15. Address Translation

Operating System: Three Easy Pieces

Memory Virtualizing with Efficiency and Control

- Why memory virtualization?
  - Two main reasons: Flexibility and Access Limitation.
- In Limited Direct Execution (LDE) for CPU virtualization, we used some hardware support to realize an efficient mechanism so that the OS could always be in control
- We want to obtain the same results (efficiency+control) in memory virtualization
  - Similarly to CPU virtualization, we will need some support from the hardware
  - Very simple at the beginning, but will grow fairly complex then.
- We want to study here mechanisms to realize an efficient virtualization of memory.

Address Translation

- The hardware support consists in the transformation of a **virtual address** to a **physical address**.
  - The desired information is actually stored in a physical address.
- The hardware, by itself, only provides mechanisms for the translation.
  - The OS is the entity that drives the hardware in the translation process
- Just like CPU virtualization, we want to create an **illusion:** the illusion that every process has all the memory for itself

3 Assumptions

- We will start the analysis of memory virtualization with 3 (unrealistic) assumptions:
  - The address space of a process must be placed contiguously in physical memory
  - The size of the address space is smaller than the size of the physical memory
  - Each address space has the same size
- Let's start with something simple, then we will relax these assumptions.

Example: Application Accessing Memory

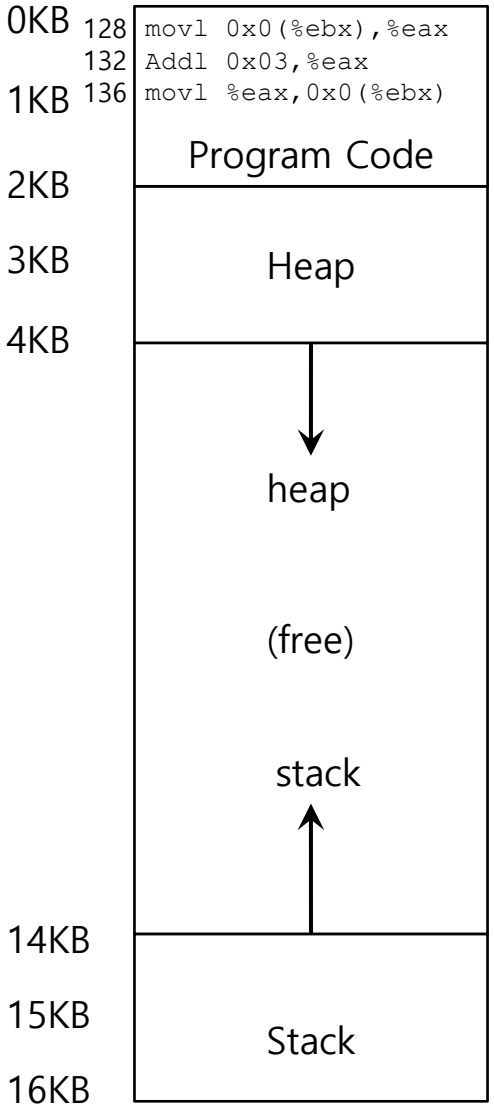
```
void func() {
        int x;
        ...
        x = x + 3; // this is the line of code we are interested in
}
```

- In C code:
  - **Load** a value from memory
  - **Increment** it by three
  - **Store** the value back into memory

