Sistemas operativos.
Memoria

Contenidos I

Memoria

- La memoria se puede definir como los circuitos que permiten almacenar y recuperar la información
- La unidad de almacenamiento es el *bit* (**bi**nary elemen**t**)aunque normalmente la consideramos estructurada en *bytes* (8 bits)
- Aunque el byte es la unidad de direccionamiento, solemos hablar de *palabras*. *palabra* se refiere a la longitud de los registros del microprocesador.
  - Asi hablamos de microprocesadores de 16 bits, de 32 bits ...
- Por razones históricas a veces se denomina *palabra* a 16 bits (2 bytes) y *doble palabra* a 32 bits

Acceso a la memoria

- Segun la manera de acceder a las las celdas de información de la memoria las podemos clasificar de
  - **memorias asociativas** Direccionables por contenidos: cuando se le pide la informacin almacenada en una dirección, la memoria asociativa est organizada de forma tal que puede encontrarla sencillamente inspeccionando los bits de la direccin y con una búsqueda asociativa (muy similar a una búsqueda en una tabla hash). Utilizada principalmente en las *caches*
  - **memorias convencionales** Cada celda es direccionable por un número al que llamaremos *dirección*

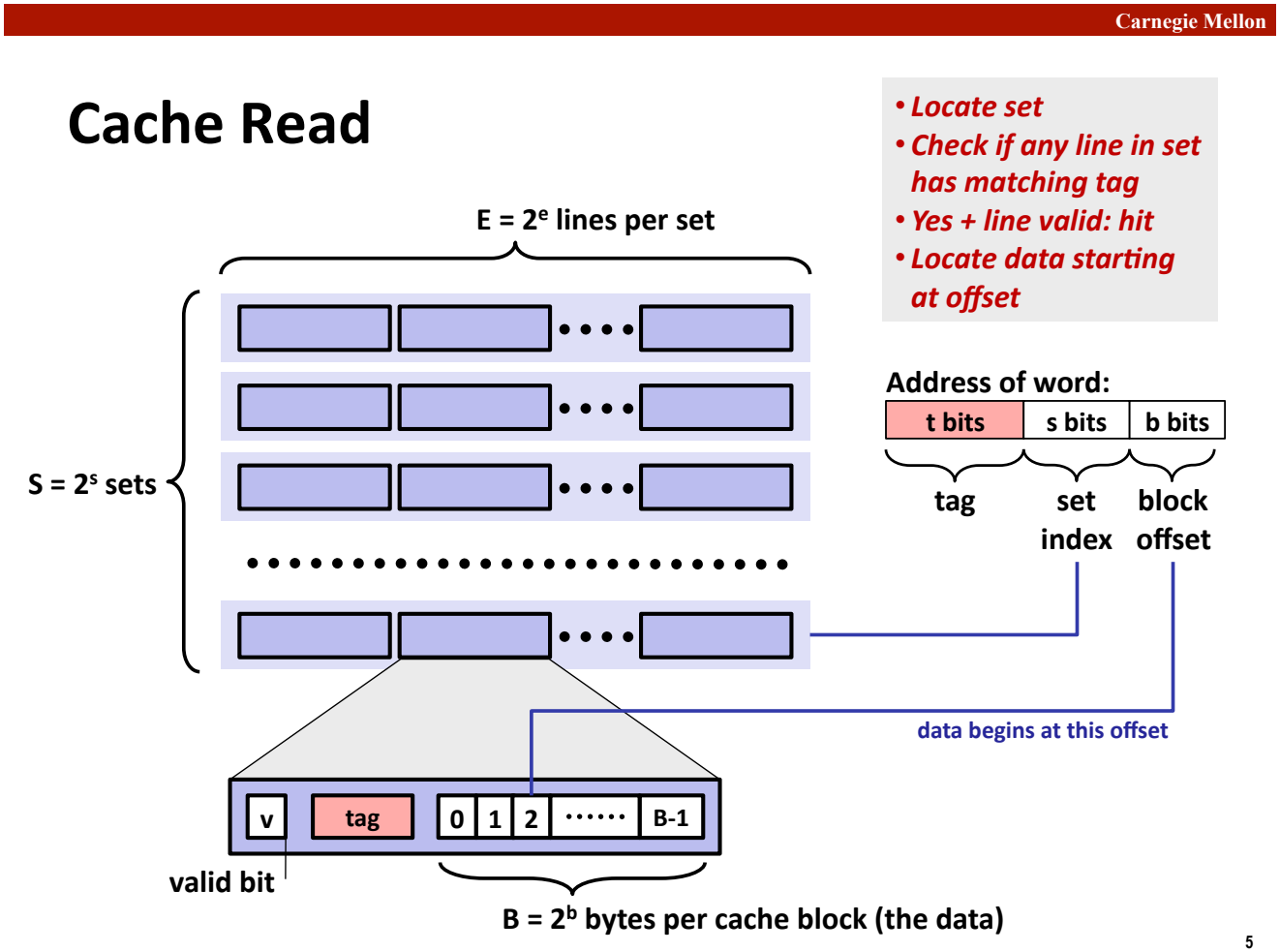
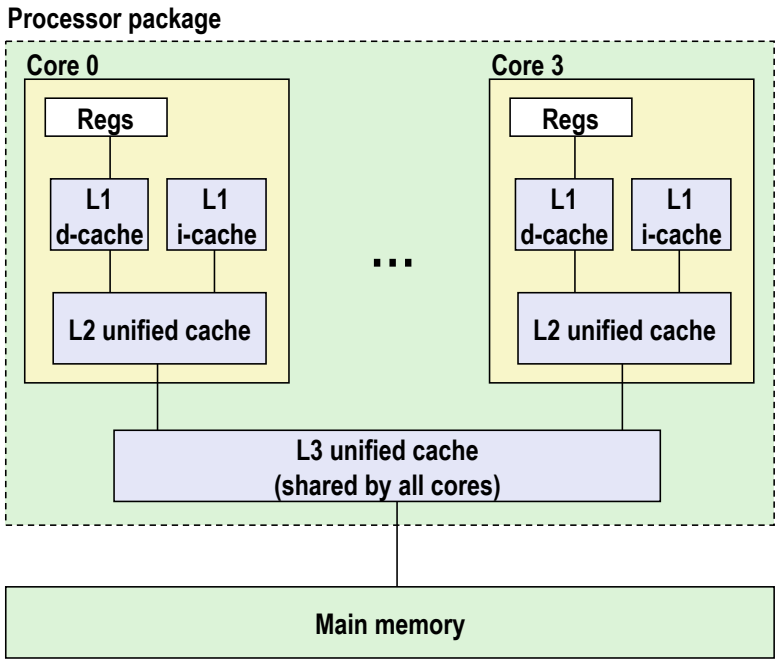


Figure: From R.E. Bryant et al. Computer Systems: A Programmer's Perspective (2nd edition), Pearson 2014

Carnegie Mellon

Intel Core i7 Cache Hierarchy



L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MB, 16-way,
Access: 30-40 cycles

Block size: 64 bytes for all caches.

17

Figure: From R.E. Bryant et al. Computer Systems: A Programmer's Perspective (2nd edition), Pearson 2014

Jerarquía de la Memoria

- El tiempo de acceso es el tiempo necesario para realizar una operación de lectura/escritura, es decir, el tiempo que transcurre desde el instante en que se pone la dirección en el bus de direcciones hasta que el dato ha sido almacenado en memoria o puesto a disposición de la CPU.
- Interesa que el acceso sea lo mas rápido posible, pero ademas interesa tener la mayor capacidad sin incurrir en un costo excesivo. Por tanto se usan las memorias más rápidas y caras para donde los accesos son más frecuentes
- Surge así lo que se llama *jerarquía de la memoria* y que se establece en base a los tiempo de acceso y capacidad disponible
  1. Registros del microprocesador
  2. Memoria cache
  3. Memoria principal
  4. Unidades de disco
  5. Unidades de cinta u opticas

Jerarquía de la Memoria

Carnegie Mellon

Examples of Caching in the Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 bytes words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware
L1 cache	64-bytes block	On-Chip L1	1	Hardware
L2 cache	64-bytes block	On/Off-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Disk firmware
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

63

Figure: From R.E. Bryant et al. Computer Systems: A Programmer's Perspective (2nd edition), Pearson 2014

- El sistema operativo es un *asignador de recursos* lo que implica que para cada recurso
  - El S.O. debe llevar contabilidad del recurso
  - El S.O. debe tener una política de asignación del recurso
  - El S.O. debe asignar el recurso a los procesos que lo necesiten
  - El S.O. debe recuperar el recurso cunado los procesos ya no lo necesotan

Administración de memoria

- En un sistema el S.O. debe llevar contabilidad de la memoria
  - El S.O. tiene que llevar contabilidad de la memoria disponible en el sistema: la memoria no contabilizada por el S.O. no está disponible para los procesos
  - El S.O. también tiene que llevar una contabilidad por proceso
- El S.O. asigna memoria a los procesos cuando se inicia su ejecución y cuando la solicitan
- Cuando el proceso termina el S.O. RECUPERA la memoria que tenía asignada
- Si el sistema tiene memoria virtual es el S.O. quien se encarga de gestionarla

