# Sistemas de Archivos

*Gisela Confalonieri*

**Departamento de Computación - FCEyN**

1 de octubre de 2024

# Estructura de la clase

# Repaso

Un **sistema de archivos** nos permite administrar y ordenar los archivos dentro de un medio de almacenamiento.

# Repaso

Un **sistema de archivos** nos permite administrar y ordenar los archivos dentro de un medio de almacenamiento.

- Partes de un sistema de archivos:

# Repaso

Un **sistema de archivos** nos permite administrar y ordenar los archivos dentro de un medio de almacenamiento.

- Partes de un sistema de archivos:
  - Archivos que almacenan datos.

# Repaso

Un **sistema de archivos** nos permite administrar y ordenar los archivos dentro de un medio de almacenamiento.

- Partes de un sistema de archivos:
  - Archivos que almacenan datos.
  - Estructura de directorios para organizar los archivos.

# Almacenamiento secundario

Las lecturas y escrituras a un medio de almacenamiento se hacen en unidades llamadas **bloques**.

# Almacenamiento secundario

Las lecturas y escrituras a un medio de almacenamiento se hacen en unidades llamadas **bloques**.

- Uno de los problemas que debe resolver un sistema de archivos es cómo **asignar** los bloques a los diferentes archivos.

# Almacenamiento secundario

Las lecturas y escrituras a un medio de almacenamiento se hacen en unidades llamadas **bloques**.

- Uno de los problemas que debe resolver un sistema de archivos es cómo **asignar** los bloques a los diferentes archivos.
- A cada archivo se le asigna una cantidad entera de bloques. ¿Qué problema ocasiona esto?

# Almacenamiento secundario

Las lecturas y escrituras a un medio de almacenamiento se hacen en unidades llamadas **bloques**.

- Uno de los problemas que debe resolver un sistema de archivos es cómo **asignar** los bloques a los diferentes archivos.
- A cada archivo se le asigna una cantidad entera de bloques. ¿Qué problema ocasiona esto?

# Almacenamiento secundario

Las lecturas y escrituras a un medio de almacenamiento se hacen en unidades llamadas **bloques**.

- Uno de los problemas que debe resolver un sistema de archivos es cómo **asignar** los bloques a los diferentes archivos.
- A cada archivo se le asigna una cantidad entera de bloques. ¿Qué problema ocasiona esto? → *fragmentación interna*.

# ¿Qué es un archivo?

- Una unidad de almacenamiento lógico.

# ¿Qué es un archivo?

- Una unidad de almacenamiento lógico.
- Un conjunto de información relacionada, con **nombre** (y otros metadatos), que se guarda en almacenamiento secundario.

# ¿Qué es un archivo?

- Una unidad de almacenamiento lógico.
- Un conjunto de información relacionada, con **nombre** (y otros metadatos), que se guarda en almacenamiento secundario.
- Desde una perspectiva de usuarie, es la porción más chica de almacenamiento (no puedo almacenar algo si no es en un archivo).

# Archivos

Atributos de un archivo:

# Archivos

Atributos de un archivo:

- Nombre
- Tipo
- Tamaño
- Permisos
- Timestamps
- ...

# Archivos

Atributos de un archivo:

- Nombre
- Tipo
- Tamaño
- Permisos
- Timestamps
- ...

Operaciones sobre archivos:

# Archivos

Atributos de un archivo:

- Nombre
- Tipo
- Tamaño
- Permisos
- Timestamps
- ...

Operaciones sobre archivos:

- Crear
- Abrir
- Escribir
- Leer
- Borrar
- ...

# Asignación contigua

- Almacenar cada archivo en un conjunto de bloques contiguos.

# Asignación contigua

- Almacenar cada archivo en un conjunto de bloques contiguos.
- ¿Qué datos necesito para acceder a un archivo?

# Asignación contigua

- Almacenar cada archivo en un conjunto de bloques contiguos.
- ¿Qué datos necesito para acceder a un archivo?

# Asignación contigua

- Almacenar cada archivo en un conjunto de bloques contiguos.
- ¿Qué datos necesito para acceder a un archivo? → la dirección del bloque inicial y el tamaño (en bloques) del archivo.

# Asignación contigua

- Almacenar cada archivo en un conjunto de bloques contiguos.
- ¿Qué datos necesito para acceder a un archivo? → la dirección del bloque inicial y el tamaño (en bloques) del archivo.
- Es fácil de implementar pero tiene limitaciones.

# Asignación contigua

- Almacenar cada archivo en un conjunto de bloques contiguos.
- ¿Qué datos necesito para acceder a un archivo? → la dirección del bloque inicial y el tamaño (en bloques) del archivo.
- Es fácil de implementar pero tiene limitaciones.
- Principal problema: encontrar espacio para un nuevo archivo. Como los archivos se van asignando y borrando, el espacio libre queda roto en pequeños pedacitos

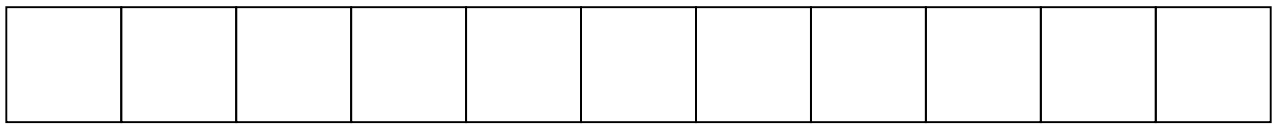
# Asignación contigua

- Almacenar cada archivo en un conjunto de bloques contiguos.
- ¿Qué datos necesito para acceder a un archivo? → la dirección del bloque inicial y el tamaño (en bloques) del archivo.
- Es fácil de implementar pero tiene limitaciones.
- Principal problema: encontrar espacio para un nuevo archivo. Como los archivos se van asignando y borrando, el espacio libre queda roto en pequeños pedacitos

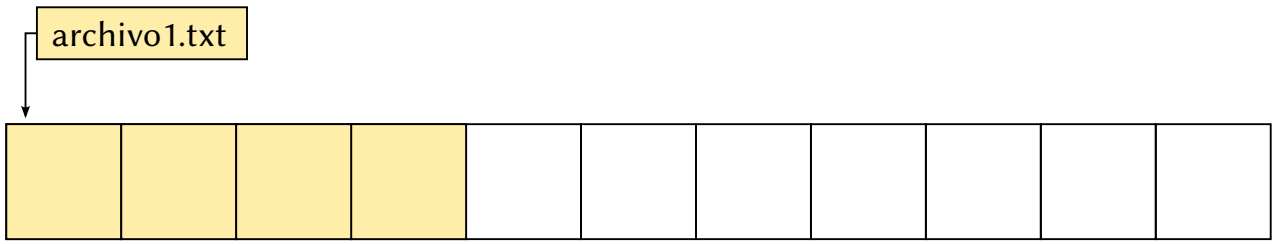
# Asignación contigua

- Almacenar cada archivo en un conjunto de bloques contiguos.
- ¿Qué datos necesito para acceder a un archivo? → la dirección del bloque inicial y el tamaño (en bloques) del archivo.
- Es fácil de implementar pero tiene limitaciones.
- Principal problema: encontrar espacio para un nuevo archivo. Como los archivos se van asignando y borrando, el espacio libre queda roto en pequeños pedacitos → *fragmentación externa*.

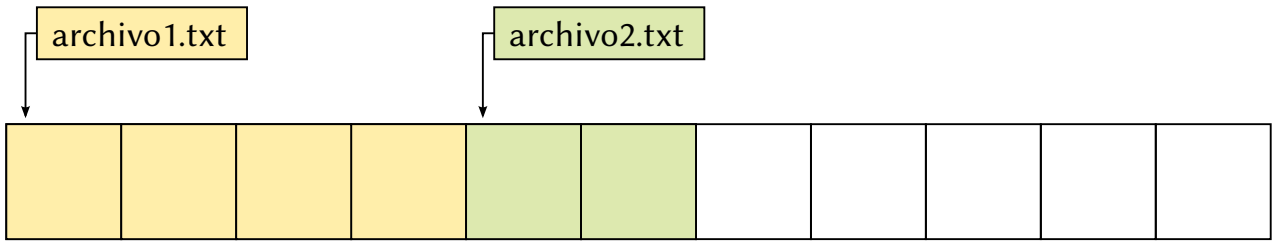
# Asignación contigua: ejemplo



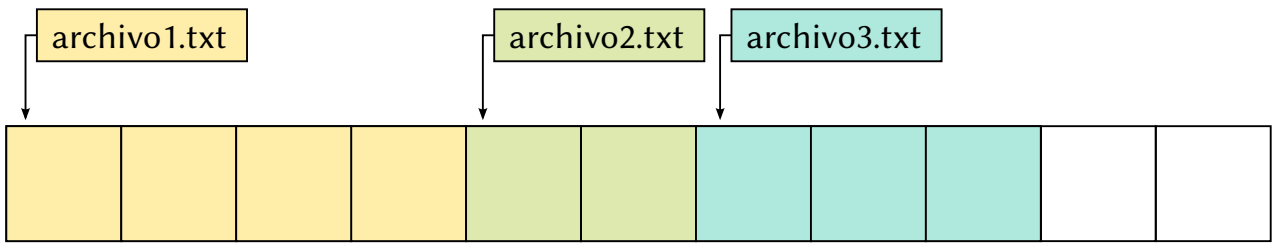
# Asignación contigua: ejemplo



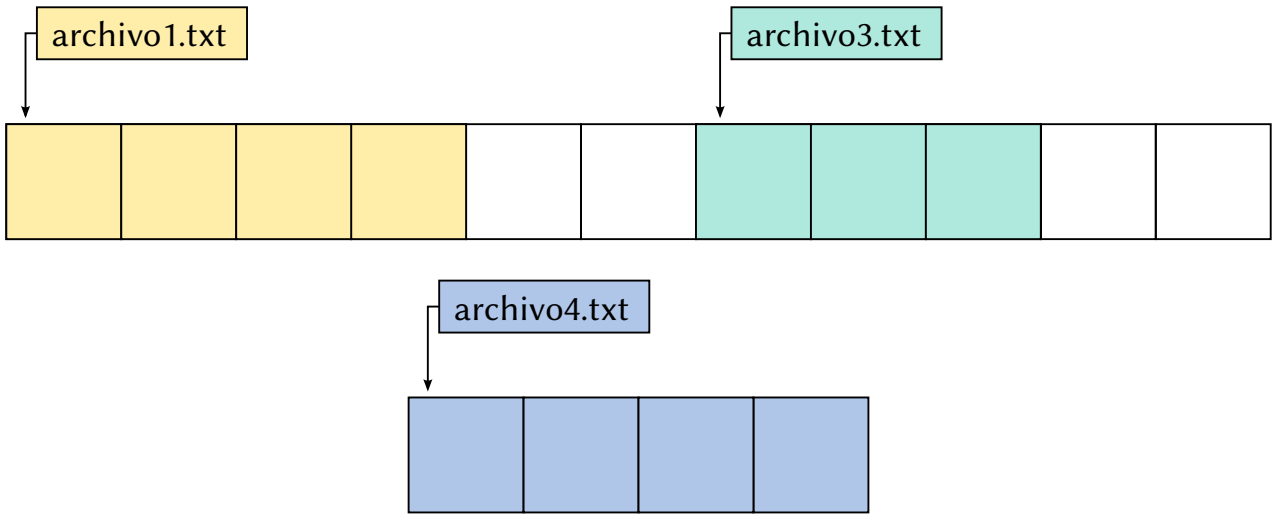
# Asignación contigua: ejemplo



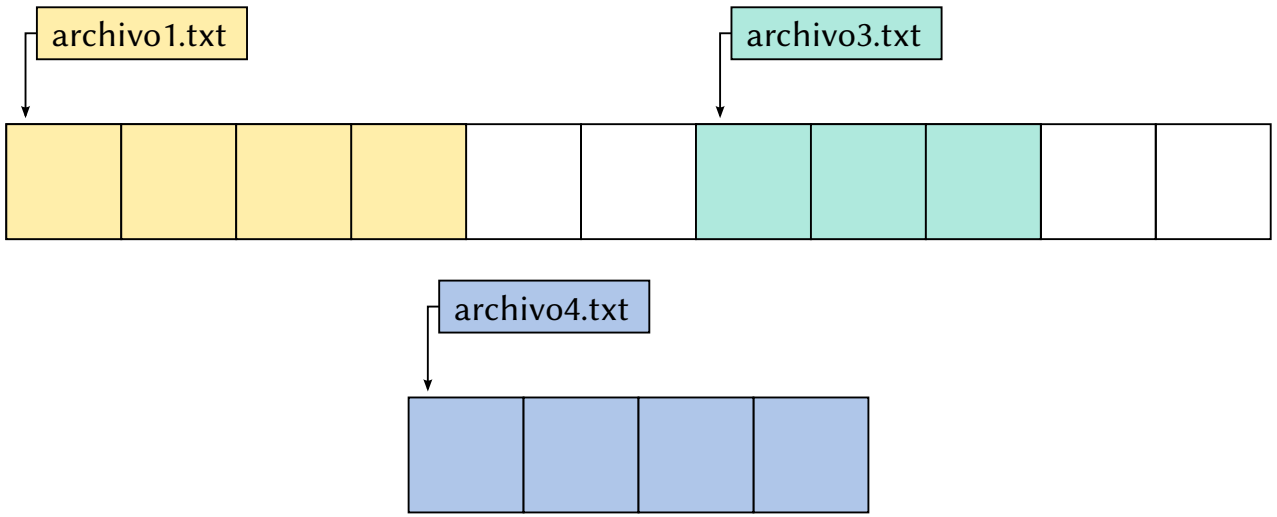
# Asignación contigua: ejemplo



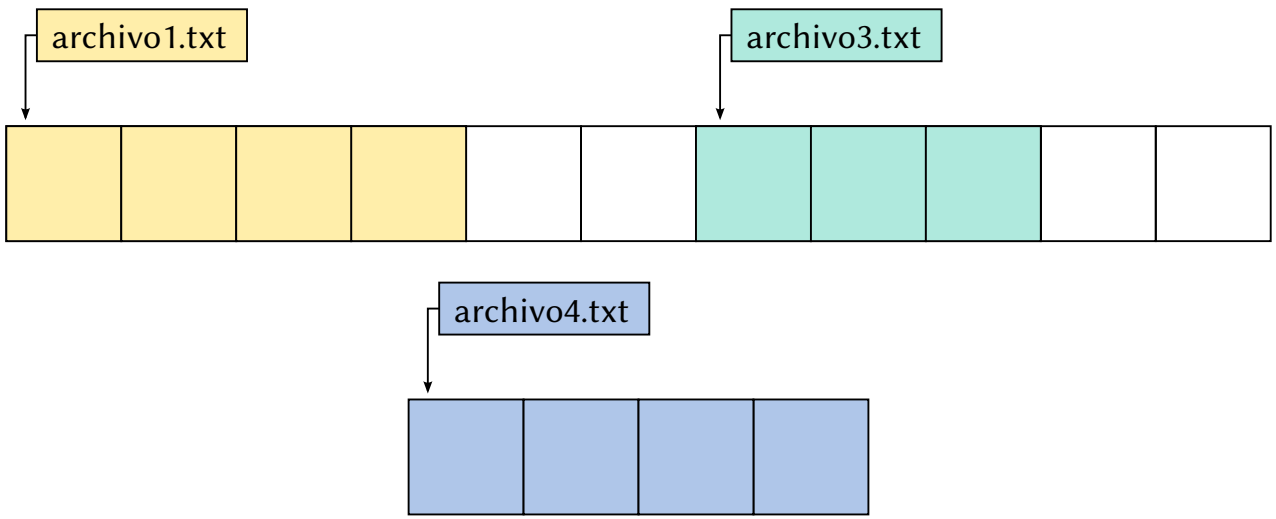
# Asignación contigua: ejemplo



# Asignación contigua: ejemplo



# Asignación contigua: ejemplo




Problema: Aunque hay espacio para `archivo4.txt`, no se lo puede almacenar debido a la *fragmentación externa*.

# Asignación contigua

- Compactar el disco es una tarea muy costosa.

# Asignación contigua

- Compactar el disco es una tarea muy costosa.
- Para reusar los "huecos", deberíamos saber qué "huecos" existen, y al crear un nuevo archivo deberíamos saber cuál será su tamaño final, y así asignar el "hueco" apropiado.

# Asignación contigua

- Compactar el disco es una tarea muy costosa.
- Para reusar los "huecos", deberíamos saber qué "huecos" existen, y al crear un nuevo archivo deberíamos saber cuál será su tamaño final, y así asignar el "hueco" apropiado.
- Existen situaciones que permiten esto: CD-ROMs y similares. En estos casos, los tamaños de archivo ya se conocen y se sabe que no cambiarán.