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

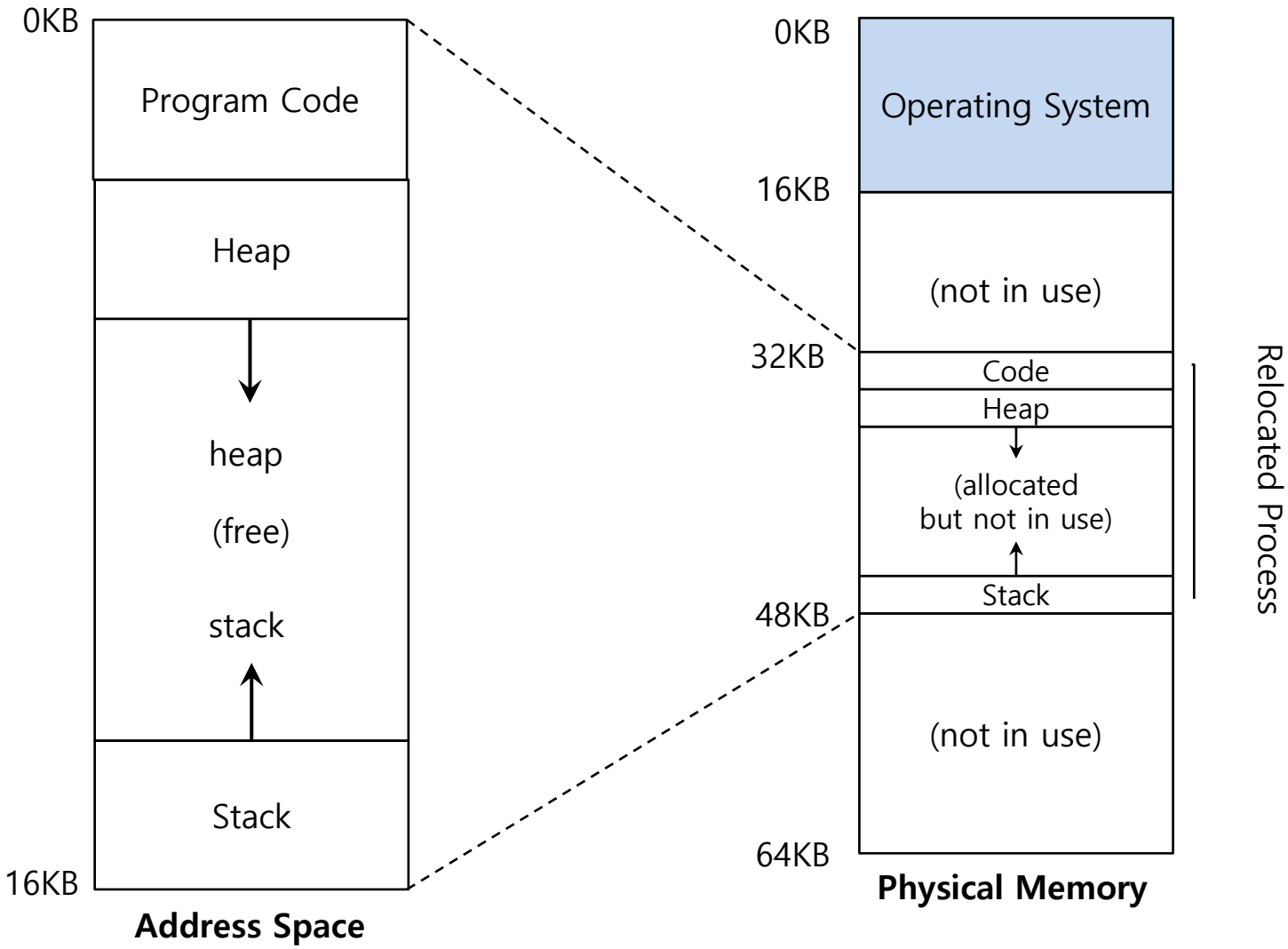
- In Assembly code:
  - **Load** the value at that address into `eax` register.
  - **Add** 3 to `eax` register.
  - **Store** the value in `eax` back into memory.

Example: Application Accessing Memory (Cont.)

