

IL FILE SYSTEM: GESTIONE DIRECTORY, SPAZIO E PERFORMANCE

Danilo Croce

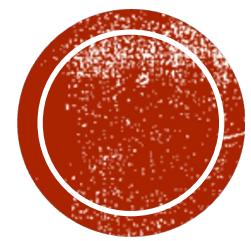
Dicembre 2023



IMPLEMENTAZIONE DEL FILE SYSTEM

- In questa lezione cercheremo di rispondere alle seguenti domande relative i File System:
 - Come memorizzare i file?
 - Come implementare le directory?
 - Come gestire lo spazio su disco?
 - Come garantire le prestazioni del file system?
 - Come garantire l'affidabilità del file system?





IMPLEMENTAZIONE DEI FILE

IMPLEMENTAZIONE DEL FILE SYSTEM

- **Come memorizzare i file?**
- Come implementare le directory?
- Come gestire lo spazio su disco?
- Come garantire le prestazioni del file system?
- Come garantire l'affidabilità del file system?



INTRODUZIONE AL LAYOUT DEL FILE SYSTEM

- **Definizione:** Il file system è il **metodo utilizzato per organizzare e memorizzare dati** sui dispositivi di memoria non volatile (dischi/SSD).
- **Importanza:** Fornisce un **modo strutturato per gestire informazioni** come file e directory su dispositivi di memoria.
- **Partizioni del Disco:** Un disco può essere **suddiviso in più partizioni**, ciascuna con un proprio file system indipendente.
- **Evoluzione:** I metodi di strutturazione del file system **variano a seconda dell'epoca del computer**, influenzando come i dati vengono gestiti e acceduti.



VECCHIO STILE - BIOS CON MBR (MASTER BOOT RECORD)

- **MBR nel BIOS:**

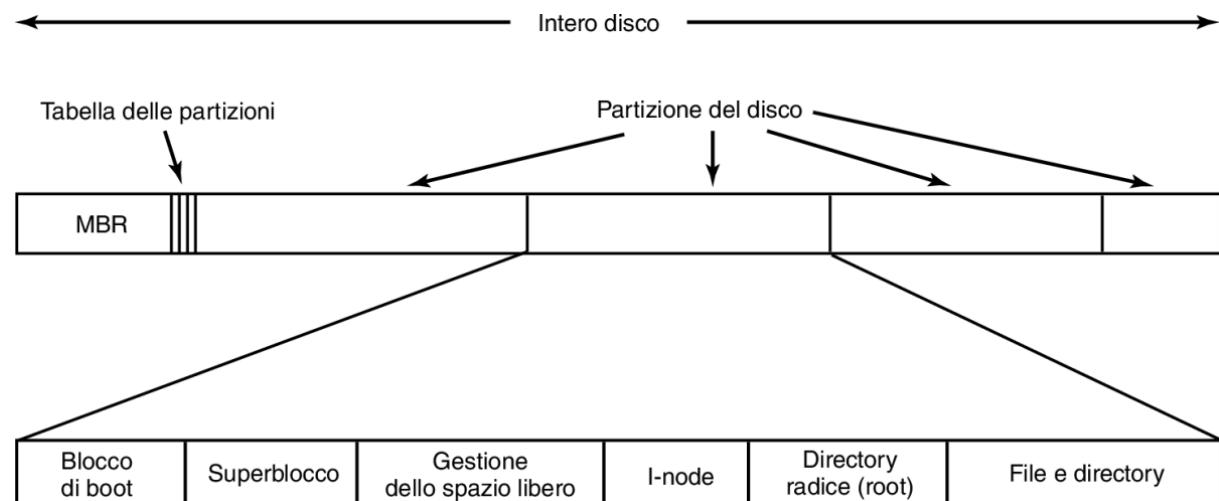
- Situato nel settore 0 del disco, l'MBR è essenziale per l'avvio del computer.
- Contiene la tabella delle partizioni con dettagli su inizio e fine di ciascuna partizione.
- Identifica la partizione attiva da cui avviare il sistema.

- **Processo di Avvio:**

- Il BIOS legge l'MBR per trovare la partizione attiva.
- Carica il **boot block** della partizione attiva per avviare il sistema operativo.

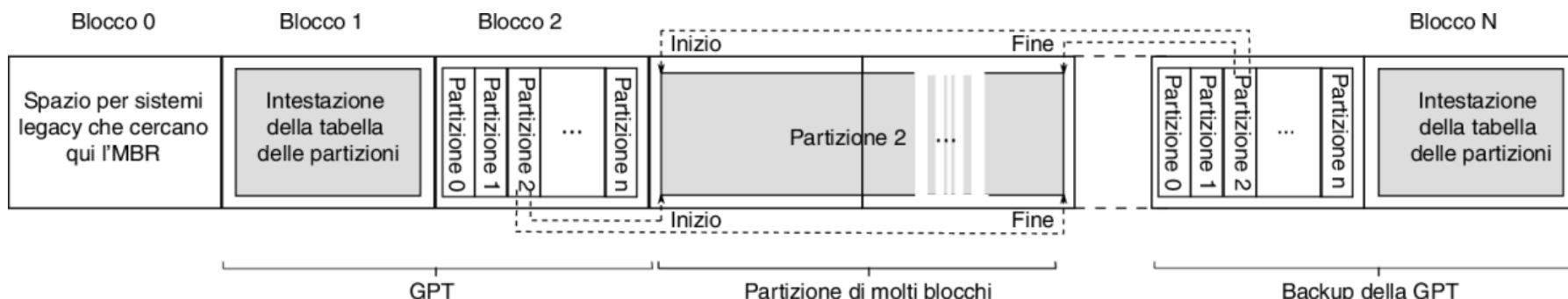
- **Layout del File System:**

- Ogni **partizione inizia con un boot block**, seguito da vari elementi di sistema.
- A destra un esempio di layout, includendo **superblocco**, **bitmap**, **I-node** e **directory radice** (vedi slide successive)



NUOVA SCUOLA - UEFI (UNIFIED EXTENSIBLE FIRMWARE INTERFACE)

- **UEFI vs BIOS:**
 - UEFI supera le limitazioni del BIOS tradizionale (max 4 partizioni primarie e 2TB per partizione), supportando dischi di dimensioni maggiori e avvio più veloce.
 - Introduce una maggiore flessibilità e compatibilità con diverse architetture hardware.
- **GPT (GUID Partition Table):**
 - UEFI utilizza la GPT per gestire informazioni più dettagliate sulle partizioni.
 - GPT supporta dischi fino a 8 ZiB e contiene un backup nell'ultimo blocco del disco.
- **EFI System Partition:**
 - Utilizza un file system FAT per memorizzare i programmi di avvio.
 - Il firmware UEFI legge la GPT e carica i file dalla partizione EFI per avviare il sistema.



IMPLEMENTAZIONE DEI FILE NEI FILE SYSTEM

- **Obiettivo Principale:** Gestire l'associazione tra i file e i blocchi del disco su cui sono memorizzati.
- **Importanza:** Fondamentale per assicurare l'integrità, l'accesso efficiente e la gestione dello spazio su disco.
- **Varietà di Metodi:** Diversi sistemi operativi adottano approcci differenti per questa associazione.
 - **Basato su**
 - Metodi basati su indici
 - Liste concatenate
 - Bitmap
 - Strutture ad albero
- **Focus:** Analisi dei vari metodi e delle loro caratteristiche specifiche nel contesto dei diversi file system.



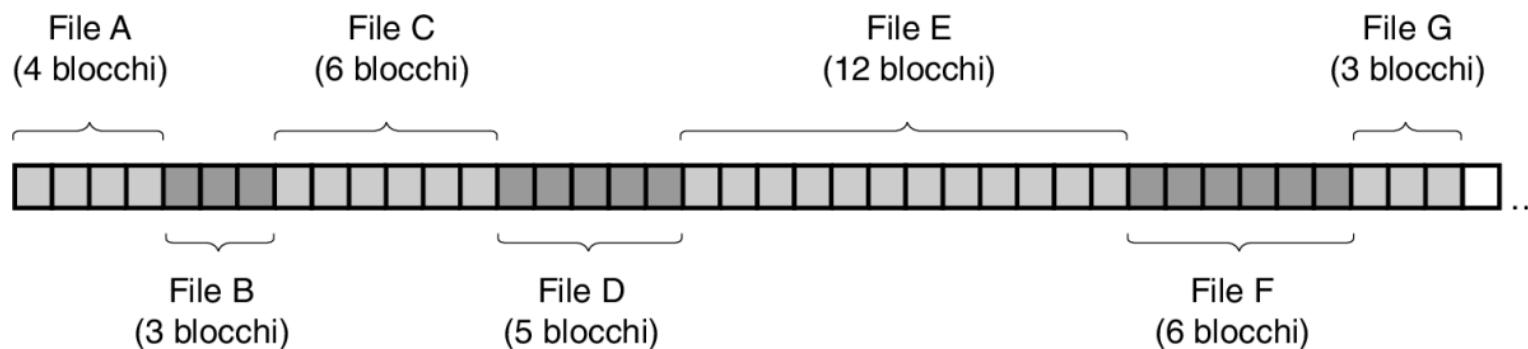
ALLOCAZIONE CONTIGUA NEL FILE SYSTEM

- **Concetto di Allocazione Contigua:**

- I file sono memorizzati come **sequenze contigue di blocchi sul disco**.
- **Esempio:** un file di 50 KB su un disco con blocchi da 1 KB occupa 50 blocchi consecutivi.
- In basso: l'allocazione contigua di sette file su disco.

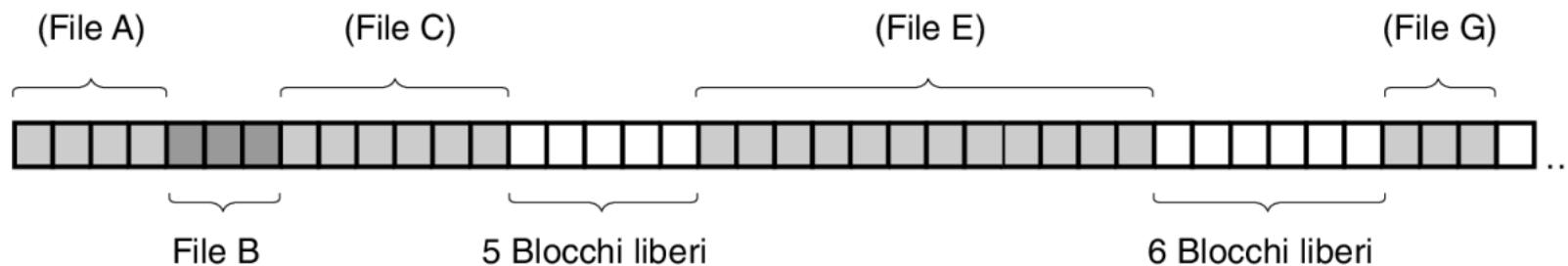
- **Implementazione e Vantaggi:**

- **Semplice da implementare:** richiede solo l'indirizzo del primo blocco e il numero totale di blocchi.
- **Alta efficienza di lettura:** l'intero file può essere letto in una sola operazione, senza ritardi.



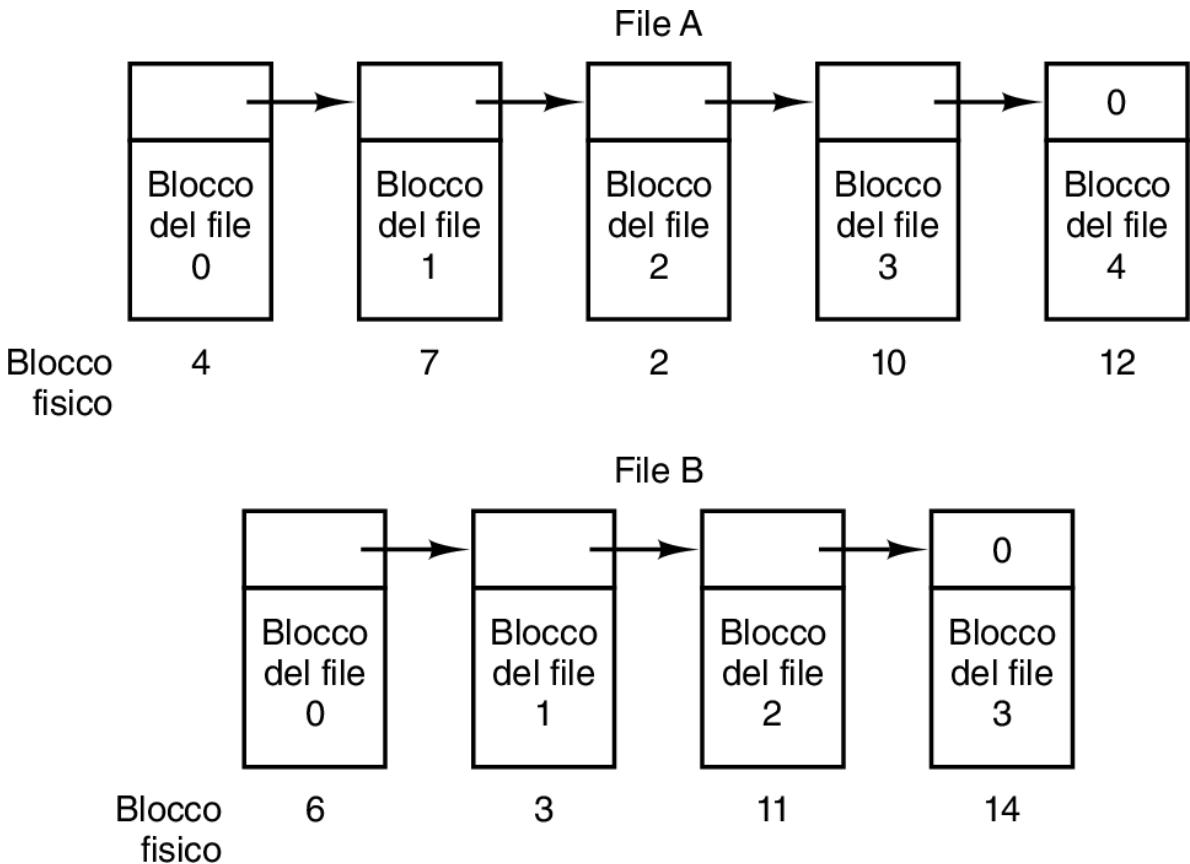
PROBLEMI E LIMITAZIONI DELL'ALLOCAZIONE CONTIGUA

- **Frammentazione del Disco:**
 - Col passare del tempo, i dischi si frammentano a causa della rimozione di file.
 - **In basso:** La frammentazione lascia intervalli di blocchi liberi
 - **Problema di allocazione di nuovi file in spazi liberi** frammentati.
- **Gestione dello Spazio Libero:**
 - Richiede una lista di spazi liberi e la conoscenza della dimensione finale dei nuovi file.
 - Problemi nella previsione della dimensione del file e nella ricerca di spazi adeguati.
- **Implicazioni Pratiche:**
 - Difficoltà nell'aggiungere nuovi file in un disco frammentato.
 - Necessità di compattazione del disco o di gestione intelligente dello spazio libero.



ALLOCAZIONE A LISTE CONCATENATE - CONCETTI BASE

- **Principio di Allocazione:**
 - I file sono organizzati come liste concatenate di blocchi su disco.
 - Ogni blocco contiene una parte di dati e un puntatore al blocco successivo.
- **Gestione dello Spazio:**
 - Efficiente utilizzo di tutti i blocchi disponibili sul disco.
 - **Minima frammentazione esterna:** riduce lo spreco di spazio non utilizzato.
- **Struttura delle Voci di Directory:**
 - Ogni voce di directory traccia solo l'indirizzo del primo blocco di un file.
 - Il percorso completo di un file è costruito seguendo i puntatori da un blocco all'altro.



ALLOCAZIONE A LISTE CONCATENATE - PRESTAZIONI E LIMITAZIONI

- **Accesso ai Dati:**
 - **Accesso Sequenziale:** Leggere un file è efficiente, procedendo blocco per blocco.
 - **Accesso Casuale (seek):** Estremamente lento, richiede la lettura sequenziale di ogni blocco precedente.
- **Dimensione dei Blocchi e Efficienza:**
 - Ogni blocco ha una dimensione effettiva ridotta a causa dello spazio occupato dai puntatori.
 - Letture e scritture di dimensioni standard (potenze di due) possono essere meno efficienti.
- **Implicazioni Pratiche:**
 - Il metodo offre un'elevata efficienza nello sfruttamento dello spazio su disco ...
 - MA introduce complessità e rallentamenti nelle operazioni di accesso casuale.
 - Adatto per file a cui si accede principalmente in modo sequenziale.



