Filesystem

Introduzione

Struttura logica

Struttura fisica

Filesystem

- **Il filesystem**
  - E' un modulo del sistema operativo che si occupa di fornire una astrazione ai meccanismi di memorizzazione di massa
  - Si basa su alcuni concetti fondamentali
    - File
    - Directory
    - Volume
- **Il concetto di file offre una visione**
  - Omogenea delle informazioni memorizzate
  - Indipendente dal tipo di dispositivo fisico su cui le informazioni vengono memorizzate
- **Un file è costituito da**
  - Un insieme di informazioni omogenee
  - Un nome simbolico
  - Un insieme di attributi
- **Un file può contenere**
  - Dati
  - Programmi
  - Riferimenti

File: Attributi

- **Ad un file sono associati alcuni attributi che ne descrivono le caratteristiche**
  - Nome
    - Nome simbolico con cui ci si riferisce ad esso
  - Tipo
    - Definisce il tipo dei dati contenuti, a volte è indicato mediante una estensione
    - Nei sistemi UNIX/Linux, distingue file ordinari da link, directory, e device
  - Locazione
    - E' un puntatore alla posizione fisica sul dispositivo
  - Dimensione
    - Dimensione dei dati espressa in bytes o blocchi
  - Protezione
    - Definisce le politiche di gestione degli accessi
  - Ora e Data
    - Indicano il momento della creazione, ultima modifica o ultimo accesso
  - Proprietario
    - Indica il nome dell'utente e del gruppo che ha creato il file

File: Operazioni

- **Sui file possono essere compiute diverse operazioni**
  - Tutte le operazioni vengono svolte da servizi di sistema operativo
  - Normalmente una libreria incapsula tali servizi, astraendone i dettagli
- **Creazione**
  - Viene aggiunto un nuovo file al file system
  - Le operazioni elementari richieste sono
    - Allocazione
    - Creazione del nuovo descrittore del file
    - Aggiunta del descrittore al file system
- **Scrittura**
  - Aggiunge dati a ad un file già creato
  - Per scrivere dati su un file è necessario fornire
    - Il nome del file
    - I dati da scrivere
  - Il file system mantiene un puntatore alla posizione in cui deve essere effettuata la scrittura successiva

File: Operazioni

- **Lettura**
  - Preleva dati da un file già creato
  - Per leggere dati da un file è necessario fornire
    - Il nome del file
    - Un puntatore alla zona di memoria destinata a contenere i dati
  - Il file system mantiene un puntatore alla posizione in cui deve essere effettuata la lettura successiva

- **Riposizionamento**
  - Sposta la posizione dei puntatori di lettura e di scrittura
  - Le operazioni consentite dipendono dal tipo di accesso del file
  - Si possono avere due situazioni
    - Il sistema operativo mantiene un solo puntatore valido per la lettura e per la scrittura
    - Il sistema operativo mantiene due puntatori distinti per lettura e scrittura

File: Operazioni

- **Cancellazione**
  - Elimina un file
  - Per eliminare un file è necessario specificarne il nome
  - Le operazioni necessarie sono
    - Deallocazione dello spazio sul dispositivo fisico
    - Aggiunta dello spazio deallocato alla lista dello spazio disponibile sul dispositivo
    - Rimozione del descrittore del file dal file system
- **Tutte le operazioni descritte richiedono l'accesso ad un file**
  - Il file system deve cercare sul dispositivo fisico il file
    - Per rendere più efficiente la ricerca, il file system mantiene una tabella dei file in uso
- **A tale scopo il sistema operativo fornisce due sevizi di base**
  - Open
    - In base al nome individua la posizione del file sul disco e copia il descrittore nella tabella dei file
  - Close
    - Sulla base di un identificatore localizza il descrittore del file nella tabella
    - Elimina il descrittore dalla tabella dei file in uso

Protezione

- **I dati memorizzati nei file di un file system necessitano di protezione**

- **Protezione da danni fisici:**
  - Malfunzionamenti dei dispositivi
  - Danni meccanici e/o elettrici
  - Soluzione: backup e mirroring

- **Protezione da accessi impropri:**
  - Riservatezza
  - Modifica o eliminazione accidentale di dati importanti
  - Soluzione: definizione di una politica di accesso

- **Con il termine protezione ci si riferisce**
  - Alla definizione di una politica di accesso
  - Alla relativa implementazione

Protezione

- **Alcune banali politiche di accesso sono:**
  - Ogni utente accede solo ai propri file
    - Si tratta di una scelta limitante, ad esempio per i gruppi di lavoro
  - Ogni utente accede a tutti i file
    - E' assente una politica di accesso

- **La soluzione consiste nell'accesso controllato**

- **Si definiscono regole di accesso ai file sulla base di:**
  - Identità e gruppo di lavoro dell'utente
  - Proprietà dei file

- **Tali regole dipendono dal tipo di operazione richiesta**
  - Lettura
  - Scrittura o eliminazione
  - Esecuzione o lista
  - Aggiunta

Protezione

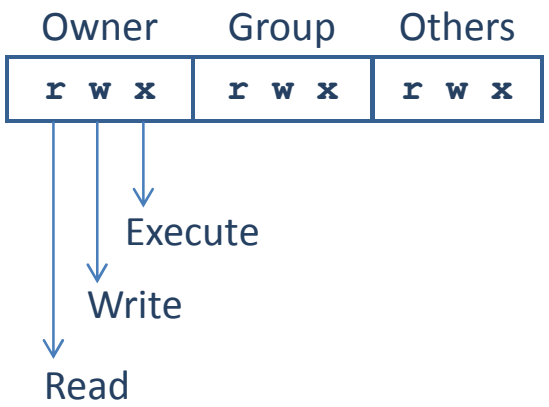
- **Lista d'accesso**
  - L’accesso ed le operazioni consentite dipendono dall'identità dell’utente
- **Ad ogni file è associata una lista di accesso che indica**
  - Quali operazioni sono consentite
  - A quali utenti
- **Quando una operazione viene richiesta**
  - Il sistema operativo controlla la lista di accesso per verificare se
    - Il richiedente è contemplato
    - Il richiedente ha il permesso di compiere quel tipo di operazione
- **Questa soluzione presenta alcuni svantaggi**
  - Le liste di accesso possono essere di dimensioni notevoli
  - Le liste di accesso devono essere create e mantenute per ogni file
  - Il tempo di accesso ad un file viene prolungato

Protezione

- **I sistemi UNIX/Linux**
  - Implementano un meccanismo di protezione versatile e semplice al tempo stesso
  - Supportano comunque anche il meccanismo delle liste di accesso
  - Tale meccanismo si basa sui concetti di identità, proprietà e operazione
- **Gli utenti sono identificati in base a**
  - Username: Identificativo dell'utente
  - Group: Identificativo di gruppo, condiviso da più utenti
- **Dal punto di vista della proprietà gli utenti sono raggruppati in tre classi**
  - Owner: Il proprietario del file
  - Group: I membri del gruppo del proprietario del file
  - Others: Tutti gli utenti
- **Le operazioni sono raggruppate in tre classi**
  - Read: Lettura, copia
  - Write: Scrittura, modifica, eliminazione
  - Execute: Esecuzione

Protezione: UNIX/Linux

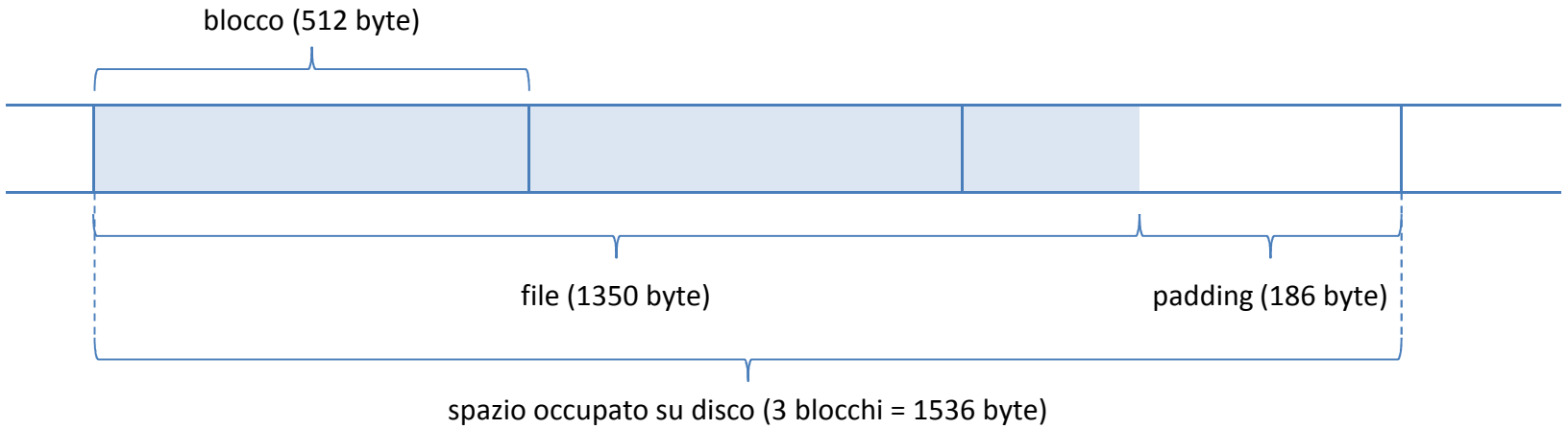
- **Ad ogni file sono associati:**
  - Owner
  - Group
  - Control access list
- **La control access list è formata da tre gruppi di bit:**
  - Ogni gruppo di bit si riferisce ad una delle tre classi di utenti
  - Ogni bit del gruppo si riferisce ad una delle tre operazioni



Owner	Group	Others
r w x	r – x	r – –
1 1 1	1 0 1	1 0 0
7	5	4

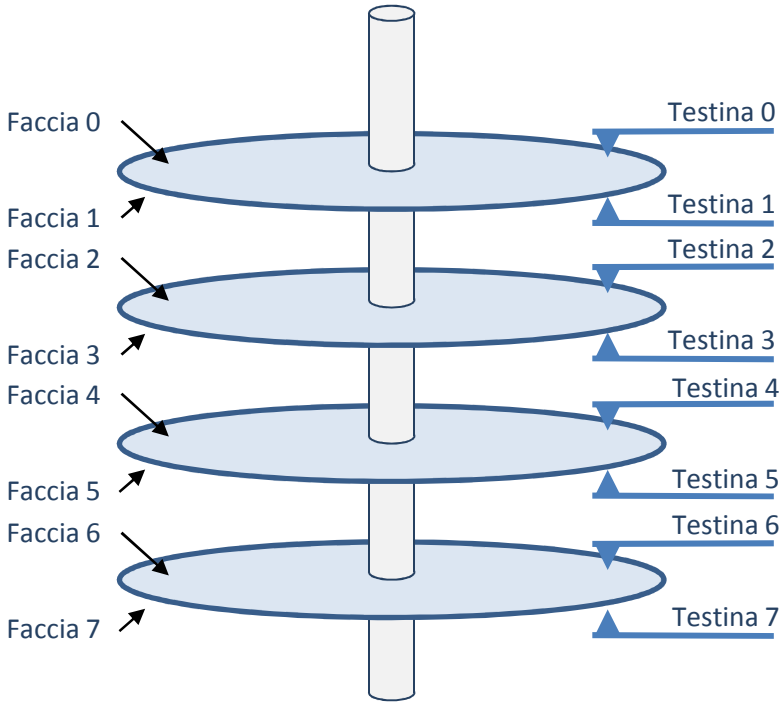
Struttura fisica

- **Un file system ha una**
  - Struttura logica
  - Struttura fisica
- **Struttura logica**
  - I dati sono organizzati in file, directory e volumi
    - Il programmatore accede al filesystem secondo la struttura logica
- **Struttura fisica**
  - I dati sono organizzati in una sequenza di blocchi
    - Di lunghezza fissa (tipicamente da 32 a 4096 byte) e dipendente dal dispositivo
  - La dimesnione fissa dei blocchi provoca un fenomendo detto "frammentazione interna"

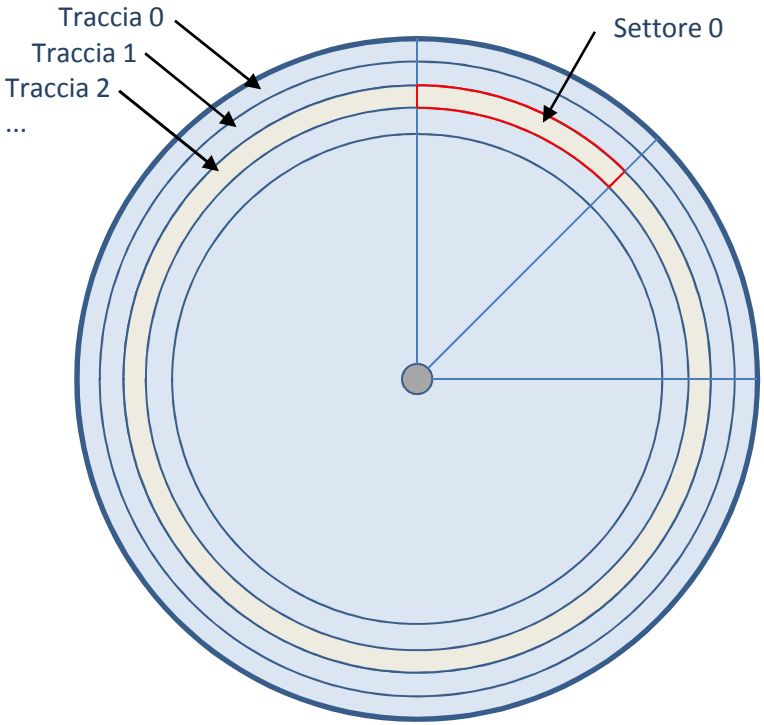


Struttura fisica

- **Un hard disk**
  - E' costituito da alcuni dischi impilati in modo coassiale
  - Ogni disco ha due faccie
  - Per ogni faccia si ha una testina di lettura scrittura

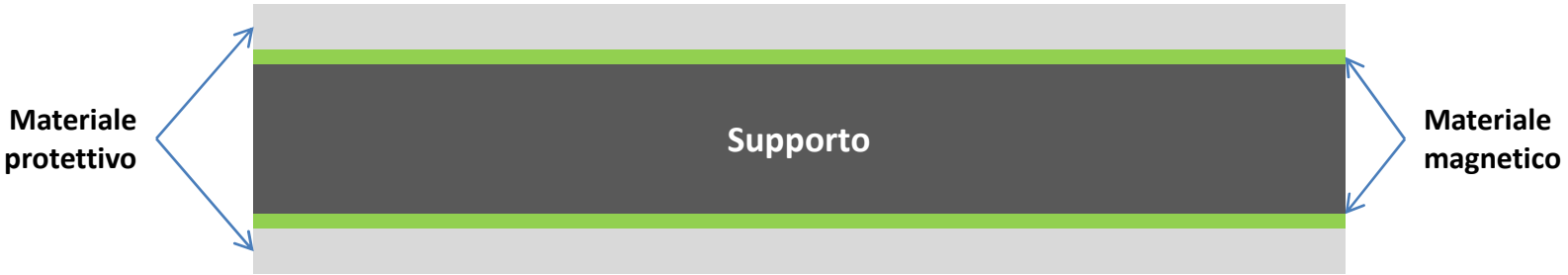


- **Ogni faccia è organizzata**
  - In traccie concentriche
  - Ogni traccia è divisa in settori
  - Ogni settore contiene un blocco di dati



