Address Translation

Chapter 15

Previously in CS212…

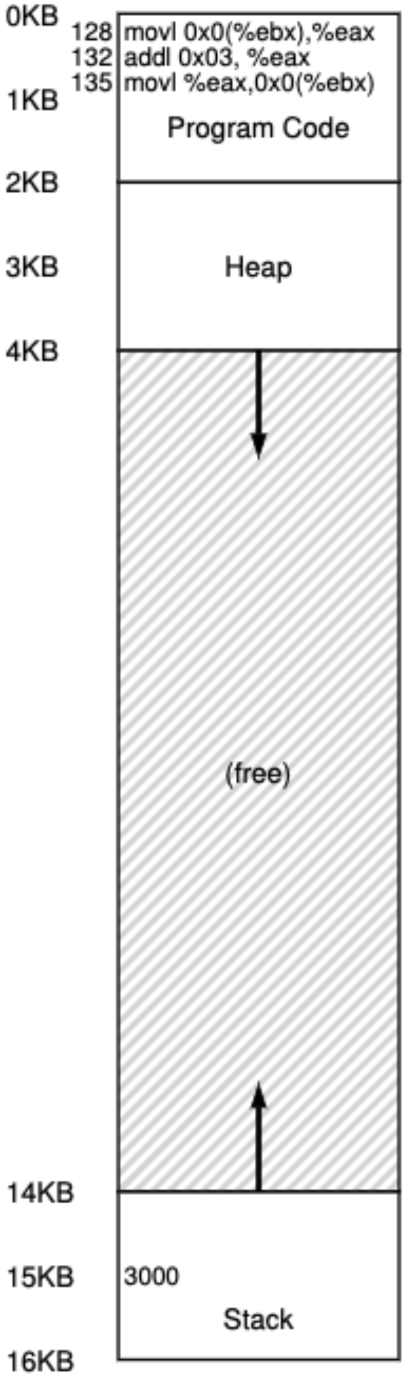
- Examined the concept of virtual memory and the address space abstraction
  - Each process gets its own space in memory
  - This space is isolated from other processes
- Reviewed the memory API provided by our OS
  - Malloc, Calloc, Realloc, Free
- Experimented with tools for checking memory usage and debugging
  - GDB Debugger
  - Valgrind
- But do we manage virtual memory?

Process Address space

- Process A is loaded into an address space
    - stores 3,000 to a variable and increments the value by 3

