21CS2109AA

Operating Systems

Session 12

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.

Example: Address Translation

- C - Language code

```
void func()
        int x;
        ...
        x = x + 3; // this is the line of code we are interested in
```

- **Load** a value from memory
- **Increment** it by three
- **Store** the value back into memory

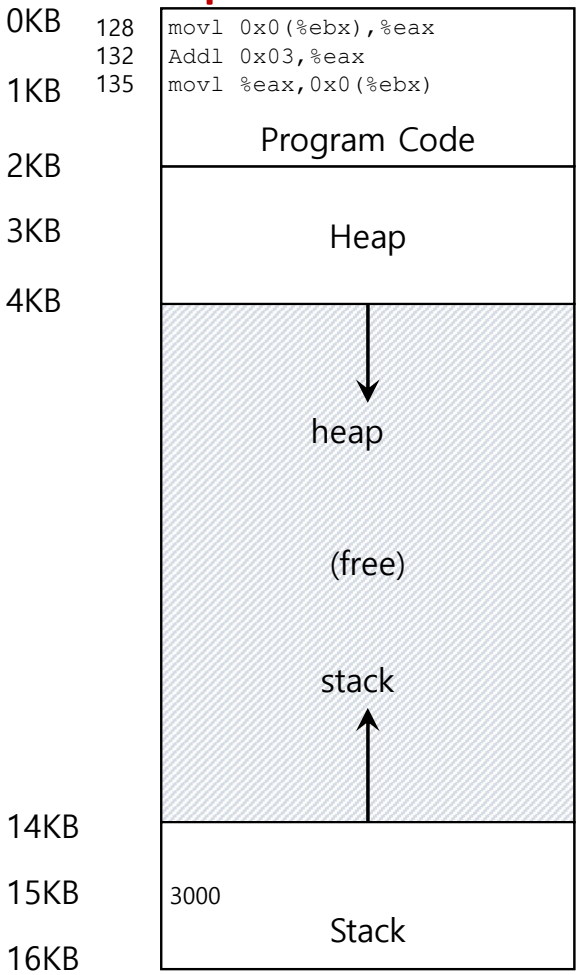
Example: Address Translation(Cont.)

- Assembly

```
128 : movl 0x0(%ebx), %eax        ; load 0+ebx into eax
132 : addl $0x03, %eax            ; add 3 to eax register
135 : movl %eax, 0x0(%ebx)        ; store eax back to mem
```

- Presume that the address of 'x' has been place in ebx register.
- **Load** the value at that address into eax register.
- **Add** 3 to eax register.
- **Store** the value in eax back into memory.

Example: Address Translation(Cont.)

