T6: Memoria Virtual

Departamento de Ingeniería de Computadores

Invierno 2020

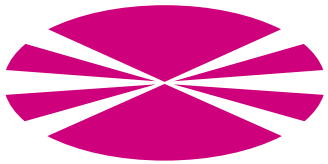


Table of Content

Memoria Virtual

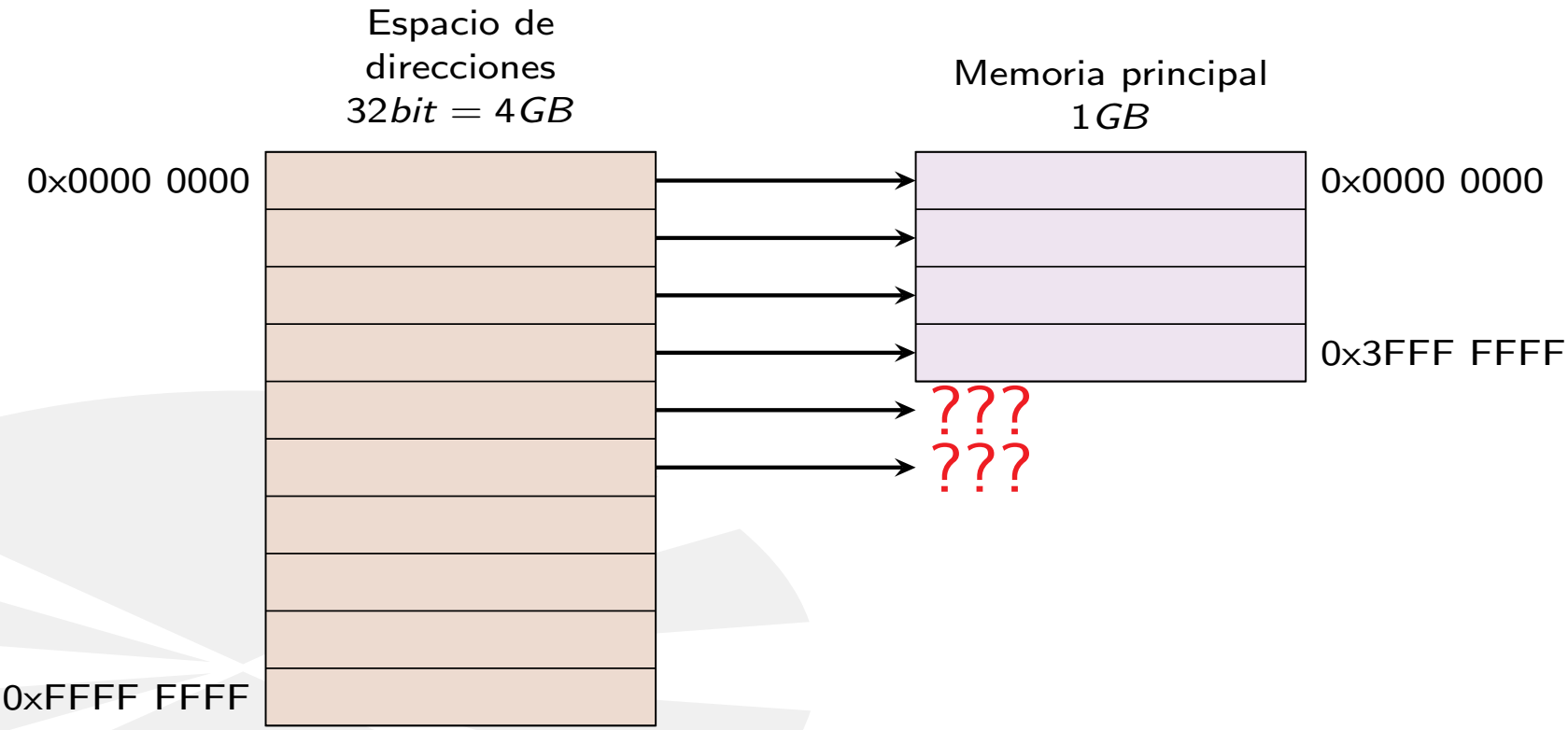
- En nuestros PCs tenemos un **espacio de direcciones** de 32 o 64 bits.
- Los programas pueden acceder a cualquier byte dentro de ese **espacio de direcciones**.
  - 32 bit: $2^{32}B = 4GB$
  - 64 bit: $2^{64}B = 16EB$
- Pero no necesariamente disponemos de tanta memoria principal.
  - P.ej., si tenemos 1GB, 30 bits son suficientes: $log_2(1G) = log_2(2^{30}) = 30$

Problemas

1. Un programa puede necesitar **más memoria de la que disponemos**.
   - Los programas pueden acceder a **direcciones que no existen** en la memoria principal.
2. Varios programas en ejecución a la vez, en conjunto pueden necesitar **más memoria de la que podemos direccionar**.
3. Varios programas en ejecución a la vez pueden **acceder a la misma dirección de memoria**.

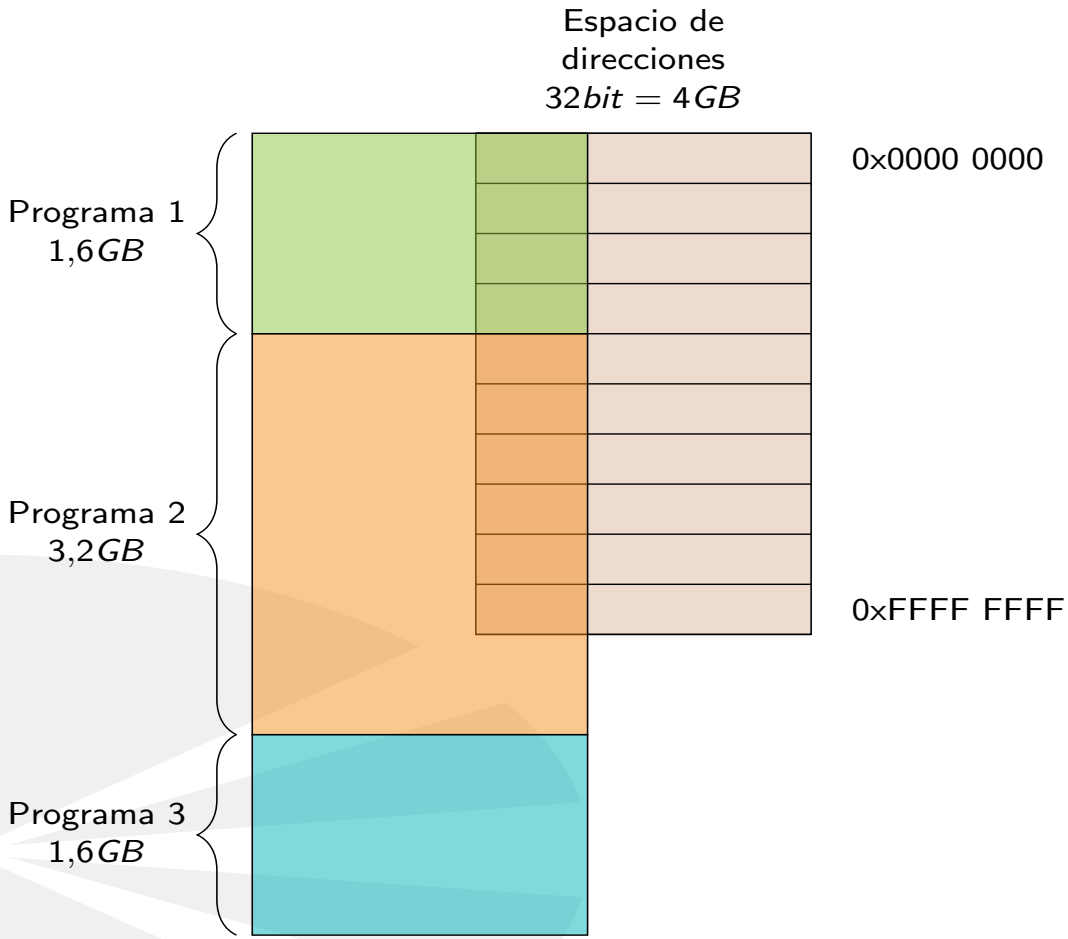
Memoria Virtual

Problema 1: Los programas pueden necesitar **más memoria de la que disponemos** → Los programas pueden acceder a **direcciones que no existen** en la memoria principal.



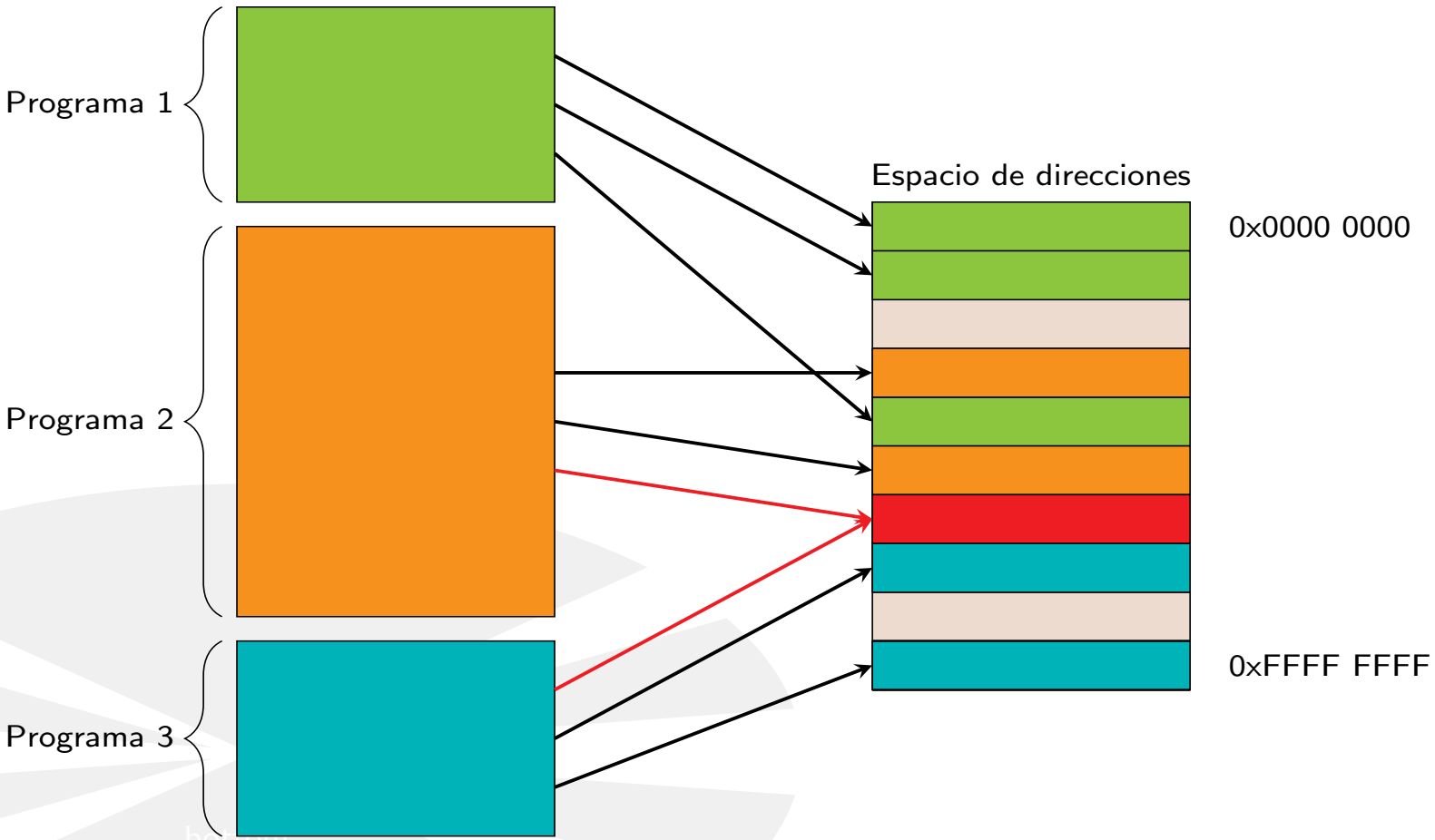
Memoria Virtual

Problema 2: Varios programas en ejecución a la vez, en conjunto pueden necesitar **más memoria de la que podemos direccionar**.



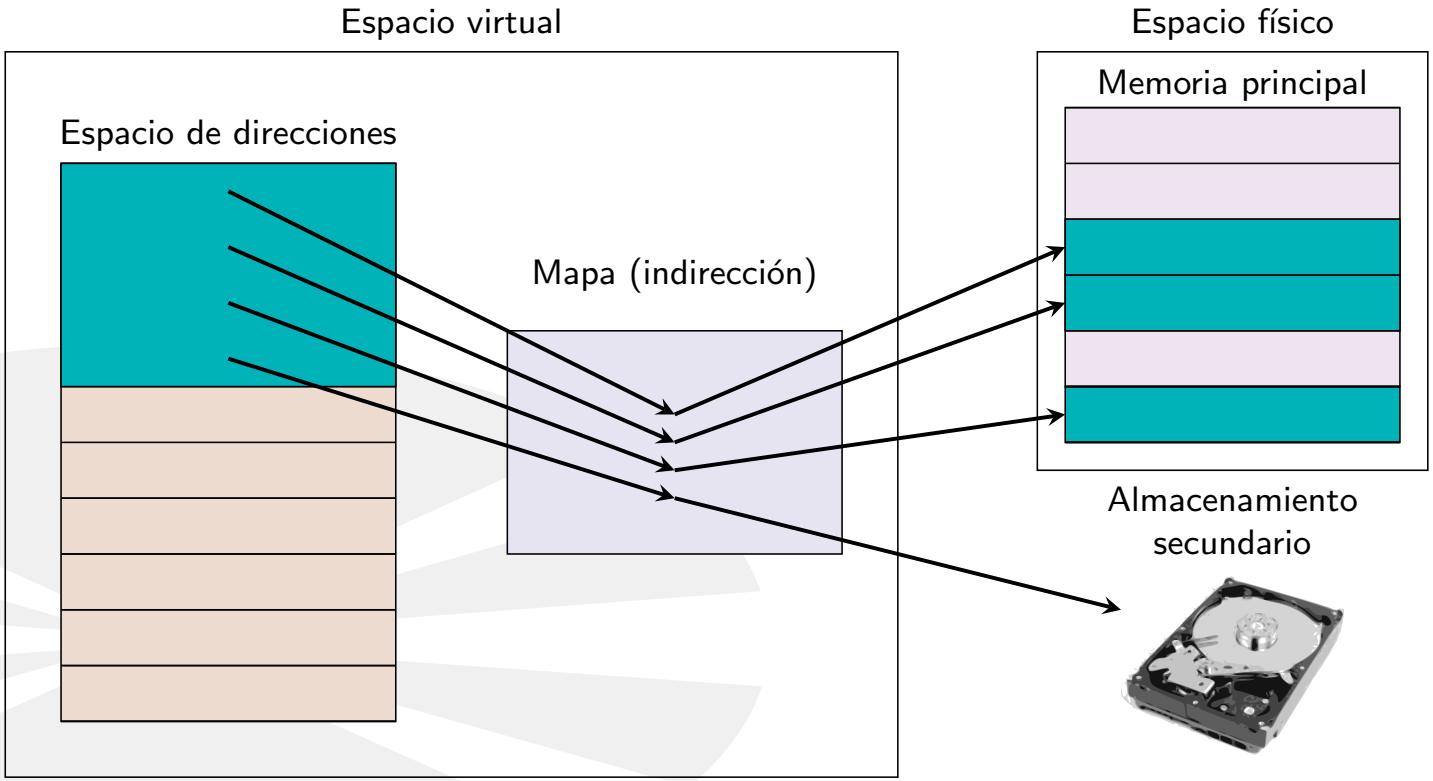
Memoria Virtual

Problema 3: Varios programas en ejecución a la vez pueden **acceder a la misma dirección de memoria**.

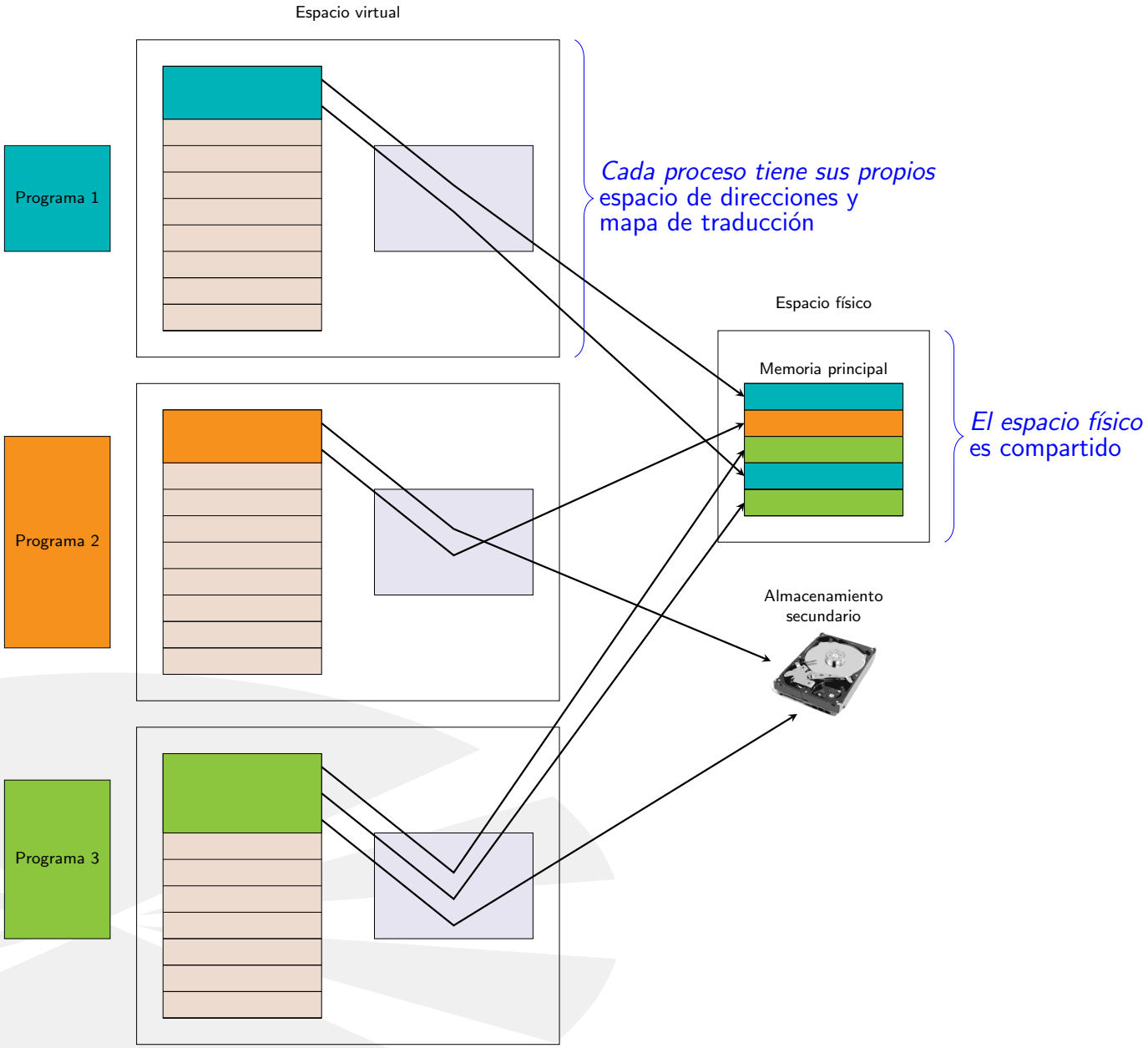


Memoria Virtual - Indirección

- Creamos una nueva capa de abstracción que nos permita mapear las direcciones con las que trabaja el programa (**direcciones virtuales**) a direcciones de memoria (**direcciones físicas**). De esto se encarga la **MMU** (Unidad de Gestión de Memoria).
- Un mapa nos permite **traducir** direcciones virtuales en direcciones físicas.
- El **sistema operativo** se encarga de transferir datos entre memoria secundaria y memoria principal.

