Memoria Simple

Semestre II-2013

Manejo de Memoria Simple

- Motivación
- Swapping (Intercambio)
- Asignación de Memoria Contigua
- Segmentación
- Paginación
- Estructuras de las Tablas de Páginas
- Ejemplos de Memoria
  - Arquitectura Intel de 32 y 64 bits

Objetivos

- Proveer una descripción detalladas de las diferentes opciones de organización de la memoria a nivel de hardware
- Discutir diversas técnicas de manejo de memoria
  - Segmentación y Paginación

Motivación

- Los programas deben ser cargados en memoria (desde disco) con la intención de aplicar la abstracción de proceso y así poder ejecutarlos
- El CPU solo puede acceder de forma directa a los datos almacenados en los registros o en la memoria principal
- La unidad de memoria solo ve:
  - Conjunto de direcciones + solicitudes de lectura
  - Conjunto de direcciones + datos + solicitudes de escritura

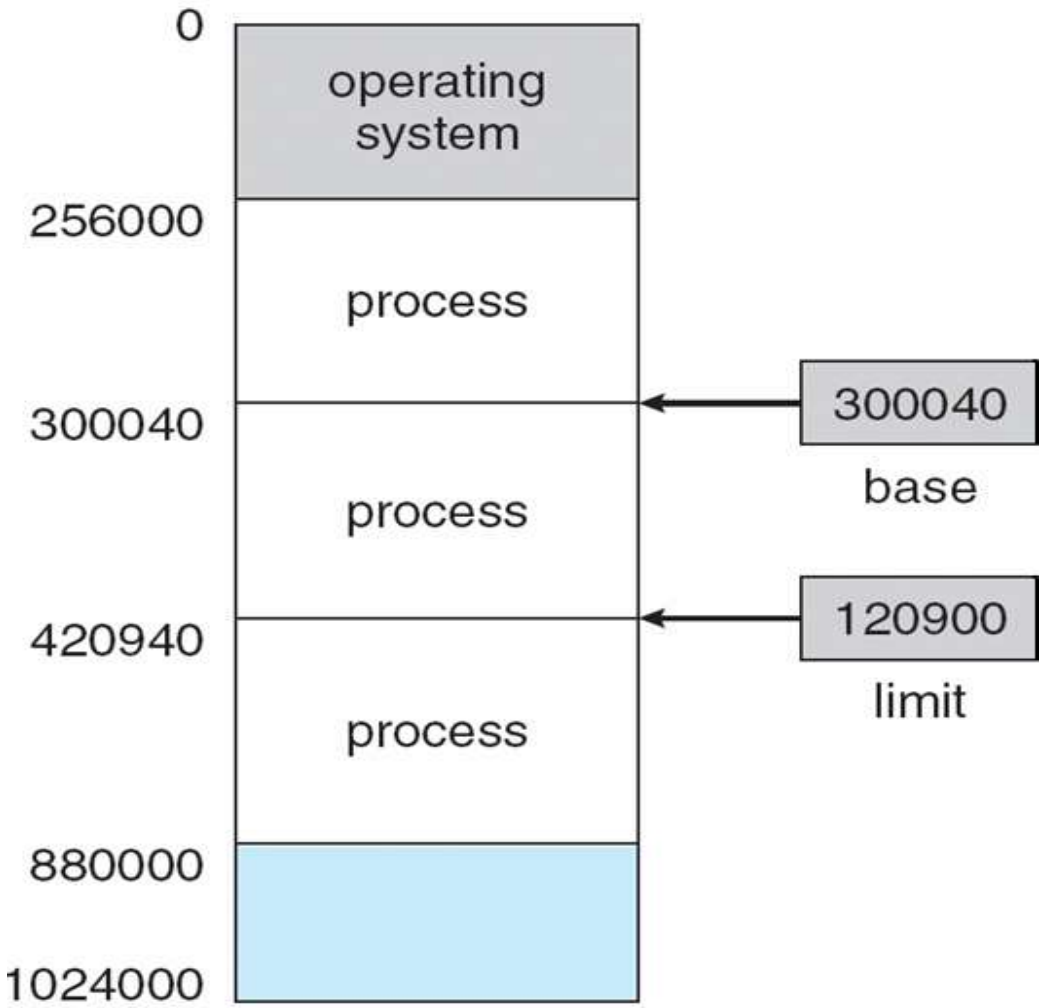
Motivación

- Los registros se acceden en un ciclo de reloj o menos
- El acceso a la memoria principal puede tomar varios ciclos de reloj, causando retraso o demora
- Una memoria cache se sitúa entre la memoria principal y los registros de CPU
- Es necesario brindar mecanismos de protección de memoria
  - Asegurar un funcionamiento correcto

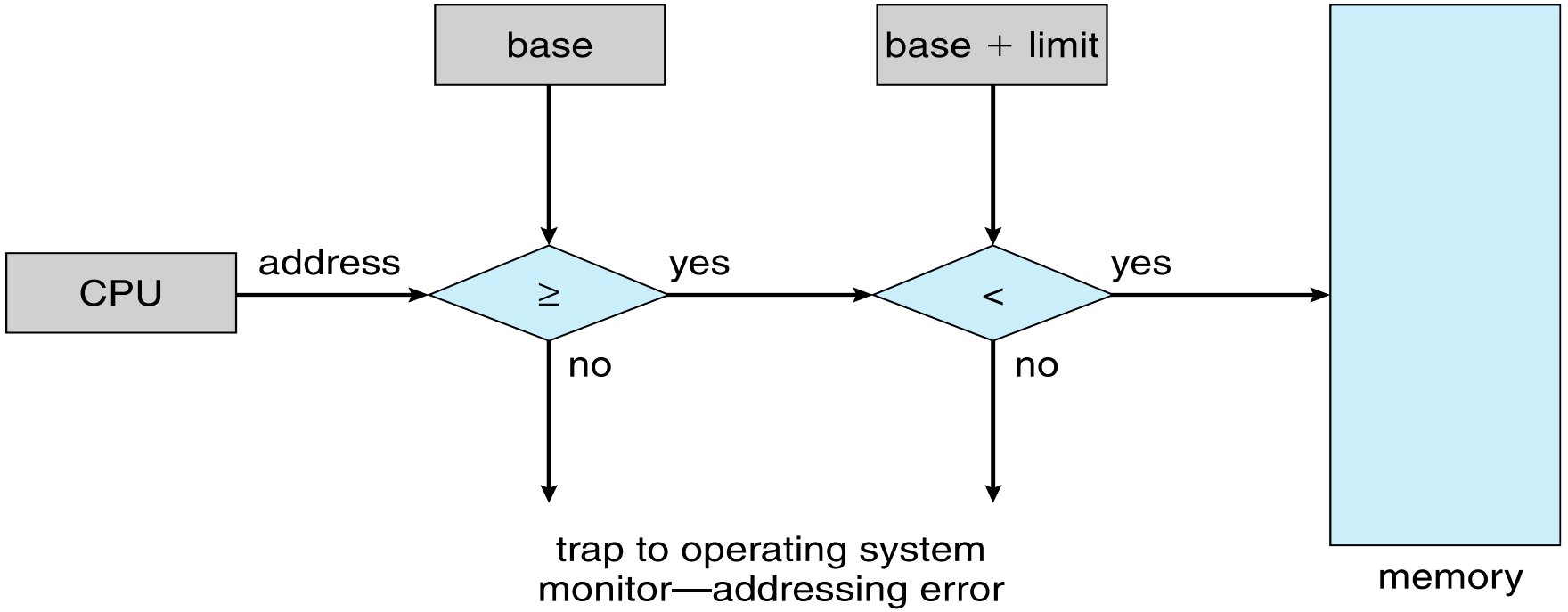
Registro Base y Limite

- Un par de registros (base y limite) son definidos dentro del espacio de direcciones lógicas
- El CPU debe verificar cada acceso a memoria generado en modo usuario para garantizar que la solicitud se encuentra entre ambos registros

Registro Base y Limite



Protección de Direcciones (Hardware)



Mapeo de Direcciones

- Considere programas en disco, los cuales están listos para ser traídos a memoria principal con la intención de ejecutarlos estilo cola
  - Si no existe ningún soporte, estos deben ser cargados a partir de la dirección 0000
- ¿Existe algún inconveniente en que siempre la primera dirección de un proceso de usuario sea 000?
  - Ideas

Mapeo de Direcciones

- Adicionalmente, las direcciones representan en diferentes formas los estados del ciclo de vida del programa
  - Direcciones de código fuente usualmente son simbólicas
  - Las direcciones de un código compilado se mapean a espacios de memoria relocalizable
    - Por ejemplo, "14 bytes desde el comienzo de este modulo"
  - Los enlazadores o cargadores mapean estas direcciones relocalizables a direcciones absolutas
    - Por ejemplo, 74014

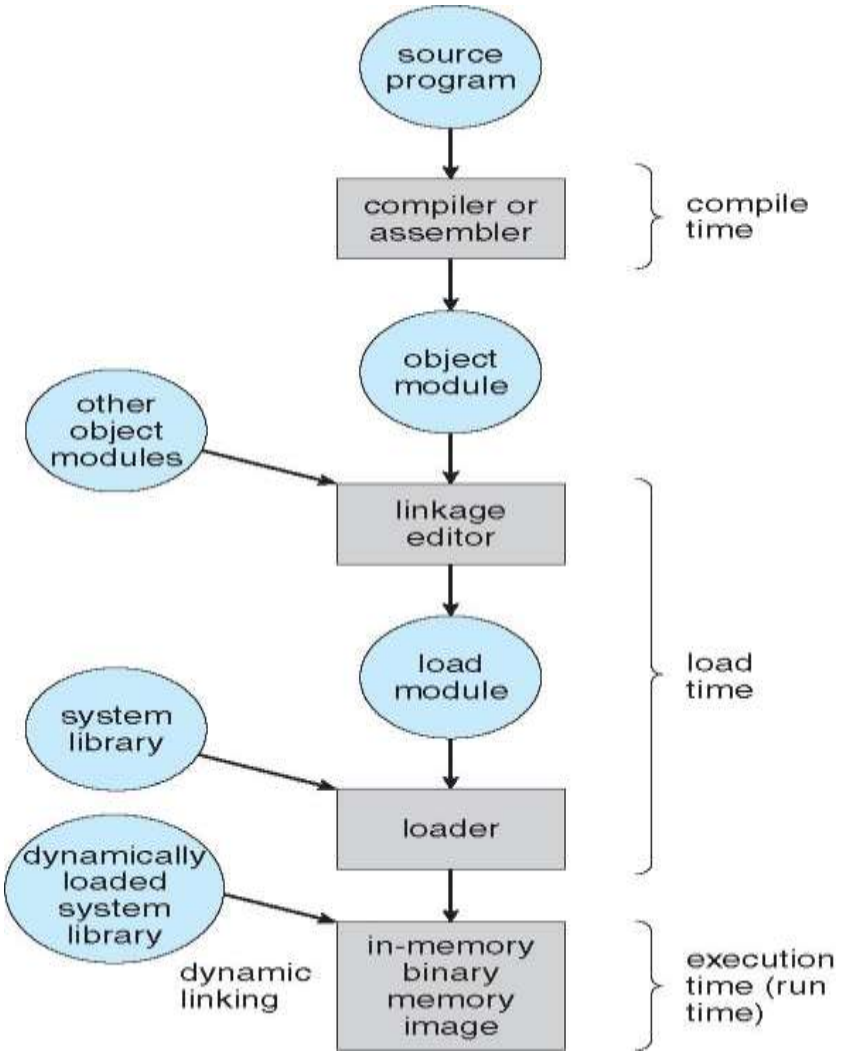
Mapeando Instrucciones a Datos en Memoria

- En el mapeo de una instrucción a un dato en memoria, tres estados diferentes podrían presentarse:
  - Tiempo de compilación
  - Tiempo de carga
  - Tiempo de Ejecución

Mapeando Instrucciones a Datos en Memoria

- Tiempo de Compilación: Si la localización de memoria se conoce a priori, código absoluto puede ser generado. Este tipo de código requiere una recopilación si la dirección de inicio cambia
- Tiempo de Carga: Es necesario la generación de código relocalizable si la posición de memoria no es conocida en tiempo de compilación
- Tiempo de Ejecución: El mapeo puede ser demorado hasta la ejecución, si el proceso puede ser movido durante su ejecución, pro ejemplo de un segmento de memoria a otro
  - Es necesario el soporte de hardware para realizar el mapeo de direcciones

