Sistemi Operativi

Laurea in Ingegneria Informatica
Università Roma Tre
Docente: Romolo Marotta

I/O e file management

1. Obiettivi ed organizzazione moduli di I/O
    1. I/O buffering
    2. I/O e Disk scheduling
2. Concetto di file e file system
    1. Operazioni su file e metodi di accesso
    2. Directory
    3. Allocazione
    4. Accenni di file system in Unix e Linux

I/O management

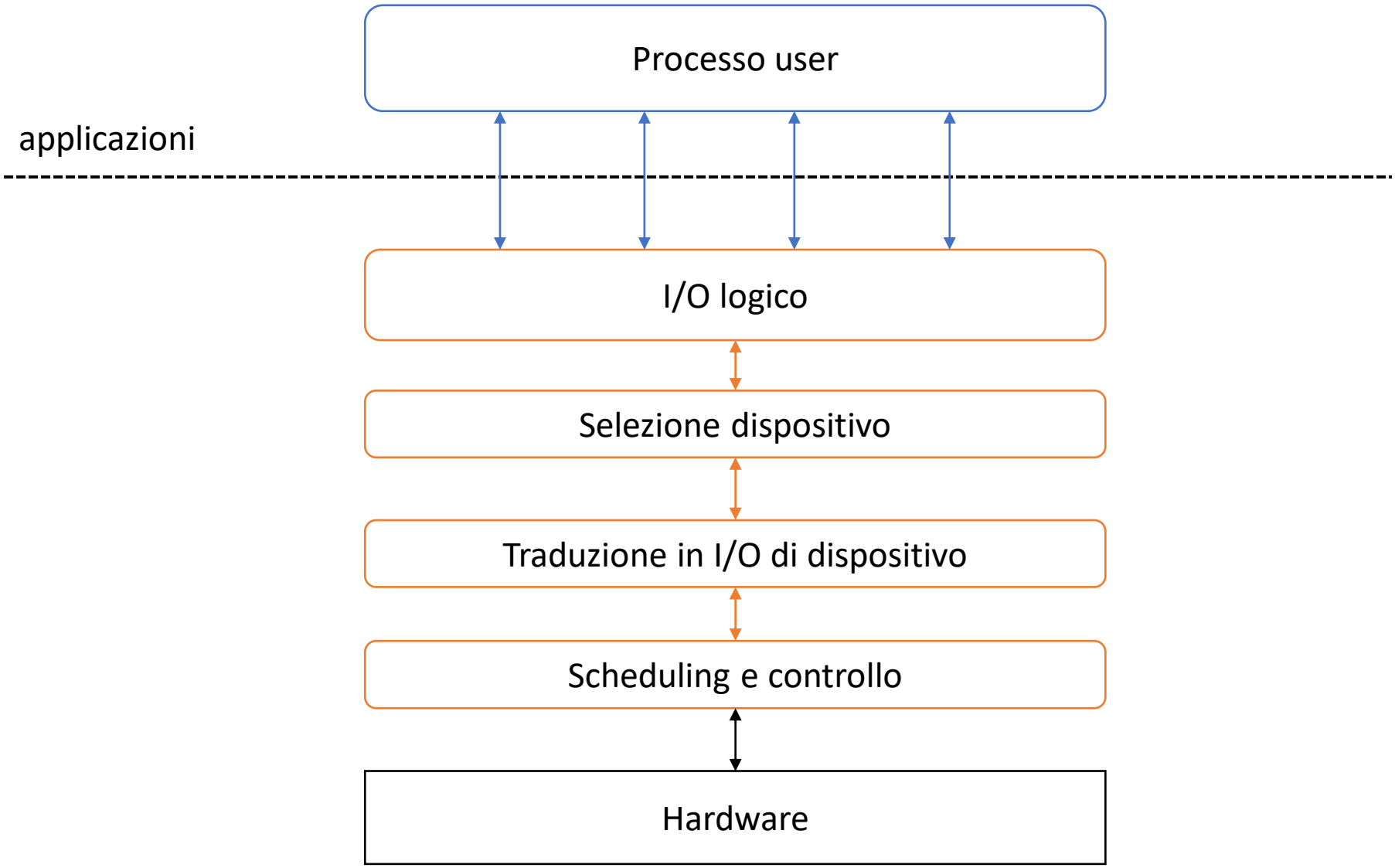
- Diversi tipi di dispositivi
  - Interazione con l'essere umano
  - Interazione con la macchina
  - Comunicazione tra macchine
- Caratteristiche differenti:
  - Applicazioni
  - Data rate
  - Controllo
  - Unità del trasferimento di dati
  - Rappresentazione del dato
  - Condizioni di errore
  - Gestione dei consumi

I/O management

Obiettivi:

- Efficienza
  - Tipicamente l'interazione con dispositivi di I/O è il collo di bottiglia
  - Multiprogrammazione allevia il problema, ma richiede swapping
  - Swapping richiede operazioni di I/O
- Generalità
  - Necessità di trattare i dispositivi in modo **uniforme**
  - Fornire servizi di I/O con interfacce standard (indipendenti dal tipo di dispositivo)
  - Progettazione gerarchica e modulare tesa a nascondere dettagli di basso livello

Modello di organizzazione moduli di I/O



I/O management

- Evoluzione delle funzioni di I/O
  - Programmed I/O
    - Processore **attende** la terminazione di comandi e **copia** dati da I/O a memoria
  - Interrupt-driven I/O
    - Processore **non attende** la terminazione di comandi e **copia** dati da I/O a memoria
  - Direct memory access
    - Processore **non attende** la terminazione di comandi e **non copia** dati da I/O a memoria

I/O buffering

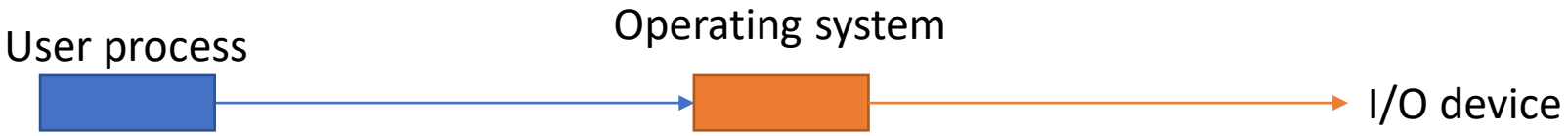
L'I/O diretto su memoria dei processi ha implicazioni non banali:

- area destinata all'I/O non è swappabile
  - sottoutilizzo delle risorse
- area destinata all'I/O è swappabile
  - Deadlock
    - Processo bloccato in attesa di I/O e poi swappato
    - I/O in attesa che il processo sia riattivato
- I/O viene effettuato su memoria riservata al sistema operativo chiamato **buffer**
- L'utilizzo di buffer è in generale utile al fine di risolvere altre criticità:
  - Appianare la differenza di velocità tra produttore e il consumatore del dato
  - Appianare la differenza della taglia del dato che può essere maneggiata dal produttore e dal consumatore
  - Supportare la semantica di copia per l'I/O

