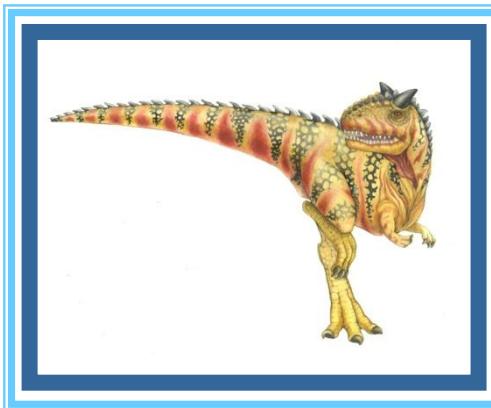


La memoria secondaria

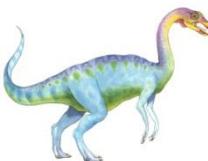




Memorie di massa e dispositivi di I/O

- ❖ I dispositivi collegati ad un calcolatore possono avere caratteristiche altamente variabili
- ❖ Rappresentano, nel loro insieme, la componente più lenta, voluminosa e variegata dell'elaboratore
- ❖ Il sistema operativo deve offrire alle applicazioni funzionalità che consentano l'accesso ai dispositivi mediante interfacce semplici e uniformi
- ❖ Deve inoltre garantire l'ottimizzazione dell'I/O, che costituisce il collo di bottiglia delle prestazioni del sistema di calcolo





Memorie di massa

- ❖ I dischi magnetici e i dispositivi di memoria non volatile (*nonvolatile memory*, **NVM**) costituiscono i supporti fondamentali di **memoria secondaria** nei computer attuali
 - **Dischi magnetici** (HDD – Hard Disk)
 - **Dischi a stato solido** (SSD – Solid State Disk)
- ❖ **Memorie terziarie**
 - Un solo drive, molti dispositivi rimovibili
 - ▶ Pen drive (flash)
 - ▶ CD, DVD, Blu-ray
 - Spazio di memorizzazione su cloud





Gestione dell'unità a disco – 1

❖ Formattazione di basso livello o fisica

- Si suddivide il disco in settori, che possono essere letti e scritti dal controllore del disco
- Dimensione standard pari a 4KB
- Nel caso di NVM devono essere inizializzate le pagine e creata la tabella **FTL** (Flash Translation Layer)
- In entrambi i casi, la formattazione di basso livello inserisce nel dispositivo una speciale struttura dati per ogni “blocco” di memoria:
 - ▶ Intestazione, dati, coda
 - ▶ L'intestazione e la coda contengono informazioni (numero settore/pagina, codice ECC) ad uso del controllore

❖ La formattazione fisica è eseguita dal costruttore dell'unità come parte del processo produttivo





Gestione dell'unità a disco – 2

- ❖ Per poter impiegare un dispositivo per memorizzare i file, il SO deve mantenere le proprie strutture dati sul disco (HDD/SSD)
 - Si **partiziona** il dispositivo in uno o più gruppi di cilindri/pagine, ognuno dei quali rappresenta un “disco logico”
 - **Formattazione logica** o “creazione di un file system”
 - ▶ Strutture dati del SO per la descrizione dello spazio libero/occupato e creazione di una directory iniziale vuota
 - Per migliorare le prestazioni, la maggior parte dei file system accorpa i blocchi in gruppi, detti cluster
 - ▶ I/O su disco fatto per blocchi
 - ▶ I/O via file system fatto per cluster (accesso sequenziale)
 - ▶ File e metadati vicini su HDD per diminuire i movimenti della testina





Gestione dell'unità a disco – 3

- ❖ Un **disco di avviamento** o **disco di sistema** ha una partizione di boot
- ❖ Il primo blocco logico del dispositivo è il **blocco di avvio** o **boot block**
- ❖ La **partizione di boot** contiene il SO; altre partizioni possono contenere altri SO, altri file system o essere partizioni raw
 - Viene montata all'avvio del sistema
 - Altre partizioni possono essere montate automaticamente o manualmente (al boot o successivamente)
- ❖ Al momento del montaggio di ogni partizione, si verifica la coerenza del file system (controllando la correttezza dei metadati)
 - Si aggiorna la tabella di montaggio





