udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA

Memoria Virtual

Adín Ramírez
adin.ramirez@mail.udp.cl

Sistemas Operativos (CIT2003-1)
1er. Semestre 2015

Adaptado de Dave Eckhardt

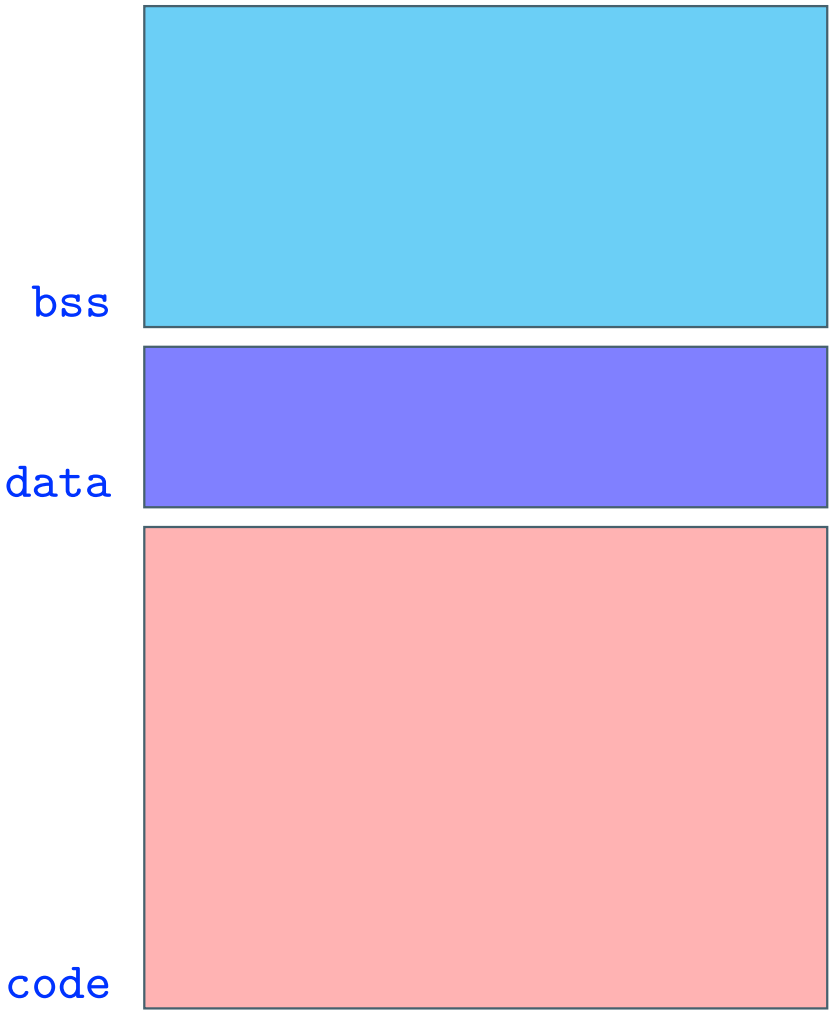
Objetivos de la clase

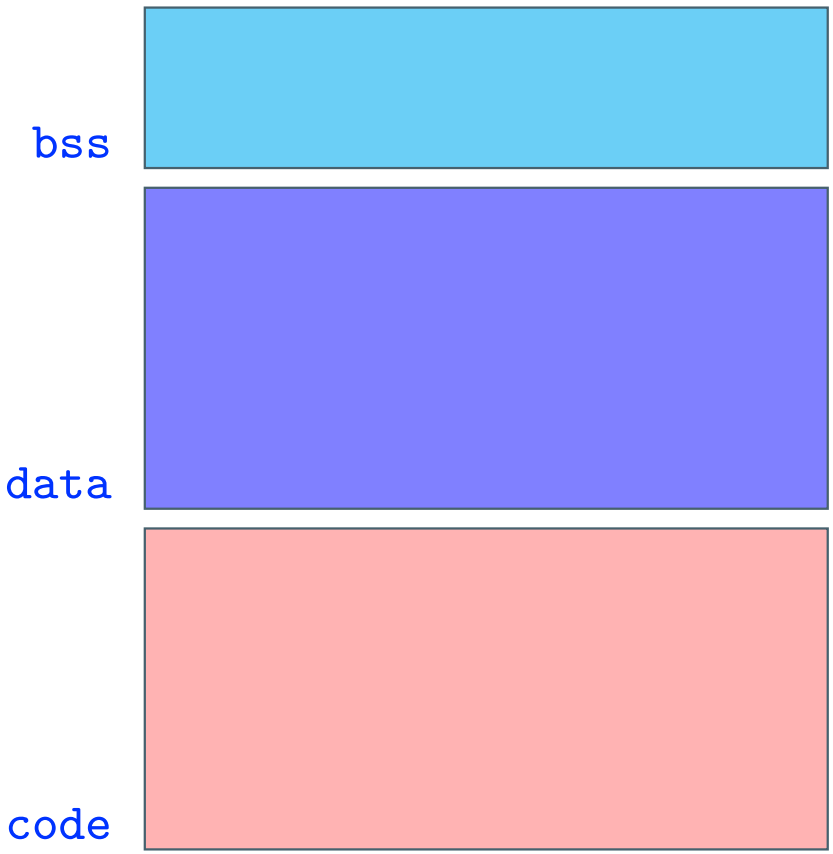
- ★ Revisen las lecturas
- Memoria lógica vs. física
- Mapeo de memoria contigua
- Fragmentación
- Paginación
  - Teoría de tipos
  - Mapa disperso

Memoria lógica vs. física

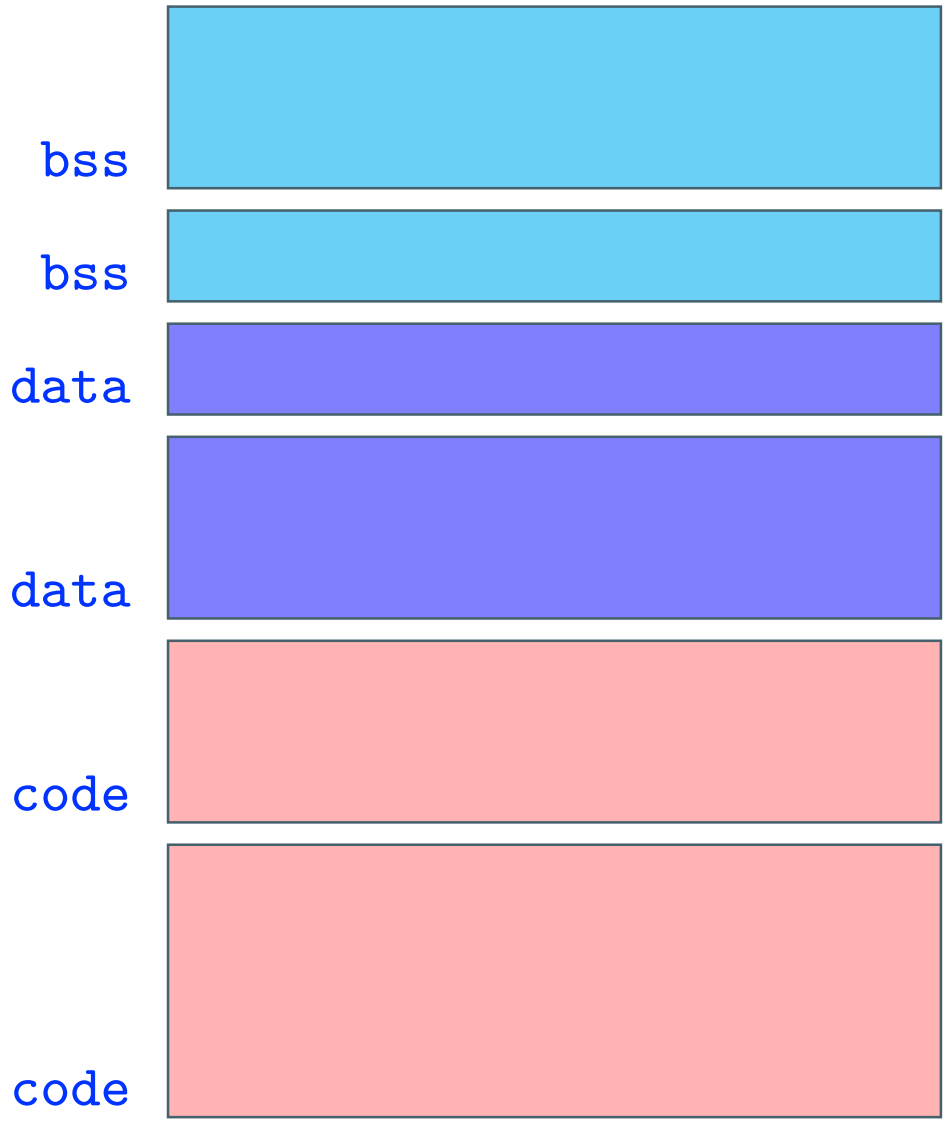
- Todo es acerca de espacios de memoria
  - Un problema complejo
    - E.g., IPv4 $\Rightarrow$ IPv6 es sobre la falta de espacio de memoria
  - Combinar los archivos objeto `.o` cambia los espacios de memoria
  - ¿Y entre programas?

