Tema 5. Gestión de memoria

Gonzalo Álvarez Dpto. Arquitectura y Tecnología de Computadores. UPV/EHU

Contenidos

1. Introducción. Estructura de un programa en memoria
2. Bibliotecas y montaje.
3. Bibliotecas de enlace dinámico.
4. Ubicación de programas en memoria.
5. Direccionamiento físico y virtual.
6. Soporte para memoria virtual.
7. Llamadas al sistema relacionadas con la carga de programas.

Bibliografía:

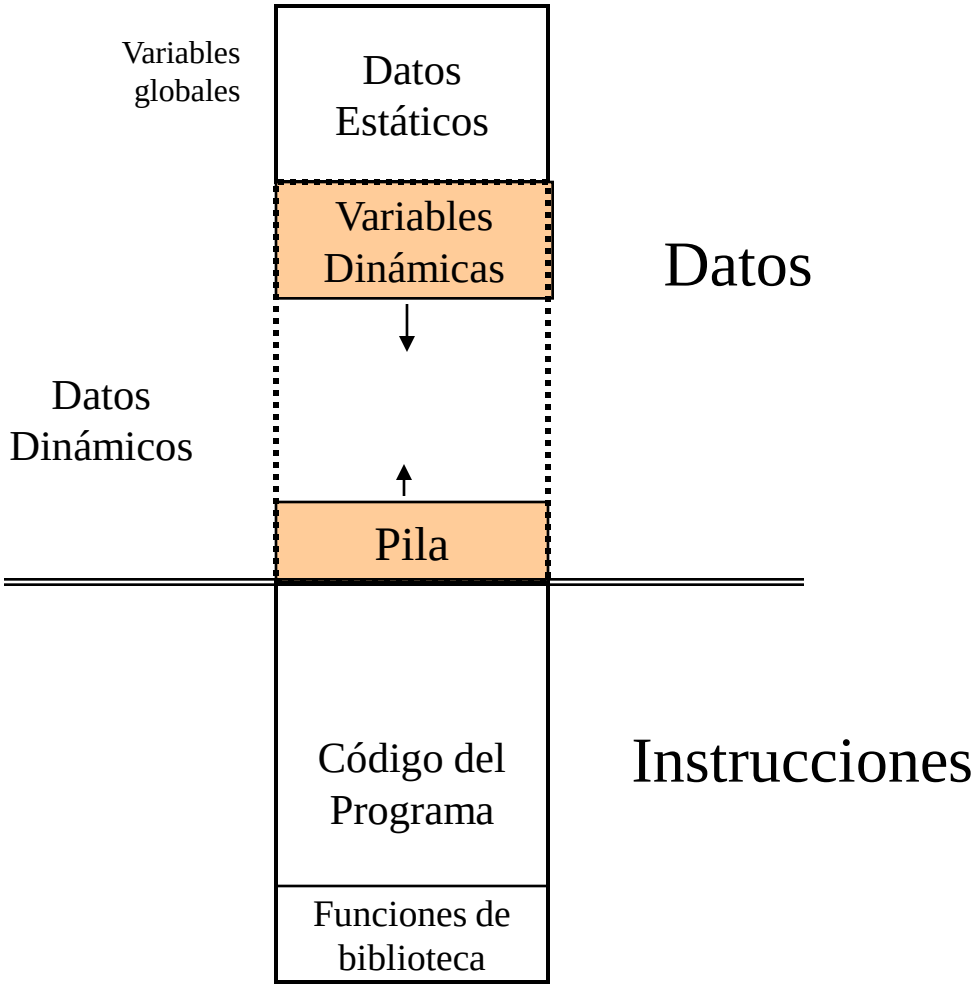
[Apartados 1.3 y 2.4] C. Rodríguez, I. Alegria, J. González, A. Lafuente: Descripción Funcional de los Sistemas Operativos. Síntesis, 1994

[Capítulos 7 y 8] W. Stallings: Sistemas Operativos. 5º Ed. Pearson Prentice-Hall, 2005.

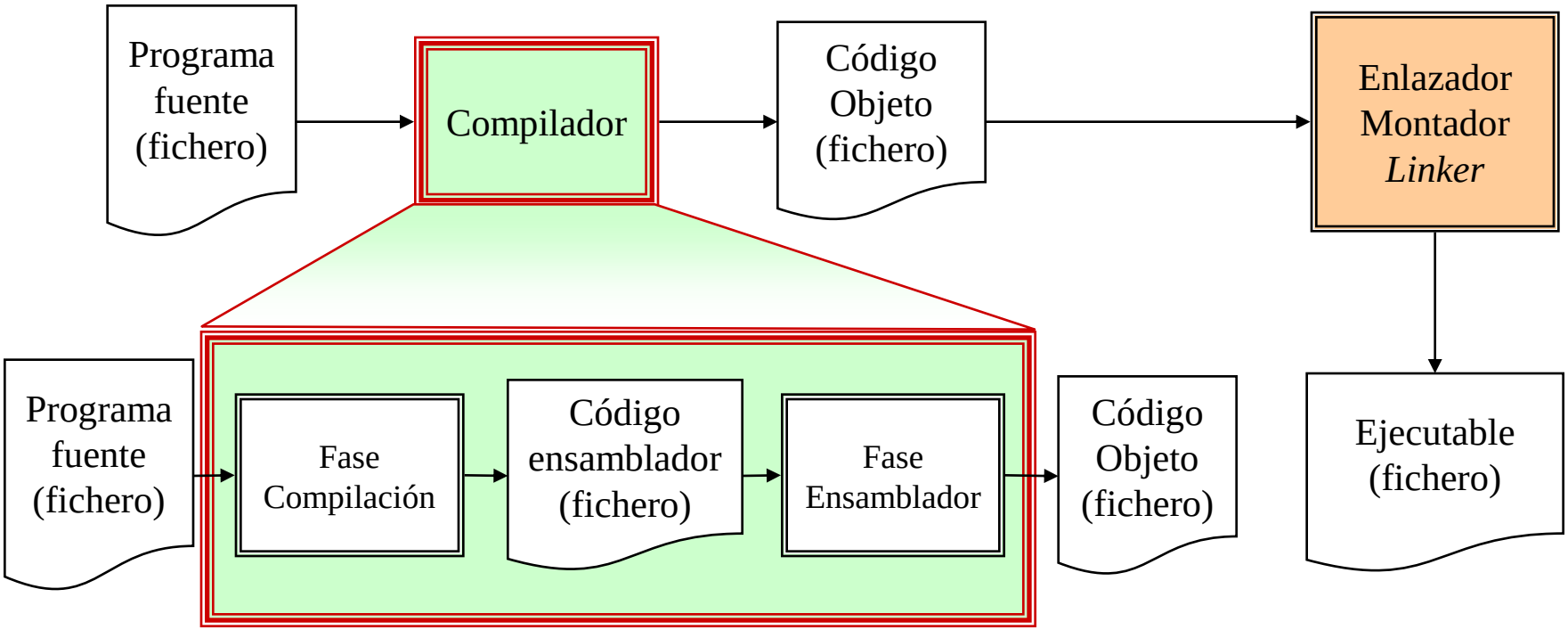
[Capítulos 11 y 12] G. Nutt: Sistemas Operativos. 3º Ed. Pearson Addison Wesley, 2001

[Capítulos 9 y 10] A. Silberschartz: Operating Systemas Concepts. Sixth, John Wiley & Son, 2003

5.1 Introducción: Estructura de un programa en memoria

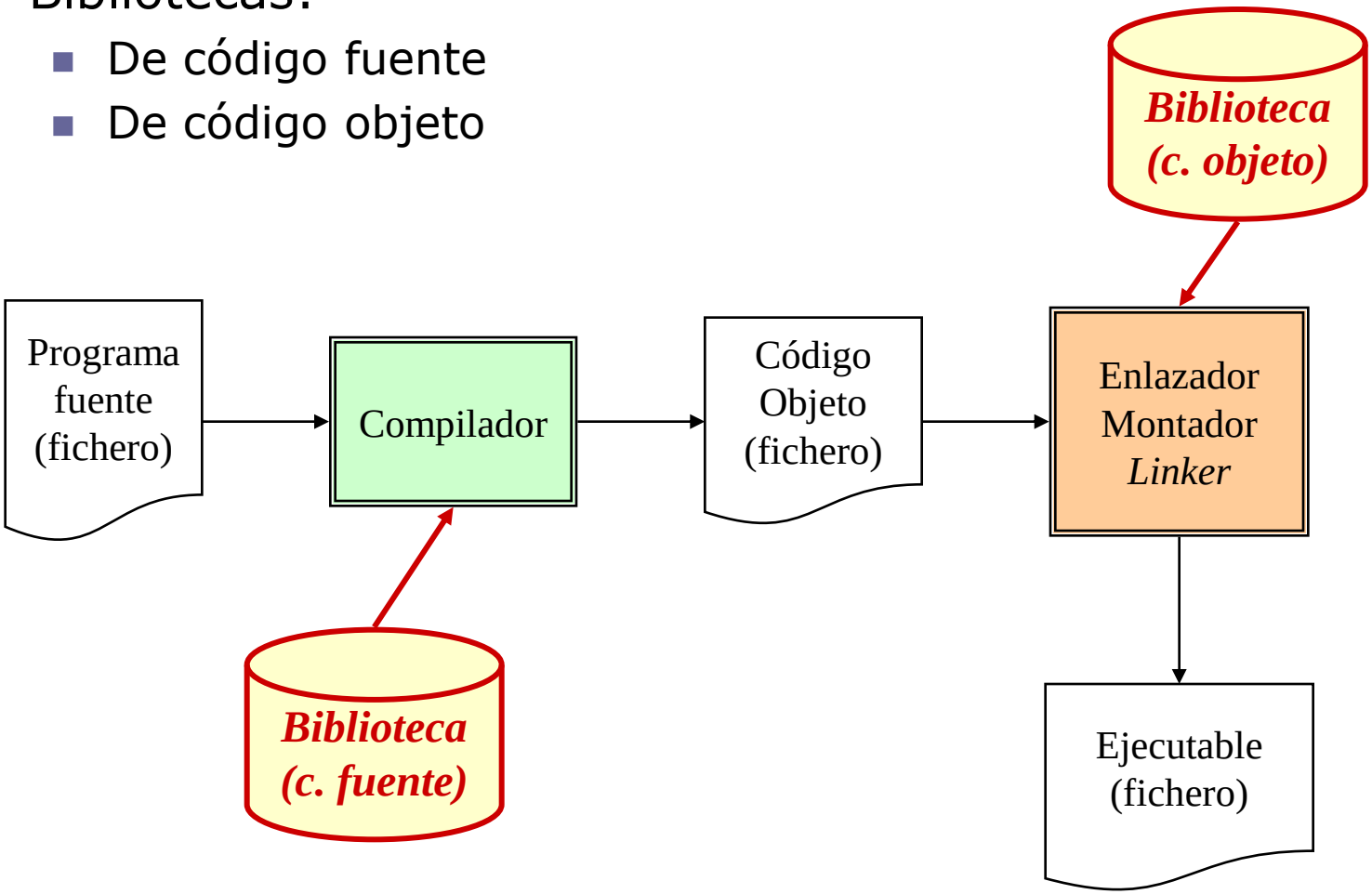


Compilación.



