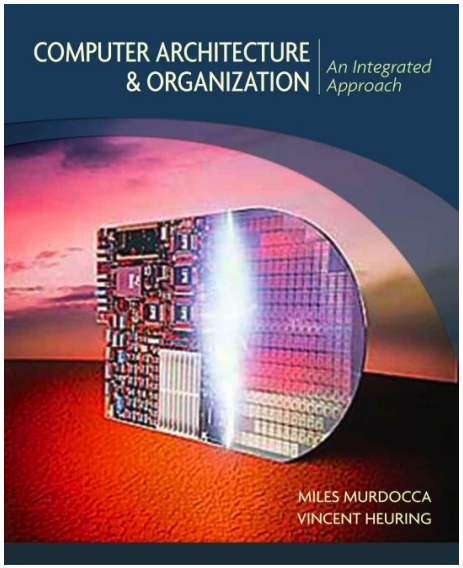
Organización de las Computadoras

Leonardo Giovanini



Organización de la memoria

Contenidos

Desarrollo y ejecución de programas

Desarrollo y ejecución de un programa

Compilación

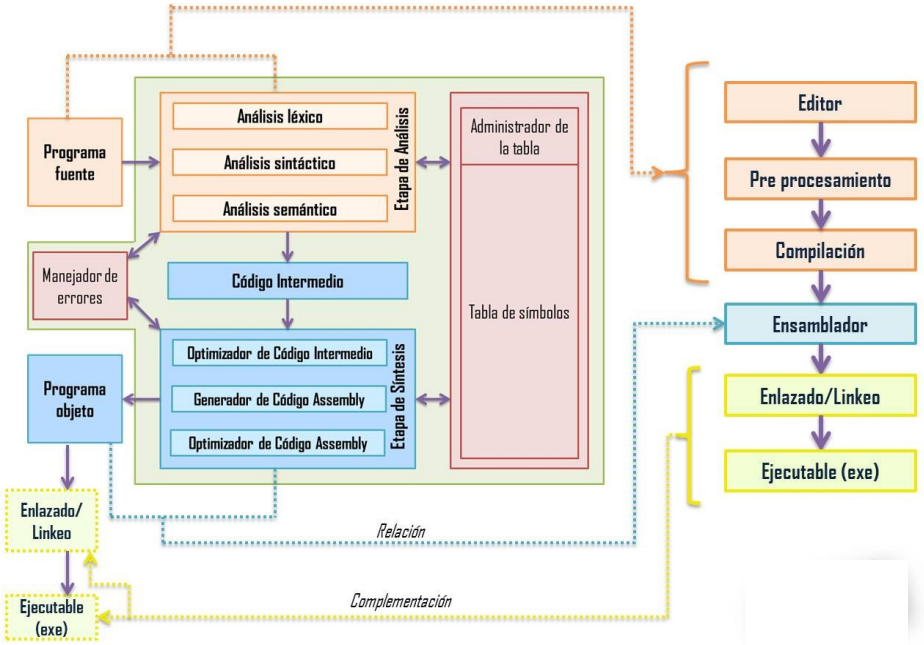
Un **Compilador** traduce un pragrama escrito en ***lenguaje de alto nivel*** (*código fuente*), el cual es *independente de la architectura* de la CPU, en ***lenguaje ensamblador***, el cual *depende de la arquitectura* de CPU utilizada.

Código fuente

```
int f, g, y; // global variables

int main (void)
{
  f = 2;
  g = 3;
  y = sum (f, g);
  return y;
}

int sum (int a, int b) {
  return (a + b);
}
```

Código ensamblador

```
.data
f:
g:
y:

.text
main:
  addi  $sp, $sp, -4  # make stack frame
  sw    $ra, 0($sp)   # store $ra on stack
  addi  $a0, $0, 2    # $a0 = 2
  sw    $a0, f        # f = 2
  addi  $a1, $0, 3    # $a1 = 3
  sw    $a1, g        # g = 3
  jal   sum           # call sum procedure
  sw    $v0, y        # y = sum (f, g)
  lw    $ra, 0($sp)   # restore $ra from stack
  addi  $sp, $sp, 4   # restore stack pointer
  jr    $ra           # return to operating system

sum:
  add   $v0, $a0, $a1 # $v0 = a + b
  jr    $ra           # return to caller
```

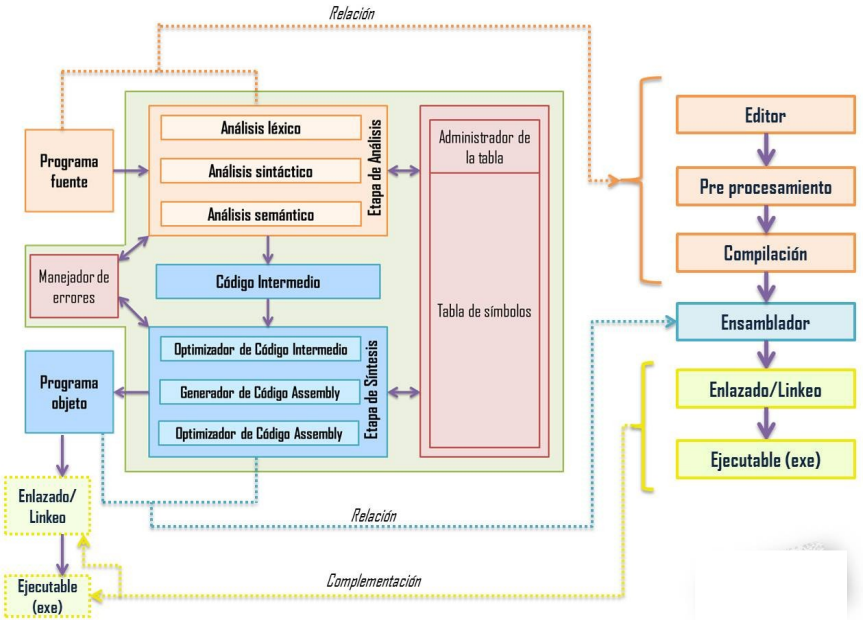


Desarrollo y ejecución de un programa
Ensamblado

El **lenguaje ensamblador** es un lenguaje de programación de bajo nivel que consiste en un conjunto de mnemónicos que representan instrucciones básicas de los procesadores.

Implementa una **representación simbólica** de los códigos de máquina binarios e información necesarias para programar un procesador y es la representación ***directa del código máquina***.

```
0x00400000   main:   addi  $sp, $sp, -4
0x00400004           sw    $ra, 0($sp)
0x00400008           addi  $a0, $0, 2
0x0040000C           sw    $a0, f
0x00400010           addi  $a1, $0, 3
0x00400014           sw    $a1, g
0x00400018           jal   sum
0x0040001C           sw    $v0, y
0x00400020           lw    $ra, 0($sp)
0x00400024           addi  $sp, $sp, 4
0x00400028           jr    $ra
0x0040002C   sum:    add   $v0, $a0, $a1
0x00400030           jr    $ra
```



Un **ensamblador** traduce programa en ***lenguaje ensamblador*** en ***código de máquina*** en dos pasos:

- Asigna una dirección a cada instrucción y busca los símbolos (etiquetas y nombre de variables);
- Produce el código de máquina y la tabla de símbolos.

El código de máquina y la tabla de símbolos son almacenadas en el **archivo objeto**.

Desarrollo y ejecución de un programa
Enlazado

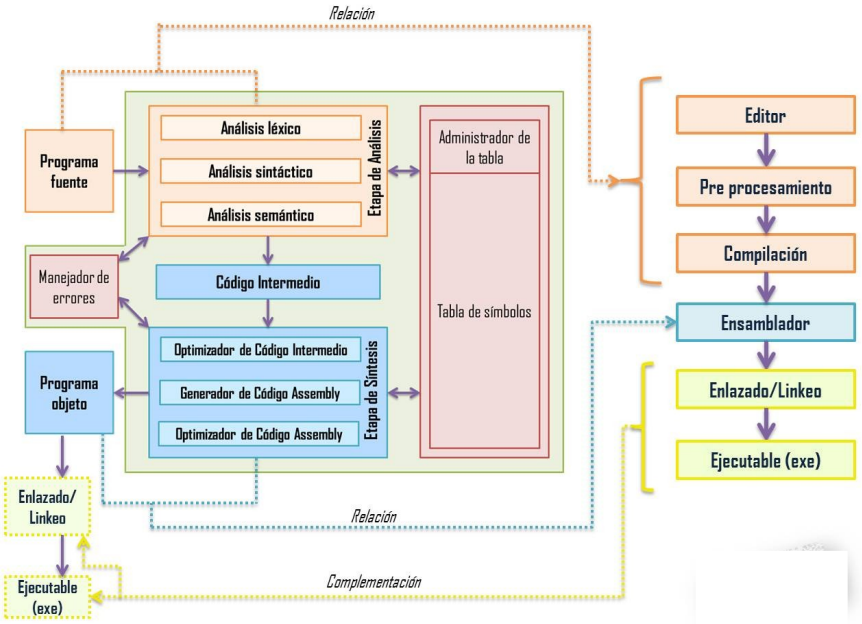
El **enlazador** organiza los datos e instrucciones en un archivo que puede ***ser ejecutado*** por el procesador una vez cargado en memoria.

Text Size	Data Size
0x34 (52 bytes)	0xC (12 bytes)

Address	Instruction
0x00400000	0x23BDFFFC
0x00400004	0xAFBF0000
0x00400008	0x20040002
0x0040000C	0xAF848000
0x00400010	0x20050003
0x00400014	0xAF858004
0x00400018	0x0C10000B
0x0040001C	0xAF828008
0x00400020	0x8FBF0000
0x00400024	0x23BD0004
0x00400028	0x03E00008
0x0040002C	0x00851020
0x00400030	0x03E0008

Address	Data
0x10000000	f
0x10000004	g
0x10000008	y

```
addi $sp, $sp, -4
sw   $ra, 0 ($sp)
addi $a0, $0, 2
sw   $a0, 0x8000 ($gp)
addi $a1, $0, 3
sw   $a1, 0x8004 ($gp)
jal  0x0040002C
sw   $v0, 0x8008 ($gp)
lw   $ra, 0 ($sp)
addi $sp, $sp, -4
jr   $ra
add  $v0, $a0, $a1
jr   $ra
```



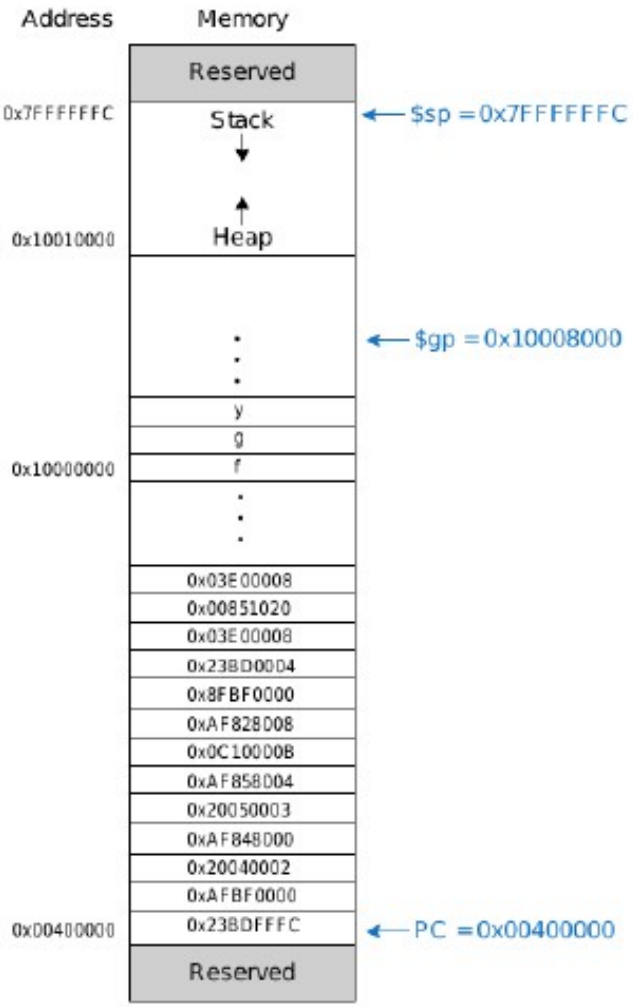
Un enlazador toma los objetos generados en el proceso de compilación, la información de todos los recursos necesarios (biblioteca), quita aquellos recursos que no necesita, y enlaza el código objeto con sus bibliotecas con lo que finalmente produce un fichero ejecutable.

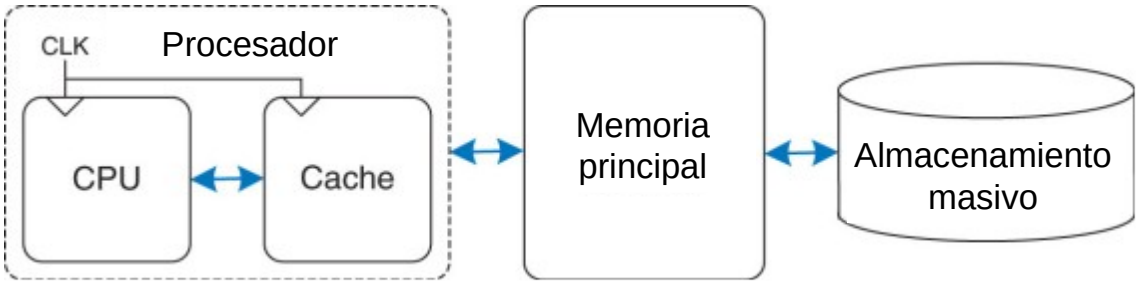
En el caso de los programas enlazados dinámicamente, el enlace entre el programa ejecutable y las bibliotecas se realiza en tiempo de carga o ejecución del programa.

Desarrollo y ejecución de un programa

Carga y Ejecucion

El sistema operativo lee el archivo ejecutable del dispositivo de almacenamiento y lo carga en la memoria de la computadora para su ejecución.





Dependiendo del entorno utilizado,

- En ***procesadores avanzados*** debe ubicarse en el contexto de un **proceso**, y la gestión esta a cargo de la unidad de manejo de memoria,
- En ***procesadores convencionales*** se ubica en la memoria disponible y la gestión esta a cargo del programador.

La ubicación de cada segmento del programa (código, datos, variables locales y pila) depende de las características técnicas del procesador definidas en el ISA.