ALLOCAZIONE A LISTE CONCATENATE CON FAT

- **Ottimizzazione dell'Allocazione a Liste Concatenate:**
 - Eliminazione degli svantaggi dell'allocazione a liste concatenate spostando i puntatori in una tabella di memoria (**FAT - File Allocation Table**).
 - Ogni blocco del disco è rappresentato come una voce nella FAT... **IN MEMORIA RAM**
- **Struttura della FAT:**
 - Contiene la sequenza dei blocchi di ciascun file.
 - Esempio:
 - File A utilizza i blocchi 4, 7, 2, 10, 12;
 - File B i blocchi 6, 3, 11, 14.
 - Sequenze terminate da un indicatore speciale (es. -1) per marcare la fine.
 - In memoria principale
- **Vantaggi della FAT:**
 - L'intero blocco è disponibile per i dati, ottimizzando lo spazio.
 - Accesso casuale semplificato: la sequenza dei blocchi è interamente in memoria.

Blocco fisico	
0	
1	
2	10
3	11
4	7
5	
6	3
7	2
8	
9	
10	12
11	14
12	-1
13	
14	-1
15	

Il file A inizia qui ←

Il file B inizia qui ←

← Blocco non utilizzato



LIMITAZIONI E APPLICAZIONI DELLA FAT

- **Gestione in Memoria:**

- La FAT deve essere **mantenuta interamente in memoria principale**.
- Richiede una quantità significativa di memoria: per un disco da 1 TB con blocchi da 1 KB, la FAT richiederebbe fino a 3 GB di RAM.

- **Implicazioni di Efficienza:**

- Lo **spazio e la velocità influenzano la dimensione** della voce della FAT (da 3 a 4 byte per voce).
- L'approccio non è ottimale per dischi di grandi dimensioni a causa dell'elevato consumo di memoria.

- **Utilizzo Pratico:**

- Originariamente implementato in MS-DOS, ancora supportato da Windows e UEFI.
- **Comunemente usato** in dispositivi portatili come **schede SD** in fotocamere, lettori musicali e altri dispositivi elettronici.



I-NODE

- **I-node (Index Node) nei File System UNIX-like**
 - **Definizione:** Struttura dati fondamentale nei file system come ext2/ext3/ext4 in Linux.
 - **Contenuto:** Contiene tutte le informazioni su un file, esclusi il nome e il contenuto. Include metadati come permessi, proprietario, timestamp e indirizzi dei blocchi di dati.
 - **Funzione:** Ogni file e directory è rappresentato da un I-node univoco, indicizzato in una tabella di I-node.
- **Confronto con FAT (File Allocation Table)**
 - **Gestione dei File:**
 - FAT si basa su una tabella di allocazione per tracciare i file, mentre i sistemi basati su I-node utilizzano una tabella di I-node.
 - I sistemi I-node separano le informazioni sul file dalla sua posizione fisica sul disco.
 - **Informazioni sui File:**
 - FAT fornisce meno dettagli sui file, concentrandosi principalmente sull'allocazione dello spazio.
 - I sistemi I-node offrono una gestione più dettagliata dei metadati, inclusi permessi e proprietà.
 - **Efficienza e Performance:**
 - I file system basati su I-node tendono a essere più efficienti e performanti, specialmente su dischi di grandi dimensioni, grazie alla loro struttura avanzata.



FUNZIONAMENTO E VANTAGGI DEGLI I-NODE

- **I-node (Index-Node) nei File System**

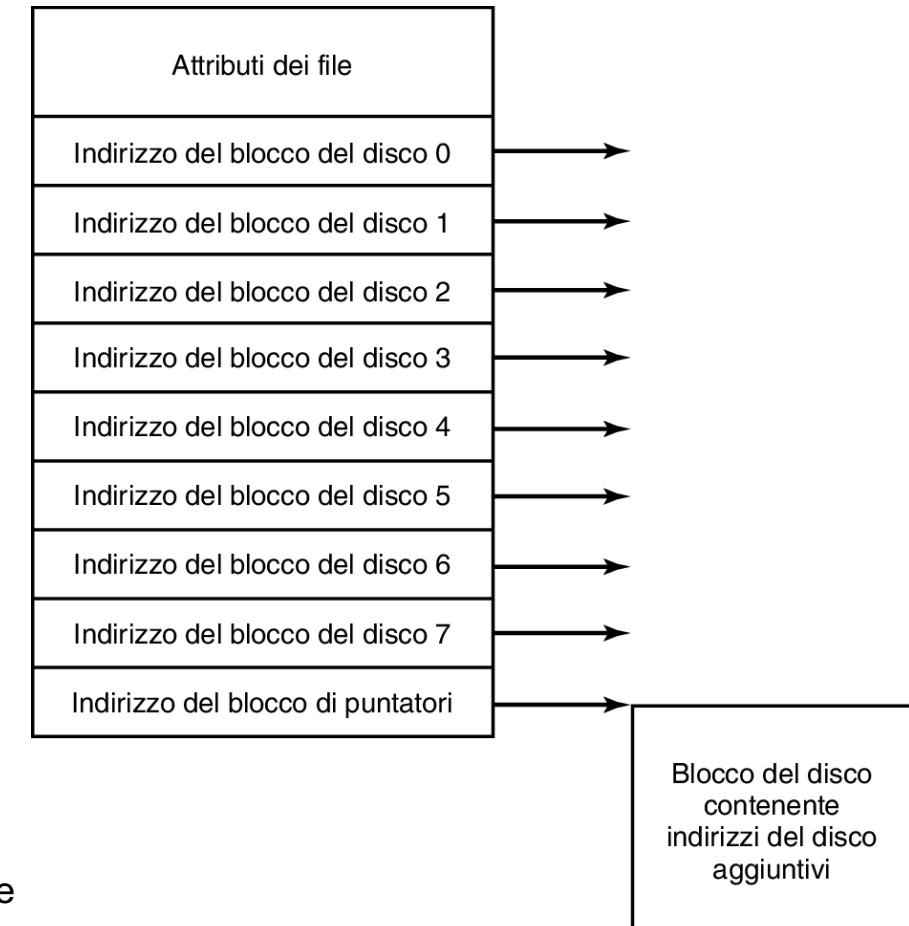
- Gli I-node sono strutture dati che elencano gli attributi e gli indirizzi dei blocchi dei file.
- Ogni I-node rappresenta un file, fornendo un metodo efficiente per trovare tutti i suoi blocchi di dati.

- **Efficienza della Memoria con I-node**

- **Solo gli I-node dei file aperti sono mantenuti in memoria**, riducendo significativamente l'utilizzo della memoria.
- L'array degli I-node in memoria è proporzionale al numero di file aperti, non alla dimensione del disco.

- **Esempio e Struttura**

- Ogni I-node ha una dimensione fissa e contiene informazioni quali dimensione del file, permessi, proprietario, e indirizzi dei blocchi di dati.



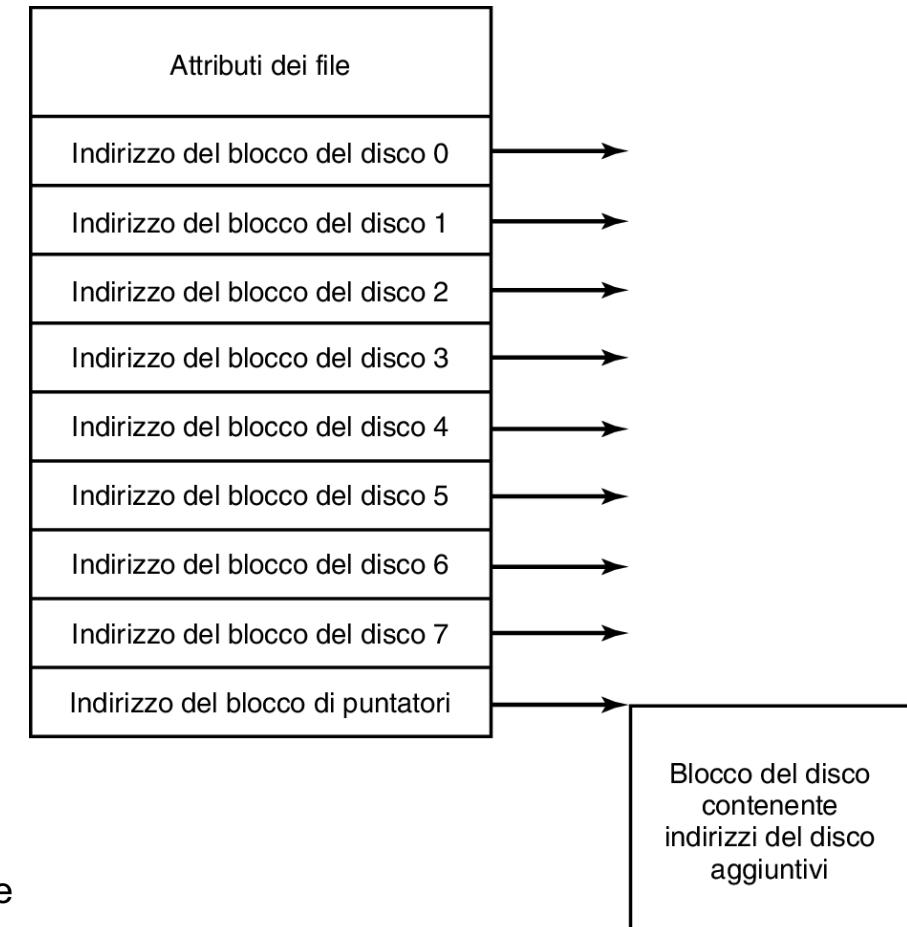
GESTIONE DEI FILE DI GRANDI DIMENSIONI E CONFRONTO CON NTFS

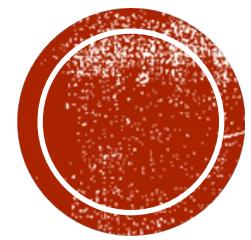
- **Gestione File di Grandi Dimensioni**

- Gli I-node hanno uno spazio limitato per gli indirizzi del disco.
- Per file che superano il limite, uno degli indirizzi nell'I-node punta a un blocco contenente ulteriori indirizzi di blocchi di dati.
- Questo sistema permette di gestire file di dimensioni molto grandi con efficacia.

- **I-node nei File System UNIX e Windows NTFS**

- Gli I-node sono un concetto fondamentale in UNIX e nei suoi file system derivati.
- NTFS, il file system di Windows, utilizza una struttura simile con I-node più grandi che possono contenere file di piccole dimensioni all'interno dell'I-node stesso.





IMPLEMENTAZIONE DELLE DIRECTORY

IMPLEMENTAZIONE DEL FILE SYSTEM

- Come memorizzare i file?
- **Come implementare le directory?**
- Come gestire lo spazio su disco?
- Come garantire le prestazioni del file system?
- Come garantire l'affidabilità del file system?



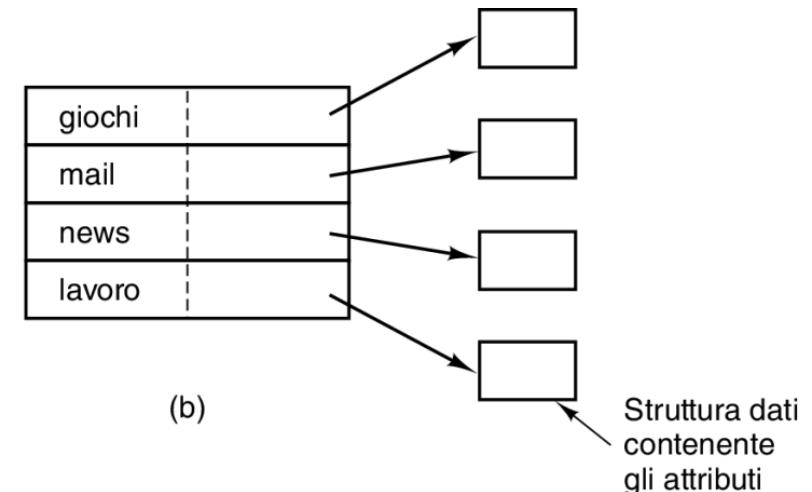
IMPLEMENTAZIONE DELLE DIRECTORY - CONCETTI DI BASE

- **Funzione Principale:** Le directory mappano i nomi ASCII dei file sulle informazioni necessarie per localizzare i dati su disco.
- **Metodi di Allocazione:** Variano a seconda del sistema operativo, includendo indirizzi di blocchi contigui, il primo blocco nelle liste concatenate, o i numeri degli I-node.

- a) Una semplice directory contenente voci a dimensione fissa con gli indirizzi del disco e gli attributi nella voce della directory.
- b) Una directory in cui ogni voce fa soltanto riferimento a un I-node.

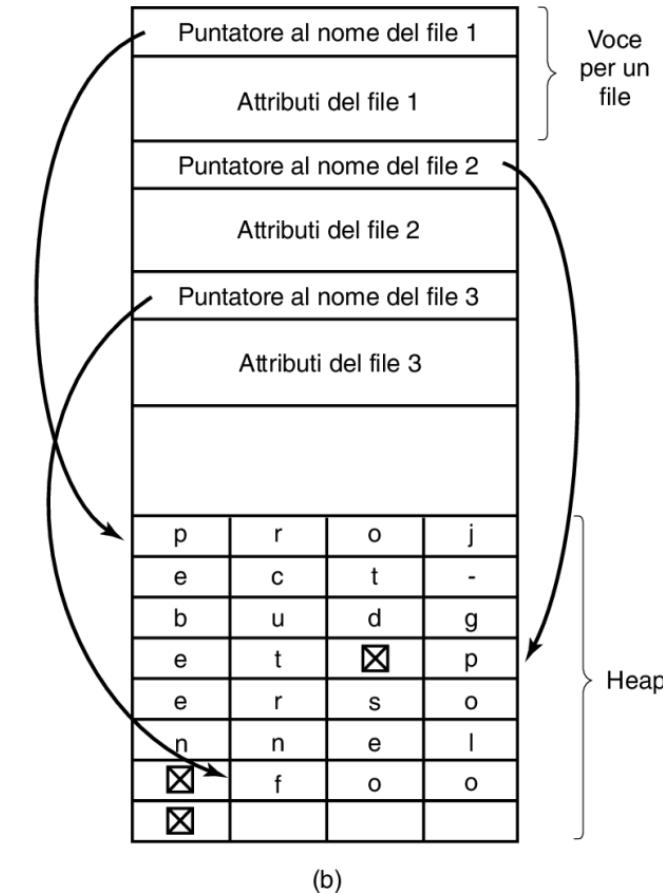
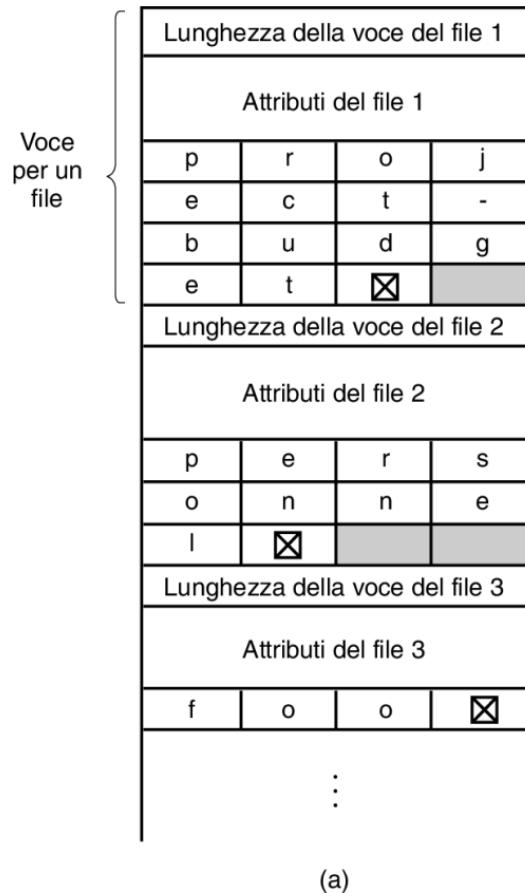
giochi	attributi
mail	attributi
news	attributi
lavoro	attributi

(a)



IMPLEMENTAZIONE DELLE DIRECTORY - GESTIONE DEI NOMI DEI FILE

- **Nomi di File Variabili:** Supporto per nomi di file di lunghezza variabile, con un limite tipico di 255 caratteri.
 - **Strutture di Directory:**
 - a) **Voci di Directory di Lunghezza Variabile:** l'header di lunghezza fissa seguito dal nome del file.
 - b) **Gestione degli Heap:** le voci di directory di lunghezza fissa con nomi dei file gestiti in uno heap separato.
 - **Efficienza e Limitazioni:** gestiscono i nomi di lunghezza variabile ma presentano sfide
 - nella gestione degli spazi vuoti (un file viene cancellato)



OTTIMIZZAZIONE DELLA RICERCA NELLE DIRECTORY CON TABELLE DI HASH

- **Ricerca Lineare Tradizionale:**

- Inizialmente, i file in una directory venivano cercati linearmente dall'inizio alla fine.
- Questo metodo può diventare lento in directory con un gran numero di file.

- **Uso delle Tabelle di Hash:**

- Introduzione di tabelle di hash in ogni directory per accelerare il processo di ricerca.
- Il nome di un file è sottoposto a hashing per generare un indice nell'intervallo da 0 a $n-1$.
- La voce corrispondente nella tabella di hash indica il punto di partenza per la ricerca del file.

