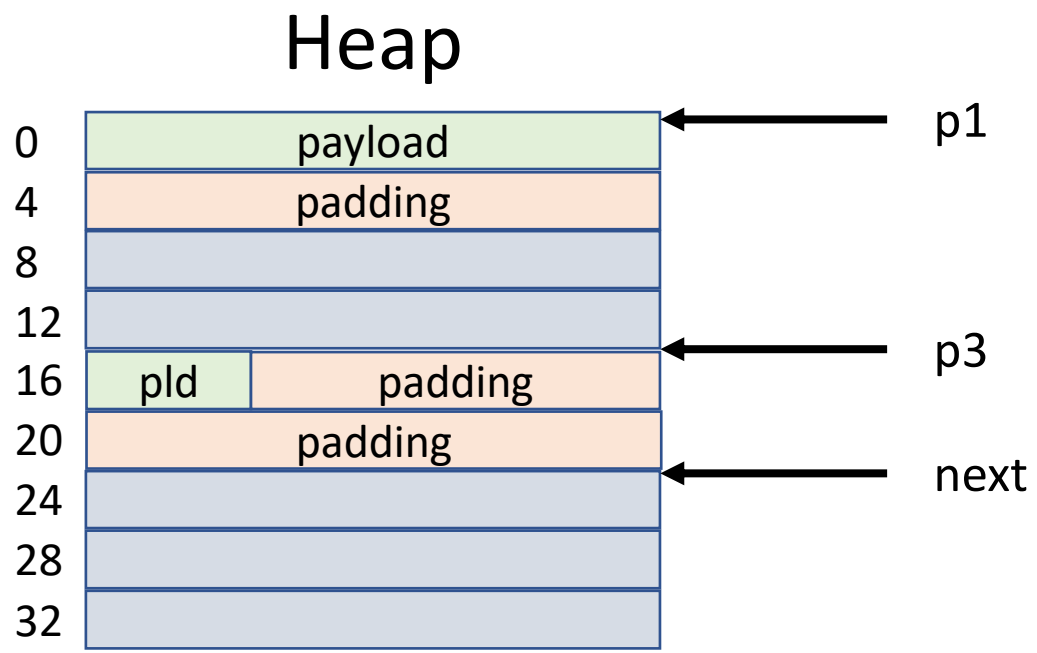
Super Simple Memory Allocator: Alignment Requirement

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



Super Simple Memory Allocator: Alignment Requirement

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



Super Simple Memory Allocator: Alignment Requirement

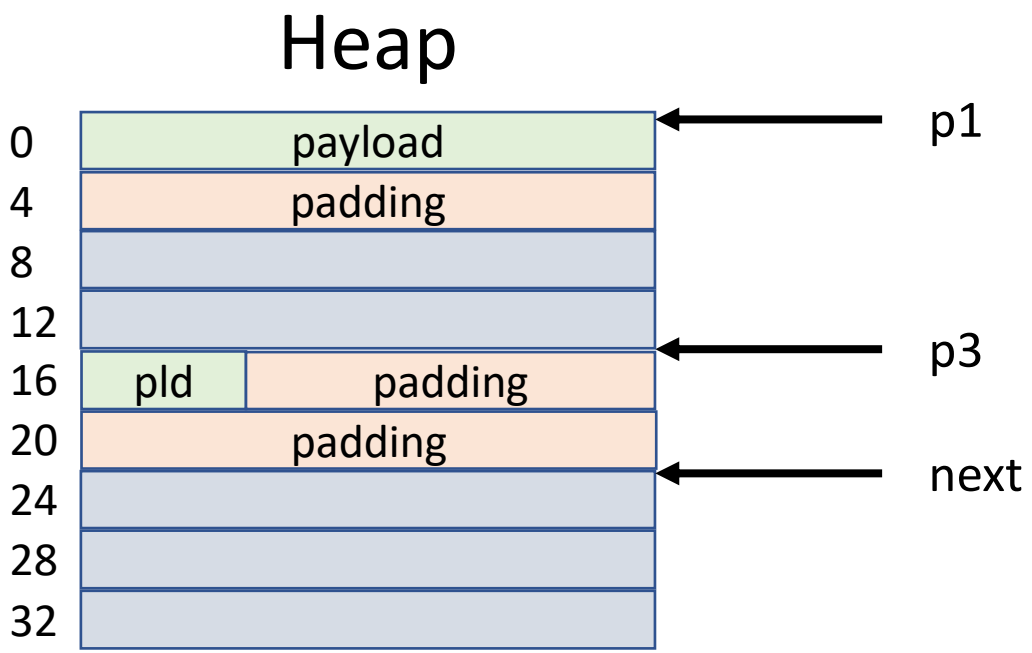
```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```

Problems

How do we free?

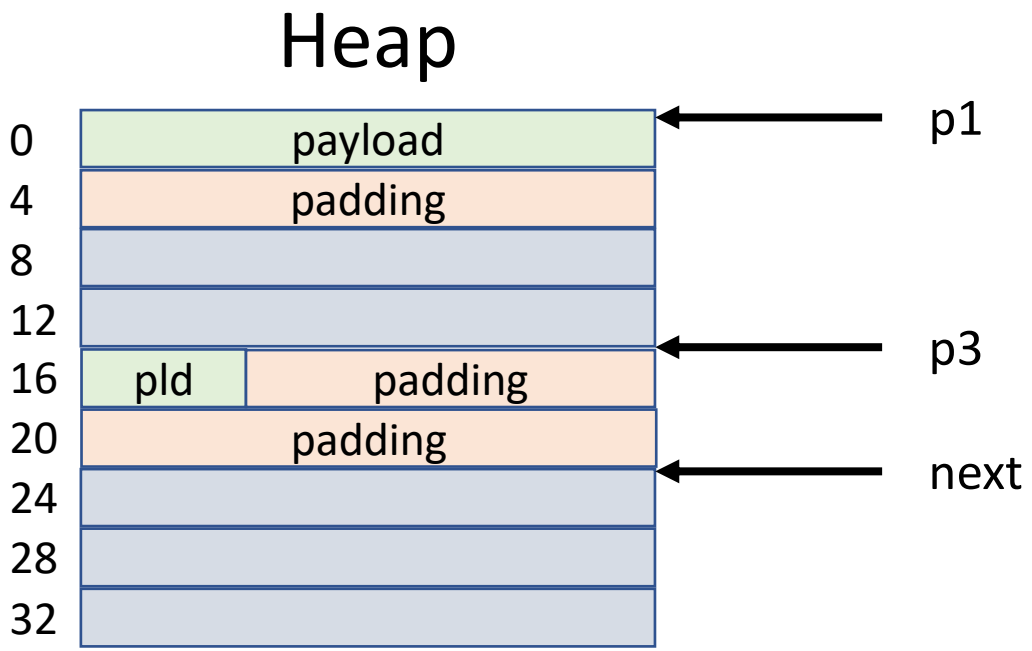
Leaves holes?

