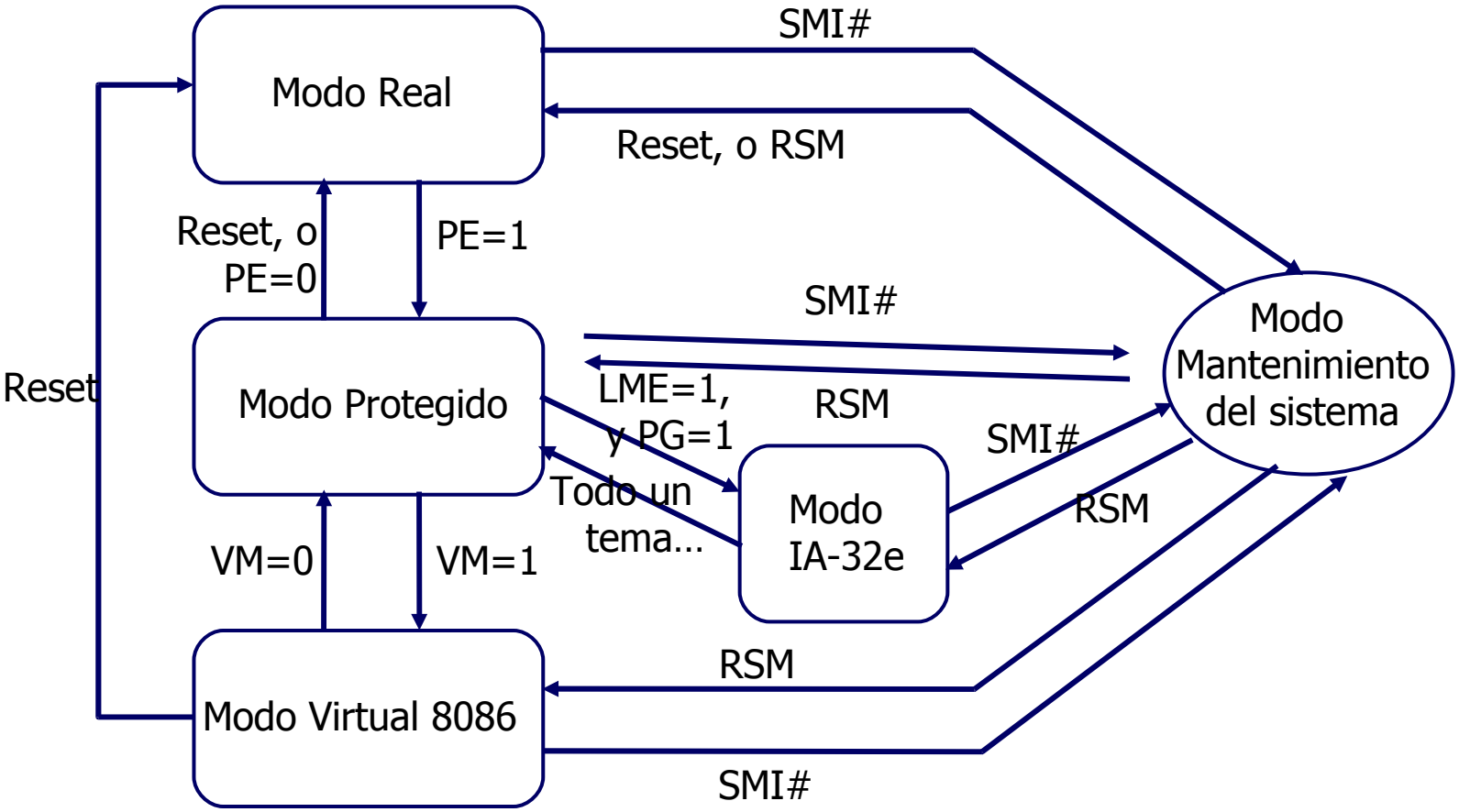
Arquitectura IA-32

Autor:Alejandro Furfaro

Modos de Funcionamiento



Como vemos nuestra PC?

- Nuestra PC es un entorno sumamente amigable (por lo general :))
- Como usuarios finales resolvemos la mayor parte de los problemas...
  - Usando Wizards
  - Clickeando sin mas en un ícono
  - De vez en cuando pulsando el botón de reset... :(
- Como programadores de aplicaciones resolvemos nuestras necesidades
  - Pidiendo recursos vía System Calls (malloc, fopen, free, printf, scanf, etc.)
  - Enviando requerimientos para acceder a la E/S
  - Usando librerías de código que nos facilitan la vida

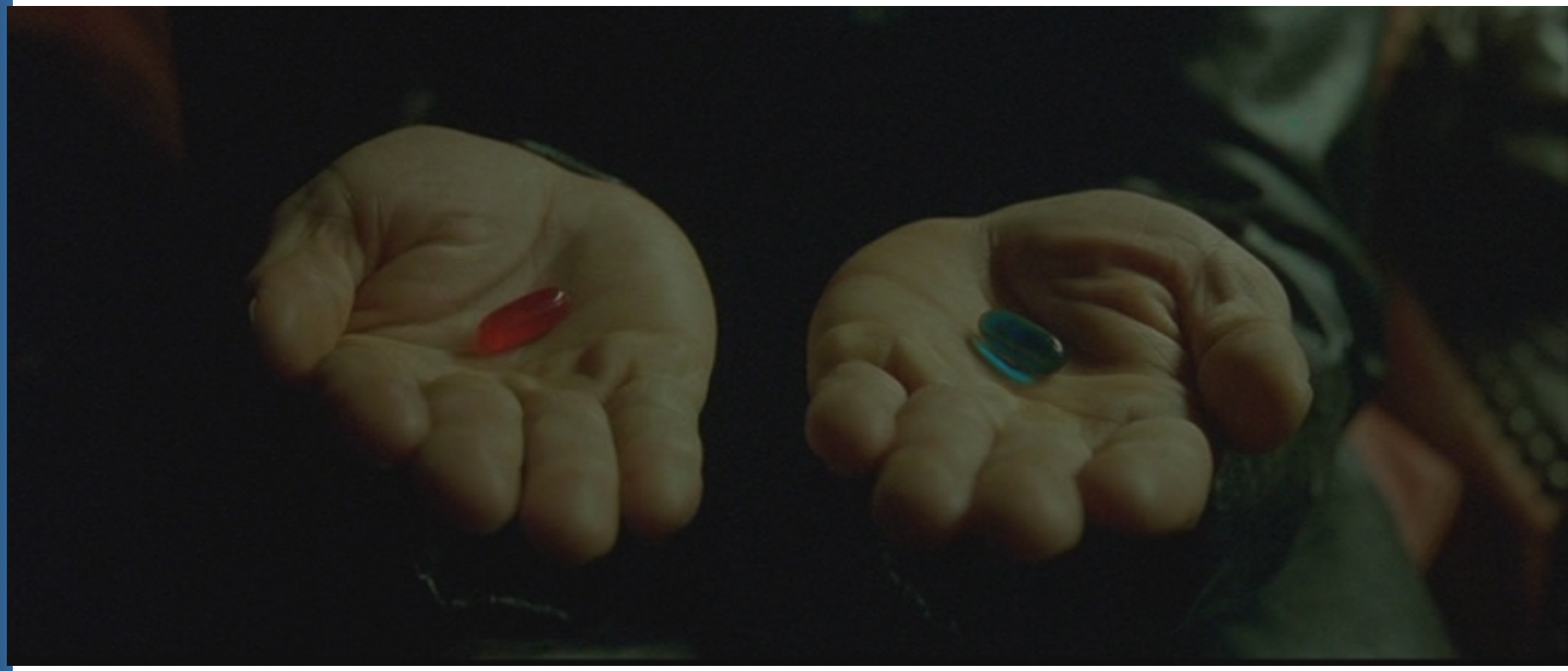
Pero... ¿Como se logra todo esto?

- Simple.
- Un grupo de programadores con profundo conocimiento del procesador y de los detalles de hardware del sistema, construyen una pieza de código fundamental llamada Sistema Operativo.
- Sobre el sistema operativo se sostienen todos los recursos de software que hacen la vida mas fácil a los usuarios y a los programadores de aplicaciones.

Pero al final de cuentas.....

¿Que es el Modo Protegido???

¿Se acuerdan?



The choice

- O nos quedamos con el wizard que nos resuelve la vida sin tener que pensar...
- O nos decidimos a enfrentar las cosas como son realmente, y entenderlas, aprendiendo a, si es necesario, hacer todo desde cero y a pulmón.

¡¡¡¿¿Y que hicimos nosotros para merecer esto???!!!

- Y.....Para empezar nos inscribimos en una carrera que entre otras cosas estudia las plataformas de hardware y sus posibilidades (todas sus posibilidades!!). Es decir... elegimos la píldora roja :)
- La construcción de un Sistema Operativo es a veces una tarea gigantesca, como Linux, pero en ocasiones puede requerir un número mucho menor de rutinas que, aunque de muy bajo nivel, provean un conjunto de recursos base suficientes para administrar un sistema de menor tamaño, como un embeeded system.

FAQ's

- FAQ#1: ¿Y acaso un embedded system no puede hostear Linux?.
  - Por supuesto.
  - µCLinux y otras implementaciones son la implementación de Linux para Micro Controllers.
  - PC's embedded basadas en procesadores IA-32 pueden aceptar un Linux cualquiera.
- FAQ#2: ¿Para que necesitamos saber esto entonces? ¿No está todo hecho?
  - Si Torvalds lo hubiese pensado de esta forma hoy solo existiría Windows como alternativa para nuestra PC.
  - Y si estas cosas no se estudian en esta carrera ¿donde?
- Conclusión: Hay que dominar la arquitectura del procesador para la programación de sistemas.

En suma.... Modo Protegido es:

- El conjunto de recursos de hardware que se requieren para esta difícil tarea y sus reglas de funcionamiento.
- Su dominio permite entender como funcionan las cosas en el mundo real.
- Y ese es nuestro trabajo
- En definitiva... Es la píldora roja.

Como se diseña un sistema de Protección

- El diseño del sistema de Protección en cualquier procesador se basa en los requerimientos de los sistemas operativos modernos
- Estos parten de considerar la existencia de diferentes tareas o procesos correspondientes a diferentes usuarios, todas coexistiendo en el mismo sistema de cómputo.

Requerimientos de los Sistemas Operativos Modernos

- Área de memoria exclusiva para cada tarea para almacenar su código y sus datos. (Área Local)
- Área de memoria común a todas las aplicaciones, para que éstas puedan acceder a datos globales del sistema, o a código propio del Sistema Operativo de modo de permitir la comunicación entre las aplicaciones. (Área Global).
- Cada tarea podrá acceder únicamente a su Área Local y al Área Global, pero nunca podrá acceder al Área Local de otra tarea. De este modo el Sistema Operativo garantiza la integridad (PROTECCION ;)) del código y de los datos propios de cada tarea

Requerimientos de los Sistemas Operativos Modernos

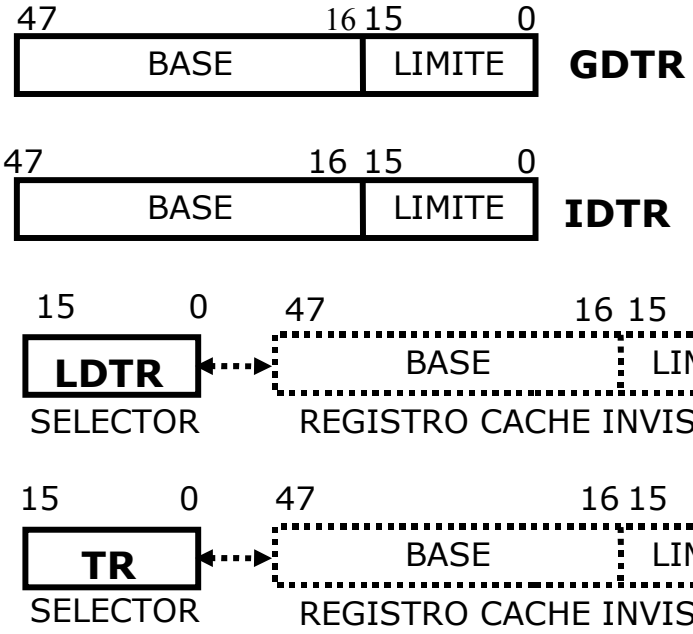
- Alta velocidad de procesamiento
- Gran capacidad de Direccionamiento de memoria
  - Amplio espacio de direccionamiento para memoria RAM
  - Capacidad de Gestión de memoria de cada tarea por el método de Memoria Virtual
- Capacidad de implementar Multitarea de manera rápida y segura.
- En cada momento la CPU ejecuta una tarea de la lista que mantiene, poniendo a su disposición todos los recursos de hardware de la máquina, incluyendo la cantidad de memoria requerida por la aplicación.

¿Como enfrentar el resto de esta presentación?

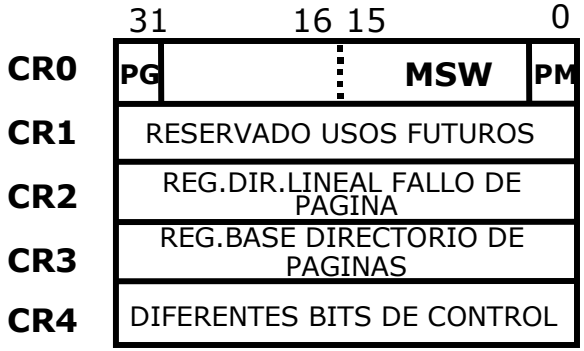
Dejando de pensar como un programador de aplicaciones, y comenzando a pensar como el programador de un sistema operativo.

Visión de la arquitectura para el programador de Sistemas Operativos

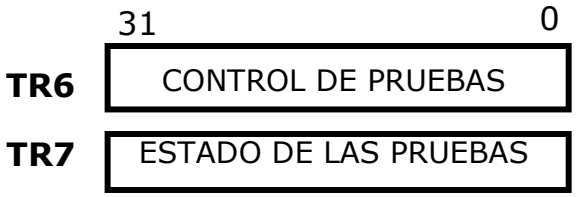
Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3



REGISTROS DE DEBUGGING



REGISTROS DE CONTROL

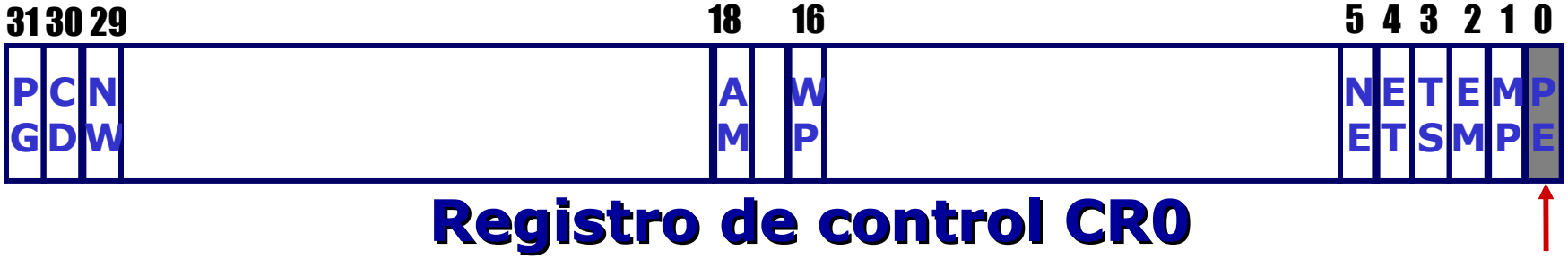


REGISTROS DE PRUEBA DE LA TLB

Ingreso al entorno de protección

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3

Para poner al procesador en Modo Protegido todo lo que se requiere es setear el bit PE, (bit 0 del registro CR0).



Registro de control CR0

Es un lamentable error considerar que esto es todo lo que hay hacer.

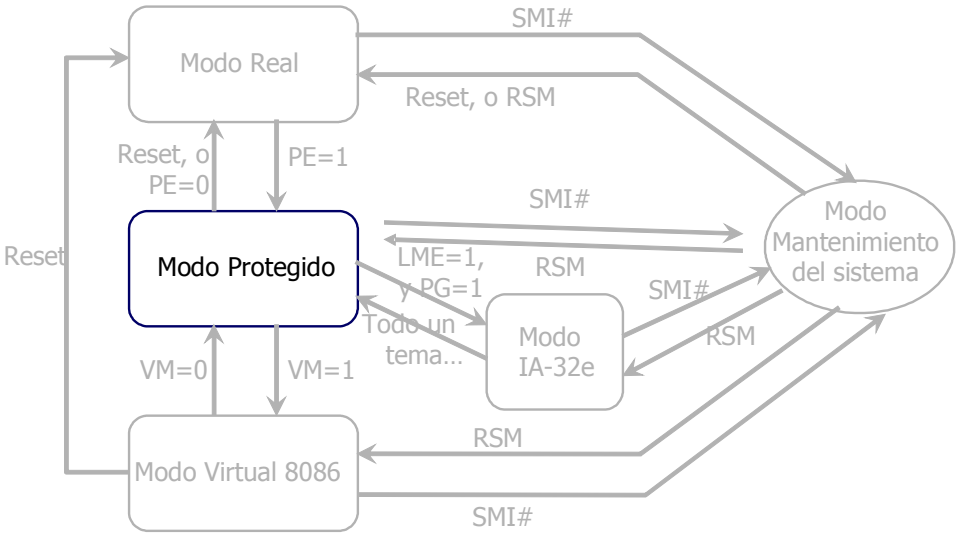
Setear PE es solo el principio. De hecho el siguiente programa funciona sin problemas. Veamos...

Ingreso al entorno de protección

```
use16
org 100h
inicio:
    cli
    mov eax,cr0   ;lee el registro de control 0.
    or al,1       ;setea el bit PE Protection Enable.
    mov cr0,eax;pasa a modo protegido.
    jmp short $+2;vacía la cola de ejecución.
mp:    in al,60h   ;lee el teclado
    dec al        ;compara con esc
    jnz mp        ;sino es esc sigue
    mov eax,cr0
    and al,0feh    ;resetea el PE.
    mov cr0,eax   ;retorna a modo real.
    jmp short $+2;vacía la cola de ejecución.
    sti       ;habilita las interrupciones.
    mov ah,4ch    ;retorna al dos con el
    int 21h       ;servicio 4ch de la int 21h.codigo.
```

Modo Protegido

Gestión de la memoria



Acceso a la memoria en MP

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3

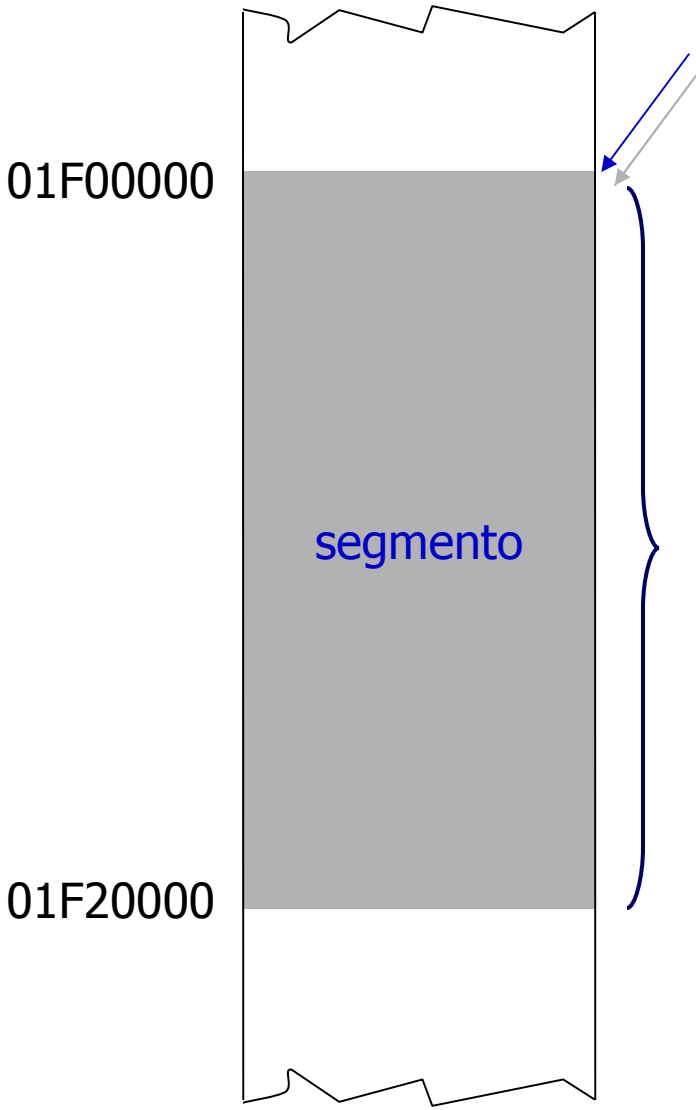
Para acceder a la memoria, los programas seguirán trabajando con segmentos, del mismo modo en que lo hacen en Modo Real.

La diferencia está en la información que se necesita en un entorno protegido para definir un segmento:

- Dirección a partir de la cual comienza el segmento. La llamamos **Dirección Base**.
- Tamaño del segmento. Intel lo denomina **Límite**.
- Permisos de acceso al segmento: Read Only, Código/Datos, y demás características que ahora serán rigurosamente chequeadas. Los denominaremos **Atributos.**

Acceso a la memoria en MP

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3



Dirección Base = 01F00000h

Tamaño = 20000h (128 Kbytes)

Para definir (o **describir**) un segmento en Modo Protegido se necesita un número de :

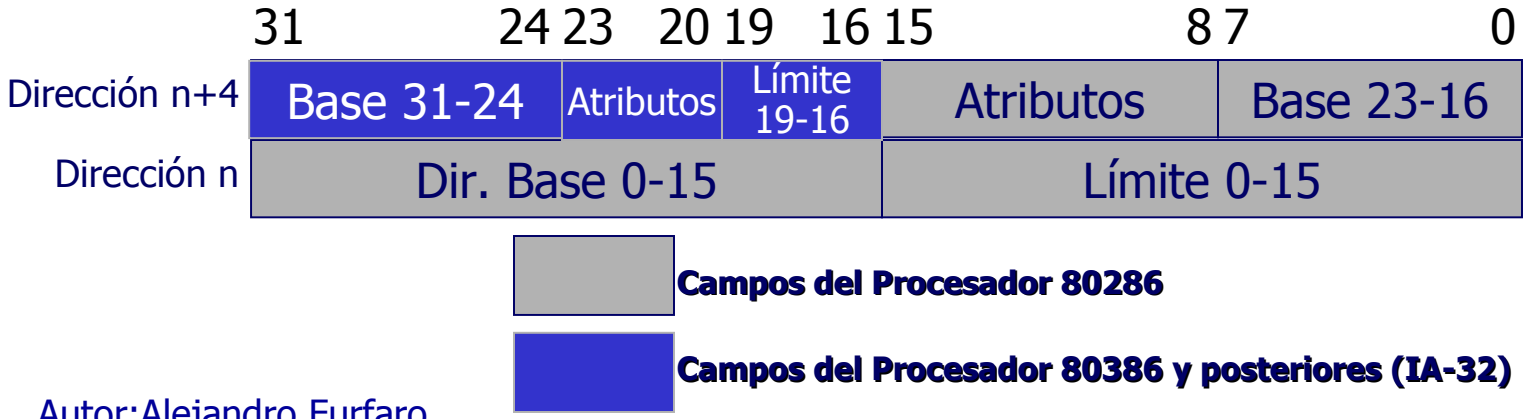
32 bits para la dirección base
+ 32 bits para el tamaño
n bits para los atributos.....

> 64 bits!!!!!

Descriptores de Segmento

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3

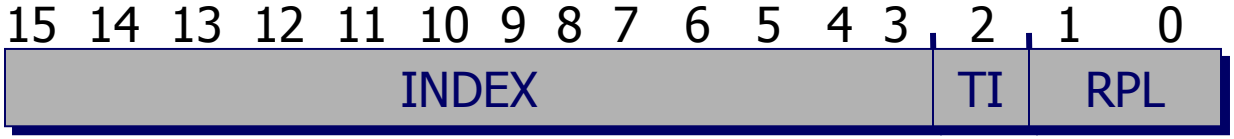
- En Modo protegido, la información necesaria para definir (describir) un segmento se almacena fuera del procesador en la memoria RAM.
- La estructura que define un segmento se denomina descriptor.
- Los descriptores se agrupan en tablas.



Selectores de Segmento

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3

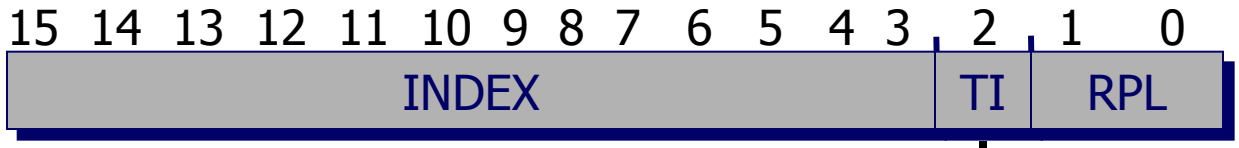
- En modo real los registros de segmento contenían toda la información necesaria para definir el segmento.
- En Modo Protegido los registros de segmento siguen conteniendo información necesaria para acceder al segmento, pero esa información se utiliza ahora para ubicar en la memoria del sistema, al descriptor del segmento a acceder.
- Por tal motivo al contenido de un registro de segmento, en Modo Protegido se lo denomina **Selector**.
- Esta denominación se extiende al Modo Real ya que siempre un registro de segmento "**selecciona**" con su contenido a un segmento determinado en memoria.



Formato de un Selector

Tablas de descriptores: GDT y LDT

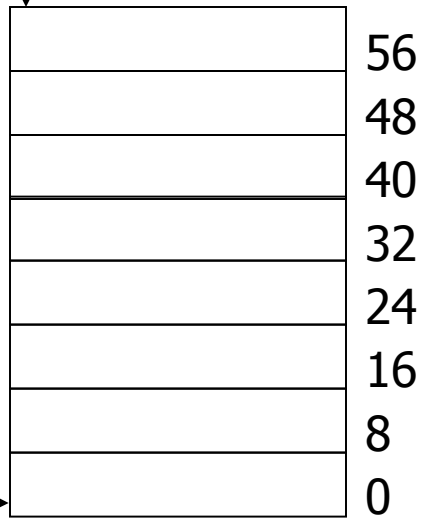
Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3

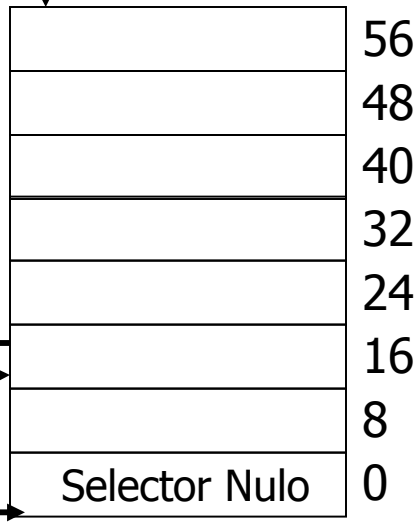


TI=1 Local Decriptor Table (LDT)

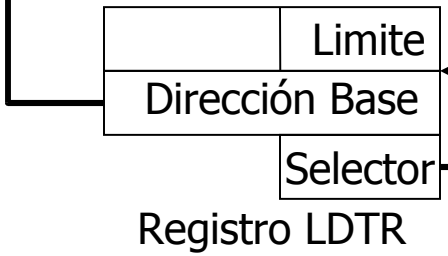
TI=0 Global Decriptor Table (GDT)

Área de memoria exclusiva de cada tarea para almacenar su código y sus datos. (Área Local)



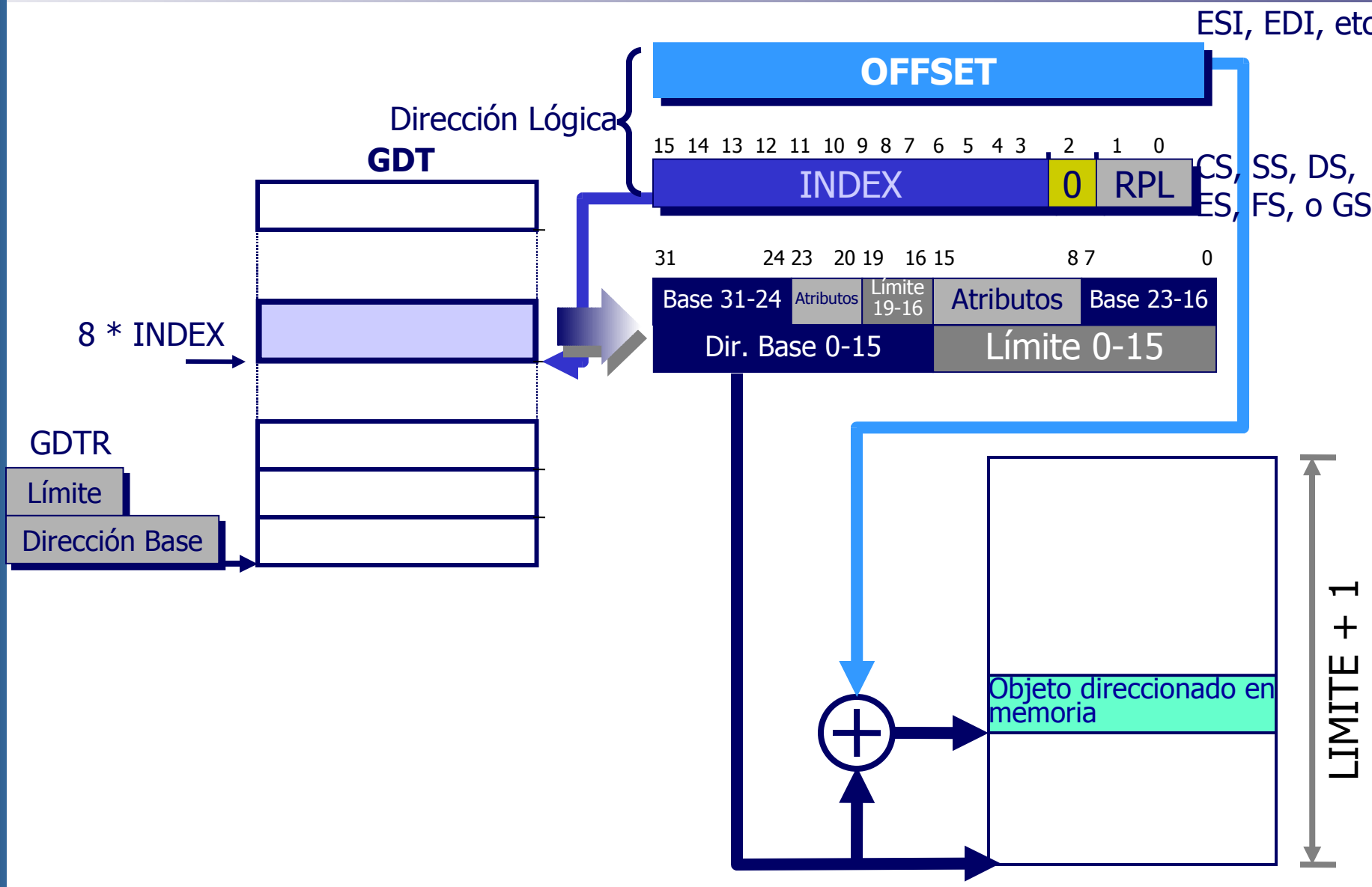


Área de memoria común a todas las tareas, para que éstas puedan acceder a datos globales del sistema, o a código propio del Sistema Operativo (Base de modo de permitir la comunicación entre las aplicaciones. (Área Global).



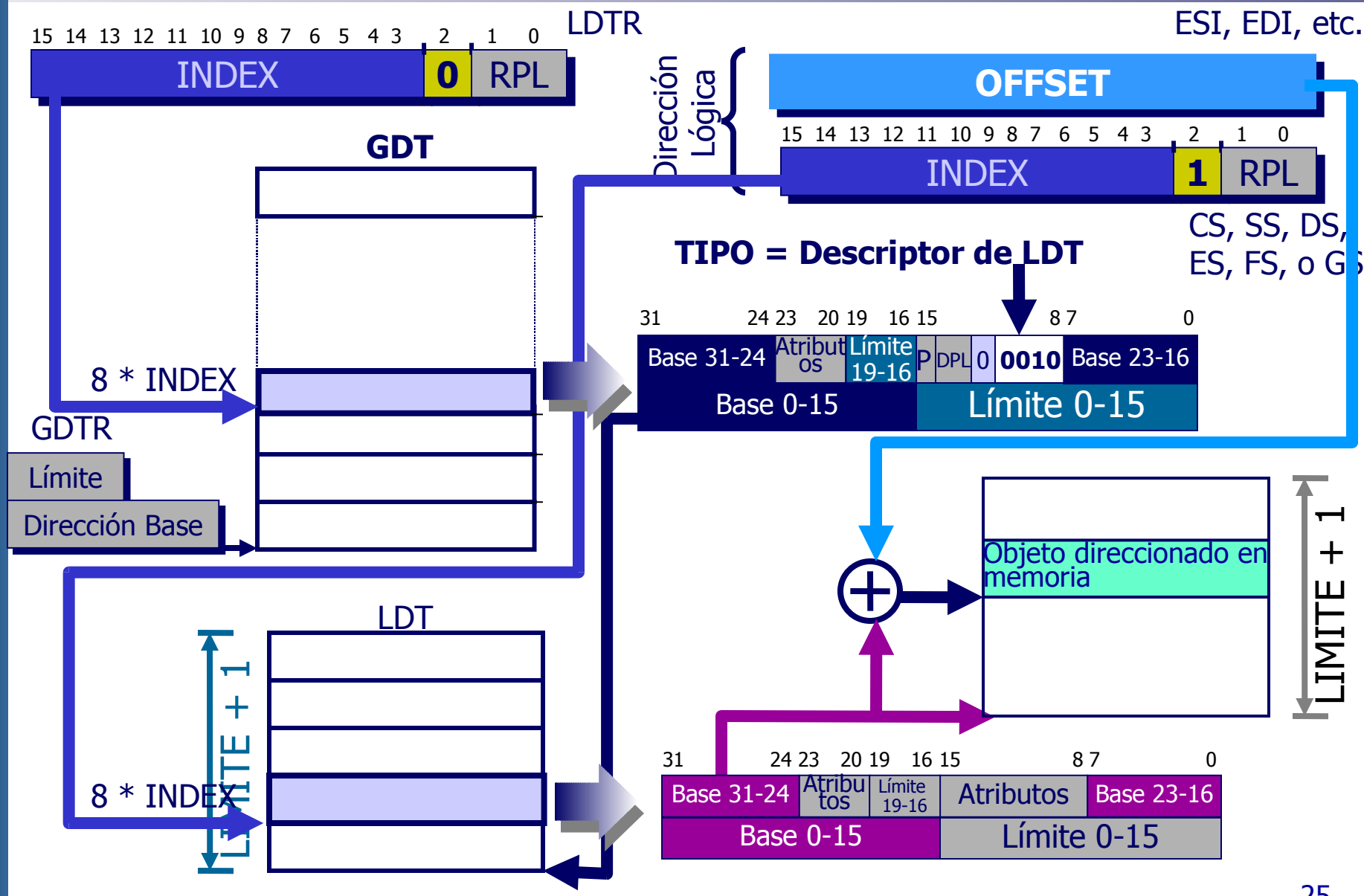
Translación de dirección lógica a lineal (Selector con TI=0)

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3



Translación de dirección lógica a lineal (Selector con TI=1)

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3

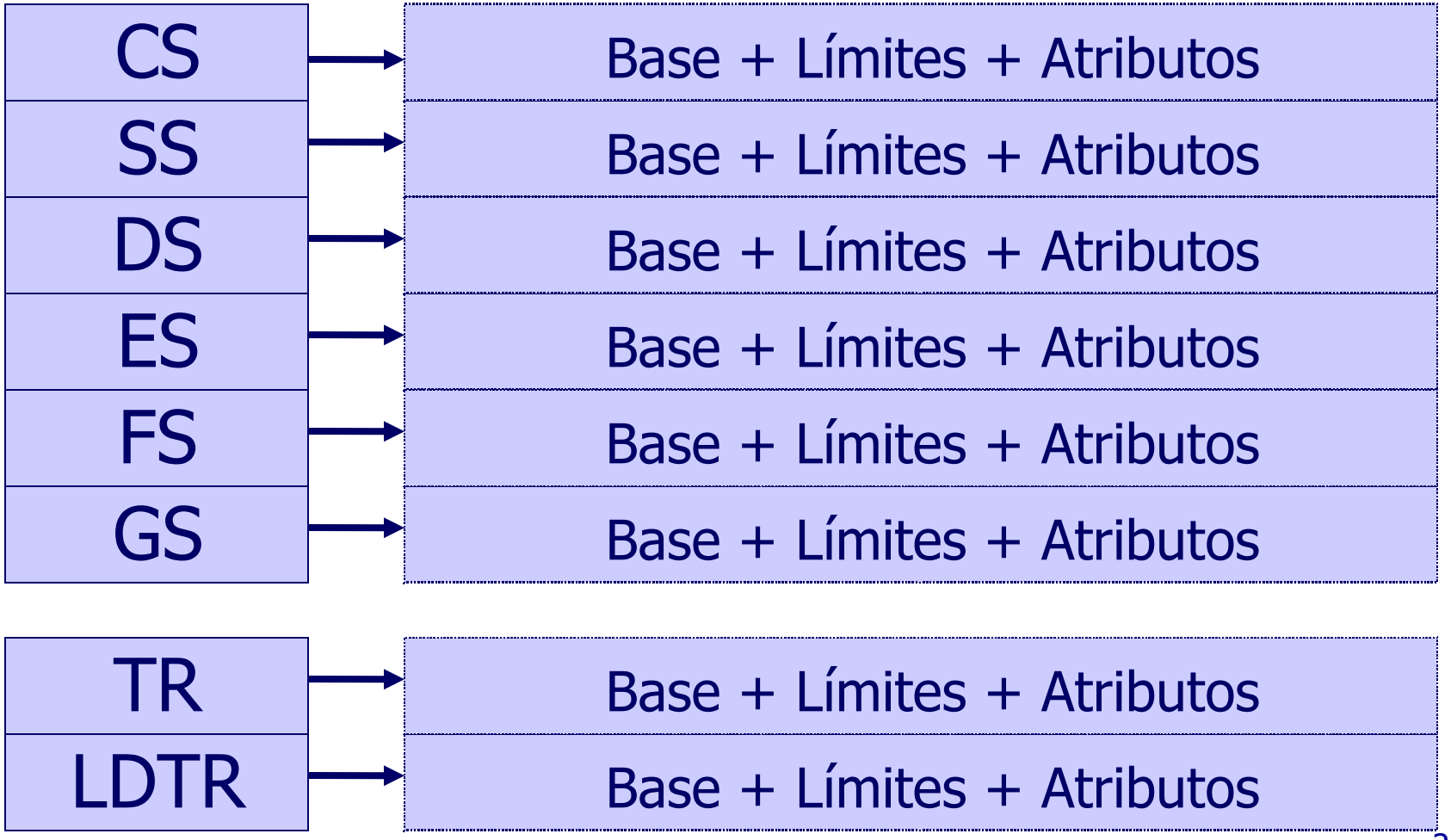


Registros hidden. "Memorizando" los descriptores

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3

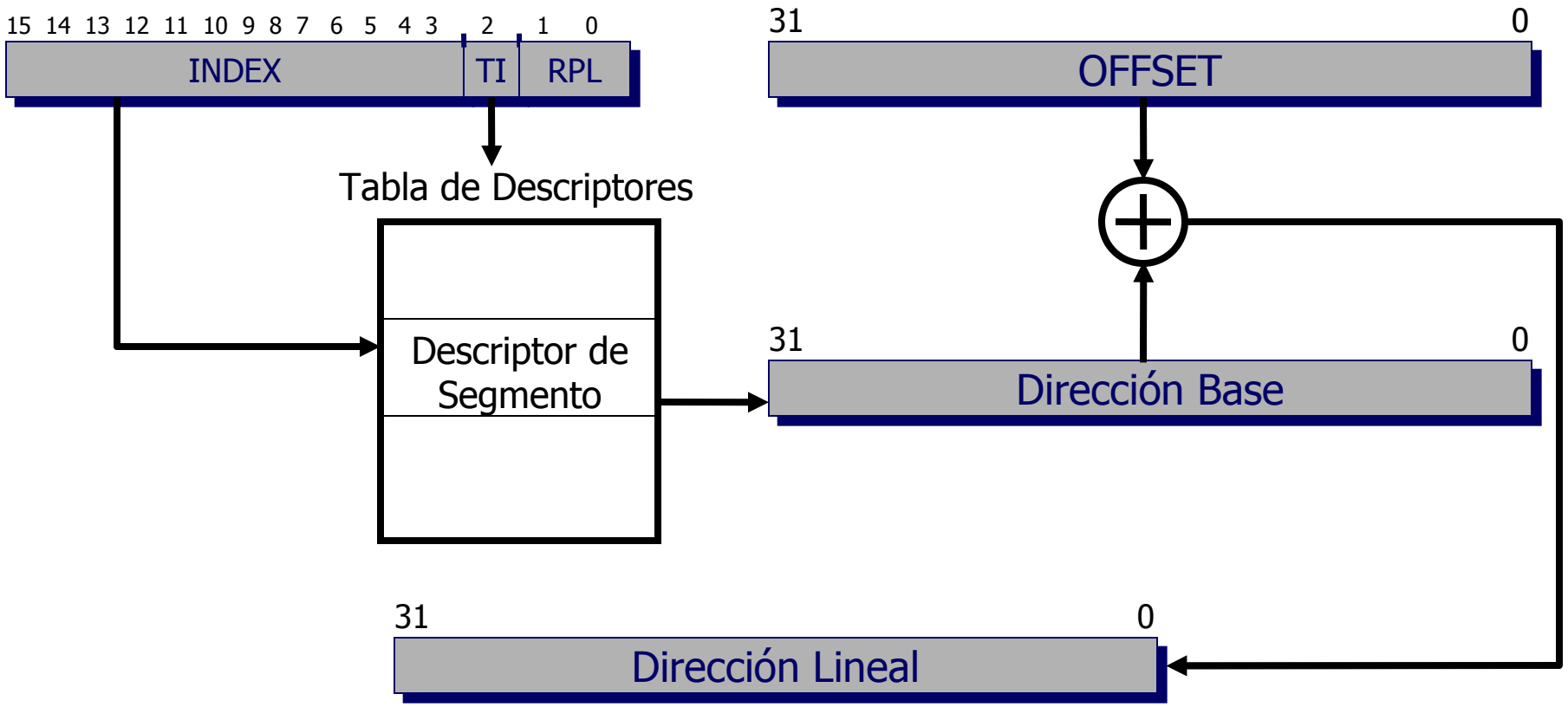
Para evitar un acceso a la GDT o LDT cada vez que necesita un descriptor de segmento, el procesador mantiene un registro caché invisible por cada registro cuyo contenido sea un selector.

Esos registros no son accesibles ni siquiera al programador del S.O.



Dirección lineal

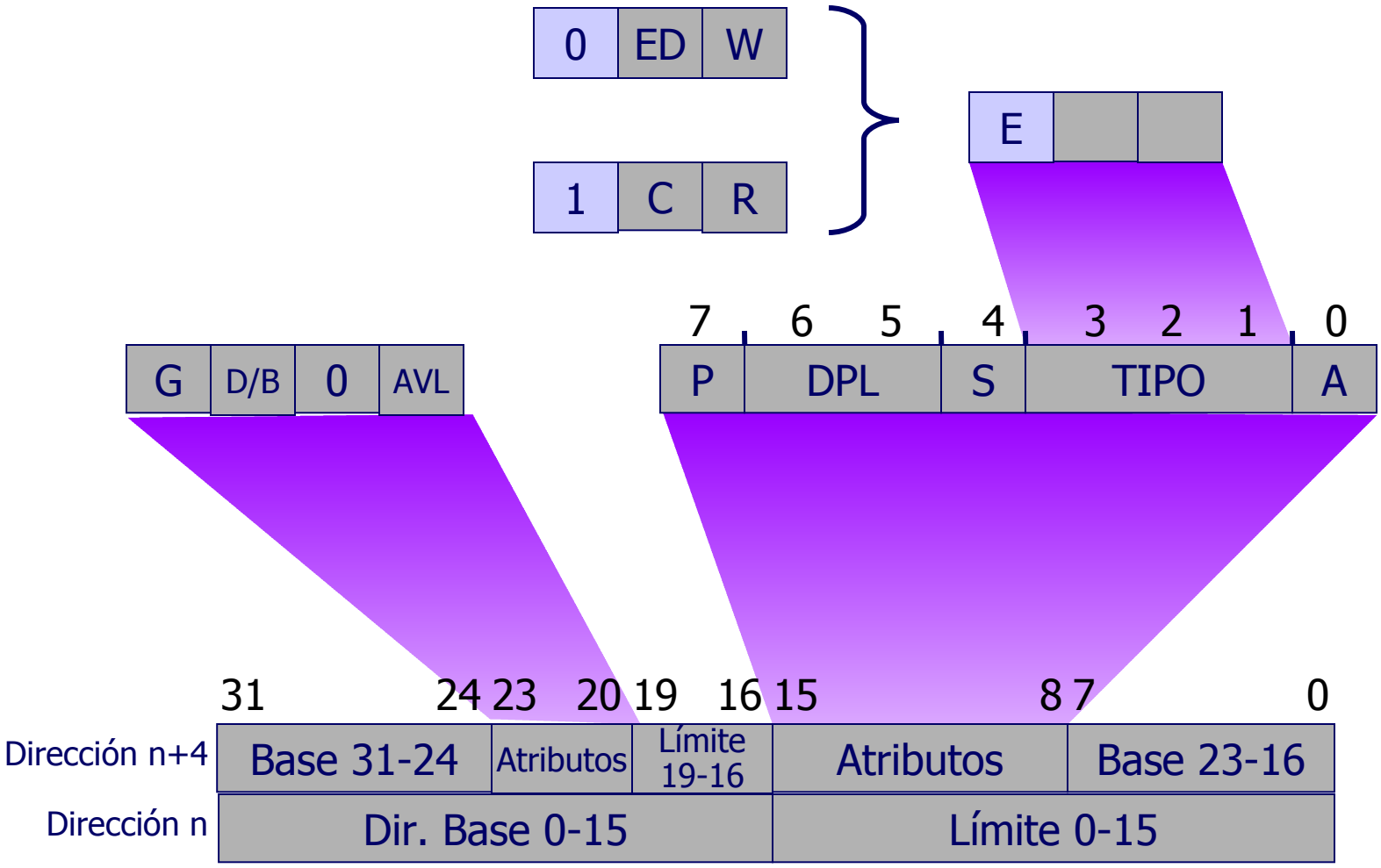
Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3



La Dirección Lineal recibe ese nombre por ser la salida de la Unidad de Segmentación un espacio contiguo y consecutivo de direcciones de memoria

Descriptores de Segmento

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3



Descriptor de Segmento

Campo Tipo para S=0 (Descriptor de Sistema).

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3

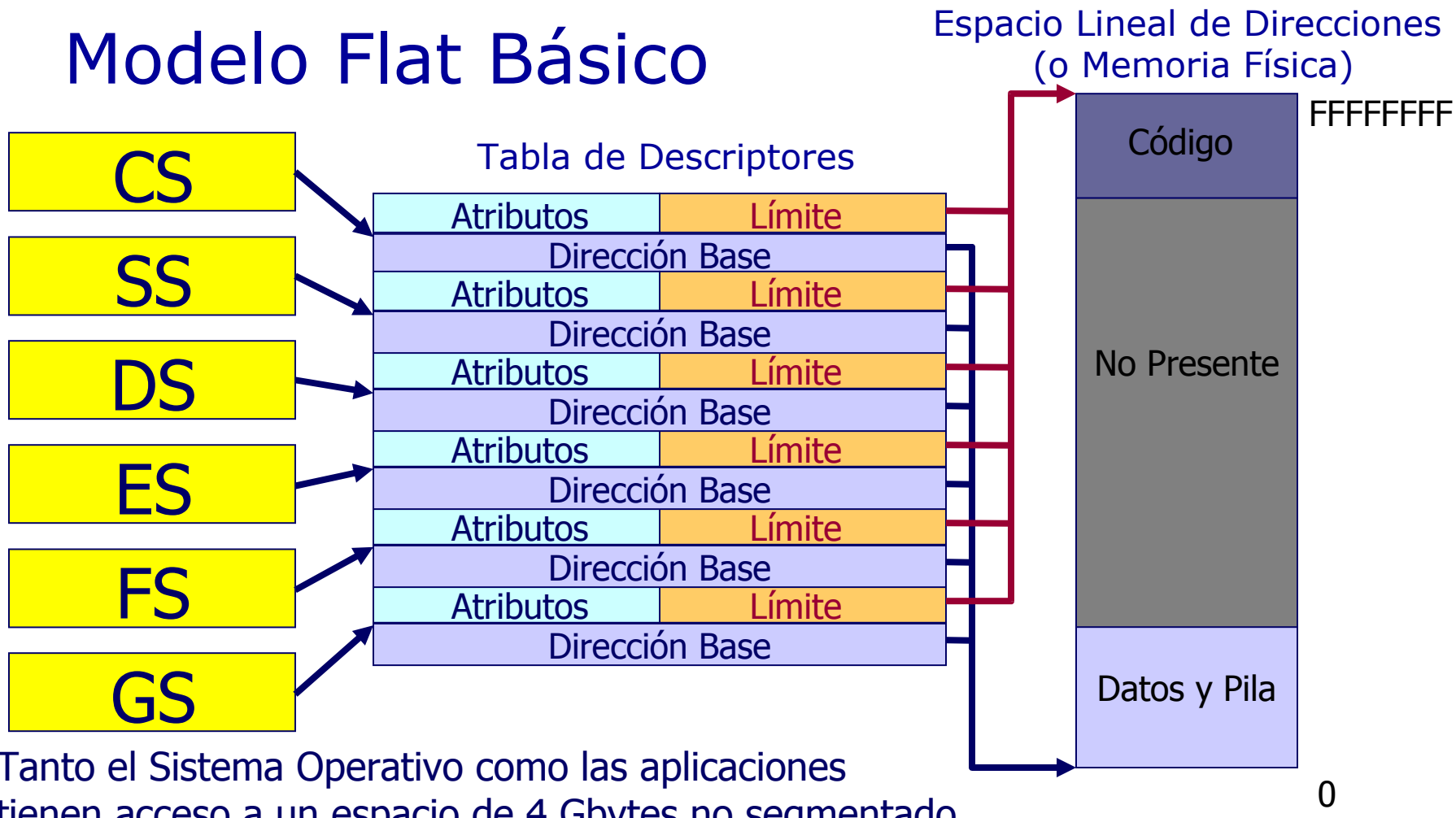
Campo Tipo					Descripción
Valor	11	10	9	8	
0	0	0	0	0	Reservado
1	0	0	0	1	TSS de 16 bits disponible
2	0	0	1	0	LDT
3	0	0	1	1	TSS de 16 bits ocupada
4	0	1	0	0	Puerta de llamada de 16 bits
5	0	1	0	1	Puerta de tarea
6	0	1	1	0	Puerta de Interupcion de 16 bits
7	0	1	1	1	Puerta de Trap de 16 bits
8	1	0	0	0	Reservado
9	1	0	0	1	TSS de 32 bits disponible
10	1	0	1	0	Reservado
11	1	0	1	1	TSS de 32 bits disponible
12	1	1	0	0	TSS de 32 bits ocupada
13	1	1	0	1	Reservado
14	1	1	1	0	Puerta de Interrupción de 32 bits
15	1	1	1	1	Puerta de Trap de 32 bits

- Cuando el Bit S es 0, el descriptor se denomina descriptor de sistema.
- En ciertos casos, el descriptor corresponde a un segmento para uso del sistema operativo.
- En otras ocasiones es simplemente un mecanismo que utilizará el Sistema Operativo.

Modelos de Memoria

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3

Modelo Flat Básico



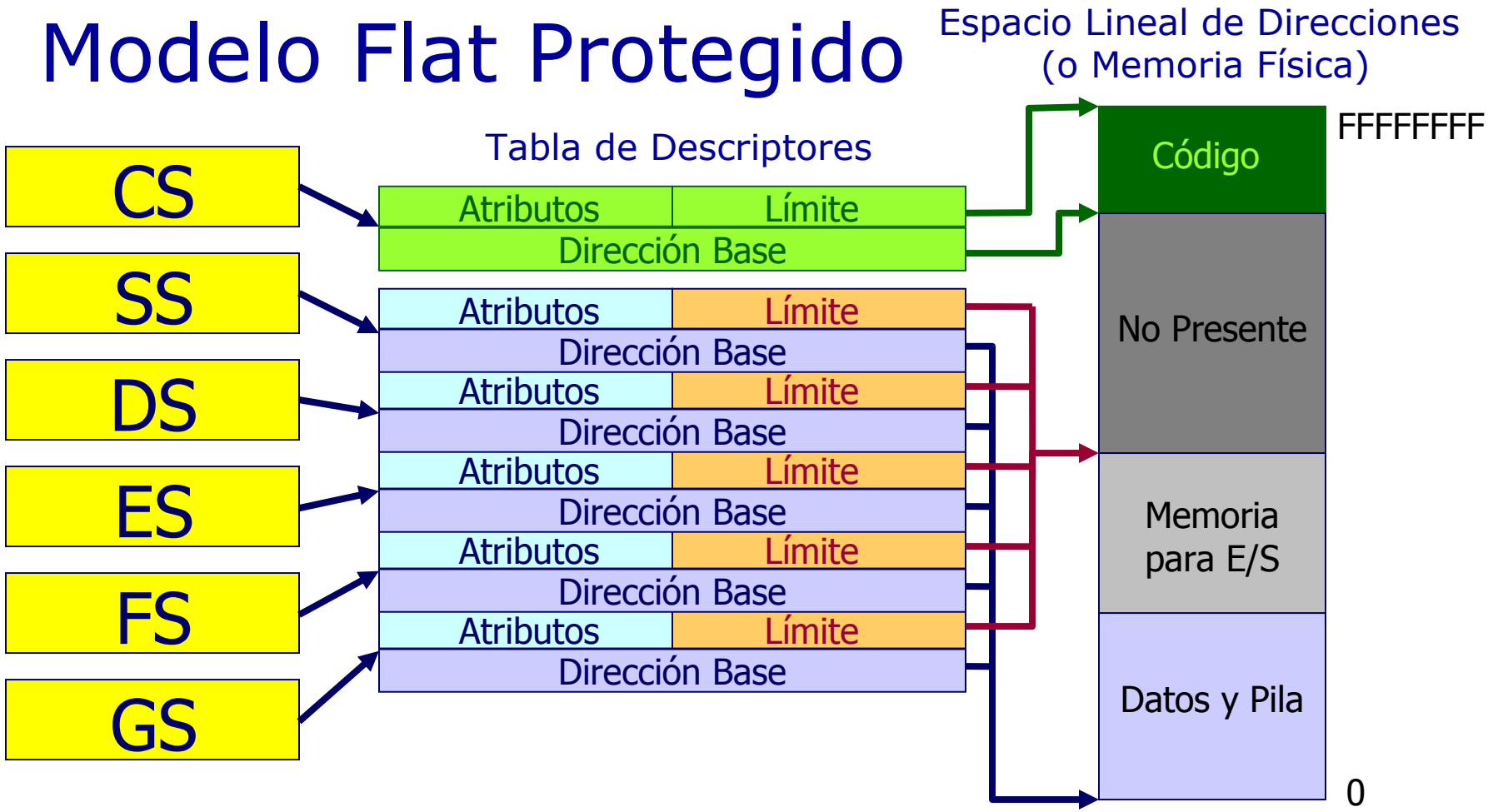
Tanto el Sistema Operativo como las aplicaciones tienen acceso a un espacio de 4 Gbytes no segmentado.

Se evitan las excepciones por exceso en el límite de memoria ya que el límite de todos los descriptores es FFFFFFFF. Aún si se accede a áreas en las que no existe memoria física.

Modelos de Memoria

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3

Modelo Flat Protegido

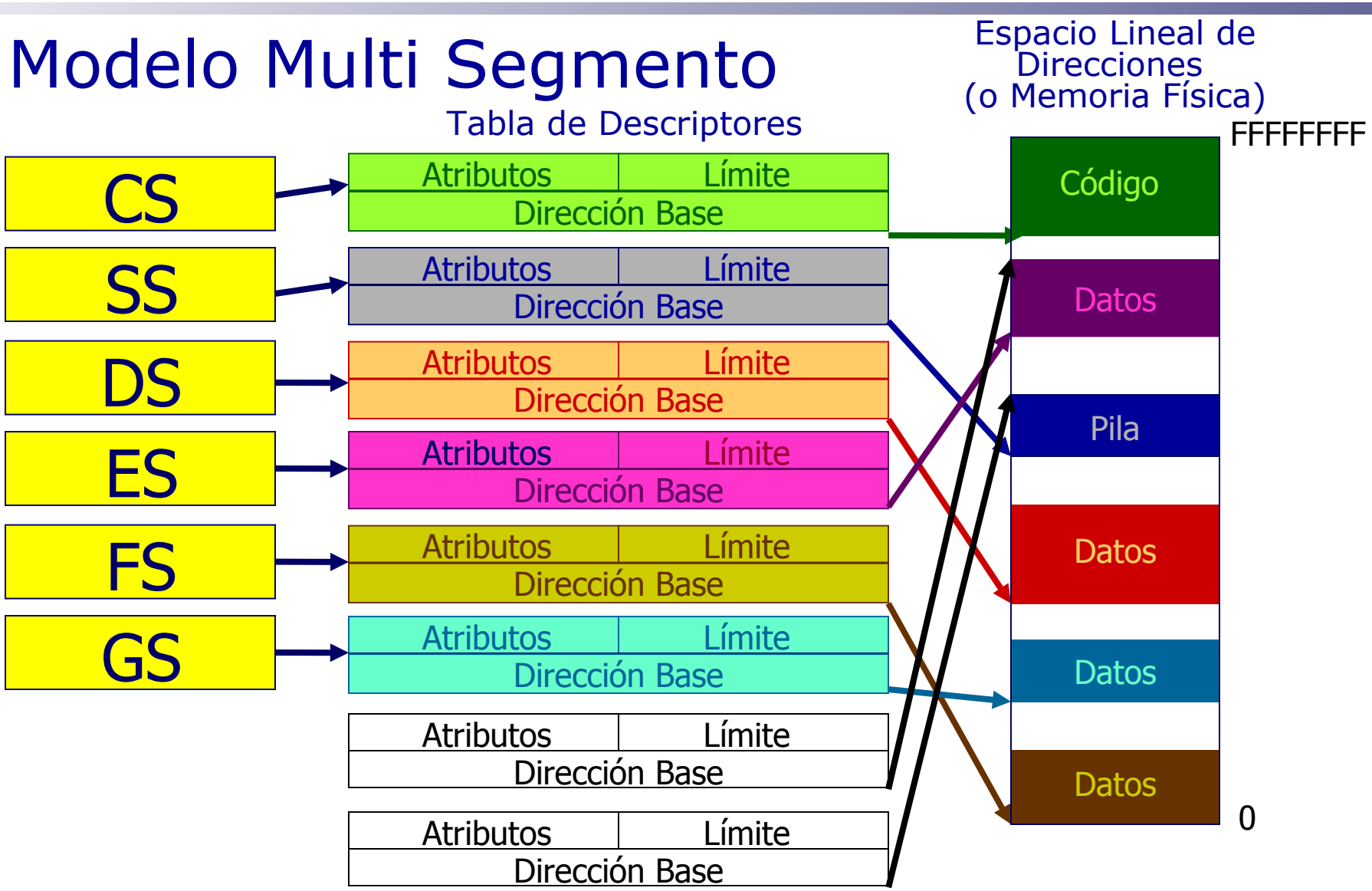


Los segmentos tienen el límite acorde a la memoria física instalada en el sistema.

Modelos de Memoria.

