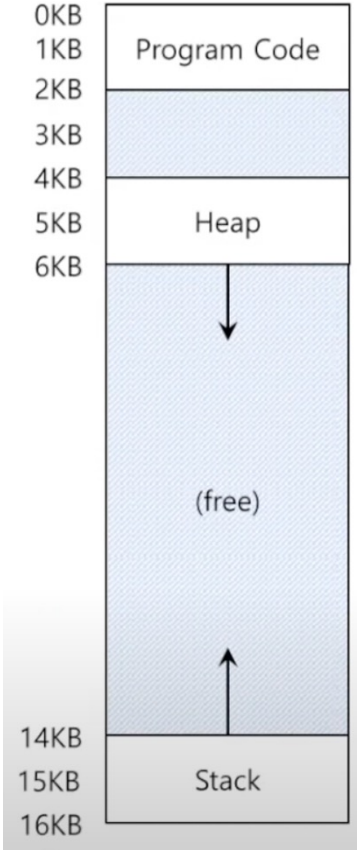
抽象-分段

邰颖
南京大学
智能科学与技术学院

基址和界限寄存器方法的低效性



- 大块的“空闲”空间
  - “空闲”空间占用了物理内存。
  - 当地址空间无法完全装入物理内存时，运行程序会变得困难。

分段（Segmentation）：泛化的基址与界限方法

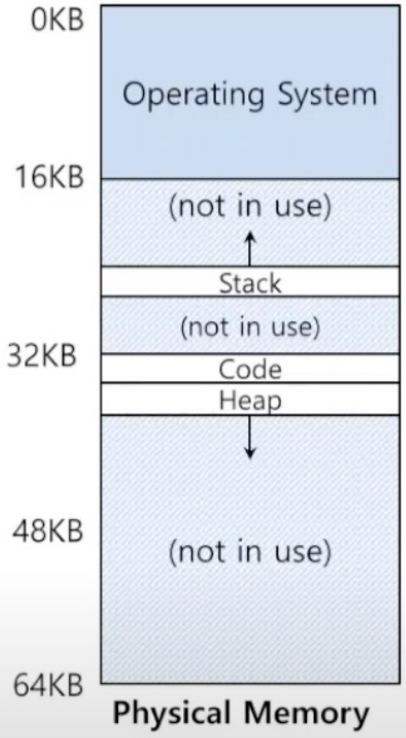
- **动机：**为什么不为一个进程地址空间中的每个逻辑段使用一对基址寄存器和界限寄存器？