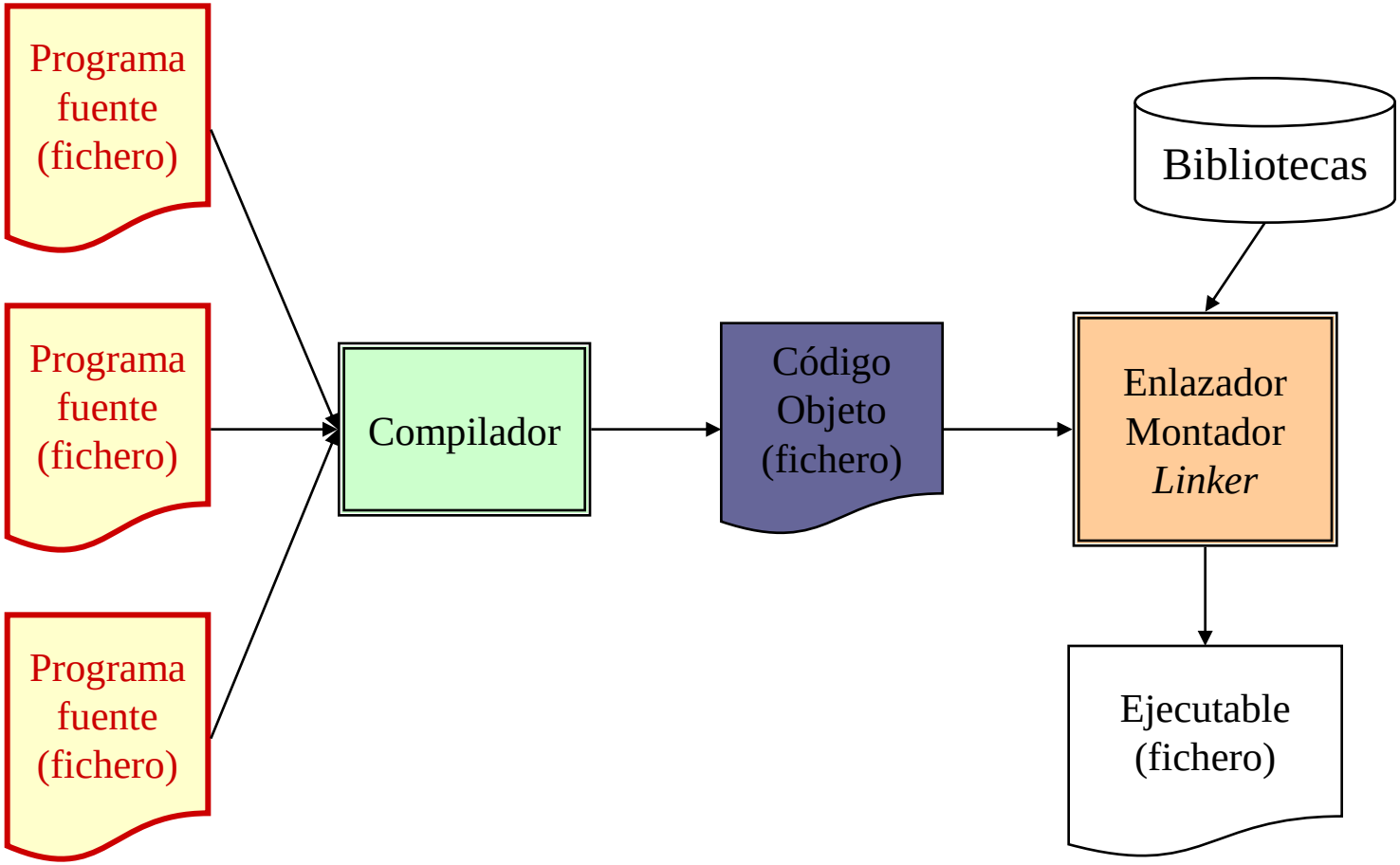
Compilación (I).

- Bibliotecas:
  - De código fuente
  - De código objeto



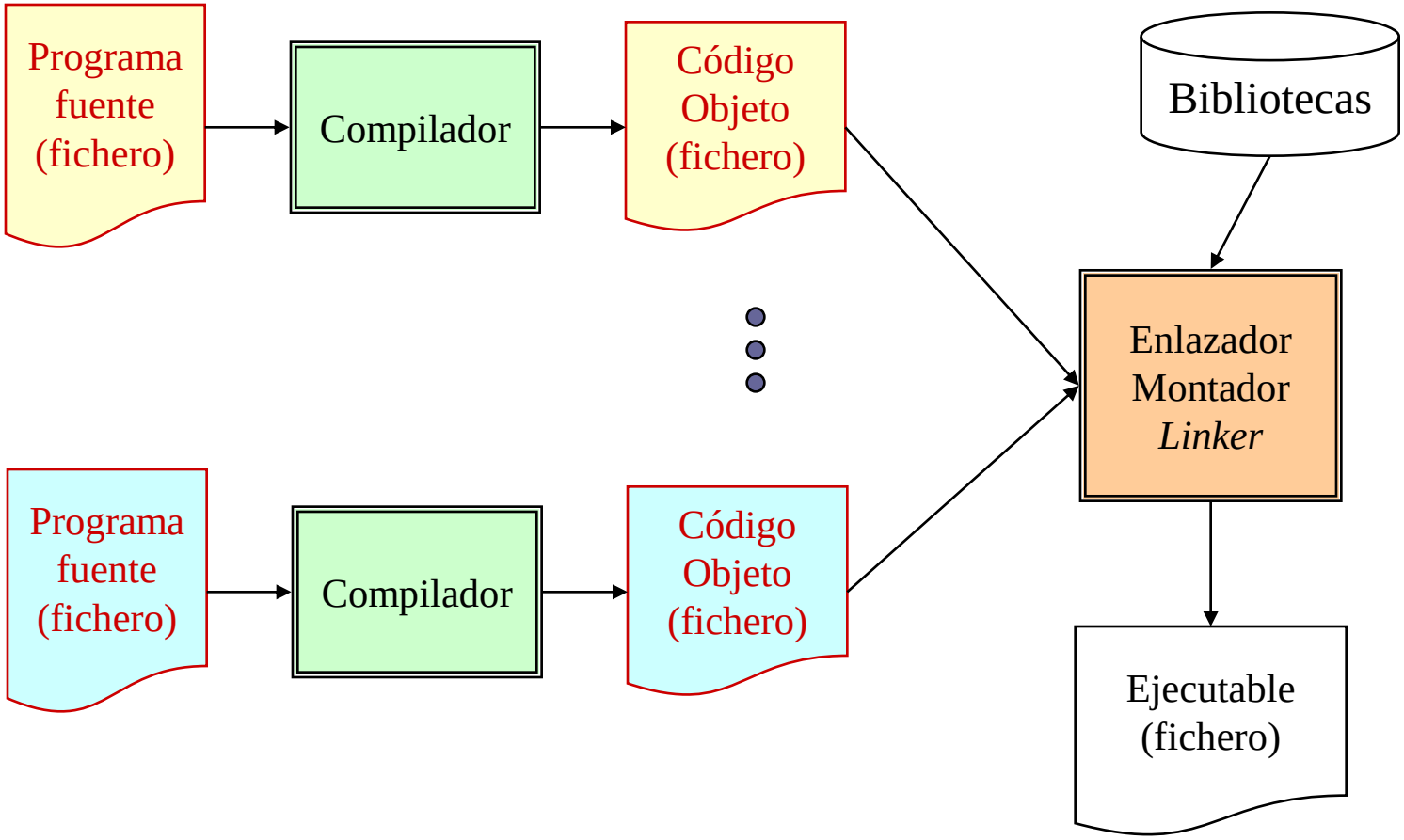
Compilación (II).

- Varios Fuentes ⇨ Un solo módulo objeto



Compilación (III).

- Varios Fuentes ⇨ Varios módulos objeto



Ejemplo de fases de compilación con gcc

- Tarea 5.1
  1. Utilizando la ayuda de *man* analiza cómo monitorizar las fases de una compilación.
  2. Prueba con la compilación del programa `copia.c`.
  3. Localiza y comenta cada una de las fases.
  4. Intenta probar con otra versión del compilador.

  Tarea individual.

  Entregable: Traza de la compilación comentado/señalando cada una de las fases.

