

# Sistemi Operativi File System (parte 1)

Docente: Claudio E. Palazzi  
[cpalazzi@math.unipd.it](mailto:cpalazzi@math.unipd.it)

Crediti per queste slides al Prof. Tullio Vardanega

# Considerazioni generali – 1

- La maggior parte dell'informazione applicativa (i dati) ha **durata**, **ambito** e **dimensione** più ampi della vita delle applicazioni che la usano
  - Queste sono le 3 esigenze principali
    - Persistenza dei dati
    - Possibilità di condivisione dei dati tra applicazioni distinte
    - Nessun limite di dimensione fissato a priori
- Il *file system* è il servizio di S/O progettato per soddisfare questi bisogni

# Considerazioni generali – 2

- Il termine *file* designa un insieme di dati correlati, residenti in **memoria secondaria** e trattati unitariamente
  - *File* = raccoglitore, *dossier*
- Il termine *file system* (FS) designa la parte di S/O che si occupa di
  - Organizzazione
  - Gestione
  - Realizzazione
  - Accesso
- Di e ai *file*

# Aspetti generali – 3

- La progettazione di FS affronta 2 problemi chiave
  - **Cosa** occorre offrire all'utente applicativo e secondo quali forme concrete
    - Modalità di **accesso** a *file*
    - Struttura **logica** e **fisica** di *file*
    - Operazioni ammissibili su *file*
  - **Come** ciò possa essere realizzato in modo pratico ed economico
    - Garantendo la massima indipendenza dall'architettura fisica di supporto

# Il *file*

- Il *file* è un concetto logico realizzato tramite **meccanismi di astrazione**
  - Per salvare informazione su memoria secondaria e potendola ritrovare in seguito senza conoscerne né la struttura logica e fisica né il funzionamento
  - All'utente non interessa come ciò avviene
  - Interessa invece poter designare le proprie unità di informazione mediante **nomi logici** unici e distinti
  - L'utente vede e tratta solo nomi di *file*
  - Le caratteristiche distintive di un *file* sono
    - **Attributi** (tra cui il nome utente)
    - **Struttura interna**
    - **Operazioni ammesse**

# Attributi di *file* – 1

- **Nome**

- Stringa composta da 8 – 255 caratteri, inclusi numeri e caratteri speciali
- Con  $\geq 1$  estensioni che possono designare il “tipo” di *file* come visto dall'utente
  - **MS-DOS** (base di Windows 95 e Windows 98)
    - Nomi da 1 – 8 caratteri, con  $\leq 1$  estensione da 1 – 3 caratteri, **designante**, senza distinzione tra maiuscolo e minuscolo (*case insensitive*)
  - **UNIX** (base di GNU/Linux)
    - Nomi fino a 14 (ora 255) caratteri, *case sensitive*, con estensioni, solo **informative**, senza limite di numero e di ampiezza
- In generale, l'utente può configurare presso il S/O l'associazione tra l'**ultima** estensione del *file* ed il tipo applicativo corrispondente

# Attributi di *file* – 2

- **Dimensione corrente**
- **Data di creazione**
  - Può non essere mostrata
- **Data di ultima modifica**
  - Indica la “freschezza” del contenuto
- **Creatore e possessore**
  - Possono essere distinti
    - P.es.: il compilatore crea *file* di proprietà dell'utente
- **Permessi di accesso**
  - Lettura, scrittura, esecuzione

# Attributi di *file* – 3

**Protezione**

Permesso di accesso al *file*

*Flag*

***Password***

Chiave di accesso al *file*

**Creatore**

Identità del processo che ha creato il *file*

**Proprietario**

Identità del processo utilizzatore del *file*

**Uso**

0 – lettura/scrittura      1 – sola lettura (*read-only*)

**Visibilità**

0 – normale      1 – *file* non visibile (*hidden*)

**Livello**

0 – normale      1 – *file* di sistema

**Archiviazione**

0 – salvato (*backed up*)      1 – non salvato

**Tipo di contenuto**

0 – ASCII      1 – binario

**Tipo di accesso**

0 – sequenziale      1 – casuale (*random*)

**Permanenza**

0 – normale      1 – da eliminare dopo l'uso (*temporary*)

**Accesso esclusivo**

0 – libero       $\neq 0$  – bloccato (*locked*)

