

# SISTEMAS OPERATIVOS: SISTEMAS DE FICHEROS



Ficheros, directorios y sistema de ficheros

# 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 prácticas** y las **prácticas** de forma progresiva.

# Lecturas recomendadas

## Base



1. Carretero 2020:
  1. Cap. 6
2. Carretero 2007:
  1. Cap. 9.1-9.5,
  2. Cap. 9.8-9.10 y 9.12

## Recomendada



1. Tanenbaum 2006:
  1. (es) Cap. 6
  2. (en) Cap. 6
2. Stallings 2005:
  1. 12.1-12.8
3. Silberschatz 2006:
  1. 10.3-10.4,
  2. 11.1-11.6 y 13

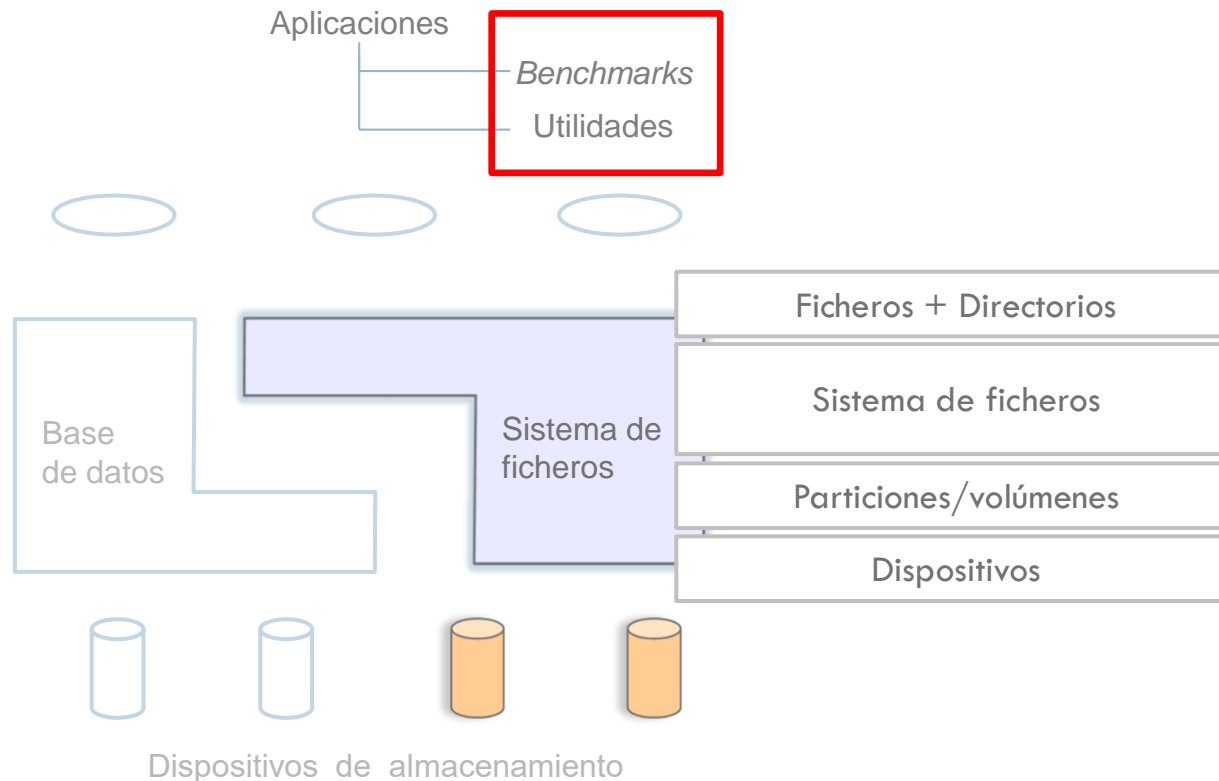
# Contenidos

- Introducción
- Fichero
- Directorio
- Sistema de ficheros
- Particiones/Volúmenes
- Dispositivos
- Software de sistema
- Sistema de ficheros (gestor)

# Contenidos

- Introducción
- Fichero
- Directorio
- Sistema de ficheros
- Particiones/Volúmenes
- Dispositivos
- **Software de sistema**
- Sistema de ficheros (gestor)

# Software de sistema



# Benchmarks

- *Benchmarks:*
  - ▣ Permiten medir las **prestaciones del sistema de ficheros** (y toda dependencia del mismo)
  - ▣ Diseñados para medir **diferentes aspectos:** latencia, ancho de banda, número de ficheros procesados por unidad de tiempo, etc.
  - ▣ Ejemplos trabajando con metadatos: fdtree, mdtest, etc.
  - ▣ Ejemplos trabajando con datos: iozone, postmark, IOR, etc.

# Consistencia del sistema de archivos

- Fallos en software pueden que la información (y metadatos) quede inconsistente.
- Solución:
  - ▣ Disponer de herramientas que revisen el sistema de archivos y permita reparar los errores encontrados.
- Dos aspectos importantes a revisar:
  - ▣ Comprobar que la **estructura física** del sistema de archivos es coherente.
  - ▣ Verificar que la **estructura lógica** del sistema de archivos es correcta.



# Consistencia del sistema de archivos

- ❑ Fallos en software pueden que la información (y metadatos) quede inconsistente.
- ❑ Solución:
  - ❑ Disponer de herramientas que revisen el sistema de archivos y permita reparar los errores encontrados.
- ❑ Dos aspectos importantes a revisar:

- ❑ Comprobar que la **estructura física** del sistema de archivos es coherente.

  - ❑ Verificar que la **estructura lógica** del sistema de archivos es correcta.

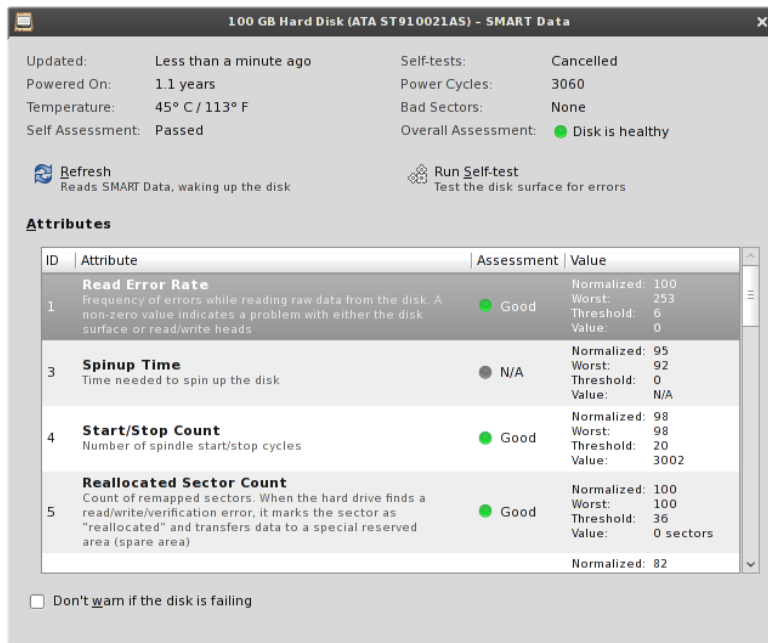
# Consistencia del sistema de archivos

## estructura física

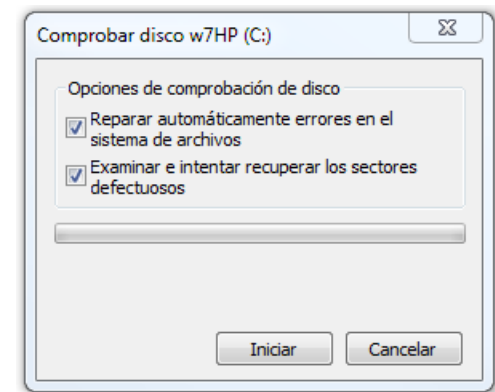
10

Alejandro Calderón Mateos 

- ❑ Lógica del controlador:
  - ❑ Se realizan pruebas del estado del disco-controlador
  - ❑ Ej.: S.M.A.R.T.



- ❑ Superficie del disco:
  - ❑ Se lee/escribe los bloques de disco uno a uno para comprobar problemas en la superficie de parte del disco.
  - ❑ Ej.: si lo leído es diferente a lo escrito



# Consistencia del sistema de archivos

- Fallos en software pueden que la información (y metadatos) quede inconsistente.
- Solución:
  - ▣ Disponer de herramientas que revisen el sistema de archivos y permita reparar los errores encontrados.
- Dos aspectos importantes a revisar:
  - ▣ Comprobar que la **estructura física** del sistema de archivos es coherente.
  - ▣ Verificar que la **estructura lógica** del sistema de archivos es correcta.

# Consistencia del sistema de archivos

## estructura lógica

12

Alejandro Calderón Mateos 

- Estructuras en disco:
  - ▣ Comprobar que las estructura de datos en disco son coherentes para la partición, directorios y archivos
  - ▣ Ej.: fsck en Linux, scandisk en Windows

```
acaldero@phoenix:/tmp$ sudo fsck -f /dev/loop1
fsck desde util-linux-ng 2.17.2
e2fsck 1.41.12 (17-May-2010)
Paso 1: Verificando nodos-i, bloques y tamaños
Paso 2: Verificando la estructura de directorios
Paso 3: Revisando la conectividad de directorios
Paso 4: Revisando las cuentas de referencia
Paso 5: Revisando el resumen de información de grupos
/dev/loop1: 11/28560 ficheros (0.0% no contiguos), 5161/114180 bloques
acaldero@phoenix:/tmp$
```

# Consistencia del sistema de archivos

## estructura lógica

- Sistema de ficheros en disco:
  - ▣ Se comprueba que el contenido del superbloque responde a las características del sistema de archivos.
  - ▣ Se comprueba que los mapas de bits de nodos-i se corresponden con los nodos-i ocupados en el sistema de archivos.
  - ▣ Se comprueba que los mapas de bits de bloques se corresponden con los bloques asignados a archivos.
  - ▣ Se comprueba que ningún bloque esté asignado a más de un archivo.
- Directorios:
  - ▣ Se comprueba el sistema de directorios del sistema de archivos, para ver que un mismo nodo-i no está asignado a más de un directorio.
- Archivos:
  - ▣ Se comprueba que los bits de protección y privilegios.
  - ▣ Se comprueba el contador de enlaces.

# Backup (copia de seguridad)

## ¿Dónde?

14

Alejandro Calderón Mateos 

### □ Lugar:

- Distante del sistema principal
- Protegido del agua, fuego, etc.
  - Armarios ignífugos



### □ Medio:

- Disco duro
  - V: capacidad y precio, l: delicado
- Cinta
  - V: capacidad y precio, l: lentitud



# Backup (copia de seguridad)

## ¿Cómo?

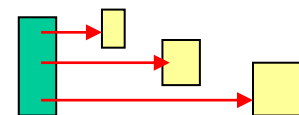
15

Alejandro Calderón Mateos 

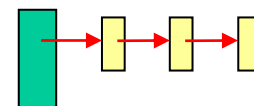
- **Completo** (*full backup*):  
copiar todo el contenido del sistema de ficheros.



- **Diferencial** (*differential backup*):  
contiene todos los ficheros que han sido modificados desde la última copia de seguridad **completa**.



- **Incremental** (*incremental backup*):  
contiene todos los ficheros que han sido modificados desde la última copia de seguridad, ya sea **completa** o **diferencial**



# Backup (copia de seguridad)

## ¿Cuándo?

16

Alejandro Calderón Mateos 

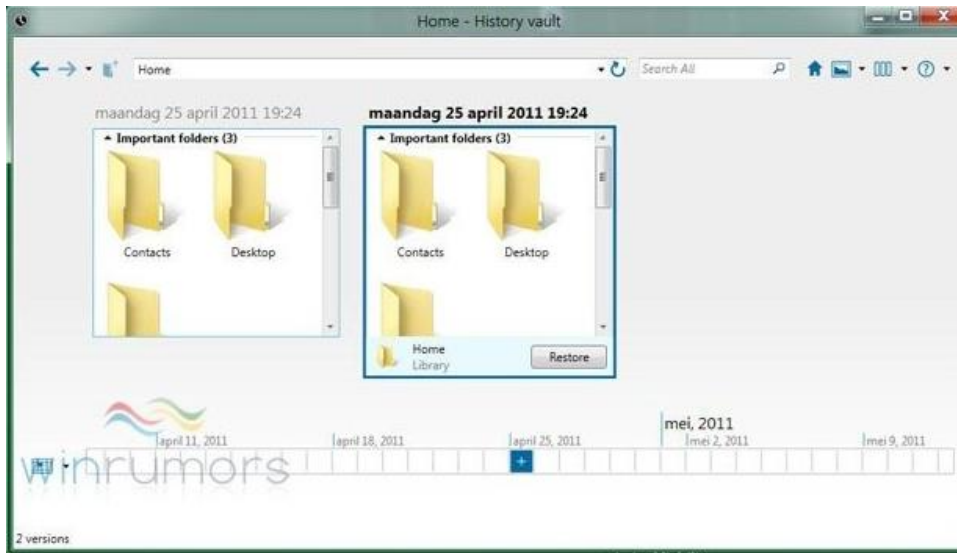
- **En parada** (*Off-line*)
  - ▣ La copia de seguridad se realiza en los periodos de tiempo en los que no se utilizan los datos del sistema.
- **En línea** (*On-line*):
  - ▣ La copia de seguridad se realiza mientras se utiliza el sistema.
  - ▣ Uso de técnicas que eviten problemas de consistencia:
    - *Snapshots*  
copia solo lectura del estado del sistema de ficheros.
    - *Copy-on-write*  
escrituras después de *snapshot* se realizan en copias.



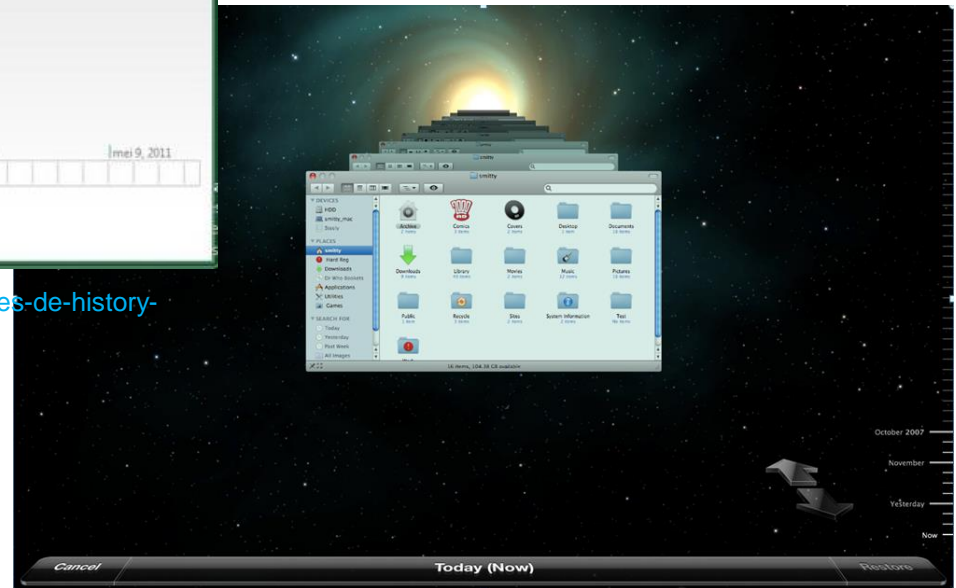
# Backup (copia de seguridad)

17

Alejandro Calderón Mateos 



<http://www.genbeta.com/sistemas-operativos/primeras-imagenes-de-history-vault-el-time-machine-de-windows-8>