# Tabla de asignación de archivos (FAT)

- Permite almacenar los archivos de forma *no secuencial*, guardando para cada bloque de un archivo, una **referencia** al siguiente (al estilo lista enlazada).

# Tabla de asignación de archivos (FAT)

- Permite almacenar los archivos de forma *no secuencial*, guardando para cada bloque de un archivo, una **referencia** al siguiente (al estilo lista enlazada).
- La tabla tiene una entrada por cada bloque y se indexa por número de bloque.

# Tabla de asignación de archivos (FAT)

- Permite almacenar los archivos de forma *no secuencial*, guardando para cada bloque de un archivo, una **referencia** al siguiente (al estilo lista enlazada).
- La tabla tiene una entrada por cada bloque y se indexa por número de bloque.
- Cada entrada de la tabla contiene el número de bloque del siguiente bloque en el archivo. El último bloque de un archivo se señala con un valor especial de EOF. Si el bloque no está en uso, se señala con otro valor especial.

# Tabla de asignación de archivos (FAT)

- Permite almacenar los archivos de forma *no secuencial*, guardando para cada bloque de un archivo, una **referencia** al siguiente (al estilo lista enlazada).
- La tabla tiene una entrada por cada bloque y se indexa por número de bloque.
- Cada entrada de la tabla contiene el número de bloque del siguiente bloque en el archivo. El último bloque de un archivo se señala con un valor especial de EOF. Si el bloque no está en uso, se señala con otro valor especial.
- ¿Qué datos necesito para acceder a un archivo?

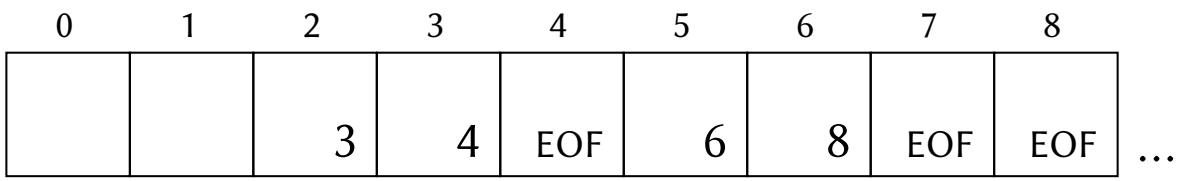
# Tabla de asignación de archivos (FAT)

- Permite almacenar los archivos de forma *no secuencial*, guardando para cada bloque de un archivo, una **referencia** al siguiente (al estilo lista enlazada).
- La tabla tiene una entrada por cada bloque y se indexa por número de bloque.
- Cada entrada de la tabla contiene el número de bloque del siguiente bloque en el archivo. El último bloque de un archivo se señala con un valor especial de EOF. Si el bloque no está en uso, se señala con otro valor especial.
- ¿Qué datos necesito para acceder a un archivo?

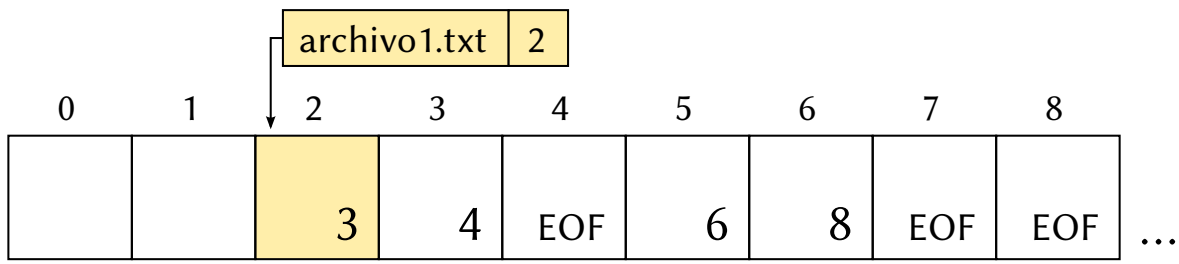
# Tabla de asignación de archivos (FAT)

- Permite almacenar los archivos de forma *no secuencial*, guardando para cada bloque de un archivo, una **referencia** al siguiente (al estilo lista enlazada).
- La tabla tiene una entrada por cada bloque y se indexa por número de bloque.
- Cada entrada de la tabla contiene el número de bloque del siguiente bloque en el archivo. El último bloque de un archivo se señala con un valor especial de EOF. Si el bloque no está en uso, se señala con otro valor especial.
- ¿Qué datos necesito para acceder a un archivo? $\rightarrow$ El número de bloque del primer bloque del archivo.

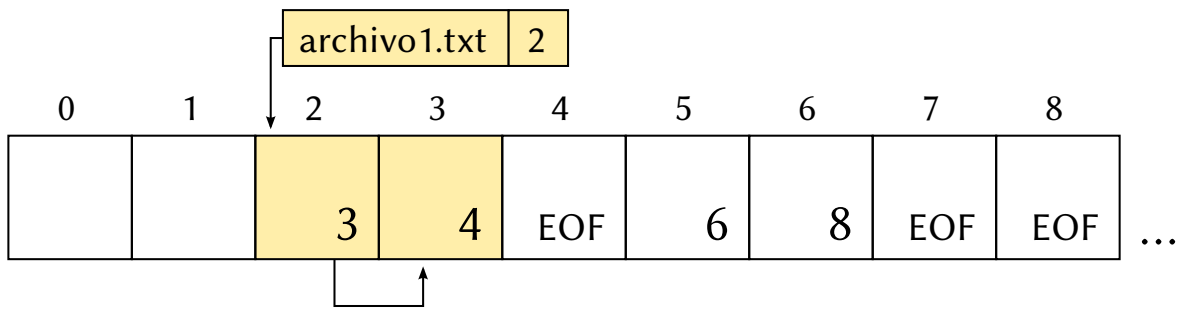
# Tabla de asignación de archivos (FAT): ejemplo



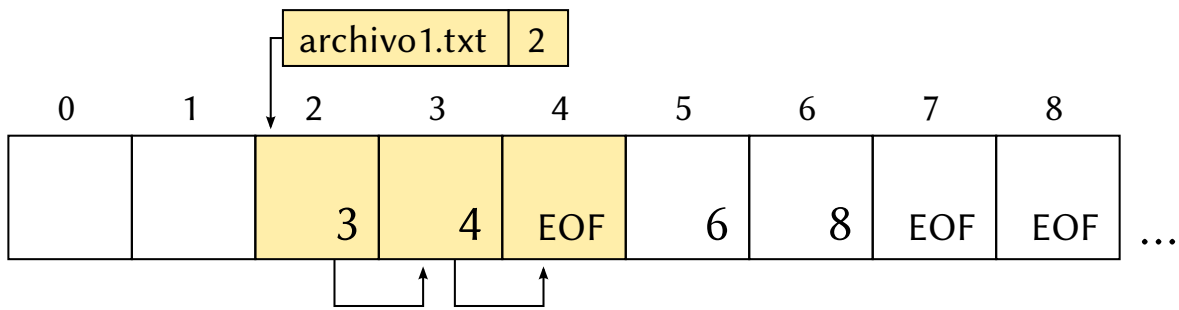
# Tabla de asignación de archivos (FAT): ejemplo



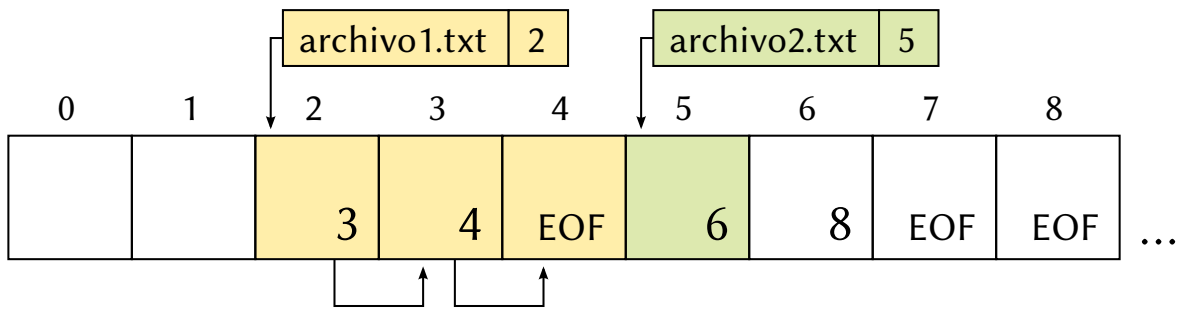
# Tabla de asignación de archivos (FAT): ejemplo



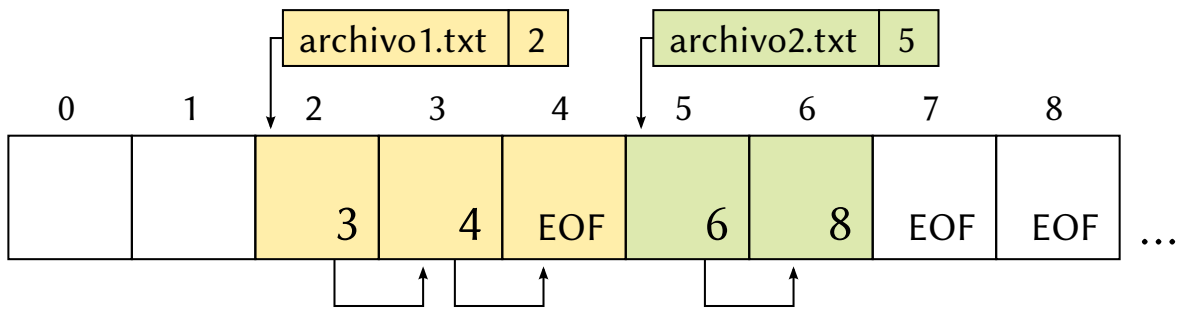
# Tabla de asignación de archivos (FAT): ejemplo



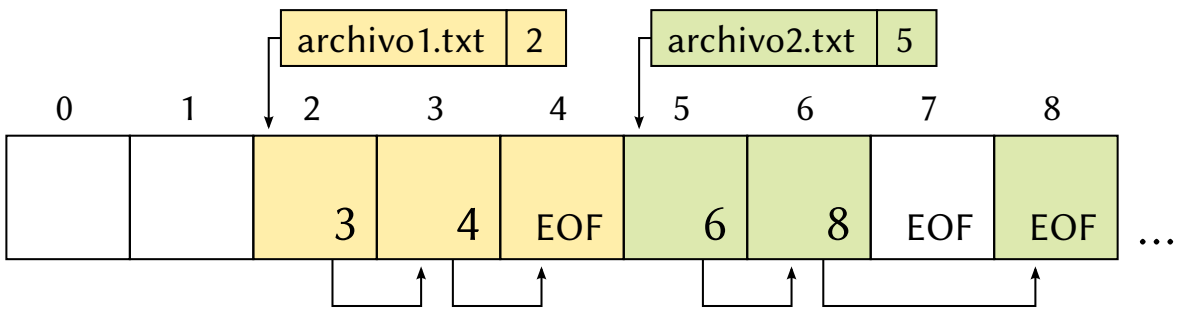
# Tabla de asignación de archivos (FAT): ejemplo



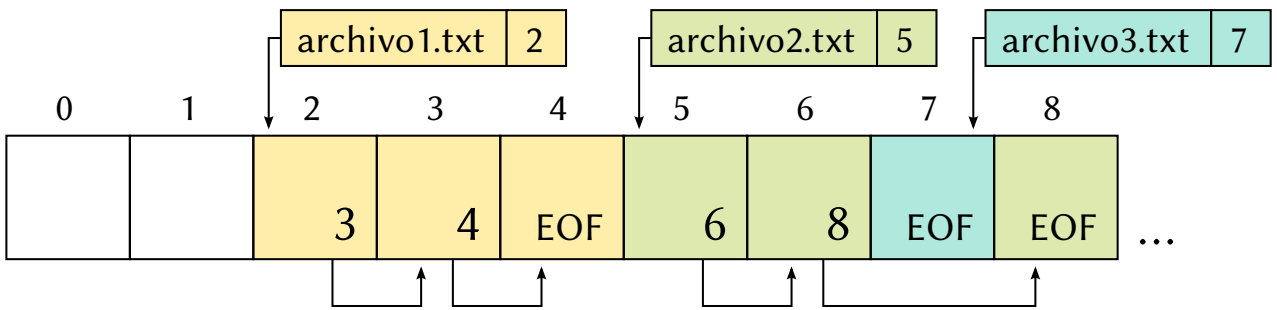
# Tabla de asignación de archivos (FAT): ejemplo



# Tabla de asignación de archivos (FAT): ejemplo



# Tabla de asignación de archivos (FAT): ejemplo



# Tabla de asignación de archivos (FAT)

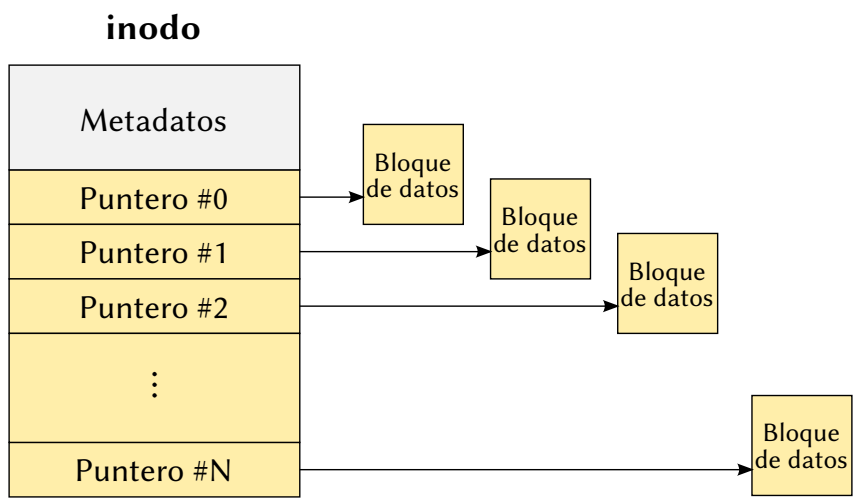
- La principal desventaja de FAT es que la tabla completa debe estar en memoria.

# Tabla de asignación de archivos (FAT)

- La principal desventaja de FAT es que la tabla completa debe estar en memoria.
- Considerando un disco de 1 TB y bloques de 1-KB, la tabla necesitaría $10^9$ entradas. Si cada entrada es de 4 bytes, la tabla ocuparía 3 GB de memoria principal.

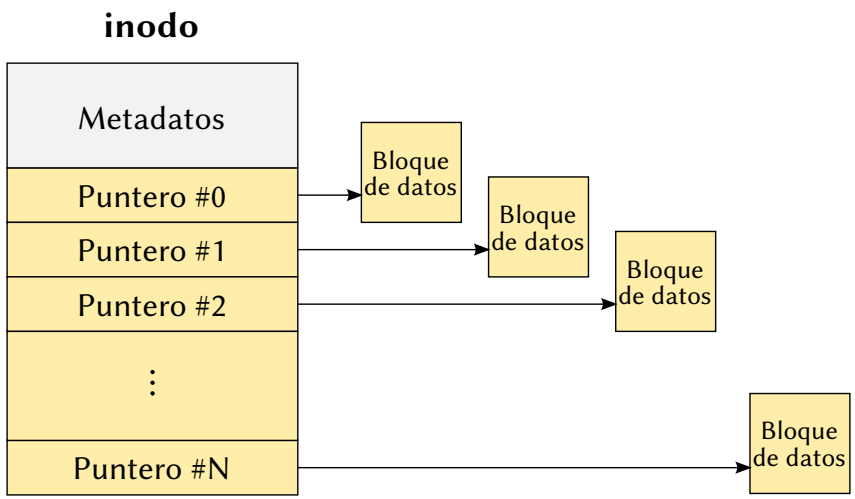
# Inodos

- En un sistema con **inodos**, cada archivo tiene su propio **índice de bloques**, con **punteros** a los bloques de datos que conforman del archivo.



# Inodos

- En un sistema con **inodos**, cada archivo tiene su propio **índice de bloques**, con **punteros** a los bloques de datos que conforman del archivo.
- El inodo debe cargarse en memoria sólo cuando el archivo correspondiente es abierto.



# Inodos

- La i-ésima entrada en el índice apunta al iésimo bloque del archivo.

# Inodos

- La i-ésima entrada en el índice apunta al iésimo bloque del archivo.
- Ojo: mantener este índice requiere espacio. ¿Qué tan grande debe ser la estructura de índices?

