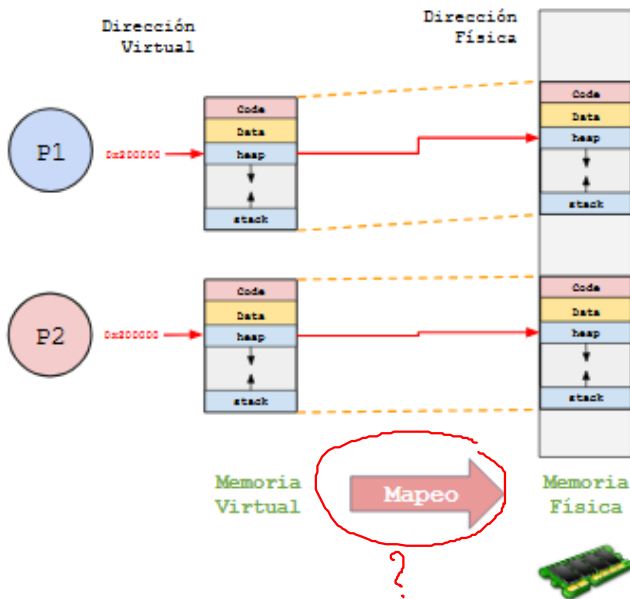
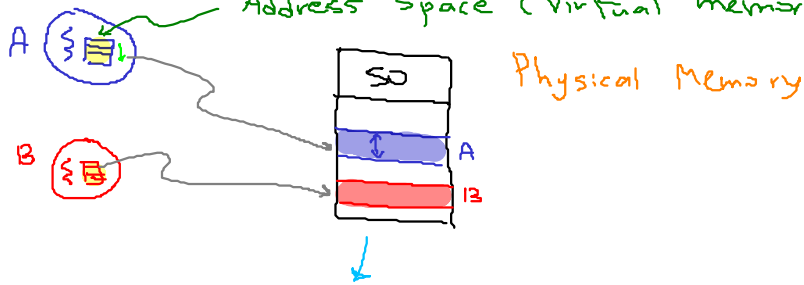


# 11/09/2025 - Sistemas Operativos (Vde@)

## 1. Repaso clase anterior

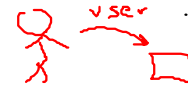
### a. Memoria Virtual vs. Memoria Física



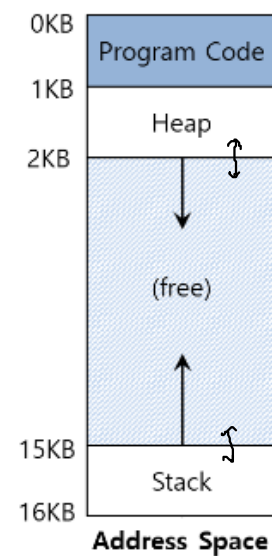
### b. Mapa de Memoria

- Program code: Programa

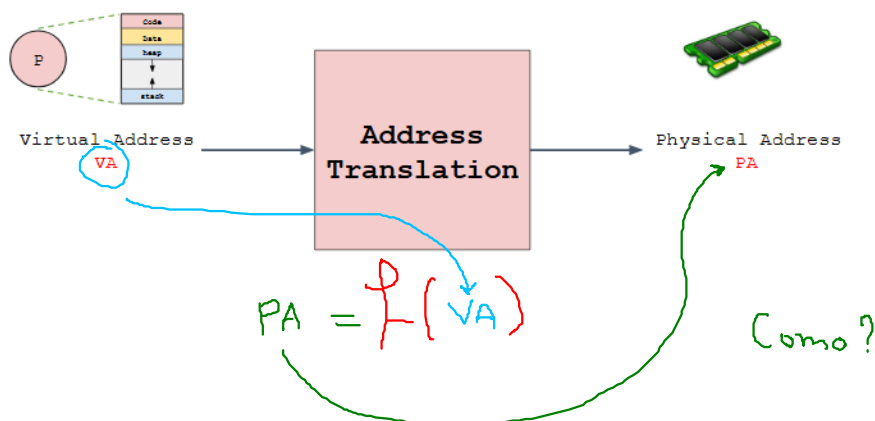
\* - Heap: Memoria dinámica (malloc/free)



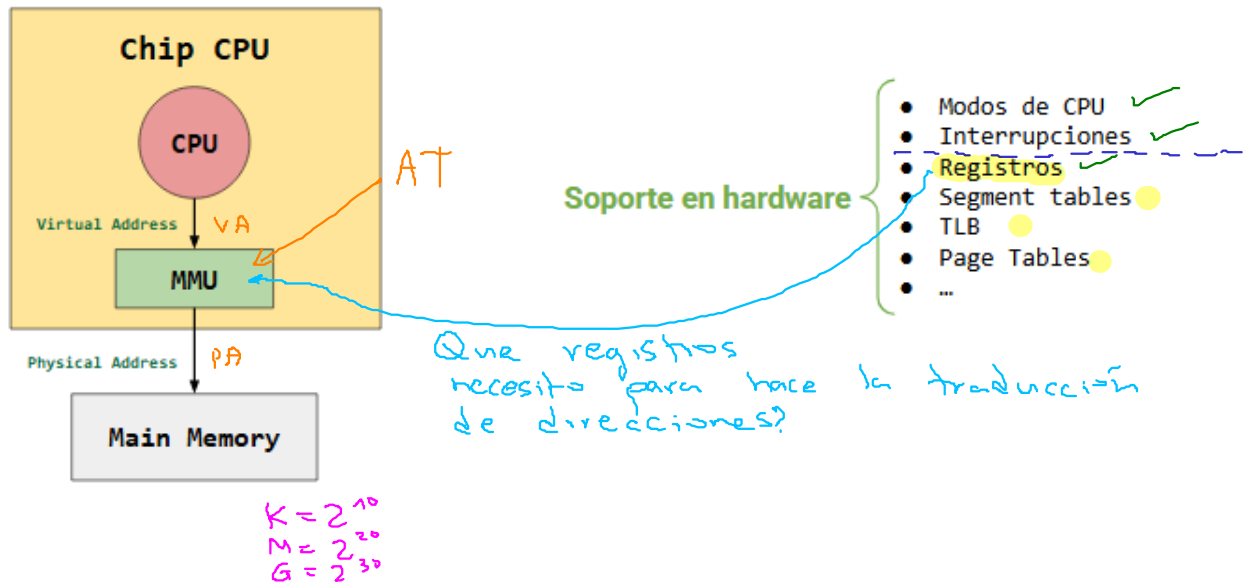
- Stack: Memoria automática.



### c. Address translation (AT)

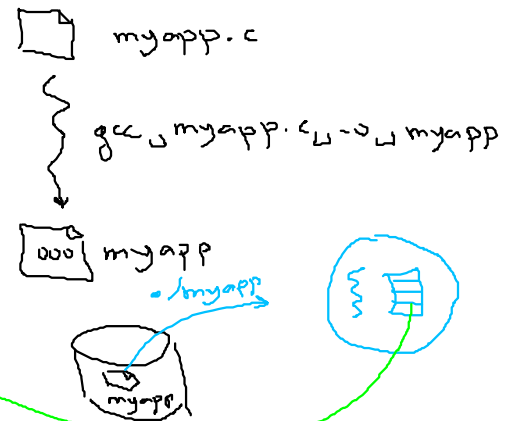


## 2. MMU (Management Memory Unit)



### Ejemplo

Para el siguiente fragmento de código asuma que la variable  $x$  está en la dirección 15360 (15K) y que las direcciones de las instrucciones generadas en lenguaje ensamblador (similar al MIPS) están en las direcciones mostradas a continuación.



