

Operating Systems

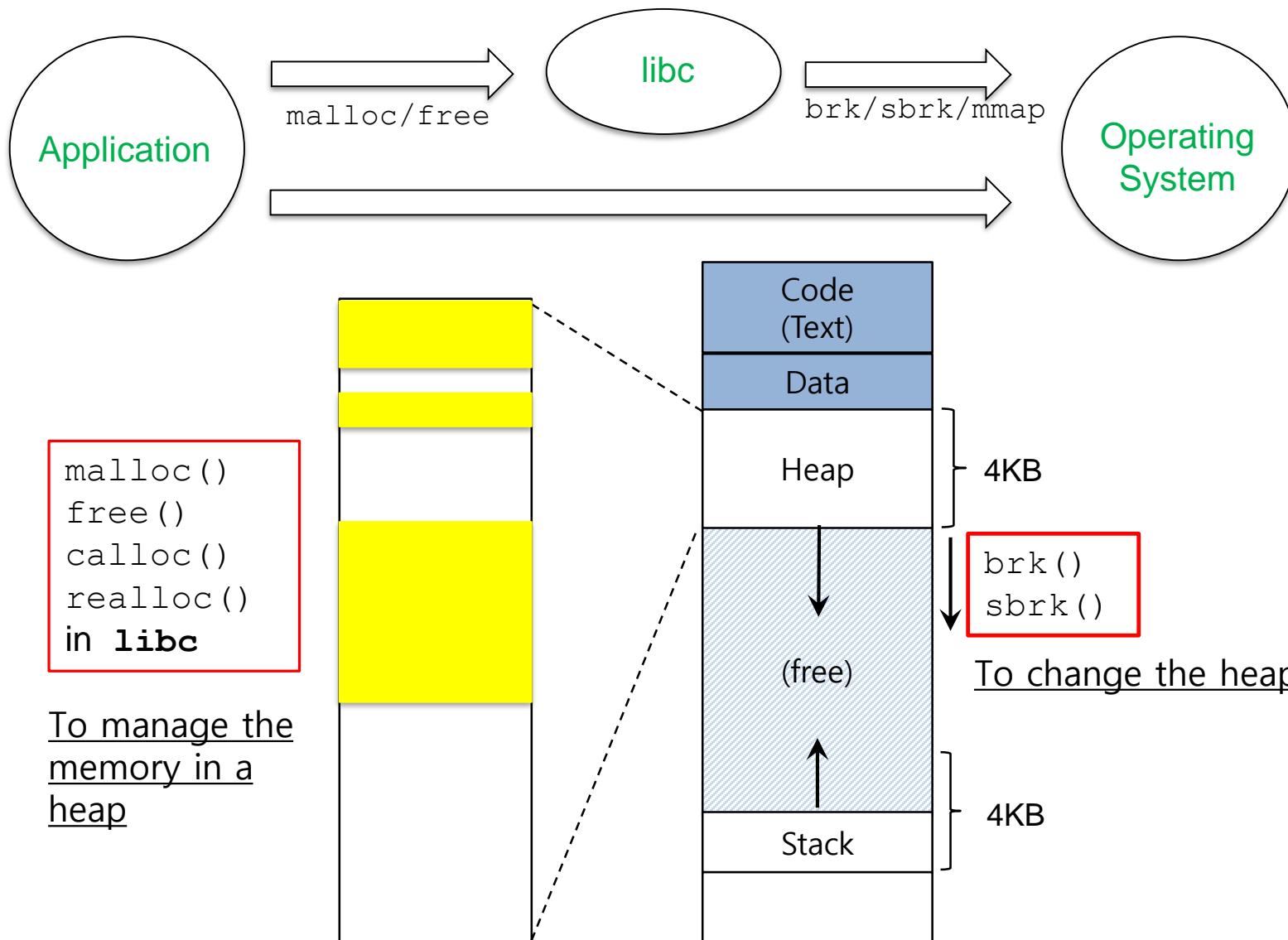
KAIST

14. Memory API

Overview

- ❑ malloc/free
- ❑ calloc/realloc
- ❑ brk/sbrk
- ❑ mmap/munmap

Virtual Address Space



malloc()

```
#include <stdlib.h>

void* malloc(size_t size)
```

- Allocate a memory region on the heap.

- Argument

- size_t size : size of the memory block(in bytes)
 - size_t is an unsigned integer type.

- Return

- Success : a void type pointer to the memory block allocated by malloc
 - Fail : a null pointer

sizeof()

- Routines and macros are utilized for size in malloc instead typing in a number directly.
- Two types of results of sizeof with variables
 - ◆ The actual size of 'x' is known at run-time.

```
int *x = malloc(10 * sizeof(int));  
printf("%d\n", sizeof(x));
```

4

- ◆ The actual size of 'x' is known at compile-time.

```
int x[10];  
printf("%d\n", sizeof(x));
```

