

Chapter 15. Address Translation

Operating System: Three Easy Pieces

Dr. Jianhui Yue

Slides adapted from Dr. Youjip Won

- ▣ 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

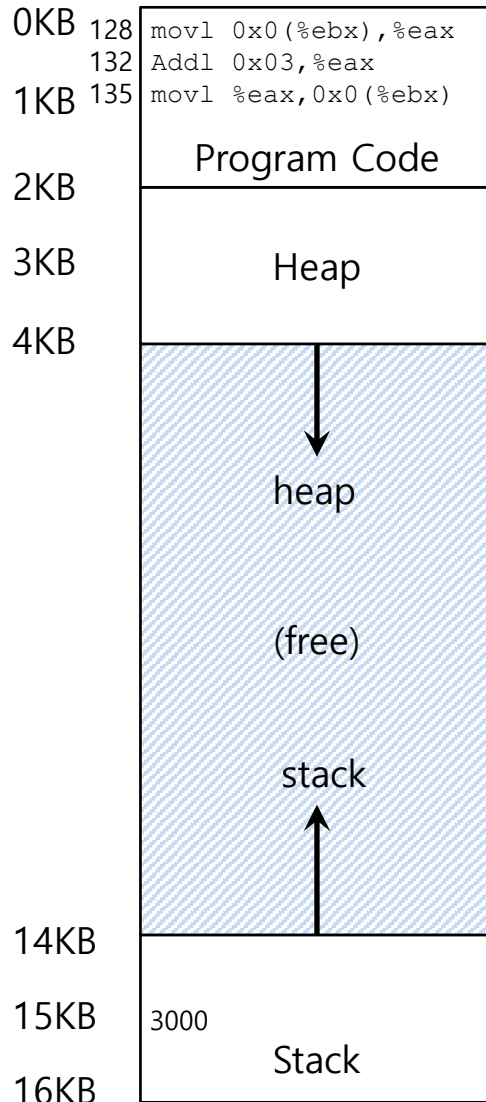
Example: Address Translation(Cont.)

▣ 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
```

