

Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación



# IIC2343 – Arquitectura de Computadores

## Multiprogramación

**Profesor:** Hans-Albert Löbel

# ¿Qué es multiprogramación?

- Es una forma básica de procesamiento paralelo, utilizando un **único procesador**.
- Al ser sólo un único procesador, el paralelismo generado **no es 100% real**.
- Se basa en la **ejecución intercalada** de los programas, en intervalos de tiempo muy pequeños.
- Para el usuario, este esquema de ejecución es **percibido como paralelismo real**.

# ¿Qué contiene/define un programa?

1. Memoria:  
Código, variables, stack

Buscamos que hayan múltiples programas en memoria.

2. Estado de procesamiento (CPU):  
PC, SP, registros de usuario, flags

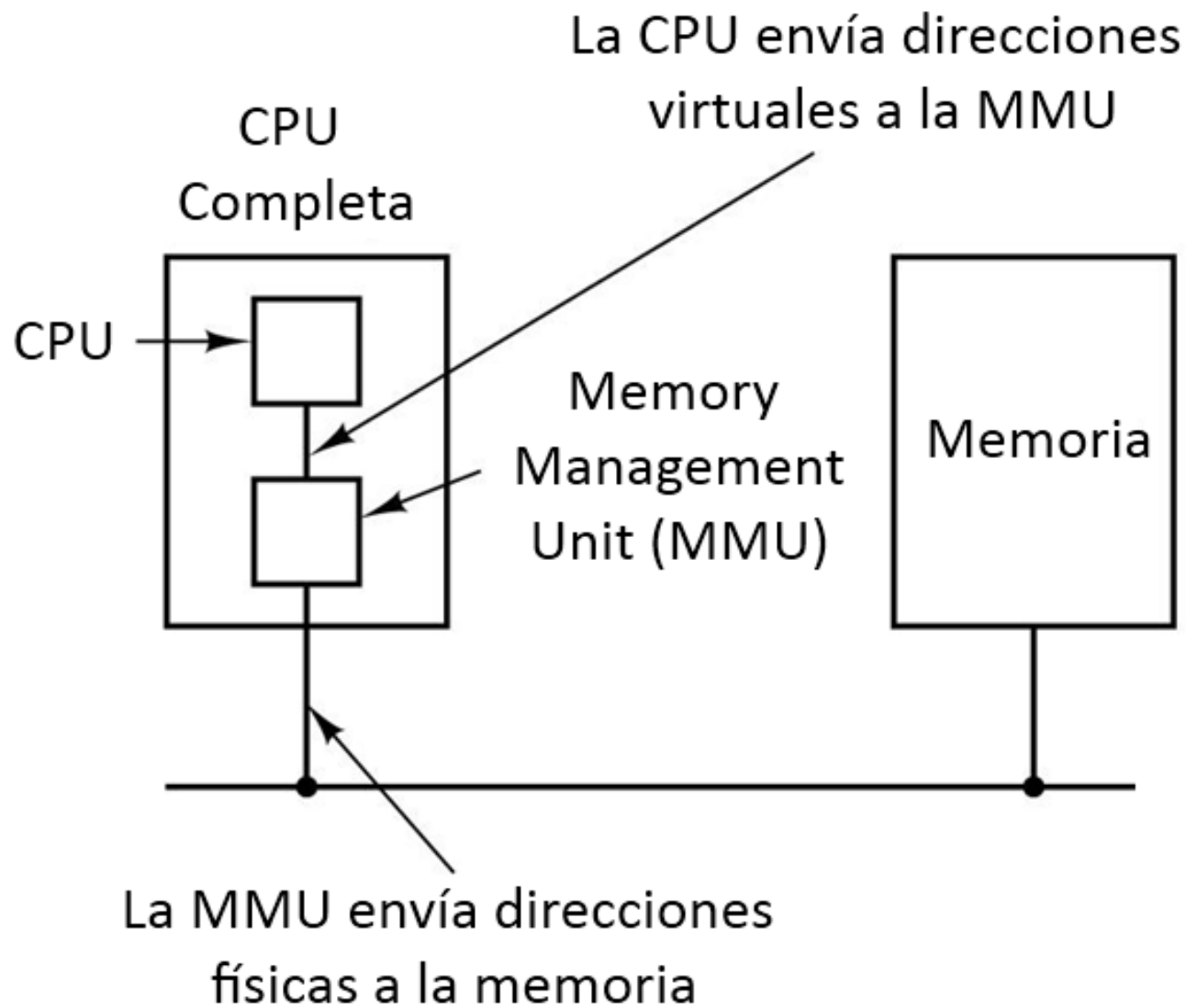
Necesitamos que la CPU maneje múltiples estados.

# ¿Como darle memoria a muchos programas distintos?

- Simple: Una parte de la memoria a cada programa
- Problemas:
  1. El programador tiene que saber a priori el espacio de memoria correspondiente
  2. No hay protección: es posible escribir en los datos de otro programa
  3. Tamaño de memoria fijo para cada programa

# Memoria Virtual soluciona todos los problemas anteriores

- Cada programa trabaja sobre un **espacio virtual de memoria**, equivalente al **espacio direccionable completo**.
- Un "**intermediario**" transforma las direcciones de ese espacio virtual de memoria a direcciones físicas.

