File System

Motivazioni
File
Directory
Implementazione

Motivazioni

I dati dal punto di vista dell'utente

- Necessità di memorizzare enormi quantità di informazioni
- Necessità di memorizzare in modo permanente informazioni
- Necessità di accedere contemporaneamente agli stessi dati da parte di più processi
- Necessità di accedere ai dati in maniera ottimizzata

I dati dal punto di vista del Sistema Operativo:

Il file system

Scopi del file system

- garantire un accesso permanente, conveniente e consistente
- garantire un uso efficiente delle risorse di memorizzazione

Il File System

Un file system è l'insieme di algoritmi e strutture dati che realizzano la traduzione tra operazioni logiche sui file e le informazioni memorizzate sui dispositivi fisici (dischi, nastri)

Un file system rappresenta una astrazione unificata dei dispositivi fisici effettivi

Elementi di un file system:

- logici:
  - file
  - struttura di directory
- software:
  - chiamate di sistema
  - routine di gestione
  - algoritmi di scheduling
  - device driver

Il File

- Dal punto di vista dell'utente un file è un insieme di dati correlati e associato ad un nome
- Dal punto di vista del sistema operativo, un file è un insieme di byte (eventualmente strutturato)
- Il nome:
  - è una sequenza (limitata) di caratteri
  - l'insieme di caratteri leciti dipende dal sistema operativo
  - la maggior parte dei sistemi operativi moderni distingue fra lettere maiuscole e minuscole (a volte si usano caratteri UNICODE)
  - alcuni sistemi operativi dividono il nome in due parti separate da un punto ".", l'estensione permette di classificare il file

Nomi: MS-DOS

- Nomi di 8+3 caratteri ('case' non significativo, molti caratteri non utilizzabili), l'estensione è utilizzata dal sistema operativo per trattare correttamente il file:
  - .com, .exe, .bat: file eseguibili
  - .c: sorgenti C
  - .doc: file World
- problemi:
  - più programmi possono usare la stessa estensione
  - lo stesso file può essere elaborato da più programmi
  - l'estensione può essere gestita in modo non corretto dagli utenti

Nomi

- UNIX: nomi lunghi (solo "/" non è utilizzabile)
  le estensioni non sono gestite (ma spesso utilizzate dagli utenti e da alcuni applicativi)
- Windows NT ed evoluzioni successive: nomi lunghi codificati in UNICODE (ma non tutti i caratteri sono utilizzabili), parziale distinzione fra lettere maiuscole e minuscole, possibilità di associare più flussi di dati

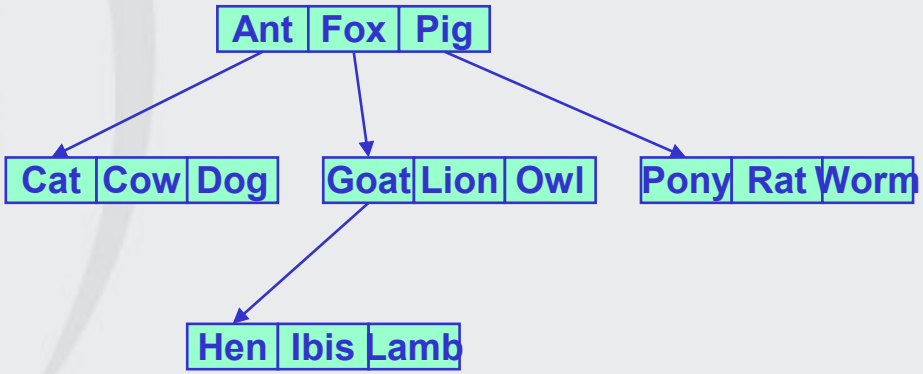
Struttura di file

- Si distinguono tre tipi diversi di strutture:
  - Sequenza di byte (ovvero nessuna struttura)
    la struttura interna del file è gestita dai programmi applicativi

  - Sequenza di record di dimensione fissa
    le operazioni di lettura restituiscono un record, le operazioni di scrittura sovrascrivono o appendono un record

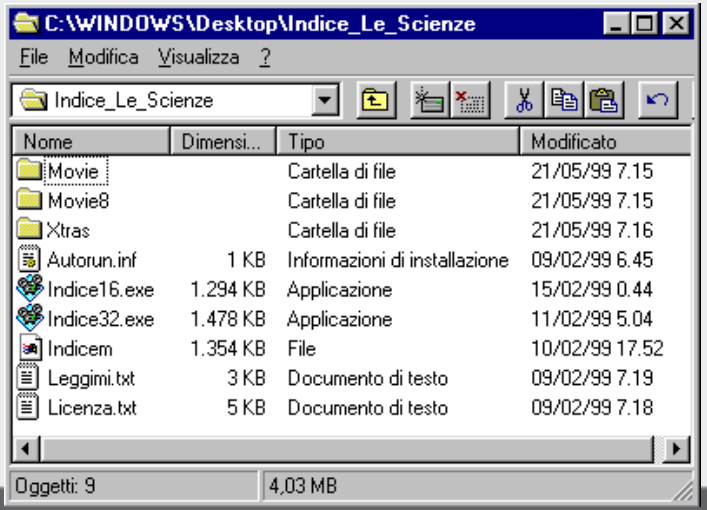
Struttura di file

- Albero di record di lunghezza anche diversa, caratterizzati da una ***chiave***, in base alla quale si ordina l'albero
  l'operazione base non è ottenere il record successivo, ma un record particolare individuato tramite la chiave



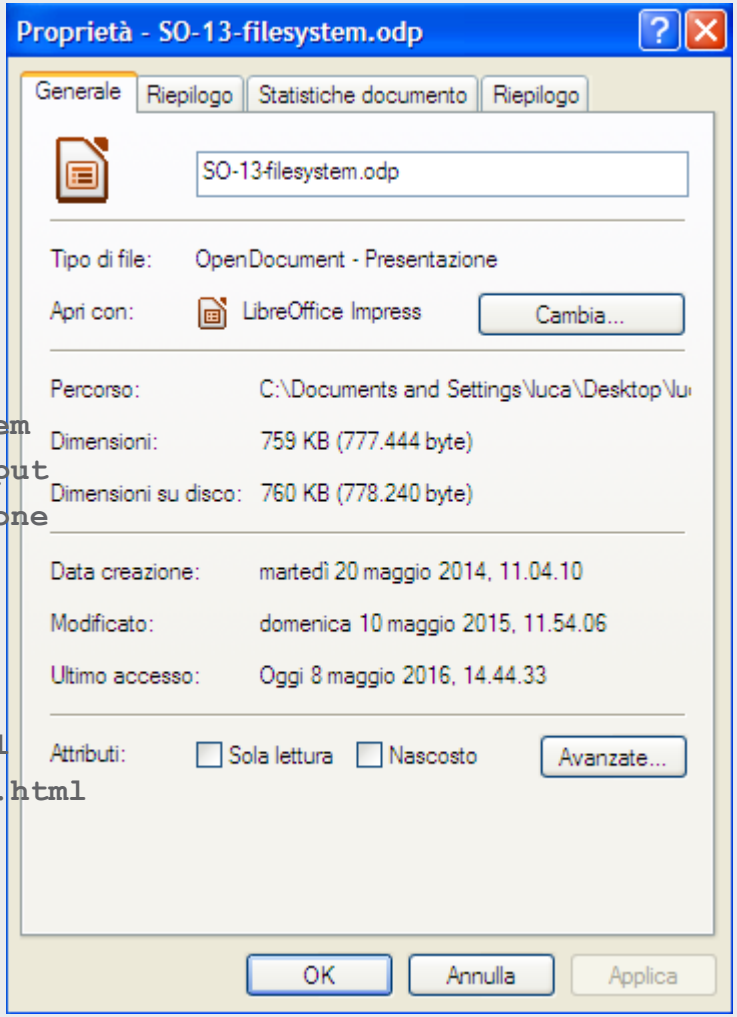
Attributi dei file

Name	Hidden	Key position
Type	System	Key length
Protection	Archive	Creation time
Password	Ascii/binary	Last access time
Creator	Random access	Last change time
Owner	Lock	Current size
Read-only	Record length	Maximum size



Attributi dei file

```
-rw-r--r--   1 luca     2351 Apr 25 10:01 0home.htm
drwxr-xr-x   2 luca      512 May 22 03:09 File-System
drwxr-xr-x   2 luca      512 Apr 25 10:02 Input-Output
dr-xr-xr-x   3 luca      512 Apr 18 14:29 Introduzione
drwxr-xr-x   2 luca      512 Apr 18 14:29 Memoria
dr-xr-xr-x   3 luca      512 Apr 18 14:29 Processi
-rw-r--r--   1 luca      650 Apr 25 10:01 home.html
dr-xr-xr-x   2 luca      512 Apr 18 14:29 imm
-rw-r--r--   1 luca     2011 Apr 25 10:01 index.html
-rw-r--r--   1 luca     1808 Mar  4 18:53 programma.html
-rw-r--r--   1 luca   138240 Mar 18 21:09 sis.ppt
drwxr-xr-x   2 luca      512 May  2 20:02 test
```



Operazioni sui file

- create
- delete
- open
- close
- read
- write
- seek
- get attributes
- set attributes
- rename

Tipi di file

- La maggior parte dei file system sono costituiti da directory e file
- I **file ordinari** sono costituiti dalle informazioni utilizzate dagli utenti
- Le **directory** gestiscono la struttura del file system
- In UNIX si hanno anche **file speciali a caratteri** per gestire i dispositivi di I/O e **file speciali a blocchi** per modellizzare i dischi
- MS-DOS riserva alcuni nomi per scopi speciali: con, nul
- I **file ordinari** sono spesso classificati in file ASCII costituiti da linee di testo e file binari, gli altri
- I file binari hanno generalmente una struttura interna gestita dai programmi

