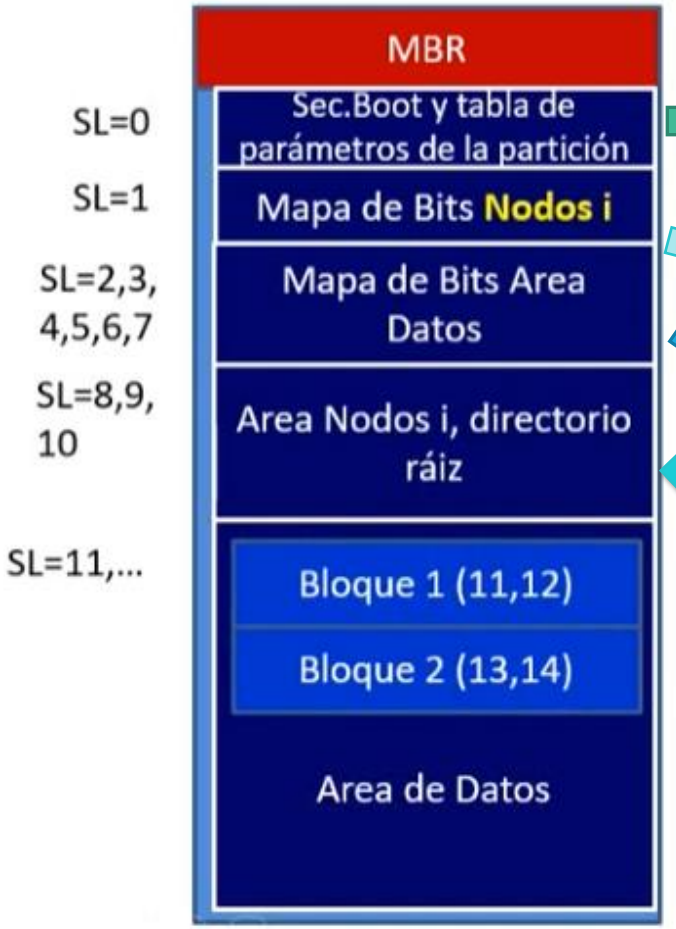
SISTEMAS OPERATIVOS

INGENIERÍA CIVIL INFORMÁTICA

GONZALO CARREÑO

GONZALOCARRENOB@GMAIL.COM

Áreas del sistema de archivos



Contiene: el código de arranque del sistema operativo instalado en la partición y la tabla de parámetros de la partición.

Indica que nodos i están libres y cuales están ocupados

Nodo i: por cada archivo hay una entrada en el directorio llamada nodo i

Indica que bloques están libres y que bloques están ocupados, lleva el control del espacio libre

Los nodos-i (1 por cada archivo) en el directorio raíz de la partición

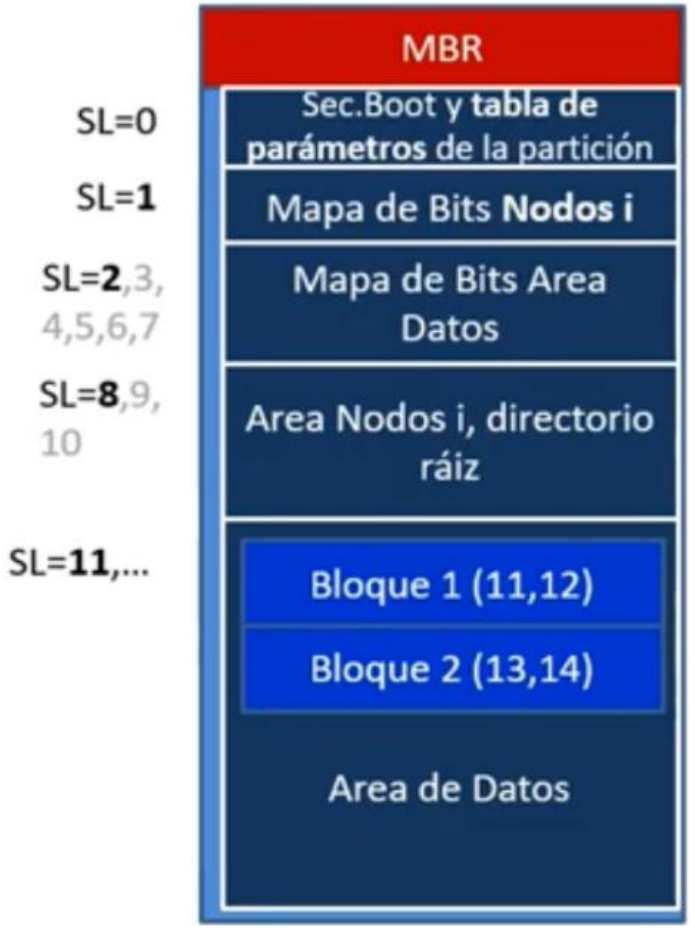
El área para los bloques de los archivos

Un nodo i es una estructura de datos que va a describir las características los detalles de cada uno de esos archivos. Por lo tanto, por cada archivo va a existir un nodo i en el directorio raíz.

Áreas del sistema de archivo

Con los datos que están en la tabla de parámetros de la partición podemos calcular en que sector lógico inicia cada una de las áreas

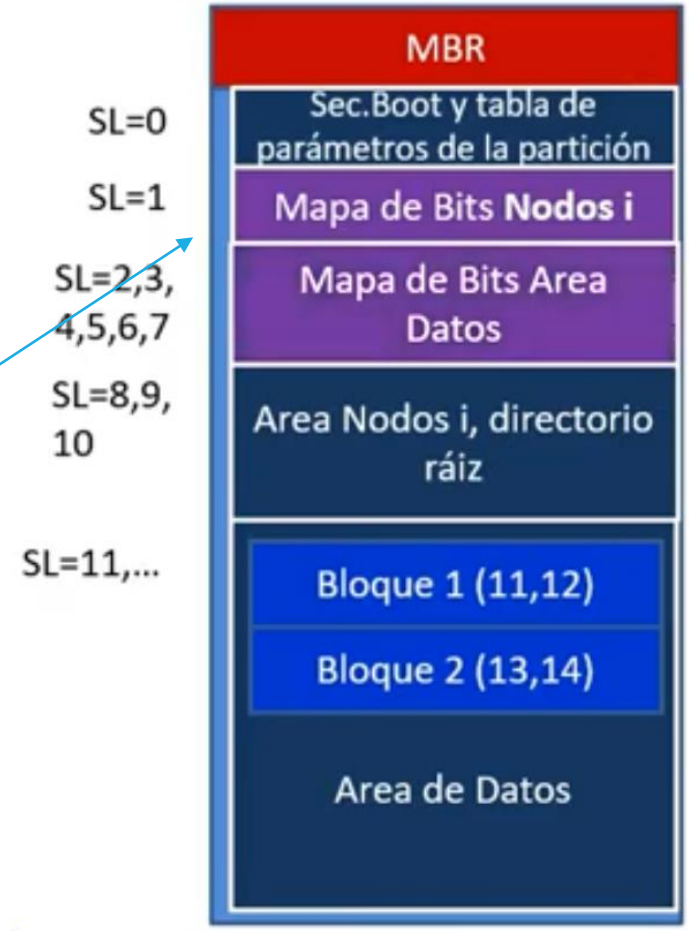
- **SL_Mapa_bits_nodos_i** = sec_reservados (1)
- **SL_Mapa_bits_área_datos** = sec_reservados (1) + sec_mapa_bits_nodos_i (1)
- **SL_Area_nodos_i** = sec_reservados (1) + sec_mapa_bits_nodos_i (1) + sec_mapa_bits_área_datos (6)
- **SL_Area_datos** = sec_reservados (1) + sec_mapa_bits_nodos_i (1) + sec_mapa_bits_área_datos (6) + sec_nodos_i (3)



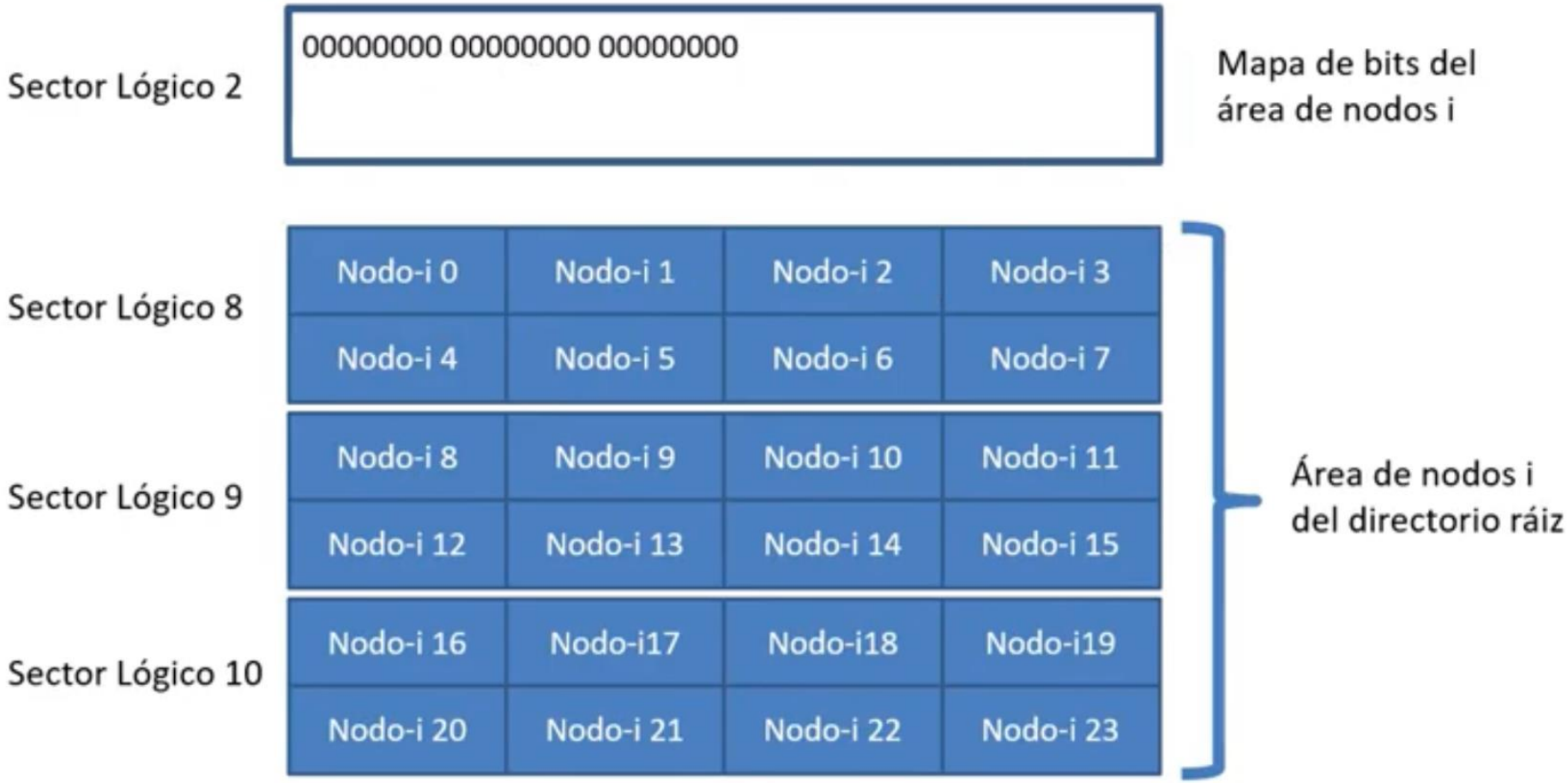
Mapas de bits

Nuestro sistema de archivos tendrá 2 mapas de bits

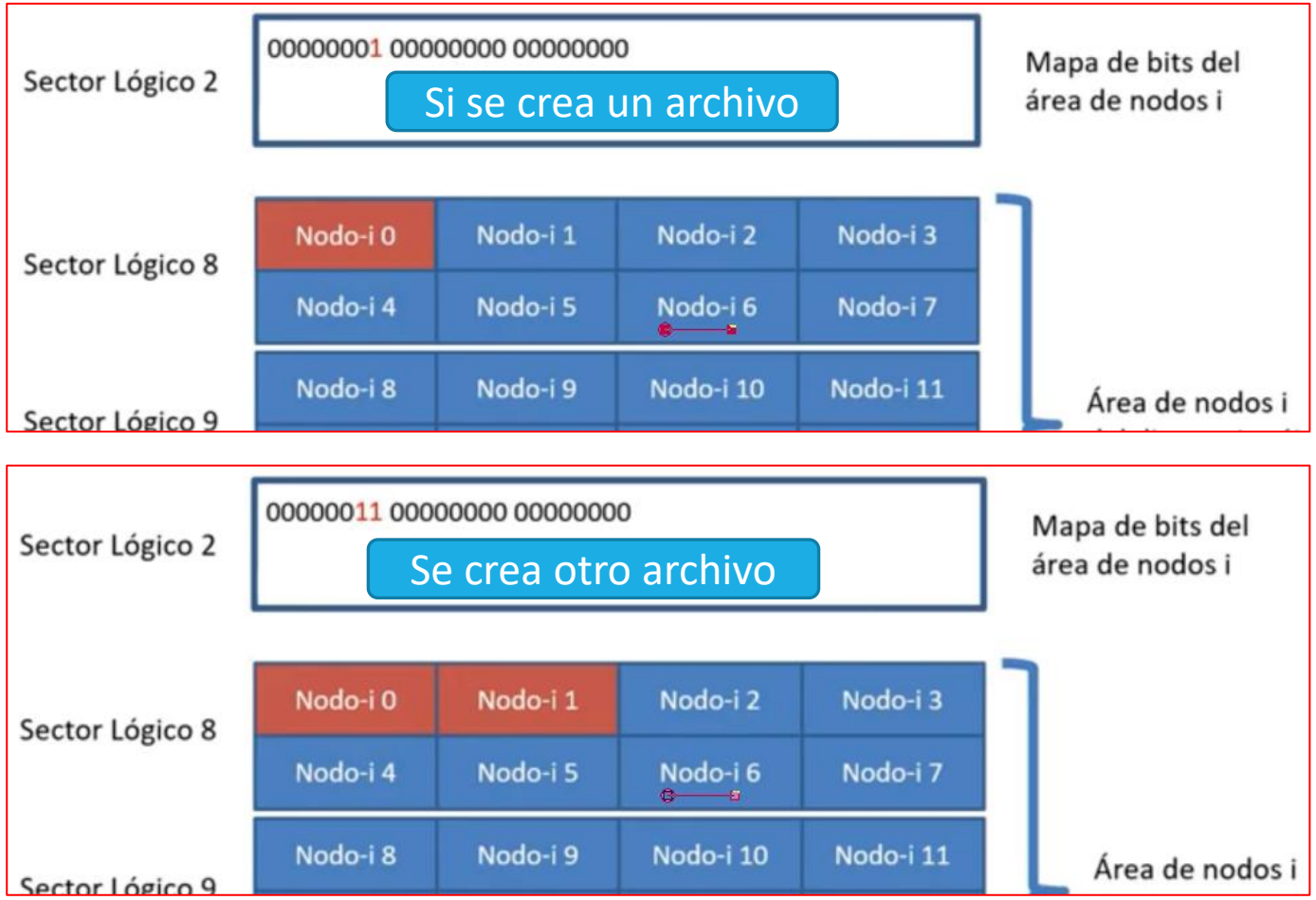
- Se utilizara para llevar el control de los nodos-i libres y ocupados.
- Por cada nodo-i existe un bit que indica su estado.
- Si el bit es 0 significa que el nodo-i esta libre, y si es 1 esta ocupado.



