



UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI INFORMATICA

Alberto Ceselli
(alberto.ceselli@unimi.it)

Informatica II

Sistemi Operativi

DIGIP - a.a. 2015/16



Sistemi Operativi

(modulo di Informatica II)

Implementazione del file system

Patrizia Scandurra

Università degli Studi di Bergamo

a.a. 2014-15

Sommario

- Realizzazione del file system
- Realizzazione della directory
- Metodi di allocazione dei file
- Gestione dello spazio libero

Realizzazione del File System



- Come vengono memorizzati i file e le directory
- Come gestire lo spazio su disco
- Quali strutture dati e operazioni sono necessarie al sistema operativo per una gestione efficiente e affidabile del file system

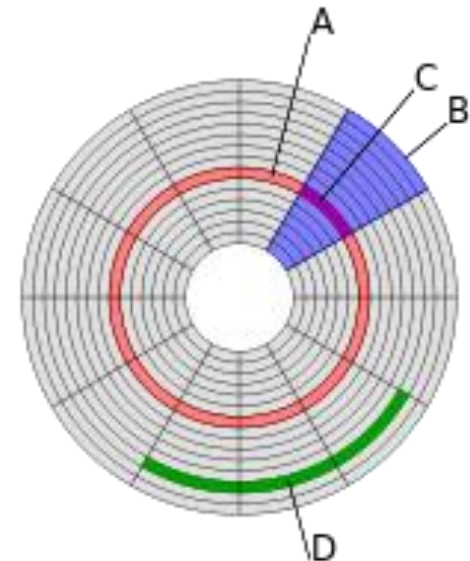
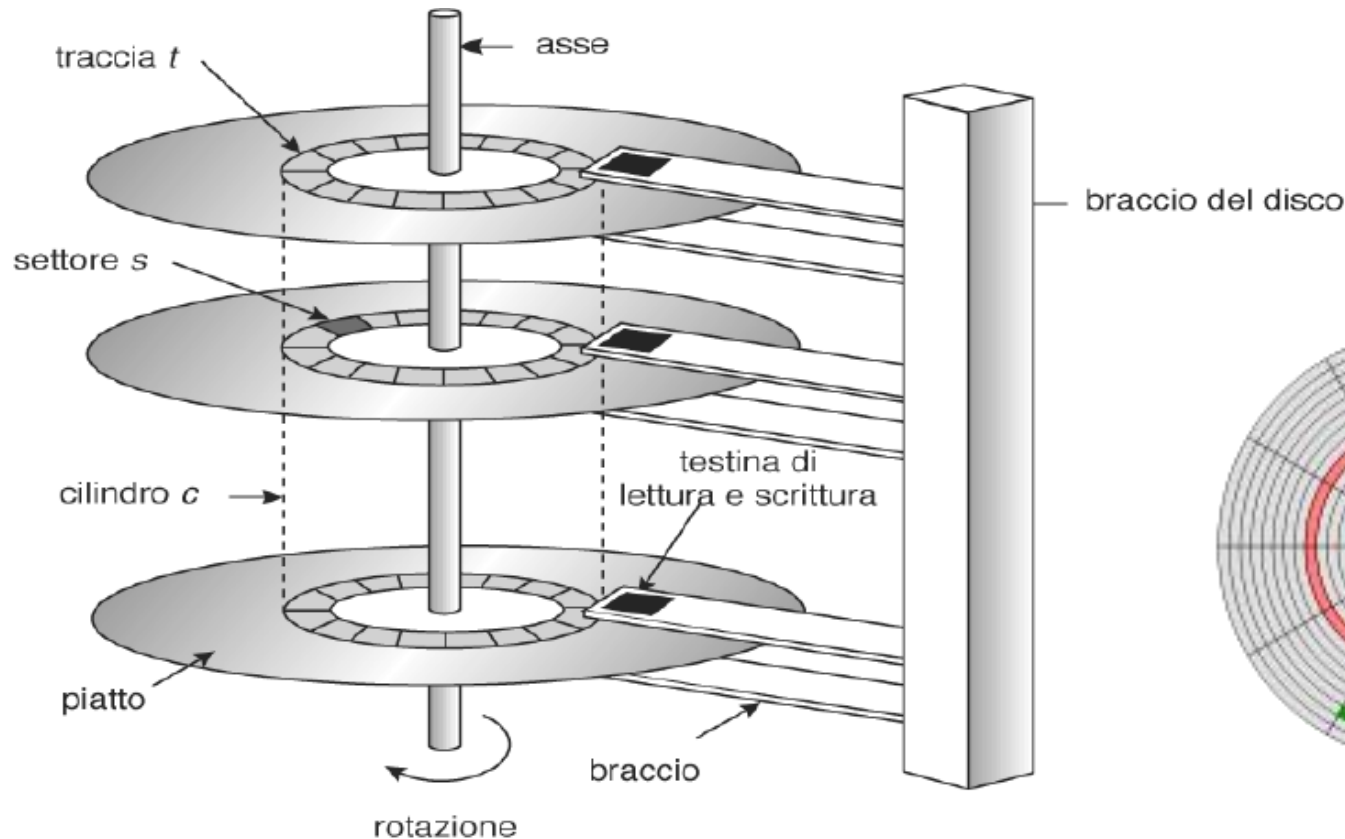
Memorie di massa

- Il file system risiede permanentemente in memoria di massa
- Tipicamente questa memoria è fornita dai dischi, i quali vengono usati per due principali motivi:
 - Possono essere **riscritti** localmente (leggo e riscrivo lo stesso blocco)
 - Permettono l'**accesso**, sia sequenziale che diretto, **ai blocchi fisici** che memorizzano i diversi file
- La dimensione dei blocchi varia da dispositivo in dispositivo
 - Da 32 a 4096 Byte
 - Solitamente 512 Byte

Il disco rigido o disco fisso (hard disk drive)

Un disco è costituito da più piatti:

- Ogni piatto ha una testina che permette la lettura e la scrittura
- Il disco gira, quindi leggere i settori della stessa traccia non comporta lo spostamento della testina



Struttura del file system

- **File:**
 - Unità logica di memorizzazione
 - **Blocco di controllo di file (File Control Block - FCB)** struttura di memorizzazione che contiene i *metadati* di un file (**descrittore del file**)

permessi sul file
date del file (creazione, accesso, scrittura)
proprietario, gruppo, ACL del file
dimensione del file
blocchi di dati del file

NO contenuto, ma info sulla **locazione** del file nel dispositivo di massa

- La struttura del **file system** risiede in modo permanente in un'unità di memorizzazione secondaria (ad es. il disco)
- La struttura di un file system è organizzata in “livelli”

Struttura a strati del file system

- Il livello superiore si basa sui servizi offerti dal livello inferiore

programmi applicativi



file system logico



modulo di organizzazione dei file



file system di base



controllo I/O



dispositivi

File System logico: gestisce i metadati e la struttura delle directory

- Gli attributi dei file vengono salvati in **File Control Blocks (FCB)**

Modulo di organizzazione dei file: traduce un indirizzo di un blocco logico in un indirizzo di un blocco fisico; Comprende anche il gestore dello spazio libero (mantiene la lista dei blocchi liberi)

File System di base: invia dei comandi di base al controllo dell'I/O per scrivere e leggere blocchi fisici

- E.g. Leggi il blocco nell'unità 1, cilindro 12, superficie 4, settore 2

Controllo dell'I/O: driver dei dispositivi e gestore degli interrupt

- Si occupa del trasferimento dell'informazione dalla memoria di massa a quella centrale

Alcuni tipi di file system

- **FAT12, FAT16, FAT32** – IBM OS/2, MS-DOS, prime versioni di Windows fino a Windows ME, per floppy disk e memorie flash e drive USB da vari SO
- **VFAT (Virtual FAT)**– Windows95, per supportare nomi lunghi (fino a 255 caratteri UTF-16) che nomi brevi (8+3 caratteri) in modo trasparente
- **exFAT (FAT64)** – da Windows Vista per memorie flash
- **NTFS (New Technology File System)** – Windows (da Windows NT in poi)
- **UFS (Unix file system)**
- **Ext (Extended file system), ReiserFS, Reiser4, UMDOS (FAT)** - per GNU/Linux
- **High Sierra** – ISO 9660 CD-ROM
- **UDF (Universal Disk Format)** - CD-RW e DVD

Partizioni e montaggio

- Tipicamente un disco può contenere diverse partizioni
 - ognuna delle quali può avere un file system diverso
- Un partizione senza file system è detta **raw partition**
 - Un esempio è la partizione usata per lo *swap* nei sistemi Unix
- Un'altra partizione priva di file system, ma con un formato proprio, è la **partizione di boot**
 - Poiché all'avvio i driver del file system non sono ancora stati caricati
 - Consiste in un insieme di blocchi che vengono caricati in memoria centrale e contengono le **istruzioni di avvio del SO**
 - Un'immagine di boot può lanciare più SO
 - Una volta scelto il SO la partizione radice **viene montata**
 - Essa contiene il kernel del SO ed eventuali file di sistema
- Altre partizioni possono essere montate su richiesta
 - Il SO inserisce le informazioni della partizioni montate nella **tabella di montaggio**

Realizzazione del file system

Un file system è **l'insieme di strutture dati** per:

- la memorizzazione,
- l'organizzazione,
- la navigazione e l'accesso in lettura- scrittura

dei dati

Diverse **strutture dati**

- **su dispositivo** (su disco)
- **in memoria centrale**
 - `open()` e `close()` di file segnano il passaggio da dispositivo a memoria e viceversa

Strutture del file system su disco

Per ogni partizione (volume):

- **Boot Control Block** contiene informazioni necessarie per avviare (bootstrap) il SO contenuto nel disco (vuoto se SO non presente)
- **Volume control block:** contiene dettagli riguardanti il volume; detto anche *Super block* in UFS o *master file table* in NTFS
 - # e dimensione dei blocchi fisici
 - # e puntatori ai blocchi liberi,
 - # e puntatori ai FCB
- **Directory (principale)**
- **Blocchi di controllo dei file (FCB)**



Strutture del file system in memoria

- **La tabella delle partizioni (partition table) o di montaggio** contiene info su ciascun volume montato
- **Struttura delle directory** contiene le informazioni relative alle directory attualmente in uso dai processi
 - contiene un puntatore alla tabella delle partizioni per le partizioni montate
- **Tabella generale dei file aperti**
 - Una copia del descrittore di file per ciascun file aperto
 - Un contatore dei processi che hanno correntemente aperto quel file
- **Tabella dei file aperti per processo**
 - Contiene puntatori agli elementi della tabella generale dei file aperti
 - Altri campi: puntatore alla posizione corrente nel file, tipo di accesso richiesto al momento della open, ecc..

