Struttura del File System

Per l'anno accademico 2015-16, studiare solo le slide:

69, 70	struttura del file system,
73, 74	modulo di organizzazione dei file e file system logico
79	file control block (inode)
82	file descriptor per file aperti
88, 89	virtual file system
92, 111, 112	allocazione blocchi, schema conbinato: inode

Sommario

- Struttura del file system
- File Control Block
- Allocazione
- Gestione dello spazio libero
- Implementazione delle directory
- Prestazione
- Ripristino

Struttura del file system

- La memoria secondaria è costituita dai dischi, su cui sono memorizzati i file.
- I trasferimenti tra memoria centrale e dischi si effettuano per blocchi, composti da uno o più settori.
- Il sistema operativo può far uso di uno o più file system, ciascuno composto da più livelli che mappano la logica del file system sulla memoria secondari vera e propria.

Struttura del File System

- Il file system è **strutturato per livelli**:

5.Programmi applicativi

4.File system logico

3.Modulo di organizzazione dei file

2.File System di Base

1.Controllo dell'I/O

0. Device

1. Controllo dell'I/O

- Il livello più basso, il **controllo dell'I/O**, è costituito dai driver dei dispositivi che ricevono istruzioni ad alto livello (generiche) e le traducono in istruzioni per il dispositivo (di basso livello e specifiche).
- La memoria secondaria è un tipo specifico di I/O (con caratteristiche molto specifiche):
  - Vedremo meglio la memoria secondaria nella prossima lezione
  - vedremo meglio i driver per i dispositivi nella lezione sull'I/O

2.File system di base

- Questo strato:
  - si occupa di inviare i comandi di lettura e scrittura ai driver dei dispositivi
  - Si occupa di gestire buffer di memoria e cache che conservano i blocchi dei file, le directory, i dati del file system.

3. Modulo di organizzazione dei file

- Questo strato:
  - controlla l'allocazione dei blocchi fisici e la loro corrispondenza con quelli logici.
  - Effettua la traduzione degli indirizzi dei blocchi logici in quelli fisici.
  - Comprende il gestore dello spazio libero che registra i blocchi non assegnati e li mette a disposizione del modulo di organizzazione dei file quando sono richiesti.

File system logico

- Questo strato:
  - Gestisce i meta-dati ovvero tutte le strutture dati del file system ma non i dati stessi dei file.
  - Gestisce in particolare i File Control Block (FCB) che contengono le informazioni sui file, dal nome, alla data di accesso/scrittura, ai permessi di accesso.

Principali File System

- UFS (Unix File System) è il file system adottato dai sistemi unix-like (è a sua volta basato su Berkeley Fast File System (FFS).
- NTFS (New Technology File System), file system proprietario, proprio dei sistemi operativi microsoft basati su kernel NT,
- Sistemi riconosciuti su tutti i SO perché basati su standard. Sono usati, in particolare per i sistemi removibili. Per esempio per i supporti ottici si una il formato ISO 9660.
- Altro…. soprattutto in ambito distribuito

Strutture dati (o metadati)

- In un file system si usano molte strutture dati, che variano nei diversi file system/sistemi operativi ma rispondono a principi generali.
- Queste strutture sono mantenute in parte su disco e in parte in memoria:
  - su **disco**, prevalentemente per ragioni di dimensione;
  - In **memoria**, prevalentemente per ragioni di velocità.

Strutture su disco

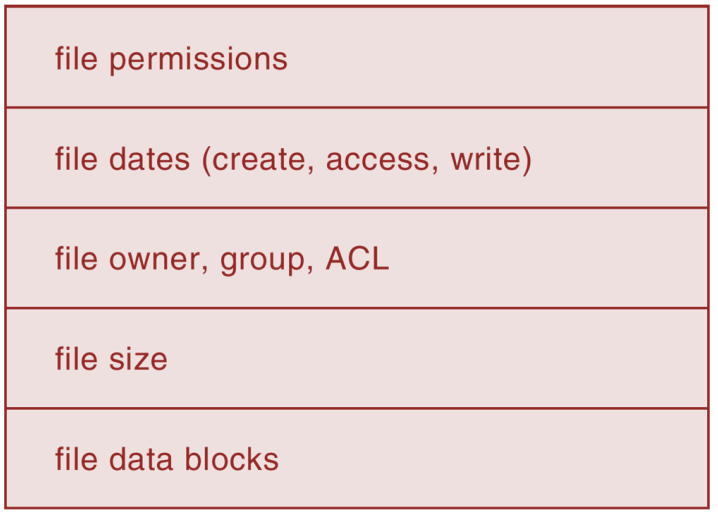
- Tra le strutture memorizzate su disco ci sono:
  - Il blocco di controllo dell'avviamento (**boot control block**), contenente le informazioni necessarie all'avviamento del sistema operativo;
  - I blocchi di controllo dei volumi (**volume control block**) contenenti i dettagli relativi a ciascun volume;
  - Le **strutture delle directory** utilizzate per memorizzare i file;
  - I blocchi di controlli dei file (**file control block**) che contengono le informazioni sui file.

Strutture in memoria

- Tra le strutture memorizzate su RAM:
  - La tabella di montaggio che contiene le informazioni riguardanti i volumi (partizioni) montati
  - La struttura delle directory relativa alle directory accedute recentemente dai processi
  - La tabella generale dei file aperti, contenente copia del FCB dei file aperti
  - I buffer per i blocchi dei file, durante la lettura e scrittura

File Control Block

- Per il sistema operativo i file vengono memorizzati in un opportuno descrittore, detto File Control Block che contiene, tra l'altro:
  - Nome
  - Proprietario
  - Dimensioni
  - Permessi di accesso
  - Posizioni dei blocchi