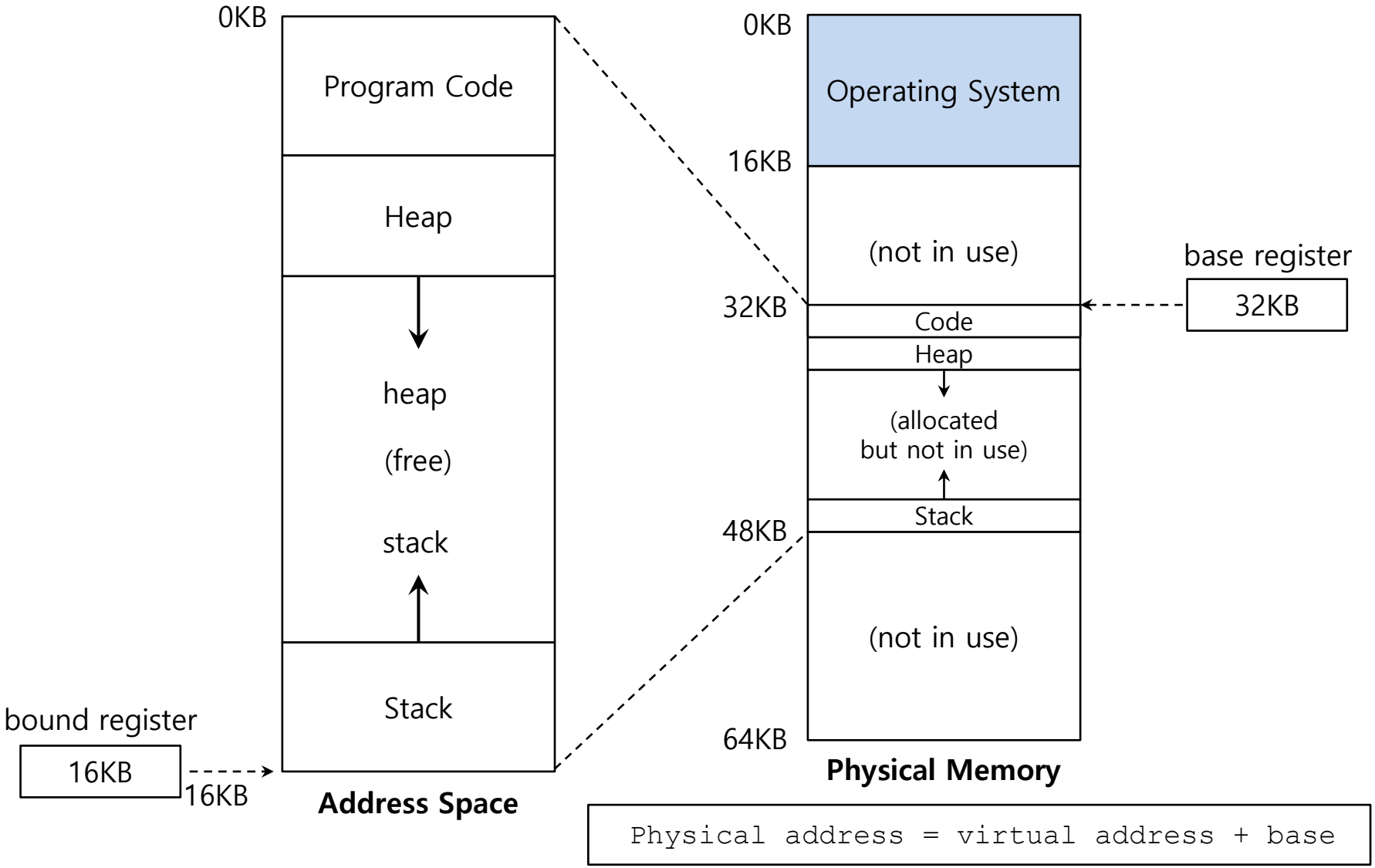


- Fetch instruction at address 128
- Execute this instruction (load from address 15KB)
- Fetch instruction at address 132
- Execute this instruction (no memory reference)
- Fetch the instruction at address 136
- Execute this instruction (store to address 15 KB)