Struttura fisica

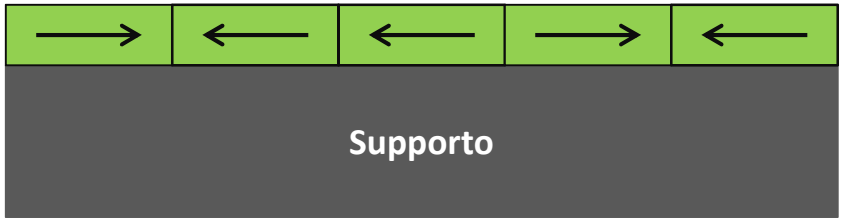
- **Ogni disco è formato da diversi strati**
  - Un supporto in alluminio, vetro, ceramica o materiali più complessi
    - Molto rigido
    - A bassissima deformabilità
    - Spessore dell'ordine del millimetro
  - Uno strato di materiale magnetico
    - E' il materiale in grado di memorizzare l'informazione binaria
    - Spessore dell'ordine del 10-20nm
  - Uno strato di protezione in carbonio
    - Previene danneggiamenti allo stato magentico
    - Spessore dell'ordine di pochi micron



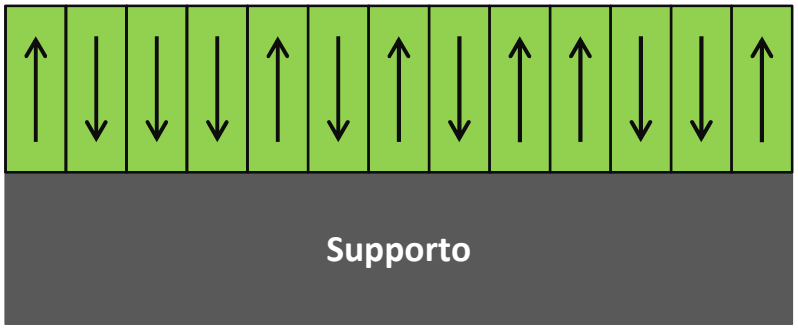
Struttura fisica

- **Ogni traccia del disco**
  - E' costituita da una sequenza di piccole regioni dette "domini magnetici"
  - Ogni regione
    - Costituisce un dipolo magnetico
    - Genera un campo magnetico
  - La polarità del campo magnetico determina il valore logico del bit
  - Si hanno due possibili orientamenti dei dipoli magnetici
    - Orientamento longitudinale: Minore densità, maggiore affidabilità
    - Orientamento verticale: Maggiore densità, mionore affidambilità

Orientamento Longitudinale



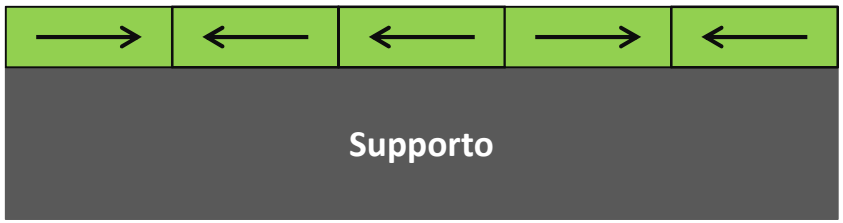
Orientamento Verticale



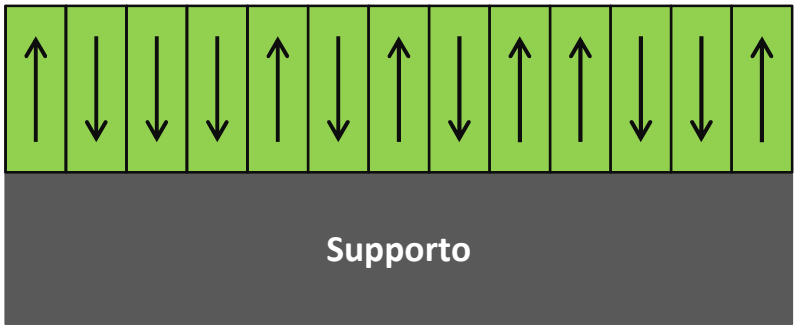
Struttura fisica

- **Ogni settore del disco**
  - Contiene tre sezioni logiche
    - Un preambolo, con informazioni di allineamento e sul settore
    - La parte di dati veri e propri
    - Un codice per il rilevamento e la correzione degli errori
  - Al crescere della densità
    - Aumenta
    - Anche i singoli byte o piccoli gruppi di byte sono corredati di un semplice codice per il rilevamento e (più raramente) la correzione degli errori

Orientamento Longitudinale



Orientamento Verticale



Organizzazione

- **I moderni dispositivi di memorizzazione di massa**
  - Consentono di salvare milioni di file
  - Consentono la condivisione "logica" di dati tra più file
  - Consentono la condivisione "fisica" di dati tra più file

- **Si rende necessaria una strutturazione efficiente**
  - In termini di utilizzo delle risorse (spazio dul dispositivo)
  - In termini di semplicità di accesso

- **Un file system è organizzato secondo una struttura gerarchica**
  - Volume Realizza il cocetto di "disco logico"
  - Directory E' un indice di un gruppo di file
  - File: Contiene effettivamente i dati

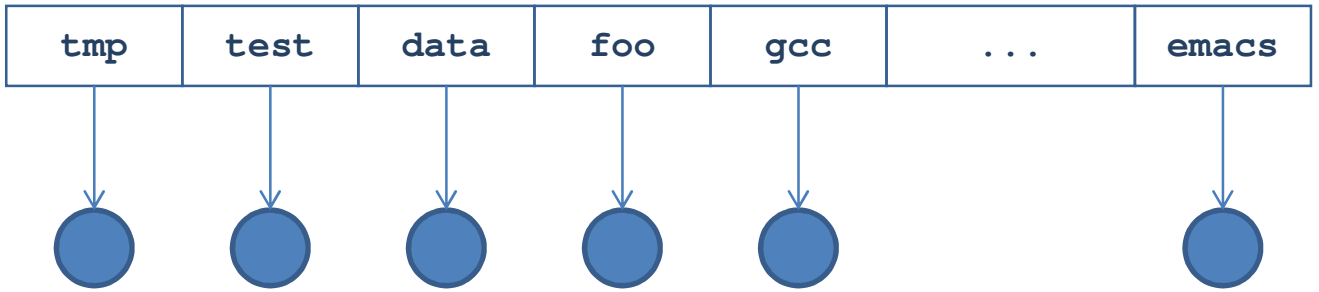
- **Non tutti i file system dispongono di partizioni**
  - Si pensi ai filesystem per memorie allo stato solido (USB sticks, SD cards, ...)
  - L'elemento fondamentale e indispensabile per l'organizzazione dei file è la directory

Directory

- **Si è detto che la directory è di fatto un "indice"**
- **Sulle directory è possibile compiere le seguenti operazioni**
  - Ricerca di un file
    - Recupero delle informazioni su un file (descrittore) sulla base del nome
  - Creazione di un file
    - Aggiunta di un nuovo descrittore di file
  - Rimozione di un file
    - Eliminazione di un descrittore di file
    - Il descrittore deve essre prima individuato tramite una operazione di ricerca
  - Elenco dei file
    - Produce l'elenco dei nomi ed eventualmente altre informazioni relative ai file memorizzati
  - Cambiamento del nome di un file
    - Modifica del nome di un file già presente nella directory
    - Il descrittore deve essre prima individuato tramite una operazione di ricerca
  - Cambiamento delle proprietà di un file
    - Modifica di uno o più campi del descrittore di un file già presente nella directory
    - Il descrittore deve essre prima individuato tramite una operazione di ricerca

Directory a singolo livello

- **Tutti i file sono contenuti in una sola directory**
- **Questa struttura è molto semplice ma presenta alcuni problemi:**
  - Nel caso di molti file
    - E' difficile garantire che tutti i nomi siano diversi
    - Le dimensioni della directory diventano grandi e la ricerca risulta più lenta
  - Non è possibile separare i file associati a utenti diversi



Directory a due livelli

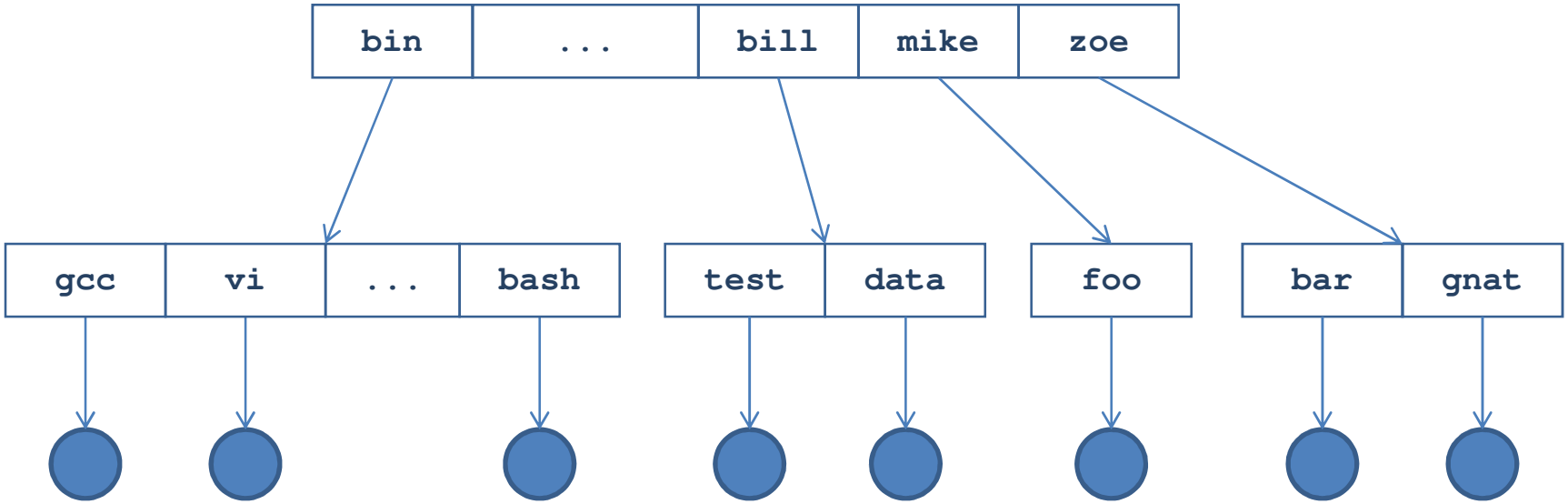
- **Si hanno due livelli di directory**
  - Directory principale (root)
    - Contiene un elenco di directory, tipicamente una per ogni utente
  - Directory utente (home)
    - Contiene i file di un singolo utente

- **Gestione**
  - I singoli utenti vedono e gestiscono solo i file nella propria home directory
  - La gestione della root directory è affidata ad un amministratore del sistema

- **Accesso**
  - Quando un utente richiede l'accesso ad un proprio file
    - Il sistema operativo ne cerca il nome nella home directory dell'utente
  - Per accedere ai file di altri utenti si utilizza il "path"name
    - La composizione del nome dell'utente e del nome del file

Directory a due livelli

- **Spesso gli applicativi sono accessibili a tutti gli utenti**
  - A tale scopo esiste una directory specifica
  - I file eseguibili vengono
    - Dapprima cercati nella directory dell'utente
    - Quindi, se la ricerca fallisce, nella directory degli applicativi



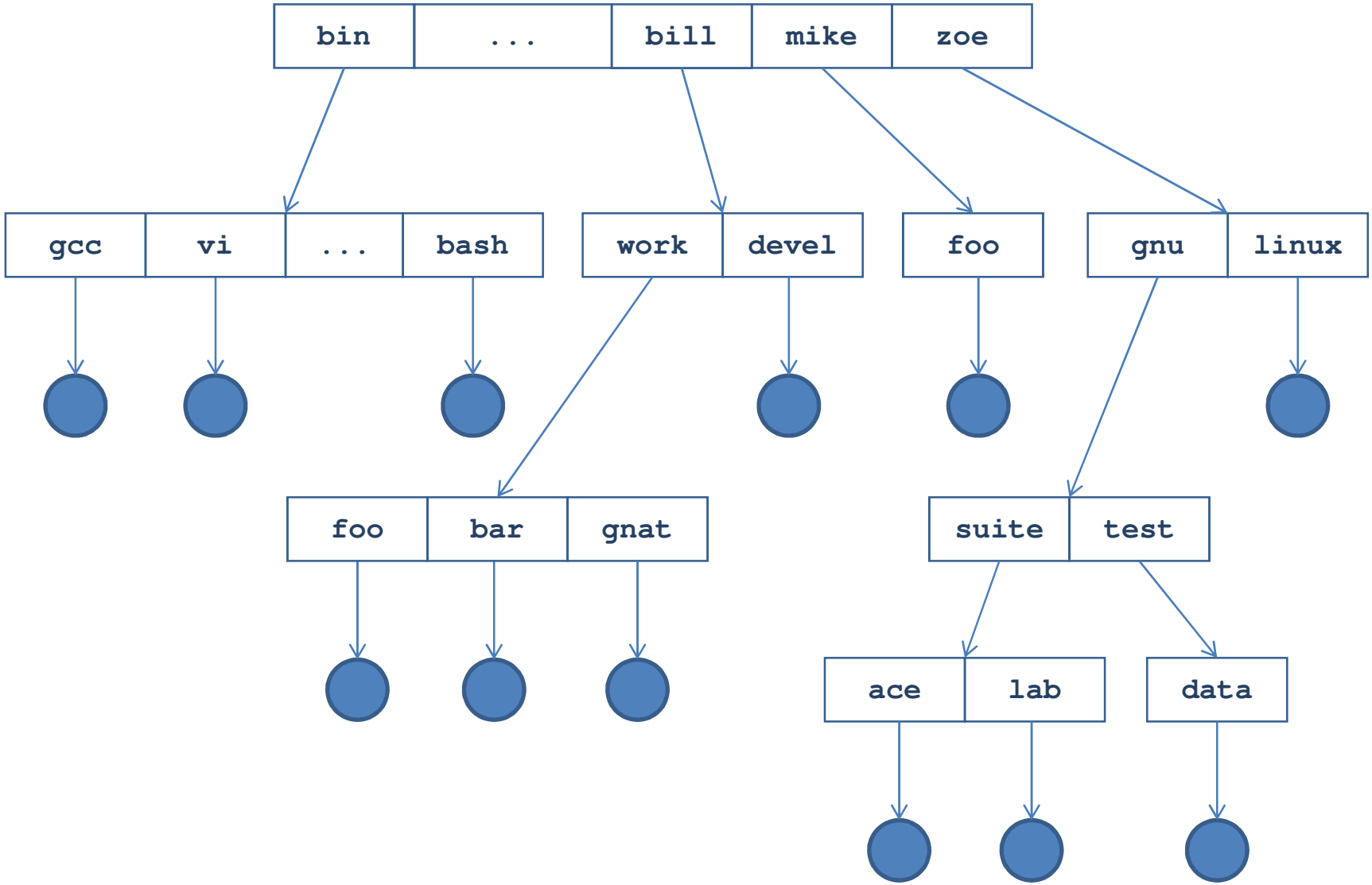
Directory ad albero

- **E' una estensione del caso precedente**
  - Ad un numeo arbitrario di livelli