- **一个段（Segment）：**是地址空间中具有特定长度的一个**连续区域**。
  - **典型的地址空间中逻辑上不同的段：**代码段、栈段、堆段。

- **每个段可以放置在物理内存的不同部分。**
  - **基址寄存器和界限寄存器是按段存在的（每个段一对）。**

分段在物理内存中的映射

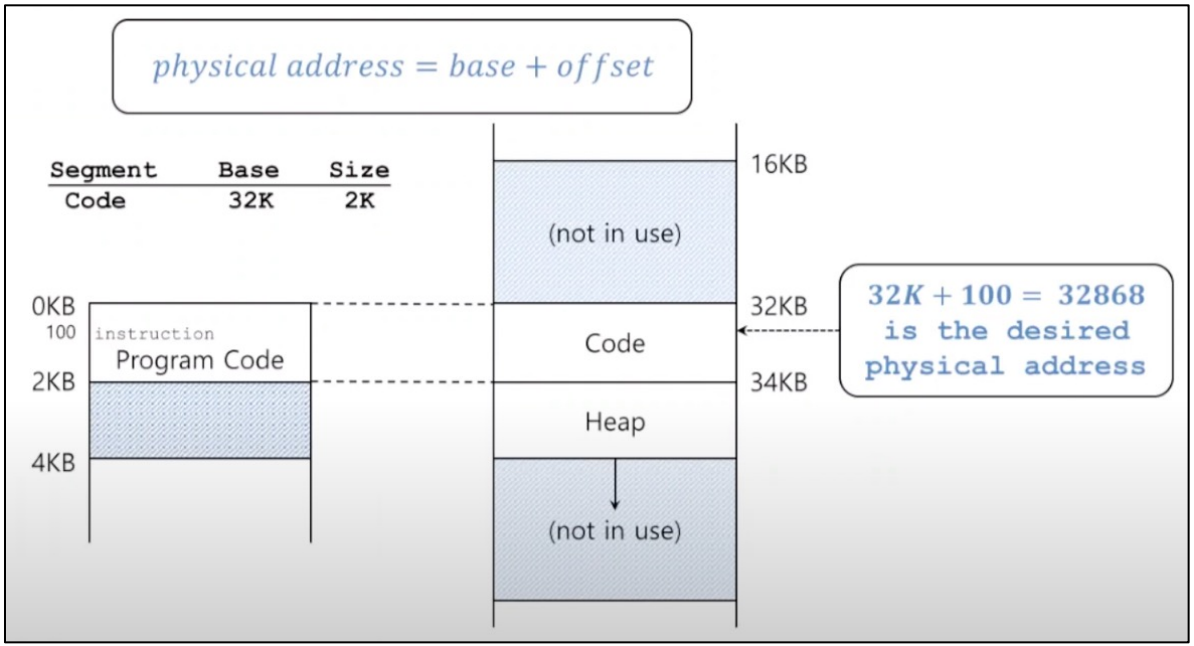
•如何在物理内存中放置不同的段。注意它们不需要是**连续**的。

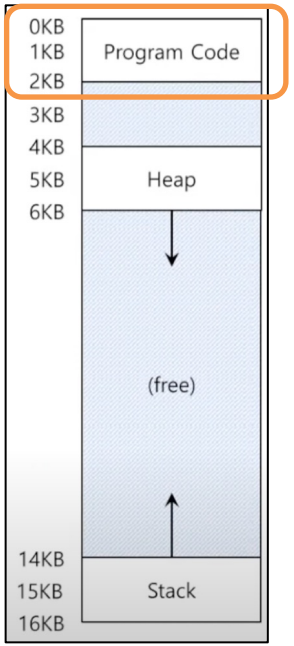


Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

分段：地址转换示例

- 假设引用虚拟地址 100（位于代码段中）
  - 由于代码段在虚拟地址空间中从 0 开始，因此偏移量为 100。

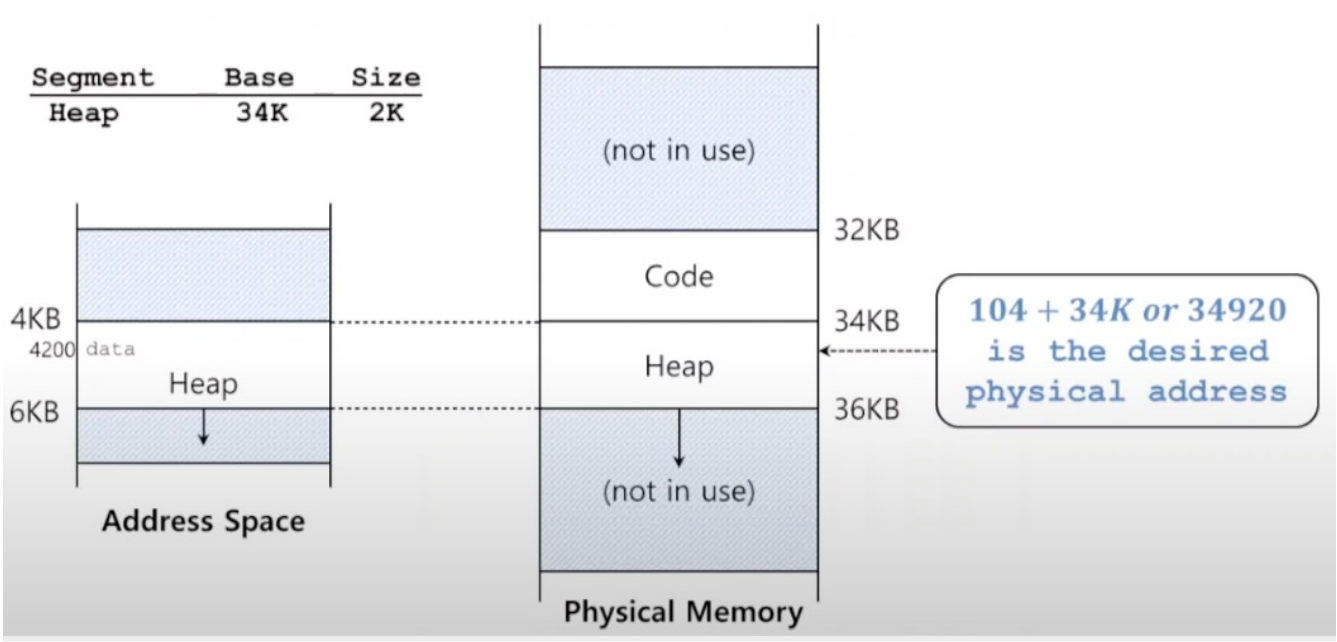


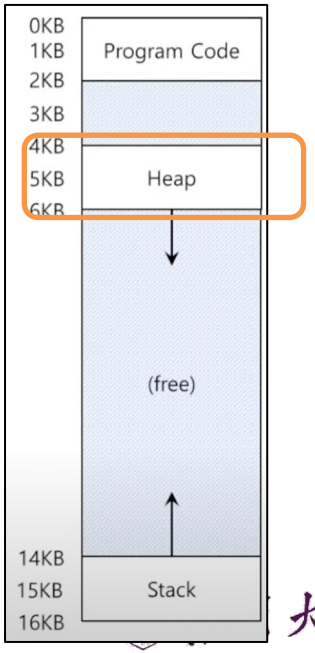


地址转换示例（续）

虚拟地址位置 + 对应段的基址 ≠ 正确的物理地址

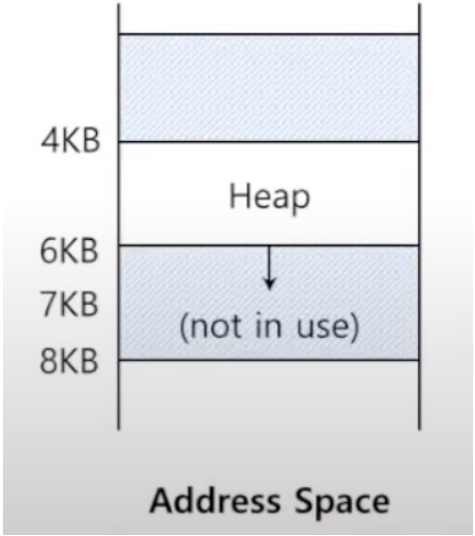