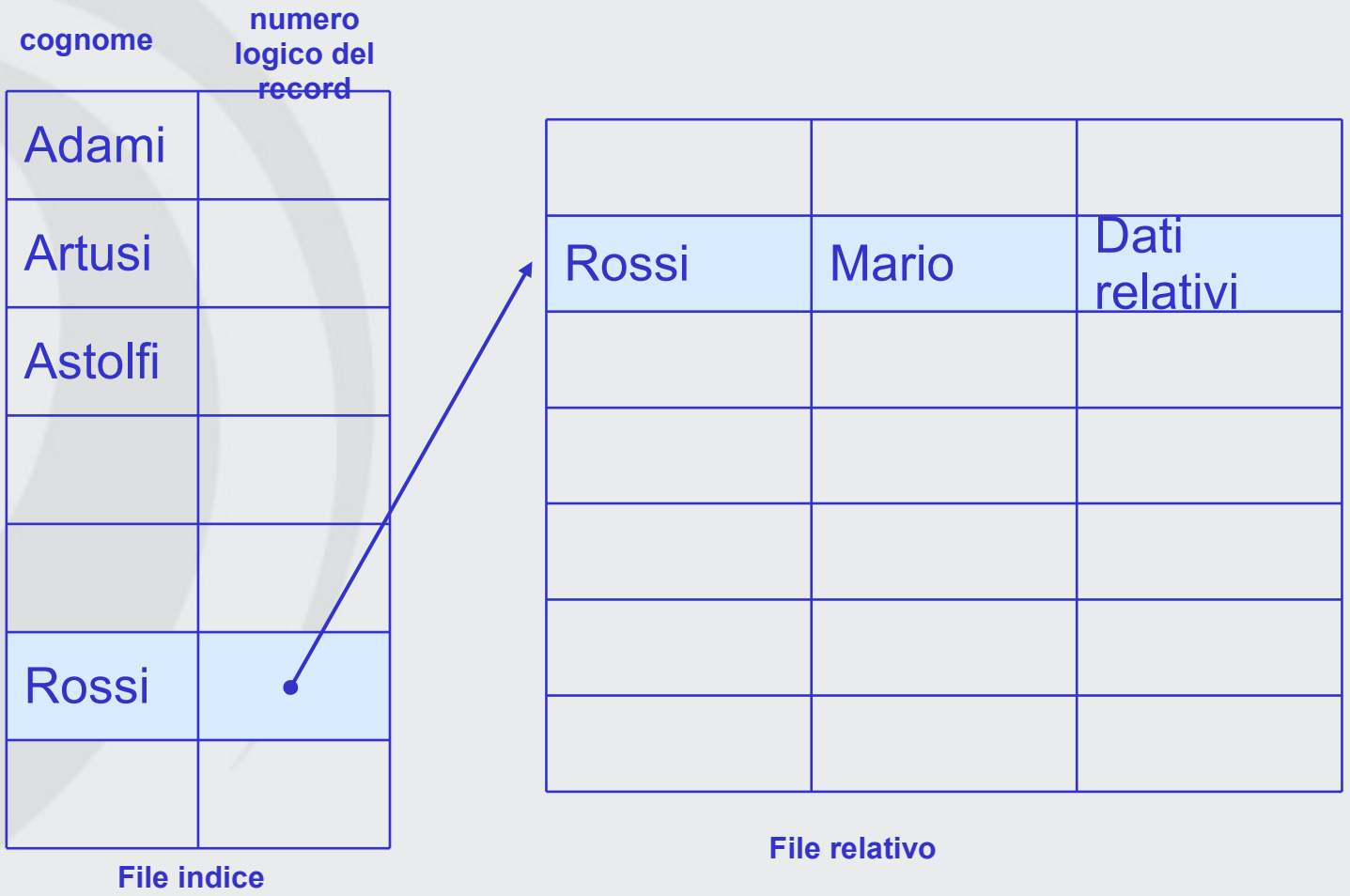
Tipi di file

Tipo di file	Usuale estensione
Eseguibile	exe, com, bin, nessuna
Oggetto	obj, o
Codice sorgente	c, cc, java, f, asm
Batch	bat, sh
Testo	txt
Elaboratore di testi	tex, doc, rtf
Libreria	lib, a, so, dll
Stampa o visualizzazione	ps, pdf, dvi
Archivio	zip, tar, tgz
Immagini, audio, multimedia	gif, tiff, jpg, au, wav, mpeg, mov

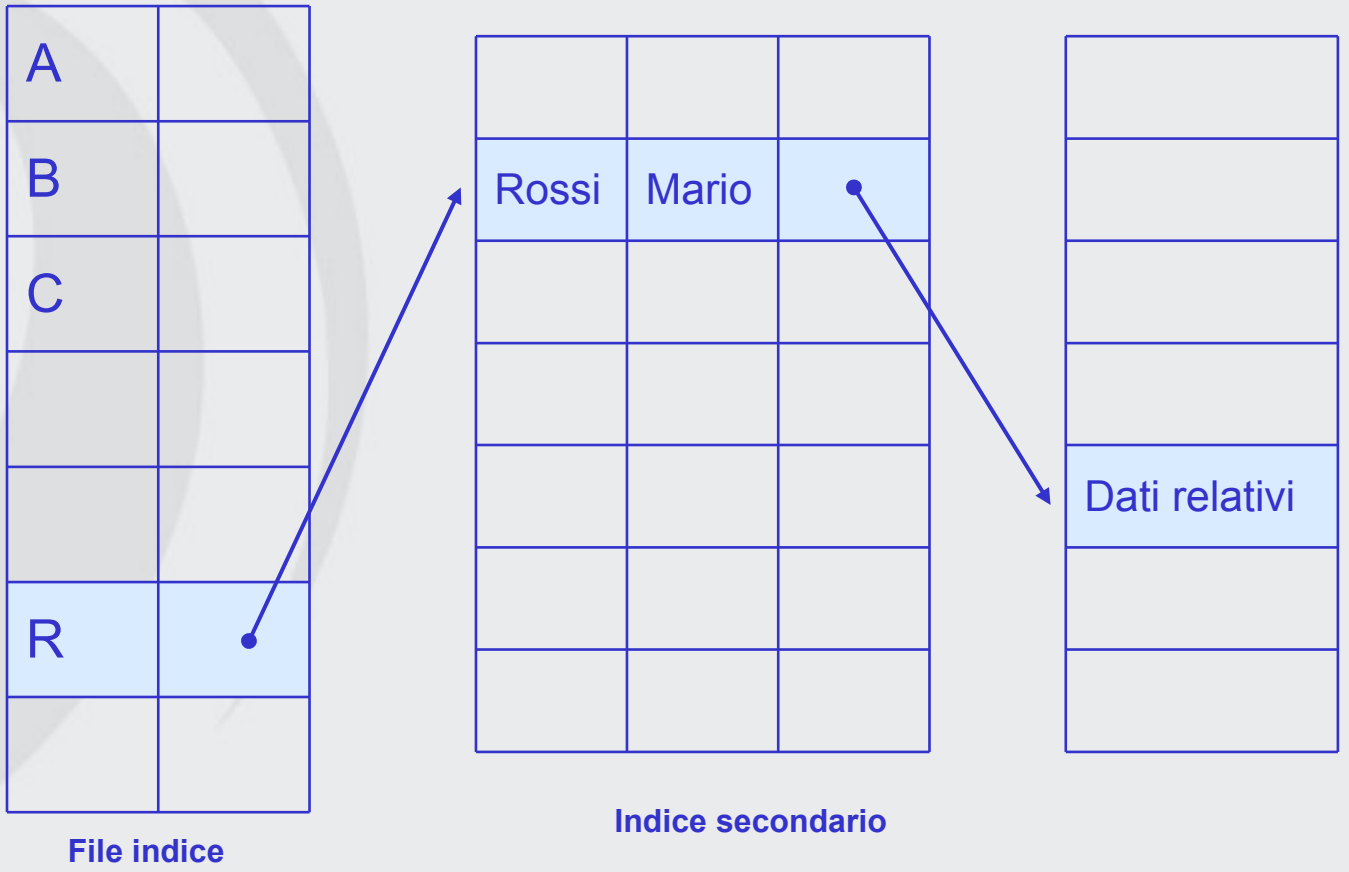
Accesso ai file

- Sequenziale
  - usato nei primi sistemi operativi
  - si basa sul modello di nastro
  - per accedere ad un dato occorre leggere tutte le registrazioni precedenti
- Casuale (random)
  - si basa sul modello disco
  - si può accedere ad ogni dato direttamente
  - implementato dai moderni sistemi operativi

Accesso indicizzato



Accesso indicizzato



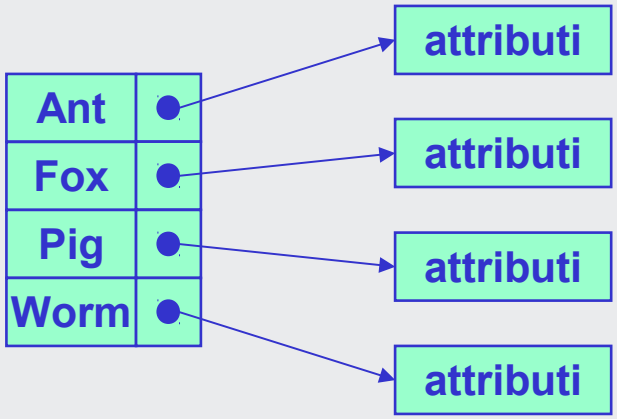
Directory

- Una directory è spesso essa stessa un file che contiene una voce per ogni file
- Due sono le organizzazioni utilizzate:
  - ogni voce contiene il nome e gli attributi del file

Ant	attributi
Fox	attributi
Pig	attributi
Worm	attributi

Directory

- ogni voce contiene il nome e un puntatore ad una struttura separata che contiene gli attributi del file

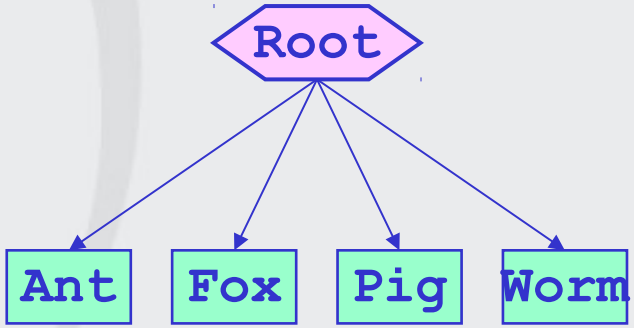


Directory

- La struttura del file system risultante può essere di tre tipi:
  - una unica directory
  - una directory per ogni utente
  - un albero di directory arbitrario

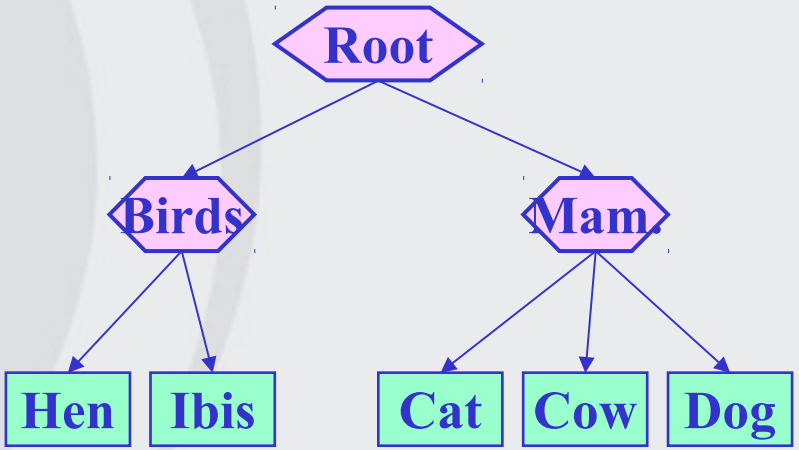
Directory unica

- Tutti i file di tutti gli utenti in una sola directory
- facile da implementare
- impraticabile in ambiente multiutente: causa conflitti sui nomi



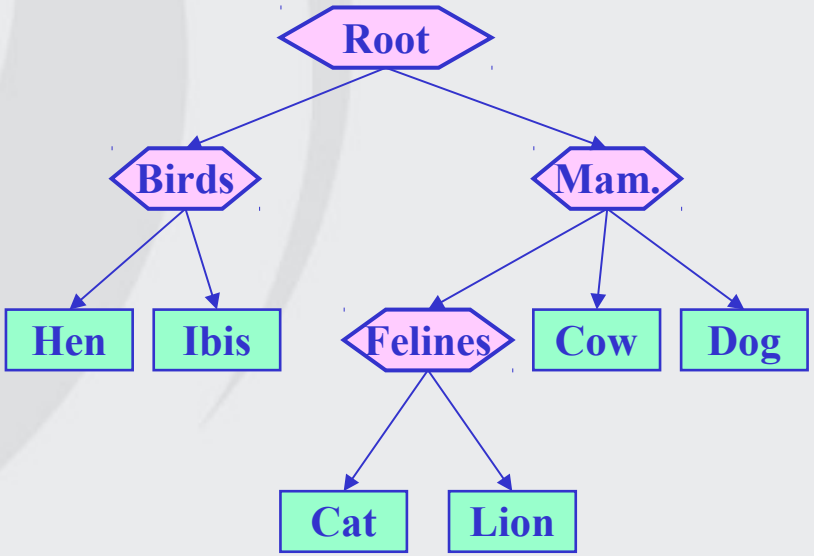
File System a due livelli

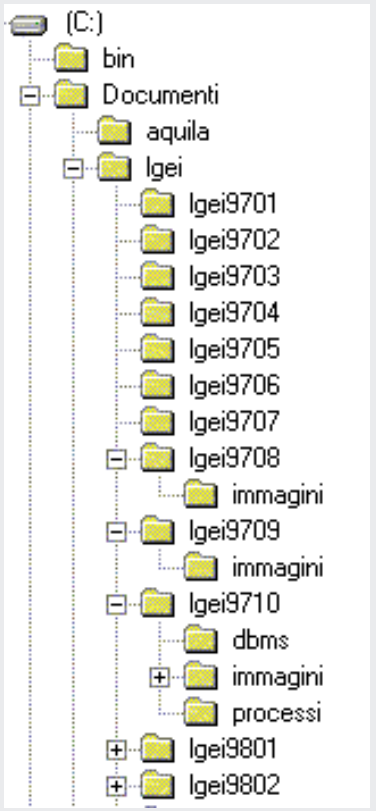
- Tutti i file di ogni utenti in una directory separata
- possono ancora esistere conflitti sui nomi



File System ad albero

- Più directory per ogni utente organizzazione gerarchica
- conflitti sui nomi minimi
- organizzazione flessibile





