Intro al taller de Ext2

Sistemas Operativos
DC - UBA - FCEN

16 de octubre de 2025

Presentación

- Vamos a programar un Sistema de Archivos `ext2`.

Presentación

- Vamos a programar un Sistema de Archivos `ext2`.
- ¿Qué debemos conocer y tener para lograrlo?

Presentación

- Vamos a programar un Sistema de Archivos `ext2`.
- ¿Qué debemos conocer y tener para lograrlo?
  - Lo que aprendimos en las clases teórica y práctica sobre `ext2`.

Presentación

- Vamos a programar un Sistema de Archivos `ext2`.
- ¿Qué debemos conocer y tener para lograrlo?
  - Lo que aprendimos en las clases teórica y práctica sobre `ext2`.
  - Un disco al que podemos acceder a cualquiera de sus bloques.

El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.

El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).

El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).

API de disco

```
int read(unsigned int lba, unsigned char * buffer);
int write(unsigned int lba, unsigned char * buffer);
```

El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).

API de disco

```
int read(unsigned int lba, unsigned char * buffer);
int write(unsigned int lba, unsigned char * buffer);
```

- ¿Qué tamaño tiene el disco?

El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).

API de disco

```
int read(unsigned int lba, unsigned char * buffer);
int write(unsigned int lba, unsigned char * buffer);
```

- ¿Qué tamaño tiene el disco?

El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).

API de disco

```
int read(unsigned int lba, unsigned char * buffer);
int write(unsigned int lba, unsigned char * buffer);
```

- ¿Qué tamaño tiene el disco? A priori no se conoce.
- ¿Qué tamaño tiene cada bloque?

El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).

API de disco

```
int read(unsigned int lba, unsigned char * buffer);
int write(unsigned int lba, unsigned char * buffer);
```

- ¿Qué tamaño tiene el disco? A priori no se conoce.
- ¿Qué tamaño tiene cada bloque?

El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).

API de disco

```
int read(unsigned int lba, unsigned char * buffer);
int write(unsigned int lba, unsigned char * buffer);
```

- ¿Qué tamaño tiene el disco? A priori no se conoce.
- ¿Qué tamaño tiene cada bloque? 1024 bytes.
- ¿Por dónde se empieza? 

Boot block

- Bloque de Booteo o Master Boot Record

Boot block

- Bloque de Booteo o Master Boot Record
- Está en la primera parte del disco. Es un espacio reservado de 1024 bytes.

Structure of a classical generic MBR

Address		Description		Size (bytes)
Hex	Dec			
+000h	+0	Bootstrap code area		446
+1BEh	+446	Partition entry #1	*Partition table* (for primary partitions)	16
+1CEh	+462	Partition entry #2		16
+1DEh	+478	Partition entry #3		16
+1EEh	+494	Partition entry #4		16
+1FEh	+510	55h	*Boot signature*[a]	2
+1FFh	+511	AAh		
			Total size: 446 + 4×16 + 2	**512**

Boot block

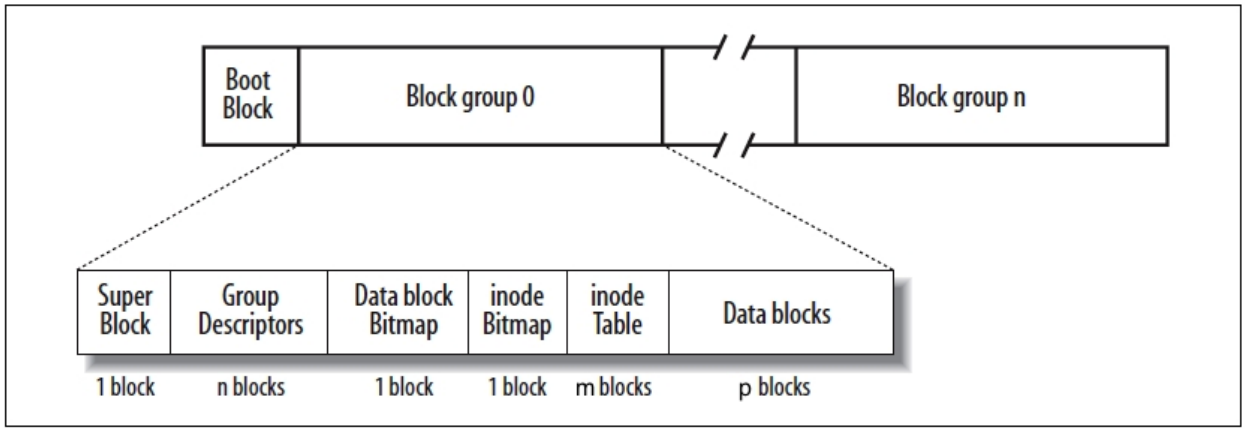
- Bloque de Booteo o Master Boot Record
- Está en la primera parte del disco. Es un espacio reservado de 1024 bytes.

Structure of a classical generic MBR

Address		Description		Size (bytes)
Hex	Dec			
+000h	+0	Bootstrap code area		446
+1BEh	+446	Partition entry #1	*Partition table* (for primary partitions)	16
+1CEh	+462	Partition entry #2		16
+1DEh	+478	Partition entry #3		16
+1EEh	+494	Partition entry #4		16
+1FEh	+510	55h	*Boot signature*[a]	2
+1FFh	+511	AAh		
			Total size: 446 + 4×16 + 2	**512**

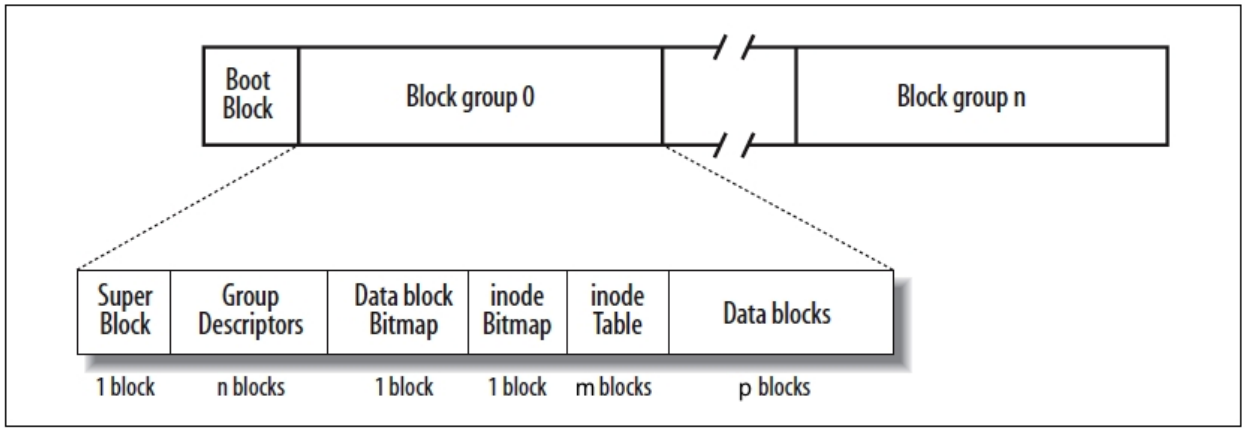
- Sólo contiene los datos necesarios para iniciar la máquina y nada más (esto es así en TODOS los sistemas de archivos).