- **Il descrittore si chiama inode nei file system unix**

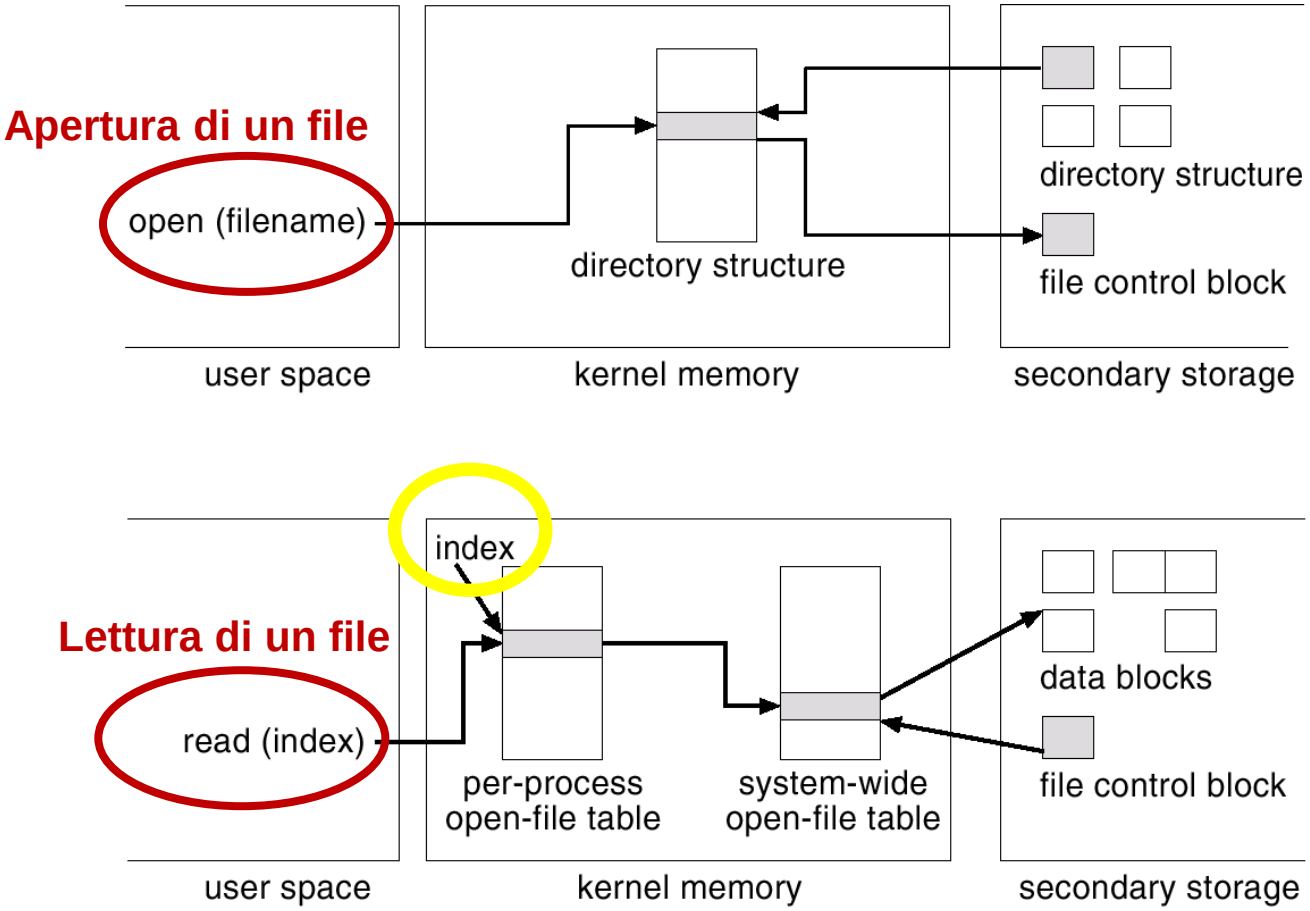
FCB: creazione

- Le applicazioni per creare un nuovo file effettuano una chiamata al file system logico (system call) che alloca un nuovo FCB.
- Il sistema carica quindi la directory appropriata in memoria centrale, la aggiorna con il nuovo FCB e la risalva su disco.
- Alcuni SO (incluso Unix e Linux) trattano le directory esattamente come i file e li distinguono con un campo apposito.
- Una volta creato il file per essere letto o scritto deve essere **aperto** (attraverso la system call **open**)

FCB: apertura

- La open:
  - Controlla se il file sia già in uso da parte di altri processi
  - Per farlo lo cerca nella tabella GENERALE dei file aperti e nel caso lo trovi, la aggiorna riferendo anche al nuovo processo
  - Se non lo trova, lo cerca nella directory e aggiunge l'opportuno elemento alla tabella GENERALE dei file aperti copiando in tabella anche il FCB originale
  - Aggiorna la tabella dei file aperti del PROCESSO, in cui copia il puntatore al FCB nella tabella GENERALE. In questa tabella locale al PROCESSO è mantenuto il puntatore alla posizione di lettura/scrittura

File Control Block: **file descriptor**



Partizioni

- Non necessariamente ogni partizione contiene un file system (con la sua struttura di directory), ma esistono dischi privi di struttura logica (raw partition).
- Queste partizioni ospitano altro:
  - Il loader del sistema operativo
  - La memoria virtuale
  - Data base con struttura specifica
  - Informazioni sulla struttura RAID

Partizione d’avviamento

- Le informazioni relativi all’avviamento del sistema sono registrate in una apposita partizione d’avviamento
- In questa fase (il caricamento) il SO non ha ancora a disposizione i driver di dispositivo del file system e dunque non potrebbe interpretarne correttamente il formato.
- Invece questa partizione vede il loader come un insieme sequenziale di blocchi che vengono caricati in memoria come immagine d’avviamento
- L’immagine di avviamento può contenere informazioni sui più sistemi operativi che possono co-esistere su un disco

Mounting delle partizioni

- Nella fase di caricamento del SO si esegue il mounting della partizione radice (**root partition**) che contiene il SO e in particolare il kernel
- In certi SO il montaggio delle altre partizioni avviene immediatamente dopo quello della root partition, in altri avviene in una fase successiva
- Durante il mounting delle partizioni, il SO controlla che il dispositivo contenga un file system valido e se l'operazione va a buon fine il sistema aggiunge il volume nella tabella di montaggio in memoria (che contiene appunto i volumi montati)

