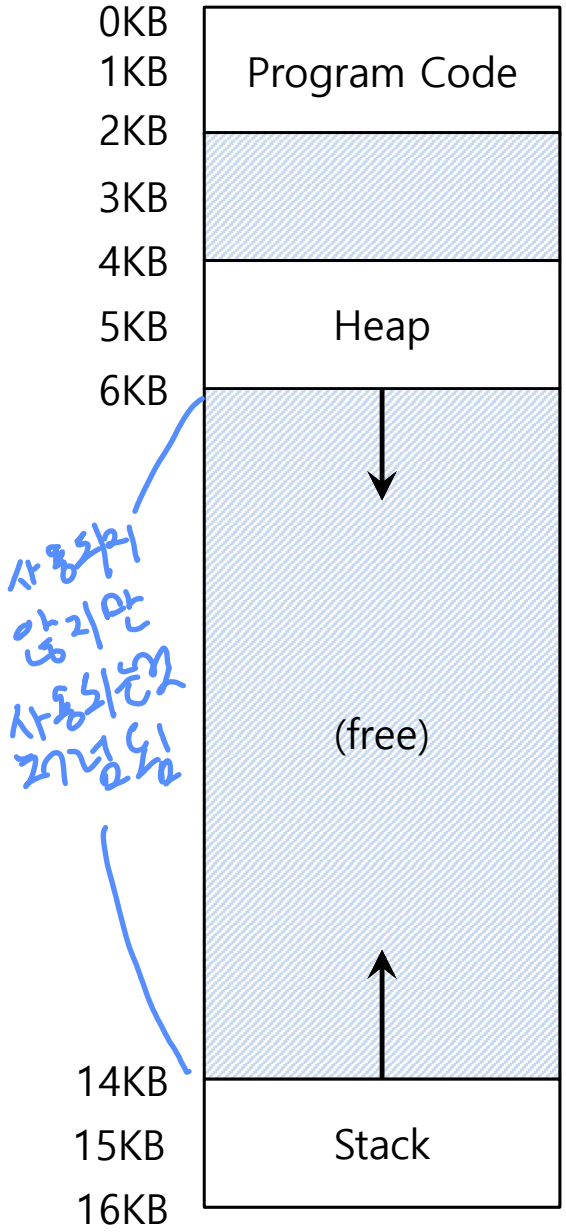
Operating System: Segmentation

Sang Ho Choi (shchoi@kw.ac.kr)
School of Computer & Information Engineering
KwangWoon University