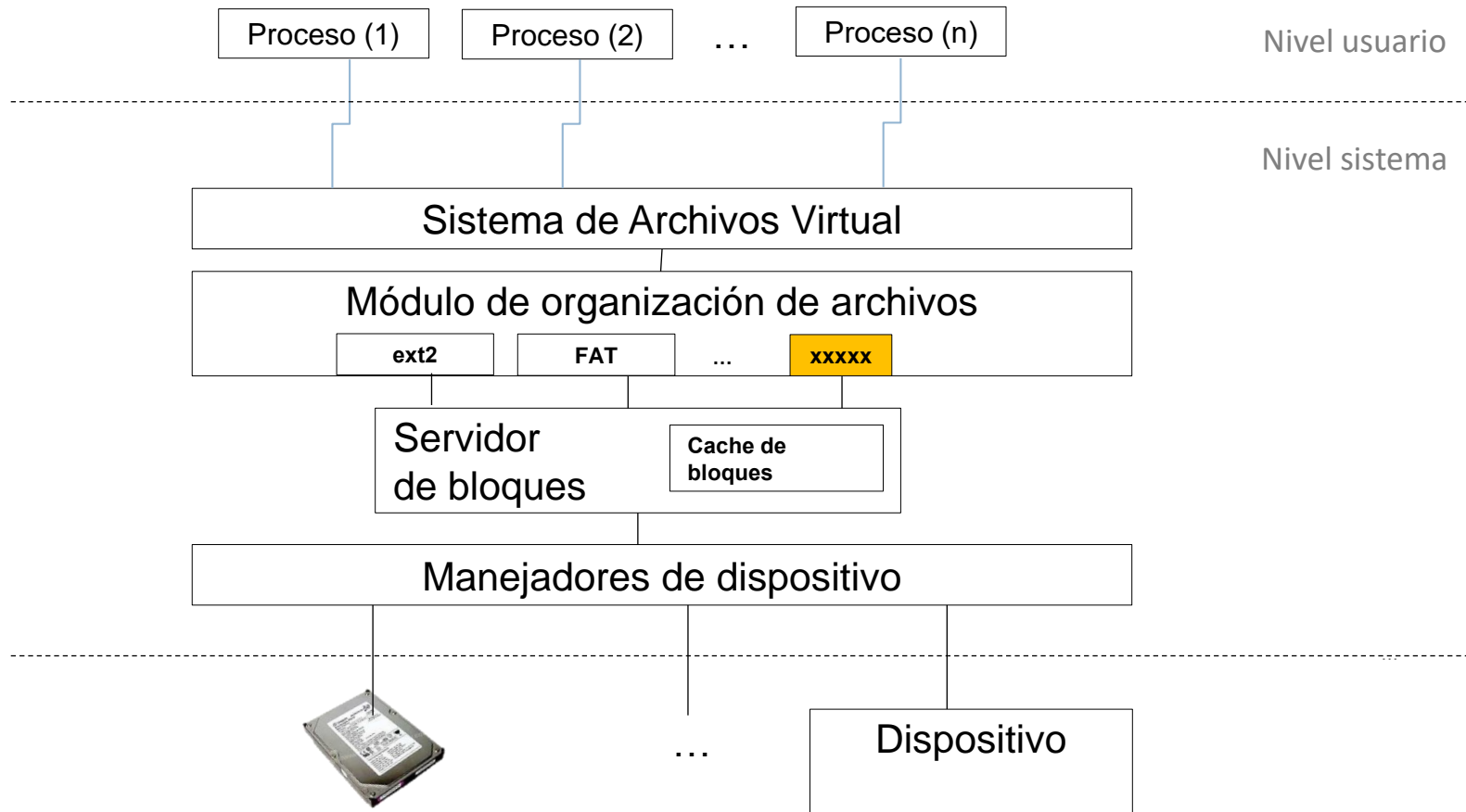


[http://www.reghardware.com/2007/11/08/review\\_leopard\\_pt2/page2.html](http://www.reghardware.com/2007/11/08/review_leopard_pt2/page2.html)

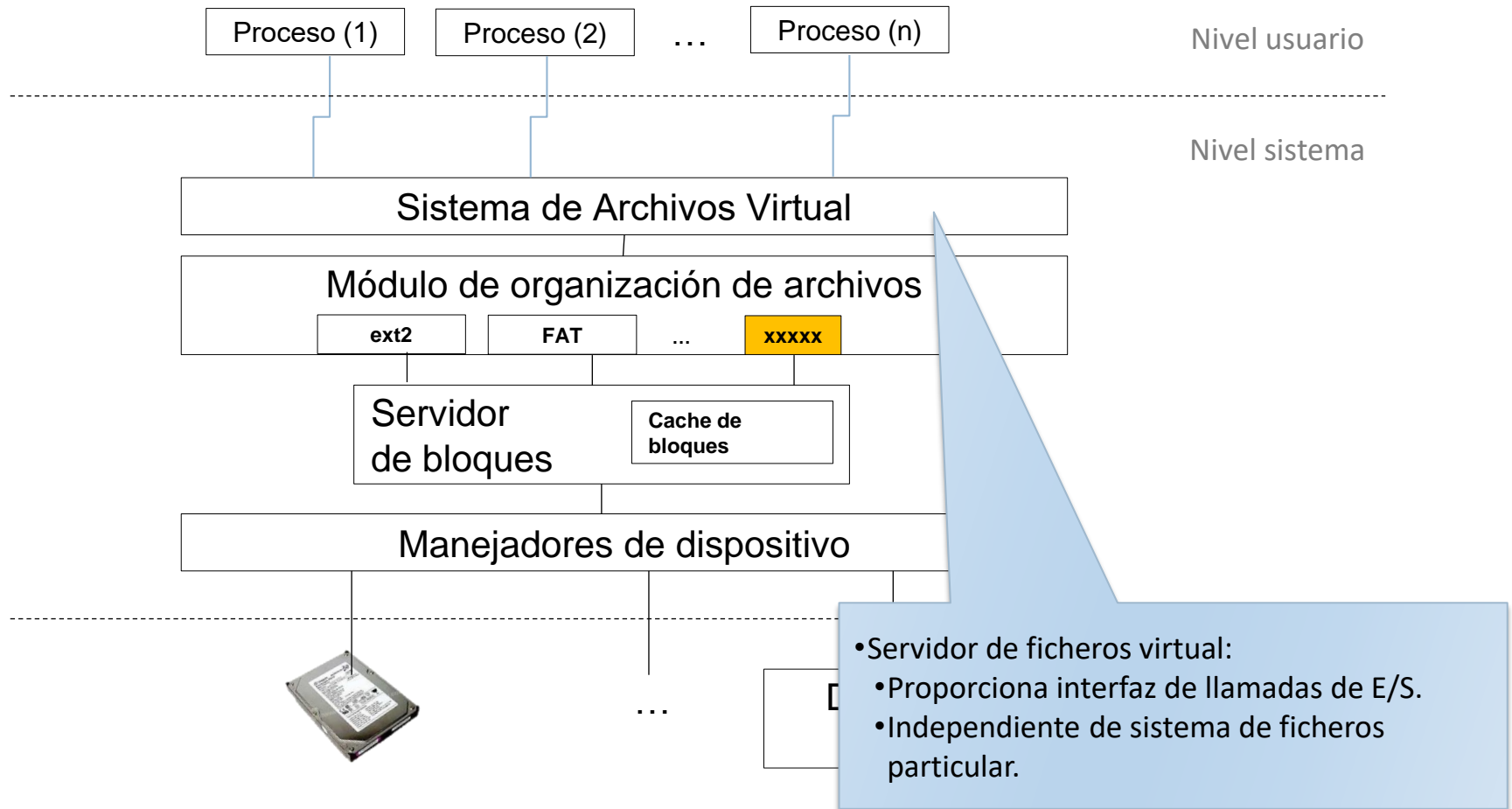
# Contenidos

- Introducción
- Fichero
- Directorio
- Sistema de ficheros
- Particiones/Volúmenes
- Dispositivos
- Software de sistema
- **Sistema de ficheros (gestor)**

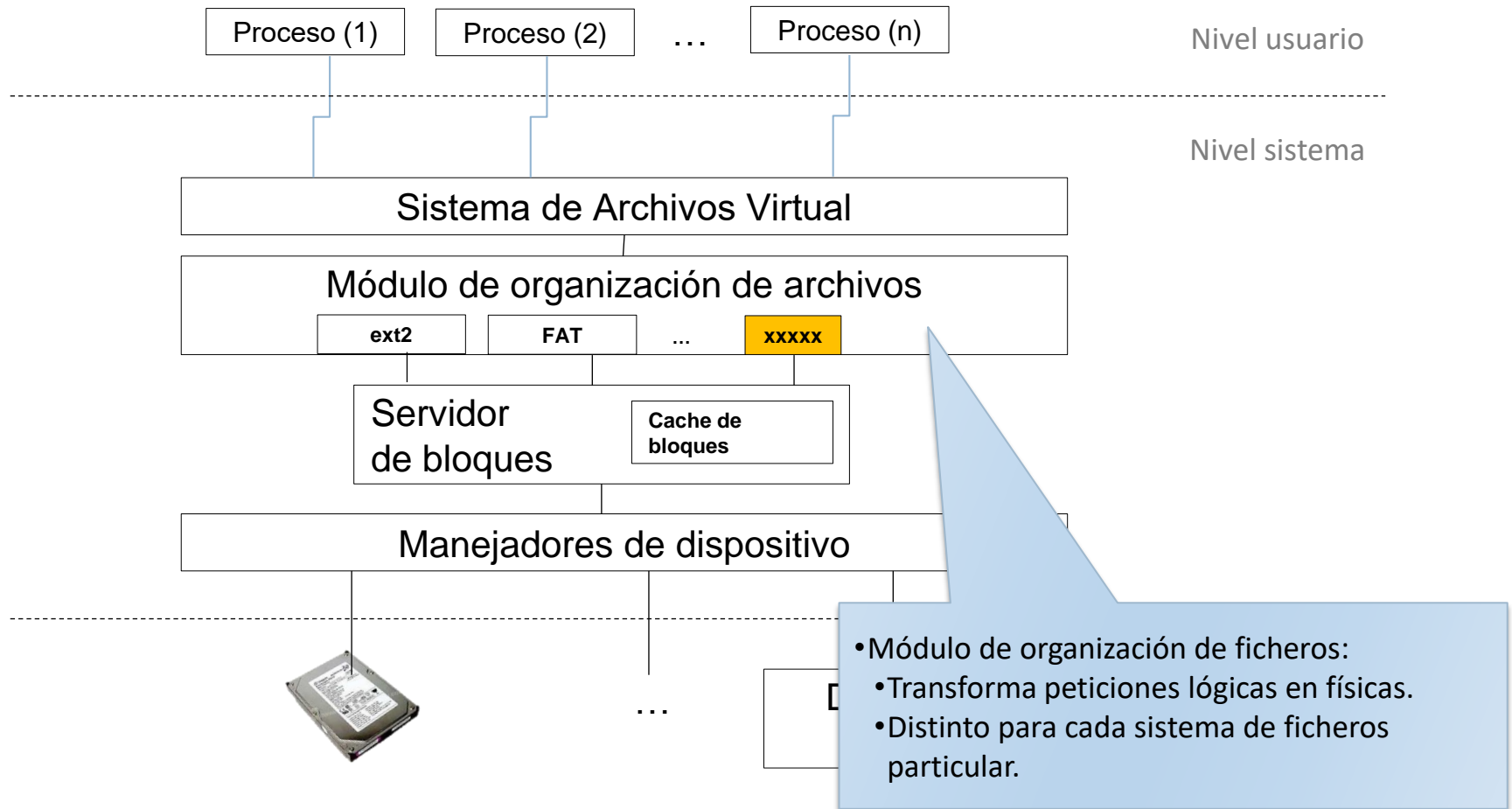
# Aspectos de partida (relativos a la arquitectura)...



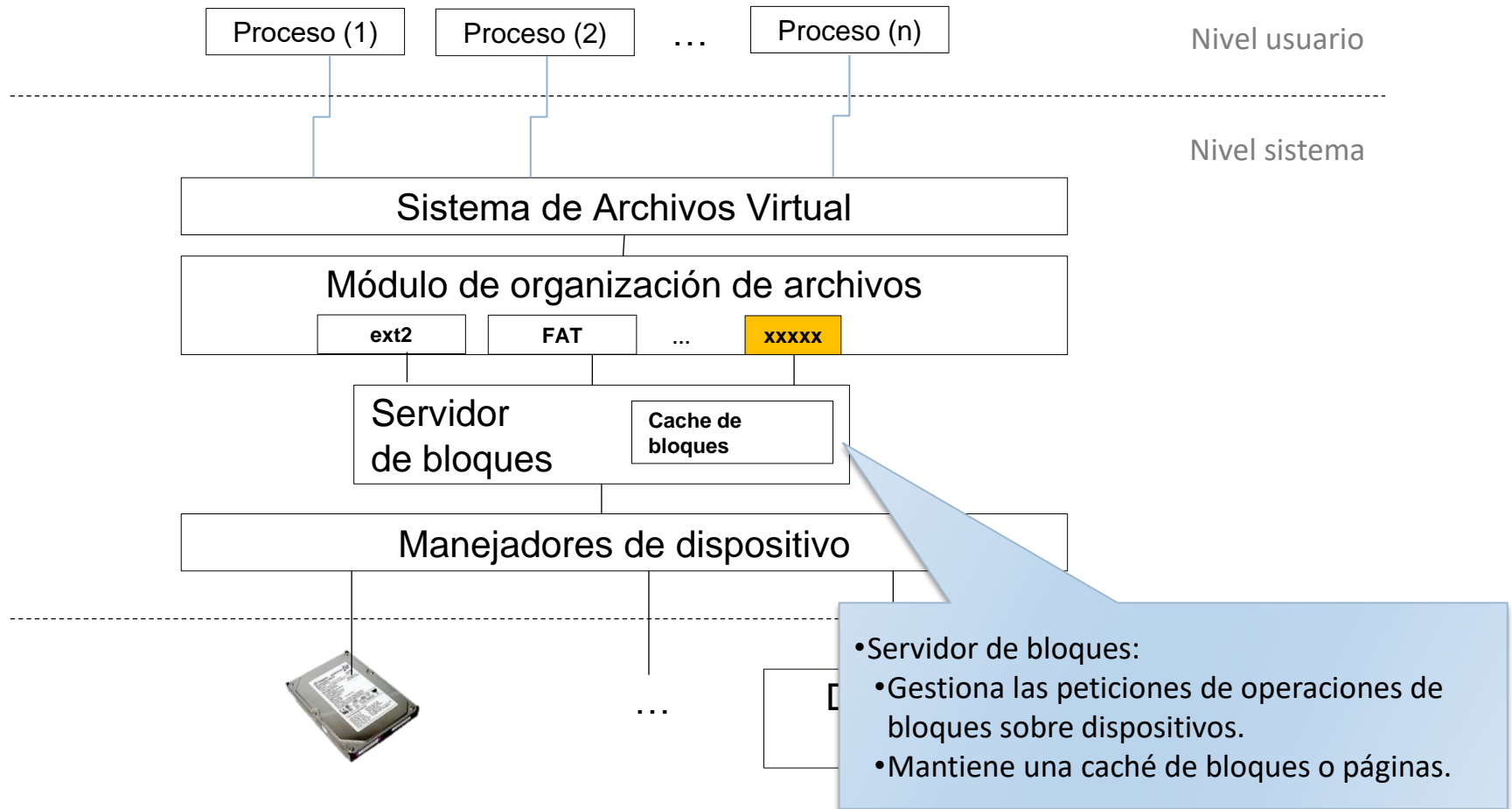
# Aspectos de partida (relativos a la arquitectura)...



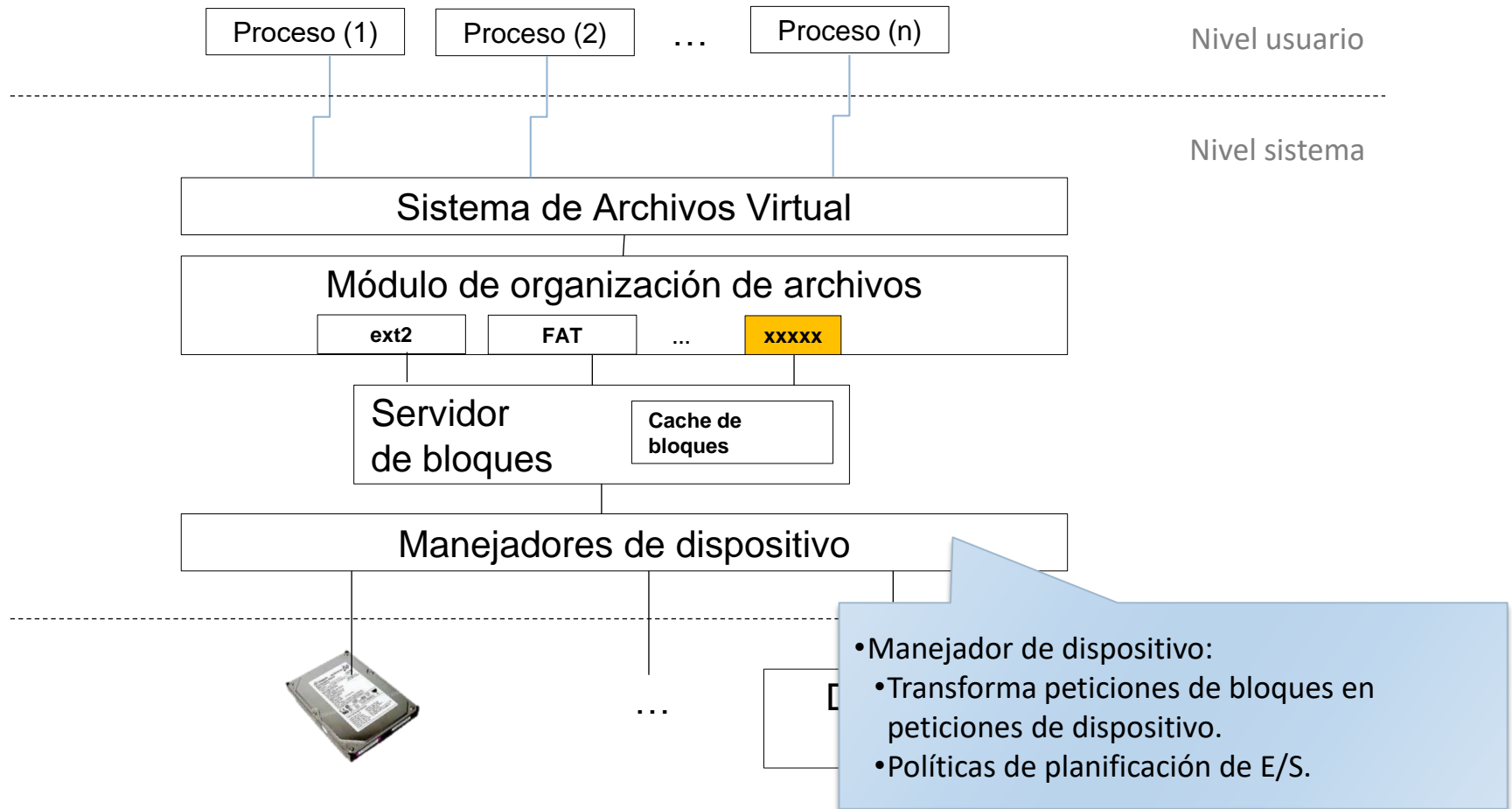
# Aspectos de partida (relativos a la arquitectura)...