Mounting in Windows

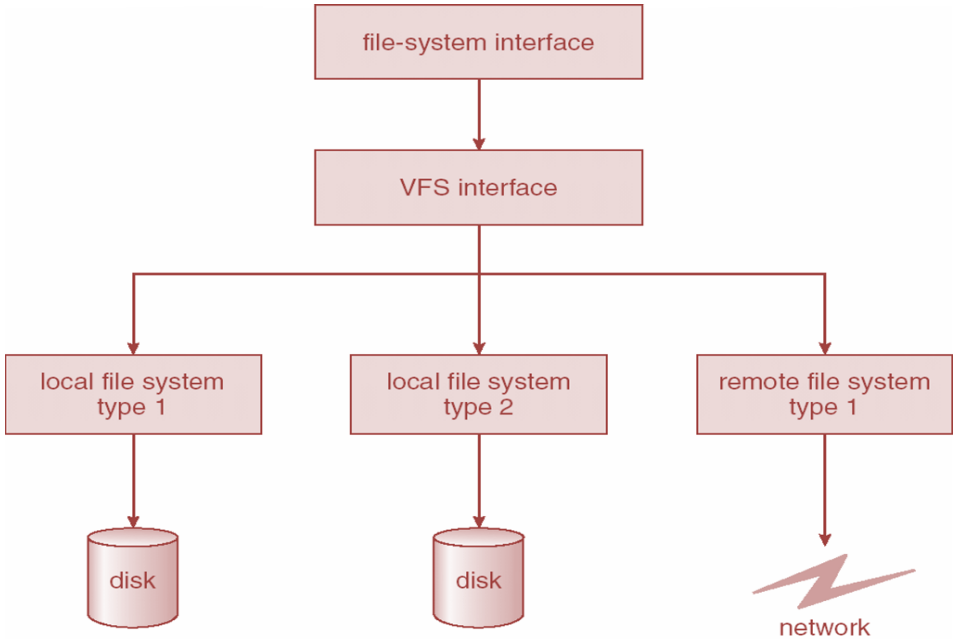
- Nei sistemi Windows, originalmente il montaggio di ogni volume era effettuato in uno spazio di nomi separato (ogni nome è costituito da una lettera seguito da :, come c:, d: ecc). Quindi il SO manteneva un puntatore che associa ciascuna lettera ai dati della partizione
- Nei sistemi più recenti, il mounting può avvenire in qualunque punto di una struttura di directory esistente (guardare il comando **net use**)

Mounting in Unix

- Nei sistemi Unix-like il mounting del file system può avvenire in qualunque punto di una directory esistente
- Il sistema imposta un flag nell'inode di quella directory (mantenuto in memoria) e imposta un campo nell'inode che punta alla tabella di montaggio.
- La tabella di montaggio a sua volta punta al superblocco della nuova partizione appena montata

Virtual File System

- Per gestire più file system i sistemi operativi devono introdurre livelli d'astrazione che rendano efficiente la scrittura delle applicazioni
- In particolare nei sistemi Unix-like si utilizza un approccio object-oriented detto **Virtual File Systems** (VFS)



VFS

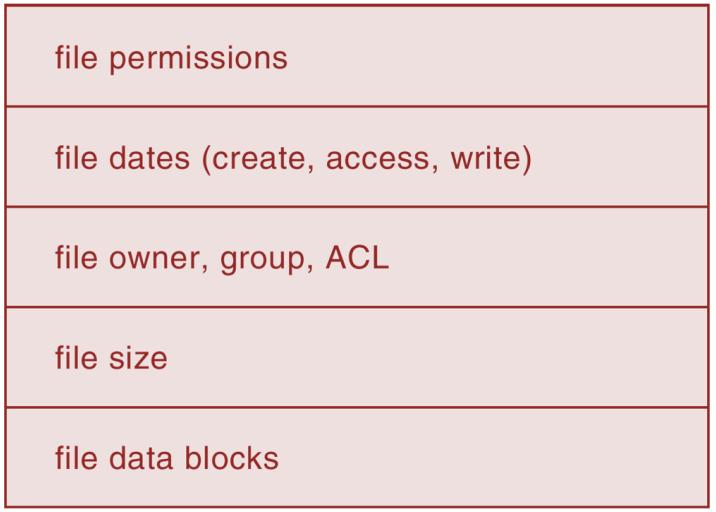
- Il Virtual File System:
  - Permette di utilizzare la stessa interfaccia alla system call (API) nei diversi file systems
  - Consente al programmatore di usare operazioni generiche e indipendenti dal file system, indipendentemente dai dettagli implementativi
  - Recapita la chiamata al file system specifico.
  - L'API in realtà è una interfaccia al Virtual File System piuttosto che ad uno specifico file system

vnode

- VFS è basato su una struttura che rappresenta il file e le directory ed è detta vnode:
  - Consente l'integrazione di file system di rete (Network File System, NFS)
  - Mentre gli inode hanno un id univoco solo all'interno del singolo file system, l'identificatore del vnode è univoco per tutta la rete e per ciascun file

Riassunto

- Design del file system:
  - File (tipi, accessi, …)
  - Directory (ciclicità)
  - Partizioni (mounting)
- Implementazione del file system:
  - File control block



Prestazioni

- Gli algoritmi di allocazione e gli algoritmi di gestione delle directory hanno un notevole impatto sulle prestazioni del file system
- Vedremo quindi diverse alternative per:
  - L'allocazione dei blocchi:
    - Ci sono diverse soluzioni che vediamo tra un po', e
  - La gestione delle directory:
    - Lista lineare
    - Tabella di hash

Gestione delle directory

- Lista lineare
- Tabella di hash

Lista Lineare

- La soluzione più semplice consiste nel realizzare la directory come una lista contenente i nomi dei file e i puntatori ai rispettivi blocchi
  - Creare un file: esaminare la directory per verificare che non esista già un file con quel nome e poi crearlo
  - Cancellare un file: cercarlo e aggiustare la lista:
    - Marcandolo libero e lasciandolo lì
    - Agganciandolo a una lista di elementi liberi
    - Ricopiandoci sopra l'ultimo elemento della lista e accorciandola

Lista Lineare

- Il principale vantaggio di questo approccio è che l'elenco dei file della directory si ottiene molto semplicemente
- Il principale svantaggio nella realizzazione della directory come lista lineare è la necessità di cercare con un consto lineare (tutte le operazioni comportano una ricerca):
  - Usare memoria più veloce mettendo le directory più utilizzate nella cache
  - Usare strutture più adatte alla ricerca come B-Tree