Procesamiento de un Programa de Usuario



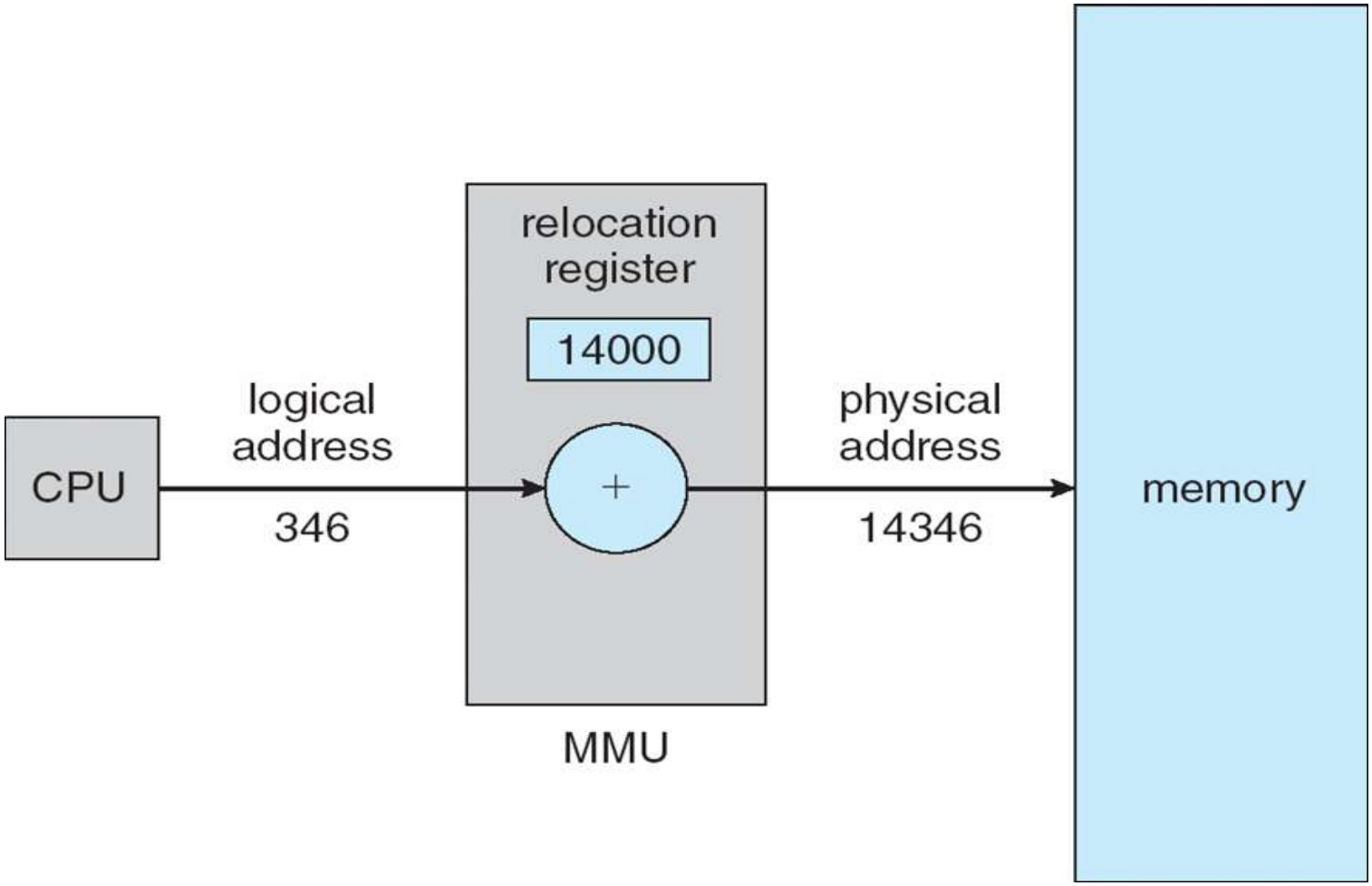
Espacio de Direcciones (Lógico vs. Físico)

- El concepto del espacio de direcciones lógicas debe ser claramente diferenciado del espacio de direcciones físico
  - Las direcciones lógicas – son generadas por el CPU; también conocidas como direcciones virtual
  - Las direcciones físicas – son las direcciones vistas por la unidad de memoria
- Las direcciones lógicas y físicas son las mismas en tiempo compilación; sin embargo, estas direcciones difieren en tiempo de ejecución
- El espacio de direcciones lógico es un conjunto de direcciones lógicas generadas por un programa
- El espacio de direcciones físico es un conjunto de direcciones físicas generadas por un programa

Unidad de Manejo de Memoria (MMU)

- Dispositivo de hardware encargado de mapear las direcciones virtuales a físicas
- Iniciemos considerando un esquema simple, donde el valor en el registro relocalización es añadido a cada dirección generada por los procesos de usuario en el momento que estos la envían a la memoria
  - El registro base ahora es llamado registro de relocalización
  - MS-DOS sobre el Intel 80x86 usaba 4 registro de relocalización
- Los programas de usuario lidian con direcciones lógicas, estos nunca lidian con direcciones físicas reales
  - En tiempo de ejecución el mapeo ocurre cuando una referencia es realizada a una porción de memoria
  - Direcciones lógicas están separadas de las direcciones físicas

Relocalización Dinámica usando el Registro de Relocalización



Relocalización Dinámica usando el Registro de Relocalización

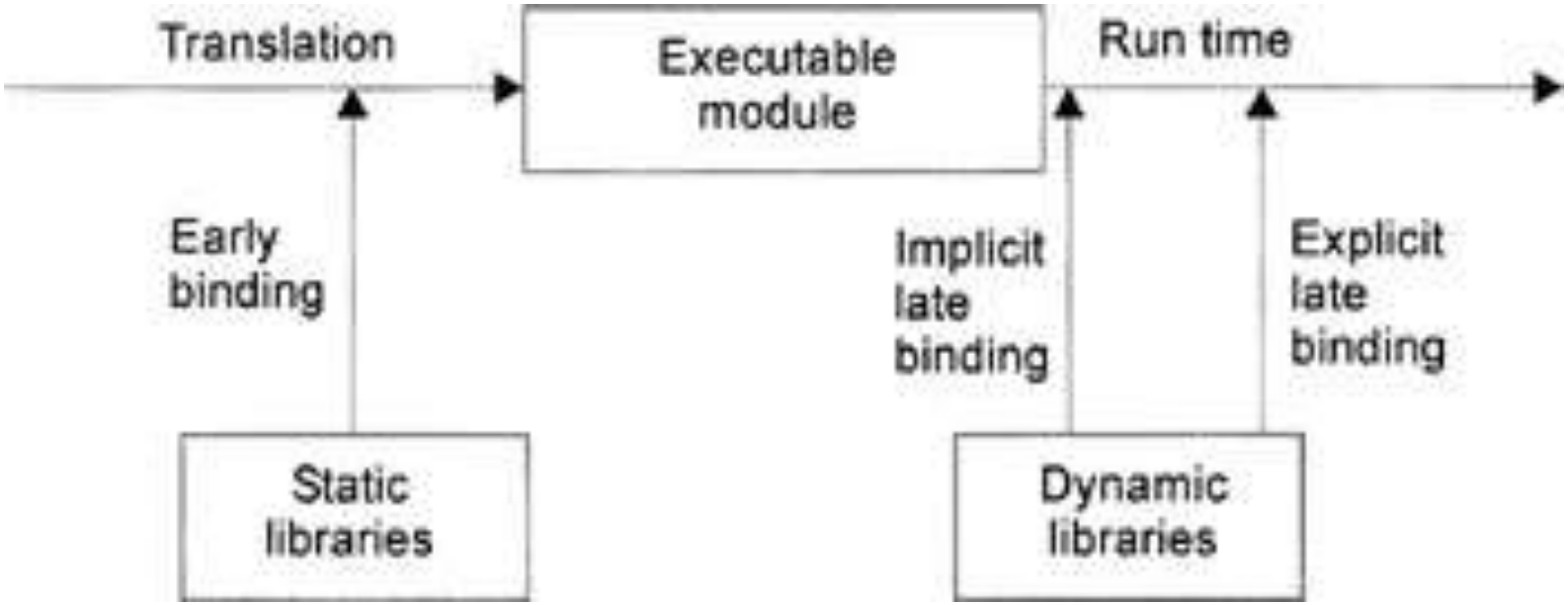
- La rutina no es cargada sino hasta que es invocada
- Se utiliza mejor el espacio en memoria
  - Rutinas que no se usan, jamás serán cargadas en memoria
- Todas las rutinas se mantienen en disco en un formato de carga relocalizable
- Es útil cuando se pretende lidiar con una cantidad grande de casos infrecuentes dentro de porciones de código
- No es necesario ningún soporte especial por parte del SO
  - La implementación recae en el diseño del programa
  - El SO podría proveer bibliotecas vía el soporte de carga dinámica de bibliotecas

Enlace Dinámico

- Enlace estático – El código del programa y las bibliotecas del sistema son combinados por el cargador en una imagen binaria del programa
- Enlace dinámico – El enlace se pospone hasta el tiempo de ejecución
- Pequeñas piezas de código, stub, se usan para determinar la rutina necesaria de la biblioteca la cual se encuentra residente en memoria
- El stub se reemplaza así mismo con la dirección de la rutina, y entonces ejecuta la rutina
- El SO determina si existe un espacio de direcciones para esa rutina en el proceso invocador
  - Si no existe, lo agrega

Enlace Dinámico

- El enlace dinámico es particularmente útil para bibliotecas
  - ¿Por qué?
- El sistema siempre conoce que bibliotecas son compartidas



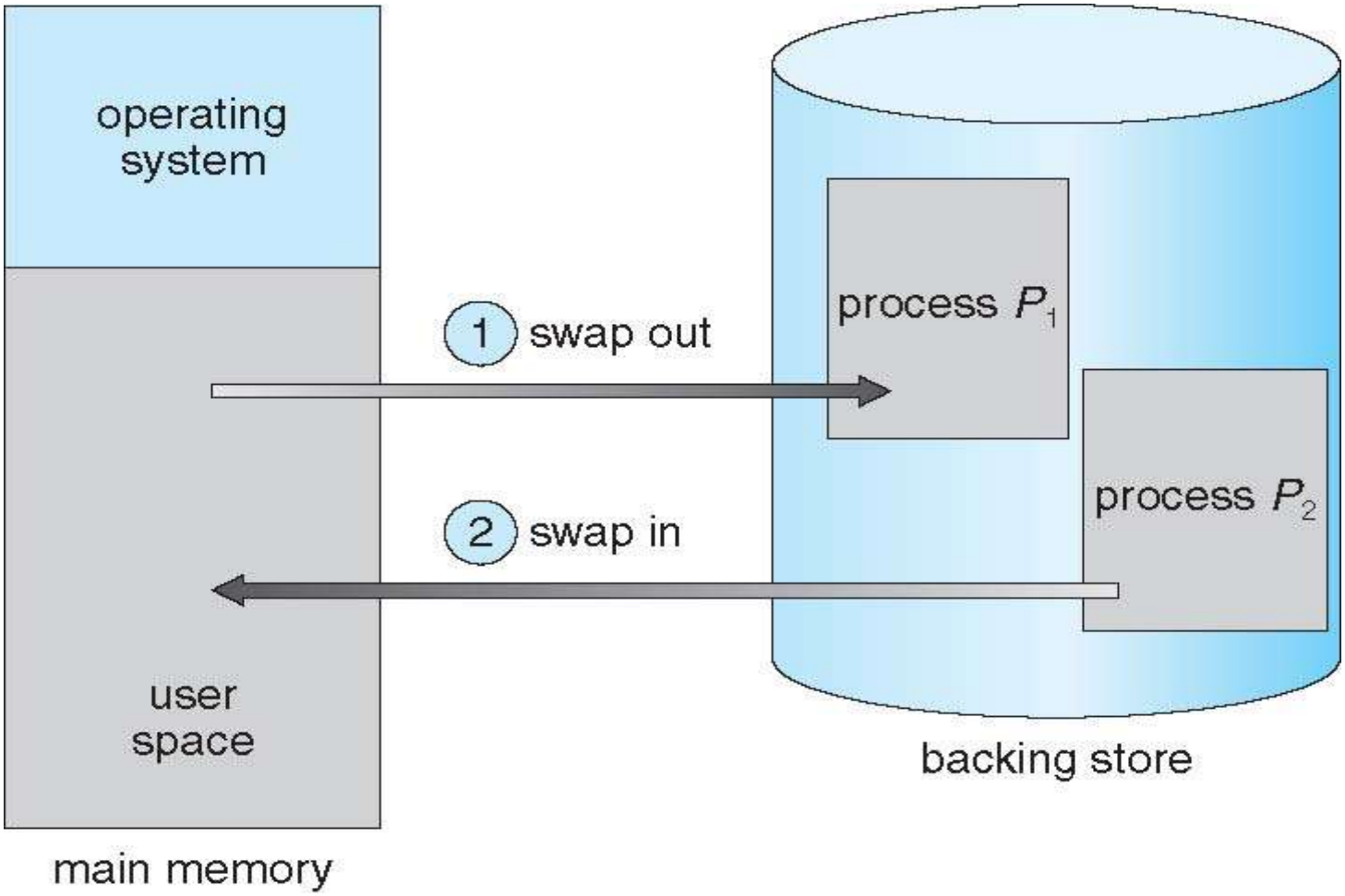
Intercambio

- Un proceso puede ser intercambiado temporalmente desde la memoria principal a un backing store, y luego ser traído de vuelta a memoria principal para continuar su ejecución
  - La cantidad de memoria necesaria por un proceso puede exceder la memoria física del sistema
- Backing store – Porción del disco lo suficientemente grande para albergar copias de imágenes de memoria de procesos de usuario
- Roll out, rool in – Proceso de intercambio de procesos, el cual es usado por ejemplo por algoritmos del planificación basados en prioridades, con la intención de ejecutar primero los procesos de mayor prioridad
- El sistema mantiene una ready queue de procesos listo para ejecutarse

Intercambio

- ¿Cuando un proceso es intercambiado (swapped out/in) necesita ser cargado en la misma dirección física?
- Depende del método de mapeo de direcciones
  - Ideas
- Modificaciones del mecanismo de swapping pueden observarse en varios SO (UNIX, Linux, y Windows)
  - El swapping normalmente se encuentra desactivado
  - Inicia la cantidad de memoria asignada sobrepasa un umbral (threshold)
  - Se deshabilita nuevamente una vez que la demanda baja del umbral establecido

Vista Esquemática del Proceso de Swapping



Tiempo de Cambio de Contexto (Swapping)

- Si el próximo proceso que debe cargarse en CPU no se encuentra en memoria, es necesario desalojar un proceso y cargar el proceso que se ha seleccionado para ejecutarse
- De esta manera el tiempo invertido en cambios de contexto podría ser largo
- Un proceso de 100MB que deba intercambiarse a disco con una tasa de transferencia de 50MB/s
  - El tiempo de desalojo será de 2000 ms o 2 s
  - Adicional al tiempo de cargar el nuevo proceso a memoria
  - En este caso el tiempo total que tomaría una cambio de texto sería de alrededor de 4000 ms o 4 s

Tiempo de Cambio de Contexto (Swapping)

- Este tiempo puede mejorarse si se reduce la cantidad de memoria a intercambiar – Esto conociendo en realidad cuanta memoria será utilizada
  - Se utilizan las llamadas al sistema request memory () y release memory () para informar al SO de la cantidad de memoria requerida
- Existen otras consideraciones que deben tomarse en cuenta
  - Solicitudes de I/O pendientes
    - ¿Qué pasa con ellas?
  - Transferir las I/O pendientes al espacio de kernel
    - Double Buffering – Agrega sobrecarga
- El swapping estándar no se utiliza en los SO modernos
  - Pero si modificaciones del mismo
    - Ejemplo