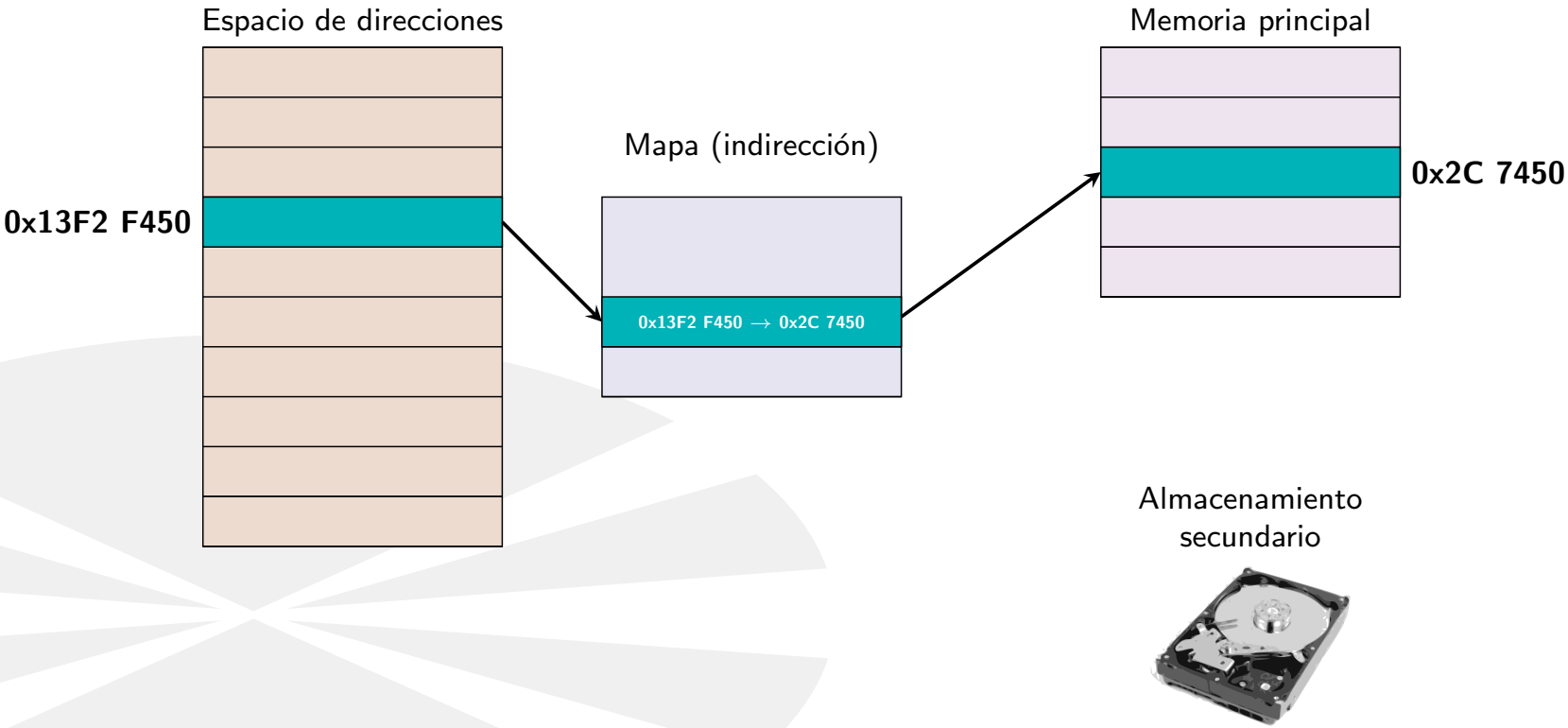


Memoria Virtual - Indirección



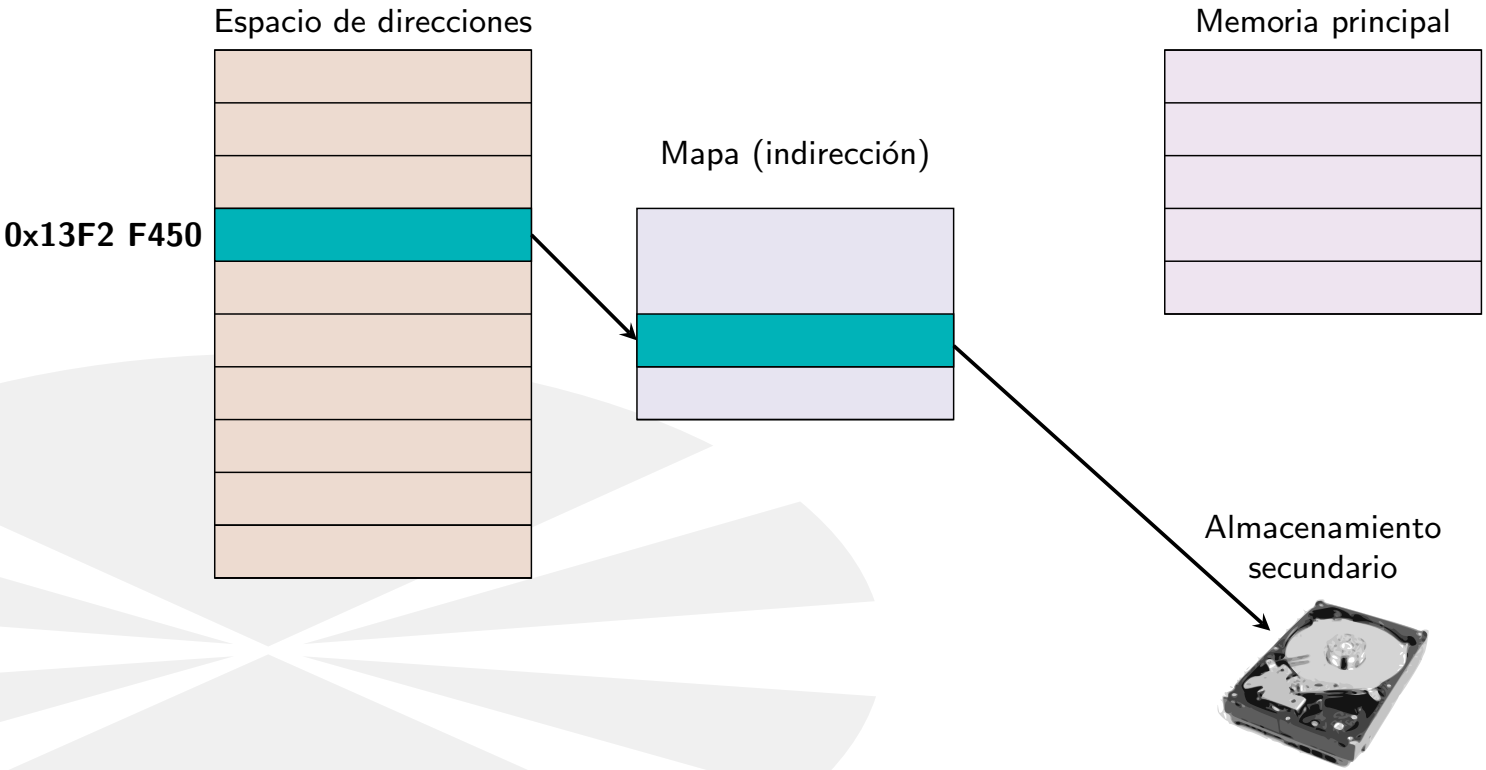
Memoria Virtual - Funcionamiento

- Cuando la CPU necesita un dato, debe estar en memoria principal
- Al buscar una dirección virtual en el mapa, pueden pasar 2 cosas:
  1. El dato al que referencia ya se encuentra en memoria principal.
  2. El dato no está en memoria principal. En ese caso, el sistema operativo se encarga de **transferirlo** desde memoria secundaria (swapping).
- Una vez que el dato ya está en memoria principal, podemos **traducir** la dirección virtual a dirección física.



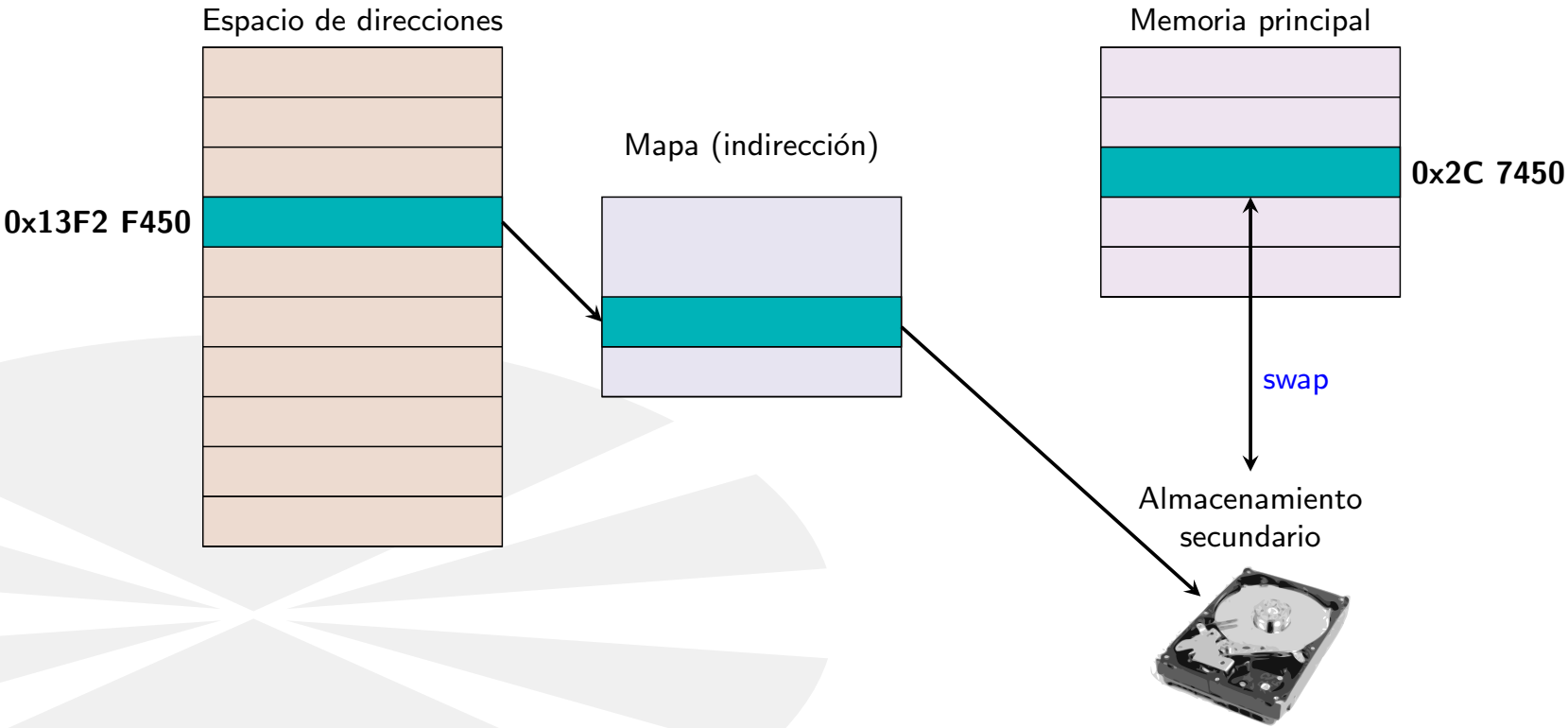
Memoria Virtual - Funcionamiento

- Cuando la CPU necesita un dato, debe estar en memoria principal
- Al buscar una dirección virtual en el mapa, pueden pasar 2 cosas:
  1. El dato al que referencia ya se encuentra en memoria principal.
  2. El dato no está en memoria principal. En ese caso, el sistema operativo se encarga de **transferirlo** desde memoria secundaria (swapping).
- Una vez que el dato ya está en memoria principal, podemos **traducir** la dirección virtual a dirección física.



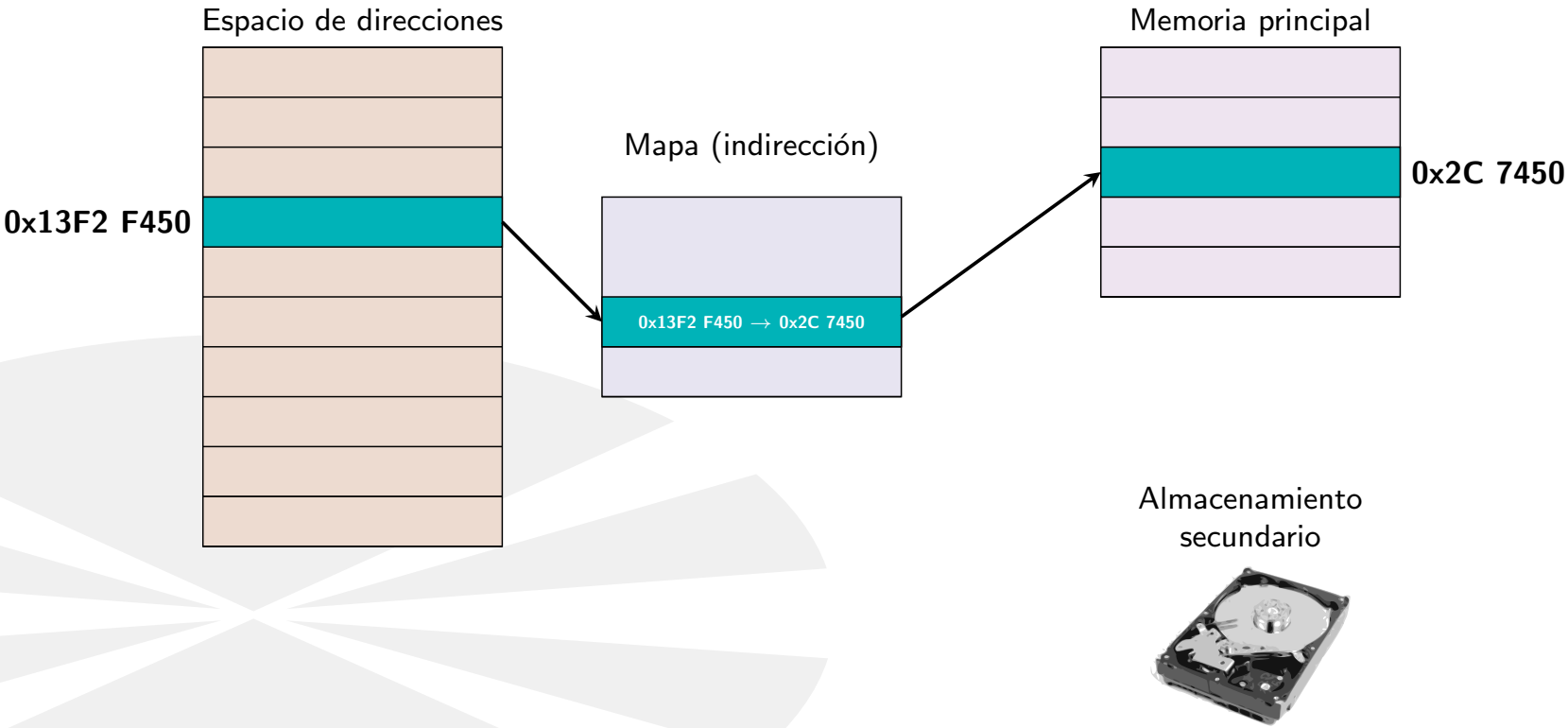
Memoria Virtual - Funcionamiento

- Cuando la CPU necesita un dato, debe estar en memoria principal
- Al buscar una dirección virtual en el mapa, pueden pasar 2 cosas:
  1. El dato al que referencia ya se encuentra en memoria principal.
  2. El dato no está en memoria principal. En ese caso, el sistema operativo se encarga de **transferirlo** desde memoria secundaria (swapping).
- Una vez que el dato ya está en memoria principal, podemos **traducir** la dirección virtual a dirección física.



Memoria Virtual - Funcionamiento

- Cuando la CPU necesita un dato, debe estar en memoria principal
- Al buscar una dirección virtual en el mapa, pueden pasar 2 cosas:
  1. El dato al que referencia ya se encuentra en memoria principal.
  2. El dato no está en memoria principal. En ese caso, el sistema operativo se encarga de **transferirlo** desde memoria secundaria (swapping).
- Una vez que el dato ya está en memoria principal, podemos **traducir** la dirección virtual a dirección física.



Memoria Virtual - Conceptos Básicos

- Similar a cuando en memoria caché transferíamos líneas, aquí también dividiremos nuestro espacio en bloques.
- Cuando un bloque no se encuentra en memoria principal, se conoce como **excepción**.
- Tenemos 2 tipos de sistemas:
  1. Sistemas **paginados**
     - Los bloques son todos del **mismo tamaño**.
     - Los bloques se llaman **páginas** y una excepción es un **fallo de página**.
     - El mapa de traducción es una **tabla de páginas**.
  2. Sistemas **segmentados**
     - Los bloques son de **diferente tamaño**.
     - Los bloques se llaman **segmentos** y una excepción es un **fallo de segmento**.
     - El mapa de traducción es una **tabla de segmentos**.

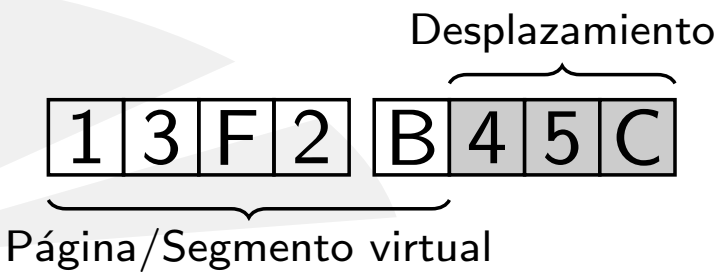
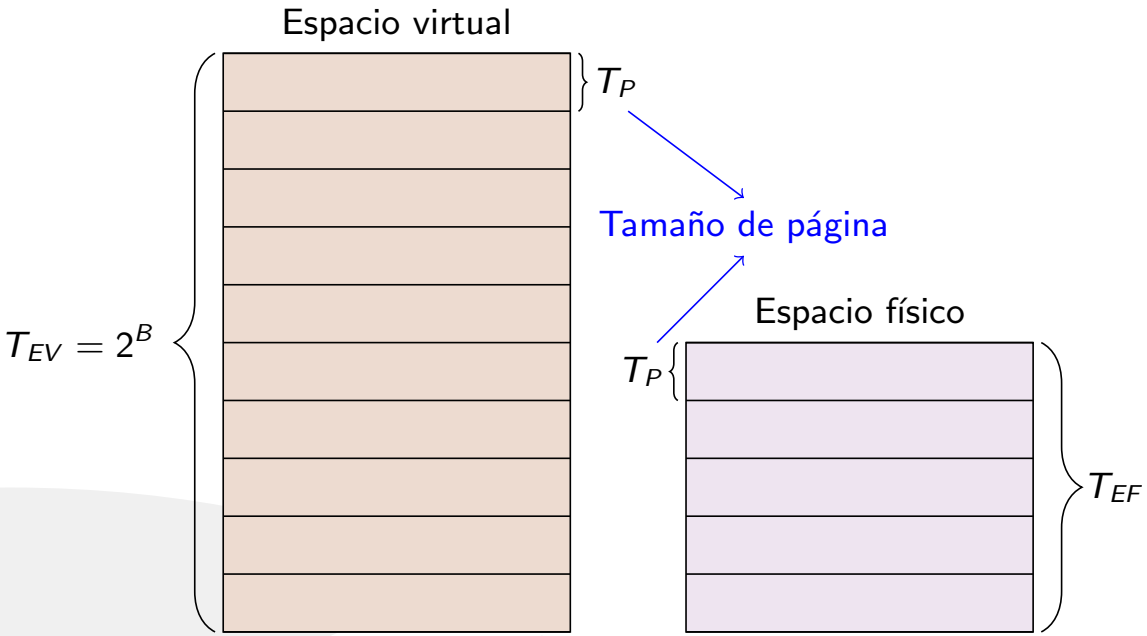


Table of Content

Sistemas Paginados - Conceptos Básicos

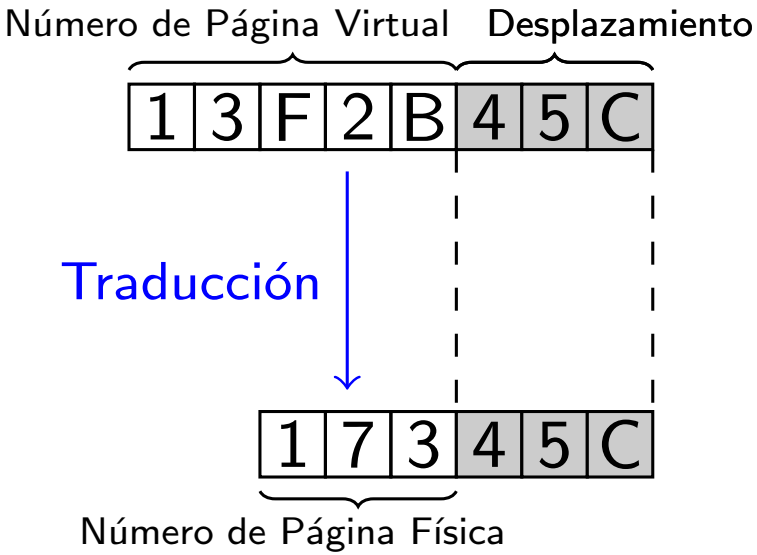
- Dividimos tanto el **espacio virtual** como el **espacio físico** en **páginas** del mismo tamaño.
- Tenemos por tanto **páginas virtuales** y **páginas físicas**.
- La ubicación de las páginas en la memoria física es determinada por el **sistema operativo**.