What happens when we get to the end



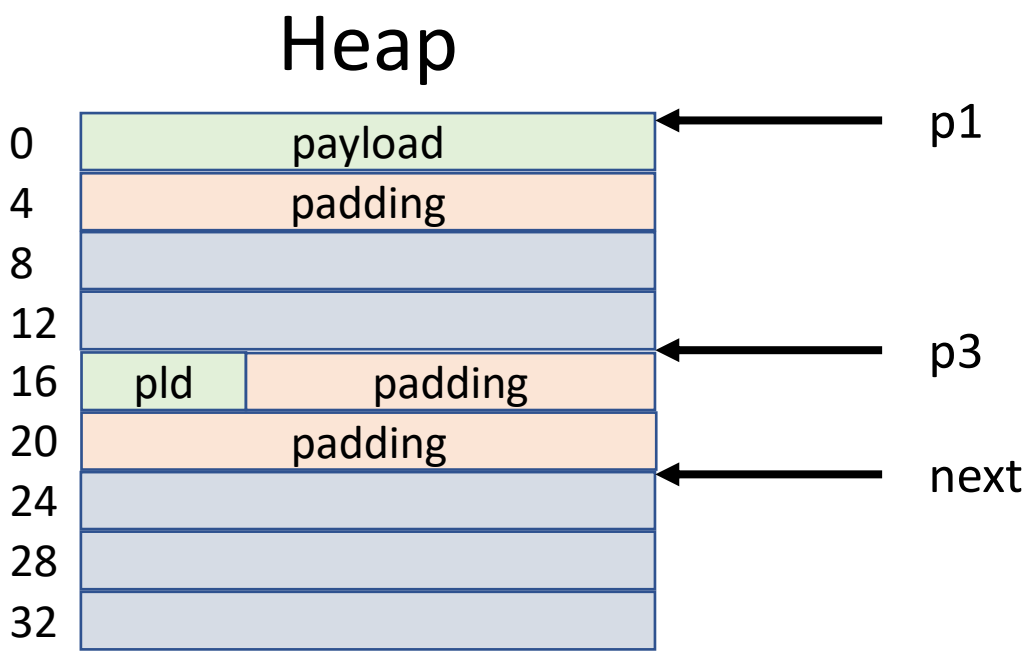
Memory Allocator Requirements

1. Keep track of block sizes for freeing
2. Arbitrary Request Sequences
3. Immediate Response
    1. no buffering
    2. or reordering
4. Use only the heap
5. Block Alignment
6. No modification of allocated blocks



Memory Allocator Requirements

1. Keep track of block sizes for freeing
    1. metadata
2. Arbitrary Request Sequences
3. Immediate Response
    1. no buffering
    2. or reordering
4. Use only the heap
5. Block Alignment
6. No modification of allocated blocks

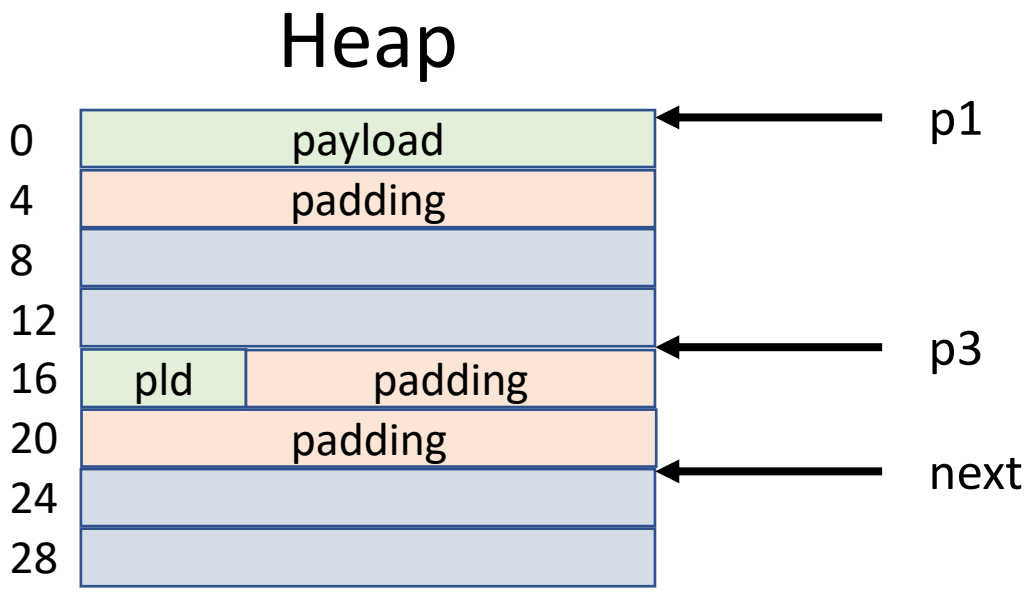


Goals

1. Maximize throughput – requests per second
2. Maximize Memory Utilization
    1. minimize padding
    2. minimize holes (fragmentation)
    3. minimize metadata

Memory Allocator Requirements

1. Keep track of block sizes for freeing
    1. metadata
2. Arbitrary Request Sequences
3. Immediate Response
    1. no buffering
    2. or reordering
4. Use only the heap
5. Block Alignment
6. No modification of allocated blocks



Goals

1. Maximize throughput – requests per second
2. Maximize Memory Utilization
    1. minimize padding
    2. minimize holes (fragmentation)
    3. minimize metadata

Fragmentation

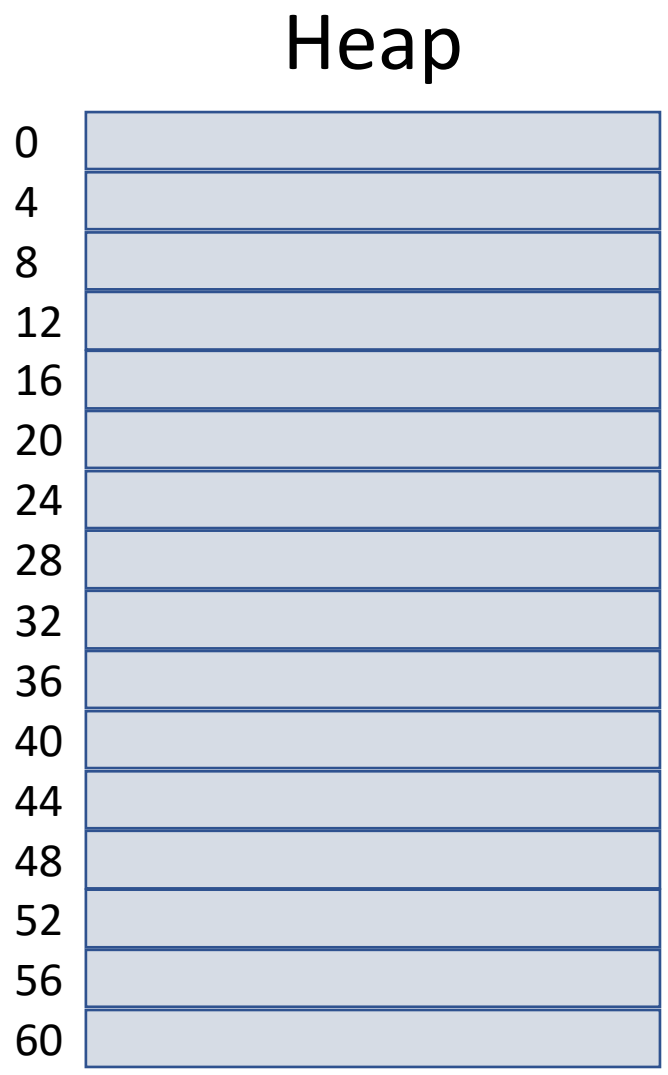
Internal – allocated block is larger than payload

1. allocator may have min block size rules
2. alignment requirements

External – Enough free bytes, but not contiguous block is large enough for a request.