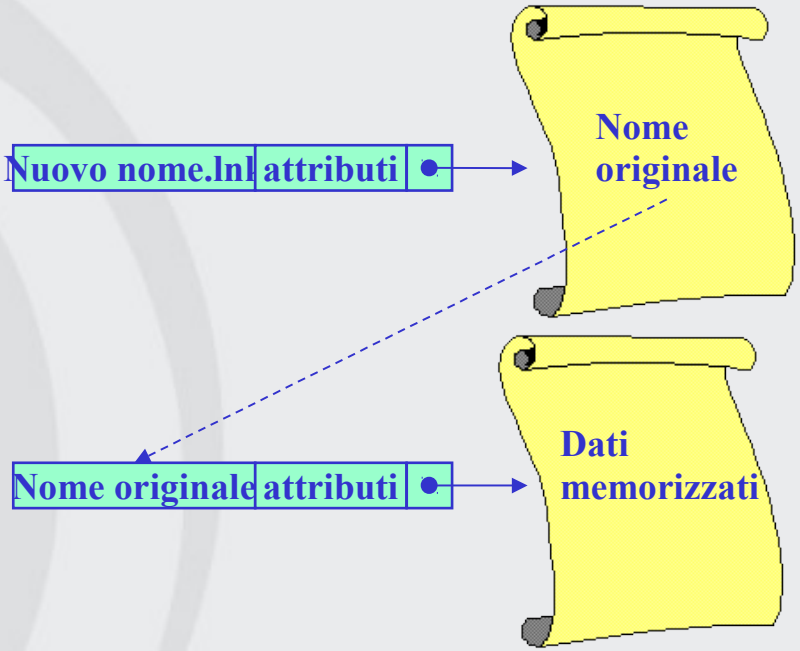
Operazioni su directory

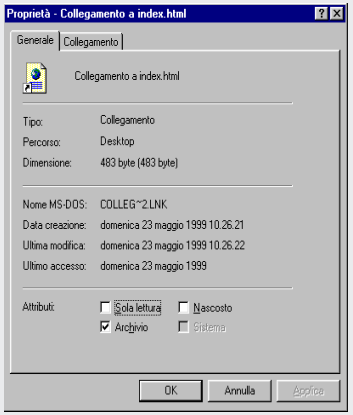
create	crea una directory vuota ad eccezione di . e ..
delete	cancella una directory vuota
opendir	apre la directory per la consultazione
closedir	chiude la directory
readdir	restituisce la voce successiva
rename	cambia nome
link	aggiunge una voce alla directory
unlink	elimina una voce

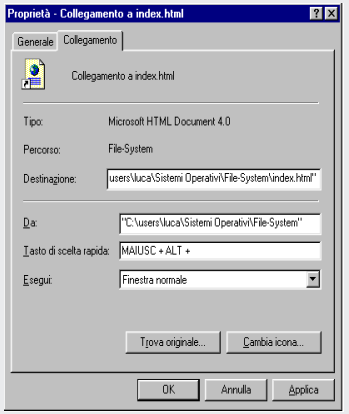
Link

- Sono scorciatoie per accedere a file o directory
  - usati frequentemente
  - condivisi
  - usati attraverso nomi diversi
- Permettono di avere più di un punto di accesso per lo stesso file o directory
  - le informazioni rimangono in una unica copia

Implementazione di Windows



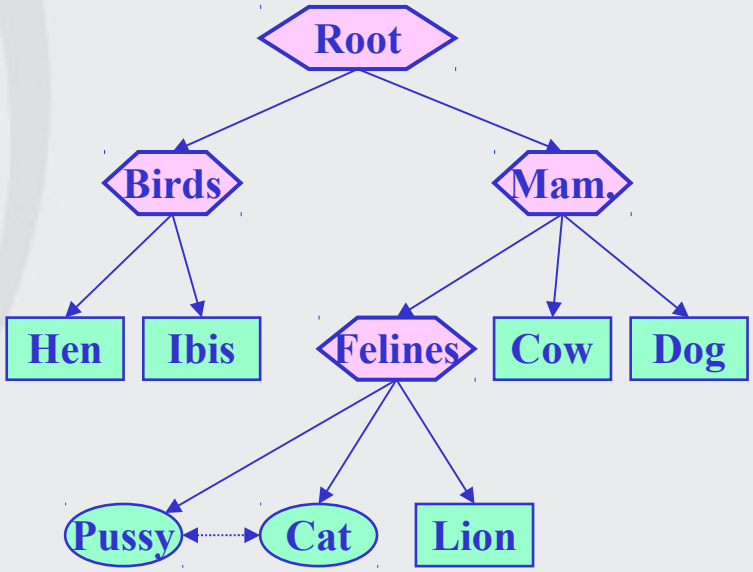




Viene creato un file che contiene un certo numero di informazioni fra cui il nome del file a cui ci si riferisce
È possibile un solo livello di collegamento

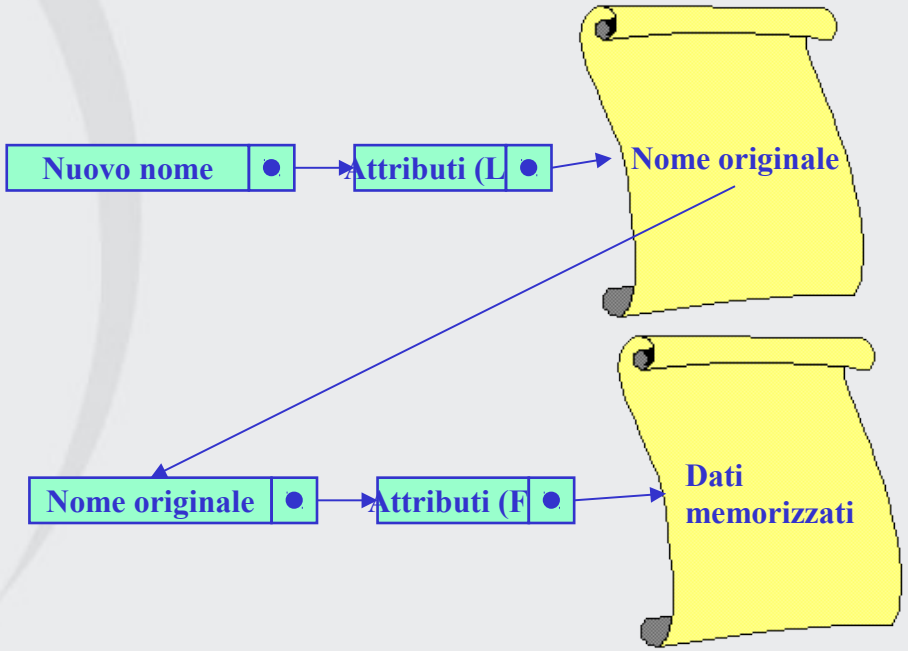
Implementazione UNIX

- Sono possibili più livelli di *collegamento*
  - Problema: il file system non è più un albero, ma un grafo che può contenere cicli



Implementazione UNIX

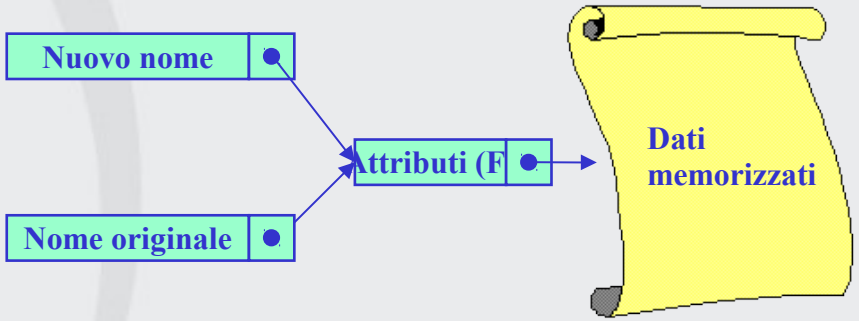
- Due tipi di link:
  - soft link: simili ai collegamenti di windows
    il file creato contiene solo il nome del riferimento



  - hard link

Implementazione UNIX

- Due tipi di link:
  - soft link: simili ai collegamenti di Windows
    il file creato contiene solo il nome del riferimento
  - hard link: non viene creato alcun file, si fa riferimento agli stessi dati fisici



- i due nomi diventano equivalenti
- NTFS ha introdotto con Windows un meccanismo analogo

Effetti sull'uso dei link

- *nuovo* è un soft link a *vecchio*
  - posso usare indifferentemente *nuovo* o *vecchio* per modificare i miei dati
  - se cancello *vecchio* perdo i miei dati, *nuovo* rimane, ma nel momento in cui lo uso avrò un errore (è un puntatore non inizializzato correttamente)
  - se cancello *nuovo* perdo solo un modo di accedere ai dati
- *nuovo* è un hard link a *vecchio*
  - posso usare indifferentemente *nuovo* o *vecchio* per modificare i miei dati
  - se cancello *vecchio* posso ancora accedere ai dati tramite *nuovo*
  - se cancello *nuovo* posso ancora accedere ai dati tramite *vecchio*
  - se cancello *vecchio* e poi lo ricreo ho ora due insiemi di dati diversi

Allocazioni dei file

- Esigenze da soddisfare:
  - accesso veloce ai dati
  - utilizzazione efficiente del disco
- Metodi di allocazione:
  - contigua
  - a liste
  - indicizzata

Allocazione contigua

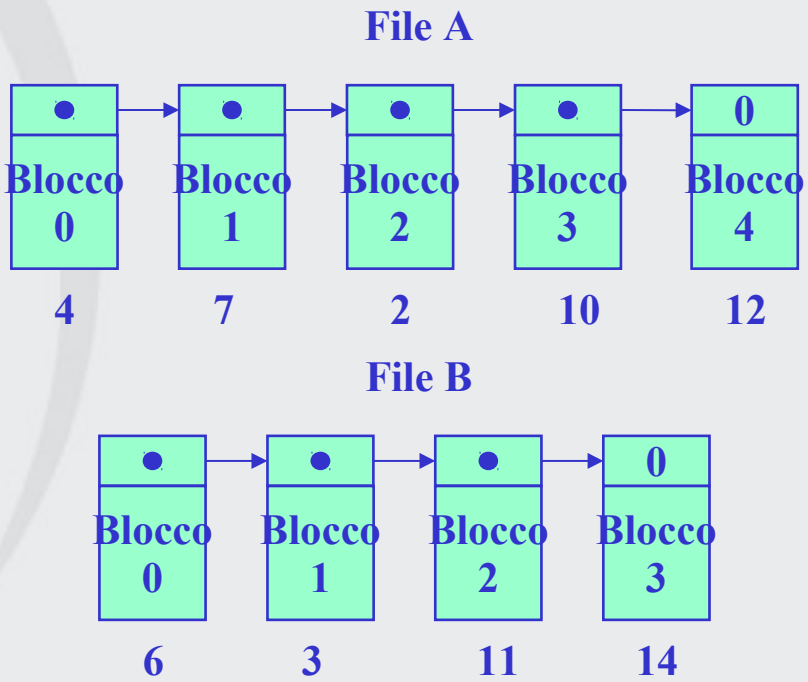
- Ogni file occupa un insieme contiguo di blocchi su disco, allocati al momento della creazione del file
- implementazione semplice, è sufficiente una tabella che contiene

Nome file	Blocco di partenza	lunghezza

- occorre sapere subito la dimensione del file
- per l'allocazione si usano algoritmi simili a quelli per la gestione di memoria primaria: first fit, best fit
- prestazioni eccellenti lettura e scrittura avvengono tramite un unico blocco
- problemi:
  - frammentazione
  - espansione dei file

Allocazione con liste

- Un file è gestito tramite una lista di blocchi



Allocazione con liste

- le directory contengono solo i puntatori al primo blocco
- estendere un file è semplice
- non esiste frammentazione esterna
- lentezza di accesso (l'accesso casuale non è semplice)
- i blocchi su disco devono contenere un puntatore (la dimensione del blocco logico non è una potenza di due)

