

Grupo ARCOS  
Departamento de Informática  
Universidad Carlos III de Madrid

# Lección 1 (b)

## Introducción

Diseño de Sistemas Operativos  
Grado en Ingeniería Informática y Doble Grado I.I. y A.D.E.



# Objetivos generales

1. Conocer **cómo es por dentro**.
  1. Ejecución asíncrona.  
Posibles estructuras internas.
2. Conocer **cómo es por fuera**.
  1. Librerías estáticas, dinámicas, módulos, ejecutables, etc.

# A recordar...

---

Antes de clase

Clase

Después de clase

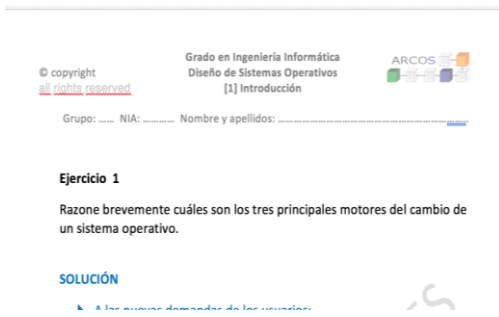

Preparar los pre-requisitos.

Estudiar el material asociado a la **bibliografía**:  
las transparencias solo no son suficiente.  
Preguntar dudas (especialmente tras estudio).

Ejercitar las competencias:

- ▶ Realizar todos los **ejercicios**.
- ▶ Realizar los **cuadernos de laboratorios** y las **prácticas** de forma progresiva.

# Ejercicios, cuadernos de prácticas y prácticas

Ejercicios ✓	Cuadernos de prácticas ✓	Prácticas ✗
 <p>© copyright <a href="#">all rights reserved</a></p> <p>Grado en Ingeniería Informática Diseño de Sistemas Operativos [1] Introducción</p> <p>ARCOS</p> <p>Grupo: ____ NIA: _____ Nombre y apellidos: _____</p> <p><b>Ejercicio 1</b></p> <p>Razone brevemente cuáles son los tres principales motores del cambio de un sistema operativo.</p> <p><b>SOLUCIÓN</b></p> <p>▶ A las nuevas demandas de los usuarios</p>	 <p>DISEÑO DE SISTEMAS OPERATIVOS</p> <p>GRADO EN INGENIERÍA INFORMÁTICA DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y ADMINISTRACIÓN DE EMPRESAS</p> <p>uc3m   Universidad Carlos III de Madrid</p> <p><b>Compilación y depuración de software orientado a eventos: descubriendo el <u>nanokernel</u></b></p>	

# Lecturas recomendadas

---

## Base



1. Carretero 2007:
  1. Cap. 2

## Recomendada



1. Tanenbaum 2006:
  1. Cap. I
2. Stallings 2005:
  1. Parte uno. Transfondo.
3. Silberschatz 2006:
  1. Cap. I


# Contenidos

---

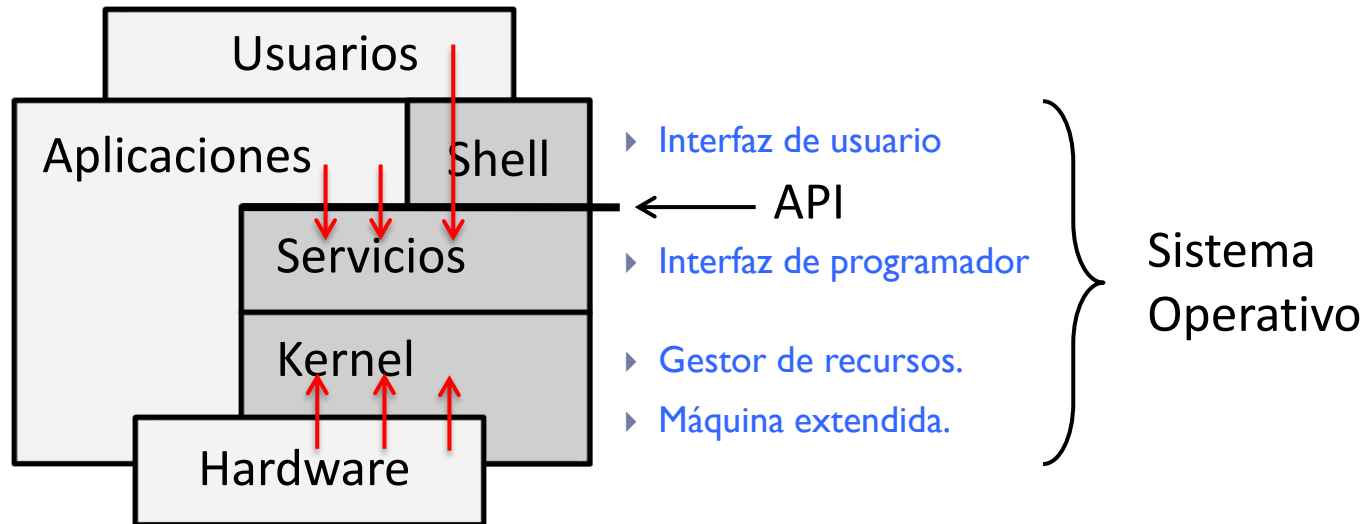
1. **Orientado a eventos**
  1. Ejecución asíncrona e interrupción
  
2. **Modular**
  1. Núcleo y módulos

# Contenidos

---

1. Orientado a eventos 
  1. **Ejecución asíncrona** e interrupción
2. Modular
  1. Núcleo y módulos

# Estructura del sistema operativo

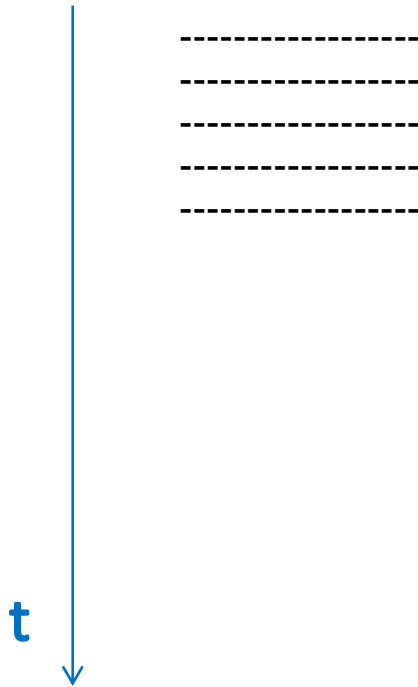




# Ejecución asíncrona

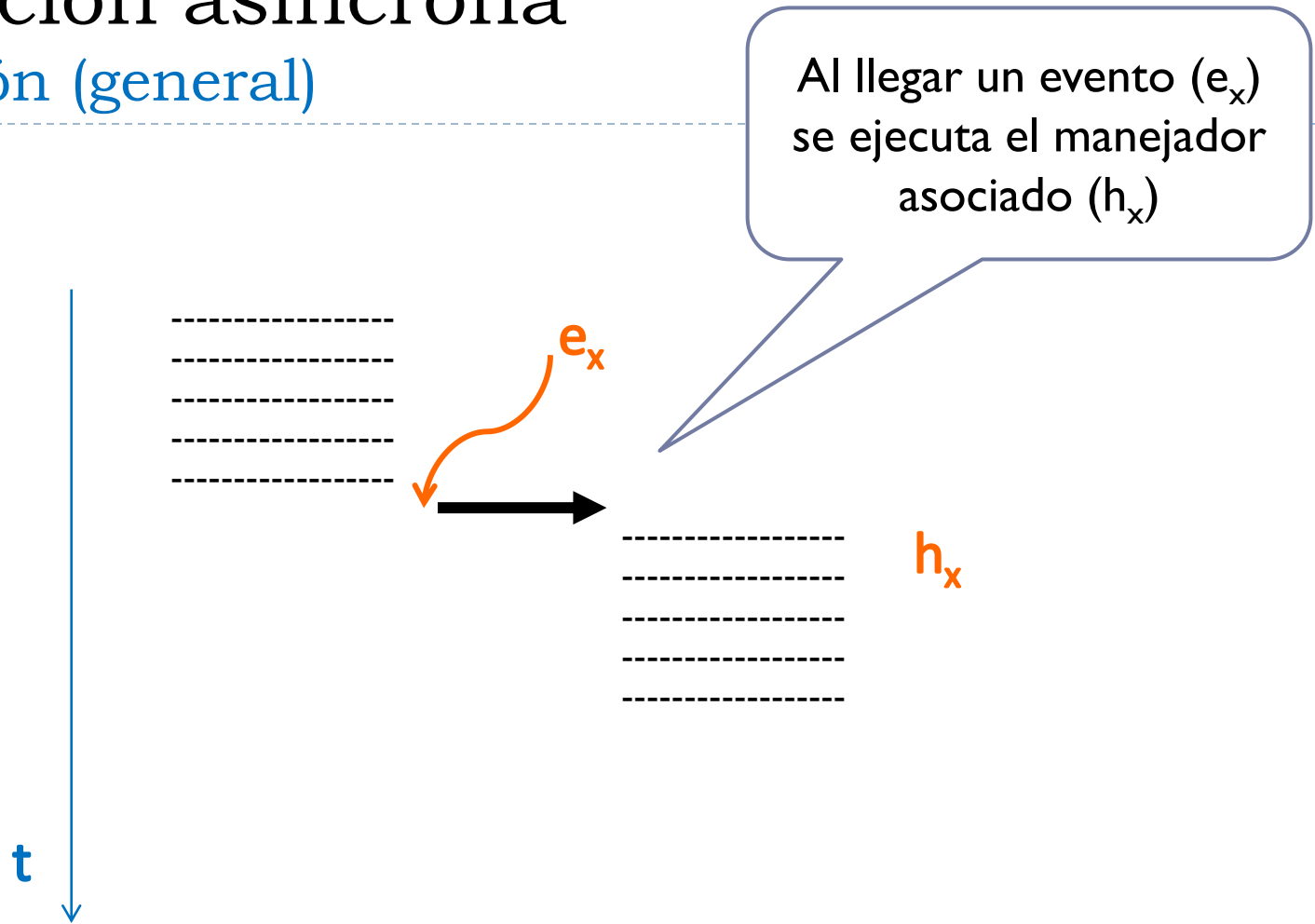
## ejecución (general)

---



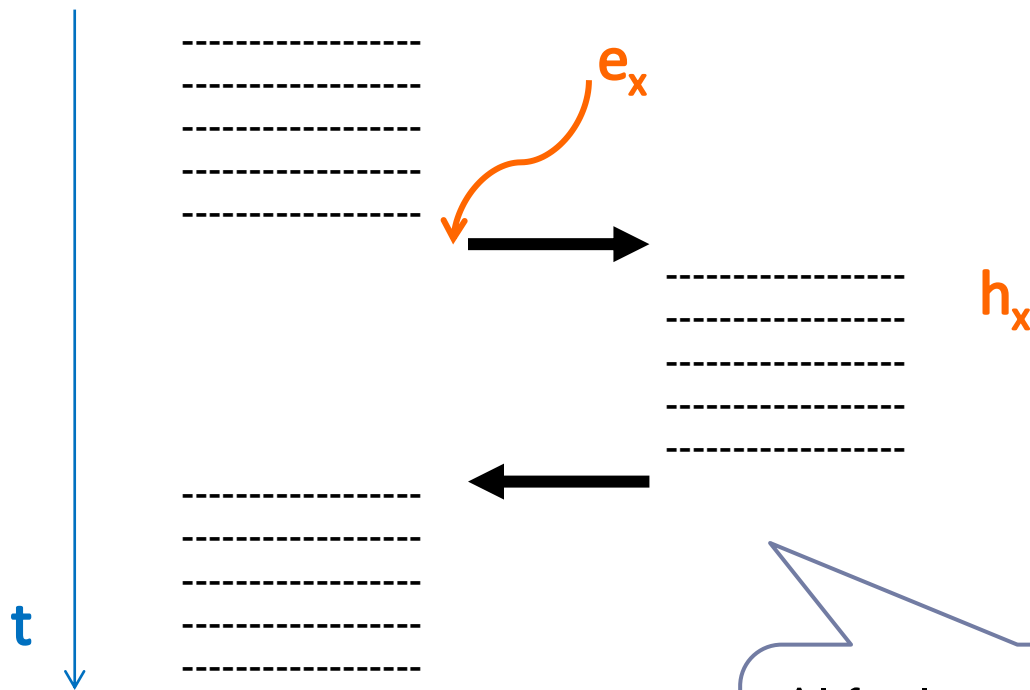
# Ejecución asíncrona

## ejecución (general)



# Ejecución asíncrona

## ejecución (general)



Al finalizar el manejador, se continúa la ejecución por donde fue interrumpida

# Ejecución asíncrona

## código (general)

---

```
int main ( ... )  
{  
  ...  
  On (event1, handler1) ;  
  ...  
}
```