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 3

Modelo Multi Segmento



Manejo de Memoria: Tabla GDT

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 2

Una GDT por cada procesador presente en el sistema
Se almacenan en un array denominado ***cpu_gdt_table***

Linux's GDT	Segment Selectors
null	0x0
reserved	
reserved	
reserved	
not used	
not used	
TLS #1	0x33
TLS #2	0x3b
TLS #3	0x43
reserved	
reserved	
reserved	
kernel code	0x60 (__KERNEL_CS)
kernel data	0x68 (__KERNEL_DS)
user code	0x73 (__USER_CS)
user data	0x7b (__USER_DS)

Linux's GDT	Segment Selectors
TSS	0x80
LDT	0x88
PNPBIOS 32-bit code	0x90
PNPBIOS 16-bit code	0x98
PNPBIOS 16-bit data	0xa0
PNPBIOS 16-bit data	0xa8
PNPBIOS 16-bit data	0xb0
APMBIOS 32-bit code	0xb8
APMBIOS 16-bit code	0xc0
APMBIOS data	0xc8
not used	
not used	
not used	
not used	
not used	
double fault TSS	0xf8

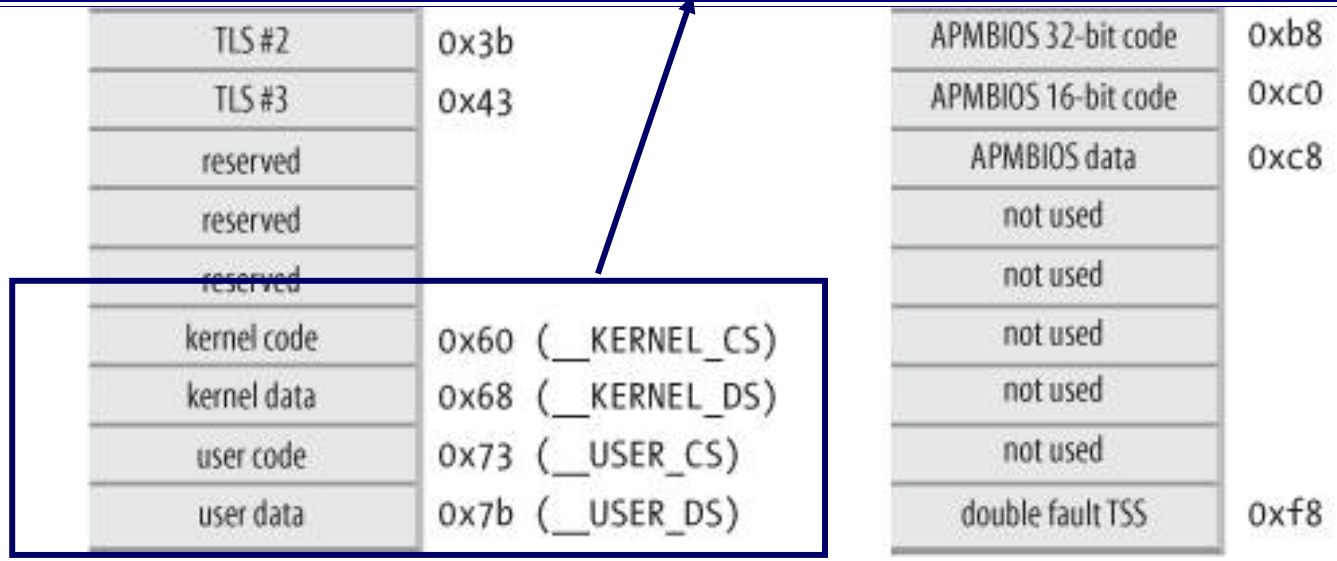
Manejo de Memoria: Tabla GDT

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 2

Segmentación:
Basada en Modelo FLAT

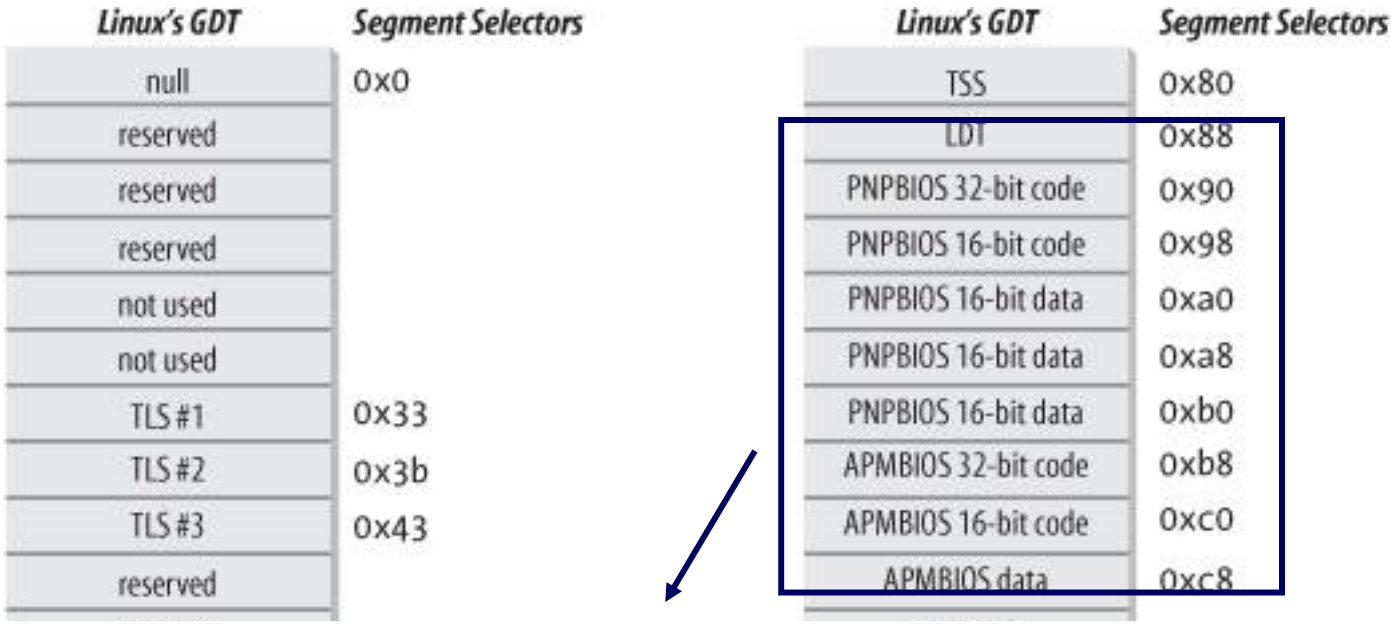
Linux's GDT *Segment Selectors* *Linux's GDT* *Segment Selectors*

Segment	Base	G	Limit	S	Type	DPL	D/B	P
user code	0x00000000	1	0xfffff	1	10	3	1	1
user data	0x00000000	1	0xfffff	1	2	3	1	1
kernel code	0x00000000	1	0xfffff	1	10	0	1	1
kernel data	0x00000000	1	0xfffff	1	2	0	1	1



Manejo de Memoria: Tabla GDT

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 2



Linux's GDT	Segment Selectors
null	0x0
reserved	
reserved	
reserved	
not used	
not used	
TLS #1	0x33
TLS #2	0x3b
TLS #3	0x43
reserved	

Linux's GDT	Segment Selectors
TSS	0x80
LDT	0x88
PNPBIOS 32-bit code	0x90
PNPBIOS 16-bit code	0x98
PNPBIOS 16-bit data	0xa0
PNPBIOS 16-bit data	0xa8
PNPBIOS 16-bit data	0xb0
APMBIOS 32-bit code	0xb8
APMBIOS 16-bit code	0xc0
APMBIOS data	0xc8

- Segmentos para Advanced Power Management (APM): El código del BIOS usa estos segmentos, de modo que cuando el driver APM de Linux invoca funciones del BIOS para obtener el estado o configurar un dispositivo APM, usará estos segmentos de código y datos.
- Segmentos para servicios Plug and Play (PnP) del BIOS. Como en el caso previo, el código del BIOS utiliza estos segmentos, de modo que cuando un driver PnP de Linux invoque funciones del BIOS para detectar recursos utilizados por dispositivos PnP, usará estos segmentos de código y datos.

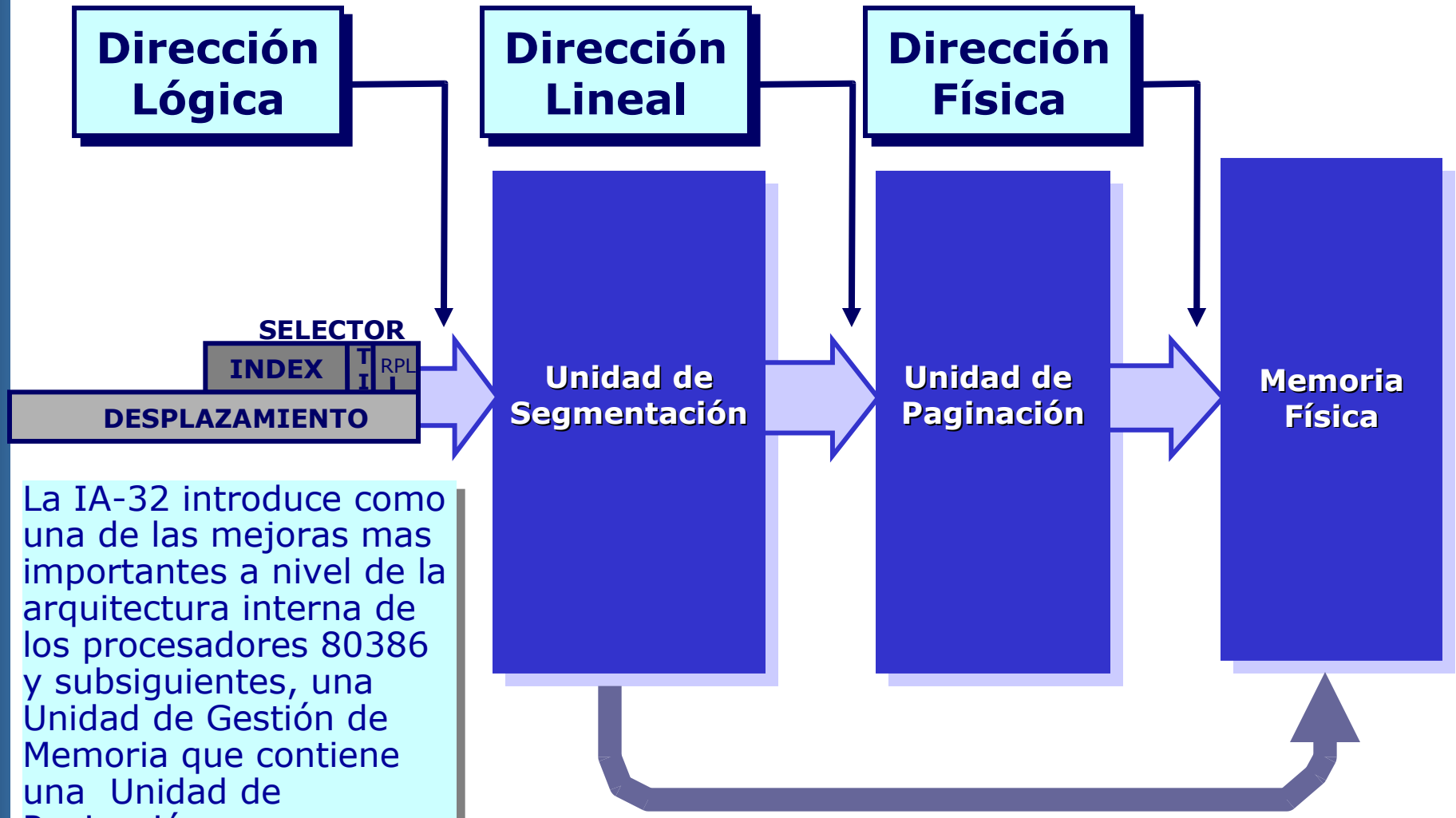
Paginación de Memoria

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4

- La segmentación posee claras ventajas para proveer un entorno flexible en la programación de aplicaciones.
- Para Administración de la memoria por parte del sistema operativo, la variabilidad del tamaño de los segmentos introduce complejidad en el diseño de un sistema de memoria virtual.
- Los sistemas operativos como UNIX desde su concepción trabajaron la memoria en bloques de tamaño uniforme.
- La Paginación cumple con ese requisito.

Estructura de la MMU

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4



La IA-32 introduce como una de las mejoras mas importantes a nivel de la arquitectura interna de los procesadores 80386 y subsiguientes, una Unidad de Gestión de Memoria que contiene una Unidad de Paginación a continuación de la Unidad de Segmentación

Paginación de Memoria

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4

- En los primeros procesadores (hasta el Pentium Pro), el tamaño de página es fijo: 4Kbytes.
- A partir del Pentium Pro cada tarea puede optar por tener páginas de 4 Kbytes, 2 Mbytes, o 4 Mbytes.
- Las páginas son contiguas y a diferencia de los segmentos no se solapan.
- El máximo tamaño de un espacio lineal es 4 Gbytes

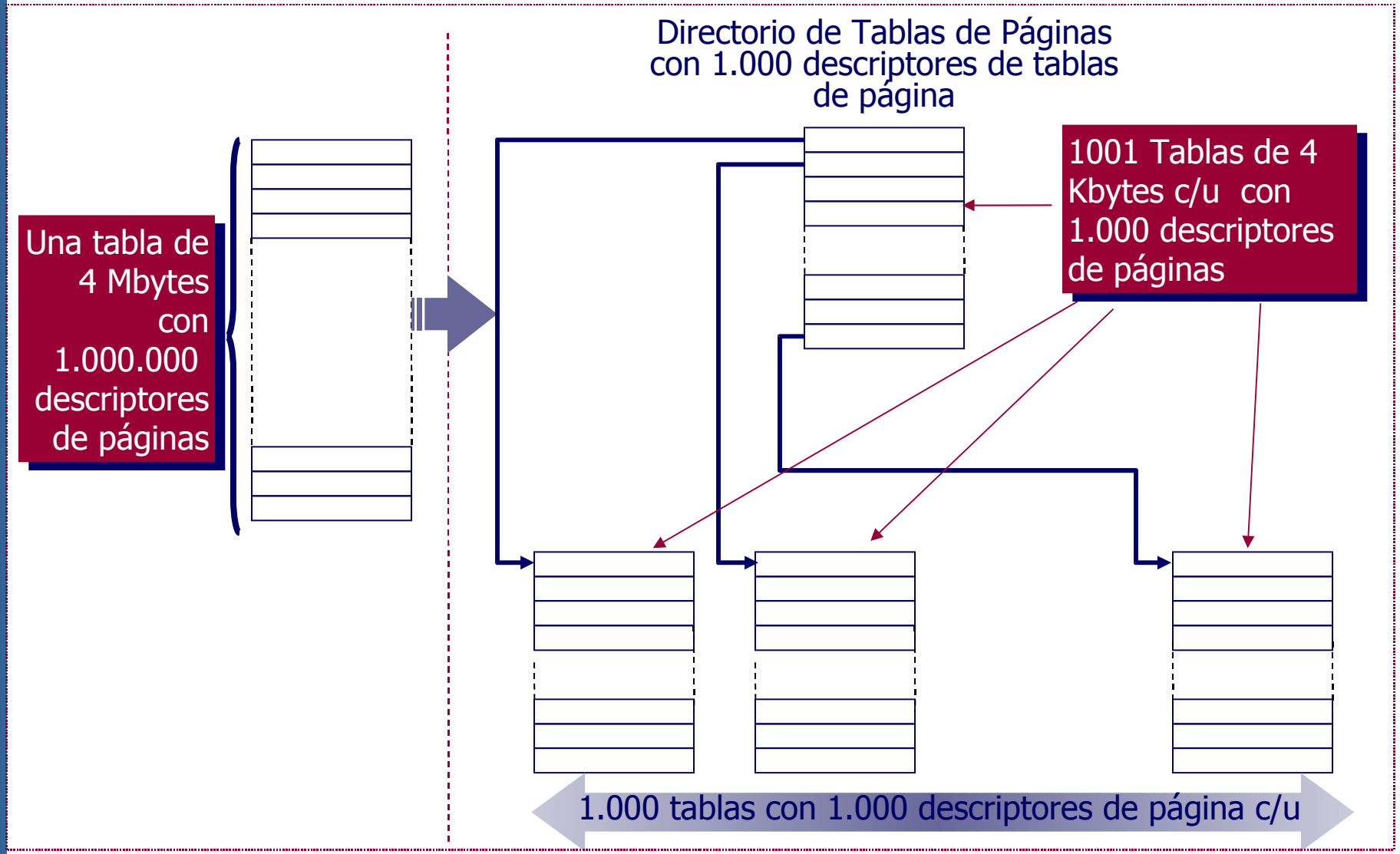
Paginación de Memoria en 4Kbytes

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4

- El espacio Lineal se divide entones en 1.000.000 de páginas de 4 Kbytes c/u, o en 1000 páginas de 4 Mbytes, o en una combinación de ambos tamaños.
- Se necesita un descriptor para cada página que especifique:
  - Dirección Base de la página: Se necesitan 20 bits (comienzan en múltiplos de 4K ya que no se solapa, de modo que los 12 bits menos significativos serán 0), o 10 bits en el caso de páginas de 4 Mbytes.
  - Atributos. Del mismo modo que los segmentos
  - No se requiere límite ya que su tamaño es fijo.
- Tomando el caso de páginas de 4 Kbtes, a 4 bytes por cada descriptor y un millón de descriptores de página, se necesita una tabla de descriptores de página de 4 Mbytes.
- La primer PC 386 venía con 4 Mbytes de RAM.
- Evidentemente se necesita otro approach

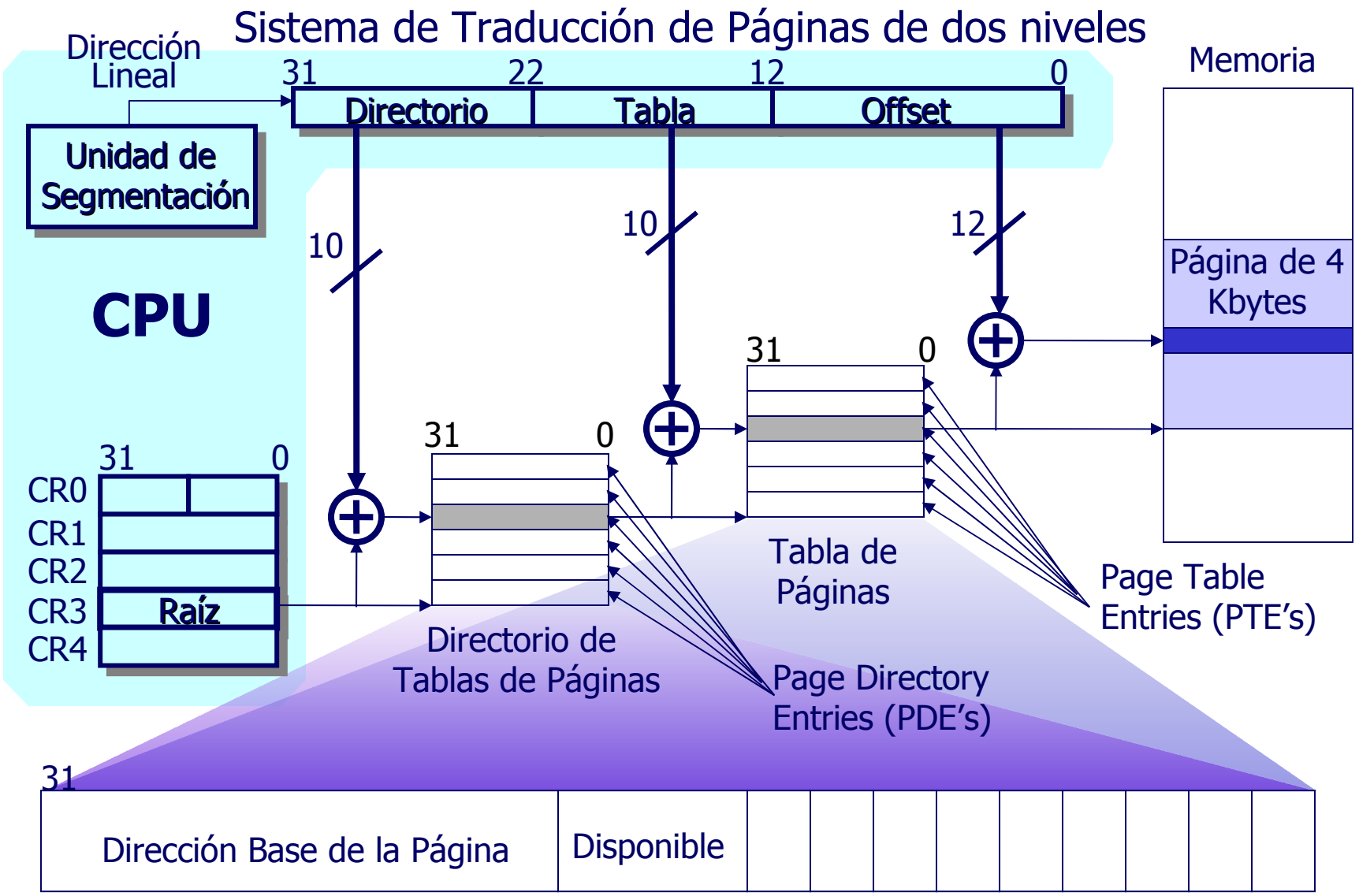
Tabla de Páginas de Memoria

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4



Paginación de Memoria en dos niveles

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4



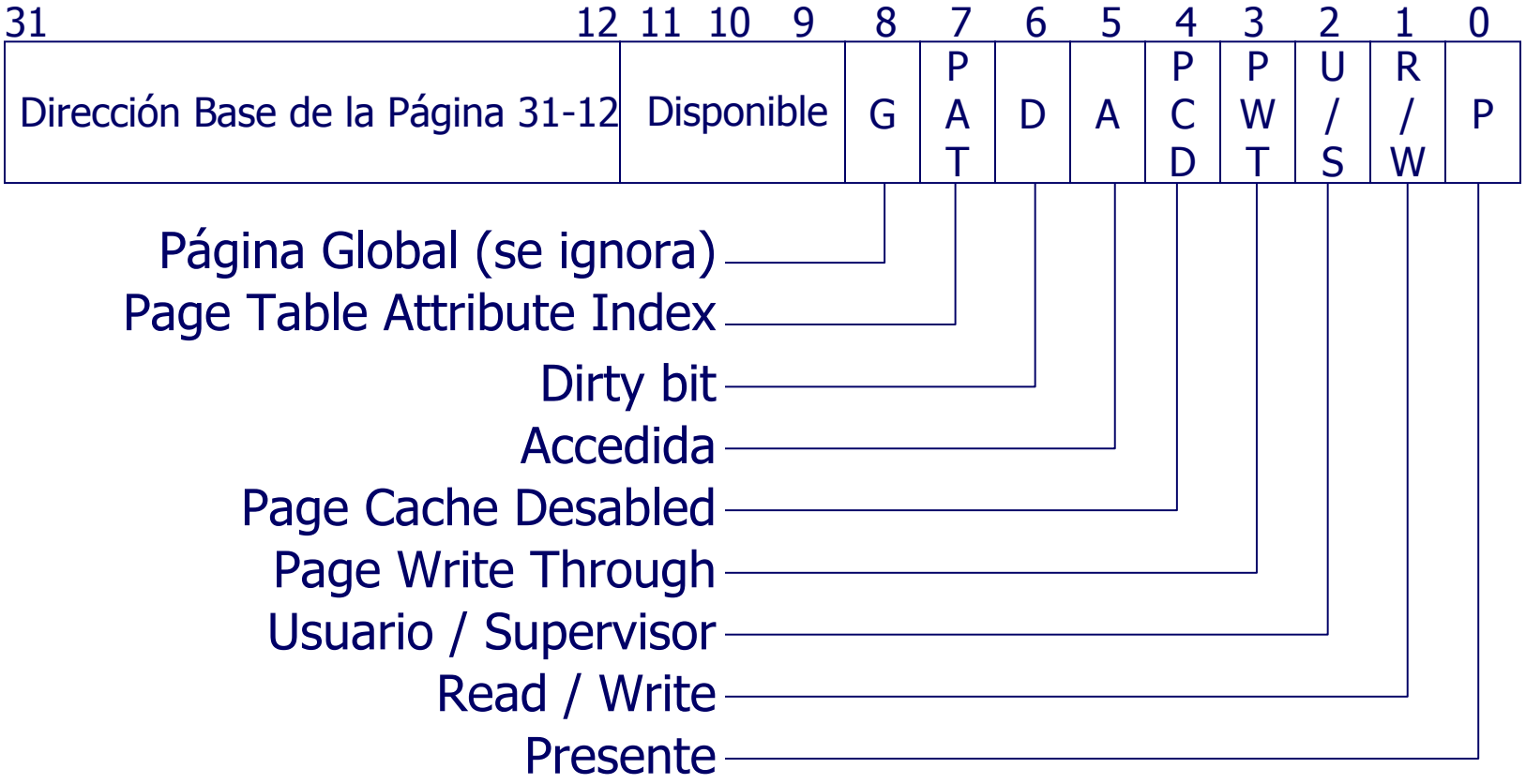
Entrada de Directorio de Página (4Kbyte)

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4

Descriptor de Tabla de Página de 4 Kbytes

Entrada de Tabla de Página

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4

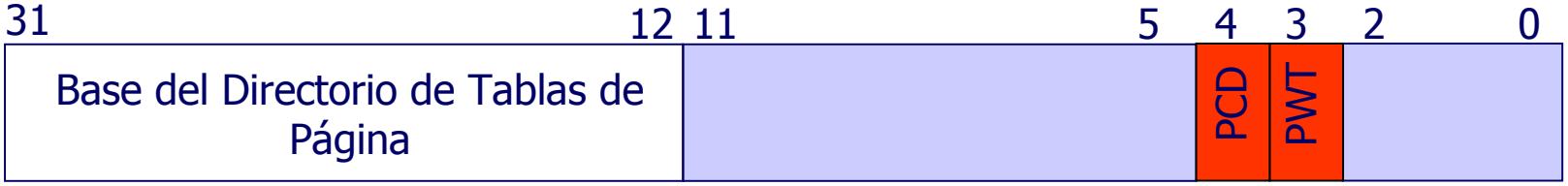


Descriptor de Página de 4 Kbytes

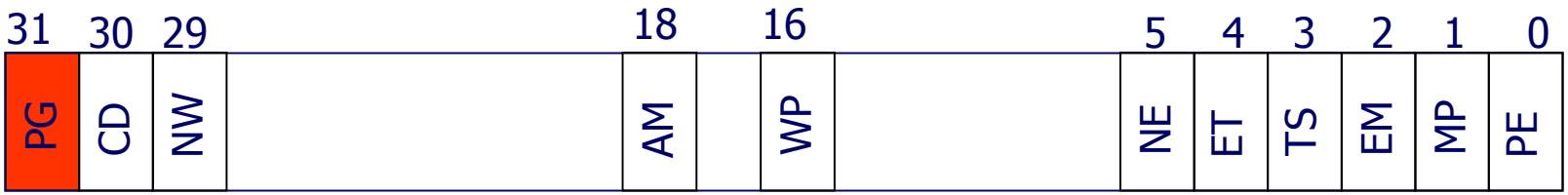
Registros de control asociados

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4

CR3



CR0



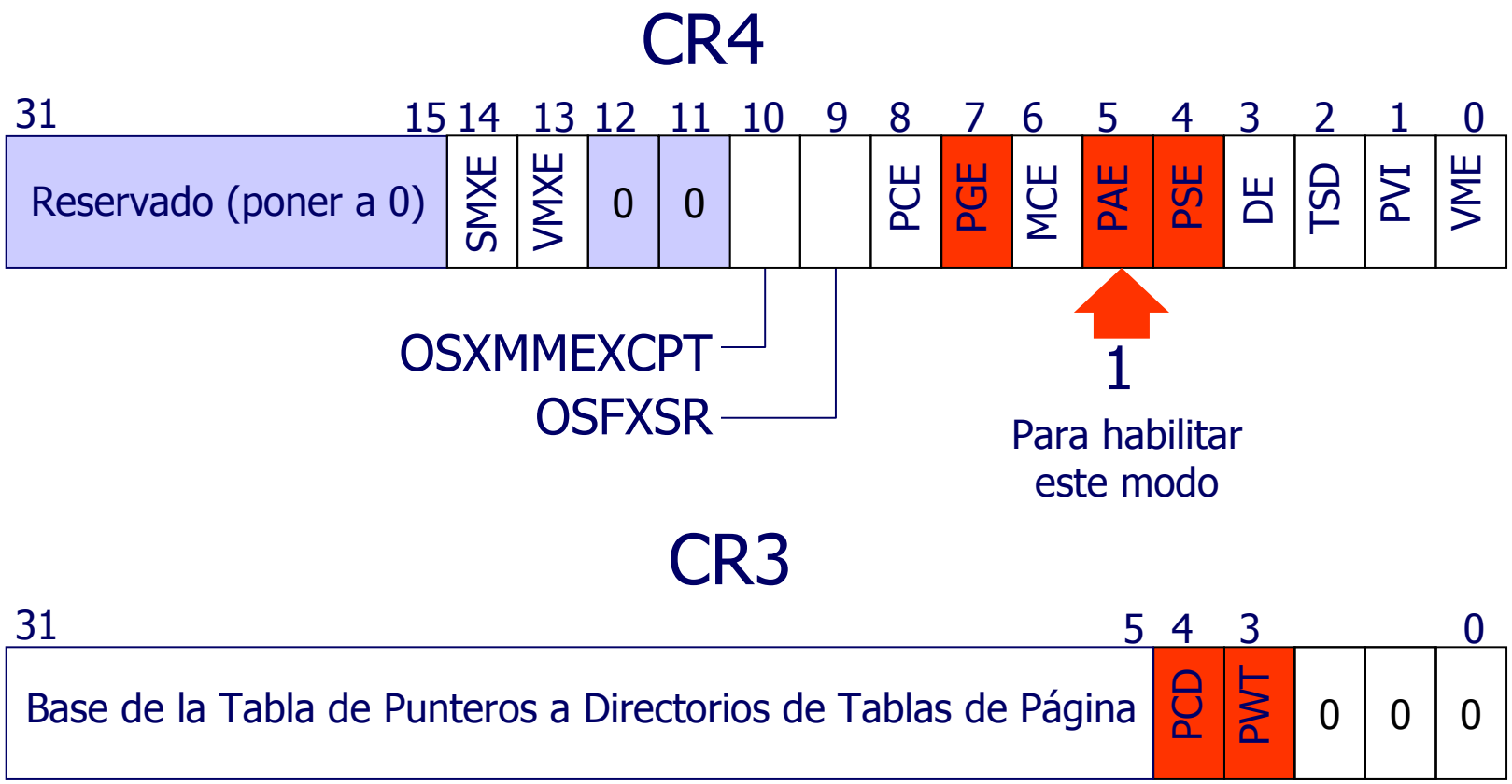
Direccionamiento físico de 36 bits con PAE

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4

- Desde el procesador Pentium Pro, se advirtió que para los servidores de alta performance los 4 Gbytes de memoria física resultarían insuficientes.
- Se implementa PAE (Phisical Address Extension)
- Se habilita con el bit 5 del CR4.
- El procesador habilita 4 líneas de Address adicionales para direccionar 64 Gbytes de memoria física.
- El procesador soporta dos tamaños de página: 4 Kbytes y 2 Mbytes.

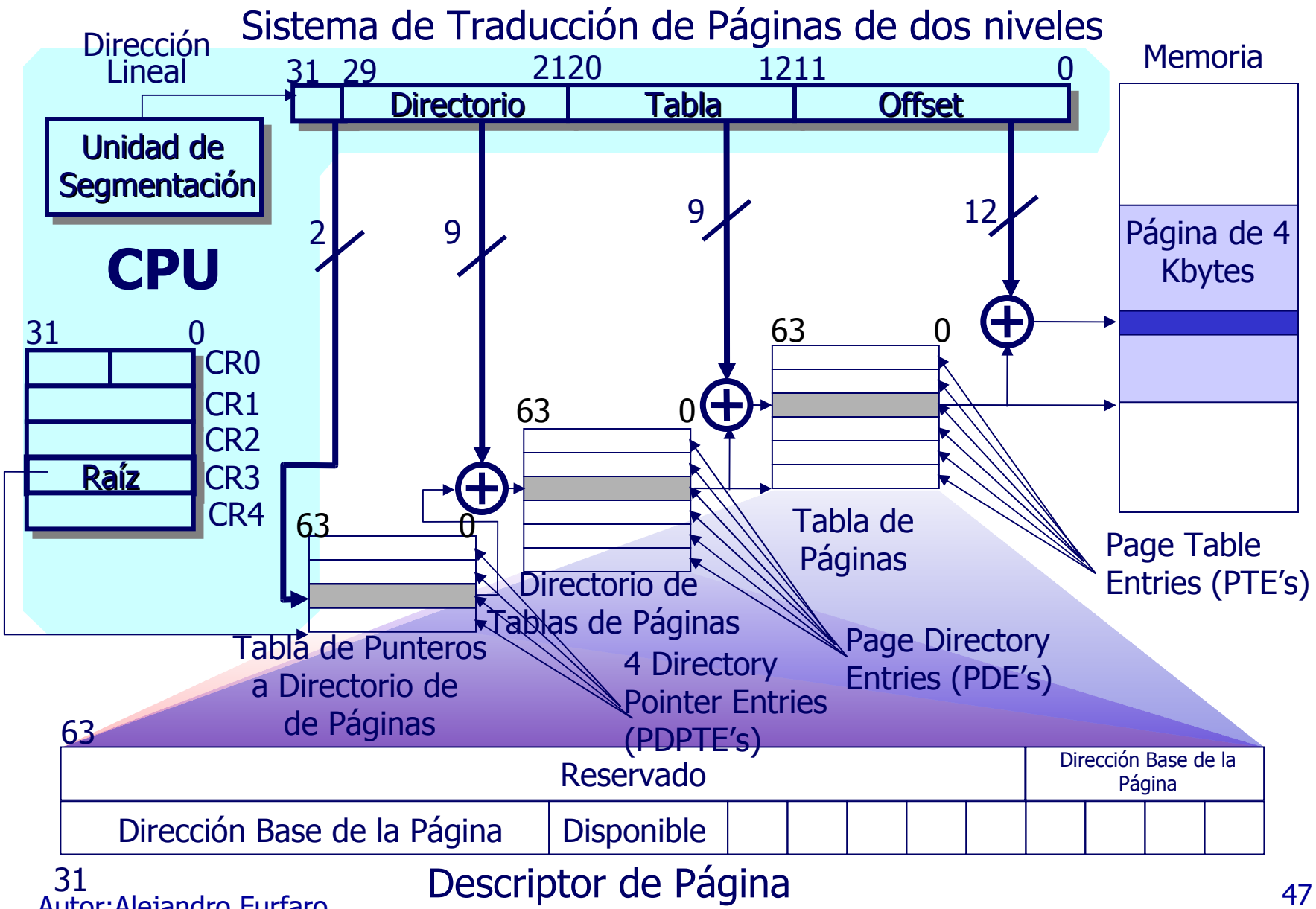
Registros de control asociados

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4



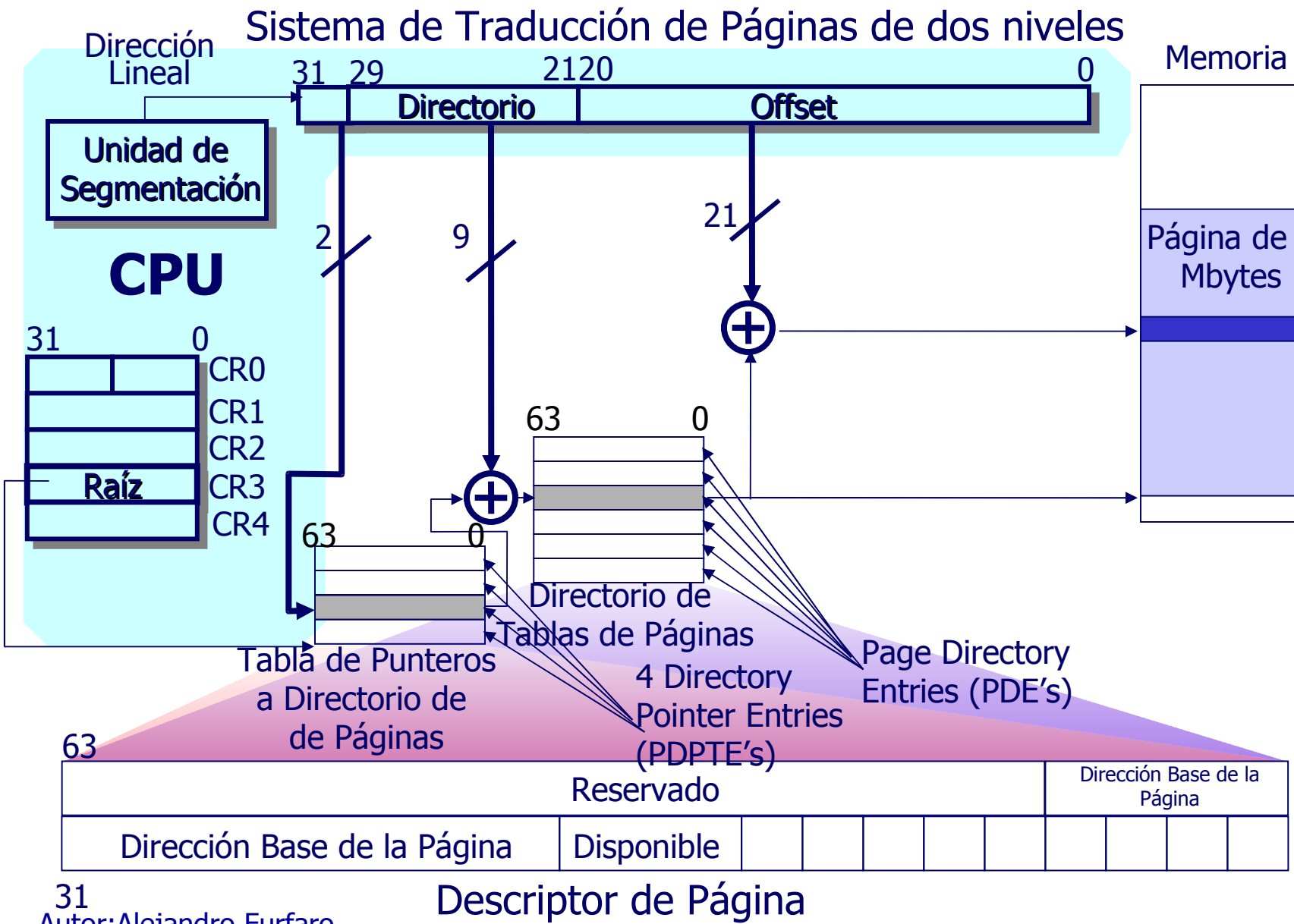
Paginación de Memoria con PAE activa

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4



Paginación de Memoria con PAE activa

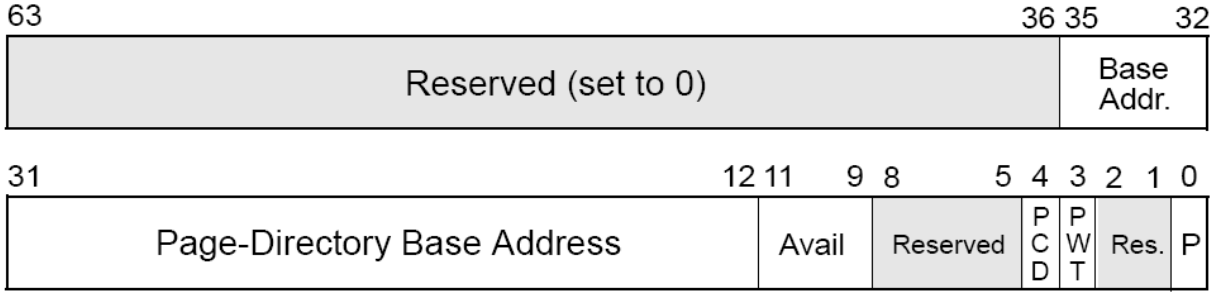
Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4



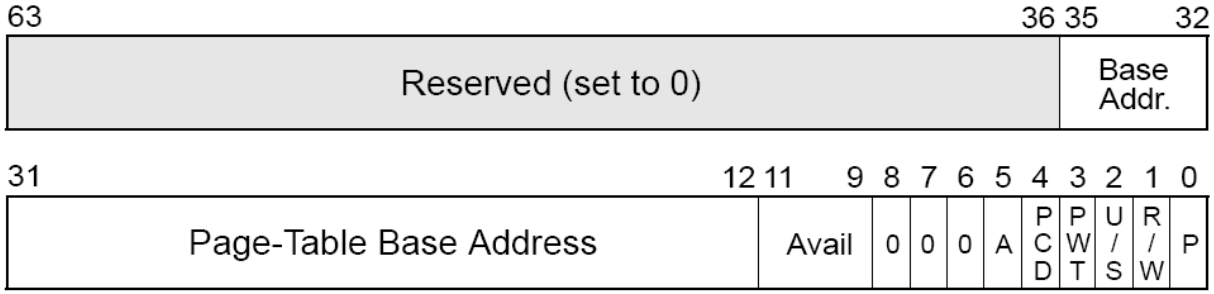
Paginación de Memoria con PAE activa

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4

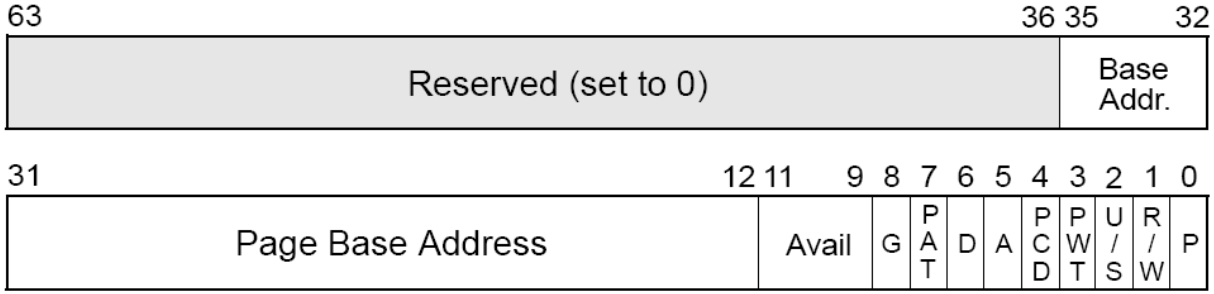
Page-Directory-Pointer-Table Entry



Page-Directory Entry (4-KByte Page Table)



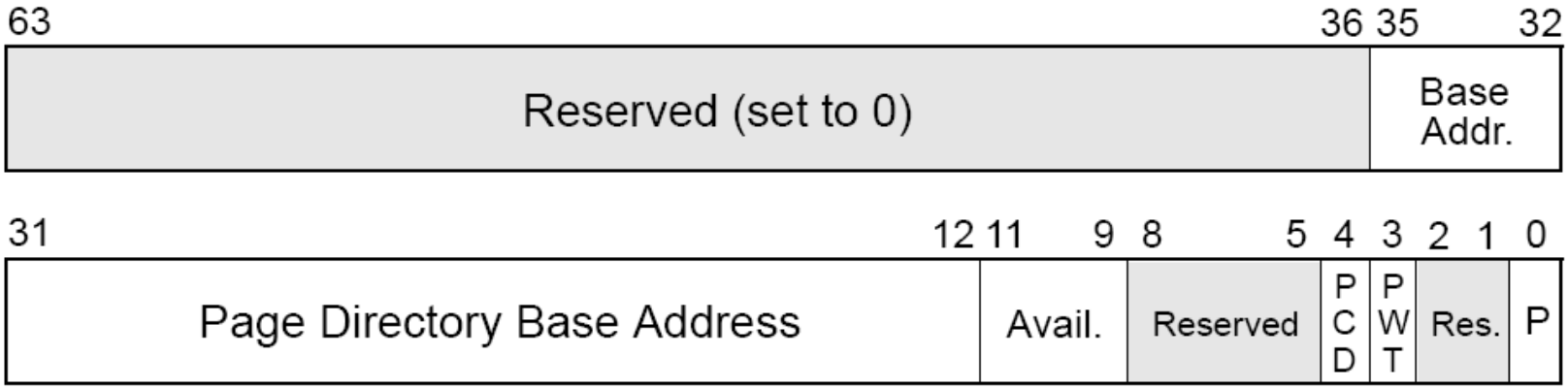
Page-Table Entry (4-KByte Page)



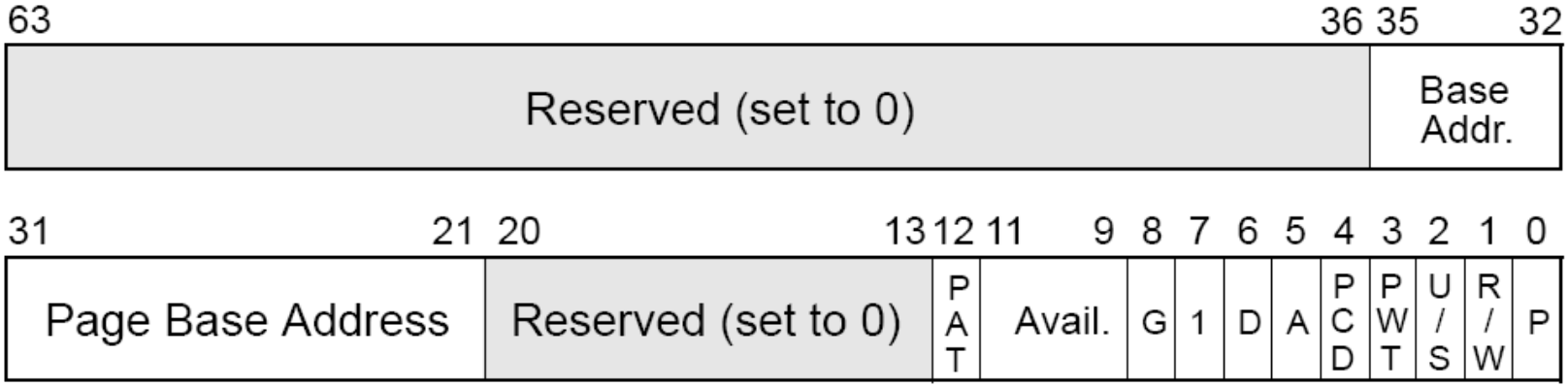
Paginación de Memoria con PAE activa

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4

Page-Directory-Pointer-Table Entry



Page-Directory Entry (2-MByte Page)



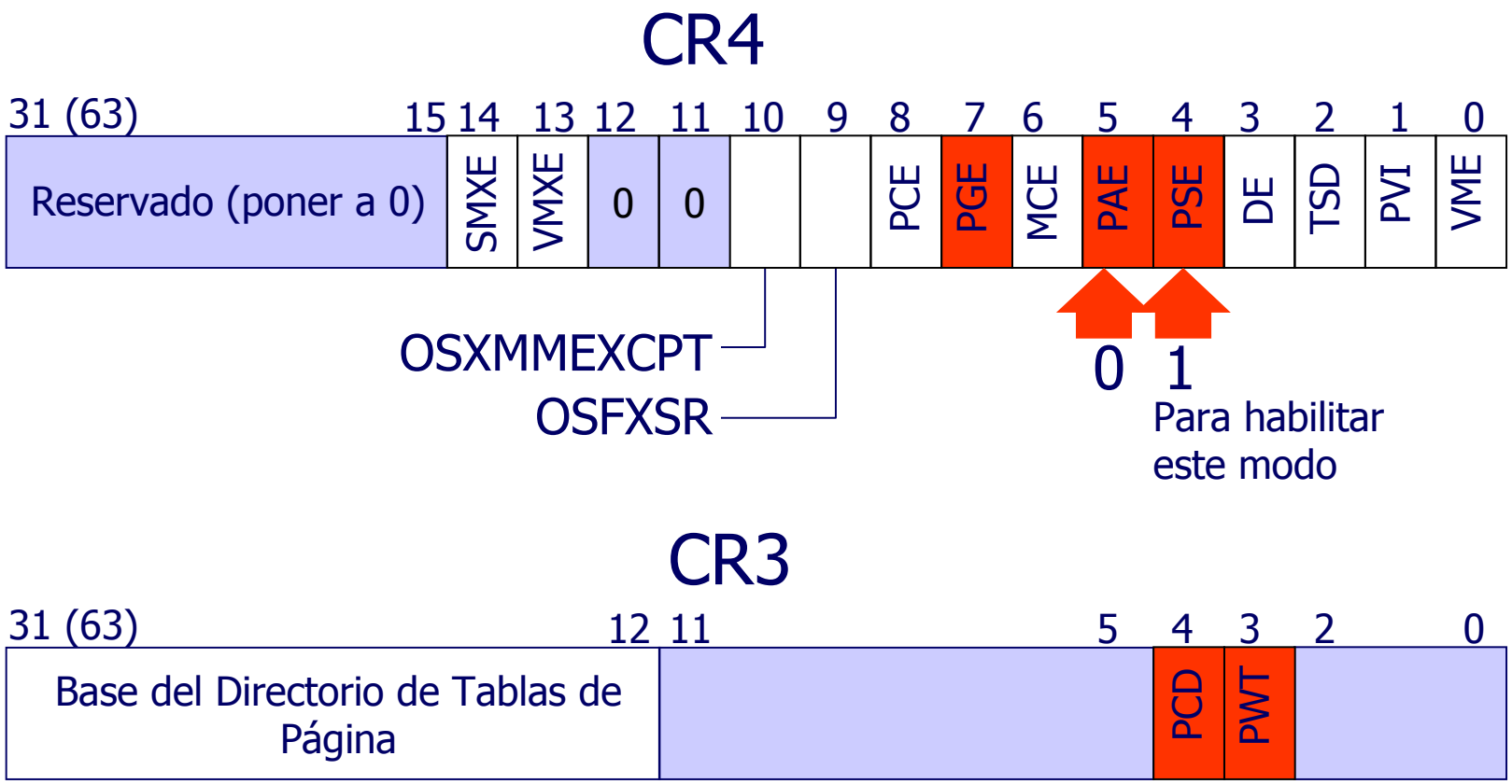
Paginación de Memoria con PSE-36

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4

- Alternativa para direccionar memoria física con 36 bits.
- Permite manipular 64 Gbytes de memoria física.
- Si está disponible CPUID devuelve bit 17 de EDX seteado
- PAE en CR4 DEBE estar deshabilitado!!

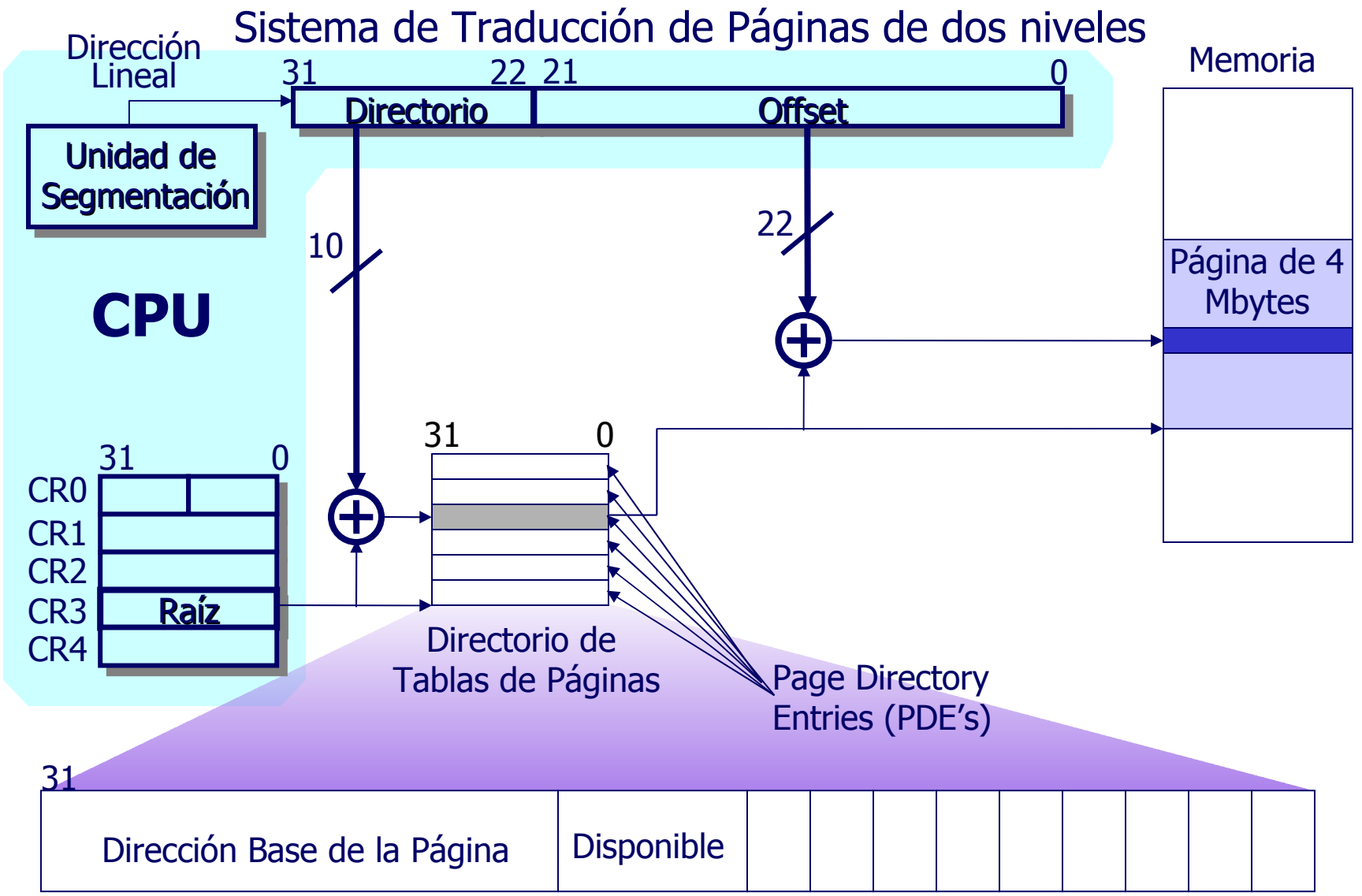
Registros de control asociados

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4



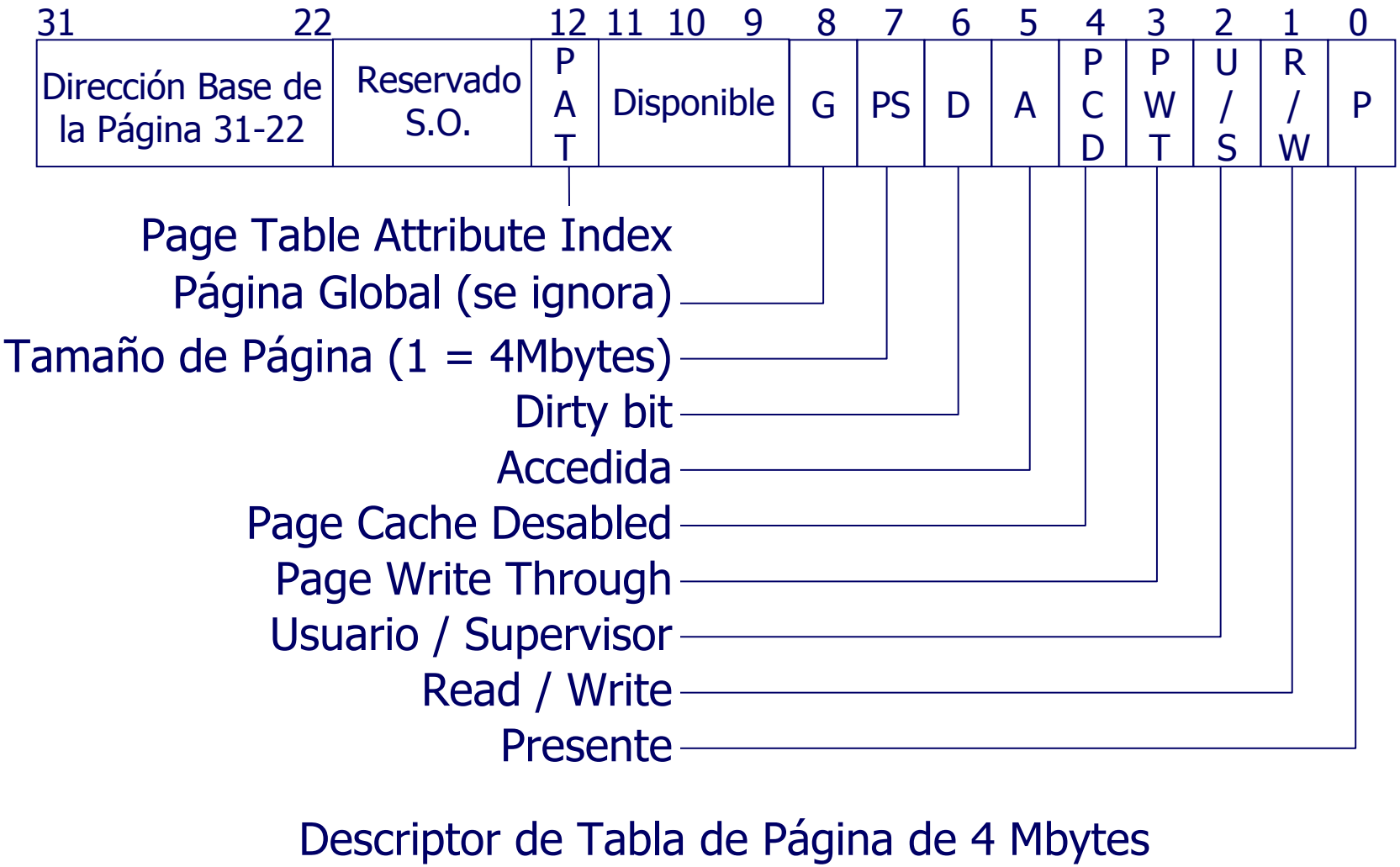
Paginación de Memoria en 4 Mbytes

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4



Entrada de Directorio de Página (4Mbyte)

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4



Descriptor de Tabla de Página de 4 Mbytes

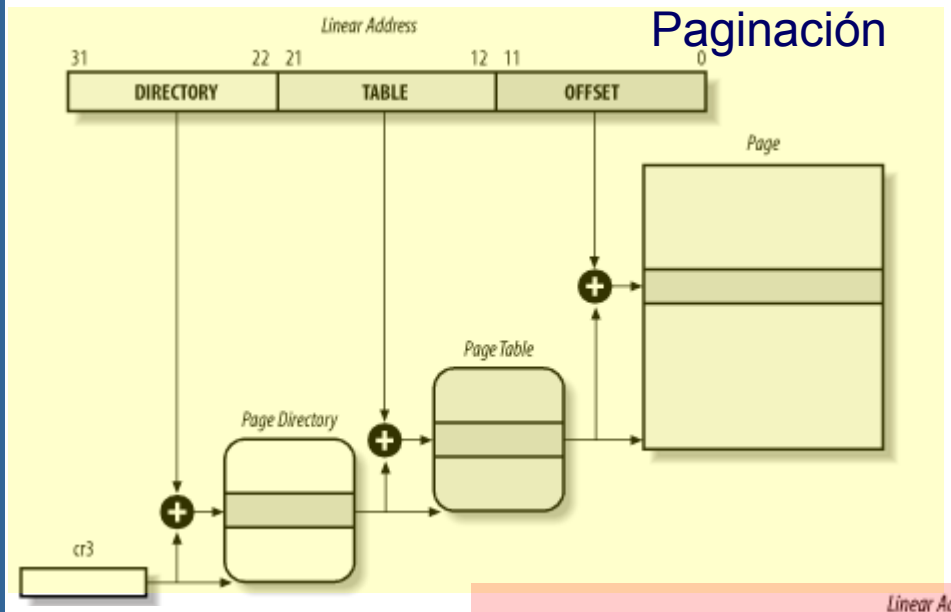
Manejo de Memoria en Linux

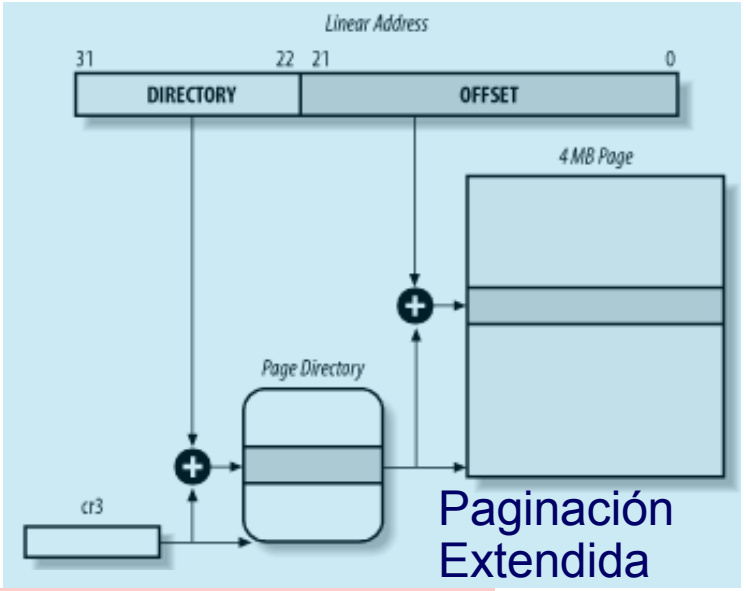
Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 2

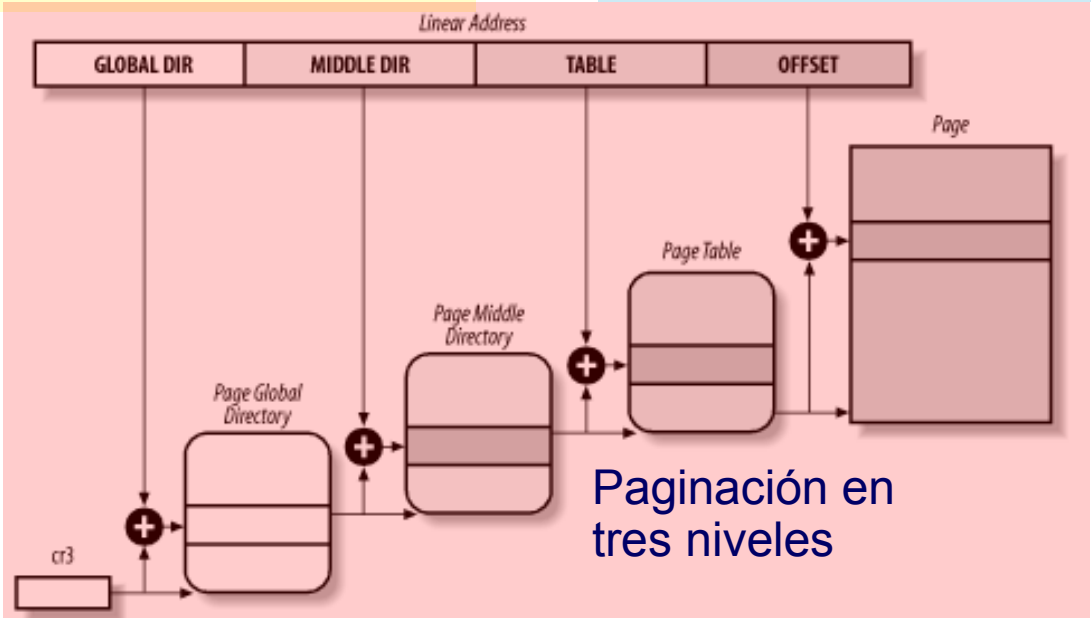
- Paginas de tamaño fijo (4 u 8 KB)
- Definiciones:
  - PAGE = Es el rango de direcciones lineales mapeados dentro de esa página, junto con los datos contendidos por dichas direcciones (En la jerga, “data chunk”).
  - Page Frame = Es el área de memoria que contiene una página, por eso también se la puede encontrar bajo el nombre de physical page, o page container
- Paging Unit. Convierte direcciones lineales en físicas
- Extended Paging. A partir del pentium se tiene la posibilidad de definir páginas de 4Mbytes.
- Three-level paging para Procesadores de 64 bit.

