SISTEMI OPERATIVI

(MODULO DI INFORMATICA II)

Realizzazione del file system

Prof. Luca Gherardi

Prof.ssa Patrizia Scandurra (anni precedenti)

Università degli Studi di Bergamo

a.a. 2012-13

Sommario

- Realizzazione del file system
- Realizzazione della directory
- Metodi di allocazione dei file
- Gestione dello spazio libero
- Recupero del file system
- File system basato sulla registrazione delle attività

Realizzazione del File System

- Una specifica realizzazione del file system risponde alle seguenti domande:
  - Come vengono memorizzati i file e le directory?
  - Come viene gestito lo spazio sul dispositivo di memorizzazione?
  - Quali strutture dati e algoritmi vengono usate dal SO per gestire in modo efficiente e affidabile il file system?

Struttura del file system (1)

- Il file system risiede permanentemente in memoria di massa
- Tipicamente questa memoria è fornita dai dischi, i quali vengono usati per due principali motivi:
  - Possono essere **riscritti** localmente (leggo e riscrivo lo stesso blocco)
  - Permettono l'**accesso**, sia sequenziale che diretto, **ai blocchi fisici** che memorizzano i diversi file

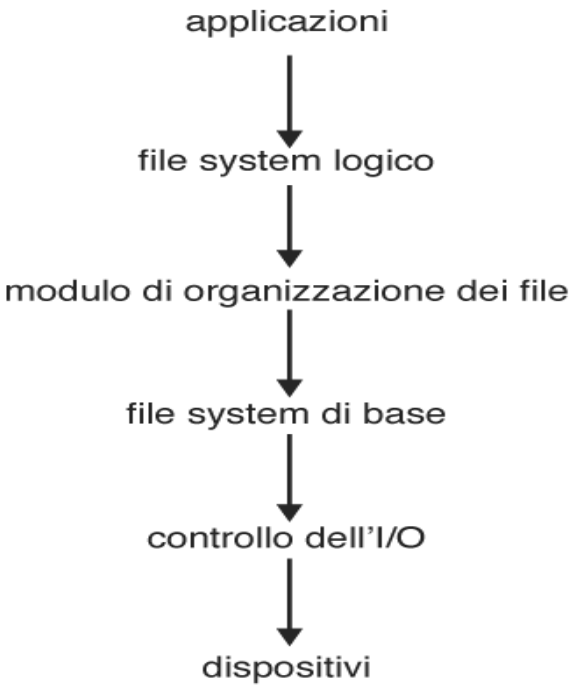
- La dimensione dei blocchi varia da dispositivo in dispotivo
  - Da 32 a 4096 Byte
  - Solitamente 512 Byte

Struttura del file system (2)

- Il file system sono tipicamente realizzati secondo una struttura a strati
  - Il livello superiore si basa sui servizi offerti dal livello inferiore
- Livelli di un file system:
  - **Controllo dell'I/O**: driver dei dispositivi e gestore degli interrupt
    - Si occupa del trasferimento dell'informazione dalla memoria di massa a quella centrale
  - **File System di base**: invia dei comandi di base al controllo dell'I/O per scrivere e leggere blocchi fisici
    - E.g. Leggi il blocco nell'unità 1, cilindro 12, superficie 4, settore 2
  - **Modulo di organizzazione dei file**: conoscendo il metodo di allocazione traduce un indirizzo di un blocco logico in un indirizzo di un blocco fisico
    - Comprende anche il gestore dello spazio libero (mantiene la lista dei blocchi liberi)
  - **File System logico**: gestisce i metadati e la struttura delle directory
    - Gli attributi dei file vengono salvati in File Control Blocks (FCB)

Struttura del file system (3)

- Il codice è più riutilizzabile
  - Diversi file system possono essere implementati nello stesso SO riusando gli strati di basso livello
- La stratificazione introduce tuttavia un maggiore over-head



Realizzazione del file system (1)

- Un file system richiede la definizione di diverse strutture dati
  - Alcune risiedono in memoria di massa, altre in memoria centrale
- Strutture dati in memoria di massa:
  - **Boot control block**: contiene informazioni necessarie per avviare il SO contenuto nel disco (se il disco non contiene SO è vuoto)
  - **Volume control block**: contiene dettagli riguardanti il volume
    - Numero di blocchi fisici e dimensione
    - Contatore dei blocchi liberi e puntatori ai blocchi
    - Contatore dei FCB presenti nel volume e puntatori ad essi
  - **Struttura della directory**: organizza i file
  - **File control blocks**: memorizza gli attributi dei file

Realizzazione del file system (2)

- Strutture dati in memoria centrale:
  - **Tabella di montaggio**: contiene informazioni relative ai volumi attualmente montati
  - **Struttura delle directory** (in parte): contiene le informazioni relative alle directory attualmente in uso dai processi
  - **Tabella dei file aperti** (livello SO)
  - **Tabelle dei file aperti** (livello processo)
  - **Blocchi de file system**: durante la loro lettura e scrittura (nei buffer)

Realizzazione del file system (3)

- Esempio 1: l'invocazione di una chiamata di sistema per la creazione di un file comporta l'uso di queste strutture dati

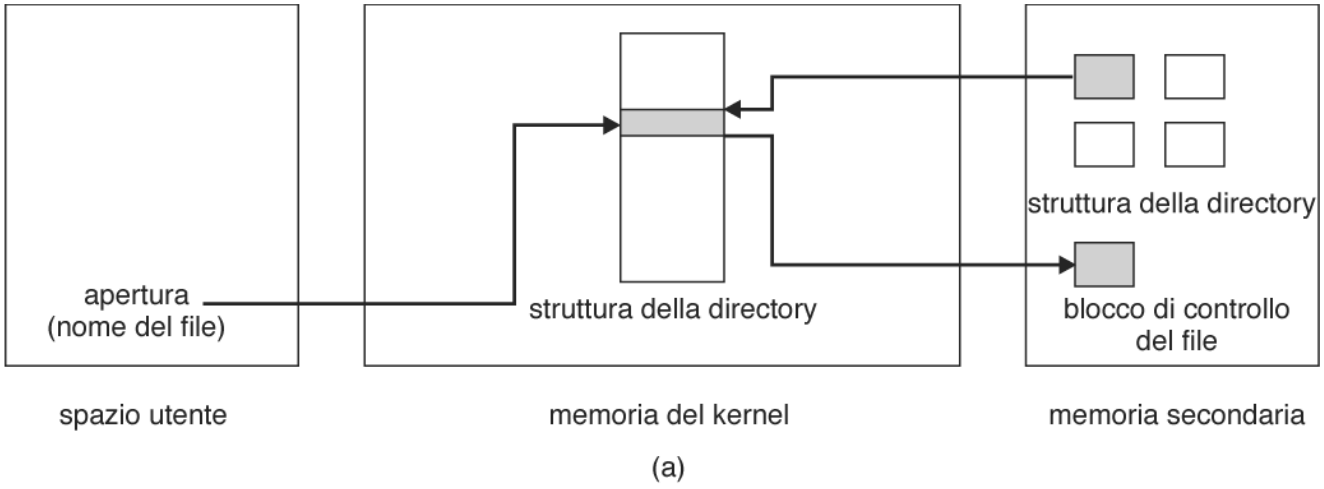
1. Il SO crea e alloca un nuovo FCB
2. Carica in memoria centrale la directory che lo dovrà contenerlo
3. Aggiorna la directory
4. Riscrive la directory in memoria di massa

Realizzazione del file system (4)

- Esempio 2: l'invocazione di una chiamata di sistema per l'apertura e la chiusura (`open()` e `close()`) di un file comporta l'uso di queste strutture dati

- Il SO controllo la tabella dei file aperti per vedere se il file è già aperto
  - In caso negativo
    - Aggiunge il FCB alla tabella del SO
  - In ogni caso
    - Aggiunge alla tabella del processo chiamante un nuovo elemento che punta alla tabella del SO
    - Incrementa il contatore delle aperture del file
- Alla chiusura del file il SO decrementa il contatore e rimuove l'elemento di quel file dalla tabella del processo chiamante
  - Se il contatore di aperture vale 0 il FCB viene rimosso dalla tabella del SO

Realizzazione del file system (5)