  Tiempo estimado de realización 0:20 min.

```
[[acaf0000@g000008 tema5]$ gcc -v  copia.c -o copia
Usando especificaciones internas.
Objetivo: x86_64-redhat-linux
Configurado con: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --
enable-shared --enable-threads=posix --enable-checking=release --with-system-zlib --enable-
__cxa_atexit --disable-libunwind-exceptions --enable-libgcj-multifile --enable-
languages=c,c++,objc,obj-c++,java,fortran,ada --enable-java-awt=gtk --disable-dssi --enable-plugin
--with-java-home=/usr/lib/jvm/java-1.4.2-gcj-1.4.2.0/jre --with-cpu=generic --host=x86_64-redhat-
linux
Modelo de hilos: posix
gcc versiÃ³n 4.1.2 20080704 (Red Hat 4.1.2-46)
 /usr/libexec/gcc/x86_64-redhat-linux/4.1.2/cc1 -quiet -v copia.c -quiet -
dumpbase copia.c -mtune=generic -auxbase copia -version -o /tmp/cc825pqw.s
ignorando el directorio inexistente "/usr/lib/gcc/x86_64-redhat-linux/4.1.2/../../../../x86_64-
redhat-linux/include"
la bÃºsqueda de #include "..." inicia aquÃ-:
la bÃºsqueda de #include <...> inicia aquÃ-:
 /usr/local/include
 /usr/lib/gcc/x86_64-redhat-linux/4.1.2/include
 /usr/include
Fin de la lista de bÃºsqueda.
GNU C versiÃ³n 4.1.2 20080704 (Red Hat 4.1.2-46) (x86_64-redhat-linux)
        compilado por GNU C versiÃ³n 4.1.2 20080704 (Red Hat 4.1.2-46).
GGC heurÃ-sticas: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
Compiler executable checksum: 1ea42026fd4a888d943ded4a09bd850e
 as -V -Qy -o /tmp/ccuSzfem.o /tmp/cc825pqw.s
GNU assembler version 2.17.50.0.6-12.el5 (x86_64-redhat-linux) using BFD version 2.17.50.0.6-
12.el5 20061020
 /usr/libexec/gcc/x86_64-redhat-linux/4.1.2/collect2 --eh-frame-hdr -m elf_x86_64 --hash-
style=gnu -dynamic-linker /lib64/ld-linux-x86-64.so.2 -o copia /usr/lib/gcc/x86_64-redhat-
linux/4.1.2/../../../../lib64/crt1.o /usr/lib/gcc/x86_64-redhat-
linux/4.1.2/../../../../lib64/crti.o /usr/lib/gcc/x86_64-redhat-linux/4.1.2/crtbegin.o -
L/usr/lib/gcc/x86_64-redhat-linux/4.1.2 -L/usr/lib/gcc/x86_64-redhat-linux/4.1.2 -
L/usr/lib/gcc/x86_64-redhat-linux/4.1.2/../../../../lib64 -L/lib/../lib64 -L/usr/lib/../lib64
/tmp/ccuSzfem.o -lgcc --as-needed -lgcc_s --no-as-needed -lc -lgcc --as-needed -lgcc_s --no-
as-needed /usr/lib/gcc/x86_64-redhat-linux/4.1.2/crtend.o /usr/lib/gcc/x86_64-redhat-
linux/4.1.2/../../../../lib64/crtn.o
```

```
43-sisd00% gcc -v kopiatu.c -o kopiatu
Reading specs from /usr/local/lib/gcc-lib/sparc-sun-solaris2.6/2.8.1/specs
gcc version 2.8.1
 /usr/local/lib/gcc-lib/sparc-sun-solaris2.6/2.8.1/cpp -lang-c -v -undef -
D__GNUC__=2 -D__GNUC_MINOR__=8 -Dsparc -Dsun -Dunix -D__svr4__ -D__SVR4 -
D__sparc__ -D__sun__ -D__unix__ -D__svr4__ -D__SVR4 -D__sparc -D__sun -D__unix -
Asystem(unix) -Asystem(svr4) -D__GCC_NEW_VARARGS__ -Acpu(sparc) -Amachine(sparc)
kopiatu.c /var/tmp/ccmTaq5y.i
GNU CPP version 2.8.1 (sparc)
#include "..." search starts here:
#include <...> search starts here:
 /usr/local/include
 /usr/local/sparc-sun-solaris2.6/include
 /usr/local/lib/gcc-lib/sparc-sun-solaris2.6/2.8.1/include
 /usr/include
End of search list.
 /usr/local/lib/gcc-lib/sparc-sun-solaris2.6/2.8.1/cc1 /var/tmp/ccmTaq5y.i -
quiet -dumpbase kopiatu.c -version -o /var/tmp/ccmTaq5y.s
GNU C version 2.8.1 (sparc-sun-solaris2.6) compiled by GNU C version 2.8.1.
 /usr/ccs/bin/as -V -Qy -s -o /var/tmp/ccmTaq5y1.o /var/tmp/ccmTaq5y.s
/usr/ccs/bin/as: WorkShop Compilers 5.0 Alpha 03/27/98 Build
 /usr/ccs/bin/ld -V -Y P,/usr/ccs/lib:/usr/lib -Qy -o kopiatu
/usr/local/lib/gcc-lib/sparc-sun-solaris2.6/2.8.1/crt1.o /usr/local/lib/gcc-
lib/sparc-sun-solaris2.6/2.8.1/crti.o /usr/ccs/lib/values-Xa.o
/usr/local/lib/gcc-lib/sparc-sun-solaris2.6/2.8.1/crtbegin.o -
L/usr/local/lib/gcc-lib/sparc-sun-solaris2.6/2.8.1 -L/usr/ccs/bin -L/usr/ccs/lib
-L/usr/local/lib /var/tmp/ccmTaq5y1.o -lgcc -lc -lgcc /usr/local/lib/gcc-
lib/sparc-sun-solaris2.6/2.8.1/crtend.o /usr/local/lib/gcc-lib/sparc-sun-
solaris2.6/2.8.1/crtn.o
ld: Software Generation Utilities - Solaris/ELF (3.0)
44-sisd00%
```

Otra versión con una máquina sun sparc. con SO UNIX Solaris2.

Lenguajes de Alto Nivel y Bibliotecas de código (*library*)

- Compilador
  - 1 sentencia de un Lenguaje Alto Nivel (LAN) => n instrucciones de lenguaje máquina
- Traducción + Optimización
  - + eficiente
  - Compacto
- Problema de ubicación de las variables (lugar de la memoria donde se van a colocar)
  - El compilador decide donde colocarlas (Reubicación en tiempo de Compilación)
  - El compilador supone que se colocarán en la dirección 0. Cuando el programa se cargue en memoria se REUBICAN las direcciones. (Reubicación en tiempo de carga)
- Reubicación estática
  - Tabla de reubicación

De código Fuente a código Objeto.

Programa fuente (fichero)

```
...
int Global1, x, Resul;

main()
{
  Global1 = 35;
  x = 3;
  Resul = Global1 + x*x;
  escribir(Resul);
}

void escribir(int R)
{
  printf("El resultado es %d\n", R);
}
```

Direcciones Lógicas

Compilador

Código Ensamblador/Objeto

```
0:    Global1:
4:    x:
8:    Resul:
      PROC main
12:   PUSH r30          ;; main
16:    MOV r30,r31      ;;
20:   MOV r1, 35
24:   ST   r1,  @Global1
28:   MOV r1, 3
32:   ST r1,  @x
36:   LD r2, @Global1
40:   LD r3 ,@x
44:   MULT r4, r3,r3
48:   ADD r5, r2,r4
52:   ST r5, @Resul
56:   PUSH r5
60:   CALL escribir
...

128:  RET

      PROC escribir
132:  PUSH r30   ;; escribir
...
252:  RET
```

Segunda fase

Código Ensamblador/Objeto

```
0:    Global1:
4:    x:
8:    Resul:
      PROC main
12:   PUSH r30        ;; main
16:    MOV r30,r31    ;;
20:   MOV r1, 35
24:   ST   r1,  @Global1
28:   MOV r1, 3
32:   ST r1,  @x
36:   LD r2, @Global1
40:   LD r3 ,@x
44:   MULT r4, r3,r3
48:   ADD r5, r2,r4
52:   ST r5, @Resul
56:   PUSH r5
60:   CALL escribir
...

128:  RET

      PROC escribir
132:  PUSH r30   ;; escribir
...
252:  RET
```

Link

Formatos de ejecutables:
a.out,
COFF,
ELF,
...

Formatos de ejecutables

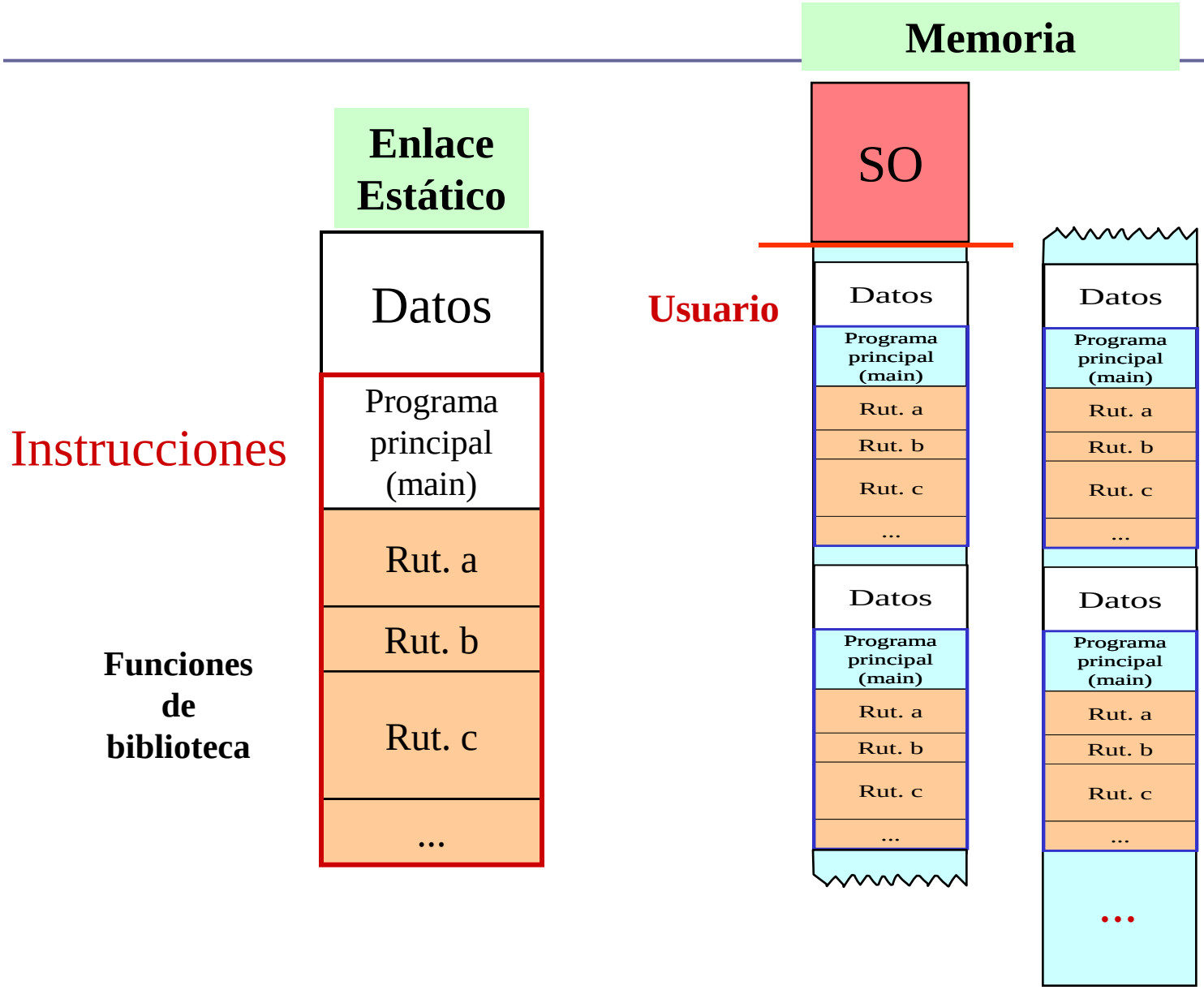
- Linux soporta diferentes formatos de ejecutables:
  - ELF (Executable and Linking Format)
    - Desarrollado por Unix System Laboratories
    - http://www.skyfree.org/linux/references/ELF_Format.pdf
    - Estándar para varios SOs.
  - a.out (Assembler OUTput Format)
    - versiones antiguas de Linux
  - Formatos de otros SO:
    - EXE de MS-DOS
    - COFF de Unix-BSD
  - Formatos definidos por el usuario
    - Se reconoce el tipo de ejecutable:
      - por el MAGIC-NUMBER (128 bytes del fichero)
      - o por la extensión