- 假设引用虚拟地址 **4200**（位于堆区）
  - 堆区在虚拟地址空间中的起始地址为 **4096**（**4KB**），因此偏移量：4200−4096=104





段错误或段违规

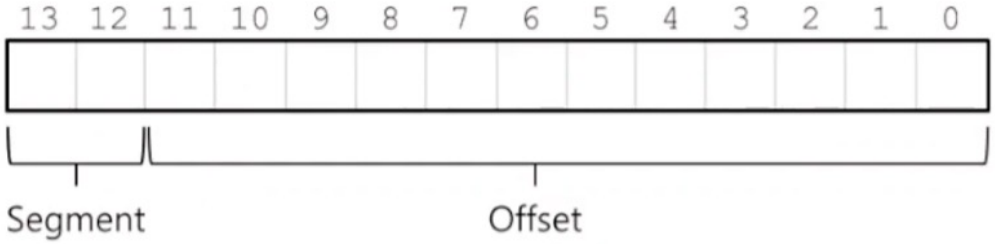
- 如果访问了非法地址（如 **7KB**），超出了堆的末尾，操作系统会生成段错误（**segmentation fault**）
  - 硬件检测到该地址超出范围（**out of bounds**）。



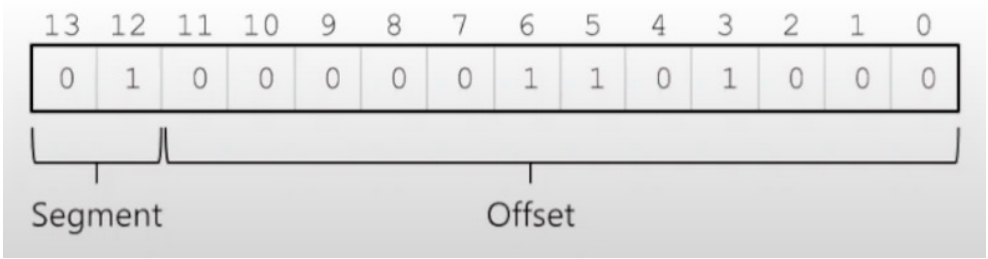
引用哪个段

- 如果只给定一个虚拟地址，硬件如何确定段和偏移量？

- #1 显式方法（Explicit approach）
  - 根据虚拟地址的高几位比特（top few bits）划分地址空间为不同的段



  - 示例：虚拟地址 4200（01000001101000）



引用哪个段

```
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)
```

段MASK
- SEG_MASK = 0x3000(11000000000000)

