Engineering &
Computer Science
Syracuse University

CIS 657 – Principles of Operating Systems

Topic: Memory – Address Translation

Endadul Hoque

Acknowledgement

- Youjip Won (Hanyang University)
- OSTEP book – by Remzi and Andrea Arpaci-Dusseau (University of Wisconsin)

Memory Virtualizing with Efficiency and Control

- How to build an efficient virtualization of memory?
- How to maintain control over memory to enable protection?
- How to provide the flexibility needed by applications?

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

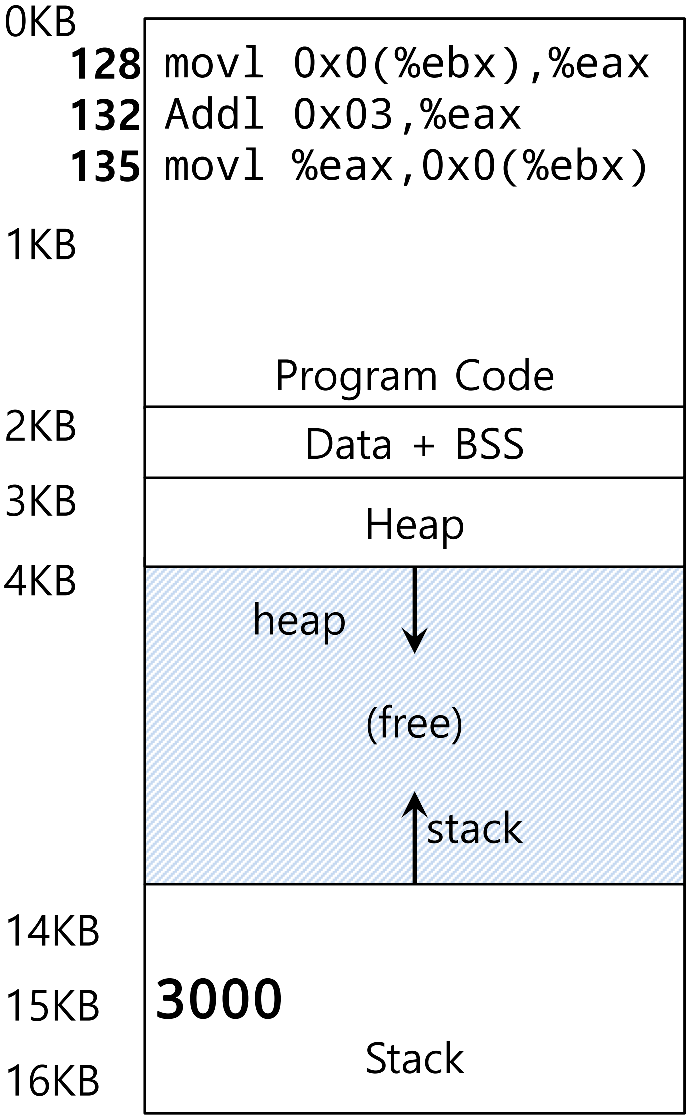
Example: Address Translation

- 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



- 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)

