

A detailed close-up photograph of a computer motherboard. The image shows various components including blue plastic connectors, yellow RAM modules, and a large blue heat sink. The motherboard itself is black with intricate circuitry and gold-plated pins visible in the connectors. The lighting is bright, highlighting the textures and colors of the hardware.

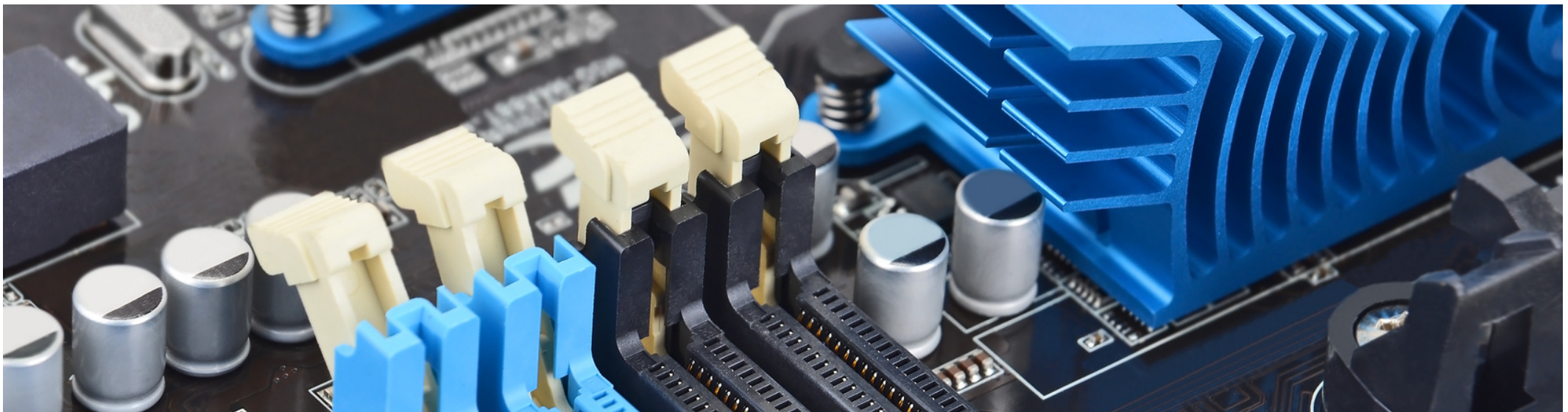
Lecture

Operating System

16. Segmentation

16. Segmentation

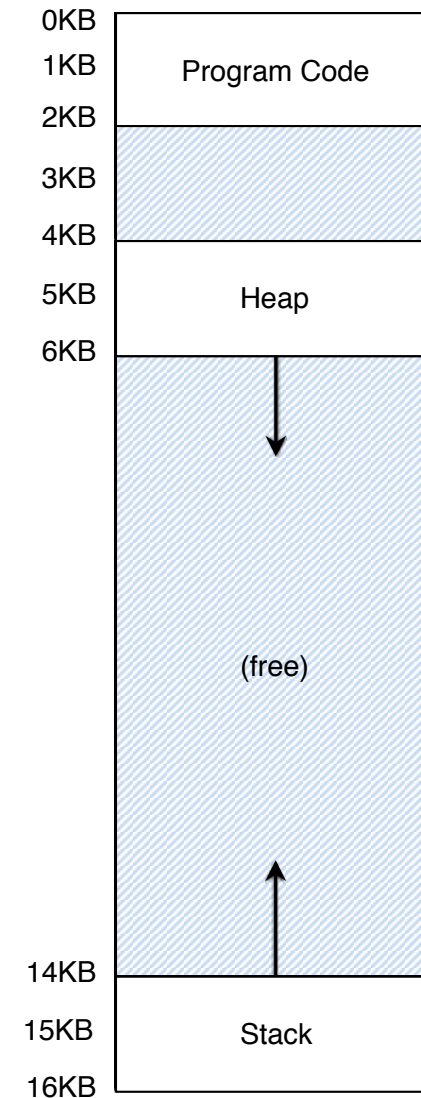
- 1. How do we support a large address space with (potentially) a lot of free space between the stack and the heap?**
- 2. How does Segmentation work?**
- 3. Advantages and Disadvantages of Segmentation**