- **Gestione delle Collisioni:**

- Creazione di liste concatenate per gestire più file che condividono lo stesso valore hash.
- La ricerca verifica tutte le voci nella catena per trovare il file desiderato.



GESTIONE DELLA CACHE PER RICERCHE EFFICIENTI

- **Caching delle Ricerche:**

- Salvataggio dei risultati di ricerche comuni nella cache per accesso rapido.
- Prima di avviare una ricerca, si verifica se il file si trova nella cache.

- **Vantaggi e Limitazioni:**

- La cache aumenta l'efficienza delle ricerche, specialmente per file frequentemente richiesti.
- Efficace quando la maggior parte delle ricerche riguarda un numero limitato di file.

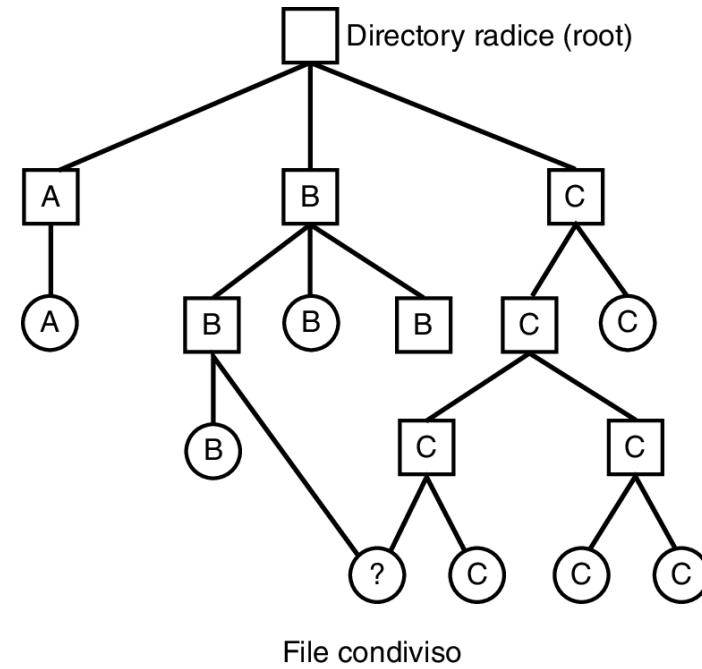
- **Complessità Amministrativa:**

- L'uso di tabelle di hash e cache introduce una maggiore complessità nella gestione delle directory.
- Più adatto a sistemi con directory molto estese, dove si prevede un elevato numero di file.



FILE CONDIVISI E LINK NEI FILE SYSTEM

- **File Condivisi:** Essenziali in ambienti collaborativi per permettere a più utenti di lavorare sugli stessi file.
- **Tipi di Link:**
 - **Hard Link:** Puntano direttamente all'I-node di un file condiviso.
 - **Link Simbolico (Soft Link):** Puntano al nome di un file piuttosto che all'I-node.
- **Gestione degli Hard Link:**
 - Un file con hard link **viene rimosso solo quando non ci sono più riferimenti ad esso.**
 - **Efficienza di spazio:** una sola voce di directory per ciascun hard link.
 - **Ideali per la gestione di file condivisi** tra più proprietari.



Un file system con un file condiviso tra due utenti.

- Esempio: Un file di un utente può essere presente anche nella directory di un altro.



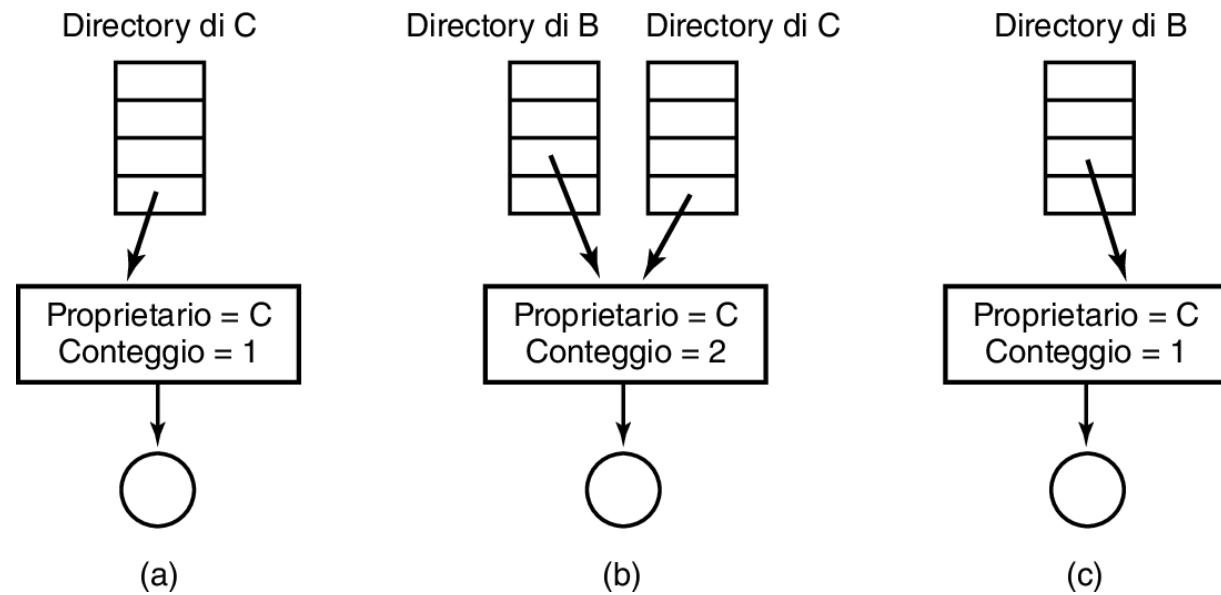
GESTIONE E PROBLEMI DEI FILE CONDIVISI: HARD LINKS

- **Vantaggi degli Hard Link:**

- **Spazio-efficienti:** usano un solo I-node indipendentemente dal numero di link.
- Gestione trasversale degli utenti: il file rimane accessibile finché almeno un hard link è presente.

- **Problemi e Limitazioni:**

- Il file permane fino all'eliminazione di tutti gli hard link, **potenzialmente causando confusione sulla proprietà del file.**
- **A destra:** Illustrano come i file condivisi sono gestiti nel file system e le implicazioni dell'eliminazione di hard link.



- a) Situazione prima del link.
- b) Dopo la creazione del link.
- c) Dopo che il proprietario originale elimina il file.



GESTIONE E PROBLEMI DEI FILE CONDIVISI: LINK SIMBOLICI

▪ **Vantaggi dei Link Simbolici:**

- **Maggiore flessibilità:** possono riferirsi a nomi di file oltre i confini del file system e su macchine remote.
- **Meno efficienti in termini di spazio:** richiedono un I-node per ogni link simbolico.

▪ **Problemi e Limitazioni:**

- Link simbolici diventano **invalidi** alla rimozione del file originale.
- **Overhead maggiore** nella risoluzione del percorso rispetto agli hard link.
- **Gestione più complessa**, ma con benefici in termini di flessibilità e organizzazione.

▪ **Problemi Comuni:**

- **I file con più percorsi** possono essere **processati più volte da programmi di backup o di ricerca.**
 - Rischio di **duplicazione dei file** su unità di backup.
- Necessità di software avanzato per gestire correttamente i file condivisi e i loro link.





SPAZIO DISCO

IMPLEMENTAZIONE DEL FILE SYSTEM

- Come memorizzare i file?
- Come implementare le directory?
- **Come gestire lo spazio su disco?**
- Come garantire le prestazioni del file system?
- Come garantire l'affidabilità del file system?



GESTIONE DELLO SPAZIO SU DISCO - CONSIDERAZIONI GENERALI

- **Memorizzazione dei File:** I file sono generalmente memorizzati su disco, e ci sono due modi principali per farlo:
 - **allocazione contigua**
 - **suddivisione in blocchi non contigui.**
- **Allocazione Contigua vs Blocchi:**
 - **Allocazione Contigua:** Richiede spostamenti di file se le loro dimensioni aumentano, simile alla gestione della memoria con segmentazione.
 - **Blocchi Non Contigui:** I file vengono spezzettati in blocchi di dimensioni fisse, consentendo una maggiore flessibilità e un migliore utilizzo dello spazio su disco.
- **Dimensione dei Blocchi:**
 - La scelta della dimensione dei blocchi è un compromesso tra spazio ed efficienza.
 - La dimensione comune di 4 KB è un compromesso tra lo spazio su disco e le prestazioni di trasferimento dei dati.



EFFICIENZA E PRESTAZIONI - ANALISI DEL COMPROMESSO

▪ Prestazioni di Trasferimento Dati:

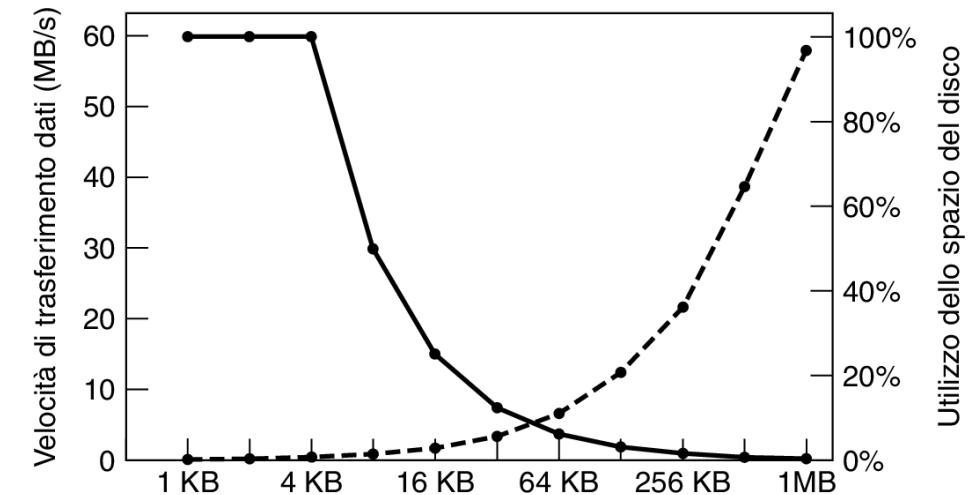
- I dischi magnetici con blocchi più grandi consentono il trasferimento di più dati per operazione di lettura/scrittura.
- **MA:** blocchi grandi portano a spreco di memoria

▪ Efficienza dello Spazio:

- Blocchi più piccoli minimizzano lo spreco di spazio con file piccoli.
- **MA:** significa distribuire la maggior parte dei file su più blocchi e incorrere in più ricerche e ritardi (*addirittura di rotazione nei dischi*) per leggerli
- L'efficienza dello spazio diminuisce con l'aumento della dimensione dei blocchi (oltre la dimensione media dei file).

▪ Conflitto tra Prestazioni ed Efficienza:

- Le prestazioni migliori richiedono blocchi più grandi, ma ciò può comportare uno spreco maggiore di spazio su disco.
- La scelta ottimale della dimensione del blocco deve bilanciare il tempo di trasferimento e l'efficienza dello spazio. In genere 4 KB.



La curva tratteggiata (con scala a sinistra) indica la velocità di trasferimento dati di un disco. La curva continua (scala a destra) esprime l'efficienza nell'utilizzo dello spazio del disco. Tutti i file sono di 4 KB.



IMPLICAZIONI PER DISCHI MAGNETICI E MEMORIA FLASH

- **Dischi Magnetici:**

- La scelta delle dimensioni dei blocchi è influenzata dal tempo di ricerca e dal ritardo di rotazione.
- Con l'aumento della dimensione dei blocchi, si incrementa la velocità di trasferimento ma si riduce l'efficienza dello spazio.

- **Memoria Flash:**

- Diversamente dai dischi magnetici, la memoria flash può avere sprechi di spazio sia con blocchi grandi che piccoli a causa delle dimensioni fisse delle pagine flash.

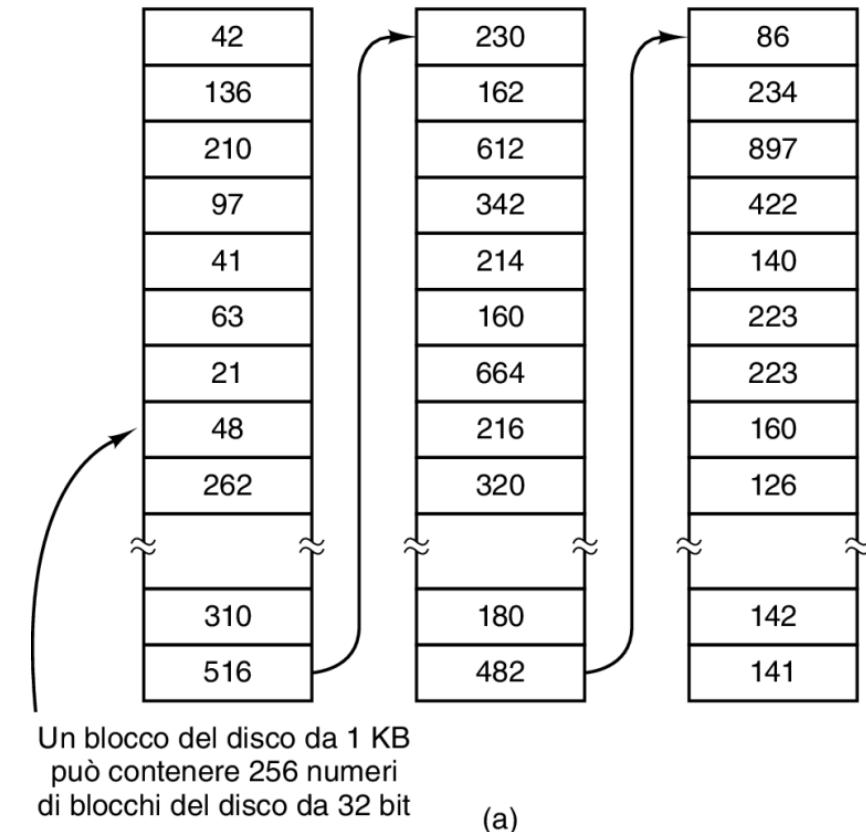
- **Tendenze Attuali:**

- Con l'incremento della capacità dei dischi (TB), potrebbe essere vantaggioso considerare blocchi più grandi per accettare una minore efficienza dello spazio in cambio di prestazioni migliori.



GESTIONE DEI BLOCCHI LIBERI NEL DISCO

- Come tenere traccia dei blocchi liberi?
- **Metodo 1: Lista Concatenata**
 - Utilizzo di una lista concatenata di blocchi del disco.
 - Nella lista solo i blocchi liberi
 - Si usano gli stessi blocchi del disco liberi per ospitare le liste
 - Ogni blocco contiene numeri di blocchi del disco liberi.
 - **Esempio:** Con blocchi da 1 KB e numeri da 32 bit, ogni blocco lista può contenere numeri di 255 blocchi liberi (1 blocco riservato al puntatore del blocco successivo).
 - Per 1 TB, servono 4 milioni di entrate
 - **Efficienza:** Richiede meno spazio solo se il disco è quasi pieno.



GESTIONE DEI BLOCCHI LIBERI NEL DISCO

- *Come tenere traccia dei blocchi liberi?*
- **Metodo 2: Bitmap**
 - Utilizzo di una bitmap per tracciare i blocchi liberi.
 - Un bit per ogni blocco del disco
 - 1 indica libero
 - 0 indica allocato.
 - **Esempio:** Per un disco da 1 TB, serve una bitmap da 1 miliardo di bit.
 - **Efficienza:** Richiede meno spazio rispetto alla lista concatenata, tranne in dischi quasi pieni
 - La lista concatenata deve considerare meno blocchi liberi e quindi ha meno blocchi occupati

1001101101101100
0110110111110111
1010110110110110
0110110110111011
1110111011101111
1101101010001111
0000111011010111
1011101101101111
1100100011101111
~
0111011101110111
1101111101110111

Una bitmap

(b)



OTTIMIZZAZIONE E PROBLEMI NELL'USO DELLA LISTA CONCATENATA

- **Modifiche alla Lista dei Blocchi Liberi**

- Tracciamento di serie di blocchi consecutivi anziché blocchi singoli.
 - A ciascun blocco può essere associato un conteggio a 8, 16 o 32 bit, che rappresenta il numero di blocchi liberi consecutivi.
 - Nell'ipotesi migliore, un disco fondamentalmente vuoto è rappresentato da due numeri:
 - l'indirizzo del primo blocco libero
 - seguito dal conteggio dei blocchi liberi.
- **Efficienza:** Migliore per dischi quasi vuoti; meno efficiente per dischi frammentati.
 - Se il conteggio viene memorizzato c'è overhead eccessivo dovuto a indice e conteggio

- **Sfide nella Progettazione di Sistemi Operativi**

- Scelta della struttura dati ottimale senza dati anticipati sull'uso del sistema.
- Esempi: Differenze nella gestione dei file e nell'utilizzo del disco tra diversi dispositivi e ambienti.