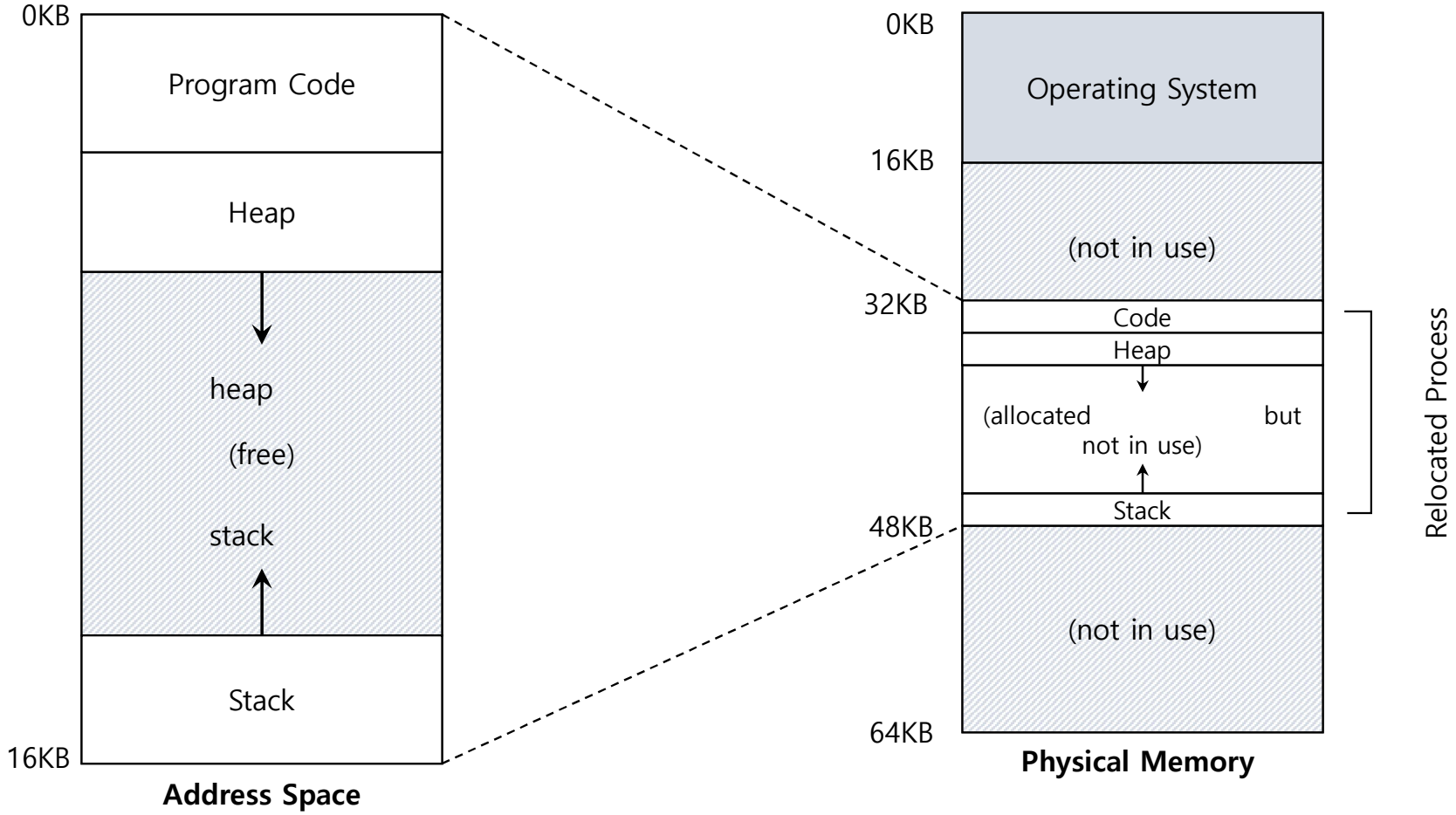


- Fetch instruction at address 128
- Execute this instruction (load from address 15KB)
- Fetch instruction at address 132
- Execute this instruction (no memory reference)
- Fetch the instruction at address 135
- Execute this instruction (store to address 15 KB)

Relocation Address Space

- From the program's perspective, its **address space starts at address 0** and grows to a maximum of 16 KB; all memory references it generates should be within these bounds.
- However, to virtualize memory, the OS wants to place the process somewhere else in physical memory, not necessarily at address 0. Thus, we have the problem: how can we **relocate** this process in memory in a way that is **transparent to the process?**
- **How** can provide the illusion of a virtual address space starting at 0, when in reality the address space is located at some other physical address?

