Memoria Virtual

Arquitectura de Computadores – IIC2343

En los comienzos de la computación, los programadores pasaban mucho tiempo tratando de hacer caber los programas en la memoria, que era poca y cara

La solución era usar memoria secundaria (p.ej., disco):

- *overlays*
- de responsabilidad del programador

1961: automatización del proceso de overlays –> **memoria virtual**

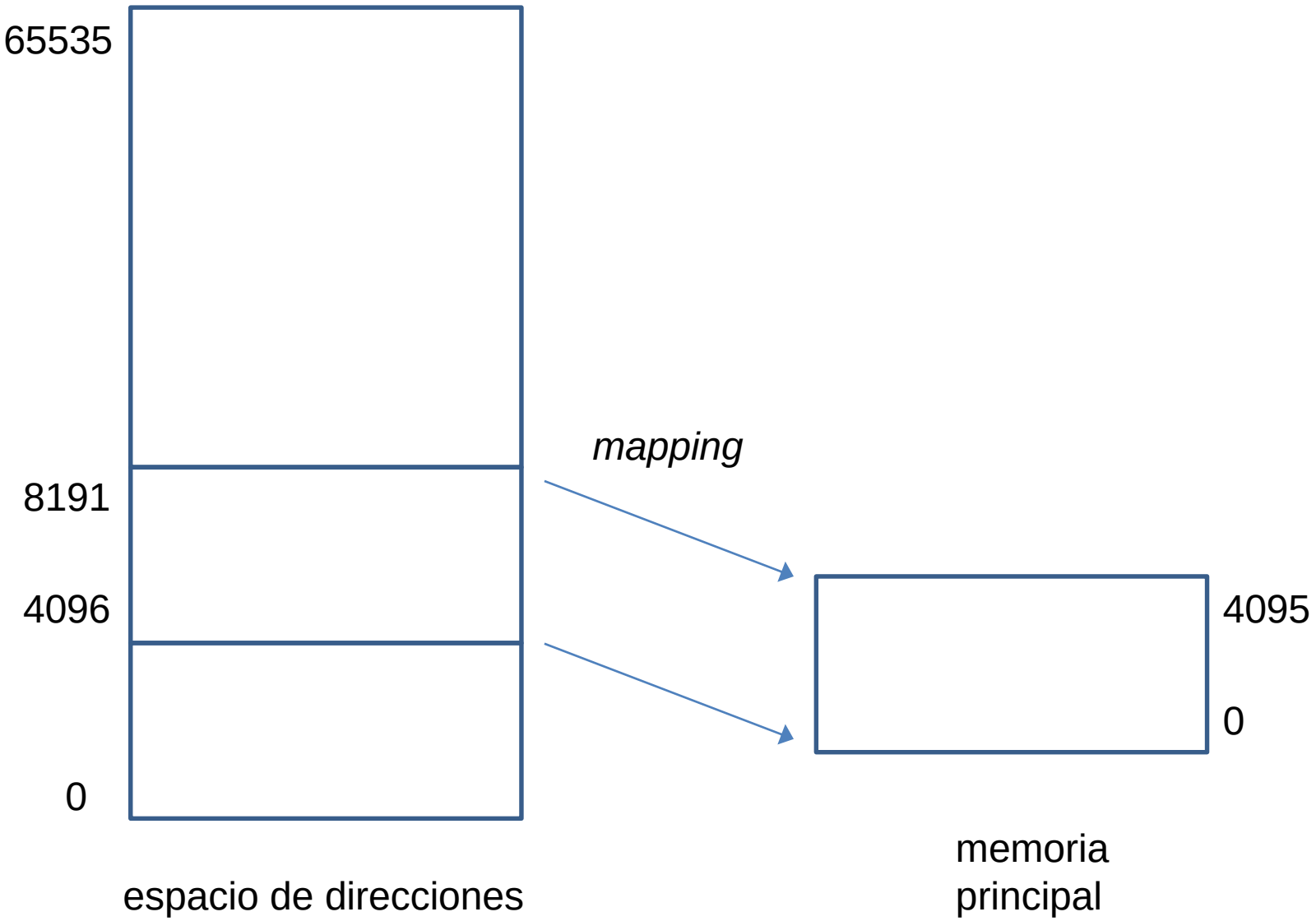
- usada hasta nuestros días

La idea está basada en separar los conceptos de **espacio de direcciones** y **localidades de memoria**

P.ej., supongamos un computador con direcciones de 16 bits y memoria de 4096 palabras:

- 16 bits pueden direccionar 65536 palabras: 0 – 65535 → este es el espacio de direcciones
- podemos decirle al computador que haga corresponder las direcciones 4096 a 8191 referenciadas en el programa con las direcciones 0 a 4095 de la memoria

Definimos una **correspondencia** o ***mapping*** desde el espacio de direcciones a las localidades reales de memoria


65535
8191
4096
0
mapping
4095
0
espacio de direcciones
memoria
principal

¿Qué pasa si un programa salta a una dirección entre 8192 y 12287?