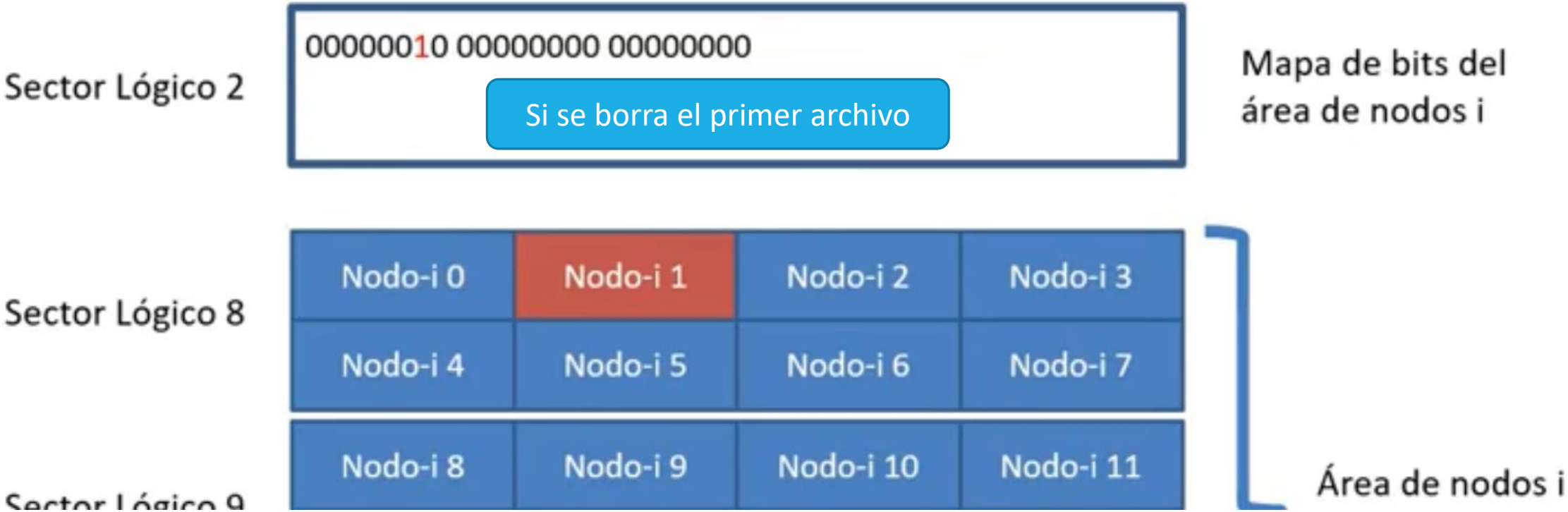
Mapa de bits del área de nodos i



Mapa de bits del área de nodos i



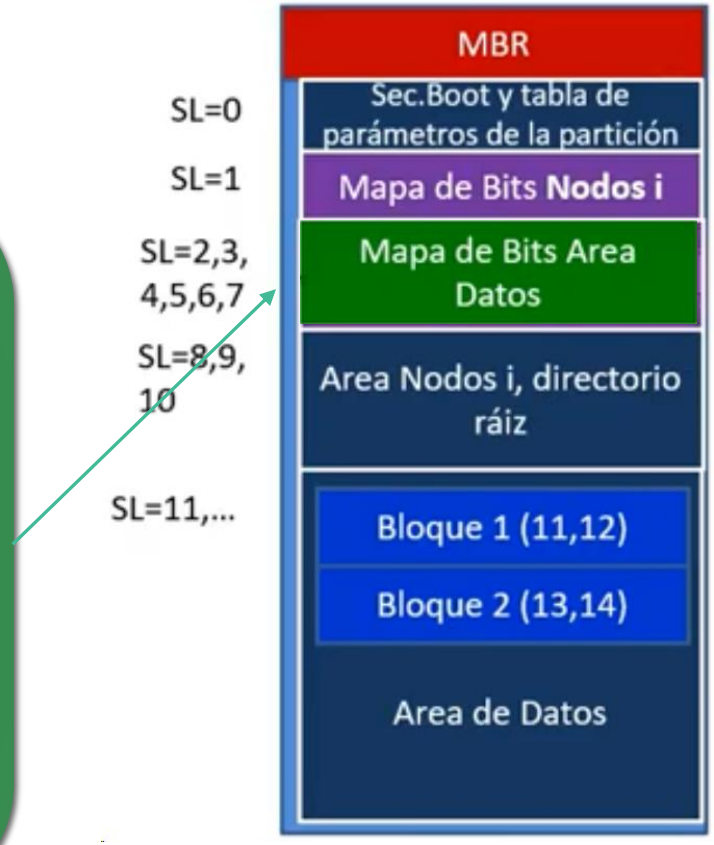
Mapa de bits del área de nodos i



Mapa de bits del área de datos

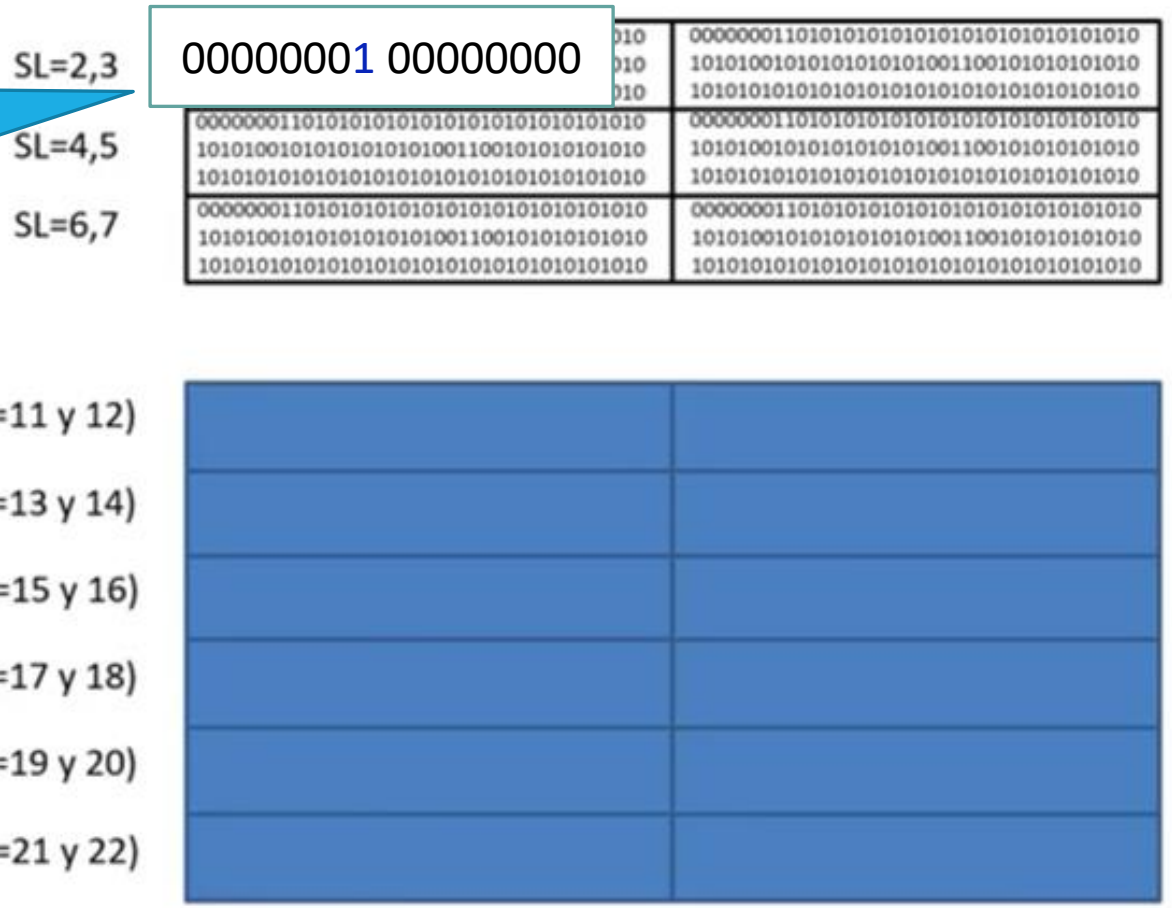
Nuestro sistema de archivos tendrá dos mapas de bits.

- Se utiliza para llevar el control de los bloques del área de datos. (necesitamos saber que bloques están libres y cuales están ocupados)
- La información contenida en los archivos se almacena en los bloques en el área de datos. (El primer bloque no parte del 0 ya que es un valor reservado utilizado para indicar que no se apunta ningún bloque)
- Su funcionamiento es idéntico al del mapa de bits de los nodos-i. (Si el bit correspondiente a un nombre esta en 0 quiere decir que ese bloque esta libre y puede ser utilizado para escribir un archivo. Si el bit en cambio esta en 1 quiere decir que ese bloque esta ocupado y no debe ser utilizado)