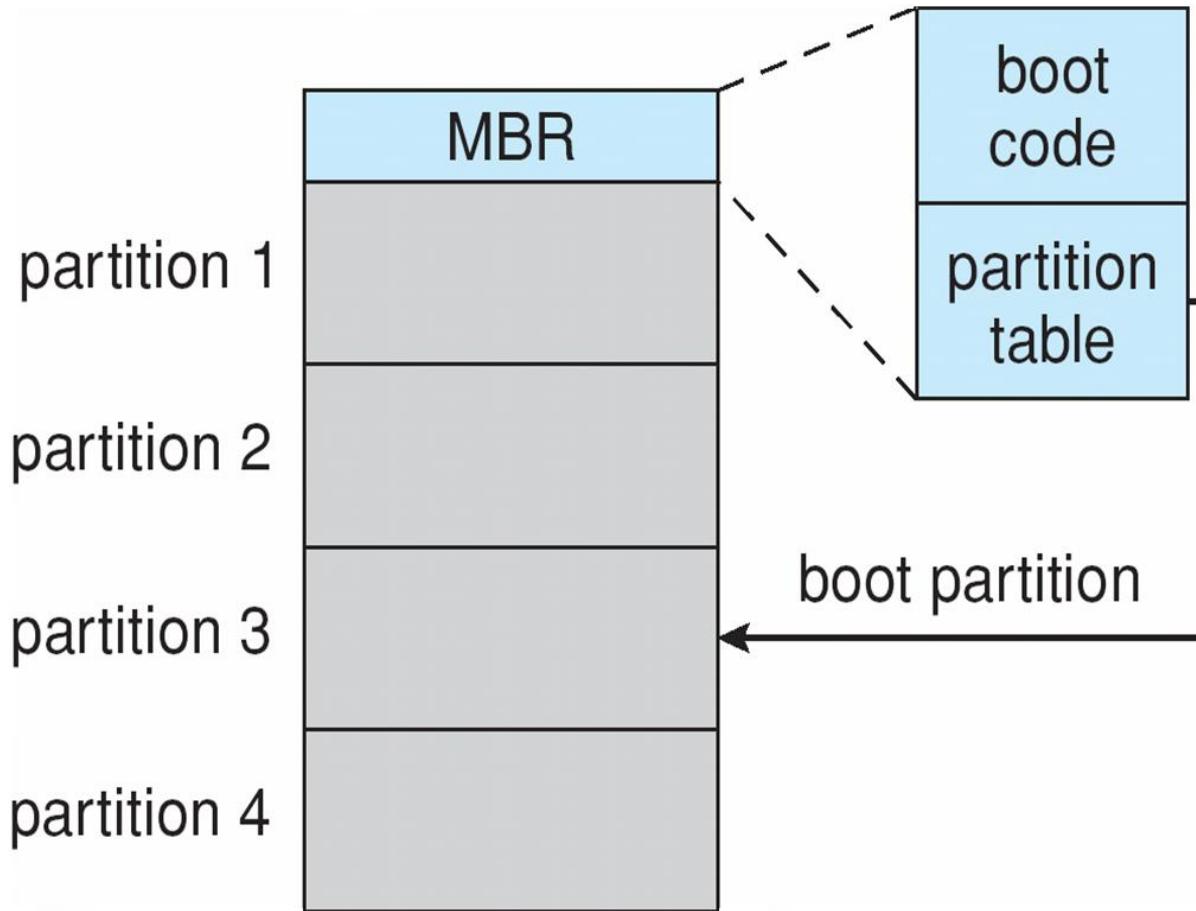
Gestione dell'unità a disco – 3

- ❖ Nel **boot block** sono contenute le informazioni necessarie all'inizializzazione del sistema
- ❖ In Windows si chiama **MBR** (Master Boot Record)
 - Esecuzione del codice del **bootstrap loader** contenuto nel firmware
 - Lettura/esecuzione del codice contenuto nell'MBR, che contiene anche una tabella delle partizioni, con un flag che identifica la partizione di boot
 - Caricamento, dalla partizione di boot, del kernel e dei sottosistemi del SO





Avviamento dal disco di Windows



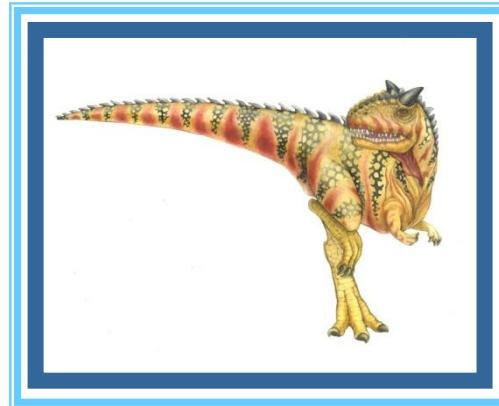


Gestione dell'area di swap

- ❖ La memoria virtuale impiega lo spazio su disco come un'estensione della memoria centrale
 - Pratica attualmente meno comune, grazie all'incremento nella capacità delle memorie
- ❖ L'obiettivo principale nella progettazione e realizzazione dell'area di swap è di fornire la migliore produttività per il sistema di memoria virtuale
- ❖ Lo spazio di swap può essere ricavato all'interno del normale file system o, più comunemente, si può trovare in una partizione separata del disco
 - **Partizione raw:** adotta algoritmi ottimizzati rispetto alla velocità di accesso piuttosto che all'occupazione di spazio (frammentazione)



Realizzazione del file system





Struttura del file system – 1

- ❖ Struttura del file:
 - Unità di memorizzazione logica
 - Collezione di informazioni correlate
 - **File control block:** struttura dati del kernel che “descrive” il file
- ❖ Il file system risiede nella memoria secondaria
 - Fornisce una semplice interfaccia per la memorizzazione non volatile, realizzando il mapping fra indirizzi logici e fisici
 - Fornisce la possibilità di accedere alla memoria in maniera efficiente, per memorizzare e recuperare rapidamente le informazioni





