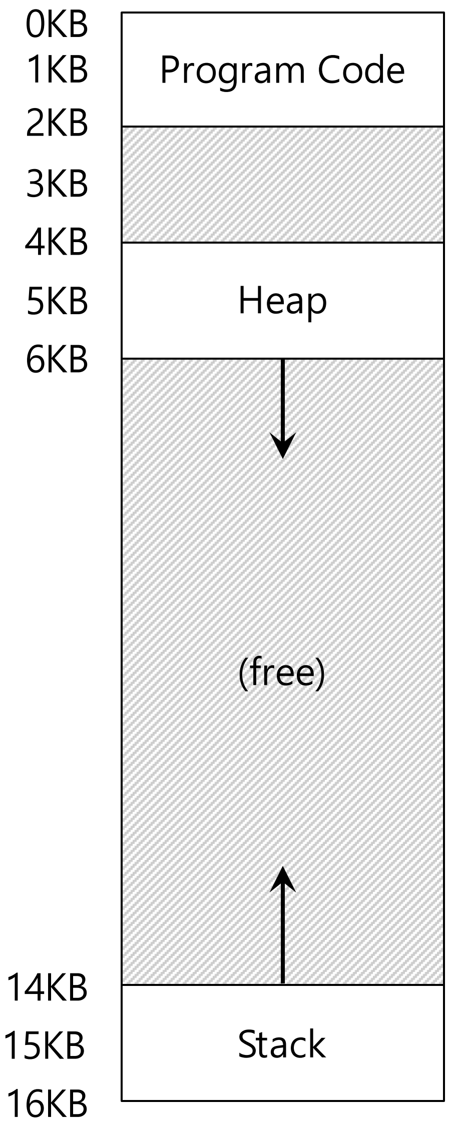
Memory Virtualization: Segmentation

Prof. Patrick G. Bridges

Why not just Base and Bound?



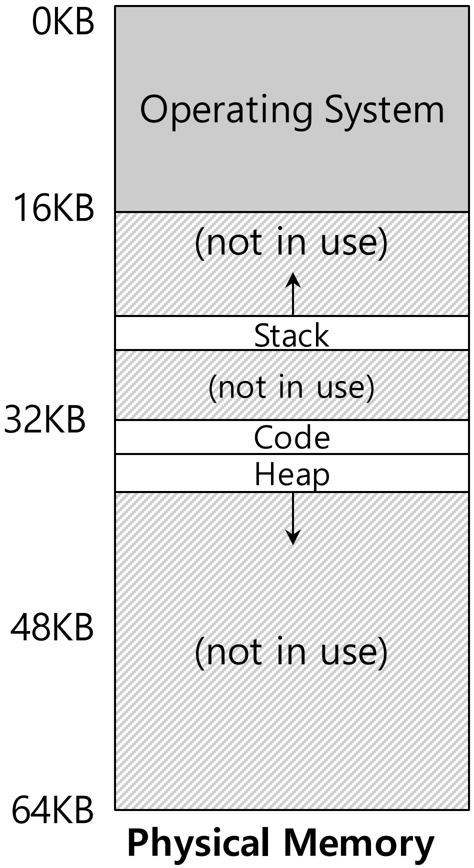
- Big chunk of "free" space
- "free" space takes up physical memory.
- Hard to run when an address space does not fit into physical memory