Swapping en Dispositivos Móviles

- Generalmente no es soportado
  - Basado en una memoria flash
    - Muy poca cantidad de espacio
    - Número limitado de ciclos de escritura
    - Existe un pobre desempeño en la transferencia de datos de la memoria flash y el CPU en las plataformas móviles
- En su lugar se utiliza otro método → Liberar memoria si queda poca
  - iOS pregunta a las aplicaciones si desean liberar memoria
    - Los datos de solo lectura pueden ser desalojados de la memoria y puestas en la memoria flash de ser necesario
    - Un fallo a la hora de liberación de memoria resulta en la terminación de la aplicación
  - Android termina aplicaciones si la cantidad de memoria disponible es baja, pero primero guarda el estado de la aplicación en memoria flash para mejorar el tiempo de reinicio de la misma

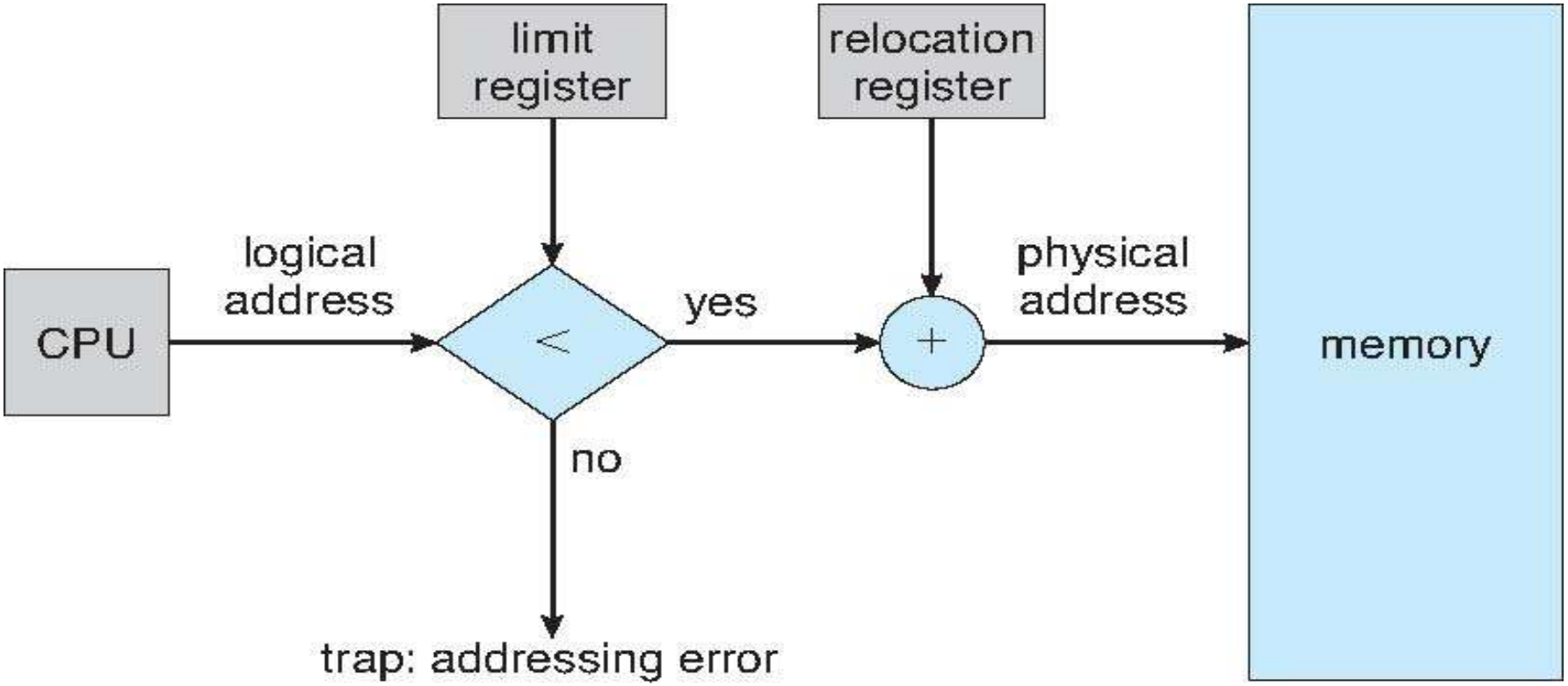
Asignación Contigua

- En la memoria principal se deben cargar tanto el SO como los procesos de usuario
- Recurso limitado, por lo que debe ser asignado eficientemente
- La asignación contigua es uno de los primeros métodos
- La memoria principal usualmente se particiona en dos:
  - SO residente, usualmente asignado a las direcciones bajas de memoria junto al vector de interrupciones
  - Los procesos de usuario se mantienen en posiciones altas de memoria
  - Cada proceso consta de un espacio de direcciones de memoria contiguo

Asignación Contigua

- Los registros de relocalización son usados para proteger a los procesos de usuario (unos de otros), y para cambiar código y datos del SO
  - El registro base contiene valor de pequeñas direcciones físicas
  - El registro limite contiene el rango de direcciones lógicas – cada dirección lógica debe ser menor que el valor del registro limite
  - La MMU mapea direcciones lógicas dinámicamente
  - Esto permite acciones como que el código del kernel sea transitorio y pueda cambiar de tamaño

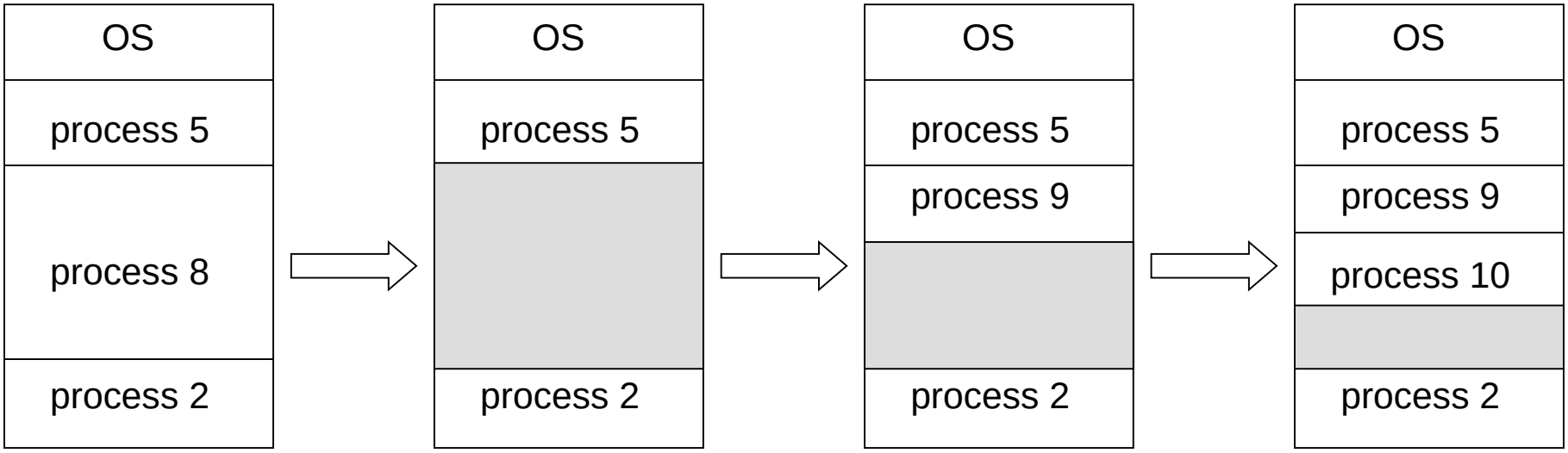
Soporte de Hardware para la Relocalización y Registro Limite



Asignación Contigua

- Asignación de particiones
  - El grado de multiprogramación limita el número de particiones
  - Particiones de tamaño variable por eficiencia podrían ser utilizadas (el tamaño es dado por los requerimientos del proceso)
  - Agujeros – Con bloques de tamaño variables provoca que agujeros de diferentes tamaños se dispersen en toda la memoria
  - Cuando un proceso llega al sistema, se le asigna un porción de memoria contigua capaz de almacenarlo
  - Cuando el proceso termina se libera la partición, y si existe espacio libre de forma contigua se combinan
  - El SO mantiene información sobre:
    - Particiones asignadas
    - Particiones o agujeros libres

Asignación Contigua



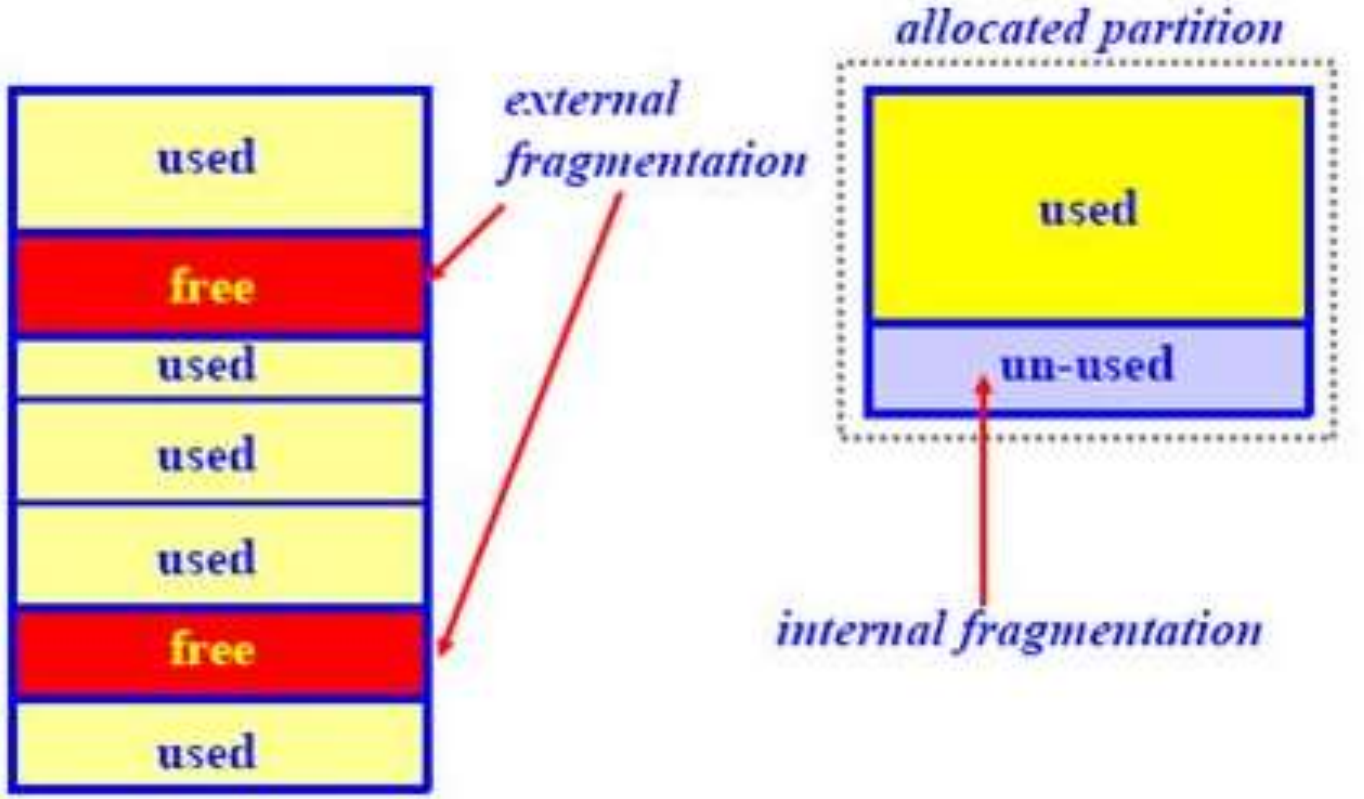
Problemas de Asignación Dinámica

- ¿Cómo satisfacer una solicitud de tamaño n con una lista de agujeros libres?
  - Primer Ajuste: Asigna el primer agujero libre con el tamaño suficiente para satisfacer la solicitud
  - Mejor Ajuste: Asigna el agujero mas pequeño con el tamaño suficiente para satisfacer la solicitud. Esto implica una búsqueda en una lista.
    - Produce el menor desperdicio posible
  - Peor Ajuste: Asigna el agujero mas grande a la solicitud que se pretende satisfacer. Esto implica una búsqueda en un lista.
    - Produce el mayor desperdicio posible
  - Primer y Mejor ajuste son mejores que Peor ajuste en términos de velocidad y utilización del espacio de almacenamiento respectivamente.

Fragmentación

- Fragmentación Externa – Espacio total no contiguo de memoria existente para satisfacer solicitudes
- Fragmentación Interna – La cantidad de memoria asignada en una solicitud podría ser mayor a lo deseado, la diferencia entre lo asignado y lo requerido es espacio sin utilizar dentro de la partición. A este fenómeno se le conoce como fragmentación interna
- ¿Cómo expresaría estos conceptos gráficamente?

Fragmentación



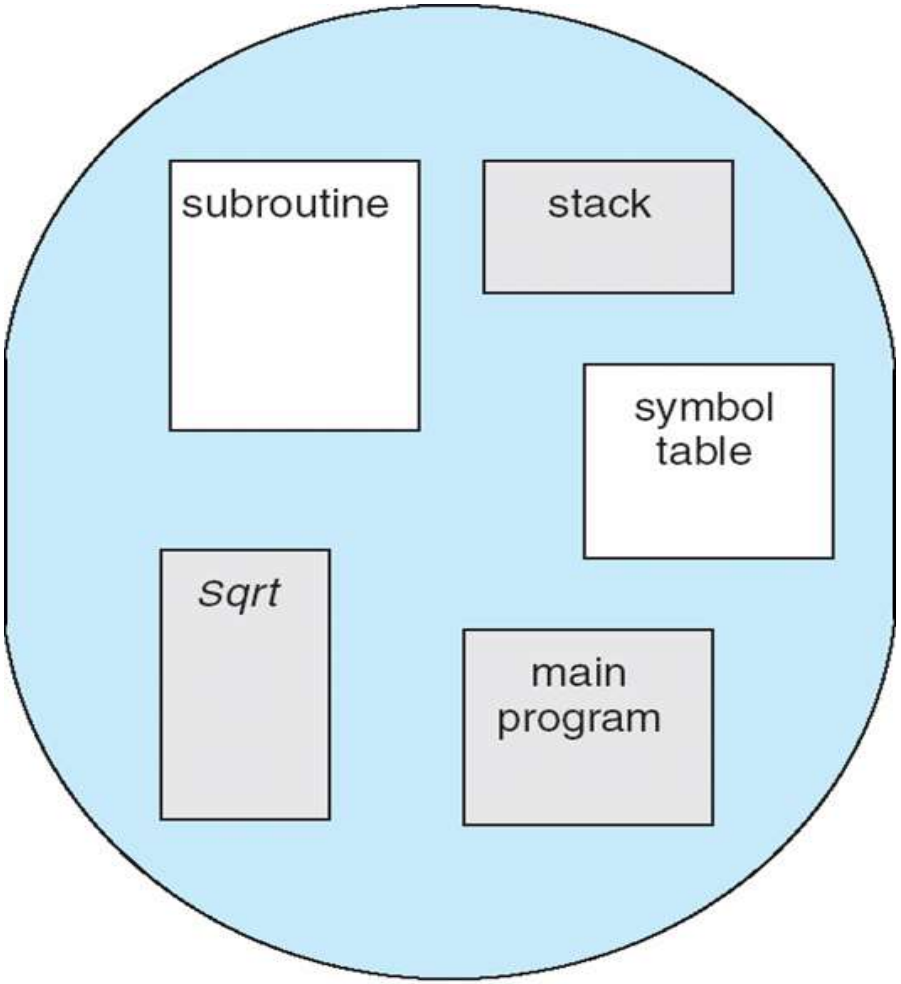
Fragmentación

- La fragmentación externa puede ser reducida realizando compactación
  - Compactar o unir las porciones de memoria libres que se encuentran esparcidas en toda la memoria principal
    - Implicaciones
  - La compactación solo es posible si la relocalización es dinámica, y se realiza en tiempo de ejecución
    - ¿Por qué?