Apertura del file

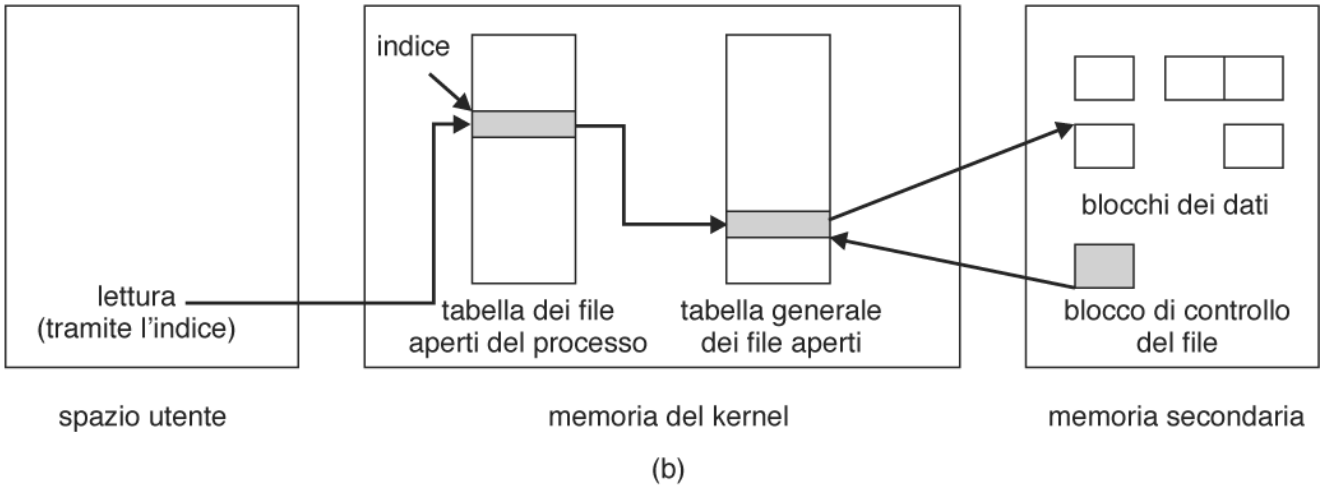


Tabelle dei
file aperti

Partizioni e montaggio

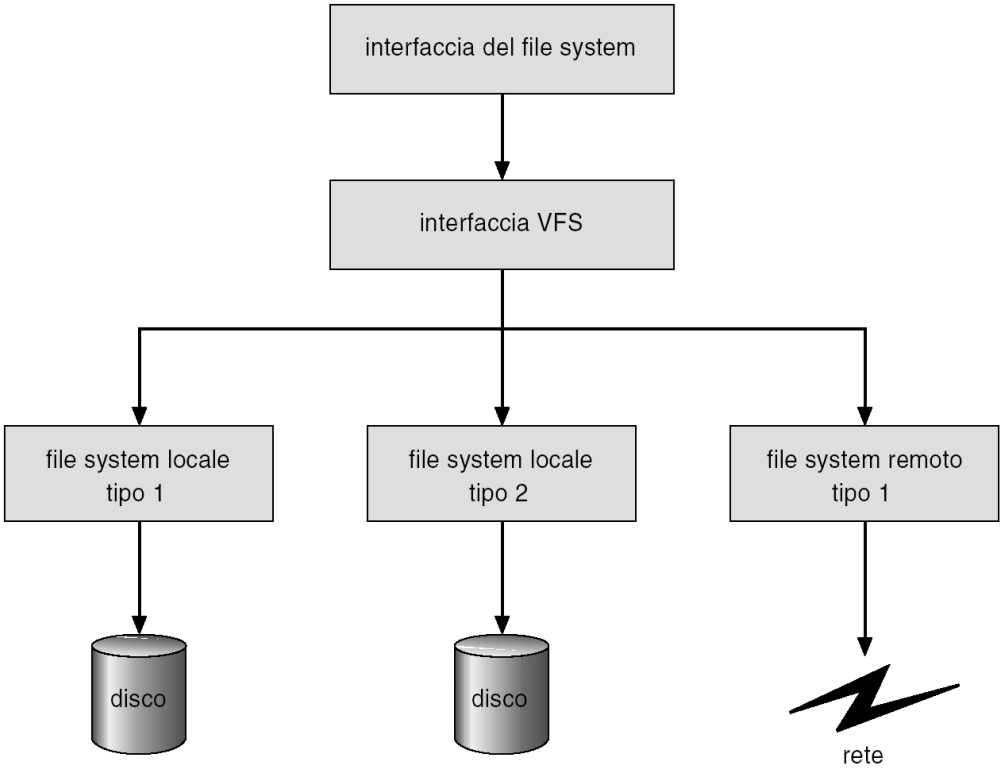
- Tipicamente un unico disco può contenere diverse partizioni
  - ognuna delle quali può avere un file system diverso
- Un partizione senza file system è detta **raw partition**
  - Un esempio è la partizione usato per lo swap nei sistemi Unix
- Un'altra partizione priva di file system, ma con un formato proprio, è quella di **boot**
  - Poiché all'avvio i driver del file system non sono ancora stati caricati
  - Consiste in un insieme di blocchi che vengono caricati in memoria centrale e contengono le **istruzioni per l'avvio del SO**
  - Un'immagine di boot può lanciare più SO
- Una volta scelto il SO la partizione radice **viene montata**
  - Essa contiene il kernel del SO ed eventuali file di sistema
  - Altre partizioni possono essere montate in modo automatico o su richiesta
  - Il SO inserisce le informazioni della partizioni montate nella **tabella di montaggio**

File System virtuali (1)

- I SO moderni devono prevedere la possibilità di **integrare diversi tipi di file system** in un'unica struttura a directory
- Questo problema viene risolto introducendo il **Virtual File Systems** (**VFS**)
  - Utilizza una tecnica "orientata agli oggetti" per implementare il file system
- VFS permette che **la stessa interfaccia di sistema** (API) sia usata per differenti tipi di file system
- È organizzato su tre livelli

File System virtuali (2)

- L'interfaccia del file system definisce le chiamate di sistema per operare sui file e sui FCB
- L'interfaccia VFS
  - **Separa** le operazione generiche del file system dalla loro realizzazione
  - Permette di rappresentare in modo univoco un file all'interno della rete (per usare anche file system di rete)
  - Ricevuta una chiamata ad un'operazione avente come parametro un file, **delega** l'esecuzione all'appropriata implementazione dell'API (in base al file system necessario)



Realizzazione della struttura delle directory (1)

- **Realizzazione a lista lineare**

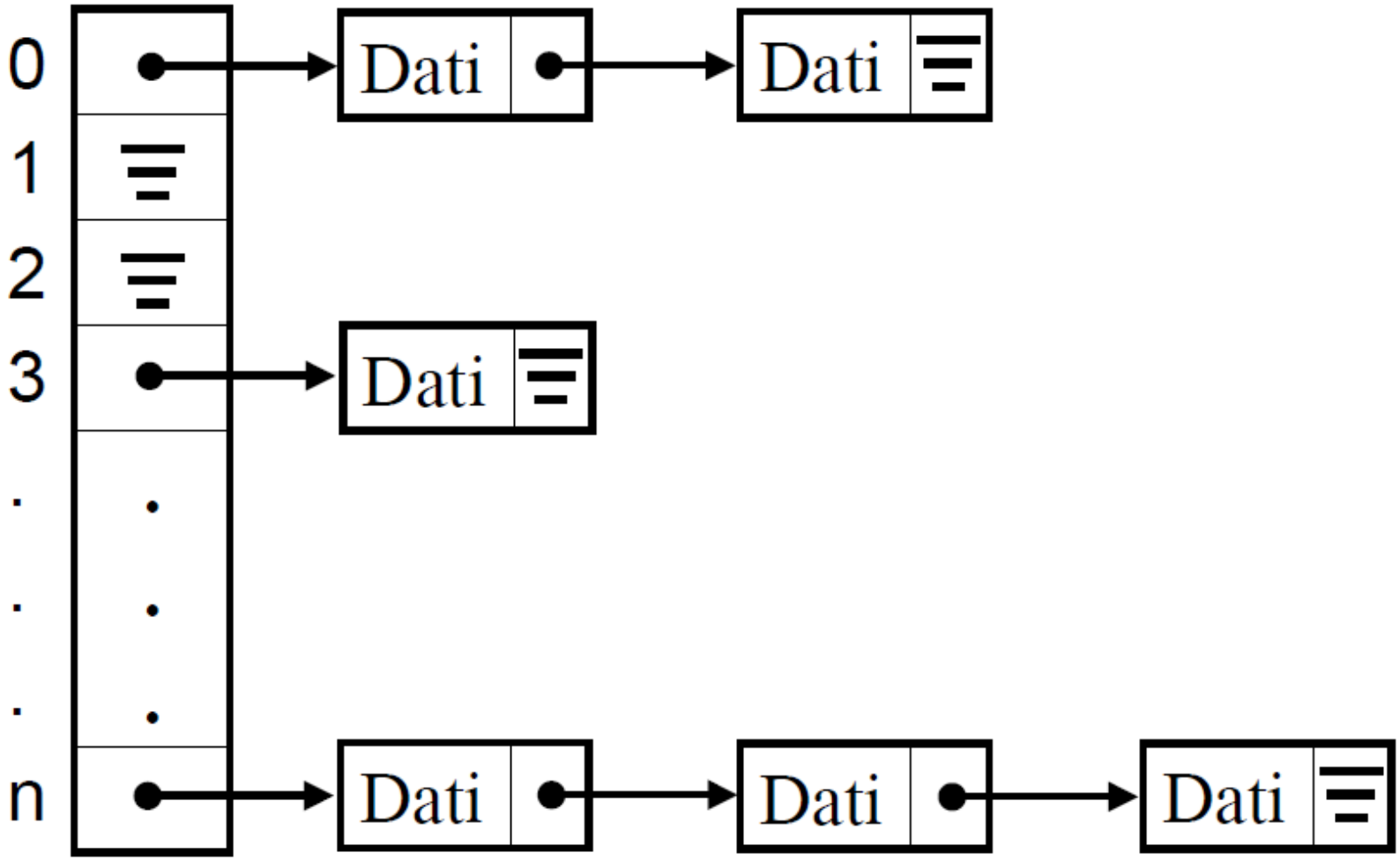
- È il modo più semplice per implementare la directory
- Consiste in un array contenente identificatore del file e puntatore al FCB
- È tuttavia poco efficiente
  - Le operazioni sui file richiedono di eseguire ricerche all'interno della lista
  - Il tempo di ricerca è proporzionale alla dimensione
- Una lista ordinata può ridurre i tempi permettendo una ricerca binari
  - Tuttavia richiede maggiori tempi di gestione, in quanto gli elementi devono essere mantenuti ordinati