5.2 Bibliotecas y Montaje.

Reutilizar las funciones más comunes

- Bibliotecas de funciones en Código de LAN
  - Evitar reprogramar. Pero hay que recompilar.
- Bibliotecas de funciones en Código Objeto (código ya compilado)
  - Problema de referencias a funciones que no aparecen en el propio código (por parte de los programas)
  - Referencias a variables globales (por parte de esas funciones)
- El problema de las REFERENCIAS NO RESUELTAS
- Montador – Enlazador (*Linker*)
  - Resolver las referencias no resueltas (Tabla de ref. no resueltas)
  - Combinar programas con diferentes LAN
    - Código objeto con el mismo formato
    - Técnicas comunes de paso de parámetros y acceso a variables globales
- ***Reubicación en tiempo de Montaje***.

Enlace Estático



5.3 Bibliotecas de Enlace Dinámico

- Problema: Con enlace estático, cada ejecutable contiene el código de las funciones que usa. El código de una determinada función estará replicado en cada uno de los ejecutables que la utilicen.
  - Ocupando espacio en disco
  - Ocupando espacio en memoria (cuando se ejecute el programa)

- Mejora**: Enlace dinámico**.
  - Evitar replicar el código de las funciones más utilizadas.

    En memoria, sólo vamos a tener una copia del código de aquellas funciones de biblioteca que más se utilizan. Los ejecutables no tienen copia del código de las funciones de biblioteca

- Ventajas: Ahorro de espacio de almacenamento, los programas ocupan menos espacio
  - En memoria
  - En disco

Bibliotecas de Enlace Dinámico.
Mecanismo de llamada

- Problema: Si los programas no van a tener una copia del código de las funciones. ¿Cómo se puede llamar a esas funciones?

- Solución: Rutina de Enlace + Biblioteca de enlace dinámico.

  Los programas disponen de una función especial (*rutina de enlace*) encargada de llamar a esas funciones de enlace dinámico.

  Las funciones de enlace dinámico se obtendrán de las Bibliotecas de enlace dinámico, y se cargarán en memoria (pero sólo una copia).

- Funcionamiento:

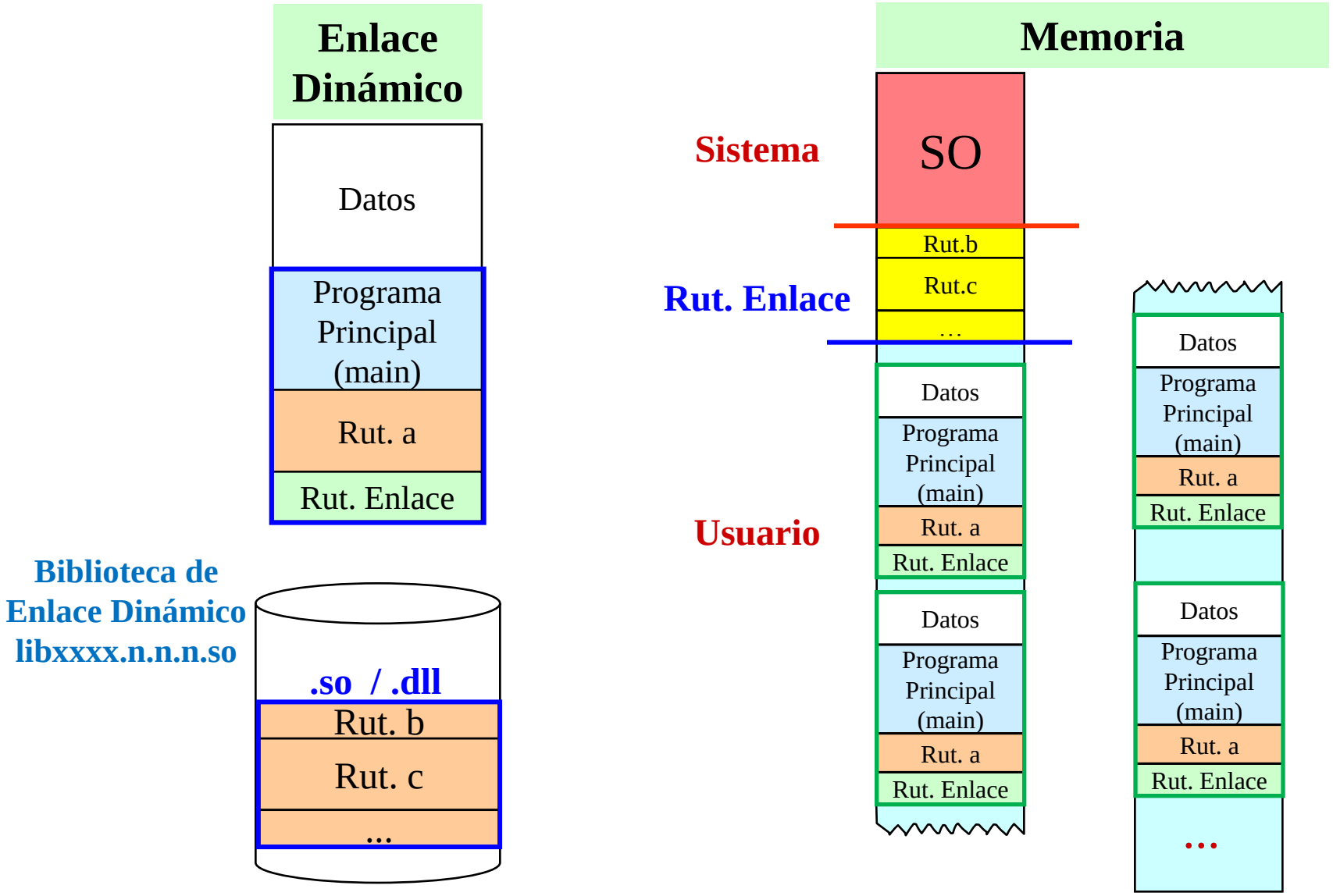
  En la fase de compilación se indica al compilador que queremos utilizar funciones de enlace dinámico. Al generar el ejecutable, no se añade la función de biblioteca, sino una función especial (*rutina de enlace*) que será la encargada de localizar la ubicación real de la función y llamarla.

  Las funciones de enlace dinámico se cargarán en una zona específica de la memoria, pero SOLO aquellas funciones que aún no estén cargadas.

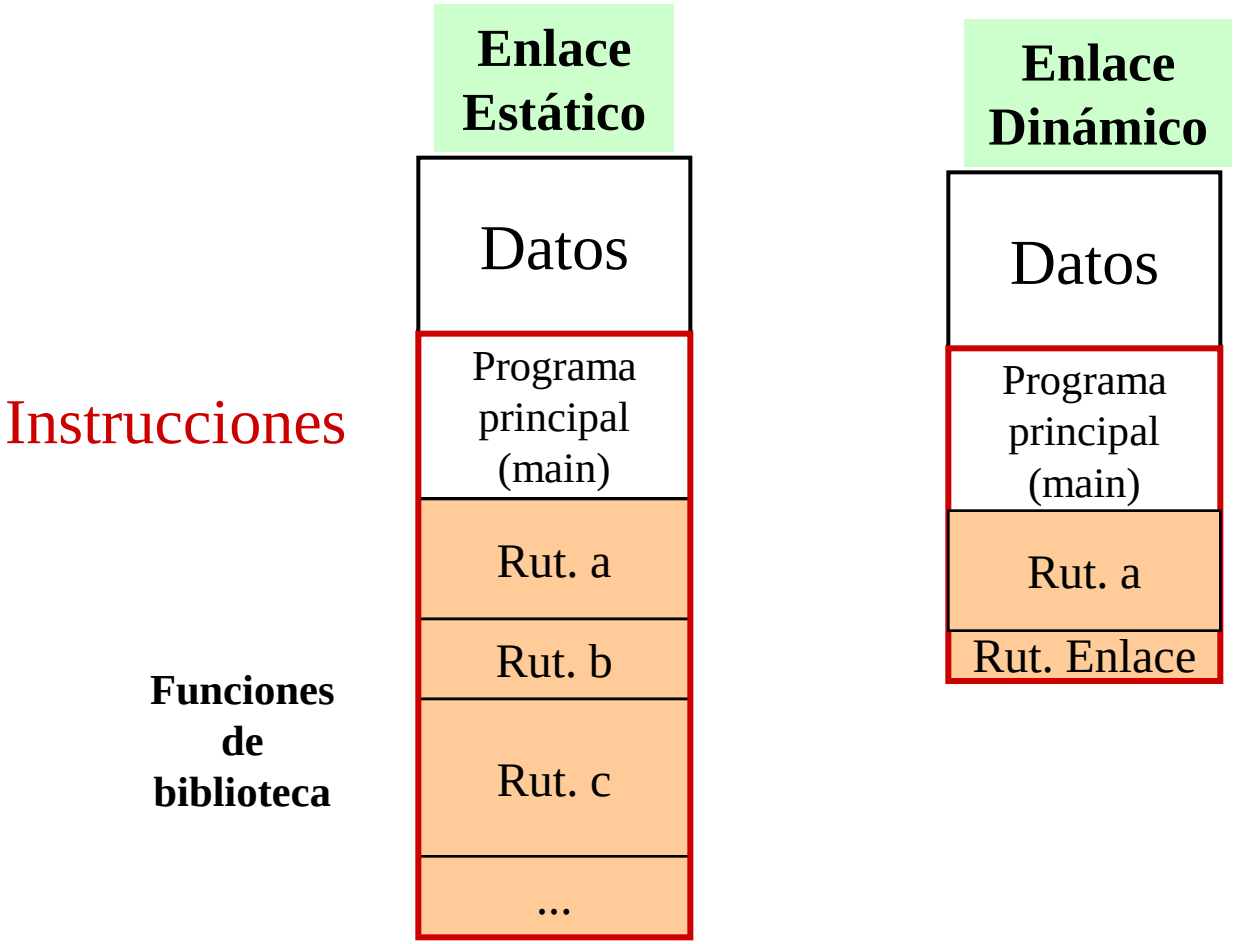
  ¿Cuando se cargan esas funciones?
  - Método1: Cuando se carga el programa
  - Método2: Cuando se llama a la función