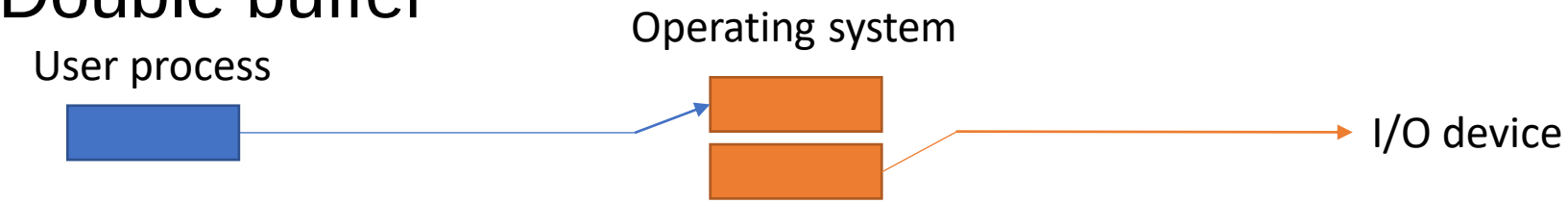
I/O buffering

- No buffering

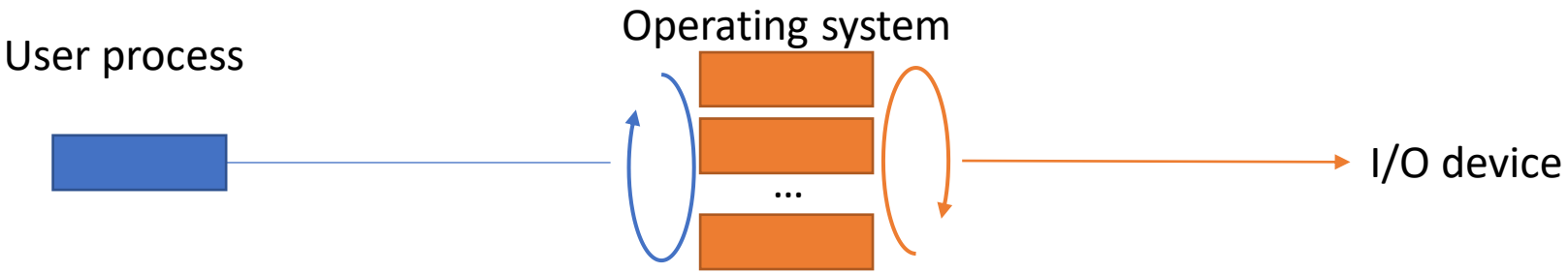
- Single buffer



- Double buffer



- Circular buffer



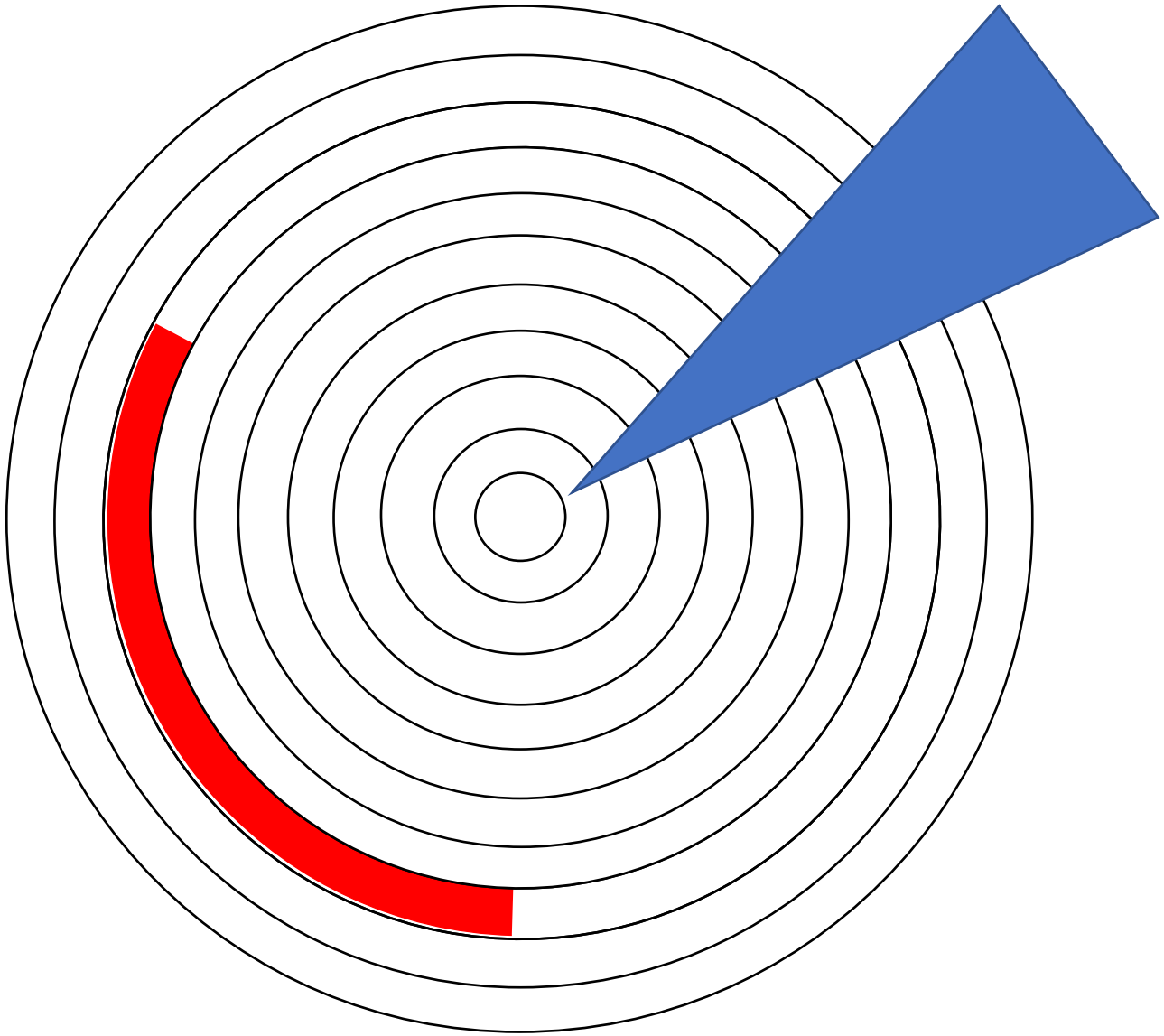
I/O scheduling

- Definisce la pianificazione per cui un dispositivo di I/O viene attivato per le sue operazioni
- In alcuni casi, mantenere l'ordine tra richieste di I/O e le effettive operazioni può essere necessario (e.g., terminale)
- FCFS non è necessariamente la soluzione più efficiente
  - Forte dipendenza dalle peculiarità dell'hardware caratteristico del dispositivo di I/O

Hard disk – caratteristiche salienti

- Ogni blocco è accessibile in lettura/scrittura ad ogni istante di tempo
- L'usura del dispositivo è essenzialmente legata all'usura delle parti meccaniche coinvolte
  - Non ci sono relazioni dirette tra operazioni di scrittura/lettura ed usura del dispositivo
  - Relazioni indirette tra operazioni ed usura
    - Per leggere/scrivere è necessario muovere la testina
- Tempo di accesso
  - Tempo di accodamento della richiesta
  - Tempo di acquisizione del canale di I/O (potrebbe essere condiviso con altri dispositivi)
  - Seek time: tempo per spostare la testina sulla traccia corretta
  - Ritardo di rotazione: tempo di rotazione per allineare l'inizio del settore di interesse alla testina
  - Tempo di trasferimento: tempo speso a ruotare il disco affinché tutto il settore venga letto dalla testina

Hard disk



Hard disk – Tempi di accesso

- Tempo di trasferimento
  - B=byte da trasferire
  - N=byte per traccia
  - R=tempo di rivoluzione
  - BR/N
- Ritardo di rotazione
  - in media occorre metà giro per posizionarsi sul settore corretto
  - Tipicamente il disco gira ad una velocità costante (da 5400rpm a 15000rpm)
  - Da 5.5ms a 2ms
- Seek time
  - in media occorre un terzo del full seek time (dalla traccia più interna a quella più esterna)
  - tipicamente nell'ordine dei millisecondi

Hard disk – Tempi di accesso

Ritardo di rotazione

- Perché metà giro è la distanza attesa per allinearsi al settore di interesse?
- Una volta che la testina raggiunge la traccia di interesse, l'inizio del settore può trovarsi ad una qualsiasi distanza angolare dalla testina
- In altre parole, la distanza angolare X è uniformemente distribuita tra 0 e l'angolo giro (2π)
  - $X \sim f(x) = \frac{1}{2\pi}$ per $x \in [0,2\pi]$, 0 per $x \notin [0,2\pi]$
  - $\mathbb{E}(X) = \int_{-\infty}^{+\infty} x f(x)\, dx = \int_0^{2\pi} \frac{x}{2\pi} dx = \left[\frac{x^2}{4\pi}\right]_0^{2\pi} = \frac{4\pi^2}{4\pi} = \pi$

Hard disk – Tempi di accesso

Seek time

- Perché un terzo del full seek time è la distanza attesa per raggiungere la traccia di interesse?
- Si calcola come il rapporto tra la somma di tutte le distanze possibili tra due tracce (A) e il numero di movimenti distinti che la testina può effettuare (B)
- Sia N il numero di tracce del disco
- La traccia di partenza può essere scelta tra N
- La traccia destinazione può essere scelta tra N
- $B = N^2$

Hard disk – Tempi di accesso

- Data la traccia i, il numero di tracce $D(i,j)$ da attraversare per raggiungere una traccia j è

$$D(i,j) = |i - j|$$

- La somma di tutte le possibili distanze tra una specifica traccia i e una generica traccia j è

$$A_i = \sum_{j=1}^{N} D(i,j) = \sum_{j=1}^{N} |i - j|$$

- La somma di tutte le possibile distanze A è

$$A = \sum_{i=1}^{N} A_i = \sum_{i=1}^{N} \sum_{j=1}^{N} |i - j|$$

$$\sum_{i=1}^{N} \left[\sum_{j=1}^{i} (i - j) + \sum_{j=i+1}^{N} (j - i) \right]$$

Hard disk – Tempi di accesso

Seek time

$$\sum_{i=1}^{N}\left[\sum_{j=1}^{i}(i-j)+\sum_{j=i+1}^{N}(j-i)\right]$$

$$\sum_{i=1}^{N}\left[\sum_{j=1}^{i}(i-j)+\sum_{j=1}^{N}(j-i)-\sum_{j=1}^{i}(j-i)\right]$$

$$\sum_{i=1}^{N}\left[\sum_{j=1}^{i}2(i-j)+\sum_{j=1}^{N}(j-i)\right]$$

$$\sum_{i=1}^{N}\left[2i^2-i(i+1)+\frac{N(N+1)}{2}-Ni\right]$$

$$\sum_{i=1}^{N}\left[i^2-i+\frac{N(N+1)}{2}-Ni\right]$$

$$\frac{N(N+1)(2N+1)}{6}-\frac{N(N+1)}{2}+\frac{N^2(N+1)}{2}-\frac{N^2(N+1)}{2}$$

$$\frac{2N^3+3N^2+N}{6}-\frac{N(N+1)}{2}$$

$$\frac{N^3}{3}-\frac{N}{12}$$

Hard disk – Tempi di accesso

Seek time

- Perché un terzo del full seek time è la distanza attesa per raggiungere la traccia di interesse?

