Sistemi Operativi

Laboratorio – linea 2

4

Matteo Re

Dip. di Informatica
Università degli studi di Milano

matteo.re@unimi.it

http://homes.di.unimi.it/re/solabL2.html

Lezione 4:

Memorie di massa

Due tipi principali di dispositivi di memoria secondaria:

Direct Access Storage Devices (DASDs)

1) dischi magnetici:
   Hard disks (alta capacità, basso costo, veloci)
   Floppy disks (bassa capacità, bassissimo costo, lenti)
2) dischi ottici
   CD-ROM

Serial Devices
Nastri magnetici (accesso **sequenziale** estremamente veloce)

Comparazione tempi di accesso

- **Random access memory (RAM)**
  - capacità tipica è dell'ordine dei Gb
  - tempi di accesso tipici: 5ns – 60ns

- **Memoria di massa: dischi magnetici/ottici**
  nastri magnetici
  - capacità tipica è dell'ordine delle centinaia di Gb (arrivano anche a Tb)tempi di accesso tipici : 8ms – 12ms

Problemi inerenti alle operazioni di I/O su memoria di massa

Negli ultimi 40 anni l'incremento in velocità dei processori e della mamoria primaria hanho superato di gran lunga l'incremento di velocità dei dispositivi di memoria secondaria.

Operazioni di read/write su dispositivi di memoria secondaria sono, al momento, almeno quattro ordini di magnitudine più lenti di operazioni corrispondenti effettuate su memoria primaria.

Si presume che questo gap continuerà a crescere nell'immediato futuro.

Lo sviluppo di metodi che rendano più efficienti le operazioni di I/O su disco è un campo di ricerca molto attivo.

Unità di misura:

Spaziali:

- byte: 8 bits
- kilobyte (KB): 1024 or 210 bytes
- megabyte (MB): 1024 kilobytes or 220 bytes
- gigabyte (GB): 1024 megabytes or 230 bytes

Temporali:

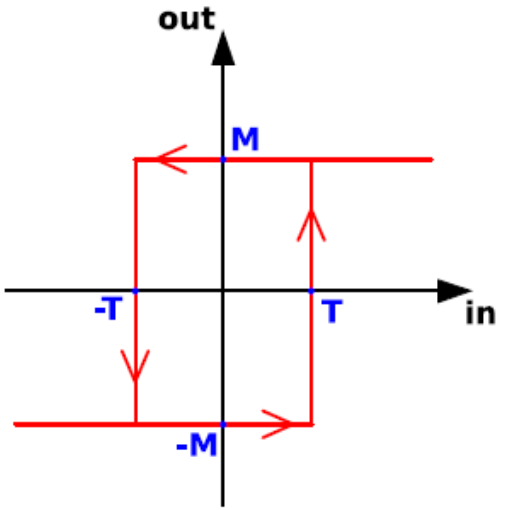
- nanosecond (ns) one- billionth (10-9) of a second
- microsecond (s) one- millionth (10-6) of a second
- millisecond (ms) one- thousandth (10-3) of a second

Primary versus Secondary Storage:

Il costo della memoria primaria (a parità di dimensione) è diverse centinaia di volte maggiore se comparato con quello della memoria secondaria ma ha tempi di accesso che sono minori di diversi ordini di grandezza.

Il disco fisso (*hard disk*) è generalmente una memoria magnetica.
Viene sfruttato il fenomeno del *ciclo di isteresi* di elementi magnetici (L'isteresi è la caratteristica di un sistema di reagire in ritardo alle sollecitazioni applicate e in dipendenza dello stato precedente).

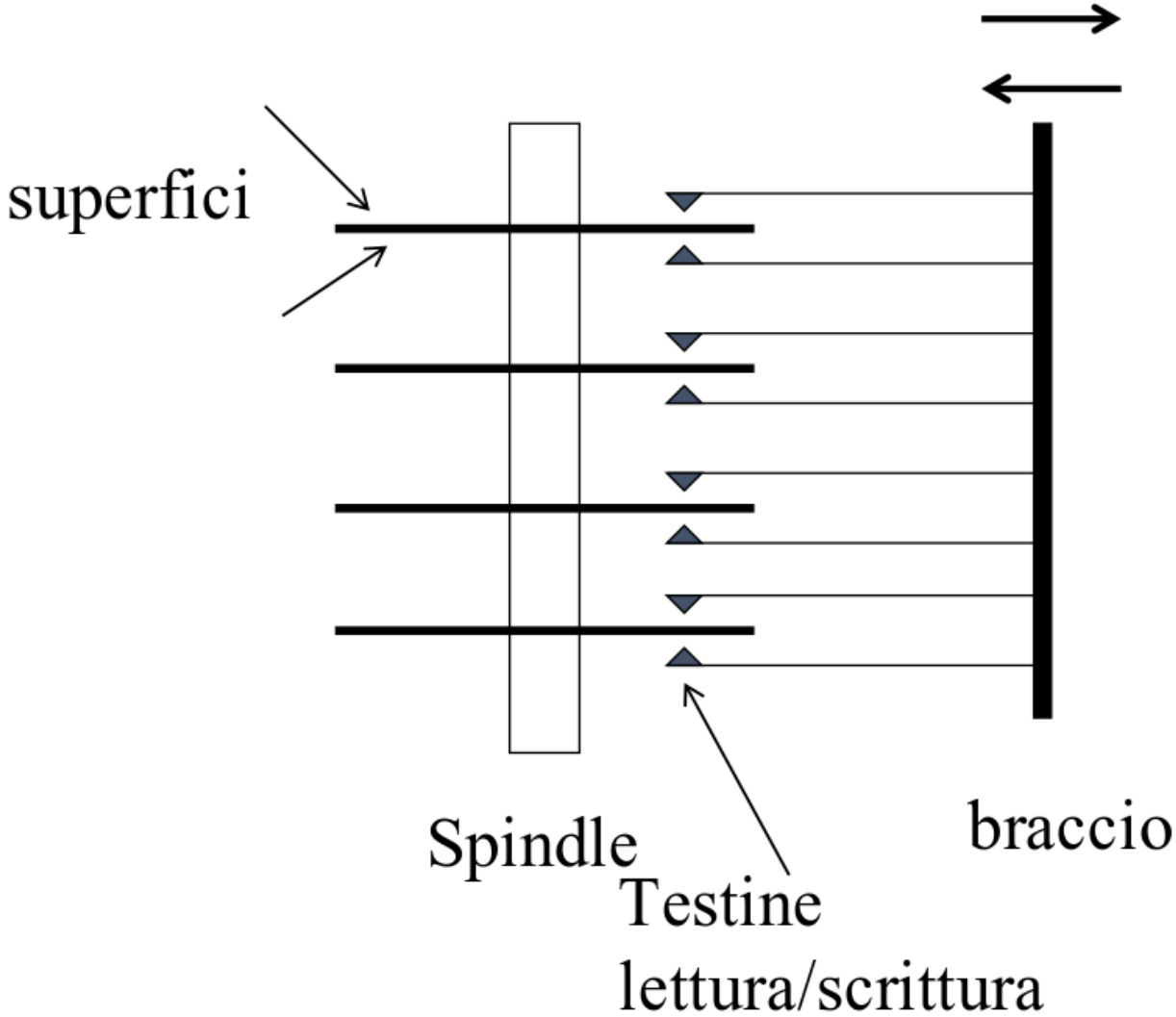
Un ciclo di isteresi può essere ottenuto anche elettronicamente (*Schmitt trigger*).
Le memorie USB, invece, sono basate su transistor NAND.



DISCHI MAGNETICI:

- Bit di dati (0 o 1) sono scritti su piatti circolari rivertiti di materiali ferromagnetici. I piatti sono detti dischi.
- Un disco ruota rapidamente (non si ferma mai)
- Delle testine leggono/scrivono bit di dati in determinate osizioni nel momento in cui queste passano sotto di esse.
- Nei disk drive dei pc sono presenti più piatti (schema nella prossima slide).

Dischi magnetici:

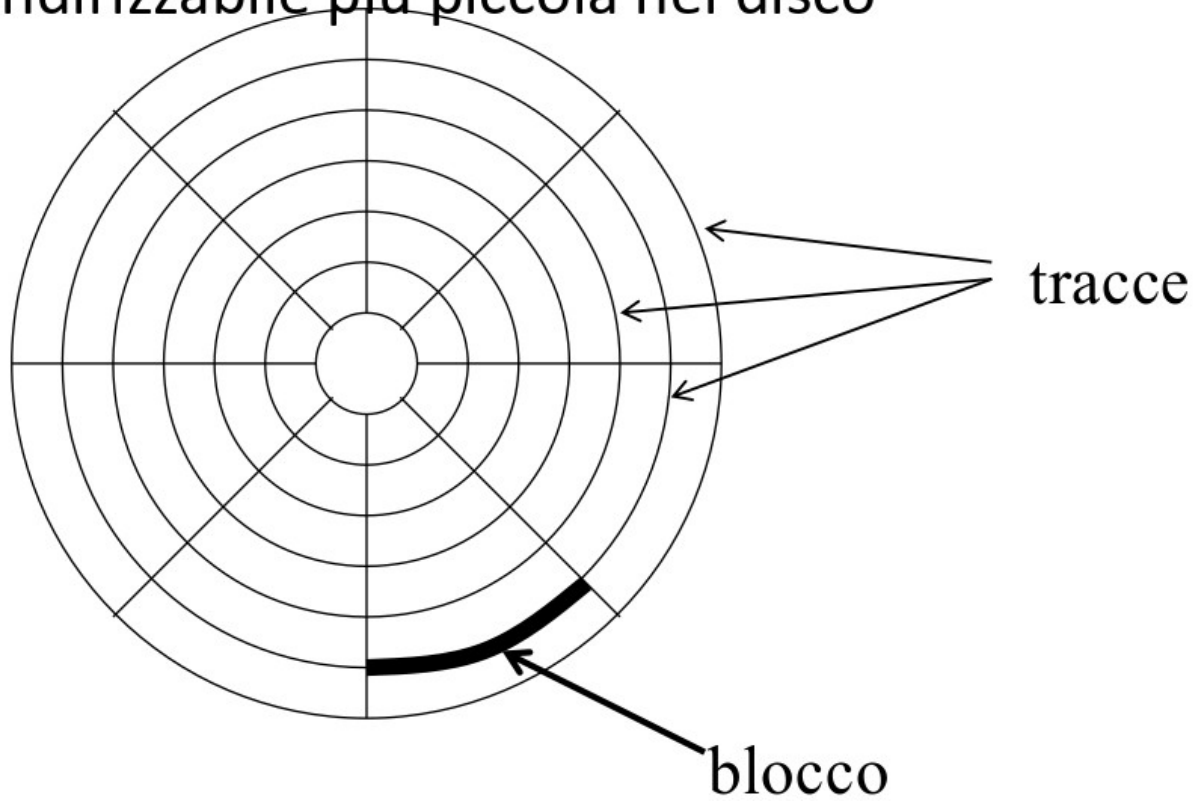


Superficie di un disco magnetico (organizzazione)

Il disco contiene **tracce** concentriche

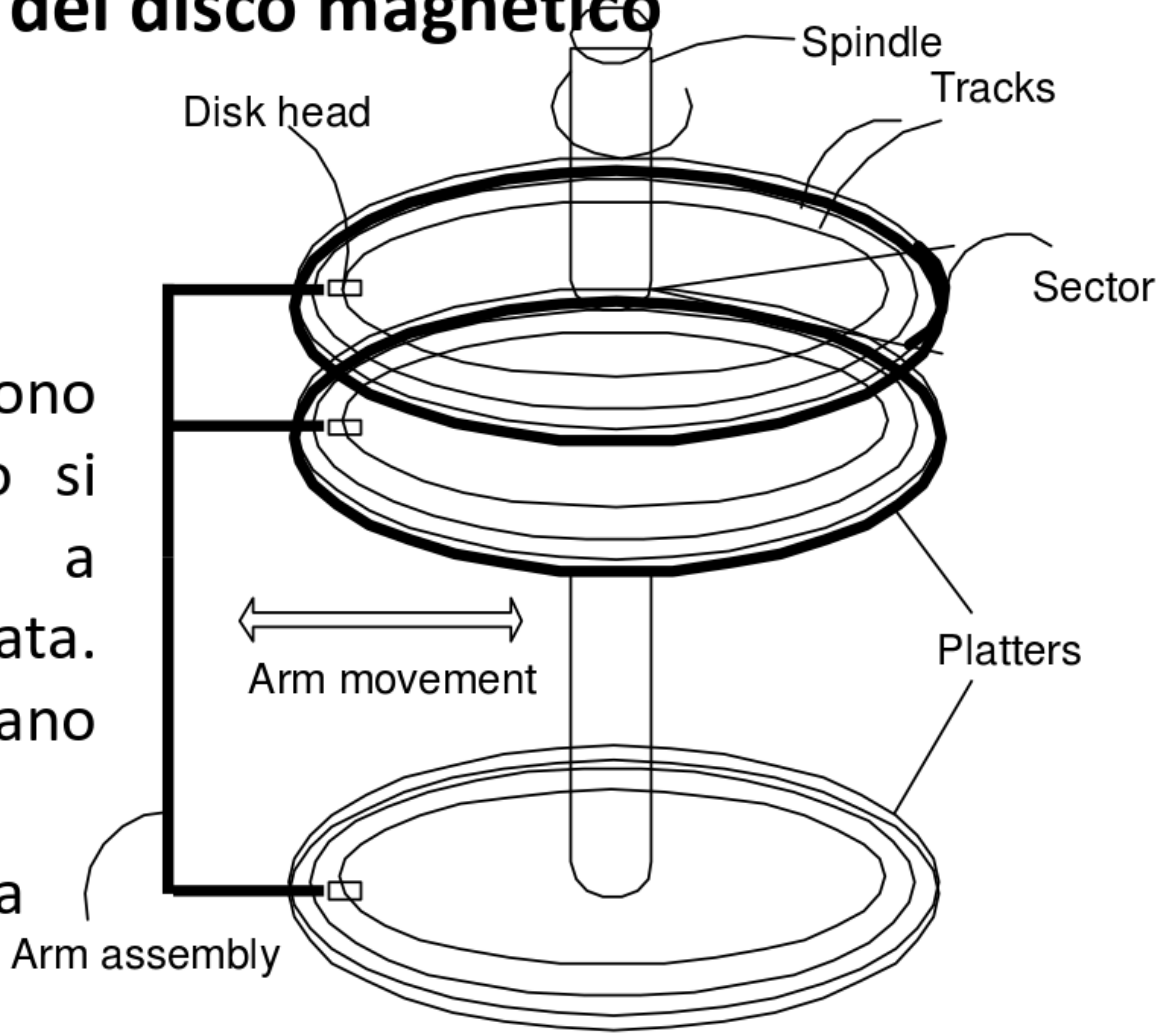
Le tracce sono divise in **settori** (spicchi)

Il **blocco** è l'unità indirizzabile più piccola nel disco



componenti del disco magnetico

- I piatti girano (es. 90rps).
- I bracci delle testine si muovono verso il centro dei piatti o si allontanano da esso fino a raggiungere la traccia desiderata. Tracce sotto le testine formano un cilindro.
- In un dato momento solo una testina legge/scrive



Controllore del disco

Incluso nel disco agisce da interfaccia tra la CPU e lo hardware del disco.

Il controllore ha una **cache interna** (tipicamente dell'ordine dei Mb) che viene utilizzata come buffer per per richieste di operazioni di lettura/scrittura.