Realizzazione della struttura delle directory (2)

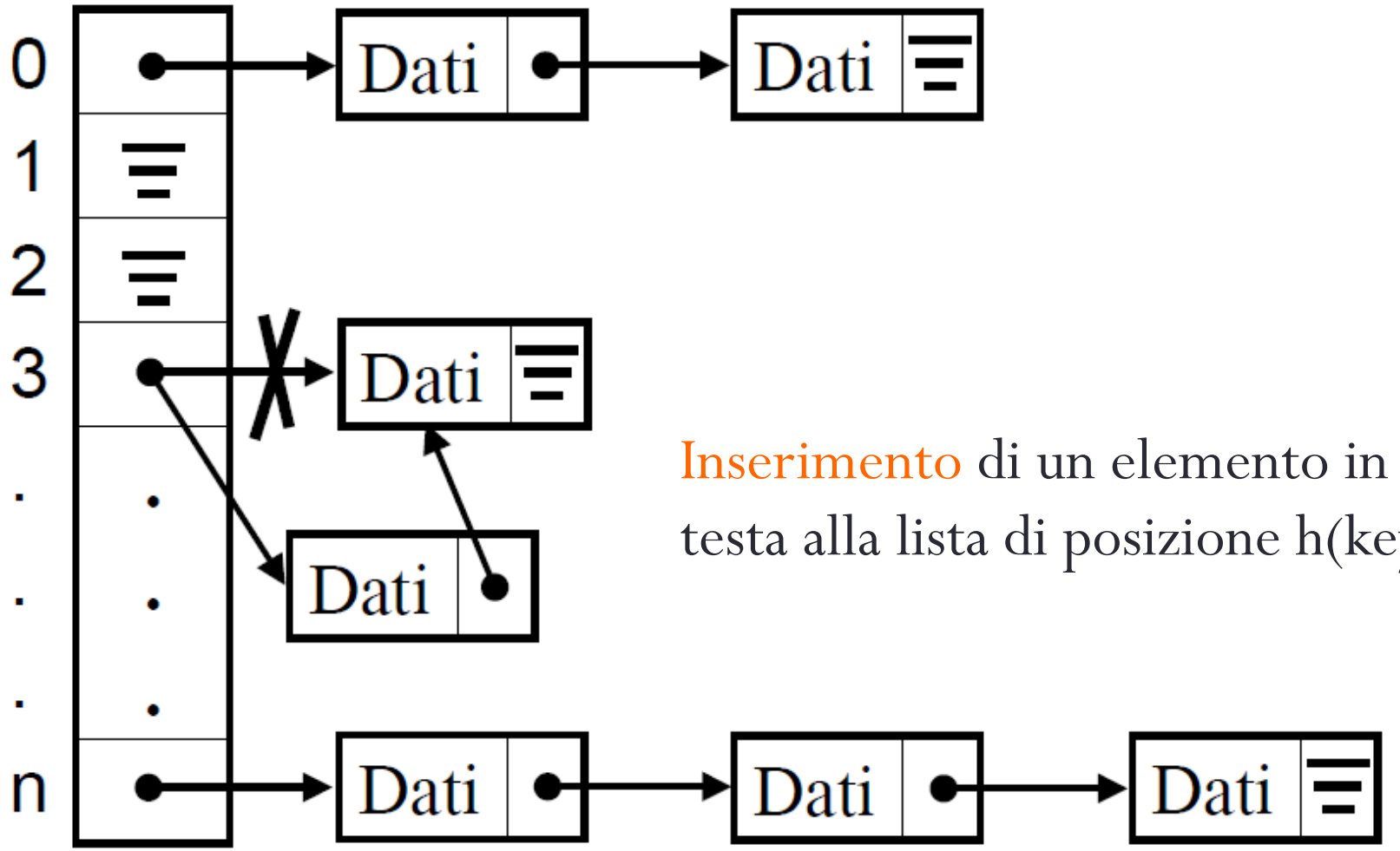
- **Realizzazione tramite tabella di hash**

- In questo caso si usa una lista lineare in combinazione con una tabella di hash
  - Chiave: nome del file
  - Valore: puntatore all'identificatore del file nella lista lineare
- Problema: evitare le **collisioni**
  - Situazioni in cui due chiavi diverse ritornano lo stesso valore
  - Per evitare collisioni la dimensione della tabella deve essere inferiore al numero di file
  - In alternativa si può usare come valore una lista concatenata di coppie <nome_file, puntatore_alla_lista_ordinata>
    - Aumenta i tempi di ricerca

Realizzazione della struttura delle directory (3)
Tabella di hash



Realizzazione della struttura delle directory (4)
Tabella di hash



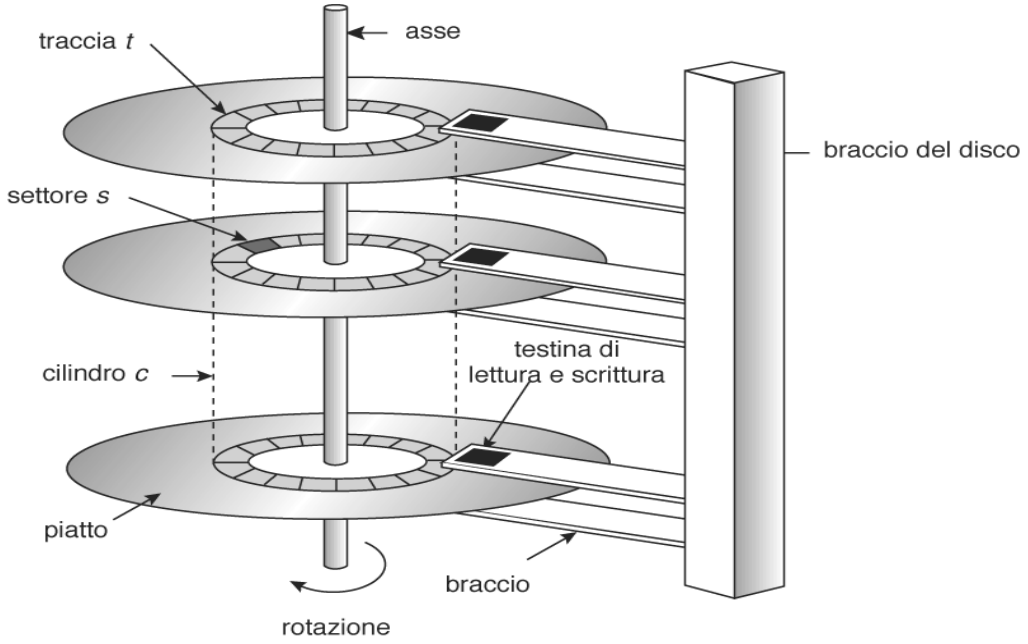
Inserimento di un elemento in testa alla lista di posizione h(key)

Metodi di allocazione dei file

- Un metodo di allocazione indica come i file devono essere allocati ai blocchi del disco:
- In modo da garantire un utilizzo rapido ed un accesso rapido
  - **Allocazione contigua**
  - **Allocazione concatenata**
  - **Allocazione indicizzata**

