udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA

Memoria Virtual (cont.)

Adín Ramírez
adin.ramirez@mail.udp.cl

Sistemas Operativos (CIT2003-1)
1er Semestre 2015

Adaptado de Dave Eckhardt

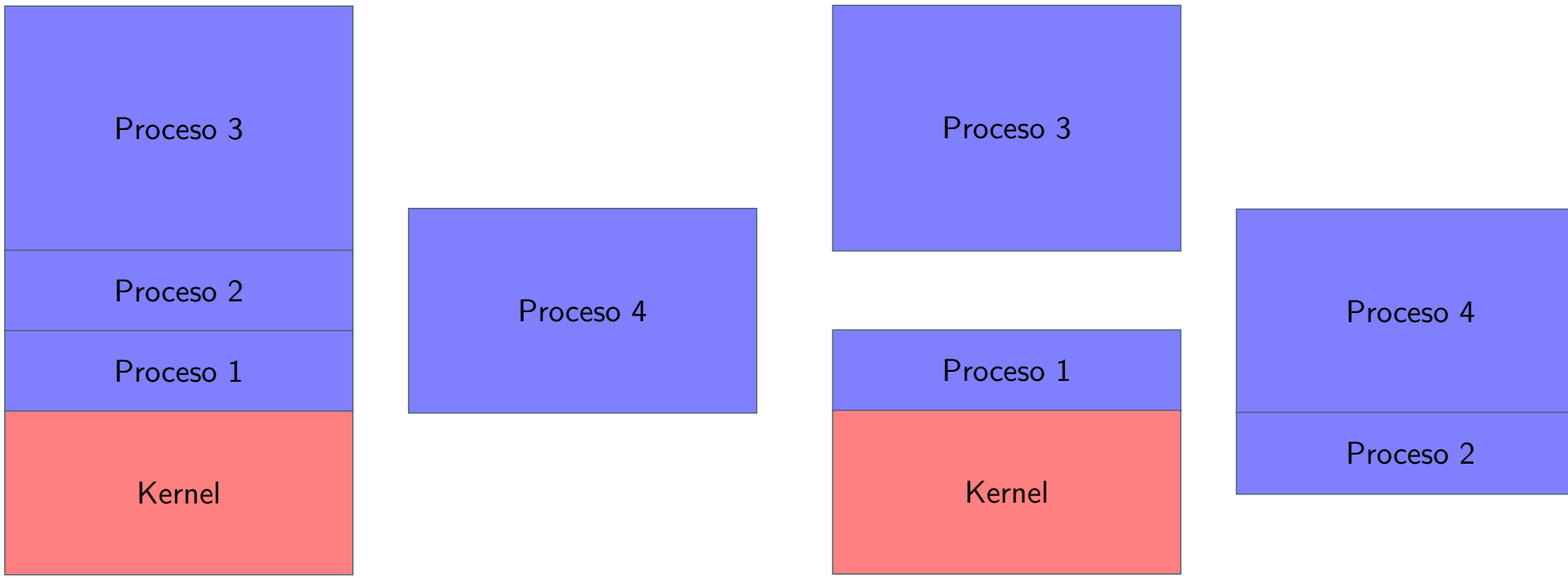
Clase pasada

- Problema de mapeo: direcciones lógicas vs. físicas
- Mapeo de memoria contigua (base y límite)
- *Swapping* —tomar turnos en memoria
- Paginación
  - Un arreglo mapea los números de página a números de cuadro
  - Observación: una tabla típica está ocupada de manera dispersa
  - Respuesta: estructuras de datos dispersas (e.g., arreglo de dos niveles)

Swapping

- Procesos multi-usuario
  - Las sumatoria de las demandas de los procesos $>$ memoria del sistema
  - Objetivo: permitir que cada proceso pueda tener $100\,\%$ de la memoria del sistema
- Tomar turnos
  - Temporalmente enviar los procesos a disco
  - Un demonio de cambio de contexto que envía los procesos al y desde el disco
  - Puede demorar segundos por proceso
  - Crea un problema de fragmentación externa