40

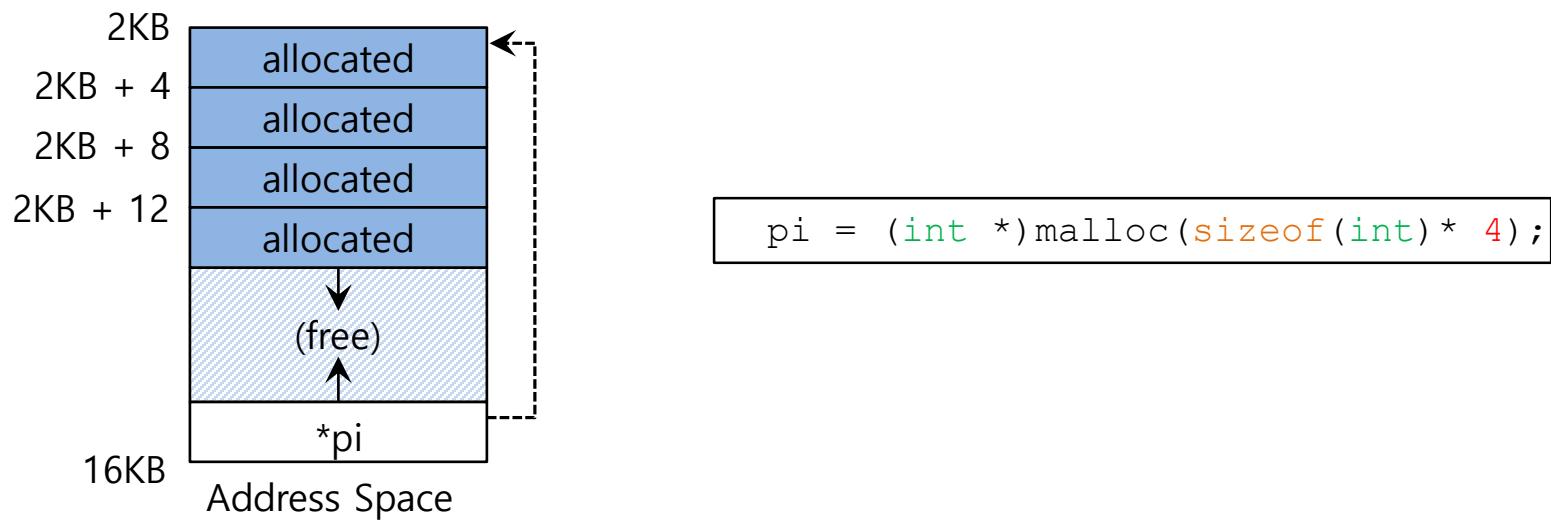
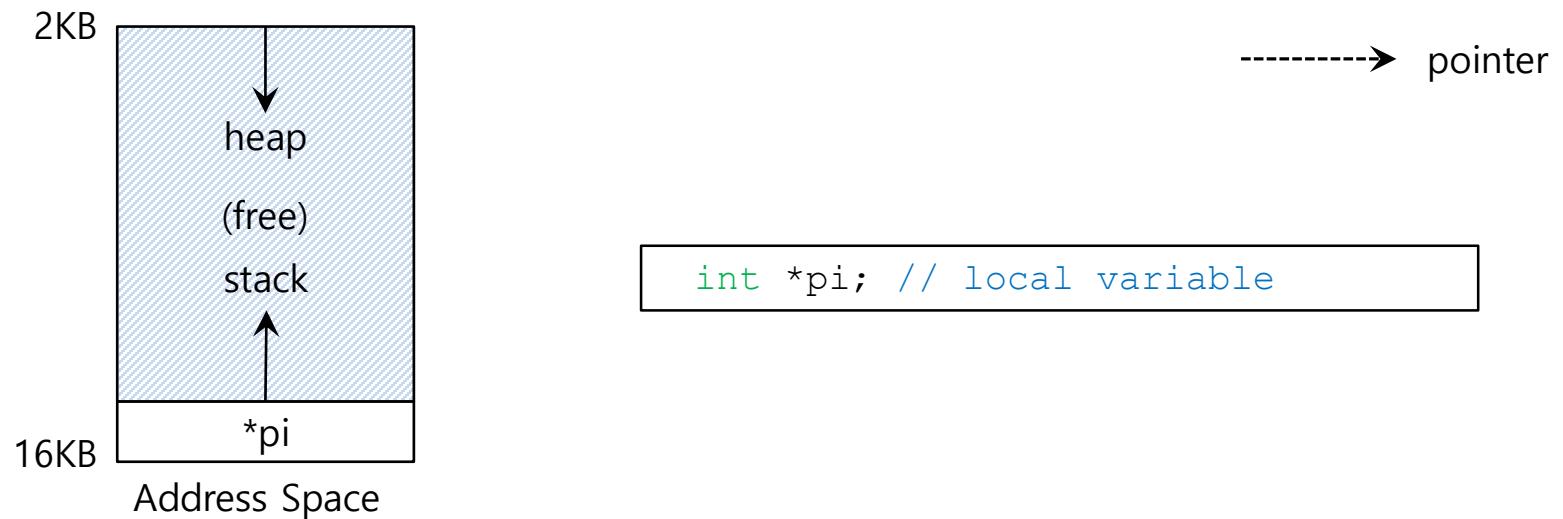
Memory API: free()

```
#include <stdlib.h>

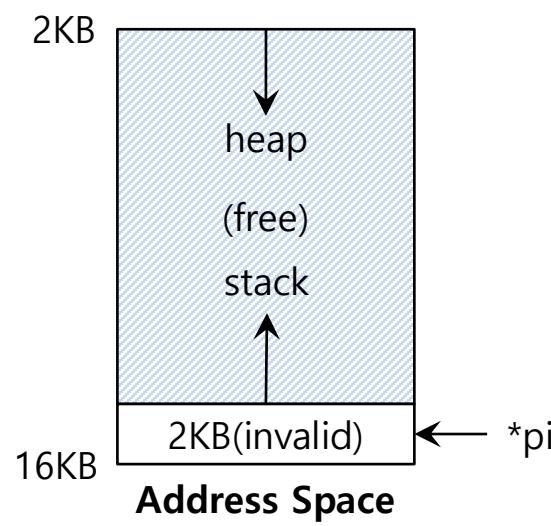
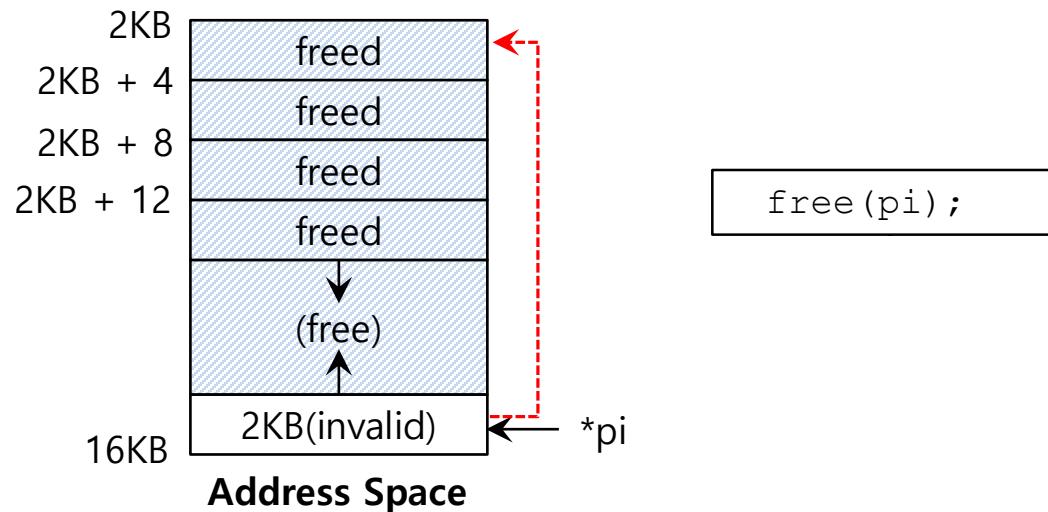
void free(void* ptr)
```

