

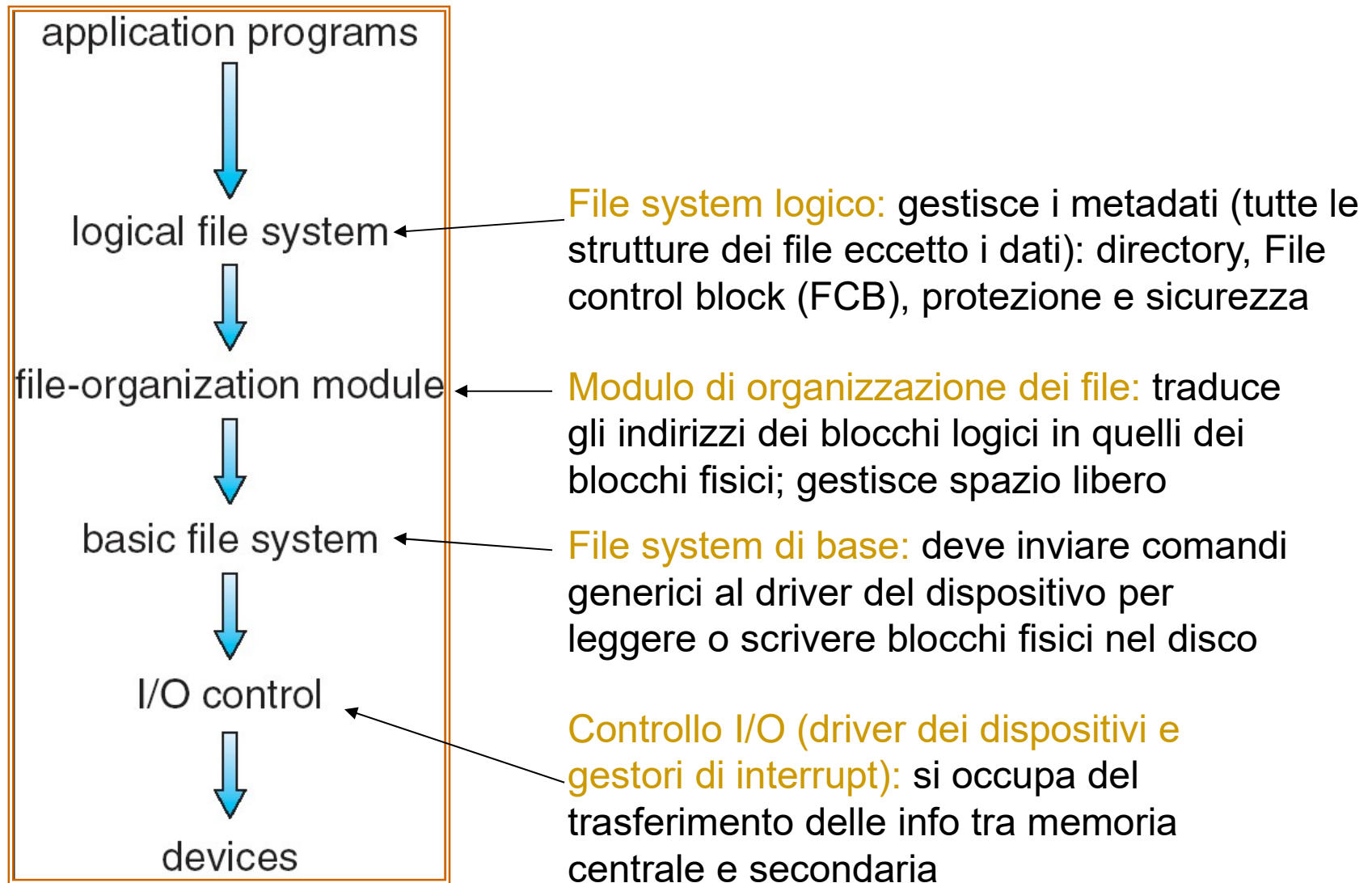
# Implementazione del File System

Capitolo 11 -- Silberschatz

# Implementazione del File System

- File system:
  - Definizione dell'aspetto del sistema agli occhi dell'utente
  - Algoritmi e strutture dati che permettono di far corrispondere il file system logico ai dispositivi fisici
- Il file system risiede in un'unità di memorizzazione secondaria (disco)
  - Per **leggere** un blocco dal disco, lo si deve portare in memoria principale
  - Per **modificarlo**, lo si deve portare in memoria principale e dopo averlo modificato, lo si riscrive nella stessa posizione del disco
  - Al blocco ci si può accedere con accesso diretto, muovendo opportunamente la testina di lettura
- Il file system è organizzato in livelli.

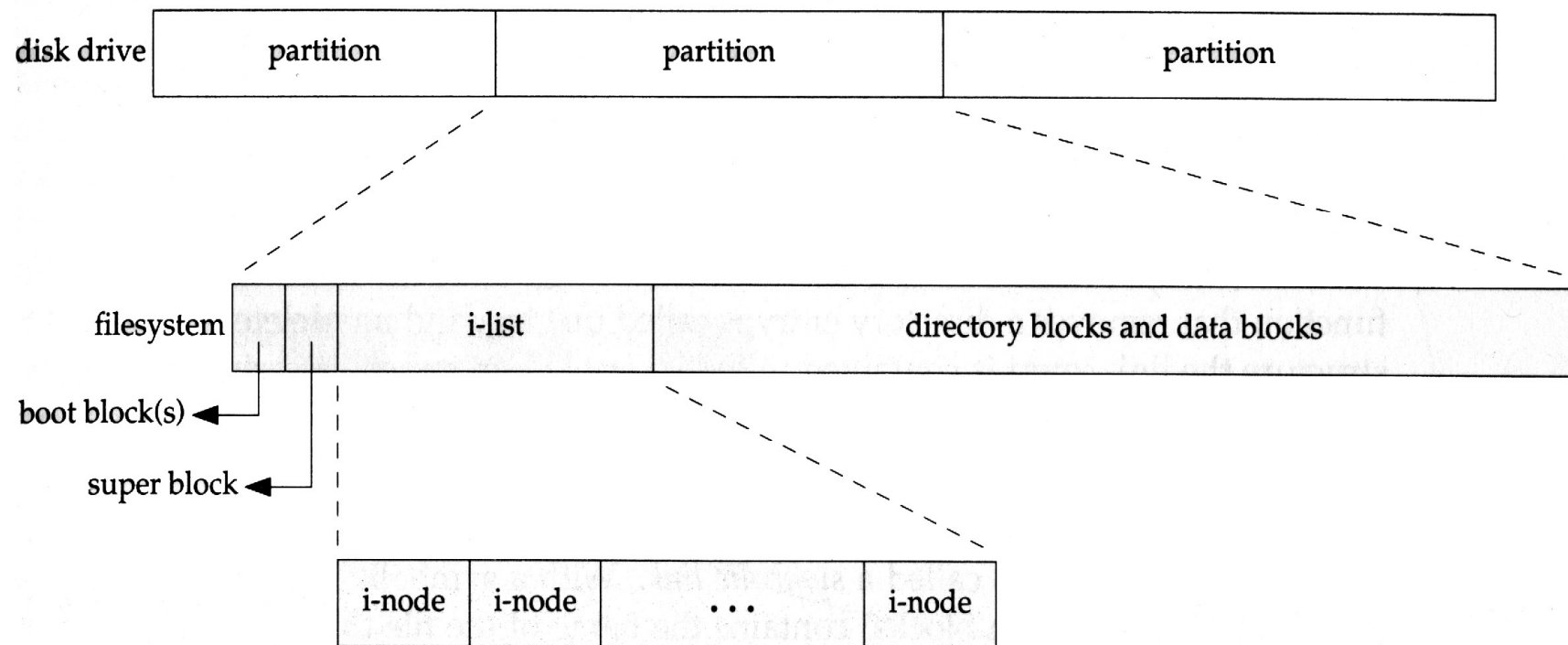
# File System a strati



# Strutture presenti sul disco

- **Boot control block** – Informazioni necessarie per il caricamento del sistema operativo (*UFS - boot block, NTFS - partition boot sector*)
- **Volume control block** – contiene dettagli del volume, numero di blocchi per partizione, taglia dei blocchi, blocchi liberi ...(*UFS – superblock, NTFS – master file table*)
- **Struttura delle directory** – usata per organizzare i file ...(*UFS – nomi file e i-node associato, NTFS – master file table*)
- **File control block (FCB)** – informazioni sul file (*UFS – inode*)

