Mechanism: Address Translation

Prof. Yongtae Kim

Computer Science and Engineering
Kyungpook National University

Virtualization of Memory

- **CPU virtualization is done by limited direct execution (LDE)**
  - The program run directly on the hardware through system calls (efficiency)
  - The OS ensures that it maintains control over the hardware (control)
- **Similarly, memory virtualization will attain both efficiency and control while providing the desired virtualization**
  - Efficiency dictates that we make use of hardware support
  - Control implies that the OS ensures that no application is allowed to access any memory but its own; thus, to protect applications from one another
  - Finally, we will need a little more from the VM system; flexibility that programs to be able to use their address spaces in whatever they would like
- **How to efficiently and flexibly virtualize memory?**
  - Hardware-based address translation (or simply address translation) allows the hardware to change the virtual address to a physical address
  - The OS must get involved at key points to set up the hardware
- **The goal of the memory virtualization is to create a beautiful illusion that the program has its own private memory**

Assumption and Example

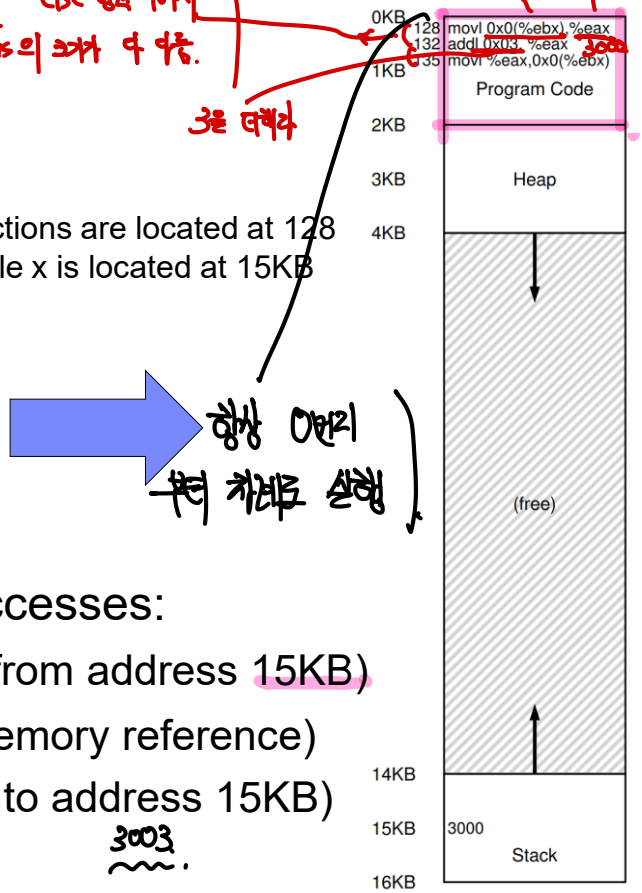
- **We will assume that:**
  1) The user's address space must be placed contiguously in physical memory
  2) The size of address space is less than the size of physical memory
  3) Each address space is exactly the same size
- **Let's look at a simple example**

```
void func() {
    int x = 3000; // thanks, Perry.
    x = x + 3;    // line of code we are interested in
    ...
```

