File System

Il File System

- Il SO utilizza i dispositivi di memoria secondaria per permettere agli utenti/processi di poter accedere a grandi quantità di dati persistenti, cioè che sopravvivono alla terminazione dei processi o alla mancanza di alimentazione elettrica
- Lato HW servono periferiche specifiche, come dischi e CD
- Lato SW occorrono astrazioni e meccanismi che consentano la rappresentazione, l'archiviazione e l'accesso ai dati immagazzinati in memoria secondaria, per evitare che gli utenti debbano lavorare a 'basso livello'
  - vedendo i dispositivi come sequenze lineari di blocchi di dati e
  - invocando operazioni di lettura e scrittura di blocchi di dati
- Il file system è quella componente del SO che organizza e gestisce i blocchi dei dispositivi di memoria sottostanti per fornire tali astrazioni e meccanismi
  - I processi e gli utenti vedono la memoria secondaria tramite la struttura astratta rappresentata dal file system

Il File System

- Realizza i concetti astratti di:
  - *file*: unità logica di memorizzazione persistente dei dati
  - *directory*: raggruppamento di file (e directory) strutturato gerarchicamente
  - *partizione/volume*: raggruppamento gerarchico di file e directory associato ad un particolare dispositivo fisico (es. disco) o ad una porzione di esso
- Le caratteristiche di file, directory e partizioni/volumi sono del tutto indipendenti dalla natura e dal tipo di dispositivo fisico utilizzato
- Il concetto di file astrae la memoria secondaria
  - così come il concetto di **processo/thread** astrae il processore e il concetto di **spazio di indirizzi** astrae la memoria principale

Il File System

- L'interfaccia del file system (punto di vista dell'utente del file system) è costituita dalle system call che permettono l'utilizzo della memoria secondaria da parte dei processi e degli utenti
  - Agli utenti interessa come sono identificati i file e le directory, le operazioni che si possono fare su di essi, come sono organizzate le directory e altre simili questioni di interfaccia
- La realizzazione del file system (punto di vista dell'implementatore del file system) è costituita dalle strutture dati e dagli algoritmi per la gestione della memoria secondaria da parte del SO
  - Agli implementatori interessa il modo in cui sono memorizzati file e directory, com'è gestito lo spazio su disco e come fare affinché tutto funzioni con efficienza e affidabilità

Tipi di file system

- Esistono svariati tipi di file system e la maggior parte dei SO ne supporta più di uno, anche contemporaneamente
- UNIX utilizza lo UNIX file system (UFS), che si fonda a sua volta sul Berkeley Fast File System (FFS)
- Windows supporta i file system FAT, FAT32 e NTFS (Windows NT file system)
- Sebbene Linux supporti oltre 40 file system diversi, il file system standard di Linux è noto come file system esteso, le cui versioni più recenti sono ext3 ed ext4
- Esistono anche file system distribuiti in cui un file system su un server è "montato" da uno o più computer client attraverso una rete (es. NFS - Network File System)
- Anche i dischi rimovibili hanno uno specifico file system: ad esempio, molti CD-ROM usano lo standard ISO 9660

Obiettivi

- Descrivere funzione e interfacce dei file system
- Presentare i metodi di accesso ai file
- Presentare la realizzazione dei file mappati in memoria
- Descrivere la struttura della directory
- Spiegare protezione e condivisione dei file
- Descrivere i dettagli dell'implementazione dei file system locali e della struttura della directory
- Presentare i metodi di allocazione dei blocchi e gli algoritmi per la gestione dello spazio libero
- Spiegare i problemi relativi all'efficienza e alle prestazioni del file system

Interfaccia del file system

- File
- Metodi di accesso
- File mappati in memoria
- Directory
- Protezione
- Condivisione

Interfaccia del file system

- File
- Metodi di accesso
- File mappati in memoria
- Directory
- Protezione
- Condivisione

File

- È un contenitore per un insieme di informazioni correlate, registrate in memoria secondaria, a cui è stato dato un nome
  - Astrae dalle caratteristiche fisiche delle unità di memoria del sistema
- *Punto di vista del SO*: è un'unità di memoria logica formata da un insieme di informazioni correlate
  - Il SO associa i file a dispositivi fisici, solitamente non volatili
  - Compito del SO è consentire di effettuare operazioni on-line sui file
- *Punto di vista dell'utente*: è la più piccola porzione logica di memoria secondaria
  - I dati si possono scrivere in memoria soltanto all'interno di file
- I file sono utili per i seguenti scopi:
  - memorizzare grandi quantità di dati
  - supportare la creazione di dati permanenti, capaci di sopravvivere ai processi che li hanno creati
  - fornire uno strumento semplice per permettere a più processi di condividere informazioni e/o comunicare (es. pipe)

Struttura di un file

- Un file è formato da una sequenza di record logici (bit, byte, righe, ecc.) il cui significato è definito dal creatore/utente
  - I record logici sono impaccati in blocchi fisici (che sono trattati in maniera unitaria dall'I/O sul dispositivo di memoria secondaria), tutti della stessa dimensione
- La struttura di un file dipende dal suo tipo e deve corrispondere alle aspettative del programma che dovrà manipolarlo
- Alcuni esempi di tipi di file e relativa struttura:
  - testo: sequenza di caratteri organizzati in righe e pagine, separati da *newline* e *pagebreak*
  - programma sorgente: sequenza di procedure e funzioni, ciascuna composta da dichiarazioni ed istruzioni
  - eseguibile: sequenza di sezioni di codice che il caricatore può trasferire in memoria principale

Attributi di un file (metadati)

- Nome: il nome simbolico di un file, serve per riferirlo
- Identificatore: etichetta unica che identifica il file all'interno del file system (è il nome che il sistema usa per il file)
- Tipo: necessario per sistemi che gestiscono tipi diversi di file; tre tecniche principali per identificare il tipo di un file
  - meccanismo delle estensioni (Windows)
  - attributo associato al file (es. programma creatore del file, in macOS)
  - *magic number* (in UNIX, sequenza di bit, normalmente posta all'inizio del file, che definisce il formato in cui i dati sono memorizzati)

Tipi di file: nome.estensione

Tipo di file	Estensione usuale	Funzione
Eseguibile	exe, com, bin, o nessuna	Programma, in linguaggio di macchina, eseguibile
Oggetto	obj, o	Compilato, in linguaggio di macchina, non collegato
Codice sorgente	c, cc, java, pas, asm, a	Codice sorgente in vari linguaggi di programmazione
Batch	bat, sh	Comandi all'interprete dei comandi
Testo	txt, doc	Testi, documenti
Elaboratore di testi	wp, tex, rtf, doc	Vari formati per elaboratori di testi
Libreria	lib, a, so, dll	Librerie di procedure per programmatori
Stampa o visualizzazione	ps, pdf, jpeg	File ASCII o binari in formato per stampa o visione
Archivio	arc, zip, tar	File contenenti più file tra loro correlati, talvolta compressi, per archiviazione o memorizzazione
Multimediali	mpeg, mov, rm, mp3, avi	File binari contenenti informazioni audio o A/V

Attributi di un file (metadati)

- Nome: il nome simbolico di un file, serve per riferirlo
- Identificatore: etichetta unica che identifica il file all'interno del file system (è il nome che il sistema usa per il file)
- Tipo: necessario per sistemi che gestiscono tipi diversi di file; tre tecniche principali per identificare il tipo di un file
- Locazione: puntatore al dispositivo di memoria secondaria ed alla posizione del file nel dispositivo
- Dimensione: dimensione corrente del file (in byte, parole o blocchi)
- Protezione: informazioni per il controllo delle operazioni sul file (es. lettura, scrittura, esecuzione)
- Ora, data: relative a creazione, ultima modifica o ultimo uso; utili per la protezione ed il controllo dell'utilizzo del file
- Proprietario: in un SO multiutente, indica l'utente proprietario

Descrittore di file

- I valori degli attributi di un file sono immagazzinati in una struttura dati del SO denominata descrittore di file (FCB, *file control block*)
  - In UNIX, i FCB sono detti inode
- Come il file, anche il suo descrittore dev'essere persistente ed è quindi mantenuto in memoria secondaria
- I descrittori dei file appartenenti ad una directory sono organizzati tramite la struttura della directory, una struttura dati del SO che risiede sullo stesso dispositivo dove sono memorizzati i file

Operazioni sui file

Compito del SO è consentire agli utenti di effettuare operazioni *online* sui file (cioè durante l'esecuzione delle applicazioni)

Tipicamente i file system forniscono perlomeno 6 operazioni di base

- Creazione: il file system (FS) reperisce spazio fisico per il file e crea l'elemento relativo al file nella directory
- Scrittura: dato il nome del file e le informazioni da scrivere nel file, il FS
  - effettua una ricerca nella directory per individuare la posizione del file nel dispositivo di memoria
  - scrive le informazioni nel file
  - mantiene ed aggiorna un puntatore alla locazione interna al file in cui deve avvenire la successiva operazione di scrittura/lettura
- Lettura: dato il nome del file e la posizione in memoria dove collocare le informazioni lette, il FS si comporta in maniera simile al caso della scrittura
- Riposizionamento (seek): il FS ricerca l'elemento corrispondente al file nella directory ed assegna un nuovo valore al puntatore interno al file
- Cancellazione: il FS ricerca ed elimina l'elemento corrispondente al file nella directory e, eventualmente, rilascia lo spazio fisico occupato dal file
- Troncamento: il FS ricerca l'elemento corrispondente al file nella directory ed azzera la lunghezza del file, rilasciando lo spazio fisico occupato dal file

Operazioni sui file

- Altre operazioni possono essere ottenute combinando le operazioni di base
  - Copia di un file
  - Ridenominazione di un file
- Sono inoltre necessarie operazioni che consentano agli utenti di leggere ed impostare gli attributi di un file, es.
  - per determinare la lunghezza di un file o la data di ultima modifica
  - per modificare il proprietario

Apertura e chiusura dei file

- La maggior parte delle operazioni sui file richiedono la ricerca nella struttura della directory (che sta su disco) dell'elemento associato al file
- Per migliorare l'efficienza, il SO
  - mantiene in memoria principale una tabella generale dei file aperti in cui vengono inserite informazioni relative a file che sono attualmente in uso
  - copia temporaneamente porzioni di file aperti in memoria principale
- Inoltre, nei sistemi in cui più processi possono aprire un file simultaneamente, che è la norma oramai, ogni processo ha anche una propria tabella dei file aperti
- Per predisporre tali strutture dati, la maggior parte dei SO richiede che il programmatore *apra* un file esplicitamente tramite la system call `open()` prima che il file possa essere usato e *chiuda* il file dopo aver terminato di usarlo tramite la system call `close()`
  - Altri SO *aprono* un file implicitamente quando viene fatto il primo riferimento ad esso e lo *chiudono* automaticamente quando termina il lavoro o il programma che l'ha aperto

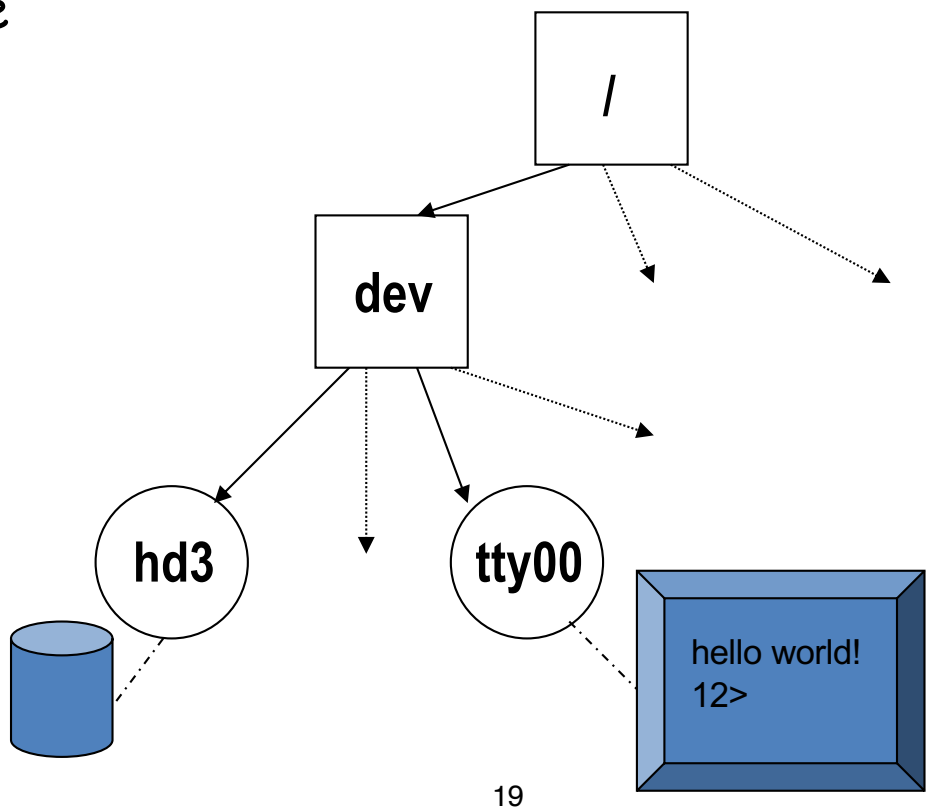
Apertura e chiusura dei file

Le informazioni contenute nelle tabelle dei file aperti sono gestite tramite due operazioni:

- `open()`, eseguita prima di ogni altra operazione sul file, inserisce nelle tabelle dei file aperti le informazioni sui file su cui il processo invocante vuole operare prelevandole dalla struttura della directory (in memoria secondaria)
- `close()`, eseguita come ultima operazione sul file, elimina dalle tabelle dei file aperti le informazioni sui file su cui il processo invocante ha terminato di operare copiando i metadati aggiornati del file nella struttura della directory (in memoria secondaria)

File speciali

- Alcuni SO usano file speciali che corrispondono a dispositivi di I/O
- Possono essere
  - a blocchi (usati per modellare i dischi) o
  - a caratteri (usati per modellare dispositivi di I/O seriali, quali terminali, stampanti, reti)
- Sono inclusi nella gerarchia delle directory come qualsiasi altro file, quindi permettono di fare riferimento ad un dispositivo come se fosse un file



Interfaccia del file system

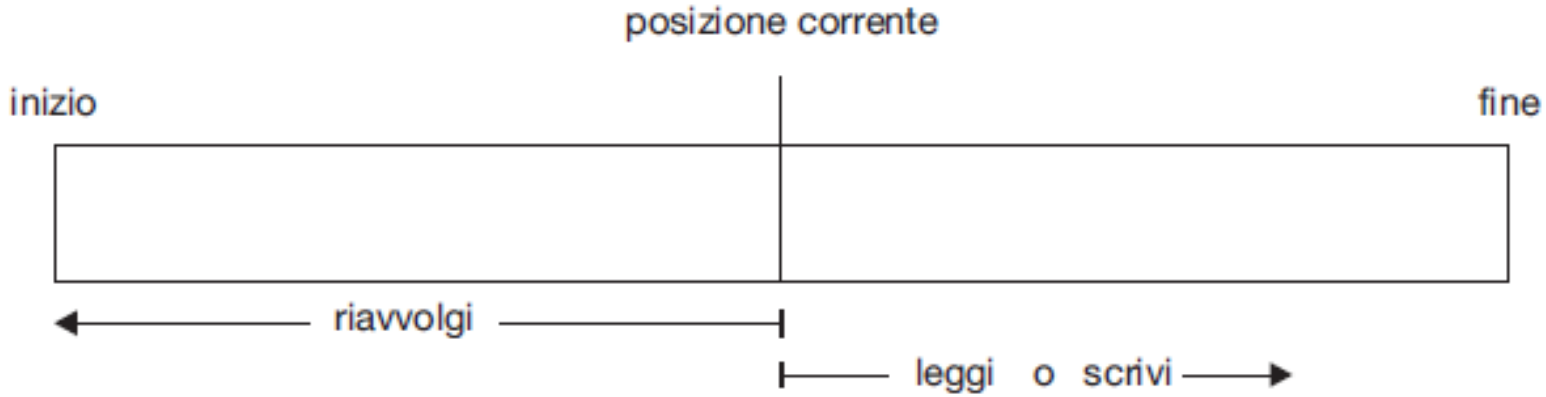
- File
- Metodi di accesso
- File mappati in memoria
- Directory
- Protezione
- Condivisione

Metodi di accesso

- Stabiliscono le modalità con le quali i processi possono accedere i file
- Ogni metodo di accesso presuppone una organizzazione interna dei file che a questo livello di astrazione sono visti come sequenze di record logici (unità di scrittura/lettura) numerati consecutivamente
- Sono indipendenti dal tipo di dispositivo e dalla tecnica di allocazione dei blocchi di memoria secondaria
- I più diffusi sono:
  - accesso sequenziale
  - accesso diretto
  - accesso tramite indice

Accesso sequenziale

- I record del file vengono acceduti sequenzialmente, uno dopo l'altro
- Il puntatore interno al file viene gestito automaticamente dalle operazioni di lettura/scrittura, `readnext(f,&V)` e `writenext(f,V)`, ma può essere modificato tramite `seek()`



- Modello che fa riferimento ai nastri magnetici e ai CD-ROM
- È il metodo di accesso più comune, usato per esempio da editor di testo e compilatori

Accesso diretto

- I record logici di cui è composto il file
  - sono numerati tramite un numero relativo `n` rispetto all'inizio del file e
  - sono accessibili indipendentemente e in un ordine qualsiasi
- Operazioni di lettura/scrittura: `readd(f,n,&V)` e `writed(f,n,V)`
- Non è necessario mantenere un puntatore interno al file perché gli accessi sono indipendenti l'uno dall'altro
- Modello che fa riferimento ai dischi

Accesso diretto

- Non tutti i SO gestiscono ambedue i tipi di accesso
  - Alcuni richiedono che si definisca il tipo di accesso al file al momento della sua creazione, così che poi si possa accederlo solo nel modo definito
- Questa tabella mostra come si possa facilmente simulare l'accesso sequenziale ad un file ad accesso diretto mantenendo una variabile `cp` che memorizza la posizione corrente

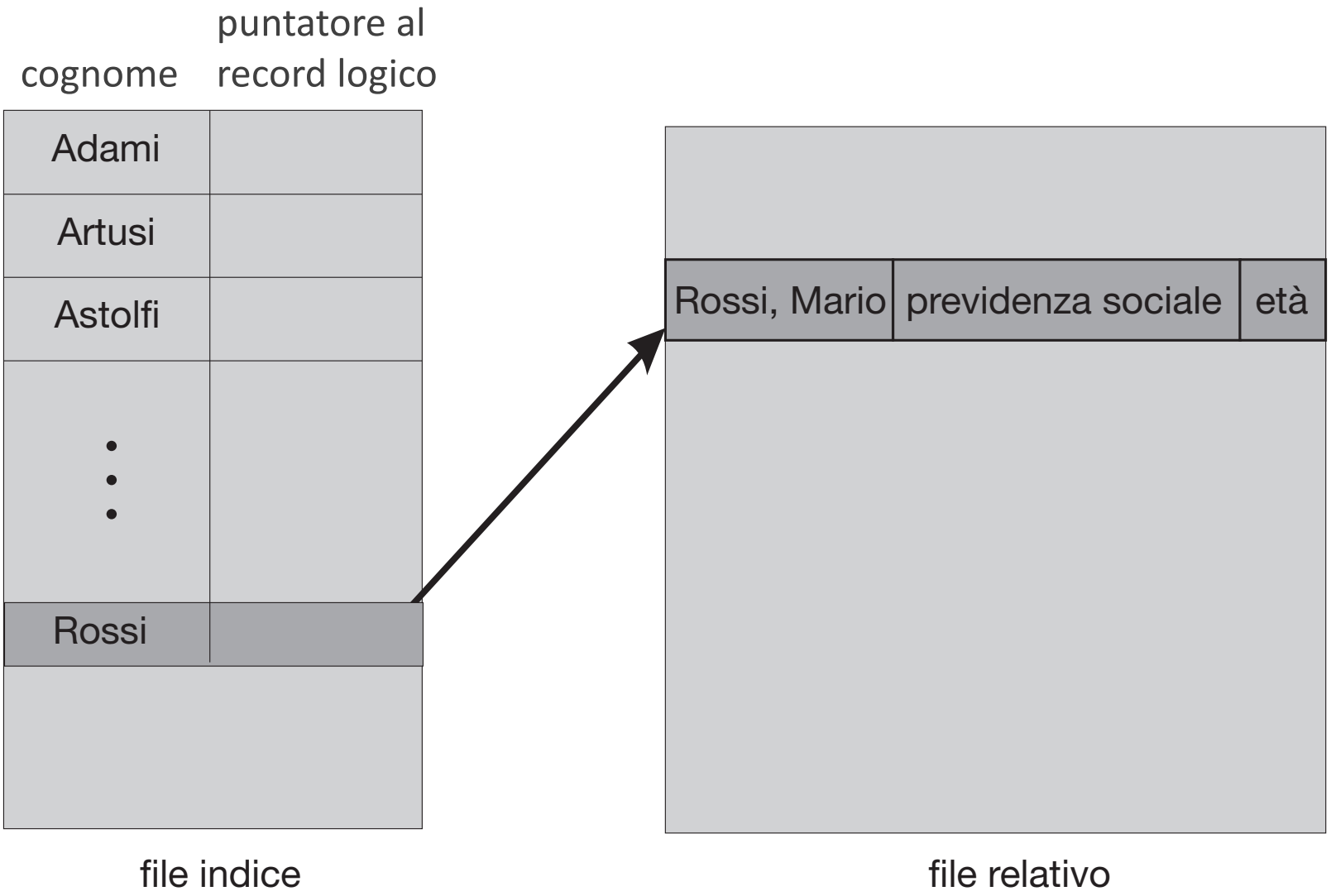
Accesso sequenziale	Realizzazione tramire Accesso diretto
`seek(init)`	`cp = 0  %cp = current position`
`readnext(f,&V)`	`readd(f,cp,&V);` `cp = cp+1;`
`writenext(f,V)`	`writed(f,cp,V);` `cp = cp+1;`

- Al contrario, è piuttosto macchinoso simulare l'accesso diretto ad un file che invece prevede l'accesso sequenziale

Accesso tramite indice

- Ogni record logico del file ha una chiave d'accesso
- Al file è associato un file indice il quale contiene associazioni tra chiavi e puntatori a record logici
- Per accedere un record, si effettua una ricerca nel file indice usando la chiave
  - Non è necessario mantenere un puntatore interno al file perché gli accessi sono indipendenti l'uno dall'altro
- Operazioni di lettura/scrittura:
  `readk(f,key,&V)` e `writek(f,key,V)`
- Vantaggio: la ricerca nel file indice può essere molto più veloce che nel file
  - Infatti il file può avere dimensioni molto maggiori del suo indice e richiedere un gran numero di operazioni di I/O
  - Inoltre l'indice può essere mantenuto in memoria principale (se troppo grande, si può indicizzare l'indice!)