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

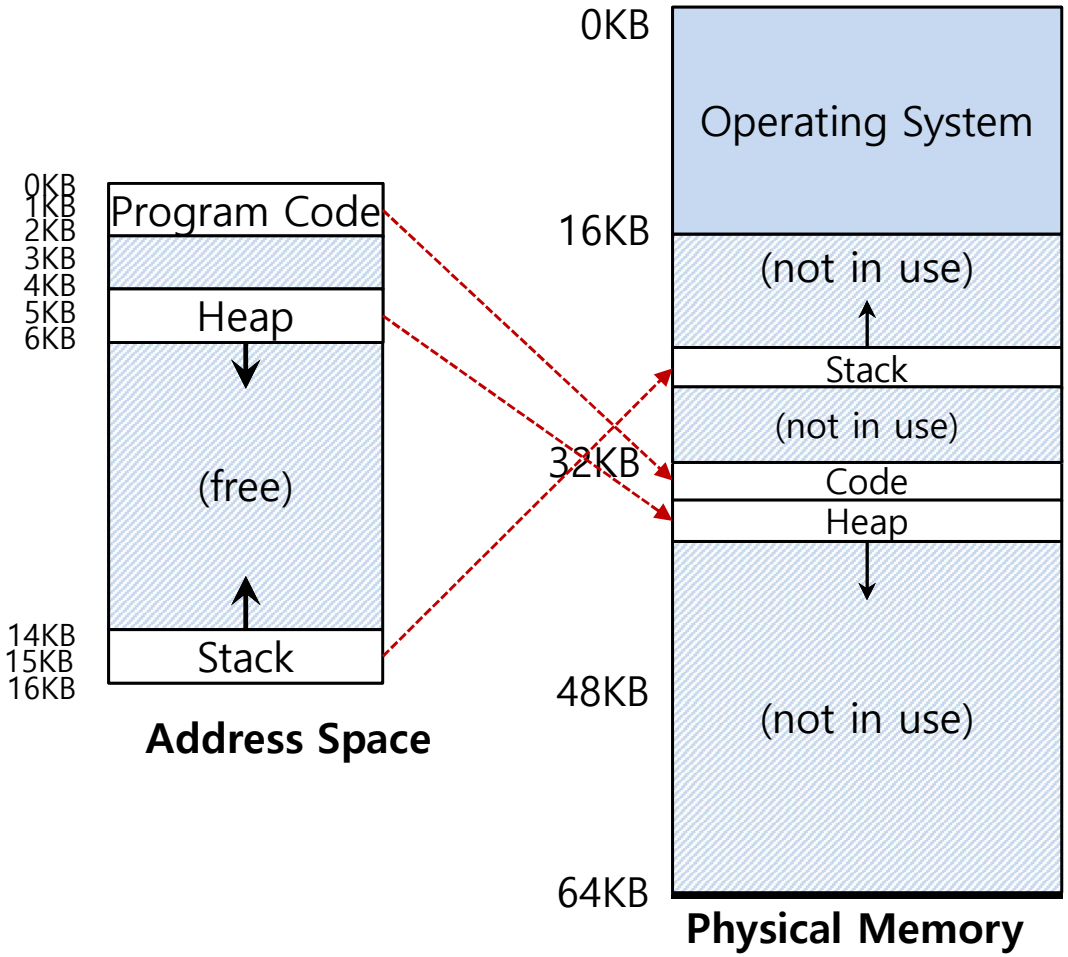
유연성, 효율성 ↓

Segmentation

- Divide address space into logical segments
  - Segment is just a contiguous portion of the address space of a particular length
  - Each segment corresponds to logical entity in address space
  - Logically-different segment: **code, stack, heap**

- Each segment can independently
  - be placed in different part of physical memory
  - grow or shrink
  - **Base** and **bounds** exist **per each segment**

Placing Segment In Physical Memory



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

$$physical\ address = offset + base$$

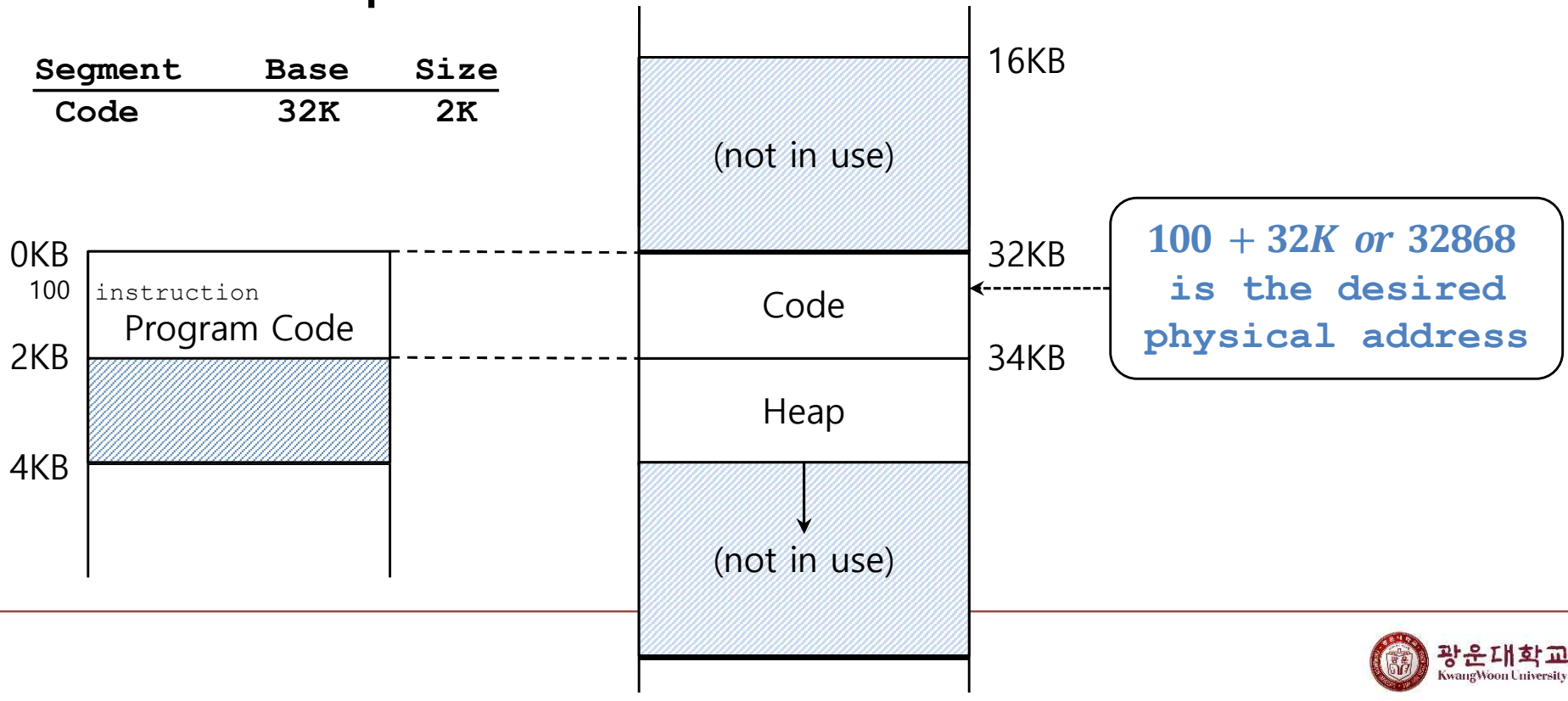
$$offset = virtual\ address - start\ address\ of\ segment$$

offset (세그먼트내의 상대적위치)