Accesso ai dati

- Quando un programma legge un byte dal disco il s.o. localizza la sua posizione sulla superficie (traccia/settore) e legge l'intero blocco in una speciale area di memoria che funziona da buffer.

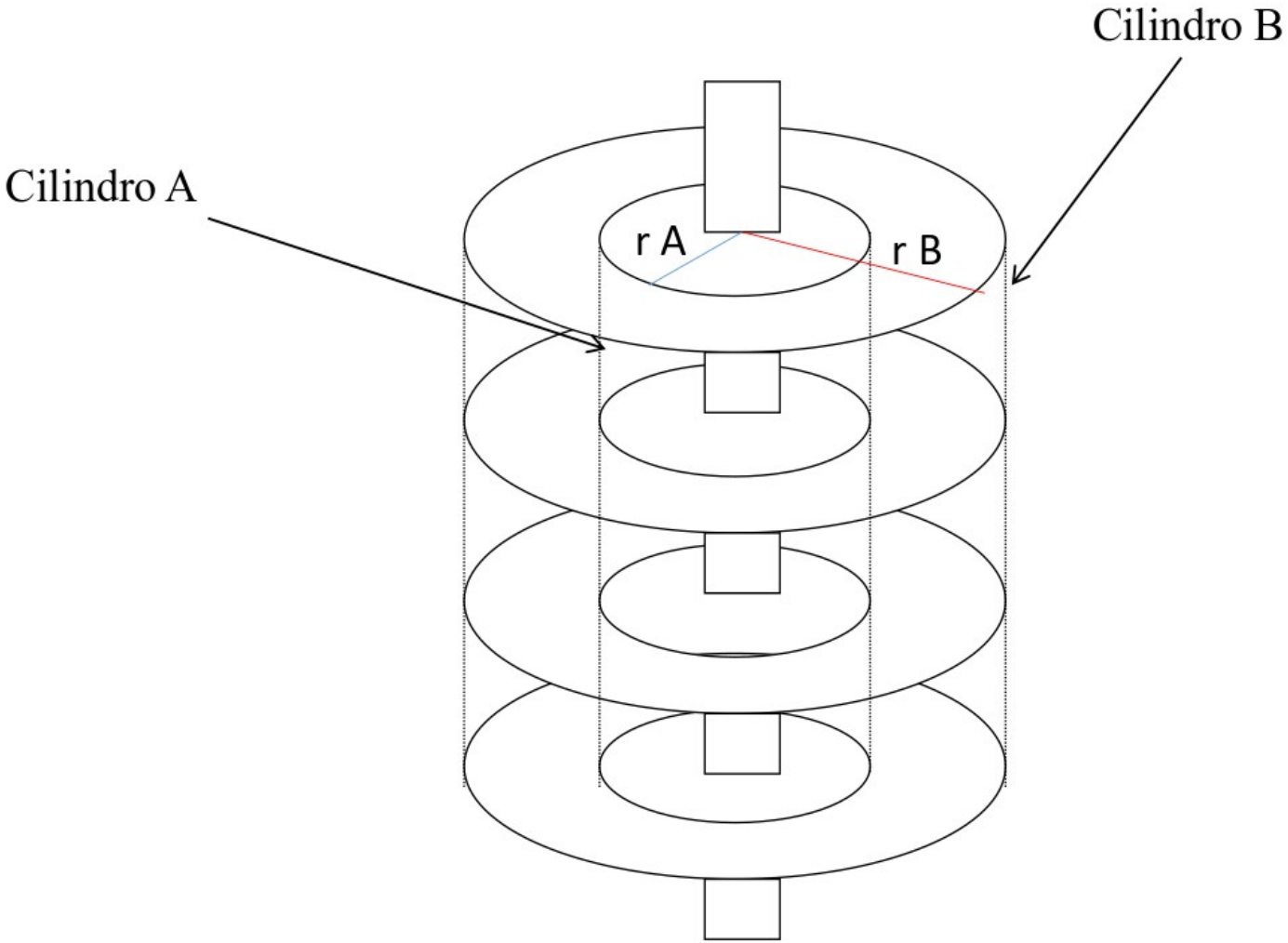
- Il collo di bottiglia nell'accesso al disco è il movimento dei bracci delle testine.
  - Ha quindi senso immagazzinare il file in tracce che occupano la medesima posizione su diversi piatti e superfici piuttosto che su diverse tracce della superficie di un singolo piatto.

Accesso ai dati : cilindri

- Un cilindro è un set di tracce posizionate ad un determinato raggio del disco (inteso come insieme di piatti)
  - Un cilindro è il set di tracce alle quali è possibile accedere **senza muovere l bracci delle testine**.

- Tutte le informazioni contenute in un cilindro possono essere lete/scritte senza muovere le testine (una volta che ci si è posizionati su di esso)

Accesso ai dati : cilindri



CALCOLO DEI BLOCCHI :

$$blocksPerPlatterSide = (cylindersPerPlatter) * (SectorsPerPlatter)$$

$$blocksPerPlatter = (blocksPerPlatterSide) * (HeadsPerPlatter)$$

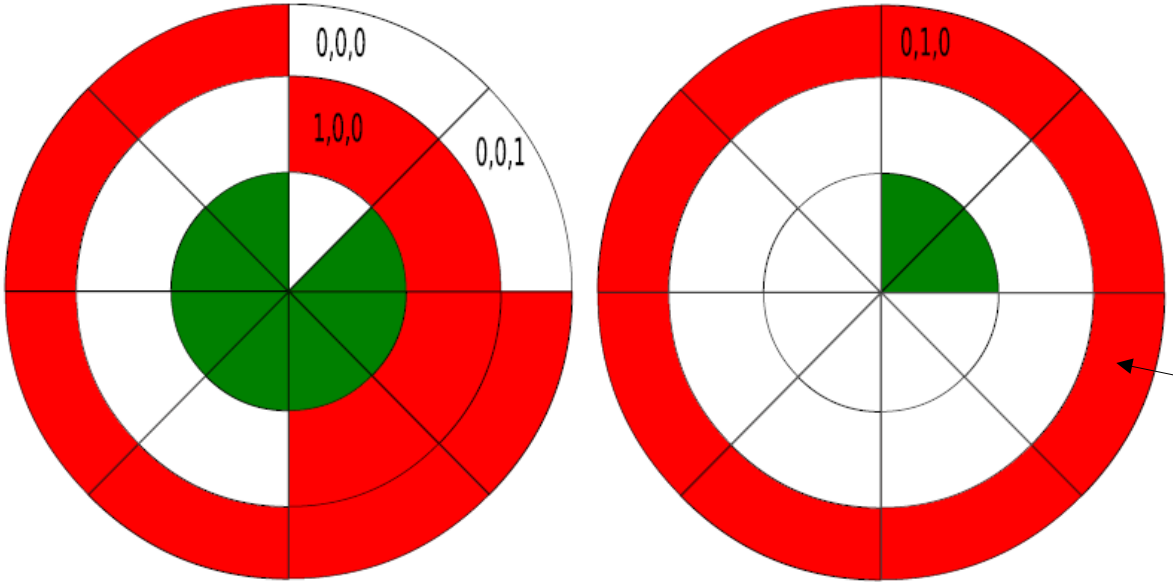
$$blocksPerPlatter = (cylindersPerPlatter) * (SectorsPerPlatter) * (HeadsPerPlatter)$$

$$blocks = (Cylinders) * (Heads) * (Sectors)$$

Example

Un floppy disk con 80 cilindri, 2 testine, 18 settori $\rightsquigarrow$ 2880

CSH :



CSH:
Cylinder-head-sector

Nell'esempio:

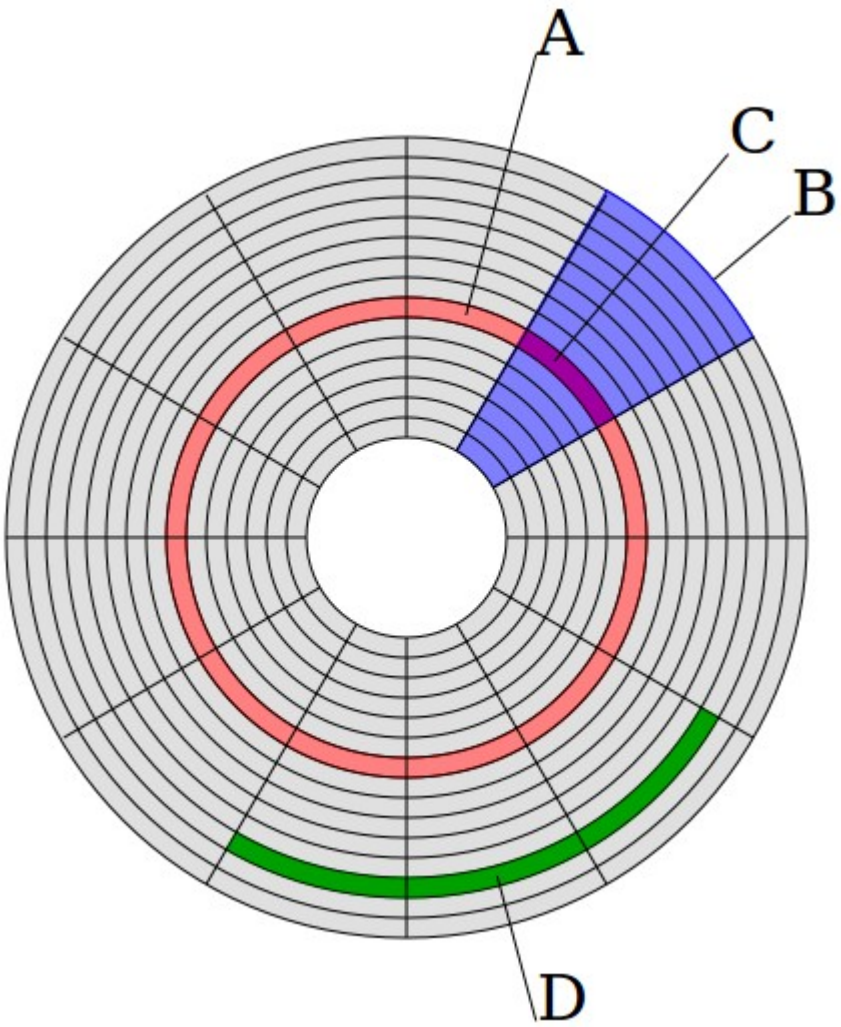
3 cilindri (C)
2 testine (H)
8 settori (S)

Totale: 48 blocchi

- $C = 3H = 2S = 8$ totale blocchi 48
- zona (partizione) rossa 0,0,2 $\rightsquigarrow$ 1,0,3

$$(1*(2*8)+0*8+3*1)-(0*(2*8)+0*8+2*1) = 19-2 = 17$$

In realtà 18 perché contiamo da zero



A

Traccia

B

Settore

C

Settore di una traccia

D

Insieme di frammenti
di tracce contigui

TEMPO DI LETTURA/SCRITTURA :

$$T = TempoDiRotazione + TempoDiRicerca + TempoDiAccesso$$

Il tempo di rotazione è detto anche *latenza*

Il tempo di ricerca (*seek time*) può essere ottimizzato con algoritmi opportuni

Ottimizzazione SEEK TIME (tempo di ricerca) :

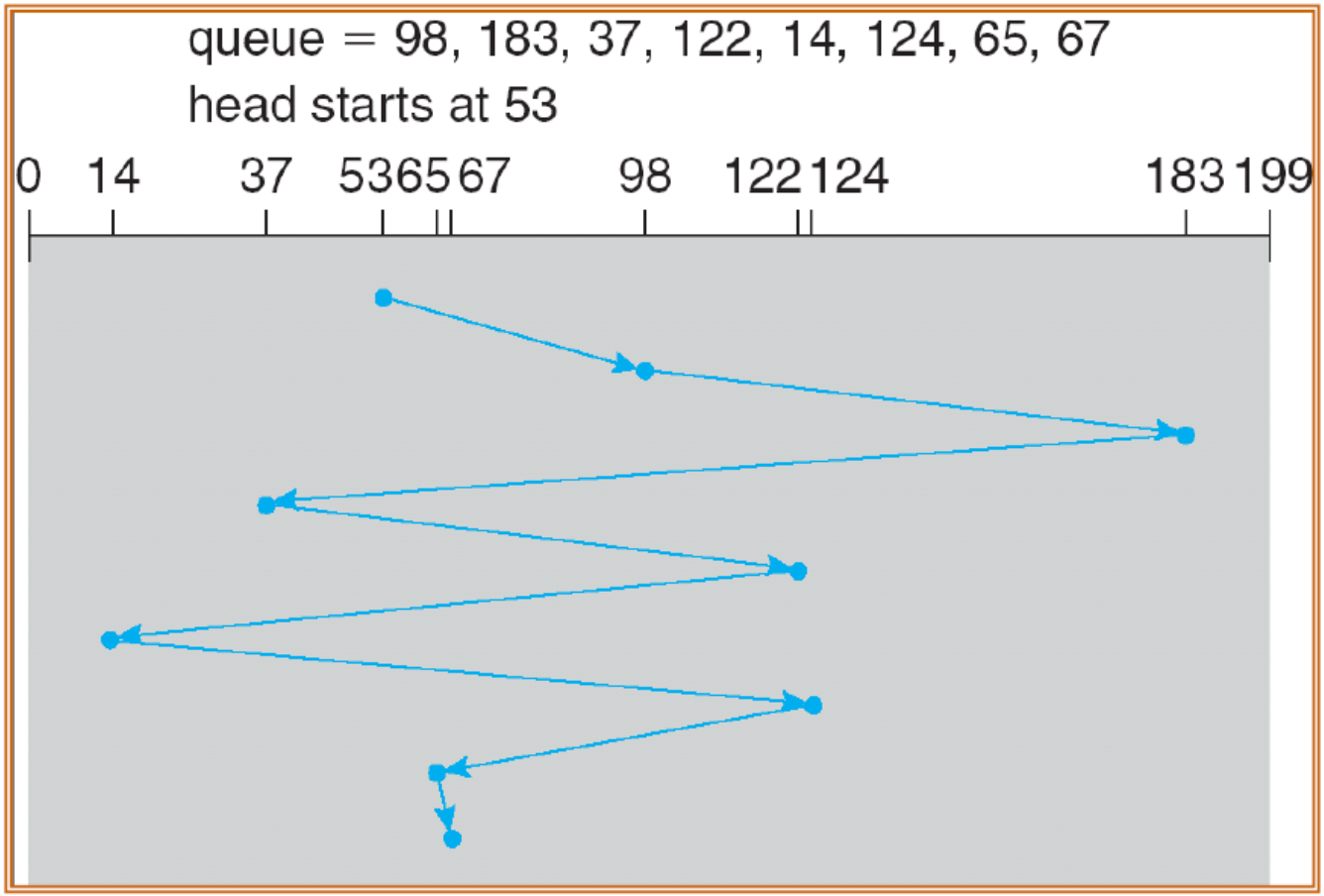
Esempio:

98 183 37 122 14 124 65 67 posizione di partenza: 53

- First Come First Served
- Shortest Seek First
- Scan / Look (Elevator)

First Come First Served (FCFS)

Il movimento totale della testina è 640 cilindri



Shortest Seek First (altname: SSTF)

