

# **Operating Systems**

**KAIST**

## **13. The Abstraction: Address Space**

---

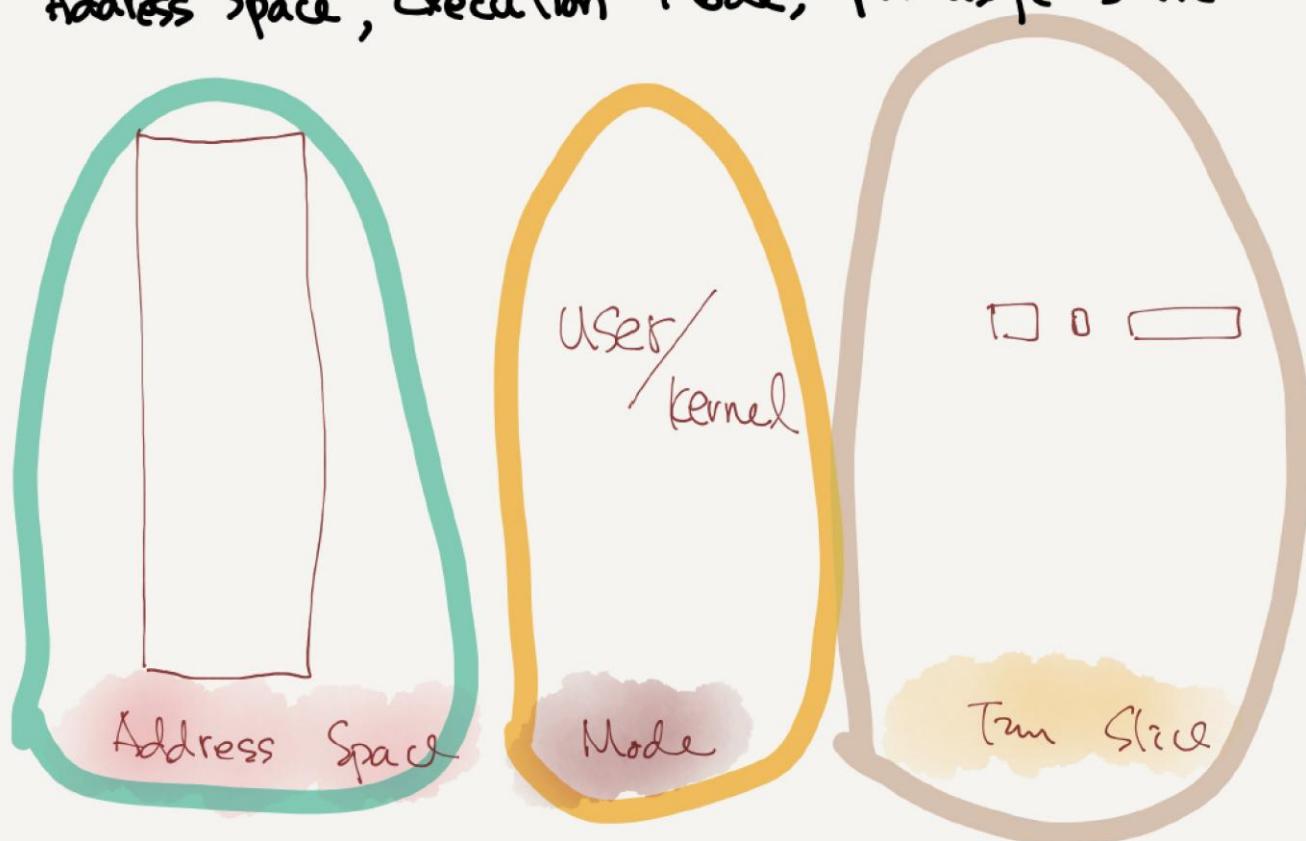
# Memory Virtualization

## ❑ What is **memory virtualization**?

- ◆ OS virtualizes its physical memory.
- ◆ OS provides an **illusion memory space** per each process.
- ◆ It seems to be seen like **each process uses the whole memory** .