- Il seek time atteso è

$$\frac{A}{B} = \frac{\frac{N^3}{3}}{N^2} = \frac{N}{3}$$

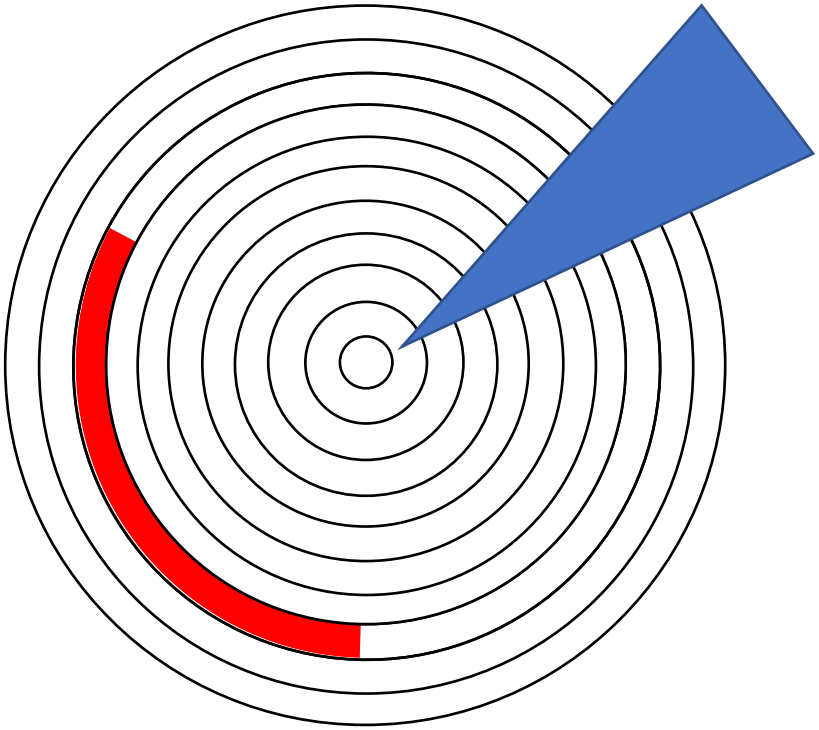
Hard disk – Accesso sequenziale

- Esempio
  - 500 settori per traccia
  - Average seek time = 4ms
  - Velocità di rotazione = 15000rpm = 250rps
  - Seek time per tracce adiacenti trascurabile
- Dati posizionati su 5 tracce consecutive
  - Tempo di rivoluzione = 4ms
  - Tempo di trasferimento = 5x(tempo di rivoluzione) = 20ms
  - Tempo di seek = 4ms (da considerare una sola volta)
  - Ritardo di rotazione complessivo = 5x(tempo di rivoluzione)/2 = 10ms
  - Tempo totale per l'accesso = 20ms + 10ms + 4ms = 34ms

Hard disk – accesso casuale

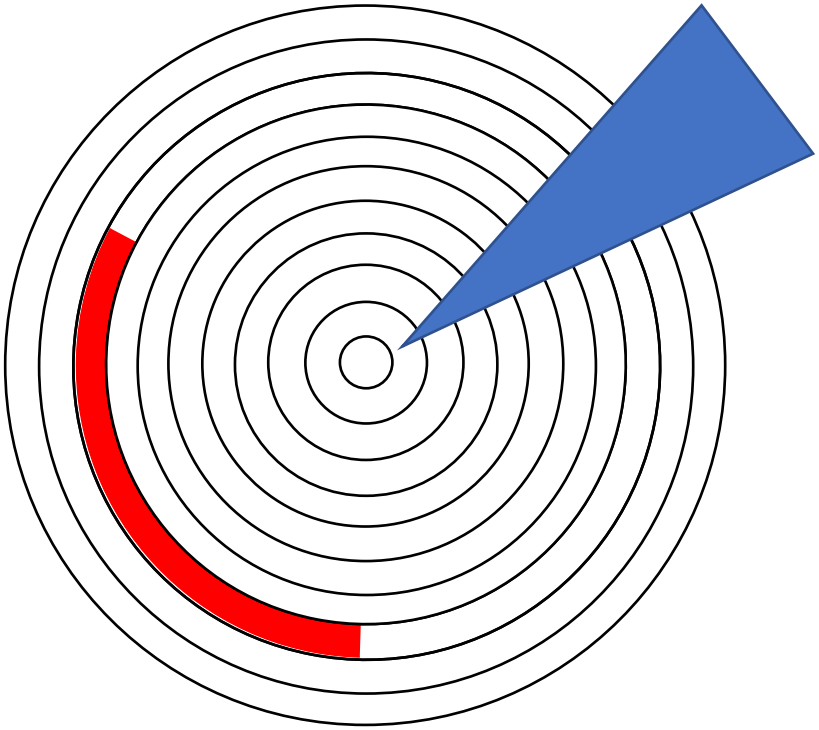
- Esempio
  - 500 settori per traccia
  - Average seek time = 4ms
  - Velocità di rotazione = 15000rpm = 250rps
  - Seek time per tracce adiacenti trascurabile
- Dati posizionati su 2500 settori non consecutivi
  - Tempo di rivoluzione = 4ms
  - Tempo di trasferimento = 5x(tempo di rivoluzione) = 20ms
  - Tempo totale di seek = 2500x(tempo di seek) = 10s
  - Ritardo di rotazione complessivo = 2500x(tempo di rivoluzione)/2 = 5s
  - Tempo totale per l'accesso = 20ms + 10s + 5s = 15.02s

Disk scheduling



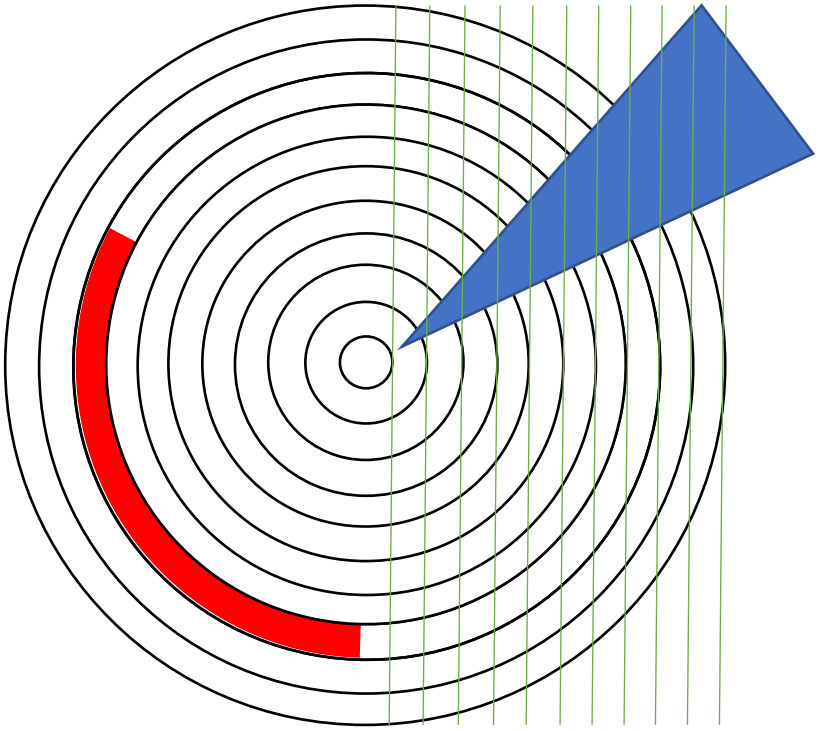
Disk scheduling

- Minimizzare i movimenti meccanici è un aspetto cruciale per massimizzare le performance di accesso a disco
- Seek time è oggetto di numerosi algoritmi di ottimizzazione
- Metodologia
  - Esecuzione sintetizzata come una sequenza di richieste identificate dal numero di traccia
- Metrica
  - numero di tracce complessivamente attraversate



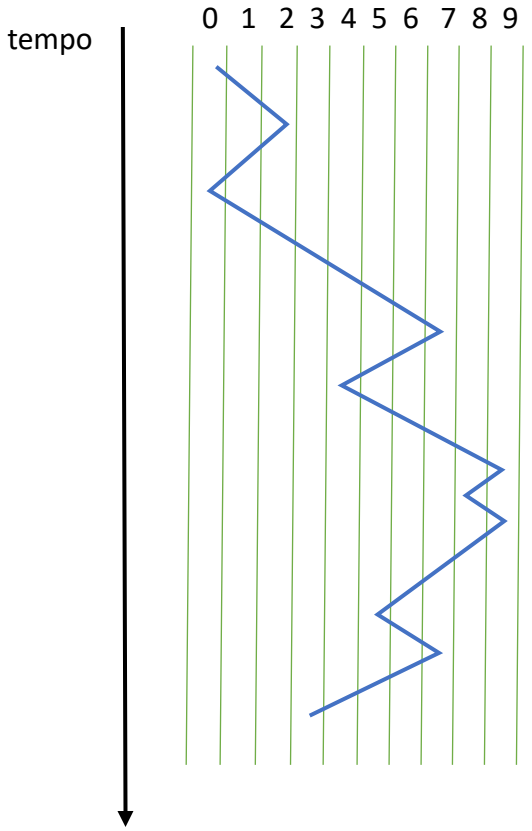
Disk scheduling

- Minimizzare i movimenti meccanici è un aspetto cruciale per massimizzare le performance di accesso a disco
- Seek time è oggetto di numerosi algoritmi di ottimizzazione
- Metodologia
  - Esecuzione sintetizzata come una sequenza di richieste identificate dal numero di traccia
- Metrica
  - numero di tracce complessivamente attraversate