Mapa de bits del área de datos

Este bit debe iniciar en 1 ya que el 0 no lo usaremos para nombrar bloques



Cada bit mapea un bloque

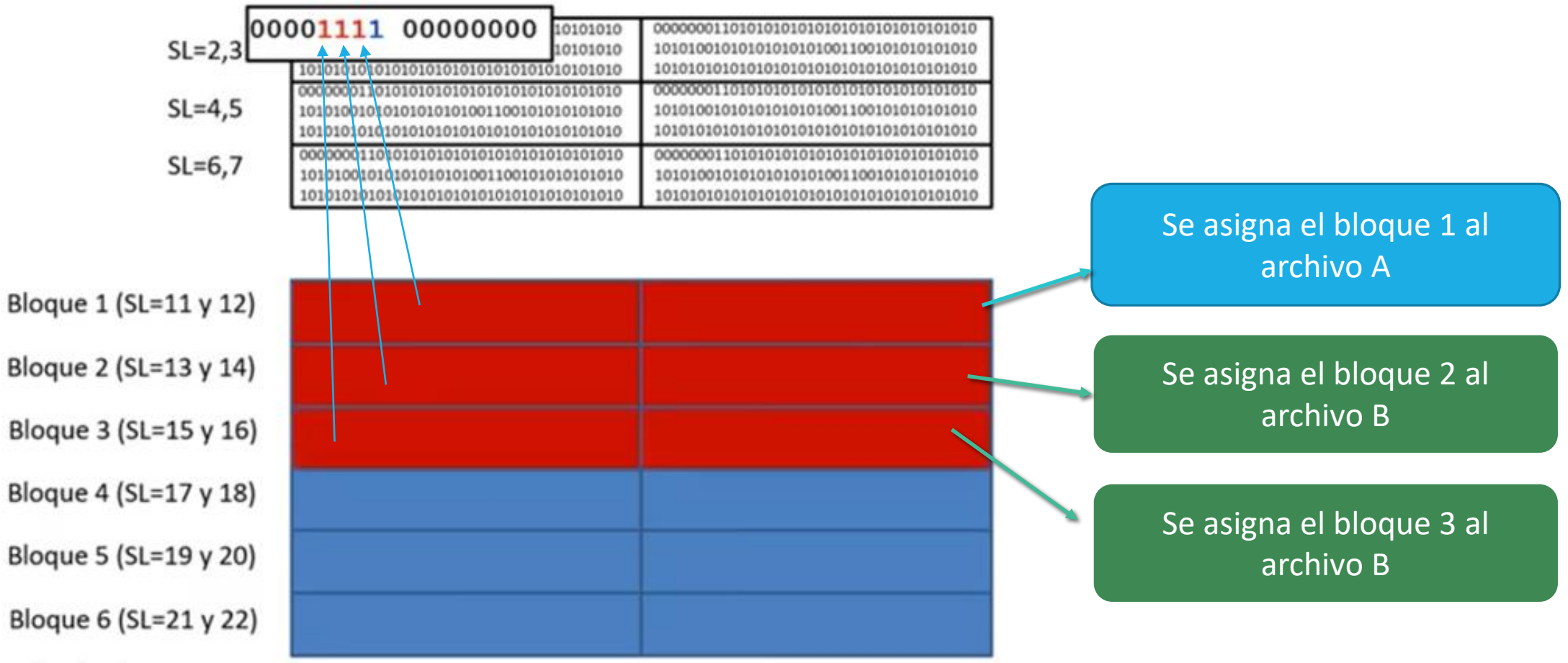
6 sectores para el mapa de bits

=512x6=3072 bytes

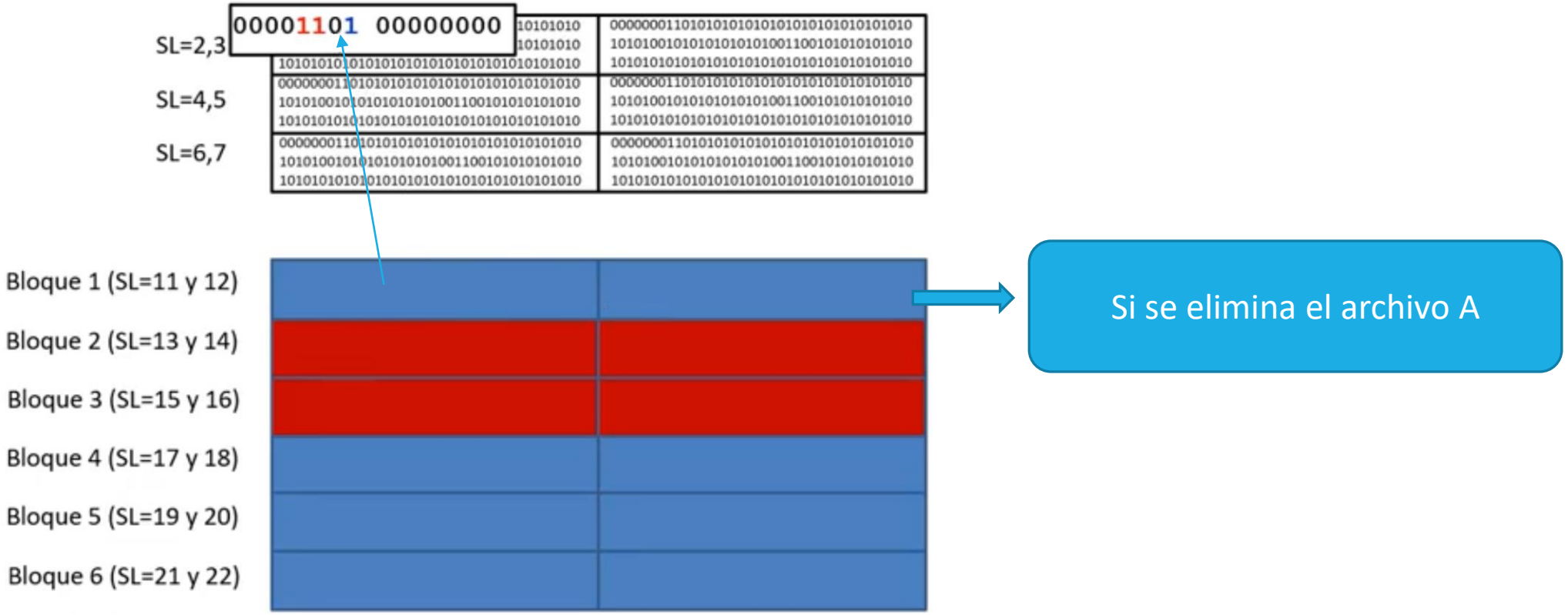
=3072x8=24576 bits

Se pueden mapear 24576 bloques

Mapa de bits del área de datos



Mapa de bits del área de datos



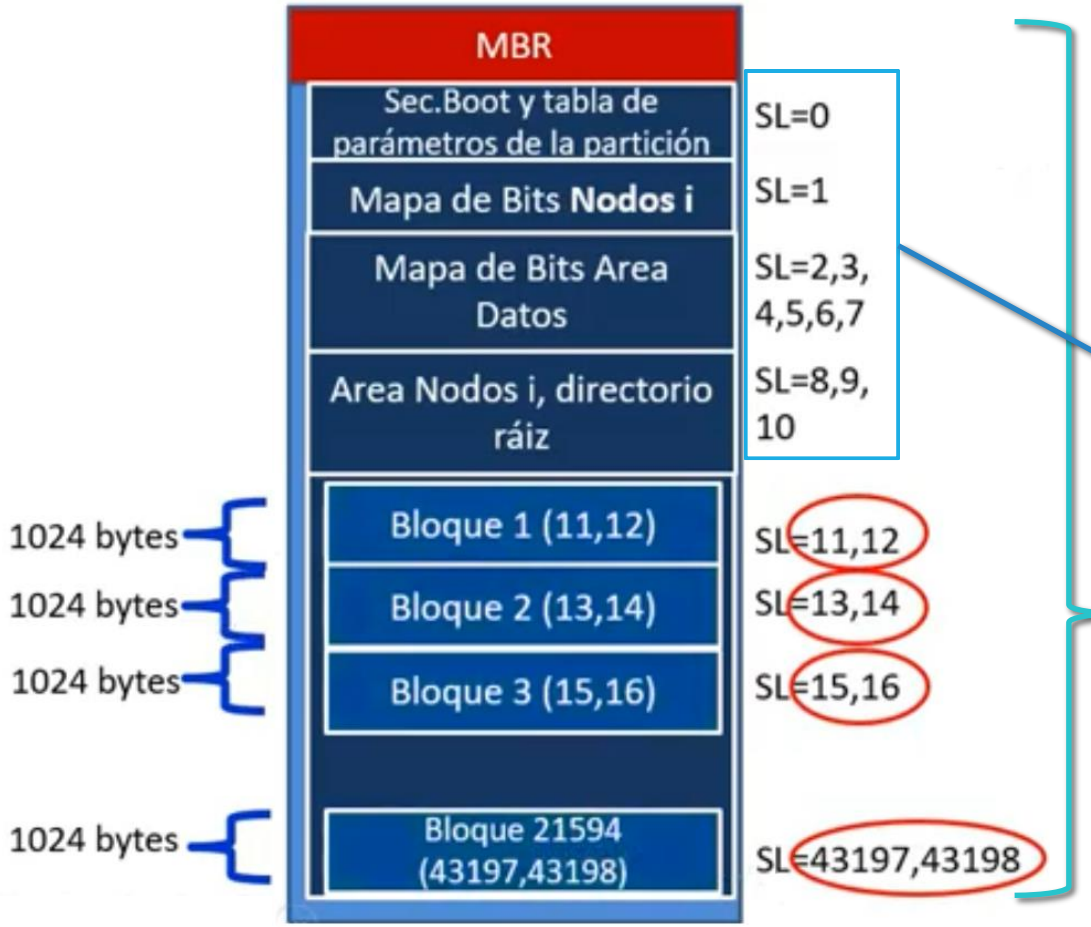
Bloques

- Se forma de uno o mas sectores
- Si es mas de uno, usualmente el tamaño es potencia de 2
  - 2 sectores x bloque
  - 4 sectores x bloque
  - 8 sectores x bloque
  - 16 sectores x bloque
  - 32 sectores x bloque
  - 64 sectores x bloque
  - 128 sectores x bloque
  - …

Bloques

- En los bloques se almacena el contenido de los archivos o apuntadores a otros bloques
- Es la unidad mínima de asignación para los archivos
  - Un archivo por mas pequeño que sea va a ocupar un bloque completo
- El primer bloque del sistema de archivos será el bloque 1
  - El numero 0 es un valor reservado.

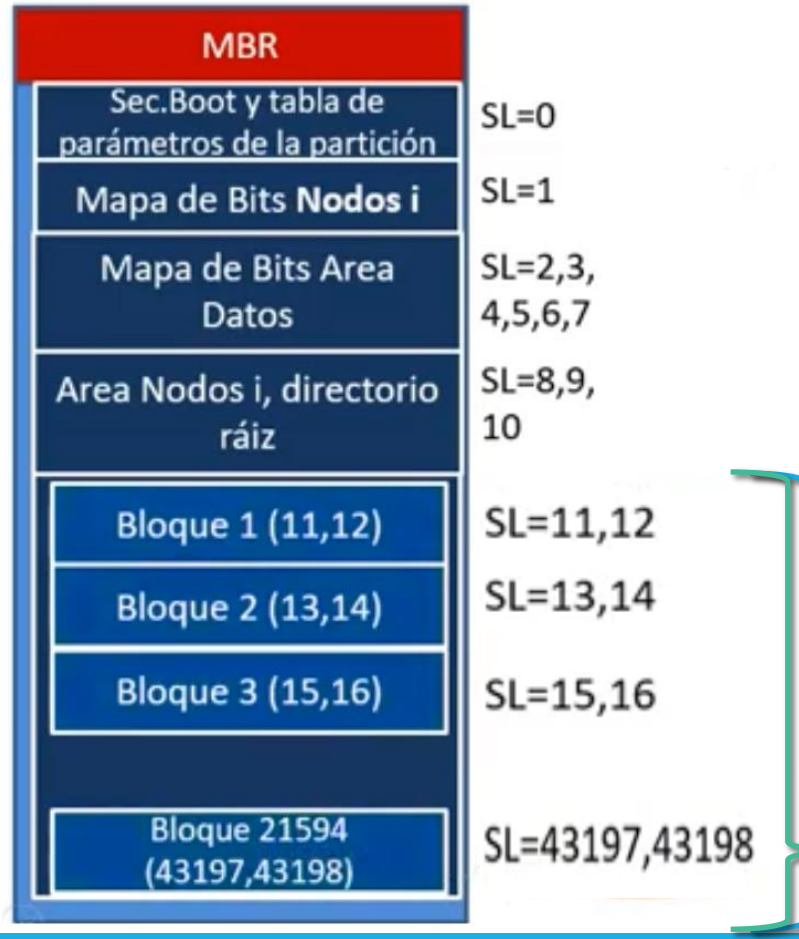
Bloques



- En nuestro sistema de archivos un bloque será conformado por dos sectores
- 1 bloque = 2x512 bytes = 1024 bytes
- La partición tiene 43199 sectores, menos 11 sectores que usamos para las áreas criticas = 43188 sectores para los bloques.

43199 – 11 = 43188 sectores

Bloques



- En nuestro sistema de archivos un bloque será conformado por dos sectores
- 1 bloque = 2x512 bytes = 1024 bytes
- La partición tiene 43199 sectores, menos 11 sectores que usamos para las áreas criticas = 43188 sectores para los bloques.
- 43188 sectores en el área de archivos
- 43188 entre 2 sectores x bloque = 21594 bloques.

43188 sectores/ 2 sectores x bloque

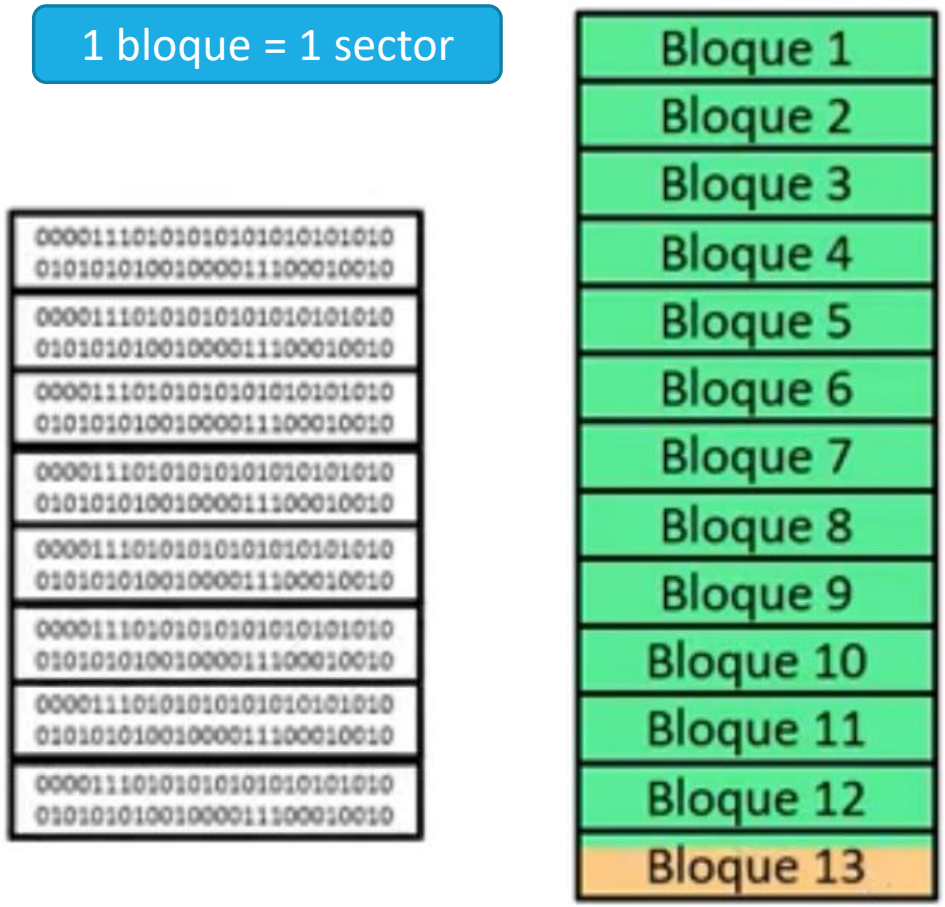
= 21594 bloques

Bloques grandes vs bloques pequeños

¿Cuántos sectores por bloque?

Bloques pequeños

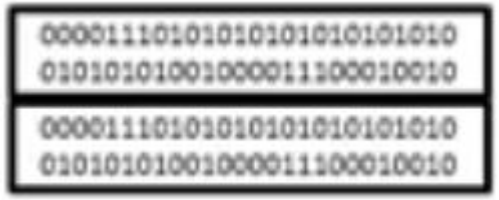
- Pocos sectores por bloque
- Muchos bloques por archivo
- Se desperdicia menos espacio en el ultimo bloque de los archivos.
- Muchos bloques por partición -> Muchos sectores para el mapa de bits.

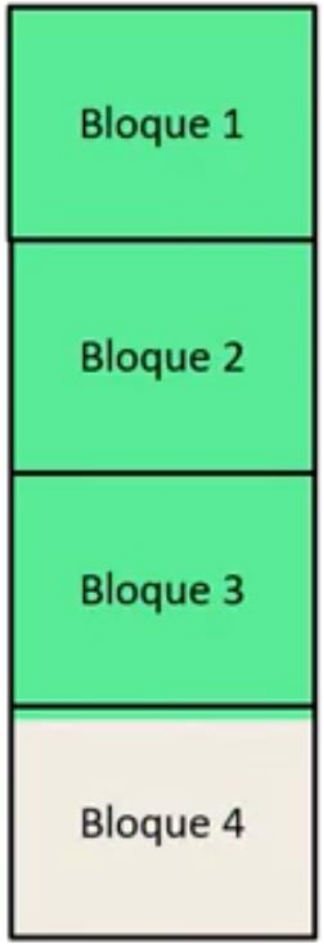


Bloques grandes vs bloques pequeños

¿Cuántos sectores por bloque?