# Inodos

- La i-ésima entrada en el índice apunta al iésimo bloque del archivo.
- Ojo: mantener este índice requiere espacio. ¿Qué tan grande debe ser la estructura de índices?
  - Queremos que sea lo más chico posible.

# Inodos

- La i-ésima entrada en el índice apunta al iésimo bloque del archivo.
- Ojo: mantener este índice requiere espacio. ¿Qué tan grande debe ser la estructura de índices?
  - Queremos que sea lo más chico posible.
  - Pero si es muy pequeño, no podrá almacenar la cantidad de punteros suficientes para un archivo grande.

# Inodos: Punteros con indirección

- Es deseable que los inodos tengan un tamaño fijo.

# Inodos: Punteros con indirección

- Es deseable que los inodos tengan un tamaño fijo.
- Además, los primeros bloques de un archivo suelen ser accedidos con más frecuencia.

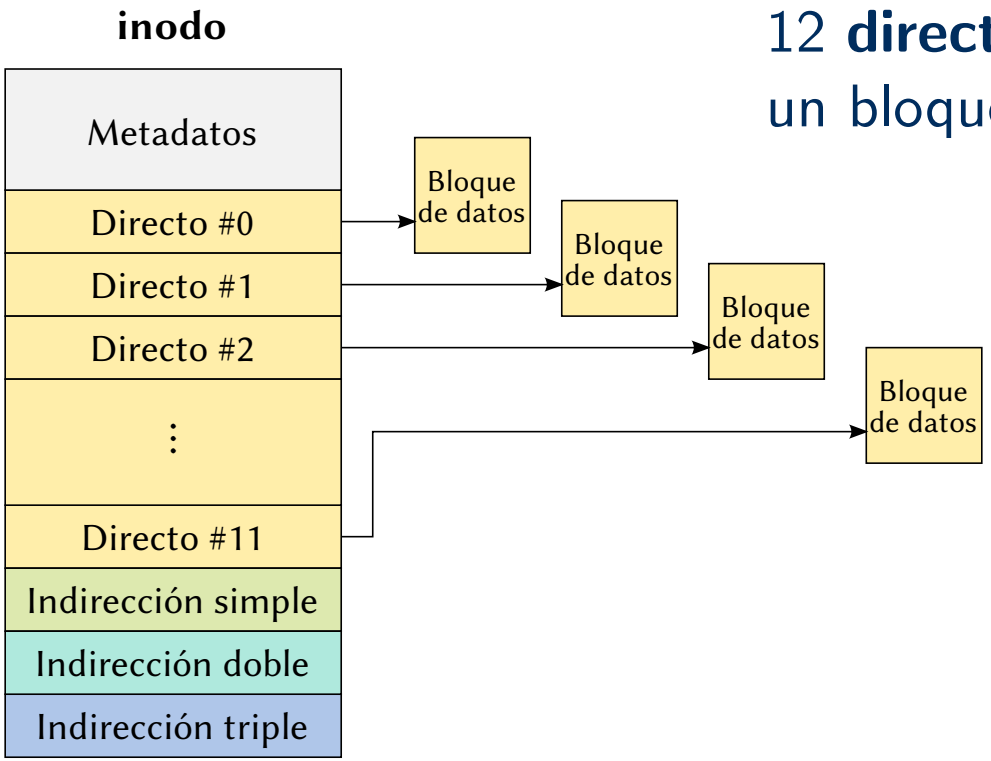
# Inodos: Punteros con indirección

- Es deseable que los inodos tengan un tamaño fijo.
- Además, los primeros bloques de un archivo suelen ser accedidos con más frecuencia.
- Por eso, se utilizan **punteros con indirección**.

# Inodos: Punteros con indirección

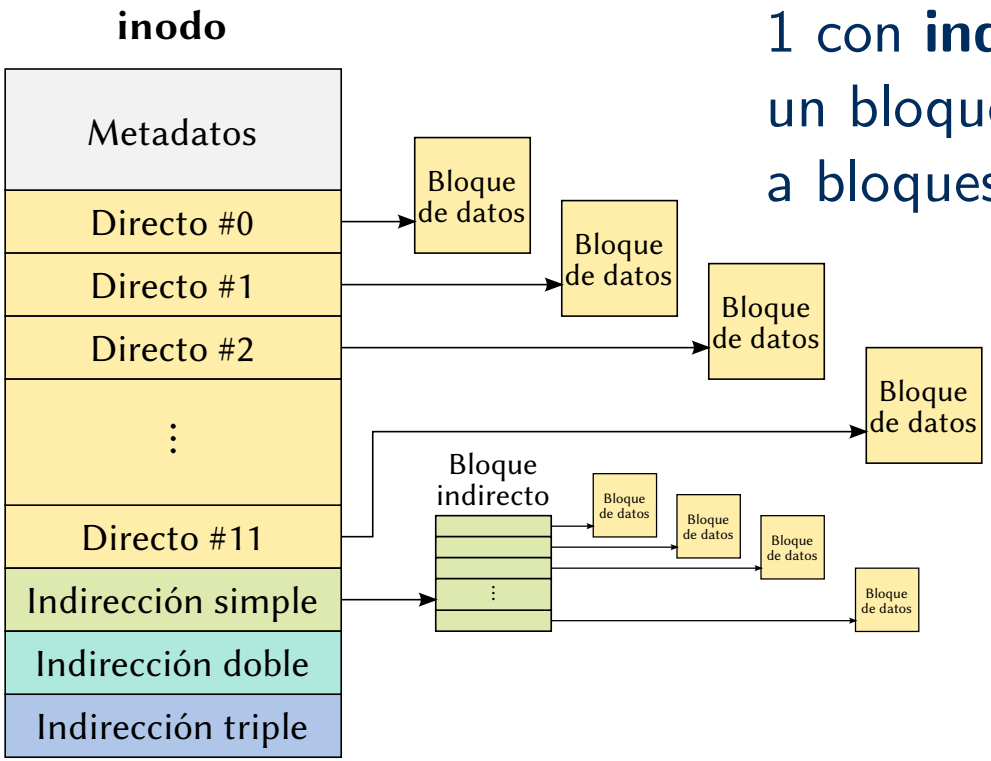

- Es deseable que los inodos tengan un tamaño fijo.
- Además, los primeros bloques de un archivo suelen ser accedidos con más frecuencia.
- Por eso, se utilizan **punteros con indirección**.
- Por ejemplo, en ext2, todos los inodos contienen 15 punteros a bloques, de cuatro tipos distintos.

# Inodos: Punteros con indirección



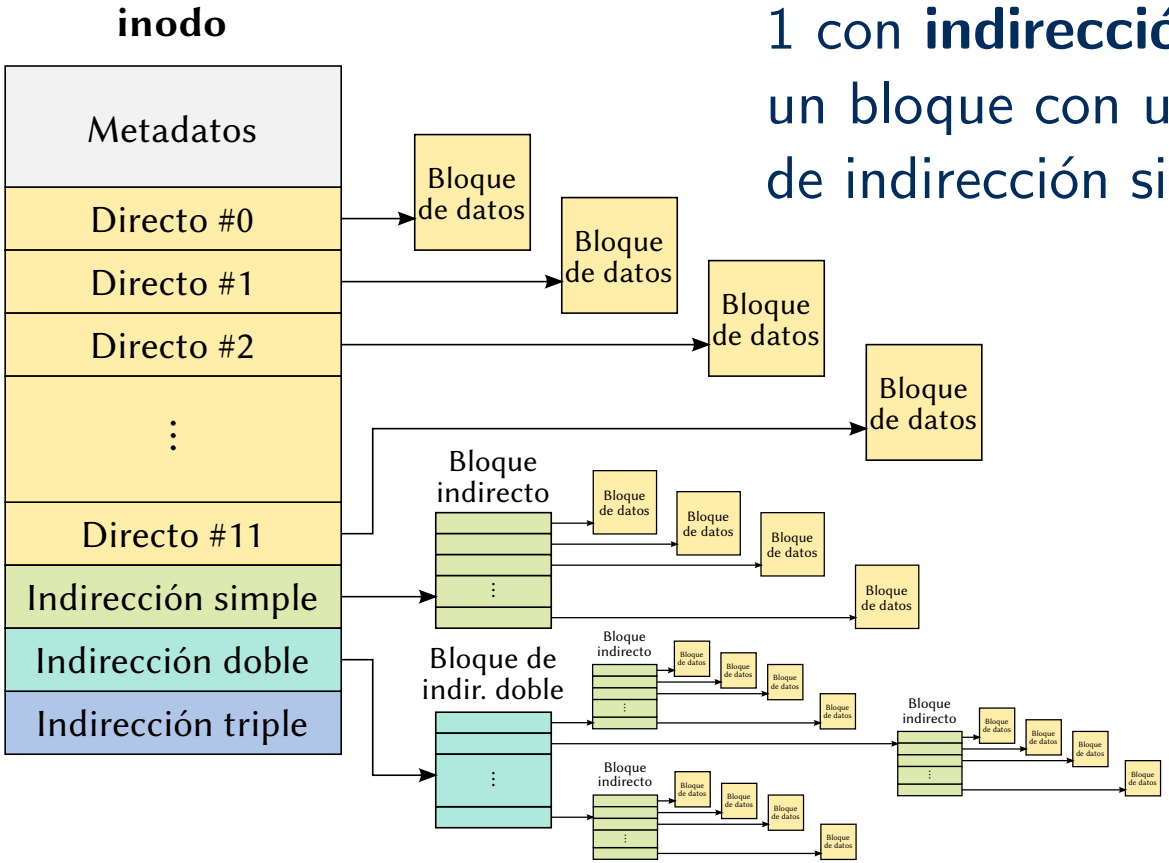

12 **directos**: apuntan directamente a un bloque de datos.

# Inodos: Punteros con indirección



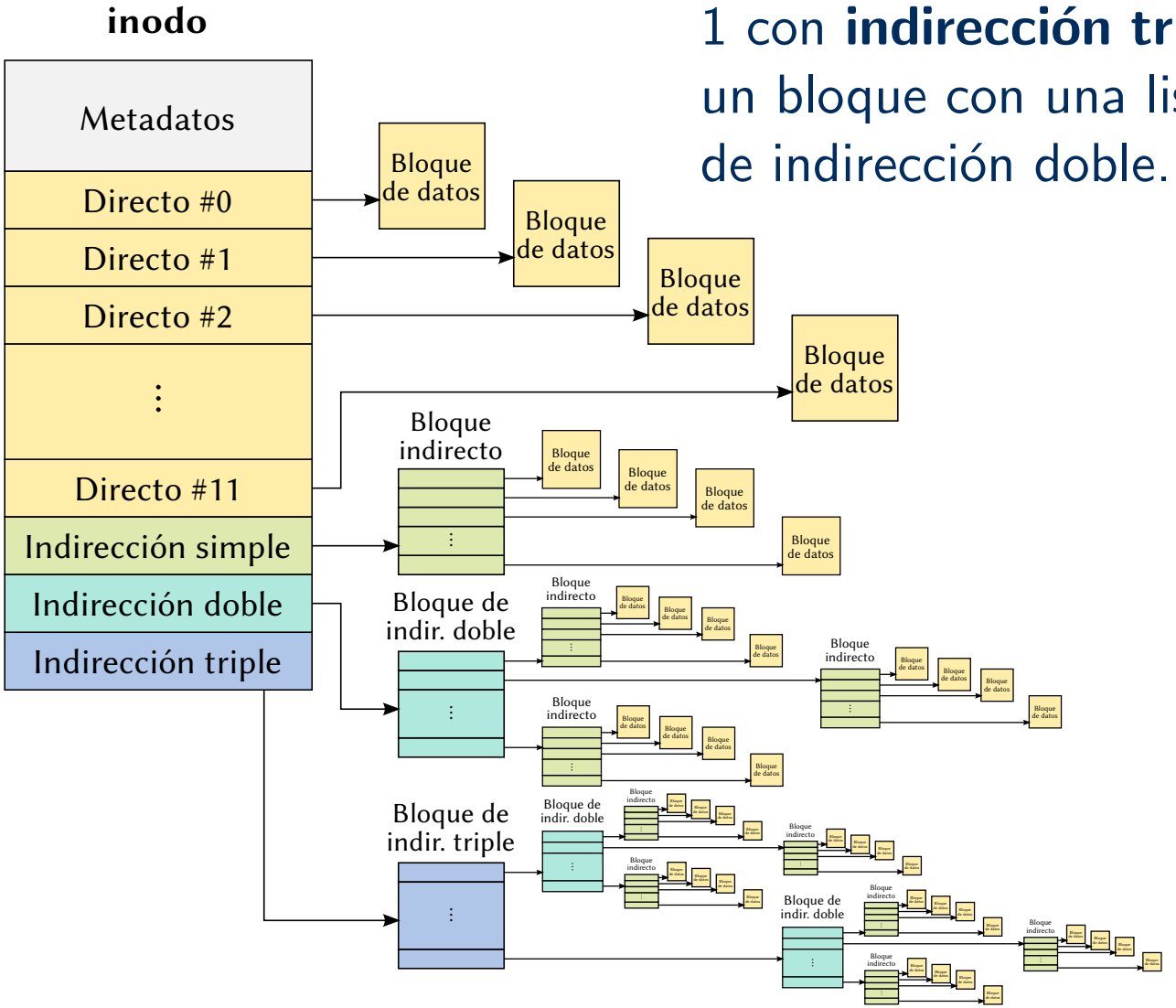

1 con **indirección simple**: apunta a un bloque con una lista de punteros a bloques de datos.

# Inodos: Punteros con indirección




1 con **indirección doble**: apunta a un bloque con una lista de punteros de indirección simple.

# Inodos: Punteros con indirección




1 con **indirección triple**: apunta a un bloque con una lista de punteros de indirección doble.

# Directorios

- Antes de poder leer un archivo (syscall `read()`), hay que abrirlo.

# Directorios

- Antes de poder leer un archivo (syscall `read()`), hay que abrirlo.
- Para abrir un archivo (syscall `open()`), se debe especificar la ruta (absoluta o relativa).

# Directorios

- Antes de poder leer un archivo (syscall `read()`), hay que abrirlo.
- Para abrir un archivo (syscall `open()`), se debe especificar la ruta (absoluta o relativa).
- El SO "recorre" el *path* hasta encontrar la **entrada de directorio** correspondiente al archivo.

# Directorios

- Antes de poder leer un archivo (syscall `read()`), hay que abrirlo.
- Para abrir un archivo (syscall `open()`), se debe especificar la ruta (absoluta o relativa).
- El SO "recorre" el *path* hasta encontrar la **entrada de directorio** correspondiente al archivo.
- La entrada de directorio provee la información necesaria para encontrar los bloques de disco donde está almacenado el archivo:

# Directorios

- Antes de poder leer un archivo (syscall `read()`), hay que abrirlo.
- Para abrir un archivo (syscall `open()`), se debe especificar la ruta (absoluta o relativa).
- El SO "recorre" el *path* hasta encontrar la **entrada de directorio** correspondiente al archivo.
- La entrada de directorio provee la información necesaria para encontrar los bloques de disco donde está almacenado el archivo:
  - En FAT:

# Directorios

- Antes de poder leer un archivo (syscall `read()`), hay que abrirlo.
- Para abrir un archivo (syscall `open()`), se debe especificar la ruta (absoluta o relativa).
- El SO "recorre" el *path* hasta encontrar la **entrada de directorio** correspondiente al archivo.
- La entrada de directorio provee la información necesaria para encontrar los bloques de disco donde está almacenado el archivo:
  - En FAT:

# Directorios

- Antes de poder leer un archivo (syscall `read()`), hay que abrirlo.
- Para abrir un archivo (syscall `open()`), se debe especificar la ruta (absoluta o relativa).
- El SO "recorre" el *path* hasta encontrar la **entrada de directorio** correspondiente al archivo.
- La entrada de directorio provee la información necesaria para encontrar los bloques de disco donde está almacenado el archivo:
  - En FAT: el número del primer bloque.