# Unix Filesystem (UFS)



# Strutture del file system in memoria

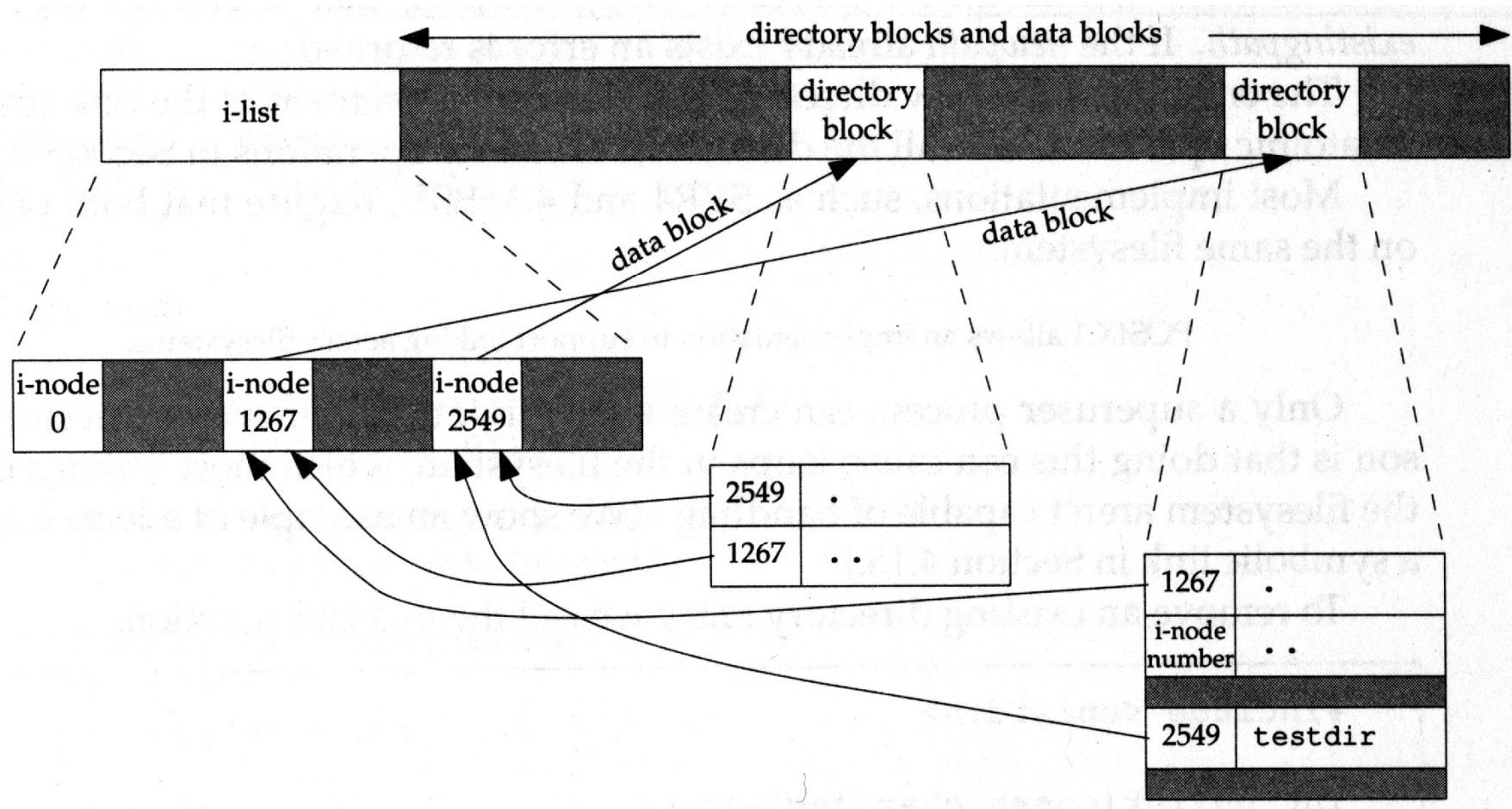
- Quando si monta un file system vengono caricate in memoria questa serie di informazioni che si eliminano solo allo smontaggio:
  - Tabella di montaggio (mount table) – contiene informazioni circa i volumi montati
  - Cache per le strutture delle directory recentemente accedute
  - Tabella dei file aperti di sistema
  - Tabella dei file aperti per processo

# File control block

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

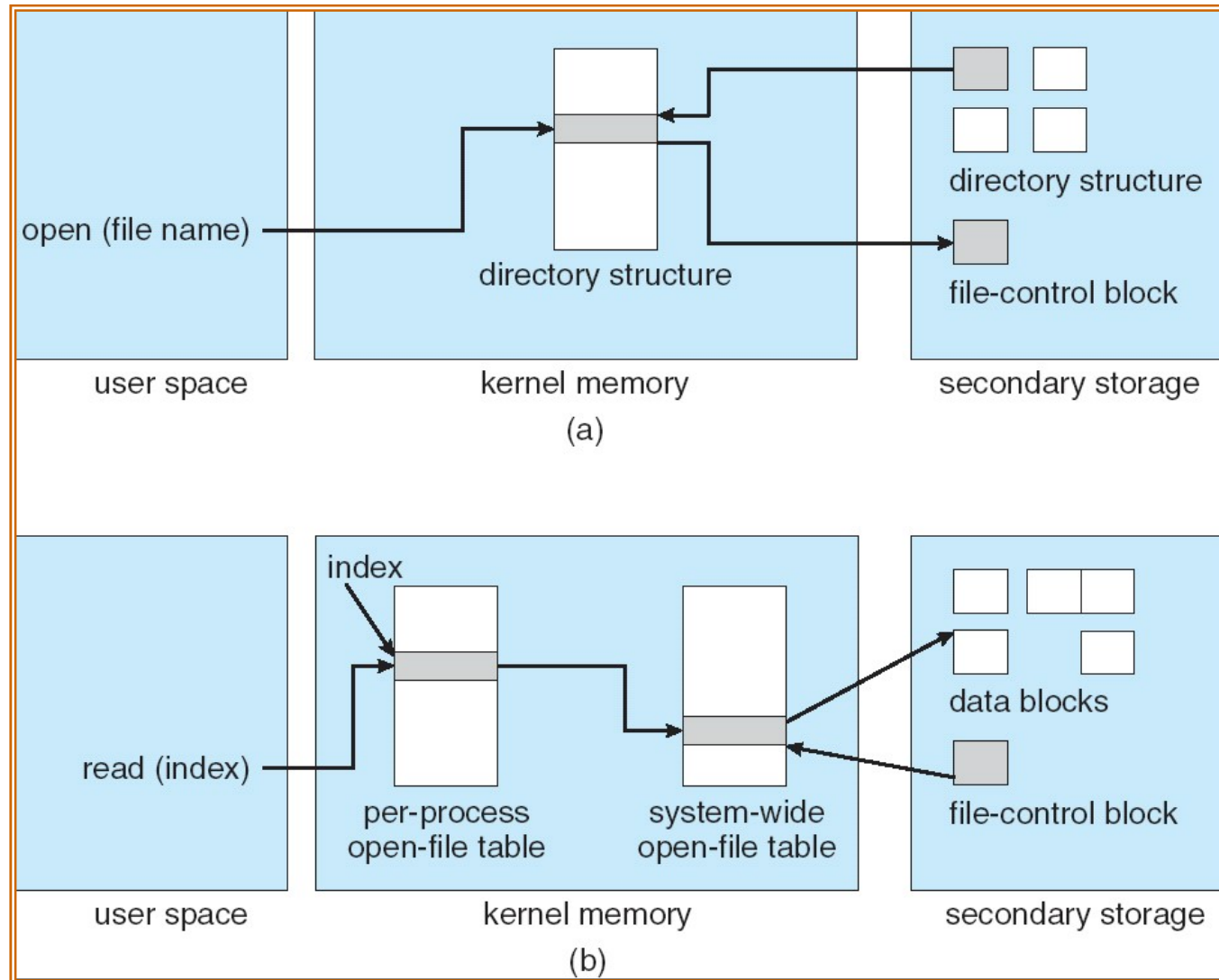
- Creazione di un file
  - Creazione (o allocazione) di un nuovo FCB
  - Carica in memoria la directory appropriata
  - Modifica la directory (nome file e FCB) e riscrittura su disco

