# Operating System: Address Space

Sang Ho Choi (shchoi@kw.ac.kr)
School of Computer & Information Engineering
KwangWoon University

여러개 CPU 가상화

이제 memory 가상화

# 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

각각의 프로세스가
전체 메모리를 사용하는것처럼.

# Memory Virtualization: Goal

- Transparency 투명성
  - Processes should not be aware that memory is shared
  - Provides a convenient abstraction for programming
    - (i.e. a large, contiguous memory space)

프로세스가 메모리가 공유됨을 알면 X

크고 연속적인

- Efficiency 효율성
  - Minimizes fragmentation due to variable-sized requests (space)
  - Gets some hardware support (time)

프로세스의 요청에 따라 다양한 크기의 공간들이 할당되는데 이때 공간과 공간 사이에서 생기는 조각들을 최소화 시켜야함.

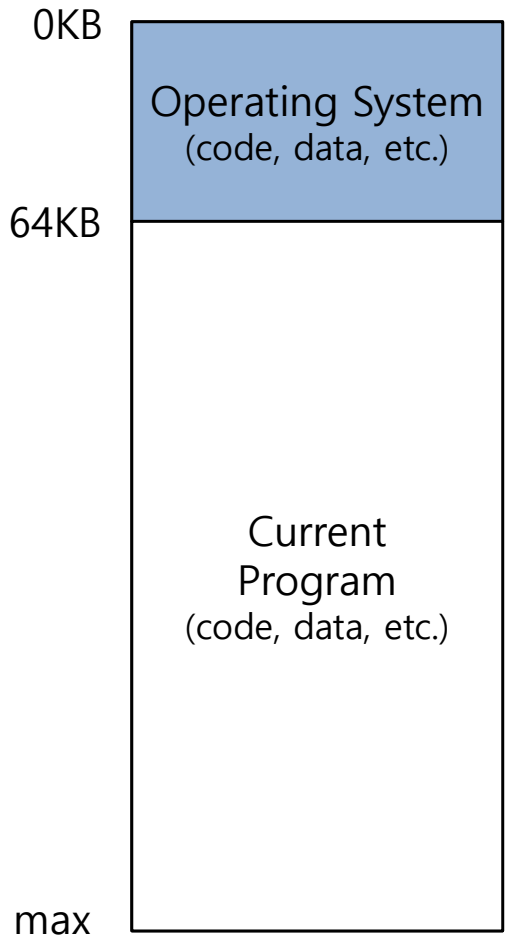
시간적측면에서는 너무느리게 실행되지 않게끔 하드웨어가 도와야함.

- Protection 보안성.
  - Protect processes and the OS from another process
  - Isolation: a process can fail without affecting other processes
  - Cooperating processes can share portions of memory