# Aspectos de partida (relativos a la arquitectura)...

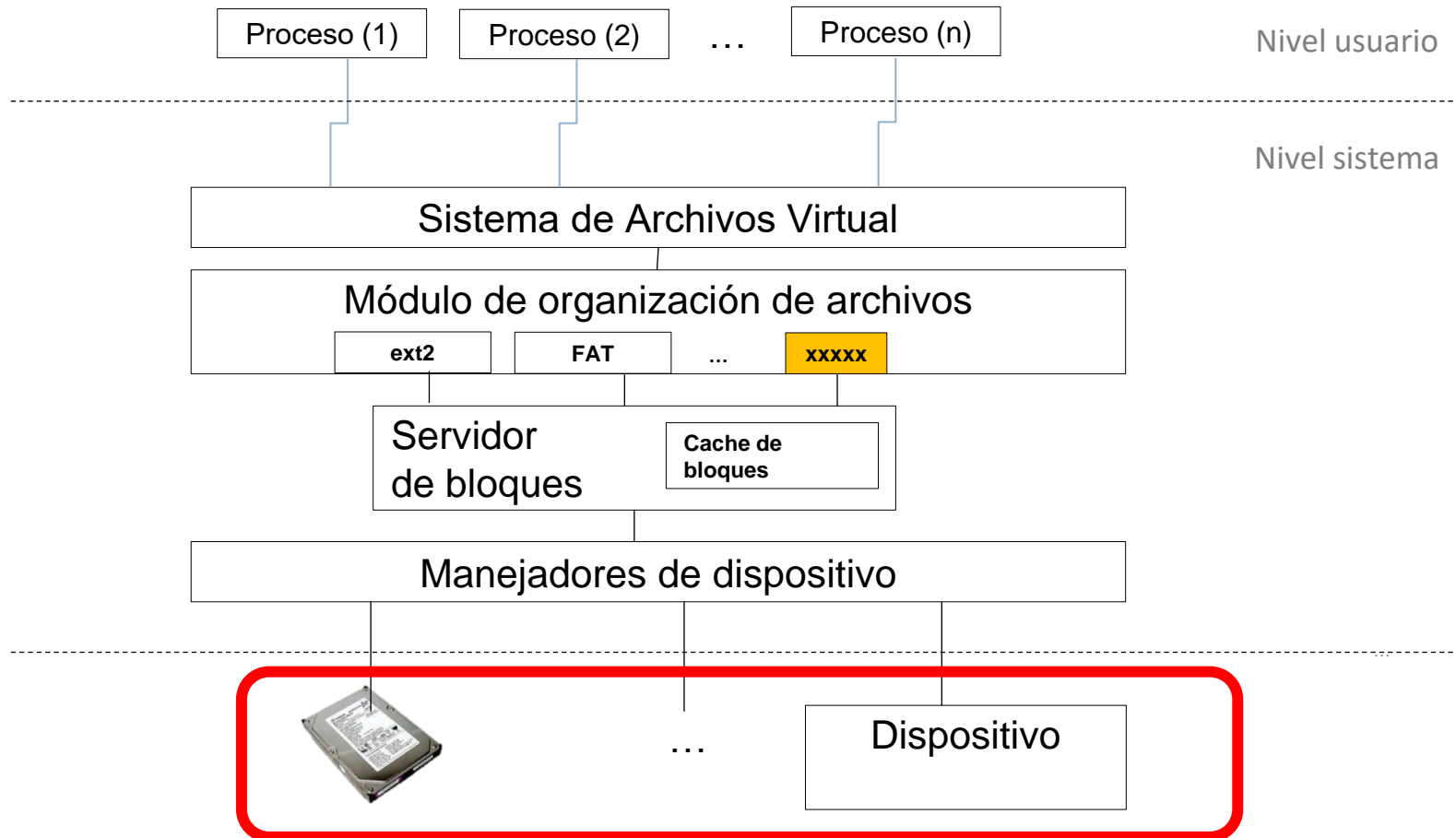


# Aspectos de partida (relativos a la arquitectura)...



# Aspectos de partida (relativos a la arquitectura)...

## a) bloques de disco



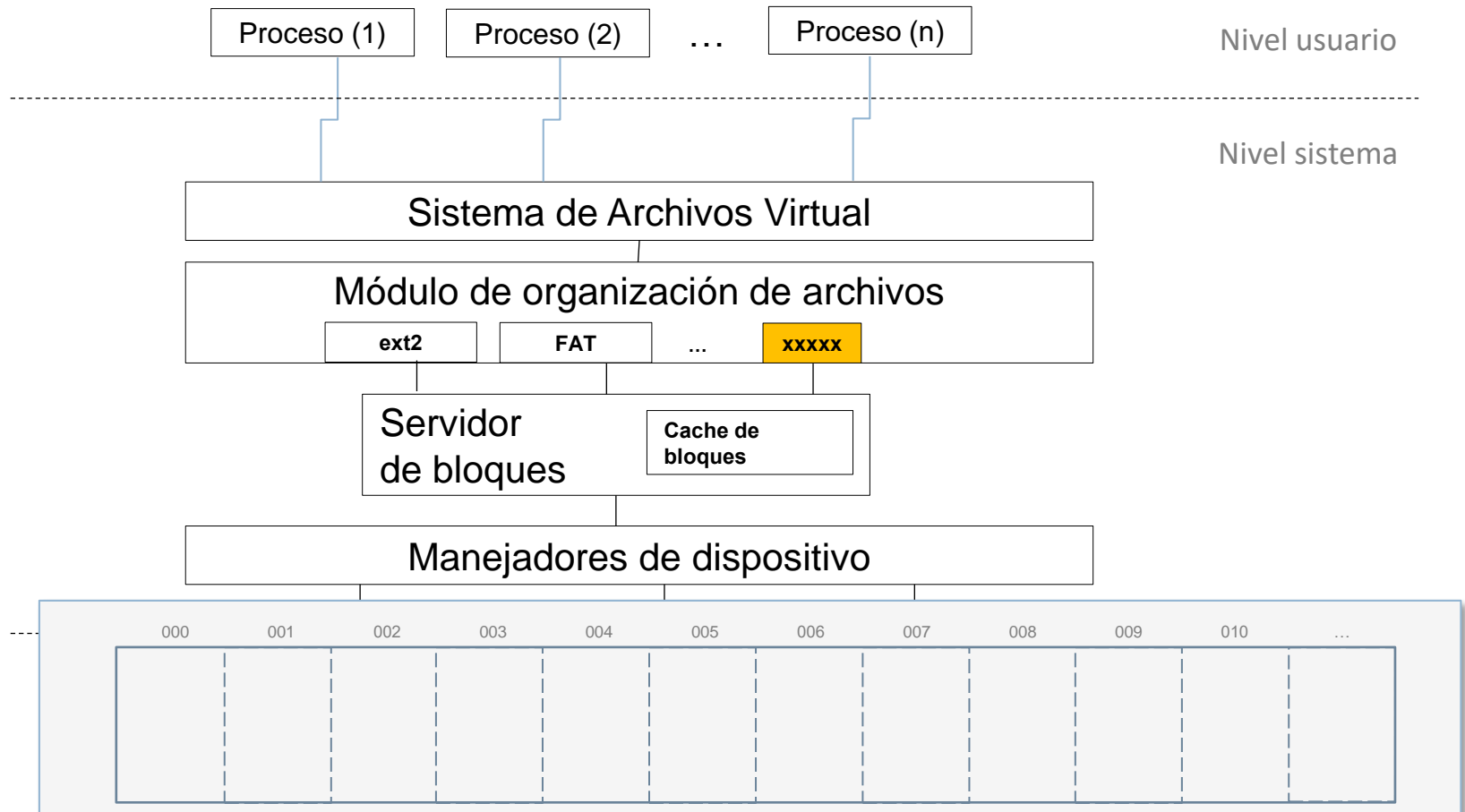


# Aspectos de partida (relativos a la arquitectura)...

## a) bloques de disco

25

Alejandro Calderón Mateos 

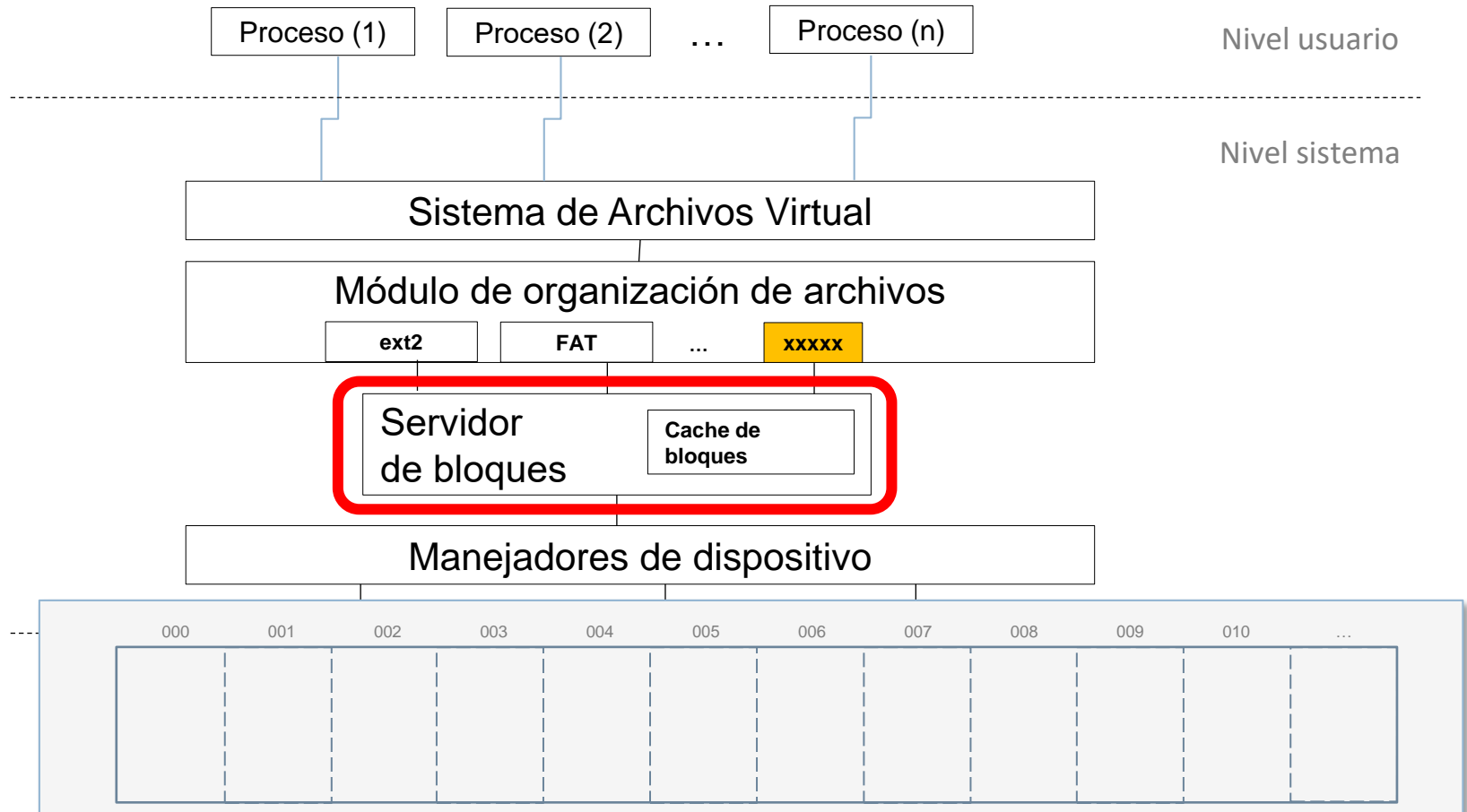


# Aspectos de partida (relativos a la arquitectura)...

## b) cache de bloques de disco

26

Alejandro Calderón Mateos 



# Aspectos de partida (relativos a la arquitectura)...

## b) cache de bloques de disco

27

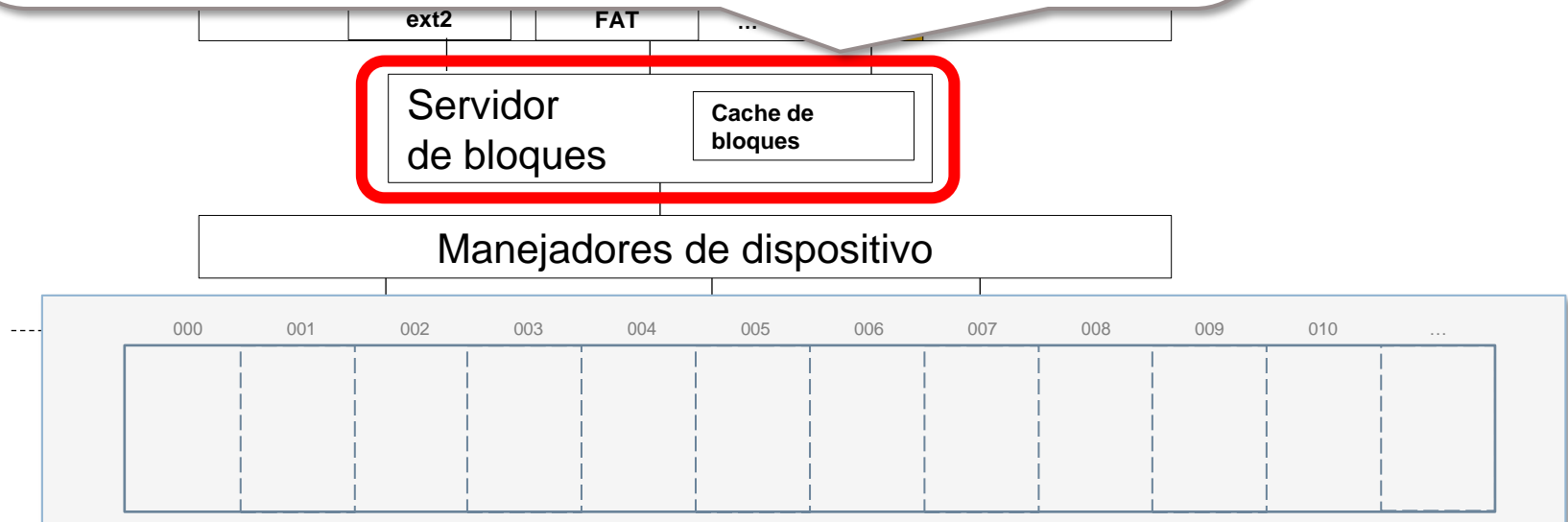
Alejandro Calderón Mateos



- ▶ **getblk:** busca/reserva en caché un bloque (a partir de un v-nodo, desplazamiento y tamaño dado).
- ▶ **brelse:** libera un bloque y lo pasa a la lista de libres.
- ▶ **bwrite:** escribe un bloque de la caché a disco.
- ▶ **bread:** lee un bloque de disco a caché.
- ▶ **breada:** lee un bloque (y el siguiente) de disco a caché.

Nivel usuario

Nivel sistema



# Servidor de bloques

- Se encarga de:
  - Emitir los mandatos genéricos para leer y escribir bloques a los manejadores de dispositivo (usando las rutinas específicas de cada dispositivo).
  - Optimizar las peticiones de E/S.
    - Ej.: cache de bloques.
    - Puede estar integrado con el gestor de memoria virtual.
  - Ofrecer un nombrado lógico para los dispositivos.
    - Ej.: /dev/hd3 (tercera partición del primer disco)

# Servidor de bloques

- Funcionamiento general:
  - ▣ Si el bloque está en la cache
    - Copiar el contenido (+ actualizar metadatos de uso del bloque)
  - ▣ Si no está en la caché
    - Leer el bloque del dispositivo y guardarlo en la cache
    - Copiar el contenido (y actualizar los metadatos)
    - Si el bloque ha sido escrito (sucio / *dirty*)
      - Política de escritura
    - Si la cache está llena, es necesario hacer hueco
      - Política de reemplazo

# Servidor de bloques

## □ Funcionamiento general:

- **Lectura adelantada** (*read-ahead*):
    - Leer un número de bloques a continuación del requerido y se almacena en caché (mejora el rendimiento en accesos consecutivos)
- 
- Leer el bloque del dispositivo y guardarlo en la cache
  - Copiar el contenido (y actualizar los metadatos)
  - Si el bloque ha sido escrito (sucio / *dirty*)
    - Política de escritura
  - Si la cache está llena, es necesario hacer hueco
    - Política de reemplazo

# Servidor de bloques

- **Escritura inmediata** (*write-through*):
  - Se escribe cada vez que se modifica el bloque (– rendimiento, + fiabilidad)
- **Escritura diferida** (*write-back*):
  - Sólo se escriben los datos a disco cuando se eligen para su reemplazo por falta de espacio en la cache (+ rendimiento, –fiabilidad)
- **Escritura retrasada** (*delayed-write*):
  - Escribir a disco los bloques de datos modificados en la cache de forma periódica cada cierto tiempo (30 segundos en UNIX) (compromiso entre anteriores)
- **Escritura al cierre** (*write-on-close*):
  - Cuando se cierra un archivo, se vuelcan al disco los bloques del mismo.