1 bloque = 4 sectores

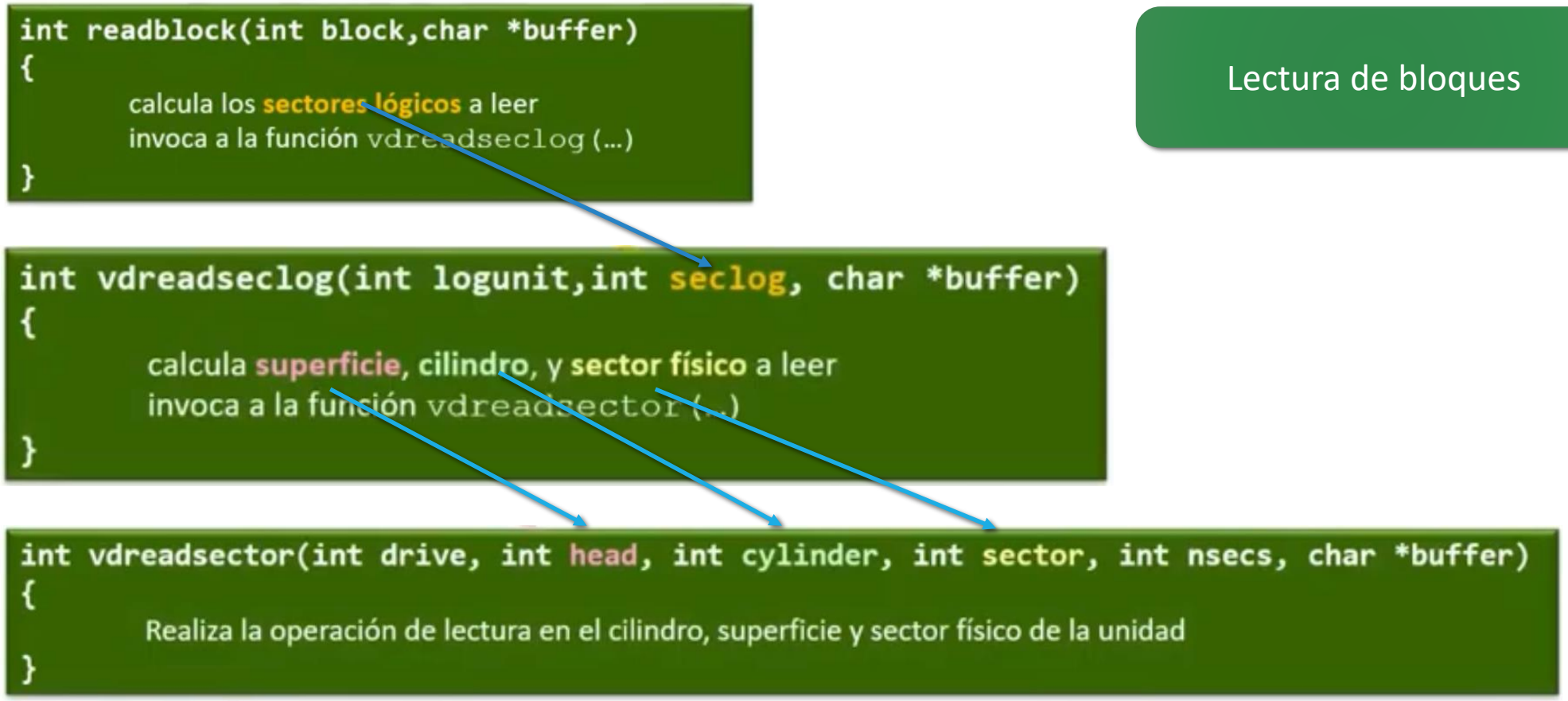




Bloques grande

- Muchos sectores por bloques
- Pocos bloques por archivo
- Se desperdicia mucho espacio en el ultimo bloque de los archivos
- Pocos bloques por partición -> Pocos sectores para el mapa de bits

Funciones lectura y escritura de bloques



Funciones lectura y escritura de bloques

```
int writeblock(int block,char *buffer)
{
        calcula los sectores lógicos a escribir
        invoca a la función vdwriteseclog(…)
}
```

Escritura de bloques

```
int vdwriteseclog(int logunit,int seclog, char *buffer)
{
        calcula superficie, cilindro, y sector físico a escribir
        invoca a la función vdwritesector(…)
}
```

```
int vdwritesector(int drive, int head, int cylinder, int sector, int nsecs, char *buffer)
{
        Realiza la operación de escritura en el cilindro, superficie y sector físico de la unidad
}
```

Nodos i

Nodo índice

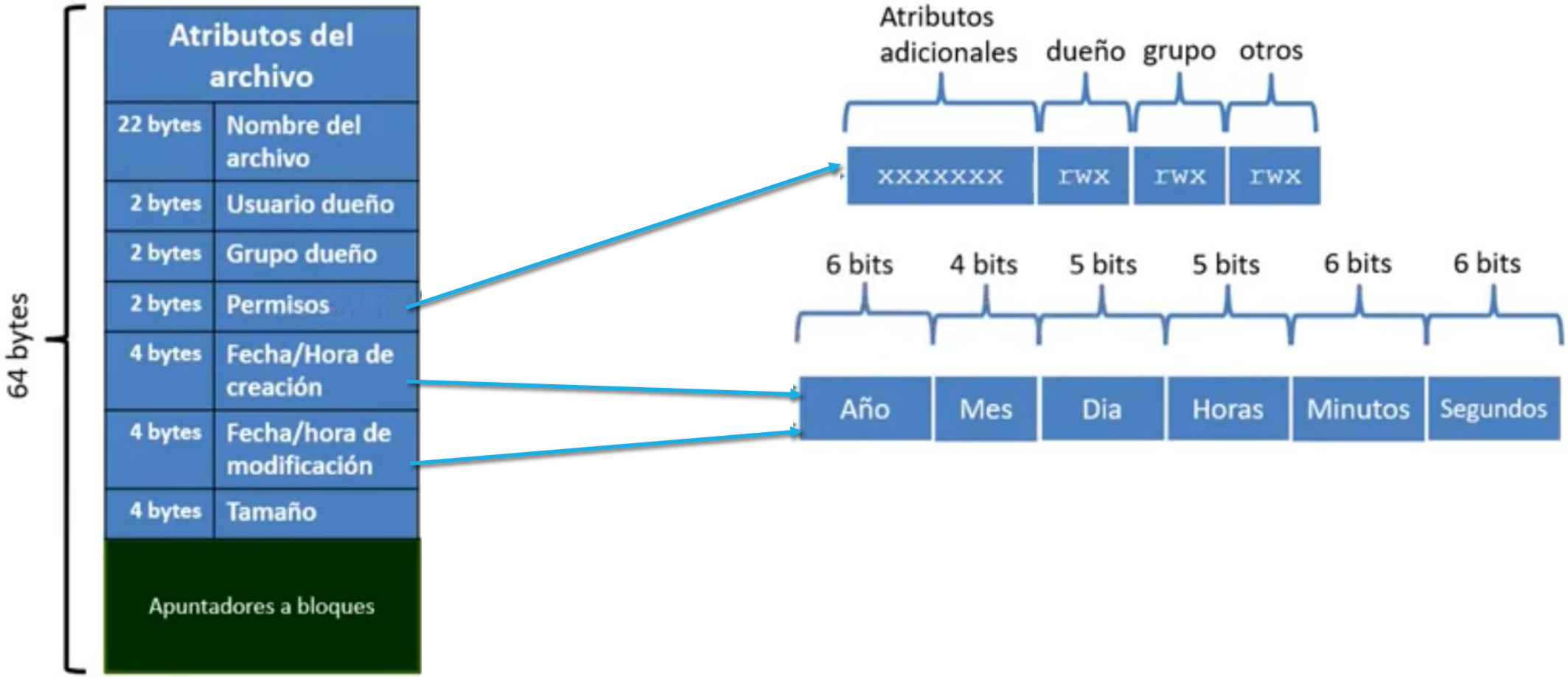
Estructura de datos propia de los sistemas de archivos.

- Común en los sistemas operativos tipo UNIX, por ejemplo Linux

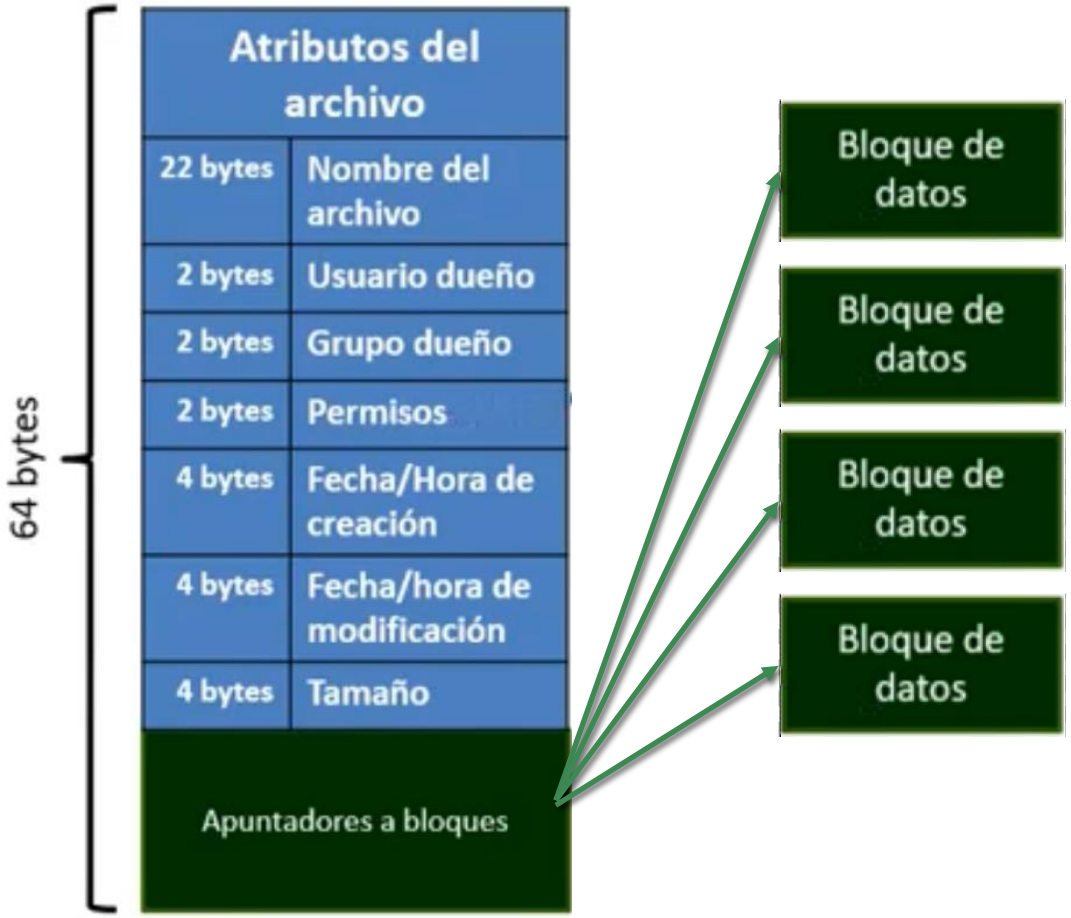
Un nodo-i contiene la descripción de:

- Un archivo regular
- Un directorio
- Enlaces simbólicos
  - Puede tener más de un nombre en distintos o incluso en el mismo directorio

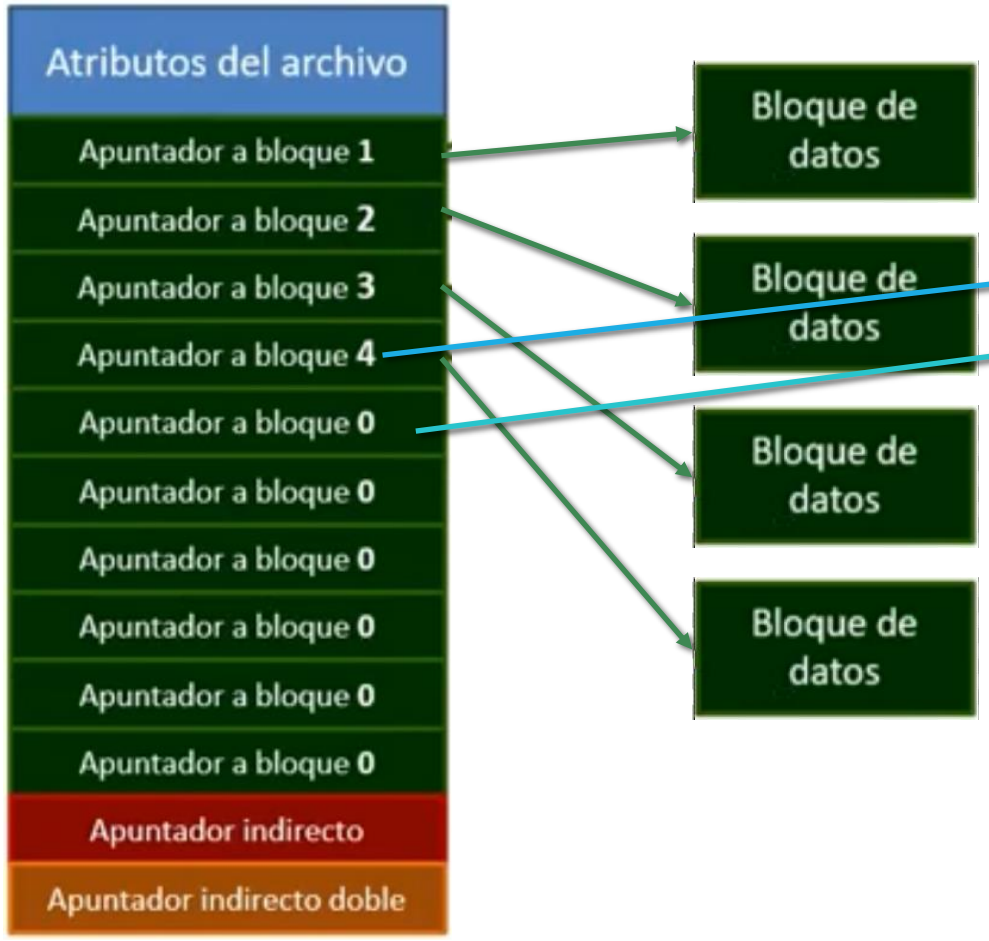
Nodos i


64 bytes
Atributos del archivo
22 bytes
Nombre del archivo
2 bytes
Usuario dueño
2 bytes
Grupo dueño
2 bytes
Permisos
4 bytes
Fecha/Hora de creación
4 bytes
Fecha/hora de modificación
4 bytes
Tamaño
Apuntadores a bloques
Atributos adicionales
dueño
grupo
otros
xxxxxxx
rwx
rwx
rwx
6 bits
4 bits
5 bits
5 bits
6 bits
6 bits
Año
Mes
Dia
Horas
Minutos
Segundos

