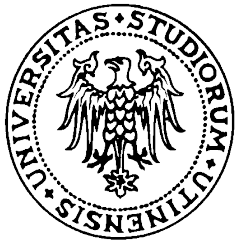
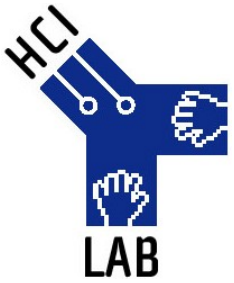
Realizzazione del file system



Fabio Buttussi

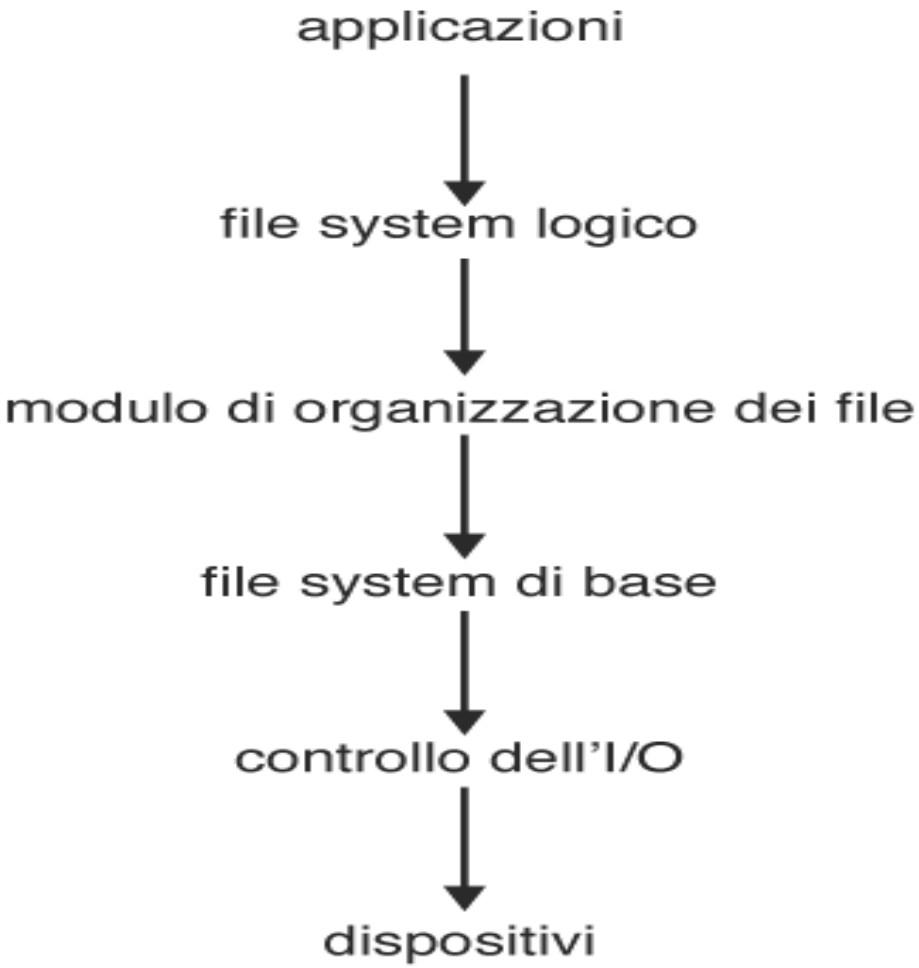
HCI Lab
Dipart. Di Matematica ed Informatica
Università degli studi di Udine

www.dimi.uniud.it/buttussi



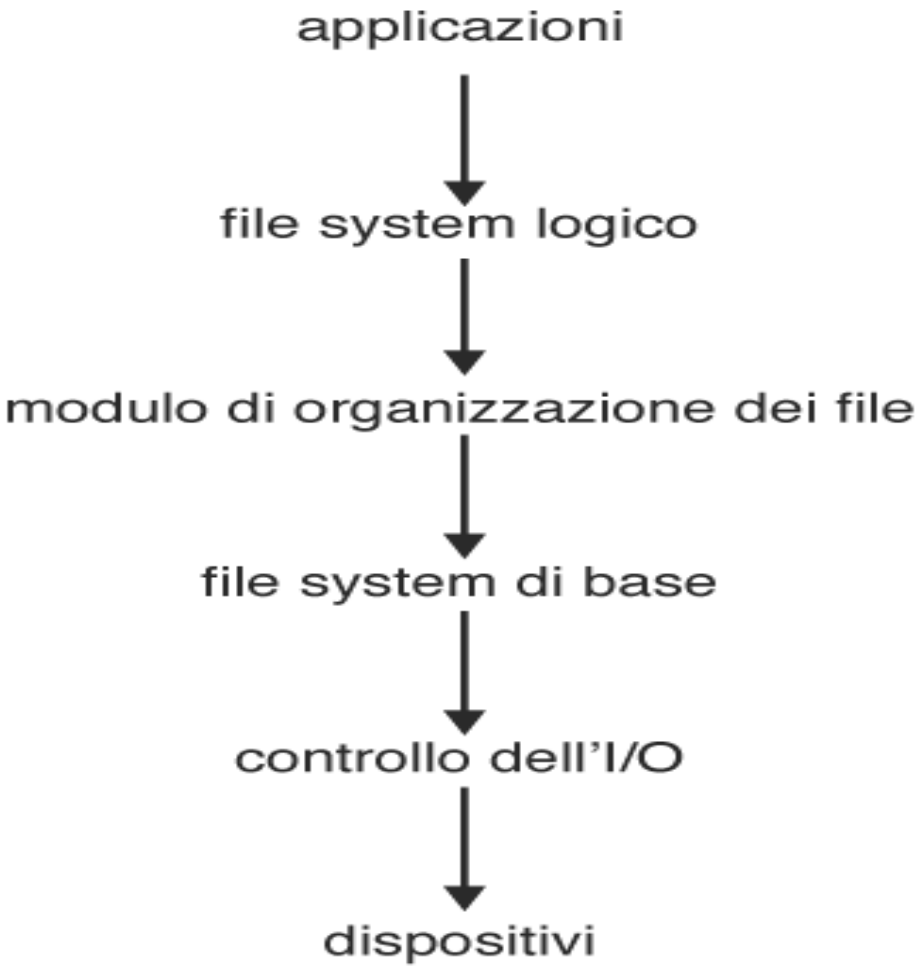
Struttura del file system

- Il file system risiede in **memoria secondaria**
- E' tipicamente organizzato in **livelli**
- Ogni livello utilizza le funzioni dei livelli inferiori per implementare funzioni per i livelli superiori



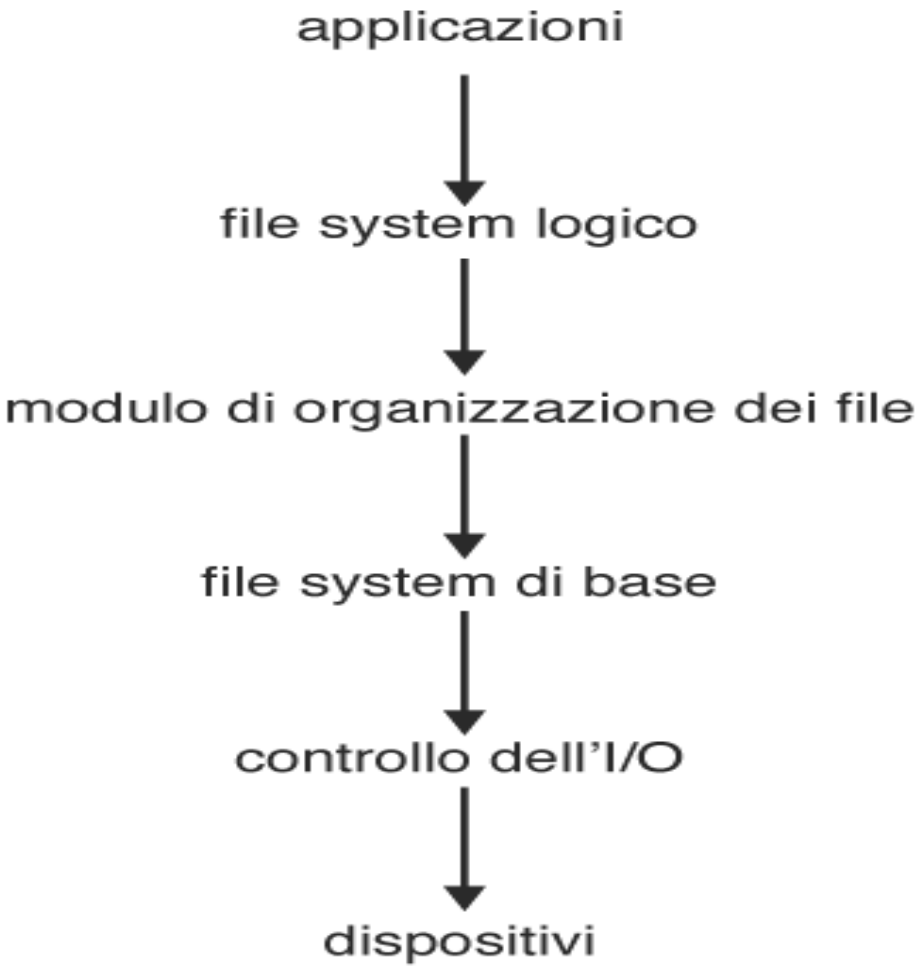
Struttura del file system

- Il **controllo dell'I/O** (es: driver dei dispositivi) si occupa del trasferimento delle informazioni tra **memoria centrale** e **memoria secondaria**
- Il **file system di base** invia comandi ai driver dei dispositivi per **leggere e scrivere blocchi fisici**