# Esempio dell'organizzazione del filesystem di Unix

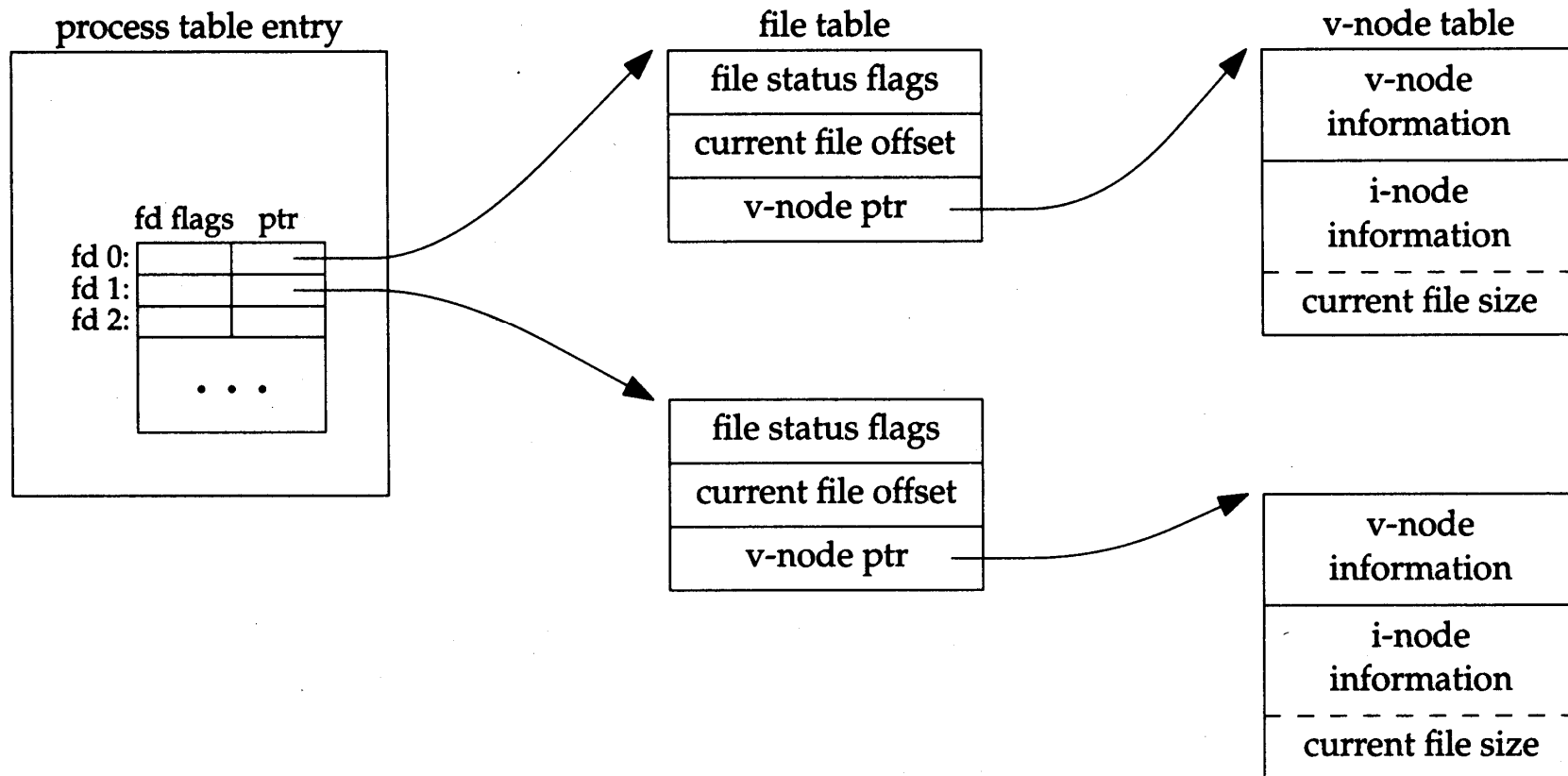




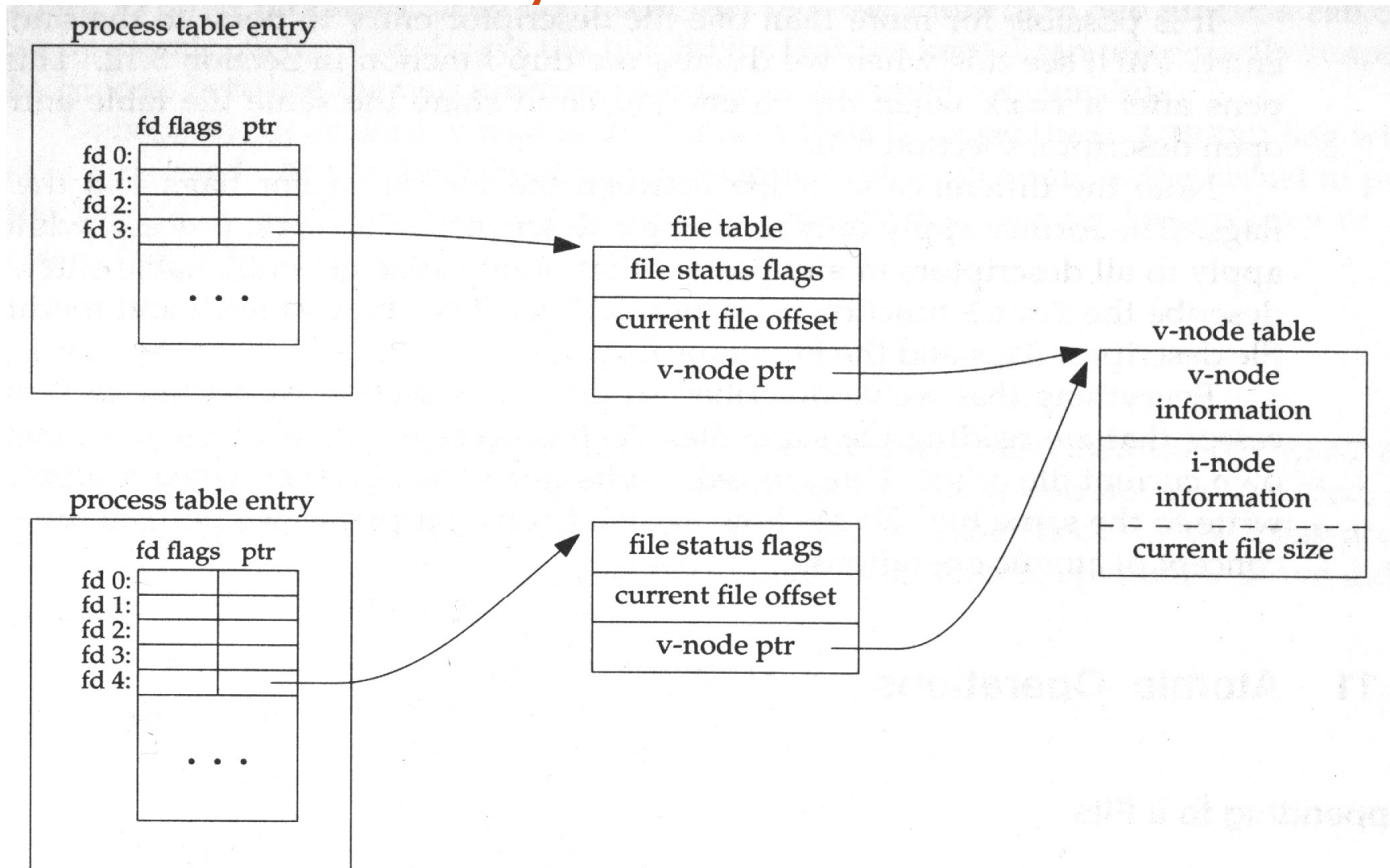
# open e read



# Strutture dati di file aperti nel FileSystem di Unix



# 2 processi su uno stesso file nel FileSystem di Unix



# Partizioni e montaggio

- Un disco può presentare diverse partizioni:
  - raw : non contiene alcun file system; è una parte del disco privo di struttura logica
  - coked : parte del disco con una struttura logica
- Root partition – contiene il kernel del sistema operativo e altri file. Viene montata in fase di boot
- La tabella di montaggio tiene traccia di tutti i file system montati
  - Windows monta ciascun volume in uno spazio di nomi separato (E:, F:, G: ...)
  - In Unix un file system può essere montato su ogni directory (un flag nell' i-node della directory indica che quella directory è un “punto di montaggio”)

# Implementazione di una directory

- Lista di nomi di file con puntatori ai blocchi dei dati:
  - semplice da programmare,
  - richiede tempo per la ricerca,
  - Lista ordinata, B-albero.
- Tabella hash – lista lineare con una tabella hash:
  - diminuisce il tempo di ricerca nella directory;
  - collisioni – due nomi di file vengono associati alla stessa posizione dalla funzione hash;
  - dimensione fissa.