Compiler

```
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
```

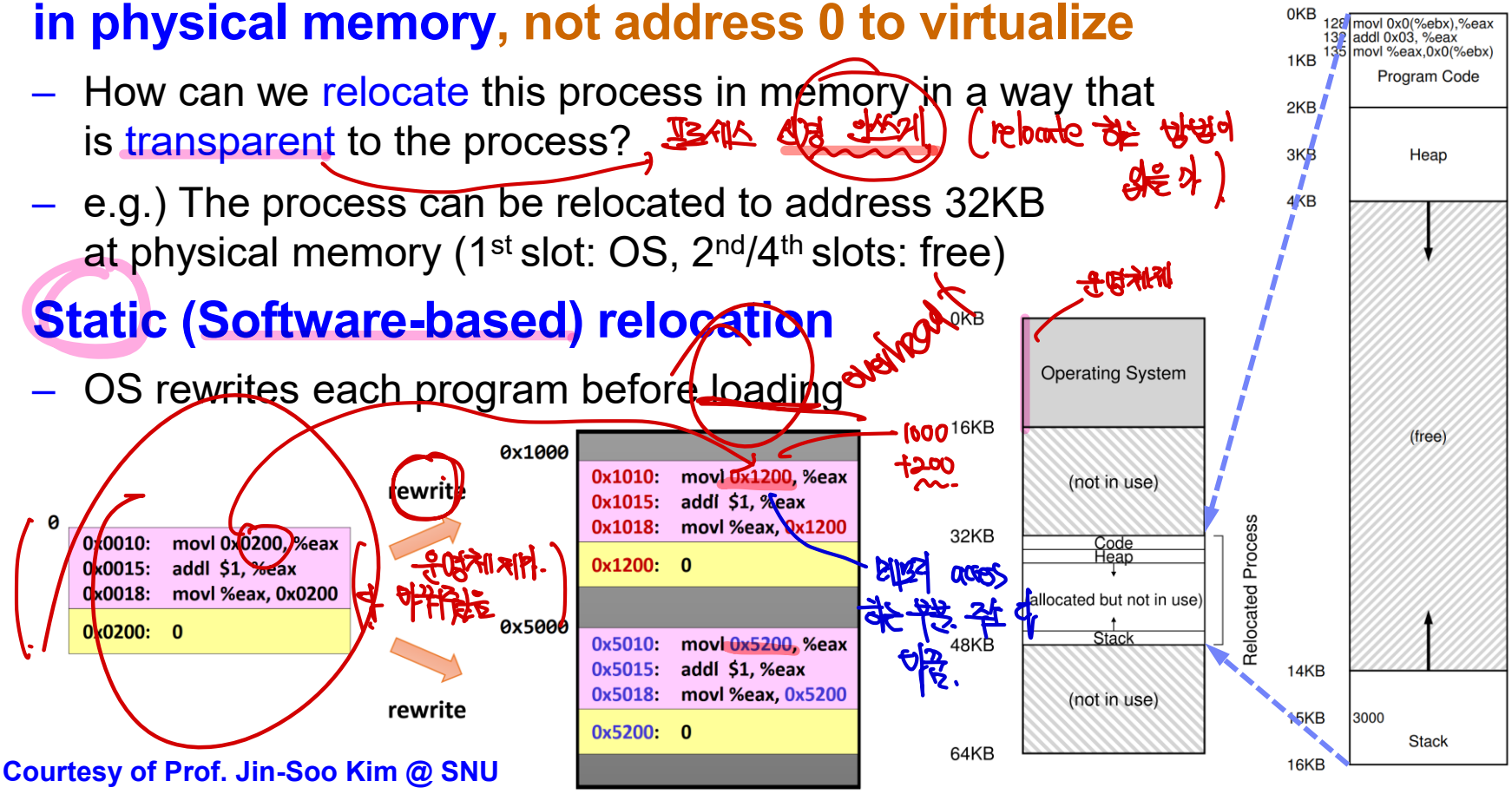
- Instructions are located at 128
- Variable x is located at 15KB



- To run the instructions, the following memory accesses:
  1) Fetch instruction at address 128 & Execute (load from address 15KB)
  2) Fetch instruction at address 132 & Execute (no memory reference)
  3) Fetch instruction at address 135 & Execute (store to address 15KB)

Process on Physical Memory

- **From the program's perspective:**
  - The address space start at address 0 and grows to a maximum of 16KB
- **The OS wants to place the process somewhere else in physical memory, not address 0 to virtualize**
  - How can we relocate this process in memory in a way that is transparent to the process?
  - e.g.) The process can be relocated to address 32KB at physical memory (1st slot: OS, 2nd/4th slots: free)
- **Static (Software-based) relocation**
  - OS rewrites each program before loading