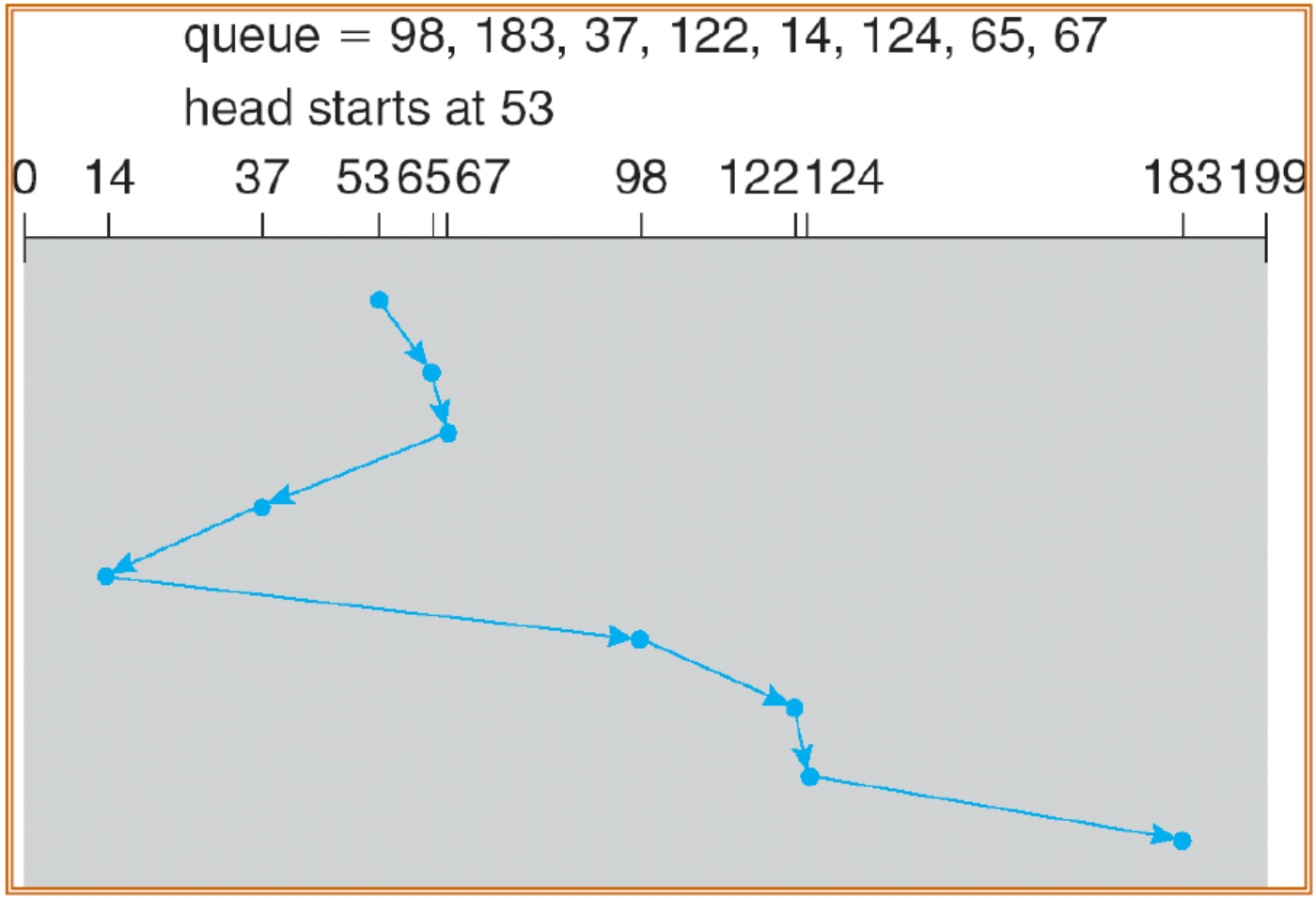
Seleziona la richiesta con il minimo tempo di ricerca (seek time) dalla posizione corrente della testina.

L'immagine (nella prossima slide) mostra che il movimento totale della testina è di 263 cilindri.

Non è ottimale (ad es. nella figura che vedremo servire prima le richieste per 53, 37 e 14 riduce il movimento totale).

SHORTEST SEEK FIRST

Il movimento totale della testina è 263 cilindri



SCAN (ascensore)

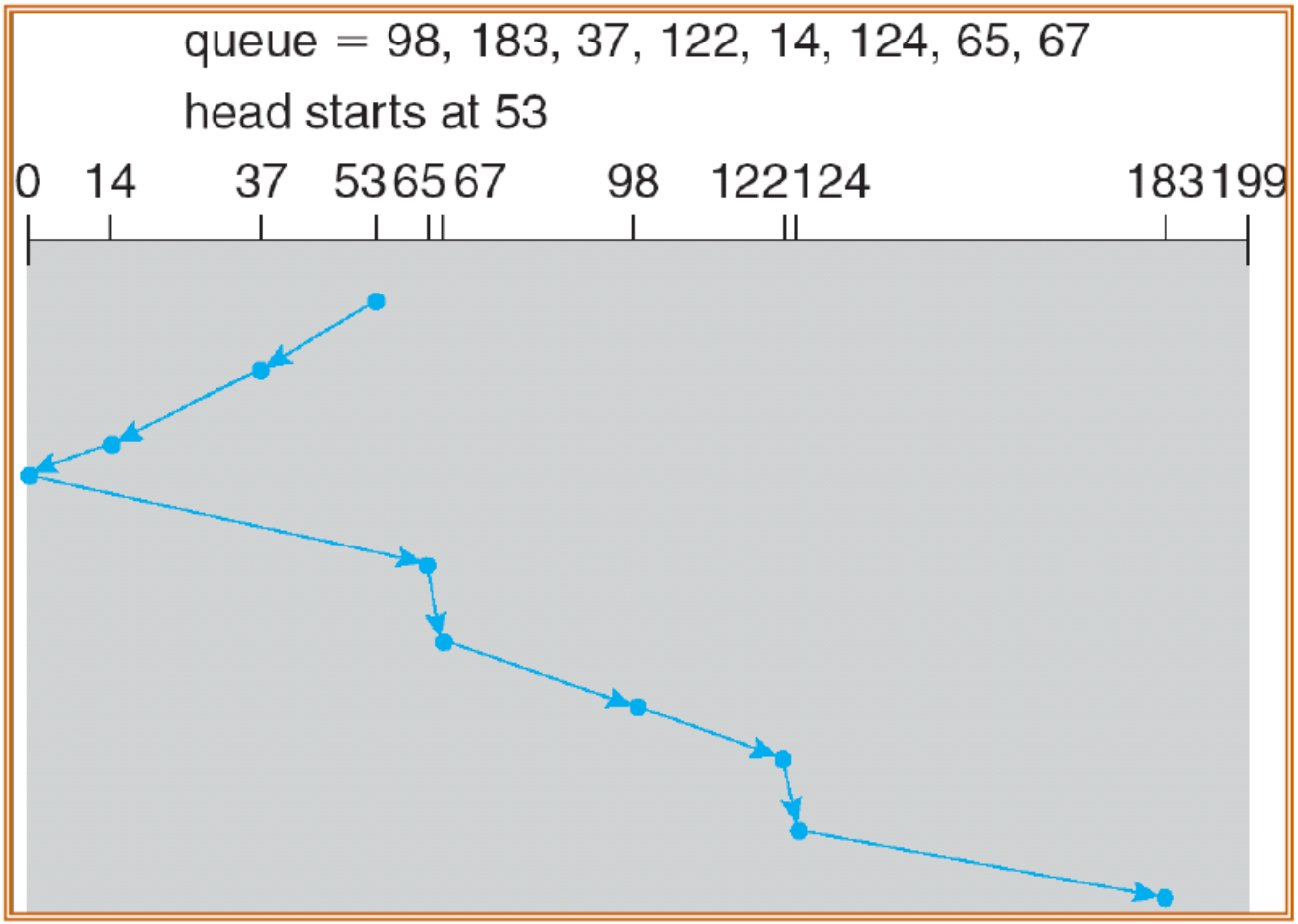
Il braccio del disco si muove da un estremo all'altro estremo, servendo le richieste che incontra. All'altro estremo il movimento viene invertito e il servizio continua.

Talvolta chiamato algoritmo dell'ascensore

L'illustrazione mostra che il movimento totale della testina è 208 cilindri.

SCAN

Il movimento totale della testina è 208 cilindri



Stima capacità :

- Track capacity = # of sectors/track * bytes/sector
- Cylinder capacity = # of tracks/cylinder * track capacity
- Drive capacity = # of cylinders * cylinder capacity
- Number of cylinders = # of tracks in a surface

ESERCIZIO:

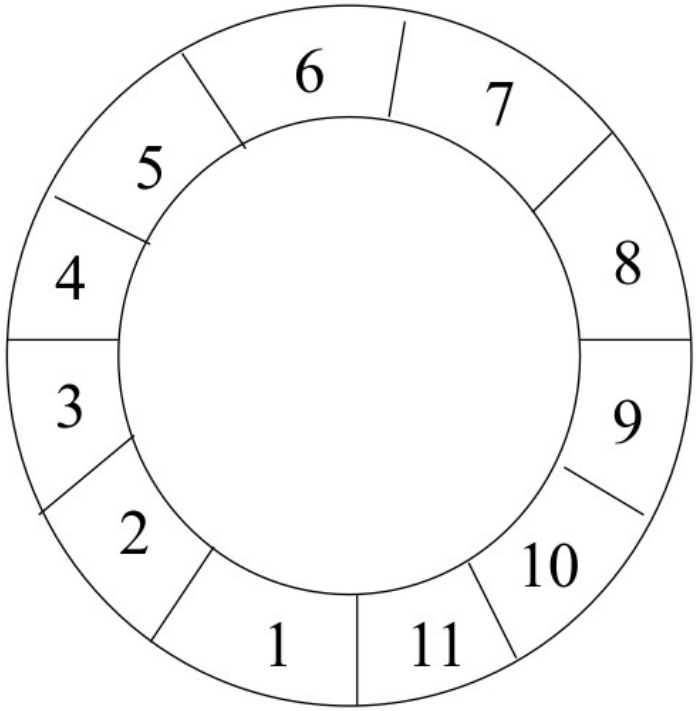
Salvare un file di 20000 record su un disco con le seguenti caratteristiche:

# of bytes per sector = 512
# of sectors per track = 40
# of tracks per cylinder = 11
# of cylinders = 1331

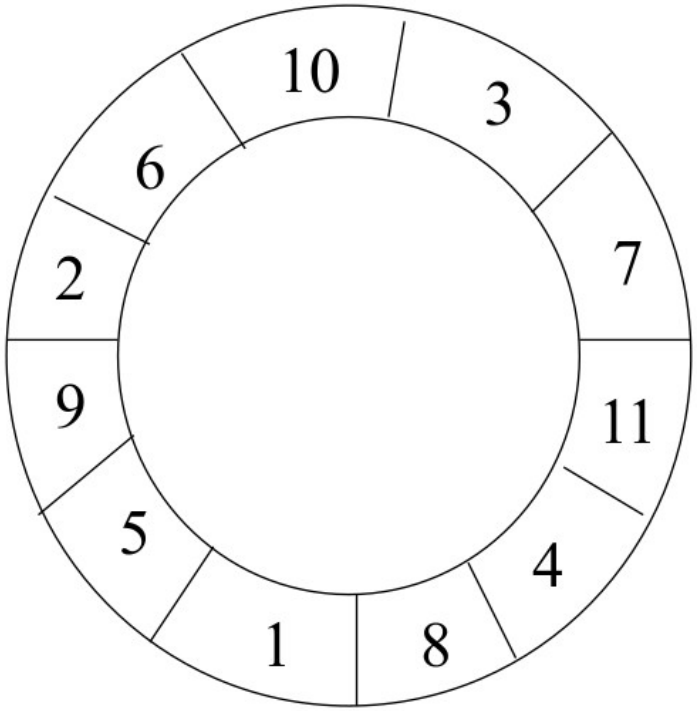
1) Quanti cilindri sono richiesti se ogni record occupa 256 byte?

2) Qual è la capacità totale del disco?

Organizzazione delle tracce mediante settori :



Physically adjacent sectors



Sectors with 3:1 interleaving

ESERCIZIO :

- Supponiamo di voler leggere consecutivamente i settori di una traccia ordinati dal primo all'ultimo : sectors 1, 2,...11.
- Suponiamo che due settori consecutivi non possano essere letti in assenza di interlacciamento.
- Quante rivoluzioni sono necessarie per leggere l'intero disco?

  - Senza interlacciamento
  - Con interlacciamento 3:1

- *Nota: Al giorno d'oggi molti controllori dei dischi sono veloci e quindi l'interlacciamento non è più così commune.*

Tempo di ricerca (SEEK TIME) :

- Tempo di ricerca è il tempo richiesto per muovere il braccio della testina sul cilindro desiderato.
- E' la componente che incide di più sul tempo di accesso.

Tipicamente:

– 5 ms per muoversi da una traccia alla successive (track-to-track)

– 50 ms tempo di ricerca massimo (per spostarsi al di fuori di una traccia quando ci si trova la suo interno)

– 30 ms tempo di ricerca medio (da una **qualsiasi** traccia ad una **qualsiasi** altra traccia)

Tempo di ricerca medio (s) :

• Il tempo di ricerca dipende unicamente dalla velocità con cui si muovono i bracci delle testine e dal numero di tracce che devono essere attraversate per raggiungere l'obiettivo.

• Data la conoscenza delle seguenti informazioni (che sono costanti per ogni specifico modello di disco) :
  – Hs = tempo richiesto perchè la testina inizi a muoversi
  – Ht = tempo richiesto perchè la testina si muova da una traccia alla successiva
• Il tempo necessario perché la testina si muova di n tracce è:
  • *Seek(n)= Hs+ Ht*n*

Latenza (latenza rotazionale) (r) 1 :

- Latenza è il tempo richiesto necessario perchè il disco ruoti in modo che il settore che ci interessa sia sotto la testina di lettura/scrittura.
- Gli hard disk ruotano a circa 5000-7000 rpm,
  - 12-8 msec per rivoluzione.
- Note:
  - Latenza minima = 0
  - Latenza massima = tempo per una intera rivoluzione del disco
  - **Latenza media (*r*)** = (min + max) / 2
    = max / 2
    = tempo ½ rivoluzione del disco

Tempo di trasferimento (1) :

- Il tempo di trasferimento è il tempo richiesto perché una testina passi attraverso un blocco.
- Si calcola usando la seguente formula:

$$\text{Transfer time} = \frac{\text{number of sectors}}{\text{track capacity in number of sectors}} \times \text{rotation time}$$

- es. Se ci sono St settori per traccia, il tempo per trasferire un settore è 1/ St * tempo 1 rivoluzione.

Tempo di trasferimento (2) :

- Il tempo di trasferimento dipende unicamente dalla velocità a cui ruotano i piatti e dal numero di settori che deve essere trasferito.

- Dato:

  – St = numero totale settori per traccia

E' possibile calcolare il tempo di trasferimento per n settori contigui sulla stessa traccia come segue:

Tempo trasferimento = (*n/St)*(*1000/*R) , *in ms*

Sistemi Operativi

Laboratorio – linea 2

4

ESERCIZIO:

Dato il seguente disco:

- 20 superfici
800 tracce/superficie
25 settori/traccia
512 bytes/settore
- 3600 rpm (revolutions per minute)
- 7 ms track-to-track seek time
28 ms avg. seek time
50 ms max seek time.