Segmentos del espacio de direcciones virtuales de un proceso

- Código (text).
- Static Data. Para variables globales inicializadas y static C vars. Para variables globales no inicializadas (BSS).
- Heap. Asignación dinámica (malloc).
- Stack. Stack frames de las llamadas a función: argumentos y variables locales (automatic C vars), direcciones de retorno.

brk() System call

- Establece el final del data segment, que es el final del heap.
- brk() establece la dirección (argumento) y devuelve 0 si tiene éxito.
- sbrk() función C. Añade un desplazamiento (argumento) (puede ser cero) y actuliza el espacio disponible (el valor puede ser negativo implicando un decremento del espacio disponible)

malloc() C library function

- Si tiene éxito devuelve un puntero a un bloque disponible de size (argumento) bytes libres del heap, devuelve NULL si error. malloc() hace uso de las llamadas brk() y sbrk() para gestionar el heap.
- free(ptr) libera la memoria obtenida con malloc
- ver calloc() y realloc()
- Normalmente una llamada brk() se invoca para satisfacer varios malloc cuando es necesario
- malloc() también puede hacer uso de la llamada mmap() para obtener bloques muy grandes de memoria

Administración de memoria: ejemplo

- Compilar y probar el siguiente programa de ejemplo

```
/* this example comes from
http://www.enseignement.polytechnique.fr/informatique/INF583/ */

#include <stdlib.h>
#include <stdio.h>
double t[0x02000000];
void segments()
{
 static int s = 42;
 void *p = malloc(1024);
 printf("stack\t%010p\nbrk\t%010p\nheap\t%010p\n",
 "static(BSS)\t%010p\nstatic(initialized)\t%010p\ntext\t%010p\n",
  &p, sbrk(0), p, t, &s, segments);
}
int main(int argc, char *argv[])
{
  segments();
  exit(0);
}
```

Administración de memoria: ejemplo

- Output

```
stack 0x7fff12f59468
brk 0x116ed000
heap 0x116cc010
static(BSS) 0x00601060
static(initialized) 0x00601038
text 0x004005d4
```

Administración de memoria: ejemplo

- Compilar y probar el siguiente programa de ejemplo

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <limits.h>

#define TROZO  100*1024*1024
#define PUNTO  (10*1024*1024)

void accede (char * p, unsigned long long tam)
{
  unsigned long long i;
  for (i=0; i< tam; i++){
     p[i]='a';
     if ((i%PUNTO)==0)
        write (1,".",1);  /*imprime un punto cada 10 Mbytes accedidos*/
  }
}
```

Administración de memoria: ejemplo

```
main (int argc,char*argv[])
{
  char *p;
  unsigned long long total=0, cantidad=TROZO;
  unsigned long long maximo=ULLONG_MAX;

  if (argv[1]!=NULL){
        maximo=strtoull(argv[1],NULL,10);
        if (argv[2]!=NULL)
            cantidad=strtoull(argv[2],NULL,10);
  }
  while (total<maximo && (p=malloc(cantidad))!=NULL){
        total+=cantidad;
        printf ("asignados %llu (total:%llu) bytes en %p\n", cantidad,total,p);
        accede (p,cantidad);
getchar();

  }
  printf ("Total asignacion: %llu\n",total);
 sleep(10);
}
```

Output

- salida del comando pmap PID

```
3742:   ./a.out
0000000000400000      4K r-x--  /home/barreiro/teaching/teaching-so/examples_C
0000000000600000      4K r----  /home/barreiro/teaching/teaching-so/examples_C
0000000000601000      4K rw---  /home/barreiro/teaching/teaching-so/examples_C
00007ff7b22c9000 307212K rw---    [ anon ]
00007ff7c4ecc000   1588K r-x--  /lib/x86_64-linux-gnu/libc-2.13.so
00007ff7c5059000   2048K -----  /lib/x86_64-linux-gnu/libc-2.13.so
00007ff7c5259000     16K r----  /lib/x86_64-linux-gnu/libc-2.13.so
00007ff7c525d000      4K rw---  /lib/x86_64-linux-gnu/libc-2.13.so
00007ff7c525e000     24K rw---    [ anon ]
00007ff7c5264000    132K r-x--  /lib/x86_64-linux-gnu/ld-2.13.so
00007ff7c545f000     12K rw---    [ anon ]
00007ff7c5480000     16K rw---    [ anon ]
00007ff7c5484000      4K r----  /lib/x86_64-linux-gnu/ld-2.13.so
00007ff7c5485000      8K rw---  /lib/x86_64-linux-gnu/ld-2.13.so
00007fff4562e000    132K rw---    [ stack ]
00007fff456e6000      4K r-x--    [ anon ]
ffffffffff600000      4K r-x--    [ anon ]
 total           311216K
```

Output

- después de otra asignación