Todos los .o tienen el mismo espacio de memoria

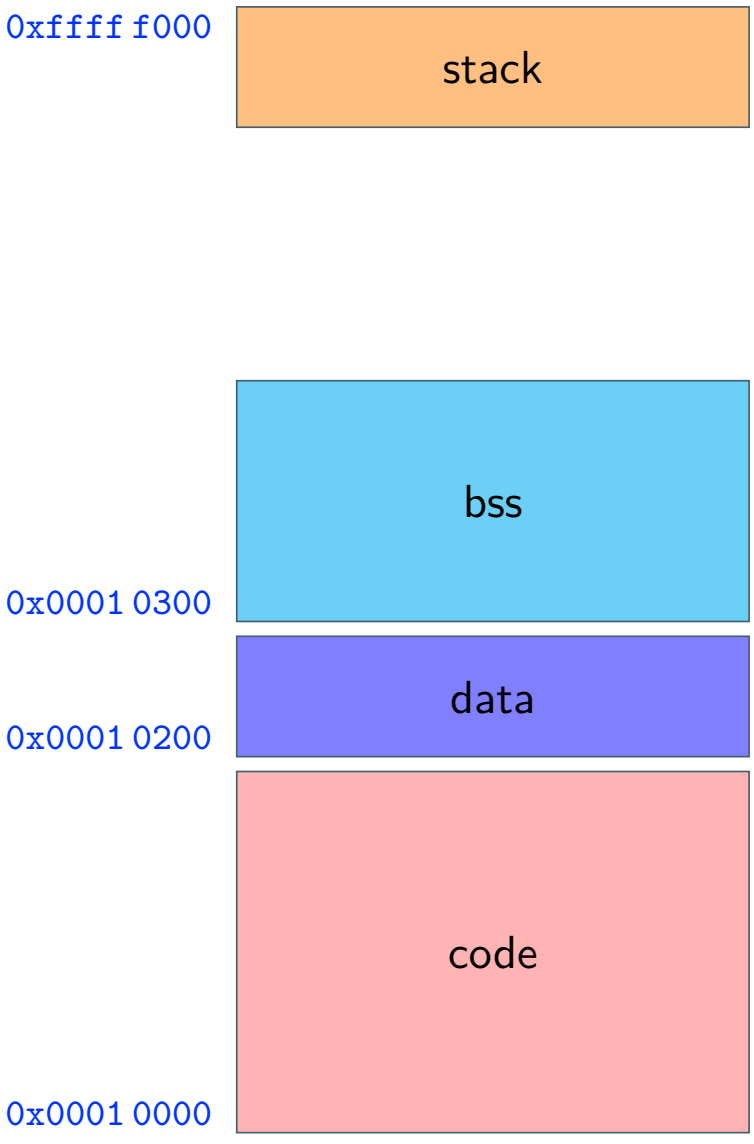


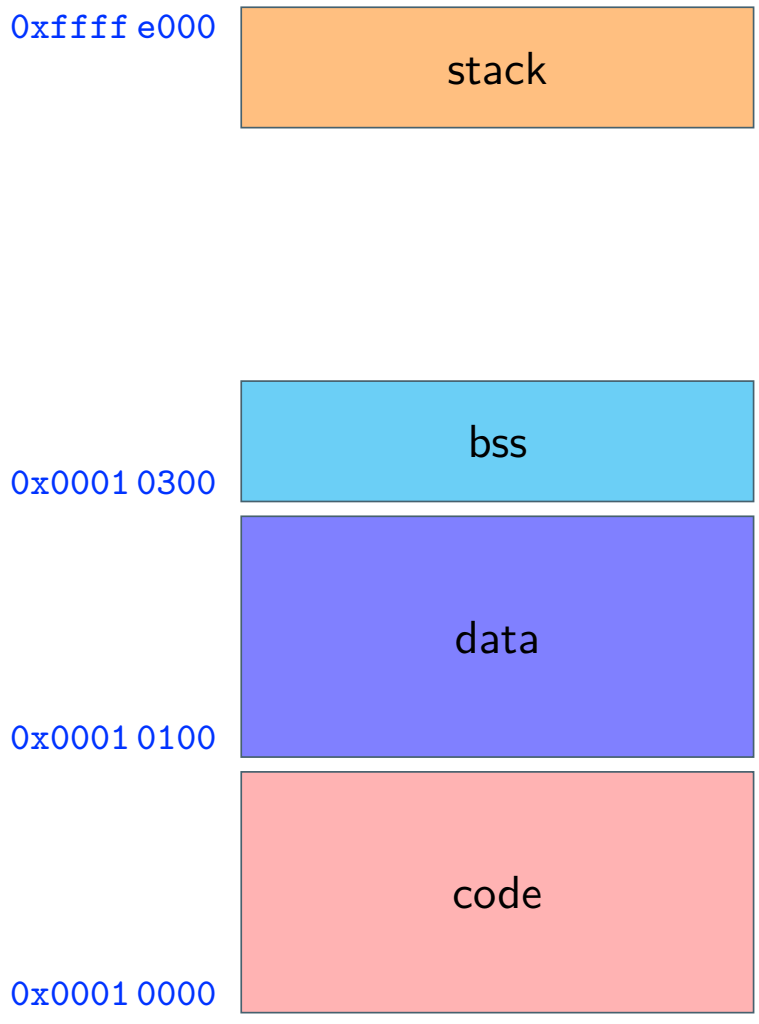


El `linker` combina los `.o` y cambia los espacios de memoria



Y para dos programas





Direcciones lógicas vs. físicas

- Direcciones lógicas
  - Cada programa tiene su propio espacio de memoria
  - Operaciones
    - Recuperar: dirección $\Rightarrow$ datos
    - Almacenar: dirección y datos $\Rightarrow$
  - Como lo ve el programador, compilador, y el *linker*
- Direcciones físicas
  - Donde termina el programa en la memoria
  - No pueden cargarse todos en la misma dirección (e.g., `0x 1000 0000`)

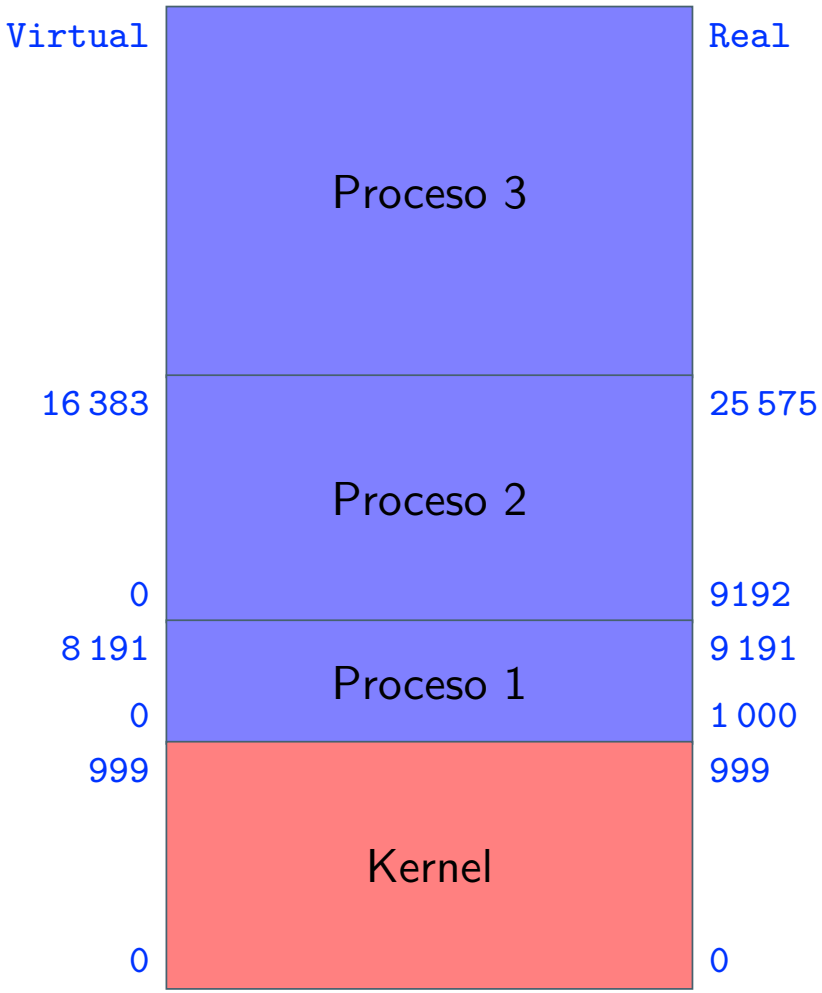
Reconciliar memoria física y lógica

- Los programas pueden tomar turnos en la memoria
  - Requiere el intercambio (*swap*) de programas al disco
  - Muy lento
- Podemos ejecutar programas en otras direcciones más allá de las enlazadas (*linked*)
  - Requiere usar un *linker* para "relocalizar" antes de ejecutar el programa
  - Técnica usada por algunos sistemas operativos antiguos
  - Lento, complejo, o ambos
- No existe la magia
  - Insertar un nivel de indirección

