IOWA STATE UNIVERSITY

Department of Electrical and Computer Engineering

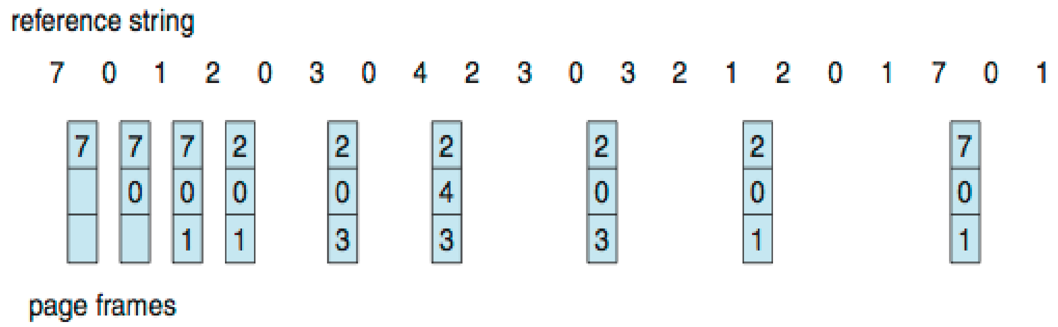
Lecture 25: Memory APIs

Agenda

- Recap
- Prefetching & Thrashing
- Memory APIs
  - malloc() / free()
  - calloc()
  - realloc()

Recap

- Page Replacement Algorithms
  - The Optimal Algorithm
  - FIFO Algorithm
  - LRU Algorithm
  - Clock Algorithm

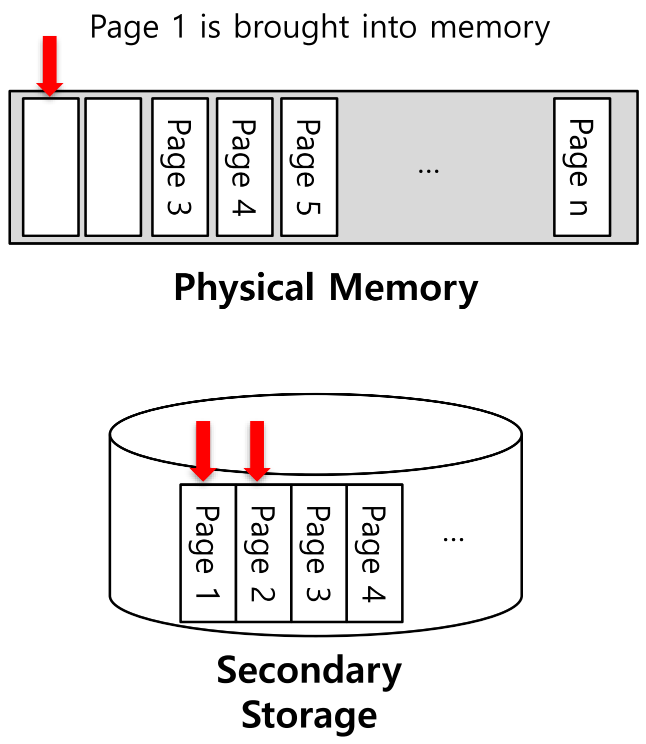


Agenda

- ~~Recap~~
- Prefetching & Thrashing
- Memory APIs
  - malloc() / free()
  - calloc()
  - realloc()