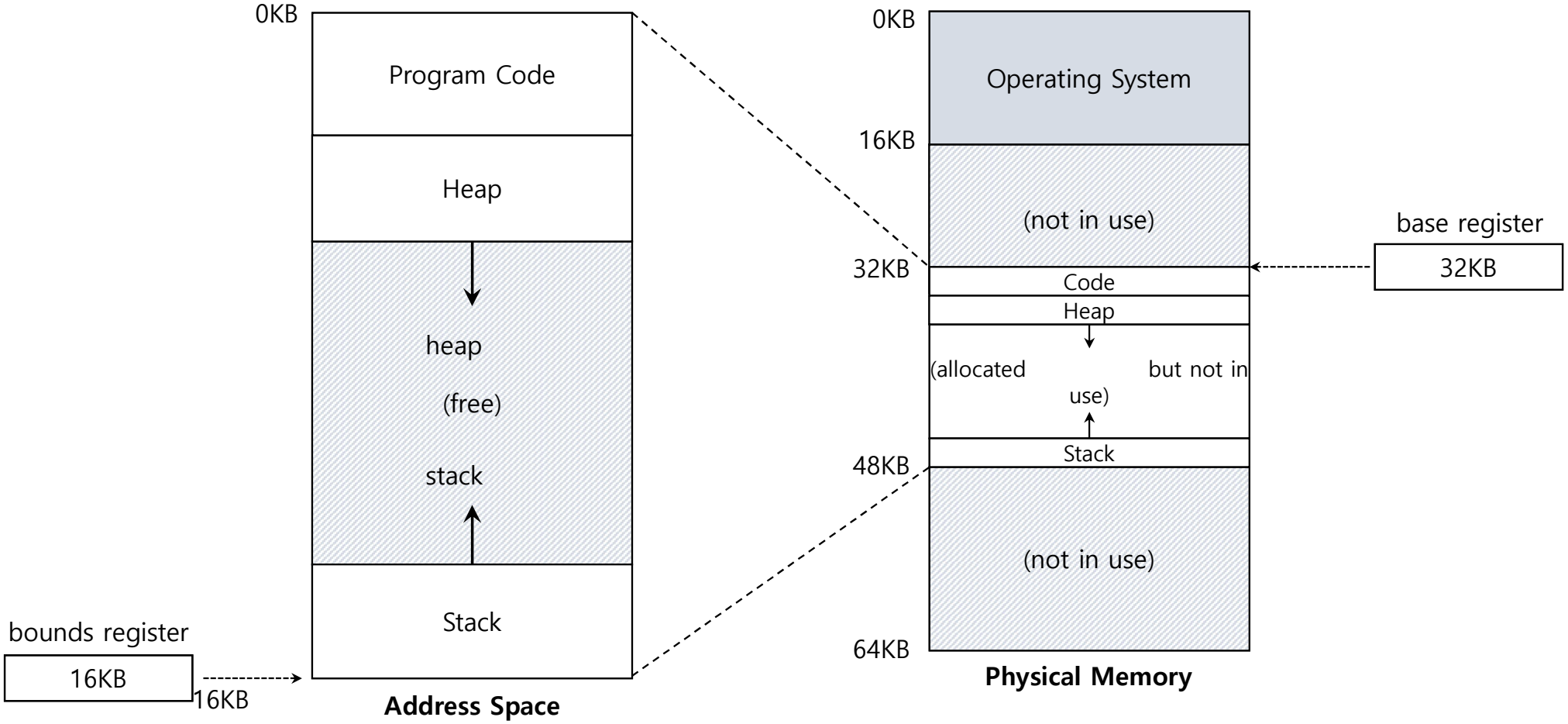
A Single Relocated Process



Base and Bounds Register

- Two hardware registers within each CPU: one is called the base register, and the other the bounds (sometimes called a limit register).
- This base-and-bounds pair is going to allow us to place the address space anywhere we'd like in physical memory, and do so while ensuring that the process can only access its own address space.
- when a program starts running, the OS decides where in physical memory it should be loaded and sets the base register to that value.
- In the example above, the OS decides to load the process at physical address 32 KB and thus sets the base register to this value.

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.

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

  - Each memory reference generated by the process is a **virtual address.** Every virtual address must **not be greater than bound** and **negative.**