Courtesy of Prof. Jin-Soo Kim @ SNU

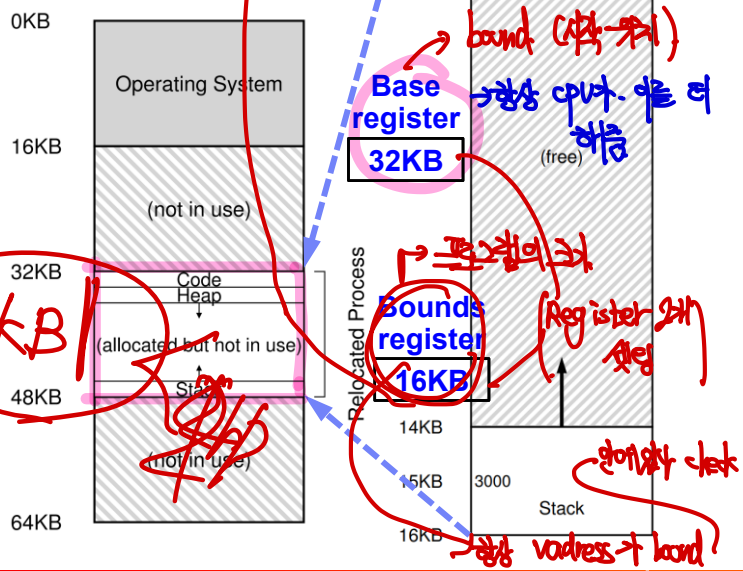
Dynamic (Hardware-based) Relocation (1)

- For efficient address translation, the base and bounds scheme (referred to as **dynamic relocation**) was introduced
  - We need two hardware registers within CPU: base and bounds (limit) registers
  - This base-and-bounds pair allows us to place address space anywhere in physical memory
- Each program is written and compiled **as if it is loaded at address zero**, and when it starts running
  - The OS decides where in physical memory it should be loaded and thus sets the base register to this value
- Processor translates the address by

physical address = virtual address + base

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

0 ≤ virtual address < bounds



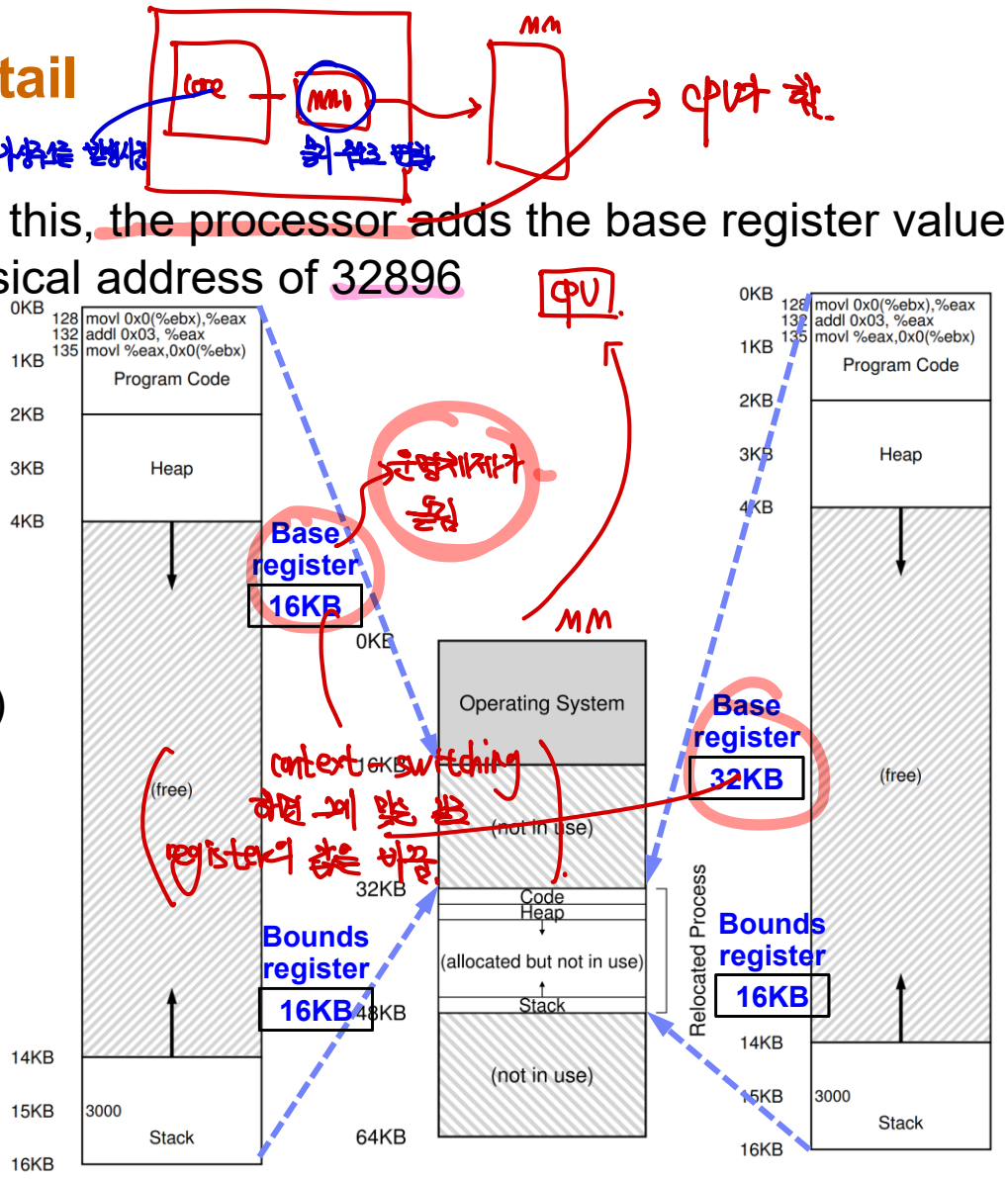
Dynamic (Hardware-based) Relocation (2)

