Principios a los Sistemas Operativos

Memoria Principal

Armando Arce, Tecnológico de Costa Rica, arce@itcr.ac.cr

Tecnológico de Costa Rica

Memoria Principal

El proceso administrador de memoria lleva un registro de las partes de la memoria que se están utilizando y aquellas que no.

- Este asigna espacio a los procesos cuando ellos lo solicitan.
- Así como también libera el espacio cuando los procesos terminan.
- Adicionalmente, el administrador de memoria realiza la administración de los intercambios entre memoria y disco.

Protección de memoria

En un sistema multiusuario, no es deseable que los procesos lean o escriban en la memoria perteneciente a otros usuarios.

- La protección del espacio de memoria se consigue haciendo que el hardware de la CPU compare todas las direcciones generadas en modo usuario con el contenido de dos registros: base y límite.

Protección de memoria

El *registro base* almacena la dirección de memoria física legal más pequeña, mientras que el *registro límite* especifica el tamaño del rango.

- Cualquier intento, en modo usuario, por acceder a alguna dirección fuera de ese rango provocará una interrupción (error fatal).

Protección de memoria

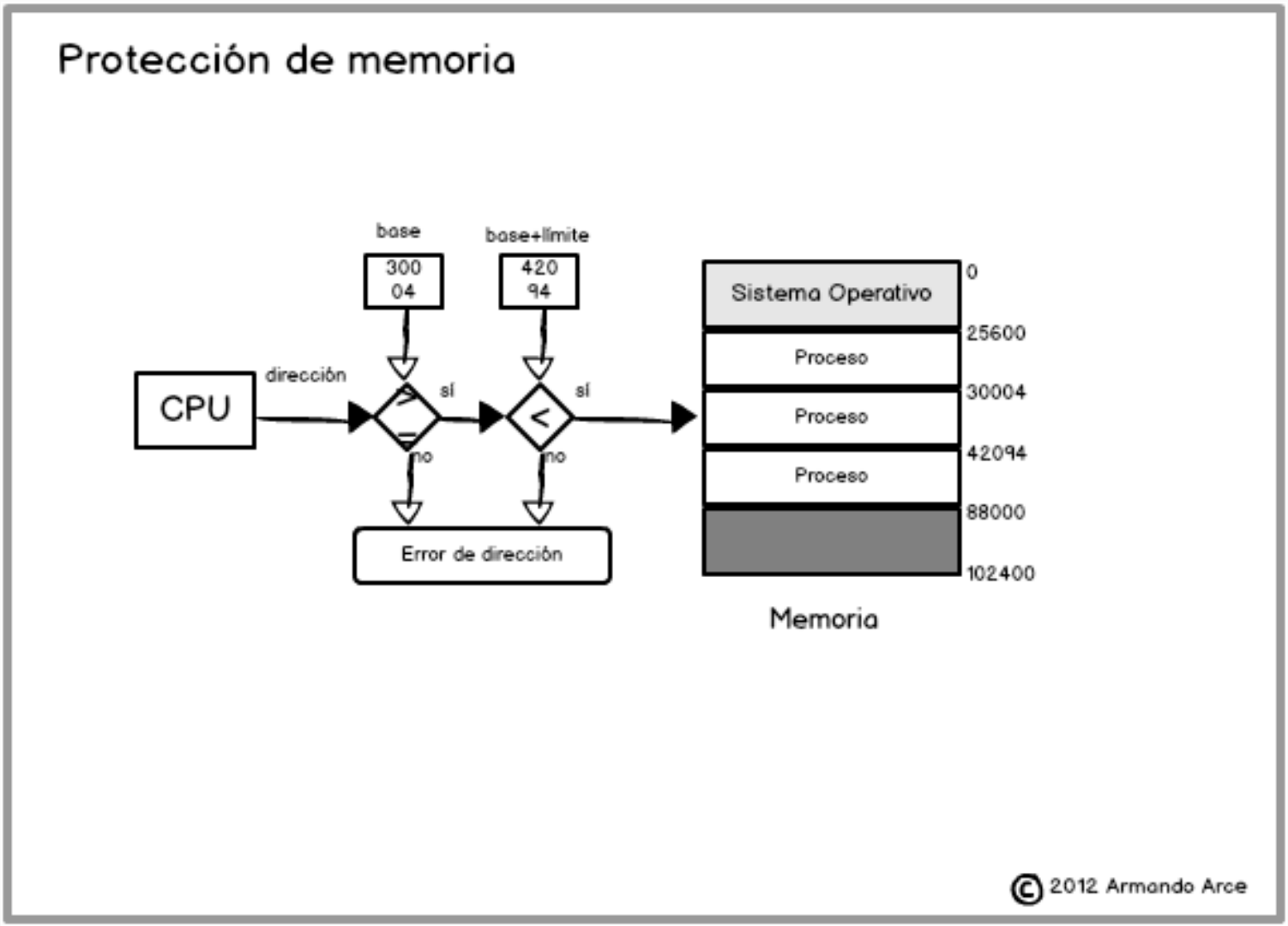

Figure 1:

Protección de memoria

Los registros base y límite sólo pueden ser cargados por el sistema operativo, que utiliza una instrucción privilegiada especial que evita que los programas de usuario cambien el contenido de esos registros.

- Por su parte el sistema operativo, que se ejecuta en modo kernel, tiene acceso no restringido a la memoria tanto del sistema operativo como de los usuarios.
- Una ventaja adicional del registro base es que el proceso se puede desplazar en la memoria mientras se encuentra en ejecución. Solo hay que modificar el registro base.

Reasignación de direcciones

Para poder ejecutarse, un programa deberá ser cargado en memoria y colocado dentro de un proceso.

- Normalmente, un *compilador* se encargará de resignar direcciones simbólicas a direcciones reubicables.
- El *cargador* se encargará, a su vez, de reasignar las direcciones reubicables a direcciones absolutas. Cada operación de resignación constituye una relación de un espacio de direcciones a otro.

Reasignación de direcciones

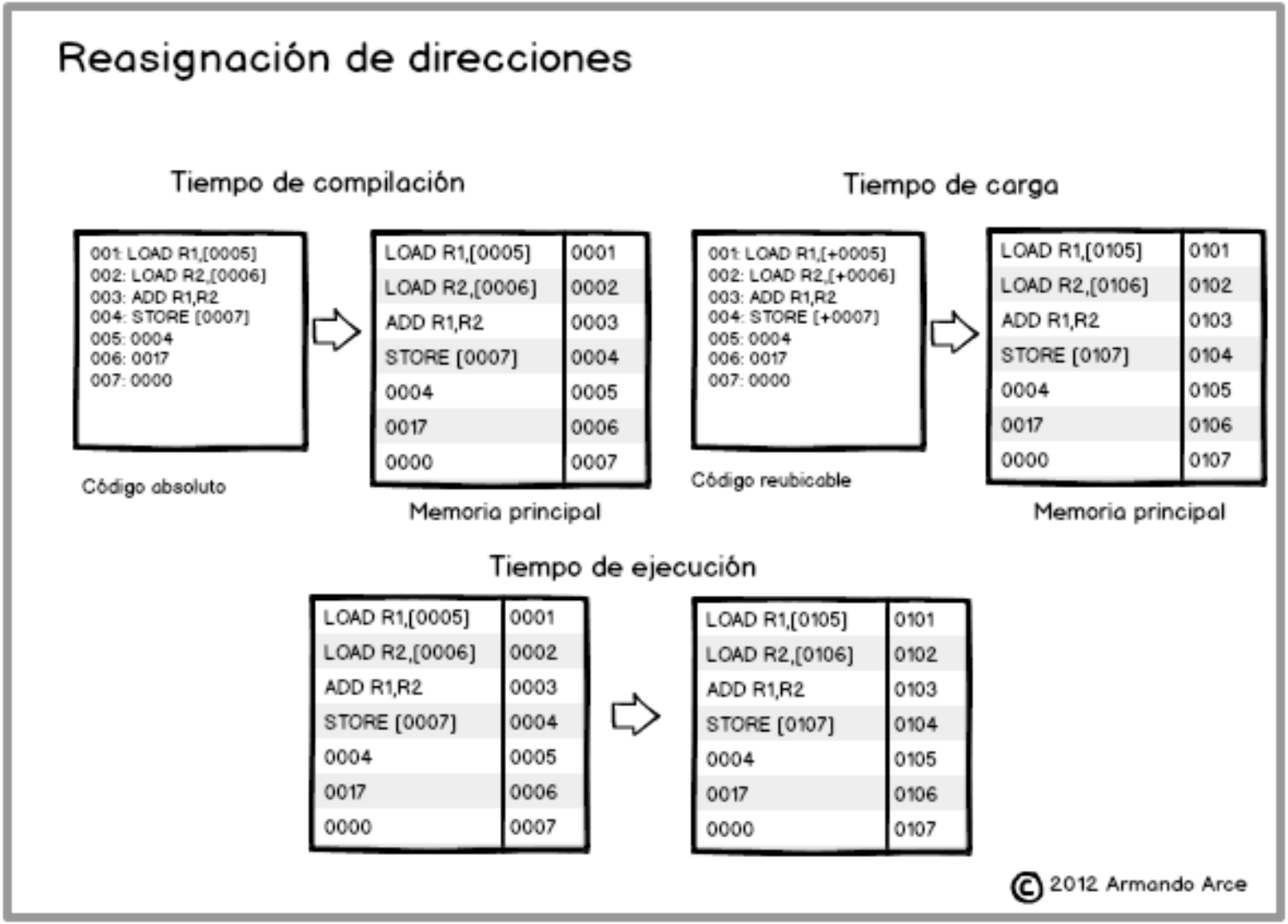

Figure 2:

Reasignación de direcciones

La resignación de instrucciones y datos a direcciones de memoria se realizar en cualquiera de los siguientes pasos:

- Tiempo de compilación: Si se sabe dónde va a residir el proceso en memoria, el compilador puede generar código absoluto. Si la ubicación inicial cambia, entonces es necesario recompilar ese código.

Reasignación de direcciones

- Tiempo de carga: El compilador también puede generar código reubicable. En este caso la reasignación se retarda hasta el momento de carga. Si cambia la dirección inicial, únicamente se debe volver a cargar el código para ubicarlo en su nueva posición.
- Tiempo de ejecución: Si el proceso puede desplazarse durante su ejecución desde un segmento de memoria a otro, entonces es necesario retardar la reasignación hasta el instante de la ejecución.

Espacios de direcciones lógico y físico

Una dirección generada por la CPU se denomina comúnmente *dirección lógica*, mientras que una dirección vista por la unidad de memoria se denomina *dirección física*.

Los métodos de reasignación en tiempo de compilación y en tiempo de carga generan direcciones lógicas y físicas idénticas. Sin embargo, el esquema de reasignación de direcciones en tiempo de ejecución hace que las direcciones lógica y física difieran. El este caso, usualmente decimos que la dirección lógica es una dirección virtual.

Espacios de direcciones lógico y físico

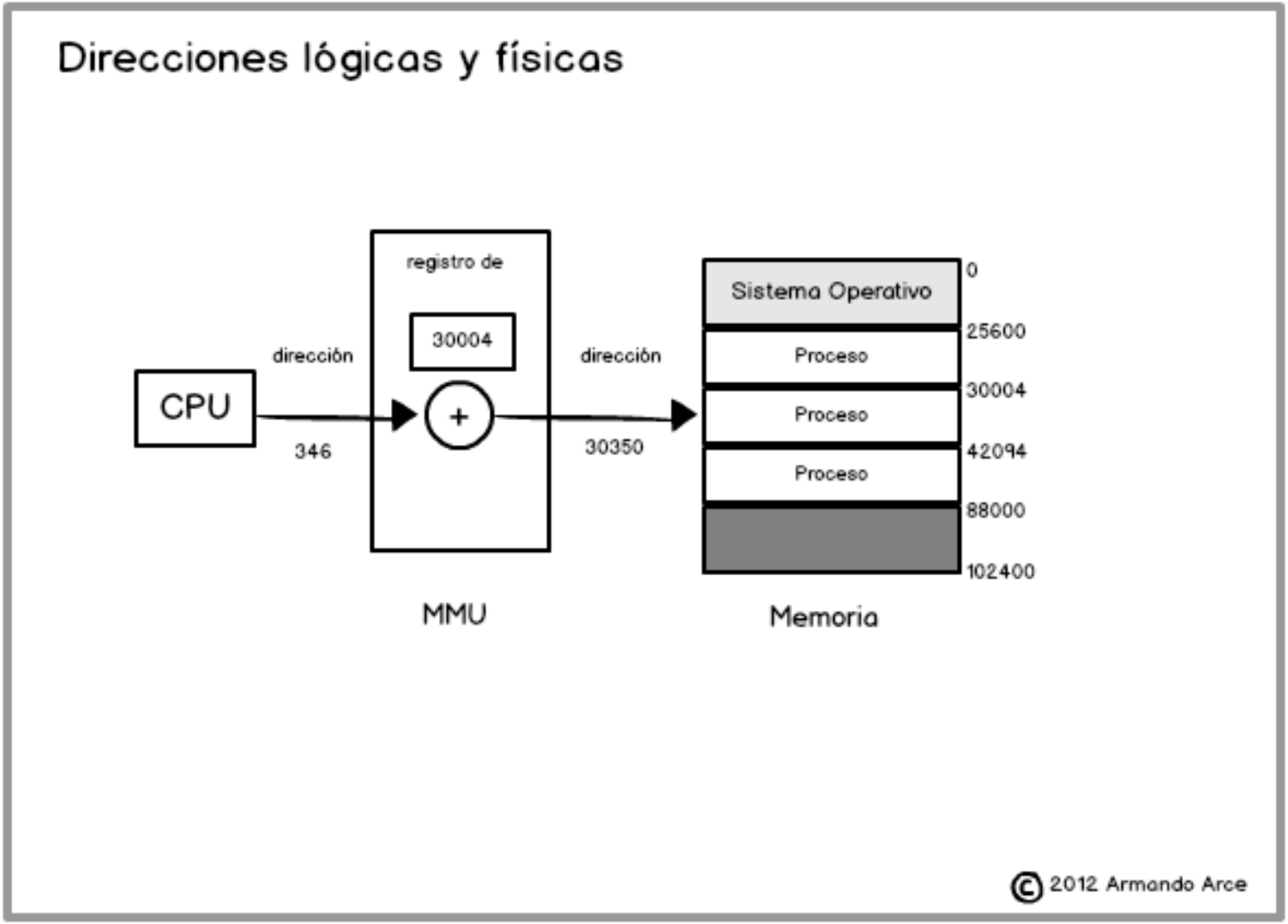

Figure 3:

Espacios de direcciones lógico y físico

La correspondencia entre direcciones virtuales y físicas en tiempo de ejecución es establecida por un dispositivo hardware que se denomina “unidad de administración de memoria” *MMU* (memory management unit). * Aquí al registro base se le llama *registro de reubicación*. * El valor contenido en este registro se suma a todas las direcciones generadas por el proceso de usuario en el momento de enviarlas a memoria.