Hash Table

- Un altro approccio alla realizzazione della directory consiste nel velocizzare la ricerca mediante l'uso di una **hash table**
- Per individuare la corretta posizione del file nella tabella si usa una funzione di hash sul suo nome
- Inserimento e rimozione sono molto semplici ma occorre risolvere le **collisioni** che possono essere prodotte dalla funzione di hash:
  - Gestendo le collisioni al crescere della tabella
  - Associando ad ogni indice una lista di nomi

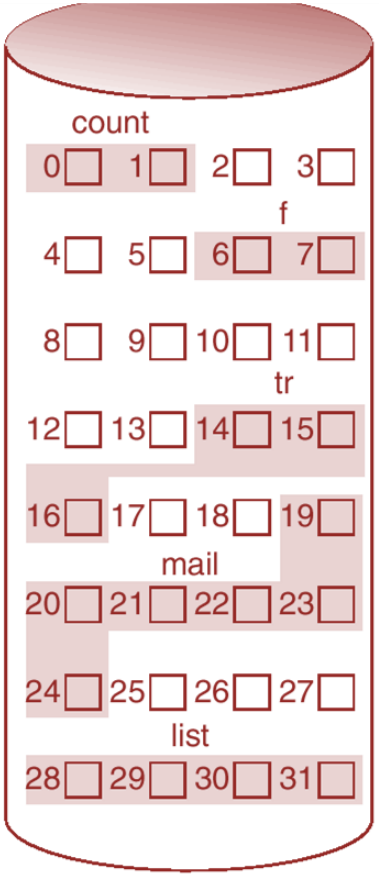
Allocazione dei blocchi

- L'altro elemento che incide fortemente sulle prestazioni è il **metodo di allocazione dei blocchi**, che specifica come i blocchi del disco sono allocati ai diversi file.
- Ci sono sostanzialmente tre metodologie per allocare i blocchi:
  - Contiguous allocation (allocazione **contigua**).
  - Linked allocation (allocazione **concatenata**).
  - Indexed allocation (allocazione **indicizzata**).

Allocazione contigua

- Nell'**allocazione contigua** ogni file deve occupare un insieme di blocchi contigui del disco.
  - Gli indirizzi del disco definiscono un ordinamento lineare dei blocchi.
  - L'accesso sequenziale al blocco **b+1** seguente all'accesso al blocco **b** non richiede spostamento della testina.
  - L'accesso diretto è fatto semplicemente calcolando la posizione assoluta in base a quella relativa.
  - Il file è definito dalla posizione iniziale e dalla lunghezza.
  - L'accesso è molto semplice.

Allocazione contigua



directory

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

Allocazione contigua

- La difficoltà sta nel reperire una porzione di disco sufficientemente grande da contenere tutto il file: il sistema di gestione dello spazio libero deve risolvere questo problema e alcune soluzioni sono lente.
- Il problema è simile a quello dell'allocazione della memoria e in questo caso le strategie più utilizzate sono **first fit** e **best fit**.
- Il sistema soffre di **frammentazione esterna** che viene risolta con routine di **deframmentazione.**

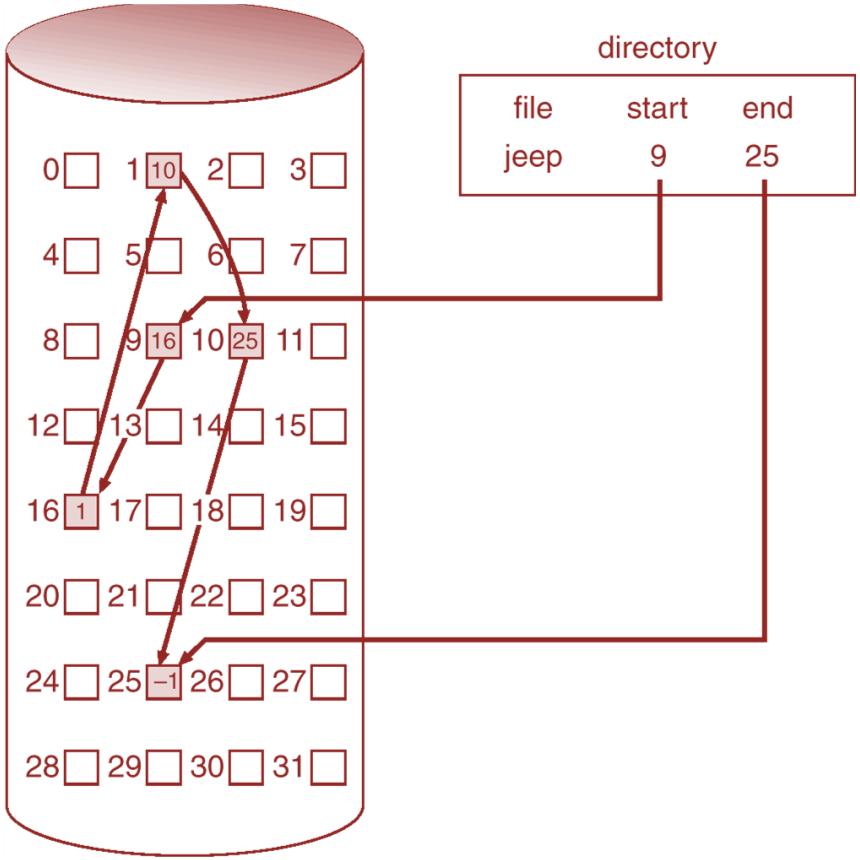
Allocazione contigua

- Se lo spazio allocato è quasi uguale alla dimensione del file il file è difficilmente **estendibile**, poiché occorre **copiare** il contenuto del file in una nuova allocazione più ampia prima di consentire l'estensione.
- Se per evitare la copia si alloca uno spazio più grande della dimensione attuale del file (**preallocazione**), si genera **frammentazione interna**.
- Si possono usare meccanismi per compattare lo spazio, recuperando quello perso in frammentazione esterna mediante copie di spostamento (ma l'operazione è lenta e non risolve la frammentazione interna).