Teoría de tipos

- Comportamiento de la memoria física
  - Recuperar: dirección ⇒ datos
  - Almacenar: dirección y datos ⇒
- Como considera el proceso a la memoria
  - Recuperar: dirección ⇒ datos
  - Almacenar: dirección y datos ⇒
- Objetivo: cada proceso tiene su propia memoria
  - `proc-id` ⇒ Recuperar: dirección ⇒ datos
  - `proc-id` ⇒ Almacenar: dirección y datos ⇒
- ¿Qué es lo que realmente sucede?
  - `proc-id` ⇒ `map`: (memoria virtual ⇒ memoria física)
  - La máquina hace "recuperar y mapear" y "almacenar y mapear"

Funciones de mapeo simples



- P_1
  - `if` $V > 8191$ `error`
  - `else` $P = 1000 + V$
- P_2
  - `if` $V > 16383$ `error`
  - `else` $P = 9192 + V$
- Espacio de memoria
  - Dirección base
  - Límite

Mapeo de memoria contigua

- El procesador contiene dos registros de control
  - Memoria base
  - Memoria límite
- Cada acceso a la memoria revisa

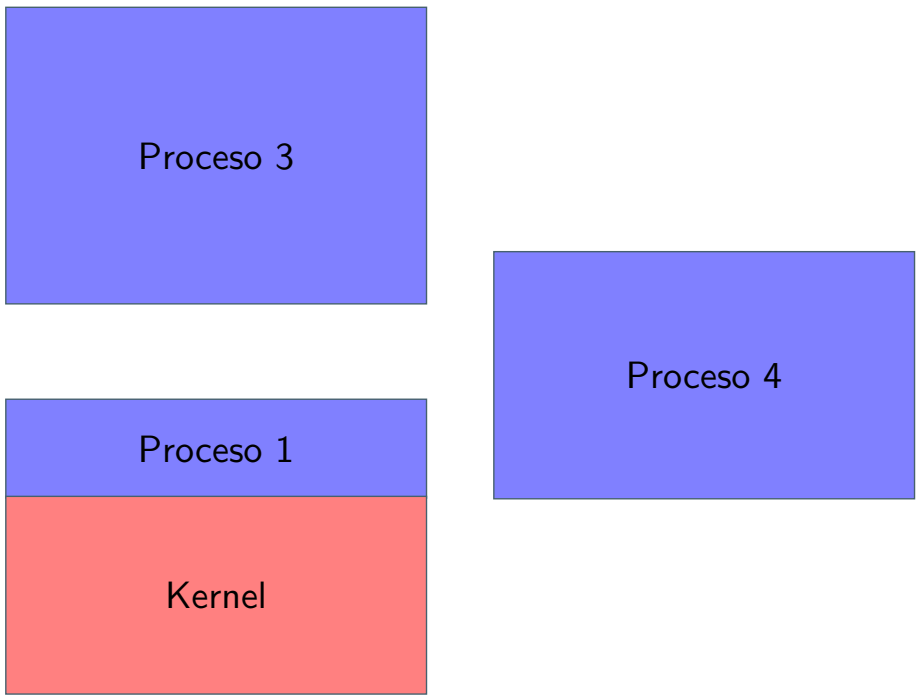
```
if V < limit
  P = base + V;
else
  ERROR // ¿han visto este error antes?
```

- Durante cada cambio de contexto
  - Salvar y cargar los registros visibles al usuario
  - Cargar la memoria base del proceso, y los registros del límite

Problemas con alojamiento contiguo

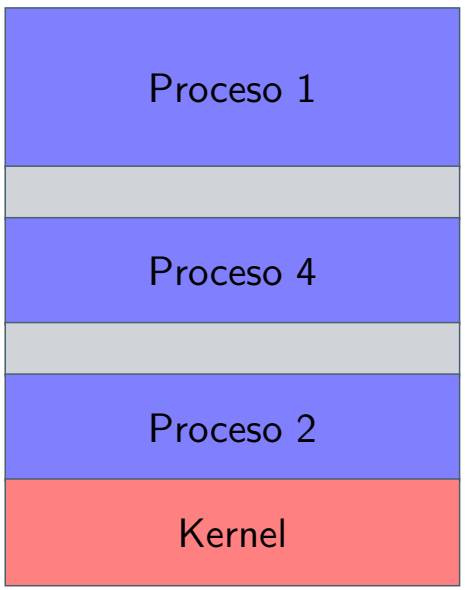
- ¿Cómo hacemos crecer un proceso?
  - Debemos incrementar el valor del límite
  - No podemos expandirnos dentro de la memoria de otro proceso
  - Debemos de mover todo el espacio de memoria a una localización con más espacio
    - Muy cara solución
- Fragmentación
  - ¿Tenemos que tener todo el programa en memoria al mismo tiempo?
- Residencia parcial de la memoria

¿Podemos ejecutar el proceso 4?



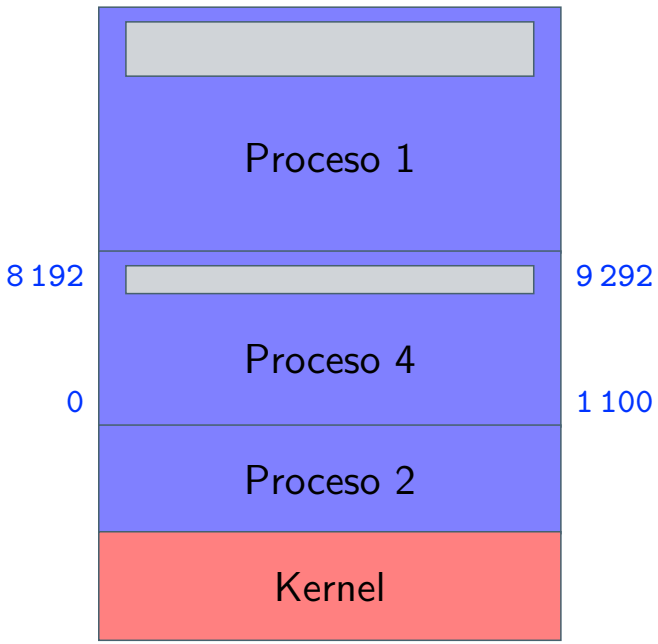
- El término de procesos crea "agujeros"
- Nuevos procesos pueden ser más grandes
- Puede requerir que movamos todo el espacio de memoria

Terminología: fragmentación externa



- Los pedazos de memoria disponibles son muy pequeños
- No podemos almacenar objetos grandes
- Puede deshabilitar muchas partes de la memoria
- ¿Cómo lo podemos resolver?
  - Compactando (muy costoso)
  - También conocido como: “detener y copiar”

Terminología: fragmentación interna



- Los alojamientos de memoria se redondean
  - Barrera de 8 K (alguna potencia de 2)
- Se desperdicia memoria dentro de cada segmento
- No se puede resolver a través de compactación
- Los efectos no son fatales