malloc(12)

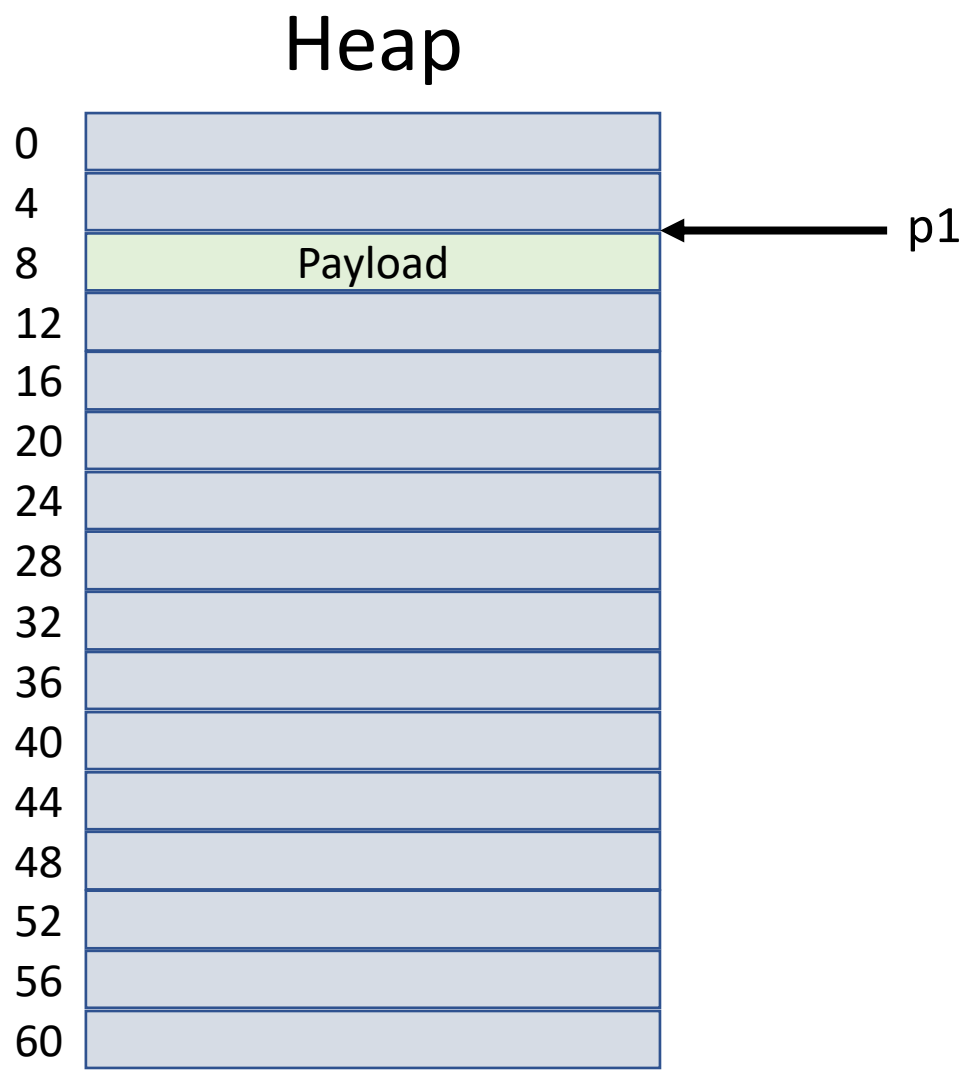
Implicit Free List

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



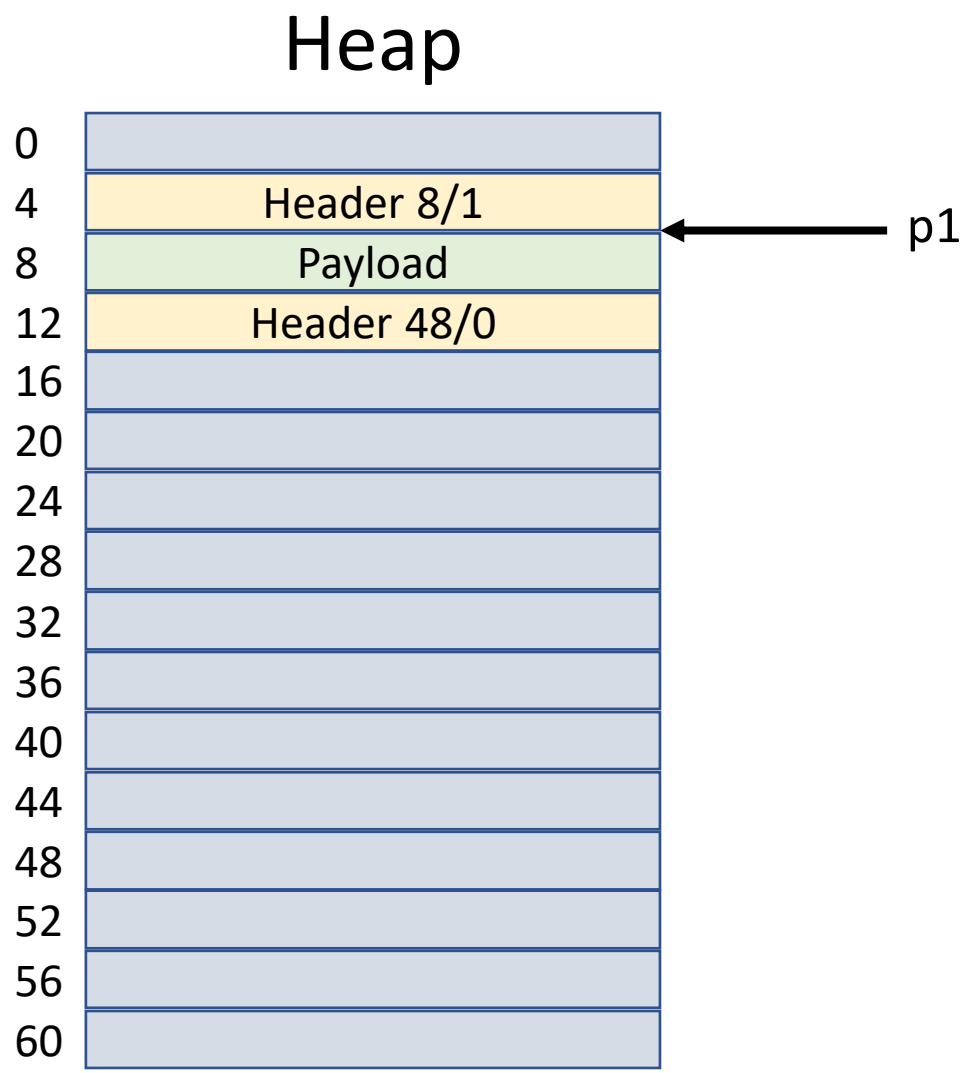
Implicit Free List

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



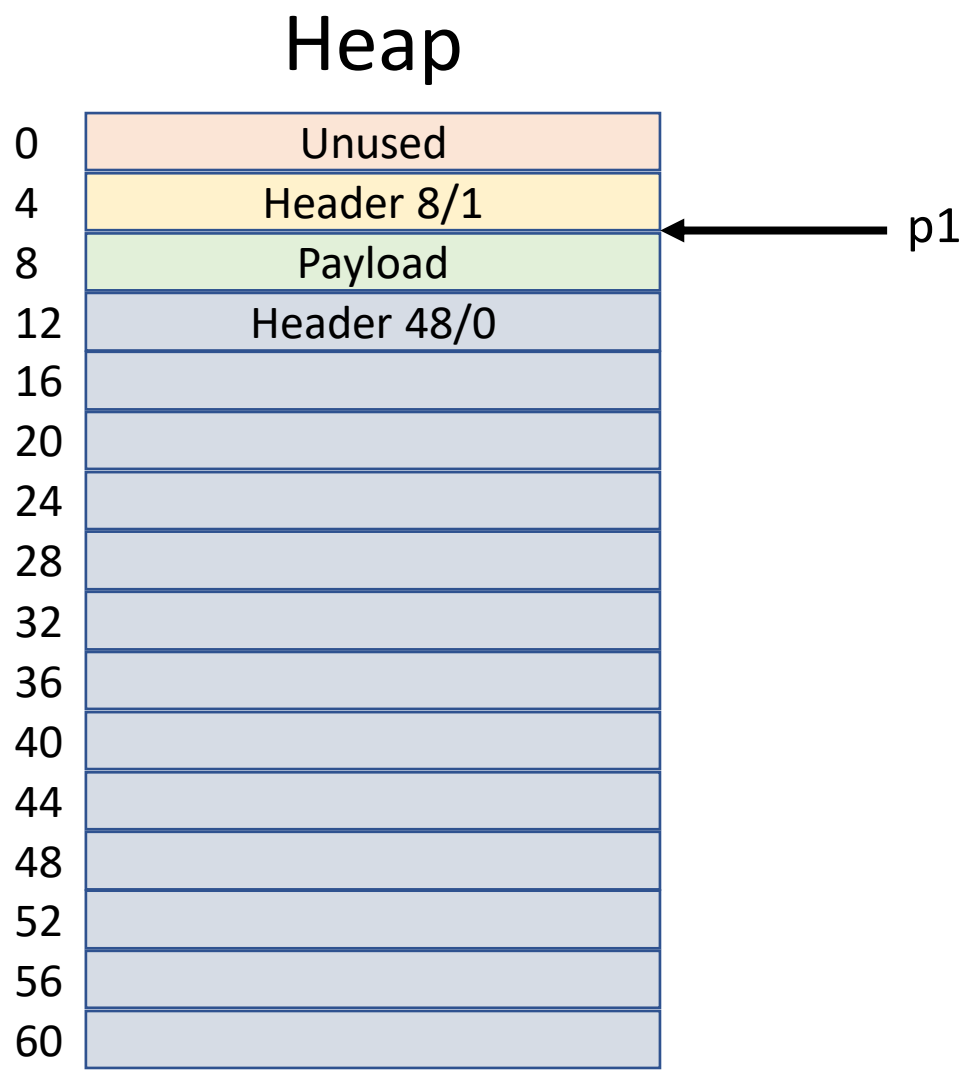
Implicit Free List