Uso delle strutture del file system (1)

Esempio 1: l'invocazione di una chiamata di sistema per la **creazione di un file** comporta l'uso di queste strutture dati

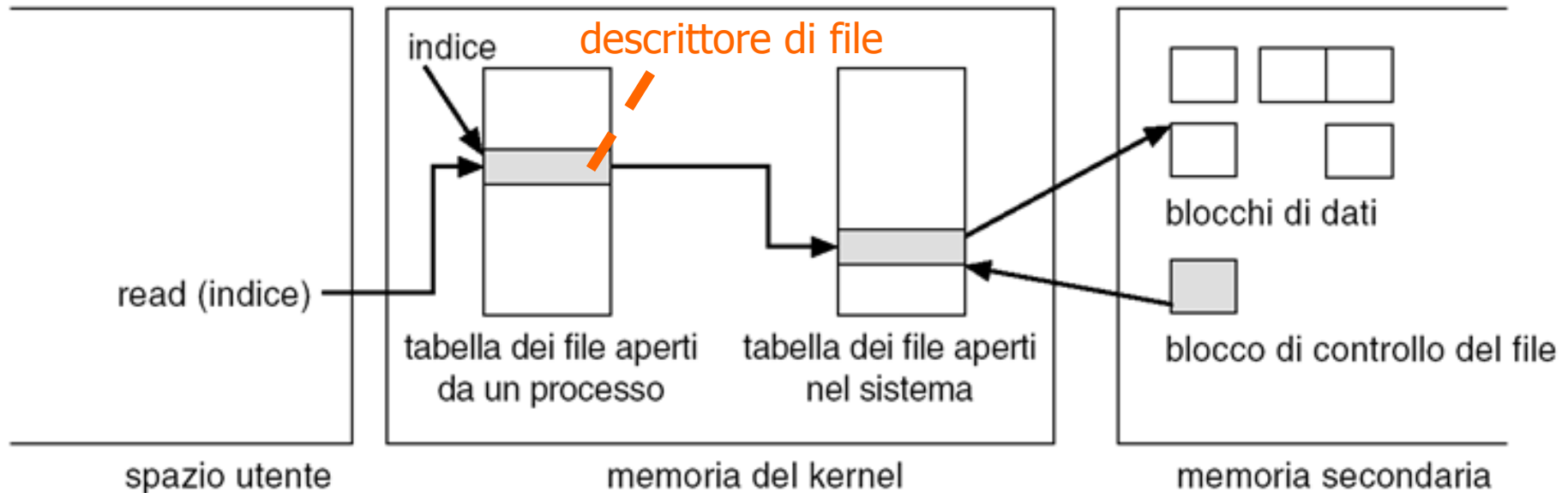
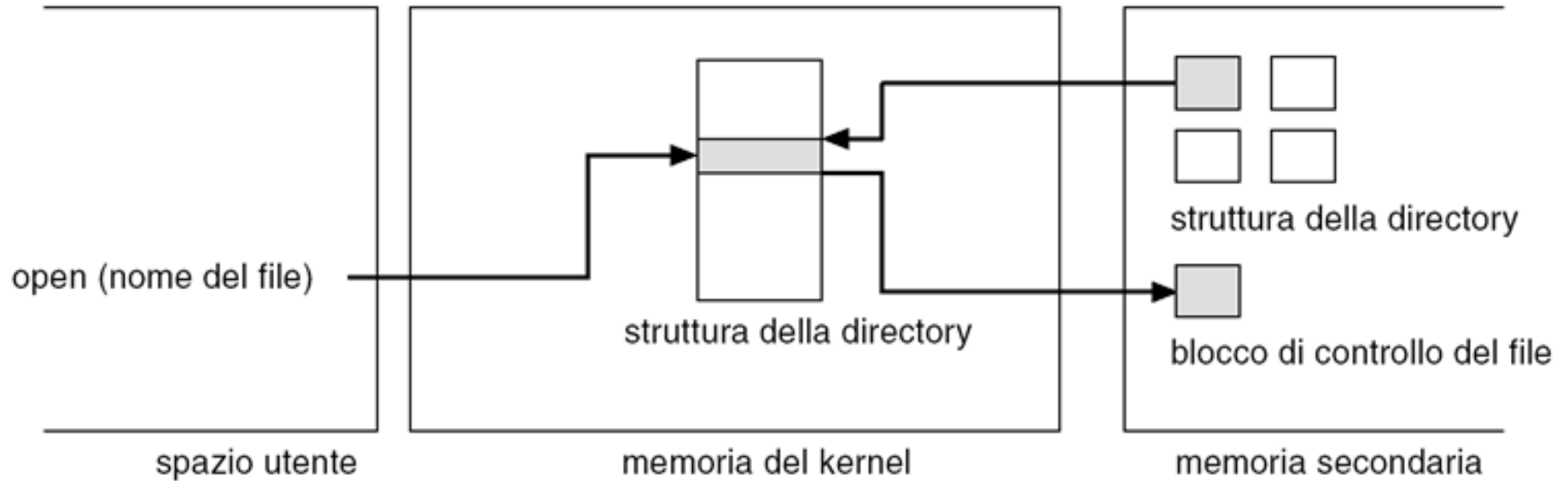
1. Il SO crea e alloca un nuovo FCB
2. Carica in memoria centrale la directory che lo dovrà contenerlo
3. Aggiorna la directory
4. Riscrive la directory in memoria di massa

Uso delle strutture del file system (2)

Esempio 2: l'invocazione di una chiamata di sistema per l'apertura e la chiusura (`open()` e `close()`) di un file:

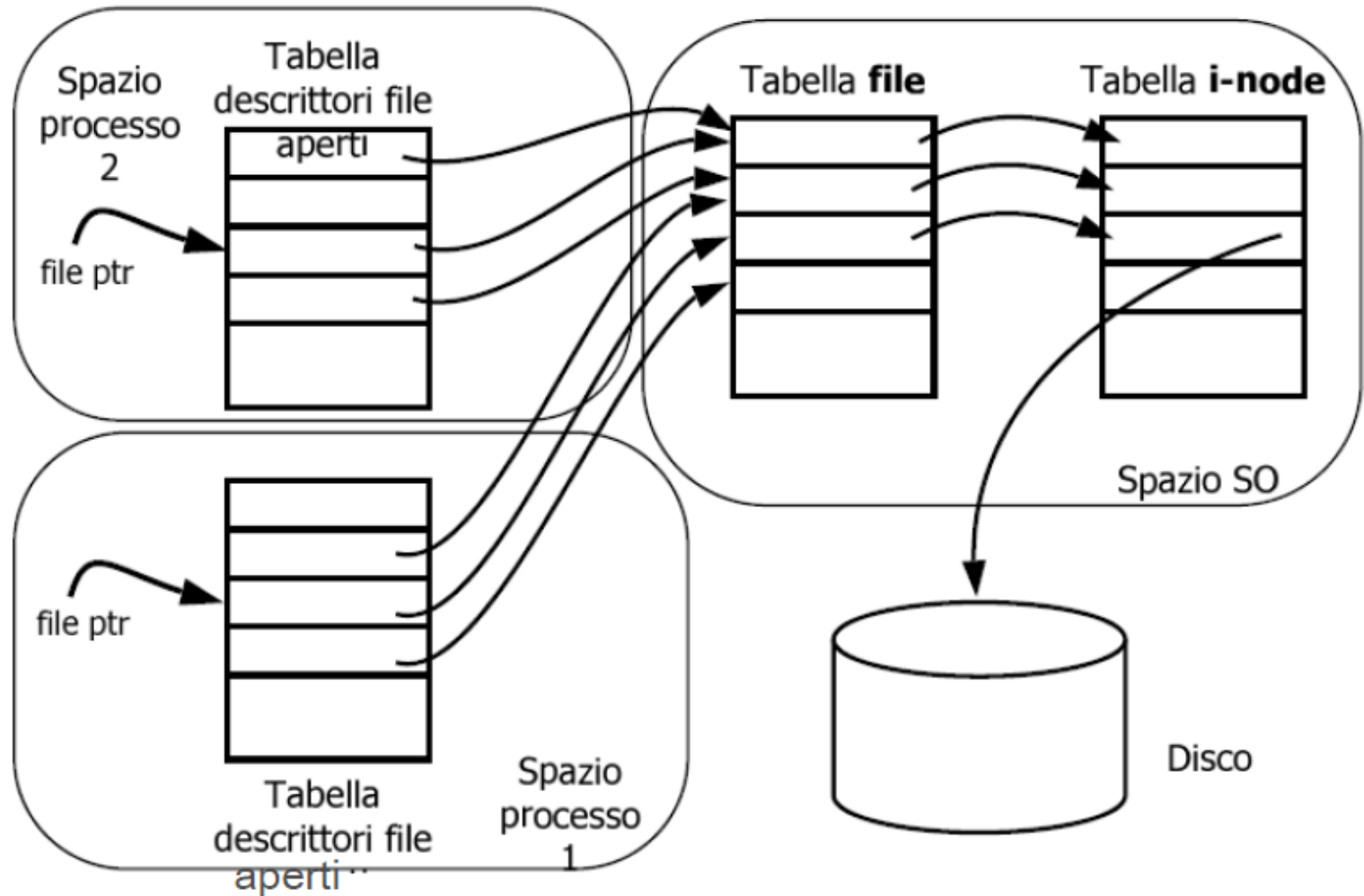
- All'apertura del file, il SO controlla la tabella generale dei file aperti per vedere se il file è già aperto
 - In caso negativo: tramite la struttura della directory, aggiunge il FCB alla tabella generale del SO
 - In ogni caso:
 - Aggiunge alla tabella del processo chiamante un nuovo elemento che punta alla tabella generale del SO
 - Incrementa il contatore delle aperture del file
- Alla chiusura del file il SO decrementa il contatore e rimuove l'elemento dalla tabella del processo chiamante
 - Se il contatore di aperture è 0 il FCB viene rimosso anche dalla tabella generale del SO