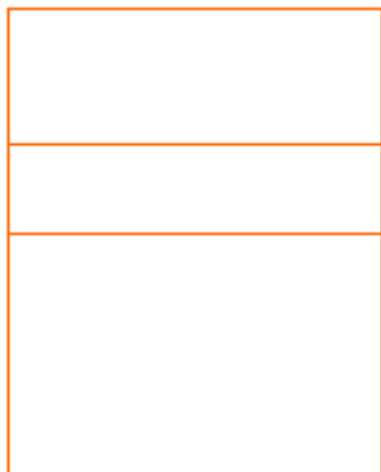


## Memoria virtual

## Memoria física

P1

100



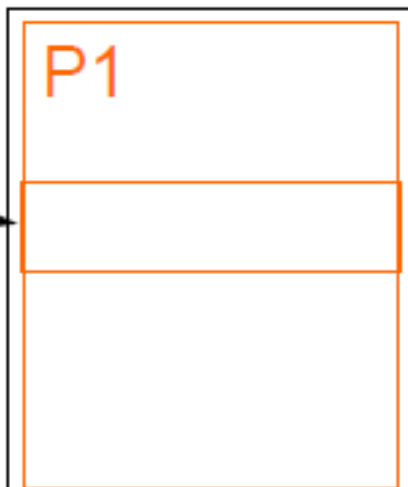
P2

100



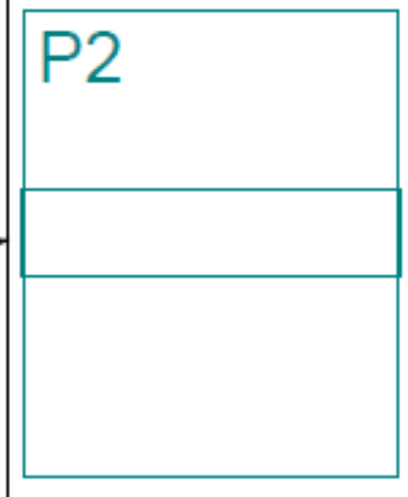
P1

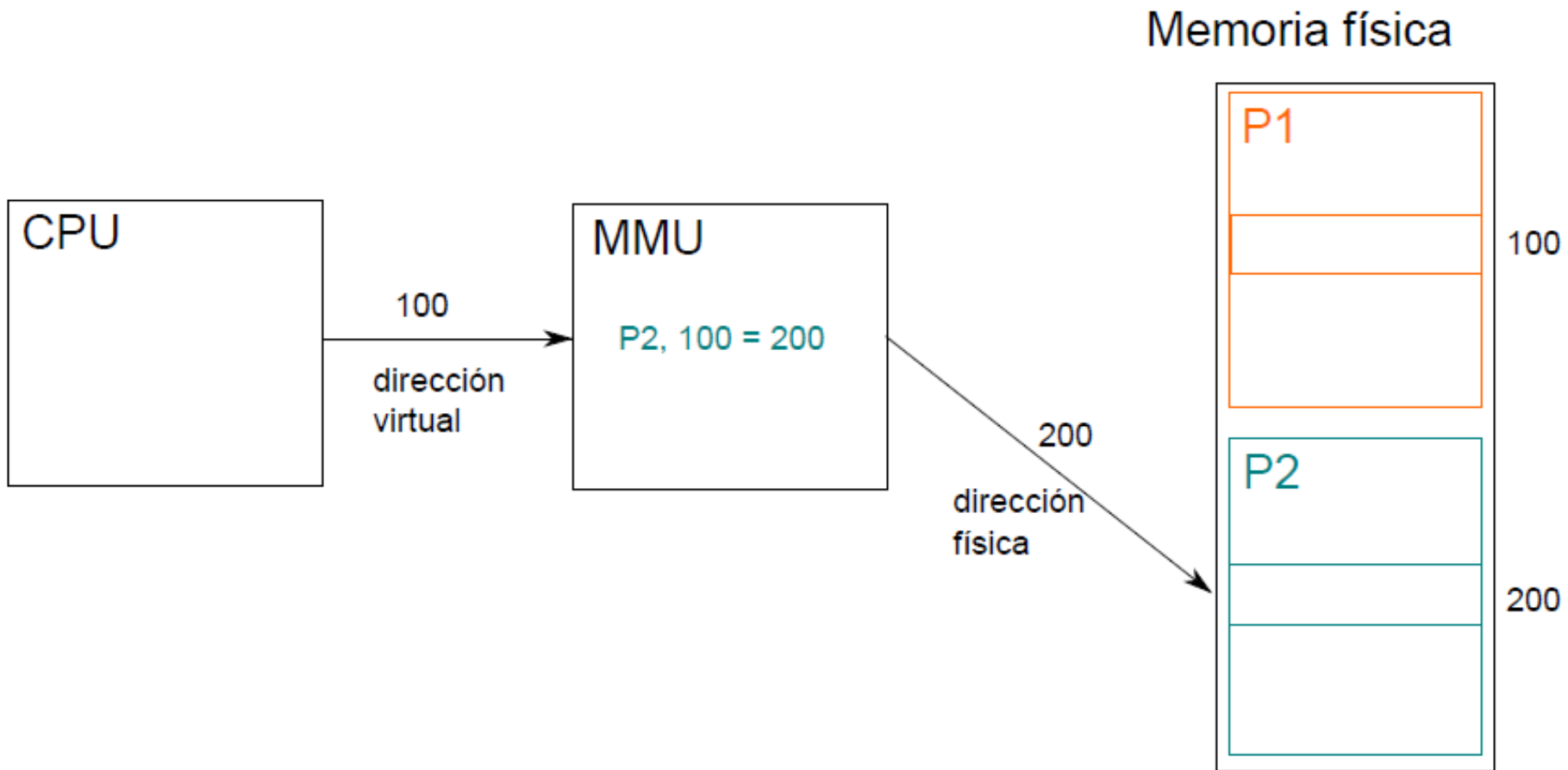
100



P2

200







# ¿Cómo se pueden mapear direcciones virtuales a físicas?

- Se puede utilizar **una tabla por cada programa**, donde se indica el mapeo de cada dirección
- Estas tablas pueden guardarse en la memoria principal
- ¿Problema?  
Una tabla podría tener el mismo tamaño que la memoria física
- Se introduce el concepto de **paginación** para evitar este problema

## Páginas de memoria son parecidas al concepto de bloque

- Una **página** es conjunto de palabras de memoria virtual
- Un **marco** es el símil de una página, pero en memoria física
- Ahora se almacenan **tablas de páginas**
- Tamaños habituales de página: 1KB
- Dada una dirección: los **n** bits más significativos definen el número de página, el resto la ubicación dentro de la página (offset)

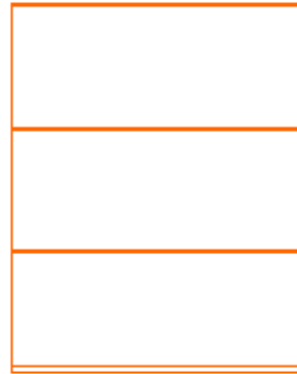
Página virtual	Marco físico
0	2
1	3
2	4

Página virtual	Marco físico
0	5
1	6
2	7

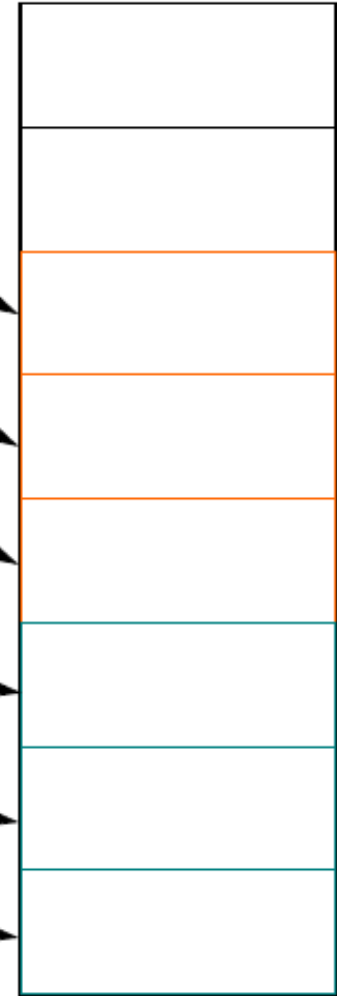
Memoria virtual

Memoria física

P1



P2



Página virtual	Marco físico	Validez
0	2	1
1	3	1
2	4	1
3	x	0
4	x	0
5	x	0
6	x	0
7	x	0

# El tiempo de acceso a las tablas de paginación es fundamental

- ¿Cuántos accesos a memoria son necesarios para obtener un dato?
- Para mejorar el rendimiento y evitar este doble acceso a memoria, se agrega una caché dedicada a almacenar entradas de tabla de página.
- Esta caché es conocida como *Translation Lookaside Buffer* (TLB)