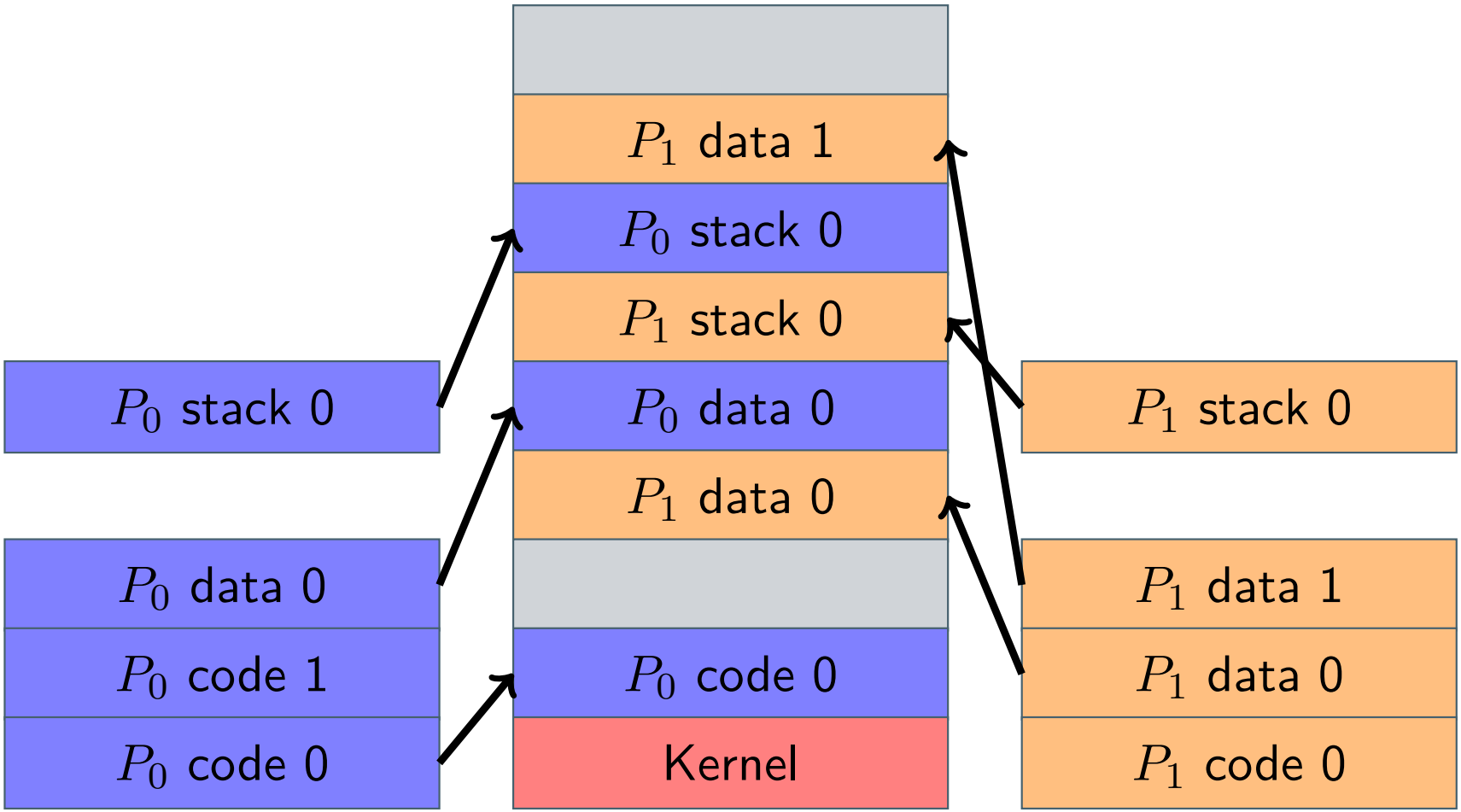
Fragmentación externa



Beneficios de la paginación

- Problema del crecimiento de procesos
  - Cualquier proceso puede usar cualquier cuadro libre para cualquier propósito
- Problema de compactación de la fragmentación
  - Los procesos no necesitan ser contiguos
- Demoras por cambiar los procesos (*swapping*)
  - Podemos cambiar parte de los procesos en lugar del proceso completo

Residencia parcial



Entradas en la tabla de paginación (PTE)

- Bit válido o presente —establecido por el SO
  - Si el puntero al cuadro es válido, no tenemos que dar error
- Bits de protección —establecidos por el SO
  - Permisos de escritura, lectura, y ejecución
- Bit sucio
  - El hardware lo establece de 0 a 1 cuando los datos han sido almacenados en la página
  - El SO establece de 1 a 0 cuando la página ha sido escrita a disco
- Bit de referencia
  - Hardware establece de 0 a 1 cuando cualquier página ha sido accesada
  - OS lo utiliza para poder mover las páginas

¿Qué vamos a discutir hoy?

- La optimización a través de TLB
- Residencia parcial en memoria en acción
- El manejador de error de página

Accesos a memoria

- Los programas requieren accesos a la memoria
- El procesador hace dos accesos a la memoria
  - Dividir la dirección en el número de página y el offset (paso) intra-página
  - Agregar al registro base a la tabla de paginación
  - Obtener la entrada de la tabla de paginación (PTE) de la memoria
  - Agregar la dirección del cuadro, offset intra-página
  - Obtener datos de la memoria
- Puede ser peor
  - x86 directorio de páginas y tabla de páginas
    - Necesita tres acceso físicos por cada dirección virtual
  - x86-64 tiene un sistema de mapeo de páginas de cuatro niveles

Translation Lookaside Buffer (TLB)