确定段选择
- SEG_SHIFT = 12

偏移量MASK
- OFFSET_MASK = 0xFFF (00111111111111)

- **显式方法（使用最高 2 位）的缺点**
  - 如果只有 3 个段，则使用 2 位来存储段信息是**浪费**的。
  - 限制段的大小，例如**最大段大小为 4KB（12位）**。

- **#2 隐式方法**
  - 根据**虚拟地址的生成方式**确定段，例如：如果地址来自**程序计数器（PC）**，则属于**代码段**

引用哪个段：示例

```
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)
```

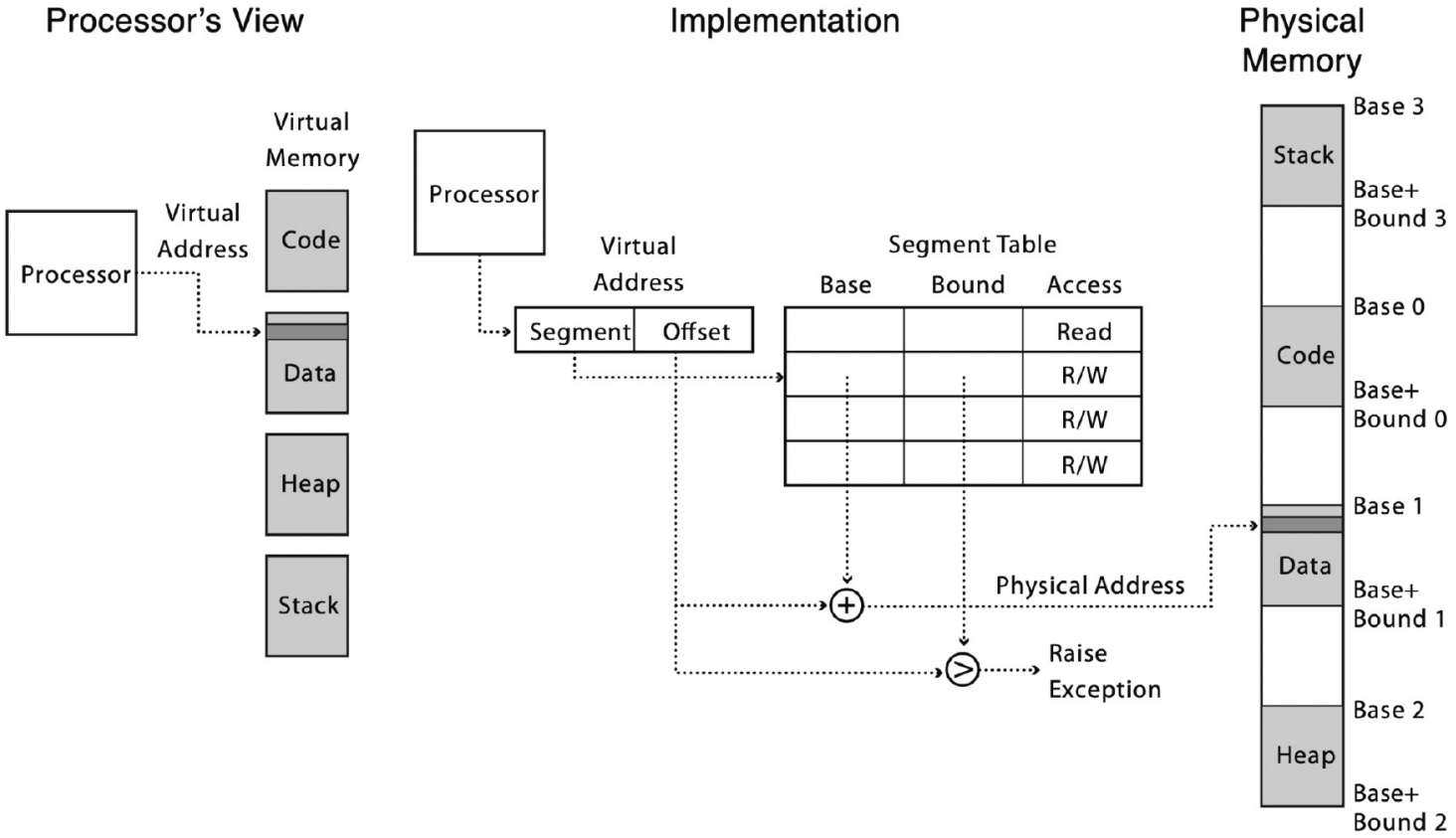
- SEG_MASK = 0x3000(11000000000000)
- SEG_SHIFT = 12
- OFFSET_MASK = 0xFFF (00111111111111)

