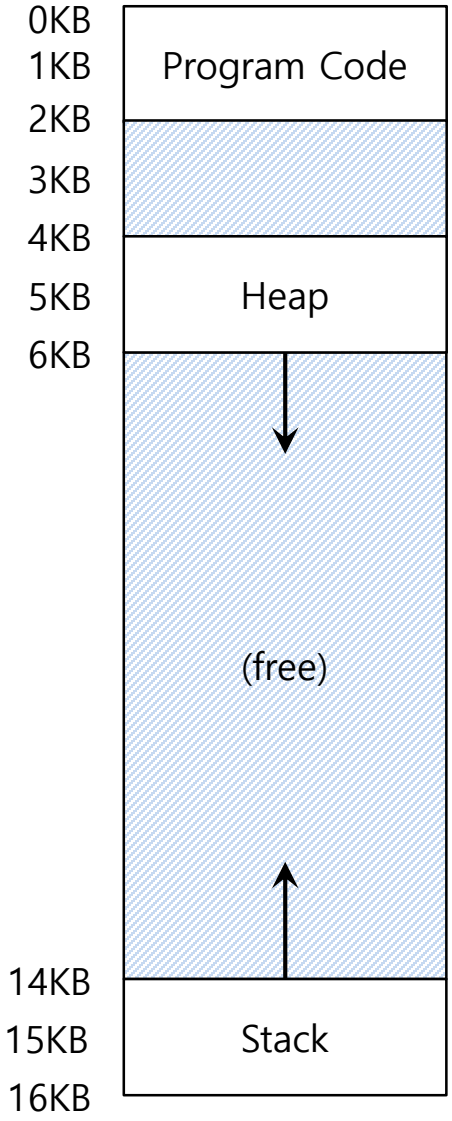
Operating Systems

16. Segmentation

Inefficiency of the Base and Bound Approach



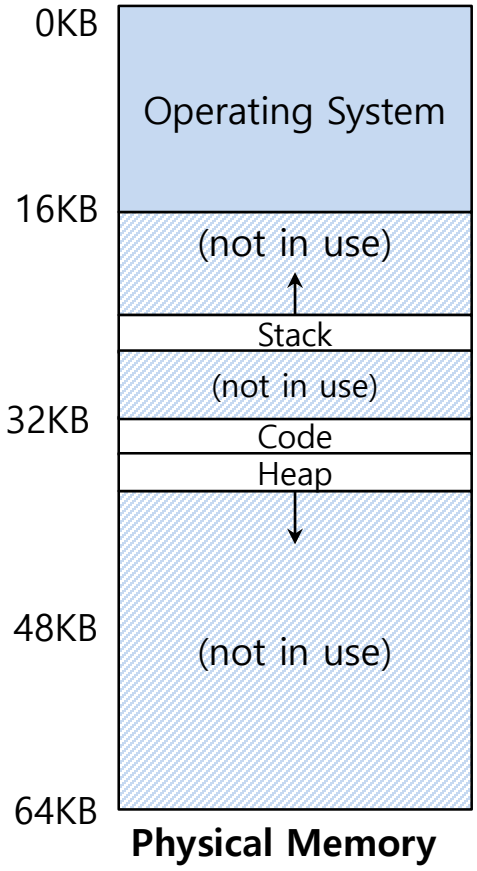
- **Big chunk of "free"** space
- "free" space **takes up** physical memory.
- Hard to run when an address space **does not fit** into physical memory

Segmentation

- Segment is just **a contiguous portion** of the address space of a particular length.
  - Logically-different segment: code, stack, heap

- Each segment can be **placed** in **different part of physical memory**.
  - **Base** and **bounds** exist **per each segment**.