A Single Relocated Process



Base and Bound Registers



Relocation and Address Translation

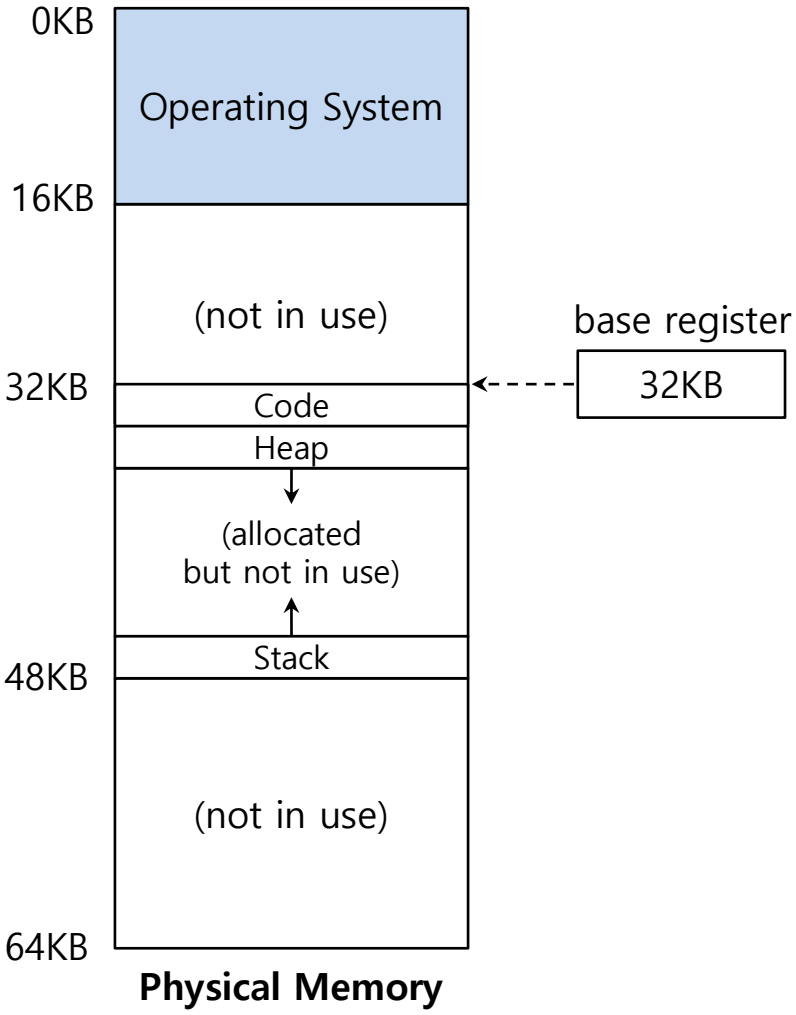
```
128 : movl 0x0(%ebx), %eax
```

- **Fetch** instruction at address 128

```
128 → 128 + 32768 = 32896
```

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

```
15K → 15K + 32K = 47K
```



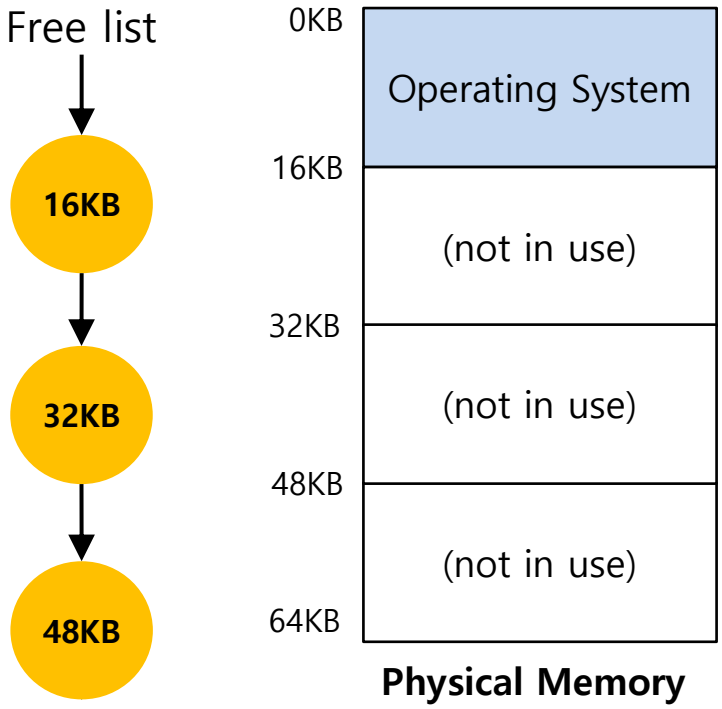
OS Issues for Memory Virtualizing

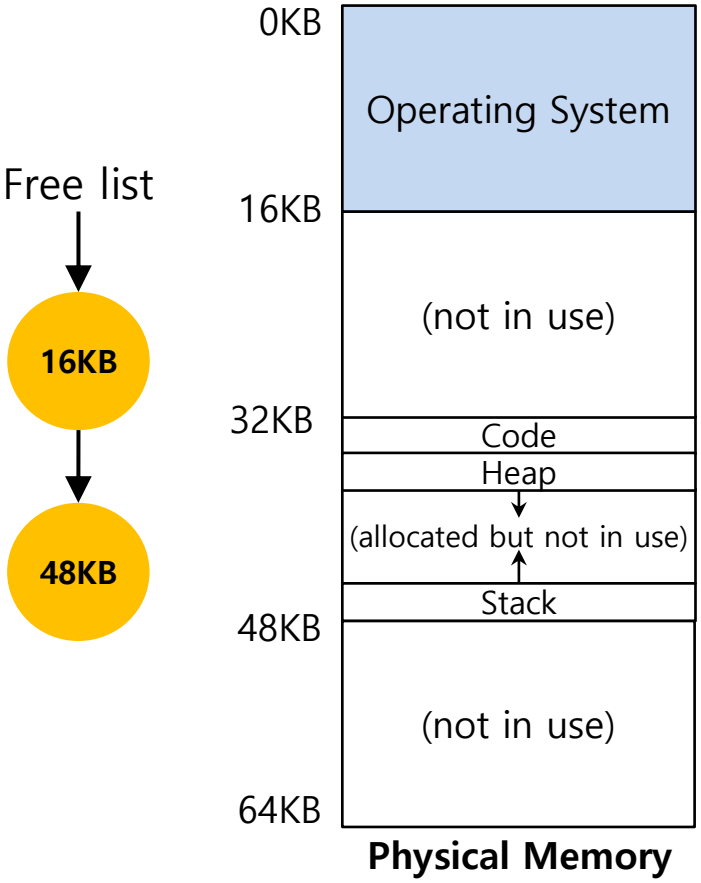
- Hardware support is good, but we also need some OS implementation
- The OS must **take action** to implement **base-and-bounds** approach.
- Three critical junctures:
  - When a process **starts running:**
    - Finding space for address space in physical memory
  - When a process is **terminated:**
    - 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 room** for a new address space.
  - free list : A list of the ranges of the physical memory which are not in use.
  - Easy with 16KB slots