Struttura del file system – 3

- ❖ Le operazioni di I/O su disco avvengono con “granularità” di blocco
 - Ciascun blocco è composto da uno o più settori (512–4096 byte per settore)
 - Dimensione media attuale dei blocchi 4KB
 - I dispositivi fisici vengono controllati da **device driver**
- ❖ Anche i dispositivi NVM hanno normalmente blocchi di 4KB e utilizzano metodi di trasferimento simili a quelli delle unità a disco
- ❖ **File system per l'utente**
 - Definizione (dell'aspetto) di file e directory e loro operazioni
- ❖ **File system per il SO**
 - Scelta di algoritmi e strutture dati che mettano in corrispondenza il file system logico con i dispositivi fisici di memorizzazione





Tipi di file system – 1

- ❖ Esistono diversi tipi di file system e non è raro che i sistemi operativi ne prevedano più di uno
 - CD–ROM: **ISO 9660**
 - UNIX: **UFS** (UNIX File System) che si fonda sul Berkeley Fast File System (FFS)
 - Windows: **FAT**, **FAT32**, **exFAT**, **NTFS**
 - Linux: **ext2**, **ext3**, **ext4** (extended file system), **xfs**, ma ne supporta più di centotrenta tipi diversi
 - **FUSE** (File system in USErspace): è un progetto open-source, rilasciato sotto la licenza GPL e LGPL, volto alla realizzazione di un modulo integrabile al kernel Linux che permetta agli utenti non privilegiati di creare un proprio file system senza scrivere codice a livello kernel





Tipi di file system – 3

- ❖ Google ha progettato un file system (proprietario) per soddisfare le esigenze di memorizzazione e recupero dati specifiche dell'azienda
 - **GoogleFS**, detto anche **BigFiles**, conserva i dati raccolti da Google utilizzando tecnologie non convenzionali, data la grandezza dei file
 - Dati immagazzinati in maniera permanente in file di circa 100GB ciascuno, solo raramente eliminati, sovrascritti o compressi
 - File con accesso per sola lettura
- ❖ **Oracle ASM** (Automatic Storage Management) permette di gestire file, directory, volumi per mezzo di direttive SQL in ambiente DBMS

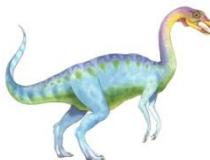




Metodi di allocazione

- ❖ La natura ad accesso diretto dei dischi garantisce flessibilità nell'implementazione dei file
- ❖ **Problema:** allocare lo spazio disco ai file in modo da avere spreco minimo di memoria e rapidità di accesso
- ❖ Il metodo di allocazione dello spazio su disco descrive come i blocchi fisici del disco vengono allocati ai file
 - **Allocazione contigua**
 - **Allocazione concatenata**
 - **Allocazione indicizzata**
- ❖ Anche se alcuni SO dispongono di tutti i metodi, più spesso un sistema usa un unico metodo per tutti i file di un dato file system





Allocazione contigua – 1

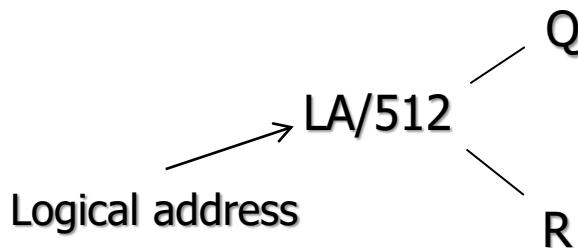
- ❖ Ciascun file occupa un insieme di blocchi contigui sul disco
- ❖ Per reperire il file occorrono solo la locazione iniziale (# blocco iniziale) e la lunghezza (numero di blocchi)
 - Accesso sequenziale \Rightarrow Nessuno o, al più, un solo spostamento della testina su un cilindro attiguo
 - Accesso casuale \Rightarrow Performance ottimali
- ❖ Problemi
 - Allocazione dinamica dello spazio disco
 - Frammentazione esterna
 - Necessità di conoscere a priori la dimensione dei file (i file non possono crescere, soprattutto per *best-fit*)
 - Compattazione dello spazio disco



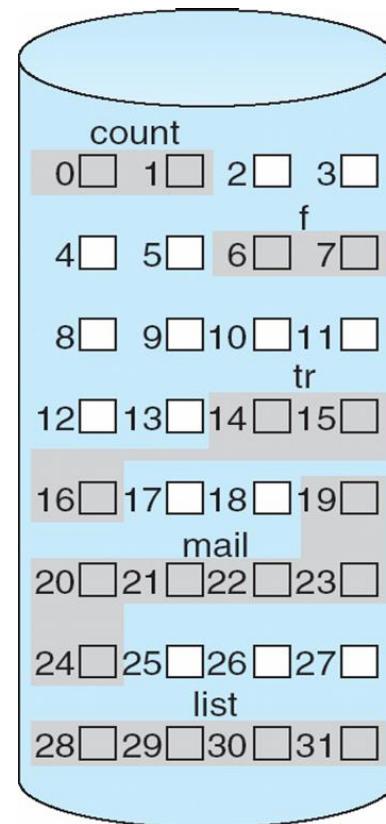


Allocazione contigua – 2

- ❖ Mapping da blocchi logici a blocchi fisici (hp: dim. blocco pari a 512 byte/word)



Il blocco da accedere è il Q-esimo a partire dall'indirizzo del blocco iniziale; lo spostamento all'interno del blocco è pari ad R



directory		
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Elemento di directory