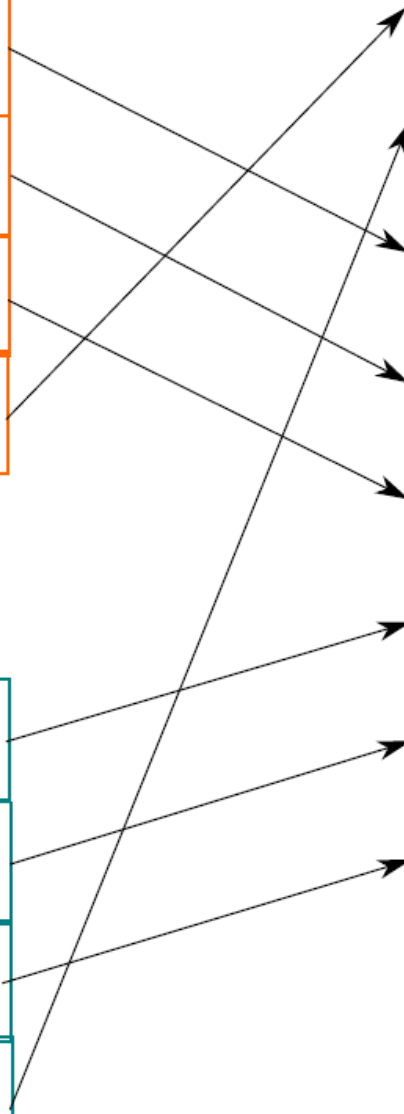
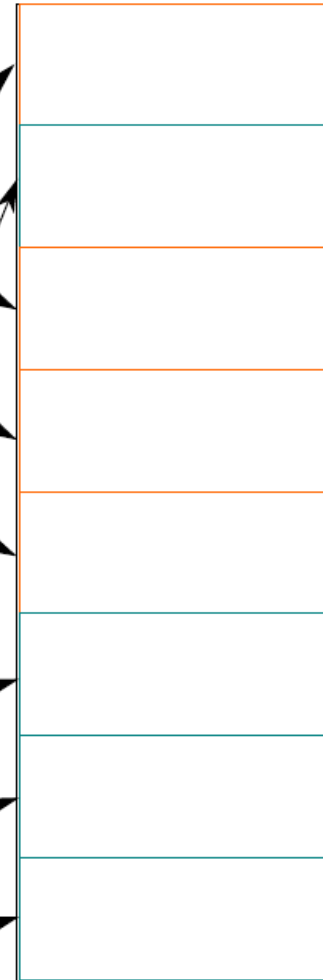
Memoria virtual