Partición de ext2



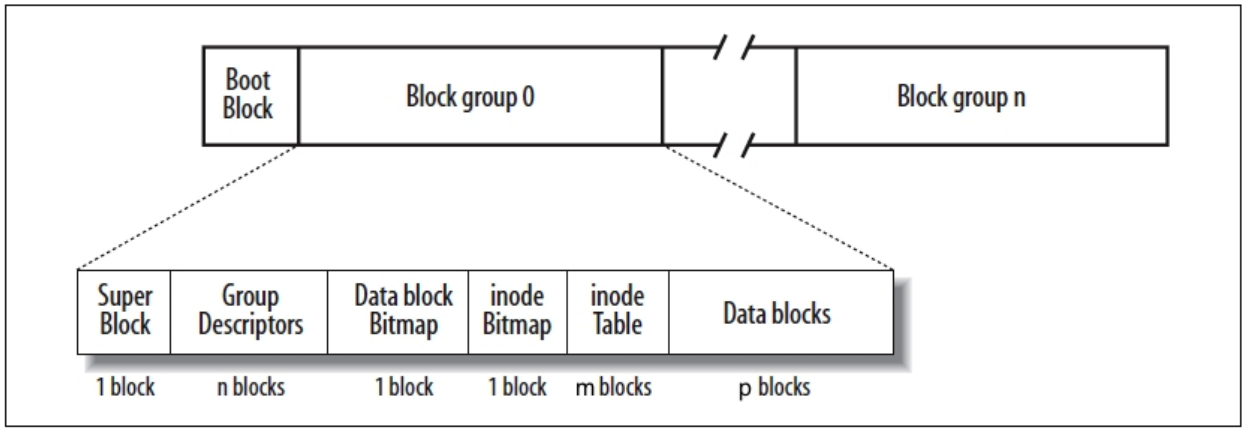
- En un sistema de archivos `ext2`, los bloques del disco están particionados en grupos de bloques contiguos. En cada grupo, hay:

Partición de ext2



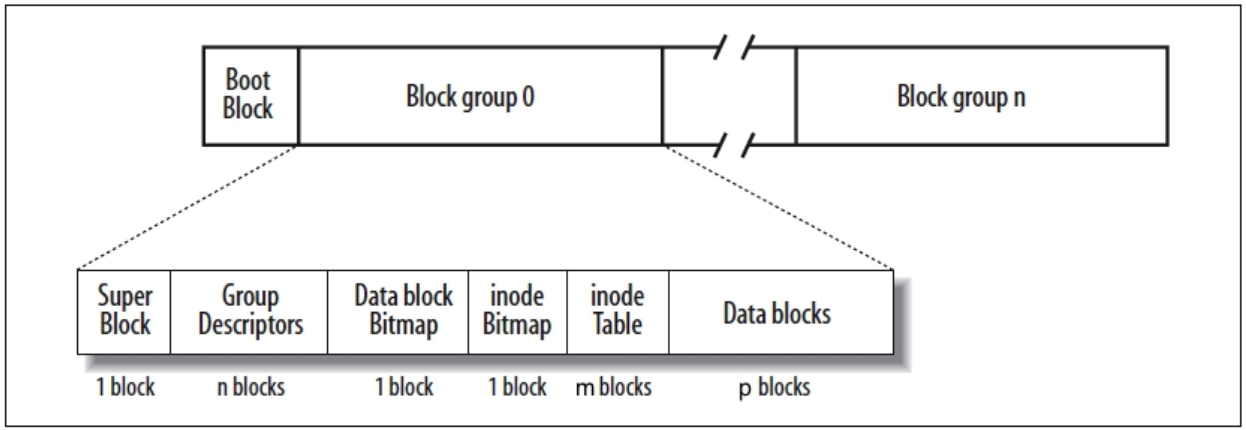
- En un sistema de archivos `ext2`, los bloques del disco están particionados en grupos de bloques contiguos. En cada grupo, hay:
  - bloques reservados para almacenamiento de datos archivos y directorios (*data blocks*).

Partición de ext2



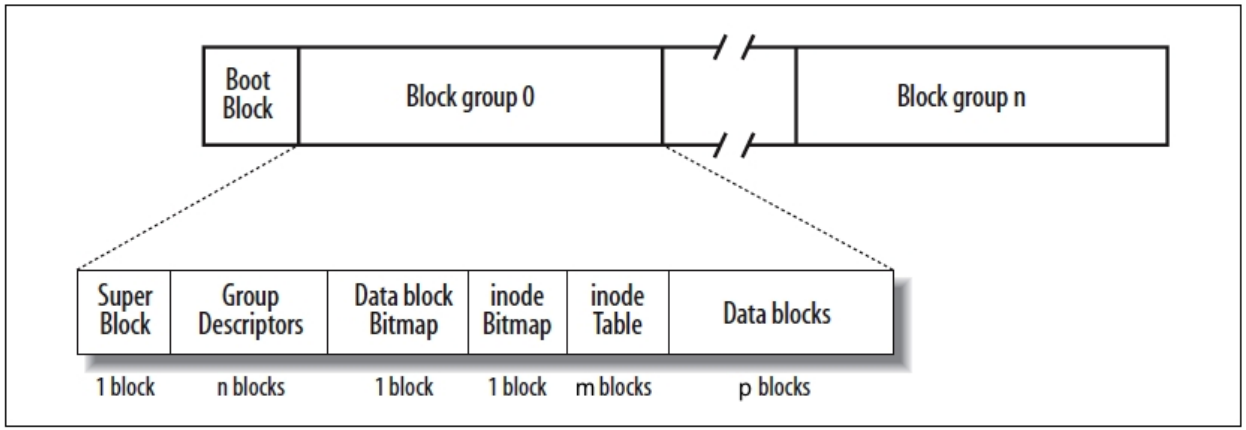
- En un sistema de archivos `ext2`, los bloques del disco están particionados en grupos de bloques contiguos. En cada grupo, hay:
  - bloques reservados para almacenamiento de datos archivos y directorios (*data blocks*).
  - bloques reservados para los inodos (*inode table*), cada bloque contiene muchos inodos. Así que los inodos están repartidos a lo largo de todo el disco.

Partición de ext2



- En un sistema de archivos `ext2`, los bloques del disco están particionados en grupos de bloques contiguos. En cada grupo, hay:
  - bloques reservados para almacenamiento de datos archivos y directorios (*data blocks*).
  - bloques reservados para los inodos (*inode table*), cada bloque contiene muchos inodos. Así que los inodos están repartidos a lo largo de todo el disco.
  - bloques reservados para información de bloques e inodos libres/ocupados (*data block bitmap* y *inode bitmap*).