Calcolare:

- Latenza media
- Capacità del disco
- Tempo richiesto per leggere l'intero disco,un cilindro alla volta

ESERCIZIO :

60 sec per min

Average latency=(60*1000/3600)/2=8.33 ms
Disk capacity=20*800*25*512=193.31MB
Time to read the entire disk=20*800*7=112 sec

Componenti costo (temporale) di accesso ai dati nel disco magnetico :

Il tempo di accesso ad un settore di una traccia è composto da tre componenti principali:

Time Component	Action
Seek Time	Time to move the read/write arm to the correct cylinder
Rotational delay (or latency)	Time it takes for the disk to rotate so that the desired sector is under the read/write head
Transfer time	Once the read/write head is positioned over the data, this is the time it takes for transferring data

ASTRAZIONI FORNITE DAL S.O. :

Labbiamo visto come sono costruiti e come funzionano dal punto di vista fisico le memorie di massa.

Quali sono le astrazioni fornite dal sistema operativo che permettono di utilizzarle?

FILE SYSTEM
FILE
DIRECTORY

DEVICE LOGICO :

L'astrazione fornita dal s.o. per il disco e quella del device a blocchi. Il blocco è un blocco logico, potenzialmente diverso dal blocco fisico.

I device a blocchi sono file speciali, identificati da :

Major number: identica la categoria del device (disco IDE, floppy)

Minor number: numero d'ordine del device all'interno di una categoria

File speciali e mknod :

I file speciali si creano con /usr/bin/mknod generalmente in /dev

- Device a blocchi b
- Device a caratteri c
- Named pipe p (non ha major e minor)

PARTIZIONI :

Lo spazio di memoria di uno hard-disk e ripartito in porzioni indipendenti (partizioni): in linea di principio possono contenere anche sistemi differenti. Generalmente contengono sotto-file-system il cui backup e/o aggiornamento e indipendente.

Partition table sector (PTS): Contiene la descrizione di 4 partizioni
(primarie) agli offset 446, 462, 478, 494

Partizione: Una zona contigua del disco (CHS)

Partizione estesa: Una partizione che permette una nuova suddivisione (partizioni logiche) grazie ad un nuovo PTS

PARTIZIONI :

Lo spazio di memoria di uno hard-disk e ripartito in porzioni indipendenti (partizioni): in linea di principio possono contenere anche sistemi differenti. Generalmente contengono sotto-file-system il cui backup e/o aggiornamento è indipendente.

Partition table sector (PTS): Contiene la descrizione di 4 partizioni (primarie) agli offset 446, 462, 478, 494

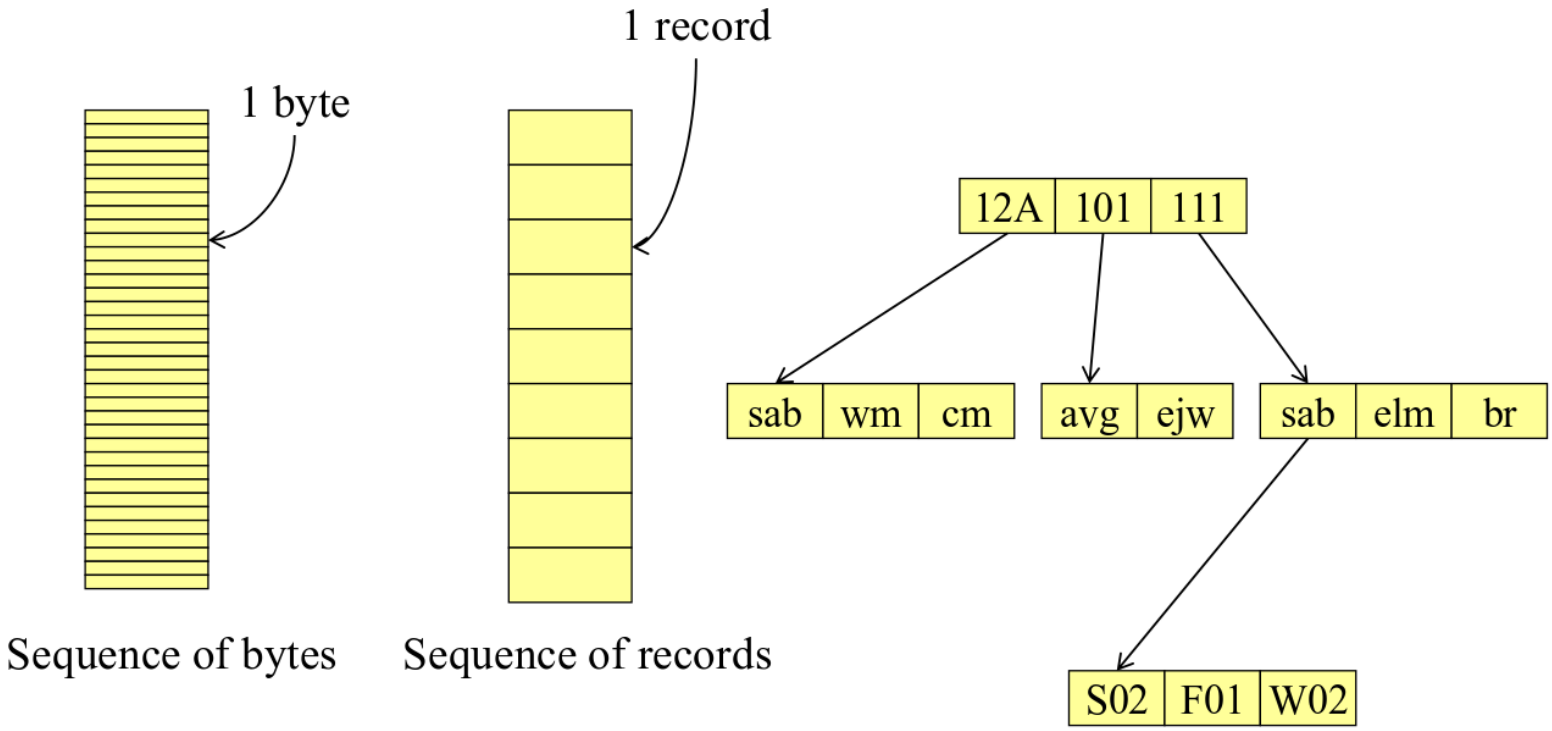
Partizione: Una zona contigua del disco (CHS)

Partizione estesa: Una partizione che permette una nuova suddivisione (partizioni logiche) grazie ad un nuovo PTS

PARTITION TABLE :

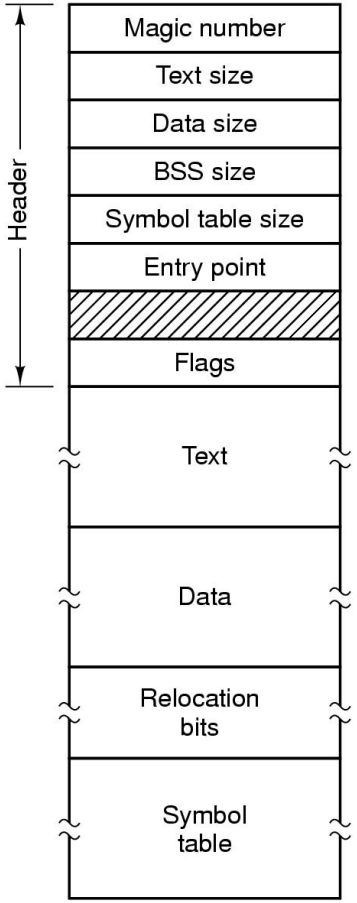
```
struct partition {
char active;
char begin[3];
char type;
char end[3];
int start;
int length;
};
```

FILE:


1 byte
1 record
12A
101
111
sab
wm
cm
avg
ejw
sab
elm
br
S02
F01
W02
Sequence of bytes
Sequence of records

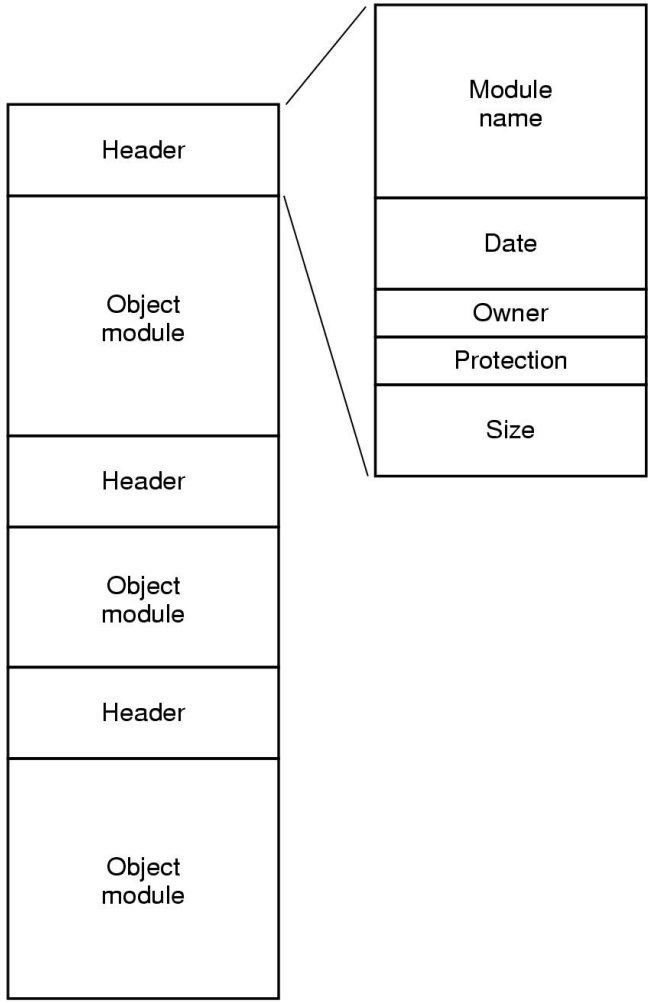
TIPI DI FILE:

Eseguibile



(a)

Archivio



(b)

TIPI DI ACCESSO AI FILE:

Accesso sequenziale

- Lettura di tutti I byte/record in sequenza
- Non è possibile saltare da una posizione all'altra
- E' possibile "riavvolgere" (ritornare ad una posizione precedente e ripartire)
- Conveniente se stiamo leggendo da nastro magnetico
- Utile quando vogliamo accedere a **interi** file

Accesso random

- Byte o record letti in qualsiasi ordine
- E' possibile spostarsi in una data posizione ed iniziare a leggere/scrivere

FILE: ATTRIBUTI

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

DIRECTORY:

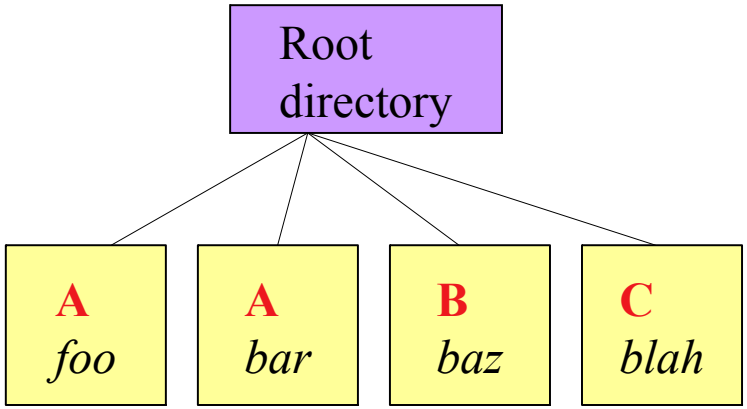
Agli esseri umani piace raggruppare oggetti secondo una data logica

I file system permettono questa operazione mediante le directory (conosciute anche come “cartelle”, folder)

Raggrupare oggetti rende più semplice:

- Trovare I file. Basta ricordarsi in quale directory sono contenuti
- Trovare file correlati (non mi ricordo più il nome di un file ma mi ricordo che stava nella stessa directory di …)

(ipotetico) Sistema a singola directory:



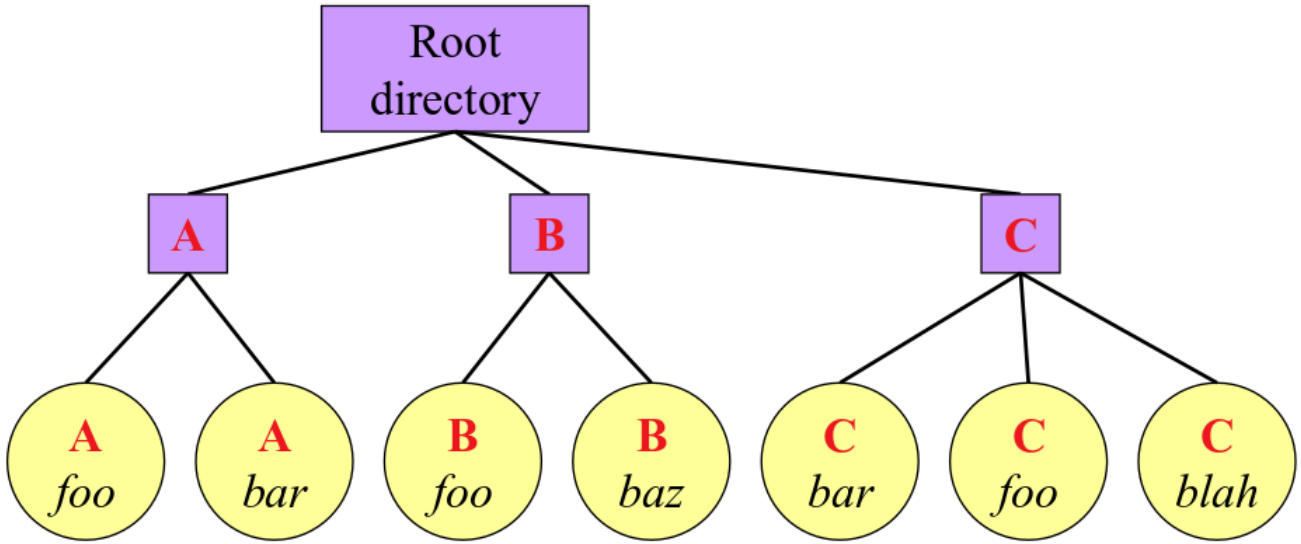
Una sola directory nel file system

In questo esempio la directory:

- Contiene 4 file (*foo*, *bar*, *baz*, *blah*)
- I file appartengono a 3 persone diverse: A, B, e C (mostrate in rosso)

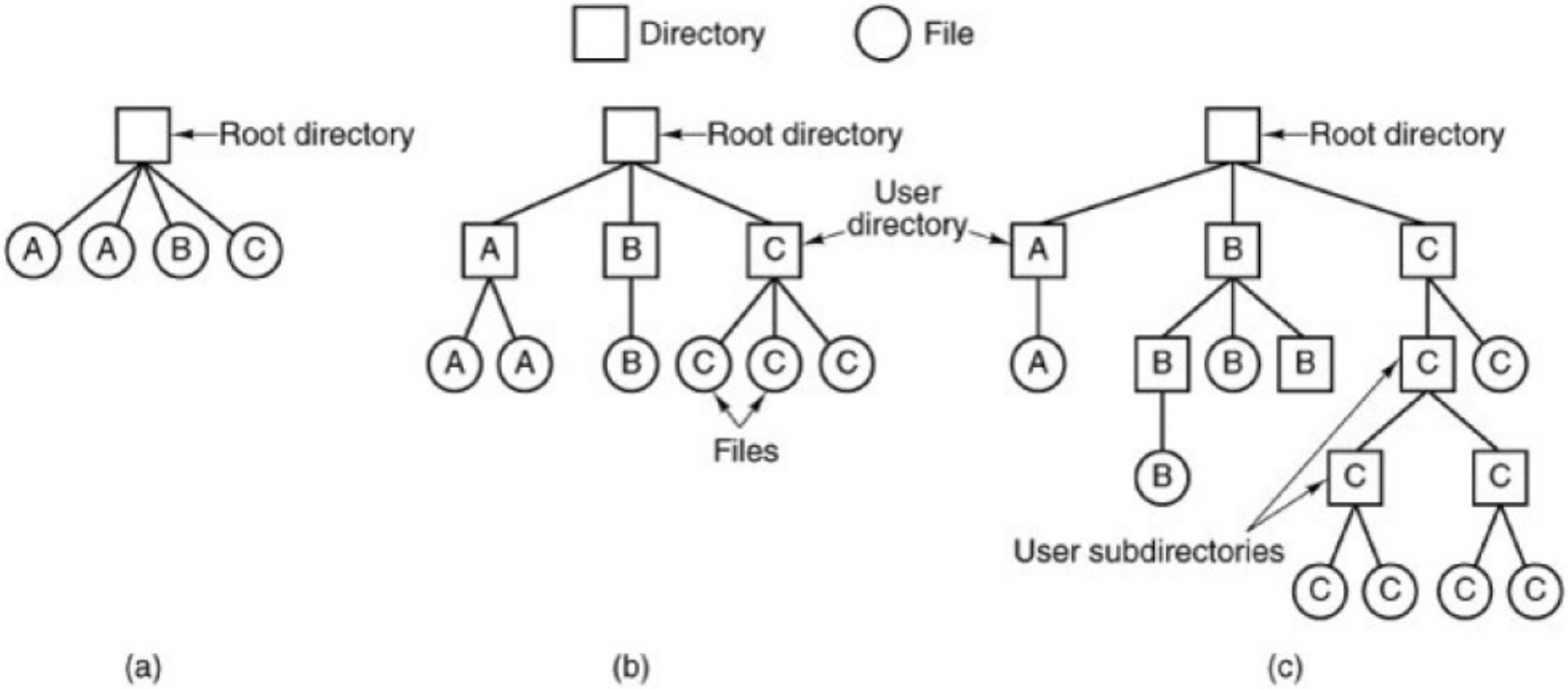
Problema: Cosa accade se utente B vuole creare un file di nome *foo*?

