Hacia la memoria virtual

Recordatorio: fases de la compilación

```
int nota[128];
int media[128];
for (i=0;i<128;i++) {
  if (i>7 &&i<64) {
    nota[i] = media[i]/2;
  }
  else {
    a = nota[i]*media[i]
  }
}
```

Ensamblado

```
.data
nota:  .word 128
media: .word 128
.text
1  bucleFor: cmp r0,#128
2         beq finBucle
3         cmp r0,#7
4         bne ramaElse
5         cmp r0,#64
6         bge ramaElse
7         ldr r1,[r4,r0,lsl#2]
8         lsr r1,r1,#1
9         str r1,[r5,r0,lsl#2]
10        b finIf
11 ramaElse: ldr r1,[r4,r0,lsl#2]
12        ldr r2,[r5,r0,lsl#2]
13        mul r3,r1,r2
14 finIf:    add r0,r0,#1

15        b bucleFor
16 finBucle: b .
```

- ¿Cuántos bloques de cache ocupa el código?
- ¿Serán bloques consecutivos? ¿Habrá conflictos?

Enlazado

```
.data
nota:  .word 128
media: .word 128
.text
1  bucleFor: cmp r0,#128
2         beq finBucle
3         cmp r0,#7
4         bne ramaElse
5         cmp r0,#64
6         bge ramaElse
7         ldr r1,[r4,r0,lsl#2]
8         lsr r1,r1,#1
9         str r1,[r5,r0,lsl#2]
10        b finIf
11 ramaElse: ldr r1,[r4,r0,lsl#2]
12        ldr r2,[r5,r0,lsl#2]
13        mul r3,r1,r2
14 finIf:    add r0,r0,#1

15        b bucleFor
16 finBucle: b .
```

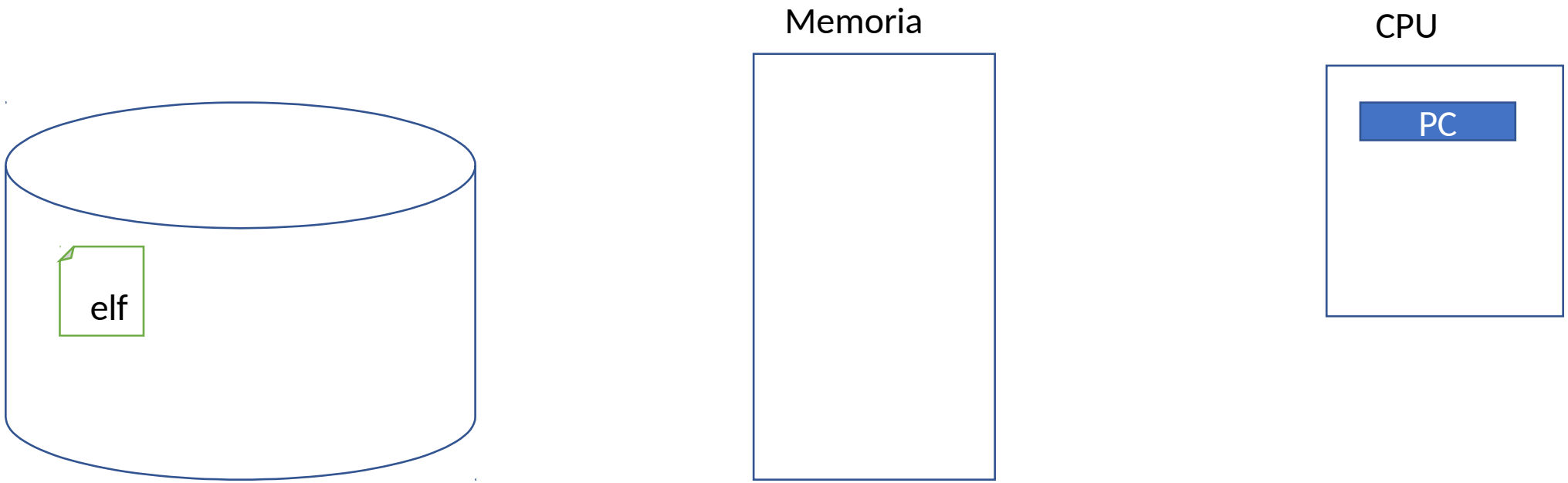
Enlazado

Fichero ELF

.data 0x000000000
.text 0x0C0000000
entry 0x0C0000000
Symbol table
…
0100100010001
0000100101010
….