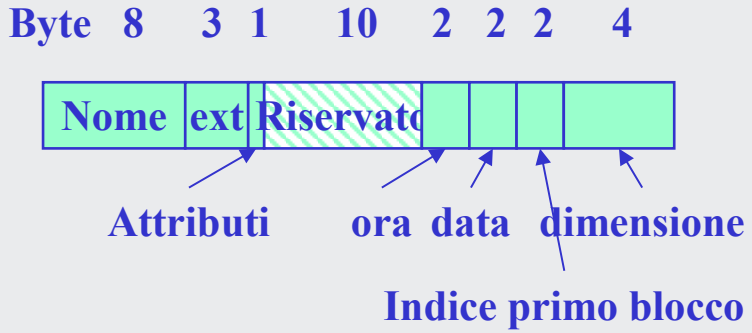
Allocazione indicizzata

- Risolve i problemi precedenti
  - Tutti i puntatori sono memorizzati insieme in un unico blocco (blocco indice)
  - il blocco indice viene conservato in memoria primaria
  - l'accesso casuale è ottimizzato in quanto la catena di puntatori è interamente in memoria
  - il blocco indice può raggiungere dimensioni notevoli

File Allocation Table

- MS-DOS utilizza l'allocazione indicizzata
  - ogni **partizione** ha una sua FAT
  - la tabella ha una voce per ogni blocco: il numero del blocco successivo
  - per i blocchi non usati un puntatore **nullo**
  - per limitare l'occupazione di memoria i blocchi possono essere di grande dimensione

- Struttura delle directory:



FAT

FAT12: usa 12 bit (al massimo 4096 blocchi)

- Dimensione: 4096*12 bit = circa 6 KB
- Per un disco di 100MB la dimensione del blocco risulta 32KB

FAT16: usa 16 bit (circa 65000 blocchi)

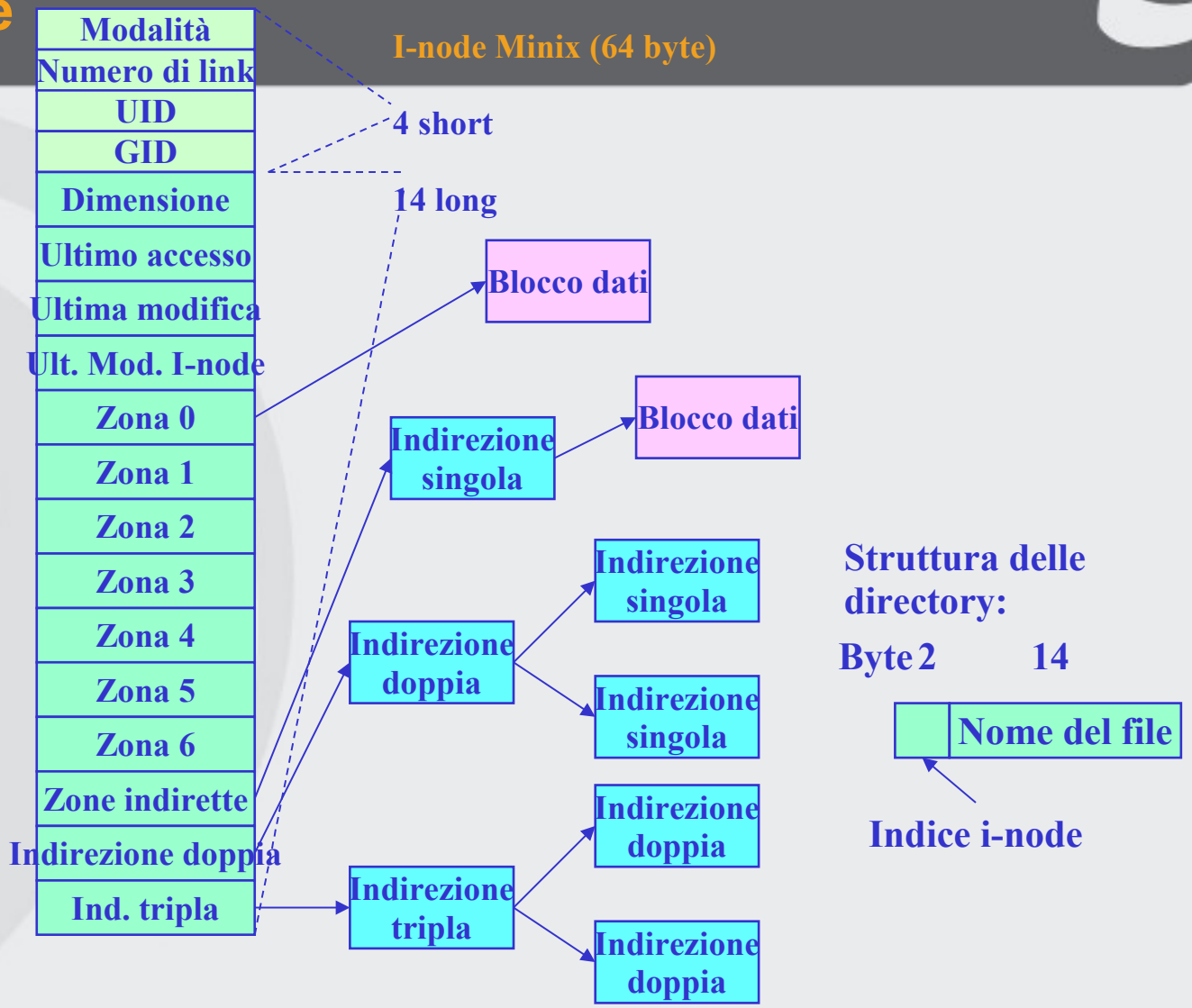
- Dimensione: 65536*2 Byte = circa 128 KB
- Per un disco di 100MB la dimensione del blocco risulta 2KB

Dim. blocco	FAT-12	FAT-16	FAT-32
512B	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1 GB	2 TB
32 KB		2 GB	2 TB

I-Node

- Il file system UNIX è basato sugli i-node:
  - gli attributi dei file sono conservati separatamente dalle directory in una struttura dati chiamata **i-node** (index-node)
  - ogni i-node contiene anche i puntatori ai primi blocchi del file
  - se non sono sufficienti uno dei blocchi **(blocco a indirezione semplice)** è utilizzato per contenere altri indirizzi di blocchi
  - se nemmeno questo è sufficiente si utilizza un secondo livello **(blocco a indirezione doppia)**, nei casi estremi si può arrivare ad avere **blocchi a indirezione tripla**

I-Node



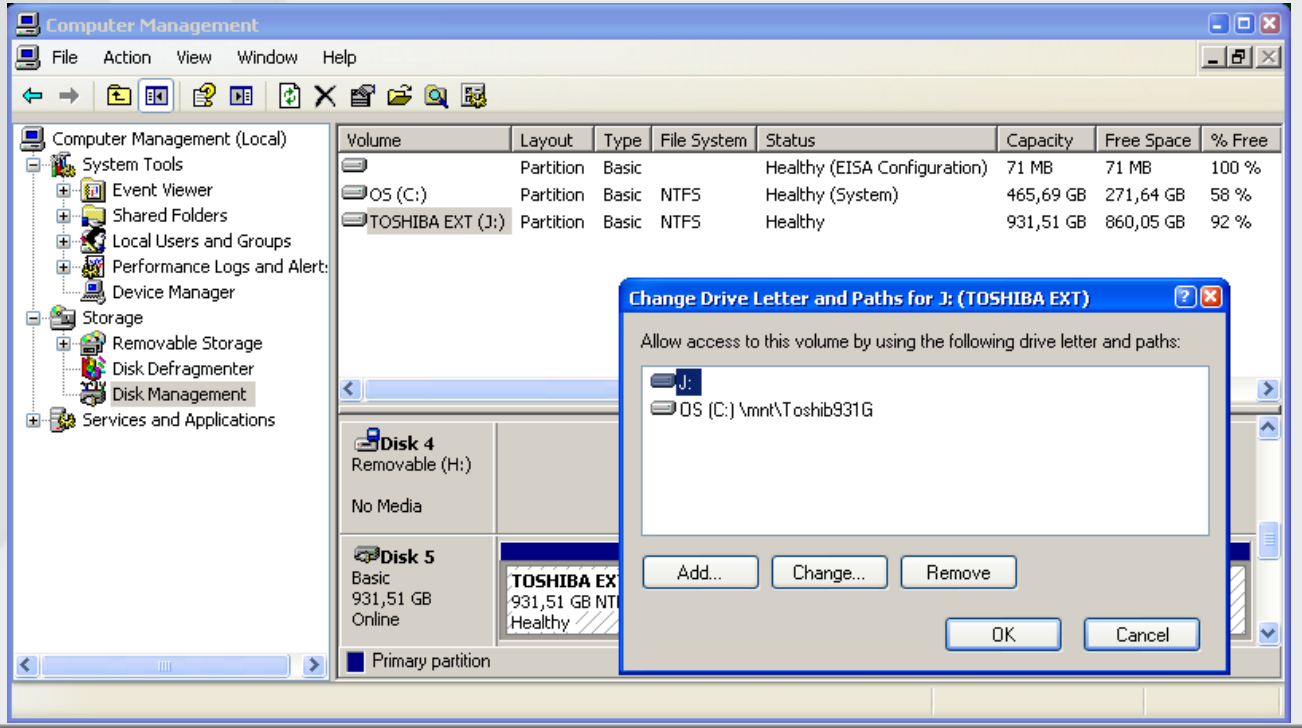
I-Node Linux Ext2 128 byte 12 indirizzi diretti a blocchi dati

Utilizzo di un File-System

- Comando UNIX mount
- Uso: mount <dispositivo> <directory>
- NB:
  - La directory deve essere vuota
  - Il comando può avere dei parametri opzionali
  - Il comando può essere eseguito solo dall'amministratore del sistema
- Esempi:
  - mount -t iso9660 /dev/cdrom /cdrom
  - mount -t nfs pippo.unipv.it:/users /users
  - mount -t iso9660 /tmp/cdrom-image /cdrom -o loop (monta un file come se fosse un dispositivo)

Utilizzo di un File-System

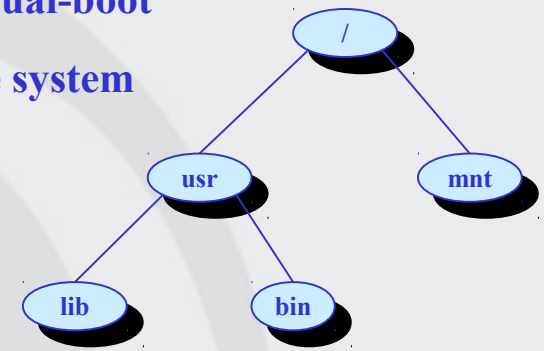
- WINDOWS tradizionalmente non ha un comando equivalente, tutti i dispositivi sono caricati automaticamente dal sistema e sono visti come drive (lettera:)
- FS remoti possono essere montati dall'utente
- Con NTFS esiste la possibilità di montare un FS in una directory



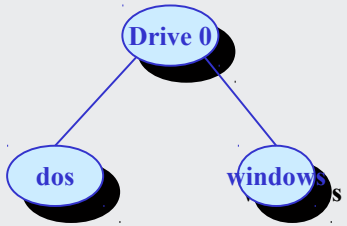
Mount

Esempio di macchina dual-boot

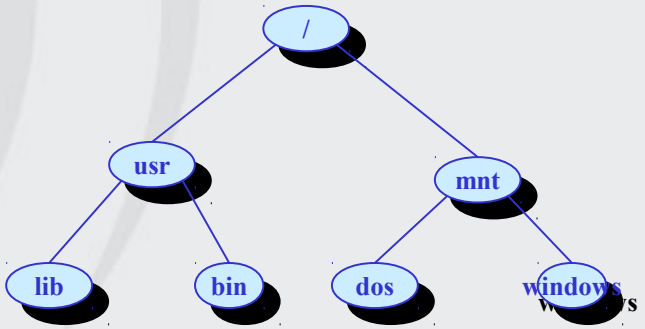
Prima di montare il file system



Partizione dos (non visibile)




```
mount -t msdos /dev/hda1 /mnt
```



Ciò che dos vedeva come c:\windows\... diventa sotto UNIX /mnt/windows/... (purché il sistema sia in grado di trattare file-system FAT)

