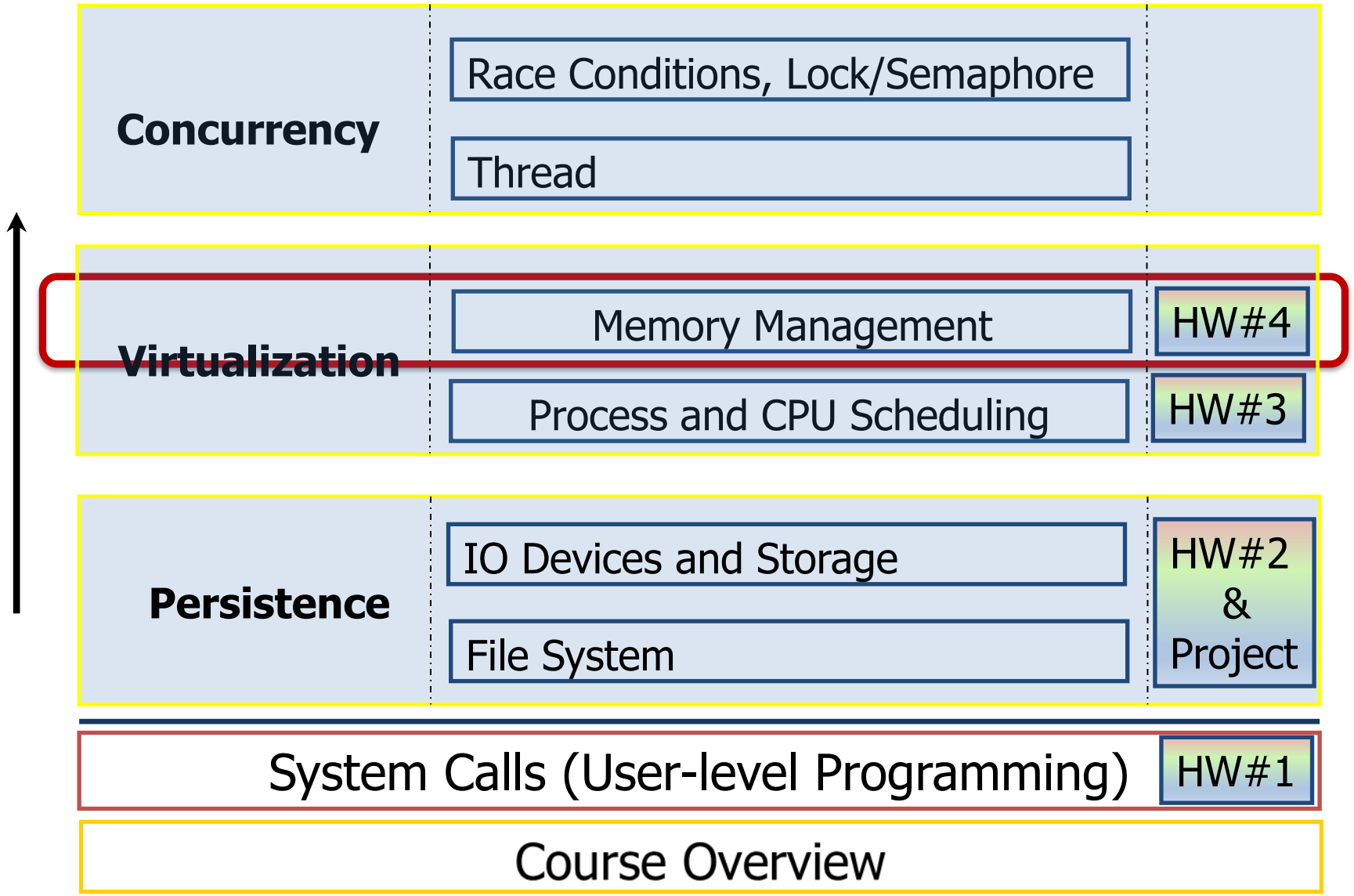
Lecture 9: Virtualizing Memory – Address Space and Address Translation

The Course Organization (Bottom-up)

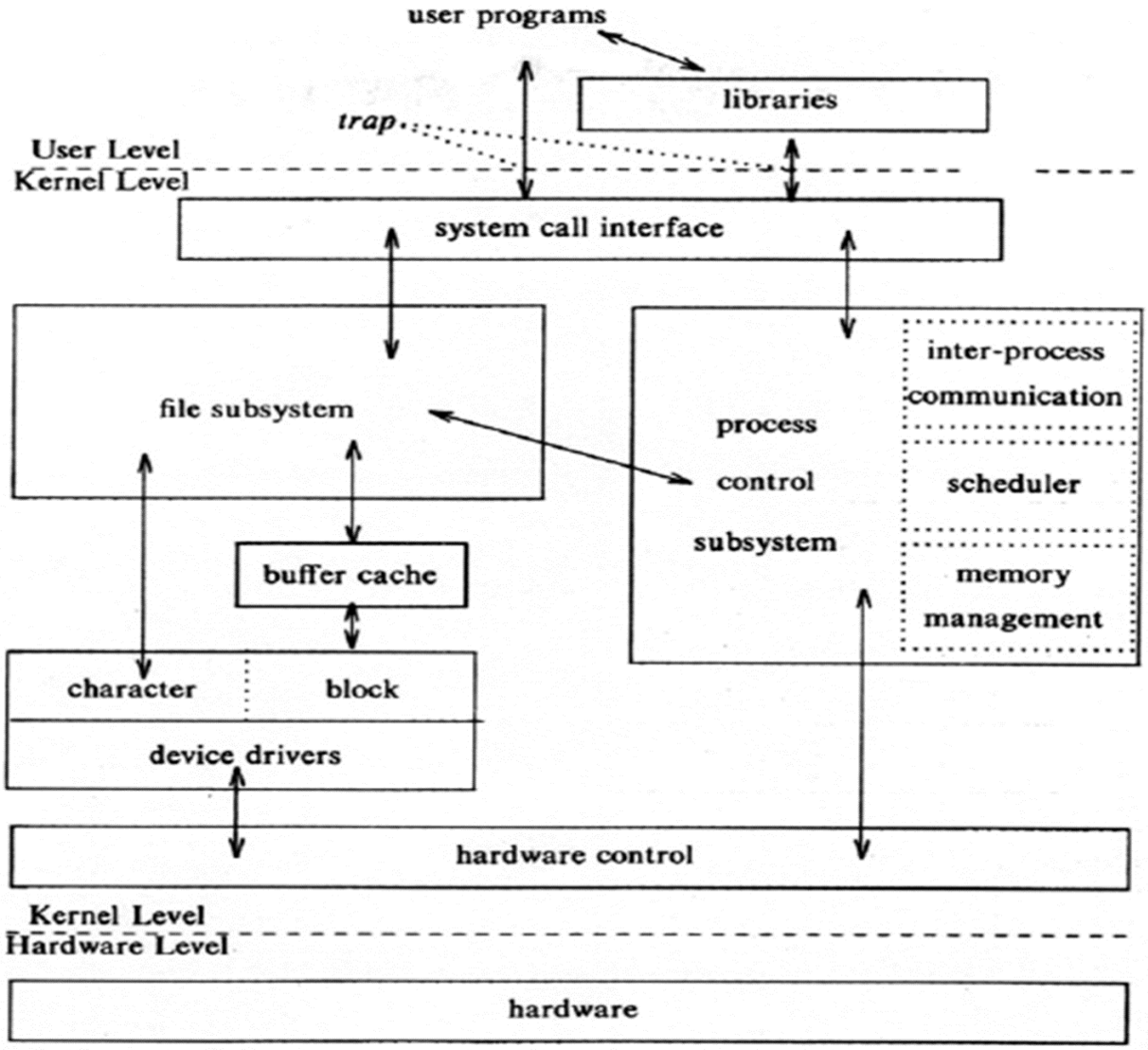


OS – Resource management via virtualization

OS provides services via **System Call** (typically a few hundred) to run **process**, access memory/devices/files, etc.