Paginación en Linux (Hardware)

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 2

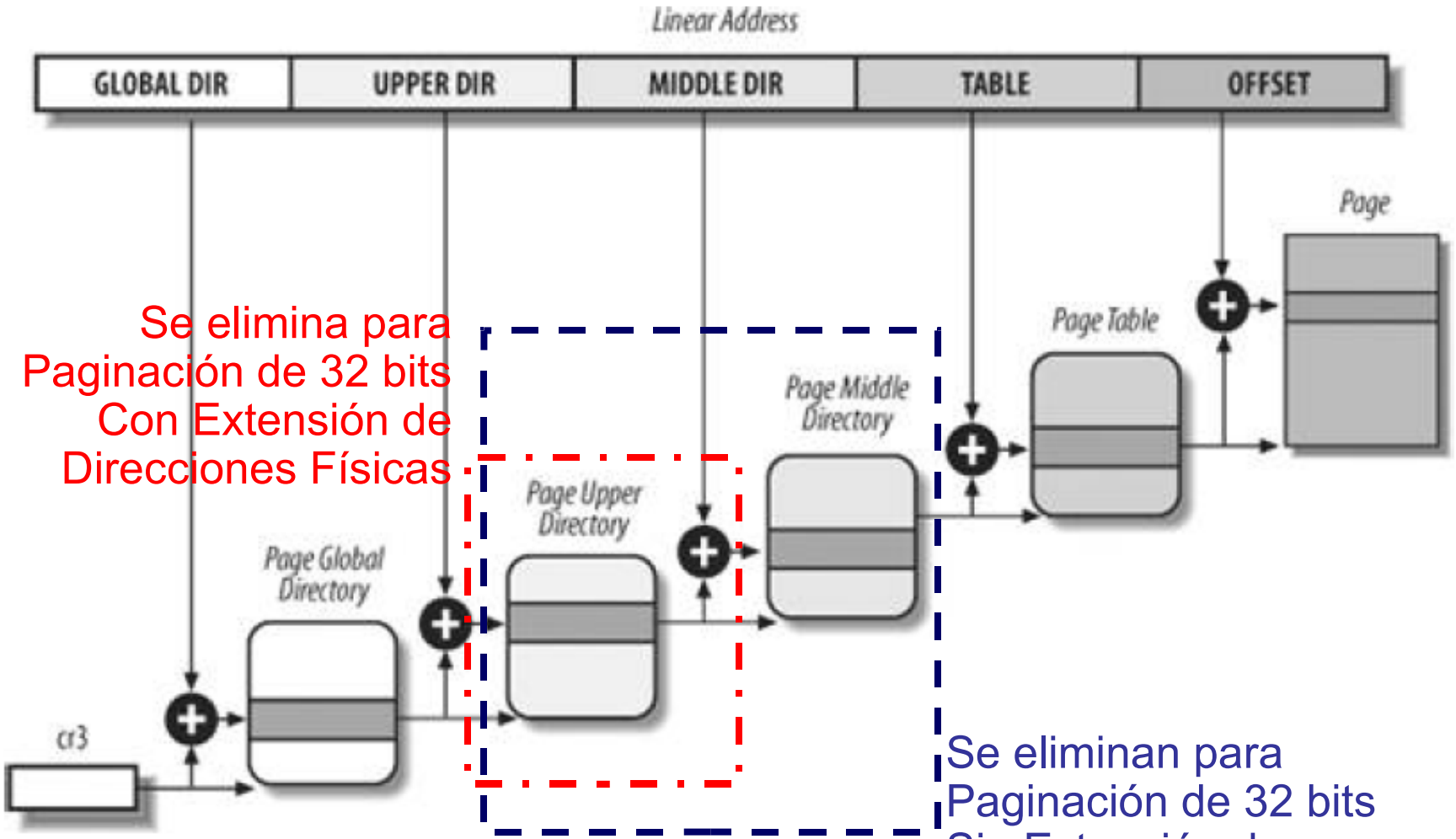






Paginación en LINUX

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 2



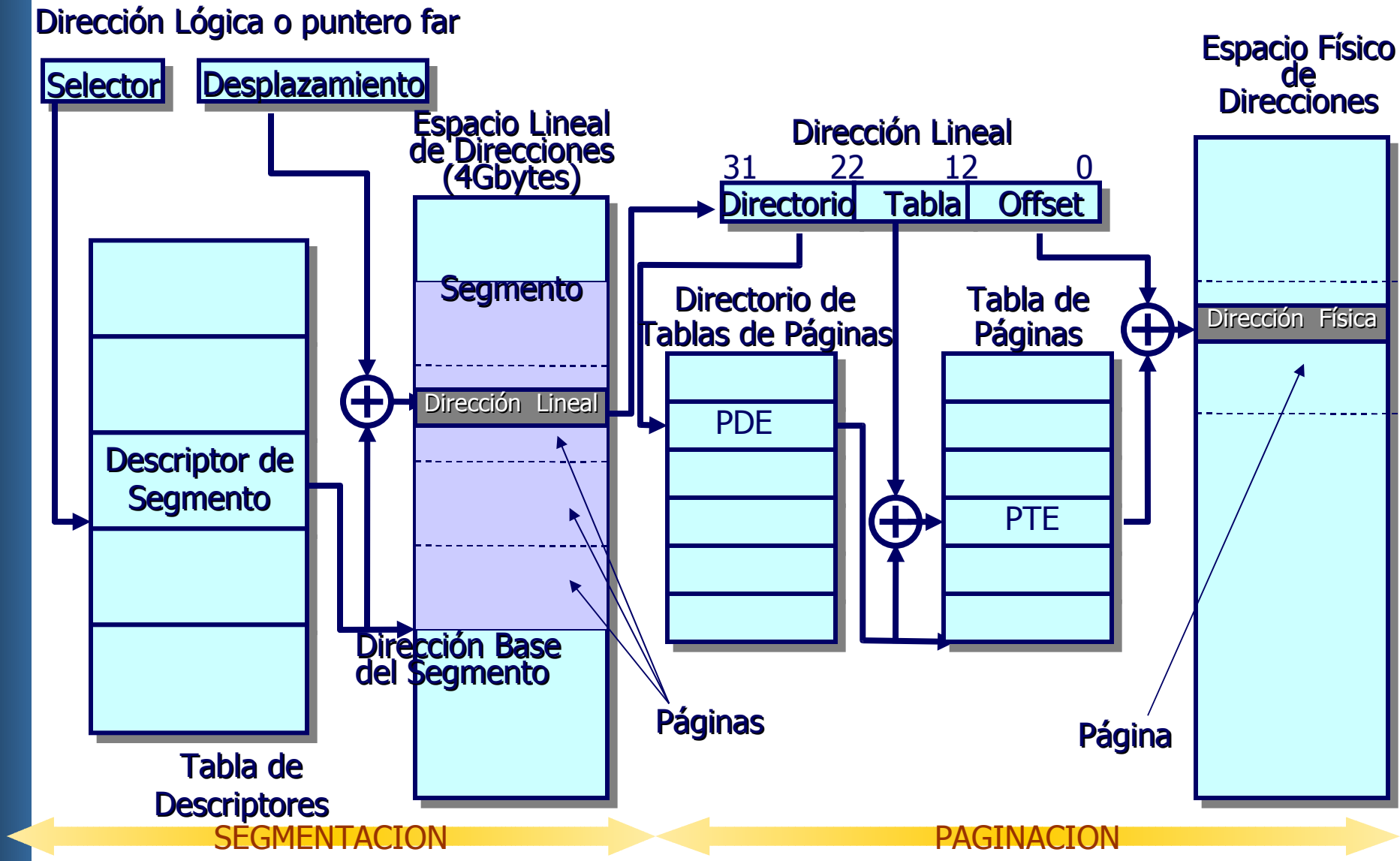
Paginación en LINUX

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 2

Plataforma de Hardware	Tamaño de Página	N° de bits de Direcciones utilizados	N° de niveles de paginación	División de la Dirección Lineal
alpha	8 KB	43	3	10 + 10 + 10 + 13
ia64	4 KB	39	3	9 + 9 + 9 + 12
ppc64	4 KB	41	3	10 + 10 + 9 + 12
sh64	4 KB	41	3	10 + 10 + 9 + 12
x86_64	4 KB	48	4	9 + 9 + 9 + 9 + 12

Segmentación y Paginación de Memoria

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4



Direccionamiento físico de 36 bits

Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4

- Pentium Pro. Introduce el mecanismo conocido como Physical Address Extention (PAE).
  - Habilita la generación de direcciones físicas de 36 bits
  - El procesador tiene los 4 pines adicionales
  - Se administra mediante paginación
- Secuencia de habilitación
  - Setear PG (bit 31 de CR0)
  - Setear PAE (bit 5 de CR4)
- Soporta dos tamaños de páginas
  - 4 Kbytes
  - 2 Mbytes

Direccionamiento físico de 36 bits

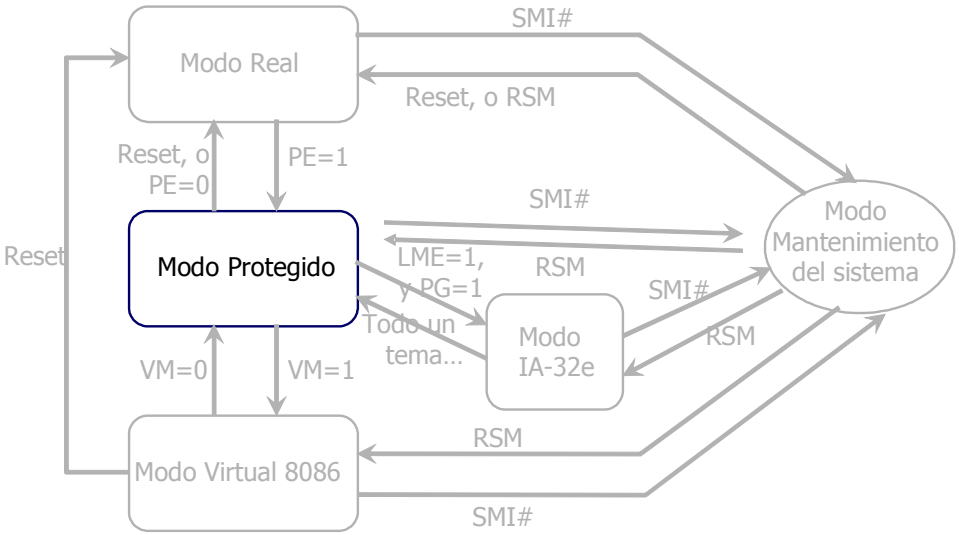
Ref: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 4

- Cambios en las estructuras de paginación
  - PTE pasa de 32 a 64 bits. Hay 512 PTE's, por tabla
  - Page Directory Pointer Table
    - Tabla de 4 entradas.
    - Cada entrada es un puntero a un Page Directory Table
  - CR3 apunta a la Page Directory Pointer Table. Contiene los 27 bits mas significativos de la dirección forzando a esta tabla a comenzar alineada a 32 bytes

CR3

Modo Protegido

Gestión de interrupciones



Interrupciones en Modo Protegido

- En cualquiera de los modos de trabajo del procesador, las interrupciones se identifican mediante un número de un byte llamado ***tipo***.
  - Acompaña al código de operación en la instrucción INT ***type***, para el caso de las interrupciones por software, por ejemplo INT 80h
  - Es provisto por el hardware en interrupciones que ingresan por el pin ***INTR*** del procesador.
  - Está asociado al pin ***NMI*** (***tipo*** 2)
  - Está asociado a una instrucción (***INTO***, por ejemplo, genera una interrupción de ***tipo*** 4).
- Tiene 256 tipos diferentes de interrupción.
- El sistema de interrupciones fundado por el 8086, es en este sentido idéntico en sus sucesores, hasta los procesadores Core2Duo actuales.

Interrupciones en Modo Protegido

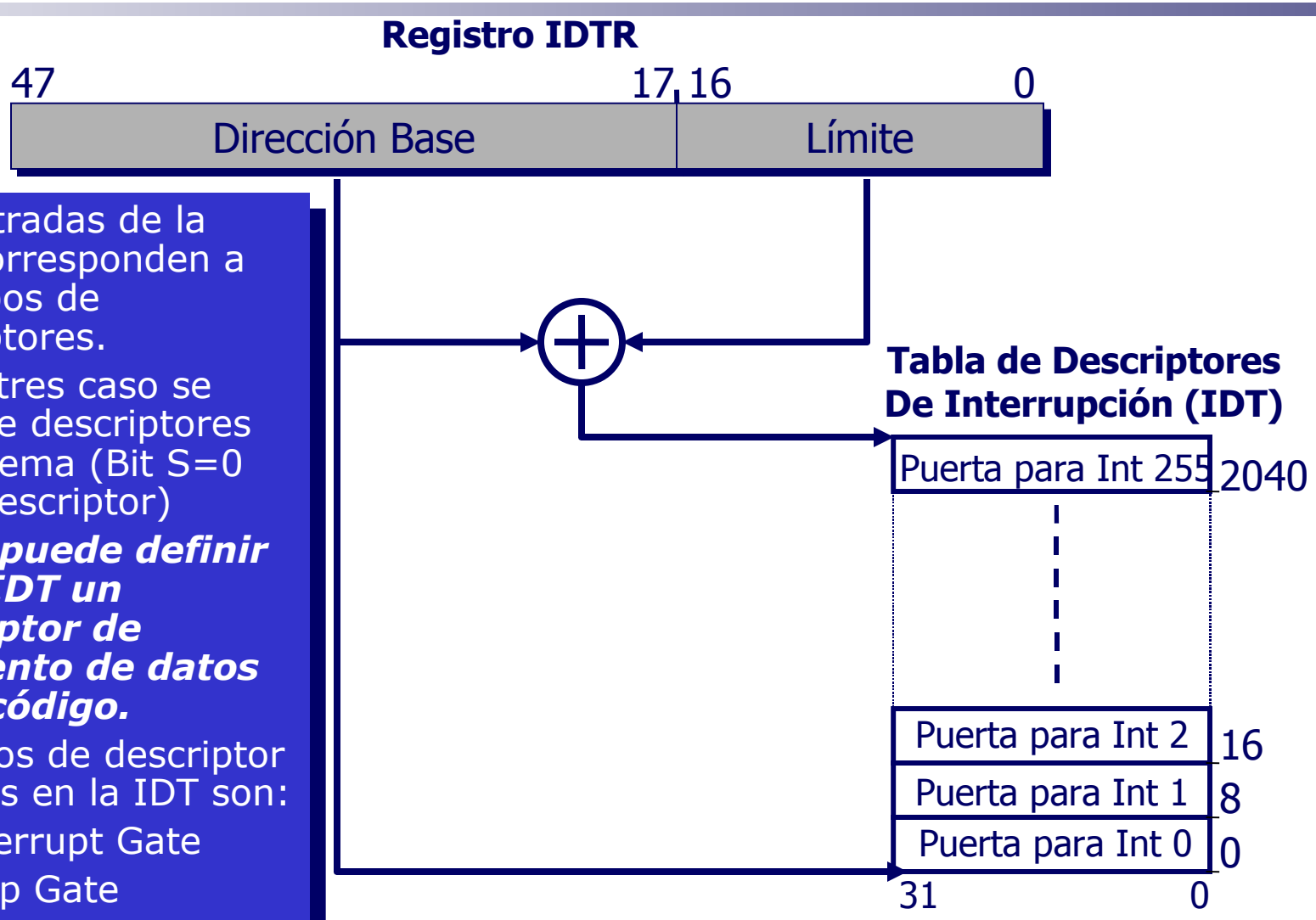
- En modo protegido existe una tabla llamada IDT (Interrupt Descriptor Table), que a diferencia del Modo Real, no almacena vectores de interrupción, sino que como su nombre lo indica almacena Descriptores.

- Esta tabla tiene únicamente 256 entradas, ya que esa es la cantidad de tipos de interrupciones diferentes que maneja el microprocesador.

Interrupciones en Modo Protegido

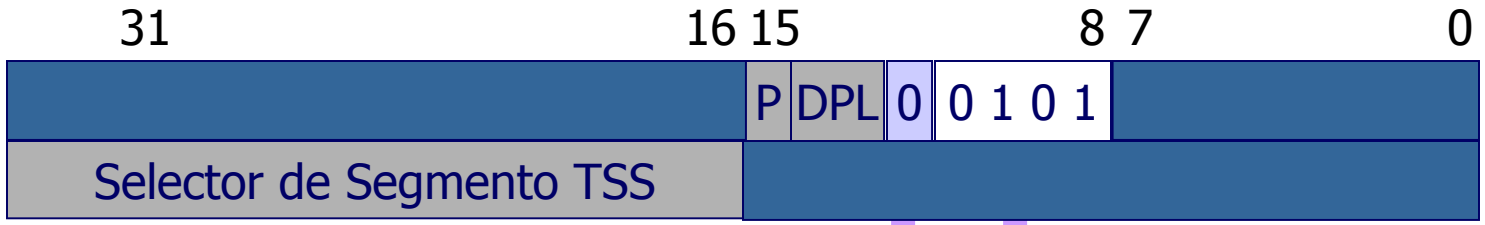
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 6

- Las entradas de la IDT, corresponden a tres tipos de descriptores.
- En los tres caso se trata de descriptores de Sistema (Bit S=0 en el descriptor)
- ***No se puede definir en la IDT un descriptor de segmento de datos ni de código.***
- Los tipos de descriptor posibles en la IDT son:
  - Interrupt Gate
  - Trap Gate
  - Task Gate

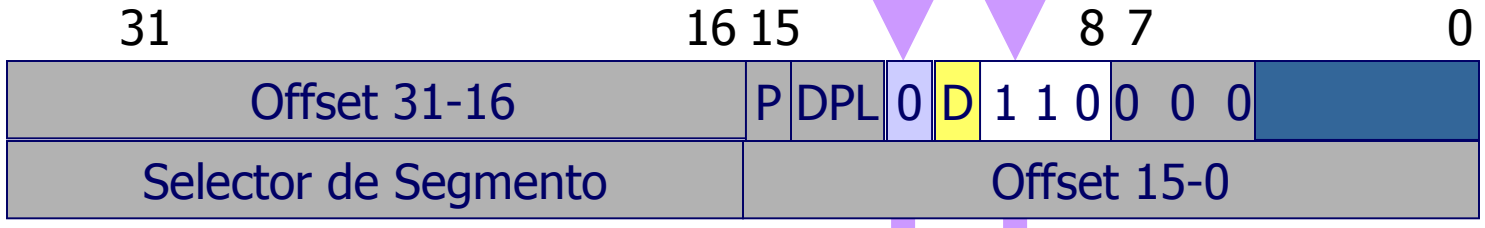


Tipos de descriptores en la IDT:

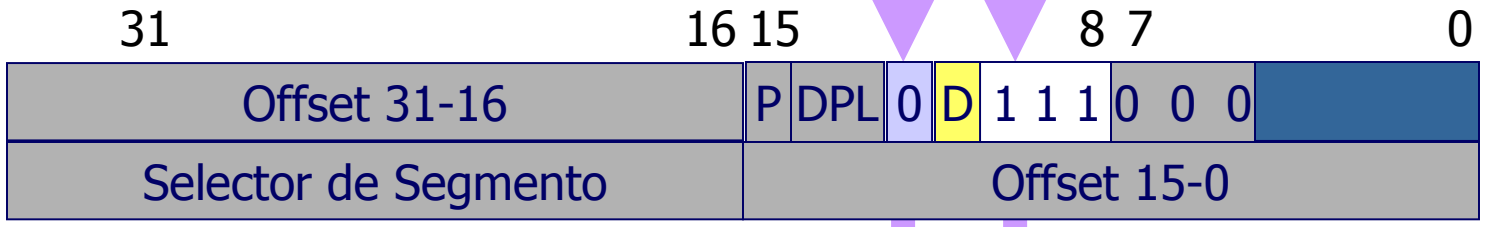
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 6



Descriprtor de Segmento de Puerta de Tarea



Descriprtor de Segmento de Puerta de Interrupción



Descriprtor de Segmento de Puerta de Excepción (o T rap)

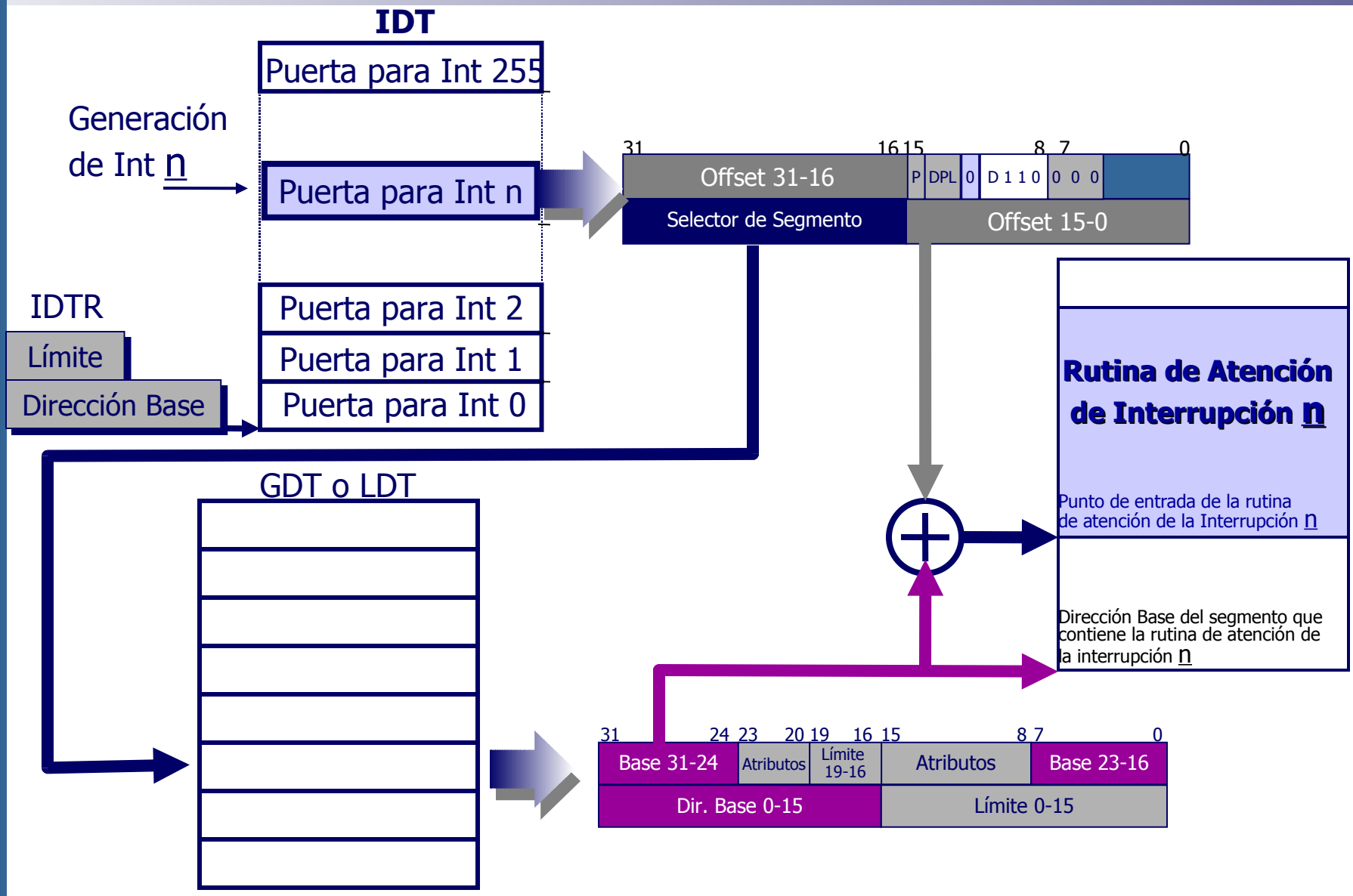
Campo Tipo. Predefinido para cada descriptor

D : Tipo de Puerta
0: 16bits
1: 32bits

0 Bit S=0 => DESCRIPTOR DE SISTEMA

Procedimiento de Interrupción

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 6



Tipos de Interrupciones predefinidos

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 6

Vector No.	Mne-monic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	#DB	RESERVED	Fault/Trap	No	For Intel use only.
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT 3 instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD2 instruction or reserved opcode.[1]
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (Zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9		Coprocessor Segment Overrun (reserved)	Fault	No	Floating-point instruction.[2]
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.
12	#SS	Stack-Segment Fault	Fault	Yes	Stack operations and SS register loads.
13	#GP	General Protection	Fault	Yes	Anymemory reference and other protection checks.
14	#PF	Page Fault	Fault	Yes	Any memory reference.
15	—	(Intel reserved. Do not use.)	No		
16	#MF	x87 FPU Floating-Point Error (Math Fault)	Fault	No	x87 FPU floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Fault	Yes (Zero)	Any data reference in memory.[3]
18	#MC	Machine Check	Abort	No	Error codes (if any) and source are model dependent.[4]
19	#XF	SIMD Floating-Point Exception	Fault	No	SSE/SSE2/SSE3 floating-point instructions[5]
20-31	—	Intel reserved. Do not use.			
32-255	—	User Defined (Nonreserved) Interrupts	Interrupt		External interrupt or INT n instruction.

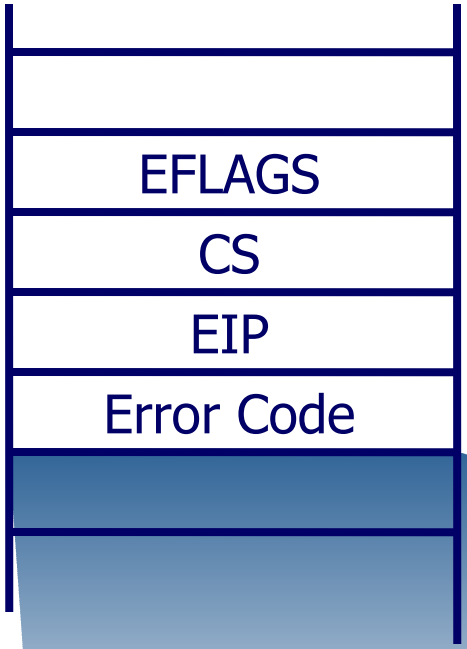
Notas:
1. La instrucción UD2 fue introducida con el procesador Pentium Pro.
2. Los procesadores IA-32 posteriores al Intel386 no generan esta excepción.
3. Esta excepción fue introducida en el procesador Intel486.
4. Esta excepción fue introducida en el procesador Pentium y mejorada en la familia de procesadores P6.
5. Esta excepción fue introducida en el procesador Pentium III.

Tipos de Interrupciones predefinidos

- ***Fault***: Excepción que puede corregirse permitiendo al programa retomar la ejecución de esa instrucción sin perder continuidad. El procesador guarda en la pila la dirección de la instrucción que produjo la falla.
- ***Traps***: Excepción producida inmediatamente a continuación de una instrucción de trap. Algunas permiten al procesador retomar la ejecución sin perder continuidad. Otras no. El procesador guarda en la pila la dirección de la instrucción a ejecutarse luego de la instrucción trapeada.
- ***Aborts***: Excepción que no siempre puede determinar la instrucción que la causó, ni permite recuperar la ejecución de la tarea que la causó. Reporta errores severos de hardware o inconsistencias en tablas del sistema.

Manejo de la Pila sin cambio de nivel de privilegio

Stack del procedimiento Interrumpido (es el stack que utiliza el handler de ininterrupción)



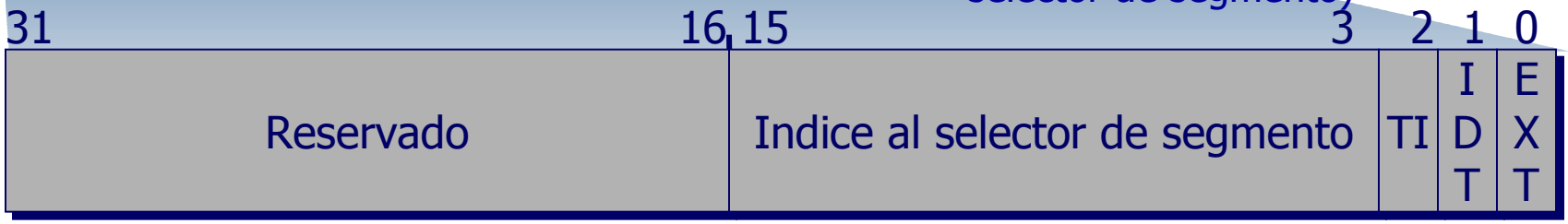
ESP antes de transferir el control al handler de Interrupción

ESP después de transferir el control al handler de Interrupción

EXT: External Event (bit 0): Se setea para indicar que la excepción ha sido causada por un evento externo al procesador

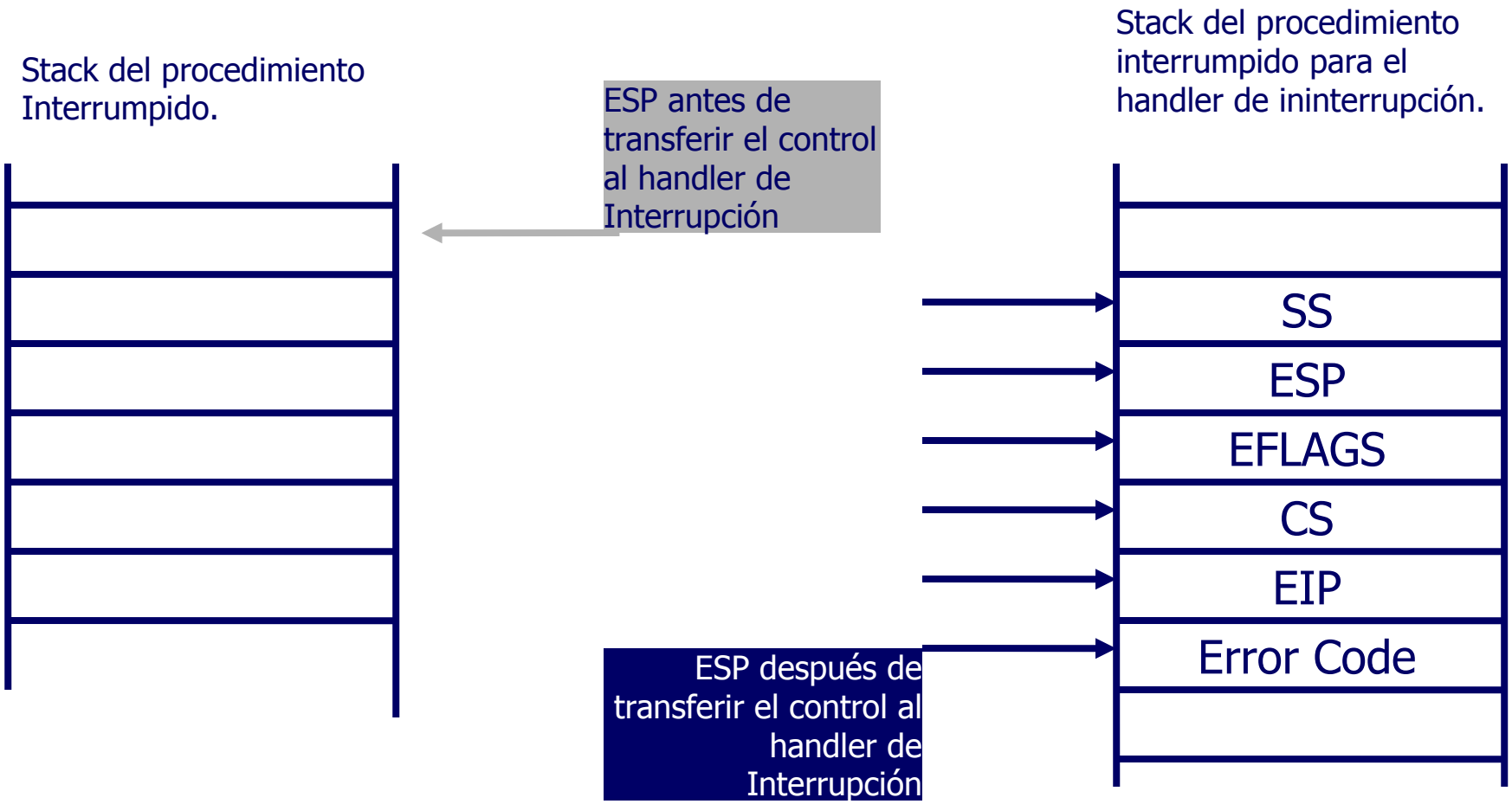
IDT: Descriptor Location (bit 1): Cuando está seteado indica que el campo Segment Selector Index se refiere a un descriptor de puerta en la ID: Cuando está en cero indica que dicho campo se refiere a un descriptor en la GDT o en la LDT de la tarea actual.

TI: GDT/LDT (bit 2): Tiene significado cuando el bit anterior está en cero. Indica a que tabla de descriptores corresponde el selector del campo Indice. 0 GDT , 1 LDT (idéntico significado que en el selector de segmento)



Manejo de la Pila con cambio de nivel de privilegio

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 6



Interrupciones en los procesadores x86

Notas:

- La instrucción UD2 fue introducida con el procesador Pentium Pro.
- Los procesadores IA-32 posteriores al Intel386 no generan esta excepción.
- Esta excepción fue introducida en el procesador Intel486.
- Esta excepción fue introducida en el procesador Pentium y mejorada en la familia de procesadores P6.
- Esta excepción fue introducida en el procesador Pentium III.

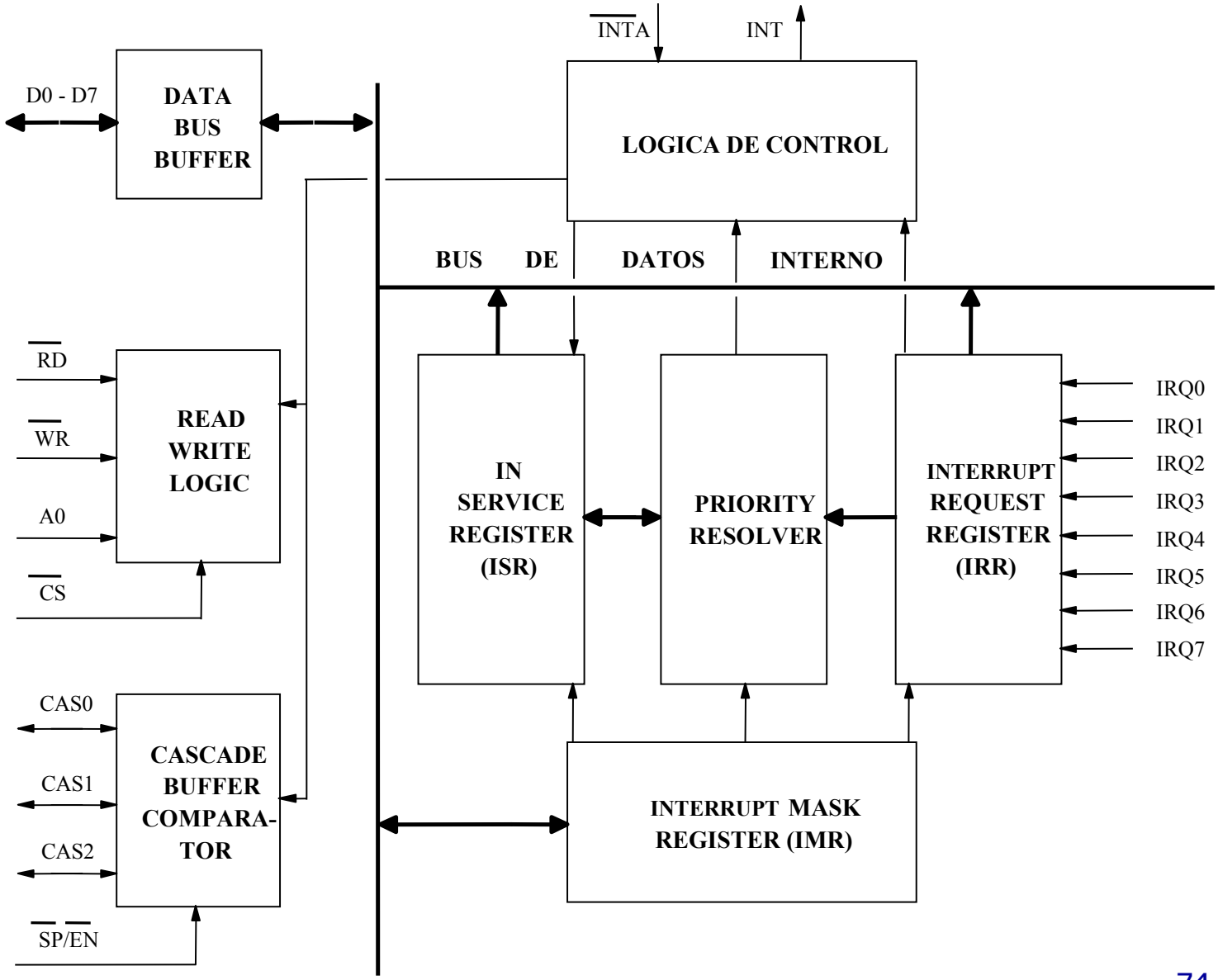
Tipo	Mnemonico	Descripción	Origen
0	#DE	Divide Error	Instrucciones DIV e IDIV.
1	#DB	Debug	Cualquier referencia a código o datos.
2	NMI	Interrupt	Interrupción Externa No Enmascarable.
3	#BP	Breakpoint	Instrucción INT 3.
4	#OF	Overflow	Instrucción INTO.
5	#BR	BOUND Range Exceeded	Instrucción BOUND.
6	#UD	Invalid Opcode (UnDefined Opcode)	Instrucción UD2[1] u opcodes reservados.
7	#NM	Device Not Available (No Math	Instrucción Floating-point o WAIT/FWAIT.
8	#DF	Double Fault	Cualquier instruction que pueda generar una excepción, una NMI, o una INTR.
9	#MF	CoProcessor Segment Overrun (reserved)	Instruction Floating-point.[2]
0A	#TS	Invalid TSS	Conmutación de Tareas o accesos a TSS.
0B	#NP	Segment Not Present	Carga de selectores en registros de segmento o accesos a segmentos de sistema.
0C	#SS	Stack Segment	Fallas en operaciones de Stack y en cargas de selectores en el registro SS.
0D	#GP	General Protection	Cualquier referencia a memoria y otros chaqueos de protección
0E	#PF	Page Fault	Cualquier referencia a memoria.
0F		Reservada	
10	#MF	Floating-Point Error (Math Fault)	Instrucciones Floating-point o WAIT/FWAIT.
11	#AC	Alignment Check	Cualquier referencia a datos en memoria.[3]
12	#MC	Machine Check	Los códigos de error (si lo hubiere) y el origen son modelo dependiente.[4]
13	#XF	SIMD Floating-Point Exception[5]	Instrucción Floating-Point que utilice SIMD
14		Reservada	
15		Reservada	
16		Reservada	
17		Reservada	
18		Reservada	
19		Reservada	
1A		Reservada	
1B		Reservada	
1C		Reservada	
1D		Reservada	
1E		Reservada	
1F		Reservada	
20-31		Reservada para fab. Hardware y S.O.	
32-255		Maskable Interrupts	Interrupciones Externas desde el terminal INTR o por la instrucción INT *n*.

Autor: Alejandro Fu

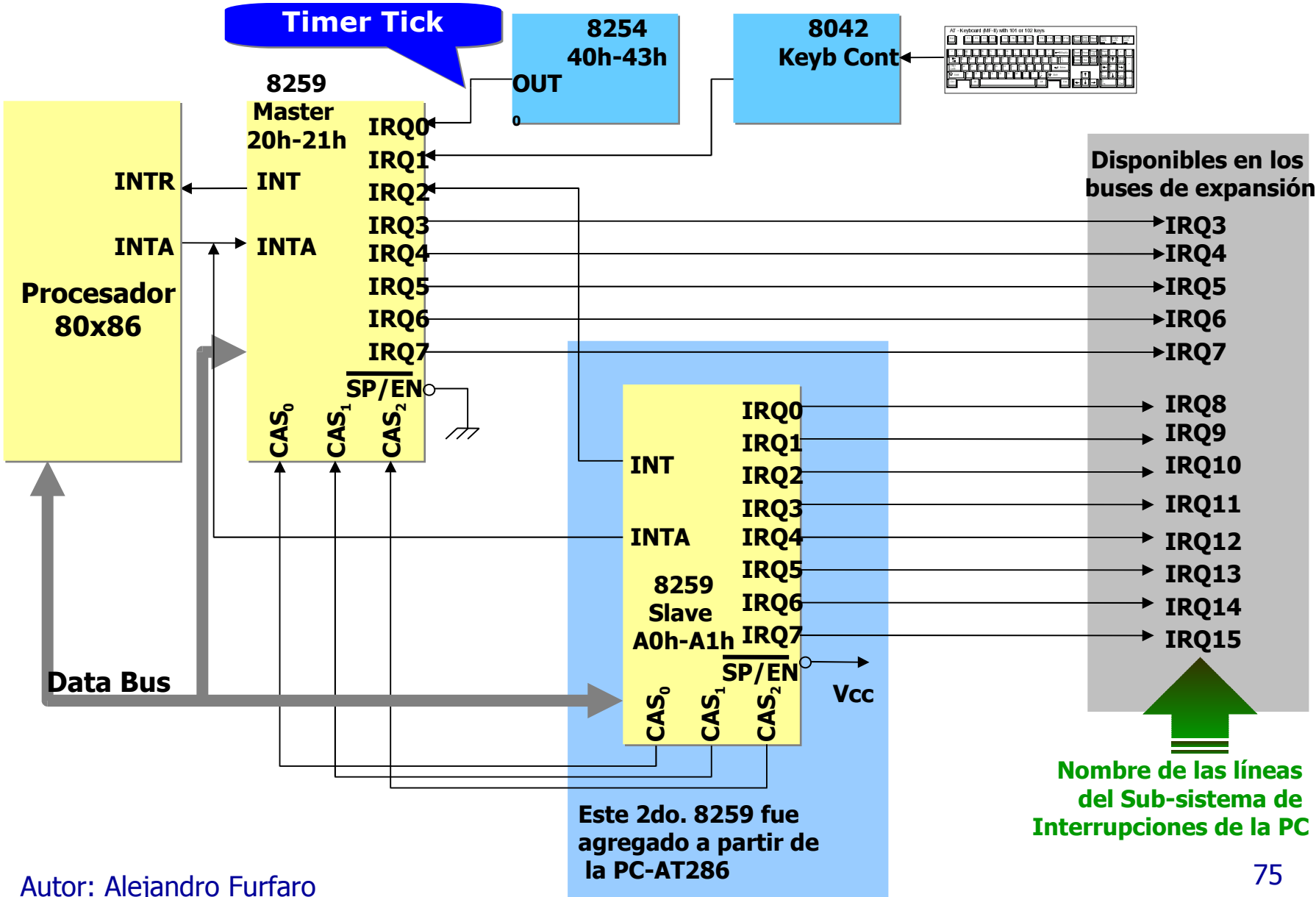
Interrupciones en la PC

Tipo	Mnemonico	Descripción	Origen	Asiganción en la PC
0	#DE	Divide Error	Instrucciones DIV e IDIV.	No se usa.
1	#DB	Debug	Cualquier referencia a código o datos.	No se usa.
2	NMI	Interrupt	Interrupción Externa No Enmascarable.	No se usa.
3	#BP	Breakpoint	Instrucción INT 3.	No se usa.
4	#OF	Overflow	Instrucción INTO.	No se usa.
5	#BR	BOUND Range Exceeded	Instrucción BOUND.	Print Screen.
6	#UD	Invalid Opcode (UnDefined Opcode)	Instrucción UD2[1] u opcodes reservados.	No se usa.
7	#NM	Device Not Available (No Math	Instrucción Floating-point o WAIT/FWAIT.	No se usa.
8	#DF	Double Fault	Cualquier instruction que pueda generar una excepción, una NMI, o una INTR.	Hardware IRQ0 (timer tick) *
9	#MF	CoProcessor Segment Overrun (reserved)	Instruction Floating-point.[2]	Hardware IRQ1 (teclado)
0A	#TS	Invalid TSS	Conmutación de Tareas o accesos a TSS.	Hardware IRQ2 (reservada) **
0B	#NP	Segment Not Present	Carga de selectores en registros de segmento o accesos a segmentos de sistema.	Hardware IRQ3 (COM2)
0C	#SS	Stack Segment	Fallas en operaciones de Stack y en cargas de selectores en el registro SS.	Hardware IRQ4 (COM1)
0D	#GP	General Protection	Cualquier referencia a memoria y otros chaqueos de protección	Hardware IRQ5 (disco fijo)
0E	#PF	Page Fault	Cualquier referencia a memoria.	Hardware IRQ6 (disco flexible)
0F		Reservada		Hardware IRQ7 (impresora)
10	#MF	Floating-Point Error (Math Fault)	Instrucciones Floating-point o WAIT/FWAIT.	Servicio de Video.
11	#AC	Alignment Check	Cualquier referencia a datos en memoria.[3]	Información del equipo.
12	#MC	Machine Check	Los códigos de error (si lo hubiere) y el origen son modelo dependiente.[4]	Memory size.
13	#XF	SIMD Floating-Point Exception[5]	Instrucción Floating-Point que utilice SIMD	Servicio de E/S de disco flexible.
14		Reservada		Servicio de port serie.
15		Reservada		Servicio de Cassette/Network.
16		Reservada		Servicio de teclado.
17		Reservada		Servicio de impresora.
18		Reservada		BASIC.
19		Reservada		Restart system.
1A		Reservada		Servicio de reloj en tiempo real.
1B		Reservada		Control-Break (definida por el usuario).
1C		Reservada		Timer tick (definida por el usuario).
1D		Reservada		Video parameters pointer.
1E		Reservada		Disk parameters pointer.
1F		Reservada		Tabla de caracteres gráficos.
20-31		Reservada para fab. Hardware y S.O.		
32-255		Maskable Interrupts	Interrupciones Externas desde el terminal INTR o por la instrucción INT *n*.	

Controlador de Interrupciones de Hardware: el PIC 8259



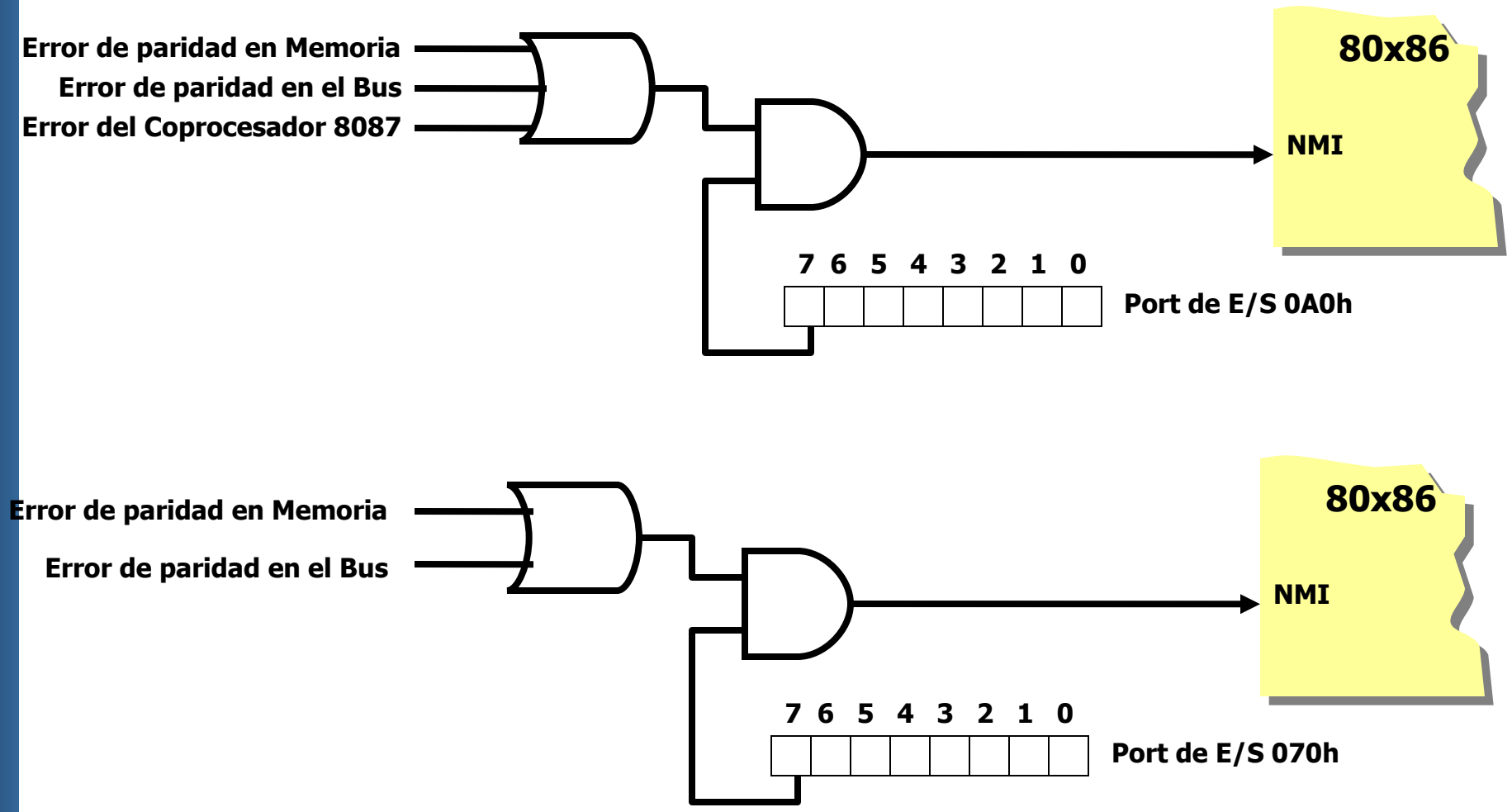
Configuración del PIC 8259 en la PC



Asignación y Tipo de las IRQ's

IRQ	Tipo	Descripción
IRQ0	08h	Timer tick (T=55 mseg.)
IRQ1	09h	Teclado
IRQ2	0Ah	INT desde 8259A esclavo
IRQ8	70h	Servicio de reloj en tiempo real.
IRQ9	71h	Redireccionamiento por soft. a IRQ2
IRQ10	72h	Reservada
IRQ11	73h	Reservada
IRQ12	74h	Reservada.
IRQ13	75h	Coprocesador numérico.
IRQ14	76h	Controlador de disco rígido.
IRQ15	77h	Reservada.
IRQ3	0Bh	COM2
IRQ4	0Ch	COM1
IRQ5	0Dh	LPT2
IRQ6	0Eh	Controlador de disco flexible (Floppy)
IRQ7	0Fh	LPT1

Interrupción No Enmascarable (NMI)



Programación del PIC 8259

- El 8259, presenta al procesador una interfaz de programación a través de dos direcciones de E/S.
  - La primer PC tenía un solo PIC en las direcciones de port 20h y 21h.
  - La PC AT 286, incluyó un segundo PIC, ya que la PC original ya había agotado la asignación de IRQ's, y seguían apareciendo nuevos dispositivos: Placas de red, placas de sonido, etc.
  - Este PIC es accesible en las direcciones de port A0h y A1h.

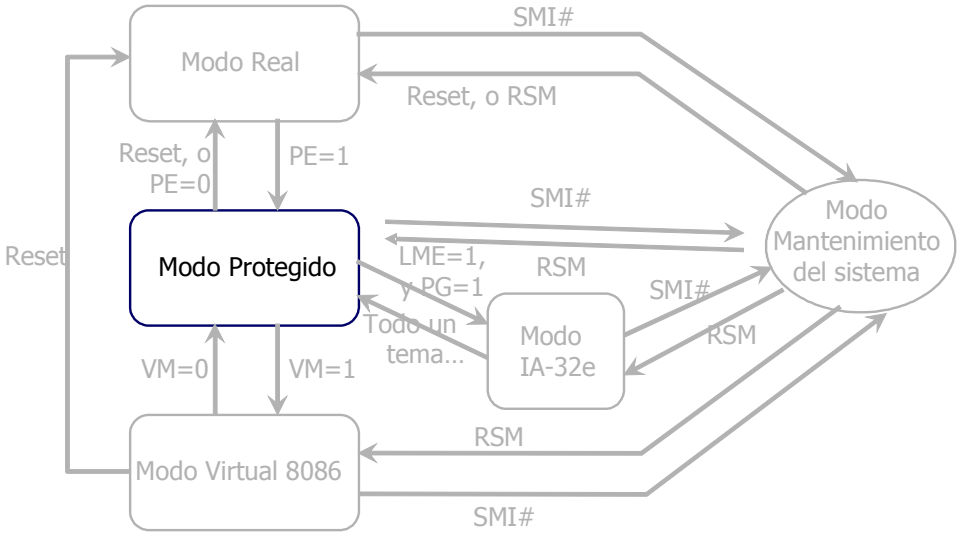
- Palabras de Comando de Inicialización:
  - Son una secuencia de entre dos y cuatro bytes que envía el procesador al 8259A antes de comenzar la operación normal, a fin de configurarlo.
  - A los efectos del PIC 8259, la secuencia de Inicialización es una operación atómica, es decir, que no puede dividirse
  - El 8259 detecta la secuencia de inicialización cuando recibe en la dirección de port par (A0 = 0), una palabra con el bit D4=1. Palabras de Comando de Operación: Una vez inicializado el 8259A, estas palabras le definen diversas operaciones a realizar. Luego de la inicialización, se pueden enviar en cualquier momento.

Programación del PIC 8259

```
; Inicialización PIC #1
        mov al,11h      ;ICW1: IRQs activas por flanco, Modo cascada, ICW4 Si.
        out  20h,al
        mov al,8        ;ICW2: INT base para el PIC N#1 Tipo 8.
        out  21h,al
        mov al,04h      ;ICW3: PIC N#1 Master, tiene un Slave conectado a IRQ2 (0000 0100b)
        out  21h,al
        mov al,01h      ;ICW4: Modo No Buffered, Fin de Interrupción Normal, procesador 8086
        out  21h,al
; Antes de inicializar el PIC N#2, deshabilitamos las Interrupciones del PIC N#1
        mov al,0FFh   :OCW1: Set o Clear el IMR
        out  21h,al
; Inicialización PIC N #2
        mov al,11h      ;ICW1: IRQs activas por flanco, Modo cascada, ICW4 Si.
        out  0A0h,al
        mov al,070h     ;ICW2: INT base para el PIC N#1 Tipo 070h.
        out  0A1h,al
        mov al,02h      ;ICW3: PIC N#2 Slave, IRQ2 es la línea que envía al Master (010b)
        out  0A1h,al
        mov al,01h       ;ICW4: Modo No Buffered, Fin de Interrupción Normal, procesador 8086
        out  0A1h,al
```

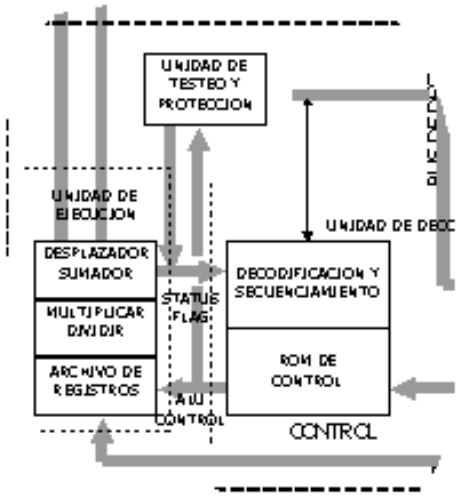
Modo Protegido

Sistema de Protección



Mecanismos de Protección

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5



Cuando el procesador pasa a Modo Protegido se pone en funcionamiento la Unidad de Protección.

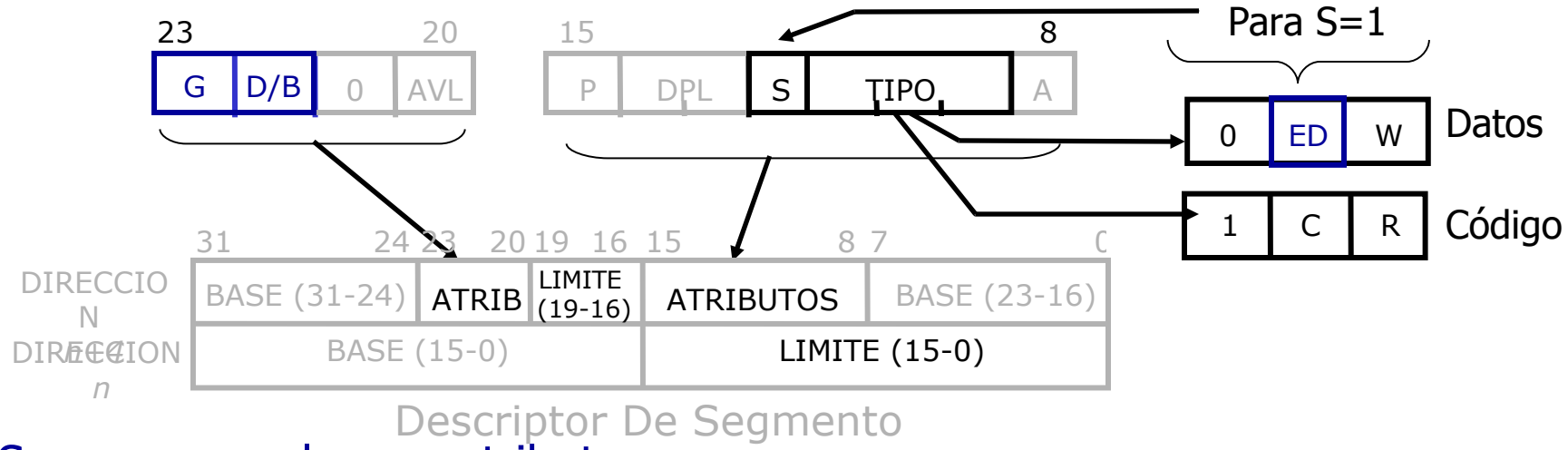
Esta Unidad supervisa las operaciones internas del procesador para la decodificación y ejecución de las instrucciones, comprobando el cumplimiento de una serie de reglas que constituyen el entorno de protección necesario para implementar de manera eficiente la multitarea.