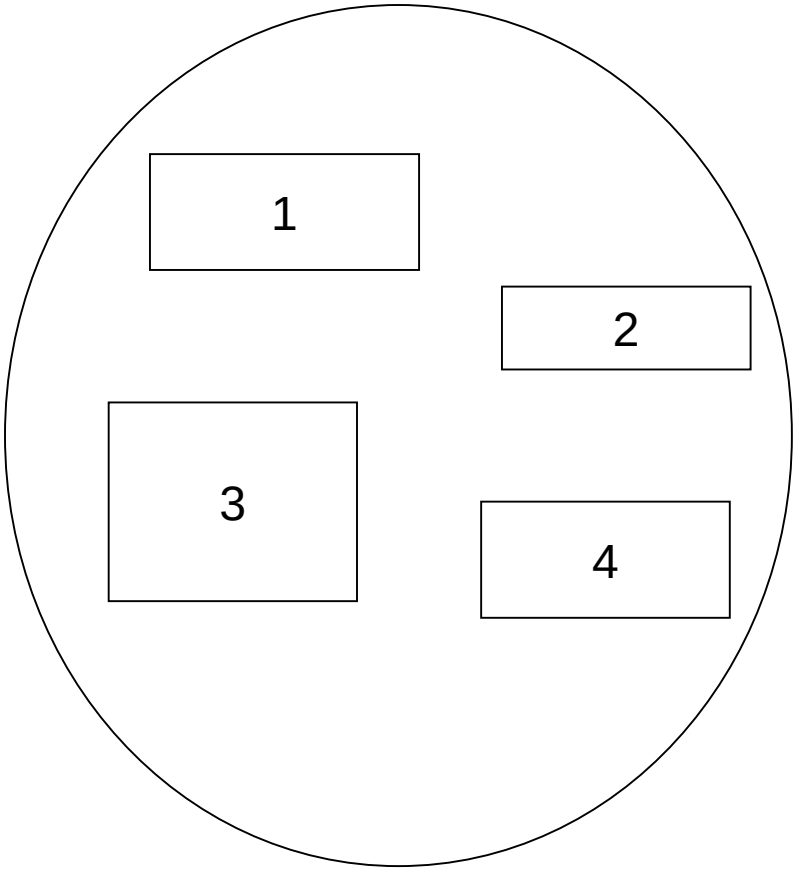
Segmentación

- Esquema de manejo de memoria que brinda el soporte al usuario de visualizar como se organiza la memoria principal
- Un programa es una colección de segmentos
  - Un segmento es una unidad lógica como:
    - El programa principal
    - Procedimientos
    - Funciones
    - Métodos
    - Objetos
    - Variables locales y globales
    - Bloques comunes
    - Pila
    - Tabla de Símbolos
    - Arreglos

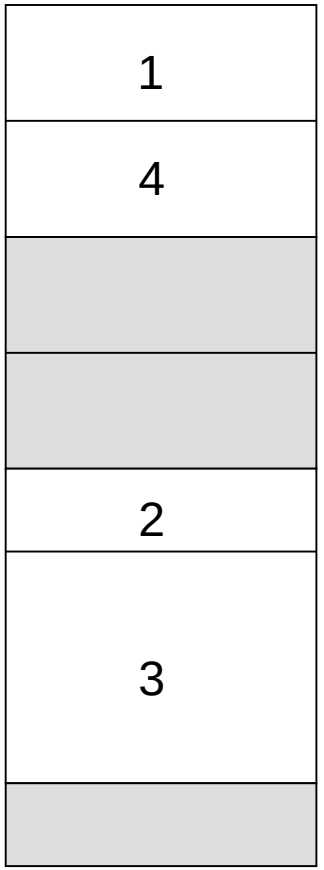
Un Programa desde la Perspectiva del Usuario



Vista Lógica de la Segmentación



user space



physical memory space

Arquitectura de Segmentación

- El espacio de direcciones lógico consiste en una tupla:
  - <#segmento, desplazamiento>
- Tabla de segmentos – Mapea el espacio de direcciones en dos dimensiones con una dirección física. Cada entrada en la tabla contiene:
  - Base – Contiene la dirección física de comienzo de segmento en memoria
  - Limite – Especifica la longitud del segmento

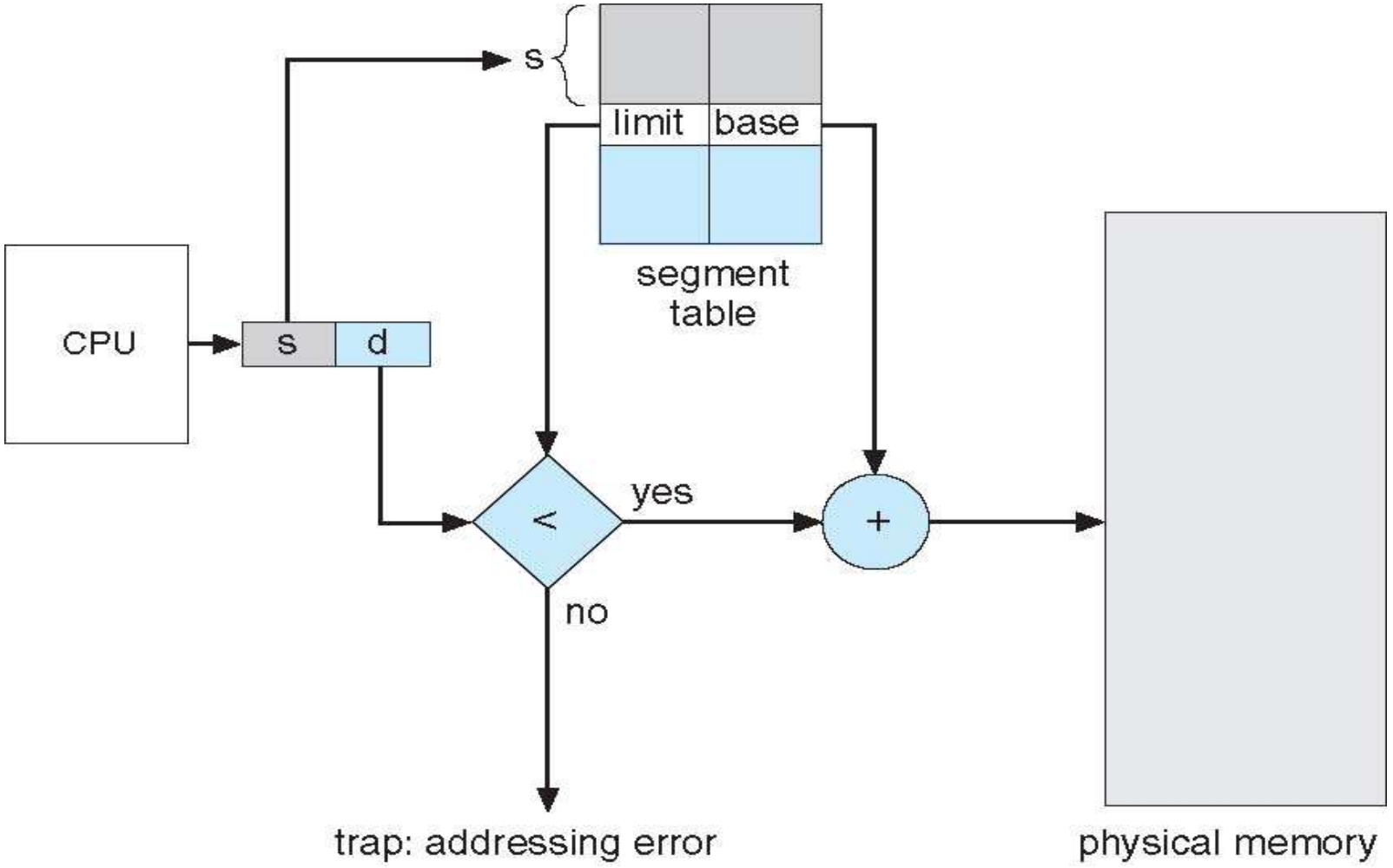
Arquitectura de Segmentación

- Segment-table base register (STBR) – Apunta a la posición de memoria en donde se localiza la tabla de segmentos
- Segment-table length register (STLR) – Indica el número de segmentos usados por el programa
  - Un número de segmento s es válido si s < STLR

Arquitectura de Segmentación

- Protección
  - Cada entrada en la tabla de segmento tiene asociado
    - Un bit de validación. 0 → Segmento no válido
    - Privilegios de lectura/escritura/ejecución
  - Bits de protección son asociados con cada uno de los segmentos

Hardware de Segmentación



Paginación

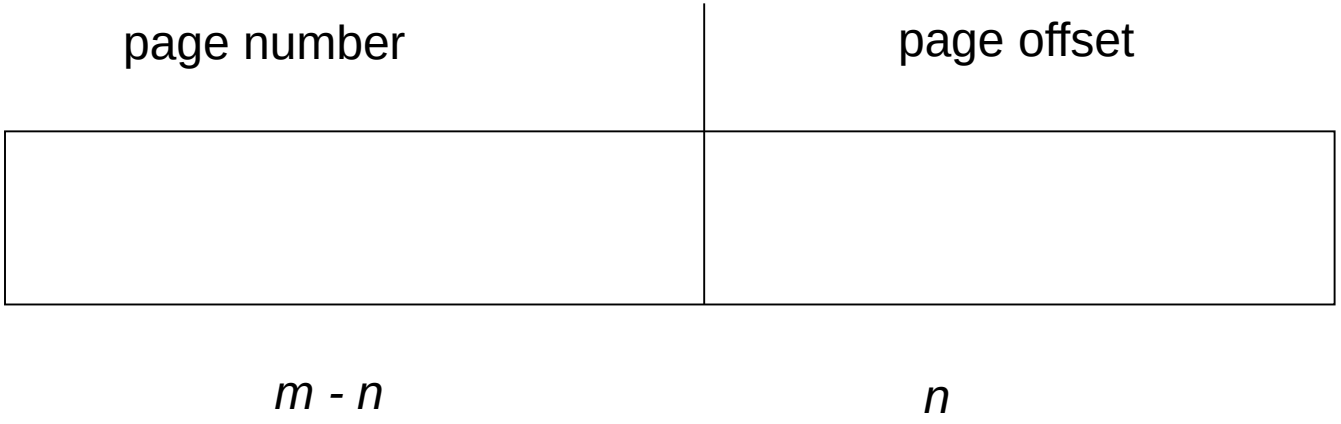
- El espacio de direcciones de un proceso no necesariamente debe ser contiguo
  - Evita la fragmentación externa
  - Evita el problema que genera tener trozos de memoria variable
- Se divide la memoria física en bloques de tamaño fijo llamados marcos
  - El tamaño es potencia de 2, generalmente entre 512 B y 16MB
  - ¿Por qué?
- El espacio de direcciones lógico se divide el bloques del mismo tamaño llamados páginas
- Es necesario mantener el estado de los marcos libres

Paginación

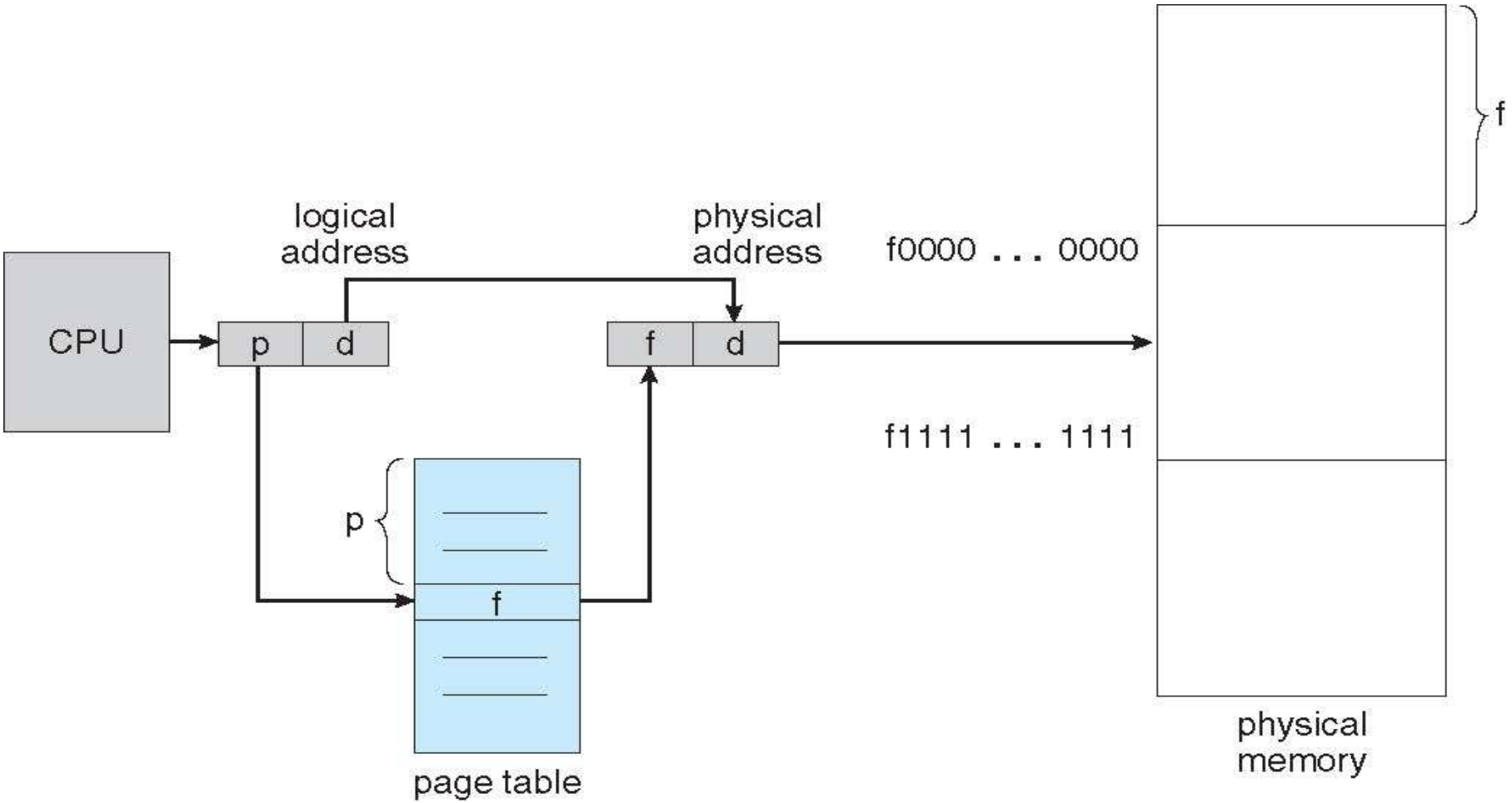
- Para ejecutar un programa con N marcos, es necesario ubicar N marcos libres en memoria principal para cargarlo
- Se utiliza un tabla de páginas para realizar el proceso de traducción de direcciones
- Bajo este esquema sufre de fragmentación interna