- ◆ **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.)

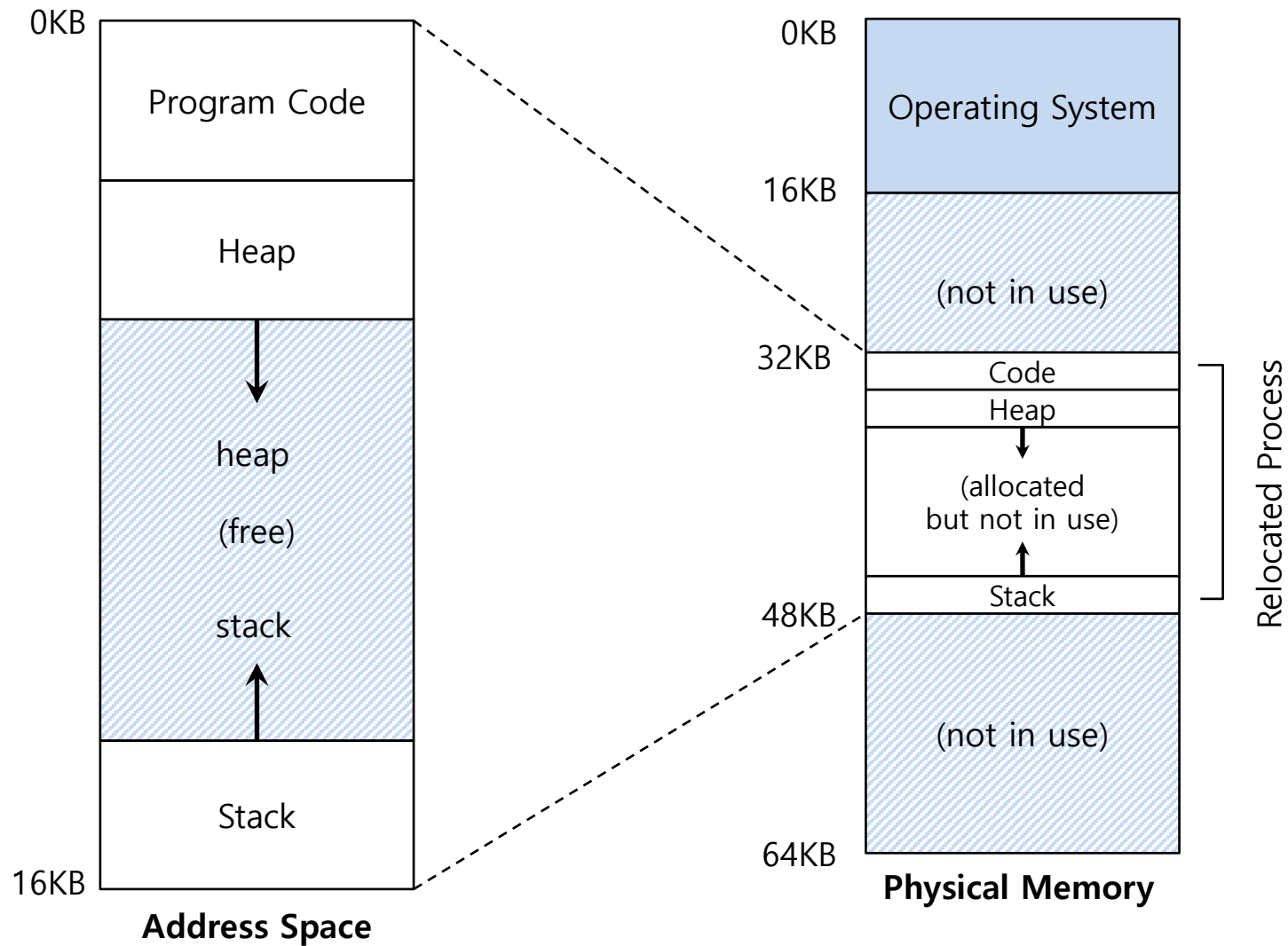


- Fetch instruction at address 128
- Execute this instruction (load from address 15KB)
- Fetch instruction at address 132
- Execute this instruction (no memory reference)
- Fetch the instruction at address 135
- Execute this instruction (store to address 15 KB)