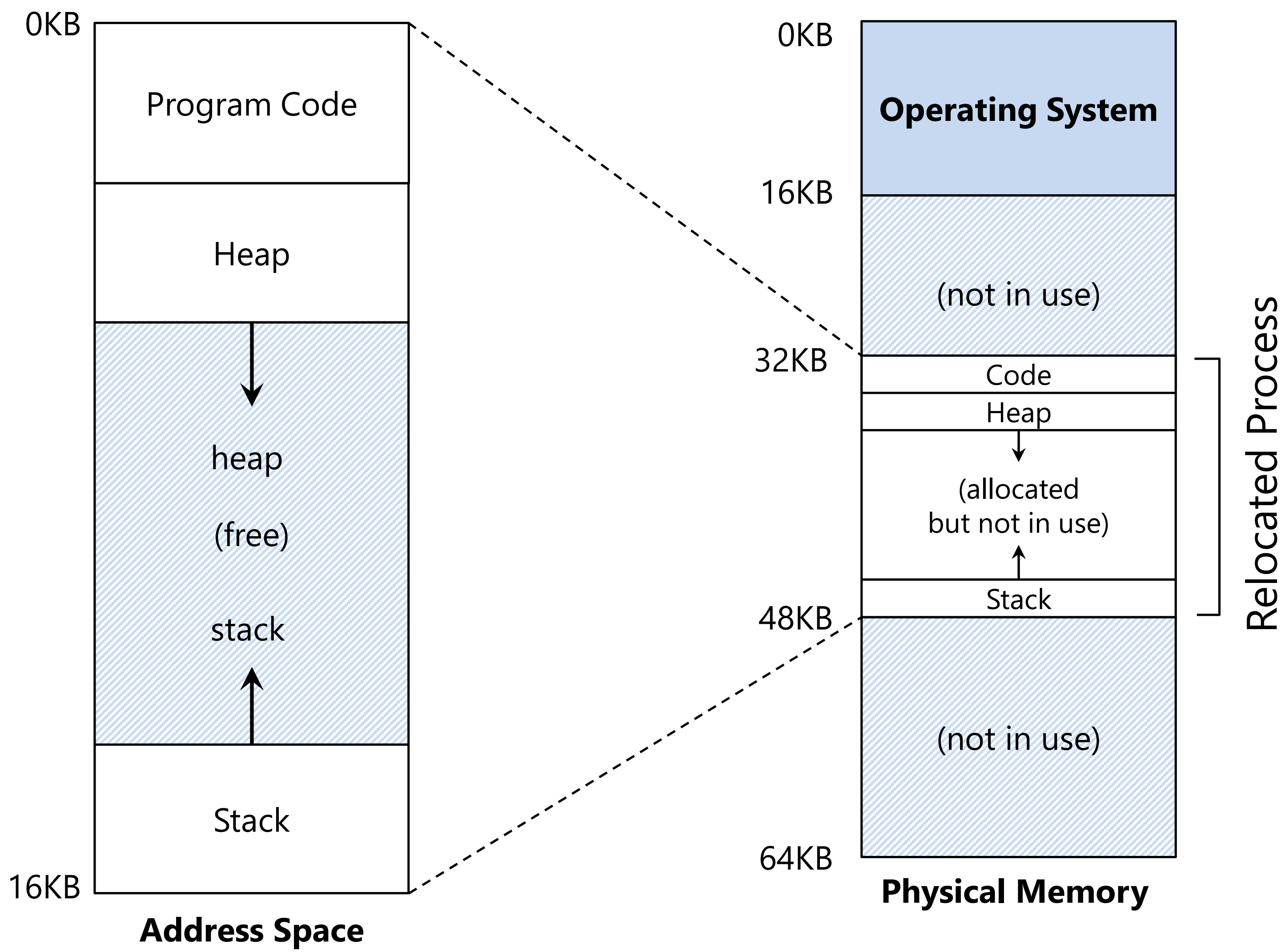
Relocation Address Space

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

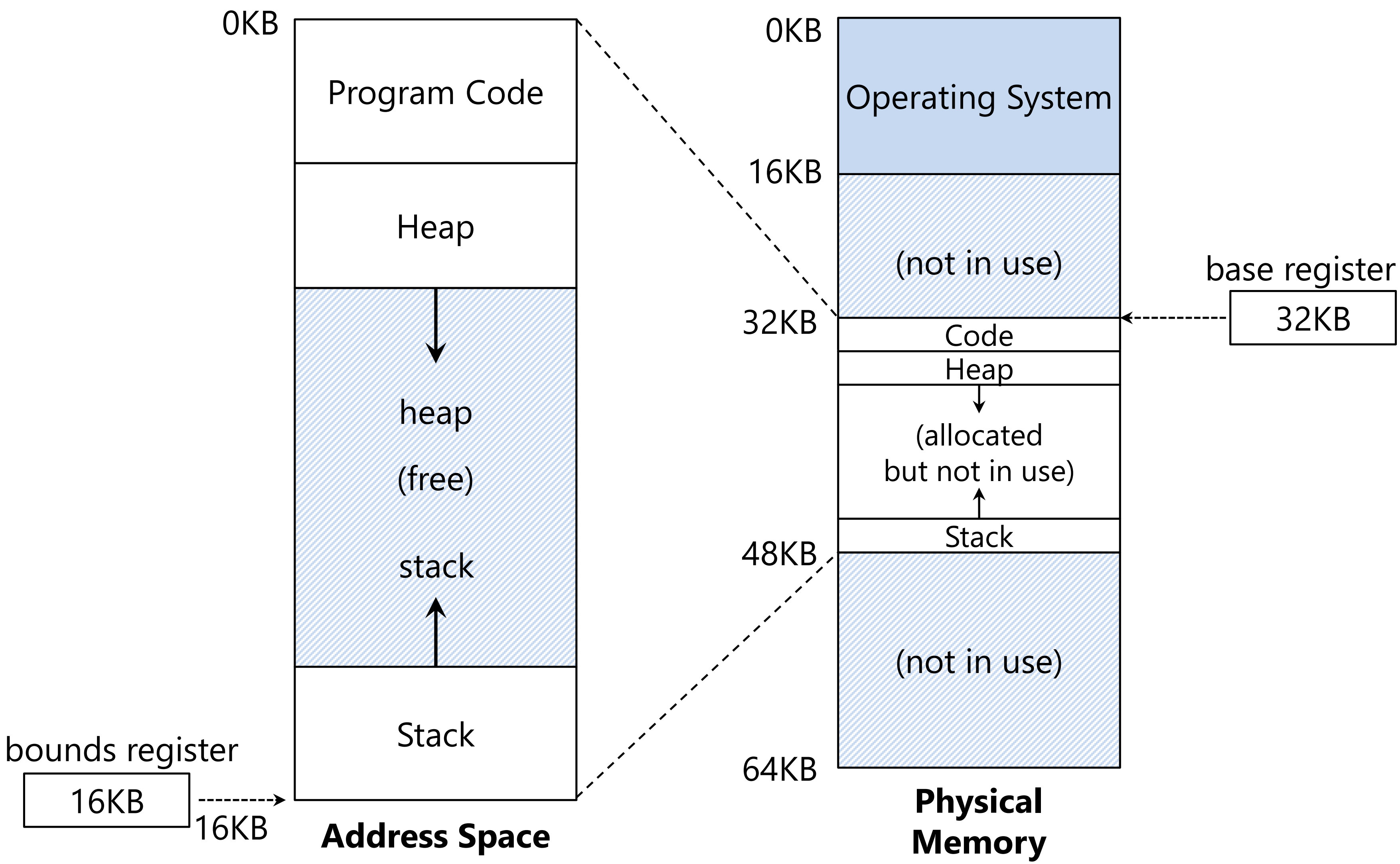
Working Assumptions

1. The address space must be placed **contiguously** in physical memory
2. The size of the address space is **less than** the size of physical memory
3. Each address space is exactly the **same size**

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.