Sistema con due livelli di directory:

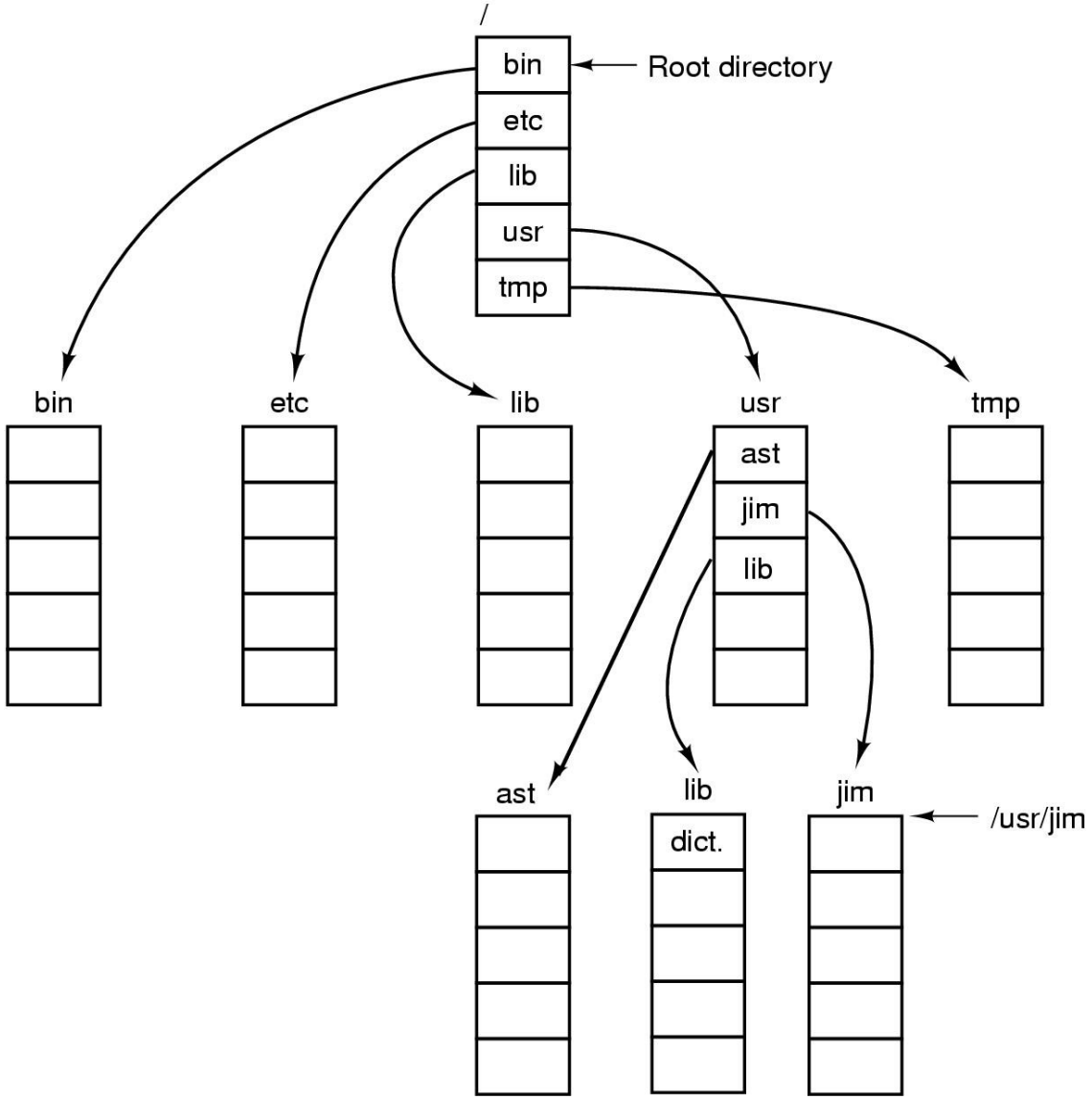


Risolve il problema dei nomi dei file: ogni utente ha una sua directory
Più utenti possono creare file con lo stesso nome
Di default, gli utenti possono accedere solo ai file nella loro directory
Estensione: permettere agli utenti di accedere anche ai file nelle directory degli altri utenti

Sistema con directory organizzate secondo una gerarchia:



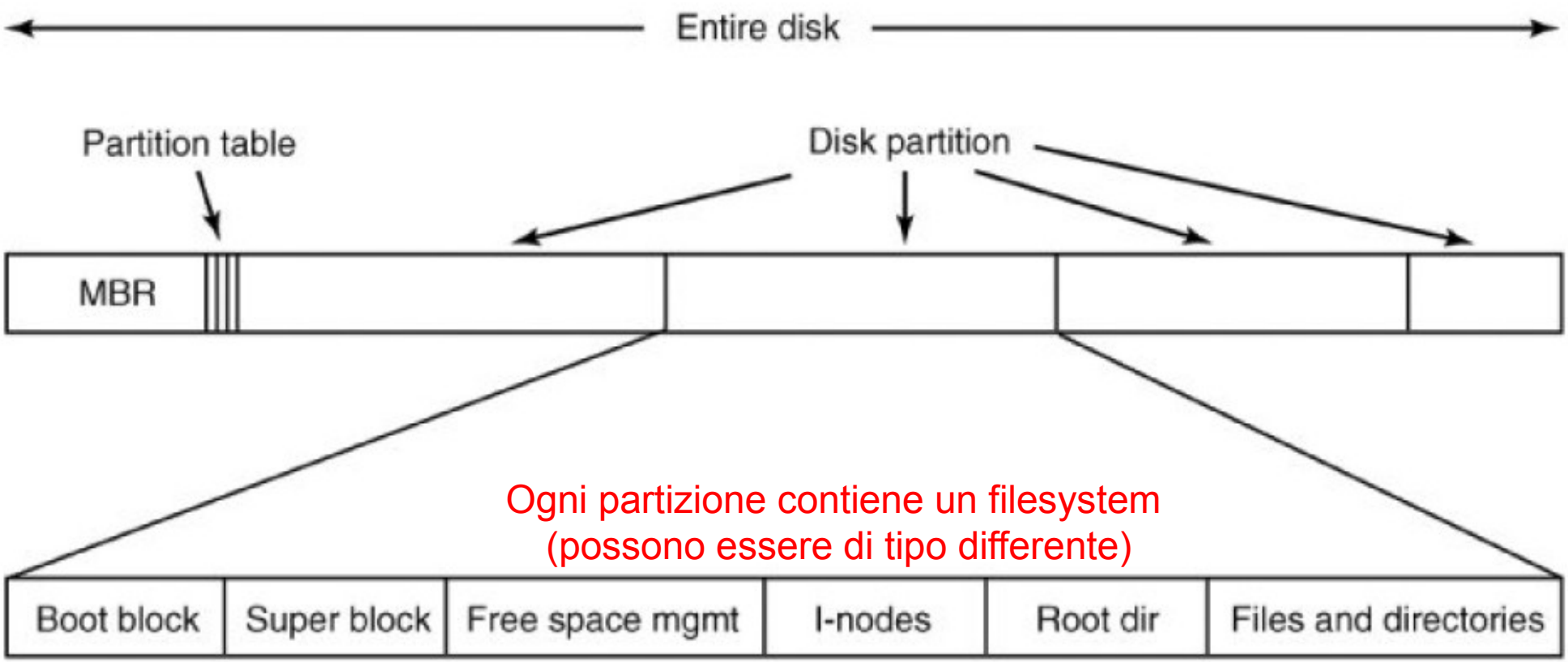
UNIX directory tree:



Problemi inerenti alla implementazione di un file system:

- In che modo è possibile organizzare un disco utilizzando uno o più file system?
- Come fa il file system ad associare I blocchi fisici del disco con I vari file?
- Come fa il file system a gestire lo spazio libero?
- Come vengono gestite le directory?
- Come è possibile migliorare un file system?
  - Performance?
  - Affidabilità?

Layout dischi:



Unix filesystem (case study: ext2)

Il filesystem unix ext2 è organizzato in una struttura comprendente:

super_block

inodes

directory

files

Vedremo come sono strutturati i singoli componenti del filesystem ext2 e quali sono i vantaggi di tale organizzazione.

```
/*
 * Structure of the super block
 */
struct ext2_super_block {
        __u32   s_inodes_count;         /* Inodes count */
        __u32   s_blocks_count;         /* Blocks count */
        __u32   s_r_blocks_count;       /* Reserved blocks count */
        __u32   s_free_blocks_count;    /* Free blocks count */
        __u32   s_free_inodes_count;    /* Free inodes count */
        __u32   s_first_data_block;     /* First Data Block */
        __u32   s_log_block_size;       /* Block size */
        __s32   s_log_frag_size;        /* Fragment size */
        __u32   s_blocks_per_group;     /* # Blocks per group */
        __u32   s_frags_per_group;      /* # Fragments per group */
        __u32   s_inodes_per_group;     /* # Inodes per group */
        __u32   s_mtime;                /* Mount time */
        __u32   s_wtime;                /* Write time */
        __u16   s_mnt_count;            /* Mount count */
        __s16   s_max_mnt_count;        /* Maximal mount count */
        __u16   s_magic;                /* Magic signature */
        __u16   s_state;                /* File system state */
        __u16   s_errors;               /* Behaviour when detecting errors */
        __u16   s_pad;
        __u32   s_lastcheck;            /* time of last check */
        __u32   s_checkinterval;        /* max. time between checks */
        __u32   s_creator_os;           /* OS */
        __u32   s_rev_level;            /* Revision level */
        __u16   s_def_resuid;           /* Default uid for reserved blocks */
        __u16   s_def_resgid;           /* Default gid for reserved blocks */
        __u32   s_reserved[235];        /* Padding to the end of the block */
};
```

ext2 super block

Il filesystem ext2 è descritto da un blocco detto super block che è memorizzato in posizione fissa all'inizio del fs stesso e ne descrive le caratteristiche, dimensioni, struttura e stato.

3 tipi fondamentali di informazioni:

Caratteristiche del filesystem
(es. dimensioni, struttura, ...)

Parametri modificabili
(es. maximal mount count)

Variabili di stato
(es. file system state, free blocks, mount count, ...)

```
/*
 * Structure of the super block
 */
struct ext2_super_block {
        __u32   s_inodes_count;         /* Inodes count */
        __u32   s_blocks_count;         /* Blocks count */
        __u32   s_r_blocks_count;       /* Reserved blocks count */
        __u32   s_free_blocks_count;    /* Free blocks count */
        __u32   s_free_inodes_count;    /* Free inodes count */
        __u32   s_first_data_block;     /* First Data Block */
        __u32   s_log_block_size;       /* Block size */
        __s32   s_log_frag_size;        /* Fragment size */
        __u32   s_blocks_per_group;     /* # Blocks per group */
        __u32   s_frags_per_group;      /* # Fragments per group */
        __u32   s_inodes_per_group;     /* # Inodes per group */
        __u32   s_mtime;                /* Mount time */
        __u32   s_wtime;                /* Write time */
        __u16   s_mnt_count;            /* Mount count */
        __s16   s_max_mnt_count;        /* Maximal mount count */
        __u16   s_magic;                /* Magic signature */
        __u16   s_state;                /* File system state */
        __u16   s_errors;               /* Behaviour when detecting errors */
        __u16   s_pad;
        __u32   s_lastcheck;            /* time of last check */
        __u32   s_checkinterval;        /* max. time between checks */
        __u32   s_creator_os;           /* OS */
        __u32   s_rev_level;            /* Revision level */
        __u16   s_def_resuid;           /* Default uid for reserved blocks */
        __u16   s_def_resgid;           /* Default gid for reserved blocks */
        __u32   s_reserved[235];        /* Padding to the end of the block */
};
```

ext2 super block

Alcune caratteristiche vengono definite al momento della creazione e poi non Possono più essere modificate:

- numero blocchi
- dimensione blocchi

…

Altre caratteristiche vengono agigornate automaticamente (stato fs):

- montato/smontato
- conteggio eventi mount
- numero blocchi liberi
- numero i-nodes liberi

…

Unix filesystem (case study: ext2)

Il filesystem ext2 è suddiviso logicamente in più parti, dette **cylinder groups**, che vengono gestite come entità separate pur facendo parte dello stesso fs.