- El tamaño del espacio virtual (T_{EV}) depende del tamaño de nuestras direcciones.
- El tamaño del espacio físico (T_{EF}) es la memoria de la que disponemos.
- El tamaño de página (T_P) es arbitrario. Tendremos $\frac{T_{EV}}{T_P}$ páginas virtuales y $\frac{T_{EF}}{T_P}$ páginas físicas.

Sistemas Paginados - Conceptos Básicos

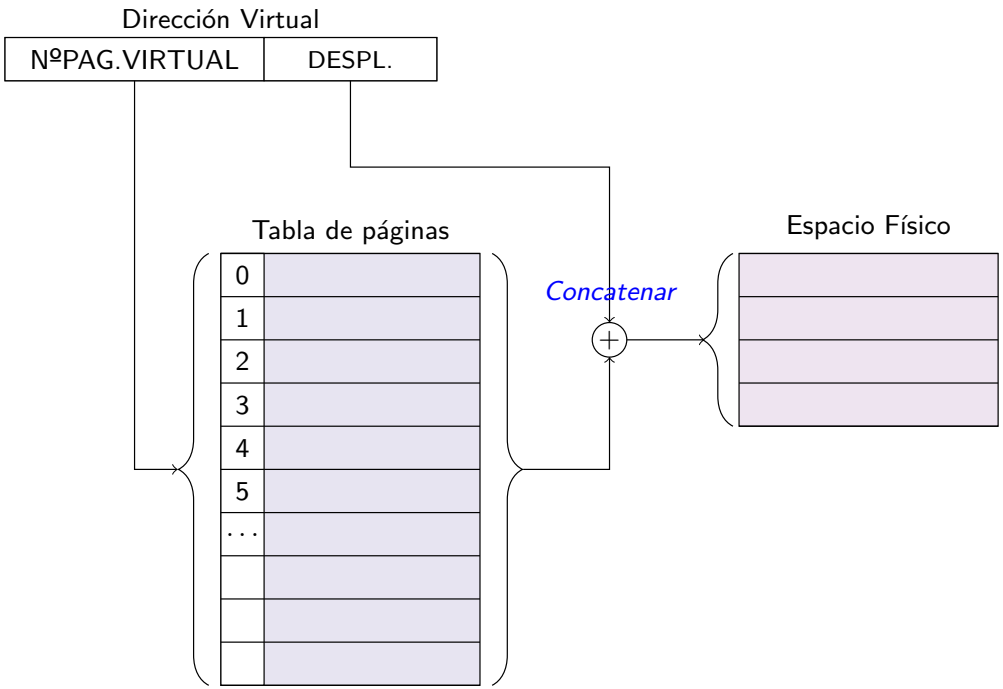
- Las **direcciones virtuales** se dividen en 2 campos:
  - Número de página virtual
  - Desplazamiento: Permite direccionar todos los bytes de una página: $log_2(T_P)$ bits
- Las **direcciones físicas** también:
  - Número de página física (traducida mediante la **tabla de páginas**)
  - Desplazamiento



Sistemas Paginados - Esquemas de Traducción

Traducción Directa

- Las direcciones virtuales se traducen en direcciones físicas mediante una tabla de acceso directo llamada **tabla de páginas**
  - Indexada por el número de página
  - Bit de residencia
  - Bit de validez



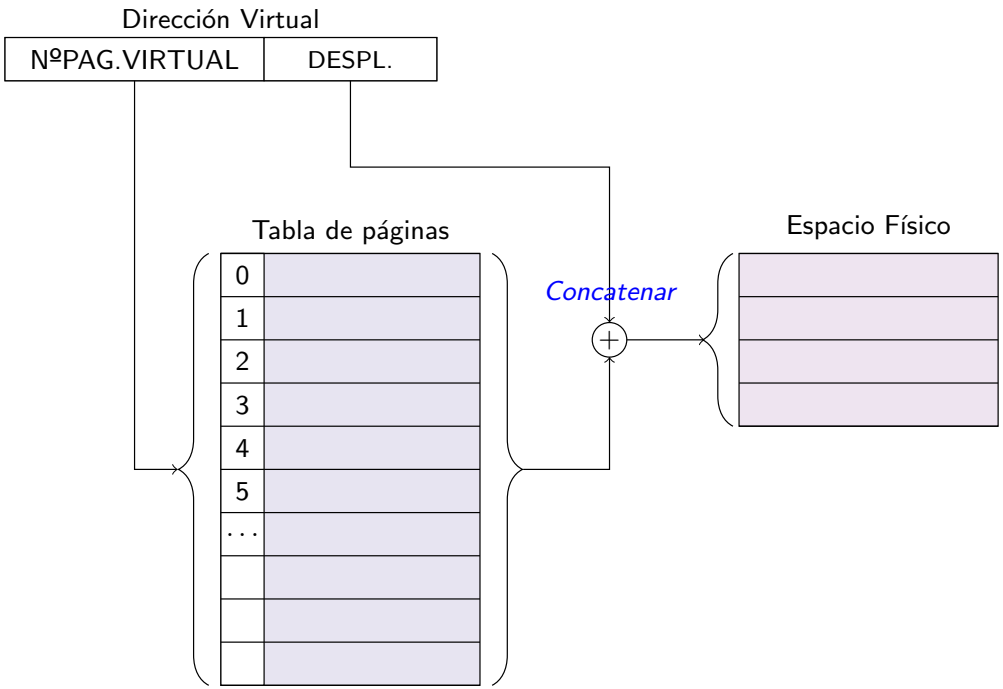
Entrada de la tabla de páginas

0	BITS CONTROL	NºPAG.FÍSICA

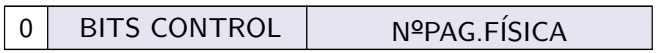
Sistemas Paginados - Esquemas de Traducción

Traducción Directa

- La tabla reside en memoria
- Contiene tantas entradas como páginas virtuales tenemos ($N_{PV} = T_{EV}/T_P$)
- Cada entrada ocupa lo necesario para almacenar:
  - Bits de control (residencia, validez, etc.) (N_{ctrl} bits)
  - Número de página física ($T_{DF} = log_2(T_{EV}/T_P)$ bits)
- La tabla ocupa por tanto $N_{PV} * (N_{ctrl} + T_{DF})$ bits



Entrada de la tabla de páginas



Sistemas Paginados - Ejemplo

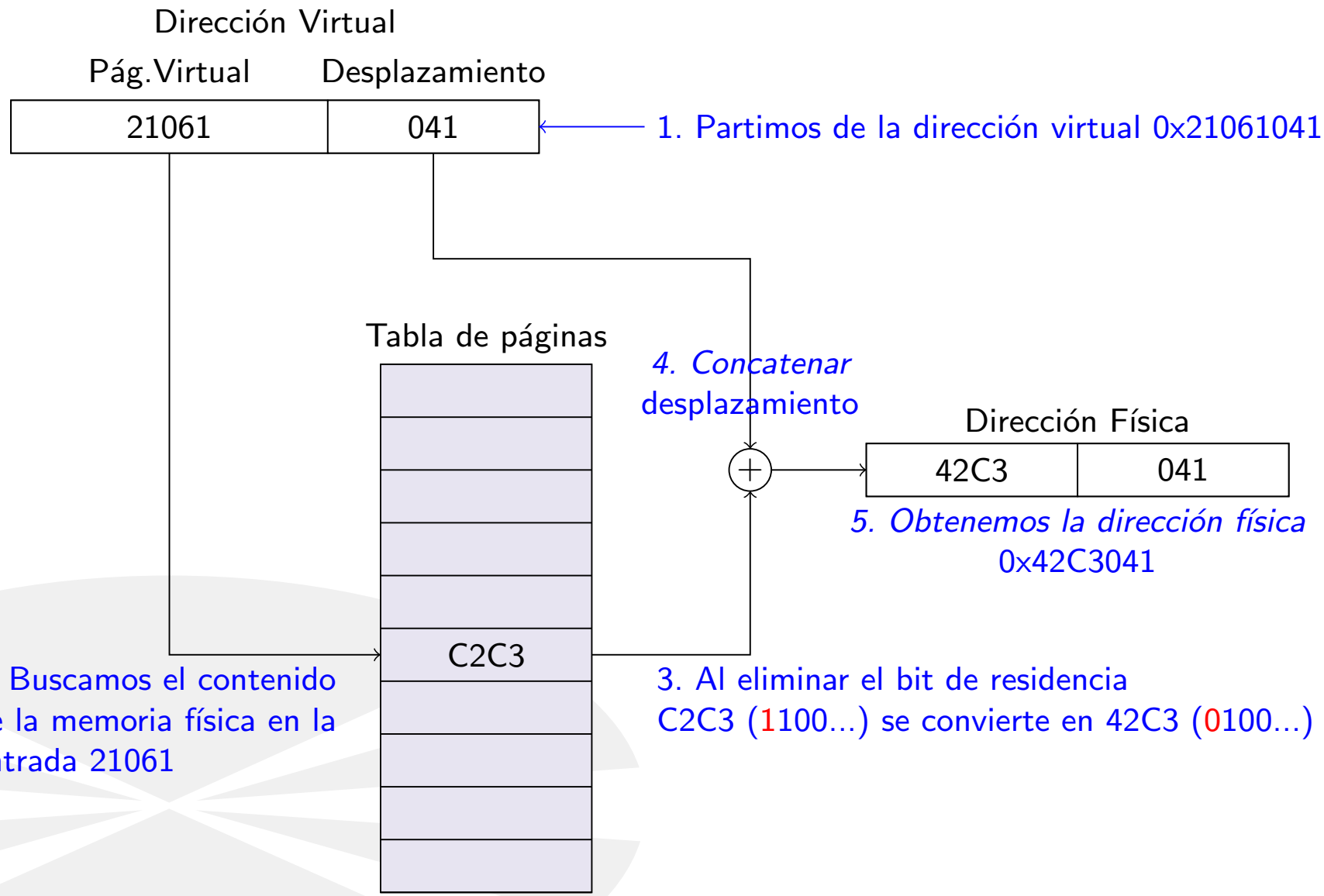
Si tenemos la siguiente configuración:

- Un espacio virtual paginado de 4GB (EV)
- El espacio físico es de 128 MB (EF)
- Cada página ocupa 4KB (T_P)
- Cada entrada de la tabla de páginas contiene un bit de residencia y el número de página física

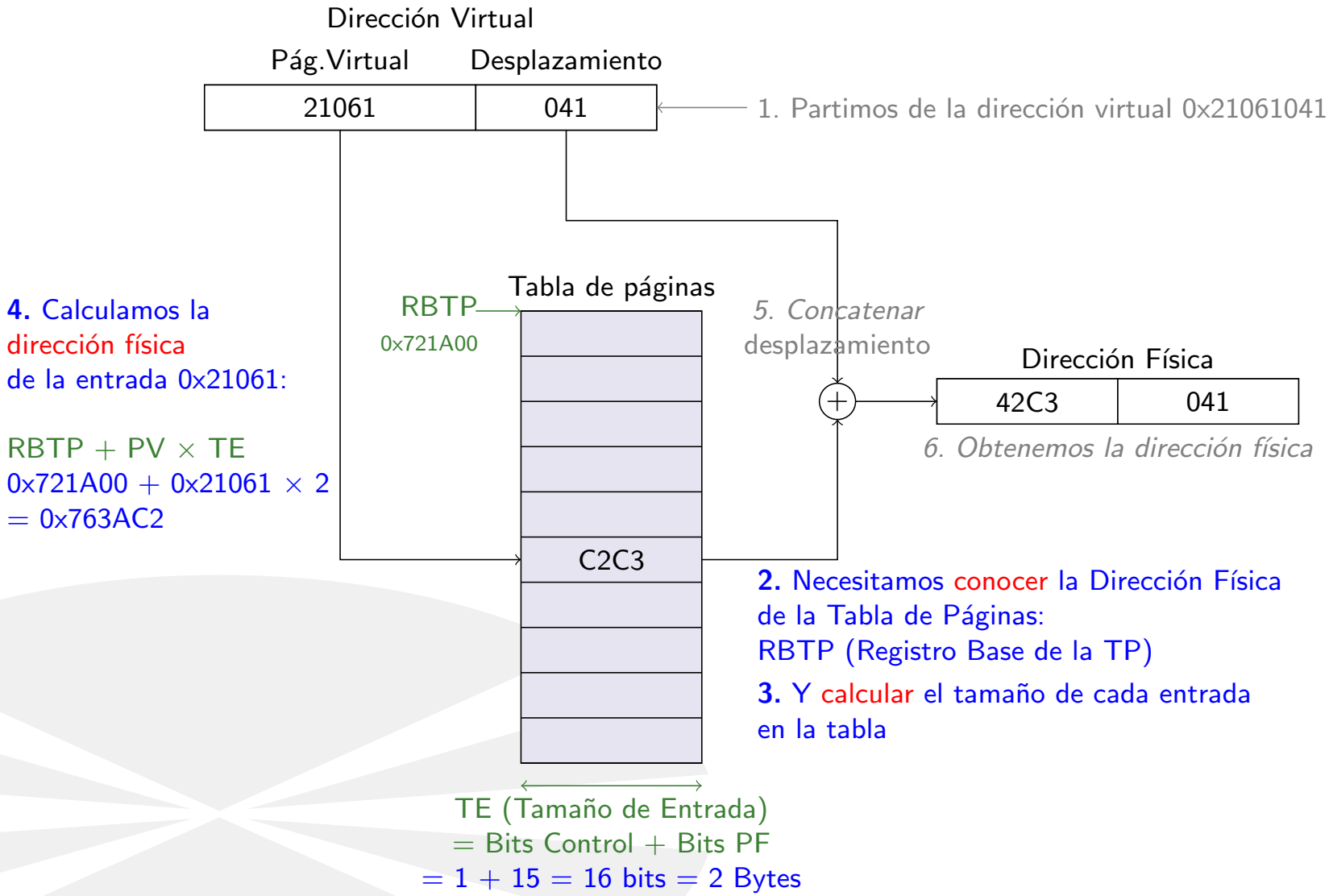
¿Cuánto ocupa en memoria la tabla de páginas?

- $|EV| = 2^{32}B, |EF| = 2^{27}B, T_P = 2^{12}B$
- Número de páginas virtuales: $\frac{|EV|}{T_P} = 2^{32-12} = 2^{20}$
- Número de páginas físicas: $\frac{|EF|}{T_P} = 2^{27-12} = 2^{15}$
- Tamaño de entrada en la tabla: $1 + log_2(2^{15}) = 16bits \rightarrow 2bytes$
- Tamaño de tabla: 2^{20} entradas $\times 2$ B/entrada $= 2^{21}$ B $= 2$MB

Sistemas Paginados - Ejemplo (Nivel básico)

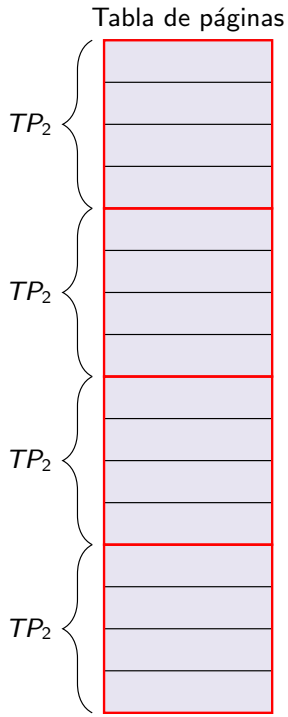


Sistemas Paginados - Ejemplo (Nivel avanzado)



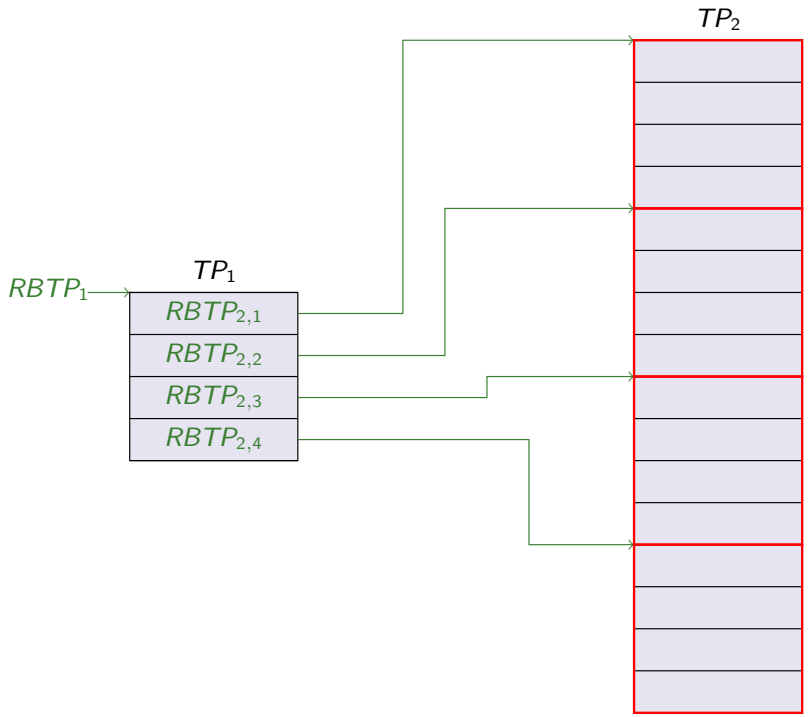
Sistemas Paginados - Paginación en 2 niveles

- En la traducción directa, la TP ocupa un espacio considerable en Memoria Física.
- Solución: Dividir las entradas de la TP en bloques de igual tamaño.
  - Cada uno de esos bloques será una "TP de nivel 2" (TP_2).
- Cada una de las TP_2 puede cargarse individualmente en MF cuando sea necesario.



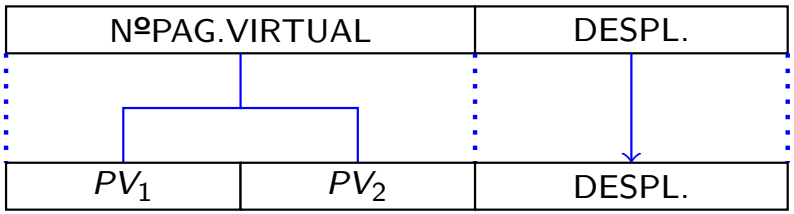
Sistemas Paginados - Paginación en 2 niveles

- Añadimos una TP de nivel 1 que contiene las direcciones físicas de las tablas de nivel 2.
- El formato de todas estas tablas es idéntico. Contienen **bits de control** y **páginas físicas**.
- TP_1 tendrá tantas entradas como tablas de nivel 2 tengamos.
- Cada TP_2 tendrá tantas entradas como resulte de dividir el número de páginas virtuales entre el número de entradas en TP_1.