Nodos i


64 bytes
Atributos del archivo
22 bytes
Nombre del archivo
2 bytes
Usuario dueño
2 bytes
Grupo dueño
2 bytes
Permisos
4 bytes
Fecha/Hora de creación
4 bytes
Fecha/hora de modificación
4 bytes
Tamaño
Apuntadores a bloques
Bloque de datos
Bloque de datos
Bloque de datos
Bloque de datos

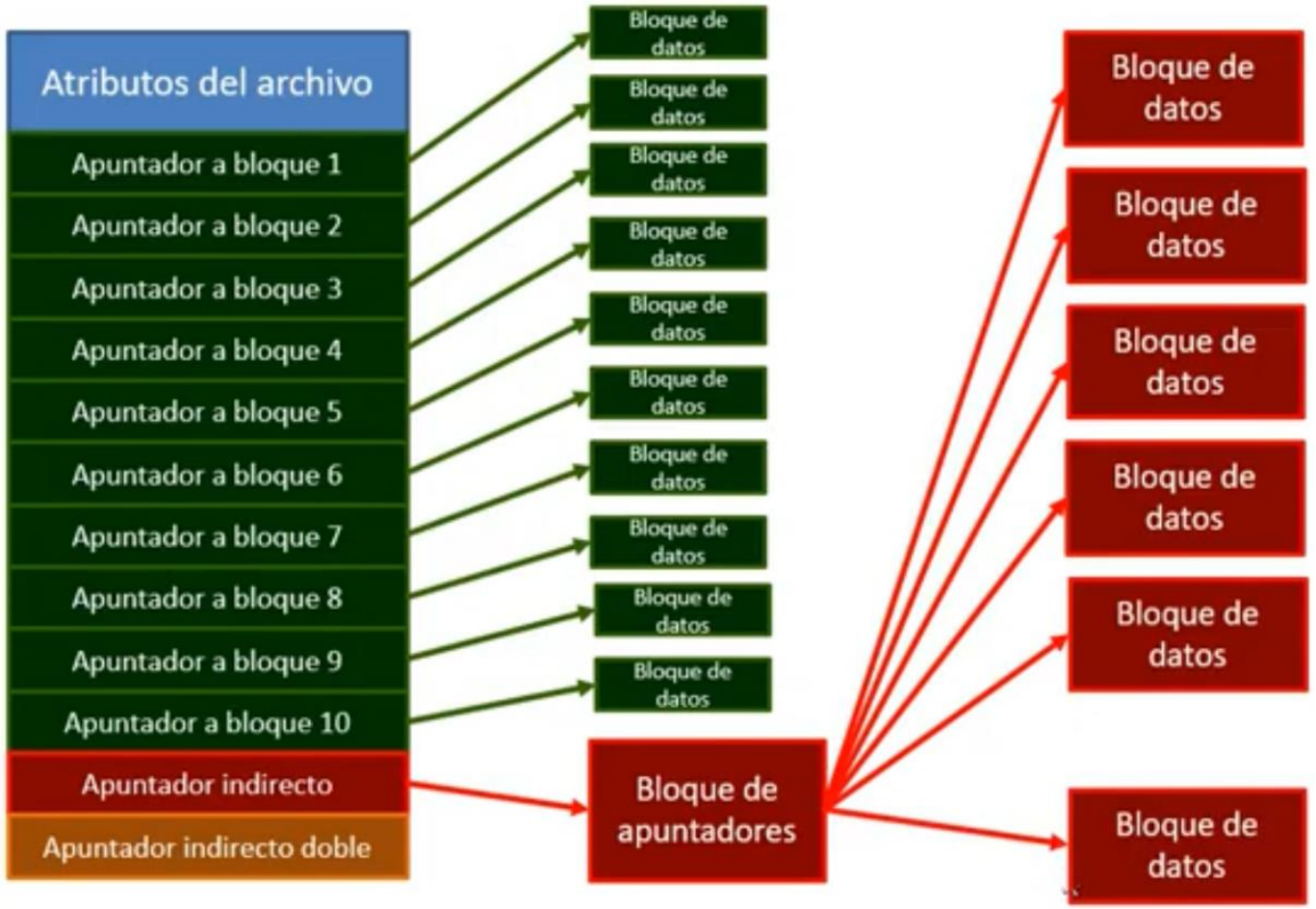
Nodos i



- Los apuntadores a bloques son de 16bits
  - Máximo 65535 bloques
- Cada apuntador indica un bloque del archivo
- Si el apuntador es 0 entonces no apunta a ningún bloque

Con 10 apuntadores un archivo podría medir 10 Kbytes
Podría tener un pequeño archivo de texto solamente

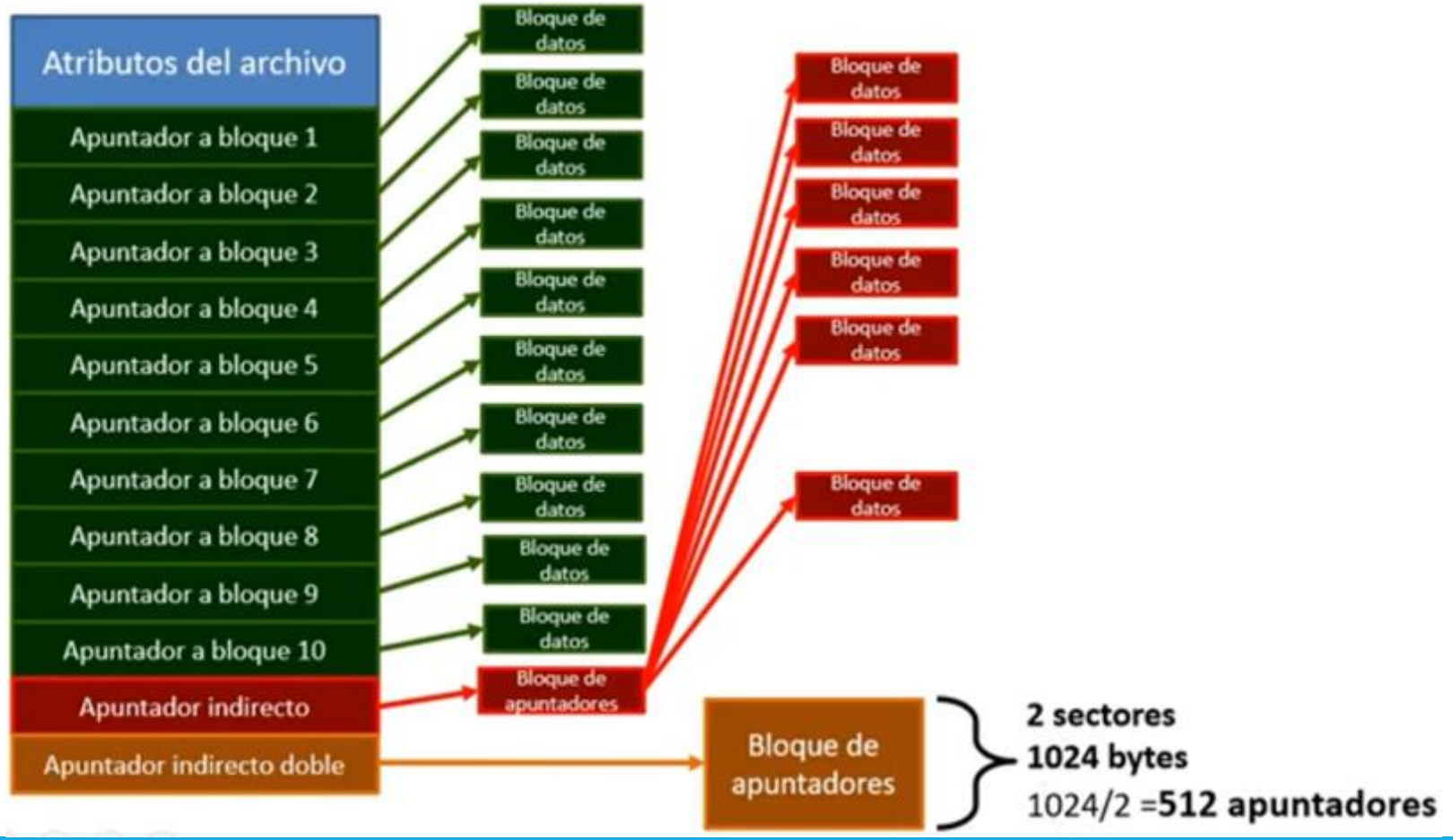
Nodos i



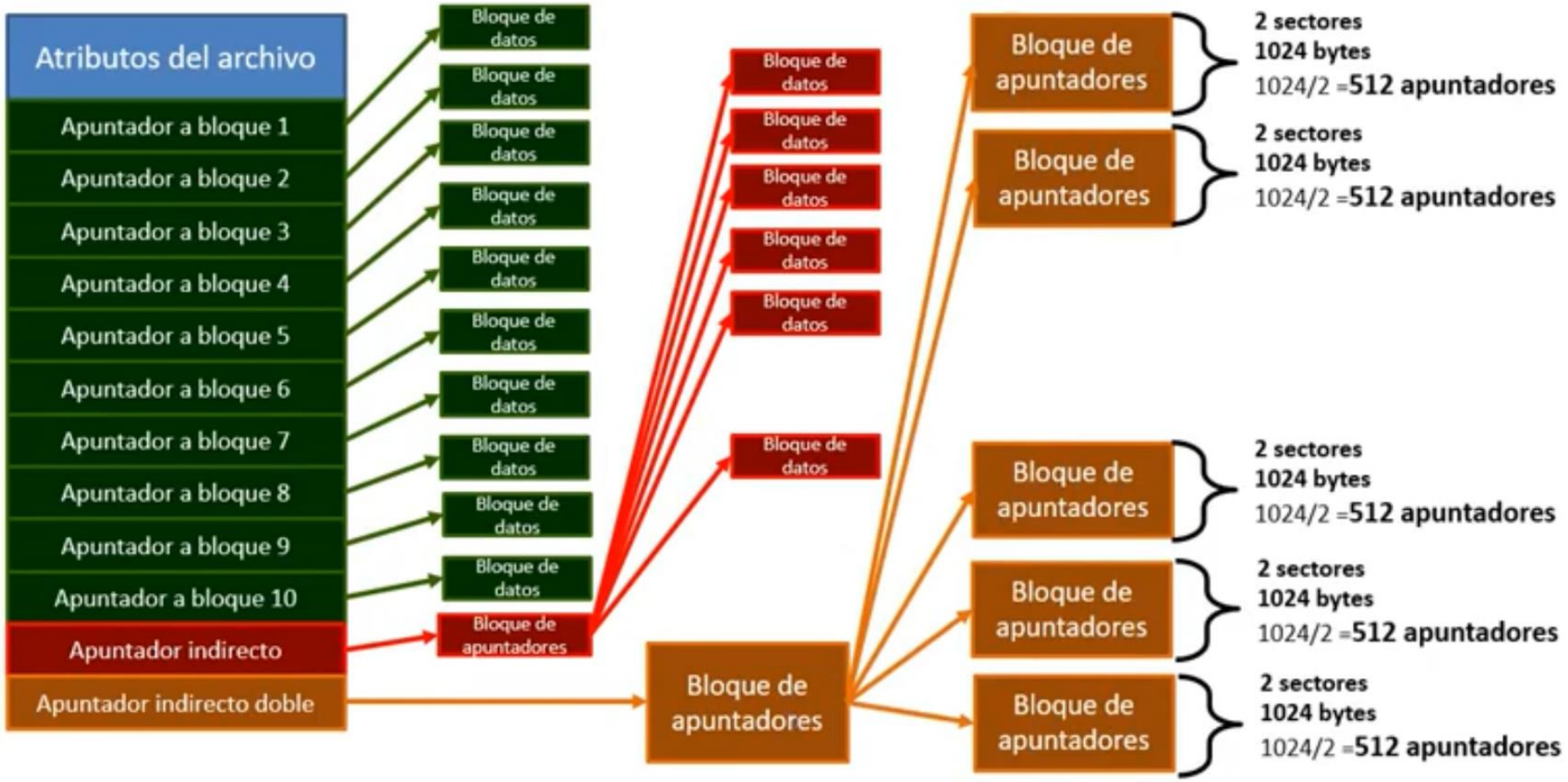
Ahora un archivo puede tener 10+512=522 bloques

- 522 Kbytes máximo por archivo
- Medio minuto de audio MP3 a 128 Kbps (Calidad media)