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



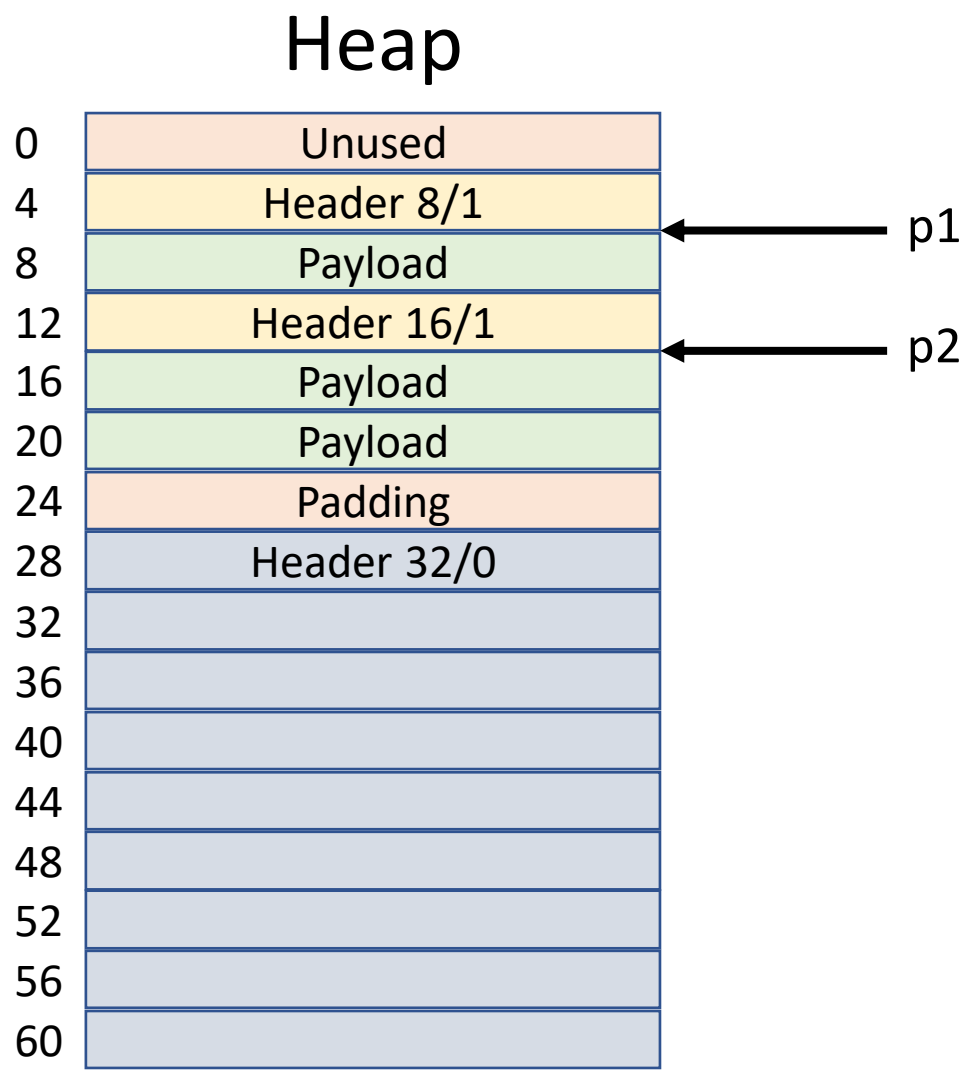
Implicit Free List

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



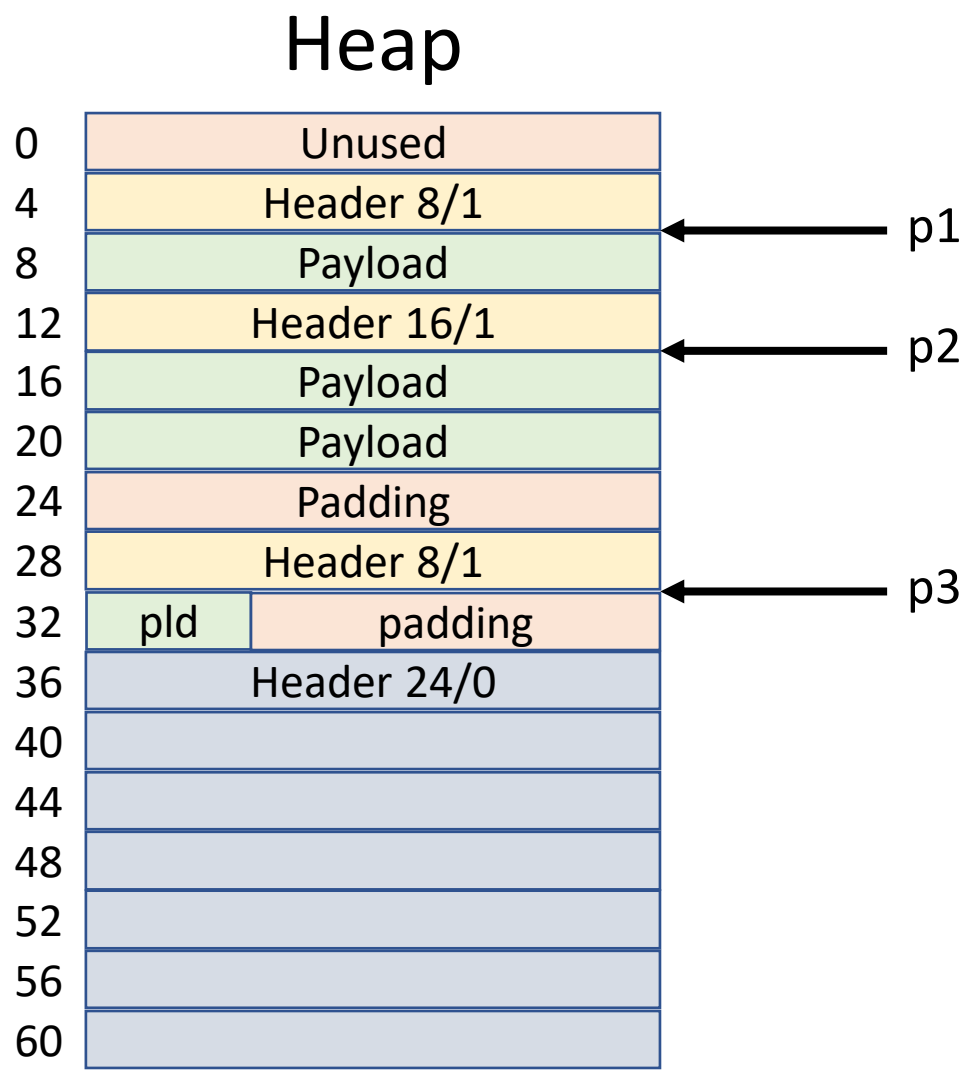
Implicit Free List

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



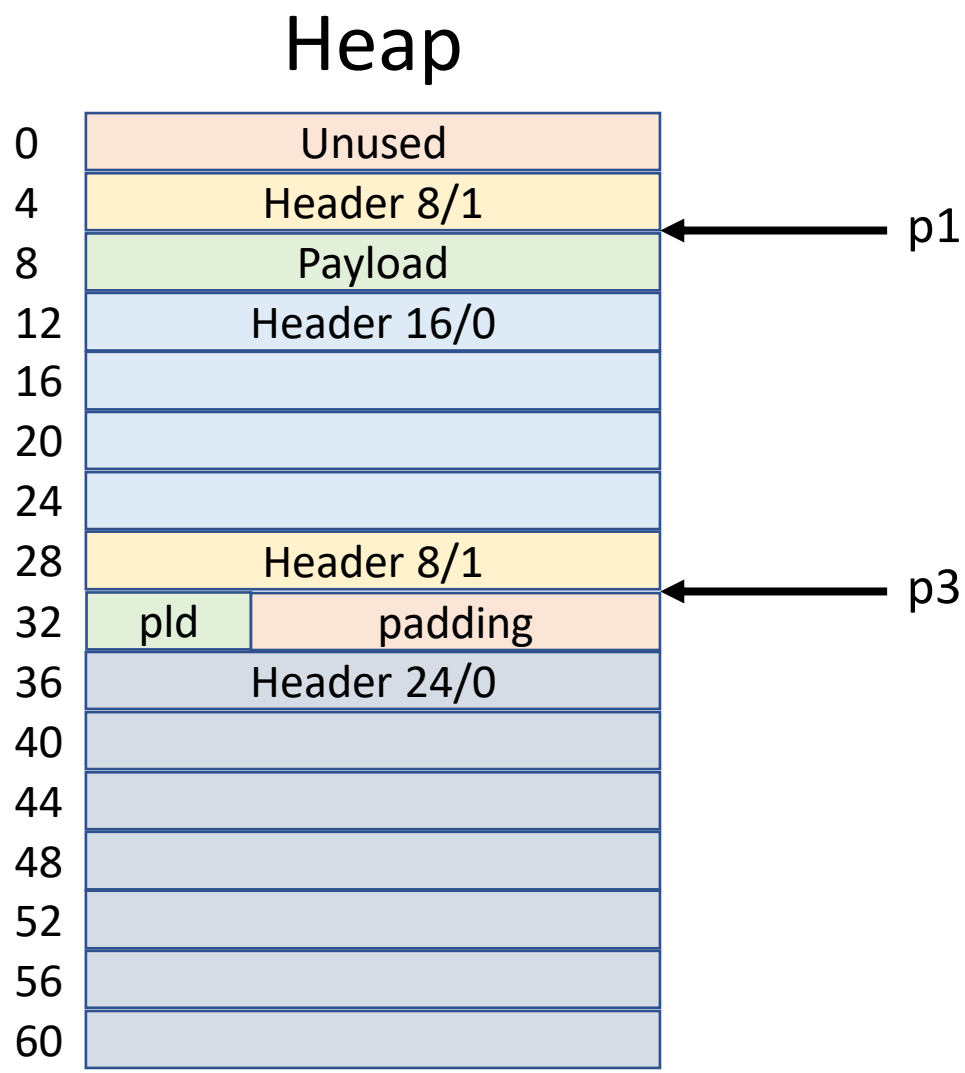
Implicit Free List