Struttura del file system

- Il **modulo di organizzazione dei file** traduce gli **indirizzi** dei blocchi **logici** in indirizzi **fisici** e comprende il **gestore dello spazio libero**
- Il **file system logico** gestisce tutte le **strutture** del file system ad eccezione del contenuto dei file

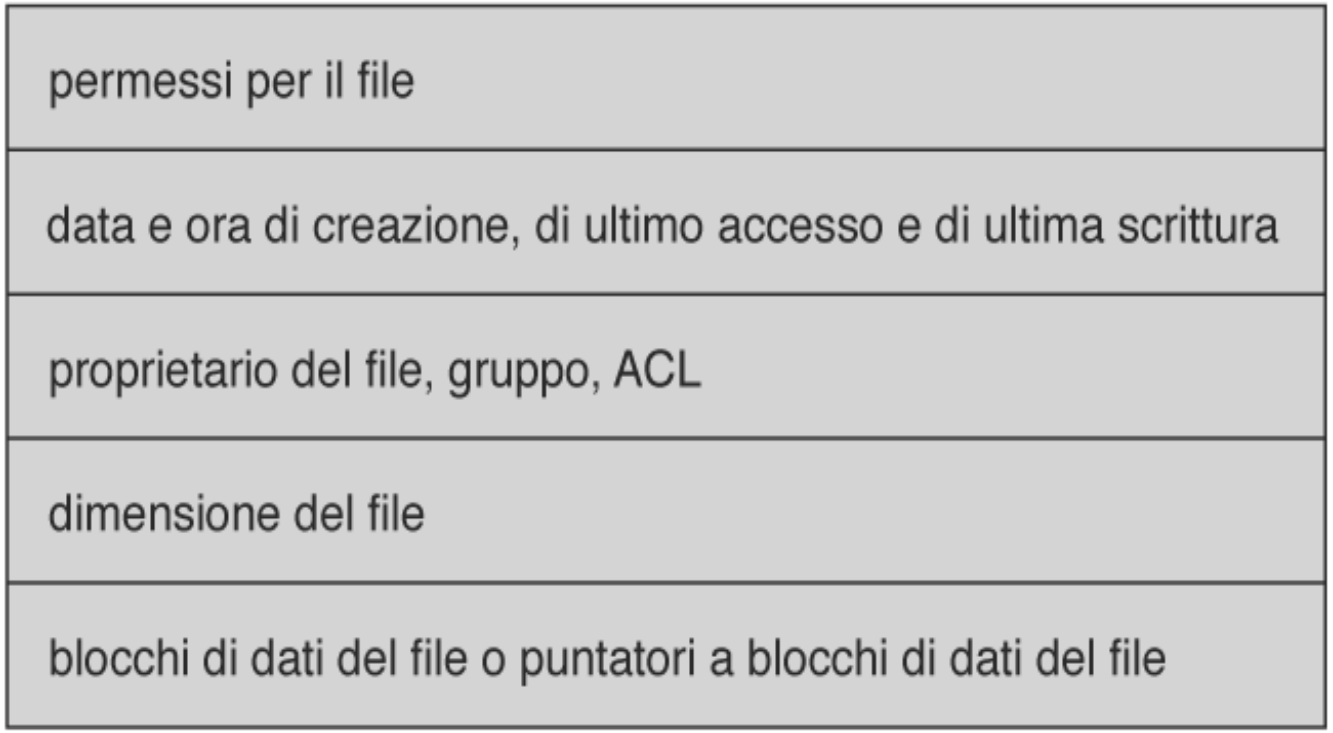


Strutture dati su disco

- **Blocchi di controllo dei file**: dettagli sui file; nell'UFS si chiamano ***inode***; nell'NTFS sono memorizzati nella tabella principale dei file
- **Strutture delle directory**: nel caso di NTFS sono memorizzate nella tabella principale dei file; in UFS comprendono nomi dei file e numeri di inode associati
- **Blocchi di controllo dei volumi**: contengono i dettagli di ogni volume (numero e dimensione dei blocchi, contatore blocchi liberi e puntatori); superblocco in UFS, tabella principale dei file in NTFS
- **Blocco di controllo dell'avviamento**: contiene le informazioni necessarie al sistema per avviare un sistema operativo da quel volume

Blocco di controllo dei file

- Il file system logico mantiene le strutture dei file tramite i blocchi di controllo dei file (**FCB**), contenenti informazioni sui file



Strutture dati in memoria

- Servono per la **gestione del file system** e per migliorare le **prestazioni**
- I dati si caricano al momento del **montaggio** e si eliminano allo smontaggio
- Comprendono:
  - Tabella di montaggio
  - Struttura della directory
  - Tabella generale dei file aperti
  - Tabella dei file aperti per processo