- 假设虚拟地址：4200 = 0001 0000 0110 1000（二进制）
- 执行按位与操作：0x1000 = 4096(十进制)

```
VirtualAddress: 0001 0000 0110 1000
SEG_MASK:       0011 0000 0000 0000
结果:           0001 0000 0000 0000
```

- 右移12位（SEG_SHIFT）得到段选择符: 0x1000 >> 12 = 01
- 偏移量： 0000 0110 1000 = 104

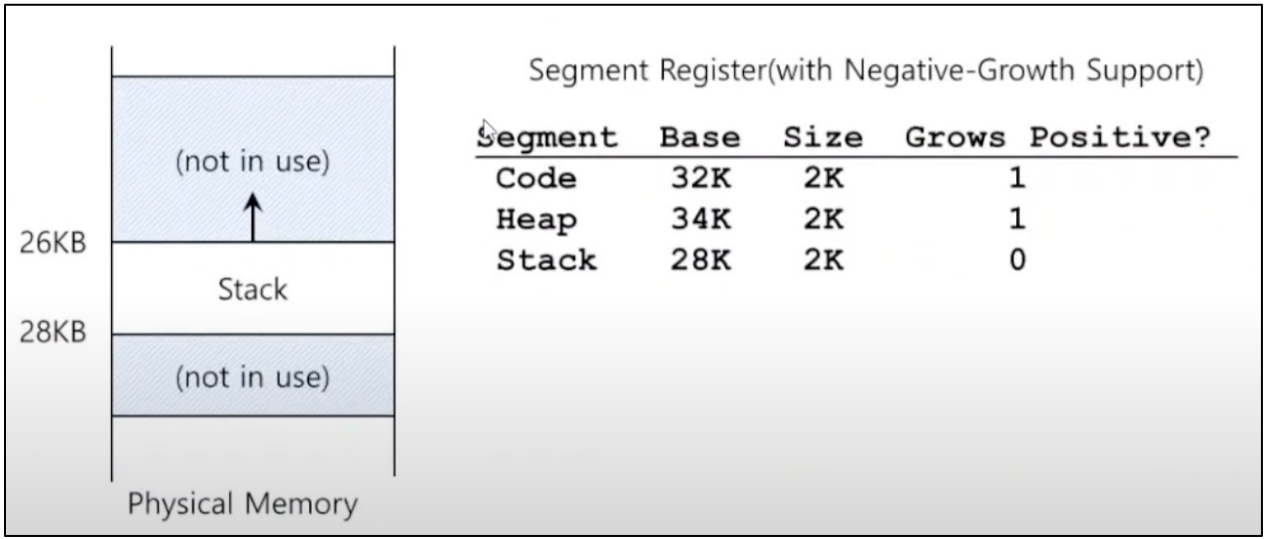
虚拟内存如何通过分段映射到物理内存



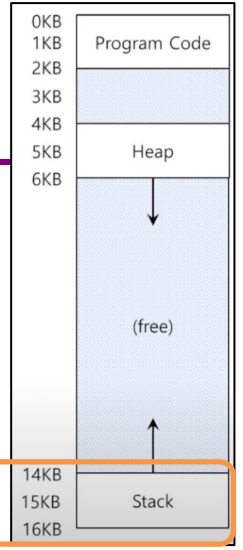
关于栈段（Stack Segment）

- 栈是向低地址（反向）增长的！而代码段、堆段是向高地址（正向）增长

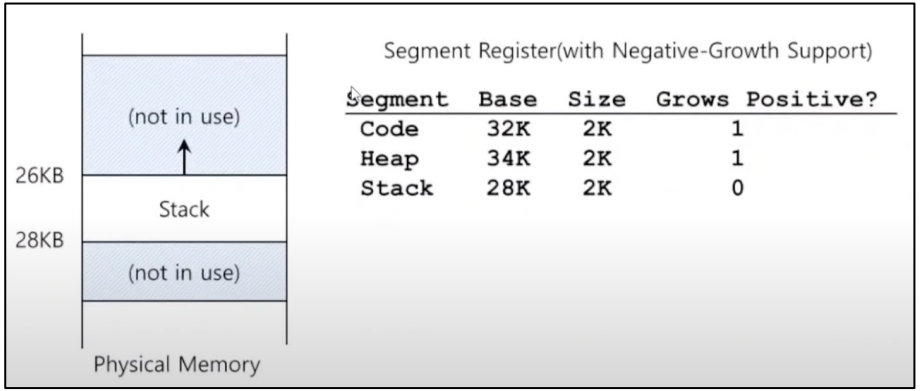
- 需要额外的硬件支持：
  - 硬件需要检查段的增长方向。
  - 1：正向增长，0：负向增长。