The OS **manages resources** such as *CPU*, *memory* and *disk* via **virtualization**.

- many programs to run (processes) → Sharing the CPU
- many processes to *concurrently* access their own instructions and data → Sharing memory
- many processes to access devices → Sharing disks



The Design Of The Unix Operating System (Maurice Bach, 1986)

Part I: Address Space

Memory Virtualization

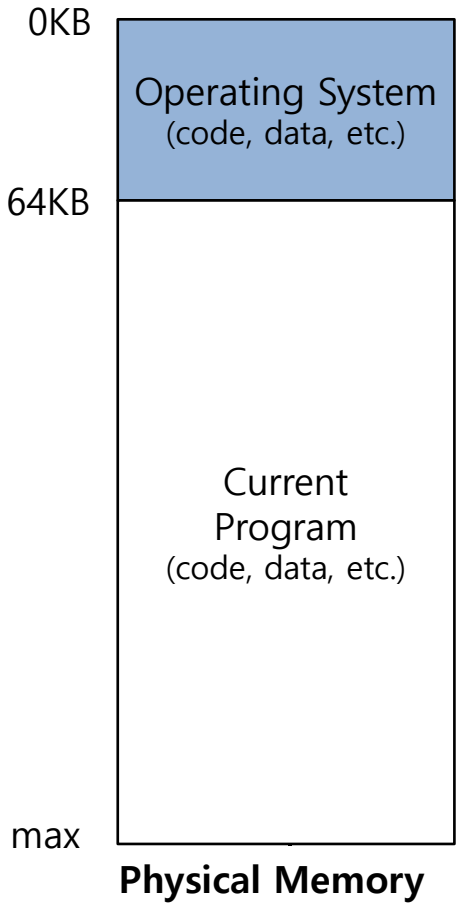
- What is **memory virtualization**?
  - OS virtualizes its physical memory.
  - OS provides an illusion memory space per each process.
  - It seems to be seen like each process uses the whole memory .

Benefit of Memory Virtualization

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

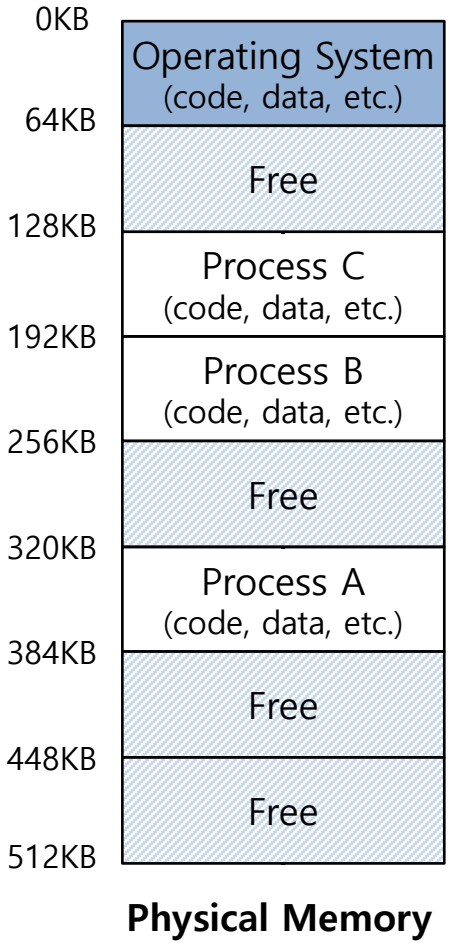
OS in The Early System

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



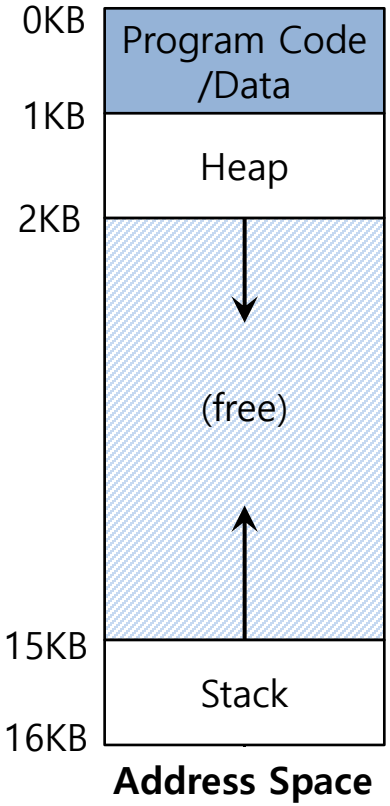
Multiprogramming and Time Sharing

- **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



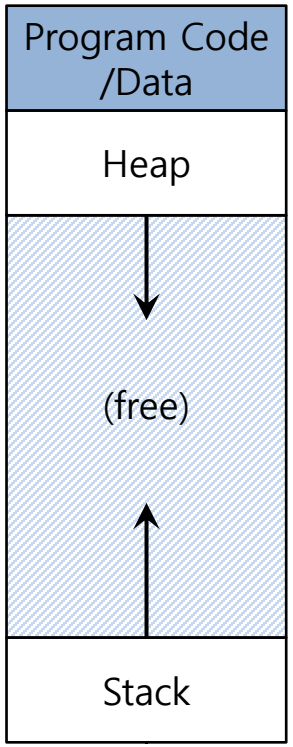
Address Space

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



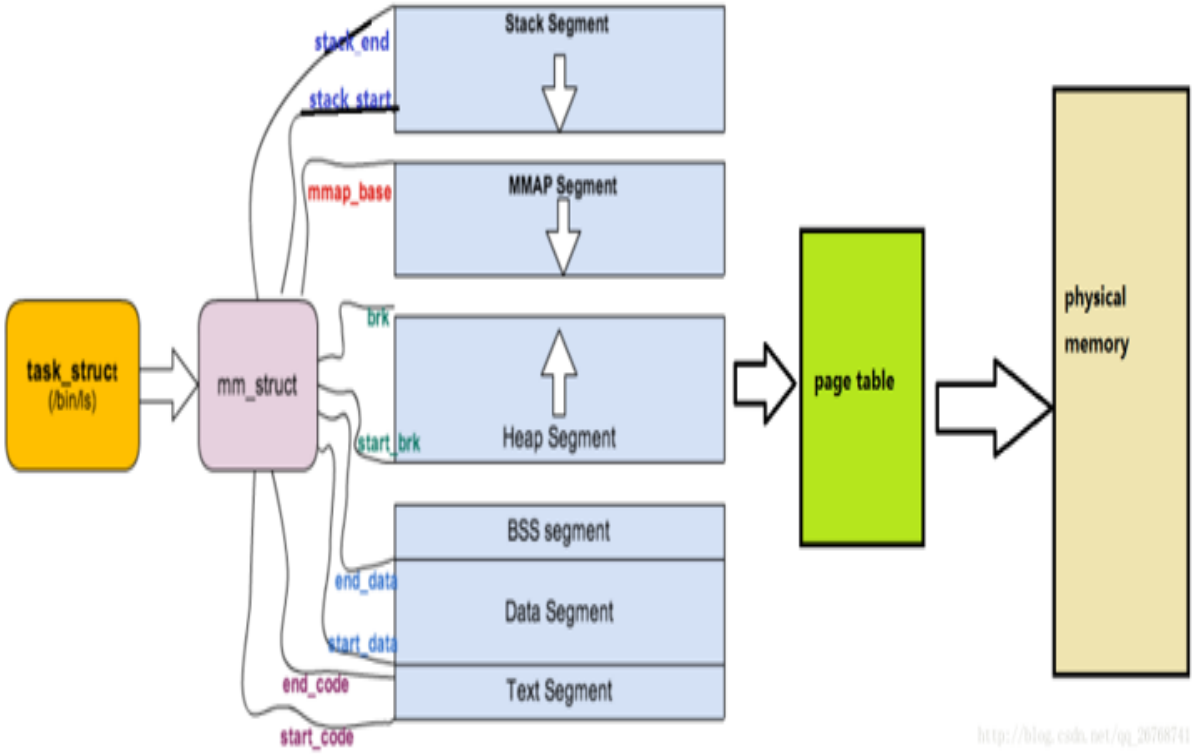
Address Space(Cont.)

