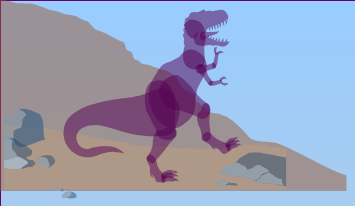
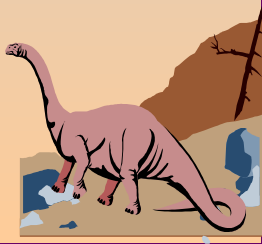


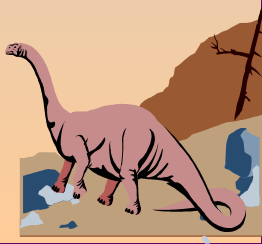
Capitolo 11: Realizzazione del File System

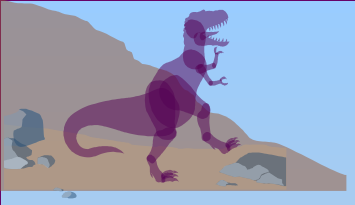
- Struttura del file system
- Realizzazione del file system
- Realizzazione delle directory
- Metodi di allocazione
- Gestione dello spazio libero
- Efficienza e prestazioni
- Ripristino
- File system con annotazione delle modifiche
- NFS
- Esempio: il file system WAFL



File system

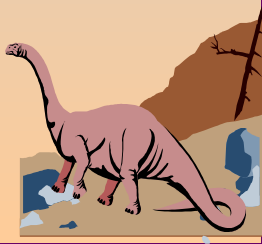
- Tutte le applicazioni informatiche hanno bisogno di memorizzare e recuperare informazioni. Abbiamo tre requisiti essenziali per la memorizzazione delle informazioni a lungo termine:
  - Si deve potere memorizzare un'enorme quantità di informazioni.
  - Le informazioni devono sopravvivere alla terminazione del processo che le usa.
  - Più processi devono poter accedere alle informazioni in modo concorrente.
- Il *file system* è l'insieme delle strutture dati e dei metodi che ci permettono la registrazione e l'accesso a dati e programmi presenti in un sistema di calcolo.
- Il file system risiede permanentemente nella memoria secondaria, progettata per mantenere in maniera non volatile grandi quantità di dati.





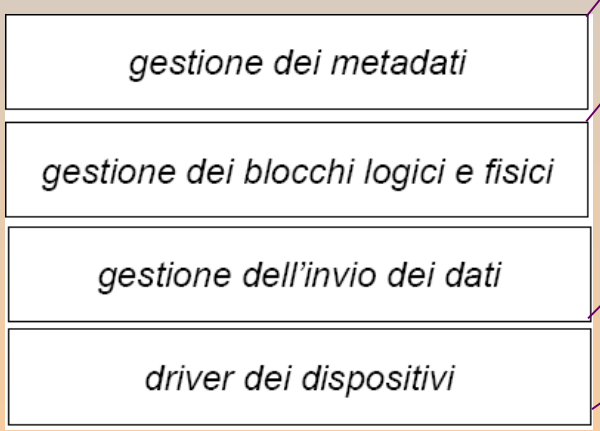
Struttura del file system

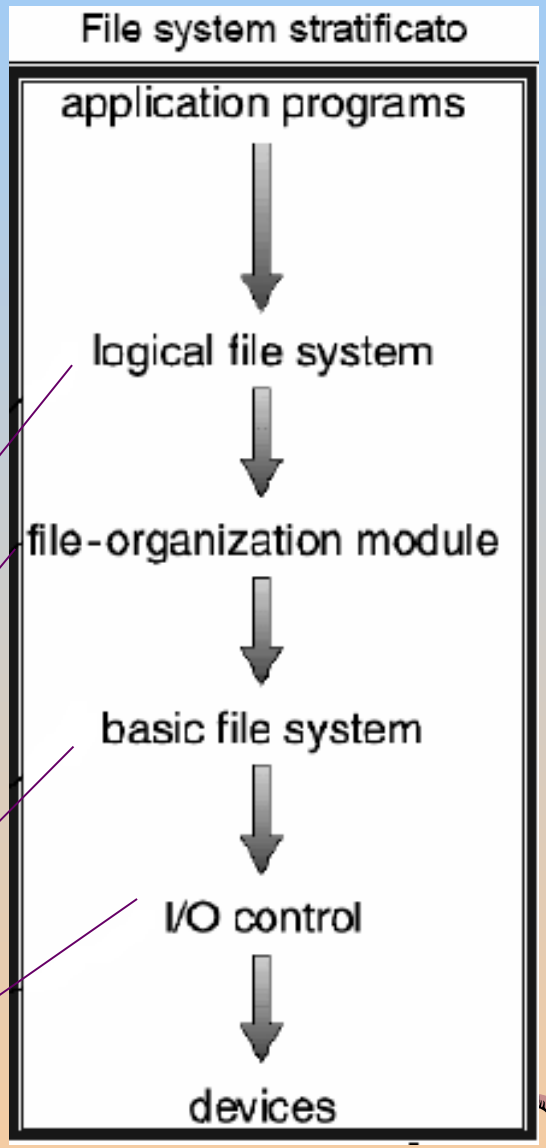
- In generale i dischi sono il supporto di memoria secondaria su cui viene conservato il file system.
- Per migliorare l'efficienza dell'I/O, i trasferimenti di I/O tra memoria e disco vengono eseguiti in unità di **blocchi**.
- Ogni blocco è costituito da uno o più settori, la dimensione dei quali può variare da 32 a 4096 byte a seconda del tipo di unità a disco.
- I dischi hanno due caratteristiche importanti:
  - Possono essere riscritti localmente:
    - è possibile leggere un blocco dal disco, modificarlo e quindi riscriverlo nella stessa posizione.
  - E' possibile accedere direttamente a qualsiasi blocco,
    - quindi risulta semplice accedere a qualsiasi file, sia in modo sequenziale che casuale,
    - e passare da un file all'altro solamente spostando le testine di lettura-scrittura e attendendo la rotazione del disco.



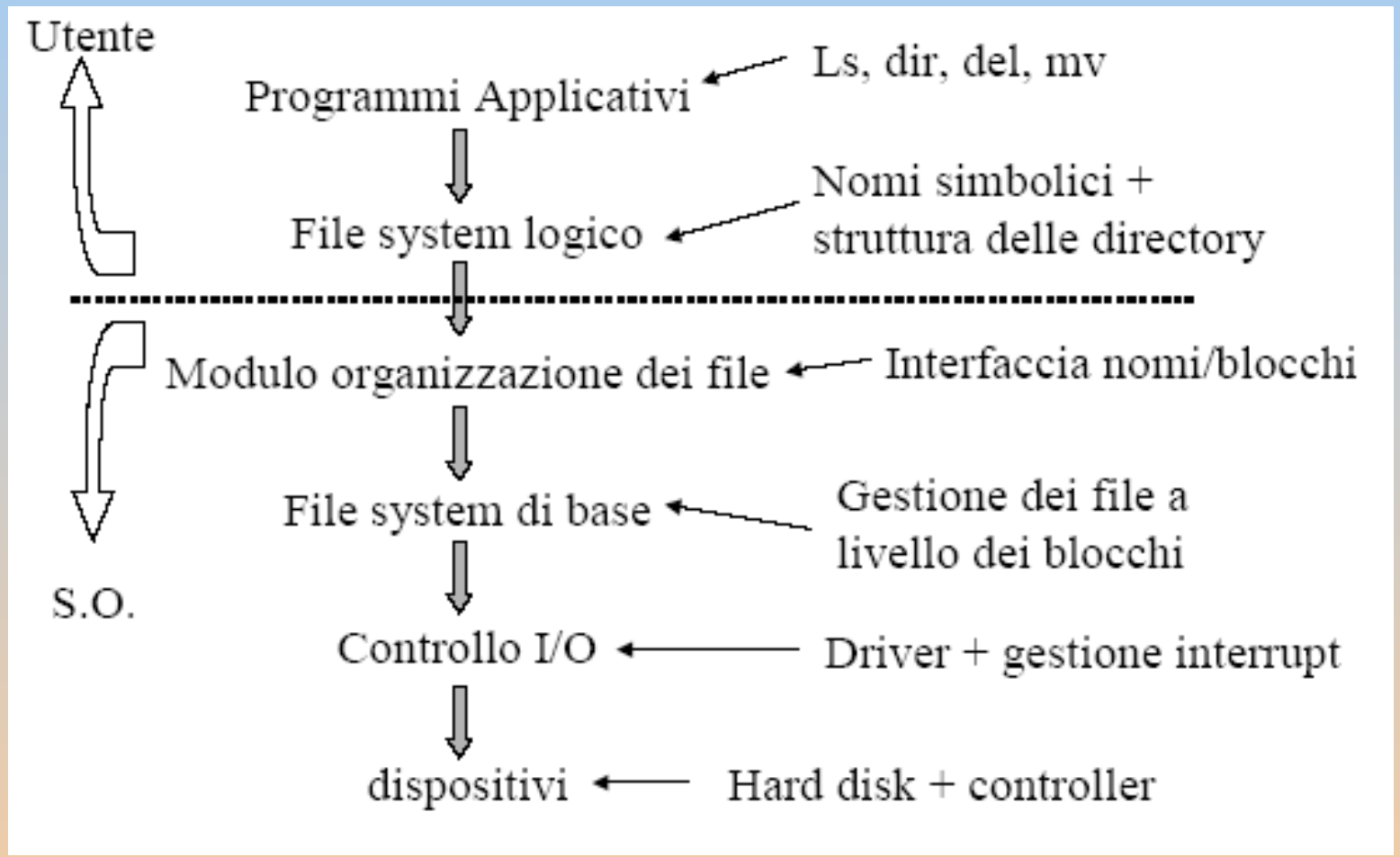
Struttura del file system (II)

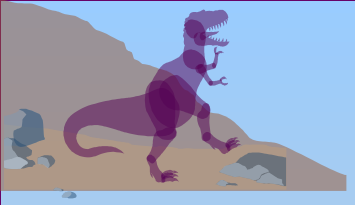
- File system:
  - Interfaccia utente;
  - Algoritmi e strutture dati.
- Il file system risiede nella memoria secondaria (dischi).
- Il file system viene organizzato secondo livelli (stratificato).





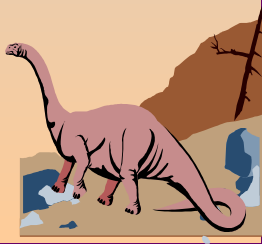
File system stratificato

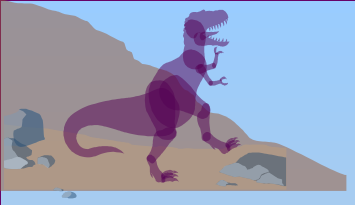




File system stratificato (II)

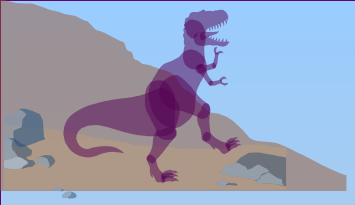
- Il livello più basso, il **controllo dell'I/O**, è costituito dai *driver dei dispositivi* e dai gestori di interrupt per trasferire le informazioni tra memoria e il sistema dei dischi.
  - Un driver di dispositivo può essere pensato come un traduttore.
  - Il suo input consiste di comandi ad alto livello; il suo output consiste di istruzioni di basso livello.
- Il **file system di base** deve soltanto inviare dei generici comandi all'appropriato driver di dispositivo per leggere e scrivere blocchi fisici sul disco.
  - Ogni blocco fisico è identificato dal suo indirizzo numerico del disco.





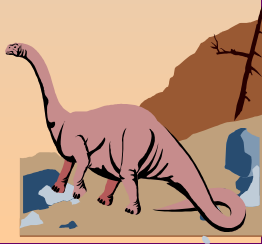
File system stratificato (III)

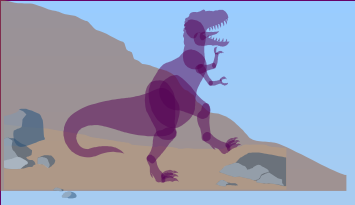
- Il **modulo di organizzazione dei file** è a conoscenza dei file e dei loro blocchi logici, cosi' come dei blocchi fisici del disco.
  - Conoscendo il tipo di allocazione dei file utilizzato e la locazione dei file, il modulo di organizzazione dei file può tradurre gli indirizzi dei blocchi logici (numerati da 0 a N) che il file system di base deve trasferire, negli indirizzi dei blocchi fisici.
- Il modulo di organizzazione dei file comprende anche il gestore dello spazio libero,
  - il quale registra i blocchi non allocati e li mette a disposizione del modulo di organizzazione dei file quando sono richiesti.
- Infine, il **file system logico** utilizza la struttura di directory per fornire al modulo di organizzazione dei file le informazioni di cui questo necessita.
- Dato un nome simbolico di file. Il file system logico è anche responsabile della protezione e della sicurezza.
  - Alcuni S.O., tra cui UNIX, trattano una directory esattamente come un file, con un campo che indica che si tratta di una directory.
  - Altri S.O., ad es. Windows NT, usano system call diverse per file e directory.