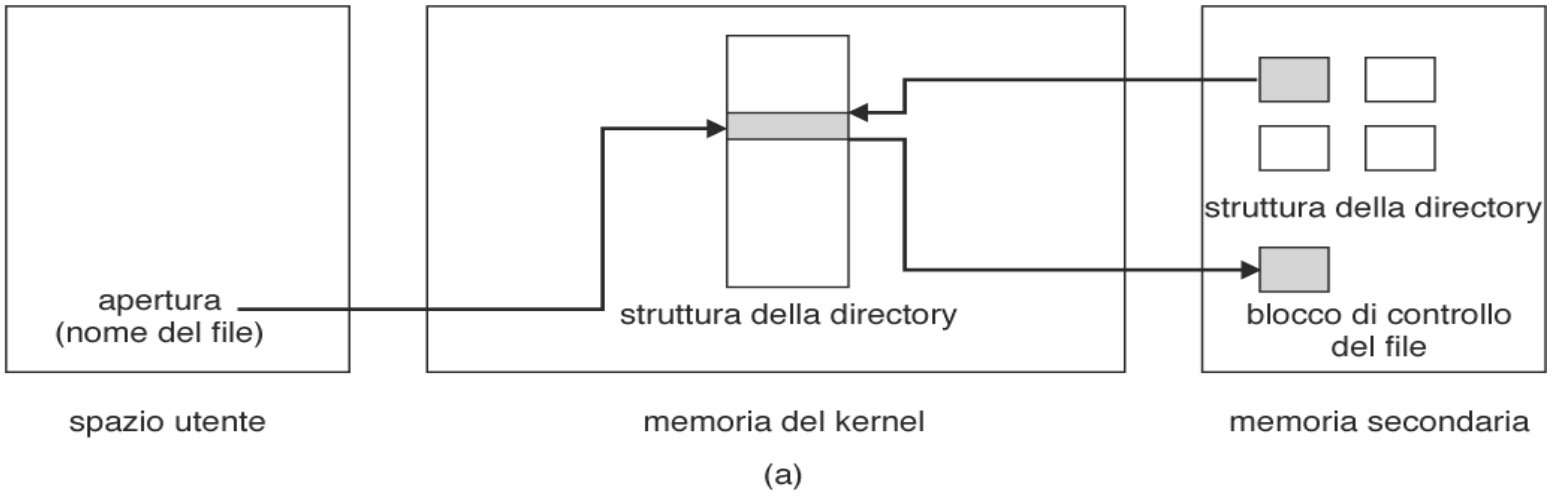
Creazione di un file

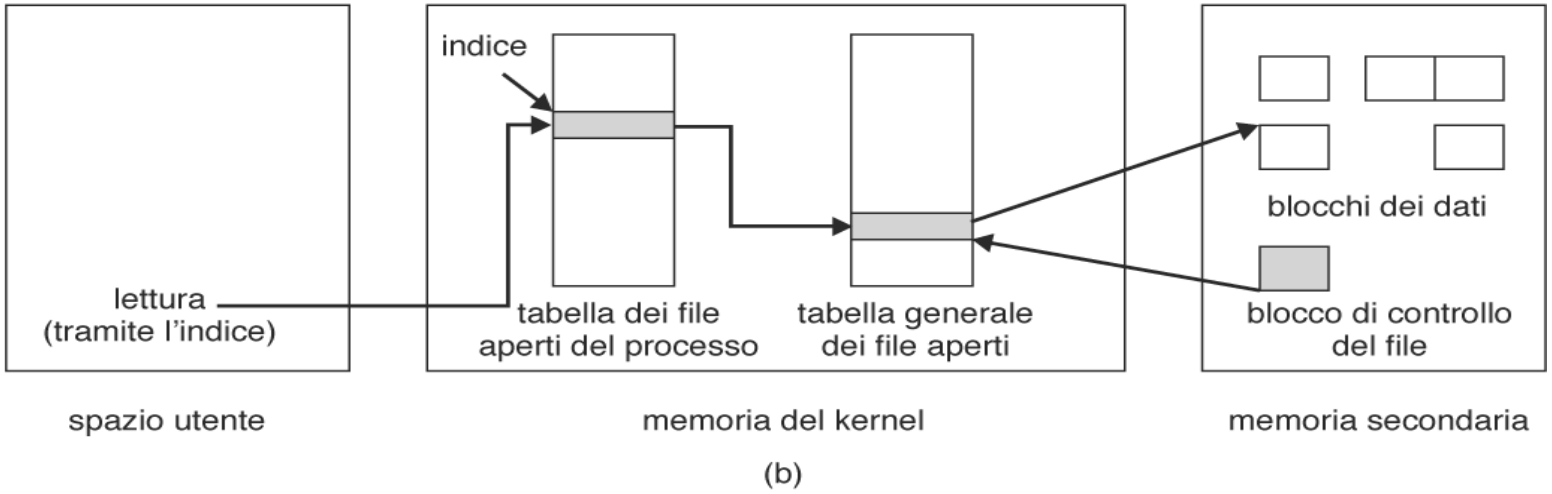
1. Chiamata al **file system logico** (conosce il formato della struttura di directory)
2. File system logico **crea** e **alloca** (o solo alloca) **nuovo FCB**
3. **Caricamento** della **directory** appropriata in **memoria**
4. **Aggiornamento** della **directory** con **nuovo nome di file e FCB** associato
5. **Scrittura** della **directory** su **disco**

Apertura di un file

1. Chiamata **open()** con nome del file come parametro
2. **Esame** della **tabella generale dei file aperti**; se file gia' aperto, **aggiunta** di un elemento alla **tabella dei file aperti del processo**, altrimenti
3. **Ricerca** del nome del file all'interno della directory
4. **Copia dell'FCB** nella tabella generale dei file aperti
5. Creazione di un elemento nella tabella dei file aperti del processo (**puntatore corrente, modalità d'accesso**)
6. L'elemento della tabella dei file aperti (il cui nome viene detto ***descrittore di file*** o *maniglia del file*) viene utilizzato per le successive operazioni

Strutture del file system in memoria



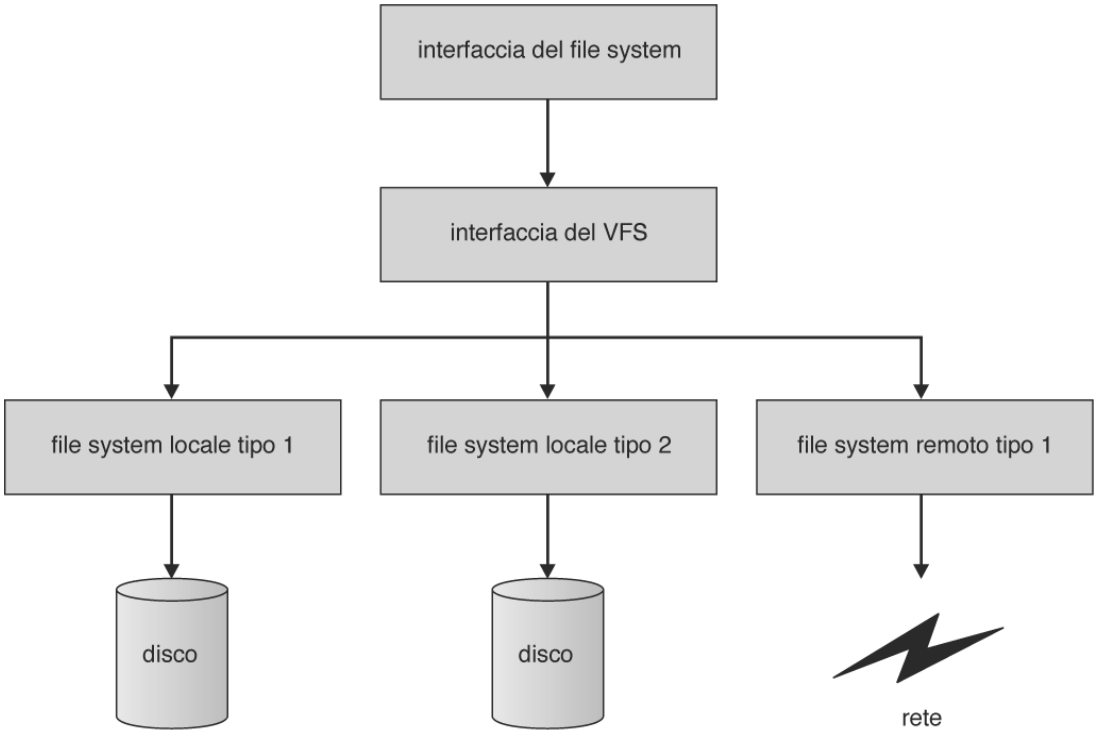


Montaggio del file system

- **Partizione d'avvio** contiene un **boot loader** per il caricamento del **SO**
- All'avvio, viene montata la **partizione radice** (kernel del SO e file di sistema)
- **Montaggio degli altri volumi** automatico (Windows) o esplicito (Unix)
- **Verifica** del file system montato e **aggiornamento** della tabella di montaggio
- Unix: flag nell'inode delle directory di montaggio, puntatore nell'inode alla tabella di montaggio, puntatore nella tabella di montaggio al superblocco del file system montato

File system virtuali

- I file system virtuali (VFS) forniscono una modalita' object-oriented per implementare i file system, permettendo l'utilizzo di **file system differenti nello stesso sistema**



File system virtuali

- L'**interfaccia del file system** e' basata sulle chiamate di sistema open(), read(), write(), close() e sui descrittori dei file

- L'interfaccia del file system virtuale **separa le operazioni generiche dalla loro realizzazione** e permette una rappresentazione univoca di un file in rete (basata su ***vnode***)

- Il VFS attiva le **operazioni specifiche dei file system** per gestire le richieste locali e invoca le procedure dei protocolli di rete per le richieste remote

Realizzazione delle directory

- **Lista lineare** contenente i **nomi** dei file con **puntatori** ai blocchi di dati
  - Semplice da implementare
  - La **ricerca** lineare dei file e' **costosa**
  - Si puo' ovviare usando **cache**, **liste ordinate** (permettono ricerca binaria) o **alberi**

- **Tabella hash**
  - Riduce il tempo di ricerca
  - Problema delle **collisioni** – due nomi sono associati alla stessa locazione
  - Dimensione fissa (ridimensionamento, concatenazione)

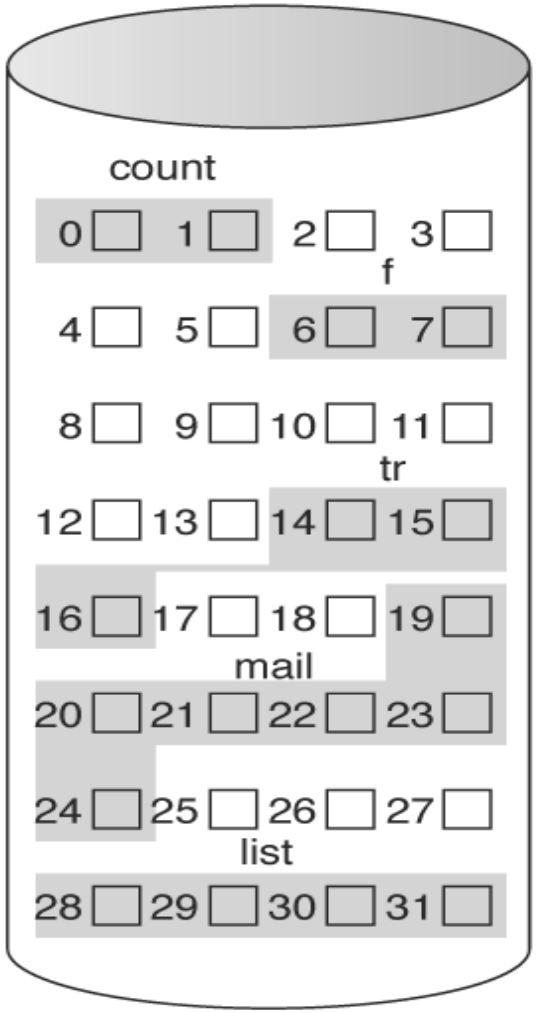
Metodi di allocazione

- Determinano come i **blocchi** su disco vengono **allocati** ai singoli **file**
- Devono garantire un **utilizzo efficiente dello spazio su disco** ed un **rapido accesso** ai file
- Metodi principali:
  - Allocazione **contigua**
  - Allocazione **concatenata**
  - Allocazione **indicizzata**
- Alcuni sistemi utilizzano contemporaneamente più di un metodo