## Process

- Unit of Isolation
- Address Space, Execution Mode, CPU usage state



## Address Space (User)

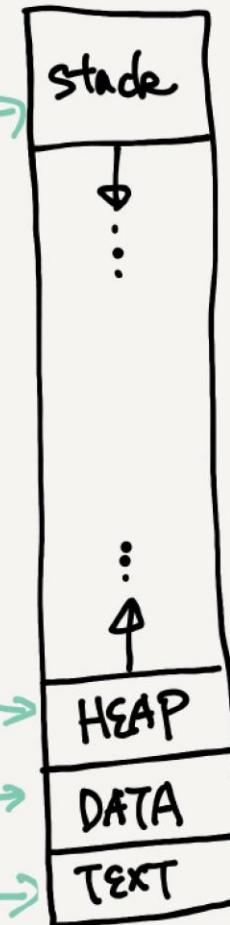
Objects in the address space

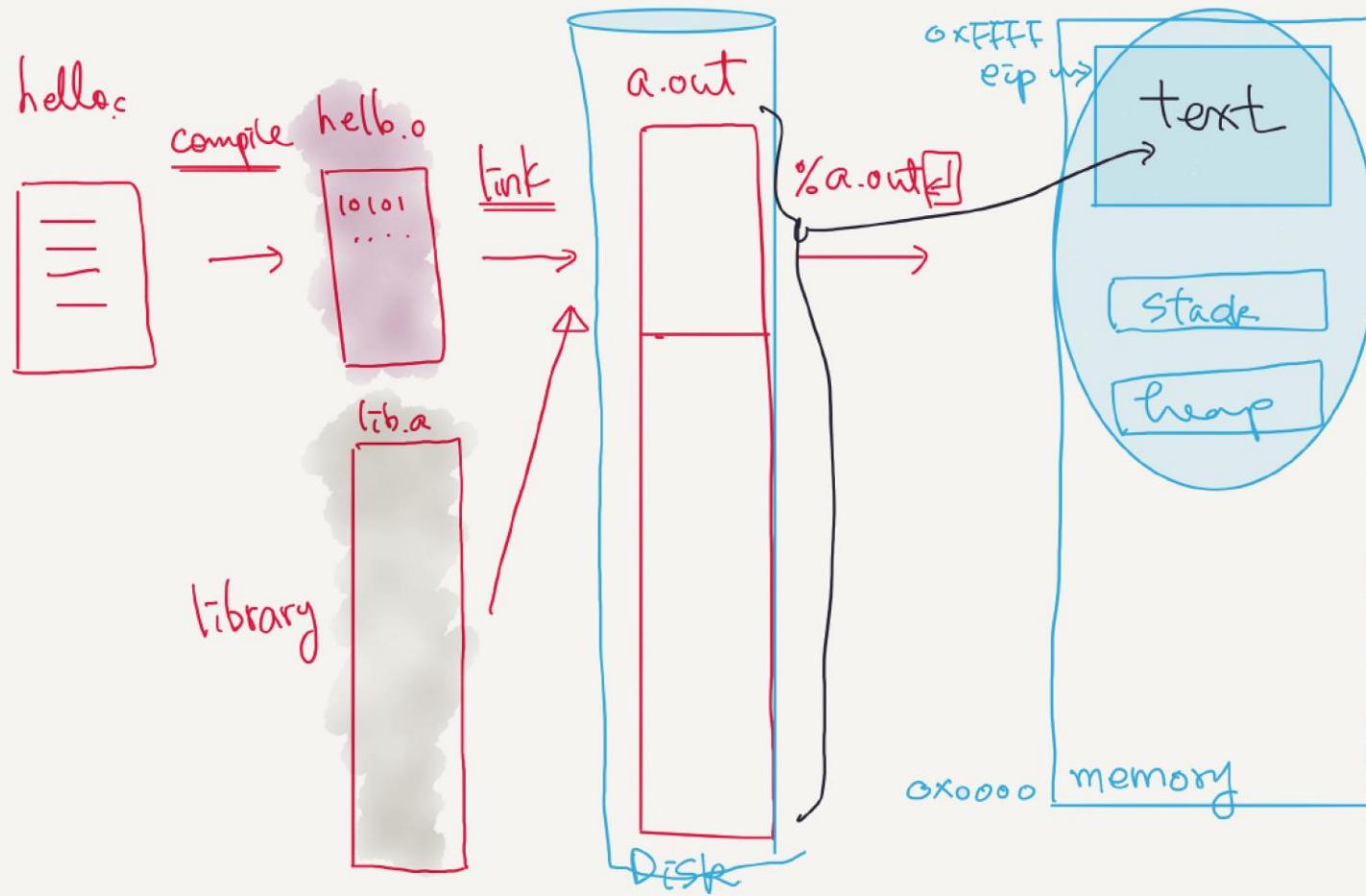
Instructions

local variables  
function call return address

Dynamic Memory  
: malloc();