I) Asociar el manejador  
(handler I) al evento

# Ejecución asíncrona

## código (general)

---

```
void handler1 ( ... )  
{  
}
```

2) Codificar la función  
manejador que tratará el evento

...

```
int main ( ... )  
{  
  ...  
  On (event1, handler1) ;  
  ...  
}
```

1) Asociar el manejador  
(handler 1) al evento

# Ejecución asíncrona código (general)

---

```
int global1;
```

```
...
```

```
void handler1 ( ... )
```

```
{  
}
```

```
...
```

```
int main ( ... )
```

```
{
```

```
...
```

```
On (event1, handler1);
```

```
...
```

```
}
```

3) Para comunicar funciones,  
se usa variables globales

2) Codificar la función  
manejador que tratará el evento

1) Asociar el manejador  
(handler 1) al evento

# Ejemplo de ejecución asíncrona

## Señales

signal.h

```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>

void sig_handler (int signo)
{
    if (signo == SIGINT)
        printf("received SIGINT\n");
}

int main(void)
{
    if (signal(SIGINT, sig_handler) == SIG_ERR)
        printf("\ncan't catch SIGINT\n");

    sleep(60); // simula un proceso largo.

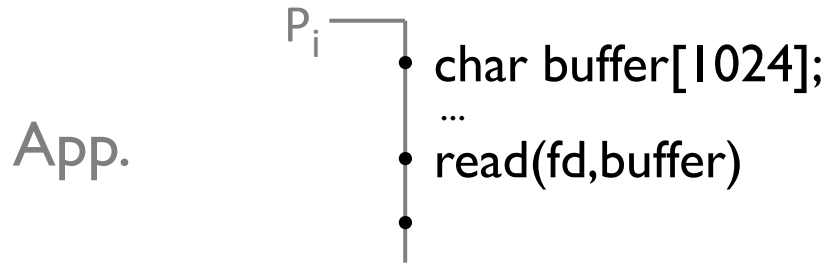
    return 0;
}
```



# Ejecución asíncrona

## ejemplo simplificado

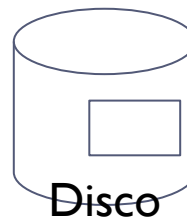
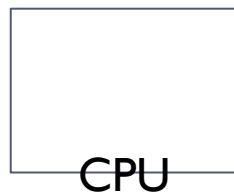
---



S.O.

---

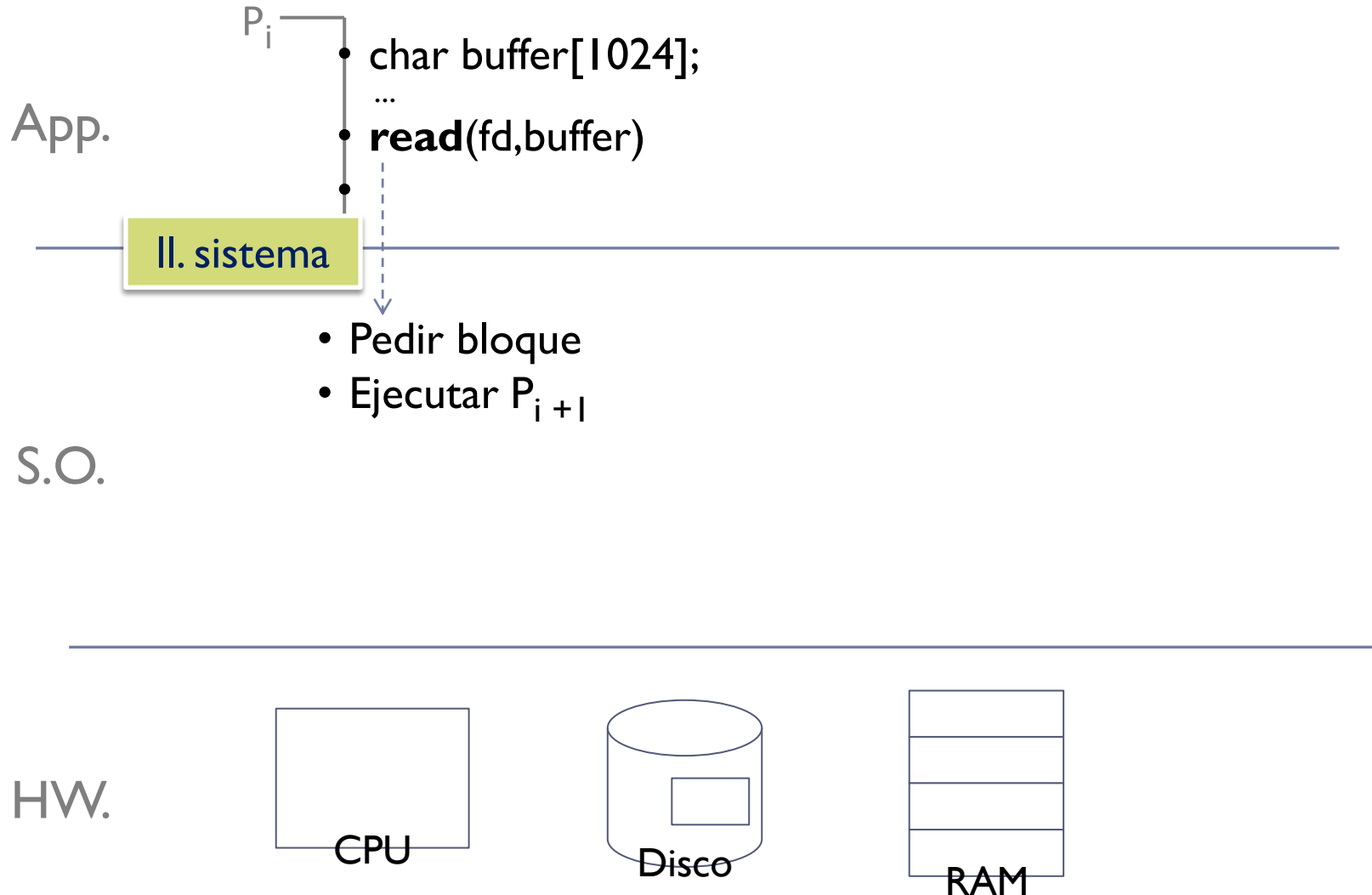
HW.





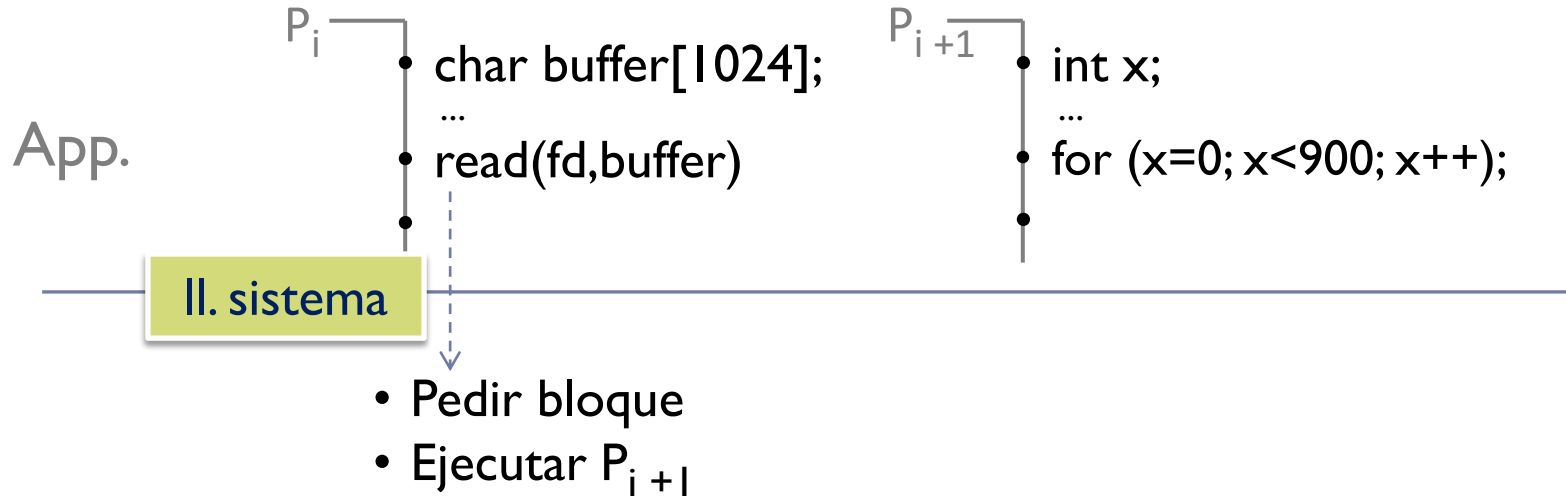
# Ejecución asíncrona

## ejemplo simplificado



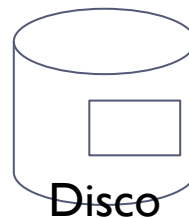
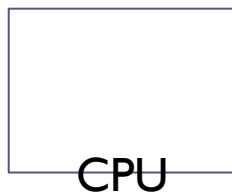
# Ejecución asíncrona

## ejemplo simplificado



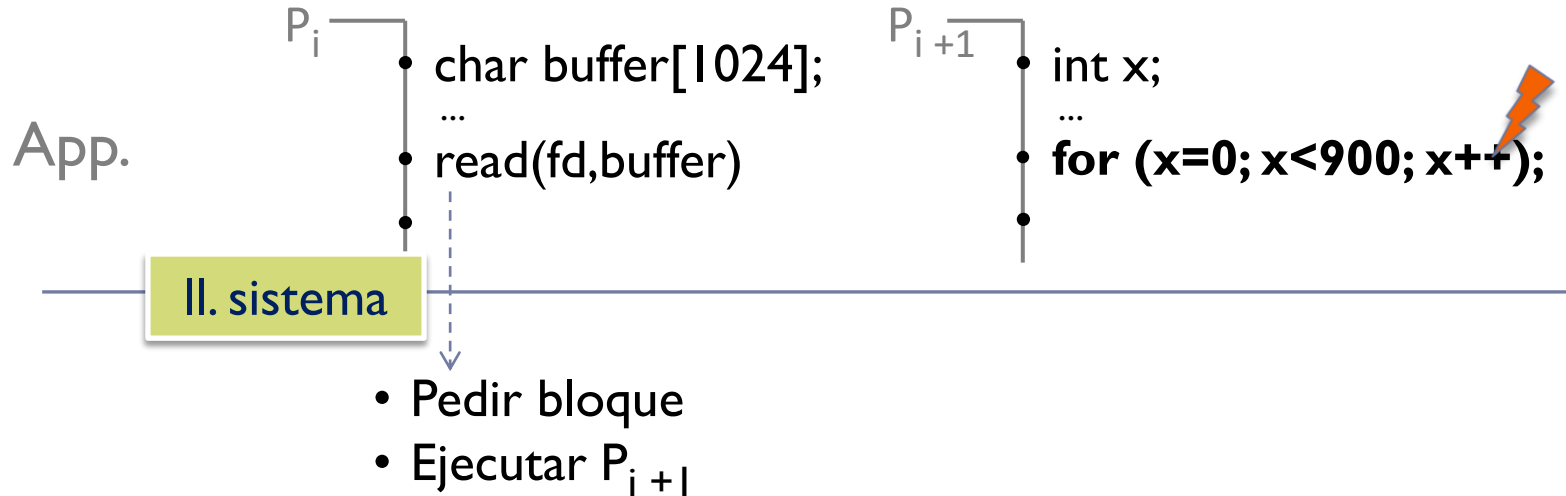
S.O.