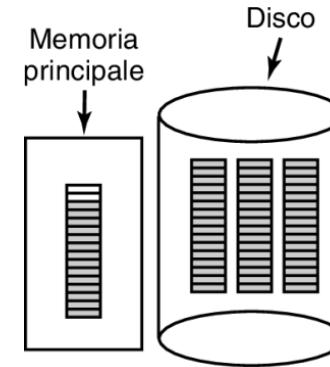
GESTIONE DEI BLOCCHI LIBERI CON LISTA DI PUNTATORI

• Funzionamento della Free List

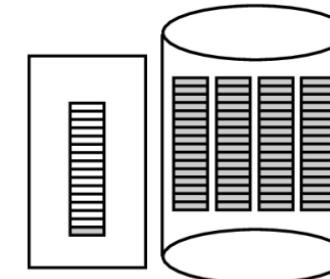
- La gestione dei blocchi liberi può utilizzare una lista concatenata di puntatori, nota come "free list".
- Solo un blocco di puntatori è mantenuto in memoria contemporaneamente, ottimizzando così l'utilizzo della memoria.
- Quando si crea un file, i blocchi necessari vengono allocati dai puntatori disponibili nel blocco in memoria.

• Ottimizzazione del Flusso di I/O

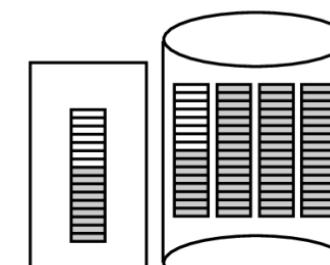
- Questo metodo evita I/O su disco inutili mantenendo una lista di blocchi liberi direttamente accessibili in memoria.
- Al riempimento del blocco di puntatori in memoria, un nuovo blocco viene letto da disco per proseguire con le operazioni.



(a)



(b)



(c)

a) Un blocco quasi pieno di puntatori a blocchi del disco liberi in memoria e tre blocchi di puntatori su disco.

b) Situazione dopo aver liberato un file da 3 blocchi.

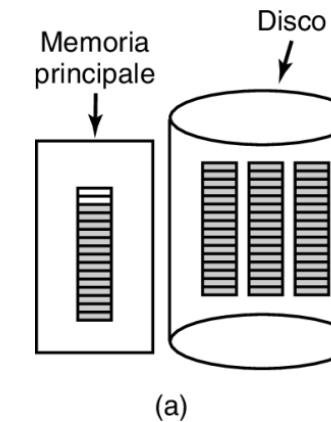
c) Una strategia alternativa per la gestione dei tre blocchi liberi. Le voci in grigio rappresentano i puntatori a blocchi del disco liberi.



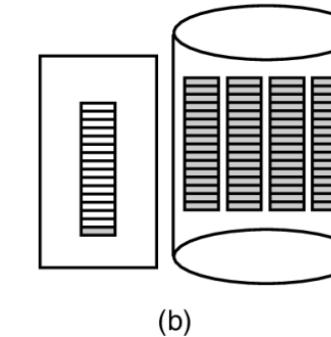
GESTIONE DEI PUNTATORI E APPROCCI ALTERNATIVI

• Efficienza e File Temporanei

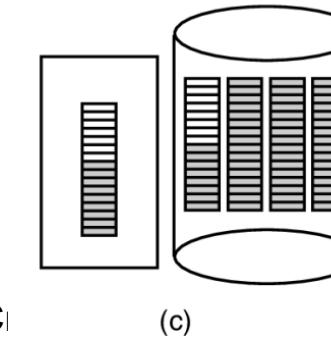
- La presenza di file temporanei può portare a frequenti operazioni di I/O su disco se il blocco di puntatori in memoria è quasi pieno.
 - Vedi Fig a) VS Fig b)
- Una strategia alternativa prevede di dividere il blocco pieno di puntatori per gestire meglio i blocchi liberi senza I/O su disco.
 - Vedi Fig a) VS Fig c)



(a)



(b)



(c)

a) Un blocco quasi pieno di puntatori a blocchi del disco liberi in memoria e tre blocchi di puntatori su disco.

b) Situazione dopo aver liberato un file da 3 blocchi.

c) Una strategia alternativa per la gestione dei tre blocchi liberi. Le voci in grigio rappresentano i puntatori a blocchi del disco liberi.



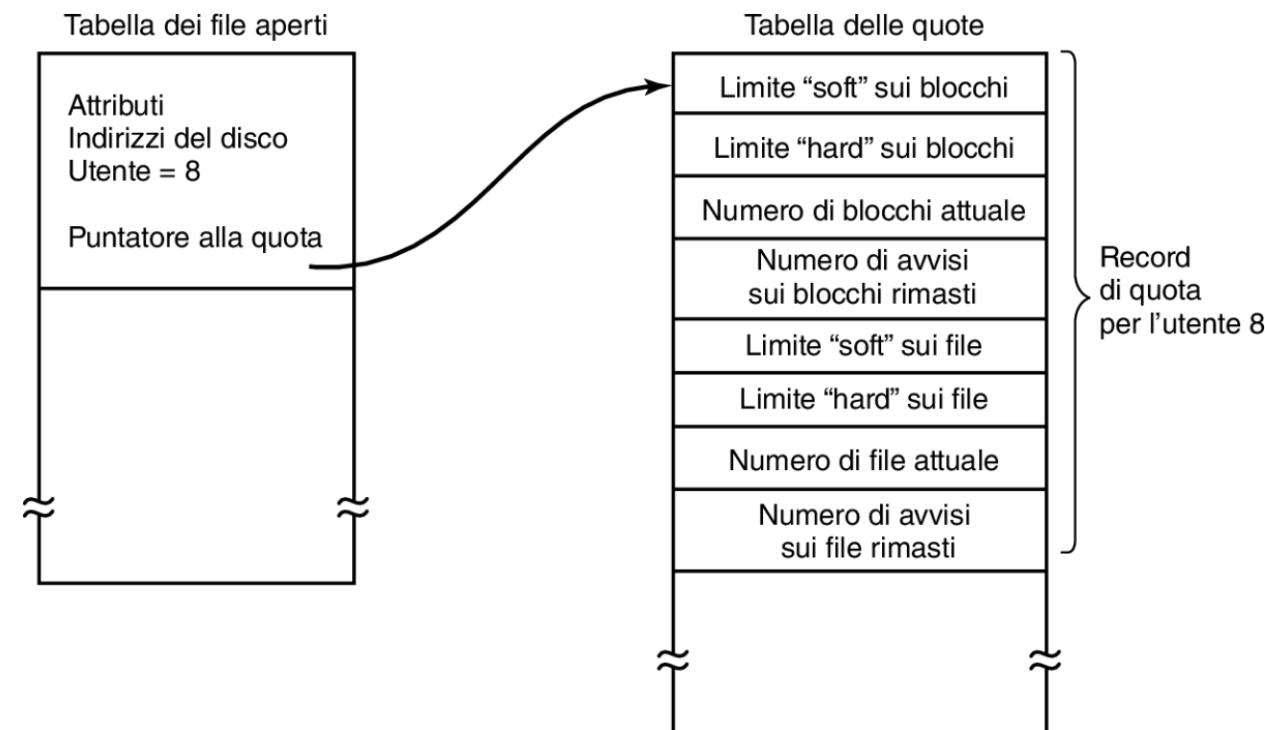
MECCANISMO DI QUOTA DEL DISCO NEI SISTEMI OPERATIVI MULTIUTENTE

• Assegnazione delle Quote

- Amministratore di sistema assegna a ogni utente un numero massimo di file e blocchi.
- Il sistema operativo controlla che gli utenti non superino la loro quota.

• Gestione e Controllo delle Quote

- Ogni apertura di file coinvolge il controllo degli attributi e degli indirizzi del disco.
- Attributi includono l'identificazione del proprietario del file.
- Incrementi della dimensione del file sono contabilizzati nella quota del proprietario.



CONTROLLI E LIMITI DELLE QUOTE

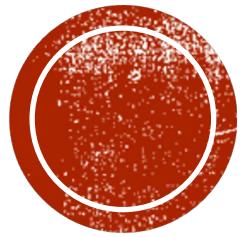
- **Tabelle delle Quote**

- Tabella separata tiene traccia delle quote di ogni utente con file aperti.
- Record delle quote aggiornati e riscritti sul file delle quote alla chiusura dei file.

- **Limiti delle Quote e Consequenze**

- Limiti "soft" e "hard" per le quote:
 - "soft" può essere temporaneamente superato, "hard" mai.
- Violazioni dei limiti "hard" o ignorare gli avvisi dei limiti "soft" possono portare alla restrizione dell'accesso.
- Gli utenti possono superare temporaneamente i limiti "soft" durante una sessione di lavoro, ma devono rientrare nei limiti prima di scollegarsi.





INTERMEZZO COME È ORGANIZZATO EXT2

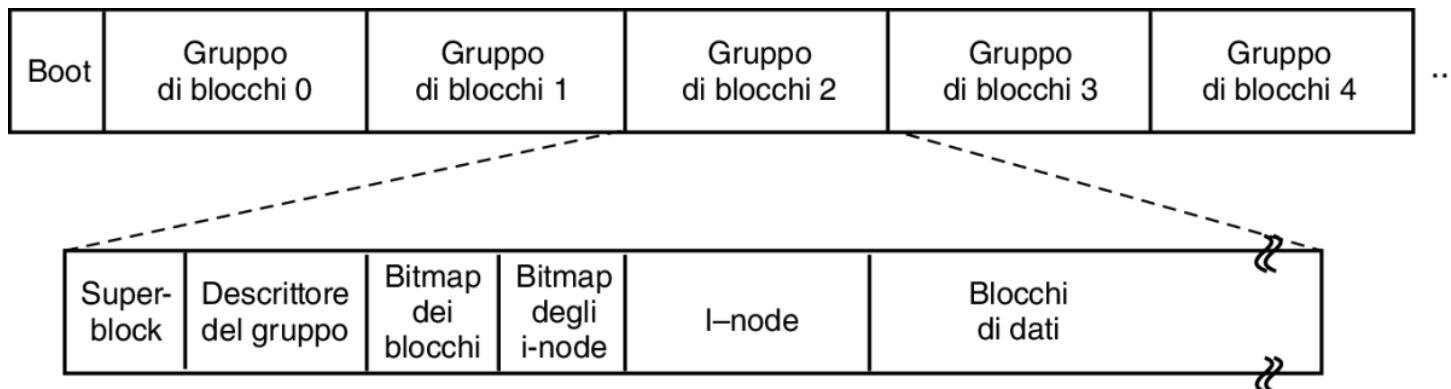
ORGANIZZAZIONE E GESTIONE NEL FILE SYSTEM EXT2 (CAP 10.6.3 LIBRO)

- **Componenti Chiave del File System Ext2**

- **Superblocco**: Informazioni sul layout, numero di I-node, e blocchi del disco.
- **Descrittore del Gruppo**: Dettagli sui blocchi liberi, I-node, e posizione delle bitmap.
- **Bitmap**: Traccia i blocchi liberi e gli I-node liberi, seguendo il design di MINIX 1.

- **I-node e Blocchi di Dati**

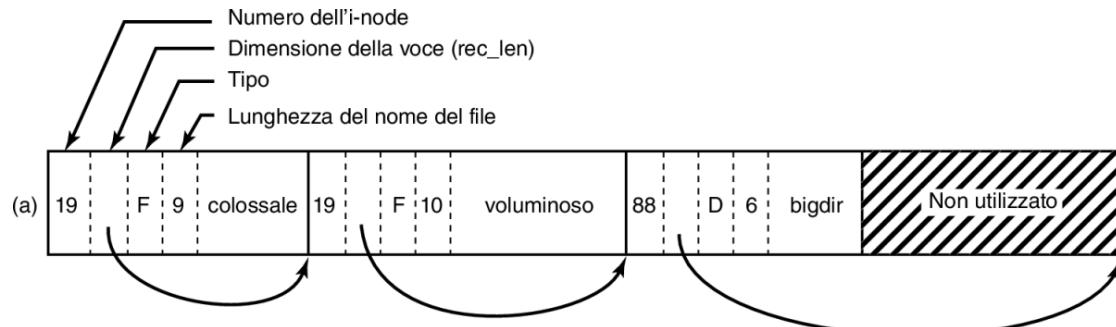
- **I-node**: Numerati, descrivono un solo file, contengono informazioni di contabilità e indirizzi dei blocchi di dati.
- **Blocchi di Dati**: Archiviano i file e le directory, non necessariamente contigui sul disco.



GESTIONE DEGLI I-NODE E DEI FILE NEL FILE SYSTEM EXT2 DI LINUX

- **Collocazione degli I-node e dei File**

- Gli **I-node delle directory sono distribuiti tra i gruppi di blocchi del disco.**
- Ext2 cerca di posizionare **i file nella stessa area di blocchi della directory genitore** per minimizzare la frammentazione.
- Utilizzo di bitmap per determinare rapidamente aree libere dove allocare nuovi dati del file system.



Una directory di Linux con tre file.



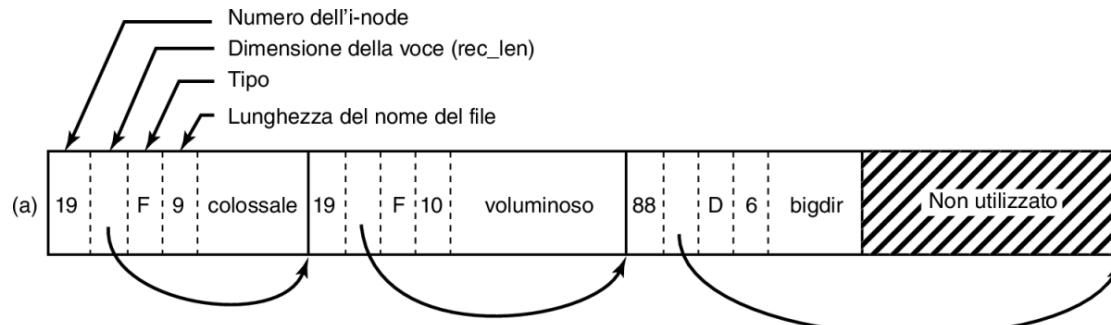
La stessa directory dopo l'eliminazione del file «Voluminoso».



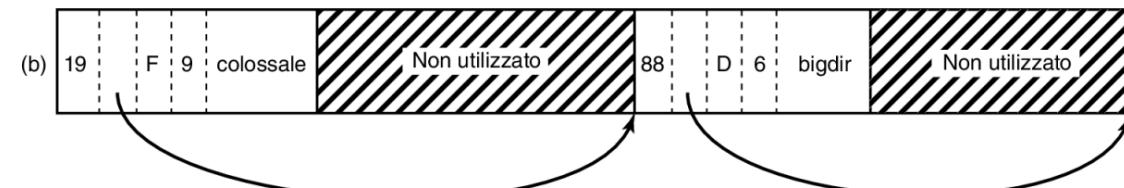
GESTIONE DEGLI I-NODE E DEI FILE NEL FILE SYSTEM EXT2 DI LINUX

- **Preallocazione di Blocchi**

- Ext2 preallocata otto blocchi aggiuntivi per un nuovo file per ridurre la frammentazione dovuta a scritture successive.
- Questa strategia bilancia il carico del file system e migliora le prestazioni riducendo la frammentazione.



Una directory di Linux con tre file.



La stessa directory dopo l'eliminazione del file «Voluminoso».



STRUTTURA E GESTIONE DELLE DIRECTORY IN EXT2

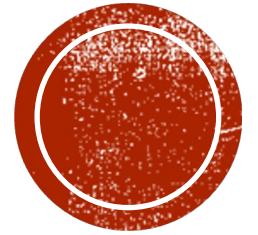
▪ Accesso ai File

- Utilizzo di **chiamate di sistema**, come `open`, per accedere ai file.
- L'analisi del percorso del file inizia dalla directory corrente del processo o dalla directory radice.

▪ Ricerca e Accesso ai File

- Le **ricerche nelle directory sono lineari** ma ottimizzate tramite una cache delle directory recentemente accedute.
- Per aprire un file, il percorso viene analizzato per localizzare l'I-node della directory e, infine, l'I-node del file stesso.
- Uso di **Soft link** e **Hard link** per fare riferimento ai file





PERFORMANCE

IMPLEMENTAZIONE DEL FILE SYSTEM

- Come memorizzare i file?
- Come implementare le directory?
- Come gestire lo spazio su disco?
- **Come garantire le prestazioni del file system?**
- Come garantire l'affidabilità del file system?