#### Lenguaje C

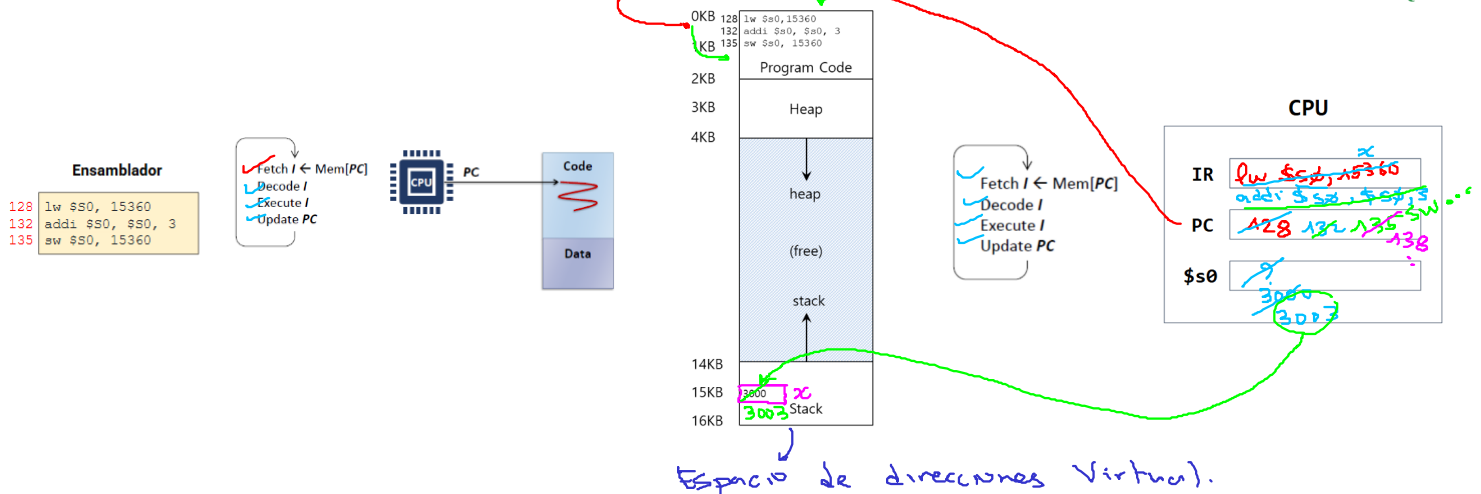
```
void func() {
    int x = 3000;
    x = x + 3;
}
```

*local (stack)*

#### Ensamblador

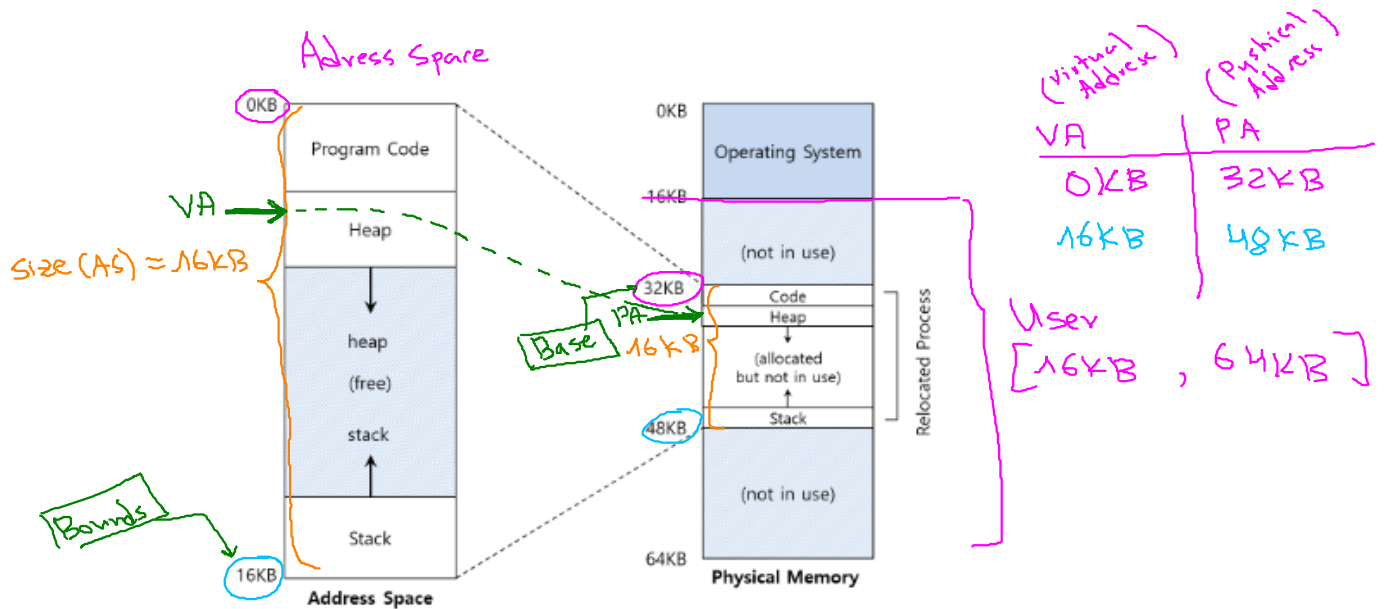
```
128 lw $s0, 15360
132 addi $s0, $s0, 3
135 sw $s0, 15360
```

¿Cómo se ejecutaría el código asociado a la instrucción  $x = x + 3$ ?

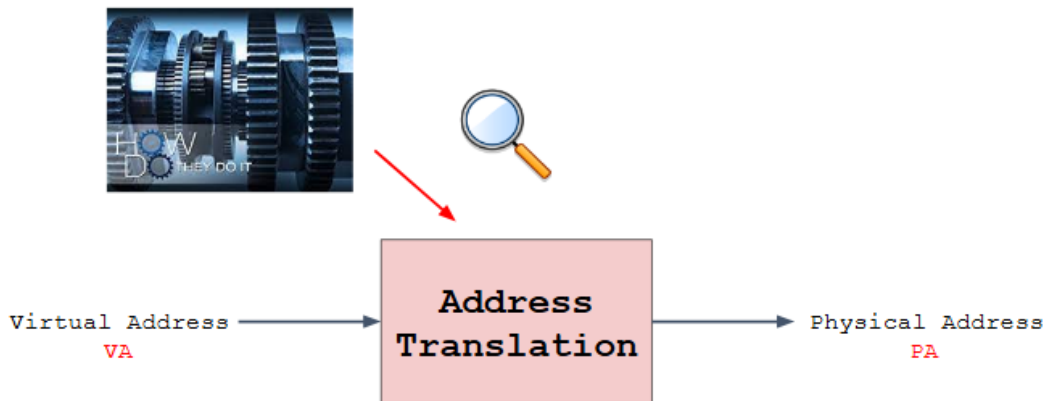


Pregunta importante? Donde se hubica el código en memoria Física?