■ Si el bloque no ha sido escrito (sucio / *dirty*)

■ Política de escritura

■ Si la cache está llena, es necesario hacer hueco

■ Política de reemplazo

# Servidor de bloques

- Funcionamiento general:
  - ▣ Si el bloque está en la cache
    - Copiar el contenido (+ actualizar metadatos de uso del bloque)
  - ▣ Si no está en la caché
    - Leer el bloque del dispositivo y guardarlo en la cache

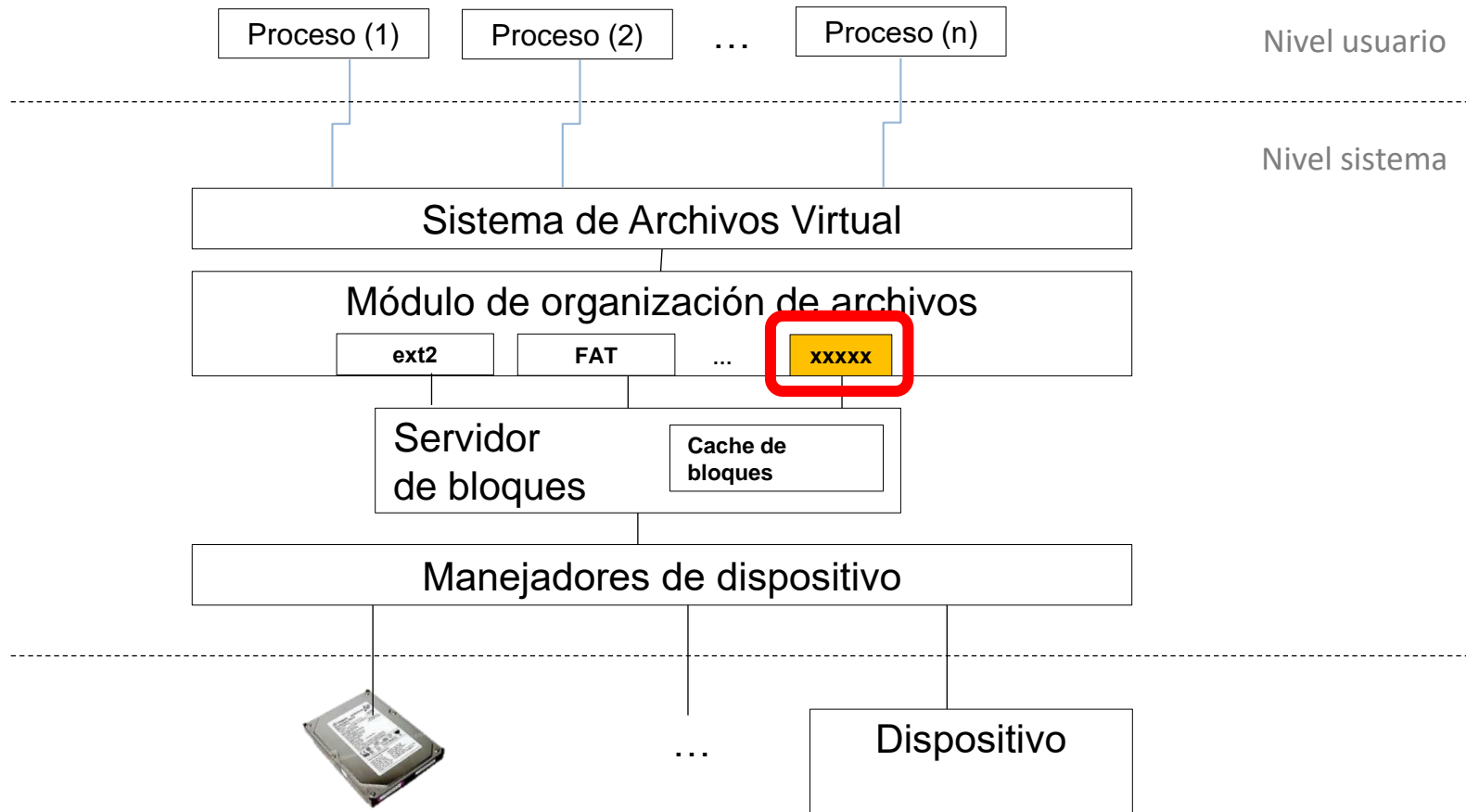
- **FIFO** (*First in First Out*)
- **Algoritmo del reloj** (*Segunda oportunidad*)
- **MRU** (*Most Recently Used*)
- **LRU** (*Least Recently Used*) <- + frecuentemente usada