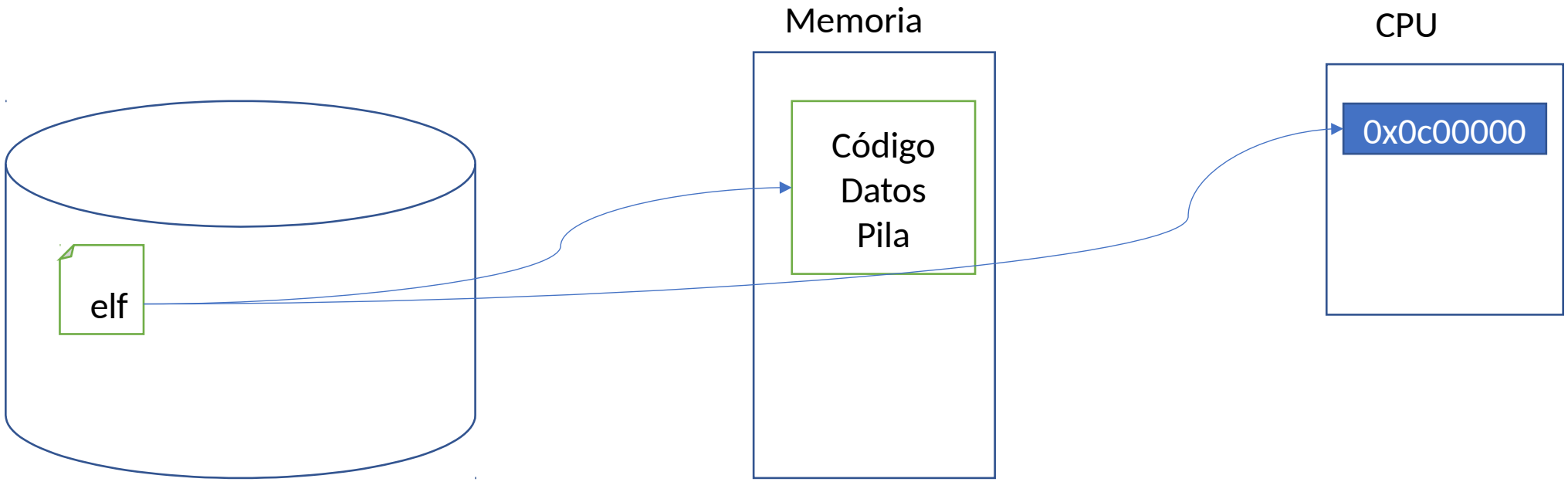
Sistema real

- El ejecutable se almacenará en el disco



Ejecución

- Escribir parte del ejecutable en memoria
  - Al menos, las instrucciones y los datos globales.
  - Además, crear espacio para la pila
- Escribir en PC la dirección de la primera instrucción
  - El *entry point* especficiado en el fichero ELF



- **¿En qué direcciones de memoria se escriben el código y los datos?**

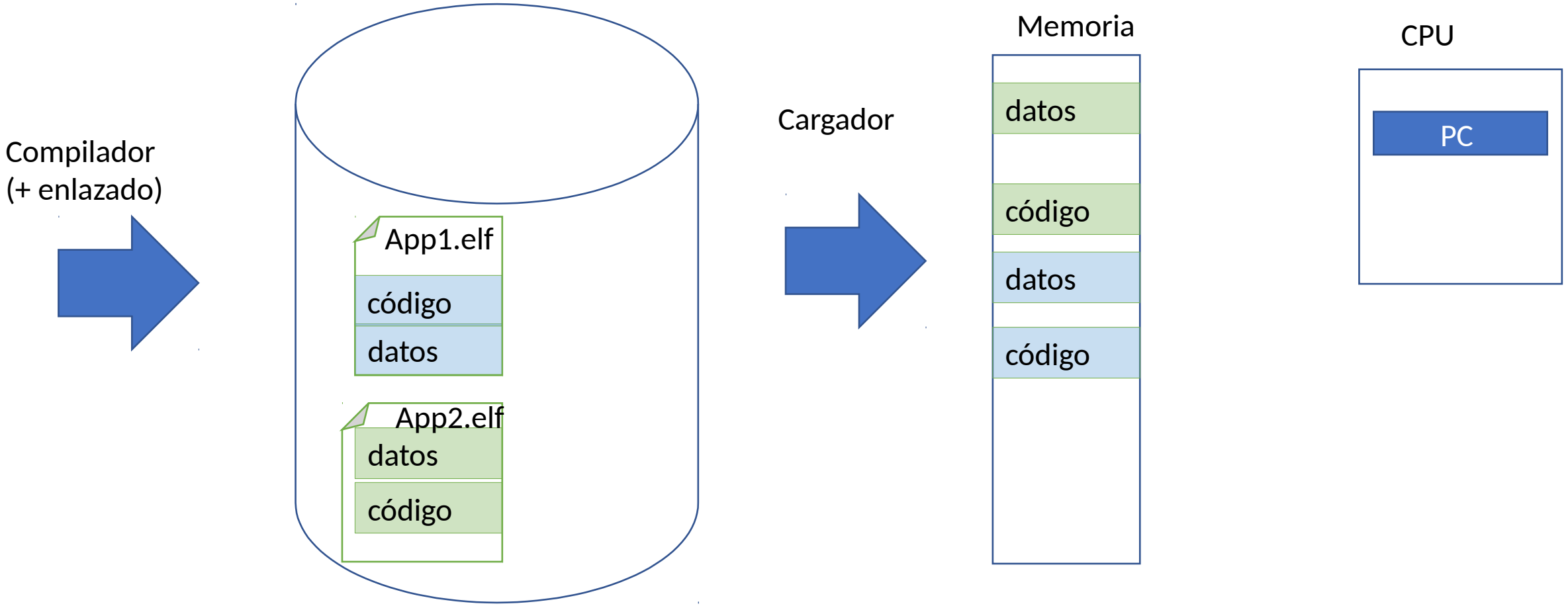
Sobre le ejecución....

