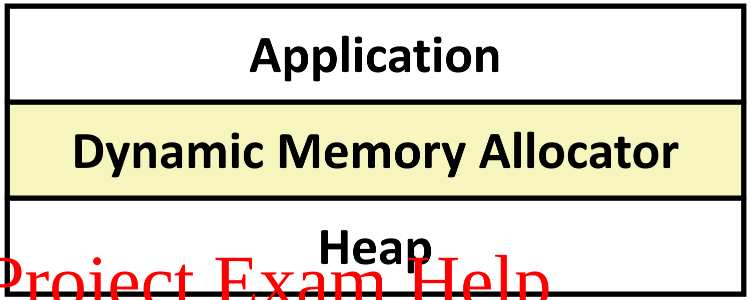
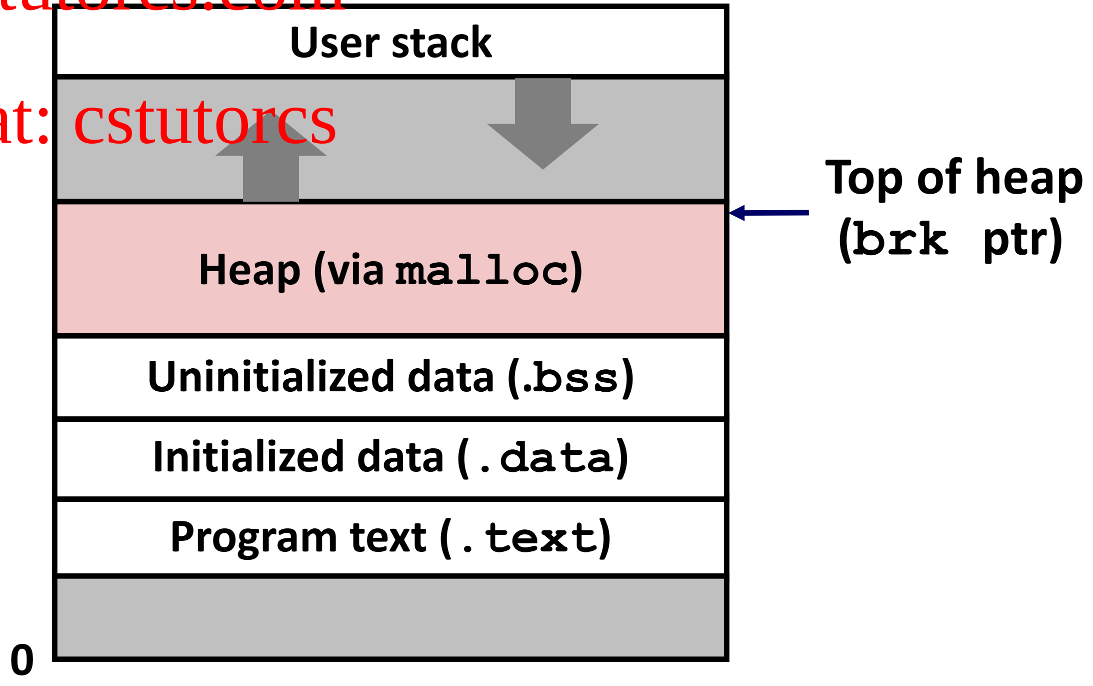
Dynamic Memory Allocation

- Programmers use ***dynamic memory allocators*** (such as `malloc`) to acquire VM at run time.
  - For data structures whose size is only known at runtime.
- Dynamic memory allocators manage an area of process virtual memory known as the ***heap***.





Dynamic Memory Allocation

- Allocator maintains heap as collection of variable sized ***blocks***, which are either ***allocated*** or ***free***
- Types of allocators
  - ***Explicit allocator***: application allocates and frees space
    - E.g., `malloc` and `free` in C
  - ***Implicit allocator:*** application allocates, but does not free space
    - E.g. garbage collection in Python, Java, ML, and Lisp

- **Will discuss simple explicit memory allocation today**

The `malloc` Package

`#include <stdlib.h>`

`void *malloc(size_t size)`

- Successful:
  - Returns a pointer to a memory block of at least **`size`** bytes (typically) aligned to 8-byte boundary
  - If **`size == 0`**, returns NULL
- Unsuccessful: returns NULL (0) and sets **`errno`**

`void free(void *p)`

- Returns the block pointed at by **`p`** to pool of available memory
- **`p`** must come from a previous call to **`malloc`** or **`realloc`**

Other functions

- **`calloc`**: Version of **`malloc`** that initializes allocated block to zero.
- **`realloc`**: Changes the size of a previously allocated block.
- **`sbrk`**: Used internally by allocators to grow or shrink the heap

malloc Example

```
void foo(int n, int m) {
    int i, *p;

    /* Allocate a block of n ints */
    p = (int *) malloc(n * sizeof(int));
    if (p == NULL) {
        perror("malloc");
        exit(0);
    }

    /* Initialize allocated block */
    for (i=0; i<n; i++)
        p[i] = i;


    /* Return p to the heap */
    free(p);
}
```