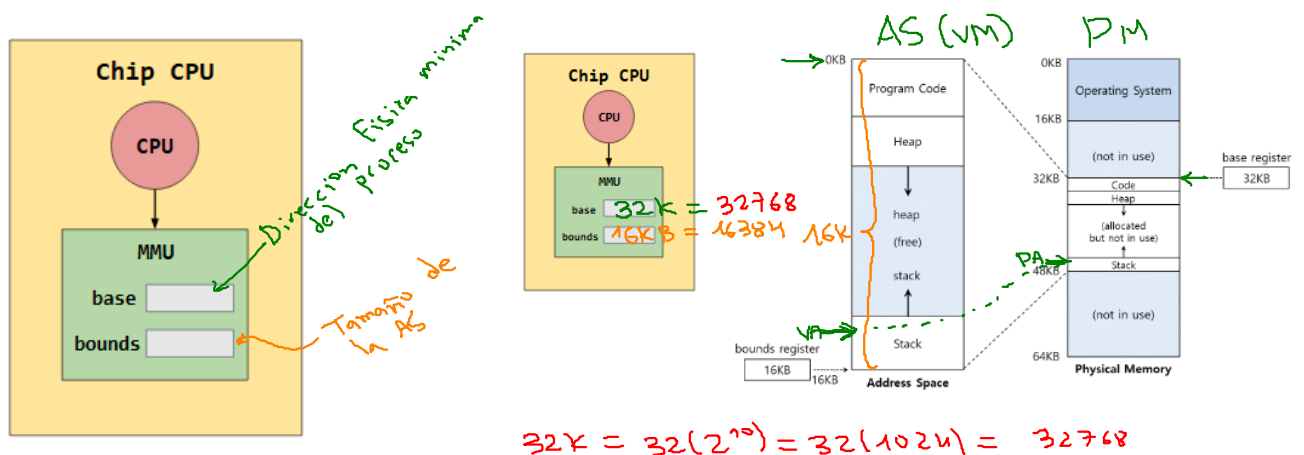
*Real*



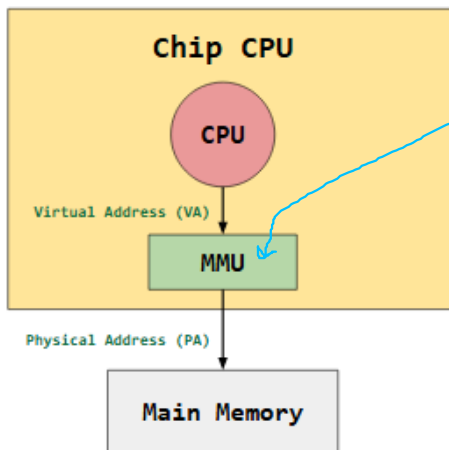
### ¿Cómo virtualizar memoria de manera eficiente y flexible?



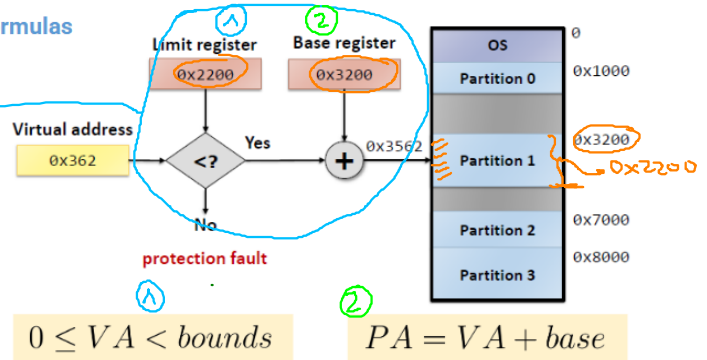
### 3. Reasignación dinámica de Memoria (Base & Bound)



## 4. Traducción de direcciones usando Dynamic Reloc (Base & Bound)

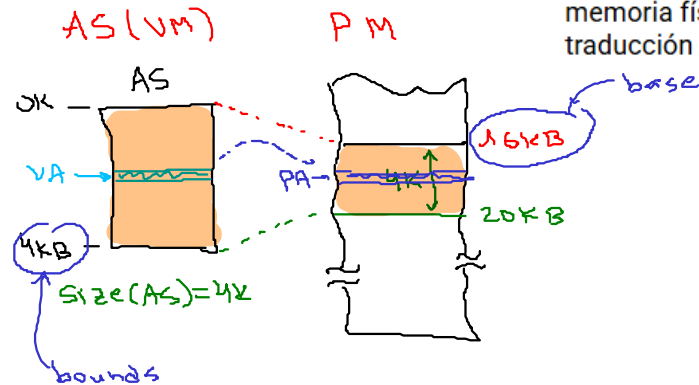


### Fórmulas



### Ejemplo:

Un proceso con un espacio de direccionamiento de 4KB fue asignado a una memoria física a partir de 16KB. Teniendo en cuenta lo anterior, realizar la traducción de direcciones virtuales a físicas mostradas en la siguiente tabla:



Virtual Address (VA)	Physical Address (PA)
0	?
1 KB	?
3300	?
4400	?

- bounds = 4K
- base = 16K = offset

①  $0 \leq VA < 4K$

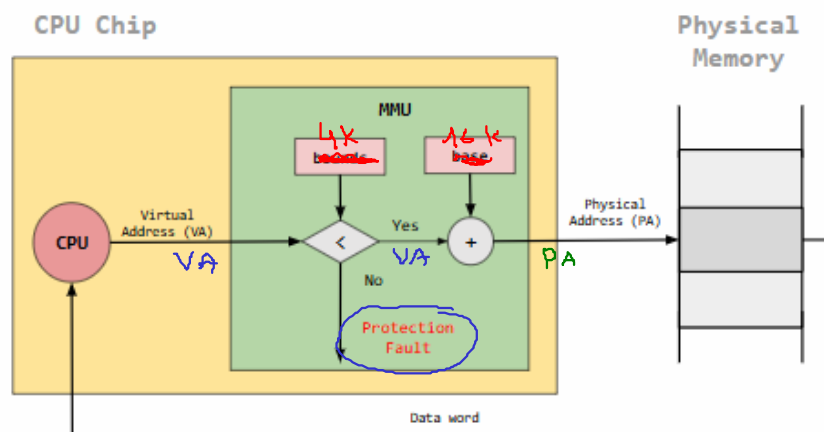
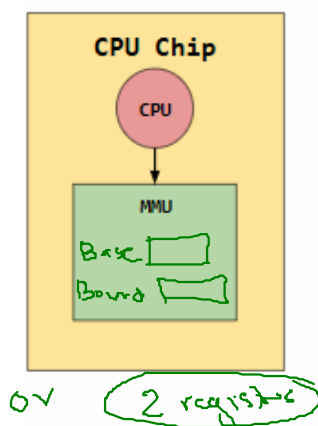
②  $PA = VA + 16K$

### ① Límites

$$0 \leq VA < \text{bounds} \quad \checkmark$$

### ② Conversión

$$PA = VA + \text{offset} \quad \text{base}$$

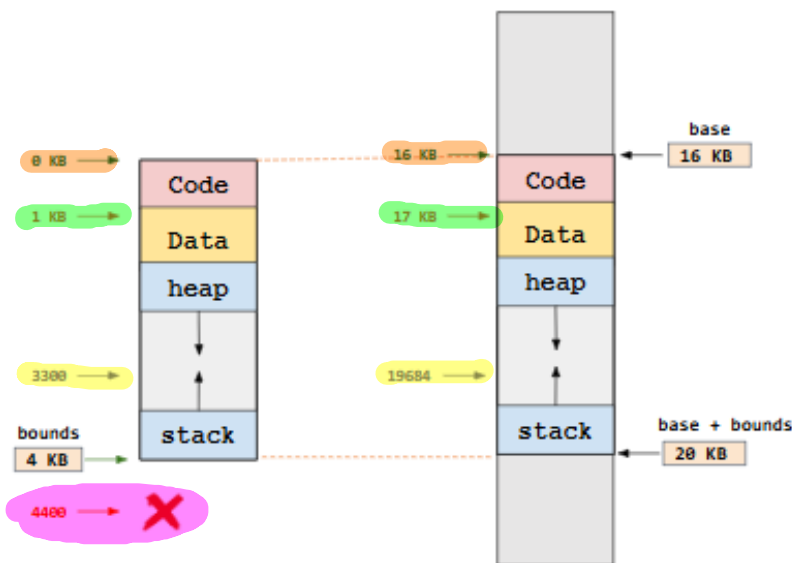


VA	① $0 \leq VA < 4K$	② $PA = VA + 16K$
0	$0 \leq 0 < 4K$ (✓)	$0 + 16K = 16K$
1K	$0 \leq 1K < 4K$ (✓)	$1K + 16K = 17K$
3300	$0 \leq 3300 < 4K$ (✓)	$3300 + 16K = 3300 + 16(1024) = 19684$
4400	$0 \leq 4400 < 4096$ (F)	Segm. Fault (—)