```
3742:   ./a.out
0000000000400000      4K r-x--  /home/barreiro/teaching/teaching-so/examples_C
0000000000600000      4K r----  /home/barreiro/teaching/teaching-so/examples_C
0000000000601000      4K rw---  /home/barreiro/teaching/teaching-so/examples_C
00007ff7abec8000 409616K rw---    [ anon ]
00007ff7c4ecc000   1588K r-x--  /lib/x86_64-linux-gnu/libc-2.13.so
00007ff7c5059000   2048K -----  /lib/x86_64-linux-gnu/libc-2.13.so
00007ff7c5259000     16K r----  /lib/x86_64-linux-gnu/libc-2.13.so
00007ff7c525d000      4K rw---  /lib/x86_64-linux-gnu/libc-2.13.so
00007ff7c525e000     24K rw---    [ anon ]
00007ff7c5264000    132K r-x--  /lib/x86_64-linux-gnu/ld-2.13.so
00007ff7c545f000     12K rw---    [ anon ]
00007ff7c5480000     16K rw---    [ anon ]
00007ff7c5484000      4K r----  /lib/x86_64-linux-gnu/ld-2.13.so
00007ff7c5485000      8K rw---  /lib/x86_64-linux-gnu/ld-2.13.so
00007fff4562e000    132K rw---    [ stack ]
00007fff456e6000      4K r-x--    [ anon ]
ffffffffff600000      4K r-x--    [ anon ]
 total           413620K
```

Fragmentación de la memoria

- Al igual que ocurre con los sistemas de ficheros, la memoria puede presentar fragmentación inter y externa
  - **fragmentación interna** Memoria que se despedicia por ser las unidades de asignación de tamaño fijo y no ser los requerimientos de los procesos múltiplos exactos de la unidad de asignación
  - **fragmentación externa** Memoria que no se puede asignar por no estar contigua, típica de los sistemas que utilizan segmentación

Intercambio

- Se denomina intercambio (*swap*) a una zona de disco que se utiliza como memoria auxiliar
- Todo proceso para ejecutarse debe estar en memoria. Utilizando una zona de disco como intercambio puede aumentarse el grado de multiprogramación.
  - Si el planificador seleción un proceso que está en el intercambio, es necesario traerlo a memoria principal, lo que aumenta el tiempo del cambio de contexto
  - Para poder intercambiar procesos que están pendientes de una operación de e/s, todas las operaciones de e/s se hacen sobre los buffers del sistema

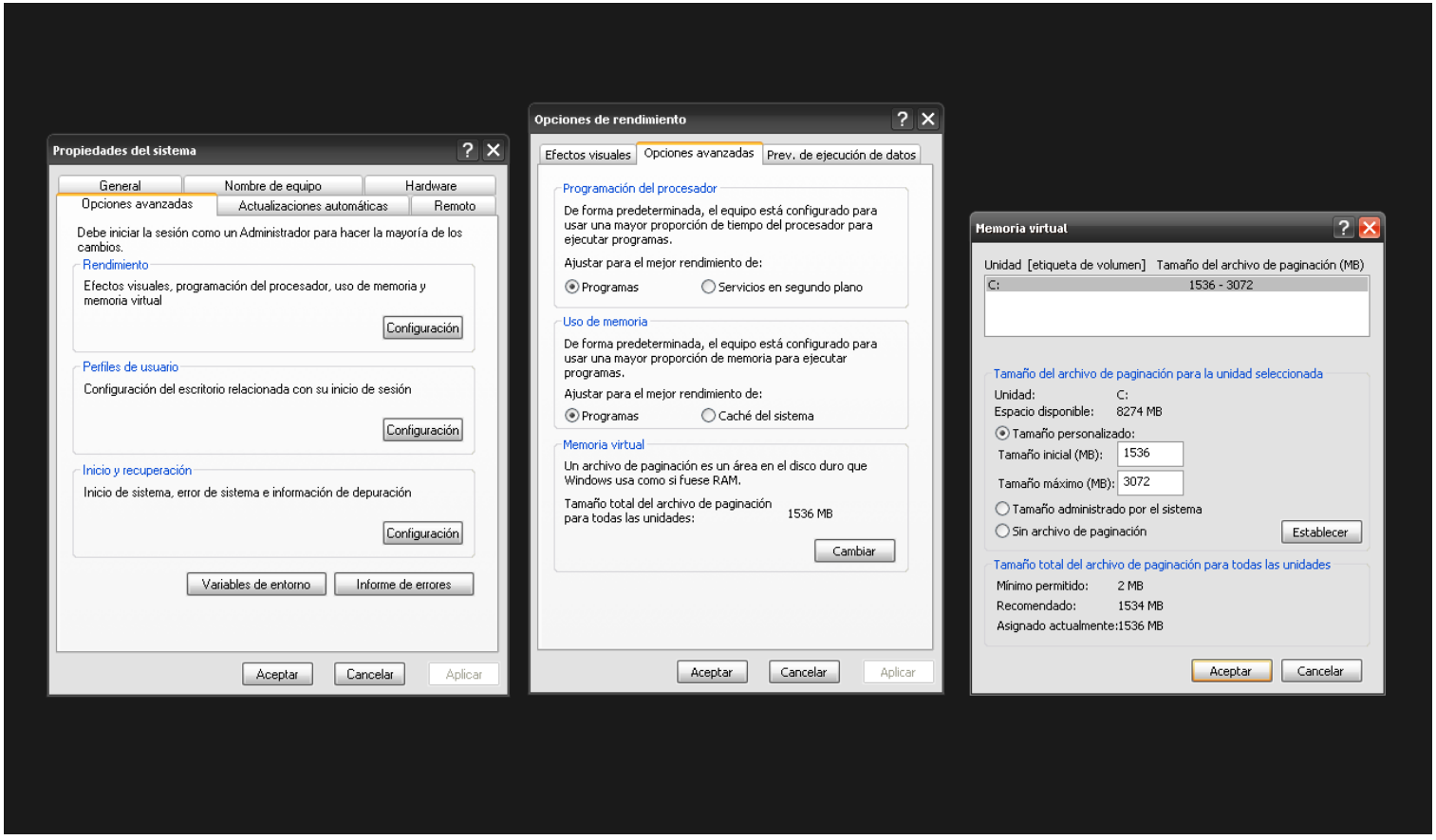
Intercambio

- En sistemas antiguos se intercambiaban procesos enteros para aumentar el grado de multiprogramacón
  - Es lo que se denomina *swapping*
- En sistemas modernos, con memoria virtual, lo que se intercambia son las páginas poco referenciadas de los procesos
  - Es lo que se denomina *paging*

Intercambio

- La zona de intercambio puede ser una partición dedicada o un fichero de disco
- Utilizar un archivo de intercambio es mas flexible, puede cambiarse facilmente su ubicación y/o su tamaño
- Un archivo de intercambio es menos eficiente, puesto que utiliza las indirecciones del sistema de archivos
- Los sistemas *windows* utilizan archivo de intercambio, y en los sistemas UNIX es más usual utilizar partición de intercambio, aunque puede utilizarse tambien archivo de intercambio

Archivo Intercambio Windows



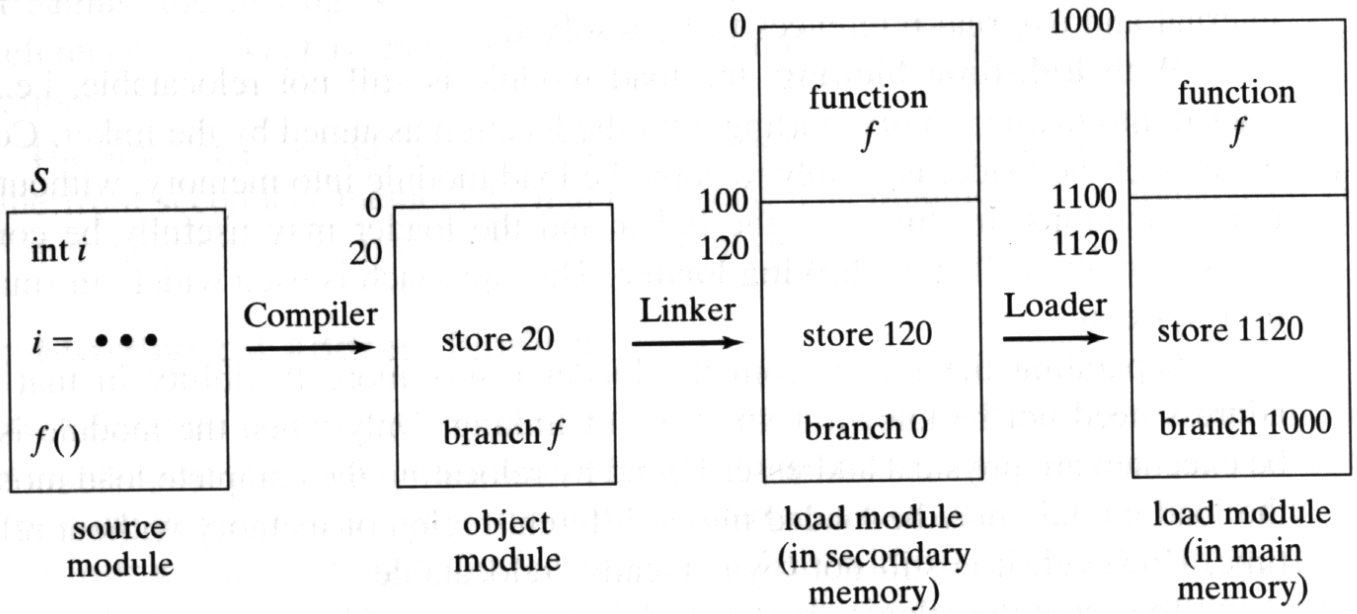
Administración de memoria: relocalización

- Partimos de código fuente $->$ (compilación) $->$ código objeto
- Varios archivos código objeto $->$ (enlazado) $->$ fichero ejecutable
- Fichero ejecutable $->$ (carga y ejecución) $->$ proceso en memoria
- Código fuente $->$ fichero ejecutable $->$ proceso en memoria
- En el fichero fuente tenemos variables, funciones, procedimientos ...
- En el proceso en memoria tenemos contenidos de direcciones de memoria, saltos a direcciones donde hay código
- ¿Dónde y cuándo se hace esa transformación?

Administración de memoria: relocalización