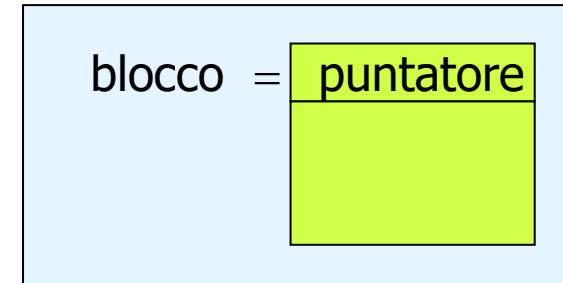




Allocazione concatenata – 1

- ❖ Ciascun file è una lista concatenata di blocchi: i blocchi possono essere sparsi ovunque nel disco

- Ciascun blocco contiene un puntatore al blocco successivo
- Il file termina quando si incontra un blocco con puntatore vuoto
- Non si ha spreco di spazio (no frammentazione esterna)
- Quando necessita di un nuovo blocco da allocare ad un file (il file può crescere), il SO invoca il sottosistema per la gestione dello spazio libero
- L'efficienza può essere migliorata raccogliendo i blocchi in cluster (aumenta, però, la frammentazione interna)



❖ Problemi

- Accesso casuale impossibile: reperire un blocco può richiedere molte operazioni di I/O ed operazioni di seek
- Affidabilità legata ai puntatori



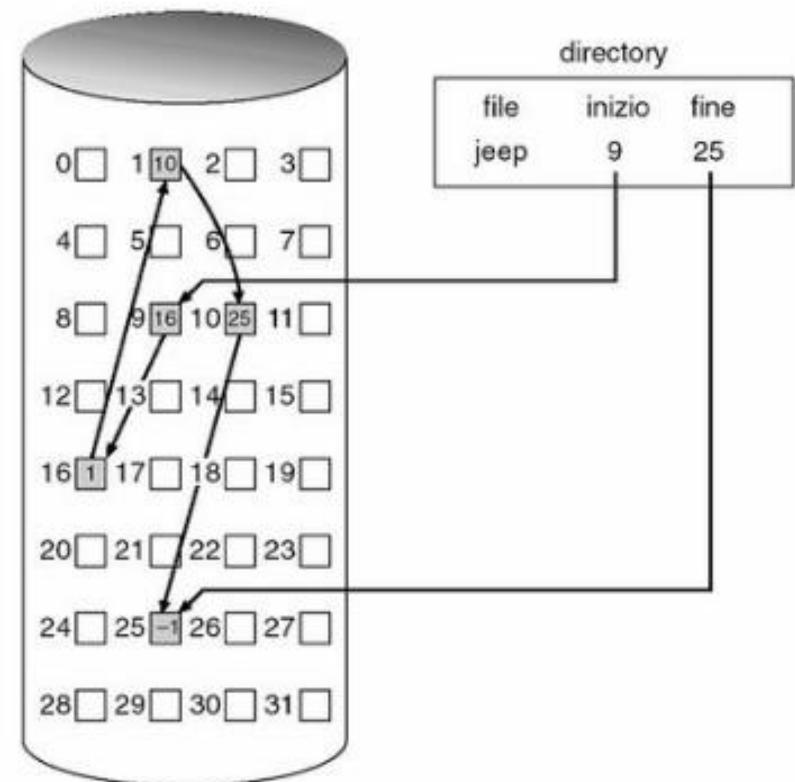


Allocazione concatenata – 2

- ❖ Nella directory, si mantiene l'indirizzo dei blocchi iniziale e finale
- ❖ Mappatura da indirizzi logici a indirizzi fisici

LA/511
Q
R

Il blocco da accedere è il Q-esimo nella catena di blocchi che costituiscono il file; lo spostamento all'interno del blocco è pari ad R+1





File Allocation Table (FAT) – 1

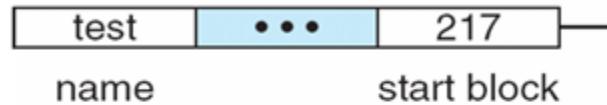
- ❖ Variante del metodo di allocazione concatenata implementata in MS–DOS e OS/2
 - Per contenere la FAT si riserva una sezione del disco all'inizio di ciascun volume
 - La FAT ha un elemento per ogni blocco del disco ed è indicizzata dal numero di blocco
 - L'elemento di directory contiene il numero del primo blocco del file
 - ▶ L'elemento della FAT indicizzato da quel blocco contiene a sua volta il numero del blocco successivo del file
 - ▶ L'ultimo blocco ha come elemento della tabella un valore speciale di fine file
 - I blocchi inutilizzati sono contrassegnati dal valore 0
 - Facilita l'accesso casuale: informazione relativa alla localizzazione di ogni blocco "concentrata" nella FAT
 - ▶ Può essere soggetta a caching