- We have three lines of code at bytes: 128, 132, and 135

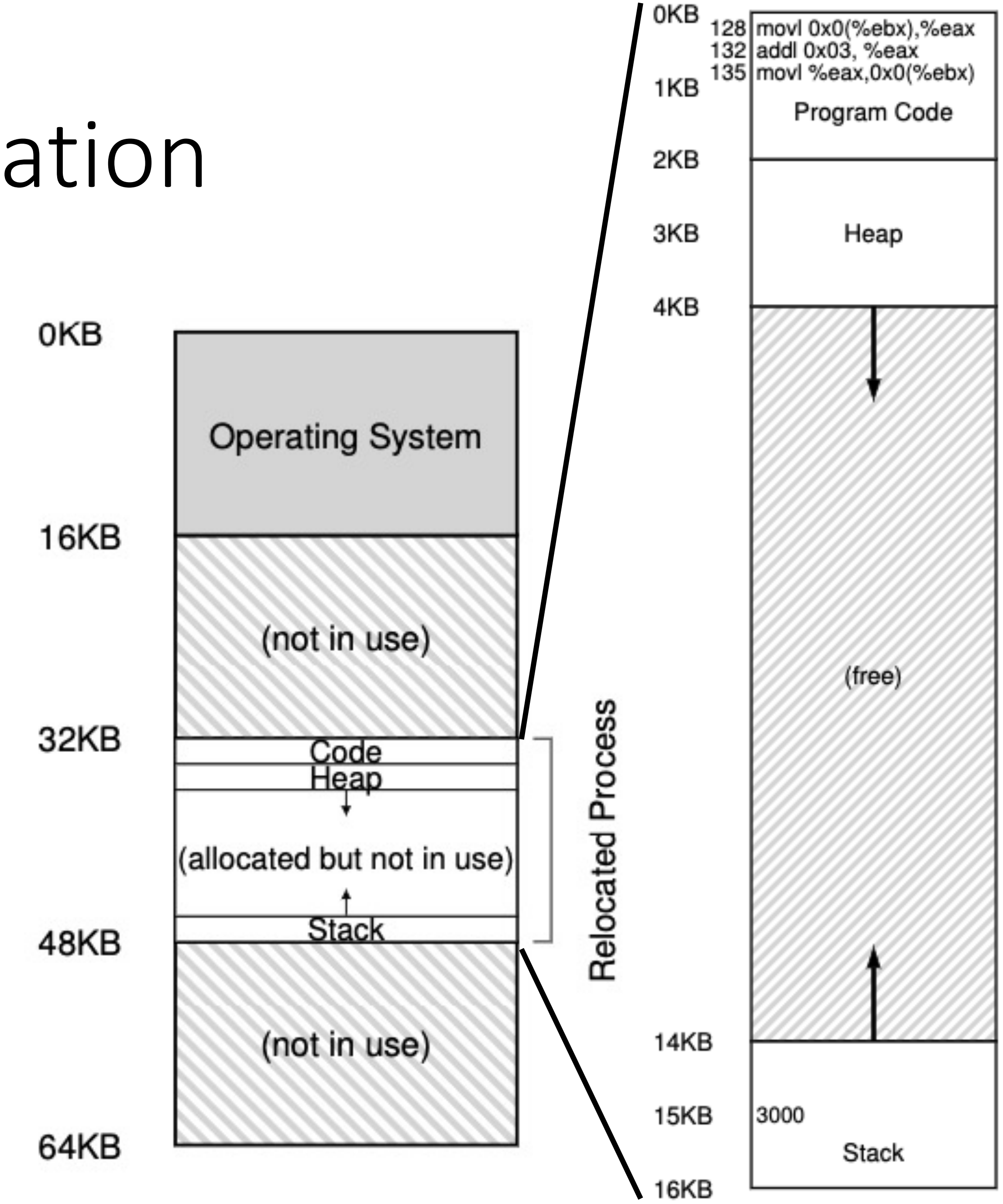
- We have the value of the variable (3,000) stored at 15KB



```
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
```

We Need Address Translation

- Assuming the process image is in a contiguous block of memory…
- OS decided where in physical memory the process goes
- We need to know how to go from virtual addresses to physical addresses
  - 128 B + 32,768 B = 32,896 B
  - 15 KB + 32 KB = 47 KB



```
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
```

Memory Management Unit

- Hardware built into the CPU

- Keeps two registers
  - base - starting address
  - bound - can be the size of the process image or the physical address for the end of the process image

- Converts virtual to physical addresses
  - Physical address = virtual address + base
  - Checks to ensure the bounds are not violated

Example

- Assume a process is loaded to physical address at 32KB and all process images are 64KB. Compute the translations.