global and static variables

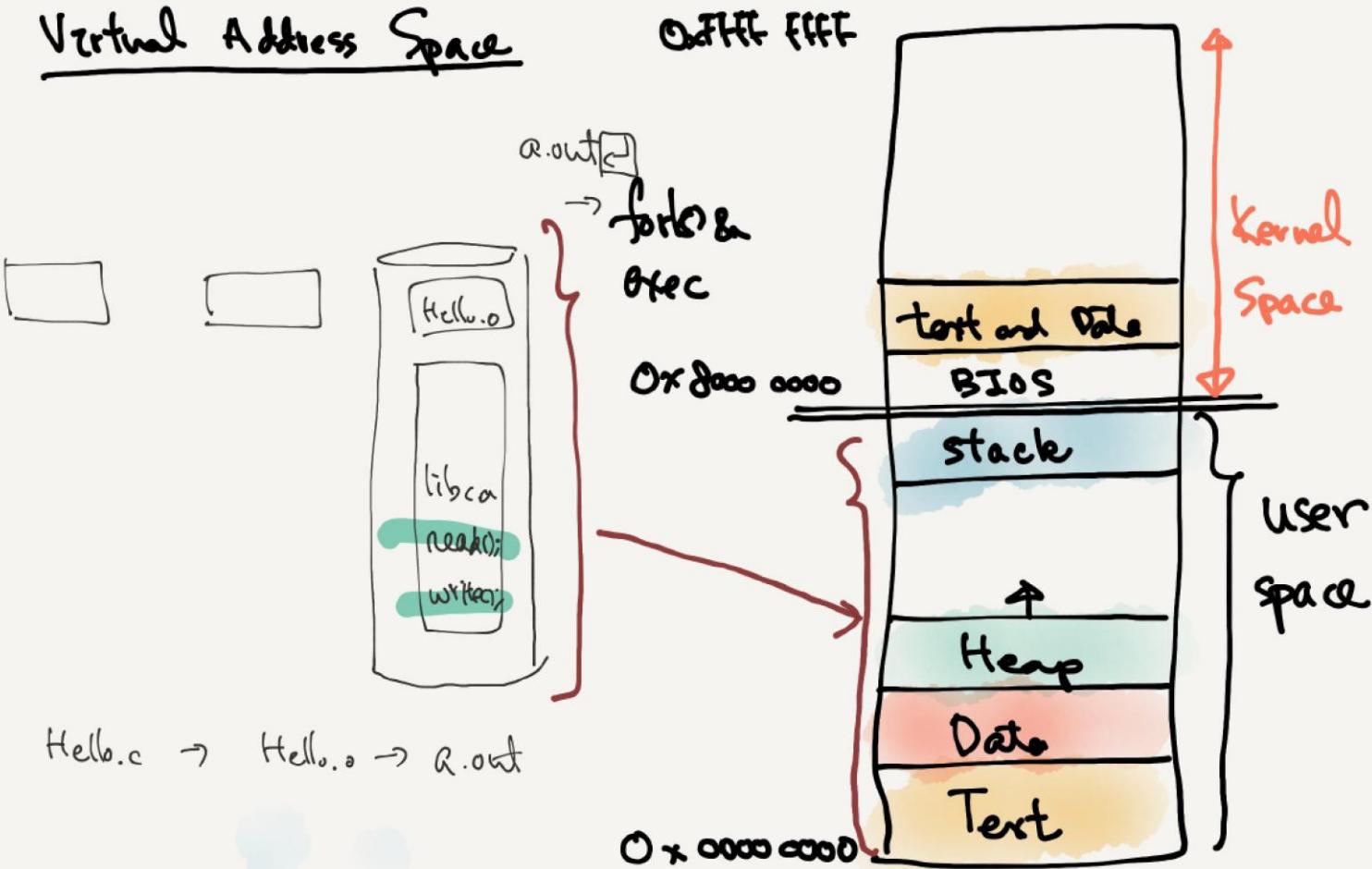




from lecture -1

but, where is kernel?

## Virtual Address Space



Process Address Space  
(Virtual Address)

# Process structure: struct thread

pintos/src/threads/thread.h

```
struct thread
{
    /* Owned by thread.c. */
    tid_t tid;
    enum thread_status status;
    char name[16];
    uint8_t *stack;
    int priority;
    struct list_elem allelem;

    /* Shared between thread.c and synch.c. */
    struct list_elem elem;           /* List element. */

#ifndef USERPROG
    /* Owned by userprog/process.c. */
    uint32_t *pagedir;              /* Page directory. */
#endif

    /* Owned by thread.c. */
    unsigned magic;                 /* Detects stack overflow. */
};
```

enum thread\_status

```
{
```

THREAD_RUNNING,	/* Running thread. */
THREAD_READY,	/* Not running but ready to run. */
THREAD_BLOCKED,	/* Waiting for an event to trigger. */
THREAD_DYING	/* About to be destroyed. */

```
};
```

/\* Thread identifier. \*/  
/\* Thread state. \*/  
/\* Name (for debugging purposes). \*/  
/\* Saved stack pointer. \*/  
/\* Priority. \*/  
/\* List element for all threads list. \*/