- Si dichas direcciones se generan en tiempo de compilación (y/o enlazado) decimos que se trata de **código absoluto** (ej: ficheros .COM de MS-DOS)
  - Habría que conocer en que dirección de memoria donde se va a ejecutar el programa en el momento de compilación/enlazado
  - El ejecutable no sería muy portable
- Si las direcciones se generan en el momento de carga (el fichero ejecutable tiene unas *referencias relativas*), se trata de **relocalización estática** (ej: ficheros EXE de MS.DOS)
  - Una vez cargado no puede moverse a otro sitio de la memoria
  - Solo puede haber intercambio si los procesos vuelven al mismo sitio de la memoria (particiones fijas)

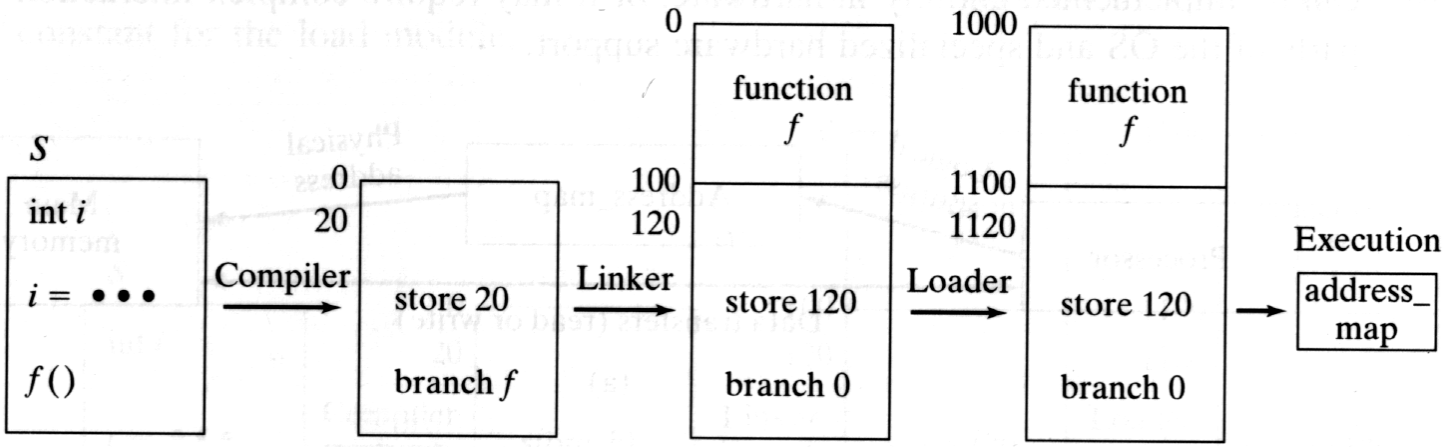
Relocalización Estática



Administración de memoria: relocalización

- Si las direcciones se generan en tiempo de ejecución (el proceso al ejecutarse maneja unas referencias que no son las direcciones de memoria reales a las que accede), se trata de **relocalización dinámica** (ej: ficheros EXE windows XP)
  - No va a haber ningún problema con el intercambio, los procesos pueden salir de memoria y volver a ella en cuanquier sitio
  - Aparece una distinción entre el *espacio virtual o lógico de direcciones* y el *espacio físico de direcciones* al que realmente se accede
  - Es NECESARIO que el *hardware* haga dicha traslación
- Todos los sistemas actuales usan relocalización dinámica
- En el caso de relocalización dinámica el enlazado puede posponerse hasta el momento de ejecución, es lo que se conoce como enlace dinámico (p.e. las DLL de windows, lib*.so de linux)

Relocalización Dinámica



Protección

- La memoria debe estar protegida de manera que
  - Un proceso no pueda acceder *directamente* a la memoria del S.O.
  - Un proceso no pueda acceder a la memoria de otros procesos
- Las maneras más elementales de implementar la protección
  - Con dos registros límite
  - Con un registro base y un registro límite

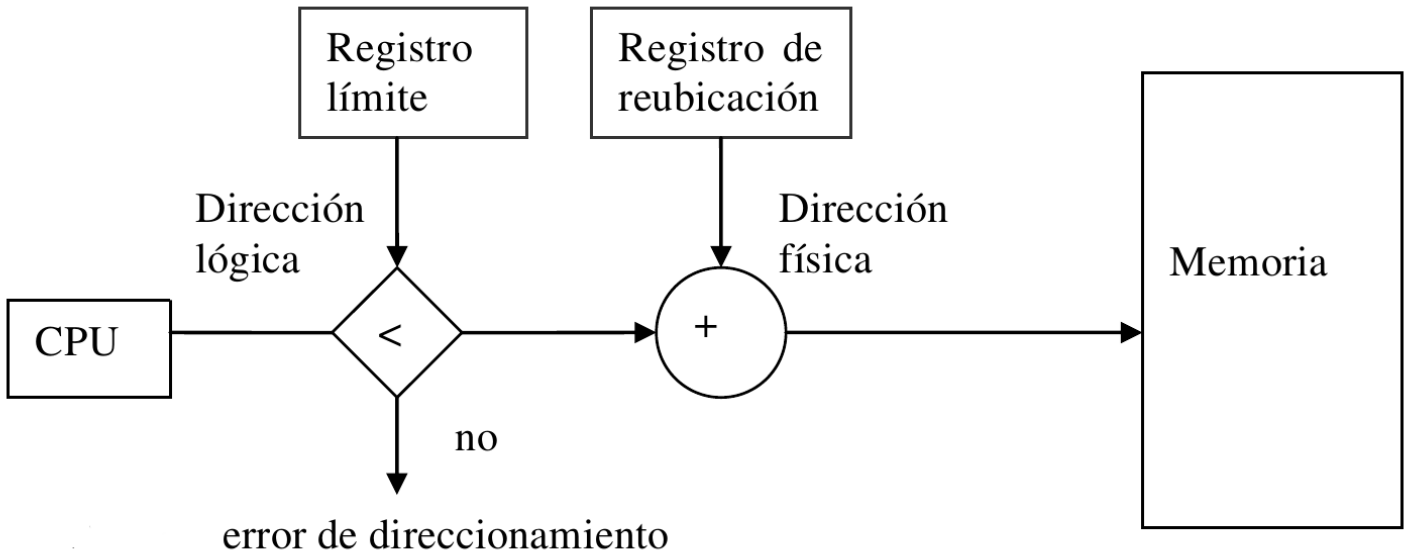
Protección

- Con dos registros límite
  - Cada ve que un proceso genera una dirección se comprueba si es mayor que el registro límite inferior y menor que el superior. De no ser así se genera una excepción
  - El hardware ha de proporcionar dichos registros límite
  - El cambio de contexto actualizará el valor de dichos registros límite
  - El cambio de valores de los registros límite ha de ser una instrucción *privilegiada*

Protección

- Con registro base y registro límite
  - Cada ve que un proceso genera una dirección se comprueba si es menor que el registro límite. En caso de serlo, la dirección física se obtiene sumando el valor del registro base. De no ser así se genera una excepción.
  - El hardware ha de proporcionar dichos registros
  - El cambio de contexto actualizará el valor de dichos registros
  - El cambio de valores de los registros límite ha de ser una instrucción *privilegiada*
  - Este esquema proporciona además una manera de relocalización dinámica

Registros base y mímite



Protección

- En sistemas actuales la protección la proporciona el sistema de direccionamiento
- Tanto la segmentación como la paginación proporcionan una protección eficaz de la memoria
- Sigue siendo necesario, obviamente, que haya al menos dos modos de ejecución, modo usuario y modo sistema (kernel, supervisor . . .)

Esquemas simples

Sistemas no multiprogramados

Sistemas Multiprogramados

Sistemas simples: sistemas no multiprogramados

- El sistema mas simple de administración de memoria es *carecer de él*: válido en sistemas dedicados
- En un sistema operativo no multiprogramado solo dos zonas de memoria: el S.O y el proceso de usuario