Uso delle strutture del file system (3)



Strutture dati del file system in memoria- tabelle dei file aperti

In UFS:

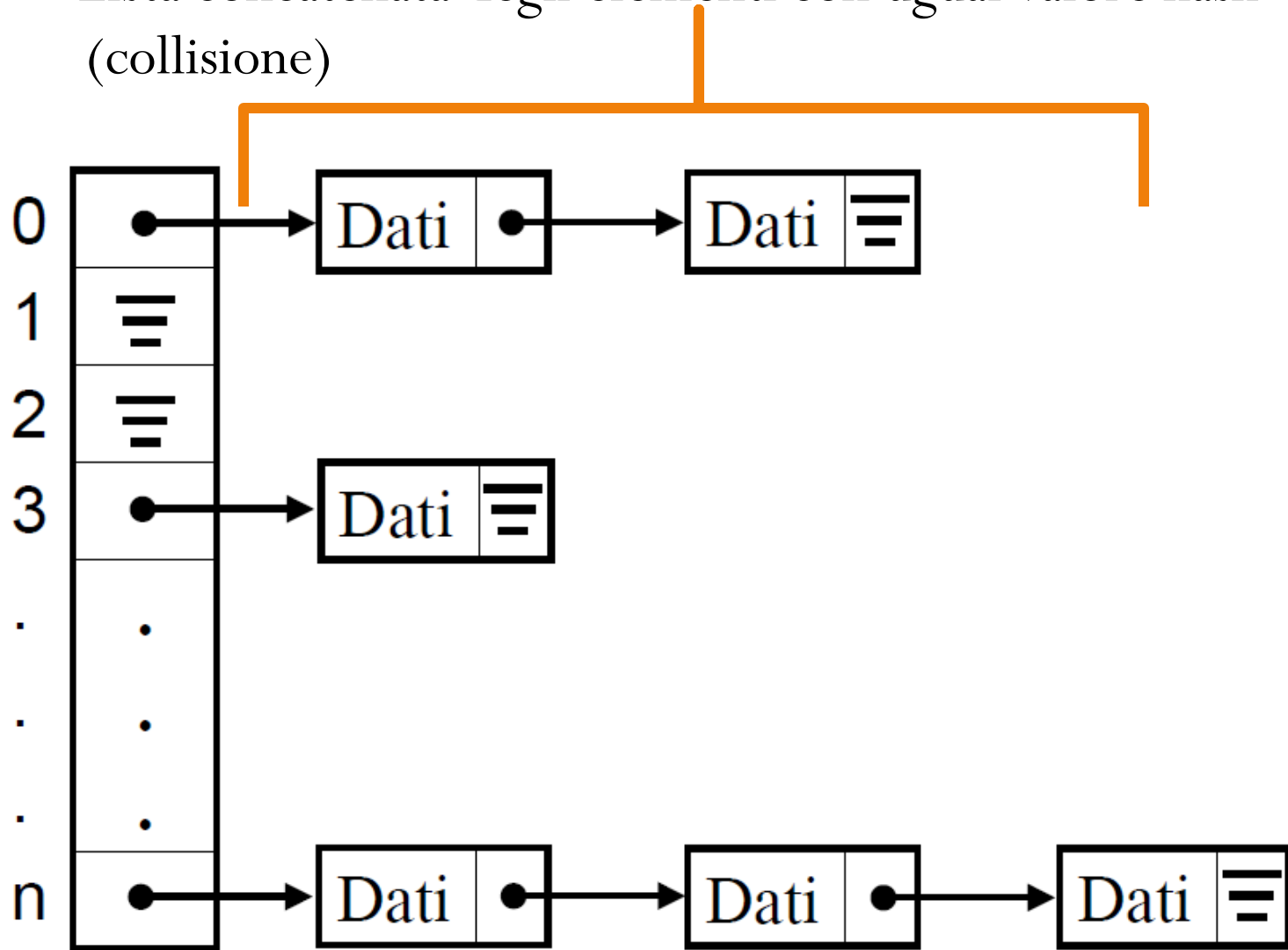


Realizzazione della struttura delle directory

- **Lista** – lista di nomi di file con puntatori ai FCB
 - Semplice da programmare, ma lenta per trovare un file -- *ricerca lineare* $O(n)$
- **Tabella hash** – un vettore di dimensione fissa per memorizzare gli elementi della directory ed una tabella indice “di hash”
 - Ogni elemento ha un *valore hash* $h(\text{key})$ calcolato sulla chiave key, ed usato come posizione nel vettore
 - Diminuisce il tempo di ricerca nella directory -- $O(1)$ in *media*, ma presenta il **problema delle collisioni** – situazioni in cui due file sono identificate dalla stessa posizione generata dalla funzione di hash
 - Es. soluzione: lista concatenata per ogni elemento (rallenta però le ricerche) – *chained-overflow hash table*

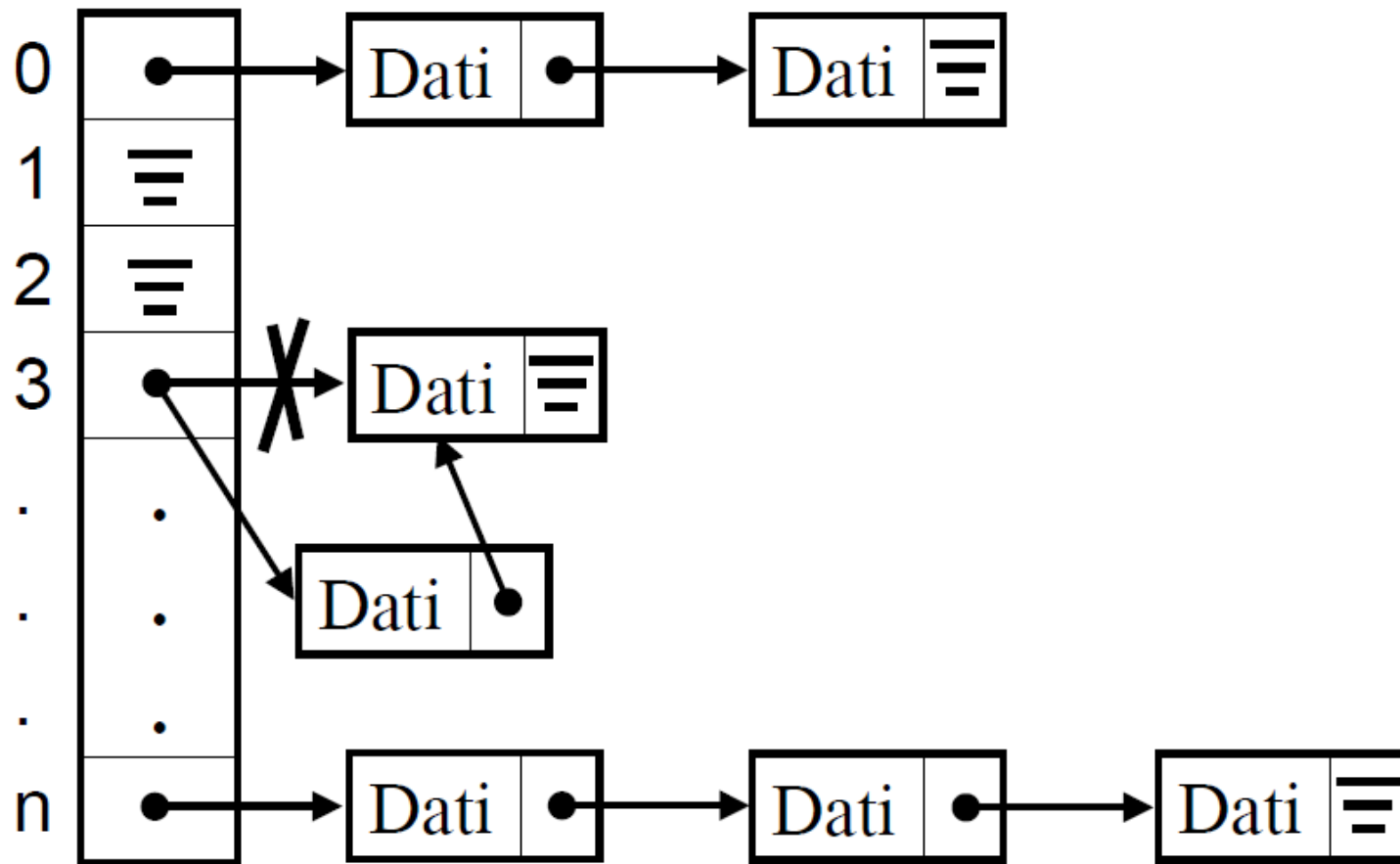
Schema di tabella Hash - *chained-overflow hash table*

