Assume the following,

- Initially, the heap has a capacity of 16 words.

- 1 word = 4 bytes.
- Heap is double-word aligned
(a) \( p_1 = \text{malloc}(4 \times \text{sizeof(int)}) \)

(b) \( p_2 = \text{malloc}(5 \times \text{sizeof(int)}) \)

(c) \( p_3 = \text{malloc}(6 \times \text{sizeof(int)}) \)

(d) \text{free}(p_2) \)

(e) \( p_4 = \text{malloc}(2 \times \text{sizeof(int)}) \)
Last Class

1. Virtual Memory / Addressing

Memory Protection

More memory that what is physically available

2. Dynamic Memory Allocation

- Heap (Text, Data, Stack)
- malloc & free

Explicit

Implicit
return null  (if there is not enough space)

malloc \rightarrow we do not know what values will be there in the newly allocated block.

calloc

(\text{void \ *} \text{calloc} \ (\text{size-t num, size-t size}));

returns a pointer to zero-initialized memory block \text{ (size = num \ *} \text{ size) \ bytes.}
malloc

What if there is no space on the heap?

malloc → call the `malloc()` system call to get more memory for the heap

→ a request to the OS.
sbak(
)

void * sbak (int in a);

If everything goes right, sbak

⇒ will return the old bak value.

something went wrong

⇒ return -1

printf ⇒ strierror (errno);

sbak would set this depending on what went wrong.
Design Goals (Allocator)

1. Throughput (↑)
   If it does 100 mallocs and 100 frees in a second,
   Throughput = 200 operations per second.

2. Memory Utilization
   VM is limited
   \[ \text{Allocated Memory} \]
   \[ \text{Total Heap Size} \]

Tradeoff → Throughput × Memory Utilization
Fragmentation

Poor heap utilization because of Fragmentation.

Internal Fragmentation

→ When an allocated block is larger than the payload.

When could this happen?

→ Larger block size due to some alignment restriction → (payload).

→ Minimum block size restriction
How much space is wasted by Internal Fragmentation.

\[ \text{SUM (all allocated blocks)} - \text{SUM (all payloads)} \]

External Fragmentation

1. Requesting 3 words.
2. Even though the heap has 4 words of free space, it cannot satisfy the request.
Typically → to avoid external fragmentation.

Small # of (over) Large #
large free blocks of small free blocks.

Design Considerations

1. Free block organization
   - Lists?
   - Pointers?

2. Placement
   - How do we choose a block?
Splitting

After we allocate a free block, what do we do with the rest?

Coalescing

What do we do with the block that was just freed?

part of a

Free
Allocated
Allocated
Free