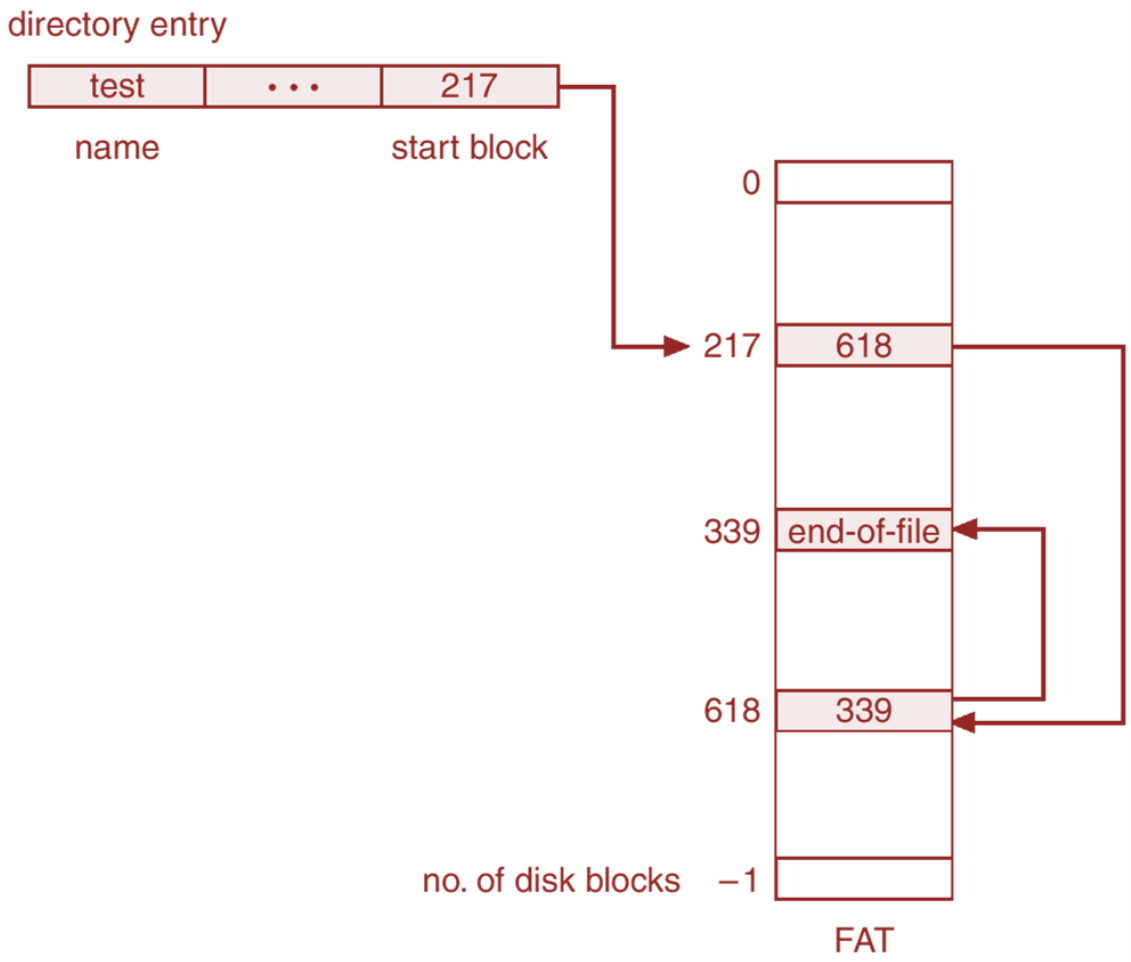
Allocazione concatenata

- Per risolvere i problemi di frammentazione introdotti dall'allocazione contigua si può utilizzare l'**allocazione concatenata**.
- Ogni file è costituito da una lista concatenata di blocchi del disco che possono essere distribuiti in qualunque parte del disco stesso.
- La directory contiene un puntatore al primo blocco del file.

Allocazione concatenata



File-Allocation Table



Allocazione concatenata

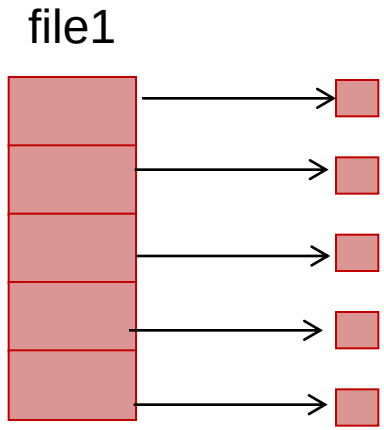
- Non esiste **frammentazione esterna** perché ogni blocco viene allocato singolarmente (quindi tutti i blocchi sono candidati ad essere allocati).
- Non esiste **frammentazione interna** perché non è necessario pre-allocare il file (che cresce al bisogno).
- In questo caso l'allocazione è semplificata a spese dell'accesso.
- **Problemi**:
  - È relativamente efficace nell'accesso sequenziale, perché comporta lo spostamento delle testine al nuovo blocco.
  - È inefficace nell'accesso diretto, perché occorre scorrere il file per trovare la locazione i-esima.
  - Usa molto spazio per i puntatori, poiché ogni blocco c'è un puntatore al successivo.

Clustering

- Per risolvere quest'ultimo problema solitamente non si allocano blocchi ma gruppi (**cluster**) di blocchi (ad esempio 4):
  - **Vantaggi**: meno puntatori, meno movimenti della testina, più semplice la gestione dei blocchi liberi.
  - **Svantaggi**: frammentazione interna (parte del blocco può rimanere inutilizzata).