Address Translation on Segmentation

$$physical\ address = offset + base$$

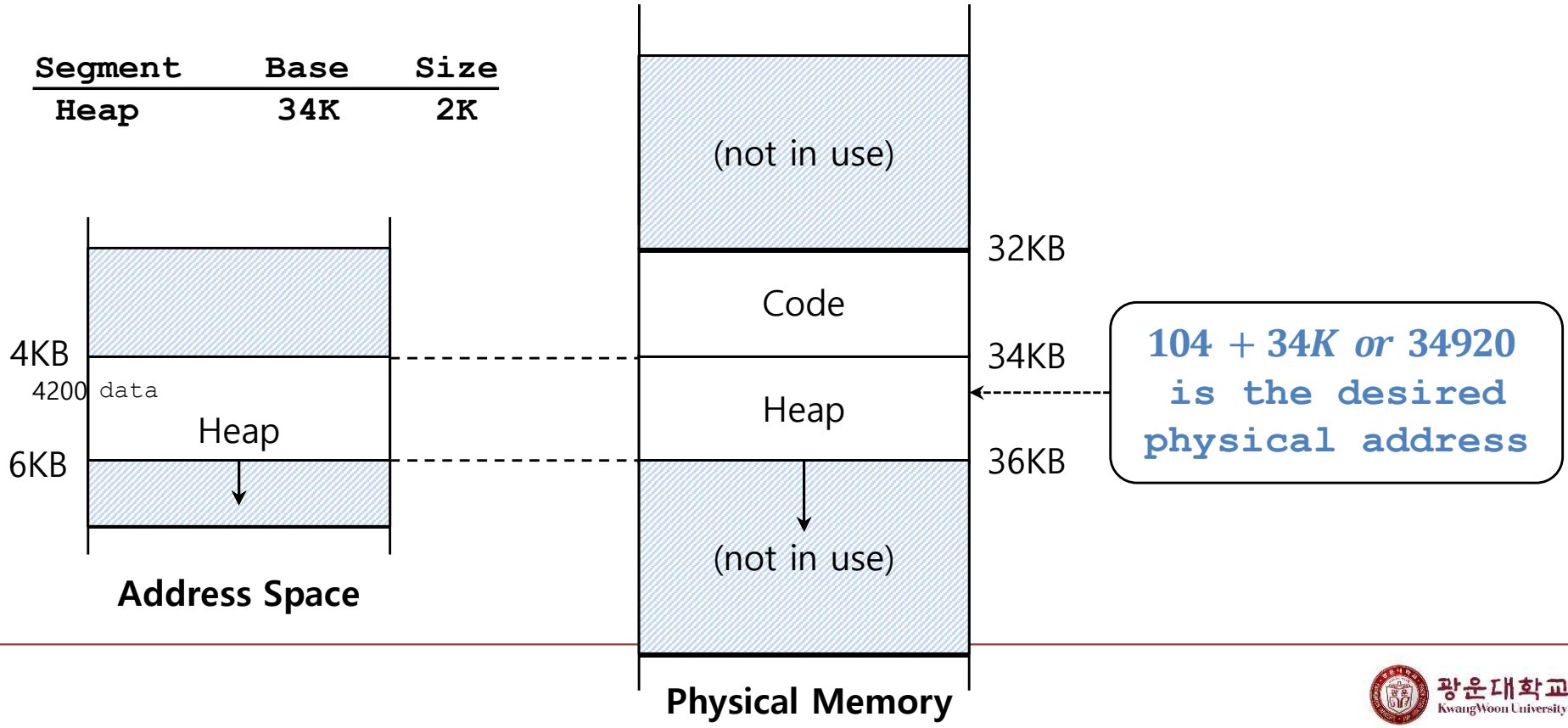
- The `offset` of virtual address 100 is `100`
  - The code segment **starts at virtual address 0** in address space



Address Translation on Segmentation (Cont.)