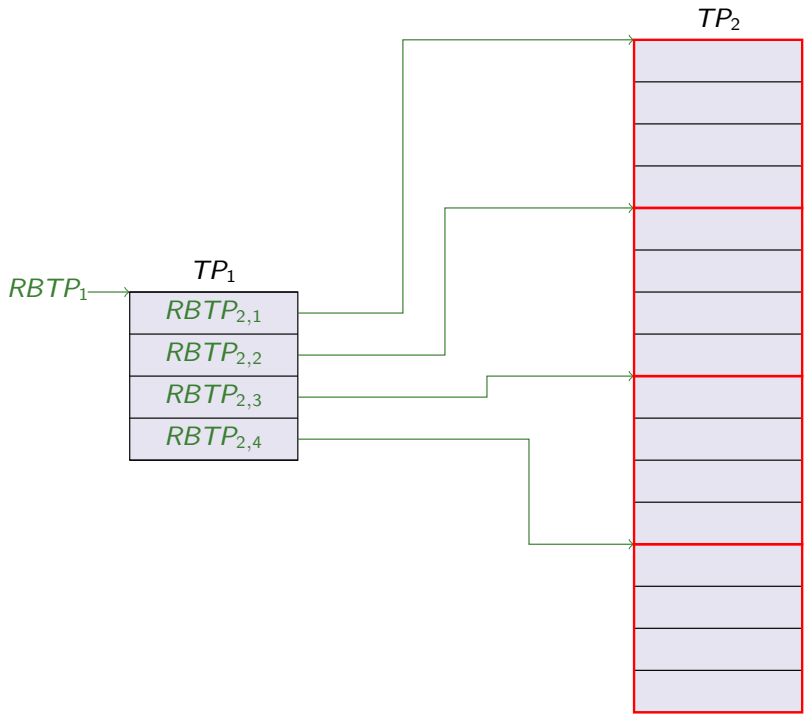


Sistemas Paginados - Paginación en 2 niveles

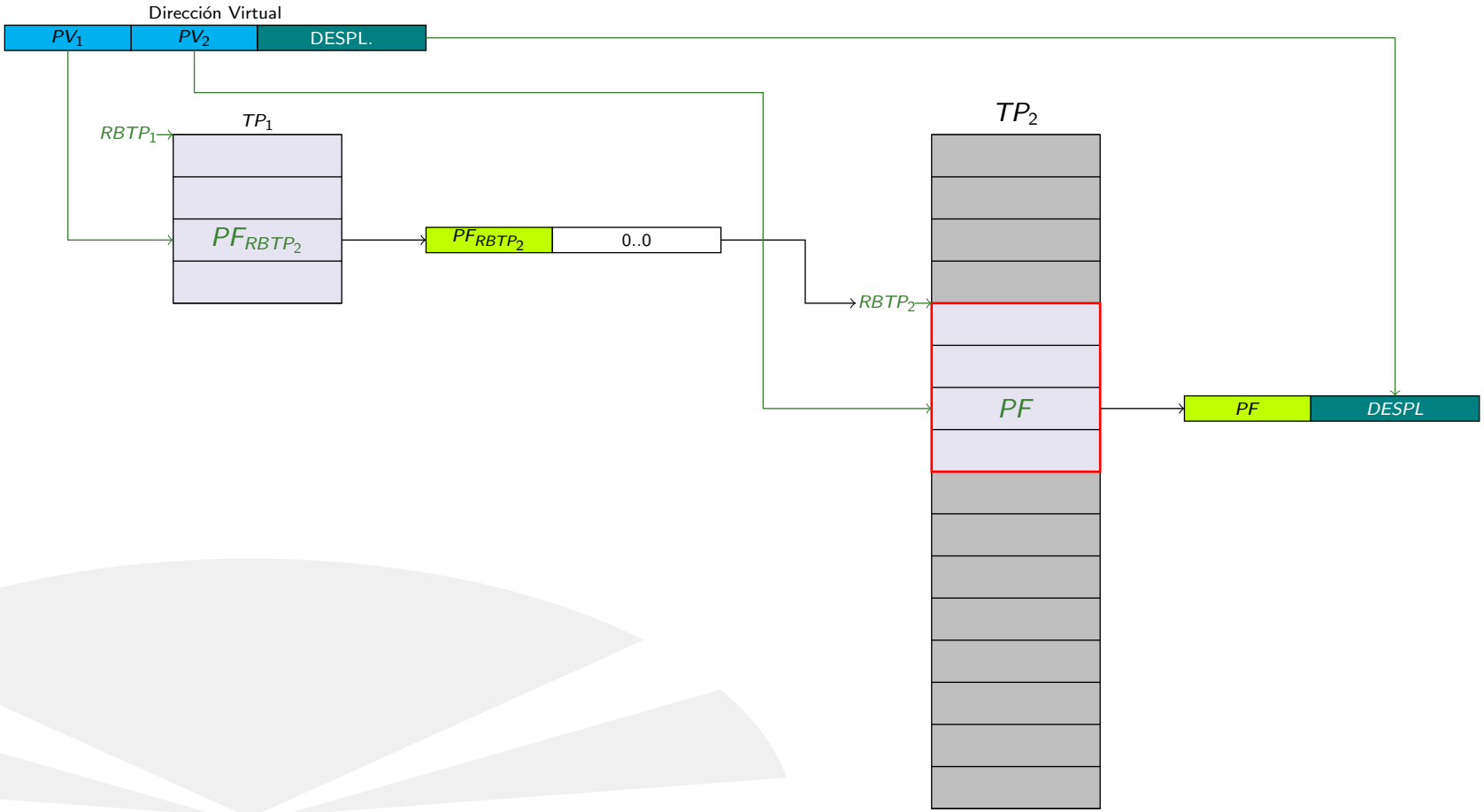
- Las **Direcciones Virtuales** contienen ahora **Página Virtual de nivel 1** y **Página Virtual de nivel 2**



- La traducción se hace ahora en 2 pasos.
  1. Se busca PV_1 en TP_1 para obtener la **Página Física que contiene** $RBTP_2$. A esa página física se le concatena un desplazamiento 0.
  2. Se busca PV_2 en la TP_2 ubicada en $RBTP_2$ para obtener la **Página Física que buscamos**. A esa página física se le concatena el desplazamiento de la Dirección Virtual.



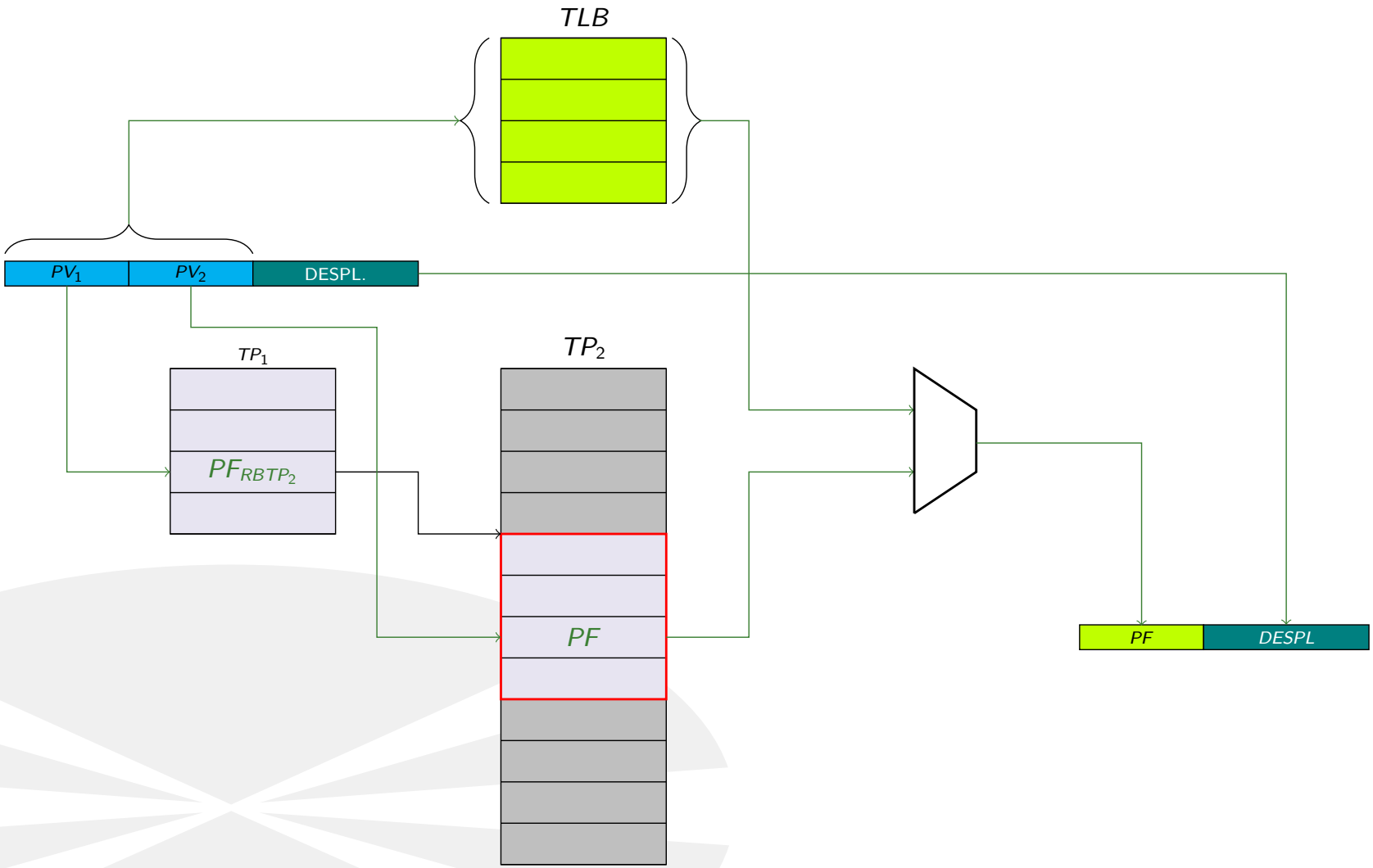
Sistemas Paginados - Paginación en 2 niveles



Sistemas Paginados - Aceleración de la traducción

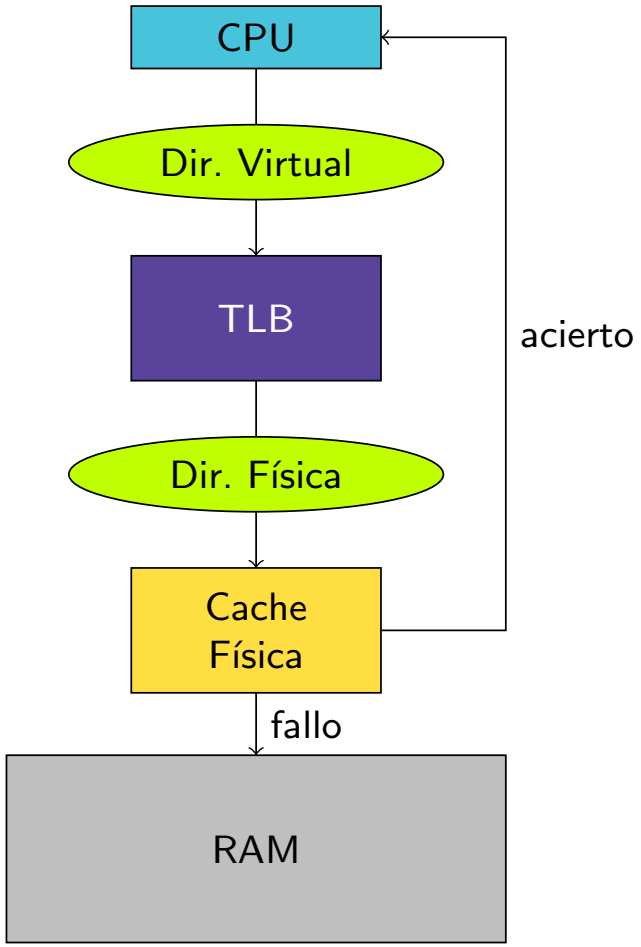
- Traducción directa y asociativa combinada
  - Combina la ventaja del bajo coste hw de la traducción directa con la ventaja de la alta velocidad de traducción asociativa
  - La memoria asociativa o TLB (Translation Lookaside Buffer) almacena los pares {Pág.Virtual - Pág.Física} **más recientemente referenciados** junto con los bits de gestión que se requieran
  - Funciona como una pequeña caché para las direcciones ya traducidas
  - Su éxito está justificado por el principio de localidad
  - Tamaños típicos de TLB: 32 a 512 entradas

Sistemas Paginados - TLB



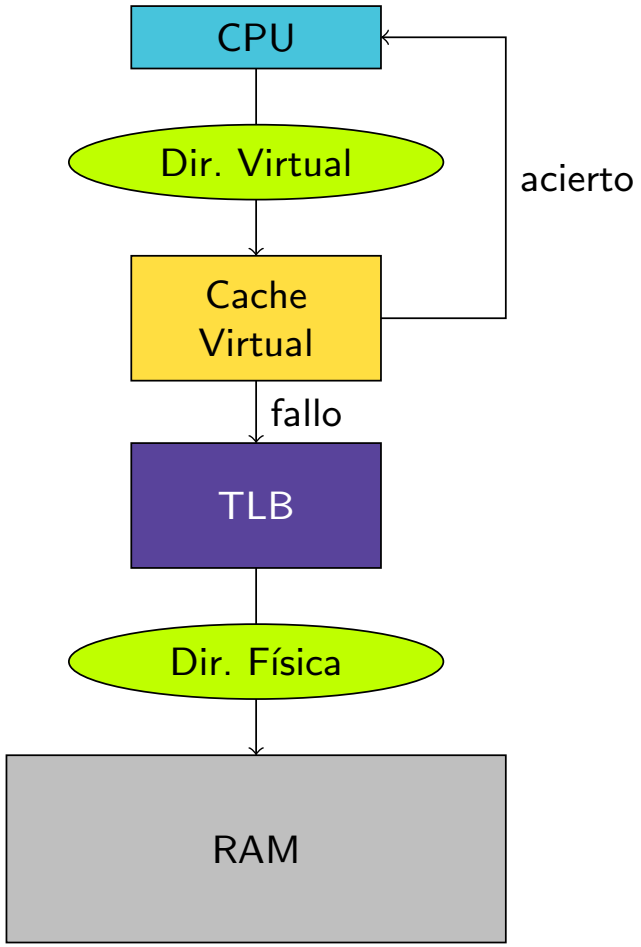
Sistemas Paginados - Cachés Virtuales

- Caché física
  - Es el sistema que conocemos hasta ahora
  - Debemos traducir la dirección virtual antes de acceder a la caché
- Caché virtual
  - Usamos direcciones virtuales para indexar la caché
  - Solo traducimos la dirección si tenemos un fallo en la caché
  - Problema: Dos procesos no pueden compartir la caché virtual
    - Necesitamos identificar el PID del proceso en cada línea
    - O bien purgamos la caché en cada cambio de contexto

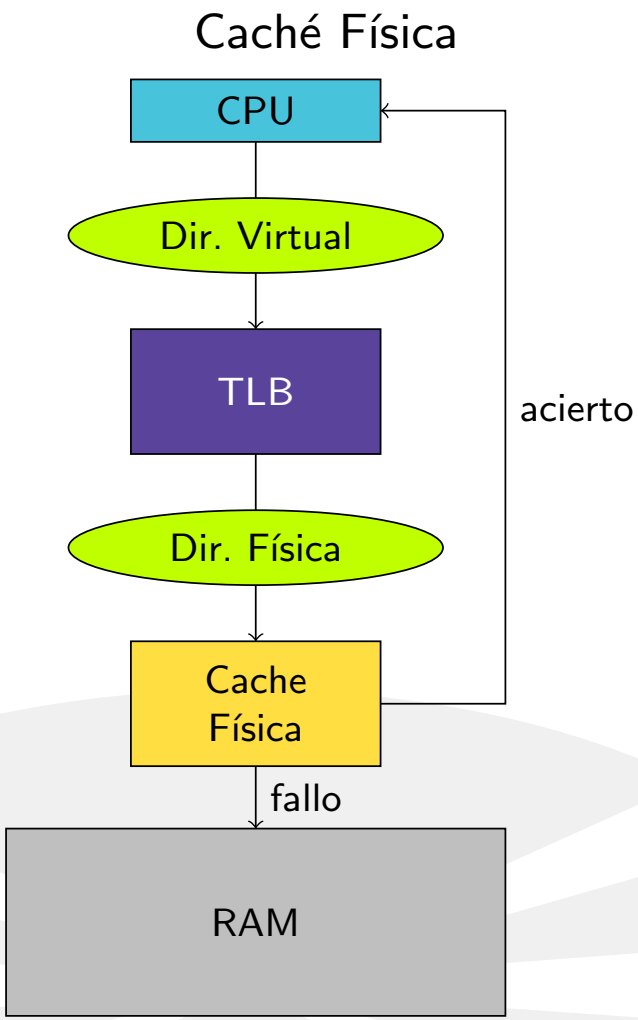


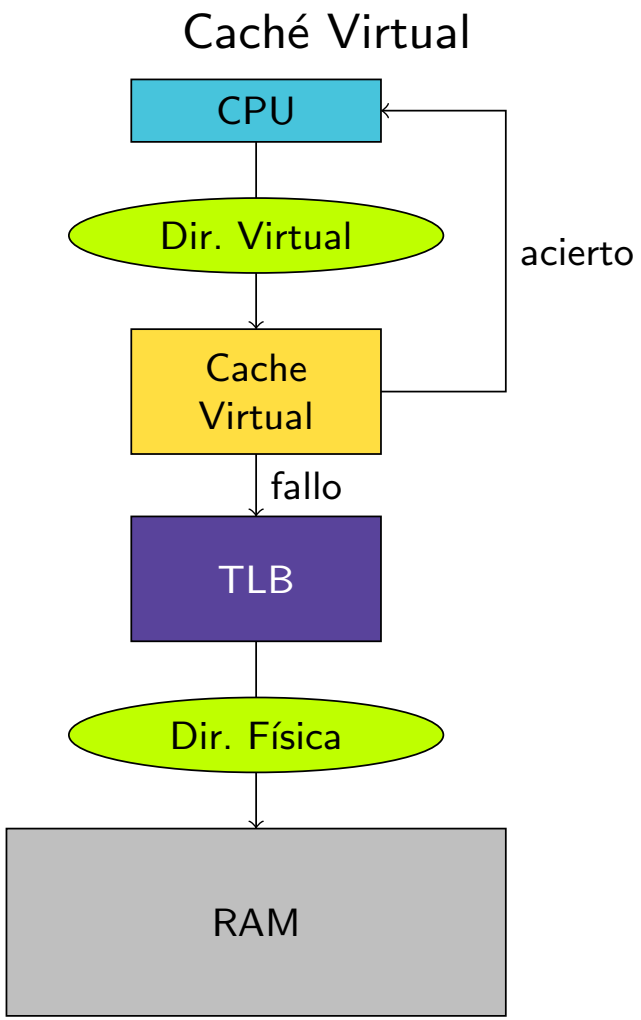
Sistemas Paginados - Cachés Virtuales

- Caché física
  - Es el sistema que conocemos hasta ahora
  - Debemos traducir la dirección virtual antes de acceder a la caché
- Caché virtual
  - Usamos direcciones virtuales para indexar la caché
  - Solo traducimos la dirección si tenemos un fallo en la caché
  - Problema: Dos procesos no pueden compartir la caché virtual
    - Necesitamos identificar el PID del proceso en cada línea
    - O bien purgamos la caché en cada cambio de contexto



Sistemas Paginados - Cachés Virtuales

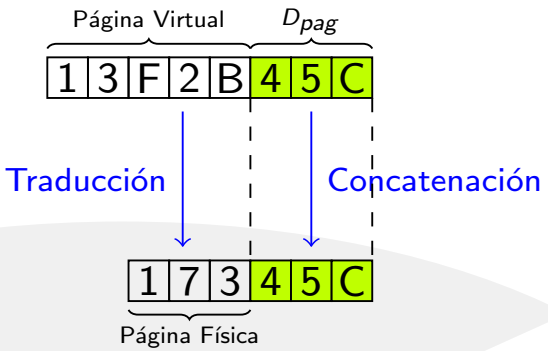




Sistemas Paginados - Virtually Indexed Physically Tagged

Caché con Índices Virtuales y Etiquetas Físicas (VIPT: Virtually Indexed Physically Tagged)