Prefetching

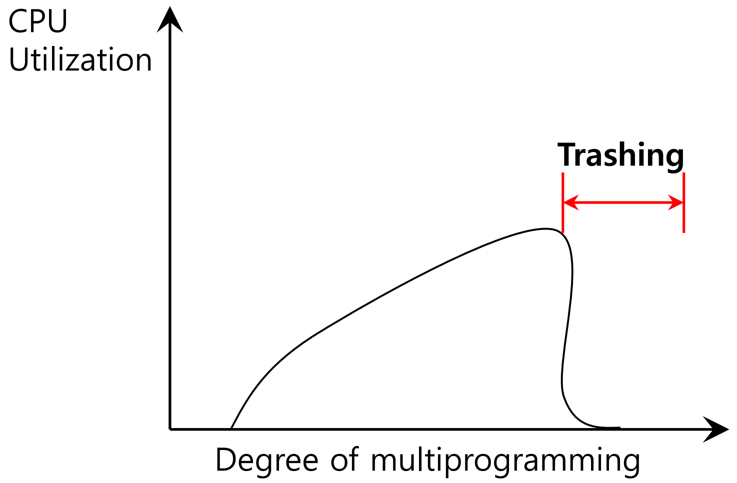
- The OS guess that a page is about to be used, and thus bring it in ahead of time.



Page 2 likely soon to be accessed, so bring it into memory together with (requested) page 1

Thrashing

- The OS keeps swapping pages in and out for processes
  - Usually happen when the memory is oversubscribed
    - the memory demands of the set of running processes exceeds the available physical memory significantly
  - Low CPU utilization



Agenda

- ~~Recap~~

- ~~Prefetching & Thrashing~~

- Memory APIs
  - malloc() / free()
  - calloc()
  - realloc()

Memory API: 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

Memory API: malloc()

- Instead of typing in a number directly for `size` in `malloc`, use `sizeof` to ensure the requested number of bytes is accurate
- e.g., `sizeof()`

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

```
4
```

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

```
40
```

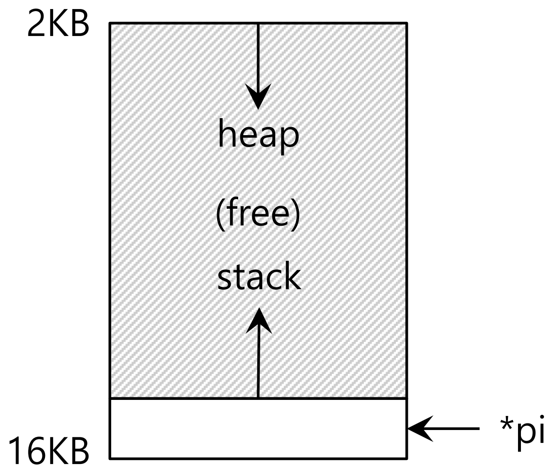
Memory API: free()

```
#include <stdlib.h>

void free(void* ptr)
```

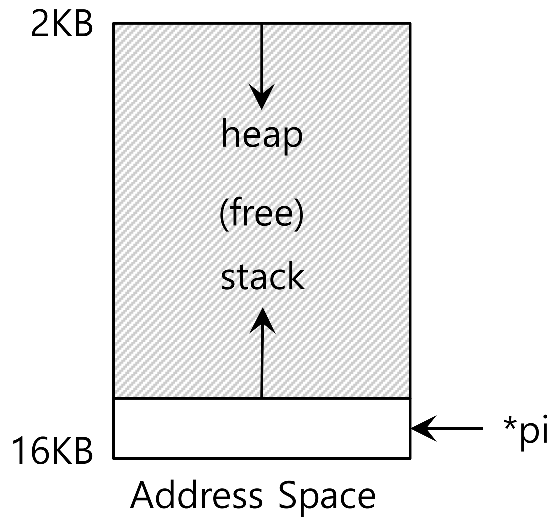
- Free a memory region allocated by a call to `malloc`.
  - Argument
    - `void *ptr`: a pointer to a memory block allocated with `malloc`
  - Return
    - none