■ Si el bloque no está en la cache, es necesario hacer un nuevo

■ Política de reemplazo

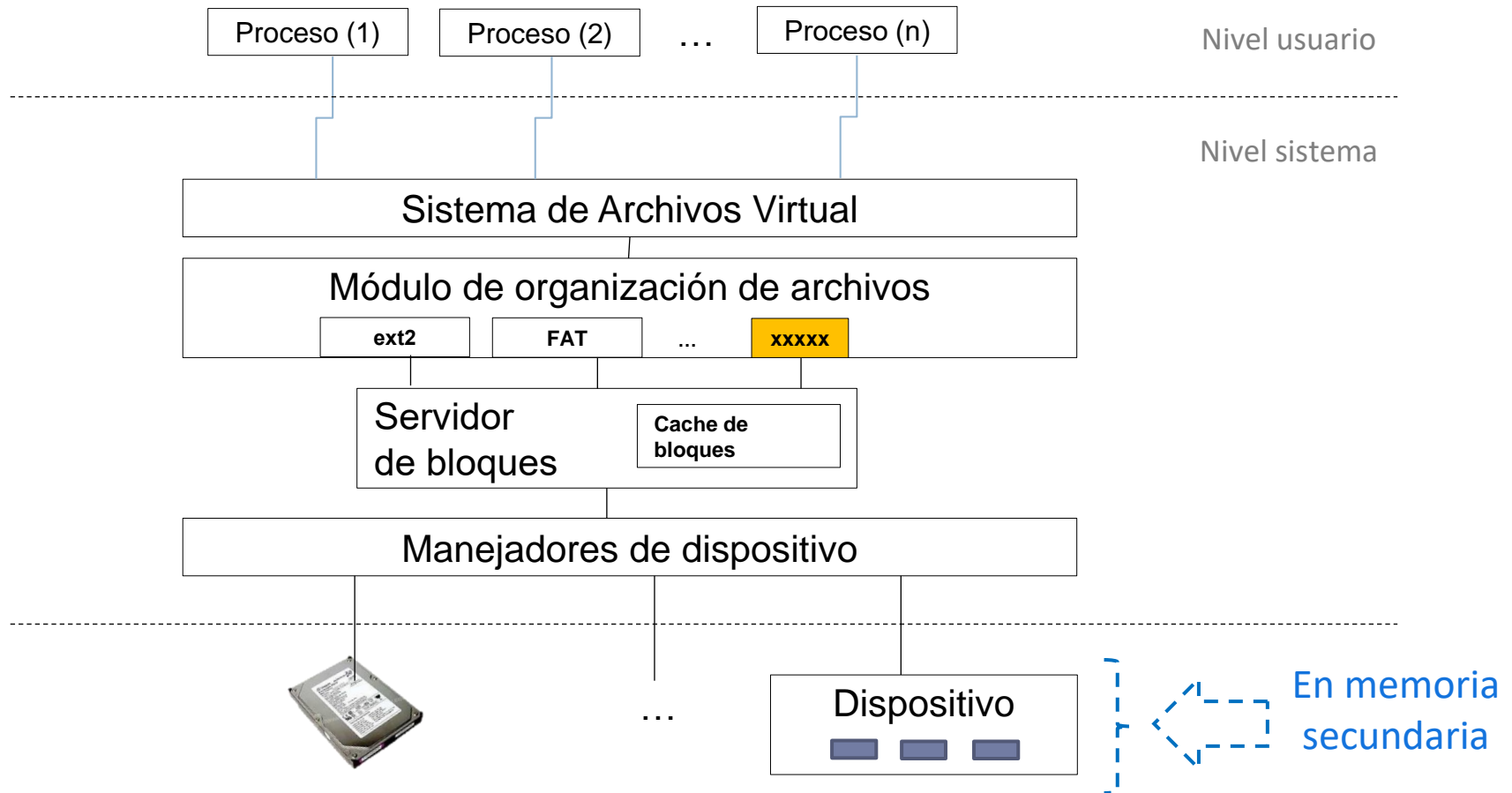


# Aspectos a desarrollar (relativos a la arquitectura)...



# Aspectos a desarrollar (relativos a la arquitectura)...

## (1) Estructuras de datos en disco...

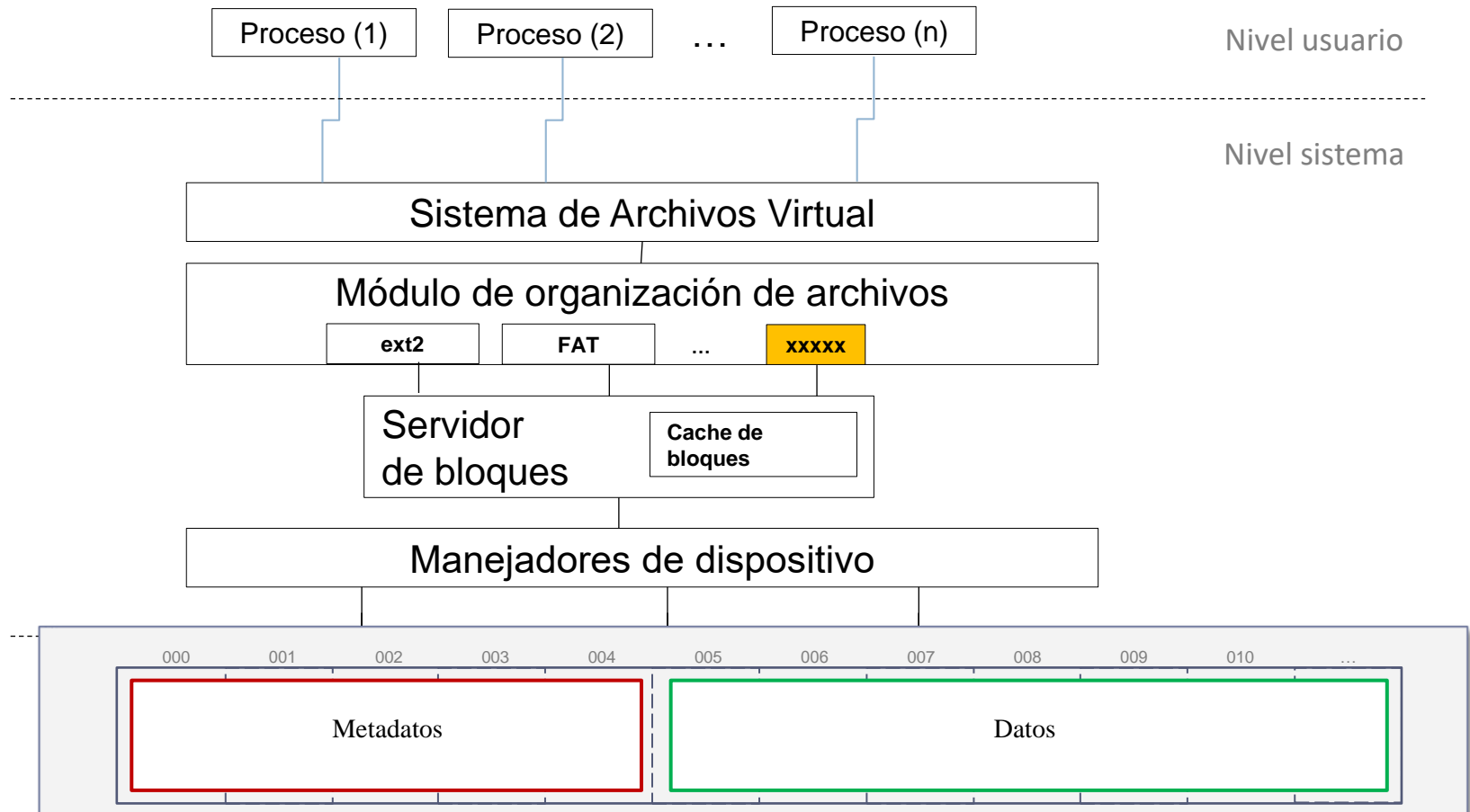


# Aspectos a desarrollar (relativos a la arquitectura)...

## (1) Estructuras de datos en disco...

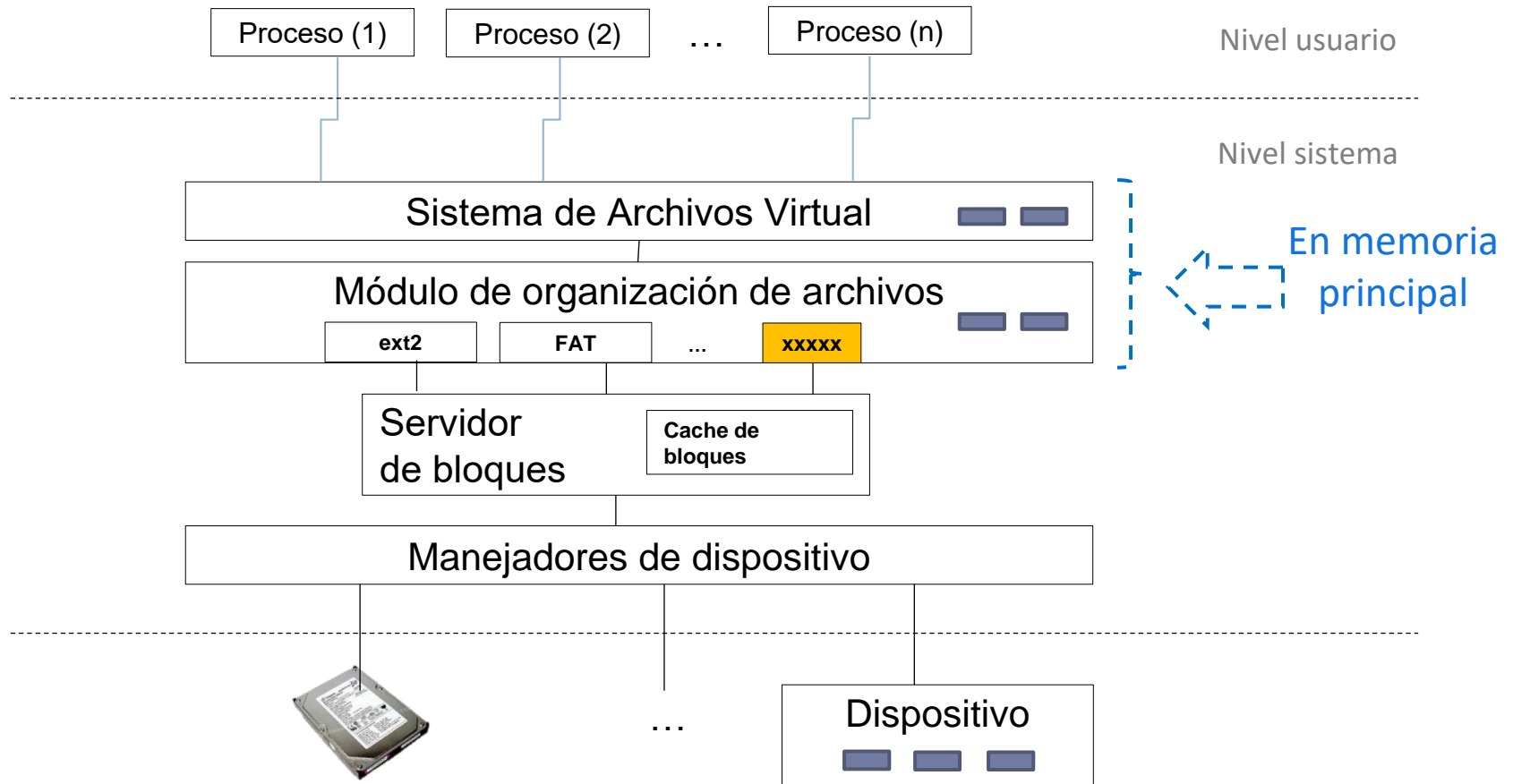
35

Alejandro Calderón Mateos 



# Aspectos a desarrollar (relativos a la arquitectura)...

## (2) Estructuras de datos en memoria...



# Aspectos a desarrollar (relativos a la arquitectura)...

## (3a) Gestión estructuras disco/memoria...

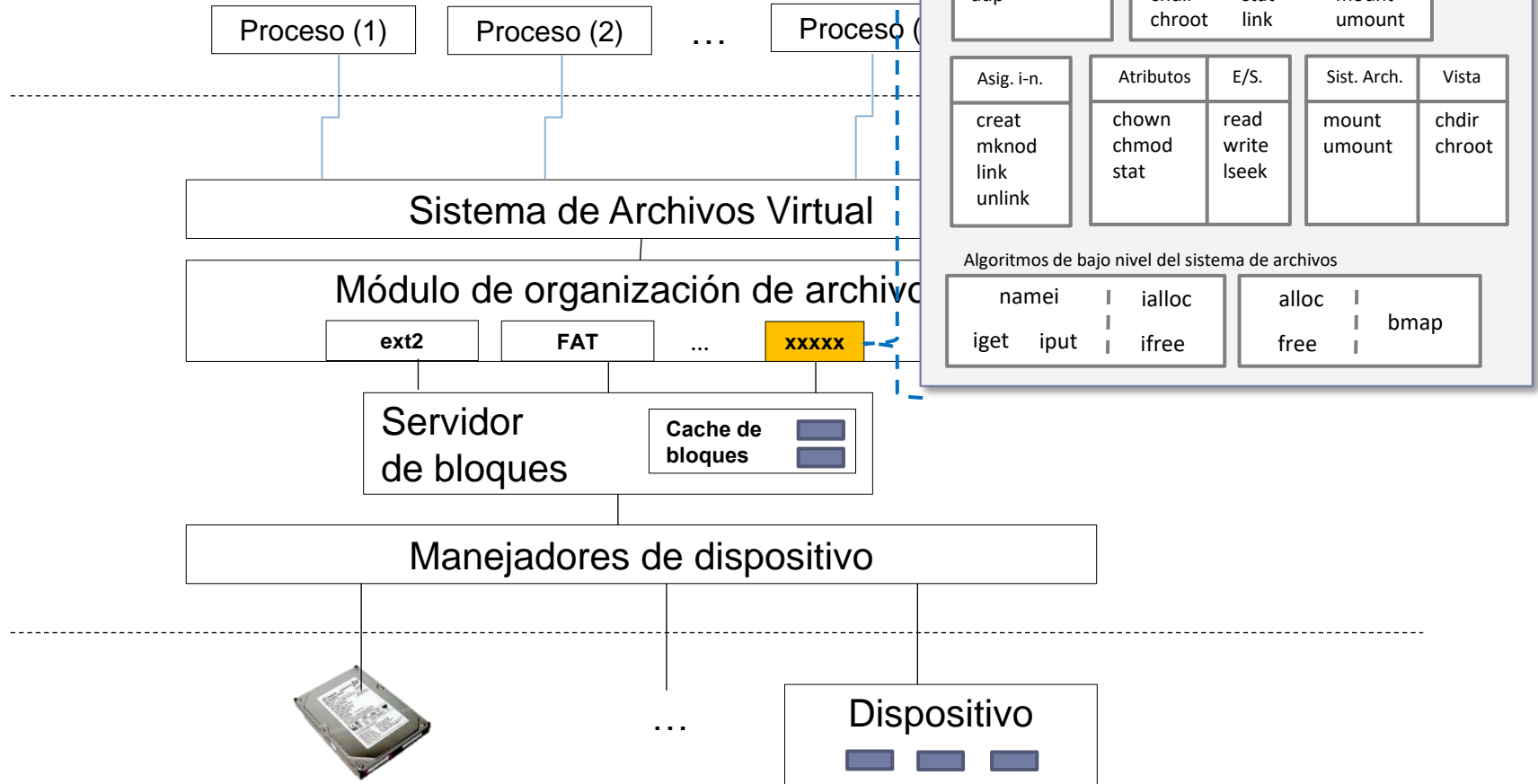


# Aspectos a desarrollar (relativos a la arquitectura)...

## (3b) Llamadas al sistema...

38

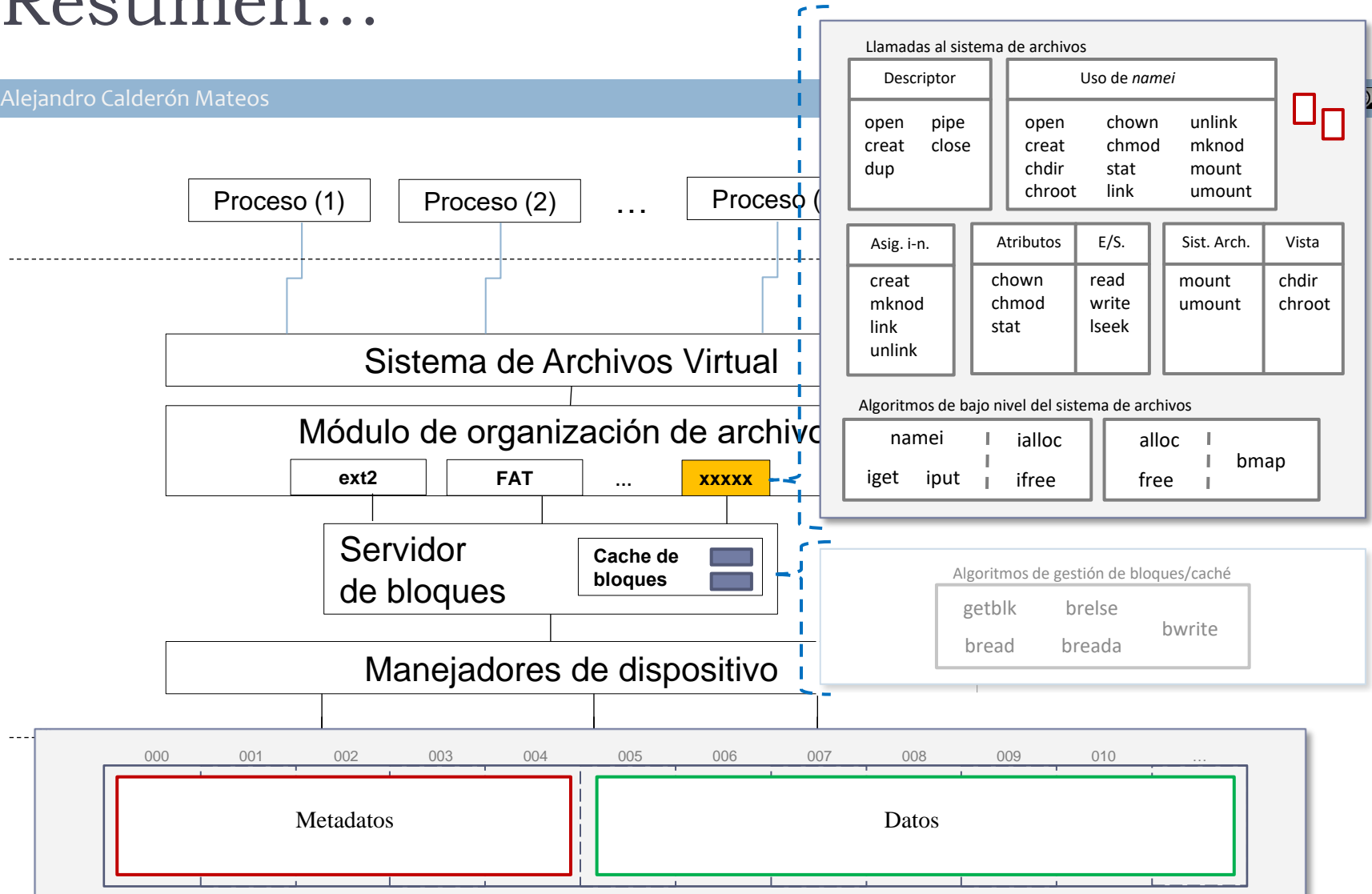
Alejandro Calderón Mateos



# Resumen...

39

Alejandro Calderón Mateos



# Resumen simplificado...

40

[http://www.ual.es/~acorral/DSO/Tema\\_4.pdf](http://www.ual.es/~acorral/DSO/Tema_4.pdf)

Alejandro Calderón Mateos 

## Llamadas al sistema de archivos

Descriptor	Uso de <i>namei</i>	Asig. i-n.	Atributos	E/S.	Sist. Arch.	Vista
open pipe creat close dup	open chown unlink creat chmod mknod chdir stat mount chroot link umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

xx

## Algoritmos de bajo nivel del sistema de archivos

namei	ialloc	alloc	
iget iput	ifree	free	bmap

d-entradas

montajes

punteros de posición

ficheros abiertos

i-nodos en uso

módulos de s. ficheros

## Algoritmos de gestión de bloques/caché

getblk	brelse	bread	breada	bwrite
--------	--------	-------	--------	--------

000	001	002	003	004	005	006	007	008	009	010	...
Bloque de arranque	Super-bloque	Asignación de recursos	000 001 002 003 004	i-nodos							



# (1) Estructuras de datos en disco...

41

[http://www.ual.es/~acorral/DSO/Tema\\_4.pdf](http://www.ual.es/~acorral/DSO/Tema_4.pdf)

Alejandro Calderón Mateos 

Llamadas al sistema de archivos

Descriptor	Uso de <i>namei</i>	Asig. i-n.	Atributos	E/S.	Sist. Arch.	Vista
open pipe creat close dup	open chown unlink creat chmod mknod chdir stat mount chroot link umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

xx

Algoritmos de bajo nivel del sistema de archivos

namei	ialloc	alloc	
iget iput	ifree	free	bmap

d-entradas

montajes

punteros de posición

ficheros abiertos

i-nodos en uso

módulos de s. ficheros

Algoritmos de gestión de bloques/caché

getblk	brelse	bread	breada	bwrite
--------	--------	-------	--------	--------

000	001	002	003	004	005	006	007	008	009	010	...
Bloque de arranque	Super-bloque	Asignación de recursos	000 001 002 003 004	i-nodos							

ARCOS @ UC3M

Sistemas Operativos – Ficheros, directorios y sistemas de ficheros

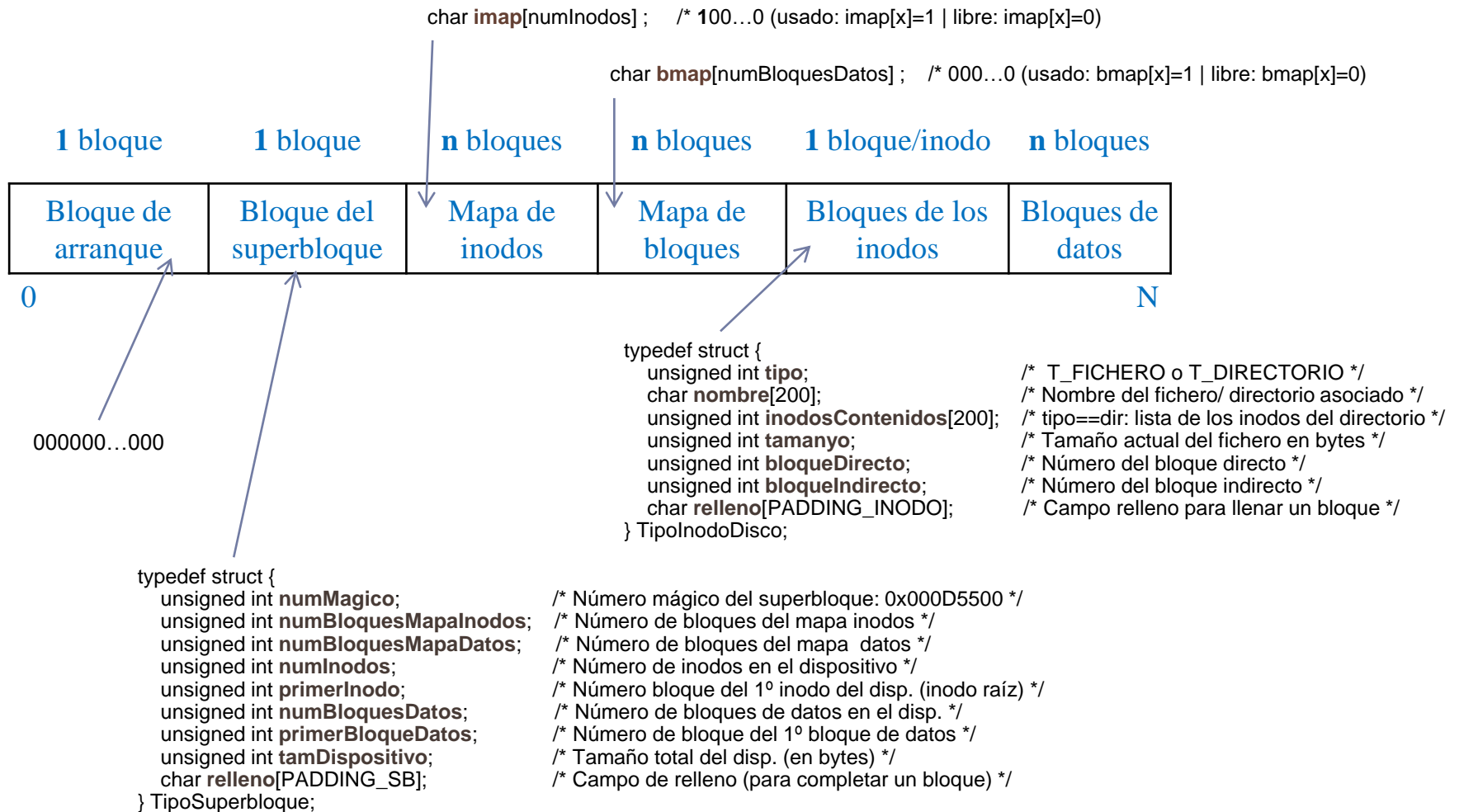
# Ejemplo de organización en disco

<https://github.com/acaldero/nanofs>



42

Alejandro Calderón Mateos



ARCOS @ UC3M

Sistemas Operativos – Ficheros, directorios y sistemas de ficheros

## (2) Estructuras de datos en memoria...

43

[http://www.ual.es/~acorral/DSO/Tema\\_4.pdf](http://www.ual.es/~acorral/DSO/Tema_4.pdf)

Alejandro Calderón Mateos 

Llamadas al sistema de archivos

Descriptor	Uso de <i>namei</i>	Asig. i-n.	Atributos	E/S.	Sist. Arch.	Vista
open pipe creat close dup	open chown unlink creat chmod mknod chdir stat mount chroot link umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

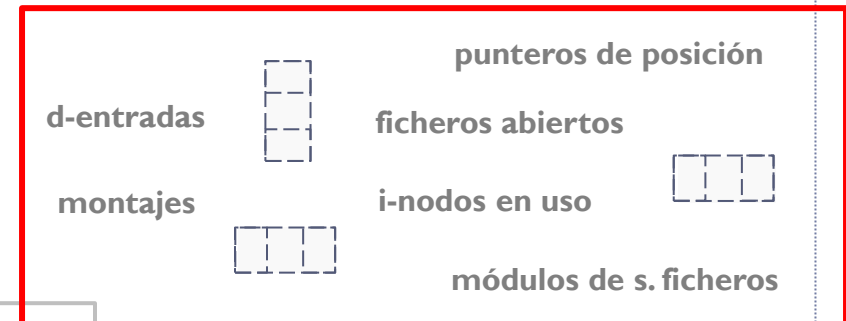
XX

Algoritmos de bajo nivel del sistema de archivos

namei	ialloc	alloc	
iget	iput	free	bmap

Algoritmos de gestión de bloques/caché

getblk	brelse	bread	breada	bwrite
--------	--------	-------	--------	--------

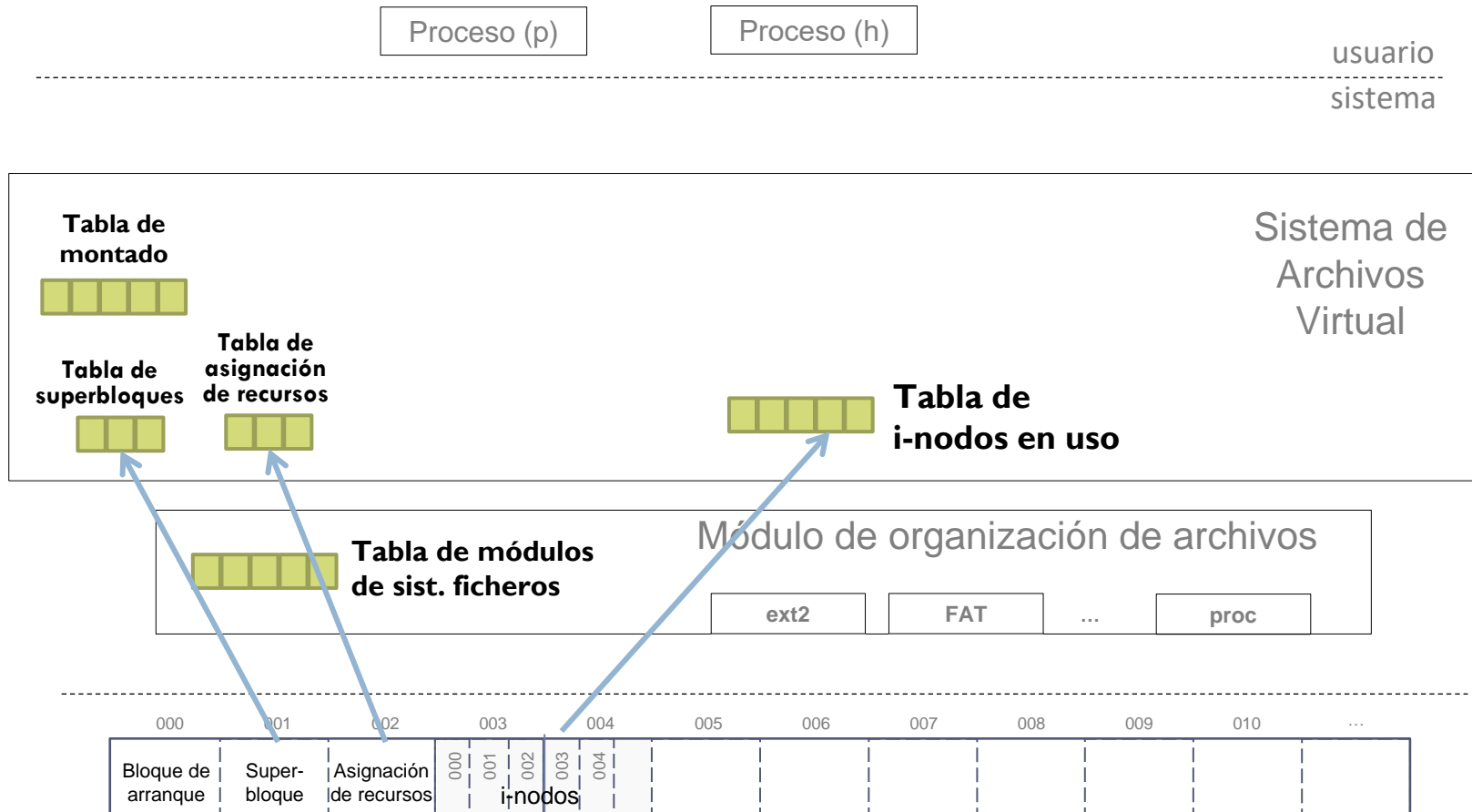


000	001	002	003	004	005	006	007	008	009	010	...
Bloque de arranque	Super-bloque	Asignación de recursos	000	001	002	003	004				
			i-nodos								