Base and Bound Approach

Inefficiency

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

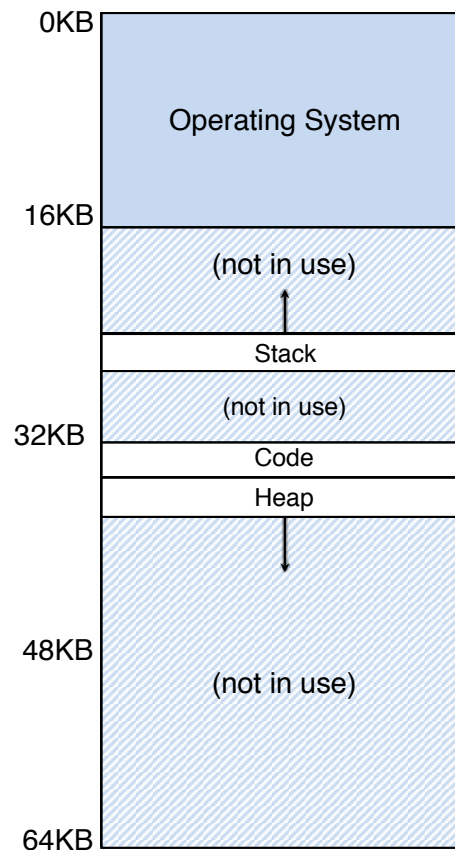


Segmentation

- **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**.

