Tema 12:
Memoria Virtual

Arquitectura de Computadoras

Ing. Nicolás Majorel Padilla (npadilla@herrera.unt.edu.ar)

http://microprocesadores.unt.edu.ar/arqcom/

Temas que veremos

- Necesidad de Memoria Virtual.
- Idea de Paginación.
- Diseño y Características.
- Vinculación con Caché.
- TLB.

Lectura recomendada

- Computer Organization and Design, RISC-V Edition (2da ed, 2021):
  - Sección 5.7: *Virtual Memory*
  - Sección 5.13: *Real Stuff: The ARM Cortex-A8 and Intel Core i7 Memory Hierarchies*
  - Sección 5.16: *Fallacies and Pitfalls*

Repaso – Concepto de Multiprogramación

- Mediante multiprogramación varios programas se ejecutan concurrentemente.
  - Objetivo: aprovechar mejor los recursos del sistema.
- *¿Quién se encarga de administrar los recursos?*
  - El Sistema Operativo.
- La memoria principal es un recurso más.
  - En un momento dado, varios programas pueden estar en memoria al mismo tiempo.
  - Cada programa debe creer que es el único (transparencia).
  - Los programas no deben interferir entre ellos (protecciones).

Necesidad de Memoria Virtual

- Supongamos que tenemos un programa P1 que requiere un espacio de direcciones de 4 KiB.
- Supongamos también que tenemos un programa P2 que requiere un espacio de direcciones de 4 KiB.
- Supongamos que tenemos 8 KiB de memoria principal (RAM).
- P1 escribe un dato en la dirección 0x2000.
  - Esta dirección de memoria corresponde a la memoria principal, y se denomina **dirección física** o real.
- Luego, por algún motivo, se produce un cambio de contexto y pasa a ejecutarse P2.
- P2 lee un dato de la dirección 0x2000.
  - ¡Error! P2 no lee “su” dato, sino que lee el dato de P1.

Necesidad de Memoria Virtual

- Una posible solución sería borrar toda la memoria al hacer el cambio de contexto.
  - Inviable: sería insoportablemente lento.
  - Además, sólo se estaría usando la mitad de la memoria (no se aprovecharía el recurso).
- Una mejor solución sería que P1 tenga sus datos en la primera mitad de la memoria, y P2 en la segunda mitad (o viceversa).
  - Inviable: el programador debería saber de antemano en qué lugar de memoria se ubicará su programa.
  - Y por lo tanto, podría saber dónde están los demás programas. ¡Alerta de seguridad!

Necesidad de Memoria Virtual

- Los programas deben ser **transparentes** entre sí.
  - El programador de P2 no conoce ni debe conocer nada acerca de P1.
  - Si conoce algo acerca de P1, lo puede hackear.
- Una mejor solución hace que los programadores de P1 y P2 accedan a direcciones **virtuales** (o lógicas).
  - Luego, el SO se encarga de ubicar a P1 y P2 en lugares físicos de memoria…
  - …y de proveer un mecanismo para traducir esas direcciones virtuales a direcciones físicas.

¿Cómo se hace esto? Paginación

- Se divide la memoria principal en partes iguales, de tamaño fijo, a las que se denomina **marcos**.
  - Para nuestro ejemplo anterior, supondremos marcos de 1 KiB.
- Se divide el espacio de direcciones virtuales en partes iguales, del mismo tamaño fijo que los marcos, a las que se denomina **páginas**.
  - Cada página de un programa será ubicada en un marco de memoria principal.
- Para hacer la traducción de direcciones virtuales a direcciones físicas, el SO utiliza una tabla, que se denomina **tabla de páginas**.
  - Esta tabla simplemente indica en qué número de marco fue alojada cada página del programa.
  - Cada programa tendrá su propia tabla de páginas.

Continuación del ejemplo

- Supongamos que P1 se ubica en los primeros 4 KiB de la memoria principal, y P2 en los segundos 4 KiB.
  - *¿Cómo serían sus tablas de páginas?*

# Página (virtual)	# Marco (físico)
0	0
1	1
2	2
3	3

# Página (virtual)	# Marco (físico)
0	4
1	5
2	6
3	7

- Supongamos que P1 quiere acceder a su dirección 500 (virtual).
  - Corresponde a la página 0, que está ubicada en el marco 0 de memoria principal.
  - Por tanto su dirección física también será 500.
- Supongamos que P2 quiere acceder a su dirección 500 (virtual).
  - Corresponde a la página 0, que está ubicada en el marco 4 de memoria principal.
  - Por tanto su dirección física será 4096 + 500 = 4596.

Continuación del ejemplo

- ¿Podría P1 acceder a datos de P2?
  - Sólo si tuviese acceso a la tabla de páginas de P2.
  - Se hace que las tablas de páginas pertenezcan al SO, y se prohíbe cualquier acceso de un programa a una tabla de páginas que no sea la propia.
- ¿Pueden las páginas de P1 y de P2 estar ubicadas en marcos que no sean contiguos?
  - Por supuesto, no hay ningún tipo de restricción con respecto a la ubicación.
- ¿Qué sucede si P1 necesita 5 KB de memoria?
  - Localidad: se mantienen las páginas más usadas. El resto, en el nivel siguiente de la jerarquía.
  - Las tablas de página deben indicar si una página está ubicada o no en un marco. Bit de validez.

Continuación del ejemplo

- ¿Pueden dos páginas virtuales, una de P1 y una de P2, estar mapeadas al mismo marco?
  - Sí, puede darse el caso. Usado por programas para compartir datos o código.
- ¿Qué sucede si P1 quiere acceder a una página y no está ubicada en ningún marco?
  - Se produce una excepción por **fallo de página**.
  - El SO se encarga de traer la página y actualizar las tablas.

Memoria Virtual – Síntesis

- Cada programa posee un espacio de direcciones propio y contiguo e infinito.
  - Pero este espacio de direcciones es virtual. Puede ser mayor que la memoria física instalada.
  - La suma del espacio requerido por varios programas puede ser mayor que la memoria física instalada.
- El SO le hace creer que es contiguo.
  - En realidad, cualquier página puede estar ubicada en cualquier marco.
- El SO realiza la traducción de dirección virtual a física mediante tablas.
- El SO además garantiza:
  - **Transparencia**, al crear la ilusión de que es contiguo.
  - **Seguridad**, porque cada programa tiene acceso a su propia tabla.
  - **Eficiencia**, porque mantiene en memoria las páginas más utilizadas.