### Datos:

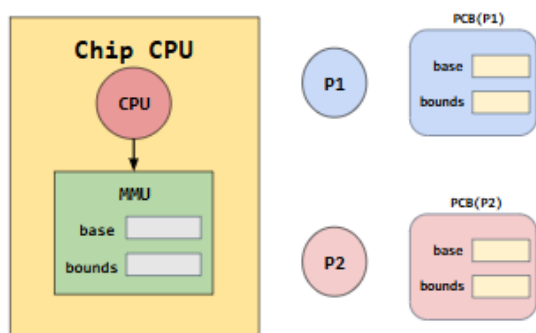
- base = 16 KB
- bounds = 4KB

Virtual Address (VA)	Physical Address (PA)
0	16KB
1 KB	17KB
3300	19684
4400	Dirección fuera de rango



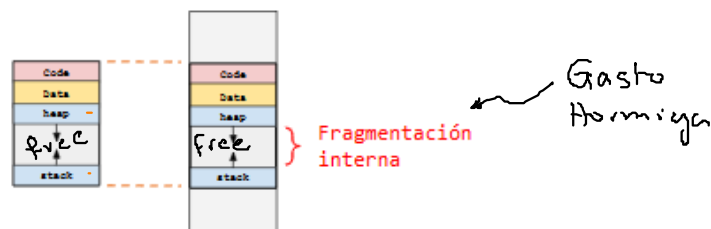
### Ventajas

- Rápida y simple. ✓
- Ofrece protección ✓
- Poco overhead (2 registros por proceso) ✓



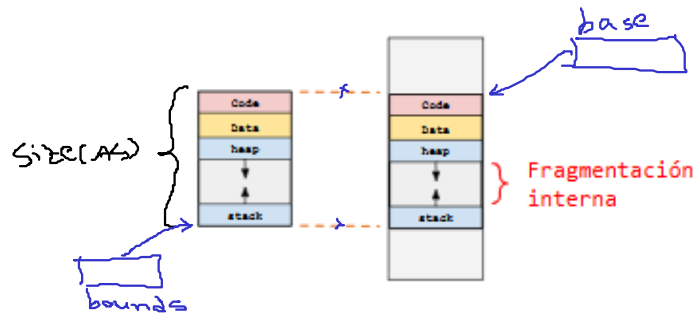
### Desventajas

- No es flexible.
- Gasto de memoria (sobre todo para espacios de direccionamiento grandes) → Fragmentación interna.



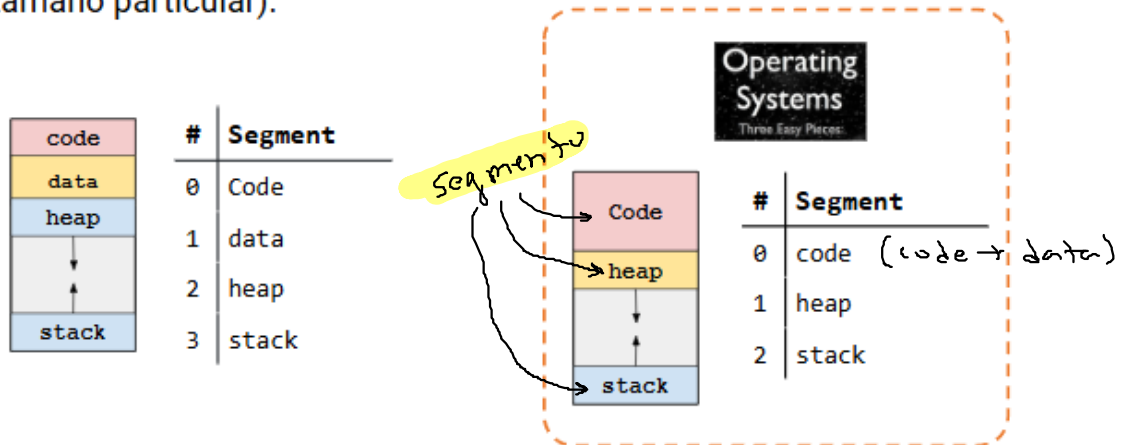
## ¿Cómo soportar grandes espacio de direcciones?

- ¿Cómo soportar un espacio de direcciones grande con un (potencial) espacio libre grande entre el stack y el heap?

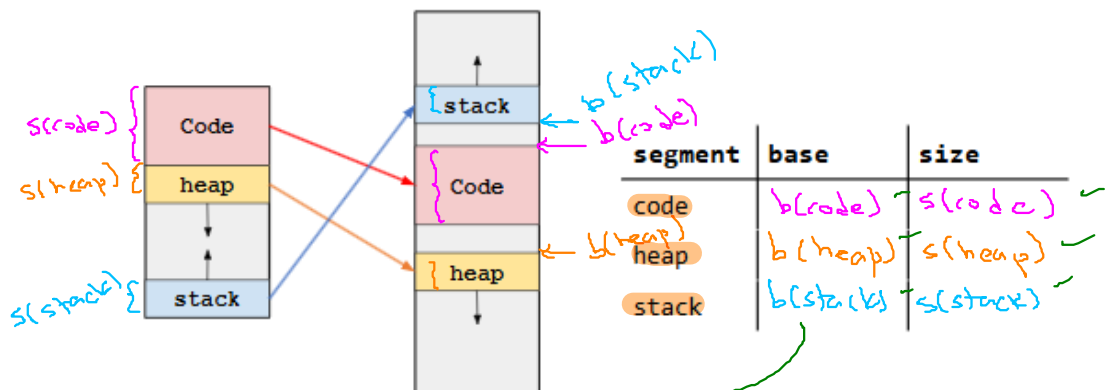


### 1. Segmento

- Segmento:** porción contigua del espacio de direccionamiento (con un tamaño particular).

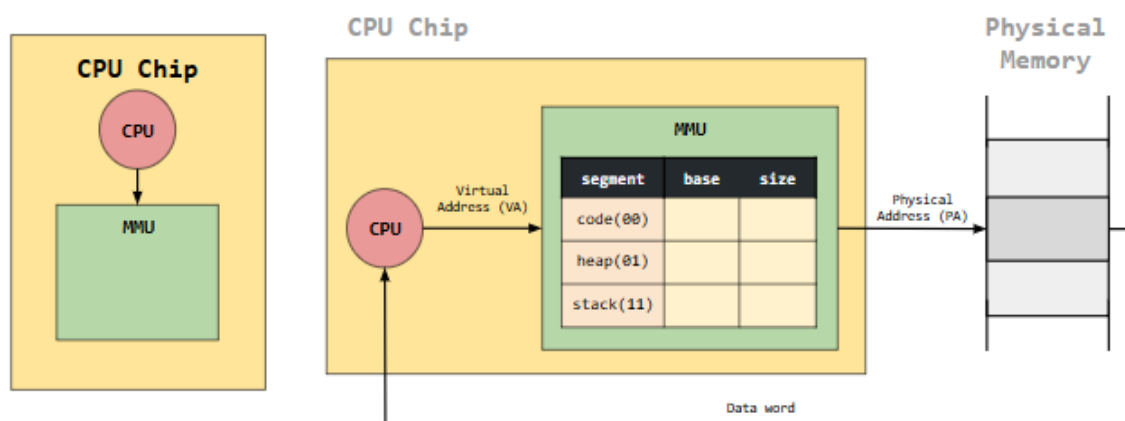
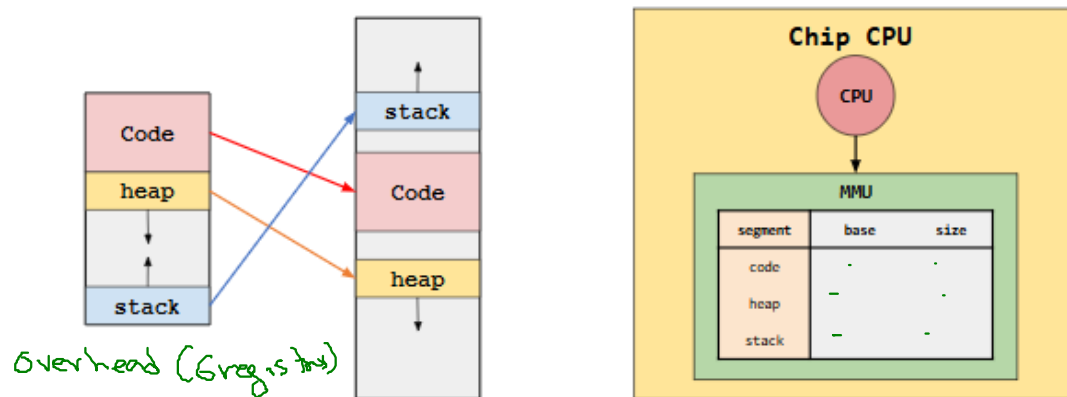