Desarrollo y ejecución de un programa

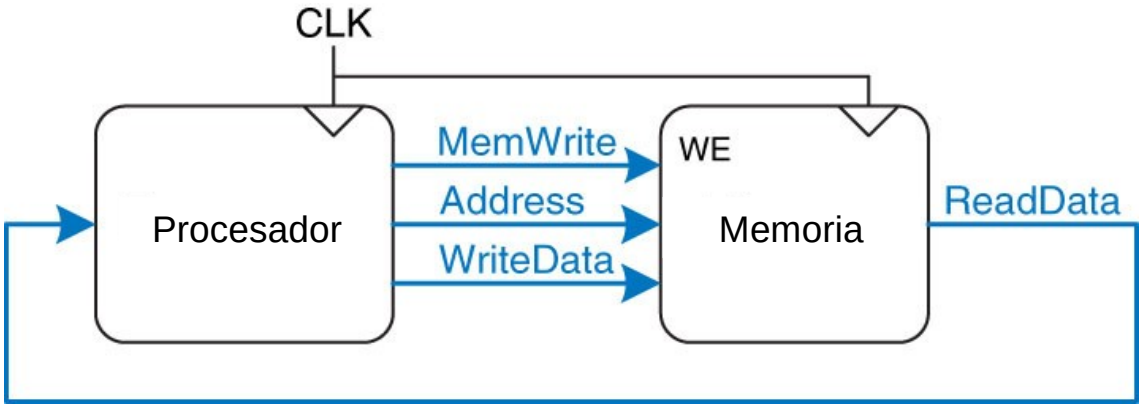
Ejecucion

Los ***registros y la memoria son los únicas formas de almacenamiento*** a la que el procesador puede acceder directamente para ***ejecutar un programa***.

Sin embargo, puede almacenar datos y programas en perifericos.

La unidad de memoria sólo ve desde el procesador:

- Requerimentos de lectura mas las correspondientes direcciones;
- Requerimientos de escritura mas las correspondientes direcciones y datos.

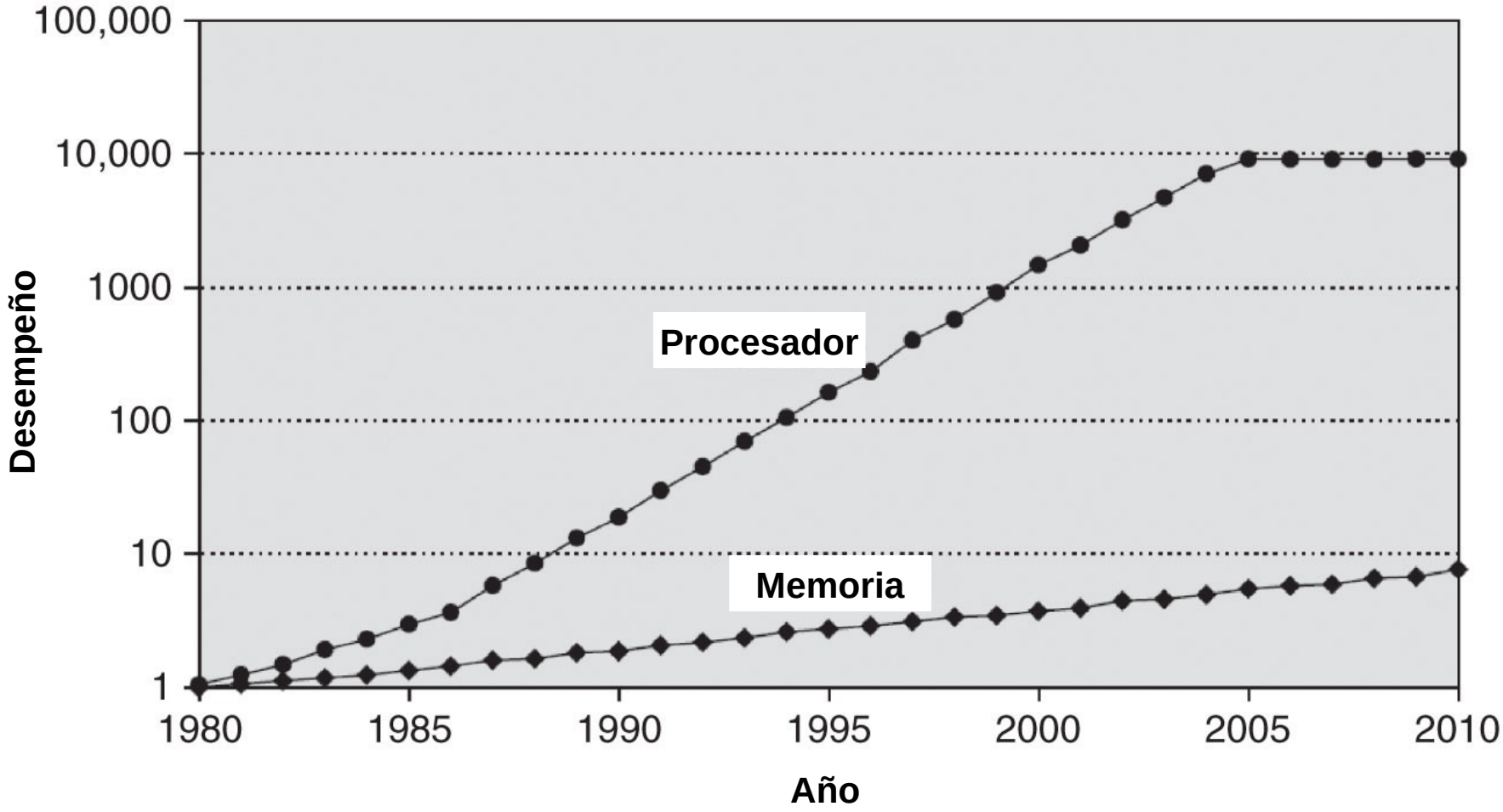


Por lo tanto, el ***desempeño de una computadora*** depende de los ***desempeños de su procesador y su sistema de memoria***

Instruction time = (CPU clock cycles + Memory-stall clock cycles) * Cycle time

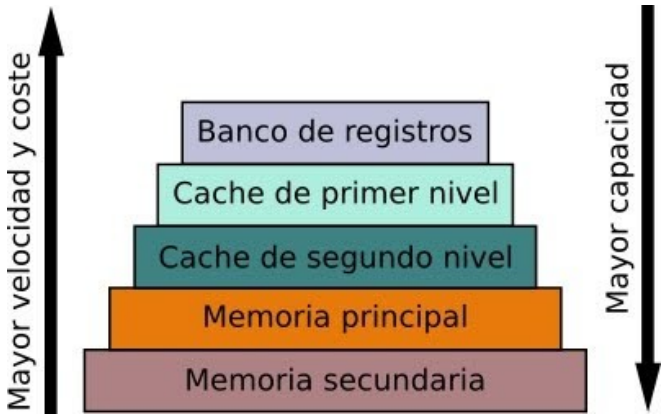
Desarrollo y ejecución de un programa

Ejecución



Jerarquia de almacenamiento

Jerarquia de almacenamiento



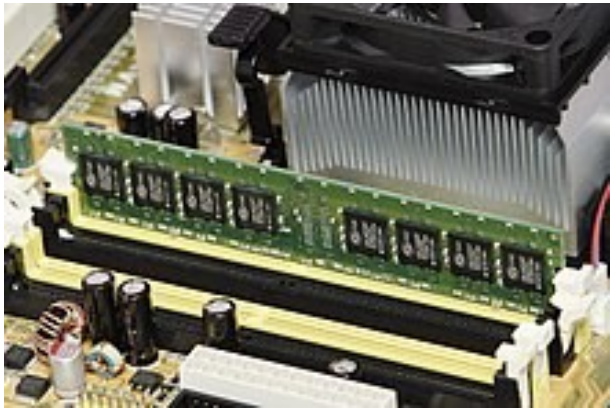
Para mejorar el desempeño de una computadora y mantener su costo bajo el sistema de almacenamiento esta organizado en una jerarquia determinada por

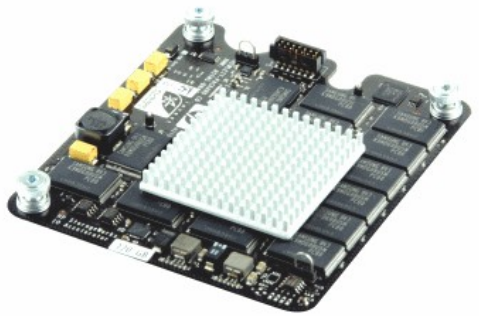
- **Velocidad**
- **Costo**
- **Volatilidad**

Hay cuatro niveles de almacenamiento

- ***Interno*** – Esta constituido por los registros del procesador y consiste en una pequeña cantidad de almacenamiento rápido, aunque algunos tienen funciones de hardware específicas. Este nivel se trata con mecanismos de direccionamiento distintos a la memoria principal;
- **Principal** – Esta constituida por la memoria del sistema y las memorias caches. Consiste en circuitos integrados computadora que almacenan información para uso inmediato. Esta organizadaen palabras de longitud de palabra fija que agrupan celdas de memoria que almacena un bit;
- **Almacenamiento masivo on-line (*secundario*) –** Esta constituido por las unidades de disco duro, estado sólido y discos flash USB. No es accesible directamente por el procesador, el cual utiliza canales de entrada/salida para accederlos y transferir los datos deseados al almacenamiento primario. El almacenamiento masivo on-line retiene datos cuando se apaga la alimentación.
- **Almacenamiento masivo off-line (*terciario*) –** Esta constituida por unidades de almacenamiento masivos extraibles (cintas magneticas, discos ópticos). Se utiliza para archivar información a la que rara vez se accede, de modo que se utiliza para almacenar y/o transportar cantidades extraordinariamente grandes de datos.

Jerarquia de almacenamiento





Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Jerarquia de almacenamiento

Escala de tiempos

Evento	Latencia	Escala
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 µs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia

Generación de las direcciones

Generación de las direcciones

Durante su desarrollo, un programa pasa a traves de diferentes pasos (***Compilación***, ***Enlazado*** y ***Carga***) antes de ser ejecutado.

Las direcciones son representadas de diferentes maneras a lo largo de estas etapas.

- El codigo fuente tipicamente utiliza direcciones simbólicas;
- Un compilador típicamente relaciona la dirección simbólica con una dirección relocalizable; y
- El enlazador y/o el cargador relacionan la direccion relocalizable con una dirección absoluta (física).

La relación (***binding***) entre las direcciones de las instrucciones y datos con las direcciones en memoria se pueden calcular durante:

- **La compilación -** si uno conoce la posición de memoria donde se ubicará el proceso;
- **La carga -** si no se conoce la dirección de memoria. En este caso, el compilador debe generar un **código relocalizable** cuya dirección de inicio cambia cada vez que se cargar; y
- **La ejecucion -** si el proceso puede moverse durante la ejecución desde un lugar de memoria a otro.

Generación de las direcciones

Direcciones logicas y fisicas

Una dirección ***identifica de forma única*** una posición en la memoria.

Hay dos tipos de direcciones: la ***dirección lógica*** y la ***dirección física***.

La dirección lógica es la ***dirección vista por el usuario*** (programa) y se utiliza como una ***referencia*** para acceder a la dirección física.

La dirección física que refiere a ***una posición*** en la unidad de memoria del sistema.

La ***diferencia fundamental*** entre la dirección lógica y la física es que el procesador genera una ***dirección lógica durante la ejecución de un programa***, mientras que la ***dirección física se genera cuando se accede la unidad de memoria***.

El ***espacio de direcciones virtuales*** es el conjunto de todas las direcciones a las que puede hacer referencia un programa, mientras que el ***espacio de direcciones físicas*** es el conjunto de todas las direcciones de memoria físicas a las que accede un programa.

Por lo tanto

- Las direcciones logicas y fisicas son **iguales** para la generación de direcciones en la **compilación** y la **carga** del programa;
- Las direcciones logicas y fisicas son **diferentes** para la generación de direcciones en la **ejecución** del programa.

Generación de las direcciones

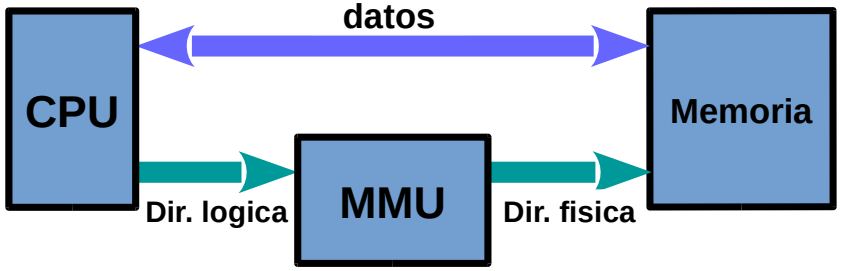
Direcciones logicas y fisicas

Los programas solo trabajan con direcciones lógicas

- Nunca ven las **direcciones físicas** utilizadas por el procesador durante su ejecucion; y
- La direccion lógica es **independiente** de las direcciones físicas disponibles en el sistema.

Estas ideas obligan a introducir hardware especializado en el sistema: **la Unidad de Manejo de Memoria (MMU).**

Es un circuito especializado que (durante la ejecución) mapea las direcciones lógicas en direcciones físicas.



La dirección de salida del procesador a la unidad de administración de memoria es la dirección logica mientras que la salida de la MMU es la dirección física, traduciendo las direcciones logicas a direcciones físicas.

Las funciones de la MMU son

- **Gestionar** el acceso de los programas al sistema de memoria;
- **Automatizar** las transferencias de datos **entre niveles** en la jerarquía de memoria;
- **Aislar** la ejecucion de los programas; y
- **Ampliar los espacios de direcciones** de las arquitectura, conectando los procesadores a memorias físicas más grandes.

Manejo de memoria

Manejo de memoria

El manejo de memoria es un ***conjunto de tecnicas aplicados a la gestion memoria*** de una computadora para ***asignar dinámicamente*** la memoria necesaria a los programas que lo soliciten, y liberarla para reutilizarla cuando ya no sea necesaria.

El manejo de memoria realiza un ***seguimiento del estado*** de cada bloque de memoria, ya sea asignada o libre. Determina cómo se asigna la memoria entre los procesos, decidiendo cuanta memoria recibe, cuándo y donde se la asigna.