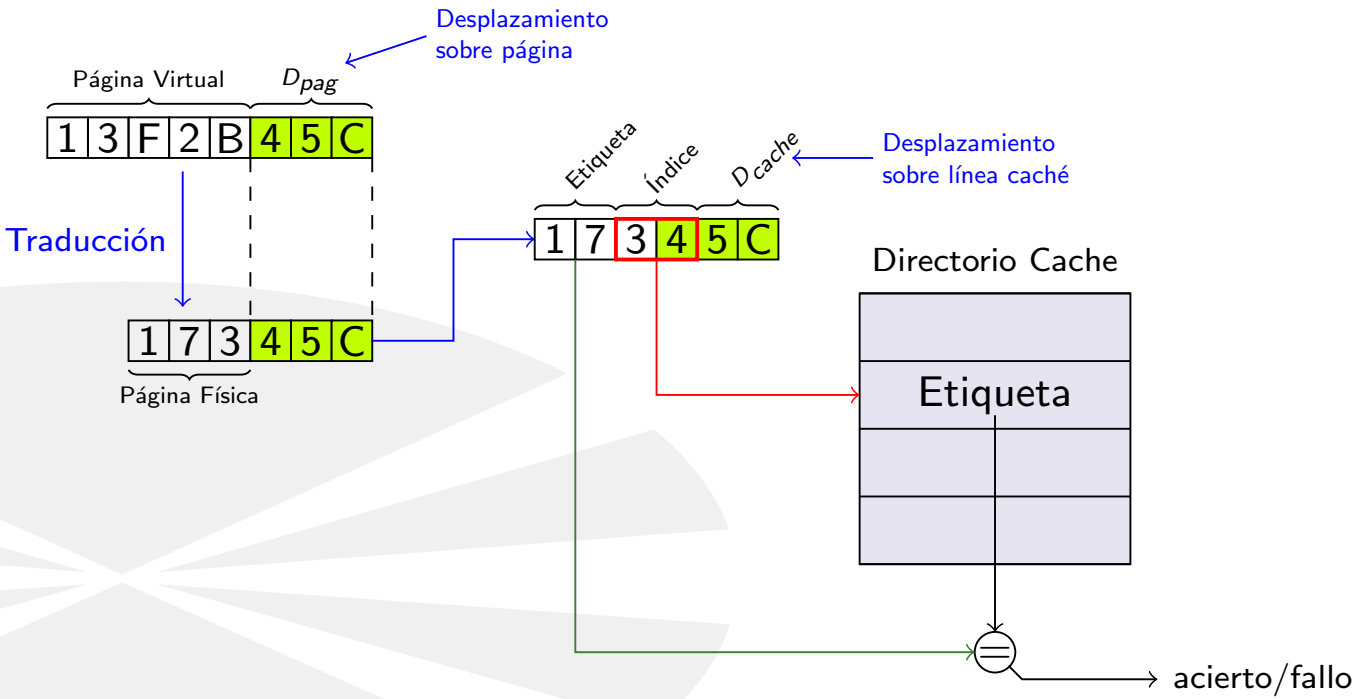
- En la traducción de Direcciones Virtuales, en realidad sólo se traduce la Página Virtual a Página Física
- El desplazamiento es idéntico, y por tanto forma parte de la Dirección Física
- La Dirección Física se divide en Etiqueta, Índice y Desplazamiento para acceder a la caché
- Si la caché está organizada de tal modo que el índice caiga dentro del desplazamiento de página, conocemos índice y desplazamiento caché antes de hacer la traducción
- Entonces, podemos buscar la línea caché con la dirección virtual, y luego verificar la etiqueta



Sistemas Paginados - Virtually Indexed Physically Tagged

Caché con Índices Virtuales y Etiquetas Físicas (VIPT: Virtually Indexed Physically Tagged)

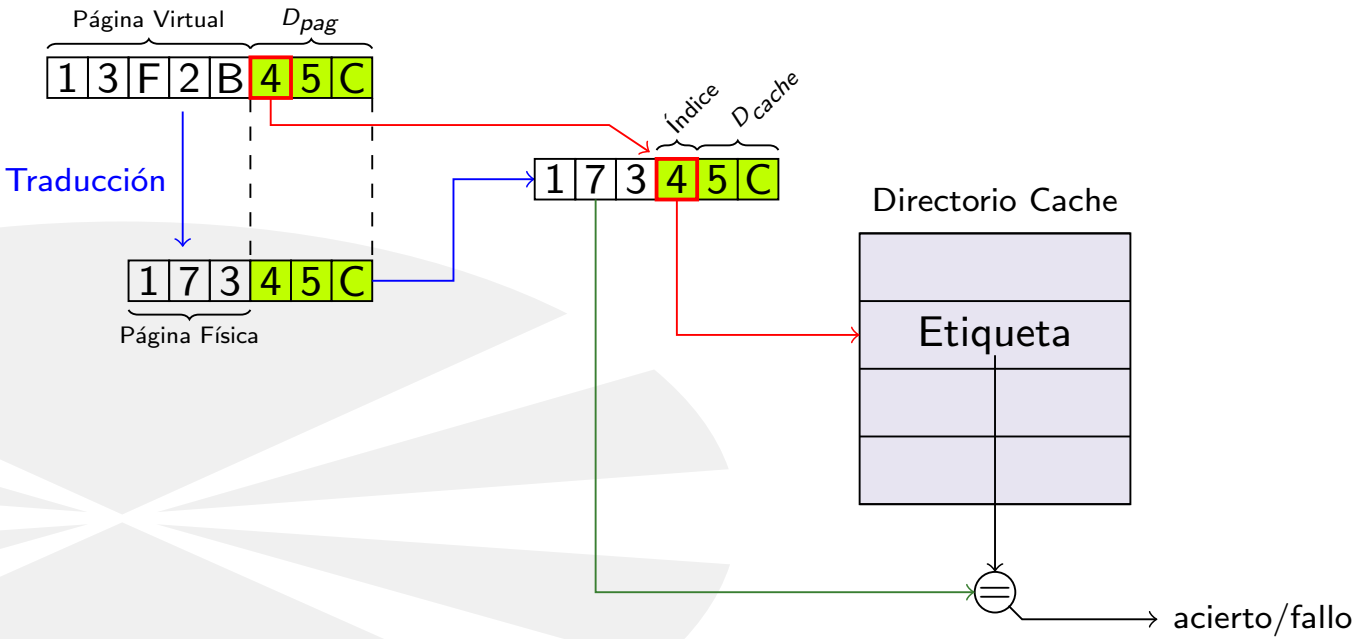
- En la traducción de Direcciones Virtuales, en realidad sólo se traduce la Página Virtual a Página Física
- El desplazamiento es idéntico, y por tanto forma parte de la Dirección Física
- La Dirección Física se divide en Etiqueta, Índice y Desplazamiento para acceder a la caché
- Si la caché está organizada de tal modo que el índice caiga dentro del desplazamiento de página, conocemos índice y desplazamiento caché antes de hacer la traducción
- Entonces, podemos buscar la línea caché con la dirección virtual, y luego verificar la etiqueta



Sistemas Paginados - Virtually Indexed Physically Tagged

Caché con Índices Virtuales y Etiquetas Físicas (VIPT: Virtually Indexed Physically Tagged)

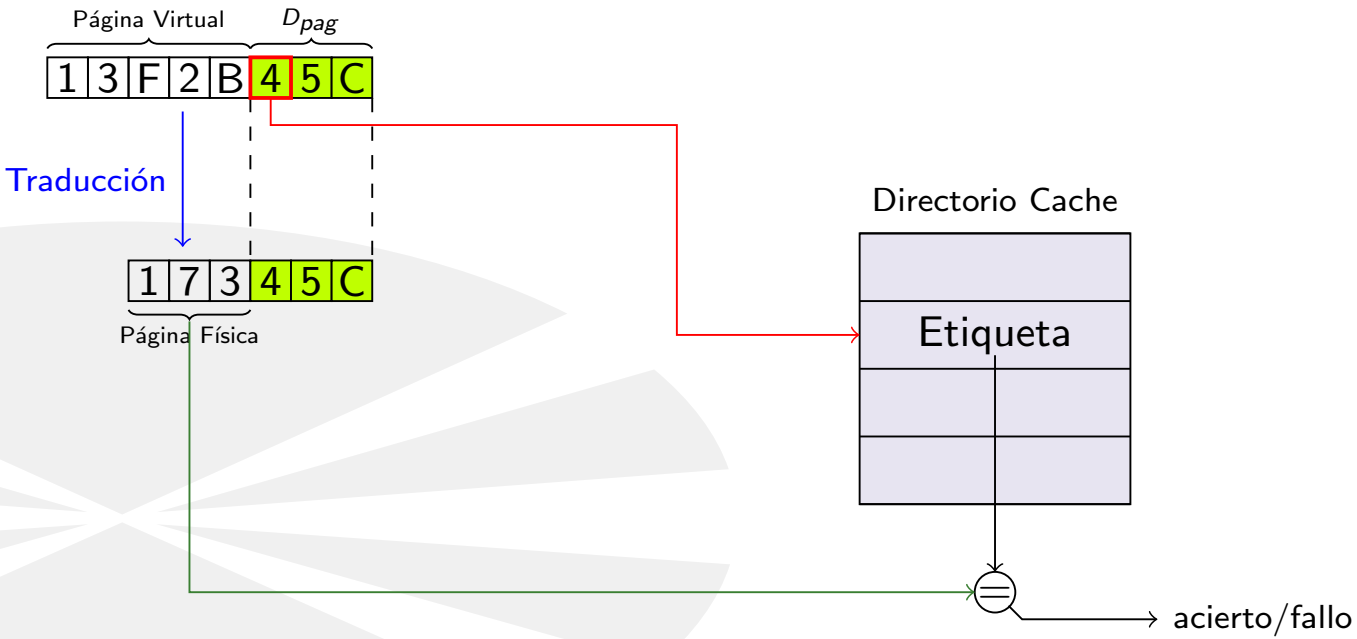
- En la traducción de Direcciones Virtuales, en realidad sólo se traduce la Página Virtual a Página Física
- El desplazamiento es idéntico, y por tanto forma parte de la Dirección Física
- La Dirección Física se divide en Etiqueta, Índice y Desplazamiento para acceder a la caché
- Si la caché está organizada de tal modo que el índice caiga dentro del desplazamiento de página, conocemos índice y desplazamiento caché antes de hacer la traducción
- Entonces, podemos buscar la línea caché con la dirección virtual, y luego verificar la etiqueta



Sistemas Paginados - Virtually Indexed Physically Tagged

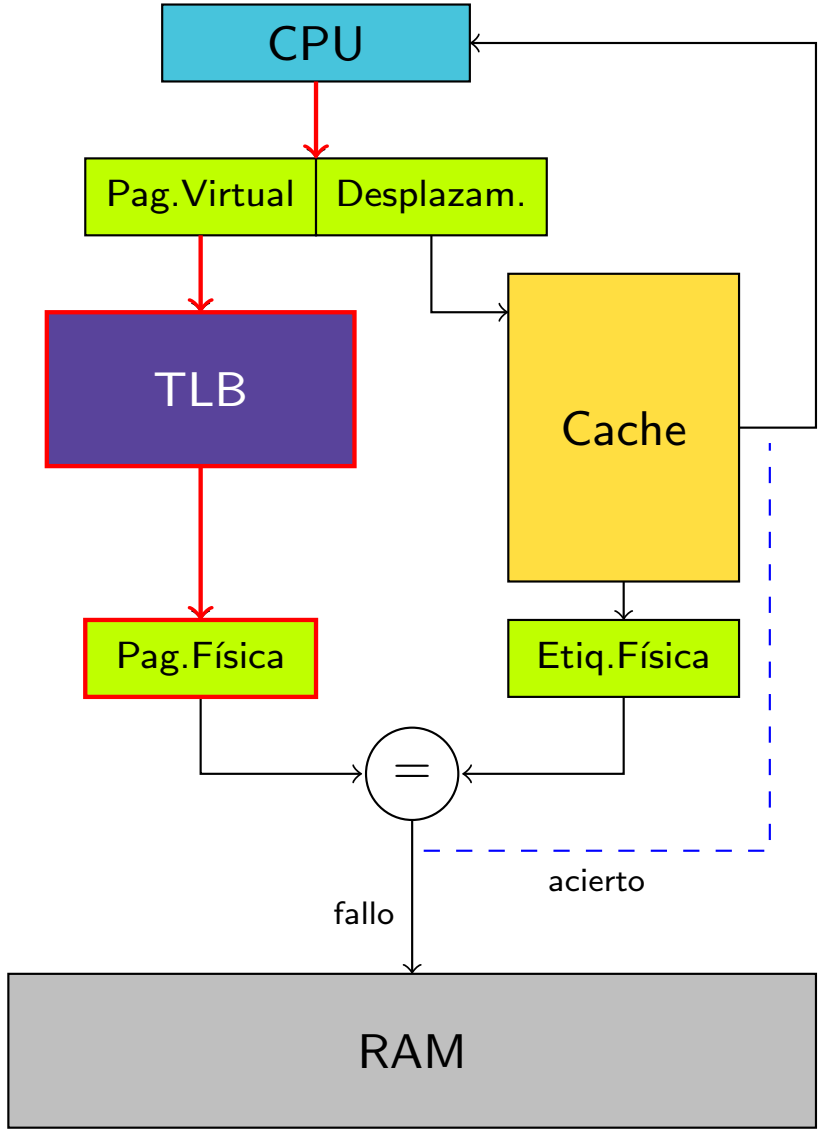
Caché con Índices Virtuales y Etiquetas Físicas (VIPT: Virtually Indexed Physically Tagged)

- En la traducción de Direcciones Virtuales, en realidad sólo se traduce la Página Virtual a Página Física
- El desplazamiento es idéntico, y por tanto forma parte de la Dirección Física
- La Dirección Física se divide en Etiqueta, Índice y Desplazamiento para acceder a la caché
- Si la caché está organizada de tal modo que el índice caiga dentro del desplazamiento de página, conocemos índice y desplazamiento caché antes de hacer la traducción
- Entonces, podemos buscar la línea caché con la dirección virtual, y luego verificar la etiqueta



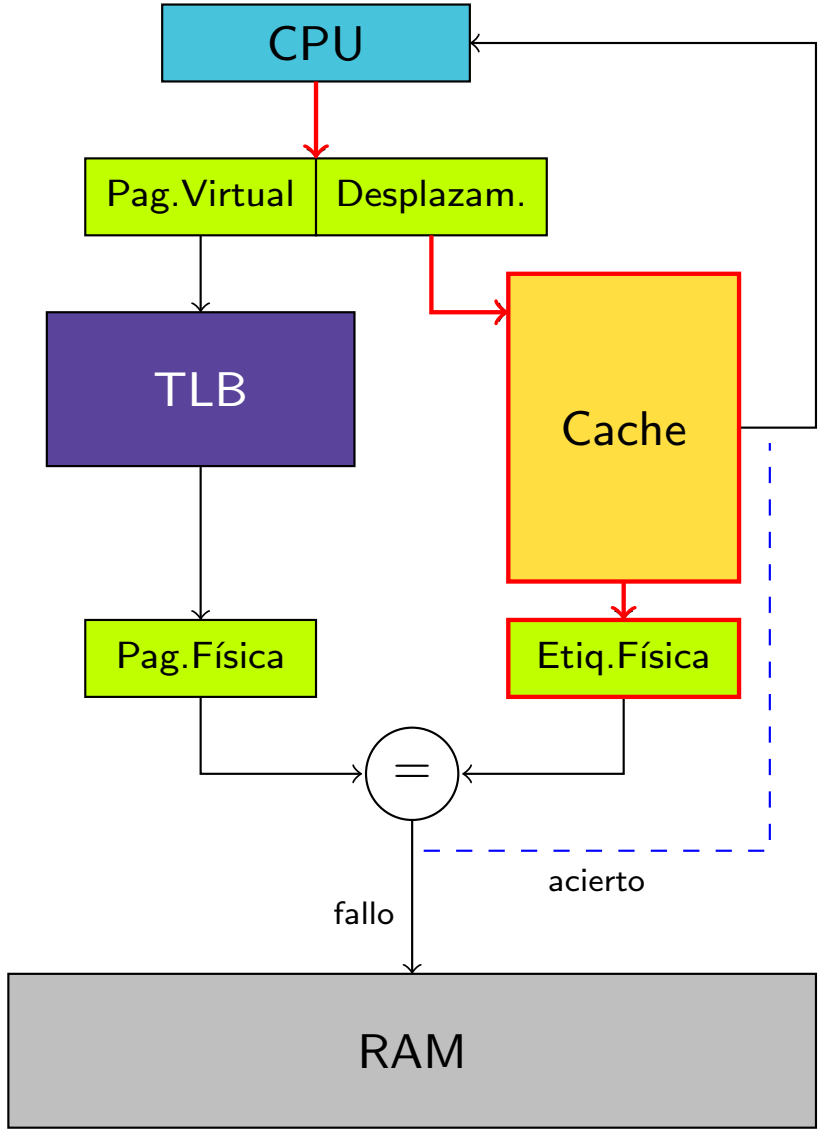
Sistemas Paginados - Virtually Indexed Physically Tagged

1. Buscamos en la TLB y en la caché **al mismo tiempo**
   1. La TLB se indexa con la **Página Virtual** y se obtiene la **Página Física**
   2. La caché se indexa con el **Desplazamiento** y se obtiene la **Etiqueta Física** almacenada en el directorio caché
2. Si la página y la etiqueta coinciden, tenemos un acierto y la caché envía el dato a la CPU



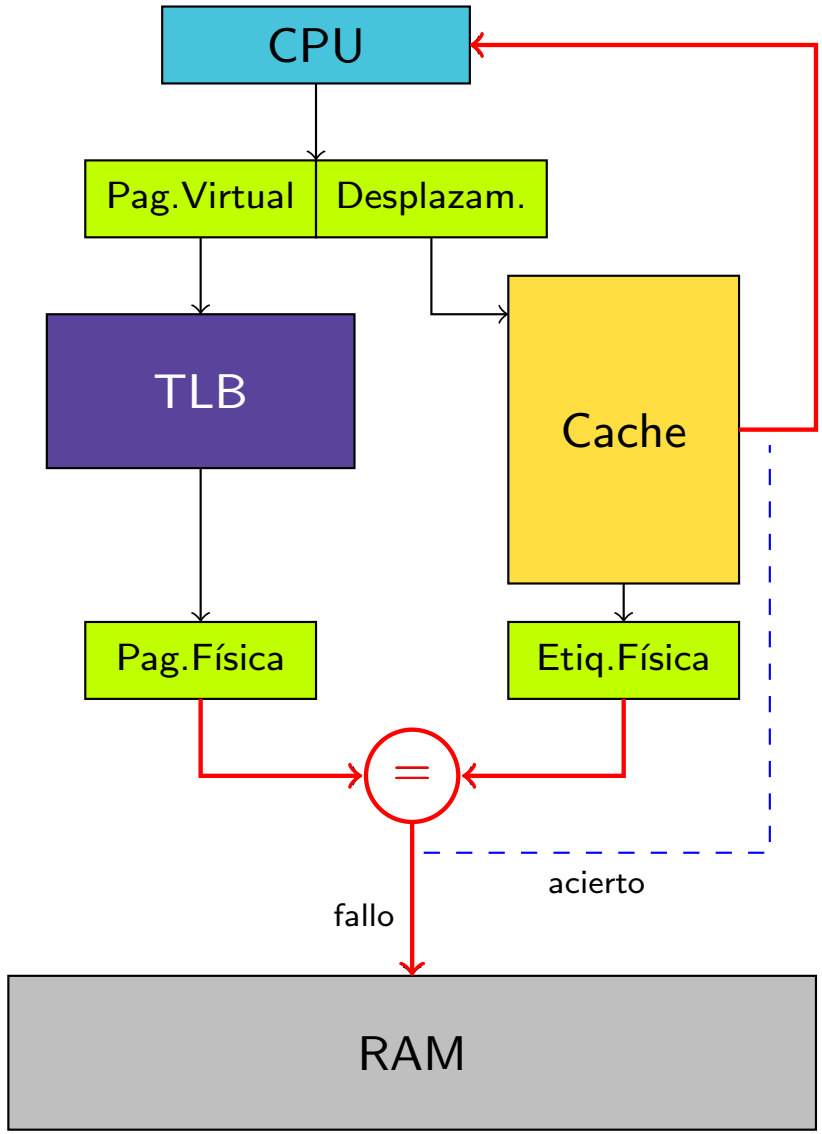
Sistemas Paginados - Virtually Indexed Physically Tagged

1. Buscamos en la TLB y en la caché **al mismo tiempo**
   1. La TLB se indexa con la **Página Virtual** y se obtiene la **Página Física**
   2. La caché se indexa con el **Desplazamiento** y se obtiene la **Etiqueta Física** almacenada en el directorio caché
2. Si la página y la etiqueta coinciden, tenemos un acierto y la caché envía el dato a la CPU