Placing Segment In Physical Memory

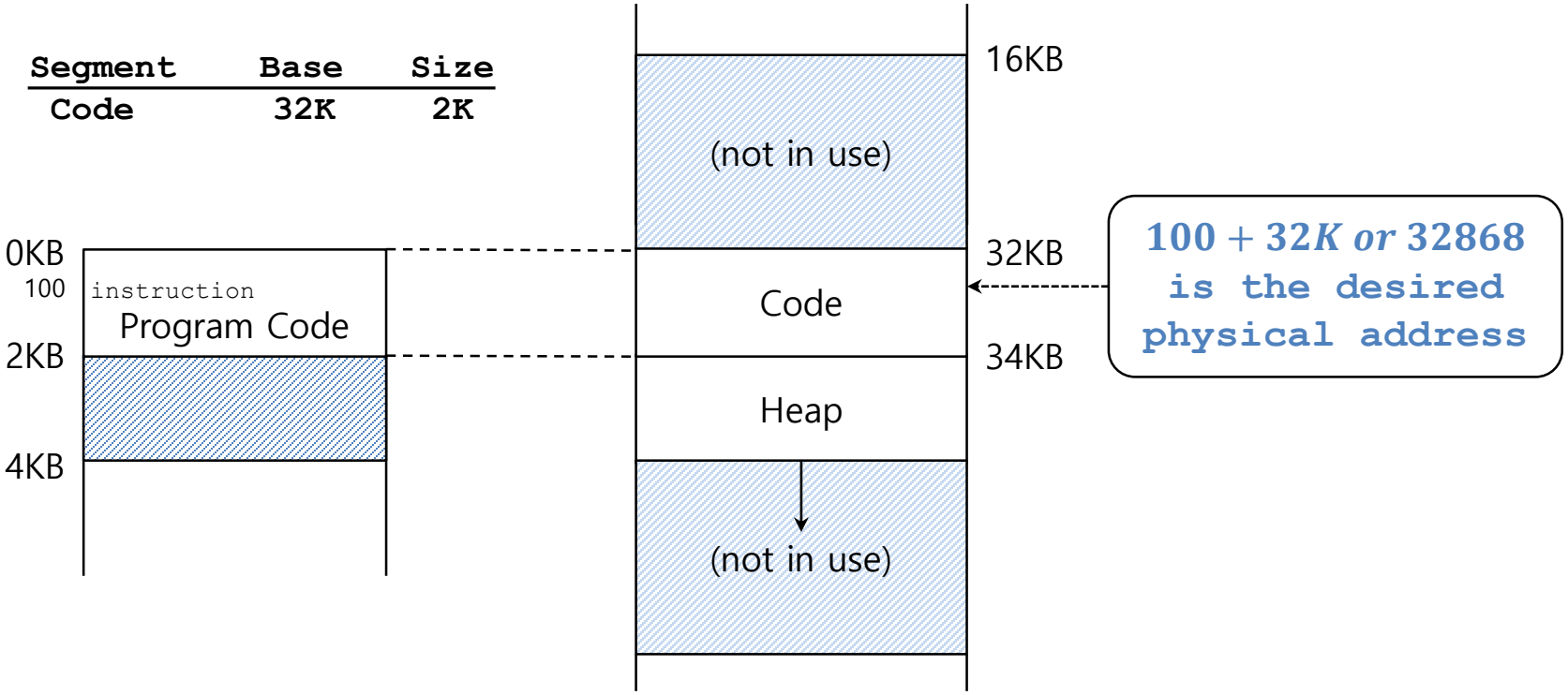


Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

Address Translation on Segmentation: code

$$physical\ address = offset + base$$

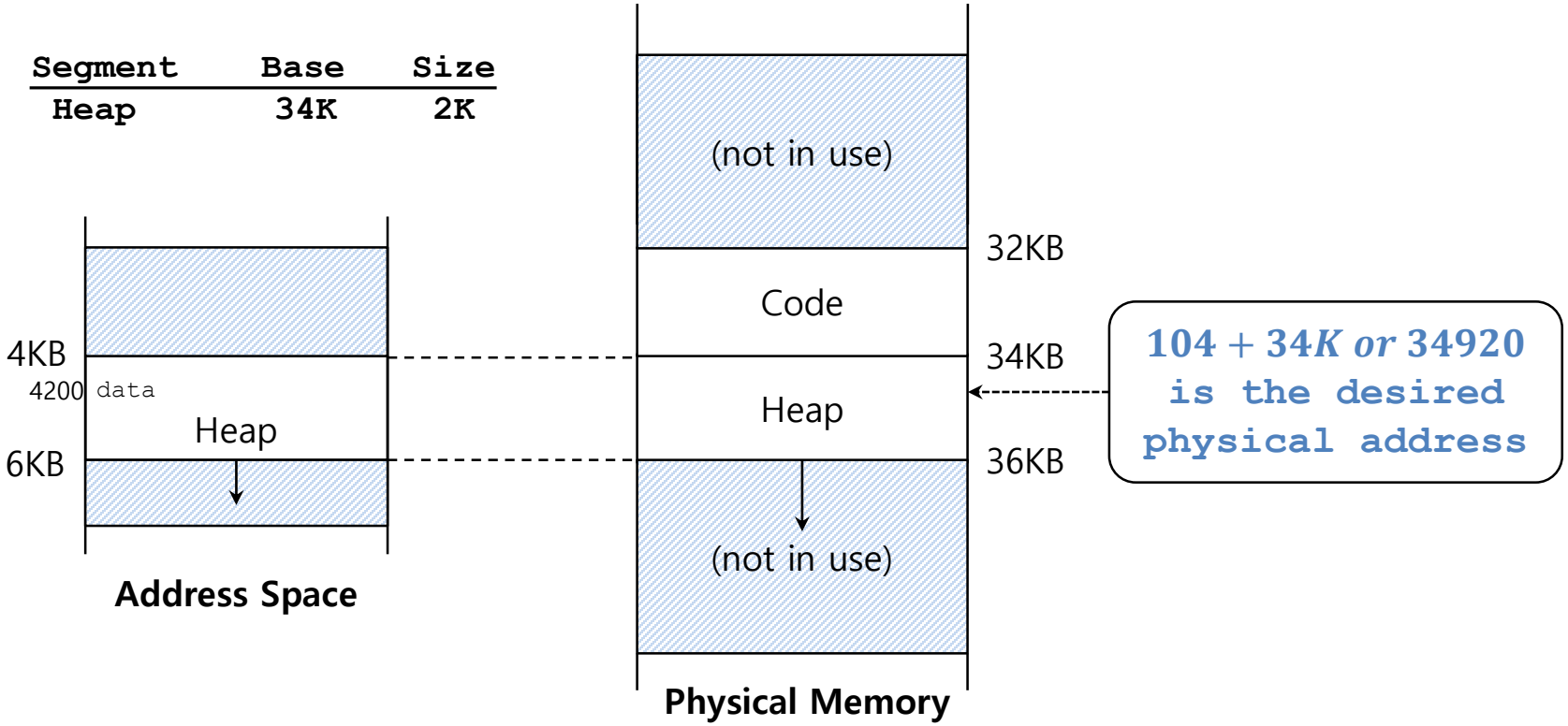
- The `offset` of virtual address 100 is `100`.
  - The code segment **starts at virtual address 0** in address space.



Address Translation on Segmentation: heap

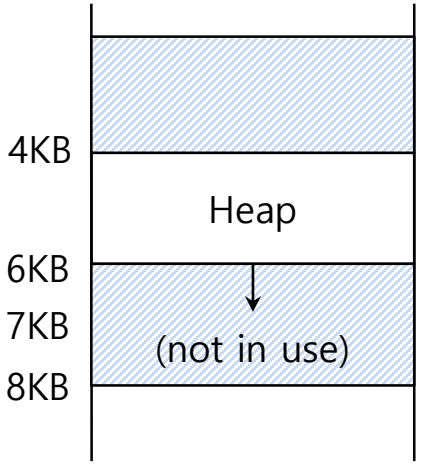