Las tecnicas de manejo de memoria son

- **Paginacion** – es una tecnica que divide la memoria fisica y el espacio de direcciones logicas en unidades de tamaño fijo. La MMU asigna unidades de memoria física con unidades de memoria logica de modo que el espacio de direcciones logicas es continuo. La paginacion requiere soporte de hardware en forma de tablas que contienen la dirección física y otros datos (bits de protección de acceso y de estado, entre otros) de cada pagina;
- **Segmentacion** - es una técnica de gestión que proporciona al usuario de un "*espacio de direcciones lineal y contiguo*". Los segmentos son áreas de memoria que corresponden a agrupaciones lógicas de información. La segmentacion requiere soporte de hardware en forma de tablas que contienen la dirección física, tamaño y otros datos (bits de protección de acceso y de estado, entre otros) de cada segmento;
- **Memoria virtual** – es una tecnica que separa las direcciones utilizadas por un proceso (direcciones logicas) de las direcciones físicas reales, lo que permite la separación de procesos y aumenta el tamaño del espacio de direcciones logicas más allá de la cantidad de memoria fisica disponible mediante el intercambio con almacenamiento secundario.

Manejo de memoria

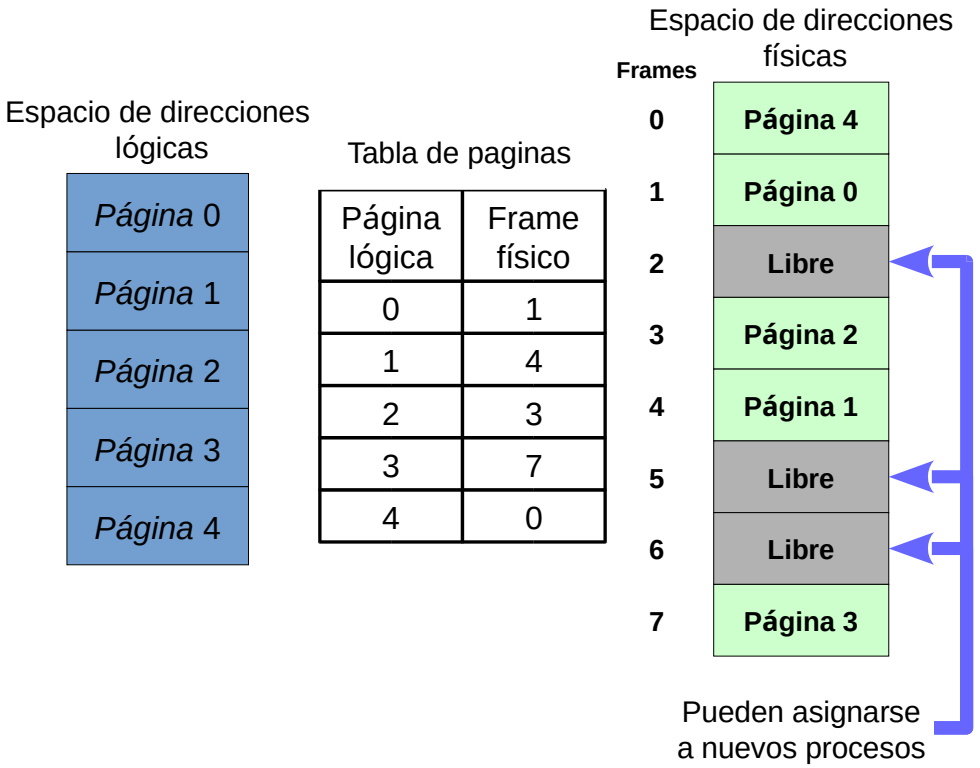
Paginación

La paginacion es una tecnica de gestion de memoria que divide la memoria fisica y el espacio de direcciones logicas en unidades de tamaño fijo llamados frames y paginas, respectivamente.

A cada ***página logica*** se le asigna a un ***marco físico*** de tamaño similar en la memoria principal.

Los marcos no tienen que ser contiguos en la memoria fisica, por lo que cada proceso puede dispersarse en toda la memoria y puede verse como una colección de cuadros que no son necesariamente contiguos.

Utiliza una ***tabla de páginas*** para realizar un seguimiento de dónde se encuentra cada página lógica dentro del espacio de las direcciones fisicas y su estado.

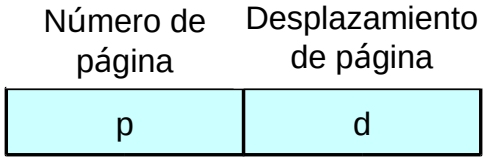


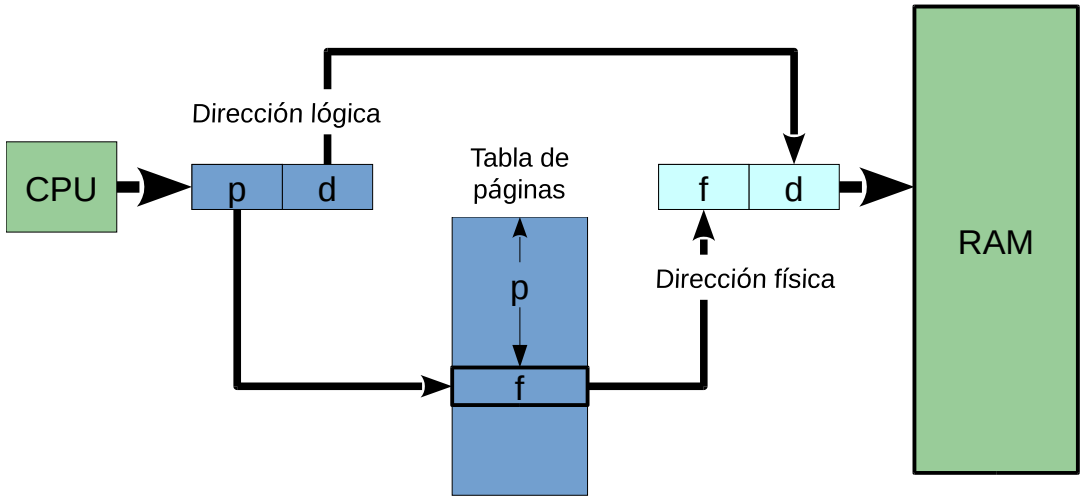
Manejo de memoria

Paginación

La direccion logica generada por el procesador es dividida en partes

- **Numero de pagina** (***p***) – utilizado como un índice en la **tabla de página** la cual contiene la direccion base de cada página en la memoria fisica; y
- **Offset de la pagina** (***d***) – combinado con la dirección base define la direccion física de la unidad de memoria.





Esta informacion es almacenada en tablas que contienen las entradas de cada pagina y frame.

Una opción es usar un banco dedicado de registros de hardware para almacenar la tabla de páginas.

Las principales caracteristicas de este enfoque son

- Traducción rápida;
- Cambio de contexto lento - la tabla de páginas completa tiene que ser recargada; y
- Limitacion de tamaño - algunas tablas de páginas pueden ser demasiado grandes.

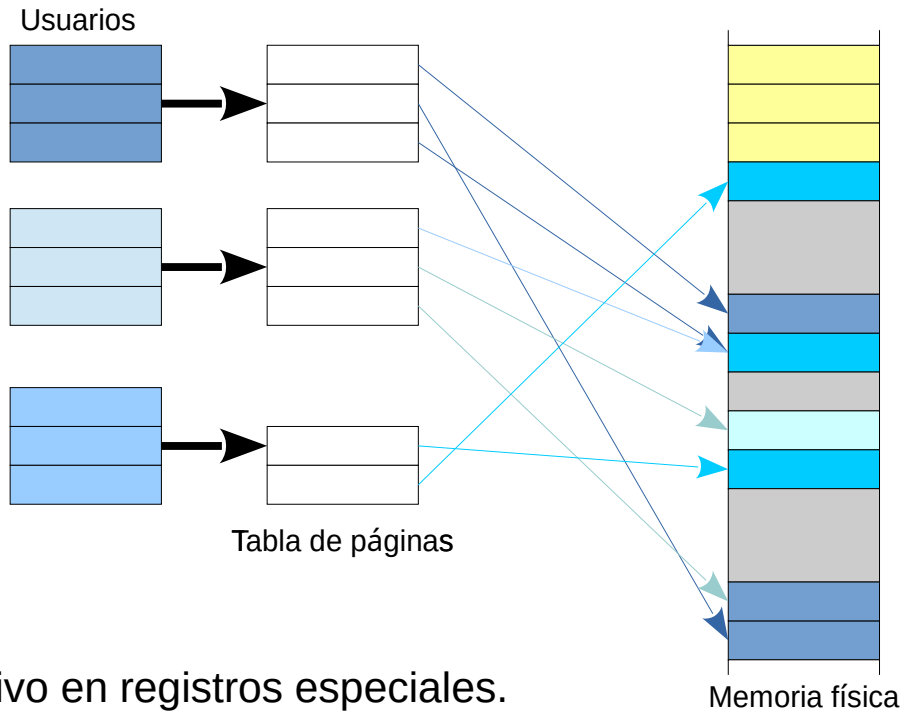
Manejo de memoria

Paginación

Cada usuario tiene una tabla que contiene las entradas de cada pagina.

El espacio necesario por las tablas de paginas es proporcional al espacio de direcciones y el numero de procesos.

En general es muy grande para ser almacenadas en registros.



La idea es almacenar la tabla del proceso activo en registros especiales.

Las tablas de página se almacenan en la memoria principal y la información para accederlas se almacena en dos registros

- **Registro de base de la tabla de página** (**PTBR**) – la cual almacena la direccion física de la tabla de página de cada proceso; y
- **Registro de longitud de tabla de página** (**PTLR**) – la cual indica el tamaño de la tabla de página.

Cambiar la tabla de página sólo requiere cambiar dos registros.

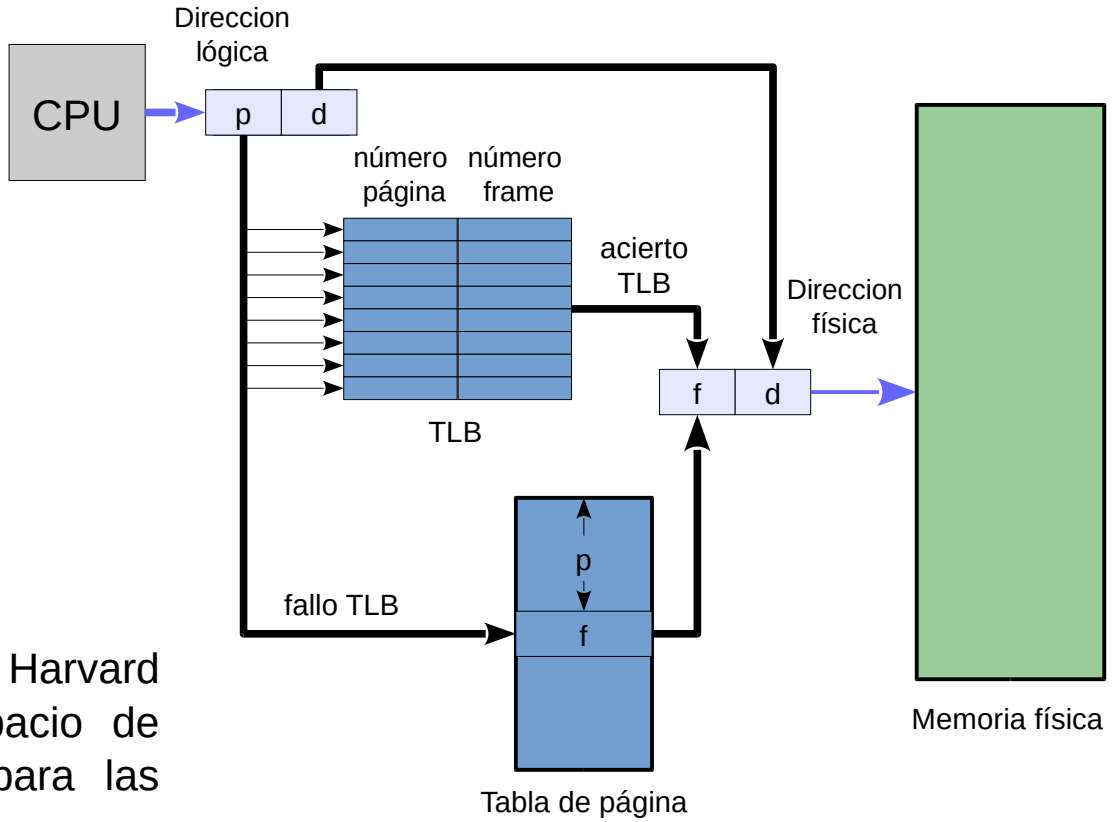
Manejo de memoria

Paginación

La solucion es utilizar una tabla de hardware de rapida lectura basada en memorias asociativas llamada **Translation Look-aside Buffer** (**TLB**).

La TLB almacena las paginas y frames (*p,f*) de las páginas utilizadas con mayor frecuencia.

Sus principales caracteristiscos son su pequeño tamaño y elevada velocidad de traduccion.



En una arquitectura de Harvard o Harvard modificada, puede existir un espacio de direcciones virtuales separados para las instrucciones y los datos.

Esto lleva al uso de distintos TLB para cada tipo de acceso, un búfer de instrucciones (ITLB) y un búfer de datos (DTLB).

Ademas, los TLB pueden tener múltiples niveles, un pequeña y totalmente asociativo que es extremadamente rápido (L1-TLB) y una más grande que es más lenta (L2-TLB).