Allocazione contigua

- **Ogni file** occupa un **insieme di blocchi contigui** su disco
- **Buone prestazioni** (numero di seek e tempo di seek bassi)
- **Semplice** – sono richiesti solo la locazione iniziale (numero di blocco) e la lunghezza (numero di blocchi)
- Permette sia accesso **sequenziale** che **diretto**
- Problema dell'**allocazione dinamica**; **frammentazione**
- In generale, i **file non possono crescere di dimensione**

Allocazione contigua



directory

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

Sistemi basati sulle estensioni

- Alcuni **nuovi file system** (ad esempio, il Veritas File System) usano un meccanismo di **allocazione contigua modificato**

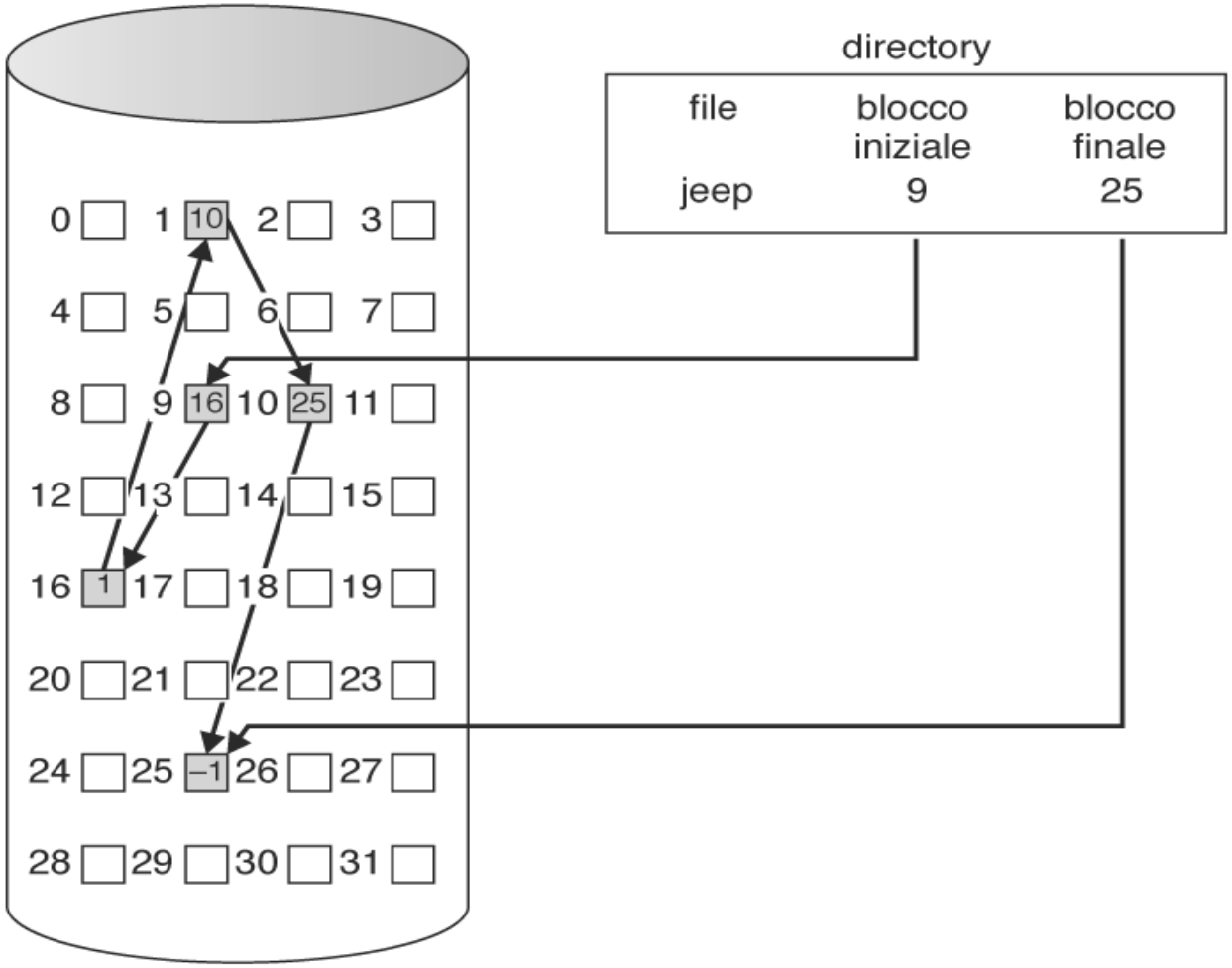
- Porzioni di spazio contiguo chiamate **estensioni** vengono aggiunte se necessario

- Per tenere traccia dei blocchi occupati, si registrano **locazione iniziale**, **numero di blocchi** e indirizzo del **primo blocco dell'estensione seguente**

Allocazione concatenata

- Risolve i problemi dell'allocazione contigua
- Ogni file e' composto da una **lista concatenata di blocchi sparsi su disco**
- La directory contiene un **puntatore al primo e all'ultimo blocco del file**
- **Ogni blocco** contiene un **puntatore al blocco successivo** (lo spazio per i dati è quindi inferiore alla dimensione del blocco)
- La lettura di un file si basa sul leggere i blocchi nell'ordine definito dai puntatori