Memoria física

P1



P2



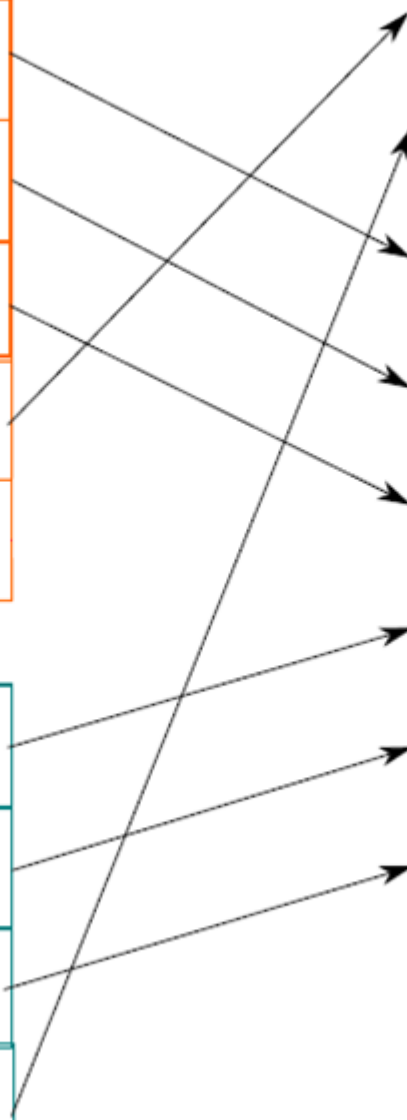
Memoria virtual

Memoria física

P1



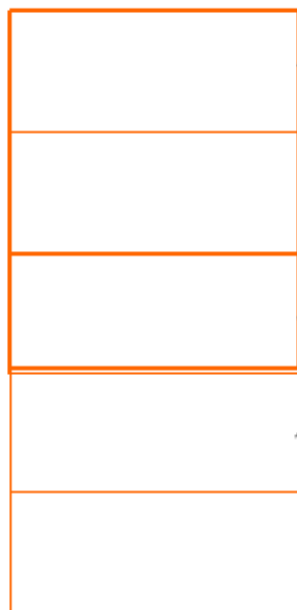
P2



Memoria virtual

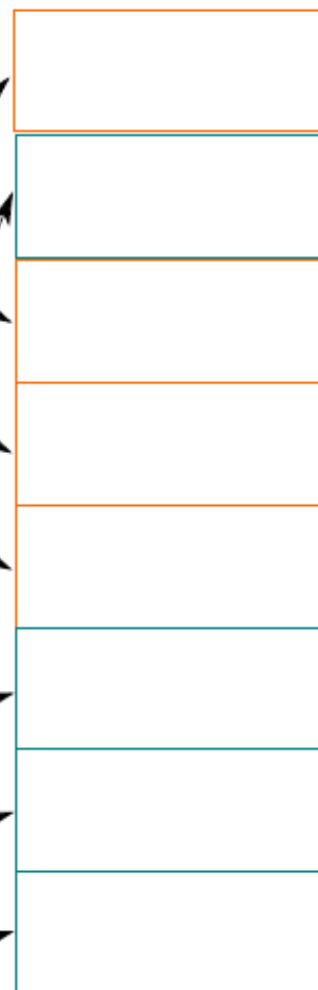
Memoria física

P1

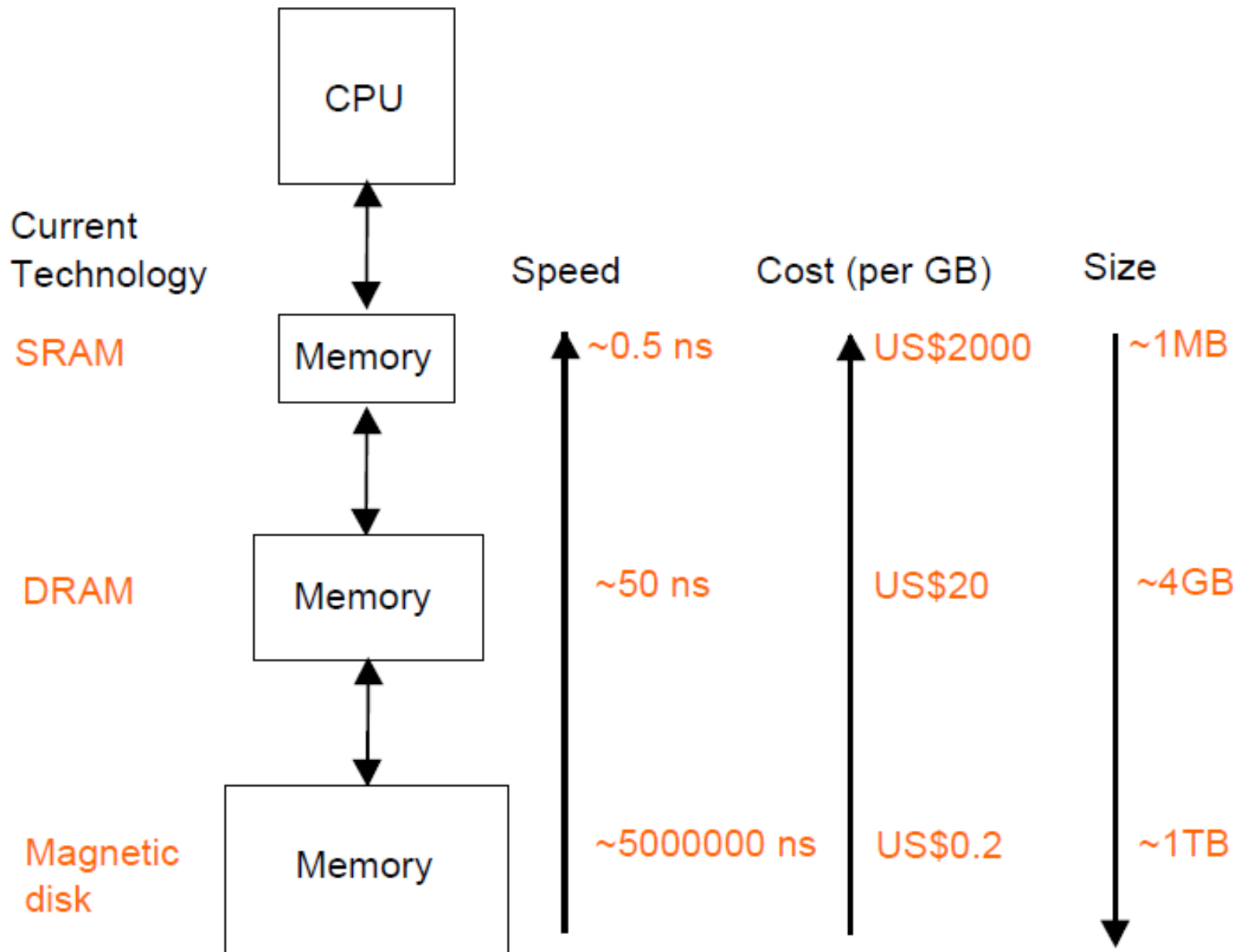


page fault

P2





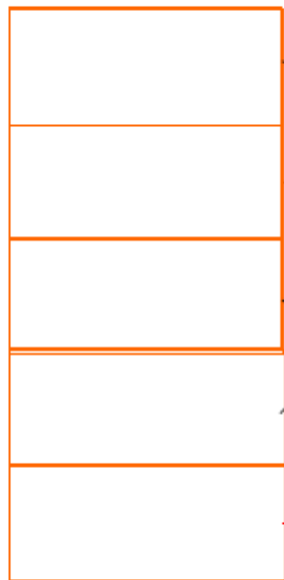


Memoria virtual

Memoria física

Disco: Swap File

P1



page fault

P2



swap out

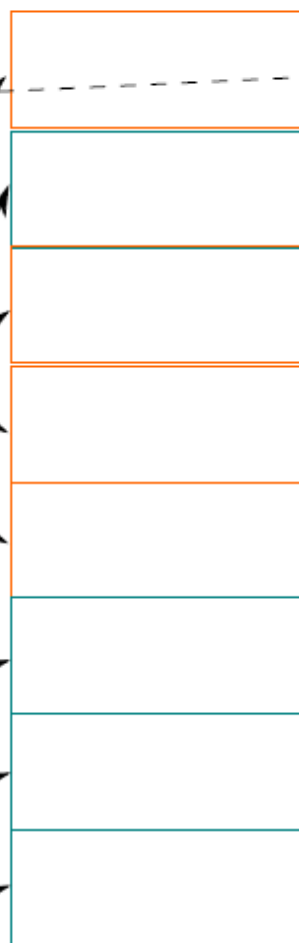
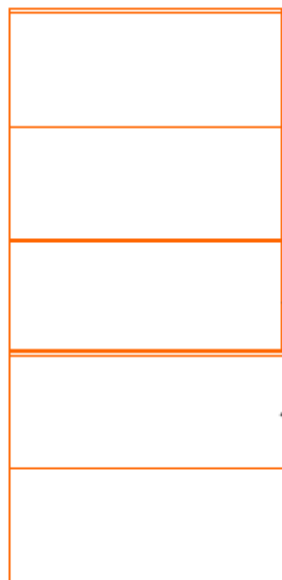


Memoria virtual

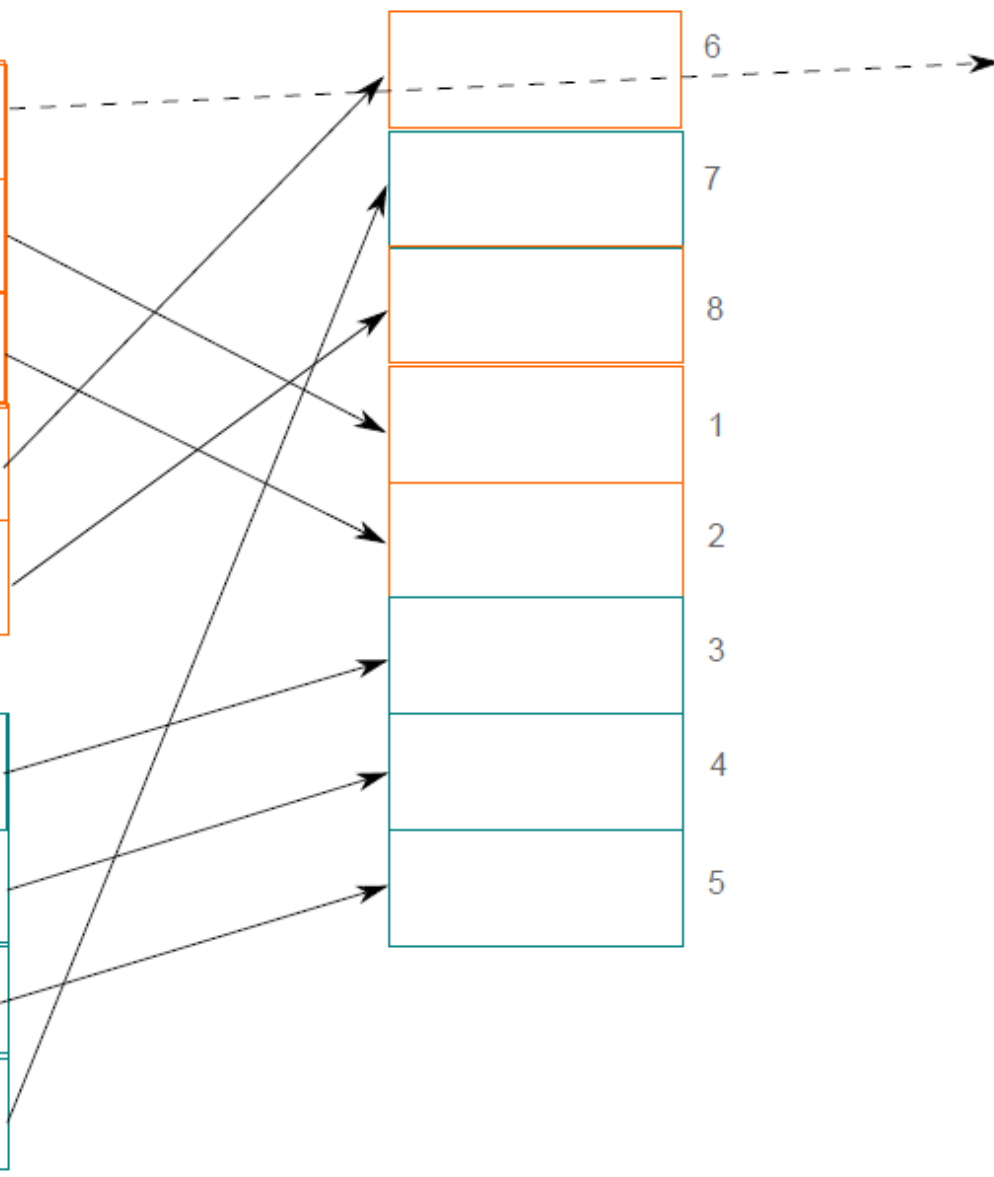
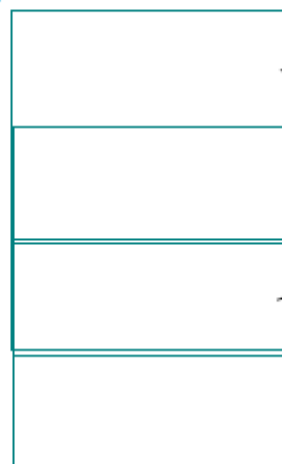
Memoria física