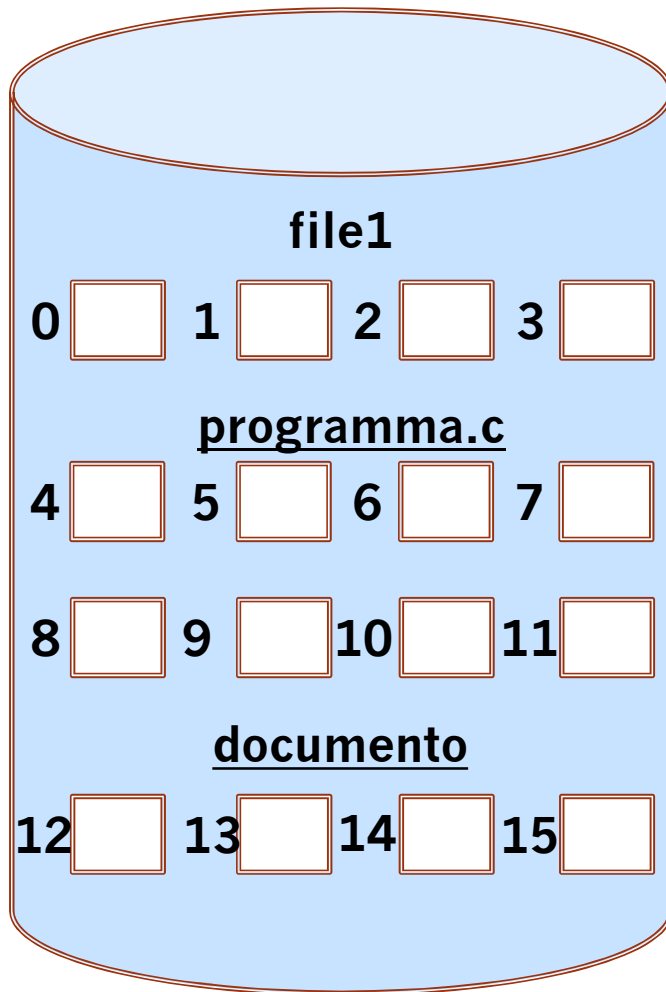
# Metodi di allocazione

- Un metodo di allocazione specifica il modo in cui i blocchi di un file vengono allocati nel disco:
  - Allocazione contigua.
  - Allocazione linkata.
  - Allocazione indicizzata.

# Allocazione contigua

- Ogni file occupa un certo numero di blocchi contigui su disco.
- Facile – è definita dall'indirizzo del primo blocco del file su disco e dalla lunghezza (numero di blocchi).
- Accesso
  - Accesso sequenziale: il FS memorizza l'indirizzo dell'ultimo blocco a cui si è acceduto → un nuovo accesso è immediato o al più necessita di accedere al blocco successivo
  - Accesso diretto: se si vuole accedere all' $i$ -mo blocco di un file che comincia al blocco  $b$ , si accede direttamente al blocco  $b + i - 1$

# Allocazione contigua dello spazio del disco

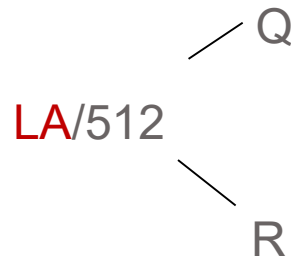


File	Blocco iniziale	Lunghezza (in blocchi)
file1	0	4
programma.c	4	8
documento	12	4



# Allocazione contigua

- Mappatura da logica a fisica
  - Assumiamo sa qui in avanti che un blocco fisico sia costituito da 512 byte



Blocco al quale accedere =  $Q + \text{blocco di partenza}$   
Indice nel blocco =  $R$

**LA** = Logical Address

Nota che  $0 \leq LA \leq |\text{file}|-1$

# Allocazione contigua

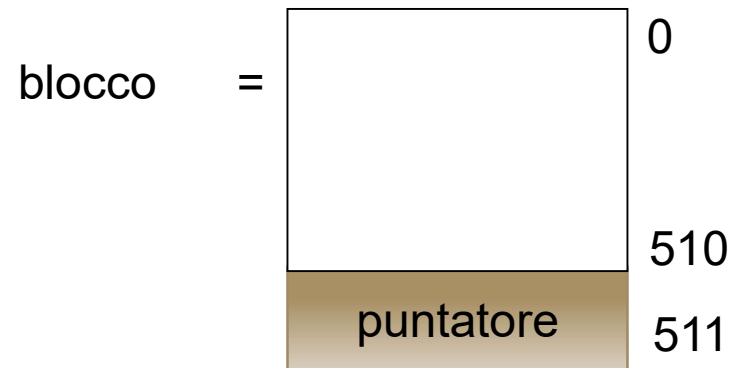
- Svantaggi:
  - Frammentazione esterna (problema dell'allocazione dinamica della memoria): assegnando e liberando lo spazio per i file, lo spazio libero del disco viene frammentato in tanti buchi.
  - La taglia dei file non può crescere.

# Estensione

- Molti nuovi file system (e.g., Veritas File System) usano uno schema di allocazione contigua modificato.
  - Inizialmente viene allocato un pezzo contiguo di spazio e poi, quando il pezzo non è più sufficientemente, viene aggiunta un' estensione.
  - Un' **estensione** è un altro pezzo di spazio contiguo. Un file consiste in una o più estensioni.

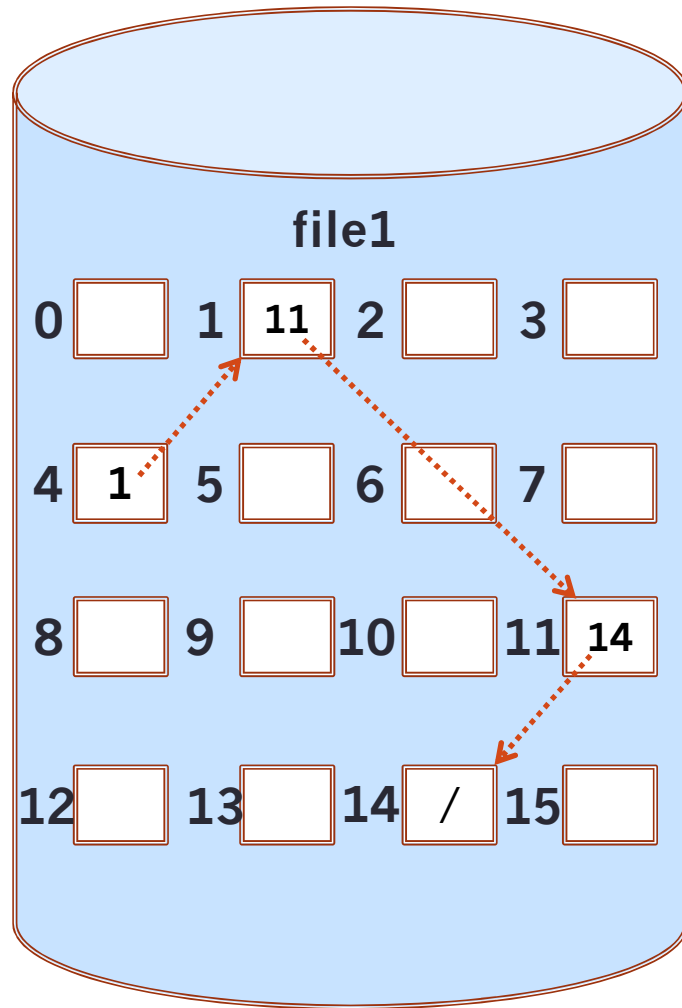
# Allocazione linkata

- Ogni file è una lista linkata di blocchi del disco: i blocchi possono essere sparpagliati ovunque nel disco.



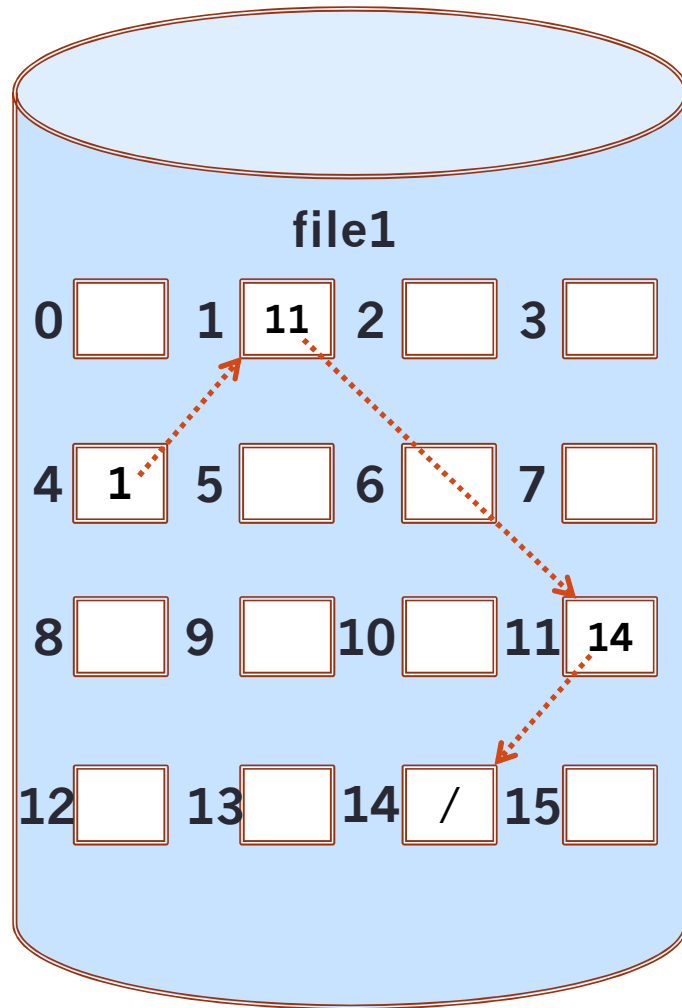
N.B. stiamo assumendo che il blocco sia di 512 parole che il puntatore al prossimo blocco sia costituito da 1 parola