Memoria Virtual

- Es manejada en forma conjunta por el SO y por el procesador.
  - Hardware provee soporte a las funciones necesarias del SO.
  - Normalmente, a través de una Unidad de Manejo de Memoria (MMU).
- Se encarga de administrar dos niveles de la jerarquía de memoria: Memoria Principal (primario) y Disco (secundario).
  - Usa a la Memoria Principal como un caché del disco.
- Analogías con caché:
  - Bloque de memoria → Páginas
  - Bloque de caché → Marco
  - Fallo de caché → Fallo de página

Aspectos de diseño de MV

- Tamaño de las páginas.
  - En sus comienzos, en 1962, las páginas fueron de 4 KiB.
  - 50 años después, ¡siguen siendo de 4 KiB!
- Proceso de traducción de direcciones virtuales a direcciones físicas.
- Política de reemplazo de páginas.
- Política de escritura.

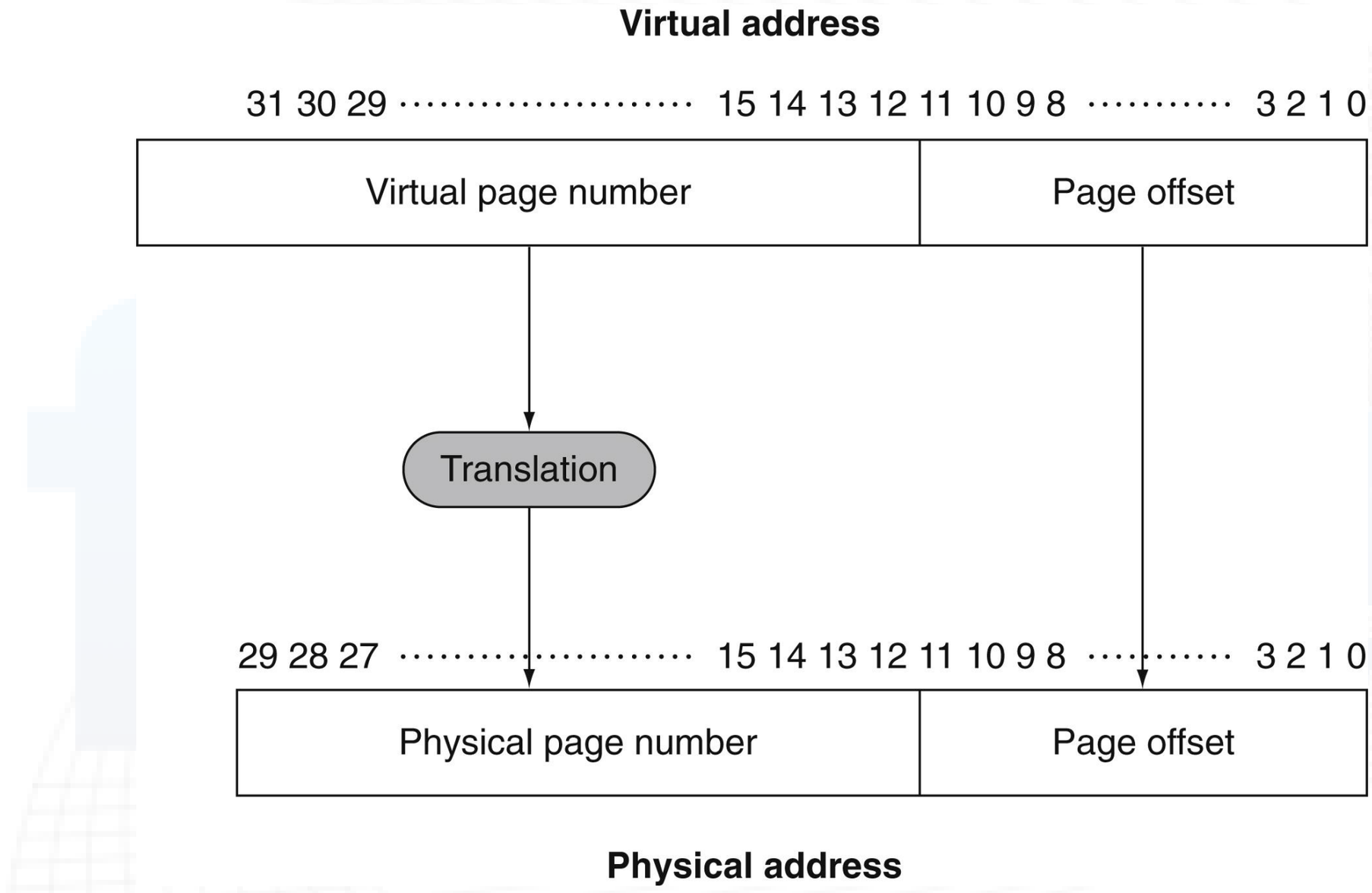
Mapeo de direcciones

- Se parte de un espacio de direcciones virtuales de 2^N bytes.
  - N es la cantidad de bits de las direcciones virtuales.
  - RISC-V soporta 3 distintos tamaños: 32, 39 y 48, que dan lugar a tres esquemas de paginación diferentes.
- Se busca traducir a un espacio de direcciones físicas de 2^M bytes.
  - M es la cantidad de bits de las direcciones físicas.
  - Es un máximo fijado al diseñar el sistema de memoria virtual de los procesadores (y los SO).
  - Realmente, está fijado por la cantidad de memoria RAM instalada.
- Normalmente N > M.
  - Así es como se genera la ilusión de una memoria infinita.
  - *¿Pueden ser iguales?*
  - *¿Puede ser N < M?*

Mapeo de direcciones

- La dirección virtual que entrega el procesador se descompone en dos campos:
  - Número de página virtual (VPN, *Virtual Page Number*).
  - Offset de página.
- La dirección física también se descompone en dos campos:
  - Número de marco (PPN, *Physical Page Number*).
  - Offset de página.
- La cantidad de bits de los offset es la misma, porque páginas y marcos son de igual tamaño.
- El mapeo consiste en traducir un VPN a un PPN.
  - **Los offset no se traducen**.