HW.

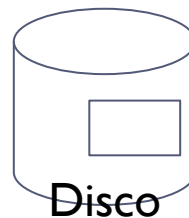
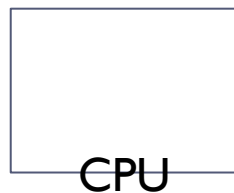


# Ejecución asíncrona

## ejemplo simplificado

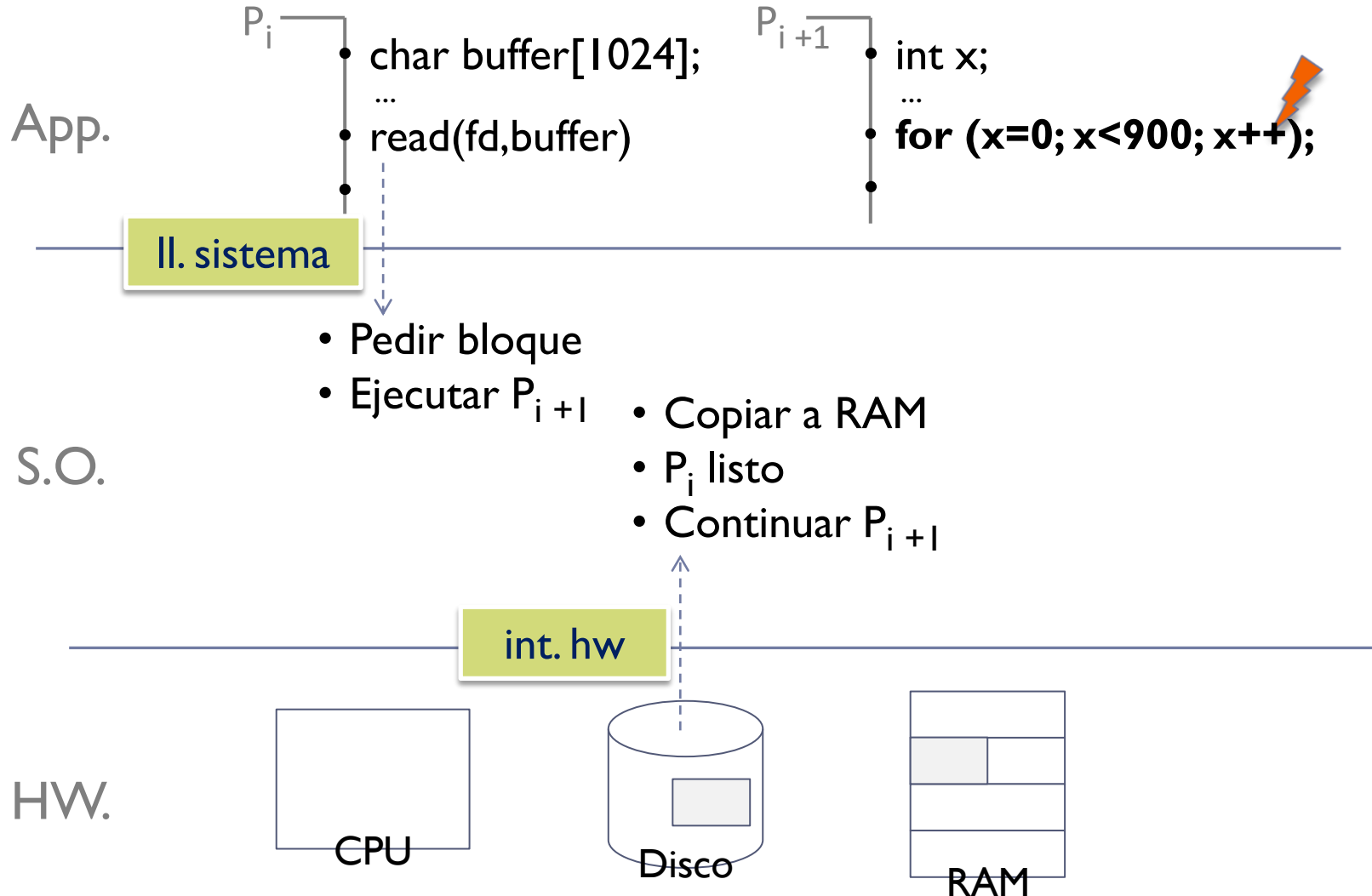


HW.