- ▣ Free a memory region allocated by a call to malloc.
 - ◆ Argument
 - void *ptr : a pointer to a memory block allocated with malloc
 - ◆ Return
 - none

Memory Allocating



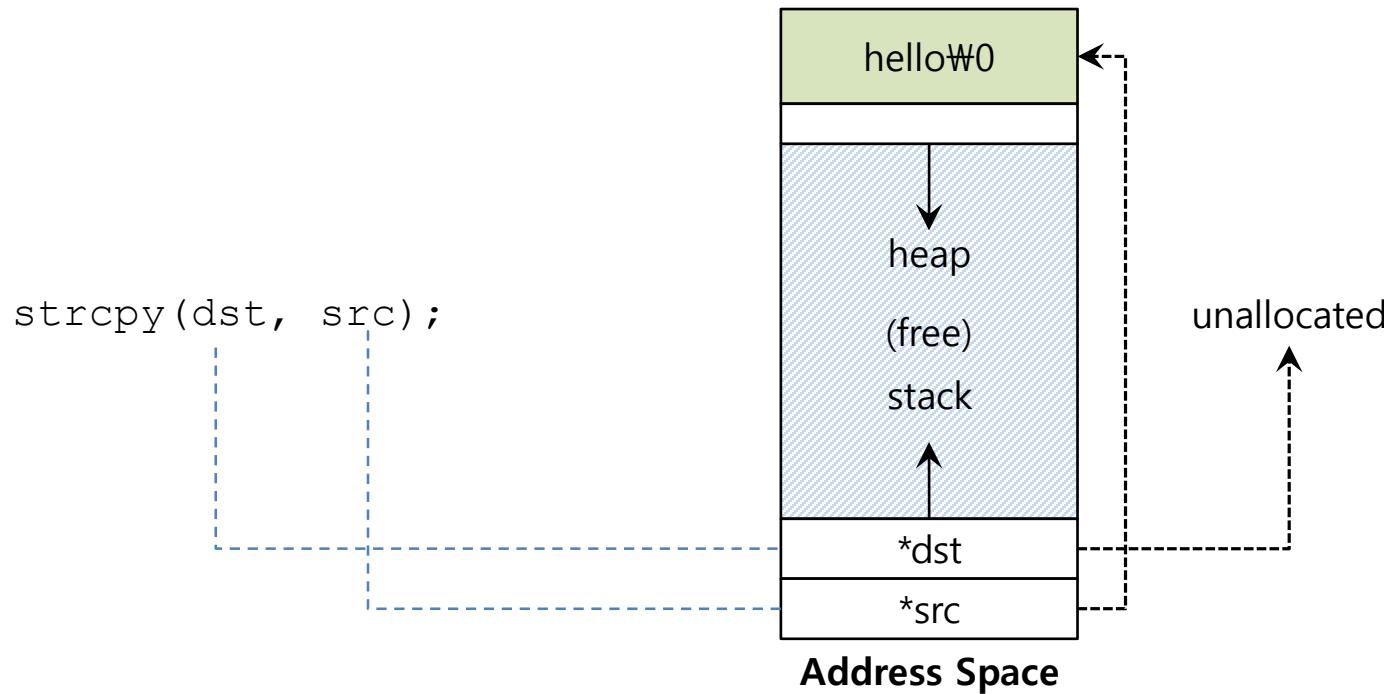
Memory Freeing



Forgetting To Allocate Memory

Incorrect code

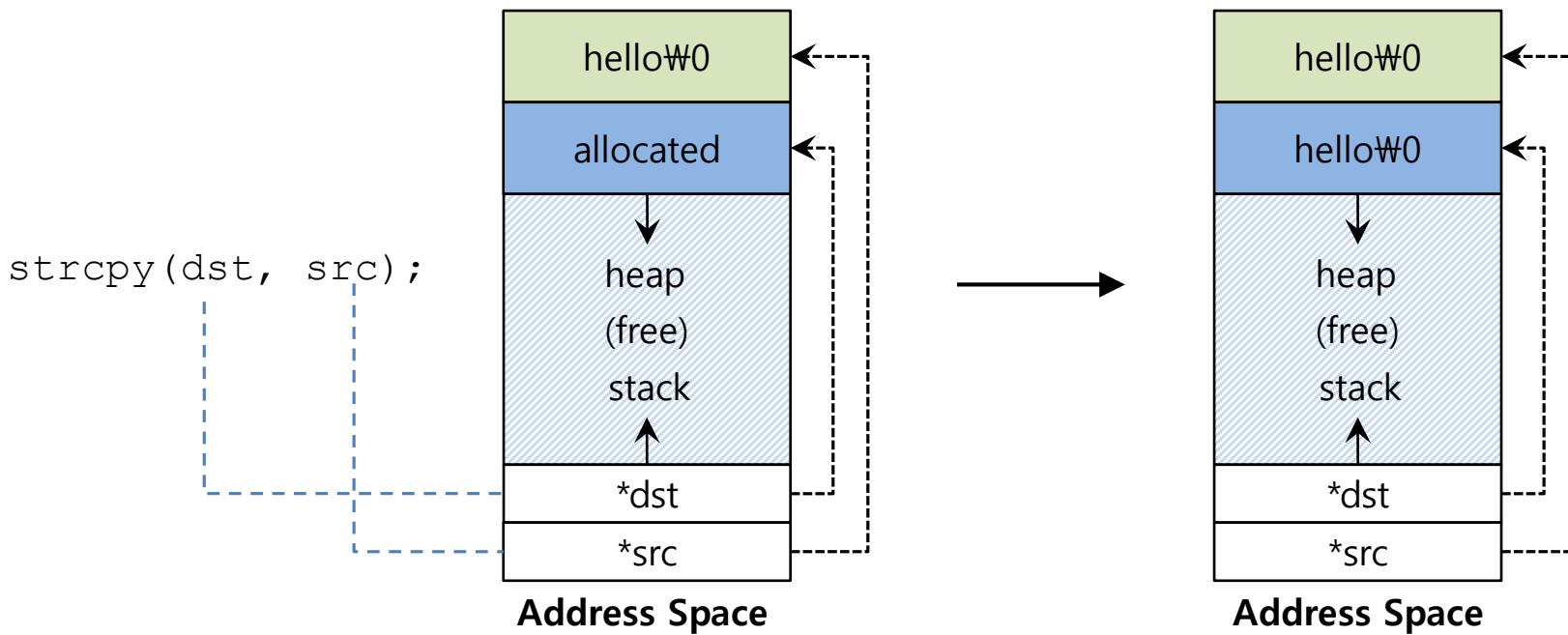
```
char *src = "hello"; //character string constant  