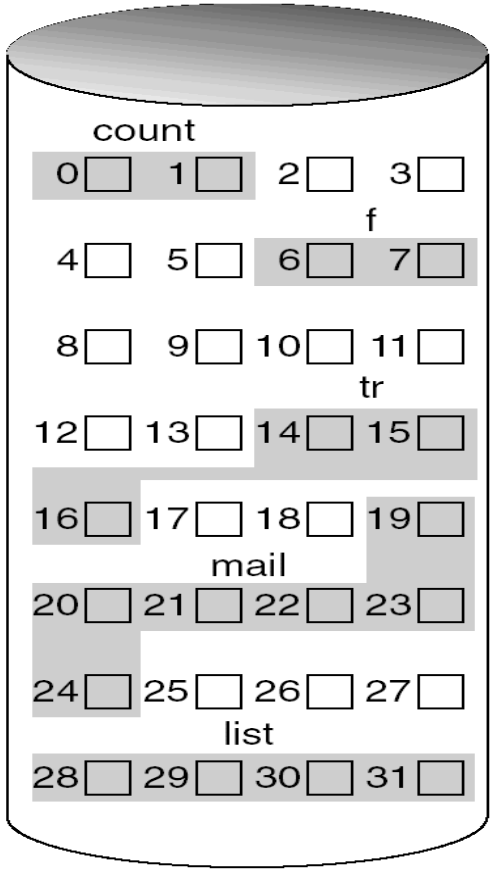
Struttura del disco

- Un disco è costituito da più piatti:
- Ogni piatto ha una testina che permette la lettura e la scrittura
- Il disco gira, quindi leggere i settori dello stessa traccia non comporta lo spostamento della testina



Allocazione contigua (1)

- Questo metodo alloca i file in **blocchi contigui** del disco
- Per questo motivo il numero di **spostamenti della testina** richiesti per accedere a tutti i blocchi di un file è **trascurabile**
- Allo stesso mondo è trascurabile il tempo di ricerca
- Un file è definito da
  - Blocco di inizio
  - Numero di blocchi
- Questi dati sono salvati nel FCB



directory

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

Allocazione contigua (2)

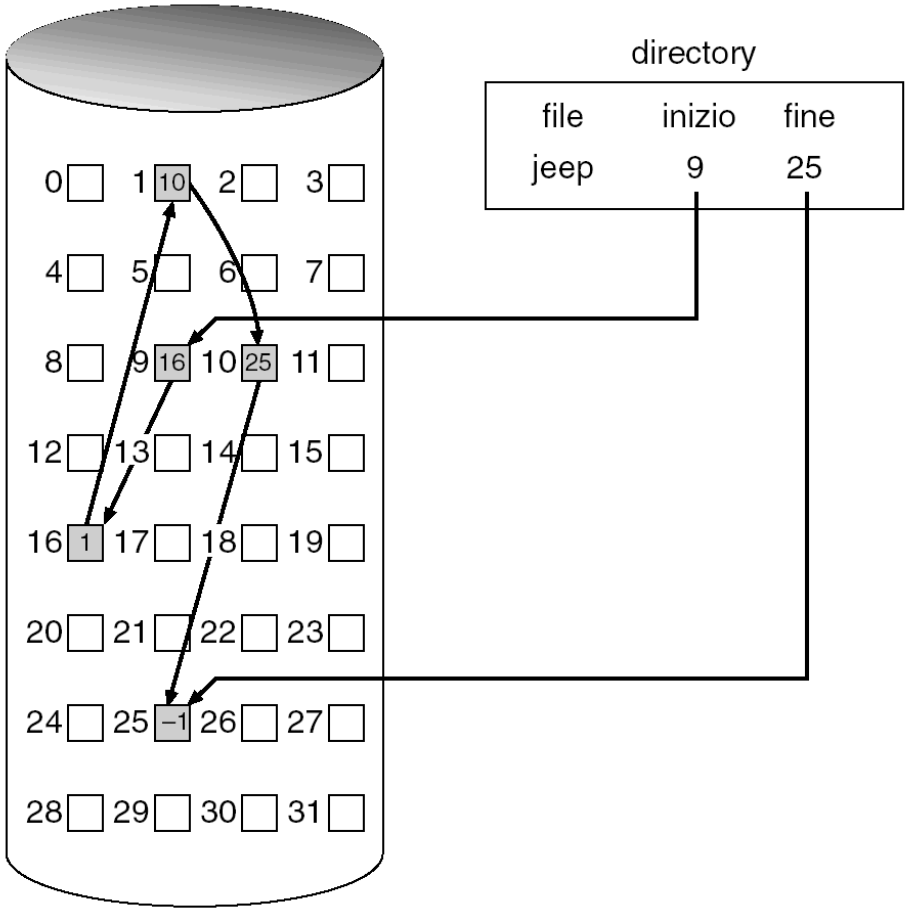
- **Pro**: l'accesso ai file può essere eseguito semplicemente sia in modo sequenziale che diretto (meno veloce)
- **Contro 1**: è necessario **trovare lo spazio libero** per allocare il file
  - Diversi algoritmi
    - First-fit: primo buco abbastanza grande
    - Best-fit: il buco che approssima meglio la dimensione (per eccesso)
  - Si presenta il problema della **frammentazione esterna**
  - Può essere risolta tramite la **compattazione**
    - Si spostano i dati su un altro disco
    - Si crea una cosi una zona contigua di spazi libero
    - Si riportano i file su disco originale allogandoli in modo contiguo

Allocazione contigua (3)

- La compattazione richiede molto tempo e può essere fatta
  - Off line: non è possibile usare il volume
  - On line: peggiora le prestazioni
- **Contro 2**: è necessario **stimare quanto spazio allocare** ad un file
  - Quando un file finisce lo spazio disponibile
    - Il processo può essere interrotto
    - Il file può essere spostato in una zona di memoria più grande
  - Allocare più spazio di quello inizialmente necessario può portare alla **frammentazione interna**
    - Specialmente nel caso di file che crescono lentamente
  - Alcuni sistemi usano l'**allocazione contigua estesa**
    - Quando un file necessita di spazio gli viene assegnata una seconda area contigua
    - Ogni area è caratterizzata da: blocco d'inizio, numero blocchi, area successiva

Allocazione concatenata (1)

- Ogni file è composto da un lista concatenata di blocchi sparsi all'interno del disco
- La directory memorizza per ogni file
  - Puntatore al primo blocco
  - Puntatore all'ultimo blocco
- Il puntatore non è visibile all'utente, quindi "ruba" spazio
  - Blocco da 512 Byte, puntatore da 4 Byte
  - Spazio utile: 508 Byte



Allocazione concatenata (2)

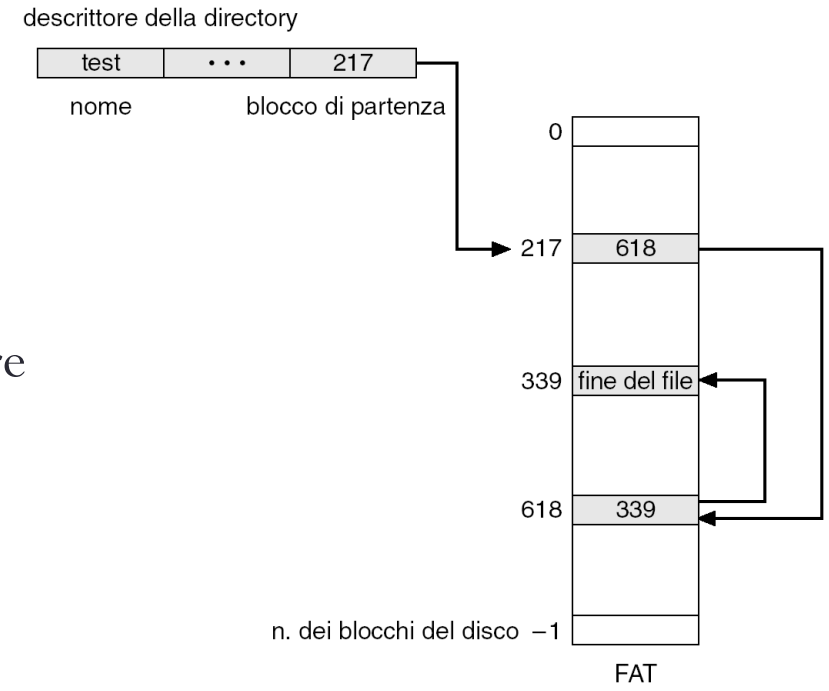
- **Creazione di un file**: si aggiunge un elemento alla directory con
  - Puntatore al primo blocco settato a null
  - Dimensione pari a 0
- **Scrittura di un file**:
  - Si cerca un blocco libero e si effettua la scrittura dei dati
  - Lo si concatena all'ultimo blocco del file (se presente)
  - Si aggiornano i puntatori della directory all'ultimo e al primo blocco (nel caso il file sia ancora vuoto)
- **Lettura di un file**: si scorrono in sequenza i blocchi seguendo la concatenazione