Mapeo de direcciones



- *¿Cuál es el tamaño del espacio de direcciones virtual?*
- *¿Cuál es el tamaño de la memoria física?*

Tabla de páginas

- Mencionamos anteriormente que la traducción se realiza mediante tablas.
- El índice con el cual se accede a la tabla es el número de página virtual (VPN).
- Con ello, se obtiene una **entrada de la tabla de páginas** (PTE, *Page Table Entry*), que contiene básicamente tres datos:
  - Un bit de Validez, que indica si esa página está o no en memoria.
  - Varios bits de estado, que indican los permisos de acceso de la página, si la página fue modificada, ayudas para algoritmos de reemplazo, etc.
  - El número de marco (PPN) donde está alojada la página.

Tabla de páginas

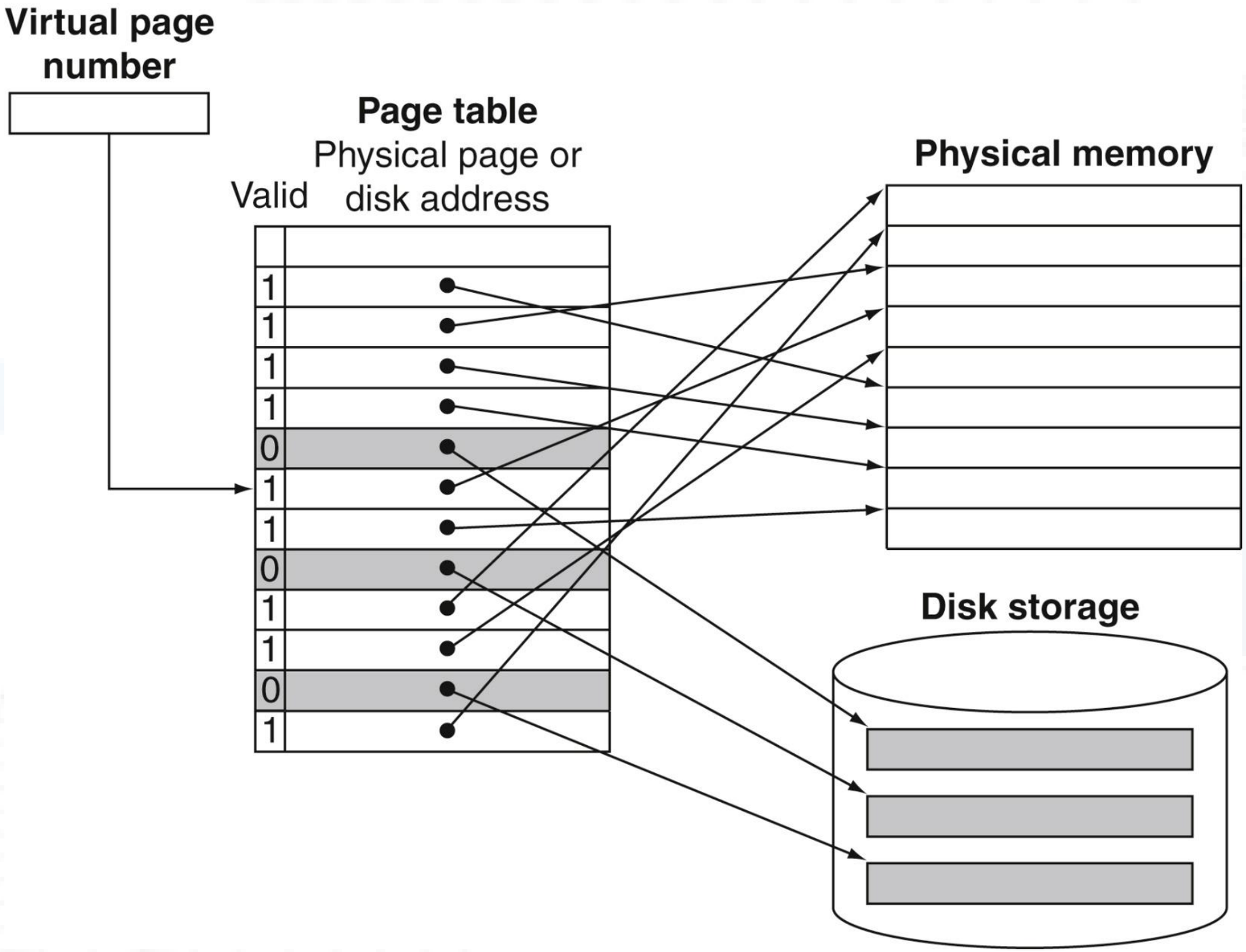
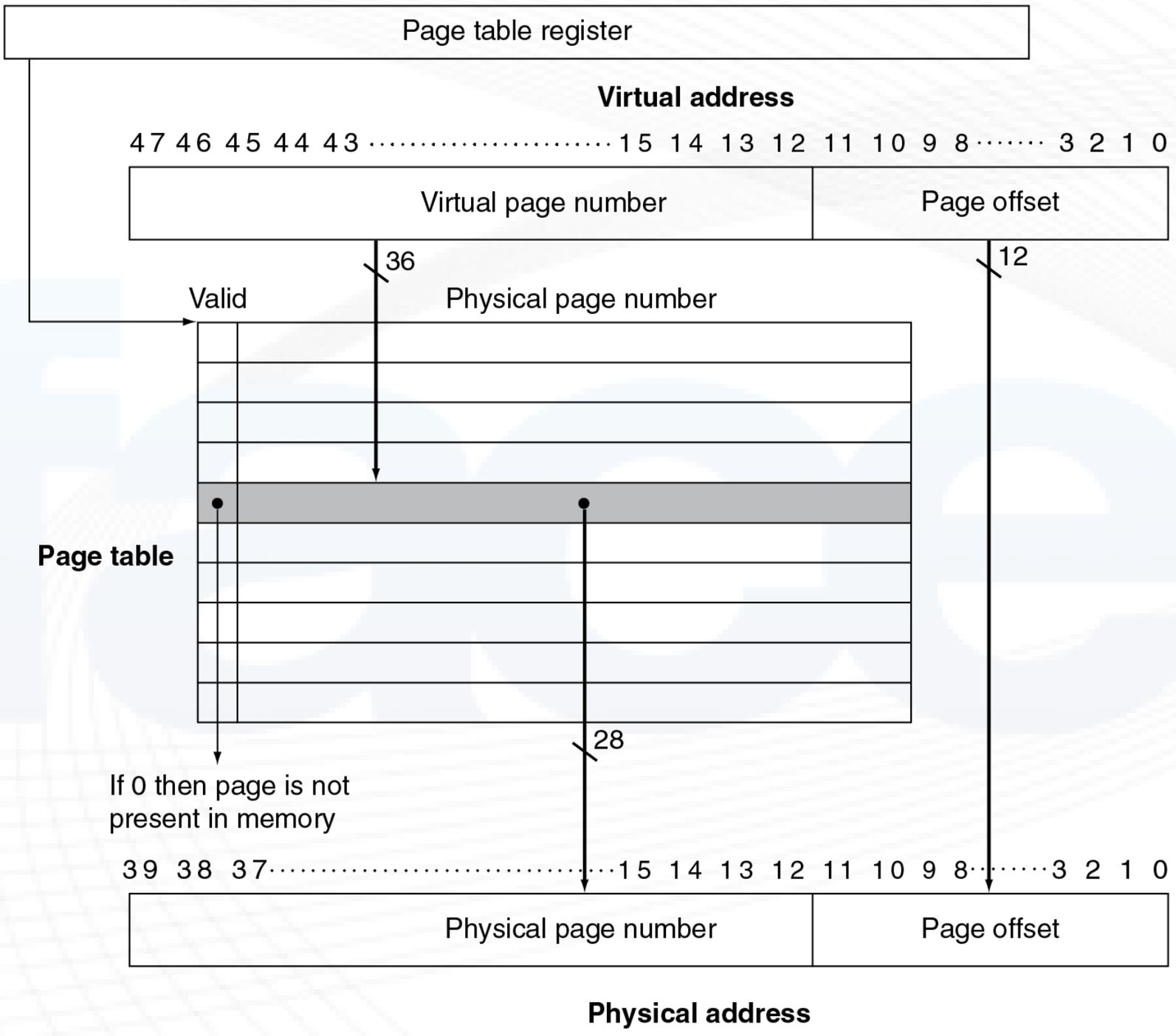