Carga dinámica

Para obtener una mejor utilización del espacio de memoria, se puede utilizar un mecanismo de carga dinámica.

- Con la carga dinámica, una rutina no se carga hasta que se la invoca; todas las rutinas se mantienen en disco en un *formato de carga reubicable*.
- Según este método, el programa principal se carga en memoria y se ejecuta.

Carga dinámica

Cuando una rutina necesita llamar a otra rutina, la rutina que realiza la invocación comprueba primero si la otra ya ha sido cargada, si no es así, se invoca el cargador de montaje reubicable para que cargue en memoria la rutina deseada y para que actualice las tablas de direcciones del programa con el fin de reflejar este cambio. Después, se pasa el control a la rutina recién cargada.

Carga dinámica

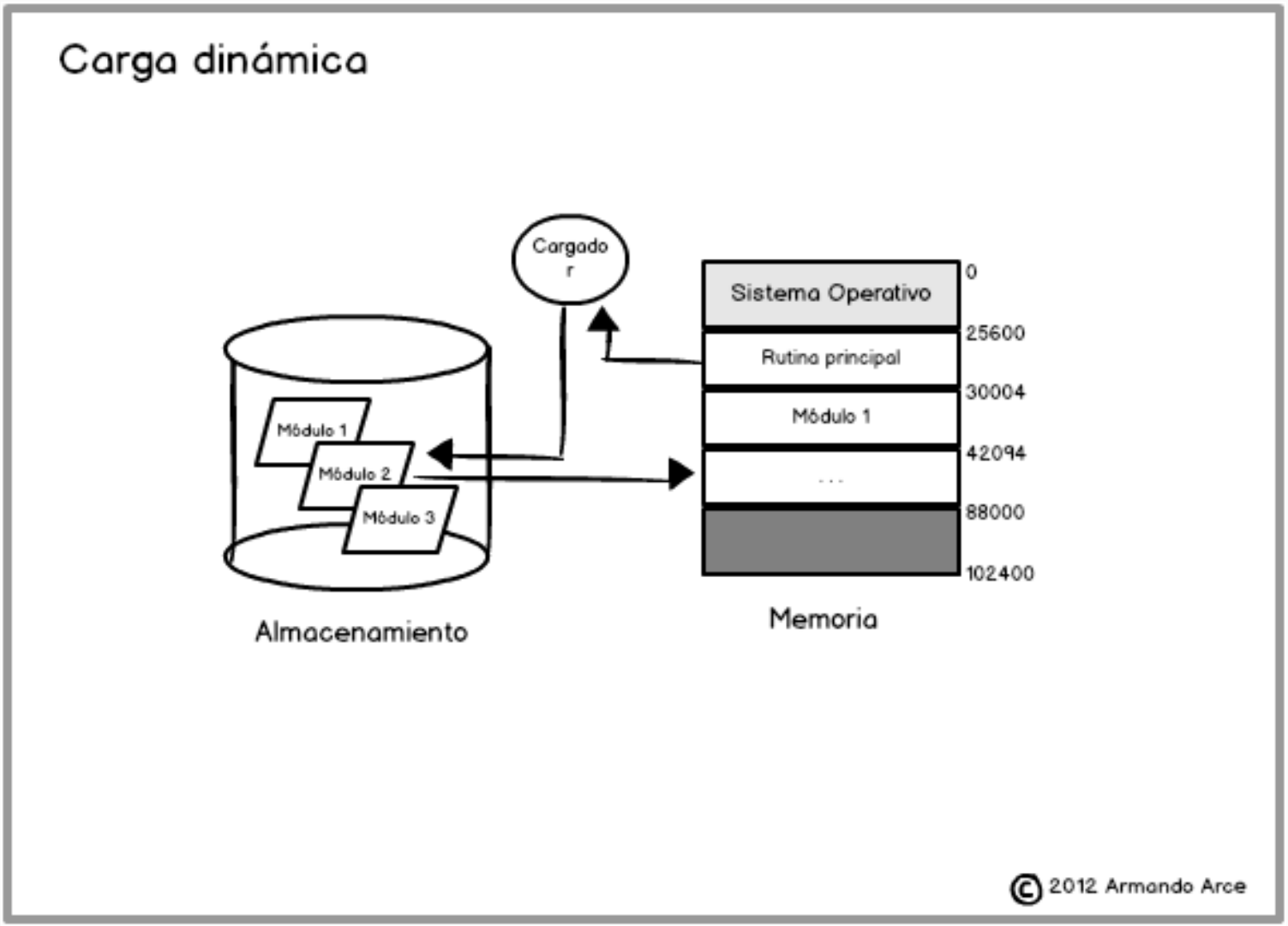

Figure 4:

Carga dinámica

El mecanismo de carga dinámica no requiere de ningún soporte especial por parte del sistema operativo.

- Es responsabilidad de los usuarios diseñar sus programas para poder aprovechar dicho método.

Montaje dinámico y bibliotecas compartidas

Algunos sistemas operativos sólo permiten el montaje estático, aquí las bibliotecas del sistema se tratan como cualquier módulo objeto y son integradas (incrustadas) dentro de la imagen binaria del programa.

- Este mecanismo hace que se desperdicie tanto espacio de disco como memoria principal.

Montaje dinámico y bibliotecas compartidas

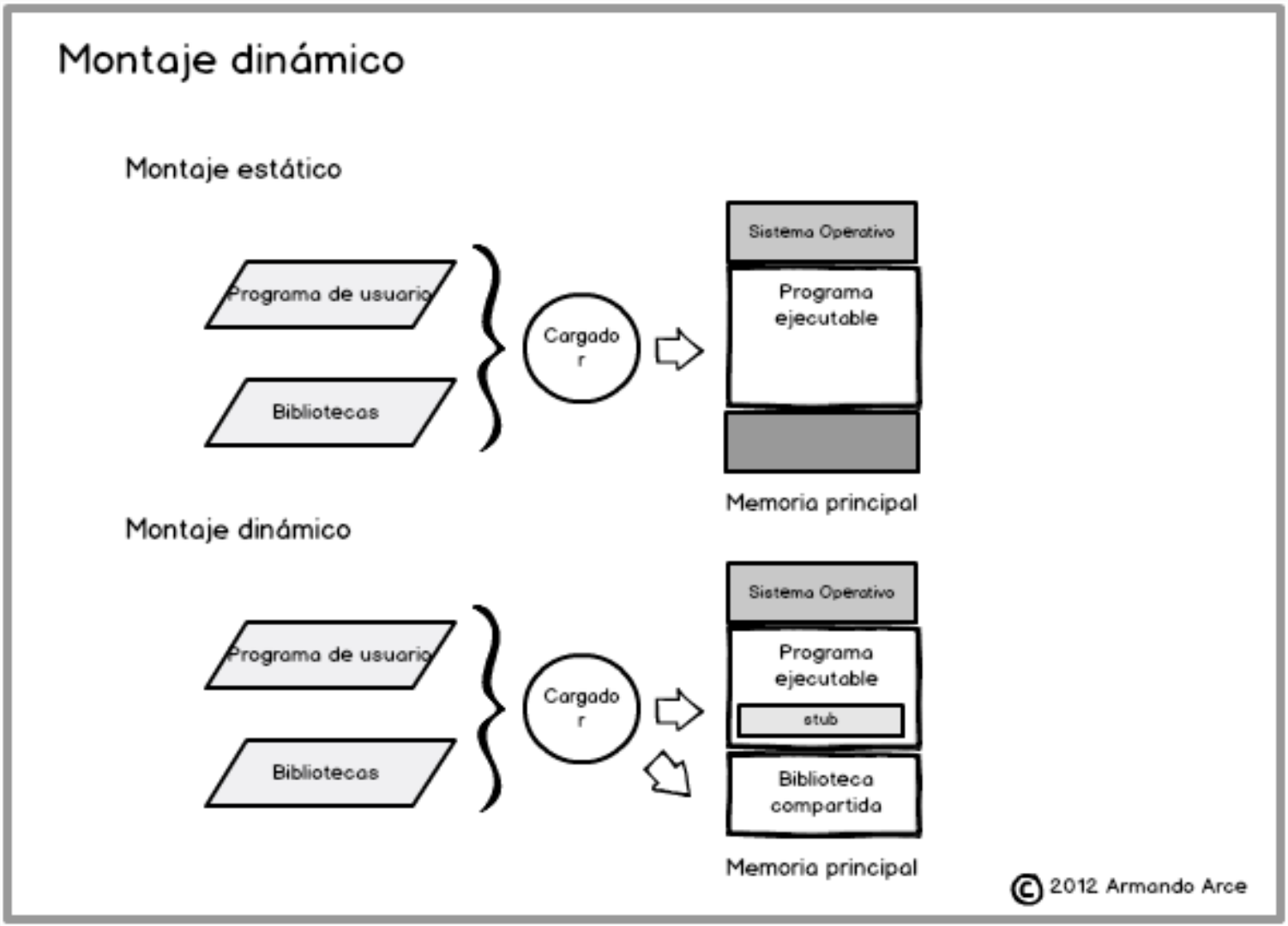

Figure 5:

Montaje dinámico y bibliotecas compartidas

Con el montaje dinámico, se incluye un *stub* dentro de la imagen binaria para cada referencia a una rutina de biblioteca.

- El *stub* es un pequeño fragmento de código que indica cómo localizar la rutina adecuada de alguna biblioteca residente en memoria o cómo cargar la biblioteca si esa rutina no está todavía presente.
- El *stub* se sustituye a sí mismo por la dirección de la rutina y ejecuta la rutina.
- La próxima vez que se ejecuta ese segmento de código, se ejecutará directamente la rutina de biblioteca, sin tener que realizar de nuevo el montaje dinámico.
- Así, todos los procesos que utilicen una determinada biblioteca sólo necesitan ejecutar una copia del código de la biblioteca.

Montaje dinámico y bibliotecas compartidas

Puede haber más de una versión de una biblioteca cargada en memoria y cada programa necesitará conocer información sobre la versión requerida para decidir qué copia de la biblioteca hay que utilizar.

Intercambio

Un proceso debe estar en memoria para ser ejecutado.

- Sin embargo, los procesos pueden esta intercambiados temporalmente, sacándolos de la memoria y almacenándolos en un almacén de respaldo y volviéndolos a llevar luego a memoria para continuar su ejecución.