Esempio di uso di indice



Interfaccia del file system

- File
- Metodi di accesso
- File mappati in memoria
- Directory
- Protezione
- Condivisione

File mappati in memoria

- Un altro metodo molto utilizzato per accedere i file sfrutta le tecniche di memoria virtuale per trattare l'I/O dei file dal disco come l'accesso ordinario alla memoria centrale
- Normalmente, un accesso ad un file richiede
  - l'invocazione di una system call (es. **`read()`**, **`write()`**)
  - l'uso di un buffer intermedio
  - un accesso al disco
- *Mappare un file in memoria* (in alcuni SO tramite una specifica system call `mmap(file,addr)`) significa creare una associazione tra il file (o una sua porzione) e una sezione dello spazio di indirizzamento virtuale del processo (si parla anche di *memoria mappata su file*)

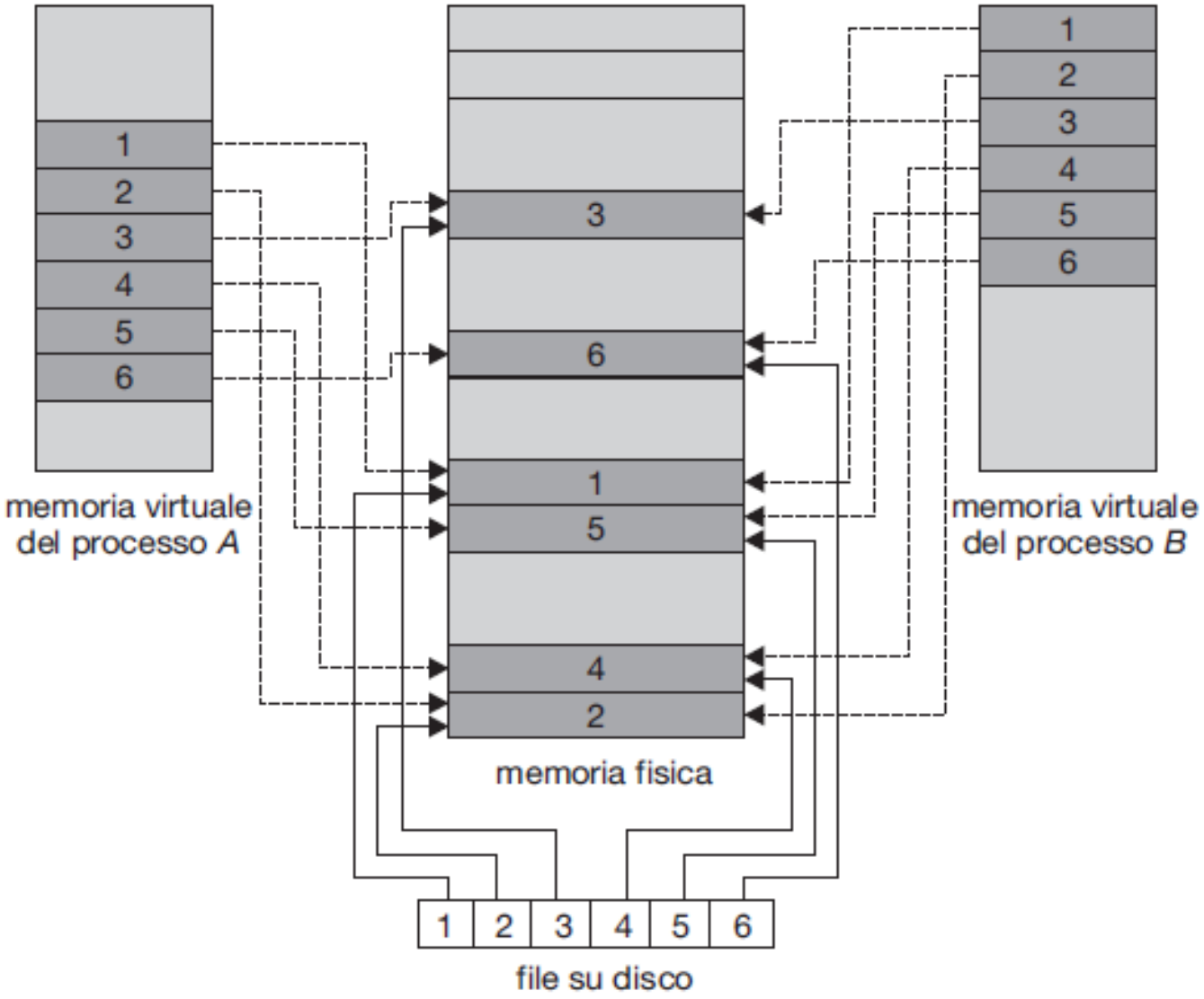
File mappati in memoria

- L'associazione indica al sistema di memoria virtuale che la copia su disco delle pagine virtuali su cui è mappato il file sono i corrispondenti blocchi del file su disco e non l'area di swap del processo
- Se ad esempio il SO implementa la memoria virtuale paginata:
  - l'accesso iniziale al file genera una interruzione page-fault
  - a seguito della quale una porzione del file delle dimensioni di una pagina viene copiata dal disco in una pagina fisica in memoria centrale
  - le successive letture e scritture sul file sono gestite come accessi ordinari alla memoria principale
- Il sistema di memoria virtuale riporterà sul file nel disco tutte le modifiche effettuate ad indirizzi di memoria virtuale associati al file
  - Anche se gli aggiornamenti in memoria non si traducono necessariamente in scritture immediate sul file nel disco

File mappati in memoria: vantaggi

- Maggiore efficienza delle operazioni di I/O:
  è possibile caricare in memoria solo le parti del file che sono effettivamente usate
- Notevole semplificazione delle operazioni di I/O:
  i dati da trasferire potranno essere acceduti direttamente nella sezione di memoria mappata (non sarà più necessario utilizzare buffer intermedi)
- Possibilità di condivisione: i processi possono mappare lo stesso file in modo concorrente, per consentire la condivisione dei dati e la comunicazione
  - Le scritture da parte di uno qualsiasi dei processi modificano i dati nella memoria virtuale e fisica e sono visibili a tutti gli altri processi che mappano la stessa sezione del file
  - Può anche essere supportata la funzionalità di copiatura su scrittura, consentendo così ai processi di condividere un file in modalità di sola lettura ma di avere una propria copia dei dati non appena li modificano

Condivisione di file tramite file mappato in memoria



Interfaccia del file system

- File
- Metodi di accesso
- File mappati in memoria
- Directory
- Protezione
- Condivisione

Directory (o cartella)

- Astrazione che consente di raggruppare più file (è paragonabile a un classificatore di documenti)
  - può contenere più file
  - può contenere a sua volta altre directory
- Anche alle directory, come ai file, è associato un descrittore che mantiene i valori degli attributi relativi
- Una directory si può considerare come una tabella di simboli (struttura della directory) che traduce nomi di file/directory nei corrispondenti elementi contenuti nella directory

Operazioni sulle directory

Da un punto di vista logico, la struttura della directory può essere organizzata in vari modi, ma deve permettere alcune operazioni fondamentali per la gestione del file system

- Ricerca di un file: scorrere la directory alla ricerca dell'elemento associato ad uno specifico file
- Creazione di un file: aggiungere nuovi elementi alla directory
- Cancellazione di un file: rimuovere elementi dalla directory
- Elencazione di una directory: elencare i file di una directory e il contenuto degli elementi della directory associati a ciascun file nell'elenco
- Ridenominazione di un file
- Attraversamento del file system: navigazione attraverso la struttura logica del file system per accedere a una specifica directory o ad uno specifico file

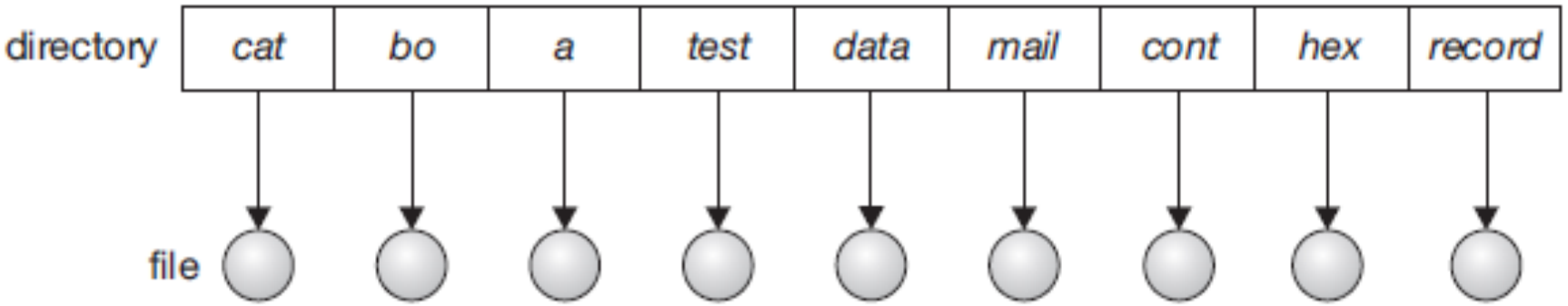
Scopi dell'organizzazione logica di una directory

- Efficienza nella ricerca dei file
- Flessibilità del meccanismo di *naming* dei file
  - Due utenti possono avere due file con lo stesso nome?
  - Un file può avere nomi diversi?
- Possibilità di raggruppare logicamente i file

  (es. tutti i documenti, tutti i programmi Java, tutti i giochi, …)

Directory a singolo livello

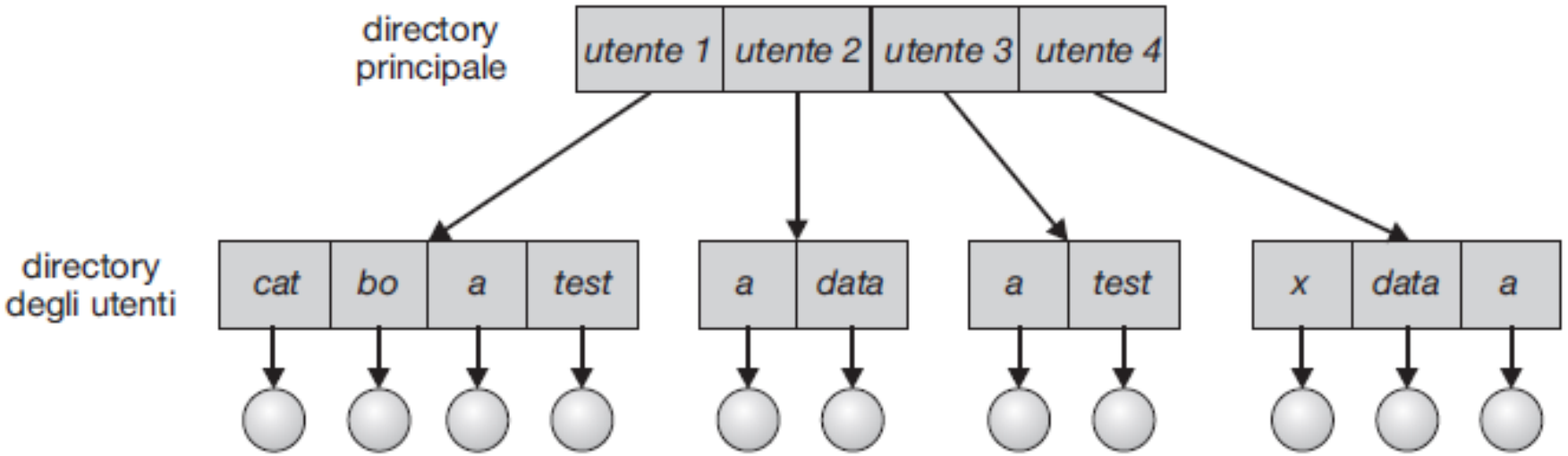
Approccio più semplice: una sola directory per tutti i file di tutti gli utenti



- Ricerca efficiente (se i file non sono molti)
- Naming: tutti i file devono avere nomi diversi
- Raggruppamento: non si possono raggruppare i file logicamente
- Limiti notevoli all'aumentare del numero dei file o nei sistemi multiutente

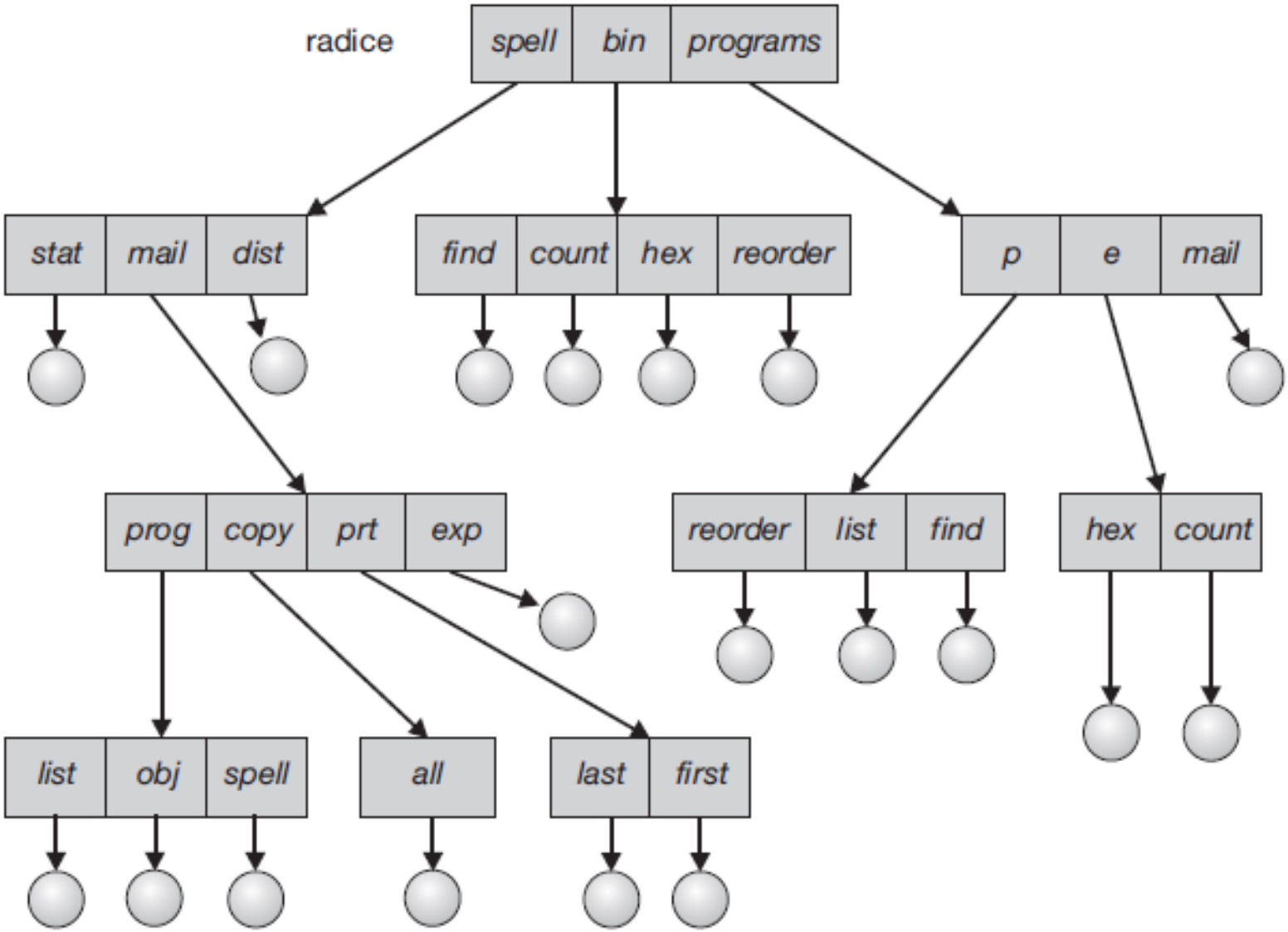
Directory a due livelli

Ogni utente ha una directory separata



- Necessità di specificare un *pathname* per individuare un file, corrispondente al percorso che si deve seguire per raggiungerlo
- Ricerca efficiente
- Naming: utenti diversi possono avere file con lo stesso nome
- Nessuna capacità di raggruppamento (a parte che a livello di utente)

Directory con struttura ad albero



Directory con struttura ad albero

- Ogni utente ha una directory di lavoro corrente (pwd, *present working directory*) a partire dalla quale viene fatta la ricerca dei file
- Ricerca efficiente
- Il nome di un file/directory è il suo *pathname*
  - Pathname assoluto (dalla radice)
  - Pathname relativo (dalla pwd)
- Se il primo carattere del pathname è il separatore (es. `/`) , allora il pathname è assoluto; altrimenti, è relativo alla pwd

  Per esempio, il comando UNIX

  ```
  cp /usr/ast/mailbox /usr/ast/mailbox.bak
  ```

  e il comando

  ```
  cp mailbox mailbox.bak
  ```

  hanno esattamente lo stesso effetto se la pwd è `/usr/ast`
- Ottime capacità di raggruppamento

Directory con struttura ad albero

- Le operazioni (es. creazione di un nuovo file) vengono eseguite relativamente alla directory corrente
- Cancellazione di un file

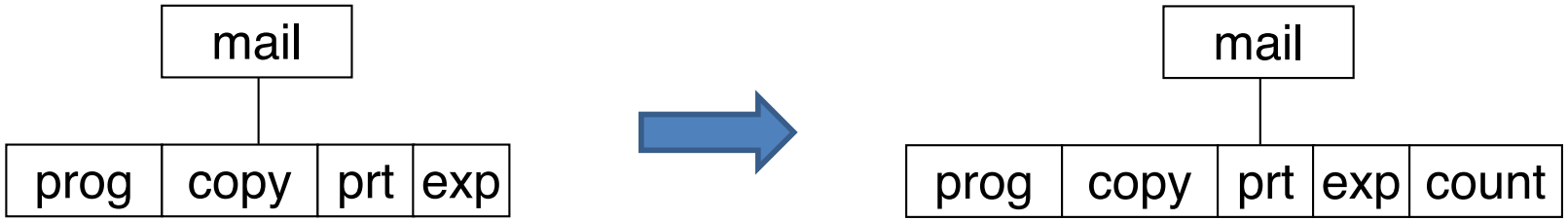
```
rm <nome-file>
```

- Creazione di una sottodirectory

```
mkdir <dir-name>
```

Esempio: se la directory corrente è `/mail` e si esegue

```
mkdir count
```



- Cancellazione della directory `mail`:
  - cancellazione dell'intero sottoalbero di cui `mail` è radice
  - oppure cancellazione della directory `mail` solo se vuota

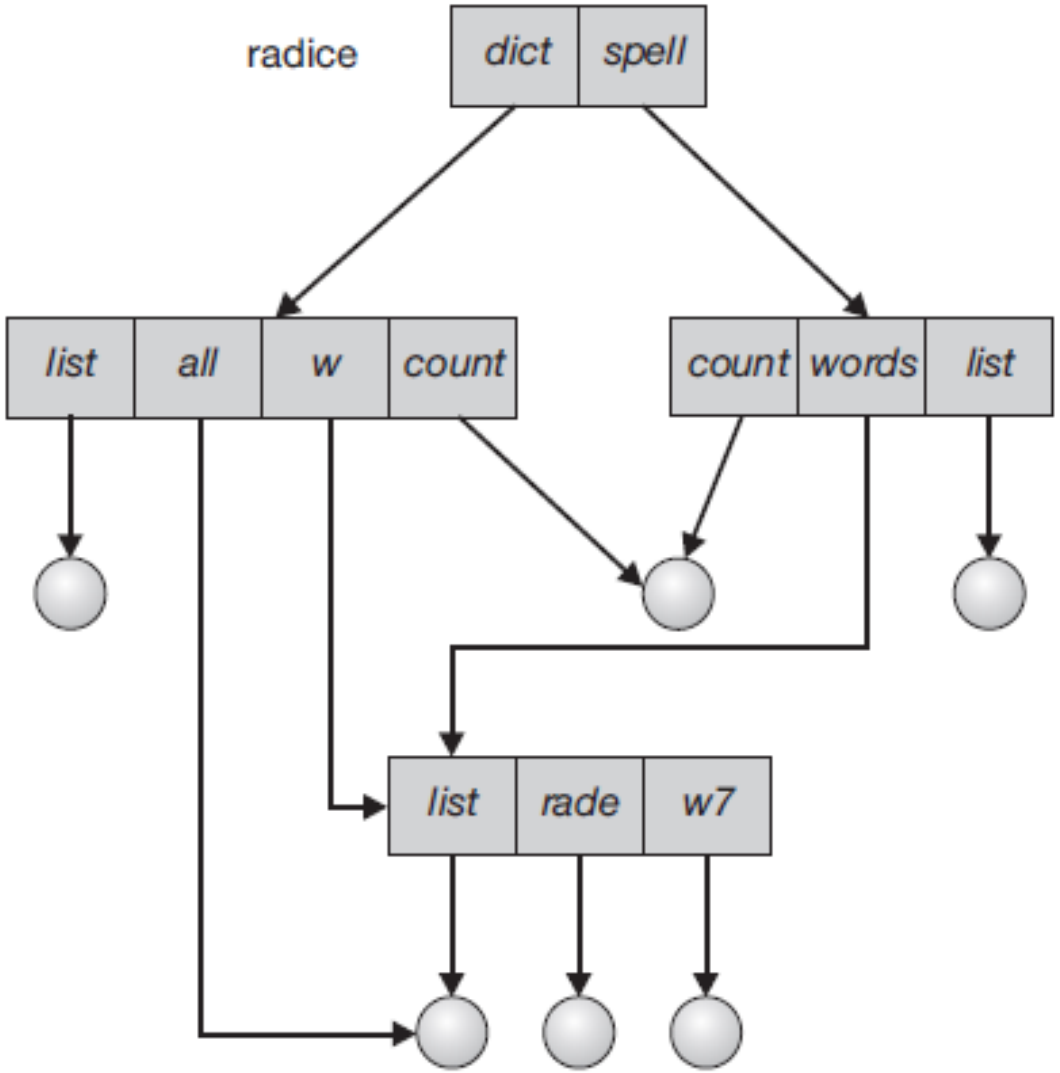
Directory con struttura a grafo aciclico (DAG)

- Rispetto alla struttura ad albero, può essere utile permettere l'aggiunta di "archi" per dare la possibilità di "vedere" lo stesso file da due o più directory diverse (cioè, dare più nomi ad uno stesso file)
  - Permette la condivisione di sottodirectory e file
  - Si possono avere due o più pathname assoluti (alias) per uno stesso file, ma ovviamente non possono esserci due file con lo stesso pathname
- Si potrebbe usare un grafo aciclico orientato (Directed Acyclic Graph, DAG)

Struttura ad albero vs. DAG

Struttura ad albero:
- ogni file è contenuto in un'unica directory

DAG
- un file può essere contenuto in una o più directory (es. `count`)
- esiste un'unica copia del file:
  - ogni modifica al file è visibile in tutte le directory che lo contengono



Condivisione e collegamenti/link

- La condivisione di file e sottodirectory si può realizzare in diversi modi
- Un modo comune è quello di creare un nuovo tipo di elemento di directory chiamato collegamento (in Windows) o link simbolico (in UNIX)
  - È in pratica un puntatore indiretto a un file o sottodirectory e si può realizzare come nome di percorso assoluto o relativo
- Quando viene fatto un riferimento a un file,
  - si effettua una ricerca nella directory
  - se l'elemento della directory individuato è contrassegnato come collegamento, il nome del file reale è incluso nell'elemento stesso e viene usato per localizzare il file (tale procedura è nota come risolvere il collegamento)