Sin memoria virtual:

- “memoria referenciada inexistente”

Con memoria virtual:

1. El contenido de la memoria principal se guarda en disco
2. Se buscan en el disco las palabras (con direcciones) 8192 a 12287
3. Las palabras 8192 a 12287 son cargadas en la memoria
4. El mapa de direcciones se cambia: las direcciones 8192 a 12287 son *mapeadas* a las localidades de memoria 0 a 4095
5. La ejecución continúa como si nada hubiera pasado

La técnica se llama **paginación**

… y los bloques de programa leidos desde el disco se llaman **páginas**

Hablaremos del *espacio de direcciones virtual*

- las direcciones a las que el programa puede hacer referencia

… del *espacio de direcciones físico*

- las localidades de memoria reales, físicamente disponibles

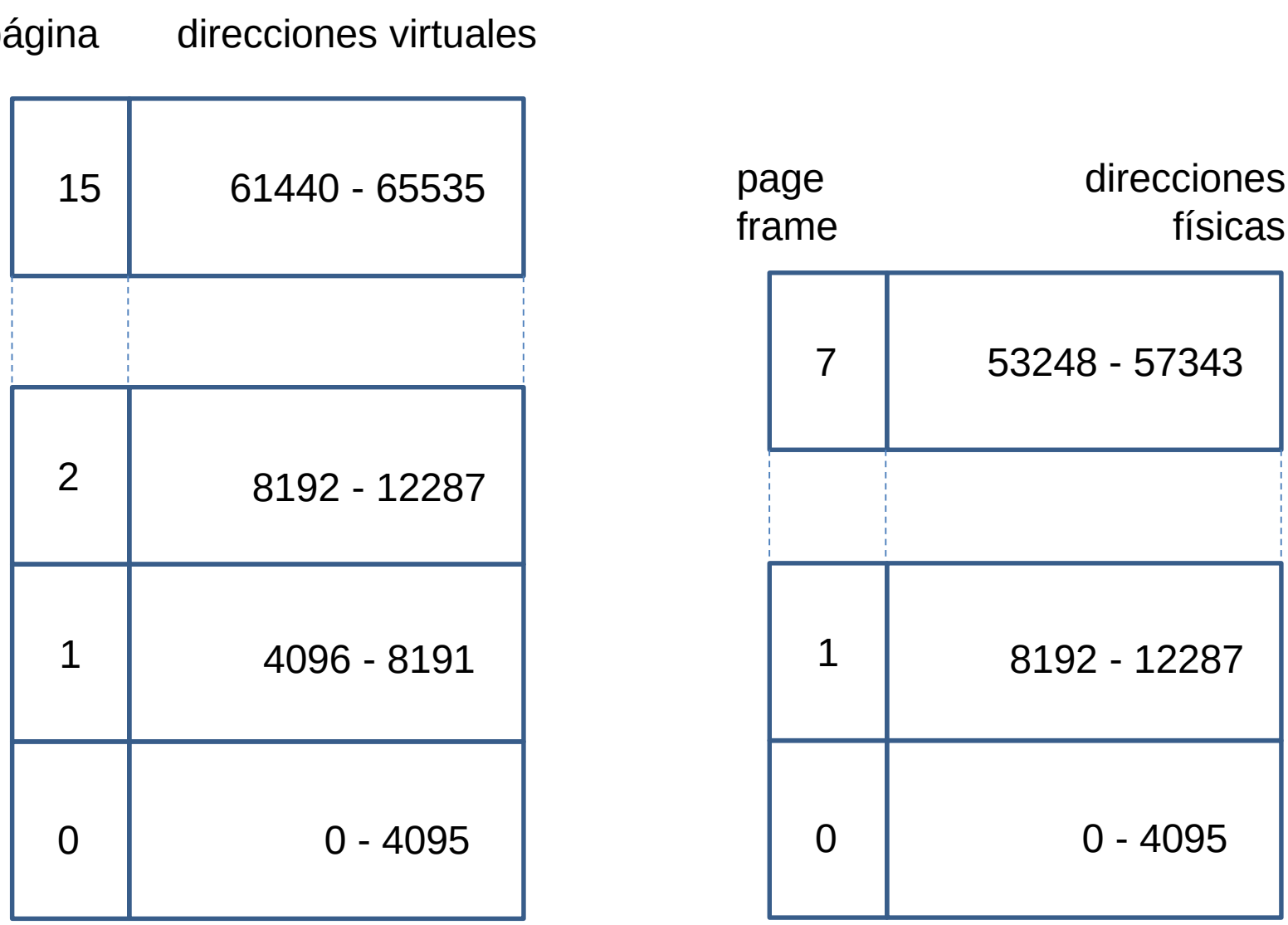
… y del *mapa de memoria* o de la *tabla de páginas*