Disco: Swap File

P1



P2

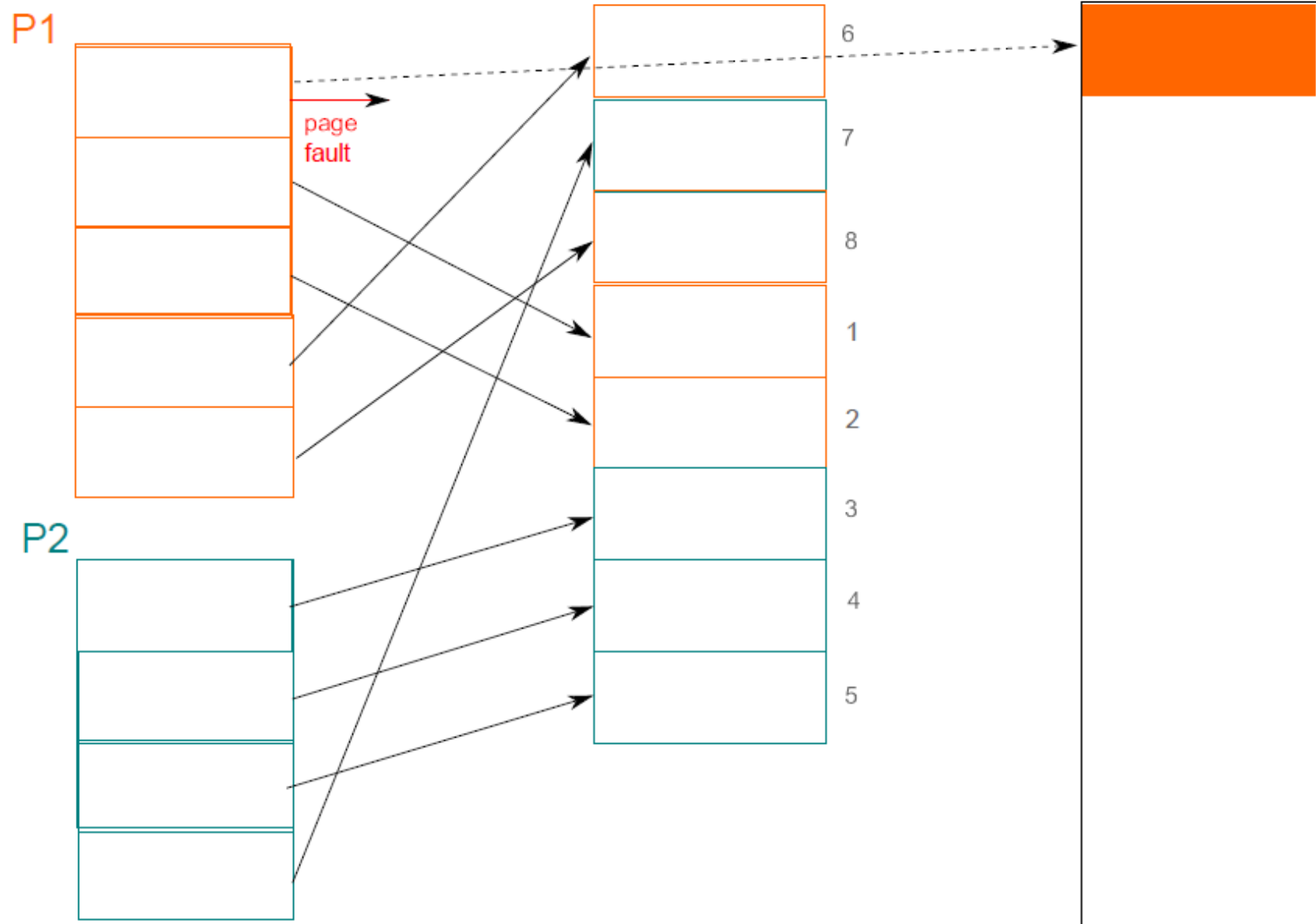


Página virtual	Marco físico	Validez	Disco
0	2	1	1
1	3	1	0
2	4	1	0
3	0	1	0
4	2	1	0
5	x	0	0
6	x	0	0
7	x	0	0