$$physical\ address = virtual\ address + base$$

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

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

Relocation and Address Translation

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

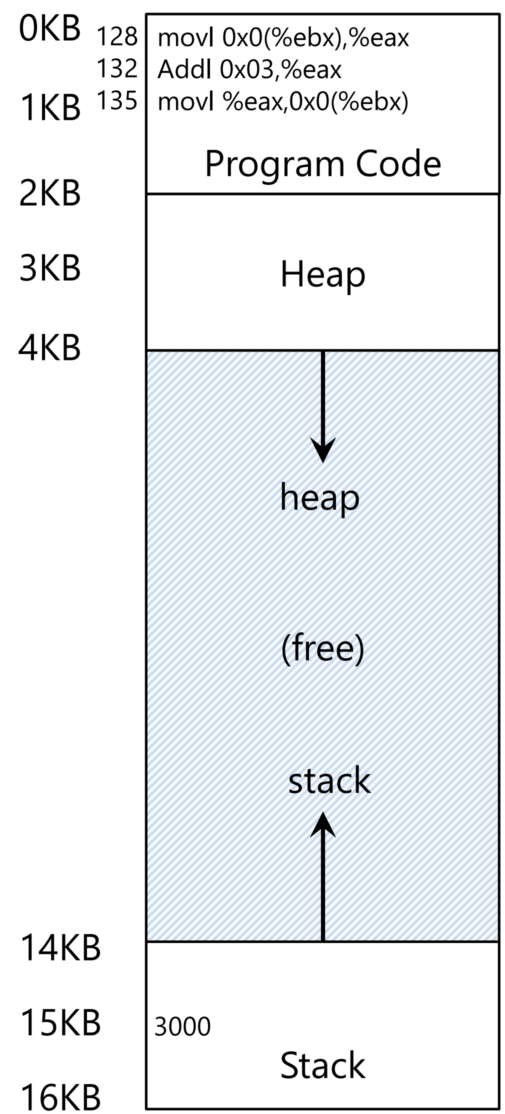
– **Fetch** instruction at address 128

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

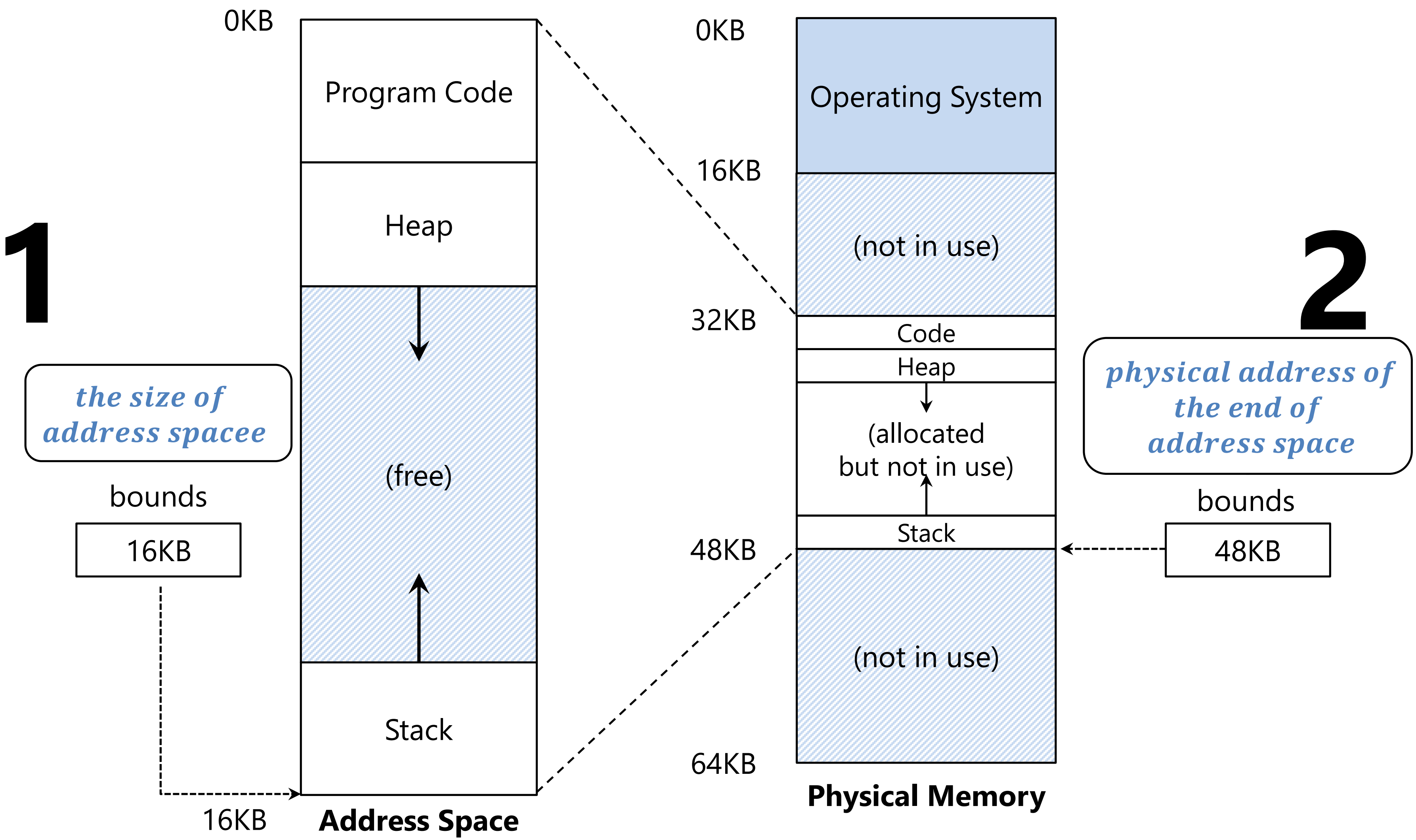
– **Execute** this instruction

- Load from address 15KB

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