Partición de ext2



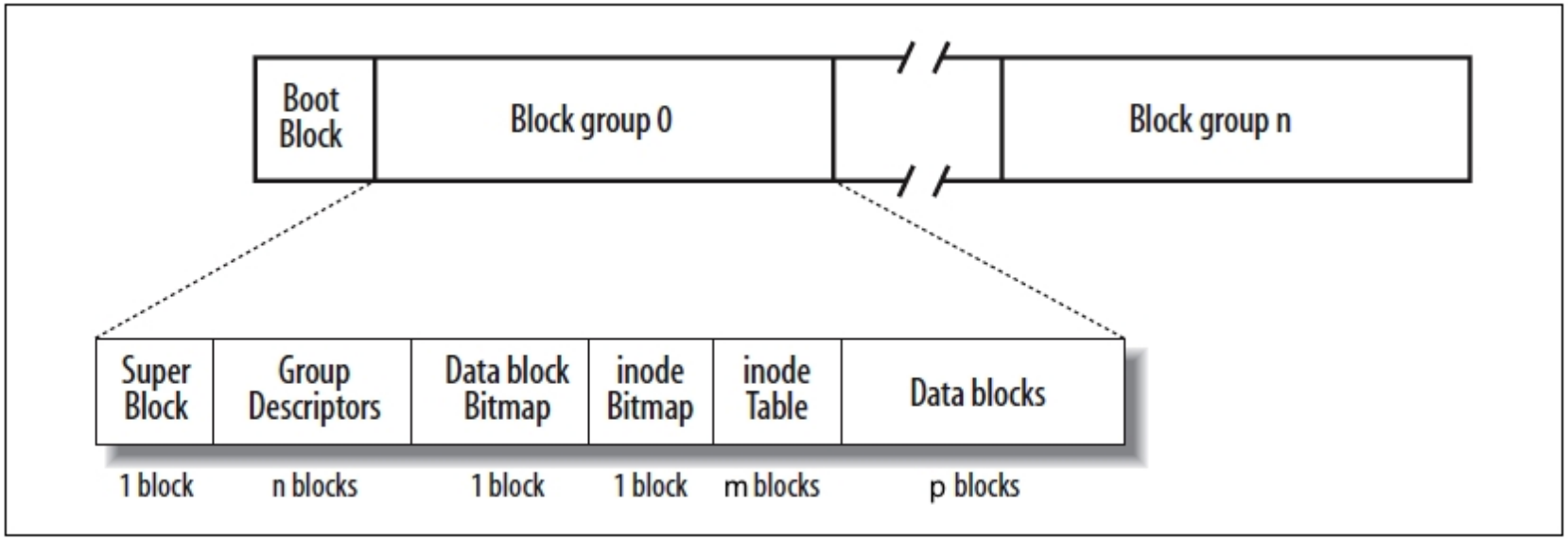
- En un sistema de archivos `ext2`, los bloques del disco están particionados en grupos de bloques contiguos. En cada grupo, hay:
  - bloques reservados para almacenamiento de datos archivos y directorios (*data blocks*).
  - bloques reservados para los inodos (*inode table*), cada bloque contiene muchos inodos. Así que los inodos están repartidos a lo largo de todo el disco.
  - bloques reservados para información de bloques e inodos libres/ocupados (*data block bitmap* y *inode bitmap*).
  - un bloque, el superbloque, que contiene información de TODO el sistema de archivos, como la cantidad de inodos, la cantidad de bloques de datos, y el inicio de la tabla de inodos.

Partición de ext2

Superblock

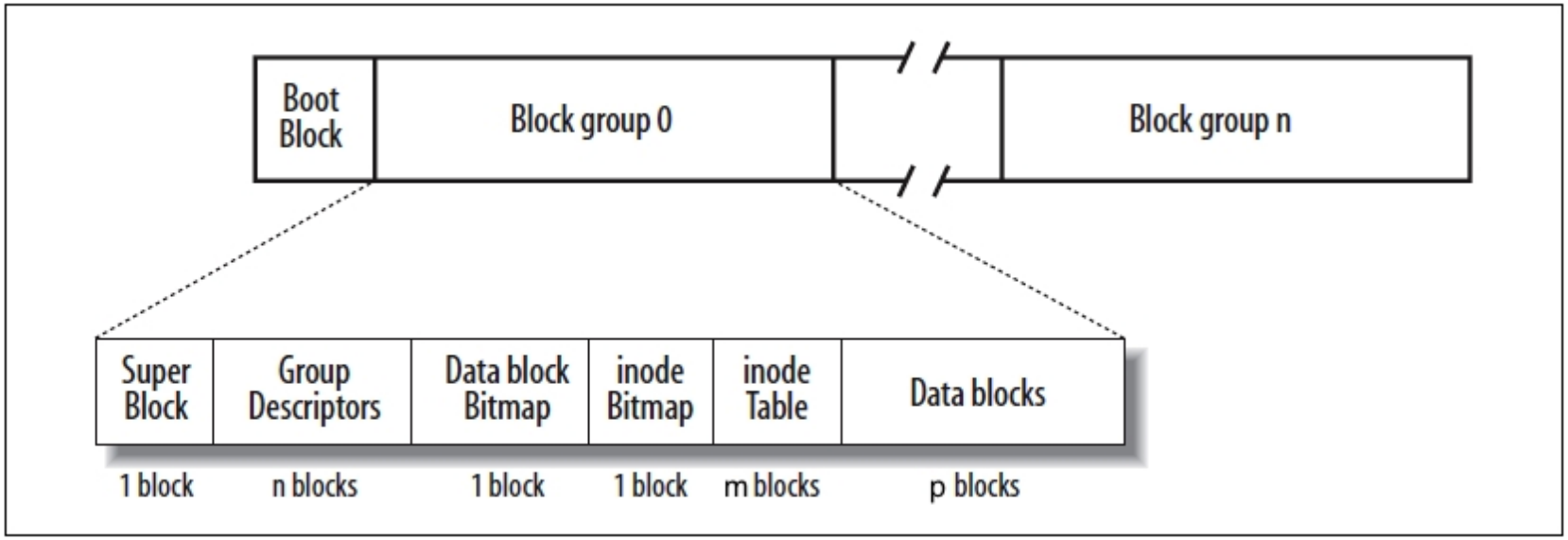
```
struct Ext2FSSuperblock {
   __le32 s_inodes_count;         /* Contador de inodos */
   __le32 s_blocks_count;         /* Contador de bloques */
   __le32 s_r_blocks_count;       /* Contador de bloques reservados */
   __le32 s_free_blocks_count;    /* Contador de bloques libres */
   __le32 s_free_inodes_count;    /* Contador de inodos libres */
   __le32 s_first_data_block;     /* Primer bloque de Datos */
   __le32 s_log_block_size;       /* Tamano del bloque */
   ...
   __le32 s_blocks_per_group;     /* Cantidad de bloques por grupo */
   ...
   __le32 s_inodes_per_group;     /* Cantidad de inodos por grupos */
   ...
   __le16 s_magic;          /* Firma magica - identifica el S.A. */
   __le32 s_first_ino;      /* Primer inodo no reservado */
   __le16 s_inode_size;     /* Tamano de la estructura del Inodo */
}
```