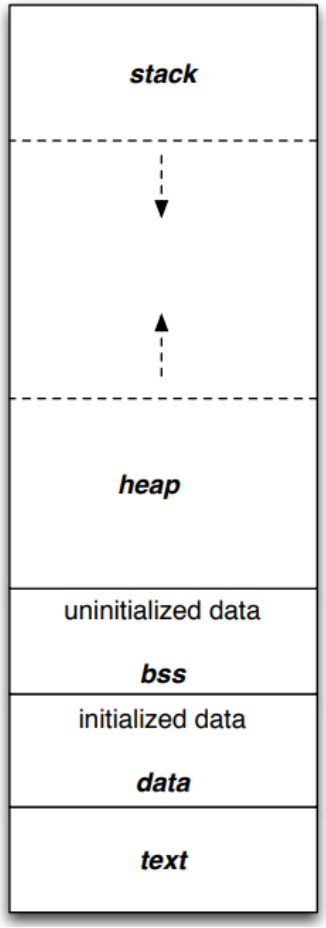
- Code
  - Where instructions live
- Data
  - Global or static local variables
- Heap
  - Dynamically allocate memory.
    - `malloc` in C language
    - `new` in object-oriented language
- Stack
  - Store return addresses or values.
  - Contain local variables arguments to routines.



Address Space

Example from Linux





Virtual Address

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

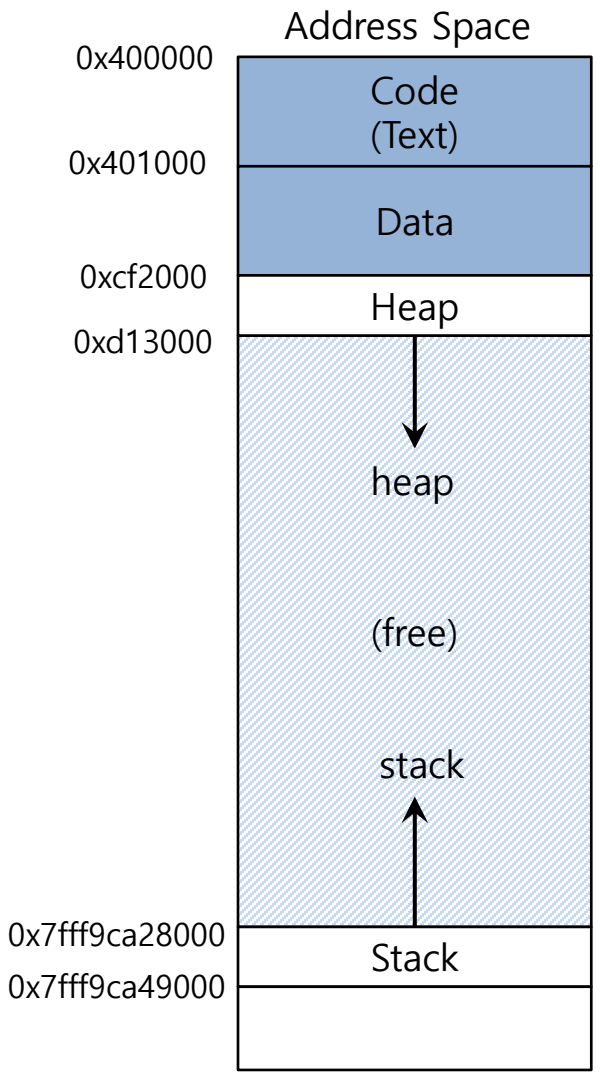
```
#include <stdio.h>
#include <stdlib.h>

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

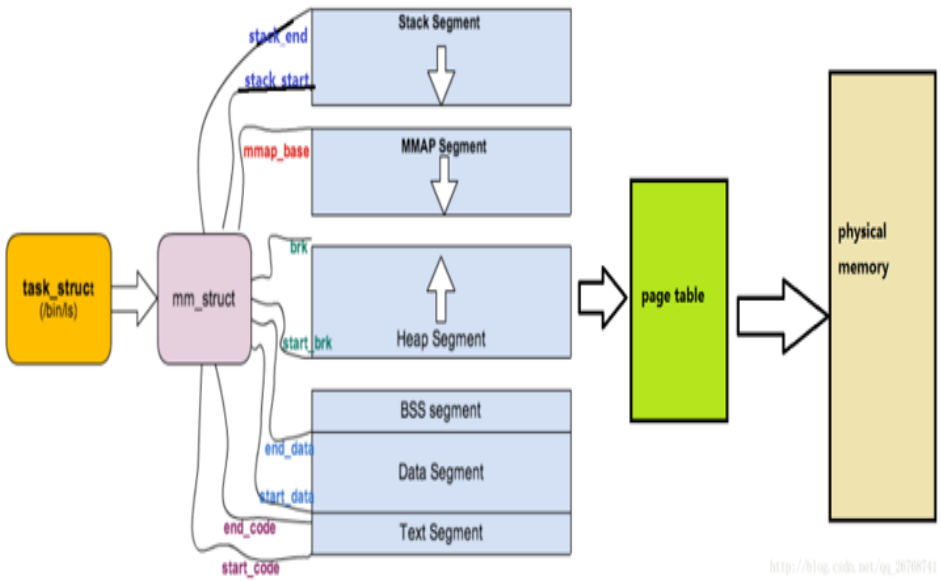
```
location of code  : 0x40057d
location of heap  : 0xcf2010
location of stack : 0x7fff9ca45fcc
```



Example: pmap and cat /proc/<pid>/maps

cat /proc/<pid_number>/maps