Allocazione Indicizzata

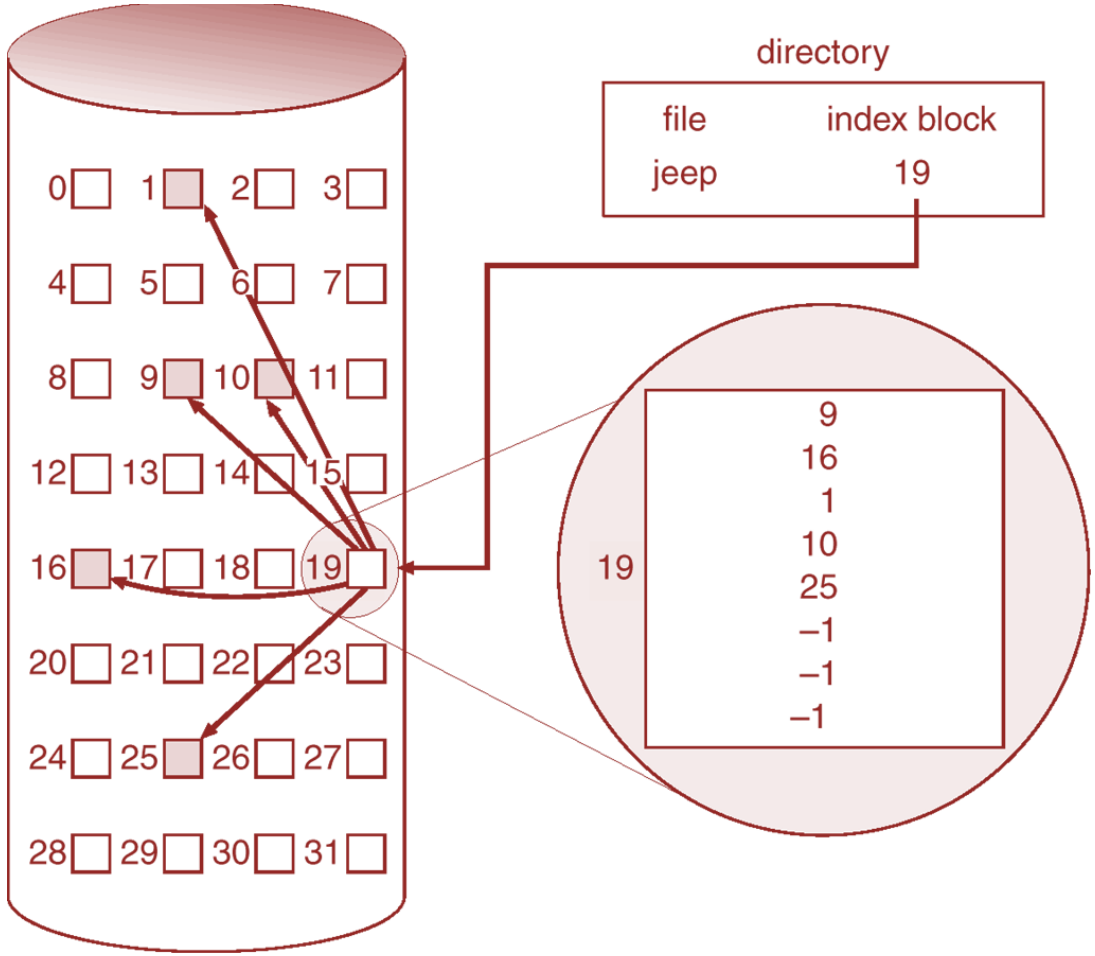
- La maggioranza dei problemi riscontrati con l'allocazione concatenata può essere risolto inserendo tutti i puntatori ai blocchi in una apposita tabella detta blocco indice (**index block**).
- Questo approccio è detto **allocazione indicizzata**.



Allocazione indicizzata

- Ogni file ha il proprio blocco indice in cui l''i-esimo elemento (del blocco indice) punta all'i-esimo elemento del file.
- Supporta l'eccesso diretto **senza frammentazione**.
- Comporta un certo **spreco di spazio** per il blocco indice che deve essere della giusta dimensione per consentire di aumentare le dimensioni del file e contemporaneamente non sprecare spazio.