char *dst;           //unallocated  
strcpy(dst, src);   //segfault and die
```



Forgetting To Allocate Memory(Cont.)

❑ Correct code

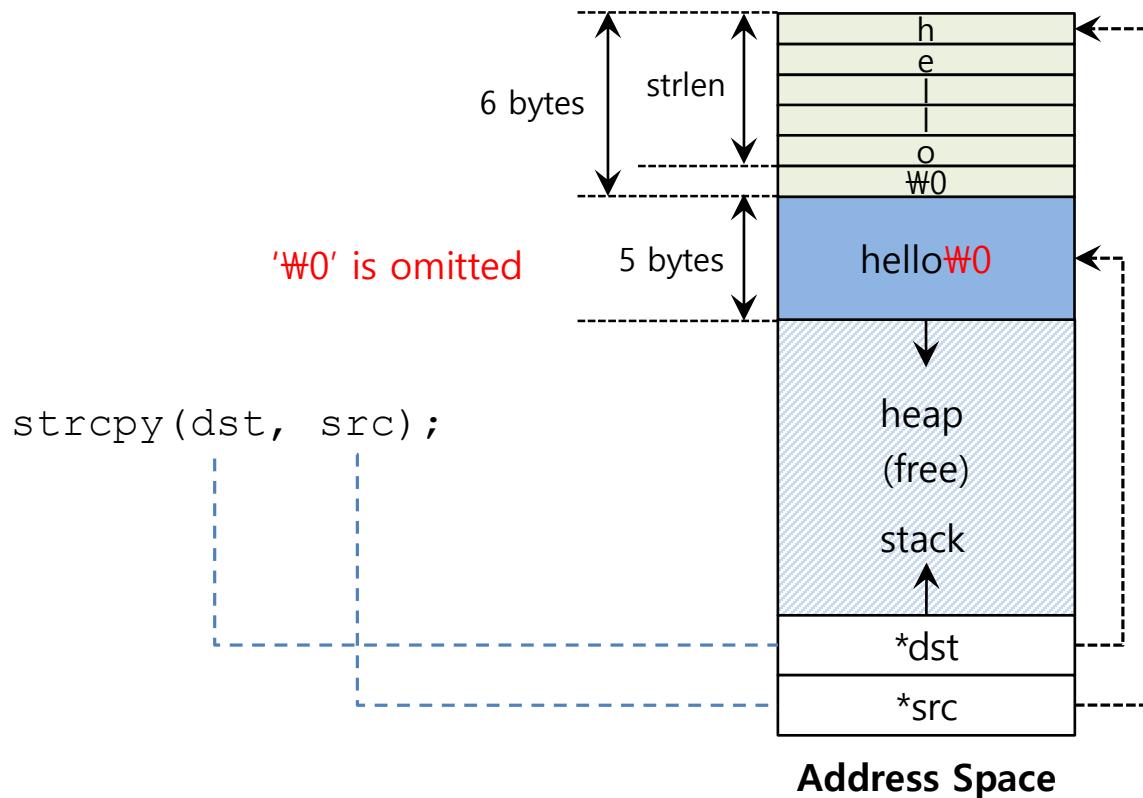
```
char *src = "hello"; //character string constant  
char *dst (char *)malloc(strlen(src) + 1 ); // allocated  
strcpy(dst, src); //work properly
```