# Allocazione Linkata



File	Blocco iniziale	Blocco finale
<i>MyFileConcatenato</i>	4	14

# Allocazione Linkata



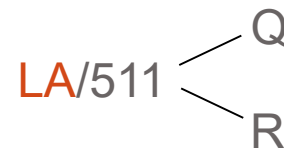
File	Blocco iniziale	Lunghezza (in blocchi)
<i>MyFileConcatenato</i>	4	4

# Allocazione linkata

- Una directory entry può consistere in:
  - puntatore al primo blocco e puntatore all'ultimo, oppure
  - puntatore al primo blocco e size del file.
- Gestione dello spazio libero – assenza di sprechi.
  - Creazione di un file: si crea un nuovo elemento nella directory, si mette a *nil* il puntatore al primo blocco e si inizializza la size a 0
  - Scrittura in un file: se quello da scrivere va oltre le dimensioni dell'ultimo blocco si cerca un nuovo blocco e lo si concatena alla fine

# Allocazione linkata

- Mapping indirizzi logici/fisici.



- Il blocco al quale accedere è il  $Q+1$ -esimo blocco nella lista linkata di blocchi che rappresentano il file.
- Indice nel blocco =  $R$

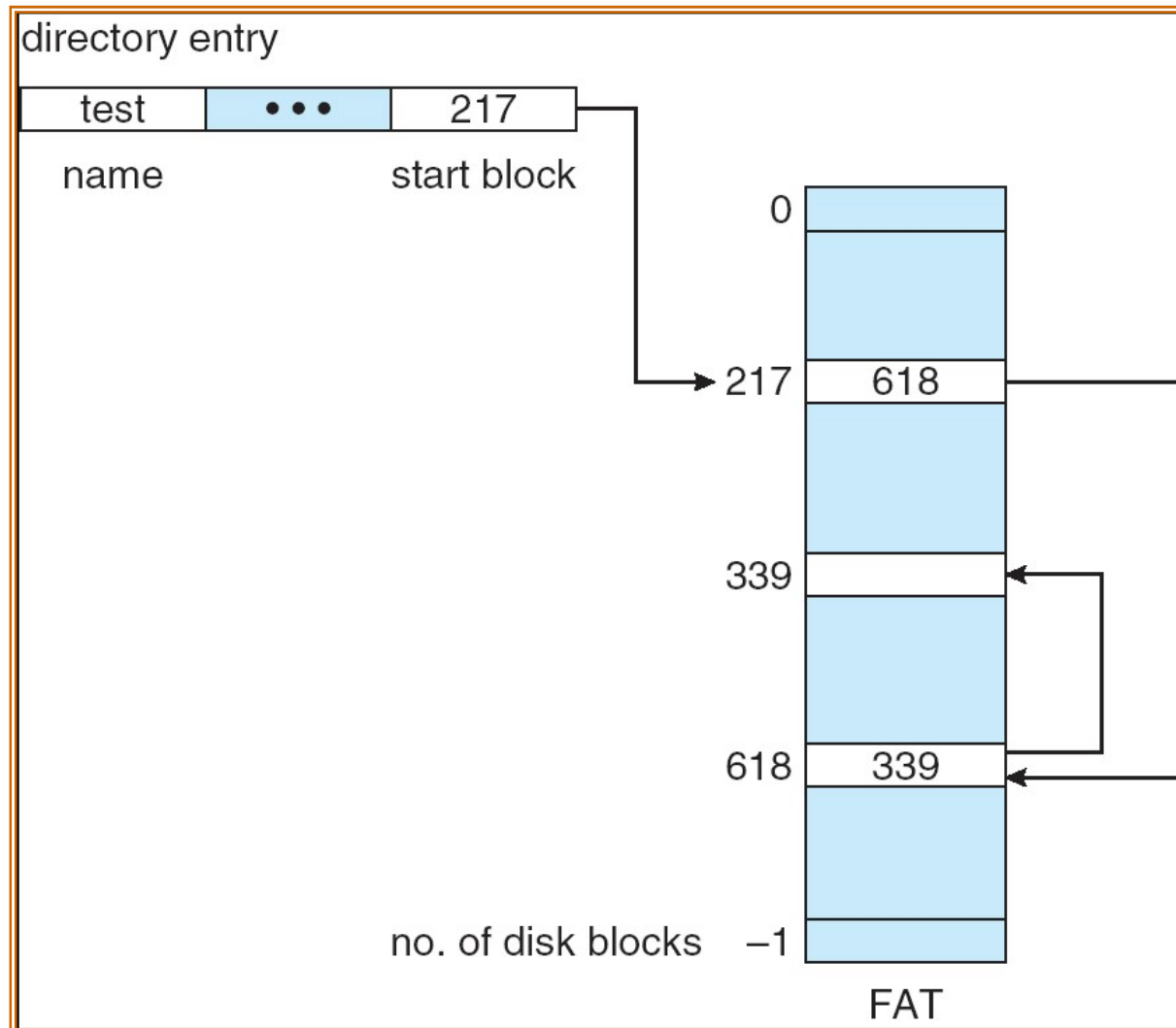
N.B. stiamo assumendo che il blocco sia di 512 parole che il puntatore al prossimo blocco sia costituito da 1 parola



# Allocazione linkata

- Svantaggi
  - Inefficiente l'accesso diretto.
  - Spreco di parte di un blocco per il puntatore.
  - Poca affidabilità.

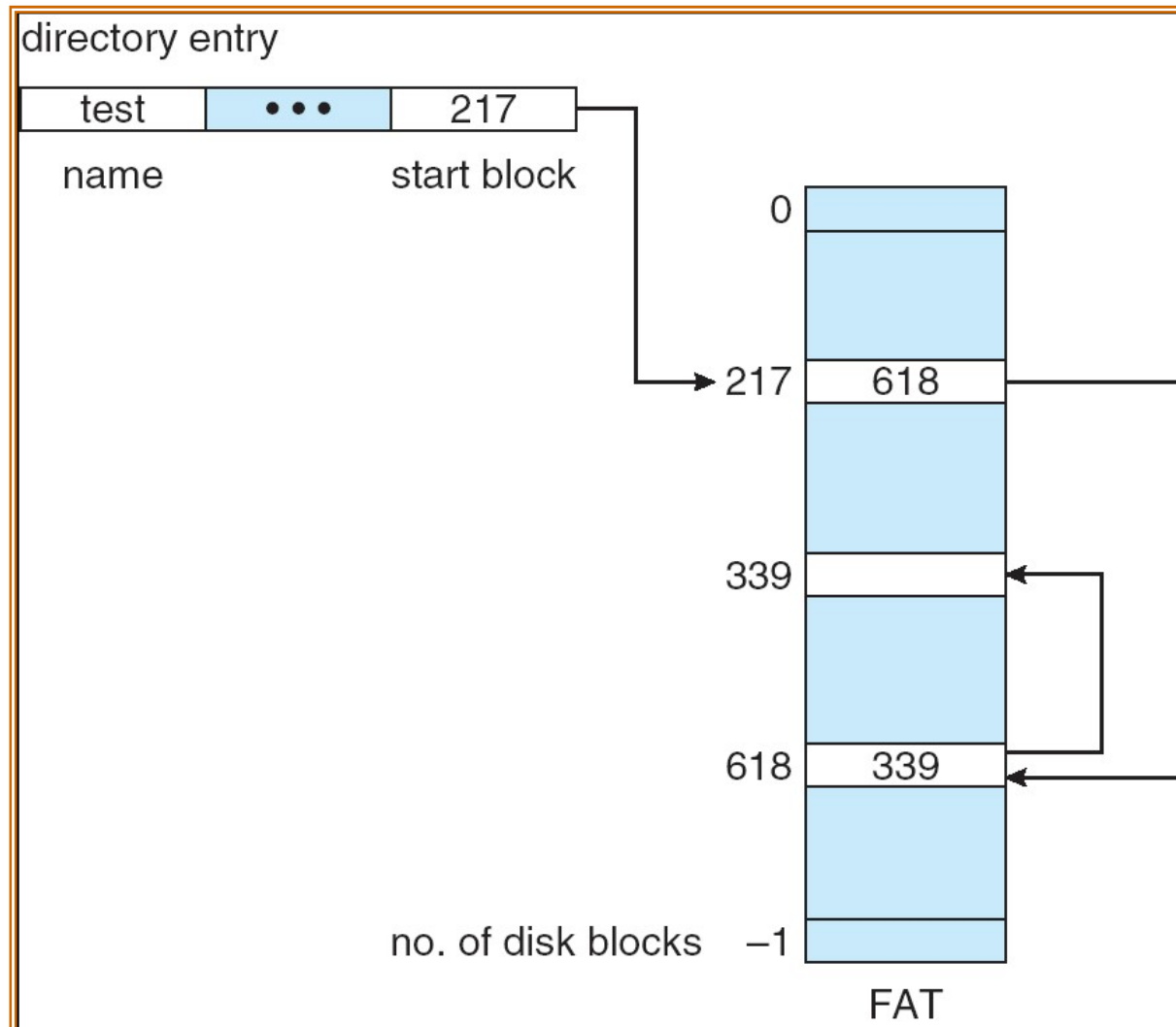
# Tabella di allocazione dei file (FAT)



Tale tabella è conservata in una sezione ben precisa del disco

- è come un array che contiene tanti elementi quanti sono i blocchi;
- in  $FAT[i]$  è contenuto il numero del blocco successivo al blocco *i*.