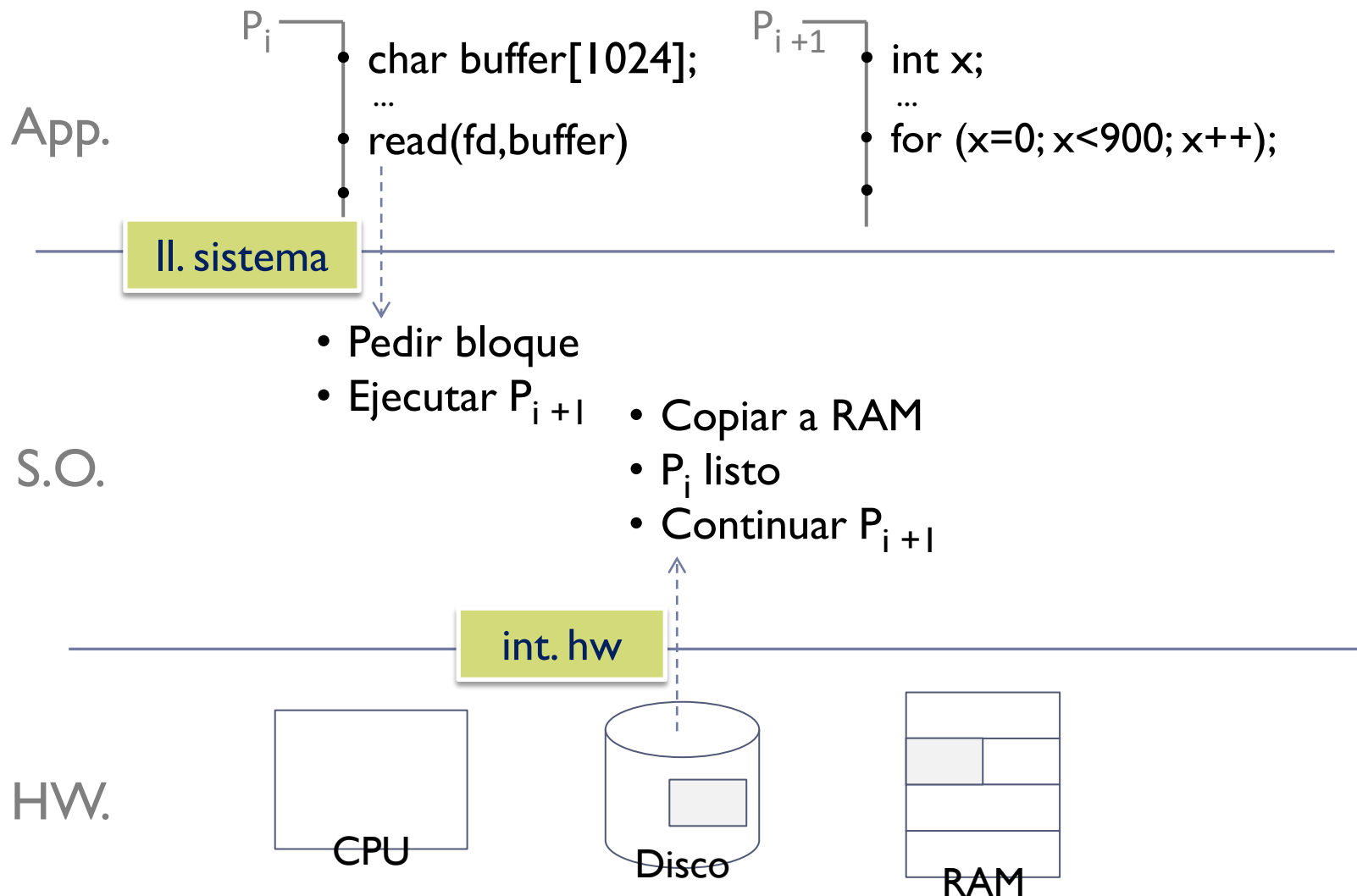
# Ejecución asíncrona

## ejemplo simplificado



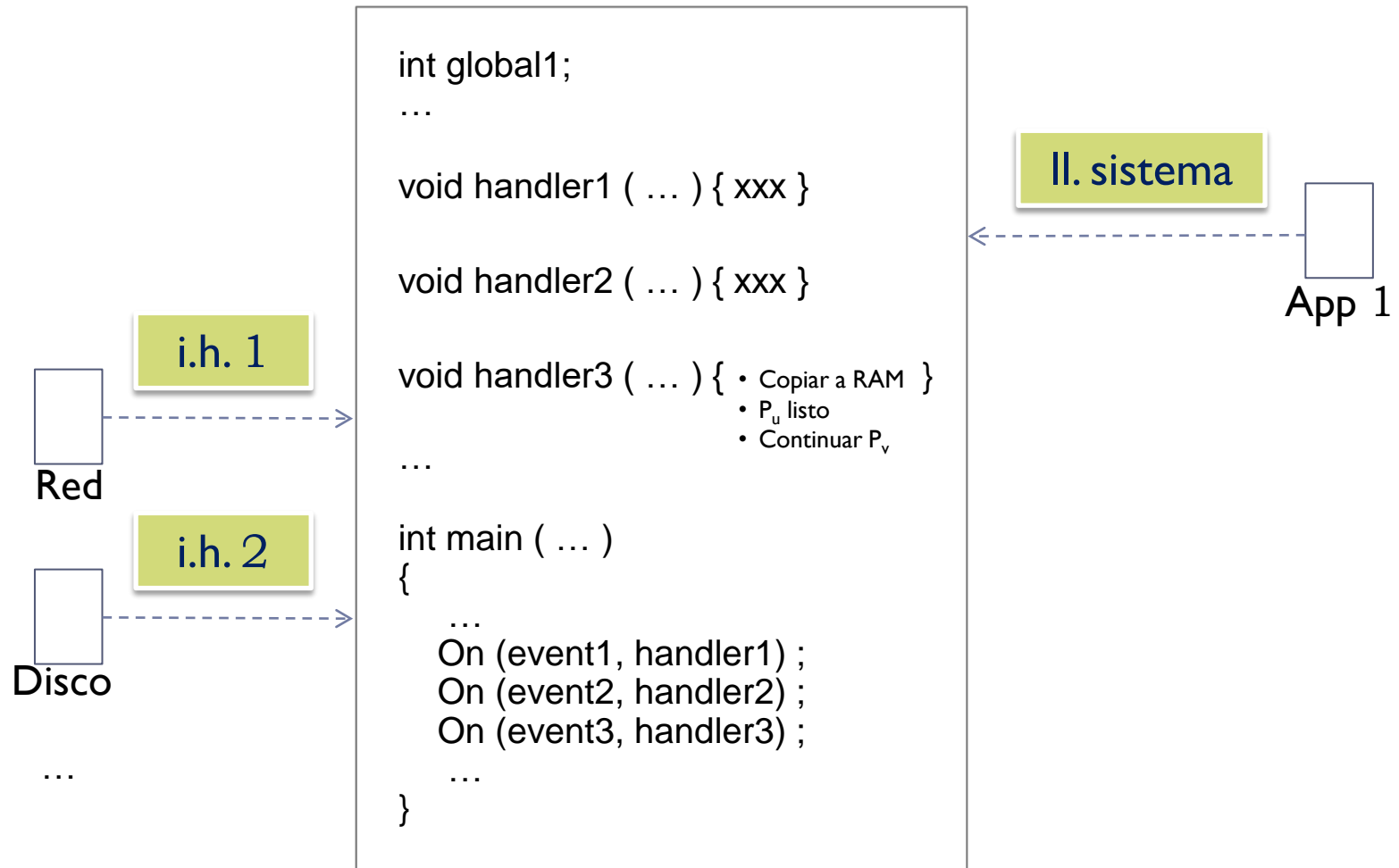
# Ejecución asíncrona

## ejemplo simplificado



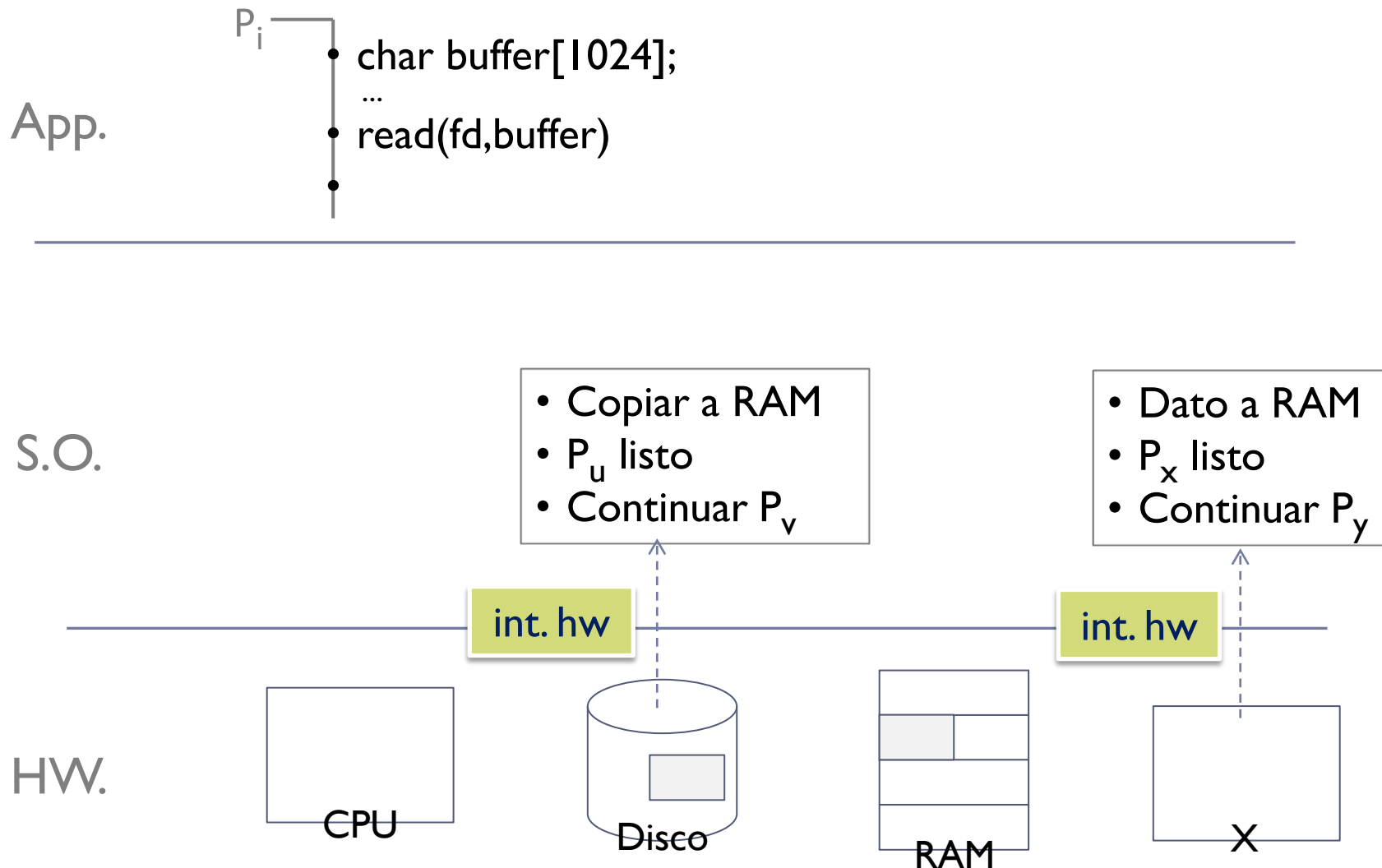
# Estructura base del sistema operativo

## código (general)



# Ejecución asíncrona

## soporte hardware



# Contenidos

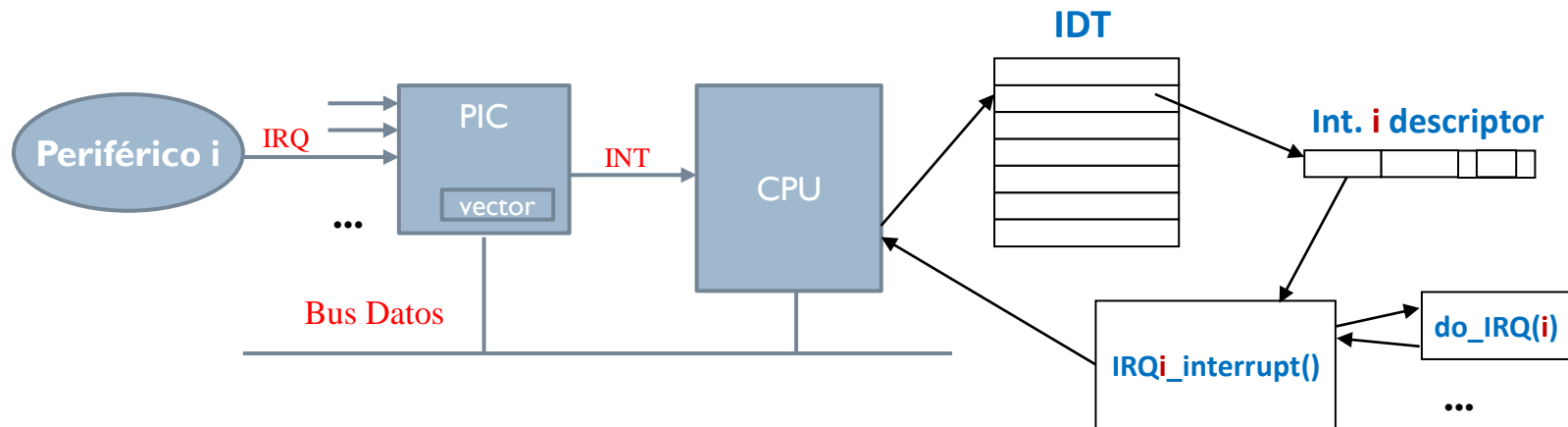
---

1. Orientado a eventos
  1. Ejecución asíncrona e **interrupción**
2. Modular
  1. Núcleo y módulos



# Interrupciones hardware

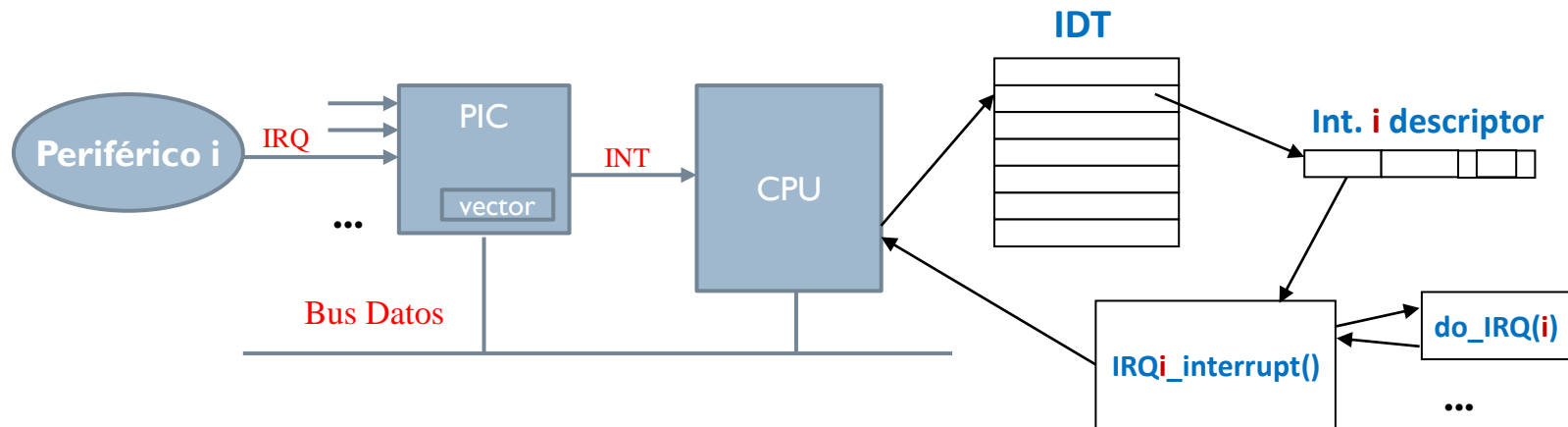
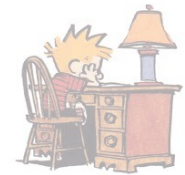
## repaso (1/4)



- ▶ Cada **periférico** (capaz de generar una petición de interrupción) dispone de una línea denominada **IRQ** (*Interrupt ReQuest*)
  - ▶ Puede haber múltiples dispositivos en una línea, se precisa muestreo para conocer el peticionario
- ▶ Todas las líneas se conectan a un **PIC** (*Programmable Interrupt Controller*)
  - ▶ Actualmente se usa un APIC (*Advanced Programmable Interrupt Controller*)
- ▶ El PIC se conecta a la **CPU** por una línea de aviso de interrupción pendiente (**INT**)
- ▶ El PIC y la CPU también están conectados por el **bus de datos**

# Interrupciones hardware

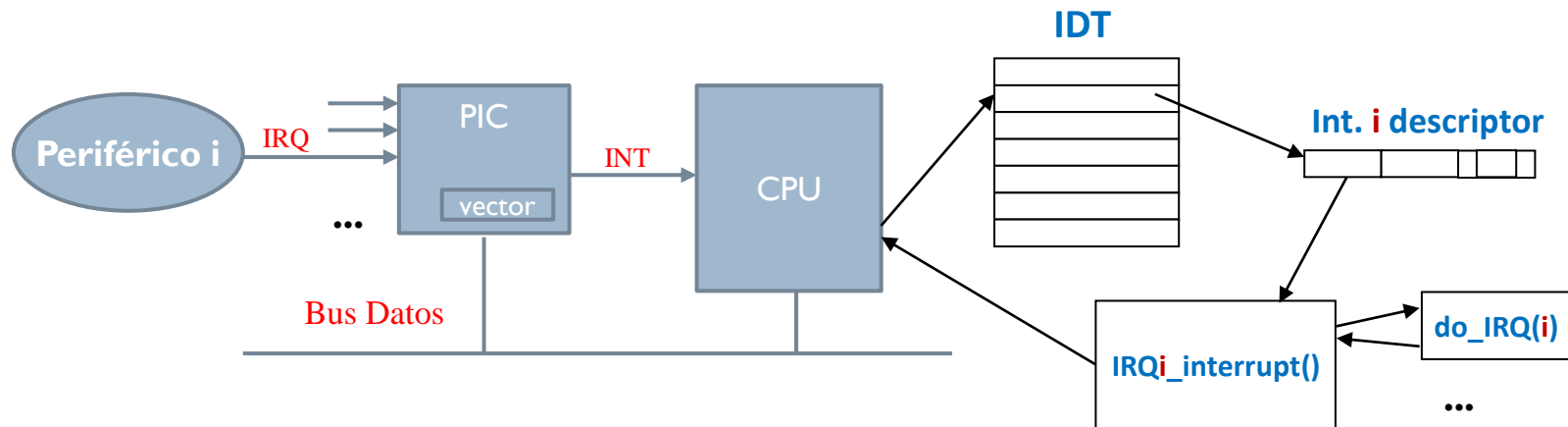
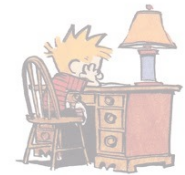
## repaso (2/4)