## Process

: struct thread

- page table

p->pagedir

- kernel stack

p->stack

- run state

p->status

process: unit of isolation

- page table

thread: unit of execution

- registers

- stacks

- local variables

- function call return address

# User stack vs. kernel stack

Executing in User Mode

- user stack

executing in the kernel

- use kernel stack

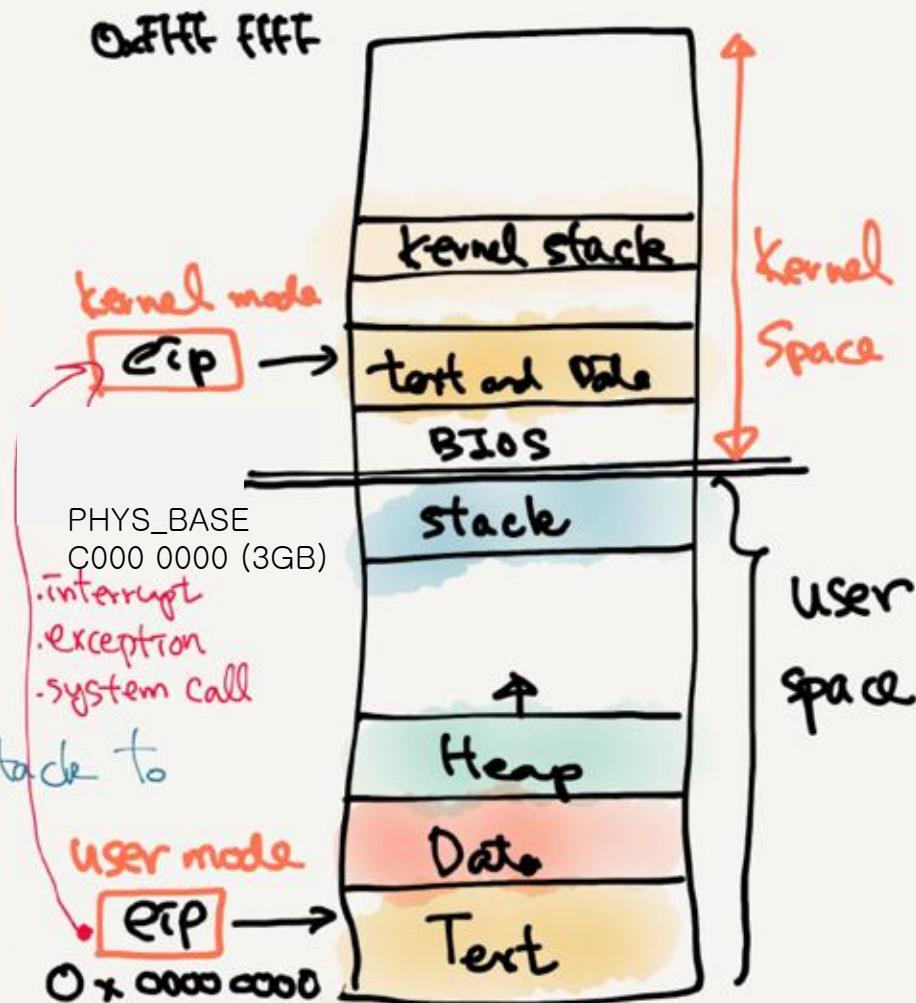
- System Call

- entering the kernel

① **switch** from user stack

kernel stack

② **raise privilege level**

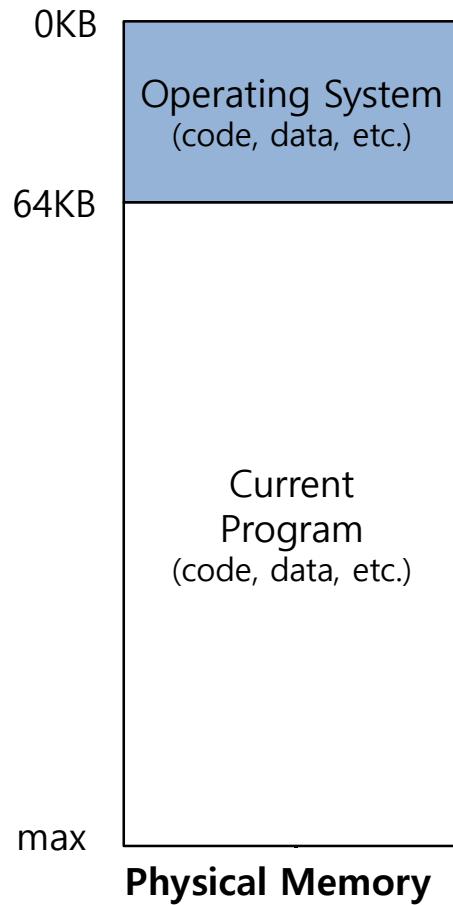


# Benefit of Memory Virtualization

- ▣ Ease of use in programming
- ▣ Memory efficiency in terms of **times** and **space**
- ▣ The guarantee of isolation for processes as well as OS
  - ◆ Protection from **errant accesses** of other processes