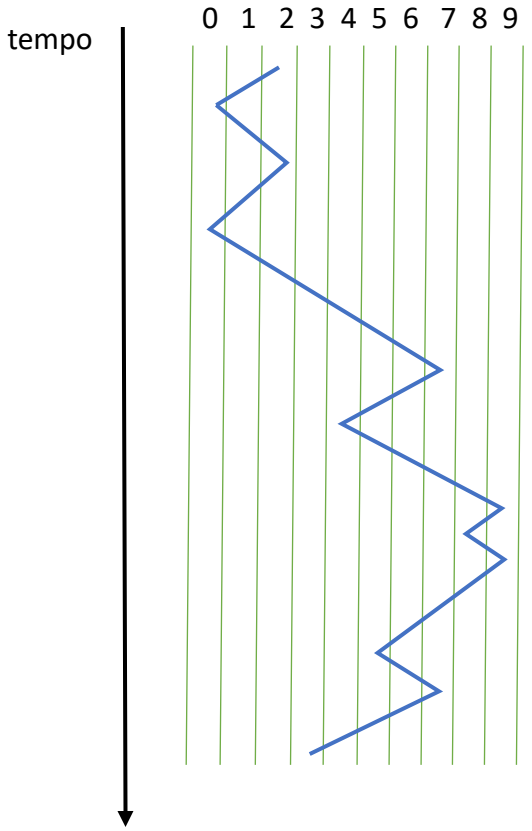
Disk scheduling

- Minimizzare i movimenti meccanici è un aspetto cruciale per massimizzare le performance di accesso a disco
- Seek time è oggetto di numerosi algoritmi di ottimizzazione
- Metodologia
  - Esecuzione sintetizzata come una sequenza di richieste identificate dal numero di traccia
- Metrica
  - numero di tracce complessivamente attraversate



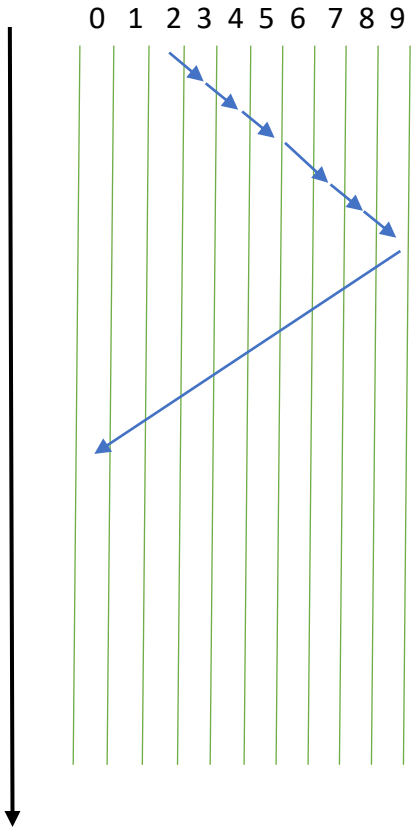
Disk scheduling

- Richieste:
  - 0,2,0,7,4,9,8,9,5,7,3
  - Posizione iniziale: 2
- FCFS
  - Richieste servite nell'ordine di arrivo
  - No starvation
  - Non minimizza seek time
  - Movimenti totali della testina: 32



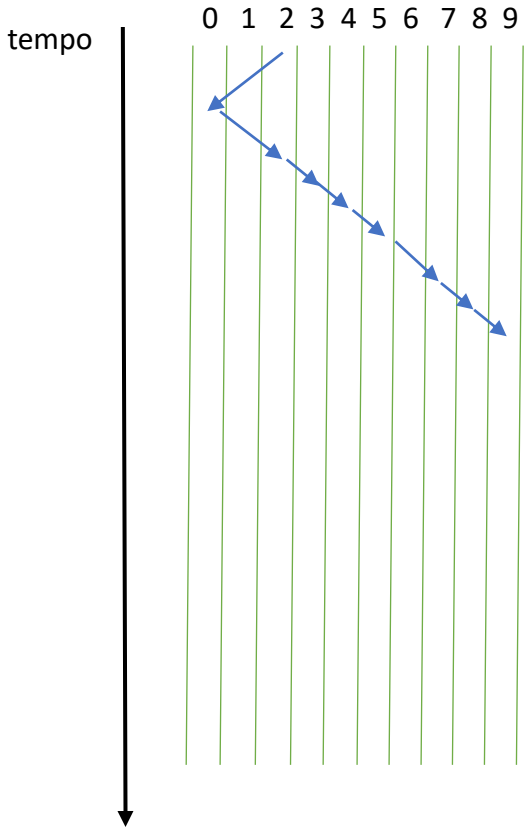
Disk scheduling

- Richieste:
  - 0,2,0,7,4,9,8,9,5,7,3
  - Posizione iniziale: 2
- Shortest-seek-time-first (SSTF)
  - Seleziona la traccia successiva che minimizza il seek time
  - 2,3,4,5,7,7,8,9,9,0,0
  - Movimenti totali della testina: 16
  - Starvation
  - Non minimizza seek time



Disk scheduling

- Richieste:
  - 0,2,0,7,4,9,8,9,5,7,3
  - Posizione iniziale: 2, direzione decrescente
- SCAN (elevator)
  - Il seek avviene in una data direzione fino a che non ci sono più richieste lungo quella direzione
  - 0,0,2,3,4,5,7,7,8,9,9
  - Movimenti totali della testina: 11
  - Performance simili a SSTF
  - Sfavorisce tracce servite in precedenza
  - Favorisce tracce agli estremi
  - Starvation



Disk scheduling

- SCAN (elevator)
  - Sfavorisce tracce servite in precedenza
  - Favorisce tracce agli estremi
  - Starvation
- CSCAN
  - Le richieste vengono soddisfatte in un'unica direzione
  - Starvation
- FSCAN
  - Due code per la gestione delle richieste
  - Una coda viene usata per lo scheduling secondo SCAN
  - Nuove richieste vengono inserite in una coda di richieste pendenti
  - Quando la coda di scheduling corrente è vuota, le due code vengono scambiate

Disk scheduling

- Gli algoritmi mostrati si focalizzano sul seek time
- Nei dischi moderni
  - il seek time è comparabile con il tempo di rotazione
  - non mostrano la posizione fisica del dato
- I produttori implementano algoritmi di scheduling nel controller del disco tesi a minimizzare sia seek time che ritardi di rotazione
- Il sistema operativo può delegare al controller se l'unico obiettivo è la performance
- Tuttavia, esistono scenari in cui alcune richieste sono prioritarie rispetto ad altre

Solid state drives

- Basati su tecnologia flash
- Le scritture deteriorano le celle flash
  - Erase
  - Program
  - Una cella flash può supportare un numero limitato di cicli Program/Erase
- La tecnologia NAND:
  - notevole densità (byte per chip)
  - Asimmetria tra unità di erase (blocco=N pagine) e program (pagina)
- Le pagine vengono gestite dal controller all'interno dell'unità SSD al fine di garantire **wear leveling**
  - I dati vengono spostati al fine di controllare operazioni di erase
- Nascondono al sistema operativo la posizione fisica dei dati
- I sistemi operativi:
  - Spesso adottano politiche di scheduling semplici (FCFS)
  - Devono notificare la cancellazione dei dati

Operazioni di I/O logico

- Necessità di astrarre dalle peculiarità del dispositivo
- Si basano su modelli di riferimento
  - Stream I/O
  - Block I/O
- Le interfacce (system call) esistono in relazione ad oggetti logici
- Per interagire con un oggetto è necessario aprire un canale di I/O
- Il sistema operativo mantiene informazioni di sessione che permettono di relazionare operazioni susseguenti sull'oggetto
- Il trasferimento di dati da/verso oggetti logico **può** riflettersi su attività che coinvolgono dispositivi hardware (e.g., disco o interfacce di rete)

File system

Concetto di file

- File: minima unità informativa archiviabile
- Il ciclo di vita di un file è scorrelato dal ciclo di vita del processo che lo ha creato
- Se il file è archiviato su un dispositivo di memoria non volatile (e.g., disco, ssd), allora il ciclo di vita del file può includere molteplici riavvii della macchina
- Un file è costituito da un insieme di record
- Un record è l'unità minima manipolabile da un'applicazione all'interno di un file

File system

- Il file system è il modulo di sistema operativo per gestire file
- Associa ad un file una serie di attributi:
  - Nome
  - Timestamp (creazione, modifica)
  - Protezione (chi può accedervi e con quali modalità)
- Gli attributi (in parte) sono memorizzati in un Record di Sistema disgiunto dal file

Operazioni su file

Creazione

- Allocazione del record di sistema e inizializzazione del file

Eliminazione

- Deallocazione del record di sistema e rilascio di tutti i record allocati per il file di interesse

Apertura

- Inizializzazione di una sessione e di un relativo indice per accessi al file nella sessione

Chiusura

- Distruzione della sessione, dei suoi metadati e dell'indice

Scrittura/Lettura

- Accesso a partire dal record a cui l'indice fa riferimento

Riposizionamento (seek)

