

File system

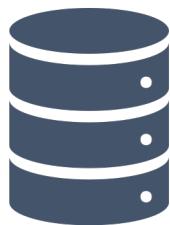
Anno Accademico 2020-2021
Graziano Pravadelli



UNIVERSITÀ
di **VERONA**

Dipartimento
di **INFORMATICA**

File system



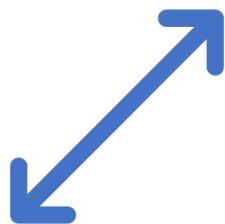
Fornisce il meccanismo per la memorizzazione e l'accesso di dati e programmi



Consiste di due parti

Collezione di file
Struttura di cartelle (directory)

Sommario



Interfaccia



Implementazione



Prestazioni



Interfaccia del file
system

Sommario



Concetto di file



Metodi di
accesso



Struttura delle
directory



Protezione



Concetto di file



SO astrae dalle caratteristiche fisiche dei supporti di memorizzazione fornendone una visione logica



File

Spazio di indirizzamento logico e contiguo
Insieme di informazioni correlate
identificate da un nome



Tipi di file

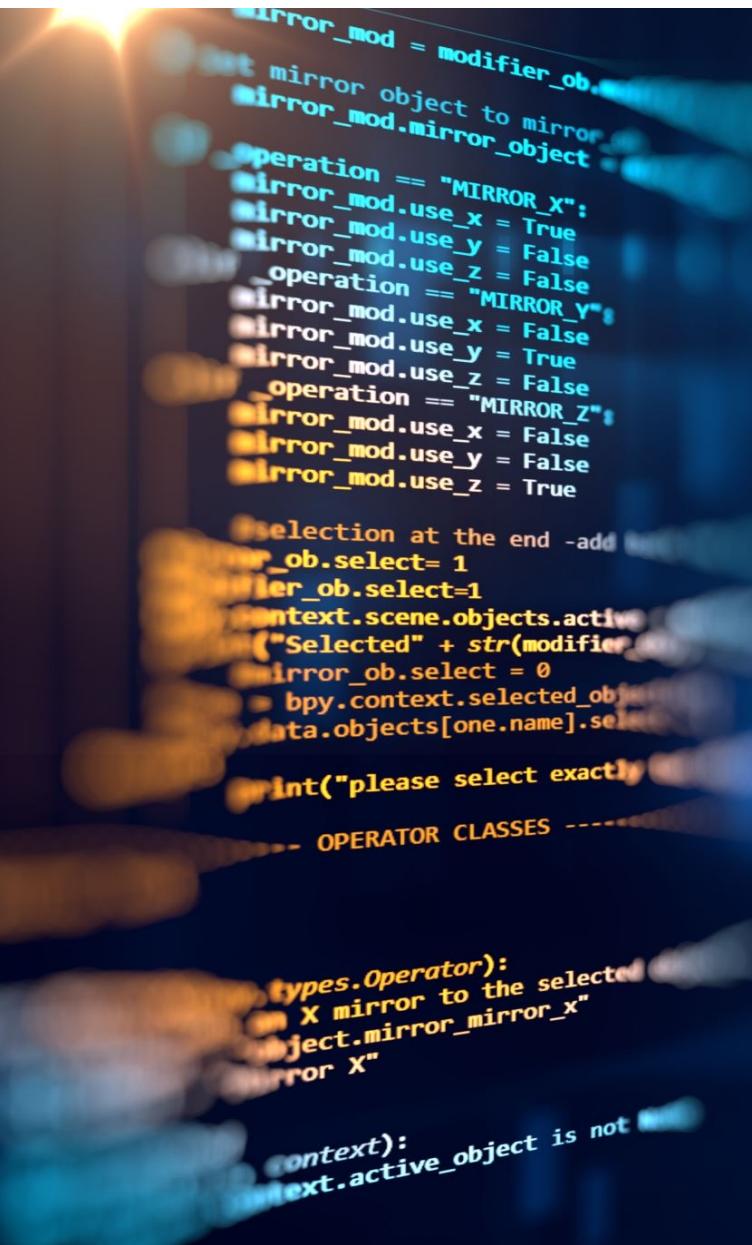
Dati Numerici, Caratteri, Binari
Programmi

Attributi di un file

- Memorizzati su disco nella struttura della directory
 - Nome
 - Unica informazione in formato “leggibile”
 - Tipo
 - Posizione
 - Puntatore allo spazio fisico sul dispositivo
 - Dimensione
 - Protezione
 - Controllo su chi può leggere, scrivere, eseguire
 - Tempo, data e identificazione dell’utente

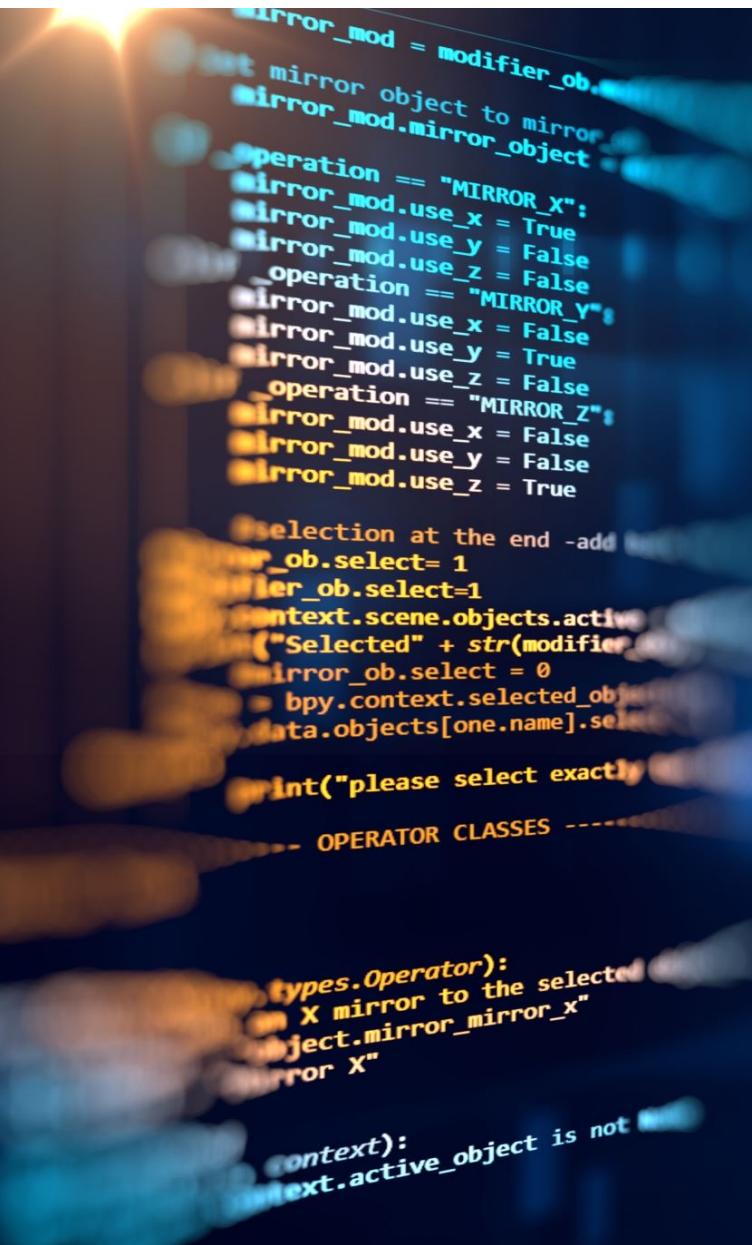
Operazioni su file (1)

- Creazione
 - Cercare spazio su disco
 - Nuovo elemento su directory per attributi
- Scrittura
 - System call che specifica nome file e dati da scrivere
 - Necessario puntatore alla locazione della prossima scrittura
- Lettura
 - System call che specifica nome file e dove mettere dati letti in memoria
 - Necessario puntatore alla locazione della prossima lettura (lo stesso della scrittura)



Operazioni su file (2)

- Riposizionamento all'interno di file
 - Aggiornamento puntatore posizione corrente
- Cancellazione
 - Libera spazio associato al file e l'elemento corrispondente nella directory
- Troncamento
 - Mantiene inalterati gli attributi ma cancella contenuto del file



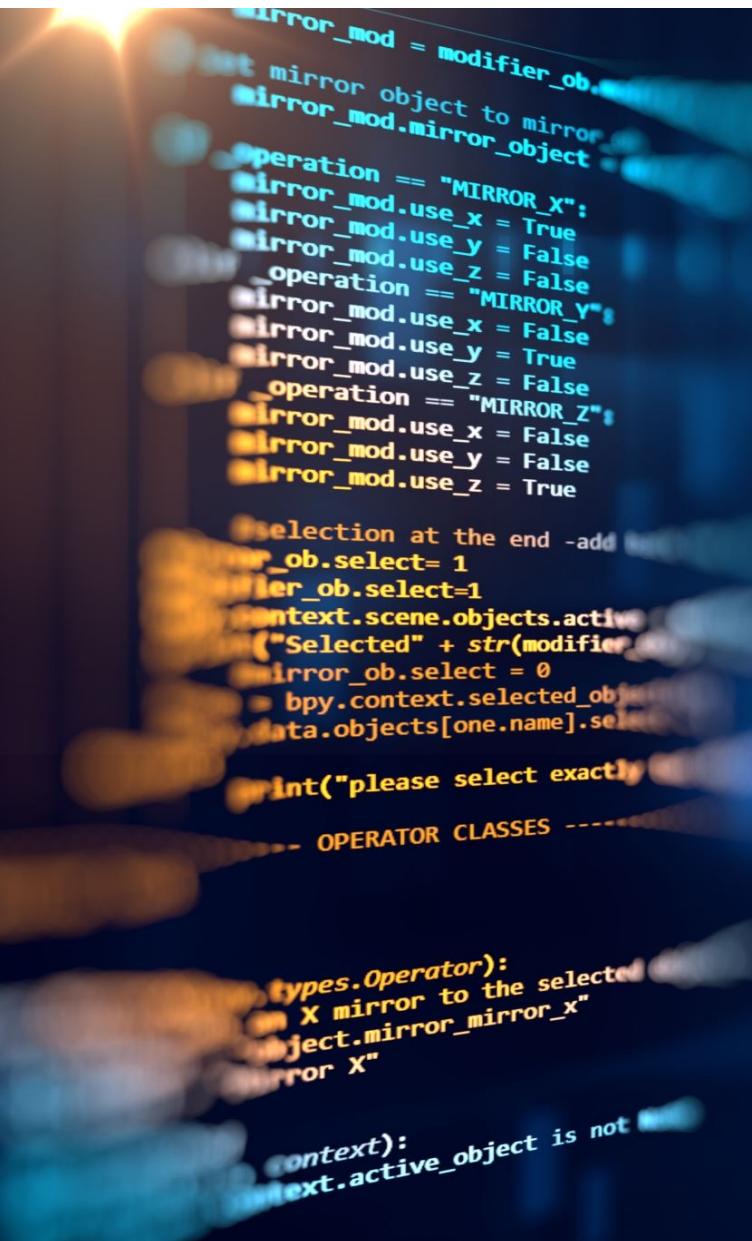
Operazioni su file (3)