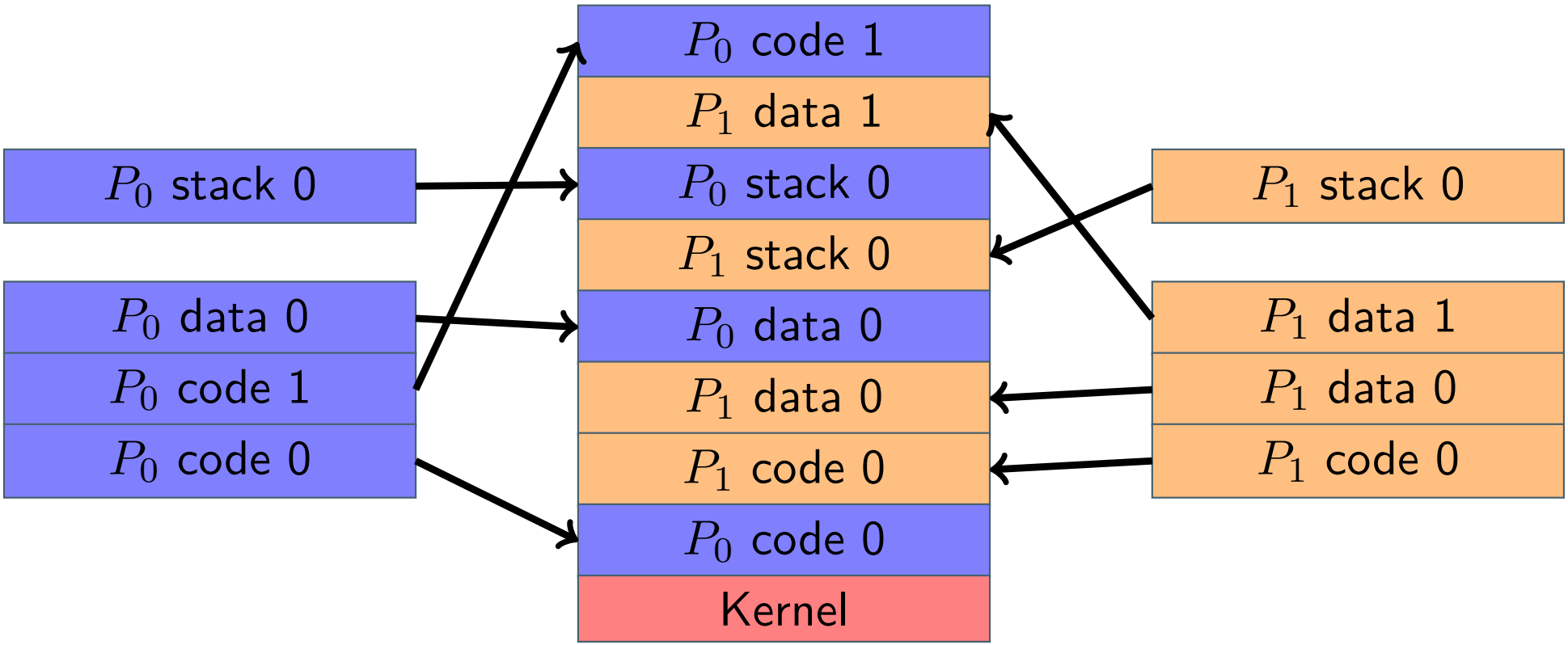
Swapping

- Procesos de mutiples usuarios
  - Sumatoria de las demandas de memoria es mayor que la memoria del sistema
  - Objetivo: permitir que cada proceso tenga 100 % de la memoria del sistema
- Tomar turnos
  - Temporalmente colocamos los procesos en el disco
    - No se pueden ejecutar
    - Se bloquean en solicitudes I/O implícitas (e.g., *swapread*)
  - El demonio de *swap* cambia los procesos de y hacia el disco
  - Puede tomar segundos por proceso
    - Analogía moderna: las laptos se suspenden hacia el disco
  - Tal vez necesitamos un mejor plan

Alojamiento contiguo lleva a la paginación

- Resuelve varios problemas
  - Problema del crecimiento de problemas
  - Problema de compactación de la fragmentación
  - Tiempo de espera en el cambio (*swap*) de procesos
  - Enfoque: dividir la memoria en grano más fino
    - Página: región pequeña de la memoria virutal (0.5 K, 4 K, 8 K, etc.)
    - Cuadro: región pequeña de la memoria física (mismo tamaño que las páginas)
  - Idea principal
    - Cualquier página puede mapearse para ocupar cualquier cuadro

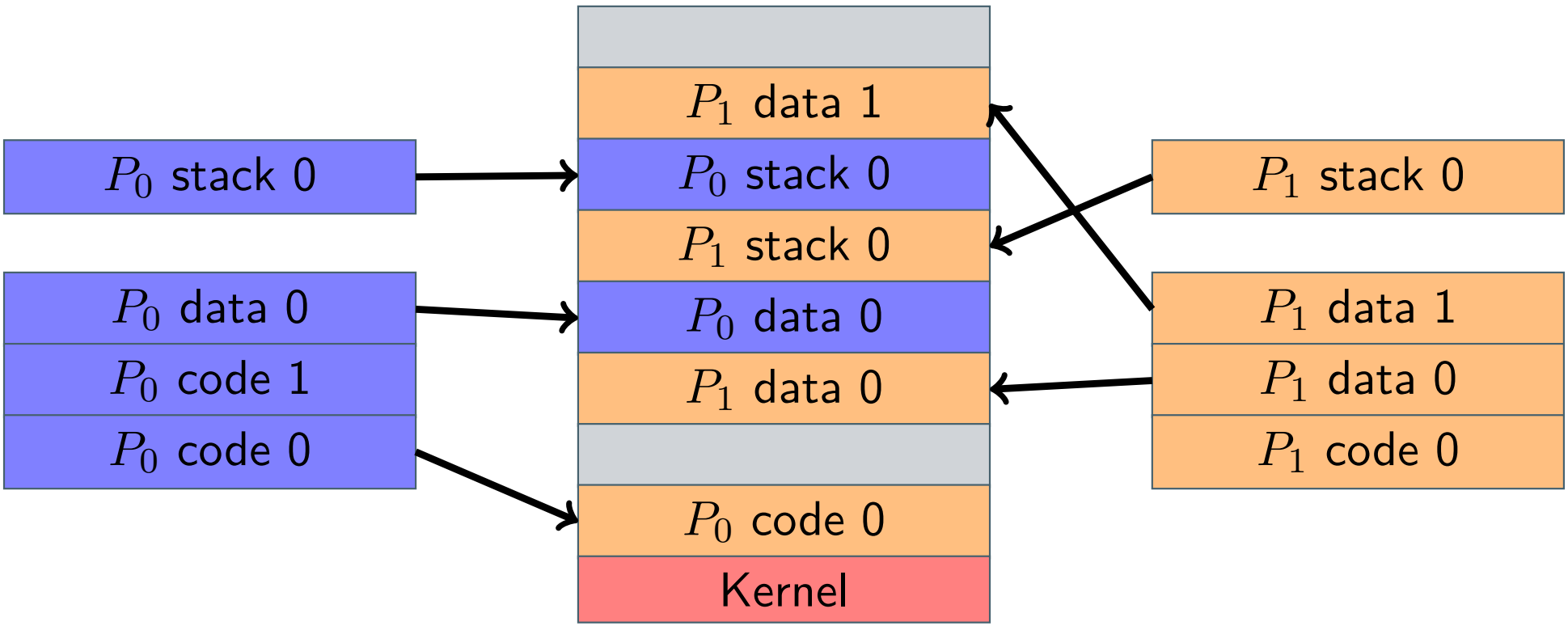
Mapeo de páginas por proceso



Problemas resueltos por la paginación

- Problema de crecimiento de procesos
  - Cualquier proceso puede utilizar cualquier cuadro disponible para cualquier propósito
- Problema de compactación de fragmentación
  - Los procesos no necesitan estar contiguos, así que no compactamos
- Problema de retraso de cambio
  - Podemos cambiar (*swap*) una parte del proceso en lugar de todo el proceso

Residencia parcial



Debemos evolucionar las estructuras de datos

- Alojamiento contiguo
  - Cada proceso está descrito por un par `(base, límite)`
- Paginación
  - ¿Está cada página descrita por `(base, límite)`?
    - Típicamente, las páginas de un sistema tienen un mismo tamaño
  - Entonces, cada página está descrita por `(dirección base)`
  - Página arbitraria $\Rightarrow$ el mapeo de cuadros requiere trabajo
    - Estructura de datos abstracta: `map`
  - Implementada como
    - Lista enlazada
    - Arreglo
    - Tabla de hash
    - Lista por saltos (*skip list*)
    - Árbol biselado (*splay tree*)