```
csci3150@csci3150-VirtualBox:~/process-memory-test$ more /proc/10534/maps
004cd000-004ce000 r-xp 00000000 08:01 136307     /home/csci3150/process-memory-test/test
004ce000-004cf000 r--p 00000000 08:01 136307     /home/csci3150/process-memory-test/test
004cf000-004d0000 rw-p 00001000 08:01 136307     /home/csci3150/process-memory-test/test
b7d93000-b7f68000 r-xp 00000000 08:01 526179     /lib/i386-linux-gnu/libc-2.27.so
b7f68000-b7f69000 ---p 001d5000 08:01 526179     /lib/i386-linux-gnu/libc-2.27.so
b7f69000-b7f6b000 r--p 001d5000 08:01 526179     /lib/i386-linux-gnu/libc-2.27.so
b7f6b000-b7f6c000 rw-p 001d7000 08:01 526179     /lib/i386-linux-gnu/libc-2.27.so
b7f6c000-b7f6f000 rw-p 00000000 00:00 0
b7f83000-b7f85000 rw-p 00000000 00:00 0
b7f85000-b7f88000 r--p 00000000 00:00 0          [vvar]
b7f88000-b7f8a000 r-xp 00000000 00:00 0          [vdso]
b7f8a000-b7fb0000 r-xp 00000000 08:01 526151     /lib/i386-linux-gnu/ld-2.27.so
b7fb0000-b7fb1000 r--p 00025000 08:01 526151     /lib/i386-linux-gnu/ld-2.27.so
b7fb1000-b7fb2000 rw-p 00026000 08:01 526151     /lib/i386-linux-gnu/ld-2.27.so
bf829000-bf84a000 rw-p 00000000 00:00 0          [stack]
```



pmap -x <pid_number>