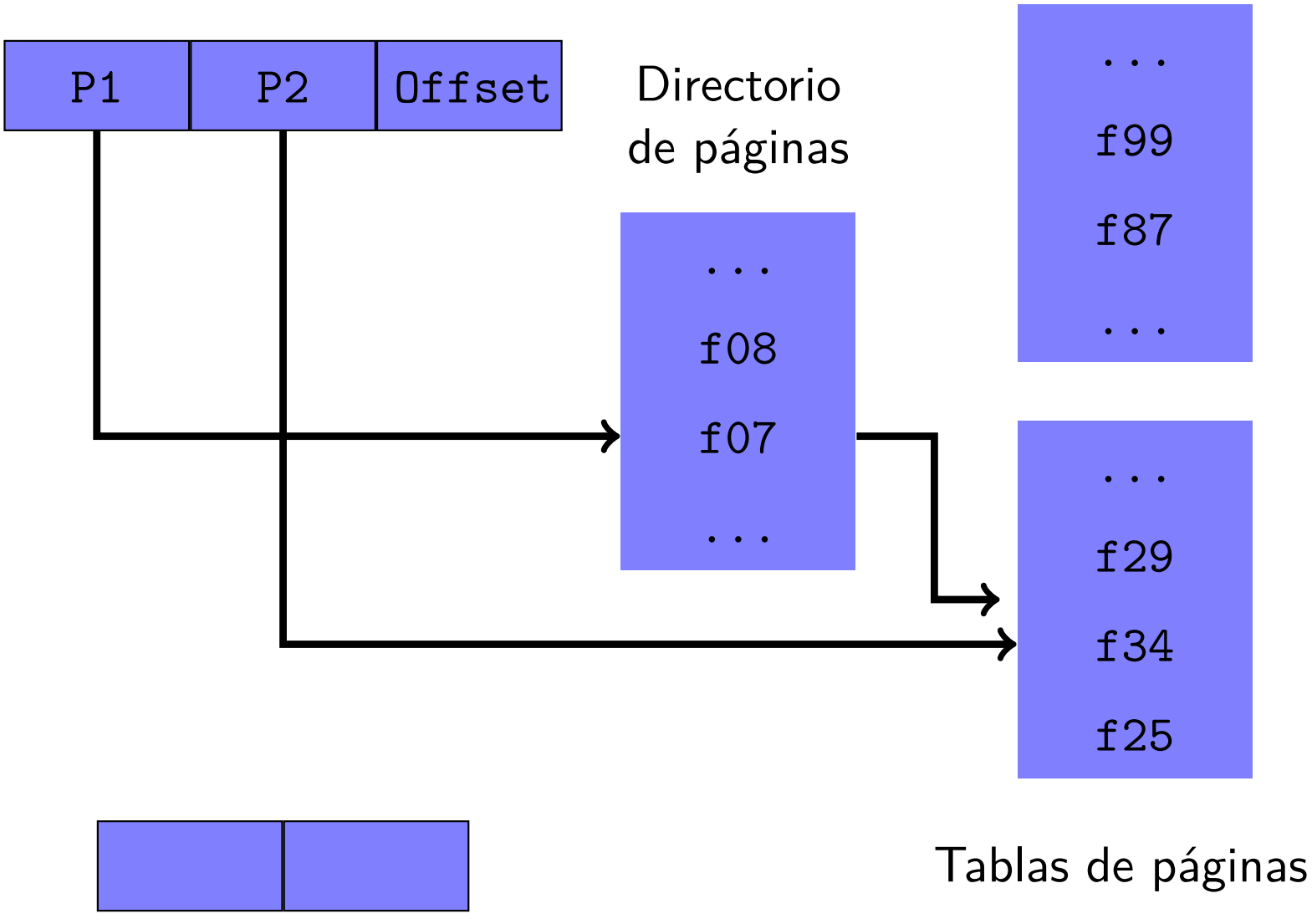
- Problema
  - No podemos permitir latencia de memoria debido a accesos dobles, triples, etc.
- Observación: "localidad de la referencia"
  - Un programa accesa típicamente memoria "cercana"
  - La siguiente instrucción a menudo está en la misma página que la instrucción actual
  - El siguiente byte de un string está a menudo en la misma página que el byte actual
  - Un arreglo es bueno, una lista enlazada no tanto
- Solución
  - Un mapa de páginas por hardware que mantenga los mapeos virtual a físico
  - Pequeña y rápida memoria en un chip
  - Gratis, en comparación con los cálculos fuera del chip