Opciones de la tabla de páginas

- Lista enlazada
  - $O(n)$, entonces $V \Rightarrow P$ se tarda más para muchas direcciones de memoria
- Arreglo
  - Tiempo de acceso constante
  - Requiere (mucha) memoria contigua para la tabla
- Tabla de hash
  - Vagamente tiempo de acceso constante
  - No está acotada (realmente)
- Árbol biselado (*splay tree*)
  - Excelente tiempo esperado amortizado
  - Muchas lecturas y escrituras a la memoria para un mapeo
  - No se ha demostrado en hardware

Solución con un arreglo

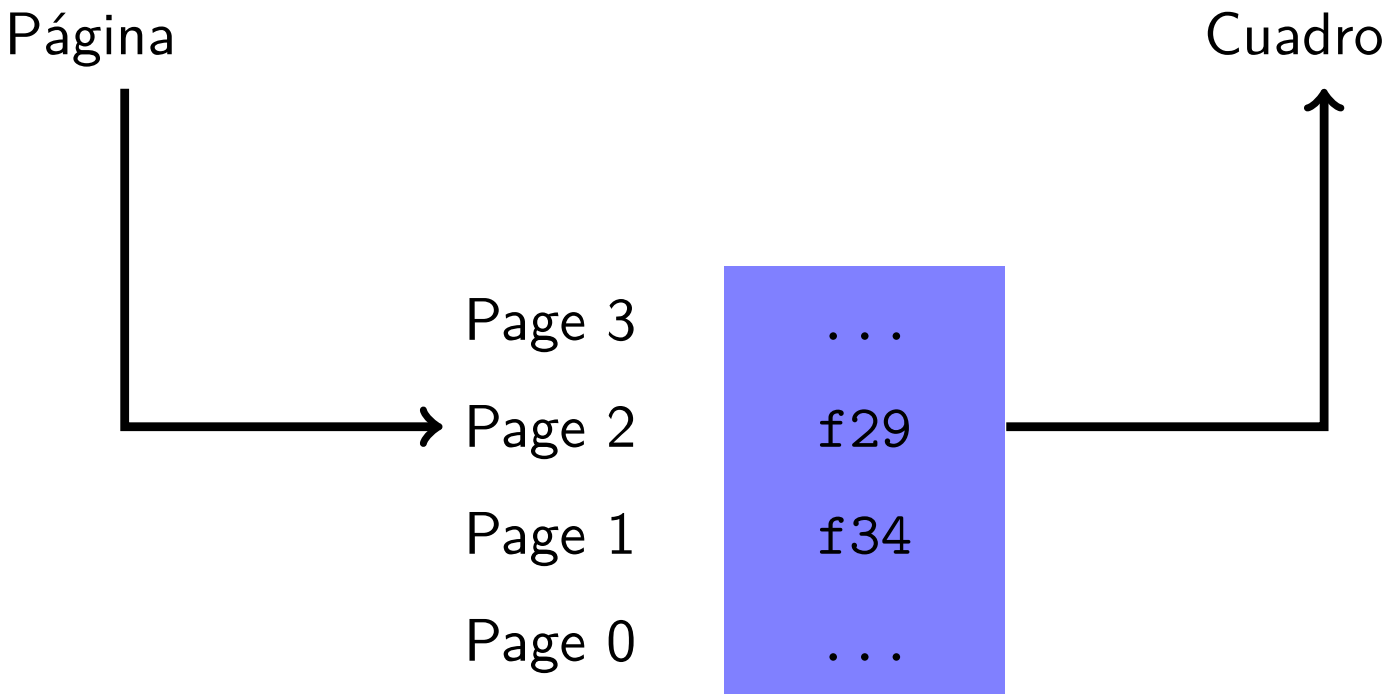
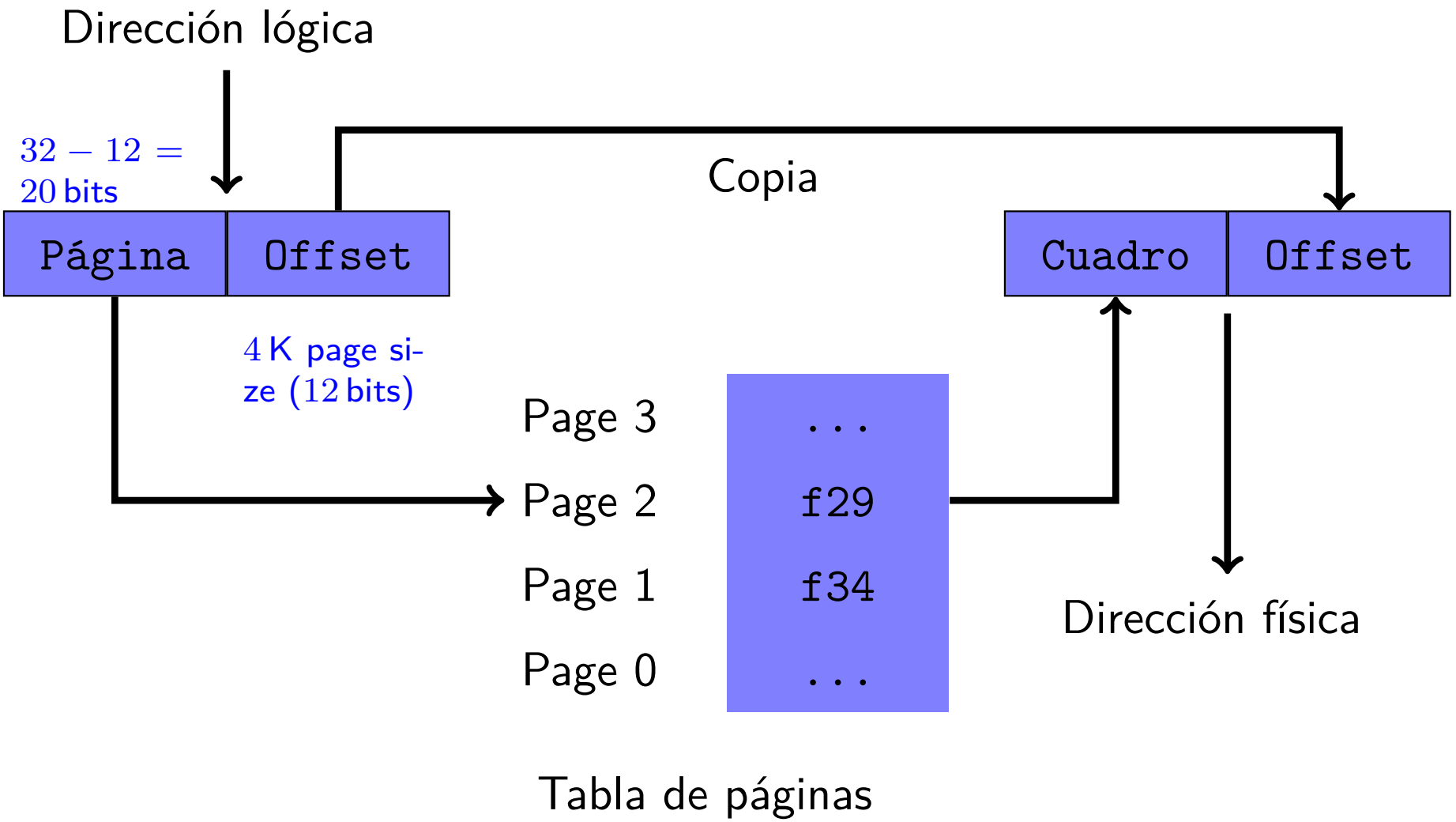


Tabla de páginas

Mapeo de direcciones



Mapeo de direcciones

- Vista de usuario
  - La memoria es un arreglo lineal
- Vista de sistema operativo
  - Cada proceso requiere N cuadros
  - Los cuadros pueden estar en cualquier parte
- Fragmentación
  - Cero fragmentación externa
  - Fragmentación interna: en promedio media página por region

Bookkeeping

- Una tabla de páginas por cada proceso
- Una tabla global de cuadros
  - Administra los cuadros disponibles
  - Típicamente recuerda quien es dueño de que cuadro
- Cambio de ambiente
  - Debe activar (cambiarse a) la tabla de páginas del proceso

Técnicas de hardware

- Número pequeño de páginas
  - La tabla de páginas puede ser unos pocos registros
  - PDP-11: 64 K espacio de memoria
    - 8 páginas de 8 K cada una —8 registros
- Caso típico
  - Tablas de paginación grandes, en memoria
    - El procesador tiene un "registro base de la tabla de páginas" (pueden llamarse de otra manera)
    - Se establecen durante el cambio de contexto (ámbito)