# Tabella di allocazione dei file (FAT)



La directory entry contiene

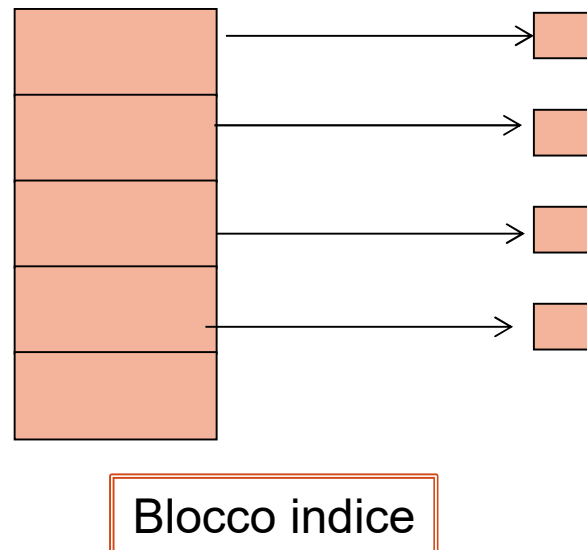
- il numero del primo blocco che sarà usato come indice iniziale della FAT

L'entry della FAT che corrisponde a blocchi inutilizzati contiene 0

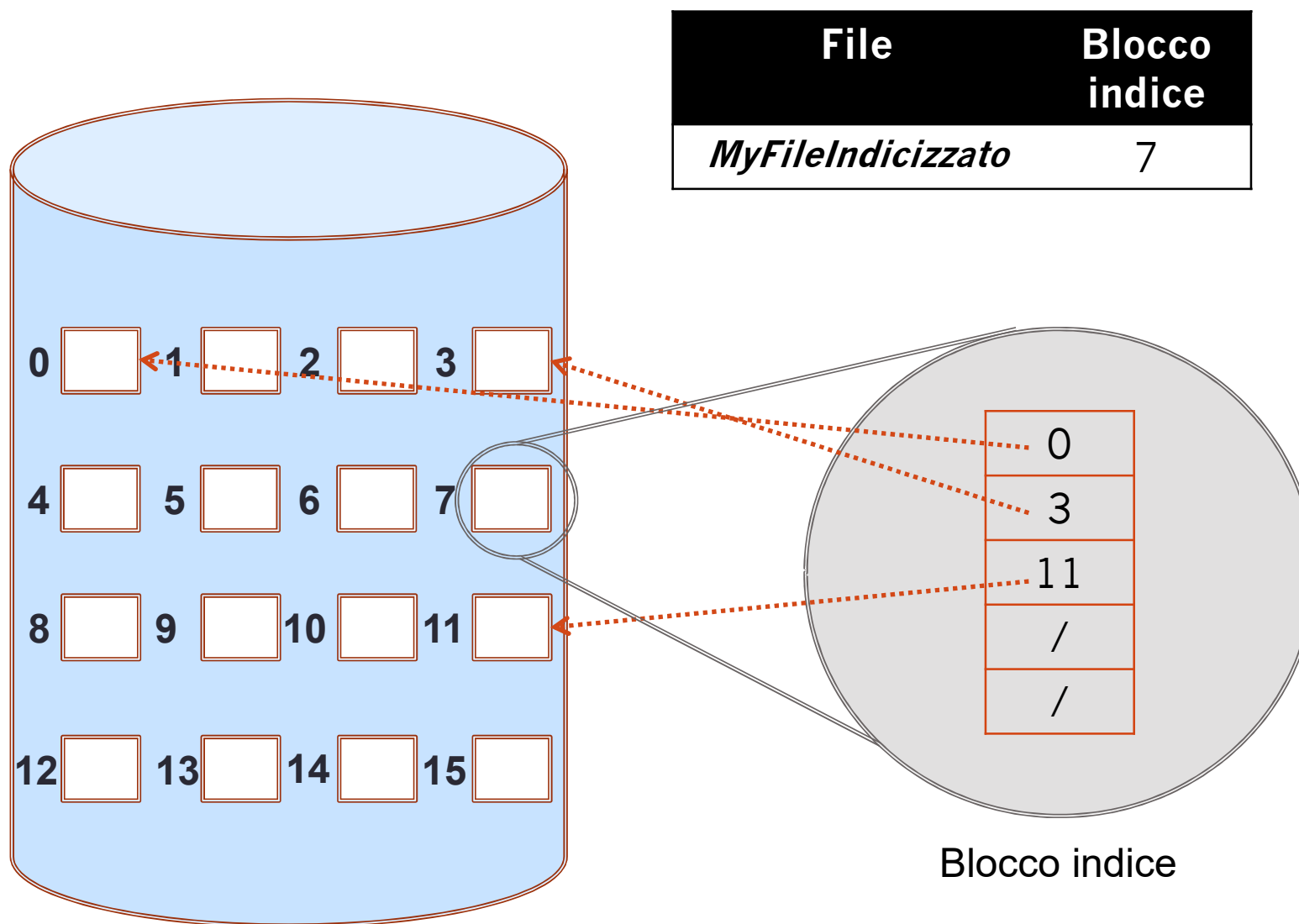
- così se serve un blocco libero basta cercare nella FAT il primo 0

# Allocazione Indicizzata

- Contiene tutti puntatori nel blocco indice.
- Vista logica.
- Ogni file ha il proprio blocco indice, cioè un array contenente gli indirizzi dei blocchi di cui il file è costituito



# Esempio di allocazione indicizzata



# Allocazione indicizzata

- Necessita di una tabella indice.
  - Creazione di un file: si alloca il blocco indice e tutto il suo contenuto è a *nil*
  - Scrittura in un file: se c'è necessità di un nuovo blocco, diciamo l' *i*-mo, si alloca un nuovo blocco e si mette il suo indirizzo nell' *i*-ma posizione del blocco indice.
- Mapping da logico a fisico. Blocco indice della dimensione di 512 parole.