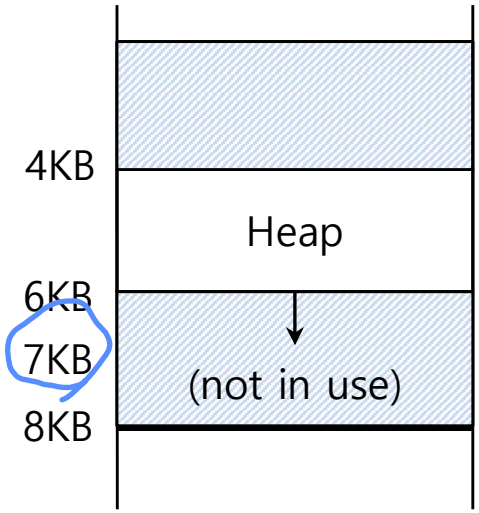
Virtual address + base is not the correct physical address

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



Segmentation Fault or Violation

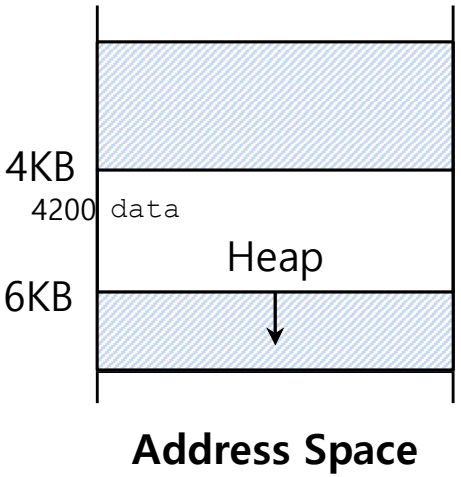
- If an illegal address such as 7KB which is beyond the end of heap is referenced, the OS occurs segmentation fault
  - The hardware detects that address is out of bounds



Address Space

Which Segment Are We Referring To?

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

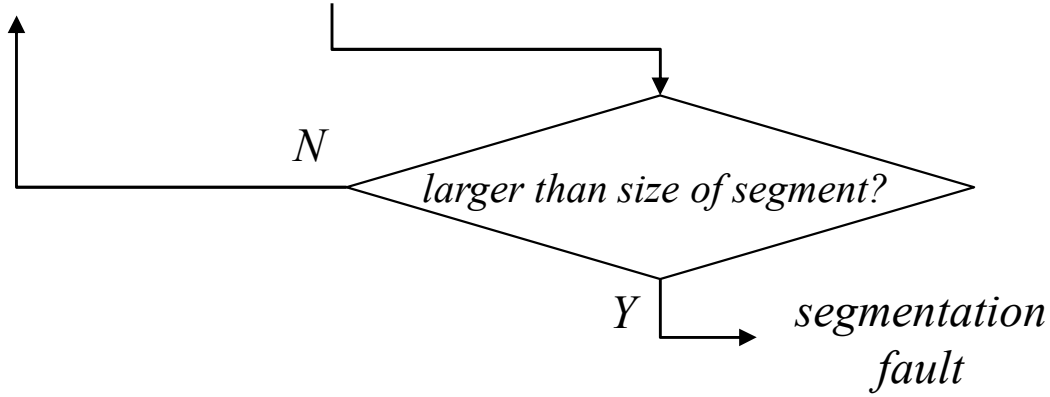


offset = virtual address - segment start address

104 = 4200 - 4096

physical address = offset + base of segment

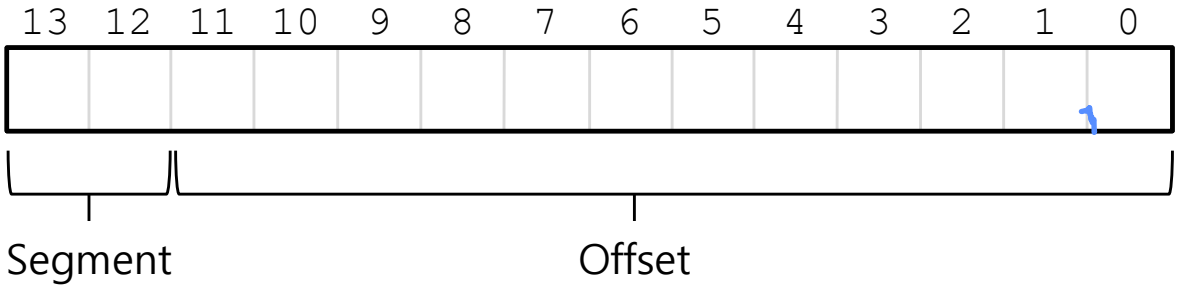
34920 = 104 + 34K



How does it know to which segment an address refers ?