- **Un utente accede ai propri file**
  - Attraverso il pathname del file

- **Per semplificare l'accesso ai file**
  - Viene definito il concetto di current working directory o CWD
  - La CWD è la directory corrente
    - Indica il punto di partenza da cuie deve essere considerato il pathname
  - I file indicati con il solo nome
    - Vengono cercati nella directory corrente
  - Se un file non è nella directory corrente:
    - Si usa il suo pathname, oppure
    - Si cambia la directory corrente fino a raggiungere quella che lo contiene, quindi si usa il nome

Directory ad albero



Directory a grafo aciclico

- **Utenti diversi vogliono condividere uno o più file**
  - Per semplificare l'accesso gli utenti vorrebbero vedere i file condivisi nella propria home
  - Esistono due soluzioni con caratteristiche differenti
    - Duplicazione
    - Riferimento

- **Duplicazione**
  - I file condivisi vengono duplicati nelle home dei diversi utenti
  - Ciò comporta spreco di spazio e problemi di allineamento del contenuto
    - Se un utente modifica un file, ne modifica solo la propria "copia locale"

- **Riferimento:**
  - Viene copiato solo il descrittore del file
    - Gli utenti accedono di fatto allo stesso file
    - Non si hanno problemi di allineamento poiché i dati sono unici
  - La gestione dei riferimenti richiede un sistema operativo più complesso

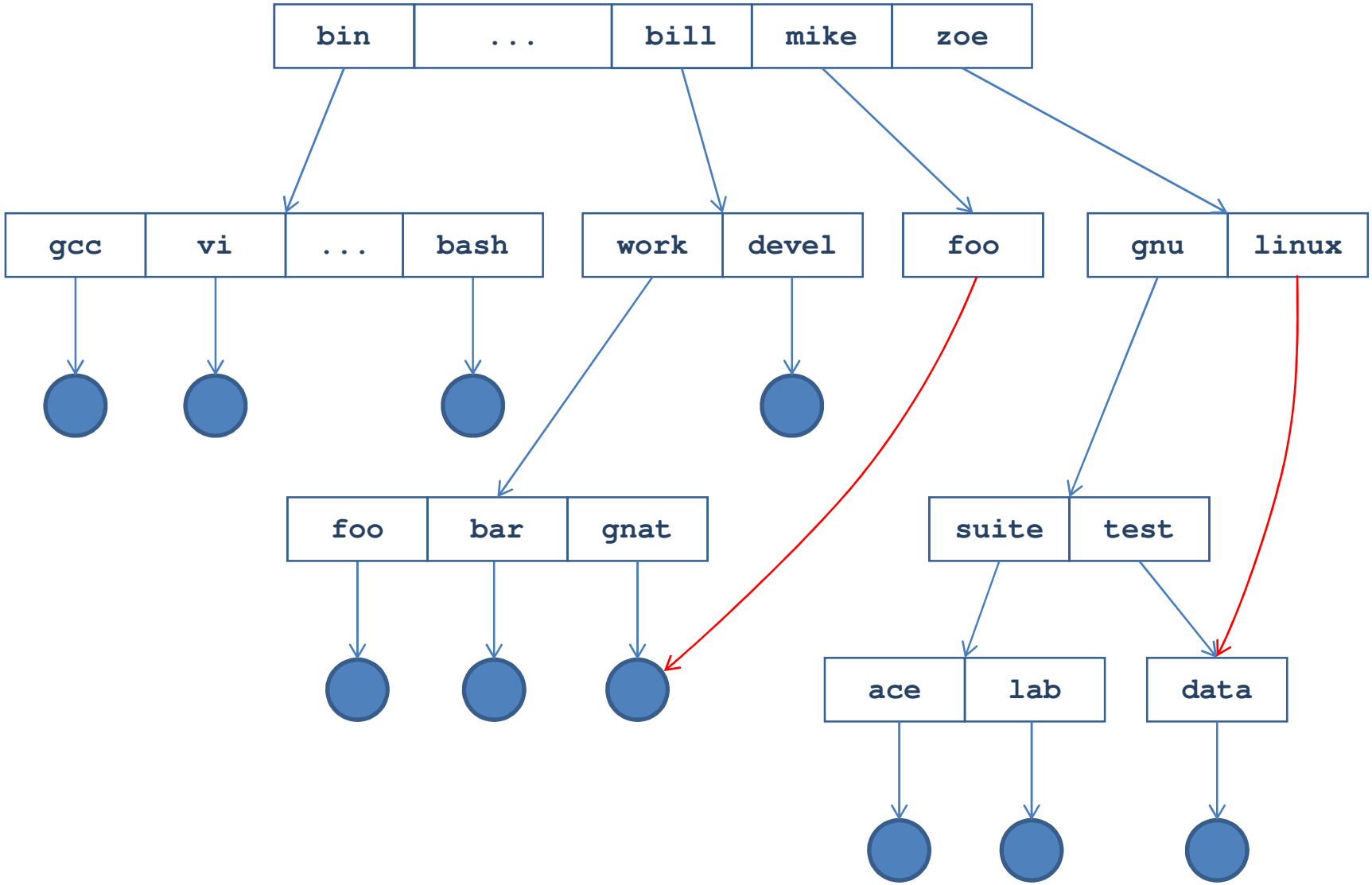
Directory a grafo aciclico

- **Soluzione migliore**
  - Riferimenti o "link"

- **Tale struttura comporta una maggiore complessità del sistema operativo**
  - Deve garantire che il grafo risultante si aciclico
  - Deve gestire coerentemente la rimozione di un file
    - Come trattare i riferimenti a quel file?
  - Deve gestire problemi di concorrenza
    - Utenti differenti possono accedere simultaneamente allo steso file
    - Si deve prevedere un meccanismo di gestione degli accessi

- **Una implementazione dei riferimenti sono link di UNIX/Linux**
  - Soft link
    - Quando un file viene rimosso i riferimenti rimangono ma puntano ad un file non esistente
  - Hard link
    - Un file viene rimosso effettivamente solo quando tutti i riferimenti ad esso sono stati eliminati

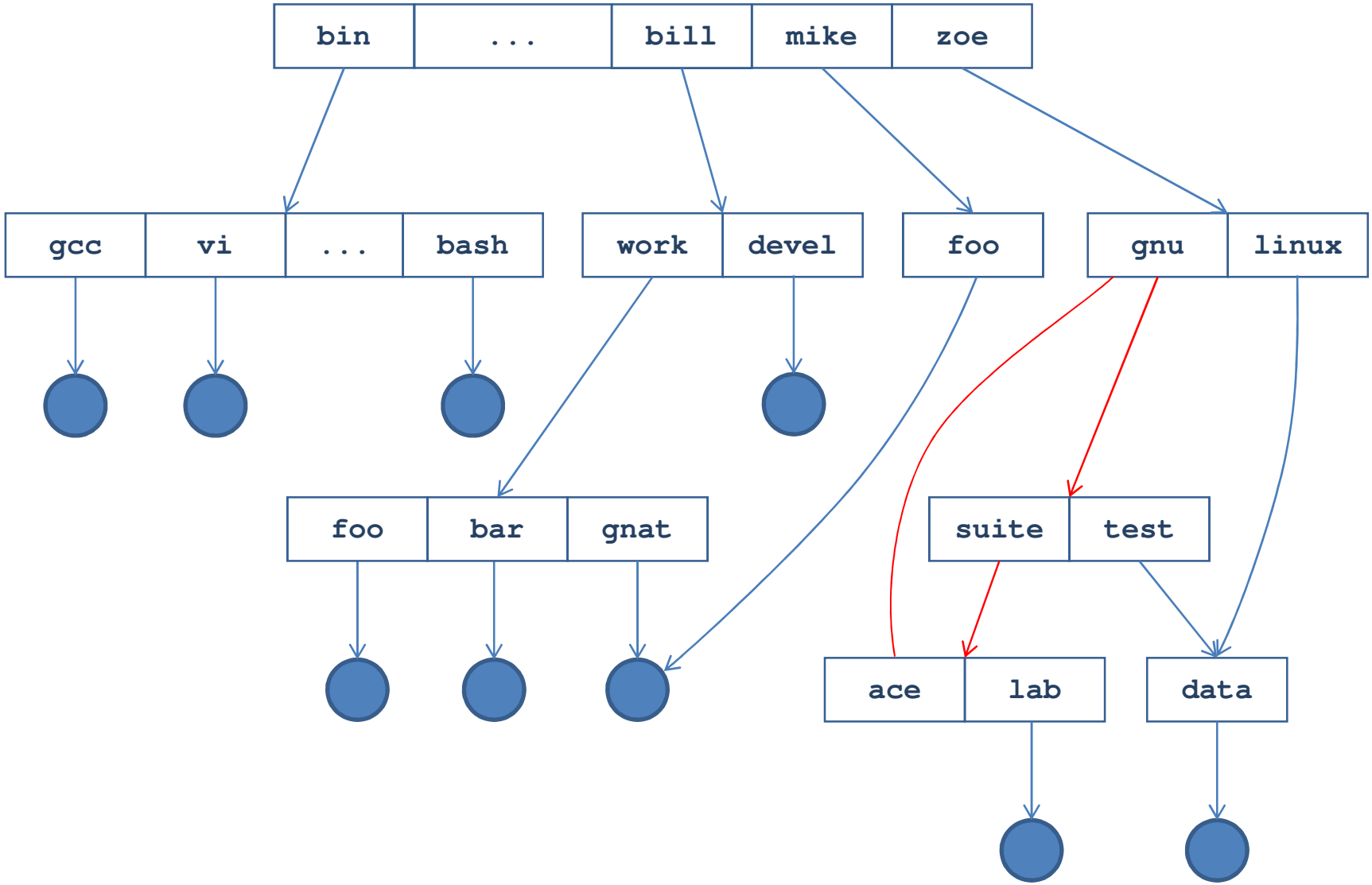
Directory a grafo aciclico



Directory a grafo generale

- **Una ulteriore estensione**
  - Consiste nel rimuovere il vincolo di aciclicità del grafo
  - Possono dunque esistere riferimenti circolari
- **Tale struttura**
  - Risulta estremamente flessibile
  - Comporta una ulteriore complicazione del sistema operativo
- **Il sistema operativo deve gestire i seguenti aspetti**
  - Nella ricerca di un file deve evitare di entrare in un loop
    - Gli algoritmi di visita di un grafo generale sono più complessi
  - Il problema della rimozione di un file diviene più complesso
- **Una soluzione semplice ma efficace a questi problemi**
  - Consiste nel limitare la profondità di un path name, ovvero il numero di directory che possono comporre un pathname
  - Nel caso di ricerca ciclica il pathename cresce continuamente
    - Ad un certo punto si raggiunge il limite e la ricerca fallisce

Directory a grafo aciclico



Implementazione

- **L'implementazione del file system**
  - Permette all'utente di disporre dei servizi descritti sino a questo punto

- **Il sistema operativo fornisce una visione logica del filesystem fornendo**
  - File, directory, link
  - Attributi e politiche di accesso
  - Operazioni
  - Mapping del file system logico sui dispositivi fisici

- **Il file system, nel senso di modulo del sistema operativo, è strutturato a livelli**
  - Ogni livello usa le funzioni fornite dal livello inferiore
  - Ogni livello realizza e fornisce funzioni al livello superiore
  - Il livello più basso è costituito dai dispositivi hardware

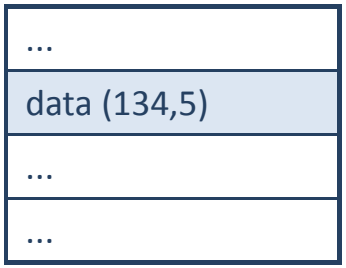
Implementazione

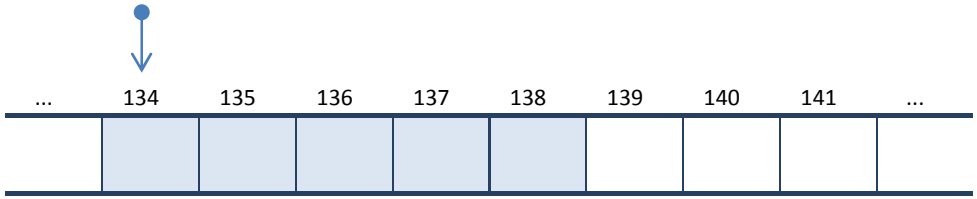
- **I livelli in cui viene spesso stratificato il file system sono i seguenti**

Applicazione	Richiedono funzioni al file system logico.
File System Logico	Sulla base del nome di un file e dell'organizzazione delle directory, genera richieste al modulo per l'organizzazione dei file. Legge i descrittori dei file restituendo la posizione.
Modulo di Organizzaione dei file	Conosce l'organizzazione logica e fisica. Traduce le richieste logiche in richieste fisiche verso il file system di base. Genera il numero assoluto di un blocco dato il suo numero relativo all' inizio del file.
File System Fisico	Invia comandi generali alla parte di controllo dell'I/O. Dato il numero di blocco assoluto genera informazioni specifiche quali faccia, settore, traccia, blocco fisico.
Controllo dell'I/O	Generano i segnali di controllo a partire dai comandi ricevuti. Tali programmi prendono il nome di driver.
Dispositivi	Eseguono i comandi richiesti attraverso i driver

Allocazione Contigua

- **I blocchi di un file sono adiacenti**
  - Un file di n blocchi è memorizzato nelle posizioni adiacenti b, b+1, …,b+n-2, b+n-1
  - Un descrittore deve indicare solo la coppia (b,n)
  - I tempi d'accesso sono contenuti poichè
    - Accesso a due blocchi successivi b e b+1 raramente richiede lo spostamento della testina in quanto hanno una probabilità elevata di trovarsi sulla stessa traccia
    - L'accesso al blocco logico i-esimo comporta l'accesso diretto al blocco fisico b+i
- **Problema**
  - Allocazione spazio per un nuovo file