- Apertura

- Ricerca del file nella struttura della directory su disco
- Copia del file in memoria e inserimento di un riferimento nella tabella dei file aperti
 - 2 tabelle per gestire i file aperti
 - Una tabella per ogni processo
 - contiene riferimenti per file aperti relativi al processo (es.: puntatore alla locazione di lettura/scrittura)
 - Una tabella per tutti i file aperti da tutti i processi
 - contiene idati indipendenti dal processo (es.: posizione sul disco, dimensione file, data accessi, n° di processi che hanno aperto il file)

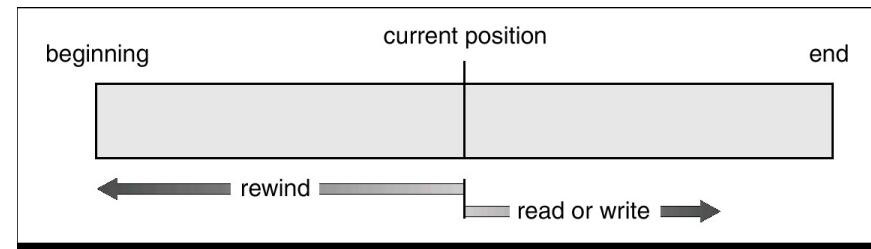
- Chiusura

- Copia del file in memoria su disco



Metodi di accesso – Sequenziale

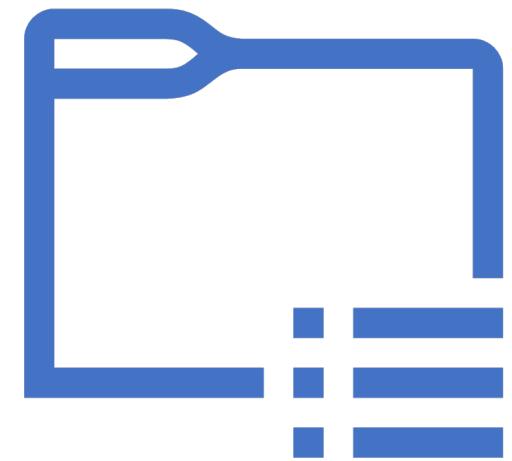
- Per esempio usato da editor e compilatori
- Operazioni permesse
 - read next
 - write next
 - reset (rewind)
- Non è permesso il rewrite
 - Rischio di inconsistenza se scrivo qualcosa a metà del file perché potrei cancellare ciò che sta dopo



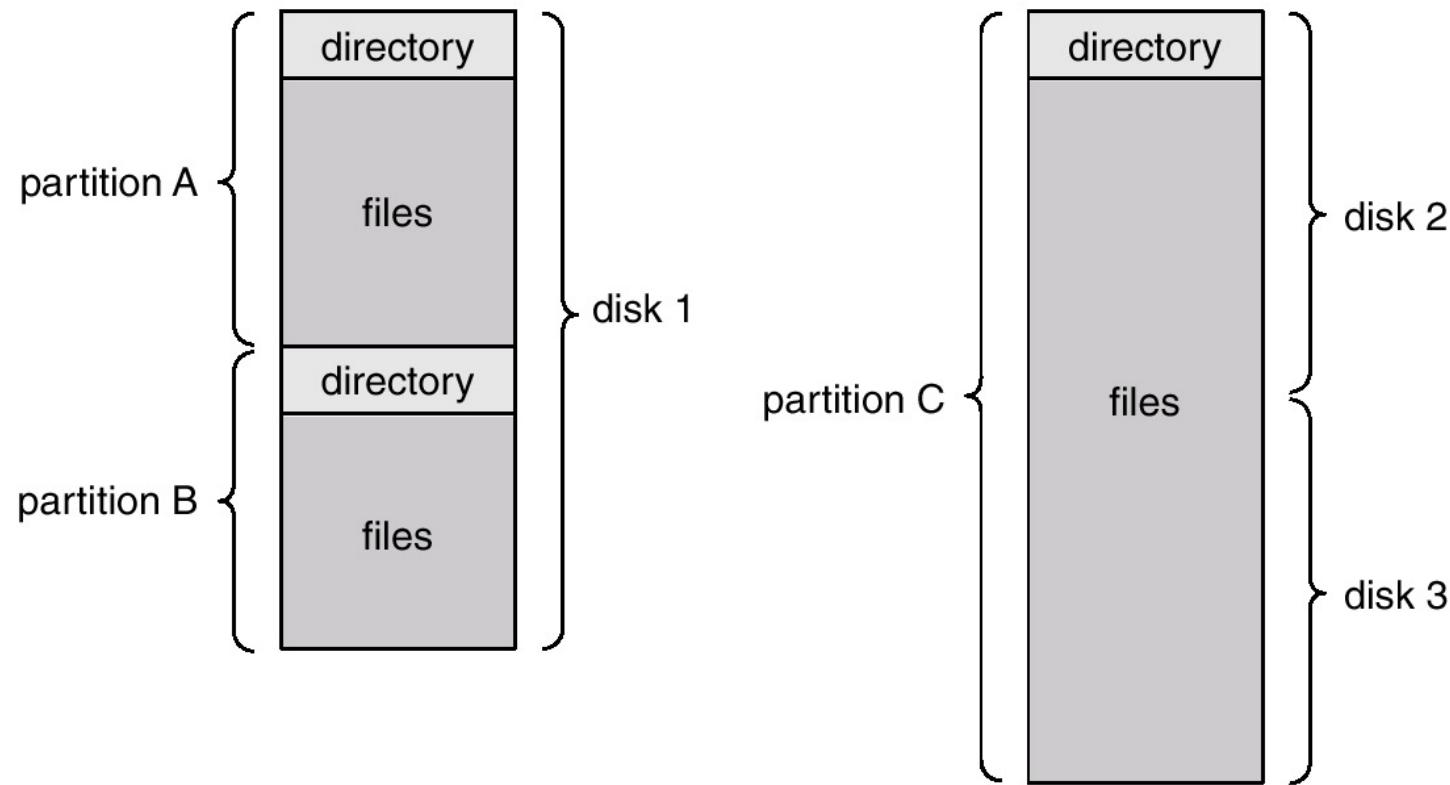
Metodi di accesso – Diretto

- Per esempio usato da database
 - File = sequenza numerata di blocchi (record)
- Operazioni permesse
 - read n
 - write n
 - position to n
 - read next
 - write next
 - rewrite n

Struttura delle directory

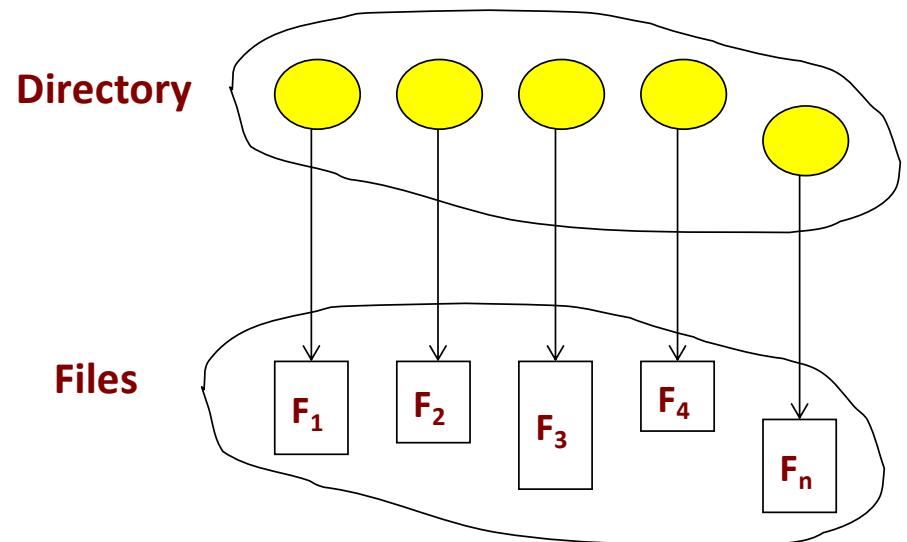


Organizzazione del file system



Struttura delle directory

- Collezione di nodi contenenti informazioni sui file
 - Residenti su disco
- Determinano la struttura del file system



Informazioni in una directory

- Per ogni file
 - Nome
 - Tipo
 - Indirizzo
 - Lunghezza attuale
 - Massima lunghezza
 - Data di ultimo accesso
 - Data di ultima modifica
 - Possessore
 - Info di protezione

Operazioni su directory

- Aggiungere un file
- Cancellare un file
- Visualizzare il contenuto della directory
- Rinominare un file
- Ricercare un file
 - Es.: cercare tutti i file il cui nome soddisfa una espressione
- Attraversare il file system



Directory – Obiettivi

Efficienza

- Rapido accesso ad un file

Nomenclatura (naming)

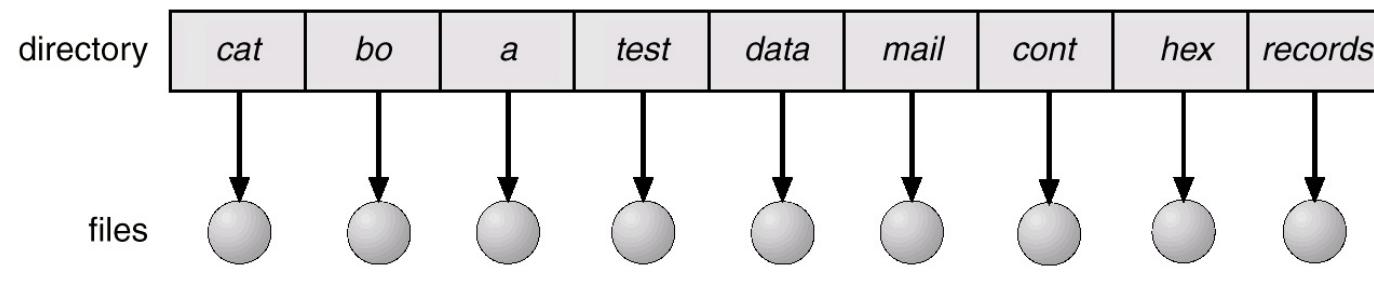
- Conveniente agli utenti
 - Stessi nomi per file diversi e utenti diversi
 - Nomi diversi per lo stesso file

Raggruppamento

- Classificazione logica dei file per criterio (tipo, protezione, etc.)