Motivazioni:

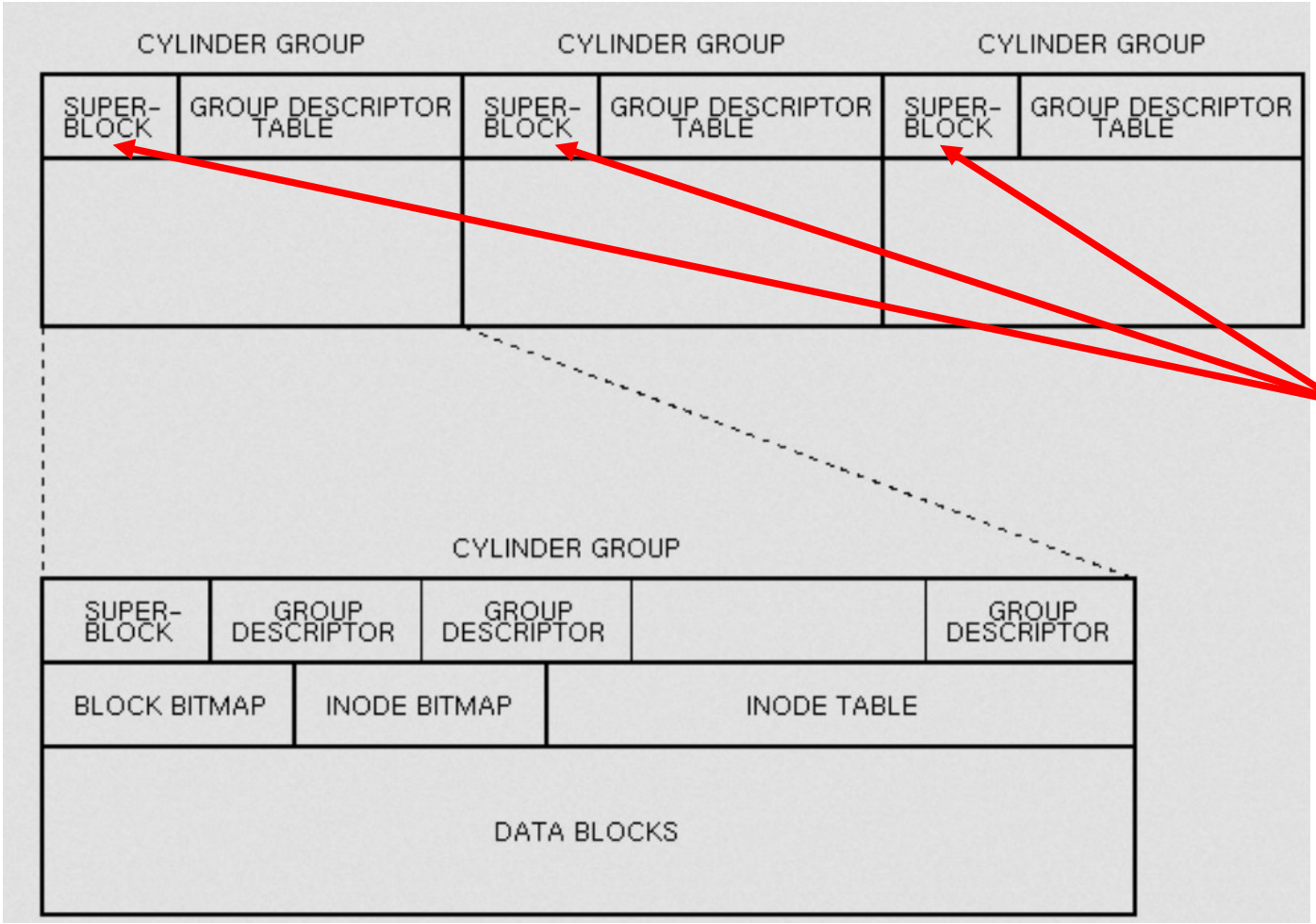
Minimizzare le conseguenze di errori. Errori in un cylinder group non si propagano a tutto il fs.

Localizzare i files nell'intorno delle loro directory per ridurre i tempi di accesso (allocazione inodes e blocchi nello stesso cylinder group delle directory.

Ulteriore protezione (dell'intero filesystem):

Sia il super block che le group descriptor tables vengono duplicati in ogni cylinder group.

Cylinder groups



ridondanza in cambio di sicurezza

Se uno dei super_block o group descriptor **viene corrotto** a causa di qualche errore esso può essere facilmente ripristinato a partire da una delle sue copie.

I-node:

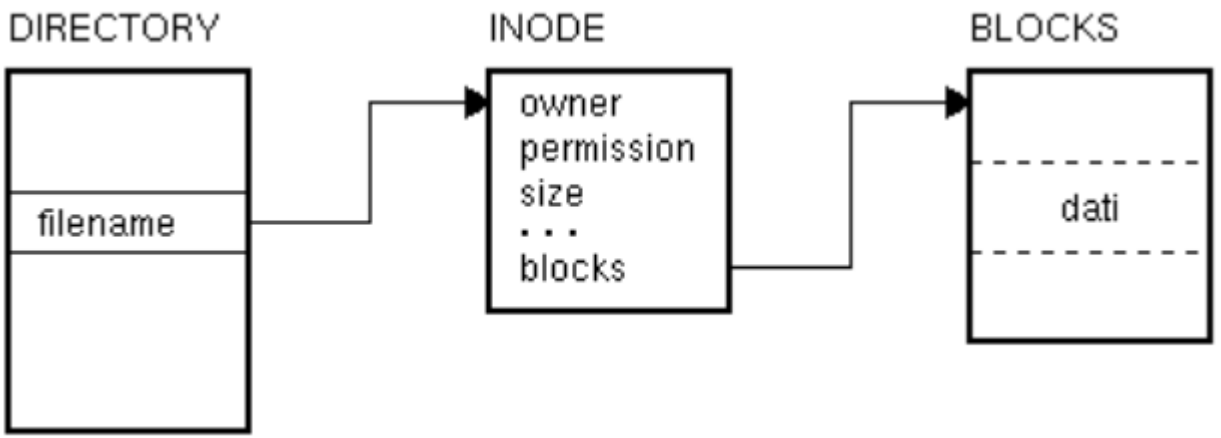
È il **descrittore** del file.

Tra gli attributi contenuti nell'*i-node*:

- tipo di file:
- ordinario
- directory
- file speciale
- proprietario, gruppo (*user-id, group-id*)
- Lunghezza del file
- Data e tempo di creazione e modifica
- diversi bit di protezione
- indirizzi di blocchi fisici (in numero che varia a seconda della realizzazione)

I-node:

È la “risorsa” principale del filesystem di tipo unix. Qui di seguito vediamo le relazioni tra la directory ed i file. L'elemento che mette in relazione directory, file e blocchi di dati su disco è l'i-dnode.



Esempio di struttura di un I-node:

(solo alcuni campi...)

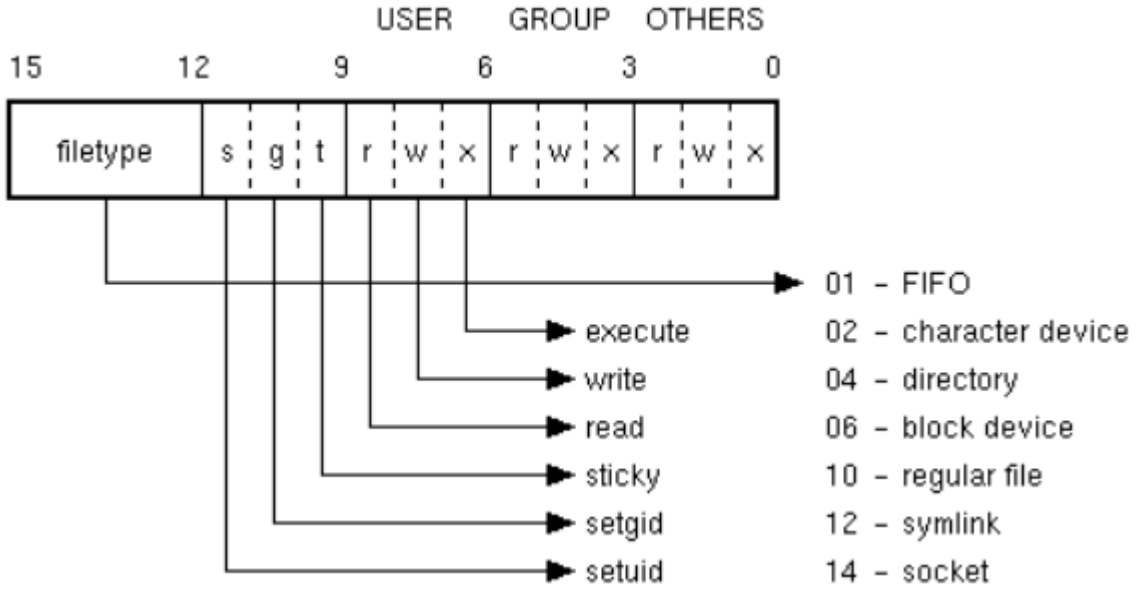
Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	39	Address of first 10 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)

```
struct ext2_inode {
        __u16   i_mode;         /* File mode */
        __u16   i_uid;          /* Owner Uid */
        __u32   i_size;         /* Size in bytes */
        __u32   i_atime;        /* Access time */
        __u32   i_ctime;        /* Creation time */
        __u32   i_mtime;        /* Modification time */
        __u32   i_dtime;        /* Deletion Time */
        __u16   i_gid;          /* Group Id */
        __u16   i_links_count;  /* Links count */
        __u32   i_blocks;       /* Blocks count */
        __u32   i_flags;        /* File flags */
        union {
                struct {
                        __u32  l_i_reserved1;
                } linux1;
                struct {
                        __u32  h_i_translator;
                } hurd1;
                struct {
                        __u32  m_i_reserved1;
                } masix1;
        } osd1;                         /* OS dependent 1 */
        __u32   i_block[EXT2_N_BLOCKS];/* Pointers to blocks */
        __u32   i_version;      /* File version (for NFS) */
        __u32   i_file_acl;     /* File ACL */
        __u32   i_dir_acl;      /* Directory ACL */
        __u32   i_faddr;        /* Fragment address */
        union {
                struct {
                        __u8    l_i_frag;       /* Fragment number */
                        __u8    l_i_fsize;      /* Fragment size */
                        __u16   i_pad1;
                        __u32   l_i_reserved2[2];
                } linux2;
                struct {
                        __u8    h_i_frag;       /* Fragment number */
                        __u8    h_i_fsize;      /* Fragment size */
                        __u16   h_i_mode_high;
                        __u16   h_i_uid_high;
                        __u16   h_i_gid_high;
                        __u32   h_i_author;
                } hurd2;
                struct {
                        __u8    m_i_frag;       /* Fragment number */
                        __u8    m_i_fsize;      /* Fragment size */
                        __u16   m_pad1;
                        __u32   m_i_reserved2[2];
                } masix2;
        } osd2;                         /* OS dependent 2 */
};
```

I-node:

Struttura i-node ext2

Contiene diversi elementi tra cui **i_mode** che contiene informazioni su tipo di file e permessi

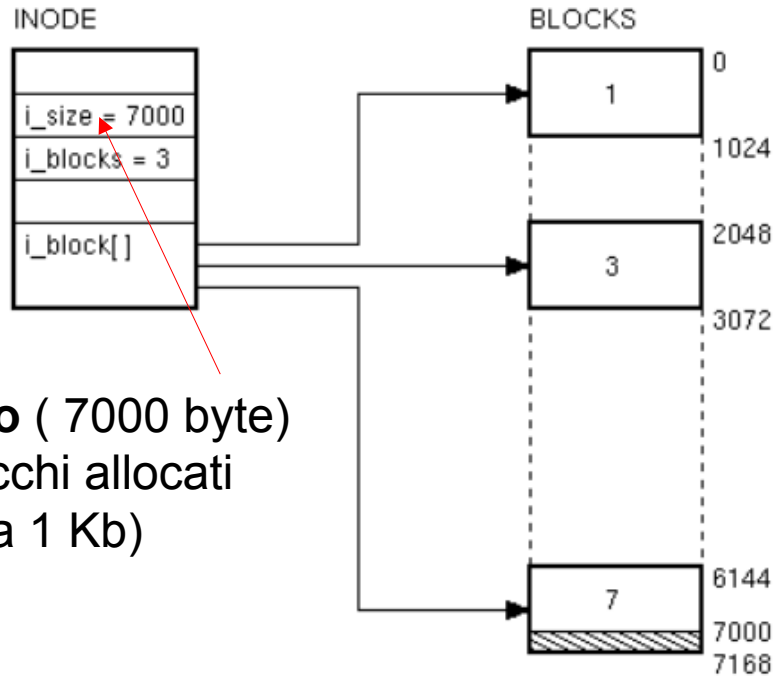


Struttura i-node ext2

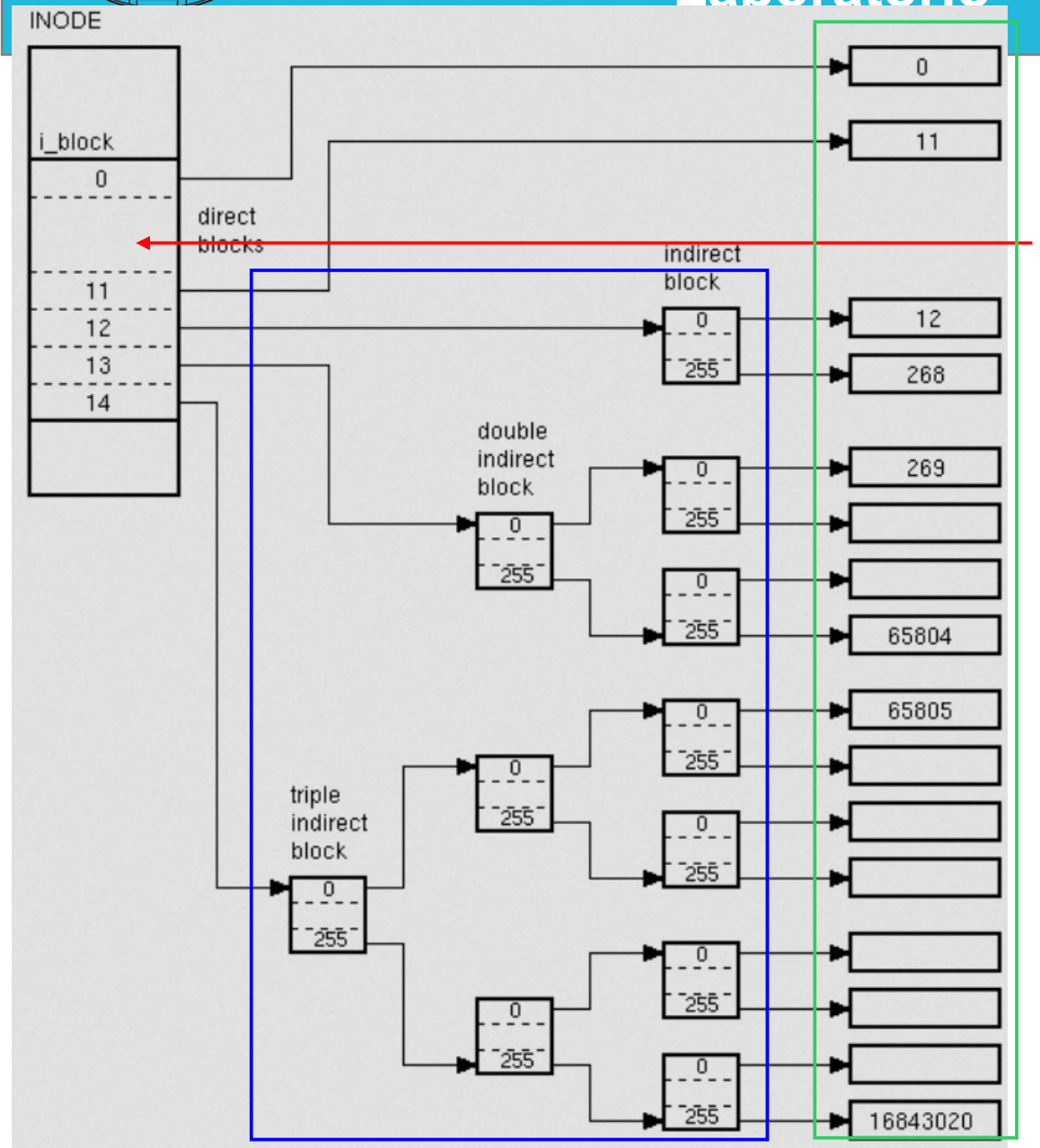
```
struct ext2_inode {
        __u16   i_mode;         /* File mode */
        __u16   i_uid;          /* Owner Uid */
        __u32   i_size;         /* Size in bytes */
        __u32   i_atime;        /* Access time */
        __u32   i_ctime;        /* Creation time */
        __u32   i_mtime;        /* Modification time */
        __u32   i_dtime;        /* Deletion Time */
        __u16   i_gid;          /* Group Id */
        __u16   i_links_count;  /* Links count */
        __u32   i_blocks;       /* Blocks count */
        __u32   i_flags;        /* File flags */
        union {
                struct {
                        __u32  l_i_reserved1;
                } linux1;
                struct {
                        __u32  h_i_translator;
                } hurd1;
                struct {
                        __u32  m_i_reserved1;
                } masix1;
        } osd1;                         /* OS dependent 1 */
        __u32   i_block[EXT2_N_BLOCKS];/* Pointers to blocks */
        __u32   i_version;      /* File version (for NFS) */
        __u32   i_file_acl;     /* File ACL */
        __u32   i_dir_acl;      /* Directory ACL */
        __u32   i_faddr;        /* Fragment address */
        union {
                struct {
                        __u8    l_i_frag;       /* Fragment number */
                        __u8    l_i_fsize;      /* Fragment size */
                        __u16   i_pad1;
                        __u32   l_i_reserved2[2];
                } linux2;
                struct {
                        __u8    h_i_frag;       /* Fragment number */
                        __u8    h_i_fsize;      /* Fragment size */
                        __u16   h_i_mode_high;
                        __u16   h_i_uid_high;
                        __u16   h_i_gid_high;
                        __u32   h_i_author;
                } hurd2;
                struct {
                        __u8    m_i_frag;       /* Fragment number */
                        __u8    m_i_fsize;      /* Fragment size */
                        __u16   m_pad1;
                        __u32   m_i_reserved2[2];
                } masix2;
        } osd2;                         /* OS dependent 2 */
};
```

i_blocks contiene il numero di blocchi occupati dal file (compresi i blocchi di indirizzamento indiretto che vedremo tra poco).