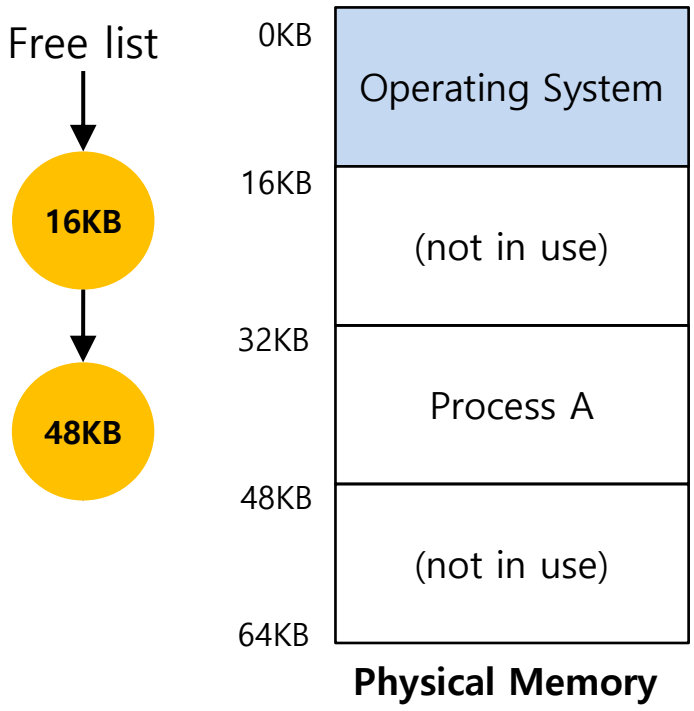
The OS searches the free list

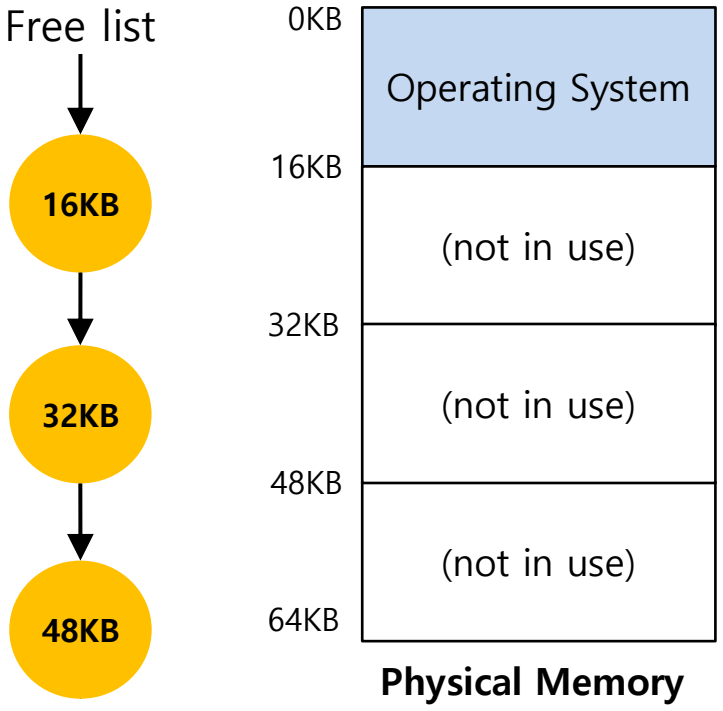




OS Issues: When a Process Is Terminated

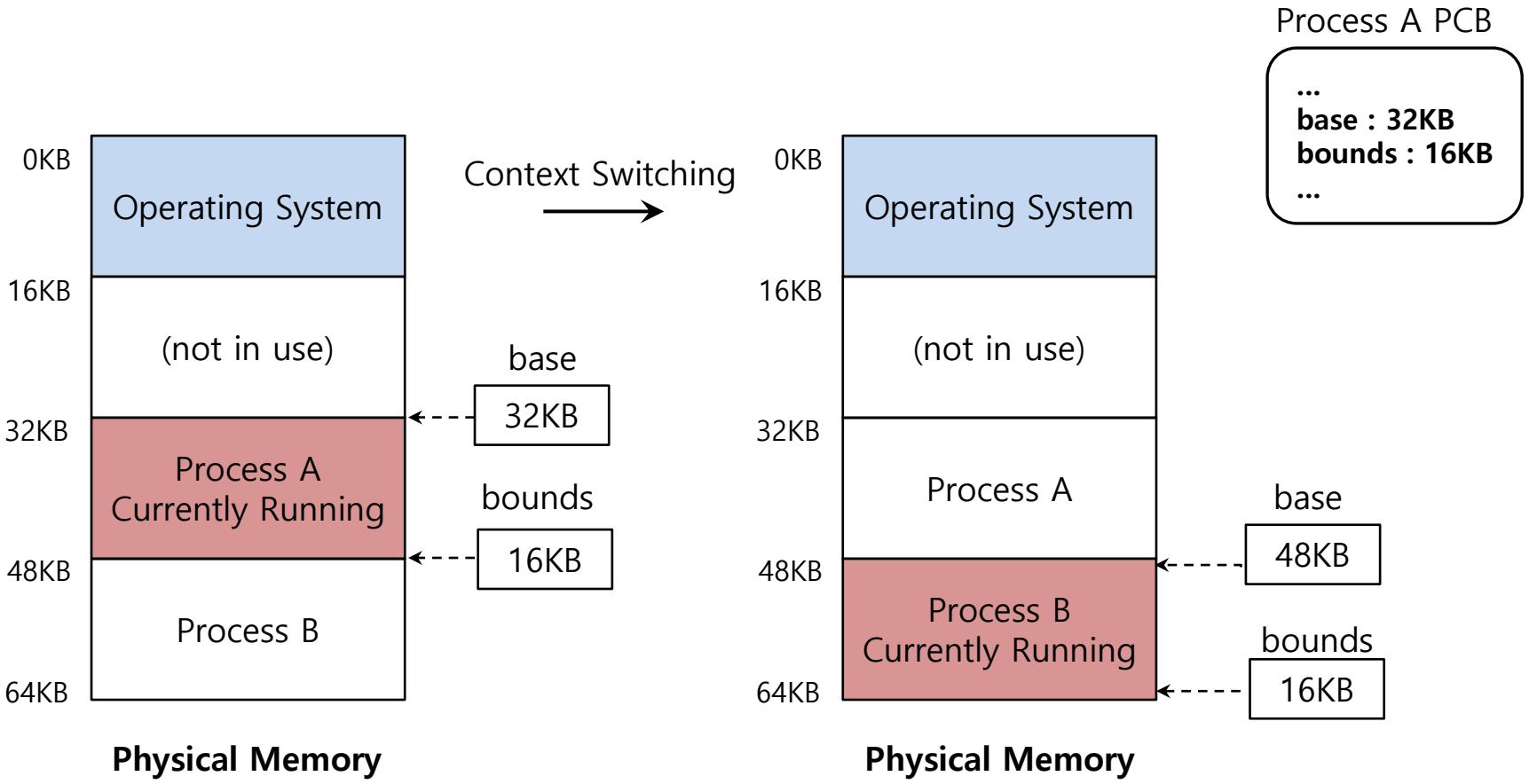
- The OS must **put the memory back** on the free list.





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 of Hardware Requirements so far

Hardware Requirements	Notes
Privileged mode	Needed to prevent user-mode processes from executing privileged operations
Base/bound registers	Need pair of registers per CPU to support address translation and bound checks
Ability to translate virtual addresses and check if within bounds	Circuitry to make translations and check limits, in this case quite simple
Privileged instruction(s) to update base/bounds	OS must be able to set these values before letting a user program run
Privileged instruction(s) to register exception handlers	OS must be able to tell hardware what code to run if exception occurs
Ability to raise exceptions	When processes try to access privileged instructions or out-of-bounds memory