Intercambio

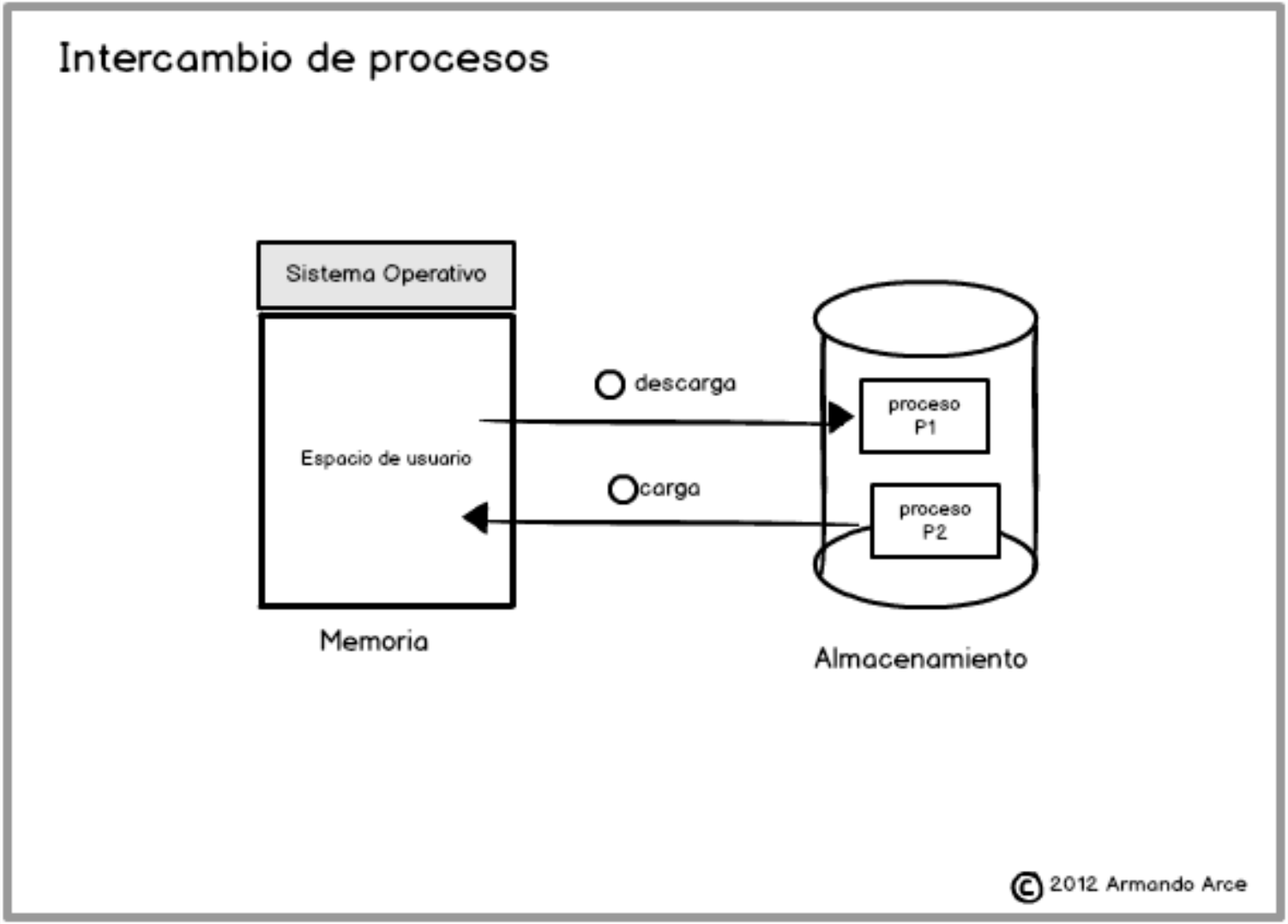

Figure 6:

Intercambio

Normalmente, un proceso descargado se volverá a cargar en el mismo espacio de memoria que ocupaba anteriormente.

- Esta restricción está dictada por el método de reasignación de las direcciones.
- Si la resignación se realiza en tiempo de ensamblado o de carga, entonces no resulta sencillo mover el proceso a una ubicación diferente.
- Sin embargo, si se está utilizando resignación en tiempo de ejecución sí que puede moverse el proceso a un espacio de memoria distinto, porque las direcciones físicas se calculan en tiempo de ejecución.

Intercambio

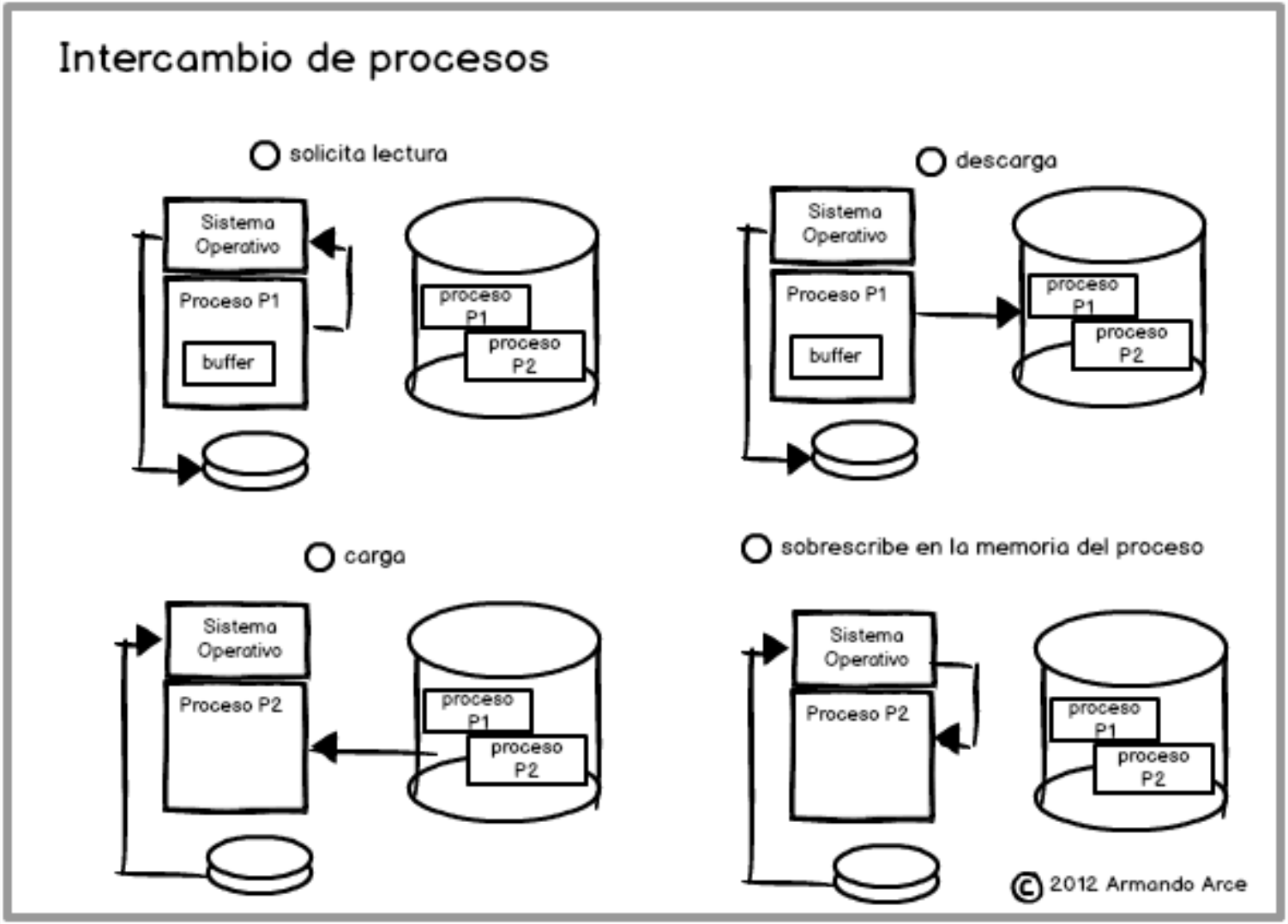

Figure 7:

Intercambio

El sistema mantiene una *cola de procesos preparados* que consistirá en todos los procesos cuyas imágenes de memoria se encuentren en el almacén de respaldo o en la memoria y estén listos para ejecutarse.

Intercambio

- Cada vez que el planificador de CPU decide ejecutar un proceso, llama al despachador, que mira a ver si el siguiente proceso de la cola se encuentra en memoria.
- Si no es así, y si no hay ninguna región de memoria libre, el despachador intercambia el proceso deseado por otro proceso que esté actualmente en memoria.
- A continuación, recarga los registros y transfiere el control al proceso seleccionado.

Intercambio

Antes de intercambiar un proceso, se debe asegurar que esté completamente inactivo.

- En este sentido, es necesario prestar atención a todas las operaciones de E/S pendientes.
- Un proceso puede estar esperando por una operación de E/S en el momento en que se realiza el intercambio con el fin de liberar memoria.

Intercambio

En este caso, si la E/S está accediendo asíncronamente a la memoria de usuario donde residen los búferes de E/S, el proceso no podrá ser intercambiado pues otro proceso podría intentar utilizar la memoria que ahora pertenece al nuevo proceso.

- Hay dos soluciones principales a este problema: no descargar nunca un proceso que tenga actividades de E/S pendientes o ejecutar las operaciones E/S únicamente con búferes del sistema operativo.

Asignación de memoria contigua

Es necesario afinar las distintas partes de la memoria principal de la forma más eficiente posible.

- La memoria está usualmente dividida en dos particiones: una para el sistema operativo residente y otra para los procesos de usuario.

Asignación de memoria contigua

Normalmente, se quiere tener varios procesos de usuario residentes en memoria del mismo tiempo.

- Por tanto, se debe considerar cómo asignar la memoria disponible a los procesos que se encuentran en la cola de entrada, esperando a ser cargados en memoria.
- En este esquema de asignación continua de memoria, cada proceso está contenido en una única sección contigua de memoria.

Asignación de memoria

Uno de los métodos más simples para asignar la memoria consiste en dividirla en varias particiones de tamaño fijo.

- Cada partición puede contener exactamente un proceso, de modo que el grado de multiprogramación estará limitado por el número de particiones disponibles.
- En este método de particiones múltiples, cuando una partición está libre, se selecciona un proceso de la cola de entrada y se carga en dicha partición.
- Cuando el proceso termina, la partición pasa a estar disponible para otro proceso.

Uso de particiones fijas

La memoria se puede partir en n partes, para mantener n procesos en memoria.

- La partición puede ser hecha en forma manual por el operador.
- Cuando llega un trabajo, se le coloca en la cola de entrada de la parte de tamaño mas pequeño de forma que lo pueda contener.
- El problema es que una partición grande puede estar vacía pero la cola de una partición pequeña estar completamente ocupada.

Uso de particiones fijas

Otro tipo de organización mantiene una sola cola. Cada vez que se libere una partición se carga la tarea mas cercana al frente de la cola que se ajuste a dicha partición.

- Otra estrategia busca en toda la cola el trabajo mas grande que se ajuste a la partición recién liberada, para no desperdiciar particiones grandes con trabajos pequeños.
- Sin embargo, esto discrimina a las tareas pequeñas y no les da importancia. Las tareas pequeñas pueden ser interactivas y merecer el mejor servicio y no el peor.

Uso de particiones fijas

Una solución al problema anterior es tener siempre una pequeña partición, que permita la ejecución de tareas pequeñas.

- Otra solución sería crear una regla para que un trabajo elegible para su ejecución no sea excluido mas de k veces.
- Cada vez que se le excluya, obtiene un punto. Cuando adquiera k puntos, no se le excluye mas.

Uso de particiones variables

Con el esquema de particiones variables el número, posición y tamaño de las particiones varía en forma dinámica al crearse o intercambiarse procesos.

- Esto mejora el uso de la memoria pero también hace mas compleja la asignación y reasignación de la memoria, así como complica el mantener un registro de esto.

Uso de particiones variables

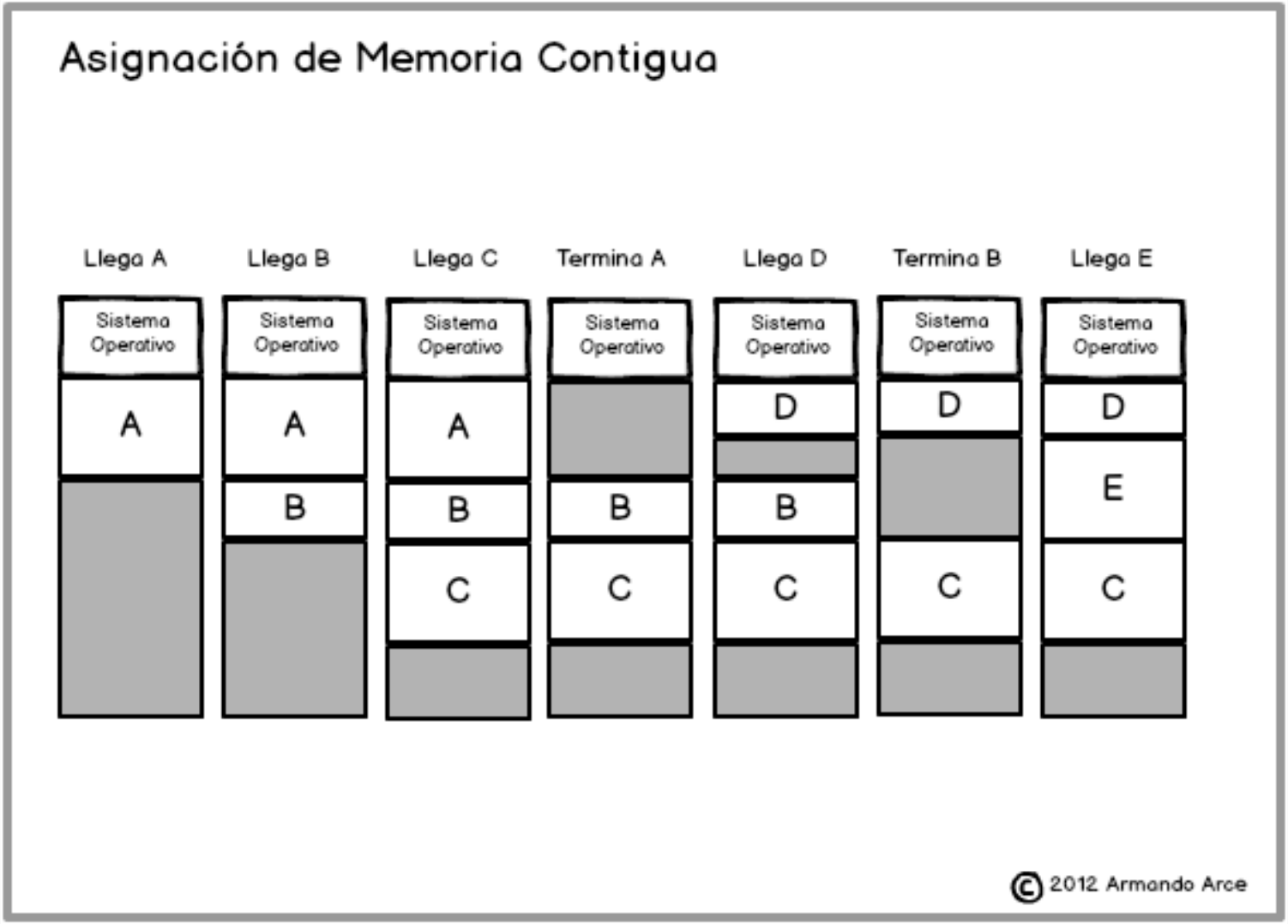

Figure 8:

Uso de particiones variables

En el esquema de particiones variables, el sistema operativo mantiene una tabla que indica qué partes de la memoria están disponibles y cuáles están ocupadas.