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

Relocation and Address Translation

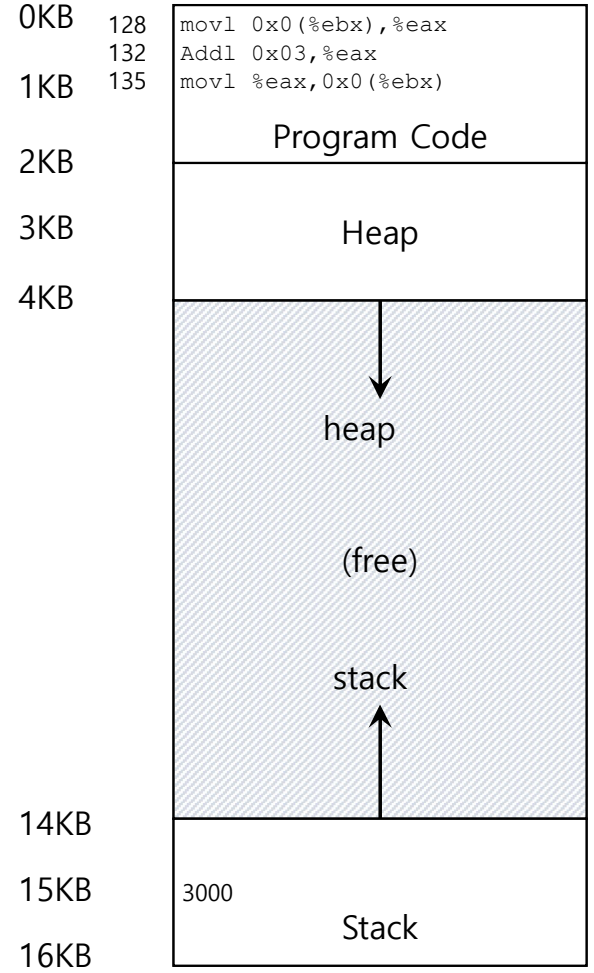
```
128 : movl 0x0(%ebx), %eax
```

- **Fetch** instruction at address 128(32KB=32768)