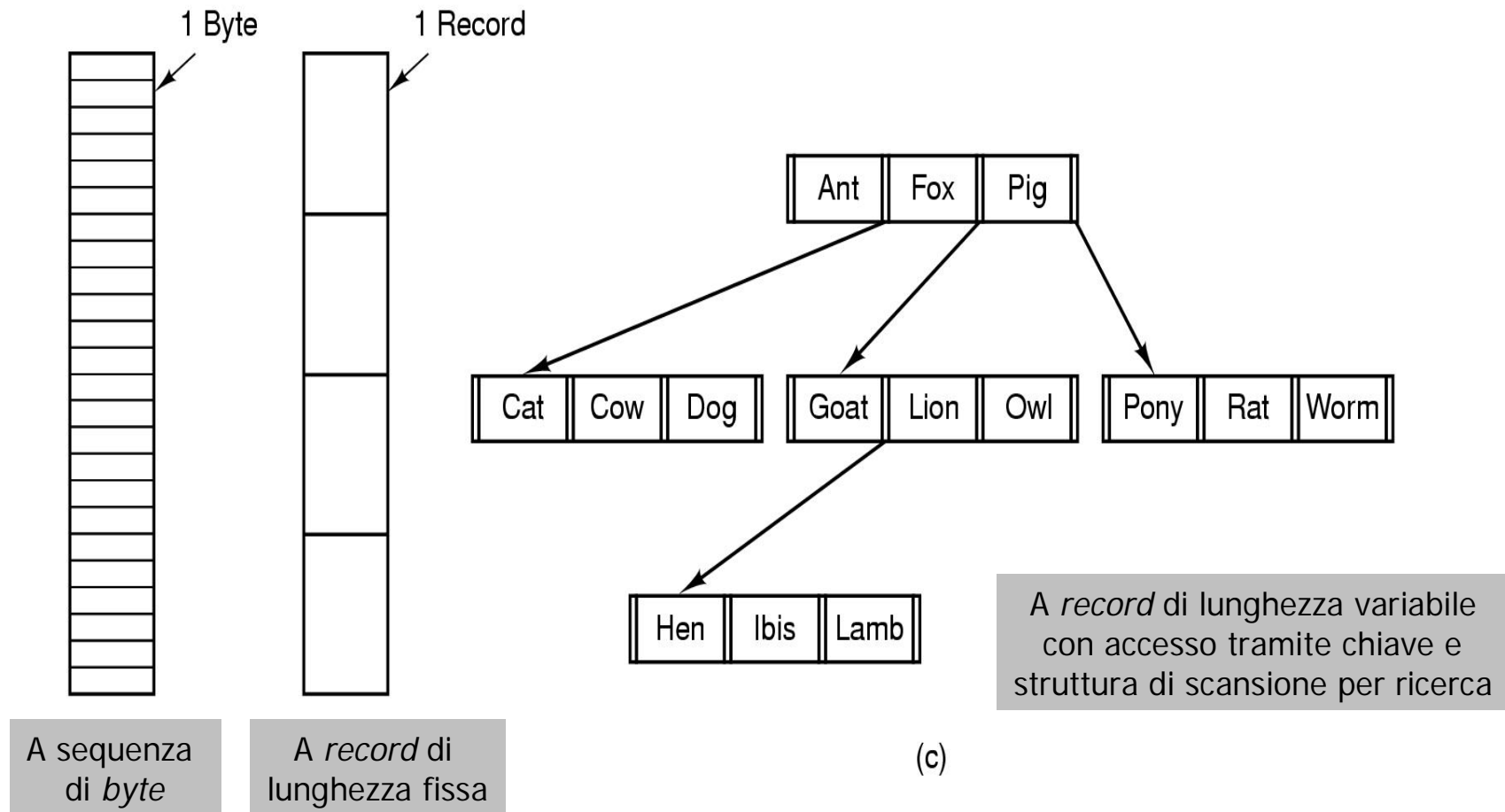
Vedi Fig. 4-4 del libro di testo



# Strutture dati di *file* – 1

- La struttura dei dati all'interno di un *file* può essere considerata da 3 punti di vista distinti
  - Livello **utente**
    - Il programma applicativo associa **autonomamente** significato al contenuto grezzo del *file*
  - Livello di **struttura logica**
    - A monte dell'interpretazione dell'utente il S/O organizza i dati grezzi in strutture logiche per facilitarne il trattamento
  - Livello di **struttura fisica**
    - Il S/O mappa le strutture logiche sulle strutture fisiche della memoria secondaria disponibile (p.es.: settori o blocchi su disco)
- Le possibili strutture **logiche** di un *file* sono
  - A sequenza di *byte*
  - A *record* di lunghezza e struttura interna fissa
  - A *record* di lunghezza e struttura interna variabile