- Inicialmente, toda la memoria está disponible para los procesos de usuario y se considera como un único bloque de gran tamaño de memoria disponible, al que se denomina agujero (hueco).
- Cuando llega un proceso y necesita memoria, se busca un hueco lo suficientemente grande como para albergar este proceso.
- Si se encuentra, sólo se asigna la memoria justa necesaria, manteniendo el resto de la memoria disponible para satisfacer futuras solicitudes.

Uso de particiones variables

En cualquier momento determinado, se tendrá una lista de tamaños de bloque disponibles y una cola de entrada de procesos.

- El sistema operativo puede ordenar la cola de entrada de acuerdo con algún algoritmo de planificación, asignándose memoria a los procesos hasta que finalmente, los requisitos de memoria del siguiente proceso ya no puedan satisfacerse, es decir, hasta que no haya ningún bloque de memoria (hueco) disponible que sea lo suficientemente grande como para albergar al siguiente proceso.

Uso de particiones variables

El sistema operativo puede entonces esperar hasta que haya libre un bloque de memoria suficientemente grande, o puede examinar el resto de la cola de entrada para ver si pueden satisfacerse los requisitos de memoria de algún otro proceso, que necesite un bloque de memoria menor.

Uso de particiones variables

Si el nuevo agujero es adyacente a otros huecos, se combinan esos agujeros adyacentes para formar otros de mayor tamaño. A este mecanismo se le conoce como *fusión de huecos*.

Uso de particiones variables

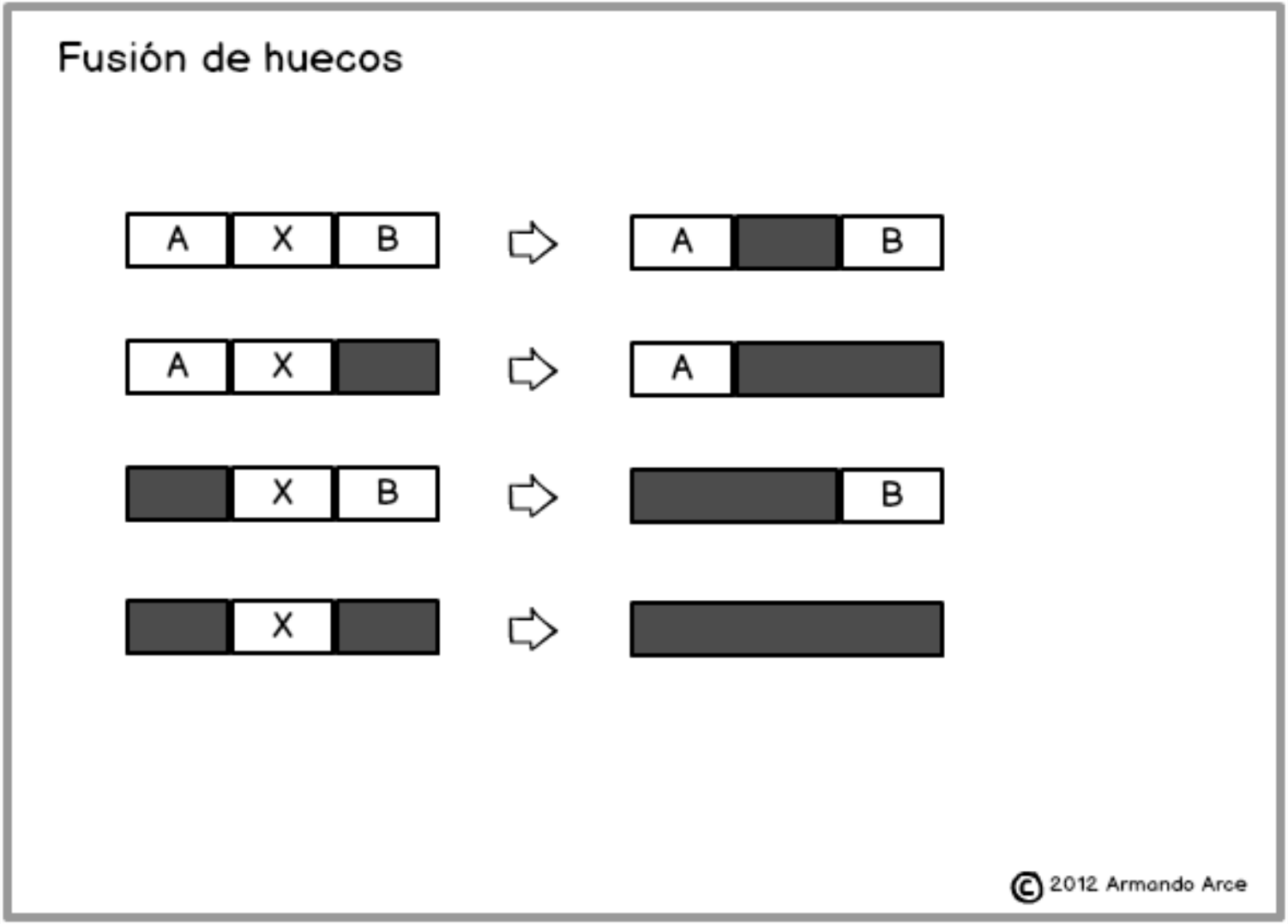

Figure 9:

Asignación dinámica de espacio

El problema general de *asignación dinámica de espacio* de almacenamiento se ocupa de cómo satisfacer una solicitud de tamaño n a partir de una lista de huecos libres. Hay muchas soluciones a este problema, entre ellas:

- Primer ajuste: Se asigna el primer hueco que sea lo suficientemente grande. La exploración inicia desde el principio del conjunto de agujeros y se detiene cuando se encuentra un hueco libre lo suficientemente grande.

Asignación dinámica de espacio

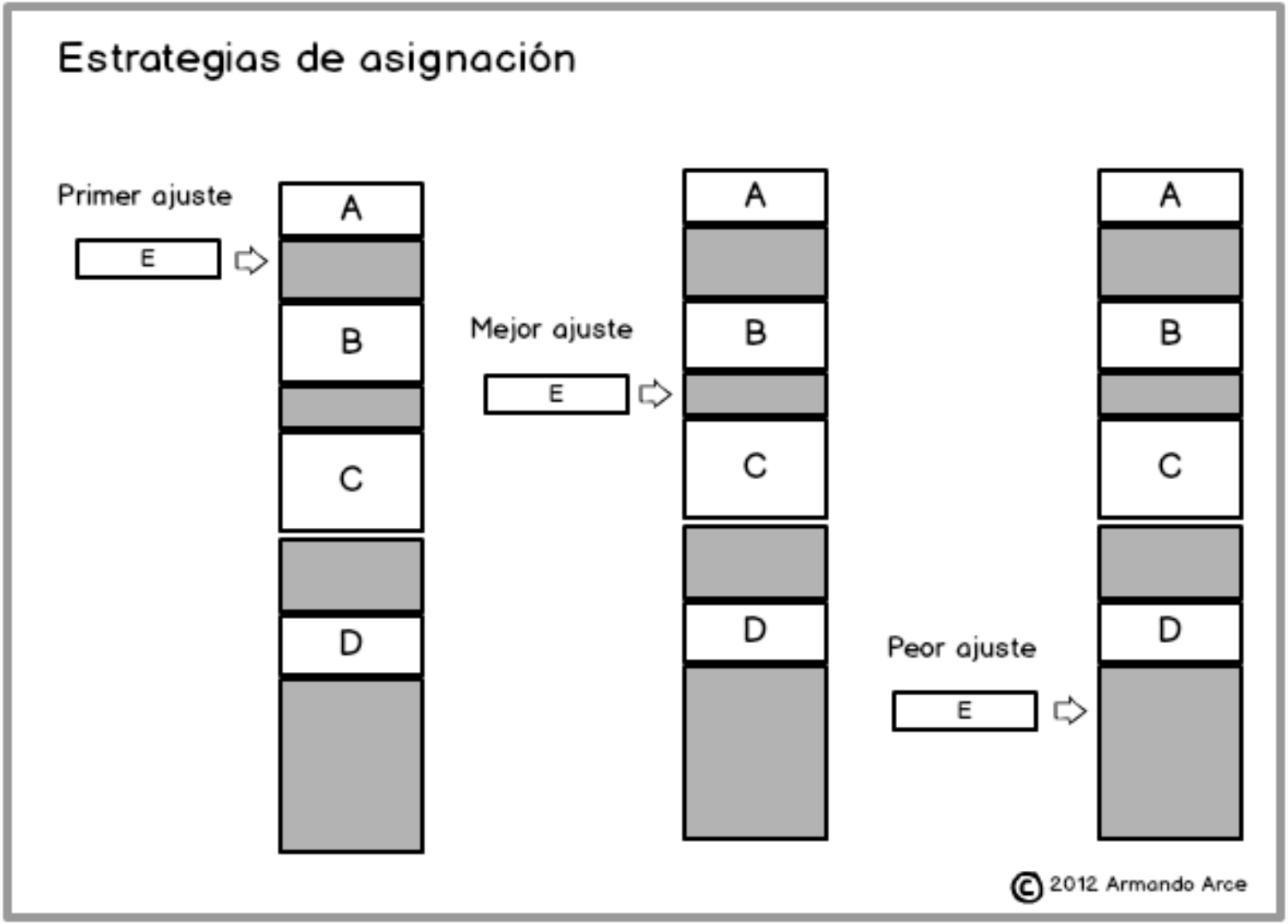

Figure 10:

Asignación dinámica de espacio

- Mejor ajuste: Se asigna el hueco más pequeño que tenga el tamaño suficiente. Se debe explorar la lista completa, a menos que ésta esté ordenada por tamaño.
- Peor ajuste: Se asigna el hueco de mayor tamaño. También, se debe explorar toda la lista, a menos que esté ordenada por tamaño.
- Siguiente ajuste: Se asigna el siguiente hueco, desde el punto en que terminó la exploración anterior, que sea lo suficientemente grande.

Asignación dinámica de espacio

En general, las estrategias del primer ajuste y mejor ajuste son mejores que la del peor ajuste en términos de tiempo y espacio de almacenamiento.

- Normalmente la estrategia del primer ajuste y el siguiente ajuste son las más rápidas de implementar, aunque el siguiente ajuste produce menos fragmentación.

Fragmentación

Tanto la estrategia de primer ajuste como de mejor ajuste sufren el problema denominado *fragmentación externa* que genera desperdicio de memoria.

- A medida que se cargan procesos en memoria y se eliminan, el espacio de memoria libre se descompone en una serie de fragmentos de pequeño tamaño no contiguos.
- El problema de la fragmentación externa aparece cuando hay un espacio de memoria total suficiente para satisfacer una solicitud, pero esos espacios disponibles no son contiguos.

Fragmentación

La fragmentación de memoria puede ser también *interna*, además de externa. Esto se da cuando se asignan bloques de memoria de un tamaño mayor al requerido por el proceso.

- De esta forma el espacio se encuentra asignado pero no es utilizado, lo que también representa desperdicio.
- La técnica general para evitar este problema consiste en descomponer la memoria física en bloques de un tamaño fijo adecuado (ni demasiado pequeño, ni demasiado grande) y asignar la memoria en unidades basadas en ese tamaño de bloque.

Fragmentación

Una solución al problema de fragmentación externa consiste en la *compactación*.

- El objetivo es mover el contenido de la memoria con el fin de situar toda la memoria libre de manera contigua, para formar un único bloque de gran tamaño.
- Sin embargo, la compactación sólo es posible si la reubicación es dinámica y se lleva a cabo en tiempo de ejecución.
- En general, la compactación de memoria no se utiliza porque es un esquema muy caro de implementar.