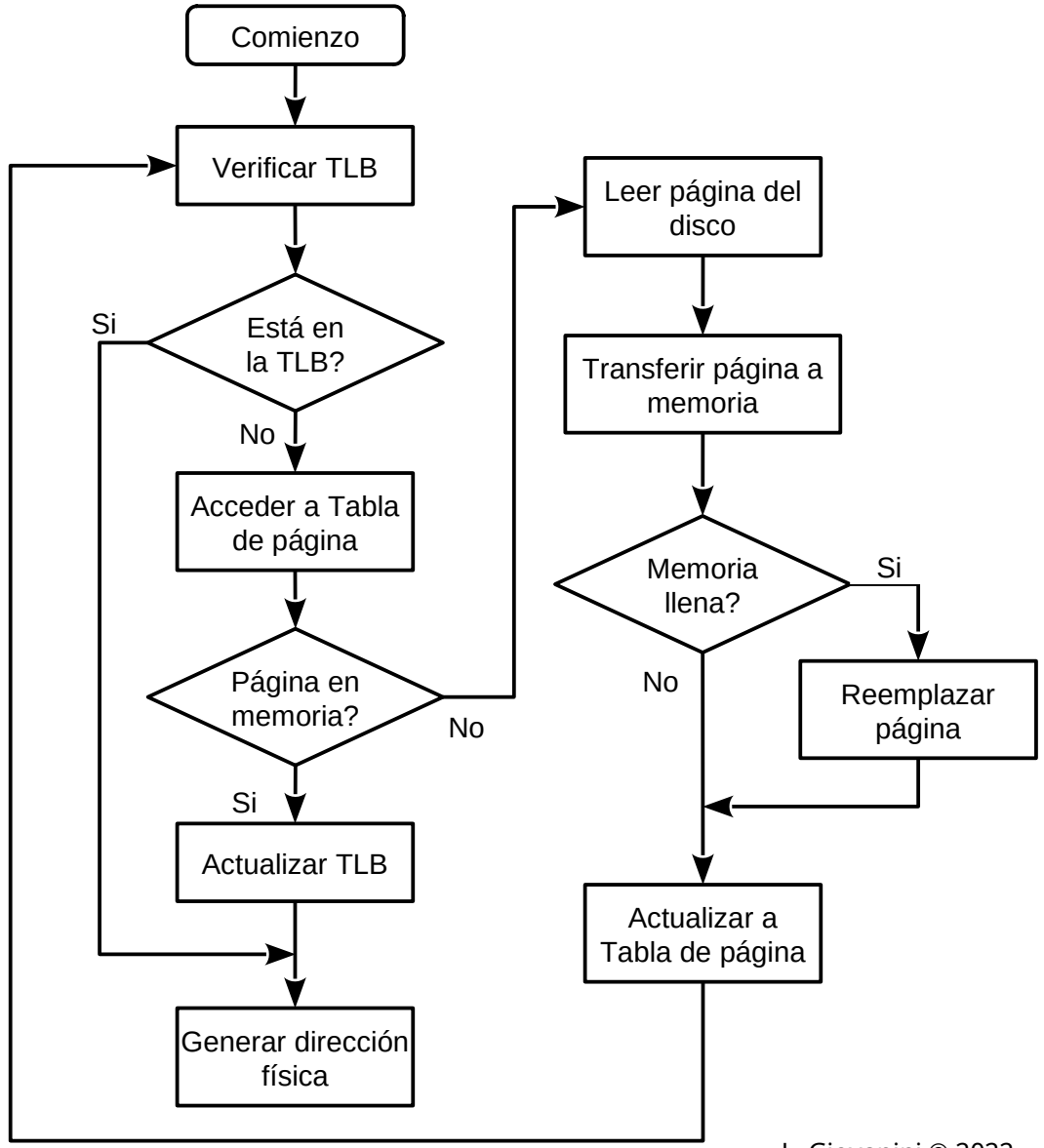
Manejo de memoria

Paginación

Si se produce un fallo de TLB, entonces la CPU verifica la tabla de páginas para encontrar la entrada de la tabla de páginas correspondiente.

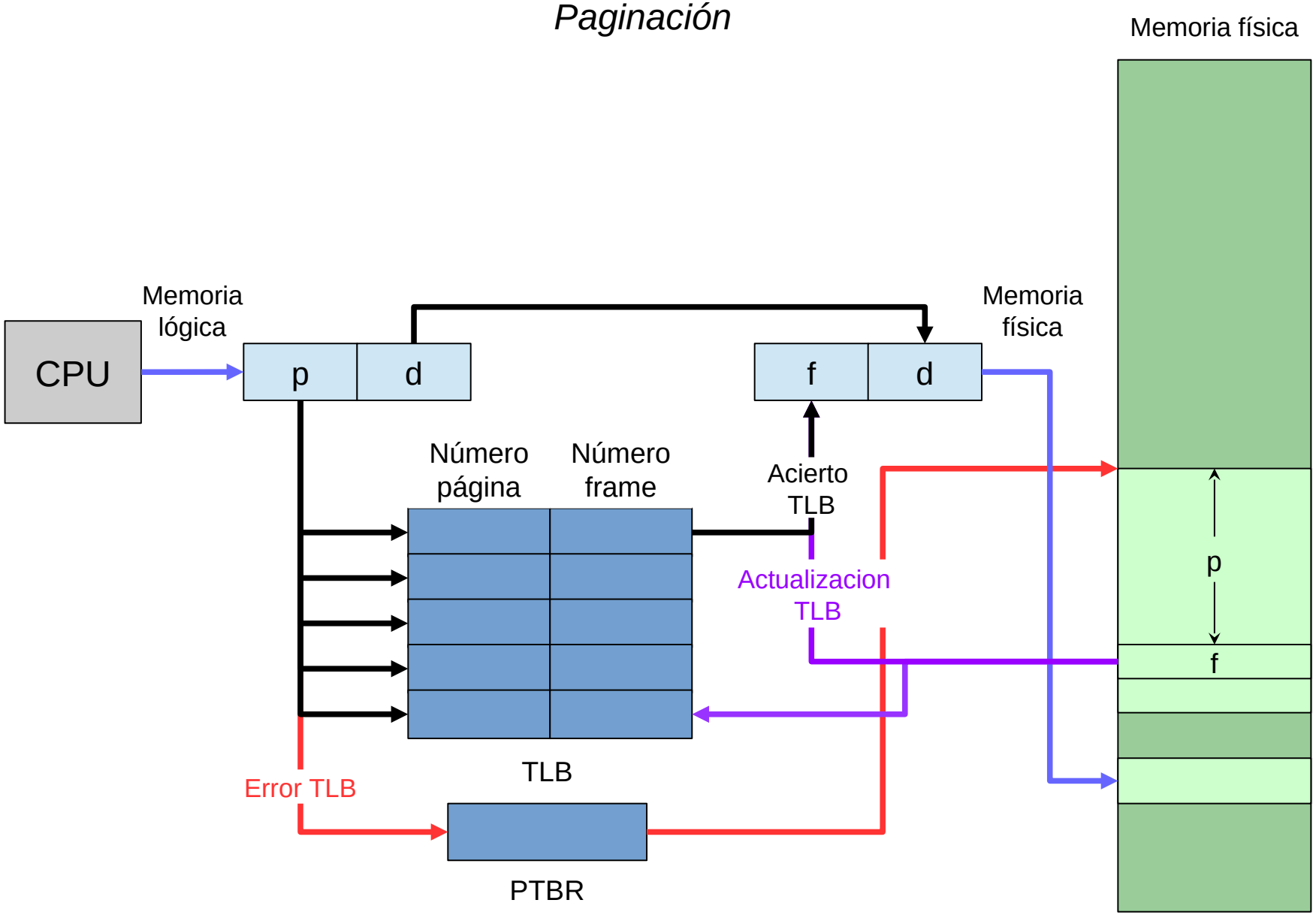
Si el bit actual es 1, entonces la página está en la memoria principal y el procesador puede recuperar el frame para formar la dirección física. Finalmente, el procesador actualiza el TLB incluyendo la nueva entrada.

Si el bit actual es 0, entonces la página deseada no está en la memoria principal y se genera una excepcion de error de página, la cual ejecuta la rutina de manejo de fallo de página.



Manejo de memoria

Paginación



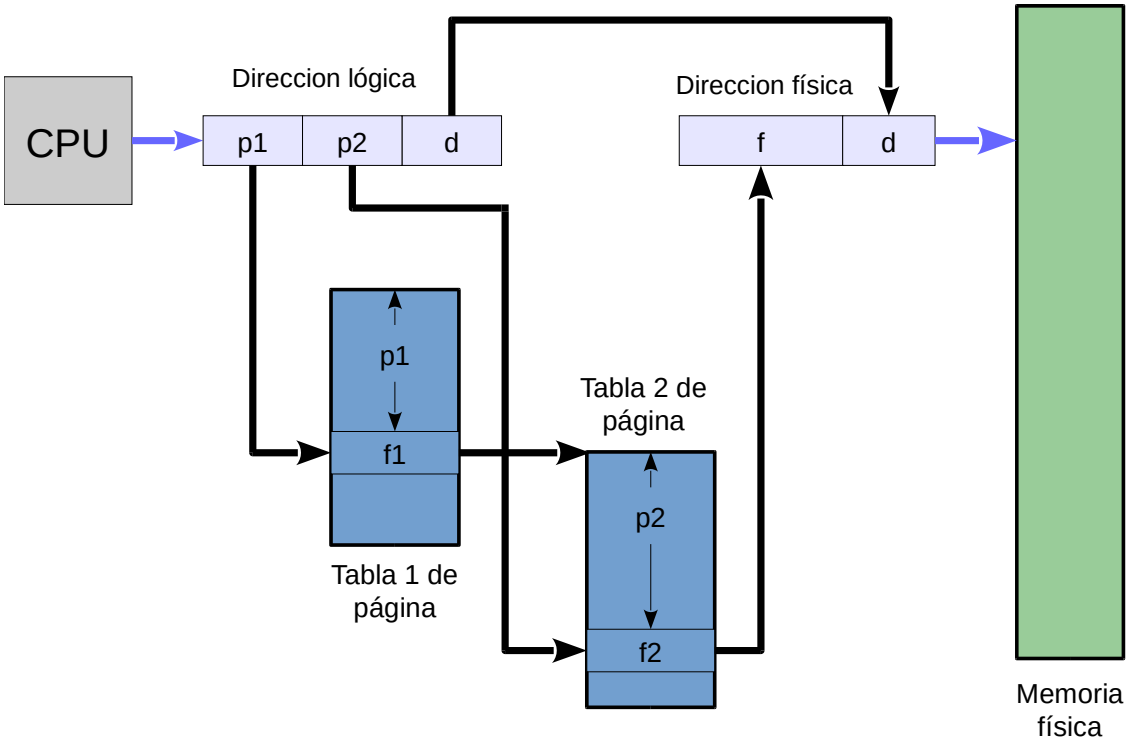
Manejo de memoria

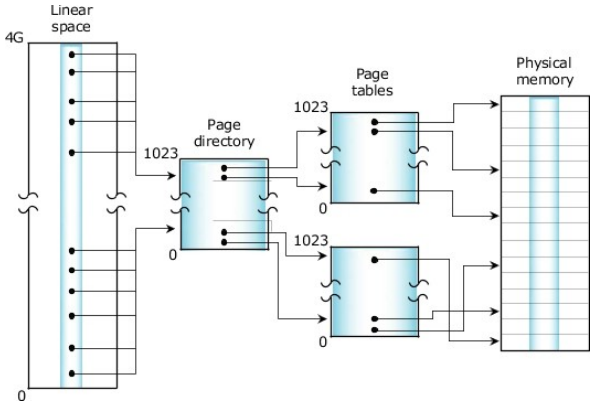
Paginación

El problema con las tablas de páginas es que pueden ser muy grandes.

Por ejemplo: *El espacio de direcciones de 32 bits con páginas de 4 KB implica que la tabla de páginas de un proceso tiene 1 millón de entradas.*

Es difícil encontrar una asignación contigua de al menos 1 MB para cada tabla de páginas.





Una solucion a este problema consiste en subdividir las tablas para que puedan alojarse en la memoria al permitir que se disemine en partes no contiguas generando esquemas de paginacion de multiples (*N*) niveles.

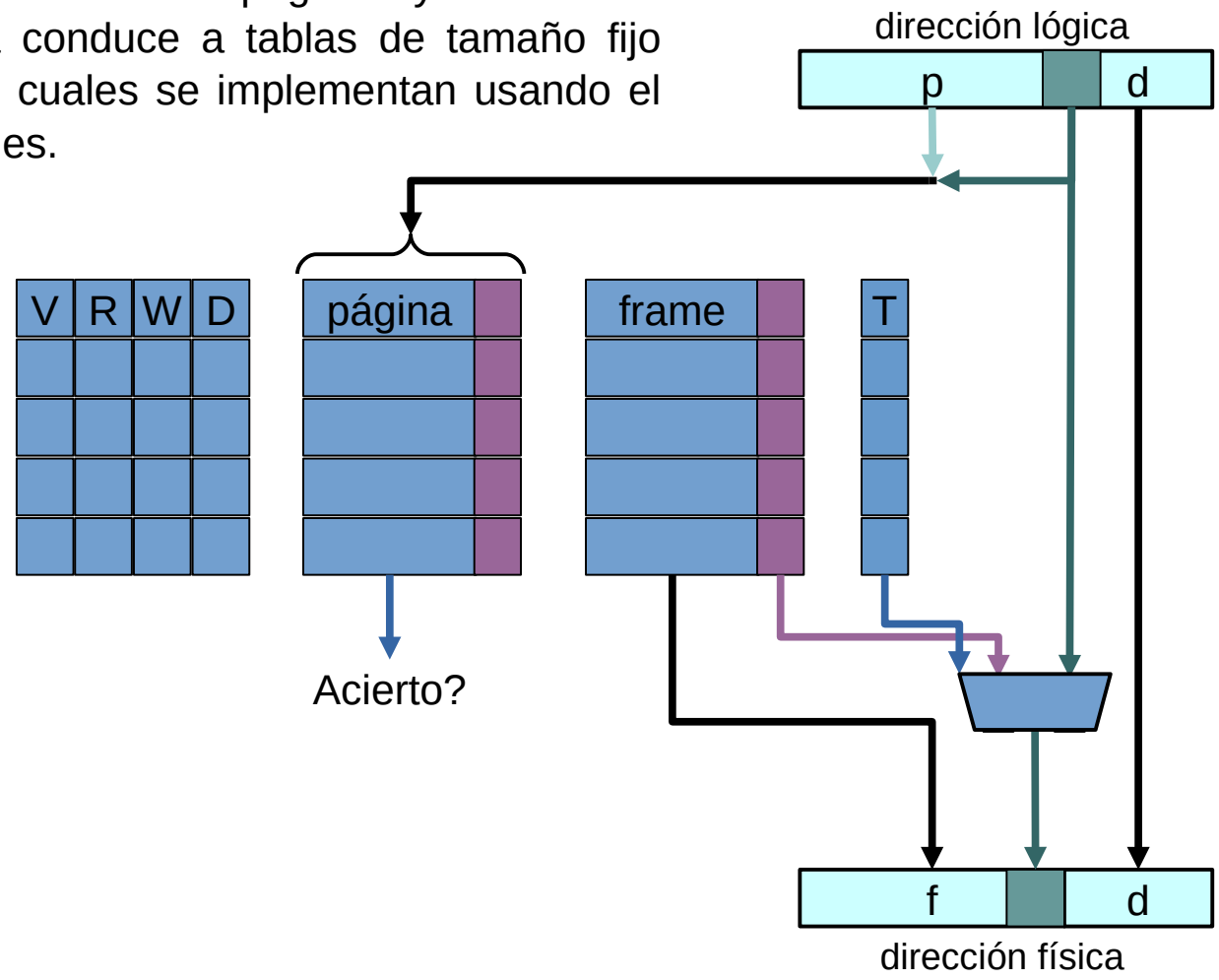
Manejo de memoria

Unidad de Manejo de Memoria

Otra solucion consiste en el uso de páginas y frames de tamaño variable. Esta idea conduce a tablas de tamaño fijo pero de ancho variable, las cuales se implementan usando el siguiente diagrama de bloques.