# OS in The Early System

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

옛날 os에서는 메모리에 그냥 프로그램 하나 올려서 했음 너무 cpu 가용량도 떨어지고 효율성도 떨어짐




**Physical Memory**

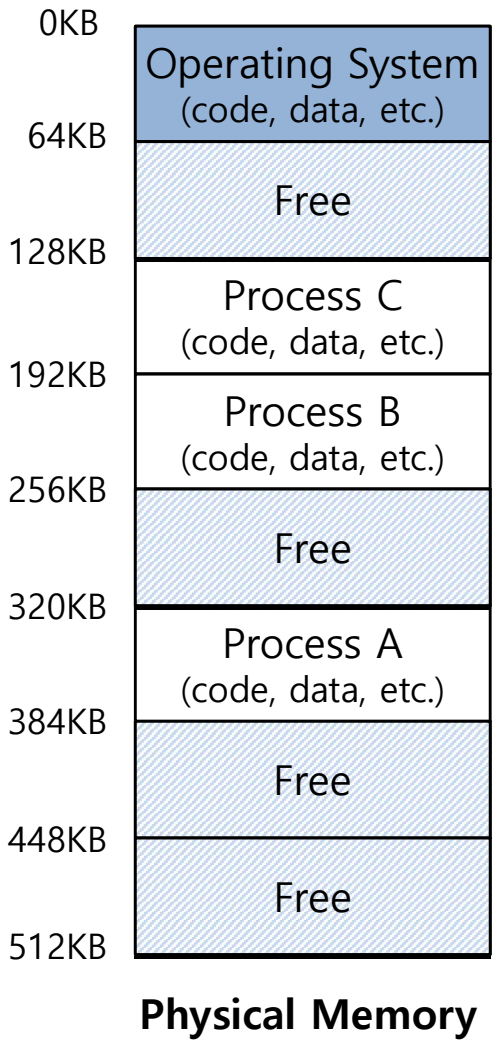
# Multiprogramming and Time Sharing

메모리 관점에서의 가상화 적용을 멀티 프로그래밍 cpu 관점에서의 가상화 적용을 time sharing

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

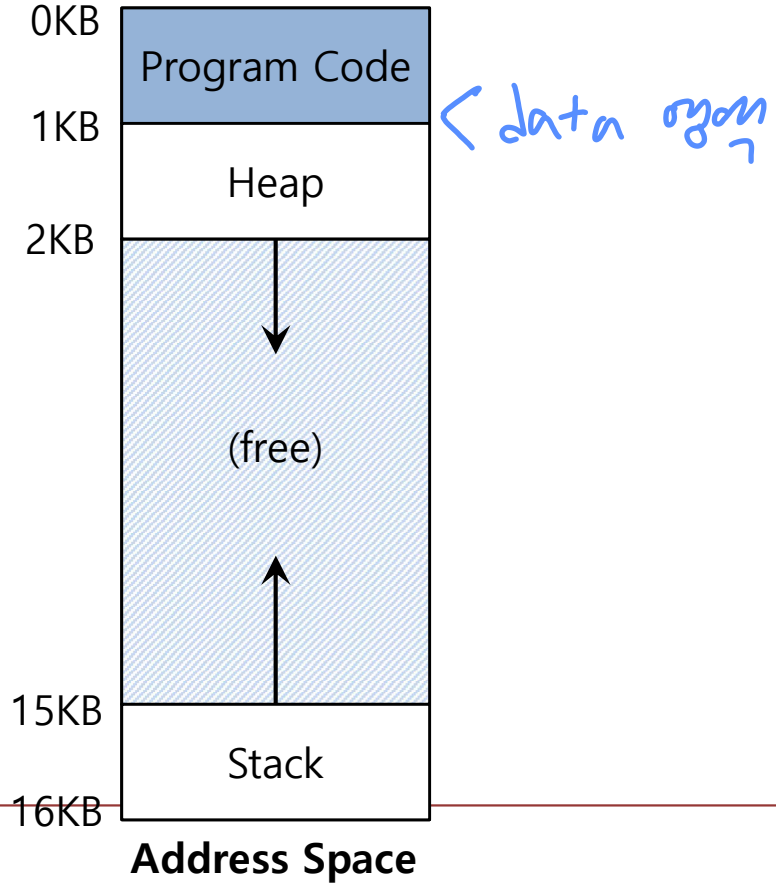
- Cause an important protection issue
  - Errant memory accesses from other processes 각각의 프로그램들이 방해받지 않으려면?

메모리 가상화를 하는 또 다른 이유중 하나는 context switch를 할때 다른 프로그램으로의 전환시 기존의 프로그램을 디스크에 저장하는 것은 너무 오래 걸리고 따라서 메모리에 그대로 두기 위해



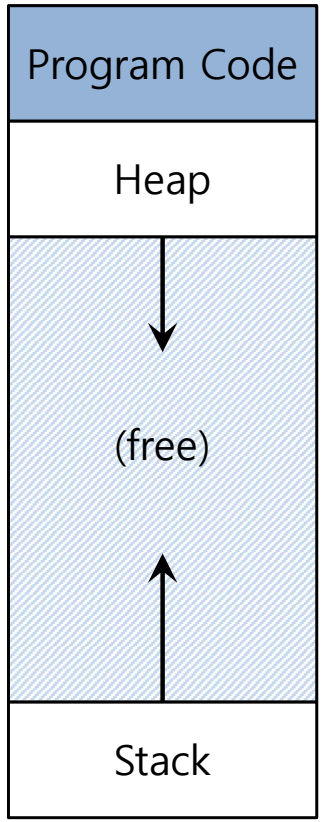
# Address Space

- 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.)