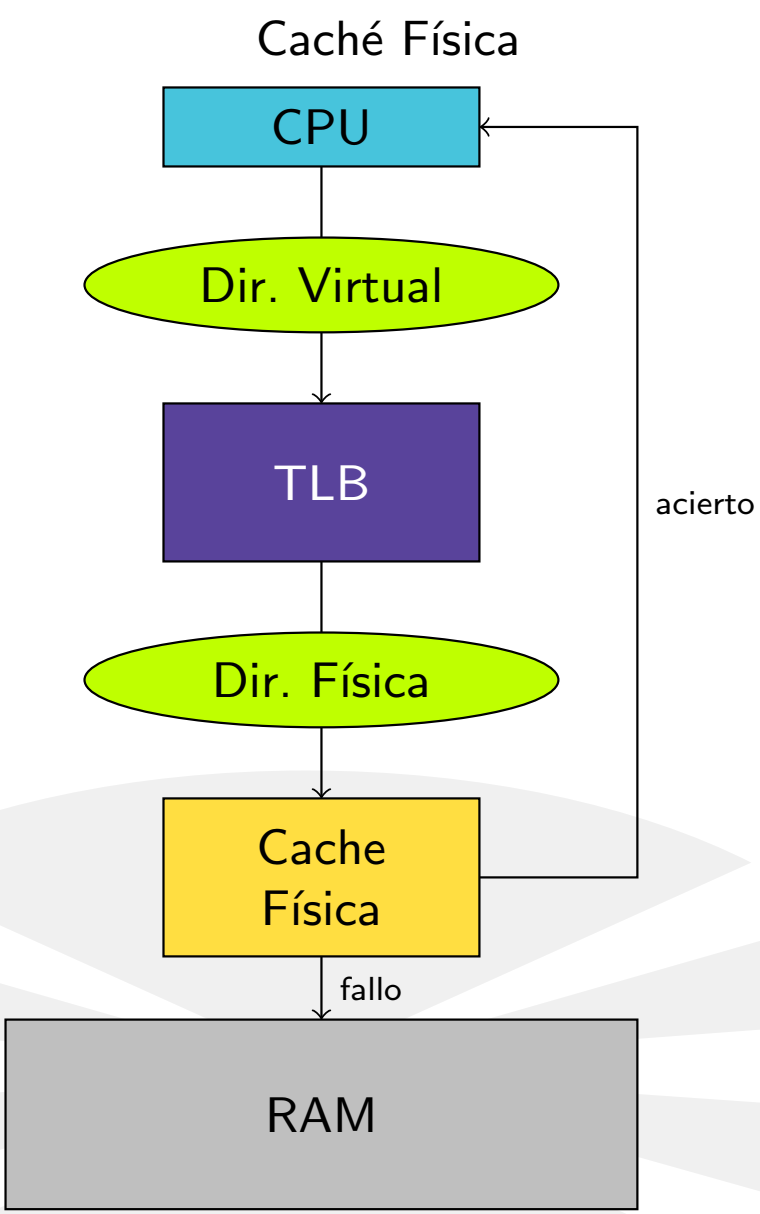


Sistemas Paginados - Virtually Indexed Physically Tagged

1. Buscamos en la TLB y en la caché **al mismo tiempo**
   1. La TLB se indexa con la **Página Virtual** y se obtiene la **Página Física**
   2. La caché se indexa con el **Desplazamiento** y se obtiene la **Etiqueta Física** almacenada en el directorio caché
2. Si la página y la etiqueta coinciden, tenemos un acierto y la caché envía el dato a la CPU



Sistemas Paginados - Virtually Indexed Physically Tagged



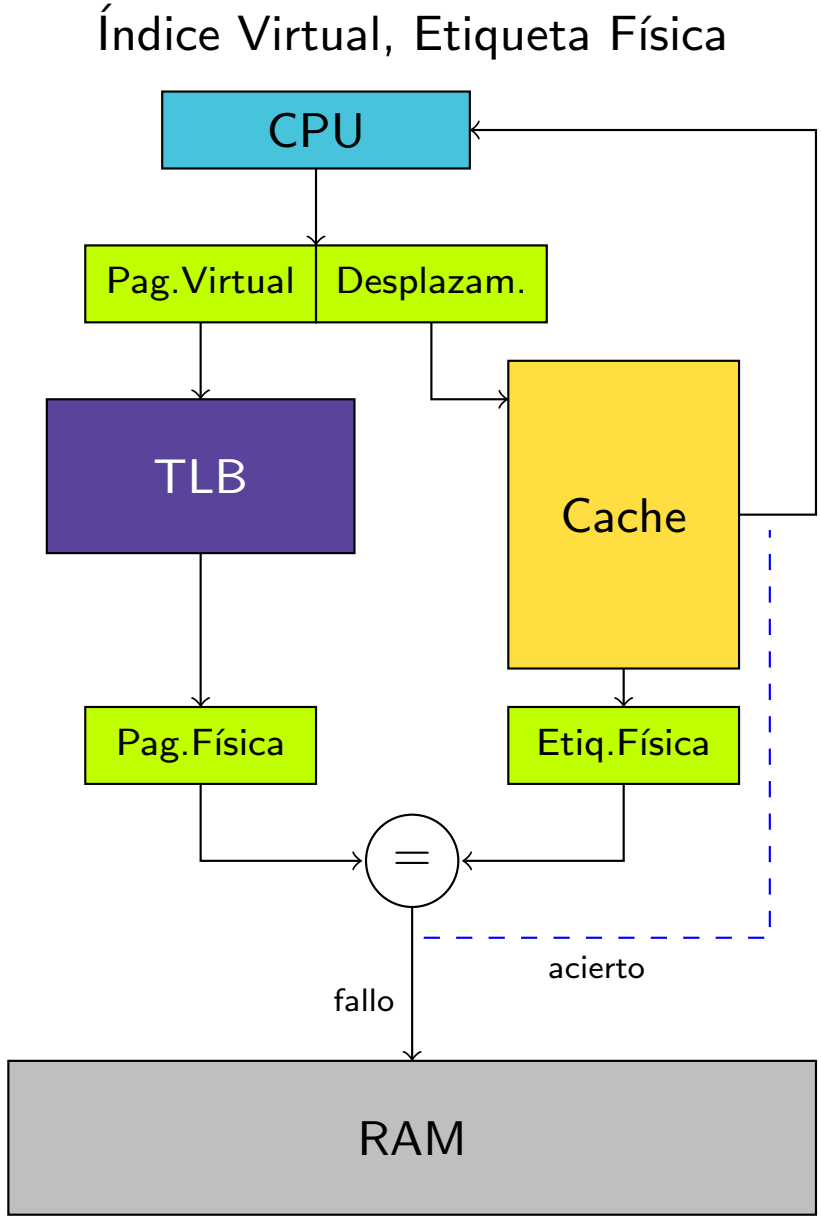
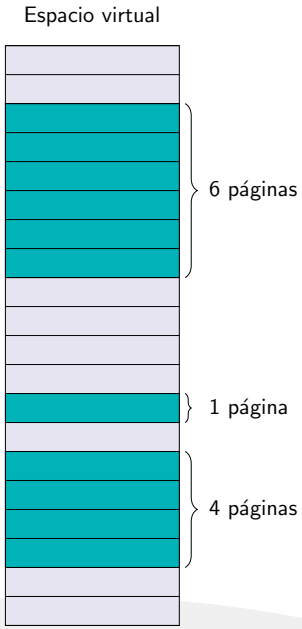


Table of Content

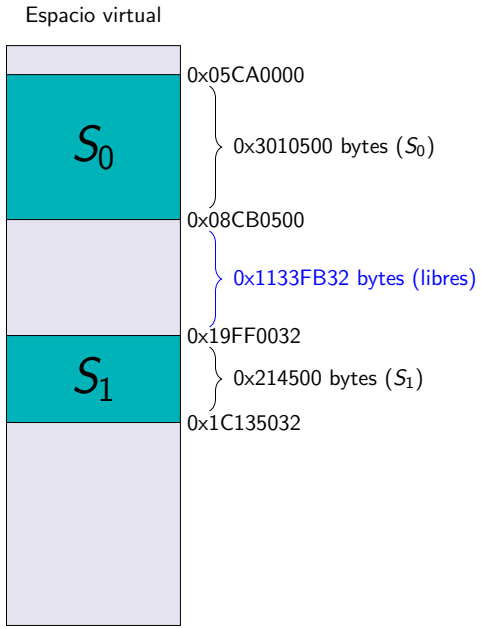
Sistemas Segmentados - Paginación vs Segmentacion

Sistema Paginado



- Las páginas tienen un tamaño fijo: no tienen relación con la estructura lógica del programa.
- Puede haber datos en una página no relacionados con los demás.
- En la tabla de páginas tenemos una entrada por cada página virtual

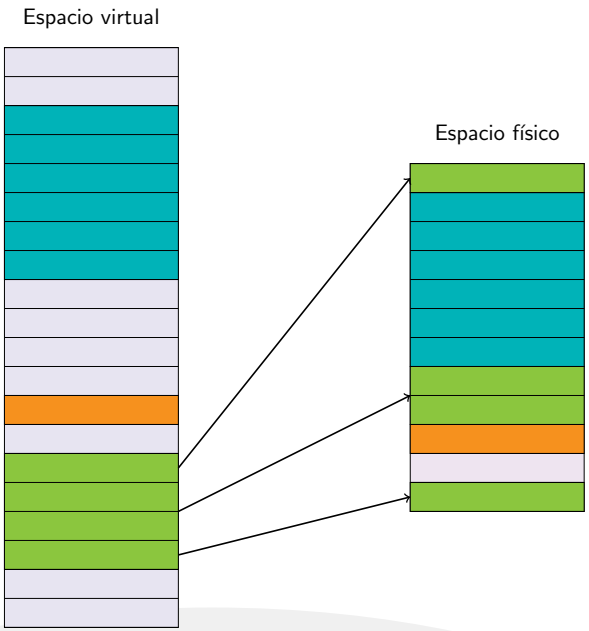
Sistema Segmentado



- Considera bloques de tamaño desigual, definidos en función de la estructura lógica del código y datos del programa
- Cada bloque se denomina segmento y tiene una **dirección base** y una **longitud**.
- Por ejemplo, S_0 Comienza en 0x05CA0000 y ocupa 0x3010500 bytes

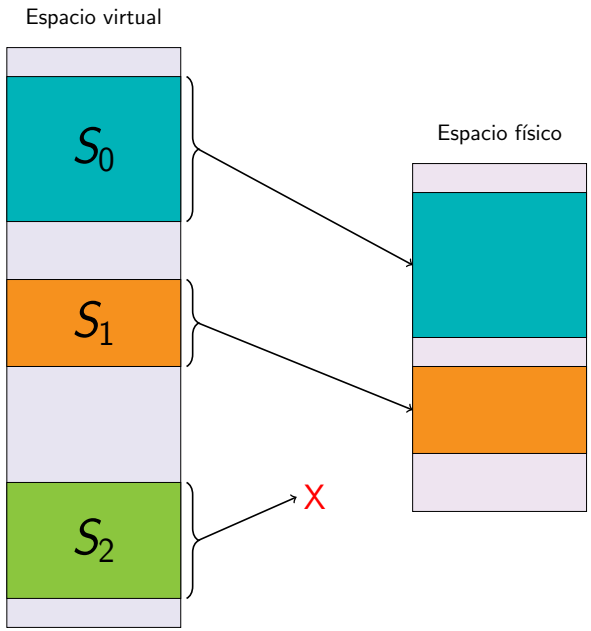
Sistemas Segmentados - Paginación vs Segmentación II

Sistema Paginado



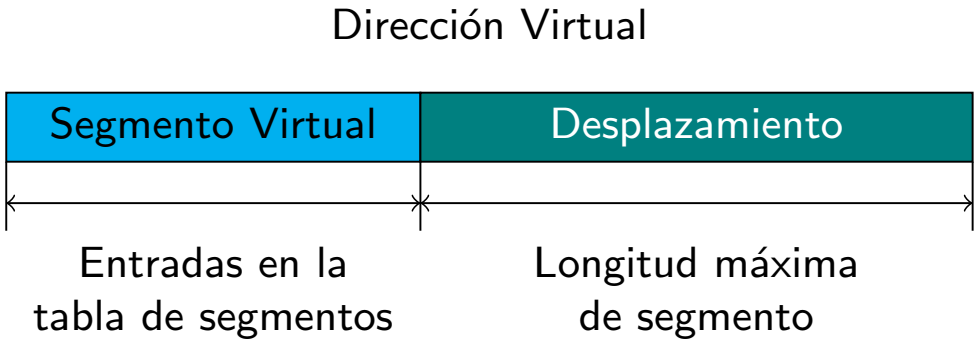
- Fragmentación interna: nos puede quedar parte de una página sin utilizar.
- No hay Fragmentación externa: direcciones virtuales consecutivas pueden ir a diferentes páginas físicas.
- Ineficiencia por el principio de localidad.

Sistema Segmentado



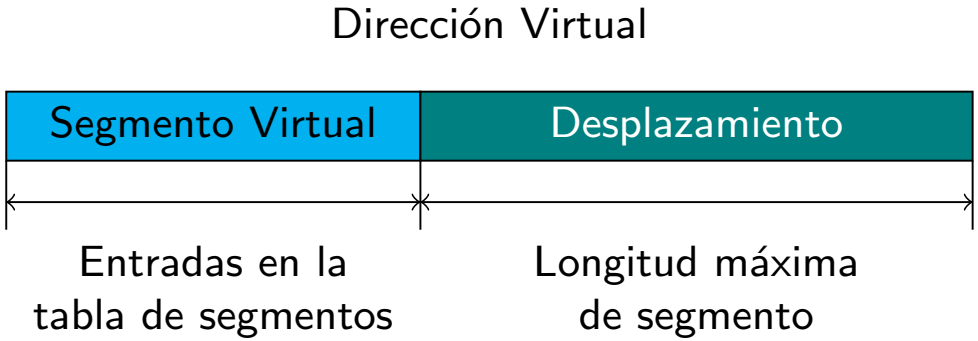
- Reduce la Fragmentación interna: Los segmentos tienen el tamaño necesario.
- Fragmentación externa: puede que los huecos libres entre segmentos no sean suficientes para alojar un nuevo segmento.

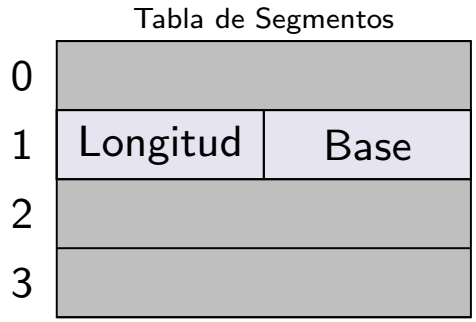
Sistemas Segmentados - Definiciones



- Direcciones virtuales:
  1. **Número de segmento virtual**: son los p bits más significativos de la dirección si la tabla de segmentos contiene 2^p entradas.
  2. **Desplazamiento** dentro del segmento: son los q bits menos significativos de la dirección si el tamaño máximo de segmento es 2^q bytes.

Sistemas Segmentados - Definiciones

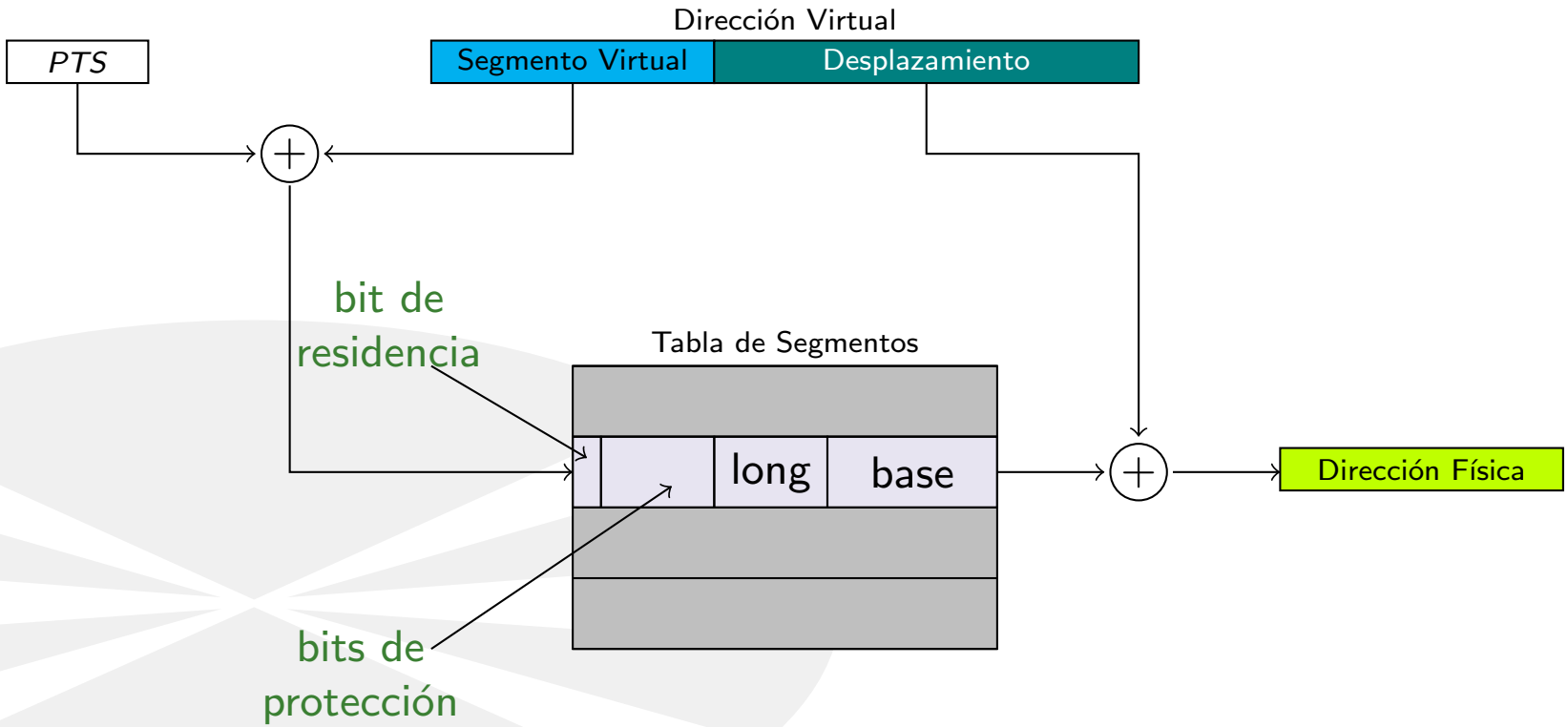




- Tabla de segmentos:
  - Contiene información para traducir el número de segmento a una dirección física de comienzo del segmento
  - Sus entradas contienen información similar a la contenida en las entradas de las tablas de páginas
  - **Longitud del segmento:** Cantidad de memoria reservada para el segmento
  - **Base:** Dirección física donde comienza el segmento
- Otros términos:
  - **Fallo de segmento**: El segmento no se encuentra en memoria principal. Análogo al fallo de página.
  - **Violación de segmento**: SIGSEGV / Segmentation Fault. Acceso a memoria fuera del segmento.

