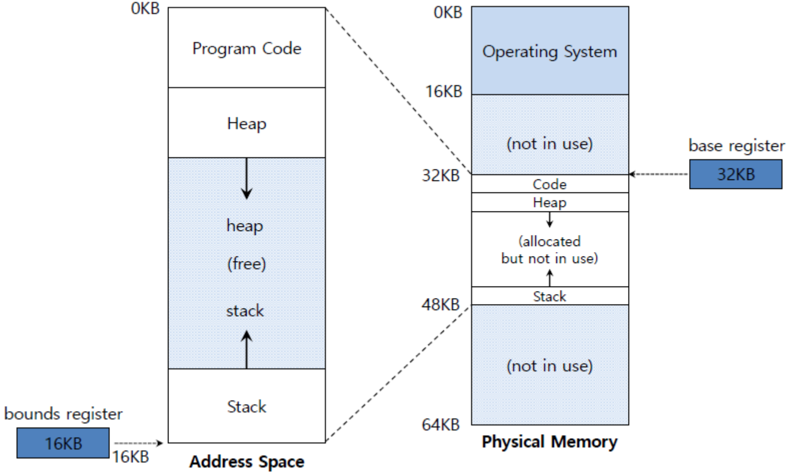
IT308: Operating Systems

Virtual memory: Segmentation

Base and Bounds

- Base register: smallest physical address (or starting location)
- Bound register: size of this process' virtual address space



Question

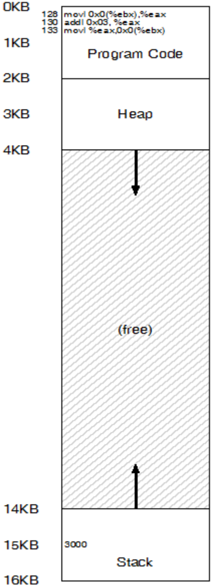
- What entity should do the address translation with base register?
- What entity should modify the base register?

Internal Fragmentation

- BOUND determines the max amount of memory available to a process
- How much memory do we allocate?
  - Empty space leads to internal fragmentation
- What if we don't allocate enough?
  - Increasing BOUND after the process is running doesn't help (why?)

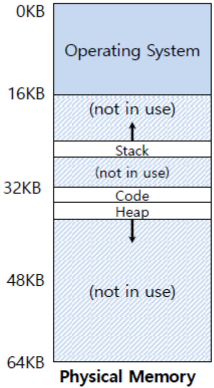
A big chunk is free

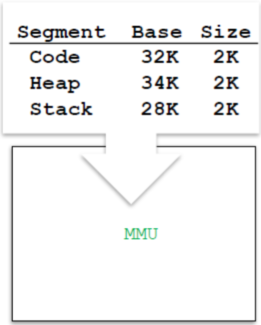
- There is a big chunk of unused memory in the middle of the address space!
- Can it be allocated to another process?
- Why not keep a base and bounds for each segment of memory (code,heap,stack)



```
128 movl 0x0(%ebx),%eax
130 addl 0x03, %eax
133 movl %eax,0x0(%ebx)
```

Segmentation Example

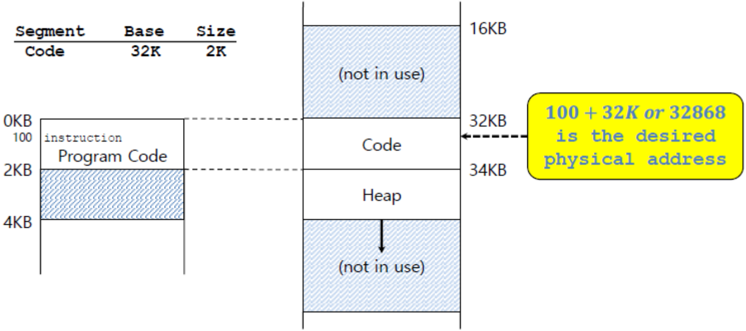




Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

Address Translation

- The code segment starts at virtual address 0.
  - The offset of virtual address 100 is 100.

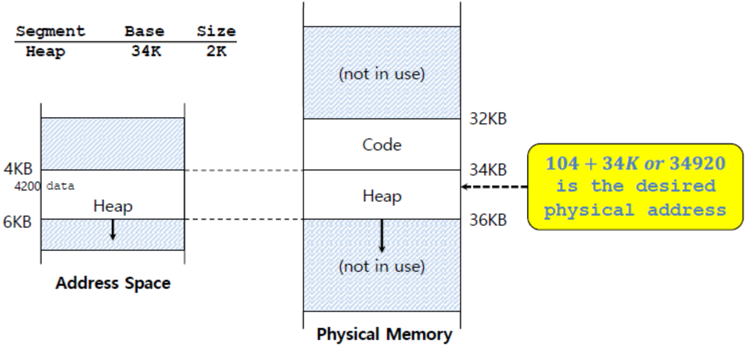


Segment	Base	Size
Code	32K	2K

$100 + 32K$ *or* 32868 is the desired physical address

Address Translation

- Virtual address + base is not the correct physical address!
- The heap segment starts at virtual address 4096.
    - The offset of virtual address 4200 is 104.



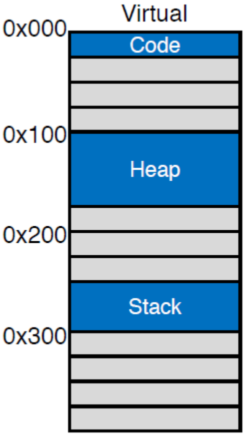
Segment	Base	Size
Heap	34K	2K

$104 + 34K$ or 34920 is the desired physical address

Which segment does an address belong to?