Lista concatenata degli elementi con ugual valore hash
(collisione)



Schema di tabella Hash - *chained-overflow* hash table

Inserimento di un elemento in testa alla lista di posizione $h(\text{key})$



Metodi di allocazione dei file

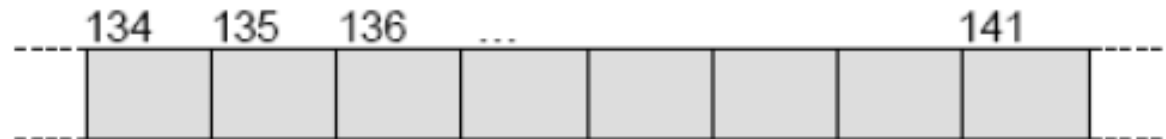
Un metodo di allocazione indica come i file devono essere allocati ai blocchi del disco:

- **Allocazione contigua**
- **Allocazione collegata**
- **Allocazione indicizzata**

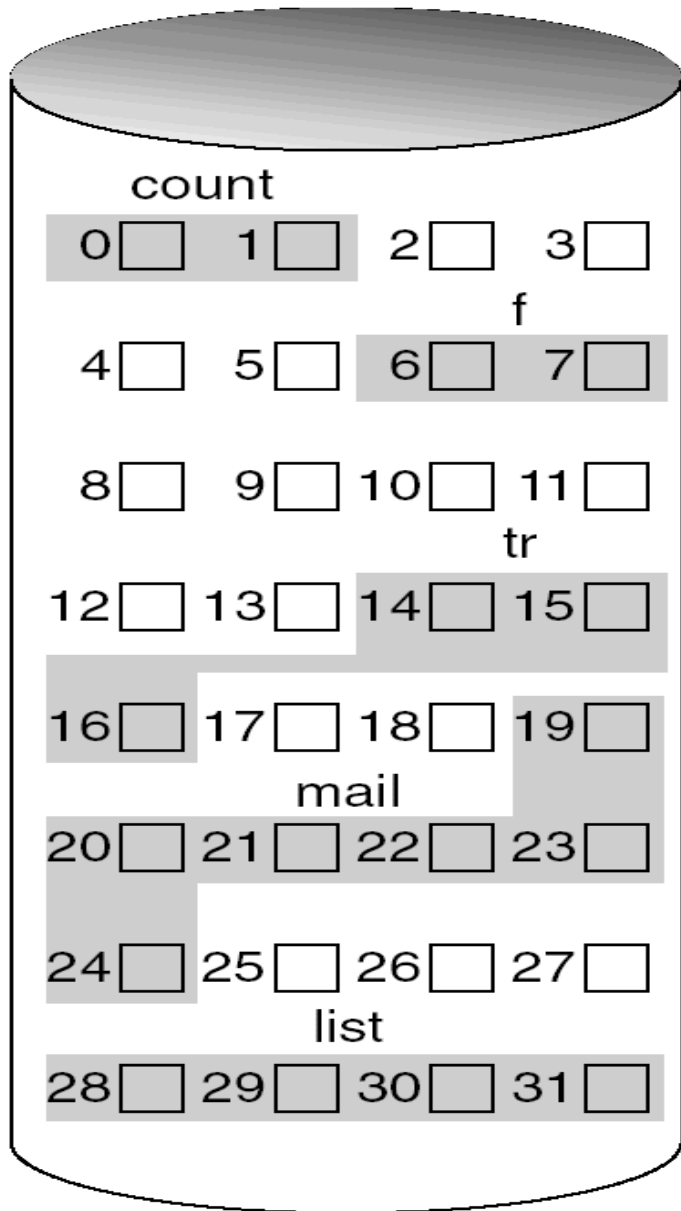
Allocazione contigua

- Ogni file occupa un certo numero di blocchi **contigui** su disco
 - Un file di n blocchi è memorizzato nei blocchi adiacenti $b, b+1, b+2, \dots, b+n$
- Facile – l'indirizzo del disco è definito da una coppia (contenuta nel descrittore del file)
(b, n) - # blocco iniziale b e lunghezza (# blocchi)

data (134, 8)



Allocazione contigua dello spazio del disco



directory

file	inizio	lunghezza
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Ordinamento sequenziale
adiacente dei blocchi

Allocazione contigua (2)

- Accesso rapido al blocco successivo
- Accesso diretto lento
- Problema di individuazione di spazio contiguo per l'allocazione
 - di un nuovo file o
 - di una estensione di file esistente
- Frammentazione esterna
 - Compattazione (*de-frammentazione*)
- Estensione dei file
 - frammentazione interna

Allocazione contigua ⁽³⁾

1. Individuazione dello spazio per un nuovo file

problema già visto.... “*problema di allocazione dinamica della memoria*”

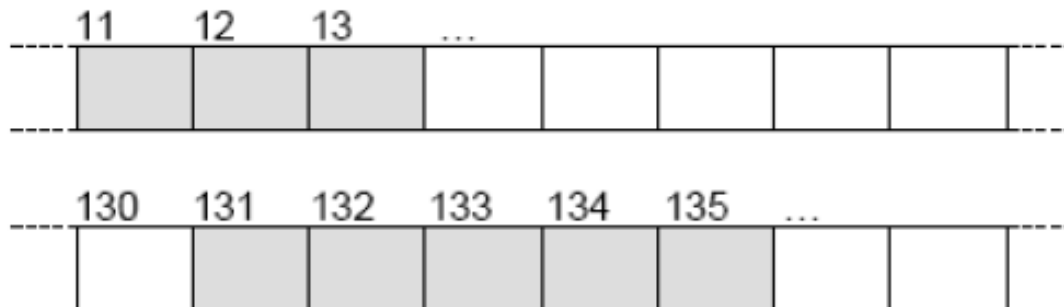
- **First-fit, best-fit, worst-fit**
- Spreco di spazio: **frammentazione esterna!!!!**

Allocazione contigua (4)

2. Individuazione dello spazio per ampliare un file

- Molti nuovi file system (ad es. Veritas o JFS) usano uno **schema di allocazione contigua “modificato”**
 - **Inizialmente viene allocato un pezzo contiguo di spazio e poi**, quando la quantità non è sufficientemente grande, **un estensione** come ulteriore spazio contiguo
 - Un file consiste in una o più estensioni
 - **(b,e,n1,n2)** – primo blocco, primo blocco dell'estensione, lunghezza principale, lunghezza dell'estensione

data (11, 131, 3, 5)

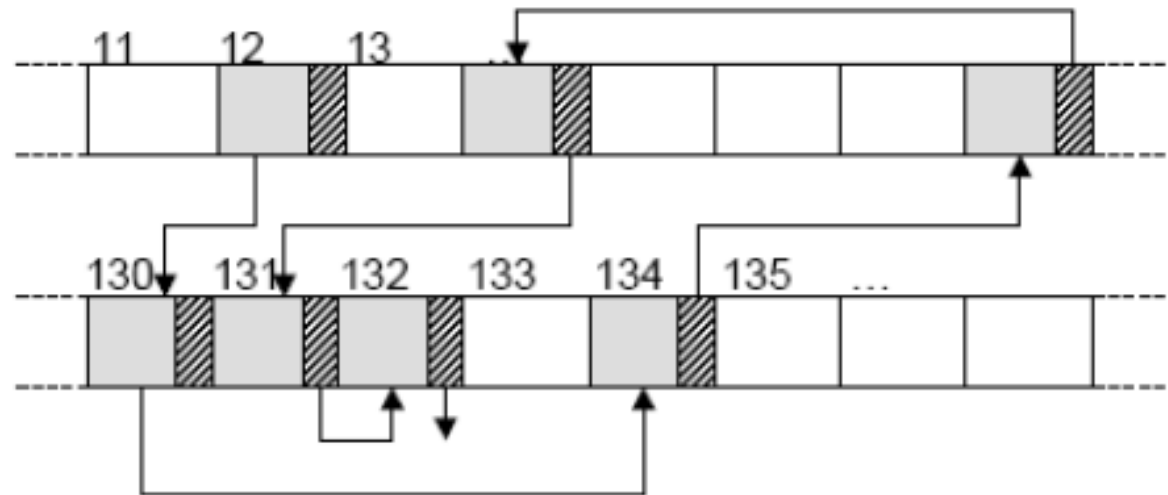
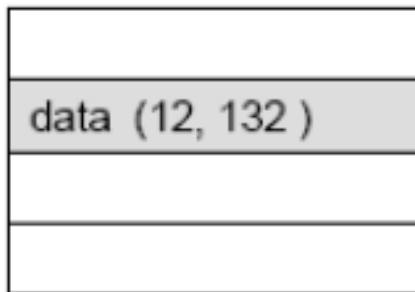