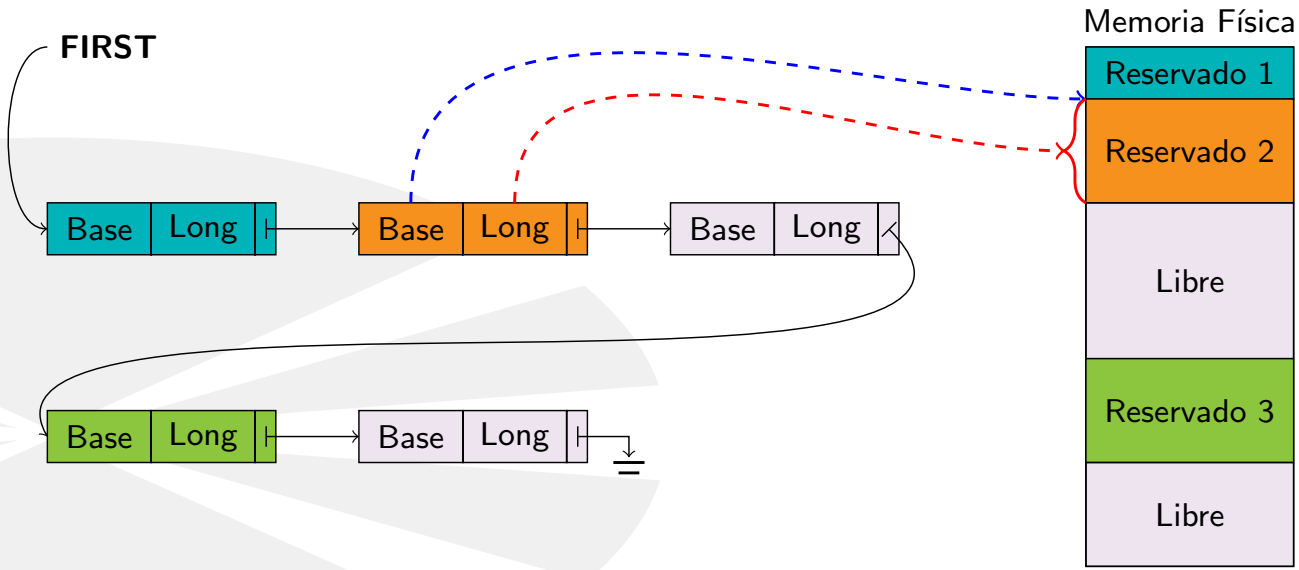
Sistemas Segmentados - Traducción

- Tipos de traducción:
  - Los mismos que en los sistemas paginados: directa, asociativa o híbrida
  - Diferencia: el desplazamiento **se suma**, no se concatena
- Ejemplo con traducción directa:
  - **PTS** (Puntero a la Tabla de Segmentos): contiene la dirección física de comienzo de la tabla de segmentos
  - Se chequea el desplazamiento contra la longitud para comprobar que la dirección virtual se encuentra dentro del segmento referenciado
  - Mayor facilidad de protección (entidad lógica)



Sistemas Segmentados - Fallo de Segmento

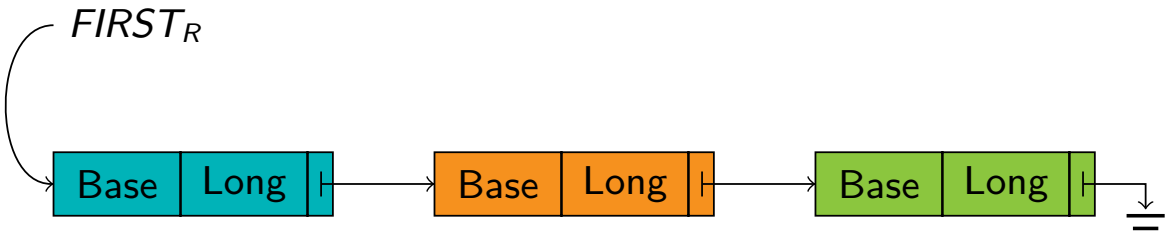
- Referencia a un segmento no residente en la memoria principal
- Algoritmos de carga similares a los de sistemas paginados: bajo demanda o precarga
- Algoritmos de ubicación más complejos
- Es necesario determinar si hay espacio de memoria principal para ubicar el segmento virtual fallado (este problema no existe en los sistemas paginados)
- Estructurar la memoria principal: se describen los segmentos en una lista enlazada



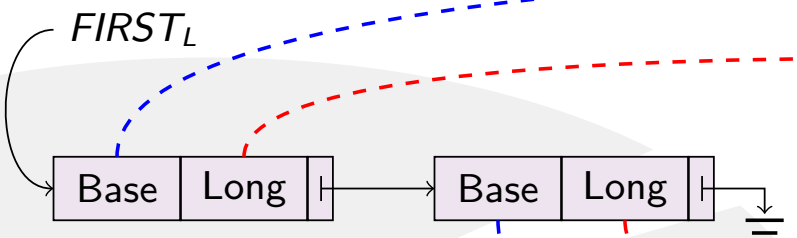
Sistemas Segmentados - Listas Enlazadas

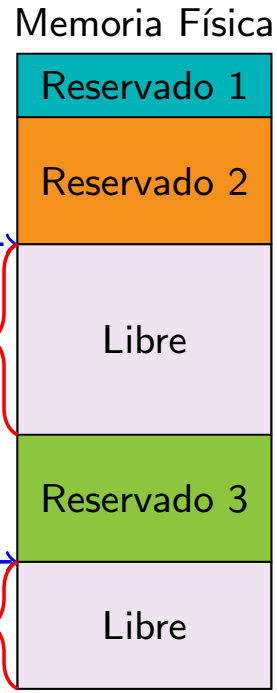
- Se acelera la asignación usando dos listas enlazadas:

a) **Lista de segmentos reservados**



b) **Lista de segmentos libres (LAVS)**





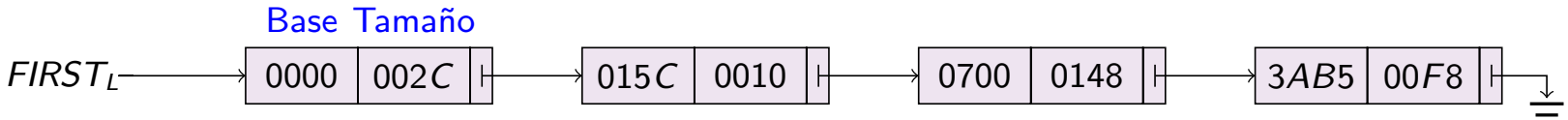
Sistemas Segmentados - Algoritmos de Ubicación

Los algoritmos de ubicación se basan en la lista de segmentos libres (LAVS)

- First-Fit
  - El segmento se ubica en el primer segmento de memoria principal que tenga un tamaño suficiente
- Best-Fit
  - El segmento se ubica en el segmento más pequeño que tenga un tamaño suficiente
- Worst-Fit
  - El segmento se ubica en el segmento más grande
- Binary-Buddy
  - Subdivisión binaria

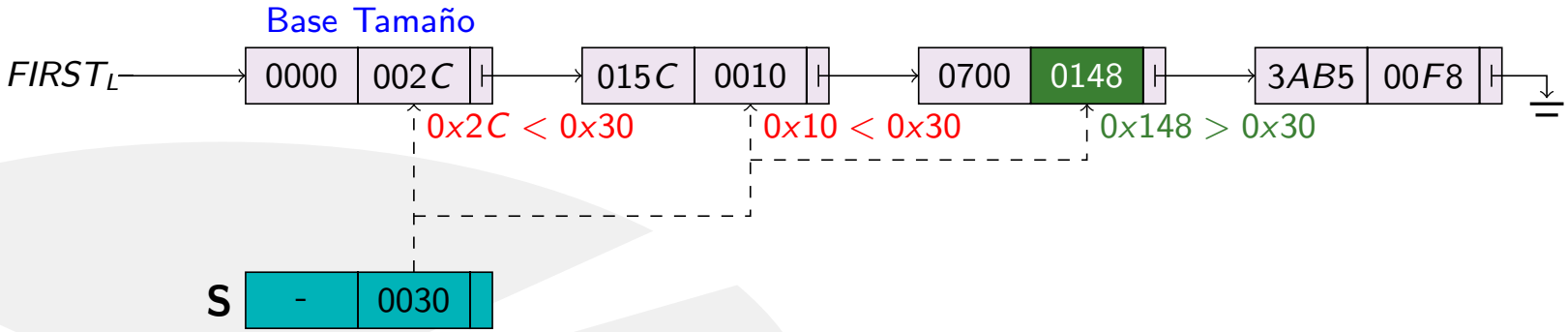
Sistemas Segmentados - First Fit

- First-Fit
  - LAVS ordenada por posición en memoria física
  - Se recorre la lista hasta encontrar un segmento con tamaño suficiente ($T(Q) > T(S)$)
  - El segmento se ubica **al final** del primer segmento de memoria principal donde quepa
  - Se actualiza la lista reduciendo el tamaño del segmento utilizado (o eliminándolo si se ocupa entero: si T(S) = T(Q)



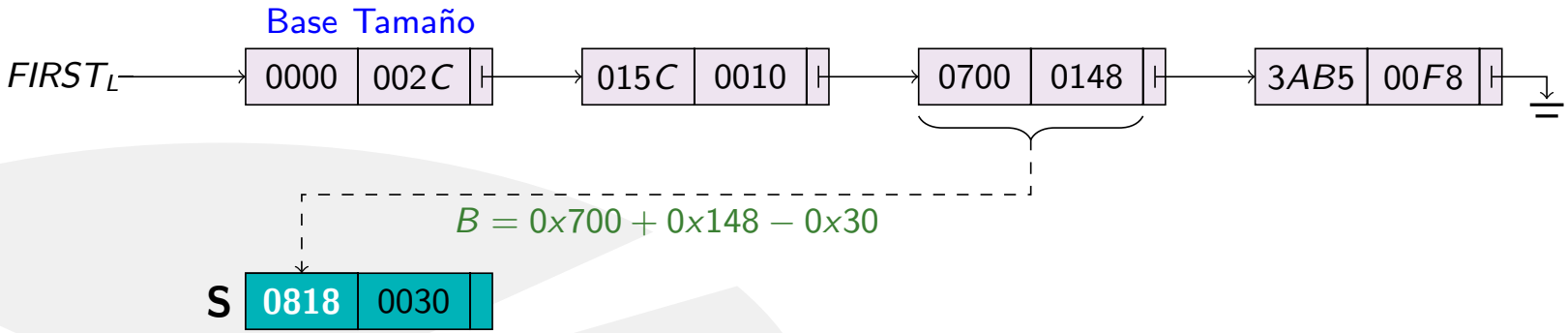
Sistemas Segmentados - First Fit

- First-Fit
  - LAVS ordenada por posición en memoria física
  - Se recorre la lista hasta encontrar un segmento con tamaño suficiente ($T(Q) > T(S)$)
  - El segmento se ubica **al final** del primer segmento de memoria principal donde quepa
  - Se actualiza la lista reduciendo el tamaño del segmento utilizado (o eliminándolo si se ocupa entero: si T(S) = T(Q)



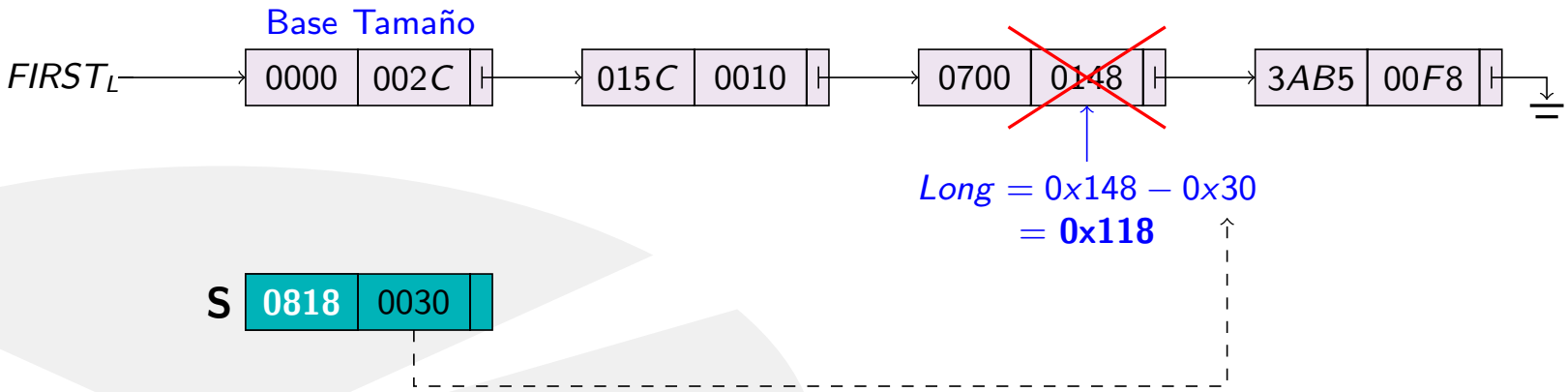
Sistemas Segmentados - First Fit

- First-Fit
  - LAVS ordenada por posición en memoria física
  - Se recorre la lista hasta encontrar un segmento con tamaño suficiente ($T(Q) > T(S)$)
  - El segmento se ubica **al final** del primer segmento de memoria principal donde quepa
  - Se actualiza la lista reduciendo el tamaño del segmento utilizado (o eliminándolo si se ocupa entero: si T(S) = T(Q)



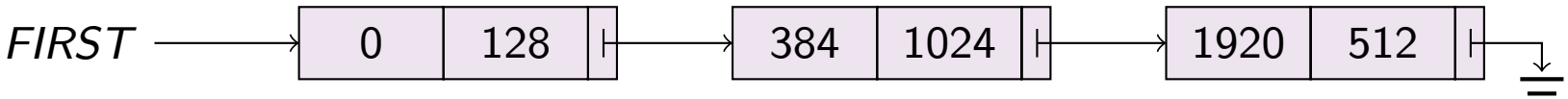
Sistemas Segmentados - First Fit

- First-Fit
  - LAVS ordenada por posición en memoria física
  - Se recorre la lista hasta encontrar un segmento con tamaño suficiente ($T(Q) > T(S)$)
  - El segmento se ubica **al final** del primer segmento de memoria principal donde quepa
  - Se actualiza la lista reduciendo el tamaño del segmento utilizado (o eliminándolo si se ocupa entero: si T(S) = T(Q)



Sistemas Segmentados - Ejemplo

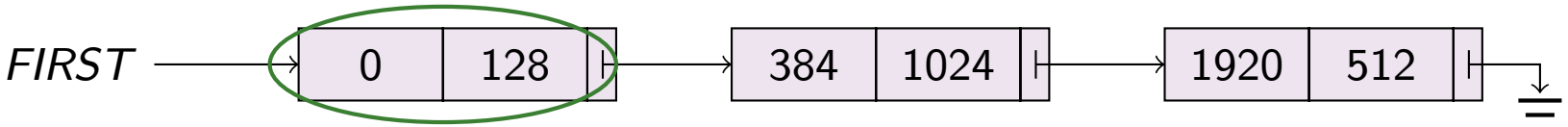
Ejemplo: Un sistema tiene la siguiente lista de segmentos libres



Se solicitan los segmentos A (tamaño 64) y B (tamaño 256)

Sistemas Segmentados - Ejemplo

Ejemplo: Un sistema tiene la siguiente lista de segmentos libres

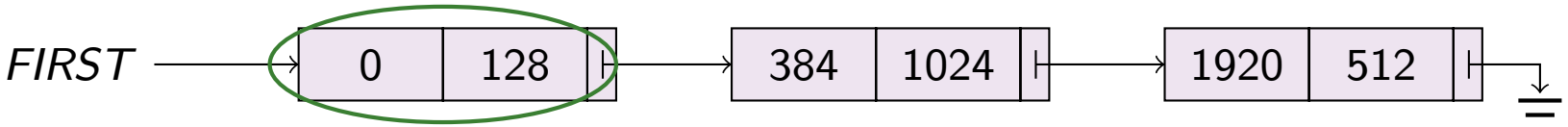


Se solicitan los segmentos A (tamaño 64) y B (tamaño 256)

1. Recorremos la lista buscando un segmento libre Q_1 de al menos 64 bytes: Q_1 = primer segmento
2. Calculamos la dirección base de A:
   $base(A) = base(Q_1) + long(Q_1) - long(A) = 0 + 128 - 64 = 64$
3. Calculamos el nuevo tamaño de Q_1: $long(Q_1) \leftarrow long(Q_1) - long(A) = 128 - 64 = 64$
4. Recorremos la lista buscando un segmento libre Q_2 de al menos 256 bytes: Q_2 = segundo segmento
5. Calculamos la dirección base de B:
   $base(B) = base(Q_2) + long(Q_2) - long(A) = 384 + 1024 - 256 = 1152$
6. Calculamos el nuevo tamaño de Q_2: $long(Q_2) = long(Q_2) - long(B) = 1024 - 256 = 768$

Sistemas Segmentados - Ejemplo

Ejemplo: Un sistema tiene la siguiente lista de segmentos libres

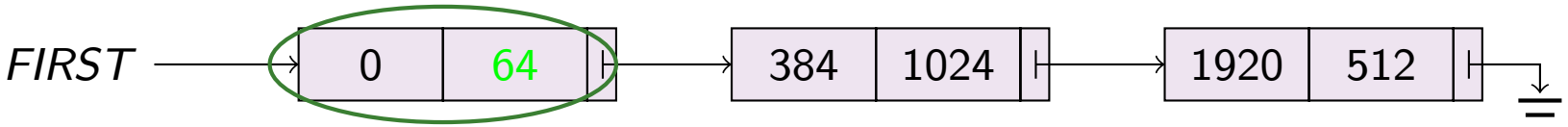


Se solicitan los segmentos A (tamaño 64) y B (tamaño 256)

1. Recorremos la lista buscando un segmento libre Q_1 de al menos 64 bytes: Q_1 = primer segmento
2. Calculamos la dirección base de A:
   $base(A) = base(Q_1) + long(Q_1) - long(A) = 0 + 128 - 64 = 64$
3. Calculamos el nuevo tamaño de Q_1: $long(Q_1) \leftarrow long(Q_1) - long(A) = 128 - 64 = 64$
4. Recorremos la lista buscando un segmento libre Q_2 de al menos 256 bytes: Q_2 = segundo segmento
5. Calculamos la dirección base de B:
   $base(B) = base(Q_2) + long(Q_2) - long(A) = 384 + 1024 - 256 = 1152$
6. Calculamos el nuevo tamaño de Q_2: $long(Q_2) = long(Q_2) - long(B) = 1024 - 256 = 768$

Sistemas Segmentados - Ejemplo

Ejemplo: Un sistema tiene la siguiente lista de segmentos libres

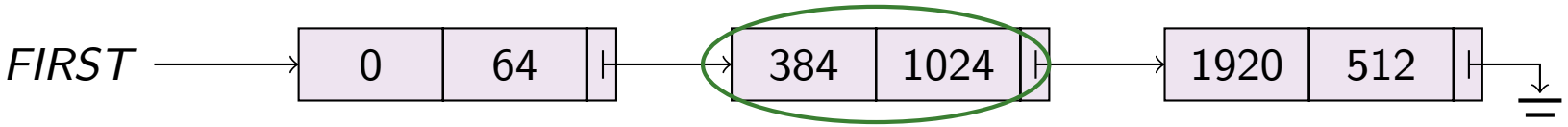


Se solicitan los segmentos A (tamaño 64) y B (tamaño 256)

1. Recorremos la lista buscando un segmento libre Q_1 de al menos 64 bytes: Q_1 = primer segmento
2. Calculamos la dirección base de A:
   $base(A) = base(Q_1) + long(Q_1) - long(A) = 0 + 128 - 64 = 64$
3. Calculamos el nuevo tamaño de Q_1: $long(Q_1) \leftarrow long(Q_1) - long(A) = 128 - 64 = 64$
4. Recorremos la lista buscando un segmento libre Q_2 de al menos 256 bytes: Q_2 = segundo segmento
5. Calculamos la dirección base de B:
   $base(B) = base(Q_2) + long(Q_2) - long(A) = 384 + 1024 - 256 = 1152$
6. Calculamos el nuevo tamaño de Q_2: $long(Q_2) = long(Q_2) - long(B) = 1024 - 256 = 768$

Sistemas Segmentados - Ejemplo

Ejemplo: Un sistema tiene la siguiente lista de segmentos libres

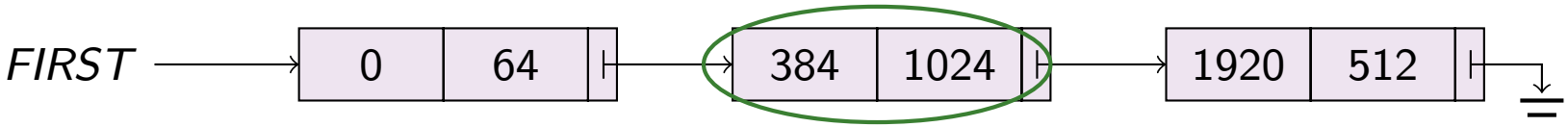


Se solicitan los segmentos A (tamaño 64) y B (tamaño 256)

1. Recorremos la lista buscando un segmento libre Q_1 de al menos 64 bytes: Q_1 = primer segmento
2. Calculamos la dirección base de A:
   $base(A) = base(Q_1) + long(Q_1) - long(A) = 0 + 128 - 64 = 64$
3. Calculamos el nuevo tamaño de Q_1: $long(Q_1) \leftarrow long(Q_1) - long(A) = 128 - 64 = 64$
4. Recorremos la lista buscando un segmento libre Q_2 de al menos 256 bytes: Q_2 = segundo segmento
5. Calculamos la dirección base de B:
   $base(B) = base(Q_2) + long(Q_2) - long(A) = 384 + 1024 - 256 = 1152$
6. Calculamos el nuevo tamaño de Q_2: $long(Q_2) = long(Q_2) - long(B) = 1024 - 256 = 768$

Sistemas Segmentados - Ejemplo

Ejemplo: Un sistema tiene la siguiente lista de segmentos libres

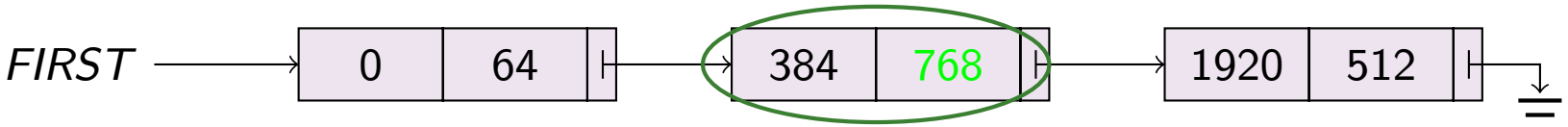


Se solicitan los segmentos A (tamaño 64) y B (tamaño 256)