Allocazione Contigua

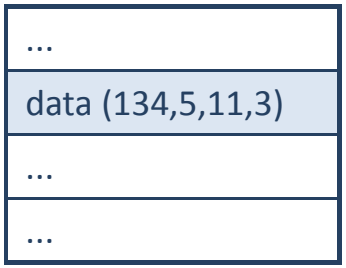
- **Dovendo creare un nuovo file di m blocchi**
  - E' necessario individuare una porzione di disco costituita da almeno m blocchi contigui
- **Si usano tre politiche:**
  - First-Fit
    - La prima zona, di almeno m blocchi, viene usata
  - Best-Fit
    - La zona più piccola, di almeno m blocchi, viene usata
  - Worst-Fit
    - La zona più grande, di almeno m blocchi, viene usata
- **Le tecniche migliori**
  - Sono le prime due
  - In particolare il metodo first-fit risulta più veloce
- **L'allocazione contigua**
  - Crea, nel tempo, zone inutilizzate di piccole dimensioni
  - Tali zone hanno una bassa probabilità di contenere un file
  - Questo fenomeno viene detto frammentazione esterna

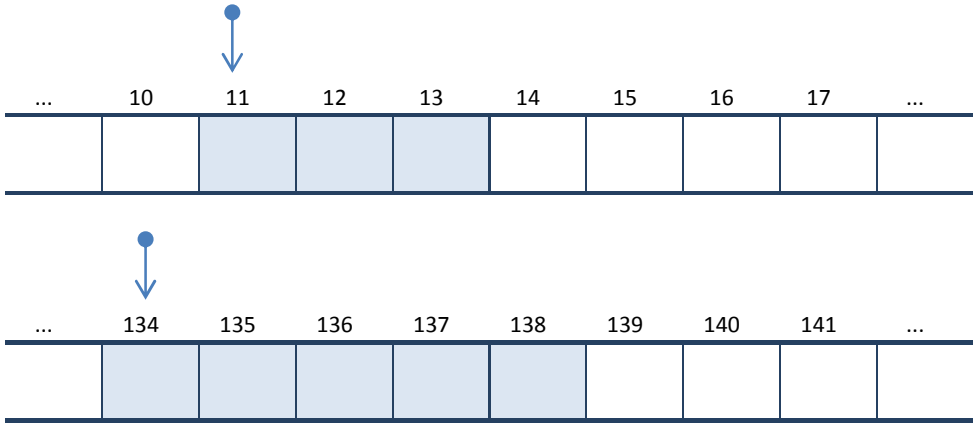
Allocazione Contigua

- **Una soluzione consiste nella compattazione dei dischi**
  - I file su un disco vengono letti e memorizzati temporaneamente altrove
    - Su un secondo disco, in una porzione libera del disco in esame o in memoria centrale
- **Si procede come segue**
  - Il disco originale viene cancellato
    - Completamente, o più spesso una porzione alla volta
  - I file vengono riscritti in sequenza, eliminando gli spazi vuoti
- **Questa operazione**
  - È molto costosa in termini di tempo
  - Deve essere compiuta con una certa frequenza
- **Una soluzione migliore**
  - Memorizzazione di un file in due zone differenti
  - Ogni zona formata da blocchi contigui
  - La zona aggiuntiva prende il nome di "extent"
  - Ancora necessari algoritmi per la ricerca di spazio disponibile

Allocazione Contigua

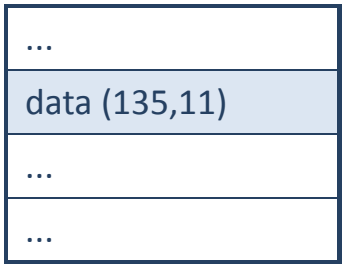
- **Un descrittore di file indica (b, n1, e, n2):**
  - b base della sezione principale
  - n1 dimensioni della sezione principale
  - e base dell'extent
  - n2 dimensione dell'extent

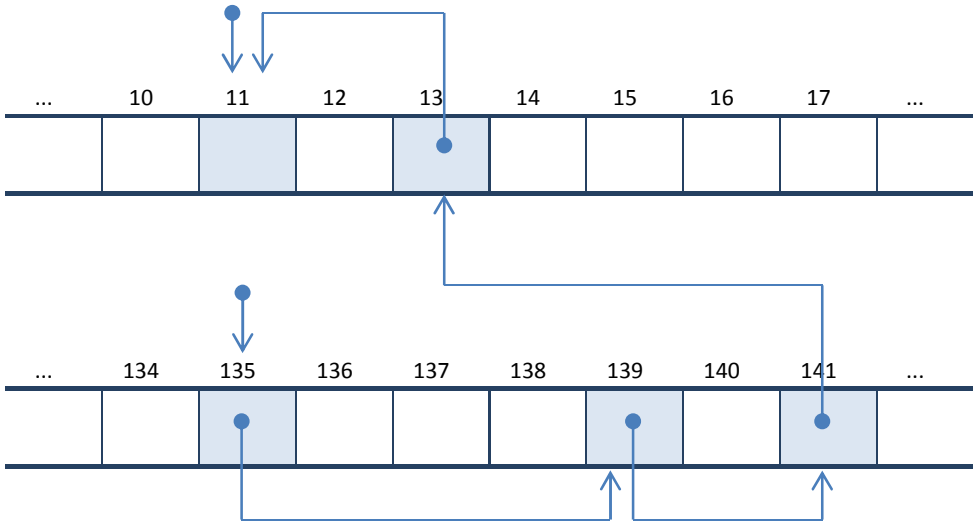




Allocazione Concatenata

- **L'idea dell'uso di un extent**
  - Può essere estesa a ad un numero maggiore di estensioni
- **In questo modo**
  - Un file è costituito da una sequenza di blocchi in cui:
  - Il descrittore contine il riferimento al primo ed all'ultimo blocco
  - Ogni blocco contiene
    - I dati
    - Un riferimento al blocco successivo





Allocazione Concatenata

- **Questa soluzione**
  - Risolve tutti i problemi tipici della allocazione contigua
  - Non si ha frammentazione esterna
- **In ogni blocco**
  - Una parte è dedicata a contenere i dati
  - Una parte è dedicata a contenere un riferimento al blocco successivo

- **La memorizzazione dei riferimenti**
  - Riduce lo spazio disponibile per i dati
  - Considerimao un esempio
    - Blocchi di 512 byte e riferimenti di 4 byte
    - Dimensione del disco 512 x 2^{32} = 2Tbyte
    - Spreco di spazio pari allo 0.78% del disco, ovvero circa 16Gbyte

Allocazione Concatenata

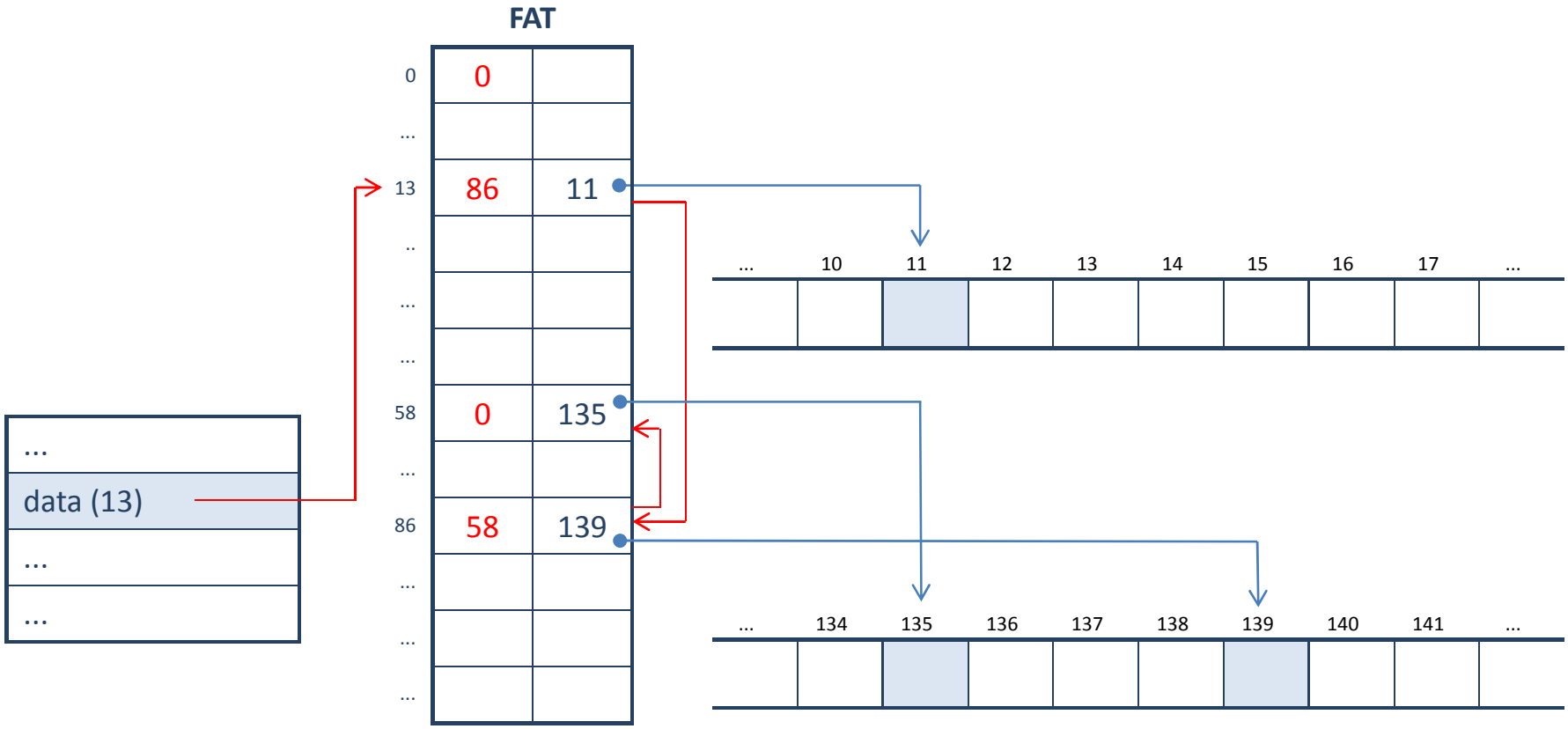
- **Anche questo metodo presenta alcuni svantaggi**
  - Per l'accesso casuale è necessario scorrere il file dall'inizio fino al blocco desiderato
  - L'accesso ad un file è meno efficiente
    - In generale comporta molti riposizionamenti della testina
- **Una soluzione consiste**
  - Nel raggruppare più blocchi in un "cluster"
  - Prevedere l'accesso ai tali gruppi, piuttosto che ai singoli blocchi
- **In questo modo si ha**
  - Miglioramento delle prestazioni
    - Minor numero di riposizionamenti della testina
  - Riduzione dello spazio utilizzato per i riferimenti
    - Uno per cluster invece di uno per blocco
  - Maggiore frammentazione interna
    - L'unità di allocazione è un cluster, che è più grande di un blocco
- **Questo approccio è utilizzato in molti sistemi operativi**

Allocazione Concatenata

- **Le liste di blocchi dell'allocazione concatenata sono fragili**
  - La perdita di un solo riferimento può rendere inaccessibili grandi quantità di dati
- **Una soluzione consiste nella costruzione di un indice per ogni partizione**
  - Detto File Allocation Table o FAT
- **La FAT**
  - Contiene tanti elementi quanti sono i blocchi del disco
    - Un blocco disponibile è indicato da uno 0 nella tabella
    - L'ultimo blocco di un file è indicao da uno speciale valore EOF
  - Ogni elemento della tabella contiene l'indice dell'elemento della FAT che che contiene il blocco successivo
  - La robustezza può essere facilmente aumentata mediante
    - Tecniche di codifica dei dati della FAT
    - Ridondanza
- **Questo metodo**
  - Comporta un tempo di accesso mediamente maggiore dell'allocazione concatenata
  - E' stato adottato da MS-DOS, OS/2 e diversi file system per memorie Flash

Allocazione Concatenata

- **Utilizzando allocazione concatenata e FAT**
  - Un descrittore di file deve semplicemente indicare il numero del primo blocco del file
  - Le informazioni sulla posizione dei blocchi successivi sono contenute nella FAT

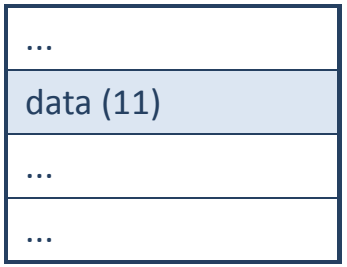


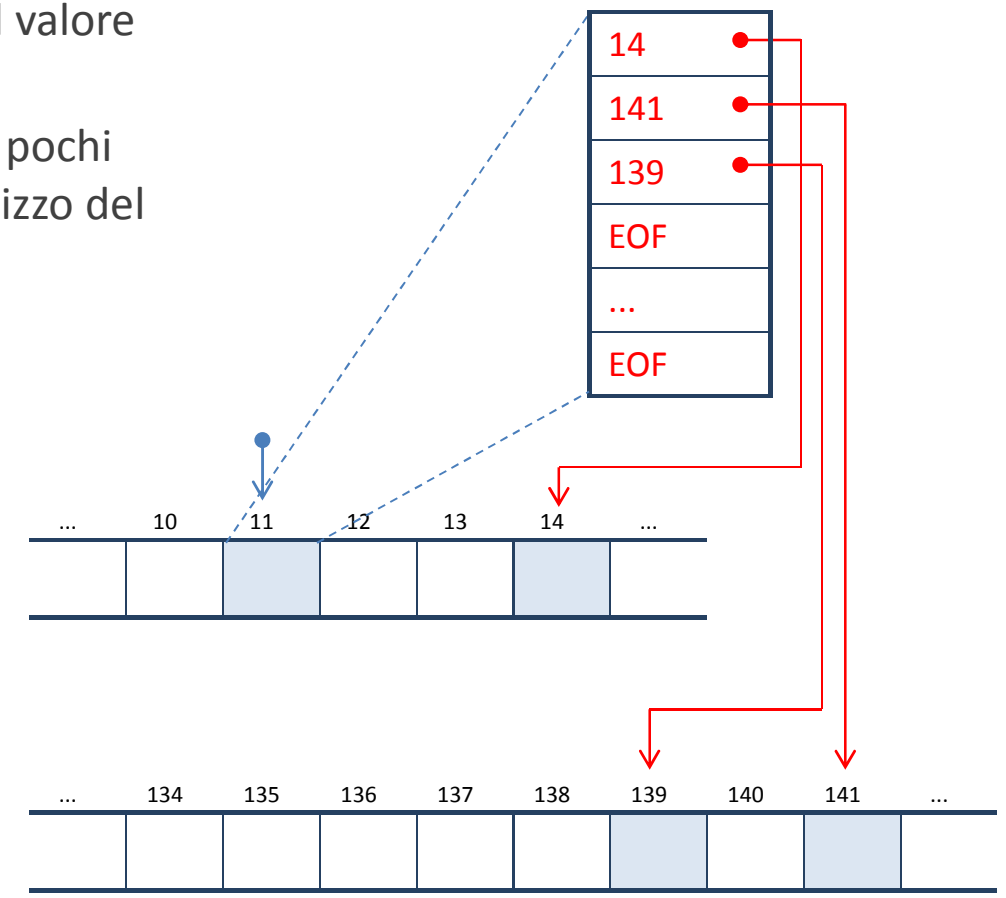
Allocazione Indicizzata

- **L'allocazione indicizzata**
  - Si basa sul raggruppamento di tutti i riferimenti
  - Risolve i problemi di scarsa efficienza della allocazione concatenata
- **I riferimenti ai blocchi di un file sono memorizzati in un unico blocco speciale**
  - Detto blocco indice
  - Il blocco indice appartiene al file stesso
- **Un blocco indice è quindi**
  - Un vettore di riferimenti ai blocchi del file
  - L'i-esimo elemento del vettore si riferisce all'i-esimo blocco del file
  - Il descrittore del file contiene il riferimento al blocco indice
- **In questo modo si ottiene**
  - Eliminzaione della frammentazione esterna
  - Accesso casuale ai blocchi molto efficiente
- **Lo spazio richiesto per i riferimenti è in generale maggiore che nel caso di allocazione concatenata**
  - Per pochi riferimeni è comunque necessario un intero blocco