Estructura de ext2



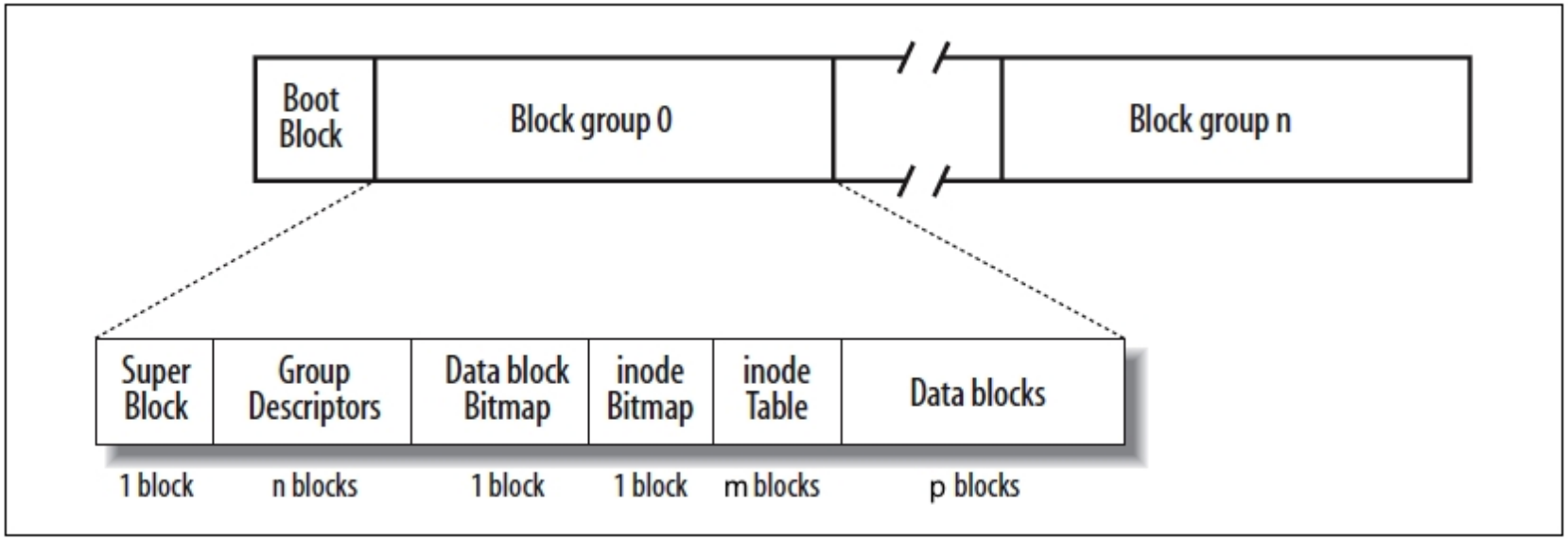
- ¿Dónde está mi archivo `/home/krypton.gis`?

Estructura de ext2



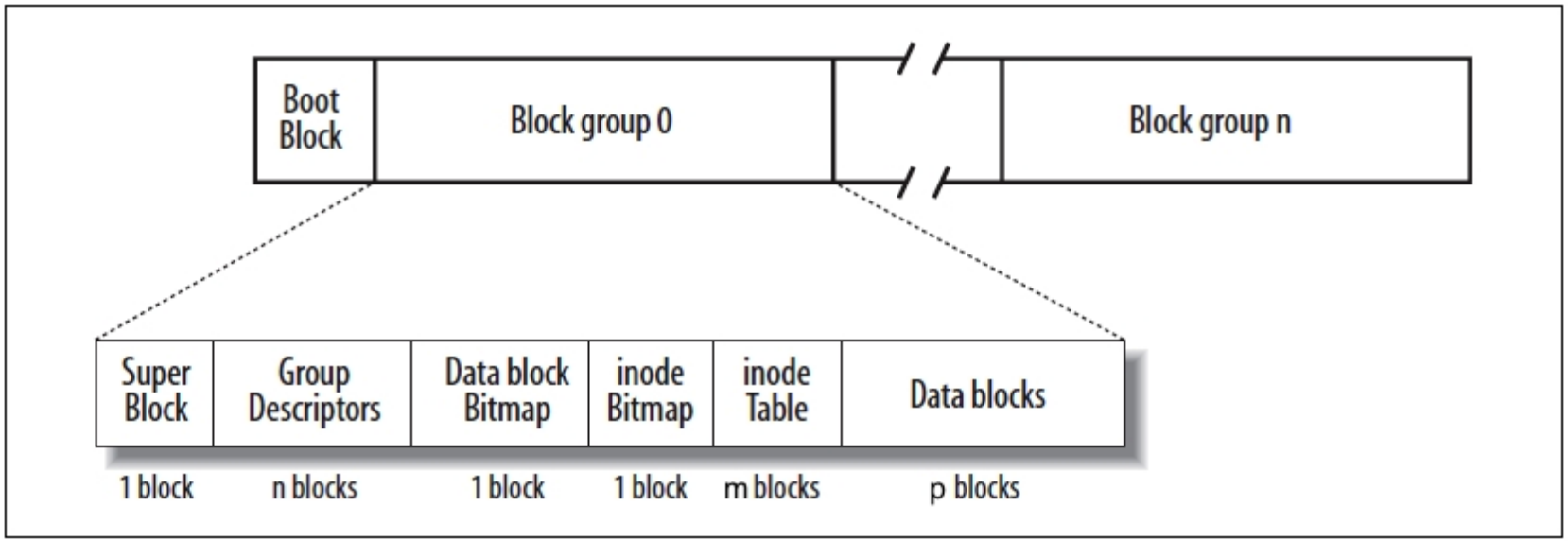
- ¿Dónde está mi archivo `/home/krypton.gis`?
- Recordemos que en `ext2` todo está representado por Inodos. ¿Cuál es el inodo de mi archivo?