- Esta técnica también es aplicable a las funciones del S.O. (Módulos)

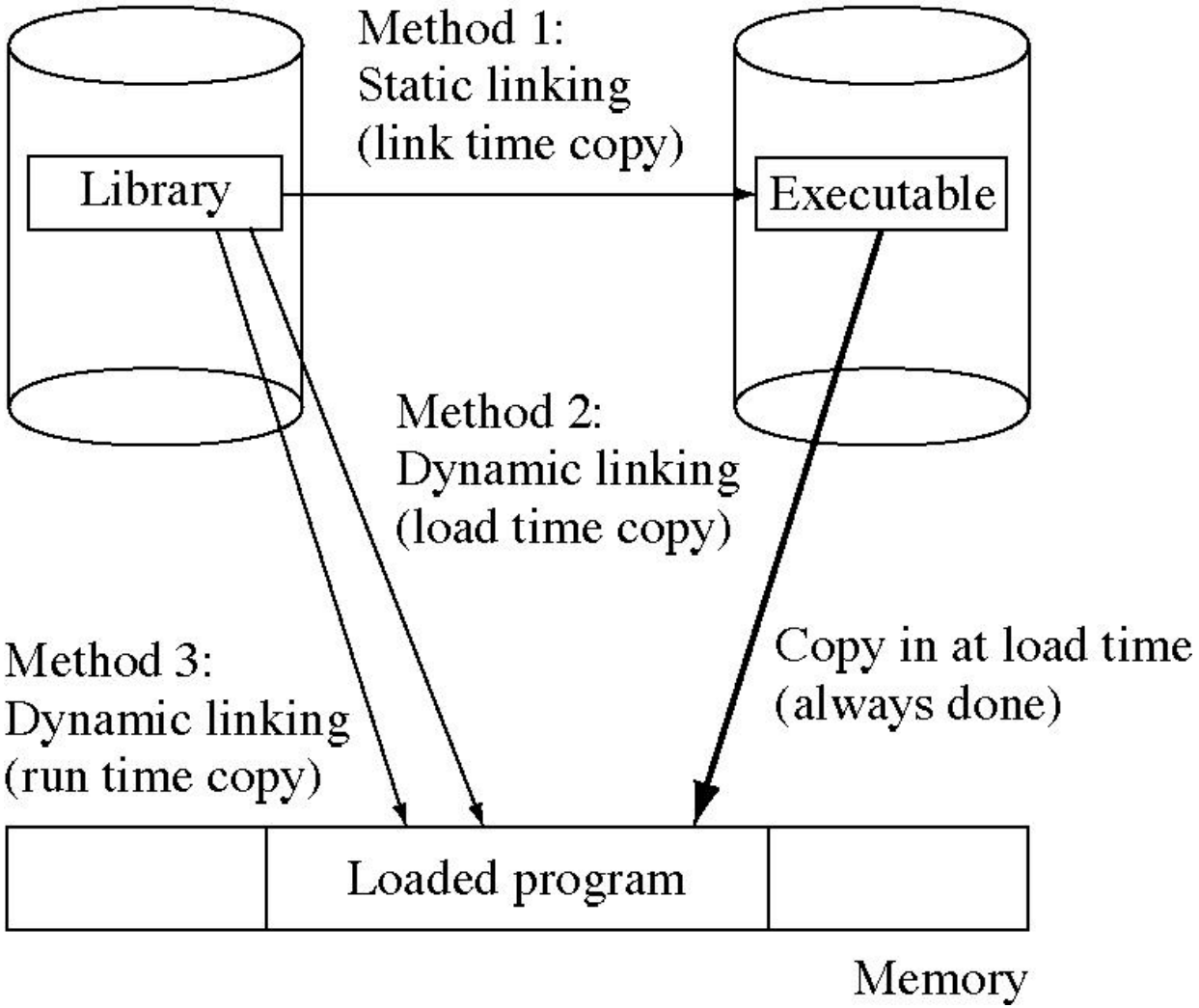
Enlace Dinámico



Enlace Estático y Dinámico



Enlace estático y dinámico



Bibliotecas de Enlace Dinámico.
Ventajas

- Menor ocupación de espacio en disco. Los ejecutables van a ser más pequeños.
- Si tengo cargado varios programas en memoria y estos utilizan funciones comunes, ocuparán menor espacio en memoria.
- Permite actualizaciones de aplicaciones sin tener que recompilar la aplicación entera (sólo las funciones).
- Permite ejecutar programas mayores que el espacio de memoria disponible (con una buena gestión de bibliotecas dinámicas).
- En el caso de aplicarlo al S.O., permite modificar el propio código del S.O. sin necesidad de parar el sistema.

Bibliotecas de Enlace Dinámico.
Inconvenientes

- Mecanismo de llamada más lento.
- Gestión de bibliotecas más complejo
- Gestión de versiones de las bibliotecas.
- Al instalar una aplicación, hay que tener en cuenta las posibles dependencias con las bibliotecas de enlace dinámico

Ejemplo de compilación con bibliotecas

Tarea 5.2. Queremos probar a crear un ejecutable tanto con bibliotecas dinámicas como estáticas. En la ayuda on-line de gcc (man) se encuentra detallado todas las opciones de gcc, pero no queda muy claro cual es la secuencia de pasos a utilizar para ambos casos. Creemos recordar una publicación de un artículo en internet con un ejemplo bastante claro de cómo crear bibliotecas estáticas y dinámicas. El título de dicho artículo era algo parecido a : "Creating a shared and static library with the gnu compiler .."

1. Localiza dicho artículo, estúdialo y realiza un esquema (secuencia de pasos) de los pasos a seguir para la creación de bibliotecas estáticas y dinámicas así como los programas de pruebas.
2. Detalla cómo se crea una biblioteca estática. Crea una biblioteca estática utilizando como ejemplo el fichero *calcula_pi.c* que te proporcionamos.
3. Detalla cómo se crea una biblioteca dinámica. Crea una biblioteca dinámica utilizando como ejemplo el mismo fichero *calcula_pi.c* que te proporcionamos.
4. Prueba ambas bibliotecas por separado con el programa *test_pi.c* y *calcula_pi.h* que te proporcionamos. Compara el tamaño de ambos ejecutables.
5. Analiza y compara el tiempo de ejecución de ambos programas con la utilidad *time*. Prueba con los valores de 10000000 y 300000000. ¿qué conclusión puedes sacar con esos resultados?
6. En otro artículo [1] nos proporciona información similar para uso de las bibliotecas dinámicas, pero además nos informan sobre unas funciones que se llaman en la carga de la biblioteca dinámica. Estudia el artículo y mira el ejemplo de uso de las funciones _init() y _fini(). Sin embargo esa forma de utilizar la funciones ha quedado un poco obsoleta. En las faqs de linux [2] se encuentra otra forma de utilizar esas funciones. Modifica el fichero *calcula_pi.c* introduciendo esas funciones y pruébalo de nuevo.

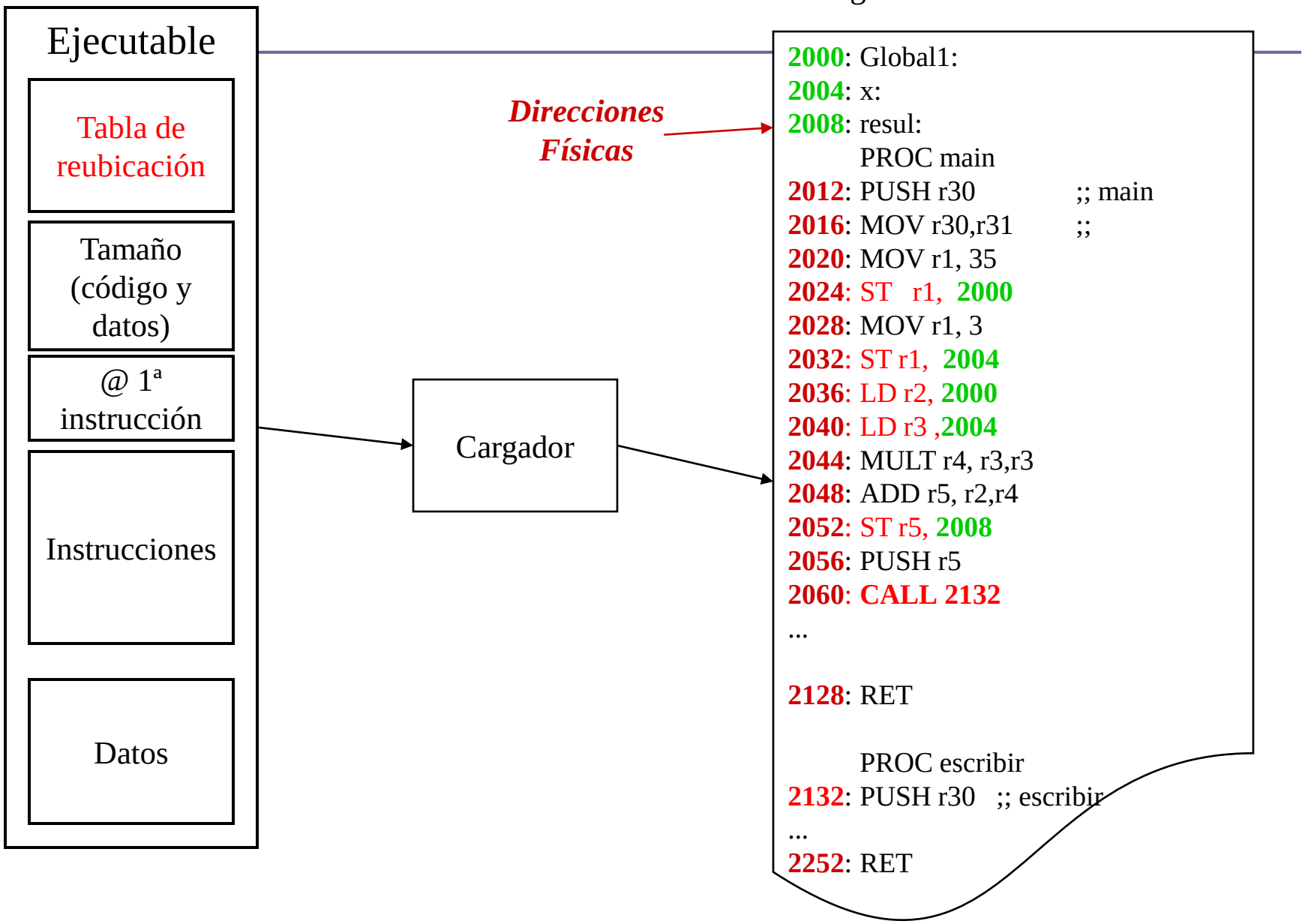
[1] http://www.ibm.com/developerworks/library/l-shobj/
[2] http://www.faqs.org/docs/Linux-HOWTO/Program-Library-HOWTO.html

5.4 Ubicación de programas en memoria
Carga de un programa en Memoria. Cargador

- Funciones del Cargador:
  1. Lee el fichero ejecutable
  2. Lo ubica en un espacio libre de la memoria:
     - Busca un espacio consecutivo libre en memoria
     - Copia el código del programa
     - Copia los datos Inicializados
     - Reserva espacio para la pila y las memoria dinámica
  3. Comienza con la primera instrucción del programa:
     - Inicializa los registros de la CPU (PC, PSW,SP,...).