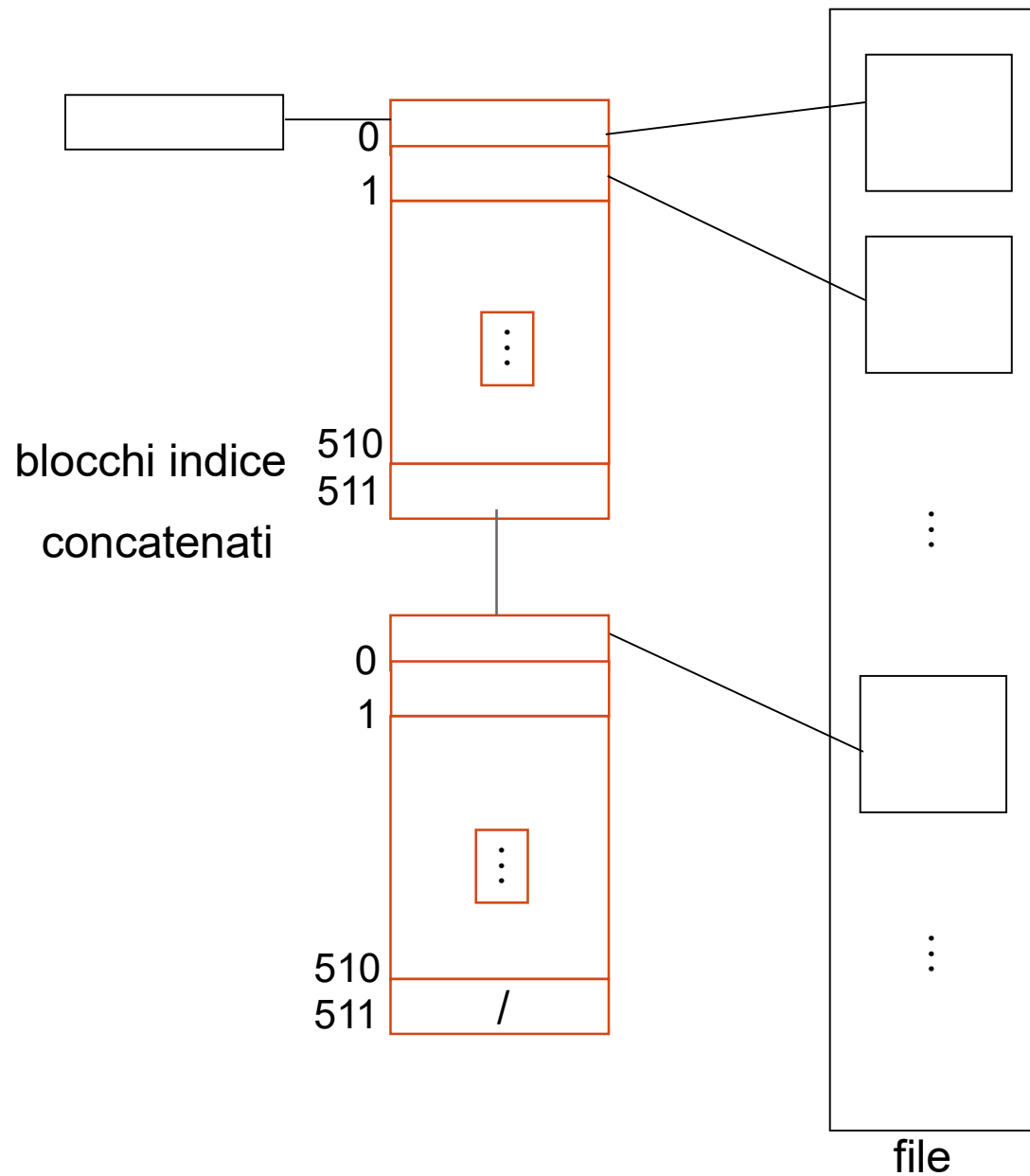


Q = indice nella tabella indice,  
R = indice nel blocco.

# Allocazione indicizzata

- Accesso diretto senza frammentazione esterna, ma
  - c'è l'overhead del blocco indice se il file è piccolo;
  - può essere troppo piccolo se il file è grande

# Allocazione indicizzata – schema concatenato





# Allocazione indicizzata – schema concatenato

- Per permettere la presenza di file lunghi vengono collegati tra loro parecchi blocchi indice; ciò è fatto ponendo come ultima parola di un blocco indice il puntatore al prossimo blocco indice.

- Mappatura da logico a fisico

$$\text{LA}/(511 \times 512) \begin{cases} Q_1 \\ R_1 \end{cases}$$

$Q_1 + 1 =$  blocco della tabella indice,

$R_1$  è usato come segue:

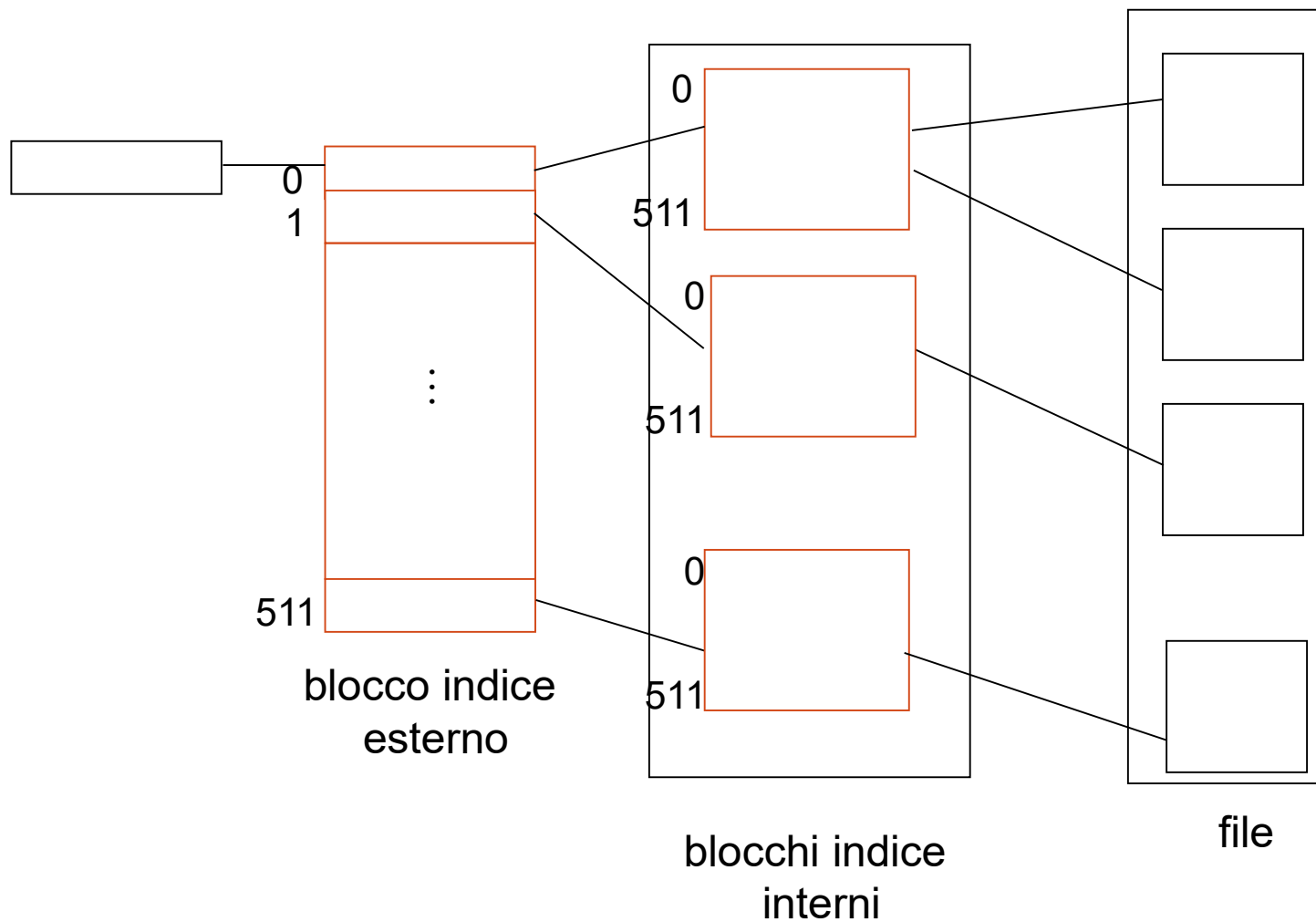
$$R_1 / 512 \begin{cases} Q_2 \\ R_2 \end{cases}$$

$Q_2 =$  indice nel blocco della tabella indice.

$R_2 =$  indice nel blocco di file.

**N.B.** Ogni blocco indice contiene 511 puntatori a blocchi di size 512 => la lunghezza massima di un file gestita da 1 solo blocco indice è 511x512

# Allocazione indicizzata 2 livelli



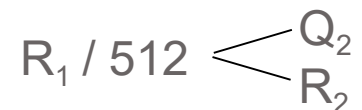
# Allocazione indicizzata

## 2 livelli

- Mappatura da logico a fisico
- Indice a due livelli (la dimensione massima del file è  $512^3$ ).