- Aggiornamento dell'indice

Troncamento

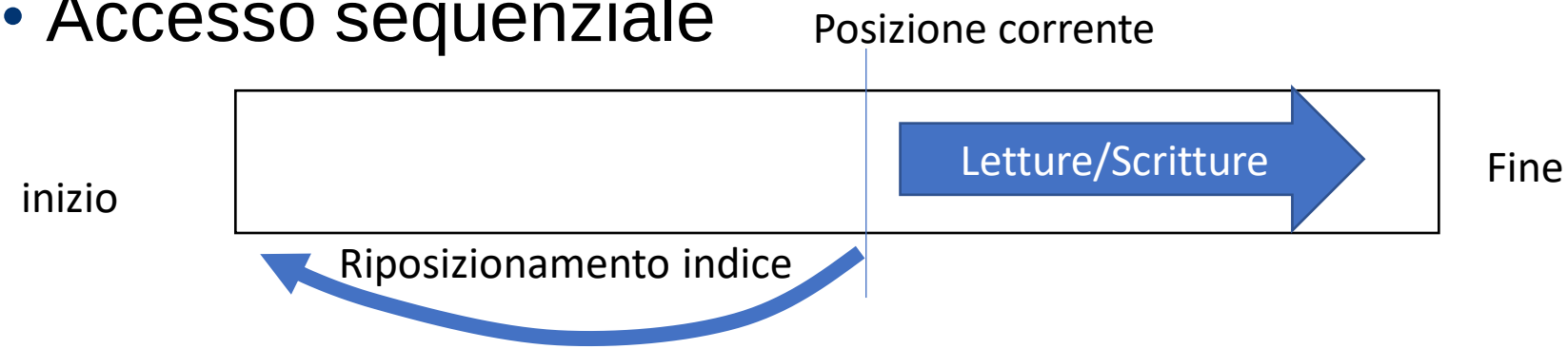
- Distruzione di record del file

Indice di accesso

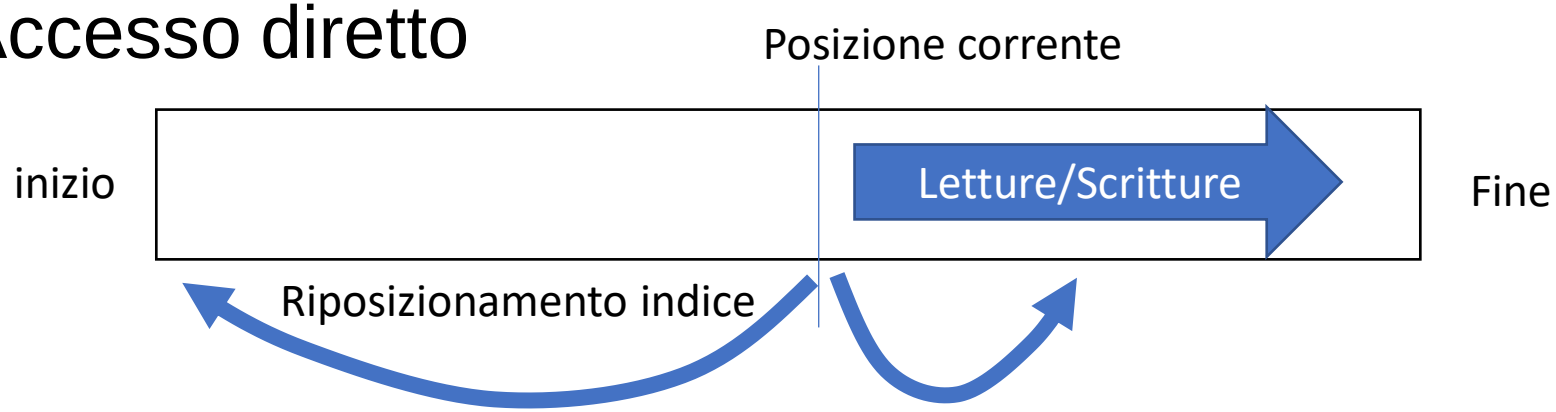
- Anche chiamato file pointer
- È un concetto di sessione
  - Non fa parte del record di sistema
- Una sessione può essere condivisa tra processi
  - Non è incluso nell'immagine di processo
- È memorizzato in una immagine di sessione
  - Struttura dati utilizzata per memorizzare tutti i metadati necessari alla gestione di accessi nell'ambito della sessione
- La modalità di aggiornamento dell'indice dipende dalle modalità di accesso ai record supportati dal file system

Metodi di accesso

- Accesso sequenziale



- Accesso diretto



- Accesso indicizzato
  - Grazie all'accesso diretto è possibile costruire indici per individuare specifici blocchi identificati da una chiave di ricerca all'interno del file

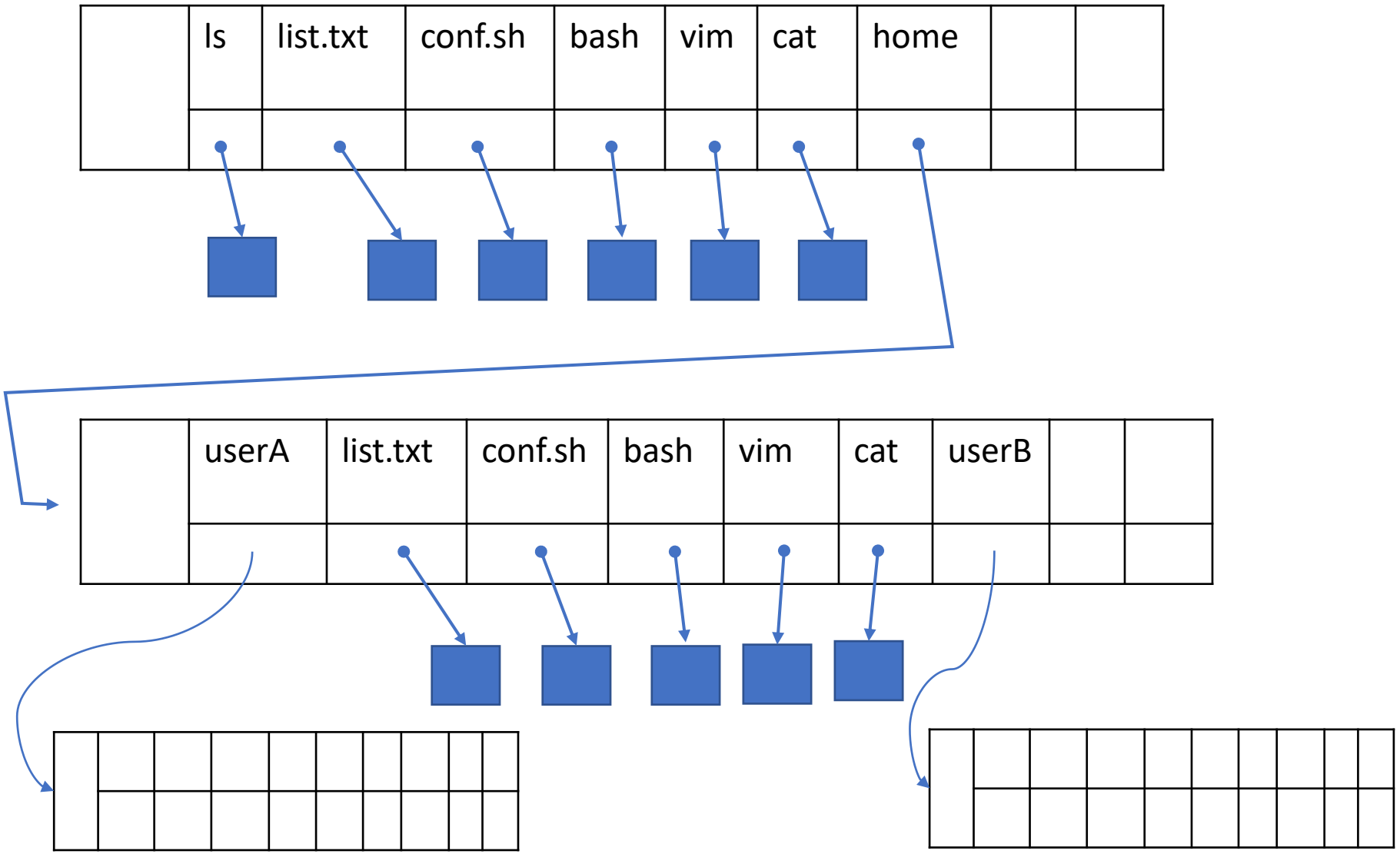
Directory

- Data un'area in cui è possibile archiviare file, il relativo file system si preoccupa di
  - Gestire l'area
  - Gestire i file in essa contenuti
- Nell'area di archivio esistono:
  - File
  - Record di Sistema
- Come ottenere l'elenco dei file archiviati in una determinata area?
  - Scansione di tutta l'area di archivio ed individuazione dei record di sistema (inefficiente)
  - Utilizzare un apposito record/file, chiamato directory, in cui memorizzare:
    - Nome del file
    - Relativo Record di sistema

Directory e gerarchie

- Data una directory, questa può contenere un solo file con un dato nome
  - Pros: semplice
  - Cons: nessuna libertà nel raggruppare e/o nominare file
- Il problema può essere risolto con una rappresentazione gerarchica delle directory
  - Albero di directory
- Una directory mantiene l'associazione tra:
  - Nome e Record di sistema (quindi file)
  - Nome e sottodirectory