Esquema de Traducción de Direcciones

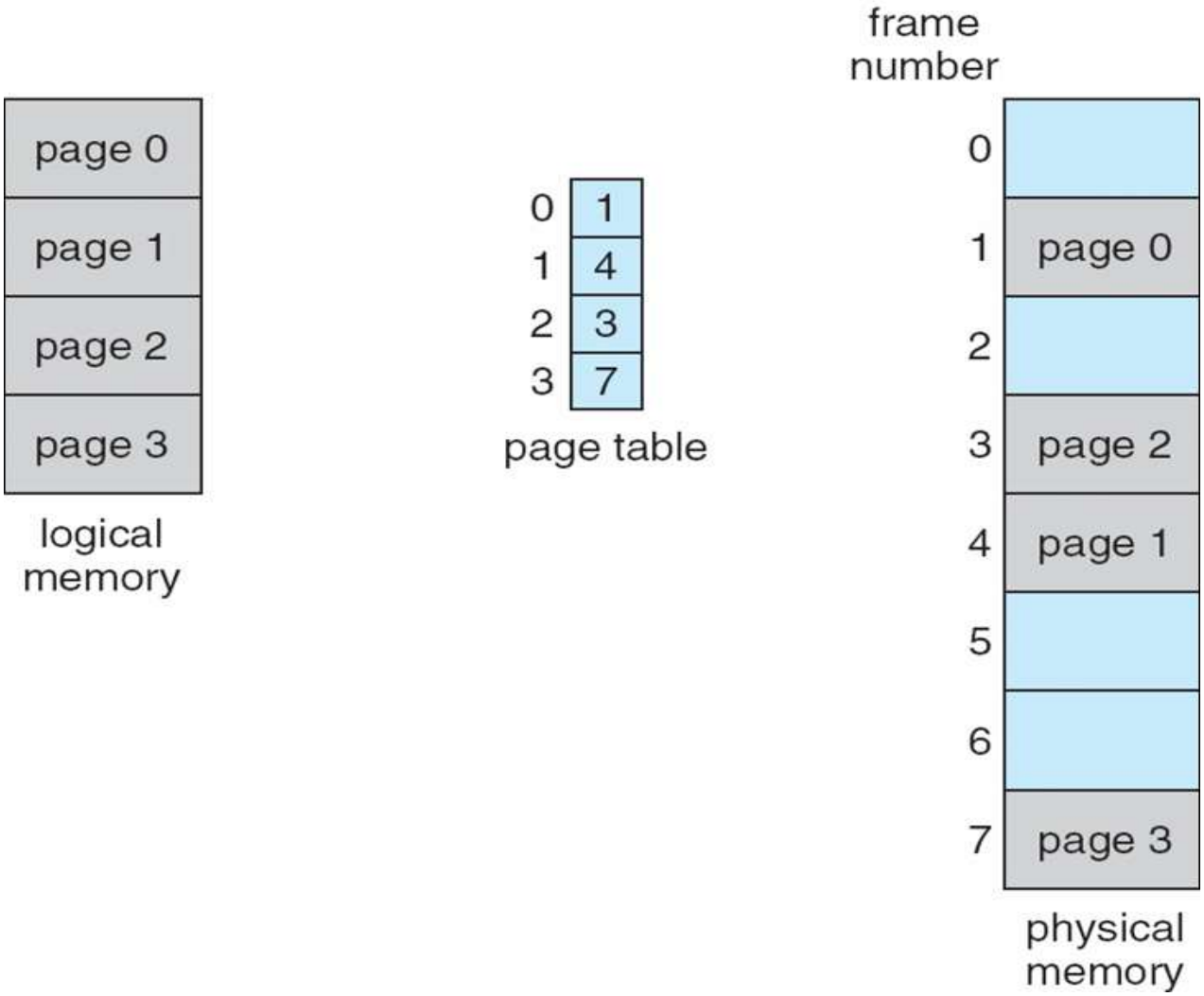
- Un dirección generado por el CPU se divide en:
  - #pagina (p) – usado como índice en la tabla de páginas, la cual contiene la base de cada marco
  - Desplazamiento (d) – se combina con la dirección base para determinar la dirección física que será enviada a la MMU



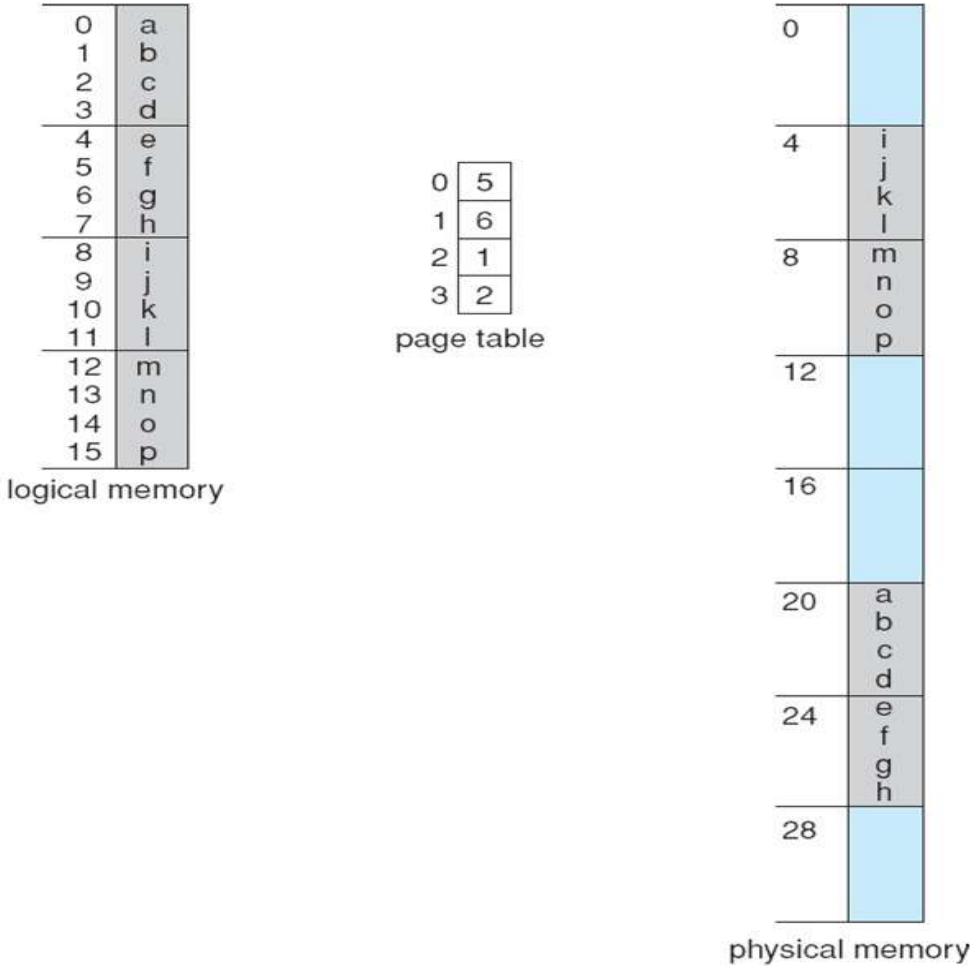
Hardware de Paginación



Modelo Lógico y Físico de Memoria Paginada



Ejemplo de Paginación



n=2 and *m*=4 32-byte memory and 4-byte pages

Paginación

- Calculando la fragmentación interna
  - Tamaño de la página: 2048B
  - Tamaño del proceso: 72766B
  - 35 paginas + 1086B → ¿Cómo se obtienen estos datos?
  - En el peor caso la fragmentación será:
    - 1 marcos – 1 B
  - En promedio la fragmentación será:
    - ½ marco de página
  - Con base a lo anterior
    - ¿Son deseables marcos pequeños?
    - ¿Qué pasa con el tamaño de la tabla de páginas?

Paginación

- La vista del espacio de direcciones del proceso y de la memoria física ahora en muy diferente
- Por implementación los procesos solo pueden acceder a su propia memoria

Marcos Libres

Before allocation

After allocation

Implementación de la Tabla de Páginas

- La tabla de página se mantiene en memoria principal
- Page-Table Base Register (PTBR) – Apunta a la tabla de páginas
- Page-Table Lenght Register (PTLR) – Indica el tamaño de la tabla de páginas
- Bajo este esquema cada acceso a datos/instrucciones requieren de dos accesos a memoria
  - Uno a la tabla de páginas y otro para recuperar el dato/instrucción
- ¿Cómo podría mejorarse esta situación?

Implementación de la Tabla de Páginas

- El inconveniente de los dos accesos a memoria puede ser mejorado una memoria cache especial para búsquedas llamado memoria asociativa o translation look-asside buffer (TLBs)
- Algunos TLBs almacenan address-space identifiers (ASIDs) en cada entrada de la TLB – Identifican a cada proceso para proveer protección a su espacio de direcciones
  - De no brindar este mecanismo de protección, la TLB debe ser vaciada en cada cambio de contexto
- Típicamente la TLB es de tamaño reducido
  - 64 a 1024 entradas

Implementación de la Tabla de Páginas

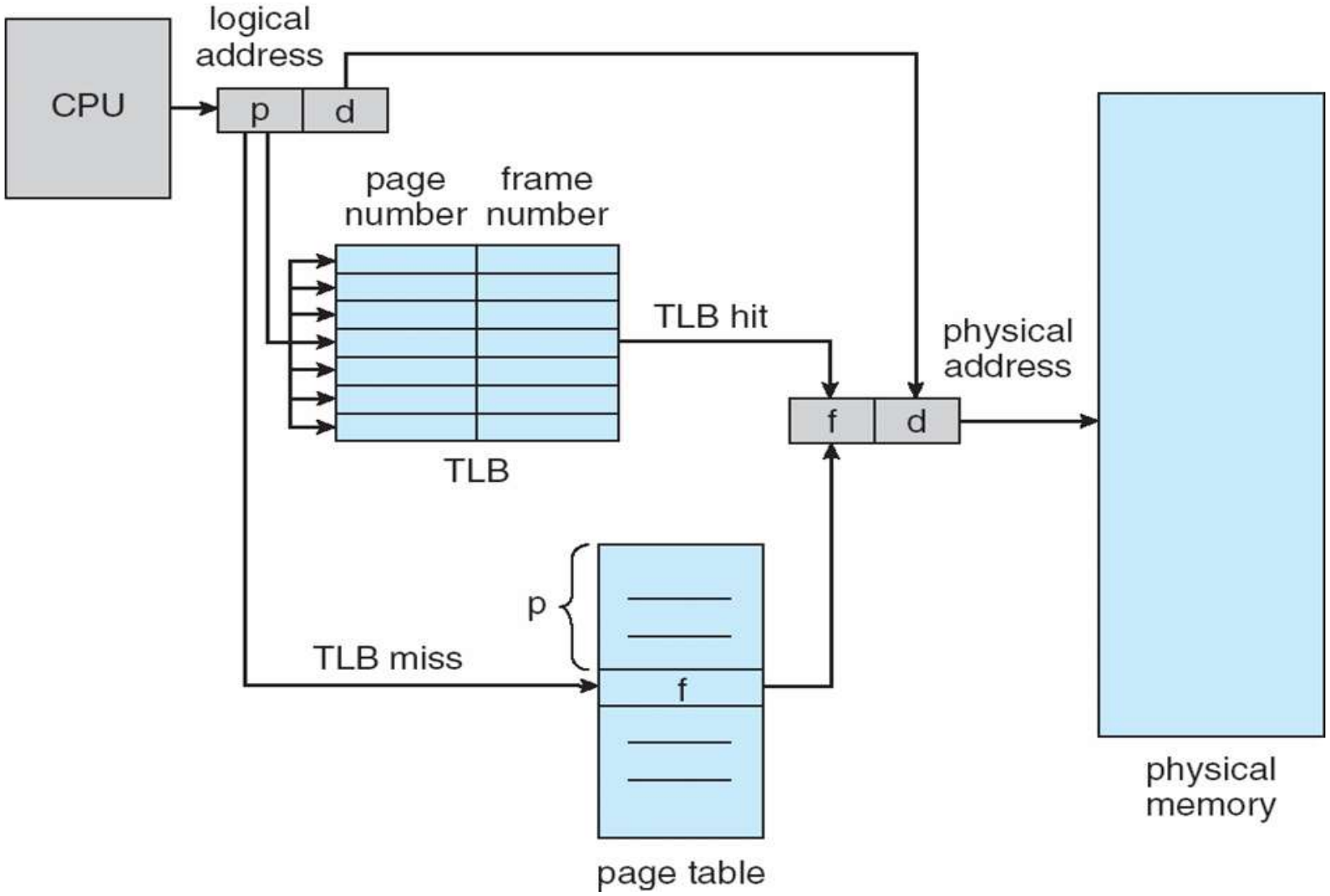
- Cuando ocurre un fallo en la TLB, el valor de la posición de memoria recuperada se almacena para los próximos accesos
  - Una política de reemplazo debe ser considerada
  - Algunas entradas pueden perdurar en la TLB para un rápido acceso