# Estructuras principales de gestión principales metadatos en disco

44

Alejandro Calderón Mateos 

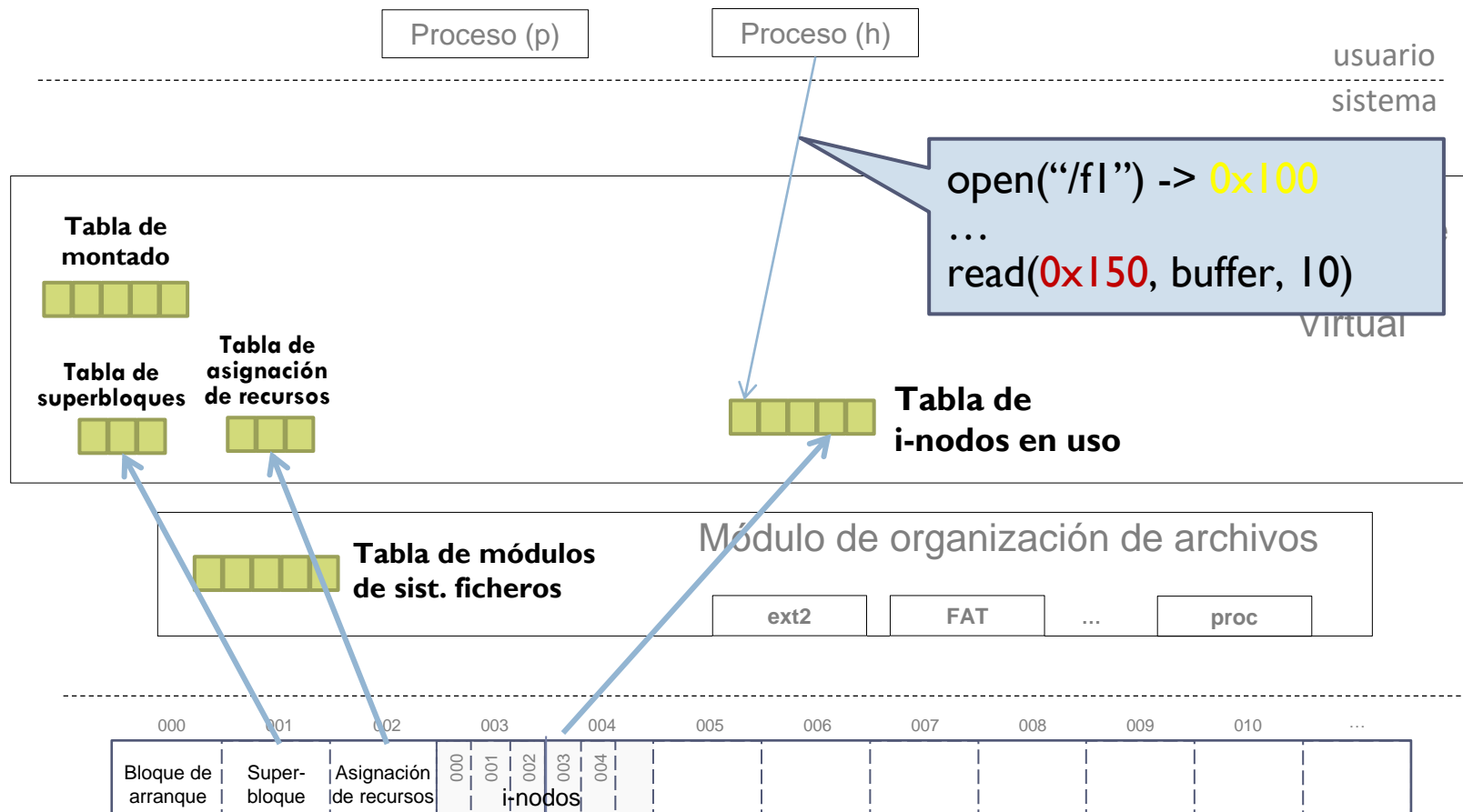


# Estructuras principales de gestión

## ¿interfaz segura para API?

45

Alejandro Calderón Mateos

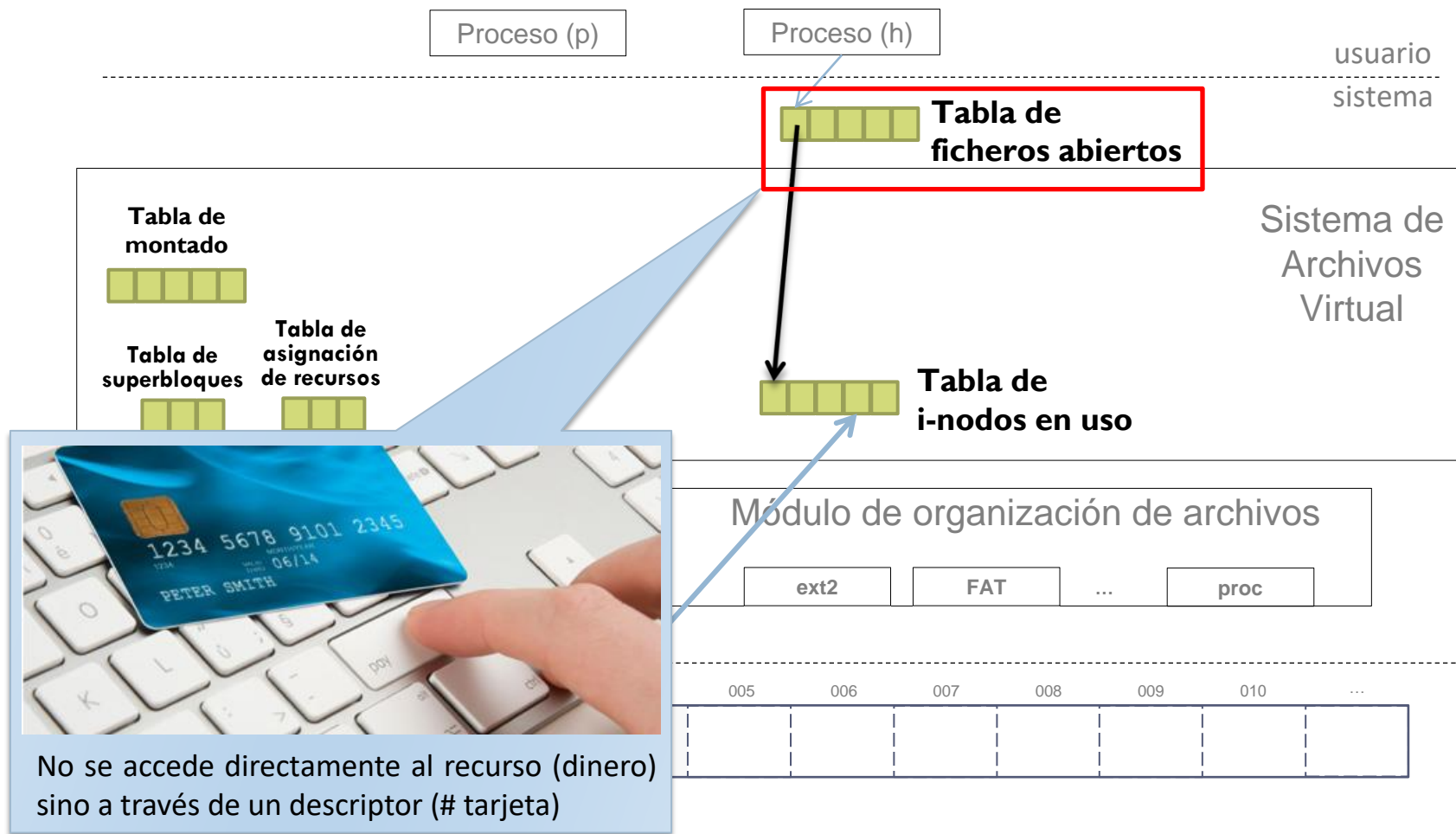


# Estructuras principales de gestión

## tabla de ficheros abiertos: interfaz segura

46

Alejandro Calderón Mateos 



ARCOS @ UC3M

Sistemas Operativos – Ficheros, directorios y sistemas de ficheros

# Estructuras principales de gestión

## tabla de ficheros abiertos: interfaz segura

47

Alejandro Calderón Mateos



**Tabla de archivos  
abiertos P1**

fd → 0	23
1	4563
2	56
3	3
4	678

**Tabla de archivos  
abiertos P2**

fd → 0	230
1	563
2	98
3	3
4	247

**Tabla de archivos  
abiertos P3**

fd → 0	2300
1	53
2	4
3	3465
4	347

Tabla de nodos-i		
...		
...		

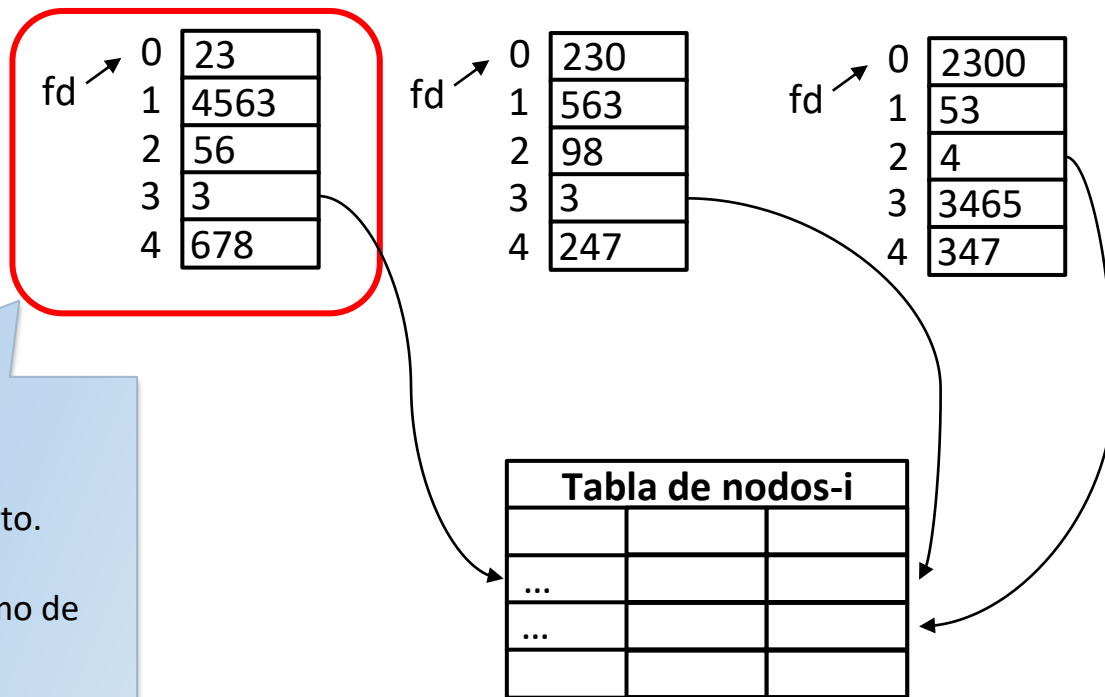


No se accede directamente al recurso (dinero)  
sino a través de un descriptor (# tarjeta)

# Estructuras principales de gestión

## tabla de ficheros abiertos: interfaz segura

**Tabla de archivos  
abiertos P1**



fd → 0	23
1	4563
2	56
3	3
4	678

**Tabla de archivos  
abiertos P2**

fd → 0	230
1	563
2	98
3	3
4	247

**Tabla de archivos  
abiertos P3**

fd → 0	2300
1	53
2	4
3	3465
4	347

**Tabla de nodos-i**

...		
...		

- Tabla incluida en el BCP del proceso.
  - Cuando se hace *fork()* se duplica.
- Tabla con una entrada por fichero abierto.
  - 0, 1 y 2 usados por defecto.
- Número de filas limita el número máximo de ficheros abiertos por proceso.
- La tabla se rellena de forma ordenada:
  - *open/creat/dup*: busca la 1ª entrada libre.
  - *close*: marca entrada como libre.