Memory Allocation

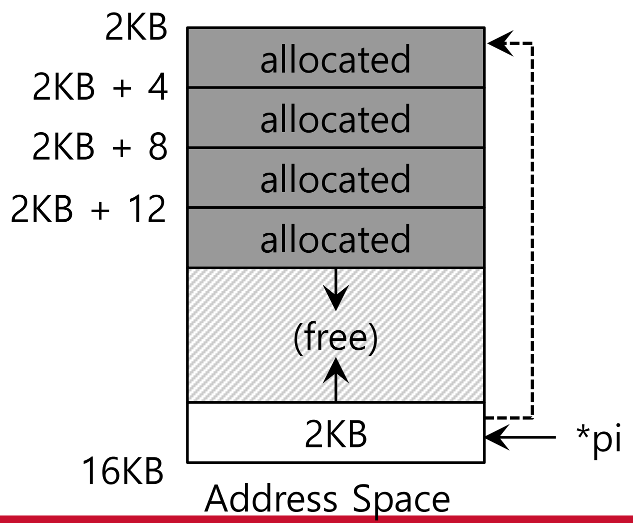


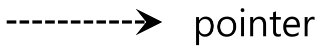
```
int *pi; // local variable
```

Memory Allocation



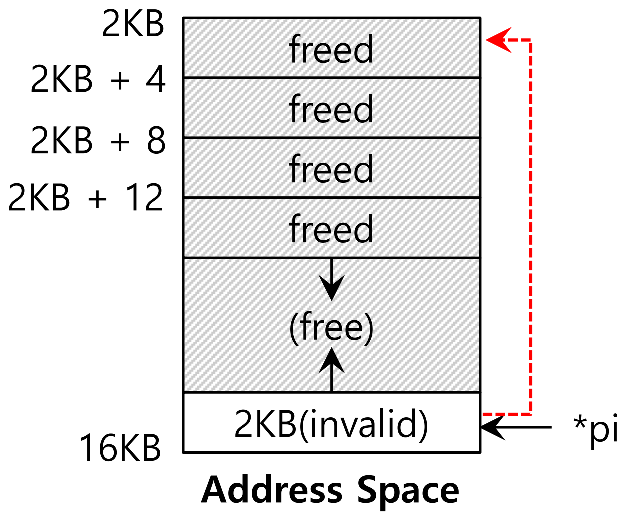
```
int *pi; // local variable
```



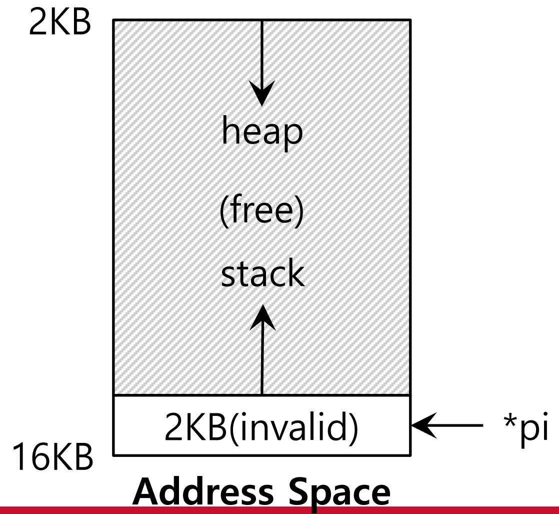


```
pi = (int *)malloc(sizeof(int)* 4);
```