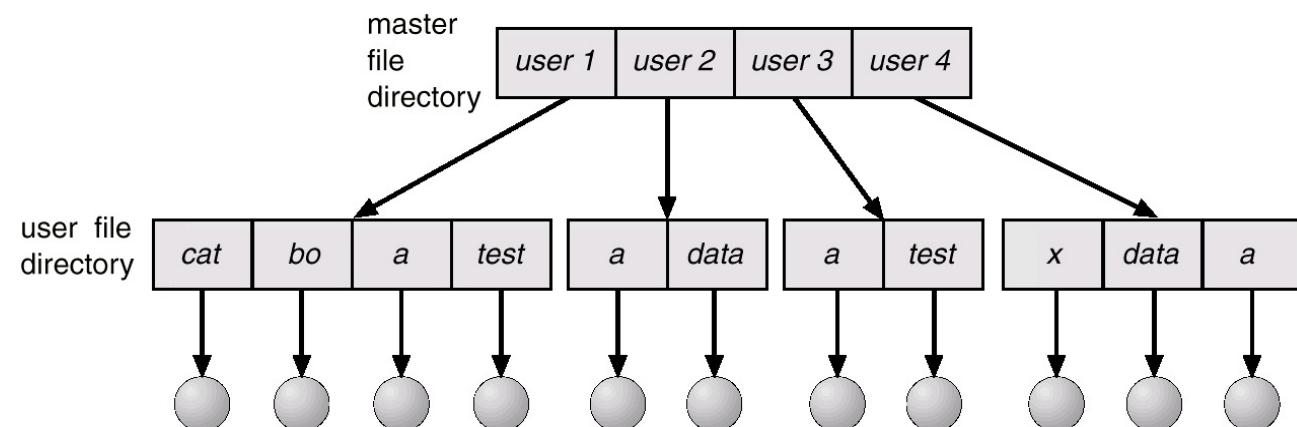
Directory a un livello

- Singola directory per tutti gli utenti
 - Problemi di nomenclatura
 - Difficile ricordare se un nome esiste già
 - Difficile inventare sempre nuovi nomi
 - Problemi di raggruppamento



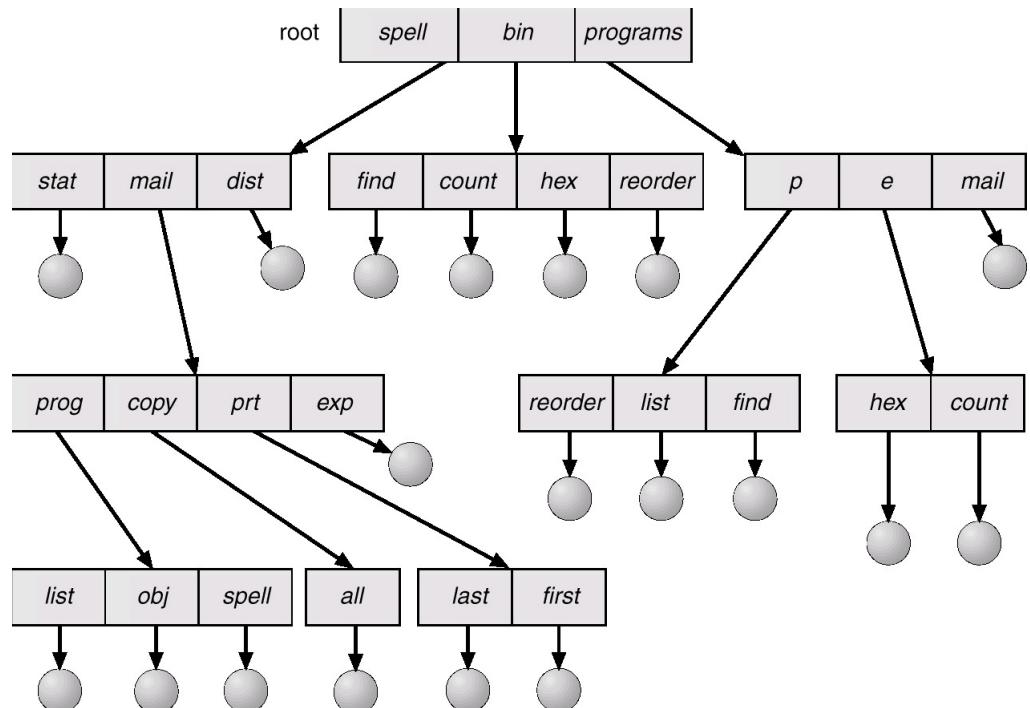
Directory a due livelli

- Directory separata per ogni utente
 - Concetto di percorso (path)
 - Possibilità di usare lo stesso nome di file per utenti diversi
 - Ricerca efficiente
 - No raggruppamento
 - Dove mettiamo i programmi di sistema condivisi dagli utenti?
 - Es.: In Unix si usa la variabile d'ambiente PATH



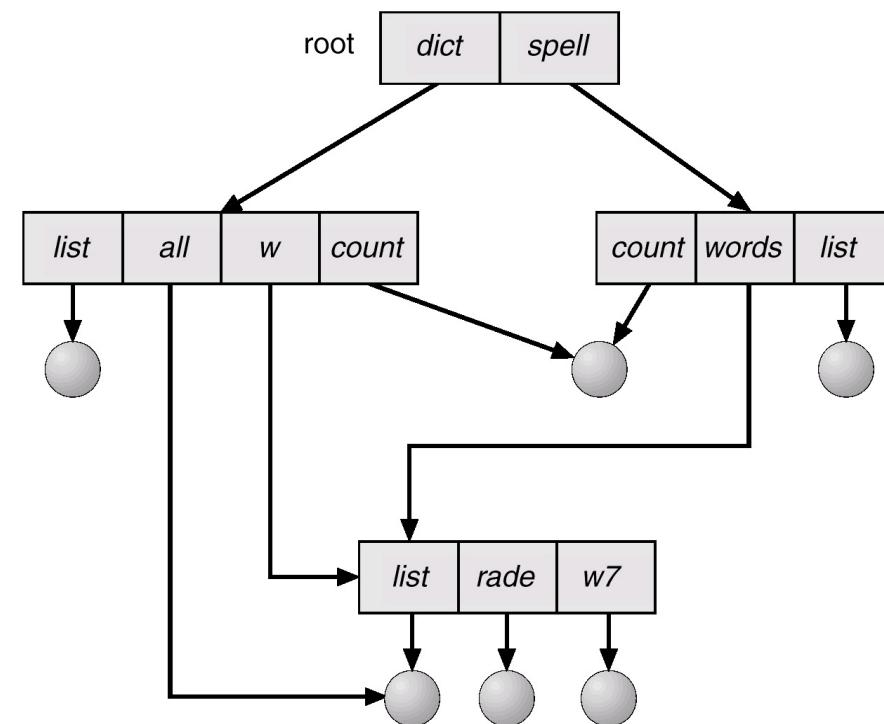
Directory ad albero

- Ricerca efficiente
- Possibilità di raggruppamento
- Concetto di directory corrente (working directory)
 - cd /spell/mail/prog
 - pwd
- Nomi di percorso assoluti o relativi



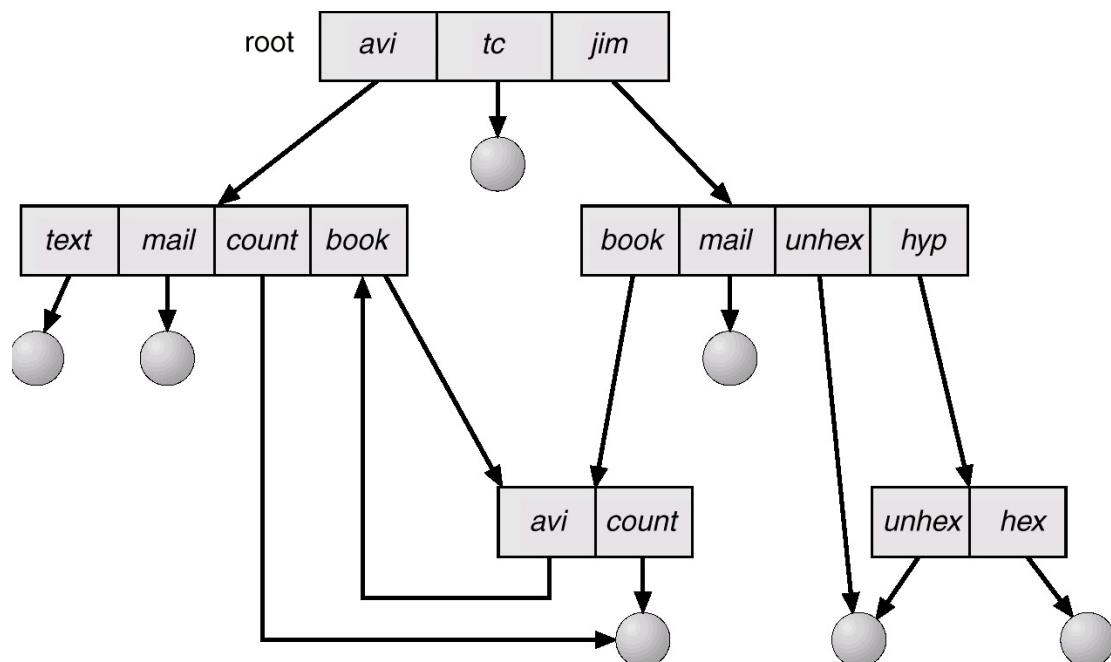
Directory ad grafo aciclico

- Struttura ad albero non permette condivisione di file e directory
- Implementazione della condivisione
 - Link simbolico
 - Contiene il pathname del file/directory reale
 - Se cancello il file reale il link rimane pendente
 - Unix: ln -s source destination
 - Hard link
 - Contatore che mantiene il # di riferimenti
 - Decrementato per ogni cancellazione di un riferimento
 - Cancellazione di file solo se il contatore vale 0
 - Unix: ln source destination