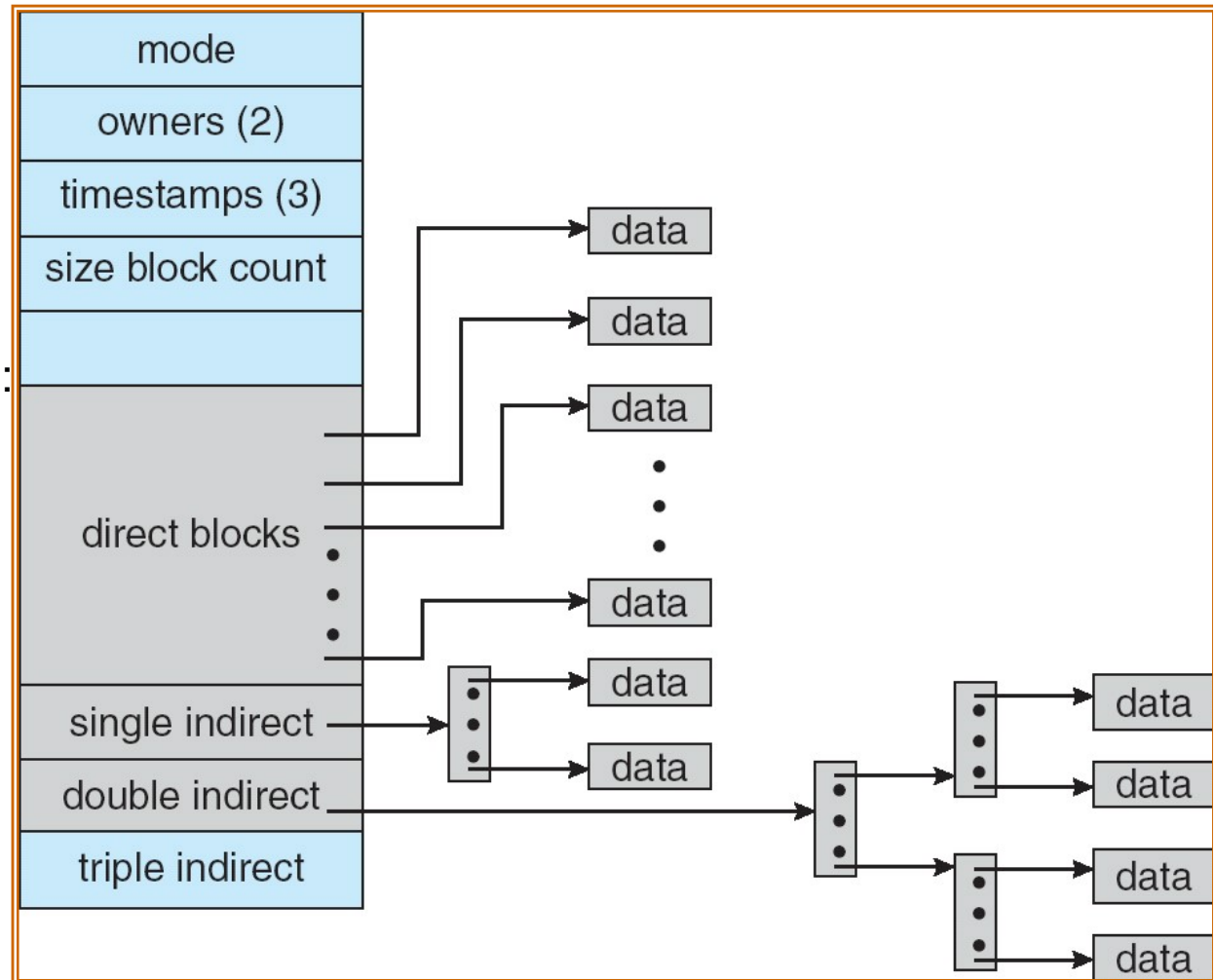
$Q_1$  = indice nel blocco indice esterno,  
 $R_1$  è usato come segue:



$Q_2$  = indice nel blocco indice interno.  
 $R_2$  = indice nel blocco di file.

# Schema combinato: UNIX (4K byte per blocco)

L' i-node (FCB) contiene  
15 puntatori:  
12 diretti ai blocchi del file  
3 puntatori a blocchi indiretti:  
-1° punta ad un blocco  
indiretto;  
-2° punta ad un blocco  
indiretto doppio;  
-3° punta ad un blocco  
indiretto triplo

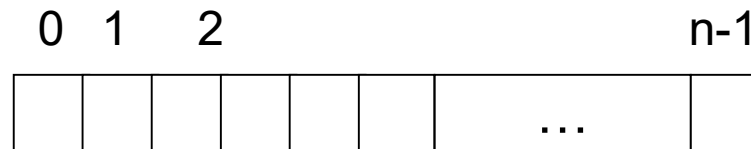


# Gestione dello spazio libero

- È necessario tener memoria dei blocchi di spazio libero per poterli utilizzare quando si ha bisogno di ingrandire un file
- Bisogna ricordare i blocchi rilasciati quando si cancella un file

# Gestione dello spazio libero –vettore di bit

- Vettore dei bit ( $n$  blocchi)



$$\text{bit}[i] = \begin{cases} 0 \Rightarrow \text{blocco}[i] \text{ occupato} \\ 1 \Rightarrow \text{blocco}[i] \text{ libero} \end{cases}$$

Calcolo del primo numero di blocco libero:

Indice del primo bit a 1

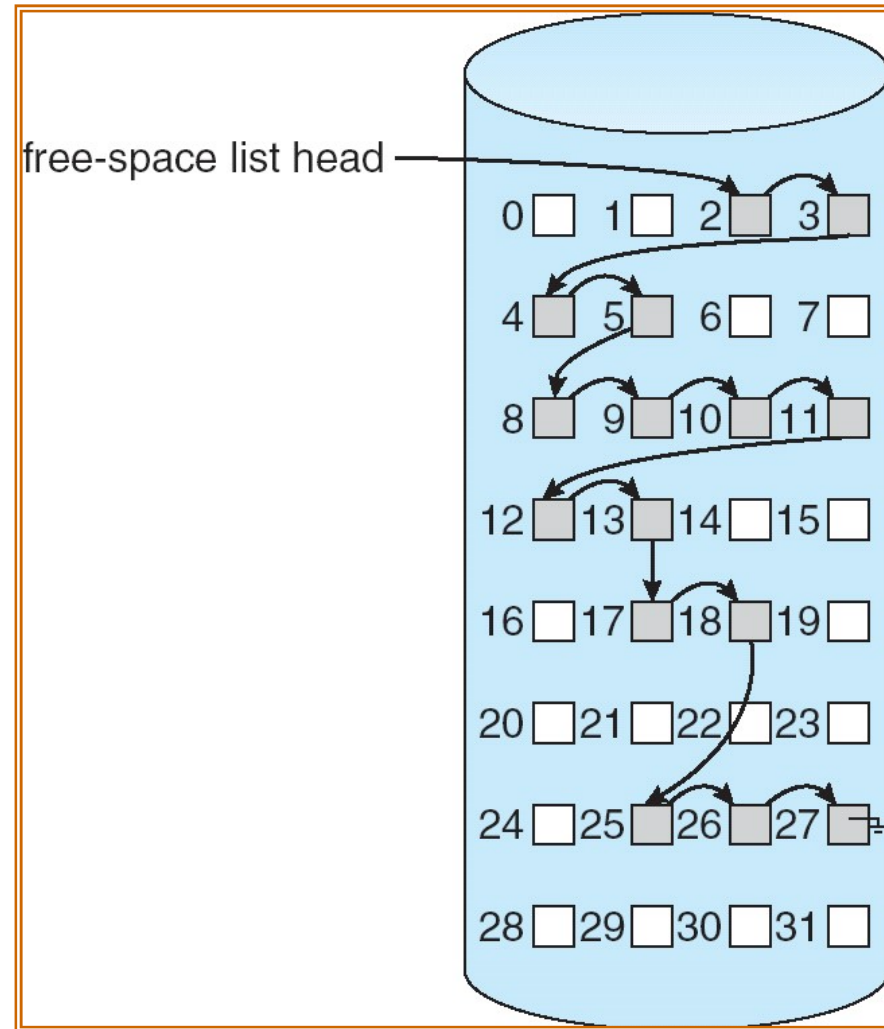
# Gestione dello spazio libero: vettore di bit

- Il vettore di bit richiede spazio extra.
  - Esempio:
    - dimensione blocco =  $2^{12}$  byte
    - dimensione disco =  $2^{30}$  byte (1 gigabyte)
    - $n = 2^{30}/2^{12} = 2^{18}$
    - => size del vettore di bit =  $2^{18}$  bit (o 32Kb)
- Semplicità di trovare blocchi liberi consecutivi sul disco.

# Gestione dello spazio libero: Lista dei blocchi liberi

Si conserva il puntatore al primo blocco in una locazione speciale del disco che viene caricato in memoria quando si vuole accedere ad un blocco libero

- Assenza di spreco di spazio.
- Non facile trovare blocchi contigui.





# Gestione dello spazio libero: altri metodi

## Raggruppamento:

- Si memorizzano in un blocco gli indirizzi di  $n$  blocchi: di questi i primi  $n-1$  sono realmente liberi mentre l'ultimo contiene gli indirizzi di altri  $n$  blocchi, e così via
- Permette di trovare rapidamente molti blocchi liberi.

## Conteggio:

- Spesso più blocchi contigui possono essere rilasciati
- Quindi ogni elemento della lista dello spazio libero è formato da un indirizzo ed un contatore: l'indirizzo punta al primo dei blocchi liberi mentre il contatore dice quanti blocchi contigui ci sono

# Gestione dello spazio libero

## Bisogna fare attenzione a:

- Nella lista dei blocchi liberi: **Proteggere il puntatore alla lista**
- Nel vettore di bit : **Consistenza**
  - deve essere tenuto nel disco;
  - la copia in memoria e su disco possono essere diverse;
  - la situazione in cui un blocco[i] ha  $\text{bit}[i] = 0$  in memoria e  $\text{bit}[i] = 1$  su disco non è accettabile.
  - Soluzione:
    - impostare  $\text{bit}[i] = 0$  nel disco;
    - allocare il blocco[i];
    - impostare  $\text{bit}[i] = 0$  in memoria.