El TLB más simple

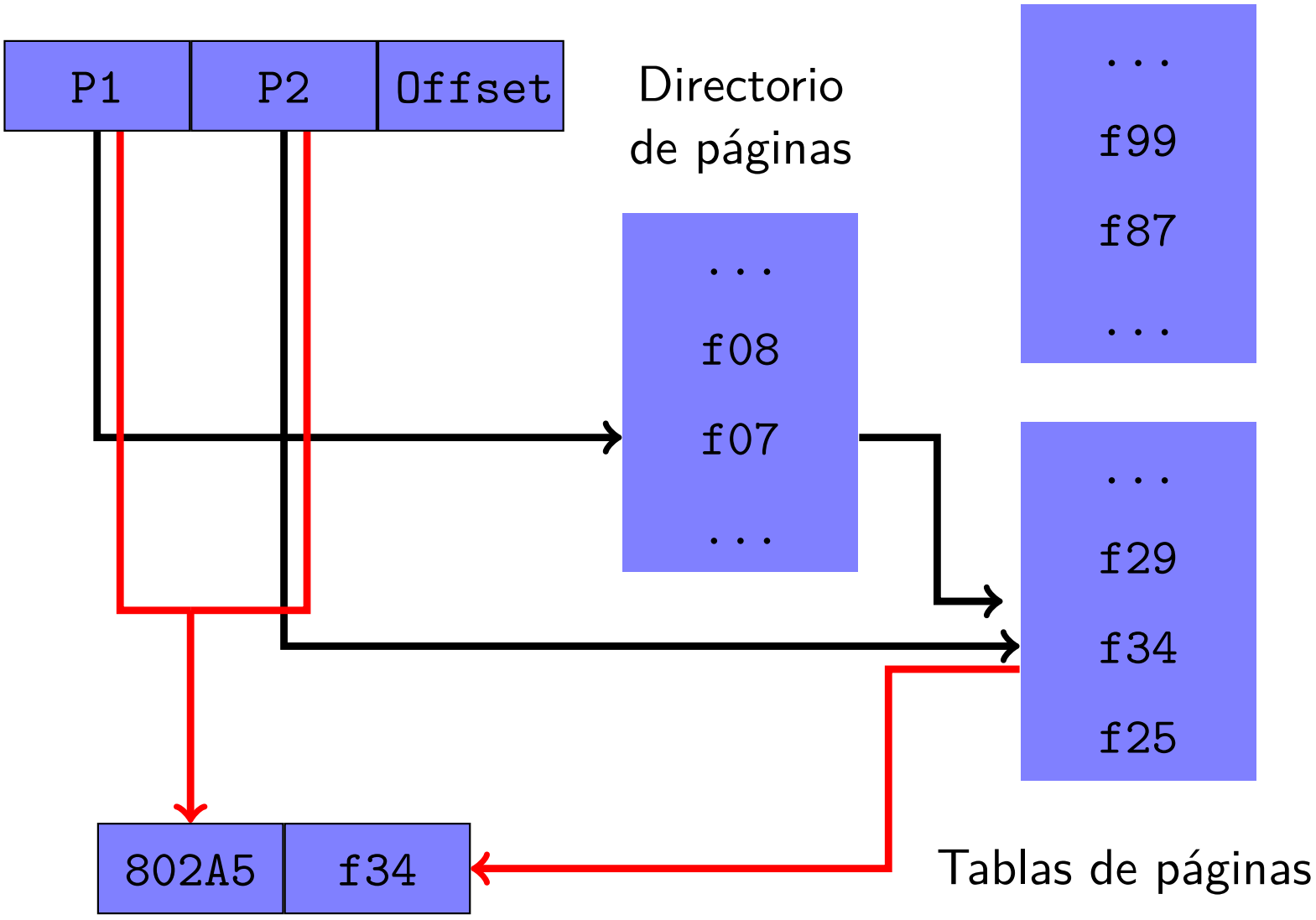
Enfoque

- Recordemos la traducción más reciente de dirección virtual a física
  - Obtenida de cualquier forma, e.g., directorio de páginas más tabla de páginas
- Observemos si la siguiente memoria solicitada está en la misma página
  - En ese caso, saltamos PD/PT y utilizamos el mismo cuadro
  - 3 veces aceleración
  - El costo son dos registros de 20 bits (nuestra suposición, ver clase anterior)