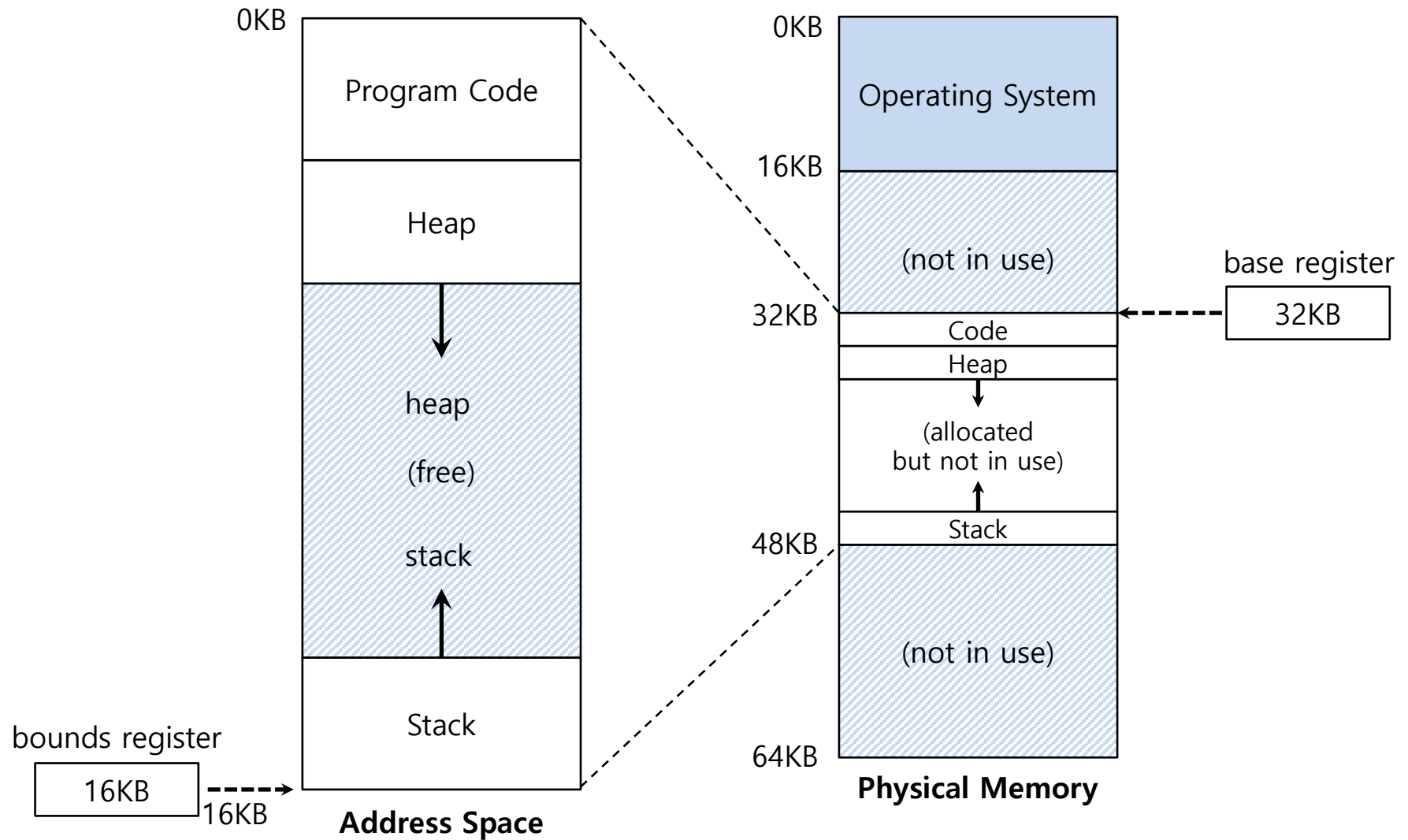
Relocation Address Space

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

$$\text{physical address} = \text{virtual address} + \text{base}$$

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

$$0 \leq \text{virtual address} < \text{bounds}$$

Relocation and Address Translation