DAG: problemi legati all'aliasing

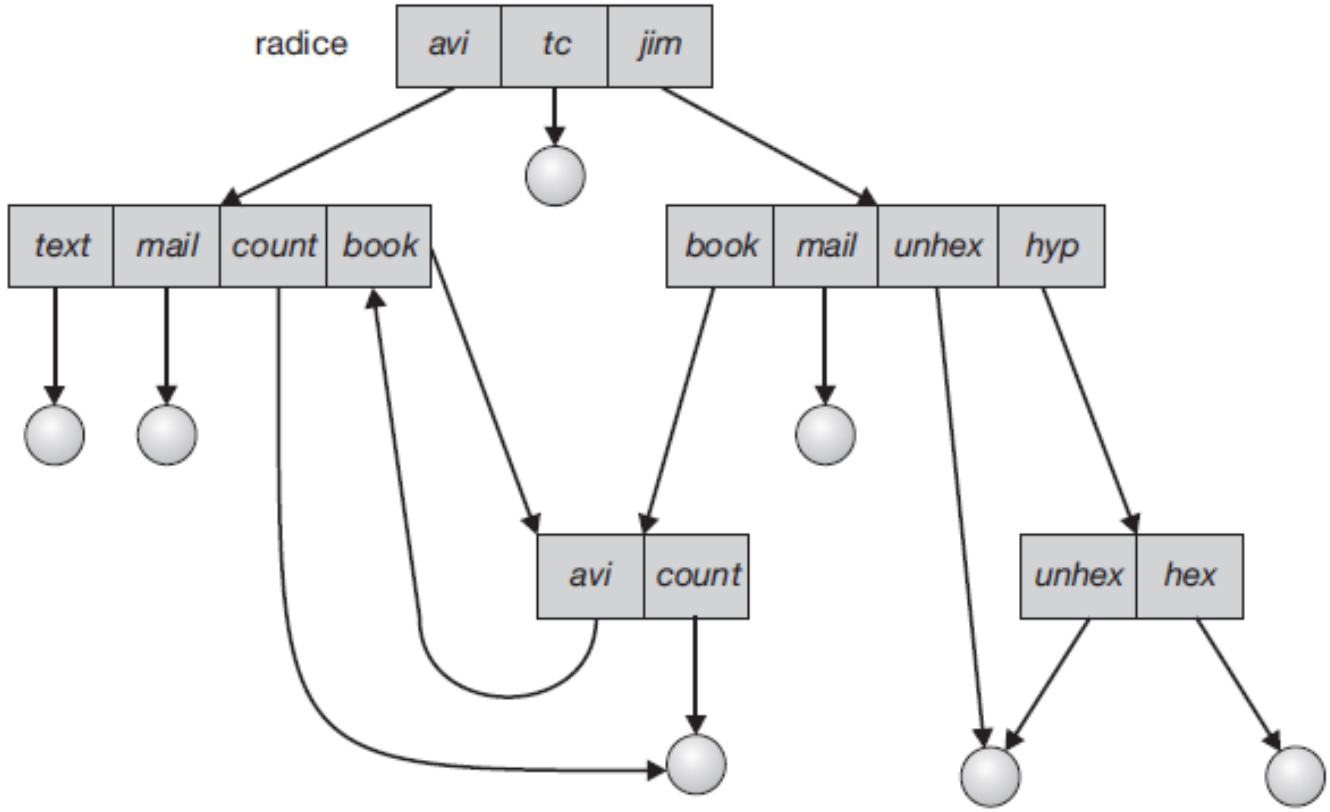
- Quando si esplora un file system non si dovrebbe prendere più volte in considerazione una stessa porzione condivisa (per motivi di *correttezza, efficienza, statistici*)
  - Soluzione: il SO ignora i collegamenti durante l'attraversamento delle directory
- Nel caso si cancelli un file con degli alias, quando si può recuperare la memoria assegnata al file?
  - Se lo si fa subito, si rischia di lasciare collegamenti 'pendenti'
  - Soluzioni:
    - Elencare i collegamenti, in modo da cancellarli tutti (inconveniente: l'elenco può richiedere molto spazio)
    - Usare un contatore dei riferimenti: recuperare la memoria quando il contatore è azzerato
- Per evitare problemi, alcuni sistemi semplicemente non consentono la condivisione di directory o l'uso di collegamenti

DAG: altri problemi

- Se in una directory con struttura a DAG si aggiungono collegamenti, per evitare la creazione di cicli (cosicché il grafo resti un DAG) si dovrebbe
  - consentire solo collegamenti a file e non a sottodirectory, oppure
  - utilizzare preventivamente un algoritmo di controllo dei cicli, che però è computazionalmente oneroso sopratutto quando il grafo si trova in memoria secondaria
- Per permettere la massima flessibilità nella condivisione dei file e delle directory, si può consentire la presenza di cicli ma poi bisogna gestire i problemi che ne conseguono

Directory con struttura a grafo generico

Se in una directory con struttura ad albero si aggiungono collegamenti in maniera arbitraria, tale struttura, in generale, si trasforma in un grafo generico, che può anche contenere cicli



Directory con struttura a grafo generico: problemi

La ricerca di un file/directory è complicata

- Bisogna implementare l'algoritmo di ricerca in modo da evitare di esplorare più volte una stessa porzione
  - Per esempio, limitando arbitrariamente il numero di directory cui accedere durante la ricerca
- Un algoritmo più semplice prevede di non percorrere mai i collegamenti durante l'attraversamento delle directory (come nel caso dei DAG)
  - Questo permette di evitare cicli nell'esplorazione senza alcun ulteriore costo computazionale

Directory con struttura a grafo generico: problemi

- Cancellazione di un file e recupero della memoria assegnata può richiedere di applicare un algoritmo di *garbage collection* per stabilire quando sia stato cancellato l'ultimo riferimento
  - Infatti, il contatore dei riferimenti ad un file potrebbe essere non nullo pur non essendo più possibile accedere il file
  - Questa anomalia è dovuta alla possibilità di autoreferenzialità (o ciclo) nella struttura delle directory
- La garbage collection richiede
  - un primo attraversamento dell'intero file system, per contrassegnare tutto ciò a cui è possibile accedere
  - un secondo attraversamento, per raccogliere tutto ciò che non è contrassegnato in un elenco di blocchi liberi
- La garbage collection per un file system basato su disco richiede però molto tempo e quindi viene effettuata di rado

Interfaccia del file system

- File
- Metodi di accesso
- File mappati in memoria
- Directory
- Protezione
- Condivisione

Protezione dei file

- La protezione riguarda l'uso illegale o le interferenze operate da utenti/programmi sotto il controllo del SO (cioè autenticati)
- I meccanismi di protezione svolgono un duplice ruolo:
  - rappresentazione delle politiche: realizzazione delle strutture dati che contengono la specifica dei vincoli di accesso alle risorse
  - controllo degli accessi: verifica che ogni accesso ad una risorsa da parte di un processo sia conforme alla politica di protezione specificata per la risorsa
- Il proprietario/creatore di un file dovrebbe essere in grado di controllare:
  - cosa può essere fatto
  - chi può farlo
- L'accesso ad un file/directory dovrebbe essere permesso o negato sulla base
  - dell'identificativo dell'utente (autenticazione) e
  - del tipo di accesso (autorizzazione)

Autenticazione

- Obiettivo: verificare la veridicità dell'identità degli utenti o dei processi
  - Normalmente un utente si identifica nel sistema presentando una credenziale, come un ID utente
  - L'autenticazione è la procedura con cui il sistema stabilisce la validità dell'identità dell'utente
- In genere, si basa su uno o più dei seguenti metodi:
  - Qualcosa che l'individuo conosce: password, PIN, risposte a domande prestabilite
  - Qualcosa (token) che l'individuo possiede: smartcard, chiave elettronica, chiave fisica
  - Qualcosa che l'individuo è (biometria statica): impronta digitale, retina, viso
  - Qualcosa che l'individuo fa (biometria dinamica): schema vocale, grafia, ritmo di battitura

Autenticazione

- Una volta completata l'autenticazione, l'identità dell'utente deve essere protetta da manomissioni, poiché altre parti del sistema si baseranno su di essa
- Dopo il login, l'id utente è associato a ogni processo eseguito con quel login:
  - l'id utente è memorizzato nel PCB
  - i processi figli ereditano l'id utente dal loro processo padre

Autorizzazione

- Obiettivo: determinare *quali* utenti possono eseguire *quali* operazioni su *quali* risorse
- Tipi di operazioni di base
  - Lettura
  - Scrittura
  - Esecuzione
  - Cancellazione
  - Aggiunta (di nuove informazione in coda al file)
  - Elencazione (degli attributi)
- Operazioni di livello superiore, come copia o ridenominazione, sono realizzate componendo le operazioni di base
  - È sufficiente quindi garantire la protezione a livello di operazioni di base

Matrice di protezione

Specifica le politiche di protezione dei file/directory (e, in generale, delle risorse)

	File A	File B	File C	Printer 1
Alice	RW	RW	RW	OK
Bob	R	R	RW	OK
Carol	RW			
David			RW	OK
Faculty	RW		RW	OK

- Ogni riga corrisponde a un dominio di protezione (utente) del sistema
- Ogni colonna rappresenta una risorsa del sistema

Rappresentazione delle politiche di protezione degli accessi

- L'uso della matrice di protezione per la specifica delle politiche è solo concettuale: in pratica comporterebbe costi elevati per via dello spreco di memoria
  - La matrice di protezione ha solitamente un numero di righe e colonne enorme ed è sparsa
- I due approcci più comuni per rappresentare concretamente le politiche di protezione degli accessi fanno uso delle seguenti strutture dati:
  - Liste di Capability (C-list)
  - Liste di Controllo degli Accessi (ACL)

Lista di capability (C-list)

- Ad ogni processo P il sistema associa una lista di capability, cioè una lista di file/directory (o risorse in generale) insieme con le operazioni permesse su quelle risorse
  - Rappresenta una riga della matrice di protezione
- Una risorsa è tipicamente rappresentata dal suo nome fisico o indirizzo

	File A	File B	File C	Printer 1
Alice	RW	RW	RW	OK
Bob	R	R	RW	OK
Carol	RW			
David			RW	OK
Faculty	RW		RW	OK

Capability List

Lista di controllo degli accessi (ACL)

- Ad ogni file/directory (o risorsa in generale) il sistema associa una ACL la quale, per ogni utente, specifica le operazioni che (i processi di) quell'utente è autorizzato a compiere sul file/directory
  - Rappresenta una colonna della matrice di protezione
- La lista può anche contenere un insieme di operazioni autorizzate di default

	File A	File B	File C	Printer 1
Alice	RW	RW	RW	OK
Bob	R	R	RW	OK
Carol	RW			
David			RW	OK
Faculty	RW		RW	OK

Access Control List

Liste di capability (C-list)

- Tipicamente le capability agiscono anche come nomi per le risorse: un utente/processo non può nemmeno nominare una risorsa che non è riferita da una capability nella sua C-list
  - Per eseguire l'operazione M sulla risorsa R, il processo invoca M specificando la capability per R come parametro
  - Il semplice possesso della capability indica che l'operazione è autorizzata
- Maggiore efficienza rispetto alla ACL nel controllo degli accessi
  - La C-list fa parte delle informazioni locali ad ogni processo, pertanto la ricerca dei diritti è effettuata localmente al processo
- Operazioni quali la revoca di tutti i diritti di accesso associati ad una risorsa sono costose poiché richiedono la ricerca e l'aggiornamento di tutte le C-list

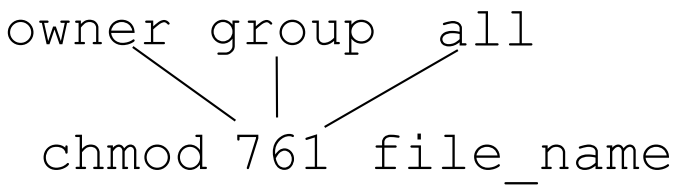
Liste di controllo degli accessi (ACL)

- Meccanismo flessibile
- Problematico gestire liste di utenti del sistema non note o che cambiano dinamicamente
- Il descrittore di file/directory dev'essere di dimensione variabile (a meno che non si usi una soluzione tipo quella introdotta da UNIX)
- Molti sistemi usano una versione condensata delle ACL: UNIX
  - 3 classi di utenti
    - Proprietario: l'utente che ha creato il file
    - Gruppo: un gruppo di utenti che condividono il file e necessitano di un accesso simile
    - Universo: tutti gli altri utenti del sistema
  - 3 modalità di accesso: lettura, scrittura, esecuzione

UNIX: 3 classi di utenti, 3 modalità di accesso

- L'amministratore del sistema crea utenti e gruppi di utenti e può cambiare il proprietario (`chown utente file`)
- Il proprietario può cambiare diritti dei file (`chmod 761 file`), per se, per il gruppo e per gli altri utenti e può cambiare il gruppo di un file (`chgrp grp file`)
- I diritti possono essere codificati tramite 3 cifre ottali: es. 761

			RWX
a) utente proprietario	7	⇒	1 1 1
b) utenti del gruppo	6	⇒	1 1 0
c) tutti gli altri utenti	1	⇒	0 0 1



Interfaccia del file system

- File
- Metodi di accesso
- File mappati in memoria
- Directory
- Protezione
- Condivisione

Condivisione di file

- La condivisione richiede meccanismi di protezione (es. controllo degli accessi) e l'uso di specifici attributi dei file/directory
- La maggior parte dei sistemi si basa sui concetti di
  - proprietario: è l'utente che ha il massimo controllo sul file: può modificare gli attributi e definire l'esatto insieme di operazioni che ciascun altro utente è autorizzato a compiere sul file
  - gruppo: un sottoinsieme di utenti autorizzati a condividere l'accesso al file
- Gli ID del proprietario e del gruppo di un determinato file (o directory) sono memorizzati insieme con gli altri attributi del file
  - Quando un utente richiede di compiere un'operazione su un file, per determinare se l'utente richiedente è il proprietario del file o se appartiene al gruppo del file, l'ID dell'utente è confrontato con l'attributo che identifica il proprietario o il gruppo
  - Il risultato indica i permessi applicabili, che il sistema quindi considera per consentire o impedire l'operazione richiesta

Condivisione di file

- Nei sistemi con più file system locali, la verifica dell'ID e dei permessi risultanti si può fare facilmente, una volta 'montati' i file system
- Nel caso invece di un disco portatile che può essere spostato tra sistemi (e file system) differenti, bisogna fare attenzione per assicurarsi che gli ID dei vari sistemi corrispondano oppure che la proprietà dei file venga resettata
  - Es. si può creare un nuovo ID utente e impostare la proprietà di tutti i file sul disco portatile a quell'ID, così da essere sicuri che nessun file sia accidentalmente accessibile agli utenti esistenti
- Nei sistemi distribuiti i file possono essere condivisi tramite la rete di comunicazione sottostante
  - Network File System (NFS): è un protocollo di rete, sviluppato da Sun Microsystems (1984), che consente ai computer di utilizzare la rete per accedere ai dischi rigidi remoti come fossero locali
  - NFS è spesso associato a sistemi Unix, sebbene sia utilizzato anche da macchine equipaggiate con SO Microsoft Windows
  - Il termine "network file system" oramai viene utilizzato come termine generico per indicare un file system in grado di gestire dispositivi di memorizzazione remoti

Realizzazione del file system

- Struttura del file system
- Operazioni del file system
- Realizzazione delle directory
- Montaggio di un file system
- Metodi di allocazione
- Gestione dello spazio libero
- Efficienza e prestazioni

Realizzazione del file system

- Struttura del file system
- Operazioni del file system
- Realizzazione delle directory
- Montaggio di un file system
- Metodi di allocazione
- Gestione dello spazio libero
- Efficienza e prestazioni

Struttura del file system

- Come qualsiasi SW complesso, il file system deve essere strutturato opportunamente per facilitarne lo sviluppo, la manutenzione, ed il riuso
- Il file system è solitamente organizzato in livelli:

  ogni livello si serve delle funzioni dei livelli inferiori per crearne di nuove impiegate ai livelli superiori
- Uno dei motivi per strutturare in questo modo il file system riguarda la separazione dei livelli che dipendono dall'HW da quelli che sono indipendenti dall'HW
- La stratificazione è anche utile per ridurre la complessità e la ridondanza del codice, ma aggiunge overhead al SO e può causare un decadimento delle prestazioni

File System: livelli di astrazione

Solitamente si possono distinguere 4 livelli

Controllo dell'I/O

- Costituito dai driver dei dispositivi e dai gestori delle interruzioni
- Si occupa del trasferimento delle informazioni tra memoria centrale e memoria secondaria, un blocco per volta
- Si appoggia direttamente sull'HW e presenta una vista astratta del dispositivo, che appare come vettore lineare di blocchi fisici di uguale dimensione
- Un driver trasforma comandi di alto livello, es. "Recupera il blocco X", in istruzioni di basso livello dipendenti dall'HW per il controllore del dispositivo di I/O che funge da interfaccia tra il dispositivo ed il resto del sistema

File system di base

- Invia comandi all'apposito driver di dispositivo per leggere e scrivere blocchi fisici sul disco
- Gestisce
  - i buffer di memoria per il trasferimento dei dati
  - le cache per i metadati del file system usati più di frequente per migliorare le prestazioni

Modulo di organizzazione dei file

- Conosce il metodo di allocazione dei file usato, così come la locazione dei file nei blocchi fisici del disco
- Traduce indirizzi di record logici in indirizzi di blocchi fisici che poi il "File system di base" si occupa di far trasferire
  - In un disco magnetico, ogni blocco fisico si identifica col suo indirizzo numerico: es. unità 1, cilindro 73, traccia 2, settore 10
- Gestisce la lista dei blocchi fisici liberi

File system logico

- Fornisce ai processi ed agli utenti una visione astratta delle informazioni presenti su disco, basata su file, directory, partizioni, volumi, ecc., che prescinde
  - dalle caratteristiche del dispositivo e
  - dalle tecniche di allocazione e di accesso alle informazioni adottate dal sistema
- Realizza le operazioni di gestione di file e directory (creazione, cancellazione, spostamento, ecc.) e le rende disponibili ai processi tramite system call
- Gestisce tutte le strutture dati per l'implementazione del file system (struttura della directory e FCB), ma non il contenuto dei file vero e proprio
- È responsabile del controllo degli accessi (realizza i meccanismi di protezione)

Realizzazione del file system

- Struttura del file system
- Operazioni del file system
- Realizzazione delle directory
- Montaggio di un file system
- Metodi di allocazione
- Gestione dello spazio libero
- Efficienza e prestazioni

Operazioni del file system

- La realizzazione delle operazioni del file system richiede diverse strutture dati, memorizzate sia nei dischi che in memoria principale
- Normalmente le strutture dati si trovano su disco
- Alcune informazioni vengono portate in memoria principale
  - sia per la gestione del file system
  - che per migliorare le prestazioni
- Vengono
  - caricate in memoria principale al montaggio del file system o quando si creano o si aprono file/directory
  - aggiornate mentre si opera sul file system
  - eliminate allo smontaggio o quando si cancellano o si chiudono file/directory

Strutture dati nei dischi

- Secondary boot sector, anche detto Boot control block o blocco di controllo dell'avviamento:
  primo blocco di una partizione, contiene informazioni su come avviare un SO memorizzato nella partizione stessa (è vuoto se la partizione non contiene un SO)
- Partition/volume control block o indice del volume o directory di dispositivo o superblocco:
  contiene informazioni sulla partizione/volume, quali numero e dimensione dei blocchi, numero e locazione dei blocchi liberi, numero e locazione dei FCB liberi
- Struttura della directory (una per file system):
  tabella i cui elementi contengono associazioni tra nome file/sottodirectory e (collegamenti a) FCB
- FCB o descrittori di file o inode UNIX:
  mantengono i valori degli attributi dei singoli file e informazioni sull'allocazione dei blocchi dei file su disco

Un tipico blocco di controllo di file

permessi per il file
data e ora di creazione, di ultimo accesso e di ultima scrittura
proprietario del file, gruppo, ACL
dimensione del file
blocchi di dati del file o puntatori a blocchi di dati del file

Strutture dati in memoria

- Tabella di montaggio:
  contiene informazioni su ciascuna partizione/volume montata
- Cache della struttura della directory:
  contiene informazioni sui file e sulle directory accedute più di recente dai processi
- Tabella generale dei file aperti:
  contiene copia dei FCB dei file aperti e informazioni sui processi che accedono ciascun file
- Tabelle dei file aperti per ciascun processo:
  per ciascun file aperto dal processo, contiene un puntatore all'elemento corrispondente della tabella generale e altre info sul file
- Vari buffer che conservano i blocchi letti/scritti di recente

Tabelle dei file aperti

- Nella tabella dei file aperti di ciascun processo sono memorizzate le informazioni sull'utilizzo dei file da parte del processo, quali ad esempio
  - il valore corrente del puntatore interno al file
  - i diritti di accesso al file
- Ogni elemento della tabella dei file aperti del processo punta a sua volta alla tabella dei file aperti del sistema, la quale contiene informazioni sui file che sono indipendenti dal processo, quali ad esempio
  - posizione dei file su disco
  - dimensioni dei file, date degli ultimi accessi, …
  - contatore delle aperture, per tenere traccia del numero di processi che hanno effettuato una `open()` e non hanno ancora effettuato la `close()` corrispondente

Esempio: creazione di un nuovo file

- Per creare un nuovo file (o directory), un processo si rivolge al file system logico
- Questo, che conosce il formato della struttura della directory,
  - alloca un nuovo FCB o,
  - se l'implementazione del file system crea tutti i FCB quando il file system viene creato, alloca un FCB tra quelli liberi
- Il sistema quindi carica in memoria la directory appropriata, la aggiorna con il nome del nuovo file e con il FCB associato, e la riscrive nuovamente nel disco

Esempio di system call sui file: `open()`

Vediamo come vengono utilizzate le strutture dati relative al file system quando viene invocata una `open(pathname,oflags)`

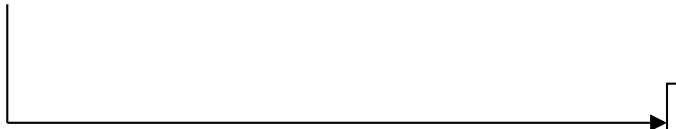
- `pathname` è il nome del file
- `oflags` indica la modalità di accesso al file: `create`, `read-only`, `read-write`, `append-only` …
  - La modalità viene controllata rispetto alle autorizzazioni (diritti di accesso) sul file che il processo possiede
  - Il file viene aperto per il processo richiedente solo se la modalità specificata nella richiesta è consentita

Esempio di system call sui file: `open()`

- Tramite la tabella generale dei file aperti, il SO controlla se il file è già in uso da parte di qualche processo
- In caso affermativo, incrementa il contatore delle aperture associato al file e aggiunge alla tabella dei file aperti del processo richiedente un elemento che punta al corrispondente elemento della tabella generale dei file aperti
- Altrimenti, cerca il nome del file nella struttura della directory; una volta trovato il file,
  - copia il relativo FCB in un nuovo elemento della tabella generale dei file aperti (con contatore delle aperture impostato a 1)
  - aggiunge un elemento alla tabella dei file aperti del processo che punta ad esso; altri campi di tale elemento possono includere un puntatore alla posizione corrente nel file (per le successive operazioni `read()` o `write()`) e la modalità di accesso con cui il file è aperto
- Restituisce un indice alla tabella dei file aperti del processo, sicché tutte le successive operazioni sul file da parte del processo avverranno usando tale indice