Tabla de páginas

- La tabla de páginas obviamente debe estar en memoria principal.
- La dirección base de esta tabla viene dada por un registro especial del procesador (PTBR, *Page Table Base Register*).
  - En RISC-V, se denomina SPTBR.
- La tabla de páginas, al igual que el PC y los registros, forma parte del estado de un programa, y debe ser preservada.
  - Solamente guardando el registro base de la tabla.
- Las entradas de la tabla de páginas pueden ser de cualquier cantidad de bits, pero normalmente se usan tamaños que sean potencias de 2.
  - En RISC-V, son de 32 o 64 bits.

Mapeo de direcciones con tabla de página



Algoritmo de traducción

- Dada una dirección virtual, se obtiene su número de página virtual (VPN).
- Con el VPN, se accede a la tabla de páginas y se obtiene una entrada (PTE).
  - Si el bit de validez no está activo, indica que la página solicitada no está en memoria, y ocurre un **fallo de página**.
    - La página deberá ser traída desde disco.
  - Si el bit de validez está activo:
    - Si no coinciden los permisos, ocurre un **fallo de protección**.
    - Si se cumplen los permisos, se devuelve el número de marco en el que está ubicada la página (PPN).

Fallos de página

- Demoran **millones de ciclos**.
- El SO se encarga de gestionarlos.
  - Hacen un cambio de contexto, para que se ejecute otro programa mientras se espera.
  - Además, porque el disco es parte del sistema de Entrada/Salida.
- La memoria virtual usa una organización asociativa para minimizar los fallos de página.
  - Cualquier página puede ir a cualquier marco.
- Se usa LRU como política de reemplazo.
- Se usa Write-Back como política de escritura.

Tamaño de la tabla de páginas

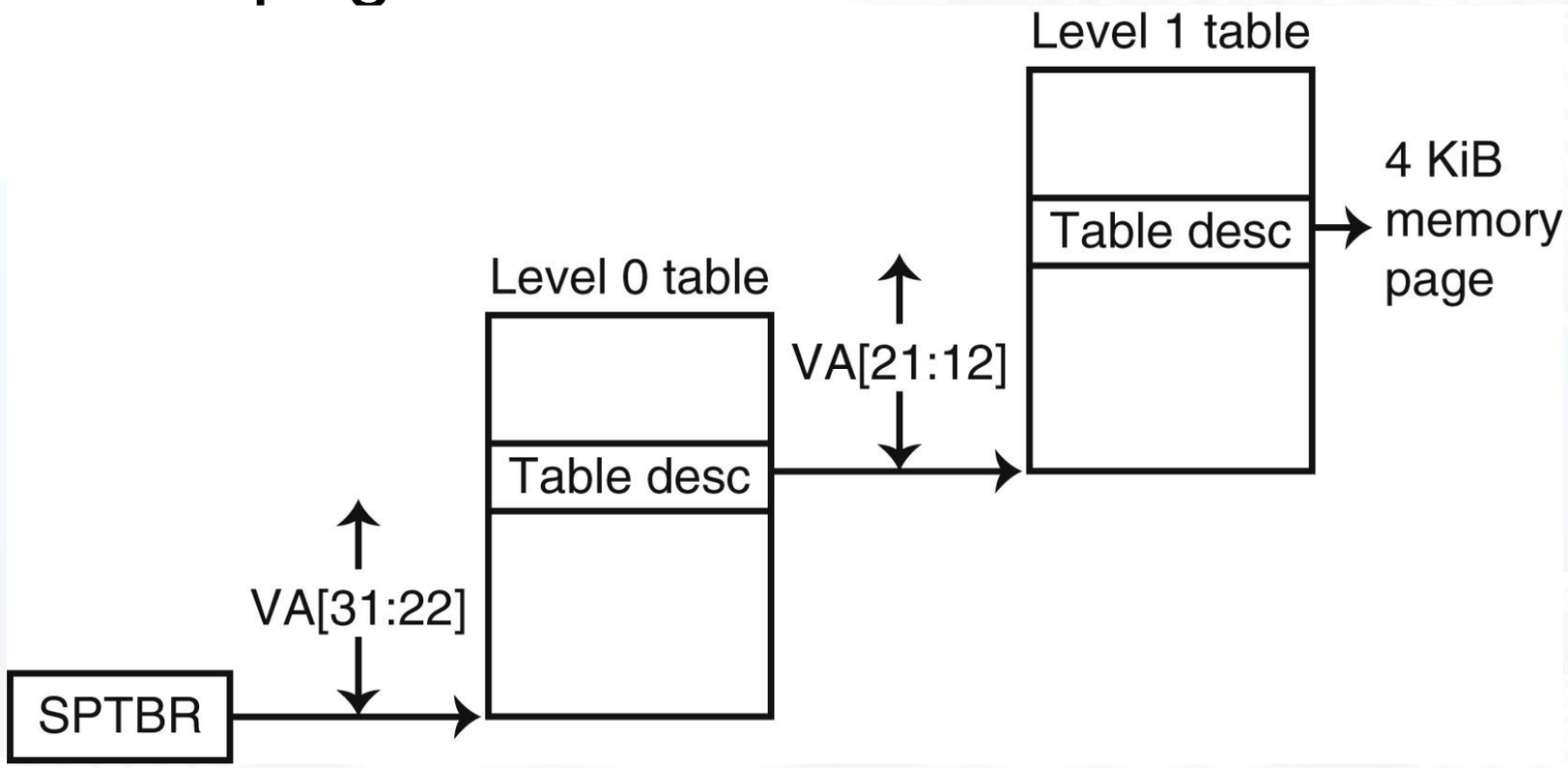
- Suponiendo direcciones virtuales de 32 bits y páginas de 4 KiB, la tabla de páginas tiene 2^{20} entradas.
- Si cada entrada es de 32 bits, la tabla ocupa 4 MiB.
- Recuerden que hay una tabla de páginas por programa.
- *¿Y si las direcciones virtuales fuesen de 48 bits, con entradas de la tabla de páginas de 64 bits?*
- El tamaño de la tabla de páginas representa un problema mayúsculo.
- Pero peor es que el mecanismo de traducción no es escalable.