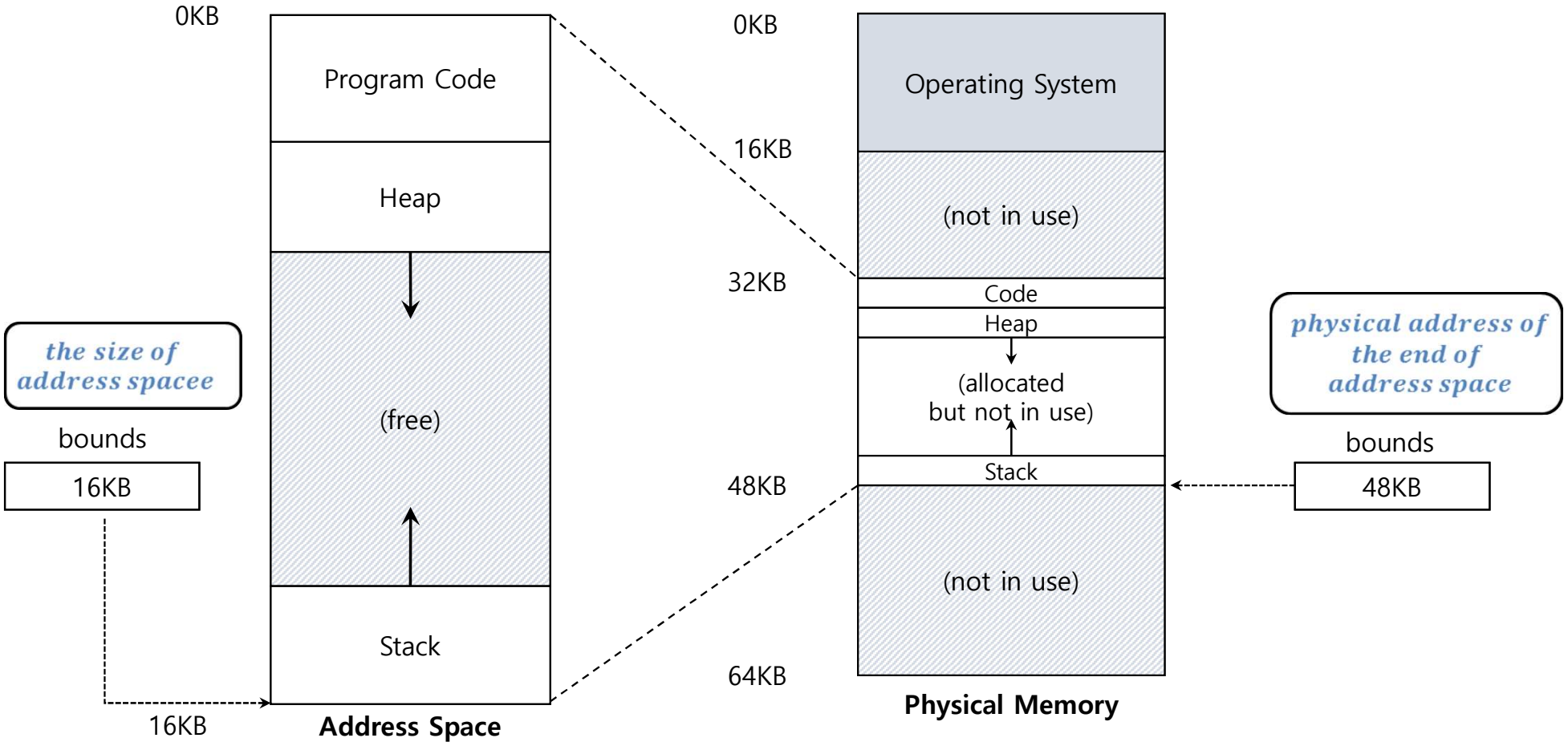
$$32896 = 128 + 32KB(base)$$

- **Execute** this instruction
  - Load from address 15KB

$$47KB = 15KB + 32KB(base)$$



Two ways of Bounds Register

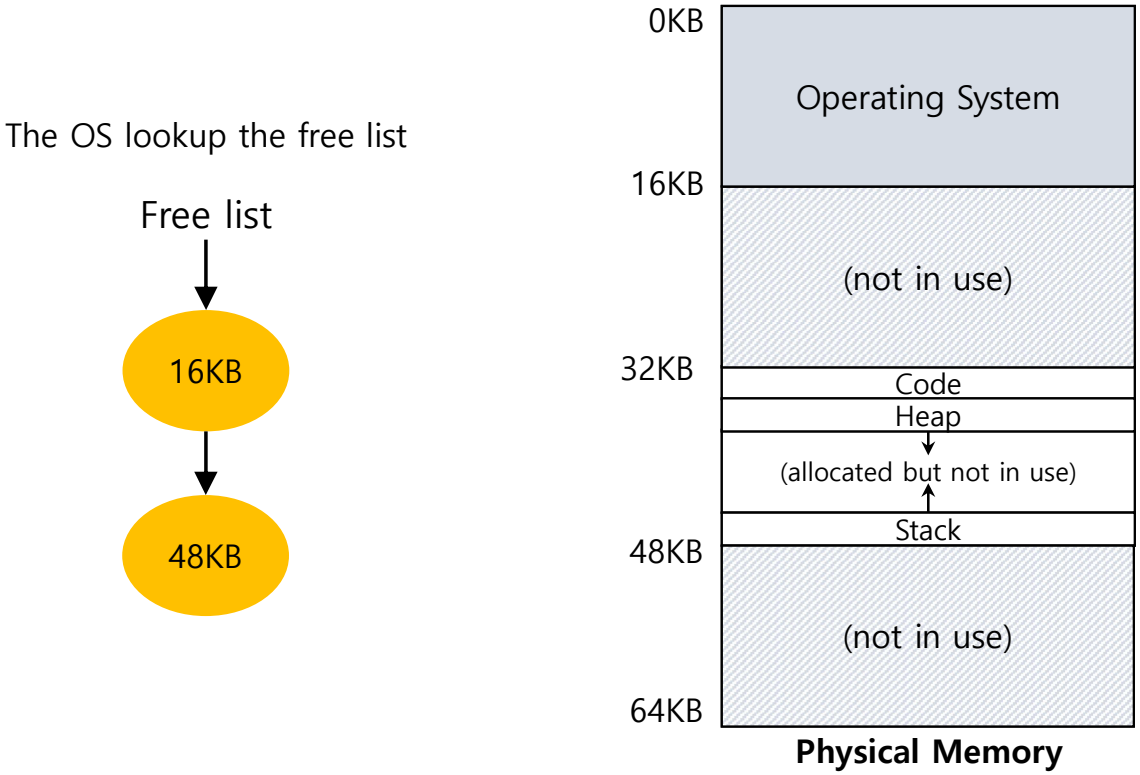


OS Issues for Memory Virtualizing

- 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