Estructura de ext2



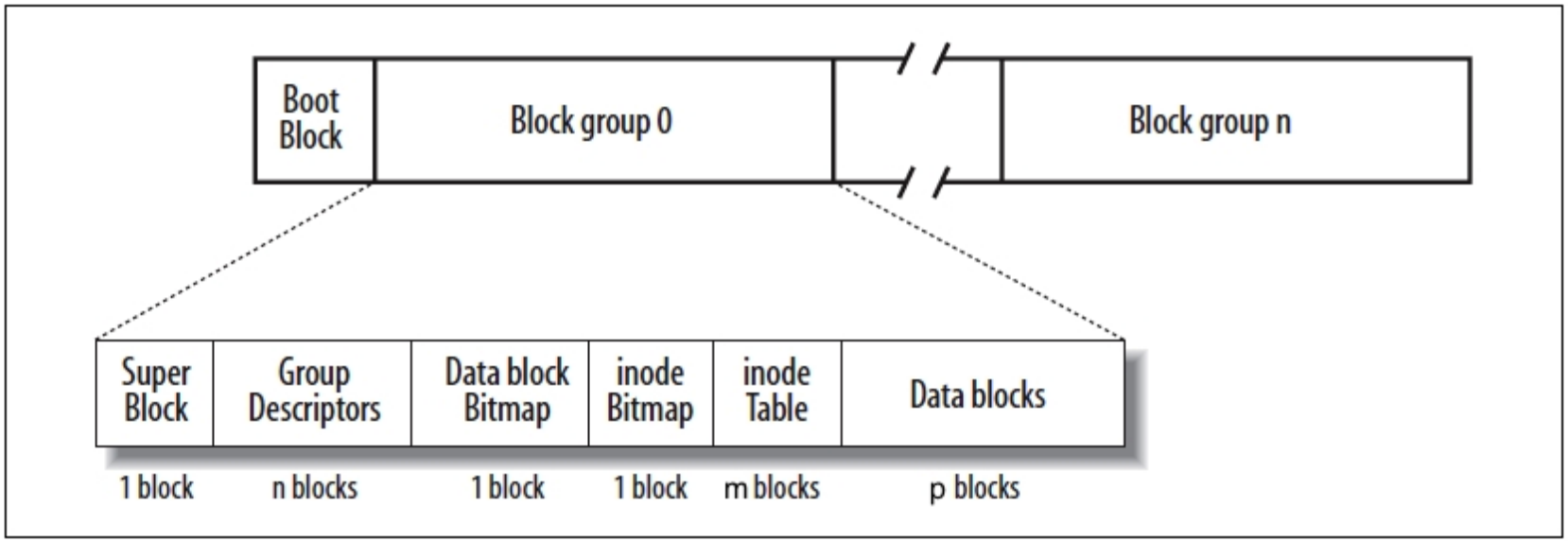
- ¿Dónde está mi archivo `/home/krypton.gis`?
- Recordemos que en `ext2` todo está representado por Inodos. ¿Cuál es el inodo de mi archivo?
- Supongamos que está en el Inodo 4483.

Estructura de ext2



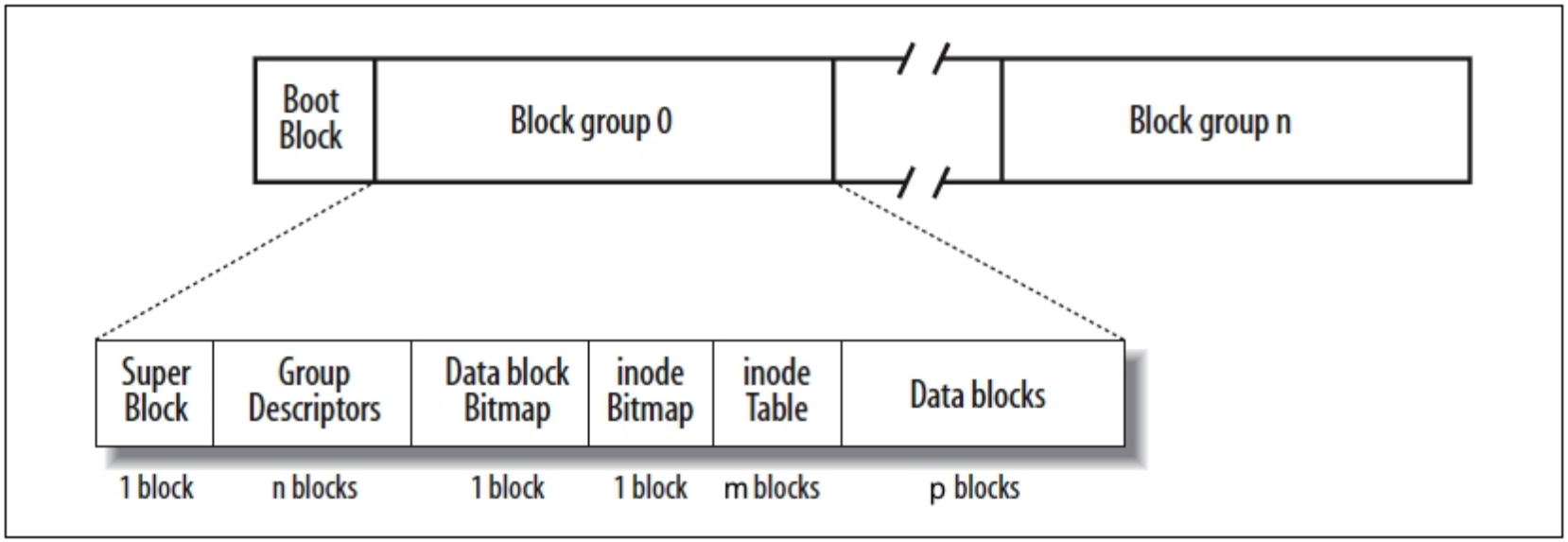
- Tenemos que calcular en qué Block Group se encuentra.

Estructura de ext2



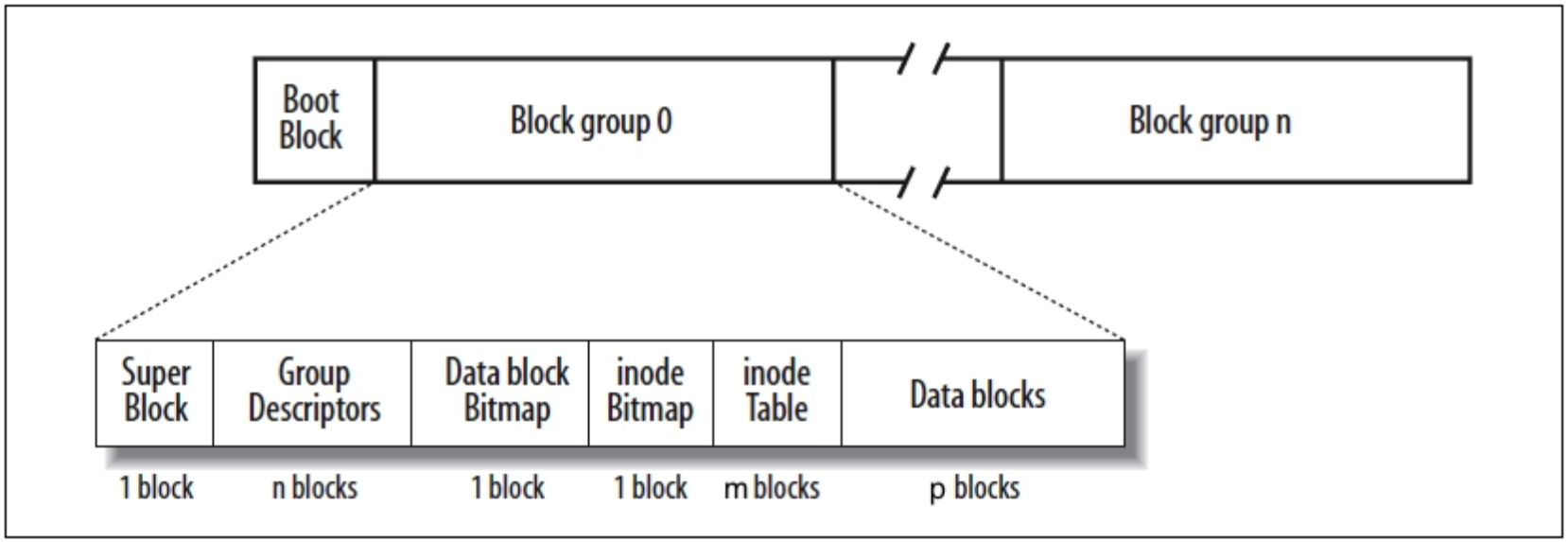
- Tenemos que calcular en qué Block Group se encuentra.
- Para eso necesitamos averiguar cuántos inodos hay por Block Group.