# Directorios

- Antes de poder leer un archivo (syscall `read()`), hay que abrirlo.
- Para abrir un archivo (syscall `open()`), se debe especificar la ruta (absoluta o relativa).
- El SO "recorre" el *path* hasta encontrar la **entrada de directorio** correspondiente al archivo.
- La entrada de directorio provee la información necesaria para encontrar los bloques de disco donde está almacenado el archivo:
  - En FAT: el número del primer bloque.
  - En inodos:

# Directorios

- Antes de poder leer un archivo (syscall `read()`), hay que abrirlo.
- Para abrir un archivo (syscall `open()`), se debe especificar la ruta (absoluta o relativa).
- El SO "recorre" el *path* hasta encontrar la **entrada de directorio** correspondiente al archivo.
- La entrada de directorio provee la información necesaria para encontrar los bloques de disco donde está almacenado el archivo:
  - En FAT: el número del primer bloque.
  - En inodos:

# Directorios

- Antes de poder leer un archivo (syscall `read()`), hay que abrirlo.
- Para abrir un archivo (syscall `open()`), se debe especificar la ruta (absoluta o relativa).
- El SO "recorre" el *path* hasta encontrar la **entrada de directorio** correspondiente al archivo.
- La entrada de directorio provee la información necesaria para encontrar los bloques de disco donde está almacenado el archivo:
  - En FAT: el número del primer bloque.
  - En inodos: el número de inodo.

# Directorios

Los **directorios** también son archivos. Sus bloques de datos representan una tabla con una entrada por cada archivo que contienen, indicando su nombre y su posición física en el disco.

# Directorios

Los **directorios** también son archivos. Sus bloques de datos representan una tabla con una entrada por cada archivo que contienen, indicando su nombre y su posición física en el disco.

- Un directorio puede contener subdirectorios. Así, podemos organizar los archivos en una *estructura jerárquica*, mediante un árbol de directorios.

# Directorios en FAT

- Consiste en una lista de entradas de tamaño fijo.

# Directorios en FAT

- Consiste en una lista de entradas de tamaño fijo.
- Cada entrada de directorio indica el índice del **primer bloque** de cada archivo.

# Directorios en FAT

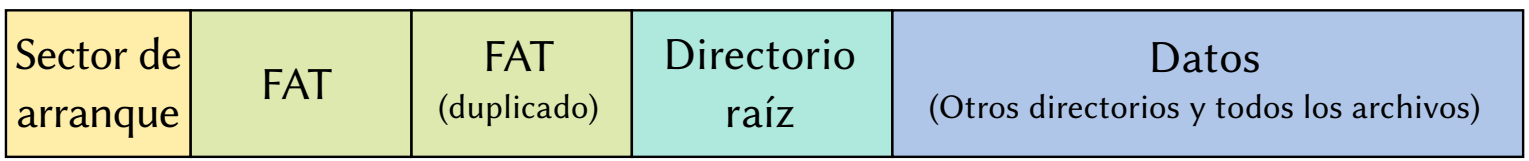
- Consiste en una lista de entradas de tamaño fijo.
- Cada entrada de directorio indica el índice del **primer bloque** de cada archivo.
- La entrada de directorio almacena también todos los **metadatos**: nombre, tamaño, fecha de último acceso, etc.

# Directorios en FAT

- Consiste en una lista de entradas de tamaño fijo.
- Cada entrada de directorio indica el índice del **primer bloque** de cada archivo.
- La entrada de directorio almacena también todos los **metadatos**: nombre, tamaño, fecha de último acceso, etc.
- El bloque del directorio **root** es distinguido. De esta forma, podemos encontrar cualquier archivo a partir de su **ruta**.

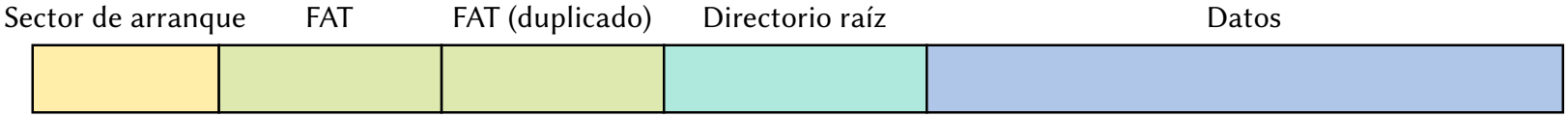
# Directorios en FAT

- Consiste en una lista de entradas de tamaño fijo.
- Cada entrada de directorio indica el índice del **primer bloque** de cada archivo.
- La entrada de directorio almacena también todos los **metadatos**: nombre, tamaño, fecha de último acceso, etc.
- El bloque del directorio **root** es distinguido. De esta forma, podemos encontrar cualquier archivo a partir de su **ruta**.



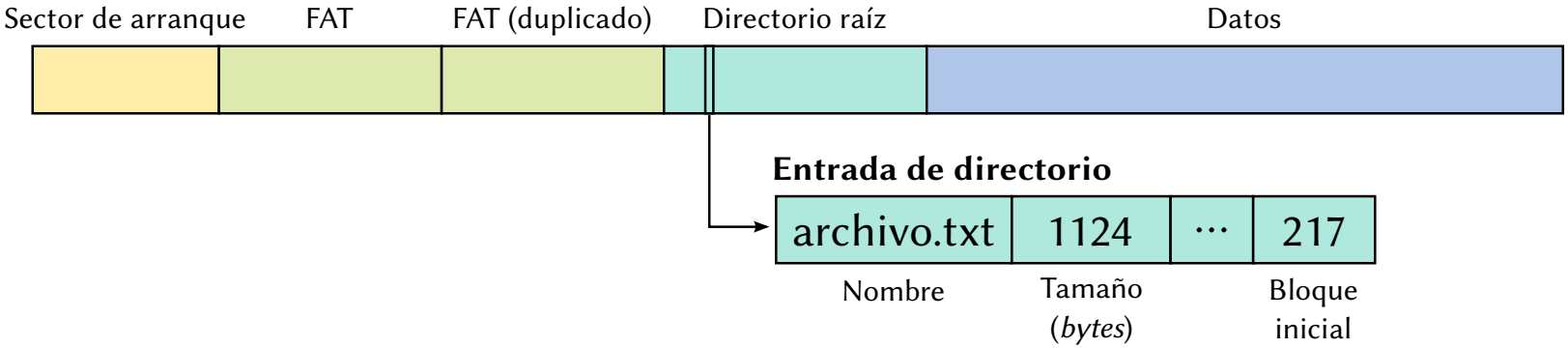

Estructura de un sistema de archivos FAT32.

# Directorios en FAT: obteniendo un archivo(*)



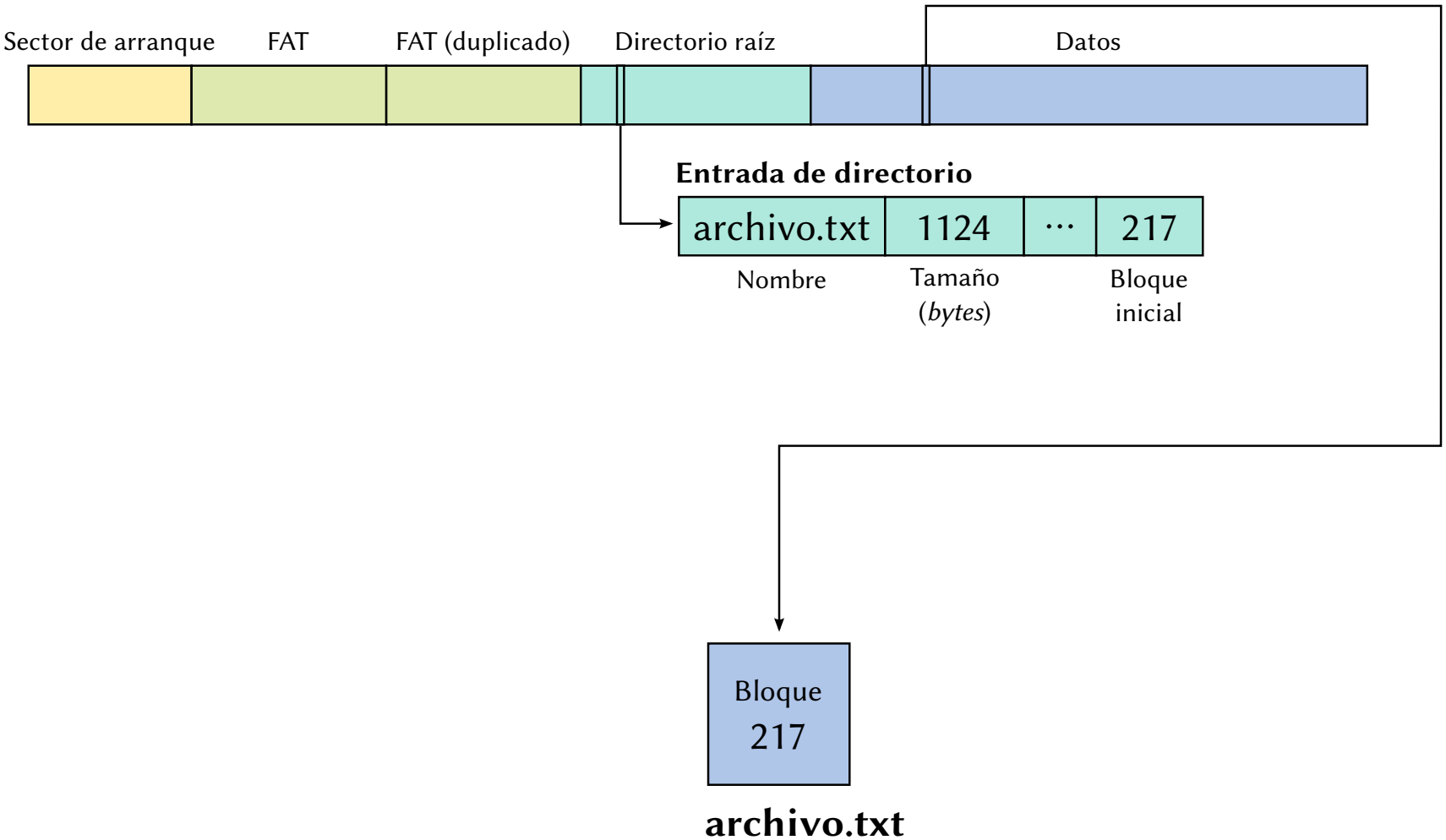

(*)Suponiendo bloques de 512 *bytes*.

# Directorios en FAT: obteniendo un archivo(*)



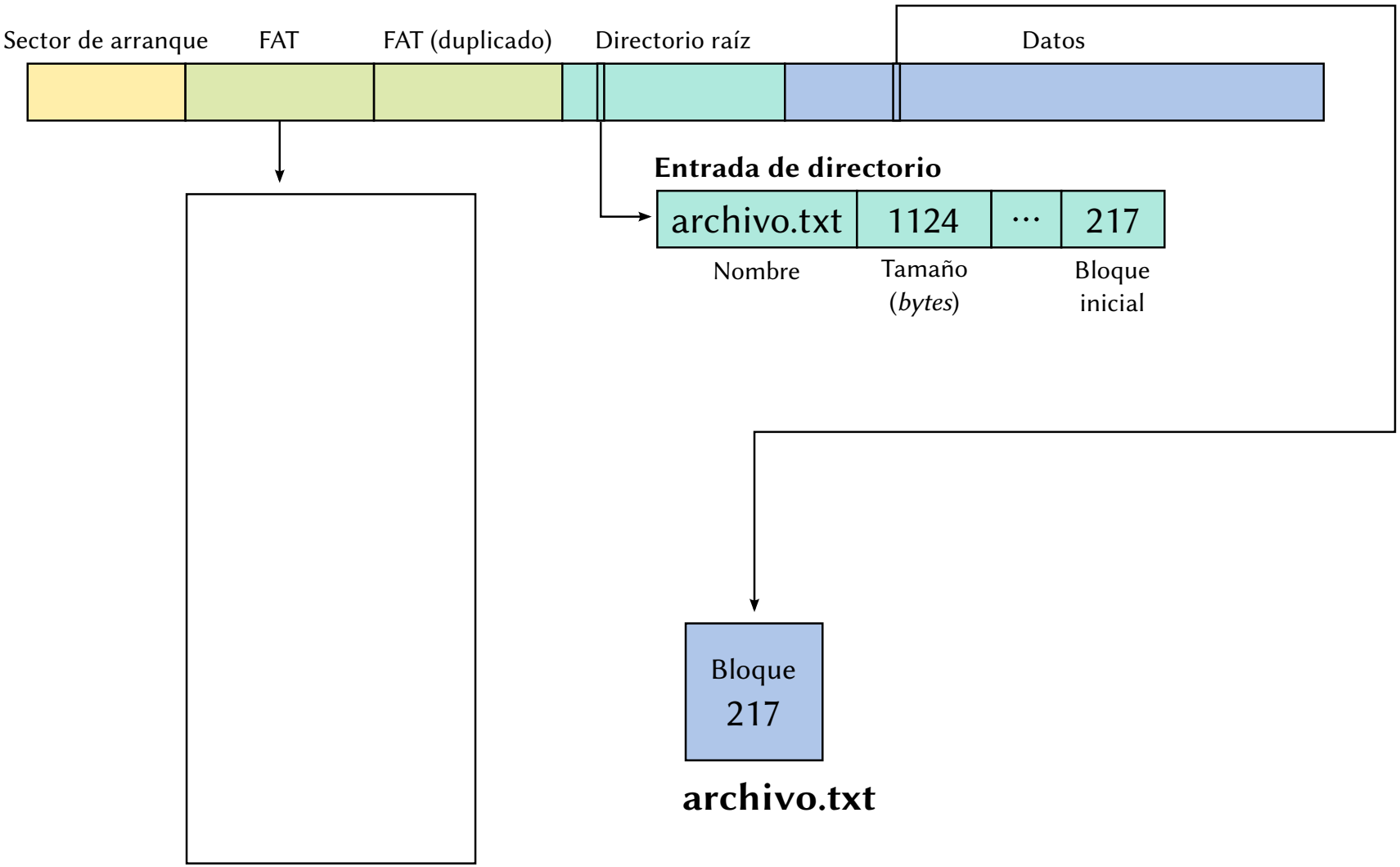

(*)Suponiendo bloques de 512 *bytes*.

# Directorios en FAT: obteniendo un archivo(*)



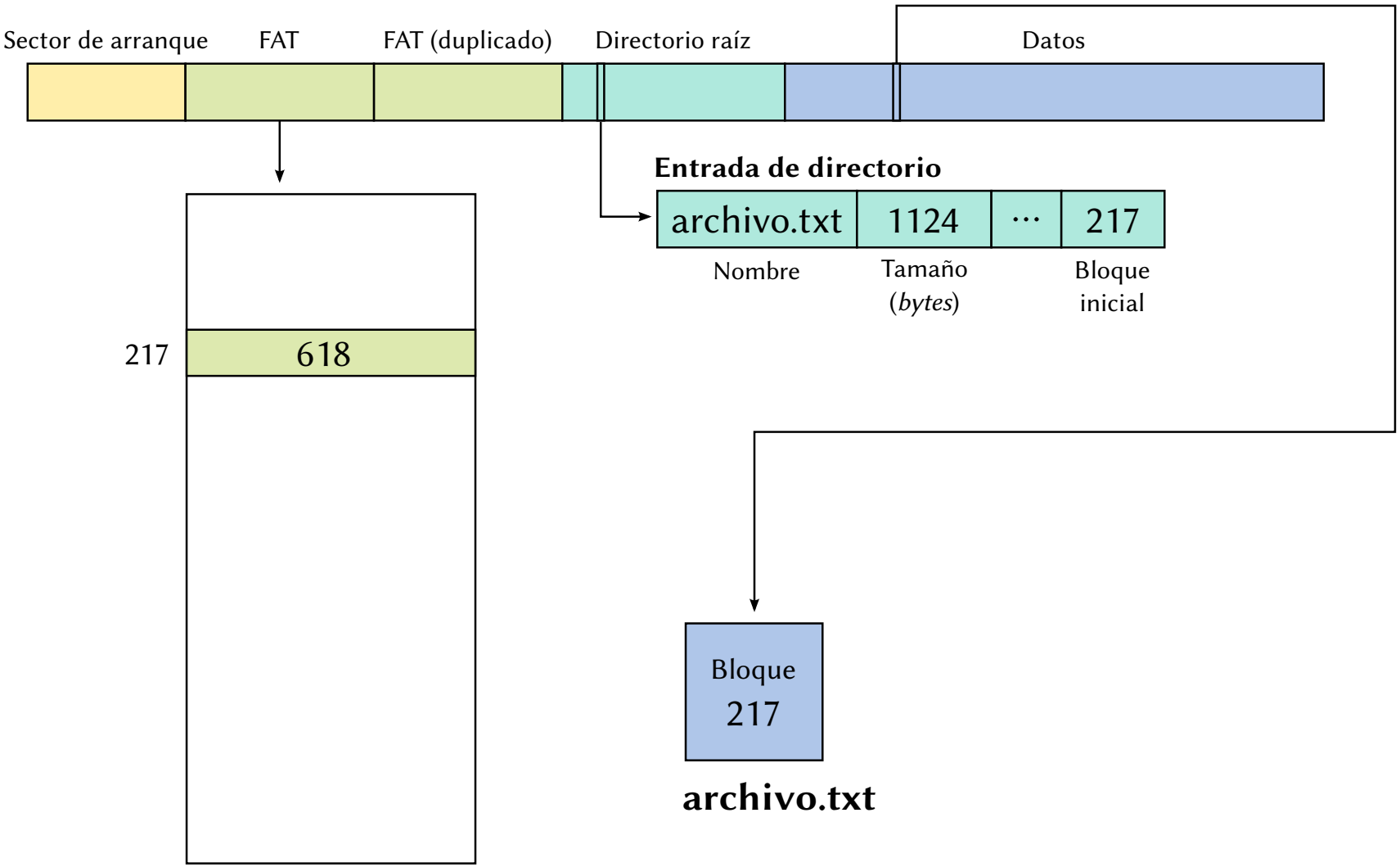

(*)Suponiendo bloques de 512 *bytes*.