Allocazione Indicizzata

- **Nel blocco indice**
  - Gli elementi che non si riferiscono a nessun blocco hanno il valore speciale "EOF"
  - Se un file è costituito da pochi blocchi si ha un sottoutilizzo del blocco indice





Allocazione Indicizzata

- **È importante scegliere una dimensione opportuna per il blocco indicie**
  - Se troppo grande
    - Viene sprecato molto spazio nel caso di file piccoli (pochi blocchi da indicizzare)
  - Se troppo piccolo
    - Non consente di trattare file di grandi dimensioni

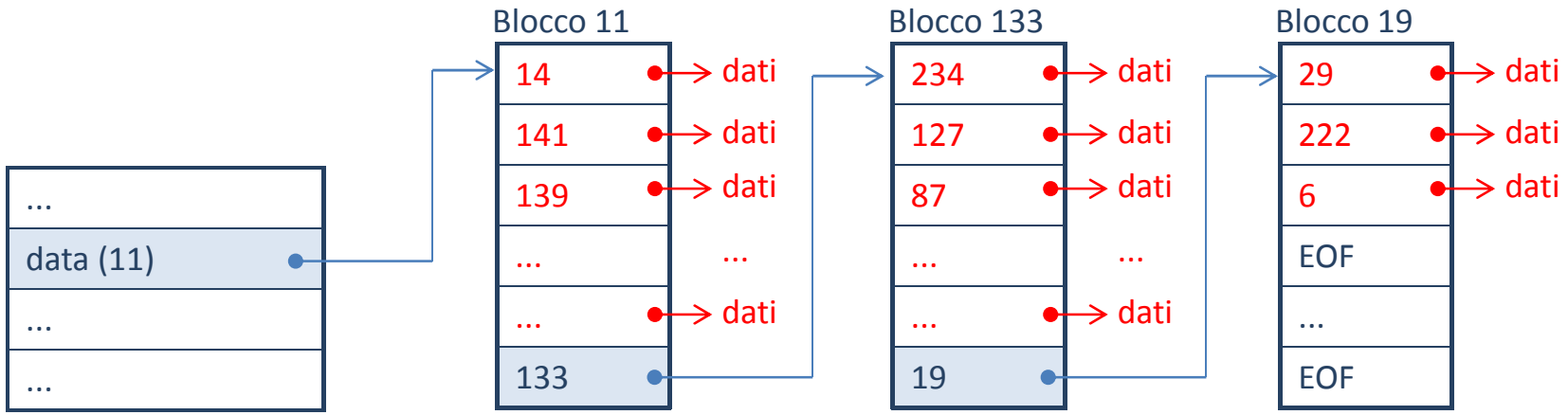
- **In genere il blocco indice**
  - Coicide con un blocco fisico

- **Esistono tre schemi possibili per risolvere i problemi legati alla gestione di file di dimensioni molto variabile**
  - Schema concatenato
  - Schema multilivello
  - Schema combinato

- **Tutti questi metodi sono utilizzati in sistemi operativi e file system reali**

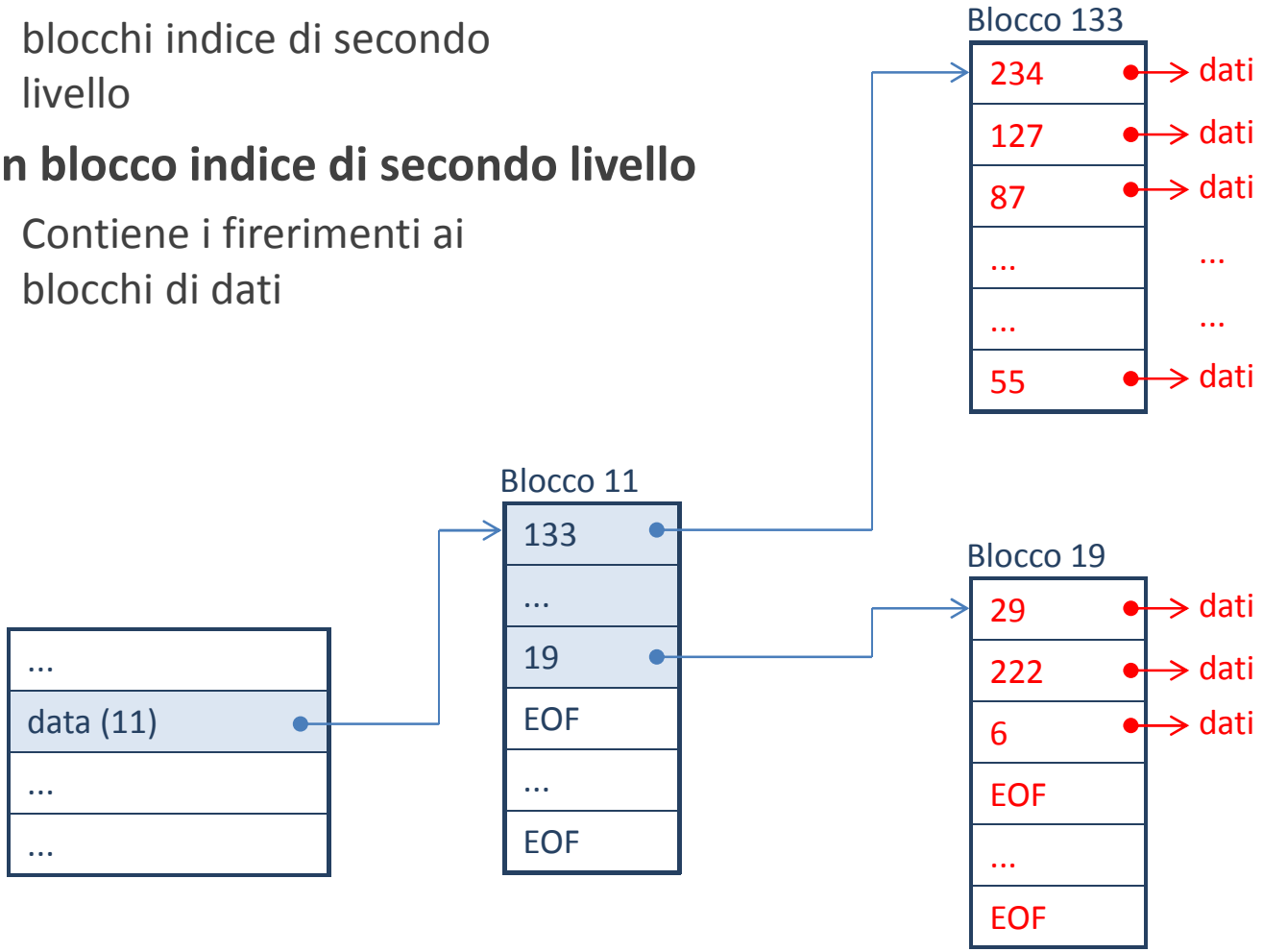
Allocazione Indicizzata – Schema concatenato

- **Il blocco indice**
  - Contiene i riferimenti ai blocchi del file
- **L'ultimo elemento del blocco indice**
  - Contiene un riferimento ad un secondo blocco indice, se il file è di grandi dimensioni
- **L'ultimo elemento dell'utimo blocco indice**
  - Contiene il valore speciale EOF ad indicare la fine del file



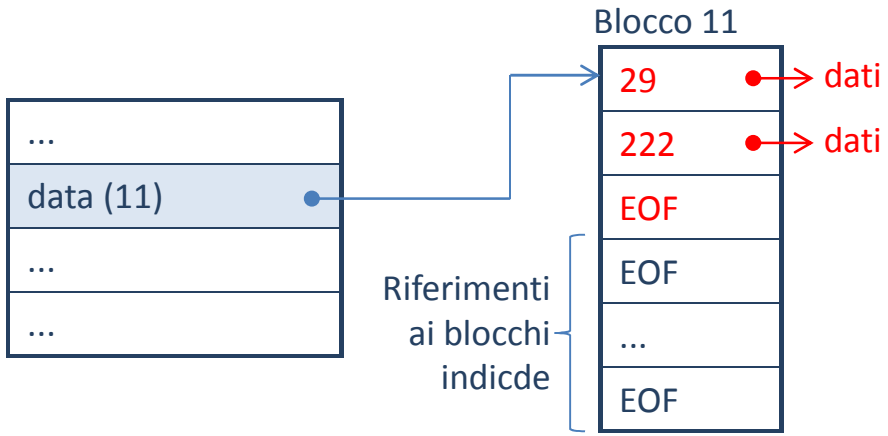
Allocazione Indicizzata – Schema multilivello

- **Un blocco indice di primo livello**
  - Contiene i riferimenti ai blocchi indice di secondo livello
- **Un blocco indice di secondo livello**
  - Contiene i firerimenti ai blocchi di dati



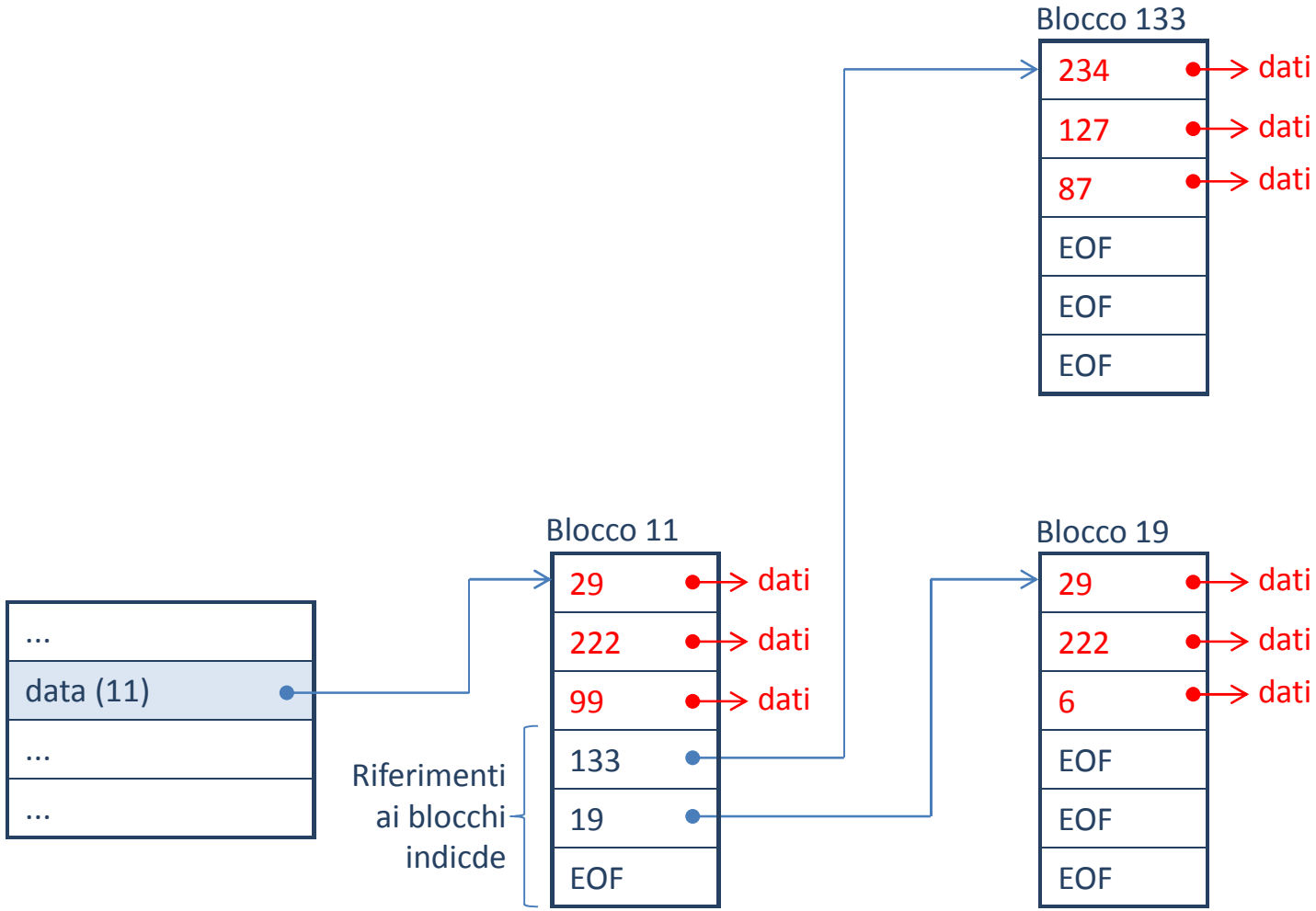
Allocazione Indicizzata – Schema combinato

- **In ogni blocco indice**
  - La parte iniziale contiene riferimenti a blocchi di dati del file
  - La parte finale contiene riferimenti a blocchi indice di secondo livello
- **Se il file ha dimensioni ridotte**
  - Non viene utilizzato il secondo livello



Allocazione Indicizzata – Schema combinato

- **Per un file di grandi dimensioni**
  - Si ricorre ai blocchi indice concatenati



Gestione dello spazio libero

- **All'atto della creazione e scrittura di un nuovo file**
  - E' necessario individuare sul disco il primo blocco libero