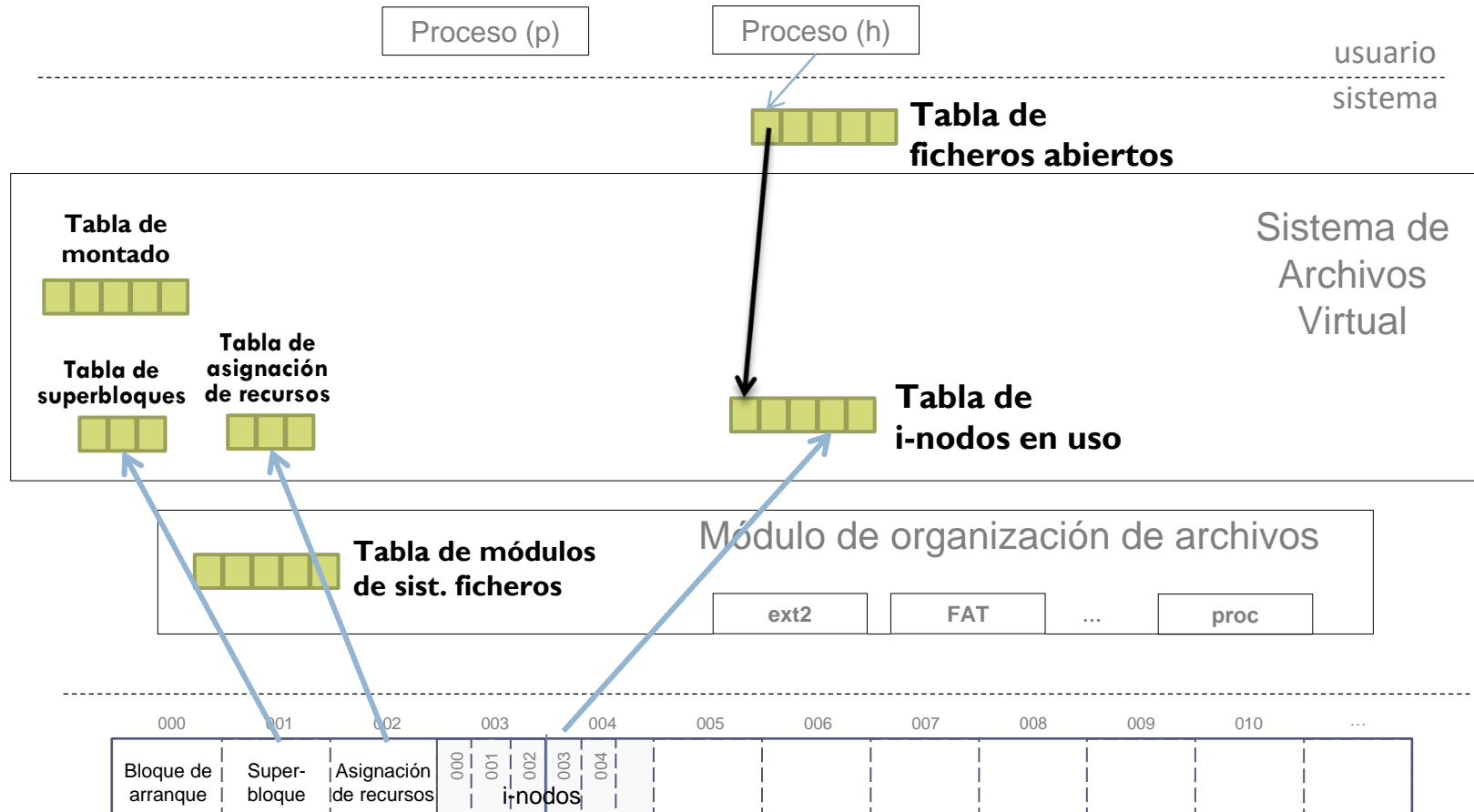


# Estructuras principales de gestión

## tabla de ficheros abiertos: interfaz segura

49

Alejandro Calderón Mateos 

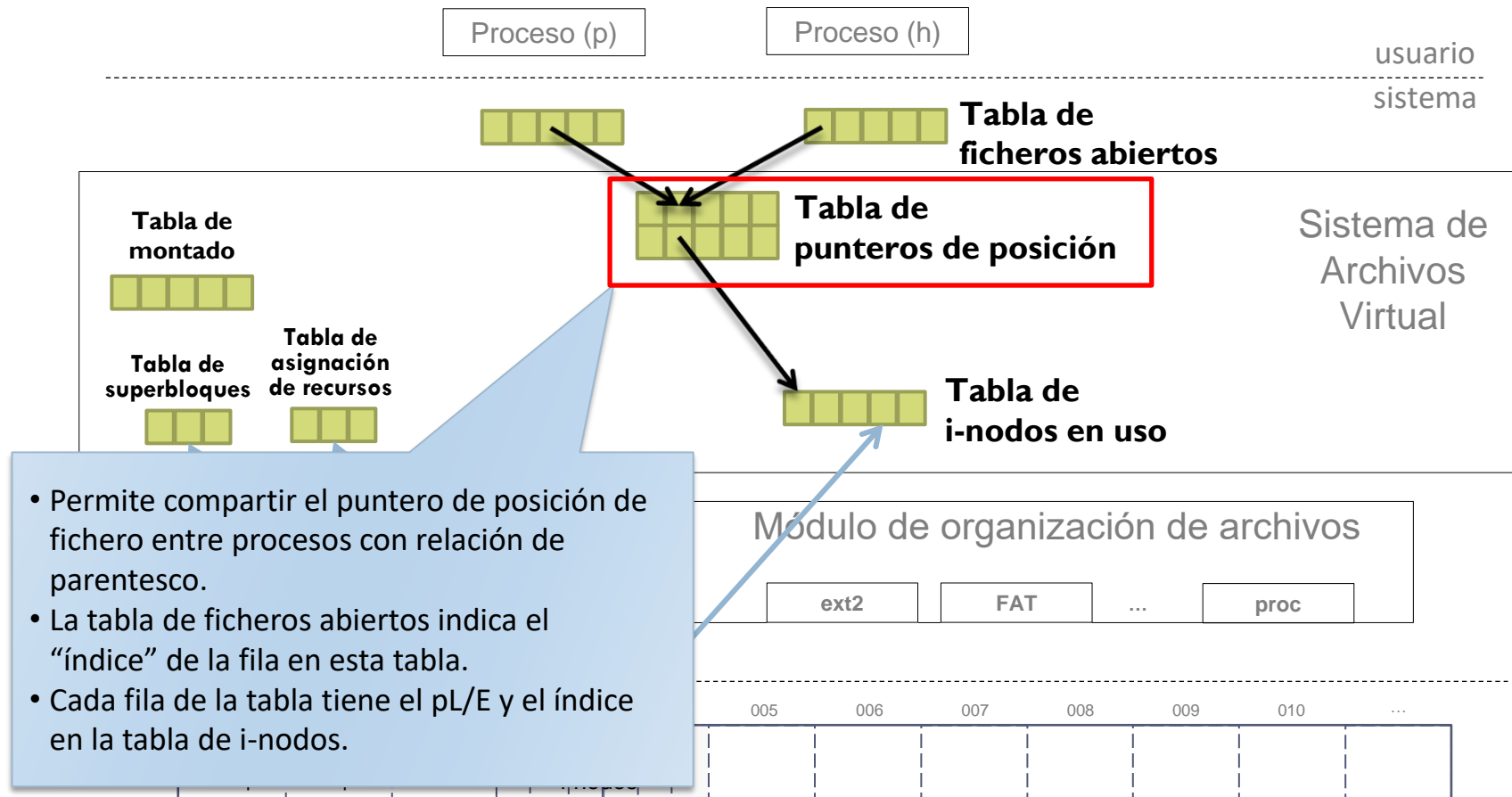


# Estructuras principales de gestión

## tabla de punteros de posición: compartición de pL/E

50

Alejandro Calderón Mateos



# Estructuras principales de gestión

## tabla de punteros de posición: compartición de pL/E

51

Sistemas operativos: una visión aplicada (© J. Carrete et al.)

Alejandro Calderón Mateos



**Tabla de archivos  
abiertos P1**

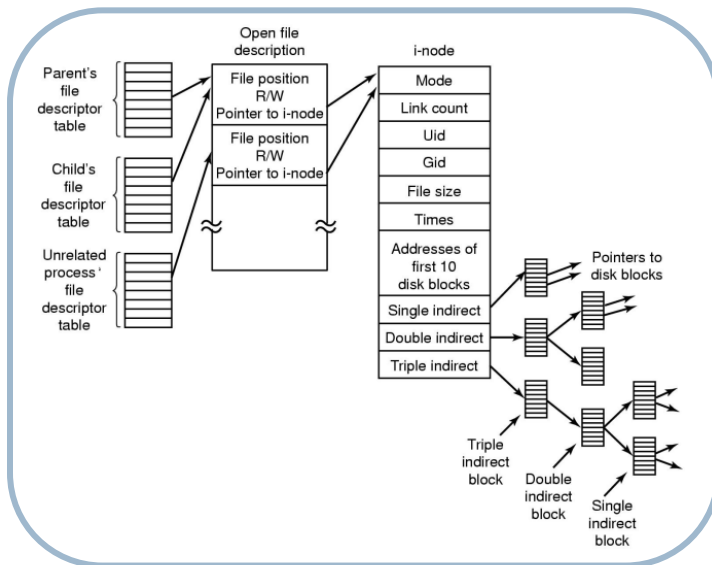
fd → 0	23
1	4563
2	56
3	3
4	678

**Tabla de archivos  
abiertos P2**

fd → 0	230
1	563
2	98
3	3
4	247

**Tabla de archivos  
abiertos P3**

fd → 0	2300
1	53
2	4
3	3465
4	347



**Tabla de  
nodos-i**


Nodo-i	Posición
92	345
92	5678

**Tabla intermedia de  
nodos-i y posiciones**

# Estructuras principales de gestión

## tabla de punteros de posición: compartición de pL/E

52

Sistemas operativos: una visión aplicada (© J. Carrete et al.)

Alejandro Calderón Mateos



**Tabla de archivos  
abiertos P1**

fd → 0	23
1	4563
2	56
3	3
4	678

**Tabla de archivos  
abiertos P2**

fd → 0	230
1	563
2	98
3	3
4	247

**Tabla de archivos  
abiertos P3**

fd → 0	2300
1	53
2	4
3	3465
4	347

- Tabla FILP (FLe Pointer)
- Entre la tabla de descriptores y (normalmente) la tabla de i-nodos.
- Guarda (principalmente) el puntero de posición del archivo.

**Tabla de  
nodos-i**


Nodo-i	Posición
92	345
92	5678

**Tabla intermedia de  
nodos-i y posiciones**

# Estructuras principales de gestión

## tabla de punteros de posición: compartición de pL/E

**Tabla de archivos  
abiertos P1**

fd → 0	23
1	4563
2	56
3	3
4	678

**Tabla de archivos  
abiertos P2**

fd → 0	230
1	563
2	98
3	3
4	247

**Tabla de archivos  
abiertos P3**

fd → 0	2300
1	53
2	4
3	3465
4	347

- ▶ La tabla de nodos-i (o i-nodos) suele tener los i-nodos en disco + información adicional solo en memoria.
- ▶ También tiene todos los datos y referencias a funciones necesarias para trabajar con ficheros y directorios:
  - ▶ Transferencia, metadatos, etc.

**Tabla de  
nodos-i**


Nodo-i	Posición
92	345
92	5678

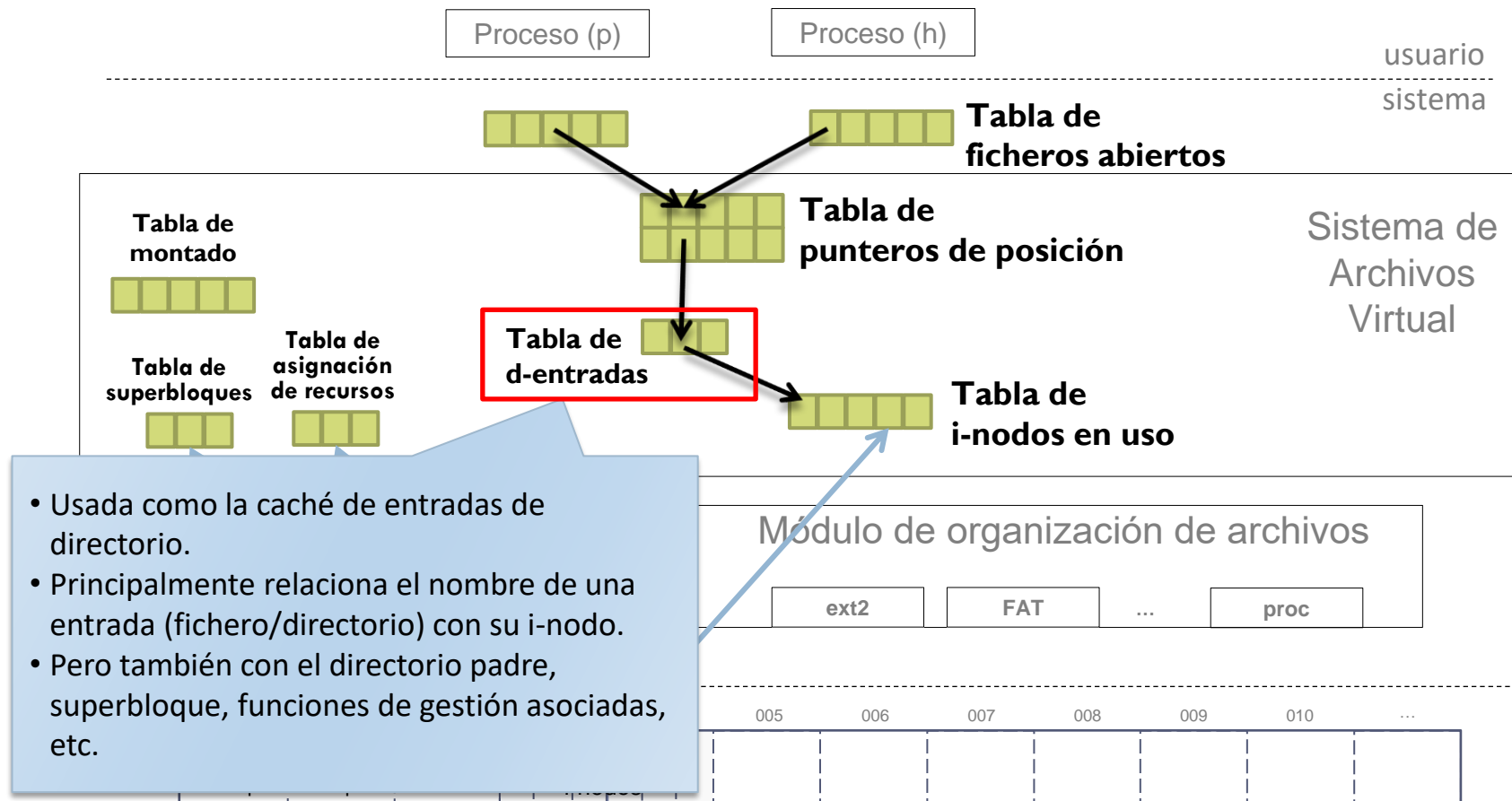
**Tabla intermedia de  
nodos-i y posiciones**

# Estructuras principales de gestión

## tabla de d-entradas: trabajar con directorios

54

Alejandro Calderón Mateos



# Estructuras principales de gestión

## tabla de d-entradas: trabajar con directorios

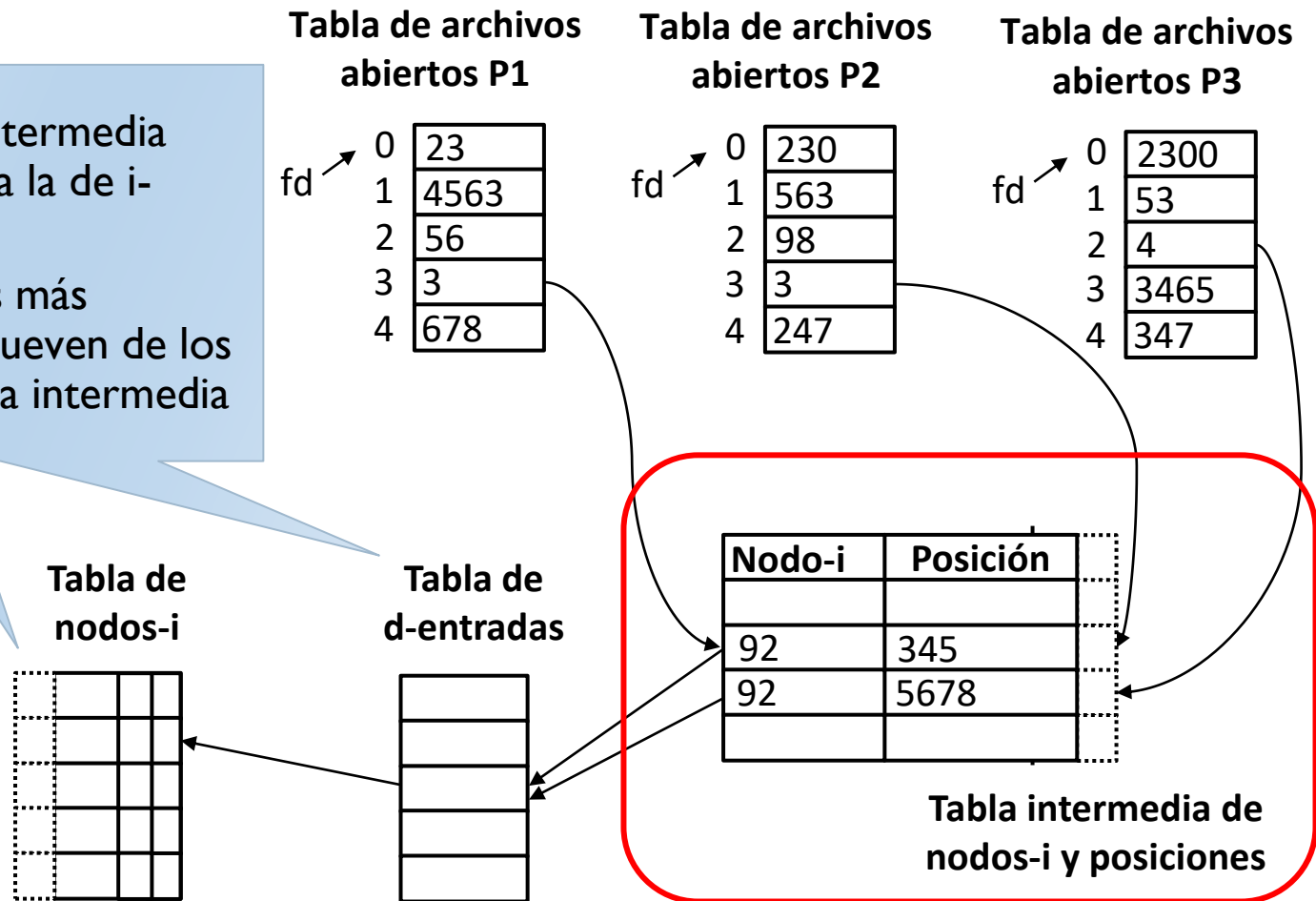


55

Sistemas operativos: una visión aplicada (© J. Carrete et al.)