# Directorios en FAT: obteniendo un archivo(*)



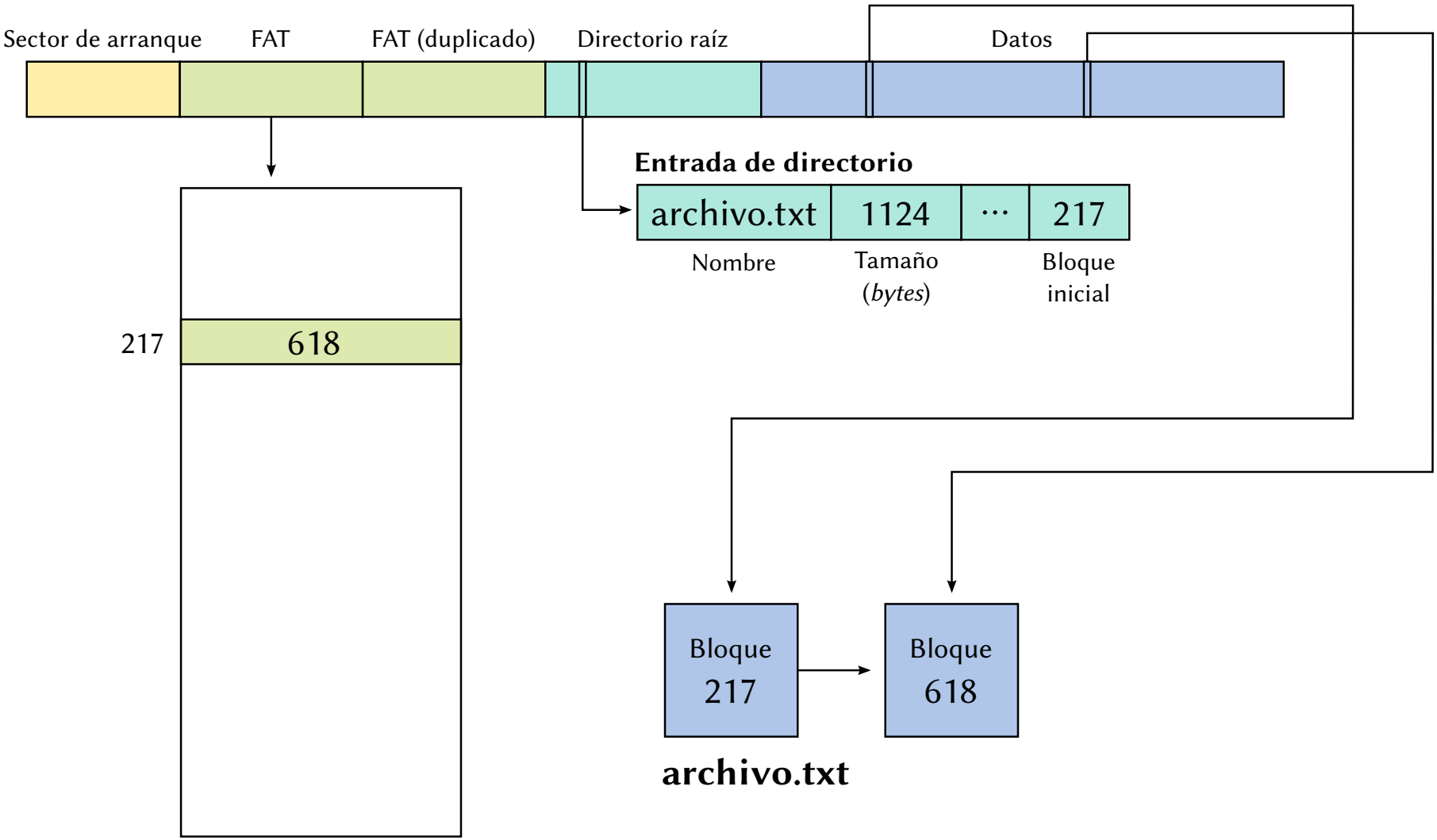

(*)Suponiendo bloques de 512 *bytes*.

# Directorios en FAT: obteniendo un archivo(*)



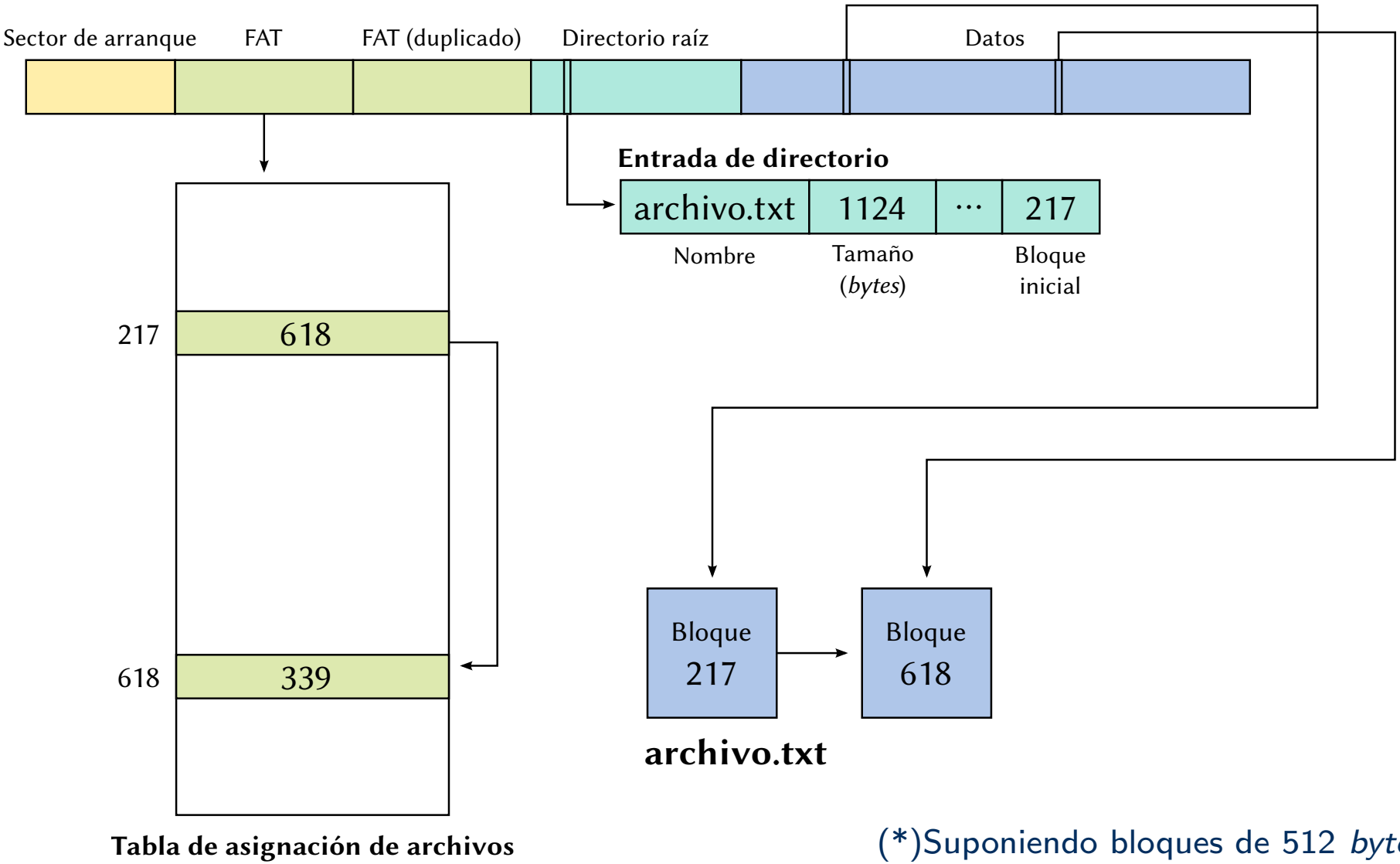

(*)Suponiendo bloques de 512 *bytes*.

# Directorios en FAT: obteniendo un archivo(*)



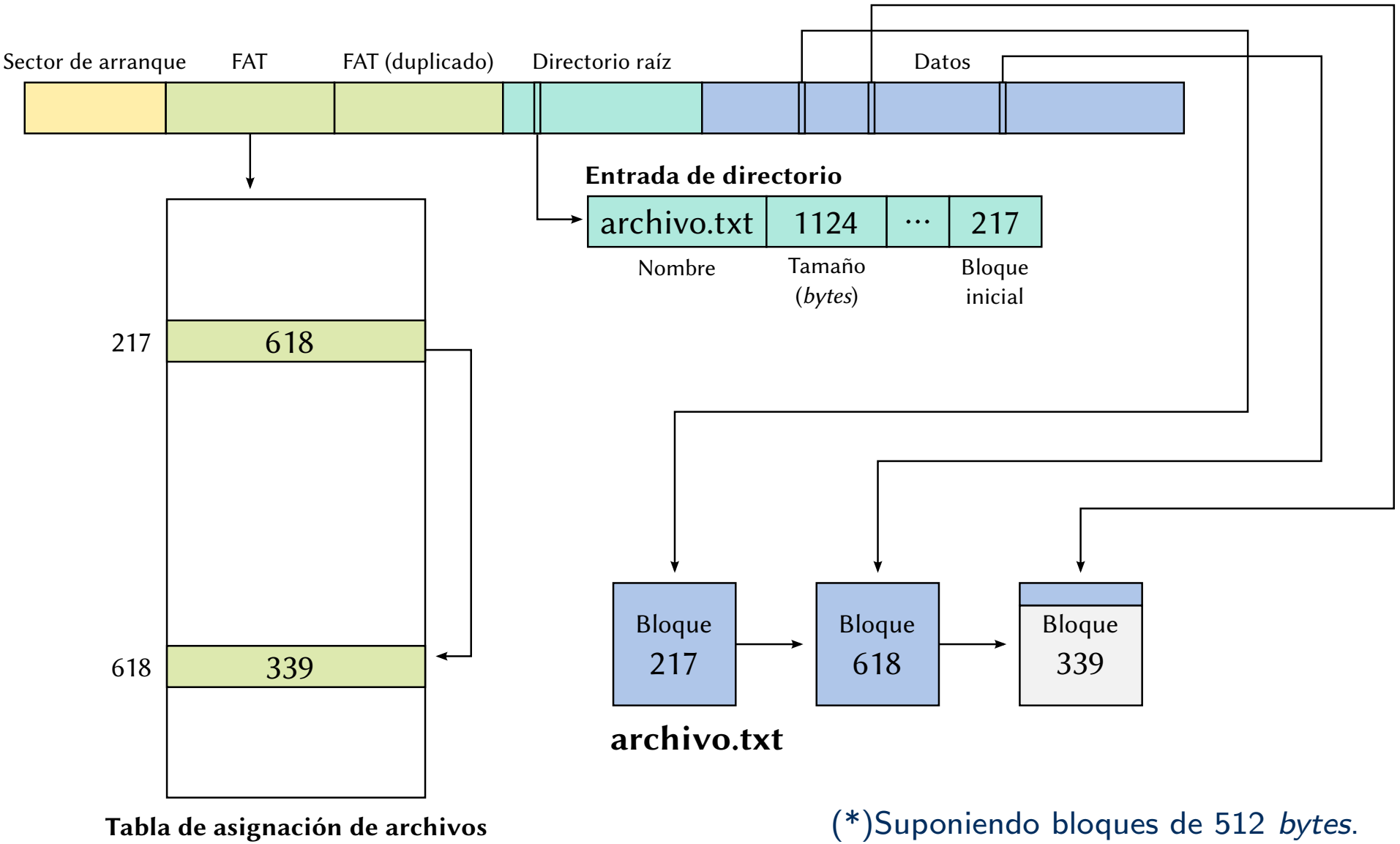

(*)Suponiendo bloques de 512 *bytes*.

# Directorios en FAT: obteniendo un archivo(*)



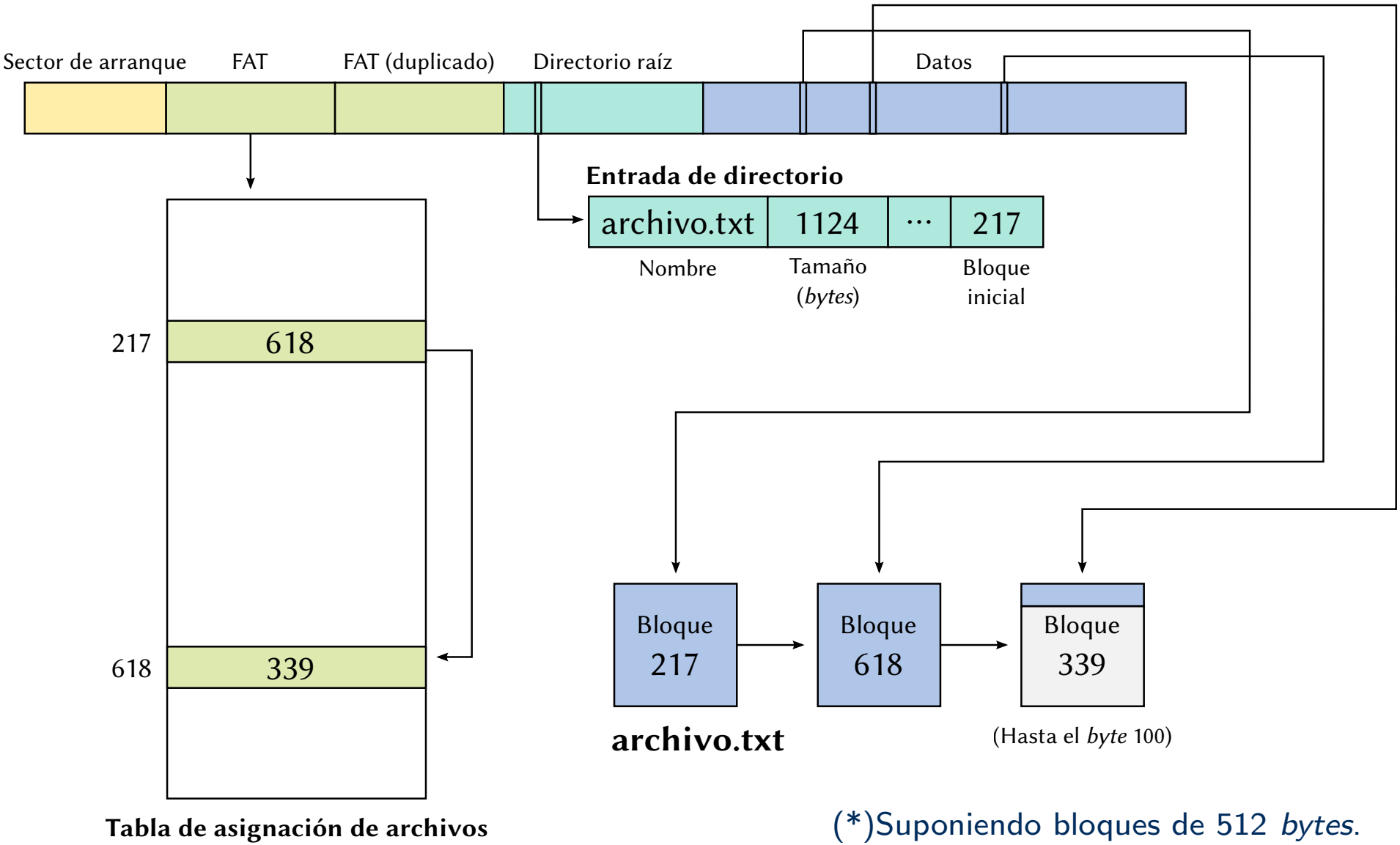

(*)Suponiendo bloques de 512 *bytes*.