Estructura de ext2



- Tenemos que calcular en qué Block Group se encuentra.
- Para eso necesitamos averiguar cuántos inodos hay por Block Group.
- Esa información está en el superbloque.

Estructura de ext2



- Tenemos que calcular en qué Block Group se encuentra.
- Para eso necesitamos averiguar cuántos inodos hay por Block Group.
- Esa información está en el superbloque.
- Tenemos que hacer la división entre nuestro número de inodo y la cantidad de inodos. Eso nos va a determinar el Block Group.

Estructura de ext2

- Recordemos que cada Block Group tiene una tabla de inodos. Esta tabla está constituida por `m` bloques de tamaño `BLOCK_SIZE`. En cada bloque habrá un conjunto de inodos.

Estructura de ext2

- Recordemos que cada Block Group tiene una tabla de inodos. Esta tabla está constituida por `m` bloques de tamaño `BLOCK_SIZE`. En cada bloque habrá un conjunto de inodos.
- Tenemos que averiguar que número de inodo corresponde de nuestra tabla de inodo. Para eso usamos el módulo con la cantidad de inodos.

Estructura de ext2

- Recordemos que cada Block Group tiene una tabla de inodos. Esta tabla está constituida por `m` bloques de tamaño `BLOCK_SIZE`. En cada bloque habrá un conjunto de inodos.
- Tenemos que averiguar que número de inodo corresponde de nuestra tabla de inodo. Para eso usamos el módulo con la cantidad de inodos.
- Luego tenemos que localizar en qué bloque está nuestro inodo a buscar.

Estructura de ext2

- Recordemos que cada Block Group tiene una tabla de inodos. Esta tabla está constituida por `m` bloques de tamaño `BLOCK_SIZE`. En cada bloque habrá un conjunto de inodos.
- Tenemos que averiguar que número de inodo corresponde de nuestra tabla de inodo. Para eso usamos el módulo con la cantidad de inodos.
- Luego tenemos que localizar en qué bloque está nuestro inodo a buscar.
- Una vez conseguido, tenemos que leer ese bloque de disco y luego del conjunto de inodos, conseguir el que nos interesa.

Estructura de ext2

- Recordemos que cada Block Group tiene una tabla de inodos. Esta tabla está constituida por `m` bloques de tamaño `BLOCK_SIZE`. En cada bloque habrá un conjunto de inodos.
- Tenemos que averiguar que número de inodo corresponde de nuestra tabla de inodo. Para eso usamos el módulo con la cantidad de inodos.
- Luego tenemos que localizar en qué bloque está nuestro inodo a buscar.
- Una vez conseguido, tenemos que leer ese bloque de disco y luego del conjunto de inodos, conseguir el que nos interesa.

Estructura de ext2

- Recordemos que cada Block Group tiene una tabla de inodos. Esta tabla está constituida por `m` bloques de tamaño `BLOCK_SIZE`. En cada bloque habrá un conjunto de inodos.
- Tenemos que averiguar que número de inodo corresponde de nuestra tabla de inodo. Para eso usamos el módulo con la cantidad de inodos.
- Luego tenemos que localizar en qué bloque está nuestro inodo a buscar.
- Una vez conseguido, tenemos que leer ese bloque de disco y luego del conjunto de inodos, conseguir el que nos interesa.

Funciones útiles

```
unsigned int blockgroup_for_inode(unsigned int inode);
unsigned int blockgroup_inode_index(unsigned int inode);
```

Inodo

- La representación de un archivo.

Inodo

- La representación de un archivo.
- Un archivo puede ser un archivo regular, un directorio, un pipe, un socket, un device, etc.

Inodo

- La representación de un archivo.
- Un archivo puede ser un archivo regular, un directorio, un pipe, un socket, un device, etc.
- A bajo nivel, en este taller, es un struct `FSInode`.

Inodo

- La representación de un archivo.
- Un archivo puede ser un archivo regular, un directorio, un pipe, un socket, un device, etc.
- A bajo nivel, en este taller, es un struct `FSInode`.

Inodo

- La representación de un archivo.
- Un archivo puede ser un archivo regular, un directorio, un pipe, un socket, un device, etc.
- A bajo nivel, en este taller, es un struct `FSInode`.

Inode

```
struct Ext2FSInode {
    unsigned short mode;
    unsigned short uid;
    unsigned int size;
    unsigned int atime;
    unsigned int ctime;
    unsigned int mtime;
    unsigned int dtime;
    unsigned short gid;
    unsigned short links_count;
    unsigned int blocks;
    unsigned int flags;
    unsigned int os_dependant_1;
    unsigned int block[15];
    unsigned int generation;
    unsigned int file_acl;
    unsigned int directory_acl;
    unsigned int faddr;
    unsigned int os_dependant_2[3];
}
```

Inodo

- ¿Dónde están los datos?

Inode

```
struct  Ext2FSInode {
    unsigned short mode;
    unsigned short uid;
    unsigned int size;
    unsigned int atime;
    unsigned int ctime;
    unsigned int mtime;
    unsigned int dtime;
    unsigned short gid;
    unsigned short links_count;
    unsigned int blocks;
    unsigned int flags;
    unsigned int os_dependant_1;
    unsigned int block[15];
    unsigned int generation;
    unsigned int file_acl;
    unsigned int directory_acl;
    unsigned int faddr;
    unsigned int os_dependant_2[3];
}
```

Inodo

- ¿Dónde están los datos?

- ¿Dónde está el nombre del archivo?


Inode

```
struct  Ext2FSInode {
    unsigned short mode;
    unsigned short uid;
    unsigned int size;
    unsigned int atime;
    unsigned int ctime;
    unsigned int mtime;
    unsigned int dtime;
    unsigned short gid;
    unsigned short links_count;
    unsigned int blocks;
    unsigned int flags;
    unsigned int os_dependant_1;
    unsigned int block[15];
    unsigned int generation;
    unsigned int file_acl;
    unsigned int directory_acl;
    unsigned int faddr;
    unsigned int os_dependant_2[3];
}
```

Inodo

- ¿Dónde están los datos? 
- ¿Dónde está el nombre del archivo?  A nivel de usuario no se hace referencia a números de inodos.