- ¿ Y, si antes de que termine esa aplicación, ejecutamos otra?
  - El proceso será similar: escribir en memoria las instrucciones y datos…
  - **¿En qué direcciones escribiremos su código / datos?**
- ¿Puede el compilador asegurar que generará direcciones diferentes para cada posible aplicación que lancemos?
  - ¿Y si ejecutamos dos veces la misma aplicación? (sin dejar que termine la primera instancia)

Más problemas....

- ¿Y si nuestra aplicación es tan grande (muchas instrucciones, muchos datos..) que no cabe en memoria principal?
  - Recordar el maletín del laboratorio:
    - 32 bits de direcciones
    - Hasta 4GB de espacio direccionable
    - Pero sólo unos pocos MB de memoria SDRAM real
    - Y otros pocos Kbytes del espacio dedicados a dispositivos de E/S

¿Quién/cuándo asigna direcciones?



¿Quién/cuándo asigna direcciones?

- **¿Quién?**
  - Compilador/enlazador
  - Cargador (parte del SO)
- **¿Cuándo?**
  - En tiempo de compilación/enlazado
  - En el momento de la carga del ejecutable a memoria
  - Durante la ejecución

IDEA FUNDAMENTAL

- La aplicación/tarea y el procesador manejan ***direcciones virtuales*** (también llamadas *lógicas* en algún contexto)
  - Son las generadas por el compilador/enlazador
- Durante la ejecución la MMU (Memory Management Unit) **traduce** las *direcciones virtuales* en *direcciones físicas*
  - Las direcciones físicas son las que se envían al controlador de memoria para solicitar el acceso a memoria principal

¿Con qué granularidad hacemos la traducción?

- Por secciones
  - se traduce la dirección de comienzo de *.text*, *.data*….
  - Se mantiene la organización interna de cada sección (como la decidió el compilador)
  - A favor: la traducción de cada sección consiste únicamente en sumar una constante
  - En contra: no soluciona el problema de los mapas virtuales grandes.

¿Con qué granularidad hacemos la traducción?

- Por palabras
  - Cada palabra del espacio virtual se traduce a una nueva ubicación en el espacio físico
  - La implementación se haría mediante una tabla con tantas entradas como palabras tenga nuestro espacio virtual
    - Si el procesador es de 32 bits, nuestras aplicaciones pueden tener espacios virtuales de 2^{32} bytes ✉ 2^{30} palabras
    - CADA aplicación (tarea) necesita una tabla con 2^{30} entradas, cada una de ellas de (por ejemplo) 4 bytes
      - IMPOSIBLE en la práctica

¿Con qué granularidad hacemos la traducción?

- Por conjuntos de palabras ✉ **página**
  - Cada **página** del espacio virtual se traduce a una nueva ubicación en el espacio físico (un marco de página o página física)
  - La implementación se haría mediante una tabla con tantas entradas como **páginas** tenga nuestro espacio virtual
    - Si el procesador es de 32 bits, nuestras aplicaciones pueden tener espacios virtuales de 2^{32} bytes
    - Tamaño de página típico: 4Kbytes
    - CADA aplicación (tarea) necesita una tabla con 2^{20} entradas, cada una de ellas de (por ejemplo) 4 bytes
      - Aún muy grande, pero veremos cómo hacerlo manejable