Strutture su disco utilizzate dal file system

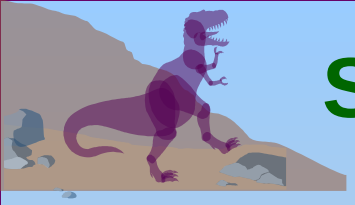
- Per realizzare un file system sono necessarie numerose strutture, Sia nei dischi che in memoria.
- Fra le strutture presenti nei dischi ci sono le seguenti:
  - **Blocco di controllo d'avviamento** (*boot control block*), che contiene le informazioni necessarie al sistema per l'avviamentodi un S.O. da quella partizione
  - **Blocchi di controllo delle partizioni** (*partition control block*), ciascuno di essi contiene i dettagli riguardanti la relativa partizione, come il numero e la dimensione dei blocchi nel disco, il contatore dei blocchi liberi ed i relativi puntatori, il contatore dei file liberi ed i relativi puntatori
  - **Strutture di directory**, che si usano per organizzare i file
  - **Descrittori di file** (es. inode), che contengono permessi di accesso ai file, proprietari, dimensioni e locazioni dei blocchi di dati, etc.
- Queste strutture variano a seconda della particolare implementazione del file system (UNIX, WINDOWS NT ed XP, etc.)





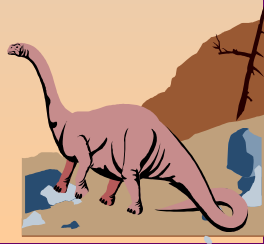
Tipico descrittore di file

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks



Strutture in memoria utilizzate dal file system

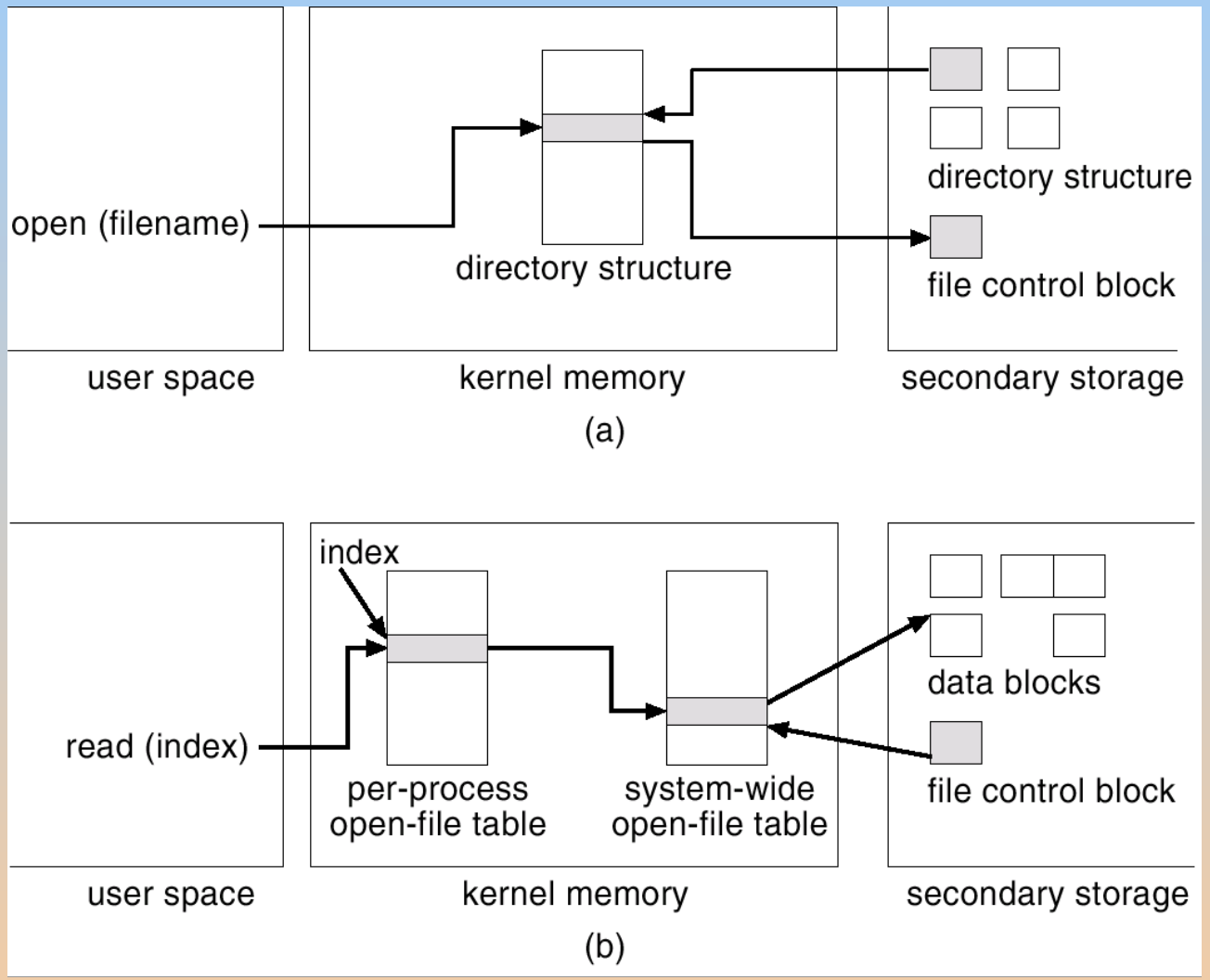
- Fra le strutture in memoria ci sono le seguenti:
  - **Tabella delle partizioni**, contenente informazioni su ciascuna delle partizioni montate
  - **Struttura di directory**, contenente le informazioni relative a tutte le directory recentemente utilizzate
  - **Tabella generale dei file aperti**, contenente una copia del descrittore di file per ciascun file aperto, insieme ad altre informazioni
  - **Tabella dei file aperti per ciascun processo**, contenente un puntatore all'appropriato elemento della tabella generale dei file aperti, insieme ad altre informazioni



Apertura di un file

- Prima che possa essere impiegato per procedure di I/O, un file deve essere aperto.
- Quando viene aperto un file, nella struttura di directory viene cercato l'elemento associato al file richiesto.
- Una volta individuato il file, le informazioni a esso associate: dimensione, proprietario, permessi d'accesso, locazione dei blocchi di dati, etc. sono copiate in una tabella mantenuta in memoria, detta dei **file aperti**,
  - contenente le informazioni su tutti i file correntemente aperti.
- Una open, attiva la ricerca nella directory corrispondente copiando il file nella tabella dei file aperti.
- L'indice di questa tabella viene riportato al programma utente e tutti i successivi riferimenti vengono effettuati attraverso tale indice.
  - L'indice prende nomi diversi: nel sistema UNIX è detto **descrittore di file**; in Windows NT **file handle** (maniglia), in altri sistemi, **file control block**.

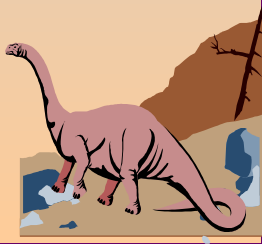
Apertura e lettura di un file



(a) si riferisce all'apertura di un file.
(b) si riferisce alla lettura di un file.

File system virtuali

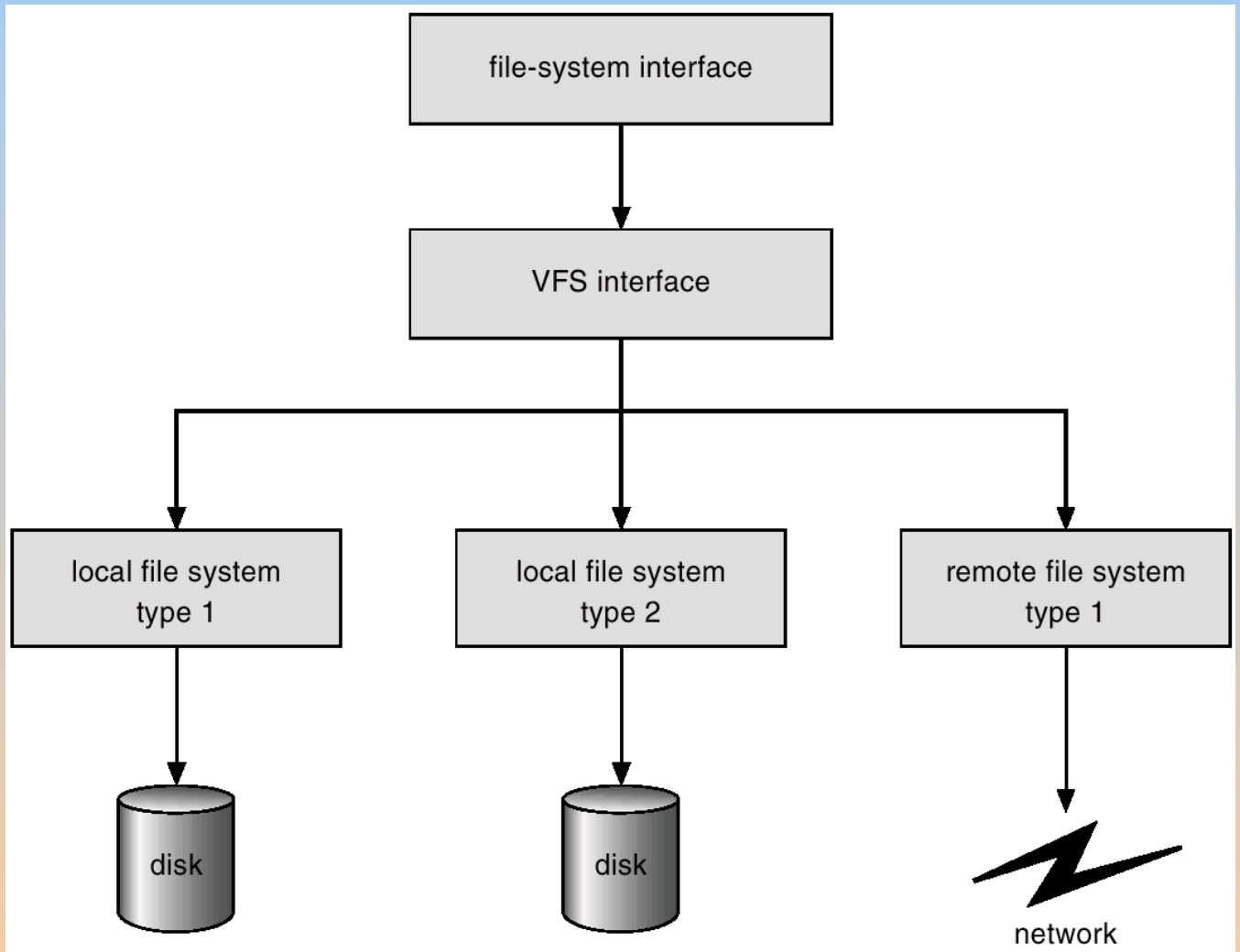
- I sistemi operativi moderni si trovano a gestire diversi tipi di file system contemporaneamente.
- Bisogna quindi considerare il modo in cui un S.O. può consentire l'integrazione di diversi tipi di file system in un'unica struttura di directory,
  - in modo da permettere agli utenti di spostarsi da un tipo di file system ad un altro.
- Un metodo potrebbe essere quello di scrivere procedure di gestione separate di file e directory per ciascun file system.
  - Poco efficiente e dispendioso in termini di dimensioni del S.O.
  - La maggior parte dei S.O. impiega tecniche orientate agli oggetti per semplificare ed organizzare in maniera modulare la soluzione



File system virtuali (II)

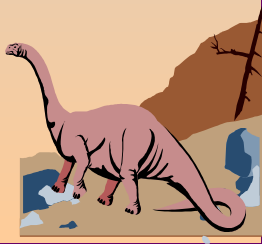
- Si possono quindi isolare le funzioni di base delle chiamate di sistema dai dettagli implementativi del singolo file system,
  - adoperando opportune strutture dati.
- In questo modo la realizzazione del file system si articola in tre strati principali.
- Il primo strato è l'interfaccia del file system, basata sulle chiamate del sistema *open, write, read, close*, e sui descrittori di file.
- Il secondo strato si chiama **file system virtuale** e svolge le seguenti funzioni:
  - Separa le operazioni generiche del file system dalla loro implementazione, definendo un'interfaccia uniforme.
  - Mantiene una struttura di rappresentazione dei file, detta **vnode**, che contiene i, indicatore numerico unico per ciascun file
  - e quindi identifica univocamente ciascun file presente su una partizione montata.
- Lo strato che realizza il protocollo NFS è il più basso dell'architettura.