Virtual address + base is not the correct physical address.
OFFSET of *Virtual address + base* is the correct physical address.

- The `offset` of virtual address 4200 is `104`.
  - The heap segment **starts at virtual address 4096** in address space.



Segmentation Fault or Violation

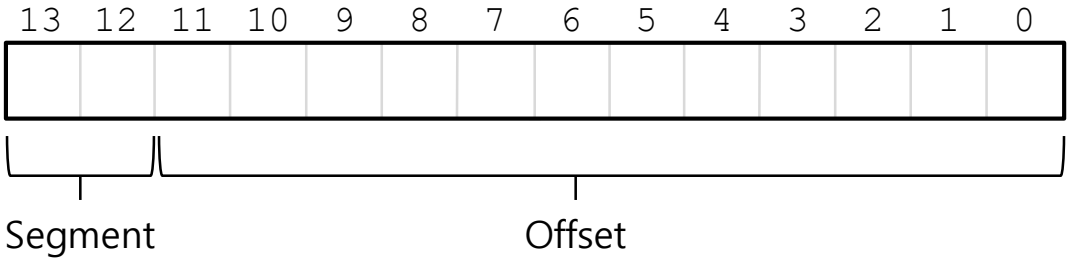
- If an **illegal address** such as 7KB which is beyond the end of heap is referenced, the OS occurs **segmentation fault**.
  - The hardware detects that address is **out of bounds**.