Allocazione concatenata (3)

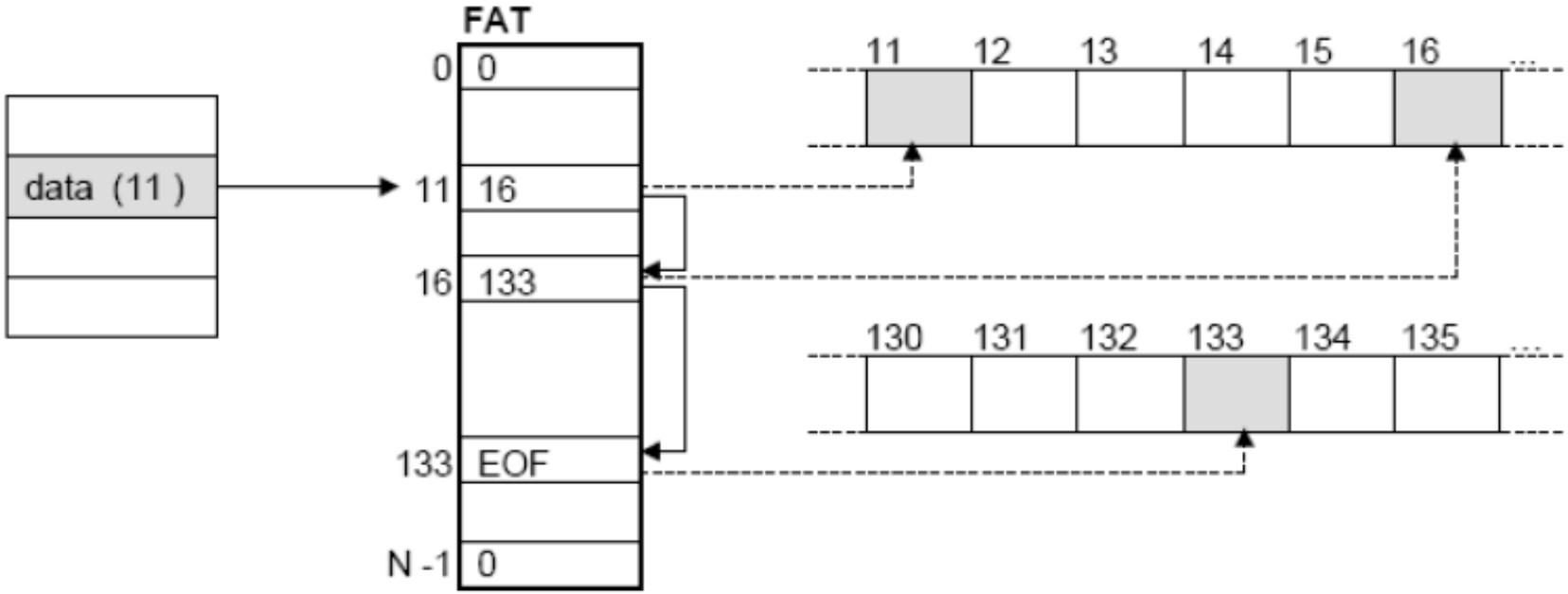
- L'allocazione concatenata risolve i problemi dell'allocazione contigua
  - Non c'è frammentazione esterna, non serve la compattazione
  - Non è necessario conoscere la dimensione del file al momento della creazione
- **Problemi**:
  - È efficiente solo per l'accesso sequenziale (a causa dei puntatori)
  - Richiede più spostamenti della testina
  - I puntatori consumano spazio
    - Una soluzione è quella di allocare cluster di blocchi anziché singoli blocchi
    - Meno spazio sprecato per i puntatori, meno spostamenti della testina
    - Aumenta però la frammentazione interna
  - Perdita o danneggiamento di un puntatore
    - Potrebbe puntare ad un blocco vuoto o un blocco di un altro file
    - Possibile soluzione: liste doppiamente concatenate

Allocazione concatenata con FAT (1)

- Una variante importante è quella della tabella di allocazione dei file (FAT)
- Nella prima parte del disco si istanzia un tabella
  - Contenente tanti elementi quanti sono i blocchi
  - Ordinata in base al numero del blocco
- La directory contiene per ogni file il puntatore al suo primo elemento nella FAT
  - Ogni elemento contiene un numero di blocco ed è concatenato al blocco successivo nel file
  - L'ultimo elemento contiene un valore end of file
  - L'elemento di un blocco che non è associato a nessun file contiene uno 0
- Seguendo i puntatori nella FAT si ricavano gli indici dei blocchi fisici di un file



Allocazione concatenata con FAT (2)



Allocazione concatenata con FAT (3)

- Per essere efficiente la FAT richiede l'uso di una cache
  - Altrimenti la testina del disco deve continuamente spostarsi tra l'inizio del disco e la locazione del blocco successivo
- Migliora le prestazioni nel caso di accesso diretto
  - La testina del disco scandisce la FAT e poi si fa uno spostamento grande per accedere al blocco voluto
- La differenza fra FAT12, FAT16 e FAT32 consiste in quanti bit sono allocati per numerare i blocchi del disco
  - Con 12 bit, il file system può indirizzare al massimo 2^12 = 4096 blocchi, mentre con 32 bit si possono gestire 2^32 = 4.294.967.296 blocchi
- L'aumento del numero di bit di indirizzo dei blocchi e la dimensione stessa del blocco sono stati resi necessari per gestire unità a disco sempre più grandi e capienti

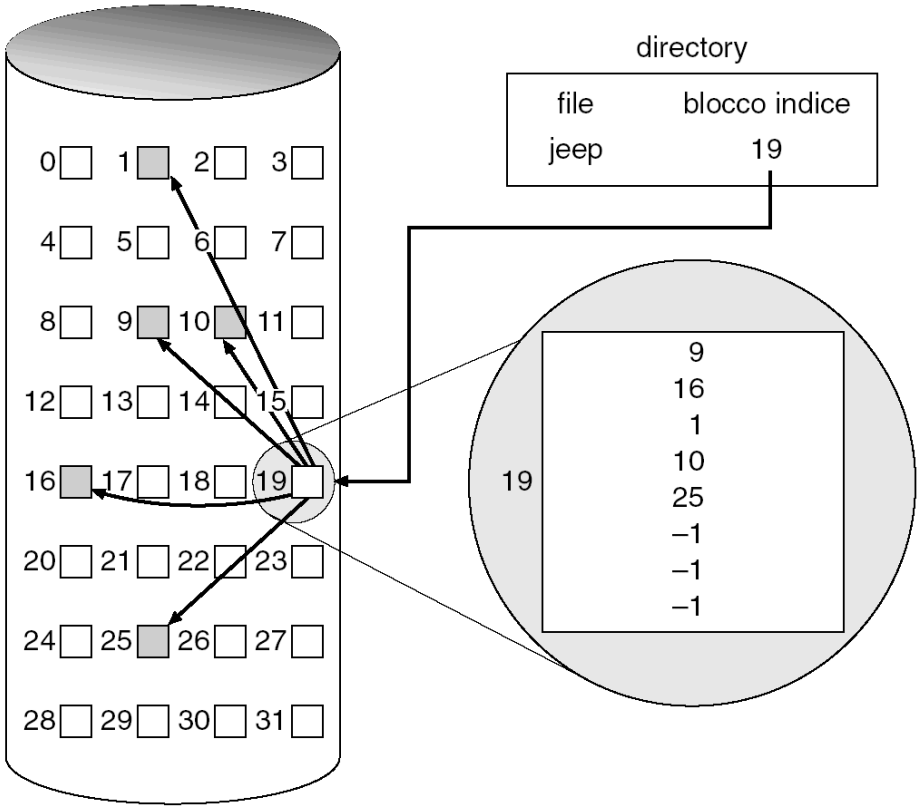
Allocazione concatenata con FAT (4)

- Il numero di bit usati per la FAT e la dimensione dei blocchi influenza la dimensione massima di una partizione

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Allocazione indicizzata (1)

- Ogni file possiede un indice dei blocchi memorizzato in un blocco indice
  - Questo blocco contiene un array di puntatori agli altri blocchi del file
  - L'i-simo elemento dell'array punta all'i-simo blocco del file
  - La directory memorizza per ogni file un puntatore al suo blocco indice
- Questo risolve il problema dell'accesso diretto presente nell'allocazione concatenata ed evita la frammentazione esterna