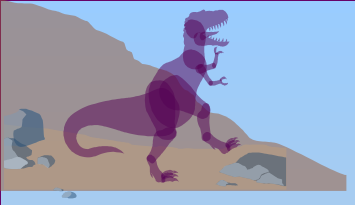
Schema di un file system virtuale



Directory

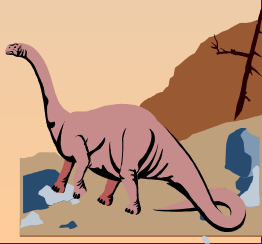
- Quando un file viene aperto il sistema operativo usa il nome di percorso fornito dall'utente per individuare l'elemento corrispondente all'interno della directory.
- La voce nella directory fornisce le informazioni necessarie a trovare i blocchi sul disco.
- A seconda del metodo di rappresentazione scelto, questa informazione può essere:
  - l'indirizzo sul disco dell'intero file (allocazione contigua),
  - l'indice del primo blocco o
  - l'indice di i-node.
- La funzione principale della struttura delle directory è di tradurre il nome ASCII del file nelle informazioni necessarie a individuare i dati.





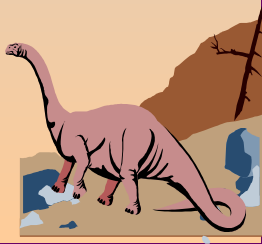
Realizzazione delle directory

- Come realizzare le directory:
  - Lista lineare di nomi di file con puntatori ai data block.
    - Facile da realizzare
    - Dispendioso da eseguire
  - Tabella hash – lista lineare con strutture dati hash.
    - Diminuisce il tempo di ricerca nella directory
    - *collisione* – situazione dove due file diversi anno lo stesso indirizzo hash
    - Dimensione fissata
- Un problema strettamente collegato all'allocazione dei dati è dove debbano essere memorizzati gli attributi.
- Una possibilità ovvia è di memorizzarli direttamente all'interno della directory.



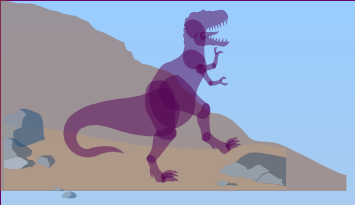
Metodi di assegnazione

- La natura ad accesso diretto dei dischi permette una certa flessibilità nell'implementazione dei file.
- In quasi tutti i casi, molti file vengono memorizzati sullo stesso disco.
- Il problema principale consiste dunque nell'allocare lo spazio per i file in modo che lo spazio sul disco venga utilizzato efficientemente e l'accesso ai file sia rapido.
- Il metodo di allocazione dello spazio su disco descrive come i blocchi fisici del disco vengono allocati ai file.
- Esistono tre metodi principali per l'assegnazione dello spazio di un disco:
  - Allocazione contigua
  - Allocazione concatenata
  - Allocazione indicizzata



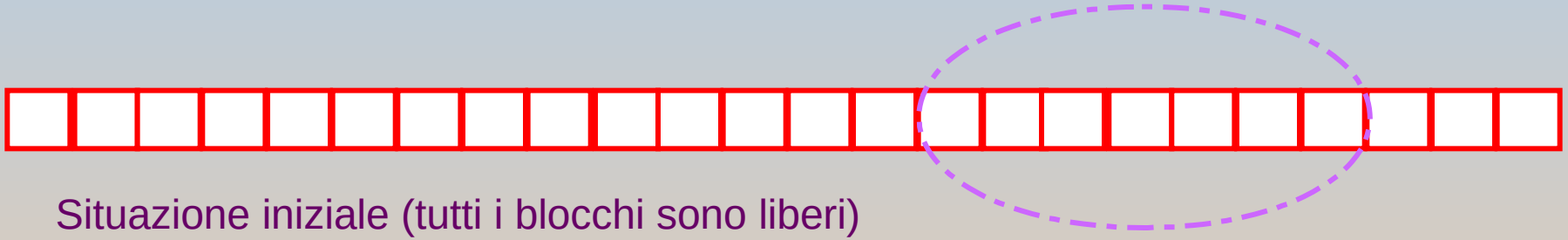
Assegnazione contigua

- Lo schema più facile consiste nel memorizzare ogni file come un blocco contiguo di dati sul disco.
- Ciascun file occupa un insieme di blocchi contigui sul disco.
- Per reperire il file occorrono solo la locazione iniziale (# blocco iniziale) e la lunghezza (numero di blocchi).
- Accesso casuale.
- Questo metodo ha due vantaggi significativi:
  - è semplice da implementare, dato che tutti i blocchi del file sono univocamente identificati da un numero
  - le prestazioni sono eccellenti dal momento che l'intero file può essere letto dal disco con singola operazione
- Svantaggi
  - non è facilmente implementabile, a meno che le dimensioni massime del file non siano note nel momento in cui i file viene creato
  - Spreco di spazio: frammentazione esterna (problema di allocazione dinamica della memoria)



Assegnazione contigua (II)

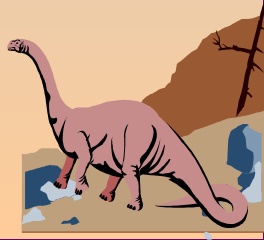
- Allocazione contigua dello spazio disco
  - ogni file viene memorizzato in un gruppo di blocchi contigui (*run*)
  - es :



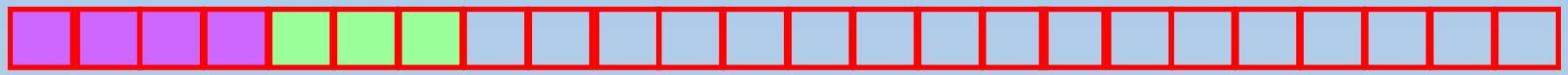
Situazione iniziale (tutti i blocchi sono liberi)

run

Situazione dopo l'allocazione del File A (4 blocchi)



Assegnazione contigua (III)



Situazione dopo l'allocazione del File A (4 blocchi) e B (3 blocchi)

File B (3 blocchi)

File D (3 blocchi)

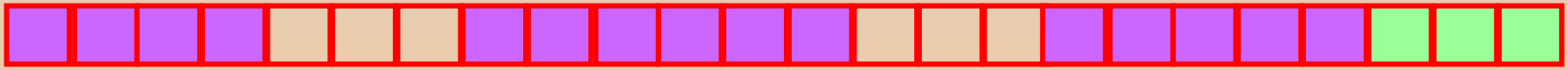
File G (3 blocchi)

File A (4 blocchi)

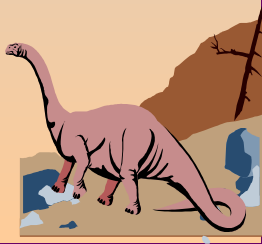
File C (6 blocchi)

File E (5 blocchi)

Situazione dopo la cancellazione di B e D

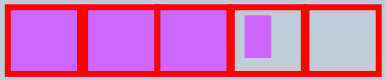


Blocchi liberi

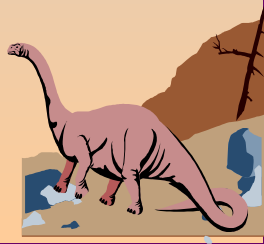


Assegnazione contigua (IV)

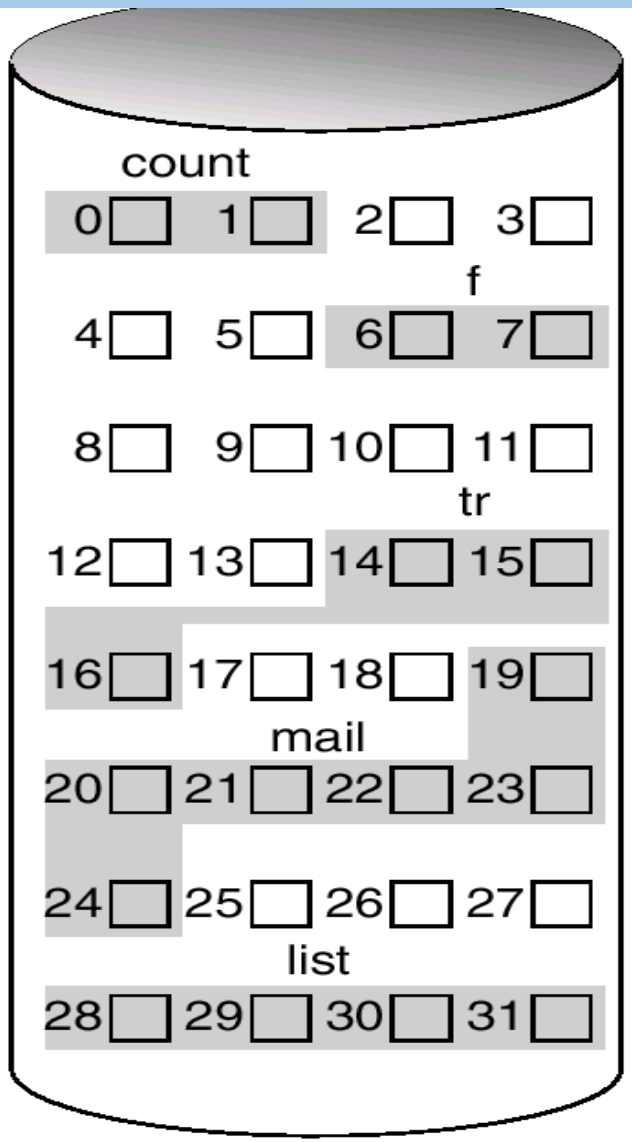
- Fenomeno della *frammentazione interna :*
  - se l'ultimo blocco non è del tutto pieno si spreca dello spazio



- L'allocazione contigua viene spesso utilizzata nei file system dei CD-ROM e dei DVD.