- ▶ El **PIC monitoriza** las líneas de IRQ esperando la llegada de una señal
- ▶ Si llega una señal entonces:
  - ▶ Asocia al IRQ del periférico un valor que guarda en un registro del PIC (llamado **vector**)
  - ▶ **Avisa a la CPU** a través de la línea de interrupción pendiente (**INT**)
  - ▶ La **CPU lee del registro** (como puerto de E/S o como dirección de memoria) el vector
  - ▶ La CPU escribe en el registro de control del PIC que ya leyó el vector
  - ▶ El **PIC desactiva la línea de interrupción pendiente**, borra el vector y **vuelve a monitorizar...**

# Interrupciones hardware

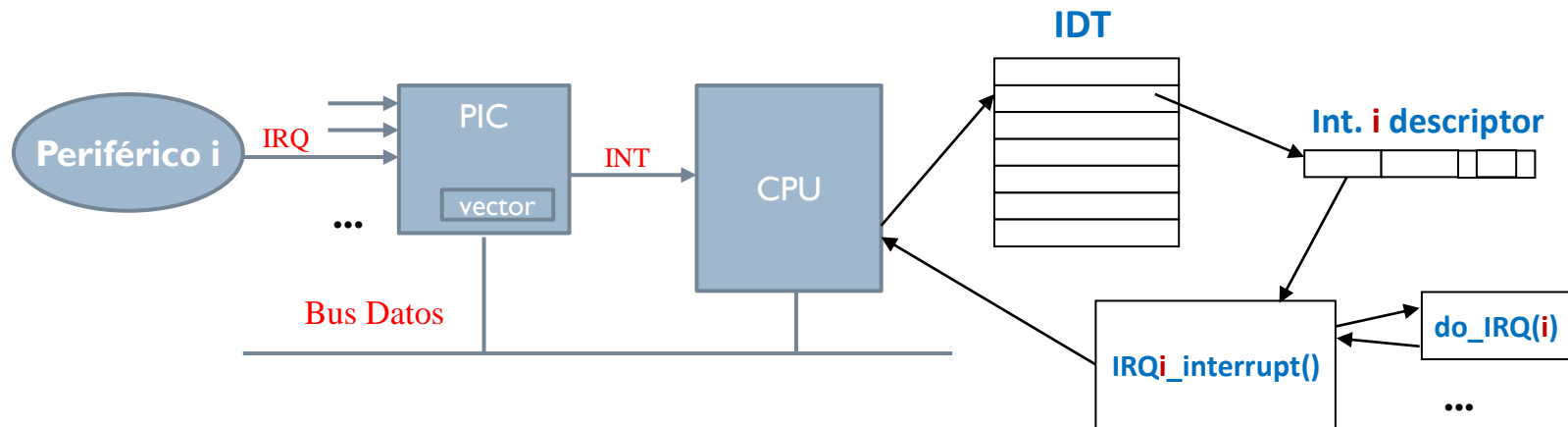
## repaso (3/4)



- ▶ El PIC puede permitir **deshabilitar selectivamente las IRQ**
  - ▶ El PIC deja de avisar a la CPU de la petición de una IRQ hasta que se habiliten: no se pierden.
  - ▶ Deshabilitar a nivel de CPU (mask/unmask) es diferente: ignora la INT
- ▶ El PIC puede permitir tener **niveles de prioridad de interrupción**
  - ▶ Se asocia a cada IRQ con una prioridad
  - ▶ Si hay varias IRQ, el PIC 'atiende' primero las de mayor prioridad (resto: deshabilitado temporalmente)
  - ▶ Si el PIC no tiene niveles de prioridad, se pueden simular por software en el sistema operativo

# Interrupciones hardware

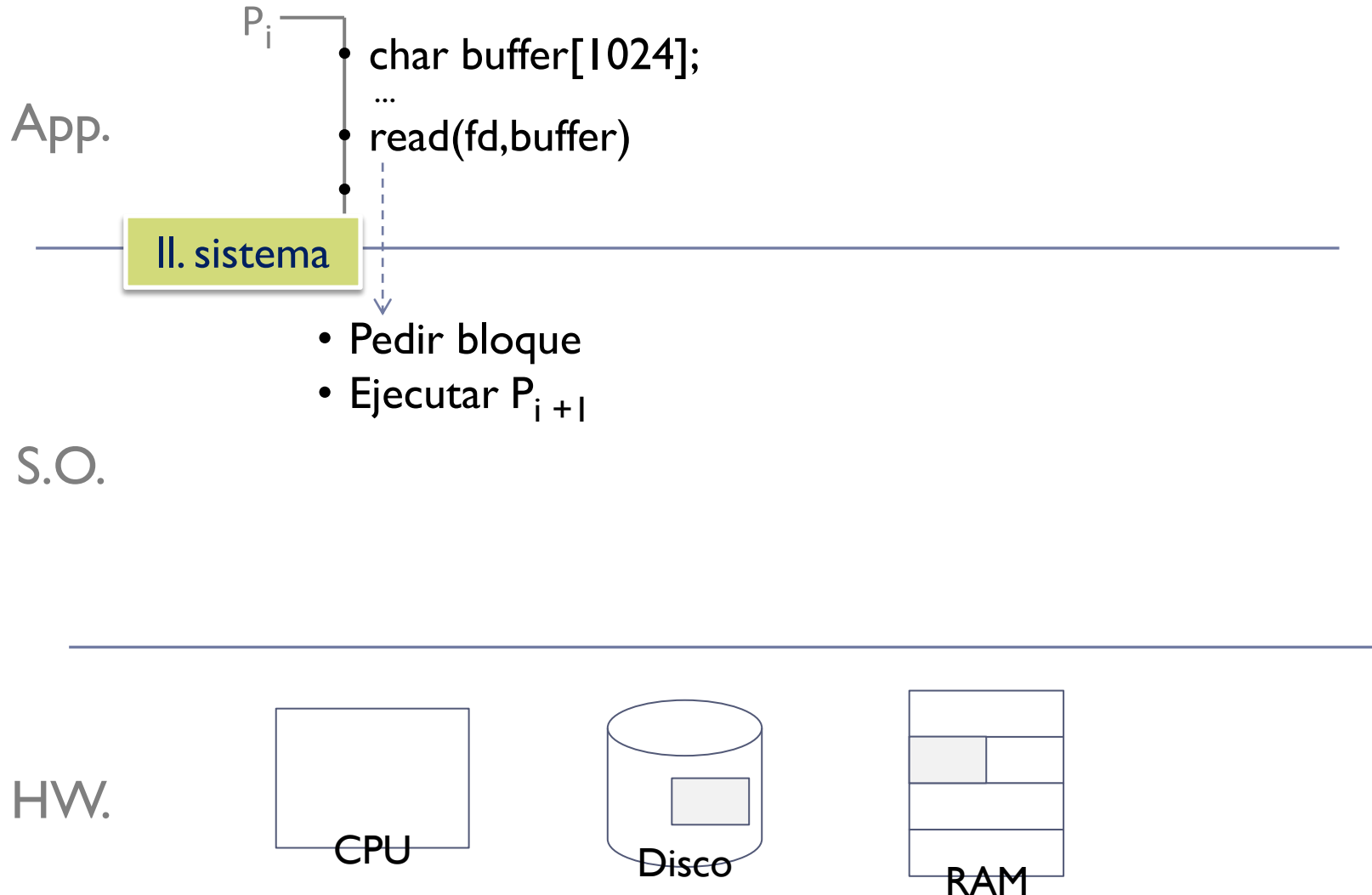
## repaso (4/4)



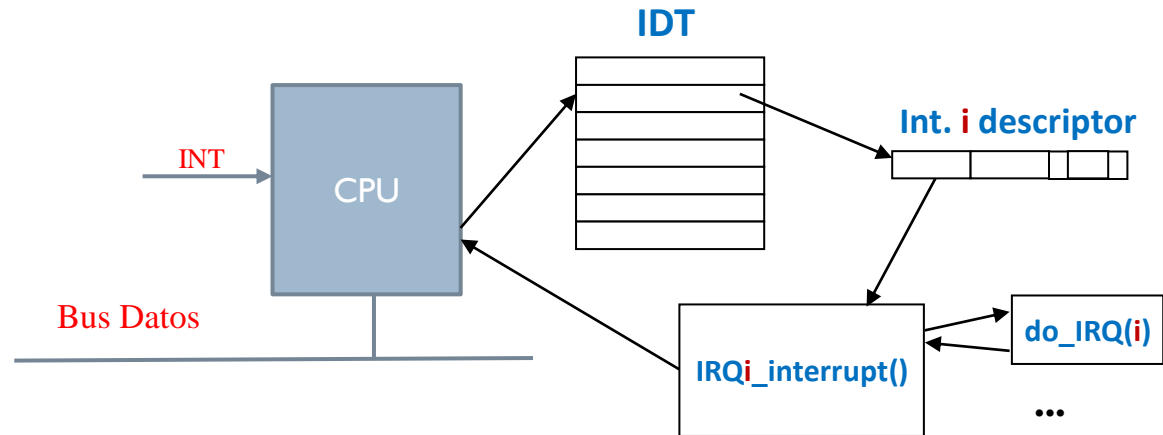
- ▶ La CPU detecta la petición de **INT**
- ▶ Acepta la interrupción: copia el vector por el bus de datos y ACK al PIC
- ▶ Busca en la *Interrupt Descriptor Table* (**IDT**) la rutina de tratamiento asociada
- ▶ Guarda el estado del procesador en pila, pasa a **modo privilegiado** y ejecuta la RTI
  - ▶ Puede que varias RTI (**do\_IRQ**) compartan una misma interrupción (uso de enumeración)
  - ▶ Puede que varias interrupciones compartan una rutina genérica común a ellas
- ▶ Recupera el estado de pila, y ejecuta **RETI** (paso a modo previo y vuelta a lo interrumpido)

# Llamada al sistema

## soporte hardware



# Llamada al sistema



- ▶ Existe una instrucción en ensamblador que genera una interrupción por software
- ▶ La CPU detecta la petición de **INT**
- ▶ Busca en la *Interrupt Descriptor Table* (**IDT**) la rutina de tratamiento asociada
- ▶ Guarda el estado del procesador en pila, pasa a **modo privilegiado** y ejecuta la RTI
  - ▶ Puede que varias RTI (**do\_IRQ**) compartan una misma interrupción (uso de enumeración)
  - ▶ Puede que varias interrupciones compartan una rutina genérica común a ellas
- ▶ Recupera el estado de pila, y ejecuta **RETI** (paso a modo previo y vuelta a lo interrumpido)

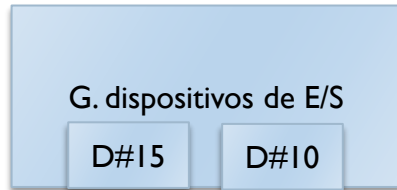
# Contenidos

---

1. **Orientado a eventos**
  1. Ejecución asíncrona e interrupción
2. **Modular**
  1. **Núcleo y módulos**

# Ejecutables (1 / 1)

---

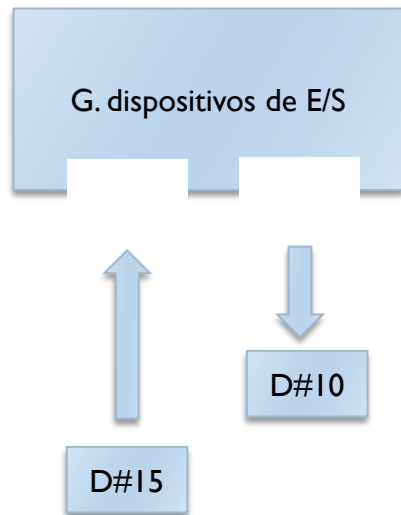


- ▶ Los primeros *kernels* tenían que:
  - ▶ Incluir código para **todos los posibles dispositivos**.
  - ▶ Cada cierto tiempo se **recompilaba** para añadir los nuevos dispositivos.
  - ▶ Se distribuía como un **conjunto de ejecutables**.