COMPARAZIONE DELLE PRESTAZIONI DEL FILE SYSTEM: MEMORIA VS. DISCO MAGNETICO

- **Velocità di Accesso**
 - **Memoria:** Accesso ultraveloce (es. 10 ns per leggere una parola a 32 bit).
 - **Disco Magnetico:** Accesso più lento a causa del tempo di ricerca della traccia (5-10 ms) e dell'attesa del posizionamento del settore sotto la testina di lettura.
- **Differenza nelle Prestazioni**
 - L'accesso a un disco magnetico può essere milioni di volte più lento dell'accesso alla memoria.
- **Ottimizzazioni nei File System**
 - Progettazione di file system con diverse ottimizzazioni per migliorare le prestazioni, considerando le significative differenze nel tempo di accesso e riducendo al minimo
 - i numeri di accesso al disco/SSD
 - il tempo di ricerca (*seek*) del dato sul disco/SSD
 - l'utilizzo dello spazio



OTTIMIZZAZIONE DELLE PRESTAZIONI DEL FILE SYSTEM

▪ **Block Cache / Buffer Cache**

- Utilizzata per ridurre i tempi di accesso al disco, mantenendo i blocchi più usati in memoria.
- Migliora le prestazioni conservando in memoria i blocchi logici del disco.

▪ **Allocazione dei Blocchi e Read Ahead**

- **Tecniche di allocazione intelligente**: blocchi vicini allocati nello stesso cilindro per minimizzare il movimento del braccio del disco.
- **Bitmap in memoria per allocare blocchi adiacenti** e migliorare l'efficienza di scrittura sequenziale.

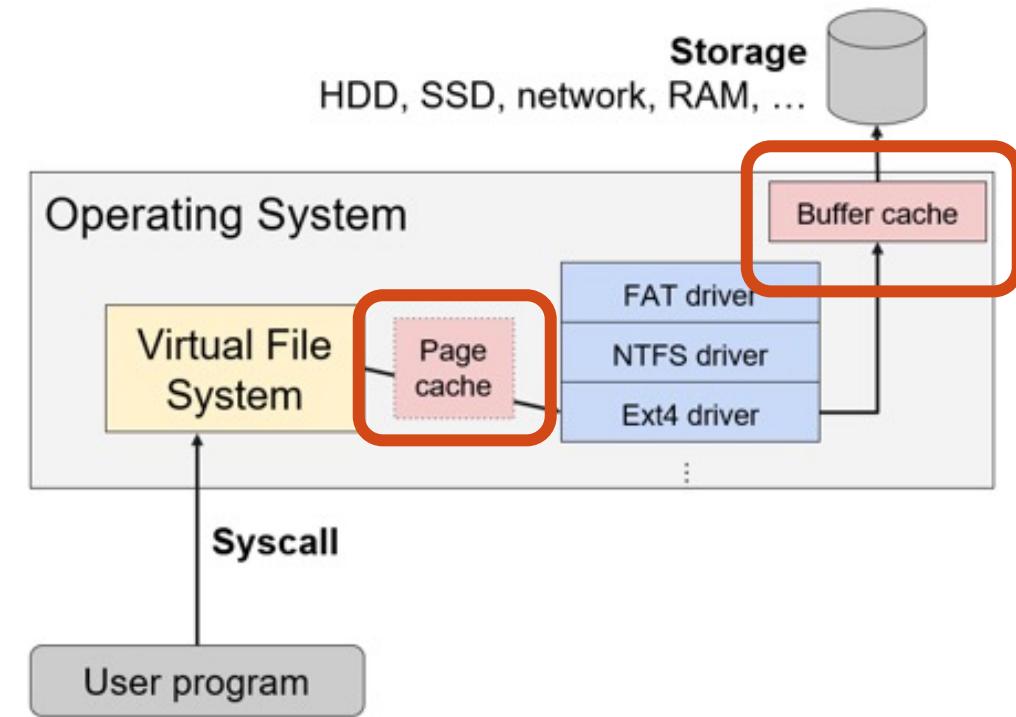
▪ **Deframmentazione**

- Con il tempo, i **dischi si frammentano**; i file sparsi portano a prestazioni inferiori.
- **Deframmentazione**: Riorganizza i file per essere contigui e raggruppa lo spazio libero.
- Windows fornisce lo strumento defrag per questa operazione;
 - consigliato regolarmente per HDD, ma non per SSD.



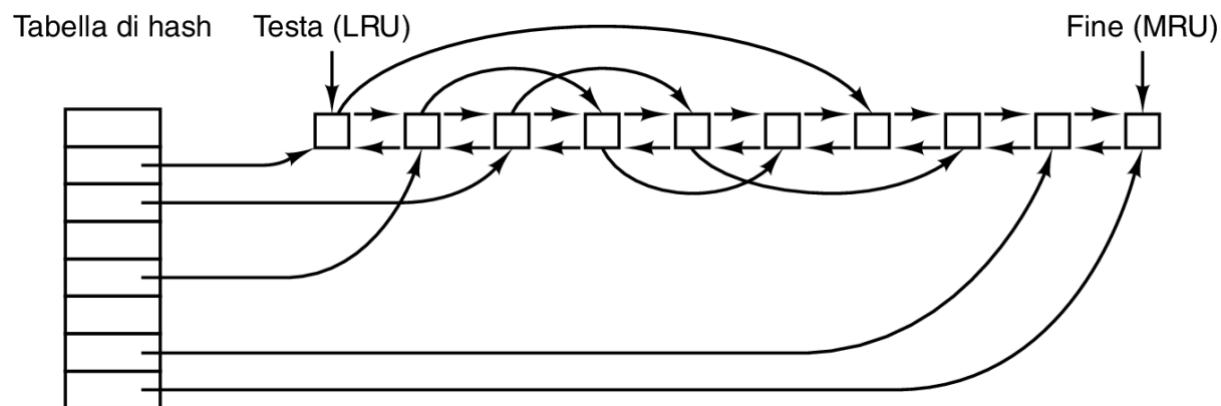
MINIMIZZAZIONE DELL'ACCESSO AL DISCO TRAMITE CACHING

- **Concetti di Caching**
 - **Buffer Cache**: Memorizza i blocchi del disco in RAM per ridurre gli accessi al disco.
 - **Page Cache**: Memorizza le pagine del filesystem virtuale (VFS, più avanti) in RAM prima di passare al driver del dispositivo.
- **Ottimizzazione del Caching**
 - **Dati duplicati**: Buffer cache e page cache spesso contengono gli stessi dati.
 - **Esempio**: file «mappati» in memoria
 - **Fusione di cache**: I sistemi operativi possono combinare buffer cache e page cache per un uso più efficiente della memoria e una riduzione degli accessi al disco.



IMPLEMENTAZIONE E FUNZIONAMENTO DELLA CACHE NEI FILE SYSTEM

- **Definizione e Scopo della Cache**
 - **Cache (block cache o buffer cache)**: Raccolta di blocchi del disco tenuti in memoria per migliorare le prestazioni.
 - **Scopo**: Ridurre i tempi di accesso al disco.
- **Gestione della Cache**
 - **Verifica** delle richieste di lettura per determinare **se il blocco richiesto è già in cache**.
 - Se **il blocco è in cache**, viene soddisfatta la richiesta senza accedere al disco.
 - Se **il blocco non è in cache**, viene prima letto da disco, portato in cache e poi utilizzato.
- **Ottimizzazione della Ricerca nella Cache**
 - **Uso di hash** per identificare rapidamente la presenza di un blocco in cache.
 - Lista concatenata per gestire i blocchi con lo stesso valore hash.



GESTIONE DELLA CACHE E ALGORITMI DI SOSTITUZIONE

• Sostituzione dei Blocchi nella Cache

- **Processo:** Quando la cache è piena, i nuovi blocchi sostituiscono quelli esistenti, che vengono riscritti su disco se modificati.
- **Algoritmi di Sostituzione:** Uso di algoritmi come FIFO, seconda chance, e LRU (Least Recently Used), simili alla paginazione.
- **Lista Bidirezionale LRU:** Ordine d'uso dei blocchi (meno recenti in testa, più recenti in fondo) per mantenere l'esatto ordine LRU.

• Limitazioni dell'Algoritmo LRU

- **Problema:** LRU può portare a inconsistenze in caso di crash, specialmente per blocchi critici come i blocchi di I-node.
- **Soluzione:** Schema di LRU modificato basato sull'importanza e la necessità immediata dei blocchi.

• Scrittura e Coerenza del File System

- **Blocchi critici:** Scrittura immediata su disco se modificati per mantenere la coerenza del file system.
- **Prevenzione della Perdita di Dati:** Utilizzo di chiamate di sistema come `sync` in UNIX per scrivere periodicamente i blocchi modificati su disco e ridurre la perdita di dati in caso di crash.



STRATEGIE DI SCRITTURA IN UNIX E WINDOWS E INTEGRAZIONE DELLA CACHE

- **Strategie di Scrittura nei Sistemi Operativi**

- **UNIX:** Uso della chiamata di sistema `sync` per scrivere periodicamente i blocchi modificati su disco.
- **Windows:** Strategia **write-through** (scrittura immediata) per i blocchi modificati, ora integrata anche con la chiamata `FlushFileBuffers`.

- **Integrazione tra Cache Buffer e Cache delle Pagine**

- **Obiettivo:** Ottimizzare l'I/O e semplificare la gestione della memoria.
- **Integrazione:** Alcuni sistemi operativi **combinano cache buffer e cache delle pagine** per una gestione efficiente dei dati.
 - **Supporto per File Mappati in Memoria:** Trattamento unificato di blocchi e pagine dei file in una singola cache.
 - In questo modo un file interamente mappato in memoria non occupa un'area di memoria utile per altre pagine, ma sfrutta il Cache Buffer del disco



OTTIMIZZAZIONE DELLA DISPOSIZIONE DEI DATI PER MINIMIZZARE I MOVIMENTI DEL BRACCIO DEL DISCO

- **Riduzione della Movimentazione del Braccio**

- **Obiettivo:** Minimizzare i movimenti del braccio del disco raggruppando i blocchi di dati utilizzati in sequenza.
- **Allocazione dei Blocchi:** Uso di bitmap in memoria per allocare blocchi vicini, riducendo i tempi di ricerca.

- **Strategie di Allocazione**

- **Gruppi di Blocchi:** Tracciamento dello spazio non per blocchi singoli, ma per gruppi consecutivi per migliorare la lettura sequenziale.
- **Posizionamento Rotazionale:** Allocazione di blocchi consecutivi di un file nello stesso cilindro per ridurre i tempi di ricerca.



DISPOSIZIONE DEGLI I-NODE E ADATTAMENTI PER SSD

- **Posizionamento degli I-node**

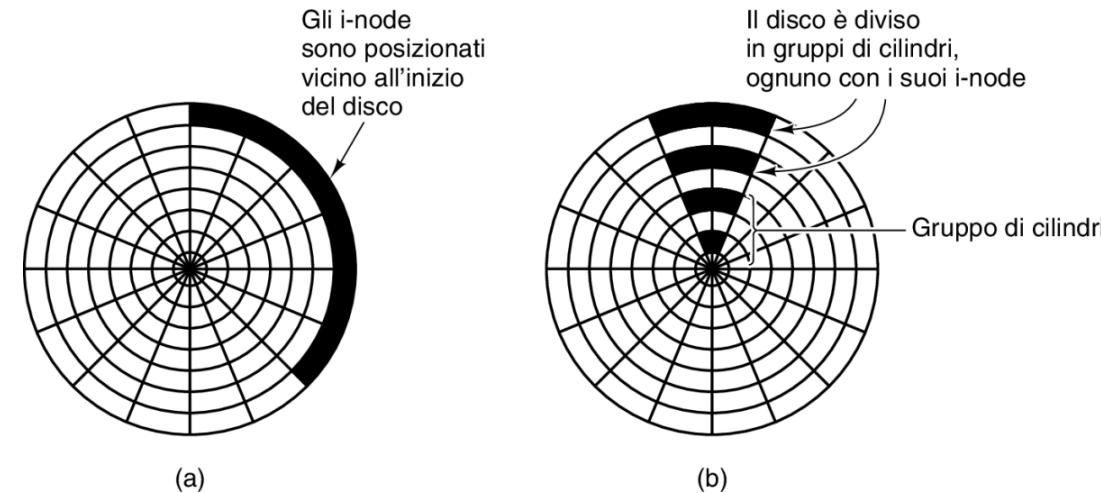
- a) Posizionamento Tradizionale: I-node vicino all'inizio del disco, risultando in ricerche più lunghe.

- **Miglioramenti nel Posizionamento**

- b) Centrare gli I-node: Posizionare gli I-node nel mezzo del disco per dimezzare il tempo di ricerca.
- Gruppi di Cilindri: Divisione del disco in gruppi di cilindri con I-node, blocchi e lista dei blocchi liberi per ciascun gruppo.

- **Considerazioni per SSD**

- **Differenze con Dischi Magnetici:** Gli SSD non hanno parti mobili, quindi i tempi di accesso casuale sono simili a quelli sequenziali (“Seek time” costante).



COMPRENDERE L'USO DELLA MEMORIA IN LINUX CON IL COMANDO `free`

- **Introduzione al Comando `free`**

- Uno strumento essenziale in Linux per monitorare l'utilizzo della memoria.
- Fornisce una panoramica dettagliata della memoria RAM, inclusa la cache e lo spazio libero.

- **Visualizzazione della Cache di Memoria**

- Usa `free -h` per un output leggibile che include la dimensione e l'utilizzo della cache.
- La colonna "buff/cache" rivela quanto spazio è utilizzato per buffer e cache, inclusi i dati letti dal disco.

- **Memoria Disponibile vs. Memoria Libera**

- Colonna "Free": mostra la memoria fisica non utilizzata attualmente.
- Colonna "Available": stima la memoria disponibile per nuovi processi, considerando anche la memoria facilmente liberabile come cache.
 - "Available" offre una visione realistica della memoria effettivamente disponibile per applicazioni senza influire sulle prestazioni.

- **Importanza per le Prestazioni del Sistema**

- La cache aiuta a velocizzare l'accesso ai file frequentemente usati, riducendo la necessità di leggere ripetutamente dal disco più lento.



DEFRAMMENTAZIONE DEL DISCO E LA SUA IMPORTANZA NEI DIVERSI SISTEMI

- **Impatto della Frammentazione**

- Con il tempo, i file e lo spazio libero si disperdono sul disco, causando una diminuzione delle prestazioni.

- **Riorganizzazione dei Dati**

- La deframmentazione consolida i file e lo spazio libero, migliorando l'efficienza di lettura/scrittura per HDD.

- **Strumenti e Pratiche**

- `defrag` in Windows riorganizza i file; raccomandato per HDD, sconsigliato per SSD per evitare usura inutile.

- **Considerazioni sui File System**

- Linux con ext3/ext4 gestisce meglio la frammentazione, riducendo la necessità di deframmentazione manuale.
 - **Preallocazione di Blocchi:** Ext4 introduce la preallocazione di blocchi. Quando si scrive un file, Ext4 preallocata un gruppo di blocchi contigui, piuttosto che uno alla volta, riducendo la frammentazione per i file in espansione.



OTTIMIZZAZIONE DELLO SPAZIO SU DISCO: COMPRESSESIONE E DEDUPLICAZIONE

- **Compressione dei Dati**

- Riduce la dimensione dei file tramite algoritmi che identificano e sostituiscono sequenze di dati ripetuti.
- File system come NTFS (Windows), Btrfs (Linux) e ZFS (vari sistemi operativi) possono comprimere dati automaticamente.

- **Deduplicazione dei Dati**

- Rileva e rimuove i dati duplicati all'interno di un intero file system, conservando una sola copia di ciascun dato unico.
- Applicata sia a livello di blocchi di disco che di porzioni di file, prevalentemente in ambienti con dati condivisi.

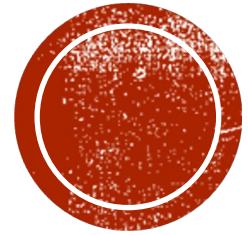
- **Metodi di Deduplicazione**

- **Inline:** Controlla i dati per duplicati durante la scrittura, potenzialmente rallentando il processo.
- **Post-Process:** Scrittura immediata dei dati, con controllo di duplicati eseguito successivamente in background.

- **Sicurezza e Affidabilità**

- **Controllo degli Hash:** Necessario per assicurare che i dati non siano falsamente identificati come duplicati a causa di collisioni hash.





AFFIDABILITÀ DEL FILE SYSTEM

IMPLEMENTAZIONE DEL FILE SYSTEM

- Come memorizzare i file?
- Come implementare le directory?
- Come gestire lo spazio su disco?
- Come garantire le prestazioni del file system?
- **Come garantire l'affidabilità del file system?**



MINACCIE ALLA AFFIDABILITÀ DEI FILE SYSTEM

- **Guasti del Disco**
 - **Blocchi Danneggiati:** Settori illeggibili che possono corrompere i dati.
 - **Errori su Intero Disco:** Fallimenti hardware che rendono il disco completamente inutilizzabile.
- **Interruzioni di Energia**
 - **Scritture Inconsistenti:** possono causare incongruenze nei dati o nei metadati sul disco.
- **Bug del Software / Corruzione dei (Meta)dadi:** Errori di programmazione portano alla scrittura di dati errati/corrotti
- **Errori Umani/Comandi Errati:** `rm * .o VS rm * .o`
- **Perdita o Furto del Computer/Accesso Non Autorizzato:** Rischi di accesso ai dati da parte di persone non autorizzate in caso di furto o smarrimento del dispositivo.
- **Malware/Ransomware:** Virus o altri software dannosi che possono infettare, criptare o distruggere i dati.