# OS in The Early System

- Load only one process in memory.
  - ◆ Poor utilization and efficiency



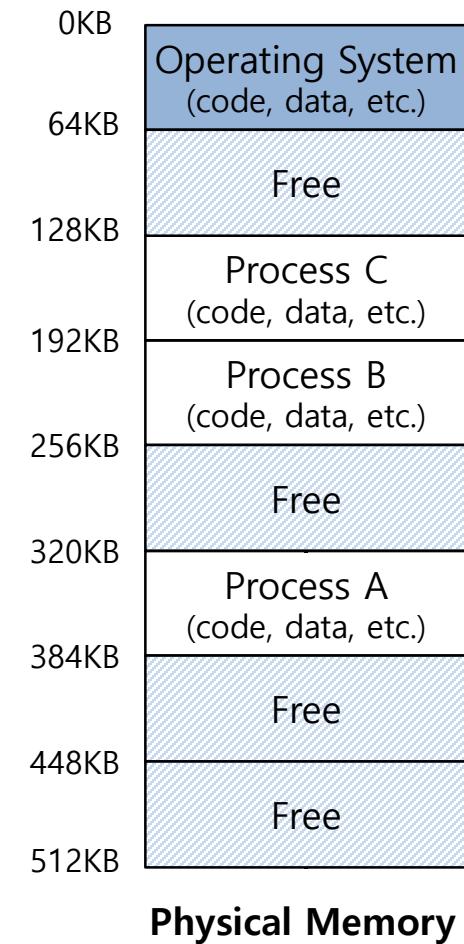
# Multiprogramming and Time Sharing

- Load multiple processes in memory.

- Execute one for a short while.
- Switch processes between them in memory.
- Increase utilization and efficiency.

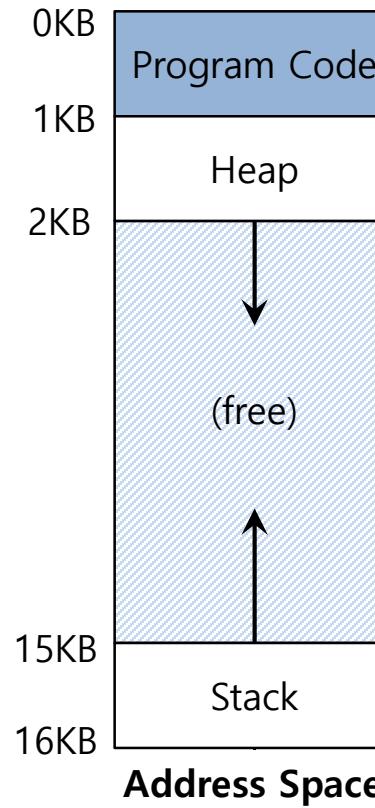
- Cause an important **protection issue**.

- Errant memory accesses from other processes



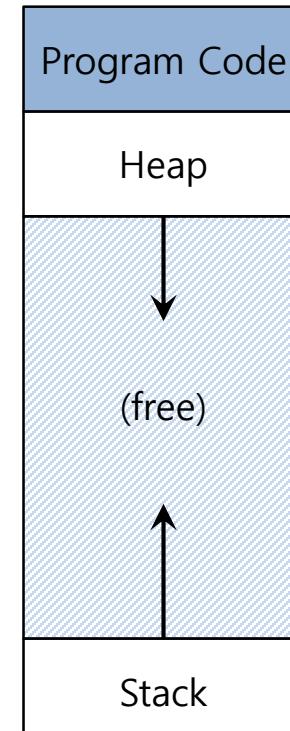
# Address Space

- OS creates an **abstraction** of physical memory.
  - The address space contains all about a running process.
  - That is consist of program code, heap, stack and etc.



# Address Space(Cont.)

- Code
  - ◆ Where instructions live
- Heap
  - ◆ Dynamically allocate memory.
    - malloc in C language
    - new in object-oriented language
- Stack
  - ◆ Store return addresses or values.
  - ◆ Contain local variables arguments to routines.