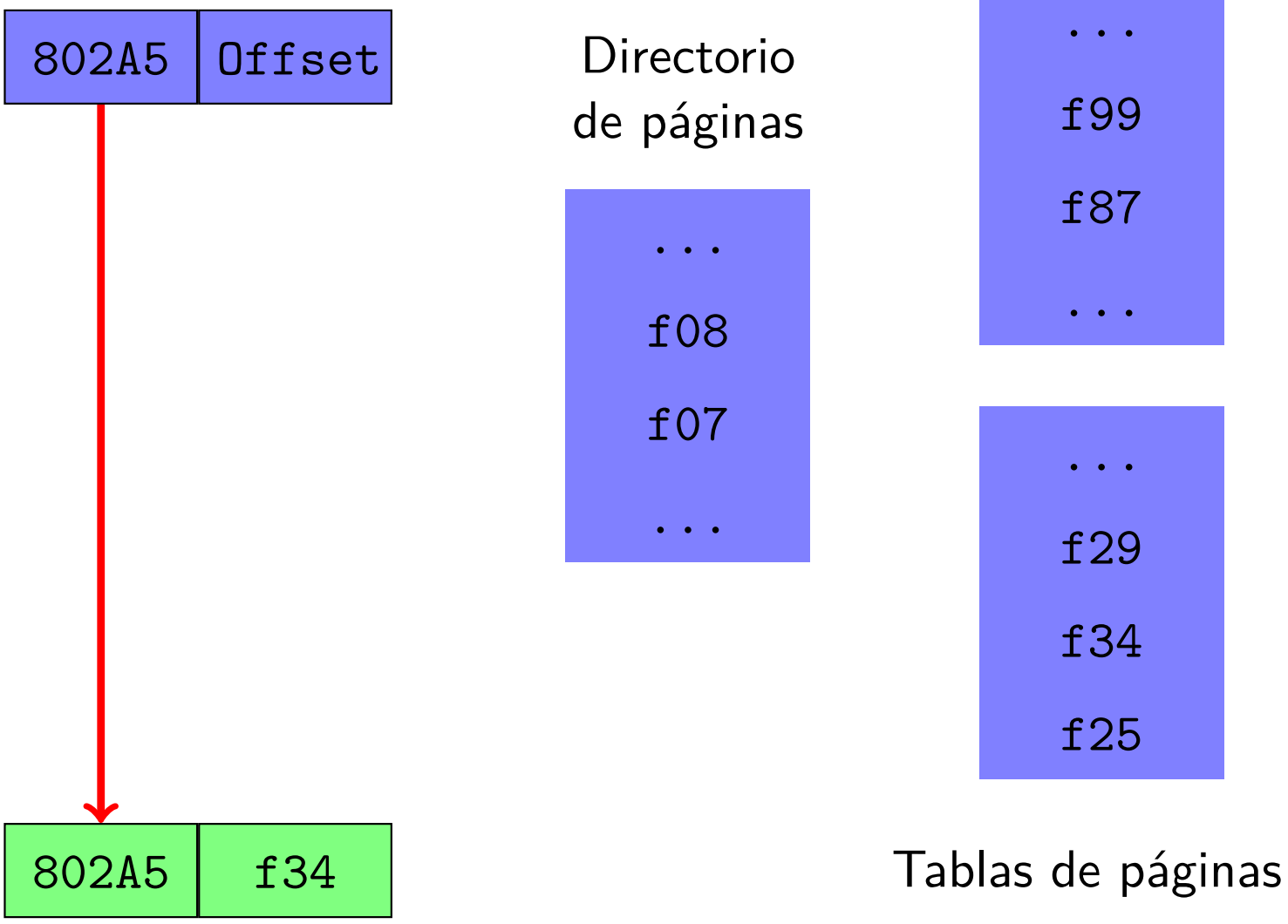
Nuestro TLB


P1
P2
Offset
Directorio
de páginas
...
f08
f07
...
...
f99
f87
...
...
f29
f34
f25
Tablas de páginas

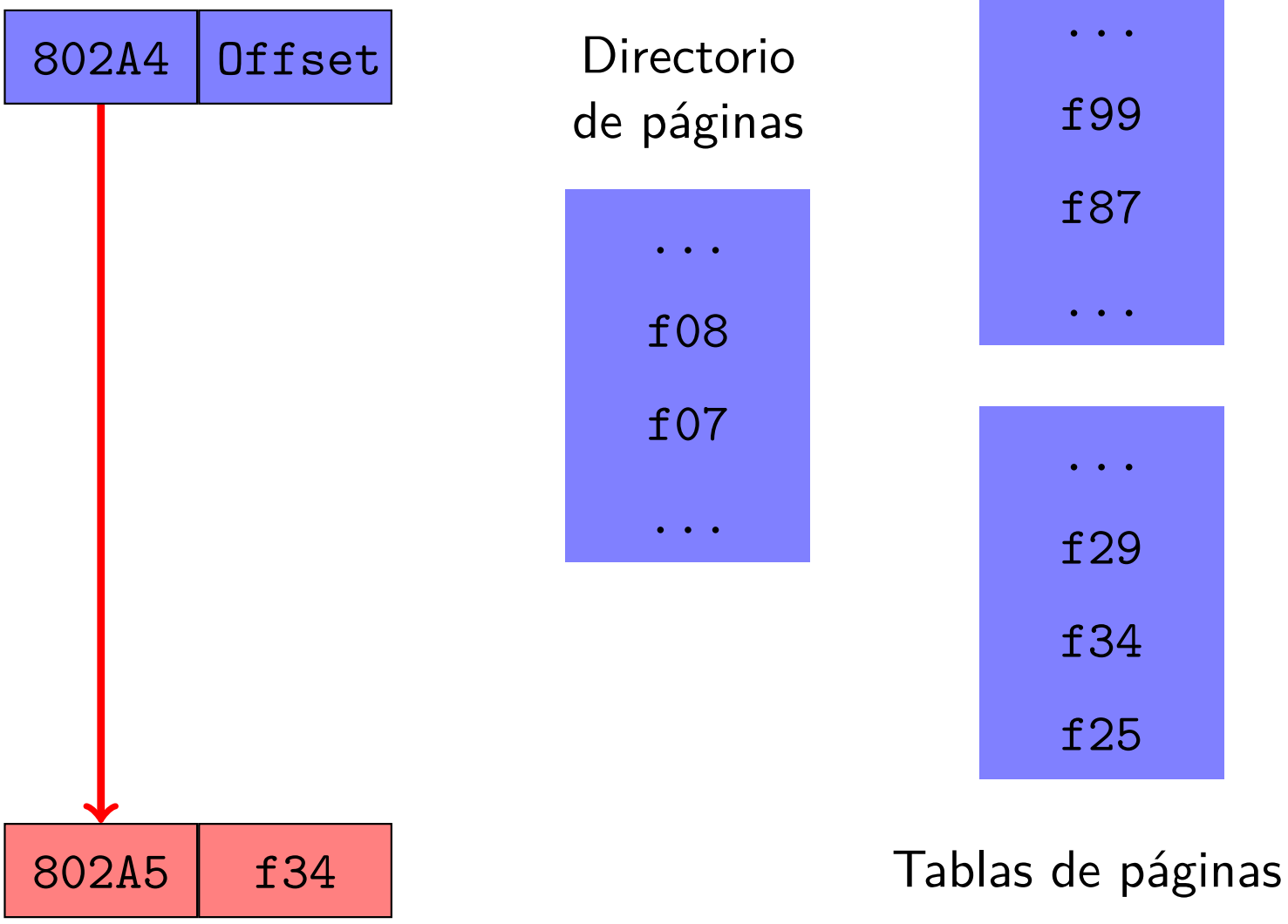
Nuestro TLB


P1
P2
Offset
Directorio
de páginas
...
f99
f87
...
...
f08
f07
...
...
f29
f34
f25
802A5
f34
Tablas de páginas