Assegnazione contigua dello spazio dei dischi

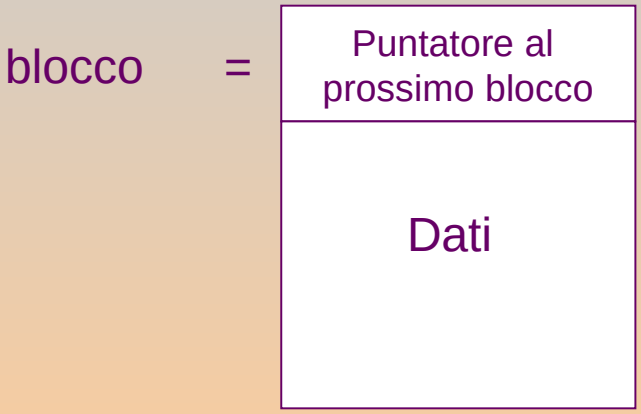


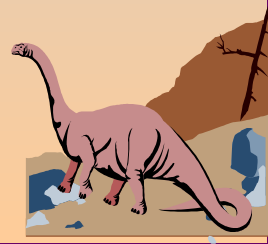
directory

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

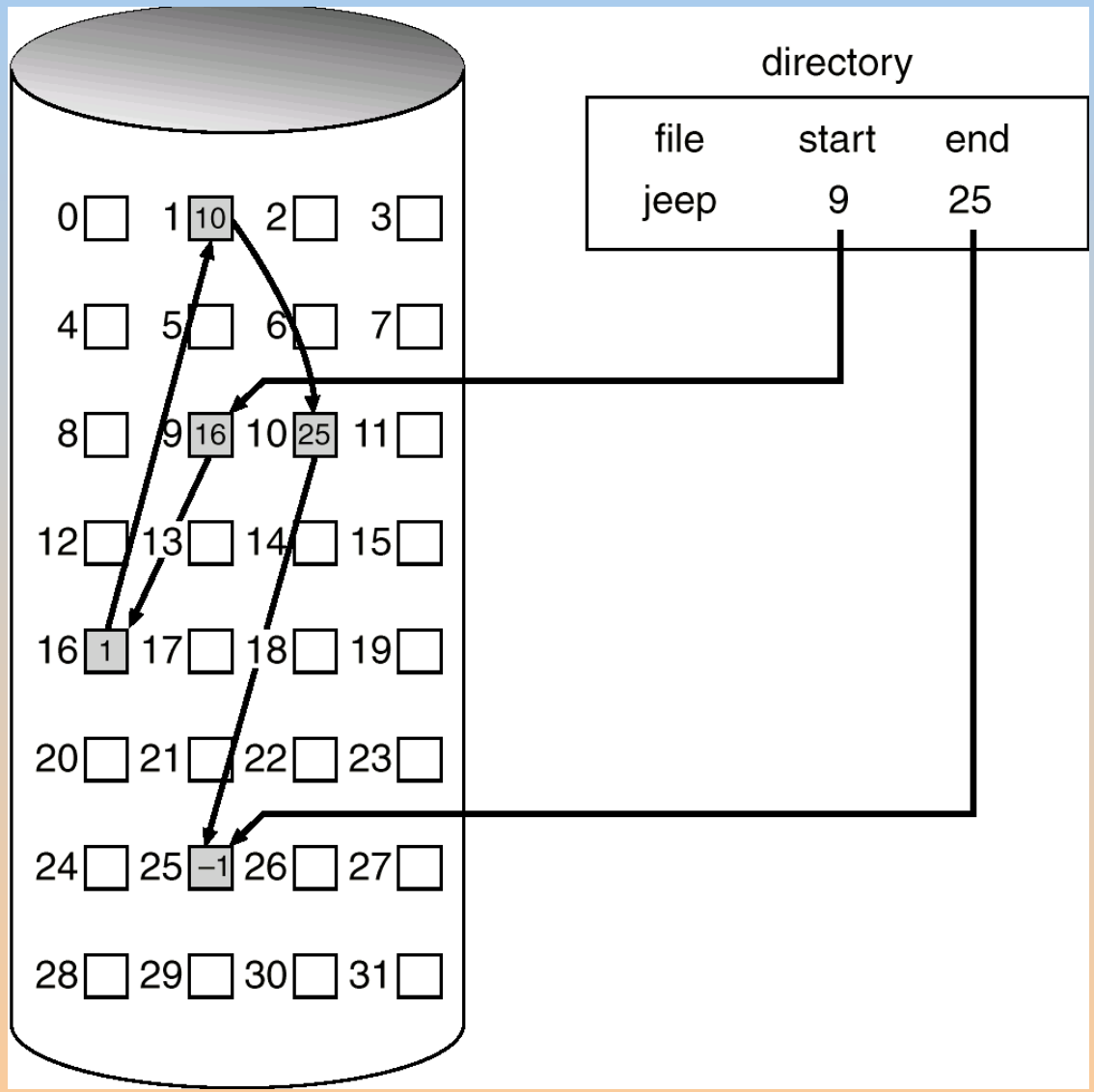
Assegnazione concatenata

- L'allocazione concatenata risolve i problemi dell'allocazione contigua.
- Ogni file è costituito da una lista concatenata di blocchi del disco i quali possono essere sparsi in qualsiasi punto del disco stesso.
- Ogni blocco contiene un puntatore al blocco successivo.
- La directory contiene un puntatore al primo e all'ultimo blocco del file.

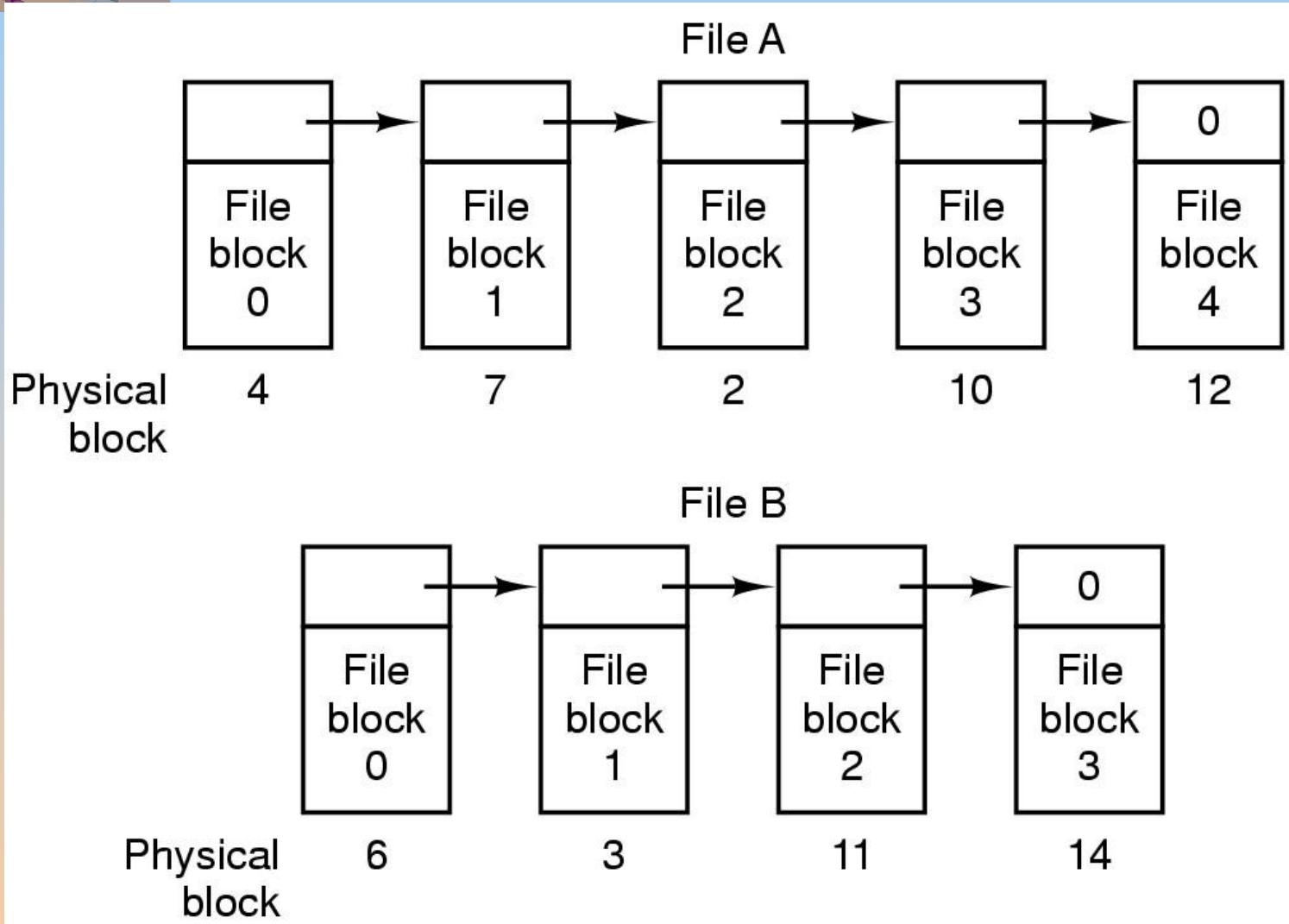




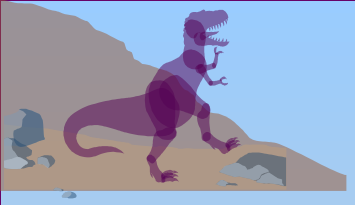
Assegnazione concatenata dello spazio nei dischi



File: lista concatenata di blocchi

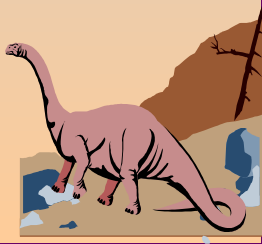


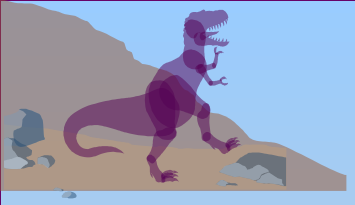
Memorizzazione come lista concatenata di blocchi



Assegnazione concatenata (II)

- Non si ha frammentazione esterna
- La memorizzazione dei riferimenti riduce lo spazio disponibile per i dati: i puntatori al blocco successivo non sono disponibili all'utente.
- Quindi se ogni blocco è formato da 512 byte e un indirizzo del disco richiede 4 byte, allora l'utente vede blocchi di 508 byte.
- Quindi con blocchi di 512 byte e riferimenti di 4 byte si ha uno spreco di spazio pari allo 0.78% del disco.
- Svantaggio principale: può essere utilizzata efficientemente solo per file ad accesso sequenziale.
- Per trovare l'i-esimo blocco di un file occorre partire dall'inizio del file e seguire i puntatori finché non si arriva all 'i-esimo blocco.
  - Ogni accesso a un puntatore implica una lettura del disco, e talvolta un posizionamento della testina sul disco.





Assegnazione concatenata (III)

- La soluzione più comune a questo problema consiste nel raccogliere i blocchi in gruppi, detti ***cluster***, e nell'allocare i cluster anziché i blocchi.
- Ad esempio, il file system può definire un cluster di 4 blocchi e operare su disco soltanto in unità di cluster, permettendo così di occupare meno spazio su disco.
  - Si ha un miglioramento delle prestazioni per via del numero minore di riposizionamenti della testina
  - Si ha una riduzione dello spazio utilizzato per i riferimenti
  - Si ha una maggiore frammentazione interna
- Un altro problema riguarda l'affidabilità, in relazione a situazioni in cui un puntatore viene perso o danneggiato.
- Le liste di blocchi sono *fragili:*
  - la perdita di un solo riferimento può rendere inaccessibile grandi quantità di dati

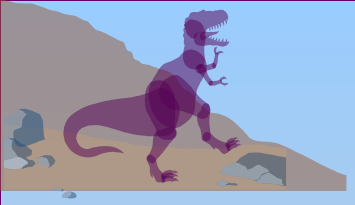


Tabella di assegnazione dei file

- Una soluzione parziale a questo problema consiste nell'utilizzare liste doppiamente concatenate
- oppure nel memorizzare il nome del file e il relativo numero di blocco in ogni blocco.
  - Questi schemi richiedono però un maggiore overhead.
- Una variante del metodo di assegnazione concatenata consiste nell'utilizzo di un indice detto **tabella di assegnazione dei file (FAT)** per ogni partizione.
- La FAT ha un elemento per ogni blocco del disco ed è indicizzata dal numero di blocco.

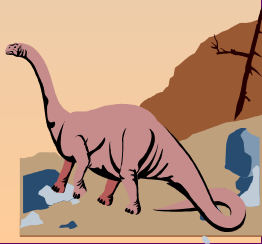


Tabella di assegnazione dei file (II)

- Viene utilizzata essenzialmente come una lista concatenata.
- Per contenerla si riserva una sezione del disco all'inizio di ciascuna partizione,
- per utilizzarla la si porta in memoria centrale.
- L'elemento di directory contiene il numero di blocco del primo blocco del file.
- L'elemento della tabella indicizzato da quel numero di blocco contiene a sua volta il numero di blocco del blocco successivo del file.
- Questa catena continua fino all'ultimo blocco, che ha come elemento della tabella un carattere speciale di fine file.
- I blocchi inutilizzati sono indicati da un valore 0 nella tabella.