Address Space

# Virtual Address

- ▣ **Every address** in a running program is virtual.
  - ◆ OS translates the virtual address to physical address

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {

    printf("location of code : %p\n", (void *) main);
    printf("location of heap : %p\n", (void *) malloc(1));
    int x = 3;
    printf("location of stack : %p\n", (void *) &x);

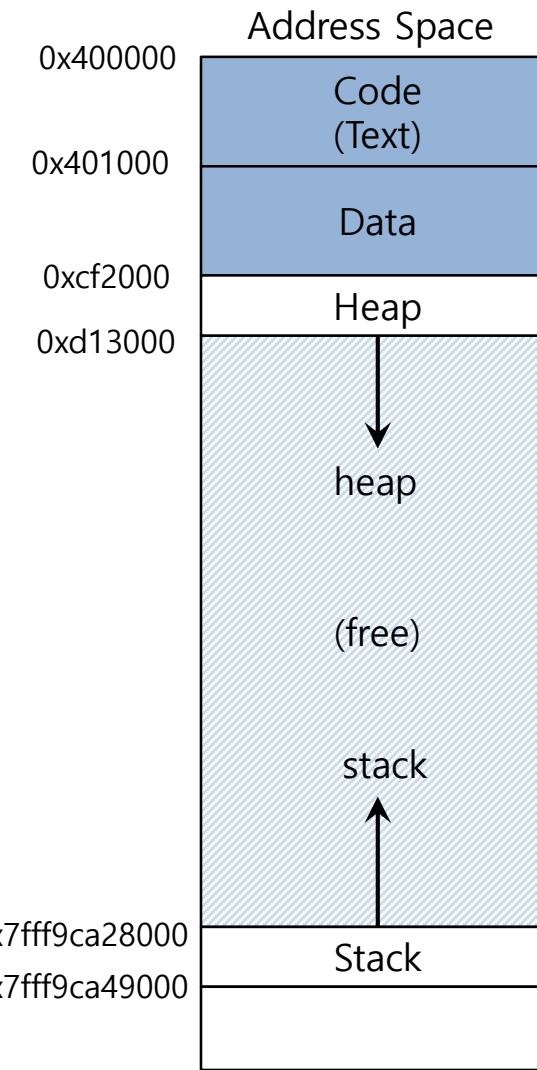
    return x;
}
```

A simple program that prints out addresses

# Virtual Address(Cont.)

- The output in 64-bit Linux machine

```
location of code : 0x40057d  
location of heap : 0xcf2010  
location of stack : 0x7fff9ca45fcc
```



# Components of Virtual Address Space

```
#include <stdio.h>
#include <stdlib.h>

int InitializedGlobal[1024] = {0,};
int UnintGlobal[1024];

int main() {
    int localVar1;
    int localVar2;
    int *dynamicLocalVar1;
    int *dynamicLocalVar2;

    dynamicLocalVar1 = malloc(sizeof(int));
    dynamicLocalVar2 = malloc(sizeof(int));

    printf("code              : 0x%x\n", main);
    printf("Data             : 0x%x\n", &InitializedGlobal);
    printf("BSS(Uninit Data) : 0x%x\n", &UnintGlobal);

    printf("stack localVar1   : 0x%x\n", &localVar1);
    printf("stack localVar2   : 0x%x\n", &localVar2);
    printf("heap dynamicLocalVar1: 0x%x\n", dynamicLocalVar1);
    printf("heap dynamicLocalVar2: 0x%x\n", dynamicLocalVar2);
    return 0;
}
```

# Components of Virtual Address Space

```
BulGok:~ yjwons$ ./a.out
code : 0x0dffde0
Data : 0xe01020
BSS (Uninit Data) : 0xe02020
stack localVar1 : 0xeee00a88
stack localVar2 : 0xeee00a84
heap dynamicLocalVar1: 0xc4c01700
heap dynamicLocalVar2: 0xc4c01710
```

Minimum heap allocation unit: 16 Byte