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



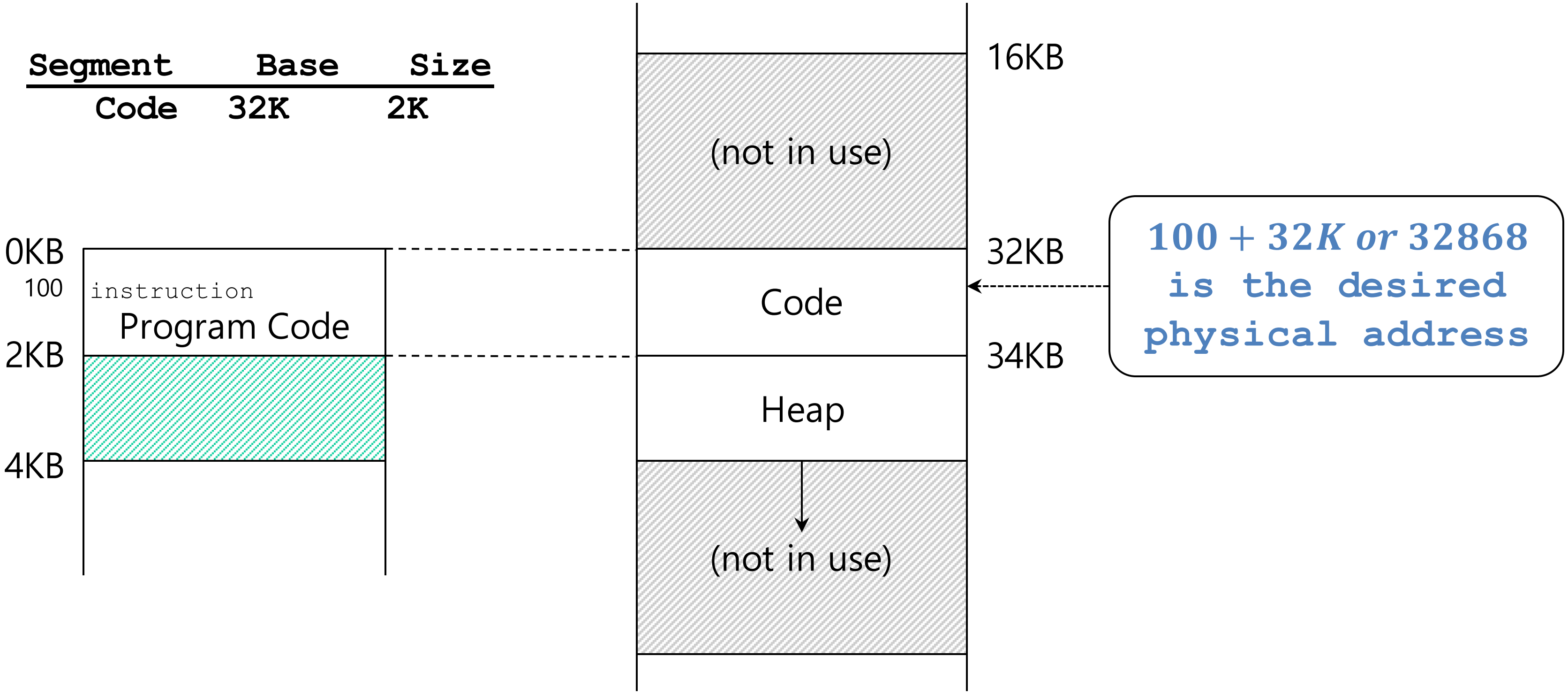
Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