- que especifica para cada dirección virtual cuál es la dirección física correspondiente

El espacio de direcciones virtual se divide en páginas de un mismo tamaño (una potencia de 2), p.ej., 512 bytes a 64 KB

El espacio físico —la memoria principal— se divide en *page frames* del mismo tamaño que las páginas:

- cada page frame de la memoria principal puede almacenar exactamente una página



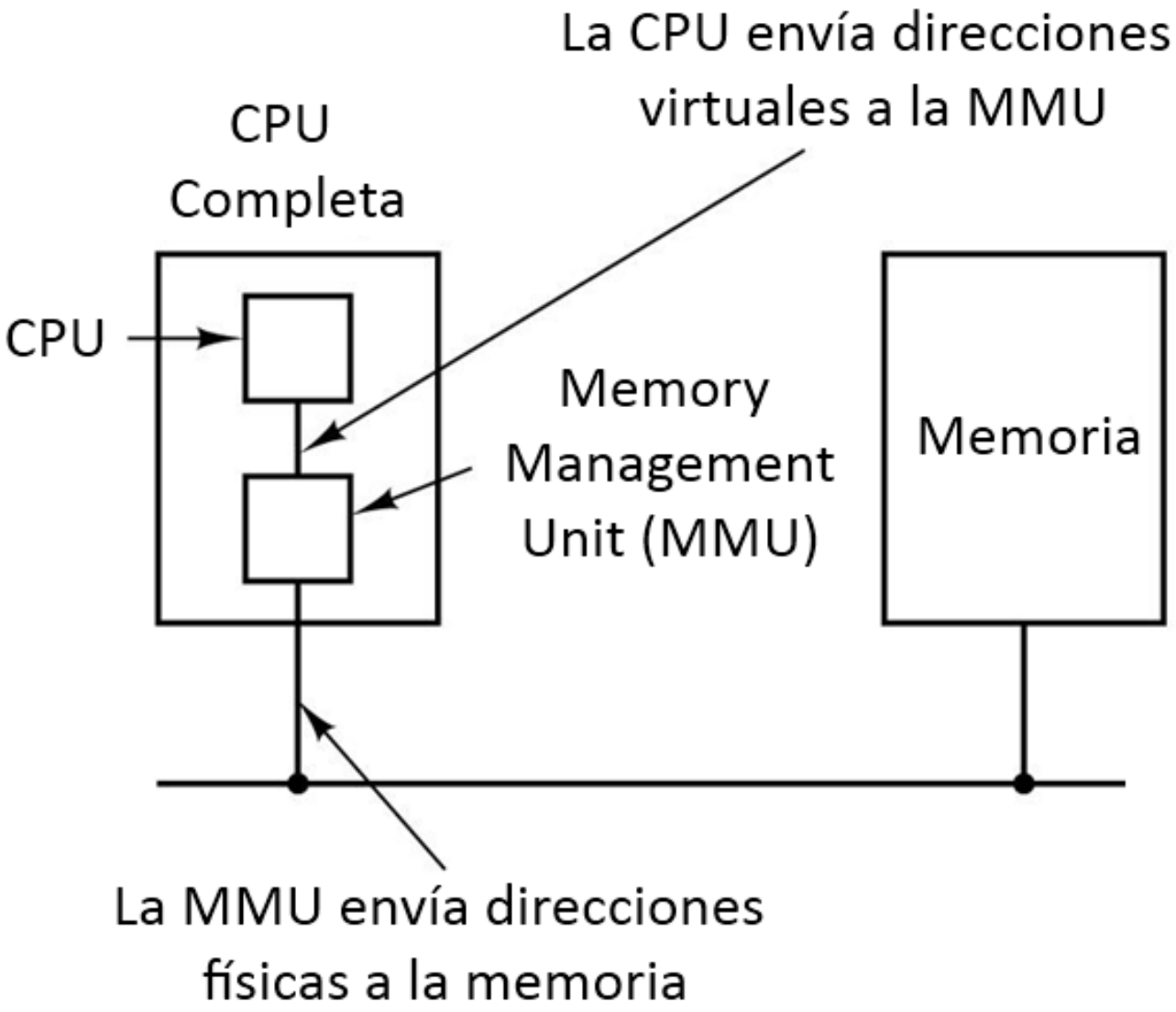
espacio de direcciones divido en páginas de 4 KB