```
csci3150@csci3150-VirtualBox:~/process-memory-test$ pmap -x 10534
10534:   ./test
Address   Kbytes     RSS   Dirty Mode  Mapping
004cd000       4       4       0 r-x-- test
004cd000       0       0       0 r-x-- test
004ce000       4       4       4 r---- test
004ce000       0       0       0 r---- test
004cf000       4       4       4 rw--- test
004cf000       0       0       0 rw--- test
b7d93000    1876     688       0 r-x-- libc-2.27.so
b7d93000       0       0       0 r-x-- libc-2.27.so
b7f68000       4       0       0 ----- libc-2.27.so
b7f68000       0       0       0 ----- libc-2.27.so
b7f69000       8       8       8 r---- libc-2.27.so
b7f69000       0       0       0 r---- libc-2.27.so
b7f6b000       4       4       4 rw--- libc-2.27.so
b7f6b000       0       0       0 rw--- libc-2.27.so
b7f6c000      12       8       8 rw---   [ anon ]
b7f6c000       0       0       0 rw---   [ anon ]
b7f83000       8       8       8 rw---   [ anon ]
b7f83000       0       0       0 rw---   [ anon ]
b7f85000      12       0       0 r----   [ anon ]
b7f85000       0       0       0 r----   [ anon ]
b7f88000       8       4       0 r-x--   [ anon ]
b7f88000       0       0       0 r-x--   [ anon ]
b7f8a000     152     152       0 r-x-- ld-2.27.so
b7f8a000       0       0       0 r-x-- ld-2.27.so
b7fb0000       4       4       4 r---- ld-2.27.so
b7fb0000       0       0       0 r---- ld-2.27.so
b7fb1000       4       4       4 rw--- ld-2.27.so
b7fb1000       0       0       0 rw--- ld-2.27.so
bf829000     132      12      12 rw---   [ stack ]
bf829000       0       0       0 rw---   [ stack ]
-------- ------- ------- -------
total kB    2236     904      56
```

Part II: Address Translation

Memory Virtualizing with Efficiency and Control

- Memory virtualizing takes a similar strategy known as **limited direct execution(LDE)** for efficiency and control.