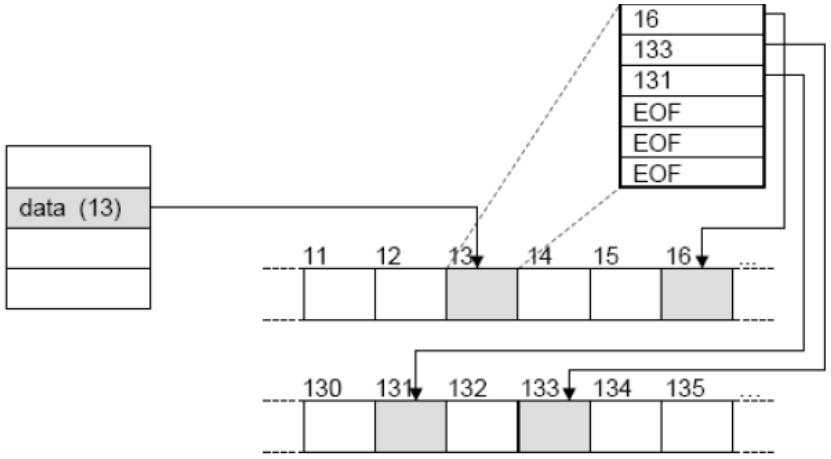


Allocazione indicizzata (2)

- **Creazione di un file**:
  - Si crea un blocco indice con tutti gli elementi settati a null
  - Si aggiunge un elemento alla directory con puntatore ad un blocco indice
- **Scrittura di un file**:
  - Si cerca un blocco libero e si effettua la scrittura dei dati
  - Si aggiorna il blocco indice
- **Lettura di un file**: si scorrono in sequenza i blocchi seguendo l'array dell'indice

Allocazione indicizzata (3)

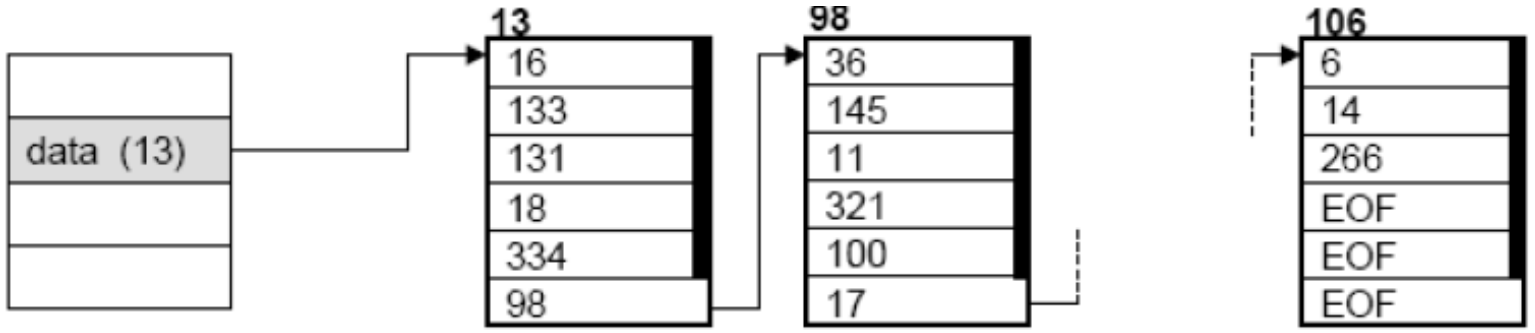
- **Problema**: l'indice richiede l'uso di un intero blocco, quindi nel caso di un file piccolo si ha frammentazione interna



- Si dovrebbero usare blocchi piccoli in modo da ridurre la frammentazione
  - Problemi con i file grandi
- **Diverse soluzioni**

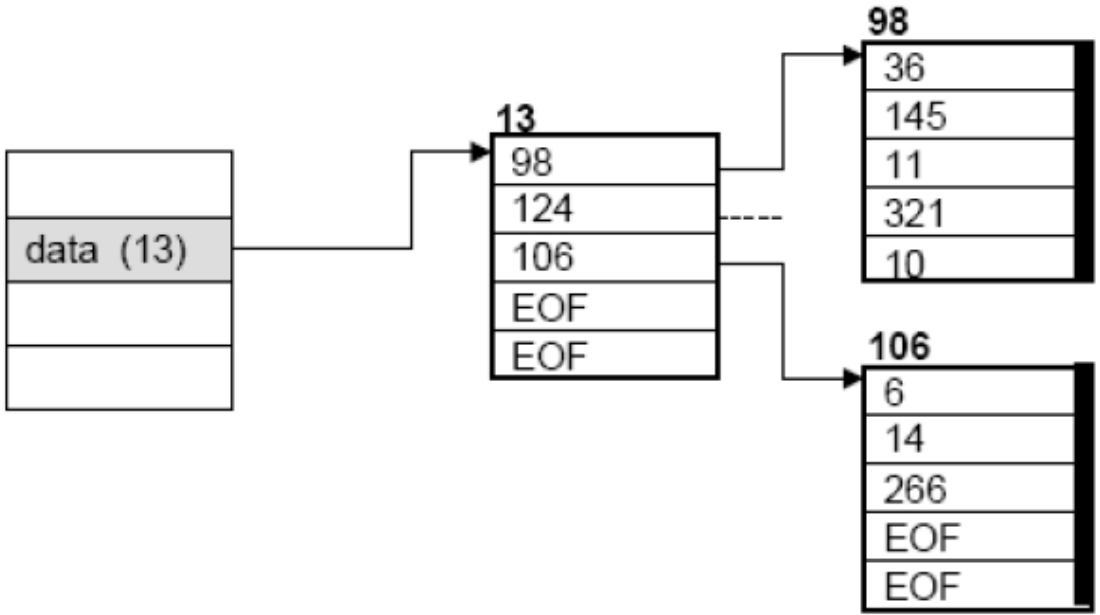
Allocazione indicizzata (4)

- Schema concatenato: si usano blocchi piccoli e nel caso di file grandi l'indice è composto da più blocchi concatenati



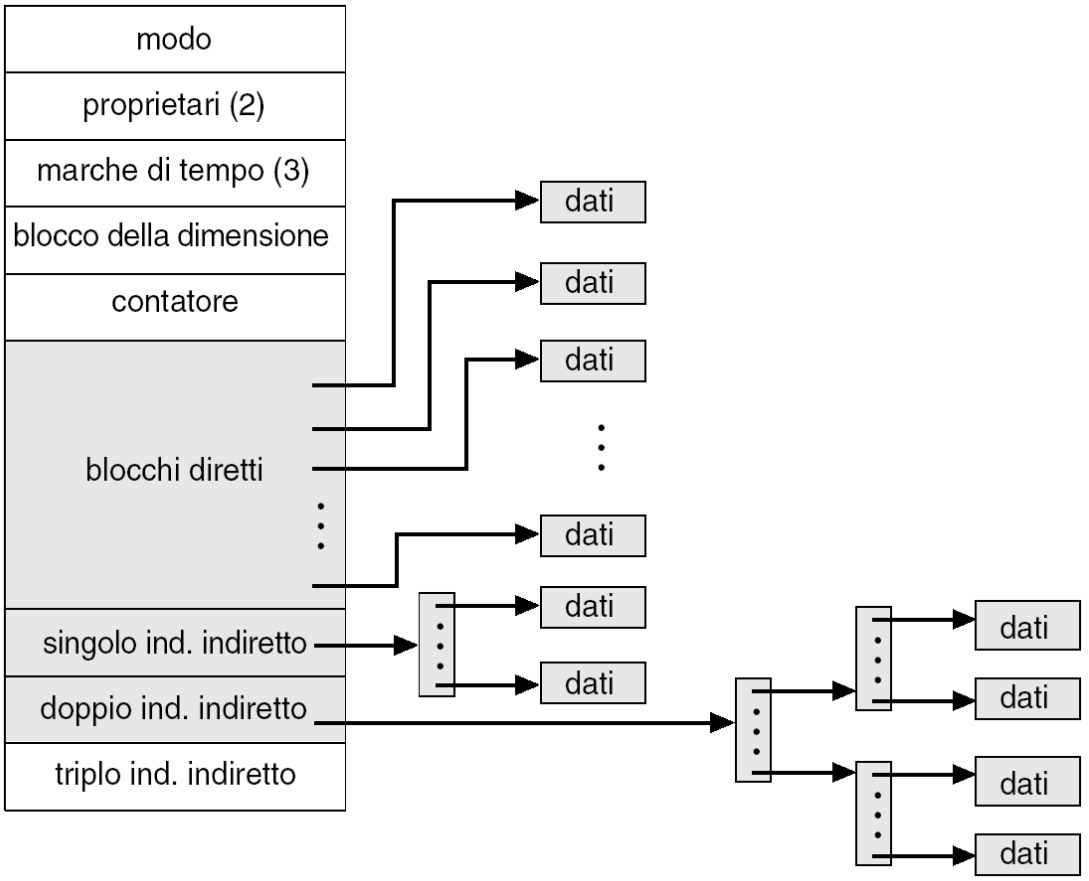
Allocazione indicizzata (5)