## Memoria virtual

## Memoria física

## Disco: Swap File

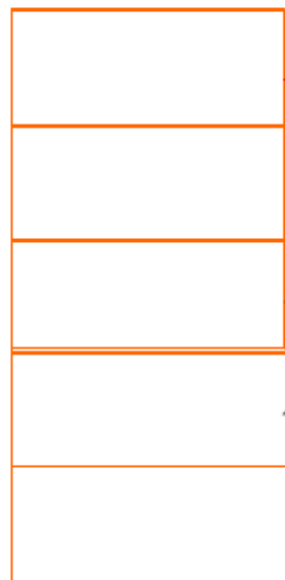


## Memoria virtual

## Memoria física

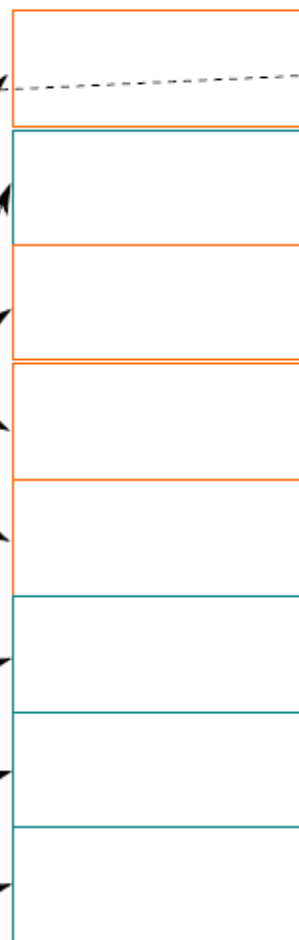
## Disco: Swap File

P1



page  
fault

P2



6

7

8

1

2

3

4

5

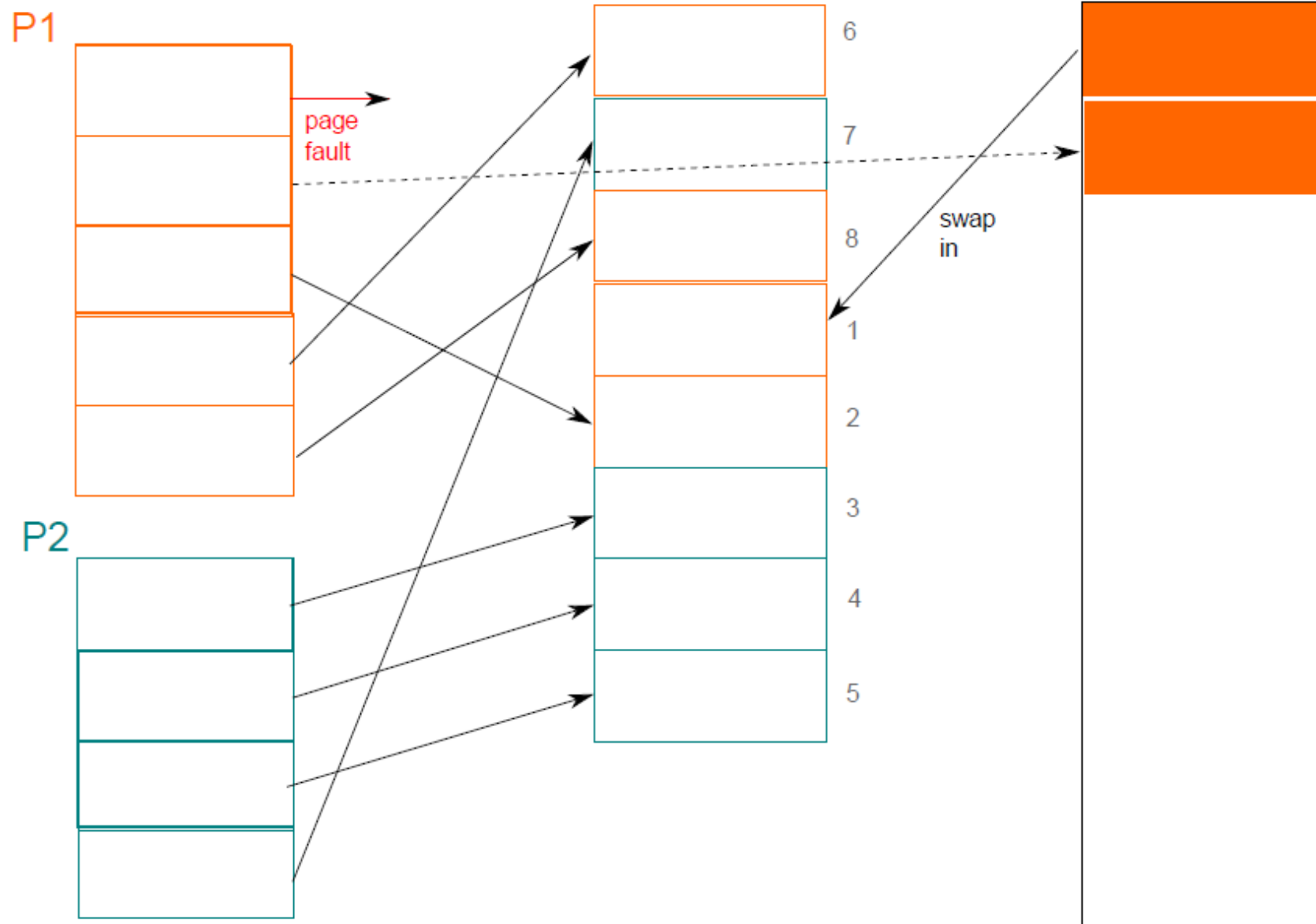
swap  
out



## Memoria virtual

## Memoria física

## Disco: Swap File

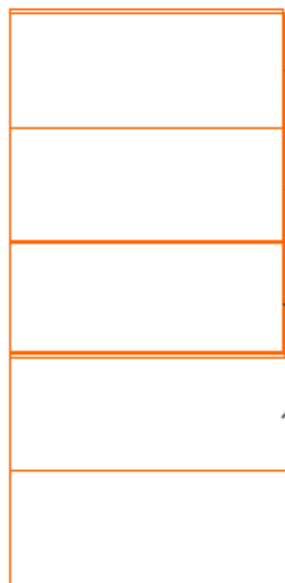


Memoria virtual

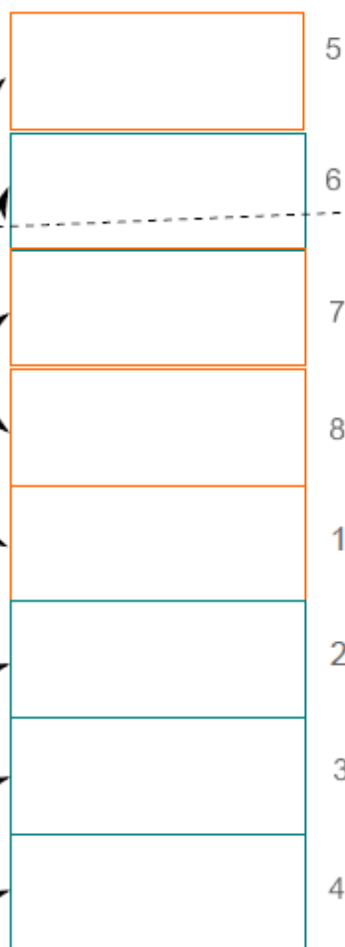
Memoria física

Disco: Swap File

P1



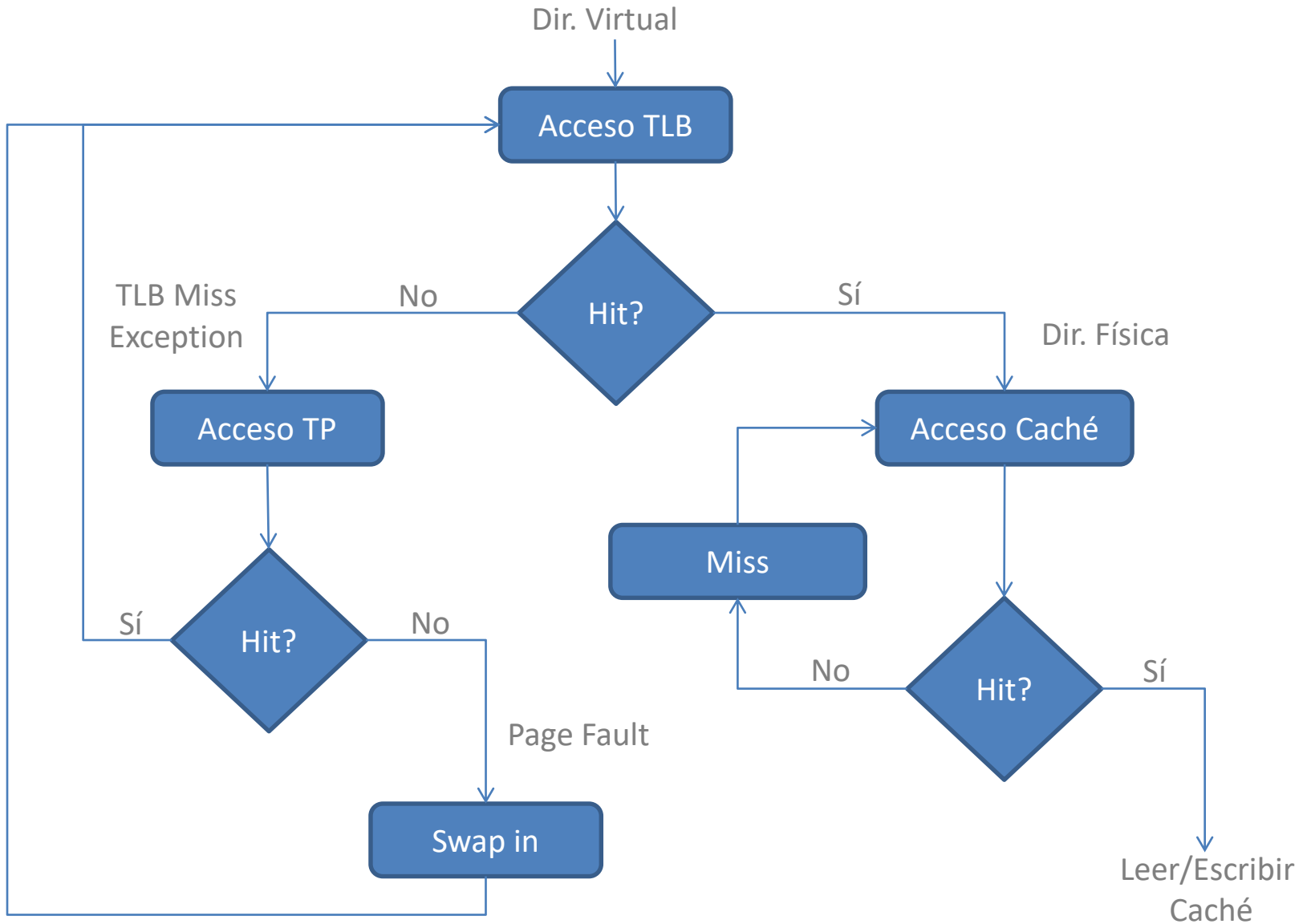
P2





Página virtual	Marco físico	Validez	Disco
0	3	1	0
1	3	1	1
2	4	1	0
3	0	1	0
4	2	1	0
5	x	0	0
6	x	0	0
7	x	0	0

# Memoria Virtual + Caché



# ¿Qué contiene un programa?

1. Memoria: código, datos, stack

Buscamos que hayan múltiples programas en memoria

2. Estado de procesamiento (CPU): PC, SP, registros, flags

Necesitamos que la CPU maneje múltiples estados

# Programa vs Proceso

- Al código ejecutable almacenado, sin ejecutar, se le denomina **programa**.
- A un programa en ejecución se le denomina **proceso**.
- Luego, un **proceso** comprende, además del código, de los **datos en memoria** y del **estado de procesamiento**.

# ¿Como ejecutar más de un proceso en una CPU?

- Simple: Se ejecuta uno a continuación del otro, utilizando al SO como intermediario (**Batch Processing**).
- Control es cedido al SO mediante un *supervisor call*.



# ¿Qué diferencia al **SO** de los otros procesos?

- SO debe realizar una serie de **tareas especiales** que **ningún otro proceso debiese poder realizar**.