Allocazione collegata

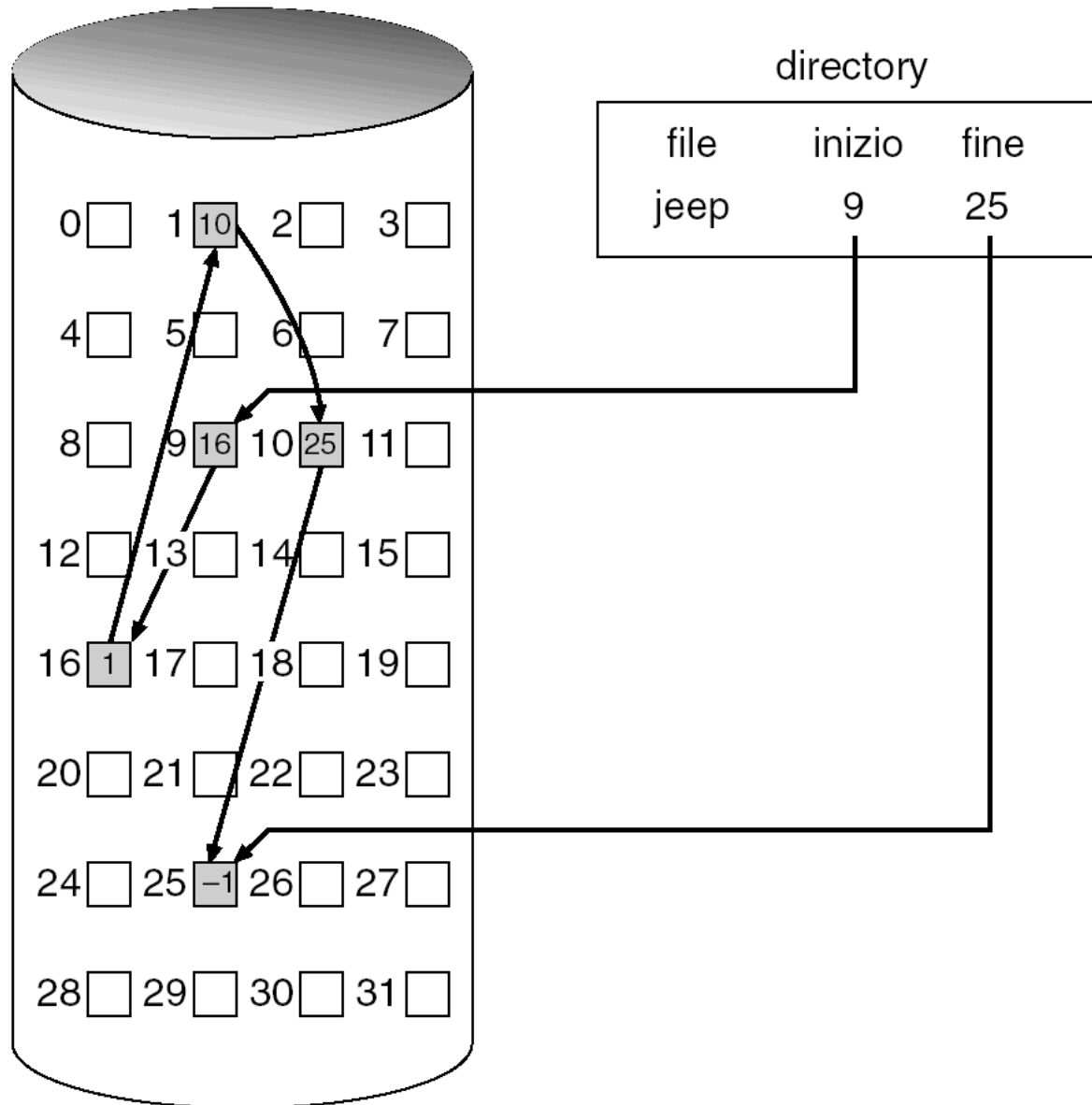
- Ogni file è **una lista collegata di blocchi del disco**: i blocchi possono essere **sparpagliati ovunque** nel disco
- Struttura di un blocco



- L'FCB contiene il puntatore al primo e all'ultimo blocco (b_1, b_m)



Allocazione collegata (2)



Allocazione collegata (3)

- **Libera gestione dello spazio** – no frammentazione esterna, no preallocazione
- **Facile** – richiede solo l'indirizzo del disco
 - Per scrivere, si cerca un blocco libero (sistema di gestione dei blocchi liberi), e si concatena questo blocco al termine della lista
 - Per leggere, si cerca il blocco seguendo la lista
- **Accesso diretto inefficiente:** per trovare l' i -esimo blocco, bisogna scandire la lista
 - i/N_b accessi prima di localizzare il blocco desiderato
 - N_b = numero di blocchi logici
- **Spreco di spazio per i puntatori** al blocco successivo
 - Possibile soluzione: uso dei “**cluster**”, ma di contro aumenta la frammentazione interna
- **Problemi di affidabilità** in caso di guasto in un blocco fisico – perdita del puntatore al successivo blocco

Allocazione concatenata con FAT

descrittore della directory



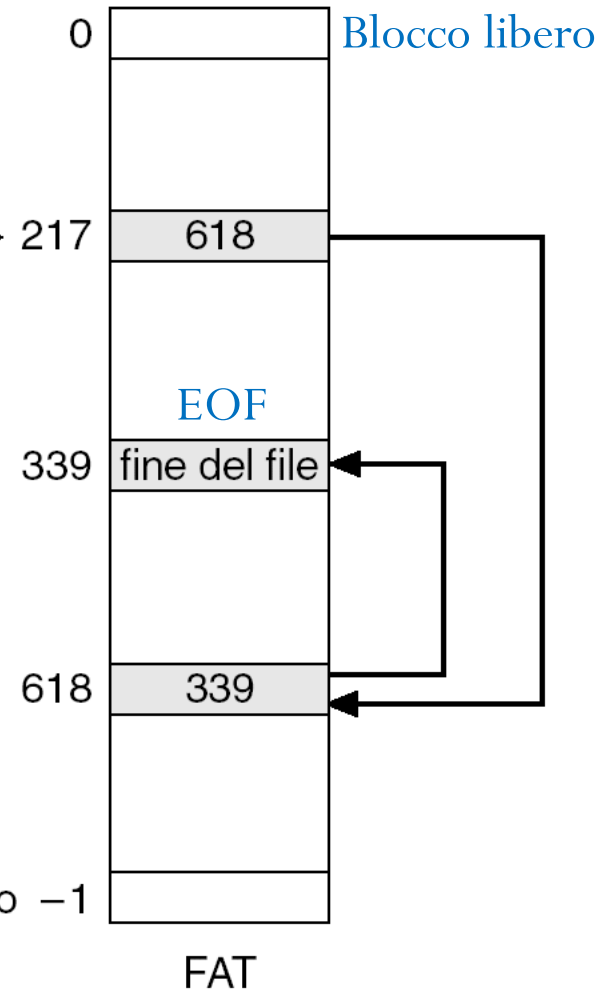
nome

blocco di partenza

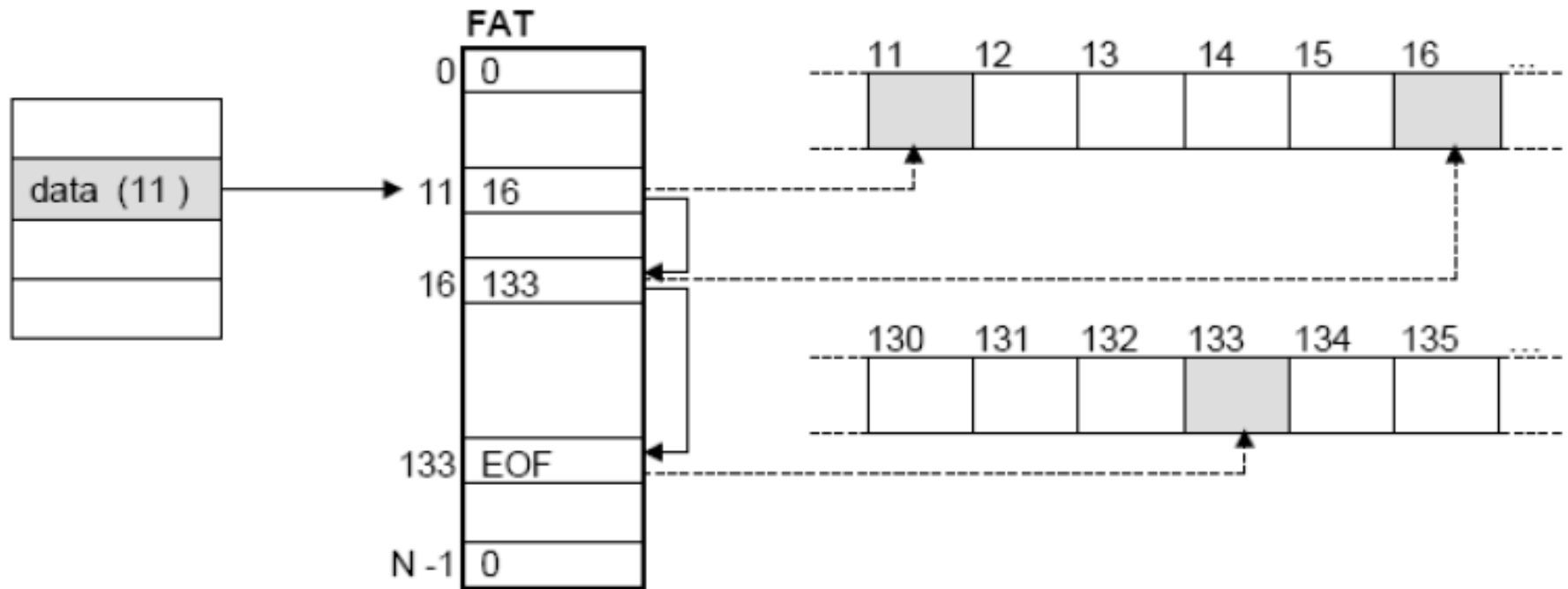
La FAT ha un elemento per ogni blocco del disco ed è **indicizzata dal numero di blocco**