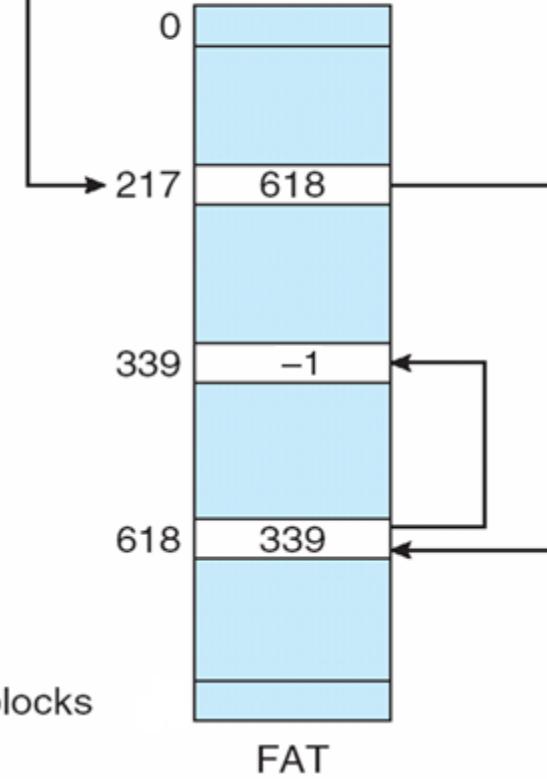
FAT – 2

directory entry



start block

no. of disk blocks



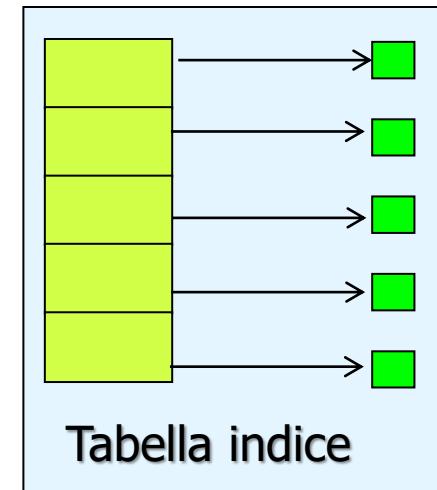
La FAT consente la memorizzazione “localizzata” dei puntatori





Allocazione indicizzata – 1

- ❖ Per ogni file, si collezionano tutti i puntatori in un unico **blocco indice**
- ❖ Richiede una tabella indice
- ❖ Accesso casuale
- ❖ Permette l'accesso dinamico senza frammentazione esterna; tuttavia c'è il sovraccarico temporale di accesso al blocco indice
- ❖ Nella directory si memorizza l'indirizzo del blocco indice
- ❖ Mappatura da indirizzi logici a indirizzi fisici per file di dim. max 256K parole e con dimensione di blocco di 512 parole: occorre un solo blocco indice