En este esquema, los bits más bajos de la dirección de la página p también se pueden utilizar como parte del desplazamiento d. diagrama de bloques.

Para resolver este problema se introduce un multiplexor, para seleccionar la fuente de información correcta, y bits extras (T) para indcar el tamaño de la página.



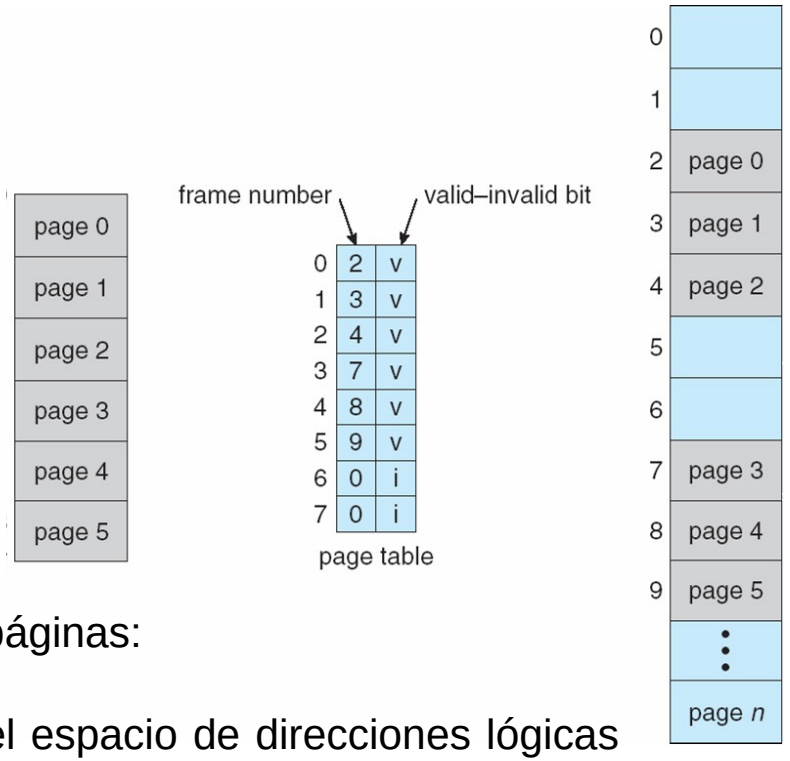
Manejo de memoria

Paginación

La protección de memoria se logra asociando los bits de protección correspondientes con cada frame.

El bit de proteccion se utiliza para indicar si sólo se puede leer; si se puede escribir; si se puede ejecutar; entre otros.

Normalmente, estos bits se almacenan en la tabla de página.



Estos bits se agregan a cada entrada de la tabla de páginas:

- ***valid*** indica que la pagina asociada es válida en el espacio de direcciones lógicas del proceso y por lo tanto puede ser utilizada;
- ***invalid*** indica que la página asociada no pertenece al espacio de las direcciones logicas del proceso.

Si el sistema tiene una **tabla de registros de longitud de páginas** puede verificar si la direccion es válida.

Cualquier error puede ser atendido a traves de una excepcion del procesador.

Manejo de memoria

Segmentación

La segmentacion es una tecnica de gestion de memoria que soprta la organización de la memoria desde la perspectiva del programador: una colección de unidades logicas que contienen diferentes tipos de informacion sin un orden preestablecido.

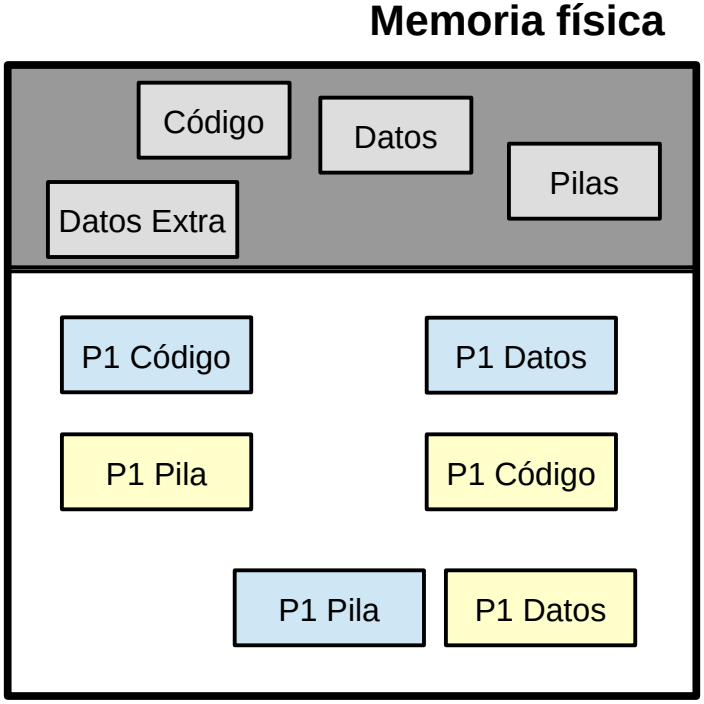
El espacio de direcciones lógicas es un conjunto de segmentos, cada uno de los cuales tiene un nombre y una longitud.

Para facilitar la implementación, los segmentos están numerados.

Una dirección lógica contiene dos tuplas:

```
<segment-number, offset>
```

Las direcciones identifican tanto el nombre del segmento como el desplazamiento dentro del segmento.



Cuando se compila un programa, el compilador crea automáticamente segmentos que reflejan el programa de entrada. Un compilador de C podría crear los segmentos de código; variables globales; heap y las pilas utilizadas por cada hilo y las biblioteca.

Manejo de memoria

Segmentación

Los registros que definen al segmento son accesibles al programador cuando solo el procesador opera en modo supervisor o a traves de excepciones.

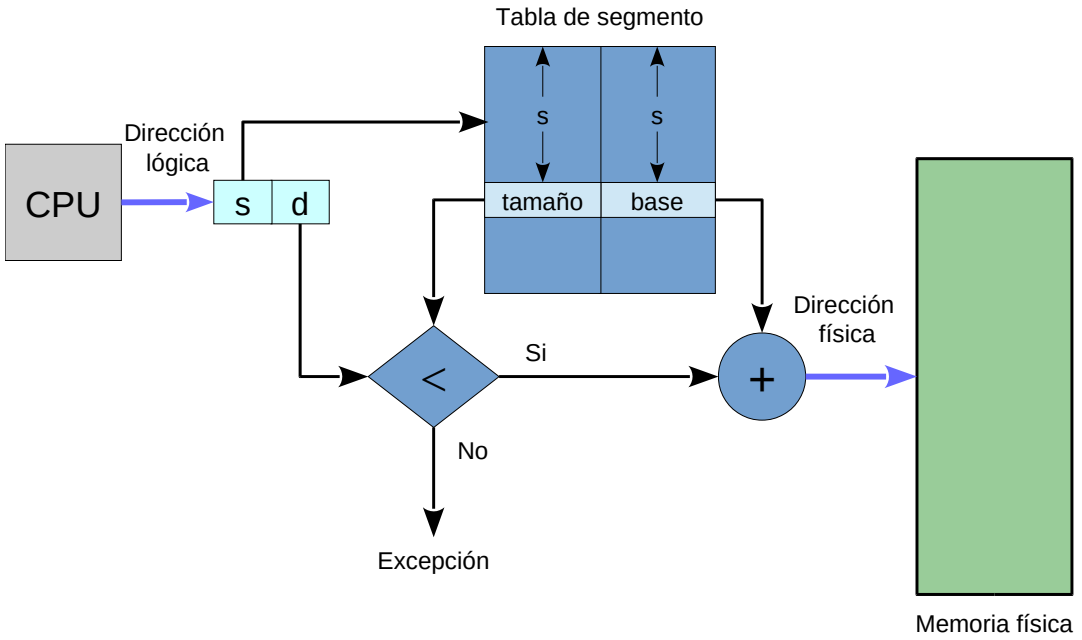
Estos registros estan almacenados en la tabla de segmento, la cual contiene dos datos:

- La direccion física de inicio del segmento (*base*); y
- El tamaño del segmento (*limite*)

Cada tabla de segmento esta definida a traves de

La **tabla de registros de base** de segmento (STBR), la cual apunta la ubicación en memoria de la tabla de segmentos; y

La **tabla de registros de tamaño** de segmento (STLR), la cual indica el número de segmentos utilizados por un programa.

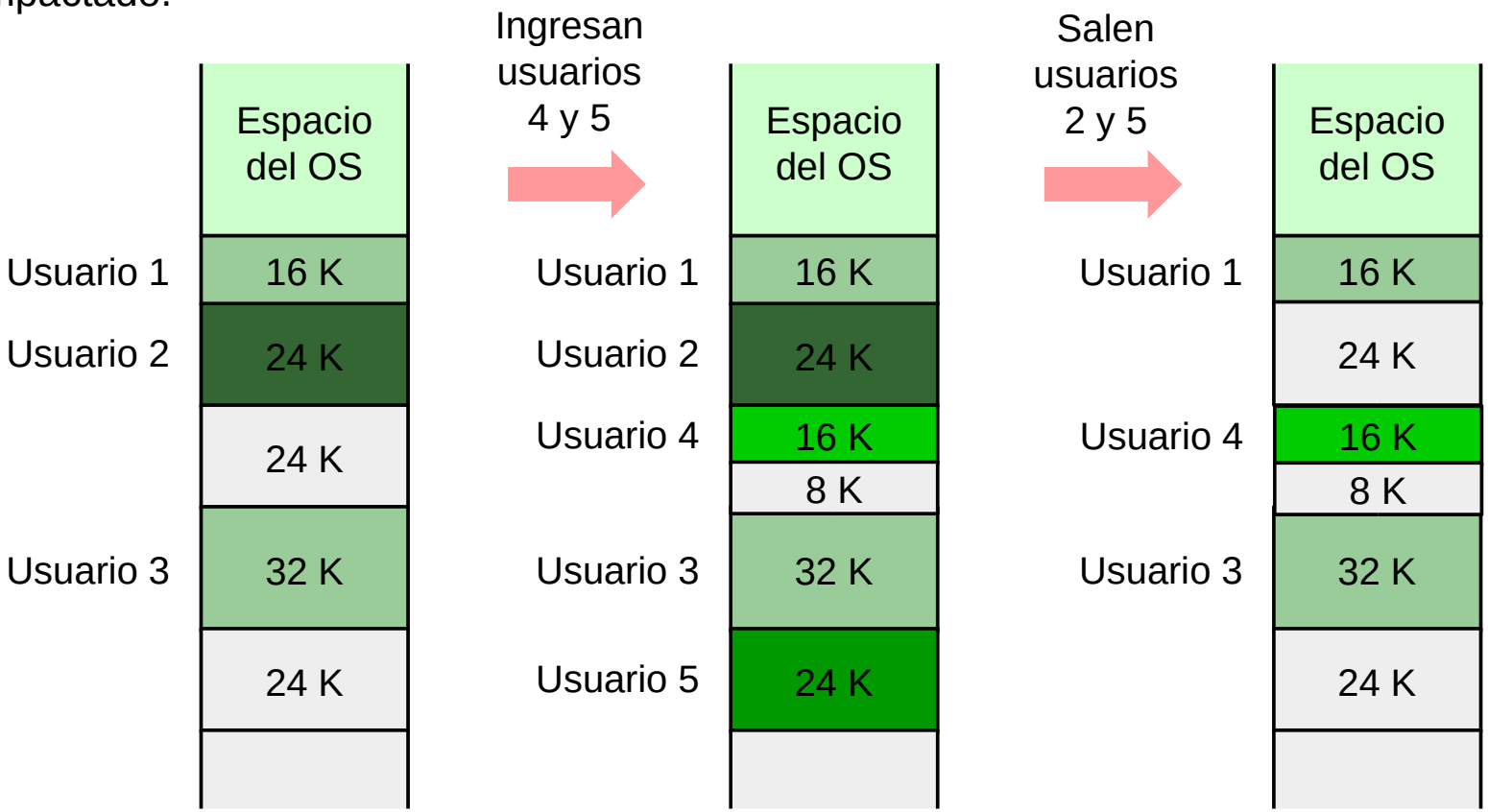


Manejo de memoria

Segmentación

Las ventaja de la segmentacion es que permite que el espacio de las direcciones fisicas de un proceso no sea contiguo.