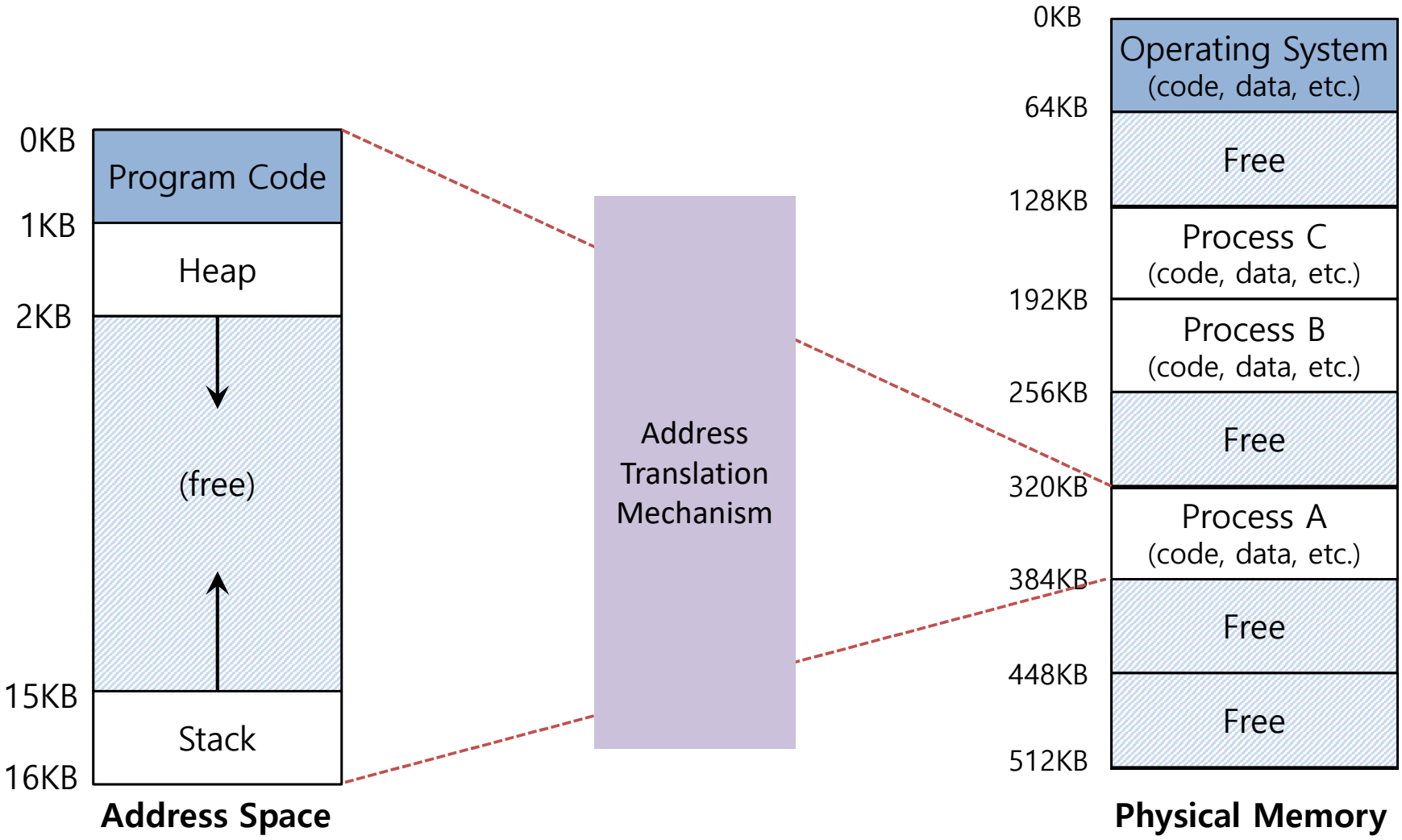
- Code
  - Where instructions live
- Heap
  - Dynamically allocate memory
    - `malloc` in C language
    - `new` in object-oriented language
- Stack
  - Store return addresses or values
  - Contain local variables arguments to routines




**Address Space**

# Address Space (Cont.)

- Memory virtualization



# 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
```

프로세스에서 사용하고 출력하는 모든 주소는 다 virtual address이고 이를 OS에서 address translation하여 physical address를 알아내 접근하고 사용한다. 어떻게 변환되는지는 나중에 강좌에서 다룰 예정