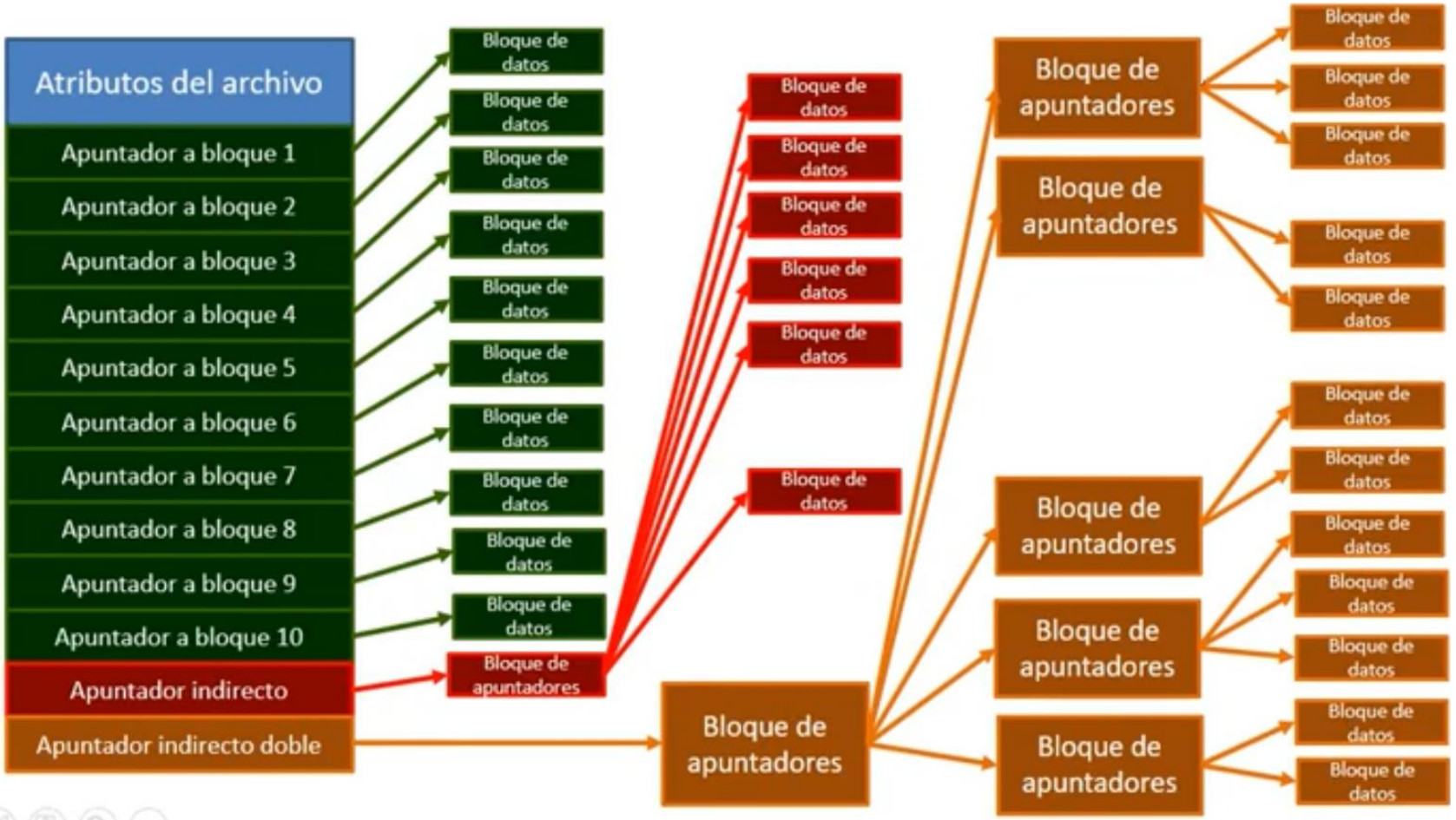
Nodos i


Atributos del archivo
Apuntador a bloque 1
Apuntador a bloque 2
Apuntador a bloque 3
Apuntador a bloque 4
Apuntador a bloque 5
Apuntador a bloque 6
Apuntador a bloque 7
Apuntador a bloque 8
Apuntador a bloque 9
Apuntador a bloque 10
Apuntador indirecto
Apuntador indirecto doble
Bloque de datos
Bloque de apuntadores
Bloque de apuntadores
2 sectores
1024 bytes
1024/2 =512 apuntadores

Nodos i


Atributos del archivo
Apuntador a bloque 1
Apuntador a bloque 2
Apuntador a bloque 3
Apuntador a bloque 4
Apuntador a bloque 5
Apuntador a bloque 6
Apuntador a bloque 7
Apuntador a bloque 8
Apuntador a bloque 9
Apuntador a bloque 10
Apuntador indirecto
Apuntador indirecto doble
Bloque de datos
Bloque de apuntadores
Bloque de apuntadores
2 sectores
1024 bytes
1024/2 =512 apuntadores

Nodos i



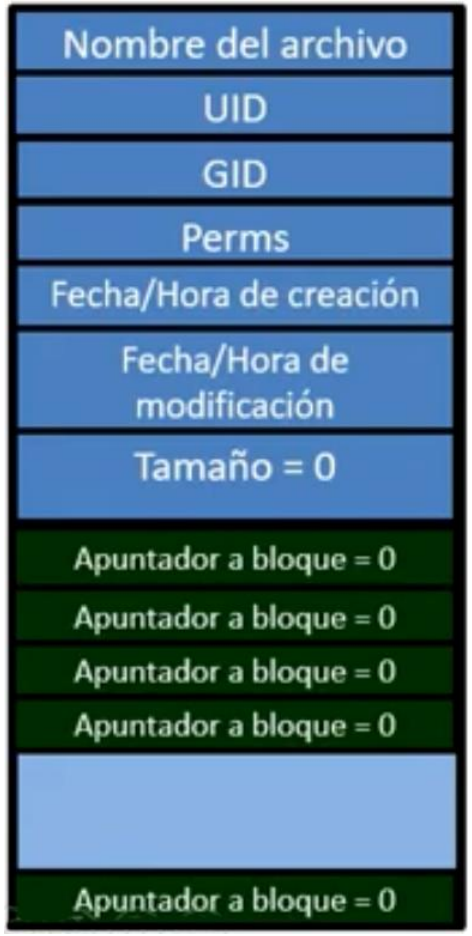
10+512+512^2 = 262666 bloques por archivo

- Mis archivos podrían medir hasta 256mb.

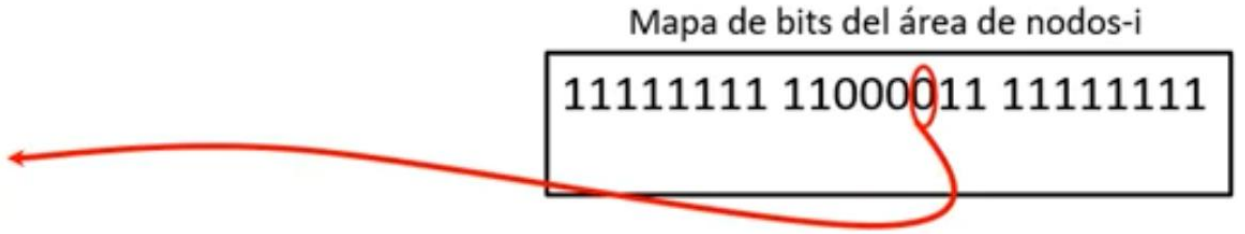
Operaciones con archivos

- Crear archivos
- Eliminar archivos
- Abrir archivos
- Lectura
- Posicionamiento dentro de un archivo
- Escritura dentro de un archivo
- Cerrar archivos
- Directorios

Operaciones con los archivos/Crear archivos

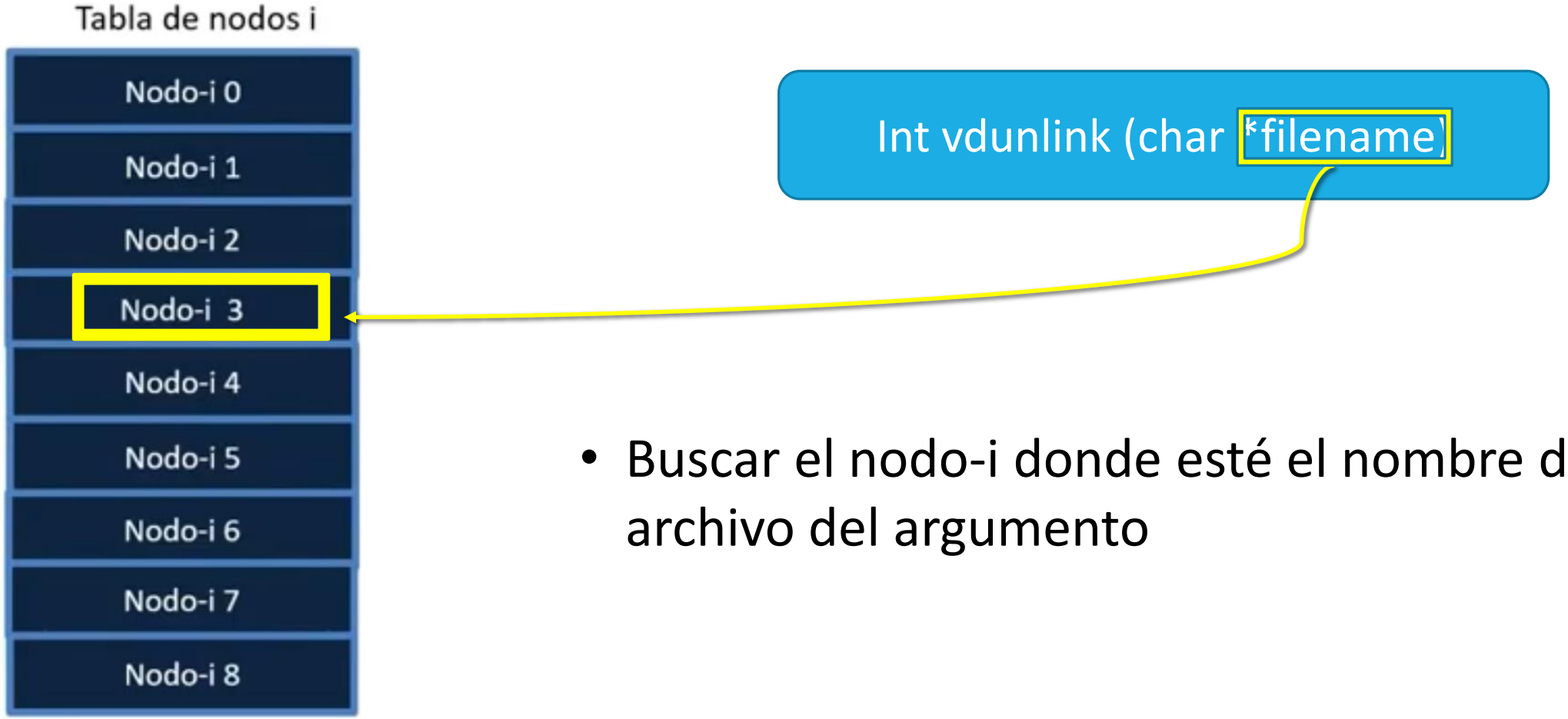


Int vdcreat(char *filename, unsigned short perms)



- Buscar en el mapa de bits de nodo-i un nodo-i libre
- Poner el nombre del archivo en el nodo-i encontrado
- Establecer el id del usuario y grupo que crea el archivo
- Establecer los permisos indicados en el segundo argumento
- Establecer fecha y hora de creación y modificación con la hora del sistema
- Establecer el tamaño del archivo en 0
- Inicializar todos los apuntadores a bloques directos, indirectos en 0s
- El archivo ya esta abierto

Operaciones con los archivos/Eliminar archivos



- Buscar el nodo-i donde esté el nombre del archivo del argumento

Operaciones con los archivos/Eliminar archivos

Int vdunlink (char *filename)

Mapa de bits del área de datos

00000001

- Buscar el nodo-i donde esté el nombre del archivo del argumento
- Recorrer los apuntadores a bloques directos para poner esos bloques como libres en el mapa de bits.
- Recorrer los apuntadores a bloques indirectos para poner esos bloques como libres en el mapa de bits.
- En el mapa de bits de nodos-i establecer el bit del nodo-i en 0

Operaciones con los archivos/Eliminar archivos

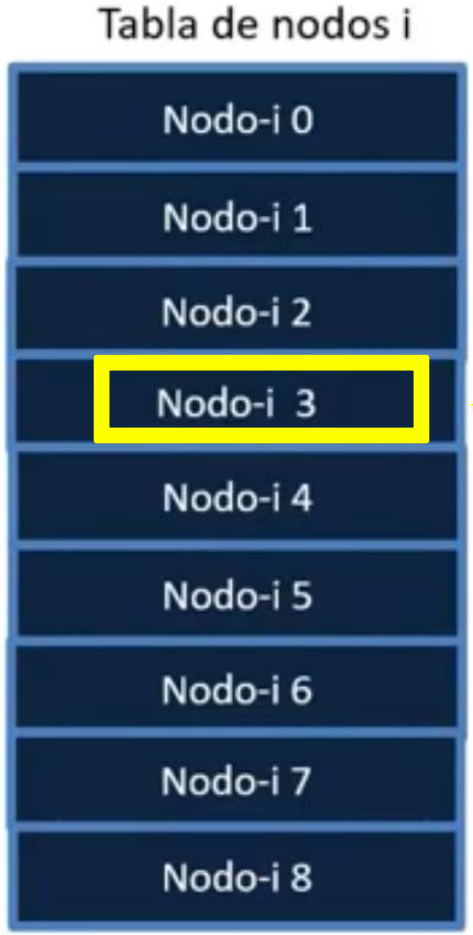
Int vdunlink (char *filename)

Mapa de bits del área de nodos i

00000111

- Buscar el nodo-i donde esté el nombre del archivo del argumento
- Recorrer los apuntadores a bloques directos para poner esos bloques como libres en el mapa de bits.
- Recorrer los apuntadores a bloques indirectos para poner esos bloques como libres en el mapa de bits.
- En el mapa de bits de nodos-i establecer el bit del nodo-i en 0

Operaciones con los archivos/Abrir archivos

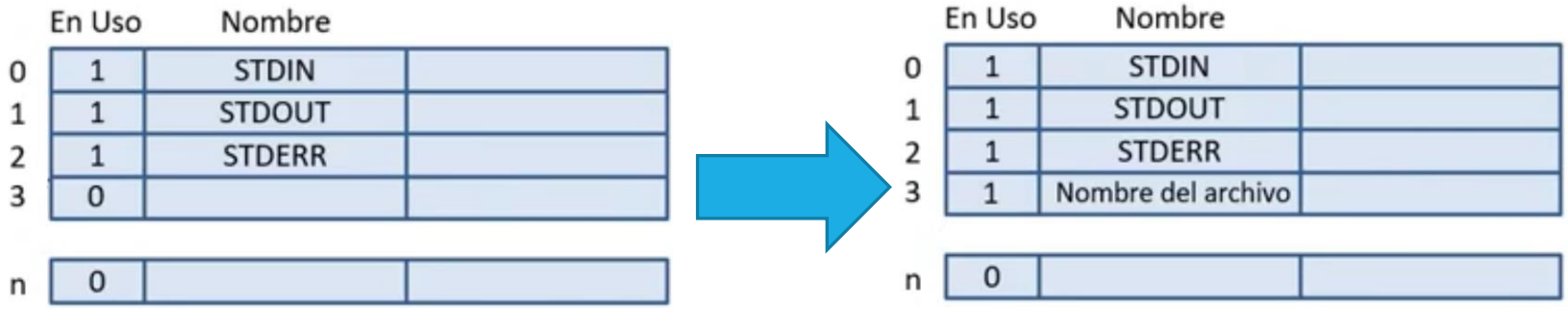


Int vdopen (char *filename, unsigned short mode)