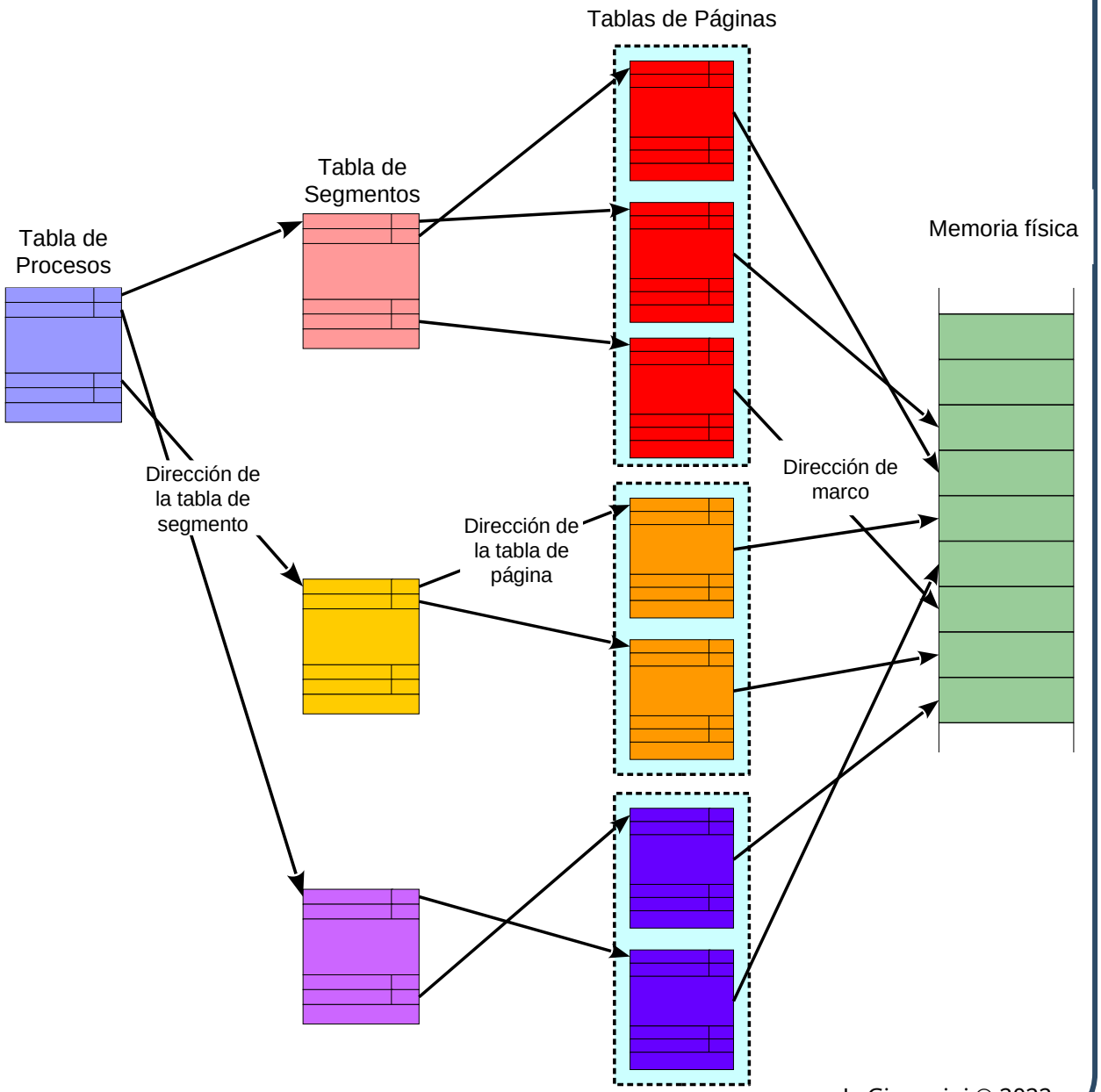
Las desventajas de la segmentacion es que sufre de **fragmentación** y necesita ser compactado.



Manejo de memoria

Segmentación

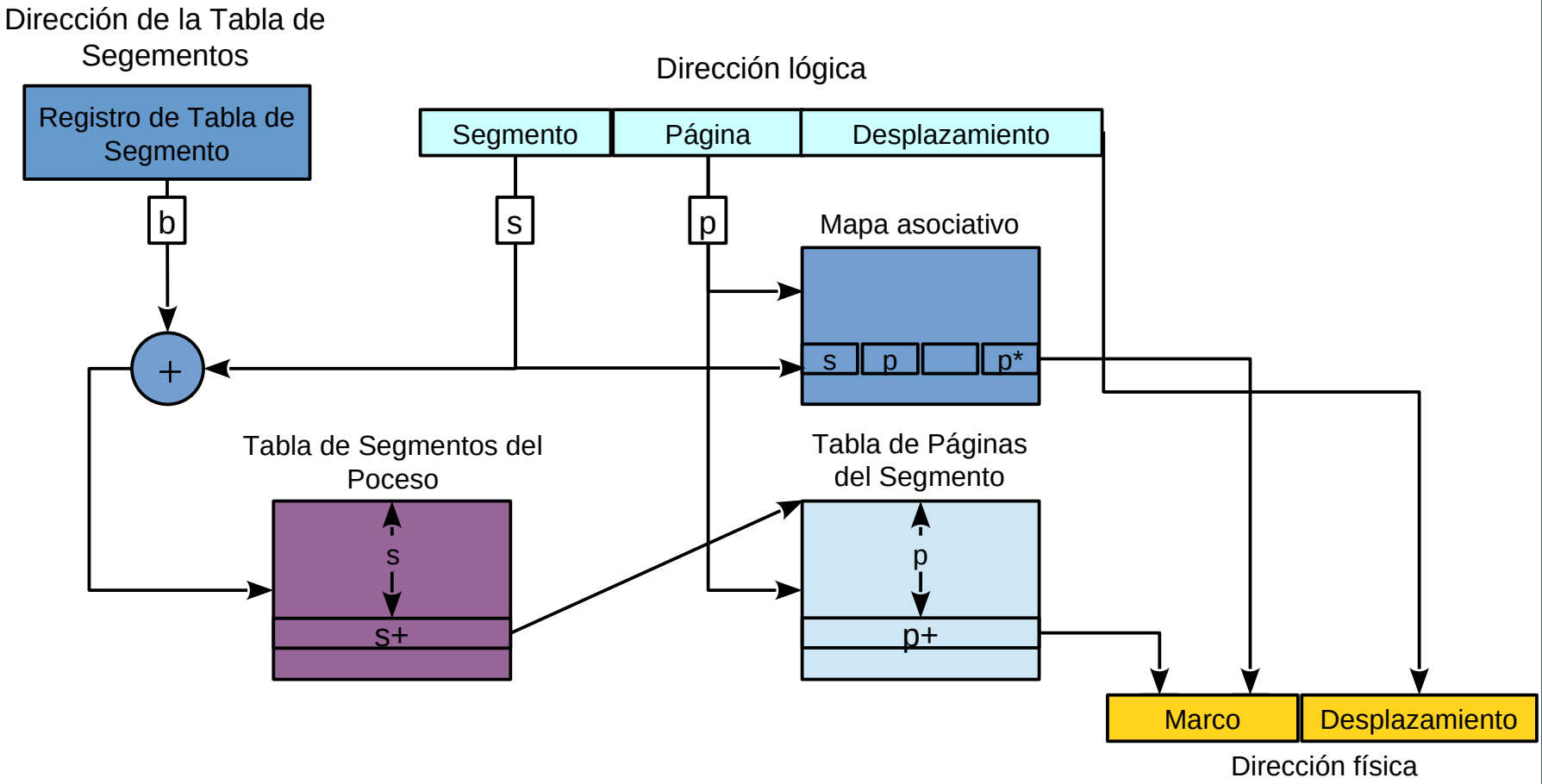
La estructura de tablas de un sistema de paginación y segmentación.



Manejo de memoria

Segmentación

Traducción de las direcciones lógicas con combinación de transformación asociativa/directa dentro de un sistema de segmentación y paginación



Manejo de memoria

Virtualizacion

La memoria virtual es una técnica de gestión de memoria que proporciona una "*abstracción idealizada de los recursos de almacenamiento que están realmente disponibles en una máquina*" que "*crea la ilusión de una gran cantidad de memoria.*"

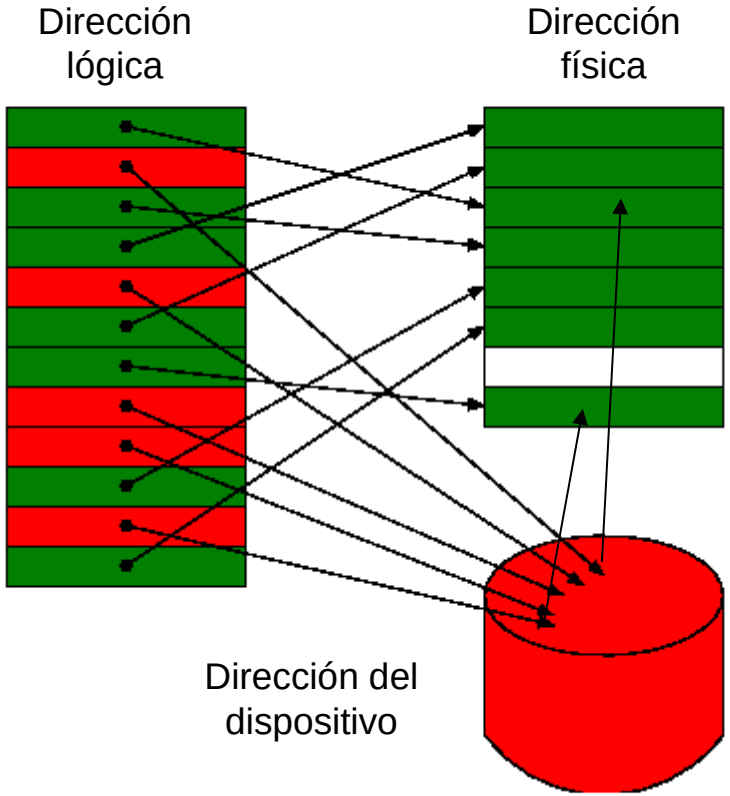
Utiliza una combinación de hardware y software para asignar las direcciones logicas utilizadas por un programa a direcciones físicas en la memoria de la computadora.

La memoria aparece, para los procesos, como un espacio de direcciones contiguas o una colección de segmentos contiguos.

El sistema administra los espacios de direcciones virtuales y la asignación de memoria real a la memoria virtual.

La MMU traduce automáticamente las direcciones logicas a direcciones físicas.

El sistema puede ampliar estas capacidades para proporcionar un espacio de dirección virtual que puede exceder la capacidad de la memoria real.



Manejo de memoria

Virtualizacion

Un proceso puede ser **intercambiado** (**swapped**) temporalmente entre la memoria y el medio de almacenamiento para su ejecución de acuerdo con los recurso disponibles.

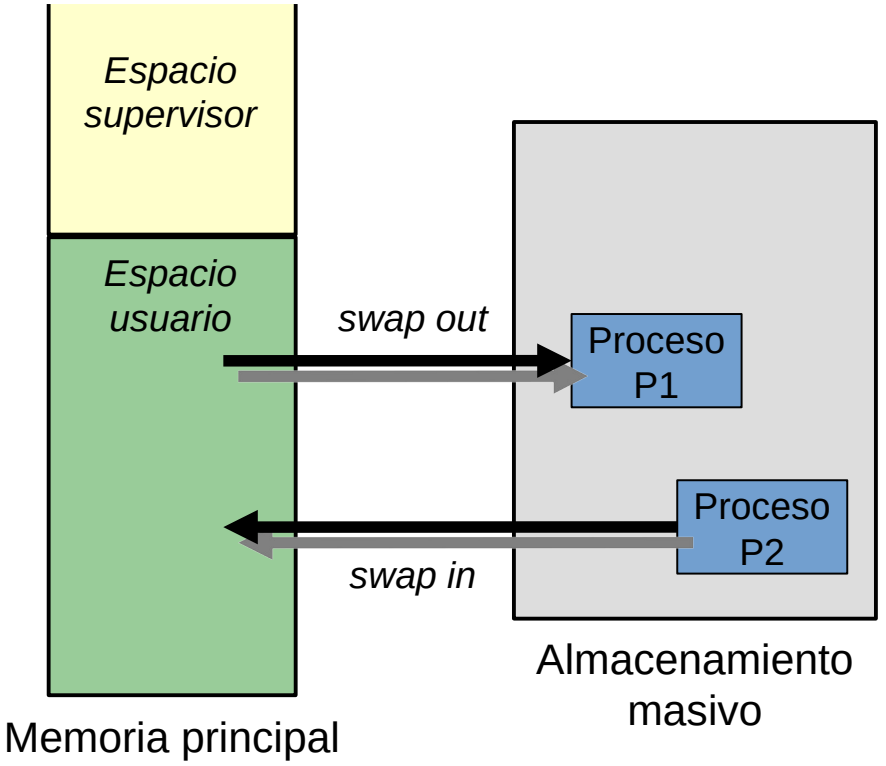
El sistema maintiene una **cola** de los proceso que estan siendo ejecutados en la memoria que tienen una imágen en el medio de almacenamiento.

Esto permite que:

- El espacio de memoria utilizado por un procesos exceda la memoria física; y
- Aumenta el número de programas que se pueden utilizar simultáneamente.

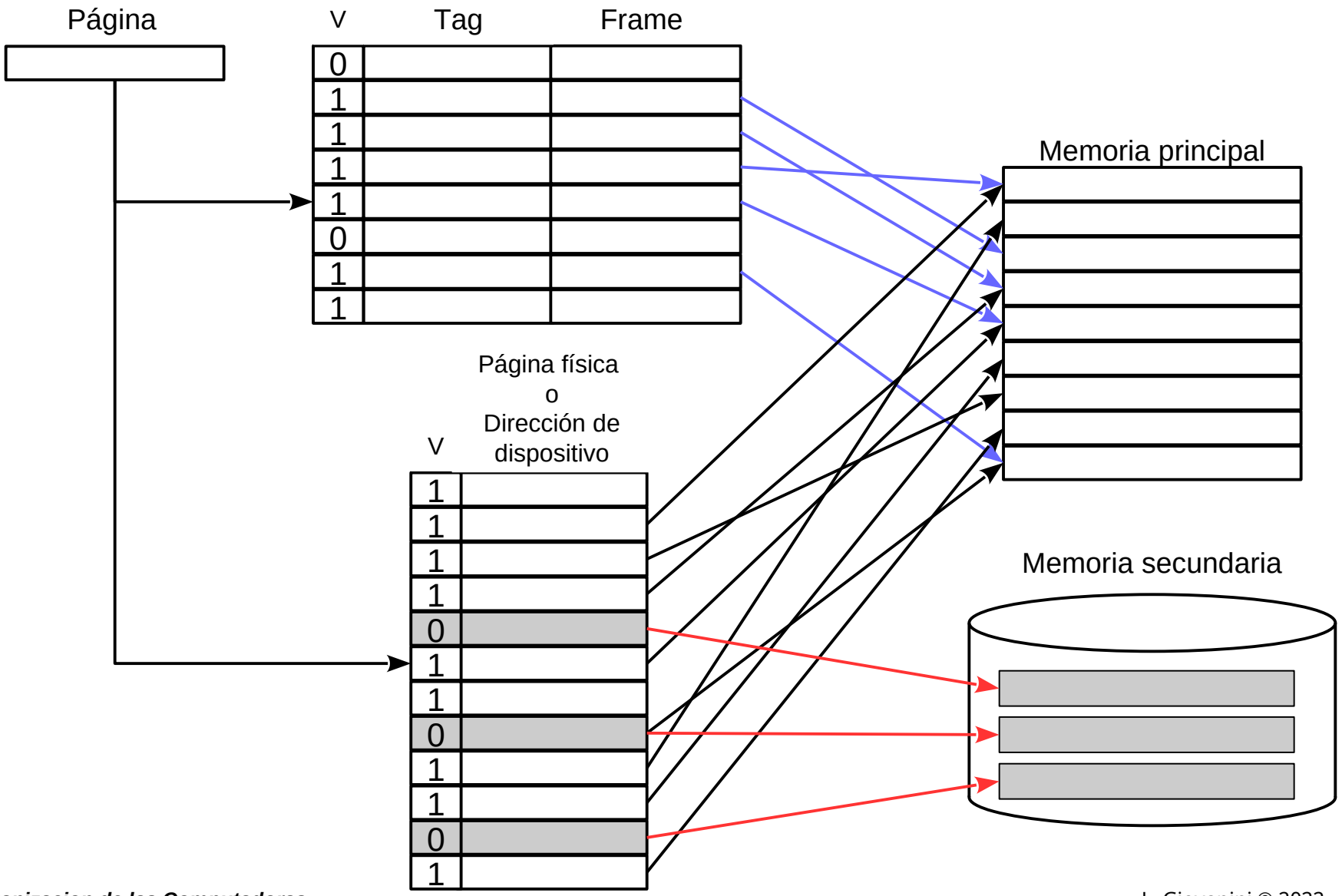
Sin embargo, tiene como **cuello de botella** la velocidad del almacenamiento secundario, por lo que requiere de

- Medios de almacenamiento veloces;
- Tamaño adecuado para almacenar todas la información y programas; y
- Permitir acceso a las copias de la imágen de memoria.