BACKUP DEL FILE SYSTEM E LA SUA IMPORTANZA

- «*I backup su disco sono generalmente effettuati per affrontare uno dei due potenziali problemi:*
 - *Recupero da un disastro.*
 - *Recupero dalla stupidità.»*
- **Necessità di Backup**
 - Il backup è cruciale per salvaguardare informazioni importanti come documenti, database, piani aziendali, ecc.
- **Modalità di Backup**
 - **Backup Completo:** Esegue una copia totale dei dati, solitamente su base settimanale o mensile.
 - **Backup Incrementale:** Copia solo i file modificati dall'ultimo backup completo, riducendo il tempo e lo spazio richiesti.



BACKUP DEL FILE SYSTEM E LA SUA IMPORTANZA

▪ Tipologie di Backup

- **Backup Fisico:** Copia sequenziale di tutti i blocchi del disco, a partire dal blocco 0 fino all'ultimo.
- **Backup Logico:** Seleziona e copia solo i file e le directory specifici, ignorando i file di sistema e i blocchi danneggiati.

• Considerazioni Tecniche

- **Comprimere i Dati:** Riduce lo spazio necessario, ma aumenta il rischio di perdita di dati a causa di errori di compressione.
- **Backup di File System Attivi:** Richiede l'uso di snapshot per garantire coerenza durante il backup di un sistema in uso.
- **Sicurezza dei Backup:** Importanza di tenere i backup in luoghi sicuri e separati per prevenire perdite o danni.



BACKUP FISICO

- **Considerazioni sul Backup Fisico**

- **Efficienza:** Semplice e veloce, può essere eseguito alla velocità del disco.
- **Gestione dei Blocchi Danneggiati:** Necessità di evitare la copia di blocchi danneggiati per prevenire errori di lettura (blocchi danneggiati ci sono sempre).
- **File Non Necessari:** Necessità di evitare la copia di file di sistema come i file di paginazione e ibernazione

- **Vantaggi**

- **Semplicità:** Facile da implementare e utilizzare.
- **Affidabilità:** Minore probabilità di errori nel processo di backup.

- **Sfide e Limitazioni**

- **Mancanza di Flessibilità:** Difficoltà nel saltare directory specifiche o nel fare backup incrementali.
- **Ripristino di Singoli File:** Non è possibile ripristinare file individuali senza un intero ripristino del sistema.



BACKUP LOGICO E IL SUO ALGORITMO

- **Funzionamento del Backup Logico**

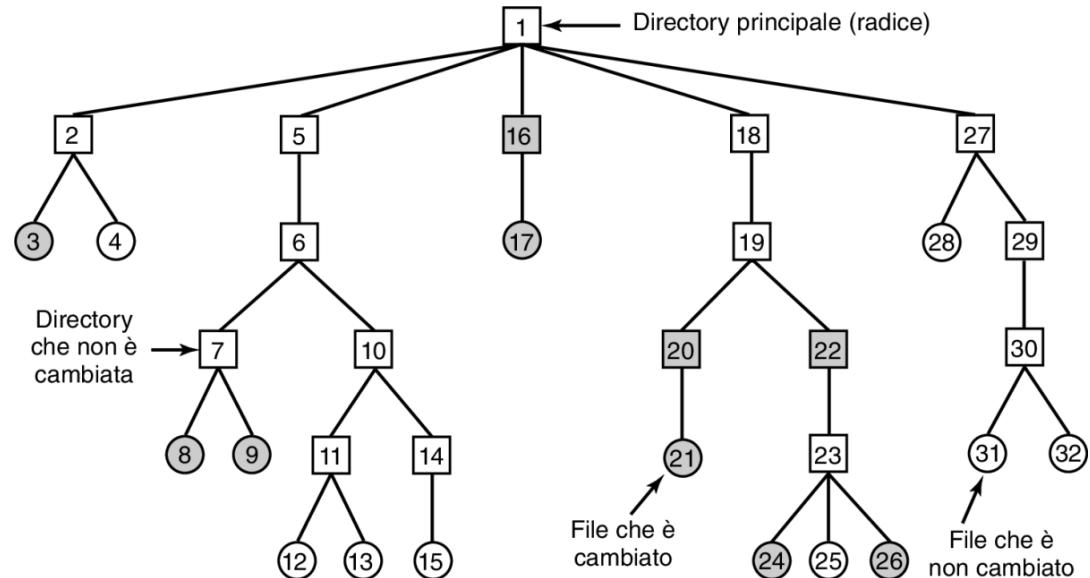
- Parte da specifiche directory e effettua il backup di tutti i file e directory modificati a partire da una data specifica.
- Permette il ripristino o il trasferimento dell'intero file system su un nuovo computer.
- Ideale per backup incrementali o completi.

- **Recupero Facilitato**

- Consente il ripristino semplice di file o directory specifici grazie alla precisa identificazione dei dati salvati.

- **Algoritmo di Backup in UNIX**

- Include file e directory modificati (in grigio nella figura), e tutte le directory lungo il percorso verso i file modificati.



Un file system di cui eseguire il backup. I quadrati sono directory, i cerchi sono i file. Gli oggetti in grigio hanno subito modifiche dopo il backup precedente. Directory e file sono contrassegnati dal numero del loro I-node.



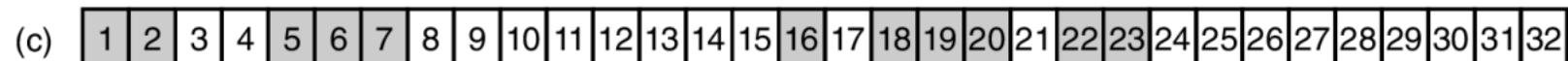
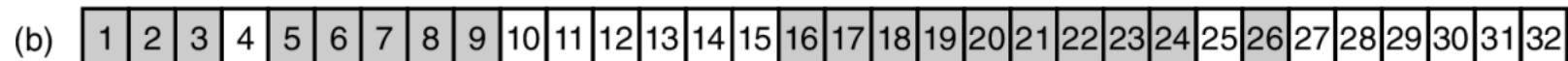
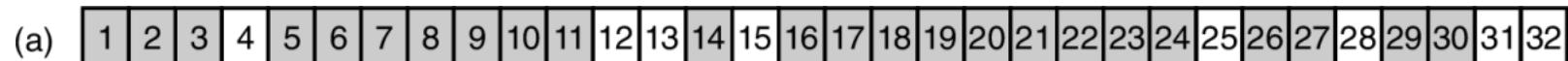
ALGORITMO DI BACKUP LOGICO

- **Utilizzo di una Bitmap per il Backup**

- L'algoritmo utilizza una bitmap indicizzata per numero di I-node, con bit impostati per I-node di file e directory modificati.

- **Fase 1: Rilevamento delle Modifiche**

- Inizia dalla directory radice, esamina tutte le voci e contrassegna nella bitmap gli I-node di file e directory modificati.
 - Include tutte le directory nel percorso verso file modificati, indipendentemente dal loro stato di modifica.
- a) mostra gli oggetti selezionati nella bitmap.



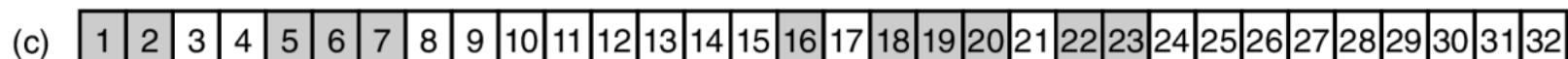
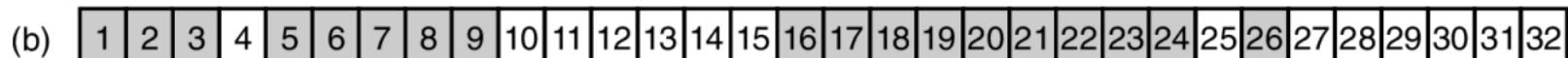
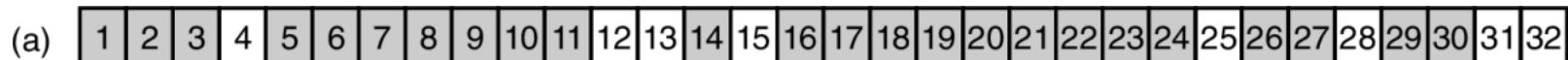
ALGORITMO DI BACKUP LOGICO (2)

- **Fase 2: Pulizia della Bitmap**

- Deseleziona le directory che non contengono né file né sottodirectory modificate.
- Lascia nella bitmap solo gli elementi che richiedono backup.
 - b) mostra la bitmap dopo la pulizia.

- **Fase 3 e 4: Backup Effettivo**

- Fase 3: Backup delle directory contrassegnate, con i loro attributi.
- Fase 4: Backup dei file contrassegnati, sempre con i relativi attributi.
 - c) indica le directory e i file da salvare.



ALGORITMO DI BACKUP LOGICO (3)

- **Ripristino dal Backup**

- Creazione di un nuovo file system vuoto e ripristino del backup completo più recente, seguito dai backup incrementali.

- **Considerazioni Aggiuntive**

- Ricostruzione della lista dei blocchi liberi.
- Gestione dei link multipli ai file.
- Trattamento dei file con buchi (spazi non scritti tra i dati).
- Esclusione di file speciali non necessari nel backup.



BONUS: rsync E ESEMPIO D'USO

▪ Definizione di rsync:

- rsync è un comando utilizzato nei sistemi basati su UNIX per la **sincronizzazione di file e cartelle tra due location diverse**, sia su una stessa macchina sia tra macchine diverse.
- Ottimizza il trasferimento dei dati **trasmettendo solo le parti di file che sono state modificate**.
- **Ideale per backup, ripristino e sincronizzazione** di dati in ambienti di rete.

▪ Funzionalità Principali:

- **Efficienza:** Trasferisce solo le differenze tra le sorgenti e le destinazioni.
- **Versatilità:** Supporta la copia di link, dispositivi, attributi, permessi, dati utente e gruppo.
- **Sicurezza:** Può utilizzare SSH per trasferimenti criptati.



BONUS: rsync E ESEMPIO D'USO (2)

■ Esempio Pratico:

- **Sincronizzazione Locale:** rsync -av /sorgente/cartella /destinazione/cartella
 - -a: modalità archivio, mantiene i permessi e la struttura dei file.
 - -v: modalità verbosa, mostra i dettagli del trasferimento.
- **Sincronizzazione Remota:** rsync -av /sorgente/cartella utente@remoto:/destinazione/cartella
 - Sincronizza la cartella dalla macchina locale a quella remota utilizzando l'account utente.



INTRODUZIONE ALLA COERENZA DEL FILE SYSTEM

- **Importanza della Coerenza:**
 - Cruciale per mantenere l'integrità dei dati.
 - Problemi di incoerenza possono sorgere a seguito di crash durante la scrittura dei blocchi.
- **Utilità per la Verifica della Coerenza:**
 - Sistemi come UNIX (fsck) e Windows (sfc) hanno utility per verificare la coerenza.
 - Eseguite all'avvio, specialmente dopo un crash.
- **File System con Journaling (vedi dopo):**
 - Progettati per gestire autonomamente la maggior parte delle incoerenze.
 - Non necessitano di controlli esterni dopo un crash.



TIPI DI CONTROLLO DI COERENZA

- **Controllo dei Blocchi:**
 - Costruzione di due tabelle con contatori per ogni blocco (per file e blocchi liberi).
 - Analisi di tutti gli I-node e **verifica della presenza dei blocchi in file e nella lista dei blocchi liberi.**
- **Situazioni Riscontrabili:**
 - a) Blocchi mancanti: non presenti in nessuna delle due tabelle
 - b) Blocchi duplicati nella lista dei blocchi liberi
 - c) Blocchi di dati presenti in più file
- **Azione Correttiva:**
 - Aggiunta dei blocchi mancanti ai blocchi liberi.
 - Assegnazione di blocchi duplicati a file diversi.

Numero del blocco

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0

Blocchi in uso

0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Blocchi liberi

(a)

Numero del blocco

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0

Blocchi in uso

0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Blocchi liberi

(b)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0

Blocchi in uso

0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Blocchi liberi

(c)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	2	1	1	1	0	0	1	1	1	0	0

Blocchi in uso

0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Blocchi liberi

(d)



VERIFICA DEL SISTEMA DELLE DIRECTORY E PROTEZIONE DELL'UTENTE

- **Controllo del Sistema delle Directory:**
 - Utilizzo di una tabella di contatori per file.
 - Confronto dei contatori dei link negli I-node con le voci di directory.
- **Tipi di Errori:**
 - Contatori dei link troppo alti o bassi.
 - Errori nel formato delle directory o I-node con modalità insolite.
- **Protezione dell'Utente contro Errori Umani:**
 - Esempio: distinzione tra 'rm *.o' e 'rm * .o' in sistemi UNIX.
 - In Windows, file cancellati vengono spostati nel cestino per un possibile recupero.



PRINCIPI E FUNZIONAMENTO DEI FILE SYSTEM CON JOURNALING

- **Definizione e Scopo**

- **File system con journaling:** Registra anticipatamente le operazioni da eseguire in un log, per garantire la coerenza in caso di crash.
- **Utilizzo:** Ampio uso in file system come NTFS (Microsoft), ext4 e ReiserFS (Linux), e opzione predefinita in macOS.

- **Esempio di Operazione: Eliminazione di un File**

- **Processo in UNIX:** Rimozione dal file dalla directory, rilascio dell'I-node, restituzione dei blocchi al pool dei blocchi liberi.
- **Problemi in caso di crash:** Perdita di accesso agli I-node e ai blocchi, o assegnazione errata di I-node e blocchi.

- **Cos'è il Journal**

- Un journal in un file system è come un **registro che tiene traccia delle modifiche che verranno apportate** al file system prima che esse avvengano effettivamente.



STRUTTURA E FUNZIONAMENTO DEL JOURNAL NEI FILE SYSTEM

▪ Come Funziona

1. **Fase di Registrazione:** Prima di eseguire qualsiasi modifica (come la creazione o la cancellazione di un file), il file system scrive un record nel journal.
 - Questo record **descrive l'operazione che verrà eseguita.**
 2. **Fase di Esecuzione:** Dopo aver registrato l'operazione, il file system procede con la modifica effettiva dei dati sul disco.
 3. **Fase di Conferma:** Una volta completata l'operazione, il file system aggiorna il journal per indicare che l'azione è stata completata con successo.
- Se si verifica un crash del sistema prima che una modifica sia completata, **al riavvio successivo il file system consulta il journal.**
- Se trova **operazioni registrate ma non confermate, procede a completarle.** Questo assicura che le modifiche parziali non lascino il file system in uno stato incoerente.

▪ Vantaggi del Journaling

- **Integrità dei Dati:** Il journaling **riduce la possibilità di corruzione del file system in caso di crash inaspettato**, assicurando che tutte le operazioni siano completate o nessuna.
- **Recupero Rapido:** Riduce significativamente il tempo di recupero dopo un crash, poiché il file system sa esattamente quali operazioni completare o annullare.



SICUREZZA DEI DATI: ELIMINAZIONE SICURA E CIFRATURA DEL DISCO

- **Cancellazione Convenzionale vs Eliminazione Sicura**
 - La cancellazione standard non rimuove fisicamente i dati dal disco, lasciandoli vulnerabili agli attacchi.
 - L'eliminazione sicura richiede la distruzione fisica o la sovrascrittura approfondita dei dati.
- **Dati Residui su Dischi Magnetici**
 - Sovrascrivere con zero non è sempre sufficiente a causa dei residui magnetici che possono essere recuperati con tecniche avanzate.
 - Gli SSD presentano sfide aggiuntive: la mappatura dei blocchi flash è gestita dalla FTL e non dal file system, rendendo la sovrascrittura meno prevedibile.
- **E' consigliato inserire sequenze di 0 e numeri casuali, ripetendo l'operazione almeno 3-7 volte**
 - **Attenzione:** molte scritture possono mettere a dura prova gli SSD



SICUREZZA DEI DATI: ELIMINAZIONE SICURA E CIFRATURA DEL DISCO (2)

- **Cifratura del Disco**

- La soluzione più efficace per proteggere i dati è cifrare l'intero disco con algoritmi robusti come l'AES.
- Sistemi operativi moderni, come Windows, offrono la cifratura del disco che lavora in background, spesso sconosciuta agli utenti.