# Directorios en FAT: obteniendo un archivo(*)



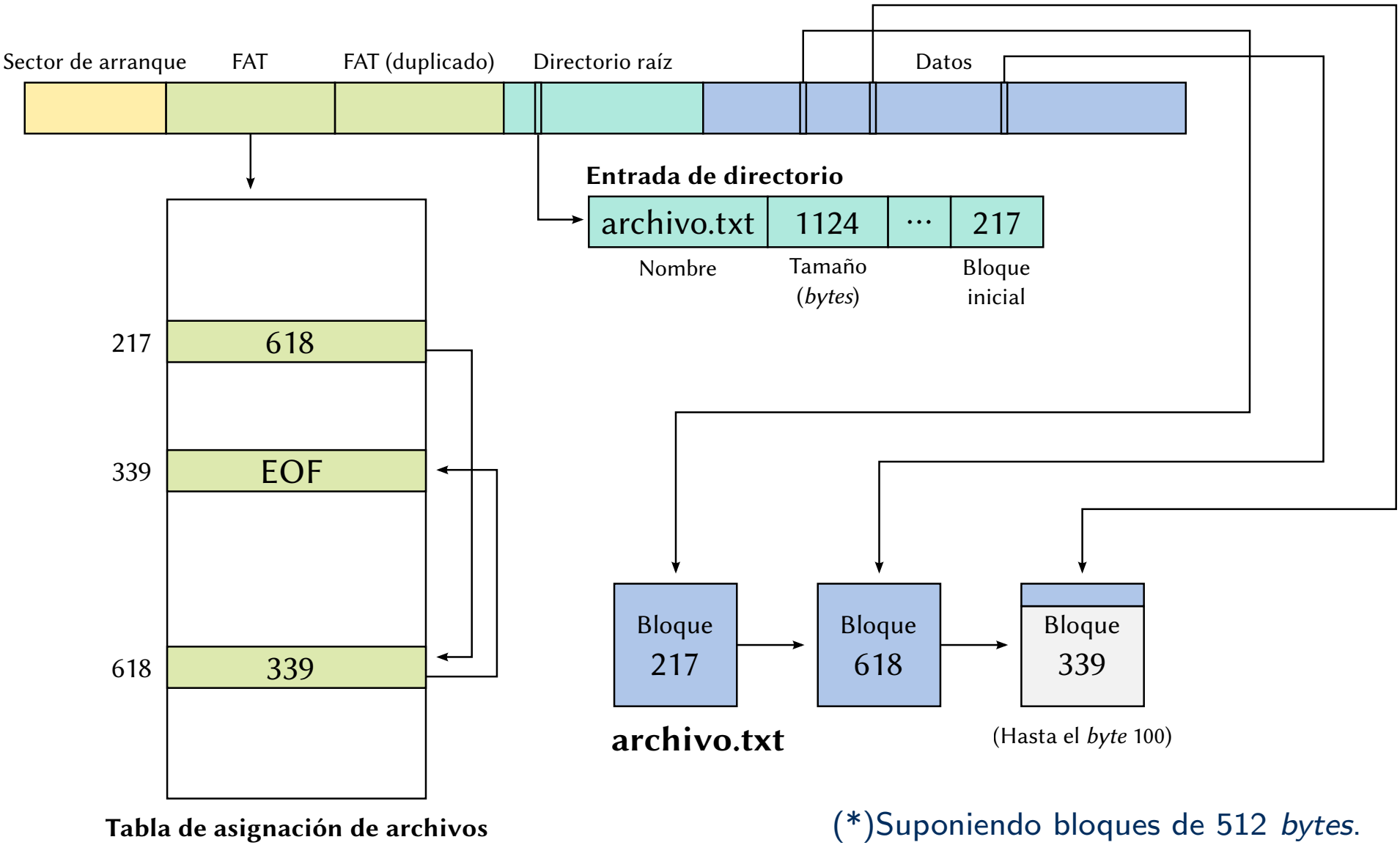

(*)Suponiendo bloques de 512 *bytes*.

# Directorios en FAT: obteniendo un archivo(*)




(*)Suponiendo bloques de 512 *bytes*.

# Directorios en FAT: obteniendo un archivo(*)




(*)Suponiendo bloques de 512 *bytes*.

# Ejercicio 1

Contamos un sistema de archivos formateado con FAT12 con bloques de 2KiB (con clusters de 1 bloque). Indicar a cuántos bloques de disco hay que acceder para leer los bytes [3.000-3.083] del archivo `/home/el/parcial.avi`.

Se puede asumir que:

- Los archivos y directorios a buscar se encuentran siempre en el primer bloque de *directory entries* correspondientes al directorio padre.
- Si un bloque se lee dos veces, se va a buscar una sola vez al disco.
- La FAT ya está cargada en memoria (el resto de los bloques a leer deberán ser contemplados).

Por convención, el primer byte de un archivo es el número 0. Si escribimos [0-4] significa todos los bytes del rango 0 a 4, incluyendo ambos extremos del rango.

Puede dejar expresado el resultado como potencias de dos. Justificar.

# Ejercicio 1 - Solución

Archivo: `/home/el/parcial.avi`. Bytes: [3.000-3.083].

- Tengo la entrada de `root` (distinguido), busco en la FAT el 1er bloque de la tabla de directorios para hallar la entrada de `home`. Cuando la encuentro, busco en la FAT el 1er bloque de la tabla de directorios para hallar la entrada de `el`. Lo mismo para `parcial.avi`. **3 LECTURAS DE DISCO**.
- Ahora miro en qué bloques están los bytes pedidos. Como los bloques son de 2KiB, tengo los bytes [0-2047] en el primer bloque y [2048-4095] en el segundo, aquí están los bytes que necesito. Para llegar al segundo bloque simplemente navego la FAT (que ya está en memoria) hasta saber cuál es el segundo bloque, y lo traigo. **1 LECTURA DE DISCO**.
- **TOTAL: 4 LECTURAS DE DISCO**.

# Directorios en ext2

- En ext2, **a cada directorio le corresponde un inodo**.

# Directorios en ext2

- En ext2, **a cada directorio le corresponde un inodo**.
- Las entradas de directorio son de longitud variable, permitiendo tener nombres de archivo más grandes.

# Directorios en ext2

- En ext2, **a cada directorio le corresponde un inodo**.
- Las entradas de directorio son de longitud variable, permitiendo tener nombres de archivo más grandes.
  - Ojo: una entrada puede estar repartida en más de un bloque.

# Directorios en ext2

- En ext2, **a cada directorio le corresponde un inodo**.
- Las entradas de directorio son de longitud variable, permitiendo tener nombres de archivo más grandes.
  - Ojo: una entrada puede estar repartida en más de un bloque.
- Cada entrada contiene la longitud de la entrada, el nombre del archivo, y el número de inodo al que refiere. El resto de metadatos están en cada inodo.

# Directorios en ext2

- En ext2, **a cada directorio le corresponde un inodo**.
- Las entradas de directorio son de longitud variable, permitiendo tener nombres de archivo más grandes.
  - Ojo: una entrada puede estar repartida en más de un bloque.
- Cada entrada contiene la longitud de la entrada, el nombre del archivo, y el número de inodo al que refiere. El resto de metadatos están en cada inodo.
- Las primeras dos entradas en todos los directorios son ‘‘.’’ y ‘‘..’’.

# Directorios en ext2

- En ext2, **a cada directorio le corresponde un inodo**.
- Las entradas de directorio son de longitud variable, permitiendo tener nombres de archivo más grandes.
  - Ojo: una entrada puede estar repartida en más de un bloque.
- Cada entrada contiene la longitud de la entrada, el nombre del archivo, y el número de inodo al que refiere. El resto de metadatos están en cada inodo.
- Las primeras dos entradas en todos los directorios son ‘‘.’’ y ‘‘..’’.
- Al igual que en FAT, el inodo del directorio **root** es distinguido: es siempre el inodo número 2.

# Ejercicio 2

Contamos con dos sistemas de archivos basados en inodos: FSA, con bloques de 1KiB; y FSB, con bloques de 2KiB. Ambos tienen 2 entradas directas, 2 indirectas, 1 doble indirecta y 1 triple indirecta. Las direcciones de bloques ocupan 2 bytes.

Indicar a cuántos bloques de disco hay que acceder para leer los bytes que se indican para cada archivo:

1. FSA: archivo: `/quiero.txt`. Bytes [0-5], [12-17].
2. FSB: archivo: `/home/aprobar.txt`. Bytes [6.500-6.600], [3.500.000].

Se puede asumir que:

- Los archivos y directorios a buscar se encuentran siempre en el primer bloque de *directory entries* correspondientes al directorio padre.
- Si un bloque se lee dos veces, se va a buscar una sola vez al disco.

Puede dejar expresado el resultado como potencias de dos. Justificar.

# Ejercicio 2 - Solución

FSA: archivo: `/quiero.txt`. Bytes [0-5], [12-17].

- Primero tengo que saber en qué inodo buscar. Para eso miro el inodo de `root` (distinguido), traigo el primer bloque de entradas de directorio (sé que lo encuentro ahí) y busco la entrada de directorio del archivo `quiero.txt`. **1 LECTURA DE DISCO**.
- Una vez que identifico el inodo, lo miro (ya está en memoria). Quiero saber en qué bloque están los bytes que preciso. Como cada bloque es de 1KiB, la primer entrada directa apunta al bloque con los bytes [0-1023]. Ahí se encuentran los bytes solicitados. **1 LECTURA DE DISCO**.
- **TOTAL: 2 LECTURAS DE DISCO**.

# Ejercicio 2 - Solución

FSB: archivo: `/home/aprobar.txt`. Bytes [6.500-6.600], [3.500.000].

- Para saber en qué inodo buscar,miro el inodo de `root` (distinguido), traigo el primer bloque de entradas de directorio (sé que lo encuentro ahí) y busco la entrada de directorio del archivo `home`. Luego hago lo mismo con el directorio `home` para encontrar la entrada de `aprobar.txt`. **2 LECTURAS DE DISCO**.
- Una vez que identifico el inodo (ya está en memoria), quiero saber en qué bloque están los bytes que preciso. Como cada bloque es de 2KiB, las dos primeras entradas directas apuntan a los bloques con los bytes [0-2047] y [2048-4095]. Como no me alcanza, tengo que mirar el primer indirecto.
- Como cada bloque es de 2KiB y las direcciones de bloque son de 2 Bytes, tenemos $2^{10}$ direcciones por bloque. Esto significa que tengo $2^{10} \cdot 2 \cdot 2^{10} = 2^{21}$ bytes direccionados en el primer indirecto. Ahí se encuentran los bytes [6.500-6.600]. **2 LECTURAS DE DISCO**
- Para el siguiente byte solicitado, el primer indirecto no me alcanza. El segundo direcciona los siguientes $2^{21}$ bytes y ahí está el byte [3.500.000]. **2 LECTURAS DE DISCO**.
- **TOTAL: 6 LECTURAS DE DISCO**.

# Enlaces

- En los sistemas de archivos con inodos, el nombre de los archivos **no aparece** en los inodos. Así, podemos referenciar el mismo inodo con diferentes nombres, y desde más de un directorio.

# Enlaces

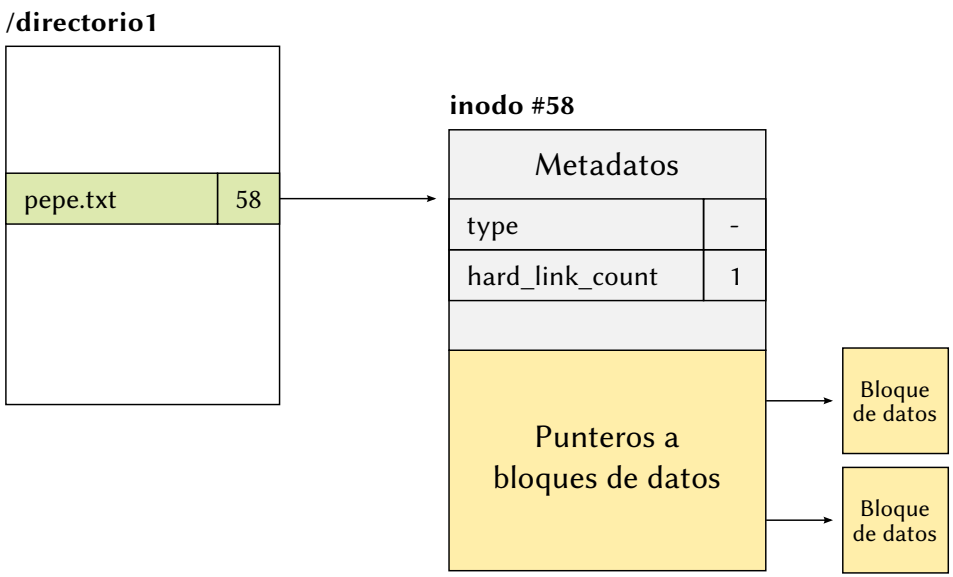
- En los sistemas de archivos con inodos, el nombre de los archivos **no aparece** en los inodos. Así, podemos referenciar el mismo inodo con diferentes nombres, y desde más de un directorio.
- Esto se conoce como **enlace duro** o **físico** (*hard link*).

# Enlaces

- En los sistemas de archivos con inodos, el nombre de los archivos **no aparece** en los inodos. Así, podemos referenciar el mismo inodo con diferentes nombres, y desde más de un directorio.
- Esto se conoce como **enlace duro** o **físico** (*hard link*).
- El nombre de archivo "." en un directorio es un enlace duro al mismo directorio. El nombre de archivo ".." es un enlace duro al directorio padre.

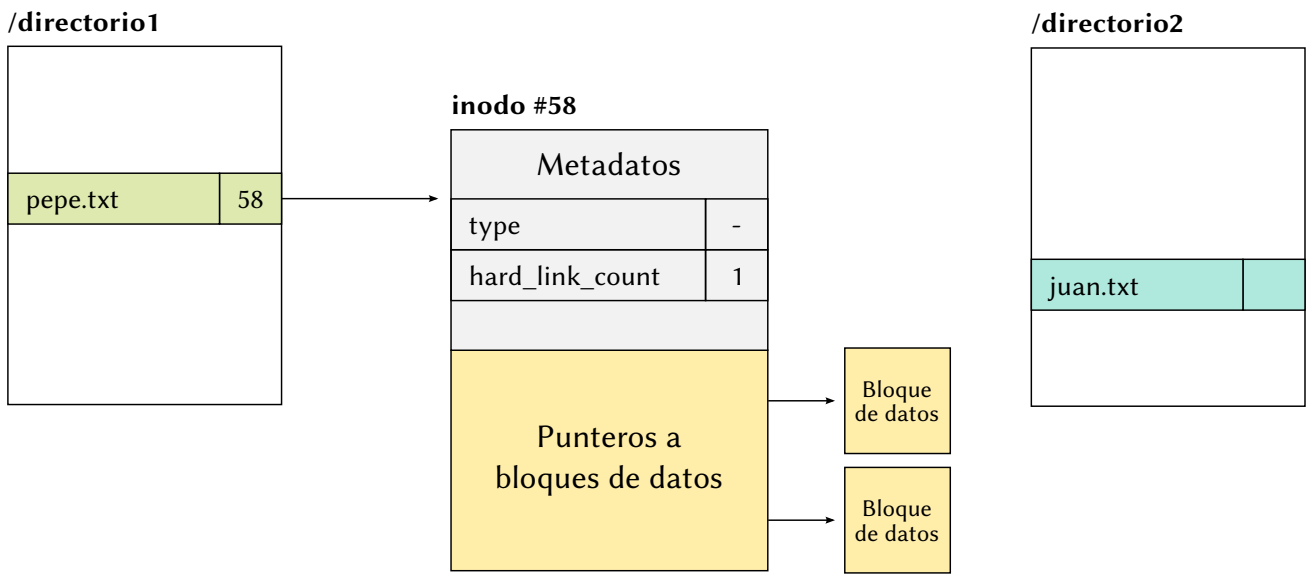
# Enlaces

- En los sistemas de archivos con inodos, el nombre de los archivos **no aparece** en los inodos. Así, podemos referenciar el mismo inodo con diferentes nombres, y desde más de un directorio.
- Esto se conoce como **enlace duro** o **físico** (*hard link*).
- El nombre de archivo "." en un directorio es un enlace duro al mismo directorio. El nombre de archivo ".." es un enlace duro al directorio padre.