Memoria Asociativa

- Memoria asociativa – Búsqueda en paralelo
- Traducción de direcciones
  - Si p se encuentra en un registro asociativo, obtenga el # marco de correspondiente
  - De otra manera obtenga el # del marco de la tabla de páginas en memoria

Page #	Frame #

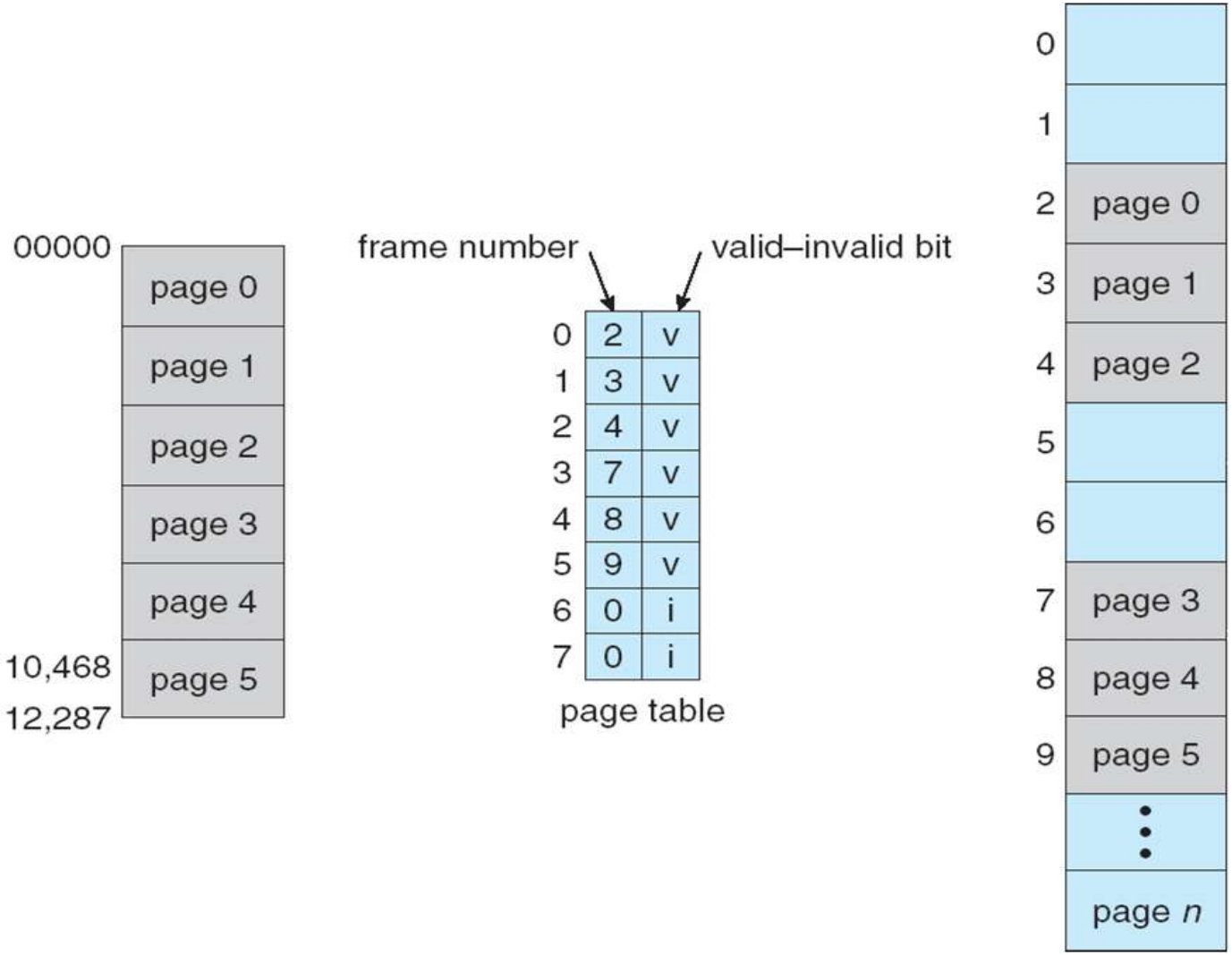
Hardware de Paginación con TLB



Protección de Memoria

- La protección de memoria se implementa asociando un bit de protección con cada marco para indicar si es posible leer o escribir en el marco
  - Adicionalmente es posible agregar más bits para confirmar si permisos de ejecución
- Un bit de no válido se asocia a cada entrada en la tabla de páginas
  - Válido – Indica que la página asociada se encuentra dentro del espacio de direcciones lógicas del proceso
  - Inválido – Indica que la página no se encuentra dentro del espacio de direcciones lógico del proceso
- Cualquier violación resulta en un trap

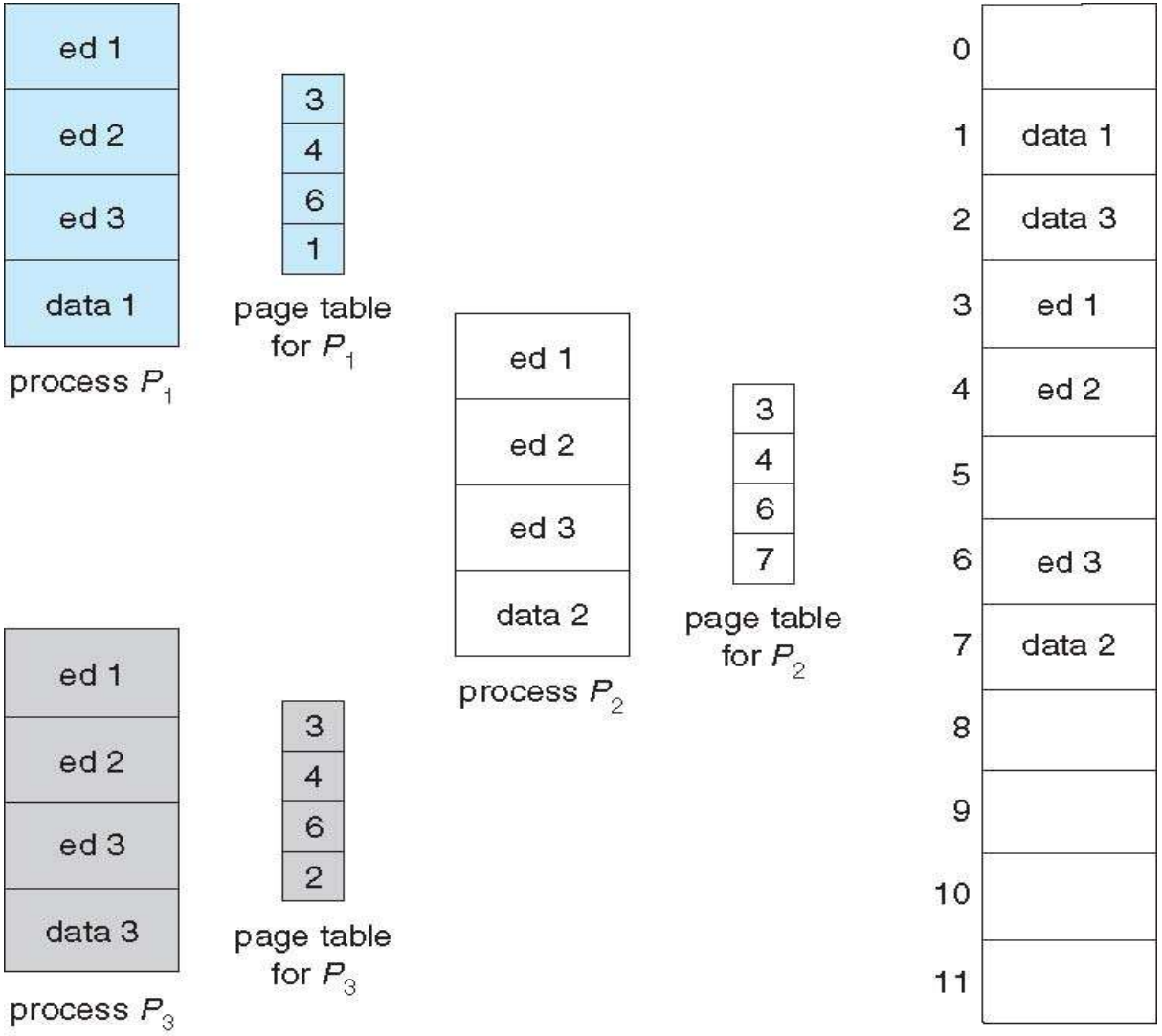
Bits de Validación en la Tabla de Páginas



Compartición de Paginas

- Código compartido
  - Una copia de código en modo solo lectura podría ser compartido entre los procesos
  - Similar a la compartición del espacio del proceso por parte de sus threads
  - También es útil en IPC se páginas de lectura/escritura se permiten
- Código privados y sus datos
  - Cada proceso mantiene por separado una copia de su código y sus datos
  - Las páginas privadas para código o datos pueden aparecer en cualquier parte del espacio de direcciones lógico del proceso

Ejemplo Páginas Compartidas



Estructura dela Tabla de Páginas

- La estructura de la tabla de páginas en memoria puede ser bastante grande
  - Considere un espacio de direcciones lógico de 32 bits
  - Páginas de tamaño 4KB (2^{12})
  - La tabla de páginas tendrá 1 millón de entradas (2^{32}/ 2^{12})
  - Si cada entrada es de 4 bytes → 4 MB del espacio físico de direcciones / memoria será utilizado solo para la tabla de páginas
    - La cantidad de memoria usada podría convertirse en un inconveniente

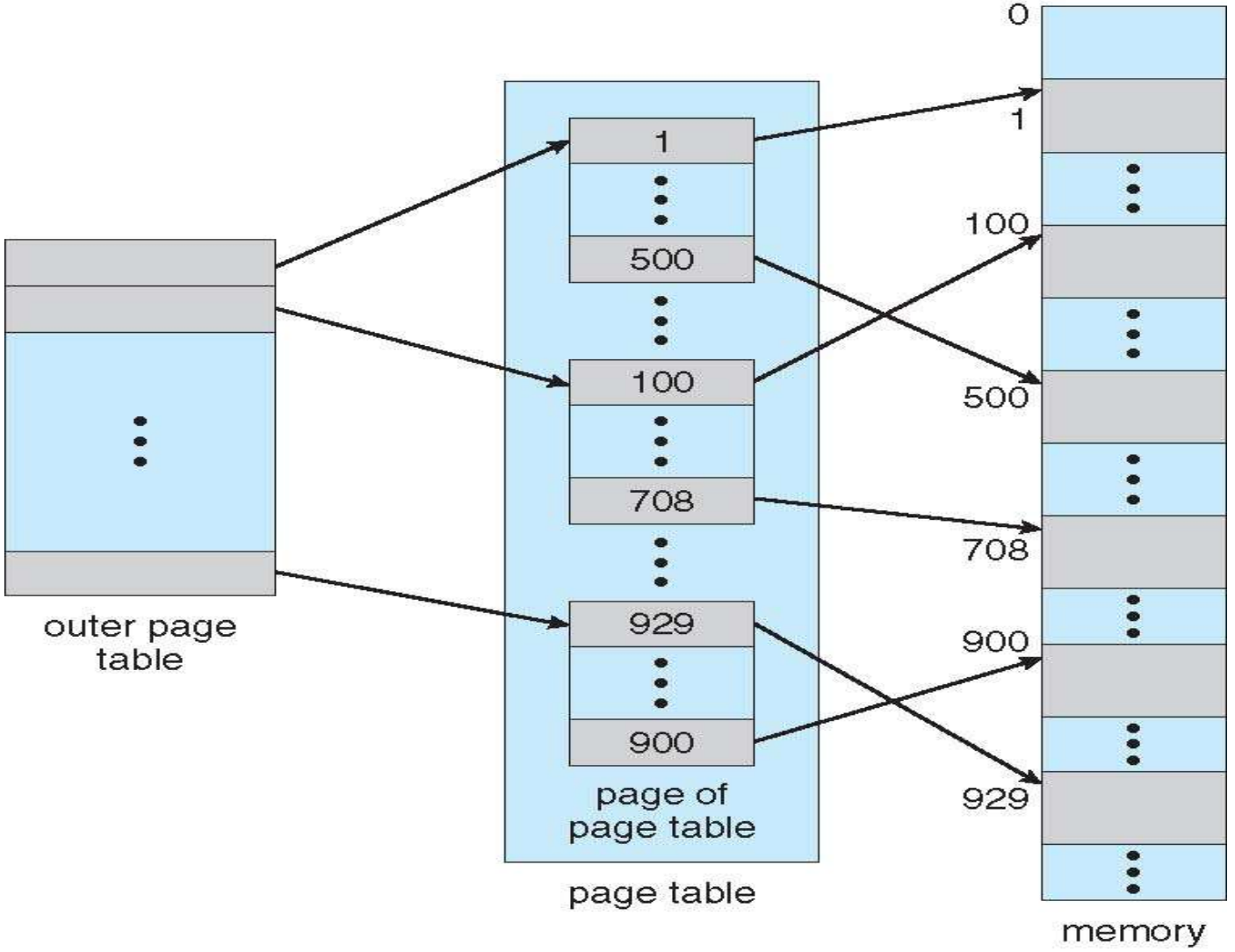
Estructura dela Tabla de Páginas

- Tabla de Paginas Jerárquicas
- Tabla de Páginas Hash
- Tabla de Páginas Invertida