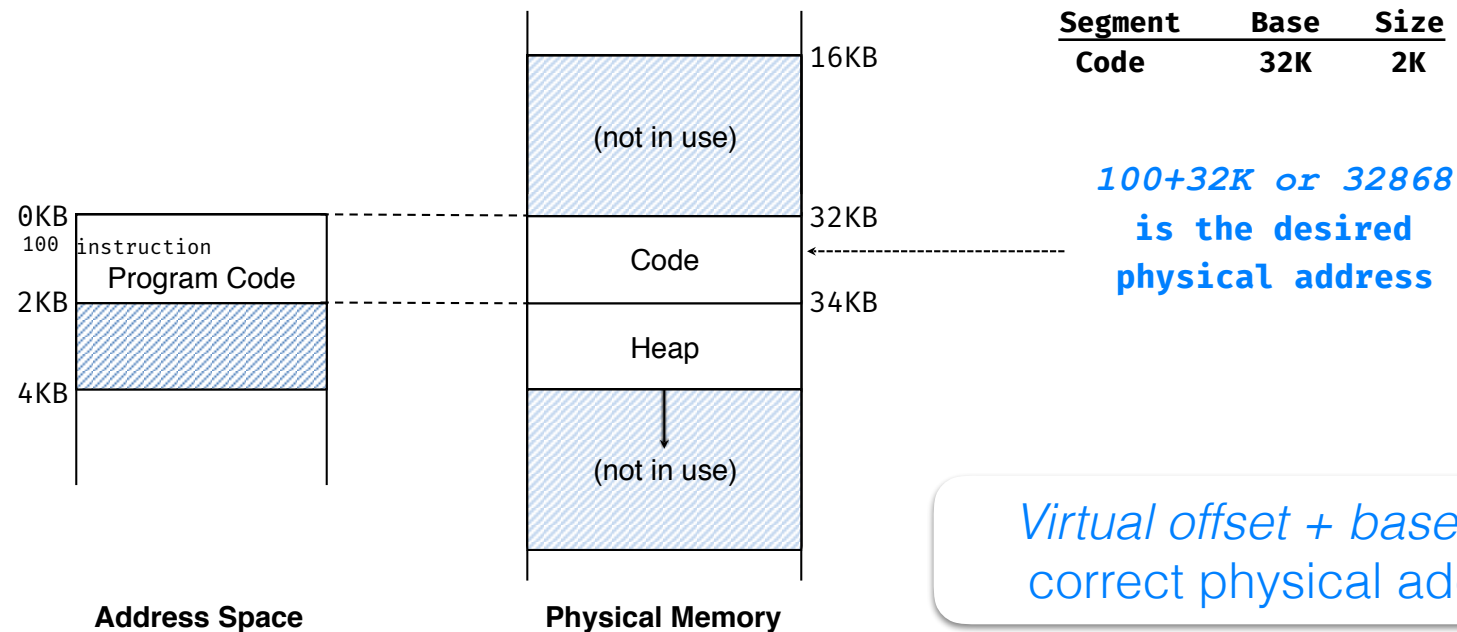
Placing Segment in Phys. Memory

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



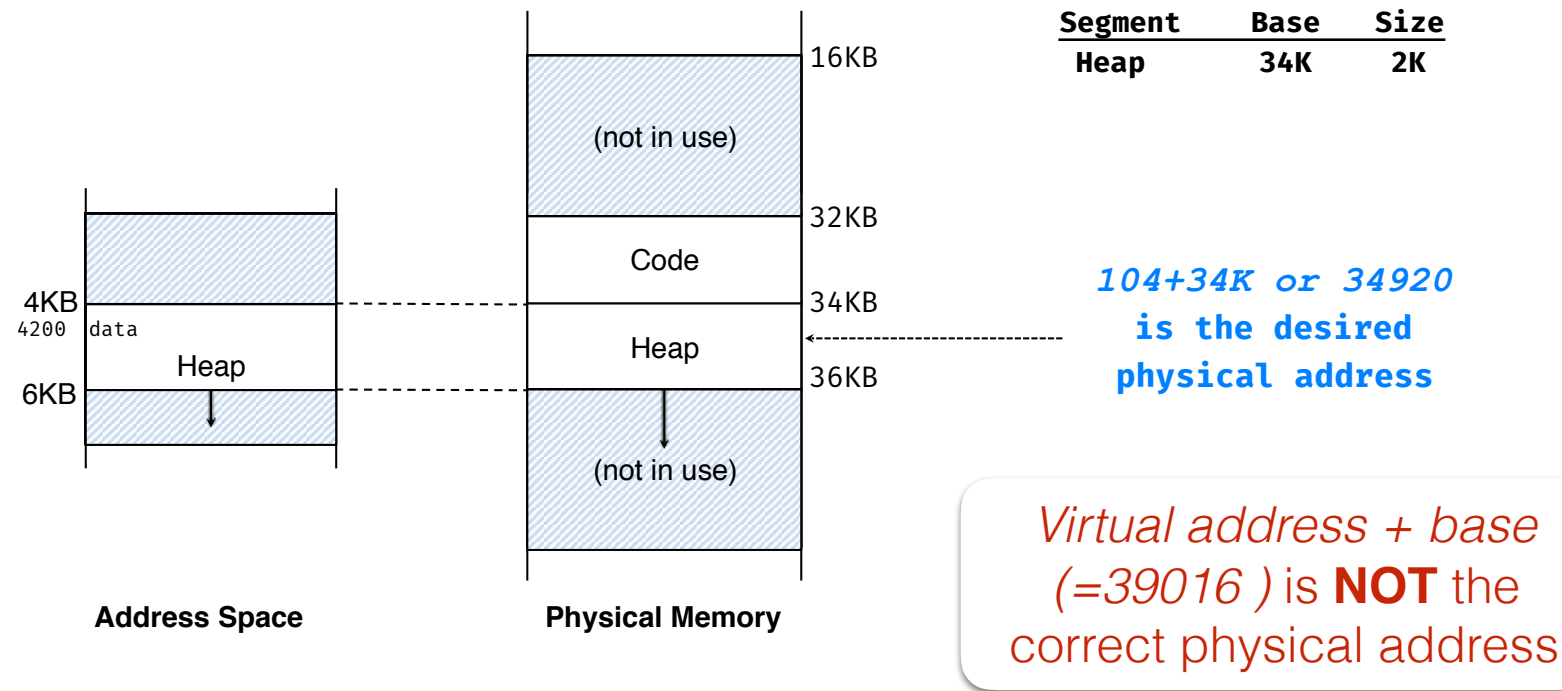
Physical Memory

Example 1: Address Translation



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

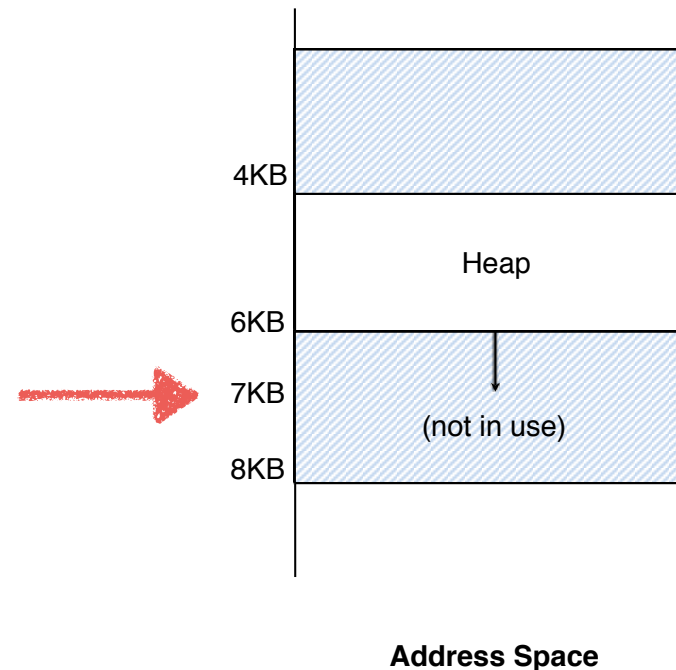
Example 2: Address Translation



- 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**.



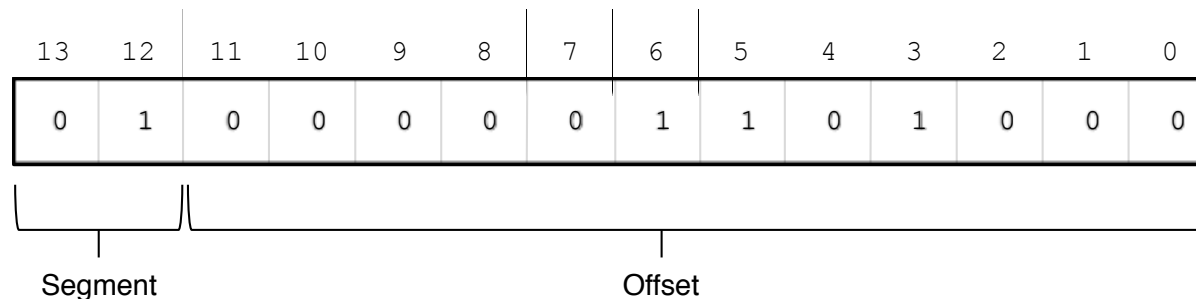
Referring to Segment

- Explicit approach

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

- Example:

- Virtual address 4200 (010000001101000)



Segment	bits
Code	00
Heap	01
Stack	10
-	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)
```

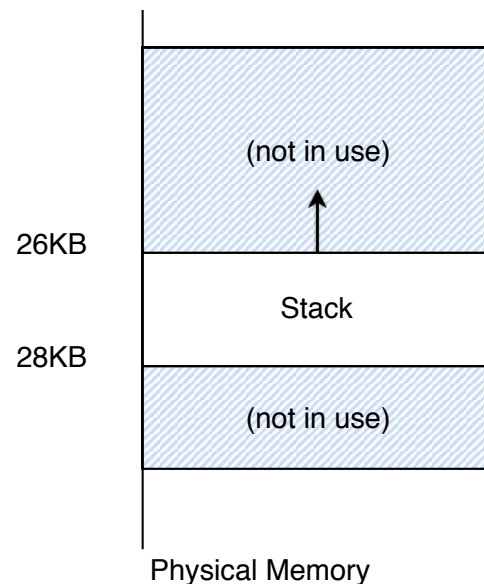
```
SEG_MASK = 0x3000(11000000000000)
```

```
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	

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**.

