Review
Virtual address space
malloc, free, calloc, realloc.

```
int brk (void *addr);
void * sbrk (int incr);  // most used.
```
On error, sbxrk returns

```
( void *, -1 )
```

![Allocators diagram](image)

- `heap` → collection of various-sized blocks
- `block` → contiguous chunk of virtual mem.

- `allocated`
- `free`
Allocations

explicit
C-malloc, free

implicit
Garbage collection "new"
free - automatic.
Alignment requirements

eg. double-word aligned

\[ \text{pl} = \& \text{malloc} \left( 1 \times \text{sizeof (int)} \right) \]

In P5, align req. is word size.

\[
\begin{array}{cccccccc}
\text{word aligned}.
\\
\text{malloc} (1) & \rightarrow & \text{give you 4 bytes.}
\end{array}
\]

payload: 1 - used

padding: 3 - wasted
Implementation issues:

1. Free block organization.
2. Placement
3. Splitting into free blocks.

Requirements

1. Handle arbitrary requests.
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.

   \[ \Rightarrow \text{throughput of } 2000 \text{ op/s} \]

2. Max. memory utilization.
Fragmentation

1. internal frag

2. external frag

malloc(6*int)
1 word = 4 bytes

\[ P_1 = \text{malloc}(2 \times \text{sizeof(int)}) \];

\[ P_2 = \text{malloc}(3 \times \text{sizeof(int)}) \];

\[ P_3 = \text{malloc}(4 \times n) \];

\[ \text{free}(P_2) \];

\[ P_4 = \text{malloc}(2 \text{ ints}) \];
Free block organization

**Implicit Free Lists**

double word aligned.

Heap block structure

```
<table>
<thead>
<tr>
<th>31</th>
<th>...</th>
<th>210</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00a</td>
<td></td>
</tr>
</tbody>
</table>
```

- block size
- payload (actual bytes user allocated)
- padding (optional)

```
OX F1234578
OX F1234570
```

```
  10000
  00000
```

blocksize = header + payload + padding

a = 1 -> allocated
a = 0 -> free.
malloc (1)

34 bytes

4 + 1 + 3 = 8 bytes

0x 00 00 00 00 08

1000
malloc (2)

0x00 00 00 08 9

2

2

3 4 bytes