- Idea clave:** Cada segmento puede ser ubicado en una parte diferentes de la memoria física.

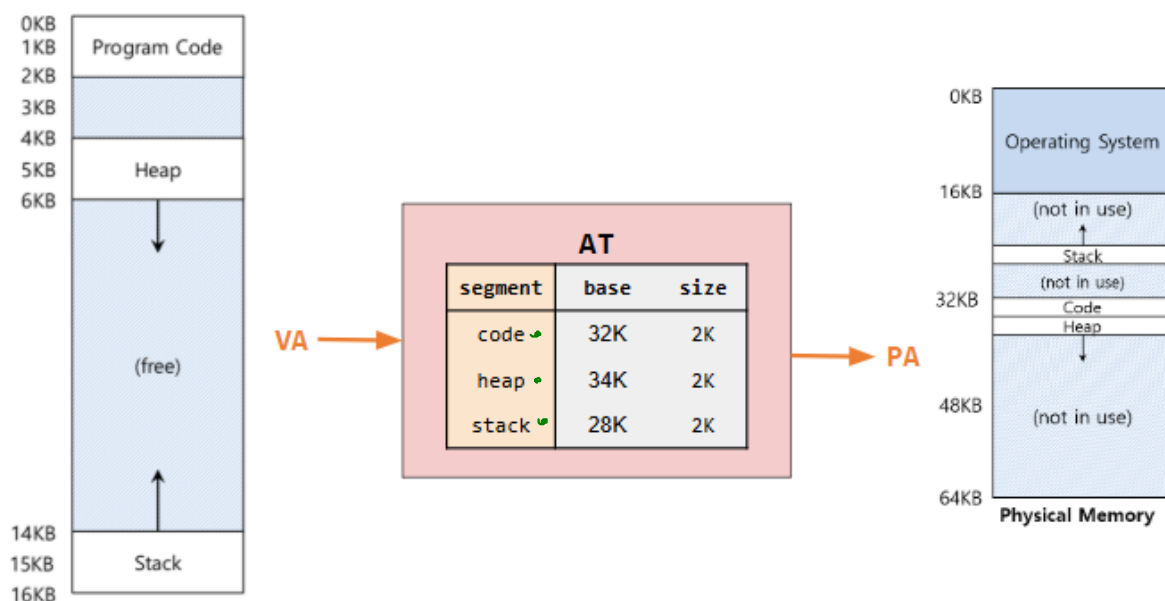


Segmentación = Base & Bound generalizado.

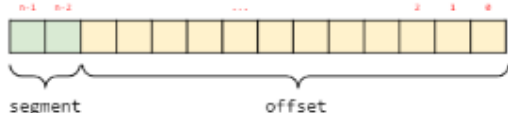
- Cada segmento puede ser ubicado en una parte diferentes de la memoria física tiene sus propios registros **base & bounds**.
- La **MMU** tendrá una tabla con tres pares de registros donde cada uno de estos pares estará relacionado a un segmento.



## 2. Traducción de direcciones (Parte 1)





segment	offset	Physical Address
code	offset = VA	PA = offset + base(code)
heap	offset = VA - VA(heap)	PA = offset + base(heap)
stack	 offset(stack) = offset - offset(max)	PA = offset + base(stack)

¿Como tengo en cuenta esta tabla en la VA?

#### Datos:

- Size(AS) = 16KB
- Size(PM) = 64KB

bits	segment	base	size
00	code	32K	2K
01	heap	34K	2K
11	stack	28K	2K
---	---	---	---

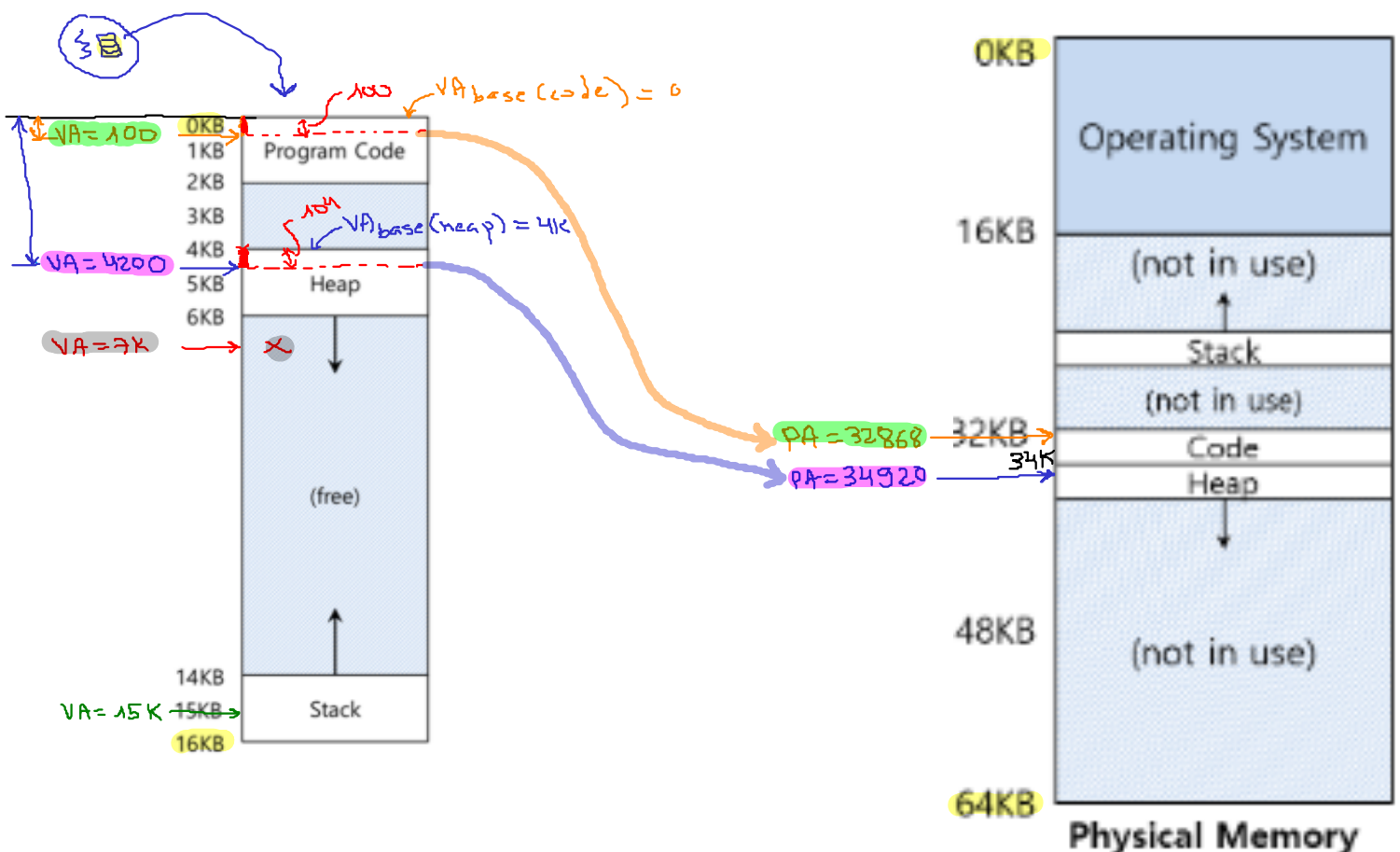
Segment	Virtual Address (VA)	Physical Address (PA)
code	100	32868
heap	4200	34920
---	7K	---
stack	15K	27K