- Tipicamenmte el S.O en las posiciones bajas de memoria y el resto para el proceso de usuario: concepto de monitor simple (IBSYS pra IBM 7094)
- Primeros ordenadores personales: S.O. en direcciones altas de la memoria (en ROM) y resto para procesos de usuario
- S.O. en direcciones bajas de la memoria y partes de S.O. en direcciones altas de memoria: Primeras versiones MS-DOS

Esquemas simples

Sistemas no multiprogramados

Sistemas Multiprogramados

Administración de memoria: Esquemas simples

- En el caso de sistemas operativos multiprogramados, el método mas simple es dividir la memoria en varias zonas
- Un proceso en cada zona
- Dos enfoques
  - zonas de tamaño fijo: El número de procesos en memoria tambien es fijo
  - zonas de tamaño variable: El número y el tamaño de las zonas puede variar

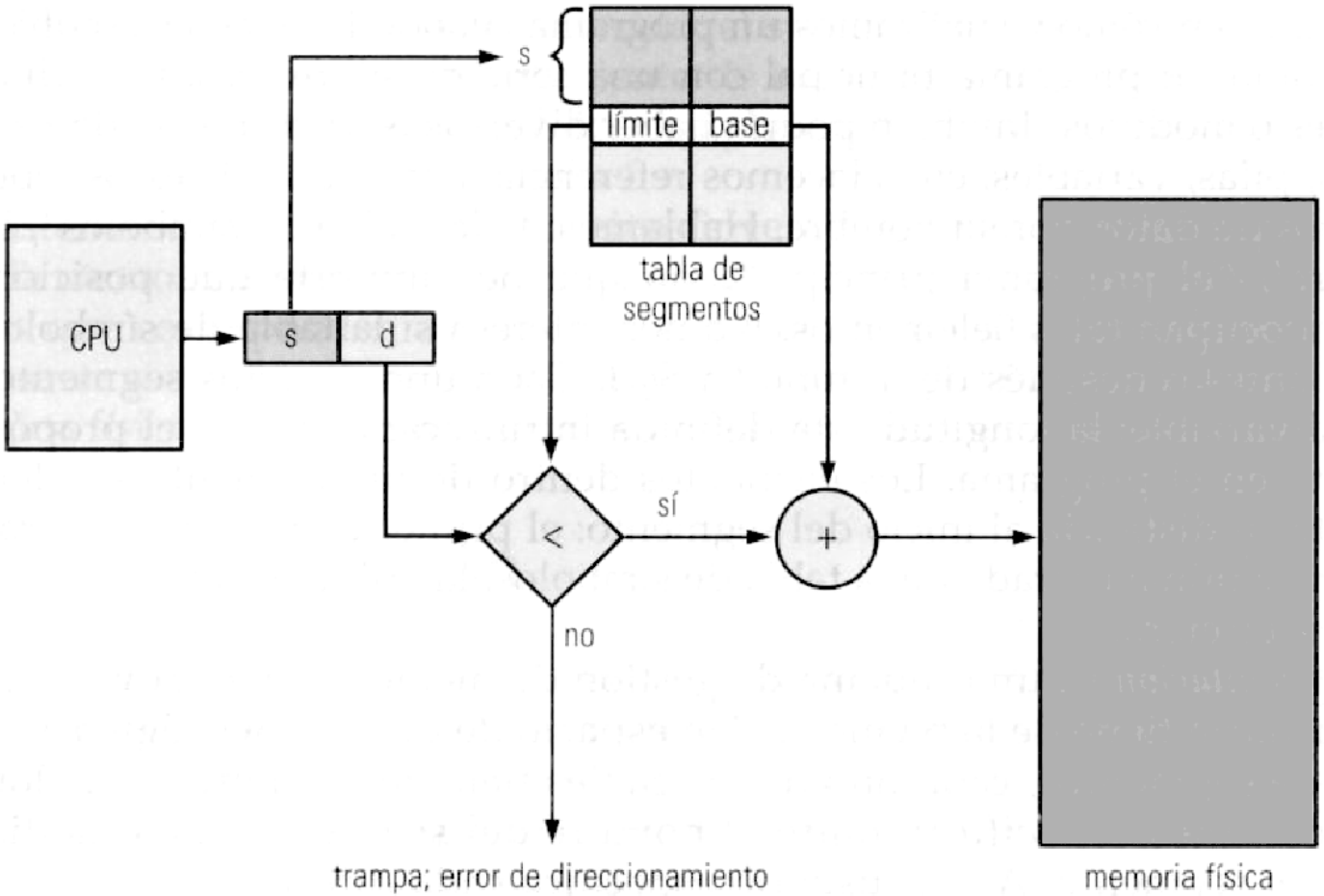
Administración de memoria: Esquemas simples

- Zonas de tamaño fijo
  - Presentaba *fragmentación externa* e *interna*
  - Usado en IBM OS/360 MFT (Multiprogramming Fixed number of Tasks)
- Zonas de tamaño variable
  - *fragmentación interna* despreciable. Si presentaba *fragmentación externa*
  - Requiere *compactaciones*
  - Usado en IBM OS/360 MVT (Multiprogramming Variable number of Tasks)

Administración de memoria: Segmentación

- El proceso considera su espacio de direcciones compuesto de entidades de tamaño variable llamadas *segmentos*
- La dirección lógica se compone de segmento y desplazamiento
- Con el número de segmento se obtiene una entrada en una tabla de segmentos, en donde hay una dirección base y un límite.
  - Si el desplazamiento es mayor que el límite se produce un error de direccionamiento.
  - La dirección de memoria física a la que se accede se obtiene sumando el desplazamiento a la base

Administración de memoria: Segmentación



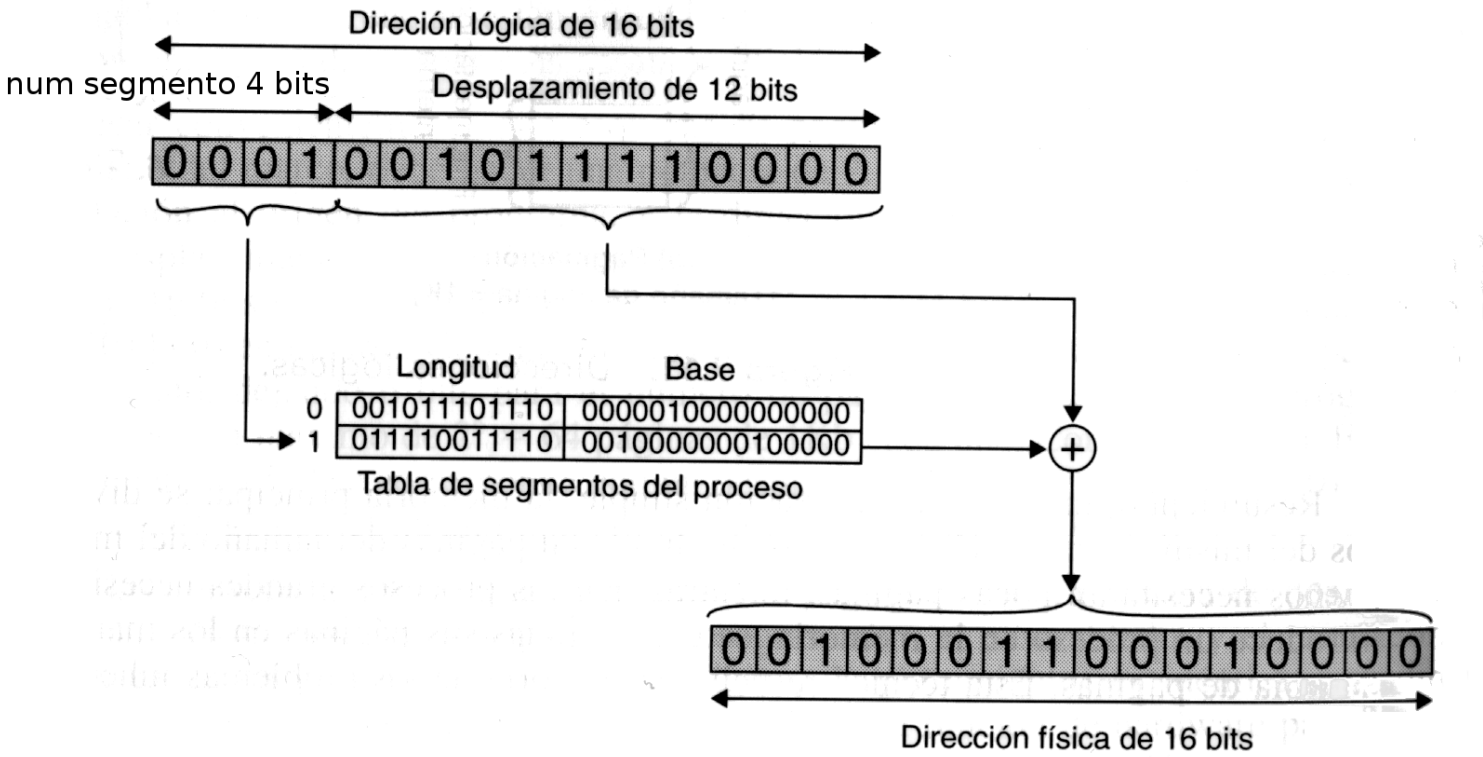
Segmentación: ejemplo

- Consideremos un sistema con segmentación con las siguientes características
  - las direcciones son de 16 bits (4 bits para el número de segmento y 12 bits para el desplazamiento)
  - cada entrada en la tabla de segmentos tine 28 bits, los 12 mas significativos para el límite y los 16 menos significativos para la base
  - Un proceso tiene dos segmentos y las dos primeras entradas de su tabla de segmentos en memoria valen 0x2EE0400 y 0x79E2020
    - ¿A donde accede una referencia a la dirección 0x12F0?
    - ¿A donde accede una referencia a la dirección 0x0342?
    - ¿A donde accede una referencia a la dirección 0x0x021F?
    - ¿A donde accede una referencia a la dirección 0x190A?

Segmentación: ejemplo

- Una referencia del proceso a la dirección de memoria 0x12F0 accede a la dirección de memoria física 0x2310
- Una referencia a la dirección de memoria 0x0342 produce un error de direccionamiento
- Una referencia a la dirección de memoria 0x0x021F accede a la dirección de memoria física 0x061F
- una referencia a la dirección de memoria 0x190A produce un error de direccionamiento

Administración de memoria: Segmentación



Fragmentación en los sistemas con segmentación

- Tiene *fragmentación externa*.
  - Dado que los segmentos son de distinto tamaño, a medida que se asignan y desasignan segmentos van quedando huecos: es posible que un segmento no pueda colocarse en memoria pues los huecos no están contiguos. Se soluciona mediante una *compactación*
- Tiene algo de*fragmentación interna*.
  - El tamaño de segmento suele ser múltiplo de alguna cantidad de memoria. Por ejemplo si el tamaño de segmento es múltiplo de 16 bytes hay **algo** de fragmentación interna

Segmentación: estrategias de asignación