Inode

```
struct  Ext2FSInode {
    unsigned short mode;
    unsigned short uid;
    unsigned int size;
    unsigned int atime;
    unsigned int ctime;
    unsigned int mtime;
    unsigned int dtime;
    unsigned short gid;
    unsigned short links_count;
    unsigned int blocks;
    unsigned int flags;
    unsigned int os_dependant_1;
    unsigned int block[15];
    unsigned int generation;
    unsigned int file_acl;
    unsigned int directory_acl;
    unsigned int faddr;
    unsigned int os_dependant_2[3];
}
```

Inodo

- ¿Dónde están los datos? 
- ¿Dónde está el nombre del archivo?  A nivel de usuario no se hace referencia a números de inodos.
- ¿El inodo directorio qué struct usa? 

Inode

```
struct  Ext2FSInode {
    unsigned short mode;
    unsigned short uid;
    unsigned int size;
    unsigned int atime;
    unsigned int ctime;
    unsigned int mtime;
    unsigned int dtime;
    unsigned short gid;
    unsigned short links_count;
    unsigned int blocks;
    unsigned int flags;
    unsigned int os_dependant_1;
    unsigned int block[15];
    unsigned int generation;
    unsigned int file_acl;
    unsigned int directory_acl;
    unsigned int faddr;
    unsigned int os_dependant_2[3];
}
```

Inodo

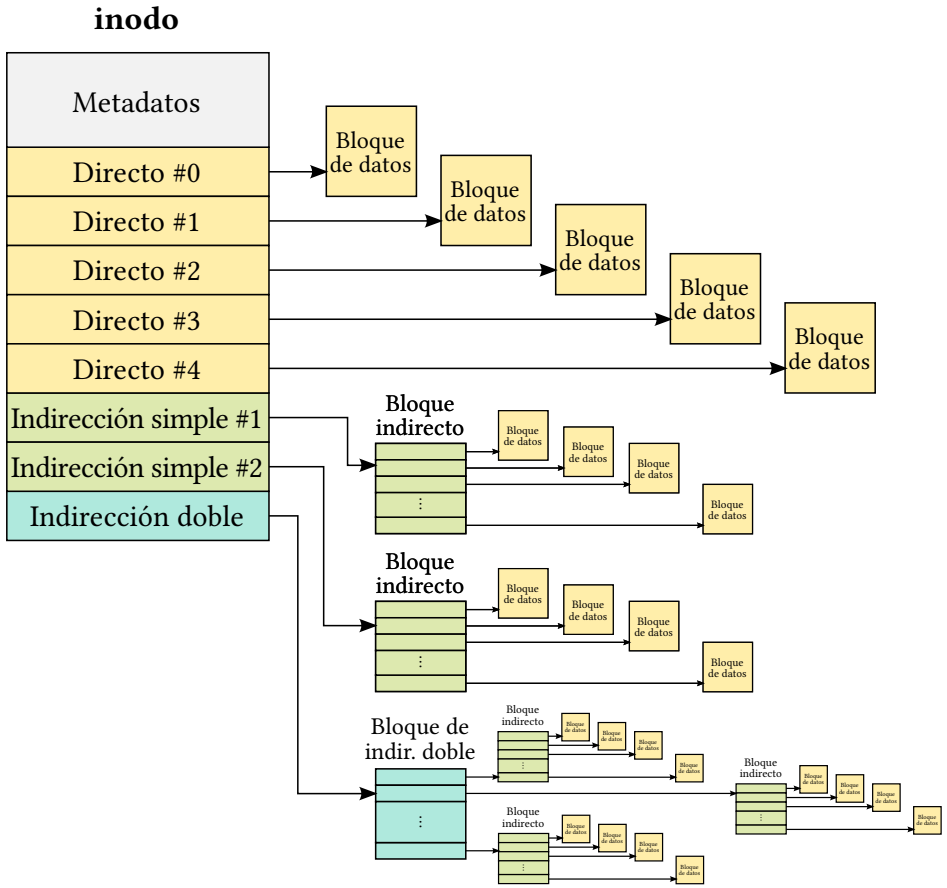
Datos

- 15 punteros a bloques con distintos sabores:

Inodo

Datos

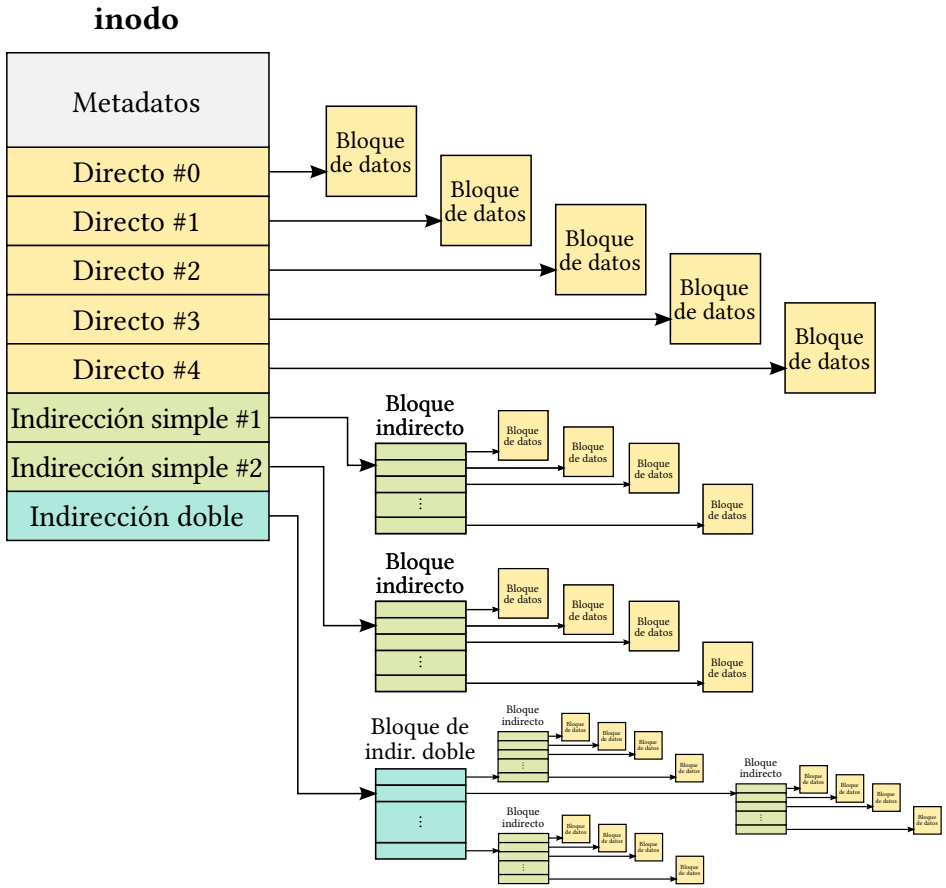
- 15 punteros a bloques con distintos sabores:
  - 12 punteros a bloques de datos directos
  - 1 puntero indirecto a bloque de datos
  - 1 puntero con una doble indirección a bloque de datos
  - 1 puntero con una triple indirección a bloque de datos
- Solo vamos a implementar hasta la doble indirección ⚠