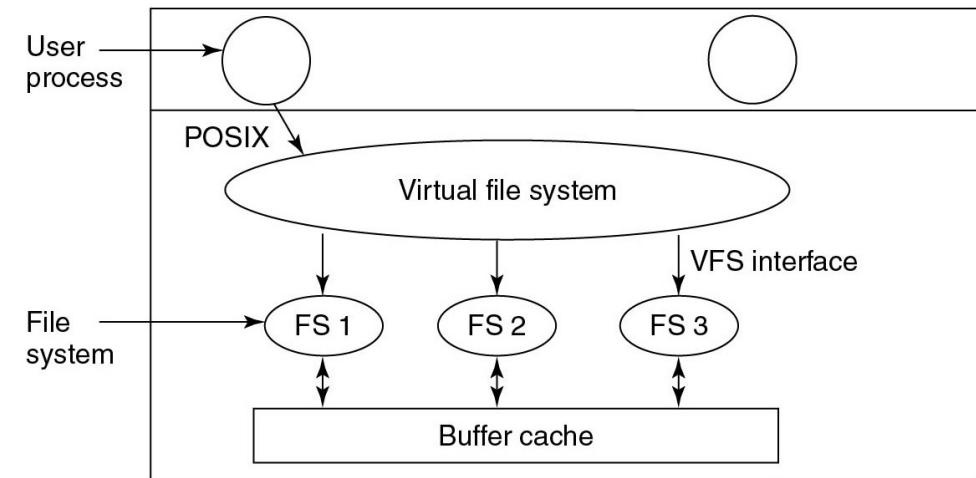
- **Implementazione della Cifratura**

- SED (Self-Encrypting Drives): Dispositivi con cifratura integrata, che tuttavia possono avere vulnerabilità di sicurezza.
- Windows utilizza AES (Advanced Encryption Standard) per cifrare i dischi, con la chiave master del volume decifrata tramite password utente, chiave di ripristino o TPM.



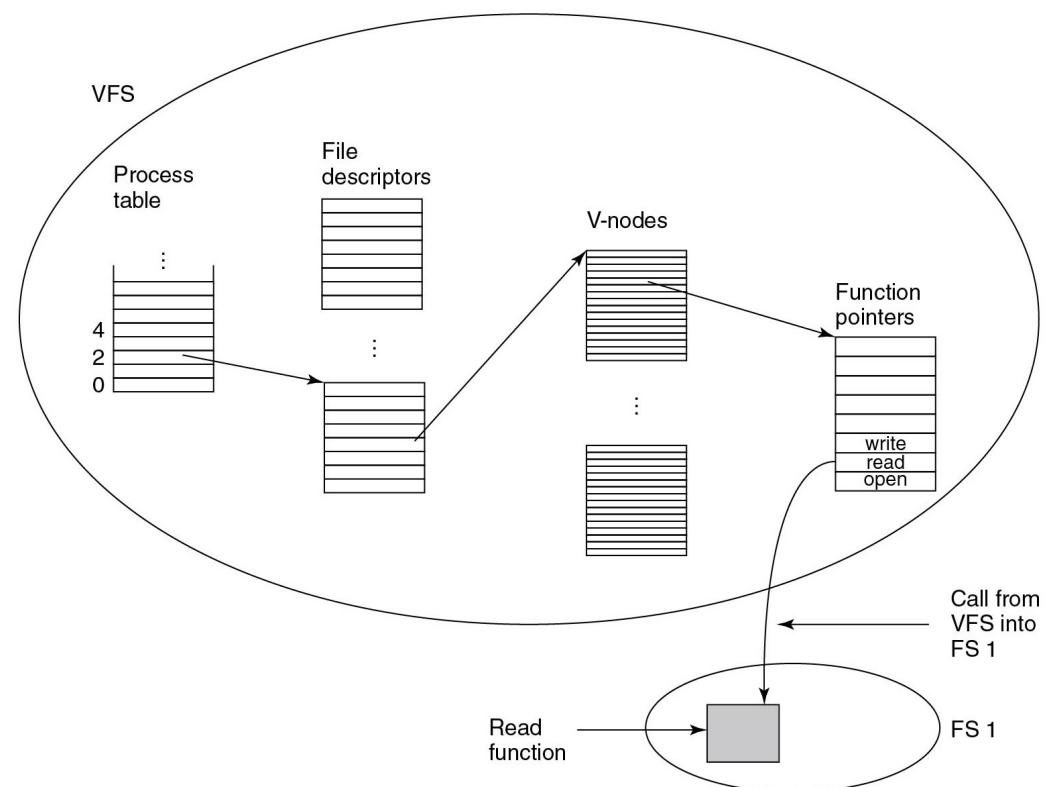
INTRODUZIONE AI FILE SYSTEM VIRTUALI

- **Diversità dei File System:**
 - Sistemi operativi moderni gestiscono diversi file system (NTFS, FAT-32, FAT-16, ecc.) simultaneamente.
 - Windows utilizza lettere di unità (C:, D:, ecc.) per gestire file system differenti.
 - Sistemi UNIX tentano di integrare più file system in una singola struttura gerarchica.
- **VFS (Virtual File System):**
 - Struttura che permette di integrare vari file system in una struttura unificata.
 - Si basa su un livello di codice comune che interagisce con i file system reali sottostanti.
 - Gestisce file system locali e remoti, come quelli in NFS (Network File System).
- **Interfacce del VFS:**
 - **Interfaccia superiore:** Interagisce con le chiamate di sistema POSIX dei processi utente (es. open, read, write).
 - **Interfaccia inferiore:** Composta da decine di funzioni che il VFS può inviare ai file system sottostanti.



FUNZIONAMENTO E STRUTTURA DEL VFS: CONCETTI CHIAVE

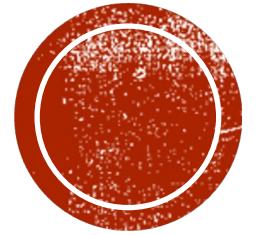
- **Superblock nel VFS:** Rappresenta il descrittore di alto livello di un file system specifico nel VFS.
 - informazioni cruciali sul file system, come il tipo, la dimensione
 - usato per identificare e interagire con il file system sottostante, facilitando l'accesso e la gestione delle sue risorse.
- **V-node nel VFS:** Astrazione di un file individuale all'interno del VFS, rappresentando un nodo nel file system virtuale.
 - **Contiene metadati** come permessi, proprietà, dimensione del file e riferimenti ai dati effettivi sul disco.
 - Il VFS sfrutta i v-node per fornire un accesso indipendente dal file system ai file, permettendo operazioni di lettura, scrittura e gestione dei file attraverso vari file system.
- **Directory nel VFS:** Struttura che gestisce l'organizzazione e il mapping dei file e delle sottodirectory all'interno del VFS.
 - **Permette al VFS di mappare i nomi dei file ai loro v-node** corrispondenti, indipendentemente dal file system in cui si trovano.
 - Facilita la navigazione e l'accesso ai file, consentendo agli utenti e ai processi di interagire con un'interfaccia unificata



AGGIUNTA DI NUOVI FILE SYSTEM E VANTAGGI DEL VFS

- **Registrazione dei File System con il VFS:** File system forniscono un vettore di funzioni richieste dal VFS al momento della registrazione.
 - Permette al VFS di sapere come eseguire specifiche operazioni su un file system registrato.
- **Montaggio e Uso del File System:** Al **montaggio**, file system fornisce informazioni al VFS (es. superblock).
 - **Esempio:** apertura di un file in `/usr` con un file system montato su `/usr`.
 - **Creazione di un v-node e mappatura di operazioni specifiche del file system reale.**
- **Gestione delle Richieste di I/O:** Tracciamento dei file aperti nei processi utente tramite v-node e tabelle dei descrittori dei file.
 - Chiamate come `read` seguono il puntatore dalla tabella dei descrittori ai v-node e alle funzioni del file system reale.
- **Aggiunta di Nuovi File System:** Relativamente semplice aggiungere nuovi file system.
 - Progettisti devono fornire funzioni che rispettino l'interfaccia VFS.
 - Il VFS rende possibile la gestione trasparente di file system eterogenei.





BONUS: RAID

INTRODUZIONE AL RAID

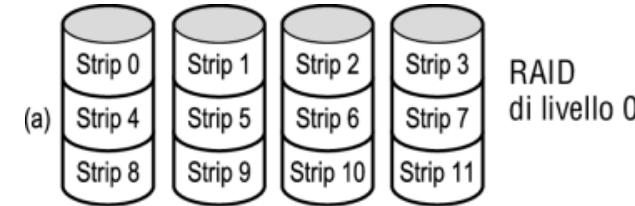
- Non solo il Sistema Operativo garantisce l'affidabilità dei dati, anche l'Hardware
- **Storia e Sviluppo:**
 - RAID nato per migliorare le prestazioni dei sistemi di memorizzazione su dischi magnetici.
 - Prima degli SSD, crescono esponenzialmente le prestazioni della CPU, ma non quelle dei dischi magnetici.
 - Patterson et al., nel 1988, proponevano l'uso di organizzazioni specifiche dei dischi per migliorarne prestazioni e affidabilità.
- **Definizione:**
 - **RAID**: Redundant Array of Inexpensive (poi Independent) Disks.
 - Inizialmente contrastato dal concetto di SLED (Single Large Expensive Disk).
- **Funzionamento Base:**
 - Installazione di un contenitore di dischi accanto al computer.
 - Utilizzo del controller RAID per migliorare prestazioni e affidabilità.
 - Supporto sia per unità SCSI, SATA che SSD.



LIVELLI RAID E LORO FUNZIONI

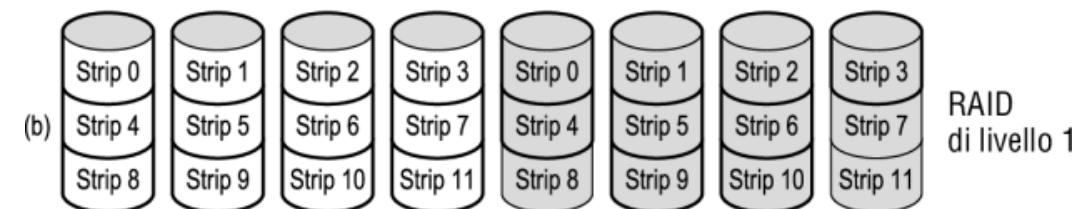
a) RAID Level 0: + storage

- Distribuzione dei dati in strip (strisce) su dischi multipli.
- Migliora le prestazioni con richieste grandi.
- Nessuna ridondanza, quindi potenzialmente meno affidabile.



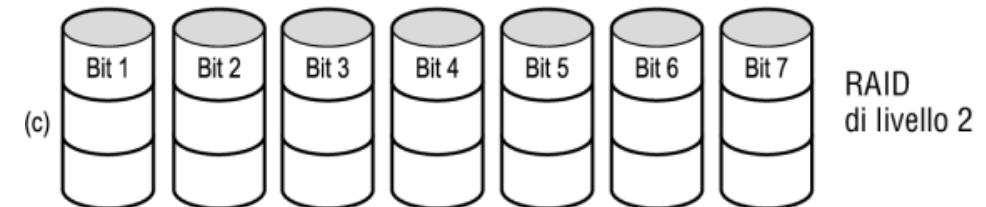
b) RAID Level 1: + redundancy

- Duplicazione dei dischi per tolleranza agli errori.
- Prestazioni di lettura migliorate, scrittura simile a un'unità singola.



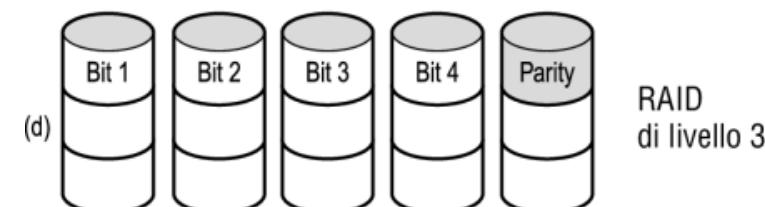
c) RAID Levels 2: + storage

- Level 2 basato su parole o byte con codice di Hamming.



d) RAID Levels 3: + redundancy, +storage

- Level 3 usa un singolo bit di parità per parola, richiede sincronizzazione delle unità.



LIVELLI RAID E LORO FUNZIONI

e) RAID Level 4: +storage, +redundancy

- Utilizzo di strip con un'unità extra per la parità.
- La parità è ottenuta

f) RAID Level 5: +storage, +redundancy

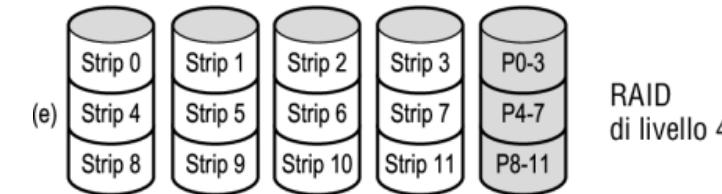
- distribuisce bit di parità in modo uniforme su tutte le unità.

g) RAID Level 6: +storage, +redundancy

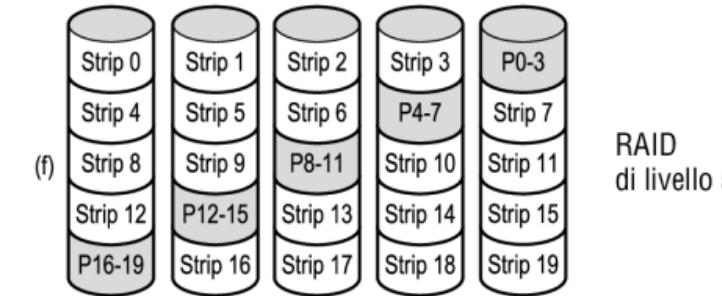
- Simile a Level 5 ma con un blocco di parità aggiuntivo.
- Maggiore affidabilità e tolleranza agli errori.

❖ possibile combinare più schemi

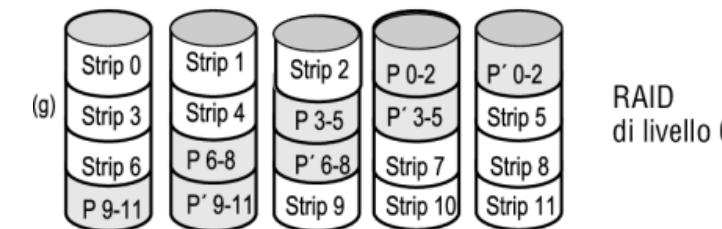
- **RAID 0 + 1:** duplicazione ridondante secondo lo schema di RAID 1 di dischi associati in schema RAID 0



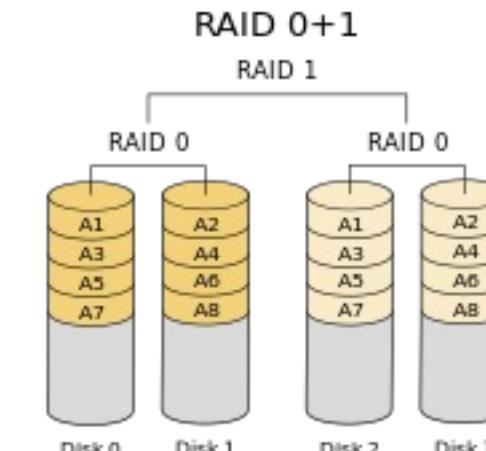
RAID
di livello 4



RAID
di livello 5



RAID
di livello 6



ESEMPIO: RAID 5 IN AZIONE

- **Scenario:**

- Immaginiamo un RAID 5 con tre dischi: Disco A, Disco B e Disco C.
- Ogni disco contiene una strip di dati e una strip di parità viene calcolata usando l'operazione XOR sui bit dei dati da Disco A e Disco B, e poi memorizzata sul Disco C.

- **Esempio di Dati:**

- **Disco A:** contiene i dati 1011.
- **Disco B:** contiene i dati 1100.

- **Calcolo della Parità (XOR):**

- Eseguiamo l'operazione XOR tra i dati di Disco A e Disco B, bit per bit.
- XOR di 1011 (Disco A) e 1100 (Disco B) = 0111.



ESEMPIO: RAID 5 IN AZIONE (2)

- **Risultato:**

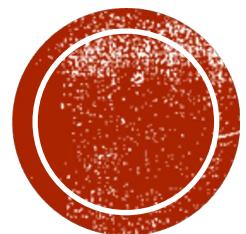
- **Disco A:** 1011 (dati originali).
- **Disco B:** 1100 (dati originali).
- **Disco C:** 0111 (risultato dell'XOR, quindi dati di parità).

- **Funzionamento in Caso di Guasto di un Disco:**

- Se, per esempio, il Disco B si guasta, **possiamo ricostruire i suoi dati.**
- XOR di 1011 (Disco A) e 0111 (Disco C, parità) = 1100
- **che sono i dati originali del Disco B.**



MA ALLA FINE...