Ademas de lo anterior se da el mapa de memoria del Address Space

#### Formulas

seg: heap, code, stack

- ①  $\text{offset} = \text{VA} - \text{VA}_{\text{base}}(\text{seg})$  ②  $0 \leq \text{offset} < \text{size}(\text{seg})$  ③  $\text{PA} = \text{offset} + \text{base}(\text{seg})$





Realizar la siguiente traducción de direcciones:

1. ¿Cual es la PA de VA = 100 ?

$$VA = 100 \longrightarrow PA = ? \quad (\text{code})$$

① Offset

$$\text{Offset} = VA - V_{\text{base}}(\text{code}) = 100 - 0 = 100$$

② Check Limits

$$0 \leq \text{Offset} < \text{size}(\text{code}) \longrightarrow 0 \leq 100 < 2K \quad \checkmark \quad (\text{ok})$$

③ PA

$$\begin{aligned} PA &= \text{offset} + \text{base}(\text{code}) = 100 + 32K = 100 + 32(2^{10}) \\ &= 100 + 32768 \\ &= 32868 \end{aligned}$$

Solucion:

$$VA = 100 \longrightarrow PA = 32868$$

2. ¿Cual es la direccion para VA = 4200 ?

$$VA = 4200 \longrightarrow PA = ? \quad (\text{segment} = \text{heap})$$

① Offset

$$\text{Offset} = VA - V_{\text{base}}(\text{heap}) = 4200 - 4K = 4200 - 4(2^{10}) = 104$$

② Check Limits

$$0 \leq \text{Offset} < \text{size}(\text{heap}) \longrightarrow 0 \leq 104 < 2K \quad \checkmark \quad (\text{ok})$$

③ PA

$$\begin{aligned} PA &= \text{offset} + \text{base}(\text{heap}) = 104 + 34K = 104 + 34(2^{10}) \\ &= 104 + 34816 \\ &= 34920 \end{aligned}$$

Solución:

$$VA = 4200 \rightarrow PA = 34920$$

2. ¿Cuál es la PA de  $VA = 7K$ ?

$$VA = 7K \rightarrow PA = ? \quad (\text{segment} = \text{heap})$$

① Offset

$$\text{Offset} = VA - V_{\text{base}}(\text{heap}) = 7K - 4K = 3K$$

② Check Limits

$$0 \leq \text{Offset} < \text{size}(\text{heap}) \rightarrow 0 \leq 3K < 2K \quad \times \quad (\text{seg. fault})$$

③ PA

$$PA = \text{---} \quad (\text{Segmentation fault})$$

Solución: No hay dirección Física.

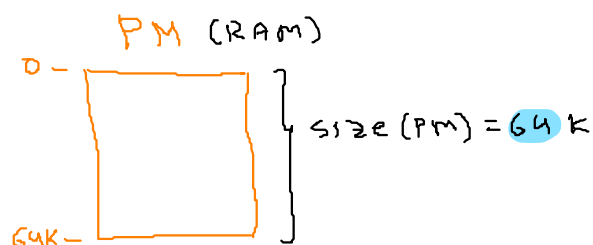
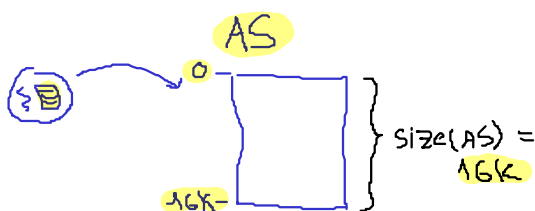
$$VA = 7K \rightarrow PA = \text{---}$$

Conclusión: La siguiente tabla muestra las direcciones:

VA	Segmento	PA
100	code	32868
4200	heap	34920
7K	heap	---

← Direcciones.  
↓ Ancho  
↓ n (Bits)

4. Sobre las direcciones: Recordemos que para este ejemplo teníamos



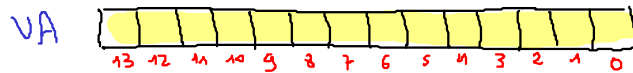
¿Cuántos bits necesito para representar una dirección?

a. Virtual:

$n = \text{numero de bits}$

$$\begin{aligned} \text{Size (AS)} &= 2^n \rightarrow 16K = 2^n \\ (2^4)(2^{10}) &= 2^n \\ 2^{14} &= 2^n \\ \downarrow \\ 14 &= n \end{aligned}$$

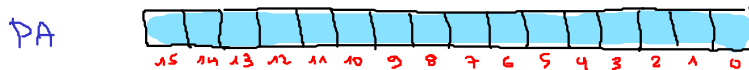
Formato de una VA (Virtual Address)



b. Física

$$\begin{aligned} \text{Size (PM)} &= 2^n \rightarrow 64K = 2^n \\ (2^6)(2^{10}) &= 2^n \\ 2^{16} &= 2^n \\ \downarrow \\ n &= 16 \end{aligned}$$

Formato de una Physical Address (PA)



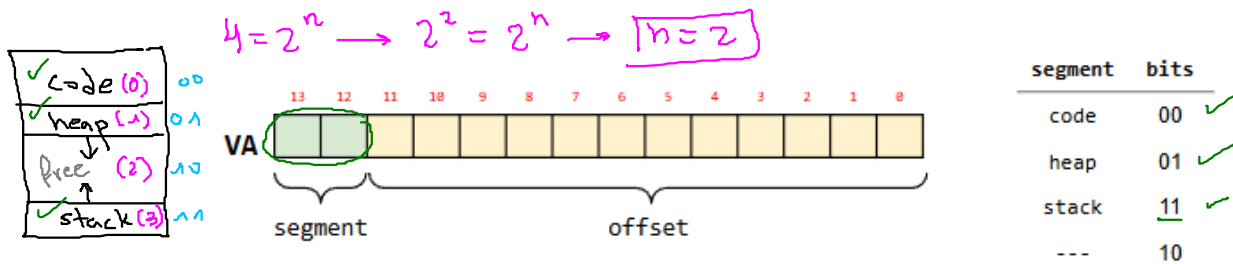
La siguiente tabla muestra las direcciones en formato hexadecimal y binario para las direcciones previamente calculadas.