# Módulos (1 / 2)

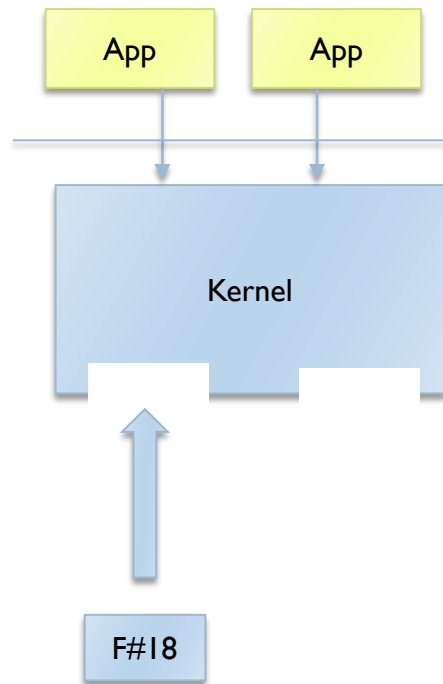
---



- ▶ Los módulos inicialmente se desarrollaron para permitir la **inclusión condicional de controladores de dispositivos (*drivers*)**
- ▶ Los módulos ofrecen **añadir dinámicamente código de un driver pre-compilado**.
- ▶ Se distribuyen como **bibliotecas dinámicas** para el kernel (.so/.dll).
- ▶ El módulo puede **descargarse** cuando el dispositivo deje de usarse.

# Módulos (2/2)

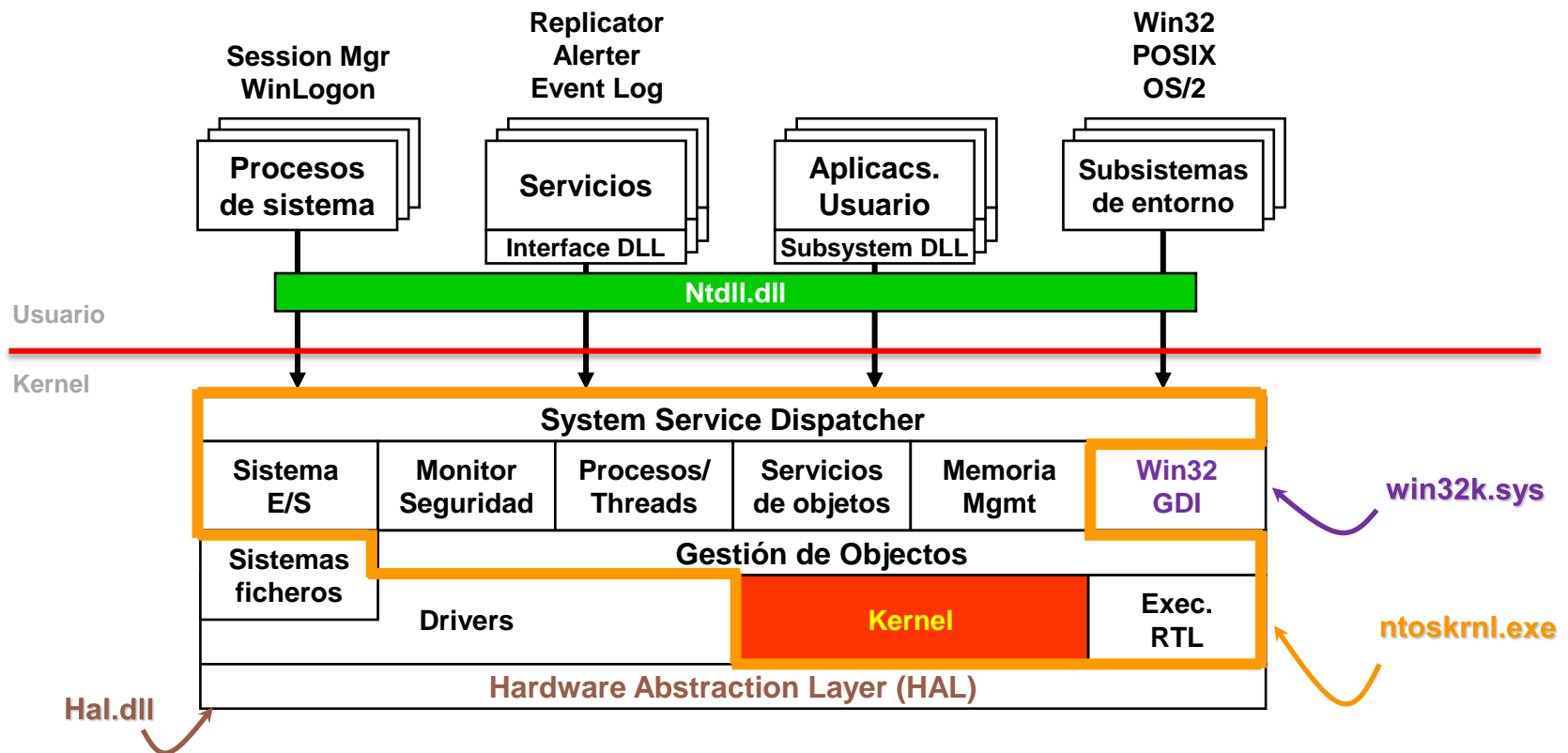
---



- ▶ La gran mayoría de **sistemas operativos modernos** tienen un kernel que permite el **uso de módulos**:
  - ▶ Linux, Solaris, BSD, Windows, etc.
- ▶ Los módulos se utilizan no solo para los drivers de los dispositivos, actualmente **también se utilizan para añadir** otro tipos de **funcionalidad**:
  - ▶ El kernel de Linux lo utiliza extensivamente para sistemas de ficheros, protocolos de red, llamadas al sistema, etc.