memoria principal dividida en page frames de 4 KB

La memoria virtual es implementada mediante una tabla de páginas, que tiene tantas entradas como páginas hay en el espacio virtual

La *unidad de manejo de memoria* (MMU) convierte las direcciones virtuales en direcciones físicas:

- puede estar en el chip de la CPU o en un chip aparte
- tiene un registro de *input*, p.ej., de 32 bits
- … y un registro de *output*, p.ej., de 15 bits


La CPU envía direcciones
virtuales a la MMU
CPU
Completa
CPU
Memory
Management
Unit (MMU)
Memoria
La MMU envía direcciones
físicas a la memoria

Dirección virtual de 32 bits:

- número de página de 20 bits, como índice a la tabla de páginas
- offset de 12 bits dentro de la página (páginas de 4K)

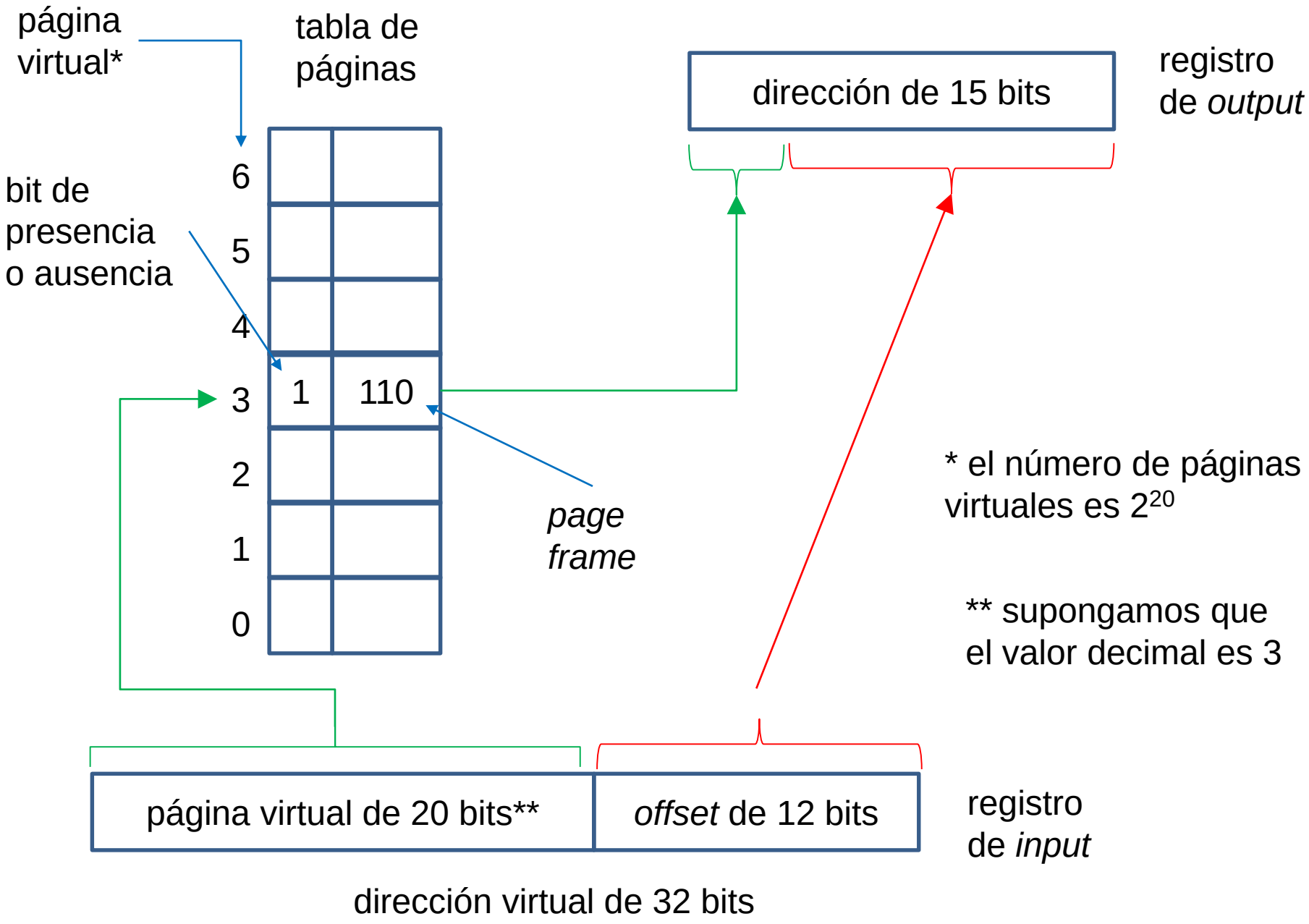
Pero, ¿está la página en memoria?