Esempio di allocazione concatenata



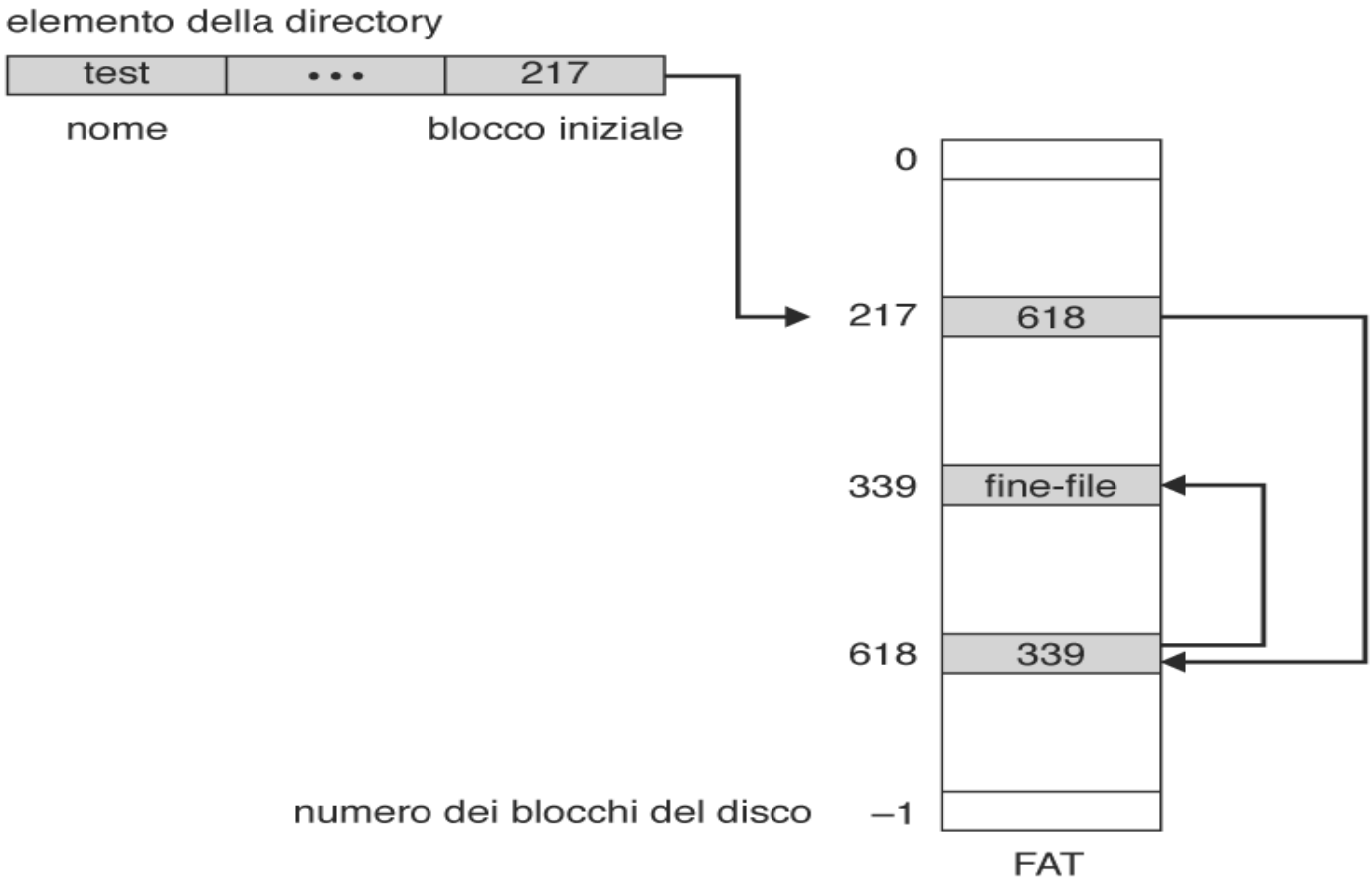
Allocazione concatenata

- La **creazione** di un **file** implica la creazione di un nuovo **elemento di directory con puntatori NIL**
- Una **scrittura** determina la **ricerca** di un blocco libero attraverso il sistema di gestione dello spazio libero e la **concatenazione** del blocco alla fine del file
- **Non esiste frammentazione esterna** e la **dimensione** dei file **non** deve essere **predefinita**
- È **efficiente** solo per l'**accesso sequenziale**; l'**accesso diretto** al blocco *i* implica il **passaggio** per i blocchi **intermedi**
- Per **sprecare meno spazio** a causa dei puntatori si possono riunire i blocchi in **cluster**
- **Problema di affidabilità** dovuto a perdita dei puntatori o utilizzo di puntatori sbagliati

Tabella di allocazione dei file (FAT)

- È una **variante** del metodo di **allocazione concatenata**, usata in MS-DOS e OS/2
- Una **sezione di disco** all'inizio di **ciascun volume** viene riservata per la FAT
- Contiene un **elemento per ogni blocco del disco** ed è **indicizzata dal numero di blocco**
- L'**elemento di directory** contiene il numero del **primo blocco** del file e l'**elemento** della **FAT** indicizzato da tale numero contiene il **numero del blocco successivo** del file e così via
- Senza cache, causa un **aumento** del numero di **spostamenti** della **testina**
- **Ottimizza** il tempo di **accesso diretto**

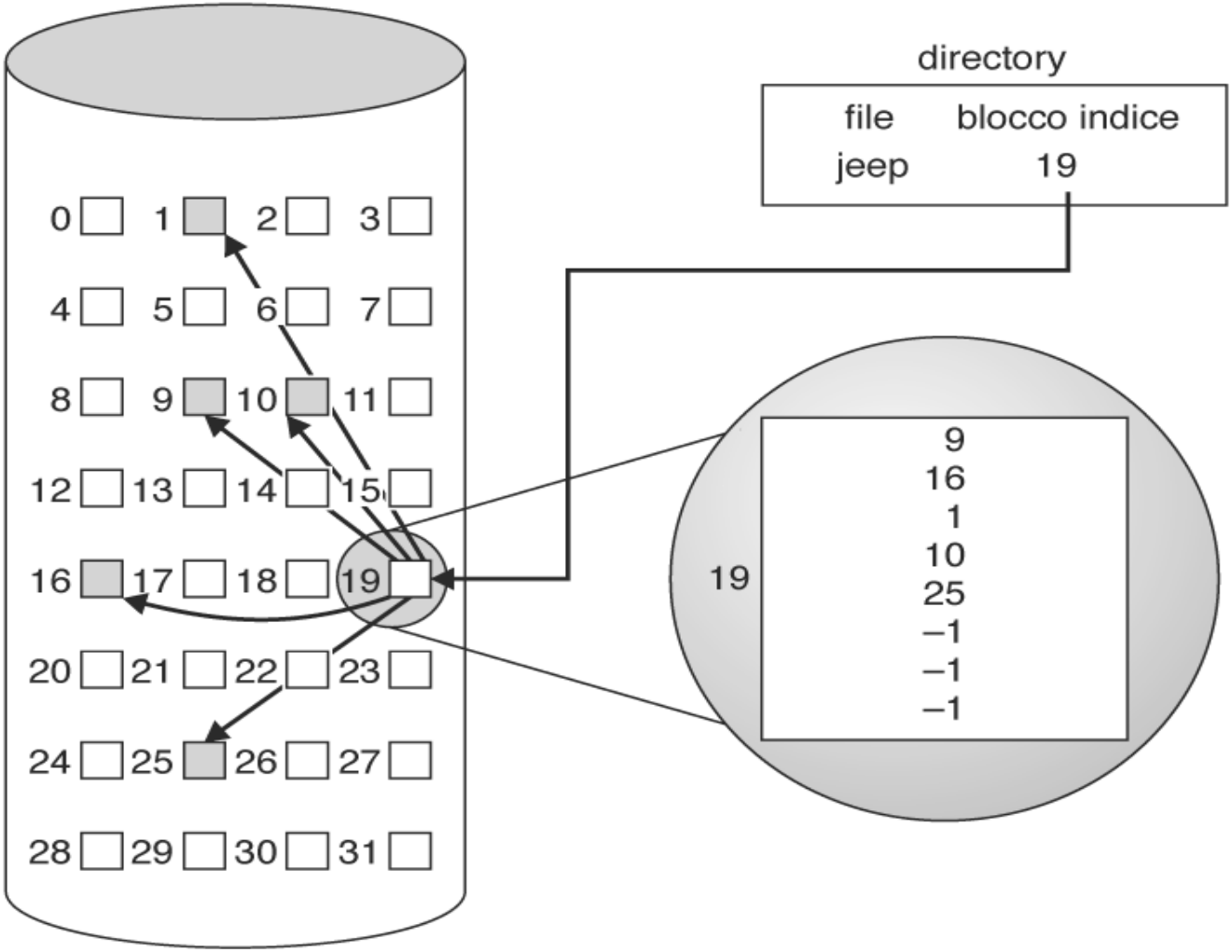
Esempio di tabella di allocazione dei file



Allocazione indicizzata

- L'allocazione indicizzata **risolve** il problema dell'**accesso diretto** dell'allocazione concatenata
- Tutti i puntatori vengono raggruppati in un **blocco indice**
- Ogni file ha il proprio blocco indice, il cui i-esimo elemento punta all'i-esimo blocco del file
- La **directory** contiene l'indirizzo del **blocco indice**
- Per leggere l'i-esimo blocco basta usare il puntatore che si trova nell'i-esimo elemento del blocco indice

Esempio di allocazione indicizzata



Allocazione indicizzata

- La **creazione** di un **file** implica la creazione di un **blocco** indice con **puntatori NIL**
- La **scrittura** dell'i-esimo blocco determina la **ricerca** di un **blocco libero** attraverso il sistema di gestione dello spazio libero e l'**inserimento** dell'indirizzo di tale blocco nell'i-esimo elemento del **blocco indice**
- **Non** esiste **frammentazione esterna** e la **dimensione** dei file **non** deve essere **predefinita**
- Si **spreca** più **spazio** per i **puntatori** rispetto all'allocazione concatenata
- Serve un **meccanismo** per **gestire** la **dimensione** del blocco **indice**