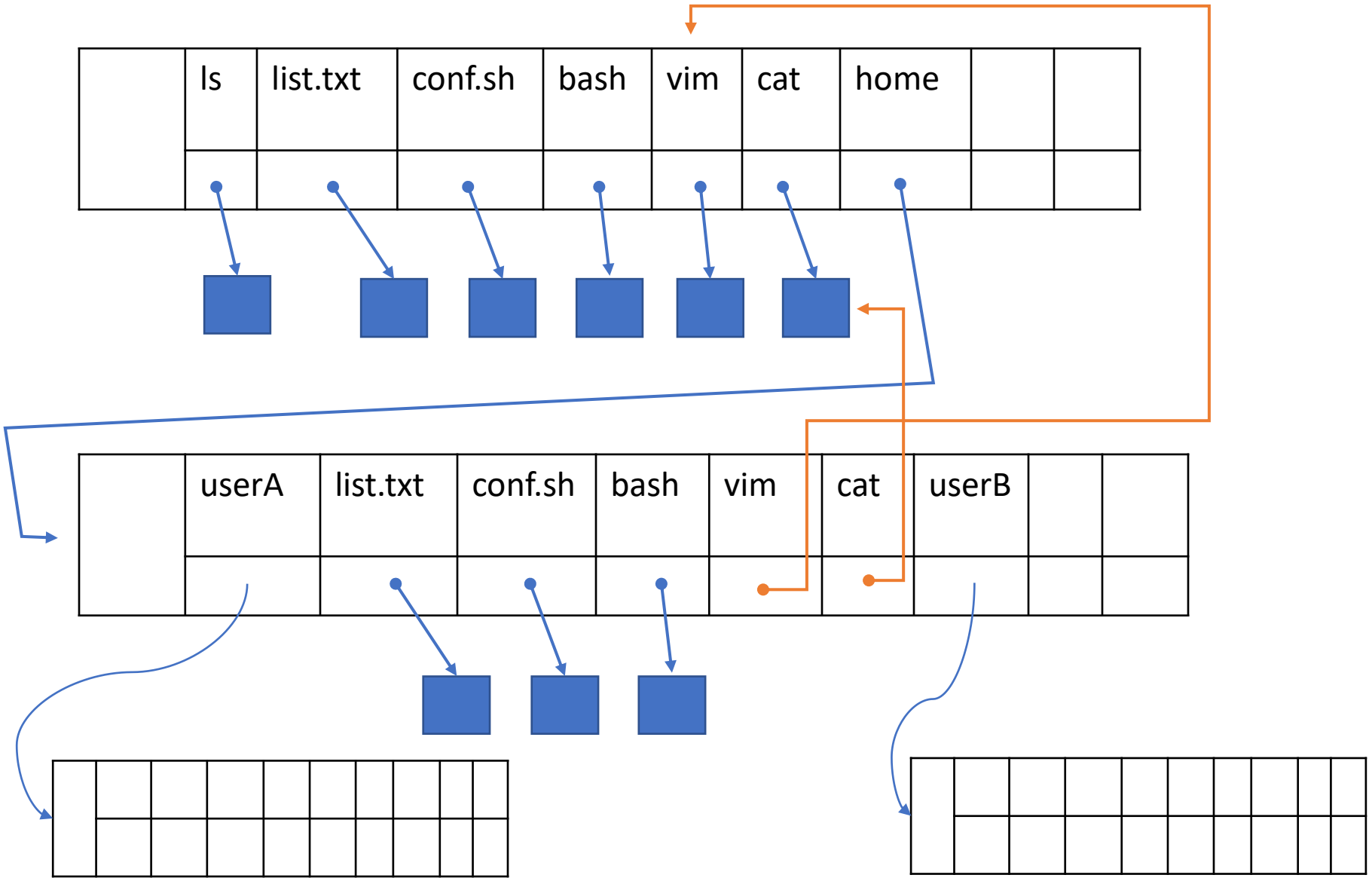
Directory e gerarchie



Directory e gerarchie

- Directory ad albero (tree-structured directory):
  - Offrono flessibilità nell'organizzare i file
  - Non possono supportare condivisione fisica di file
- Directory a grafo aciclico
  - Un Record di sistema può esser referenziato (direttamente o indirettamente) a partire da più directory

Directory e gerarchie



Directory e gerarchie

- Directory ad albero (tree-structured directory):
  - Offrono flessibilità nell'organizzare i file
  - Non possono supportare condivisione fisica di file
- Directory a grafo aciclico
  - Un Record di sistema può essere referenziato (direttamente o indirettamente) a partire da più directory
  - Una directory può essere referenziata da più directory
  - Reference count
- Directory a grafo
  - Navigazione del grafo non banale (necessario identificare cicli)
  - Garbage collection al fine di individuare file/directory non raggiungibili

Accesso e protezione

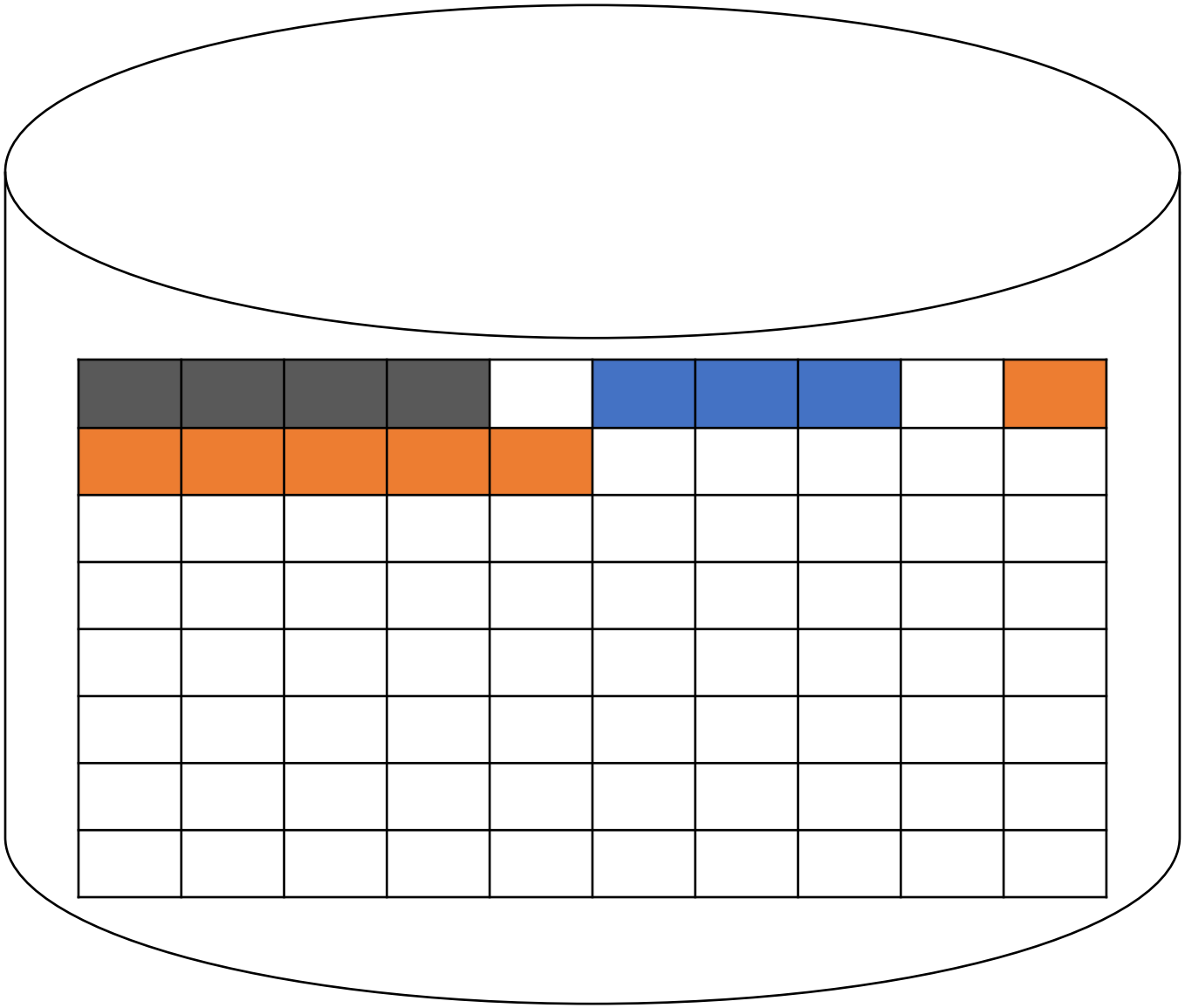
- A file e directory sono associate informazioni di protezione
  - Quali tipi di accesso sono consentiti:
    - Scrittura
    - Lettura
    - Esecuzione
  - Per quali utenti:
    - Proprietario
    - Gruppo
    - Altri
- Access control list
  - Lista di coppie <utente,permessi> per uno specifico file

Allocazione su storage

- Tipicamente l'unità di lettura/scrittura su memorie di massa è orientata al **blocco** (N byte)
- Come allocare i blocchi in un dispositivo per ospitare un file la cui taglia richiede più blocchi?
  - Allocazione contigua
  - Allocazione a catena (linked allocation)
  - Allocazione a indice