Tabla de Paginas Jerárquicas

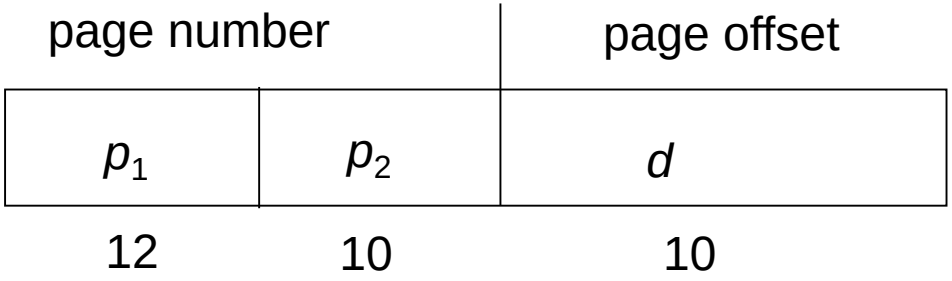
- Dividir el espacio de direcciones lógico en múltiples tablas de páginas
- Una técnica sencilla es usar una tabla de páginas de dos niveles
- Ahora tendremos un directorio a la tabla de páginas

Esquema de una Tabla de Páginas de Dos Niveles

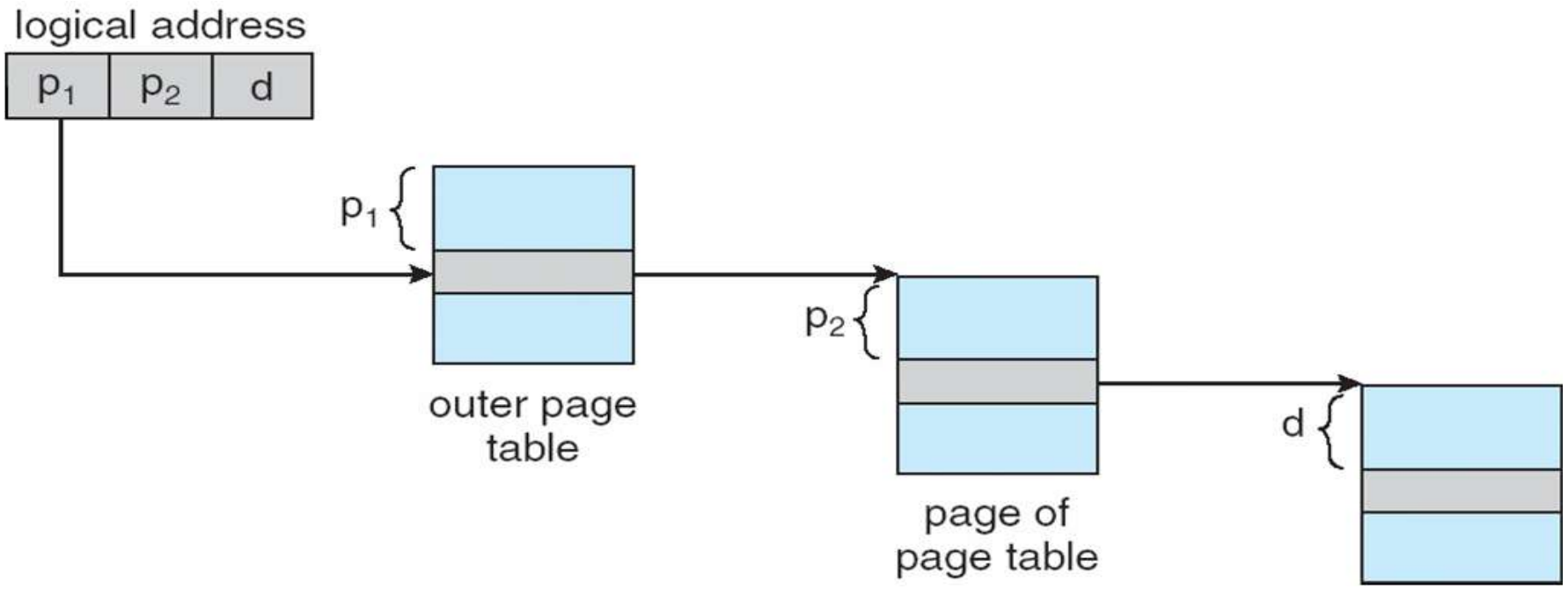


Ejemplo de Paginación en Dos Niveles

- El espacio de direcciones (una máquina de 32 bits con páginas de 1K) se divide en:
  - 22 bits para números de páginas
  - 10 bits para el desplazamiento
- Dado que la tabla de páginas es paginada, los bits de #páginas se dividen en:
  - 12 bits para el #página
  - 10 bits de desplazamiento

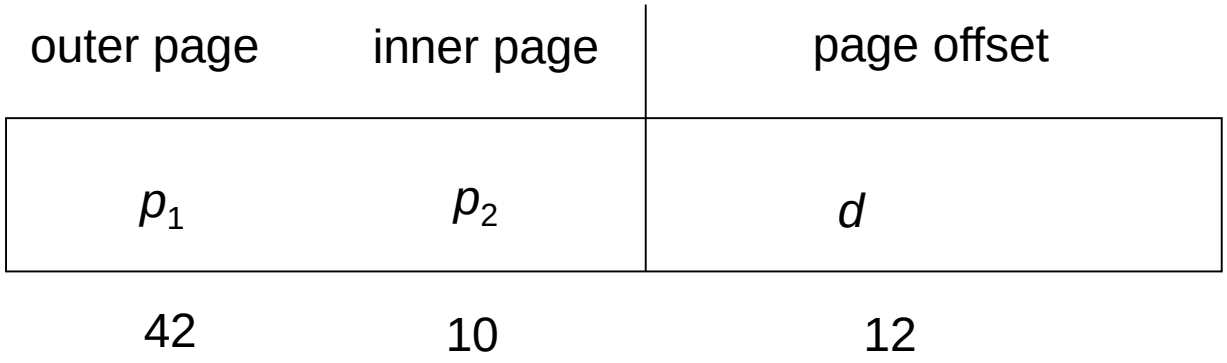


Esquema de Traducción de Direcciones

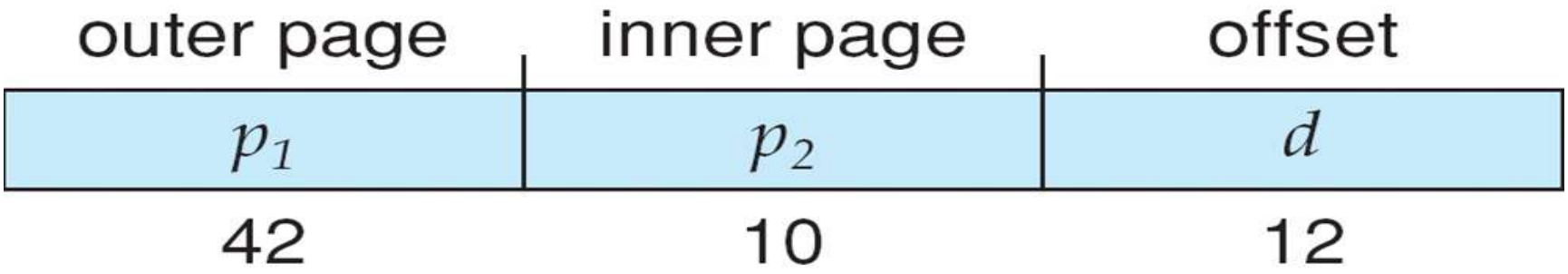


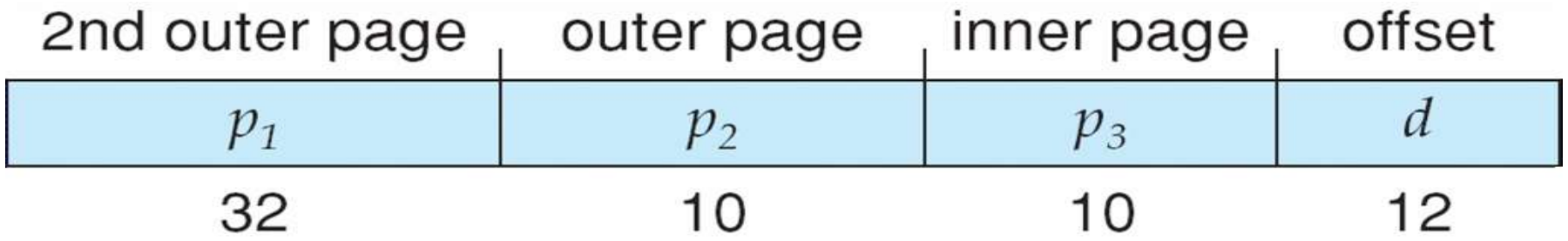
Espacio de Direcciones Lógico en 64bits

- El esquema de paginación en dos niveles podría ser insuficiente
- Si el tamaño de la página es de 4KB (2^{12})
  - En un esquema de dos niveles, las tablas de páginas de segundo nivel podrían ser de 2^{10}
  - ¿Implicaciones?



Esquema de Paginación en Tres Niveles





Tablas de Páginas Hash

- Comunes en espacios de direcciones > 32 bits
- Un número de página virtual es hashed en una tabla de páginas
  - Esta tabla de páginas contiene una cadena de elementos
  - Cada elemento contiene
    - #página virtual
    - El valor del marco de página mapeado
    - Un apuntador al siguiente elemento
  - Los números de página virtual son comparadas en la cadena de elementos en busca de una coincidencia
    - Si la coincidencia existe, el # de marco físico correspondiente se extrae

Tablas de Páginas Hash

- Una variación es usada en espacio de 64 bits → Tabla de Páginas Clusterizadas
  - Similar al hash descrito pero cada entrada hace referencia a un conjunto de páginas (por ejemplo 16, en lugar de 1)
  - Sumamente útil en espacios de memoria esparcidos