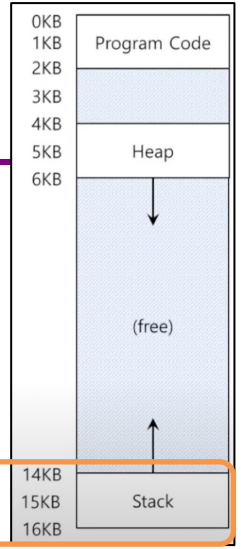
关于栈段（Stack Segment）



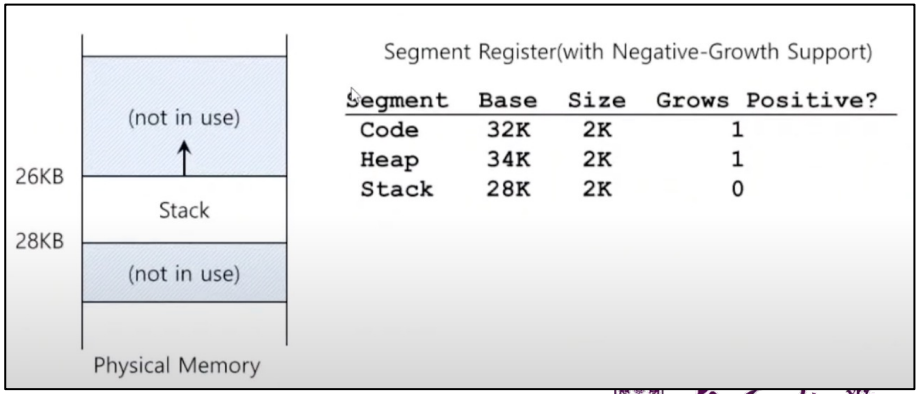
- **示例：**引用虚拟地址 15KB = 11 1100 0000 0000 = 0x3C00
- **段选择：**Segment=0x3(3)，Offset=0xC00 (3KB)
- **计算负偏移量：**
  - 负偏移量 = 偏移量 - 最大段大小 = 3KB - 4KB = -1KB
- **计算物理地址：**
  - 物理地址 = 负偏移量 + 栈段基址
  - 物理地址 = -1KB + 28KB = 27KB



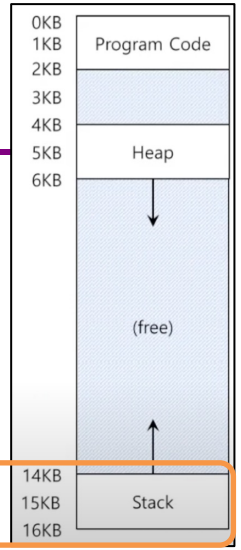
关于栈段（Stack Segment）：更多示例



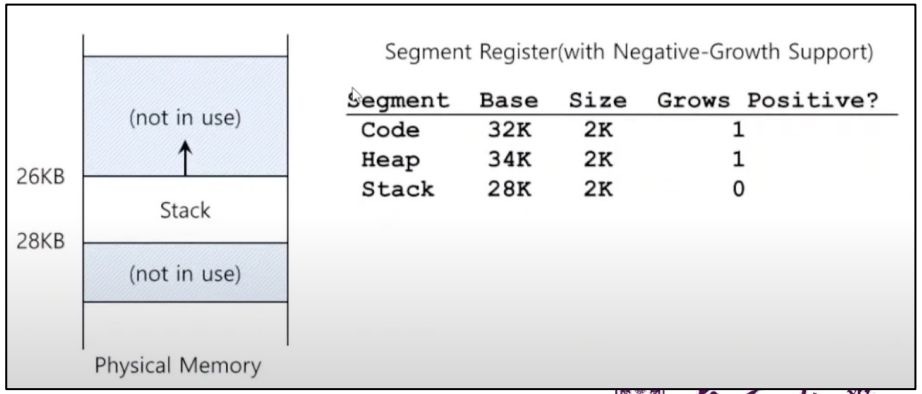
- **示例：** 引用虚拟地址 14KB = 11 1000 0000 0000
- **段选择：** Segment= 0x3(3)， Offset=2KB
- **计算负偏移量：**
  - 负偏移量 = 偏移量 - 最大段大小 = 2KB - 4KB = -2KB
- **计算物理地址：**
  - 物理地址 = 负偏移量 + 栈段基址
  - 物理地址 = -2KB + 28KB = 26KB