Address Translation on Segmentation

$$physical\ address = offset + base$$

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

Segment	Base	Size
Code	32K	2K



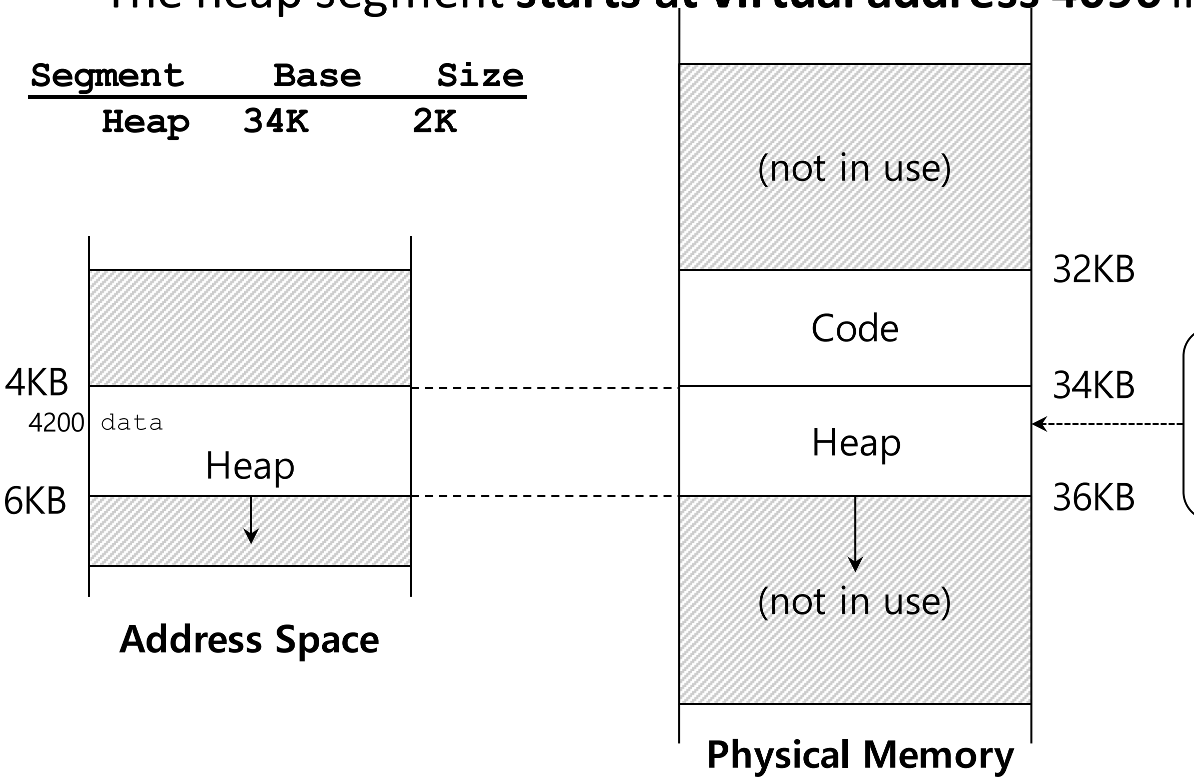
$100 + 32K$ *or* 32868 **is the desired physical address**

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.

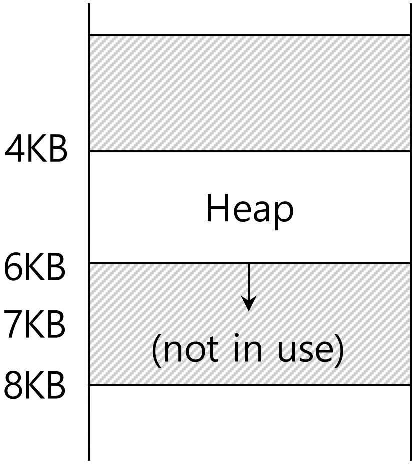
Segment	Base	Size
Heap	34K	2K



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

Segmentation Fault or Violation

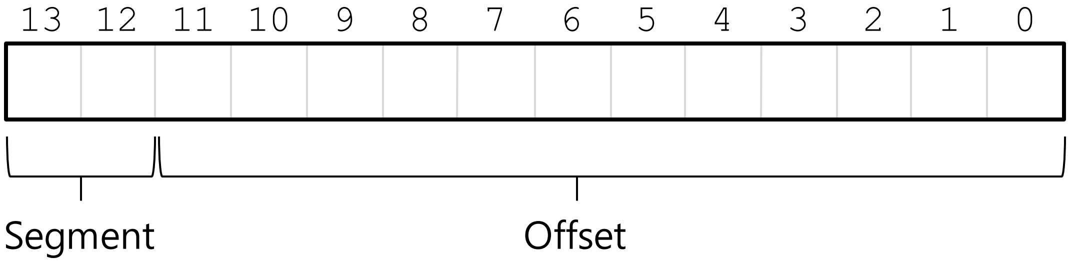
- **If an illegal address such as 7KB which is beyond the end of heap is referenced, the OS occurs segmentation fault.**
  - The hardware detects that address is **out of bounds**.