- La si usa come lista concatenata
- La si usa anche per la gestione dei blocchi liberi (marcati in maniera speciale e/o con contenuto nullo)

n. dei blocchi del disco - 1



Esempio di allocazione FAT



Per **localizzare** un blocco, cosa si fa?

FAT: Tabella di allocazione dei file

(Cont.)

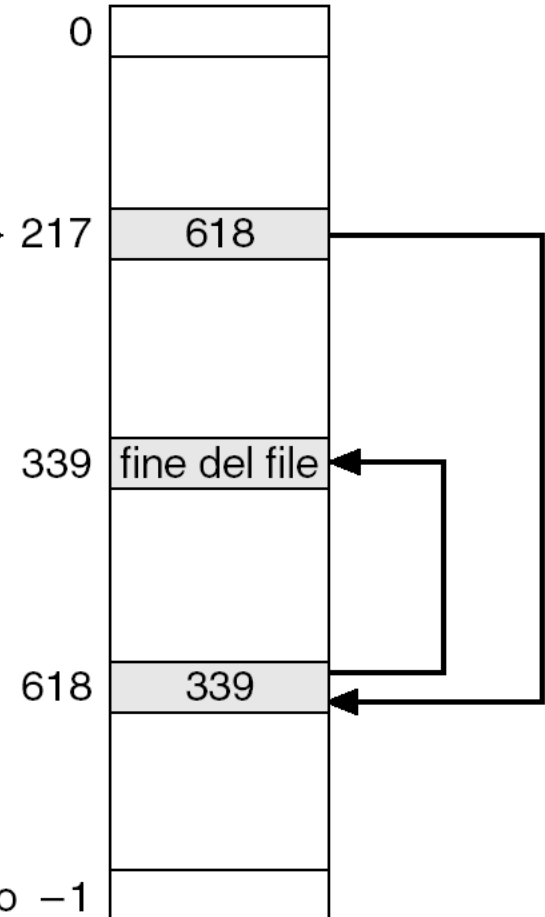
descrittore della directory



nome

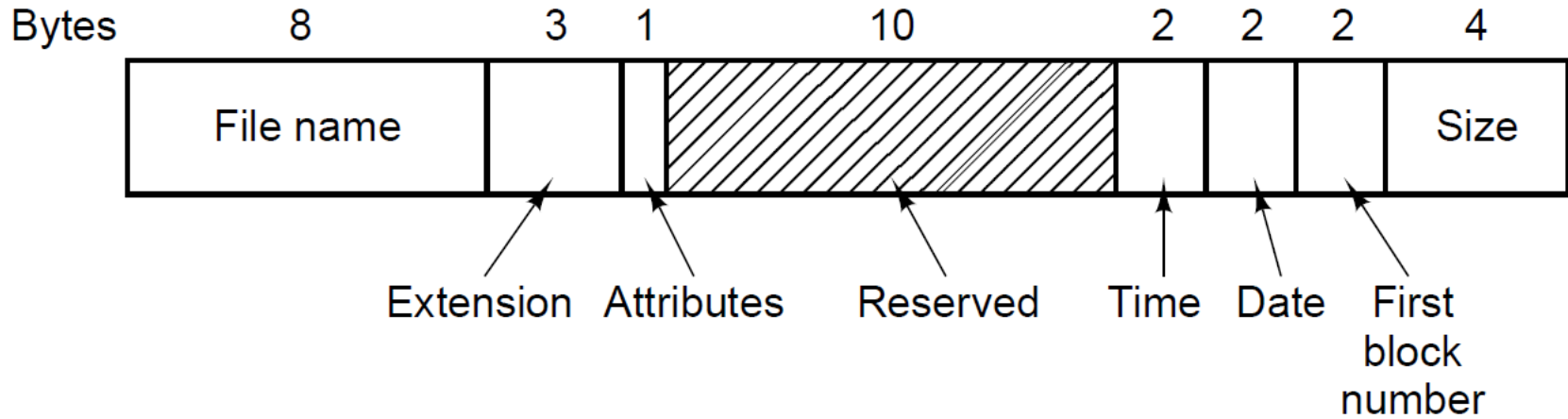
blocco di partenza

Tempo di accesso al file mediamente maggiore; ma migliora l'accesso diretto e non si ha il problema della fragilità dei puntatori



FAT

FAT: Tabella di allocazione dei file: la voce di directory dell'MS-DOS



- Lunghezza del nome fissa
- Attributi: read-only, system, archived (non usato), hidden
- Bit riservati: 10 bit non utilizzati
- Time: ore (5bit), min (6bit), sec (5bit)
- Date: giorno (5bit), mese (4bit), anno-1980 (7bit) (Y2108 BUG!)

Differenze tra FAT

La differenza fra FAT12, FAT16 e FAT32 consiste in quanti bit sono allocati per numerare i blocchi del disco

- Con 12 bit, il file system può indirizzare al massimo $2^{12} = 4096$ blocchi,
- mentre con 32 bit si possono gestire $2^{32} = 4.294.967.296$ blocchi

L'aumento del numero di bit di indirizzo dei blocchi e la dimensione stessa del blocco sono stati resi necessari per gestire unità a disco sempre più grandi e capienti

Valori ammessi come dimensione di blocco: 512 byte, 1, 2, 4, 8, 16 e 32 KB

Differenze tra FAT: dimensione max partizione

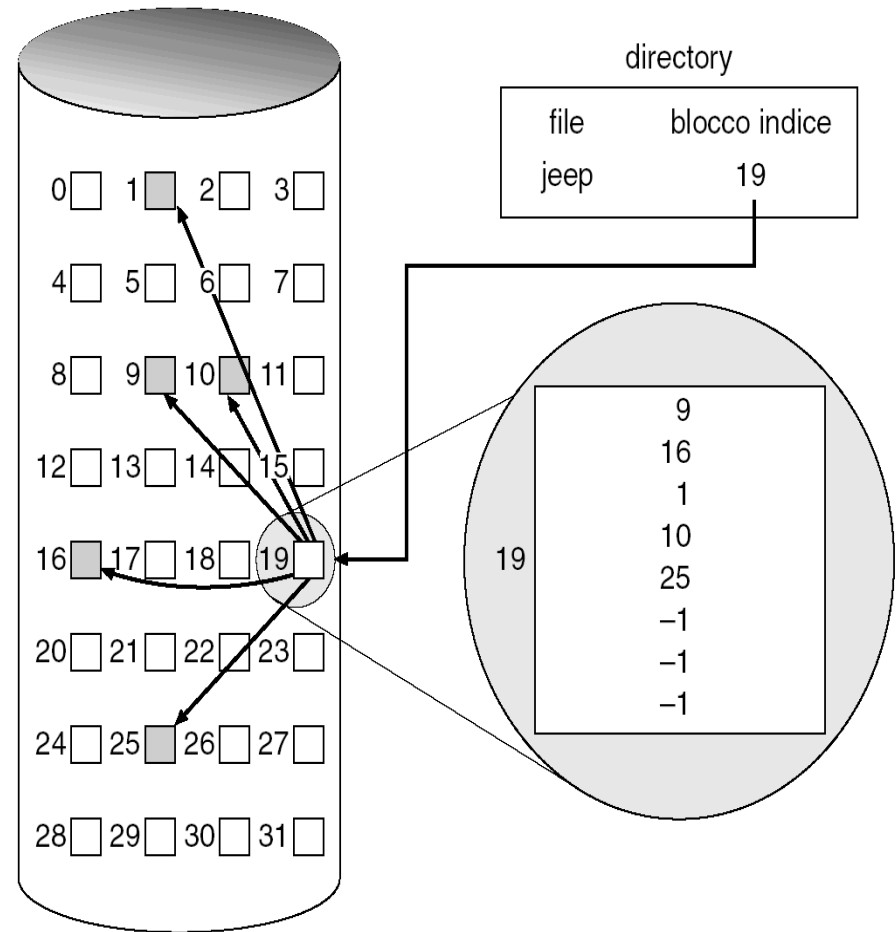
Il numero di bit usati per la FAT e la dimensione dei blocchi influenza la dimensione massima di una partizione