- Buscar el nodo-i donde esté el nombre del archivo del argumento

Operaciones con los archivos/Abrir archivos

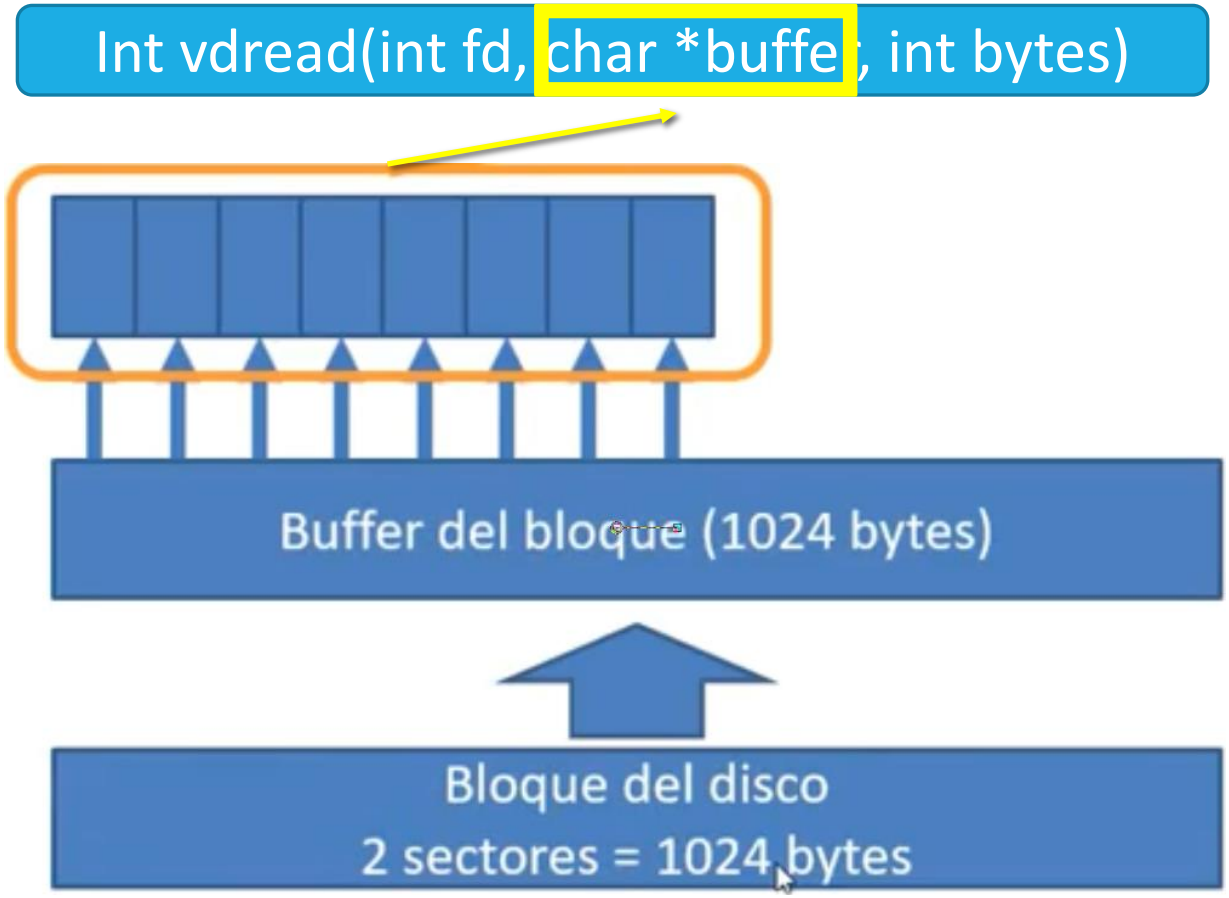
Int vdopen (char *filename, unsigned short mode)



- Buscar el nodo-i donde esté el nombre del archivo del argumento
- Abrir el nodo-i
- Transferir información del nodo-i a la tabla de archivos abiertos
- Establecer el archivo “en uso” en la tabla de archivos abiertos
- La función regresa el numero de entrada de la tabla donde está el archivo (Descriptor de archivo)

Operación con los archivos/Lectura

- Leer un bloque al buffer del bloque
- Traer los bytes solicitados al char *buffer en la función
- Cuando se terminen de leer todos los datos del buffer del bloque, leer el siguiente bloque



Operación con los archivos/Posicionamiento

Int vdseek(int fd, int offset, int whence)

Para mover el apuntador del disco

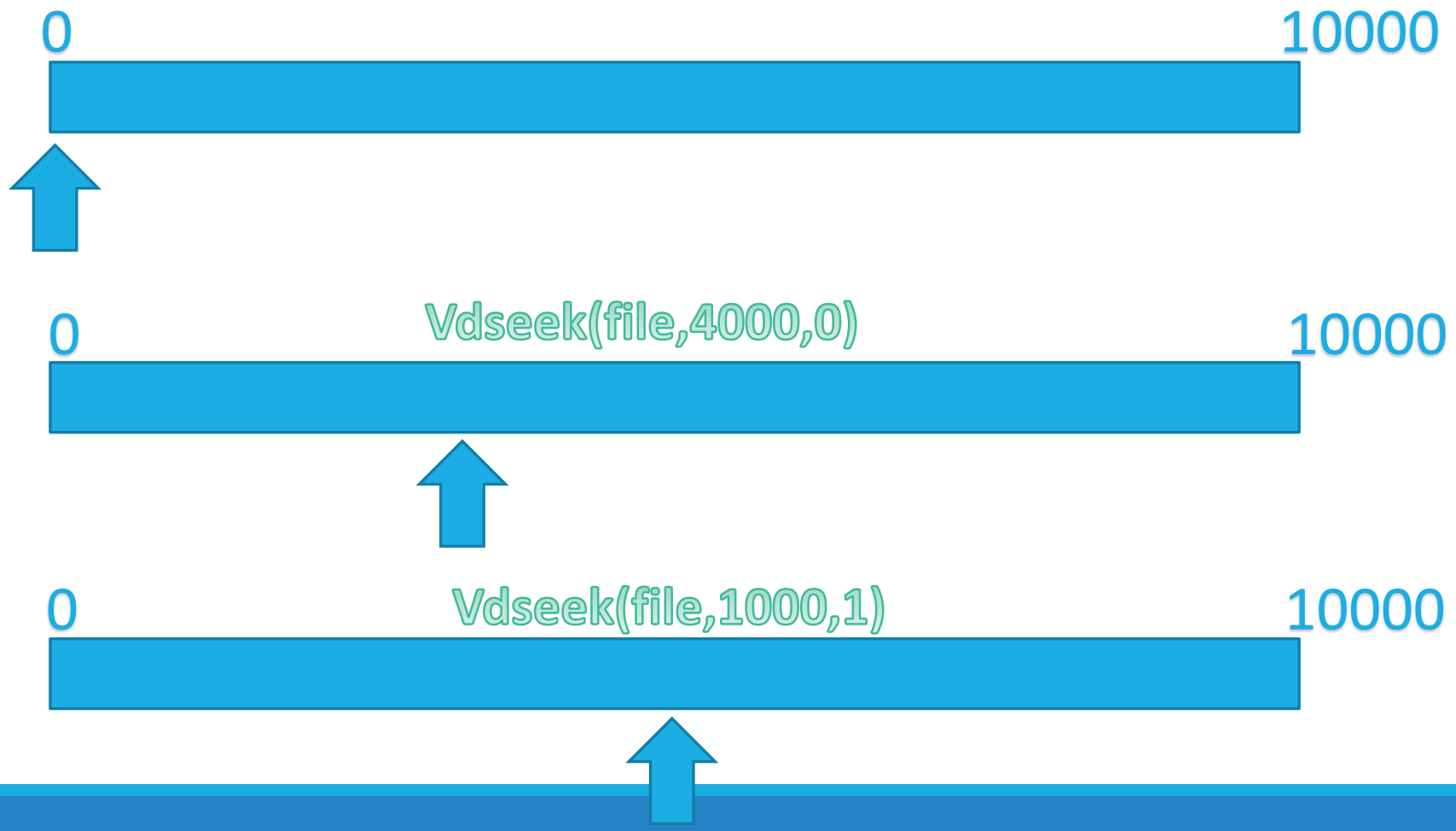
El apuntador indica a partir de que posición del archivo se va a leer o escribe

El parámetro offset indica la cantidad de bytes a moverse

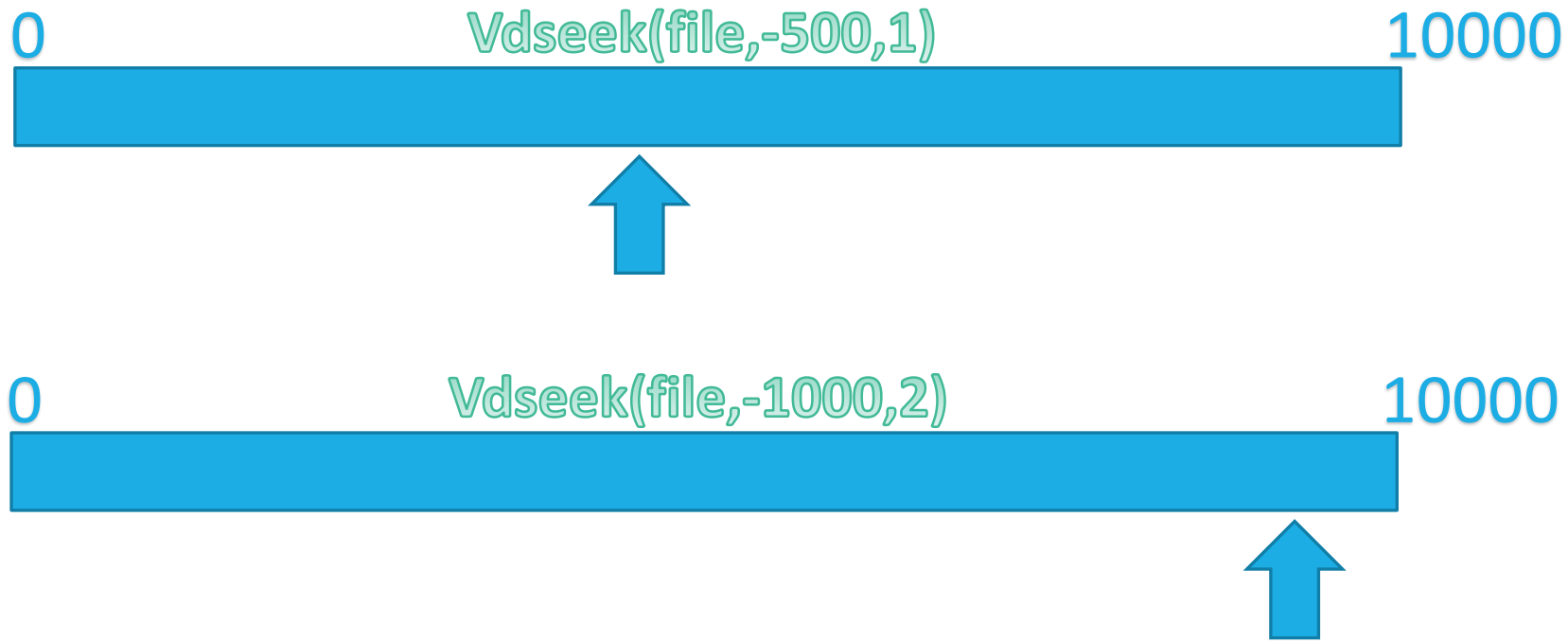
El parámetro whence indica a partir de donde se cuenta el offset

- Si es 0, será a partir del inicio
- Si es 1, será a partir de la posición actual del puntero, offset podría ser negativo
- Si es 2. será a partir del final, offset debe ser negativo