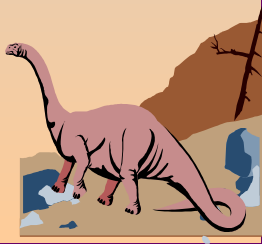
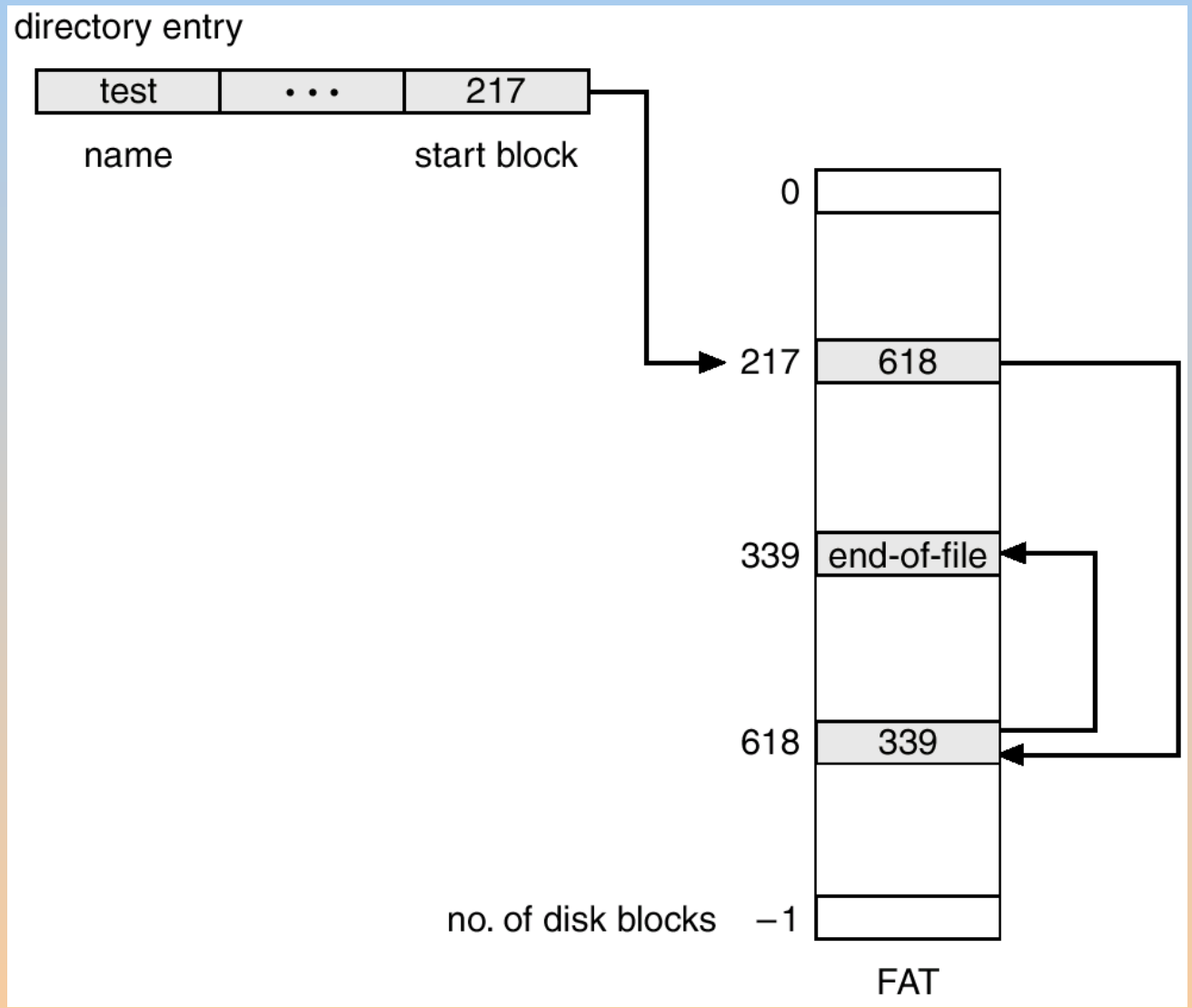
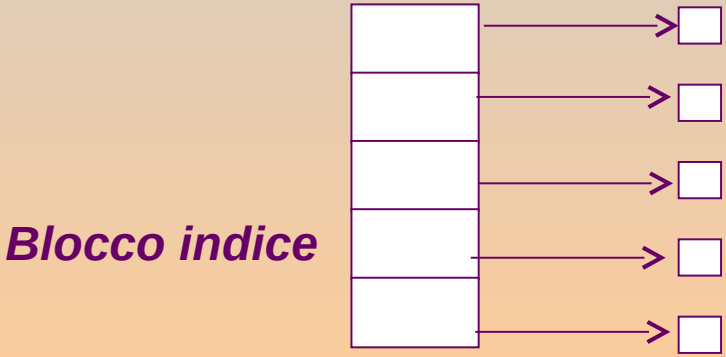


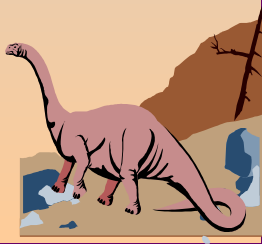
Tabella di assegnazione dei file (II)



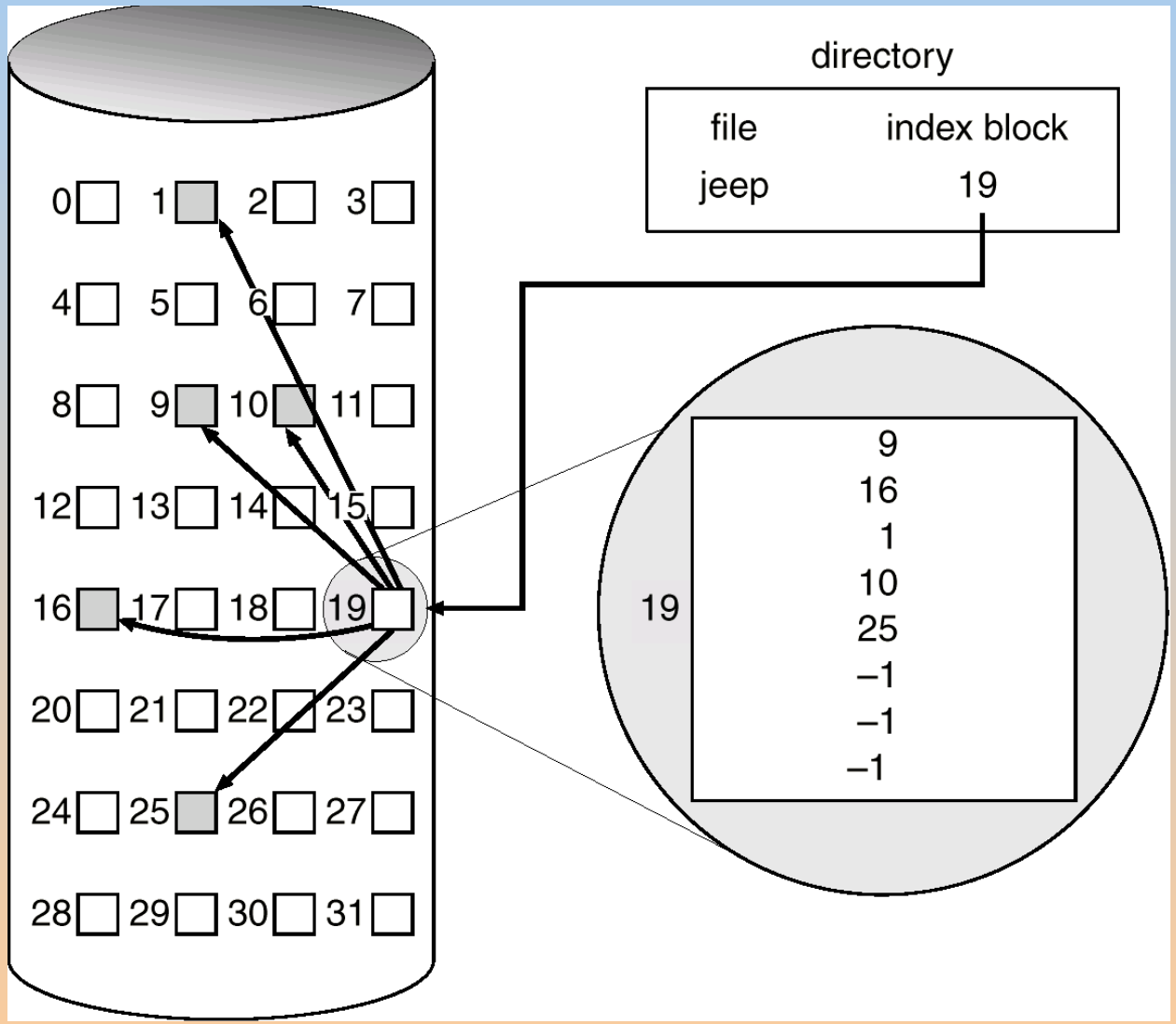
Assegnazione indicizzata

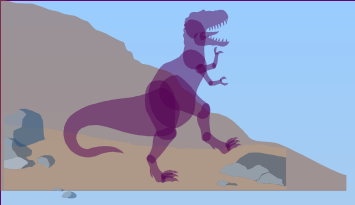
- L'elenco dei blocchi che compongono un file viene memorizzato in un blocco (o area) detto **blocco indice**.
- Per accedere ad un file, si carica in memoria il suo blocco indice e si utilizzano i puntatori in esso contenuti.
- Usando questa struttura l'intero blocco diventa disponibile per contenere dati.
- L'accesso casuale è molto più semplice, in quanto la tabella è contenuta interamente in memoria.
- E' sufficiente, come nei casi precedenti, gestire un solo intero (l'indice del primo blocco) per ogni elemento della directory per essere in grado di individuare tutti i blocchi, qualsiasi sia la grandezza del file.





Assegnazione indicizzata dello spazio dei dischi

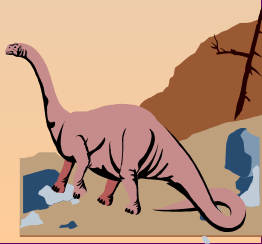




Assegnazione indicizzata (II)

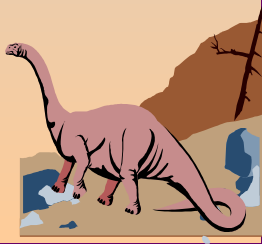
- **Vantaggi**
  - risolve il problema della frammentazione esterna.
  - permette una gestione efficiente dell'accesso diretto
  - il blocco indice deve essere caricato in memoria solo quando il file è aperto
- **Svantaggi**
  - la dimensione del blocco indice determina l'ampiezza massima del file
  - utilizzare blocchi indici troppo grandi comporta un notevole spreco di spazio

- Ogni file deve avere un blocco indice, quindi è necessario cercare di mantenere le dimensioni dei blocchi indice più piccole possible.
- Naturalmente se il blocco è troppo piccolo non può contenere un numero di puntatori sufficiente per un file di grandi dimensioni.



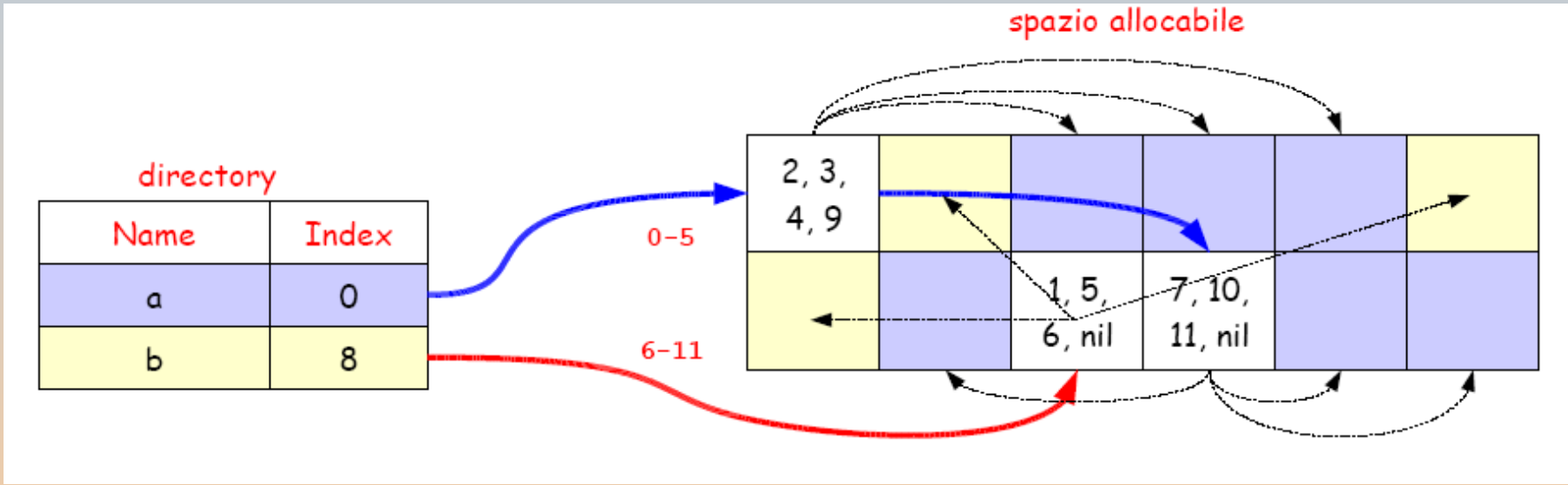
Dimensione del blocco indice

- Possibili meccanismi per memorizzare il blocco indice:
  - **Schema concatenato**: per permettere la presenza di file lunghi è possibile collegare tra loro più blocchi indice. L'ultima parola del blocco indice sarà *nil* oppure un puntatore ad un altro blocco indice.
  - **Indice a più livelli**: un blocco indice di primo livello punta ad un insieme di blocchi indice di secondo livello che, a loro volta, puntano ai blocchi dei file.
  - **Schema combinato**: Ad ogni file è associato un **inode** che contiene 15 puntatori.
    - I primi 12 vengono usati come puntatori a blocchi diretti, come in un normale indice.
    - Gli altri 3 puntano a blocchi indiretti:
    - il primo ad un **blocco indiretto singolo**, cioè ad un blocco indice che non contiene dati ma indirizzi di blocchi che contengono dati,
    - il secondo ad un **blocco indiretto doppio**,
    - il terzo ad un **blocco indiretto triplo.**