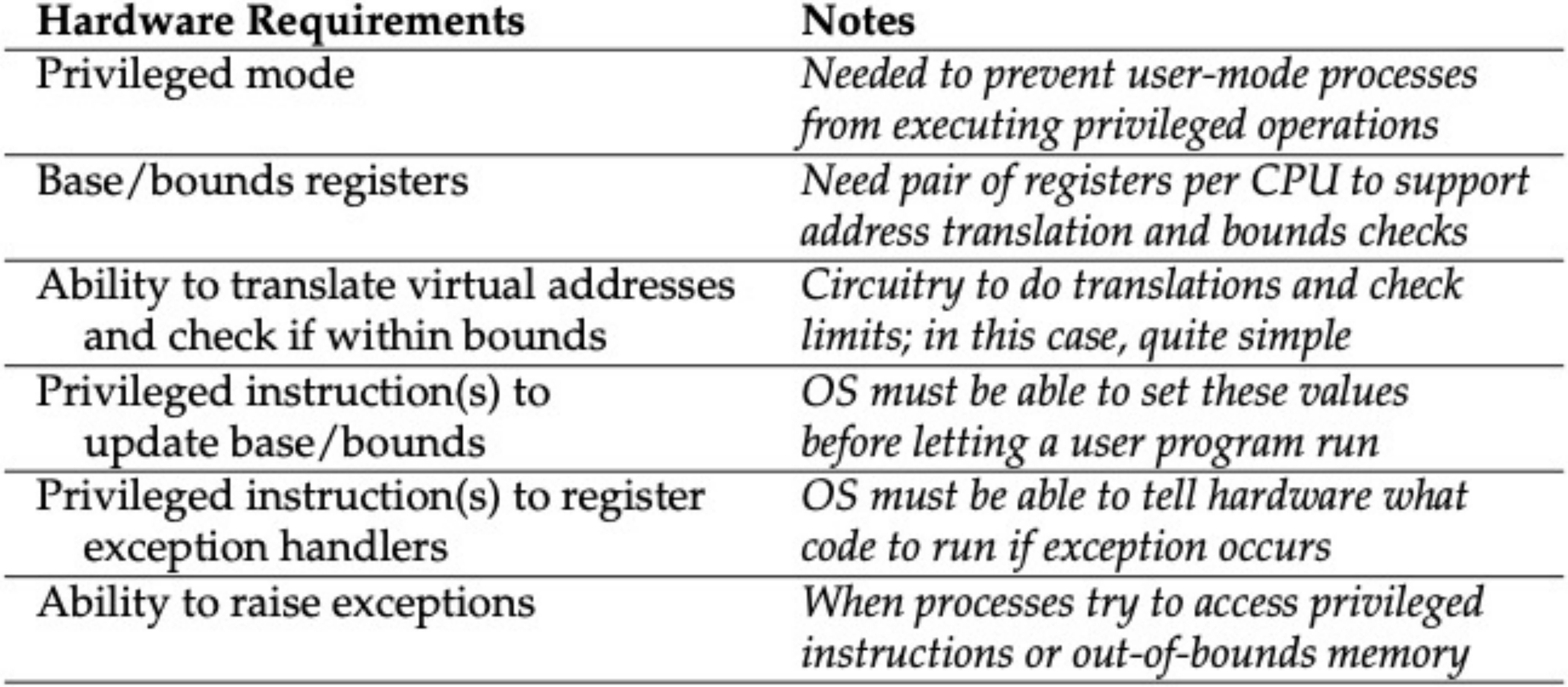
Virtual Address	Physical Address
0 Bytes	???
10 KB	???
50KB	???
70KB	???

Example

- Assume a process is loaded to physical address at 32KB and all process images are 64KB. Compute the translations

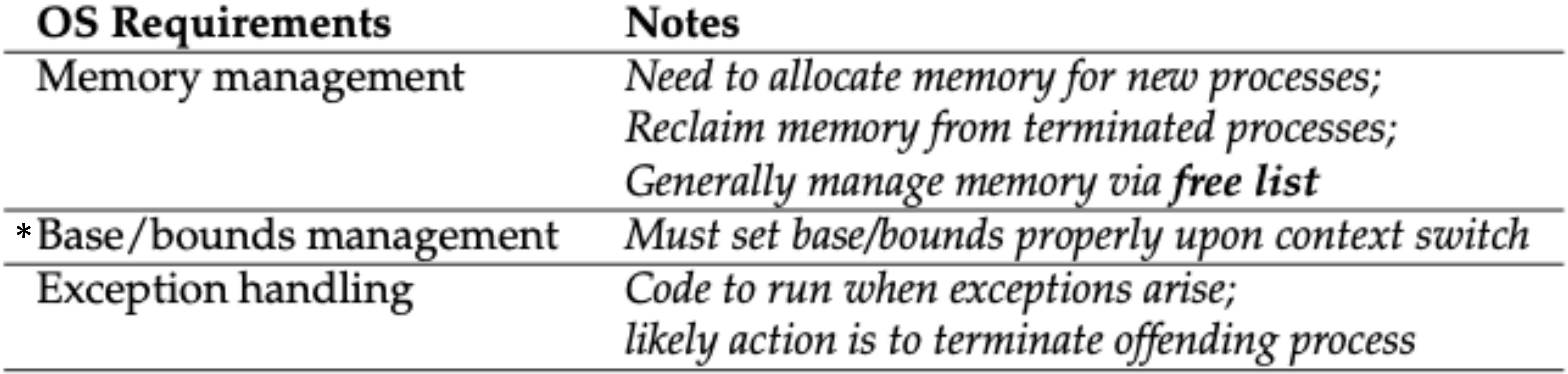
Virtual Address	Physical Address	
0 Bytes	32KB	= 0 Bytes + 32KB
10 KB	42KB	= 10KB + 32KB
50KB	82KB	= 50KB + 32KB
70KB	102KB	= 70KB + 32KB **ERROR!!!**