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



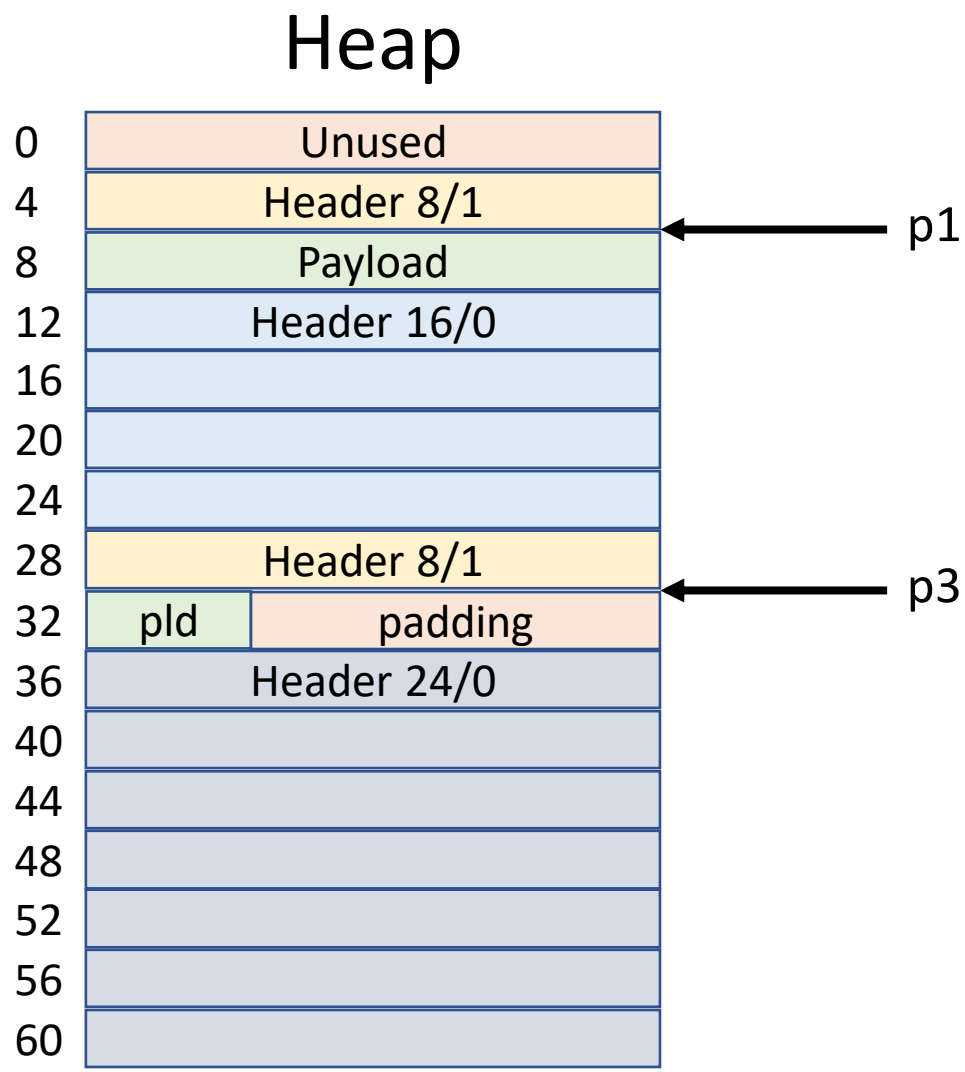
Implicit Free List

```
int *p1 = malloc(4);
int *p2 = malloc(8);
int *p3 = malloc(1);

free (p2)
```



Implicit Free List

How to pack the size of the allocated block and the allocated / free flag into 4 bytes?



Implicit Free List