Address Space

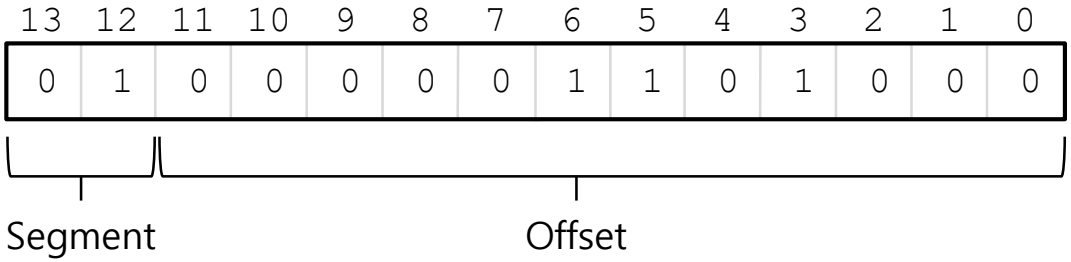
Referring to Segment

- **Explicit approach**
  - Chop up the address space into segments based on the **top few bits** of virtual address.



- Example: virtual address 4200 (01000001101000)

Segment	bits
Code	00
Heap	01
Stack	10
-	11



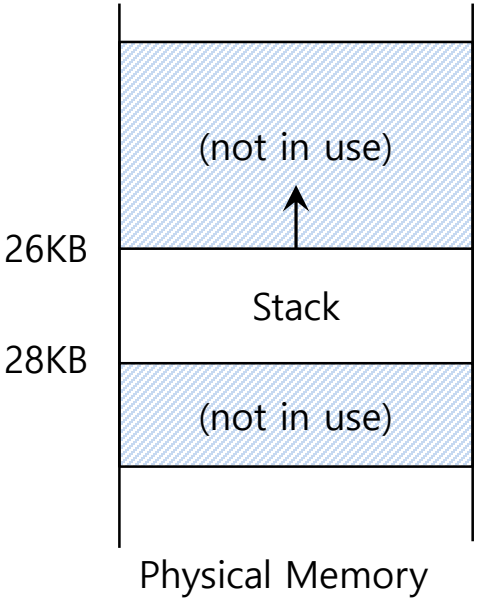
Segment selection

```
1   // get top 2 bits of 14-bit VA
2   Segment = (VirtualAddress & SEG_MASK) >> SEG_SHIFT
3   // now get offset
4   Offset = VirtualAddress & OFFSET_MASK
5   if (Offset >= Bounds[Segment])
6        RaiseException(PROTECTION_FAULT)
7   else
8        PhysAddr = Base[Segment] + Offset
9        Register = AccessMemory(PhysAddr)
```

- SEG_MASK = 0x3000(11000000000000)
- SEG_SHIFT = 12
- OFFSET_MASK = 0xFFF (00111111111111)

Referring to Stack Segment

- Stack grows **backward**.
- **Extra hardware support** is need.
  - The hardware checks which way the segment grows.
  - 1: positive direction, 0: negative direction



Segment Register(with Negative-Growth Support)

Segment	Base	Size	Grows Positive?
Code	32K	2K	1
Heap	34K	2K	1
Stack	28K	2K	0

Support for Sharing

- Segment can be **shared between address** space.
  - **Code sharing** is still in use in systems today.
  - by extra hardware support.
- Extra hardware support is need for form of **Protection bits.**
  - **A few more bits** per segment to indicate **permissions** of **read,** write and **execute**.

Segment Register Values(with Protection)

Segment	Base	Size	Grows Positive?	Protection
Code	32K	2K	1	Read-Execute
Heap	34K	2K	1	Read-Write
Stack	28K	2K	0	Read-Write