Múltiples niveles de traducción

- La solución consiste en dividir la tabla de páginas en partes iguales de tamaño fijo, y usar solamente las partes necesarias.
  - O sea…¡nuevamente paginación!
- En este caso, se dice que la traducción posee múltiples niveles de tablas de página.
  - El registro base apunta a una tabla de páginas de primer nivel.
  - La entrada de esta tabla de página contiene el marco donde se encuentra la tabla de páginas de segundo nivel.
- El número de página virtual se divide en tantos campos como niveles haya.

Traducción con dos niveles de TP

- RISC-V con direcciones virtuales de 32 bits y entradas de tabla de página de 32 bits.



- *¿Tamaño de las tablas de página en cada nivel?*
- RISC-V soporta además esquemas de paginación de 3 y de 4 niveles de TP.

Otro problema con la traducción

- Anteriormente dijimos que la tabla de páginas reside en memoria.
- Entonces cada acceso a memoria en realidad genera dos accesos:
  - Uno a la tabla de páginas, para hacer la traducción.
  - Otro al dato que se desea acceder.
- Sin embargo, la memoria es lenta.
- Si se tiene un esquema de múltiples niveles de traducción, el problema se agrava aún más.
- Idea de solución: cuando se accede a un dato de una página, se suele acceder al mismo nuevamente, o acceder a los vecinos. Todos con la misma traducción.

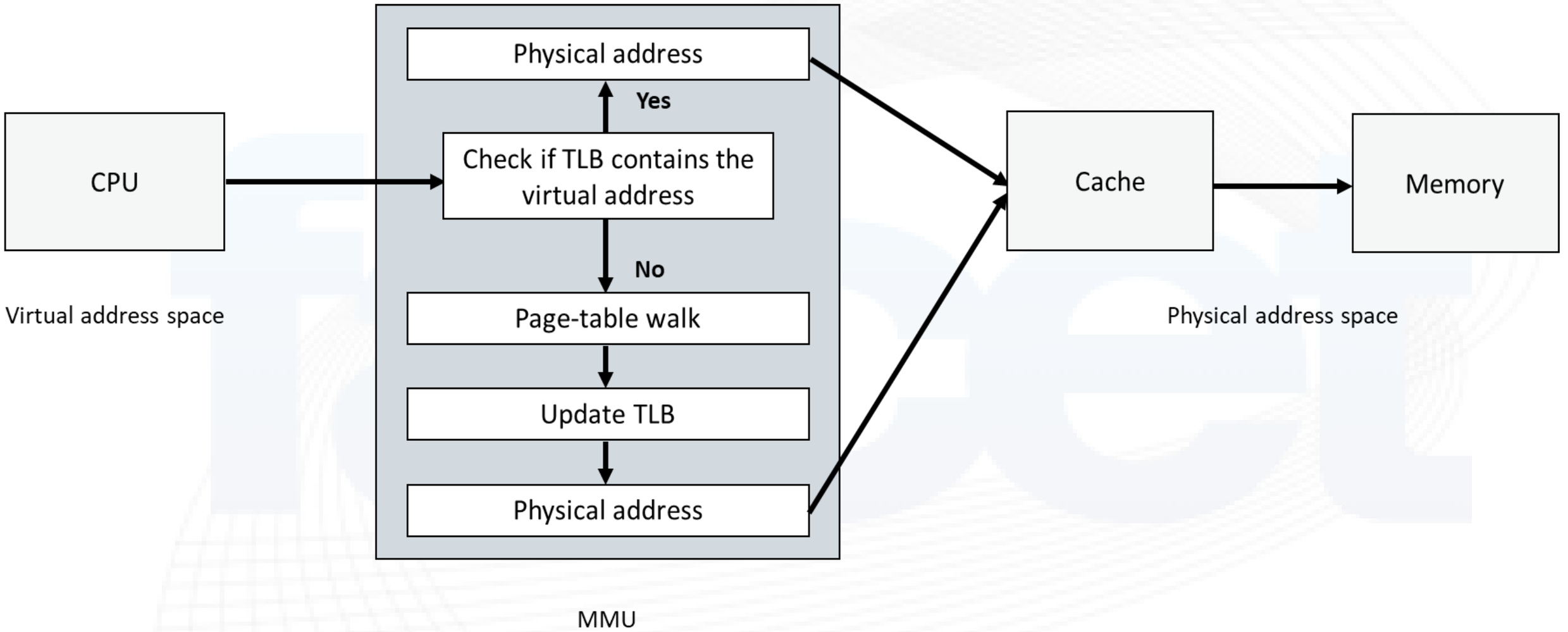
TLB

- Se aprovecha la localidad en la referencia a las páginas agregando un caché.
- Este **caché de traducciones** se denomina históricamente *Translation Lookaside Buffer* (TLB).
  - Objetivo: evitar el acceso a la/s tabla/s de páginas.
  - Almacena las últimas traducciones realizadas.
  - Mantiene una copia de la entrada de la tabla de páginas, incluyendo sus bits de estado.
  - El número de página virtual es almacenado como etiqueta, y es comparado con el proveniente de la dirección virtual.