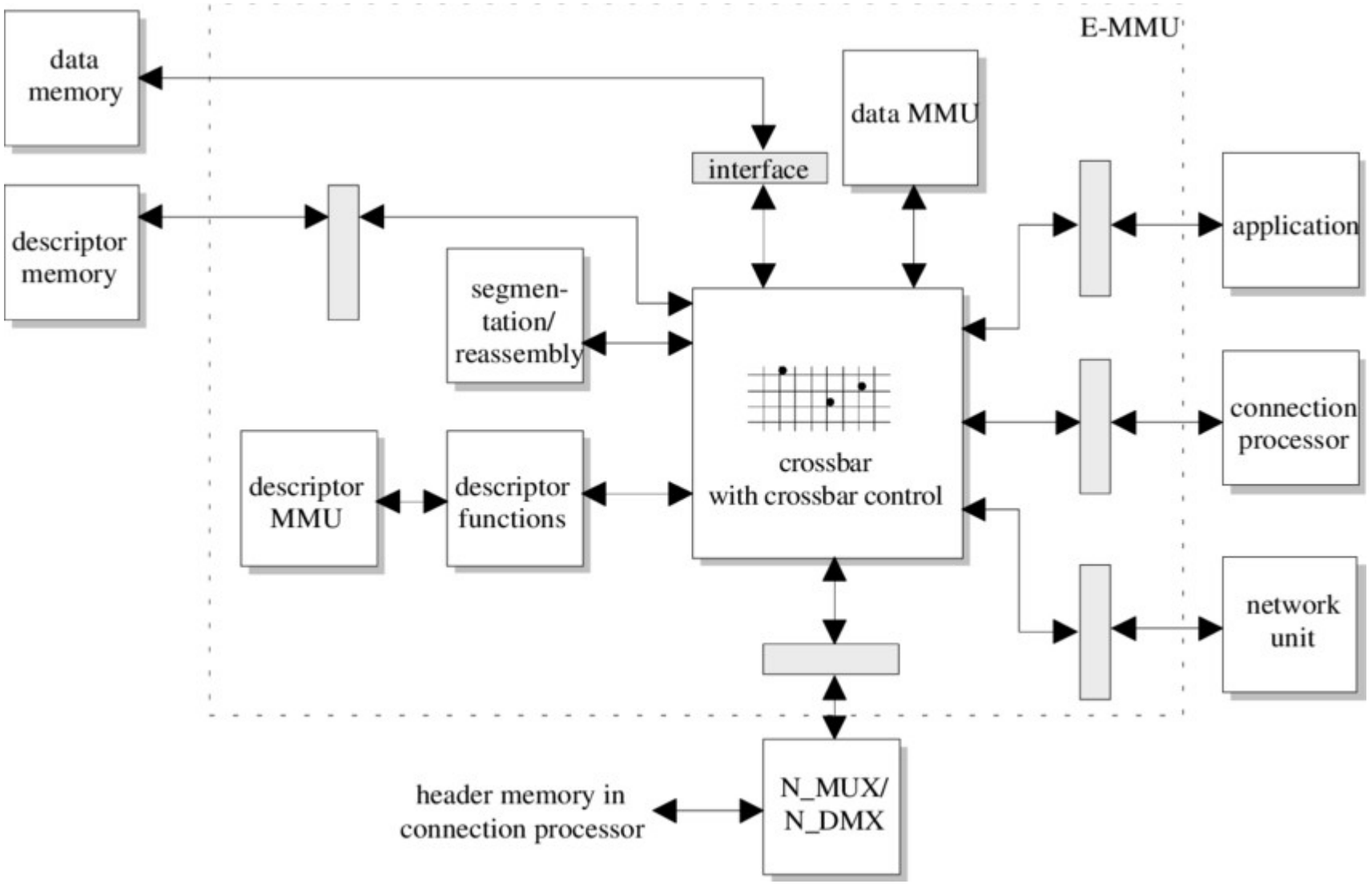
Manejo de memoria

Virtualizacion



Manejo de memoria

Virtualización



Unidad de manejo de memoria

Manejo de memoria

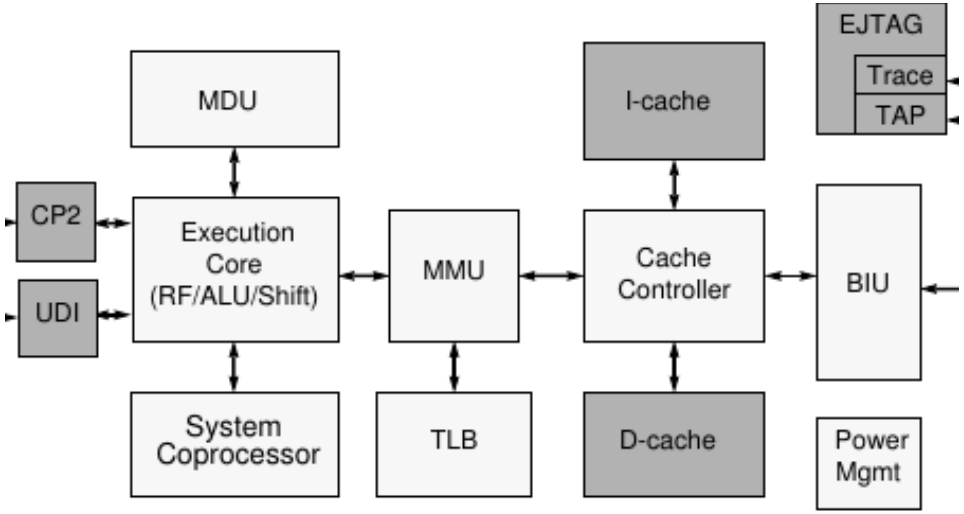
Unidad de Manejo de Memoria

La unidad de gestión de memoria es un dispositivo de hardware responsable del manejo de los accesos a la memoria por parte del procesador.

Las funciones de este dispositivo son la traducción de las direcciones lógicas (o virtuales) a direcciones físicas (o reales), la protección de la memoria, el control de caché y, en arquitecturas de computadoras más simples la conmutación de bancos de memoria.

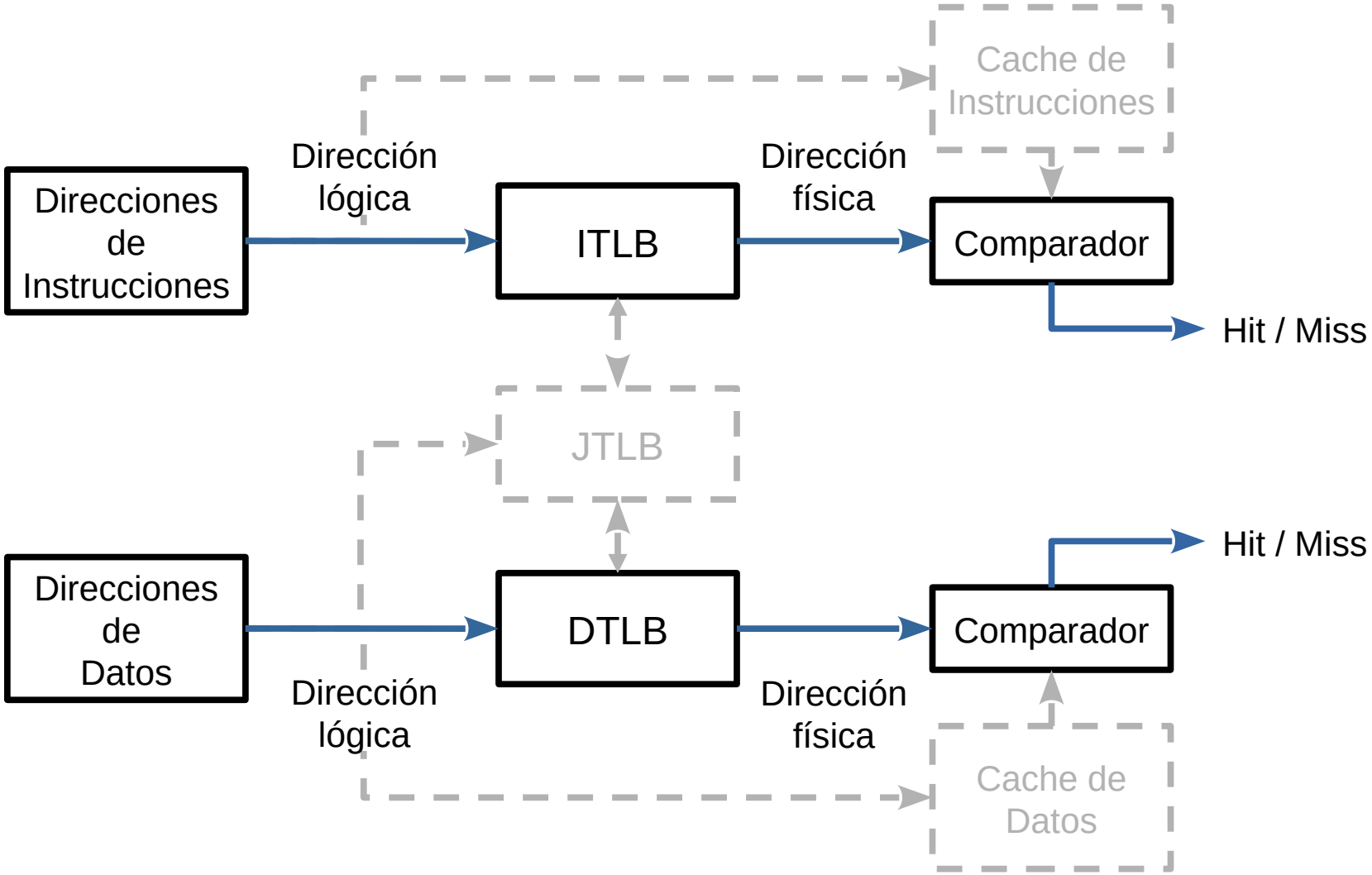
Un error de página puede indicar un error de software, que se puede prevenir usando la protección de la memoria como uno de sus beneficios: un sistema operativo puede usar la MMU para impedir el acceso no deseado a la memoria por un programa.

Una MMU también mitiga el problema de la fragmentación. Con la memoria virtual, un rango contiguo de direcciones virtuales puede asignarse a bloques no contiguos de memoria física.