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Allocazione indicizzata

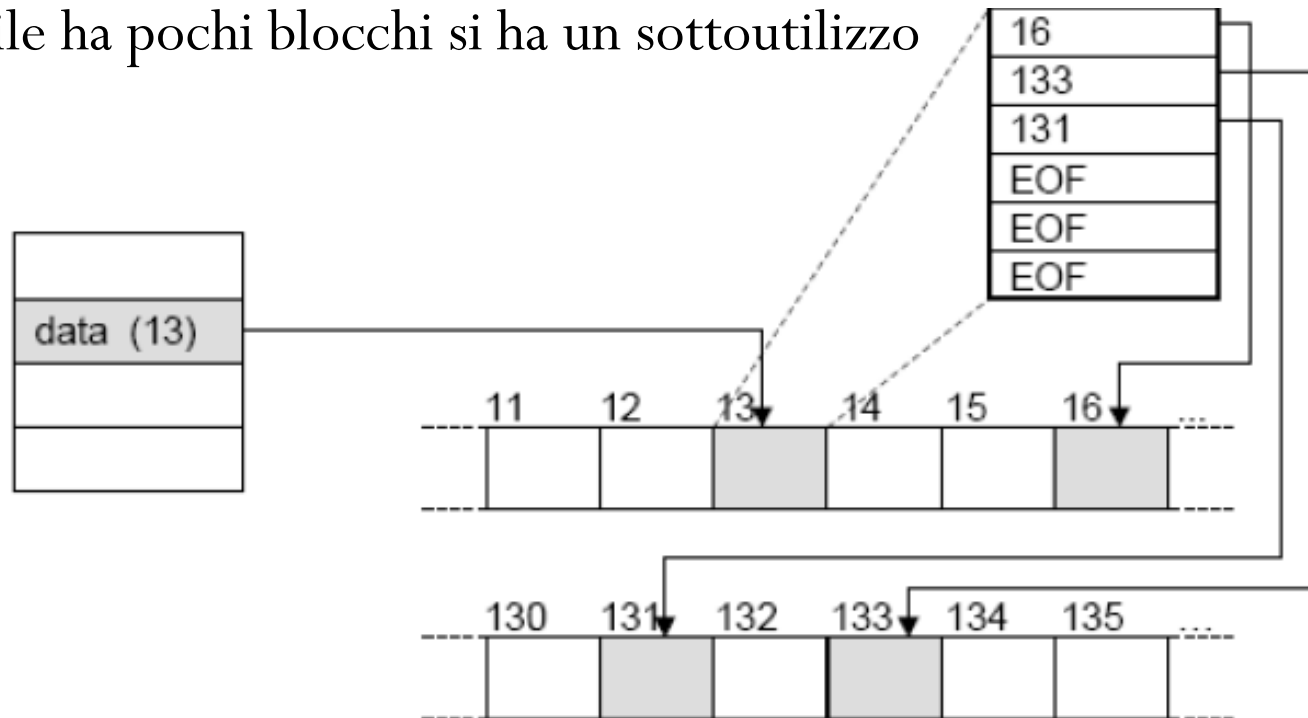
Ogni file possiede un indice dei blocchi memorizzato in un **blocco indice**

- Questo blocco contiene un array di puntatori agli altri blocchi del file
- L'*i*-esimo elemento dell'array punta all'*i*-esimo blocco del file
- La directory memorizza per ogni file un puntatore al suo blocco indice
- Questo risolve il problema dell'**accesso diretto** presente nell'allocazione concatenata ed evita la **frammentazione esterna**



Allocazione indicizzata (2)

- Nessuna frammentazione esterna
- Accesso diretto veloce
- Maggiore affidabilità, perché spesso si mantengono diverse copie del blocco indice.
- Ma ha **overhead di spazio** per il blocco indice
 - se un file ha pochi blocchi si ha un sottoutilizzo



Allocazione indicizzata ⁽³⁾

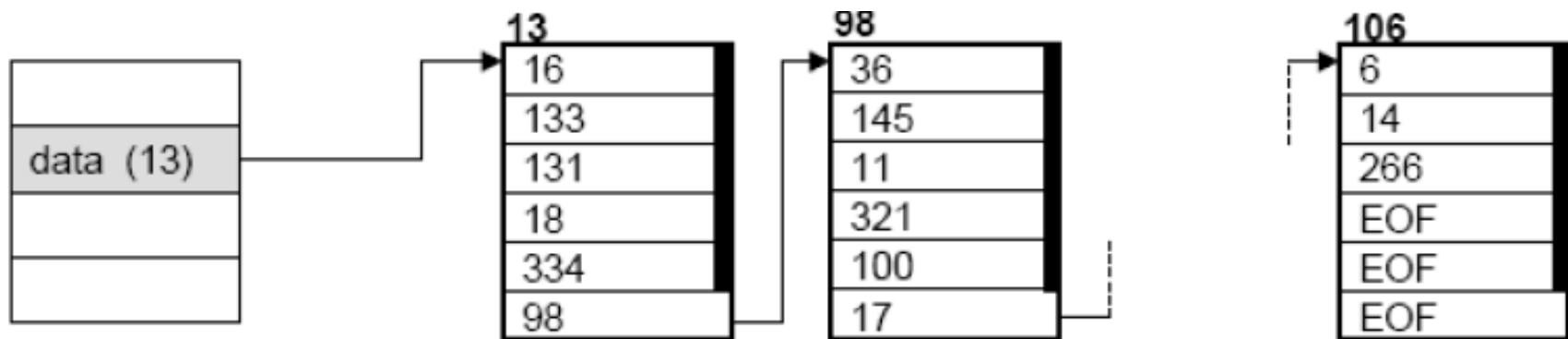
- Dimensionamento del blocco indice
 - Schema collegato
 - Indice multilivello
 - Schema combinato (usato nell'UFS in Unix)

Allocazione indicizzata (4)

- Dimensionamento del blocco indice

- Schema collegato

- Si collegano tra loro più blocchi indice (se necessari per gestire file grandi)
 - l'ultimo elemento del file indice punta ad un ulteriore blocco indice