File /etc/fstab

```
/dev/hda1       /              ext2      defaults        1 1
/dev/hda2       /home          ext2      defaults        1 2
/dev/hda4       /usr           ext2      defaults        1 2
/dev/hda3       swap           swap      defaults        0 0
/dev/fd0        /mnt/floppy    vfat      noauto,user     0 0
192.168.0.77:/  /mnt/nfs       nfs       noauto          0 0
none            /proc          proc      defaults        0 0
```

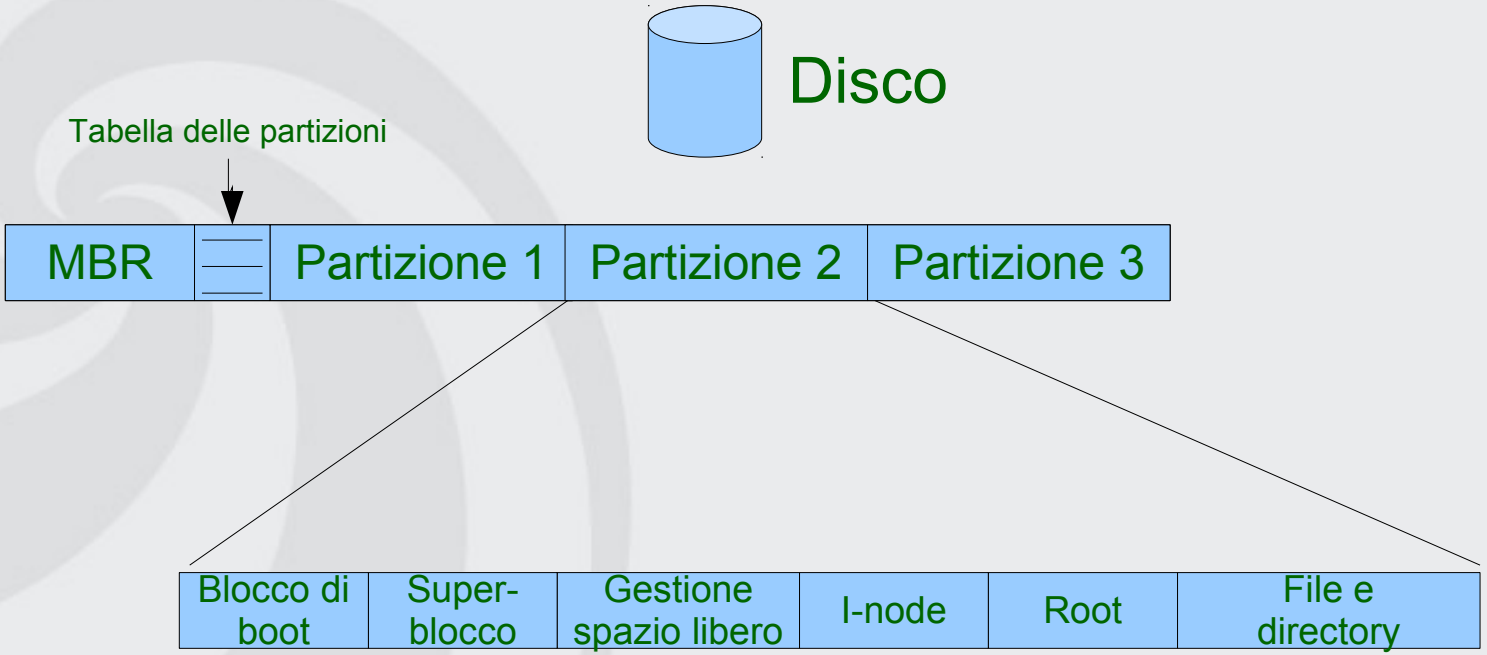
In questo caso qualunque utente può dare il comando

- mount /mnt/floppy

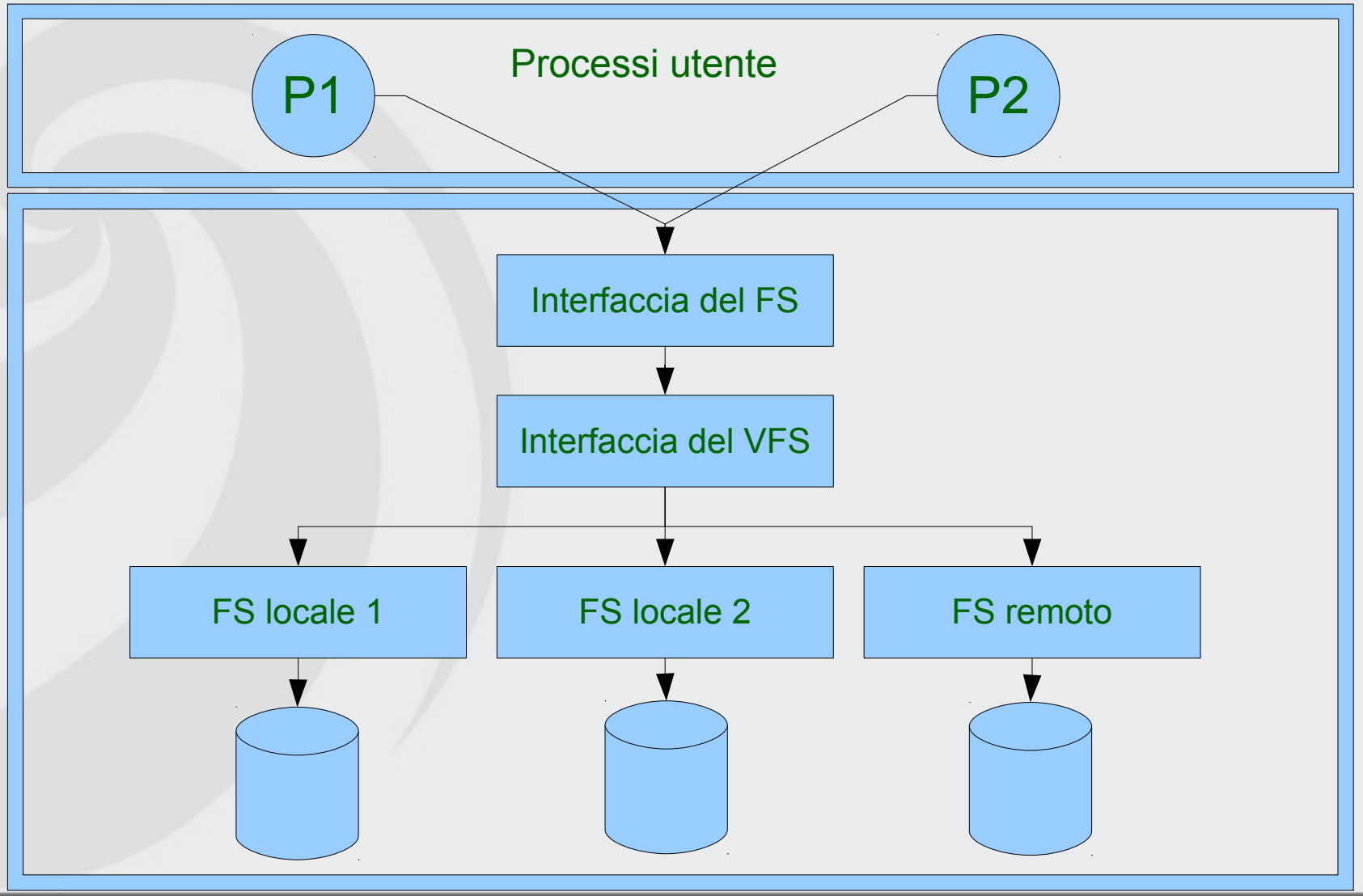
Umount

- Comando UNIX umount
- Uso: umount <dispositivo> o <directory>
- NB:
  - Il comando può essere eseguito solo dall'amministratore del sistema
- Esempi:
  - `umount /cdrom`
  - `umount /dev/cdrom`

Struttura del File System



File System Virtuali



VFS

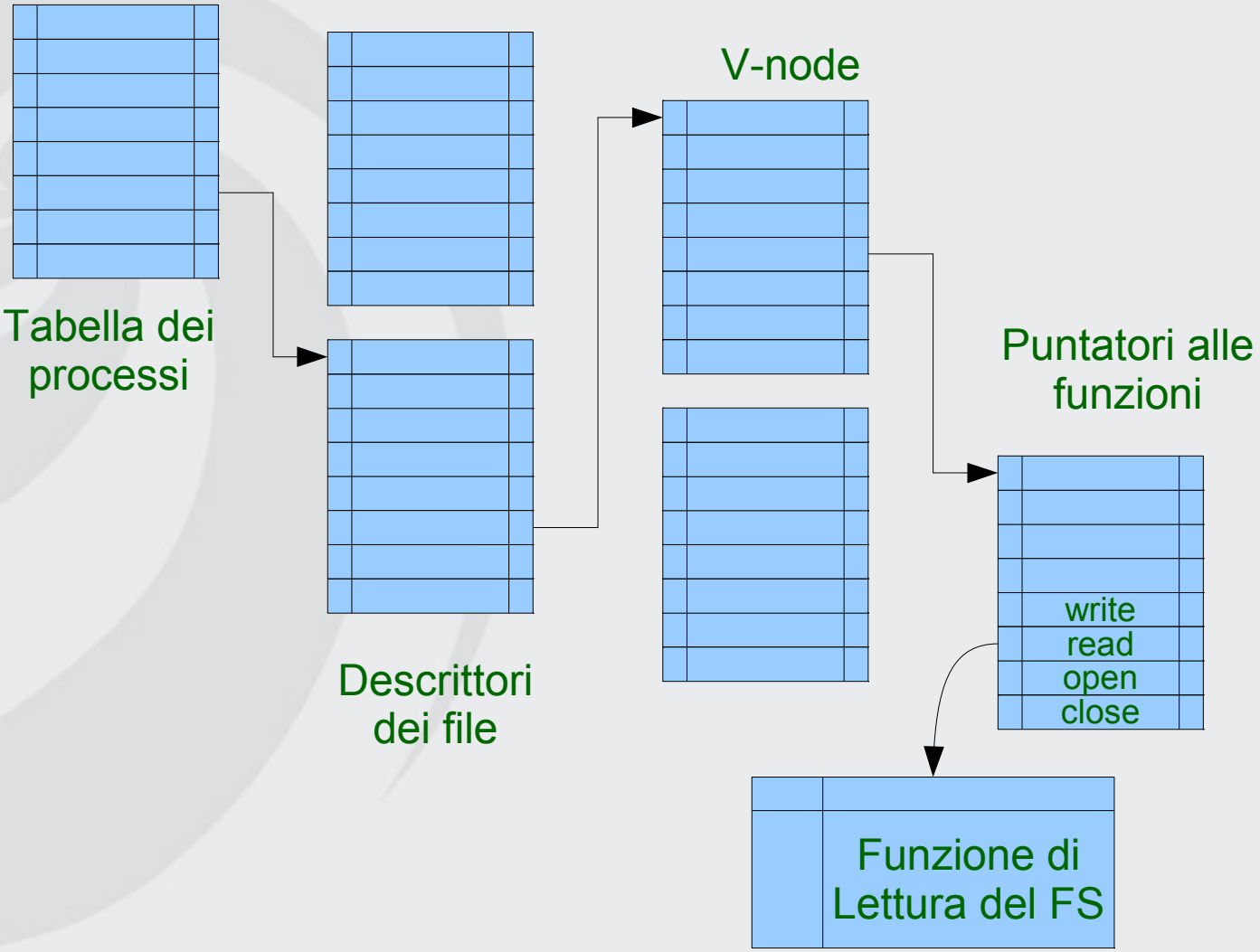


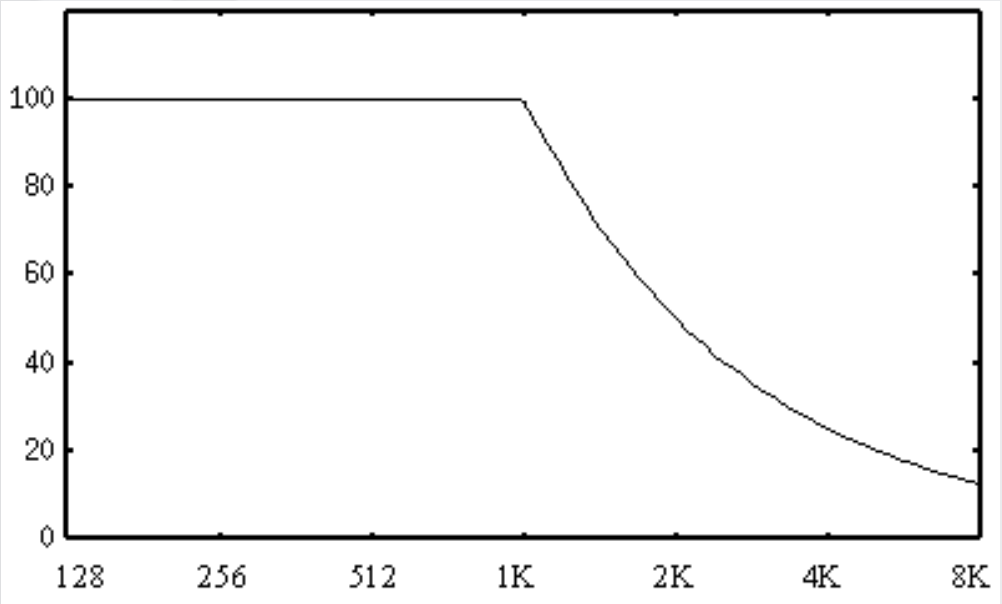
Tabella dei processi
Descrittori dei file
V-node
Puntatori alle funzioni
write
read
open
close
Funzione di Lettura del FS