Problemas extras

- El programa necesita accesar a la memoria

```
movl (%esi), %eax
```

- El procesador hace dos accesos a la memoria
  - Divide la dirección en un número de página (offset intra-página)
  - Agrega los números de página al registro base de la tabla de páginas
  - Obtiene la entrada de la tabla de páginas (PTE) de la memoria
  - Concatena la dirección del cuadro con el offset intra-página
  - Obtiene los datos del programa de la memoria desde %eax
- Solución: translation lookaside buffer —TLB— (coming soon)

Mecánicas de las entradas de la tabla de páginas (PTE)

- PTE trabajo conceptual
  - Especifica un número de cuadro
- PTE banderas
  - Bit válido
    - No establecido significa que el acceso debería de generar una excepción
  - Protección
    - bits de lectura, escritura, y ejecución
  - Bit de referencia (bit sucio —dirty bit)
    - Establecido si la página fue escrita o leída recientemente
    - Usada cuando paginamos al disco (lectura posterior)
  - Especificado por el sistema operativo para cada página/cuadro
    - Inspeccionado y actualizado por hardware

Estructura de tabla de páginas

- Problema
  - Asumamos páginas de 4 KByte, 4 Byte PTEs
  - Razón: $1024 : 1$
    - 4 GByte dirección virtual (32 bits) $\Rightarrow$ 4 MByte tabla de páginas
- Una solución: registro de tamaño de la tabla de páginas (PTLR)
  - (El nombre puede cambiar)
  - Muchos programas no utilizan todo el espacio de memoria virtual
  - Restringir un proceso a utilizar las entradas $0, \ldots, N$ de la tabla de páginas
  - Registros embebidos detectan las referencias fuera de límites ($> N$)
  - Permite pequeños PT (page table) para pequeños procesos
    - Mientras un stack no esté lejos de los datos

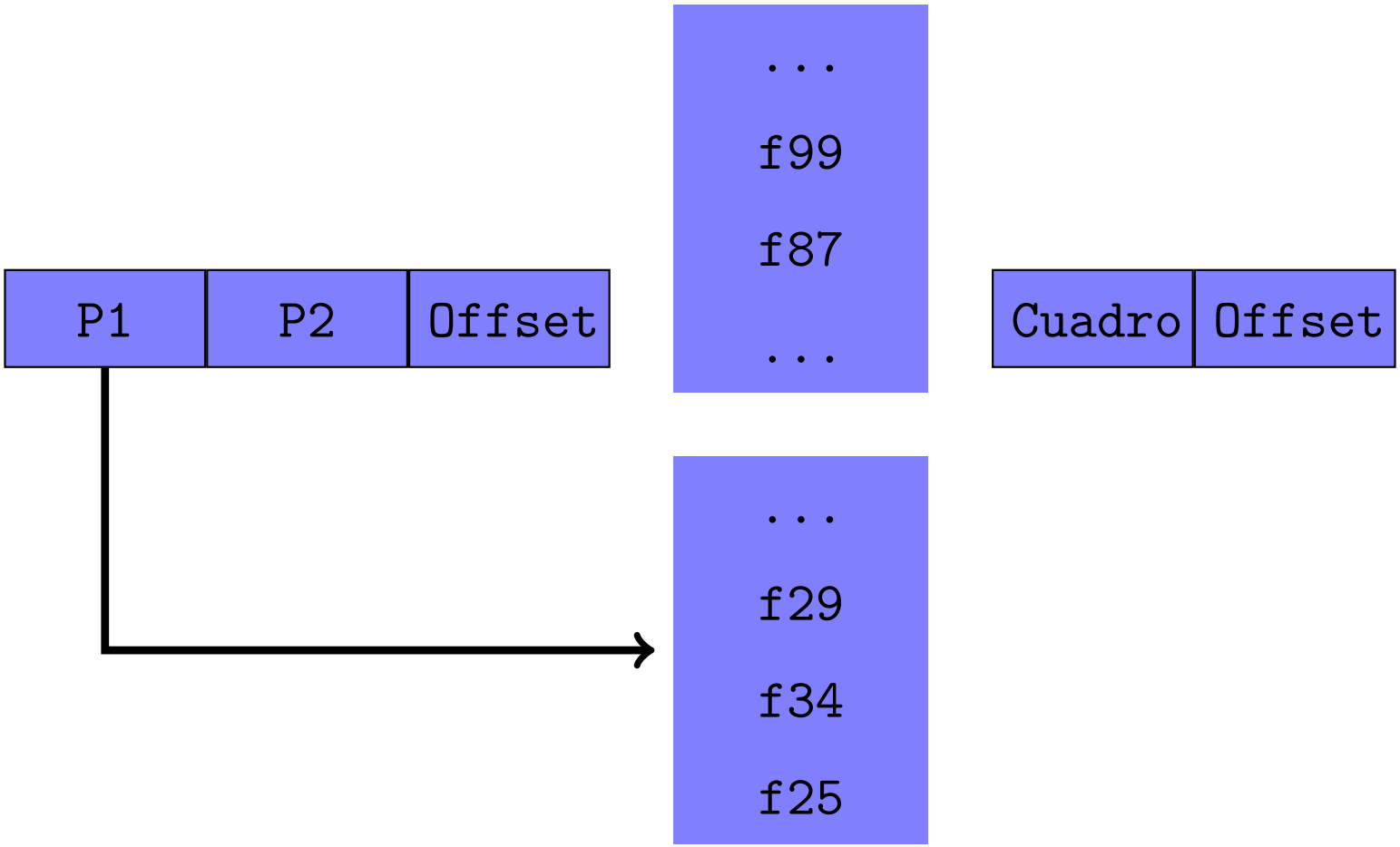
Estructura de tabla de paginación

- Observación clave
  - Cada tabla de paginación del proceso es un mapeo disperso (*sparse mapping*
  - Muchas páginas no son colocadas en cuadros
    - El espacio de direcciones es utilizado de manera esparcida
    - Hay un abismo entre el fondo del *stack* y la cima del *heap*
    - La mayoría del tiempo ocupa $99\,\%$ del espacio de direcciones
    - Algunas páginas están en el disco en lugar de la memoria
- Refinando nuestra observación
  - Las tablas de paginación no son dispersas de manera aleatoria
    - Ocupadas por regiones secuenciales de memoria
    - Text, rodata, data y bss, stack
- Una lista dispersa de listas densas

¿Cómo mapeamos una lista dispersa de listas densas?

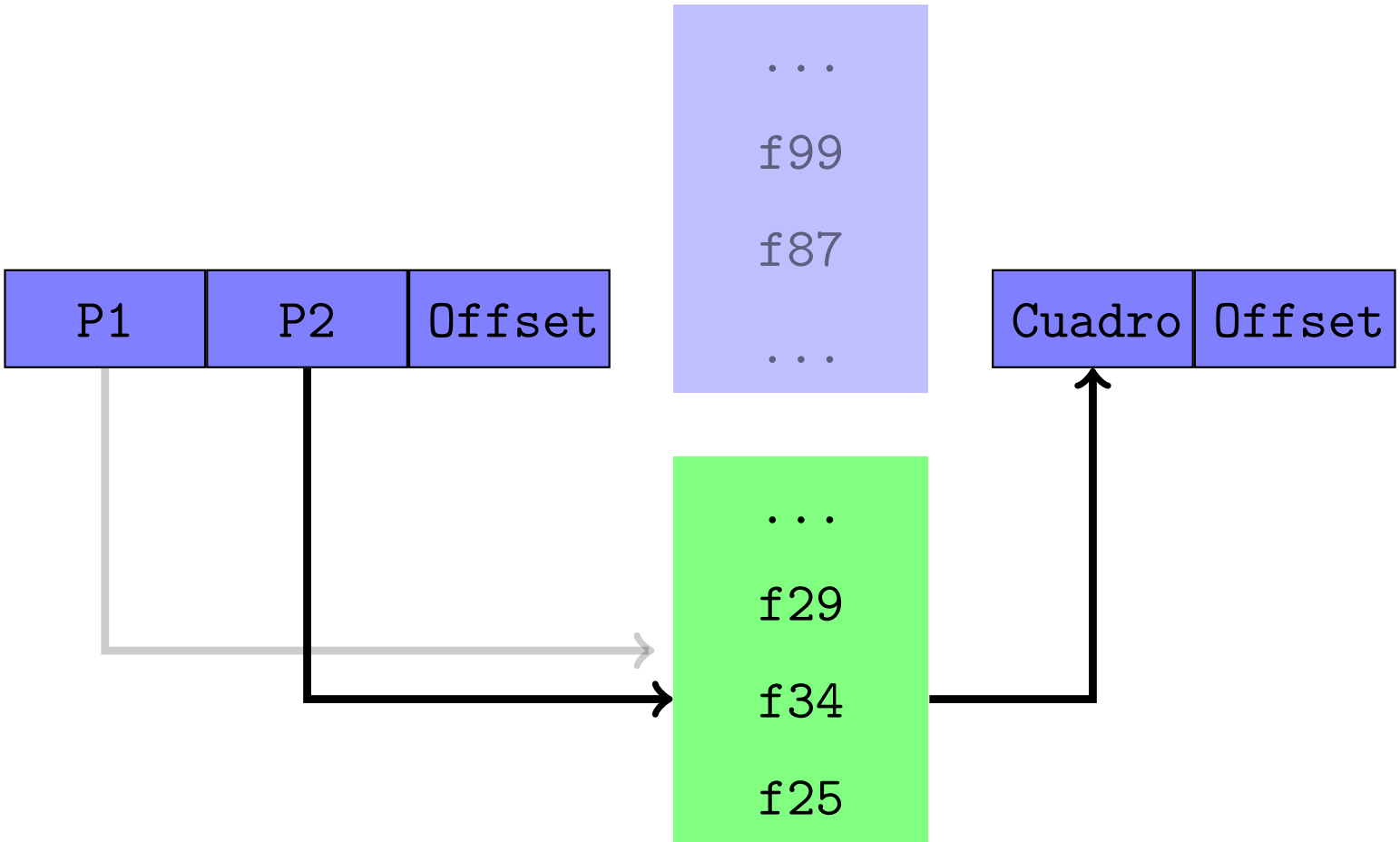
- No hay magia
  - Insertamos un nivel de indirección
- Tabla de paginación mutli nivel
  - Directorio de páginas mapea grandes partes del espacio de direcciones hacia ...
  - Tabla de paginación, que mapea páginas a cuadros
  - Conceptualmente, el mismo mapeo que la vez anterior
    - Pero la implementación es un árbol de dos niveles, no un solo paso