Inodo

Datos

- 15 punteros a bloques con distintos sabores:
  - 12 punteros a bloques de datos directos
  - 1 puntero indirecto a bloque de datos
  - 1 puntero con una doble indirección a bloque de datos
  - 1 puntero con una triple indirección a bloque de datos
- Solo vamos a implementar hasta la doble indirección ⚠

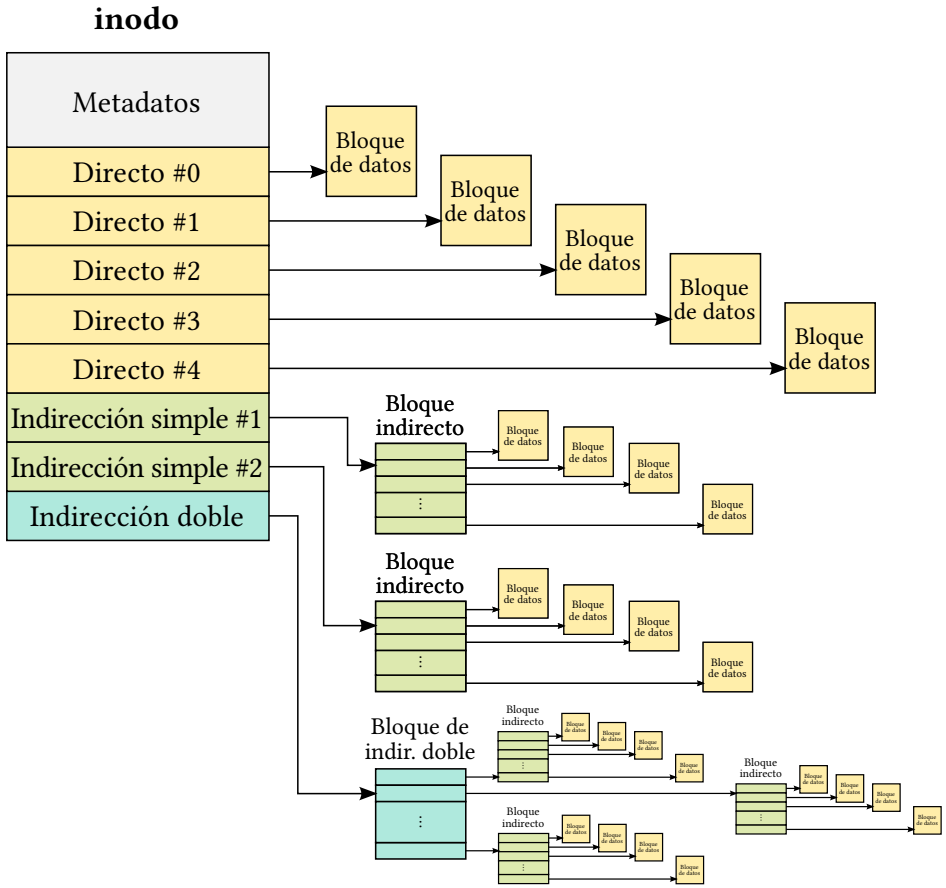


- ¿Son punteros a direcciones de memoria? ⚠

Inodo

Datos

- 15 punteros a bloques con distintos sabores:
  - 12 punteros a bloques de datos directos
  - 1 puntero indirecto a bloque de datos
  - 1 puntero con una doble indirección a bloque de datos
  - 1 puntero con una triple indirección a bloque de datos
- Solo vamos a implementar hasta la doble indirección ⚠



- ¿Son punteros a direcciones de memoria? ⚠
- ¿Los bloques a los que apuntan, están en memoria o en disco? ⚠

Inodo

Directorios

- Es un Inodo IGUAL que cualquier otro.

Inodo

Directorios

- Es un Inodo IGUAL que cualquier otro.
- Es decir, tiene la misma estructura `Ext2FSInode`.

Inodo

Directorios

- Es un Inodo IGUAL que cualquier otro.
- Es decir, tiene la misma estructura `Ext2FSInode`.
- Entonces, ¿dónde está la lista de archivos de mi directorio?

Inodo

Directorios

- Es un Inodo IGUAL que cualquier otro.
- Es decir, tiene la misma estructura `Ext2FSInode`.
- Entonces, ¿dónde está la lista de archivos de mi directorio?
- En los bloques de datos. 

Directory Entry

```
struct Ext2FSDirEntry {
    unsigned int inode;
    unsigned short record_length;
    unsigned char name_length;
    unsigned char file_type;
    char name[];
};
```

Directory Entry

```
struct Ext2FSDirEntry {
    unsigned int inode;
    unsigned short record_length;
    unsigned char name_length;
    unsigned char file_type;
    char name[];
};
```

- **Los datos** del Inodo son una lista de structs `Ext2FSDirEntry`.

Directory Entry

```
struct Ext2FSDirEntry {
    unsigned int inode;
    unsigned short record_length;
    unsigned char name_length;
    unsigned char file_type;
    char name[];
};
```

- **Los datos** del Inodo son una lista de structs `Ext2FSDirEntry`.
- Cada struct tiene tamaño variable. 

Directory Entry

```
struct Ext2FSDirEntry {
    unsigned int inode;
    unsigned short record_length;
    unsigned char name_length;
    unsigned char file_type;
    char name[];
};
```

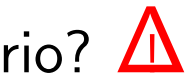
- **Los datos** del Inodo son una lista de structs `Ext2FSDirEntry`.
- Cada struct tiene tamaño variable. 
- ¡Puede haber un caso borde! 

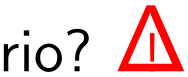
Directory Entry

```
struct Ext2FSDirEntry {
    unsigned int inode;
    unsigned short record_length;
    unsigned char name_length;
    unsigned char file_type;
    char name[];
};
```

- **Los datos** del Inodo son una lista de structs `Ext2FSDirEntry`.
- Cada struct tiene tamaño variable. 
- ¡Puede haber un caso borde! 
- Puede pasar que el struct `DirEntry` quede dividido en dos bloques.

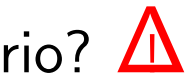
- ¿Qué pasa si busco un archivo que no existe en este directorio? 

- ¿Qué pasa si busco un archivo que no existe en este directorio? 
- Debemos buscar en cada direntry que se encuentre en el directorio hasta llegar al final.

- ¿Qué pasa si busco un archivo que no existe en este directorio? 
- Debemos buscar en cada direntry que se encuentre en el directorio hasta llegar al final.
- ¿Qué condición de corte usamos?

Inodo

Directorios

- ¿Qué pasa si busco un archivo que no existe en este directorio? 
- Debemos buscar en cada direntry que se encuentre en el directorio hasta llegar al final.
- ¿Qué condición de corte usamos?
- El campo `size` del inodo nos dice la cantidad de bytes que usa ese archivo.

Consignas del taller

Completar la implementación de los siguientes métodos:

1. `Ext2FSInode* load_inode(inode_number)`
2. `unsigned int get_block_address(inode, block_number)`
3. `Ext2FSInode* get_file_inode_from_dir_inode(from, filename)`

Momento para preguntas