Dimensioni dei blocchi

- Come scegliere la dimensione dei blocchi?
- I possibili candidati sono:
  - Parametri del disco:
    cilindro, traccia, settore
  - Parametri del sistema:
    dimensione di pagina
- Ottimizzazione dello spazio occupato

Dimensioni dei blocchi

- Se la dimensione media dei file è minore della dimensione del blocco si ha uno spreco di spazio che può essere notevole



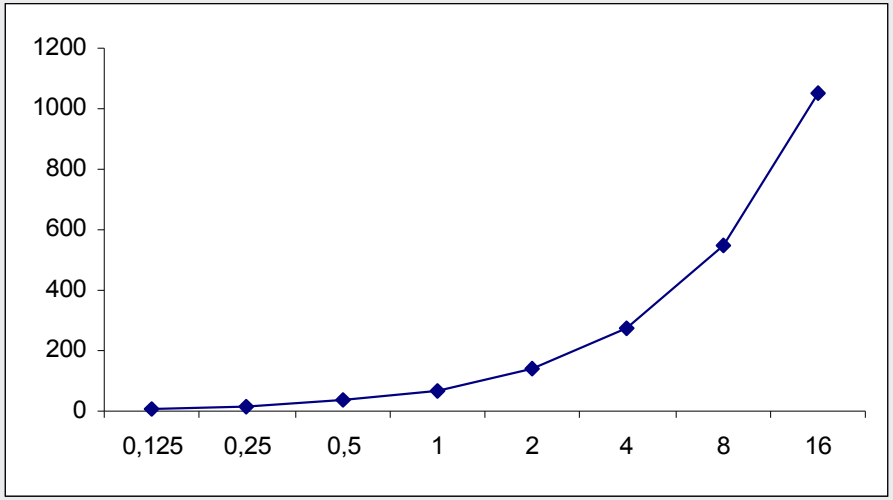
Dimensioni dei blocchi

- Ottimizzazione del tempo di accesso ai dati:

  Tempo di lettura di un blocco = 10 + (B/128K+0.5)*8.33)

Vel = DimBlocco/TempoLettura

KByte al secondo



128 Kbyte per traccia
Tempo di rotazione 8.33 ms

Dimensioni dei blocchi

- I due parametri considerati hanno esigenze opposte
- Un parametro che può essere utilizzato è la dimensione media dei file utilizzati

Gestione dei blocchi liberi

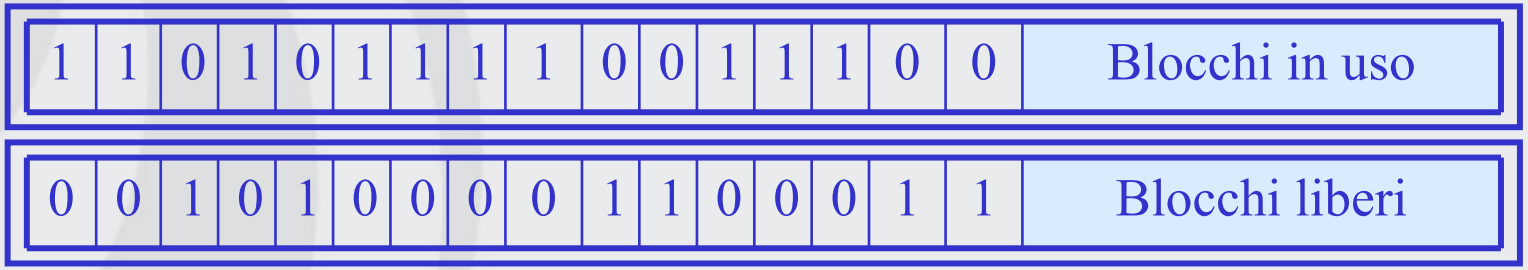
- Si utilizzano principalmente due tecniche:
  - si riservano alcuni blocchi per gestire una lista dei blocchi liberi
  - si mantiene una bitmap

Affidabilità del file system

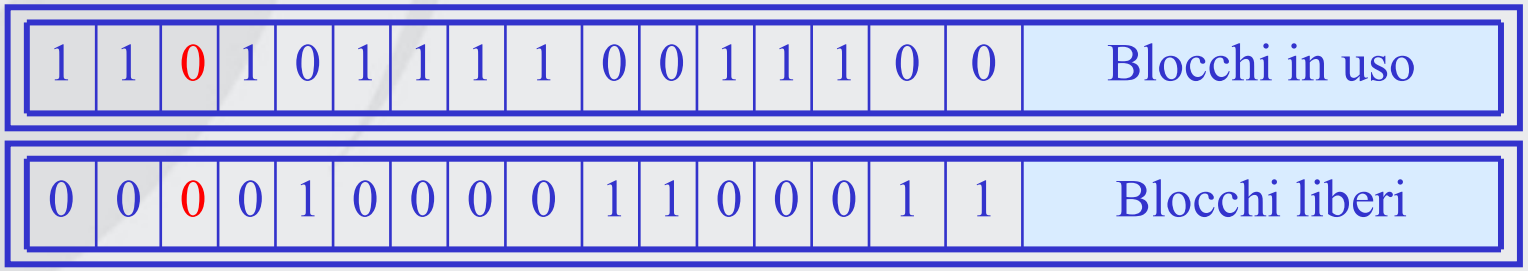
- Un file system deve essere protetto da danneggiamenti, sia hardware che software
- I dischi generalmente contengono settori di riserva, che possono sostituire settori che nel tempo si danneggiano (soluzione a posteriori)
- La soluzione più comune è il backup dei dati, tradizionalmente su nastro

Consistenza del file system

- Consistenza dei blocchi:
- Si confrontano le liste (o bitmap) dei blocchi liberi e utilizzati:
- Quattro casi possibili:
  - Nessun errore

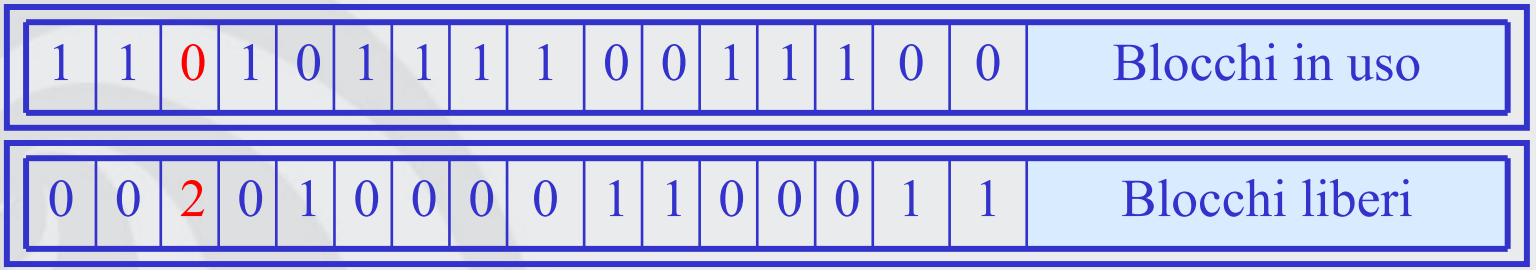


  - Blocco mancante



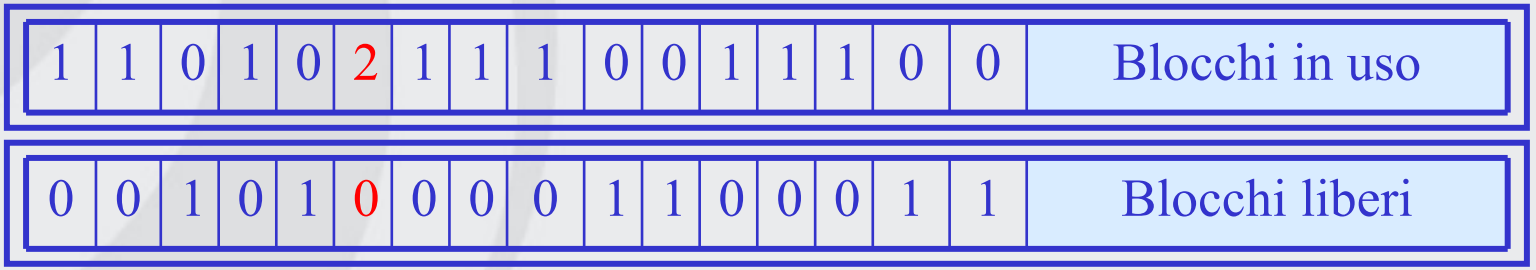
Consistenza del file system

- Blocco libero duplicato



con le bitmap non può accadere

- Blocco utilizzato duplicato



- Consistenza della struttura delle directory:
  - Il contenuto delle voci nelle directory è confrontato con i file esistenti

Consistenza del file system

- Qual è la causa delle inconsistenze?
- Uso tipico dei file:
  - si legge un blocco da disco
  - si modifica il blocco in memoria
  - lo si riscrive
- Se si verifica un crash:
  - alcuni blocchi non sono scritti
  - il file system diventa inconsistente

Prestazioni del file system

Per migliorare le prestazioni parte dei blocchi su disco sono tenuti in un buffer in memoria (cache di disco)

l'implementazione è analoga alla gestione della paginazione

- quante volte un blocco viene aggiornato?
  - periodicamente (UNIX)
  - ad ogni richiesta di scrittura (MS-DOS)

  estrarre un dischetto sotto MS-DOS non è critico, con UNIX occorre *smontare* il dispositivo

Blocchi utilizzati in sequenza dovrebbero essere memorizzati vicino per minimizzare i tempi di posizionamento delle testine (deframmentazione del disco - richiede però molto tempo)

CVMLab
Computer Vision
& Multimedia Lab

Esercizi

Università
degli Studi
di Pavia

Allocazione di file

- Un sistema UNIX (esempio file system ext2) usa blocchi di 1KB e indirizzi di 32 bit. Qual è la dimensione massima di un file se:
  - Gli I-node contengono 12 puntatori diretti a blocchi dati
  - + un puntatore ad un blocco indiretto
  - + un puntatore ad un blocco a doppia indirezione
  - Ha senso avere un puntatore ad un blocco a tripla indirezione?