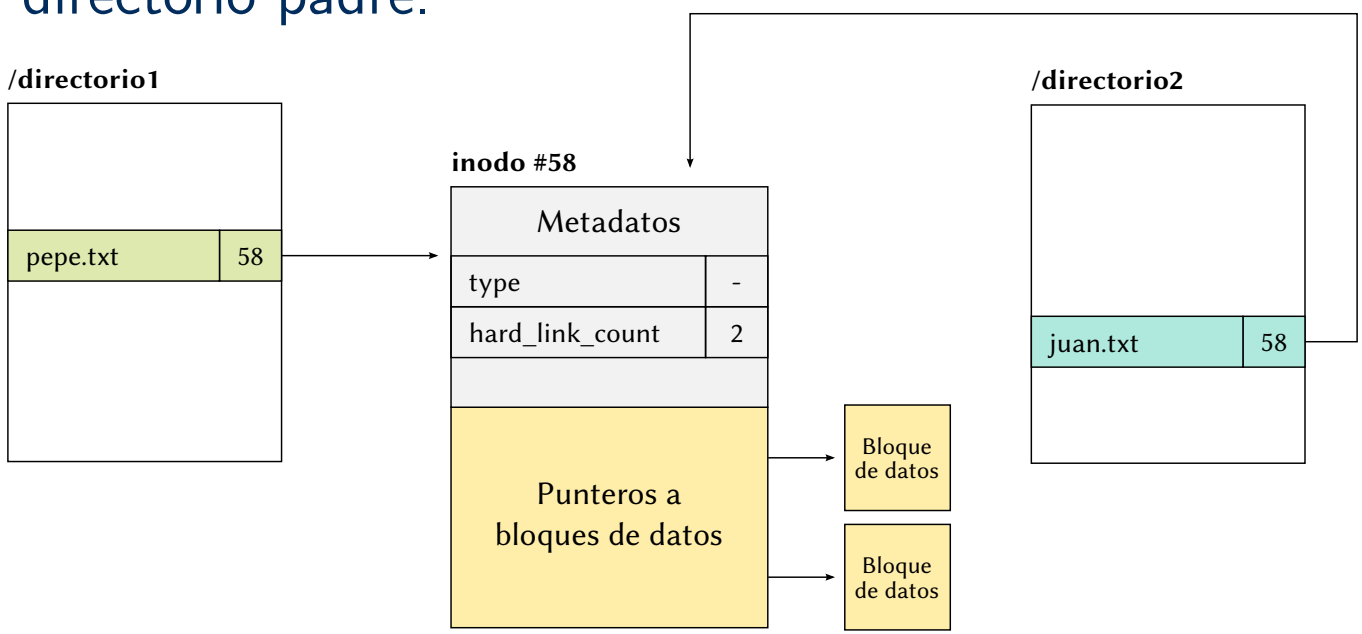
# Enlaces

- En los sistemas de archivos con inodos, el nombre de los archivos **no aparece** en los inodos. Así, podemos referenciar el mismo inodo con diferentes nombres, y desde más de un directorio.
- Esto se conoce como **enlace duro** o **físico** (*hard link*).
- El nombre de archivo "." en un directorio es un enlace duro al mismo directorio. El nombre de archivo ".." es un enlace duro al directorio padre.



# Enlaces

- En los sistemas de archivos con inodos, el nombre de los archivos **no aparece** en los inodos. Así, podemos referenciar el mismo inodo con diferentes nombres, y desde más de un directorio.
- Esto se conoce como **enlace duro** o **físico** (*hard link*).
- El nombre de archivo "." en un directorio es un enlace duro al mismo directorio. El nombre de archivo ".." es un enlace duro al directorio padre.



# Enlaces

- ¿Cómo hacemos para borrar un archivo que puede tener enlaces?

# Enlaces

- ¿Cómo hacemos para borrar un archivo que puede tener enlaces? Podemos preservar el archivo hasta que se borren todas las referencias.

# Enlaces

- ¿Cómo hacemos para borrar un archivo que puede tener enlaces? Podemos preservar el archivo hasta que se borren todas las referencias.
- ¿Cómo sé si ya borré todas las referencias?

# Enlaces

- ¿Cómo hacemos para borrar un archivo que puede tener enlaces? Podemos preservar el archivo hasta que se borren todas las referencias.
- ¿Cómo sé si ya borré todas las referencias? Se mantiene la cuenta de todas las referencias al archivo en el inodo. Cuando se crea un enlace, se incrementa el contador. Cuando un enlace se borra, se decrementa el contador. El archivo se borra cuando el contador está en cero.

# Enlaces

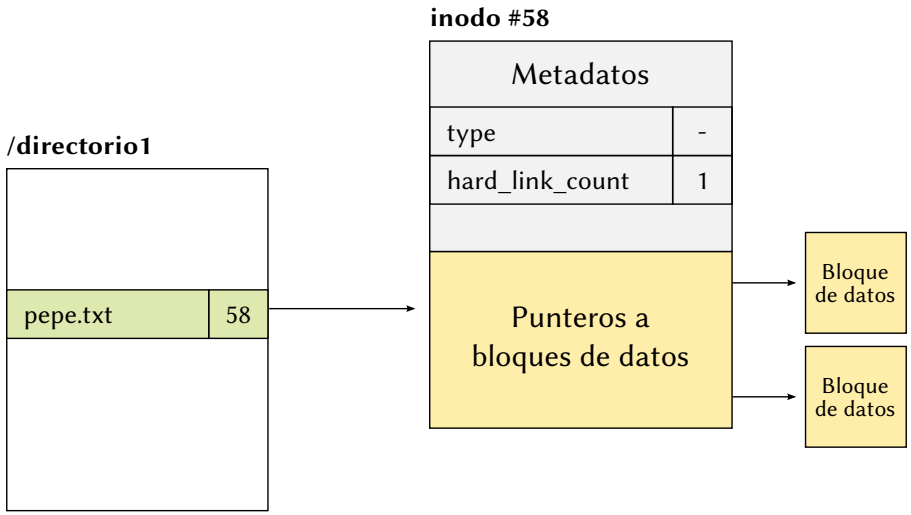
- También podemos crear **enlaces simbólicos** (*symbolic links*). Estos se implementan mediante un inodo adicional, donde se almacena el destino del enlace.

# Enlaces

- También podemos crear **enlaces simbólicos** (*symbolic links*). Estos se implementan mediante un inodo adicional, donde se almacena el destino del enlace.
- Permiten referenciar directorios en otros sistemas de archivos.

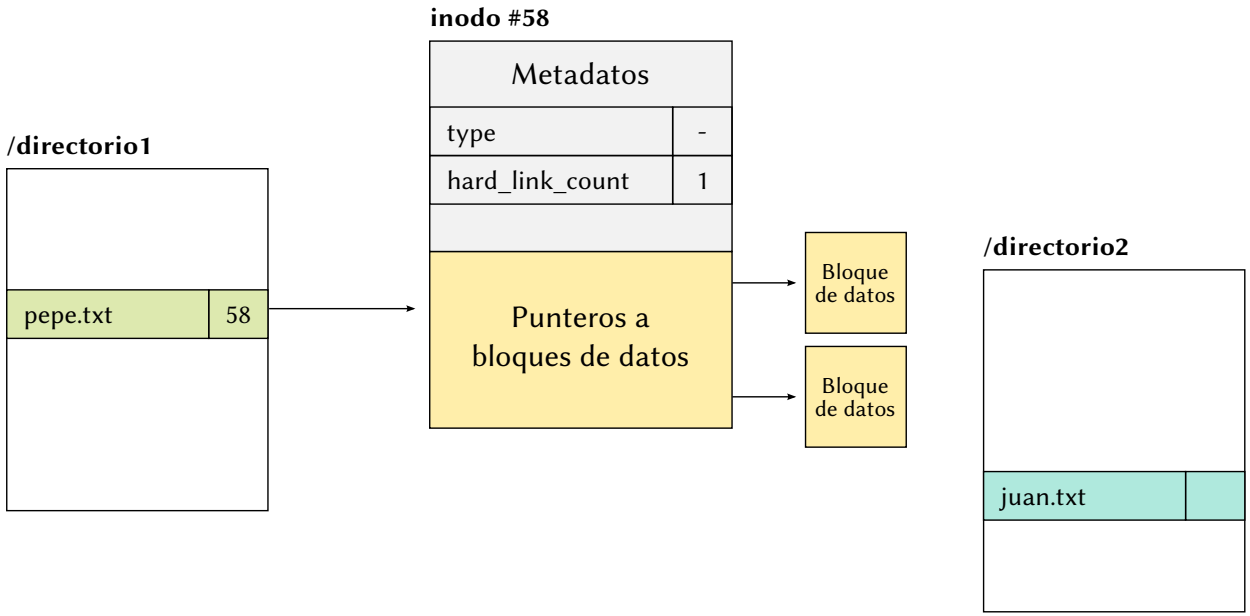
# Enlaces

- También podemos crear **enlaces simbólicos** (*symbolic links*). Estos se implementan mediante un inodo adicional, donde se almacena el destino del enlace.
- Permiten referenciar directorios en otros sistemas de archivos.



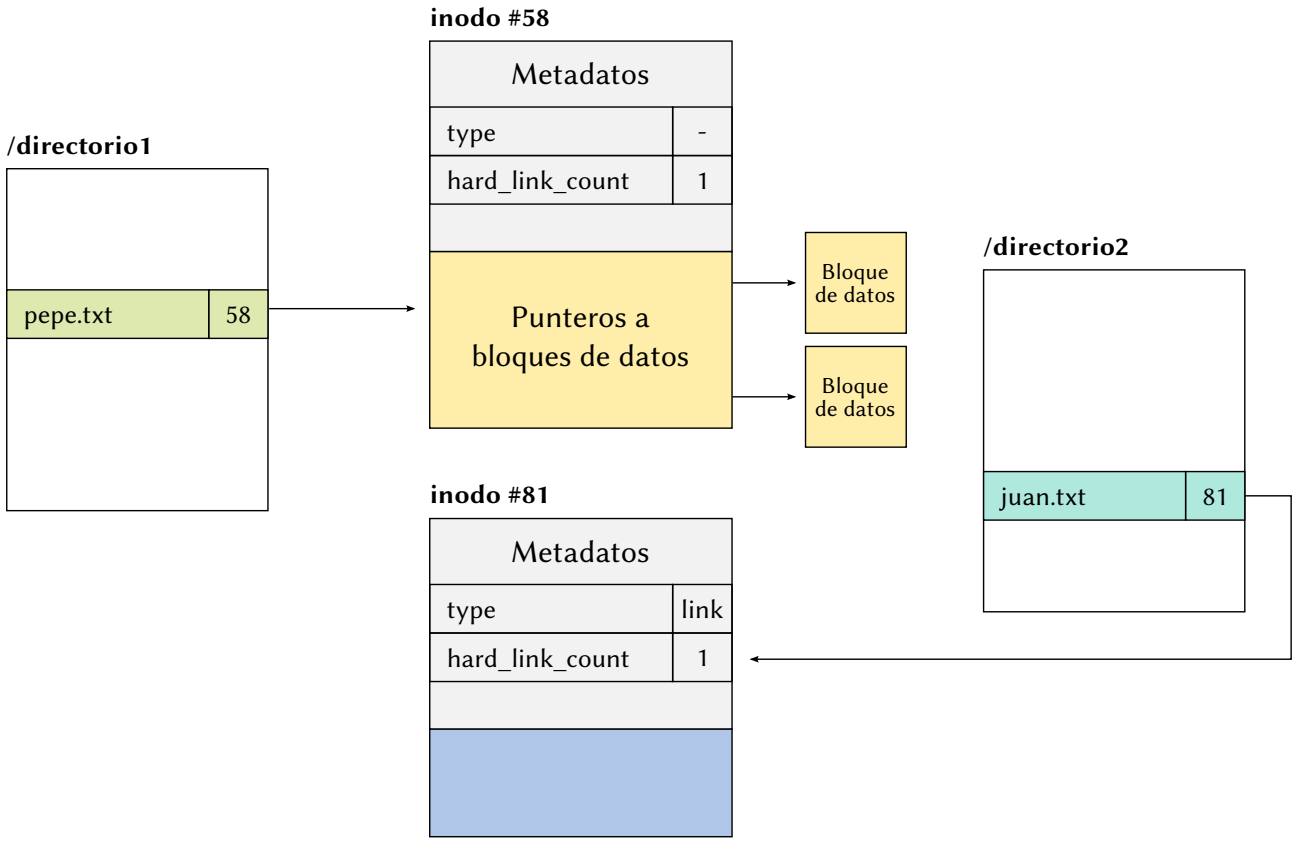
# Enlaces

- También podemos crear **enlaces simbólicos** (*symbolic links*). Estos se implementan mediante un inodo adicional, donde se almacena el destino del enlace.
- Permiten referenciar directorios en otros sistemas de archivos.



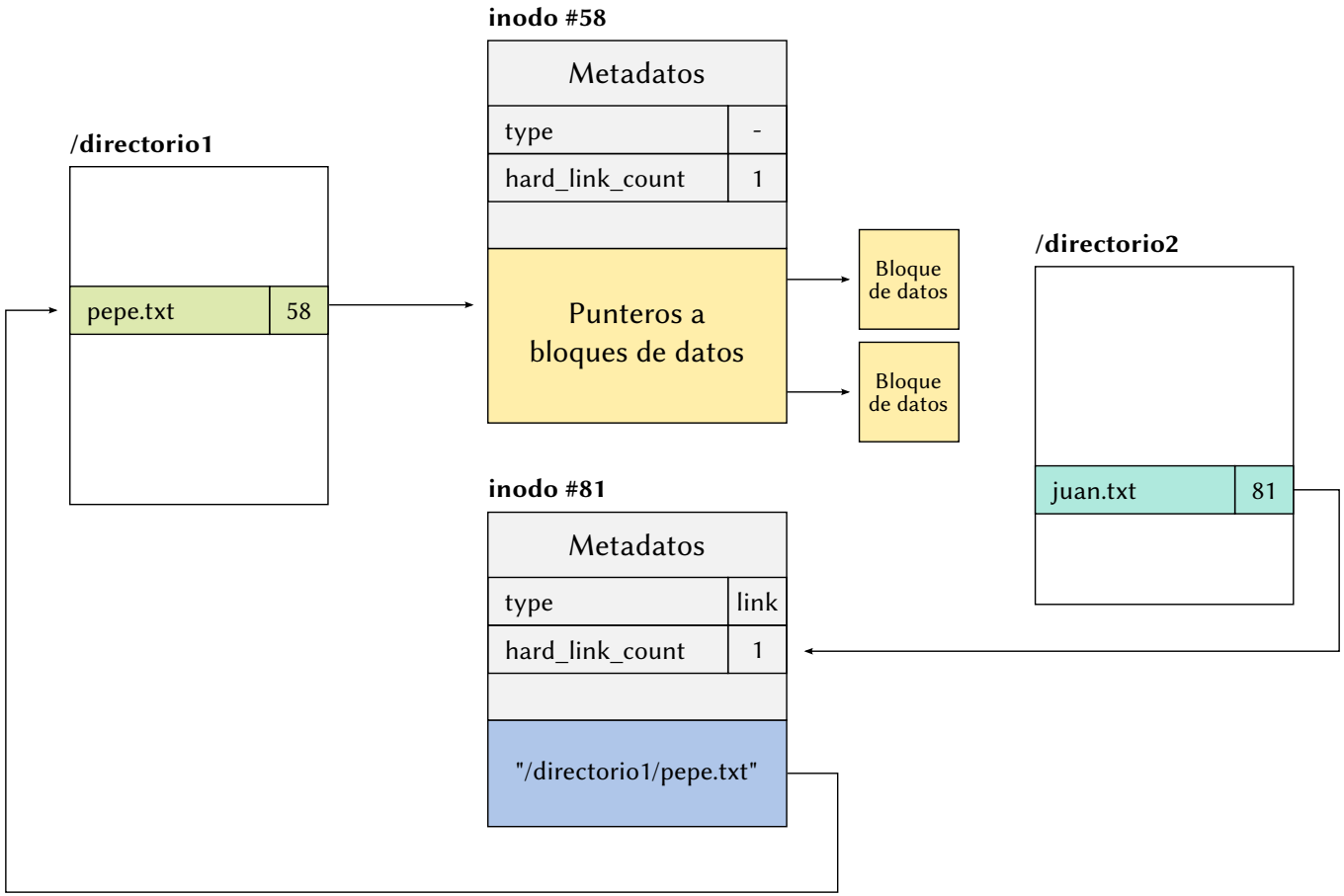
# Enlaces

- También podemos crear **enlaces simbólicos** (*symbolic links*). Estos se implementan mediante un inodo adicional, donde se almacena el destino del enlace.
- Permiten referenciar directorios en otros sistemas de archivos.