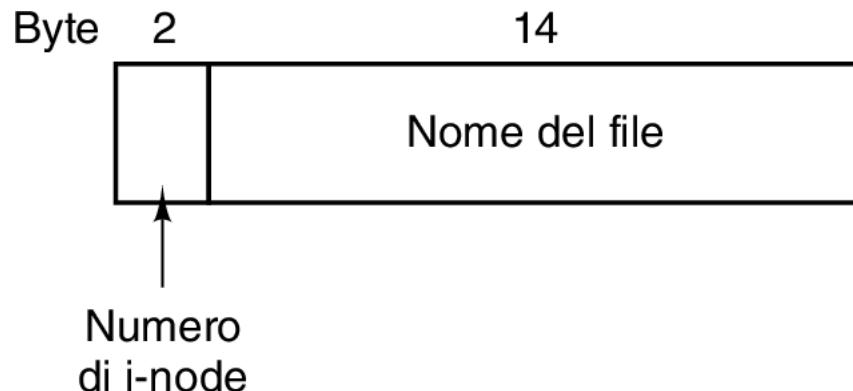


QUALE FILE SYSTEM E' USATO?

**E COME FACCIO A «MONTARE» UNA
PARTIZIONE?**

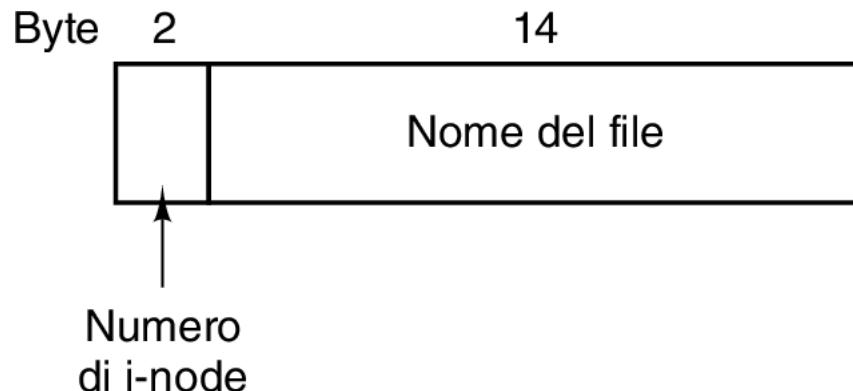
INTRODUZIONE AL FILE SYSTEM V7 DI UNIX (1979!!!)

- **Origine e Influenza:** Derivato dal MULTICS, il file system V7 è stato implementato nel PDP-11 e ha contribuito significativamente alla fama di UNIX.
- **Struttura:** Formato da un albero gerarchico con la possibilità di formare un grafo aciclico orientato tramite link.
- **Nomi dei File:** Lunghezza massima di 14 caratteri, escludendo i caratteri '/' e NUL.
- **Limitazione Numerica:** Massimo di 64K file per file system, a causa del formato delle voci di directory.



VOCI DI DIRECTORY E I-NODE IN UNIX V7

- **Voce di Directory:** Composta da un nome di file (14 byte) e un numero di i-node (2 byte).
- **i-node:** Contiene attributi del file, inclusi dimensioni, timestamp, proprietario, gruppo, e informazioni di protezione.
- **Gestione dei Link:** Contatore di link nell'i-node, che viene incrementato o decrementato con la creazione o rimozione di link.
- **Recupero di Blocchi:** Utilizzo di indirizzi diretti e indiretti (singoli, doppi e tripli) per gestire file di varie dimensioni.



ALGORITMO DI RICERCA NEL FILE SYSTEM UNIX V7

- **Ricerca di File:** Processo per localizzare file tramite un percorso, partendo dalla directory radice o dalla directory corrente.
- **Esempio di Ricerca:** Ricerca di `/usr/ast/mbox` attraverso sequenze di letture di directory e individuazione di i-node.
- **Percorsi Relativi:** Gestiti allo stesso modo dei percorsi assoluti, iniziando dalla directory di lavoro corrente.
- **Gestione delle Directory Speciali:** Uso di `.` e `..` per indicare rispettivamente la directory corrente e la genitore.



PECULIARITÀ E LIMITAZIONI DEL FILE SYSTEM V7

- **Limitazioni Temporali:** Problemi legati alla rappresentazione di date e orari.
- **Semplicità ed Efficienza:** Struttura del file system progettata per essere semplice ma efficace nell'accesso e nella gestione dei file.
- **Impatto su Sistemi Successivi:** Il design e le caratteristiche del file system V7 hanno influenzato lo sviluppo di file system UNIX successivi, tra cui quelli utilizzati in sistemi Linux moderni.



EVOLUZIONE DEI FILE SYSTEM IN LINUX - DA EXT A EXT2

- **Ext (1992): Il Primo File System di Linux**
 - Creato specificatamente per il kernel di Linux da Rémy Card.
 - Superava i limiti del file system MINIX, con una capacità massima di 2 GB.
 - Primo ad utilizzare il Virtual File System (VFS) nel kernel Linux.
- **Transizione a Ext2 (1993)**
 - Ext2 introdotto per risolvere problemi di Ext, come la immutabilità degli i-node e la frammentazione.
 - **Immutabilità:** una volta creati, gli attributi principali di un i-node (come il suo numero identificativo) non cambiano per tutta la durata della vita del file a cui sono associati.
 - Competizione con Xafs, ma Ext2 prevale per la sua maggiore affidabilità a lungo termine.



DALL'EXT3 ALL'EXT4 - INNOVAZIONE E STABILITÀ

- **Ext3: L'Introduzione del Journaling**

- Ext3 è stato sviluppato come successore di Ext2, aggiungendo il journaling per una maggiore integrità dei dati.

- **Nascita e Sviluppo di Ext4 (2006-2008)**

- Inizialmente estensioni retrocompatibili di Ext3, sviluppate tra il 2003 e il 2006.
- Scelta di biforcere Ext3 in Ext4 per evitare impatti sulla stabilità di Ext3.
- Ext4 marcato come stabile nel 2008, con rilascio nel kernel 2.6.28.

- **Adozione di Ext4 da Google**

- Google passa da Ext2 a Ext4 per il suo storage nel 2010.
- Android 2.3 adotta Ext4 al posto di YAFFS nel 2010.



FILE SYSTEM EXT4 DI LINUX

- **Journaling e Affidabilità:**

- Introduce il journaling per prevenire la perdita di dati in caso di crash.
- Utilizza un Journaling Block Device (JBD) per le operazioni di log.

- **Miglioramenti rispetto a Ext2 e Ext3:**

- Aumento della dimensione massima dei file e dei file system.
- Introduzione degli "extent" per una gestione efficiente dei blocchi di memoria contigui.
 - Gli "extent" in ext4 sono strutture che indicano un intervallo contiguo di blocchi su disco, specificando l'indirizzo di inizio e la quantità di blocchi consecutivi.
 - Questo approccio semplifica la gestione di blocchi contigui, riducendo il numero di voci necessarie per descrivere la locazione dei dati su disco, specialmente per file di grandi dimensioni.
- Compatibile con ext2 ed ext3, ma con prestazioni e capacità superiori.

- **Configurazione e Opzioni:**

- Possibilità di configurare il journaling solo per i metadati o per l'intero disco.
- Supporto a file di dimensioni fino a 16TB e file system fino a 1EB.



UNO SGUARDO AL FUTURO: BTRFS

▪ Panoramica su Btrfs - Il File System Avanzato

• Btrfs: (pronuncia: «better F S»)

- *Copy-on-Write*: Quando un file è duplicato, Btrfs condivide il file originale invece di creare una copia, riducendo lo spazio occupato.
- *Prevenzione della Perdita di Dati*: quando i dati vengono modificati, Btrfs non sovrascrive i dati esistenti, ma invece scrive le modifiche in una nuova posizione sul disco

• Caratteristiche Salienti di Btrfs

- **Supporto per File Enormi**: Gestisce file fino a 16 exbibyte (18446.7 petabyte).
- **Archiviazione Efficiente**: Riduce il sovraccarico nei metadati dei file, ottimizzando così la gestione dello spazio e delle prestazioni.
- **Supporto RAID**: Compatibile con RAID 0, 1 e 1+0 per striping e mirroring dei dati.
- **Deframmentazione e Ridimensionamento Facili**: Operazioni eseguibili mentre il filesystem è attivo.
- **Allocazione Dinamica degli Inode**: Evita l'esaurimento degli inode, salvandoli per un gran numero di piccoli file.
- **Supporto per Snapshot**: Permette la creazione e il ripristino facili degli snapshot (backup) del filesystem.
- **Supporto Checksum**: Riduce il rischio di corruzione dei dati attraverso blocchi di dati verificati costantemente.
- **Ottimizzazione per SSD**: Migliora le prestazioni degli SSD, estendendone la durata.



CONFRONTO TRA BTRFS E EXT3/EXT4

- **Btrfs: File System Avanzato**

- Progettato per l'era dei moderni dispositivi di archiviazione.
- Supporta snapshot e rollbacks, ottimale per backup e ripristini.
- Gestione nativa del RAID e miglioramento nell'integrità dei dati con checksum.
- Compressione dei dati e deduplicazione per un utilizzo efficiente dello spazio.

- **Ext3/Ext4: Affidabilità e Stabilità**

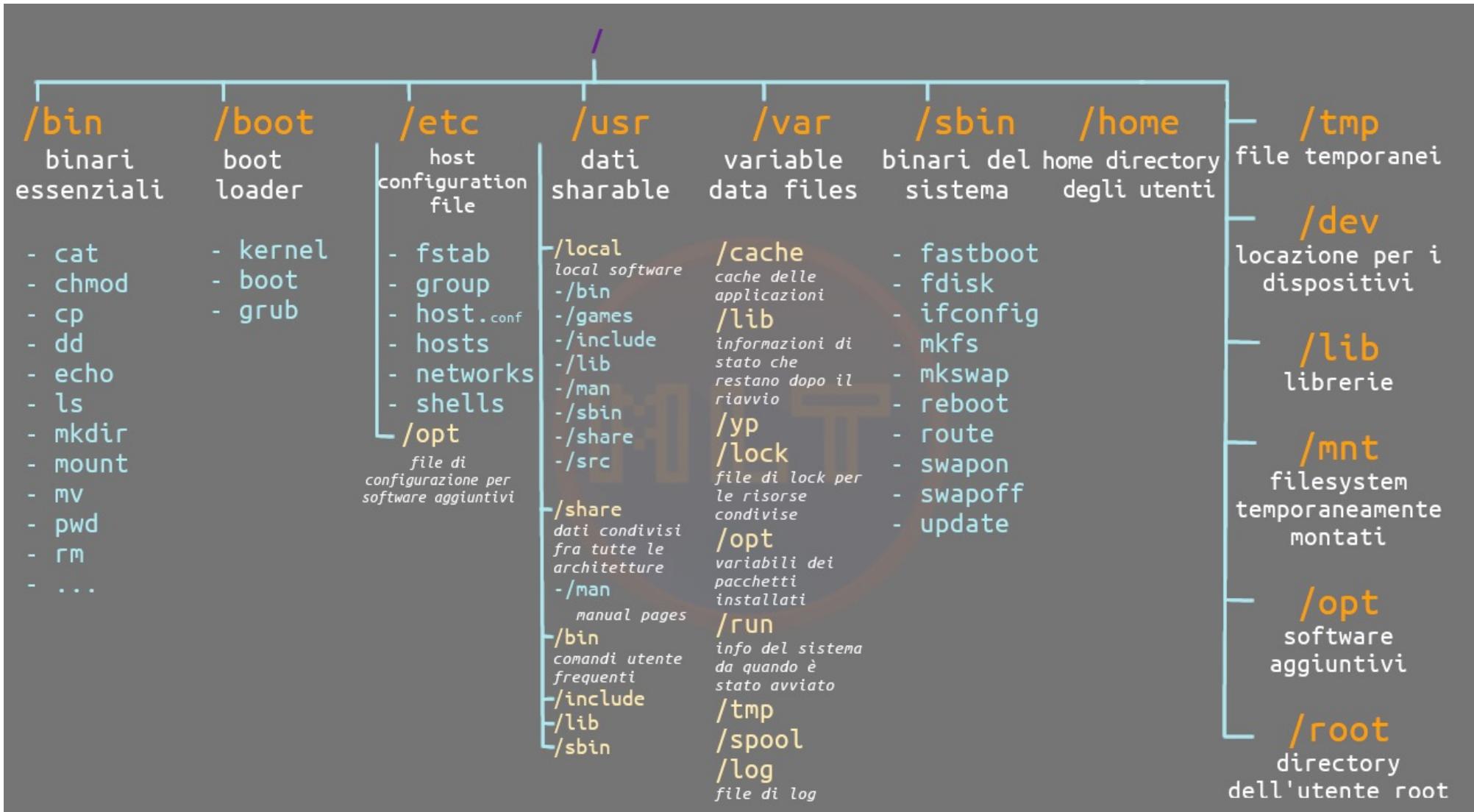
- Ext3: Affidabile con supporto journaling, ma limitato in termini di funzionalità avanzate.
- Ext4: Miglioramenti nell'efficienza, supporto per file di grandi dimensioni, riduzione della frammentazione.

- **Vantaggi e Svantaggi**

- Btrfs offre funzionalità avanzate ma è meno maturo e testato rispetto a Ext4.
- Ext3/Ext4 è noto per la sua stabilità e prestazioni, ma manca di alcune delle caratteristiche moderne di Btrfs.



STRUTTURA DELLE CARTELLE IN LINUX



STRUTTURA DELLE CARTELLI IN LINUX (2)

- **/bin:** contiene i binari dei principali comandi eseguibili dal sistema: cat, ls, pwd, ecc...
- **/boot:** contiene tutti i file necessari al Boot Loader per il processo di avvio del sistema. Inoltre, contiene il Kernel.
- **/etc:** contiene tutti i file di configurazione del sistema e di controllo
- **/usr:** contiene i pacchetti del sistema e le directory e le applicazioni dell'utente
- **/var:** contiene file temporanei, log, ecc...
- **/sbin:** ci sono i binari del sistema, quindi anche comandi come ifconfig, mkfs, fdisk , ecc...
- **/dev:** i file descrittori dei device, ovvero dei dispositivi/periferiche come gli hard disk interni. In linux anche i dispositivi vengono visti come file!
- **/home:** cartella principale dell'utente dove vengono salvati i file locali, contenente le cartelle: Documenti, Immagini, ecc...



STRUTTURA DELLE CARTELLI IN LINUX (3)

- **/lib**: contiene le librerie essenziali, incluso il compilatore C e le relative librerie.
- **/media**: contiene cartelle che servono per gestire media rimovibili «montati» automaticamente dal sistema, come cd, floppy, ecc...
- **/mnt**: usata per eseguire il mount manuale dei filesystem temporanei dei dispositivi rimovibili come USB e cd.
 - Infatti, una volta attaccata una chiavetta usb, se non “montassimo” il suo filesystem per poterne leggere il contenuto, non saremmo in grado di visualizzarne le informazioni.
 - Solitamente c’è una cartella per ogni dispositivo montato. (Vedi dopo)
- **/opt**: abbreviazione di “Optional”, contiene sia gli add-ons di alcuni software, sia programmi che non sono necessari al sistema.
 - Molte persone usano questa cartella per creare sotto-directory in cui compilare e installare programmi.
- **/root**: home dell’utente root.



VISUALIZZAZIONE DELLE PARTIZIONI E FILE SYSTEM CON MOUNT E ALTRI COMANDI

▪ Scoprire Partizioni e File System Disponibili:

- **Comando lsblk**
 - Mostra un elenco dei dispositivi di blocco, inclusi dischi e partizioni.
 - Esempio: lsblk per visualizzare tutte le partizioni e i dispositivi.
- **Comando fdisk -l**
 - Elenca dettagliatamente tutte le partizioni sui dischi, comprese quelle non montate.
 - Richiede privilegi di root: sudo fdisk -l

▪ Visualizzazione dei File System Montati:

- **Comando mount -l**
 - Elenco dei file system attualmente montati con le loro opzioni di montaggio e etichette.
 - Utile per vedere rapidamente dove e come sono montate le partizioni e i file system.

▪ Importanza di Conoscere le Partizioni:

- Fondamentale per gestire correttamente lo spazio su disco e l'organizzazione dei dati.
- Cruciale per operazioni di backup, ripristino e manutenzione del sistema.



MONTAGGIO DI PARTIZIONI E FILE SYSTEM CON IL COMANDO MOUNT

▪ Montaggio di una Partizione/File System:

- Sintassi Base:
 - mount [opzioni] <dispositivo> <directory>.
 - Esempio: `sudo mount /dev/sda1 /mnt/mydisk` per montare `/dev/sda1` in `/mnt/mydisk`.
- Creazione della Directory di Montaggio:
 - La directory di destinazione deve esistere prima del montaggio (Esempio: `mkdir /mnt/mydisk`).

▪ Opzioni di Montaggio Comuni:

- Specifica del Tipo di File System: `-t <tipo>`, es. `-t ext4`.
- Opzioni Aggiuntive: `-o <opzioni>`, es. `-o ro` per montaggio in sola lettura.

▪ Smontaggio di un File System:

- Comando `umount`:
 - Utilizzato per smontare in modo sicuro un file system o una partizione.
 - Esempio: `sudo umount /mnt/mydisk`.

▪ Note Finali:

- Queste operazioni richiedono privilegi di root.
- Smontaggio corretto è essenziale per prevenire la perdita di dati.