Esempio di system call sui file: `open()`

```
fd = open(/d1/f1, "R")
```



Tipo accesso	Cursore	Attributi	Allocazione file su disco
R	0		inizio = Blocco 30

Ogni successiva read/write da parte del processo deve fare riferimento al file tramite il *file descriptor* fd restituito dalla `open()`

Esempio di system call sui file: `close()`

- Rimuove l'elemento della tabella dei file aperti del processo il cui indice è l'argomento della `close()`
- Decrementa il contatore delle aperture dell'elemento riferito nella tabella generale dei file aperti
  - Se il contatore delle aperture si azzera, quindi il file non è più in uso,
    - Copia tutti i metadati aggiornati del file nella struttura della directory sul disco
    - Elimina l'elemento corrispondente nella tabella generale dei file aperti
- Se è il caso, copia il contenuto aggiornato del file in memoria secondaria

System call sui file

- In generale, una system call sui file attraversa i diversi livelli del file system a partire dal primo, fino ad arrivare all'ultimo, il driver, che si occupa di far svolgere al controllore del disco su cui si trova il file/directory coinvolto le operazioni necessarie a soddisfare la richiesta
- Vedremo i dettagli della realizzazione della `read(fd,bytes,&buffer)` quando parleremo del sottosistema di I/O, visto che essa coinvolge non solo il file system ma anche il sottosistema di I/O

Realizzazione del file system

- Struttura del file system
- Operazioni del file system
- Realizzazione delle directory
- Montaggio di un file system
- Metodi di allocazione
- Gestione dello spazio libero
- Efficienza e prestazioni

Realizzazione delle directory

- La funzione principale della struttura della directory è mappare il nome ASCII di un file sulle informazioni necessarie per localizzare i dati
- Ogni directory necessita del collegamento con i descrittori dei file che contiene
  - Il problema che si pone è dove debbano essere memorizzati i descrittori dei file

Realizzazione delle directory

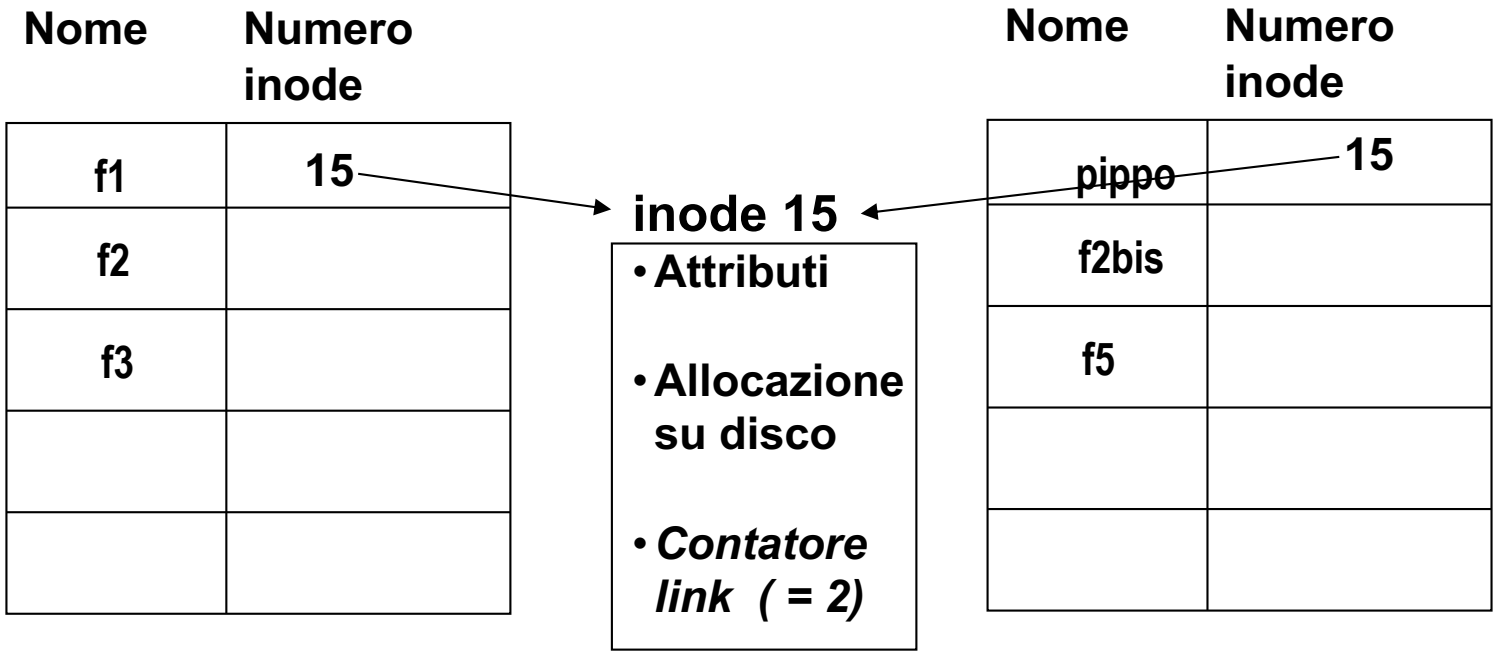
- Alcuni SO (es. Windows) realizzano le directory come file speciali con contenuto strutturato in modo specifico:
  - una tabella i cui elementi contengono nome di un file/sottodirectory, attributi corrispondenti, informazioni sulla allocazione dei *blocchi* del file su disco

Nome	Attributi	Allocazione file su disco
d1		inizio = Blocco 30
d2		inizio = Blocco 43
f1		inizio = Blocco 110

- Inconveniente: memorizzare direttamente le informazioni sui file (cioè i descrittori) negli elementi della directory crea problemi per l'implementazione dei link (e, quindi, per la condivisione di file)

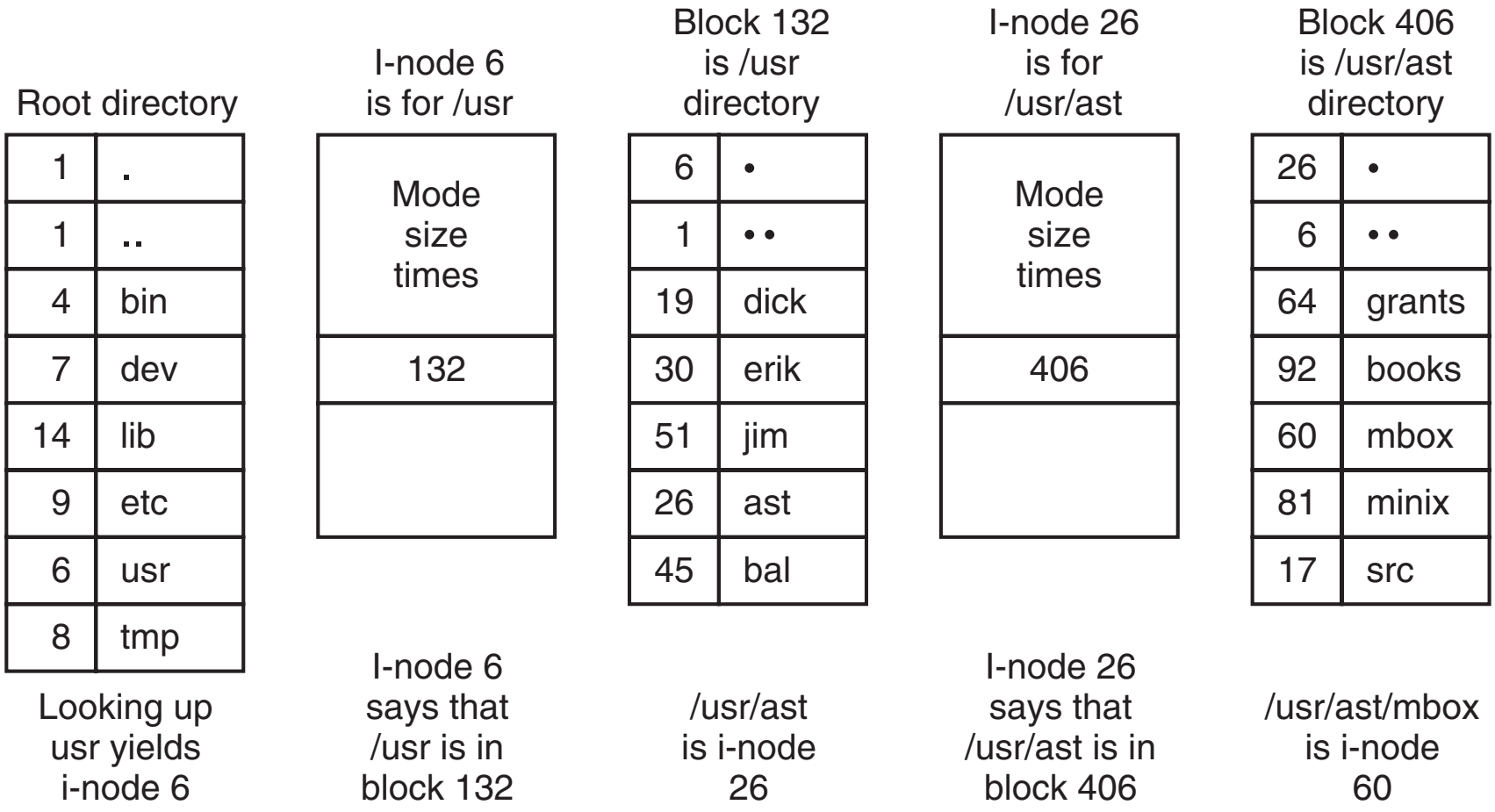
Realizzazione delle directory

- Per implementare più convenientemente i link, conviene tenere attributi e informazioni sull'allocazione in una struttura separata (es. inode UNIX) e fare in modo che ogni elemento di directory punti all'inode (unico) del file/sottodirectory

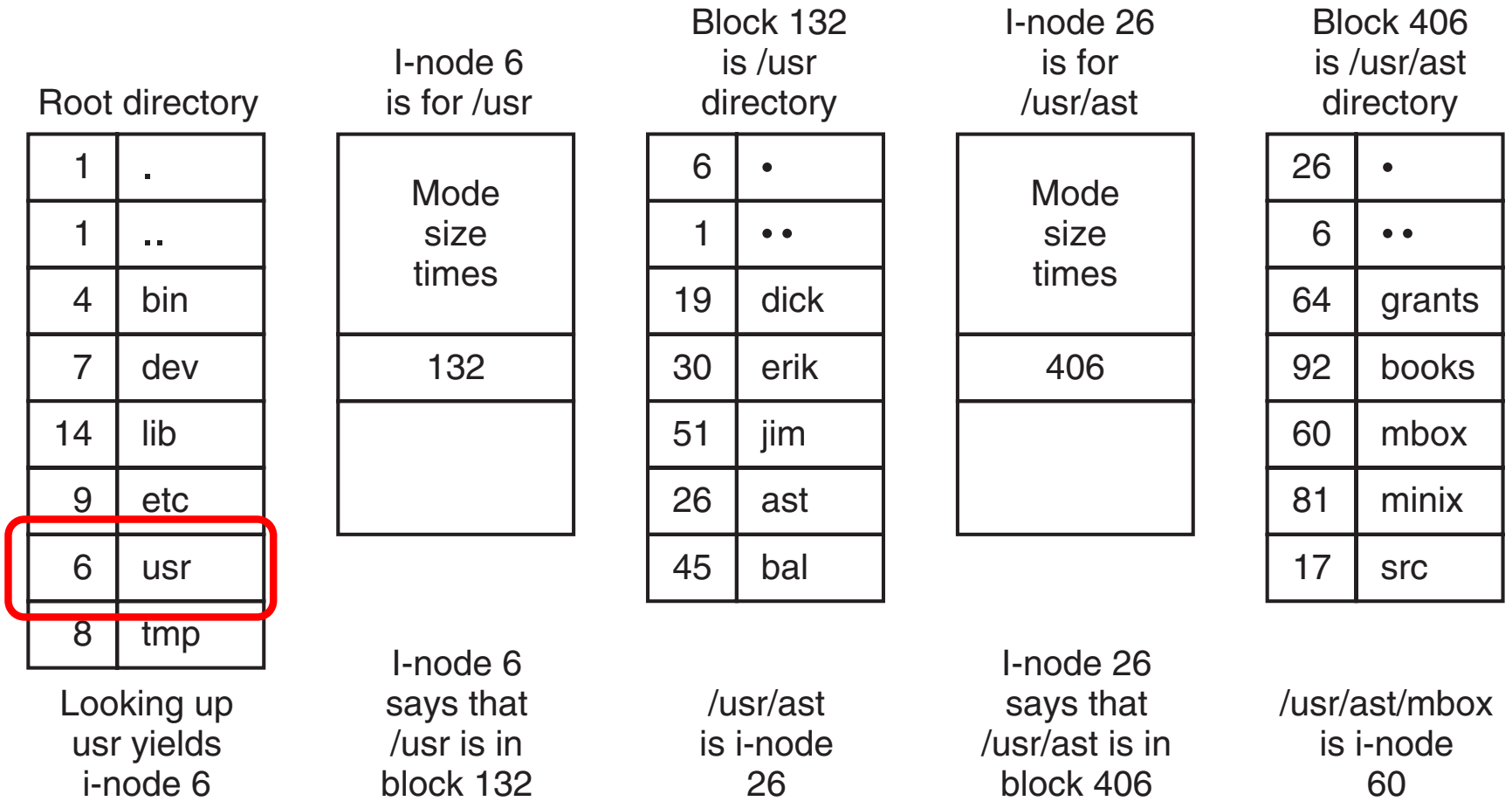


- Tramite l'uso di un contatore (inserito nell'inode) si può facilmente implementare la cancellazione di un file con più pathname: l'effettiva cancellazione avviene solo quando il contatore si azzera; negli altri casi viene solo eliminato il pathname e decrementato il contatore

Ricerca del file `/usr/ast/mbox`



Ricerca del file `/usr/ast/mbox`



Si parte dalla root directory `/` e si cerca l'indice dell'inode corrispondente a `usr`, si trova il valore 6

Si accede l'inode 6 e si trova che la tabella di directory corrispondente a `/usr` sta nel blocco dati 132

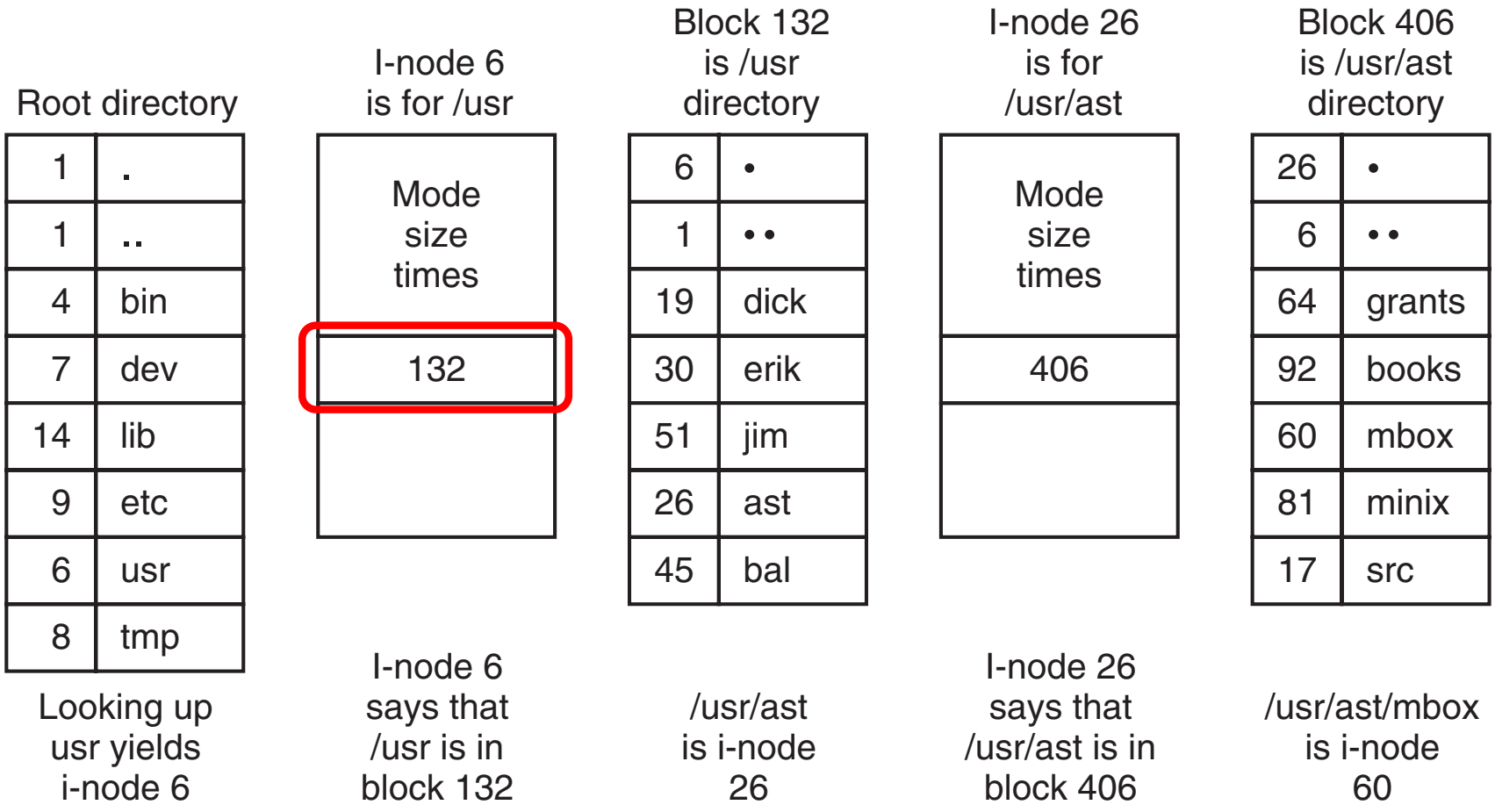
Si accede il blocco 132 che contiene la tabella di directory e si trova l'indice 26 dell'inode corrisp. a `ast`

Si accede l'inode 26 e si trova che la tabella di directory corrispondente a `/ast` sta nel blocco dati 406

Si accede il blocco 406 che contiene la tabella di directory e si trova l'indice 60 dell'inode corrisp. a `mbox`

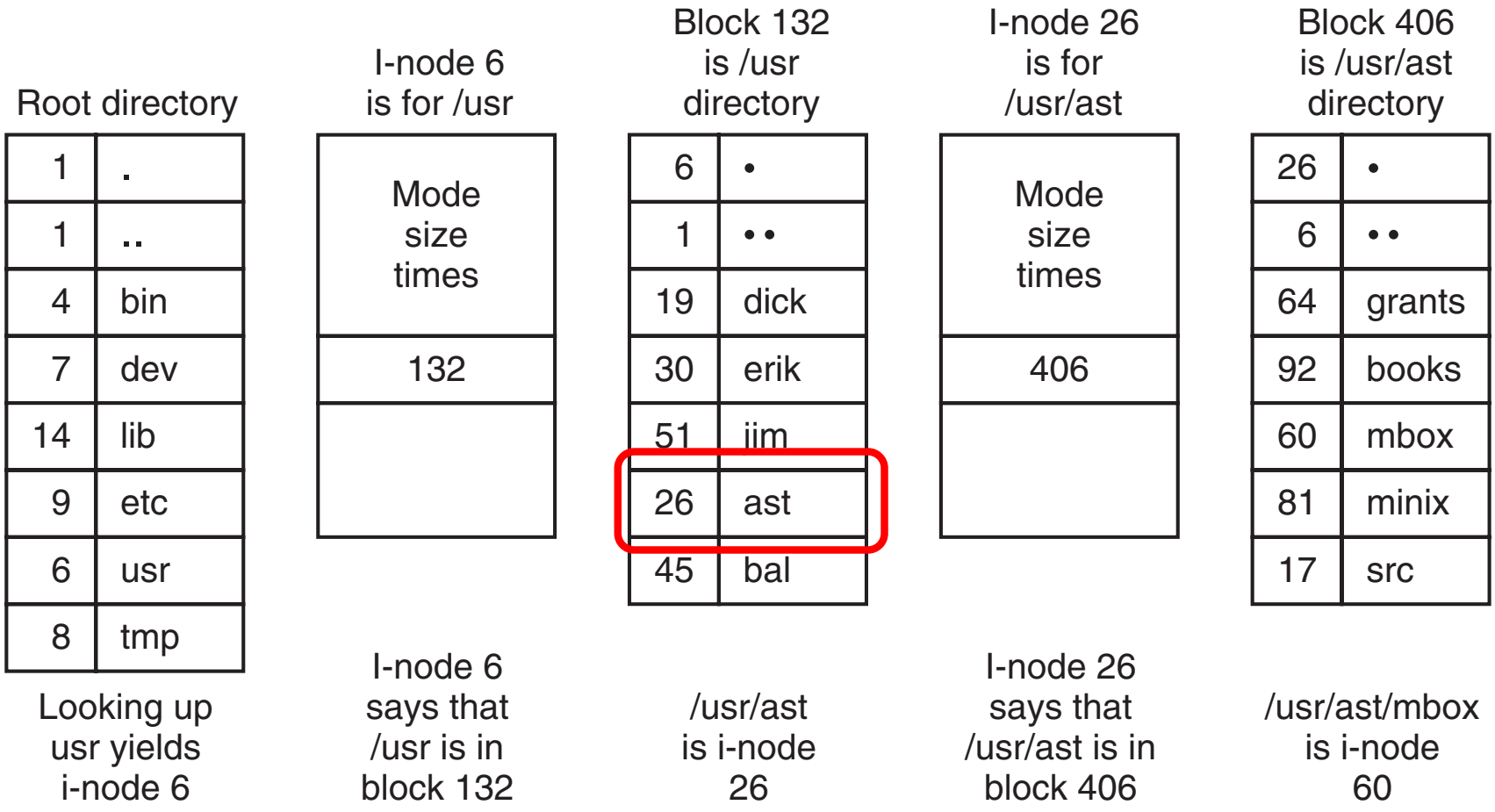
Infine, si accede l'inode 60 che corrisponde al file `/usr/ast/mbox`