Directory ad grafo generico

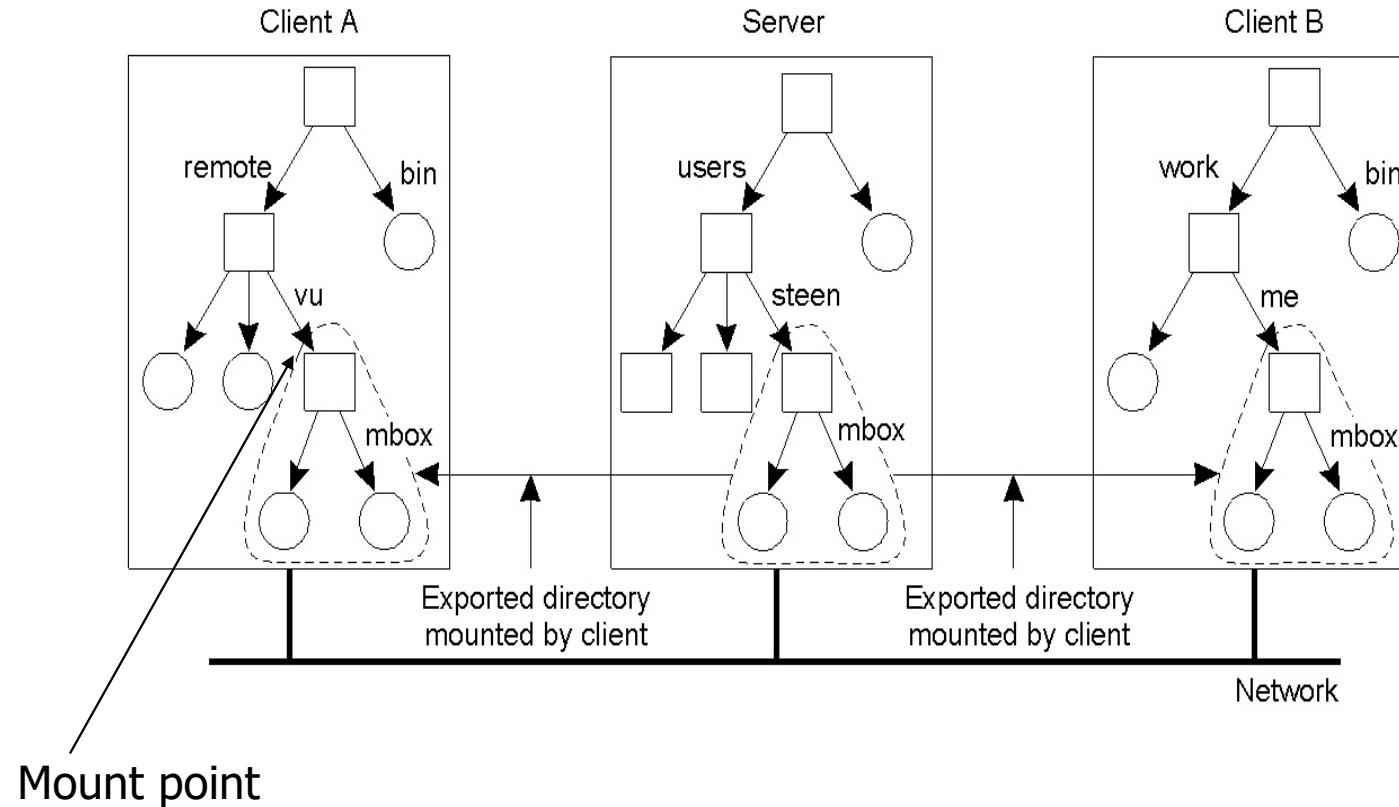
- Necessario garantire che non esistano cicli
 - Per evitare loop infiniti nell'attraversamento del grafo
- Soluzioni
 - Permettere di collegare solo file non directory
 - Usare garbage collection
 - Usare un algoritmo di controllo di esistenza di un ciclo ogni volta che si crea un nuovo collegamento



Mount di file system

- Realizzazione di file system modulari
- Possibilità di “attaccare” e “staccare” interi file system a file system preesistenti
- Mount/unmount = attaccare/staccare
- In generale un file system deve essere “montato” prima di potervi accedere
 - Punto in cui viene “montato” = mount point

Mount di file system – Esempio



Condivisione di file

- Condivisione importante in sistemi multiutente
- Realizzabile tramite uno schema di protezione
- In sistemi distribuiti, i file possono essere condivisi attraverso una rete

Protezione

- Il possessore di un file deve poter controllare
 - cosa è possibile fare su un file
 - da parte di chi
- Tipi di operazioni controllabili
 - Lettura
 - Scrittura
 - Esecuzione
 - Append (aggiunta in coda)
 - Cancellazione
 - ...

Protezione

Liste d'accesso per file/directory

- Elenco di chi può fare che cosa per ogni file/directory
- Può essere lungo

Utenti raggruppati in 3 classi (Unix)

- Proprietario
- Gruppo (utenti appartenenti allo stesso gruppo del proprietario)
- Altri

Protezione – Unix

- Per ogni classe i permessi possono essere
 - r (read)
 - w (write)
 - x (execute)
- Es.: chmod 754 nome_file
 - rwx r-x r--
 - tutto per proprietario, no write per gruppo, solo read per altri

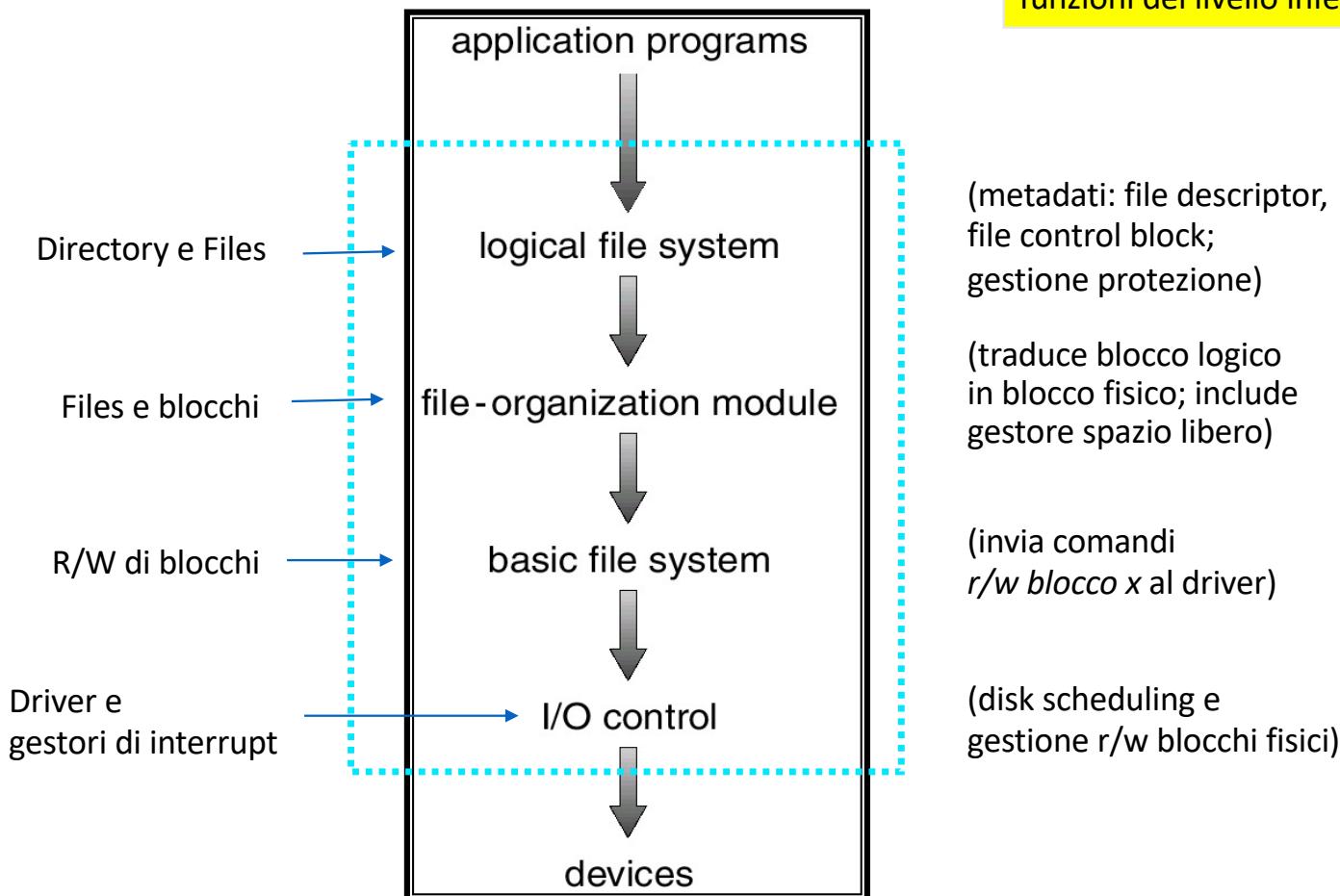
Implementazione del file system

Sommario

- Implementazione del file system
- Metodi allocazione dello spazio su disco
- Gestione dello spazio libero
- Efficienza e prestazioni
- Recupero delle informazioni

File system a livelli

Ogni livello usa
funzioni del livello inferiore



Implementazione del file system

- Necessarie varie strutture dati
 - Parte su disco
 - Parte in memoria
- Caratteristiche dipendono da SO e dal tipo di file system
 - Esistono caratteristiche comuni a tutte le tipologie