关于栈段（Stack Segment）：更多示例



- **示例：** 引用虚拟地址 16380= 11 1111 1111 1100
- **段选择：** Segment= 0x3(3)， Offset= 0xFFC(4092)
- **计算负偏移量：**
  - 负偏移量 = 偏移量 - 最大段大小 = 4092 - 4KB（4096） = -4
- **计算物理地址：**
  - 物理地址 = 负偏移量 + 栈段基址
  - 物理地址 = -4 + 28KB = 28668



共享支持

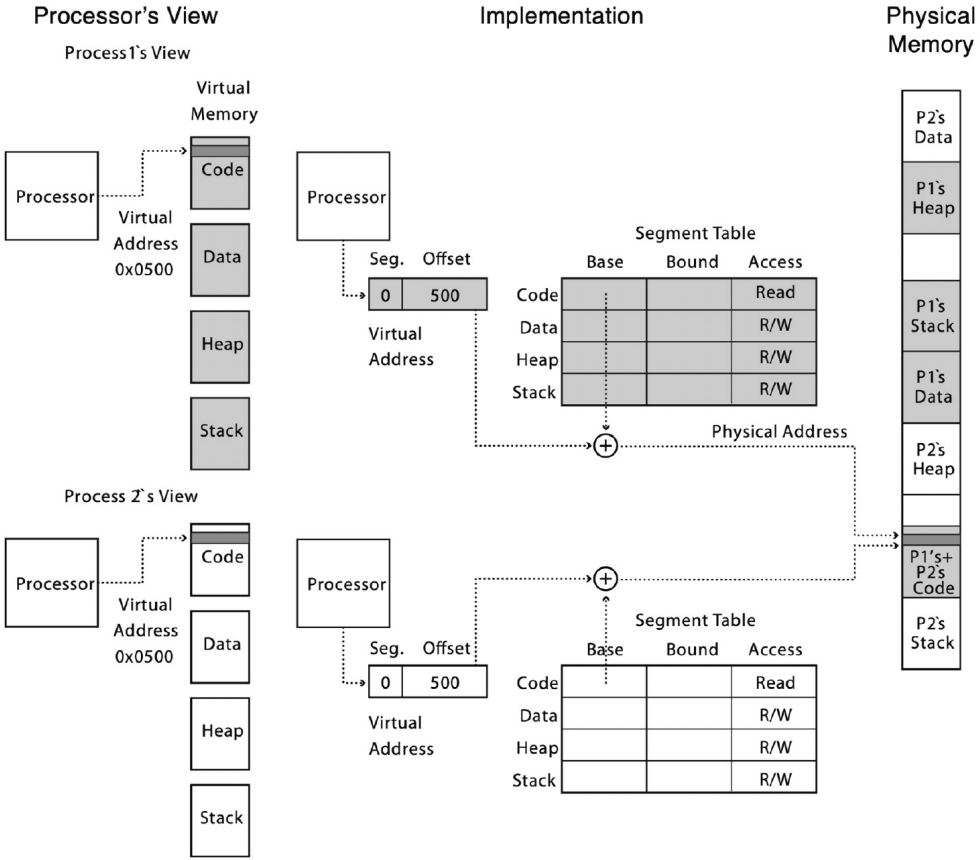
- **段可以在地址空间之间共享**
  - 代码共享在当前系统中仍然被使用

- **需要额外的硬件支持：保护位**
  - 每个段增加一些位来指示**读取、写入和执行**的权限

段寄存器值（带保护）

段	基址	大小	正向增长?	保护
代码	32K	2K	1	读-执行
堆	34K	2K	1	读-写
栈	28K	2K	0	读-写

共享支持示例：代码段



细粒度与粗粒度分段

•粗粒度（Coarse-Grained）表示分段数量较少

- 例如：代码段、堆段、栈段

•细粒度（Fine-Grained）分段提供更大的灵活性，适用于某些早期系统的地址空间管理

- 为了支持更多的段，硬件需要提供**段表（Segment Table）**的支持

操作系统对分段的支持

- 在上下文切换（context switch）时，操作系统应该做什么？
- 当一个段增长（segment grows）时，操作系统应该做什么？
- 如何管理内存中的空闲空间？
  - 段大小是可变的（Variable segment sizes）