Ricerca del file `/usr/ast/mbox`



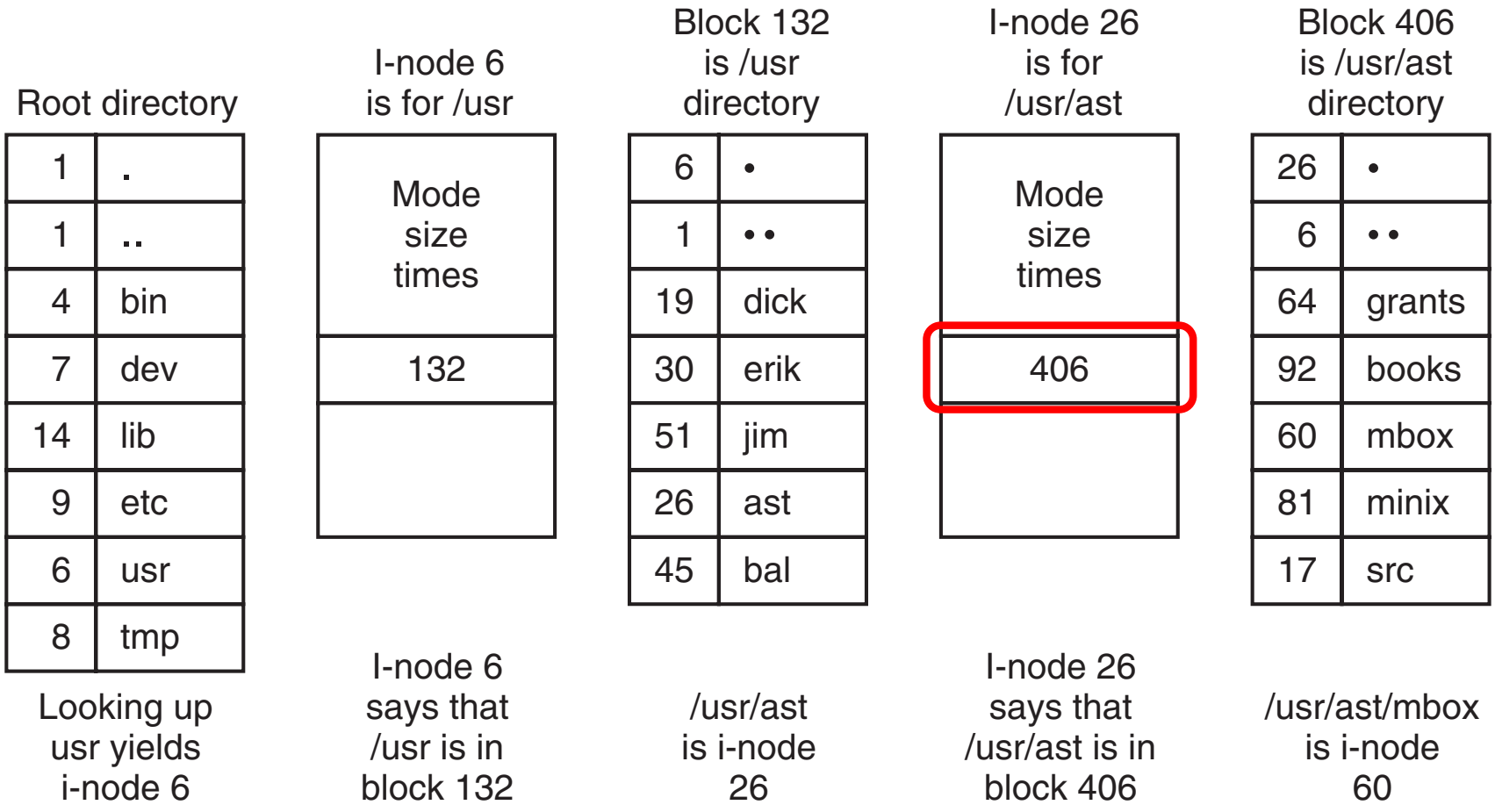
Si parte dalla root directory `/` e si cerca l'indice dell'inode corrispondente a `usr`, si trova il valore 6
Si accede l'inode 6 e si trova che la tabella di directory corrispondente a `/usr` sta nel blocco dati 132
Si accede il blocco 132 che contiene la tabella di directory e si trova l'indice 26 dell'inode corrisp. a `ast`
Si accede l'inode 26 e si trova che la tabella di directory corrispondente a `/ast` sta nel blocco dati 406
Si accede il blocco 406 che contiene la tabella di directory e si trova l'indice 60 dell'inode corrisp. a `mbox`
Infine, si accede l'inode 60 che corrisponde al file `/usr/ast/mbox`

Ricerca del file `/usr/ast/mbox`



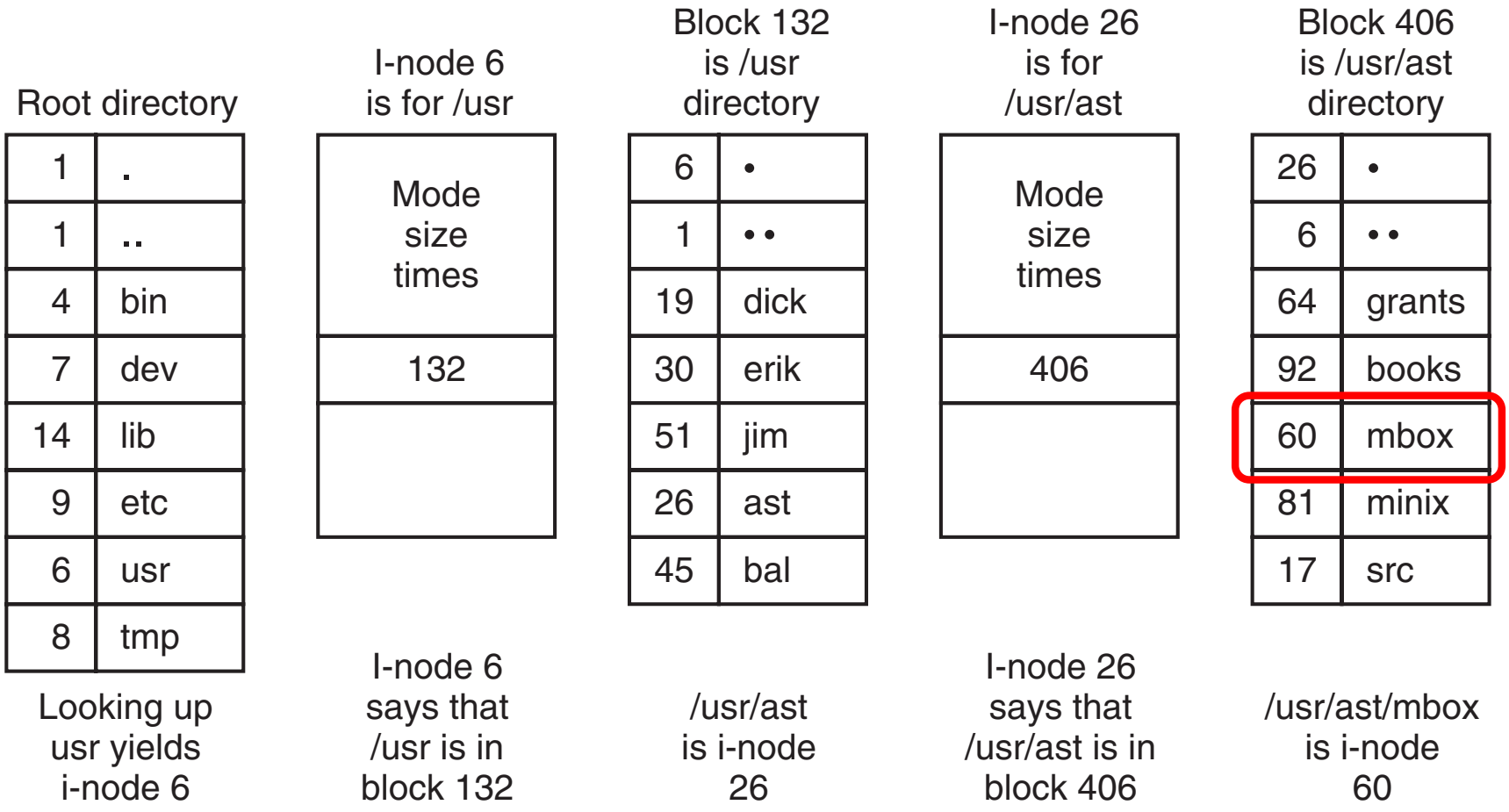
Si parte dalla root directory `/` e si cerca l'indice dell'inode corrispondente a `usr`, si trova il valore 6
Si accede l'inode 6 e si trova che la tabella di directory corrispondente a `/usr` sta nel blocco dati 132
Si accede il blocco 132 che contiene la tabella di directory e si trova l'indice 26 dell'inode corrisp. a `ast`
Si accede l'inode 26 e si trova che la tabella di directory corrispondente a `/ast` sta nel blocco dati 406
Si accede il blocco 406 che contiene la tabella di directory e si trova l'indice 60 dell'inode corrisp. a `mbox`
Infine, si accede l'inode 60 che corrisponde al file `/usr/ast/mbox`

Ricerca del file `/usr/ast/mbox`



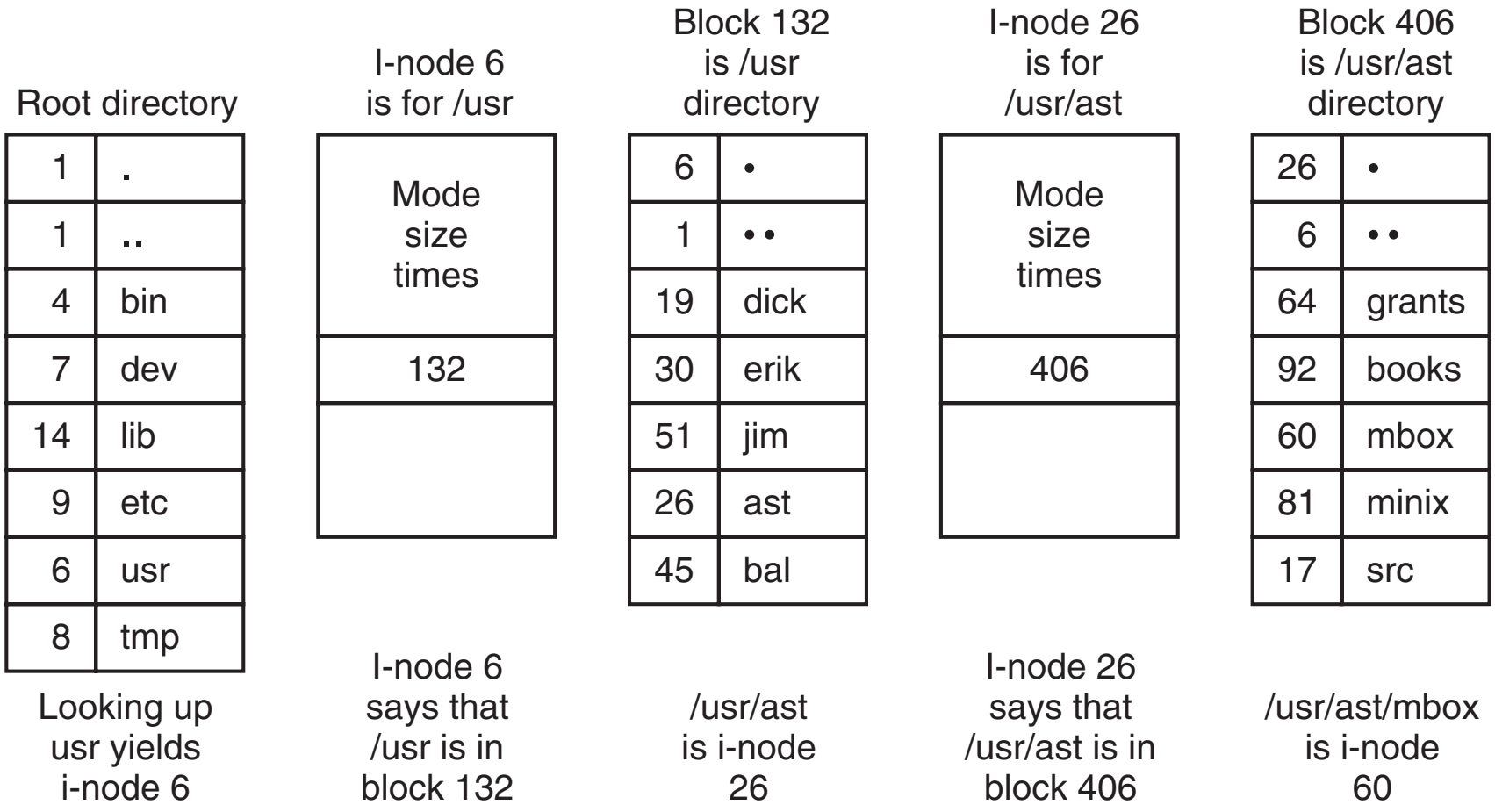
Si parte dalla root directory `/` e si cerca l'indice dell'inode corrispondente a `usr`, si trova il valore 6
Si accede l'inode 6 e si trova che la tabella di directory corrispondente a `/usr` sta nel blocco dati 132
Si accede il blocco 132 che contiene la tabella di directory e si trova l'indice 26 dell'inode corrisp. a `ast`
Si accede l'inode 26 e si trova che la tabella di directory corrispondente a `/ast` sta nel blocco dati 406
Si accede il blocco 406 che contiene la tabella di directory e si trova l'indice 60 dell'inode corrisp. a `mbox`
Infine, si accede l'inode 60 che corrisponde al file `/usr/ast/mbox`

Ricerca del file `/usr/ast/mbox`



Si parte dalla root directory `/` e si cerca l'indice dell'inode corrispondente a `usr`, si trova il valore 6
Si accede l'inode 6 e si trova che la tabella di directory corrispondente a `/usr` sta nel blocco dati 132
Si accede il blocco 132 che contiene la tabella di directory e si trova l'indice 26 dell'inode corrisp. a `ast`
Si accede l'inode 26 e si trova che la tabella di directory corrispondente a `/ast` sta nel blocco dati 406
Si accede il blocco 406 che contiene la tabella di directory e si trova l'indice 60 dell'inode corrisp. a `mbox`
Infine, si accede l'inode 60 che corrisponde al file `/usr/ast/mbox`

Ricerca di `/usr/ast/mbox`



Si parte dalla root directory `/` e si cerca l'indice dell'inode corrispondente a `usr`, si trova il valore 6
Si accede l'inode 6 e si trova che la tabella di directory corrispondente a `/usr` sta nel blocco dati 132
Si accede il blocco 132 che contiene la tabella di directory e si trova l'indice 26 dell'inode corrisp. a `ast`
Si accede l'inode 26 e si trova che la tabella di directory corrispondente a `/ast` sta nel blocco dati 406
Si accede il blocco 406 che contiene la tabella di directory e si trova l'indice 60 dell'inode corrisp. a `mbox`
Infine, si accede l'inode 60 che corrisponde al file `/usr/ast/mbox`

Organizzazione delle directory

Due schemi principali per organizzare le tabelle delle directory

- Lista lineare di elementi
  - semplice da programmare
  - l'efficienza per l'esecuzione delle operazioni può essere bassa (creazione e cancellazione di file richiedono ricerche esaustive nella tabella, il cui costo è lineare nel numero degli elementi)
  - se la lista è mantenuta ordinata si riduce il tempo di ricerca dei file, ma si allunga quello di creazione/cancellazione
- Tabella hash: il nome del file è usato per calcolare un valore hash, gli elementi della directory con lo stesso valore hash sono mantenuti in una lista lineare
  - tabelle con un numero fisso di elementi (pari al numero di valori differenti che la funzione hash può restituire)
  - velocizza la ricerca nella directory
  - creazione e cancellazione dei file sono più semplici
  - amministrazione più complessa (andrebbe usata in sistemi in cui si prevede che le directory avranno centinaia o migliaia di file)

Realizzazione del file system

- Struttura del file system
- Operazioni del file system
- Realizzazione delle directory
- Montaggio di un file system
- Metodi di allocazione
- Gestione dello spazio libero
- Efficienza e prestazioni

Montaggio di un File System

- Così come bisogna *aprire* un file prima di usarlo, un file system deve essere montato prima di poter essere reso accessibile ai processi ed agli utenti di un sistema
- Tale operazione si rende necessaria per costruire la struttura della directory
  - Un file system può essere composto da file system che risiedono su volumi differenti
    - Un volume si può pensare come un dispositivo virtuale che corrisponde a una partizione o a un gruppo di partizioni (anche di dispositivi fisici diversi)
  - I volumi devono essere montati, cioè raggruppati in un'unica struttura gerarchica, affinché siano disponibili nello spazio dei nomi del file system
- Almeno un file system deve essere presente all'avvio del SO, affinché il montaggio degli altri file system sia possibile
  - Tale file system prende il nome di root file system (tipicamente è associato ad un disco rigido installato permanentemente)

Montaggio di un File System

- L'operazione di montaggio richiede di specificare
  - il nome del volume (contenente il file system) da montare
  - il punto di montaggio (locazione che occuperà nel file system esistente), o *mount point*
  - il tipo di file system contenuto nel volume (alcuni SO lo ricavano automaticamente)
- Il SO verifica che il volume contenga un file system valido
  - chiedendo al driver del dispositivo che ospita il volume di leggere la directory di dispositivo (o indice del volume)
  - controllando che tale directory abbia il formato previsto
- Infine, il SO annota nella sua struttura della directory che un certo file system è montato nel punto di montaggio specificato
  - Questo schema consente al SO di attraversare la sua struttura della directory, passando anche tra file system di tipo diverso
- L' operazione di montaggio può essere effettuata
  - automaticamente dal SO (es. Windows, Mac-OS)
  - esplicitamente dall'utente (es. UNIX, Linux)

Montaggio in macOS

- Il SO macOS, quando rileva un disco per la prima volta (all'avvio o mentre il sistema è in esecuzione), cerca un file system nel dispositivo
- Se ne trova uno,
  - lo monta automaticamente nella directory `/Volumes` e
  - aggiunge al desktop un'icona di cartella etichettata con il nome del file system (secondo quel che è memorizzato nella directory di dispositivo o indice del volume)
- L'utente può quindi selezionare l'icona con il mouse e visualizzare il contenuto del file system appena montato

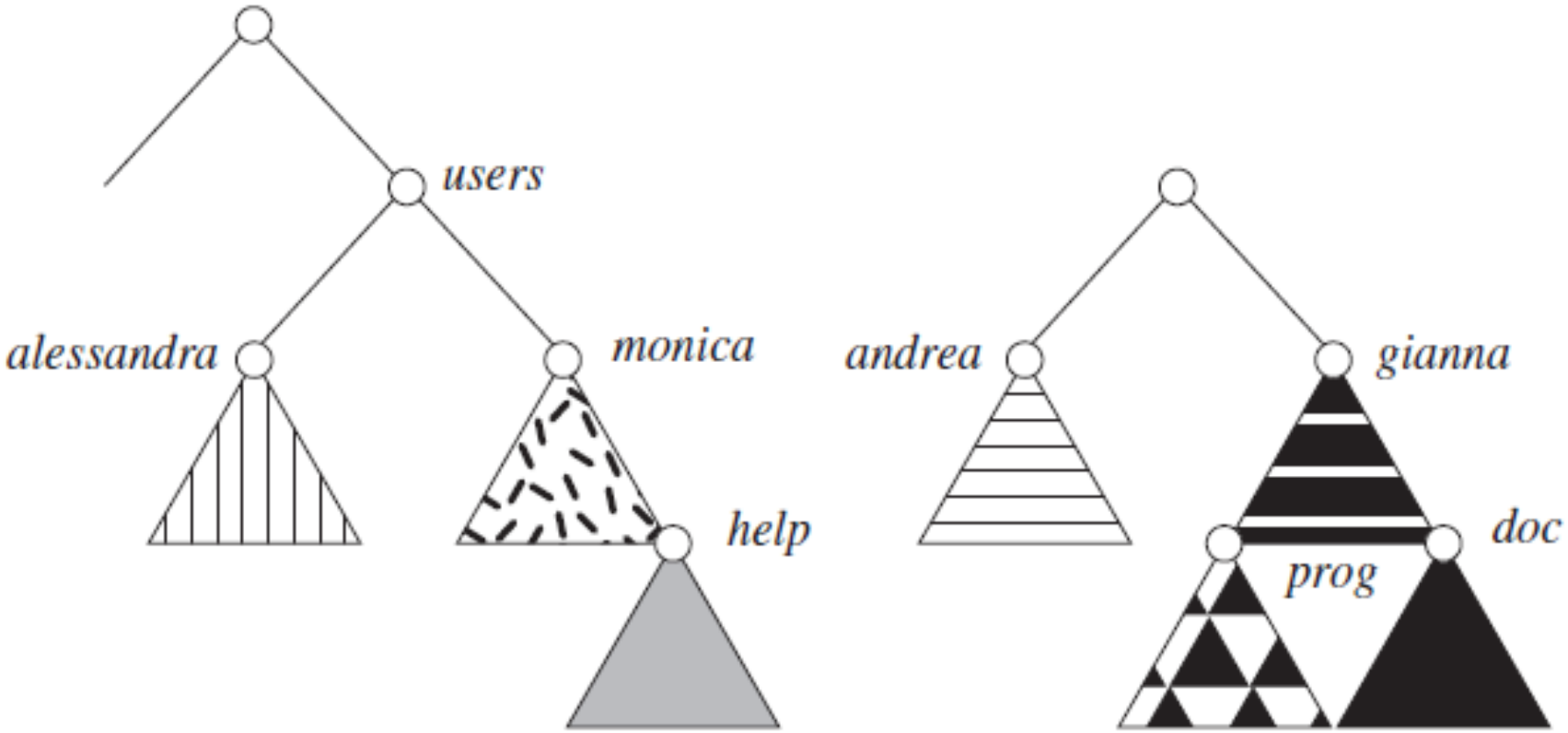
Montaggio in Windows

- I sistemi operativi della famiglia Microsoft Windows automaticamente rilevano tutti i dispositivi e montano tutti i file system individuati all'avvio
- I SO Windows mantengono una struttura della directory a due livelli estesa
  - Una lettera di unità è associata a dispositivi e volumi
  - I volumi hanno una struttura della directory a grafo generale
  - Il percorso completo di un file specifico assume la forma `lettera_di_unità:\percorso\file`
- Le versioni più recenti di Windows consentono di montare un file system in qualsiasi punto dell'albero delle directory, proprio come fa UNIX