Memory Freeing

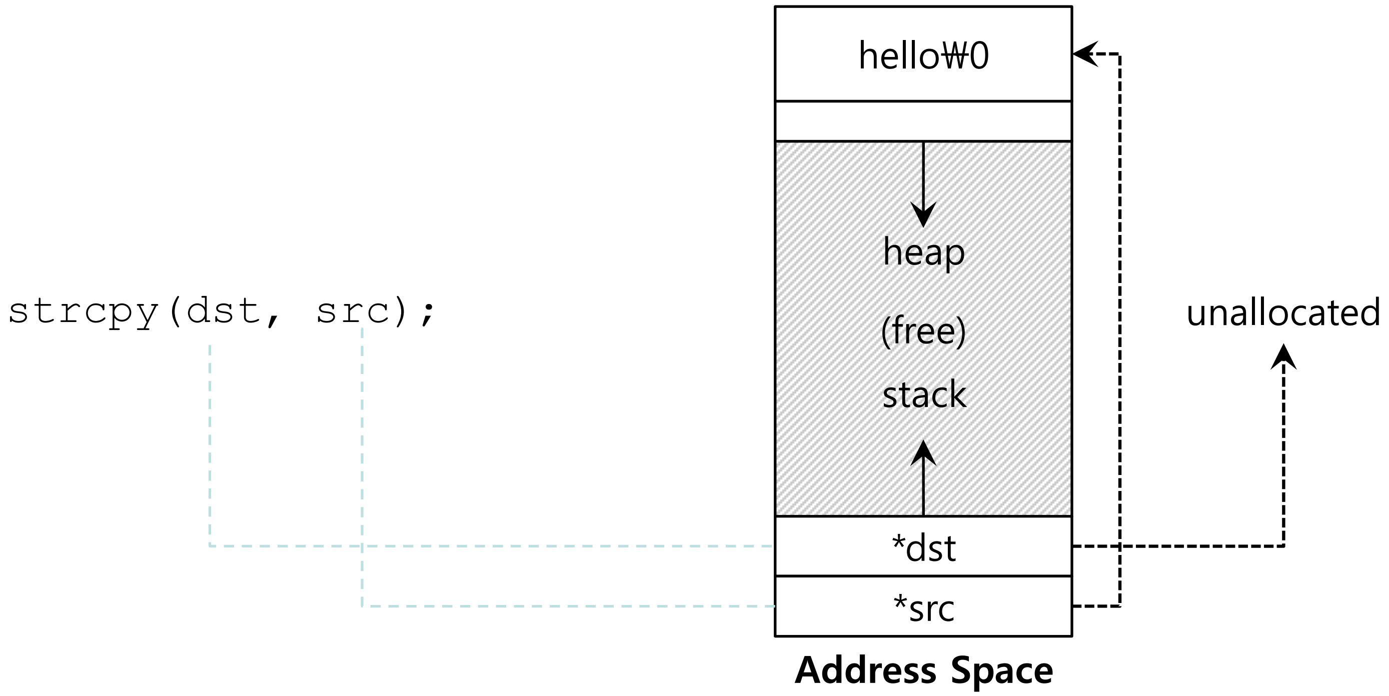


```
free(pi);
```



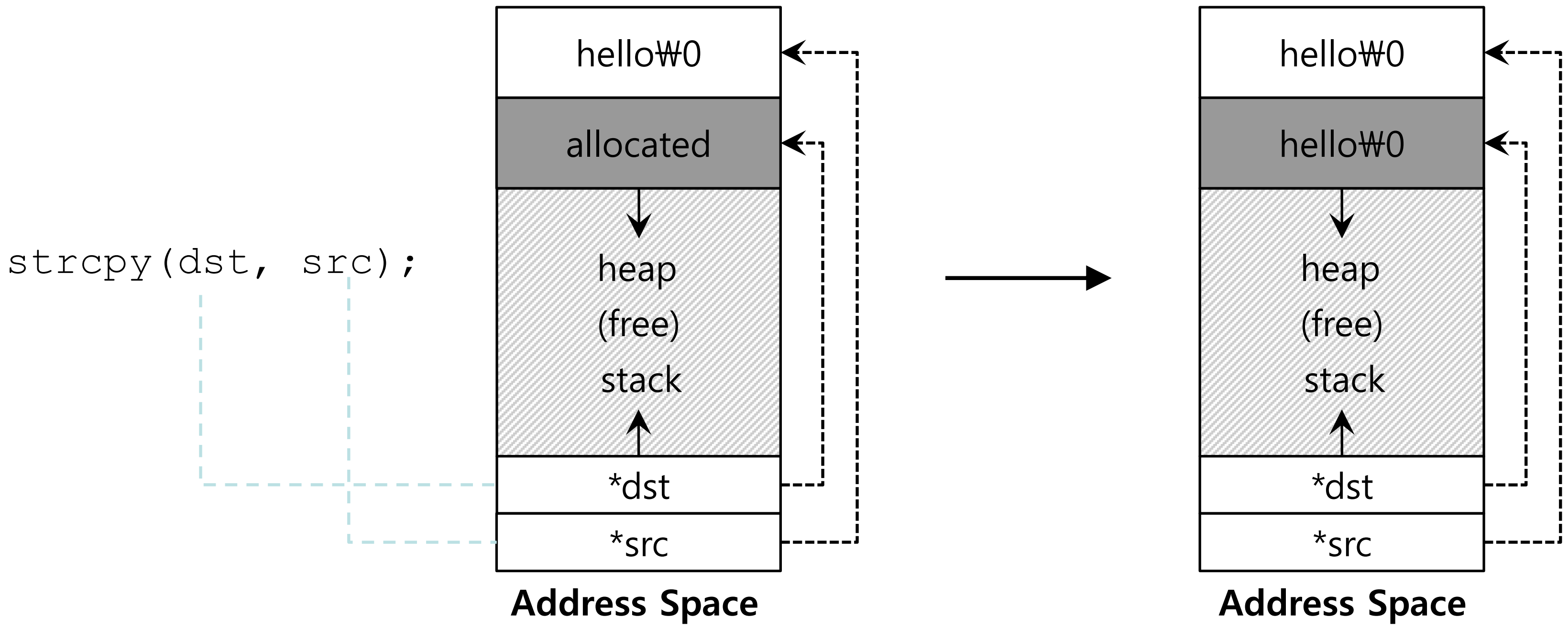
Forget To Allocate Memory

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



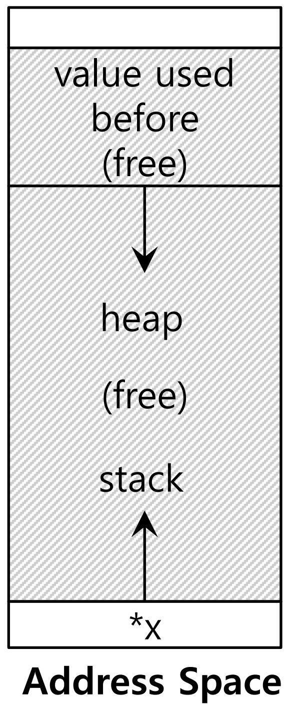
Forget To Allocate Memory (cont')

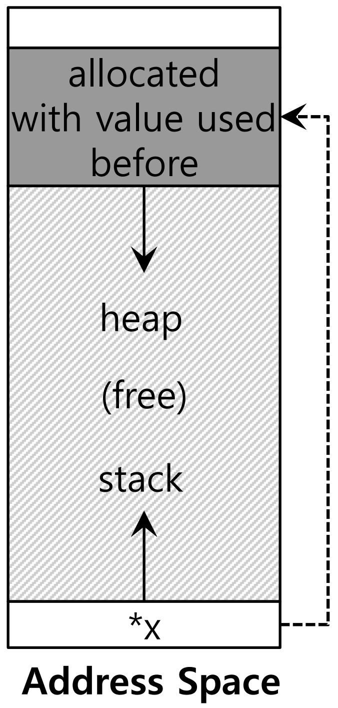
```
char *src = "hello";   //character string constant
char *dst (char *)malloc(strlen(src) + 1 ); // allocated
strcpy(dst, src);      //work properly
```