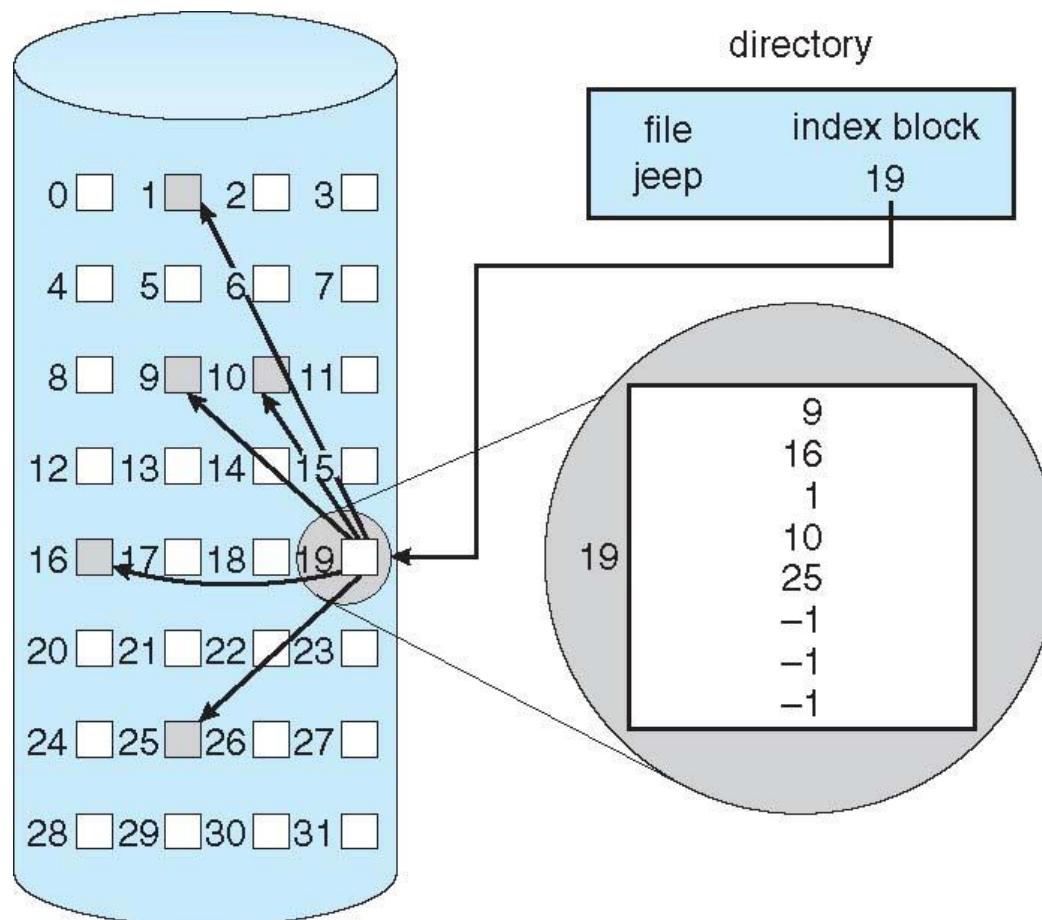
LA/512
Q
R

Q spostamento nella tabella indice
R spostamento all'interno del blocco





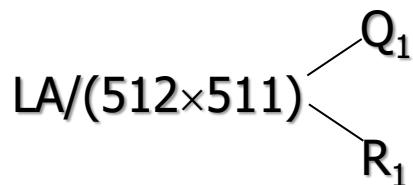
Allocazione indicizzata – 2



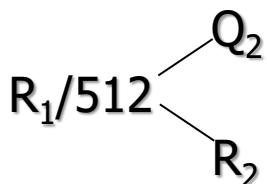


Allocazione indicizzata – 3

- ❖ Mapping fra indirizzi logici e fisici per un file di lunghezza qualunque (dim. blocco 512 byte/word)
- ❖ **Schema concatenato** — Si collegano blocchi della tabella indice
 - Il primo blocco indice contiene l'insieme degli indirizzi dei primi 511 blocchi del file, più un puntatore al blocco indice successivo