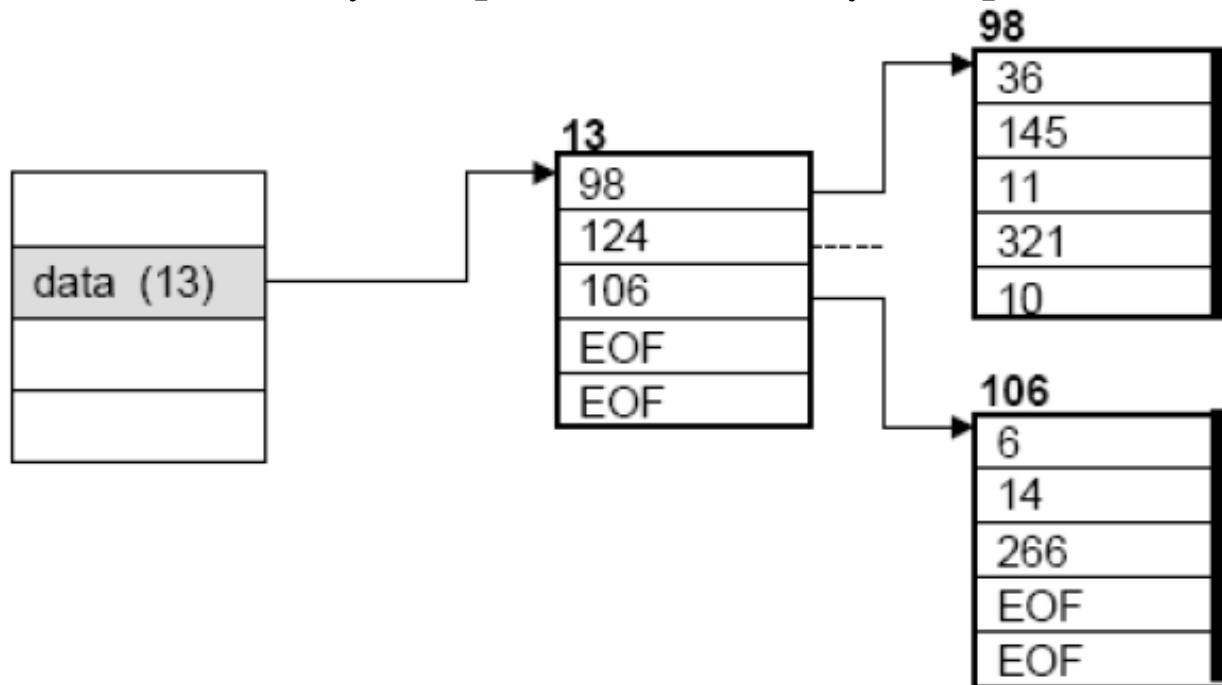


Allocazione indicizzata ⁽⁵⁾

- Dimensionamento del blocco indice

Indice multilivello

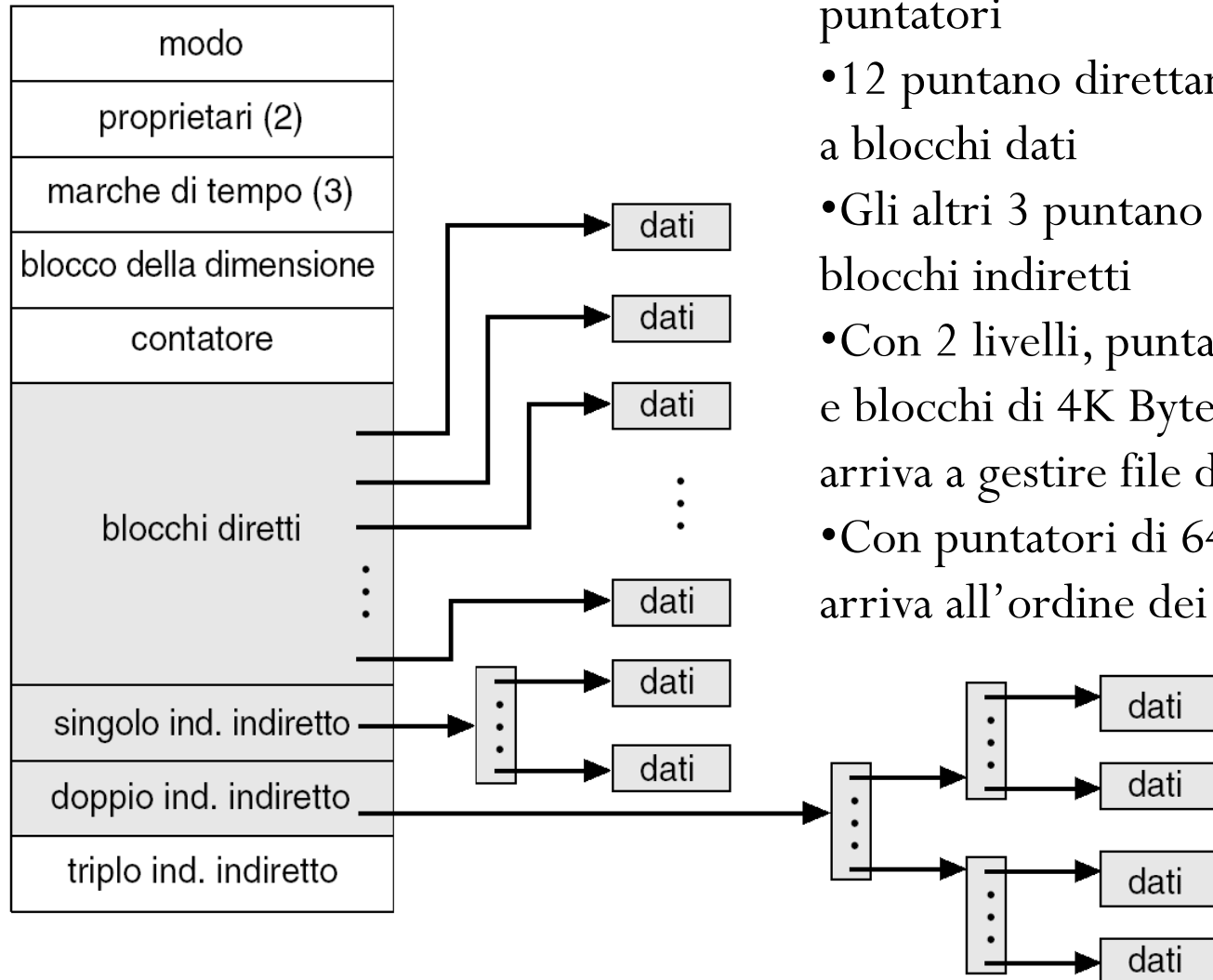
- A 2 livelli: Un **blocco indice di primo livello** punta a un insieme di **blocchi indice di secondo livello** che puntano a **blocchi dati** del file. È possibile estendere lo schema a 3-4 livelli.
- Con blocchi da 4 KByte e puntatori da 4 Byte si possono indicizzare file da 4GB



Allocazione indicizzata (5)

• Dimensionamento del blocco indice

• Schema combinato (UFS)



Ogni **inode** contiene 15 puntatori

- 12 puntano direttamente a blocchi dati

- Gli altri 3 puntano a blocchi indiretti

- Con 2 livelli, puntatori di 32 bit (4 Byte) e blocchi di 4K Byte, si arriva a gestire file di 4GB

- Con puntatori di 64 bit, si arriva all'ordine dei terabyte

Gestione dello spazio libero ⁽¹⁾

- Vettore di bit
- Lista collegata
- Raggruppamento
- Conteggio

Gestione dello spazio libero

- **Vettore di bit o bitmap** (n blocchi)

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

1011000110110



Per creare un file

- Ricerca del primo bit a 1 (allocazione concatenata o indicizzata)
- Ricerca di un blocco di bit a 1 sufficientemente grande (allocazione contigua)

Pro: semplice ed efficiente nel trovare blocchi liberi consecutivi sul disco

Contro: per essere efficiente deve però essere mantenuta in memoria centrale

- Dischi grandi richiedono vettori di bit molto grandi

Gestione dello spazio libero

- **Vettore di bit o bitmap** (n blocchi)

bit[i] = $1 \Rightarrow$ blocco[i] libero
 $0 \Rightarrow$ blocco[i] occupato

1011000110110

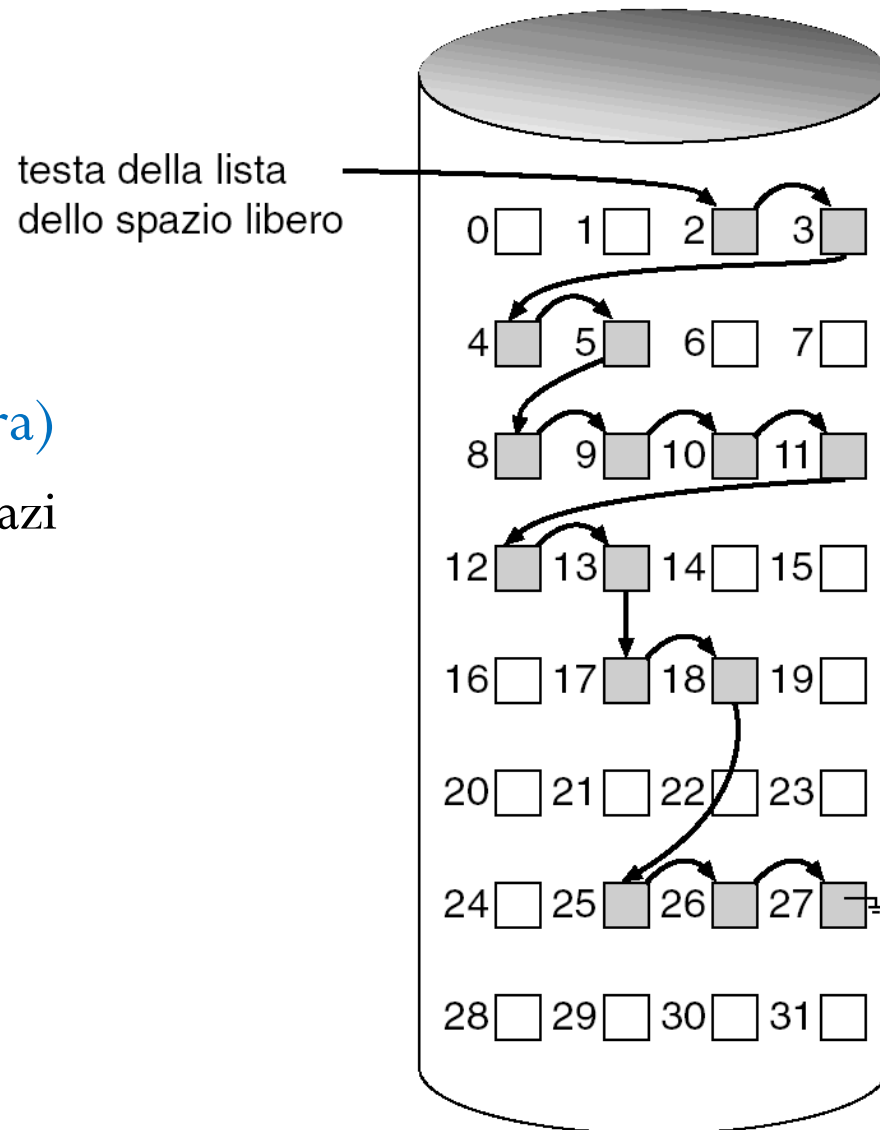


I calcolatori forniscono supporto HW

- Istruzioni per trovare lo scostamento del primo bit a 1 in una parola
- Se l'array è diviso in parole di n bit:
- $\text{indice_blocco} = (n * \text{num_parole_0}) + \text{scostamento_primo_bit_1}$
 - Num_parole_0 è il numero di parole consecutive con tutti i bit a 0 partendo dalla prima parola

Gestione dello spazio libero (3)

- Lista collegata (lista libera)
 - Non trova facilmente spazi contigui
 - No spreco di spazio



Gestione dello spazio libero ⁽⁴⁾

Altri 2 metodi (varianti della lista collegata):

- **Raggruppamento**

- **La memoria contiene un puntatore ad un blocco contenente gli indirizzi di n blocchi liberi**
- I primi $n-1$ blocchi sono effettivamente liberi, mentre l'ultimo contiene gli indirizzi di altri n blocchi liberi, e così via.

- **Conteggio**

- Si usa un array in cui ogni elemento contiene **l'indirizzo di un blocco libero** e il **numero di n blocchi liberi contigui** che seguono il blocco
 - La lista globale è più corta, perché solitamente il contatore è > 1