Forget To Initialize Memory

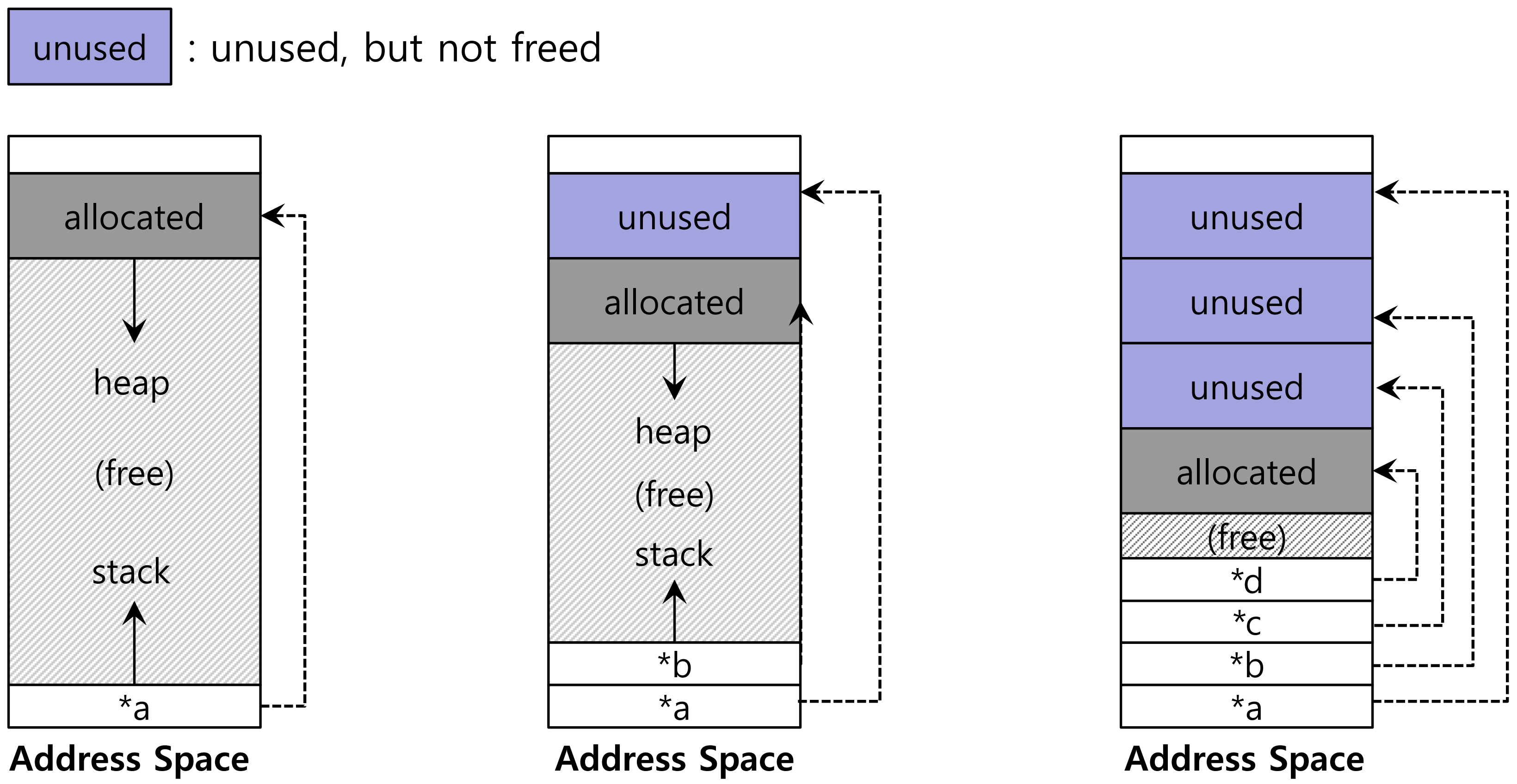
```
int *x = (int *)malloc(sizeof(int)); // allocated
printf("*x = %d\n", *x); // uninitialized memory access
```