- **Una soluzione consiste nell'uso un vettore di bit in cui:**
  - La posizione del bit indica il numero del blocco
    - Se il bit vale 0 il blocco è già utilizzato
    - Se il bit vale 1 il blocco è disponibile

- **Si individua il primo blocco libero in questo modo:**
  - Si scorrono le parole (di b bit) fino alla prima diversa da zero
    - Sia k il numero di parole uguali a zero
  - Si trova l'offset d del primo bit con valore 1
  - L'indirizzo n del blocco è dato da:

$$n = k \times b + d$$

Gestione dello spazio libero

- **Una soluzione alternativa**
  - Si basa sull'utilizzo di una lista concatenata
  - Simile a quella utilizzata per i file
- **In questo schema:**
  - La posizione del primo blocco disponibile è memorizzata in una zona riservata del disco
  - Ogni blocco disponibile contiene un riferimento al blocco disponibile successivo
- **Questa soluzione**
  - E' migliore della precedente
    - Utilizza i blocchi stessi, organizzati in modo concatenato, per contenere l'informazione
  - In particolare per dischi di grandi dimensioni
- **Per migliorare l'efficienza si può ricorrere**
  - All'uso della FAT
  - A raggruppamenti
    - Di un numero prefissato di blocchi, oppure
    - Di dimensioni variabili

Realizzazione delle directory

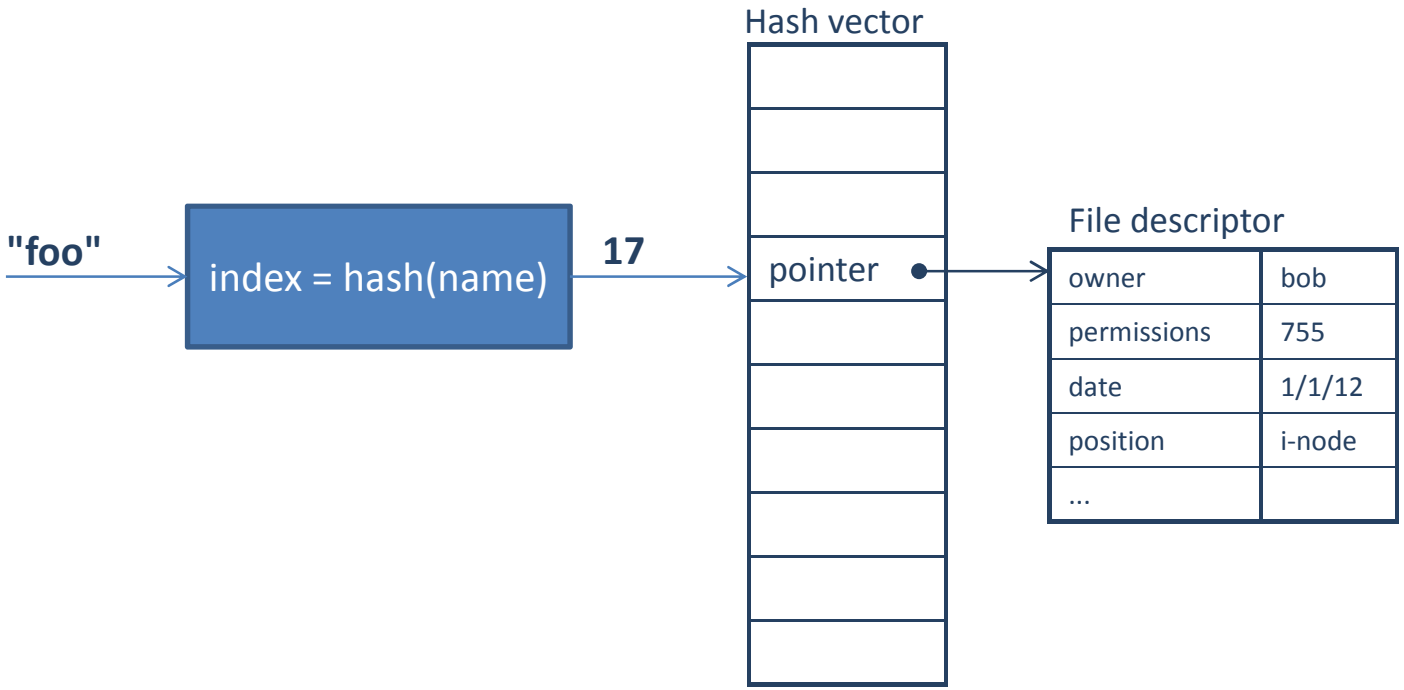
- **La soluzione più semplice consiste nel ricorrere ad una lista lineare**
- **Ogni elemento della lista contiene**
  - Il nome del file
  - Il descrittore del file
- **Tutte le operazioni sui file richiedono una ricerca del nome**
  - Eliminazione, creazione, rinomina, ...
- **La ricerca su una lista**
  - E' di semplice realizzazione
  - E' scarsamente efficiente
    - Tempo lineare nel numero degli elementi
- **Una lista ordinata lessicograficamente**
  - Consente una ricerca binaria (logaritmica)
  - Comporta problemi aggiuntivi per mantenere la lista in ordine

Realizzazione delle directory

- **Una tecnica più efficiente sfrutta le tabelle di hash**
- **Una tabella di hash è costituita**
  - Da un vettore di elementi contigui
    - I valori che è necessario estrarre
  - Una funzione di hash
    - Che restituisce la posizione del valore cercato in base alla chiave
- **Nel nostro caso**
  - Il vettore dei valorei contiene i descrittori di file
  - La chiave di ricerca è il nome del file
  - La funzione di hash
    - Prende in ingresso una stringa (nome del file)
    - Restituisce l'indice del descrittore
- **Il problema più critico consiste nella scelta della funzione di hash**
  - Deve essere semplice da calcolare
  - Deve minimizzare i clash
    - Due chiavi differenti restituiscono lo stesso indice
    - Una funzione per cui questo non accade si dice funzione di hash "perfetta"

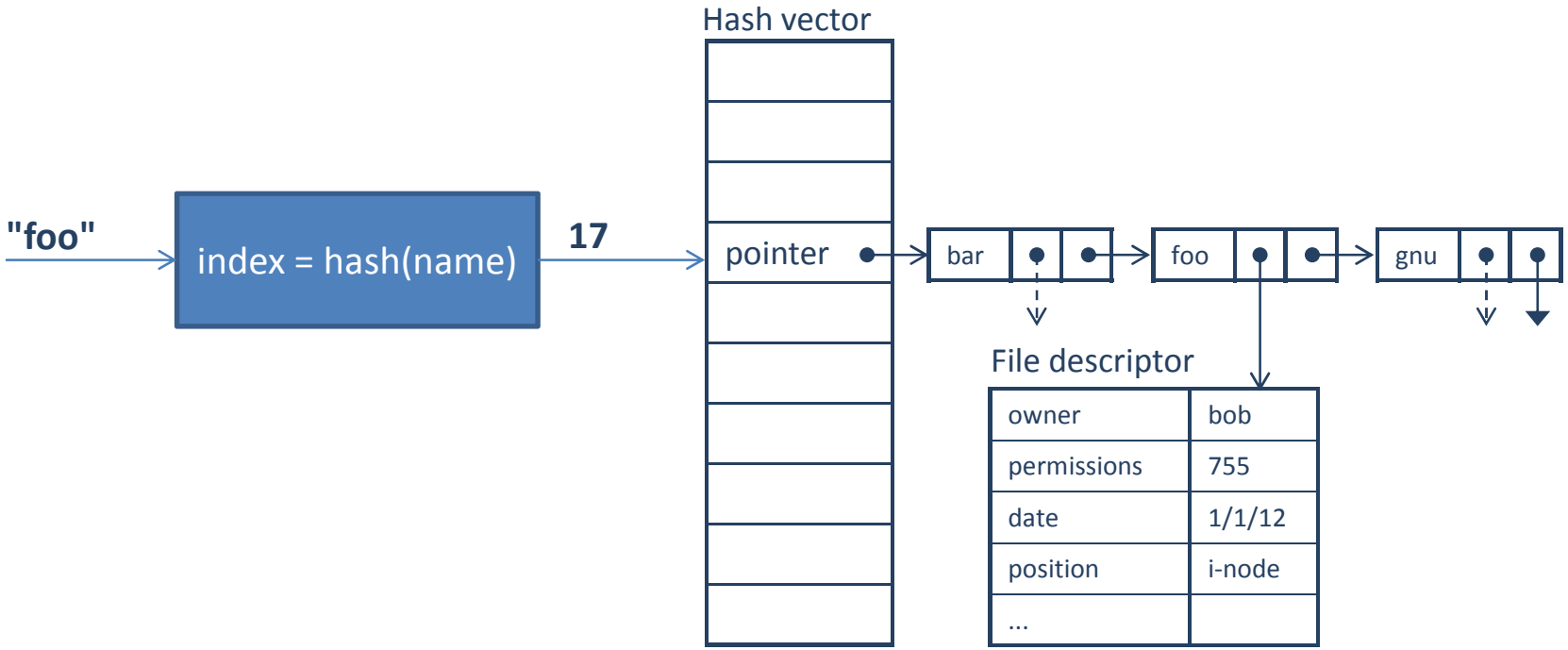
Realizzazione delle directory

- **Disponendo di una funzione di hash perfetta**
  - La ricerca avviene secondo lo schema mostrato schematicamente nella figura
  - La funzione restituisce esattamente l'indice dell'elemento cercato (unico)
  - Normalmente la tabella contiene puntatori (riferimenti) ai file descriptor



Realizzazione delle directory

- **Se la funzione di hash non è perfetta**
  - A più nomi corrisponde lo stesso indice
  - A quell'indice si trova un puntatore a una lista concatenata
    - La lista contiene i nomi dei file e i puntatori ai relativi descrittori
  - Una volta identificato l'inizio della lista
    - E' necessario scorrerla per trovare il nome cercato



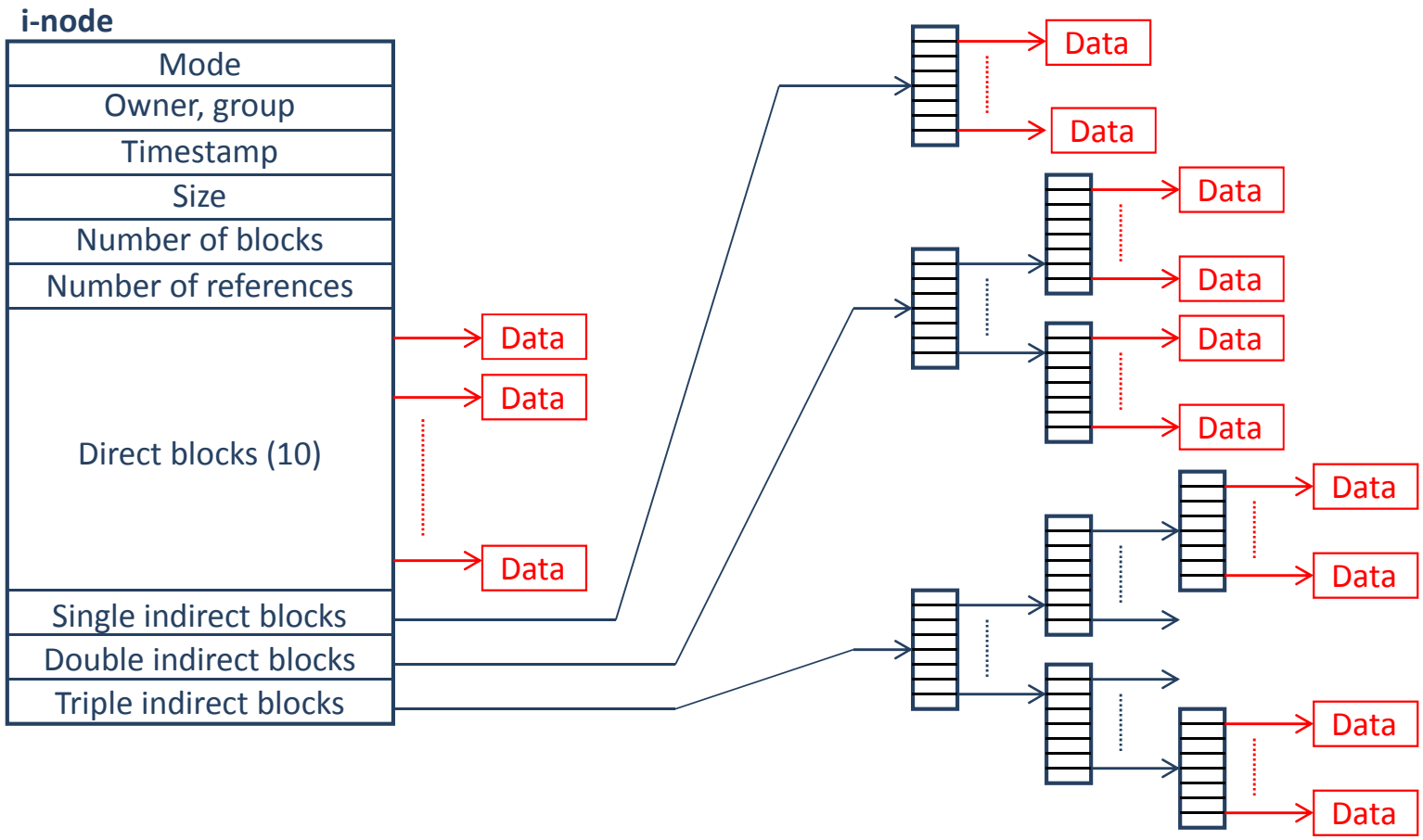
Filesystem Linux

- **Il filesystem di Linux**
  - E' organizzato in volumi
- **Ogni volume è organizzato in 4 zone distinte**

- **Blocco di boot**
  - Tipicamente occupa il primo settore del disco
  - Contiene il codice di inizializzazione del sistema operativo
- **Superblock**
  - Contiene le informazioni globali del file system (dimensione del volume, numero di blocchi liberi, ...)
- **Lista degli i-node**
  - E' una zona che contiene tutti gli i-node necessari a indicizzare i file
  - Un i-node è il descrittore di file previsto dai sistemi operativi derivati da UNIX
- **Blocchi dati**
  - Contengono i dati dei file e delle directory

Filesystem Linux

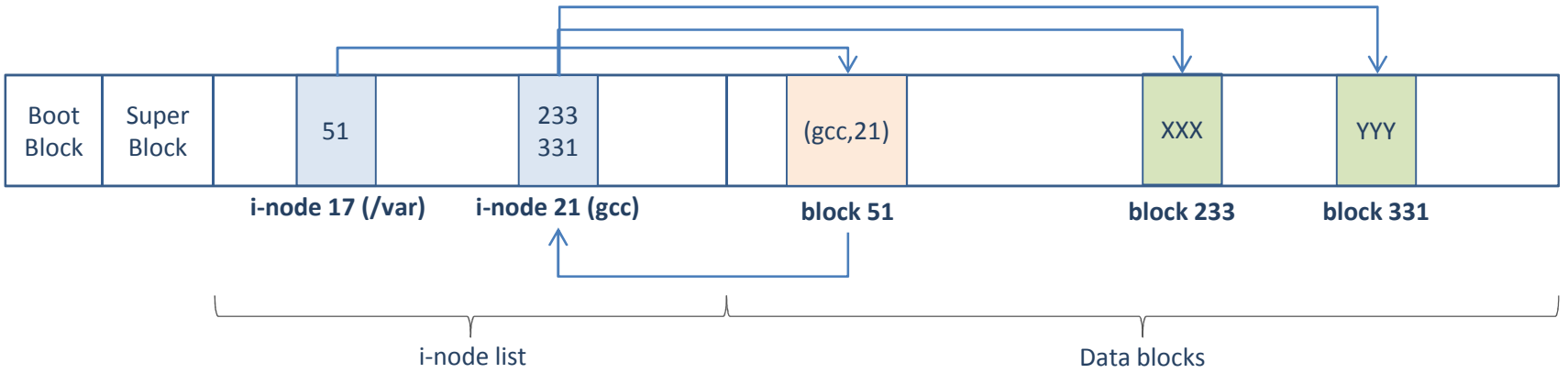
- **L'i-node è un descrittore di file che realizza lo schema indicizzato combinato**



Filesystem Linux

- **File: foo**
  - L'i-node contiene i riferimeti ai blocchi dati

- **Directory & File: /bin/gcc**
  - L'i-node contiene i riferimenti ai blocchi dati che realizzano l'indice della diectory
  - Ogni elemento dell'indice è costituito dalla coppia (nome file, numero i-node)



Filesystem Linux

- **Come abbiamo visto, il superblock contiene generali relative al volume**
  - Sono utilizzate per la gestione del filesystem
    - Creare/eliminare un file, aggiungere/rimuovere blocchi a un file esistente, ...