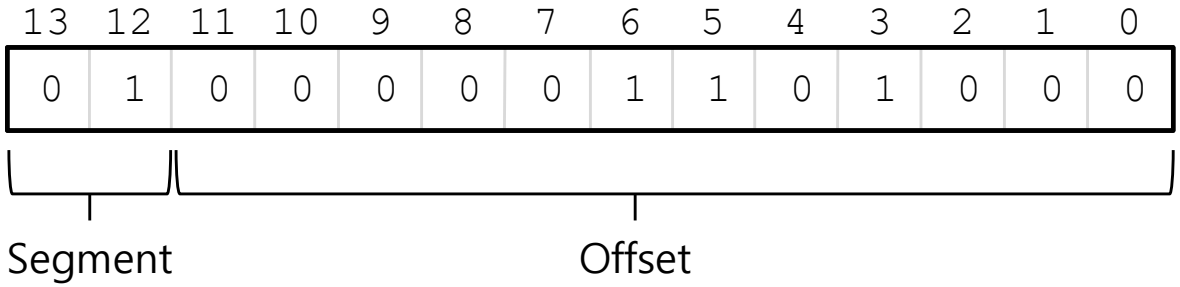
Referring to Segment

- 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
-	10
Stack	11



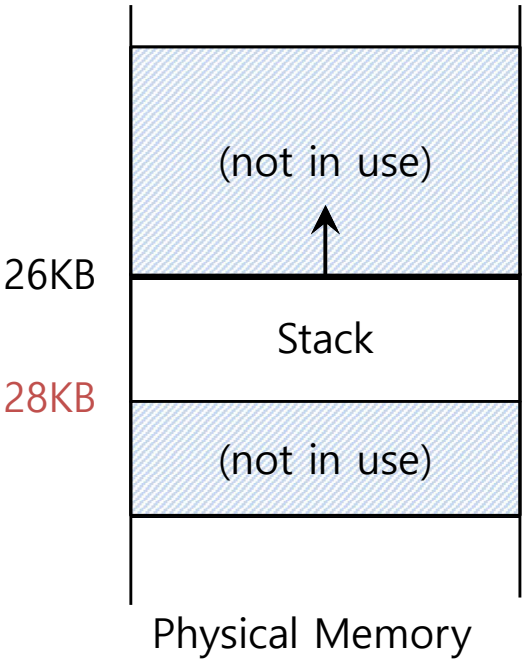
Referring to Segment (Cont.)

```
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)

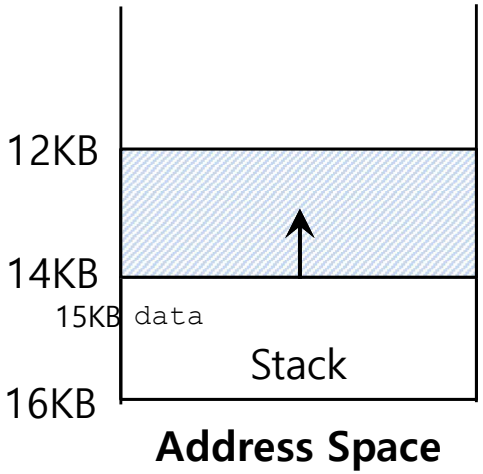
Segment	Base	Size	Grows Positive?
Code	32K	2K	1
Heap	34K	2K	1
Stack	28K	2K	0

Referring to Stack Segment (Cont.)

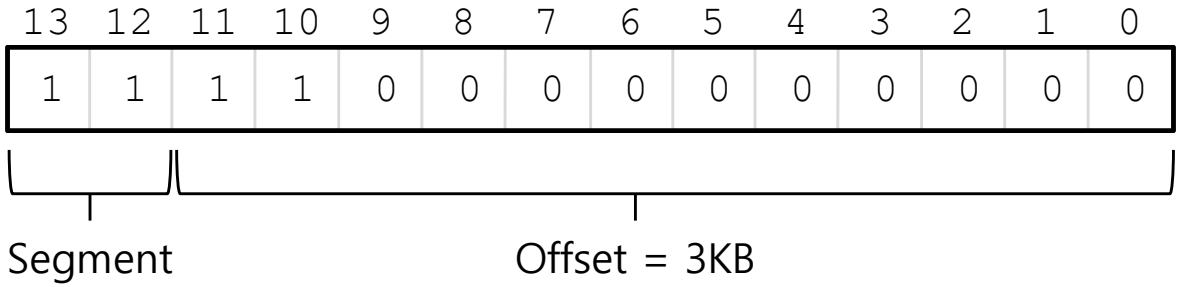
Segment Register(with Negative-Growth Support)

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

Segment	bits
Code	00
Heap	01
-	10
Stack	11



virtual address 15KB = 11 1100 0000 0000



Max. segment size = 4KB (because of 12bit offset)

Correct negative offset = 3KB – 4KB = -1KB

Physical address = 28KB – 1KB = 27KB

Support for Sharing