- Schema a più livelli: si ha un indice di primo livello che punta ad un insieme di indici di secondo livello, i quali puntano ai blocchi del file
- Con blocchi da 4 KByte e puntatori da 4 Byte si possono indicizzare file da 4GB
- È possibile estendere lo schema a 3-4 livelli



Allocazione indicizzata (6)

- Schema combinato: usato nei sistemi Unix
- Il FCB contiene 15 elementi per indicizzare i file
  - I primi 12 sono puntatore diretti a dei blocchi
  - Il 13° punta ad un blocco contenente un indice
  - Il 14° punta ad un indice a due livelli
  - Il 15° punta ad un indice a tre livelli



Scelta del metodo di allocazione (1)

- La scelta del metodo di allocazione deve considerare due fattori:
  - Efficienza di memorizzazione
  - Tempo di accesso ai dati
- La scelta è influenzata dalla modalità di accesso al file
  - L'allocazione contigua richiede un solo accesso al disco qualsiasi sia il tipo di accesso (sequenziale o diretto)
    - Anche con l'accesso diretto, noto l'indirizzo del primo blocco fisco e l'indirizzo del blocco logico desiderato, è possibile calcolare l'indirizzo fisico di accesso
  - L'allocazione concatenata invece è efficiente per l'accesso sequenziale, ma non per quello diretto

Scelta del metodo di allocazione (2)

- Per questo motivo alcuni sistemi gestiscono
  - File ad accesso diretto ➔ allocazione concatenata
  - File ad accesso sequenziale ➔ l'allocazione contigua
    - È necessario specificare a priori la dimensione del file
  - È sempre possibile convertire il formato del file
- Le prestazione dell'allocazione indicizzata invece dipendono
  - Dalla profondità dell'indice
  - Dalla dimensione del file
- Alcuni sistemi usano
  - Allocazione contigua ➔ file piccoli
  - Allocazione indicizzata ➔ file grandi

Gestione dello spazio libero (1)

- Al fine di poter trovare in modo più veloce un insieme di blocchi da allocare ad un file il sistema operativo tiene traccia dei blocchi non allocati
  - **Lista dello spazio libero**
- Creazione di un file:
  - Si cerca nella lista il numero di blocchi necessari
  - Si allocano al file
  - Si rimuovono dalla lista
- Eliminazione di un file
  - I blocchi associati al file vengono deallocati
  - Si aggiungono alla lista

Gestione dello spazio libero (2)

- Diverse possibili implementazioni:
  - Vettore di bit
  - Lista concatenata
  - Raggruppamento
  - Conteggio

Gestione dello spazio libero
Vettore di bit (1)

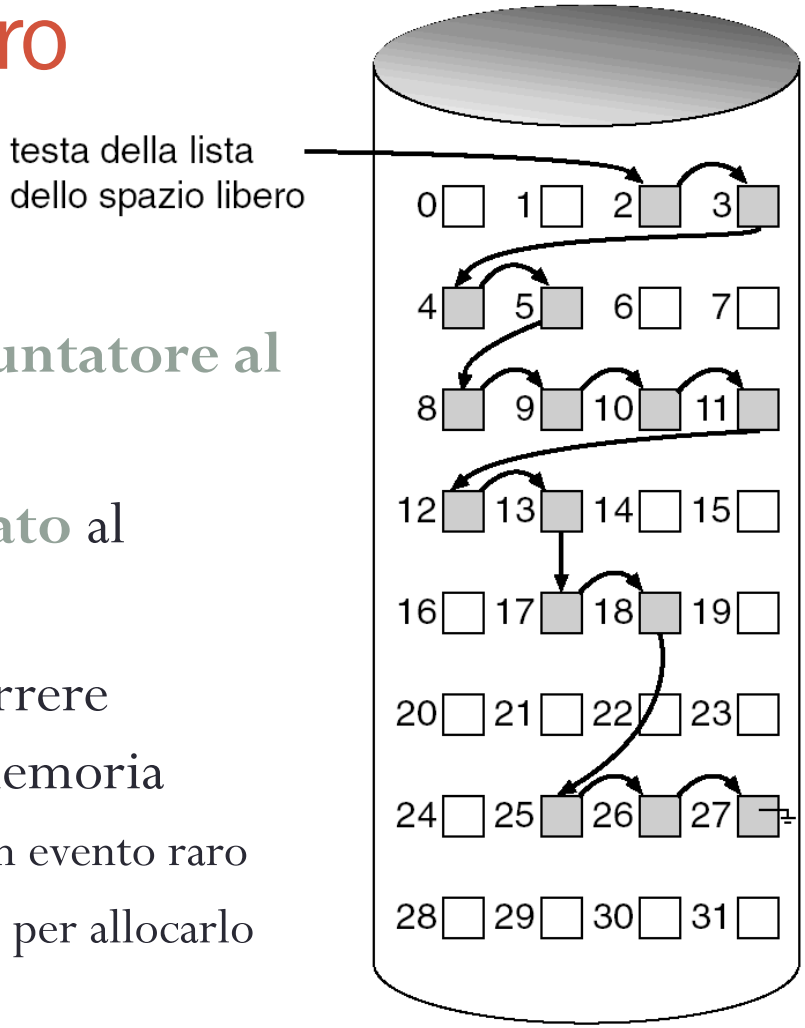
- Si memorizza un **array** di lunghezza pari al numero dei blocchi
- Il valore delle celle indica la **disponibilità** di un blocco
  - 1 libera
  - 0 occupata
- Per creare un file
  - Ricerca del primo bit a 1 (allocazione concatenata o indicizzata)
  - Ricerca di un blocco di bit a 1 sufficientemente grande (allocazione contigua)
- **Pro**: semplice ed efficiente
- **Contro**: per essere efficiente deve però essere mantenuta in memoria centrale
  - Dischi grandi richiedono vettori di bit molto grandi

Gestione dello spazio libero
Vettore di bit (2)

- I calcolatori forniscono supporto HW
  - Istruzioni per trovare lo scostamento del primo bit a 1 in una parola
  - Se l'array è diviso in parole di *n* bit:
  - `indice_blocco = (n * num_parole_0) + scostamento_primo_bit_1`
    - Num_parole_0 è il numero di parole consecutive con tutti i bit a 0 partendo dalla prima parola

101100011010

Gestione dello spazio libero
Lista concatenata



- Si memorizza in memoria centrale un **puntatore al primo blocco libero**
- Ogni blocco libero viene poi **concatenato** al successivo
- **Contro**: è poco efficiente in quanto scorrere l'intera lista richiede molti accessi alla memoria
  - Fortunatamente lo scorrimento della lista è un evento raro
  - Il SO si limita a cercare il primo blocco libero per allocarlo ad un file
  - Allocato il blocco si aggiorna il puntatore al primo blocco libero in memoria centrale

Gestione dello spazio libero
Raggruppamento e conteggio

- **Raggruppamento**
  - La memoria contiene un puntatore ad un blocco contenente gli indirizzi di n blocchi liberi
    - n-1 sono effettivamente liberi
    - L'ultimo contiene a sua volta n indirizzi di blocchi liberi

- **Conteggio**
  - Sfrutta il fatto che tipicamente si ha più di un blocco contiguo libero
  - Si usa un array in cui ogni elemento contiene
    - Indirizzo del primo blocco libero
    - Conteggio degli n blocchi contigui liberi

Efficienza e prestazioni

- I dischi sono tipicamente conosciuti come il collo di bottiglia di ogni sistema
  - A causa della loro velocità di accesso
- Gli algoritmi di allocazione dei blocchi e gestione delle directory devono essere scelti con cura al fine di ottimizzare
  - Efficienza di uso dei dischi
  - Prestazioni