Montaggio in UNIX

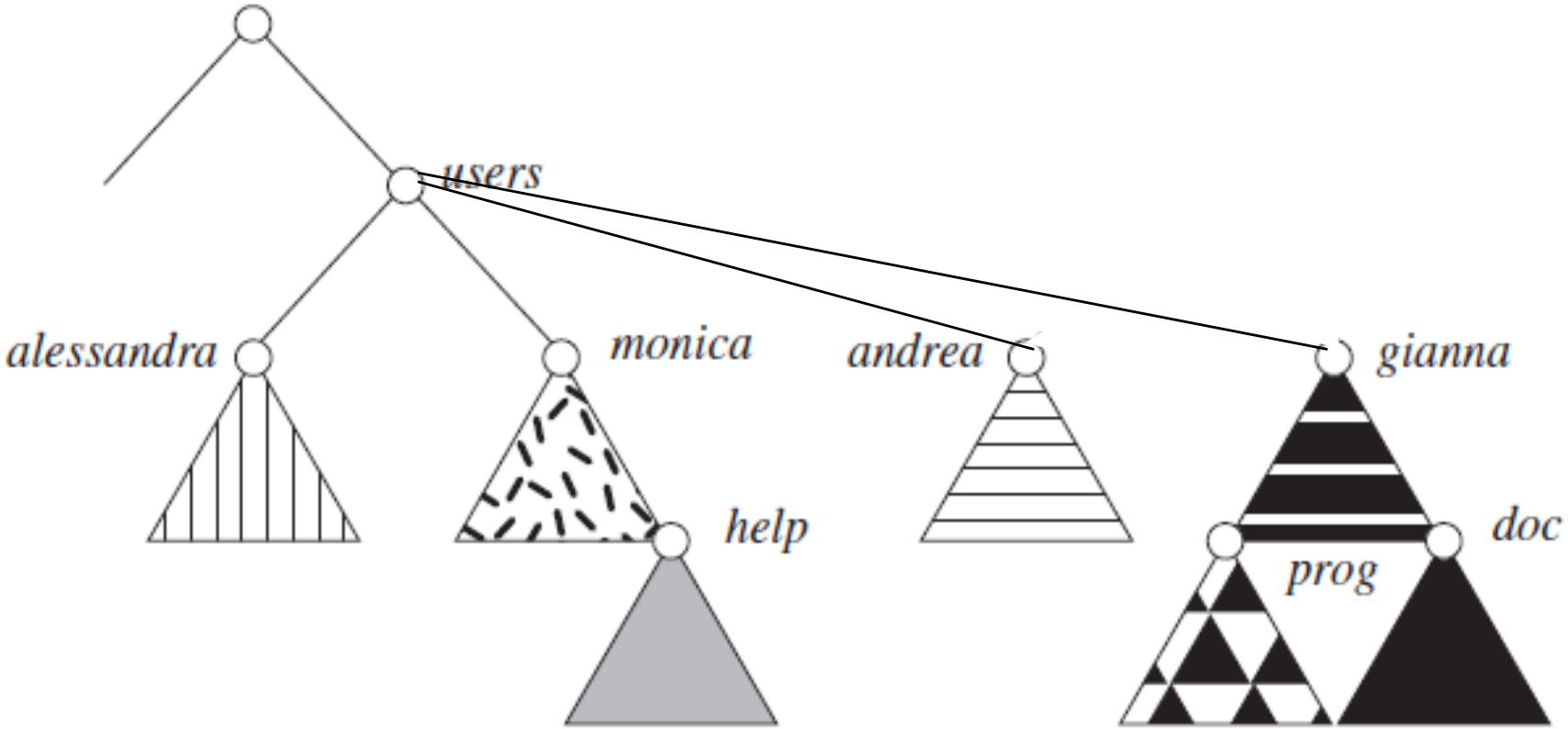
- In alcuni SO, come UNIX, i comandi `mount` sono espliciti
- Un file di configurazione del sistema contiene un elenco di dispositivi e punti di montaggio per il montaggio automatico all'avvio, ma altri montaggi possono essere eseguiti manualmente
- In UNIX, il montaggio di un file system si può effettuare in qualsiasi directory
- Il montaggio viene implementato impostando un flag nella copia in memoria dell'inode della directory di montaggio
  - Il flag indica che la directory è un punto di montaggio
  - Un campo dell'inode punta a un elemento della tabella di montaggio, che contiene un puntatore al superblocco (cioè l'indice del volume) del file system montato in quella posizione

(a) Esistente (b) Volume non montato



Un volume contenente le home directory di alcuni utenti (b) potrebbe essere montato nella directory `/users` del file system (a)

Punto di montaggio



È ora possibile accedere alla struttura della directory all'interno del volume montato: basta premettere `/users` ai nomi della directory, come in `/users/gianna`

Smontaggio di un File System

- È l'operazione opposta al montaggio
- Stacca un file system dal suo punto di montaggio, eventualmente riversando il contenuto dei buffer del kernel sul volume contenente il file system

- Per motivi di efficienza, le scritture su un file system (su un dispositivo fisico) sono eseguite in blocco, al momento più favorevole
  - Estrarre fisicamente un dispositivo senza aver prima smontato il suo file system può causare la corruzione dei dati contenuti!

Realizzazione del file system

- Struttura del file system
- Operazioni del file system
- Realizzazione delle directory
- Montaggio di un file system
- Metodi di allocazione
- Gestione dello spazio libero
- Efficienza e prestazioni

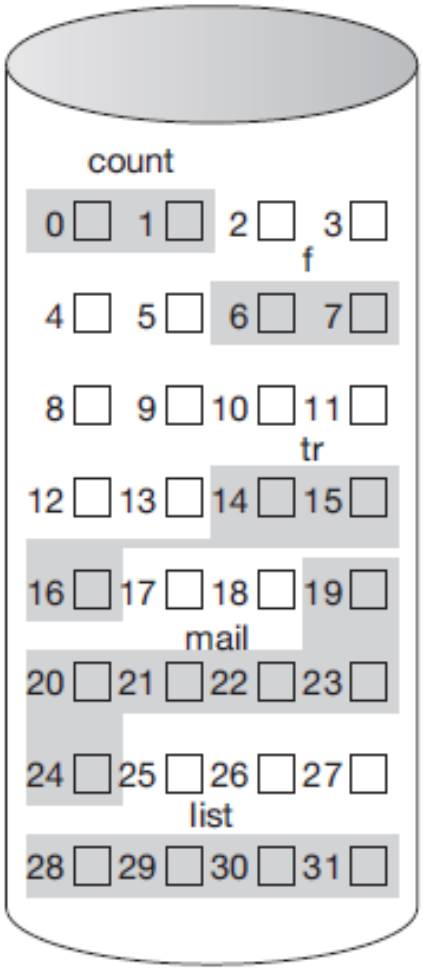
Allocazione dei file

- L'HW e il driver del disco forniscono accesso al disco visto come un vettore lineare di blocchi di dimensione fissa
- I metodi di allocazione dei file
  - scelgono i blocchi del disco da utilizzare per ciascun file e li collegano assieme per formare una struttura unica
  - stabiliscono una corrispondenza tra i record logici contenuti nei file ed i blocchi del disco
- Obiettivi:
  - usare efficientemente lo spazio su disco
  - rendere rapido l'accesso ai file
- I metodi più diffusi sono
  - allocazione contigua
  - allocazione concatenata
  - allocazione indicizzata
- Indicheremo con N_b il *numero di record logici contenuti in un blocco* ($N_b = \lfloor D_b/D_r \rfloor$ dove D_r = dimensione record e D_b = dimensione blocco)
  - Il record logico i di un file si trova nel blocco (logico) $\lfloor i/N_b \rfloor$ del file: es. se $N_b = 10$ e $i = 53$, il blocco in questione è $\lfloor i/N_b \rfloor = 5$

Allocazione contigua

Ogni file occupa un certo numero di blocchi contigui su disco

- La porzione allocata ad un file è definita dall'indirizzo B del blocco iniziale e dal numero di blocchi (informazioni da mantenere nel FCB)
- L'indirizzo del blocco in cui si trova il record logico i è $B+\lfloor i/N_b \rfloor$



directory

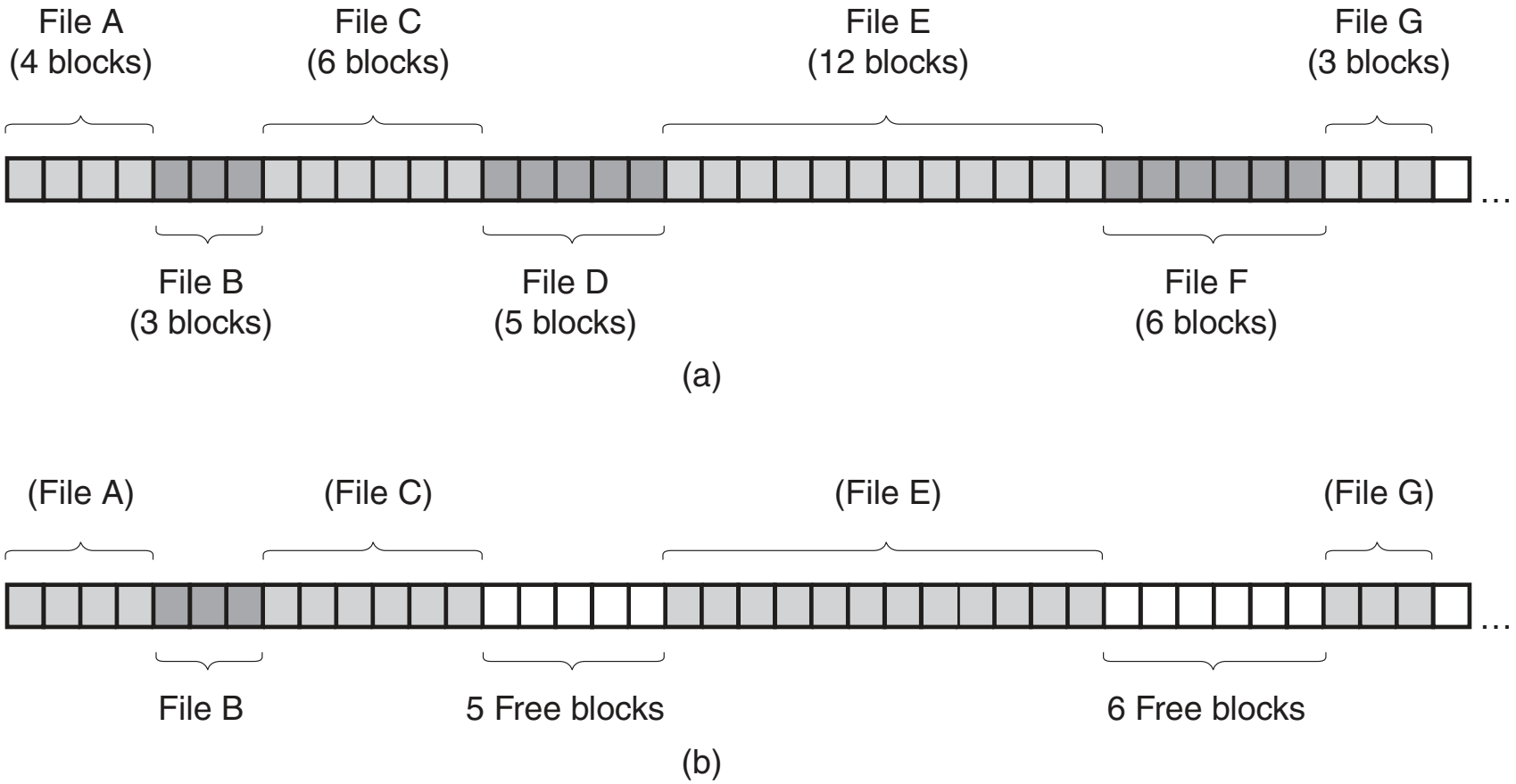
file	blocco iniziale	lunghezza
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

L'elemento della directory relativo ad un file contiene l'indirizzo del blocco iniziale ed il numero di blocchi

Allocazione contigua

- Vantaggi
  - basso costo della ricerca di un blocco
  - accesso sequenziale e accesso diretto efficienti
  - alte prestazioni (basta la sola ricerca del primo blocco per leggere l'intero file)
- Svantaggi
  - frammentazione esterna: man mano che il disco si riempie, rimangono zone contigue sempre più piccole, a volte inutilizzabili; richiede compattazione o deframmentazione
  - necessità di specificare la dimensione finale di un file al momento della sua creazione (per determinare il numero dei blocchi necessari)
  - elevato costo della ricerca dello spazio libero per l'allocazione di un nuovo file (si possono usare gli algoritmi di allocazione dinamica della memoria principale: best-fit, first-fit, worst-fit)
  - limiti sulle dimensioni dei file

Allocazione contigua dello spazio del disco: frammentazione esterna



(a) Allocazione contigua dello spazio del disco per 7 file
(b) Lo stato del disco dopo la cancellazione dei file D e F

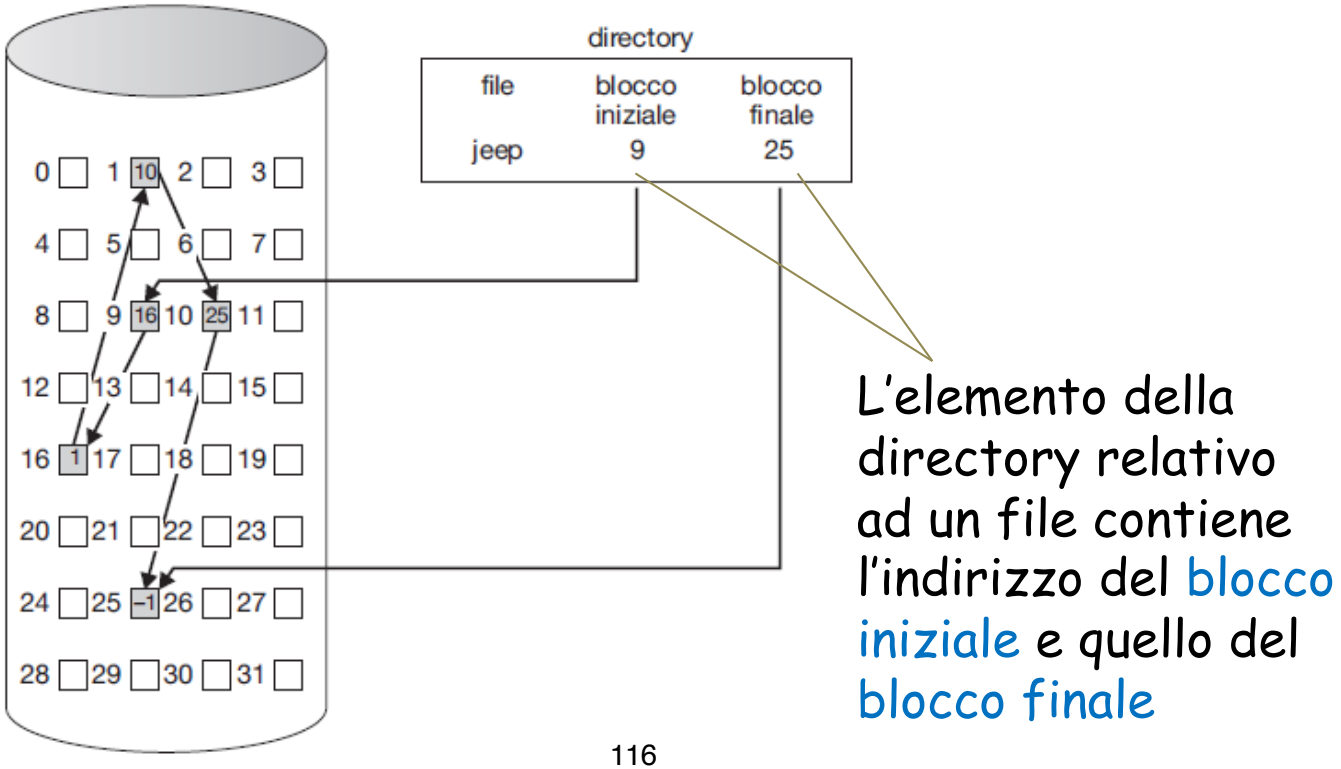
Estensione

- SO recenti (ad esempio Symantec Veritas File System, un sostituto ad alte prestazioni dell'ordinario UFS di UNIX) usano uno schema di allocazione contigua modificato
- Quando un file viene creato, gli viene inizialmente allocata una porzione contigua di spazio disco; successivamente, quando la quantità non è più sufficiente, si alloca un'estensione, cioè un'altra porzione contigua
- Un file è quindi allocato in una o più estensioni (porzioni contigue)

Allocazione concatenata

I blocchi di un file sono organizzati in una lista concatenata

- Ogni blocco contiene al suo interno (tipicamente in posizione iniziale) un puntatore al blocco successivo
- Le informazioni da mantenere nel FCB sono il puntatore al primo blocco e il numero di blocchi (o il puntatore all'ultimo blocco)
- Per determinare l'indirizzo del blocco $\lfloor i/N_b \rfloor$ (in cui si trova il record logico i) bisogna scorrere tutti i blocchi (del disco) che lo precedono a partire da quello iniziale del file!

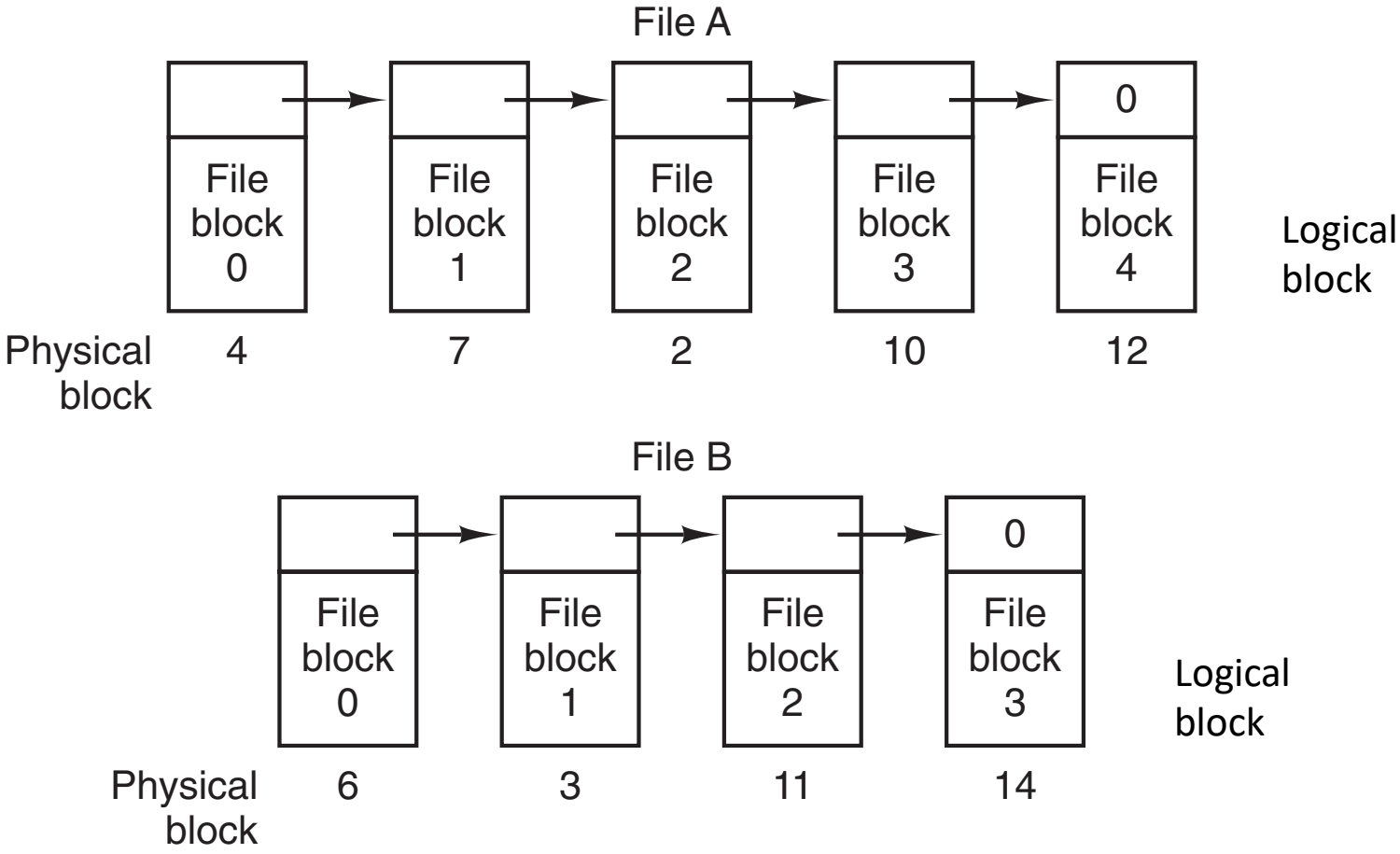


L'elemento della directory relativo ad un file contiene l'indirizzo del blocco iniziale e quello del blocco finale

Allocazione concatenata

- Vantaggi
  - Accesso sequenziale o in modalità 'append' efficienti
  - Non c'è frammentazione esterna: i blocchi assegnati ad un file possono essere sparpagliati ovunque nel disco
  - Non c'è bisogno di dichiarare preventivamente le dimensioni dei file
  - Basso costo della ricerca dello spazio libero per l'allocazione di un nuovo file o per l'espansione di un file esistente
- Svantaggi
  - Accesso diretto e ricerca di un blocco richiedono molte operazioni di I/O
  - Bassa efficienza: molti programmi leggono e scrivono in blocchi la cui dimensione è una potenza di 2, ma così non è per la quantità di dati memorizzati in un blocco, per via del puntatore al blocco successivo
  - Spazio richiesto dai puntatori (per risparmiare spazio, alcuni sistemi permettono di allocare cluster di blocchi consecutivi)
  - Scarsa robustezza

Allocazione concatenata



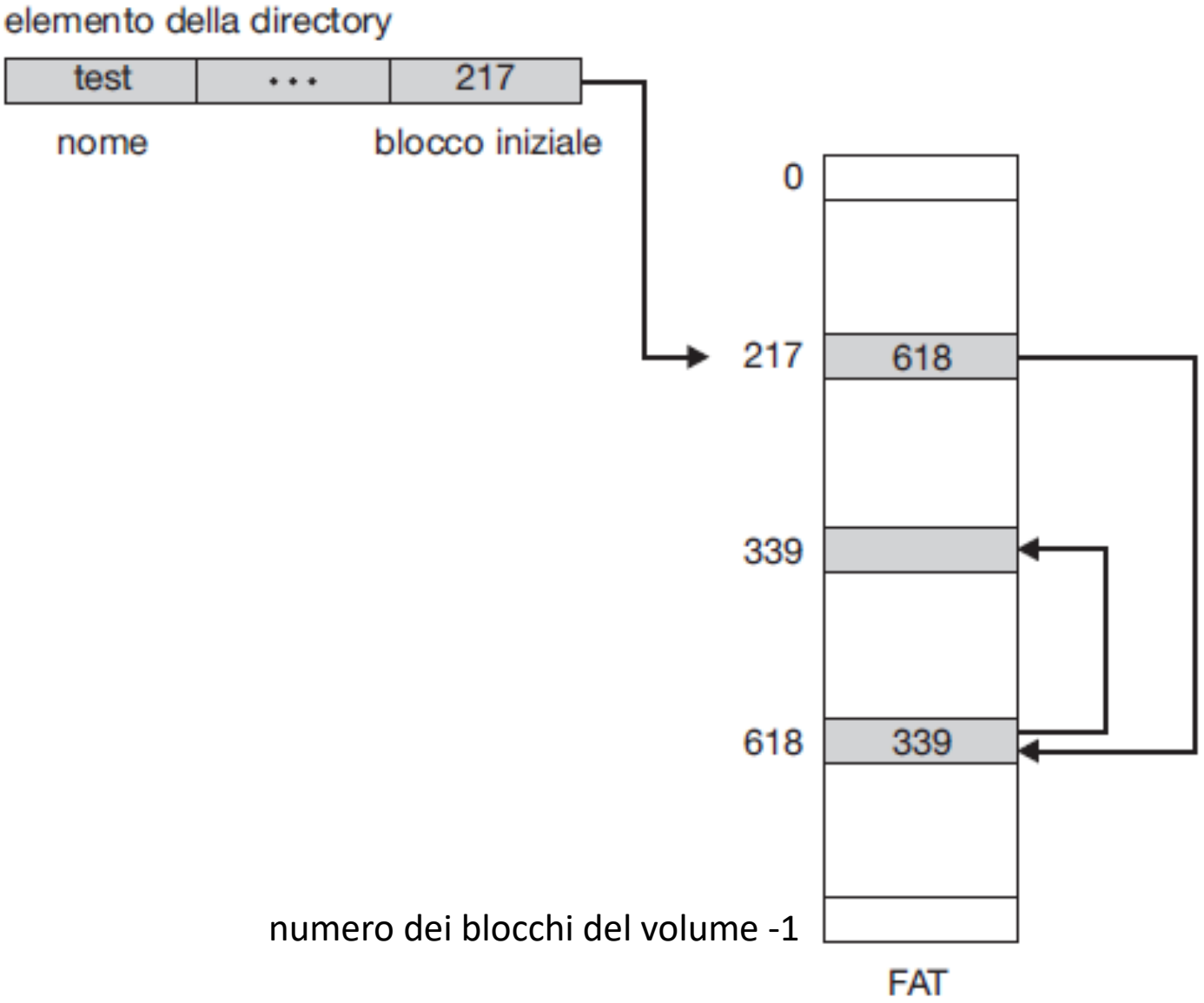
Svantaggio principale: scarsa robustezza