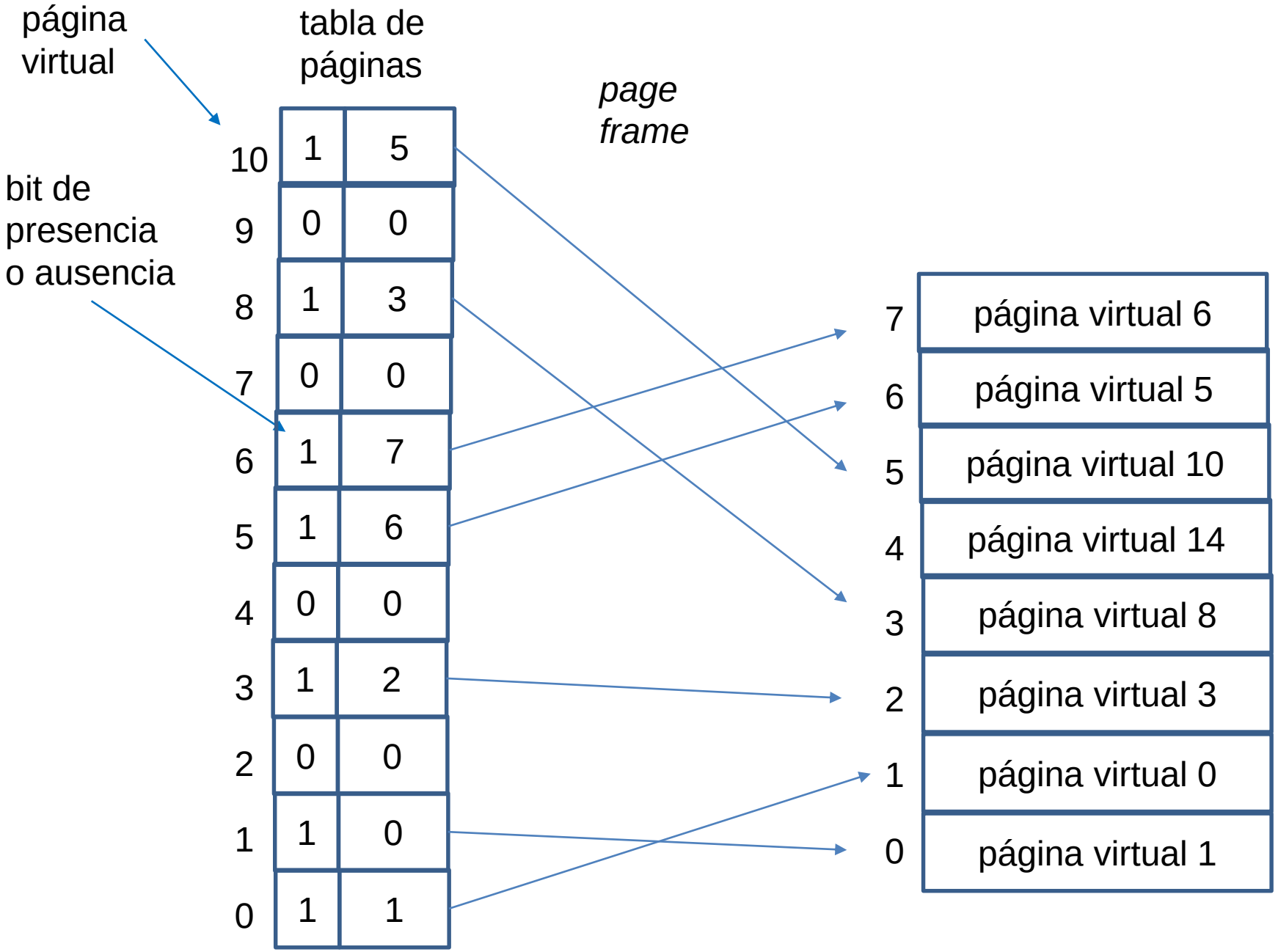
- bit de presencia(1)/ausencia(0) en cada entrada en la tabla