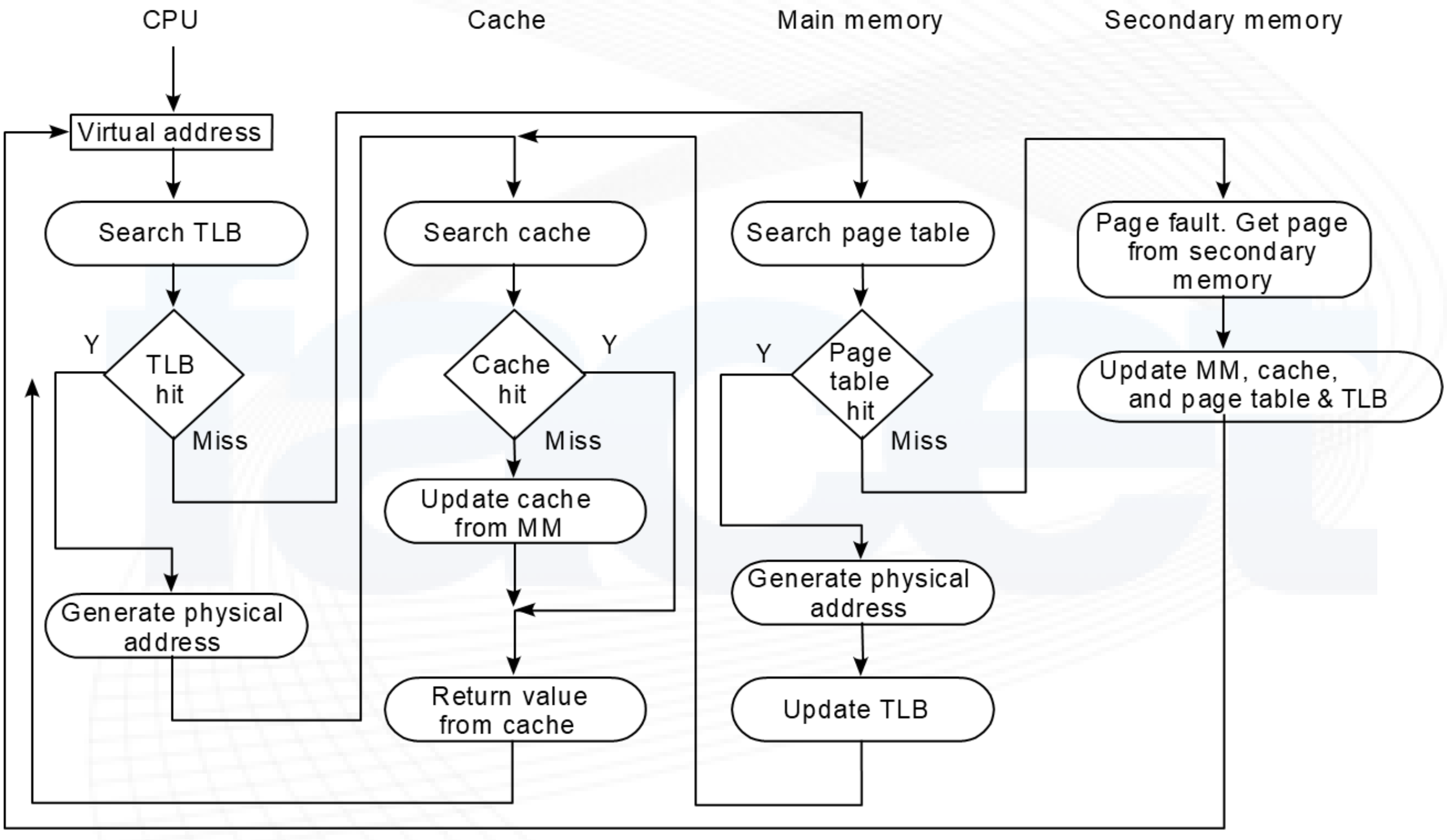
Características típicas de los TLB

- Suelen almacenar entre 16 y 1024 entradas.
  - Usualmente, asociativos puros o por conjuntos.
- Tiempos de acceso similar a caché.
  - Usualmente 0,5 o 1 ciclo.
- Tasa de fallos muy baja.
  - Usualmente entre 0,01% y 1%.
- Penalidad por fallos entre 10 y 100 ciclos.