Not Allocating Enough Memory

- Incorrect code, but work properly

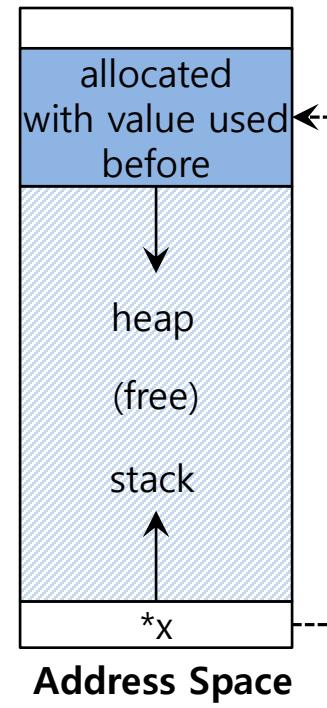
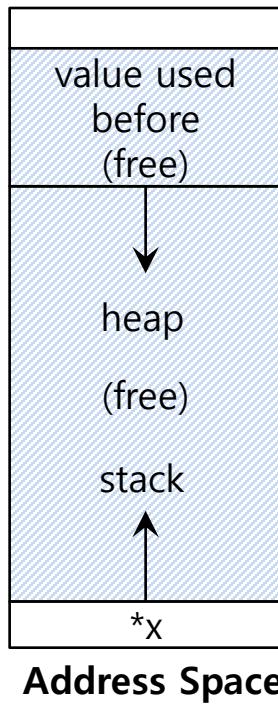
```
char *src = "hello"; //character string constant  
char *dst (char *)malloc(strlen(src)); // too small  
strcpy(dst, src); //work properly
```



Forgetting to Initialize

- Encounter an uninitialized read

```
int *x = (int *)malloc(sizeof(int)); // allocated  
printf("*x = %d\n", *x); // uninitialized memory access
```

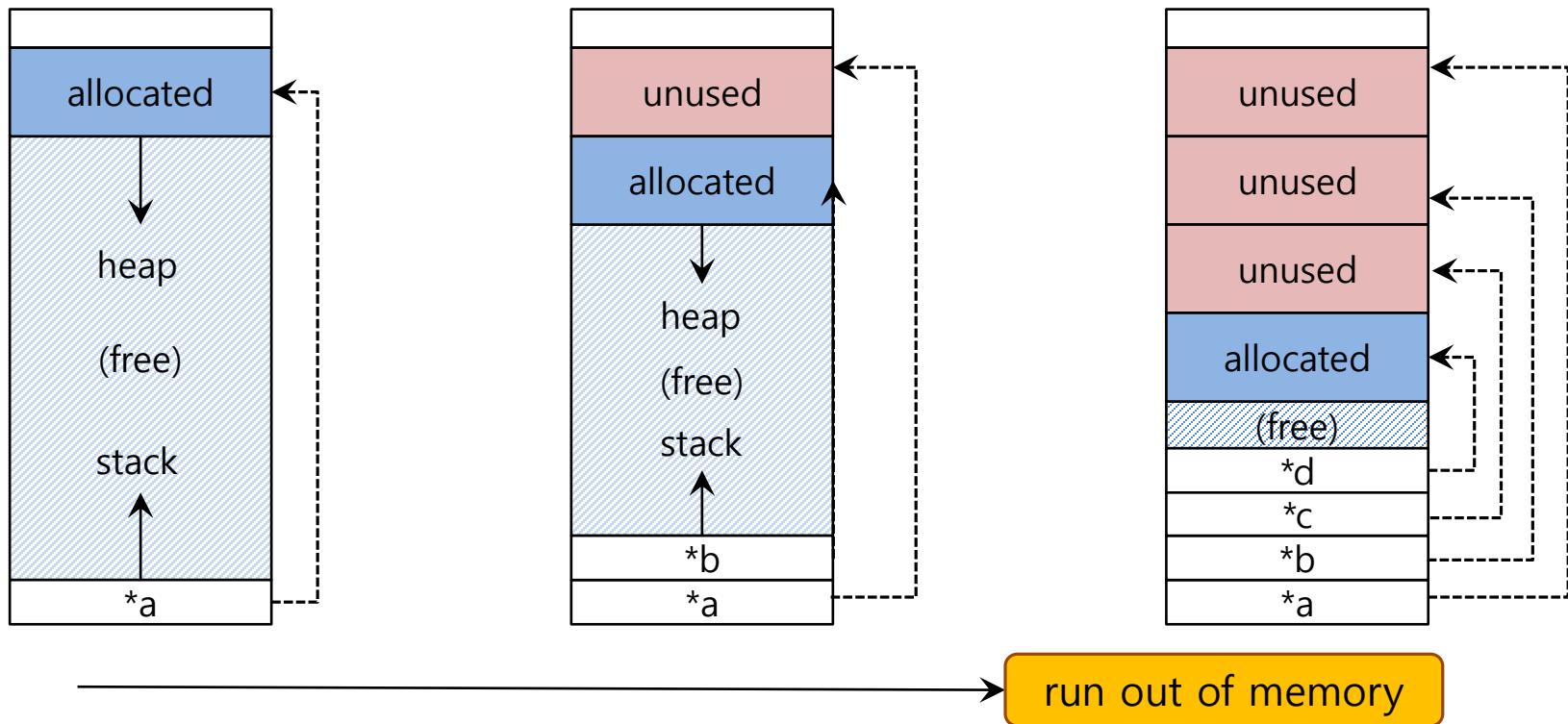


Memory Leak

- A program keeps allocating memory without freeing it.
- A program runs out of memory and eventually is killed by OS.