- El SO tiene que llevar contabilidad/gestión del uso de la memoria:
  - Conocer el espacio libre
  - Zona de memoria que ocupa un programa
  - Debe liberar esa memoria cuando el programa termine

Fase de Carga



Carga de Programas mediante Reubicación estática. Inconvenientes

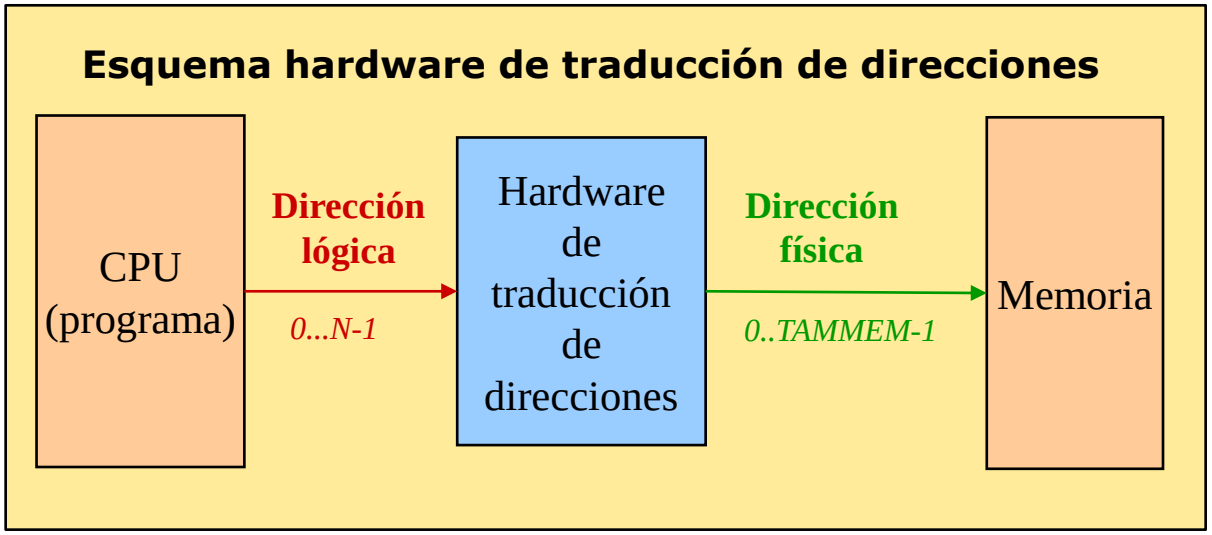
- La reubicación en tiempo de carga plantea algunos problemas.
  - Tiempo invertido en el proceso de reubicación ralentiza la fase de carga.
  - El programa una vez reubicado NO SE PUEDE MOVER. (Puede generar fragmentación de la memoria)

- Podríamos solventar estos problemas si la reubicación se realizara en el momento de acceder a memoria, en cada referencia.
  - **REUBICACIÓN DINÁMICA**

5.4 Direccionamiento Físico y Virtual

- La memoria virtual es una técnica de gestión de memoria que permite ofrecer a las aplicaciones un espacio de direcciones mayor que el espacio de direcciones de la memoria real disponible.
- Esta técnica va a permitir ejecutar aplicaciones que no estén completamente cargadas en memoria. Como consecuencia se podrán ejecutar aplicaciones que requieran un espacio de memoria mayor que el espacio de memoria real disponible. También va a permitir mantener un mayor número de programas en la memoria principal, ya que no es necesario que estén totalmente cargados en la memoria.
  - Por ejemplo, un programa puede tener un espacio de direcciones virtual de 4GB y estar ejecutándose sobre una máquina que dispone de 512MB de memoria REAL.
- La idea se basa en la utilización de almacenamiento secundario (disco) en la que se encuentra cargado la totalidad del programa. En un instante dado, en la memoria principal se encontrará únicamente aquellas partes del programa que sean realmente necesarias para la ejecución.
- Toda la gestión la realiza el sistema operativo con apoyo de hardware específico y es totalmente transparente a las aplicaciones.
- Los compiladores van a crear los ejecutables, suponiendo que disponen de un espacio de direcciones virtual. Los compiladores van a suponer que los programas se ubican a partir de la dirección (virtual) 0. Consecuentemente, durante la ejecución de un programa se van a generar referencias a memoria (direcciones virtuales) del rango [0..TAM-1] donde TAM es el tamaño del programa (Instrucciones + Datos).
- Sin embargo, cuando un programa se ejecute, este no va a estar ubicado en la dirección 0, y por tanto las direcciones que se generan durante la ejecución (direcciones virtuales), no van a ser las direcciones reales donde se encuentra las instrucciones y datos del programa(direcciones físicas).
- Para poder realizar un acceso correcto a memoria, se necesita por tanto convertir las direcciones virtuales a direcciones físicas. Esta tarea se realizará con la ayuda de hardware específico (hardware de traducción de direcciones).
- En el siguiente apartado se presentarán algunos de métodos de implementación de mecanismos de memoria virtual.

Esquema hardware de traducción de direcciones



5.5 Soporte para memoria virtual. Características deseables de la ubicación de los programas en memoria

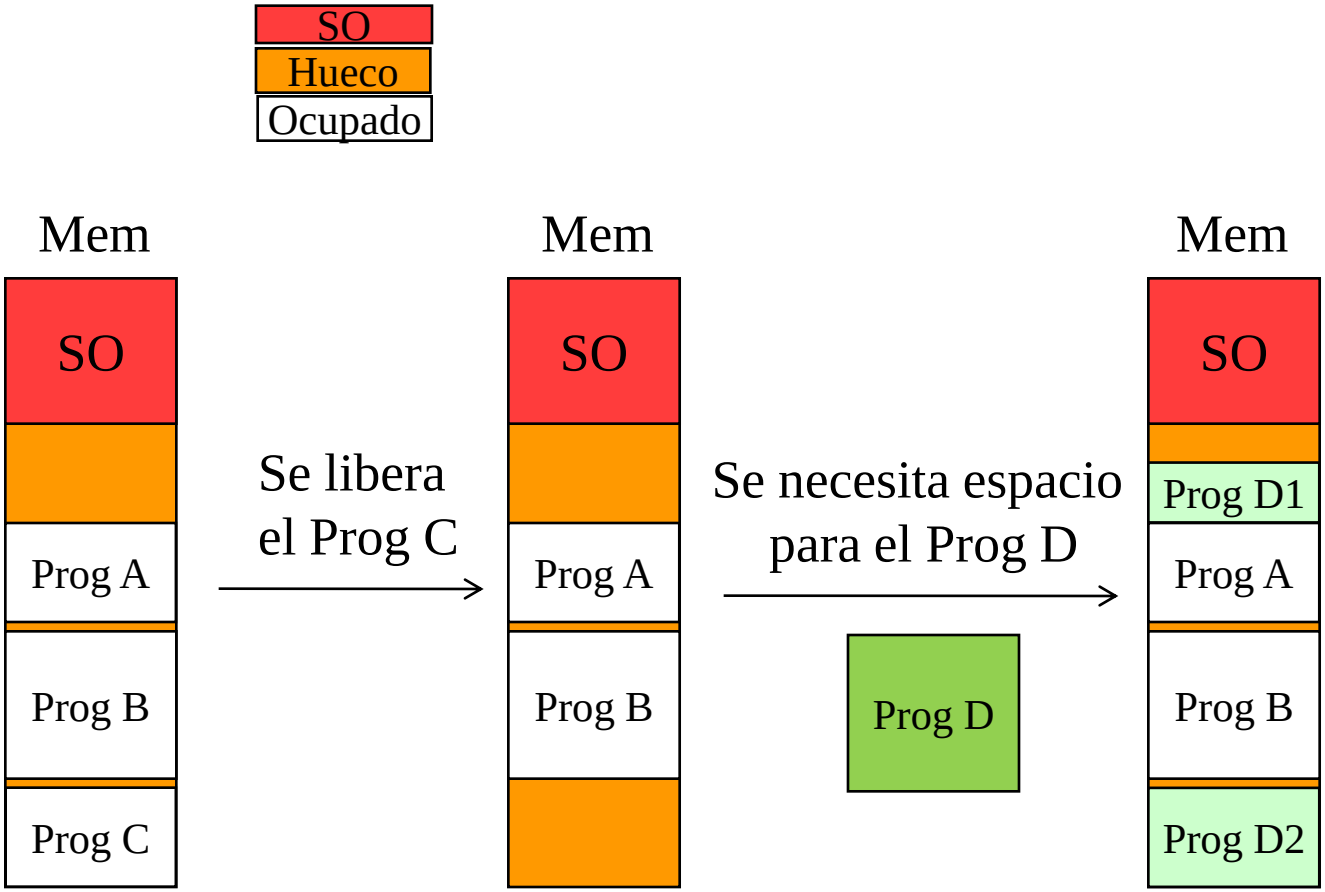
En este apartado se presentarán algunos de métodos de implementación de mecanismos de memoria virtual. Comenzaremos describiendo tres de las características más interesantes respecto a la ubicación de los programas en memoria que sería deseable que proporcionase un sistema de gestión de memoria virtual.