Si está, se usa el valor del page frame almacenado en esa entrada en la tabla:

- este valor (3 bits, si hay 8 page frames) se une a los 12 bits del offset para formar la dirección que se envía a la memoria

Si no está –> *page fault*


página
virtual*
tabla de
páginas
dirección de 15 bits
registro
de *output*
bit de
presencia
o ausencia
6
5
4
3
2
1
0
1
110
*page
frame*
* el número de páginas
virtuales es 2^{20}
** supongamos que
el valor decimal es 3
página virtual de 20 bits**
offset de 12 bits
registro
de *input*
dirección virtual de 32 bits


página virtual
tabla de páginas
page frame
bit de presencia o ausencia
10 1 5
9 0 0
8 1 3
7 0 0
6 1 7
5 1 6
4 0 0
3 1 2
2 0 0
1 1 0
0 1 1
7 página virtual 6
6 página virtual 5
5 página virtual 10
4 página virtual 14
3 página virtual 8
2 página virtual 3
1 página virtual 0
0 página virtual 1

Paginación por demanda:

- se trae una página a la memoria solo cuando la página es solicitada (no por adelantado)
- si el computador es compartido en tiempo (*timeshared*) y los procesos son sacados de la CPU después de ejecutar durante 100ms, cada programa va a ser reanudado muchas veces
- como el mapa de memoria es único para cada programa, la pregunta de si usar esta paginación o no se vuelve crítica

Principio de localidad:

- los programas no referencian su espacio de direcciones uniformemente
- las referencias tienden a ser a un grupo pequeño de páginas

Working set:

- en todo instante de tiempo, existe un conjunto de todas las páginas usadas por las referencias de memoria más recientes
- este conjunto cambia de a poco a lo largo del tiempo
- se puede adivinar cuáles páginas van a a ser requeridas cuando el programa sea reanudado

  … a partir de su *working set* cuando fue suspendido la última vez

Políticas de reemplazo de páginas:

- idealmente, el *working set* se mantiene en memoria para reducir el número de *page faults*
- pero este *working set* debe ser “descubierto” por el sistema operativo a medida que el programa es ejecutado
- si el programa hace referencia a una página que no está en memoria, hay que ir a buscarla al disco

  … y cambiarla por alguna otra página que es enviada de vuelta al disco
- se necesita un algoritmo para decidir cuál página sacar

Elegir una página aleatoriamente no es una buena idea:

- si saliera la página que produjo la *page fault*

  … se va a producir otra *page fault* en cuanto se trate de leer la próxima instrucción

Los sistemas operativos predicen cuál de las páginas en memoria es la menos “útil” :

- predicen cuándo va a ocurrir la próxima referencia a cada página
- … y sacan la página cuya próxima referencia está más adelante en el futuro (una página que no va a ser requerida en mucho tiempo)

P.ej.,

- LRU —la página que se usó por última vez hace más tiempo
- FIFO —la página que se trajo a memoria hace más tiempo

Caché + memoria virtual

(diap. #26 de “Multiprogramación”)

El programa genera una dirección virtual, y se va al TLB

El TLB tiene la entrada de la tabla de páginas, genera la dirección física, y se va al caché:

- el caché tiene la palabra => *best case*
- el caché no tiene la palabra —caché *miss*: el controlador de caché va a buscar la palabra a la memoria —*memory stall*

El TLB no tiene la entrada de la tabla de páginas:

- el control pasa al SO, para que vaya a la tabla de páginas, donde se indica si la página está en memoria o en disco:
- si está en memoria, el SO copia la entrada en la TLB
- si está en disco, hay que manejar una *page faulti => worst case*