File system – Strutture su disco

- Blocco di boot
 - Informazioni necessarie per l'avviamento del S.O.
- Blocco di controllo delle partizioni
 - Dettagli riguardanti la partizione
 - Numero e dimensione dei blocchi, lista blocchi liberi, lista descrittori liberi, ...
- Strutture di directory
 - Descrivono l'organizzazione dei file
- Descrittori di file (inode)
 - Vari dettagli sui file e puntatori ai blocchi dati

File system – Strutture in memoria

- Tabella delle partizioni
 - Informazioni sulle partizioni montate
- Strutture di directory
 - Copia in memoria delle directory a cui si è fatto accesso di recente
- Tabella globale dei file aperti
 - Copie dei descrittori di file
- Tabella dei file aperti per ogni processo
 - Puntatore alla tabella precedente ed informazioni di accesso

Allocazione dello spazio su disco

Come i blocchi su disco sono allocati ai file (o alle directory)

Obiettivi

- Minimizzare tempi di accesso
- Massimizzare utilizzo dello spazio

Alternative

- Allocazione contigua
- Allocazione a lista concatenata (linked)
- Allocazione indicizzata

Allocazione contigua

Ogni file occupa un insieme
di blocchi contigui su disco

Vantaggi

- Entry della directory semplice
 - Contiene indirizzo blocco di partenza e numero di blocchi
- Accesso semplice
 - Accesso al blocco $b+1$ non richiede spostamento testina rispetto a b
 - A meno che b non sia l'ultimo blocco di un cilindro (raro)
 - Sia accesso sequenziale che casuale
 - Per leggere blocco i di un file che inizia al blocco b basta fare $b+i$

Svantaggi

- Problemi simili a quelli dell'allocazione dinamica della memoria
 - Algoritmi best-fit, first-fit, worst-fit
 - Frammentazione esterna
 - Richiede compattazione periodica dello spazio
- Se il file cresce?

Allocazione contigua – Crescita di un file

I file non
possono
crescere
dinamicamente

Quanto spazio allocare quando
viene creato un file?

Soluzioni

Se il file deve crescere e non
c'è spazio

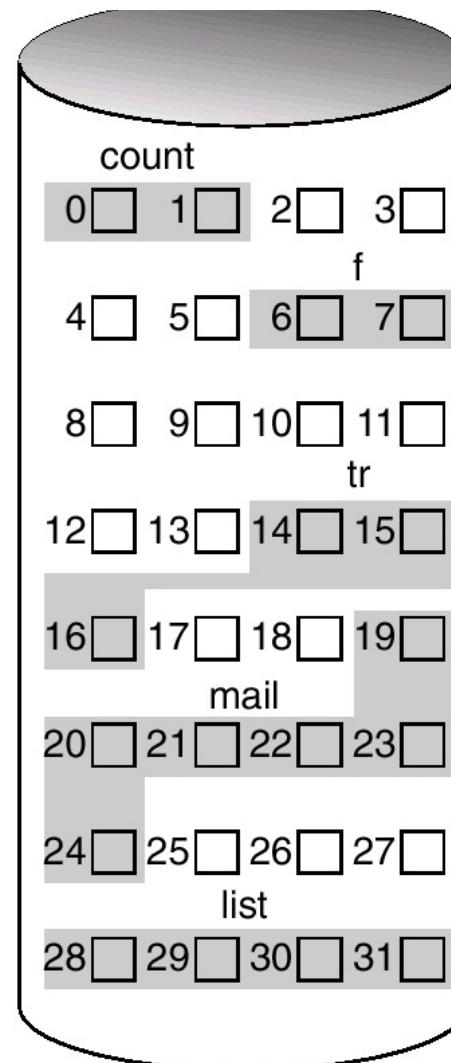
Errore e terminazione del programma → riesecuzione

Si tende a sovrastimare lo spazio necessario (spreco)

Trovare un buco + grande e
ricopiare tutto in quest'ultimo

Trasparente per l'utente, ma rallenta il sistema

Allocazione contigua



directory

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

Allocazione a lista

Ogni file è una lista di blocchi

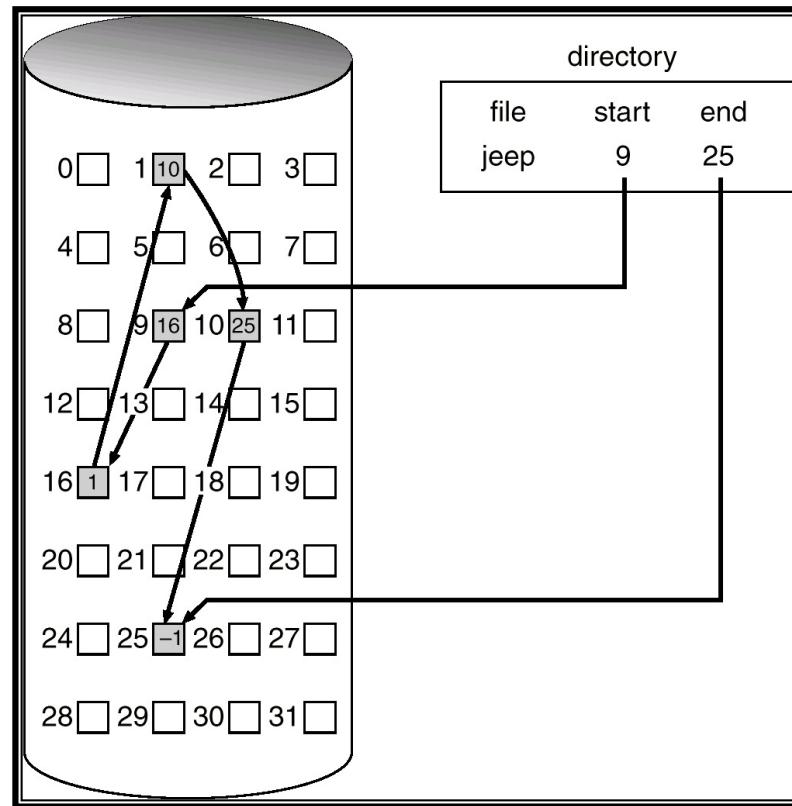
I blocchi possono essere sparsi ovunque nel disco

- Directory contiene puntatori al primo e all'ultimo blocco
- Ogni blocco contiene puntatore al blocco successivo

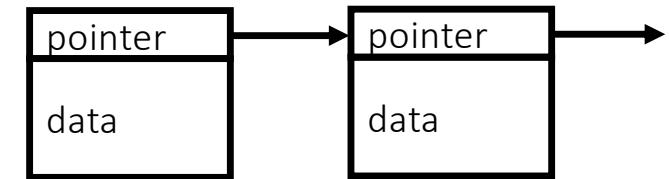
Indirizzamento

- X = indirizzo logico
- N = dimensione del blocco
 - $X / (N-1)$ = numero del blocco nella lista
 - $X \% (N-1)$ = offset all'interno del blocco

Allocazione a lista



blocco



Allocazione a lista

Vantaggi

Creazione nuovo file semplice

- Basta cercare un blocco libero e creare una nuova entry nella directory che punta al blocco

Estensione del file semplice

- Basta cercare un blocco libero e concatenarlo alla fine del file

Nessuno spreco (eccetto per il puntatore)

- Posso usare qualunque blocco
- No frammentazione esterna

Svantaggi

No accesso casuale

- Bisogna scorrere tutti i blocchi a partire dal primo

Scarsa efficienza causa riposizionamenti sparsi

Scarsa affidabilità

- Se si perde un puntatore, oppure se un errore (SW/HW) causa il prelevamento del puntatore sbagliato?
- Soluzioni (con overhead)
 - Liste doppiamente concatenate
 - Memorizzare nome file e n° di blocco in ogni blocco del file

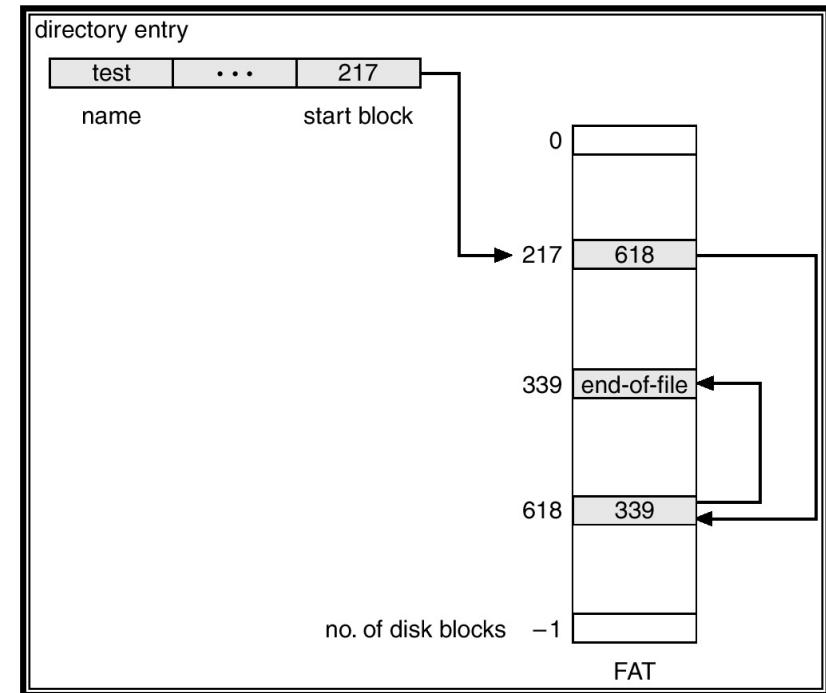
Esempio – FAT (File Allocation Table)

Tipica dei sistemi MS-DOS e OS/2

Una FAT per ogni partizione contiene un elemento per ogni blocco del disco

Usata come lista concatenata

Migliora accesso casuale, ma l'efficienza rimane scarsa



Allocazione contigua – Variante



Alcuni file system moderni usano uno schema modificato di allocazione contigua



Basato sul concetto di extent

Extent = serie di blocchi contigui su disco
(extent-based file system)