Estas reglas cubren los siguientes aspectos:

- Chequeo del Límite de los segmentos.
- Chequeo del Tipo de los segmentos.
- Chequeo de los niveles de privilegio de segmentos y páginas.
- Restricción del dominio de direccionamiento a las tareas.
- Restricción de los puntos de entrada a los procedimientos.
- Restricción en el uso del set de instrucciones.

Chequeo del Límite

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5



Se apoya en algunos atributos:

- **Bit G:** Determina si el valor de límite se mide en Bytes o en páginas de 4Kbytes. En el primer caso el rango es 0 a FFFFFh, en el segundo es de FFFh a FFFFFFFFh.
- **Bit D/B**: Determina si el segmento es de 16 o 32 bits
- **Bit ED:** Expand Down. Indica el sentido de crecimiento del segmento. Importante en el manejo de pilas

El valor del campo Límite es el tamaño del segmento menos 1, ya que su rango de valores posibles va desde 00000 a FFFFFh

Chequeo del Límite Segmentos con ED = 0.

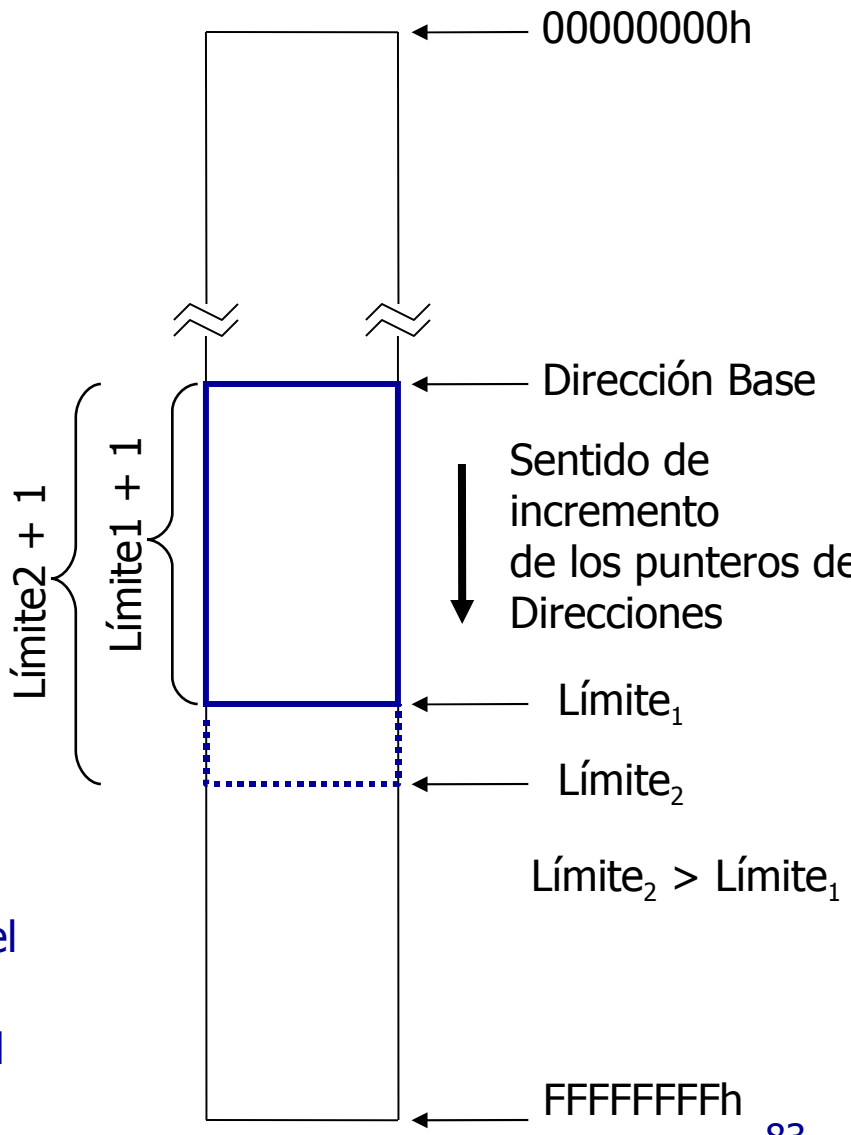
Límite Efectivo: Es la última dirección que se puede acceder dentro del segmento.

Se generará una excepción 0Dh cada vez que se intente acceder a un offset mayor que el **límite efectivo.**

Ejemplos:

- Un byte cuyo offset sea mayor que el valor del **Límite efectivo**
- Una word cuyo offset sea mayor que (**Límite efectivo** – 1)
- Una doubleword cuyo offset sea mayor que (**Límite efectivo** – 3)
- Una quadword cuyo offset sea mayor que (**Límite efectivo** – 7)

Cuando se está por alcanzar el límite de un segmento se lo puede redimensionar aumentando el valor del Límite (en la gráfica, ***límite1*** a ***límite2***), sin necesidad de relocar segmentos. No se afecta al programa en ejecución.



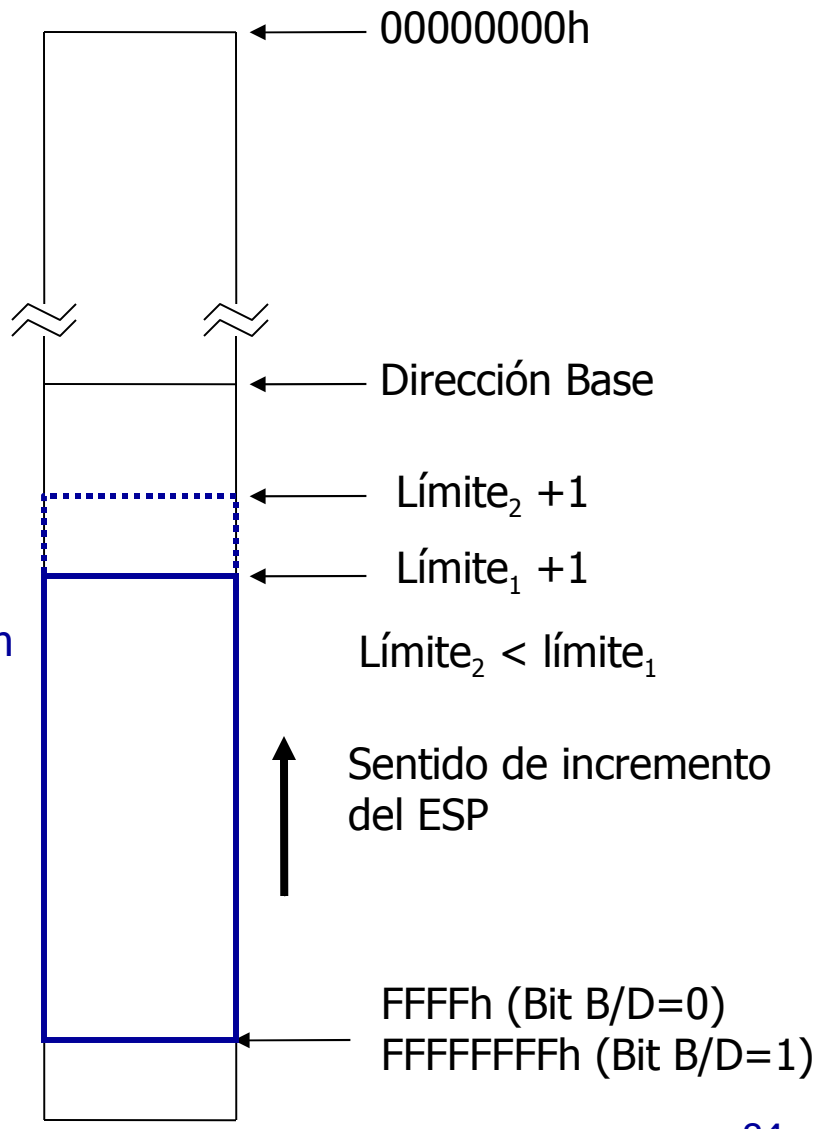
Chequeo del Límite Segmentos con ED = 1.

Límite Efectivo: Es el último offset que no puede ser accedido ya que generará la excepción 0Dh .

Para estos segmentos el rango de offsets válidos va desde:

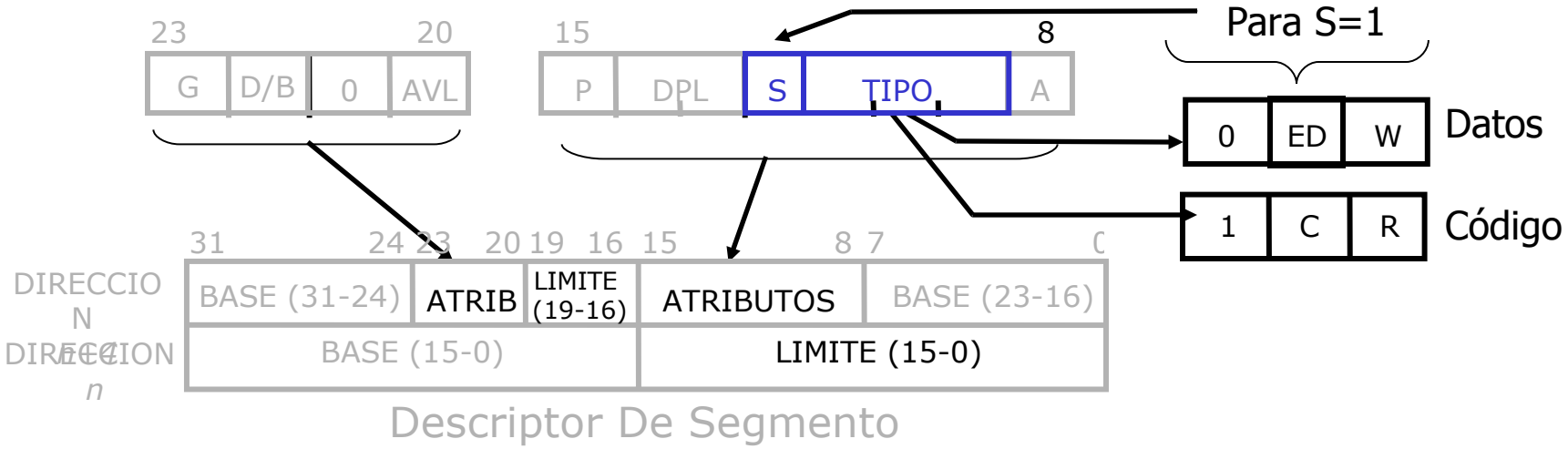
- [límite efectivo +1] hasta 0FFFFh (bit B/D=0)
- [límite efectivo +1] hasta 0FFFFFFFFh (bit B/D=1)

Este tipo de segmentos son útiles cuando contienen pilas. Los punteros de Pila en la arquitectura Intel decementan su valor a medida que la pila se llena.



Chequeo del Tipo

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5



Se toma en cuenta el bit S (System bit) de los atributos del descriptor, y en función de su valor, se comprueba el campo de cuatro bits subsiguiente

Comprobaciones en la carga de un selector.

- No se permite cargar CS con selectores que en la GDT correspondan a descriptores de segmentos (S = 1) que no sean de código
- No se permite cargar un selector de código cuyo descriptor no tenga permiso de lectura en ningún otro registro de segmento (SS, DS, ES, FS, o GS)
- En SS solo se pueden cargar selectores que correspondan a segmentos con permiso de escritura.
- LDTR solo puede ser cargado con un selector de LDT (Bit S=0 y tipo = 0010b)
- TR solo puede ser cargado con selector de TSS (bit S=0 y tipos = 0001b, 0011b, 1001b, 1011b, para TSS de 16 bits, disponible y ocupado, y TSS de 32 bits disponible y ocupado respectivamente)

Chequeo del Tipo

Comprobaciones durante la ejecución de instrucciones que acceden a segmentos cuyo selector ya está cargado.

- No se puede escribir un segmento de código
- No se puede escribir un segmento de datos con W=0
- No se puede leer un segmento de código si R=0

Chequeo del Tipo

Comprobaciones durante la ejecución de instrucciones cuyo operando es un selector de segmento.

- Las instrucciones ***CALL far*** y ***JMP far*** solo pueden acceder a descriptores de segmentos de código (conforming o no conforming), a puertas de llamada, puertas de tarea o Descriptores de TSS.
- LLDT debe tener como operando un descriptor de LDT
- LTR debe tener como operando un descriptor de TSS
- LAR debe tener como operando fuente un descriptor de segmento de código, datos, TSS, LDT, puerta de tarea o puerta de llamada.
- LSL debe tener como operando fuente un descriptor de segmento de código, datos, TSS, o LDT.
- Las entradas de la IDT deben ser puertas de interrupción, de excepción, o de tarea. De otro modo cualquier acceso a dicha tabla con, por ejemplo, la instrucción INT causará una excepción de protección general.

Chequeo del Tipo Comprobaciones misceláneas.

- Cuando se ejecuta un ***CALL far*** o ***JMP far***, se determina el tratamiento de la instrucción chequeando el campo ***tipo*** en el descriptor apuntado por el *selector que está como operando en la instrucción*.
  - Si el tipo de descriptor corresponde a un segmento de código, o una puerta de llamada, se ejecuta una llamada o un salto al segmento de código indicado.
  - Si el tipo del descriptor corresponde a un TSS o a una puerta de tarea, se ejecuta una conmutación de tarea. El procesador automáticamente controlará que el descriptor de segmento apuntado por la puerta de tarea corresponda a un descriptor de TSS

Chequeo del Tipo Comprobaciones misceláneas.

- Si en un ***call far*** el tipo de descriptor corresponde a una puerta de llamada o a una puerta de excepción (trap) o de interrupción que tengan como destino un handler de interrupción o de excepción, se controla que el descriptor de segmento definido en el descriptor de la puerta corresponda a un segmento de código.
- Cuando se retorna de una tarea anidada (mediante una instrucción IRET), el procesador controla que el campo ***previous task link*** en el TSS actual, apunte a un TSS.

Selectores de Segmento Nulo

- Cualquier intento de cargar un selector de segmento Nulo en los registros de segmento CS or SS genera una excepción de protección general (#GP).
- Se puede cargar un selector de segmento nulo en los registros DS, ES, FS, or GS. Sin embargo, al intentar acceder al segmento por medio del registro cargado con el selector de segmento nulo generará una excepción #GP.
- Cargar un registro de segmento de datos con un selector de segmento nulo puede resultar un método útil para detectar accesos a registros de segmento no utilizados o para prevenir accesos no deseados a segmentos de datos.

Niveles de Privilegio: Anillos de Protección

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5

Intel suele representar el modelo de protección en forma de anillos concéntricos.

Los mas internos tienen mayor nivel de privilegio

Los segmentos se ubican en el anillo correspondiente de acuerdo con el valor del campo DPL de su descriptor.

Uso propuesto por Intel:

Anillo 0: Kernel del S.O.

Anillos 1 y 2: Servicios del S.O.

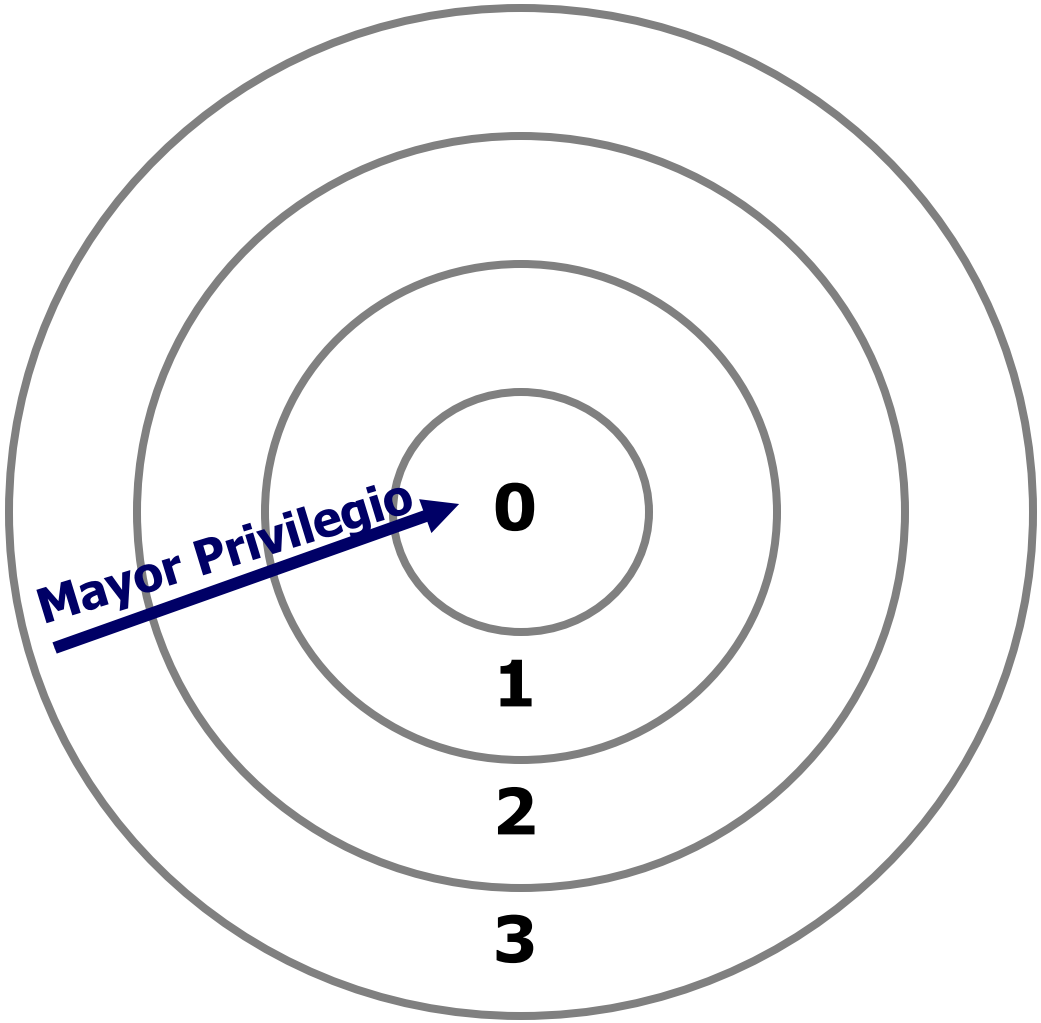
Anillo 3: Aplicaciones

Uso real en los Sistemas Operativos mas difundidos:

Anillo 0: Kernel y servicios del S.O.

Anillos 1 y 2: No utilizados

Anillo 3: Aplicaciones



Niveles de Privilegio

- Los accesos entre segmentos están regidos por reglas que contemplan los niveles de privilegio.
- Los niveles de privilegio se chequean cuando se carga el selector en el registro de segmento.
- Se definen tres tipos de nivel de privilegio

Niveles de Privilegio

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5

- **Current Privilege Level (CPL)**: Nivel de privilegio que tiene el código que está intentando acceder ya sea a un descriptor de segmento, una puerta de llamada o de tarea, etc. El procesador lo mantiene en el cache hidden del selector, ya que lo leyó directo de la tabla de descriptores.
- **Descriptor Privilege Level (DPL)**: Nivel de privilegio del segmento a ser accedido: Puerta de llamada, de tarea, TSS, o Descriptor de segmento de código o datos.
- **Requested Privilege Level (RPL)**: Es el valor que se escribe en los bits 0 y 1 de los selectores. Es ***derogable***, ya que puede sobreescribirse por programa. El procesador lo compara con el CPL y si es numéricamente menor, lo reemplaza por el CPL al momento de chequear el acceso. Si el RPL es mayor (numérico) que el CPL, el procesador usará el RPL. En suma, el procesador usa lo que se define como **Effective Priviledge Level** EPL=MAX(CPL,RPL)

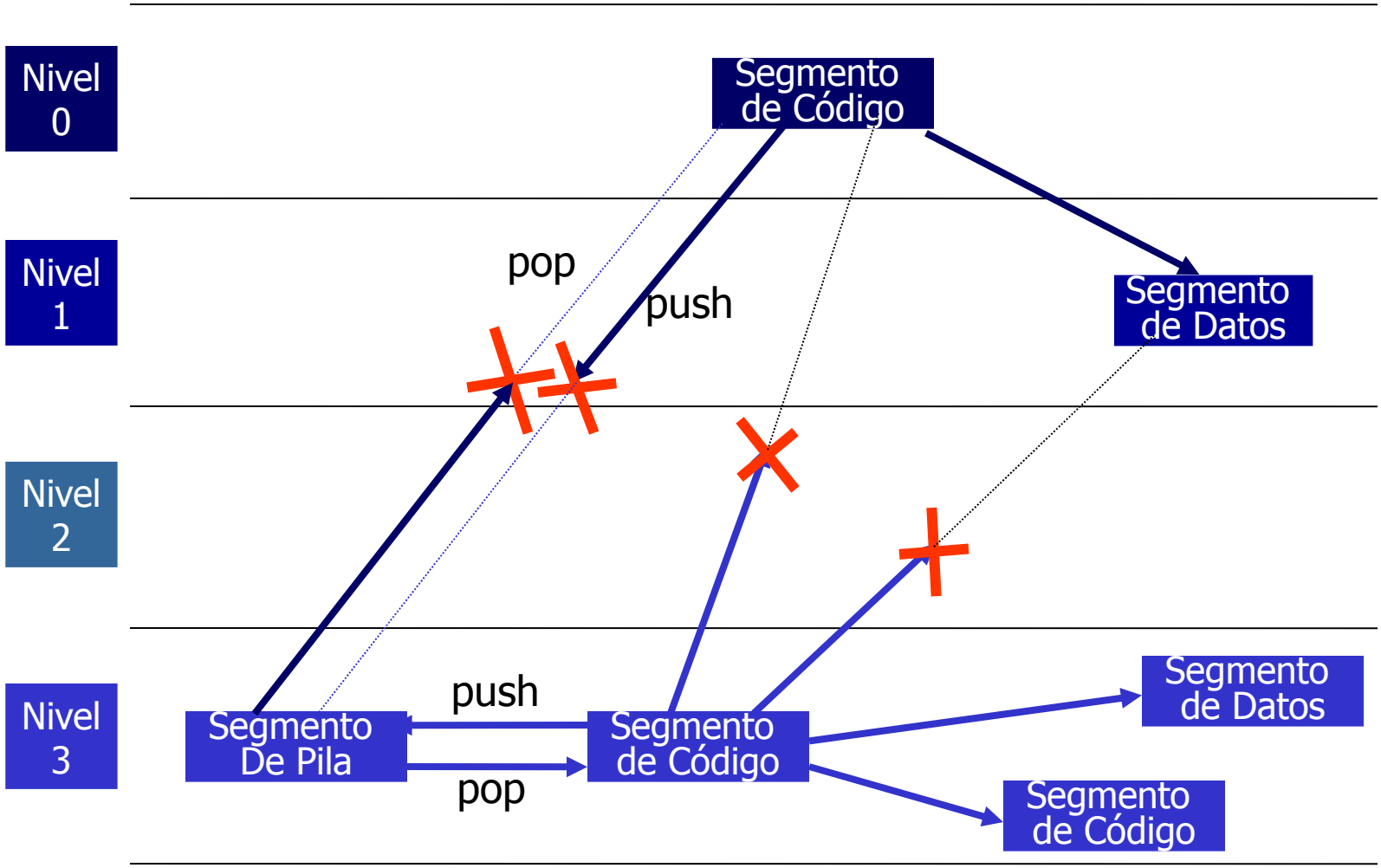
Significado del campo DPL del descriptor

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5

- **Segmento de Datos / Puerta de Llamada / TSS**: Indica el máximo valor numérico (mínimo nivel de privilegio) que debe tener el código de una tarea para acceder a este segmento. Ej.: DPL=01 implica que se puede acceder solamente desde segmentos de código con CPL=00 o CPL=01
- **Segmento de Código No Conforming (sin utilizar puerta de llamada)**: Es el nivel de privilegio que debe tener el código de una tarea para accederlo. Ej.: DPL=00 implica que solo los programas que ejecuten en segmentos con CPL=00.
- **Segmento de código Conforming, o segmento de código no Conforming accedido a través de una puerta de llamada**: Indica el mínimo valor numérico (máximo nivel de privilegio) que debe tener el código que llamó a la puerta de llamada para poder acceder al código apuntado por ésta. Ej.: DPL=10, indica que los programas de anillo 0 y 1 no pueden acceder a este segmento.

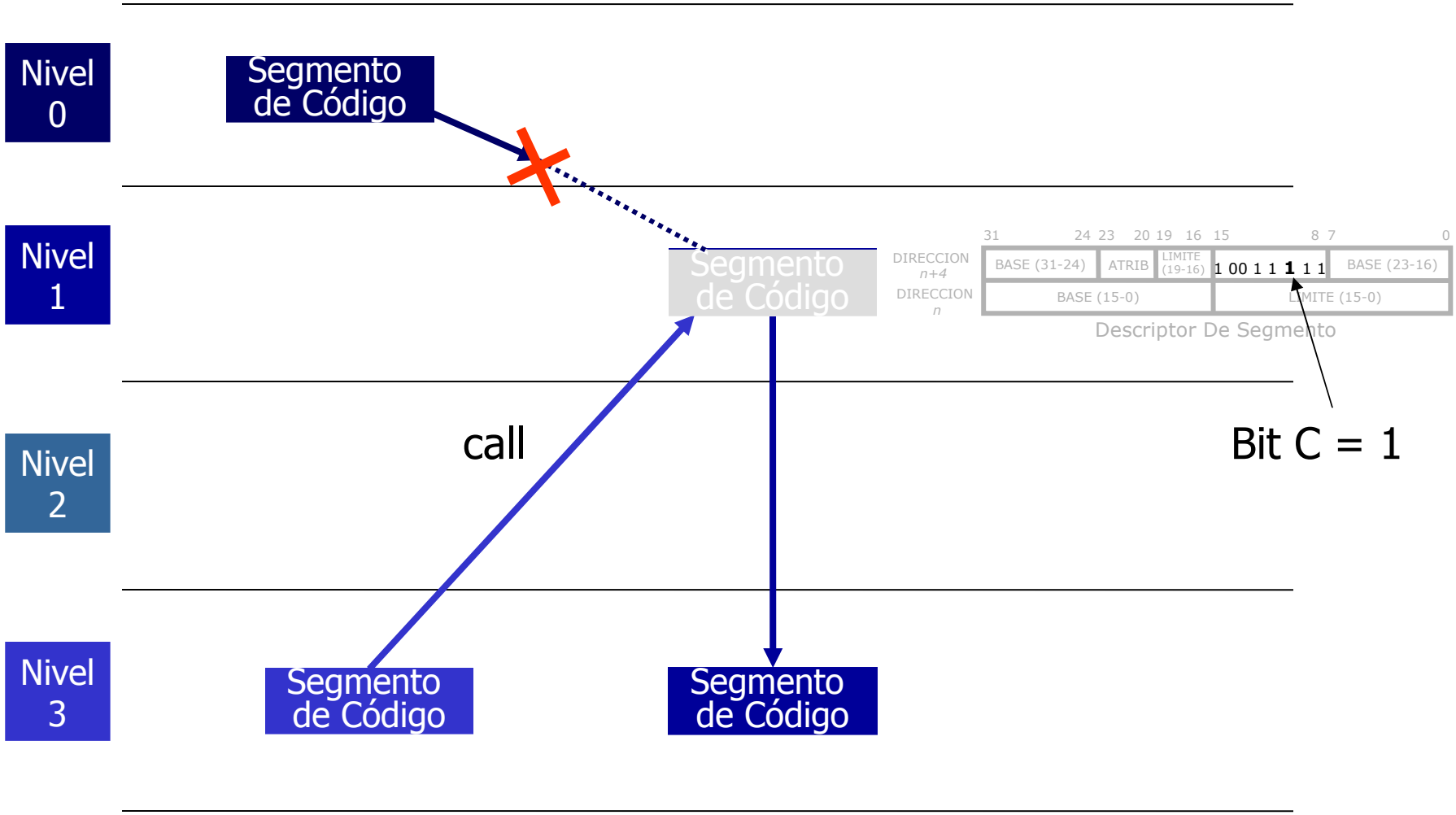
Reglas de Protección entre segmentos

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5



Acceso a un segmento conforming

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5



Puertas de Llamada. Acceso a Segmento de código No conforming

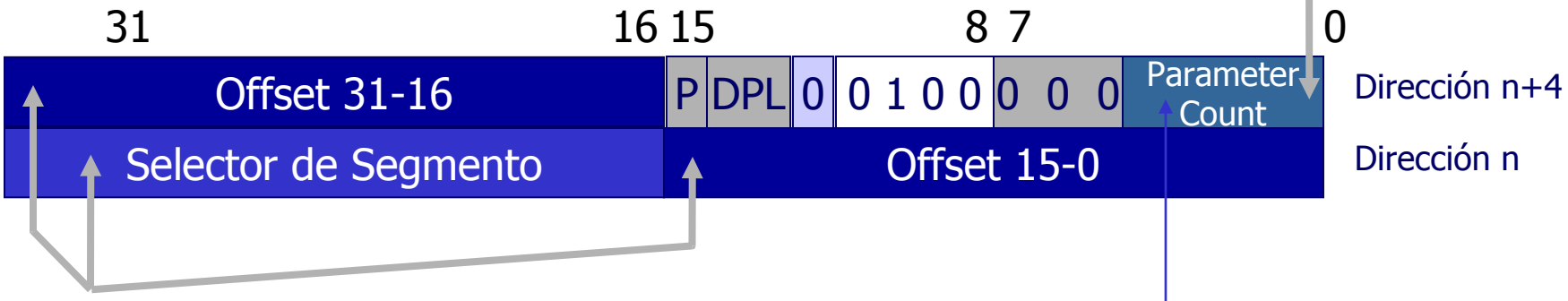
- Son un mecanismo especial para transferir el control desde un segmento de código a otro de mayor nivel de privilegio, cuando se encuentra habilitado el Sistema de Protección del procesador.
- Se basan en un descriptor de sistema (bit S=0), que debe residir en la GDT o en la LDT (nunca en la IDT)
- Se acceden efectuando un CALL o un JMP a una dirección far (dirección lógica), en la que el selector corresponde a un descriptor de Puerta de Llamada, y el offset es ignorado por el procesador.

Descriptor de Puerta de Llamada

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5

Cantidad de parámetros a transferir por la pila (de tipo word por trabajar con segmentos de 16 bits)

Descriptor de Segmento de Puerta de Llamada de 16 bits



Dirección Lógica del Punto de entrada al código de nivel de privilegio superior

2 Words

Llamada en un programa C	Assembly generado por el compilador
servicio (arg1,arg2) →	push arg2 push arg1 call _servicio

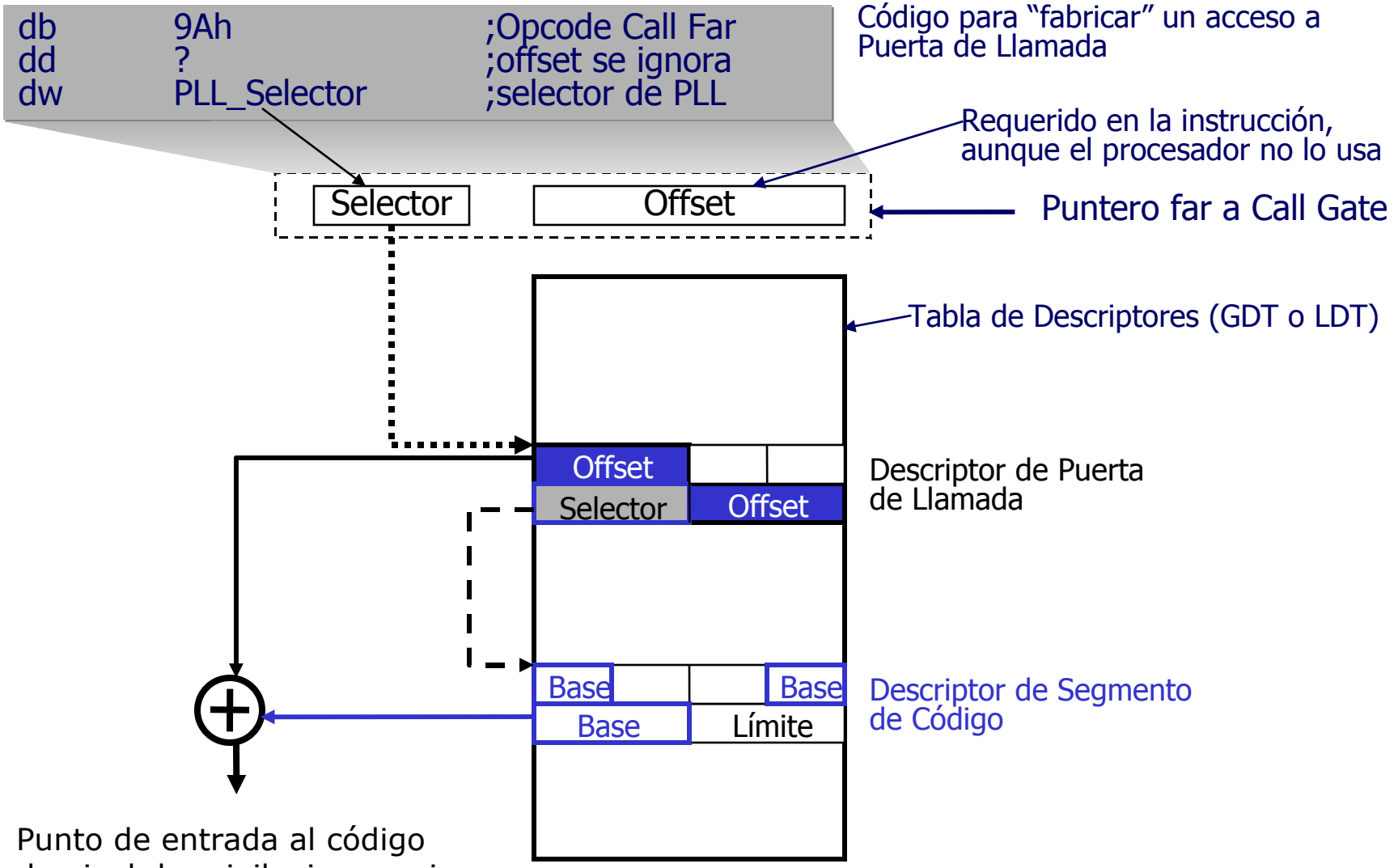
Puerta de Llamada: Acceso al código privilegiado

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5

```
db      9Ah             ;Opcode Call Far
dd      ?               ;offset se ignora
dw      PLL_Selector    ;selector de PLL
```

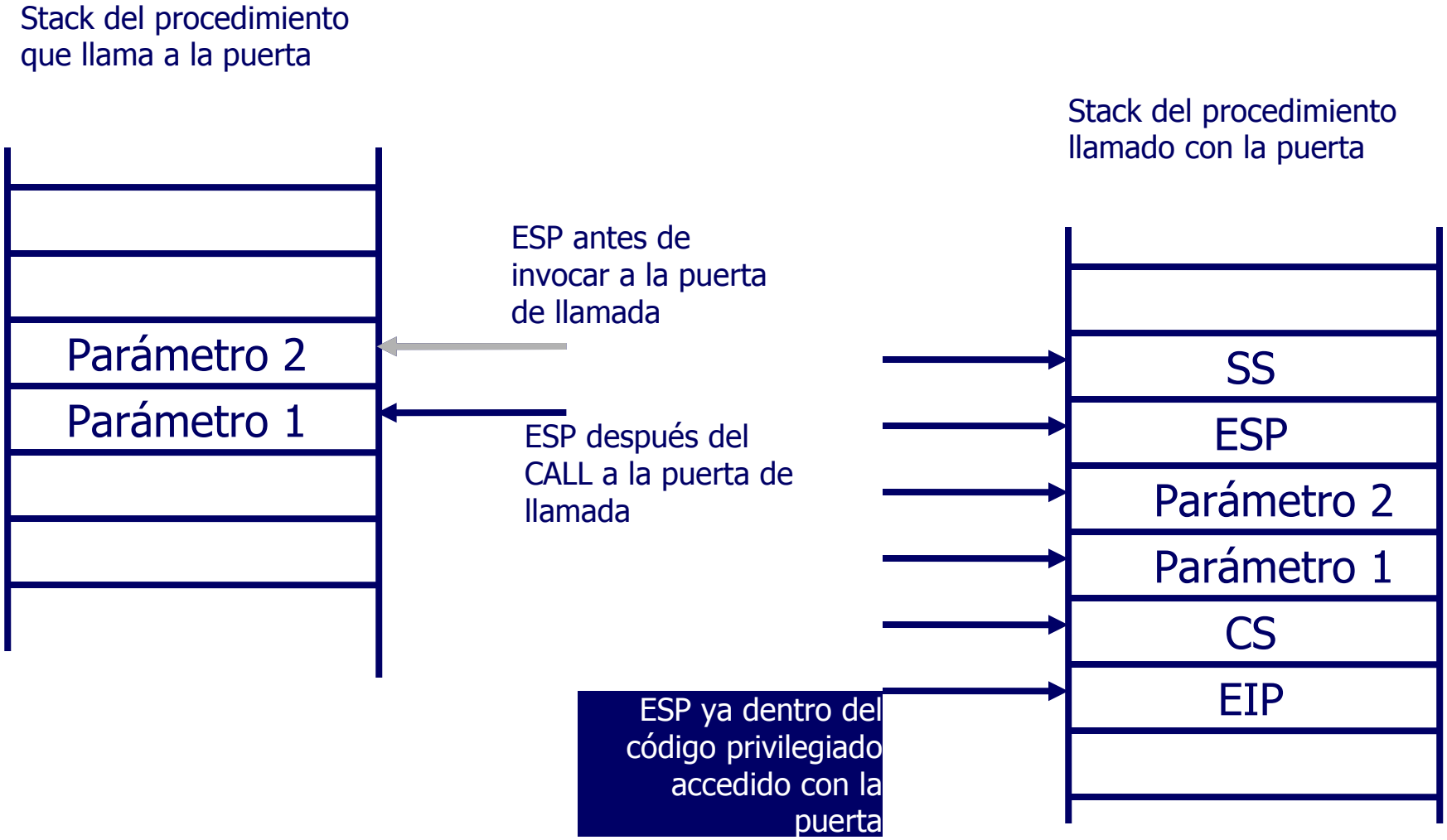
Código para "fabricar" un acceso a Puerta de Llamada

Requerido en la instrucción, aunque el procesador no lo usa



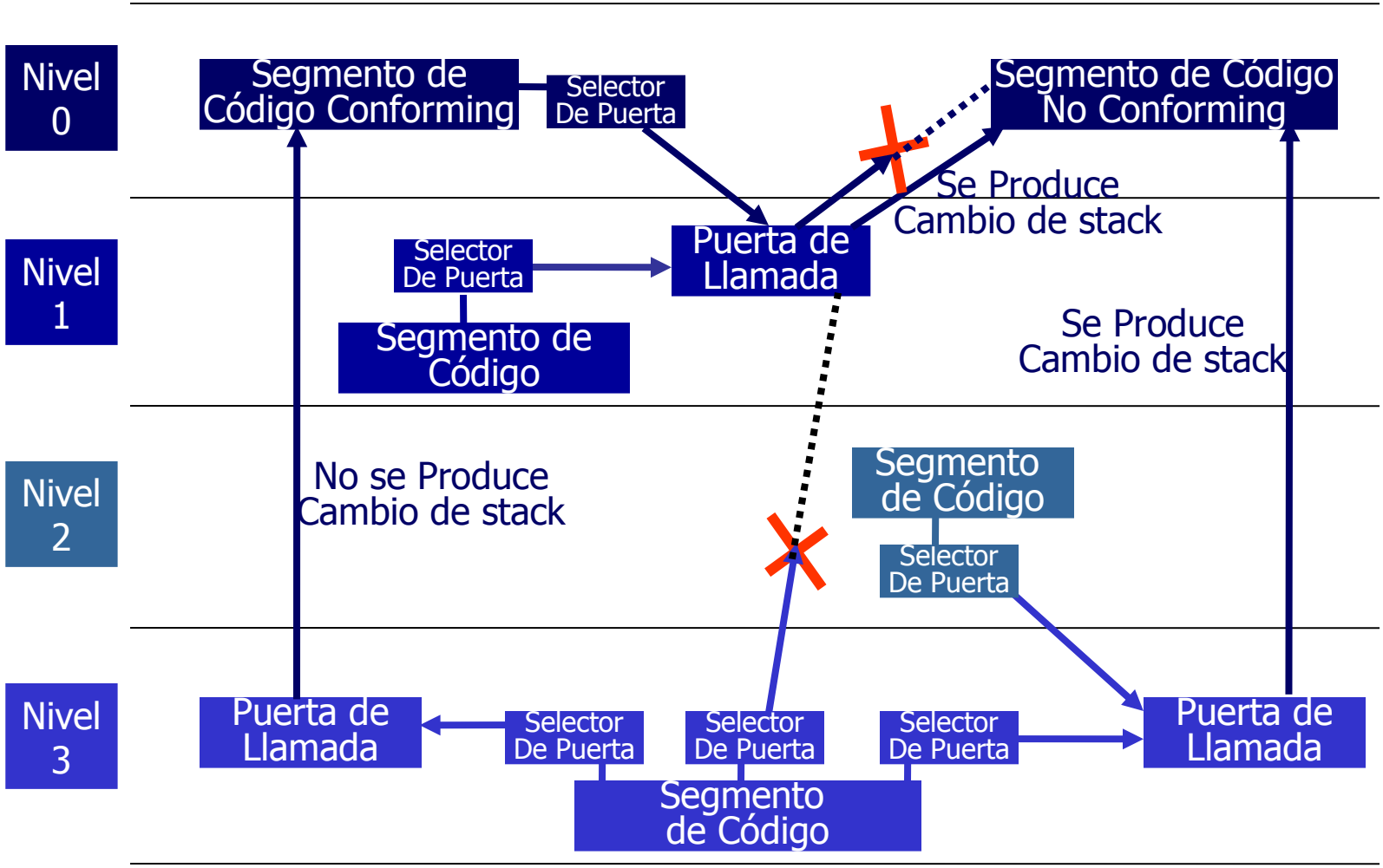
Manejo de la Pila con cambio de nivel de privilegio

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5



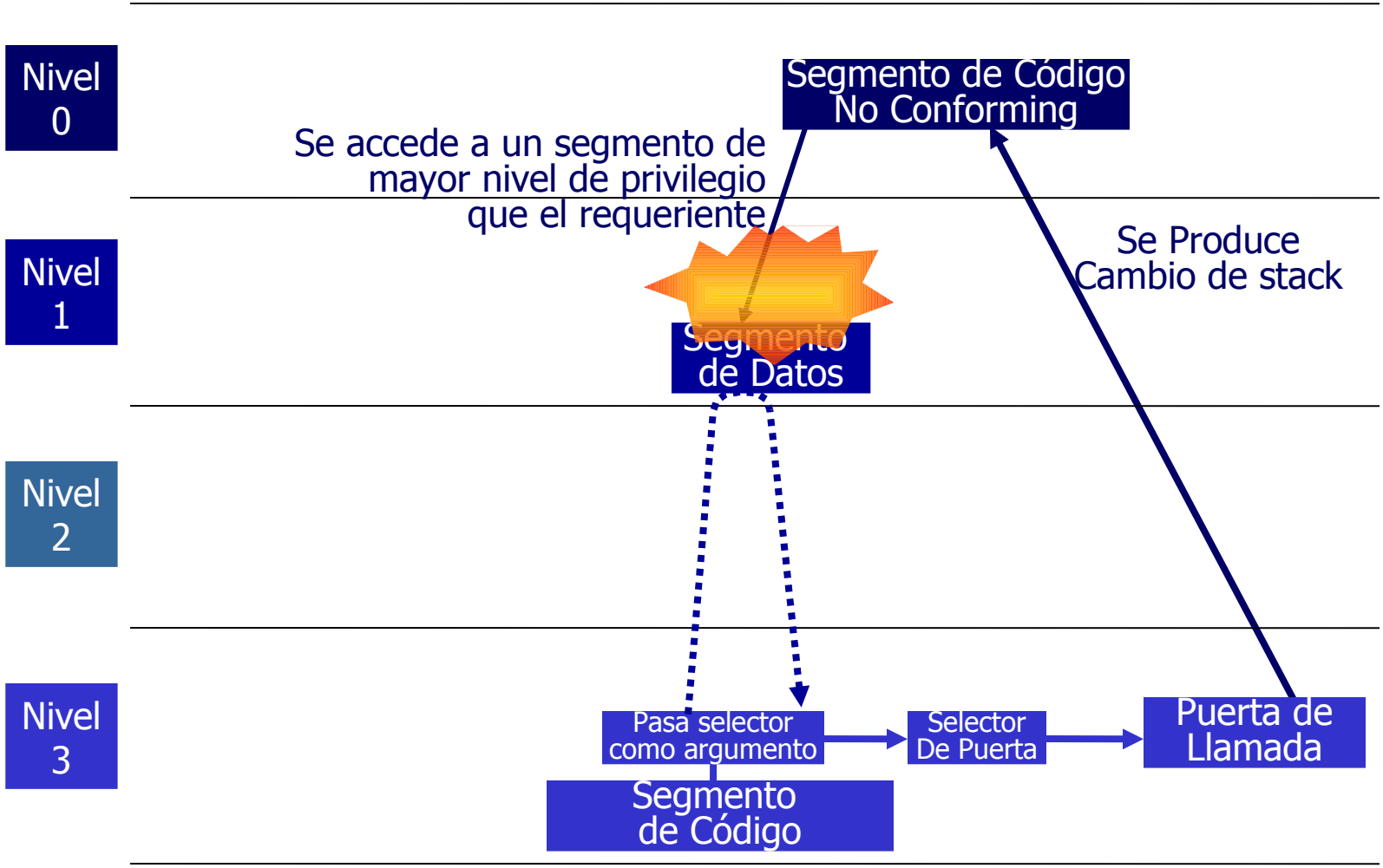
Puertas de Llamada: Resumen

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5



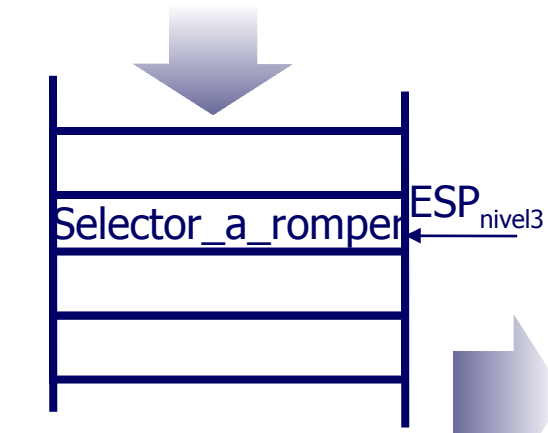
Escenario del Caballo de Troya

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5



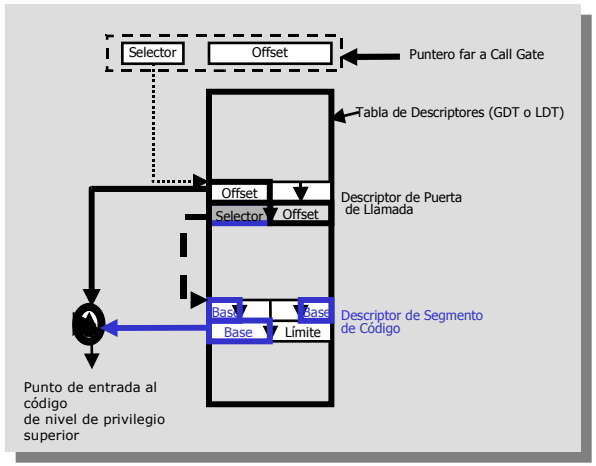
Escenario del Caballo de Troya

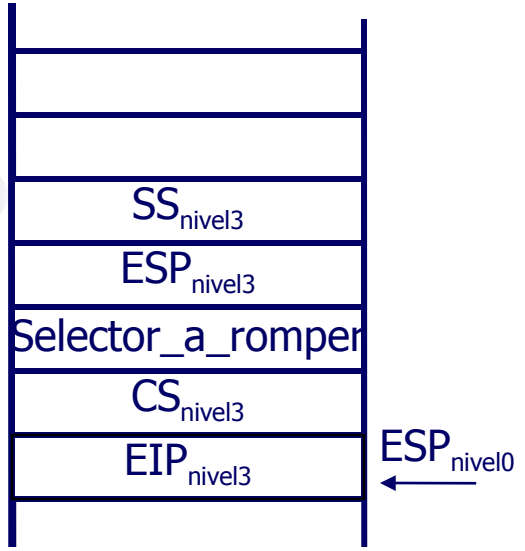

```
;Este programa ejecuta en un segmento de código con DPL = 11

mov     ax,219h     ;ax=0000 0010 0001 1001 ← RPL

;el programador intenta acceder al segmento de Nivel de Privilegio 1

push    ax
db  9Ah             ;Opcode Call Far
dd  ?           ;offset se ignora
dw  PLL_Selector    ;selector de PLL
```



Pila de Nivel 3
(procedimiento que llama)

Resolución del punto de entrada a Nivel 0





Pila de Nivel 0
(procedimiento llamado)

Escenario del Caballo de Troya

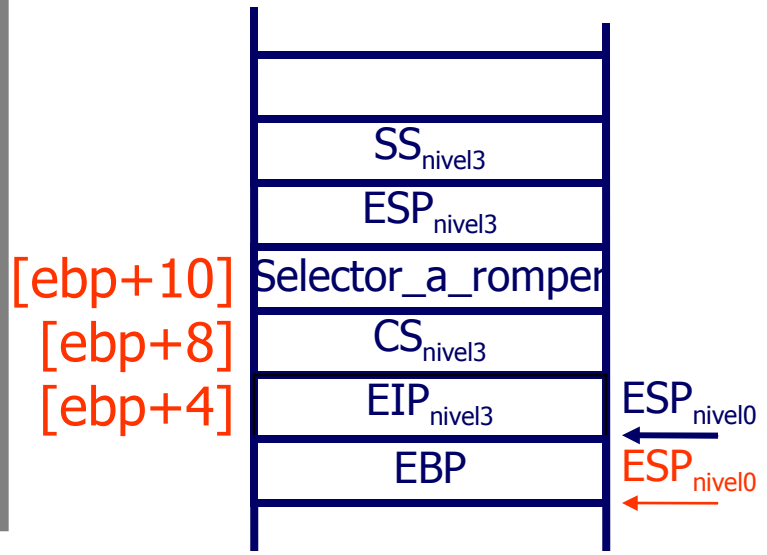

```
;Punto de entrada del programa de Nivel 0
(Offset ;que debe figurar en el descriptor de ;puerta
de ;llamada)

push ebp
mov ebp,esp

;extrae selector de CS del requeriente.
mov dx, word ptr [ebp+8]

;ajusta el nivel de privilegio del selector pasado
;como parámetro  (con RPL=01), con el del CS
de ;nivel 3, que es el privilegio del requeriente.
arpl  word ptr [ebp+10], dx

;lee selector ajustado (ahora RPL = 11)
mov ds, word ptr [ebp+10]

;En cuanto realice el acceso al segmento, excepción
;0Dh. El acceso no es posible.
;Solo debe manejarse el handler de la
Interrupción ;adecuadamente y regresar al
programa origen en ;el nivel 3, con un código de
error.
```

Se resuelve con al instrucción **ARPL**. Al igual que el bit conforming resuelve al acceso trabajando con el Nivel de Privilegio Efectivo

EPL = MAX (CPL,RPL)

ARPL r/m16, r16

Resultado:El operando destino adquiere el RPL del operando fuente, si es mayor numéricamente (menos privilegiado), caso contrario conserva el que tenía.



Restricción de Instrucciones

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 5

Existen instrucciones que, una vez en modo protegido, solo pueden ejecutarse en nivel de privilegio 0. Estas son:

- **LGDT** - Cargar registro GDTR
- **LLDT** - Cargar registro LDTR
- **LTR** - Cargar registro TR
- **LIDT** - Cargar Registro IDTR
- **MOV** - si destino es un Registro de Control
- **MOV** - si destino es un Registro de Debug
- **LMSW** - Escribir en el Machine Status Word (parte baja de CR0)
- **CLTS** - Clear Flag Task-Switched en CR0
- **INVD** - Invalidar Caché sin Write Back
- **WBINVD** - Invalidar Caché con Write Back
- **INVLPG** - Invalidar entrada de la TLB
- **HLT** - Parar el procesador..... =)
- **RDMSR** - Leer Model Specific Register
- **WRMSR** - Escribir Model Specific Register
- **RDPMC** - Leer Contador de Monitoreo de Performance
- **RDTSC** - Leer Time Stamp Counter

El Registro EFLAGS contiene el campo IOPL (bits 12 y 13) que determina el nivel de privilegio que debe tener la tarea en curso para acceder a la E/S. Por lo tanto, pueden ejecutarse IN, OUT, INS, y OUTS, desde esta tarea solo si CPL = IOPL

Protección a nivel de Páginas

- Se combina con la Protección a nivel de segmentos, aportando granularidad al sistema de protección dentro de un mismo segmento.
- Al igual que en los segmentos los chequeos se realizan dentro de la ventana de decodificación y ejecución de la instrucción.
- De acuerdo al sistema de protección de páginas (reflejado en la estructura del descriptor de página) hay dos niveles: Usuario (bit U/S=0) y Supervisor (U/S=1), y el tipo de página puede ser de Lectura o Lectura/Escritura.
- Cualquier violación al sistema de protección de páginas genera una Excepción 0Eh definida por Intel como **Page Fault** (#PF), y rebautizada bajo el misterioso y tristemente familiar nombre de "Error grave 0Eh" en algunas implementaciones.... **;-)**

Protección a nivel de Páginas

- Los CPL's 0, 1, y 2 del sistema de protección de segmentos mapean en el nivel Supervisor del esquema de páginas. El CPL3, se corresponde con el nivel Usuario.
- En el modo supervisor se accede por default a todas las páginas y en modo usuario solo a las que tienen en su descriptor el bit U/S = 1
- A partir del 80486, el bit 16 del registro CR0, se utiliza bajo el nombre WP (Write Protect).
  - Cuando el procesador esta en Modo Supervisor, por default, accede a cualquier página con permiso de lectura escritura. (Se ignora la protección de escritura)
  - Si es 1 impide al procesador escribir una página Read Only de nivel usuario desde código que ejecuta en una página en Modo Supervisor .
- El procesador chequea la protección en el PD, y en cada PT.

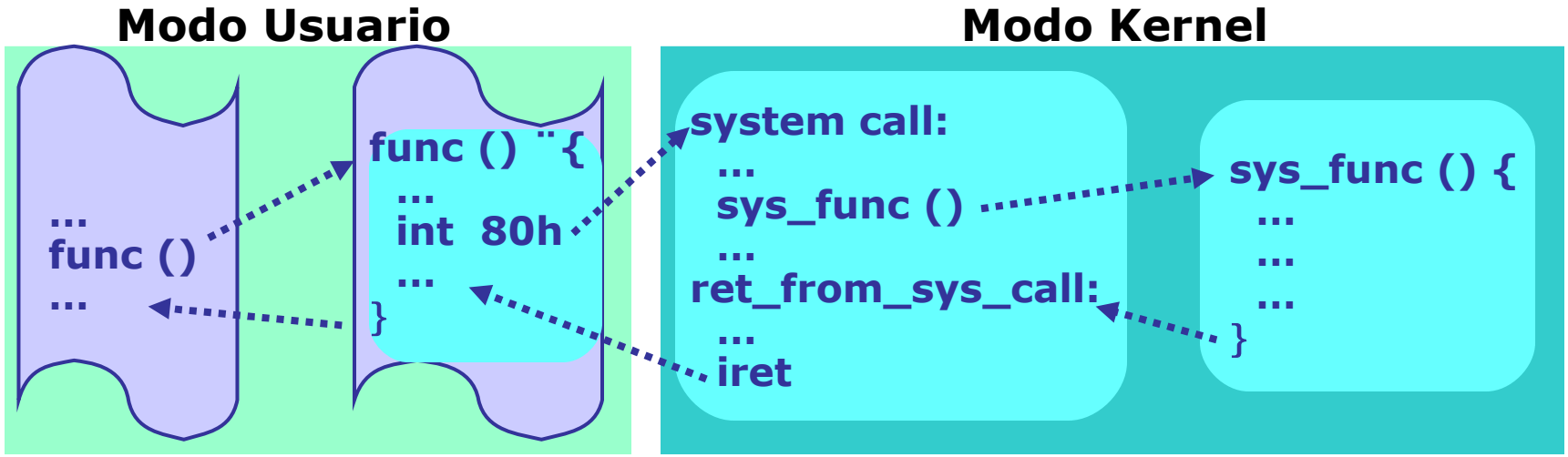
Combinación de Protección a nivel de Páginas y Segmentos

- El procesador evaluará siempre en primer lugar la protección de segmentos, ya que la Unidad de Paginación puede o no estar habilitada.
- Si genera una excepción pro segmentación no se genera la #PF.
- La protección a nivel de página no pisa a la protección a nivel de segmento. Paginar un segmento de código fijando permisos de escritura en las páginas, no permitirá escribir, ya que lo impedirá el mecanismo de protección de segmentos.
- En el caso de un segmento de datos con permiso de escritura, la paginación permite definir diferentes permisos para cada página con lo cual podremos dividirlo en áreas de lectura solamente y otras de lectura/escritura.

System Calls

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 10

- **API**: Formato de una llamada para obtener un servicio del kernel
- **System Call**: requerimiento explícito cursado al kernel para resolver un servicio.
- La inversa no es necesariamente cierta. Algunas API resuelven directamente en Modo Usuario si pasar a Modo kernel para resolver el pedido (math.lib por ejemplo).



Aplicación Invoca System Call

Accede a una rutina Wrapper en la librería standard ***libc***

handler de System Call del kernel

rutina de servicio de la System Call

System Calls de manejo de scheduler

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 7

nice( ) Cambia la prioridad estática de un proceso convencional

getpriority( ) Obtiene el máximo valor de prioridad estática para un grupo o proceso convencional

setpriority( ) Ajusta la prioridad estática para un grupo o proceso convencional

sched_getscheduler( ) Obtiene la política de scheduling de un proceso

sched_setscheduler( ) Establece la política de scheduling y prioridad real time para un proceso

sched_getparam( ) Obtiene la prioridad real time de un proceso

sched_setparam( ) Establece la prioridad real time de un proceso

sched_yield( ) Libera al procesador voluntariamente sin bloquearlo

sched_get_ priority_min( ) Obtiene el mínimo valor de prioridad real-time para una política dada

sched_get_ priority_max( ) Obtiene el máximo valor de prioridad real-time para una política dada

sched_rr_get_interval( ) Obtiene el valor del time quantum para la política Round Robin

sched_setaffinity( ) Establece la máscara de afinidad con CPU affinity de un proceso

sched_getaffinity( ) Obtiene la máscara de afinidad con CPU affinity de un proceso

System Calls de temporización

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 6

time () Retorna la cantidad de segundos transcurridos desde la medianoche de comienzo del 1ero. de Enero de 1970 (UTC)

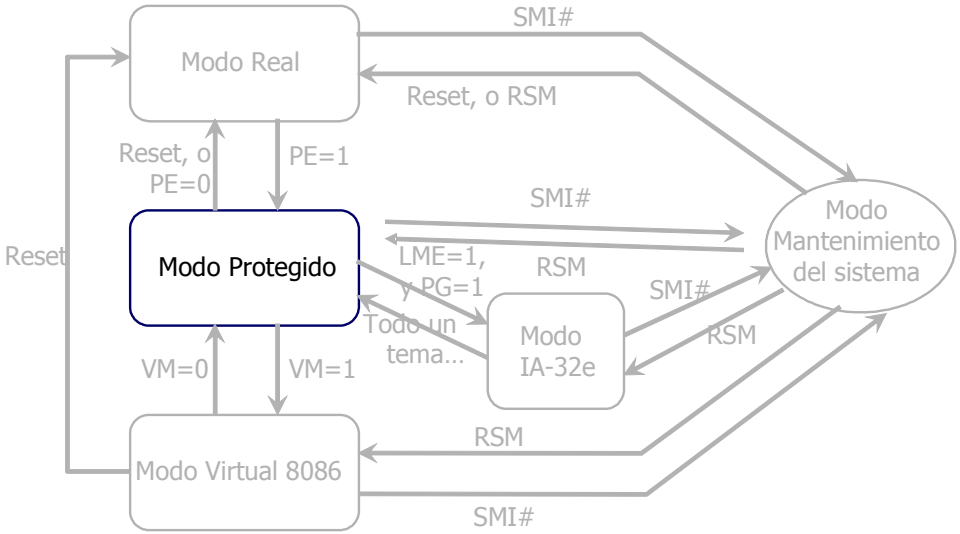
gettimeofday () Retorna un puntero a una estructura de tipo ***timeval***, que contiene la cantidad de segundos transcurridos desde la medianoche de comienzo del 1ero. de Enero de 1970 (UTC), y el número de micro segundos transcurridos del último segundo.

ctime () Convierte el tiempo numérico de calendario obtenido por time () a una cadena de la forma ASCII "Wed Jun 30 21:49:08 1993\n"

alarm () Genera una señal SIGALARM transcurrido el tiempo en segundos que se le pasa como parámetro.