- In memory virtualizing, efficiency and control are attained by hardware support.
  - e.g., registers, TLB(Translation Look-aside Buffer)s, page-table

Address Translation

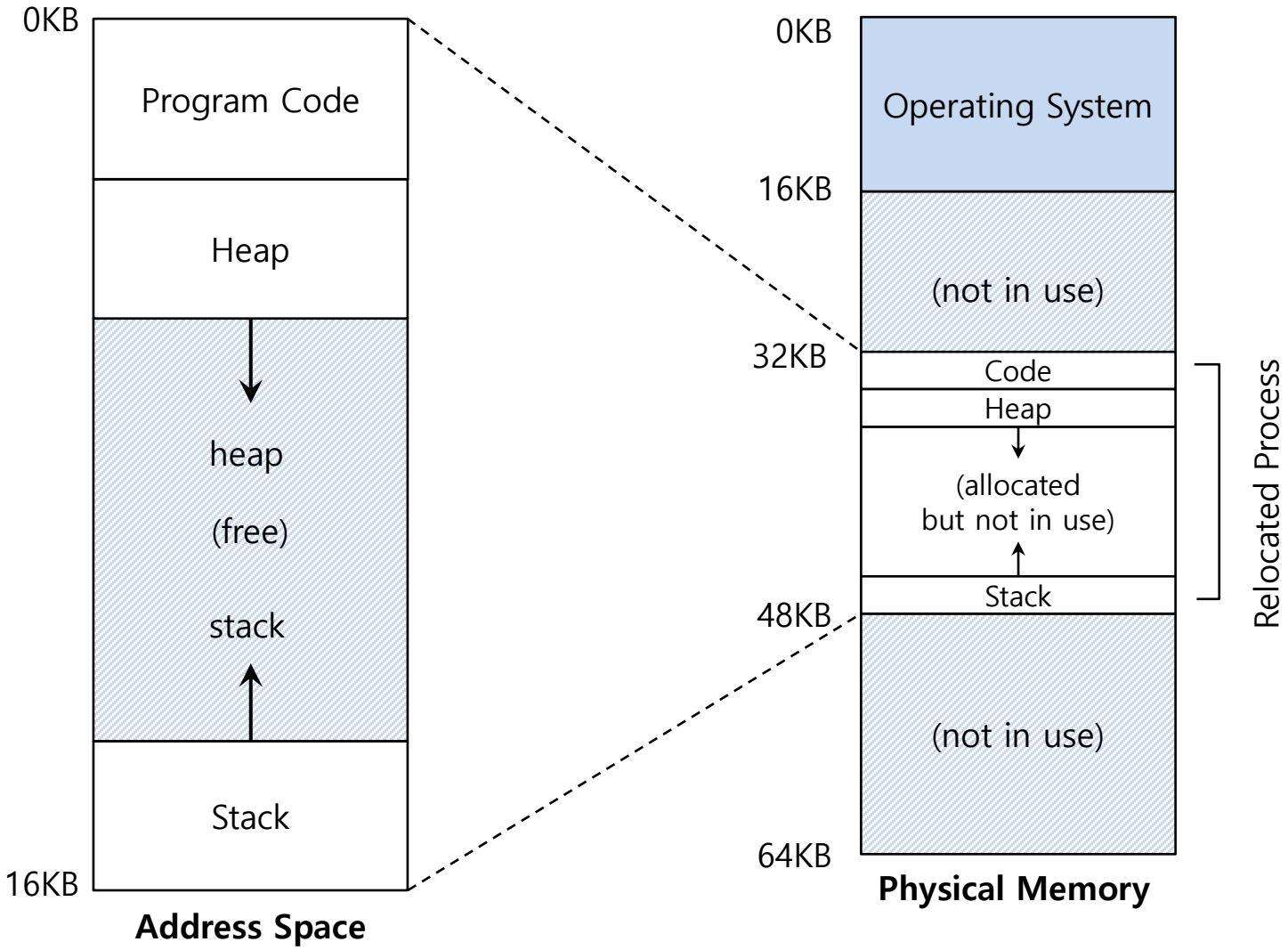
- Hardware transforms a **virtual address** to a **physical address**.
  - The desired information is actually stored in a physical address.

- The OS must get involved at key points to set up the hardware.
  - The OS must manage memory to judiciously intervene.

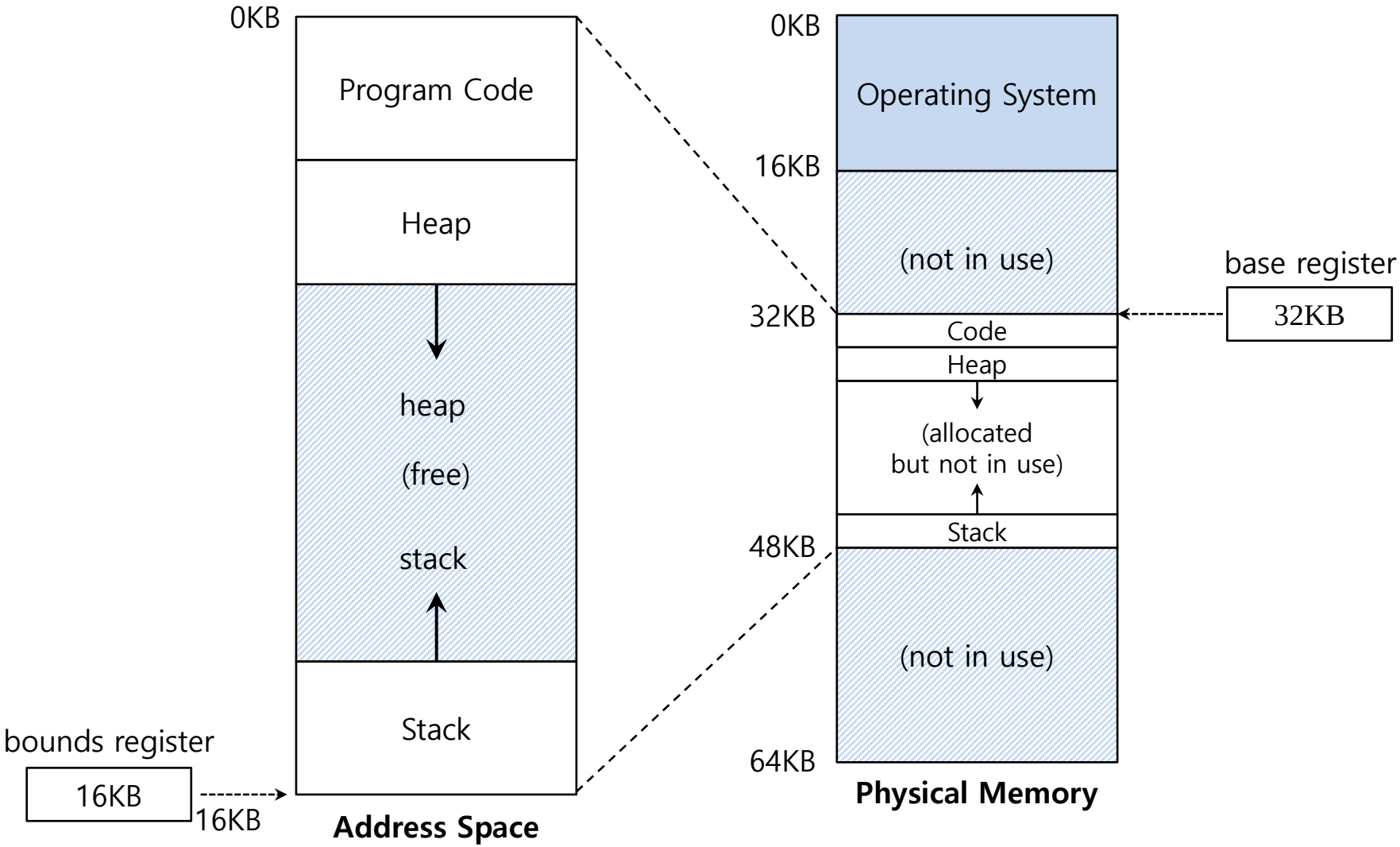
Relocation Address Space

- The address space start at address 0.
- The OS wants to place the process **somewhere else** in physical memory, not at address 0.

A Single Relocated Process



Base and Bounds Register



Dynamic(Hardware base) Relocation

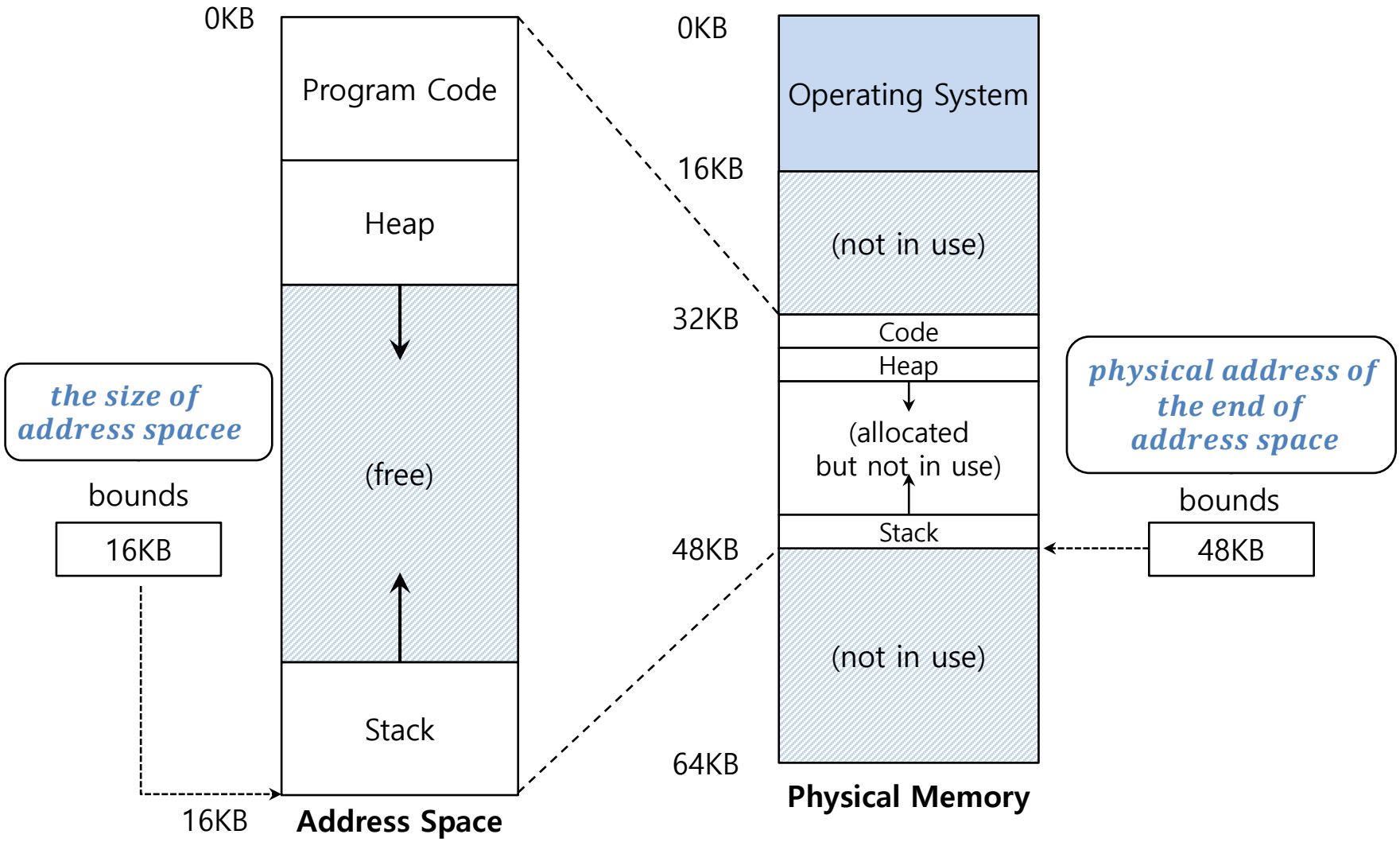
- When a program starts running, the OS decides **where** in physical memory a process should be **loaded**.
  - Set the **base** register a value.

$$phycal\ address = virtual\ address + base$$

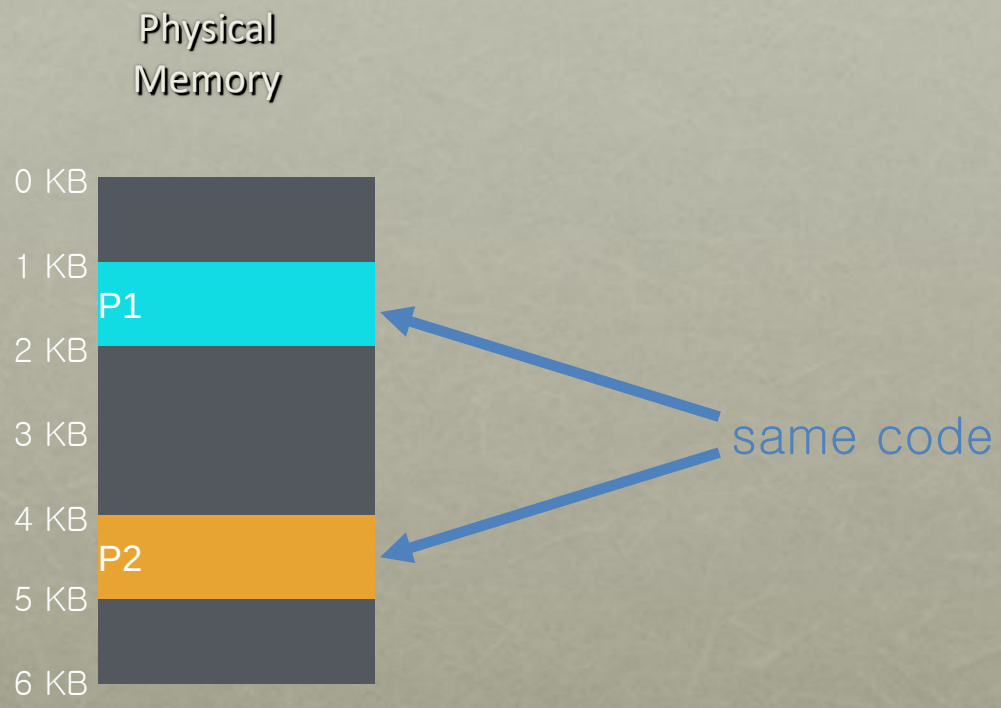
  - Every virtual address must **not be greater than bound** and **negative.**

$$0 \leq\ virtual\ addressvirtual\ address < bounds$$

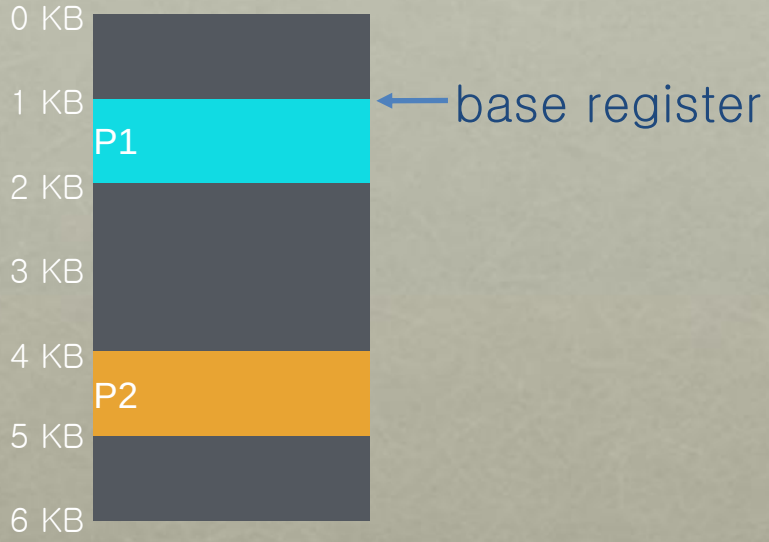
Two ways of Bounds Register



Base Register

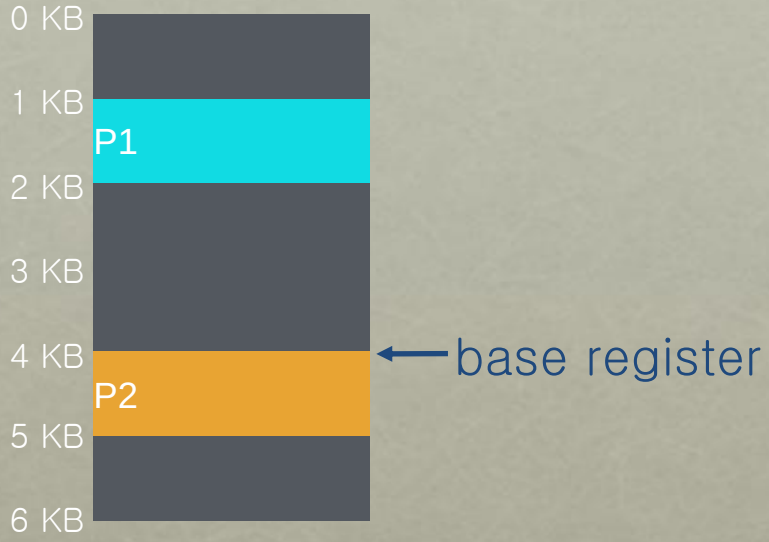


Base Register



P1 is running

Base Register



P2 is running

Base Register

(Decimal notation)

Virtual	Physical
P1: load 100, R1	

Base Register

Virtual	Physical	
P1: load 100, R1	load 1124, R1	(1024 + 100)

Base Register

Virtual	Physical
P1: load 100, R1	load 1124, R1
P2: load 100, R1	

Base Register

Virtual	Physical
P1: load 100, R1	load 1124, R1
P2: load 100, R1	load 4196, R1

(4096 + 100)

Base Register

Virtual	Physical
P1: load 100, R1	load 1124, R1
P2: load 100, R1	load 4196, R1
P2: load 1000, R1	

Base Register