Fine-Grained and Coarse-Grained segmentation

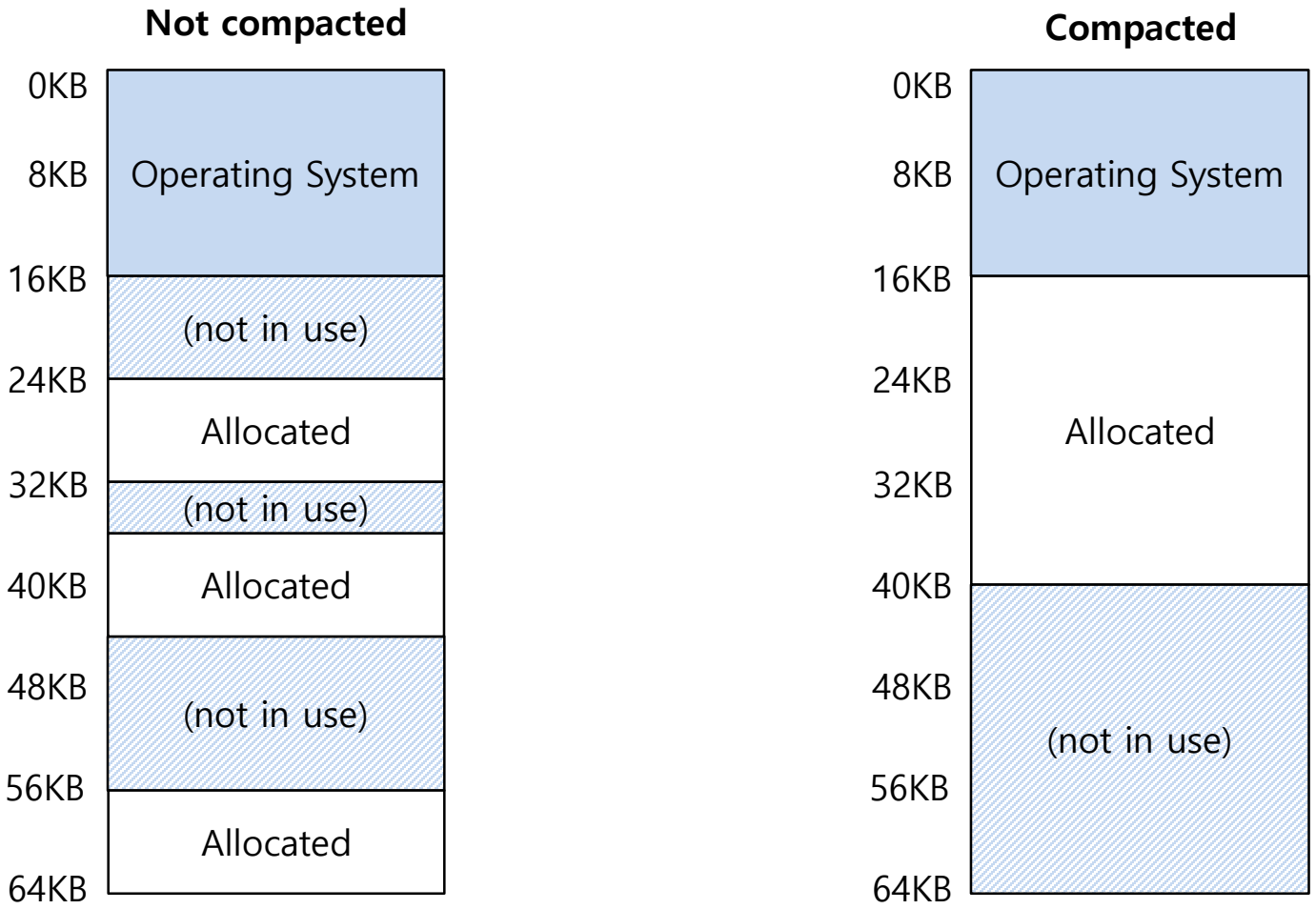
- **Coarse-Grained** means small number of segments.
  - e.g., code, heap, stack.
- **Fine-Grained** segmentation allows **more flexibility** for address space in some early system.
  - To support many segments, Hardware support with a **segment table** is required.

OS support: Fragmentation

- **External Fragmentation**: little holes of **free space** in physical memory that is too small for allocating segment.
  - There is **24KB free**, but **not in one contiguous** segment.
  - The OS **cannot** satisfy the **20KB request**.

- **Compaction**: **rearranging** the exiting segments in physical memory.
  - Compaction is **costly**.
    - **Stop** running process.
    - **Copy** data to somewhere.
    - **Change** segment register value.

Memory Compaction



History of segmentation

- In early days, OS used segmentation.
  - Burroughs B5000 (first commercial machine with virtual memory)
  - IBM AS/400
  - Intel 8086, 80286
- 80386 and later Intel CPU's support paging.
- X86-64 does not use segmentation any more in 64bit mode
  - CS,SS,DS and ES are forced to 0 and 2^24..

Summary

- Segmentation can better support sparse address spaces.
- It is also fast as the overheads of translation are minimal.
- Sharing (such as code) is easy.
- Issues
  - External fragmentation issue
  - Sparse segment