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

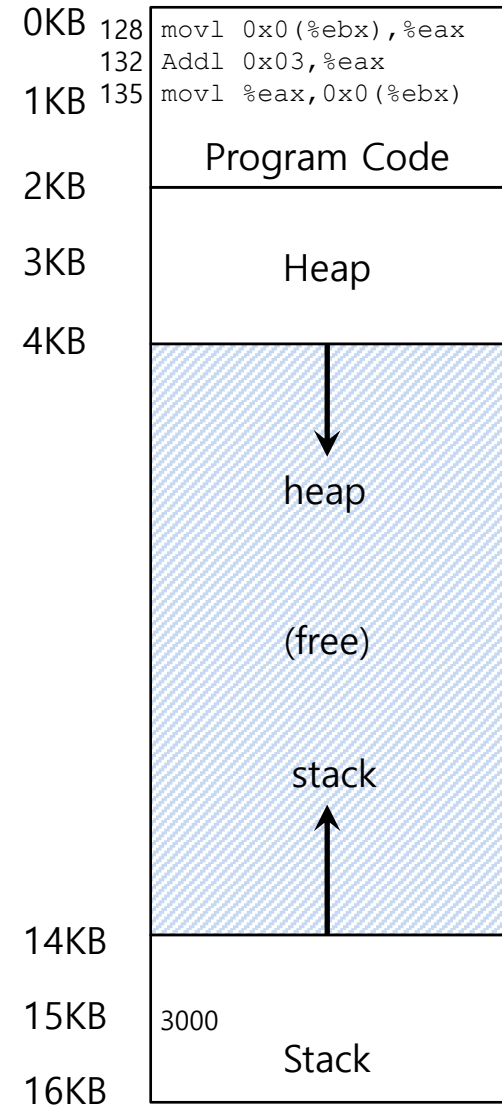
- ◆ **Fetch** instruction at address 128

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

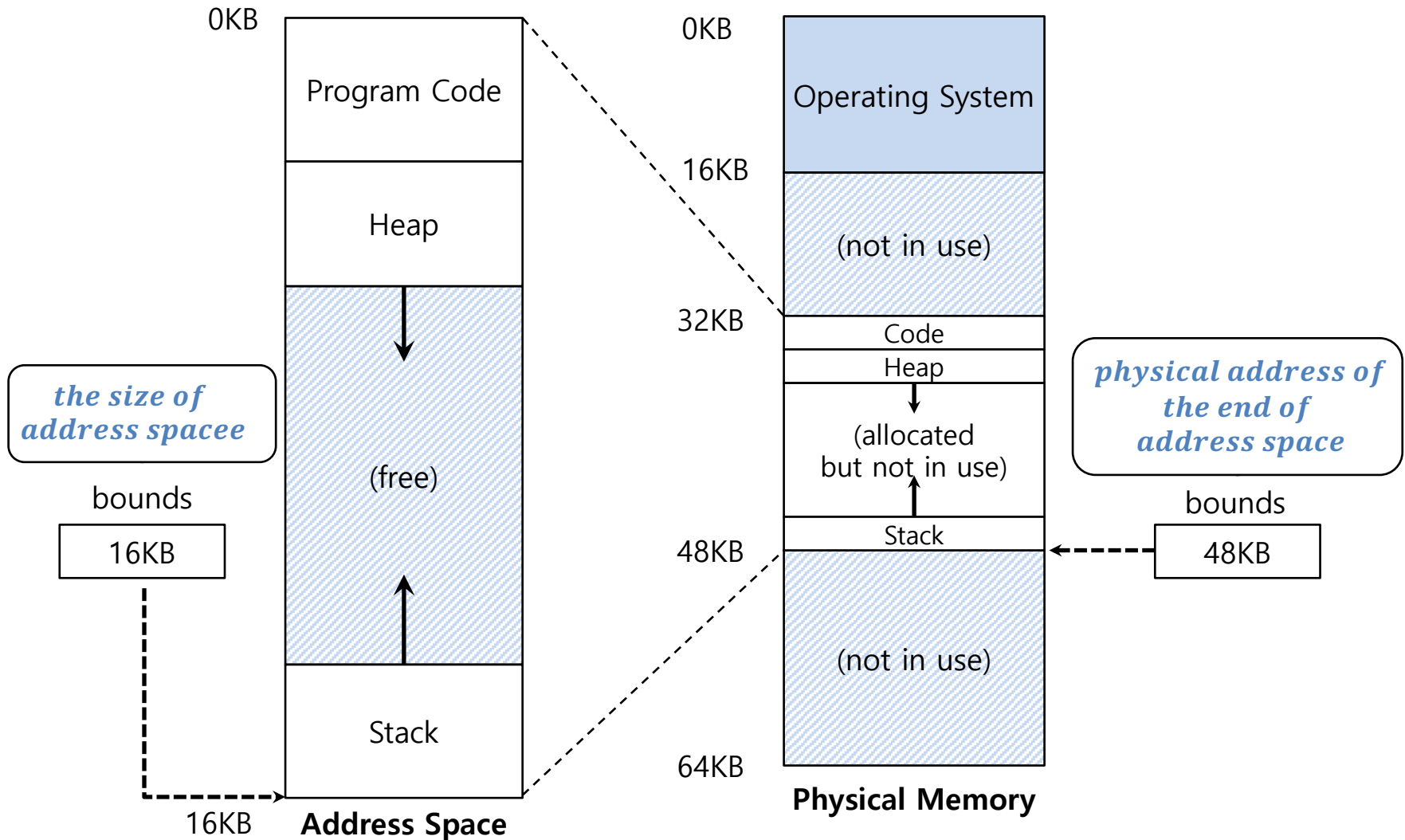
- ◆ **Execute** this instruction

- Load from address 15KB

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