1. Recorremos la lista buscando un segmento libre Q_1 de al menos 64 bytes: Q_1 = primer segmento
2. Calculamos la dirección base de A:
   $base(A) = base(Q_1) + long(Q_1) - long(A) = 0 + 128 - 64 = 64$
3. Calculamos el nuevo tamaño de Q_1: $long(Q_1) \leftarrow long(Q_1) - long(A) = 128 - 64 = 64$
4. Recorremos la lista buscando un segmento libre Q_2 de al menos 256 bytes: Q_2 = segundo segmento
5. Calculamos la dirección base de B:
   $base(B) = base(Q_2) + long(Q_2) - long(A) = 384 + 1024 - 256 = 1152$
6. Calculamos el nuevo tamaño de Q_2: $long(Q_2) = long(Q_2) - long(B) = 1024 - 256 = 768$

Sistemas Segmentados - Ejemplo

Ejemplo: Un sistema tiene la siguiente lista de segmentos libres

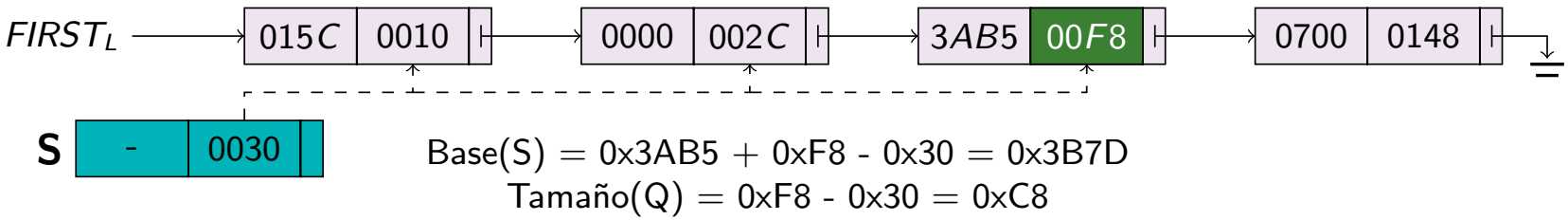


Se solicitan los segmentos A (tamaño 64) y B (tamaño 256)

1. Recorremos la lista buscando un segmento libre Q_1 de al menos 64 bytes: Q_1 = primer segmento
2. Calculamos la dirección base de A:
   $base(A) = base(Q_1) + long(Q_1) - long(A) = 0 + 128 - 64 = 64$
3. Calculamos el nuevo tamaño de Q_1: $long(Q_1) \leftarrow long(Q_1) - long(A) = 128 - 64 = 64$
4. Recorremos la lista buscando un segmento libre Q_2 de al menos 256 bytes: Q_2 = segundo segmento
5. Calculamos la dirección base de B:
   $base(B) = base(Q_2) + long(Q_2) - long(A) = 384 + 1024 - 256 = 1152$
6. Calculamos el nuevo tamaño de Q_2: $long(Q_2) = long(Q_2) - long(B) = 1024 - 256 = 768$

Sistemas Segmentados - Best-Fit/Worst-Fit

- Best-Fit
  - Igual que FIRST-FIT pero LAVS ordenada **ascendentemente** por tamaño de los segmentos libres
  - Asignar un segmento puede requerir **reordenar** la lista

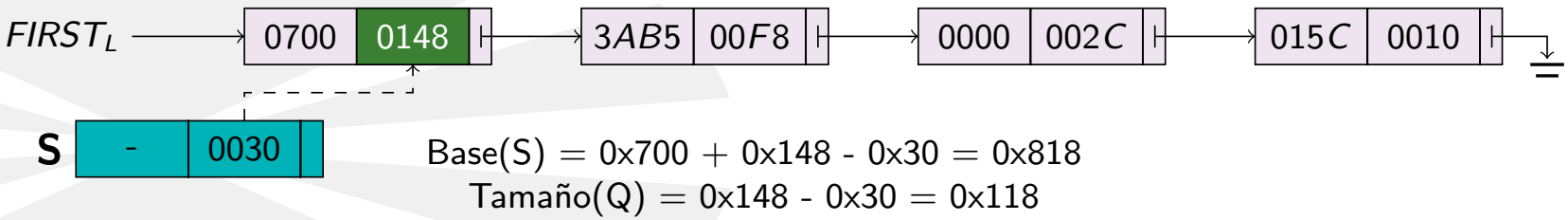


- Worst-Fit
  - Igual que FIRST-FIT pero LAVS ordenada **descendentemente** por tamaño de los segmentos libres.
  - Asignar un segmento puede requerir **reordenar** la lista.

Sistemas Segmentados - Best-Fit/Worst-Fit

- Best-Fit
  - Igual que FIRST-FIT pero LAVS ordenada **ascendentemente** por tamaño de los segmentos libres
  - Asignar un segmento puede requerir **reordenar** la lista

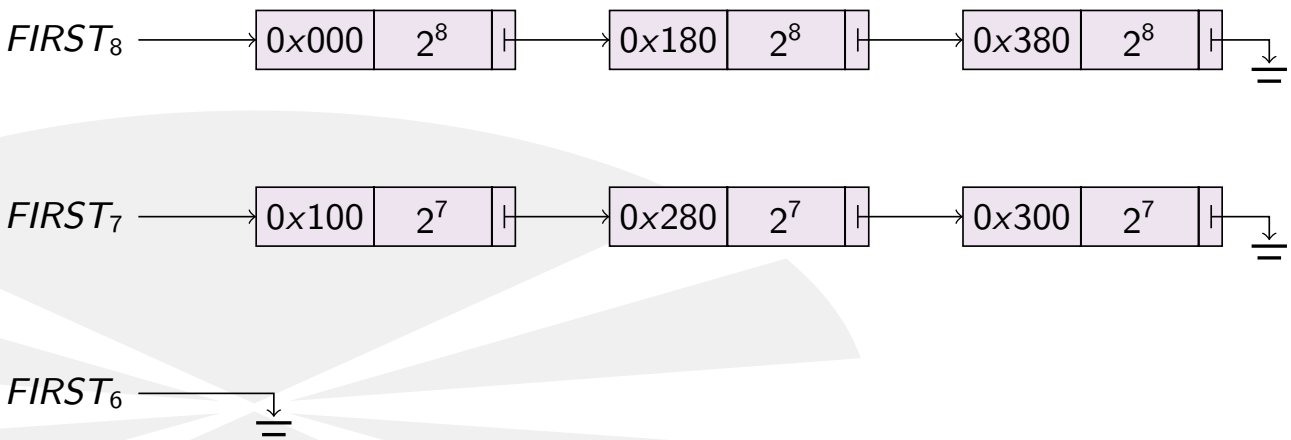
- Worst-Fit
  - Igual que FIRST-FIT pero LAVS ordenada **descendentemente** por tamaño de los segmentos libres.
  - Asignar un segmento puede requerir **reordenar** la lista.



Sistemas Segmentados - Binary Buddy

Binary Buddy

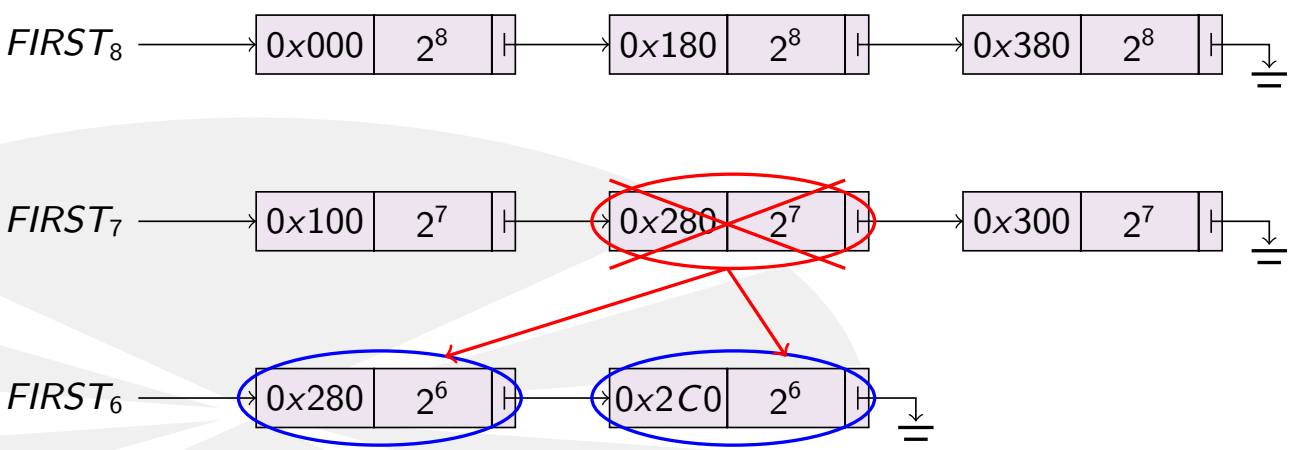
- Tamaños de segmento **potencia entera de 2:** 2^i
- Procedimiento de ubicación:
  - Se divide la lista LAVS en n listas (2^n es el tamaño máximo de segmento)
  - La i-ésima lista encadena los segmentos libres de tamaño 2^i (buddies)
  - Las n listas pueden reestructurarse de dos formas:
    1. Un segmento libre de la lista i se divide en 2 segmentos iguales y se encadenan a la lista $i-1$
    2. Dos segmentos libres consecutivos de la lista i pueden juntarse y moverse a la lista $i+1$



Sistemas Segmentados - Binary Buddy

Binary Buddy

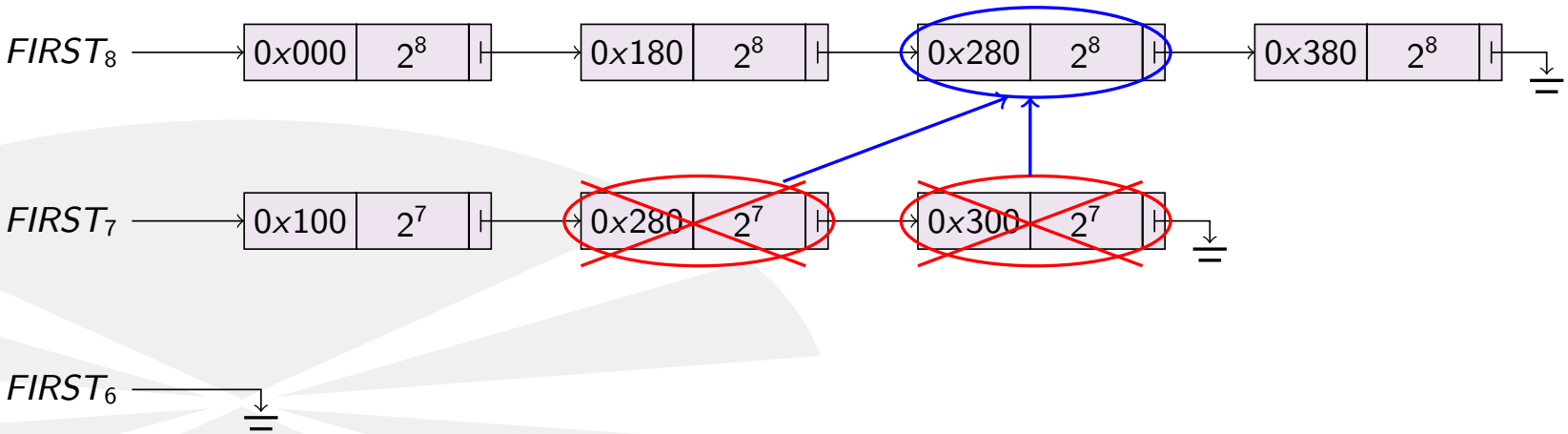
- Tamaños de segmento **potencia entera de 2:** 2^i
- Procedimiento de ubicación:
  - Se divide la lista LAVS en n listas (2^n es el tamaño máximo de segmento)
  - La i-ésima lista encadena los segmentos libres de tamaño 2^i (buddies)
  - Las n listas pueden reestructurarse de dos formas:
    1. Un segmento libre de la lista i se divide en 2 segmentos iguales y se encadenan a la lista $i-1$
    2. Dos segmentos libres consecutivos de la lista i pueden juntarse y moverse a la lista $i+1$



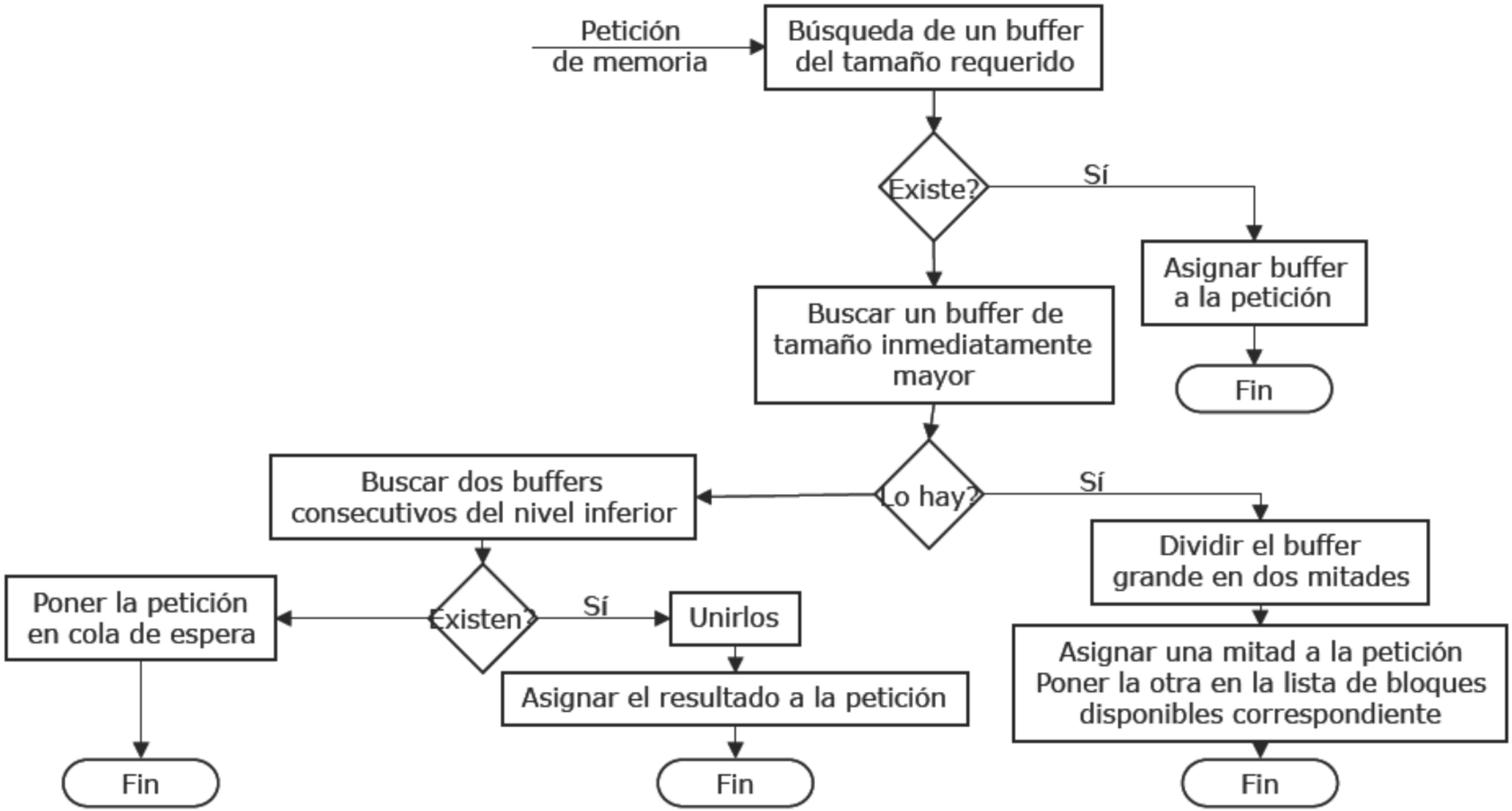
Sistemas Segmentados - Binary Buddy

Binary Buddy

- Tamaños de segmento **potencia entera de 2:** 2^i
- Procedimiento de ubicación:
  - Se divide la lista LAVS en n listas (2^n es el tamaño máximo de segmento)
  - La i-ésima lista encadena los segmentos libres de tamaño 2^i (buddies)
  - Las n listas pueden reestructurarse de dos formas:
    1. Un segmento libre de la lista i se divide en 2 segmentos iguales y se encadenan a la lista $i-1$
    2. Dos segmentos libres consecutivos de la lista i pueden juntarse y moverse a la lista $i+1$



Sistemas Segmentados - Binary Buddy



Sistemas Segmentados - Algoritmos de Ubicación

COMPARACIÓN:

- Algoritmo Best-Fit:
  - Minimiza el tamaño del segmento libre sobrante tras la ubicación
  - Luego puede generar mucha **fragmentación externa**
- Algoritmo Worst-Fit:
  - Basado en la idea de que, tras la ubicación, el segmento libre sobrante es suficientemente grande como para ser útil
- Algoritmo Binary-Buddy:
  - Reduce la **fragmentación externa**
  - Puede provocar mucha **fragmentación interna**
    - Solución: Slab allocation
- Algoritmos más eficientes: First-Fit y Binary Buddy

Sistemas Segmentados - Compactación o reemplazo

Cuando se solicita un segmento **no residente** pueden darse dos situaciones:

1. Se encuentra un segmento libre de tamaño suficiente: ubicación
2. No se encuentra tal segmento libre
   - **Compactación:** se disponen los segmentos libres de forma contigua en una zona de la memoria física creándose un único segmento libre de gran tamaño
     - Gran consumo de tiempo
     - Actualización de las tablas de segmentos o listas de segmentos residentes
   - **Reemplazo:** similar a los sistemas paginados
     - Debe tenerse en cuenta el tamaño del segmento solicitado
     - Un problema es el uso poco eficiente de la memoria física
     - El segmento reemplazado puede necesitarse en un futuro próximo
   - La elección entre compactación y reemplazo no es simple, depende de cada situación particular

Sistemas Segmentados - Paginación vs Segmentación

	Página	**Segmento**
Palabras por dirección	Una	Dos (segmento y desplazamiento)
Visibilidad	Invisible a la aplicación	Puede ser visible a la aplicación
Reemplazo	Trivial (todos los bloques son iguales)	Difícil (debe haber espacio contiguo)
Eficiencia de memoria	Fragmentación interna	Fragmentación externa
Eficiencia de disco	Sí. El tamaño de página se ajusta para equilibrar tiempo de acceso y transferencia	No necesariamente. Puede haber segmentos demasiado pequeños

Jerarquía de Memoria - Comparación

	TLB	Cache 1er nivel	Cache 2º nivel	Memoria virtual
Tamaño de bloque	4-8 bytes (1 PTE)	4-32 bytes	32-256 bytes	512-16,384 bytes
Tiempo de acierto	1 ciclo de reloj	1-2 ciclos de reloj	6-15 ciclos de reloj	10-100 ciclos de reloj
Penalización de fallo	10-30 ciclos de reloj	8-66 ciclos de reloj	30-200 ciclos de reloj	700,000-6,000,000
Tasa de fallos (local)	0.1-2%	0.5-20%	15-30%	0.00001-0.001%
Tamaño	32-8192 bytes (8-1024 PTEs)	1-128 KB	256KB-16MB	16-8192 MB
Nivel inferior	Cache de 1er nivel	Cache de 2º nivel	Memoria Principal (DRAM)	Discos
Q1: ubicación de bloques	Asociativa completamente o por conjuntos	Correspondencia directa o asociativa por conjuntos	Correspondencia directa o asociativa por conjuntos	Totalmente asociativa
Q2: identificación de bloques	Etiqueta/bloque	Etiqueta/bloque	Etiqueta/bloque	Tabla
Q3: reemplazo de bloques	Pseudo-LRU	Pseudo-LRU	Pseudo-LRU	≈ LRU
Q4: estrategia de escritura	Postescritura	Escritura directa o postescritura	Postescritura	Postescritura

Jerarquía de Memoria

Tiempos a escala

1 ciclo de CPU	0,3 ns	1 s
Acceso a caché de 1er nivel	0,9 ns	3 s
Acceso a caché de 2o nivel	2,8 ns	9 s
Acceso a caché de 3er nivel	12,9 ns	43 s
Acceso a memoria principal	120 ns	6 min
Acceso a disco SSD	50-150 μs	2-6 días
Acceso a disco rotacional	1-10 ms	1-12 meses

Fuente: `https://www.pcgamer.com/heres-a-neat-trick-for-understanding-how-long-computer-processes-take/`