# Ejemplo de módulos

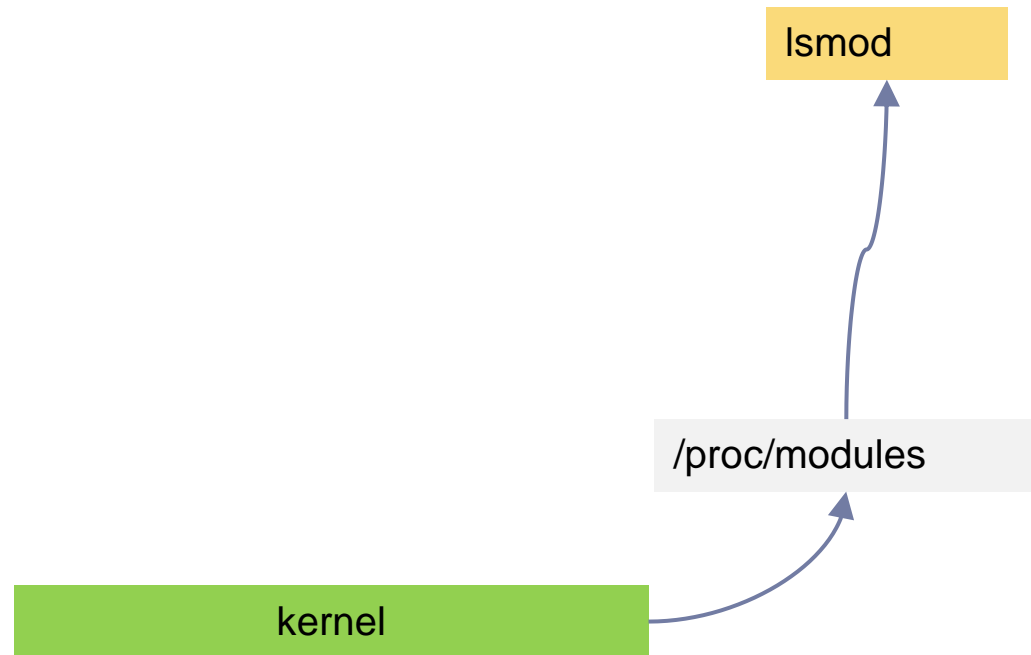
## Windows 2000



# Gestión básica de módulos:

## Linux -> listar

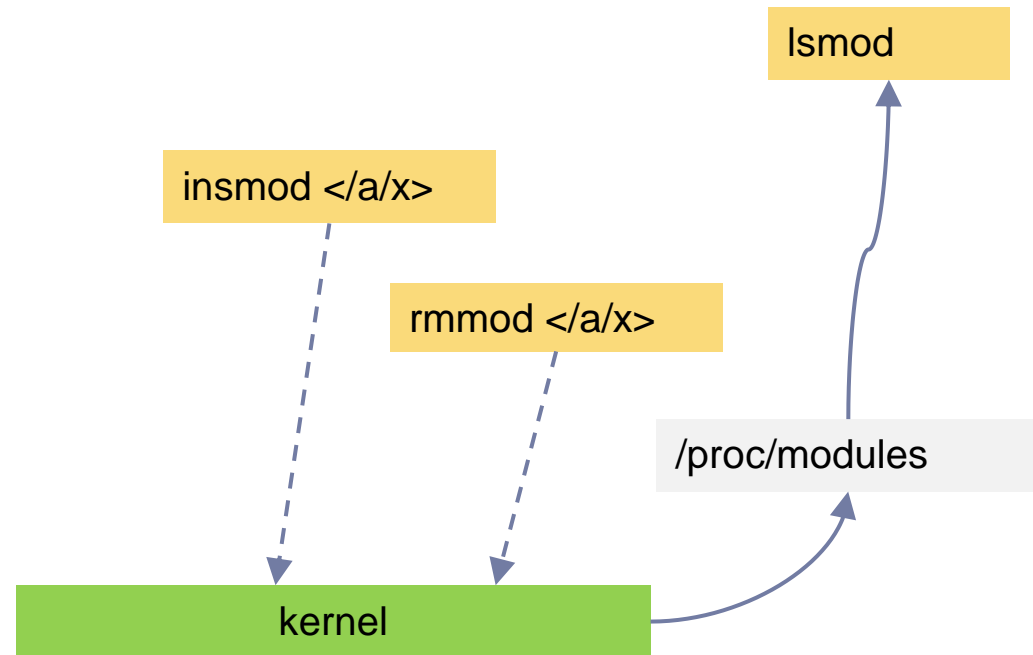
---



# Gestión básica de módulos :

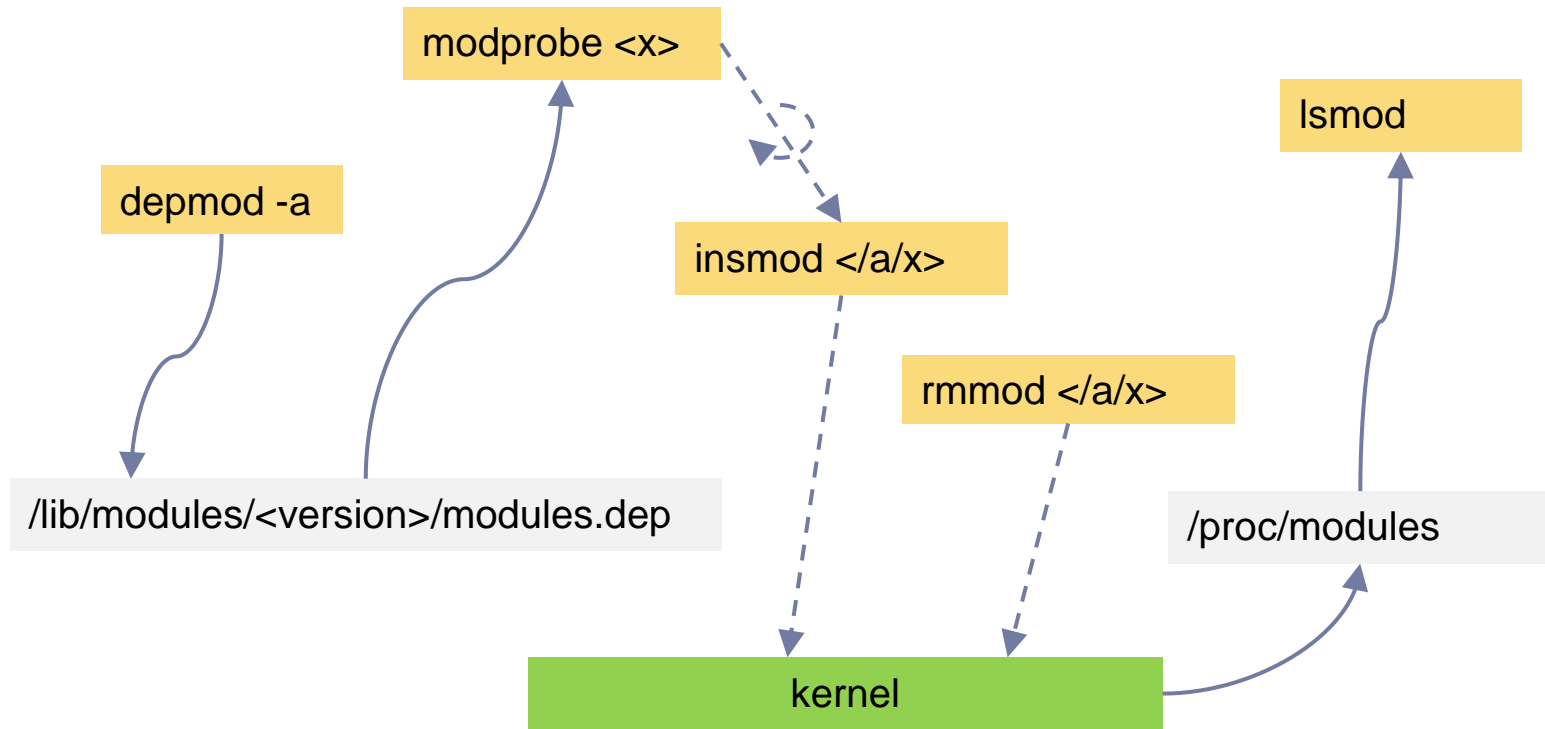
Linux -> añadir/quitar manualmente

---



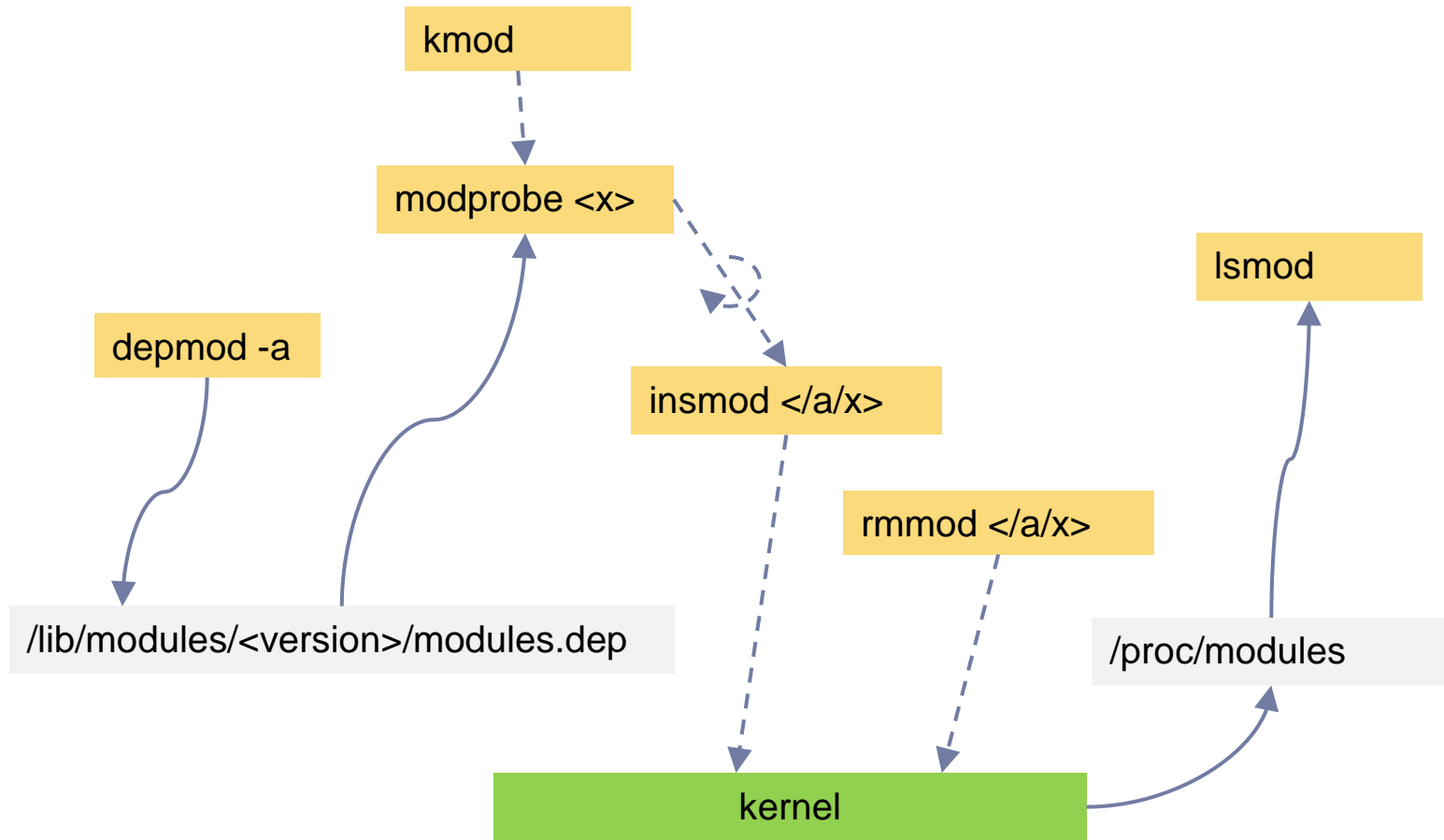
# Gestión básica de módulos

## Linux: añadir con dependencias entre módulos



# Gestión básica de módulos

## Linux



# Contenidos

---

## 1. Orientado a eventos

1. Ejecución asíncrona e interrupción

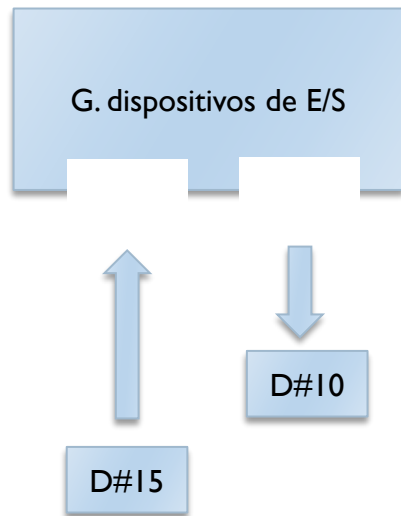
## 2. Modular

1. Núcleo y **módulos -> bibliotecas estáticas y dinámicas**



# Módulos (1 / 2)

---



- ▶ Los módulos inicialmente se desarrollaron para permitir la **inclusión condicional de controladores de dispositivos (*drivers*)**
- ▶ Los módulos ofrecen **añadir dinámicamente código de un driver pre-compilado.**
- ▶ Se distribuyen como **bibliotecas dinámicas** para el kernel (.so/.dll).
- ▶ El módulo puede **descargarse** cuando el dispositivo deje de usarse.

# Bibliotecas: versión inicial

---

declaraciones

```
extern int g1 ;
```

```
int funcion1( int p1,  
             char p2 );
```

mi.h

# Bibliotecas: versión inicial

---

declaraciones

```
extern int g1 ;  
  
int funcion1( int p1,  
             char p2 );
```

mi.h

definiciones

```
int g1 = 10 ;  
  
int funcion1( int p1,  
             char p2 )  
{  
    return p1+(int)p2 ;  
}
```

mi.c

# Bibliotecas: versión inicial

declaraciones

```
extern int g1 ;  
  
int funcion1( int p1,  
             char p2 );
```

mi.h

definiciones

```
int g1 = 10 ;  
  
int funcion1( int p1,  
             char p2 )  
{  
    return p1+(int)p2 ;  
}
```

mi.c

main.c

```
#include "mi.h"  
#include <stdio.h>  
  
int main ( int argc,  
          char *argv[] )  
{  
    int r ;  
  
    r=funcion1(5,'0') ;  
    printf("r=%d\n",r) ;  
    return 0 ;  
}
```

# Bibliotecas: versión inicial

declaraciones

```
extern int g1 ;  
  
int funcion1( int p1,  
             char p2 );
```

mi.h

definiciones

```
int g1 = 10 ;  
  
int funcion1( int p1,  
             char p2 )  
{  
    return p1+(int)p2 ;  
}
```

mi.c

main.c

```
#include "mi.h"  
#include <stdio.h>  
  
int main ( int argc,  
          char *argv[] )  
{  
    int r ;  
  
    r=funcion1(5,'0') ;  
    printf("r=%d\n",r) ;  
    return 0 ;  
}
```

# Bibliotecas: usuario vs sistema

declaraciones

```
extern int g1 ;  
  
int funcion1( int p1,  
             char p2 );
```

mi.h

definiciones

```
int g1 = 10 ;  
  
int funcion1( int p1,  
             char p2 )  
{  
    return p1+(int)p2 ;  
}
```

mi.c

bibliotecas de sistema o de usuario

bibliotecas de sistema

```
#include "mi.h"  
#include <stdio.h>  
  
int main ( int argc,  
          char *argv[] )  
{  
    int r ;  
  
    r=funcion1(5,'0') ;  
    printf("r=%d\n",r) ;  
    return 0 ;  
}
```

# Bibliotecas: compilación y enlazado

declaraciones

```
extern int g1 ;  
  
int funcion1( int p1,  
             char p2 );
```

mi.h

definiciones

```
int g1 = 10 ;  
  
int funcion1( int p1,  
             char p2 )  
{  
    return p1+(int)p2 ;  
}
```

mi.c

```
extern int g1 ;  
  
int funcion1( int p1,  
             char p2 );  
  
... // stdio.h  
  
int main ( int argc,  
          char *argv[] )  
{  
    int r ;  
  
    r=funcion1(5,'0') ;  
    printf("r=%d\n",r) ;  
    return 0 ;  
}
```

#include indica al preprocesador: “cambiarme por el contenido del archivo”

# Bibliotecas: versión completa

declaraciones

```
#ifndef _MI_H_
#define _MI_H_
extern int g1 ;

int funcion1( int p1,
             char p2 );
#endif
```

mi.h

definiciones

```
#include "mi.h"

int g1 = 10 ;

int funcion1( int p1,
             char p2 )
{
    return p1+(int)p2 ;
}
```