VA ( $n = 14$ bits)	PA ( $n = 16$ bits)
$100 = 0x64 =$ 	$32868 = 0x8064 =$ 
$4200 = 0x1068 =$ 	$34920 = 0x8868 =$ 

### 3. Formato de dirección (al usar segmentación)

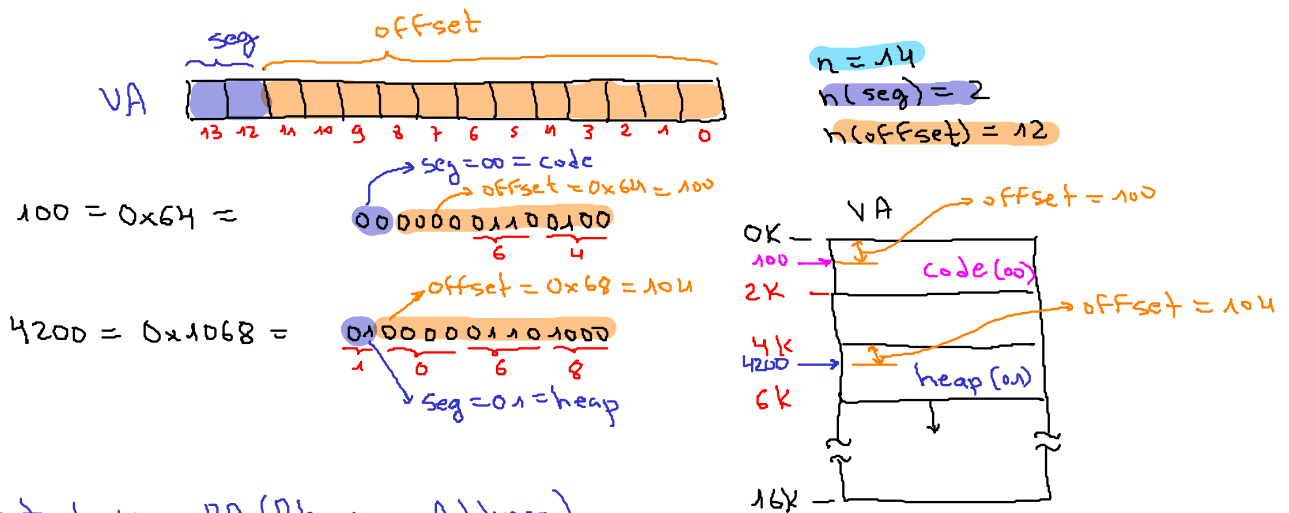
Divide el espacio de direccionamiento de tal manera que una dirección virtual está compuesta por dos partes [segment|offset]

- **segment:** Compuesto por los bits más significativos (top bits) de la dirección lógica.
- **Offset dentro del segmento elegido:** Parte compuesta por los bits menos significativos que hace referencia al offset respecto al segmento seleccionado.



Usando segmentación: ¿Cual seria el Formato para cada una de las direcciones virtuales previamente obtenidas?

Formato de una VA (Virtual Address) usando segmentación



Formato de una PA (Physical Address)



32868 = 0x8064 =  $1000000001100100$

34920 = 0x8868 =  $1000100001101000$

## Traducción de direcciones - Pseudocódigo

VA seg | offset

```

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)
    
```

Consultar la tabla de segmentos.

- WSEG\_MASK = 0x3000 (11000000000000)
- WSEG\_SHIFT = 12
- WOFFSET\_MASK = 0xFFF (00111111111111)

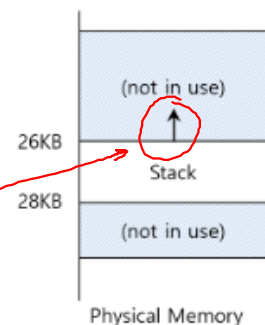
### 4. Traducción de direcciones (parte 2)

Stack es un caso especial

- La pila crece **hacia atras (grows background)**
- Es necesario agregar soporte de hardware adicional que indique el sentido de crecimiento del segmento:
  - **0**: Crecimiento en dirección negativa
  - **1**: Crecimiento en dirección positiva

segment	base	size (max 4K)	Grows Positive?
code(00)	32K	2K	1
heap(01)	34K	2K	1
stack(11)	28K	2K	0

Segment Registers (With Negative-Growth Support)

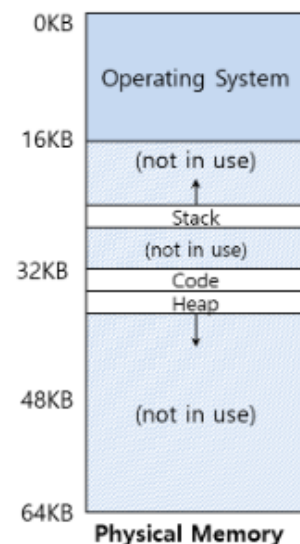


### Ejemplo

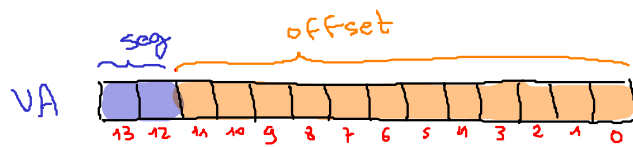
Para el mismo ejemplo que se ha venido trabajando y teniendo en cuenta los segmentos mostrados en la tabla adjunta. Si **VA = 15K**, determine:

1. ¿A cuál segmento pertenece dicha VA?
2. ¿Cual es el offset?
3. ¿Cual es la dirección física?

segment	base	size (max 4K)	Grows Positive?
code(00)	32K	2K	1
heap(01)	34K	2K	1
stack(11)	28K	2K	0



Solución: Recordemos que el formato de dirección para VA es:



$$n = 14$$

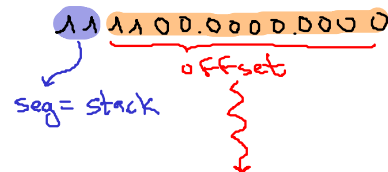
$$n(seg) = 2$$

$$n(offset) = 12$$

segment	bits
code	00
heap	01
stack	11
---	10

$$VA = 15K = 15(2^{10}) = 15360 = 0 \times \begin{matrix} 0011 & 1100 & 0000 & 0000 \end{matrix}$$

↓ (14 bits)



Como sería el caso aca?

bit de signo

Complemento a 2

$$11110000000000 + 1 = (-1)(2^{11}) + (1)(2^{10}) + 0(2^9) + \dots + 0(2^0)$$

$$= -1024 = -(2^{10}) = -1K$$

← crece hacia arriba

a. Cual es el offset maximo si  $n(offset) = 12$  bits

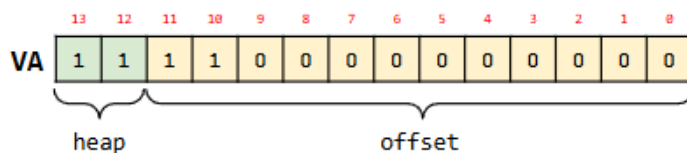
$$n \text{ bits: } 0 \leq dir < 2^n - 1$$

$$\text{luego: } n(VA) = 12 \rightarrow 0 \leq dir < 2^{12} - 1$$

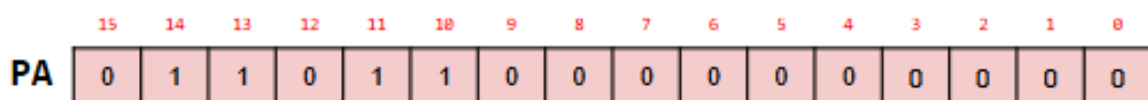
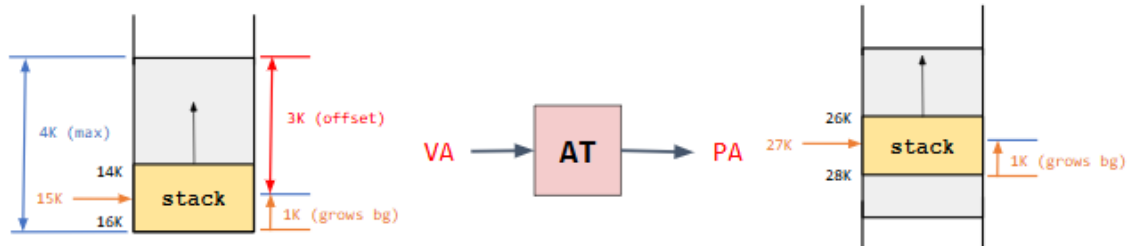
$$0 \leq dir < 4095$$

Hay 4096 direcciones:  $[0, 4095]$

4K (Direcciones)



