# Heap assumptions for lecture

- **Memory is word addressed (each word can hold a pointer)**

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro

**Allocated block**
**(4 words)**

**Free**
**(3 w**

**Free word**

**Allocated word**
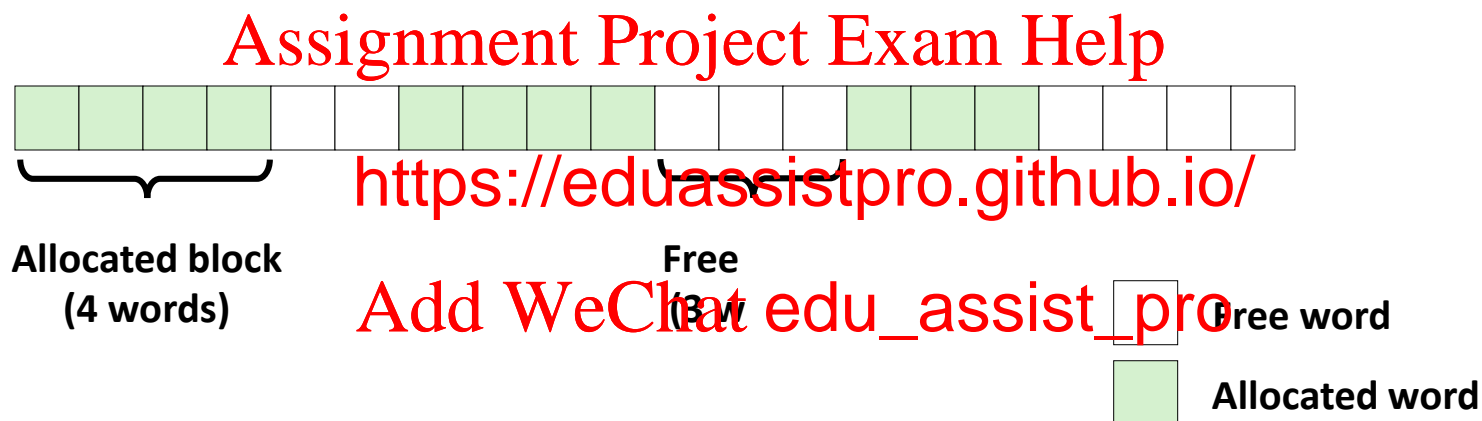
# Allocation Example

`p1 = malloc(4)`
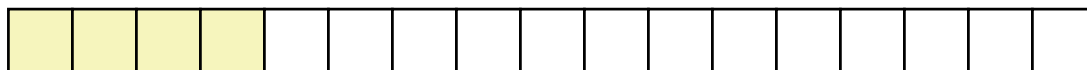
`p2 = malloc(5)`

`p3 = malloc(6)`

`free(p2)`

`p4 = malloc(2)`

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro
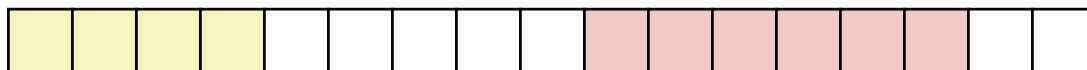
# Constraints

- **Applications**
  - Can issue arbitrary sequence of **malloc** and **free** requests
  - **free** request must be to a **malloc**'d block

- **Allocators**
  - Can't control nu ... s
  - Must respond i ... ts
    - *i.e.*, can't reorder or buffer requ...
  - Must allocate blocks from free memory
    - *i.e.*, can only place allocated blocks in free memory
  - Must align blocks so they satisfy all alignment requirements
    - 8 byte alignment for GNU **malloc** (**libc malloc**) on Linux boxes
  - Can manipulate and modify only free memory
  - Can't move the allocated blocks once they are **malloc**'d
    - *i.e.*, compaction is not allowed

# Performance Goal: Throughput

- **Given some sequence of `malloc` and `free` requests:**
  - $R_0, R_1, ..., R_k, ... , R_{n-1}$

Assignment Project Exam Help

- **Goals: maximize throughput and peak memory utilization**
  - These goals are https://eduassistpro.github.io/

    Add WeChat edu_assist_pro

- **Throughput:**
  - Number of completed requests per unit time
  - Example:
    - 5,000 `malloc` calls and 5,000 `free` calls in 10 seconds
    - Throughput is 1,000 operations/second

# Performance Goal: Peak Memory Utilization

- **Given some sequence of `malloc` and `free` requests:**
  - $R_0, R_1, ..., R_k, ... , R_{n-1}$

- *Def: Aggregate payload $P_k$*
  - `malloc(p)` returns a block with a *payload* of *p* bytes
  - After request $R_k$                                          *payload* $P_k$ is the sum of currently allocat https://eduassistpro.github.io/

- *Def: Current heap size $H_k$*
  - Assume $H_k$ is monotonically nondec
    - i.e., heap only grows when allocator uses `sbrk`

- *Def: Peak memory utilization after k requests*
  - $U_k = ( max_{i<k} P_i ) / H_k$