Alejandro Calderón Mateos

- Hay una tabla intermedia más para llegar a la de i-nodos (nodos-i)
- Las operaciones más frecuentes se mueven de los i-nodos a la tabla intermedia

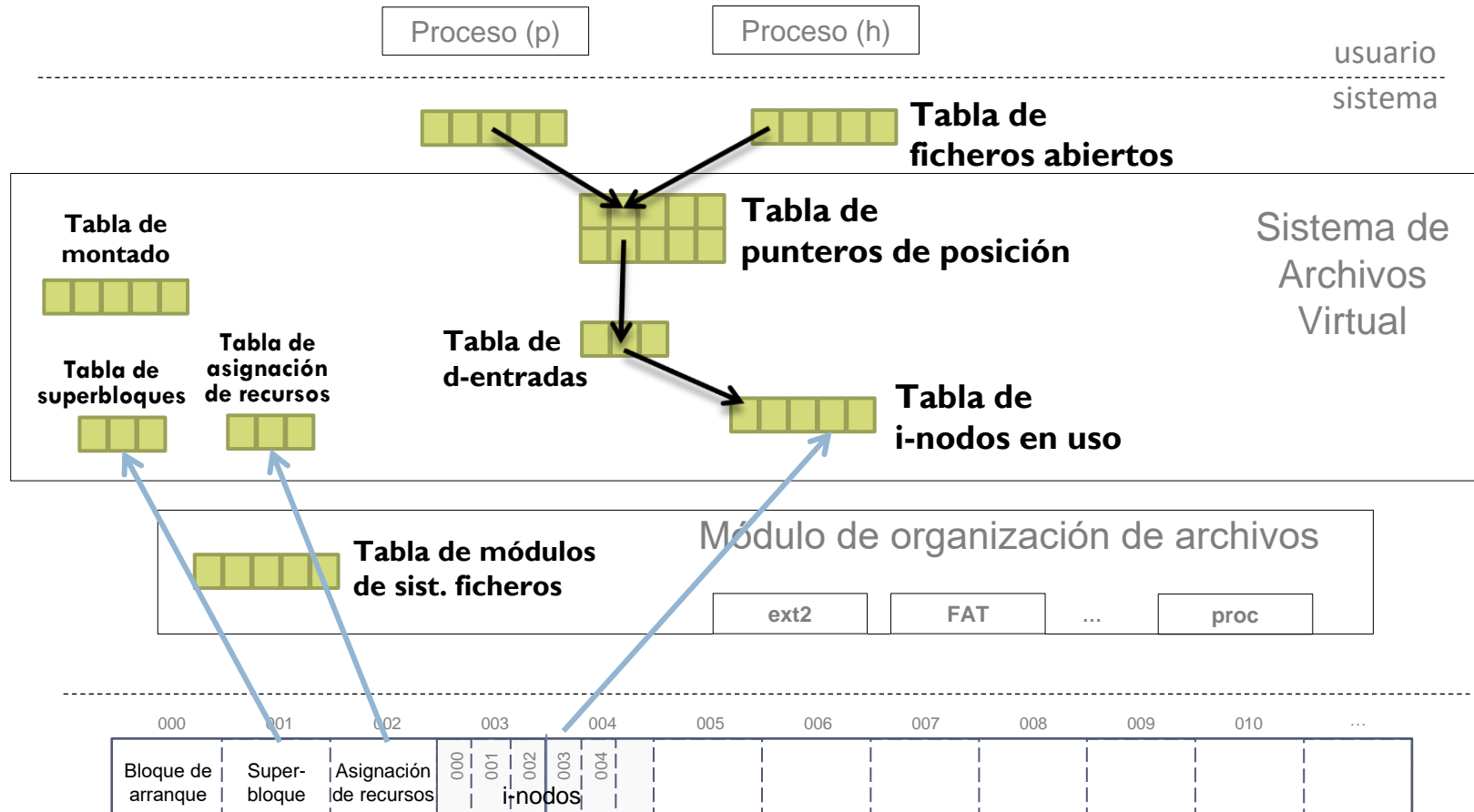


# Estructuras principales de gestión

## resumen de las principales estructuras de datos en memoria

56

Alejandro Calderón Mateos 





# Ejemplo de organización en memoria...

<https://github.com/acaldero/nanofs>



```
// Información leída del disco  
TipoSuperbloque sbloques [1] ;  
char imap [numInodo] ;  
char bmap [numBloquesDatos] ;  
TipoInodoDisco inodos [numInodo] ;
```

```
// Información extra de apoyo  
struct {  
    int posicion;  
    int abierto;  
} inodos_x [numInodo] ;
```

...

# (3a) Gestión de estructuras disco/memoria...

## Llamadas al sistema de archivos

Descriptor	Uso de <i>namei</i>	Asig. i-n.	Atributos	E/S.	Sist. Arch.	Vista
open pipe creat close dup	open chown unlink creat chmod mknod chdir stat mount chroot link umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

XX

## Algoritmos de bajo nivel del sistema de archivos

namei	ialloc	alloc	
iget iput	ifree	free	bmap

d-entradas

montajes

punteros de posición

ficheros abiertos

i-nodos en uso

módulos de s. ficheros

## Algoritmos de gestión de bloques/caché

getblk	brelse	bread	breada	bwrite
--------	--------	-------	--------	--------

000	001	002	003	004	005	006	007	008	009	010	...
Bloque de arranque	Super-bloque	Asignación de recursos	000	001	002	003	004				
			i-nodos								

# Ejemplo de rutinas de gestión i-nodos

59

<http://www.buet.ac.bd/iict/iictcourses/ict6005/lecture9.ppt>

Alejandro Calderón Mateos 

- ▶ **namei**: convierte una ruta al i-nodo asociado.
- ▶ **iget**: devuelve un i-nodo de la tabla de i-nodos y si no está lo lee de memoria secundaria, lo añade a la tabla de i-nodos y lo devuelve.
- ▶ **iput**: libera un i-nodo de la tabla de i-nodos, y si es necesario lo actualiza en memoria secundaria.
- ▶ **ialloc**: asignar un i-nodo a un fichero.
- ▶ **ifree**: libera un i-nodo previamente asignado a un fichero.

Algoritmos de bajo nivel del sistema de archivos

namei	ialloc	alloc	
iget	iput	free	bmap

d-entradas

montajes

punteros de posición

ficheros abiertos

i-nodos en uso

módulos de s. ficheros

Algoritmos de gestión de bloques/caché

getblk	brelse	bread	breada	bwrite
--------	--------	-------	--------	--------

000	001	002	003	004	005	006	007	008	009	010	...
Bloque de arranque	Super-bloque	Asignación de recursos	000	001	002	003	004				
			i-nodos								

# Ejemplo de rutinas de gestión bloques

60

<http://www.buet.ac.bd/iict/iictcourses/ict6005/lecture9.ppt>

Alejandro Calderón Mateos 

- ▶ **bmap**: calcula el bloque de disco asociado a un desplazamiento del fichero. Traduce direcciones lógicas (offset de fichero) a físicas (bloque de disco).
- ▶ **alloc**: asigna un bloque a un fichero.
- ▶ **free**: libera un bloque previamente asignado a un fichero.

Algoritmos de bajo nivel del sistema de archivos

namei	ialloc	alloc	
iget	iput	free	bmap

Algoritmos de gestión de bloques/caché

getblk	brelse	bread	breada	bwrite
--------	--------	-------	--------	--------

d-entradas

montajes

punteros de posición

ficheros abiertos

i-nodos en uso

módulos de s. ficheros

000	001	002	003	004	005	006	007	008	009	010	...
Bloque de arranque	Super-bloque	Asignación de recursos	000	001	002	003	004				
			i-nodos								

# Ejemplo: ialloc y alloc

<https://github.com/acaldero/nanofs>



61

<http://lsi.ugr.es/~jlgarrid/so2/pdf/tem2-1-2.pdf>

Alejandro Calderón Mateos 

```
int ialloc ( void )
{
    // buscar un i-nodo libre
    for (int=0; i<sbloques[0].numInodos; i++)
    {
        if (imap[i] == 0) {
            // inodo ocupado ahora
            imap[i] = 1;
            // valores por defecto en el i-nodo
            memset(&(inodos[i]),0,
                sizeof(TipolnodoDisco));
            // devolver identificador de i-nodo
            return i;
        }
    }

    return -1;
}
```

```
int alloc ( void )
{
    char b[BLOCK_SIZE];

    for (int=0; i<sbloques[0].numBloquesDatos; i++)
    {
        if (bmap[i] == 0) {
            // bloque ocupado ahora
            bmap[i] = 1;
            // valores por defecto en el bloque
            memset(b, 0, BLOCK_SIZE);
            bwrite(DISK, sbloques[0].primerBloqueDatos + i, b);
            // devolver identificador del bloque
            return i;
        }
    }

    return -1;
}
```

# Ejemplo: ifree y free

<https://github.com/acaldero/nanofs>



62

[http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix\\_unit4.pdf](http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix_unit4.pdf)

Alejandro Calderón Mateos 

```
int ifree ( int inodo_id )
{
    // comprobar validez de inodo_id
    if (inodo_id > sbloques[0].numInodos)
        return -1;

    // liberar i-nodo
    imap[inodo_id] = 0;

    return -1;
}
```

```
int free ( int block_id )
{
    // comprobar validez de block_id
    if (block_id > sbloques[0].numBloquesDatos)
        return -1;

    // liberar bloque
    bmap[block_id] = 0;

    return -1;
}
```

# Ejemplo: namei y bmap

<https://github.com/acaldero/nanofs>



63

[http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix\\_unit4.pdf](http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix_unit4.pdf)

Alejandro Calderón Mateos 

```
int namei ( char *fname )
{
    // buscar i-nodo con nombre <fname>
    for (int=0; i<sbloques[0].numInodos; i++)
    {
        if (! strcmp(inodos[i].nombre, fname))
            return i;
    }

    return -1;
}
```

```
int bmap ( int inodo_id, int offset )
{
    int b[BLOCK_SIZE/4];

    // comprobar validez de inodo_id
    if (inodo_id > sbloques[0].numInodos)
        return -1;

    // bloque de datos asociado
    if (offset < BLOCK_SIZE)
        return inodos[inodo_id].bloqueDirecto;
    if (offset < BLOCK_SIZE*BLOCK_SIZE/4) {
        bread(DISK, sbloques[0].primerBloqueDatos +
            inodos[inodo_id].bloqueIndirecto, b);
        offset = (offset - BLOCK_SIZE) / BLOCK_SIZE;
        return b[offset] ;
    }

    return -1;
}
```

## (3b) Llamadas al sistema...

### Llamadas al sistema de archivos

Descriptor	Uso de <i>namei</i>	Asig. i-n.	Atributos	E/S.	Sist. Arch.	Vista
open pipe creat close dup	open chown unlink creat chmod mknod chdir stat mount chroot link umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

### Algoritmos de bajo nivel del sistema de archivos

namei	ialloc	alloc	bmap
iget iput	ifree	free	

d-entradas

montajes

punteros de posición

ficheros abiertos

i-nodos en uso

módulos de s. ficheros

### Algoritmos de gestión de bloques/caché

getblk	brelse	bread	breada	bwrite
--------	--------	-------	--------	--------

000	001	002	003	004	005	006	007	008	009	010	...
Bloque de arranque	Super-bloque	Asignación de recursos	000	001	002	003	004				
			i-nodos								



# Ejemplo

## ll. al sistema

- ▶ **open**: localiza el i-nodo asociado al camino del fichero, ...
- ▶ **read**: localiza el bloque de datos, leer bloque de datos, ...
- ▶ **write**: localiza el bloque de datos, escribir bloque de datos, ...
- ▶ ...

65

[http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix\\_unit4.pdf](http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix_unit4.pdf)

Juanjo Calderón Mateos

### Llamadas al sistema de archivos

Descriptor	Uso de <i>namei</i>	Asig. i-n.	Atributos	E/S.	Sist. Arch.	Vista
open pipe creat close dup	open chown unlink creat chmod mknod chdir stat mount chroot link umount	creat mknod link unlink	chown chmod stat	read write lseek	mount umount	chdir chroot

### Algoritmos de bajo nivel del sistema de archivos

namei	ialloc	alloc	bmap
iget iput	ifree	free	

d-entradas

montajes

punteros de posición

ficheros abiertos

i-nodos en uso

módulos de s. ficheros

### Algoritmos de gestión de bloques/caché

getblk	brelse	bread	breada	bwrite
--------	--------	-------	--------	--------

000	001	002	003	004	005	006	007	008	009	010	...
Bloque de arranque	Super-bloque	Asignación de recursos	000 001 002 003 004 i-nodos								

# Ejemplo: mount

<https://github.com/acaldero/nanofs>



```
int mount ( void )
{
    // leer bloque 0 de disco en sbloques[0]
    bread(DISK, 1, &(sbloques[0]) );

    // leer los bloques para el mapa de i-nodos
    for (int=0; i<sbloques[0].numBloquesMapaInodos; i++)
        bread(DISK, 2+i, ((char *)imap + i*BLOCK_SIZE) );

    // leer los bloques para el mapa de bloques de datos
    for (int=0; i<sbloques[0].numBloquesMapaDatos; i++)
        bread(DISK, 2+i+sbloques[0].numBloquesMapaInodos, ((char *)bmap + i*BLOCK_SIZE);

    // leer los i-nodos a memoria
    for (int=0; i<(sbloques[0].numInodos*sizeof(TipoInodoDisco)/BLOCK_SIZE); i++)
        bread(DISK, i+sbloques[0].primerInodo, ((char *)inodos + i*BLOCK_SIZE);

    return 1;
}
```

# Ejemplo: umount

<https://github.com/acaldero/nanofs>



```
int umount ( void )
{
    // escribir bloque 0 de sbloques[0] a disco
    bwrite(DISK, 1, &(sbloques[0]) );

    // escribir los bloques para el mapa de i-nodos
    for (int=0; i<sbloques[0].numBloquesMapaInodos; i++)
        bwrite(DISK, 2+i, ((char *)imap + i*BLOCK_SIZE) );

    // escribir los bloques para el mapa de bloques de datos
    for (int=0; i<sbloques[0].numBloquesMapaDatos; i++)
        bwrite(DISK, 2+i+sbloques[0].numBloquesMapaInodos, ((char *)bmap + i*BLOCK_SIZE);

    // escribir los i-nodos a disco
    for (int=0; i<(sbloques[0].numInodos*sizeof(TipoInodoDisco)/BLOCK_SIZE); i++)
        bwrite(DISK, i+sbloques[0].primerInodo, ((char *)inodos + i*BLOCK_SIZE);

    return 1;
}
```

# Ejemplo: mkfs

<https://github.com/acaldero/nanofs>



```
int mkfs ( void )
{
    // inicializar a los valores por defecto del superbloque, mapas e i-nodos
    sbloques[0].numMagico = 1234; // ayuda a comprobar que se haya creado por nuestro mkfs
    sbloques[0].numInodos = numInodo;
    ...
    for (int=0; i<sbloques[0].numInodos; i++)
        imap[i] = 0; // free
    for (int=0; i<sbloques[0].numBloquesDatos; i++)
        bmap[i] = 0; // free
    for (int=0; i<sbloques[0].numInodos; i++)
        memset(&(inodos[i]), 0, sizeof(TipoInodoDisco) );

    // to write the default file system into disk
    umount();

    return 1;
}
```

# Ejemplo: open y close

<https://github.com/acaldero/nanofs>



69

[http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix\\_unit4.pdf](http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix_unit4.pdf)

Alejandro Calderón Mateos 

```
int open ( char *nombre )
```

```
{
```

```
    int inodo_id ;
```

```
    inodo_id = namei(nombre) ;
```

```
    if (inodo_id < 0)
```

```
        return inodo_id ;
```

```
    inodos_x[inodo_id].posicion = 0;
```

```
    inodos_x[inodo_id].abierto  = 1;
```

```
    return inodo_id;
```

```
}
```

```
int close ( int fd )
```

```
{
```

```
    if (fd < 0)
```

```
        return fd ;
```

```
    inodos_x[fd].posicion = 0;
```

```
    inodos_x[fd].abierto  = 0;
```

```
    return 1;
```

```
}
```

# Ejemplo: creat y unlink

<https://github.com/acaldero/nanofs>



70

[http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix\\_unit4.pdf](http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix_unit4.pdf)

Alejandro Calderón Mateos 

```
int creat ( char *nombre )
{
    int b_id, inodo_id ;

    inodo_id = ialloc() ;
    if (inodo_id < 0) { return inodo_id ; }
    b_id = alloc();
    if (b_id < 0) { ifree(inodo_id); return b_id ; }

    inodos[inodo_id].tipo = 1 ; // FICHERO
    strcpy(inodos[inodo_id].nombre, nombre);
    inodos[inodo_id].bloqueDirecto = b_id ;
    inodos_x[inodo_id].posicion = 0;
    inodos_x[inodo_id].abierto  = 1;

    return 1;
}
```

```
int unlink ( char * nombre )
{
    int inodo_id ;

    inodo_id = namei(nombre) ;
    if (inodo_id < 0)
        return inodo_id ;

    free(inodos[inodo_id].bloqueDirecto);
    memset(&(inodos[inodo_id]),
           0,
           sizeof(TipoInodoDisco));
    ifree(inodo_id) ;

    return 1;
}
```

# Ejemplo: read y write

<https://github.com/acaldero/nanofs>



71

[http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix\\_unit4.pdf](http://mycsvtunotes.weebly.com/uploads/1/0/1/7/10174835/unix_unit4.pdf)

Alejandro Calderón Mateos 

```
int read ( int fd, char *buffer, int size )
{
    char b[BLOCK_SIZE] ;
    int b_id ;

    if (inodos_x[fd].posicion+size > inodos[fd].size)
        size = inodos[fd].size - inodos_x[fd].posicion;
    if (size <= 0)
        return 0;

    b_id = bmap(fd, inodos_x[fd].posicion);
    bread(DISK,
        sbloques[0].primerBloqueDatos+b_id, b);
    memmove(buffer,
        b+inodos_x[fd].posicion, size);
    inodos_x[fd].posicion += size;

    return size;
}
```

```
int write ( int fd, char *buffer, int size )
{
    char b[BLOCK_SIZE] ;
    int b_id ;

    if (inodos_x[fd].posicion+size > BLOCK_SIZE)
        size = BLOCK_SIZE - inodos_x[fd].posicion;
    if (size <= 0)
        return 0;

    b_id = bmap(fd, inodos_x[fd].posicion);
    bread(DISK, sbloques[0].primerBloqueDatos+b_id, b);
    memmove(b+inodos_x[fd].posicion,
        buffer, size);
    bwrite(DISK, sbloques[0].primerBloqueDatos+b_id, b);
    inodos_x[fd].posicion += size;

    return size;
}
```

# SISTEMAS OPERATIVOS: SISTEMAS DE FICHEROS



Ficheros, directorios y sistema de ficheros