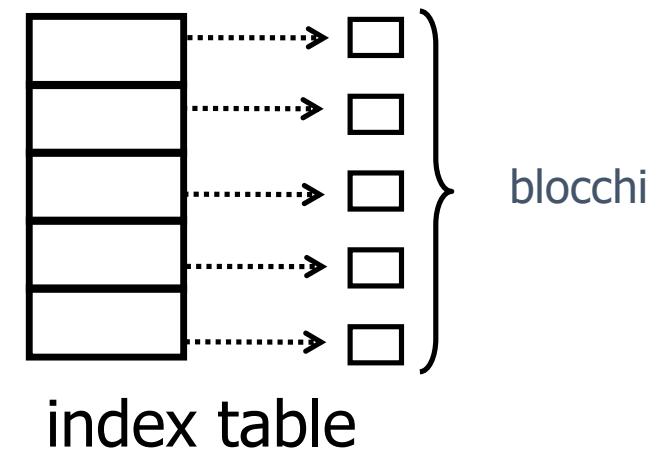


File system alloca extent anziché singoli blocchi

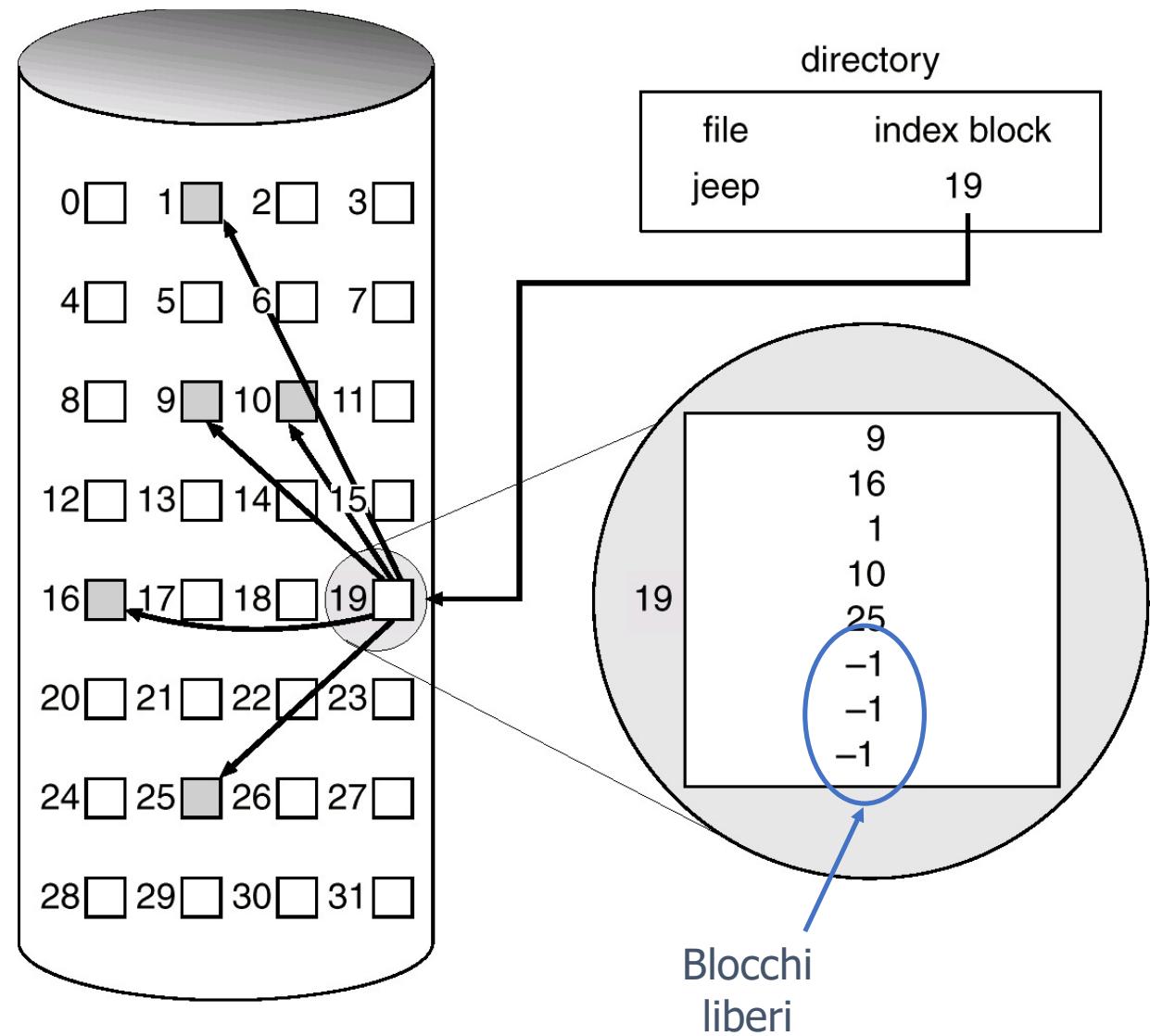
File = serie di extent
I vari extent non sono in generale contigui
• Adatto per allocazione a lista

Allocazione indicizzata

- Ogni file ha un *index block* contenente la *index table* (tabella indirizzi dei blocchi fisici)
- La directory contiene l'indirizzo del blocco indice



Allocazione indicizzata – Esempio



Allocazione indicizzata – Caratteristiche

Accesso casuale efficiente

Accesso dinamico senza frammentazione esterna

- ma con overhead del blocco indice per la index table
 - maggiore di quello richiesto per allocazione concatenata

Indirizzamento:

- X = indirizzo logico
- N = dimensione del blocco
 - X / N = offset nella index table
 - $X \% N$ = offset all'interno del blocco dati

Allocazione indicizzata – Limitazioni

La dimensione del blocco limita la dimensione del file!

- Es.: dimensione blocco = 512 parole \rightarrow massima dimensione del file = 512^2 parole = 256K parole

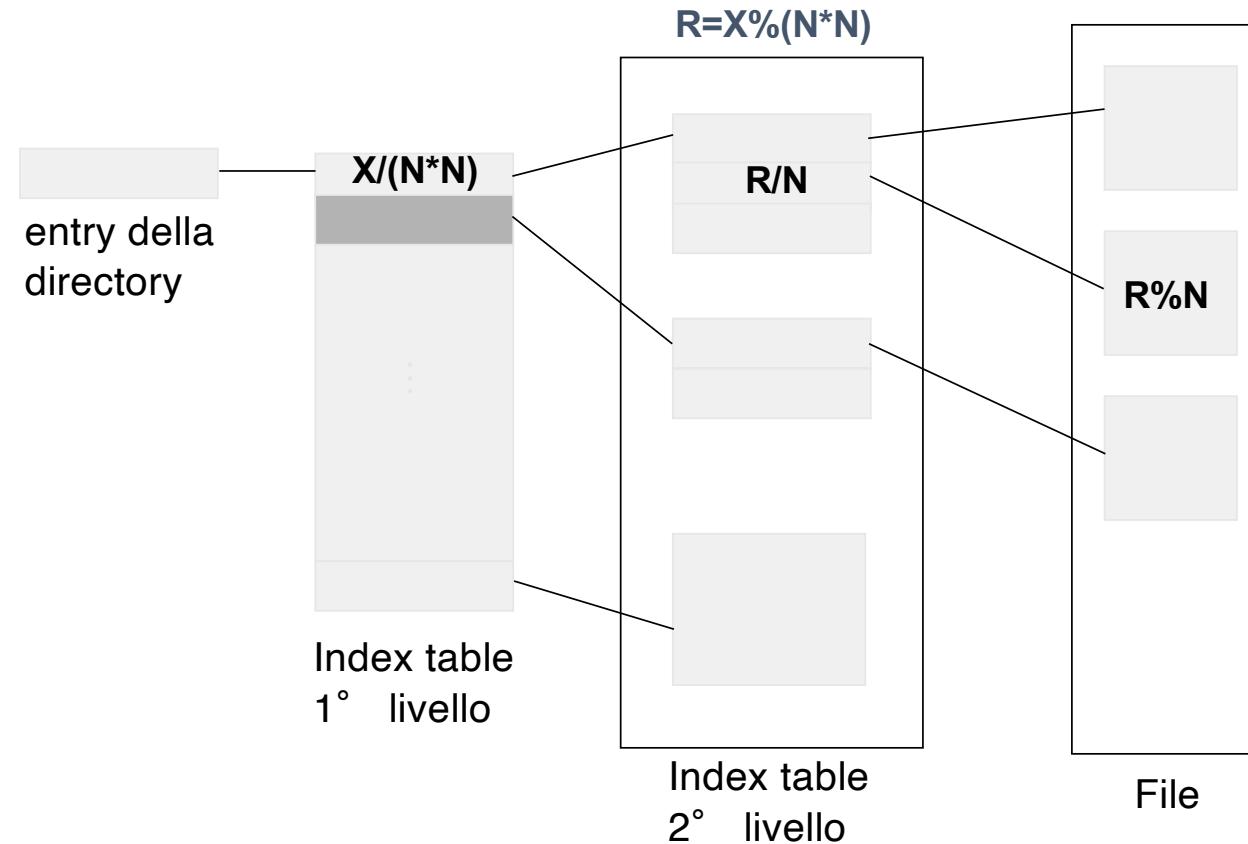
Per file di dimensione senza limiti si usa uno schema a più livelli

- Indici multilivello
- Schema concatenato
- Schema combinato (stile Unix BSD)

Indici multilivello

- Tabella più esterna contiene puntatori alle index table
 - X = indirizzo logico
 - N = dimensione del blocco (in parole)
 - $X / (N \times N)$ = blocco della index table di 1° livello
 - $X \% (N \times N) = R$
 - R usato come segue
 - R / N = offset nel blocco della index table di 2° livello
 - $R \% N$ = offset nel blocco dati
- Es.: blocchi da 4KB consentono 1K indici da 4 byte
→ due livelli di indici consentono file da 4GB