MAS DE UN PROGRAMA en Memoria: En un sistema monoprogramado, sería deseable que el sistema de gestión de memoria permita tener cargado en la memoria principal más de un programa. De esta manera podríamos reducir o eliminar el tiempo de carga de algunos programas, en concreto aquellos programas que cuando van a ser ejecutados ya se encontrarán cargados en memoria. En temas posteriores, veremos que esta característica que ahora nos parece "interesante" en un sistema monoprogramado, se convierte en una necesidad cuando hablemos de un sistema multiprogramado.

El programa no se encuentre entero. Otra de las características que vamos a intentar conseguir es la de posibilitar que un programa que se esté ejecutando no tenga que estar completamente cargado en la memoria principal. Con esta característica vamos a poder ejecutar programas que necesiten mayor capacidad de memoria que la memoria física disponible en el sistema. Esta característica va a ser muy importante en un sistema multiprogramado al permitir tener cargados un mayor número de programas en memoria al no tener que estar completamente cargados en memoria.

(NO CONTIGÜO) La tercera característica que nos va a interesar en un sistema de gestión de memoria es la de posibilitar que los programas estén almacenados en posiciones de memoria NO CONTIGÜAS. Esto nos va a ayudar a gestionar más eficientemente la memoria. Si se permite más de un programa cargado en memoria, cuando se decide eliminar de la memoria uno de los programas cargados, la porción de memoria que ocupaba queda libre, generando un "hueco" de memoria libre. Si se repite esa operación con diferentes programas se van a ir generando una serie de huecos de memoria libre. Si cada uno de esos "huecos" no son lo suficientemente grades, puede ocurrir que posteriormente, otro programa no pueda cargarse en ninguna de esas posiciones de memoria, porque no hay suficiente espacio libre contiguo como para poder ubicarse (totalmente o parcialmente). Sin embargo la suma total de memoria libre (todos los huecos) si permitiría su carga. A este fenómeno se le denomina fragmentación de la memoria. La característica de permitir que un programa se ubique de forma no contigua va a reducir la aparición de fragmentación de la memoria, al permitir ubicar un programa (o parte del mismo) si se dispone de espacio suficiente anque no esté contiguo.

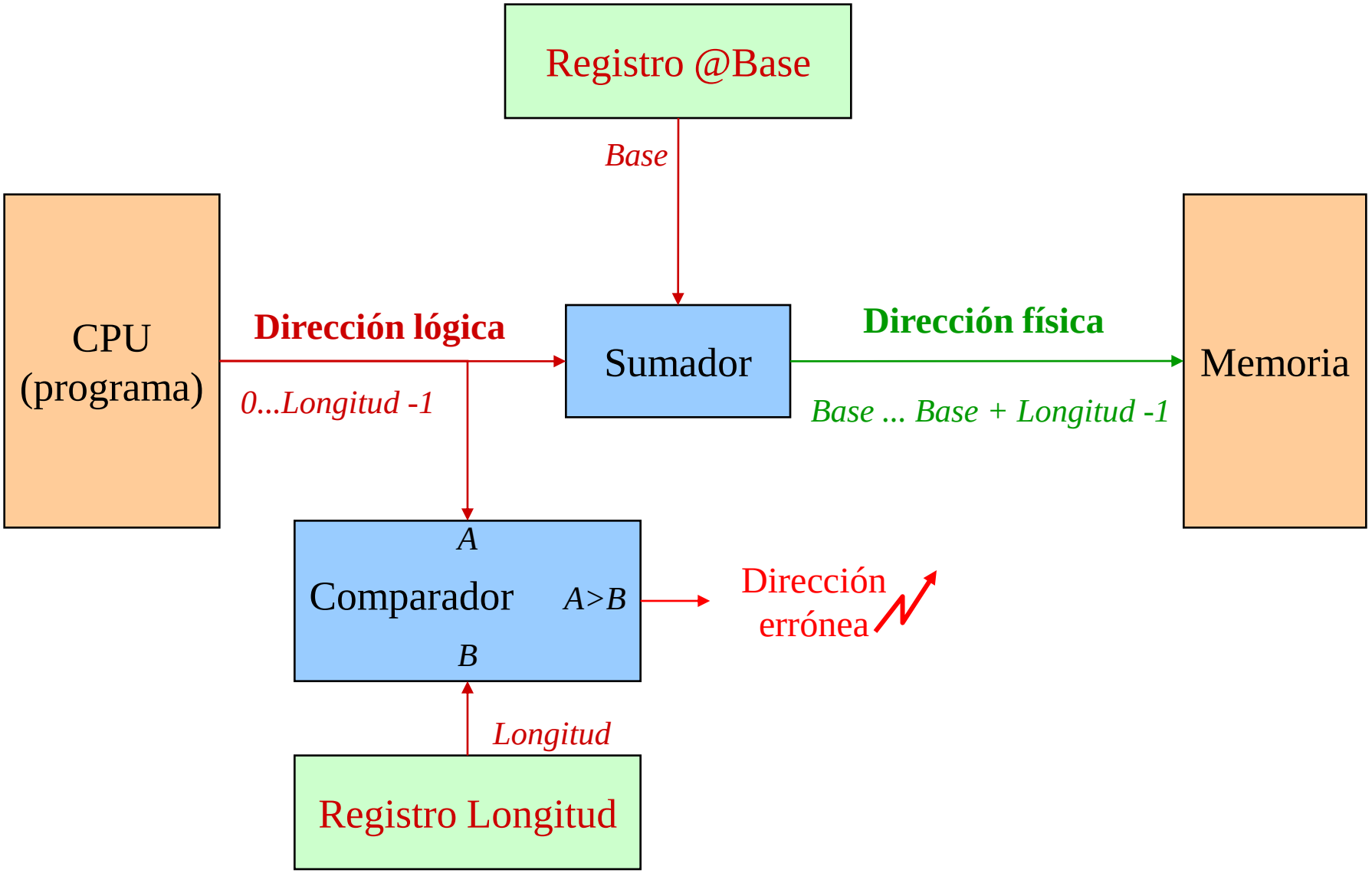
Fragmentación de la memoria



Más de un programa en memoria Entero y Contiguo

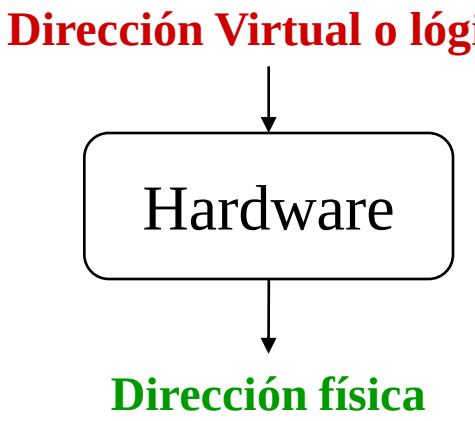
- Hasta ahora 1 programa ENTERO y CONTIGUO.
- Mayor capacidad de almacenamiento en memoria.
  - Tener cargados los programas que más se utilizan.
- Todos no entran
  - Espacio de memoria que comparten los programas más frecuentemente utilizados
  - Gestión del Espacio
  - Políticas de reemplazo
- Por ahora ejecución secuencial (hasta que uno no termina no puede empezar otro)
- Proteger a los programas de otros de accesos *involuntarios*. No olvidar que todos ellos están el la zona de USUARIO (NO PRIVILEGIADA)
- Hardware específico para la protección.
- ***Reubicación DINÁMICA*** (en tiempo de ejecución).

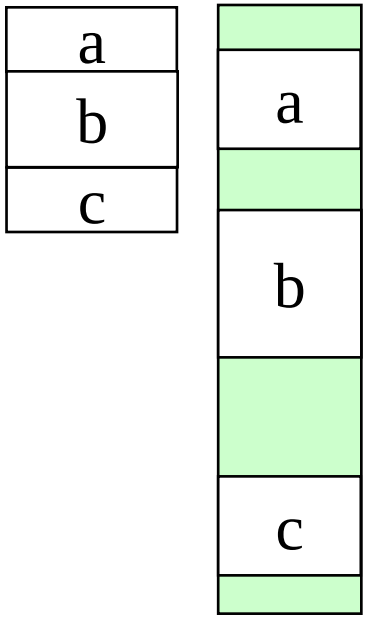
Esquema hardware de protección de memoria