- **Let's take a look at the detail**

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

  - The PC is set to 128; To fetch this, the processor adds the base register value of 32 KB (32768) to get a physical address of 32896
  - The base register is set by OS when context switching
  - In fact, the CPU first checks that the memory reference is within bounds to make sure it is legal; otherwise, it will raise an exception (protection)

- **Such a hardware is called memory management unit (MMU)**
  - located inside the processor



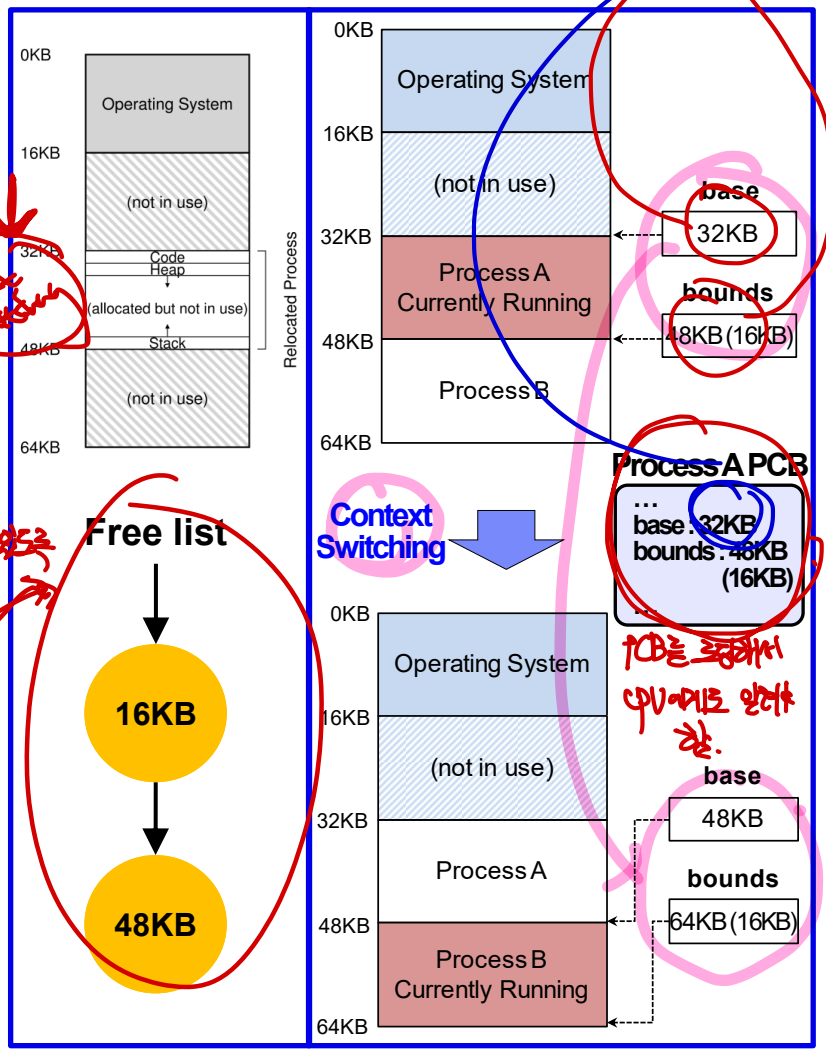
Hardware Support: A Summary

- **Hardware requirements for dynamic relocation (MMU)**

Hardware Requirements	Notes
Privileged mode	*Needed to prevent user-mode processes from executing privileged operations*
Base/bounds registers	*Need pair of registers per CPU to support address translation and bounds checks*
Ability to translate virtual addresses and check if within bounds	*Circuitry to do 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*

- **Operating System Issues**

OS Requirements	Notes
Memory management	*Need to allocate memory for new processes; Reclaim memory from terminated processes; Generally manage memory via free list*
Base/bounds management	*Must set base/bounds properly upon context switch*
Exception handling	*Code to run when exceptions arise; likely action is to terminate offending process*



Limited Direct Execution Revisited

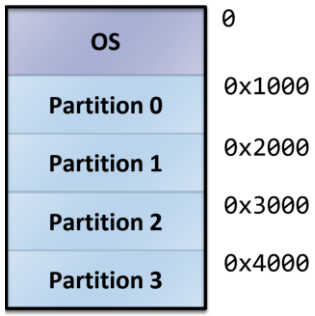
- **Initialization @ Booting**

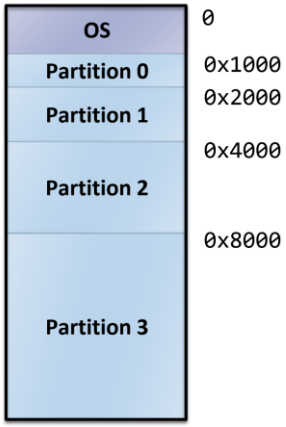
OS @ boot (kernel mode)	Hardware	(No Program Yet)
initialize trap table	remember addresses of... system call handler, timer handler, illegal mem-access handler, illegal instruction handler	