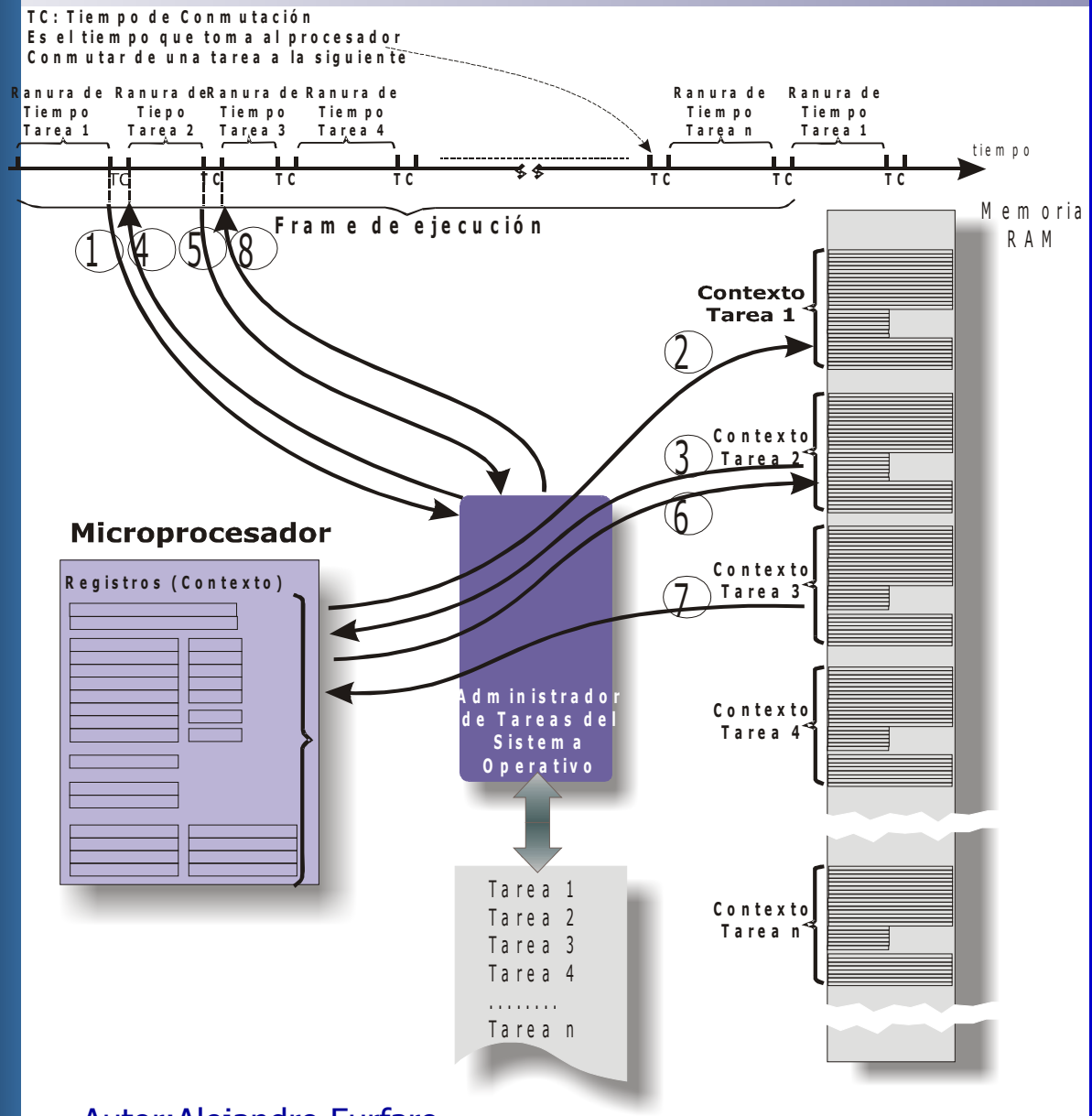
Modo Protegido

Manejo de Tareas



Conmutación entre tareas

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 7



1. Expira el tiempo de ejecución asignado por el procesador a la tarea 1
2. El procesador almacena en memoria el estado de máquina (contexto) de la tarea 1.
3. El procesador carga desde memoria el contexto de la tarea 2 tal como lo había almacenado originalmente.
4. El procesador reasume la ejecución de la tarea 2.
5. Expira el tiempo de ejecución asignado por el procesador a la tarea 2
6. El procesador almacena en memoria el estado de máquina (contexto) de la tarea 2.
7. El procesador carga desde memoria el contexto de la tarea 3 tal como originalmente lo había almacenado.
8. El procesador reasume la ejecución de la tarea 3.

Autor:Alejandro Furfaro

Manejo de Tareas

- **Tarea**: es una unidad de trabajo que un procesador puede despachar, ejecutar, y detener a voluntad, bajo la forma de:
  - La instancia de un programa (o, expresado en términos del Sistema Operativo, ***proceso***).
  - Un handler de interrupción.
  - Un servicio del kernel (Núcleo del Sistema Operativo).
- **Espacio de ejecución**: Es el conjunto de segmentos de código, datos, y pila que componen la tarea. En un sistema operativo que utilice los mecanismos de protección del procesador se requiere un segmento de pila por cada nivel de privilegio.
- **Contexto de ejecución**: Es el conjunto de valores de los registros internos del procesador en cada momento. Para poder suspender la ejecución de una tarea y poder reasumirla posteriormente, es necesario almacenar este contexto en el momento de la suspensión.
- **Espacio de Contexto de ejecución**: Se compone de un segmento de memoria en el que el kernel del S.O. almacenará el contexto completo de ejecución del procesador. Se lo denomina Task State Segment (TSS)

Concepto de Máquina Virtual

- El Sistema Operativo pondrá a disposición de cada tarea una Máquina Virtual. Se entiende por tal al subconjunto del universo de recursos del sistema que serán asignados a cada tarea:
  - CPU (% o rodaja de tiempo de ejecución)
  - Memoria (un conjunto de segmentos / páginas del total, de memoria instalada en el sistema)
  - Acceso a dispositivos de E/S (a través de módulos de software escritos por el fabricante llamados device drivers)
- No se incluyen en la máquina virtual los recursos hardware dependientes, ni algunas instrucciones que en un entorno de ejecución Multitasking son para uso exclusivo del código del Sistema Operativo.
- En cada momento la tarea activa posee entonces a su disposición una Máquina Virtual, que salvo los recursos para los que la tarea no debe tener acceso, es una imagen de la Máquina Base.

Estructuras asociadas al manejo de Tareas

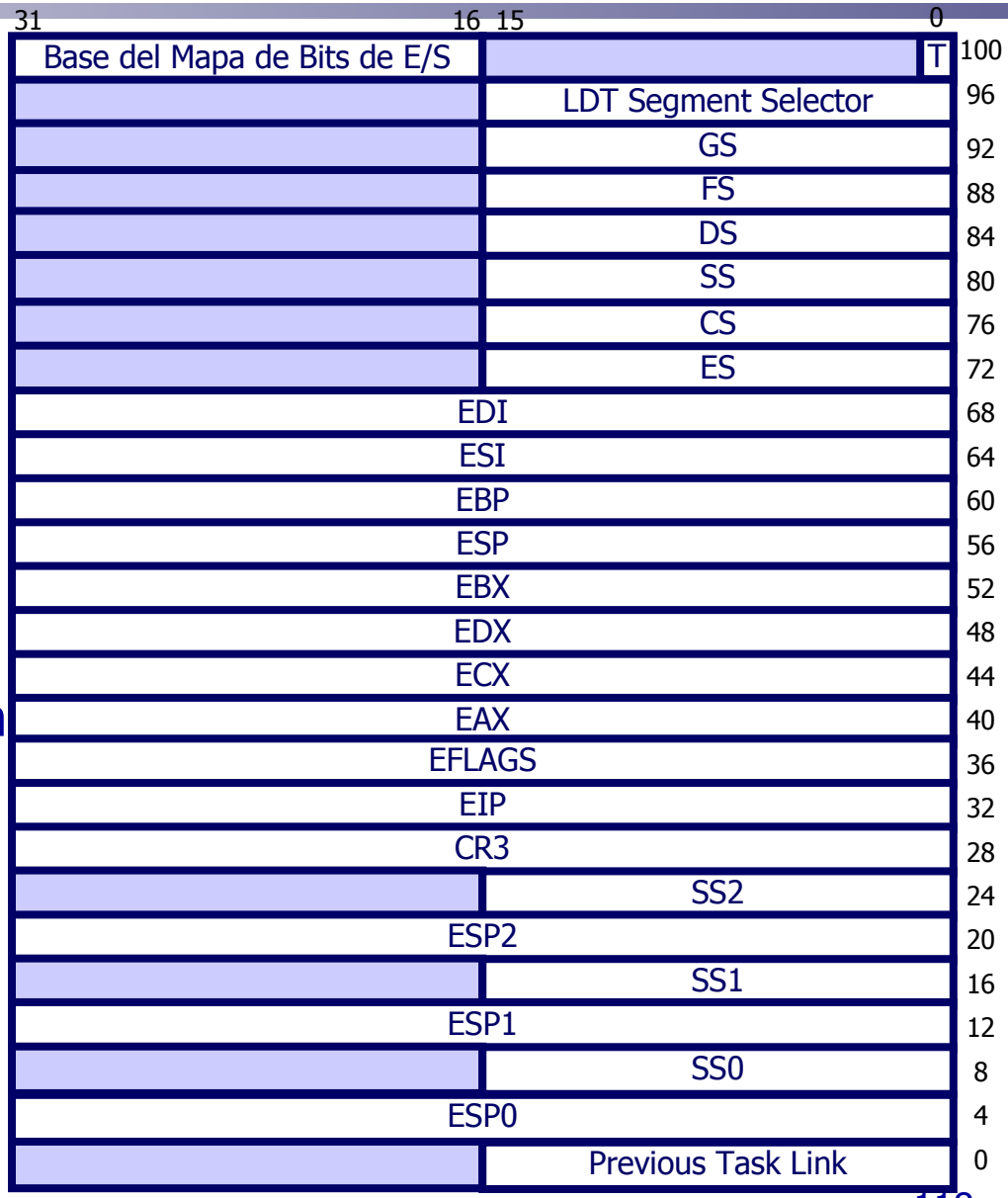
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 7

- Segmento de estado de tarea (TSS).
- Descriptor de TSS
- Descriptor de Puerta de Tarea
- Registro de tarea
- Flag NT (bit 14 de EFLAGS)

TSS (Segmento de Estado de Tarea)

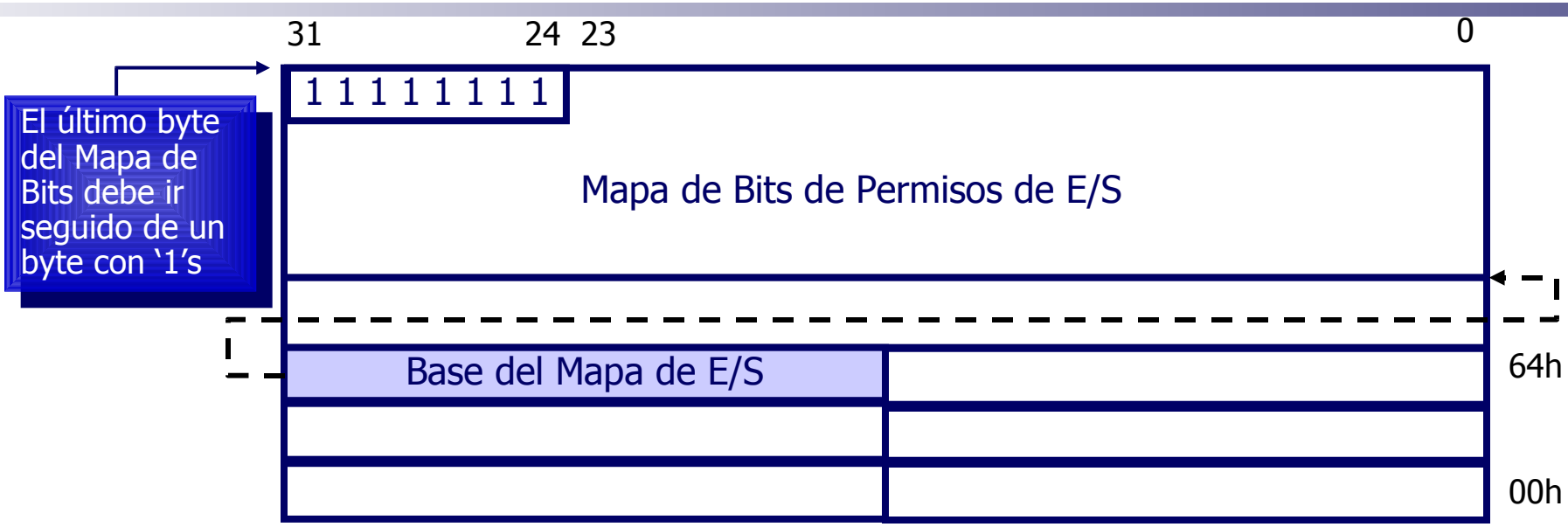
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 7

- Es el lugar de memoria en el que el S.O. Almacena el contexto de cada tarea.
- El tamaño **mínimo** de este segmento es 67h.
- Se guardan los valores de SS y ESP para los stacks de nivel 2, 1, y 0. El del nivel 3 eventualmente estará en los registros SS:ESP.
- EL Flag T genera una excepción de Debug cada vez que se conmuta de tarea (Pentium Pro en adelante), si está en '1'.
- I/O Map Base Address: Offset de 16 bits desde el inicio del TSS hasta el inicio del Mapa de permisos de E/S



TSS: Mapa de Permisos de E/S

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 7



Por default en Modo Protegido, una tarea que ejecuta con CPL=11 no puede ejecutar instrucciones de acceso a E/S (IN, OUT, INS, OUTS).

El procesador utiliza el par de bits IOPL del registro EFLAGS, para modificar este comportamiento default, de manera selectiva para cada tarea.

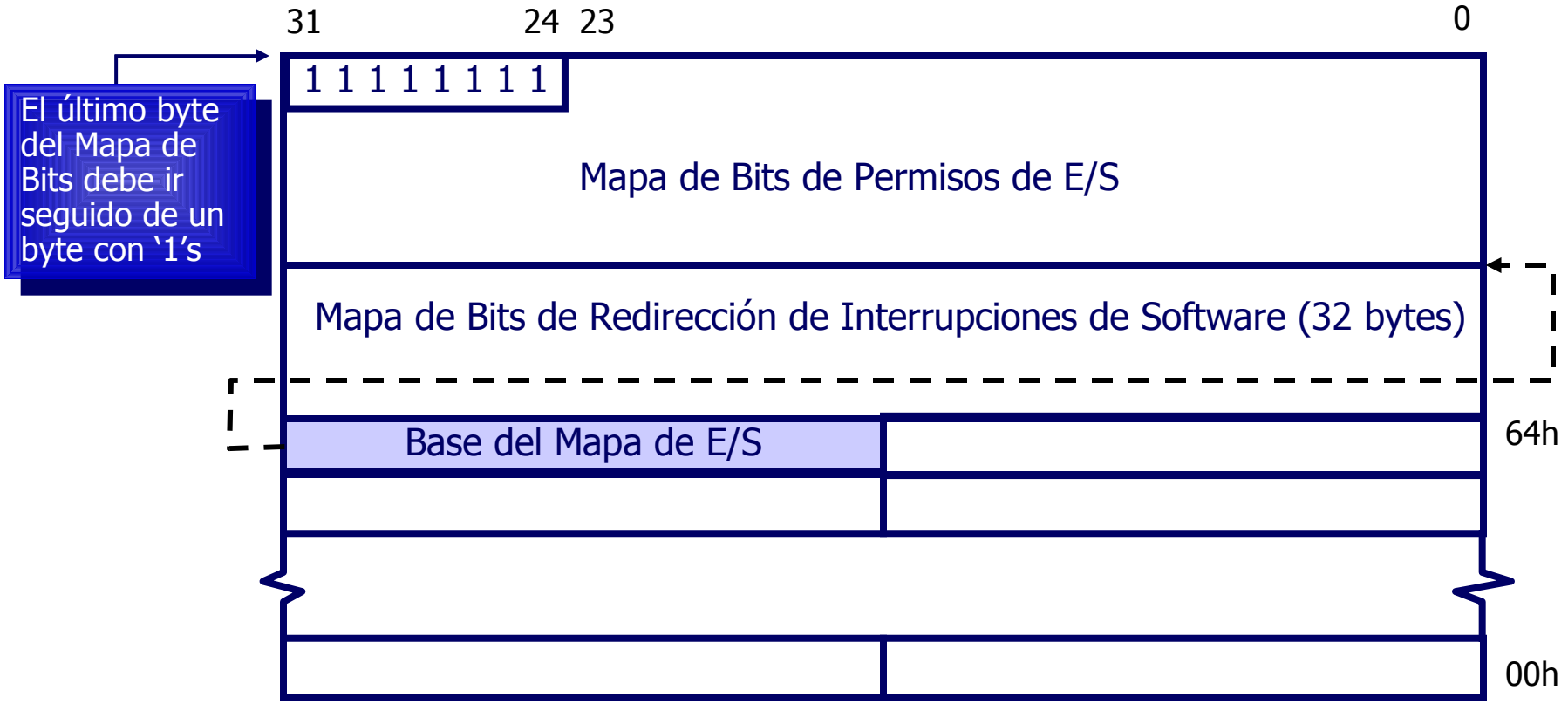
Para una determinada tarea con CPL=11, si desde el S.O. se pone el campo IOPL de Eflags en 11, se habilita el acceso a las direcciones de E/S cuyos bit correspondientes estén seteados en el Bit Map de permisos de E/S.

Así se habilita el acceso a determinados ports para determinadas tareas.

En este caso el TSS mide mas de 104 bytes (su límite será mayor que 67h)

TSS Mapa de Permisos de E/S

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 7



A partir del Pentium se incluye el Registro CR4, cuyo bit **VME** (Virtual 8086 Mode Extensions). Si **VME** = 0 en Modo Virtual 8086 el procesador responde a las interrupciones y excepciones como un 386 o un 486. Cuando **VME** = 1, salta el mecanismo y redirige las instrucciones a los handlers del entorno de ejecución del programa 8086, de acuerdo con el valor de estos bits.

Redirección de Interrupciones

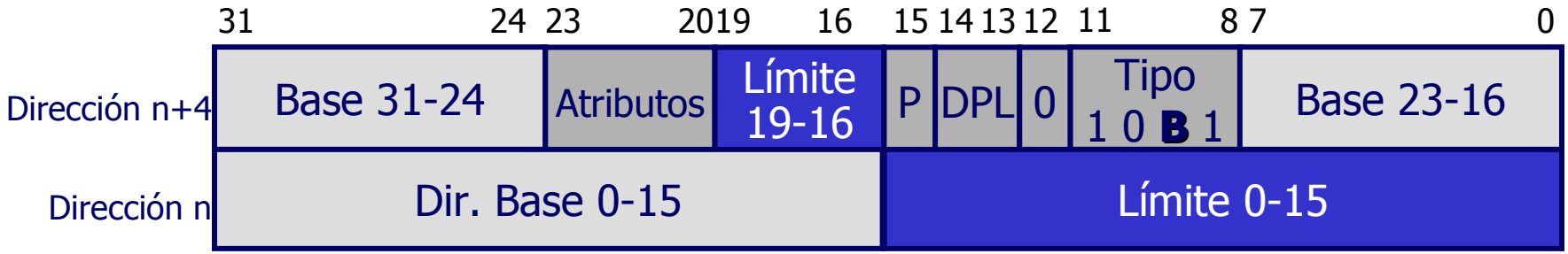
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 7

Método	VME	IOPL	Bit en Bitmap de Redir.*	Acción del Procesador
1	0	3	X	La Interruptción se dirige a un handler de interrupción de Modo Protegido: * Limpia los flags VM y TF * Si el servicio es a través de una puerta de interrupción, limpia el flag IF * Conmuta el stack al de nivel de privilegio 0 * Pushea GS, FS, DS y ES sobre el stack de Nivel de privilegio 0 * Pone GS, FS, DS y ES a 0 * Pushea SS, ESP, EFLAGS, CS y EIP de la tarea interrumpida sobre elstack de Nivel de privilegio 0 * Inicializa CS y EIP con los valores de la puerta de interrupción
2	0	< 3	X	La Interrupción se dirige al handler de la excepción de protección general (#GP) de Modo Protegido.
3	1	< 3	1	La Interrupción se dirige al handler de la excepción de protección general (#GP) de Modo Protegido; los flags VIF y VIP dan soporte para el manejo de clase 2 de las interrupcioness enmascarables de hardware .
4	1	3	1	La Interrupción se dirige al handler de interupción de Modo Protegido: (ver Acción del Procesador para método 1).
5	1	3	0	La Interrupción se redirige al handler de interrupción del programa 8086 : * Pushea EFLAGS con NT limpio e IOPL en 00 * Pushea CS y EIP (solo los 16 bits bajos) * Limpia el flag IF * Carga CS y EIP (solo los 16 bits bajos) desde la entrada seleccionada en la tabla de vectores de interrupción de la tarea actual virtual-8086
6	1	< 3	0	Interrupt redirected to 8086 program interrupt handler; VIF and VIP flag support for handling class 2 maskable hardware interrupts: * Pushea EFLAGS con IOPL en 3 y VIF copiado en IF * Pushea CS y EIP (solo los16 bits bajos) * Limpia el flag VIF * Limpia el flag TF * Carga CS y EIP (solo los16 bits bajos) desde la entrada seleccionada en la tabla de vectores de interrupción de la tarea actual virtual-8086

NOTA:
* Cuando está en 0, la interrupción de software se redirige al handler de interrupción del programa 8086; cuando está en 1, la interrupción se dirige al handler de Modo Protec

Descriptor de TSS

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 7

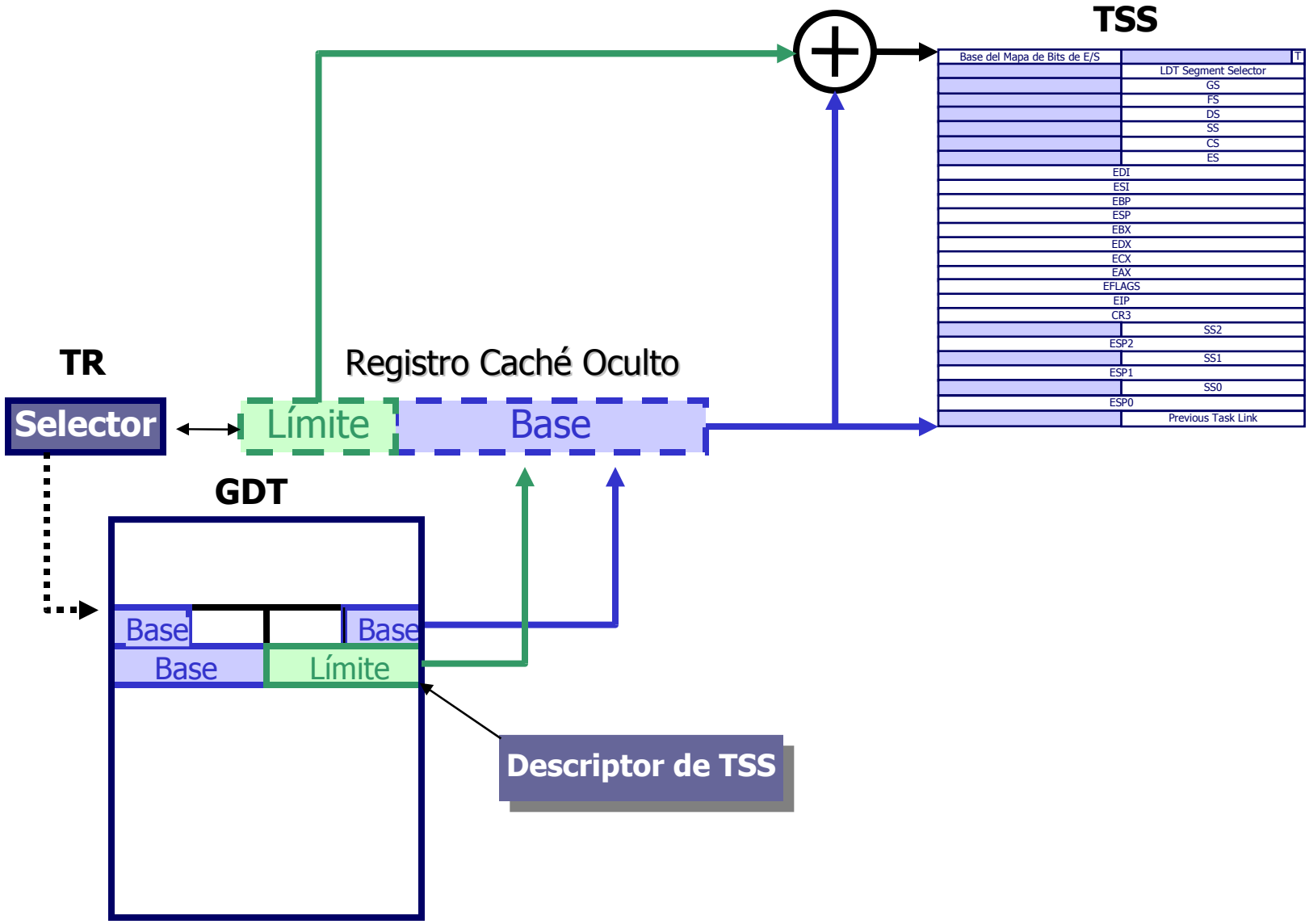


El Bit **B** (Busy) sirve para evitar recursividad en el anidamiento de tareas. Nos referiremos a él cuando analicemos el anidamiento de tareas.

El Límite debe ser mayor o igual a 67h (mínimo tamaño del segmento es 68h, o 103_{10}. De otro modo se genera una excepción tipo 0Ah o TSS inválido

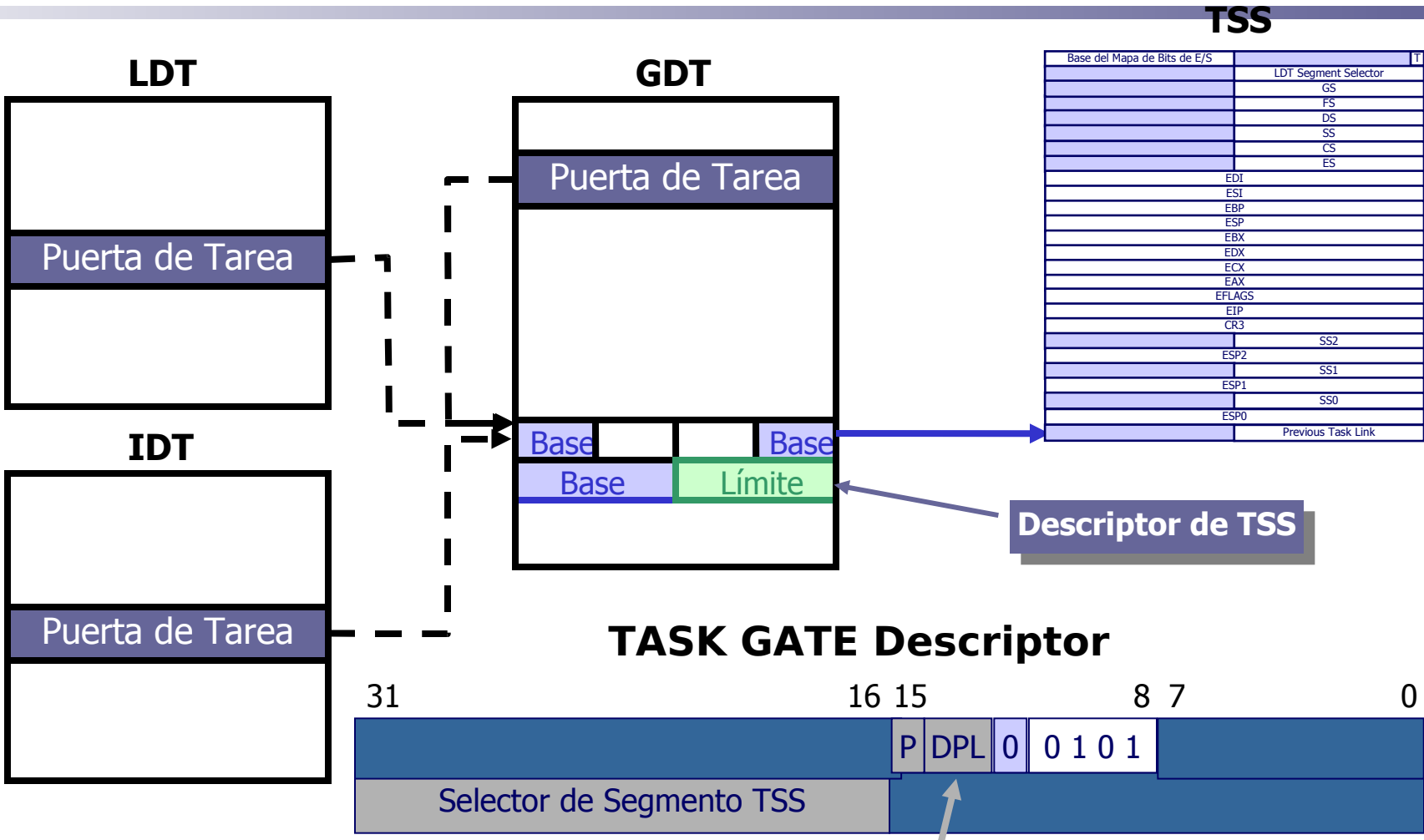
Acceso al TSS con el Task Register

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 7



Descriptor de Puerta de tarea

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 7



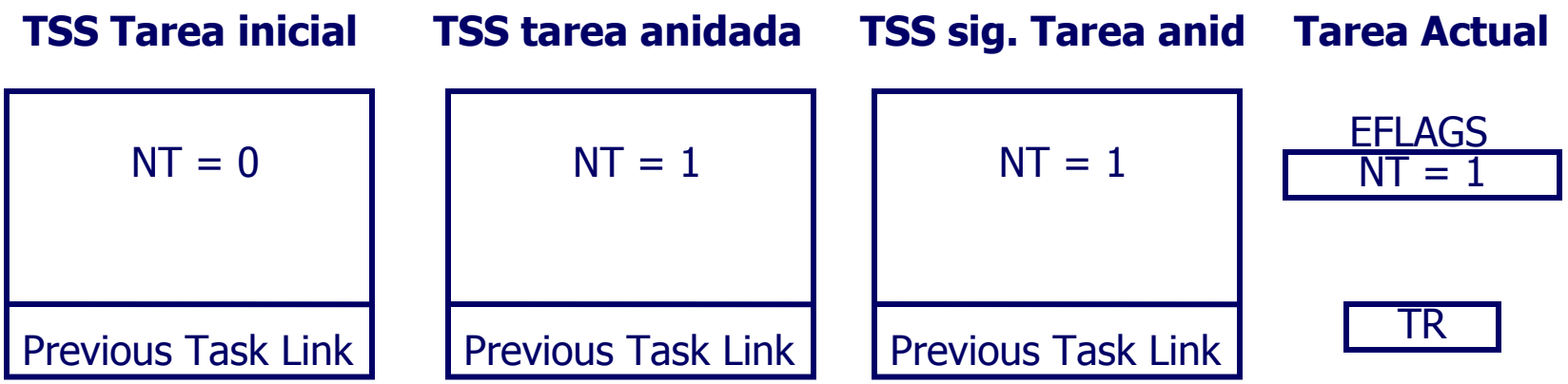
Si DPL=11 en un task gate, aún desde una tarea con CPL 11, puede efectuarse una conmutación de tareas. Este es otro mecanismo para acceder al kernel desde una aplicación.

Despacho de Tareas

- El procesador puede despachar una tarea de las siguientes formas posibles:
  - Por medio de una instrucción CALL
  - Por medio de una instrucción JMP
  - Mediante una llamada implícita del procesador al handler de una interrupción manejado por una tarea.
  - Mediante una llamada implícita del procesador al handler de una excepción manejado por una tarea.
  - Mediante la ejecución de la instrucción IRET en una tarea cuando el flag NT (bit 14 del registro EFLAGS) es "1" para la tarea actual.
- En cualquier caso se requiere poder identificar a la tarea.
- Se necesita un selector en la GDT que apunte a una puerta de tarea o a un Task State Segment (TSS). Este selector debe estar en la correspondiente posición dentro de la instrucción CALL o JMP.

Tareas Encadenadas

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 7



Cuando se conmuta a una tarea mediante un CALL, una interrupción, o una excepción, el procesador copia el TR de la tarea actual en el campo "Previous Task Link" del TSS de la nueva tarea, y luego de completar el cambio de contexto, setea el bit NT del registro EFLAGS (bit 14).

De este modo si la nueva tarea ejecuta en algún punto la instrucción IRET y el bit NT es '1', el procesador conmuta a la tarea anterior ya que tiene el selector de TSS de la tarea previa almacenado en el campo Previous Task Link del TSS de la tarea en ejecución.

En cambio si la conmutación de tarea se efectúe con un JMP no se afecta el flag NT ni se completa el campo "Previous Task Link.

Prevención de recursividad de tareas El Bit Busy

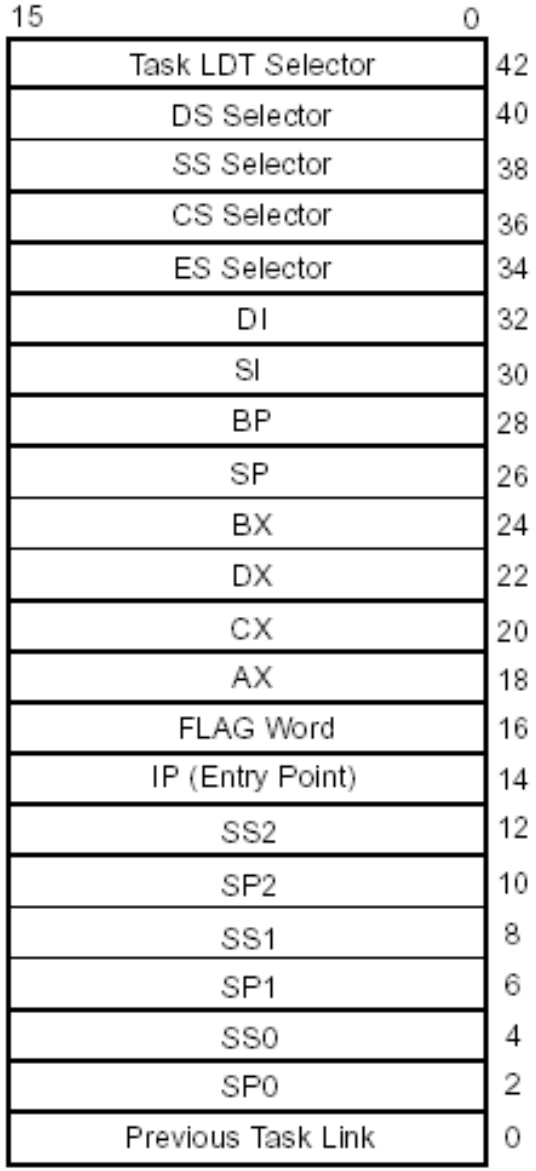
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 7

- El procesador utiliza el Bit Busy de un descriptor de TSS para prevenir la reentrancia en una tarea (esto ocasionaría la pérdida de los datos del contexto de ejecución).
- Cuando se avanza en anidamiento de tareas mediante un CALL o una interrupción, este bit debe permanecer seteado, en el descriptor de TSS de la tarea previa.
- El procesador generará una Excepción de Protección General (0Dh) si se intenta despachar mediante un CALL o una Interrupción una tarea en cuyo TSS está seteado el bit Busy. Esto no ocurre si la la tarea en cuestión se despacha con un IRET ya que es de esperar en esta condición que el bit Busy esté seteado.
- Cuando la tarea ejecuta IRET o bien mediante un JMP despacha una nueva tarea, el procesador asume que la tarea actual finalizará y se debe limpiar el bit Busy en su descriptor de TSS

TSS de 16 bits

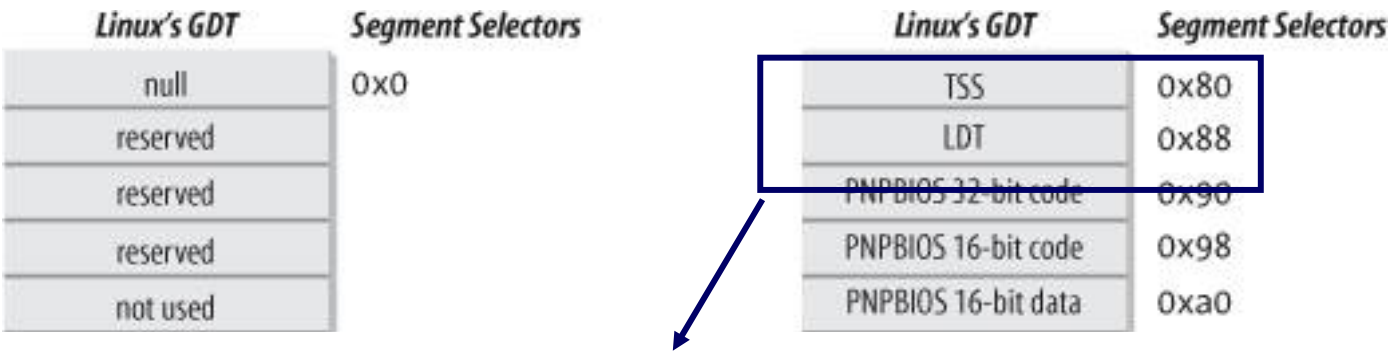
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, Capítulo 7

Por compatibilidad con el 80286 se manejan tareas de 16 bits. En eset caso el TSS tiene la siguiente estructura



Manejo de Tareas en Linux: TSS y LDT

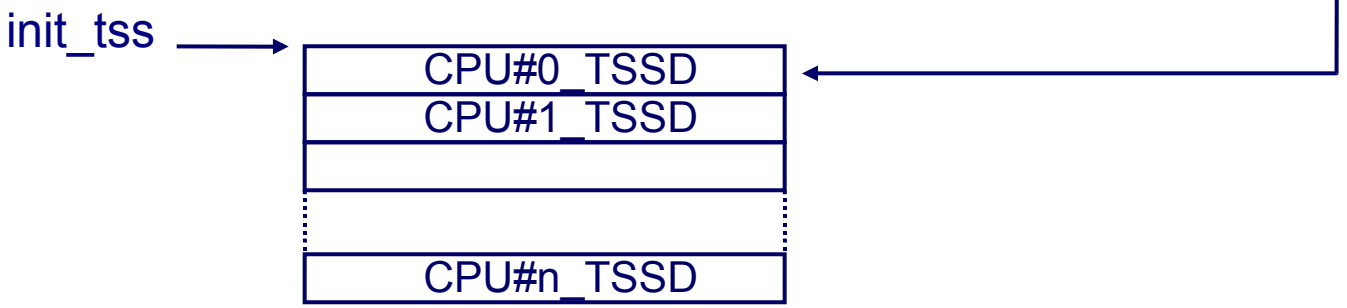
Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 2



- Una TSS por cada Procesador.
  - No hace Task Switch a través del procesador. (lo realiza mediante código)
  - Porque mantiene entonces una TSS?
    - El procesador busca aquí, el stack de PL=00 cuando sube el privilegio de una tarea (ya sea por llamada al kernel, o por entrada de una interrupción
    - Si un programa que corre en Modo User, ejecuta IN o OUT, se consulta el par de flags IOPL en EFLAGS y el IO Map del TSS.
- LDT Descriptor: uno genérico compartido por todos los procesos. No usa LDT.
  - Contiene un descriptor Null.
  - Dirección base: Se almacena en la ***default_ldt***.
  - Límite 7
  - Un proceso puede crear su propia LDT mediante la función del kernel ***modify_ldt( )***.

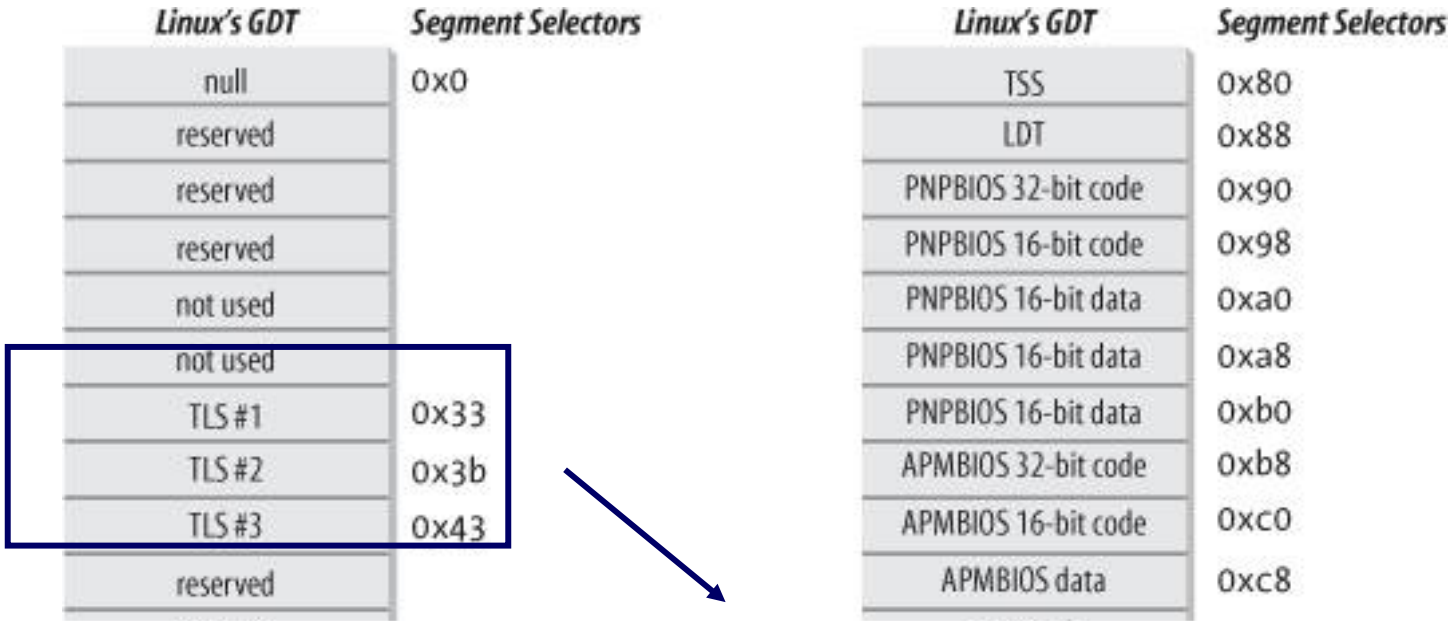
Manejo de Tareas en Linux: TSS

- Descriptores de Sistema
  - Task State Segment Descriptor (TSSD): Uno por cada procesador.
    - Limite =0xEB (236 bytes).
    - S = 1, DPL = 00, Tipo = 9 u 11 (TSS de 32 bits Available / Busy)
    - Dirección Base
    - Se apilan los descriptores de TSS de cada CPU en un array llamado ***init_tss***



Procesos en Linux usando la Tabla GDT

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 2



- Segmentos Thread-Local Storage (TLS) : Mecanismo que permite a las aplicaciones que hagan multithreading usar hasta tres segmentos conteniendo datos locales para cada thread. Las system calls ***set_thread_area( )*** y ***get_thread_area( )***, crean y liberan respectivamente un segmento TLS para el proceso en ejecución.

Modo Protegido

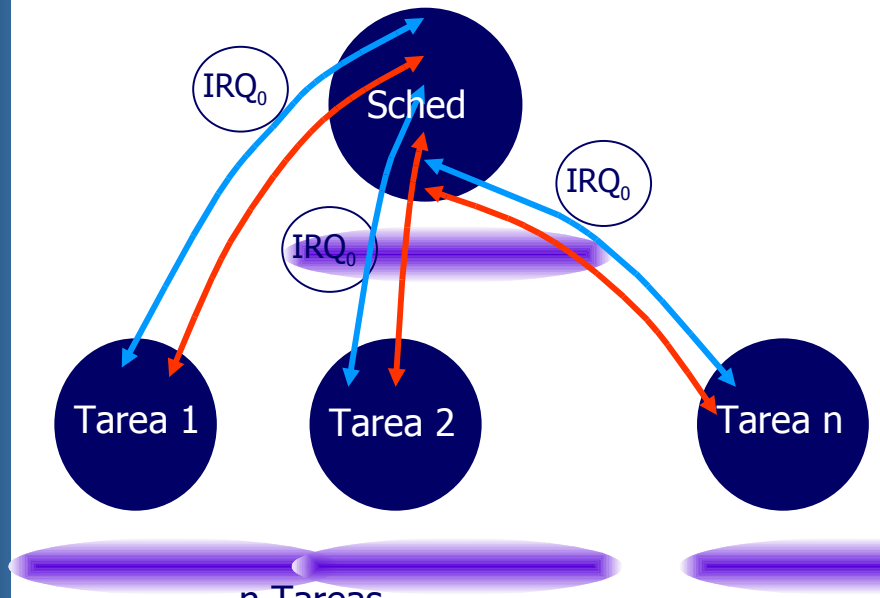
Diseño de un scheduler

Construcción de un Scheduler

- Base de tiempos: timer tick (IRQ_0)
- Existen dos alternativas
  - Colocar en la IDT una puerta de interrupción para IRQ0
  - Colocar en la IDT una puerta de tarea para IRQ0
- A continuación pros y cons de cada método

Scheduler. Contextos en juego

Puerta de Tarea en handler de IRQ_0

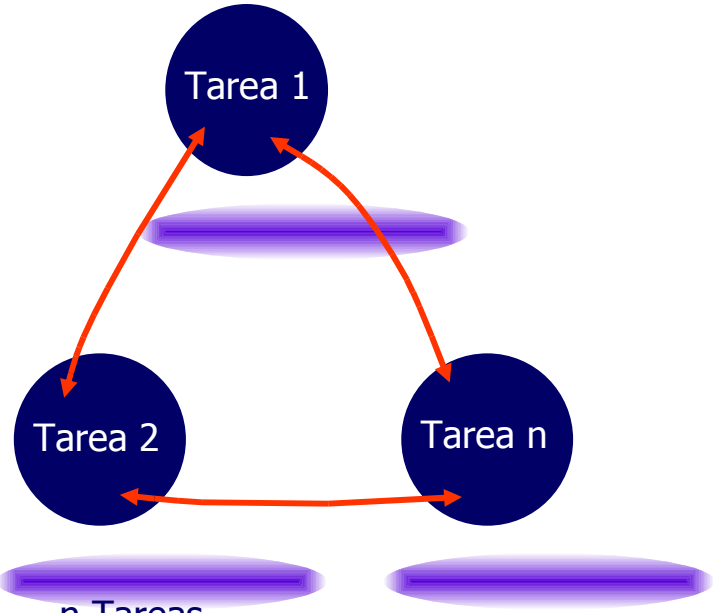


n Tareas
n+1 Contextos (n tareas + scheduler)

Conmutación a tarea scheduler
Event driven (IRQ_0).
La tarea queda interrumpida y su TSS con B=1

Conmutación desde el scheduler a la tarea

Puerta de Interrupción en handler de IRQ_0

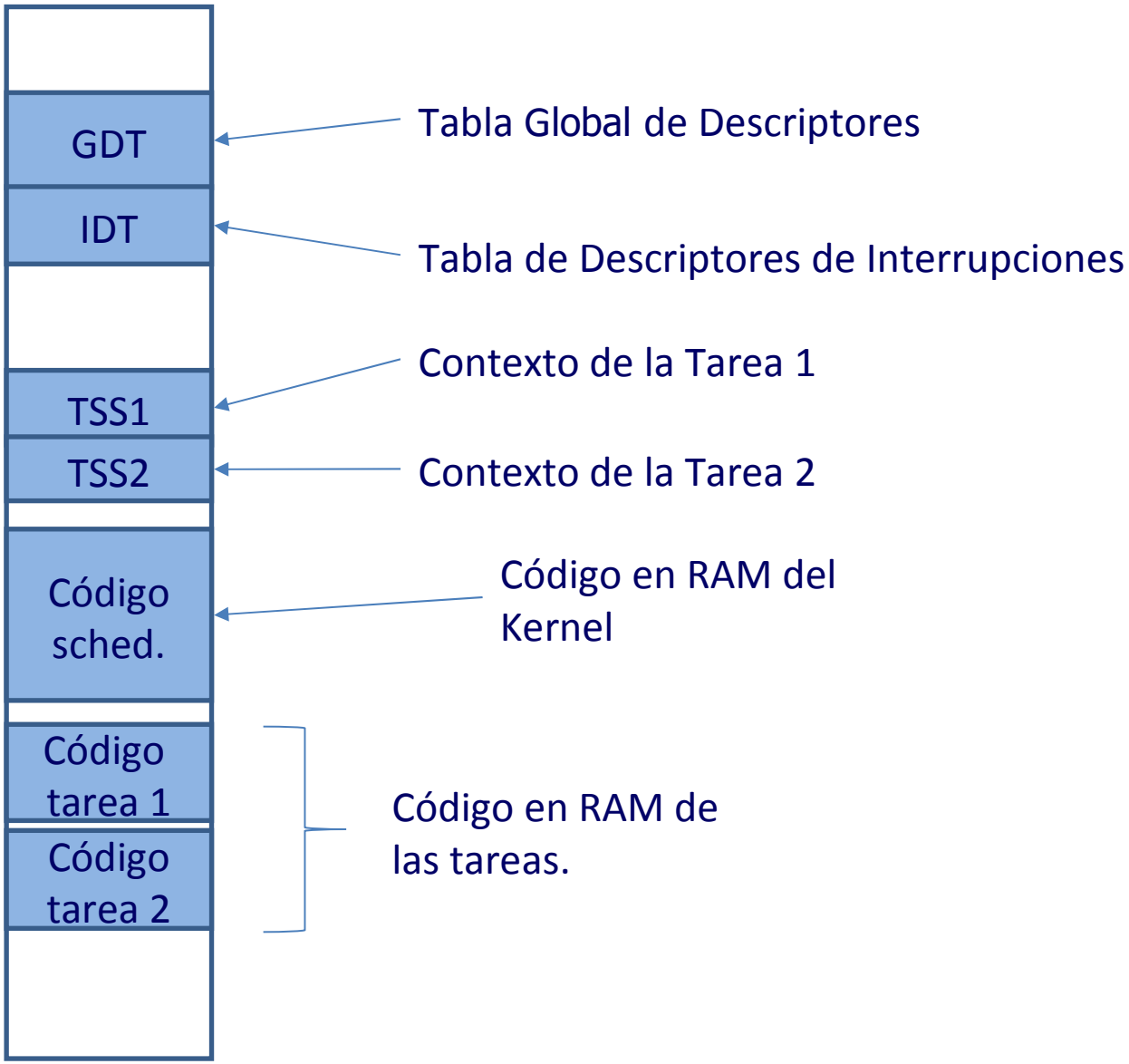


n Tareas
n Contextos (scheduler toma el contexto de la tarea que interrumpe)

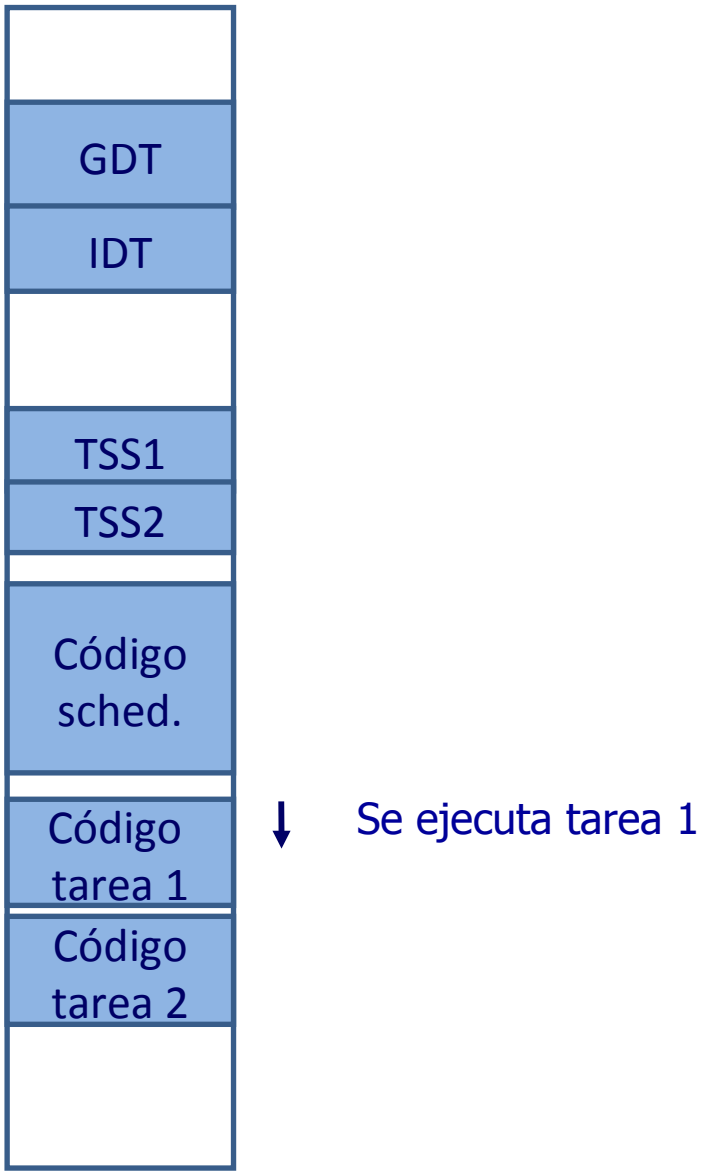
Conmutación desde la tarea r a la tarea r+1 desde el handler de la IRQ0, pero en el contexto de la tarea r.

IRQ_0 por puerta de Interrupción

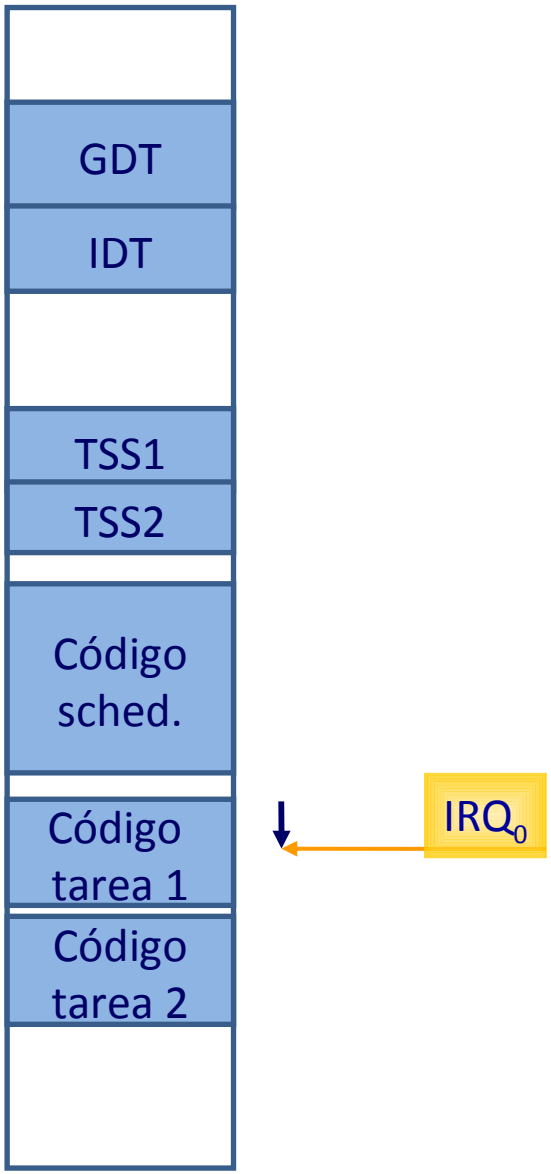
Puerta de Interrupción. Organización de la memoria (2 tareas)



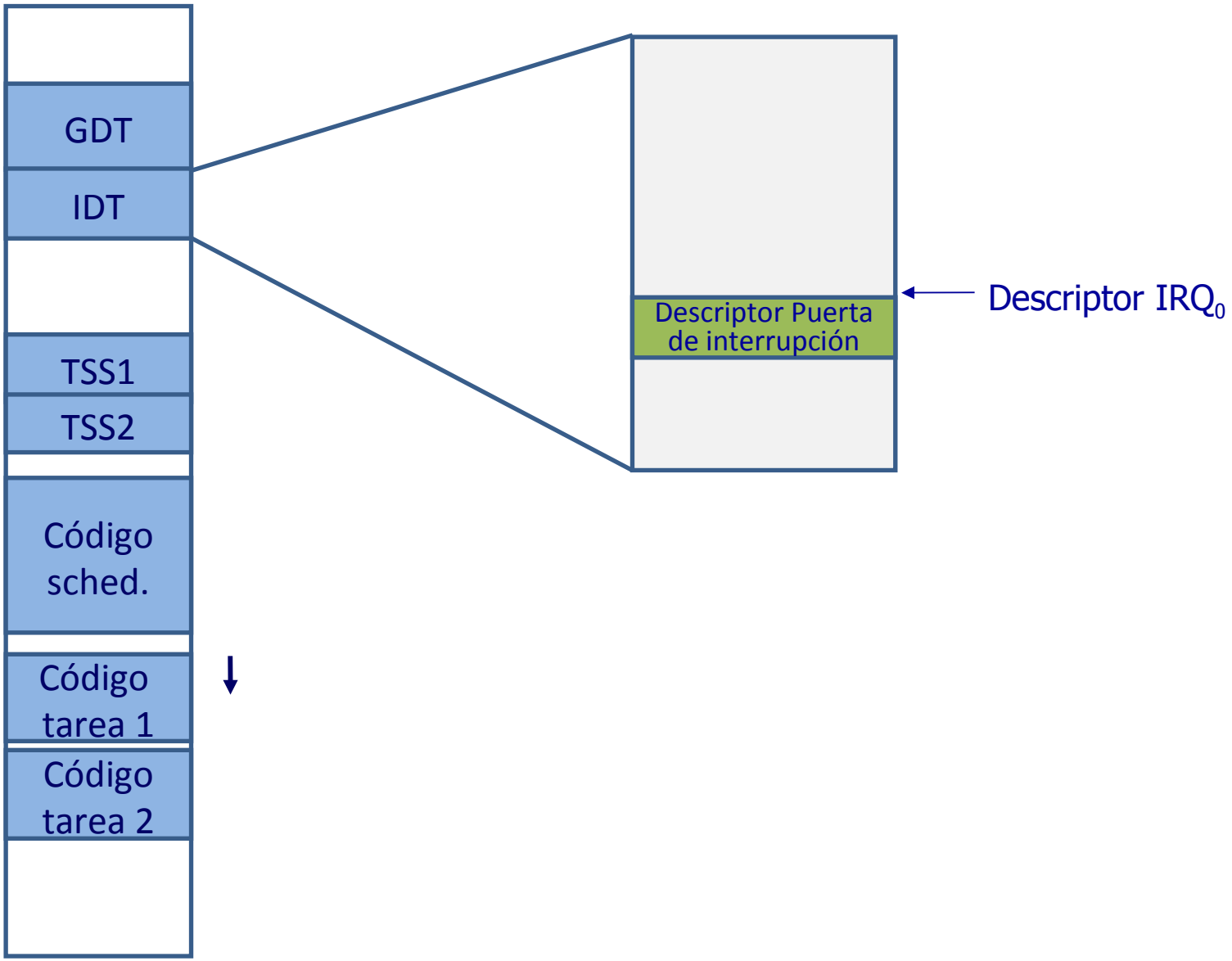
Puerta de Interrupción. Conmutación.