Address Space

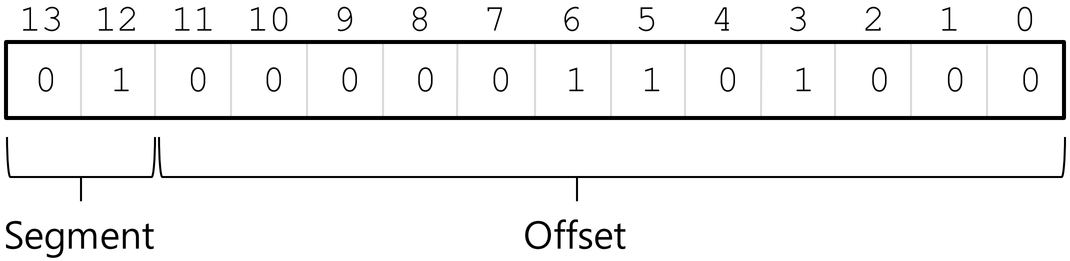
Referring to Segment

- **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



13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1	1	0	1	0	0	0

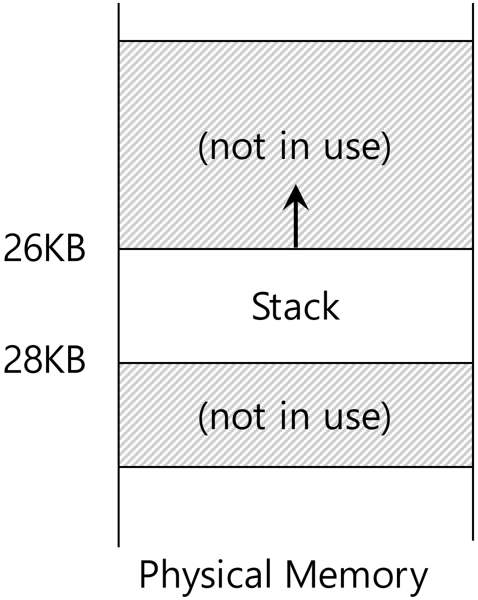
Referring to Segment(Cont.)

```
// get top 2 bits of 14-bit VA
Segment = (VirtualAddress & SEG_MASK) >> SEG_SHIFT
// now get offset
Offset = VirtualAddress & OFFSET_MASK
if (Offset >= Bounds[Segment])
    RaiseException(PROTECTION_FAULT)
else
    PhysAddr = Base[Segment] + Offset
    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

"Half of Operating Systems is Stupid Memory Management Tricks" – P. Bridges

- **Now: multiple processes, each with own address space**
- **Lots of optimization opportunities and subtle questions?**
  - How many copies of libc exist in the memory of the system at once?
  - What if we want to run more programs than we have physical memory?
  - Can physical memory be in multiple segments at the same time?

Support for Sharing

- **Segment can be shared between address space.**
  - **Code sharing** is still in use in systems today (shared libraries, etc.)
  - Needs 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

- **Who maintains these bits?**

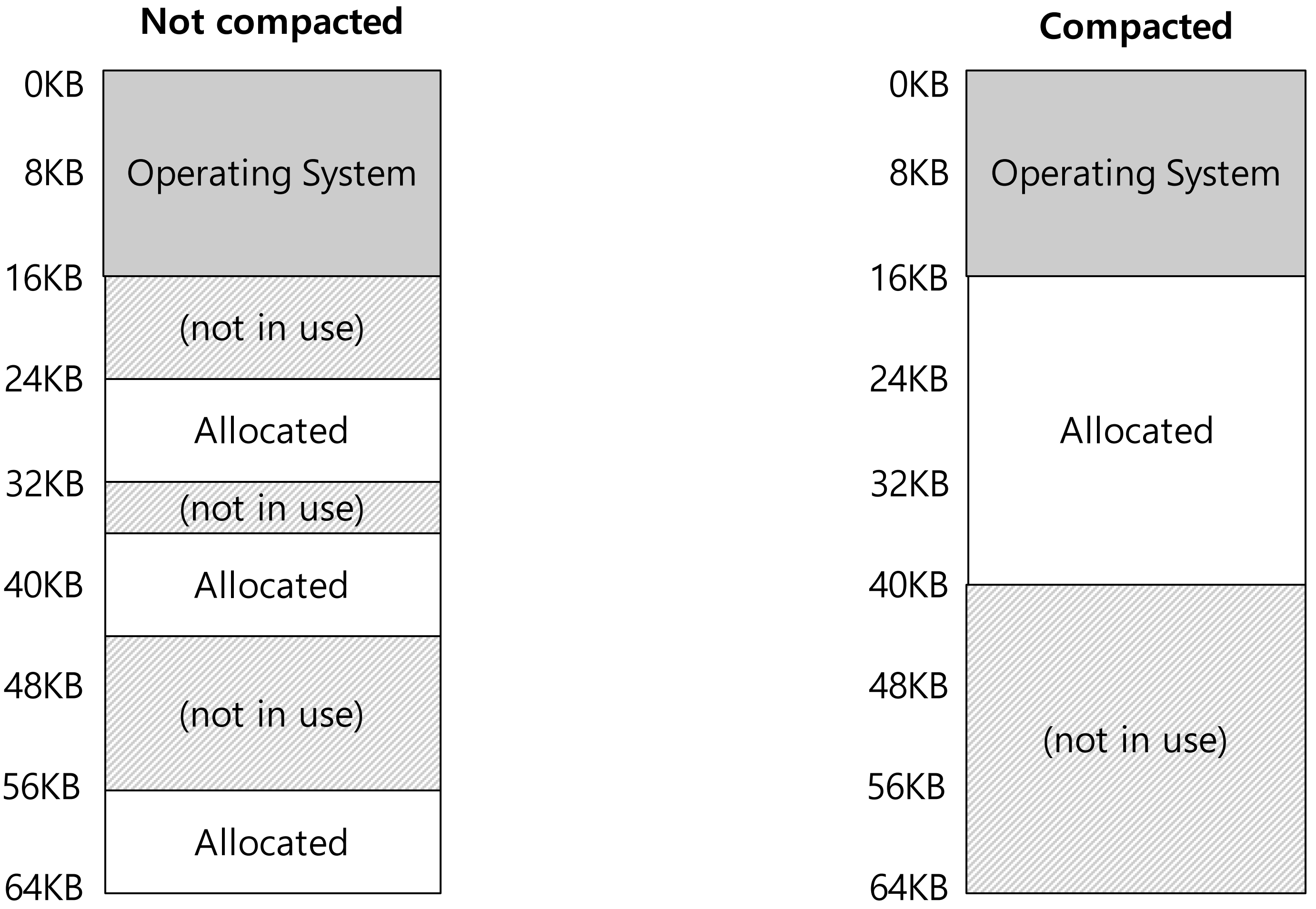
How many segments should we have?

- **Coarse-grained (few segments) means segmentation in a small number of segments.**
  - e.g., code, heap, stack.
  - Relatively easy to manage
- **Fine-grained (lots of segments) allows more flexibility for stupid OS tricks**
  - The OS can do lots of things with lots of segments (e.g. map multiple different shared libraries into multiple processes)
  - But the OS has to manage the allocation of all of these segments
  - Typically supported with a hardware **segment table**

Segmentation Problems: External Fragmentation

- **External Fragmentation: little holes of free space in physical memory that make difficulty to allocate new segments.**
  - 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.
- **The more segments you have, the worse it is.**

Memory Compaction



Whence Segmentation

- **Segmentation is variable length allocation**
  - Just like malloc free lists, with many of the same problems
  - It’s useful and flexible, but hard to mange well
  - Particularly when you have lots of segments (e.g. from either lots of segments per process or lots of processes)
- **Modern OSes make only very limited use of segmentation**
  - 32-bit mode x86 (introduced with 80286) can use segments extensively, but most OSes (e.g. Windows and Linux) don’t
  - 64-bit mode x86 forces most segments to have a base address of 0
  - With a very narrow exception usually used for thread-specific data