I-node

Dimensione blocco: 1024

Blocco indice: contiene 1024/4=256 indirizzi

→ accedo a 256 × 1024 = 256K dati

Indirezione doppia: contiene 256 indirizzi di blocchi indice

→ accedo a 256 × 256K = 64M dati

Indirezione tripla:

→ accedo a 256 × 64M = 16G dati

Numero massimo di blocchi gestibili: $12 + 256 + 256^2 + 256^3$

I-node

Dimensione blocco: 4096

Blocco indice: contiene 4096/4=1024 indirizzi

→ accedo a $1024 \times 4096 = 4M$ dati

Indirezione doppia: contiene 1024 indirizzi di blocchi indice

→ accedo a $1024 \times 4M = 4G$ dati

Indirezione tripla:

→ accedo a $1024 \times 4G = 4T$ dati

Numero massimo di blocchi gestibili: $12 + 1024 + 1024^2 + 1024^3$

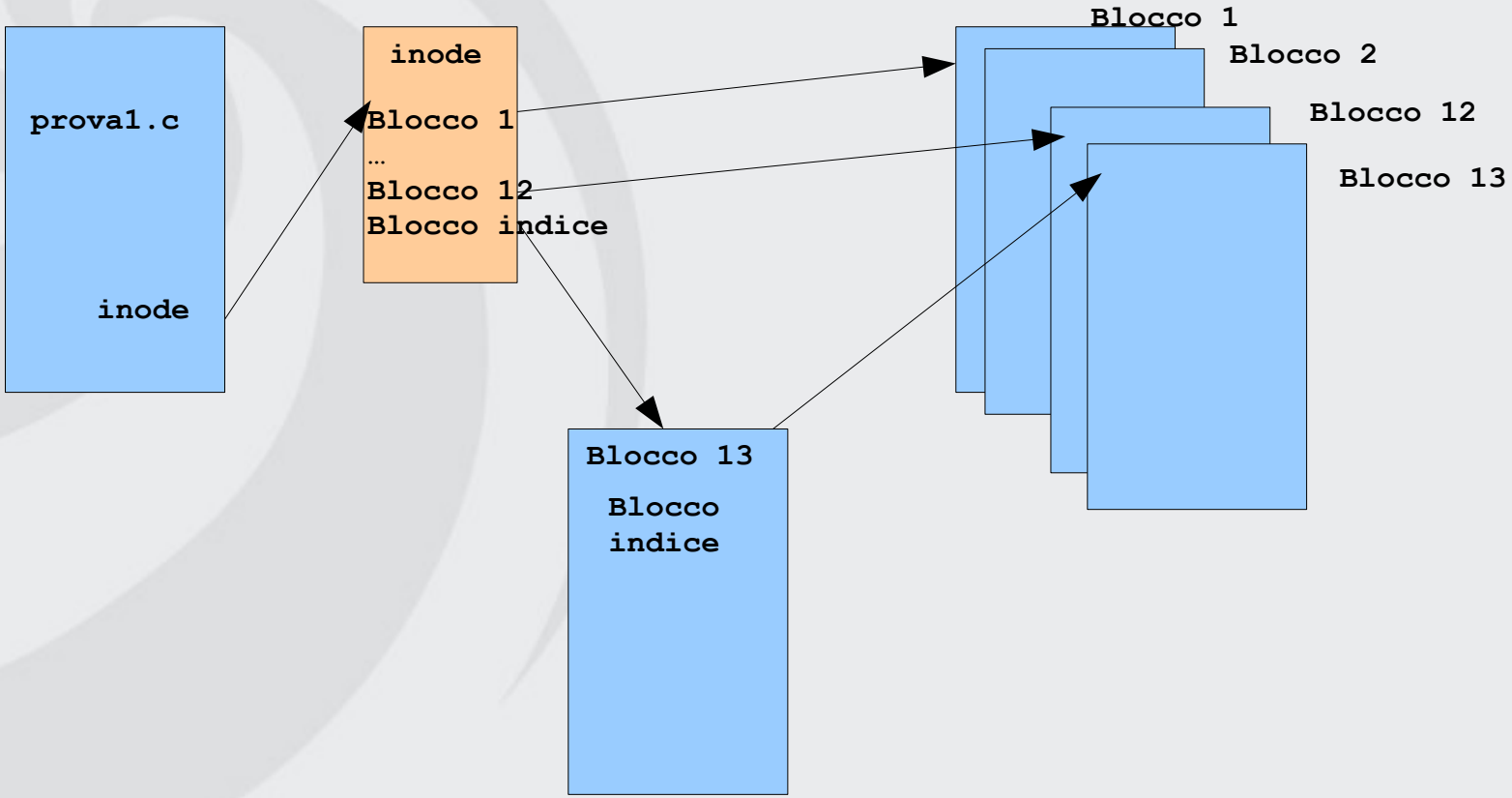
Link

```
$ ls -l prova.c
-rw-r--r--    1 luca     staff       50000 Dec 29 18:45 prova.c
$ cp prova.c prova1.c
$ ln -s prova.c prova2.c
$ cat prova2.c
```

Come varia il numero di blocchi e il numero di I-node? Valutare il tempo necessario per il comando cat

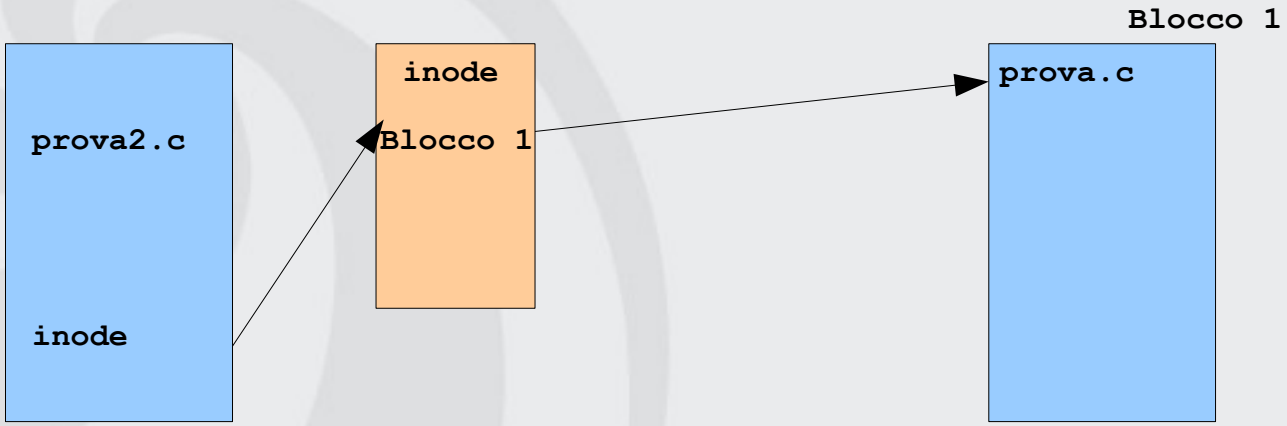
Copia

50000 byte con blocchi da 4KB occorrono 13 blocchi



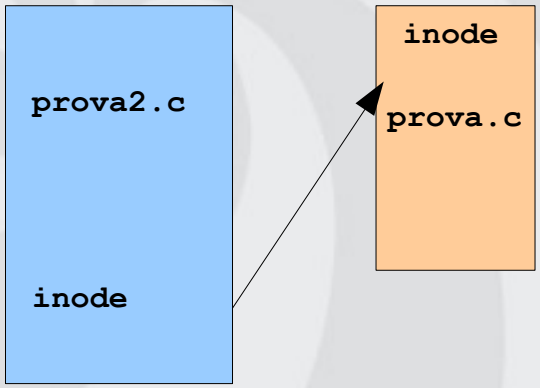
Link

prova2.c è un soft link a prova1.c



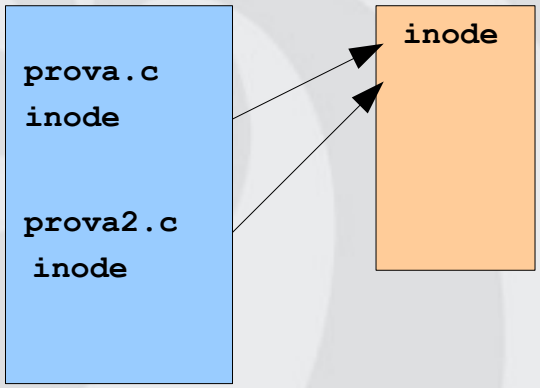
Ho usato un blocco e un inode

Link



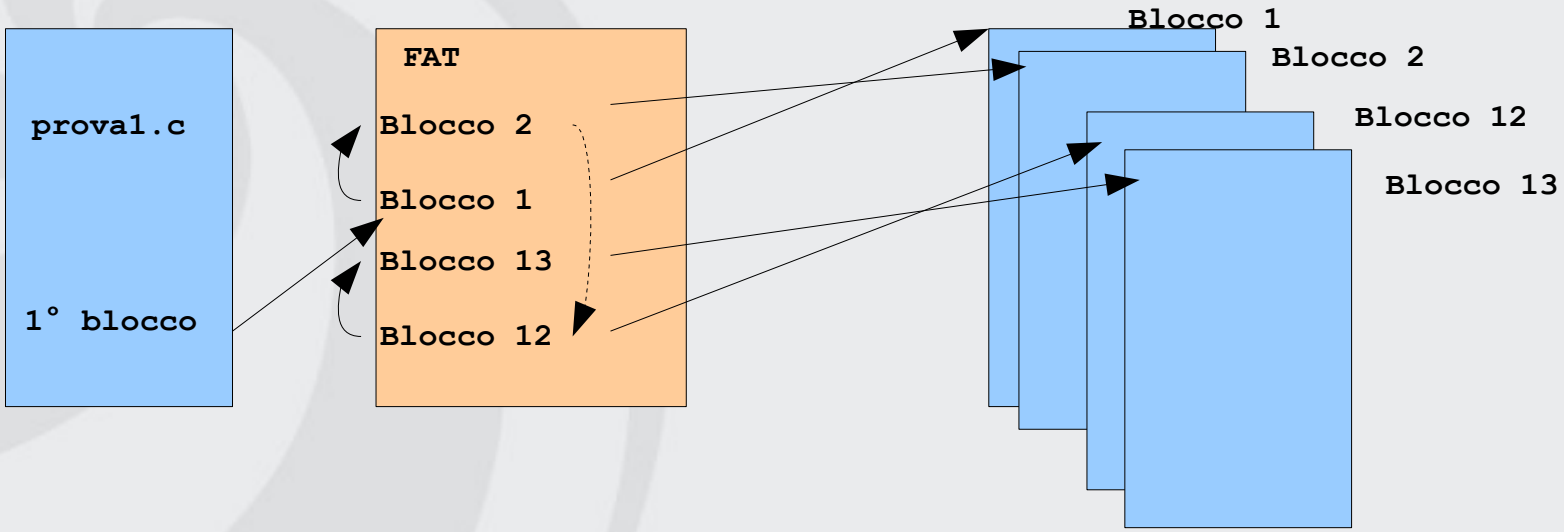
Ho usato solo un inode, il nome è al posto degli indirizzi, con nome `luuuuuuuuuuuuuuuuuuuuungo` non funziona

Hard link



Non ho usato risorse (a parte spazio nella directory)