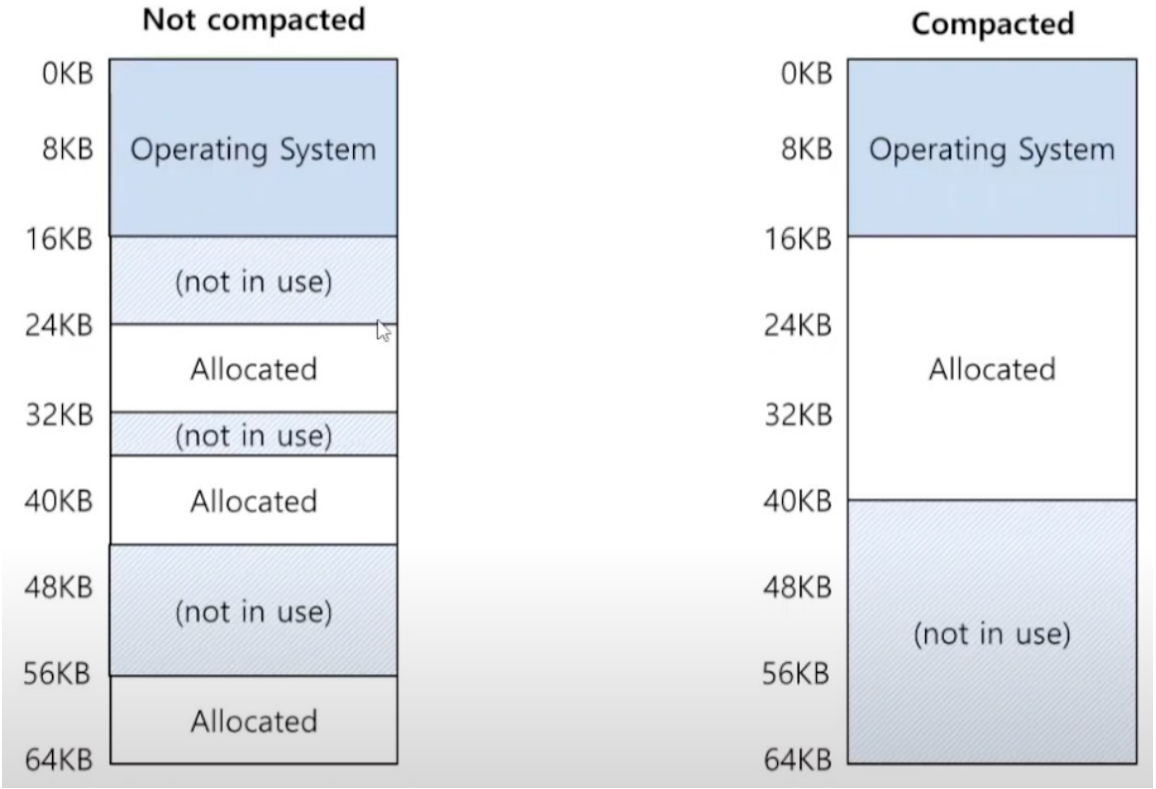
操作系统对分段的支持：碎片管理

- **外部碎片（External Fragmentation）**：物理内存中存在许多零散的空闲空间，导致难以分配新的段
  - **有24KB空闲空间**，但**不在同一连续段**中
  - 操作系统**无法满足**一个**20KB**的请求
- **紧凑化（Compaction）**：重新排列物理内存中的现有段
  - 紧凑化是**代价高昂**的操作
    - **停止**正在运行的进程
    - **复制**数据到其他地方
    - **更改**段寄存器值

操作系统对分段的支持：碎片管理(续)

- 更好的做法是使用空闲列表管理算法（**Free-List Management Algorithm**）
  - **保持**大块连续内存空间可用于分配
  - **常见算法：**
    - 最佳适配（Best-Fit）：从空闲链表中找最接近需要分配空间的空闲块
    - 最坏适配（Worst-Fit）：寻找**最大**的空闲块，防止过度分割
    - 伙伴算法（Buddy-Algorithm）：通过**成对分配**和合并管理空闲空间

内存压缩（Memory Compaction）



小结

- 介绍了什么是分段，相比基址和界限方法的区别，如何实现更高效的虚拟内存
- 介绍了如何确定段和偏移量的方法
- 介绍了不同粒度的分段
- 介绍了段的共享、操作系统对分段的支持