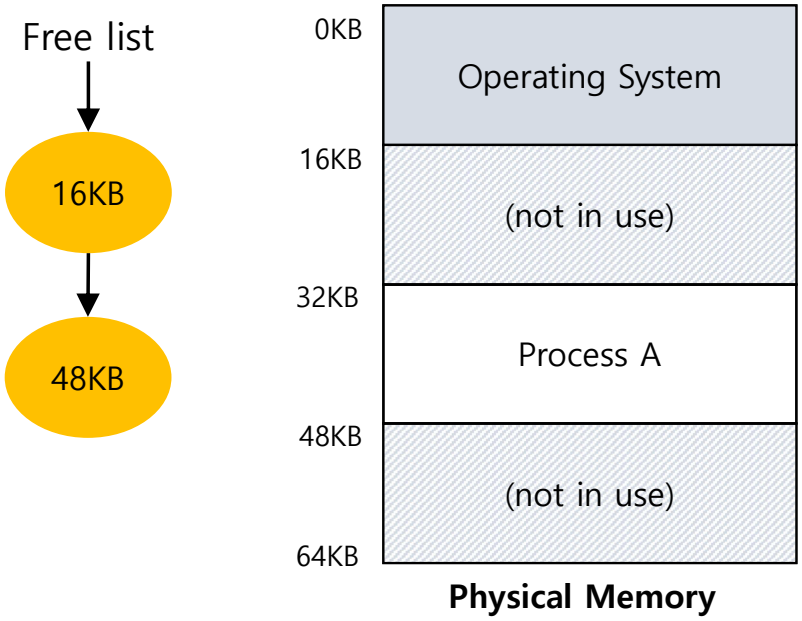
OS Issues: When a Process Starts Running

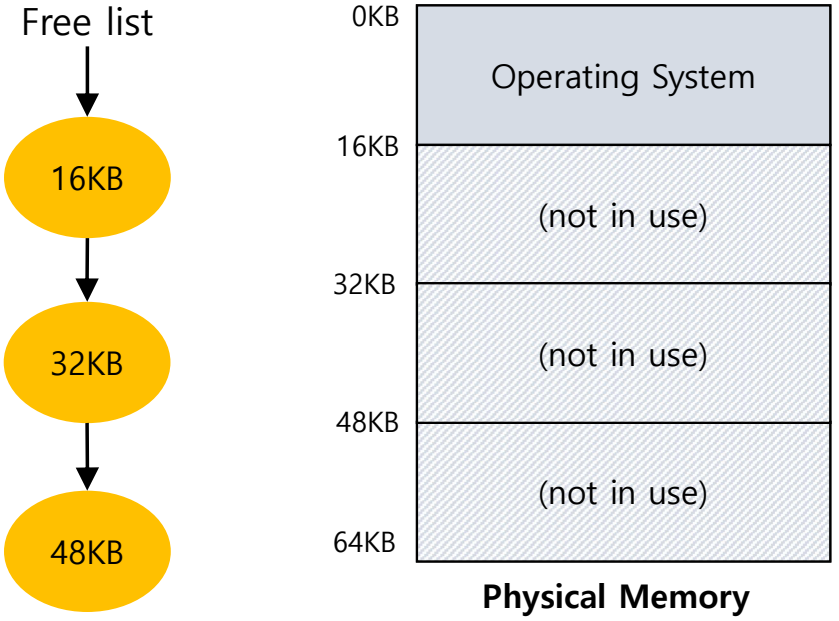
- The OS must **find a room** for a new address space.
  - free list : A list of the range of the physical memory which are not in use.



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)