mi.c

main.c

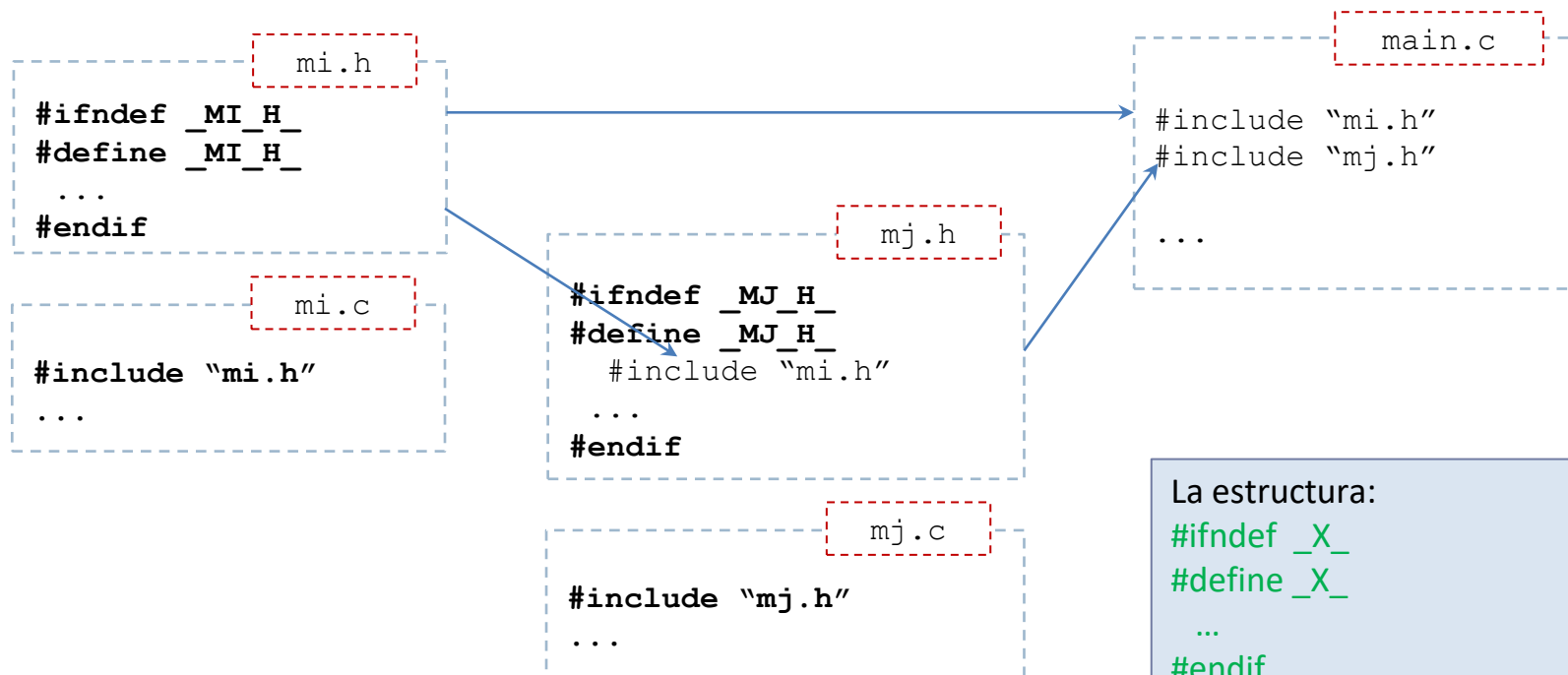
```
#include "mi.h"
#include <stdio.h>

int main ( int argc,
          char *argv[] )
{
    int r ;

    r=funcion1(5,'0') ;
    printf("r=%d\n",r) ;
    return 0 ;
}
```



# Bibliotecas: versión completa



La estructura:

```
#ifndef _X_
#define _X_
...
#endif
```

evita múltiples inclusiones  
no necesarias

# Bibliotecas: compilación y enlazado

declaraciones

```
#ifndef _MI_H_
#define _MI_H_
extern int g1 ;

int funcion1( int p1,
              char p2 );
#endif
```

mi.h

definiciones

```
#include "mi.h"

int g1 = 10 ;

int funcion1( int p1,
              char p2 )
{
    return p1+(int)p2 ;
}
```

mi.c

main.c

```
#include "mi.h"
#include <stdio.h>

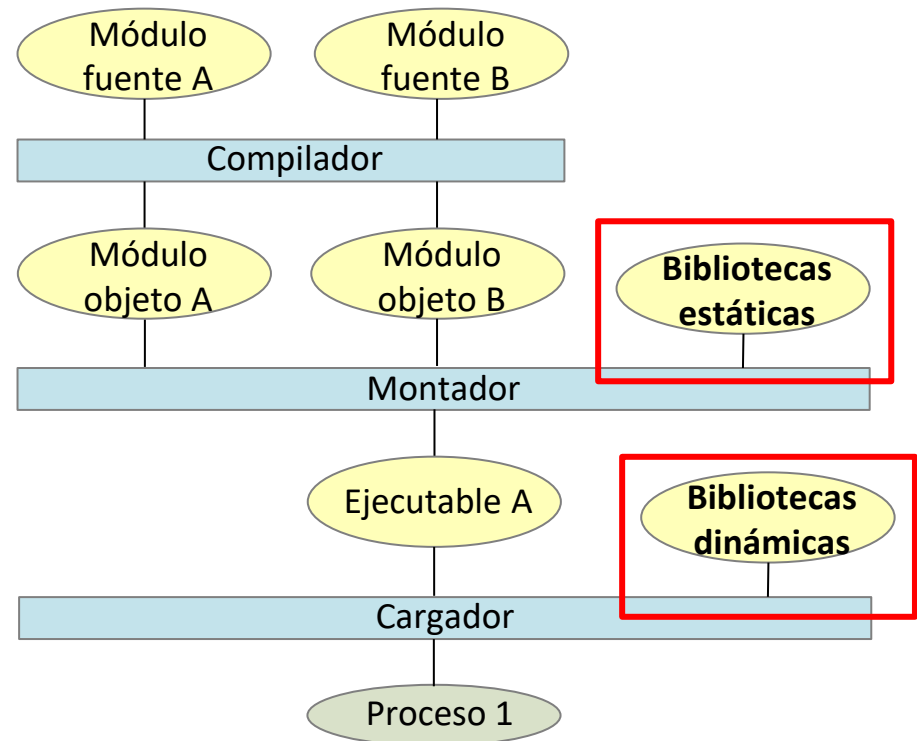
int main ( int argc,
           char *argv[] )
{
    int r ;

    r=funcion1(5,'0') ;
    printf("r=%d\n",r) ;
    return 0 ;
}
```

```
gcc -Wall -g -o mi.o -c mi.c
gcc -Wall -g -o main.o -c main.c
gcc -o main main.o mi.o
```

# Generación y ejecución de programas

- ❑ Aplicación
  - ❑ Conjunto de módulos en lenguaje de alto nivel
- ❑ Fases:
  - ❑ Compilación
  - ❑ Montaje
  - ❑ Enlazado dinámico
  - ❑ Ejecución

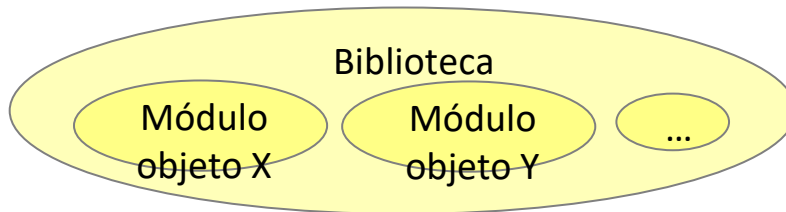


# Bibliotecas de objetos

---

## ❑ Biblioteca

- ❑ Colección de módulos objetos relacionados



## ❑ Biblioteca **estática**

- ❑ Carga y montaje en tiempo de compilación

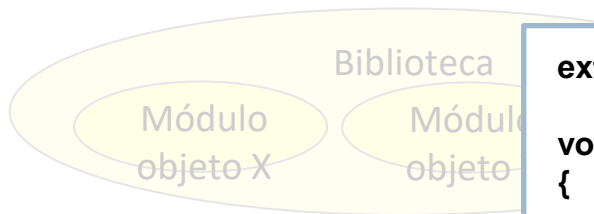
## ❑ Biblioteca **dinámica**

- ❑ Carga y montaje en tiempo de ejecución
- ❑ Se indica al montar qué biblioteca usar, carga y montaje posterior

# Bibliotecas de objetos

## □ Biblioteca

- Colección de módulos objetos relacionados



```
#include <stdio.h>

void decir ( char * str )
{
    printf("%s",str) ;
}
```

b.c

```
extern void decir ( char * str ) ;

void decir_hola( void )
{
    decir("Hola mundo...\n") ;
}
```

a.c

```
extern void decir_hola( void ) ;

int main (int argc, char *argv[])
{
    decir_hola() ;
    return 0 ;
}
```

main.c

o de compilación

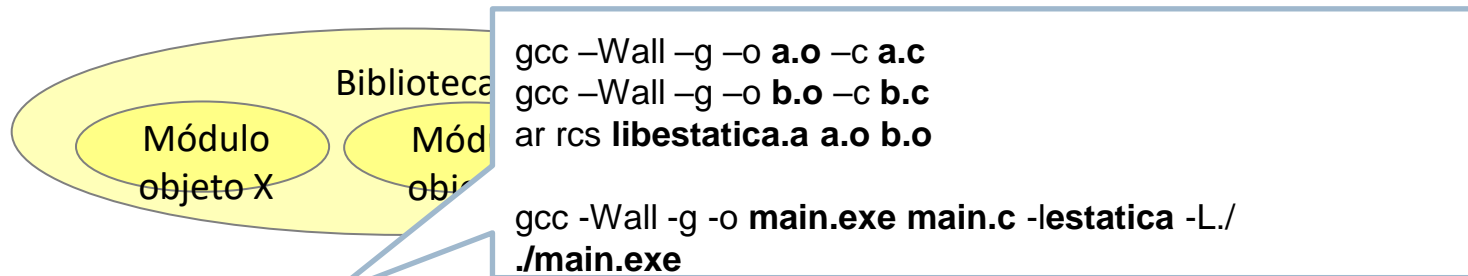
o de ejecución

lioteca usar, carga y montaje posterior

# Bibliotecas de objetos

## ❑ Biblioteca

- ❑ Colección de módulos objetos relacionados



## ❑ Biblioteca **estática**

- ❑ Carga y montaje en tiempo de compilación

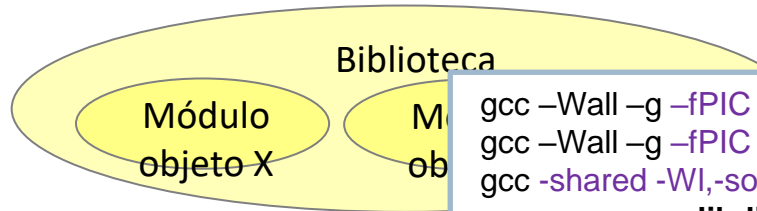
## ❑ Biblioteca **dinámica**

- ❑ Carga y montaje en tiempo de ejecución
- ❑ Se indica al montar qué biblioteca usar, carga y montaje posterior

# Bibliotecas de objetos

## ❑ Biblioteca

- ❑ Colección de módulos objetos relacionados



## ❑ Biblioteca **estática**

- ❑ Carga y montaje en tiempo de compilación

## ❑ Biblioteca **dinámica**

- ❑ Carga y montaje en tiempo de ejecución
- ❑ Se indica al montar qué biblioteca usar, carga y montaje posterior

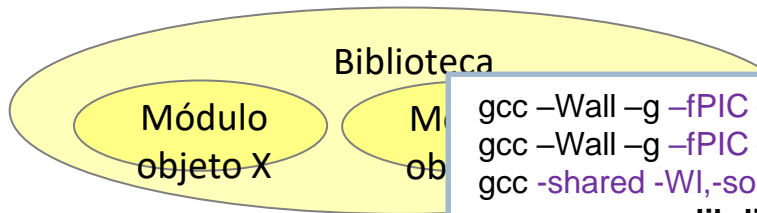
```
gcc -Wall -g -fPIC -o a.o -c a.c
gcc -Wall -g -fPIC -o b.o -c b.c
gcc -shared -Wl,-soname,libdinamica.so \
    -o libdinamica.so.1.0 a.o b.o
ln -s libdinamica.so.1.0 libdinamica.so

gcc -Wall -g -o main.exe main.c -l. -L. -ldinamica
env LD_LIBRARY_PATH=$LD_LIBRARY_PATH:. ./main.exe
```

# Bibliotecas de objetos

## ❑ Biblioteca

- ❑ Colección de módulos objetos relacionados



## ❑ Biblioteca **estática**

- ❑ Carga y montaje en tiempo de compilación

## ❑ Biblioteca **dinámica**

- ❑ Carga y montaje en tiempo de ejecución
- ❑ Se indica al montar qué biblioteca usar, carga y montaje posterior

```
gcc -Wall -g -fPIC -o a.o -c a.c
gcc -Wall -g -fPIC -o b.o -c b.c
gcc -shared -Wl,-soname,libdinamica.so \
    -o libdinamica.so.1.0 a.o b.o
ln -s libdinamica.so.1.0 libdinamica.so

gcc -Wall -g -o main.exe main.c -l. -L. -ldinamica -Wl,-rpath=$(pwd)
./main.exe
```



Grupo ARCOS  
Departamento de Informática  
Universidad Carlos III de Madrid

# Lección 1 (b)

## Introducción

Diseño de Sistemas Operativos  
Grado en Ingeniería Informática y Doble Grado I.I. y A.D.E.