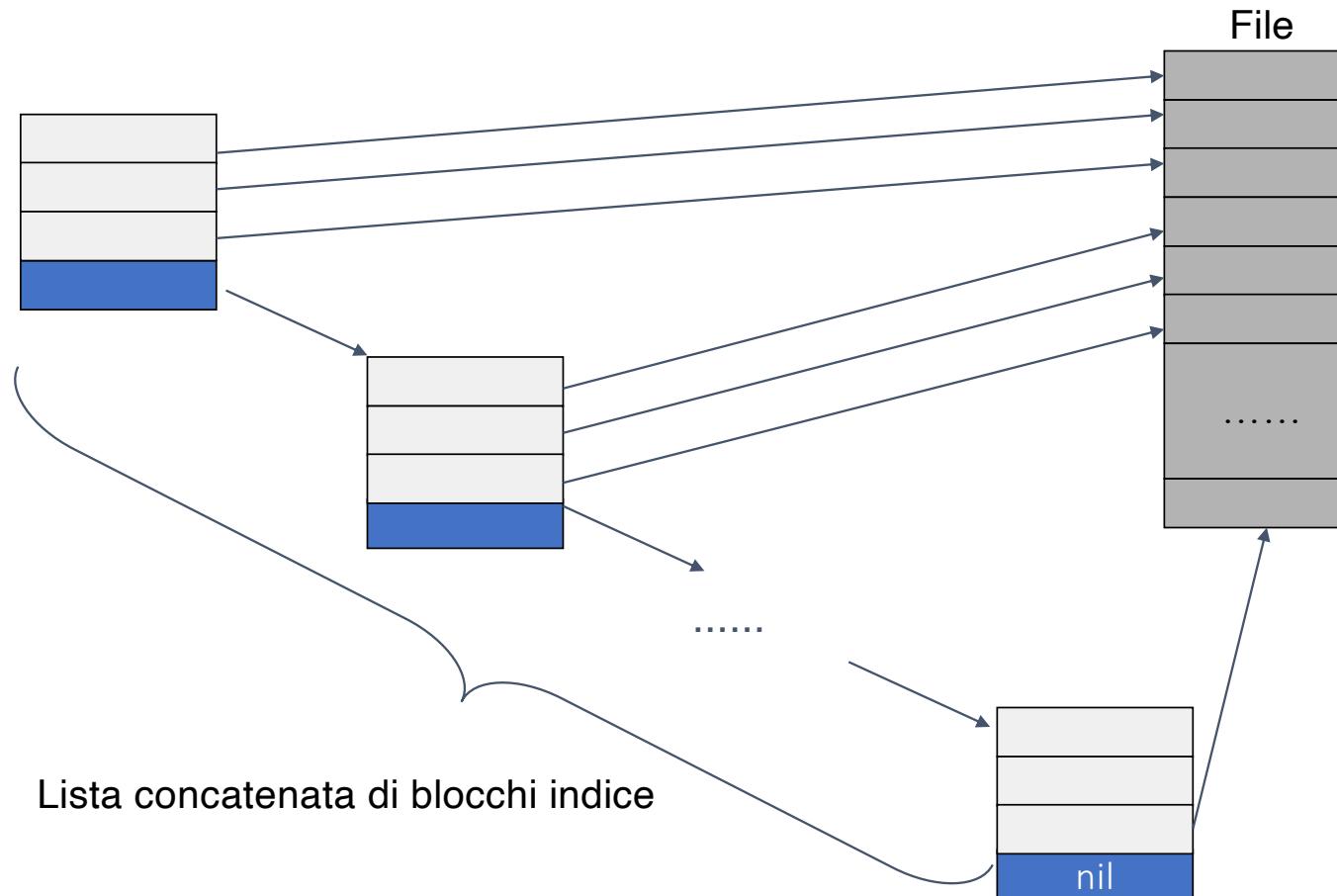
Indici multilivello



Schema concatenato

- Lista concatenata di blocchi indice
 - L'ultimo degli indici di un blocco indice punta a un'altro blocco indice
 - X = indirizzo logico
 - N = dimensione del blocco (in parole)
 - $X / (N(N-1))$ = numero del blocco indice all'interno della lista dei blocchi indice
 - $X \% (N(N-1)) = R$
 - R usato come segue
 - R / N = offset nel blocco indice
 - $R \% N$ = offset nel blocco dati

Schema concatenato

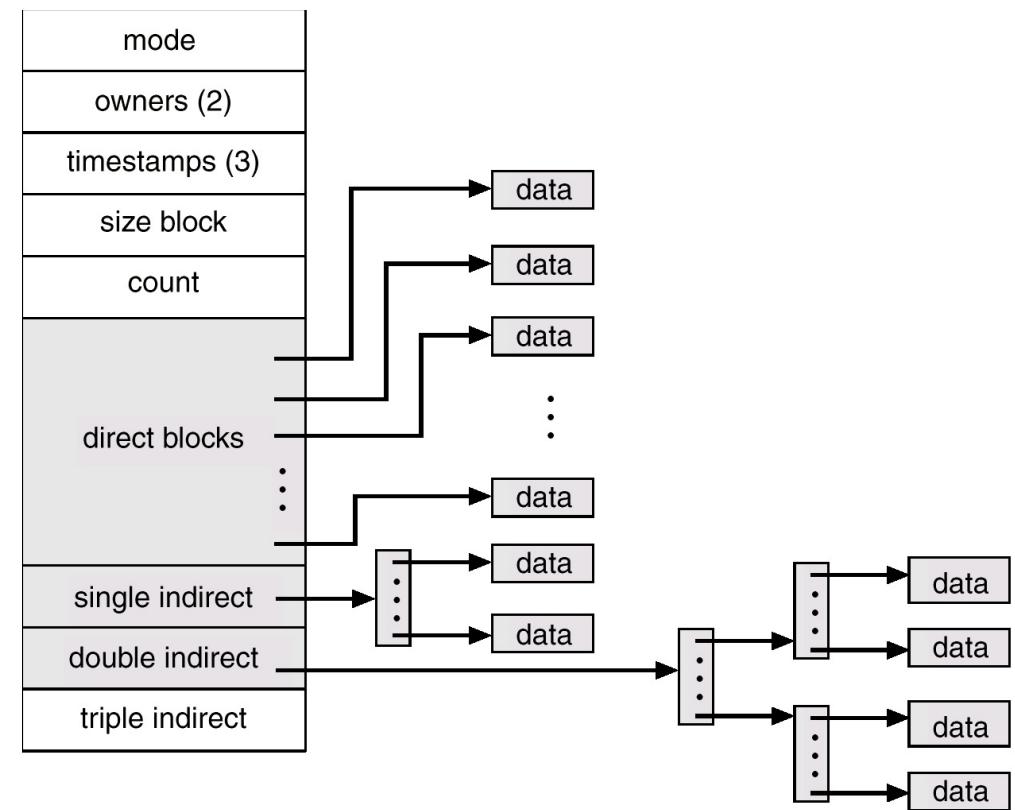


Schema concatenato – Esempio

- $N = 1KB = 256$ parole
 - 4 byte per parola
- Max file = 256 blocchi da 1KB = 256KB
 - $X = 12200$
 - $X/N = 12200/256 = 47$ (blocco 47)
 - $X \% N = 168$ (parola 168 all'interno del blocco 47)
 - $X = 644000$ Non basta un primo livello
 - $X / (N(N-1)) = 9$ (blocco della lista concatenata)
 - $X \% (N(N-1)) = R = 56480$
 - $R / N = 220$ (offset nel blocco index)
 - $R \% N = 160$ (offset nel blocco dati)

Allocazione indicizzata

- Schema combinato (Unix)
 - blocco di 4KB



Implementazione delle directory

- Stesso meccanismo usato per memorizzare file
- Le directory non contengono dati
 - Tipicamente la lista dei file (e delle directory) che essa contiene
- Problematiche:
 - Come questo contenuto viene memorizzato?
 - Come si accede al contenuto delle directory?

Implementazione delle directory

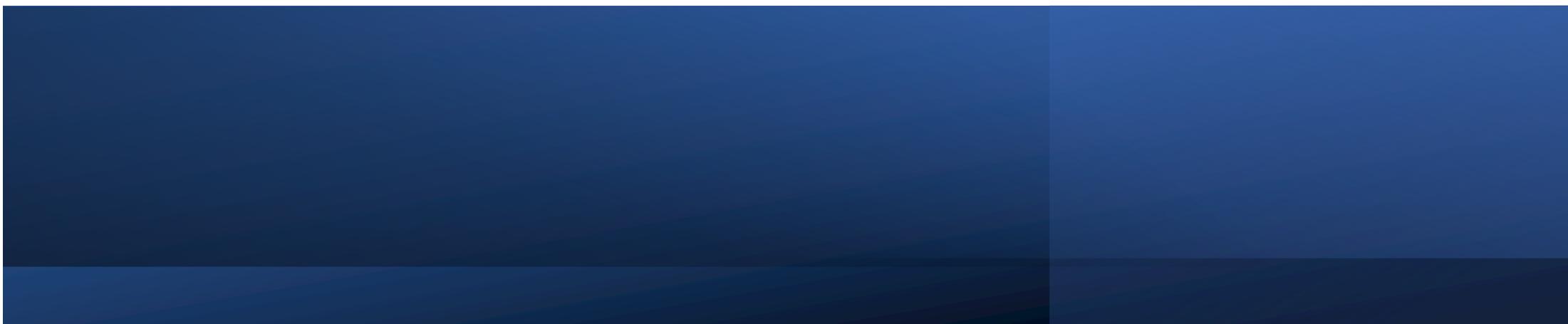
Lista lineare di nomi di file con puntatori ai blocchi dati

- Implementazione facile
- Poco efficiente
 - Lettura, scrittura e rimozione del file richiedono ricerca per trovare il file
 - Scansione lineare della lista (complessità $O(n)$)
 - Ricerca binaria su lista ordinata (complessità $O(\log n)$ ma bisogna ordinare la lista...)