Schemi di gestione del blocco indice

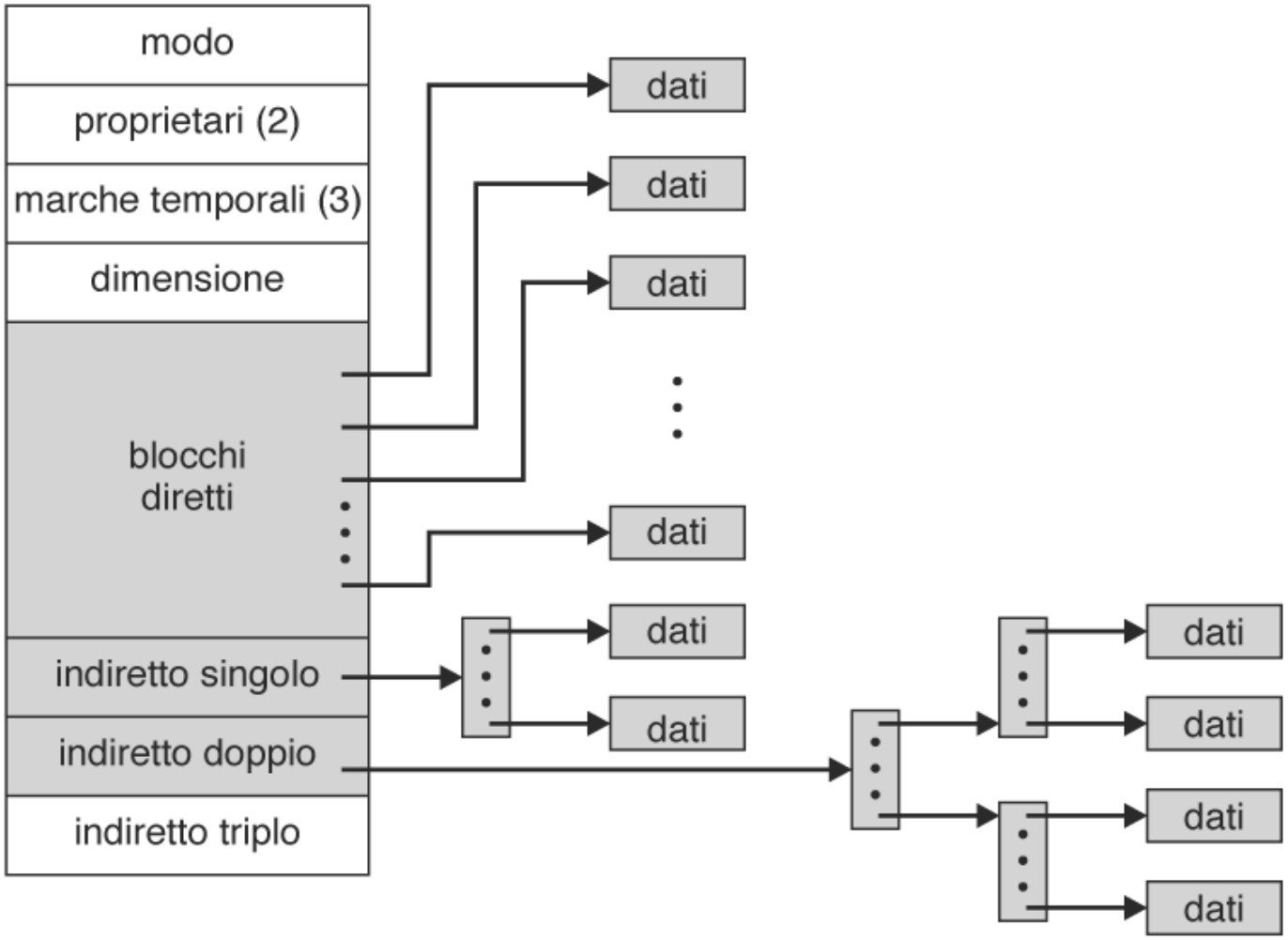
- Schema **concatenato**:
  - più blocchi indice vengono collegati tra loro secondo le necessità
  - l'ultimo elemento di un blocco indice punta al successivo

- Indice a più **livelli**:
  - un blocco indice di primo livello punta a blocchi indice di secondo livello che puntano ai blocchi dati
  - si può estendere ad ulteriori livelli

Schemi di gestione del blocco indice

- Schema **combinato**:
  - soluzione adottata nell'UFS
  - i **primi 15 puntatori** del blocco indice vengono tenuti nell'**inode** del file
  - i **primi 12** puntatori puntano a **blocchi dati**
  - i **rimanenti 3** puntatori puntano a **blocchi indiretti**:
    - il primo puntatore indiretto è l'indirizzo di un blocco indiretto singolo contenente indirizzi di blocchi dati
    - il secondo è un puntatore di blocco indiretto doppio che punta ad un blocco contenente puntatori a blocchi che puntano a dati
    - il terzo è un puntatore di blocco indiretto triplo

inode di Unix



Prestazioni

- Il metodo d'allocazione da utilizzare dipende dal **tipo di accesso** ai file impiegato
- L'allocazione **contigua** richiede **un solo accesso** per ottenere un blocco indipendentemente dal tipo d'accesso
- L'allocazione **concatenata** richiede **i accessi** per la lettura del blocco i
  - **non adatta ad accesso diretto**
- Alcuni sistemi gestiscono file ad accesso diretto usando l'allocazione contigua e file ad accesso sequenziale usando l'allocazione concatenata
- Le **prestazioni** dell'allocazione **indicizzata** dipendono dalla **struttura** dell'indice, dalla **dimensione** del file e dalla **posizione** del blocco desiderato

Gestione dello spazio libero

- Lo spazio libero su disco è composto da tutti i **blocchi non allocati** a file o directory
- Per tenere traccia dello spazio libero, si utilizza una ***lista dello spazio libero***
- I blocchi liberi vengono allocati ai file quando necessario (creazione o aggiunte) e inseriti nella lista al momento della cancellazione
- Implementazione tramite
  - **vettore di bit**
  - **lista concatenata**

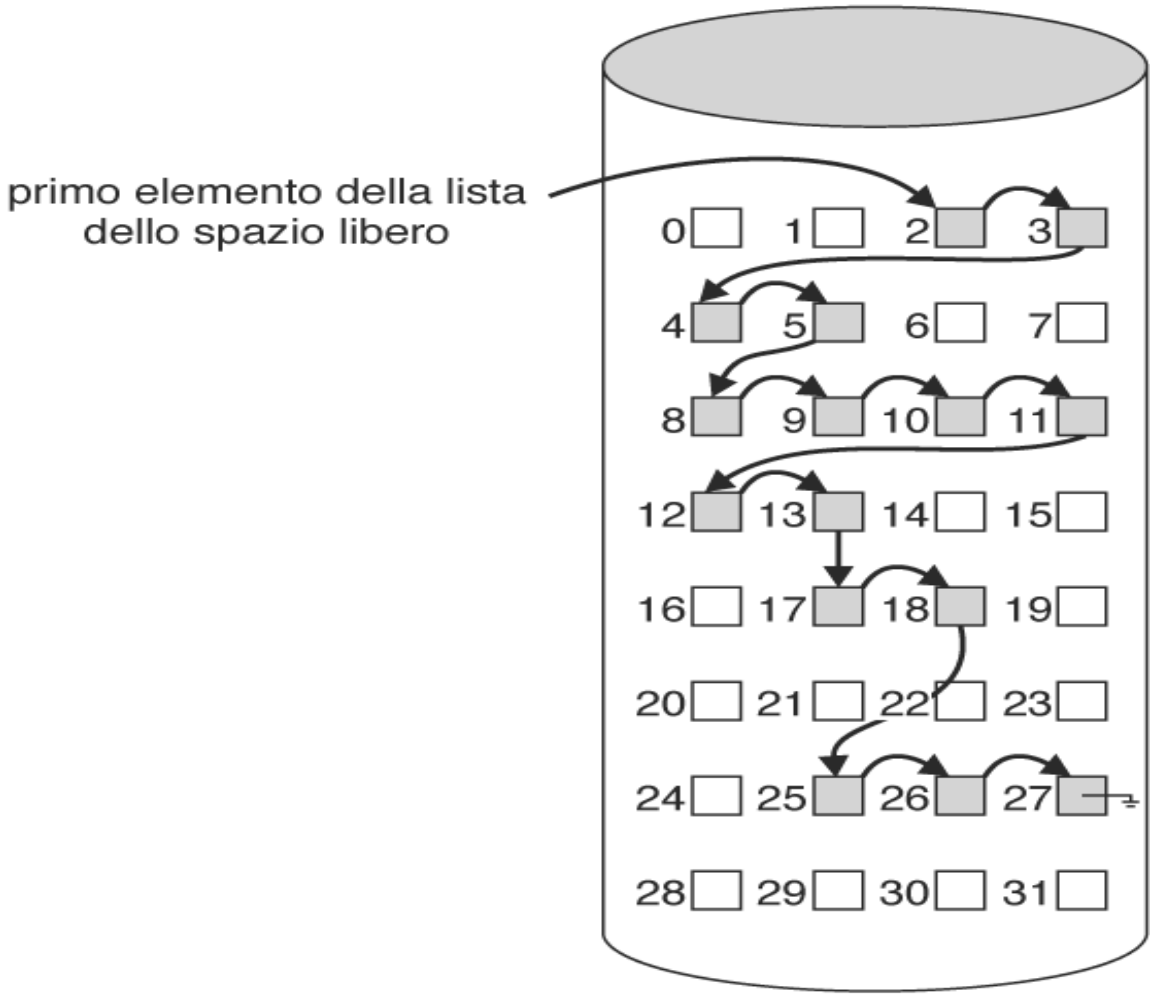
Vettore di bit

- La lista dello spazio libero è un vettore che rappresenta **ogni blocco su disco** con **un bit**
  - bit a 1 = blocco libero
  - bit a 0 = blocco assegnato
- Vantaggi:
  - semplicità
  - **efficienza** nel trovare il primo blocco libero, o *n* blocchi liberi consecutivi (supporto hw per operazioni sui bit)
- Svantaggio:
  - **dimensione** del vettore per dischi grandi
  - metodo efficiente solo se è possibile mantenere l'intero vettore in memoria centrale