Tabla de paginación multi nivel



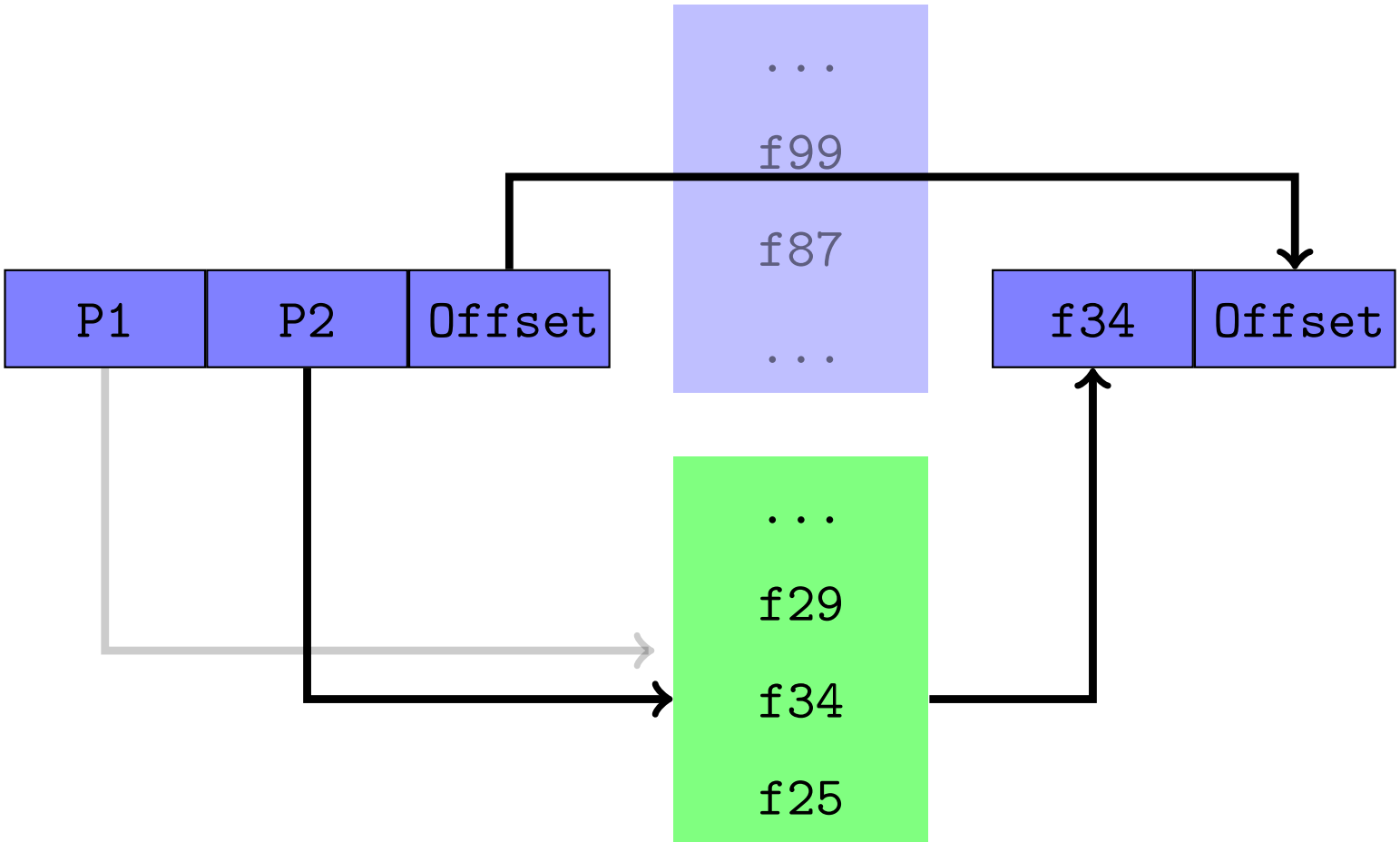
Tablas de páginas

Tabla de paginación multi nivel



Tablas de páginas

Tabla de paginación multi nivel



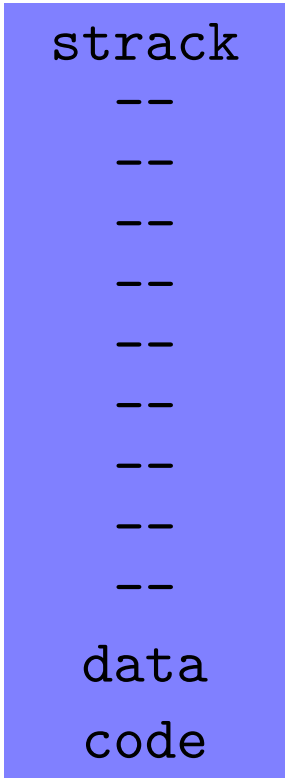
Tablas de páginas

Mapeo disperso

- Asumamos páginas de 4 KByte, 4 byte PTEs
  - Razón: $1024 : 1$
    - 4 GByte de memoria virtual (32 bits) $\Rightarrow$ 4 MByte tabla de paginación
- Ahora asumamos que el directorio de páginas tiene entradas de 4 byte (PDE)
  - 4 MByte tabla de paginación se convierte 1024 4 K tablas de paginación
  - Más 1024 entradas de paginas de directorio que apuntan a ellas
  - Resultado: 4 MByte + 4 KByte
- Espacio de memoria disperso
  - Significa que la mayoría de las páginas contribuyen a nada al mapear
  - La mayoría de las tablas de paginación contienen entradas de "no cuadro"
  - Remplazan las PT con "`null pointer`" en el directorio de páginas
  - Resultado: espacio de direcciones de 4 GB vacío especificado por un directorio de 4 KB