Ejemplo

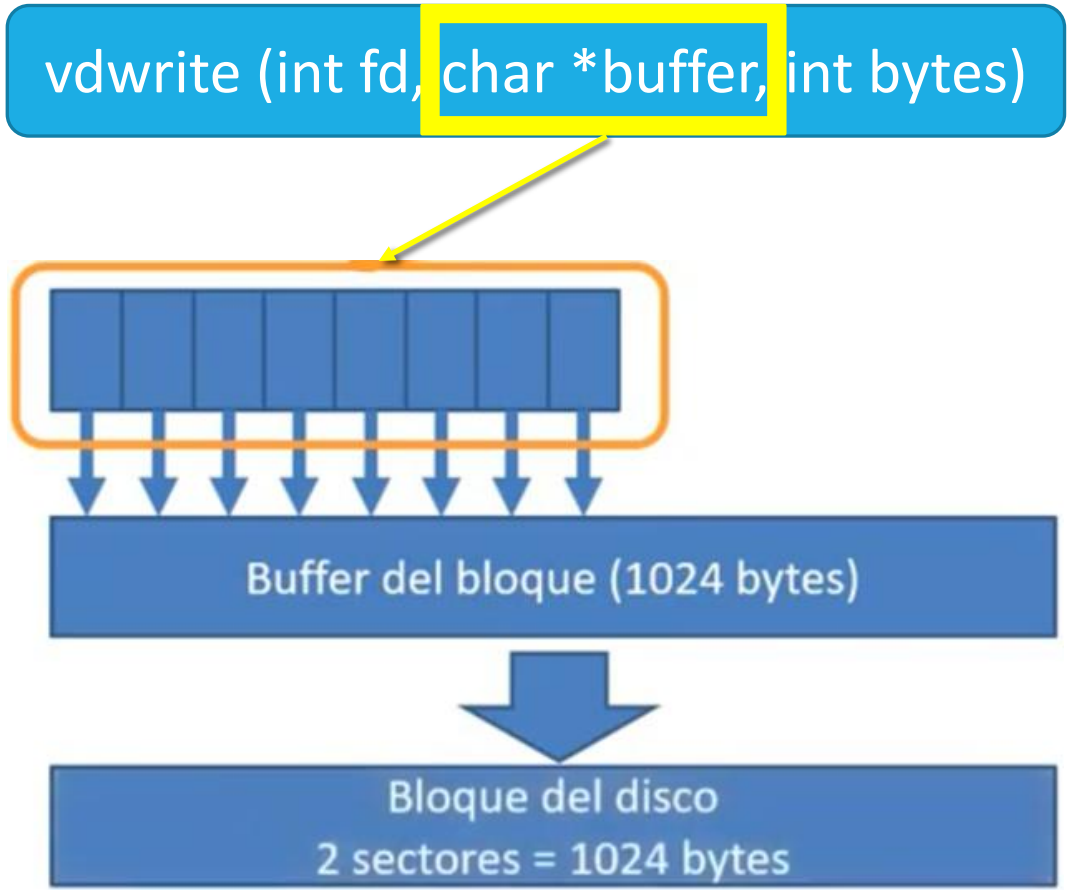


Ejemplo

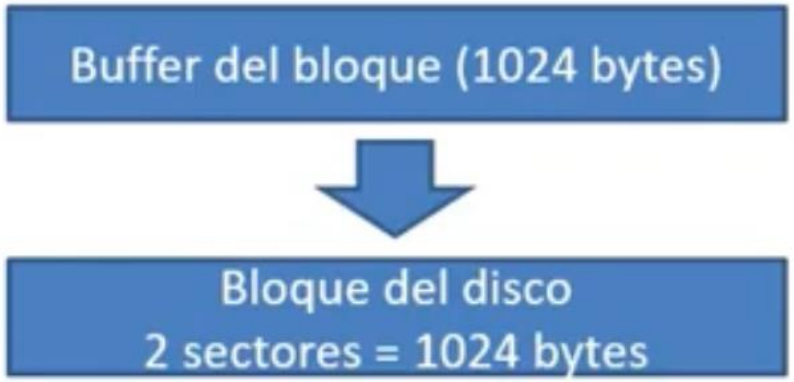


Operación con los archivos/Escritura

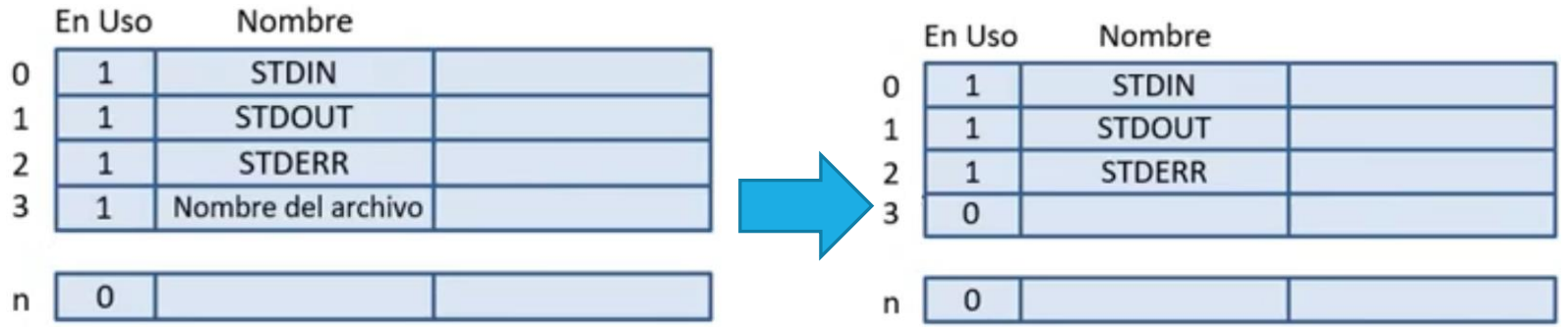
- Copiar los bytes de char *buffer al buffer del bloque
- Cuando se llena el buffer del bloque, escribir en bloque del disco



Operación con los archivos/Cerrar archivos



Int vdclose(int fd)

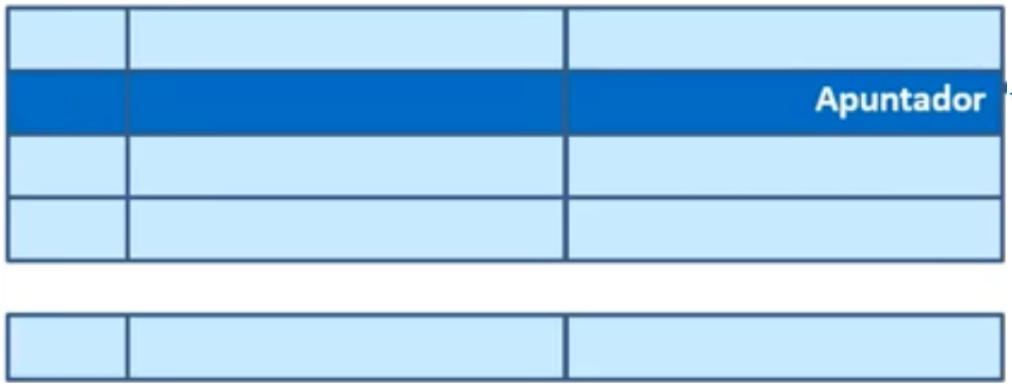


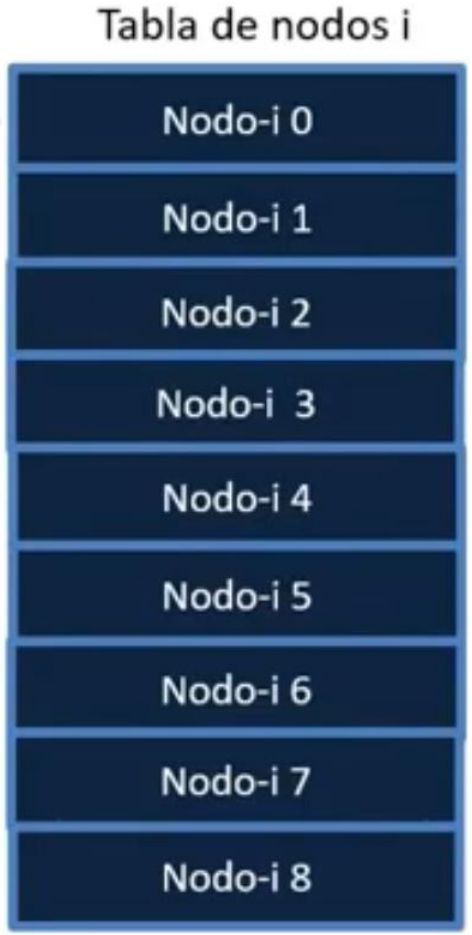
- Usar el descriptor del archivo para localizarlo en la tabla de archivos abiertos
- Vaciar buffers si hay escrituras pendientes
- Poner el bit de uso en 0
- Eliminar el nombre del archivo en la tabla

Directorios/Abrir directorio

VDDIR *vdopendir (char *dirname)

- Un directorio es una tabla de nodos-i
- Abre el directorio correspondiente a dirname indicado en el argumento y pone en una tabla de directorios abiertos.
- El valor que regresa VDDIR * es un descriptor(dirdesc) que apunta a una entrada en la tabla de directorios abiertos
- El apuntador a las entradas del directorio se posiciona en la primera entrada

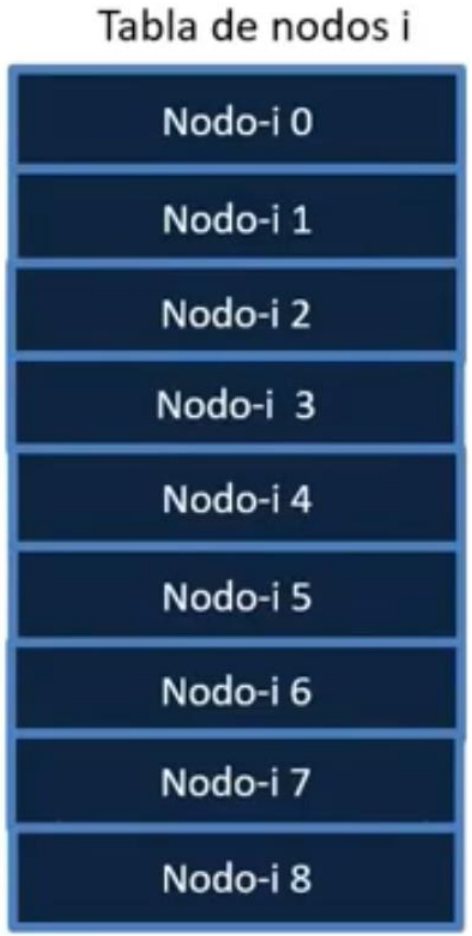




Directorios/Obtener una entrada del directorio

Struct vddirent *vdreaddir (VDDIR *dirdesc)

- Lee a memoria una estructura vddirent del directorio apuntado por dirdesc
- Cuando no hay mas entradas que leer devuelve NULL



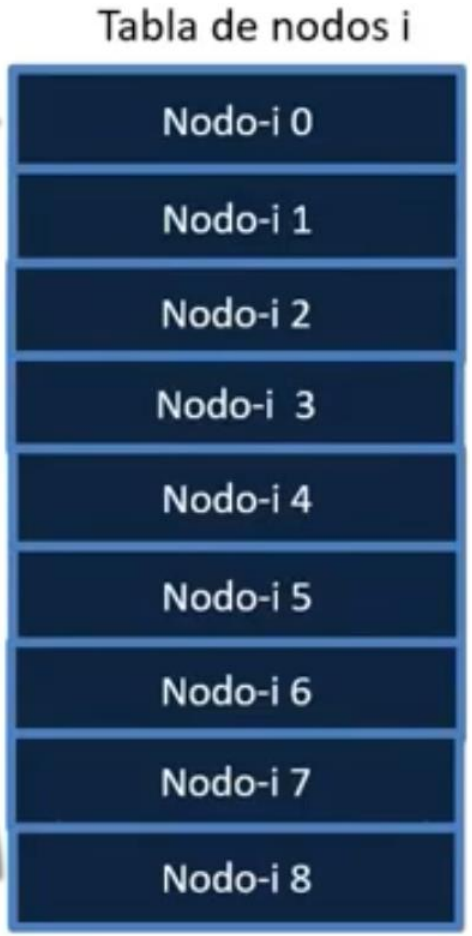
Directorios/Obtener una entrada del directorio

Struct vddirent *vdreaddir (VDDIR *dirdesc)

```
while((entry=vdreaddir(dd))!=NULL)
    printf("%s\n",entry->d_name);
```

NULL

```
while((entry=vdreaddir(dd))!=NULL)
    printf("%s\n",entry->d_name);
```



Directorios/Cerrar directorio

Int vdclosedir(VDDIR *dirdesc)

- Cierra el directorio asociado con el descriptor dirdesc
- El descriptor dirdesc ya no estará disponible después de esta llamada

Directorios/Desplegar el directorio

```
int dirv(char *dir)
{
    VDDIR *dd;
    struct vddirent *entry;

    dd=vdopendir(".");
    if(dd==NULL)
    {
        fprintf(stderr,"Error al abrir directorio\n");
        return(-1);
    }

    while((entry=vdreaddir(dd))!=NULL)
        printf("%s\n",entry->d_name);

    vdclosedir(dd);
```

Se define una variable el descriptor del directorio

Se define una apuntador a una entrada del directorio

Abrir el directorio, en este caso el punto indica directorio actual

Si vdopendir() la función regresa NULL significa que hubo error

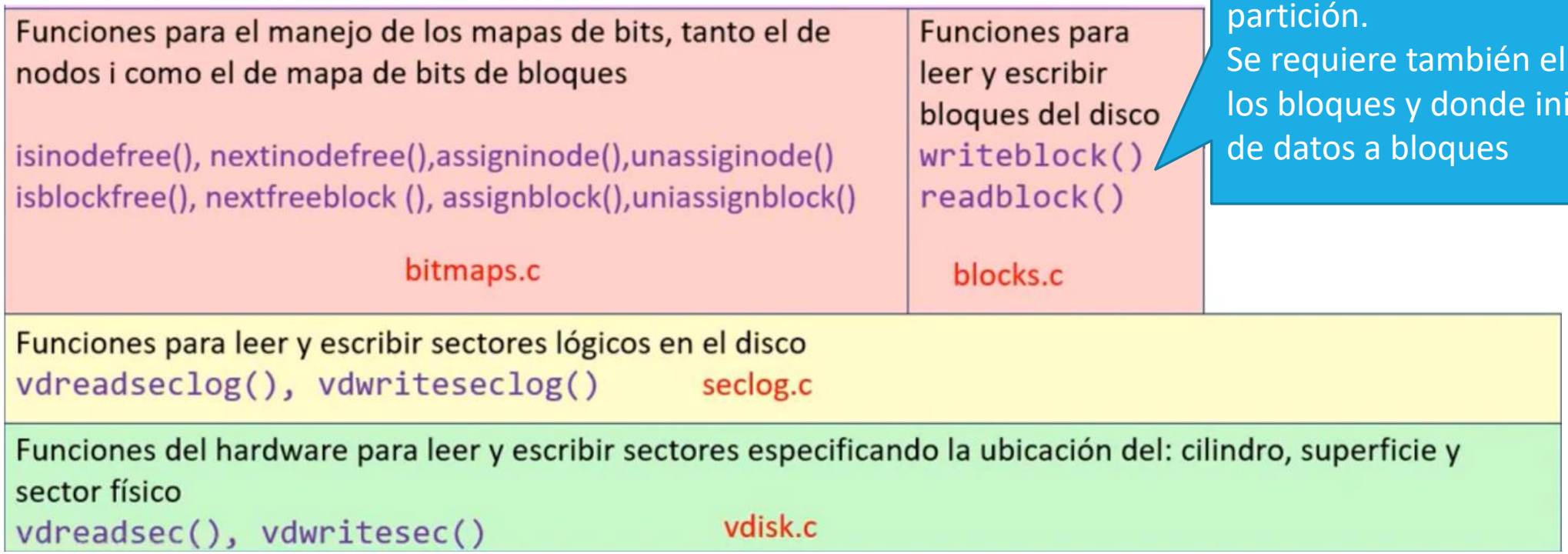
Obtener una entrada del directorio mientras la función vdreaddir() no devuelva NULL. Regresa NULL cuando no hay mas entradas en el directorio

Manda a consola el nombre del archivo en la entrada

Cerrar el directorio

Arquitectura del sistema de archivos

Arquitectura del sistema de archivos



Requiere datos de donde inician y terminan los mapas de bits, estos están en el sector de boot de la partición.
Se requiere también el tamaño de los bloques y donde inicia el área de datos a bloques

Arquitectura del sistema de archivos