- El S.O. tiene que llevar contabilidad de la memoria asignada, lo que con la segmentación supone llevar una lista de zonas ocupadas y huecos.
- A la hora de asignar un segmento en memoria puede haber varios huecos disponibles donde dicho segmento puede colocarse. Varias estrategias
  - **first fit** Primer hueco. El primer hueco que encuentra. Muy rápido, tiende a dejar huecos pequeños en las zonas bajas de la memoria y grandes en las altas (suponiendo que la búsqueda comienza por las zonas bajas)
  - **next fit** Siguiente hueco. El primer hueco que encuentra, comenzando la búsqueda a partir de donde asignó el último
  - **best fit** El hueco que mejor se ajusta. Mas lento. Tiende a generar huecos pequeños
  - **worst fit** El hueco que pero se ajusta. La idea es que dejará huecos mas grandes
- Simulaciones muestran que en general *worst fit* produce peores resultados

Segmentación: desperdicio de memoria

- En un sistema con segmentación, donde s es el tamaño promedio del segmento y k la relación entre el tamaño promedio del hueco y el tamaño promedio del segmento, la memoria desperdiciada en huecos es

$$\frac{k}{k+2}$$

- Dado que dos segmentos contiguos son dos segmentos y dos huecos contiguos son un hueco mas grande, se estima que en general hay doble número de segmentos que de huecos
- si hay n segmentos ocuparán ns, mientras que los huecos ocuparán $\frac{n}{2}ks$
- con lo que el procentaje de memoria desperdiciada en huecos es

$$\frac{memoria\ \ en\ \ huecos}{memoria\ \ total} = \frac{\frac{n}{2}ks}{\frac{n}{2}ks+ns} = \frac{k}{k+2}$$

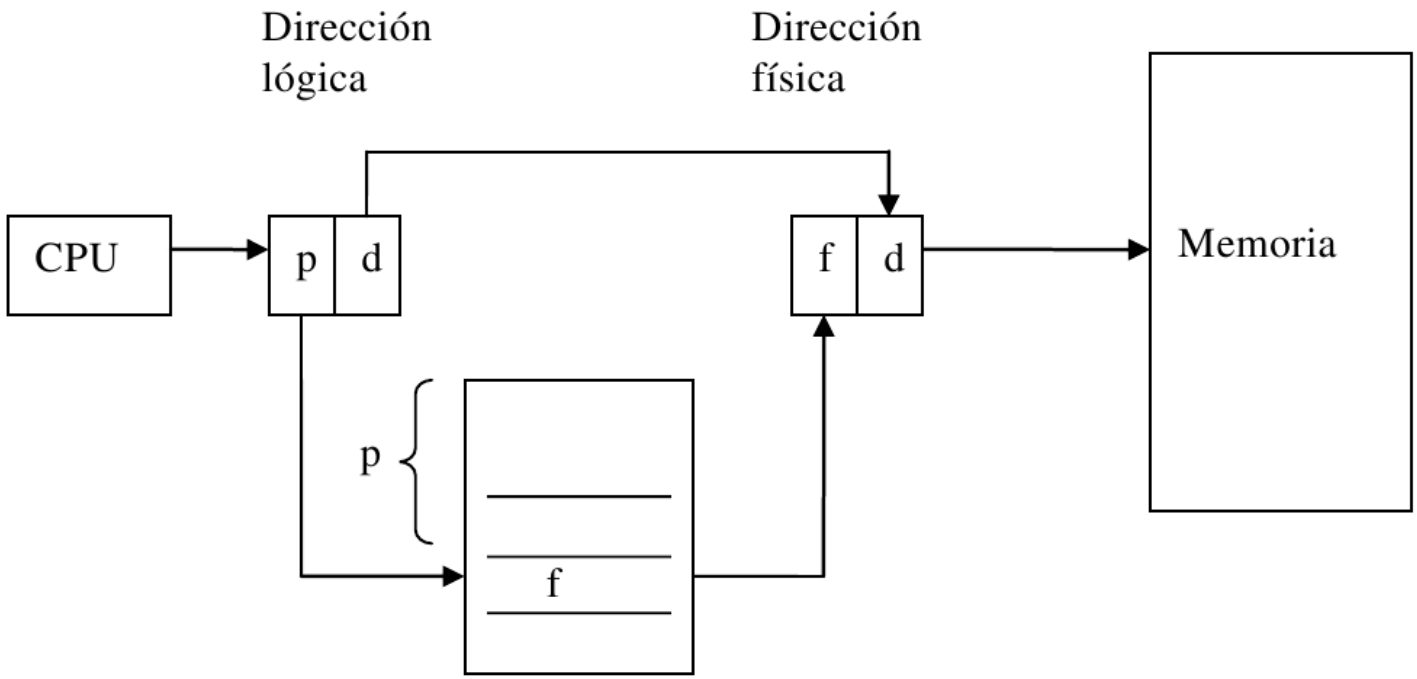
Administración de memoria: Segmentación

- REQUIERE soporte del *hardware*
- Es una manera de relocalización dinámica
- Implementa la protección de la memoria
- Permite la compartición de código (o datos) a nivel de segmento
- el intel 80286 proporcionaba segmentación. Windows 3.1 y 3.11 (en mído estndar) e IBM OS/2 hasta la versión 1.3 usaban dicha segmentación

Administración de memoria: Paginación

- Es una manera de relocalización dinámica
- El espacio de direcciones físicas está dividido en zonas de tamaño fijo llamadas *marcos de página*
- El espacio de direcciones lógico o virtual está formado por zonas de tamaño fijo denominadas páginas
- La dirección lógica se compone de número de página y desplazamiento dentro de la página
- Con el número de página se obtiene una entrada en una tabla de páginas, en donde hay una dirección base de marco de página
- La dirección de memoria física a la que se accede se obtiene sumando el desplazamiento a la dirección base del marco de página

Administración de memoria: Paginación



Paginación: Ejemplo

- Supongamos un sistema con direcciones de 16 bits donde 7 bits corresponden al número de página y 9 al desplazamiento dentro de la página
- El tamanño de página sería de 512 bytes
- Un proceso referencia la dirección 0x095f (0000 1001 0101 1111)
- Esta referencia es a la página 4 desplazamiento 0x15f

Paginación: Ejemplo

- En la entrada correspondiente a la página 4 de la tabla de páginas del proceso nos dirá la dirección de memoria fisica donde está dicha página
- Supongamos que en dicha entrada nos indica que la dirección del marco de memoria fiśica es 0xAE00 (1010 1110 0000 0000)
- Entonces la dirección de memoría física a donde realmente se accede es 0xAF5F (1010 11111 0101 1111)

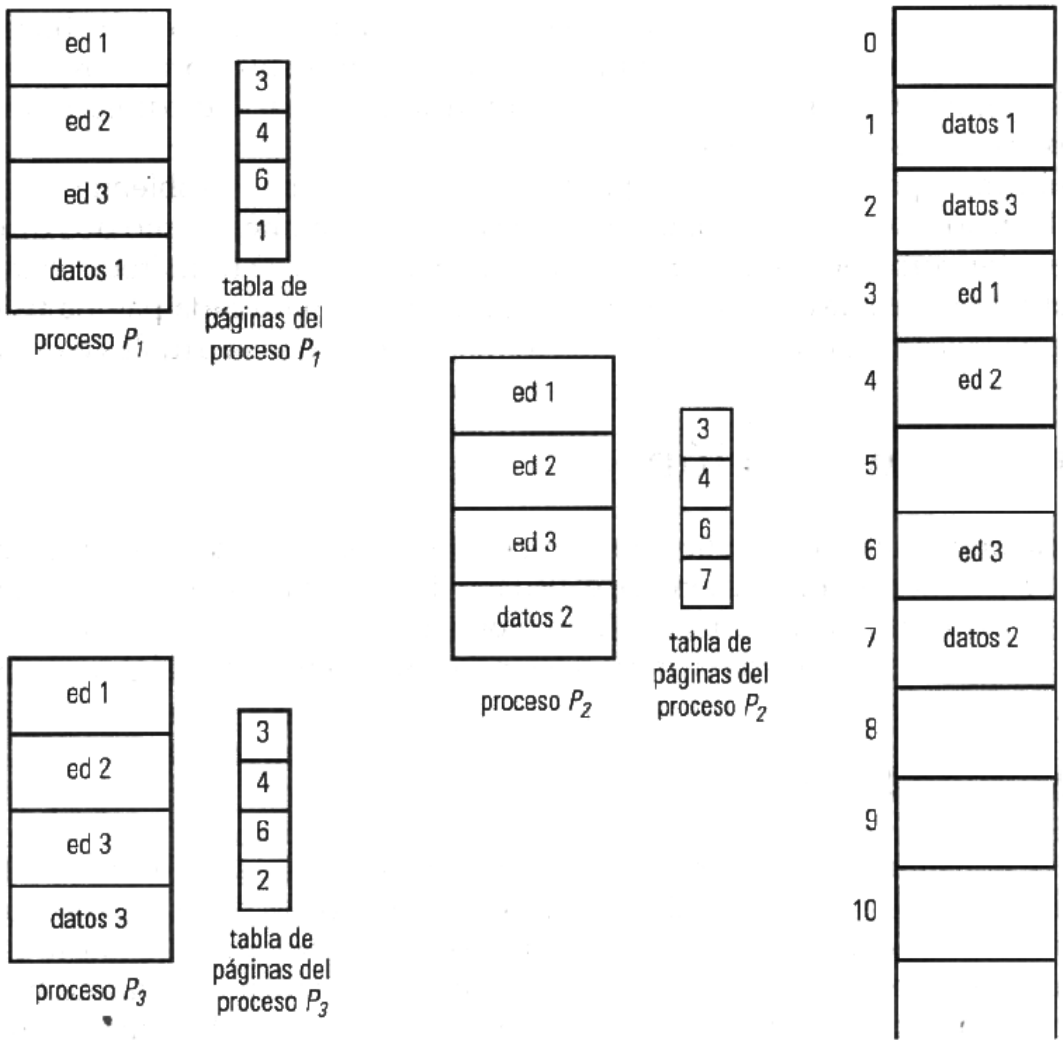
Paginación

- Con paginación la memoria de un proceso no es contigua
- El tamaño de página lo define el hardware:
  - la arquitectuta x86 de 32 bits tinene un tamaño de 4 Kbytes
  - la arquitectura SPARC de 8 Kbytes