Fragmentación

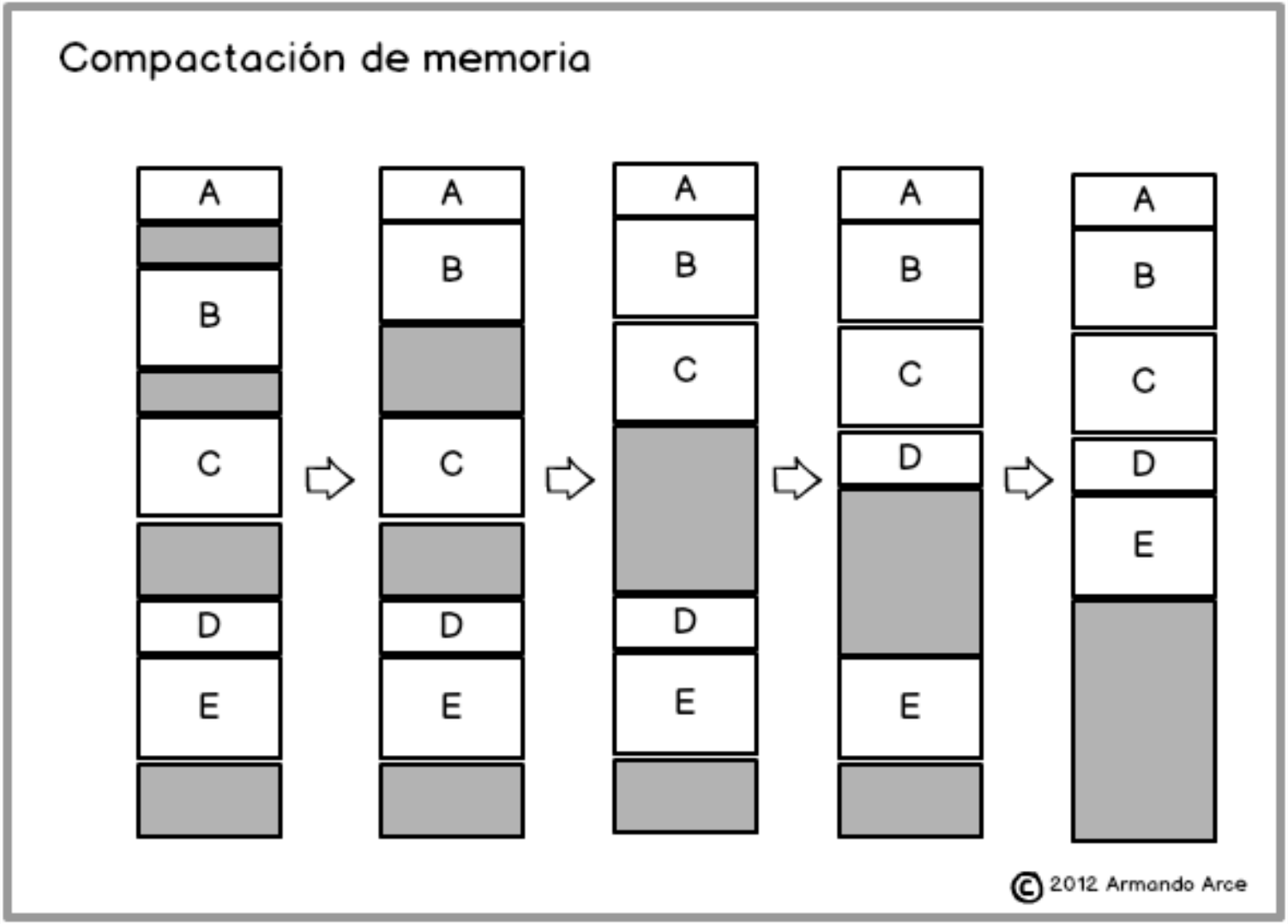

Figure 11:

Paginación

La *paginación* es un esquema de administración de memoria que permite que el espacio de direcciones físicas de un proceso no sea contiguo.

- Debido a sus ventajas, la mataría de los sistemas operativos utilizan comúnmente mecanismos de paginación de diversos tipos.
- Normalmente, el soporte para la paginación se administra mediante hardware.

Método básico

El método básico para implementar la paginación implica descomponer la memoria física en una serie de bloques de tamaño fijo denominados *marcos* y descomponer la memoria lógica en bloques del mismo tamaño denominados *páginas*.

- Cuando hay que ejecutar un proceso, sus páginas se cargan desde el almacén de respaldo en los marcos de memoria disponibles.
- El almacén de respaldo está dividido en bloques de tamaño fijo que tienen el mismo tamaño que los marcos de memoria.

Método básico

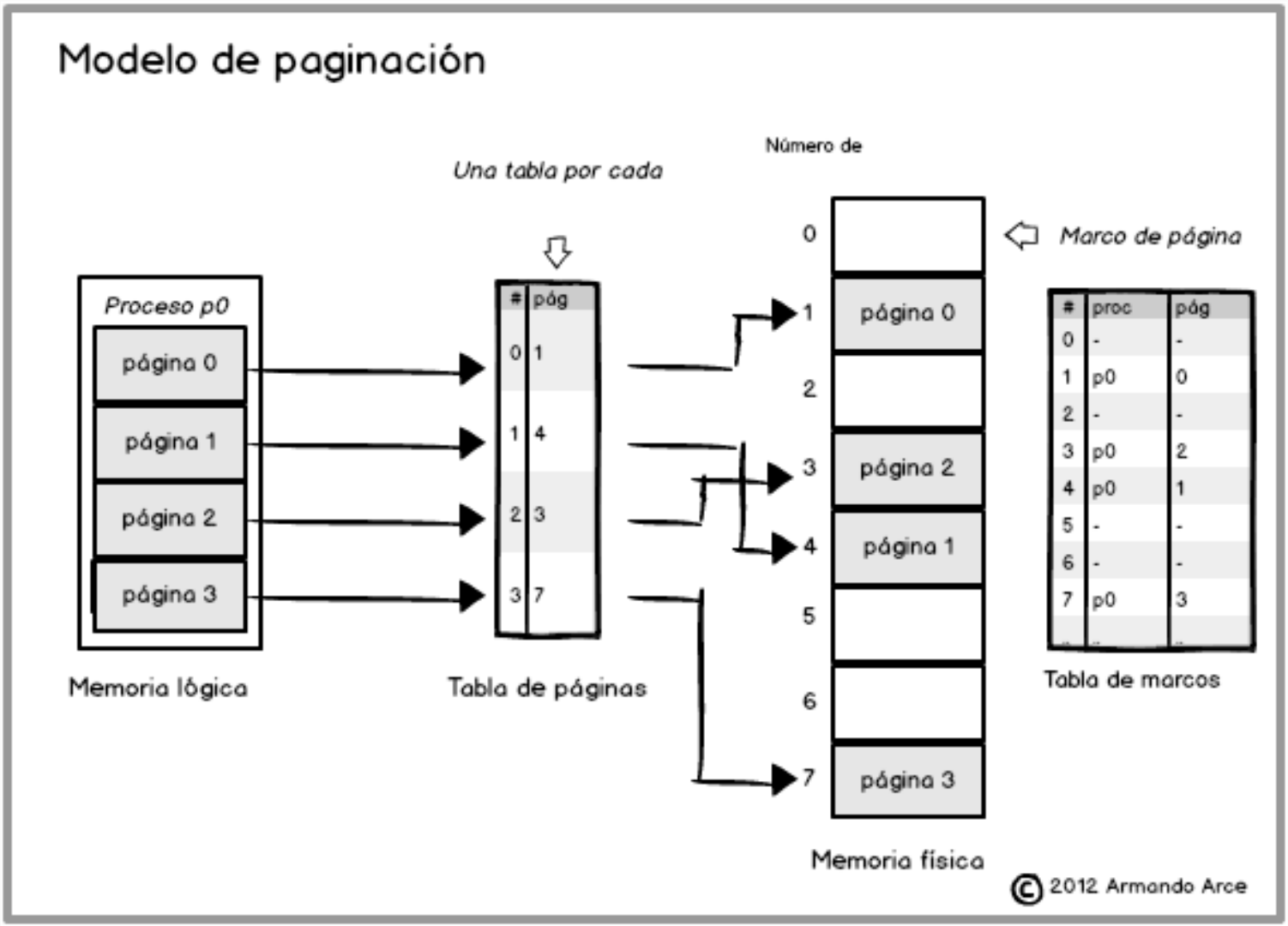

Figure 12:

Método básico

Toda dirección generada por la CPU está dividida en dos partes: *número de página* y *un desplazamiento de página*.

- El número de página es el índice para una tabla de páginas. La tabla de página contiene la dirección base en memoria física de cada página.
- La dirección base se combina con el desplazamiento para obtener la dirección de memoria física que se envía a la unidad de memoria.

Método básico

El tamaño de página (al igual que el tamaño de marco) está definido por el hardware.

- El tamaño de página es normalmente una potencia de 2, variando entre 512 bytes y 16 MB por página, dependiendo de la arquitectura de la computadora.

Método básico

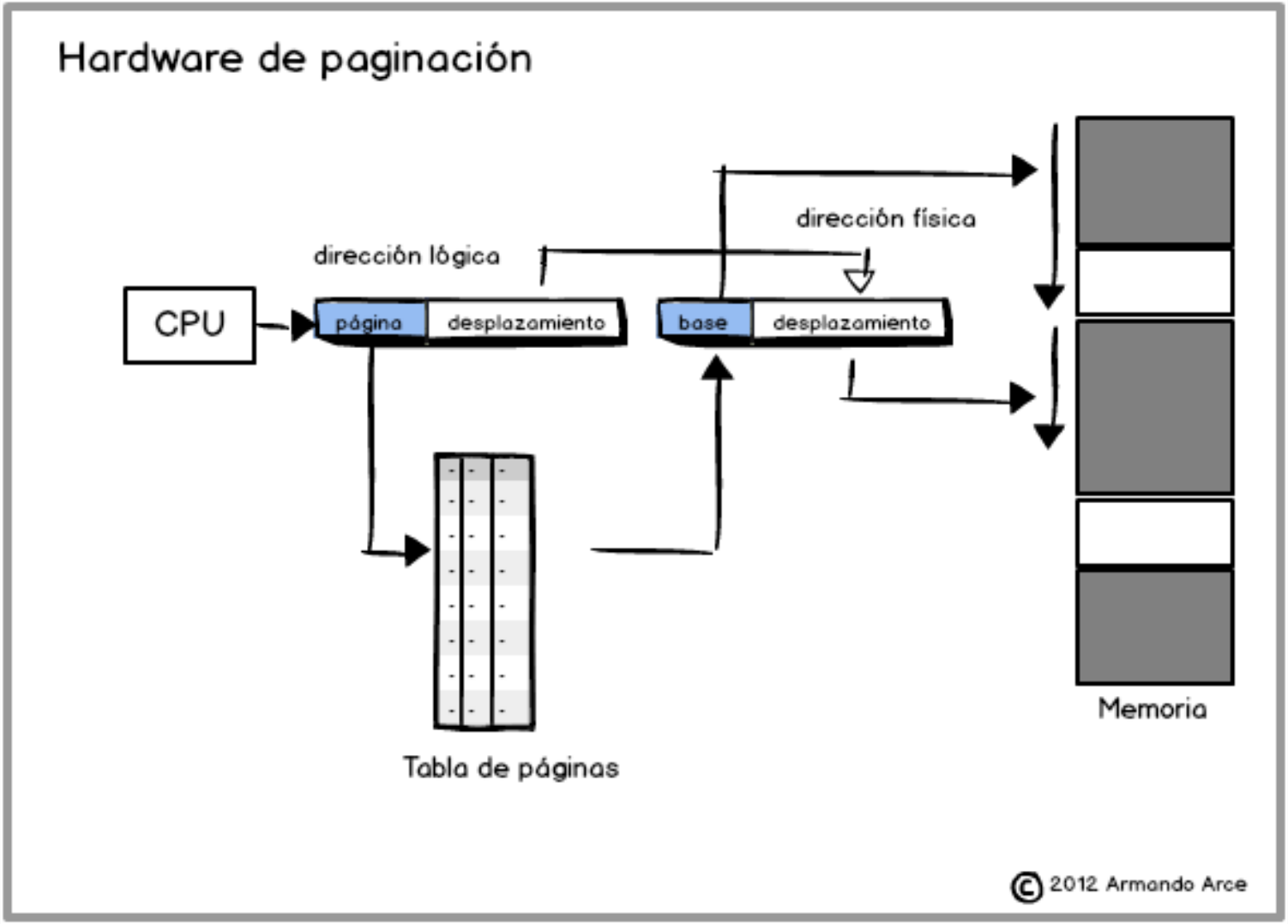

Figure 13:

Método básico

El uso del esquema de paginación, no produce fragmentación externa: todos los marcos libres pueden ser asignados a un proceso que los necesite.

- Sin embargo, sí se puede presentar fragmentación interna, pues los requisitos de memoria de un proceso pueden no coincidir exactamente con las fronteras de página y por tanto el último marco puede no estar completamente lleno. Este hecho sugiere que conviene utilizar tamaños de página pequeños.

Método básico

- Sin embargo, se necesita dedicar más recursos a la administración de cada una de las entradas en la tabla de páginas y estos recursos adicionales se reducen a medida que se incrementa el tamaño de página. Además, las operaciones de E/S son más eficientes cuanto mayor sea el número de datos transferidos.