start interrupt timer	start timer; interrupt after X ms	
initialize process table, initialize free list		

OS @ run (kernel mode)	Hardware	Program (user mode)
To start process A: allocate entry in process table, alloc memory for process, set base/bound registers, return-from-trap (into A)		
	restore registers of A, move to user mode, jump to A's (initial) PC	
		Process A runs: Fetch instruction
	translate virtual address, perform fetch	
		Execute instruction
	if explicit load/store: ensure address is legal, translate virtual address, perform load/store	
	Timer interrupt: move to kernel mode, jump to handler	
Handle timer: decide: stop A, run B, call switch() routine, save regs(A) to proc-struct(A) (including base/bounds), restore regs(B) from proc-struct(B) (including base/bounds), return-from-trap (into B)		
	restore registers of B, move to user mode, jump to B's PC	
		Process B runs: Execute bad load
	Load is out-of-bounds; move to kernel mode, jump to trap handler	
Handle the trap: decide to kill process B, deallocate B's memory, free B's entry in process table		

Note how its memory translations are handled by the hardware with no OS intervention

- **Fixed and Variable Partitions**
  - Partition size needs not be equal





OS. Hw.

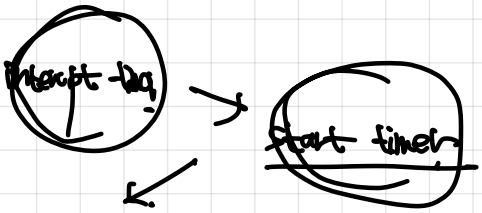
trap table

system-call handler.

timer handler

illegal mem-access handle

illegal instruct error.



process table.

initialize. free-list.

Summary

- **CPU generates a virtual (logical) address and MMU translates it into the physical address**
  - MMU includes base and bound registers, which are used to load programs