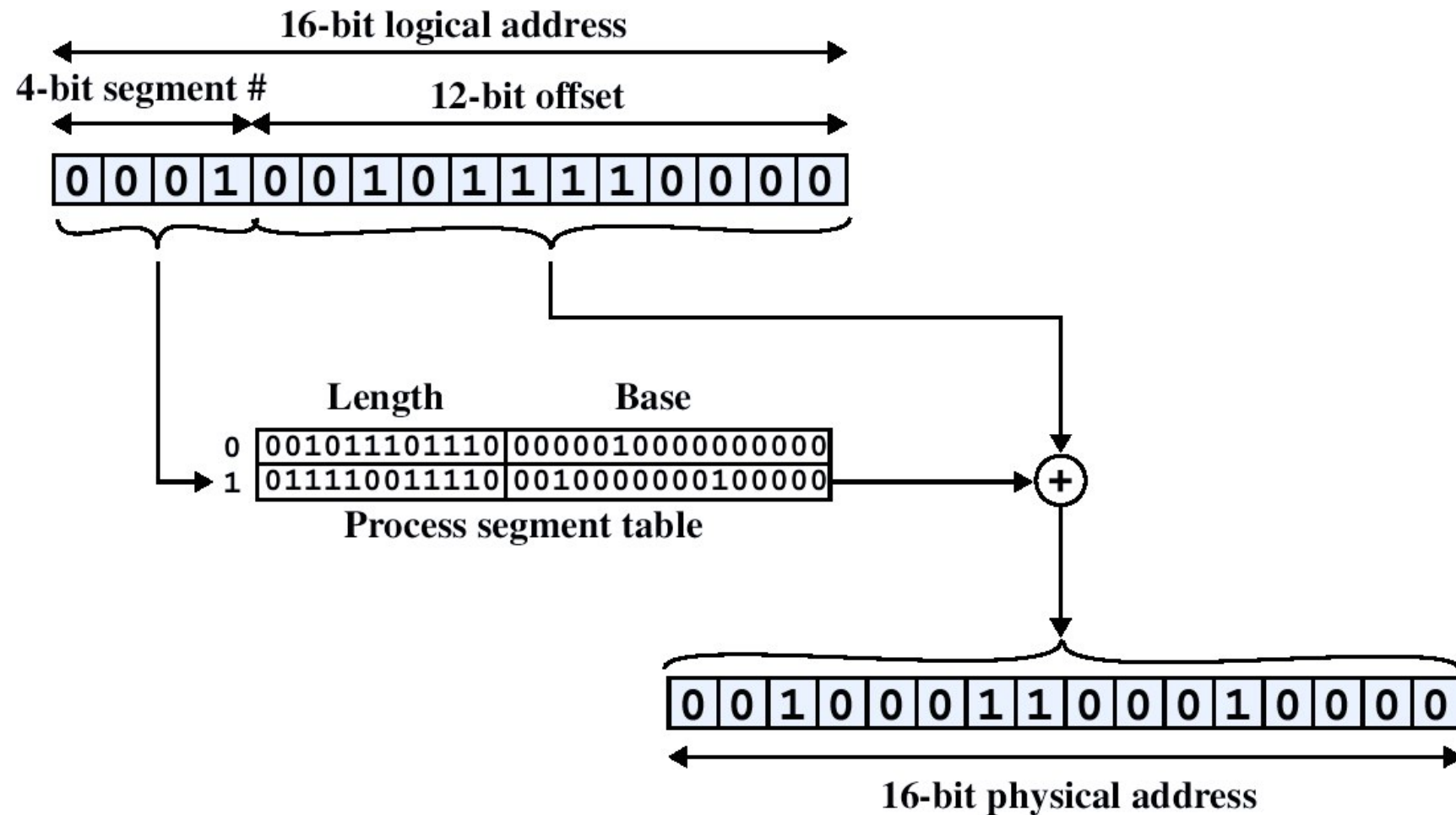
Segment Register Values(with Protection)

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

- **Coarse-Grained** means segmentation in a small number.
 - 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.