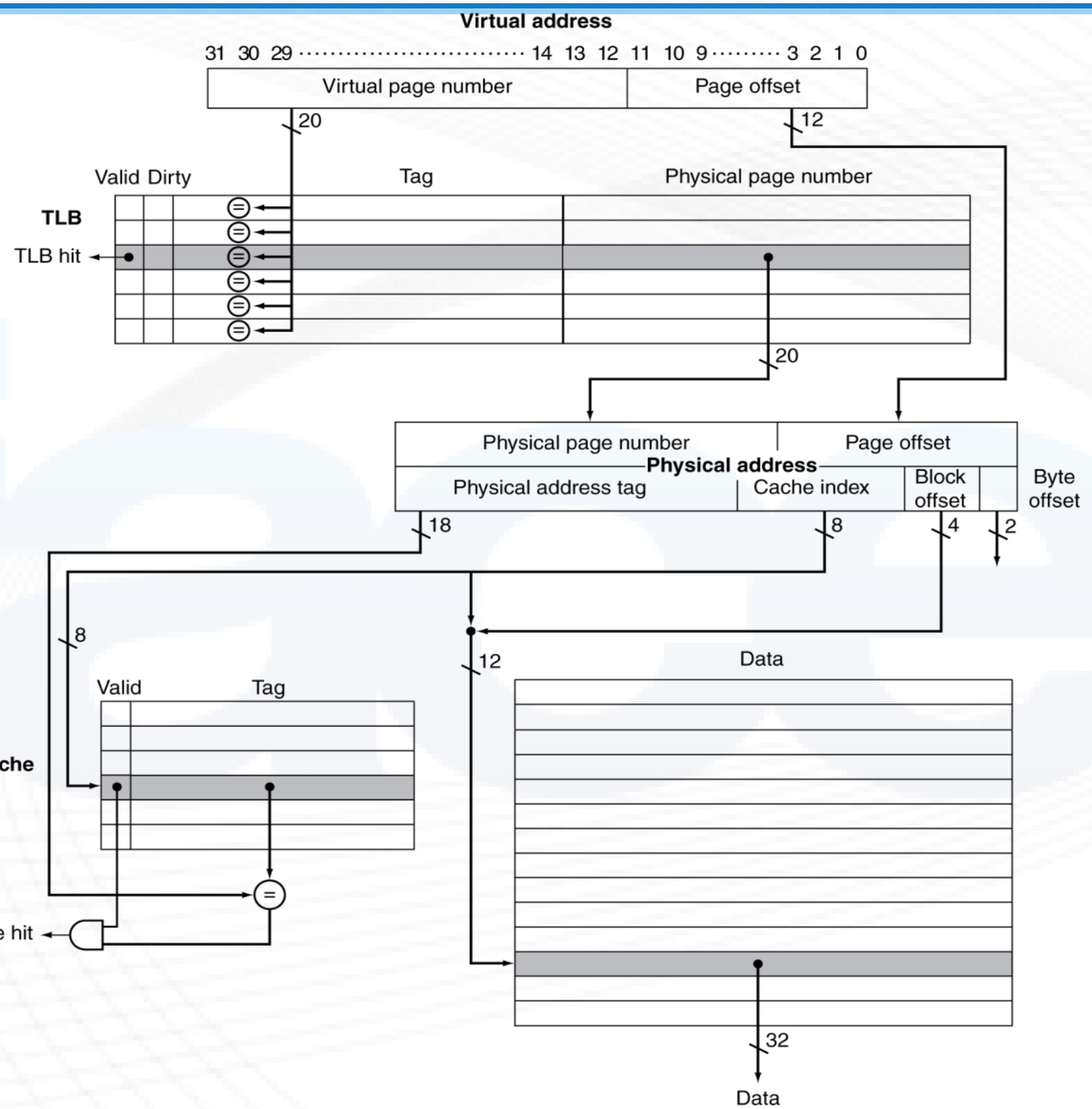
Funcionamiento de TLB con caché



Funcionamiento de TLB con caché



Funcionamiento de TLB con caché



¿TLB en paralelo con caché?

- En el ejemplo anterior, el acceso al TLB es previo al acceso al caché, para acceder al caché con direcciones físicas.
- Si se accede al caché con direcciones virtuales, puede ocurrir un **problema de aliases**.
  - Dijimos que varias direcciones virtuales pueden mapearse a una misma dirección física, para compartir datos.
  - Esto podría hacer que varios bloques de caché estén cargados con el mismo dato. Ineficiente.
  - Si se modifica un dato, deberían modificarse el resto de las entradas. Muy difícil y costoso.
- Pero acceder al caché con direcciones físicas aumenta el tiempo de acierto del caché (camino crítico).

TLB en paralelo con caché

- El TLB es accedido con los bits más significativos de la dirección virtual.
  - Los menos significativos son el offset de página y no se traducen.
- El caché es accedido con los bits menos significativos de la dirección física.
- Se puede lograr que el caché sea accedido con los bits del offset de página, de manera de no esperar la traducción.
  - Se debe cumplir que la cantidad de bits necesarios para índice + offset del caché sea menor o igual que la cantidad de bits del offset de página virtual.
- Para evitar el problema de alias, la etiqueta que se almacena en el caché es de la dirección física, no virtual.
- Esto se denomina un caché indexado virtualmente, con etiquetas físicas (VIPT, *Virtually Indexed Physically Tagged*).

TLB en paralelo con caché

- Poner en paralelo el TLB con el caché suele limitar el tamaño del caché.
  - Con páginas de 4 KiB (12 bits de offset), solamente se puede poner un caché de mapeo directo de 4 KiB.
    - *¿Cómo podríamos hacer para poner un caché de 8 KiB?*
- Otra alternativa es no ponerlo en paralelo, sino agregar una nueva etapa al pipeline.
  - Otra vez riesgos…
  - *¿Recuerdan cuánto eran los tiempos de acierto de los cachés L1 de los procesadores actuales?*

Protecciones de Memoria

- Hoy es quizás la principal función de la memoria virtual.
- Asegurar que, aunque múltiples programas compartan la memoria, ninguno escriba en el espacio de direcciones de otro.
  - Ya sea intencionalmente o no.
  - Tampoco en el espacio de direcciones del SO.
- ¿Cómo se comparten datos entre programas?
  - P1 pide a SO acceder a datos de P2.
  - Si P2 permite, SO copia la entrada de la TP de P2 a la TP de P1 (incluyendo protecciones).