Two ways to use Bounds Register



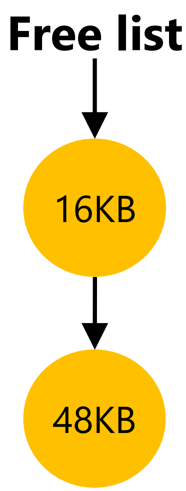
OS Issues for Memory Virtualizing

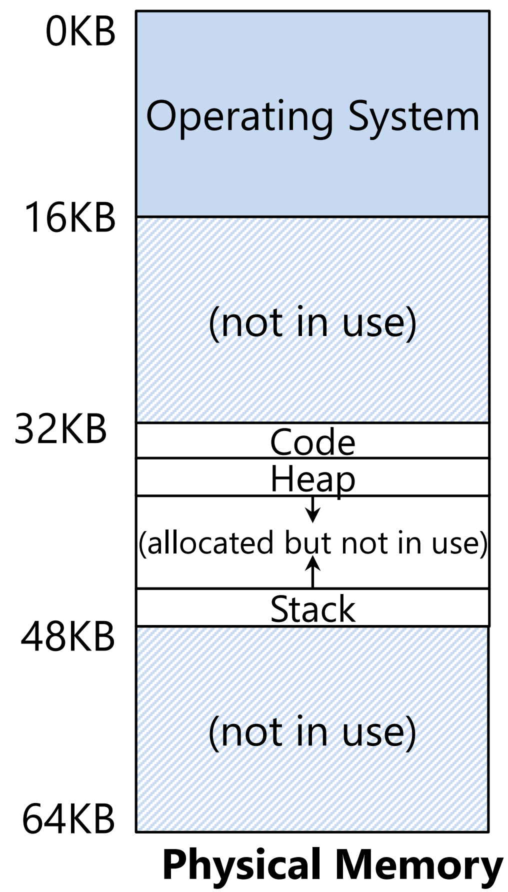
- The OS must **take action** to implement **base-and-bounds** approach.
- Three critical junctures:
  - When a process **starts running:**
    - Finding space for address space in physical memory
  - When a process is **terminated:**
    - Reclaiming the memory for use
  - When context **switch occurs:**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