Efficienza

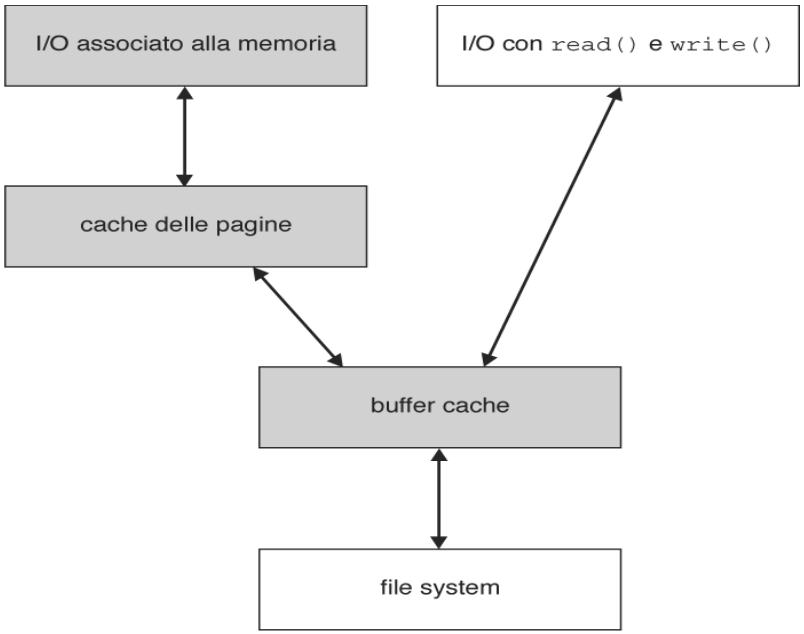
- Diverse scelte influenzano l'efficienza del disco
- **Allocazione dei FCB**: alcuni SO, tipo Unix, allocano preventivamente i FCB e li distribuiscono nel file system
  - Anche un disco senza file ha una certa quantità di spazio occupata dai FCB
  - Questa scelta può migliorare le prestazioni del file system
- **Scelta degli attributi da memorizzare per ogni file**:
  - Salvare le date di accesso ai file può essere utile per fornire informazioni all'utente
  - Tuttavia questo richiede due accessi al disco ogni volta che si legge un file
    - Uno in lettura e uno in scrittura per modificare la directory
  - Gli attributi da memorizzare vanno scelti con cura
- **Lunghezza dei puntatori**:
  - Più bit si usano per i puntatori più grandi possono essere i file (32 ➔ bit 2^{32}=4GB)
  - Tuttavia puntatori grandi richiedono più spazio per eseguire gli algoritmi di allocazione e gestione dello spazio libero (è spazio non usabile dall'utente)

Prestazioni (1)

- Una volta scelti gli algoritmi per la gestione del file system si possono adottare diverse tecniche per migliorare le prestazioni
- **Cache del disco (buffer cache)**: area riservata della memoria centrale dove vengono mantenuti i blocchi usati di recente
  - Probabilmente verranno riusati a breve
- **Cache delle pagine**: si usano tecniche di memoria virtuale non solo per la gestione dei processi ma anche per la gestione dei file
- **Buffer cache unificata**: è utile per gestire in modo efficiente l'accesso ai file
  - Sia tramite le chiamate di sistema `read()` e `write()`
  - Che tramite la mappatura dei file in memoria

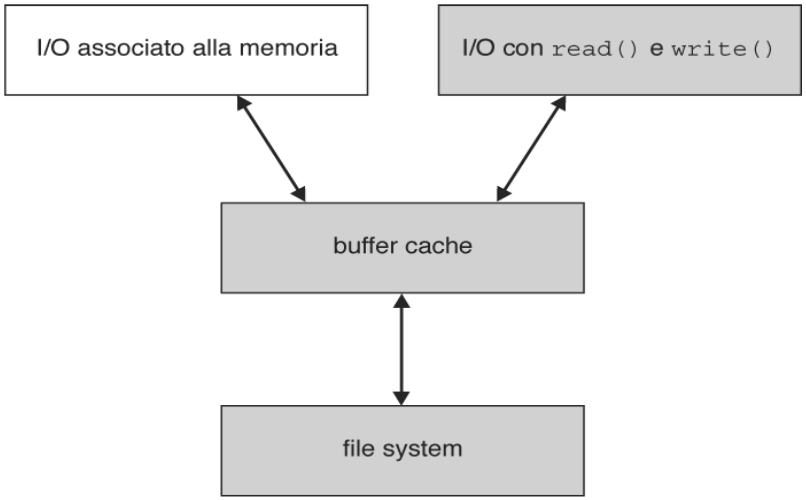
Prestazioni (2)

- Se non si usa la cache unificata
  - Le chiamata di sistema usano direttamente la cache buffer cache
  - Il sistema di memoria virtuale non può interfacciarsi direttamente con la buffer cache
    - È necessario copiare nella cache delle pagine il contenuto del file presente nella buffer cache
  - Fenomeno del **double caching**
    - Doppio passaggio di cache
  - Spreco di memoria, cicli di CPU e di I/O



Prestazioni (3)

- Con la cache unificata il problema del double caching viene risolto
- Il sistema di memoria virtuale può interfacciarsi **direttamente** con la cache del file system



Prestazioni (4)

- **Scritture sincrone e asincrone**:
  - Influenzano le prestazioni di I/O
  - Scrittura sincrona: le scritture su disco avvengono nell'ordine in cui il driver riceve le richieste
    - Il chiamante è bloccato fino a quando il dato non viene scritto
  - Scritture asincrone: le scritture avvengono sulla cache del disco
    - Il driver del dispositivo si occupa poi di trasferire i dati sul disco
    - Il chiamante non resta bloccato
  - Le scritture asincrone sono usate più frequentemente

Prestazioni (5)

- **Algoritmo di sostituzione della pagine**:
  - Il metodo di accesso al file influenza la scelta dell'algoritmo di sostituzione delle pagine
  - In caso di accesso sequenziale LRU non è ottimale
  - La pagina usata più recentemente è quella che sarà usata più in la nel tempo
  - **Rilascio indietro**:
    - Si rimuove la pagina dalla memoria quando viene fatto un riferimento alla pagina successiva
  - **Lettura anticipata**:
    - Si copiano nella cache la pagina richiesta ed n pagine successive
  - Notevoli risparmi di tempo

Ripristino del file system

- Operazioni eseguite molto frequentemente, come la creazione di un file, comportano **molti cambiamenti nelle strutture dati** del file system
  - Struttura della directory
  - Lista dello spazio libero
  - FCB
- Se un crollo del sistema avviene prima che **tutte** queste operazioni siano completate possono crearsi delle **incoerenze**
- E.g. un nuovo FCB potrebbe essere stato allocato ma la struttura delle directory potrebbe non contenere il puntatore ad esso
- Il SO adotta diverse tecniche per risolvere le incoerenze

Verifica della coerenza

- Per scoprire eventuali errori è necessario analizzare i meta-dati
- Questo richiede molto tempo
- Prima di modificare i meta-dati il SO mette un bit di modifica a 1
  - Se le modifiche terminano con successo il bit viene rimesso a 0
  - Se resta ad un 1 significa che c'è stato un crollo e al riavvio viene eseguito il **verificatore della coerenza**
- Confronta i meta-dati con i blocchi del disco per correggere eventuali errori
  - Con l'allocazione concatenata i puntatori permettono di ricostruire un file
    - Si può ricreare l'elemento della directory
  - Con l'allocazione ad indicizzazione la perdita dell'indice è grave
    - I blocchi infatti non contengono informazioni sugli altri blocchi del file

File system con registrazione delle modifiche (1)

- La verifica della coerenza comporta alcuni problemi
  - Non sempre le incoerenze sono risolvibili
  - In alcuni casi la risoluzione richiede l'intervento umano
  - Richiede molto tempo
- Si adottano quindi degli algoritmi usati nelle basi di dati
  - Si mantiene un log dove vengono salvate in modo **sequenziale** le modifiche ai meta-dati
  - Quando si opera su un file, l'operazione è considerata **committed** quando
    - Tutte le modifiche da apportare ai meta-dati sono state salvate nel log (transazione)
  - La chiamata di sistema restituisce il controllo all'utente
  - E il SO si occupa di aggiornare in modo **asincrono** i meta-dati come indicato dal log
  - Quando tutti i meta-dati sono stati aggiornati le modifiche vengono tolte dal log

File system con registrazione delle modifiche (2)

- Se il sistema crolla al momento del riavvio
  - Si analizza il log e si eseguono tutte le transazioni registrate
  - In quanto queste modifiche non erano ancora state apportate ai meta-dati

- Il log è tipicamente salvato in un'area dedicata del disco
  - Quindi accedere in modo **sequenziale e sincrono** al log per salvare le transazioni
  - È più veloce che accedere in modo **diretto e sincrono** ai blocchi contenenti i meta-dati
- Questa tecnica perciò **diminuisce il tempo** in cui un processo resta bloccato in attesa dell'aggiornamento dei meta-dati

Altre soluzioni

- Un alternativa consiste nel non sovrascrivere i meta-dati
- Quando è necessario aggiornarli
  - Si scrivono i nuovi valori in un nuovo blocco del disco
  - Si aggiorna il puntatore ai meta-dati
  - Si dealloca il blocco contenente i vecchi meta-dati

Copie di riserva e recupero dei dati

- Le tecniche viste finora permettono di recuperare eventuali inconsistenze fra meta-dati e file
- Esistono anche delle tecniche che permettono di recuperare i file in caso di guasti ai dischi (**backup e ripristino**)
- Il backup può essere
  - **Assoluto**: ogni volta si copia l'intero disco
  - **Incrementale**: la prima volta si copia l'intero disco, le volte successive solo i file modificati
    - Si basa sui valori degli attributi
    - Se data_ultima_modifica > data_backup ➔ nuovo backup del file
- Il metodo di ripristino varia in base al tipo di backup