Dynamic Relocation Hardware Requirements



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*

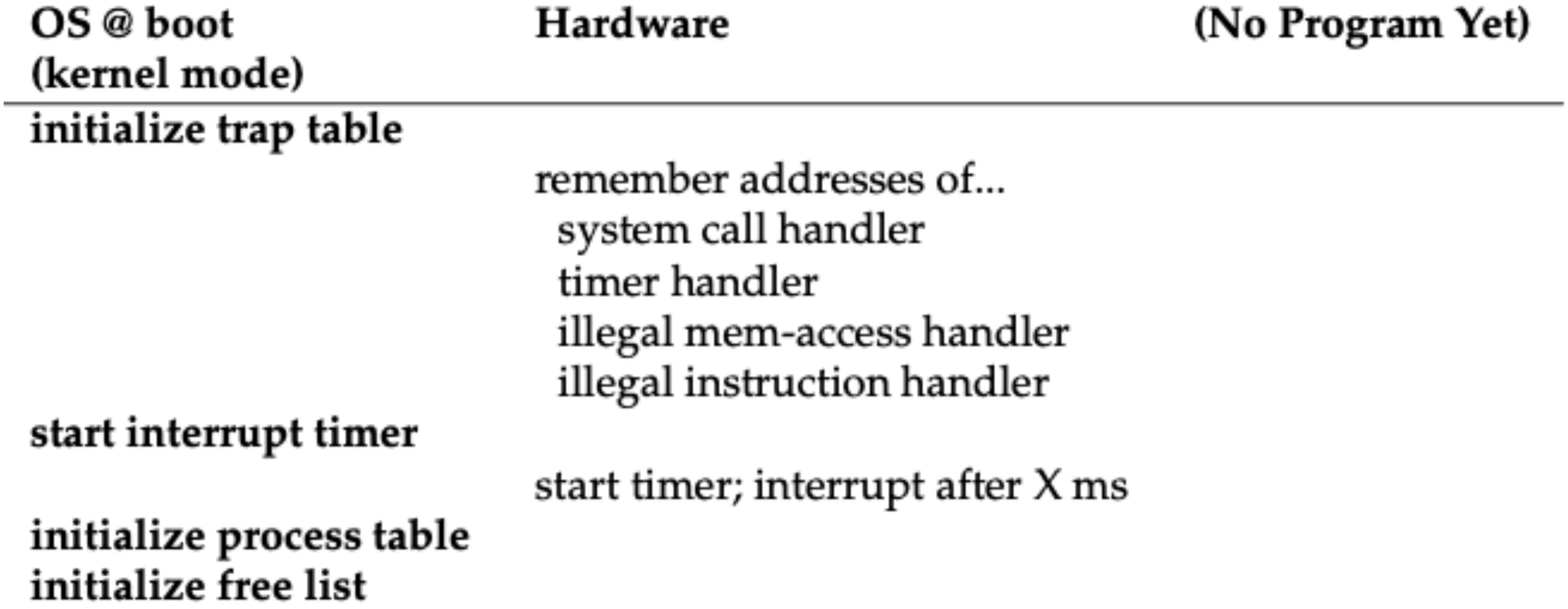
OS Dynamic Relocation Requirements



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*