- Se un link viene danneggiato, non è più possibile accedere ai (record nei) blocchi successivi
- Soluzioni: lista a doppi puntatori o Tabella di Allocazione dei File

Tabella di Allocazione dei File (File Allocation Table, FAT)

- Metodo di allocazione usato inizialmente da MS-DOS (poi anche da OS/2 e Windows)
- È una struttura dati tabellare che descrive la mappa di allocazione di tutti i blocchi
  - Contiene un elemento per ogni blocco del volume il cui valore indica se il blocco è libero oppure, se occupato, contiene l'indice dell'elemento della tabella corrispondente al blocco successivo (o un valore speciale per indicare che è l'ultimo blocco di un file)
- È memorizzata in un'area predefinita del volume (a volte in duplice copia)
  - In pratica, memorizza le informazioni sulla lista concatenata in un'unica area di memoria contigua
  - I blocchi dati dei file possono non contenere puntatori

Tabella di Allocazione dei File



FAT16

- MS/DOS prevedeva indirizzi su disco di 16 bit (inizialmente, 12 bit!), quindi il volume non poteva avere più di 2^{16} blocchi
- Perciò, un volume DOS contiene una FAT, allocata all'inizio del volume, di 2^{16} elementi di 16 bit ciascuno, detta FAT16 (ereditata poi da Windows 95)
- Pertanto, se la dimensione massima dei blocchi è fissata a 32 KB (= 2^{15} byte), si arriva a poter gestire volumi di dimensioni pari a $2^{16} \times 2^{15}$ byte = 2 GB
- Per formattare volumi di grandi dimensioni con FAT16 occorre aumentare le dimensione dei blocchi, aumentando così la frammentazione interna

FAT32

- Il problema è stato risolto con la FAT32
  - Volume da 256 GB = 2^{38} byte, **FAT32** =>
    servono blocchi di dimensione pari a $2^{38} / 2^{32} = 2^6 = 64$ byte
- Non conviene comunque avere blocchi di dimesione inferiore a 512 byte, mentre il limite massimo per la dimensione dei blocchi è fissato a 32 KB
- Per poter utilizzare un file system FAT32
  - un volume deve contenere almeno 2^{16} blocchi
  - mentre il numero massimo di blocchi è in realtà 2^{28} perché solo 28 bit si possono usare per indicizzare i blocchi
- Pertanto, con blocchi di 32 KB, teoricamente si possono gestire volumi delle dimensioni massime di $2^{28} \times 2^{15}$ byte = 8 TB!
- La dimensione massima dei file è di circa 2^{32} byte = 4 GB
  (è una conseguenza dell'elemento "lunghezza file" nel FCB:
  è formato da 4 byte ed esprime la lunghezza in termini di byte)

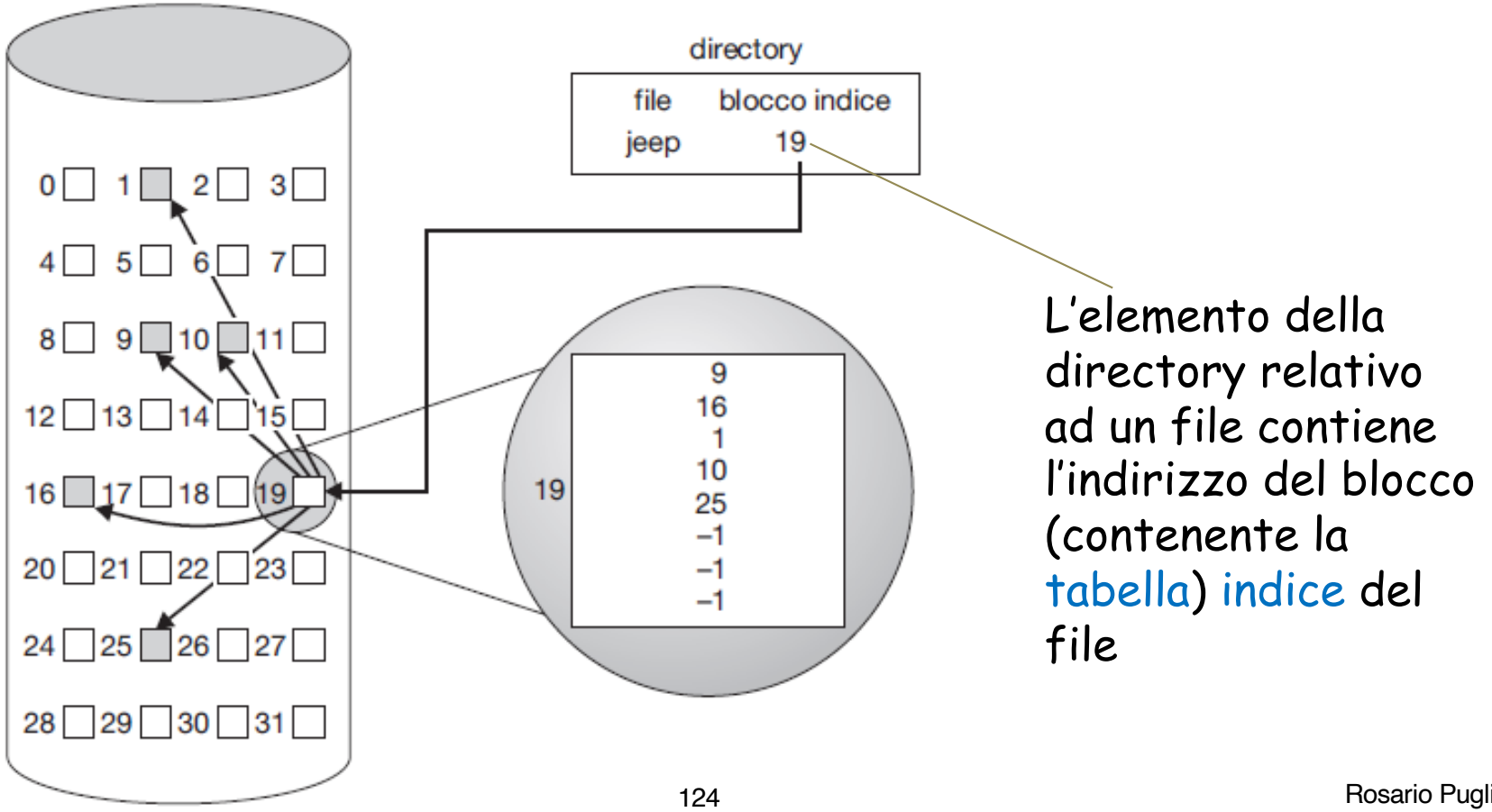
Tabella di Allocazione dei File

- Vantaggi
  - Incrementa la robustezza
  - I blocchi dati sono interamente dedicati ai dati
  - Può velocizzare notevolmente l'accesso ai file tramite copia in memoria principale (con o senza l'uso di una cache): l'accesso diretto costa n-1 accessi in memoria ed 1 su disco
- Svantaggi
  - L'intera tabella deve essere sempre mantenuta in memoria principale, quindi non si presta bene per dischi di grandi dimensioni
    - Con un disco da 1TB e dimensione del blocco di 1KB, la tabella necessita di più di un miliardo di voci, una per ciascuno dei blocchi del disco
    - Ogni voce deve avere minimo 4 byte
    - Quindi la tabella occuperà almeno 4GB di memoria principale per tutto il tempo!

Allocazione indicizzata

A ogni file è associata una tabella indice, di dimensione prestabilita, in cui sono contenuti gli indirizzi dei blocchi del file

- Tutti gli indici dei blocchi sono raggruppati in una sola locazione
- L'indirizzo del blocco $\lfloor i/N_b \rfloor$ (in cui si trova il record logico i) è contenuto nell'elemento $\lfloor i/N_b \rfloor$ della tabella indice



L'elemento della directory relativo ad un file contiene l'indirizzo del blocco (contenente la tabella) indice del file

Allocazione indicizzata

- Vantaggi
  - gli stessi dell'allocazione a lista concatenata, più
  - possibilità di accesso diretto
  - maggiore velocità di accesso
- Svantaggi
  - richiede memoria per il blocco (contenente la tabella) indice
  - scarso utilizzo dei blocchi indice (nel caso di file di dimensioni ridotte)
  - overhead per l'accesso al blocco indice
  - le dimensioni del blocco indice limitano le dimensioni del file

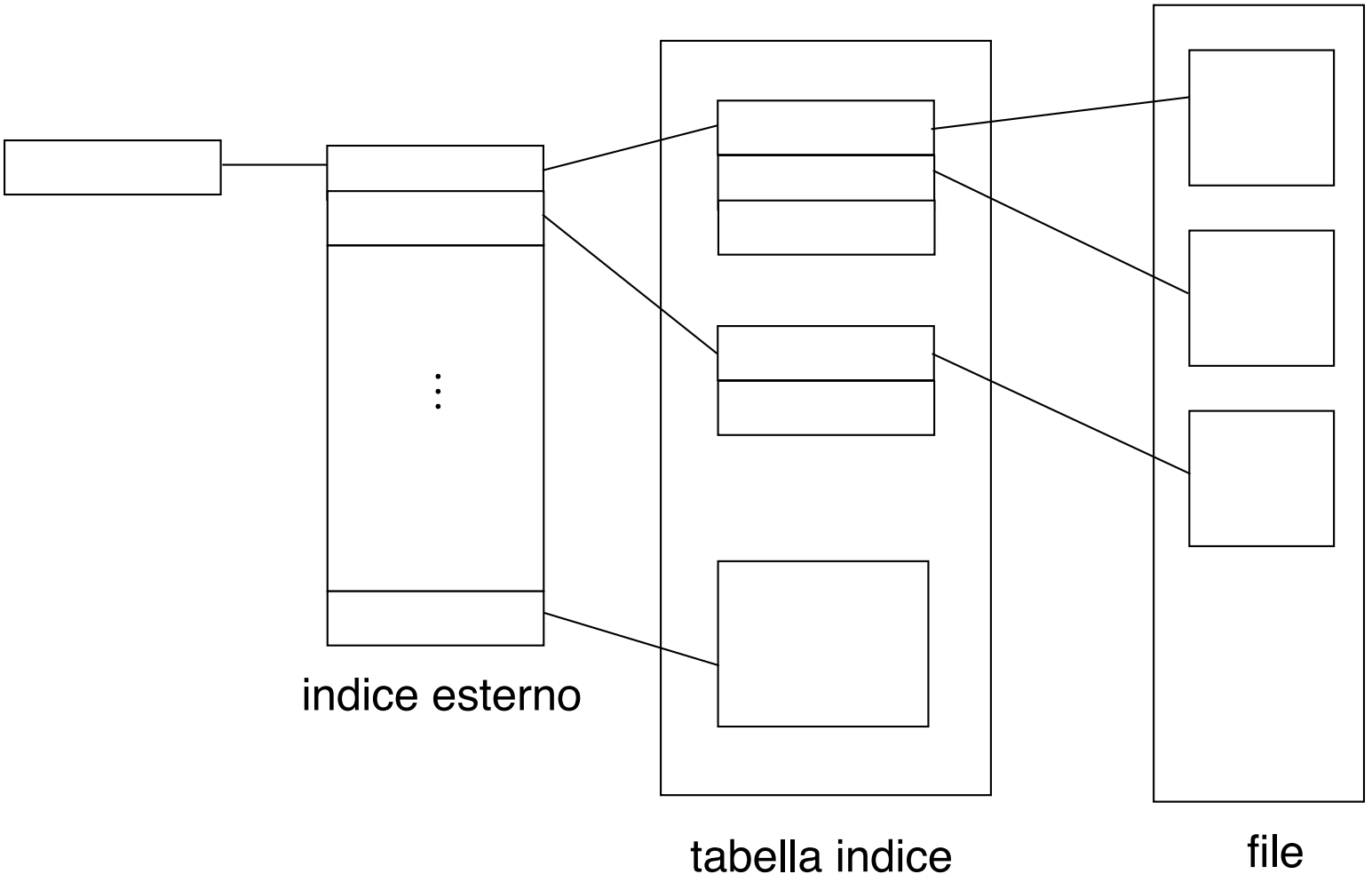
Mappatura

- Le dimensioni del blocco indice limitano le dimensioni dei file
  - Se i blocchi sono di 512 parole, ciacuna di 4 byte, e si usa un solo blocco per la tabella indice si possono indirizzare 512 blocchi per cui la dimensione massima dei file è 512 x 512 parole = 256K parole = 1MB
- Per ovviare a questa restrizione alcuni SO usano vari metodi
  - liste concatenate di blocchi indice: uno degli indirizzi nel blocco indice (es. l'ultimo) punta a un blocco indice successivo
  - più livelli di indice (cioè gerarchie ad albero)

    Se i blocchi sono di 4096 byte, un singolo blocco può memorizzare 1024 puntatori di 4 byte e quindi la dimensione massima di un file con un solo livello di indice è 1024 x 4096 byte = 4MB

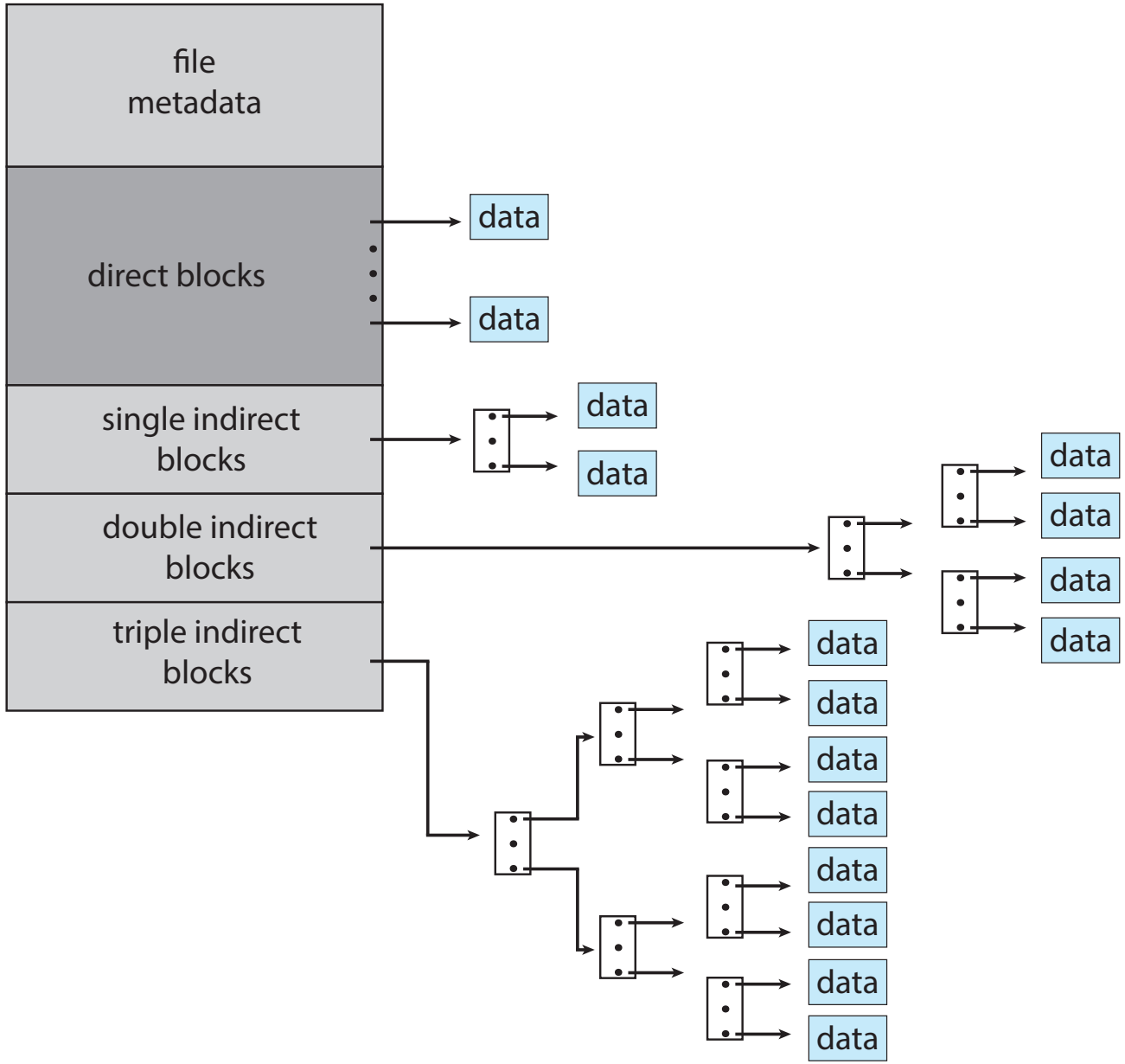
    Se si usa un doppio livello di indice, si arriva a 1024 x 4MB = 4GB
  - uno schema combinato (UNIX utilizza un metodo basato su 3 livelli)

Mappatura: indice a più livelli



Schema combinato: inode UNIX

12 blocchi diretti



Schema combinato: inode UNIX

- Gestisce efficientemente i file piccoli penalizzando gradualmente l'accesso a file grandi
  - Indirizzi 1-12: puntano direttamente a blocchi dati del file
  - Indirizzo 13: punta ad un blocco che contiene indirizzi di blocchi dati del file (singola indirezione)
  - Indirizzo 14: doppia indirezione
  - Indirizzo 15: tripla indirezione
- La dimensione massima dei file dipende da quella dei blocchi e dei loro indirizzi
  - blocchi su disco di 1024 byte, indirizzi su disco di 4 byte:
    quindi ogni blocco contiene 1024 / 4 = 256 indirizzi
  - per come sono organizzati gli inode, si ha
    maxdimfile = $(12 + 256 + 256^2 + 256^3) \times 1024$ byte $\approx 2^{24} \times 2^{10}$ byte = 8 Gbyte

Schema combinato: inode UNIX

- L'occupazione di memoria è piuttosto contenuta poiché un inode ha bisogno di essere in memoria solo quando il file corrispondente è aperto
  - Quindi, se ciascun inode occupa *n* byte e *k* è il numero massimo di file che possono essere aperti contemporaneamente nel sistema, la memoria occupata dagli inode dei file aperti è di *kn* byte
  - Questa quantità è solitamente molto inferiore a quella richiesta per la FAT, la cui dimensione è invece proporzionale alla dimensione del volume stesso

Metodi di allocazione

- Riassumendo, gli obiettivi sono:
  - usare efficientemente lo spazio su disco
  - rendere rapido l'accesso ai file
- Alcuni SO adottano più metodi di allocazione
  - Contigua: se in genere l'accesso al file è diretto o se il file è relativamente piccolo
  - Concatenata: se il file è di grandi dimensioni e se in genere l'accesso al file è sequenziale
  - Indicizzata: se il file è di grandi dimensioni e se in genere l'accesso al file è diretto

Realizzazione del file system

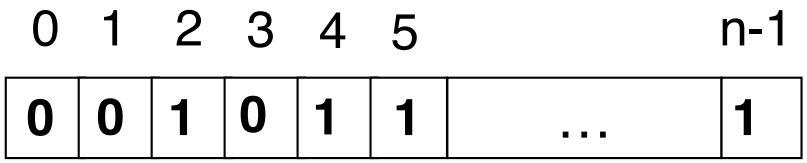
- Struttura del file system
- Operazioni del file system
- Realizzazione delle directory
- Montaggio di un file system
- Metodi di allocazione
- Gestione dello spazio libero
- Efficienza e prestazioni

Gestione dello spazio libero

- Il SO mantiene una lista dei blocchi liberi allocabili per i dati dei file
  - Normalmente, la lista è mantenuta in memoria secondaria per evitare che vada persa a seguito di arresto anomalo del sistema (crash, errore della memoria)
- Vi sono due metodi principali per gestirla
  - vettore di bit (bitmap)
  - lista concatenata di blocchi del disco

Bitmap: vettore di bit

Si usa un vettore di bit con tanti elementi quanti sono i blocchi



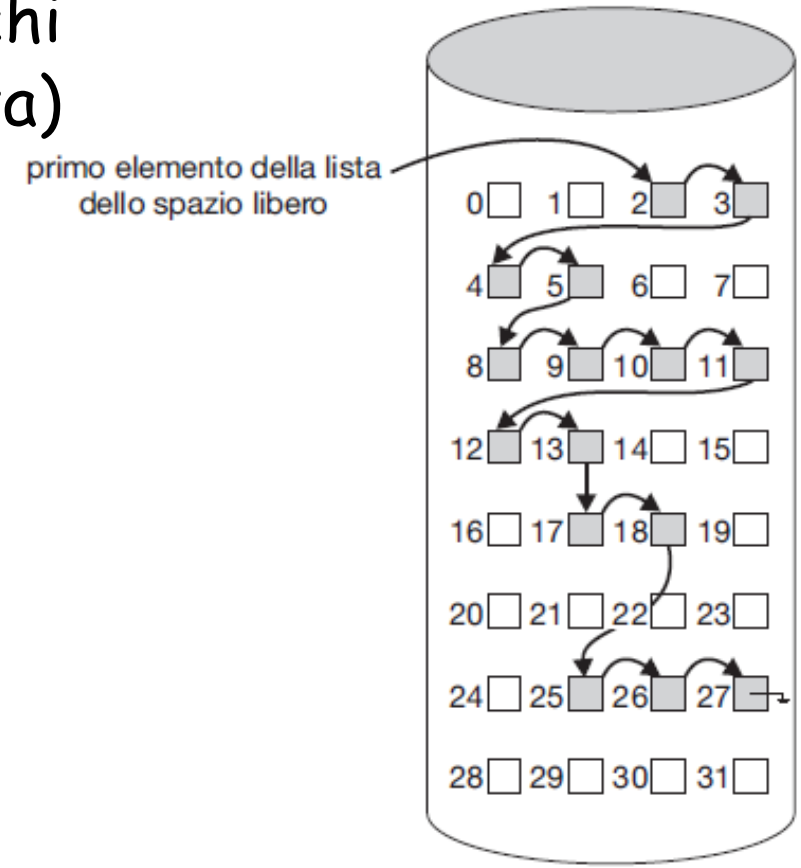
$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{blocco}[i] \text{ libero} \\ 0 \Rightarrow \text{blocco}[i] \text{ occupato} \end{cases}$$