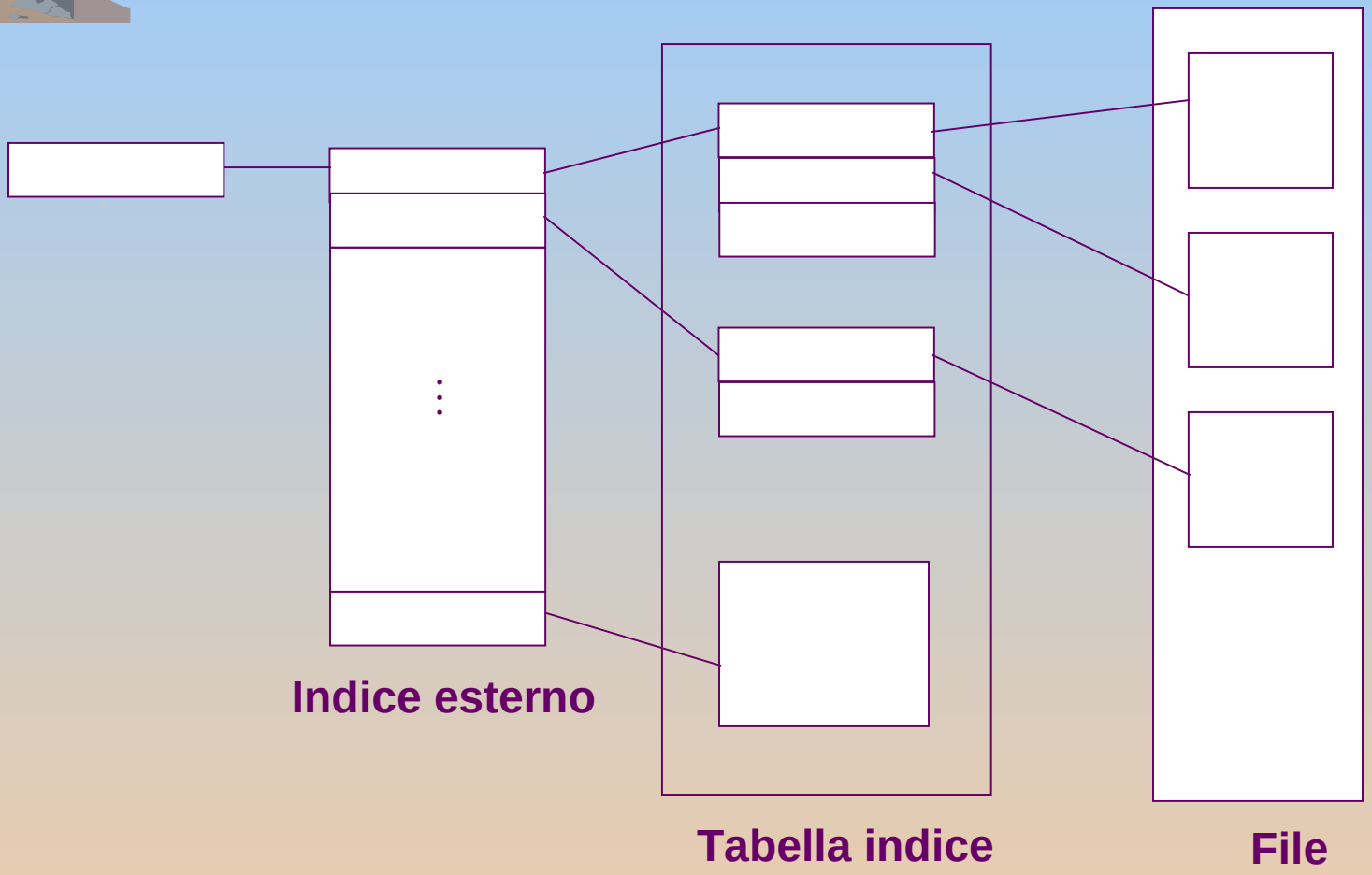


Schema concatenato

- L'ultimo elemento del blocco indice non punta al blocco dati ma al blocco indice successivo.
- Si ripropone il problema dell'accesso diretto a file di grandi dimensioni.

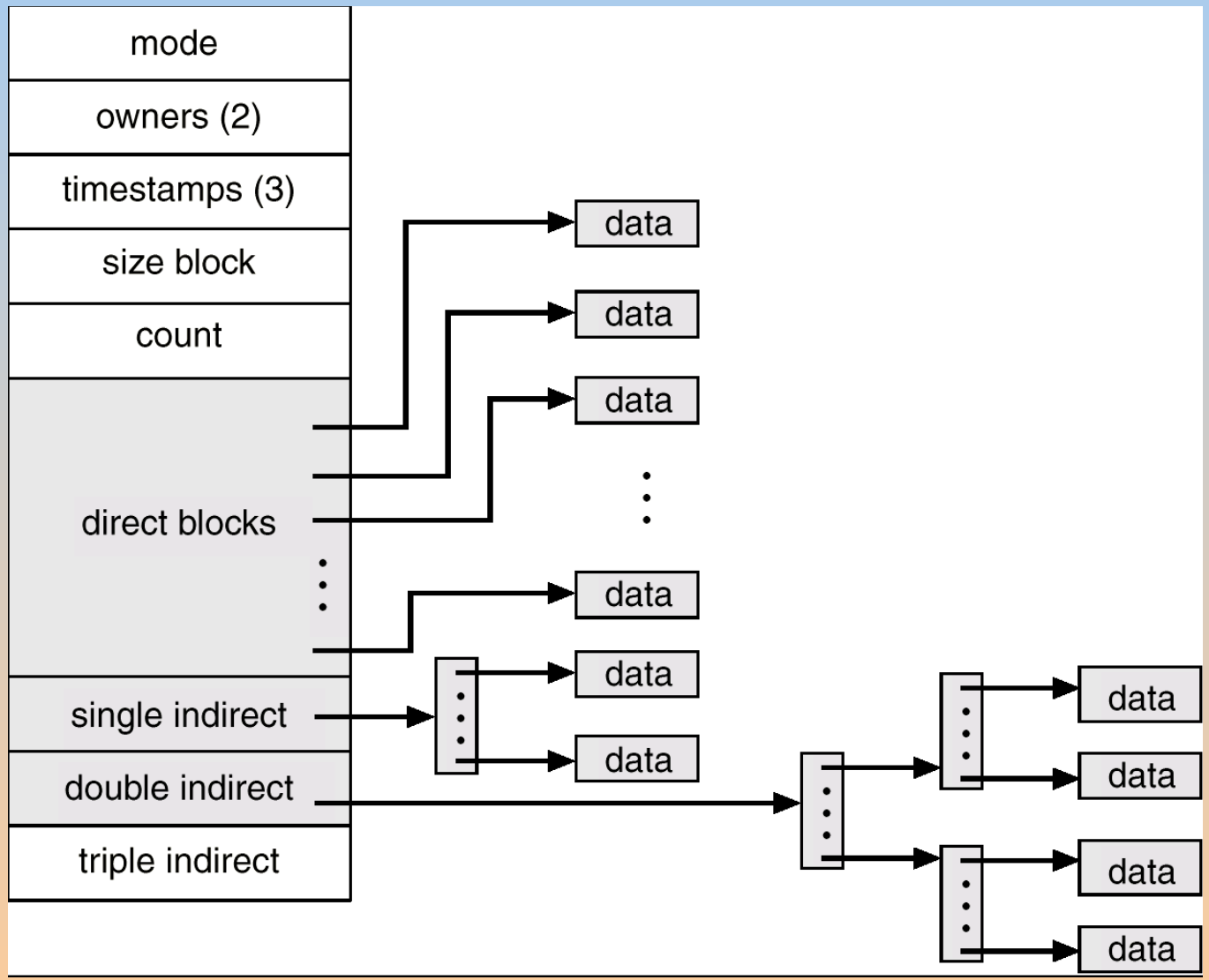


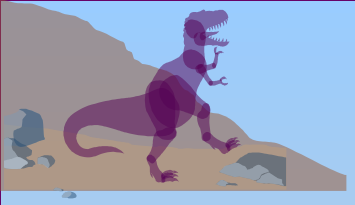
Indice a più livelli



- Si utilizza un blocco indice dei blocchi indice.
- Degradano le prestazioni, in quanto è richiesto un maggior numero di accessi al disco.

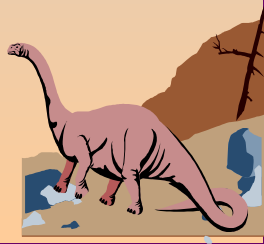
Schema combinato: Inode di UNIX (4K byte per blocco, 4 byte per puntatore: 4GB indirizzabili)





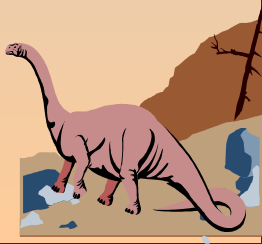
Gestione dello spazio libero

- Lo spazio su disco è limitato ed è quindi necessario riutilizzare lo spazio lasciato dai file cancellati per scrivervi, se possibile, nuovi file.
- Per tenere traccia dello spazio libero, il sistema conserva un **elenco dei blocchi liberi**
  - dove sono registrati tutti i blocchi non assegnati ad alcun file o directory.
- Per creare un file occorre cercare nell'elenco dei blocchi liberi la quantità di spazio necessaria e assegnarla al nuovo file,
- quindi rimuovere questo spazio dall'elenco dei blocchi liberi.
- Quando si cancella un file si aggungono all'elenco dei blocchi liberi i blocchi di disco ad esso assegnati.



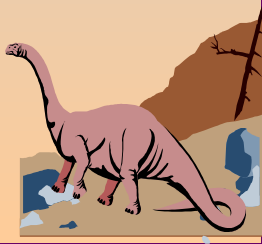
Vettore di bit

- Spesso la lista dei blocchi liberi viene implementata come una **mappa di bit**, o **vettore di bit**.
- Ogni blocco è rappresentato da un bit: se il blocco è libero il bit è 1, se il blocco è allocato il bit è 0.
- Il vantaggio di questo metodo è la sua relativa semplicità ed efficienza nel trovare il primo blocco libero o *n* blocchi liberi consecutivi nel disco.
- I vettori di bit sono efficienti solo se tutto il vettore è mantenuto nella memoria centrale,
  - soluzione non applicabile ai dischi più grandi.