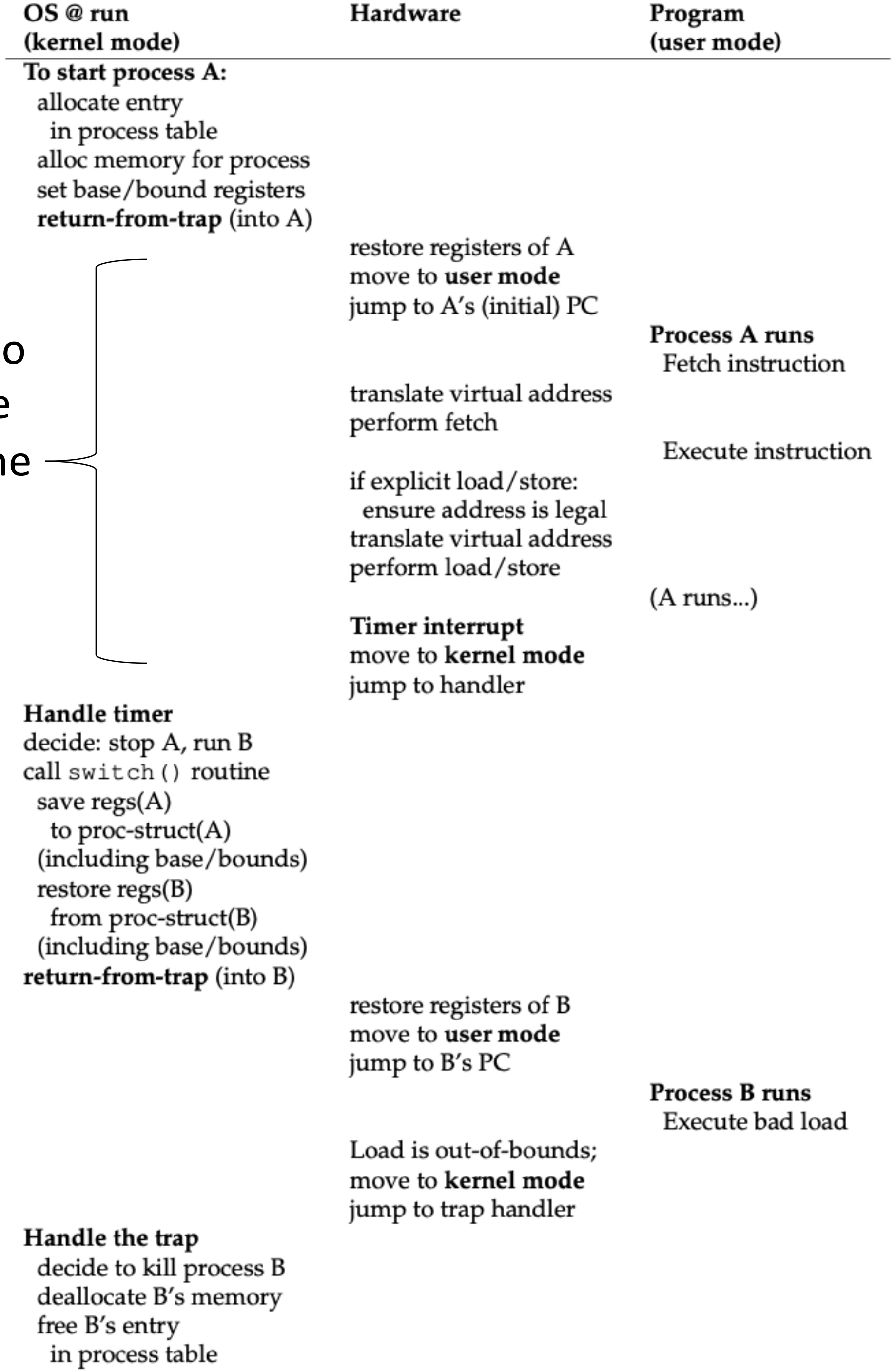
*Base and bounds are not per process; they are hardware registers per CPU. The OS needs to update the registers with the correct values in the process control block (PCB) when switching processes