How to pack the size of the allocated block and the allocated / free flag into 4 bytes?

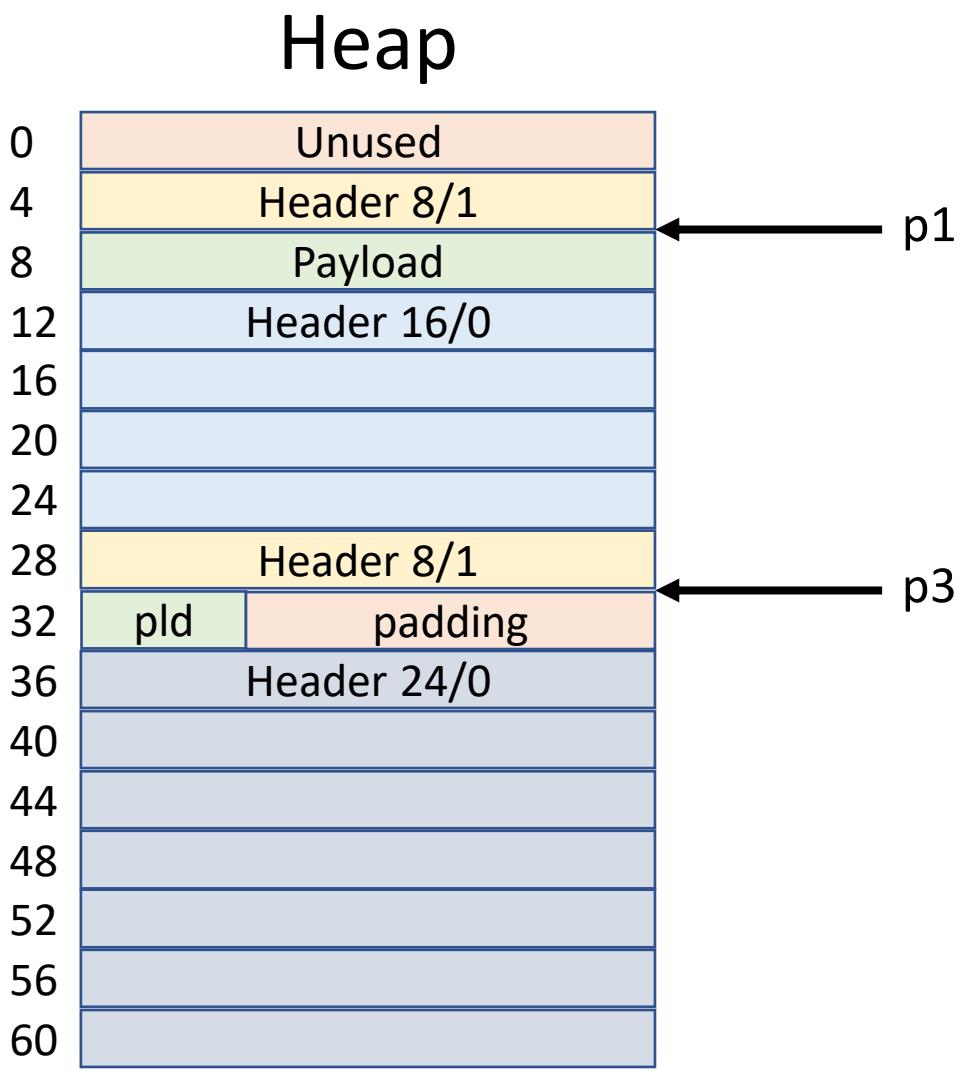
The size must be a multiple of 8

0x 0008 : 8

0x 0010 : 16

0x 0018 : 24

0x 0020 : 32

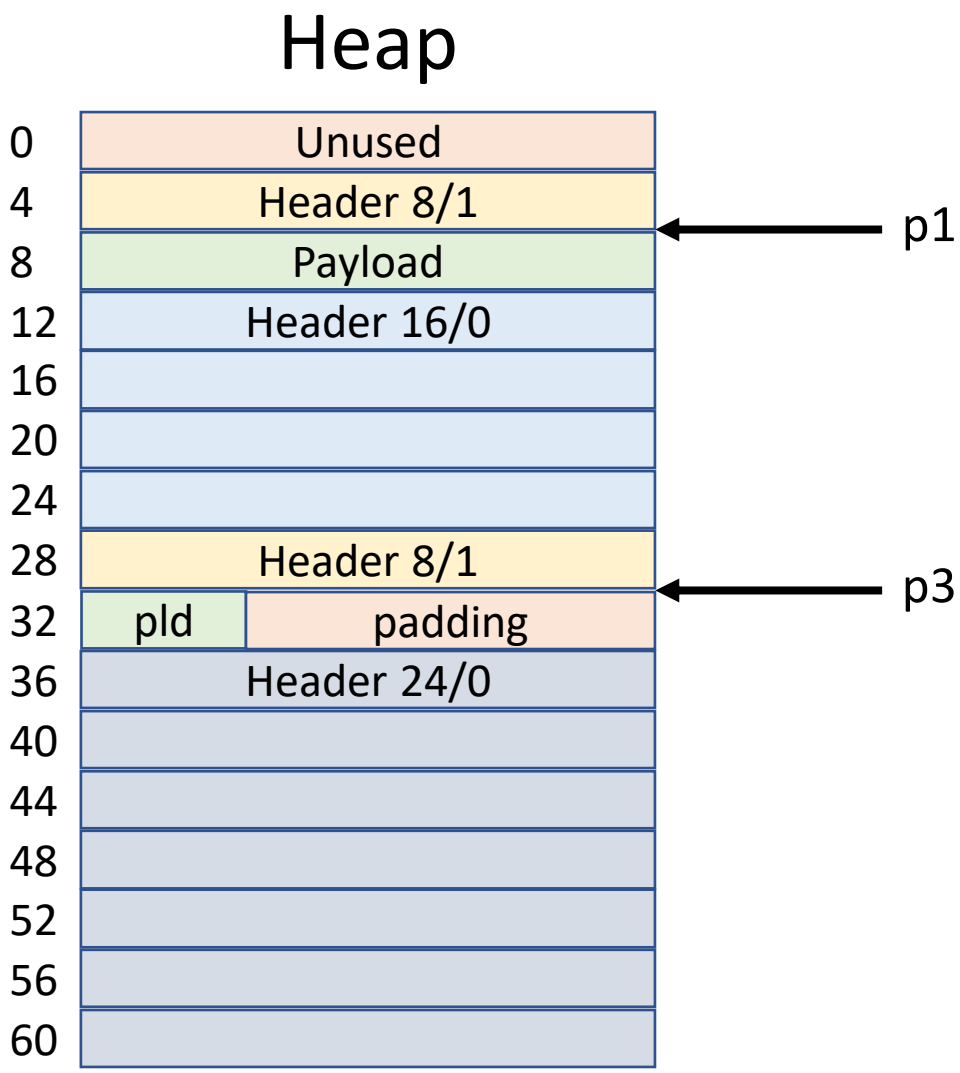


Implicit Free List