Q_1 spostamento nella tabella indice
 R_1 si utilizza come segue...



Q_2 spostamento nel blocco della tabella indice
 R_2 spostamento all'interno del blocco del file

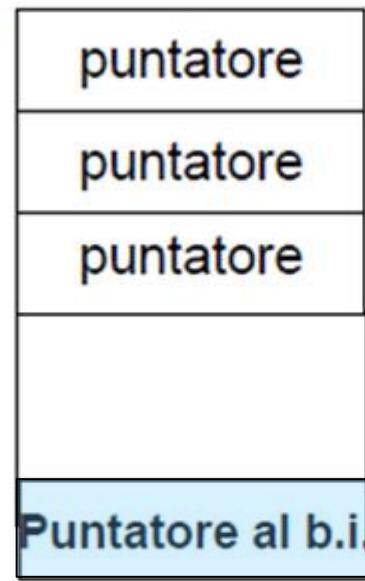




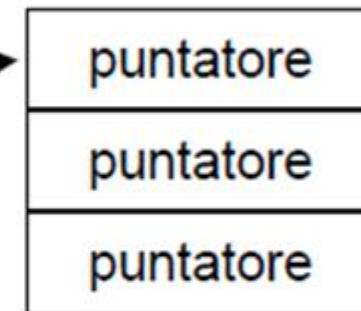
Allocazione indicizzata – 4

Indice concatenato

Blocco indice



Blocco indice





Allocazione indicizzata – 5

❖ Indice a più livelli

- Indice a due livelli



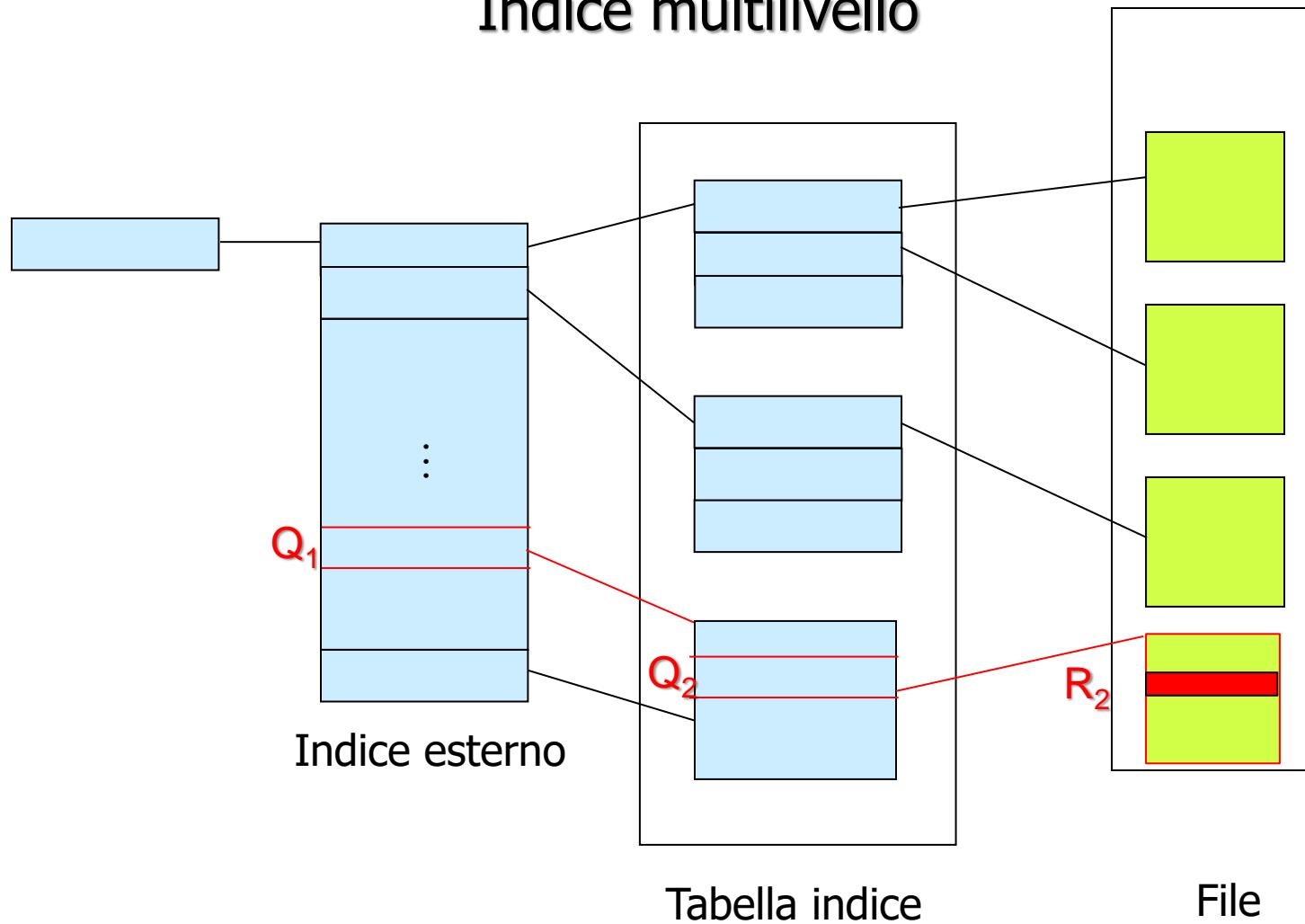
- Blocchi da 4K possono contenere 1024 puntatori da 4 byte nell'indice esterno → per un totale di 1048567 blocchi di dati e file di dimensione massima pari a 4GB