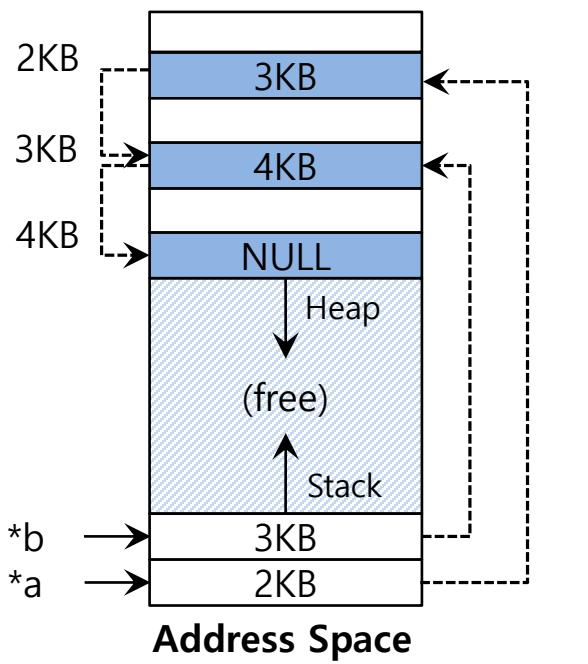
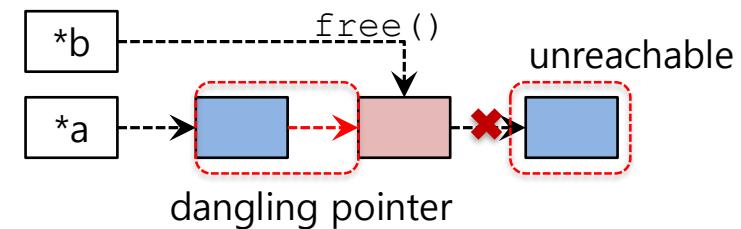
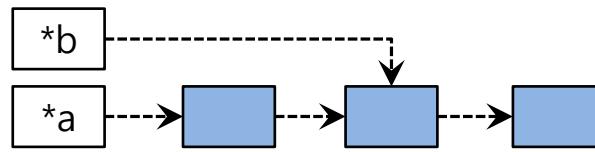
```
while(1)  
    malloc(4) ;
```

unused : allocated, but not freed

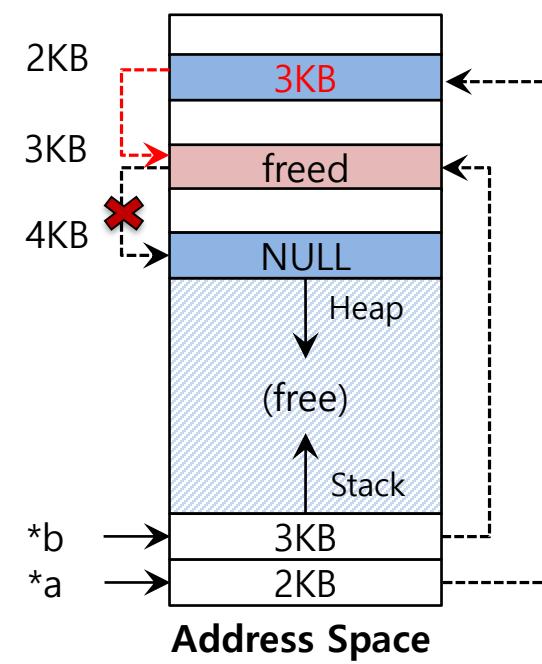


Dangling Pointer

- Freeing memory while it is being used.
 - A program accesses to memory with an invalid pointer