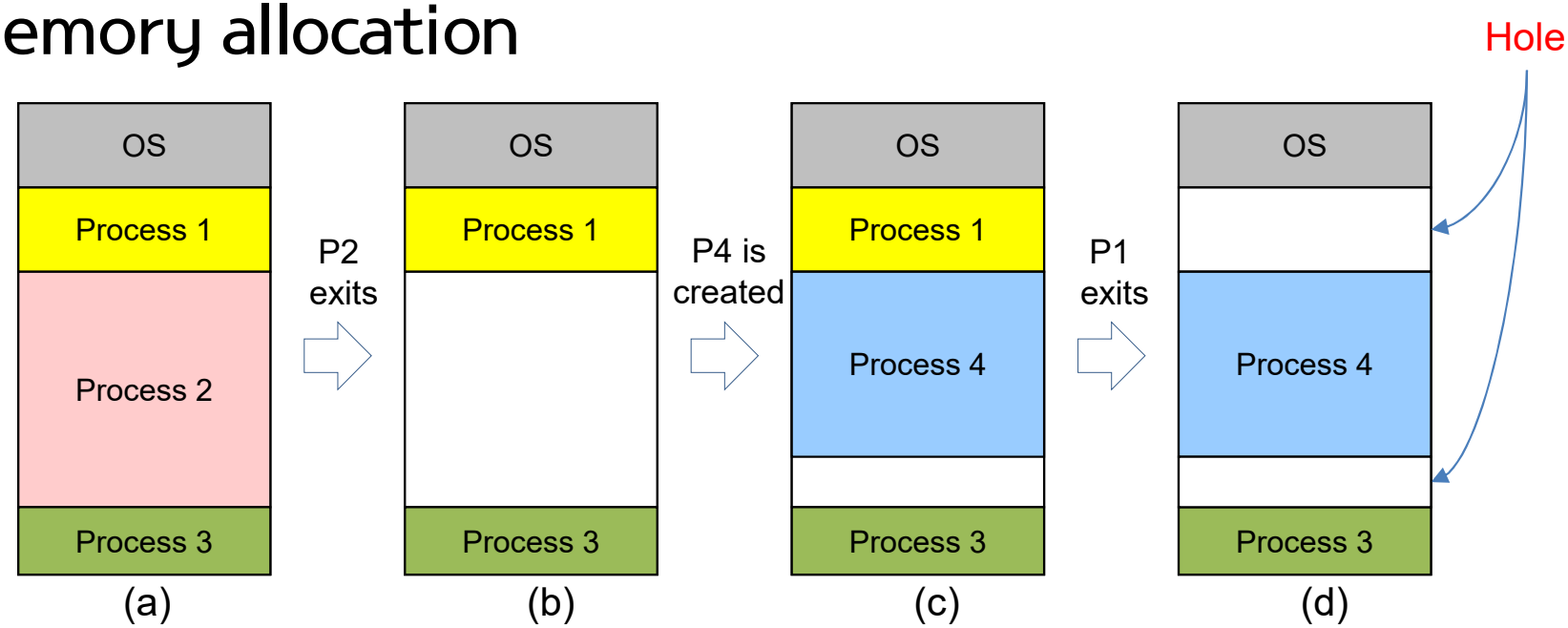
- Segment can be shared between address space
  - **Code sharing** is still in use in systems today
  - by extra hardware support
- Extra hardware support is need for form of Protection bits
  - A **few more bits** per segment to indicate **permissions** of **read**, write and **execute**

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

Fragmentation

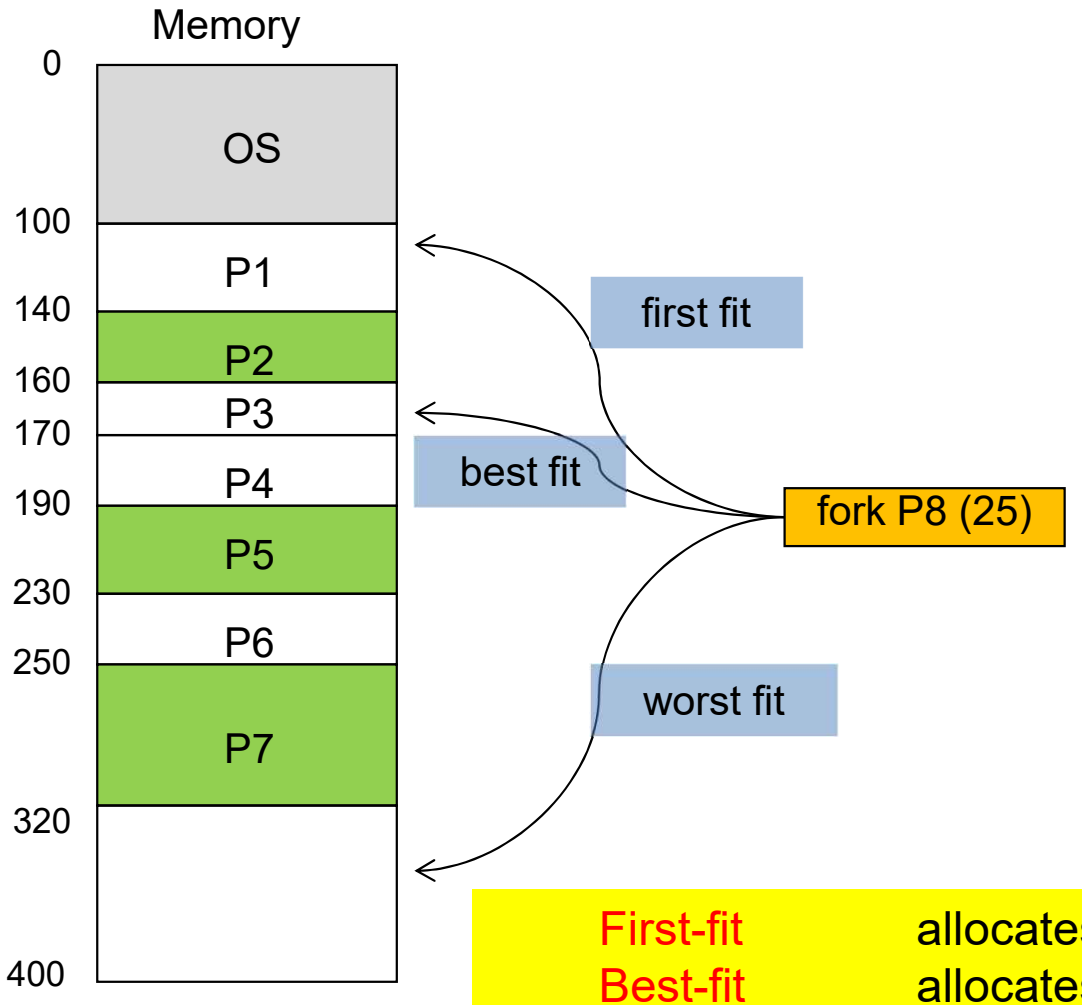
- Memory allocation

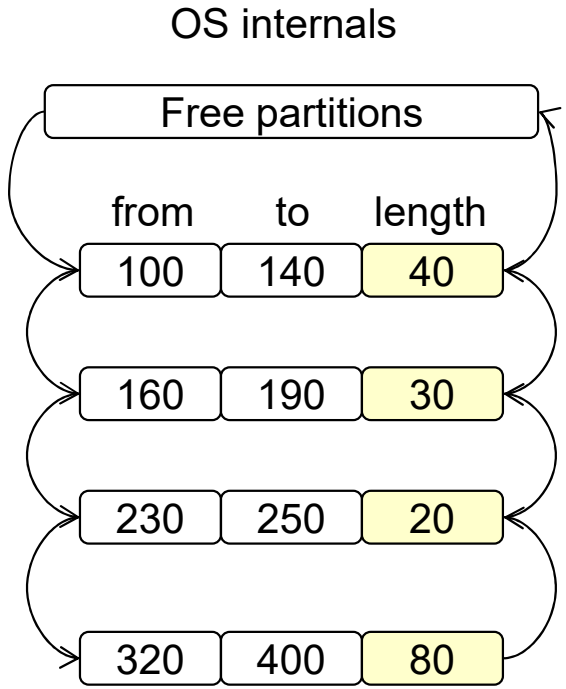


  - Hole (a large block of available memory)
    - holes of various size are scattered throughout memory
  - When a process is created,
    - it is allocated memory from a hole large enough to accommodate it
  - Operating system maintains information about
    - allocated partitions
    - free partitions (hole)