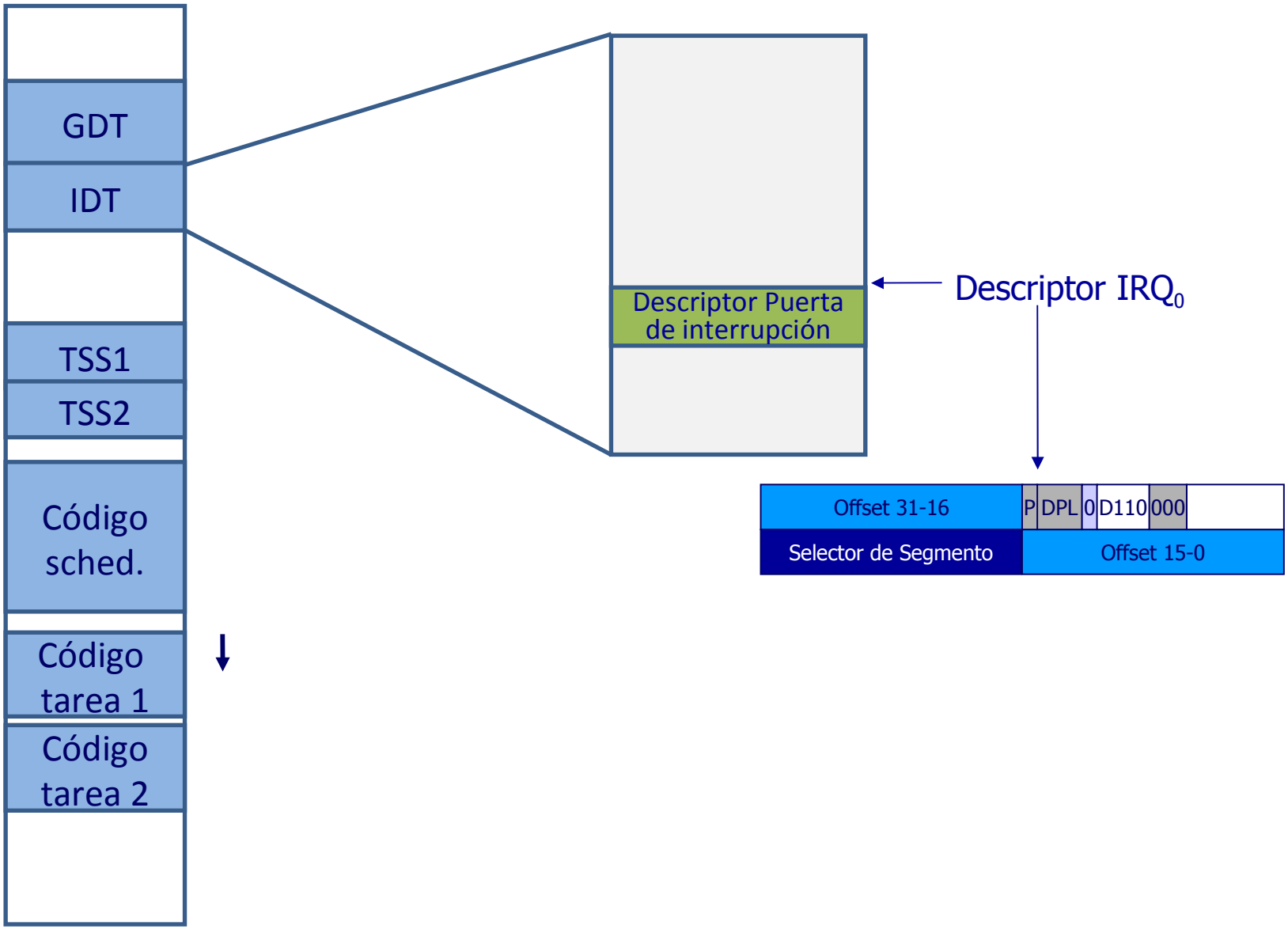
Puerta de Interrupción. Conmutación.



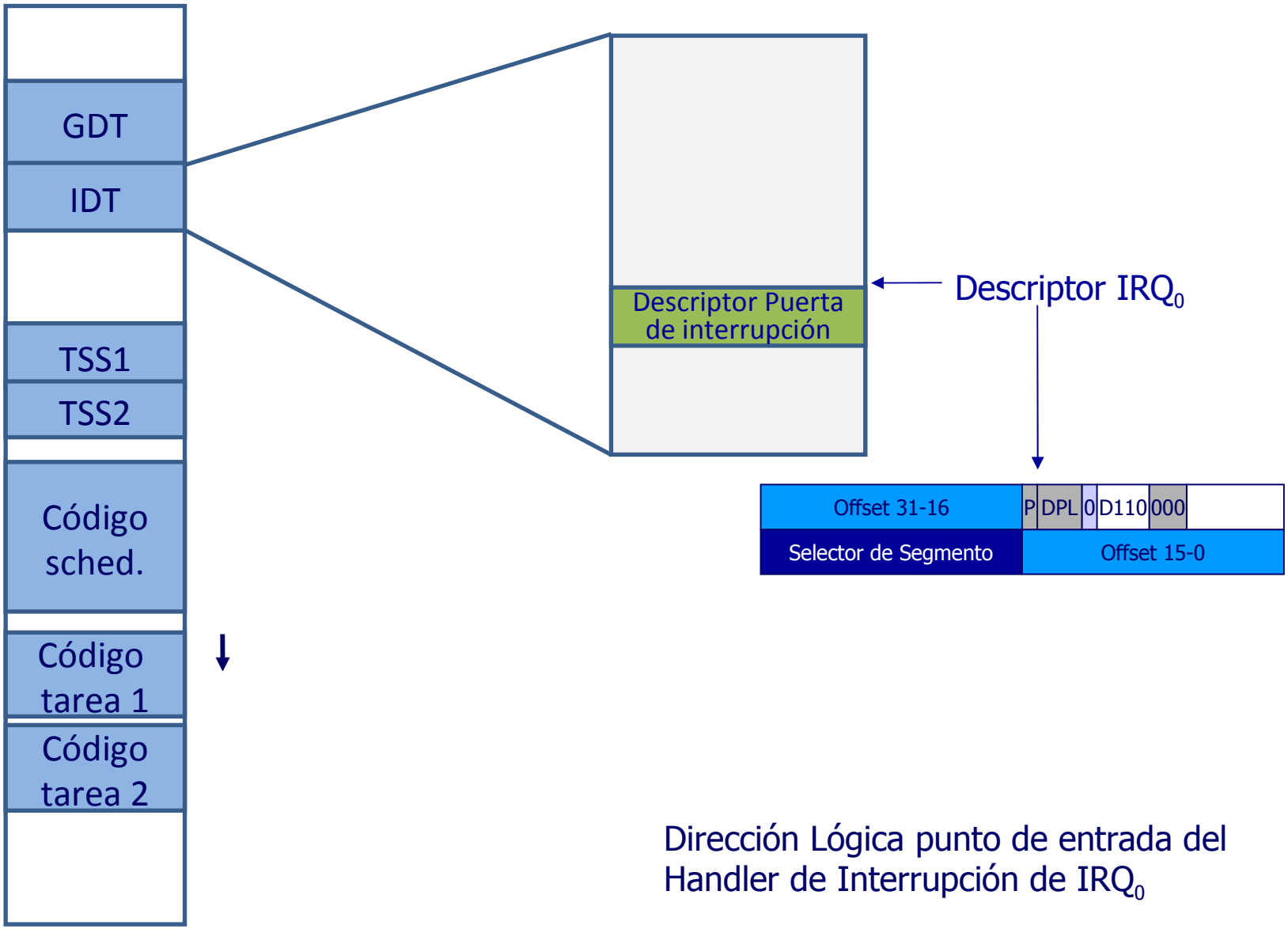
Puerta de Interrupción. Conmutación.



Puerta de Interrupción. Conmutación.

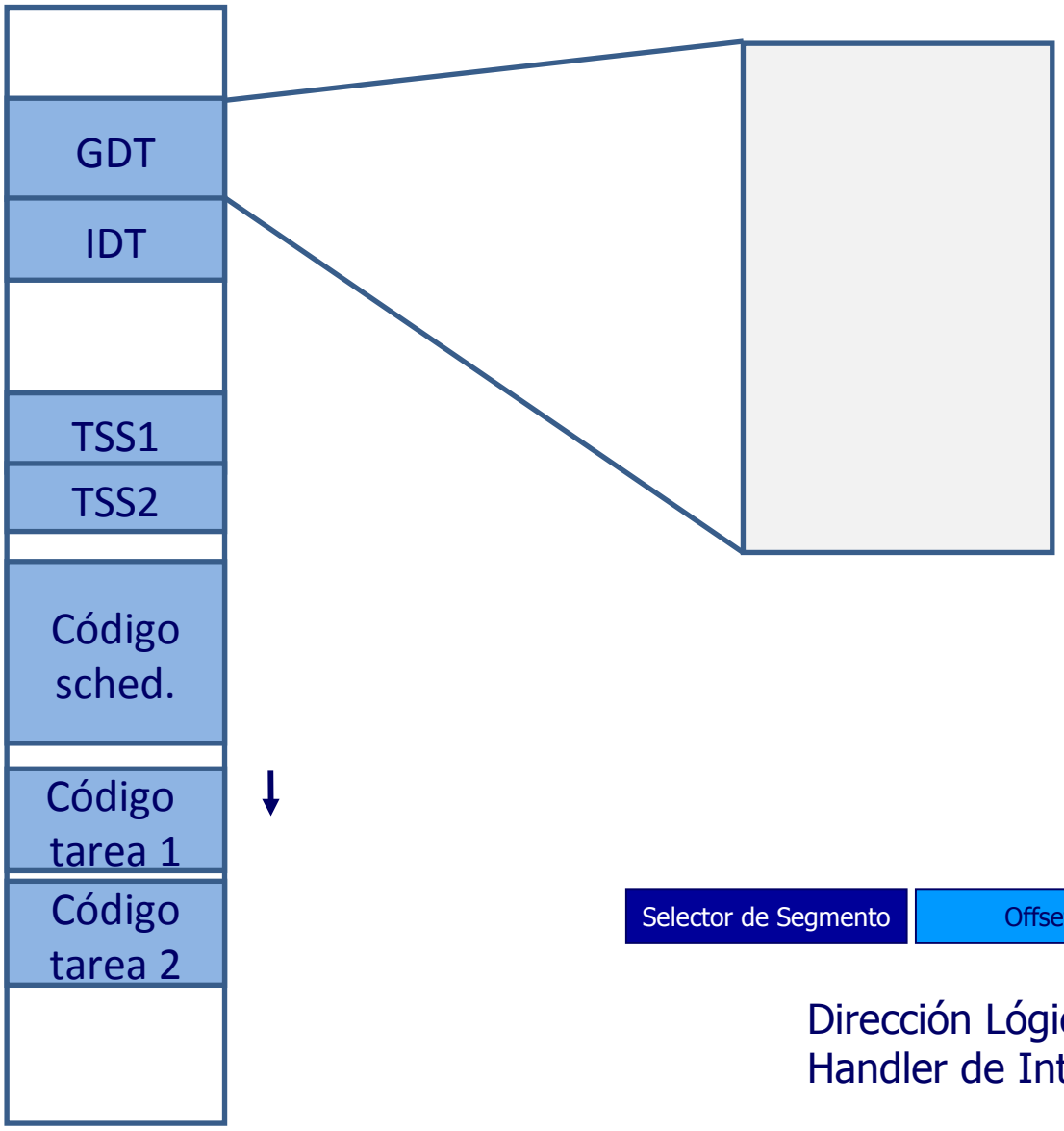


Puerta de Interrupción. Conmutación.



Dirección Lógica punto de entrada del Handler de Interrupción de IRQ_0

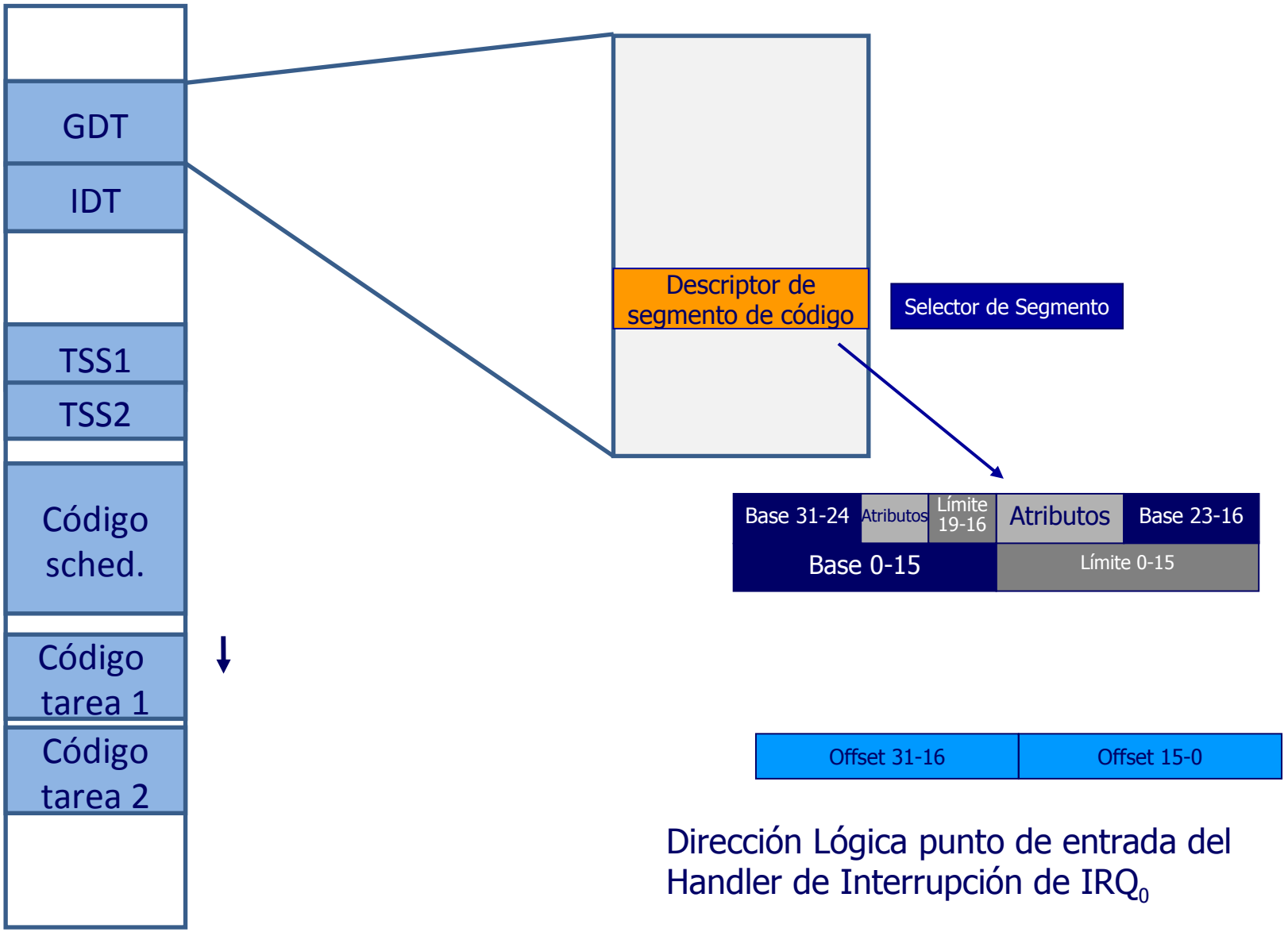
Puerta de Interrupción. Conmutación.



Selector de Segmento	Offset 31-16	Offset 15-0

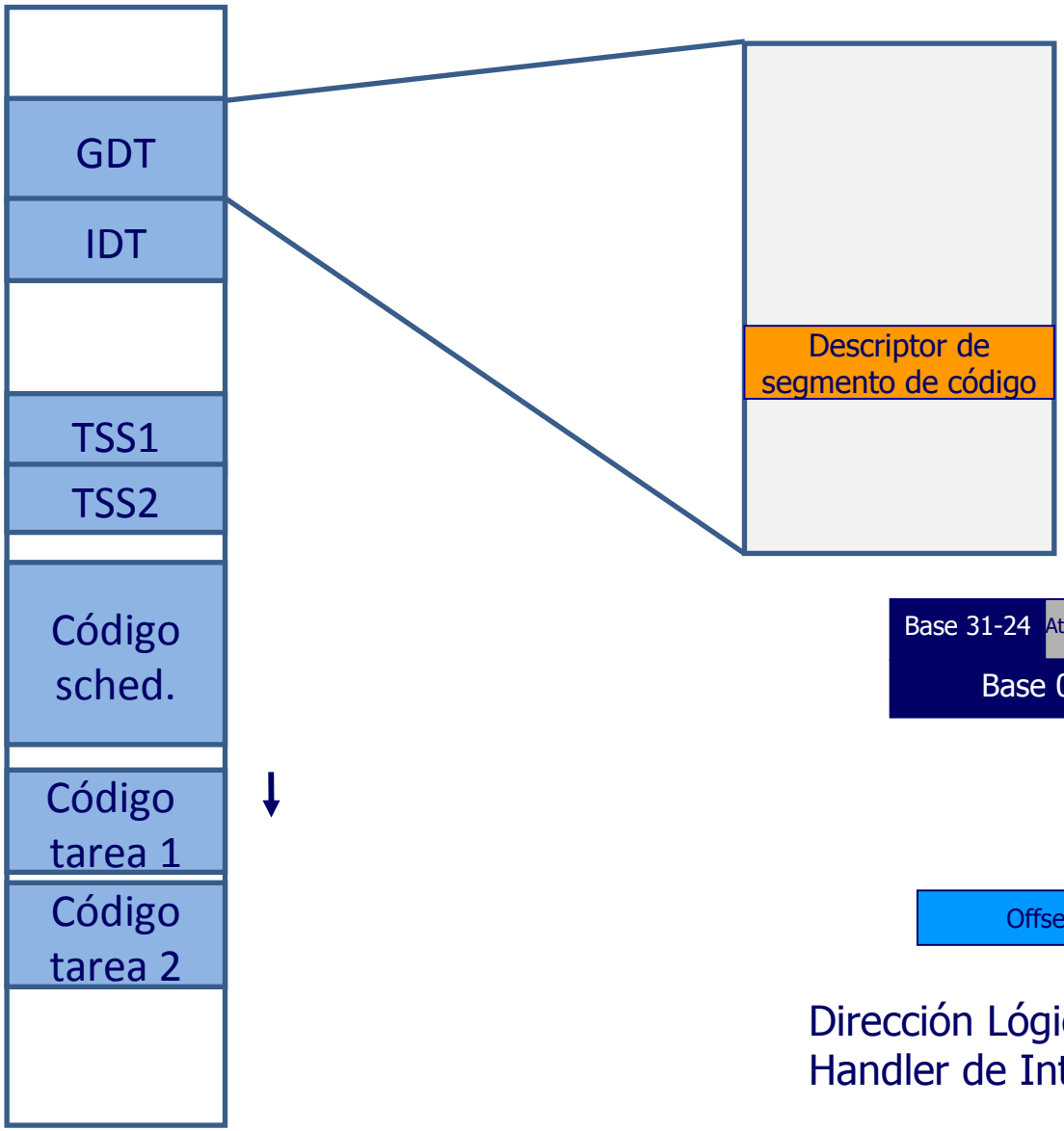
Dirección Lógica punto de entrada del Handler de Interrupción de IRQ_0

Puerta de Interrupción. Conmutación.



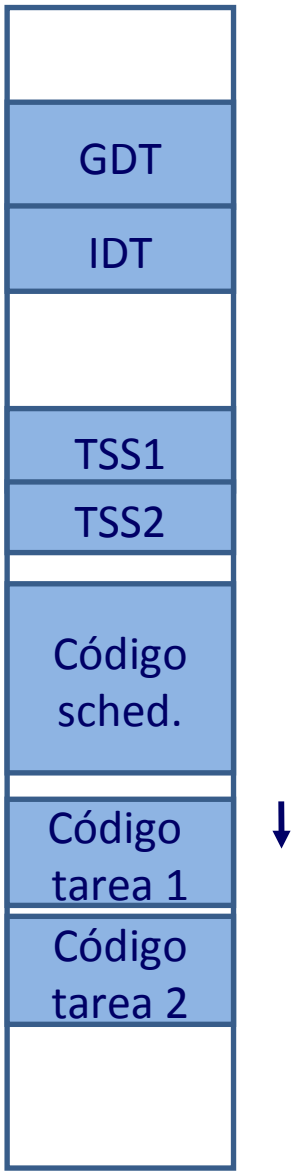
Dirección Lógica punto de entrada del Handler de Interrupción de IRQ_0

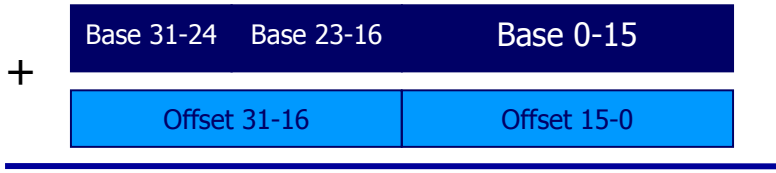
Puerta de Interrupción. Conmutación.



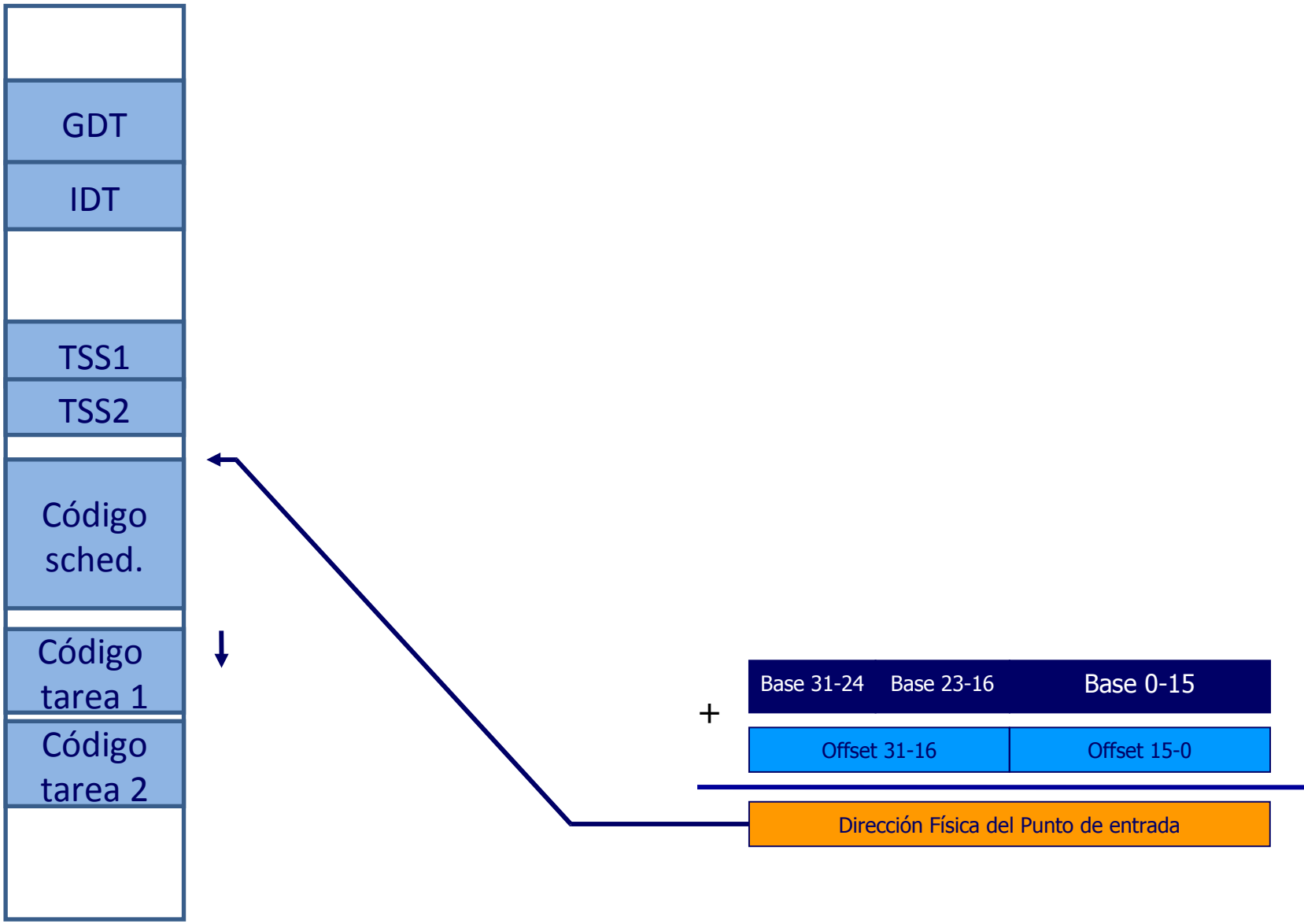
Dirección Lógica punto de entrada del Handler de Interrupción de IRQ_0

Puerta de Interrupción. Conmutación.





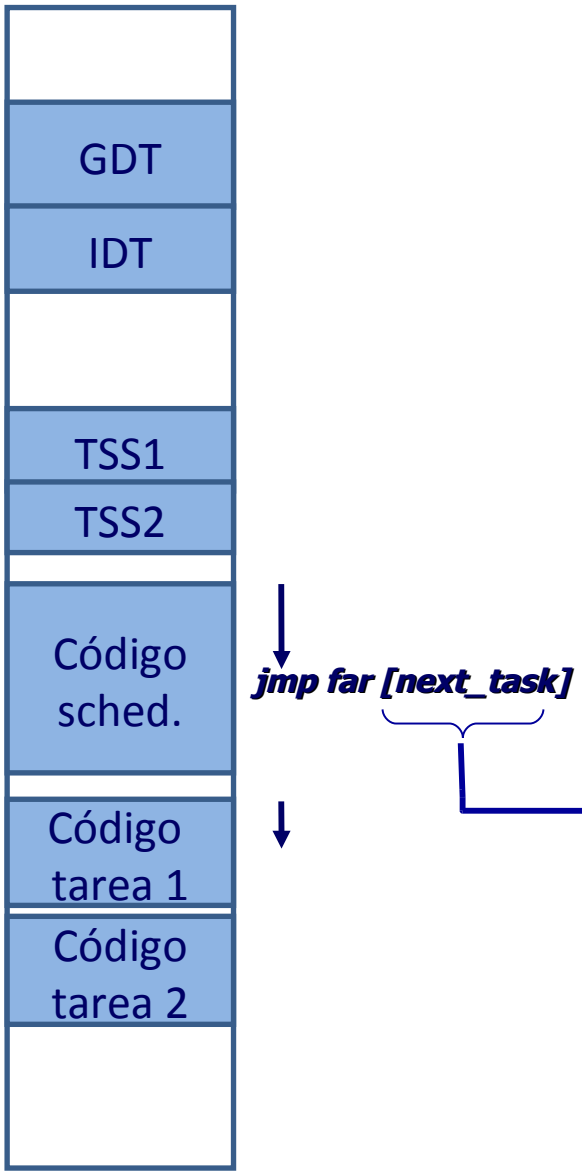
Puerta de Interrupción. Conmutación.



Puerta de Interrupción. Conmutación.

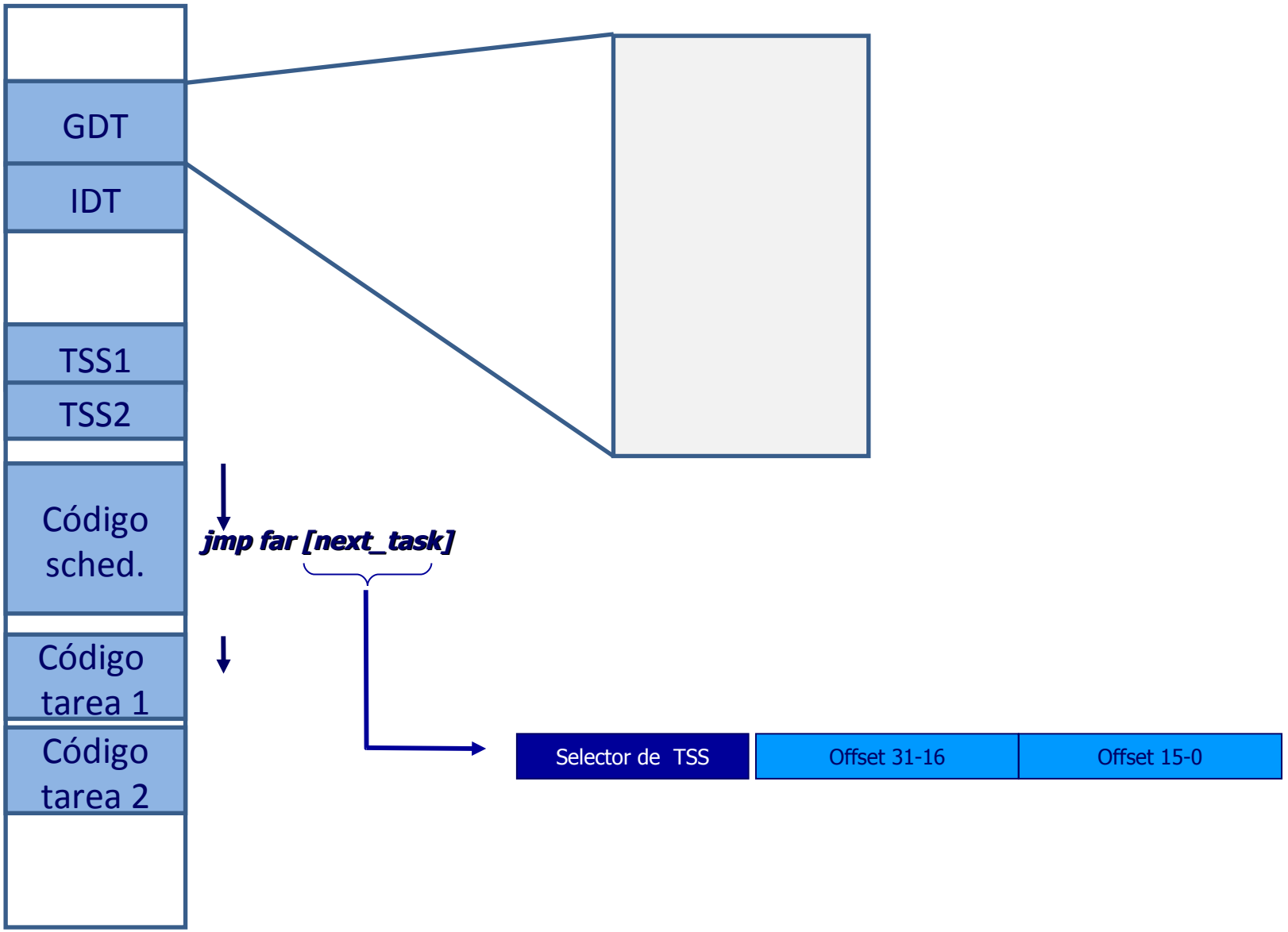
Se comienza a ejecutar el código del scheduler.
Contexto: Tarea 1
No hubo cambio de contexto

Puerta de Interrupción. Conmutación.

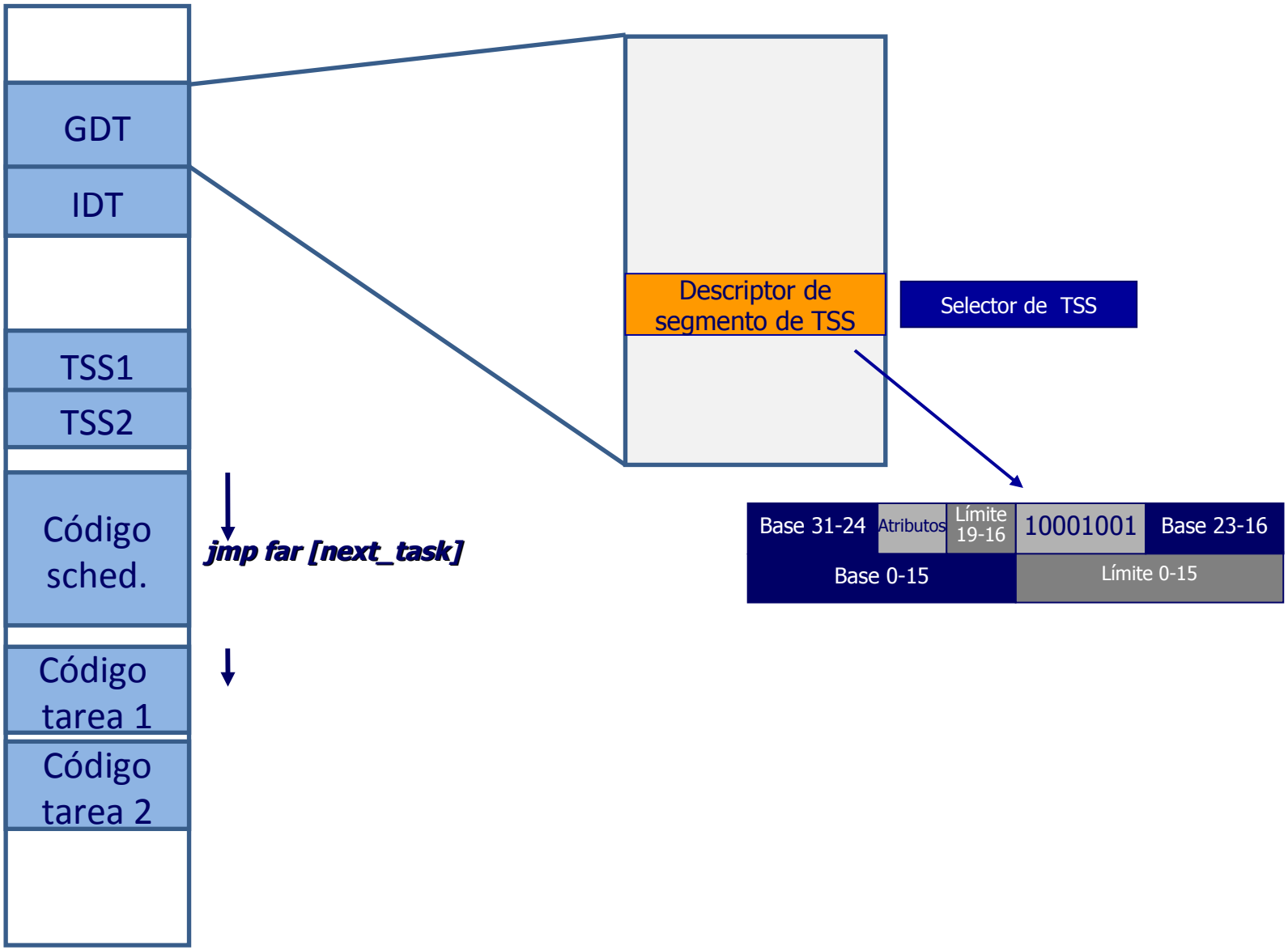


La variable ***next_task***, se compone de Selector de TSS y un offset (el que finalmente será descartado por el procesador)

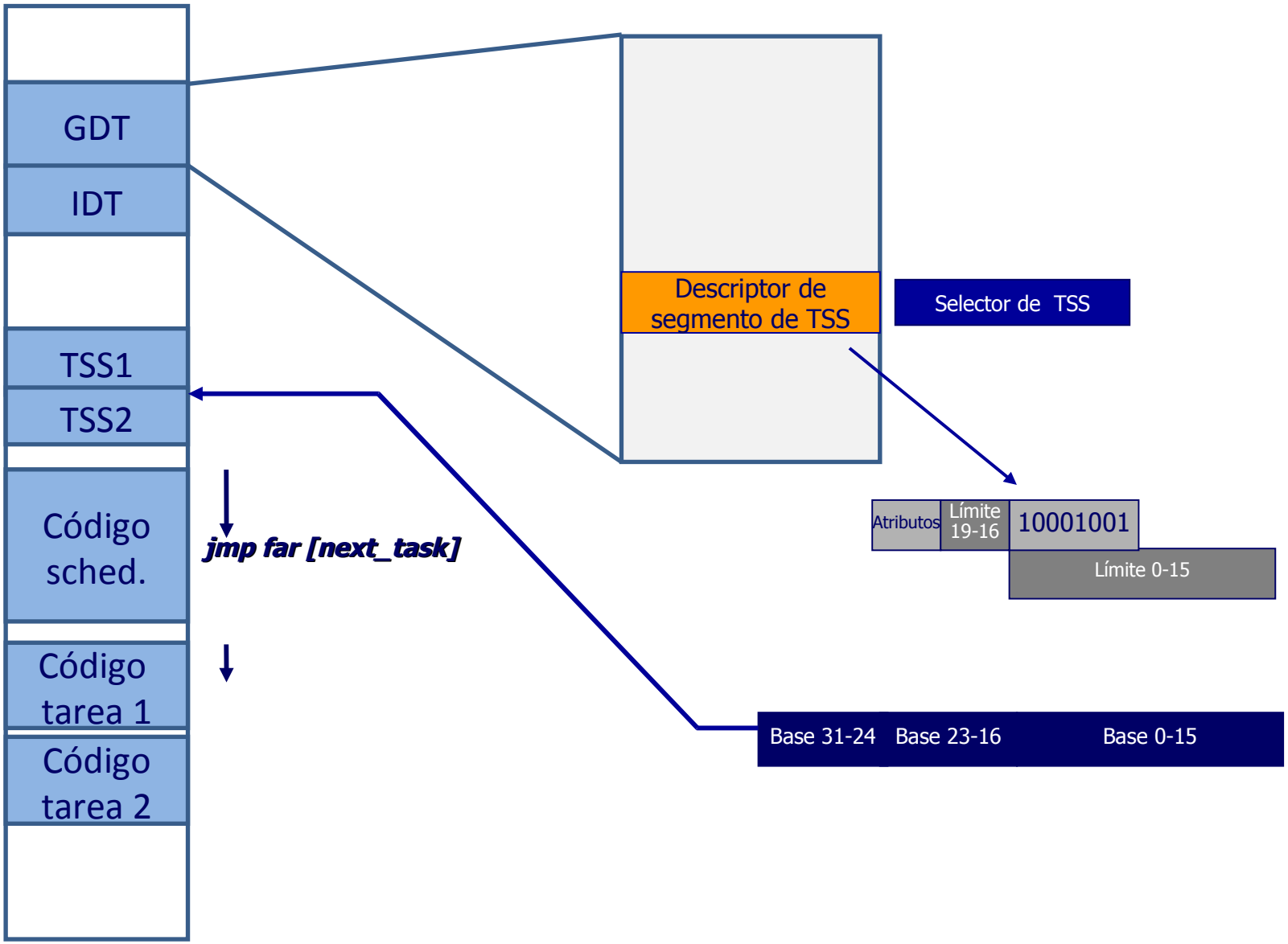
Puerta de Interrupción. Conmutación.



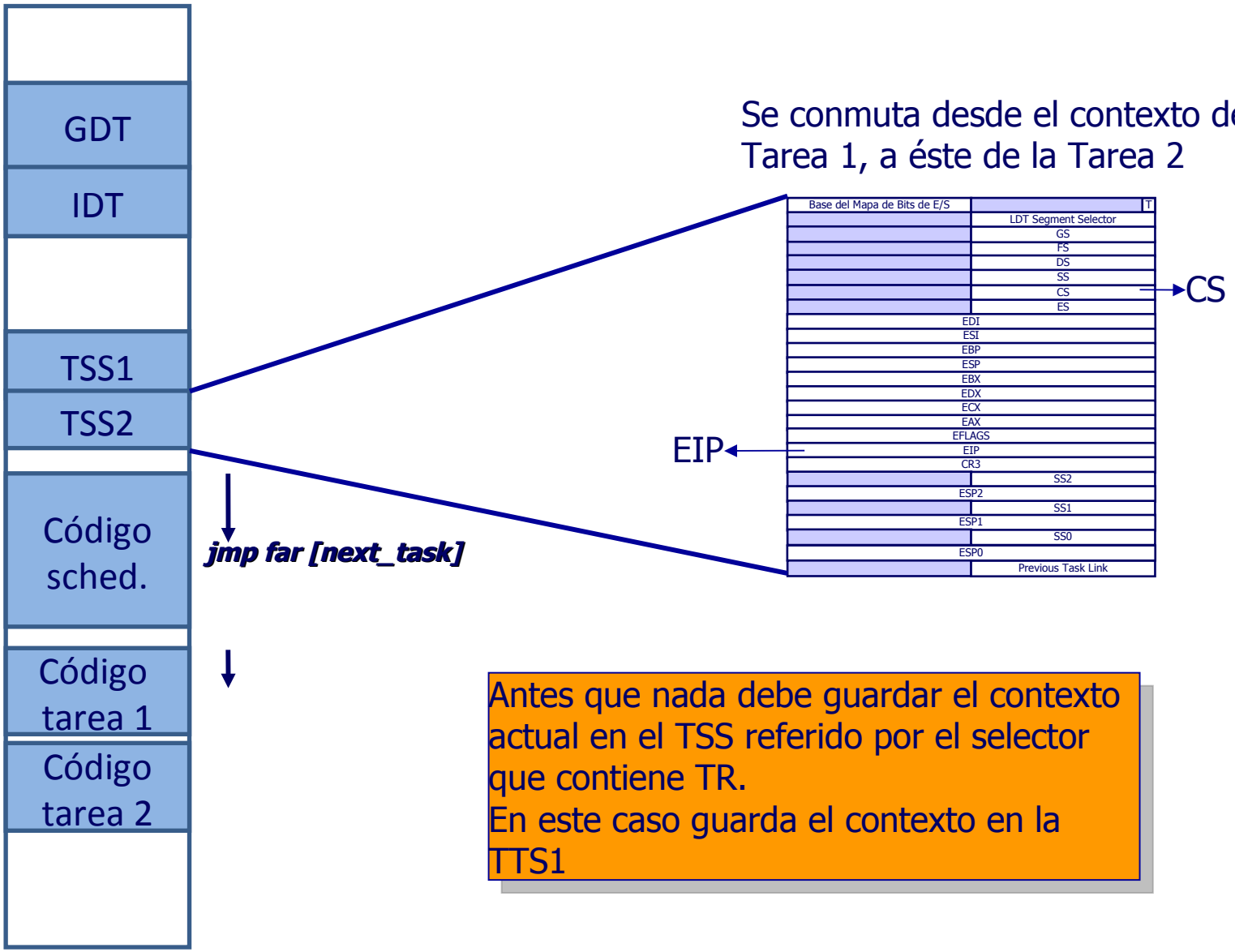
Puerta de Interrupción. Conmutación.



Puerta de Interrupción. Conmutación.

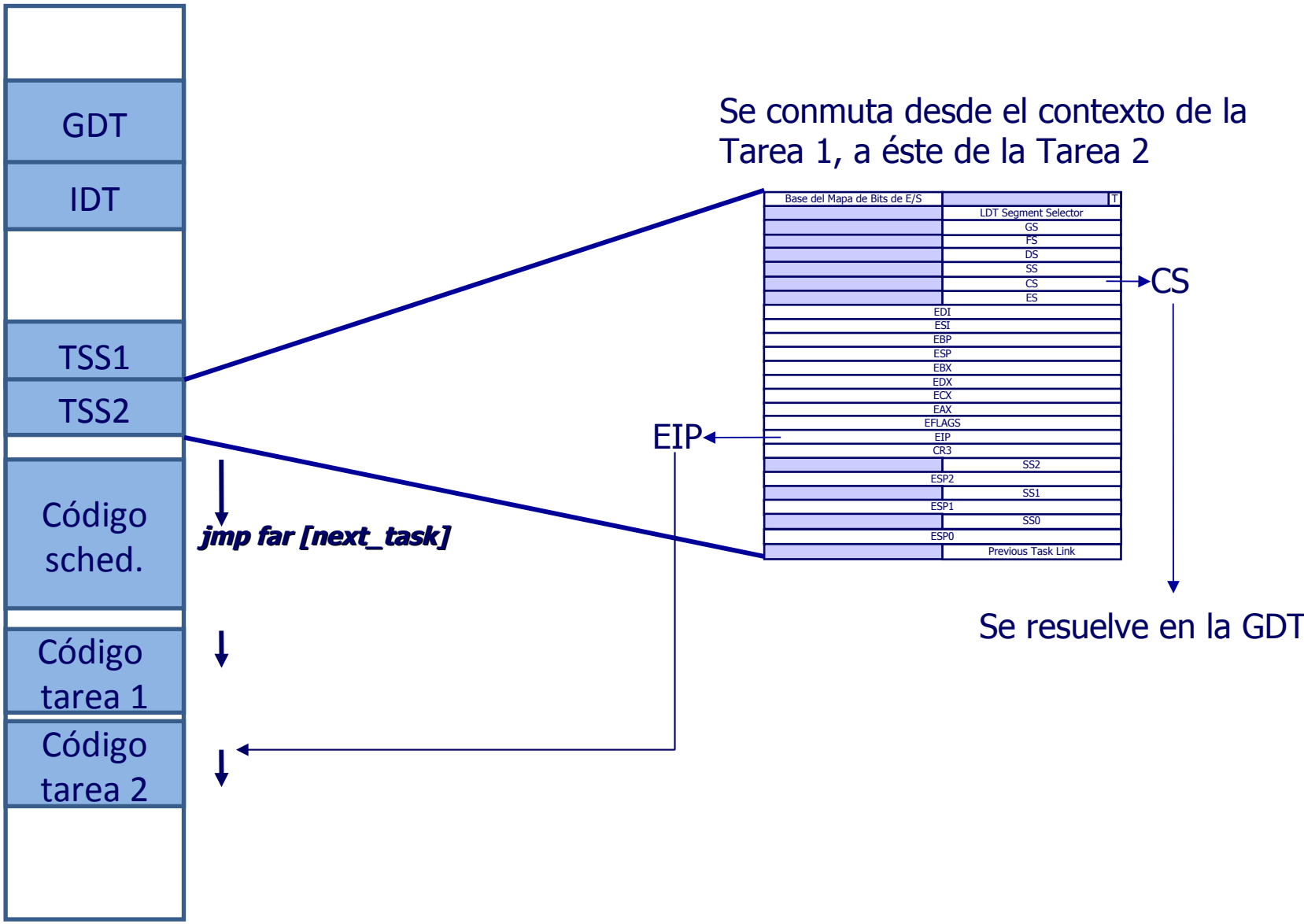


Puerta de Interrupción. Conmutación.

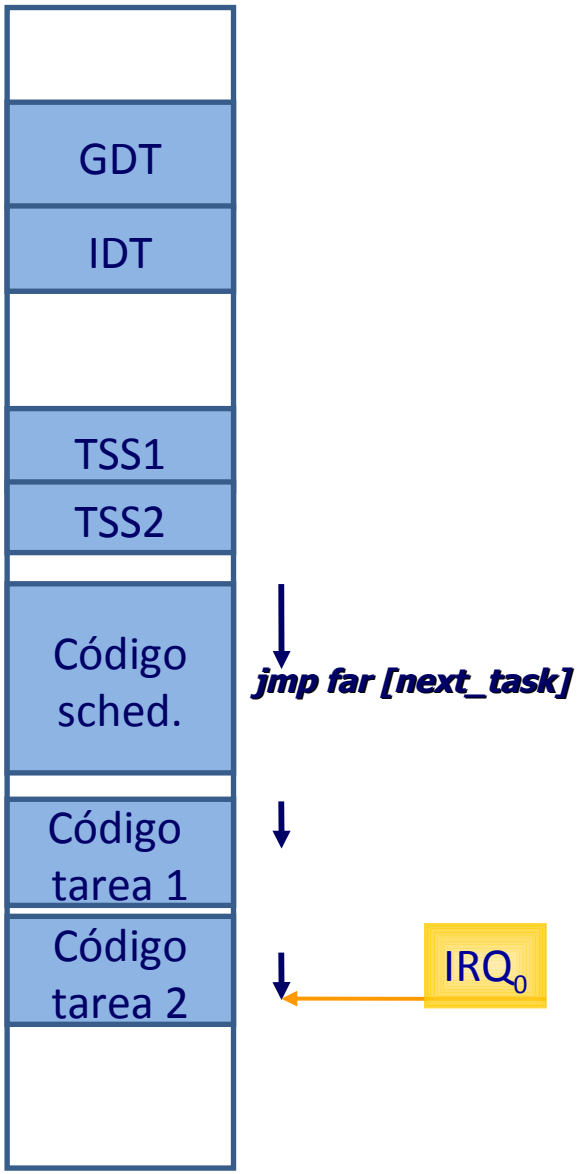


Antes que nada debe guardar el contexto actual en el TSS referido por el selector que contiene TR.
En este caso guarda el contexto en la TTS1

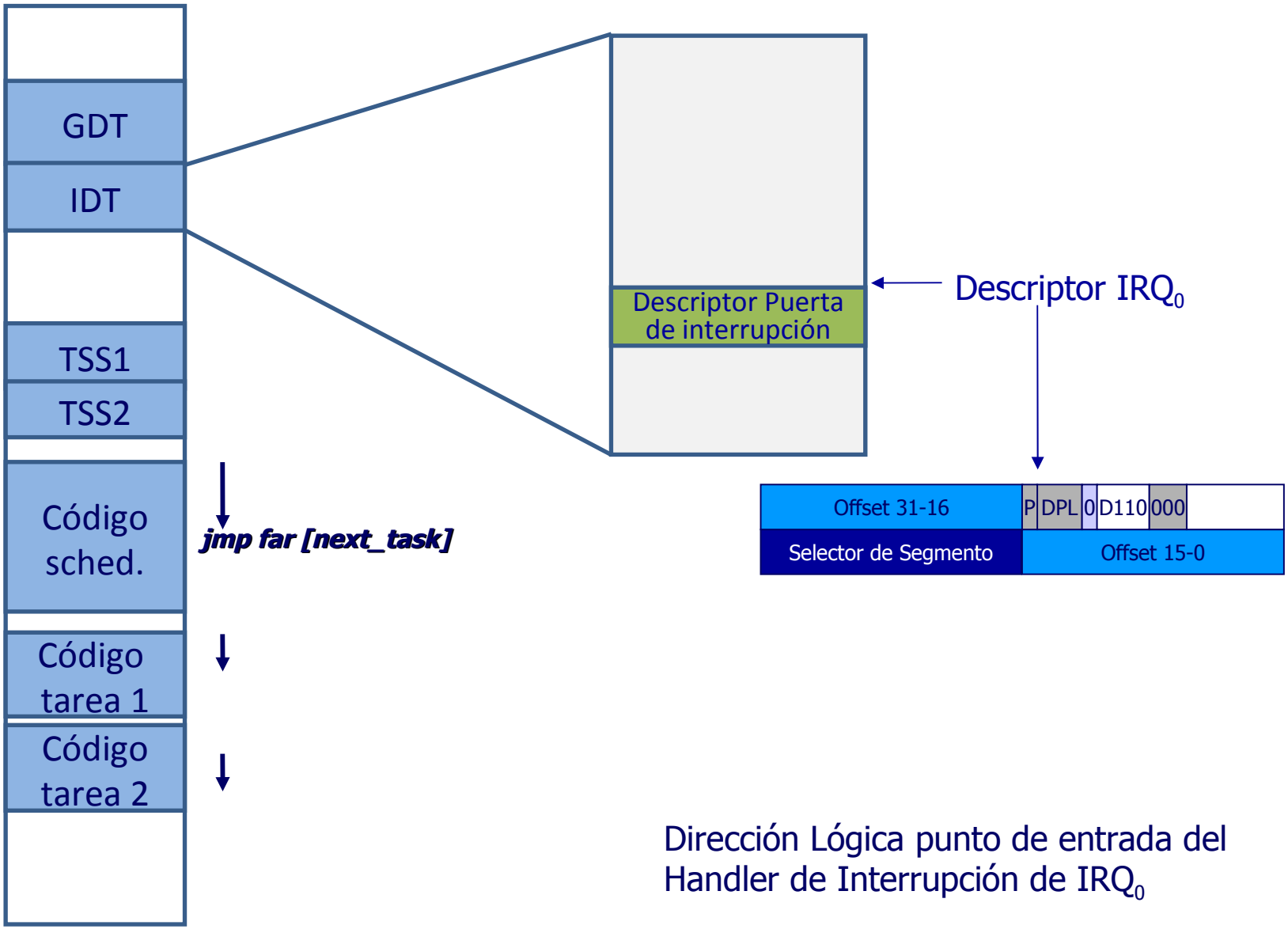
Puerta de Interrupción. Conmutación.



Puerta de Interrupción. Conmutación.

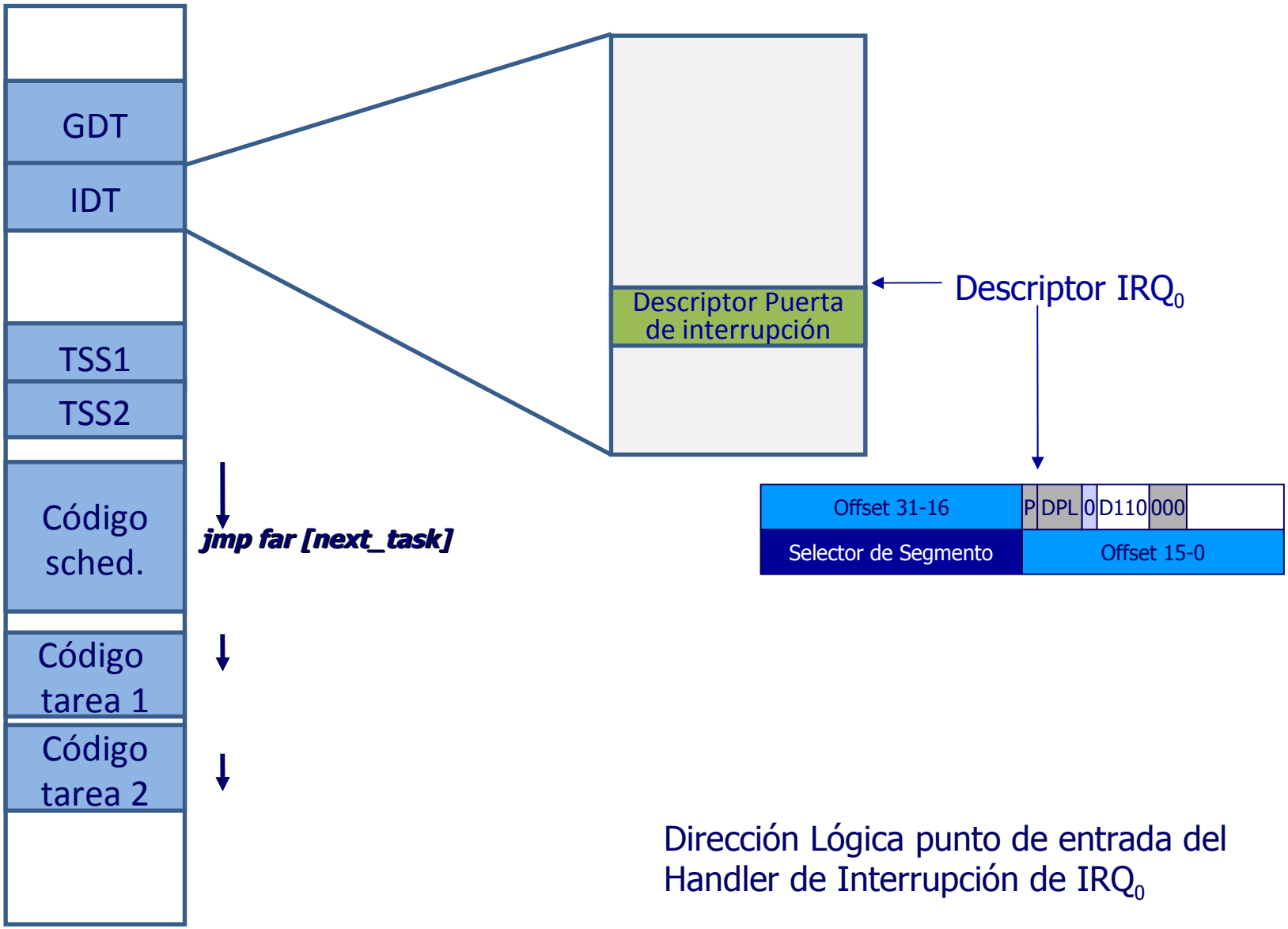


Puerta de Interrupción. Conmutación.



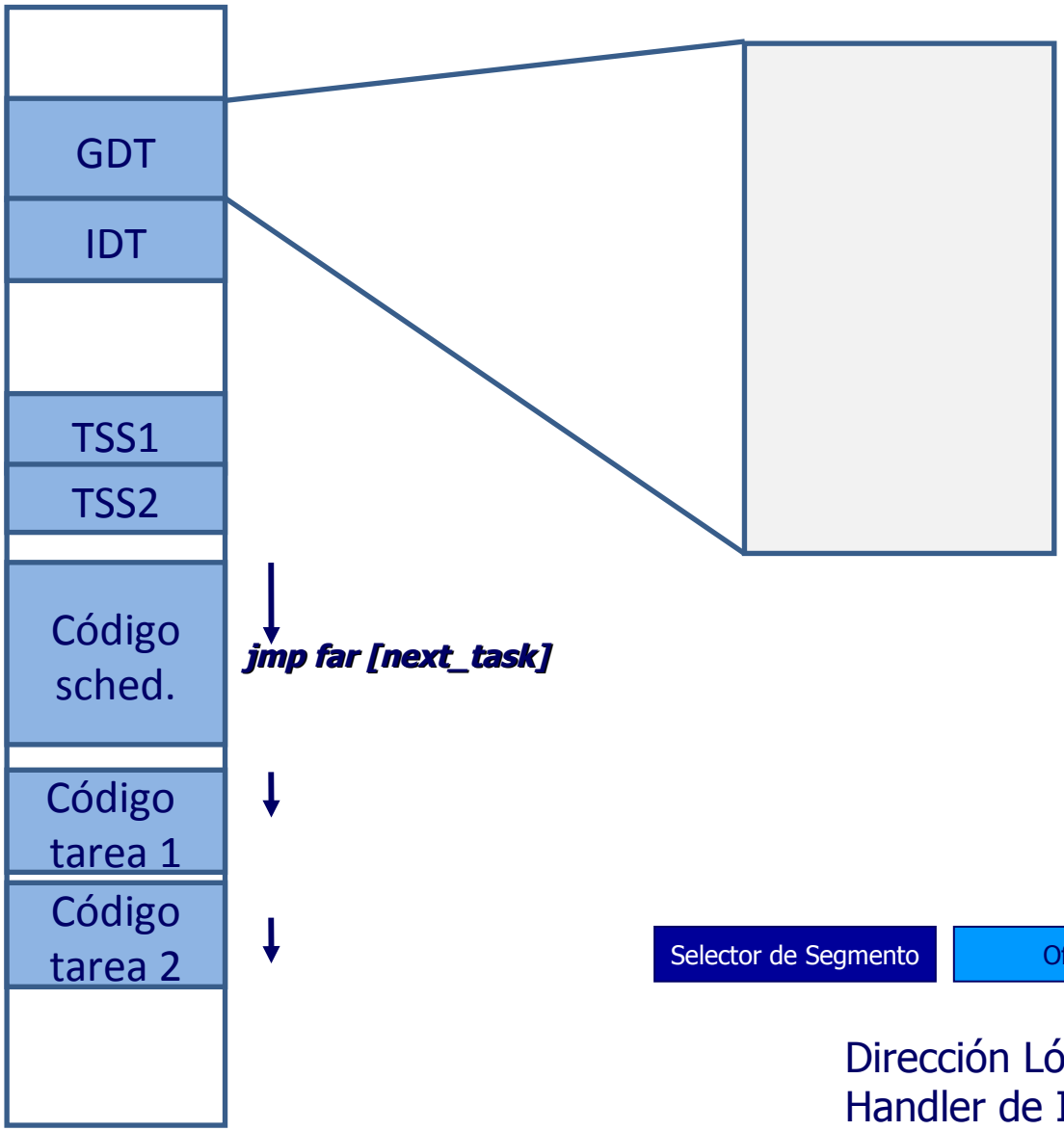
Dirección Lógica punto de entrada del Handler de Interrupción de IRQ_0

Puerta de Interrupción. Conmutación.