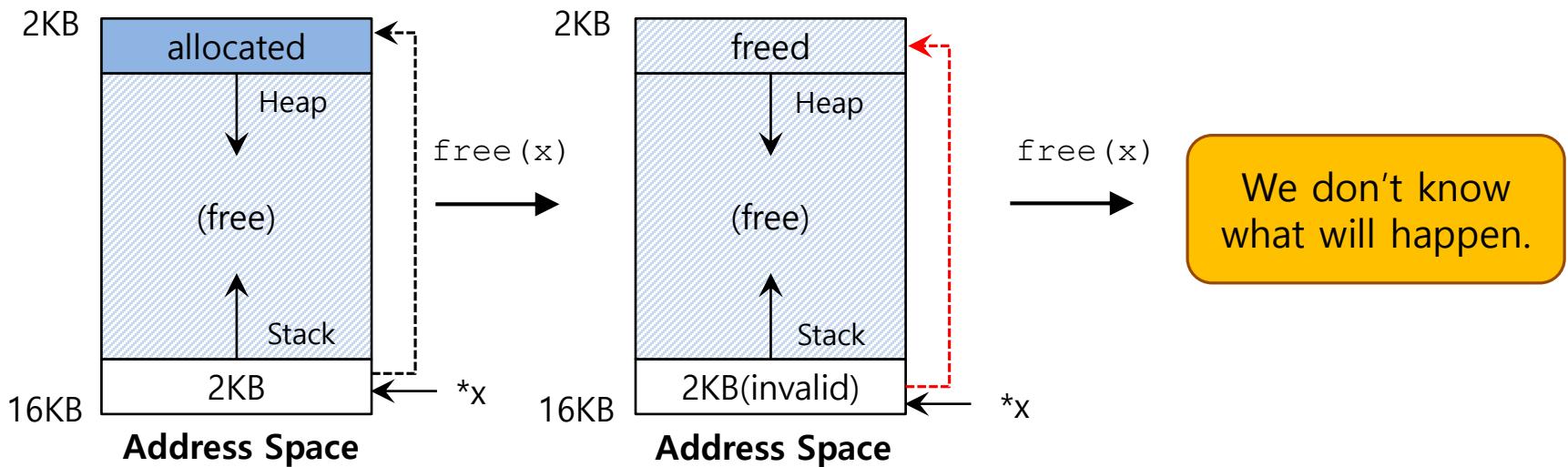
`free(b)`



Incorrect free()

- Free the memory that was freed already.

```
int *x = (int *)malloc(sizeof(int)); // allocated  
free(x); // free memory  
free(x); // free repeatedly
```



- Free the memory that was not allocated via malloc().

```
int *x = (int *)malloc(sizeof(int)); // allocated  
free(x+12); // free memory
```

Other Memory APIs: calloc() and realloc()

```
#include <stdlib.h>

void *calloc(size_t num, size_t size)
```

- Allocate memory and zeroes it before returning.

- size_t num : the number of objects to allocate
- size_t size : size of an object (in bytes)

```
#include <stdlib.h>

void *realloc(void *ptr, size_t size)
```

- Change the size of memory block.

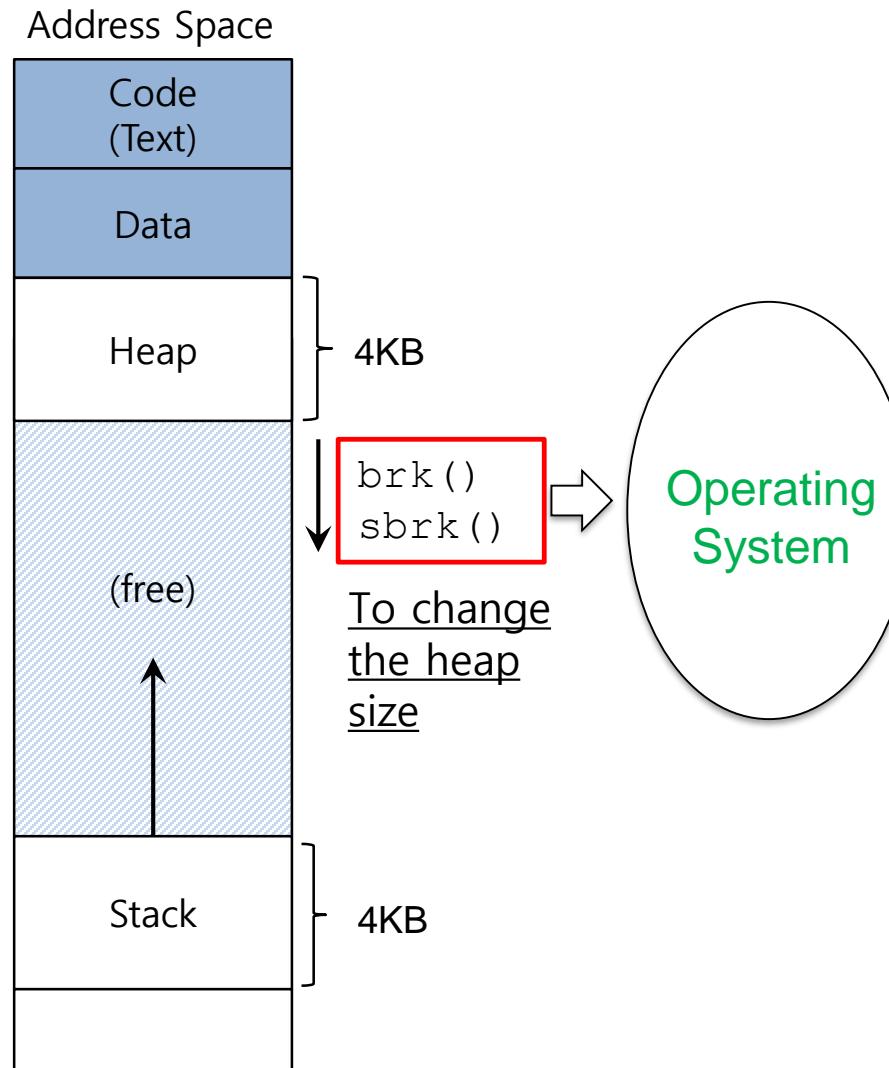
- void *ptr: Pointer to memory block allocated with malloc, calloc or realloc
- size_t size: New size for the memory block(in bytes)

System Calls

```
#include <unistd.h>

int brk(void *addr)
void *sbrk(intptr_t increment);
```

- ▣ There lacks of heap space. → Ask OS to expand heap.
- ▣ break: The location of **the end of the heap** in address space
- ▣ malloc uses **brk** system call.
 - ◆ brk is called to expand the program's *break*.
 - ◆ sbrk is similar to brk.
 - ◆ Programmers **should never directly call** either brk or sbrk.