- **Le informazioni principali contenute ne superblock sono**
  - Dimensione del volume
  - Numero di blocchi liberi nel volume
  - Parte iniziale della lista dei blocchi liberi (free-list)
  - Numero di elementi della lista dei blocchi liberi
  - Numero di elementi della lista degli i-node (i-list);
  - Numero di i-node liberi
  - Lista degli i-node liberi (free i-node list)

Filesystem Linux

- **Per poter utilizzare un disco**
  - E' necessario organizzarlo secondo questa struttura logica
  - Questa operazione è detta "formattazione"
  - Viene svolta da appositi comandi (mkfs)
- **Il processo di formattazione svolge le seguenti operazioni**
  - Vengono scritte le dimensioni del volume e delle varie liste nel superblock
  - Viene allocata la lista (i-list) di dimensione fissa contenente gli in-node
    - All'inizio gli i-node sono tutti liberi
    - Tranne quello che si riferisce alla "root directory", indicata con "/"
  - Gli i-node hanno dimensione fissa e sono predisposti in sequenza
    - In questo modo è possibile accedere ad un i-node in base al suo indice
  - Le posizioni degli i-node liberi sono riportate nella free i-node list
  - Viene costruita la lista dei blocchi liberi
    - E' una lista concatenata di blocchi cointenenti gli indici dei blocchi liberi
    - La parte iniziale della free list è contenuta nel superblock
    - La parte restante è contenuta nei normali blocchi dati del volume

Gestione del filesystem Linux

- **Tutte le informazioni necessarie alla gestione dei dati del filesystem sono contenute nelle strutture dati presenti sul disco che abbiamo appena visto**
- **Perché il sistema operativo possa gestire il filesystem perr conto di diversi processi concorrenti sono necessarie informazioni aggiuntive**
- **Linux realizza la gestione**
  - Mediante tre tabelle
  - Contenute nella memoria del sistema operativo
- **Tabella dei descrittori dei file aperti per processo**
  - Esiste una tabella per ogni processo attivo nel sistema
  - Gli elementi fanno riferimento alla tabella globale dei file aperti
- **Tabella globale dei file aperti**
  - Tabella unica che contiene una riga per ogni file aperto (per tutti i processi)
  - Tabella degli i-node
    - Detti anche in-core i-node, mentenuti per ragioni di efficienza
    - Tabella globale del sistema contenente una copia in memoria degli inode

Gestione del filesystem Linux

- **La tabella dei file aperti per processo**
  - E' associata ad ogni processo attivo nel sistema operativo
  - E' parte della sezione system-data dell'immagine del processo
- **Ogni riga contiene**
  - Un elemento associato ad un file aperto
  - L'indice della tabella costituisce l'identificatore del file
  - L'elemento contine l'indice nella tabella globale relativo a quel file
- **Per convenzione, 3 file sono sempre aperti**
  - Standard input (stdin)
    - Identificatore uguale a 0
    - Associato alla tastiera
  - Standard output (stdout)
    - identificatore uguale a 1
    - Associato alla console (video o pseudo-terminale)
  - Standard error (stderr)
    - Identificatore uguale a 2
    - Aassociato alla console (video o pseudo-terminale)

Gestione del filesystem Linux

- **La tabella globale dei file aperti**
  - Raccoglie i riferimenti a tutti i file correntemente aperti da tutti i processi attivi
  - Le tabelle dei file dei processi contengono riferimenti agli elementi di questa tabella
- **Ogni riga contiene**
  - Un elemento associato ad un file aperto
  - Un puntatore all'i-node associato al file
    - L'i-node del file viene copiato dal disco in memoria per maggiore efficienza
    - Gli i-node dei file aperti risiedono nella tabella degli i-node
  - Indice della posizione corrente nel file
    - Indica la posizione cui sarà eseguito il prossimo accesso
    - Espressa come offset in byte rispetto all'inizio del file
    - Si possono avere anche due indici, uno per la lettura e uno per la scrittura
  - Contatore dei riferimenti
    - Indica quanti riferimenti al file sono presenti e attivi nei vari processi
  - Modalità di apertura
    - Lettura (read), scrittura (write), lettura/scrittura (read/write), aggiunta (append)
  - Altre informazioni

Gestione del filesystem Linux

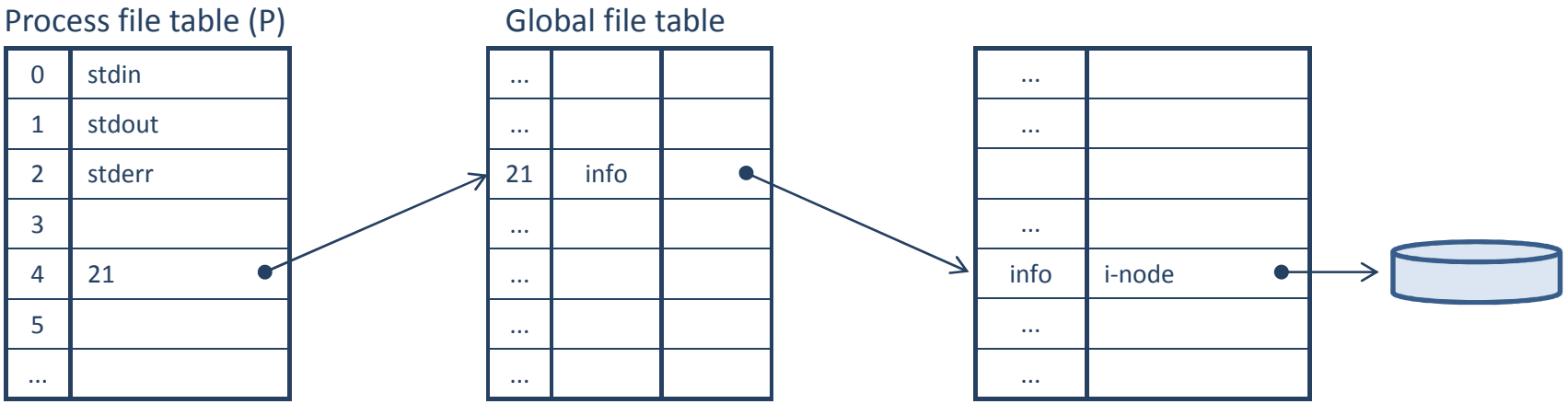
- **Per ragioni di efficienza**
  - Gli i-node dei file aperti sono copiati in memoria
  - All'atto dell'apertura del file

- **Tali i-node sono raccolti nella tabella degli i-node**

- **Un elemento della tabella contiene**
  - Una copia in memoria dell'i-node
  - Informazioni riguardanti lo stato dell'i-node
    - Processi in attesa di accedere all'i-node
    - Stato dell'i-node, cioè se ha subito modifiche rispetto alla sua copia su disco
    - Stato del file, cioè se sono stati modificati i blocchi dati
    - ...

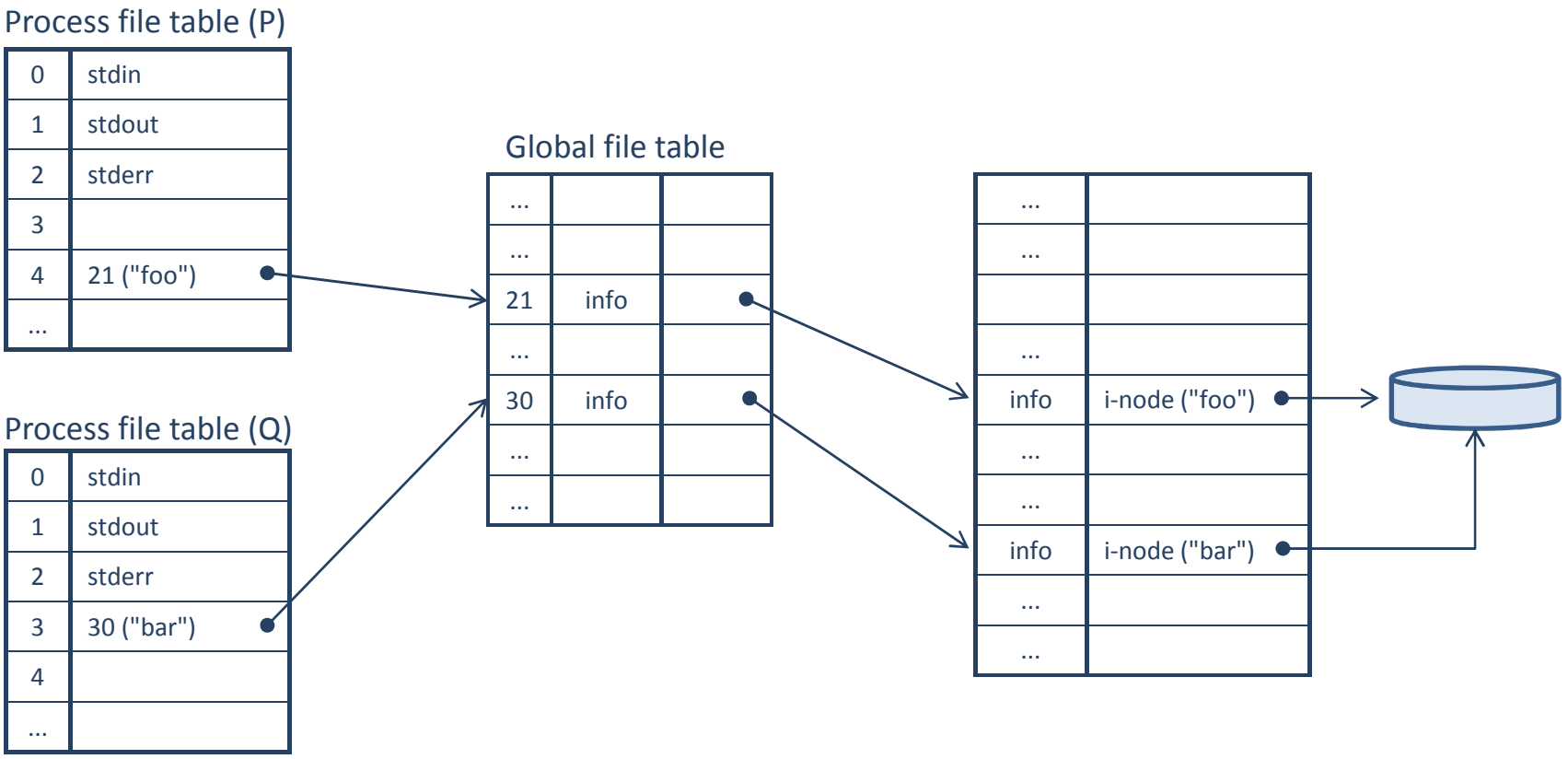
Gestione del filesystem Linux

- **Le tabelle sono organizzate come segue**



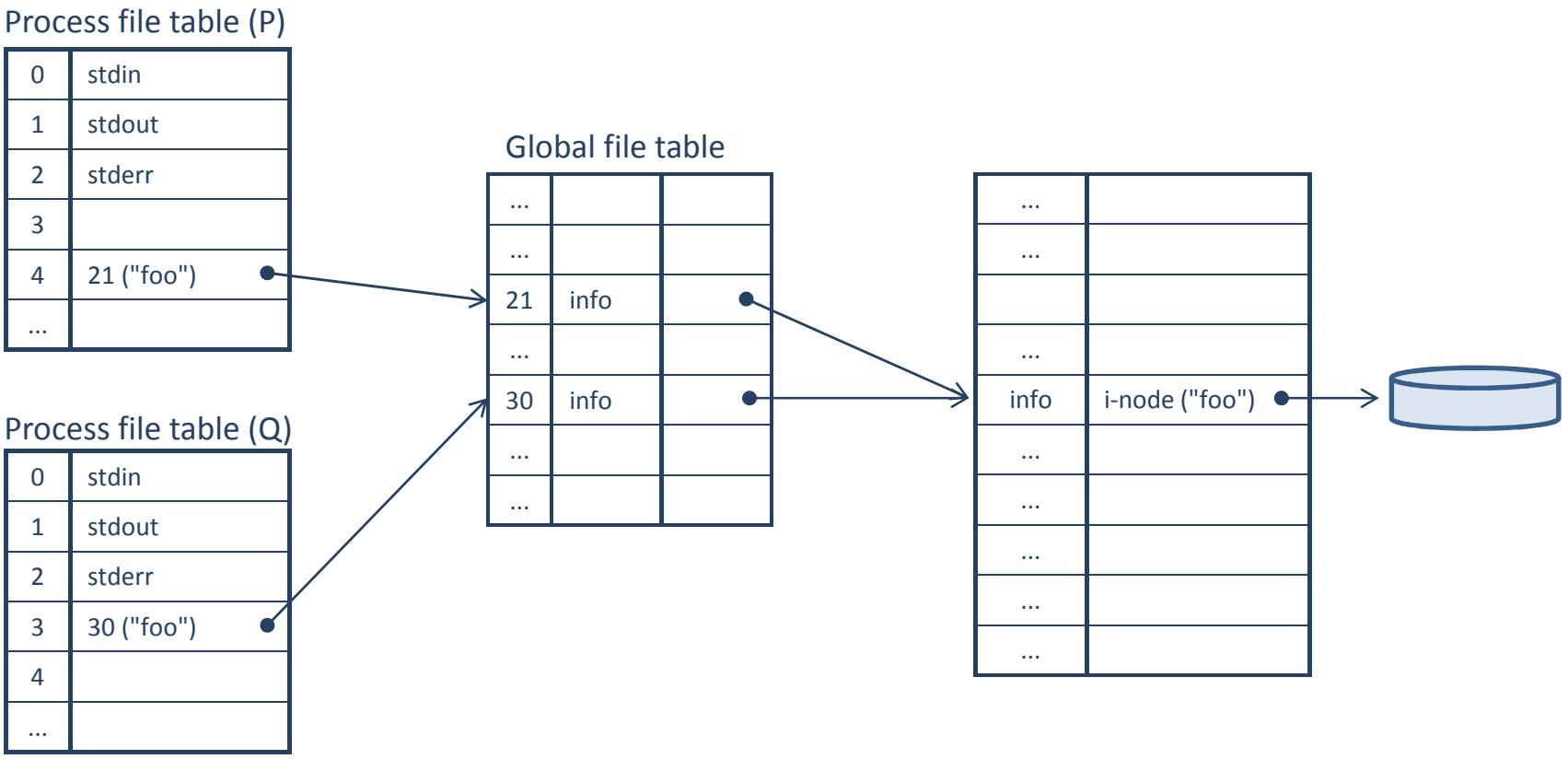
Gestione del filesystem Linux

- **Caso 1:**
  - Il processo P apre il file "foo"
  - Il processo Q apre il file "bar"