Two ways of Bounds Register

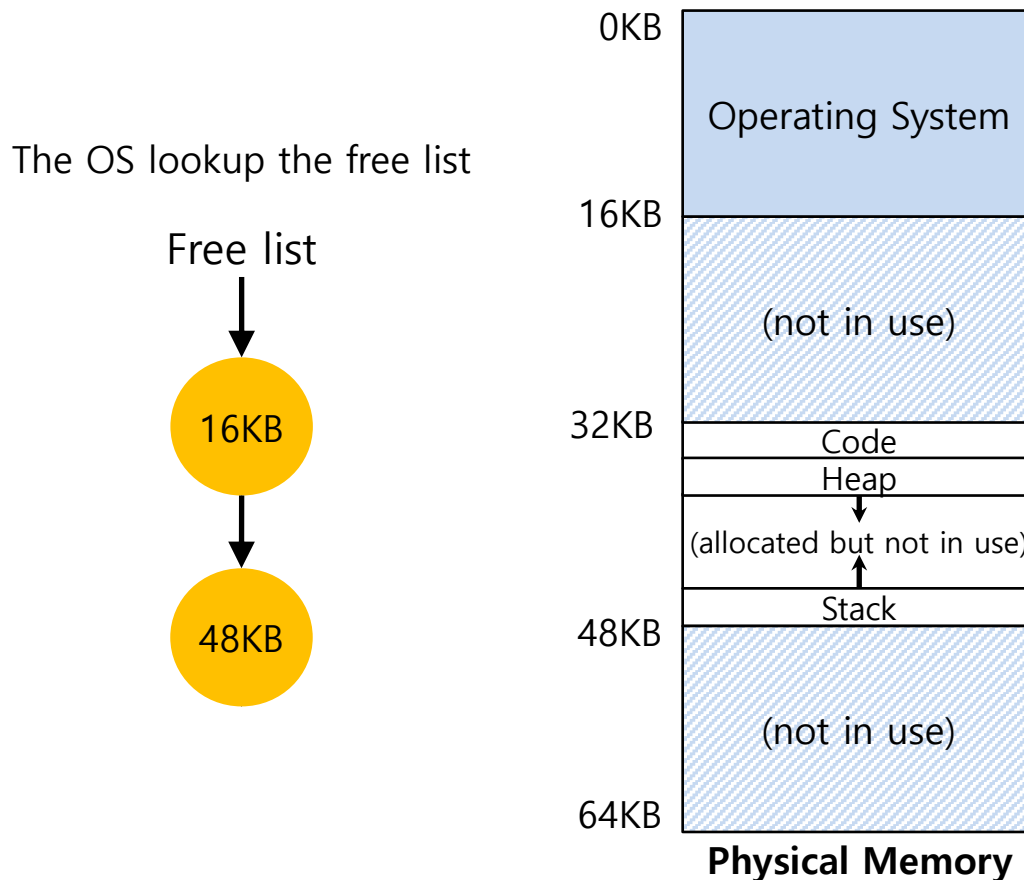


OS Issues for Memory Virtualizing

- ▣ 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

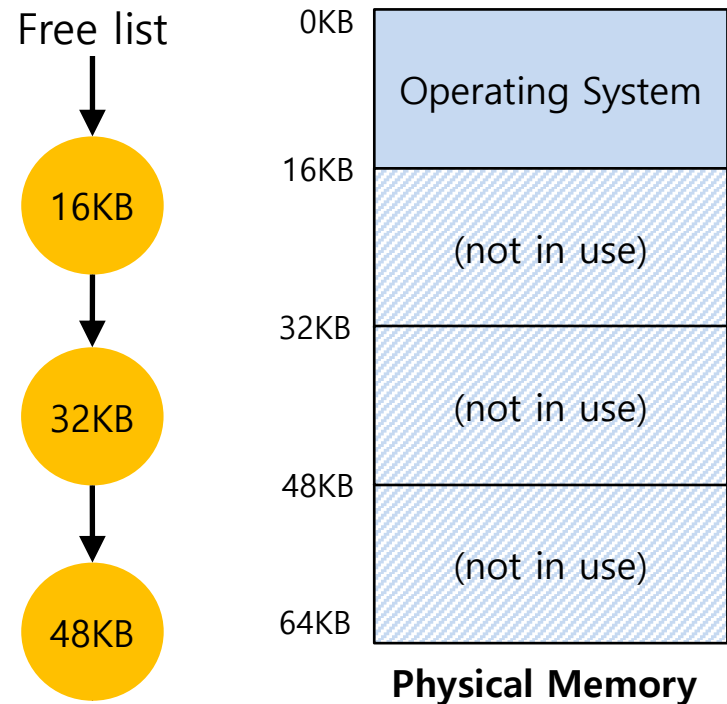
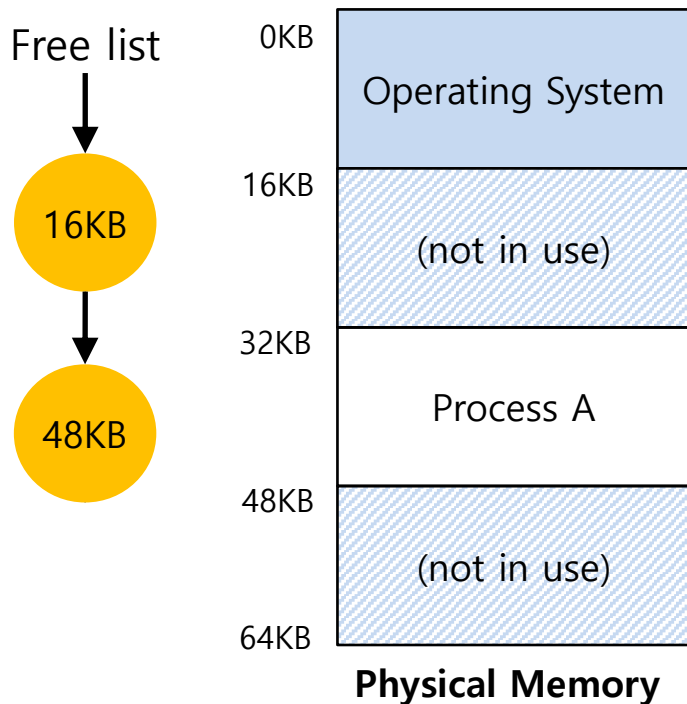
OS Issues: When a Process Starts Running

- ▣ 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)**