- La paginación no tiene fragmentación externa, pero sí interna.
  - A mayor tamaño de página mayor fragmentación interna
  - A menor tamaño de página mayor gasto adicional en, p.e., tablas de páginas

Paginación

- El sistema operativo ha de llevar ahora contabilidad de
  - los marcos de memoria física
  - las páginas asignadas a cada proceso
- El S.O. se encargará ademas de gestionar las tablas de páginas de cada proceso en el cambio de contexto
- La paginación permite compartir memoria entre distintos procesos:
  - basta que las entradas correspondientes de sus tablas de páginas apunten a los mismos marcos de memoria física

Administración de memoria: Paginación



Paginación

- Con páginación la memoria está protegida
  - un proceso solo accede a donde indica su tabla de páginas
  - dicha tabla de páginas solo puede ser modificada por el S.O.
- Requiere el soporte de HARDWARE (p.e un intel 286 no tenía paginación, un i386 sí)
- Además de la dirección del marco de memoria física, en cada entrada de la tabla de páginas hay más información: página de solo lectura, modificada, nivel de privilegio ...

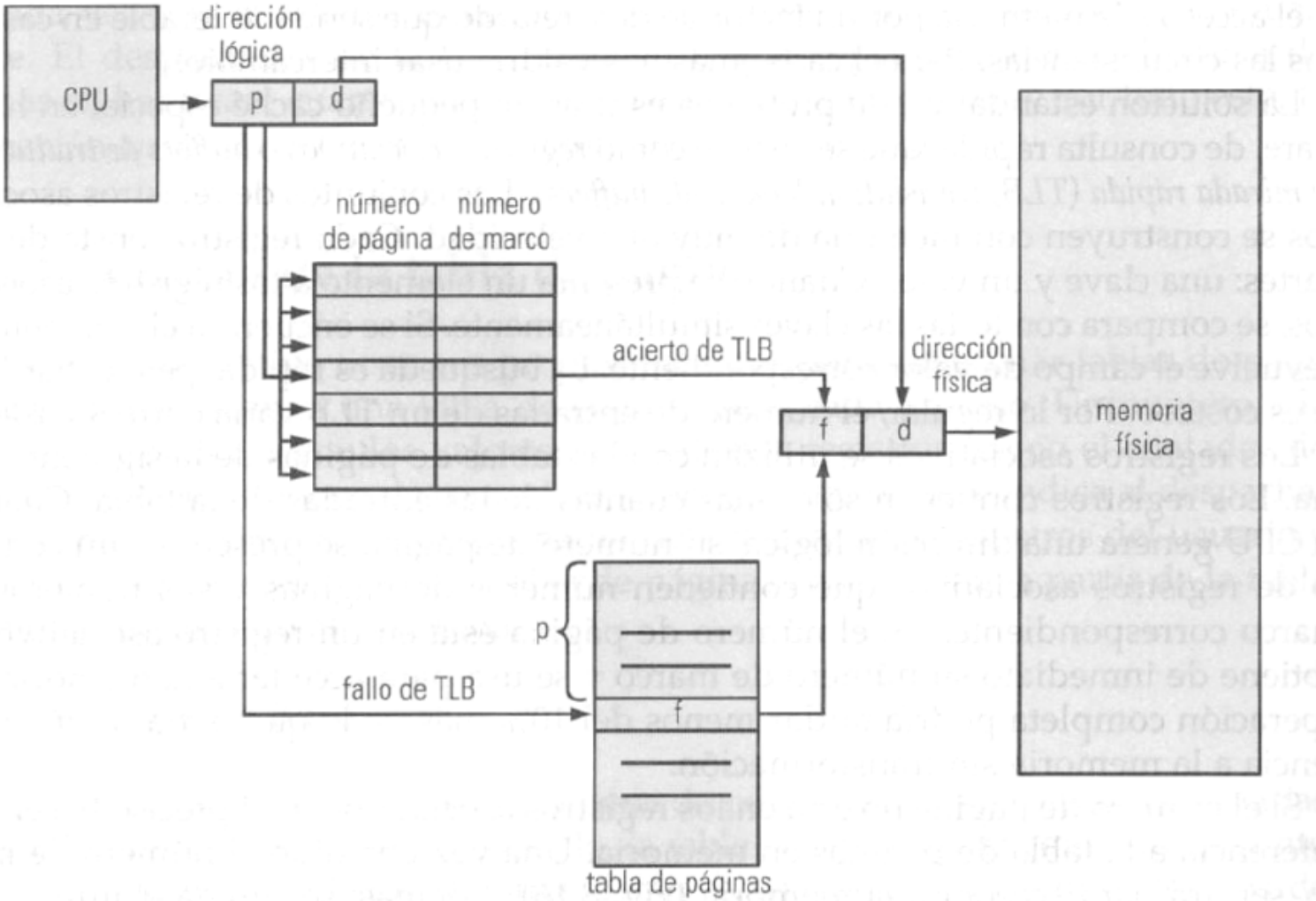
Paginación: Implementación de la tabla de páginas

- Hay dos maneras de implementar la tabla de páginas
  - **Registros dedicados:** El microporcesador tiene unos registros donde almacena la tabla de páginas del proceso en ejecución. En el cambio de contexto se salva la tabla de páginas del proceso que abandona la CPU y se carga la del que entra en CPU
    - Rápido
    - Muy costoso (hacen falta muchos registros: impracticable)
    - Cambio de contexto lento
  - **En memoria:** La tabla de páginas de un proceso se almacena en memoria. El microprocesador tiene un registro que indica en que dirección está la tabla de páginas del proceso en CPU. En el cambio de contexto solo hay que cambiar el valor de ese registro
    - Lento: Cada vez que el proceso accede a memoria son necesarios dos accesos: uno para acceder a la tabla de páginas para determinar a qué dirección hay que acceder y otro para acceder a dicha dirección

Paginación

- La implementación actual de las tablas de páginas (al igual que las de segmentos) es en memoria (implementarlas en registros dedicados sería muy costoso)
- La tabla de páginas está en memoria y un registro apunta a ella
- Se enlentece el acceso ya que son necesarios dos accesos, uno para obtener la entrada de la tabla de páginas y otro para acceder
- Se usan unas pequeñas memorias asociativas con las entradas de la tabla de página mas recientemente usadas. TLB (translation Lookaside Buffers)
  - por ejemplo, si el tiempo de acceso a memoria fuese de 100 ns y el de acceso al TLB 20, con una probabilidad del 90% de acierto en el TLB, el tiempo de acceso efectivo a memoria sería $0.9x120 + 0.1x220 = 130ns$
- El algunos sistemas hay dos niveles de TLB (un core i7 tiene un TLB1 de 64 entradas y un TLB2 de 512 entradas)

Administración de memoria: Paginación con TLB



Tablas de páginas invertidas

- En los sistemas con memoria virtual, muchas páginas de un proceso no están en memoria: sin embargo la tabla de páginas de cada proceso tiene que ser lo suficientemente grande para comprender todo el espacio de de direcciones del proceso
  - Un espacio de direcciones de 32 bits en paginas de 4K necesitaría 2^{20} páginas. La tabla de páginas tiene que tener 2^{20} entradas. Si cada entrada ocupa 4 bytes, cada tabla de páginas ocupa 4Mb
- Dos soluciones
  - **Tabla de páginas multivivel** (ver ejemplo en sistemas mixtos)
  - **Tabla de páginas invertida**
    - Una entrada por cada marco de memoria física