Fragmentation (Cont.)

- Dynamic Storage-Allocation Problem

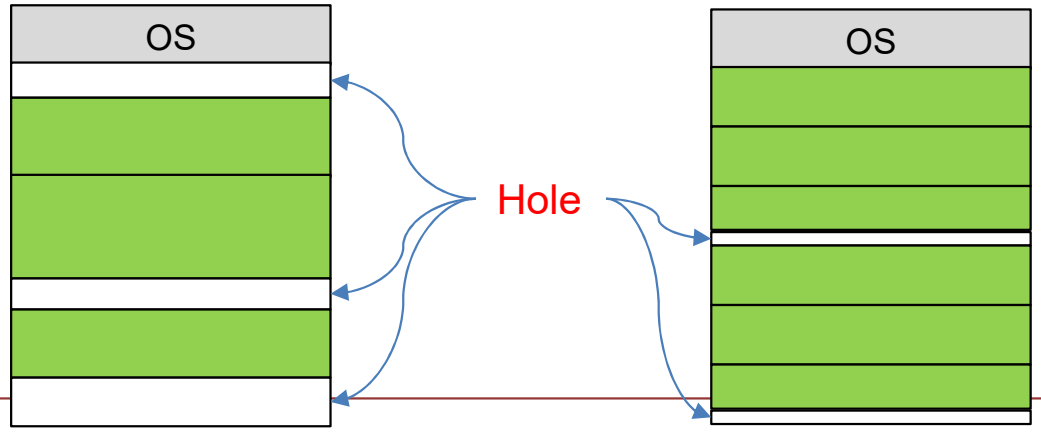




First-fit	allocates the first hole that is big enough
Best-fit	allocates the smallest hole that is big enough
Worst-fit	allocates the largest hole that is big enough

Fragmentation (Cont.)

- External Fragmentation
  - Total free memory space exists to satisfy a request, but it is not contiguous
  - Can be reduced by compaction — 홀들을 통합(아상적이만 오버헤드↑)
    - ➢ places all free memory together in one large block
- Internal Fragmentation → Paging 방식에서 나타남
  - The allocated memory may be slightly larger than requested memory
  - Size difference is internal to a partition, but is not used

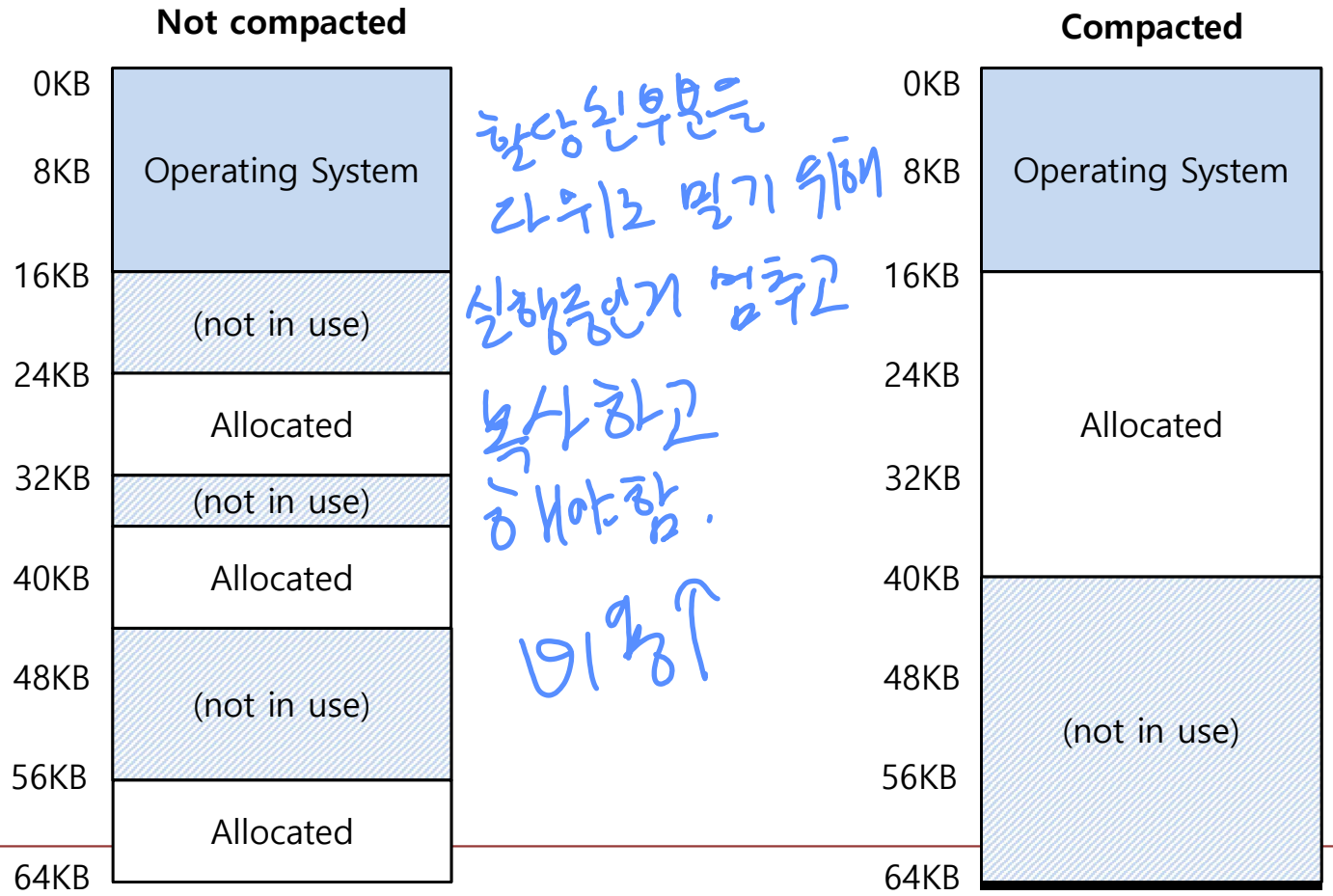


External fragmentation

Internal fragmentation

Memory Compaction

- Rearranging the exiting segments in physical memory
  - Compaction is costly
    - Stop running process, copy data to somewhere, and change register values



Segmentation: Pros

- Enables sparse allocation of address space
  - Stack and heap can grow independently
  - No internal framgmentation
- Fast, easy, and well-suited to hardware
- Easy to share segments
  - Put the same translation into base/limit pair
  - Code/data sharing at segment level (e.g. shared libraries)
- Supports dynamic relocation of each segment

Segmentation: Cons

- Each segment must be allocated contiguously
  - External fragmentation
  - May not have sufficient physical memory for large segments
- Segmentation still isn't flexible enough
  - If we have a large but sparsely-used heap all in one logical segment, the entire heap must still reside in memory