- Una de las más importantes es manejar y coordinar **múltiples procesos**, lo que implica cómo mínimo múltiples tablas de páginas.
- Necesitamos ayuda del **hardware** para facilitar este proceso.

# Sistema operativo es un proceso con privilegios especiales

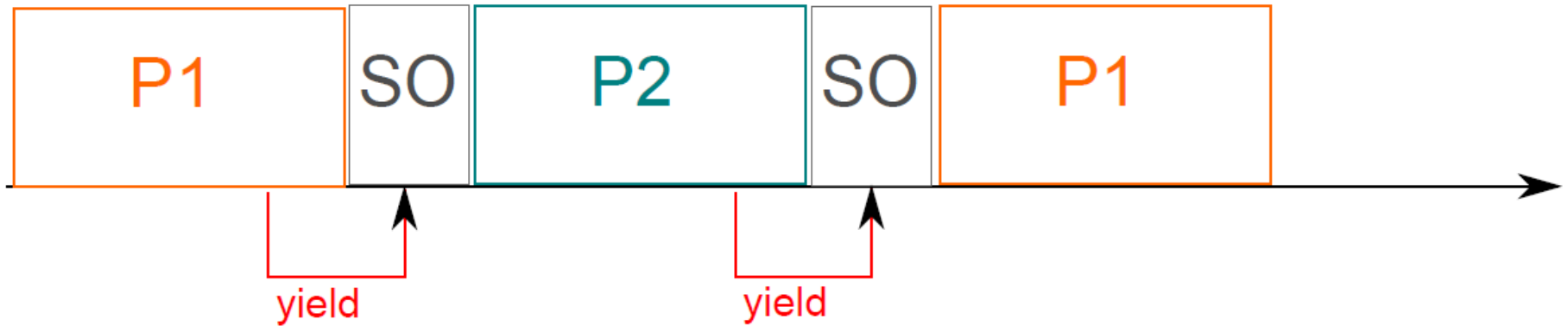
- Se agrega a la CPU un **bit de modo** en el registro status: supervisor o usuario.
- Para usar múltiples tablas de páginas, se agrega a la CPU el *Page Table Base Register* (PTBR).
- **PTBR** indica el inicio de la tabla de páginas en uso y sólo se modifica en modo supervisor.
- Otros privilegios incluyen acceso a **dispositivos de I/O** y a zonas protegidas de memoria, como por ejemplo el vector de interrupciones.

# CPU Scheduling es parte central de los SOs modernos

- *Batch processing* no nos entrega la ilusión de paralelismo que vemos en los SOs actuales.
- Necesitamos que ejecución de procesos sea intercalada.
- Para lograr esto se introduce el concepto de **scheduling**, que consiste en **agendar** el uso de la CPU entre los distintos procesos.



Cooperative Scheduling permite  
múltiples procesos “simultáneos”



## Cooperative Scheduling permite múltiples procesos “simultáneos”

- Los procesos entregan voluntariamente el control de vuelta a la CPU (`yield`).
- CPU realiza un cambio de contexto, donde se respalda el estado del proceso y se restaura el del siguiente proceso que utilizará la CPU.