FS FAT

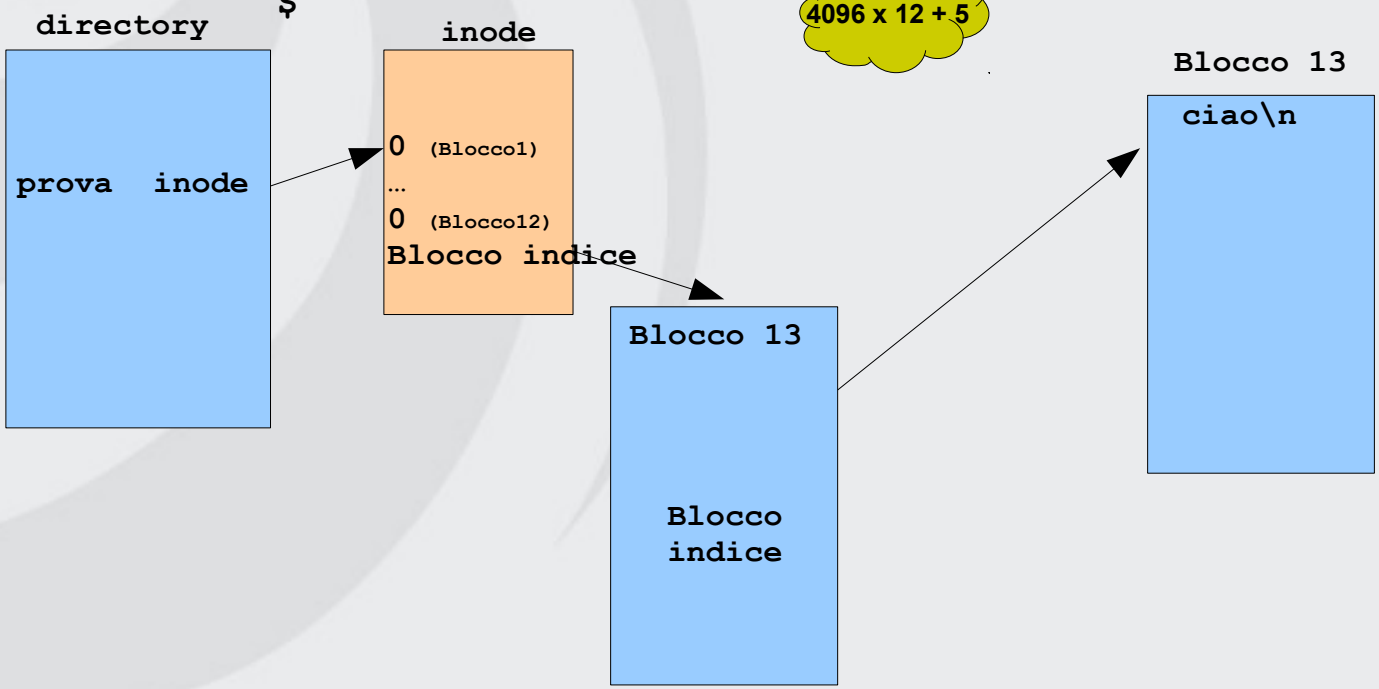

prova1.c
1° blocco
FAT
Blocco 2
Blocco 1
Blocco 13
Blocco 12
Blocco 1
Blocco 2
Blocco 12
Blocco 13

dd

```
$ echo ciao | dd bs=4096 seek=12 of=prova
0+1 records in
0+1 records out
5 bytes (5 B) copied, 0.00161027 s, 3.1 kB/s
$ ls -l prova
-rw-r--r-- 1 user user 49157 Dec 19 18:33 prova
$ du -h prova
8.0K    prova
$
```

2 blocchi

4096 x 12 + 5

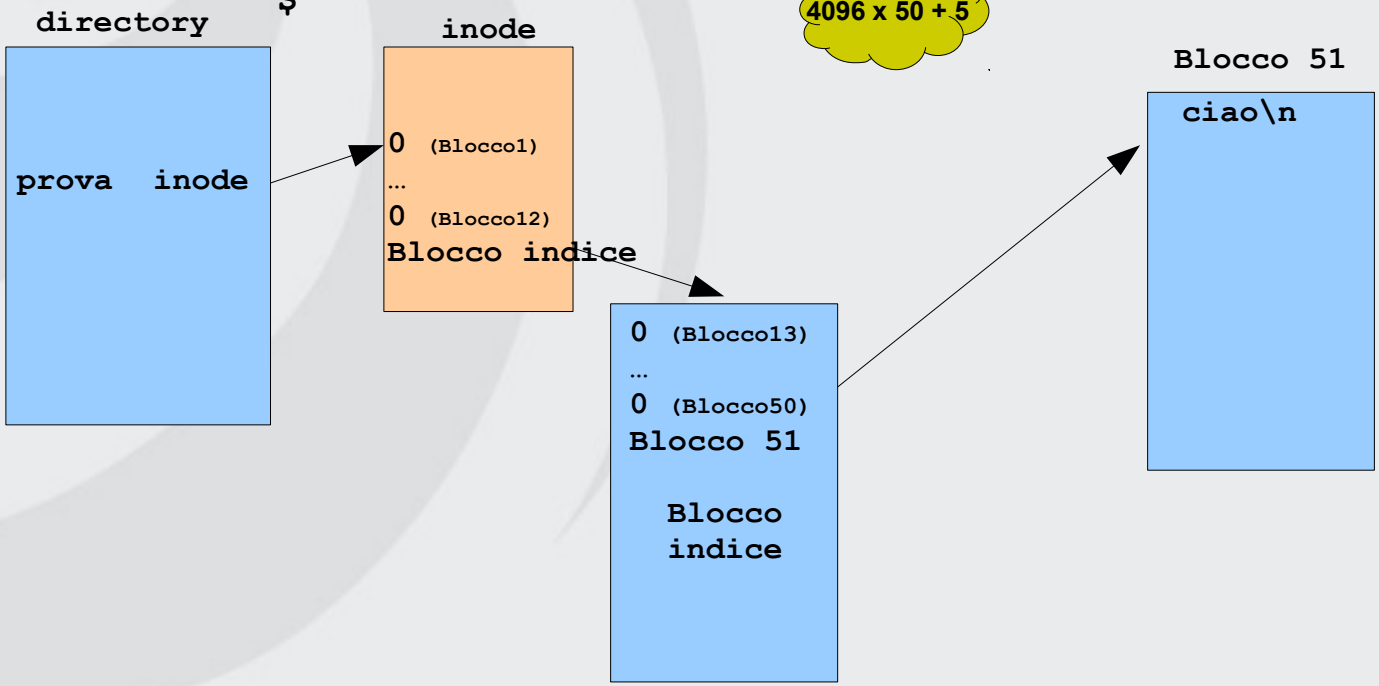


dd

```
$ echo ciao | dd bs=4096 seek=50 of=prova
0+1 records in
0+1 records out
5 bytes (5 B) copied, 0.00161027 s, 3.1 kB/s
$ ls -l prova
-rw-r--r-- 1 user user 204805 Dec 19 18:33 prova
$ du -h prova
8.0K    prova
$
```

2 blocchi

4096 x 50 + 5

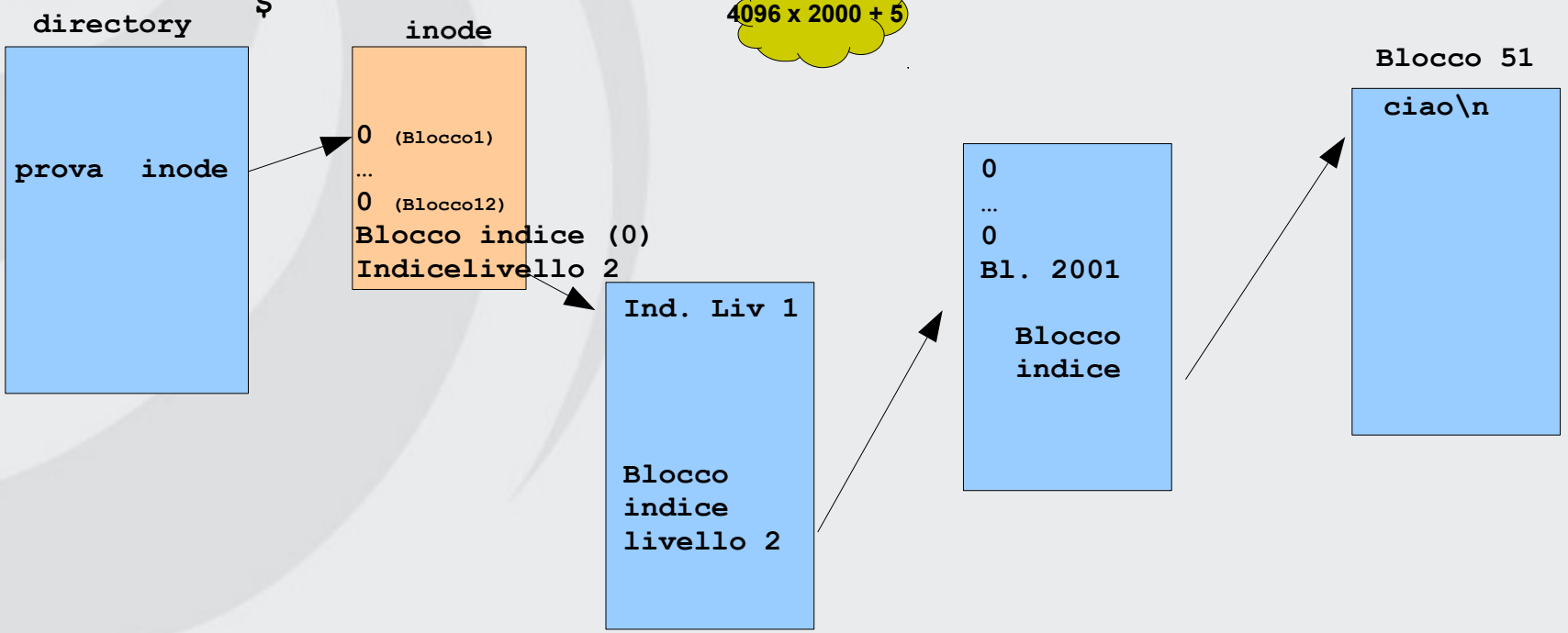


dd

```
$ echo ciao | dd bs=4096 seek=2000 of=prova
0+1 records in
0+1 records out
5 bytes (5 B) copied, 0.00161027 s, 3.1 kB/s
$ ls -l prova
-rw-r--r-- 1 user user 81920005 Dec 19 18:33 prova
$ du -h prova
12.0K    prova
$
```

3 blocchi

4096 x 2000 + 5



Allocazione di file

- Si consideri un file costituito da 100 blocchi. Si assuma che il blocco di controllo del file (e il blocco dell'indice, in caso di allocazione indicizzata) sia già in memoria (e che non lo si aggiorni immediatamente). Si calcolino quante operazioni di I/0 del disco sono necessarie con le strategie di allocazione contigua, concatenata e indicizzata (singolo livello). Nel caso di allocazione contigua, si assuma che non ci sia spazio di crescita all'inizio, ma solo alla fine.

1) Il blocco viene aggiunto all'inizio.
2) Il blocco viene aggiunto al centro.
3) Il blocco viene aggiunto alla fine.
4) Il blocco viene rimosso dall'inizio.
5) Il blocco viene rimosso dal centro.
6) Il blocco viene rimosso dalla fine.

Allocazione contigua

1) Il blocco viene aggiunto all'inizio.
   - devo copiare tutti i blocchi e scrivere quello nuovo (201 operazioni)
2) Il blocco viene aggiunto al centro.
   - devo copiare gli ultimi 50 blocchi (100),
   - scrivere quello nuovo (1 – totale 101)
3) Il blocco viene aggiunto alla fine.
   - devo solo scrivere quello nuovo (1)
4) Il blocco viene rimosso dall'inizio.
   - devo copiare 99 blocchi (198)

Posso evitarlo al prezzo di avere un singolo blocco isolato

5) Il blocco viene rimosso dal centro.
   - devo copiare gli ultimi 49 blocchi (98)
6) Il blocco viene rimosso dalla fine.
   - costo 0

Allocazione concatenata

1) Il blocco viene aggiunto all'inizio.
   - devo scrivere solo il nuovo blocco
2) Il blocco viene aggiunto al centro.
   - devo leggere i primi 50 blocchi,
   - riscrivere il 50° (ho modificato l'indirizzo),
   - scrivere quello nuovo (totale 52)
3) Il blocco viene aggiunto alla fine.
   - devo leggere tutti i blocchi,
   - riscrivere l'ultimo (ho modificato l'indirizzo),
   - scrivere quello nuovo (totale 102)

Se viene gestito anche un puntatore all'ultimo blocco vi posso accedere direttamente

4) Il blocco viene rimosso dall'inizio.
   - devo leggere il primo blocco
5) Il blocco viene rimosso dal centro.
   - devo leggere 51 blocchi
   - riscrivere il 50°
6) Il blocco viene rimosso dalla fine.
   - devo leggere 99 blocchi
   - riscrivere il 99°

Allocazione indicizzata

1) Il blocco viene aggiunto all'inizio.
   - devo scrivere solo il nuovo blocco
2) Il blocco viene aggiunto al centro.
   - devo scrivere solo il nuovo blocco
3) Il blocco viene aggiunto alla fine.
   - devo scrivere solo il nuovo blocco
4) Il blocco viene rimosso dall'inizio.
   - nessuna operazione
5) Il blocco viene rimosso dal centro.
   - nessuna operazione
6) Il blocco viene rimosso dalla fine.
   - nessuna operazione

Allocazione di file

- NB: in ogni caso non sono stati considerati i blocchi relativi a directory e indici
  - è stato sempre modificato almeno un blocco già presente in memoria (ad esempio è cambiata la dimensione del file)