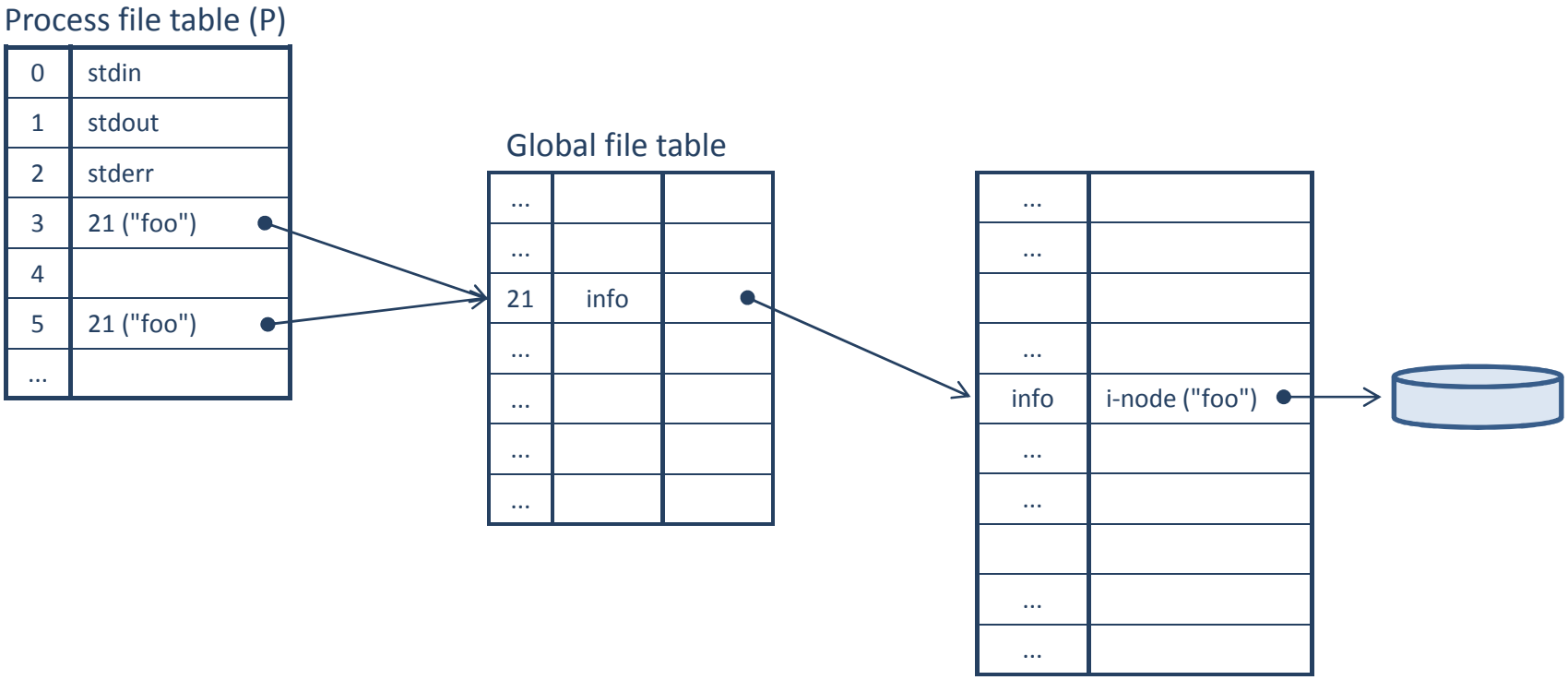
Gestione del filesystem Linux

- **Caso 2:**
  - Il processo P apre il file "foo"
  - Il processo Q apre lo stesso file "foo"



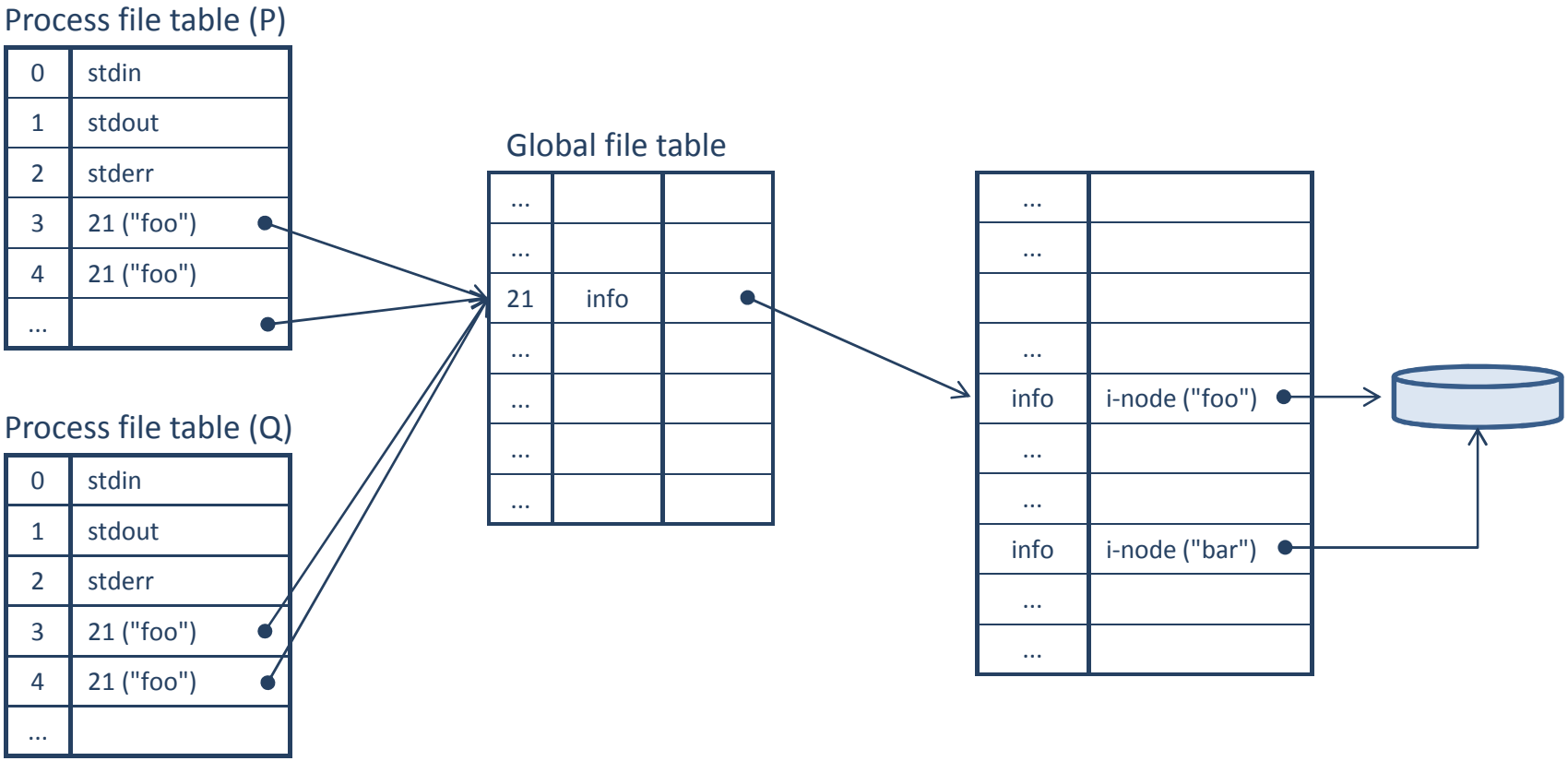
Gestione del filesystem Linux

- **Caso 3:**
  - Il processo P apre il file "foo"
  - Il processo P richiede la duplicazione del descrittore di file (dup)



Gestione del filesystem Linux

- **Caso 3:**
  - Il processo P apre il file "foo"
  - Il processo P richiede la duplicazione del descrittore di file (dup)
  - Il processo P esegue una fork() generando il processo Q



Primitive di gestione del filesystem

- **Esistono due livelli di primitive per l'accesso ai file**
  - Primitive di base, per l'accesso a basso livello
    - Offerte dalle librerie del sistema operativo
  - Primitive avanzate, per l'accesso mediante stream
    - Offerte dalla libreria standard del C
- **La gestione di un file richiede alcune operazioni**
  - Apertura
    - Creazione degli elementi delle varie tabelle, caricamento dell'i-node
  - Accesso
    - Creazione
    - Lettura e scrittura sequenziale
    - Riposizionamento, per l'accesso casuale
    - Cancellazione
  - Chiusura
    - Rilascio delle risorse e aggiornamento delle tabelle

Primitive di gestione del filesystem

- **Apre un file**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags);
int open( const char *pathname, int flags, mode_t mode );
```

- **Argomenti**
  - pathname
    - Il pathname assoluto o relativo del file
  - flags
    - Specifica le modalità di apertura
    - E' la combinazione in bitwise-or di alcuni flag predefiniti
  - mode
    - Specifica i permessi e alle modalità di gestione del file, se questo viene creato
- **Valore di ritorno**
  - Il valore dell'identificatore del file (file descriptor)
  - Il valore -1 in caso di errore

Primitive di gestione del filesystem

- **I flag una combinazione tra due insiemi di valori predefiniti**
- **Modalità di apertura**
  - O_RDONLY
    - Sola lettura
  - O_WRONLY
    - Sola scrittura
  - O_RDWR
    - Lettura/scrittura
- **Un insieme di proprietà aggiuntive**
  - O_APPEND
    - Modalità di append. Prima di ogni scrittura il puntatore di scrittura viene automaticamente posizionato alla fine del file
  - O_ASYNC
    - Modalità di accesso asincona, basata su segnali. Non è disponibile per i normali file
  - O_CLOEXEC
    - Chiude automaticamente il file nel momento in cui viene eseguita una exec()
  - O_CREAT
    - Se il file non esiste, viene creato

Primitive di gestione del filesystem

- **Il parametro mode indica**
- **Permessi del file**
  - User
    - S_IRWXU, S_IRUSR, S_IWUSR, S_IXUSR
  - Group
    - S_IRWXG, S_IRGRP, S_IWGRP, S_IXGRP
  - Others
    - S_IRWXO, S_IROTH, S_IWOTH, S_IXOTH
- **Altre opzioni di creazione**
  - Buffering e caching
  - Creazione di directory
  - Creazione forzata
  - File di grandi dimensioni
  - Troncamento
  - ...

Primitive di gestione del filesystem

- **Apre un file esistente**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat( const char *pathname, mode_t mode );
```

- **E equivalente ad open()**
- **Sottindende la combinazione di flag O_CREAT|O_WRONLY|O_TRUNC**
  - O_CREAT
    - Se non esiste, crea un nuovo file
  - O_WRONLY
    - Apre il file in sola scrittura
  - O_TRUNC
    - Se il file esiste lo "tronca", cioè ne elimina tutti i dati

Primitive di gestione del filesystem

- **Legge dati da un file**

```
#include <unistd.h>
ssize_t read( int fd, void *buf, size_t count );
```

- **Argomenti**
  - fd
    - Il descrittore di file ottenuto da una precedente operazione di apertura
  - buf
    - Puntatore ad una zona di memoria (buffer) in cui posizionare i dati letti
  - count
    - Numero massimo di byte da leggere
    - Questo valore non deve superare la dimesnione del buffer
- **Valore di ritorno**
  - Se la lettura va a buon fine, restituisce il nuemro di byte effetivamente letti
  - Se si verifica un errore, restutisce -1
    - Per conoscere il tipo di errore è necessario analizzare il valore della variabile errno

Primitive di gestione del filesystem

- **Scrive dati su un file**

```
#include <unistd.h>
ssize_t write( int fd, void *buf, size_t count );
```

- **Argomenti**
  - fd
    - Il descrittore di file ottenuto da una precedente operazione di apertura
  - buf
    - Puntatore alla zona di memoria (buffer) che contiene i dati da scrivere
  - count
    - Numero massimo di byte da scrivere
- **Valore di ritorno**
  - Se la scrittura va a buon fine, restituisce il nuemro di byte effetivamente scritti
  - Se si verifica un errore, restutisce -1
    - Per conoscere il tipo di errore è necessario analizzare il valore della variabile errno

Primitive di gestione del filesystem

- **Riposizionamento del puntatore di lettura/scrittura**

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek( int fd, off_t offset, int whence );
```

- **Argomenti**
  - fd
    - Il descrittore di file ottenuto da una precedente operazione di apertura
  - offset
    - L'entità dello spostamento, espressa in numero di byte
  - whence
    - Origine dello spostamento
    - SEEK_SET Dall'inizio del file
    - SEEK_CUR Dalla posizione corrent
    - SEEK_END Dalla fine del file
- **Valore di ritorno**
  - Se l'operazione va a buon fine, restituisce la posizione corrente nel file
  - Se si verifica un errore, restutisce -1

Primitive di gestione del filesystem

- **Chiusura di un file**

```
#include <unistd.h>
int close( int fd );
```

- **Argomenti**
  - fd
    - Il descrittore di file ottenuto da una precedente operazione di apertura
- **Valore di ritorno**
  - Se l'operazione va a buon fine, restituisce 0
  - Se si verifica un errore, restutisce -1