Dynamic Relocation with LDE



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**		

Example



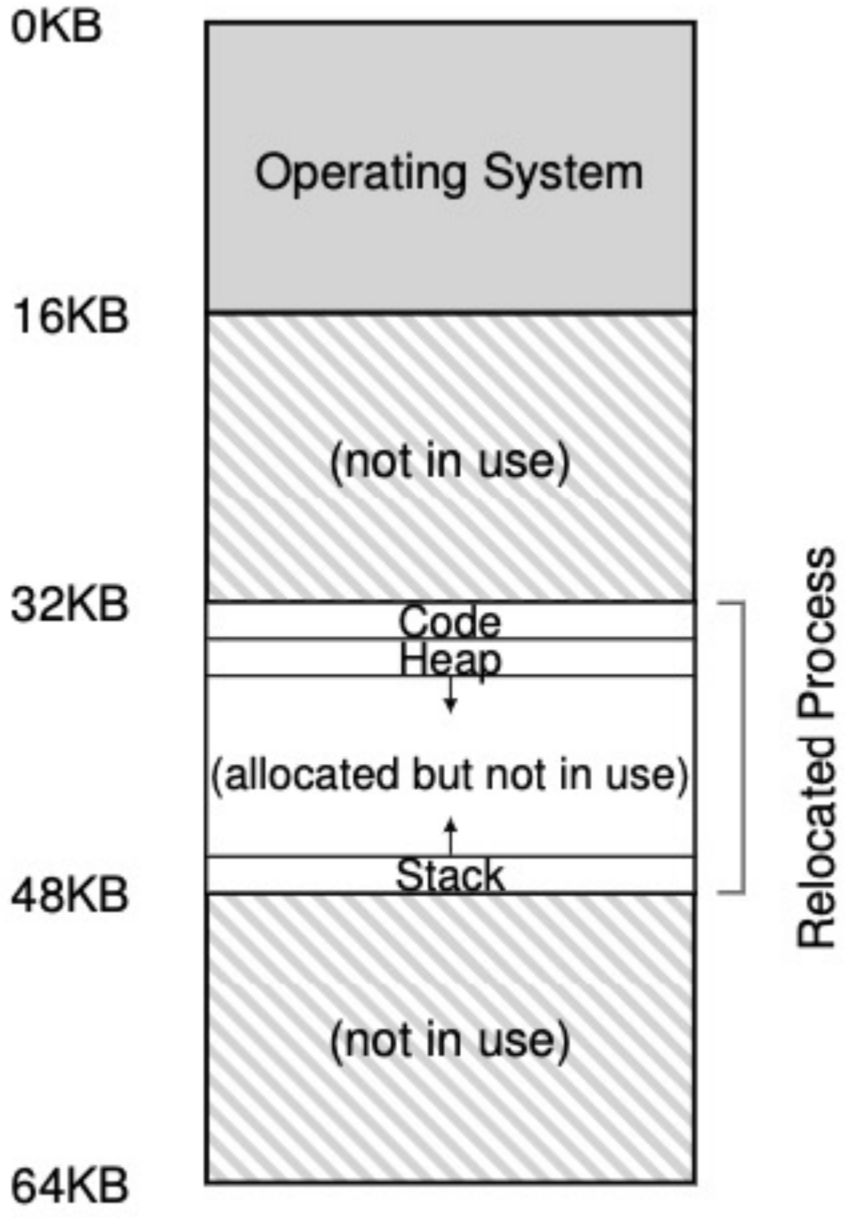
OS @ run (kernel mode)	**Hardware**	**Program (user mode)**
To start process A: allocate entry in process table alloc memory for process set base/bound registers **return-from-trap** (into A)		
	restore registers of A move to **user mode** jump to A's (initial) PC	
		Process A runs Fetch instruction
	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	
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		

The OS does not need to get involved here as the hardware can handle the address translation. Efficient!

OS steps in when there is an issues to resolve.

Next Time…

- We aren’t making the best use of our limited memory resources
  - Internal fragmentation (space between stack and heap) is wasted
- We don’t necessarily want all processes to be the same size
- Can we do better…tune in and find out!


0KB
16KB
32KB
48KB
64KB
Operating System
(not in use)
Code
Heap
(allocated but not in use)
Stack
(not in use)
Relocated Process