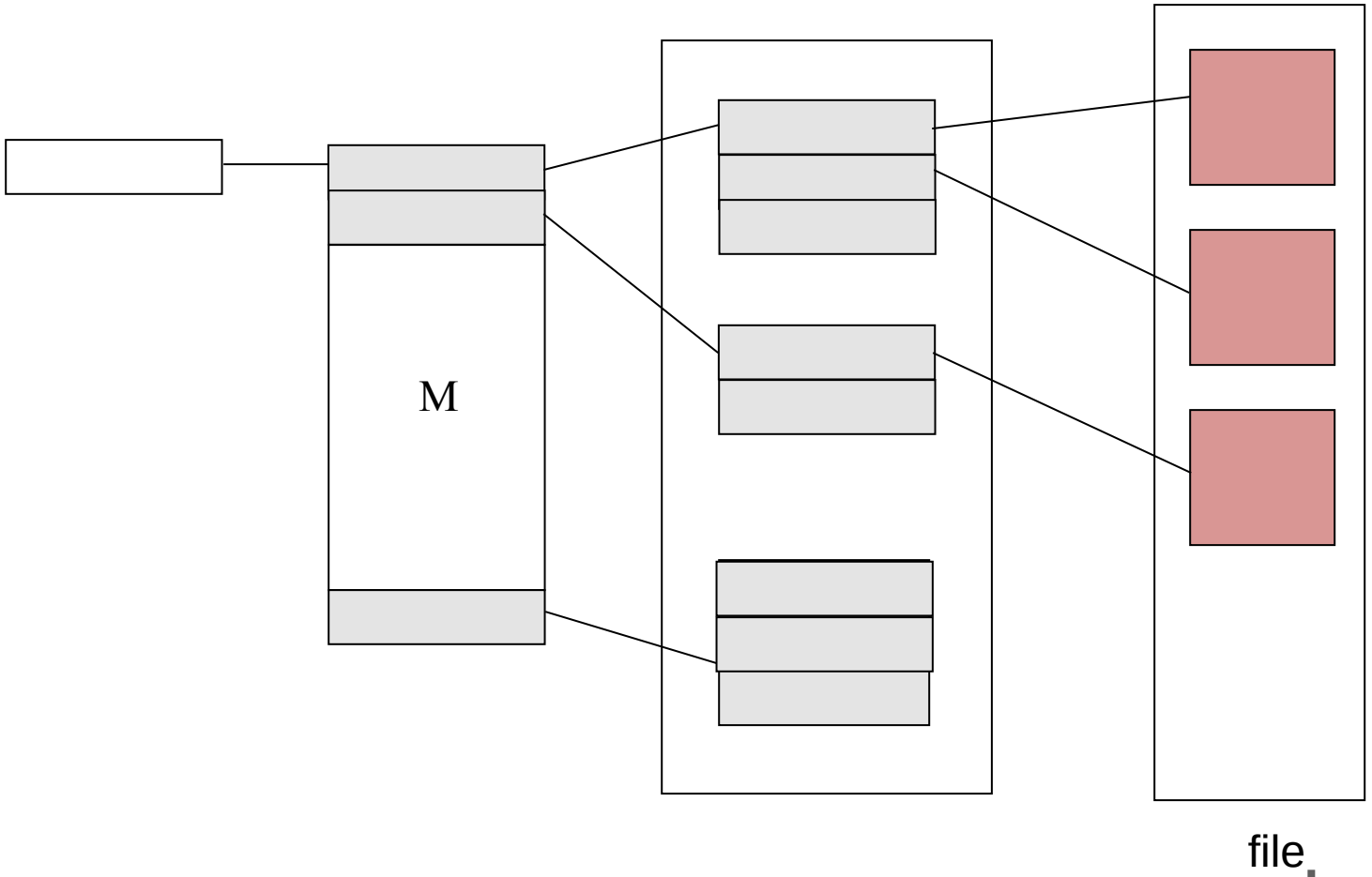
Allocazione Indicizzata



Allocazione indicizzata

- La dimensione e la **memorizzazione** del blocco indice è critica:
  - **Schema concatenato**: il blocco indice occupa esattamente un blocco. Se non è sufficiente un blocco l'ultimo puntatore del blocco indice punta ad un altro blocco indice.
  - **Indice multilivello**: il blocco indice di primo livello punta ad altri blocchi indice di secondo livello e così via
  - **Schema combinato**.

Multilivello

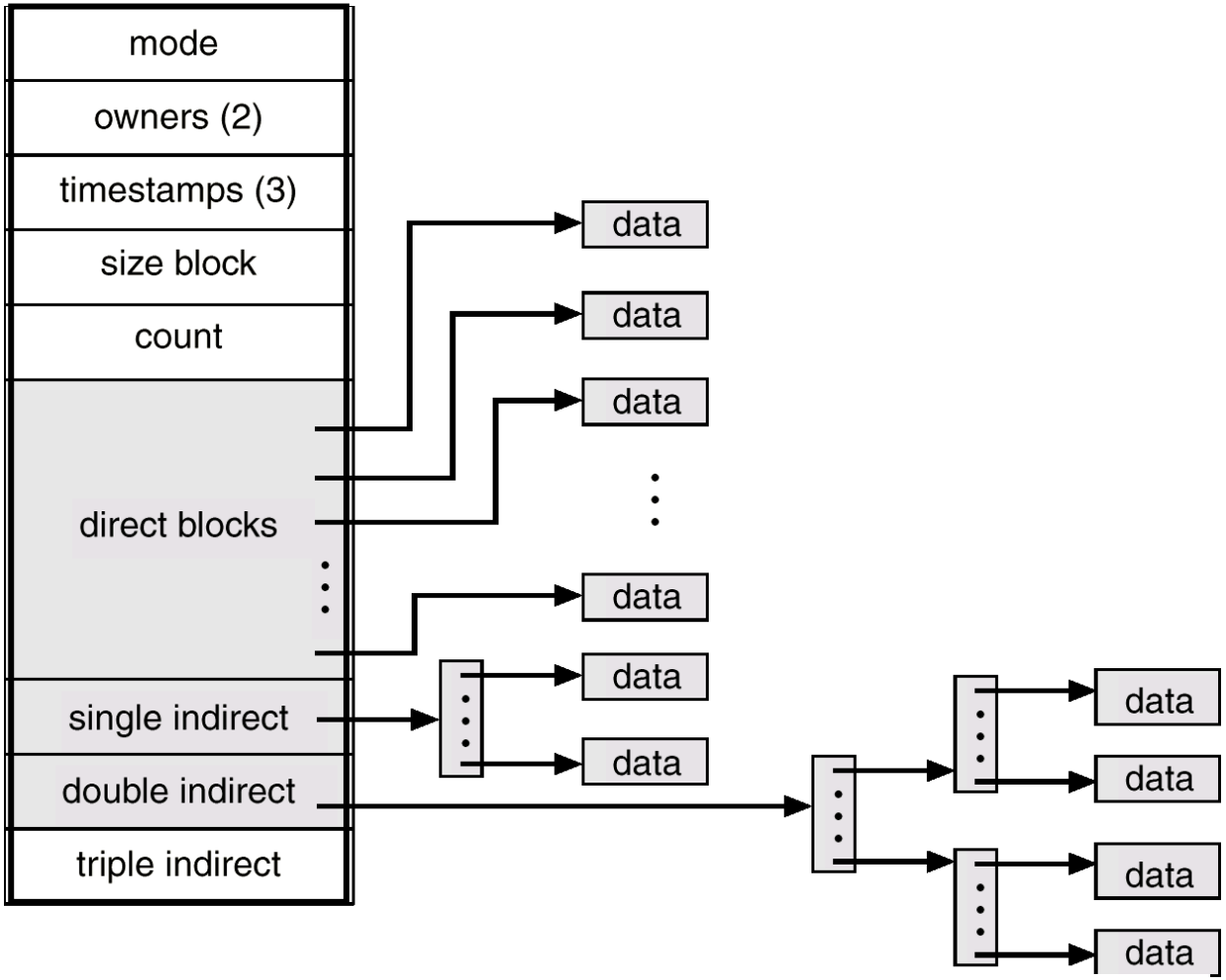


Allocazione dei Blocchi
Schema combinato

- Usato negli **inode** di alcune versioni di Unix.
- Una parte dei puntatori del blocco indice puntano direttamente ai blocchi del file (**blocchi diretti**).
- Una parte punta invece a indici multilivello (**blocchi indiretti singoli, doppi o tripli**)
- Se il file è piccolo si usano solo i blocchi diretti, più è grande e più indirezioni vengono coinvolte.

Allocazione dei Blocchi, Schema combinato

inode



Gestione dello spazio libero

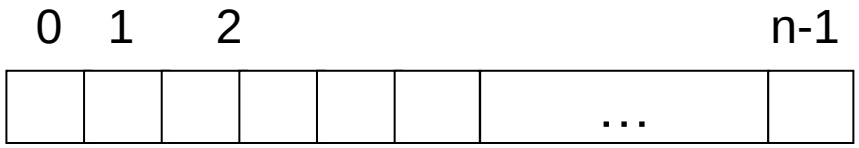
- Per tenere traccia dello spazio ancora disponibile su disco il file system gestisce la **lista dei blocchi liberi** ovvero non allocati a nessun file o directory:
  - Un blocco esce dalla lista quando viene allocato per la creazione o l'allungamento di un file
  - Un blocco rientra nella lista quando il file viene cancellato.