i_size e i_blocks possono contenere dimensioni differenti. Motivo: in unix è possibile creare file sparsi (nel senso che non tutti i blocchi necessari a raggiungere la dimensione specificata in i_size devono essere necessariamente allocati in un dato momento).



File **sparso** (7000 byte)
Solo **3** blocchi allocati
(ognuno da 1 Kb)



ext2 i-node

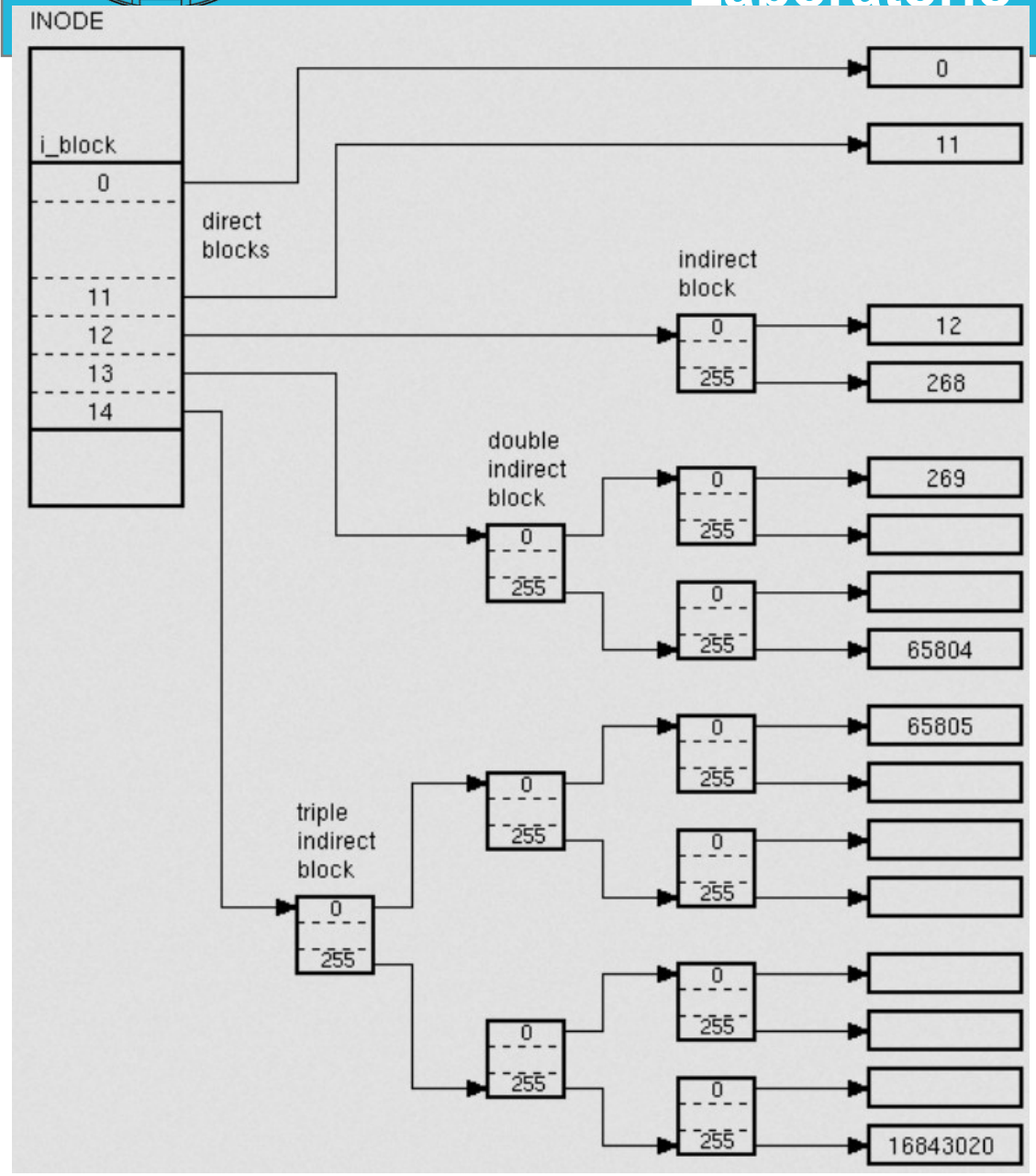
Indirizzi (posizioni sul disco)

2 tipi di blocchi: logici (contengono indirizzi) blocchi fisici (contengono i dati dei file)

Blocco indiretto: non punta ad un blocco contenente i dati del file ma ad un blocco che contiene puntatori ad altri blocchi. Assumendo una dimensione del blocco logico di 1024 byte e puntatori di 4 byte avremo 1024/4 = 256 blocchi indirizzabili. Tali blocchi sono i blocchi dal 12 al 268.

Blocco a doppia indirezione: blocchi indirizzabili 256 * 256 = 65536. Essi corrispondono ai blocchi dati dal 269 al 65804.

Grazie al blocco ad indirezione tripla sarà possibile indirizzare blocchi dati dal Blocco 65805 al blocco 16843020.



ext2 i-node

Che utilità può avere questo tipo di struttura?

Per file piccoli sono sufficienti i blocchi diretti ma, in caso di necessità, è possibile allocare più risorse (più blocchi fisici su disco).

Qui emerge molto chiaramente la natura del sistema operativo: esso tende a massimizzare l'efficienza con cui vengono gestite le risorse (in questo caso spazio disco).

Questo richiede strutture particolari, come nel caso di i-node.

DEFINIZIONI:
BloccoLogico: **indice** di un blocco logico nel file
BloccoFisico: **indice** di un blocco nel disco

IPOTESI
Nello i-node: **10 indirizzi diretti, 3 indirizzi indiretti**
Lunghezza di ogni blocco (logico e fisico):1 Kbyte
Lunghezza di ogni puntatore:=4 byte ==> numero di puntatori in ogni blocco indiretto: 210/4= 28

10 blocchi fisici indirizzabili con puntatori diretti; accesso immediato dallo i-node

28 blocchi fisici indirizzabili con puntatore indiretto semplice; 1 accesso al disco per leggere un blocco indiretto di primo livello

216 blocchi fisici indirizzabili con puntatore indiretto doppio; 2 accessi al disco, per leggere due blocchi indiretti, di secondo e di primo livello

224 blocchi fisici indirizzabili con puntatore indiretto triplo; 3 accessi al disco, per leggere 3 blocchi indiretti di terzo, di secondo e di primo livello

```
if BloccoLogico<10  BloccoFisico=&puntatore[BloccoLogico]
else  {
        ind1=(BloccoLogico-10) div 28;offset1= (BloccoLogico-10) mod 28 ;
        /* ind1 è il puntatore a un blocco indiretto di primo livello*/
        if ind1=0  BloccoFisico=&((&puntatore[10])[offset1])
        else   {
         ind2=(ind1-1) div 28; offset2= (ind1-1)mod 28 ;
                /* ind2 è il puntatore a un blocco indiretto di secondo livello*/
                if ind2=0  BloccoFisico=&(&((&puntatore[11])[offset2]))[offset1]
                else   {
                        ind3=(0; offset3= (ind2-1)mod 28;
                           /* ind3 è il puntatore all'unico blocco indiretto di terzo livello*/
                       BloccoFisico=&(&(&(&(puntatore[12])[offset3])[offset2]))[offset1]
                        }
                }
        }
```

Indirizzamento dei blocchi tramite i-node e blocchi indiretti : esempio

ESERCIZIO 1:

Ipotizziamo che in un file system UNIX i blocchi del disco abbiano ampiezza di **1Kb** e che i puntatori ai blocchi siano a **32 bit**.
Gli i-node contengono, oltre agli altri attributi, 10 puntatori diretti e 3 puntatori indiretti, I puntatori sono indici di blocco fisico.
Il primo blocco logico del file e il primo blocco del disco hanno indice 0.

Massima estensione del File System, in base al numero di blocchi indirizzabili nel disco: *232 blocchi → 232*210=* ***242*** *byte (4 Tbyte)*

Indirizzamento dei blocchi tramite i-node e blocchi indiretti : esempio

Massima lunghezza di ogni singolo file, in base al numero di blocchi indirizzabili per ogni file:

Considerato che:

- lo i-node indirizza direttamente **10 blocchi**
- ogni blocco indiretto contiene 210 div 4= 28 indirizzi
- il blocco indiretto di **primo livello** puntato dall'indirizzo indiretto semplice indirizza **28 blocchi** dati
- il blocco indiretto di **secondo livello** puntato dall'indirizzo indiretto doppio indirizza 28 blocchi indiretti di primo livello, cascuno dei quali indirizza 28 blocchi dati
  ==> possono essere indirizzati **216 blocchi** dati
- il blocco indiretto di **terzo livello** puntato dall'indirizzo indiretto triplo indirizza 28 blocchi indiretti di secondo livello, ciascuno dei quali indirizza 28 blocchi indiretti di primo livello, ciascuno dei quali indirizza 28 blocchi dati
  ==> possono essere indirizzati **224 blocchi** dati

la massima lunghezza di ogni singolo file è :
10 + 28 + 216 + 224 = 10+ 256+ 65.536+ 16.777.216 =16.843.018 blocchi

approssimativamente: 16 Gbyte ≤ massima lunghezza <17 Gbyte

Indirizzamento dei blocchi tramite i-node e blocchi indiretti : impatto dimensione blocchi (ext2)

Upper Limits	1KiB	2KiB
file system blocks	2,147,483,647	2,147,483,647
blocks per block group	8,192	16,384
inodes per block group	8,192	16,384
bytes per block group	8,388,608 (8MiB)	33,554,432 (32MiB)
file system size (real)	4,398,046,509,056 (4TiB)	8,796,093,018,112 (8TiB)
file system size (Linux)	2,199,023,254,528 (2TiB) [a]	8,796,093,018,112 (8TiB)
blocks per file	16,843,020	134,217,728
file size (real)	17,247,252,480 (16GiB)	274,877,906,944 (256GiB)
file size (Linux 2.6.28)	17,247,252,480 (16GiB)	274,877,906,944 (256GiB)

Dimensioni blocchi comunemente implementate: 1 Kb, 2 Kb, 4 Kb, 8 Kb.
http://www.nongnu.org/ext2-doc/ext2.html#BLOCK-SIZE-IMPACT

NB: dimensioni blocchi logici posono essere impostate al momento della creazione del file system. Dimensione blocchi fisici solitamente 512 bytes.

I-node speciali :

- Directory (`mkdir`)
- Link simbolici (`ln -s`)

Programmi utili per lavorare sui nomi o percorsi

- `dirname`
- `basename`

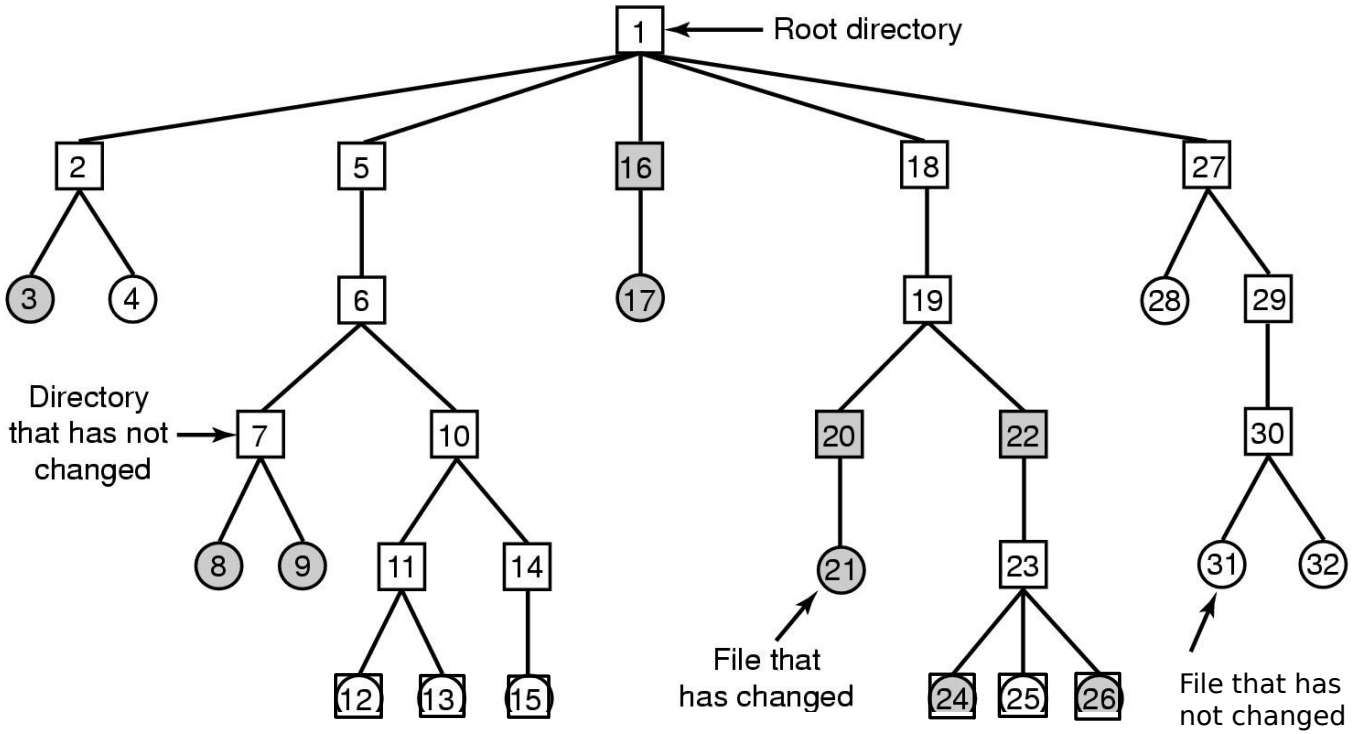
Programmi utili per lavorare sugli i-node