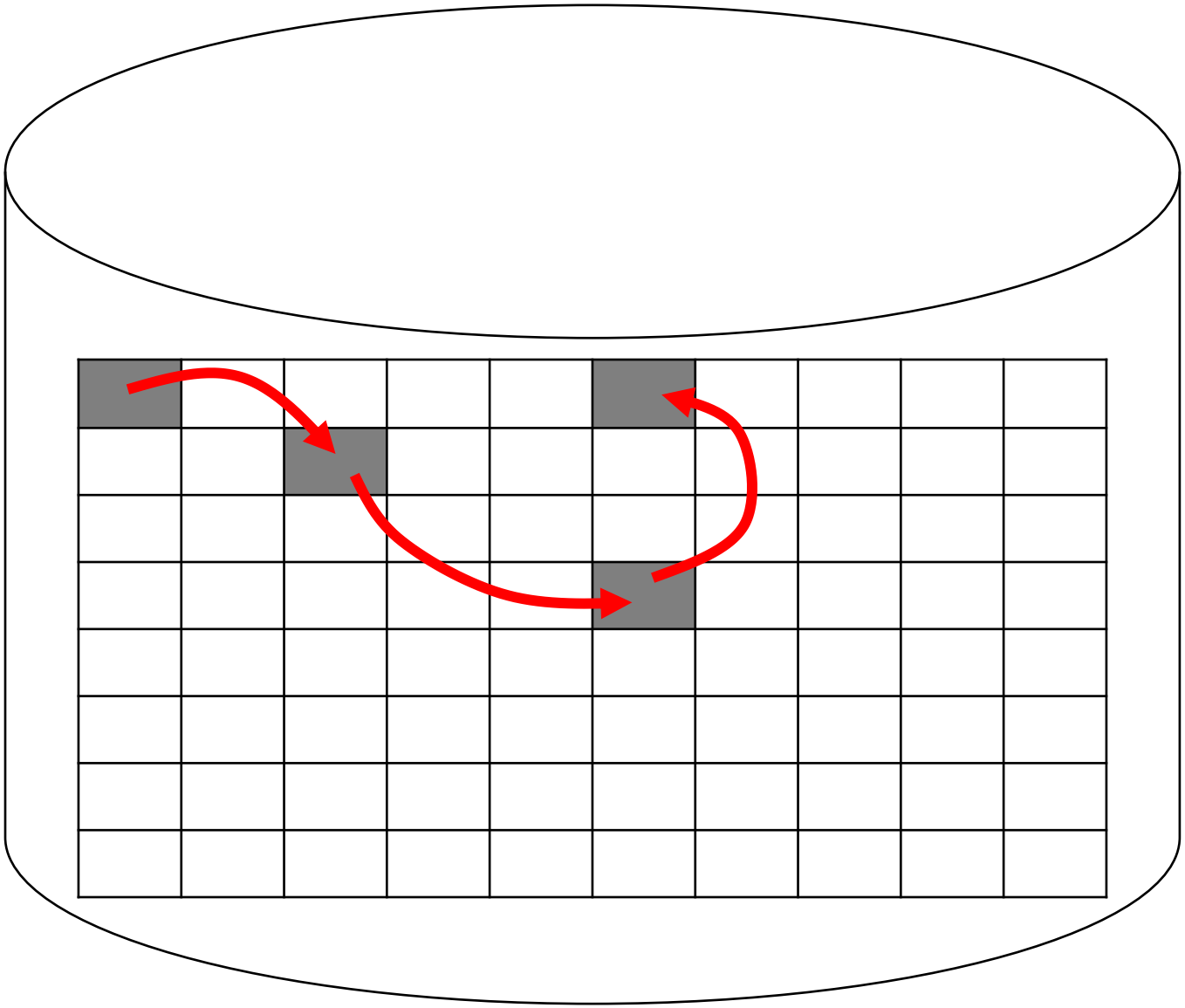
Allocazione contigua

Record di sistema
FileA 0,4
FileB 5,3
FileC 9,6



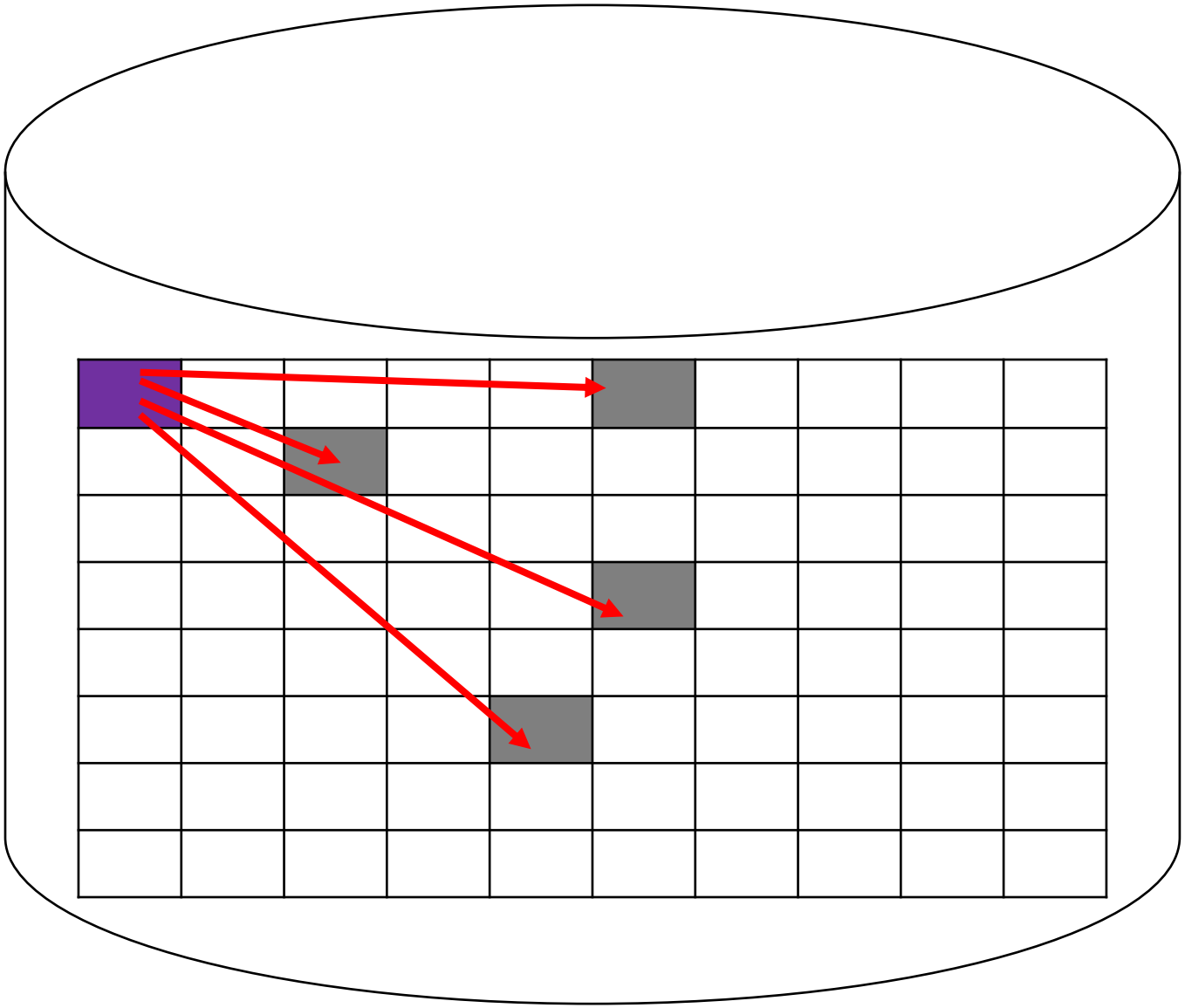
Allocazione a catena

Record di sistema
FileA 0



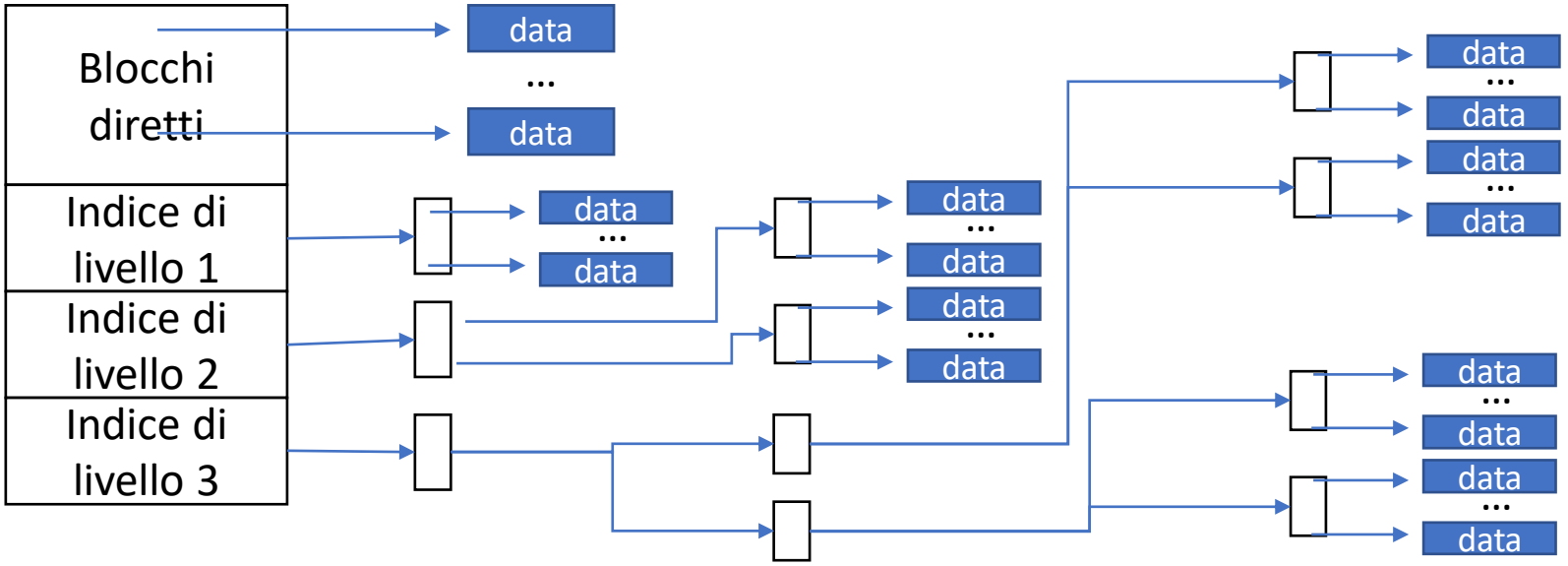
Allocazione indicizzata

Record di sistema
FileA 0



File system UNIX

- Un file è uno stream di byte (taglia di un record pari ad 1 byte)
- Accesso diretto
- Record di sistema chiamato **inode**
- Allocazione indicizzata a livelli multipli



Attributi UNIX

- Tipo di file:
  - - : file regolare
  - d : directory
  - l : link simbolico
  - b : block device
  - c : char device
- Identificatore utente e gruppo proprietari
- Permessi di accesso
  - r (4) lettura
  - w(2) scrittura
  - x(1) esecuzione
- Attributi “speciali”
  - Specifica di identificazione dinamica
    - SUID
    - SGID
  - Sticky bit (solo per directory)
    - Rimuove la possibilità di cancellare file all’interno della directory di cui non si è owner

Servizi per accesso a File

- int open(char *file_name, int option_flags [, int mode])
  - invoca la creazione/apertura di un file
  - Argomenti
    1. *file_name: puntatore alla stringa di caratteri che definisce il nome del file da aprire
    2. option_flags: specifica la modalita' di apertura (read, write etc.)
    3. mode (opzionale): specifica i permessi per owner, group, others in caso di creazione contestuale ad apertura
  - Restituzione
    - -1 in caso di fallimento,
    - un descrittore per l'accesso al file
- int close(int descriptor)
  - invoca la chiusura di un file tramite descrittore
  - Restituzione -1 in caso di fallimento

Servizi per accesso a File

- **O_RDONLY**: apertura del file in sola lettura;
- **O_WRONLY**: apertura del file in sola scrittura;
- **O_RDWR**: apertura in lettura e scrittura;
- **O_APPEND**: apertura del file con puntatore alla fine del file; ogni scrittura sul file sarà effettuata a partire dalla fine del file;
- **O_CREAT** : crea il file con modalità d'accesso specificate da mode sole se esso stesso non esiste;
- **O_TRUNC** : elimina il contenuto del file se esso gia' esistente.
- **O_EXCL** : (exclusive) serve a garantire che il file sia stato effettivamente creato dalla chiamata corrente.

Servizi per accesso a File

- ssize_t read(int descriptor, char *buffer, size_t size)
- ssize_t write(int descriptor, char *buffer, size_t size)
- off_t lseek(int fildes, off_t offset, int whence);
  - SEEK_SET
  - SEEK_CUR
  - SEEK_END

File descriptor

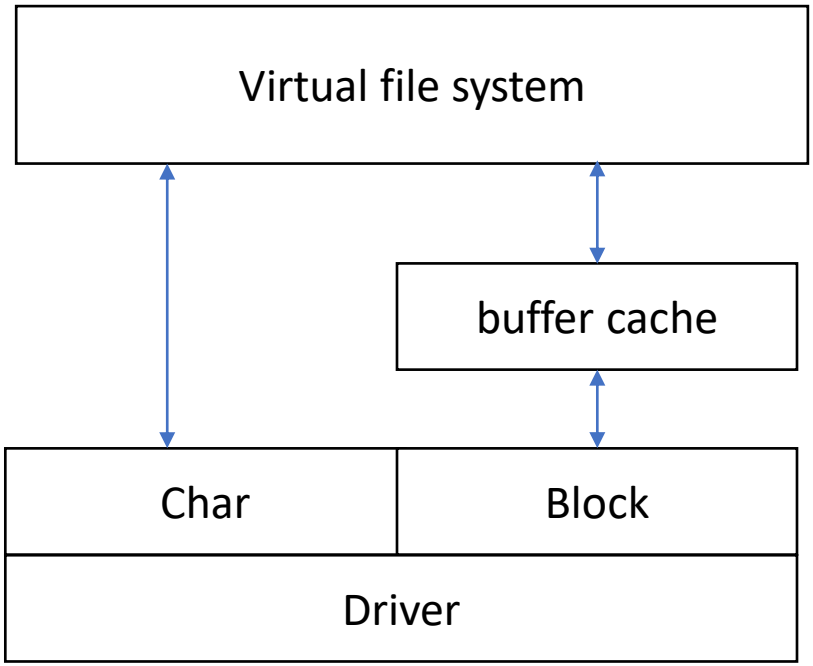
- Alcuni descrittori particolari:
  - 0: standard input
  - 1: standard output
  - 2: standard error
  - Associati a specifici dispositivi di I/O
- I descrittori vengono ereditati da processi creati tramite fork
- Duplicazione di descrittori
  - int dup(int descriptor)
  - int dup2(int oldfd, int newfd)

Altri servizi di gestione

- Link:
  - hard: int link(const char *oldpath, const char * newpath)
  - soft: int symlink(const char *oldpath, const char *newpath)
- Directory:
  - int mkdir(const char *pathname,mode_t mode)
  - int rmdir(const char *pathname)
- Permessi
  - int chmod(const char *path, mode_t mode);
  - int fchmod(int fildes, mode_t mode);
- Proprietà