    - Cada entrada contiene información de la página que contiene dicho marco (proceso y dirección lógica)
    - Aumenta el tiempo de bśqueda en la tabla de páginas: se usa una tabla *hash*
    - Usado en la arquitectura PowerPC e IBM AS/400

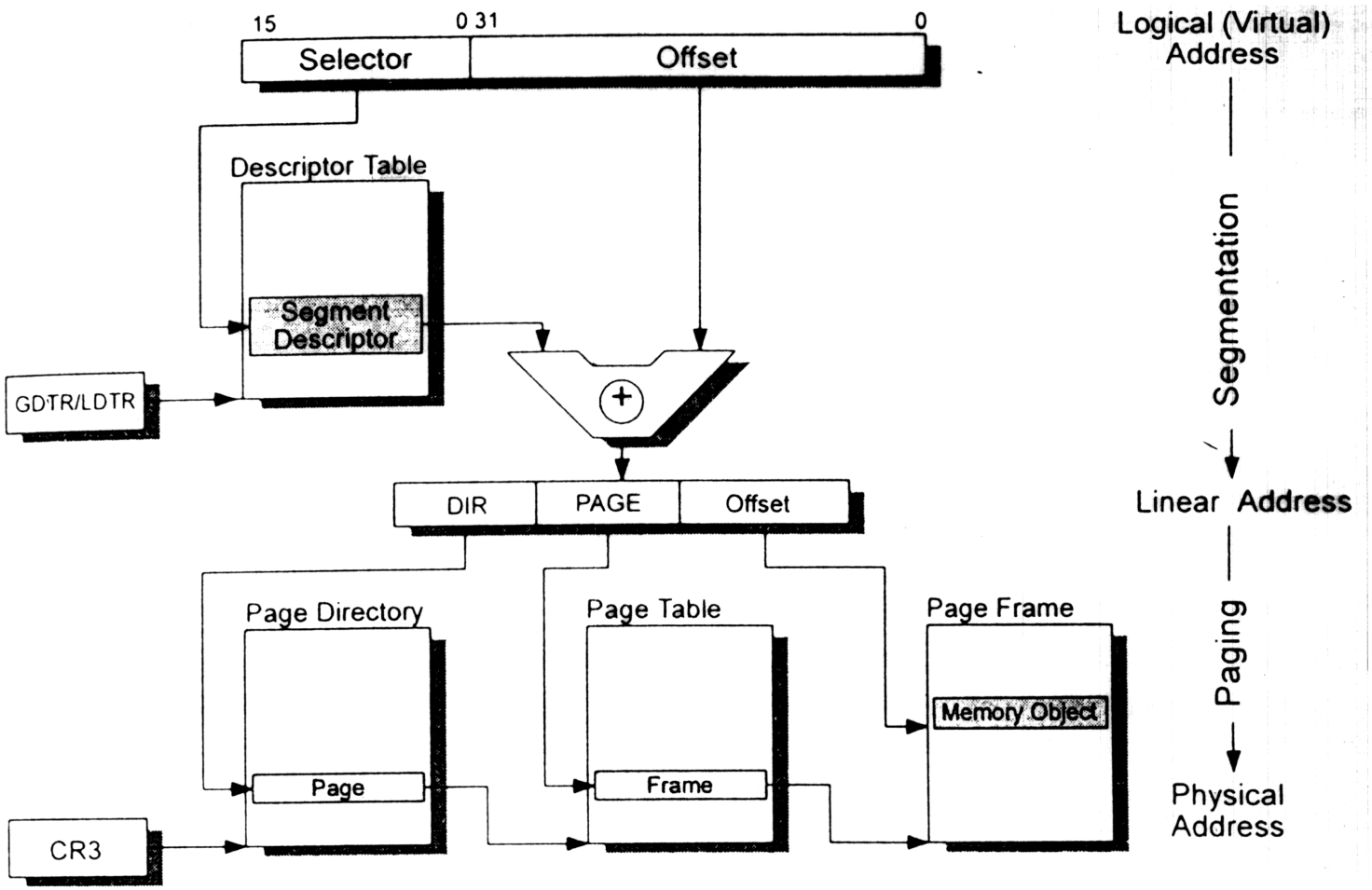
Sistemas mixtos

- Consisten en una una combinación de los sistemas vistos
- Dado que el aprovechamiento de la memoria es mejor con la paginación, en ultima instancia hay una paginación
  - **paginación segmentada**: arcaico, usado en el sistema 370 de IBM, se segmentaba la tabla de páginas de un proceso para adecuarla a su tamaño
  - **paginación en varios niveles**
  - **segmentación paginada** se paginan los segmentos
- Como ejemplo veremos como es el esquema de direccionamiento en la acrquitectura x86 de 32bits: segmentación paginada en dos niveles

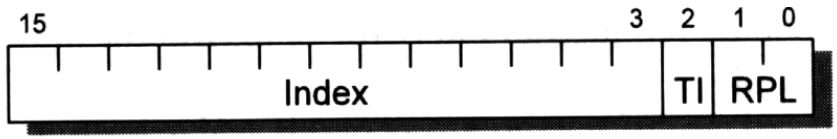
Memoria en arquitectura PC 32 bits

- Como ejemplo de sistema *actual*: la arquitectura PC de 32 bits tiene una segmentación paginada en 2 niveles
- Las direcciones se componen de
  - **selector** (16 bits) 13 bits para el número se segmento, uno para indicar que tabla usar (GDT-Global Decriptor Table o LDT-Local Descriptor Table) y 2 para el nivel de privilegio
  - **desplazamiento**
- Con los 13 bits del número segmento en *selector* se va a una tabla de descriptores (segmentos) donde hay, entre otras cosas,
  - una dirección base de 32 bits
  - un límite de 20 bits (que representa 32 bits de direccionamiento si la granularidad es de una página)

Memoria en arquitectura PC 32 bits

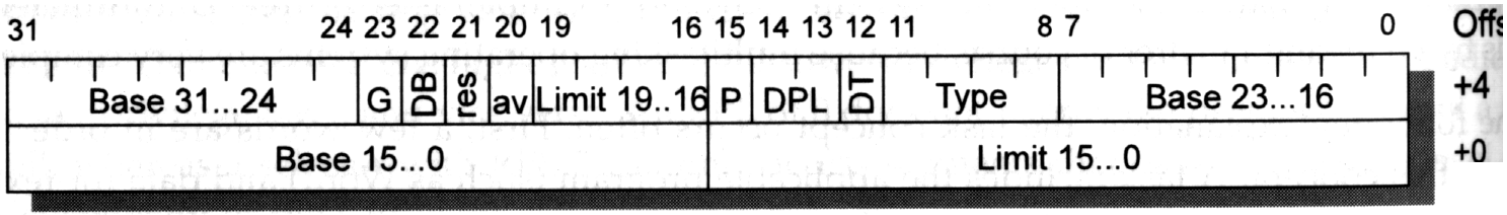


PC 32 bits: Selector de segmento



Index: Index into GDT or LDT
TI: Table Indicator
0=GDT 1=LDT
RPL: Requested Privilege Level
00=0 01=1 10=2 11=3

PC 32 bits: Descriptor de segmento

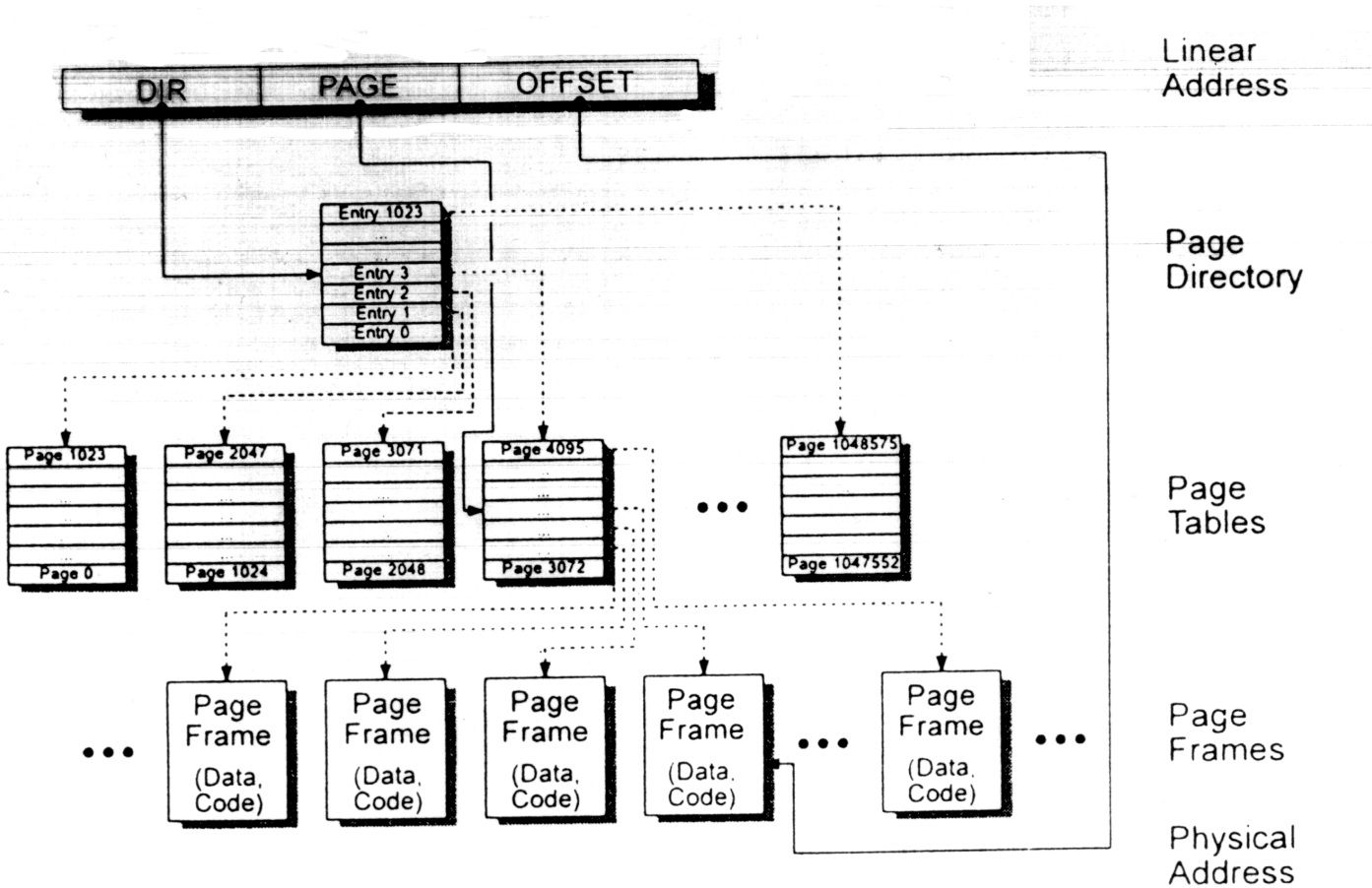


Base 31...24, Base 23...16, Base 15...0: Segment Base
Limit 19...16, Limit 15...0: Segment Limit
G: Granularity
0=Byte Granularity 1=Page Granularity
DB: Segment Size (Default/Big)
0=16-bit Operands/Addresses (Default) 1=32-bit Operands/Addresses (Big)
r: Reserved
av: Available for Operating System
P: Segment Present
0=Segment Not in Memory 1=Segment in Memory
DPL: Descriptor Privilege Level
00=0 01=1 10=2 11=3
DT: Descriptor Type
0=System Segment 1=Application Segment
Typ: Type of System or Application Segment

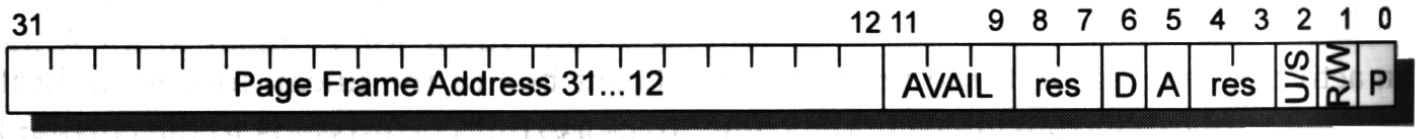
Memoria en arquitectura PC 32 bits

- Se suma la dirección base al desplazamiento y se obtiene la dirección lineal de 32 bits de la cual
  - Los 10 primeros bits corresponden a una entrada en la tabla de páginas *page directory*, de donde se obtiene la dirección de una tabla de páginas
  - Los 10 siguientes representan un índice en la tabla de páginas anterior de donde se obtiene la dirección de un marco de página
  - los 12 siguientes representa un desplazamiento dentro del citado marco
- Las páginas son de 4Kbytes
- Cada tabla de páginas tiene 1024 entradas de 4 bytes (el formato de la entrada se ve en la figura) y ocupa una página

PC 32 bits: Dirección lineal



PC 32 bits: Entrada tabla de páginas



Page Frame Address: address bits 31...12 of the page frame
AVAIL: available for the operating system
D: dirty
0=page has not yet been overwritten 1=page has been overwritten
A: accessed
0=page has not yet been accessed 1=page has already been accessed
U/S: page access level (user/supervisor)
0=supervisor (CPL=0..2) 1=user (CPL=3)
R/W: write protection (read/write)
0=page is read-only 1=page can be written to
P: page present
0=page is swapped 1=page resides in memory
res: reserved (equal 0)