Lista concatenata

- I blocchi liberi sono collegati tra loro attraverso **puntatori**
- Il puntatore al primo blocco libero nella lista si tiene in una speciale locazione del disco e viene caricato in memoria
- Schema **inefficiente**: lettura di ogni blocco per attraversare la lista (alto tempo di I/O), ma l'attraversamento della lista è un'operazione **infrequente**
- Se si utilizza la FAT, non è necessario un metodo separato per la gestione dello spazio libero
  - le informazioni sono già nella tabella

Lista concatenata



Varianti

- Variante basata su **raggruppamento**:
  - si memorizzano gli indirizzi di *n* blocchi liberi nel primo blocco libero
  - l'ultimo blocco contiene gli indirizzi di altri *n* blocchi liberi
  - permette di ottenere velocemente gli indirizzi di gruppi di blocchi liberi

- Variante basata su **conteggio**:
  - ogni elemento contiene l'indirizzo di un blocco libero e il numero di blocchi liberi contigui a questo

Efficienza di utilizzo dei dischi

- L'efficienza di utilizzo dei dischi dipende dagli **algoritmi** di **allocazione** dello **spazio** e di **gestione** delle **directory**
- In Unix gli **inode** vengono allocati **preventivamente** e i **blocchi** relativi ad ogni **file** vengono mantenuti **vicini** ai relativi blocchi degli **inode**
- In BSD Unix si utilizza uno schema a **cluster** di dimensioni variabili a seconda della grandezza del file
- E' necessario valutare l'**influenza** sull'efficienza e sulle prestazioni di ogni **informazione che si associa ad un file** (ad esempio, data di ultimo accesso, dimensione dei puntatori)
- Impatto dell'**evoluzione della tecnologia** (dimensione dei dischi, lunghezza delle strutture dati)

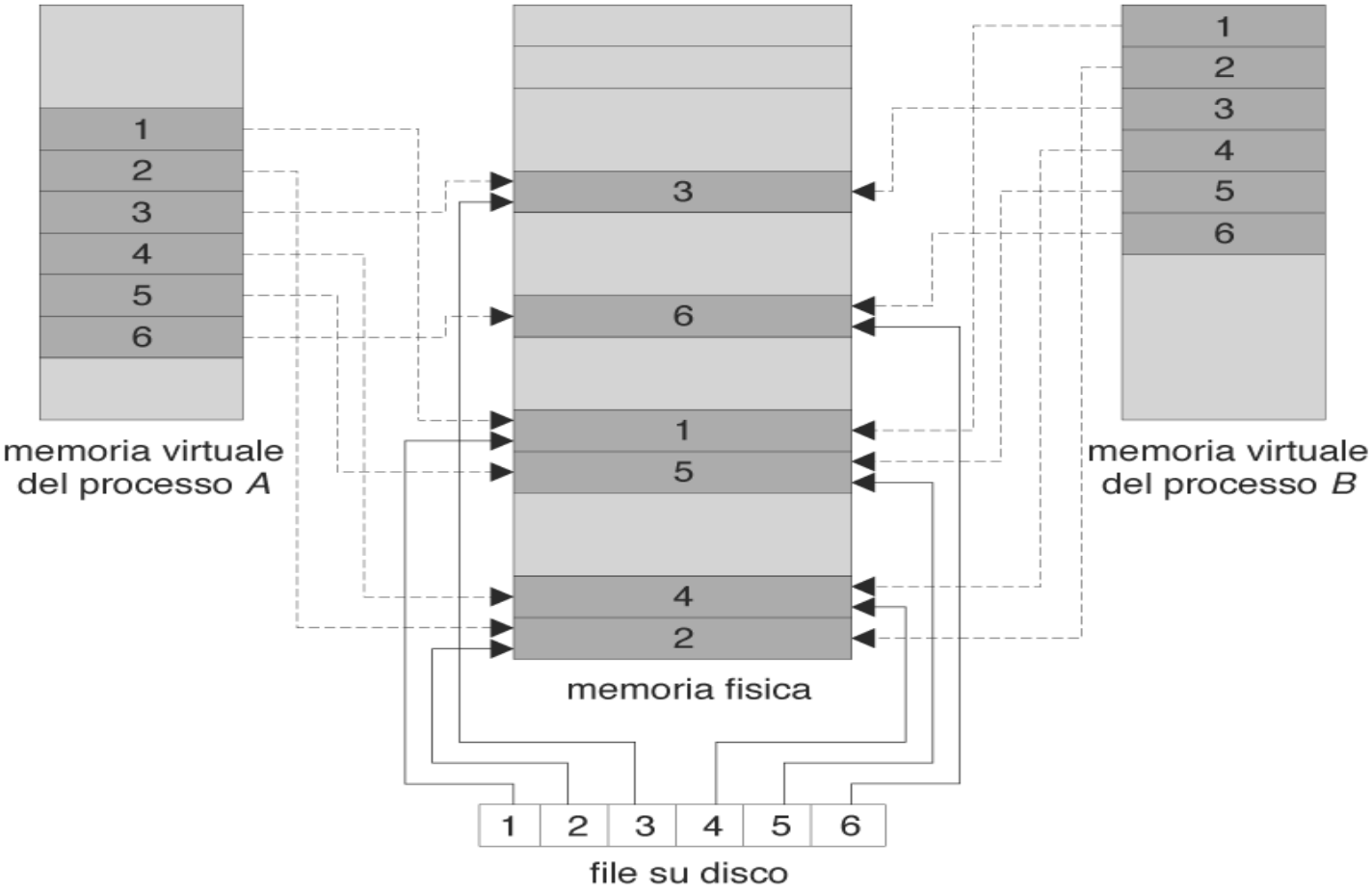
Ottimizzazione delle prestazioni

- Utilizzo di **cache interna** al **controllore** del **disco** per la memorizzazione di intere tracce
- Utilizzo di **memoria centrale** come cache del disco per memorizzare blocchi (**buffer cache**)
- Cache delle **pagine** per i file: usata per il **memory mapping** dei file; tecnica simile alla **paginazione virtuale**

Memory-Mapped Files

- Il memory-mapping dei file permette di **effettuare I/O sui file** tramite accesso alla **memoria** invece che tramite chiamate di sistema open(), read() e write()
- Si realizza associando un **blocco del disco** a una o più **pagine in memoria**
  - Accesso iniziale con richiesta di paginazione
  - Una porzione di file pari a una pagina viene letta dal file system in memoria fisica
  - Letture/scritture successive avvengono tramite accessi ordinari alla memoria
- Permette a processi differenti di mappare lo stesso file in memoria per consentire la **condivisione dei dati** (Windows)