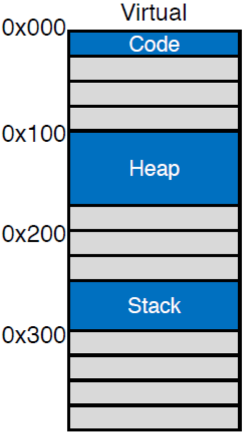
- Key idea: split virtual address into a segment and an offset
- Assume a 10-bit virtual address space
  - With the high 2-bits indicating the segment
- Assume
  - 0 ⇒ code
  - 1 ⇒ heap
  - 2 ⇒ stack

Segmentation Example



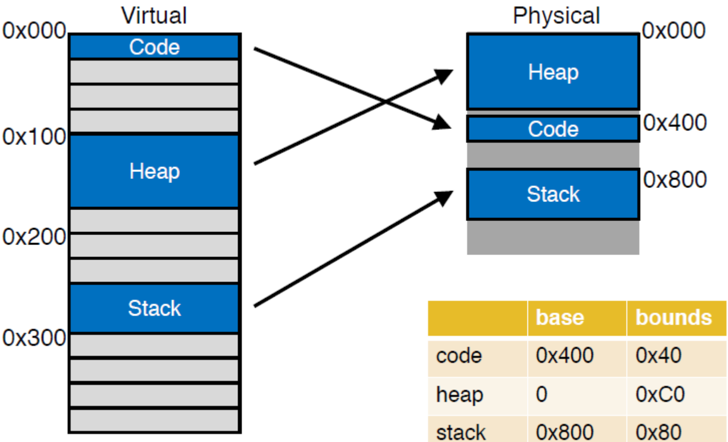
	base	bounds
code	?	?
heap	?	?
stack	?	?

Segmentation Example



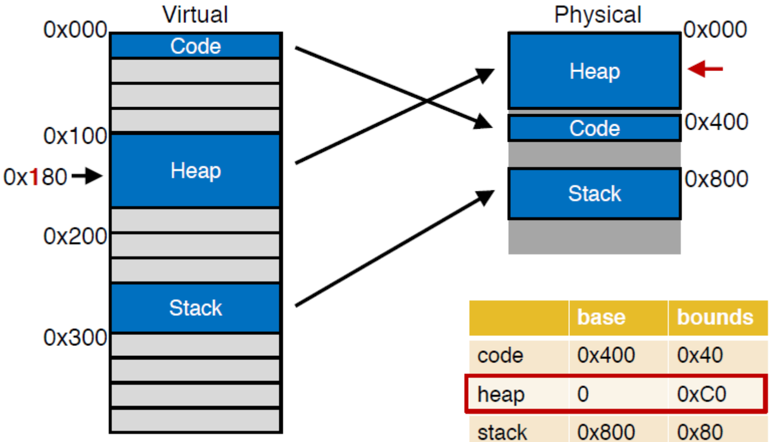
	base	bounds
code	?	0x40
heap	?	0xC0
stack	?	0x80

Segmentation Example



	base	bounds
code	0x400	0x40
heap	0	0xC0
stack	0x800	0x80

Segmentation Example



	base	bounds
code	0x400	0x40
heap	0	0xC0
stack	0x800	0x80

Segment Permissions

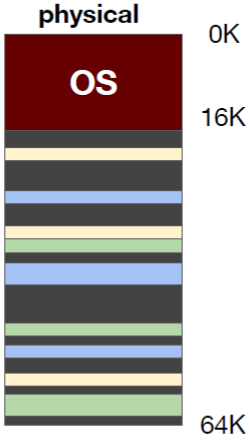
- Many CPUs (including x86) support permissions on segments
  - Read, write, and executable
- Disallowed operations trigger an exception
  - E.g., Trying to write to the code segment

Advantages of Segmentation

- Different protection for different segments
  - Read-only status for code
- Enables sharing of selected segments
- Supports dynamic relocation of each segment

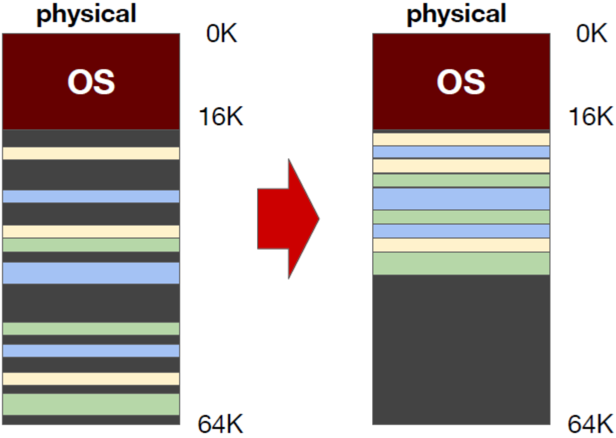
Issues with Segmentation

- OS need to able to find free space to relocate new processes' address spaces.
- After a few relocations the physical memory may look like this.
- Finding new free spaces becomes increasingly difficult, or even impossible (when?).
- This is called external fragmentation.



Solution 1 to external fragmentation

- Copy all segments' content to a contiguous region of memory, then update the base registers of all segments.
- This is called compaction. It is expensive!



Solution 2 to external fragmentation

- Be very clever when allocating new address space.
- Using smart free-list management algorithms, e.g.
  - best-fit
  - worst-fit
  - first-fit
  - next-fit
  - buddy algorithm
- However, these algorithms do NOT guarantee eliminating external fragments. They just minimize it as much as they can.