Bitmap: vettore di bit

- Vantaggi
  - Semplicità nel determinare il primo blocco libero o *n* blocchi liberi consecutivi
  - Alcuni processori (es. Intel 80386 e succ. Motorola 68020 e succ.) forniscono istruzioni per determinare la prima parola diversa da 0 nel vettore e il primo bit diverso da 0 nella parola
- Svantaggi
  - Efficiente solo se il vettore di bit viene caricato in memoria centrale
  - Il vettore di bit richiede spazio extra. Esempio:
    - dimensione blocco = 2^{12} byte (4KB)
    - dimensione disco = 2^{40} byte (1TB)
    - lunghezza vettore di bit = $2^{40}/2^{12} = 2^{28}$ bit (32MB)!
  - Organizzazione ragionevole per dischi di dimensioni non elevate

Lista concatenata di blocchi del disco

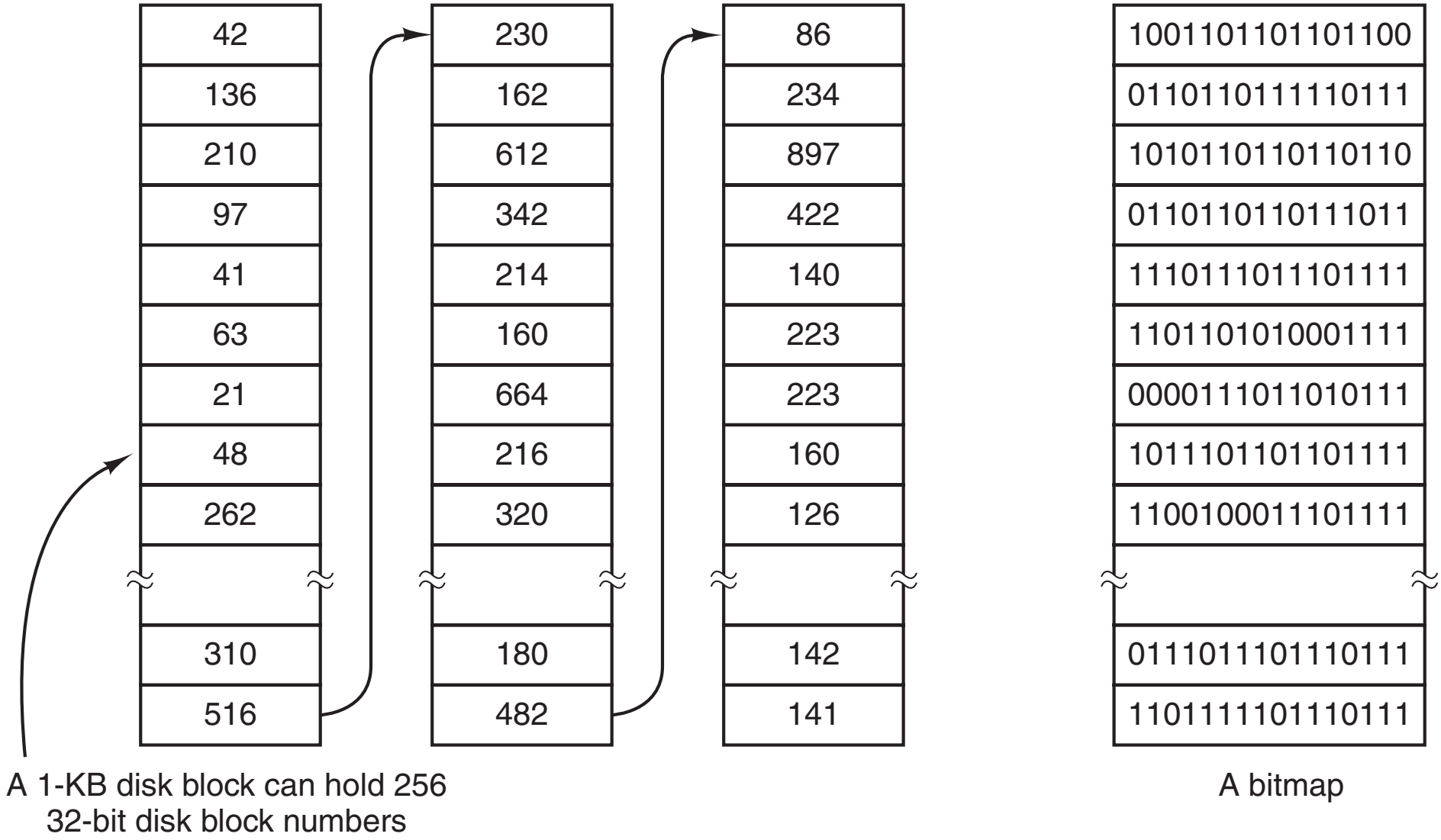
- I blocchi liberi sono collegati l'uno all'altro
- È sufficiente memorizzare il puntatore al primo blocco
- È difficile trovare blocchi liberi contigui
  (a meno che la lista dei blocchi
  non venga mantenuta ordinata)



Lista concatenata: varianti

- Conteggio
  - Ogni blocco mantiene il numero di blocchi liberi consecutivi che lo seguono e l'indirizzo del primo blocco libero successivo a questi
- Raggruppamento
  - Il primo blocco libero memorizza gli indirizzi di altri *n* blocchi liberi
  - L'ultimo di tali blocchi contiene gli indirizzi di altri n blocchi liberi, e così via
  - Permette di trovare rapidamente gli indirizzi di un certo numero di blocchi liberi

Raggruppamento vs. Bitmap



Considerazioni su Raggruppamento & Bitmap

- Raggruppamento
  - È sufficiente tenere in memoria solo un blocco di puntatori
  - Conviene che tale blocco sia solo 'mezzo' pieno così da poter gestire efficientemente (cioè senza bisogno di accedere al disco per reperire un altro blocco di puntatori) sia richieste di blocchi liberi (per la creazione di nuovi file o l'espansione di file esistenti), sia liberazione di blocchi (in seguito alla cancellazione di file)
- Bitmap
  - È semplice reperire un certo numero di blocchi liberi contigui
  - Date le dimensioni, anche la bitmap va suddivisa in blocchi
  - È sufficiente tenere in memoria solo un blocco e accedere al disco per leggerne un altro quando quello in memoria non è soddisfacente (es. contiene solo blocchi occupati)

Realizzazione del file system

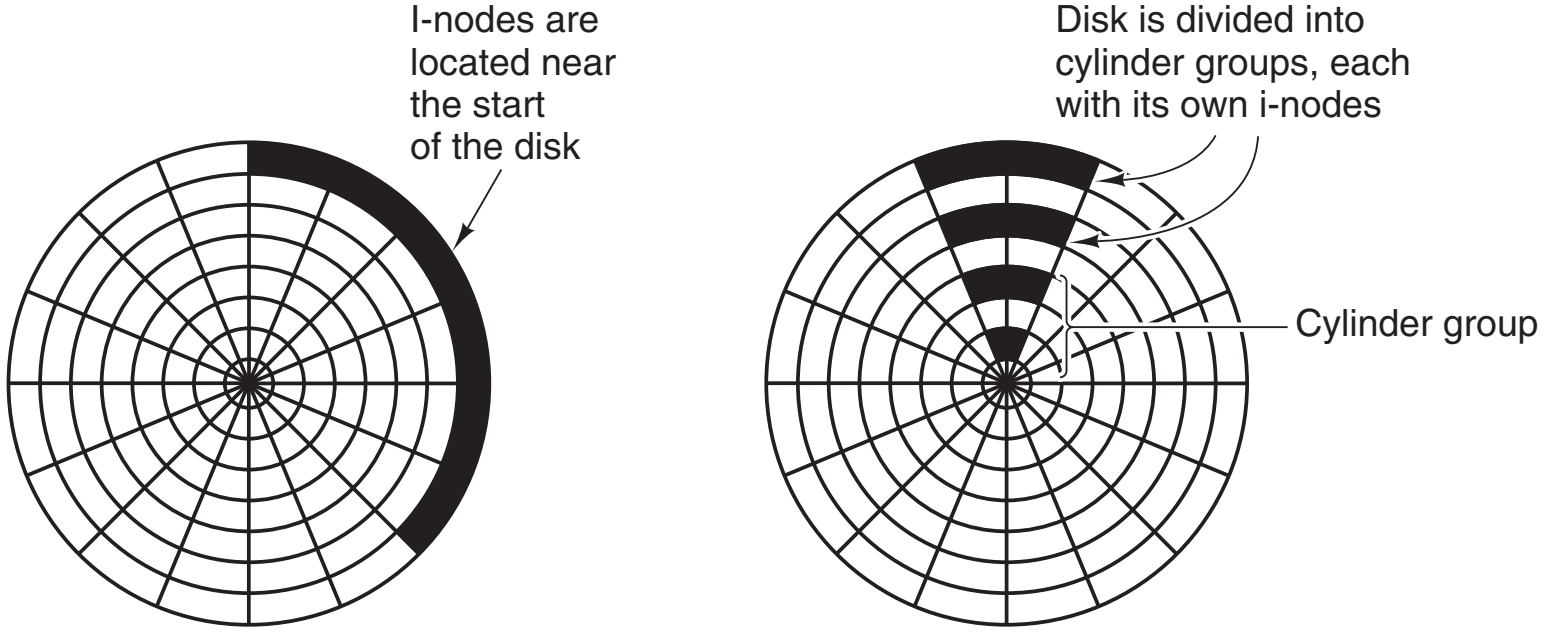
- Struttura del file system
- Operazioni del file system
- Realizzazione delle directory
- Montaggio di un file system
- Metodi di allocazione
- Gestione dello spazio libero
- Efficienza e prestazioni

Efficienza

- I dischi tendono ad essere il principale collo di bottiglia per le prestazioni di un sistema di elaborazione, essendo i più lenti tra i componenti più rilevanti
  - Leggere dalla memoria principale una parola di 32 bit può richiedere 10ns
  - La lettura da un disco fisso potrebbe avvenire a 100MB/s, quattro volte più lentamente, ma a questo va aggiunto il tempo di ricerca della traccia (5-10ms) e poi l'attesa che il settore desiderato si posizioni sotto la testina di lettura
  - Se la richiesta è di una singola parola, l'accesso alla memoria è dell'ordine di un milione di volte più veloce dell'accesso al disco!
- Vari fattori influenzano l'uso efficiente del disco, tra cui
  - Algoritmi per l'allocazione dei blocchi e la gestione dei blocchi liberi
  - Tipo di informazioni in un elemento di directory o in un FCB
  - Pianificazione degli effetti provocati dal cambiamento della tecnologia

Algoritmi per l'allocazione dei blocchi

- Algoritmi per l'allocazione dei blocchi e la gestione dei blocchi liberi
  - Es. per ridurre il tempo di ricerca della traccia su cui sono posizionati i blocchi dati, UNIX cerca di mantenere i blocchi con i dati di un file vicini al blocco che ne contiene l'inode



Posizionare gli inode vicini ai blocchi dati dei relativi file dimezza il tempo medio di ricerca rispetto al caso in cui gli inode siano posizionati tutti all'inizio del disco

Informazioni in un elemento di directory

Tipo di informazioni in un elemento di directory o in un descrittore

- La data dell'ultima scrittura fornisce informazioni all'utente per determinare se occorre aggiornare una copia di backup del file
- La data dell'ultimo accesso consente all'utente di risalire all'ultima volta che un file è stato letto
- Per mantenere aggiornate queste informazioni, quando si scrive/legge un file, si deve aggiornare un elemento di directory o un descrittore di file
  - Questa modifica richiede la lettura del blocco in memoria, la modifica del dato interessato e la riscrittura del blocco nel disco
  - Quindi, ogni volta che si scrive/legge un file, si deve leggere e scrivere anche l'elemento della struttura della directory ad esso associato
  - Ciò può essere inefficiente per file cui si accede frequentemente
- Quindi nella fase di progettazione del file system è necessario considerare l'influenza sull'efficienza e sulle prestazioni di ogni informazione che si vuole associare ai file

Effetti del cambiamento della tecnologia

Gestione delle strutture dati ed algoritmi del kernel

- File system di MS-DOS
  - Originariamente poteva gestire solamente 32 MB di memoria
  - Quando divennero comuni dischi di capacità superiore ai 100 MB, per gestire i dischi in modo da consentire file system più grandi si dovettero modificare le strutture dati e gli algoritmi usati
  - Si passò da FAT16 a FAT32
- Solaris di Sun Microsystems
  - Originariamente molte strutture dati, es. tabella dei processi e dei file aperti, avevano lunghezza fissa ed erano assegnate all'avvio del sistema
    - Una volta riempite queste tabelle, non si potevano più creare nuovi processi o aprire nuovi file
    - L'unico modo di aumentare le dimensioni di queste tabelle era la ricompilazione del kernel e il riavvio del sistema
  - Da Solaris 2 in poi, quasi tutte le strutture dati del kernel si assegnano e si rilasciano in modo dinamico, eliminando così i limiti artificiali alle prestazioni del sistema
    - Il prezzo da pagare sono algoritmi più complessi e SO più lento

Prestazioni del file system

Per migliorare le prestazioni, molti file system sono stati progettati con diverse ottimizzazioni riguardanti

- uso di memorie cache: cache del controllore & buffer/disk/block cache
- scritture sincrone/asincrone
- accesso e allocazione dei file
- riduzione del movimento del braccio del disco
- deframmentazione dei dischi

Cache del controllore

- Alcuni controllori di unità disco contengono una memoria locale di dimensioni sufficienti per la creazione di una cache interna al controllore abbastanza grande da memorizzare intere tracce
  - Qualunque richiesta di lettura di un blocco, causa la lettura di quel blocco e di buona parte della traccia su cui si trova, a seconda di quanta cache è disponibile nella memoria del controllore
- La cache del controllore del disco è indipendente dalla cache del SO, denominata buffer (o disk, o block) cache
  - Cache del controllore: memorizza blocchi che non sono stati richiesti ma è stato conveniente leggere
  - Buffer cache: memorizza blocchi esplicitamente richiesti e che il SO ipotizza possano ancora essere necessari nel breve termine

Buffer cache

- È una sezione della memoria centrale appartenente al kernel, gestita come una cache HW ma dal SO
- È la tecnica maggiormente usata per ridurre i tempi di accesso al disco
- Contiene i blocchi del disco che il SO si aspetta verranno riutilizzati a breve
- Un blocco dati viene letto dal disco solo la prima volta che viene utilizzato e poi viene mantenuto nel buffer cache fino a quando non è più necessario (o il buffer cache si riempie)
- Tutte le letture del blocco successive alla prima risultano così velocizzate

Buffer cache: lettura di un blocco

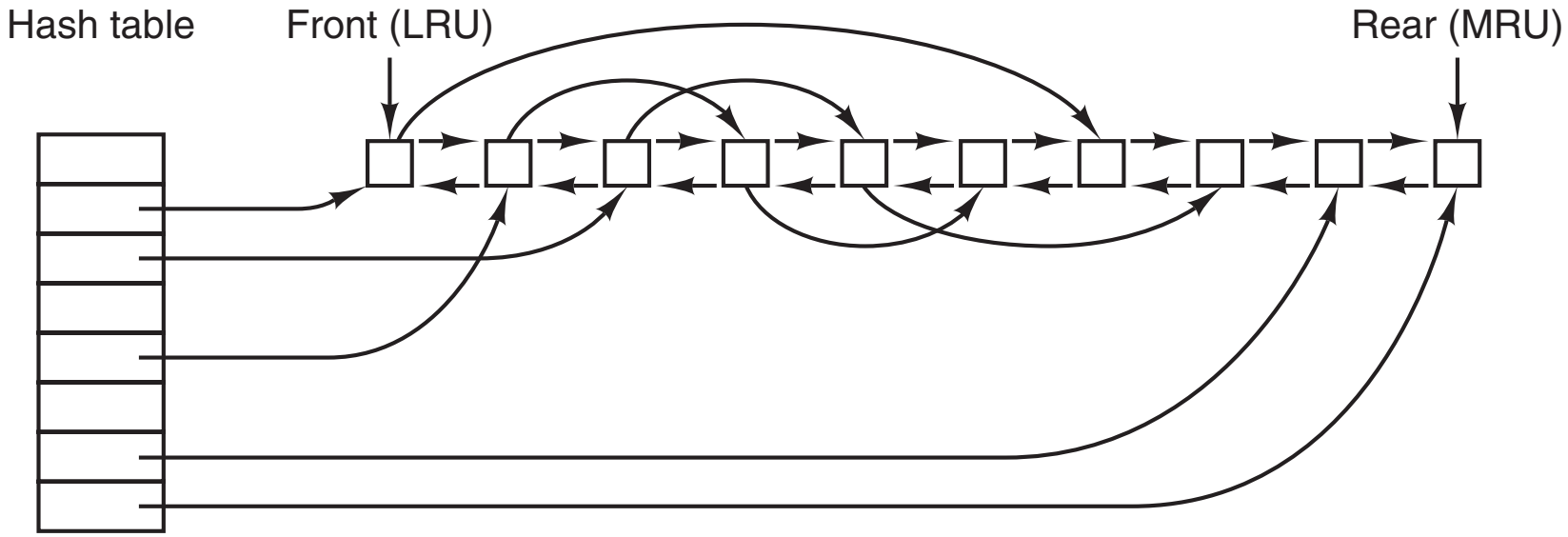
- Quando un processo utente invia una richiesta di lettura, il SO cerca i dati richiesti nel buffer cache
  - Se li trova, la richiesta viene soddisfatta senza accedere al dispositivo fisico
  - È molto probabile che ci siano poiché il SO copia intere sequenze di blocchi di dati dal disco in memoria
  - Il SO esegue anche *read-ahead* di blocchi basandosi sul presupposto che la maggior parte dei file sono acceduti in maniera sequenziale
  - Ciò fa sì che i dati seguenti a quelli contenuti in un blocco già letto siano letti molto più rapidamente dalla cache in memoria, invece di dover riaccedere al disco

Buffer cache: scrittura di un blocco

- Inserire nel buffer cache i dati da scrivere su disco è pure una buona idea
  - I dati scritti spesso vengono letti di nuovo (es. un programma sorgente viene salvato in un file, poi letto dal compilatore)
  - La scrittura dei dati nel buffer cache consente di effettuare più aggiornamenti in memoria piuttosto che dover accedere al disco ogni volta
  - Effettuando le operazioni di scrittura su disco in background, l'esecuzione del programma corrente e degli altri programmi non viene rallentata
- In un sistema tipico, circa l'85% dell'I/O dal disco può essere evitato utilizzando un buffer cache, anche se questo dipende dalla combinazione di processi in esecuzione

Buffer cache: ricerca di un blocco

Per velocizzare la ricerca, poiché nel buffer cache ci sono molti (a volte migliaia di) blocchi, questi sono organizzati con una tabella hash in cui la ricerca viene fatta sulla base del dispositivo e del numero del blocco



Tutti i blocchi sono collegati in una lista bidirezionale
in cui i blocchi sono mantenuti in ordine d'uso,
con il blocco usato meno di recente (Least Recently Used) in cima

Scritture sincrone/asincrone

Cosa succede quando un blocco nella cache è modificato?

- Scrittura sincrona: la modifica viene immediatamente riportata su disco
  - Le scritture avvengono nell'ordine in cui si rendono necessarie
  - Sicura: i dati non andranno persi se il sistema va in crash
    - Le scritture dei metadati (tabelle di directory, descrittori di file, …), tra le altre, dovrebbero essere sincrone
  - Lenta: i processi non potranno continuare fino a che le operazioni di I/O su disco non saranno completate
  - Potrebbe non essere necessaria:
    - Molte piccole modifiche apportate allo stesso blocco
    - Alcuni file sono cancellati velocemente (es. i file temporanei)

Scritture sincrone/asincrone

Cosa succede quando un blocconella cache è modificato?

- Scrittura asincrona: le modifiche non sono riportare immediatamente su disco
  - Aspettare (30 secondi?) nel caso ci siano ulteriori modifiche o il blocco venga cancellato
  - Veloce: le scritture restituiscono il controllo immediatamente
    - I dati vengono memorizzati nella cache e il controllo è restituito immediatamente al processo invocante
  - Pericolosa: può perdere dati a causa di un crash del sistema (o dell'estrazione del disco senza prima aver smontato il file system ospitato) prima che i dati siano scritti su disco permanentemente
    - Se si tratta di metadati, il file system può restare in uno stato inconsistente

Accesso e allocazione dei file

- Il metodo usato per l'accesso ai file può influenzare le prestazioni
- L'accesso sequenziale può essere ottimizzato con tecniche quali
  - *Read-ahead*: legge e inserisce nella cache il blocco richiesto ed alcune altri che lo seguono sequenzialmente (assunzione: quei blocchi saranno necessari a breve)
  - *Free-behind*: rimuove un blocco di un file non appena viene richiesto il successivo (assunzione: il blocco più recente non sarà necessario finché il file non sarà riletto dall'inizio)
- Anche il metodo usato per allocare i blocchi dei file può influenzare le prestazioni
  - Un programma che legge un file allocato in modo contiguo genererà molte richieste raggruppate, con un conseguente limitato spostamento del braccio del disco
  - Un file con allocazione concatenata o indicizzata, d'altro canto, potrebbe includere blocchi sparsi per tutto il disco, e richiedere quindi un maggiore spostamento del braccio

Riduzione del movimento del braccio del disco

Per grandi quantità di dati, la scrittura su disco tramite il file system è spesso più veloce della lettura

- Quando i dati devono essere scritti in un file sul disco
  - i blocchi sono memorizzati nella cache, che funge da buffer, mentre
  - il driver ordina la lista delle operazioni in base all'indirizzo sul disco
- Queste due azioni consentono al driver del disco di minimizzare gli spostamenti del braccio del disco e fanno sì che la scrittura dei dati rispetti i tempi ottimali per la rotazione del disco
- Quindi, a meno che le scritture richieste siano sincrone, un processo, per scrivere sul disco, scrive nella cache
  - Successivamente il sistema trasferisce i dati su disco, in maniera asincrona
- Dal punto di vista del processo, le scritture sembreranno rapidissime
- Quando i dati devono essere letti da un file sul disco
  - la tecnica read-ahead procede a qualche lettura anticipata ma, in realtà, le scritture si giovano della modalità asincrona molto più delle letture

Dischi a stato solido

- I movimenti del braccio del disco e il tempo di rotazione sono importanti per i dischi magnetici, ma non per i dischi a stato solido (SSD, solid state disk) i quali non hanno parti in movimento
- Per tali dischi, che sono costruiti con la tecnologia delle flash card, l'accesso diretto è rapido tanto quello sequenziale, e i tempi di lettura e scrittura sono notevolmente più bassi rispetto ai dischi tradizionali
- Molti dei problemi legati ai dischi tradizionali spariscono ma gli SSD soffrono dell'inconveniente che ciascun blocco può essere scritto solo un certo numero di volte
  - Quindi il SO deve prestare attenzione per fare in modo che il disco si 'consumi' in maniera uniforme

Deframmentazione dei dischi

- Un miglioramento delle prestazioni si ottiene anche spostando i file per far sì che siano contigui e mettendo tutto (o la maggior parte del) lo spazio libero in una o più grandi zone del disco (deframmentazione)
- Gli SSD non soffrono di problemi di frammentazione (anzi, la deframmentazione ridurrebbe il loro ciclo di vita)