Protecciones de Memoria

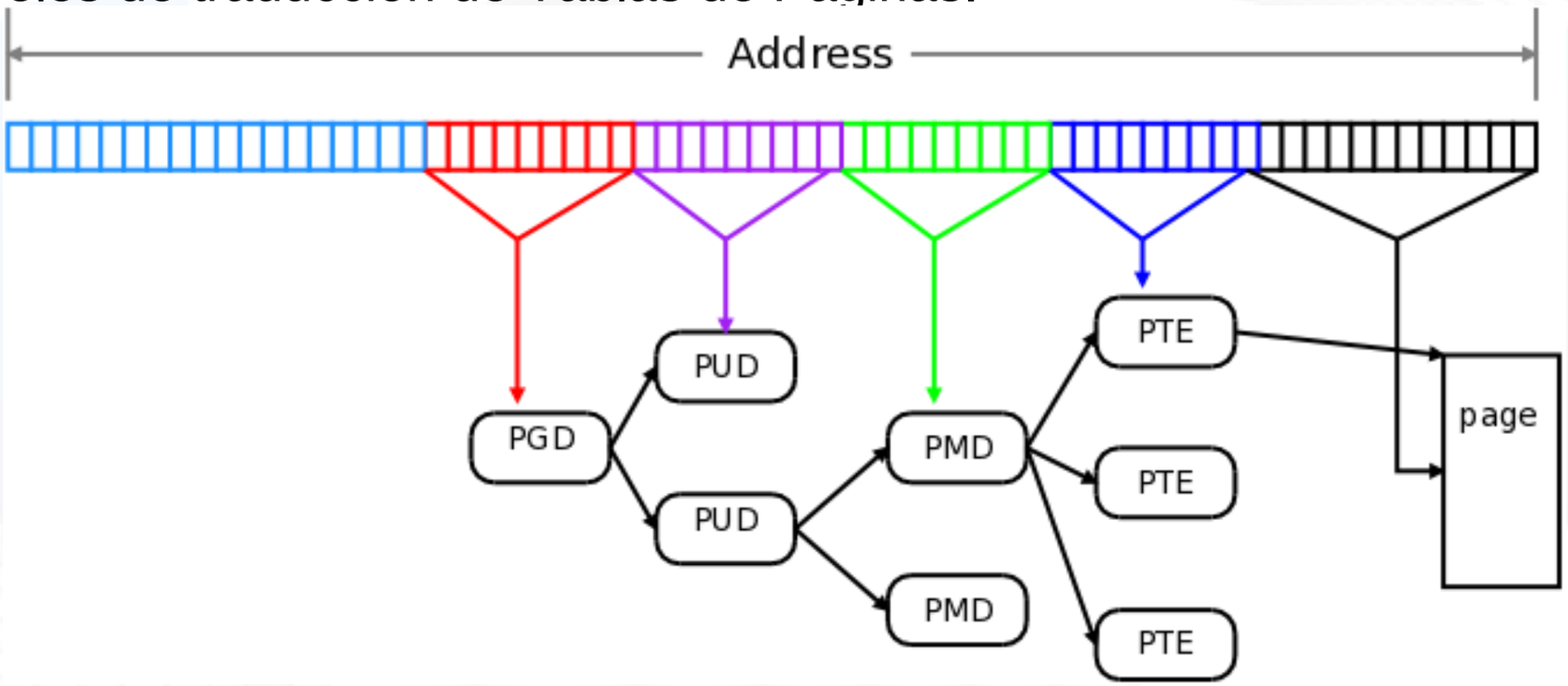
- Para poder implementar protecciones en un sistema con memoria virtual, el hardware debe proveer al SO con lo siguiente:
  - Proveer mínimamente **dos modos de ejecución** (usuario y supervisor).
  - Proveer datos de estado del procesador que un proceso de usuario puede leer pero no escribir.
    - El bit de modo de ejecución, el puntero base de la tabla de páginas y el TLB.
    - Son escritos por el SO mediante **instrucciones que sólo se ejecutan en modo privilegiado**.
  - Proveer **mecanismos para pasar de modo usuario a modo supervisor, y viceversa**.
    - Típicamente a través de una llamada al sistema (instrucciones **ecall** y **sret** en RISC-V).
- Recordemos que las tablas de páginas de los programas se guardan en el espacio de direccionamiento del SO.

Cambios de contexto

- Cuando el SO decide empezar a ejecutar un programa P1 en vez de un programa P2.
  - Se deben borrar en el TLB todas las entradas pertenecientes a P2 (*TLB flush*).
  - Puede consumir mucho tiempo.
- Se suele extender el espacio de direcciones virtual agregando un **identificador de proceso**.
  - Se mantiene un registro adicional (*ASID*).
  - Se concatena este valor con la etiqueta del TLB.
    - Hay un acierto sólo cuando coinciden la etiqueta Y el identificador de proceso.
- Una solución similar se utiliza con los cachés VIPT.

Memoria Virtual en x86-64

- Empezó en 1985 con el procesador 80386.
- La arquitectura Core (2005) soporta direcciones virtuales de 48 bits y direcciones físicas de hasta 46 bits.
- Páginas de 4 KiB.
- 4 niveles de traducción de Tablas de Páginas.



- Desde 2017, direcciones virtuales de 57 bits y direcciones físicas de 52 bits, con 5 niveles de traducción.

Memoria Virtual en perspectiva

- Desde que se inventó el sistema de memoria virtual, la capacidad de la memoria se incrementó casi 100.000 veces.
- Los sistemas actuales presentan una muy baja tasa de fallos.
- **¿Por qué se sigue usando memoria virtual?**
  - Protecciones.
  - Flexibilidad (relocalización) y transparencia.
  - Eficiencia (localidad).

- *“Solamente hay un error que se puede cometer en diseño de computadoras que es difícil reponerse: no tener suficientes bits para el manejo y direccionamiento de la memoria.”* Gordon Bell, 1976.
  - Cada 12 años (aprox.) se incrementa el espacio de direcciones virtuales.

Resumen final

- Memoria Virtual es un nivel más de la Jerarquía de Memorias.
  - Inicialmente útil para compartir memoria cuando era limitada.
  - Hoy principalmente útil por su esquema de protecciones.
- La forma más común de implementación de Memoria Virtual es a través de paginación.
- La traducción de direcciones virtuales a direcciones físicas se realiza mediante tablas de páginas.
  - Múltiples niveles de traducción para evitar problemas de tamaño.
  - TLB para acelerar el proceso de traducción.

Resumen final

- Cachés, Memoria Virtual y TLB se analizan en base a las mismas cuatro preguntas:
  - ¿Dónde se ubica un bloque?
  - ¿Cómo se identifica un bloque?
  - ¿Cómo se reemplaza un bloque?
  - ¿Cómo se manejan las escrituras?
- La performance de un procesador no depende sólo de su ISA y de su implementación, sino que está fuertemente influenciada por su jerarquía de memoria.
  - Un procesador moderno pasa la mitad del tiempo esperando que se resuelvan fallos de memoria.
- Toda la jerarquía de memoria de un procesador moderno ocupa aproximadamente la mitad del área total y consume casi la mitad de la energía.

Agradecimientos

- Las diapositivas de este tema fueron basadas en las realizadas por el Ing. Daniel Cohen.
- A su vez inspiradas en las clases del curso CS152 de la Universidad de Berkeley, California, USA.
  - Realizadas por los Prof. D. A. Patterson, John Lazzaro, Krste Asanovic.