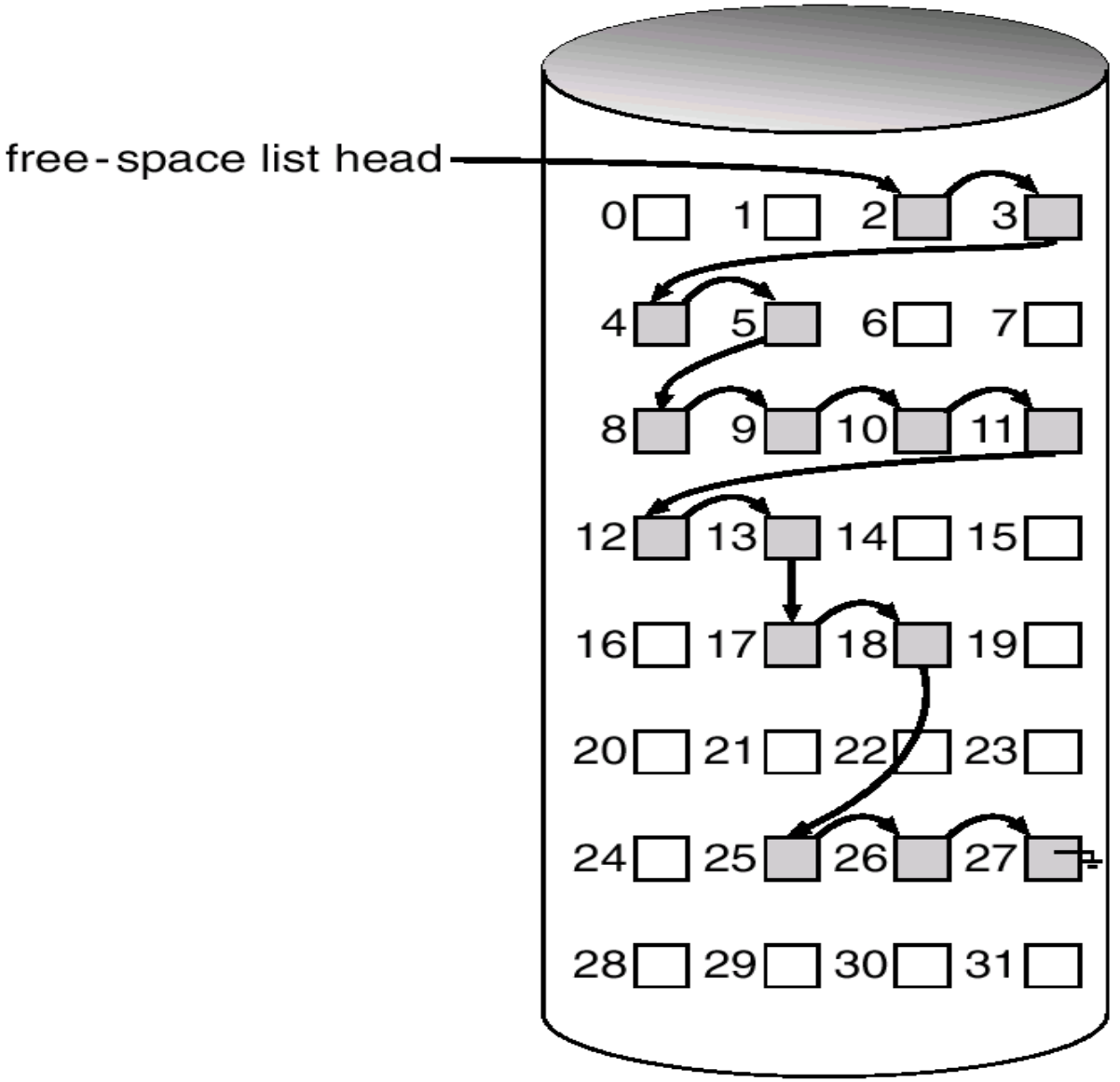


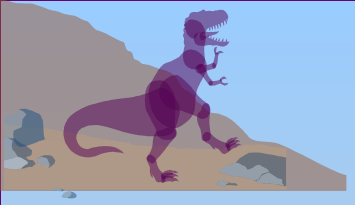
Lista concatenata

- I blocchi liberi vengono mantenuti in una lista concatenata.
- Un puntatore al primo blocco della lista (che a sua volta contiene un puntatore al secondo, e così via) viene mantenuto in una speciale locazione del disco e caricato in memoria.
- **Vantaggi**:
  - richiede poco spazio in memoria centrale
- **Svantaggi**:
  - L'allocazione di un'area di ampie dimensioni è costosa:
    - per attraversare la lista occorre leggere ogni blocco, e l'operazione richiede un notevole tempo di I/O.
  - L'allocazione di aree libere contigue è molto difficoltosa.



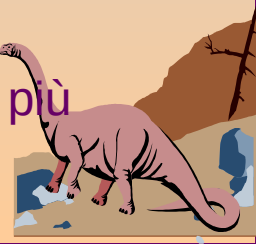
Lista concatenata dei blocchi liberi





Raggruppamento e Conteggio

- **Raggruppamento**: memorizzazione degli indirizzi di *n* blocchi liberi nel primo di questi.
- I primi n-1 di questi blocchi sono effettivamente liberi; l'ultimo blocco contiene gli indirizzi di altri n blocchi liberi, e così via.
- L'importanza di questa implementazione, è data dalla possibilità di trovare rapidamente gli indirizzi di un gran numero di blocchi liberi.
- **Conteggio**: generalmente, più blocchi contigui possono essere allocati o liberati contemporaneamente.
- Quindi, anziché tenere una lista di *n* indirizzi liberi, è sufficiente tenere l'indirizzo del primo blocco libero e il numero *n* di blocchi liberi contigui che seguono il primo blocco.
- Ogni elemento della lista dei blocchi liberi è formato da un indirizzo del disco e un contatore.
- Anche se ogni elemento richiede più spazio di quanto ne richieda un semplice indirizzo del disco,
  - se il contatore è generalmente maggiore di 1 la lista globale risulta più corta.



Efficienza

- I dischi tendono ad essere il principale collo di bottiglia per le prestazioni di un sistema essendo i più lenti tra i componenti di un calcolatore.
- L'uso efficiente di un disco dipende dalle tecniche di allocazione dello spazio su disco e dagli algoritmi di gestione delle directory.
  - Ad esempio, gli i-node di UNIX sono preallocati in una partizione.
  - Anche un disco "vuoto" impiega una certa percentuale del suo spazio per gli i-node.
  - D'altra parte, preallocando gli i-node e distribuendoli lungo la partizione si migliorano le prestazioni del file system.
- Queste migliori prestazioni sono il risultato degli algoritmi di allocazione e di gestione dei blocchi liberi adottati da UNIX.
- Questi algoritmi mantengono i blocchi di dati di un file vicini al blocco che ne contiene l'i-node allo scopo di ridurre il tempo di posizionamento.
- Anche il tipo di dati normalmente contenuti in un elemento di una directory (o di un i-node) deve essere tenuto in considerazione.

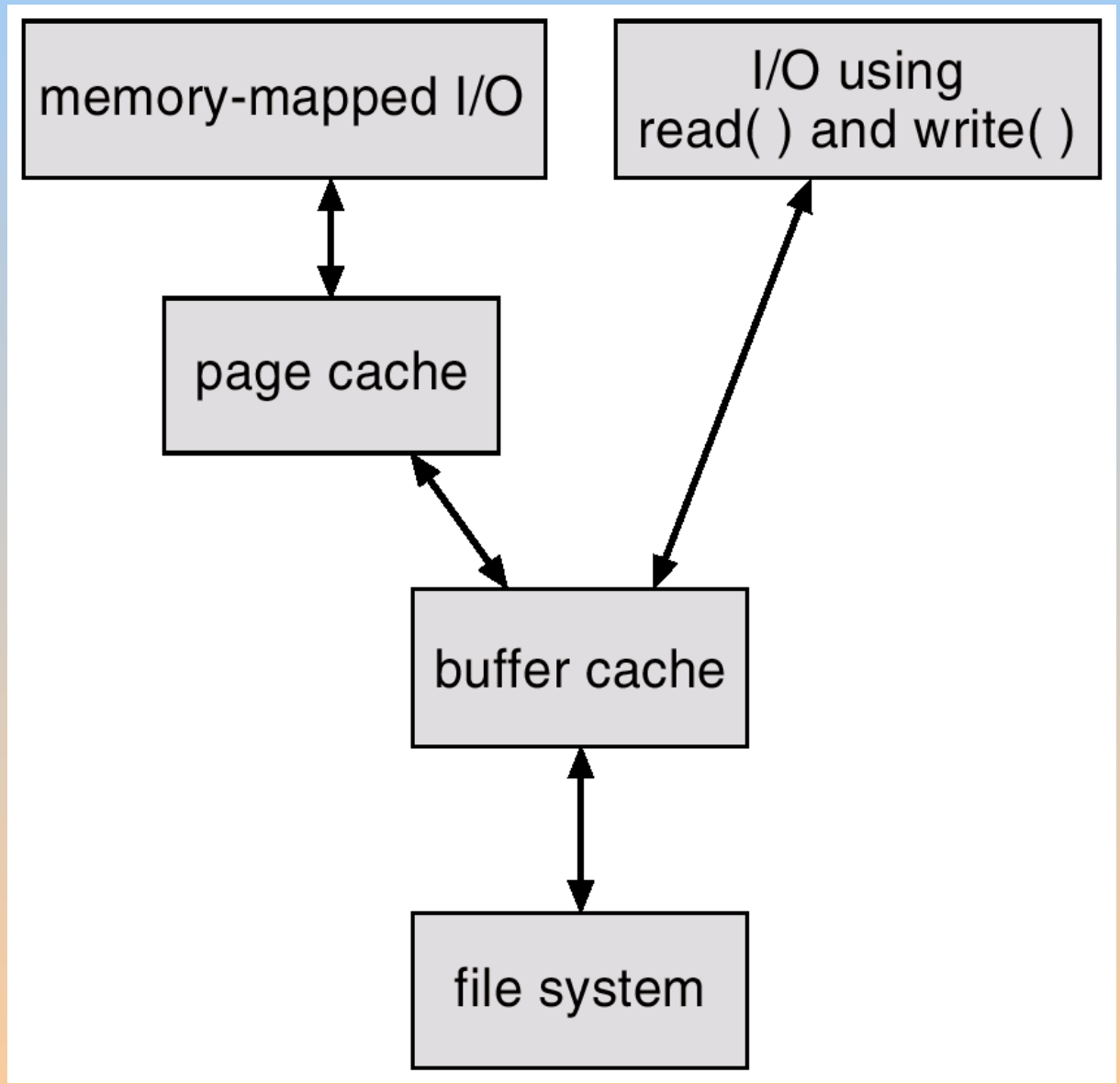
Efficienza (II)

- Solitamente viene memorizzata la *data di ultima scrittura*,
  - per fornire informazioni all'utente e per determinare se il file necessita o meno della creazione o aggiornamento di una copia di backup.
- Alcuni sistemi mantengono anche la *data di ultimo accesso* per consentire all'utente di risalire all'ultima volta che un file è stato letto.
- Il risultato del mantenere queste informazioni è che ogni volta che un file viene letto si dovrà aggiornare un campo della directory.
  - Questa modifica richiede la lettura in memoria del blocco, la modifica della sezione e la riscrittura del blocco su disco, poiché sui dischi è possibile operare solamente per blocchi (o gruppi di blocchi).
- Quindi, ogni volta che un file viene aperto in lettura, anche l'elemento della directory a esso associato deve essere letto e scritto.
- Questo requisito può risultare inefficiente per file a cui si accede frequentemente,
  - quindi al momento della progettazione del file system è necessario confrontare i benefici con i costi in termini di prestazioni.
- In generale, è necessario considerare l'influenza sull'efficienza e sulle prestazioni di ogni informazione che si vuole associare a un file.

Prestazioni

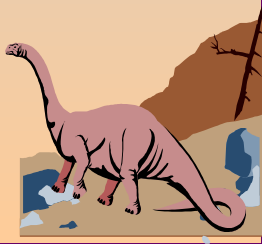
- Alcuni controllori di unità a disco hanno una memoria locale sufficiente a memorizzare un'intera traccia del disco alla volta.
- Una sezione separata della memoria centrale può essere utilizzata come **cache del disco**,
  - per tenere blocchi del disco in memoria in previsione di un loro riutilizzo entro breve tempo.
- Un'alternativa è la **cache delle pagina**,
  - una soluzione che impiega tecniche di memoria virtuale per la gestione dei dati dei file come pagine anziché come blocchi di file system,
  - l'uso degli indirizzi virtuali è più efficiente dell'uso dei blocchi fisici del disco.
- L'I/O associato alla memoria utilizza cache di pagina.
- L'I/O standard (chiamate a sistema: *read, write,* etc.) utilizza la buffer cache del disco.
- Poiche' il sistema di memoria virtuale non può interfacciarsi con la buffer cache, si deve copiare nella cache delle pagine il contenuto del file presente nella buffer cache:
  - **double caching**.