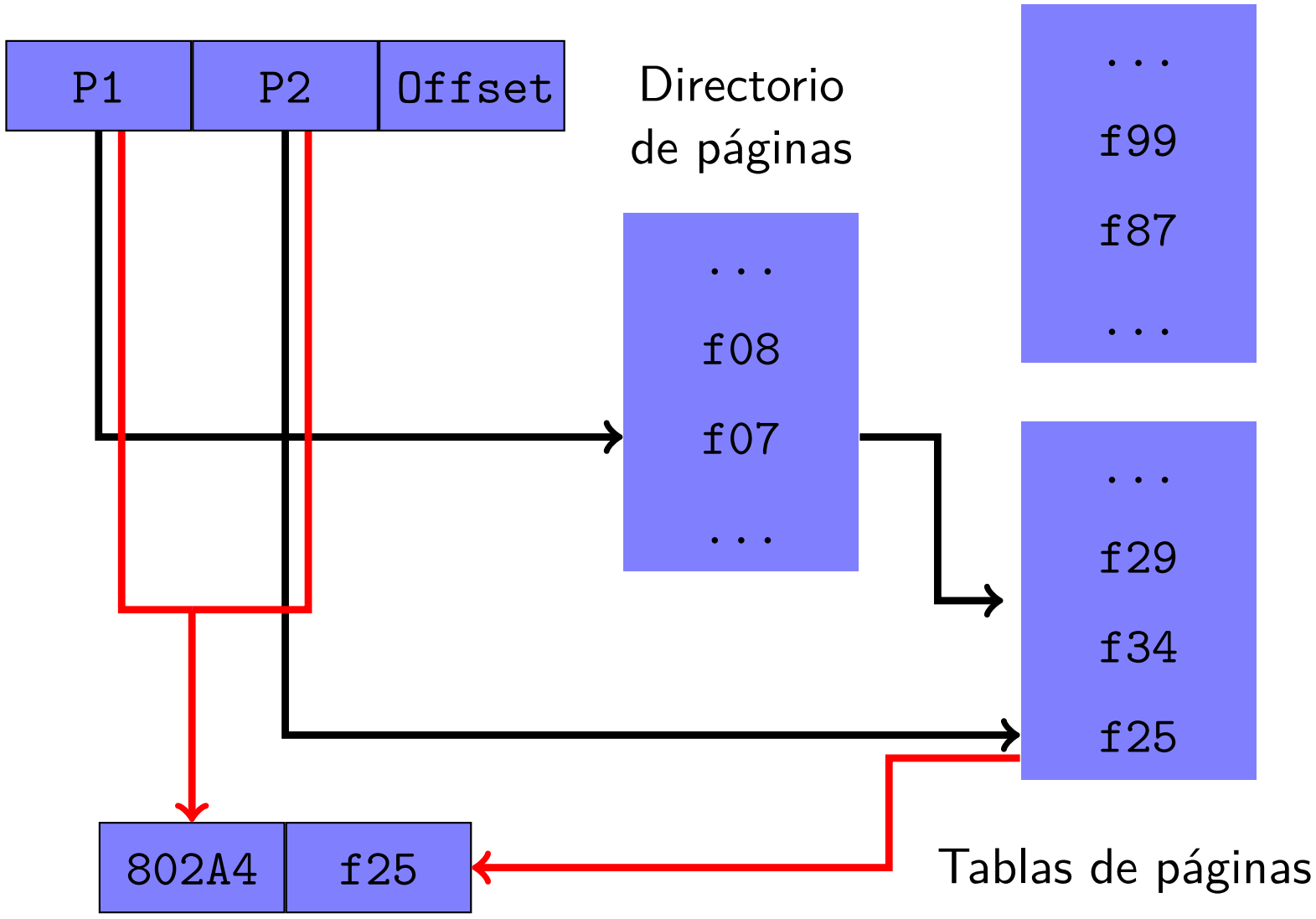
TLB hit


802A5
Offset
802A5
f34
Directorio
de páginas
...
f08
f07
...
...
f99
f87
...
...
f29
f34
f25
Tablas de páginas

TLB miss


802A4
Offset
802A5
f34
Directorio
de páginas
...
f08
f07
...
...
f99
f87
...
...
f29
f34
f25
Tablas de páginas

TLB refill


P1
P2
Offset
Directorio de páginas
...
f08
f07
...
...
f99
f87
...
...
f29
f34
f25
802A4
f25
Tablas de páginas

Instrucciones problemáticas

- ¿Pueden pensar en alguna instrucción patológica?
  - ¿Qué necesitamos para romper un TLB de una entrada?
- ¿Cuántos entradas en el TLB necesitamos?

TLB vs. cambio de contexto

- Después de estar ejecutando por un buen rato
  - el TLB está en caliente, i.e., lleno de páginas y traducciones a cuadros
- Interrupciones
  - Algún dispositivo terminó
  - Cambiamos a otra tarea
  - ¿Cuáles eran las partes de un cambio de contexto?
    - Registros generales
    - Registro base de la tabla de página
    - Contenidos (todos) del TLB

x86 TLB flush

- Declarar un nuevo directorio de páginas (establecer `%cr3`)
  - Limpia cada entrada en el TLB
  - Cuidado! No limpia las páginas globales
  - ¿Qué páginas deben de ser globales?
- Instrucción `INVLPG`
  - Invalida la entrada al TLB de una página específica
  - ¿Es eso más o menos eficiente?

x86 teoría de tipos

Versión final

- Instrucción ⇒ selector de segmento
  - `pushl` especifica el selector en `%ss`
- Proceso ⇒ (selector ⇒ (base, límite))
  - Tablas de descriptores globales y locales
- Base del segmento, dirección ⇒ dirección lineal
- TLB: dirección lineal ⇒ dirección física, o
- Proceso ⇒ (dirección lineal alta ⇒ tabla de página)
- Tabla de página: dirección lineal media ⇒ dirección del cuadro
- Memoria: dirección del cuadro, offset ⇒ datos!