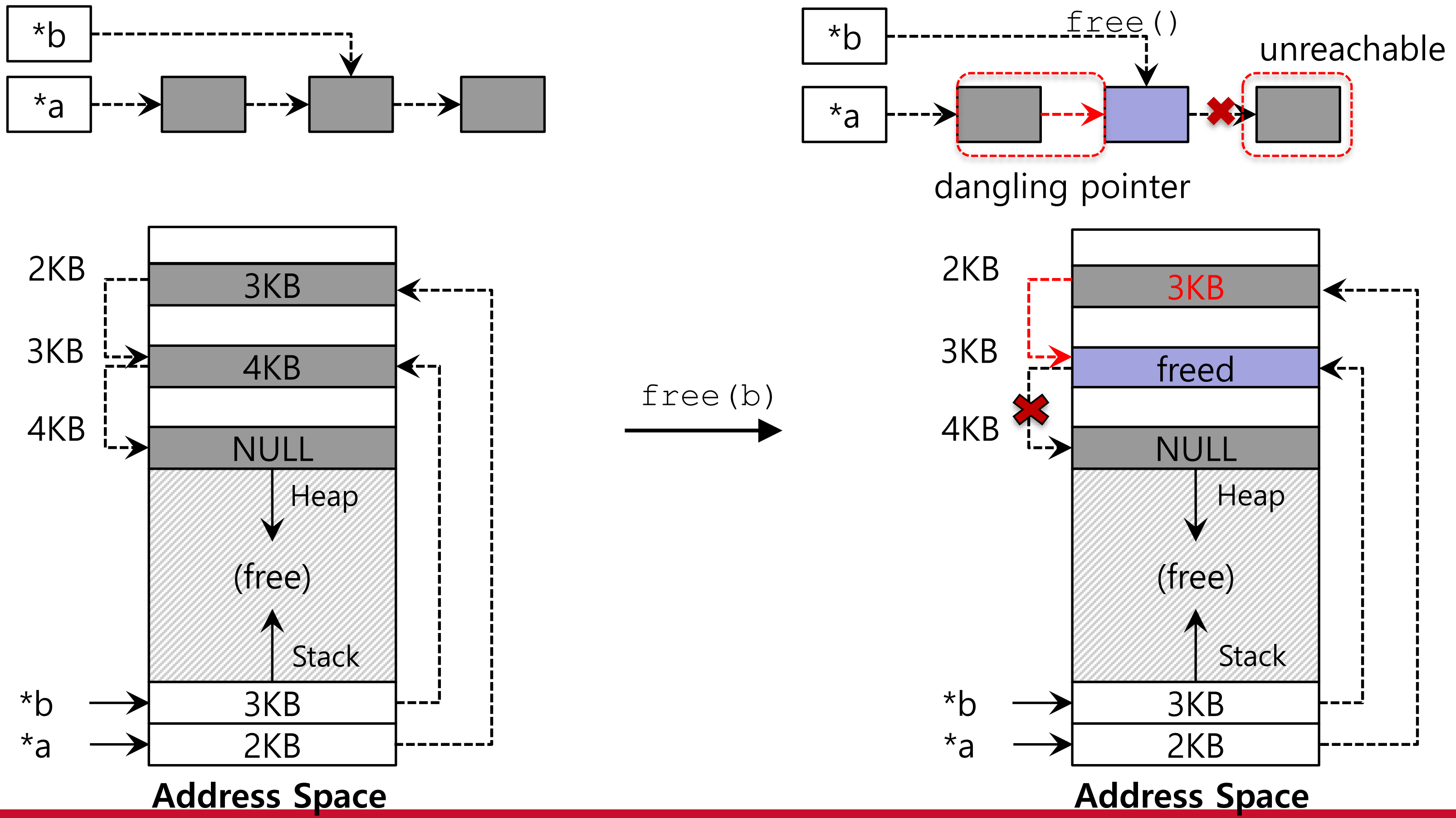
Memory Leak

- Keep allocating memory but forget to deallocate (free)