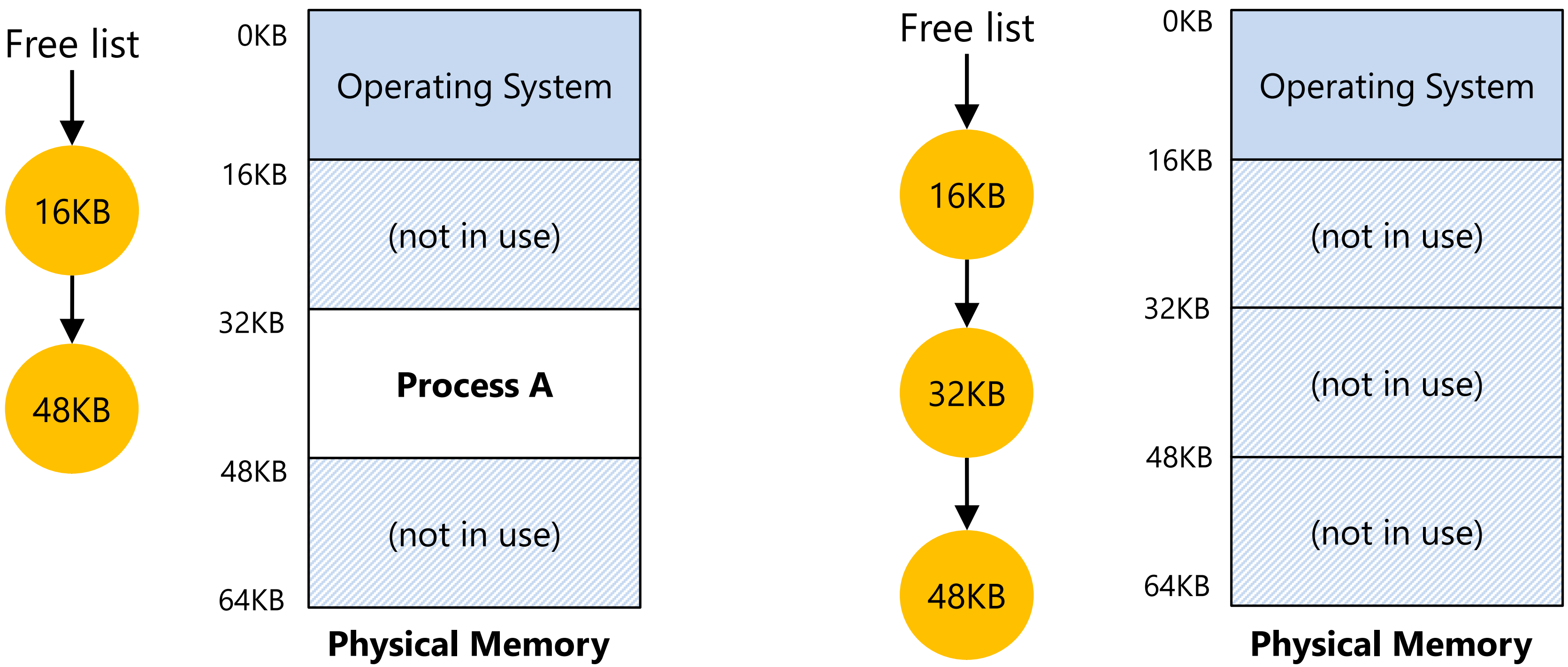
The OS lookup the free list





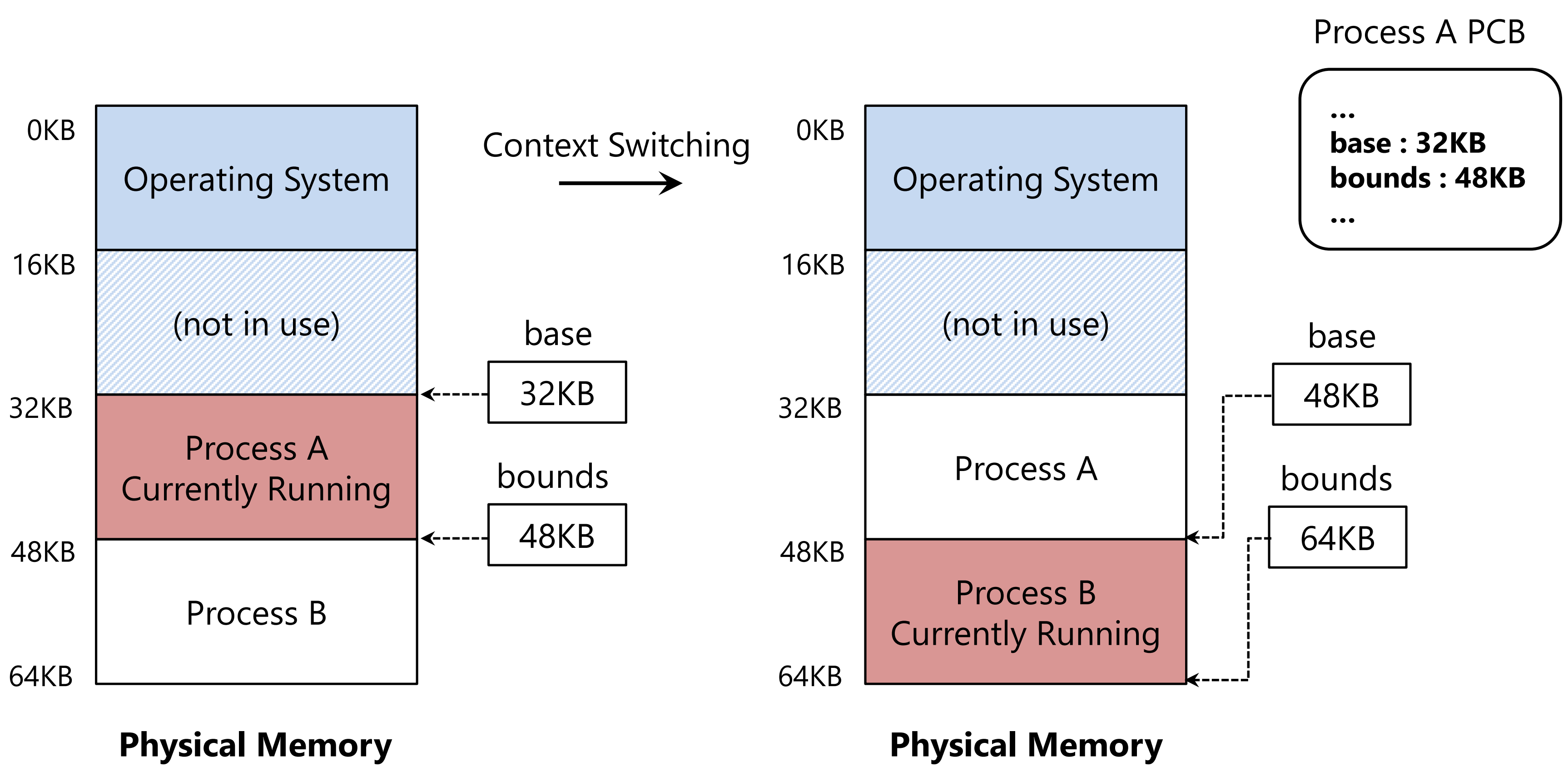
OS Issues: When a Process Is Terminated

- 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)



Hardware support

- Hardware must provide the base and bounds registers
  - A **pair of registers per CPU** to support address translation and bounds check
- Part of the memory management unit (**MMU**) of the CPU
- Hardware should provide **special instructions** to modify these registers
  - OS needs to change them when different processes run
- CPU must be able to **generate/raise exceptions** in case of illegal memory access
  - CPU must stop executing the current process and run the OS "out-of-bound" exception handler

Limited Direct Execution (w/ Dynamic Relocation)

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		

@Boot

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	
		(A runs...)
	Timer interrupt	
	move to **kernel mode**	
	jump to handler	

@Runtime

@Runtime

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		

Reading Material

- **Chapter 15** of OSTEP book – by Remzi and Andrea Arpaci-Dusseau (University of Wisconsin) http://pages.cs.wisc.edu/~remzi/OSTEP/vm-mechanism.pdf

Questions?