  - int chown(const char *pathname, uid_t owner, gid_t group);
  - int fchown(int fd, uid_t owner, gid_t group);

I/O in sistemi UNIX

- I dispositivi sono gestiti come file
- Inode possono essere associati a dispositivi
- Le effettive operazioni dipendono dall'entità associata all'inode



Altri servizi offerti da file system

- File locking
  - utile per limitare il numero di processi concorrenti che possono manipolare il file
  - Diversi approcci
    - mandatory (Windows)
    - advisory (UNIX via fcntl)
- Gestione di errori
  - Controlli di consistenza
    - Rileva eventuali problematiche (e.g., non allineamenti tra metadati e directory)
    - Alcune inconsistenze potrebbero essere irreparabili
  - Journaling
    - creazione di log delle attività su file system (transazioni)
    - all'occorrenza replay/abort delle transazioni

In-memory file system (Linux)

- ramdisk
  - creazione di un dispositivo a blocchi sintetico, su cui è installato un file system noto
  - spreco di memoria e cpu-clock
    - e.g., ai dispositivi a blocchi viene associata una RAM cache
- ramfs
  - File system che sfrutta il meccanismo di RAM cache verso il disco
  - I blocchi in cache non vengono mai riportati su disco
  - I blocchi sono sempre marcati come dirty per evitarne il replacement da algoritmi di VM
  - La memoria dedicata non è swappabile
  - Non è possibile specificare una taglia massima
- tmpfs
  - come ramfs ma swappabile ed è possibile specificare taglia massima

File management in C - Esempio 1

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#define BUFSIZE 250
#define abort(msg) do{printf(msg);exit(1);}while(0)

int main(int argc, char *argv[]) {
    int ifd, ofd, size_r, size_w, written = 0, end = 0;
    char buffer[BUFSIZE];

    /* check parameters */
    if (argc != 3) abort("usage: copy <source> <target>\n");

    /* open the input file and check errors */
    ifd=open(argv[1],O_RDONLY);
    if (ifd == -1) abort("input open error\n");
```

File management in C - Esempio 1

```
    /* opend output file and check errors */
    ofd=open(argv[2],O_WRONLY|O_CREAT|O_TRUNC,0660);
    if (ofd == -1) abort("output creation error\n");
    while(!end){
        /* read up to BUFSIZE from input file and check errors */
        size_r=read(ifd,buffer,BUFSIZE);
        if (size_r == -1) abort("read error\n");
        /* has EOF been reached? */
        end = size_r == 0;
        /* write BUFSIZE to destination file */
        size_w = write(ofd,buffer,size_r);
        if (size_w == -1) abort("write error\n");
        printf("written: %d\n", size_w);
    }
    /* close file descriptors */
    close(ifd);
    close(ofd);
}
```

File management in C - Esempio 2

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#define FILE_NAME "log.txt"
#define STDOUT 1
#define abort(msg) do{printf(msg);exit(1);}while(0)
int main() {
    int ofd;
    /* opend output file and check errors */
    ofd=open(FILE_NAME,O_WRONLY|O_CREAT|O_TRUNC,0660);
    if (ofd == -1) abort("output creation error\n");
    close(STDOUT);  /* close standard output */
    ofd = dup(ofd); /* duplicate file descriptor */
    if (ofd == -1) abort("dup failed\n");
    execlp("ls","ls",NULL); /* run 'ls' */
}
```