Tablas de Páginas Hash

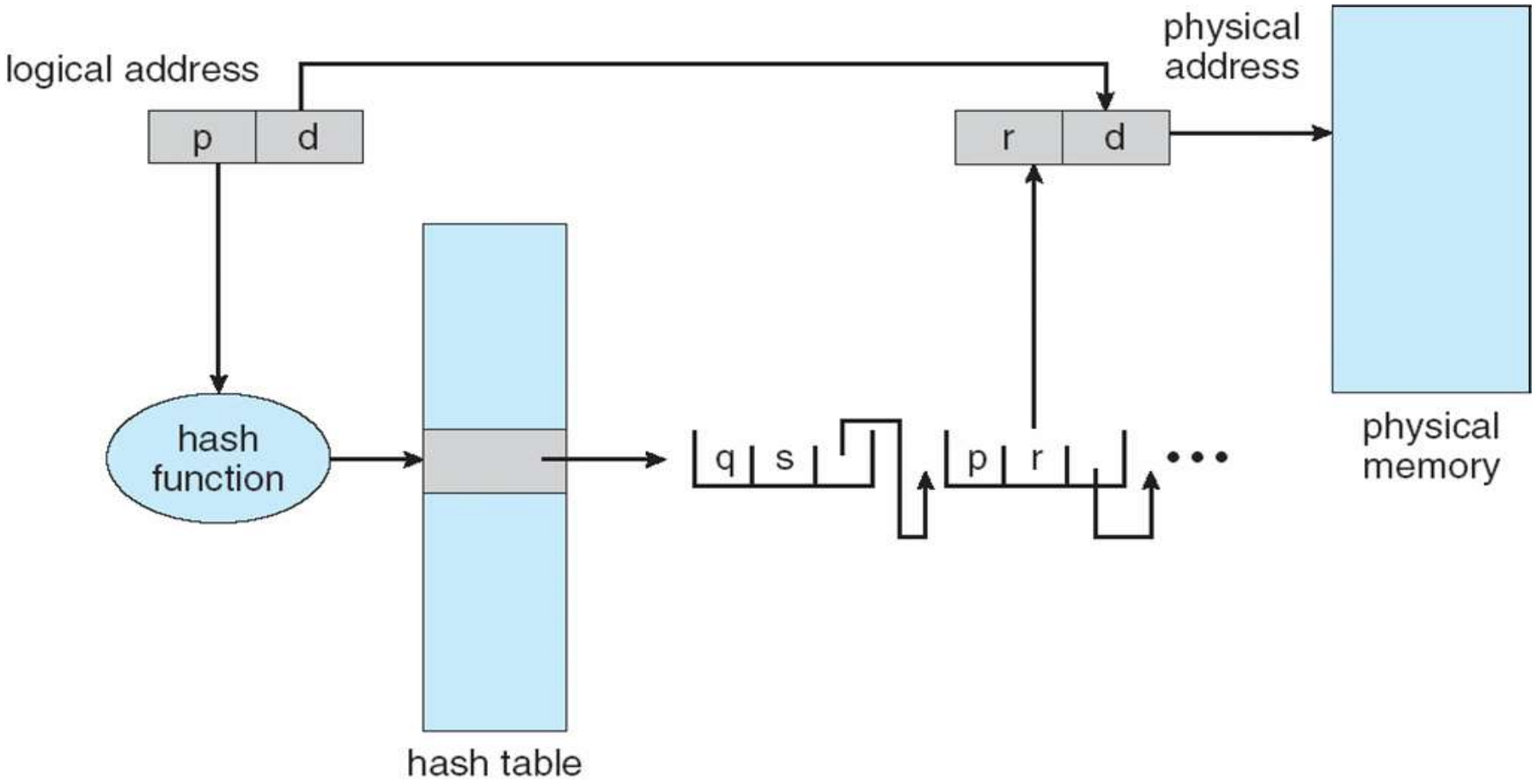


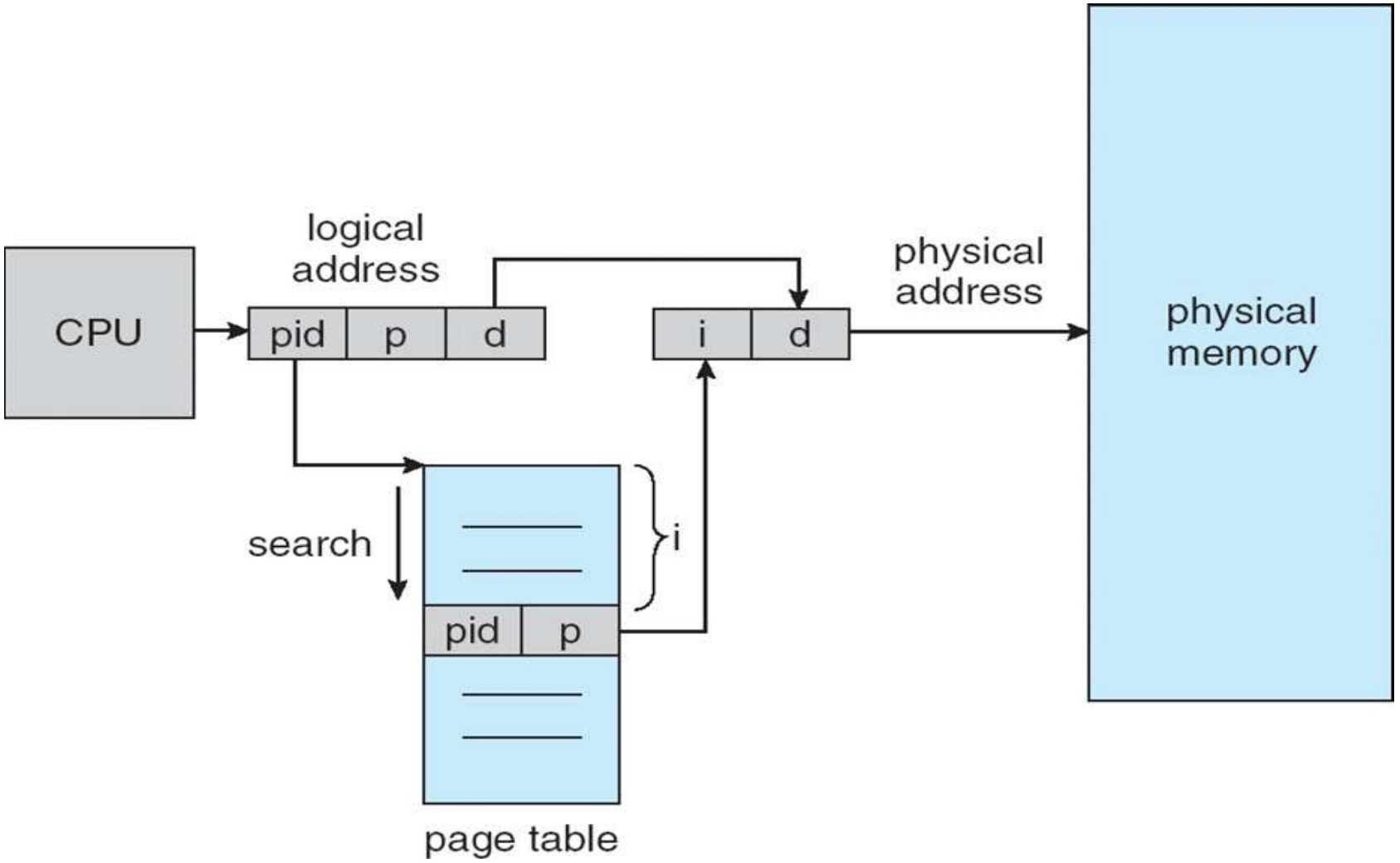
Tabla de Páginas Invertida

- En lugar de asignar una tabla de páginas a cada proceso y seguir la pista de todas sus posibles direcciones lógicas, se sigue la pista de todas las páginas físicas (marcos)
- Una tabla cuya entrada se corresponde con cada página real en la memoria
- Las entradas consisten en una dirección virtual de la página almacenada en una locación real en la memoria, con la información del proceso que es dueño de dicha página

Tabla de Páginas Invertida

- Se utiliza una tabla hash para delimitar las búsquedas
  - Una tabla hash puede acelerar los accesos
- ¿Cómo implementar memoria compartida?
  - Mapeando direcciones virtuales a direcciones físicas compartidas

Arquitectura de una Tabla de Páginas Invertida



Ejemplo: Intel 32 bits

- Chips predominantes
- Los CPUs Pentium de 32 bits → IA-32
- Discusión

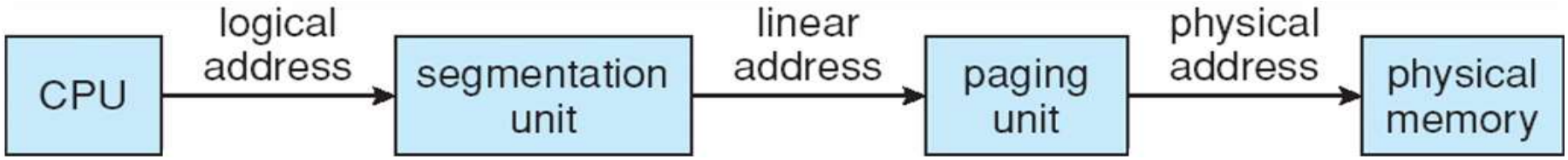
IA-32

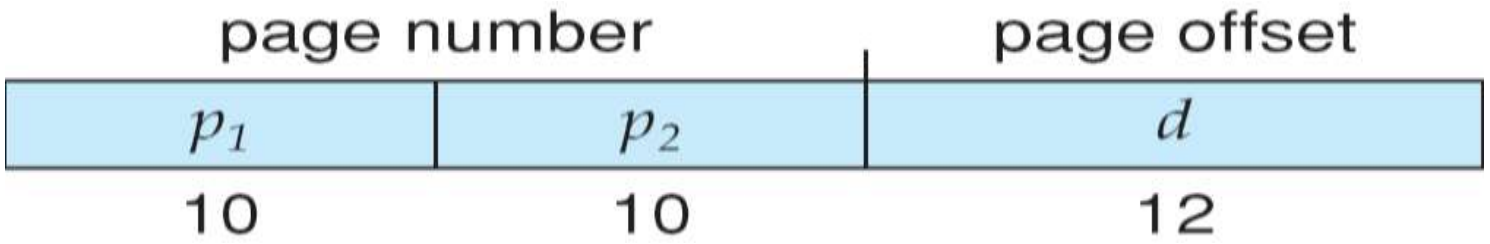
- Soporta segmentación y segmentación con paginación
  - Cada segmento puede ser de hasta 4GB
  - Hasta 16K segmentos por proceso
  - Divididos en dos particiones
    - La primera partición hasta los 8K segmentos → LDT – Local Descriptor Table
    - La segunda partición por encima de los 8K segmentos → GDT – Global Descriptor Table
  - El CPU genera direcciones lógicas
    - El selector es insumo de la unidad de segmentación
      - Produce direcciones lineales

IA-32

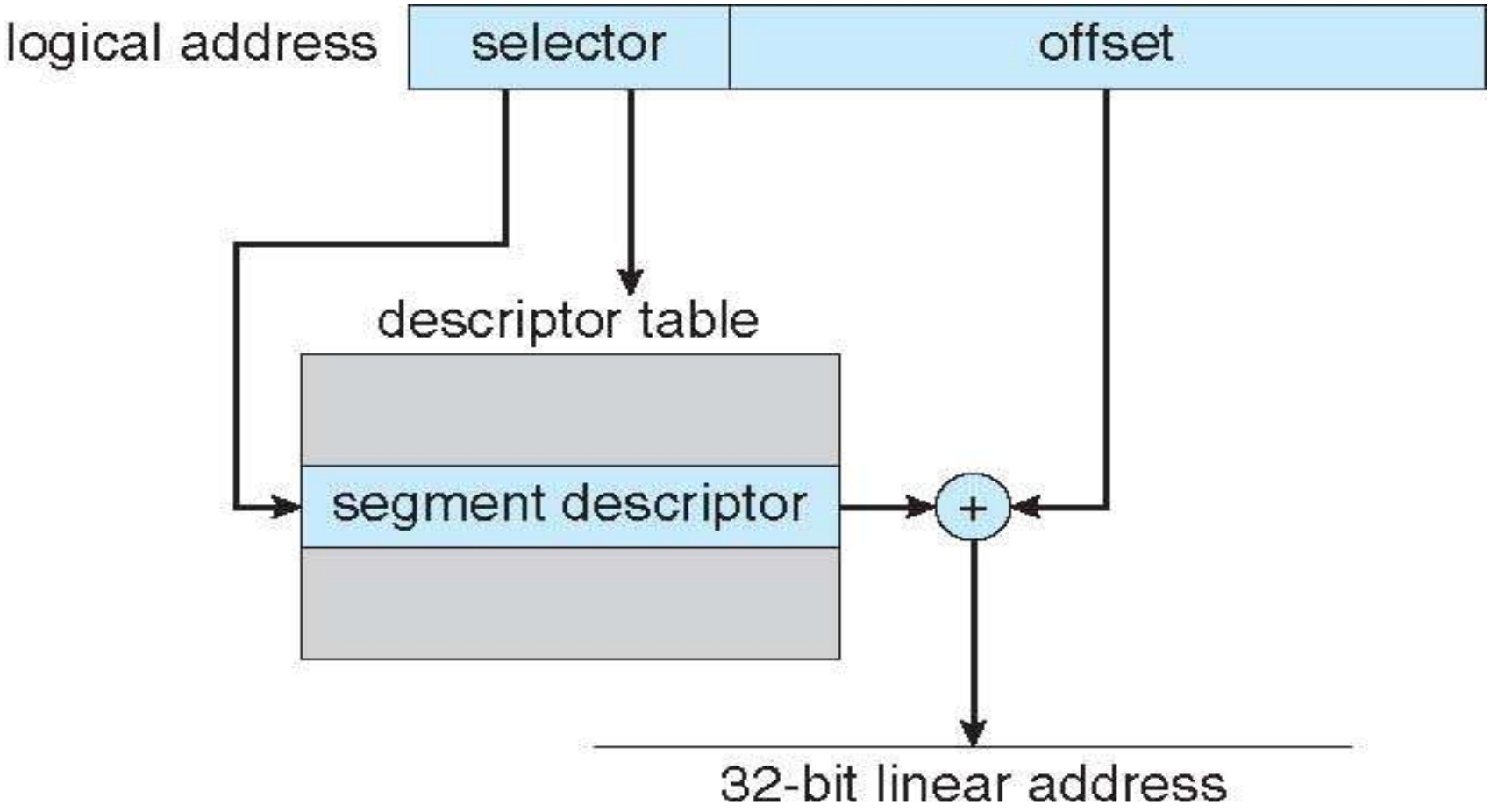
- Las direcciones lineales son insumos de la unidad de paginación
  - Genera direcciones físicas en memoria principal
  - La unidad de paginación es equivalente a la MMU
  - Los tamaños de página pueden ser de 4KB o 4MB

Traducción en IA-32

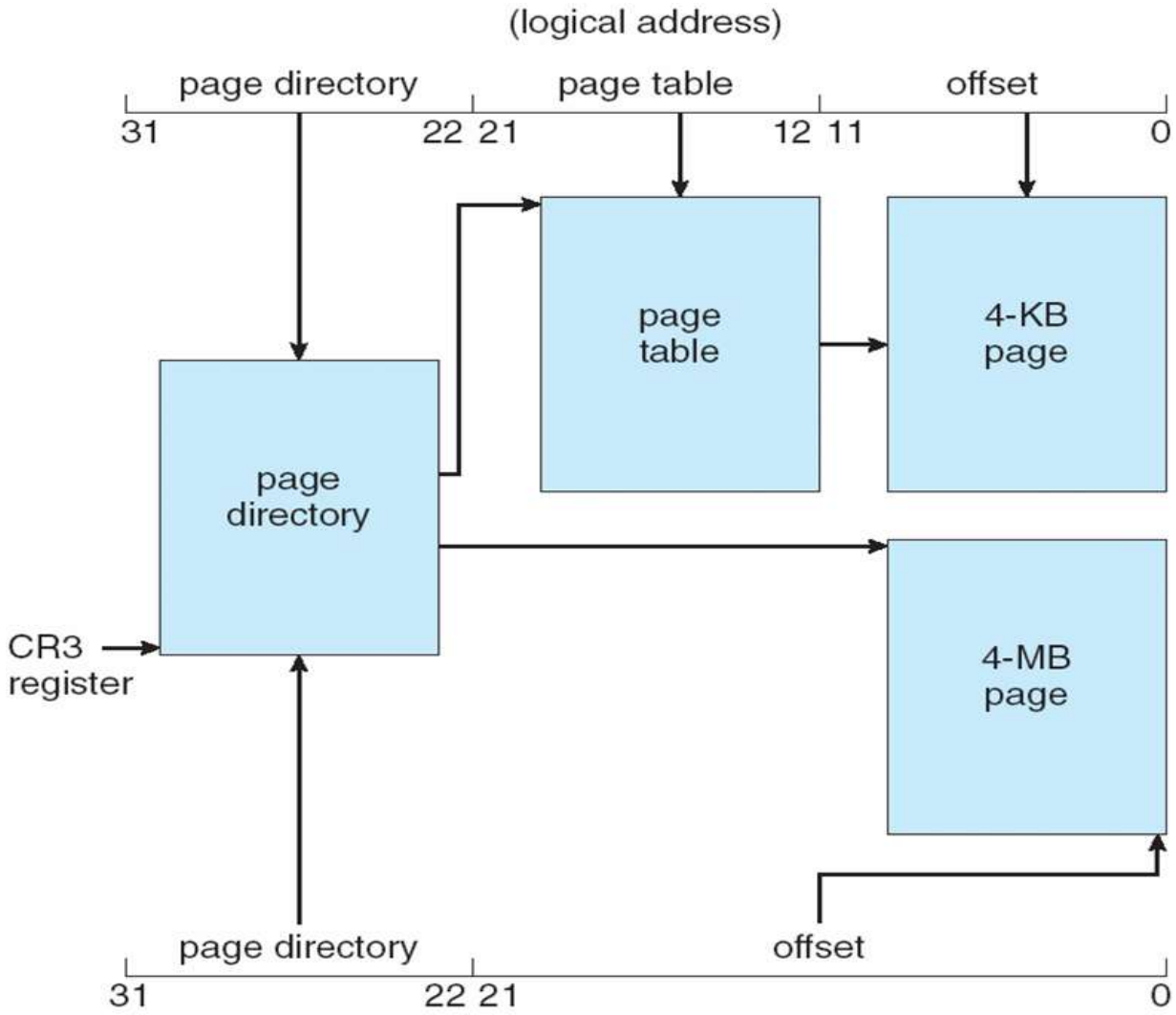




Segmentación IA-32



Arquitectura Paginación IA-32



Extensión de Direcciones IA-32

- El espacio de direcciones limitado por la arquitectura de 32 bits llevo a Intel a crear PAE – Page Address Extension
  - Paginación en un esquema de 3 niveles
  - Se toman bits de registro para apuntar al directorio de páginas