How to pack the size of the allocated block and the allocated / free flag into 4 bytes?

The size must be a multiple of 8

0x 0008 : 8 : 0000 1000
0x 0010 : 16 : 0001 0000
0x 0018 : 24 : 0001 1000
0x 0020 : 32 : 0010 0000

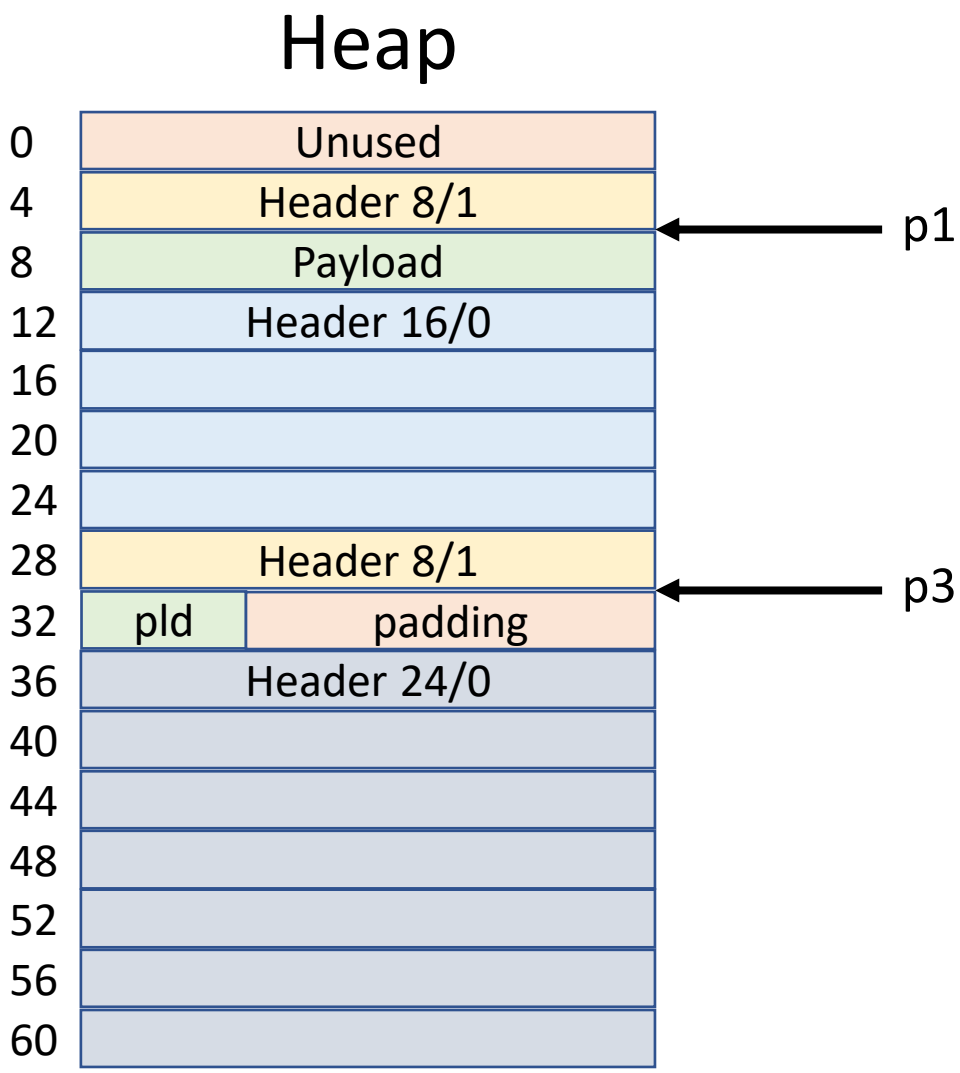


Implicit Free List

How to pack the size of the allocated block and the allocated / free flag into 4 bytes?