Dangling Pointer

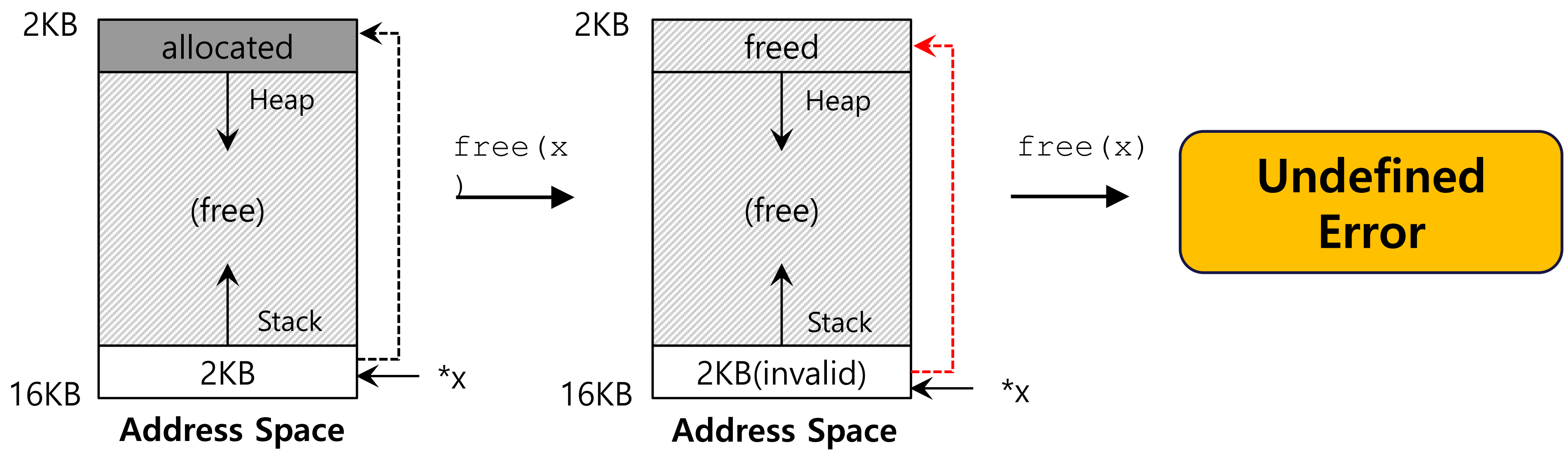
- Free memory before it is finished using
  - a process accesses to memory with an invalid pointer



Double Free

- Free memory that had been freed already

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



Other Memory APIs: calloc()

```
#include <stdlib.h>

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

- Allocate memory on the heap and zeroes it before returning.
  - Argument
    - `size_t num` : number of blocks to allocate
    - `size_t size` : size of each block(in bytes)
  - Return
    - Success : a void type pointer to the memory block allocated by `calloc`
    - Fail : a null pointer

Other Memory APIs: realloc()

```
#include <stdlib.h>

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

- Change the size of memory block.
  - A pointer returned by `realloc` may be either the same as `ptr` or a new.
  - Argument
    - `void *ptr`: Pointer to memory block allocated with `malloc`, `calloc` or `realloc`
    - `size_t size`: New size for the memory block(in bytes)
  - Return
    - Success: Void type pointer to the memory block
    - Fail : Null pointer

Related System Calls

- Internally, `malloc` library call invokes `brk/sbrk` system calls
  - expand the program's *break*.
    - *break*: The location of **the end of the heap** in address space
  - programmers should never directly call either `brk` or `sbrk`.

```
#include <unistd.h>

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

Agenda

- ~~Recap~~
- ~~Prefetching & Thrashing~~
- ~~Memory APIs~~
  - ~~malloc() / free()~~
  - ~~calloc()~~
  - ~~realloc()~~

Questions?

*acknowledgement: slides include content from “Modern Operating Systems” by A. Tanenbaum, “Operating Systems Concepts” by A. Silberschatz etc., “Operating Systems: Three Easy Pieces” by R. Arpaci-Dusseau etc., and anonymous pictures from internet.