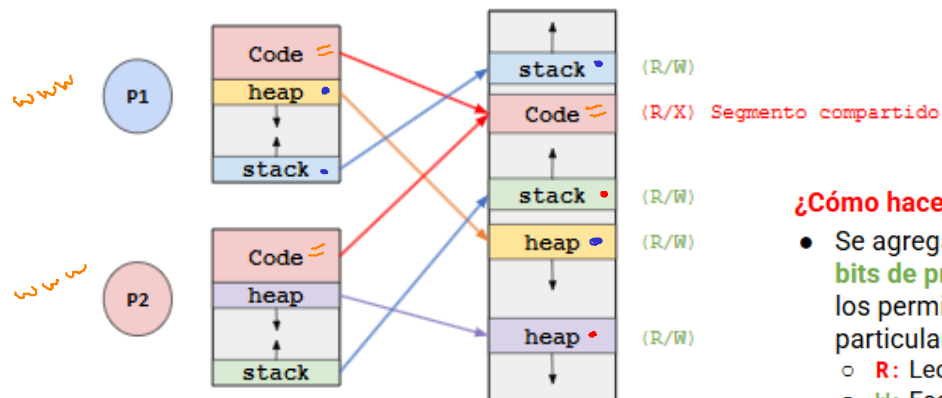
segment	base	size (max 4K)	Grows Positive?
code(00)	32K	2K	1
heap(01)	34K	2K	1
stack(11)	28K	2K	0



## 5. Segmentos compartidos

### ¿Que pasa cuando se ejecutan dos procesos con la misma imagen?

- Algunos segmentos (como el segmento de código) podrían compartirse



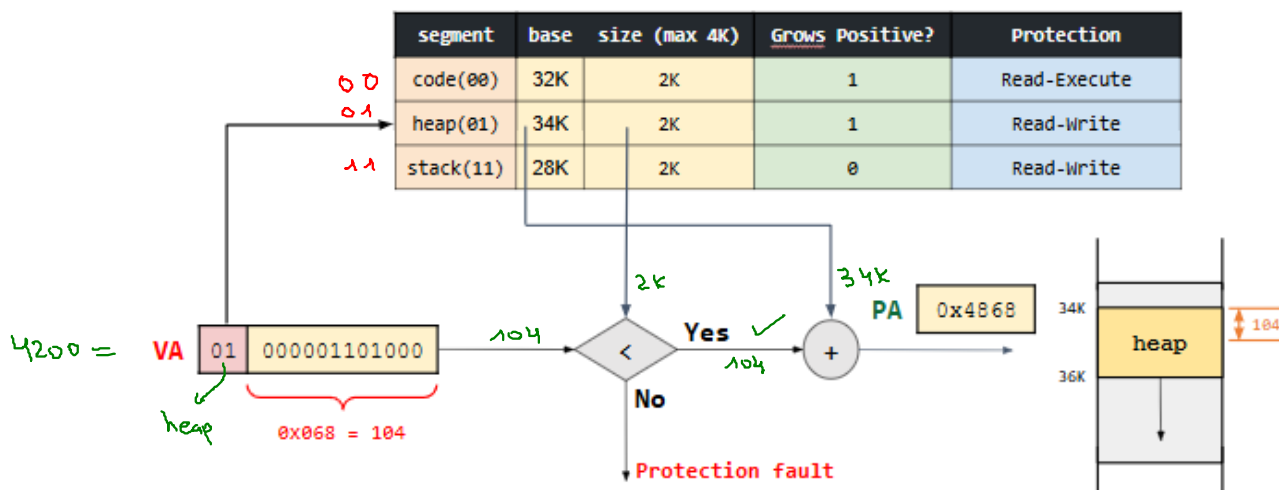
### ¿Cómo hacerlo?

- Se agrega HW adicional usando **bits de protección** para indicar los permisos de un segmento en particular:
  - R: Lectura
  - W: Escritura
  - X: Ejecución

segment	base	size (max 4K)	Grows Positive?	Protection
code(00)	32K	2K	1	Read-Execute
heap(01)	34K	2K	1	Read-Write
stack(11)	28K	2K	0	Read-Write


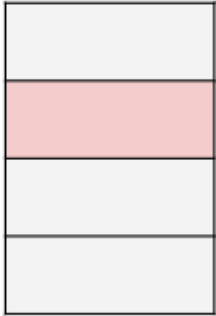

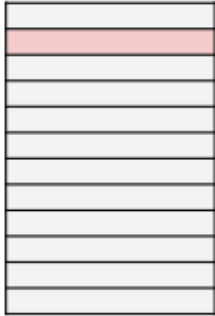
Segment Register Values (with Protection)

## 6. Hardware de segmentación





## 7. Granularidad en la segmentación

Grano grueso (Coarse-grained)	Grano fino (Fined-grained)
<ul style="list-style-type: none"> <li>Se utilizan pocos segmentos relativamente grandes (ej: code, heap, stack)</li> </ul>	<ul style="list-style-type: none"> <li>Emplea muchos segmentos pequeños <ul style="list-style-type: none"> <li>Mayor flexibilidad.</li> <li>Requiere más soporte de hw (Tabla de segmentos).</li> </ul> </li> </ul>
 	 

## 8. Ventajas y desventajas

Ventajas	Desventajas																								
<ul style="list-style-type: none"><li>Permite espacios de direcciones pequeños. El stack y el heap pueden crecer independientemente.</li><li>Ahorro de memoria al compartir segmentos.</li><li>Facilita la protección de los segmentos.</li></ul>	<ul style="list-style-type: none"><li><b>Fragmentación externa:</b> Pequeños huecos de espacio libre en memoria física difíciles de utilizar</li></ul> <p><b>Ejemplo:</b> Dada la siguiente memoria física, responder:</p> <ul style="list-style-type: none"><li>¿Cuanta memoria memoria tiene ocupada? <b>Rta:</b> 40K</li><li>¿Cuanta memoria libre? <b>Rta:</b> 24K, pero no están contiguos los segmentos.</li><li>¿Es posible ubicar un segmento con un tamaño de 20K? <b>Rta:</b> No es posible</li></ul> <table><tr><td>0KB</td><td>8KB</td><td>Operating System</td></tr><tr><td>8KB</td><td>16KB</td><td>(not in use) 8K</td></tr><tr><td>16KB</td><td>24KB</td><td>Allocated ✓</td></tr><tr><td>24KB</td><td>32KB</td><td>(not in use) 4K</td></tr><tr><td>32KB</td><td>40KB</td><td>Allocated ✓</td></tr><tr><td>40KB</td><td>48KB</td><td>(not in use) 12K</td></tr><tr><td>48KB</td><td>56KB</td><td>Allocated ✓</td></tr><tr><td>56KB</td><td>64KB</td><td></td></tr></table>	0KB	8KB	Operating System	8KB	16KB	(not in use) 8K	16KB	24KB	Allocated ✓	24KB	32KB	(not in use) 4K	32KB	40KB	Allocated ✓	40KB	48KB	(not in use) 12K	48KB	56KB	Allocated ✓	56KB	64KB	
0KB	8KB	Operating System																							
8KB	16KB	(not in use) 8K																							
16KB	24KB	Allocated ✓																							
24KB	32KB	(not in use) 4K																							
32KB	40KB	Allocated ✓																							
40KB	48KB	(not in use) 12K																							
48KB	56KB	Allocated ✓																							
56KB	64KB																								

**Solución: Compactación:** Reorganización de los fragmentos existentes en memoria física.

Not compacted	Compacted
0KB 8KB Operating System 16KB (not in use) 24KB Allocated 32KB (not in use) 40KB Allocated 48KB (not in use) 56KB Allocated 64KB	0KB 16K 8KB Operating System 16KB 24KB 24K Allocated 32KB 40KB 48KB 24K (not in use) 56KB 64KB

**Compactación** (Costoso)

**Métodos:**

1. Best-Fit
2. First-Fit
3. Worse-Fit