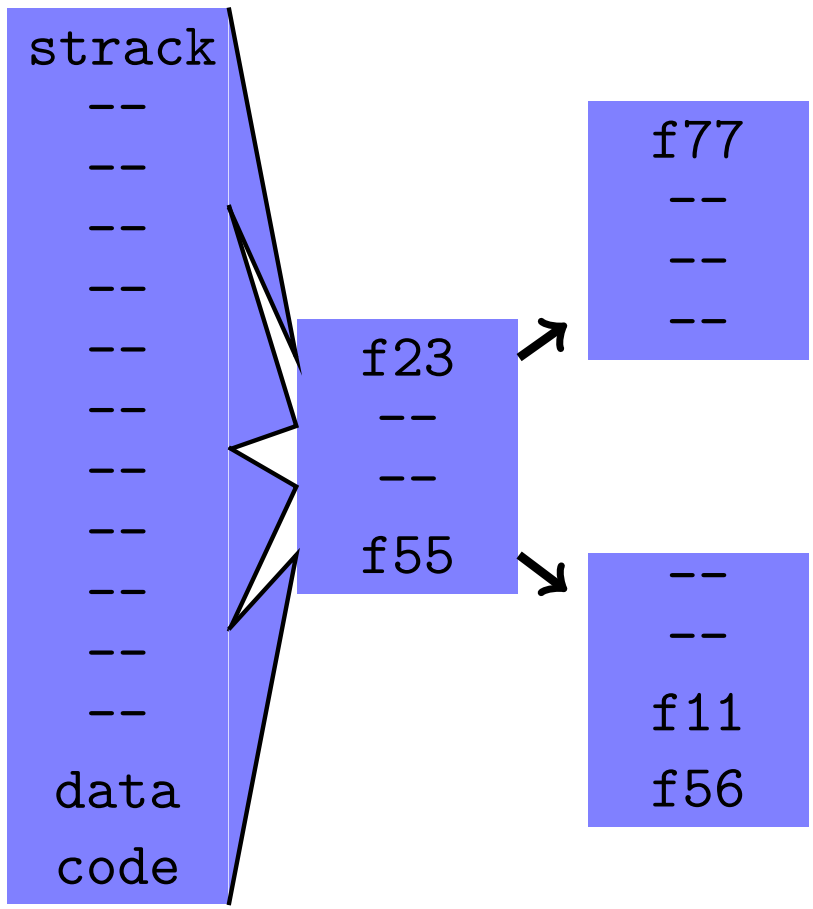
Espacio de direcciones disperso

- El espacio de direcciones esta mayormente en blanco
  - Lecturas y escrituras deberán fallar
- Comprimimos el medio
  - El espacio de direcciones disperso debe de usar una estructura de mapeo pequeña
  - El espacio de direcciones al estar lleno puede justificar una estructura de mapeo más grande



Espacio de direcciones disperso

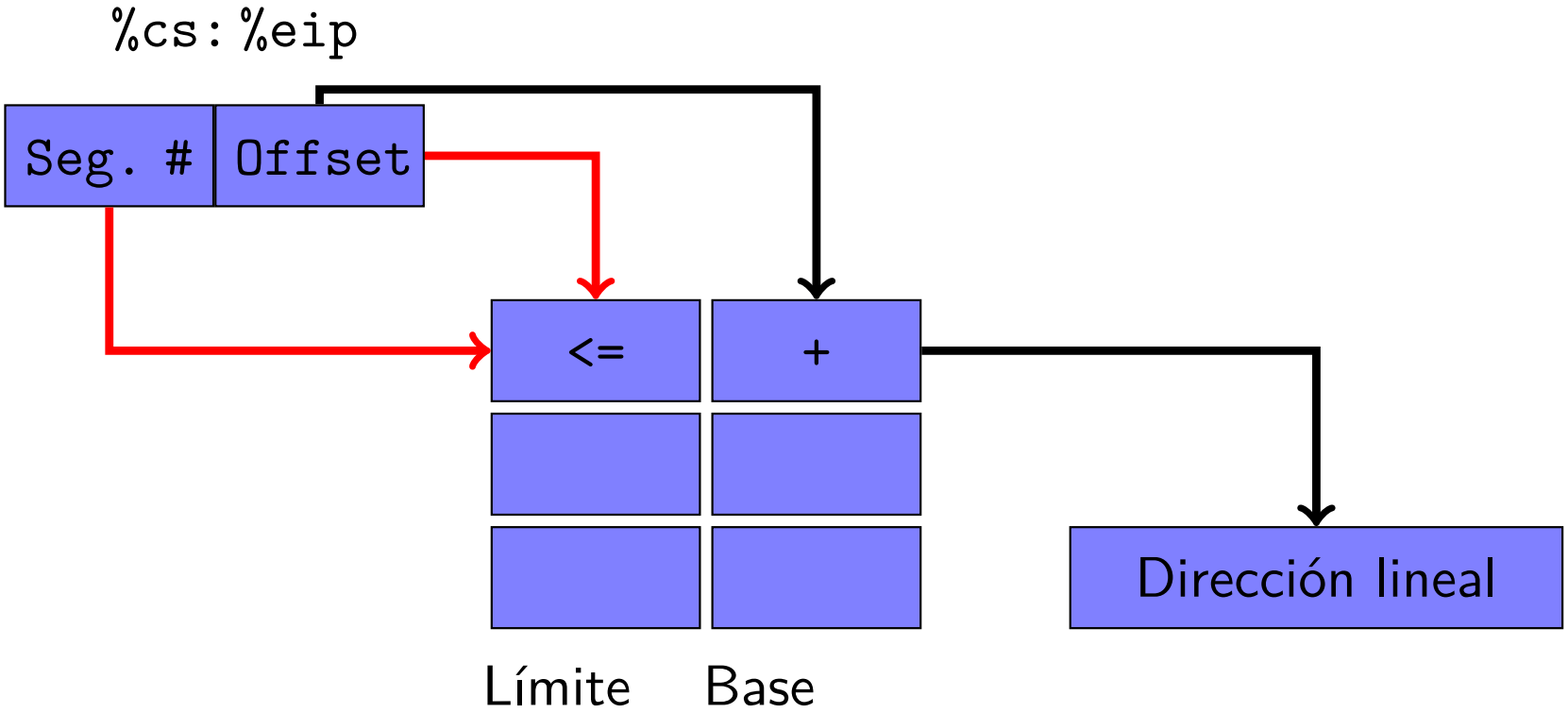
- Directorio de páginas disperso
  - Punteros a PT no vacías
  - "`null`" en lugar de PT vacías
- Caso común
  - Necesita 2 o 3 tablas de paginación
    - Un o dos mapean código y data
    - Uno mapea el stack
  - El directorio de paginación tiene 1024 entradas
    - 2-3 apunta a PTs
    - El resto no está presente
- Resultado
  - 2–3 PTs, 1 PD
  - Mapear todo el espacio de direcciones con 12–16 Kbyte, no 4 Mbyte



Segmentación

- La memoria física es (mayormente) lineal
- ¿Es la memoria virtual lineal?
  - Típicamente es un conjunto de regiones
  - Módulo = región de código + región de datos
  - Región por stack
  - Región de heap
- ¿Por qué importan las regiones?
  - Una barrera de protección natural
  - Una barrera para compartir natural

Mapeo



Segmentación y paginación

- La 80386 (lo hace todo)
  - Direcciones del procesador dirigidas a uno de los seis segmentos
    - `CS`: segmento de código
    - `DS`: segmento de datos
    - Offset de 32-bits dentro de un segmento —`CS:EIP`
  - Tabla de descriptores mapea el selector al segmento del descriptor
  - El offset es dado al descriptor de segmento, y genera una dirección lineal
  - La dirección lineal es dada al directorio de paginación, y tabla de paginación
  - Detalles en el texto

Teoría de tipos de la x86

- Instrucción ⇒ selector de segmento
  - `pushl` implícitamente especifica el selector en `%ss`
- Proceso ⇒ (selector ⇒ (base, límite))
  - Tablas de descriptores globales y locales
- Segmento, dentro de la dirección de segmento ⇒ dirección lineal
  - `cs:eip` significa `eip` más la base del segmento de código
- Proceso ⇒ (dirección lineal alta ⇒ tabla de paginación)
  - Registro base del directorio de paginación
  - Indexación del directorio de paginación
- Tabla de paginación: dirección lineal media ⇒ dirección del cuadro
- Memoria: dirección del cuadro más offset

Resumen

- El proceso emite una dirección virtual
  - Basada en segmentos o lineal
- Un proceso mágico mapea la memoria virtual a física
- La magia no existe
  - Verificar la validez de la dirección
  - Revisar permisos
  - El mapeo puede fallar (trampa que administra errores)
- Estructuras de datos determinadas por los patrones de acceso
  - La mayoría de los espacios de direcciones son alojados dispersamente

Práctica

Un proceso en un sistema que utiliza memoria virtual tiene la siguiente tabla de paginación

Página	Cuadro
0	0
1	4
2	1
3	2

Asuma que la dirección virtual está representada con 6 bits, donde los primeros dos bits son usados para determinar la página.

- ¿Cuál es el tamaño de la página y el cuadro?
- Asuma que el proceso hace las llamadas a la memoria `100110` y `001010`. Calcule la dirección física en el sistema con 8 cuadros en la memoria física.