Gestione dello spazio libero

- Esistono differenti **implementazioni** della lista dei blocchi liberi, basate a volte su strutture dati anche diverse da una lista:
  - **Bit Vector** (**Vettore di bit**)
  - **Linked List** (**Lista concatenata**)
  - **Grouping** (**Raggruppamento**)
  - **Conteggio** (**Counting**)

Bit vector

- La lista dei blocchi liberi può essere implementata mediante una mappa di bit (o **vettore di bit**) in cui ogni blocco è rappresentato da un bit.



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

- Il metodo consente di calcolare semplicemente il **primo blocco libero** o i primi n blocchi liberi consecutivi.
  - La prima parola non 0 (se è 0 è occupata) viene scandita alla ricerca del primo bit libero (cioè del primo 1).
  - Il numero del blocco è:

    (numero di bit a word) * (numero di parole 0) + offset del primo bit 1

Bit Vector

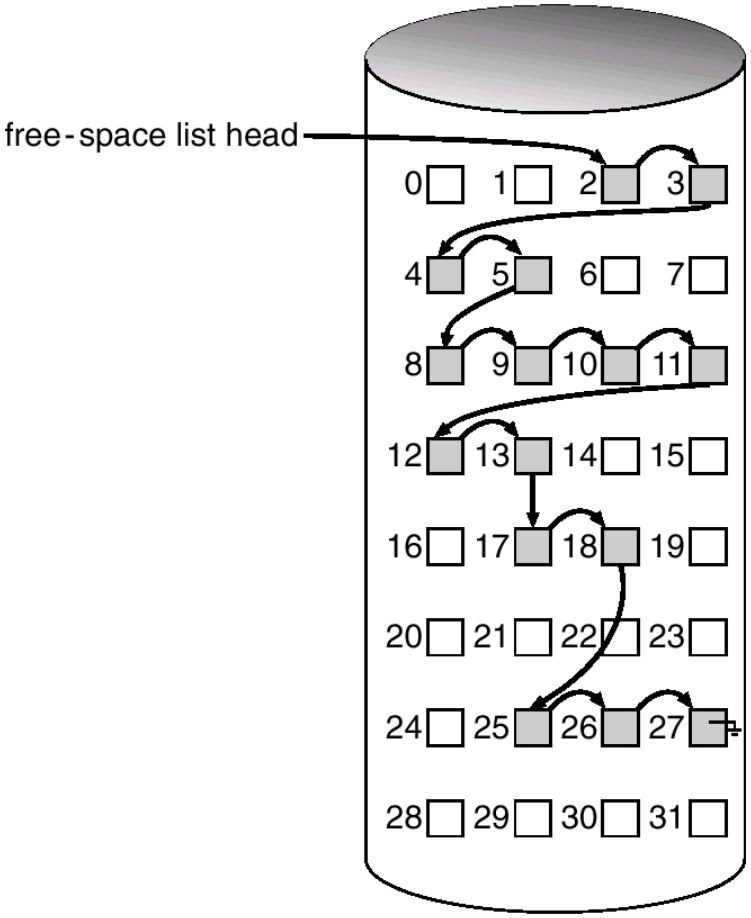
- Il vettore di bit è efficiente solo se conservato in memoria centrale.
- Richiede però **molto spazio**. Ad esempio se abbiamo al dimensione dei blocchi = 2^{12} byte (4096 byte) e la dimensione del disco =2^{30} byte (1 gigabyte) allora il numero dei bit nel vettore è:

$$n = 2^{30}/2^{12} = 2^{18} \text{ bit (o 32K byte)}$$

Lista dei blocchi liberi

- Un altro approccio consiste nel **concatenare tutti i blocchi liberi** tenendo nella memoria centrale un puntatore al primo blocco libero (alla testa)
- Il primo blocco conterrà un puntatore al blocco successivo e così via fino all'ultimo blocco che indicherà con nil il termine della lista.
- Questo meccanismo ha problemi di efficienza.

Lista dei blocchi liberi



Lista dei blocchi liberi

- Problemi di efficienza:
  - Quando si cerca un solo blocco libero la lista così realizzata è efficiente (si stacca il primo blocco libero e si riconcatena il puntatore alla testa col secondo).
  - Quando si cercano n blocchi liberi consecutivi si rischia di dover scorrere tutta la lista, con tempi di attesa piuttosto lunghi.

Alcuni correttivi

- Alcuni correttivi sono:
  - **Grouping**: il primo blocco libero contiene gli indirizzi di altri n-1 blocchi liberi più un puntatore al successivo. La lista si trasforma in un albero (a 2 livelli).
  - **Counting**: se ci sono n blocchi liberi consecutivi viene memorizzato un puntatore al primo e poi il numero di blocchi. La lista si accorcia drasticamente.

Efficienza

- L'**efficienza** del File System dipende fortemente dall'allocazione delle directory per cui è opportuno che il SO allochi un'area (grande) dell'area del disco ad accesso più veloce per la directory.
- Un'altra importante scelta riguarda il tipo (e la dimensione) dei dati che vengono mantenuti nella directory per cui bisogna evitare:
  - Aggiornamenti frequenti (data)
  - Puntatori grandi

Prestazioni

- Molti controller per migliorare le **prestazioni** dell'accesso al file system mantengono in memoria centrale informazioni cachate (per esempio la traccia che è in lettura/scrittura).
- Il SO può addirittura dedicare una parte della sua main memory a mantenere la **cache del disco**.

Prestazioni

- Si possono usare diversi algoritmi di swap sulla cache:
  - non si usa ovviamente LRU per gli accessi sequenziali
  - Se l'accesso è sequenziale è ovvio quale sarà il prossimo blocco da leggere e si può fare **lettura anticipata**.

Disco RAM

- Un'ulteriore tecnica consiste nel virtualizzare un disco in memoria (**disco RAM**) in modo trasparente all'utente che:
  - Usa il disco RAM in fase di lettura/scrittura del file.
  - È obbligatoria una implicita fase periodica di riscrittura del file su supporti permanenti.