I/O senza buffer cache unificata

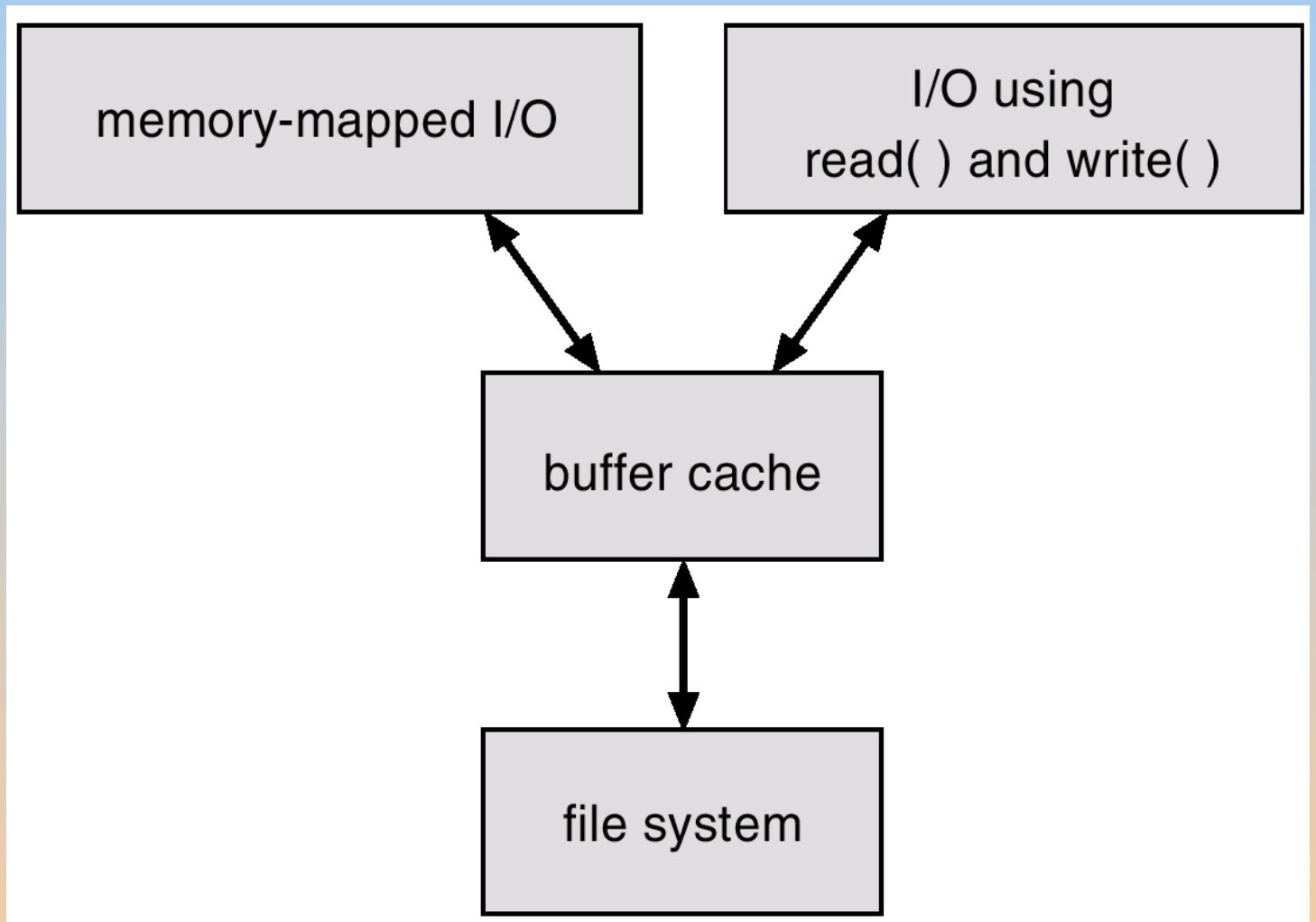


Cache unificata

- Una buffer cache unificata utilizza la stessa cache di pagina per memorizzare sia l'I/O associato alla memoria sia quello effettuato attraverso il file system.
- L'uso dei file tramite l'associazione alla memoria richiede che i file siano copiati prima nella buffer cache e quindi sottoposti a paginazione.
- Si evita così il fenomeno del *double caching*, e si permette al sottosistema di memoria virtuale di gestire i dati del file system.

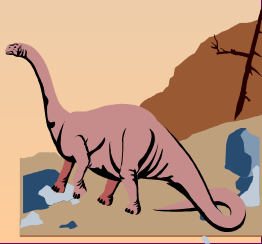


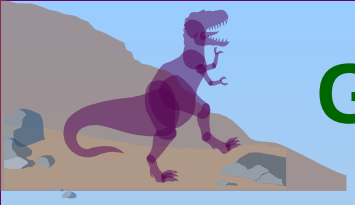
I/O con buffer cache unificata



Gestione dello spazio della cache

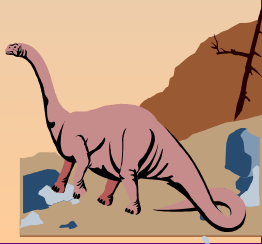
- Quando un blocco deve essere caricato in una cache già piena, un altro blocco deve esserne rimosso
  - ed eventualmente riscritto sul disco se è stato modificato dal momento del suo caricamento.
- L'algoritmo LRU è in generale ragionevole per la sostituzione dei blocchi o delle pagine..
- Alcuni sistemi ottimizzano la cache del disco adottando differenti algoritmi di sostituzione, a seconda del tipo di accesso al file.
- I blocchi di un file letto o scritto in modo sequenziale non dovrebbero essere rimpiazzati in ordine LRU,
  - poiché il blocco utilizzato più di recente verrà probabilmente riutilizzato per ultimo, o forse mai.





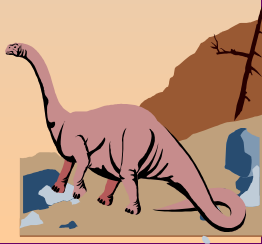
Gestione dello spazio della cache (II)

- Gli accessi sequenziali potrebbero essere ottimizzati da tecniche note come **rilascio indietro** e **lettura anticipata**.
- La prima di queste tecniche rimuove un blocco non appena si verifica una richiesta del blocco successivo;
  - i blocchi precedenti con tutta probabilità non saranno più utilizzati e quindi sprecano spazio nel buffer.
- Con la tecnica di lettura anticipata vengono letti e posti nella cache il blocco richiesto e parecchi blocchi successivi;
  - è probabile che questi blocchi verranno richiesti una volta terminata l'elaborazione del blocco corrente.
- Il recupero di questi blocchi dal disco con un unico trasferimento e la memorizzazione nella cache consentono di risparmiare una quantità di tempo considerevole.

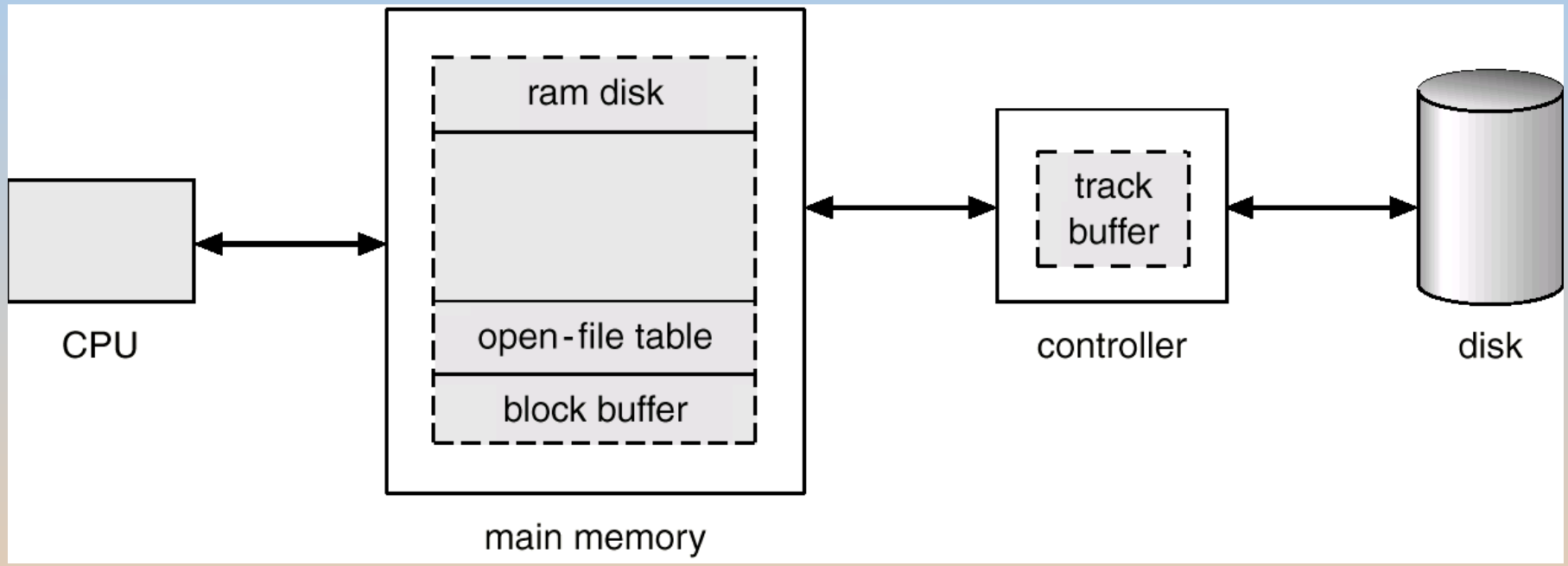


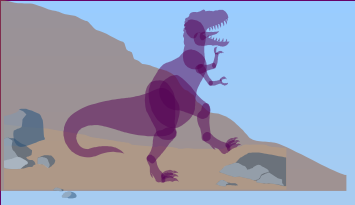
Disco RAM

- Nei personal computer viene comunemente adottata un'altra tecnica per migliorare le prestazioni.
- Una sezione della memoria viene riservata e gestita come un **disco virtuale** o **disco RAM**.
- Il driver di un disco RAM accetta tutte le operazioni standard su disco, eseguendole però in memoria invece che su un disco.
- Le operazioni su disco possono essere eseguite su disco RAM senza che gli utenti se ne accorgano, se non per la velocità elevata.
- Sfortunatamente i dischi RAM sono utili solamente come supporto temporaneo, poiché la mancanza di alimentazione o il riavvio del sistema solitamente ne cancellano il contenuto.
- In un disco RAM vengono di solito memorizzati file temporanei, come i file intermedi di compilazione.



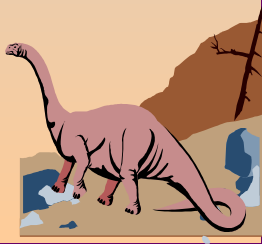
Diverse locazioni di caching del disco





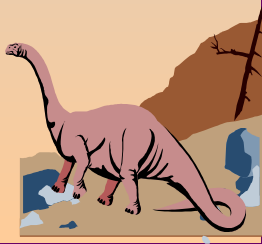
Ripristino: verifica della coerenza

- I file e le directory sono mantenuti sia in memoria centrale che su disco.
- Il verificarsi di un malfunzionamento nel sistema può comportare la perdita di dati o la loro incoerenza.
- Il **verificatore della coerenza** confronta i dati delle directory con quelli contenuti nei blocchi su disco, tentando di correggere ogni incoerenza.



Copie di riserva e recupero dei dati

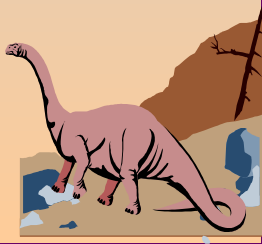
- Poiché si possono verificare malfunzionamenti e perdite di dati anche nei dischi magnetici, è necessario provvedere affinché i dati non vadano persi per sempre.
- A questo scopo è possibile utilizzare dei programmi di sistema che consentano di fare delle **copie di riserva** (**backup**) dei dati residenti su disco su altri dispositivi di memoria,
  - come floppy disk, nastri magnetici o dischi ottici.
- Il ripristino della situazione antecedente la perdita di un singolo file, o del contenuto dell'intero disco, richiederà il recupero dei dati dalle copie di riserva (**restore**).
- La gestione delle copie di riserva dipende dalla criticità dei dati:
  - Copie complete: richiedono tempo ma permettono ripristini veloci anche da perdite di dati totali
  - Copie incrementali: occupano minor spazio rispetto alle complete, permettono ripristini veloci da perdite parziali e recenti.

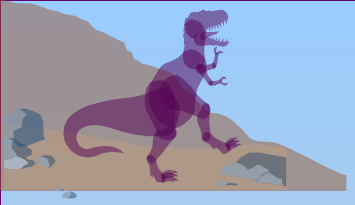


Copie di riserva e recupero dei dati (II)

- Ad esempio, se il programma di backup sa quando è stato eseguito l'ultimo backup di un file, e se la data di ultima scrittura di quel file, registrata nella directory, indica che il file da quel momento non ha subito variazioni, non sarà necessario copiare nuovamente il file.
- Quella che segue è una tipica sequenza di backup:
  - Giorno **1**. Copiatura sul supporto i backup di tutti i file contenuti nel disco; detto ***backup completo***.
  - Giorno **2**. Copiatura su un altro supporto di tutti i file modificati dal Giorno 1; si tratta di un ***backup incrementale***.
  - Giorno **3**. Copiatura su un altro supporto di tutti i file modificati dal Giorno 2.

  ....
  - Giorno **n**. Copiatura su un altro supporto di tutti i file modificati dal Giorno n-1. Ritorno al Giorno 1.





Differenze tra il file system di Unix e quello di MS-DOS

Caratteristiche	UNIX	MS-DOS
Sistema con directory gerarchiche?	Si	Si
Directory corrente?	Si	Si
Directory . e . . ?	Si	Si
Lunghezza dei nomi di file	14 o 255	8 + 3
Separatore nei nomi	/	\
“a” equivale ad “A”	No	Si
Proprietari, gruppi, protezioni?	Si	No
Concatezioni	Si	No
Attributi di file?	No	Si

File system annotati

- Gli algoritmi di ripristino basati sulla registrazione delle modifiche possono essere applicati con successo al problema della coerenza dei file system.
- I **file system annotati** (**journaling file system**) annotano tutte le modifiche ai metadati in un **giornale delle modifiche**.
- Quando le modifiche sono state annotate, le operazioni si considerano portate a termine e le chiamate di sistema ritornano.
- Il file system viene aggiornato in base al giornale in maniera asincrona, completate le modifiche al file, le annotazioni possono essere rimosse dal giornale.
- Nel caso di un crash di sistema, al ripristino del sistema sarà necessario portare a termine nel file system le modifiche annotate sul giornale.