¿Existe otra forma?

- El cálculo de la memoria es complicado
  - ¿Es esa implementación de hardware la solución óptima para cualquier SO y programa?
  - ¿Es la única solución no jugar?
- ¿Existe otra forma de resolver el problema?
  - ¿Podríamos no tener tablas de páginas?
  - ¿Cómo podría el hardware mapear de memoria virtual a física?

TLB cargados por software

- Razonamiento
  - Necesitamos un TLB por razones de rendimiento
  - El sistema operativo define cada estructura de memoria de los distintos procesos
    - Que regiones de memoria, permisos
    - Muchos procesos comparten los cuadros de `/bin/bash`
  - La unidad de hardware de mapeo de páginas impone sus propias ideas
  - ¿Por qué imponer semántica al medio?
- Enfoque
  - El sistema operativo conoce todos los mapeos de un espacio de direcciones
  - TLB contiene un subconjunto de ellos
  - La falla del TLB genera una excepción
  - El sistema operativo encuentra rápidamente el mapeo $v \Rightarrow p$

Características de TLB por software

- Mapear entradas puede ocurrir en muchas formas
  - Imaginemos un sistema con un tamaño de memoria de procesos
  - La falla del TLB se convierte en un problema aritmético
- Mapear entradas puede bloquear (*lock*) el TLB
  - Buena idea de obtener el seguro de una entrada del manejador del TLB
  - Buena idea para sistemas en tiempo real
- Lecturas futuras
  - `http://yarchive.net/comp/software_tlb.html`
- TLB por software
  - PowerPC 603, 400-series (pero no 7xx/9xx, Cell)
  - MIPS, algunas SPARC

Residencia parcial en memoria

- Código de manipulación de errores no es utilizado en cada ejecución
  - No hay necesidad de ocupar la memoria por toda la duración de la ejecución
- Las tablas alojadas pueden ser más grandes que las utilizadas
- La computadora puede ejecutar programas muy grandes
  - Más grandes que la memoria física
  - Mientras la huella activa quepa en la RAM
  - *Swapping* no puede manejar esto
- Los programas pueden empezar más rápido
  - No necesitamos cargar todo el programa antes de ejecutarlo

Solución de la memoria virtual

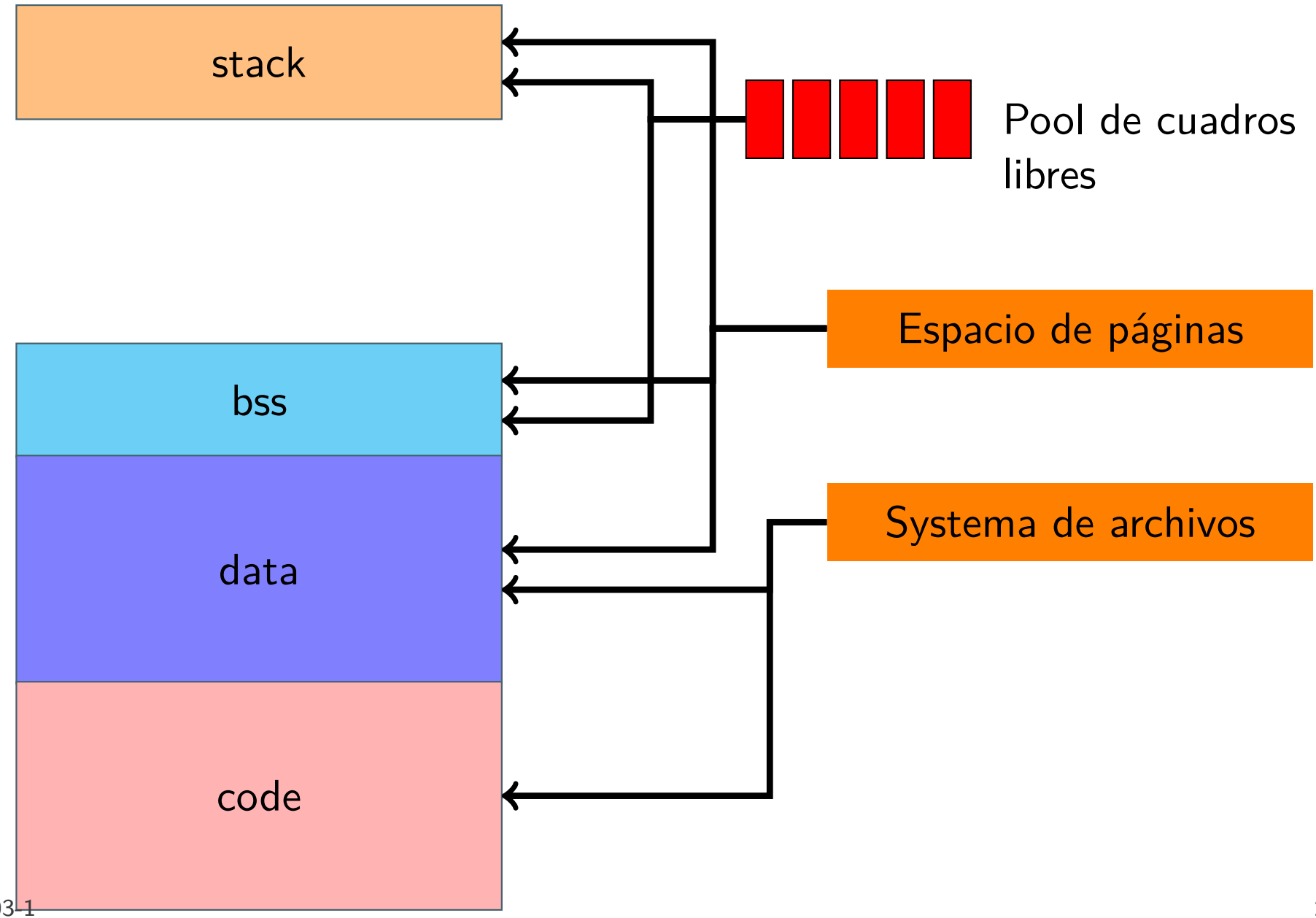
- Utilizar cuadros de RAM como cache para el conjunto de todas las páginas
  - Algunas páginas son rápidas de acceder (en un cuadro de RAM)
  - Algunas páginas son lentas de acceder (en un cuadro en memoria secundaria)
- Tablas de páginas indican que páginas son residentes
  - Páginas no residentes producen fallos en el TLB
    - Y tenemos `present=0` en la PTE, si tenemos una unidad de hardware de mapeo de páginas
  - Acceder a una página no residente genera un error de página (*page fault*)
    - El hardware invoca al manejador de excepciones de errores de página
- Esperamos que la mayoría de referencias estén dentro del cache en RAM