Método básico

El sistema operativo debe contar con una estructura de datos llamada *tabla de marcos* que le permite administrar la memoria física: qué marcos han sido asignados, qué marcos están disponibles, cuál es el número total de marcos, etc.

- Esta tabla tiene una entrada por cada marco físico que indica si está libre o asignado, y en caso de estar asignado, a qué página de qué proceso ha sido asignado.

Soporte de hardware

La mayoría de sistemas operativos asignan una tabla de páginas para cada proceso, almacenándose un puntero a la tabla de páginas, junto con los otros valores de los registros en el bloque de control de proceso (PCB).

- La tabla de páginas se mantiene en memoria principal, utilizándose un *registro base de la tabla de páginas* (PTBR, page-table base register) para apuntar a la tabla de páginas.
- Para cambiar las tablas de páginas, sólo hace falta cambiar este único registro, reduciéndose así sustancialmente el tiempo de cambio de contexto.
- Sin embargo, nótese que son necesarios dos accesos a memoria (uno a la tabla y otro a la dirección real) para acceder a cualquier dato en memoria, lo que provoca un retardo intolerable.

Soporte de hardware

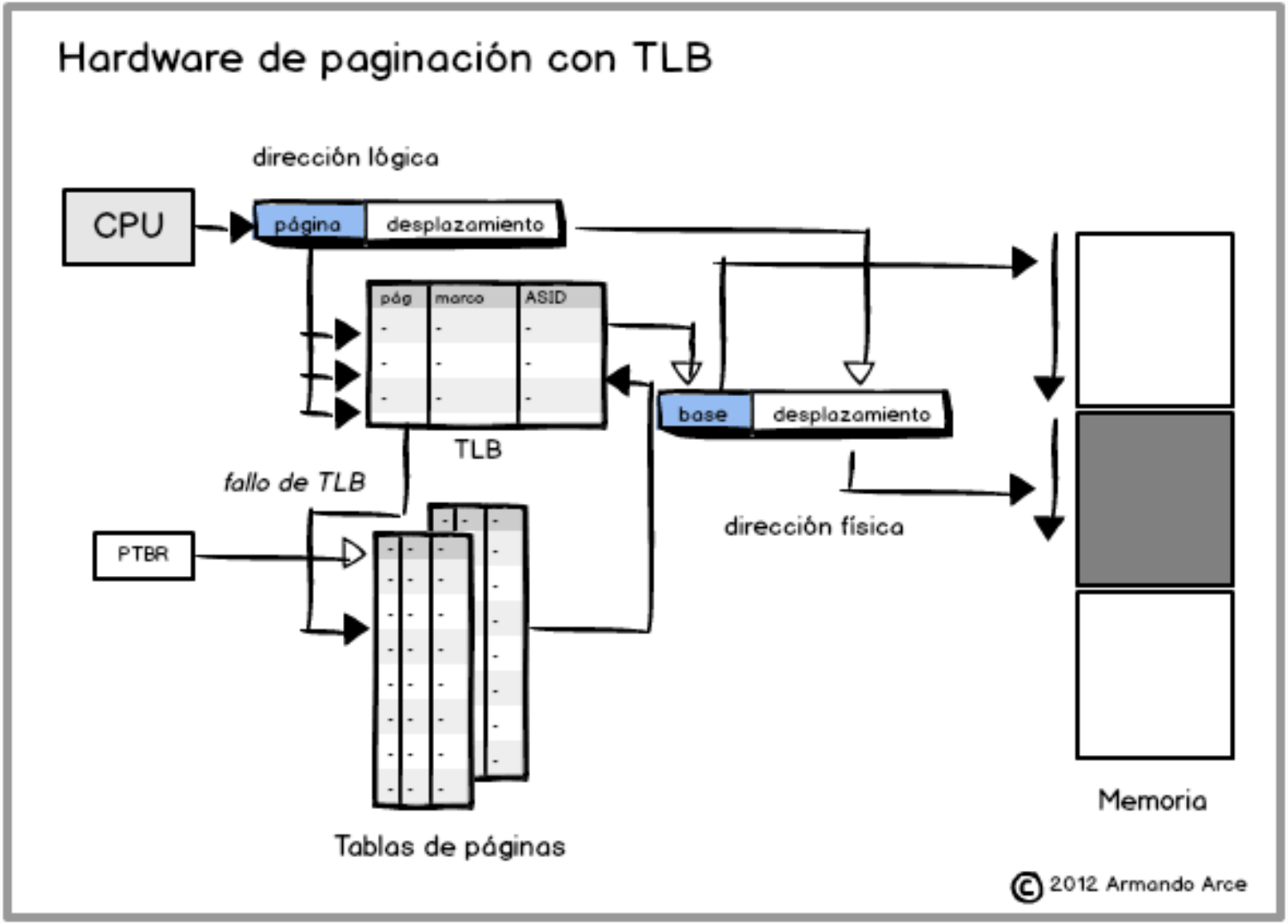

Figure 14:

Soporte de hardware

La solución al problema anterior consiste en utilizar un caché hardware denominado *búfer de consulta de traducción* (TLB, translation look-aside buffer) con capacidad de acceso asociativo de alta velocidad.

- Cuando se le presenta un elemento al TLB éste lo compara contra todas sus entradas en paralelo de forma muy rápida, si se encuentra el elemento se devuelve el correspondiente valor asociado.
- Generalmente este hardware es caro y la cantidad de entradas suele estar comprendidas entre 64 y 1024.

Soporte de hardware

El TLB trabaja de forma conjunta con las tablas de página de forma que al requerir una entrada de la tabla de página ésta es copiada en el TLB la primera vez.

- Posteriormente, dicha entrada será leída directamente del TLB y no será necesario acceder a la tabla de página.
- Cuando la entrada requerida no se encuentra en el TLB se le llama un *fallo de TLB*.
- Si todas las entradas del TLB están ocupadas hay que utilizar algún algoritmo de sustitución.

Soporte de hardware

En algunos TLBs se utiliza un *identificador del espacio de direcciones* (ASID, address-space identifier) en cada entrada del TBL.

- El identificador ASID identifica la entrada con el proceso asociado.
- De esta forma antes de recuperar una entrada del TLB se debe verificar que el ASID coincide con el identificador del proceso que accede a la dirección.
- De no existir el campo ASID en el TLB, será necesario invalidar todas las entradas cada vez que se produce un cambio de contexto.

Protección

La protección de memoria en un entorno paginado se consigue mediante una serie de bits de protección asociados a cada marco.

- Normalmente, estos bits se mantienen en la tabla de páginas. Un bit de la tabla de páginas puede indicar que la página es de lectura, escritura o ejecución.
- También se incluye el bit válido/inválido para saber si la página está presente en memoria.

Protección

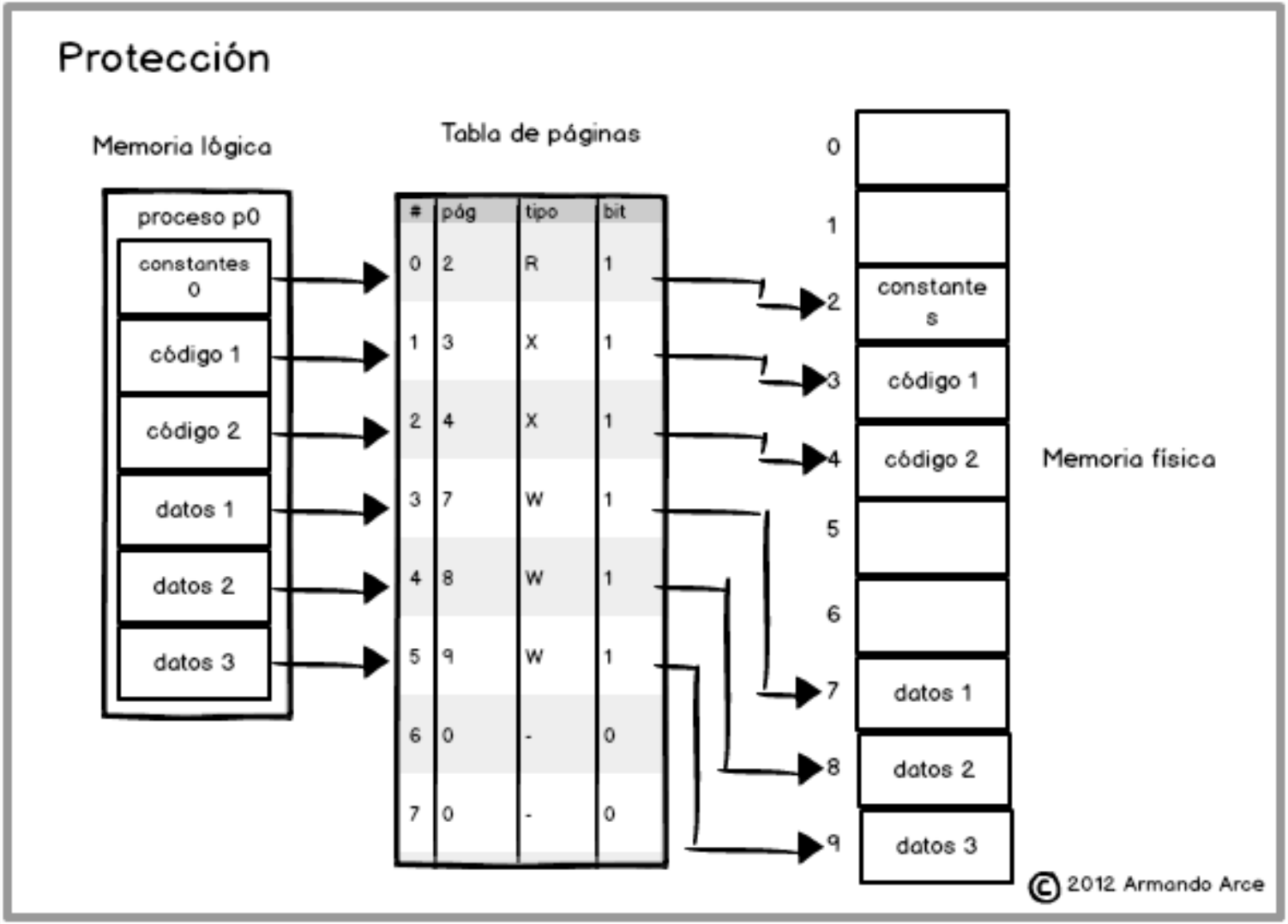

Figure 15:

Protección

Todo intento de leer, escribir o ejecutar instrucciones que se encuentren en páginas cuyos bits de protección no lo permitan provocarán una interrupción hardware del sistema (o una violación de protección de memoria).

- De igual forma al realizarse el acceso a una dirección que se encuentra en una página que tiene apagado el bit válido/inválido, se producida también una interrupción del sistema (referencia de página inválida).

Páginas compartidas

Una ventaja de la paginación es la posibilidad de compartir código común.

- Se pueden compartir páginas de código reentrante (código que no cambia durante la ejecución).
- Algunos sistemas implementan la memoria compartida mediante páginas compartidas.

Páginas compartidas

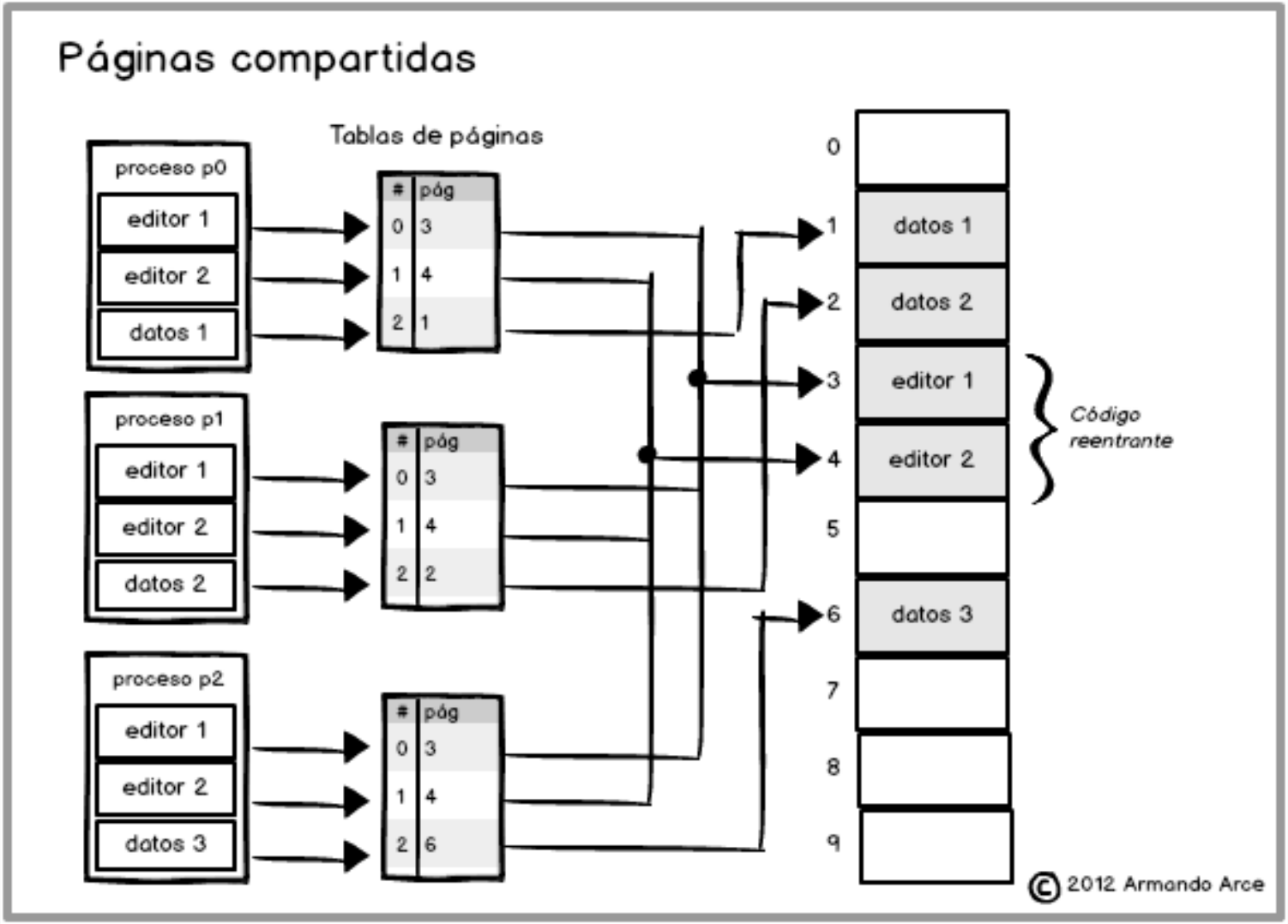

Figure 16:

Páginas compartidas

El código reentrante es código que no se auto-modifica; nunca cambia durante la ejecución.