Allocazione indicizzata – 6

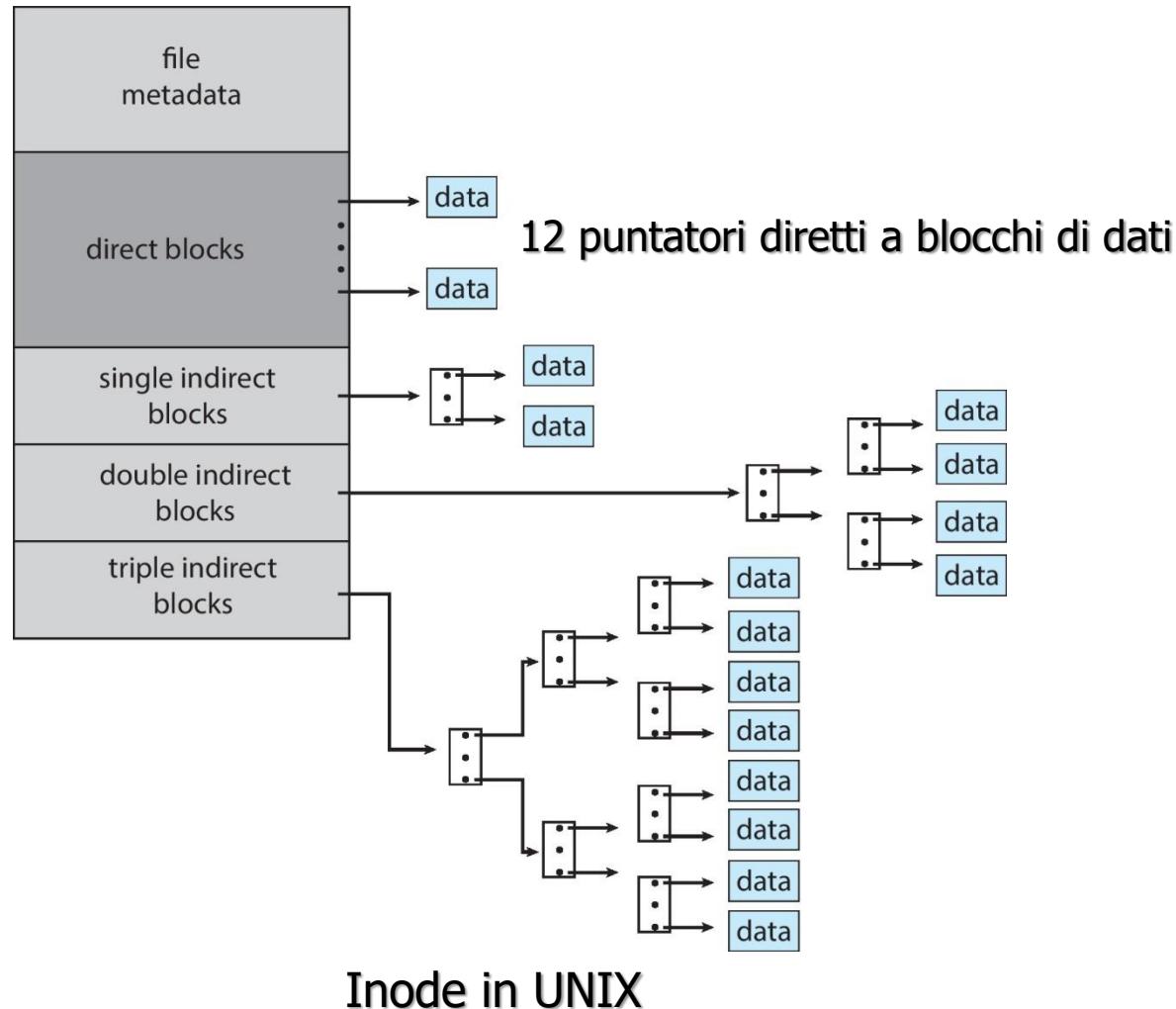
Indice multilivello





Schema combinato: UNIX UFS

4Kbyte per blocco, indirizzi a 32 bit





Scelta del metodo di allocazione – 1

- ❖ Il miglior metodo per l'allocazione di file dipende dal tipo di accesso
 - L'allocazione contigua ha ottime prestazioni sia per accesso sequenziale che casuale
 - L'allocazione concatenata si presta naturalmente all'accesso sequenziale
 - ⇒ Dichiарando il tipo di accesso all'atto della creazione del file, si può selezionare il metodo di allocazione più adatto
- ❖ L'allocazione indicizzata è “più complessa”
 - L'accesso ai dati del file può richiedere più accessi a disco (tre, nel caso di un indice a due livelli)
 - Tecniche di clustering possono migliorare il throughput, riducendo l'overhead di CPU





Esempio 1

- ❖ Si consideri un file formato da 70 record e le sue possibili allocazioni su disco di tipo sequenziale, mediante lista concatenata e con tabella indice. In ognuno di questi casi i record del file sono memorizzati uno per blocco. Le informazioni che riguardano il file sono già in memoria centrale e la tabella indice è contenuta in unico blocco. Si dica quanti accessi al disco sono necessari, in ognuna delle seguenti situazioni, per cancellare:
 - Il primo record;
 - Il quarantesimo record;
 - L'ultimo record.





Esempio 1 (cont.)

❖ Soluzione

▪ Allocazione sequenziale

- ▶ Per cancellare il primo record serve solo aggiornare l'elemento di directory (nessun accesso a disco);
- ▶ Per cancellare il quarantesimo record servono 30 letture e 30 scritture di blocchi: si legge il 41-esimo blocco e lo si copia nel 40-esimo, si legge il 42-esimo e lo scrive nel 41-esimo, etc.;
- ▶ Per cancellare l'ultimo record serve solo aggiornare l'elemento di directory (nessun accesso a disco).

▪ Allocazione concatenata (semplice)

- ▶ Per cancellare il primo record serve leggerlo per aggiornare l'elemento di directory (un accesso a disco);
- ▶ Per cancellare il quarantesimo record servono 40 letture e 1 scrittura per aggiornare il puntatore del blocco 39;
- ▶ Per cancellare l'ultimo record servono 69 letture e 1 scrittura per aggiornare il puntatore del blocco 69 (che diventa `null`).

▪ Allocazione indicizzata

- ▶ In tutti i casi deve essere letto e riscritto il solo blocco indice.





Esempio 2

- ❖ Si consideri un file formato da 80 blocchi, allocato su disco in modo sequenziale. Il blocco precedente a quelli occupati dal file è occupato, mentre sono liberi i due blocchi successivi. Si dica quante operazioni di I/O su disco sono necessarie per aggiungere un blocco all'inizio del file.
- ❖ **Soluzione**
Sono necessarie 161 operazioni di I/O su disco, 160 per spostare (leggere/scrivere) gli 80 blocchi del file, partendo dall'ultimo, più una per scrivere il nuovo blocco in testa al file.





Gestione dello spazio libero – 1

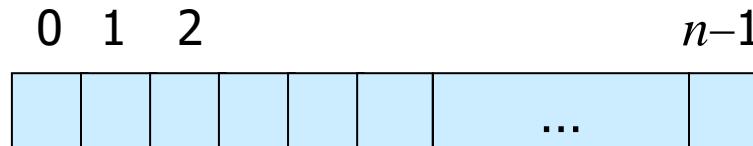
- ❖ Per tenere traccia dello spazio libero in un disco, il sistema conserva una **lista dello spazio libero**
 - Per creare un file occorre...
 - ▶ cercare nella lista dello spazio libero la quantità di spazio necessaria ed allocarla al nuovo file
 - ▶ aggiornare il contenuto della lista
 - Quando si cancella un file, si aggiungono alla lista dello spazio libero i blocchi di disco precedentemente assegnati al file
- ❖ La lista dello spazio libero può non essere realizzata come una lista





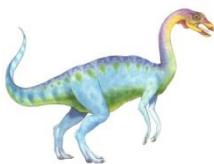
Gestione dello spazio libero – 2

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


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

- ❖ Calcolo del numero del primo blocco libero: si scorre il vettore, cercando il primo byte/parola diverso/a da 0
 - (numero di bit per parola) *
 - (numero di parole con valore 0) +
 - offset del primo bit a 1
- ❖ Buone prestazioni se il vettore è conservato in memoria centrale





Gestione dello spazio libero – 4

❖ Lista concatenata

- Si collegano tutti i blocchi liberi mediante puntatori e si mantiene un puntatore alla testa della lista in memoria centrale
 - Ogni blocco contiene un puntatore al successivo blocco libero
 - Non si spreca spazio (solo un puntatore)
 - Non è necessario attraversare tutta la lista, poiché di solito la richiesta è relativa ad un singolo blocco
 - Non facile da usare per ottenere spazio contiguo
-
- ❖ Nella FAT, il conteggio dei blocchi liberi è incluso nella struttura dati per l'allocazione (blocchi contrassegnati con 0) e non richiede quindi un metodo di gestione separato