- `stat`
- `readlink`

Backup di un file system :

Un file system di cui si vuole effettuare un backup:

- Quadrati: directory, Cerchi: file
- Grigio: elementi modificati dopo l'ultimo backup effettuato
- Directory e file etichettati mediante numero di I-node



Mappe di bit utilizzate durante il backup di un file system :

(a)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

(b)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

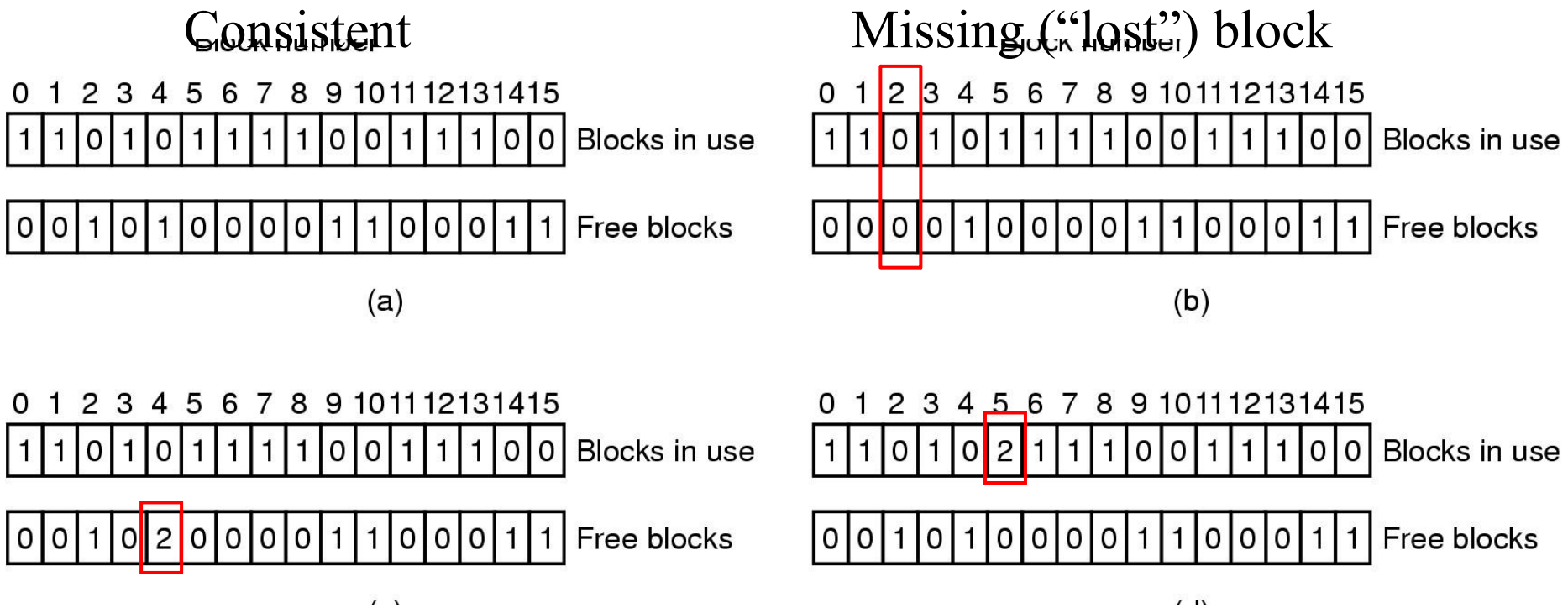
(c)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

(d)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

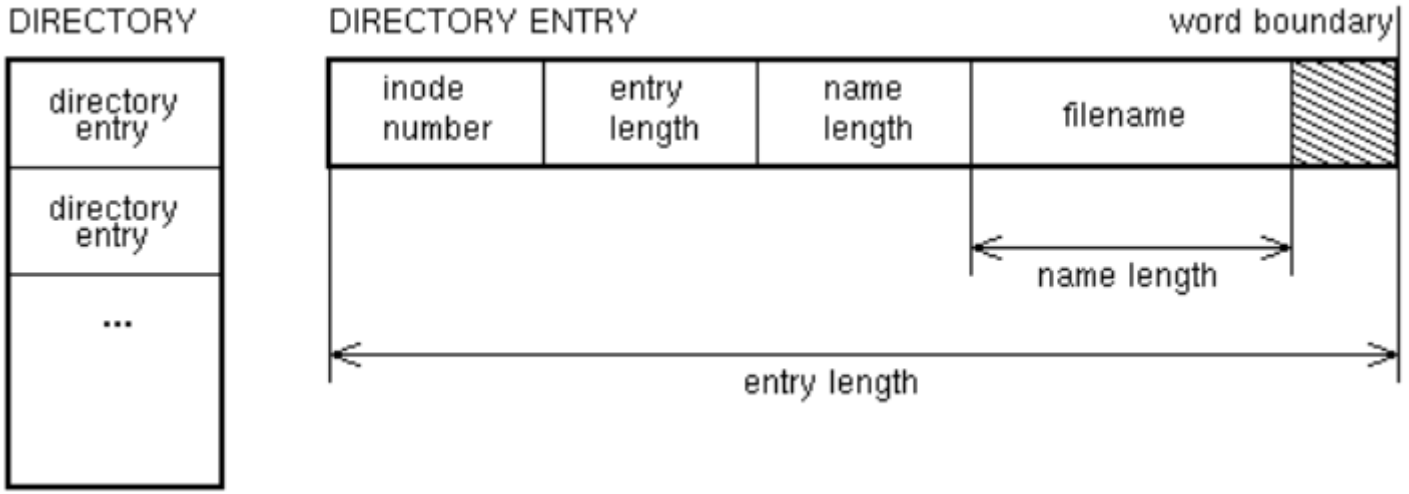
Check di consistenza di un file system :



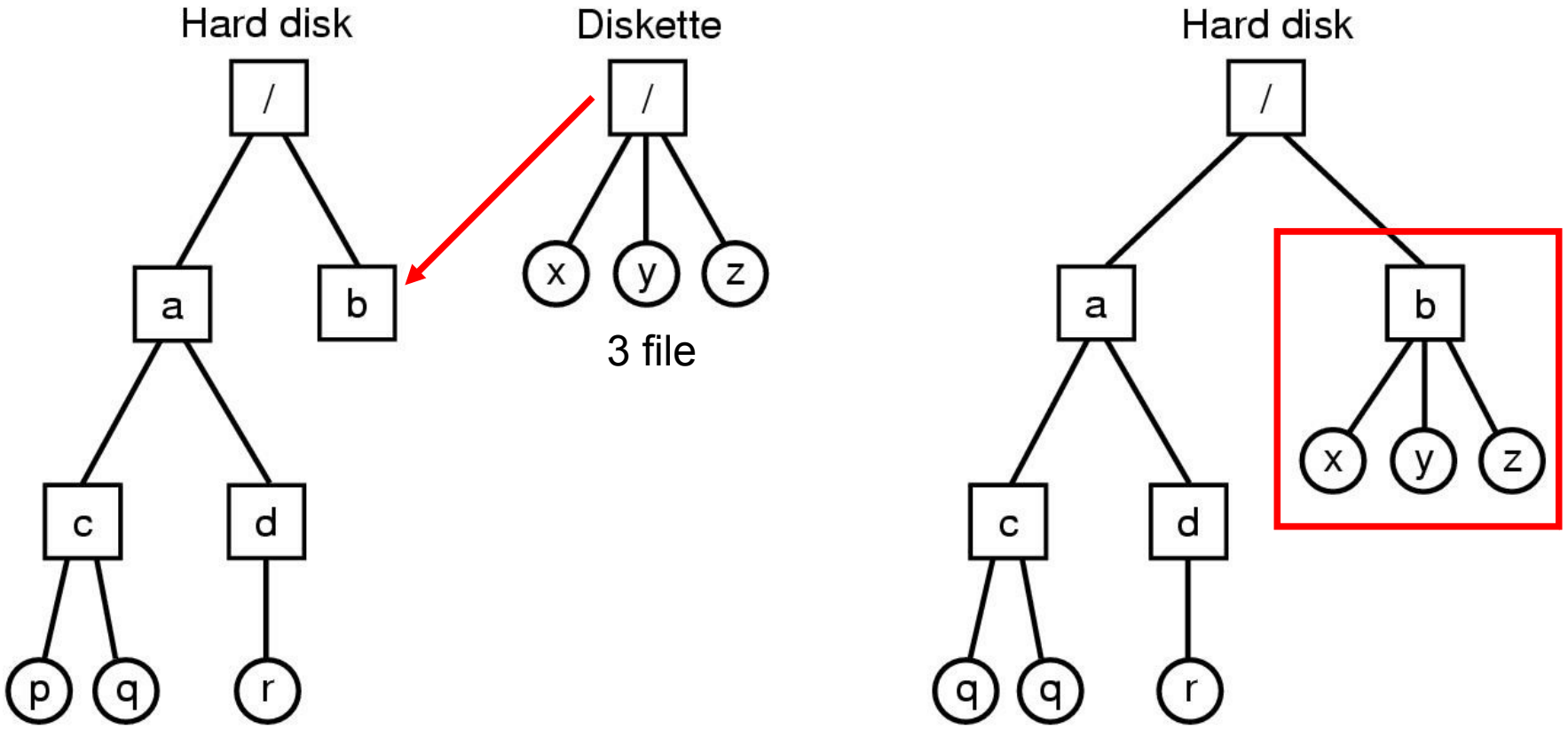
La directory (ext2) :

```
#define EXT2_NAME_LEN 255
struct ext2_dir_entry {
        __u32   inode;                  /* Inode number */
        __u16   rec_len;                /* Directory entry length */
        __u16   name_len;               /* Name length */
        char    name[EXT2_NAME_LEN];    /* File name */
};
```

La directory è un tipo particolare di file che ha, come funzione, quello di collegare l'identificativo dell' i-node di un file al nome del file. Tutte le altre informazioni sul file non hanno a che fare con la directory e sono salvate nell'i-node.



Montaggio di un file system :



(a) prima del montaggio (b) dopo il montaggio

Creare e usare un file system :

Comandi linux

- Un file system va *creato* (`mkfs`)
- Un file system va *montato* (`mount`)
- Corrispondentemente va smontato (`umount`)
- Ogni file è caratterizzato da un i-node e conosciuto tramite uno o piú link o nomi (`ln`)

ESERCIZIO (L4 E1) :

ESERCIZIO QEMU

1. Creare un disco virtuale
2. Partizionare il disco
3. Creare il file system
4. Montare il file system

Step I : creazione file disco virtuale per Qemu

(in Win)

1)Aprire prompt dei comandi

2)Portarsi nel folder contenente l'eseguibile Qemu (quello con I file bat)

3)qemu-img.exe create –f qcow2 miodiscoA.img 100M

4)Spostare il file appena prodotto nello stesso folder che contiene solab.iso

5)COPIATE il file bat di avvio e modificate la copia come segue:

```
REM ######################################################
REM # Boot disk image
REM # Adapt "-k fr" to you keyboard
REM # X11, nic and soundhw were tested and are working!
REM # try madplay 20thfull.mp2 or xinit
REM # Added '-usb -usbdevice tablet' for Seamless mouse
REM ######################################################
START qemu-system-i386w.exe -L Bios -vga std ^
-rtc base=localtime,clock=host ^
-cdrom ..\solab.iso -hda ..\miodiscoA.img -boot d ^
-net nic,model=e1000 -net user ^
-m 512 -no-acpi -no-hpet -no-reboot
```

Step II : partizionamento

(in **Qemu**)
1)Avviate Qemu usando il file bat prodotto alla slide precedente
2)Ottenete una shell di root : **sudo -s**
3)Ottenete la lista dei device disponibili (e le loro partizioni) : **fdisk -l**
4)A noi interessa /dev/sda . Usiamo fdisk per partizionarlo: **fdisk /dev/sda**
5)Per aiuto su comandi fdisk scrivete m e premete invio

6)n * p * 1 * (default) * +20M *
7)n p 2 (default) +20M
8)n p 3 (default) +20M
9)n e (default) (default)
10)n (default) +20M
11)n (default) (default)
12)P
13)w

Comandi fdisk
(e argomenti)

* = INVIO
Non scrivete **nulla** e premete invio

Step II (b) : partizionamento, tipo partizioni

(in **fdisk**)

DOPO aver creto una partizione potete cambiarne il **TIPO**

1) In fdisk utilizzare il commando **t**
2) Fdisk chiederà il **numero** della partizione su cui volete operare

1)Dopo chiederà l'identificativo numerico (esadecimale) del tipo di partizione da utilizzare. Per avere una lista complete usare il commando **L**

2)Dopo aver impostato il tipo della partizione verificate utilizzando il commando **p**

3)Se tutto è come vi aspettate (le partizioni hanno dimensione e tipo corretti) confermate come nella slide precedente utilizzando il commando per scrivere la tabella delel partizioni : **w**

Step III : creazione file system(s)

(in Qemu, dopo aver usato fdisk -l)
1)mkfs /dev/sda1
2)mkfs /dev/sda2
3)mkfs /dev/sda3
4)Non provate a formattare /dev/sda4
5)mkfs /dev/sda5
6)mkfs /dev/sda6

Step IV : montare una partizione

7)cd /mnt
8)mkdir disksda1
9)mount /dev/sda1 /mnt/disksda1

Dopo aver montato la partizione:
. Entrate in /mnt/disksda1
. Create un file con nome a vostra scelta
. Scriveteci dentro qualcosa
. Salvate il file
. Uscite dal folder disksda1

StepV : smontate la partizione

1)umount /dev/sda1

A questo punto:

-)Chiudete qemu
-)Riavviate qemu
-)Ottenete una shell di root
-)Andate in /mnt
-)Create il folder disksda1
-)montate /dev/sda1 in /mnt/disksda1
-)Verificate la presenza del file che avevate creato prima

NB: Aggiungete alla riga di comando che lancia qemu (nel file .bat) **-boot d** , avvio da cdrom.

ESERCIZIO (L4 E2) :

ESERCIZIO QEMU

dd, dumpe2fs , less, expr
Indagine dimensione (in blocchi) di un file

Step I : creazione di un filepieno di zeri (posizionatevi in /mnt/disksda1 dopo avervi montato /dev/sda1):
1)dd if=/dev/zero of=/mnt/disksda1/**miofile** **bs=10 count=1264**

Step II : forzare flush del file system buffer (assicuriamoci che tutte le modifiche siano scritte su disco) e verifichiamo il numero dei blocchi liberi:
2)sync
3)dumpe2fs /dev/sda1 | less

Annotate valore da parte a prima occorrenza 'Free blocks'

Step III : rimuovere file , forzare flush verifica numero blocchi liberi:
4)rm /mnt/disksda1/miofile
5)sync
3) dumpe2fs /dev/sda1 | less
Step IV : blocchi occupati dal file
1) expr freeblocks_senzafile – freeblocks_confile