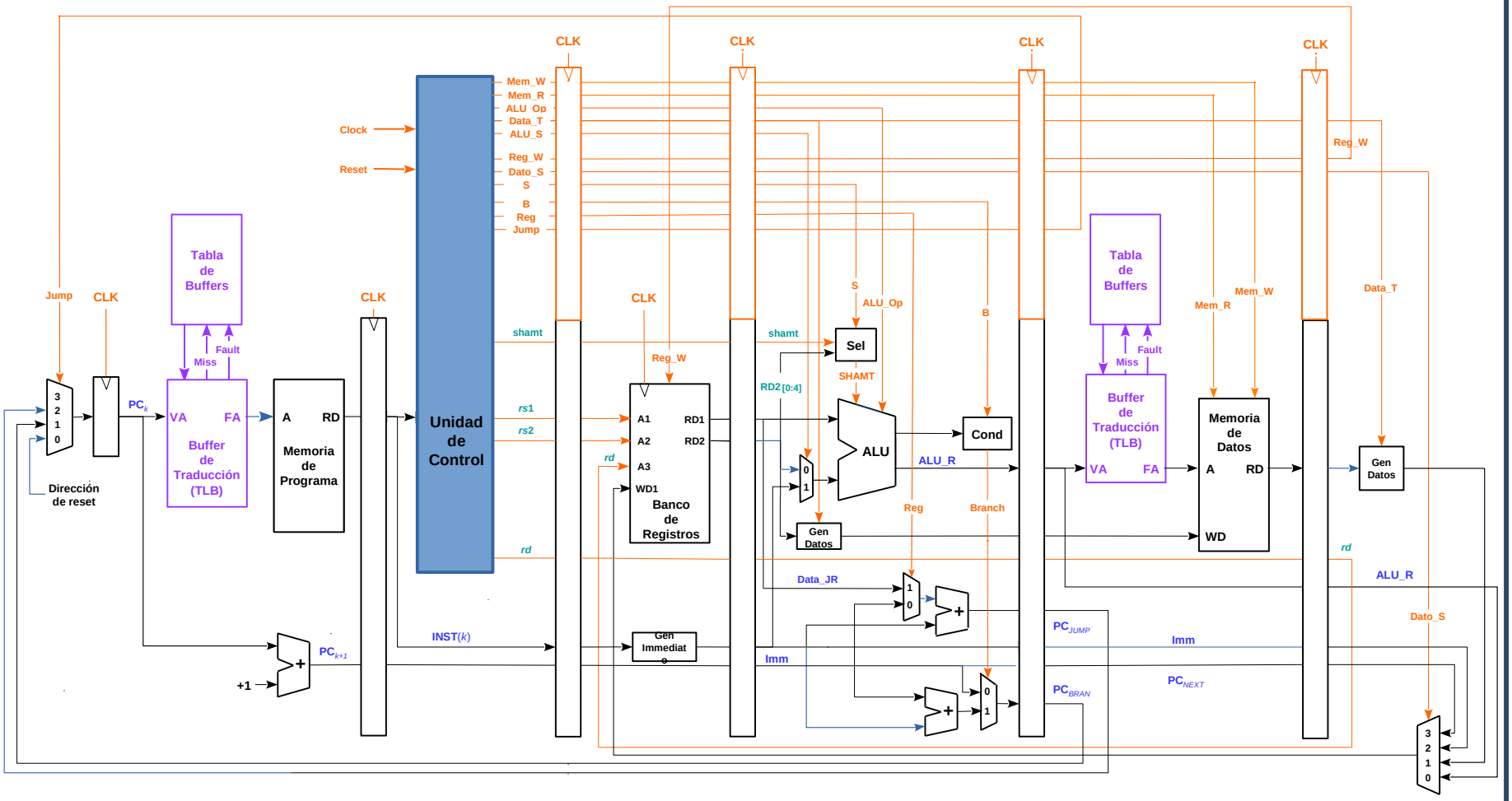
Manejo de memoria

Unidad de Manejo de Memoria



Manejo de memoria

Unidad de Manejo de Memoria



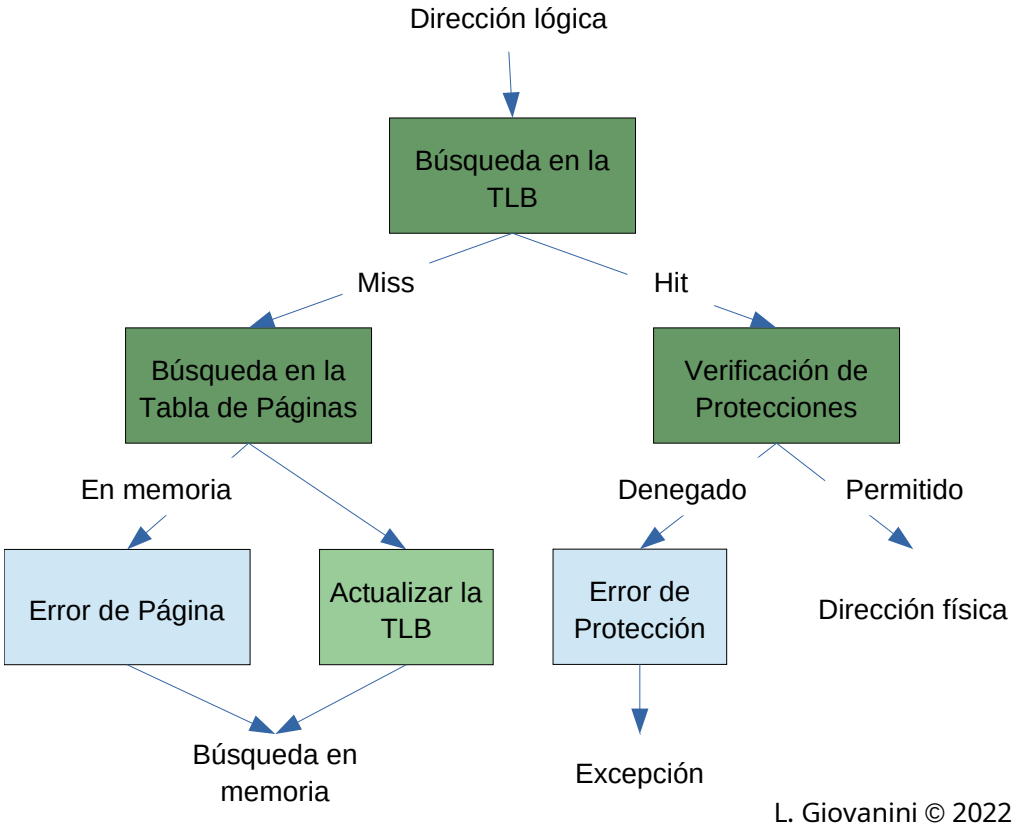
Manejo de memoria

Unidad de Manejo de Memoria

Los mecanismos para resolver errores en la operación de la TLB son

Software (MIPS, Alpha): El error de direccion debido a que no se encuentra la direccion buscada en la TLB genera una **excepcion** y el sistema operativo carga las nuevas TLB.

Hardware (SPARC v8, x86, PowerPC): La unidad de manejo de memoria carga automaticamente las nuevas TLB. Si la pagina buscada es encontrada durante el proceso de recarga, la MMU genera una excepcion para la instrucción original

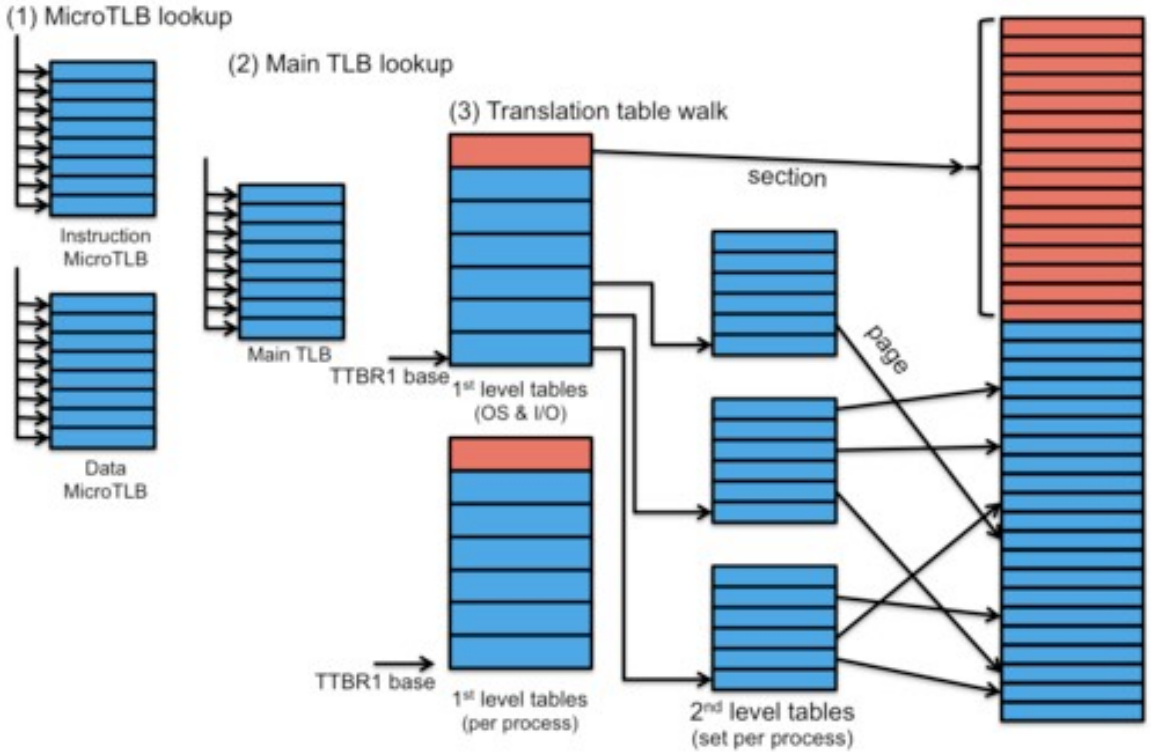


Manejo de memoria

Unidad de Manejo de Memoria - ARM

La MMU de la arquitectura ARM admite una jerarquía de dos niveles para su estructura de tabla de páginas, que depende de su tamaño ***supersecciones*** (bloques de memoria de 16 MB); ***secciones*** (bloques de memoria de 1 Mb); ***páginas grandes*** (bloques de memoria de 64 KB) y ***páginas pequeñas*** (bloques de memoria de 4 KB).

Una entrada en la tabla de primer nivel contiene un puntero a las tablas de segundo nivel o una dirección base de una sección o una supersección. Por lo tanto, si se utilizan páginas muy grandes, no tenemos que pasar por dos niveles de jerarquía.



La ventaja de las secciones y las supersecciones es que puede tener una gran región de memoria, como el sistema operativo, asignada con una sola entrada en la TLB.

La MMU se puede configurar para usar páginas pequeñas o grandes, así como secciones o supersecciones. Las secciones (o supersecciones) se pueden mezclar con las páginas

Manejo de memoria

Unidad de Manejo de Memoria - ARM

Cada acceso a la memoria se verifica con los permisos almacenados dentro tabla de páginas de cada bloque de memoria. Además, la entrada de la tabla de páginas también puede especificar cómo se comporta una región de la memoria con respecto al almacenamiento interno en caché, de la visibilidad y modificaciones a otros núcleos.

Las regiones de memoria pueden tener los siguientes atributos:

- **Execute never** – no permite las instrucciones tengan acceso a esta región de la memoria;
- **Read-only, read/write, no access** – estos modos configuran tanto el modo de usuario como el modo privilegiado. Por ejemplo, la memoria del kernel se puede etiquetar como sin acceso para el modo de usuario pero de lectura / escritura para la ejecución del modo de kernel.
- **Non-secure** – identifica la memoria como parte de una región confiable;
- **Sharable** – identifica si una región de la memoria se comparte con otros procesadores o se asigna a dispositivos. Varios modos están disponibles:
  - *Strongly ordered* – los accesos de memoria deben ocurrir en el orden del programa;
  - *Device/shared o device/non-shared* – a memoria se asigna a un dispositivo y el dispositivo se comparte o no con otros procesadores en el bus; y
  - *Normal/shared o normal/non-shared* – memoria regular que es compartida o no con otros procesadores en el bus.

Si el permiso no es válido para el acceso, la MMU atrapa con una señal de cancelación de memoria al procesador.

Manejo de memoria

Unidad de Manejo de Memoria Intel

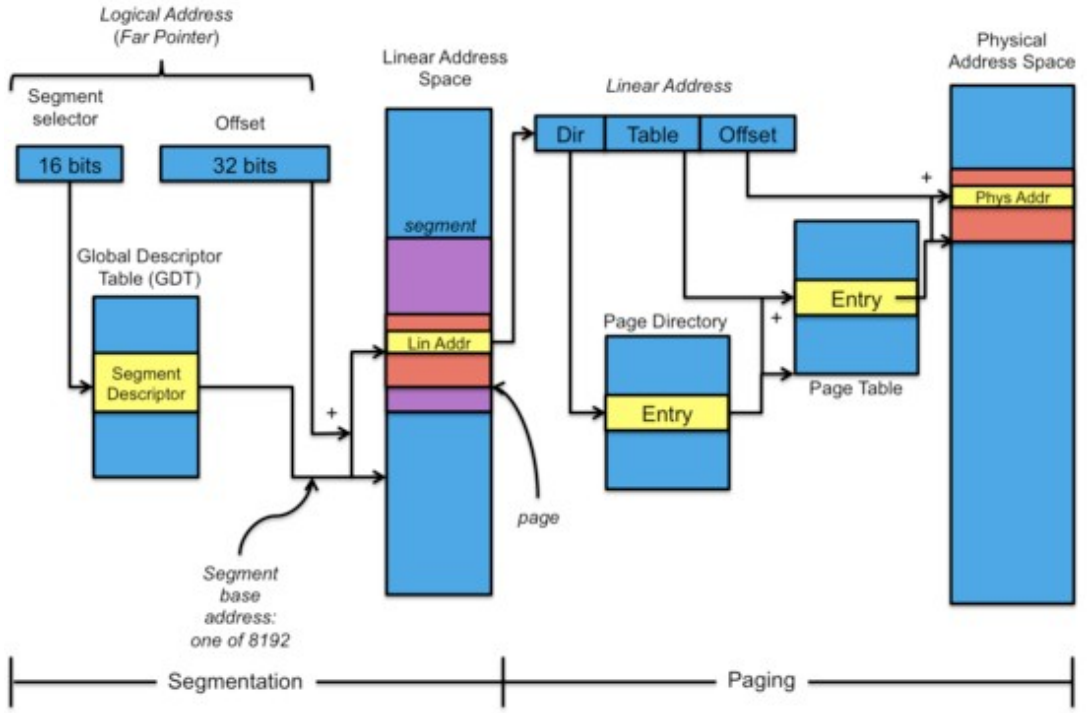
La arquitectura IA-32 admite un modelo combinado de segmentación y paginación. Hay dos tablas de segmentos que contienen las direcciones base de los segmentos:

- La tabla de descriptores locales (LDT) contiene información de segmento y es privada; y
- La tabla de descriptor global (GDT) contiene información de segmentos que se comparte entre todos los procesos.

El selector de segmento es un índice en estas tablas. Para seleccionar un segmento, el selector se carga en un registro de segmento. Además, cada instrucción que hace referencia a la memoria tiene un registro de segmento implícito, que puede ser anulado agregando un prefijo de segmento antes de la instrucción de la máquina.

Este modelo combinado utiliza la segmentación para generar una dirección lineal, la cual se trata como una dirección virtual y se traduce mediante la lógica de paginación de la MMU a través de una jerarquía de tablas de páginas de dos niveles.

La lógica de paginación se puede desactivar si no es necesaria.



Manejo de memoria

Unidad de Manejo de Memoria - Intel

Cada entrada de la GDT contiene la dirección base del segmento y banderas que definen los atributos del segmento

- **S** – esta bandera determina si el segmento contiene código o datos;
- **Accessed** – esta bandera determina si el segmento ha sido accedido desdela última vez que el sistema operativo la reseteo;
- **Dirty** – esta bandera determina si el segmento ha sido modificado desdela última vez que el sistema operativo la reseteo;
- **Data/write-enable** – esta bandera determina si el segmento es de solo-lectura o lectura-escritura;
- **Data/expansion direction** – esta bandera determina el lugar donde se memoria al segmento cuando se incrementa su tamaño al cambiar el límite;
- **Code/execute-only or execute/read** – esta bandera determina si la información contenida en un segmento de código puede ser accedida como datos o sólo ejecutada;
- **Conforming** – esta bandera indica si el código contenido en el segmento puede continuar su ejecución cuando el nivel de privilige se eleva.