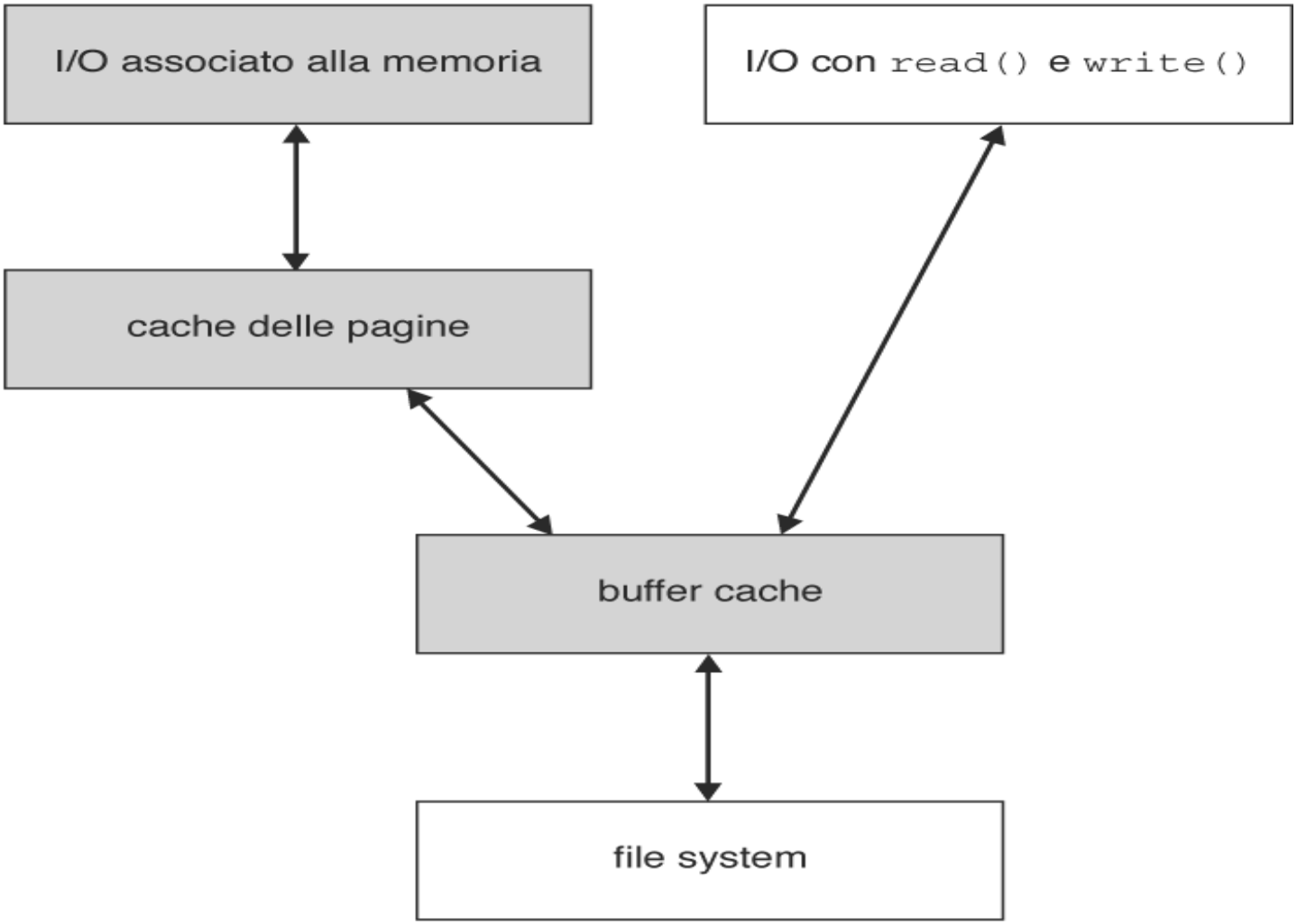
File mappati in memoria



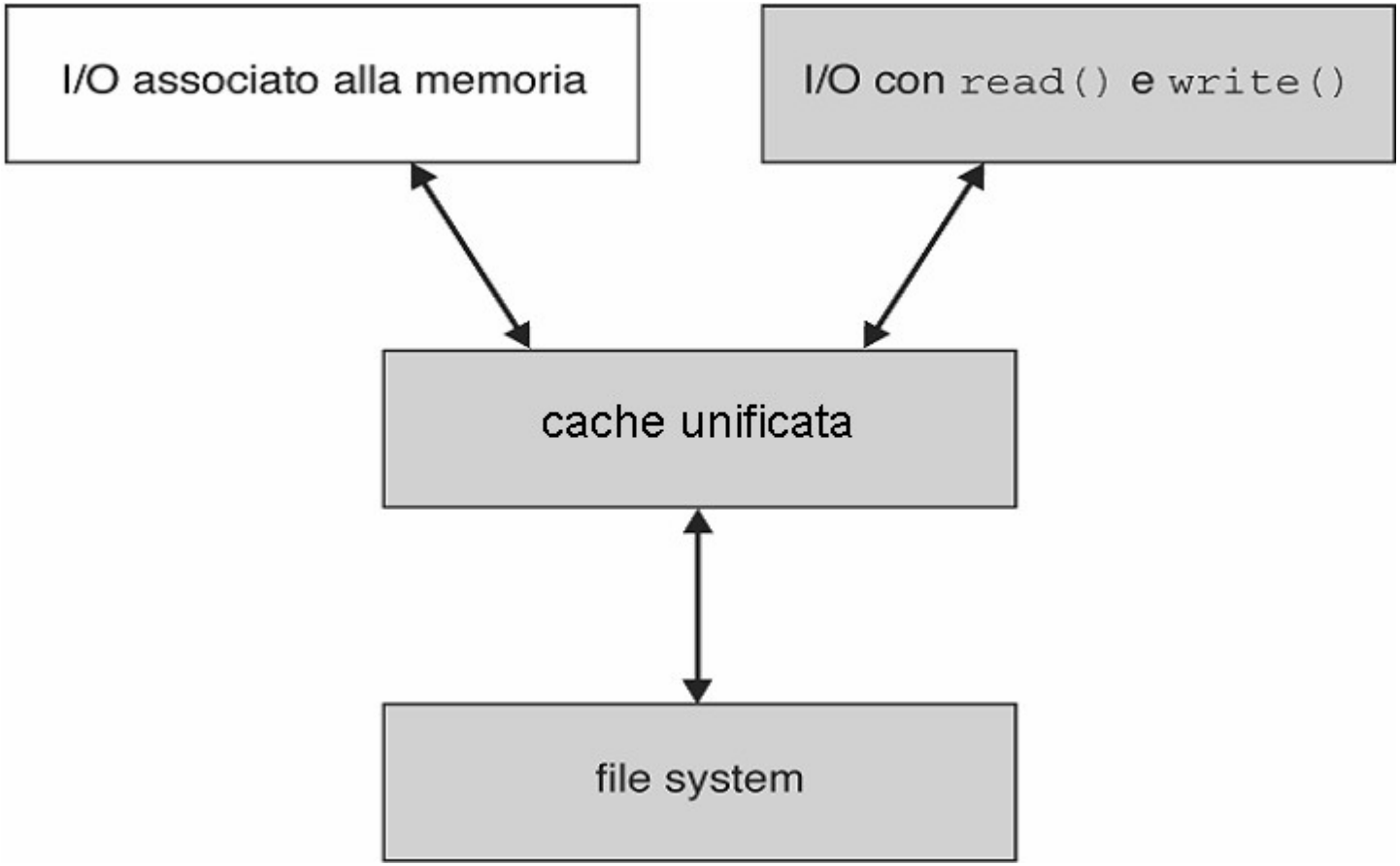
Memory-Mapped Files

- Le scritture su file in memoria **non** implicano scritture **sincrone** su disco
- Alla **chiusura** del file, i dati sono scritti su disco e la memoria liberata
- Alcuni sistemi prevedono un **specifica chiamata di sistema** per la mappatura dei file
- Solaris mappa sempre i file in memoria ma se si usa *mmap()* la mappatura avviene nello spazio di indirizzi del processo, se si usa *read()* e *write()* avviene nello spazio di indirizzi del kernel
- Unix usa meccanismi diversi per la mappatura dei file (*mmap()*) e per la memora condivisa (*shmget()* e *shmat()* di POSIX)

Prestazioni: cache unificata



Prestazioni: cache unificata



Ottimizzazione delle prestazioni

- Distribuzione delle pagine per processi e per cache delle pagine
- Ottimizzazione degli algoritmi di sostituzione delle pagine nella cache per accessi sequenziali:
  - LRU non va usata
  - ***rilascio indietro***: rimozione pagina dalla memoria appena si verifica richiesta per pagina successiva
  - ***lettura anticipata***: lettura e memorizzazione in cache di più pagine successive
  - permettono risparmio di memoria e di tempo
- Scritture sincrone e asincrone
  - Scritture asincrone più veloci delle letture perché effettuate su cache

Verifica della coerenza

- La distribuzione dei dati in più locazioni (memoria secondaria, memoria centrale, cache) può portare a problemi di **coerenza**
- In caso di malfunzionamenti, il file system può contenere file il cui **stato** è **diverso** da quello descritto dalle directory
- Al riavvio del sistema, programmi di sistema come *fsck* o *chkdsk* confrontano i dati delle directory con quelli dei blocchi fisici e tentano di correggere le incoerenze
- L'efficacia di tali programmi dipende dal tipo di allocazione e gestione dello spazio libero (es: ricostruzione file possibile con allocazione concatenata)
- Per limitare i problemi, Unix usa **scritture sincrone dei metadati**

Journaled file system

- I journaled file system trattano ogni aggiornamento del file system come una **transazione**
- Tutte le transazioni sono scritte in un **log**
  - Una transazione viene considerata **committed** nel momento in cui viene scritta su log
- Le transazioni del log vengono scritte in modo asincrono nel file system
  - Quando il file system viene aggiornato, la transazione viene rimossa dal log
- Se il file system va in crash, bisogna effettuare tutte le **transazioni ancora** presenti **nel log**
- Scrittura sincrona su log e scrittura asincrona nel file system = buone prestazioni