- De esta forma, dos o más procesos pueden ejecutar el mismo código al mismo tiempo.
- Capa proceso tendrá su propia copia de los registros y del almacenamiento de datos, para albergar los datos requeridos para la ejecución del proceso.
- Sólo es necesario mantener en la memoria física una copia de las páginas compartidas.
- La tabla de páginas de cada usuario se corresponderá con la misma copia física de estas páginas compartidas, pero las páginas de datos se harán corresponder con marcos diferentes.

Estructura de la tabla de páginas

Existen diferentes formas de estructurar la tabla de páginas.

- En general, los diferentes esquemas intentan minimizar la cantidad de espacio de memoria que requiere la tabla de página.
- Otro objetivo consiste en mejorar el tiempo de acceso a las entradas de la tabla.

Paginación jerárquica

La mayoría de sistemas computacionales soportan un gran espacio de direcciones lógicas.

- En este tipo de entorno, la propia tabla de páginas llega a ser excesivamente grande (hasta 4MB).
- Obviamente, no conviene asignar la tabla de páginas de forma contigua en memoria principal.
- Una solución simple a este problema consiste en dividir la tabla de páginas en fragmentos más pequeños y se puede llevar a cabo está división en varias formas distintas.

Paginación jerárquica

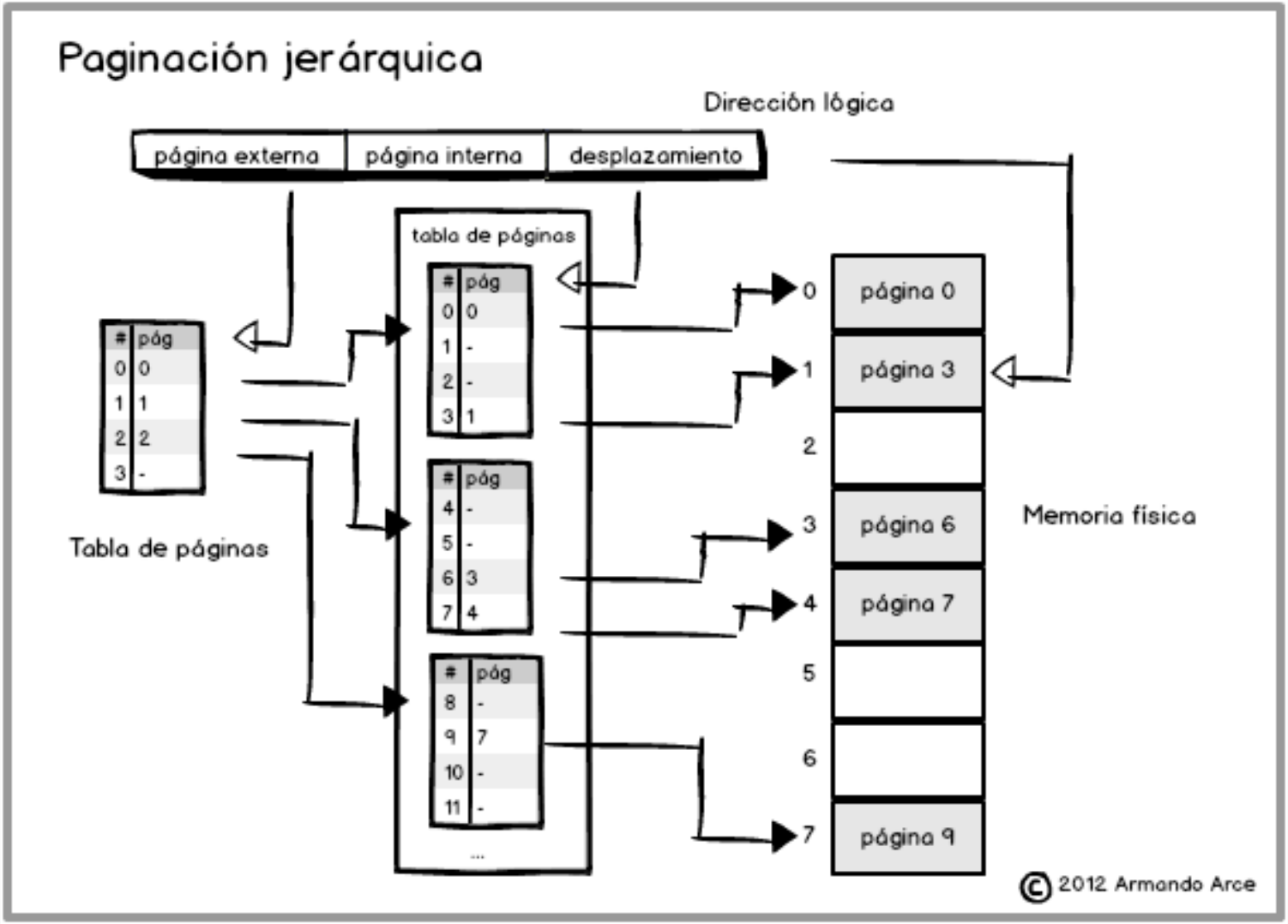

Figure 17:

Paginación jerárquica

Una de las formas consiste en utilizar un algoritmos de paginación de dos niveles, en el cual la propia tabla de páginas está también paginada (tabla de páginas externa).

- En este caso una dirección lógica se divide en tres partes: el número de página externa, el número de página, y el desplazamiento.

Paginación jerárquica

De ser necesario se puede aumentar los niveles de paginación a tres, creando una segunda tabla de páginas externa.

- El siguiente paso sería un esquema de paginación de cuatro niveles, en el que se paginaría también la propia tabla de páginas externas de segundo nivel.

Tabla de páginas hash

Un técnica común para administrar los espacios de direcciones superiores a 32 bits consiste en utilizar una *tabla hash de páginas*, donde el valor *hash* es el número de página virtual.

- Cada entrada de la tabla *hash* contiene una lista enlazada de elementos que tienen como valor *hash* una misma ubicación (con el fin de tratar colisiones).
- Cada elementos está compuesto de tres campos: el número de página virtual, el valor del marco de página maleado y un puntero al siguiente elemento de la lista enlazada.

Tabla de páginas hash

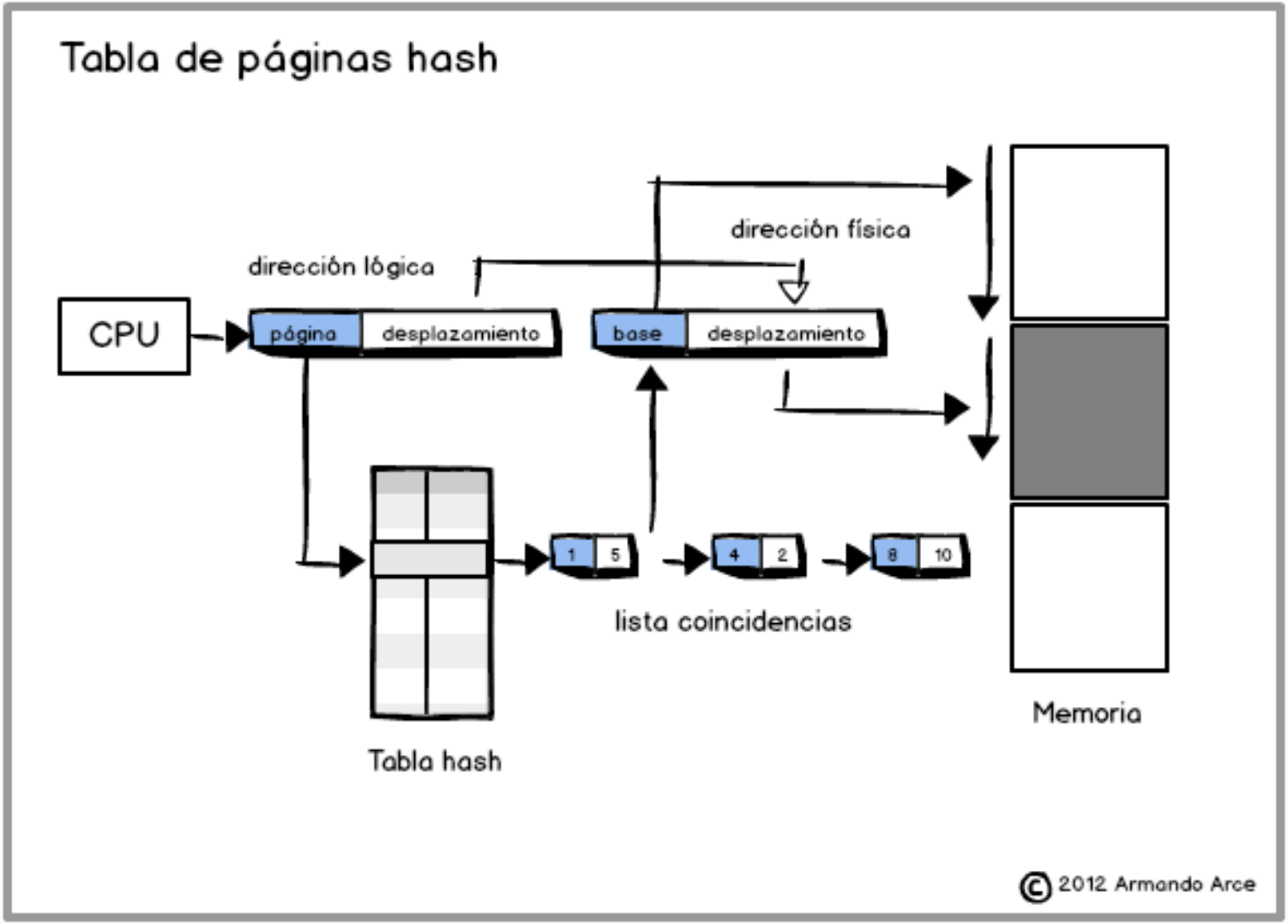

Figure 18:

Tabla de páginas hash

También se ha propuesto una variante de este esquema que resulta más adecuada para los espacios de direcciones de 64 bits.

- Esta variante utiliza *tablas de páginas en clúster*, que son similares a las tablas de páginas *hash*, salvo porque cada entrada de la tabla *hash* apunta a varias páginas en lugar de una sola página.

Tabla de páginas invertidas

La tabla de páginas incluye una entrada por cada página que el proceso esté utilizando.

- Cada tabla de página puede estar compuesta por millones de entradas.
- Estas tablas ocupan una gran cantidad de memoria física.