# Enlaces

- También podemos crear **enlaces simbólicos** (*symbolic links*). Estos se implementan mediante un inodo adicional, donde se almacena el destino del enlace.
- Permiten referenciar directorios en otros sistemas de archivos.



# Enlaces

- ¿Qué pasa si borramos un enlace simbólico?

# Enlaces

- ¿Qué pasa si borramos un enlace simbólico? No se lleva una cuenta de los enlaces simbólicos. Si se borra el enlace, el archivo original sigue igual.

# Enlaces

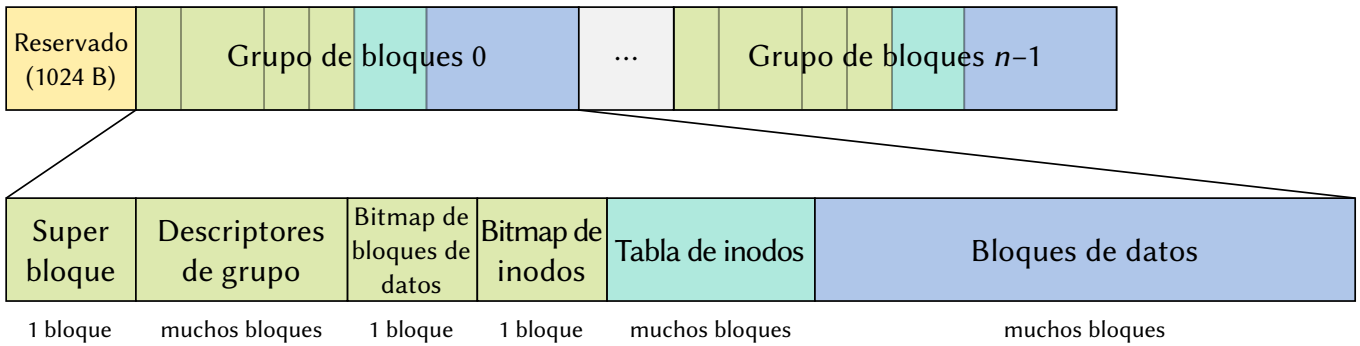
- ¿Qué pasa si borramos un enlace simbólico? No se lleva una cuenta de los enlaces simbólicos. Si se borra el enlace, el archivo original sigue igual.
- ¿Qué pasa si borramos un archivo que está siendo referenciado por un enlace simbólico?

# Enlaces

- ¿Qué pasa si borramos un enlace simbólico? No se lleva una cuenta de los enlaces simbólicos. Si se borra el enlace, el archivo original sigue igual.
- ¿Qué pasa si borramos un archivo que está siendo referenciado por un enlace simbólico? El archivo no sabe que hay referencias simbólicas. Si se borra el archivo, se libera el espacio correspondiente y el enlace queda roto.

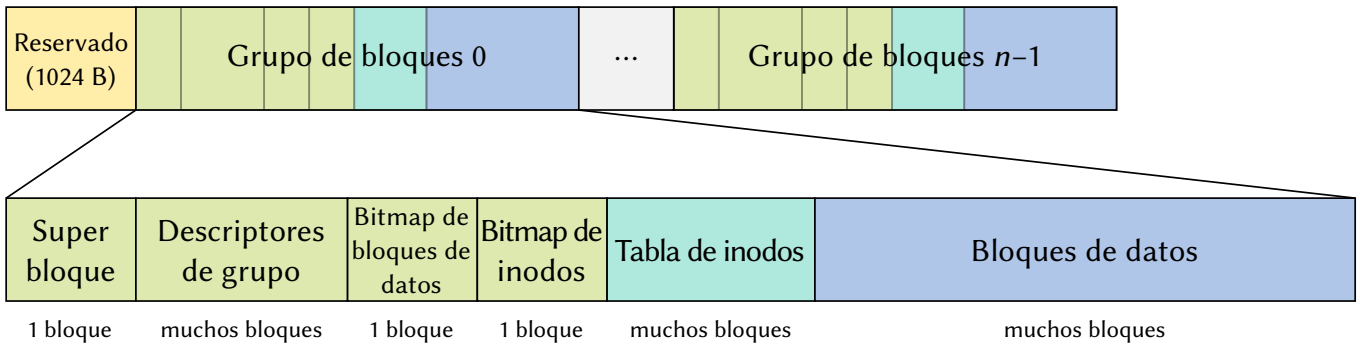
# ¿Y dónde están los inodos?

- En un sistema de archivos ext2, los bloques del disco están particionados en **grupos de bloques** contiguos.



# ¿Y dónde están los inodos?

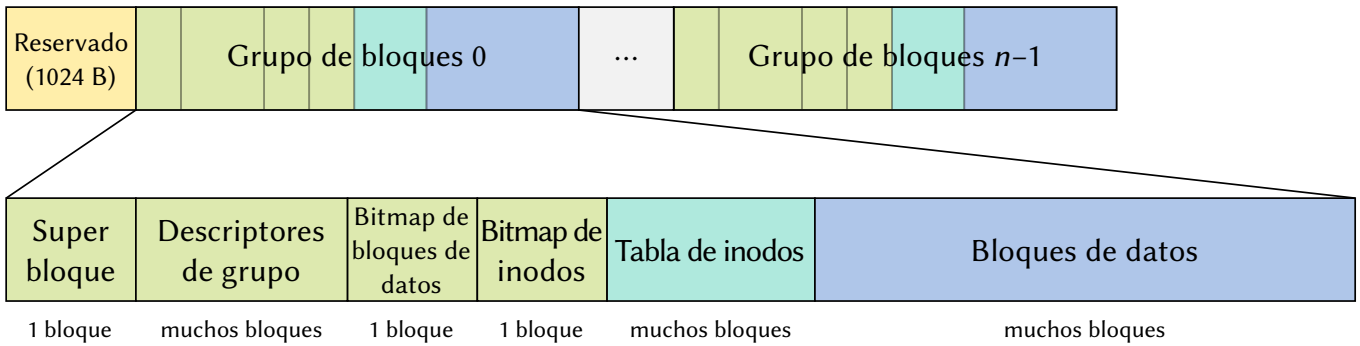
- En un sistema de archivos ext2, los bloques del disco están particionados en **grupos de bloques** contiguos.




- En cada grupo, el primer bloque es el **superbloque**, que contiene información sobre el FS, incluyendo la cantidad de inodos y cantidad de bloques de disco.

# ¿Y dónde están los inodos?

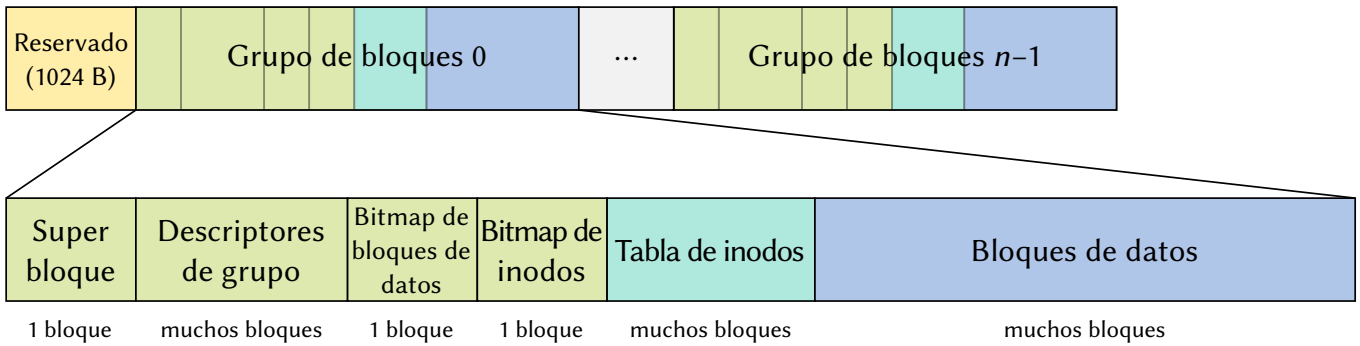
- En un sistema de archivos ext2, los bloques del disco están particionados en **grupos de bloques** contiguos.




- En cada grupo, el primer bloque es el **superbloque**, que contiene información sobre el FS, incluyendo la cantidad de inodos y cantidad de bloques de disco.
- Luego de información sobre bloques e inodos libres, siguen los **inodos** en sí mismos.

# ¿Y dónde están los inodos?

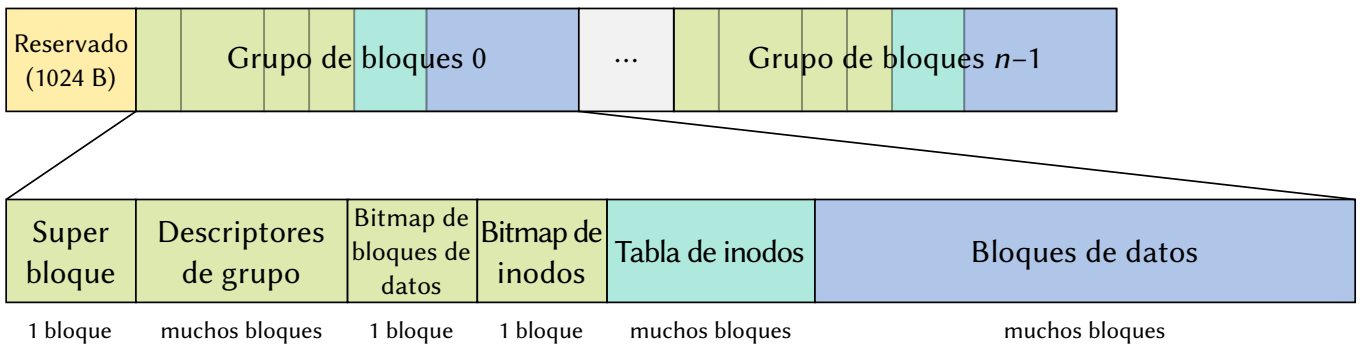
- En un sistema de archivos ext2, los bloques del disco están particionados en **grupos de bloques** contiguos.




- En cada grupo, el primer bloque es el **superbloque**, que contiene información sobre el FS, incluyendo la cantidad de inodos y cantidad de bloques de disco.
- Luego de información sobre bloques e inodos libres, siguen los **inodos** en sí mismos.
- Tras la tabla de inodos están los **bloques de datos**, donde se almacenan todos los archivos y directorios.

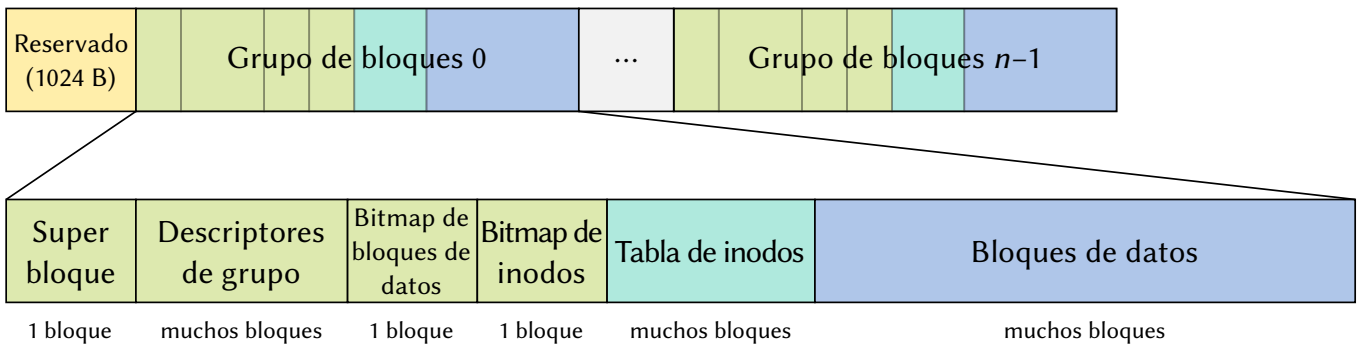
# ¿Y dónde están los inodos?

- En un sistema de archivos ext2, los bloques del disco están particionados en **grupos de bloques** contiguos.



# ¿Y dónde están los inodos?

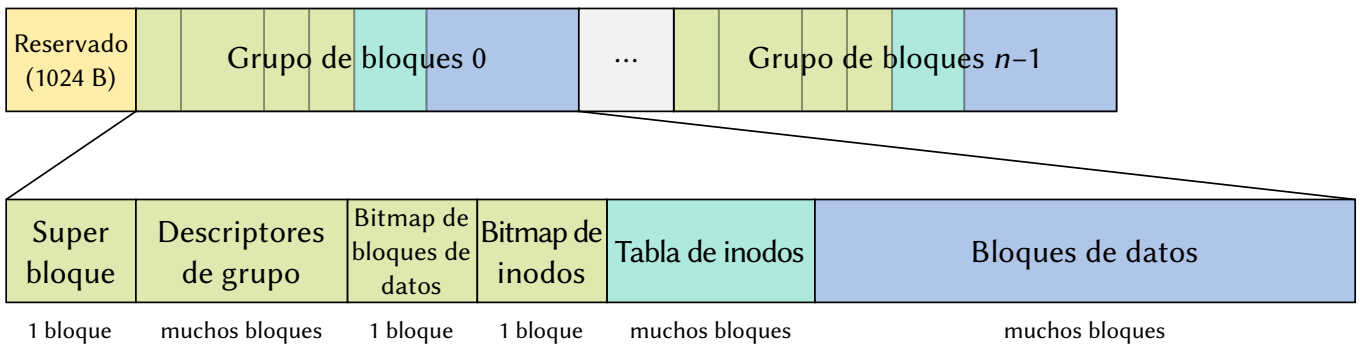
- En un sistema de archivos ext2, los bloques del disco están particionados en **grupos de bloques** contiguos.




- Esto quiere decir que los inodos están *repartidos* a lo largo de todo el disco.

# ¿Y dónde están los inodos?

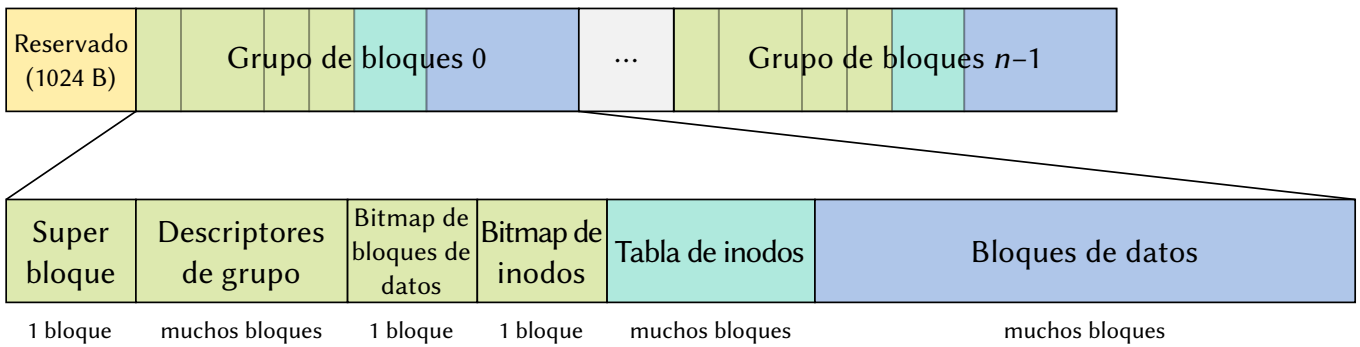
- En un sistema de archivos ext2, los bloques del disco están particionados en **grupos de bloques** contiguos.




- Esto quiere decir que los inodos están *repartidos* a lo largo de todo el disco.
- En un único bloque puede haber varios inodos.

# ¿Y dónde están los inodos?

- En un sistema de archivos ext2, los bloques del disco están particionados en **grupos de bloques** contiguos.




- Esto quiere decir que los inodos están *repartidos* a lo largo de todo el disco.
- En un único bloque puede haber varios inodos.
- Más detalles, en el **taller de ext2**.

Resumen de la clase:

- Analizamos distintos enfoques de sistemas de archivos.
  - Asignación contigua de bloques.
  - FAT
  - inodos
- Vimos cómo se manejan los directorios en FAT32 y Ext2.

Próxima clase:

- Taller ext2

Con esto se puede comenzar la guía práctica 4.


SO ASK ME
QUESTIONS
quickmeme.com