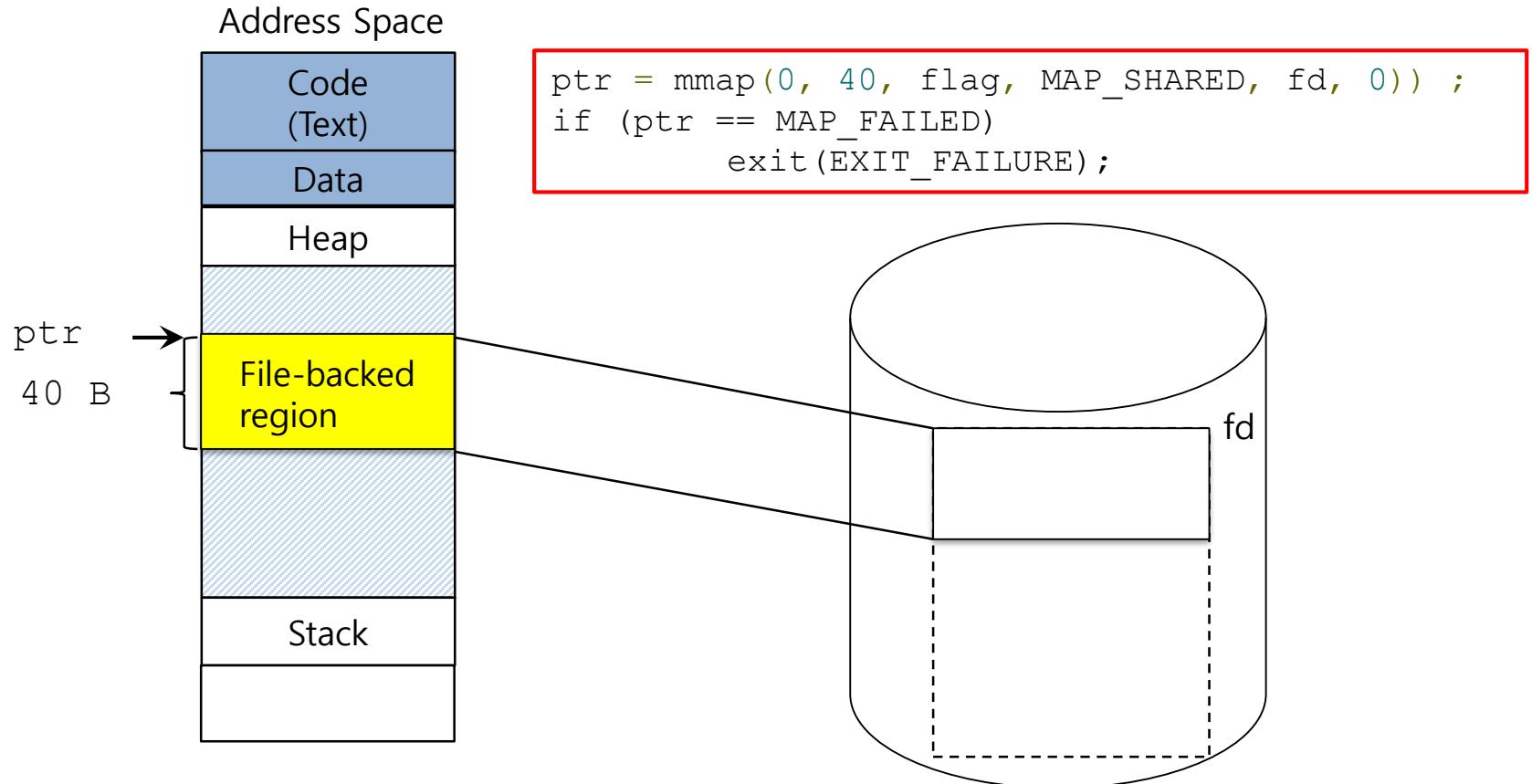
System Calls: mmap

```
#include <sys/mman.h>

void *mmap(void *ptr, size_t length, int prot, int flags,
           int fd, off_t offset)
```

- Allocate a memory region of length at ptr.
- If fd is not negative, associate the region to fd starting at offset.

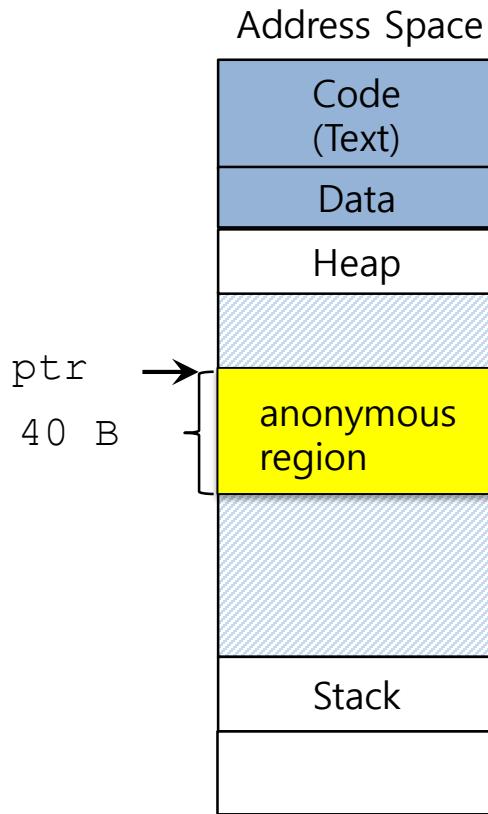
mmap: creating file-backed region



mmap: creating anonymous region

```
#include <sys/mman.h>

void *mmap(void *ptr, size_t length, int
prot, int flags, int fd, off_t offset)
```



```
ptr= mmap(NULL, 40, PROT_READ |
          PROT_WRITE, MAP_SHARED |
          MAP_ANONYMOUS, -1, 0);
if (ptr == MAP_FAILED)
    exit(EXIT_FAILURE);
```

Summary

- ❑ malloc/free
- ❑ calloc/realloc
- ❑ mmap/munmap