Virtual	Physical
P1: load 100, R1	load 1124, R1
P2: load 100, R1	load 4196, R1
P2: load 1000, R1	load 5096, R1

Base Register

Virtual	Physical
P1: load 100, R1	load 1124, R1
P2: load 100, R1	load 4196, R1
P2: load 1000, R1	load 5096, R1
P1: load 1000, R1	

Base Register

Virtual	Physical
P1: load 100, R1	load 1124, R1
P2: load 100, R1	load 4196, R1
P2: load 1000, R1	load 5096, R1
P1: load 1000, R1	load 2024, R1

Base Register

Virtual	Physical
P1: load 100, R1	load 1124, R1
P2: load 100, R1	load 4196, R1
P2: load 1000, R1	load 5096, R1
P1: load 1000, R1	load 2024, R1

Can P2 hurt P1?
Can P1 hurt P2?

How to protect process with base register?

Base Register

Virtual	Physical	
P1: load 100, R1	load 1124, R1	
P2: load 100, R1	load 4196, R1	
P2: load 1000, R1	load 5096, R1	
P1: load 1000, R1	load 2024, R1	
P1: store 3072, R1	store 4096, R1	(3072 + 1024)

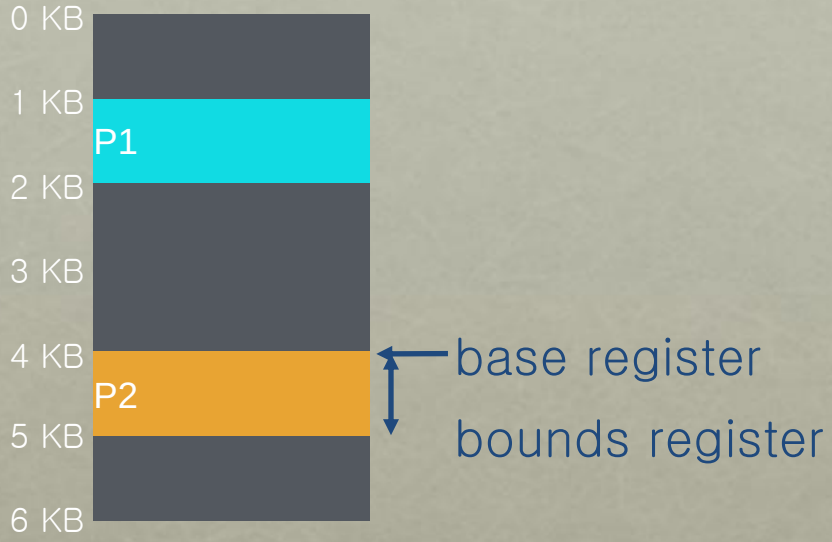
Can P2 hurt P1?
Can P1 hurt P2?

How to protect process with base register?

Base + Bounds


0 KB
1 KB
P1
2 KB
3 KB
4 KB
P2
5 KB
6 KB
base register
bounds register
P1 is running

Base + Bounds



P2 is running

Base + Bounds

Virtual	Physical
P1: load 100, R1	load 1124, R1
P2: load 100, R1	load 4196, R1
P2: load 1000, R1	load 5096, R1
P1: load 1000, R1	load 2024, R1
P1: store 3072, R1	

Can P1 hurt P2?

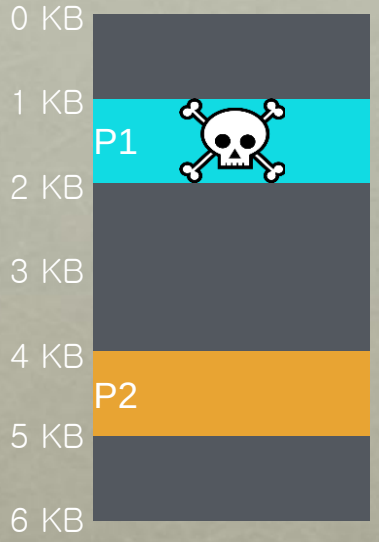
Base + Bounds

Virtual	Physical
P1: load 100, R1	load 1124, R1
P2: load 100, R1	load 4196, R1
P2: load 1000, R1	load 5096, R1
P1: load 1000, R1	load 2024, R1
P1: store 3072, R1	Exception

3072 > 1024

Can P1 hurt P2?

Base + Bounds



Virtual	Physical
P1: load 100, R1	load 1124, R1
P2: load 100, R1	load 4196, R1
P2: load 1000, R1	load 5096, R1
P1: load 1000, R1	load 2024, R1
P1: store 3072, R1	Exception

3072 > 1024

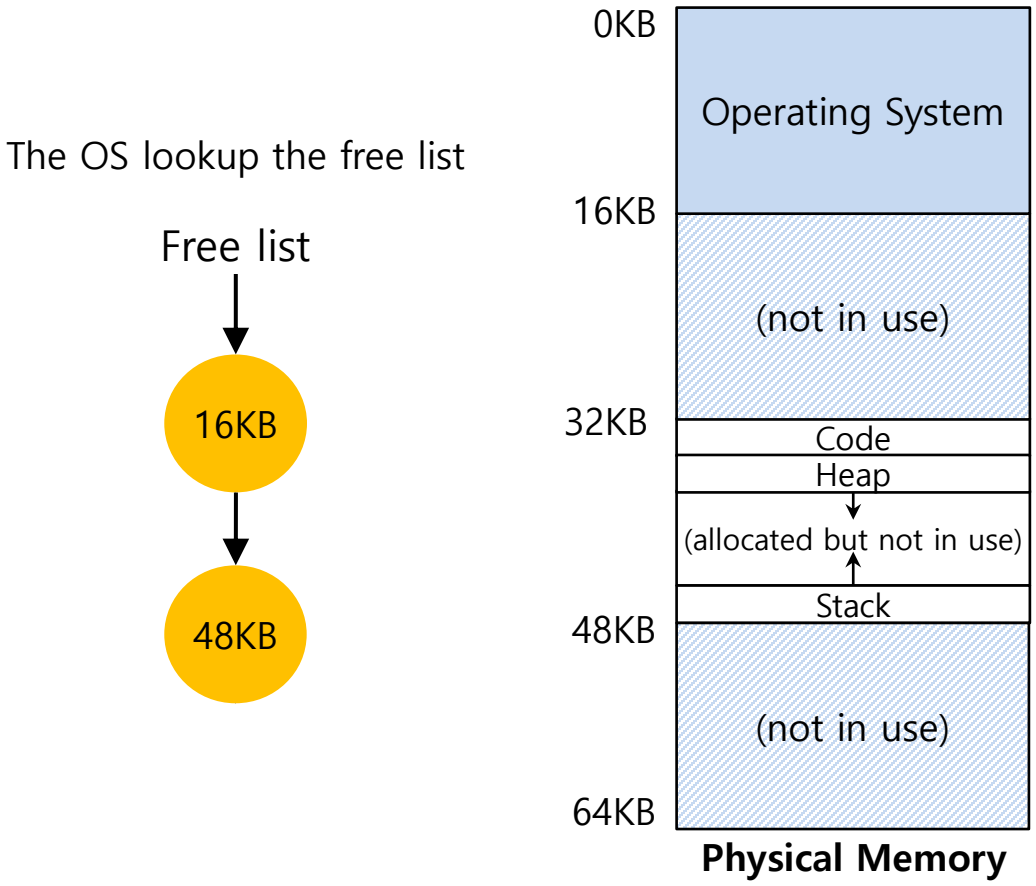
Can P1 hurt P2?

OS Issues for Memory Virtualizing