# PCB contiene toda la información de un proceso

SO tiene en memoria, para cada proceso, un **Process Control Block (PCB)**, donde se almacenan los datos respaldados.

pointer	process state
process number	
program counter	
registers	
memory limits	
list of open files	
⋮	

P1 envía *yield*; CPU pasa a modo supervisor y se carga el SO

SO guarda estado de P1 en PCB correspondiente

SO va a buscar P2 al disco

SO asigna marcos de memoria a las páginas de P2

SO modifica el PTBR para que apunte a la tabla de páginas de P2

SO limpia TLB

SO setea PC

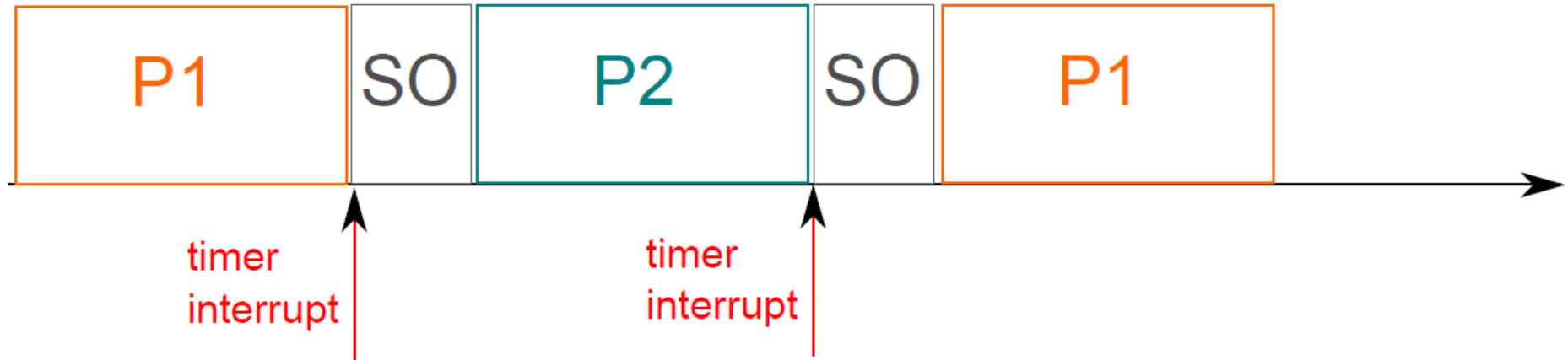
SO entrega el control a P2; CPU vuelve a modo usuario

P2 comienza a ejecutarse

## Cooperative Scheduling permite múltiples procesos “simultáneos”

- Cooperative Scheduling depende de “generosidad” de procesos para funcionar bien.
- ¿Cómo podemos evitar los problemas generados por los procesos “egoístas”?

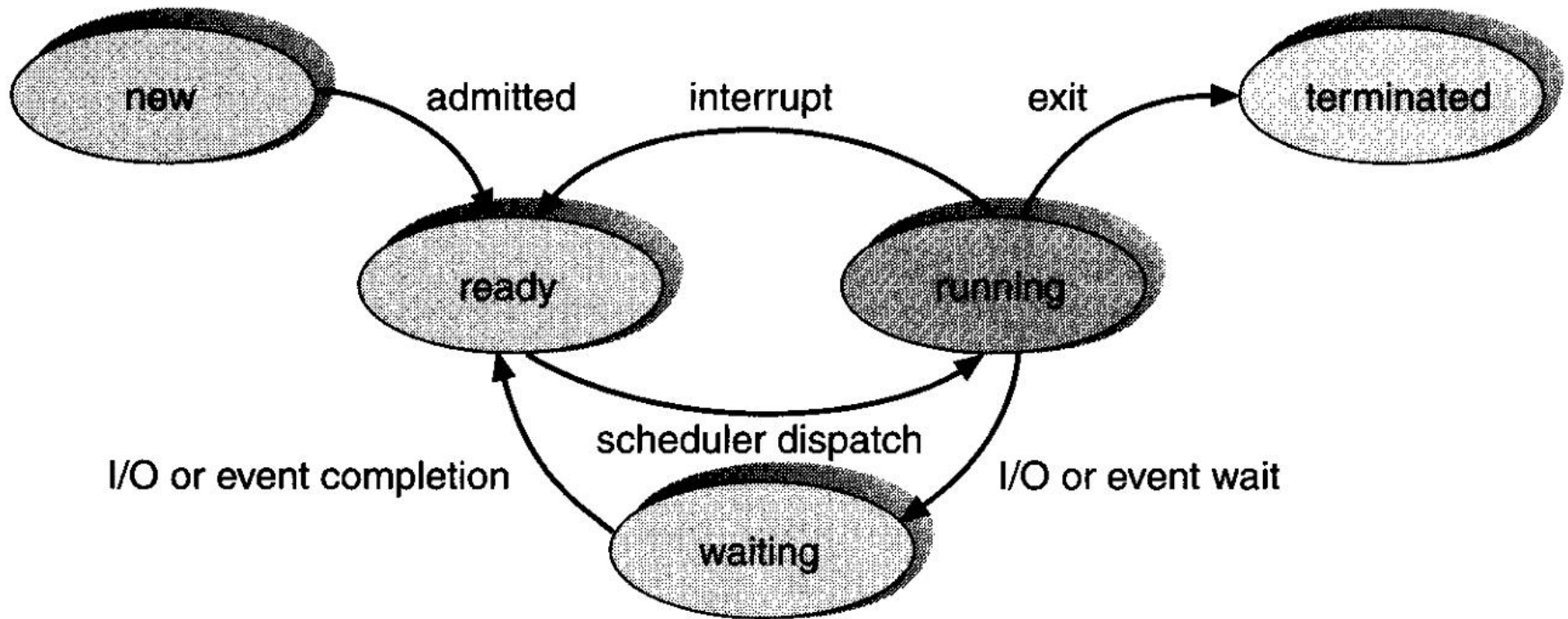
**Preemptive Scheduling** permite interrumpir un proceso y hacer un cambio de contexto



## Preemptive Scheduling permite interrumpir un proceso y hacer un cambio de contexto

- Se utiliza el **timer** del computador, que genera interrupciones periódicas.
- La ISR de esta interrupción es **controlada por el sistema operativo** y es donde se realiza el **cambio de contexto**.
- La manera de elegir el siguiente proceso debe tomar en cuenta uso de la CPU, de I/O, prioridades, etc.

## Diagrama de estados de un proceso entrega un visión más clara del preemptive scheduling





# Algoritmos de *scheduling* son esenciales para el rendimiento de los sistemas operativos

- Su principal propósito es minimizar la *inanición* de los procesos y de asegurar la *justicia* sobre el uso de los recursos.
- Algunos ejemplos:
  - First in, First out (simple)
  - Shortest Remaining Time (casi no se usa)
  - Fixed priority pre-emptive scheduling (colas de FIFO)
  - Round-robin scheduling
  - Multilevel queue scheduling (foreground-background)
  - Multilevel feedback-queue scheduling (Windows)

Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación



# IIC2343 – Arquitectura de Computadores

## Multiprogramación

**Profesor:** Hans-Albert Löbel