Tabella hash

- Tempo di ricerca migliore
- Possibilità di collisioni
 - Situazioni in cui due nomi di file collidono sulla stessa posizione

Gestione dello spazio libero



Gestione dello spazio libero

Per tenere traccia dello spazio libero su disco si mantiene una lista dei blocchi liberi

- Per creare un file si cercano blocchi liberi nella lista
- Per rimuovere un file si aggiungono i suoi blocchi alla lista

Alternative

- Vettore di bit
- Lista concatenata
- Raggruppamento
- Conteggio

Vettore di bit

Vettore di bit, uno per blocco

- $\text{Bit}[i] = 1 \Rightarrow$ blocco i libero
- $\text{Bit}[i] = 0 \Rightarrow$ blocco i occupato

Esempio: n blocchi



Calcolo del numero del primo blocco libero

- Cerca la prima parola non 0
- (<# di bit per parola>) * (# di parole a 0) + (offset del primo bit a 1)
- Es: 00000000000000000000100001000011111000000000000

Vettore di bit

La mappa di bit richiede extra spazio

- Esempio:
 - $|\text{blocco}| = 2^{12} \text{ byte (4KB)}$
 - Dimensione del disco = $2^{38} \text{ byte (256 GB)}$
 - $n = 2^{38}/2^{12} = 2^{26} \text{ bit (8MB) solo per il vettore di bit}$

Efficiente solo se il vettore è mantenibile tutto in memoria

Facile ottenere file contigui

Alternative

Lista concatenata di blocchi liberi (free list)

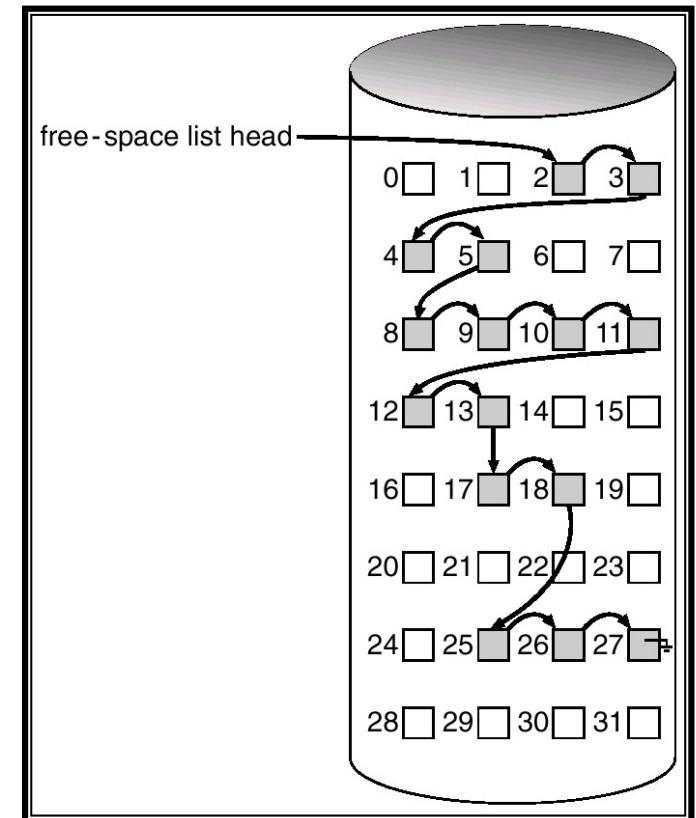
- Spreco minimo (solo per la testa della lista)
- Spazio contiguo non ottenibile

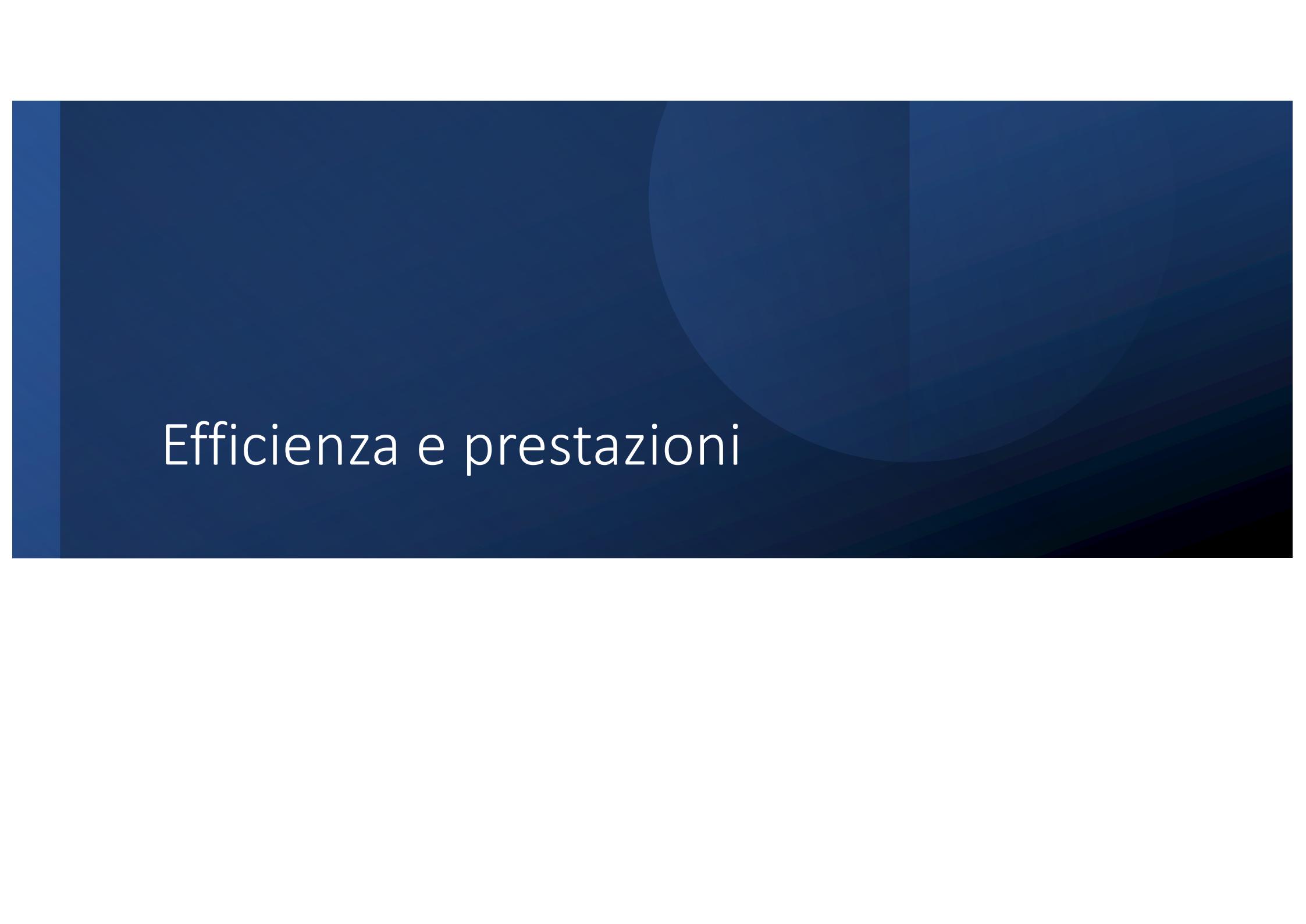
Raggruppamento

- Modifica della lista linkata (simile ad allocazione indicizzata)
- Primo blocco libero = indirizzi di n-1 blocchi liberi
- Ultima entry del blocco = indirizzo del primo blocco del gruppo successivo di n blocchi liberi
- Fornisce rapidamente un gran numero di blocchi liberi

Conteggio

- Mantiene il conteggio di quanti blocchi liberi seguono il primo in una zona di blocchi liberi contigui
- Generalmente la lista risulta + corta (se il contatore è > 1 per ogni gruppo di blocchi liberi)





Efficienza e prestazioni

Efficienza

Disco è collo di bottiglia

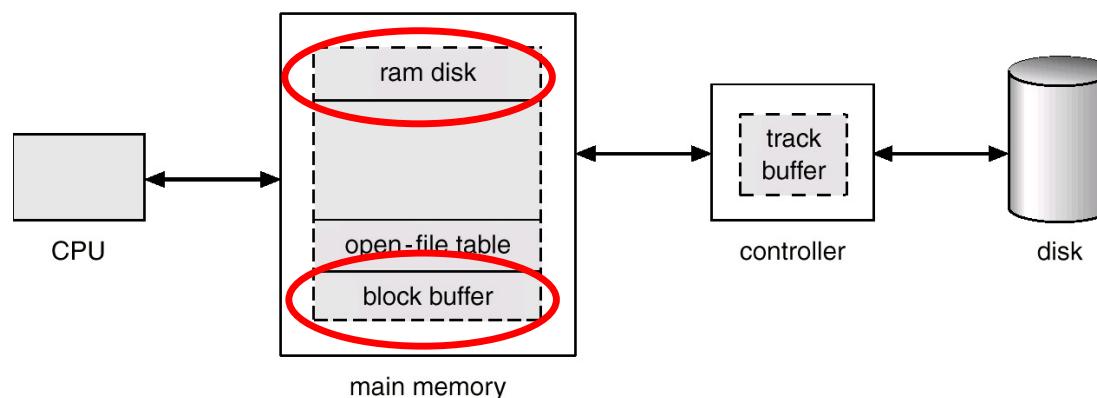
Efficienza dipende da

- Algoritmo di allocazione dello spazio su disco
 - Es.: in Unix si cerca di tenere i blocchi di un file vicini al suo i-node
 - Richiesta preallocazione i-node distribuiti sulla partizione
 - Tipo di dati contenuti nella directory
 - Es.: data di ultimo accesso di un file → lettura di un file richiede lettura e scrittura anche del blocco della directory

Prestazioni

Controller del disco possiede cache che contiene un'intera traccia ma non basta per garantire prestazioni elevate

Dischi virtuali (RAM disk)
Cache del disco (detta anche buffer cache)



Dischi virtuali

Parte della memoria gestita come disco

Driver RAM disk

- accetta operazioni standard dei dischi eseguendole in memoria
- gestito dall'utente che scrive sul RAM disk invece che sul disco vero e proprio (non è una cache)

Supporto solo per file temporanei

- Se spengo perdo tutto

Veloce

Cache del disco

Porzione di memoria che memorizza blocchi usati di frequente

- Simile alla cache tra memoria e CPU

Gestita dal SO

Sfrutta principio della località spazio temporale

Trasferimento dati nella memoria del processo non richiede spostamento di byte

Possibili problemi di consistenza tra disco e cache

File system log structured

Registrano ogni cambiamento del file system come una transazione

- Tutte le transazioni sono scritte su un log
- Una transazione è considerata avvenuta quando viene scritta sul log
- Le transazioni sul log sono scritte in modo asincrono nel file system
- Quando il file system è modificato, la transazione viene cancellata dal log
- Se il sistema va in crash, le transizioni non avvenute sono quelle presenti sul log
- Garantisce ottimizzazione del numero di seek