The size must be a multiple of 8

0x 0008 : 8 : 0000 1000
0x 0010 : 16 : 0001 0000
0x 0018 : 24 : 0001 1000
0x 0020 : 32 : 0010 0000



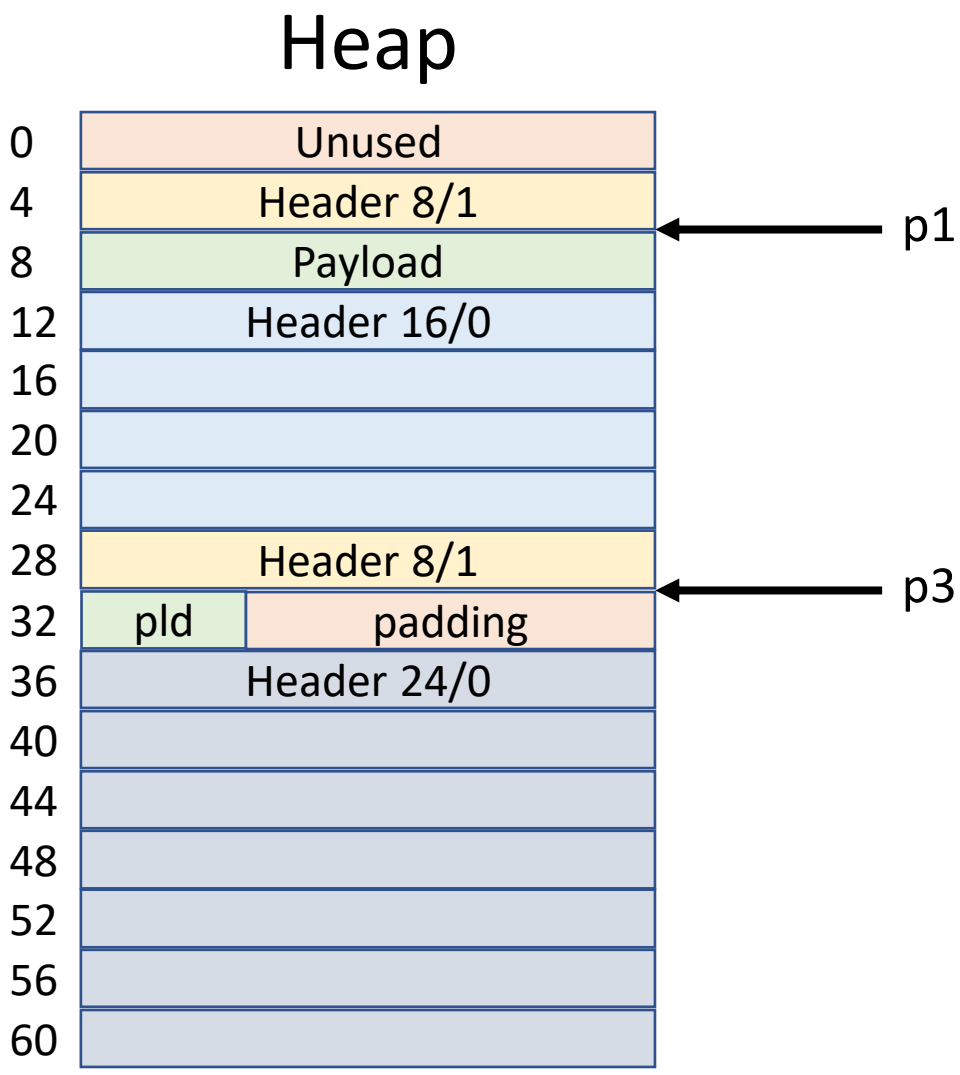
Implicit Free List

How to pack the size of the allocated block and the allocated / free flag into 4 bytes?

The size must be a multiple of 8

0x 0008 : 8 : 0000 1000
0x 0010 : 16 : 0001 0000
0x 0018 : 24 : 0001 1000
0x 0020 : 32 : 0010 0000

Just use the b0 bit for the flag



Implicit Free List

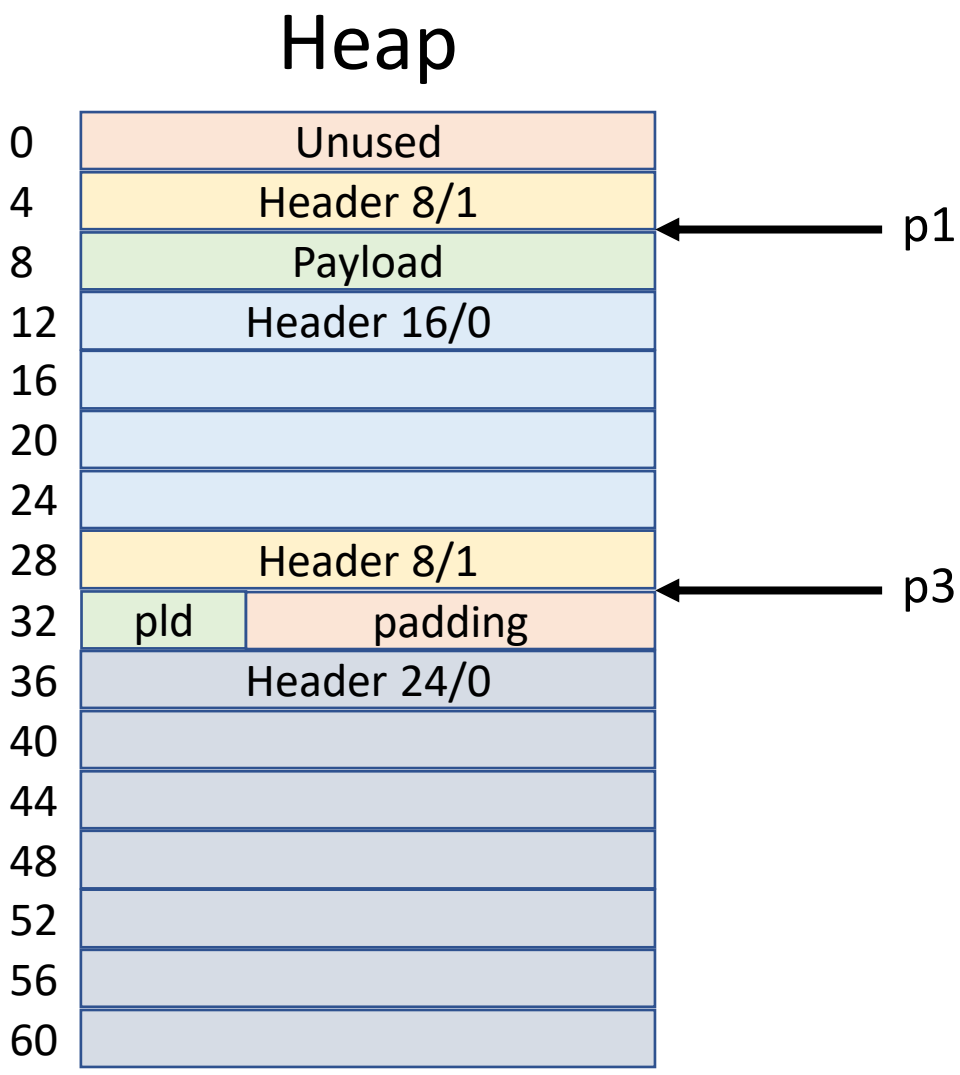
How to pack the size of the allocated block and the allocated / free flag into 4 bytes?

The size must be a multiple of 8

0x 0008 : 8 : 0000 1000
0x 0010 : 16 : 0001 0000
0x 0018 : 24 : 0001 1000
0x 0020 : 32 : 0010 0000

Just use the b0 bit for the flag

8/0 -> 0000 1000 = 8
8/1 -> 0000 1001 = 9
16/1 -> 0001 0001 = 17



Implicit Free List

What is the largest block of memory we can allocate?

Binary:
11111111 11111111 11111111 11111000
Hex: 0xFF FF FF F8
Decimal: 4,294,967,288 bytes