Errores de páginas

Razones y respuestas

- La dirección es inválida o ilegal: debe entregar una excepción de software
  - Unix: `SIGSEGV`
  - Mach: entrega el mensaje al puerto de excepciones del hilo
  - Dependiendo del SO debemos encontrar el manejador
- El proceso hace crecer su stack —tenemos que darle un nuevo cuadro
- El cache falla, entonces debemos buscar en el disco
  - ¿En que parte del disco exactamente?

Satisfacer errores de página



Inicio

- Un proceso realiza una referencia a la memoria
  - TLB falla
  - (PT: no está presente)
- Sorpresa! nos vamos al kernel
  - El procesador coloca el cuadro de la excepción en el stack (x86)
  - Transfiere a través de la interrupción de error de página DTE
  - Ejecuta el manejador de la excepción

En el manejador

- Clasificamos la dirección de error
  - Dirección ilegal $\Rightarrow$ entregamos un error, sino
- Revisamos la región de `código` o `rodata` del ejecutable
  - Determinamos que sector del archivo ejecutable
  - Ejecutamos `read()` del archivo a un cuadro que no esté utilizado
- Si hay un residente hacemos `r/w` data, y sacamos la página
  - En algún lugar de la partición de las páginas
  - Encolamos la lectura a disco en un cuadro libre
- Si es el primer uso de la página `bss/stack`
  - Alojamos un cuadro lleno de ceros
  - Insertamos en el PT

Esperando al disco

- Bloqueamos al proceso (como en casos anteriores)
  - Cambiamos la ejecución a otro proceso
- Manejamos la interrupción de I/O
  - Llenamos la PTE (`present=1`)
  - Marcamos el proceso como ejecutable
- Restauramos los registros, cambiamos la tabla de páginas
  - La instrucción que dió el error se reiniciar de manera transparente
  - Una instrucción puede dar más de un error de página!

Regiones de memoria vs. tablas de páginas

- ¿Qué es lo que debe de hacer el manejador de errores de páginas?
  - ¿Matar el proceso?
  - ¿Copiar la página? ¿Marcar como lectura/escritura?
  - ¿Obtener la página desde un archivo? ¿Cuál? ¿Dónde?
- La tabla de páginas no es una buena estructura de datos
  - El formato está definido por hardware
  - La definición por página es repetitiva
  - No hay suficientes bits para codificar la metadata del sistema operativo
    - La dirección de los sectores del disco pueden tener más de 32 bits

Modelo de memoria dual

- Lógica
  - La memoria de un proceso es una lista de regiones
  - Los agujeros entre las regiones son direcciones ilegales
  - Métodos por región
    - `fault()`, `evict()`, `unmap()`
- Física
  - La memoria de un proceso es una lista de páginas
  - Los errores son delegados a métodos por región
  - Muchas páginas inválidas pueden hacerse válidas
    - Pero algunas veces un manejador de errores de una región retornará un error
    - Lo manejamos como el error en el caso de agujeros

Error de páginas

- Examinar la dirección de error
- Buscar: dirección $\Rightarrow$ región
- `region->fault(addr, access_mode)`
  - Rápidamente resuelve el problema
  - O empezar una forma de arreglarlo, bloquear el proceso, ejecutar el calendarizador

Resumen

- TLB: no hay más misterio
- Espacio de direcciones de un proceso
  - Lógicas: lista de regiones
  - Hardware: lista de páginas
- Manejador de errores es complejo
  - Carga bajo demanda del archivo, página del área de páginas, etc.