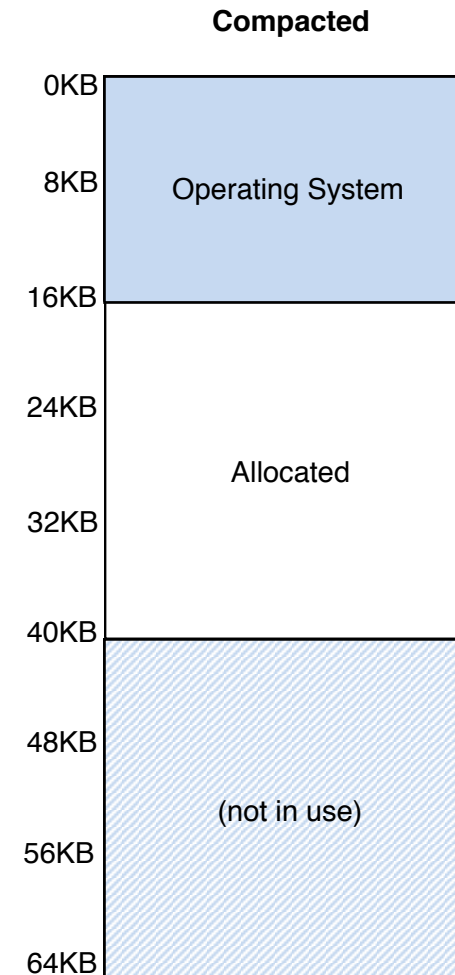
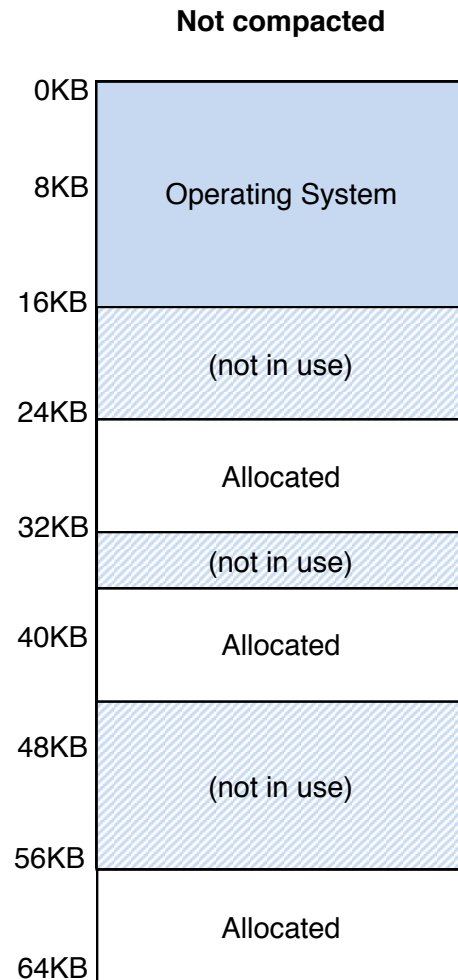
Segment Table for each Process



OS support: Fragmentation

- **External Fragmentation**: little **holes** of **free space** in physical memory that make difficulty to allocate new segments.
 - Example:
 - 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



Advantages of Segmentation

- Enables sparse allocation of address space
 - **Stack** and **heap** can grow **independently**
 - **Heap**: If no data on free list, dynamic memory allocator requests more from OS (e.g., UNIX: malloc calls `sbrk()`)
 - **Stack**: OS recognizes reference outside legal segment, extends stack *implicitly*
- Different **protection** for different segments
 - Read-only status for code
- Enables **sharing** of selected segments
- Supports **dynamic relocation** of each segment

Disadvantages of Segmentation

- Each segment must be allocated **contiguously**
 - May not have sufficient physical memory for large segments
- Solution:
 - Paging (will be discussed later)

A close-up photograph of a computer motherboard. The image shows various components including blue plastic connectors, yellow plastic connectors, and a large blue heat sink. The motherboard is black with visible circuitry and components like capacitors.

Thanks

Questions?