Tabla de páginas invertidas

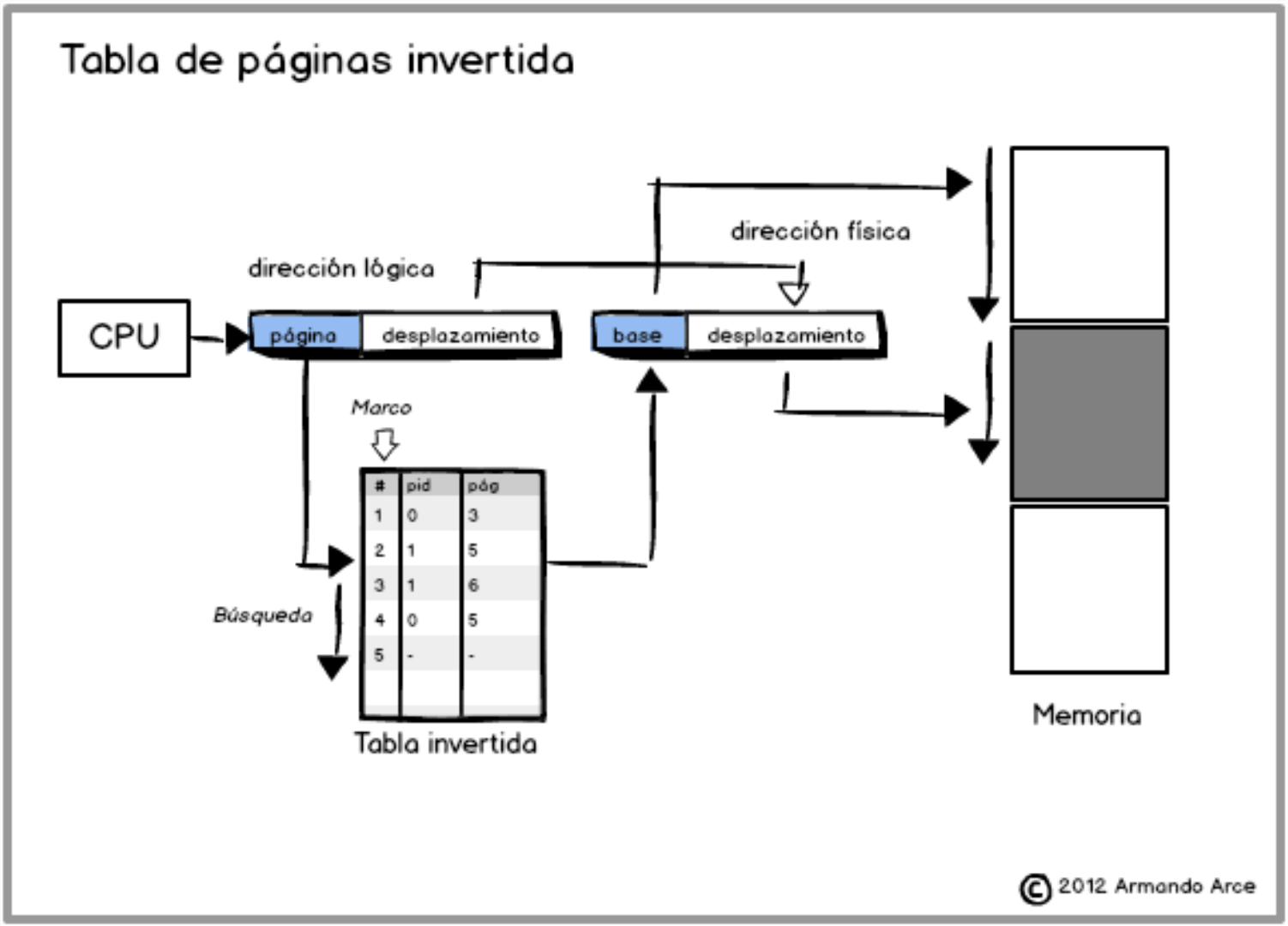

Figure 19:

Tabla de páginas invertidas

Para resolver este problema, se puede utilizar una *tabla de páginas invertida*. Las tablas de páginas invertidas tienen una entrada por cada página real (o marco) de la memoria.

- Cada entrada está compuesta de la dirección virtual de la página almacenada en dicha ubicación de memoria real, e incluye información acerca del proceso que posee dicha página.
- Por tanto, en el sistema sólo habrá una única tabla de páginas y esa tabla sólo tendrá un entrada por cada página de memoria física.

Tabla de páginas invertidas

Aunque este esquema permite reducir la cantidad de memoria necesaria para almacenar cada tabla de páginas, incrementa la cantidad de tiempo necesaria para almacenar cada explorar la tabla cuando se produce una referencia a una página.

- Puede ser necesario explorar toda la tabla pues ésta se encuentra ordenada por número de marco (dirección física) y no por número de página (dirección virtual).

Tabla de páginas invertidas

Para aliviar este problema, se utiliza una tabla *hash*, con el fin de limitar la búsqueda a una sola entrada de la tabla o a una pocas entradas.

- Por tanto, ahora se requerirán dos accesos a memoria para traducir la dirección virtual: uno para la tabla *hash* y otro para la tabla invertida.
- Sin embargo, hay que recordar que generalmente se utilizará un mecanismo tipo TLB para hacer este acceso más eficiente.

Segmentación

Un espacio lógico de direcciones es una colección de segmentos y cada segmento tiene un nombre y una longitud.

- Las direcciones especifican tanto el nombre del segmento como el desplazamiento dentro de ese segmento.
- El usuario especifica, por tanto, cada dirección proporcionando dos valores: un nombre de segmento y un desplazamiento.

Segmentación

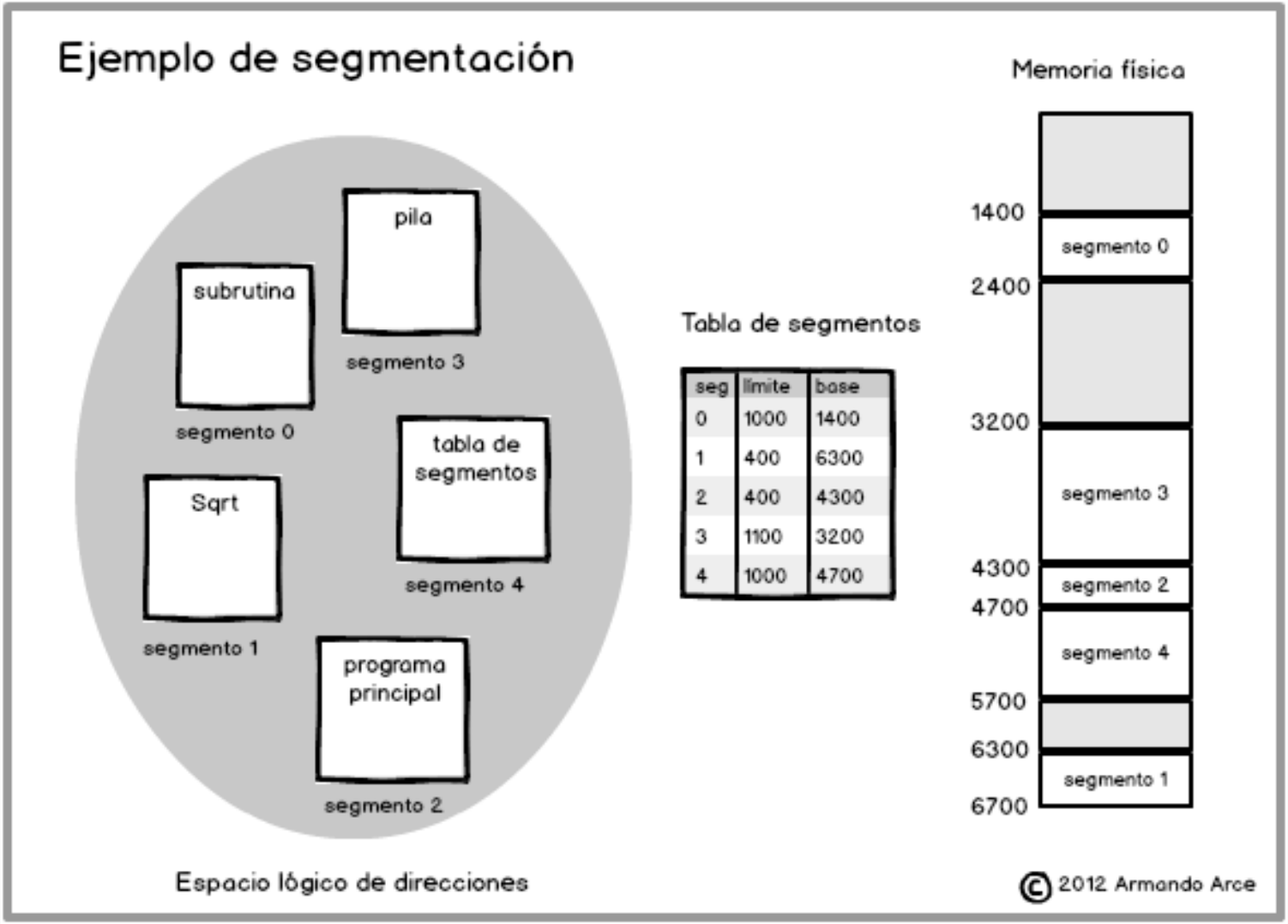

Figure 20:

Segmentación

Por simplicidad de implementación, los segmentos están numerados y se hace referencia a ellos mediante un número de segmento, en lugar de utilizar un nombre de segmento, así una dirección lógica estará compuesta por una pareja del tipo:

Segmentación

Normalmente, el programa del usuario se compila y el compilador construye automáticamente los segmentos para reflejar el programa de entrada. Un compilador C, podría crear segmentos separados para los siguientes elementos:

- El código
- Las variables globales
- El cúmulo de memoria a partir del cual se asigna la memoria
- Las pilas utilizadas por cada hilo de ejecución
- La biblioteca C estándar

Hardware de segmentación

El mapeo de dirección direcciones bidimensionales definidas por el usuario sobre las direcciones físicas unidimensionales, se llevan a cabo mediante una tabla de segmentos.

- Cada entrada de la tabla de segmentos tiene una dirección base del segmento y un límite del segmento.
- La dirección base del segmento contiene la dirección física inicial del lugar donde el segmento reside dentro de la memoria, mientras que el límite del segmento especifica la longitud de éste.

Hardware de segmentación

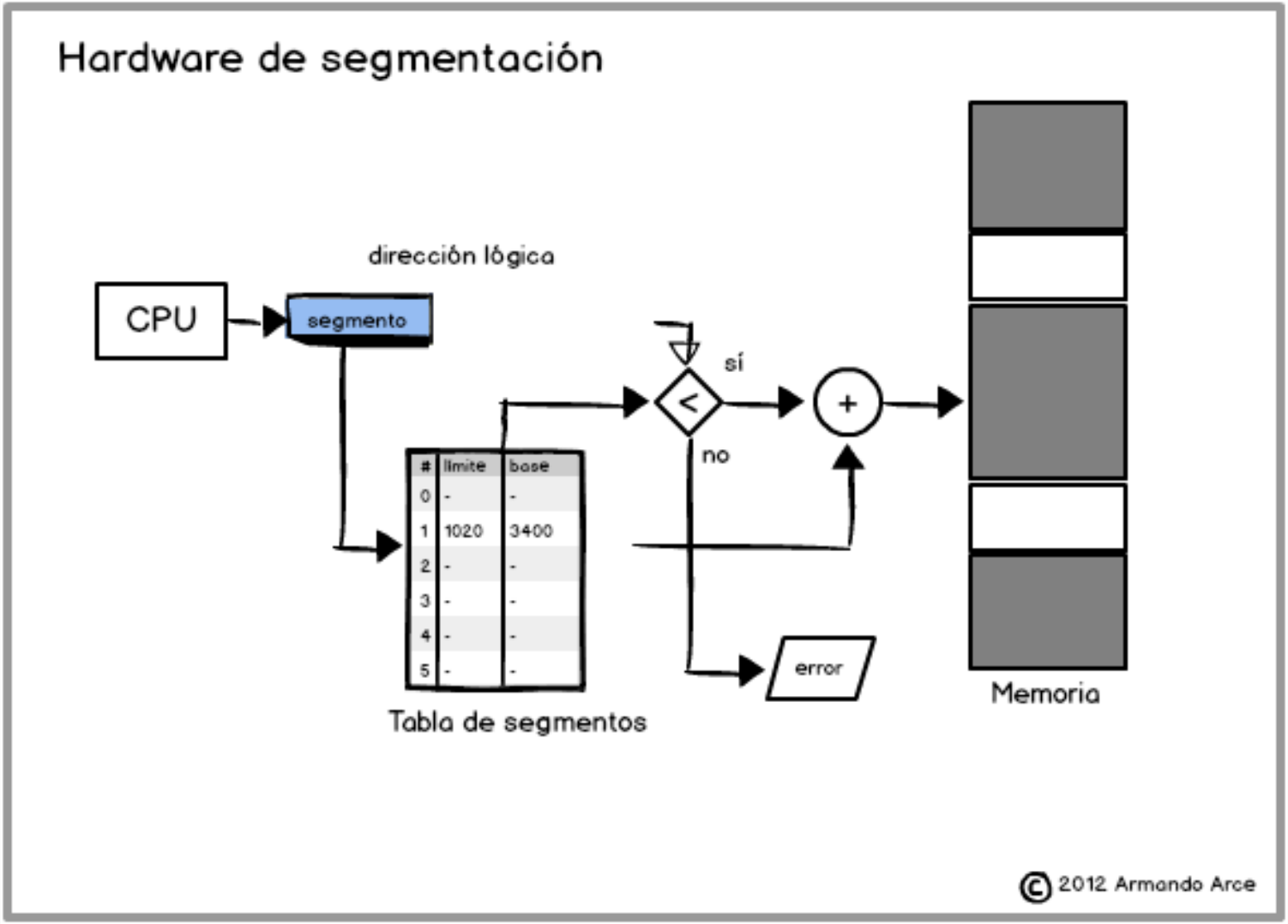

Figure 21:

Referencia bibliográfica

A.SILBERSCHATZ, P. GALVIN, y G. GAGNE, Operating Systems Concepts, Cap. 8, 9a Edición, John Wiley, 2013.