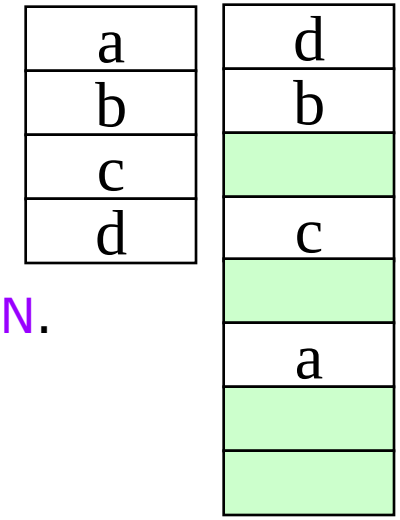


Un Programa (o varios) NO CONTIGUO y NO ENTERO

- **NO CONTIGUO**

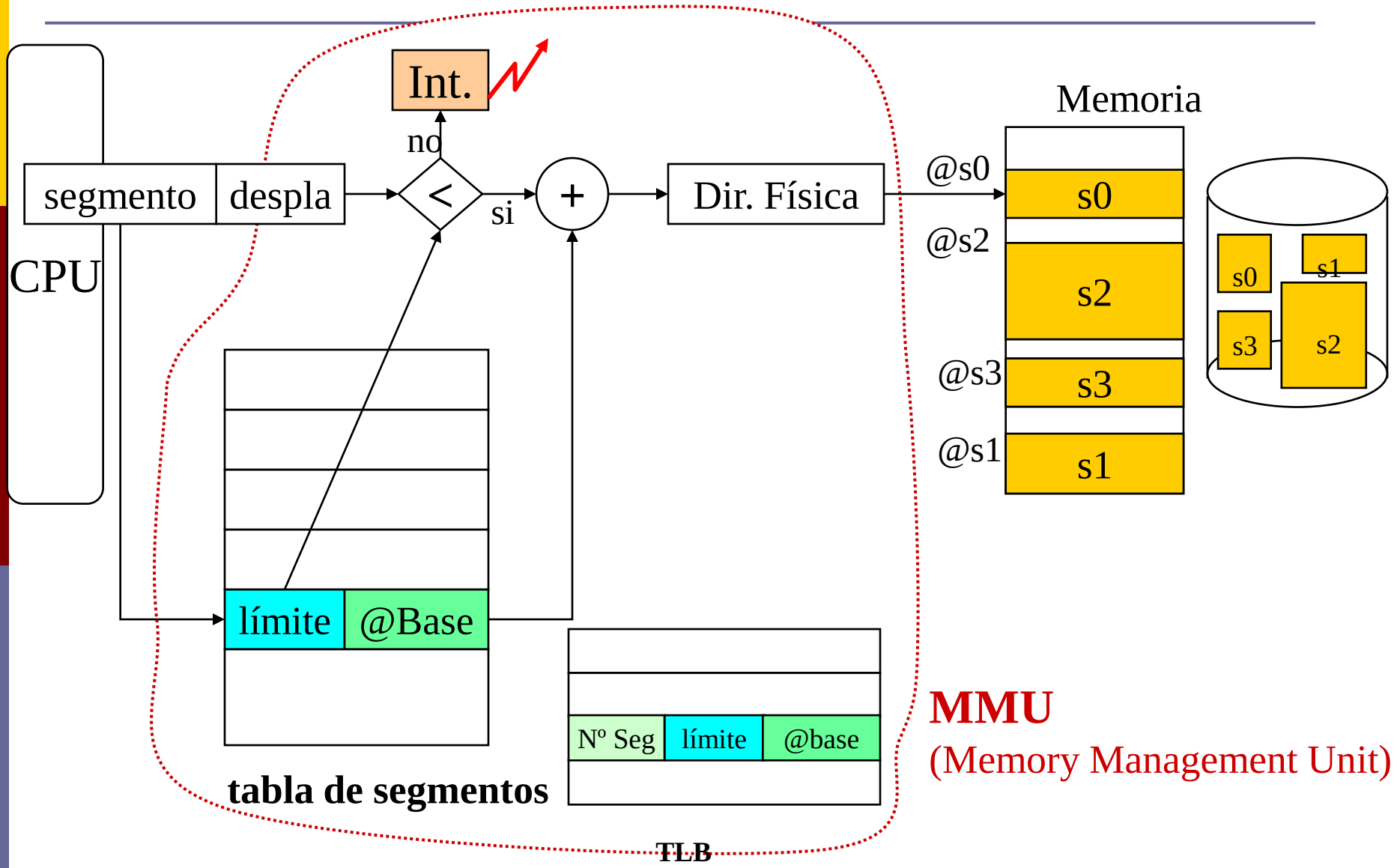






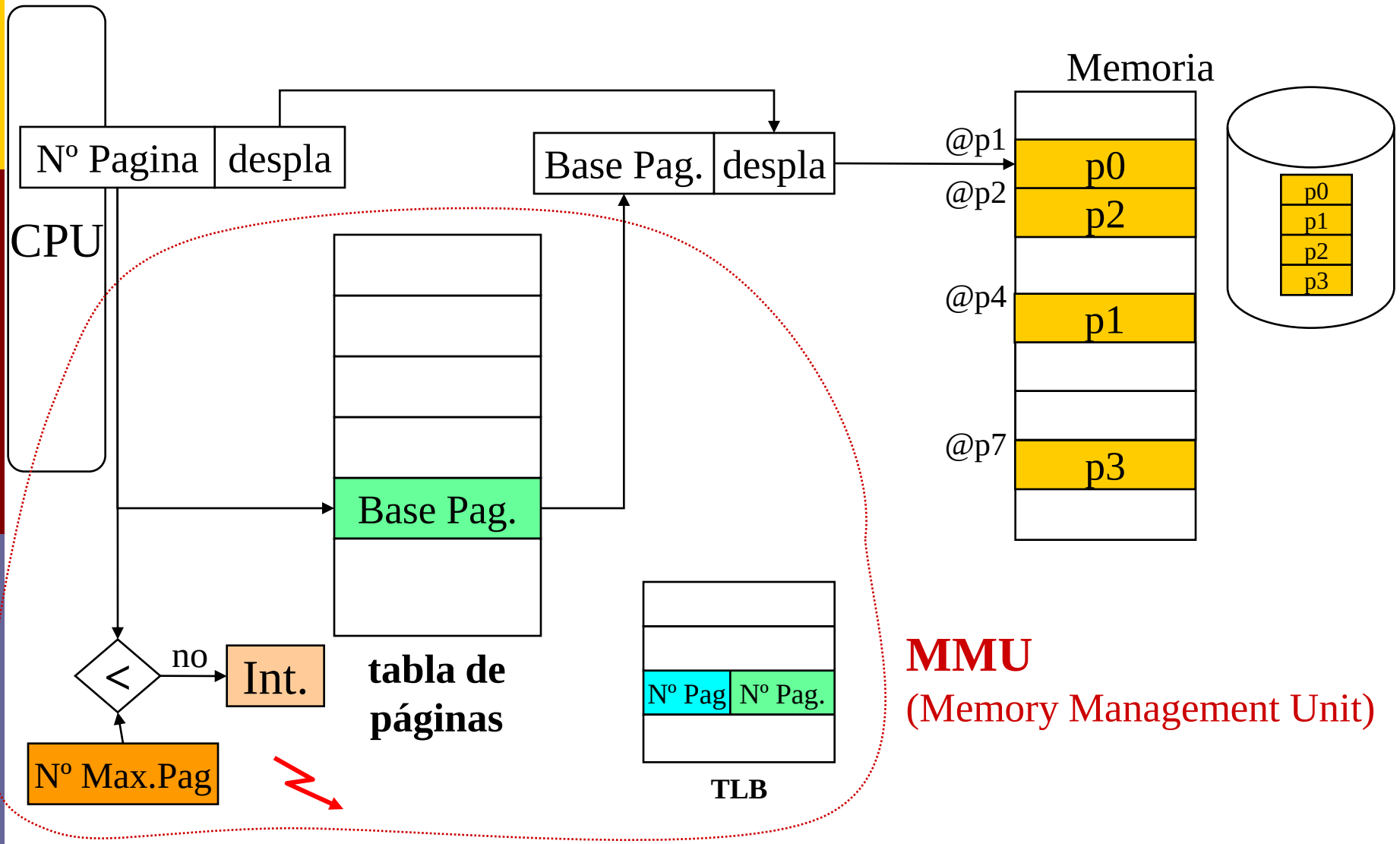
- Dividir el programa en trozos
  - Trozos de diferente tamaño SEGMENTACIÓN.
    Lo hace el Compilador / Sistema Operativo
  - Trozos de igual tamaño PAGINACIÓN
    Por Hardware / Sistema Operativo

Esquema hardware de Segmentación



MMU
(Memory Management Unit)

Esquema hardware de Paginación



Programa NO CONTIGUO y NO ENTERO

- **NO ENTERO**

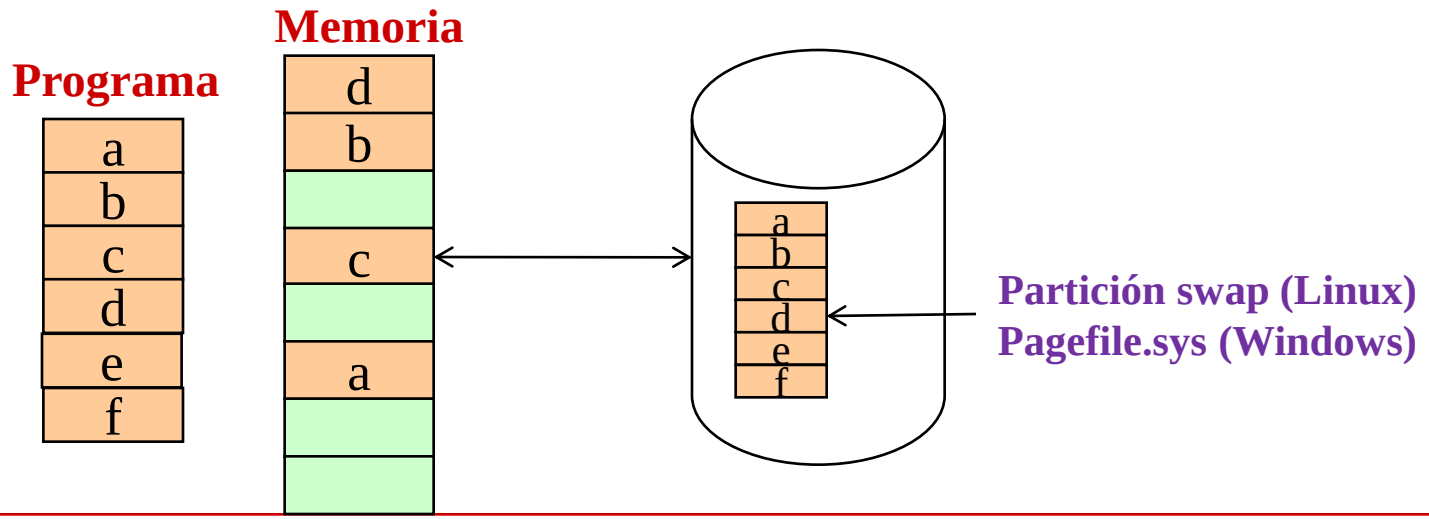
  No introducir TODO el programa en la memoria

  (Solo una pequeña porción, un segmento o una página)

  - Si lo que buscamos no está en la memoria principal

    Fallo de acceso (Fallo de página).

    - Rutina de atención al fallo (traer el código necesitado de algún otro dispositivo de almacenamiento)
    - Políticas de Reemplazo - TRASHING o SOBREPAGINACIÓN
    - Carga con anticipación / Carga por demanda



Ubicación de programas en memoria.
Mas de un programa en memoria

- Más de un programa en memoria
  - NO CONTIGUO
  - NO ENTERO
- Reubicación Estática versus Reubicación Dinámica

5.6 Llamadas al sistema relacionadas con la carga de programas

int **execlp**(char * path, char *arg0, char *arg1, ..., char *argn, NULL);

int **execv**(const char *path, char *const argv[]);

…

// Carga un nuevo programa reemplazando al que lo llama

```
#include <stdlib.h>

int system (const char * orden);  /// Equiv a    /bin/sh –c orden
```

Tarea 5.4 Pruebas con las funciones de carga de programas:
execlp
system

Utilidades interesantes

- gcc, as, ld, collect2
- ar : crear bibliotecas
- ldconfig: configure dynamic linker run time bindings
- Análisis de ejecutables/objetos/bibliotecas
  - nm : list symbols from object files
  - objdump : display information from object files.
  - file : determine file type

Enlaces de Interés:

- http://www.faqs.org/docs/Linux-HOWTO/Program-Library-HOWTO.html
- http://www.ibm.com/developerworks/library/l-shobj/