- The OS must **take action** to implement **base-and-bounds** approach.
- Three critical junctures:
  - When a process **starts running:**
    - Finding space for address space in physical memory
  - When a process is **terminated:**
    - Reclaiming the memory for use
  - When context **switch occurs:**
    - Saving and storing the base-and-bounds pair

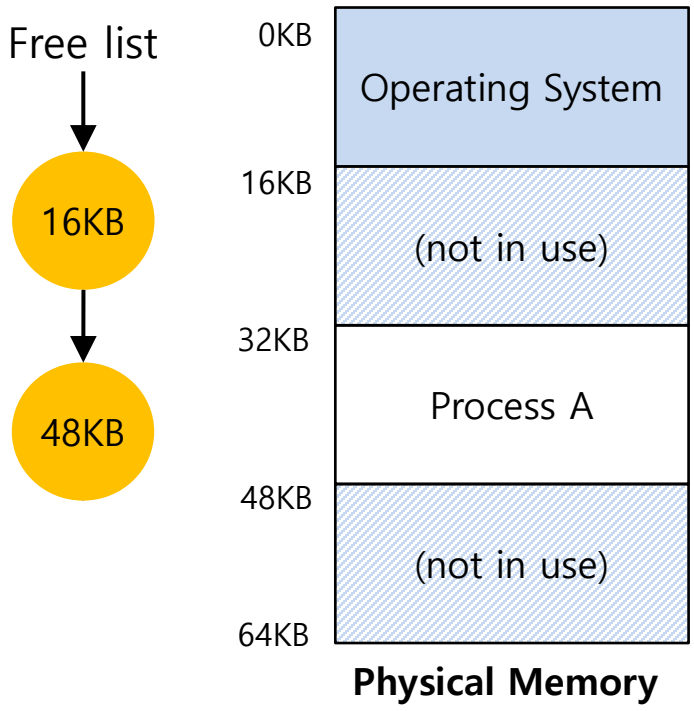
OS Issues: When a Process Starts Running

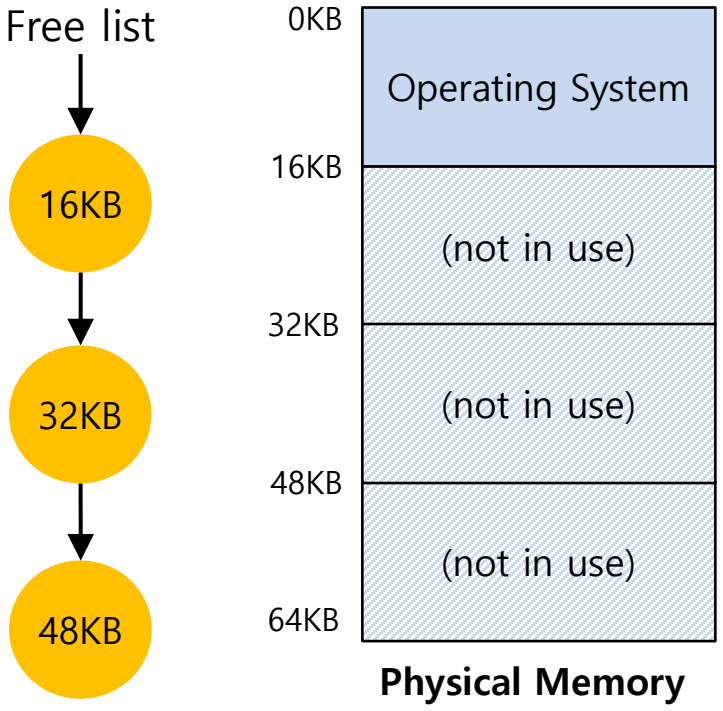
- The OS must **find a room** for a new address space.
  - free list : A list of the range of the physical memory which are not in use.



OS Issues: When a Process Is Terminated

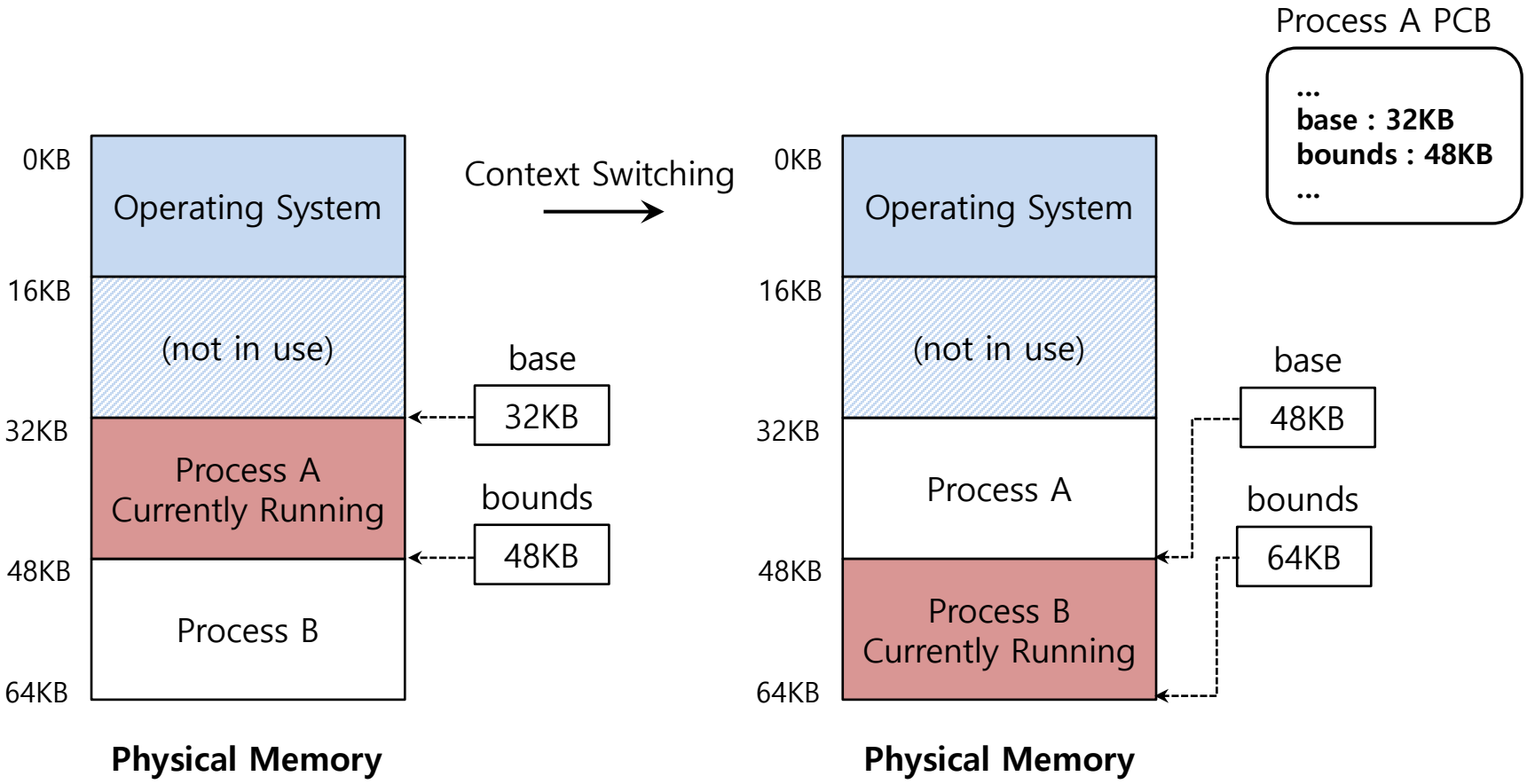
- The OS must **put the memory back** on the free list.





OS Issues: When Context Switch Occurs

- The OS must **save and restore** the base-and-bounds pair.
  - In **process structure** or **process control block(**PCB)



Summary

- Address Space – an **abstraction** of physical memory created by OS
  - The address space contains all about a running process.
  - That is consist of program code, data, heap, and stack
- Address Translation – Hardware-based Address Translation
  - The hardware transforms each memory access (e.g., an instruction fetch, load, or store), changing the virtual address provided by the instruction to a physical address where the desired information is actually located .
  - OS performs memory management – keeping track of which locations are free and which are in use, and judiciously intervening to maintain control over how memory is used.
  - The Base and Bounds Approach
    - When a program starts running, the OS decides where in physical memory a process should be loaded
    - The hardware checks whether virtual addresses are in bounds.
- Next: Memory Management (Chapters 16, 17, 18, 19, 20, 21, 22, 23)