Dirección Lógica punto de entrada del Handler de Interrupción de IRQ_0

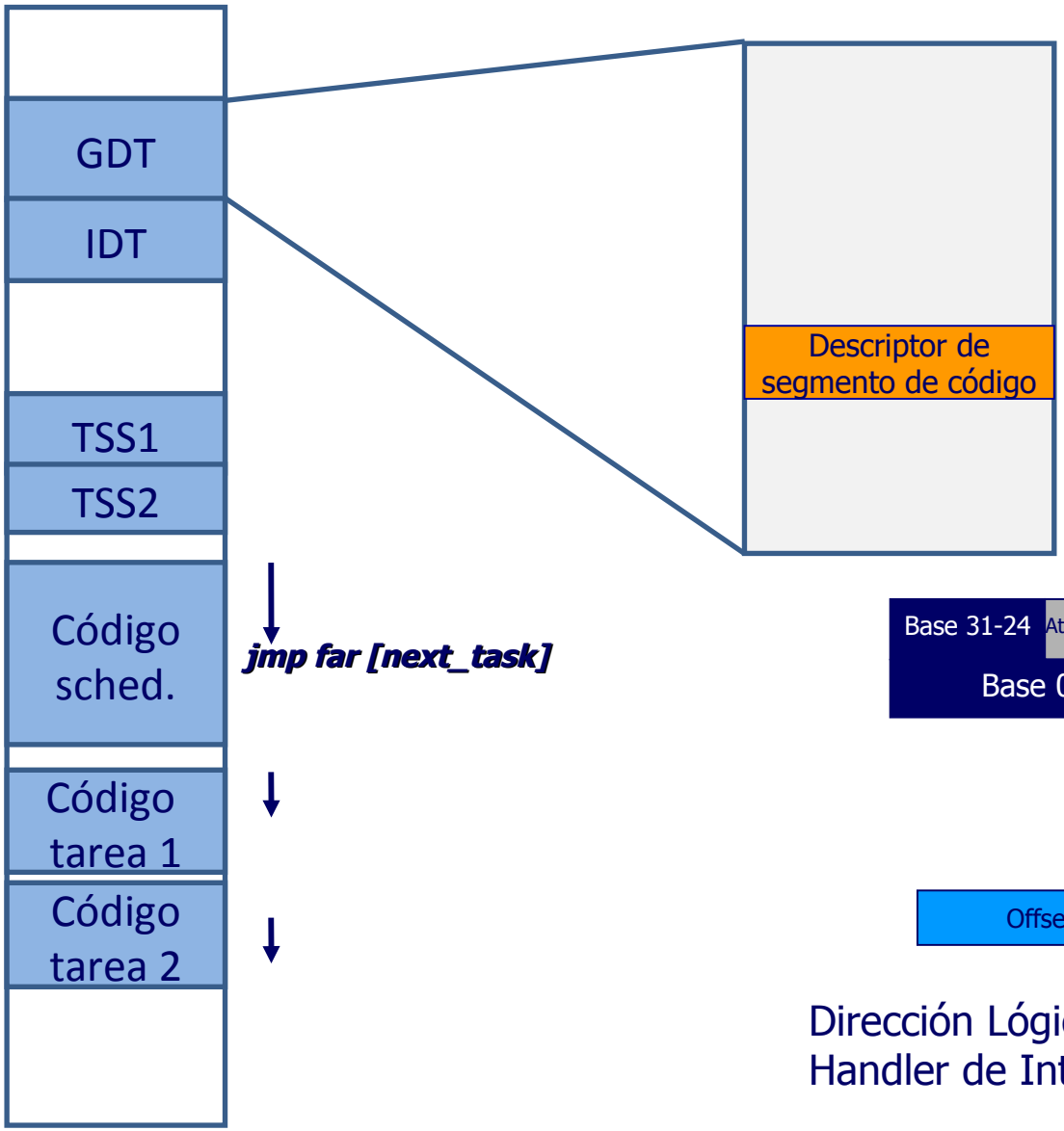
Puerta de Interrupción. Conmutación.



Selector de Segmento	Offset 31-16	Offset 15-0

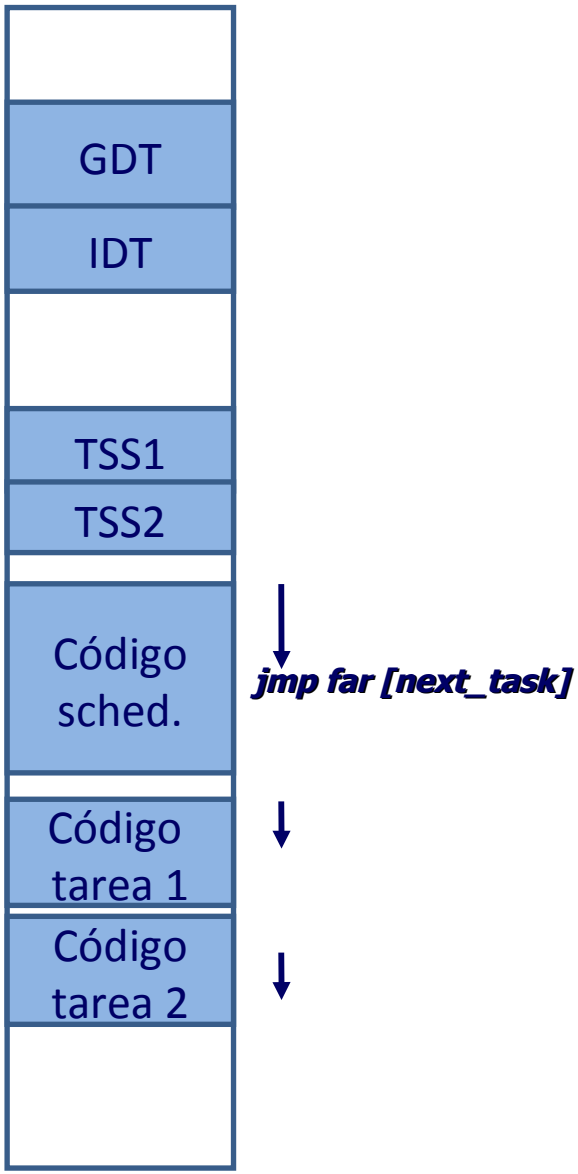
Dirección Lógica punto de entrada del Handler de Interrupción de IRQ_0

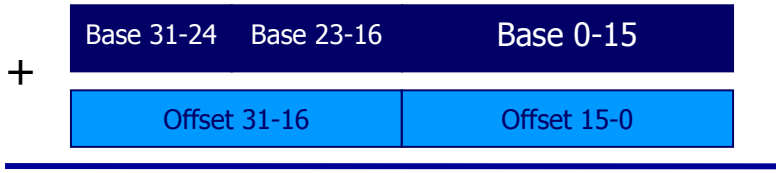
Puerta de Interrupción. Conmutación.



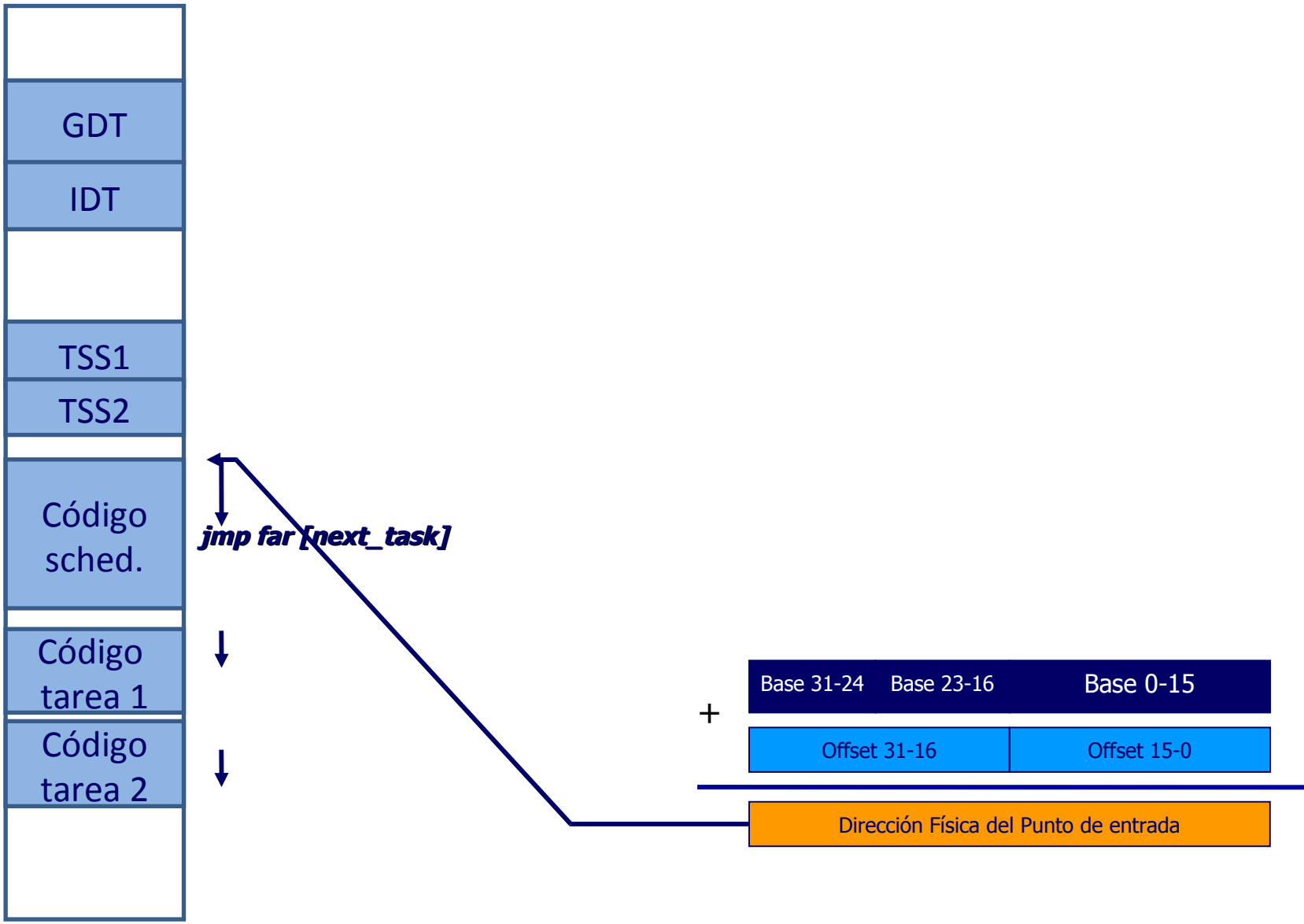
Dirección Lógica punto de entrada del Handler de Interrupción de IRQ_0

Puerta de Interrupción. Conmutación.

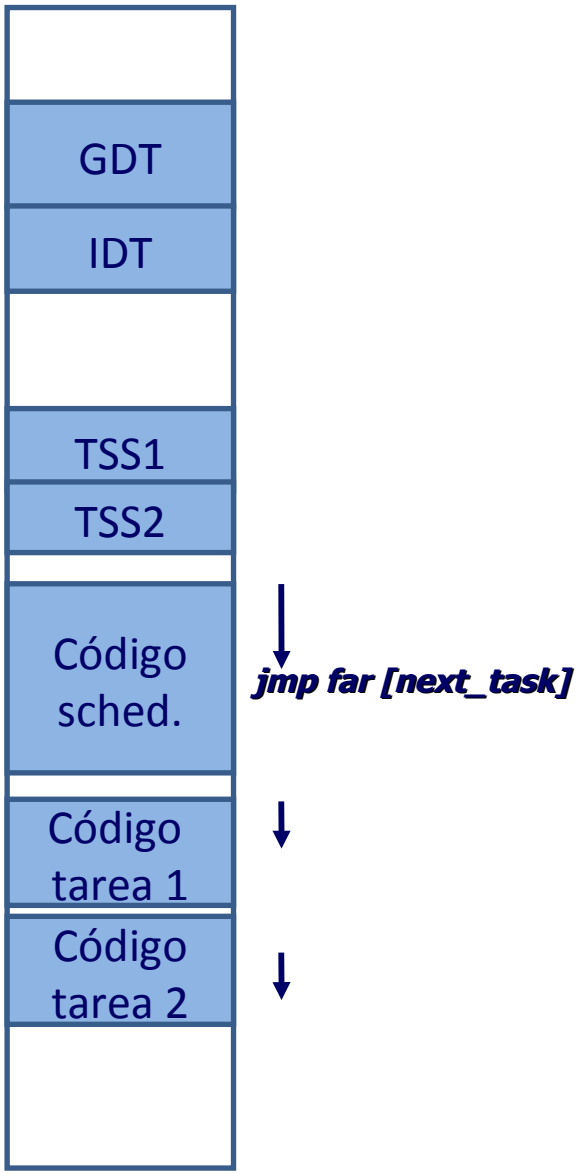




Puerta de Interrupción. Conmutación.

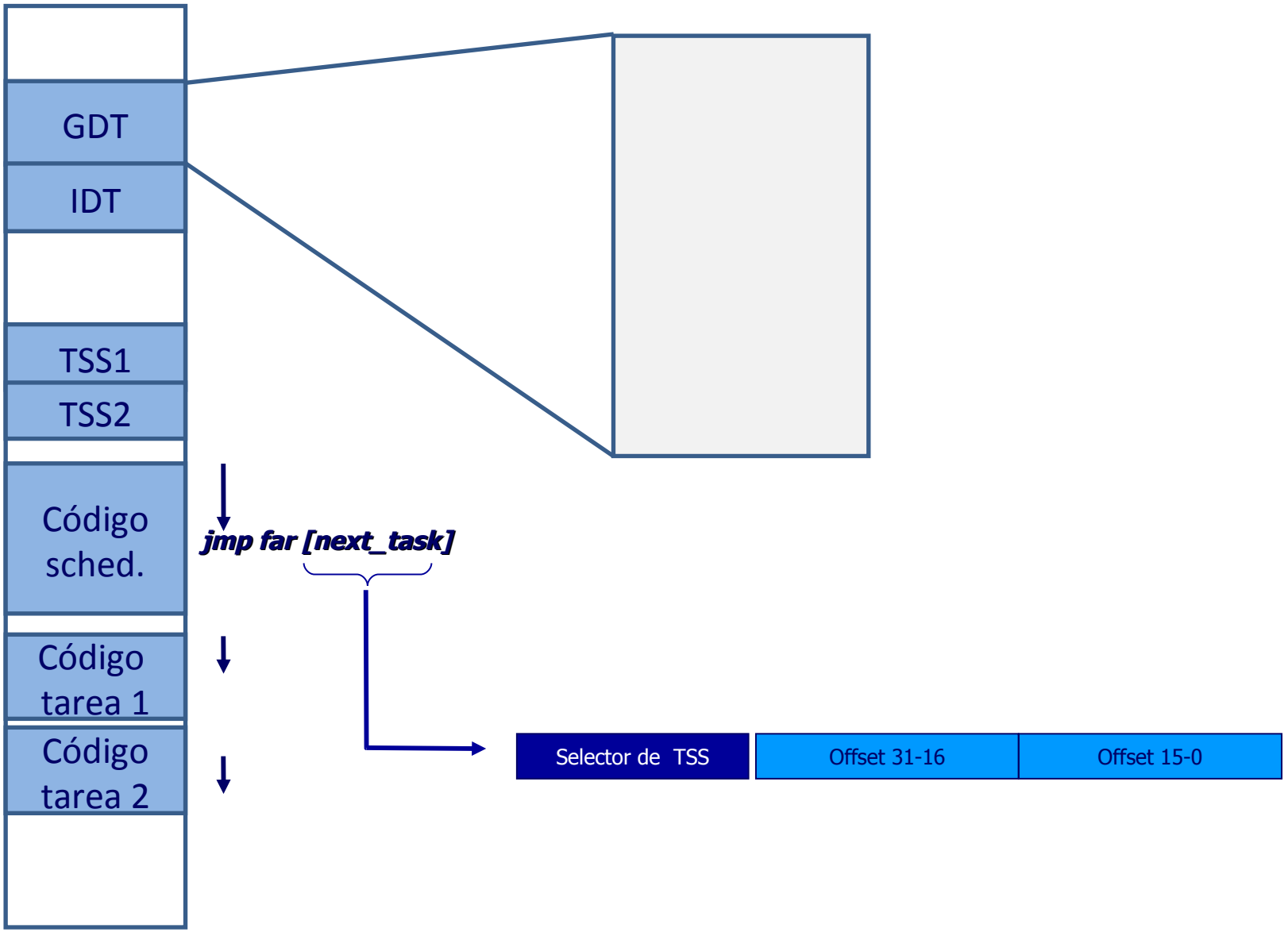


Puerta de Interrupción. Conmutación.

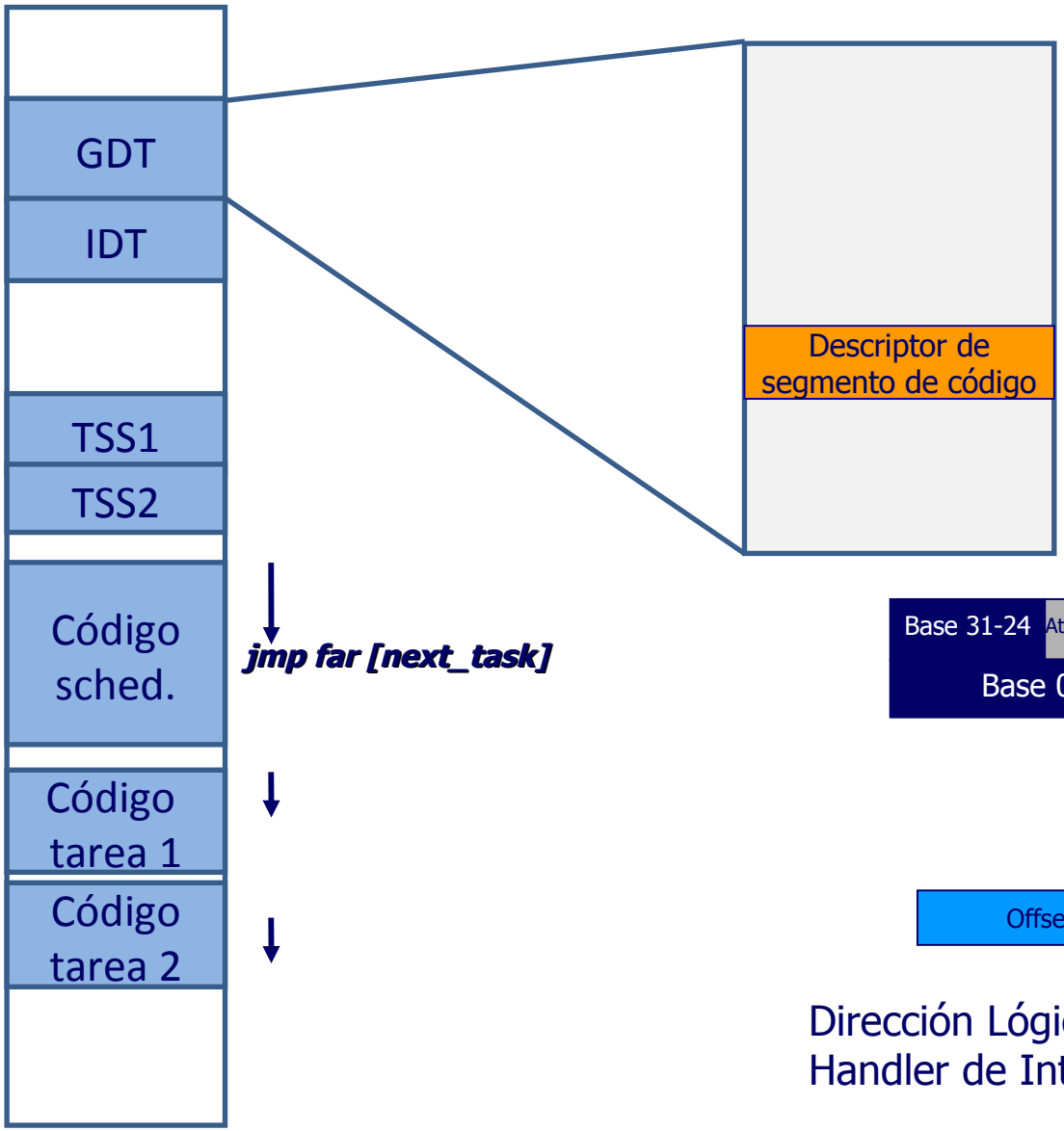


Se comienza a ejecutar el código del scheduler.
Contexto: Tarea 2
No hubo cambio de contexto

Puerta de Interrupción. Conmutación.

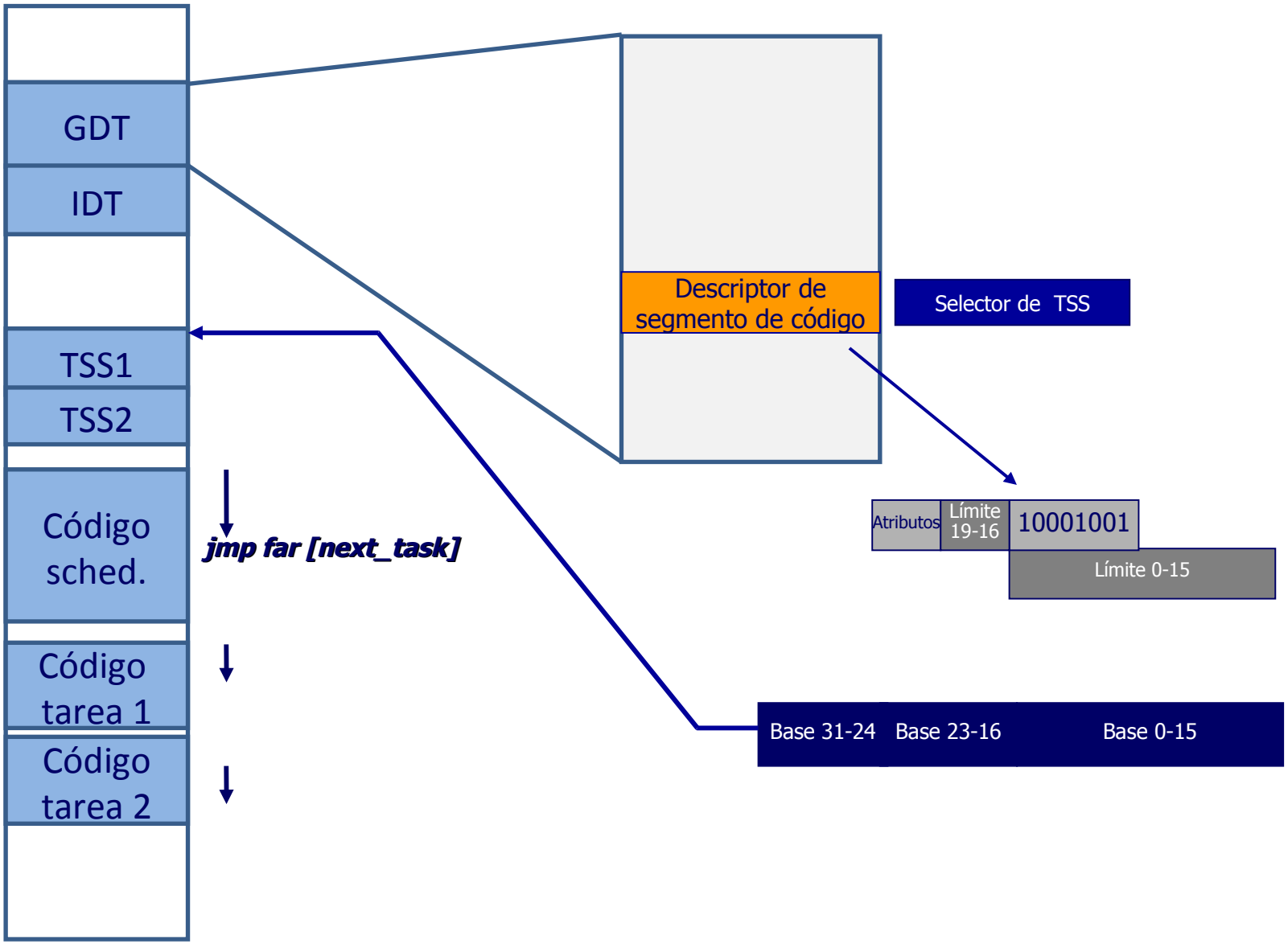


Puerta de Interrupción. Conmutación.

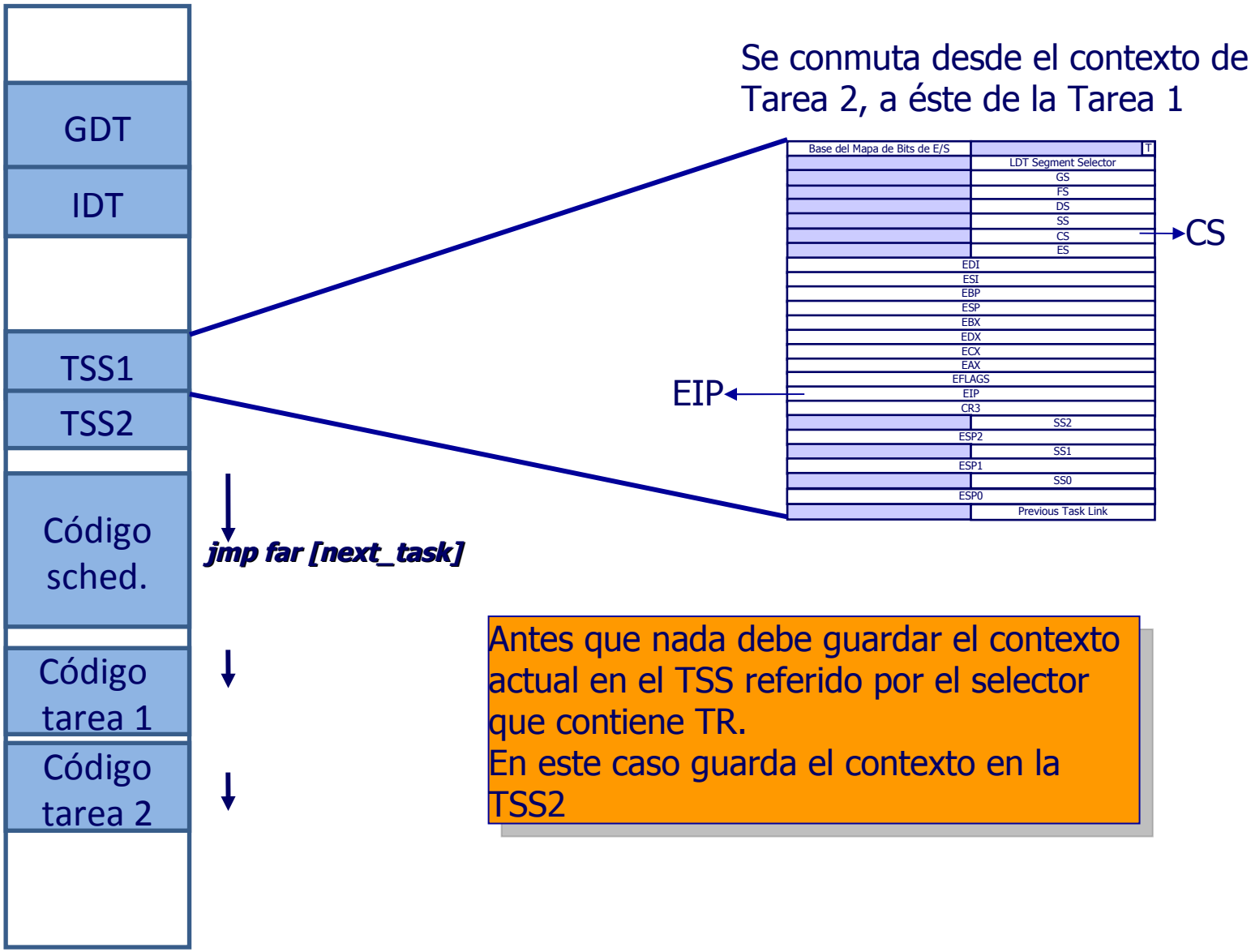


Dirección Lógica punto de entrada del Handler de Interrupción de IRQ_0

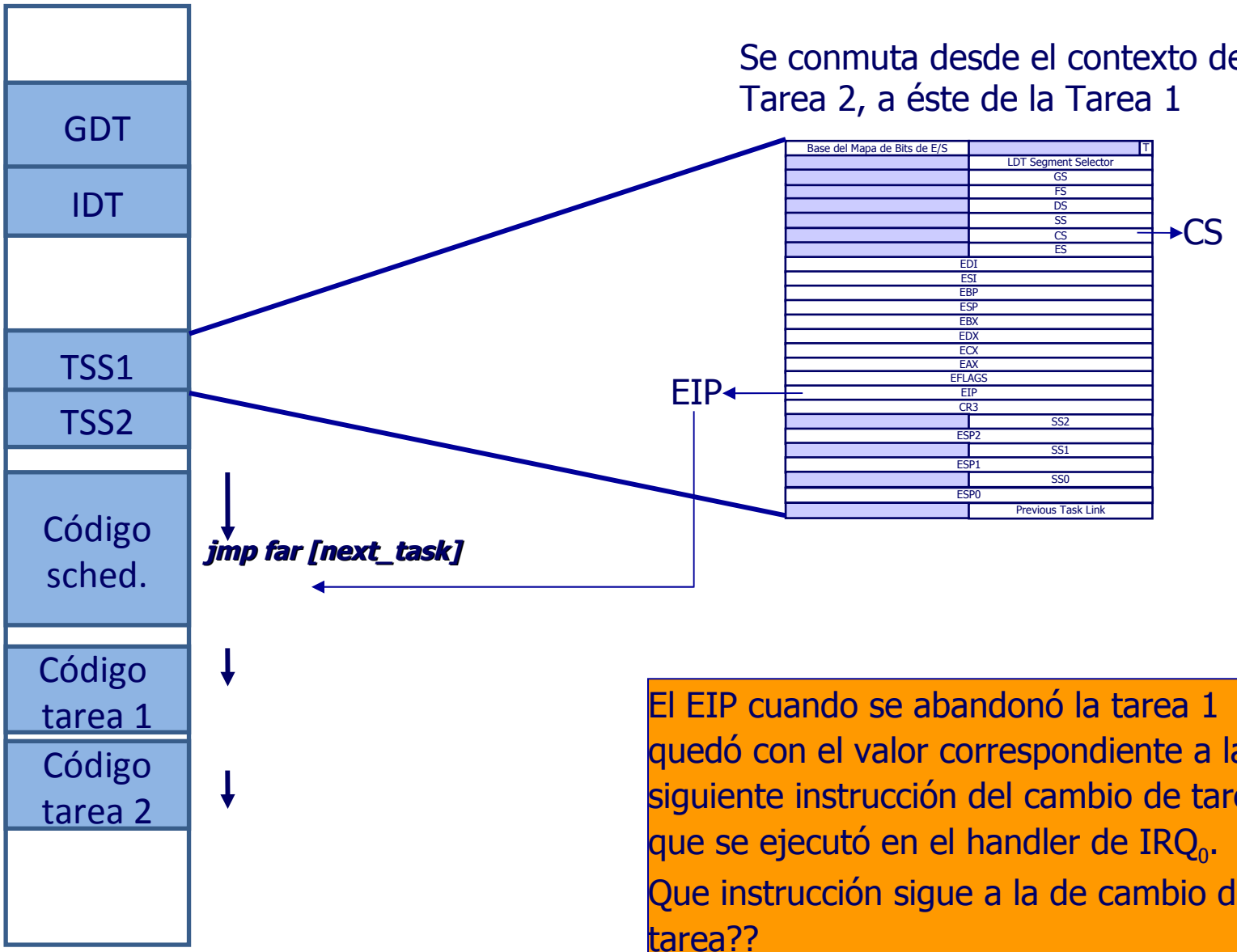
Puerta de Interrupción. Conmutación.



Puerta de Interrupción. Conmutación.



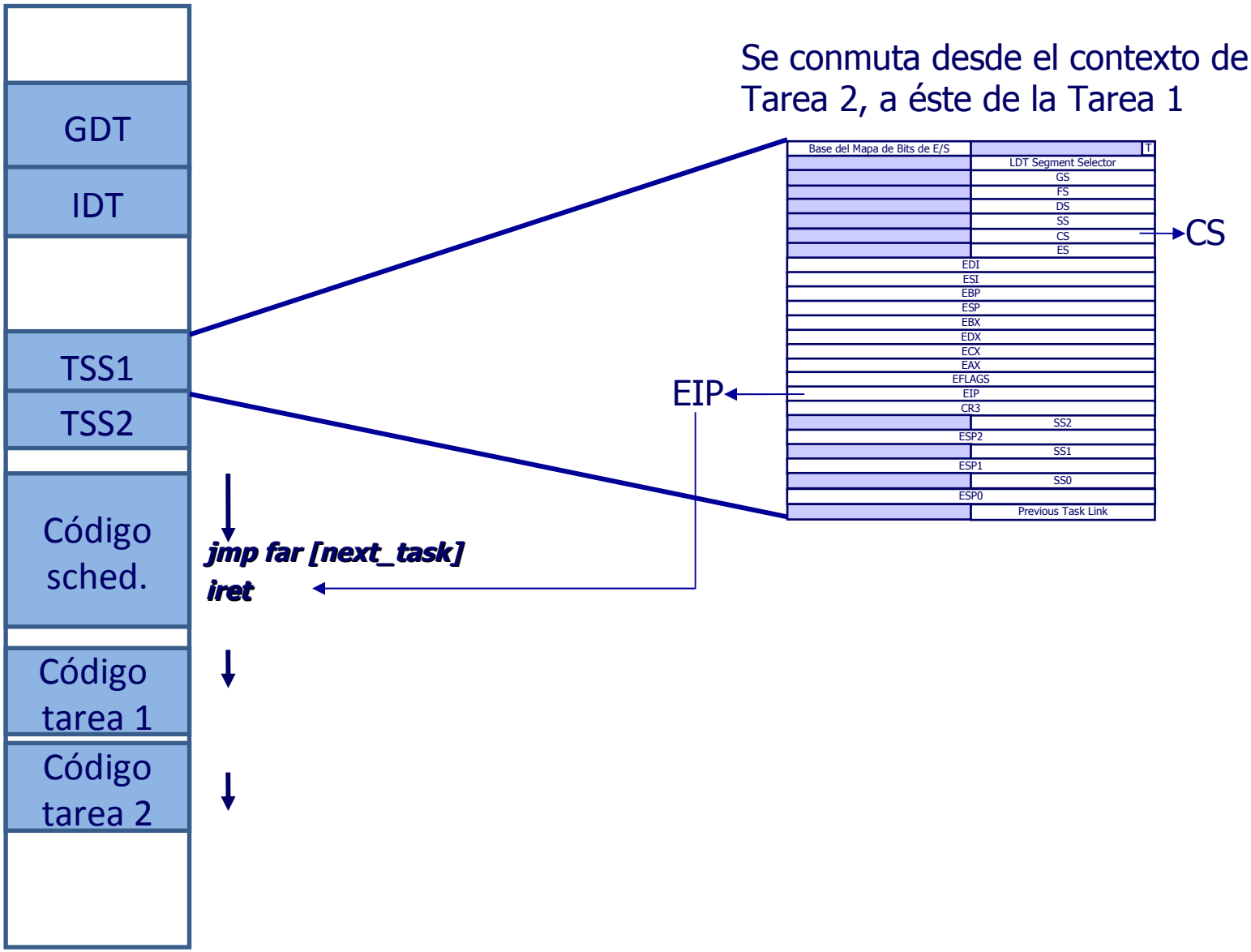
Puerta de Interrupción. Conmutación.



El EIP cuando se abandonó la tarea 1 quedó con el valor correspondiente a la siguiente instrucción del cambio de tarea, que se ejecutó en el handler de IRQ_0.

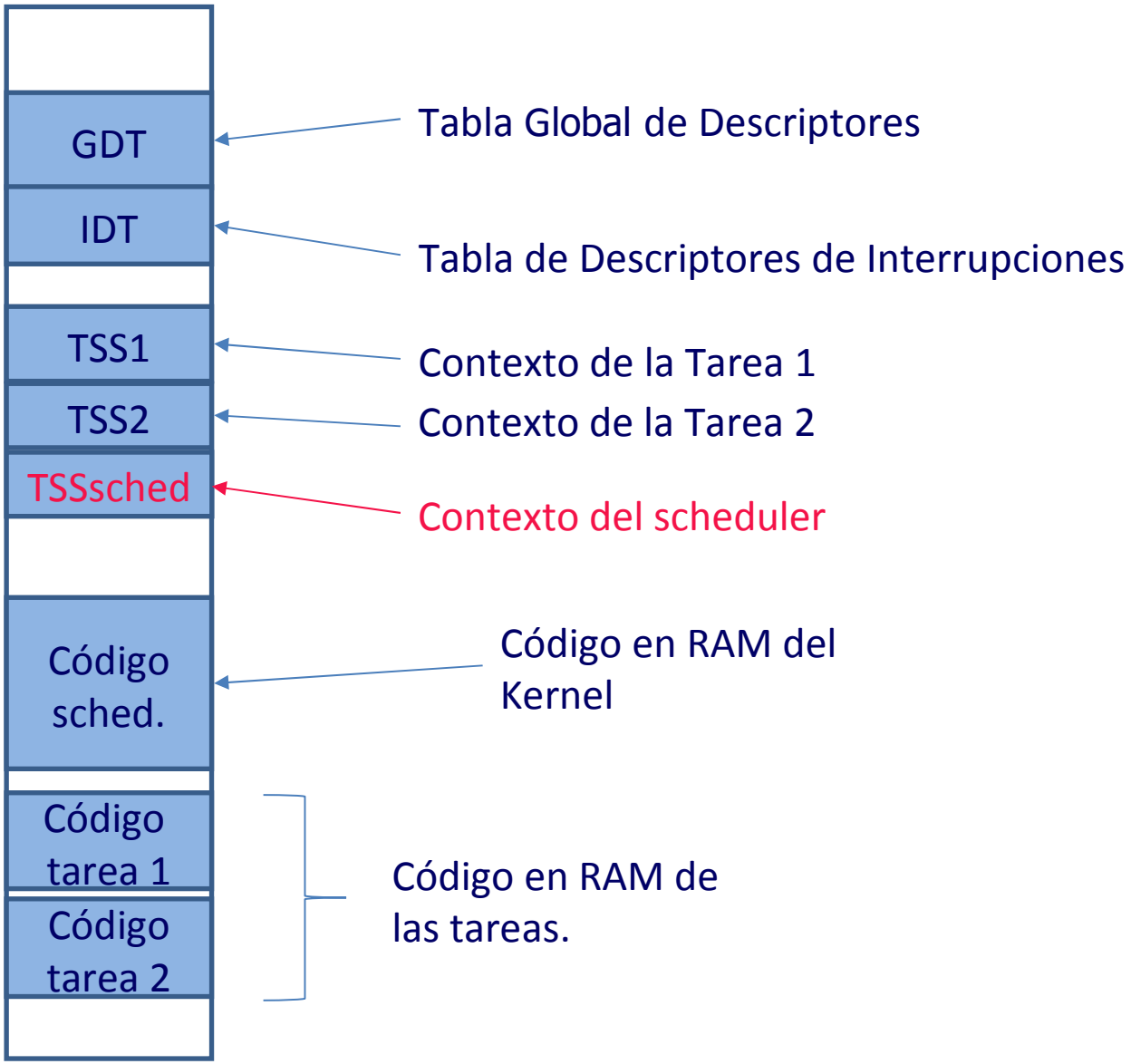
Que instrucción sigue a la de cambio de tarea??

Puerta de Interrupción. Conmutación.



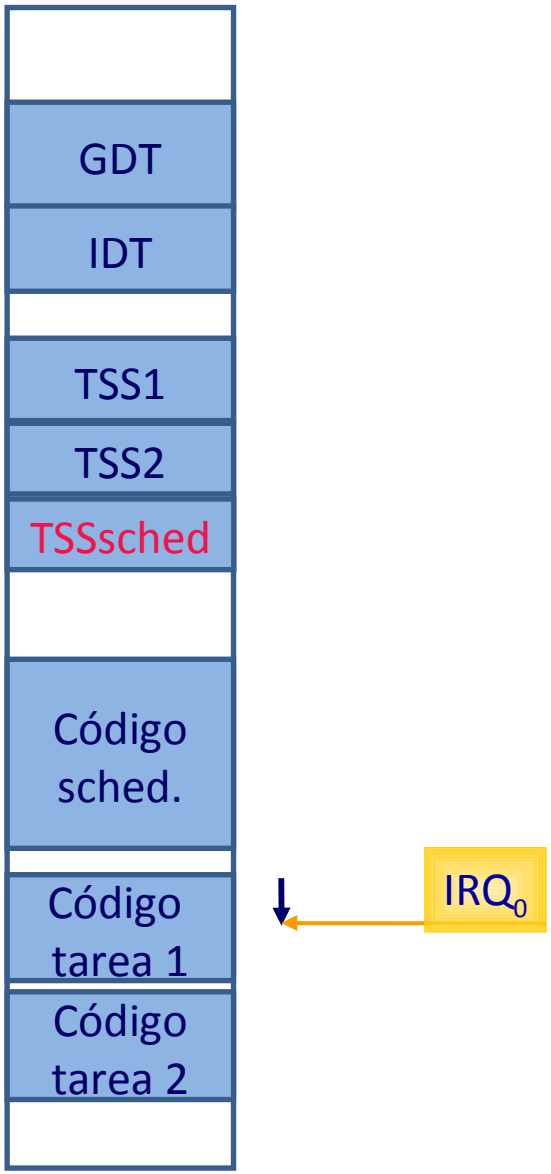
IRQ_0 por puerta de Tarea

Puerta de Tarea. Organización de la memoria (2 tareas)

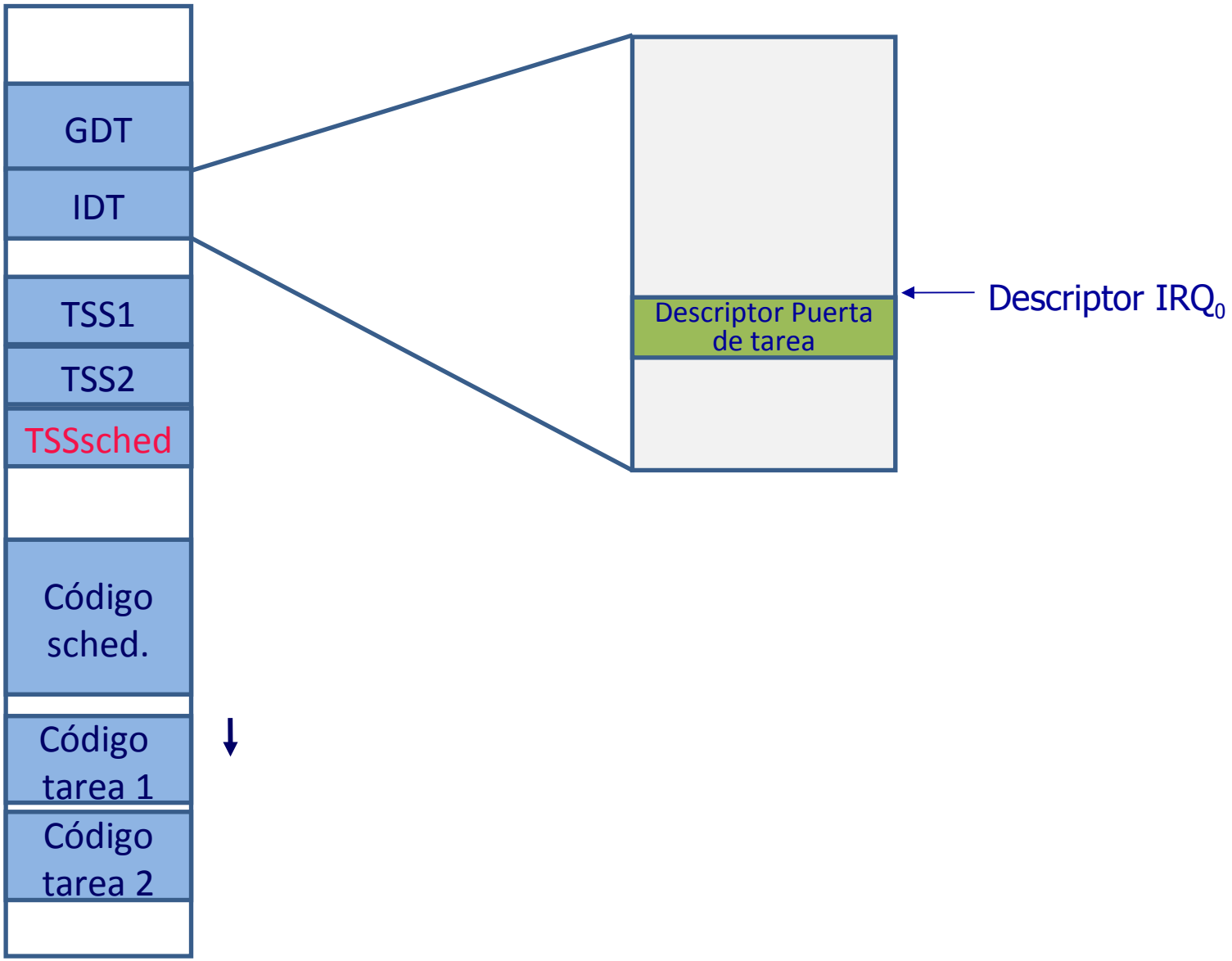


Puerta de Tarea. Conmutación.

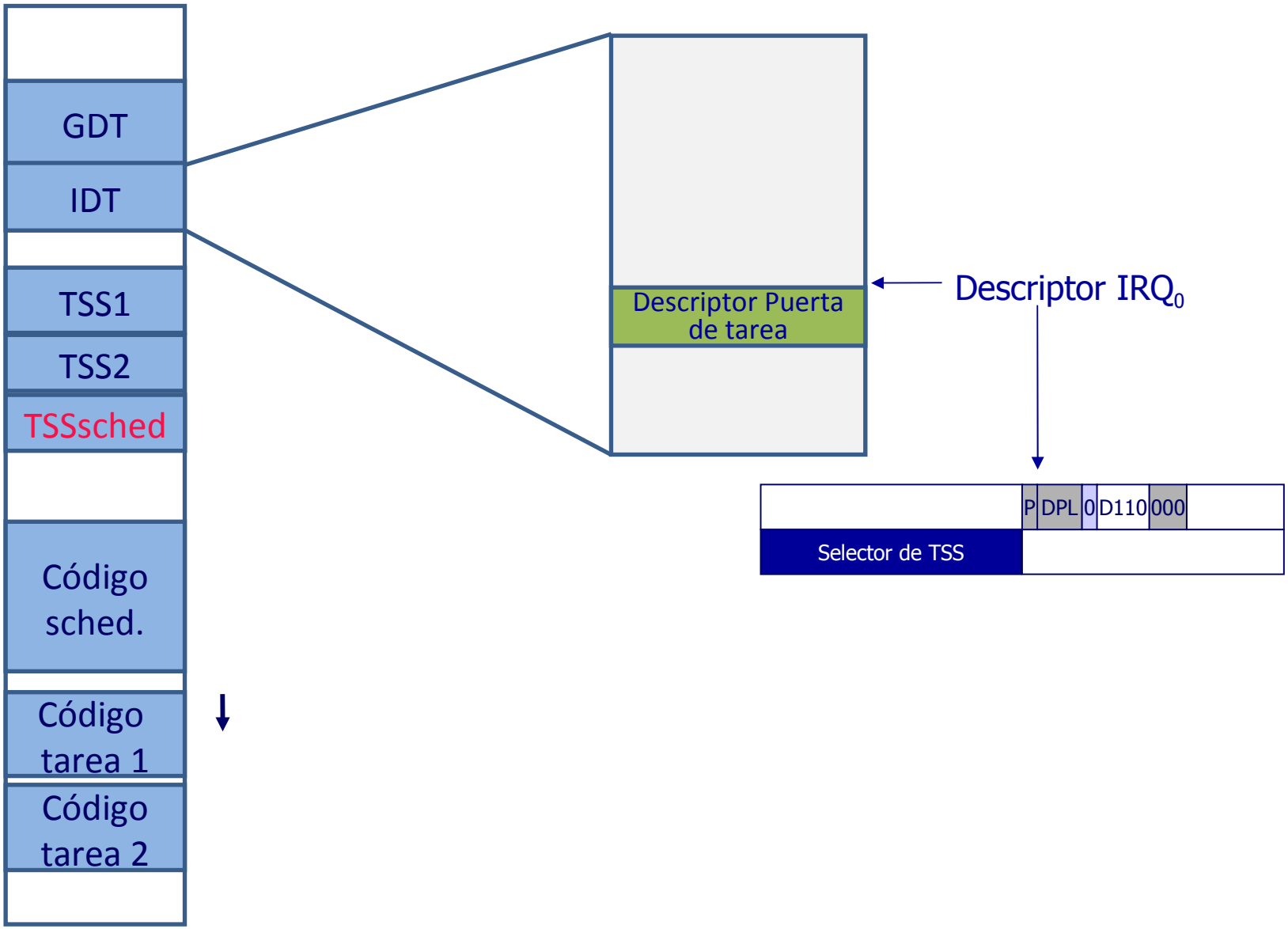
Puerta de Tarea. Conmutación.



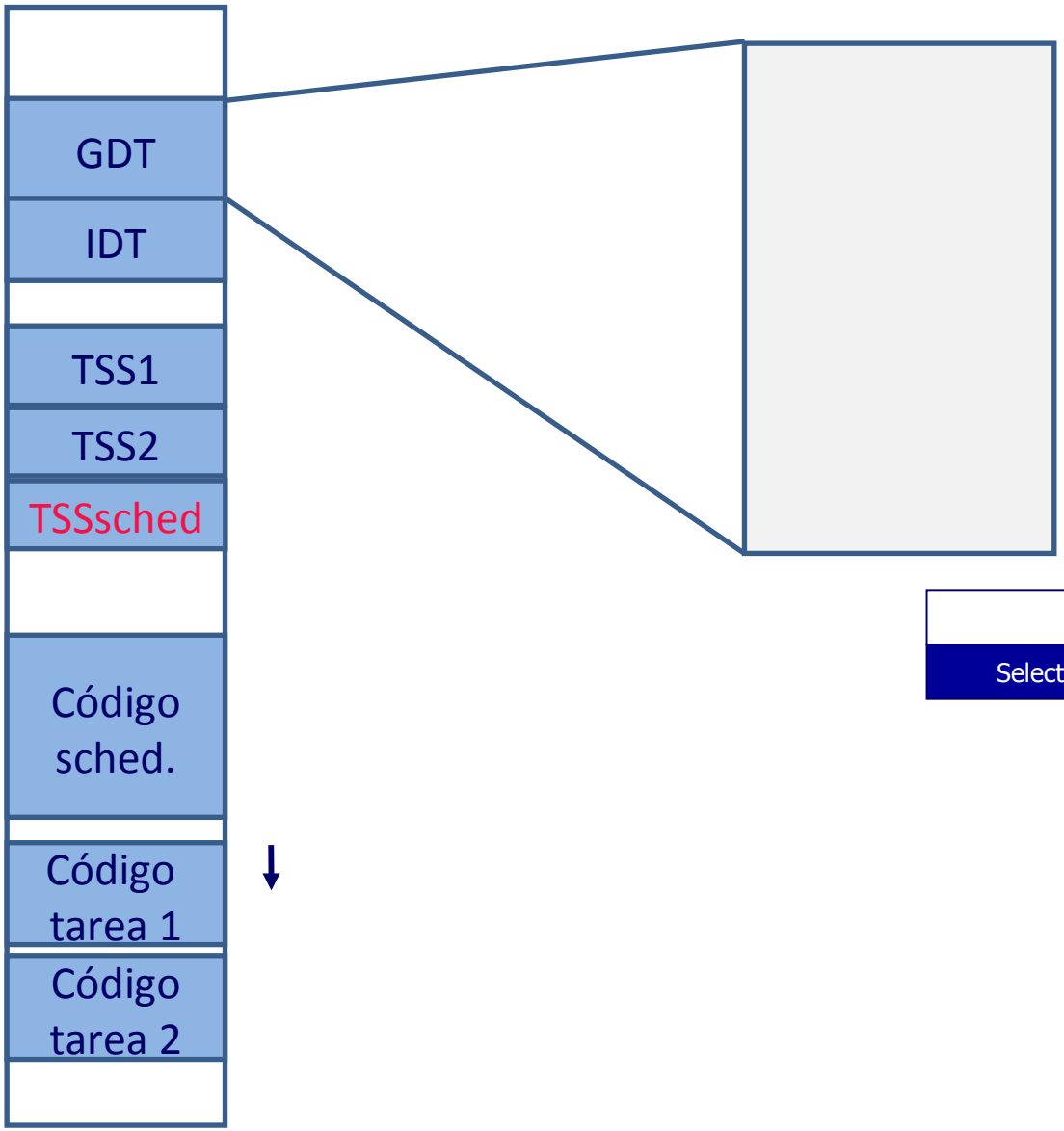
Puerta de Tarea. Conmutación.



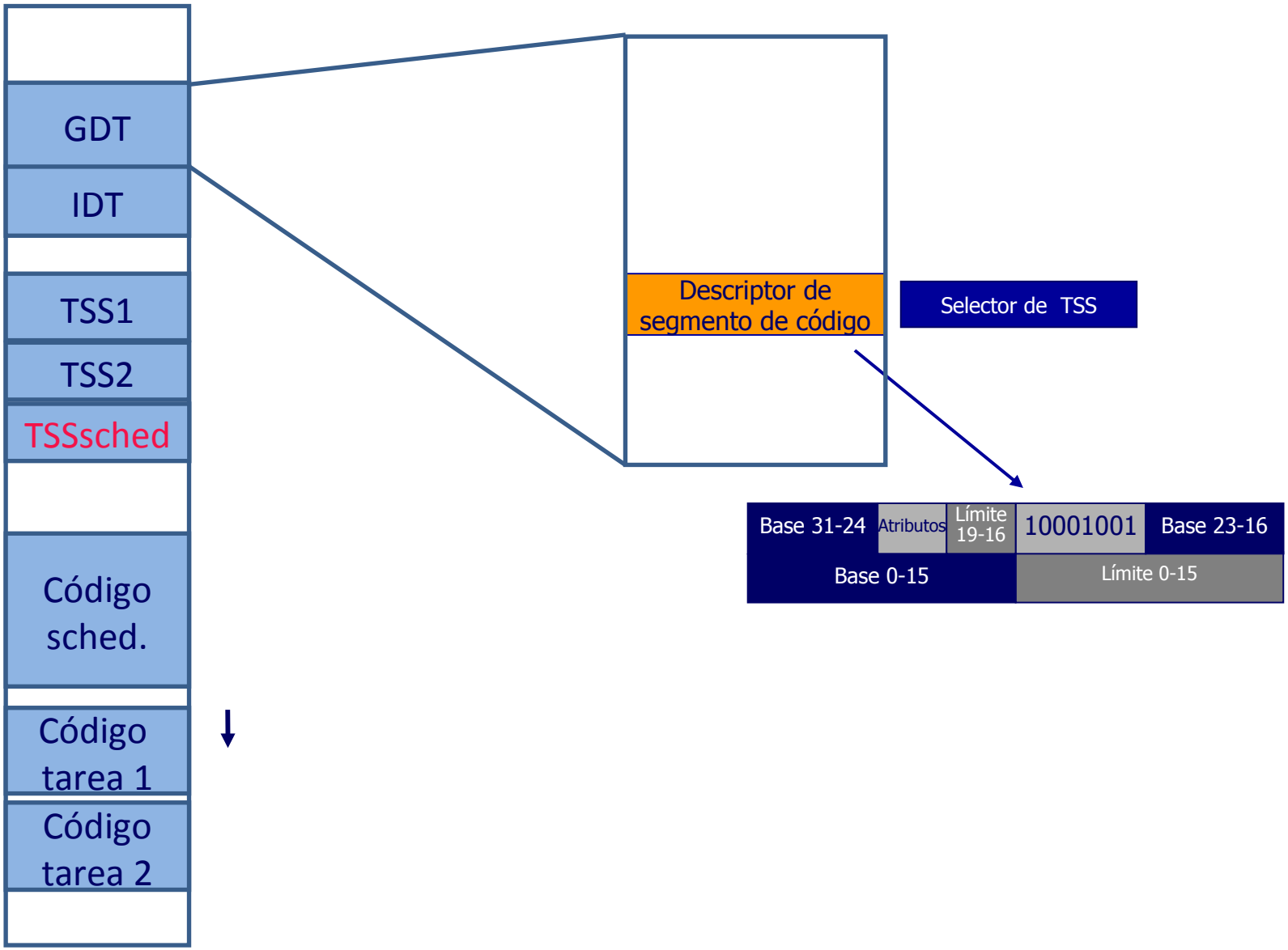
Puerta de Tarea. Conmutación.



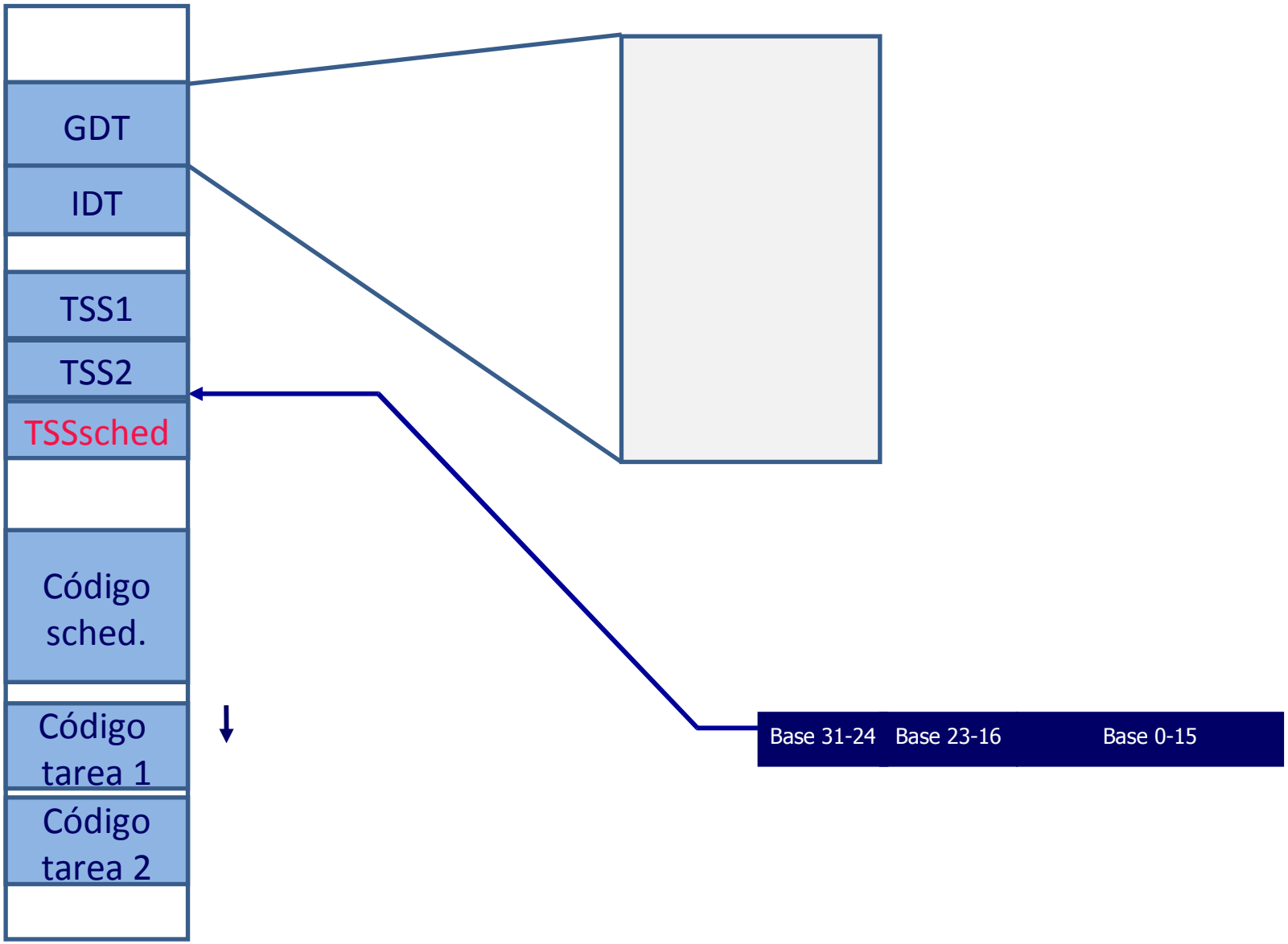
Puerta de Tarea. Conmutación.



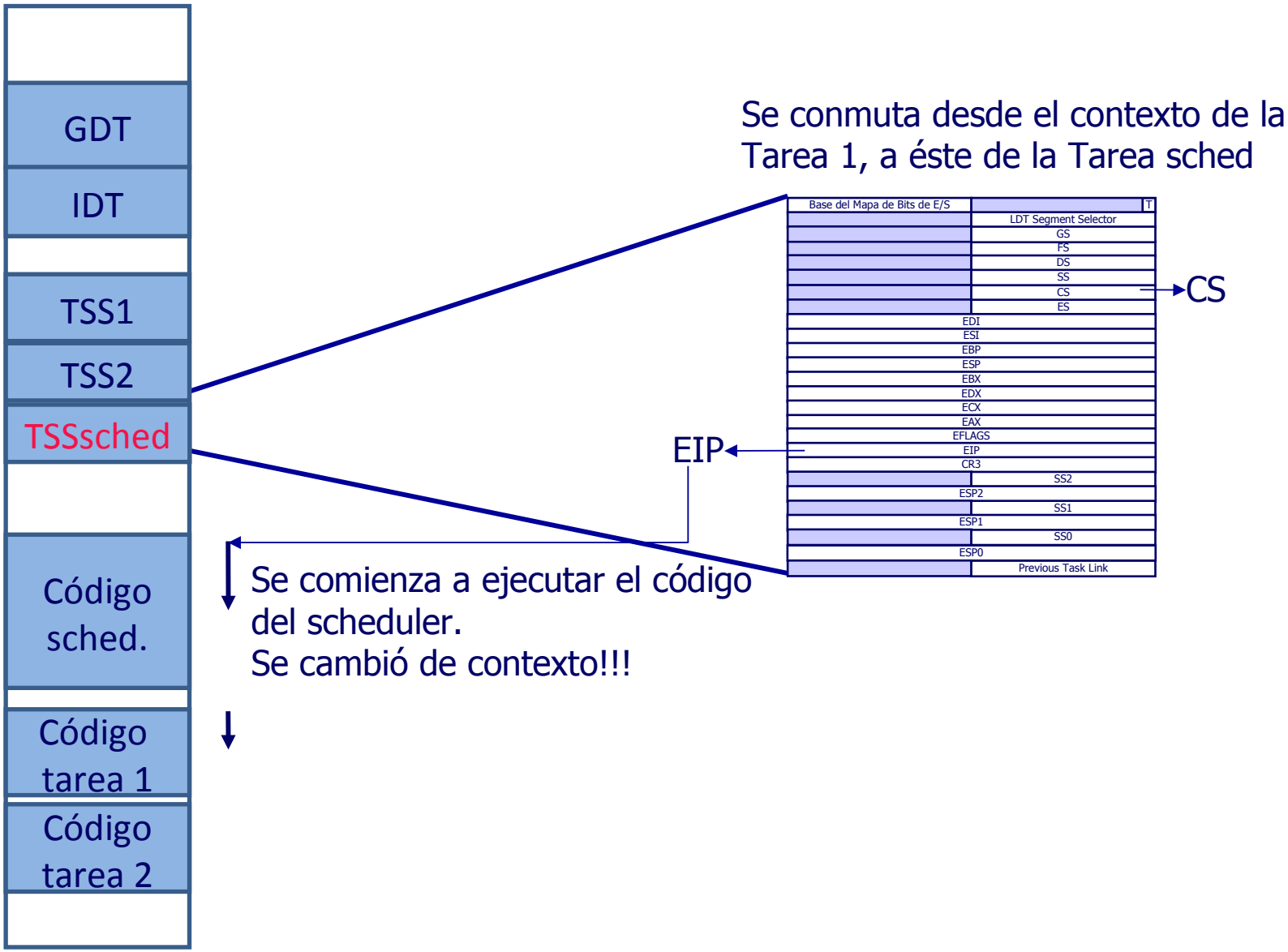
Puerta de Tarea. Conmutación.



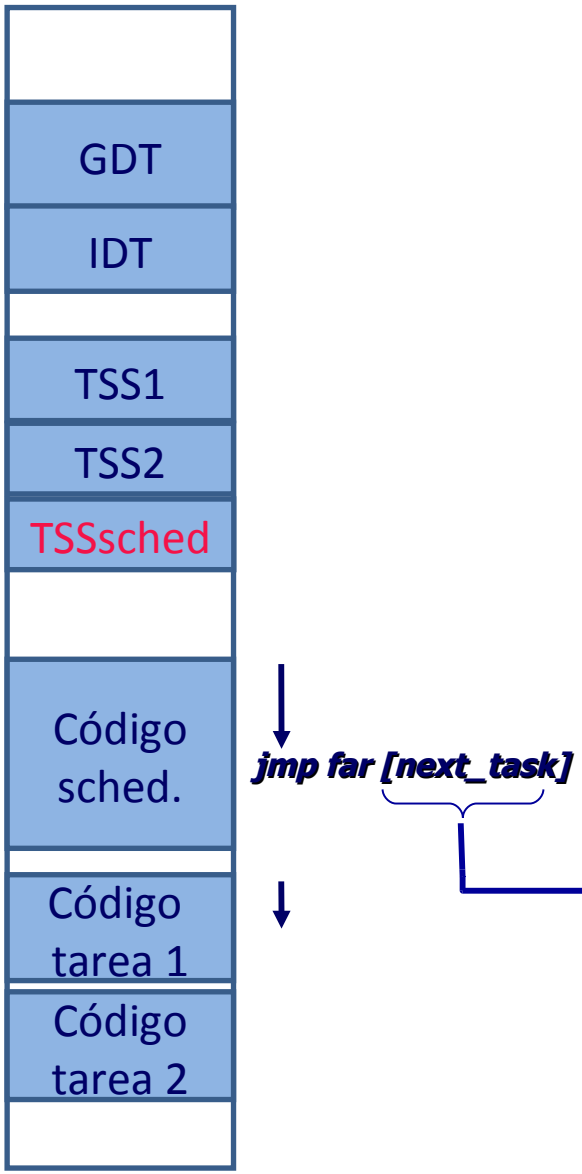
Puerta de Tarea. Conmutación.



Puerta de Tarea. Conmutación.

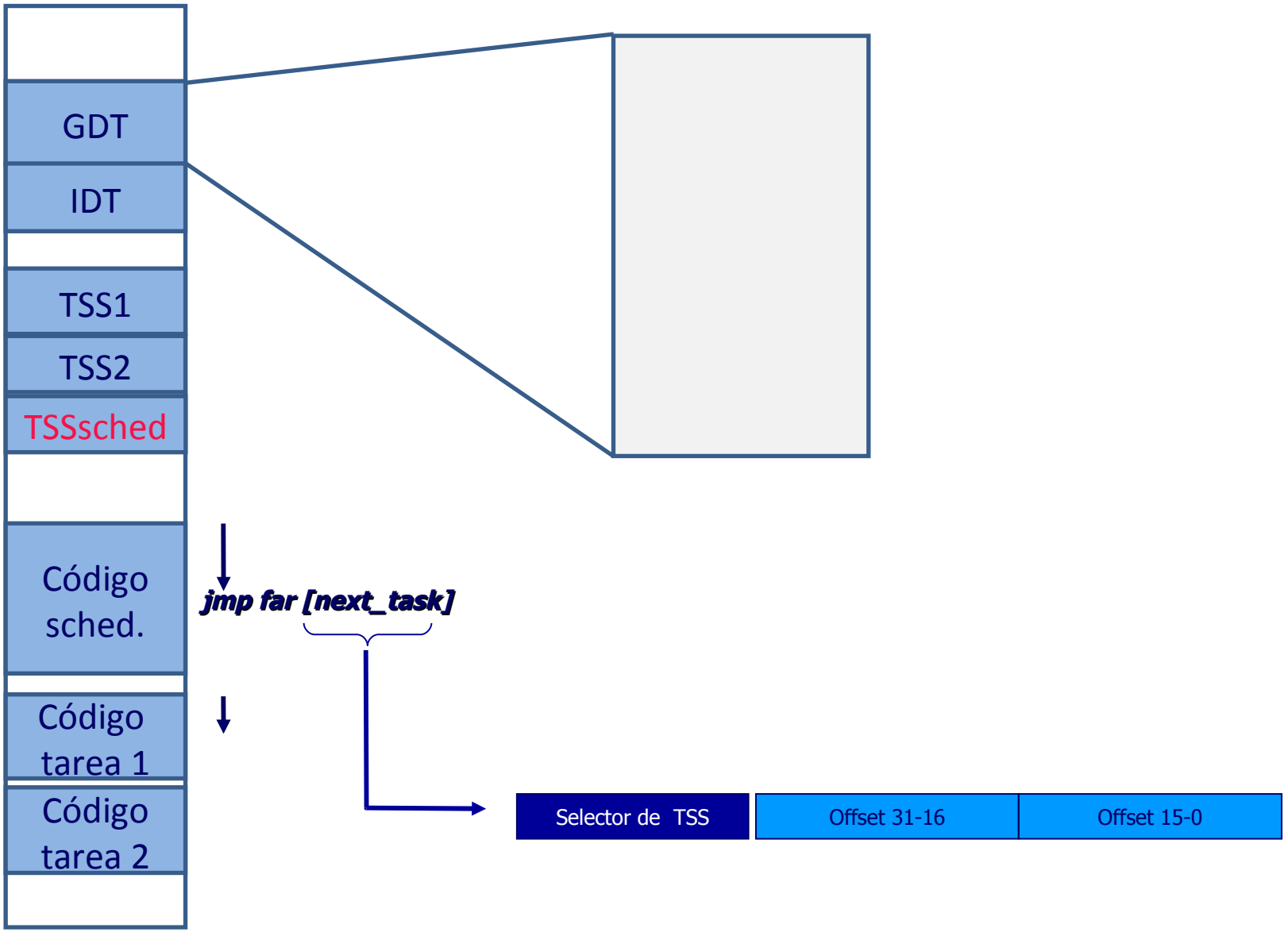


Puerta de Tarea. Conmutación.

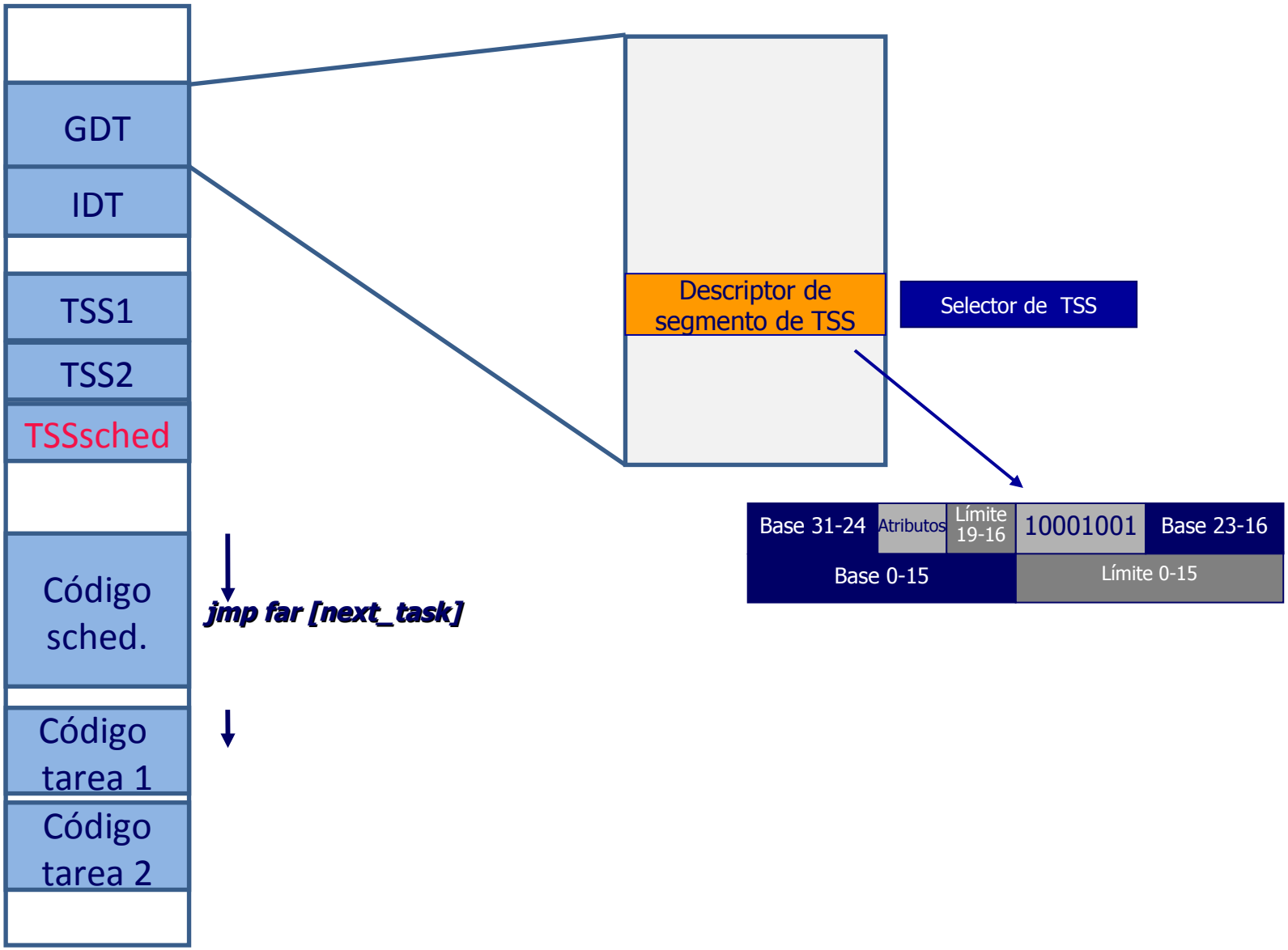


La variable ***next_task***, se compone de Selector de TSS y un offset (que finalmente será descartado por el procesador)

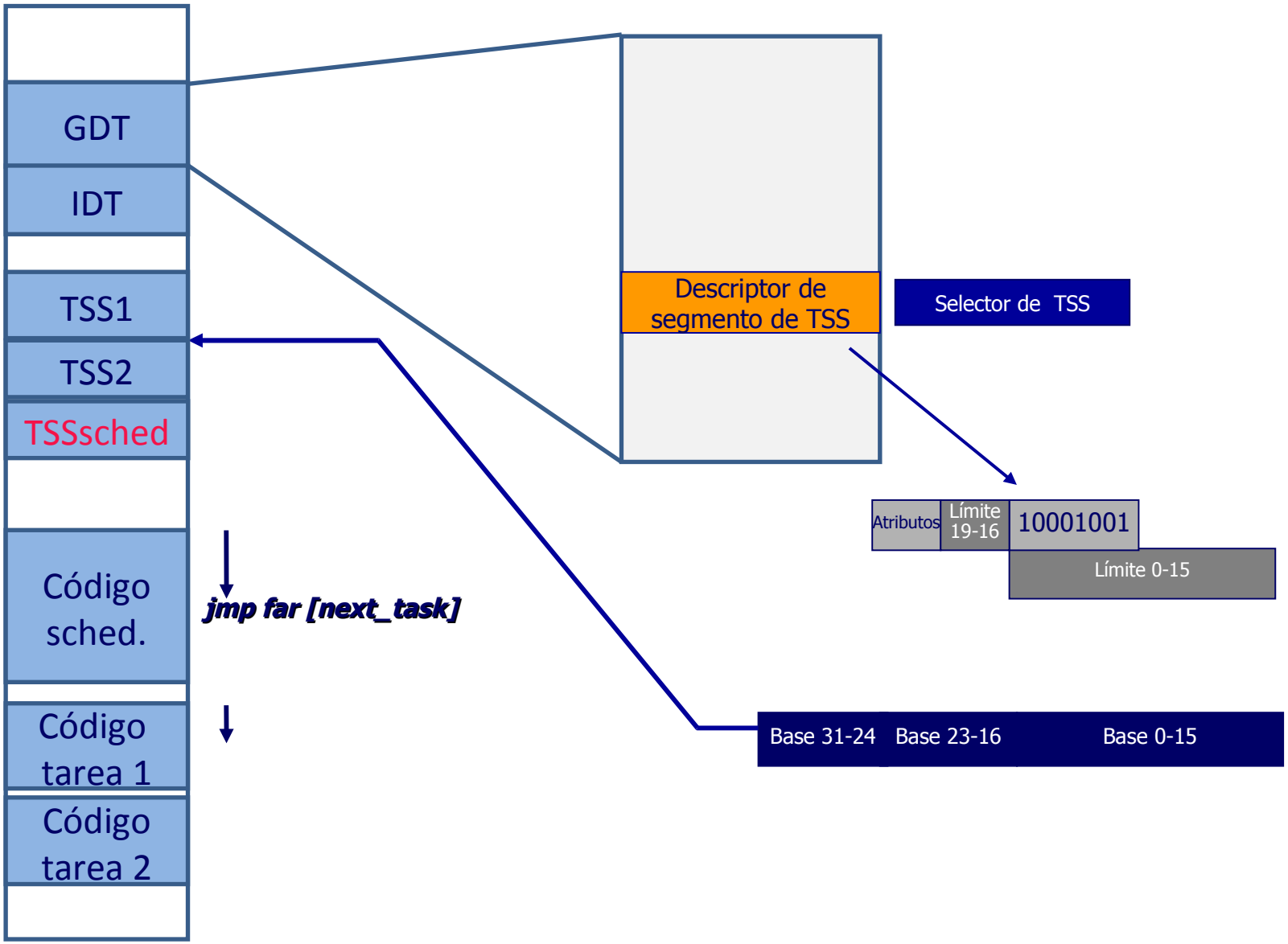
Puerta de Tarea. Conmutación.



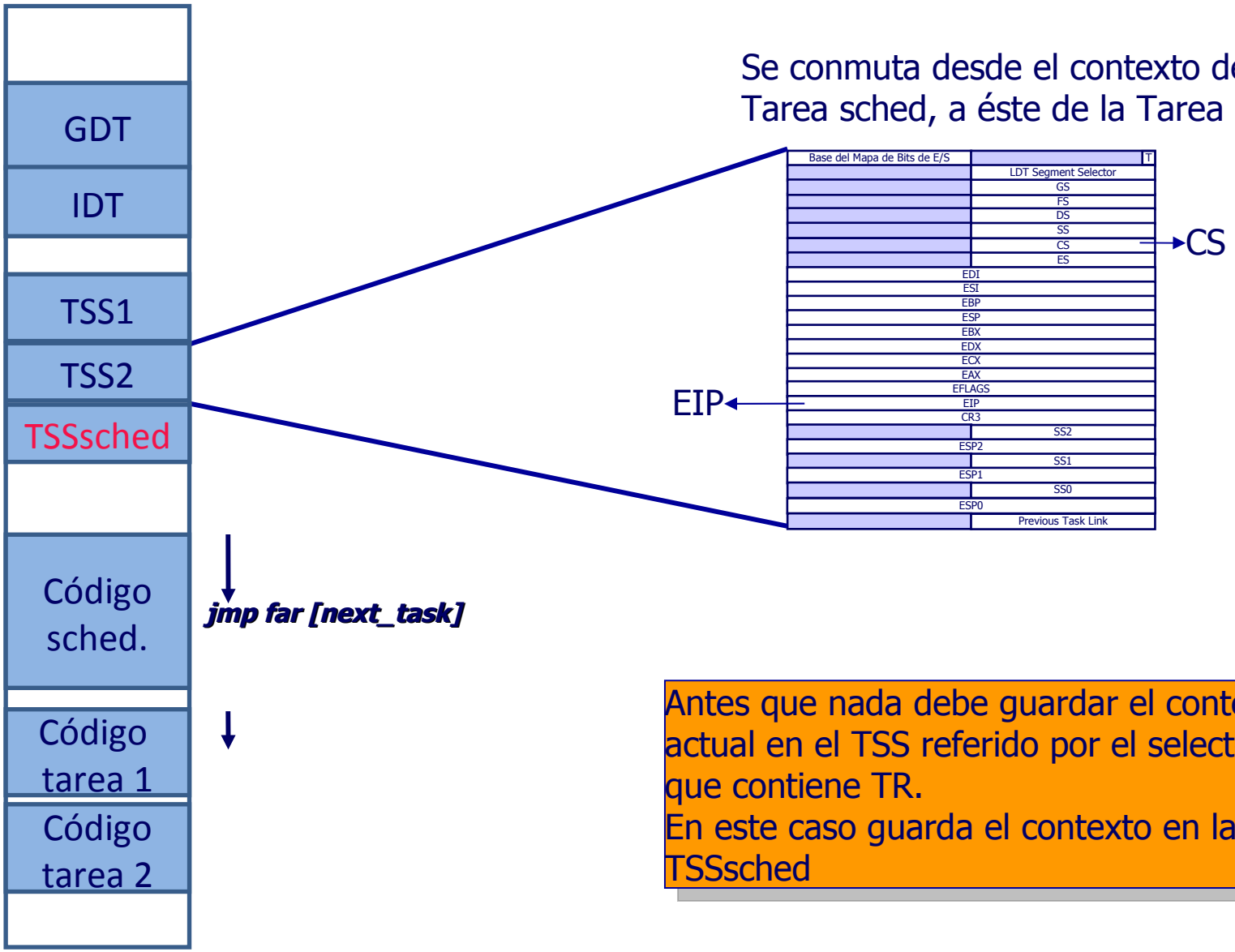
Puerta de Tarea. Conmutación.



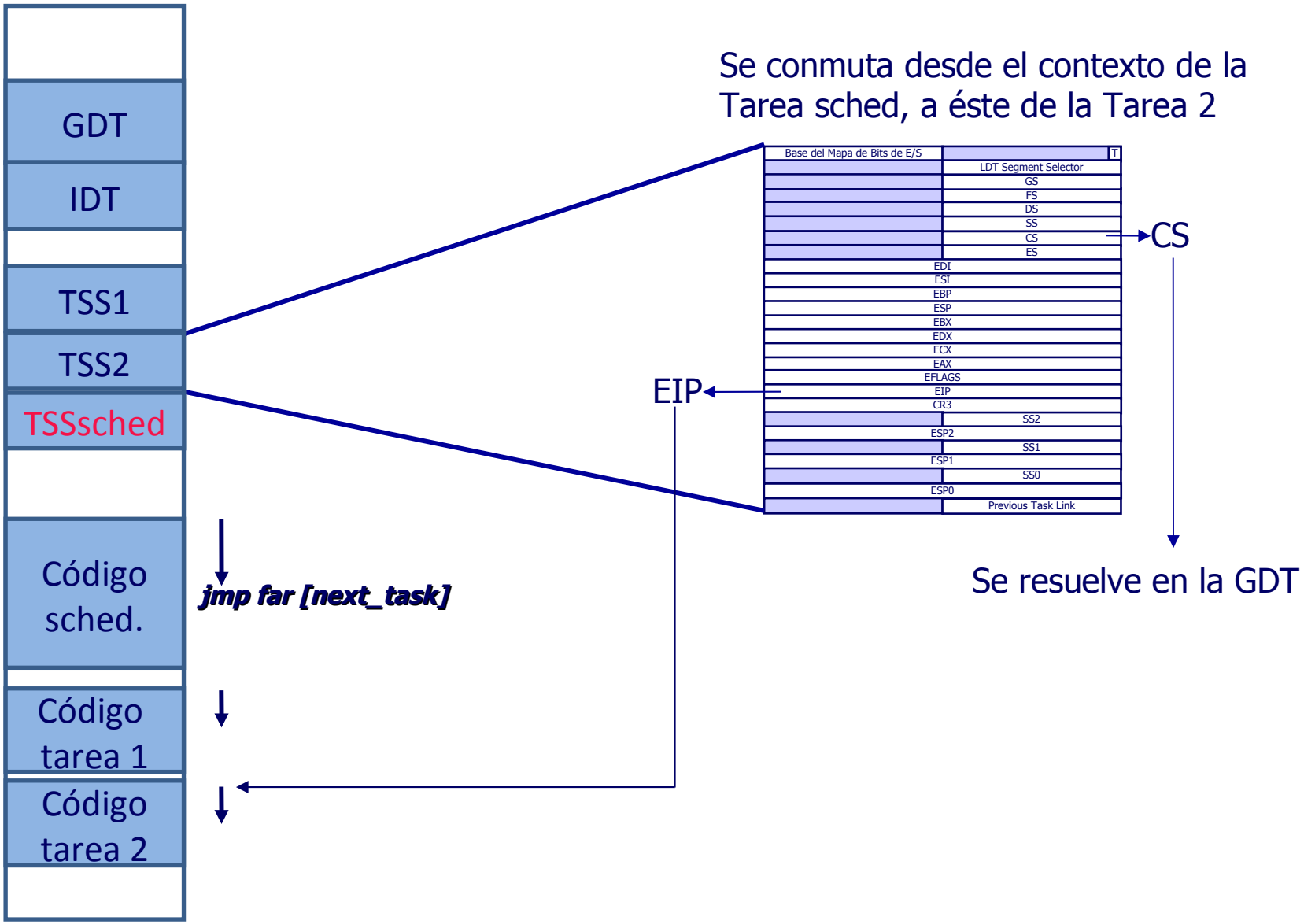
Puerta de Tarea. Conmutación.



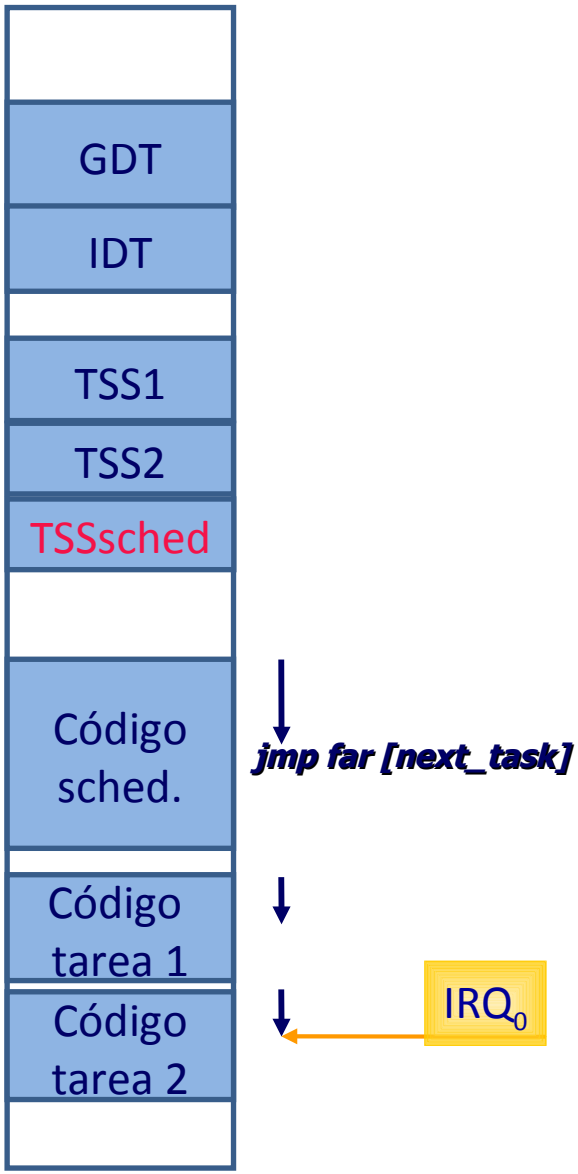
Puerta de Tarea. Conmutación.



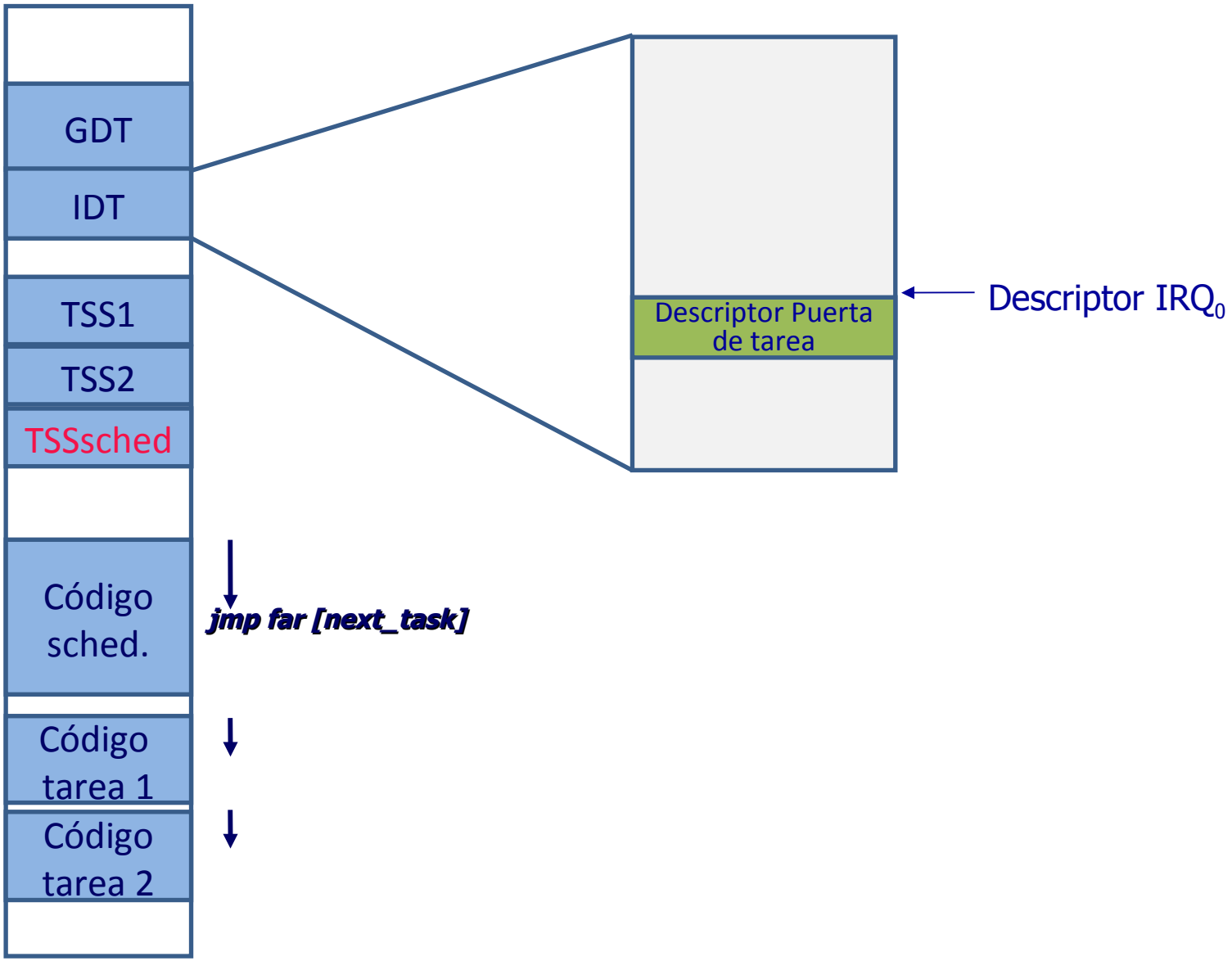
Puerta de Tarea. Conmutación.



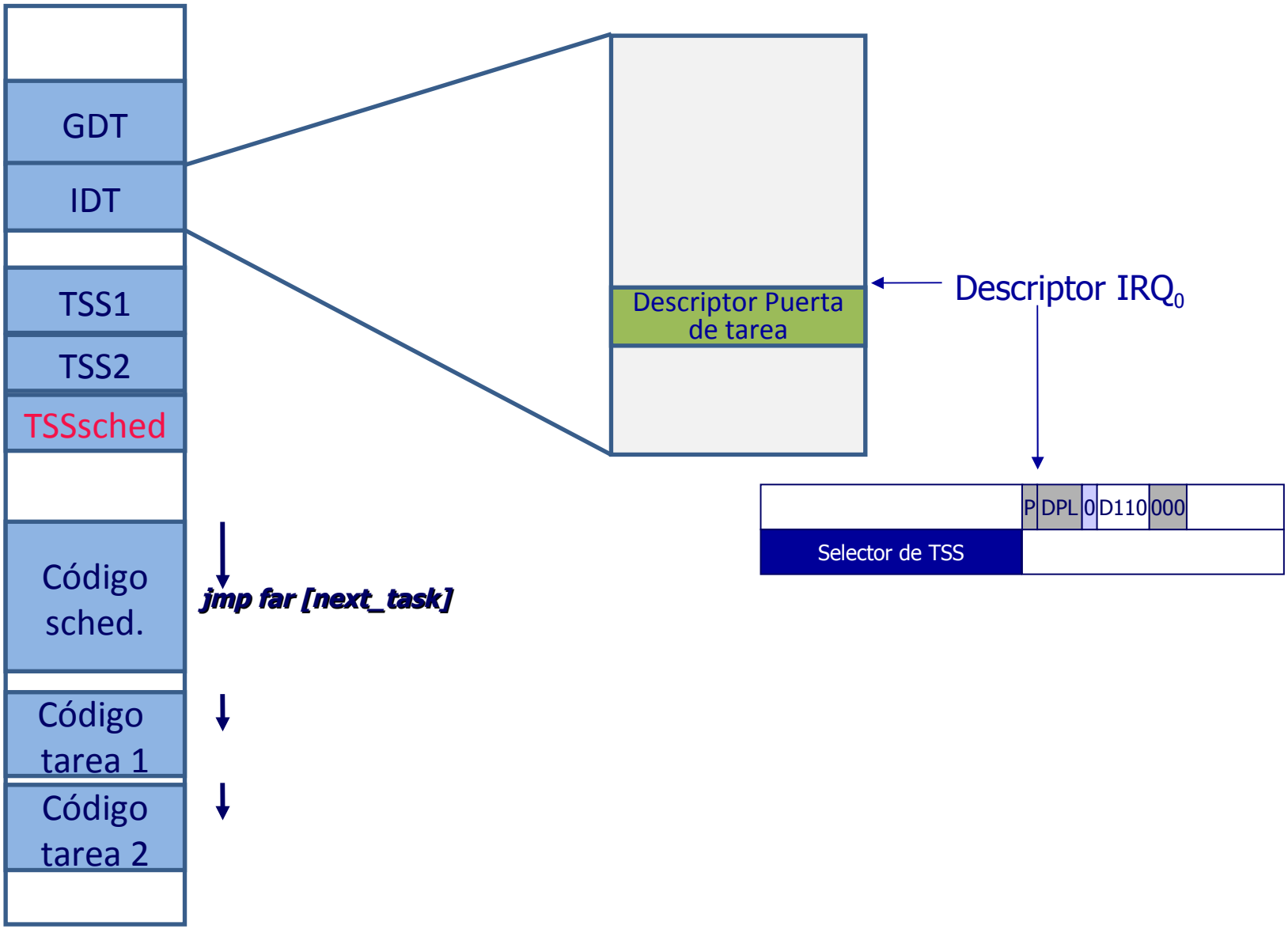
Puerta de Tarea. Conmutación.



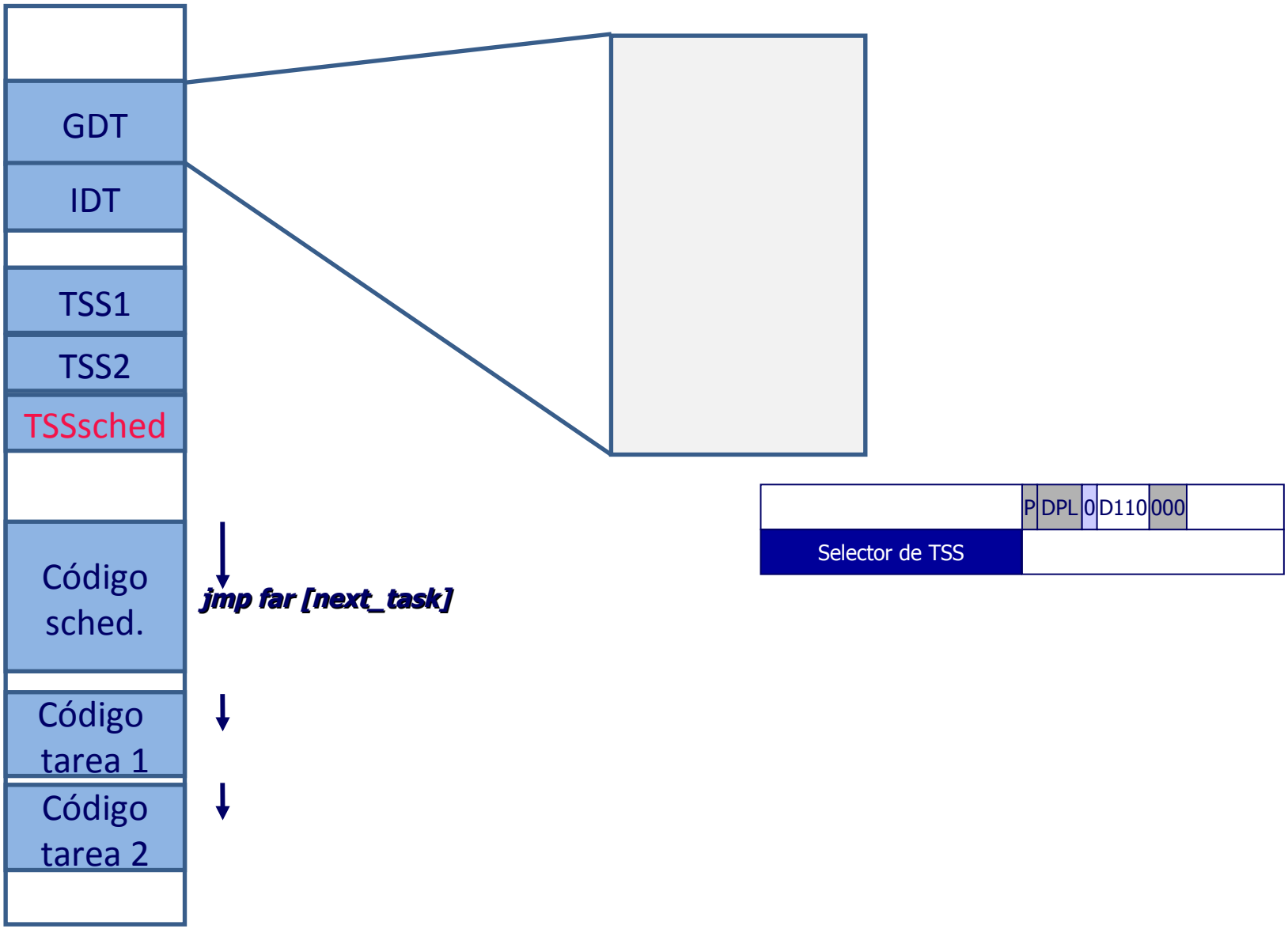
Puerta de Tarea. Conmutación.



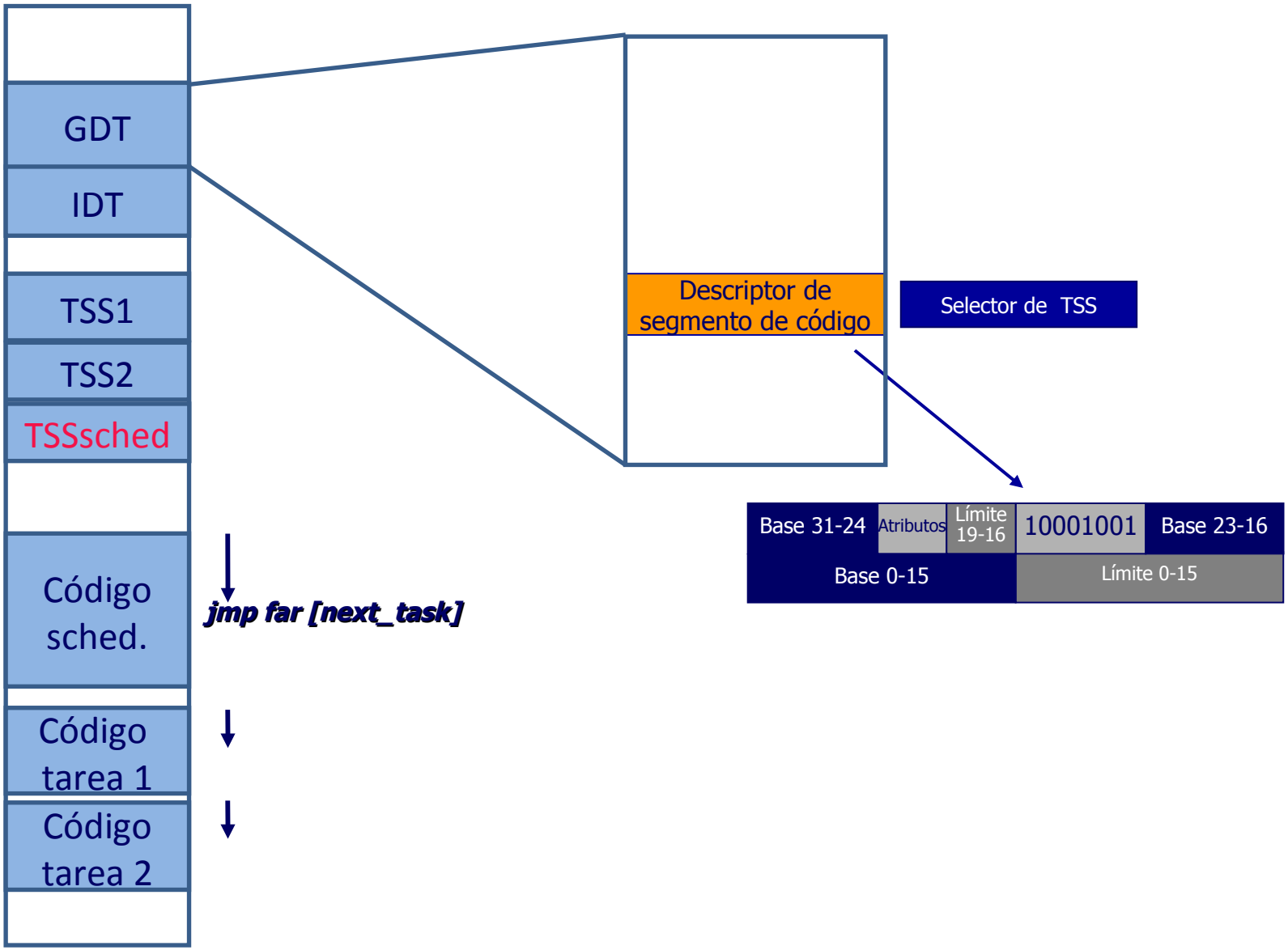
Puerta de Tarea. Conmutación.



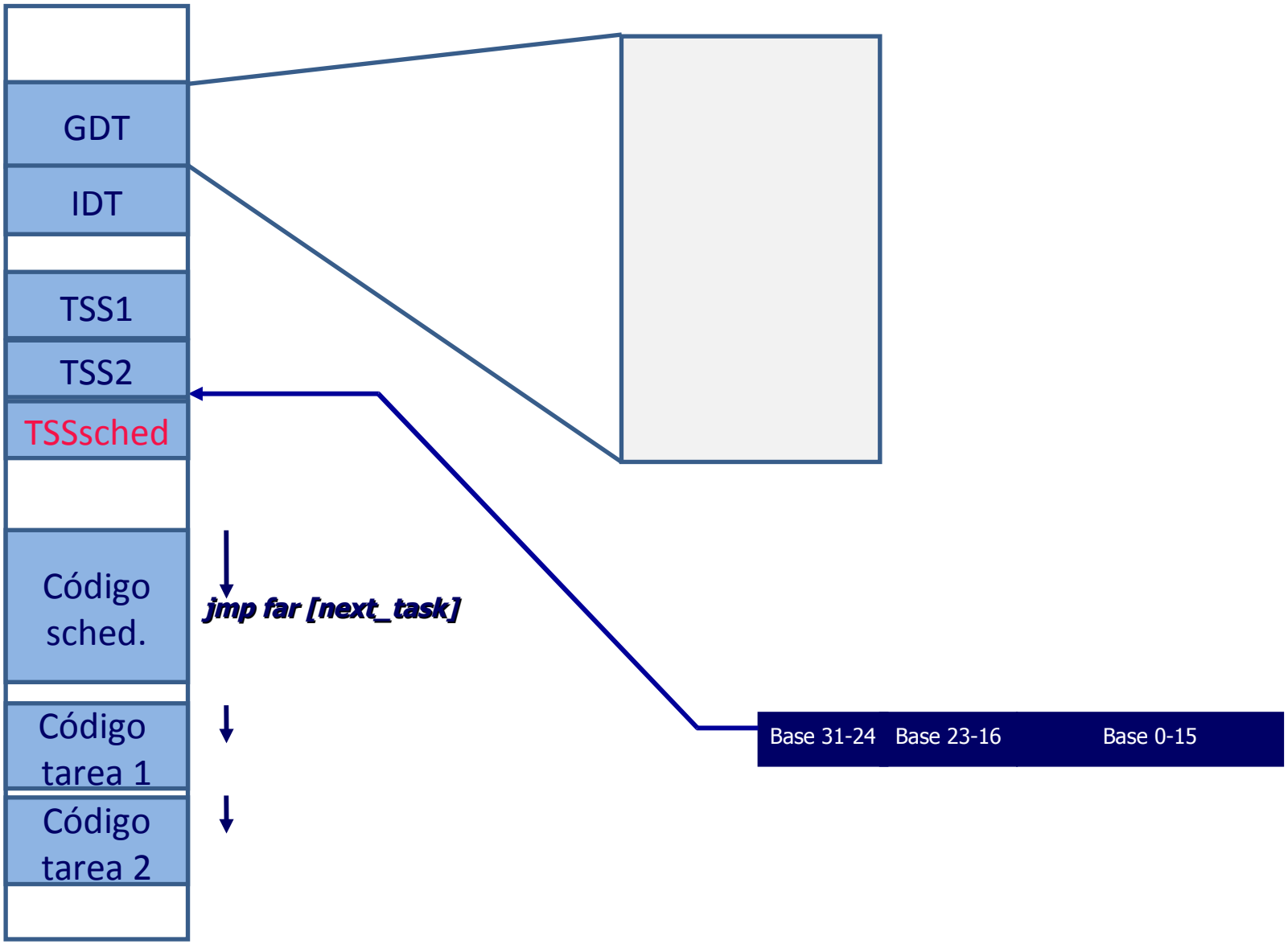
Puerta de Tarea. Conmutación.



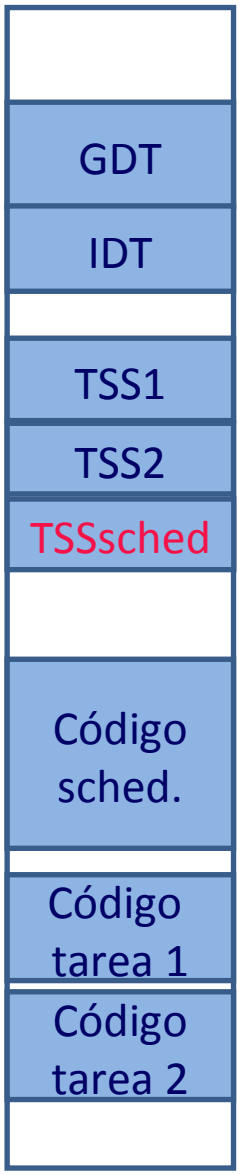
Puerta de Tarea. Conmutación.



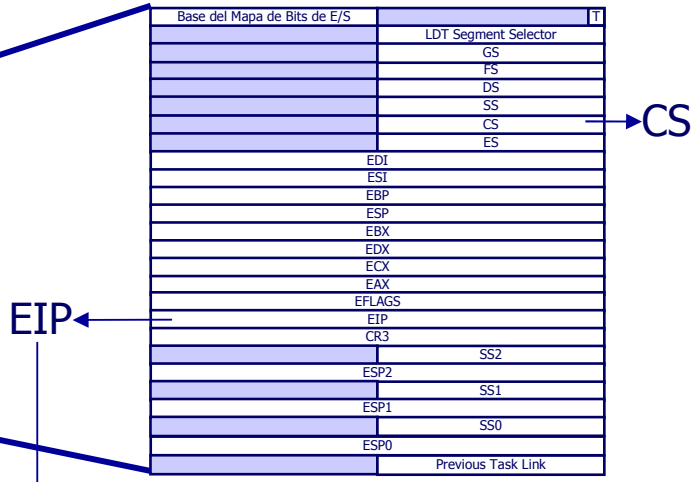
Puerta de Tarea. Conmutación.



Puerta de Tarea. Conmutación.



Se conmuta desde el contexto de la Tarea 2, a éste de la Tarea sched

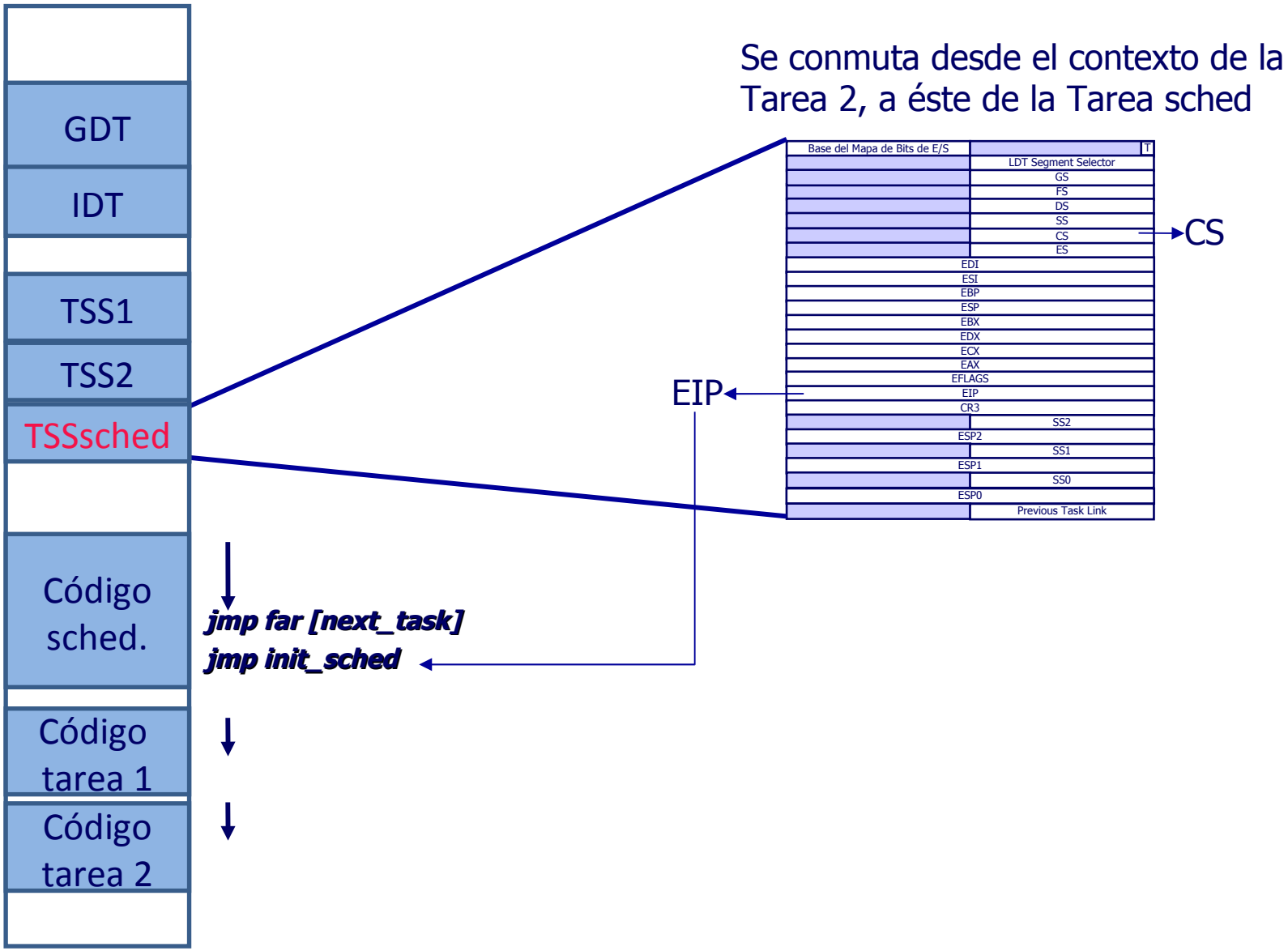


jmp far [next_task]

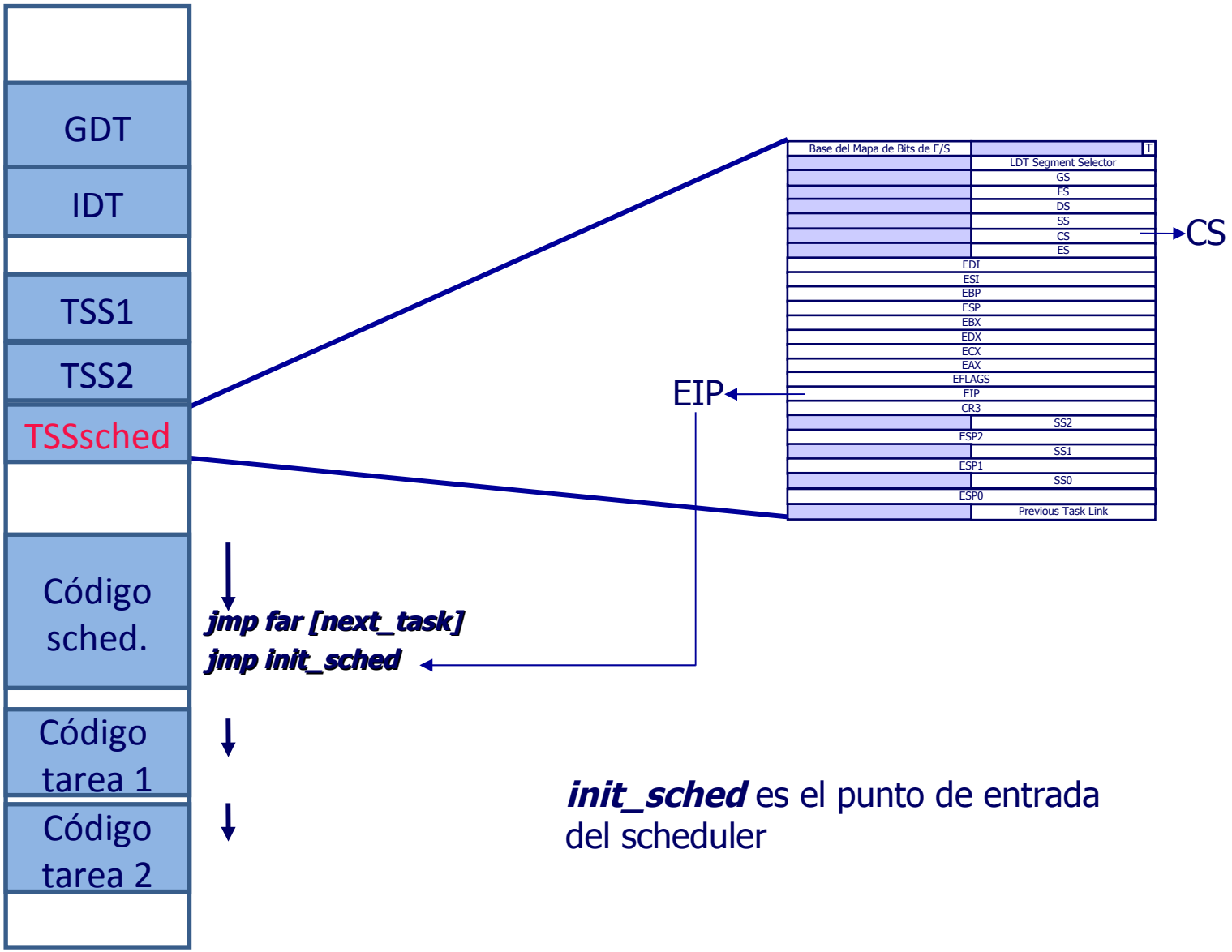
El EIP cuando se abandonó la tarea sched quedó con el valor correspondiente a la siguiente instrucción del cambio de tarea, que se ejecutó en el handler de IRQ_0.

Que instrucción sigue a la de cambio de tarea??

Puerta de Tarea. Conmutación.



Puerta de Tarea. Conmutación.



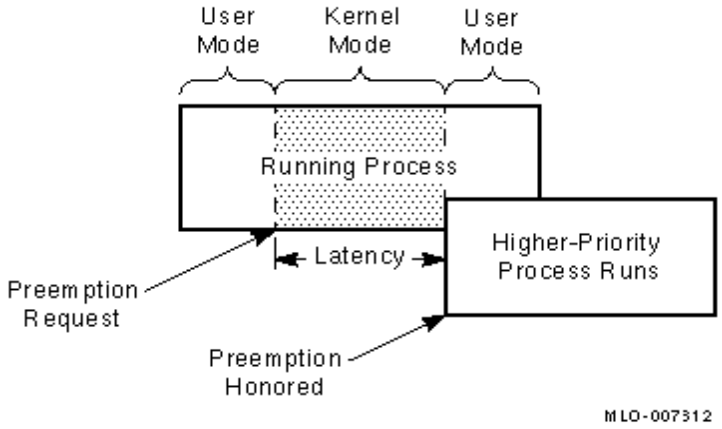
init_sched es el punto de entrada del scheduler

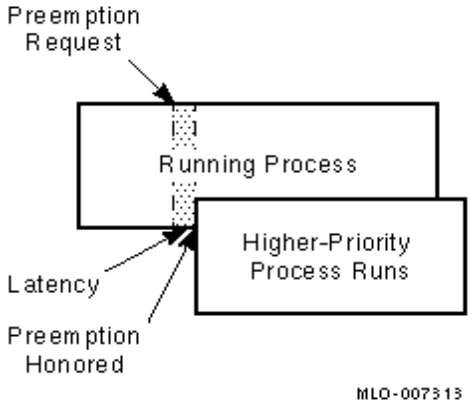
Linux

Manejo de Tareas

Process Preemption

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 7





- Preemption.
  - Se maneja con el campo need_resched del descriptor de proceso
- Los procesos en Linux 2.6. son interrumpibles (preempted).
  - Cuando un proceso entra en estado ***TASK_RUNNING***, el kernel chequea su prioridad y la compara con el corriente. Si su prioridad es mayor el proceso actual es interrumpido (preempted).
  - Cuando a causa de una interrupción se debe despertar a un proceso (ponerlo ***TASK_RUNNING***) y su prioridad es mayor que la del proceso en curso, el kernel setea el bit ***current->need_resched*** y por lo tanto antes de salir de la interrupción se llamará a la función ***schedule()*** para pasar al nuevo proceso.

Conmutación de Procesos

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 3

- Linux mantiene una única TSS por cada CPU.
- A pesar del no uso de jmp far para conmutar no puede prescindir de al menos un TSS por cada CPU:
  - Al aumentar el nivel de privilegio de una tarea, el procesador busca en el TSS actual (cuyo selector contiene el registro TR), los valores de SS y ESP correspondientes al mayor nivel de privilegio.
  - Para acceder a E/S desde una tarea cuyo CPL no sea el adecuado debe consultarse el IO BitMap del TSS de la tarea.
- Al usar un único TSS por CPU, Linux debe actualizarle determinados campos en cada process switch.

Conmutación de Procesos

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 3

- El contexto de hardware de cada tarea se almacena en una estructura del tipo ***thread*** en ***task_struct***.
- Esta estructura puede ser tomada por cualquier CPU presente en el sistema ya que contiene los datos que de guardarse en la TSS no podrían ser tomados por otra CPU.
- Observaciones:
  - Solo guarda los registros de propósito general estrictamente necesarios. El resto va al stack
  - Se almacena el estado de la FPU, y debug registers que no son contemplados en el TSS
  - Se mantiene el bitmap de E/S del proceso en esta estructura

macro switch_to()

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 3

- Se invoca desde ***schedule ()*** mediante la línea

  switch_to (prev, next, prev)

- Se ejecuta el siguiente código

```
movl   prev,%eax   ;eax = prev
movl   next,%edx   ;edx = next
movl   %eax,%ebx   ;ebx = eax = prev
pushfl          ;guarda flags y ebp en el stack de nodo kernel
push    %ebp       ;de prev
;la estructura thread ocupa el offset 484 dentro de task_struct
movl   %esp, 484(%eax)    ; salva esp en prev->thread.esp
movl   484(%edx), %esp    ; carga esp con next->thread.esp
movl   $1f, 480(%eax) ; guarda ret addr en  prev->thread.eip
pushl  480(%edx)  ; guarda ret addr en  next->thread.eip
jmp__switch_to
```

rutina __ switch_to

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 3

- Ejecuta los siguientes pasos:
  - Si cambiaron, salva los registros de la FP, MMX, y XMM.

```
unlazy_fpu (prev_p) ;
```

  - Macro que devuelve el índice a la CPU que está ejecutando el proceso (lo saca de thread_info->cpu).

```
smp_processor_id ( ) ;
```

  - Carga stack de nivel 0 de netx_p en la única TSS de la cpu……. mmmm.

```
Init_tss [cpu ].esp0 = next_p ->thread.esp0 ;
```

  - Carga en la GDT los segmentos TLS de netx.

```
cpu_gdt_table[cpu][6] = next_p->thread.tls_array[0];
cpu_gdt_table[cpu][7] = next_p->thread.tls_array[1];
cpu_gdt_table[cpu][8] = next_p->thread.tls_array[2];
```

  - Salva en la estructura thread los registros FS y GS (El kernel no los usa, pero las aplicaciones pueden usarlos.

```
movl %fs, 40 (%esi) ;esi apunta a prev_p -> thread
movl %gs,44 (%esi)
```

  - Carga los nuevos valores de fs y gs.

```
movl 40 (%ebx),%fs ;ebx apunta a next_p -> thread
movl 44 (%ebx),%gs
```

rutina __ switch_to

Ref: Understanding the Linux Kernel 3erd. Ed. D. Bovet. Cap 3

- Salva los valores de los debug registers, si el proceso al que se va a activar los había utilizado.

```
if (next_p->thread.debugreg[7]){
loaddebug(&next_p->thread, 0);
loaddebug(&next_p->thread, 1);
loaddebug(&next_p->thread, 2);
loaddebug(&next_p->thread, 3);
/* no 4 and 5 */
loaddebug(&next_p->thread, 6);
loaddebug(&next_p->thread, 7);
}
```

- Salva en el TSS de la CPU que corresponda los valores de IO BitMap, si prev_p o next_p tienen permisos customizados de E/S.

```
if (prev_p->thread.io_bitmap_ptr || next_p->thread.io_bitmap_ptr)
  handle_io_bitmap(&next_p->thread, &init_tss[cpu]);
```

- Retorna

```
return prev_p;
// El compilador genera este código:
//       movl %edi,%eax
//       ret
```