# Struttura logica di *file* – 1



# Struttura logica di *file* – 2

- **Sequenza di *byte* (*byte stream*)**
  - La strutturazione logica più rudimentale e flessibile
    - La scelta di UNIX (→ GNU/Linux) e MS Windows
  - Il programma applicativo sa come dare significato al contenuto informativo del *file*
    - Minimo sforzo per il S/O
  - L'accesso ai dati utilizza un puntatore relativo all'inizio del *file*
  - Lettura e scrittura operano a blocchi di *byte*

# Struttura logica di *file* – 3

- ***Record* di lunghezza e struttura fissa**
  - Gli spazi non utilizzati sono riempiti da caratteri speciali (p.es.: `NULL` o `SPACE`)
  - Il S/O **deve** conoscere la struttura interna del *file*
    - Per muoversi al suo interno
  - L'accesso ai dati è sequenziale e utilizza un puntatore al *record* corrente
  - Lettura e scrittura operano su *record* singoli
  - Scelta ormai obsoleta e legata a specifiche limitazioni dell'architettura di sistema

# Struttura logica di *file* – 4

- ***Record* di lunghezza e struttura variabile**
  - La struttura interna di ogni *record* viene descritta e identificata univocamente da una chiave (*key*) posta in posizione fissa e nota entro il *record*
  - Le chiavi vengono raccolte in una tabella a se stante, ordinata per chiave, contenente anche i puntatori all'inizio di ciascun *record*
  - L'accesso ai dati avviene per chiave
  - Uso abbastanza diffuso in sistemi *mainframe*

# Modalità di accesso – 1

- **Accesso sequenziale**
  - Viene trattato un gruppo di *byte* (oppure un *record*) alla volta
  - Un puntatore indirizza il *record* (o gruppo) corrente e avanza a ogni lettura o scrittura
    - La lettura può avvenire in qualunque posizione del *file*, la quale però deve essere raggiunta sequenzialmente
      - Come su un nastro
    - La scrittura può avvenire solo in coda al *file* (*Append*)
      - Ovviamente!
  - Sul *file* si può operare solo sequenzialmente
    - Ogni nuova operazione fa ripartire il puntatore dall'inizio

# Modalità di accesso – 2

- **Accesso diretto**

- Opera su *record* di dati posti in posizione **arbitraria** nel *file*
  - Posizione determinata rispetto alla base (*offset* = 0)

- **Accesso indicizzato**

- Per ogni *file* una tabella di chiavi ordinate contenenti gli *offset* dei rispettivi *record* nel *file*
  - Informazione di navigazione non più nei *record* ma in una struttura a parte ad accesso veloce (*hash*)
    - Principio delle base di dati
- Ricerca binaria della chiave e poi accesso diretto
- Denominato **ISAM** (*indexed sequential access method*) da IBM
  - Consente accesso sia indicizzato che sequenziale

# Classificazione

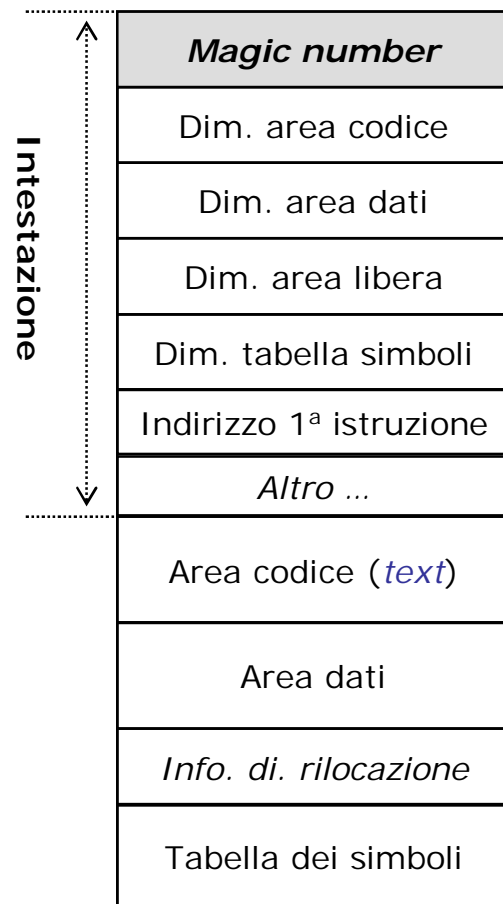
UNIX → GNU/Linux

Windows

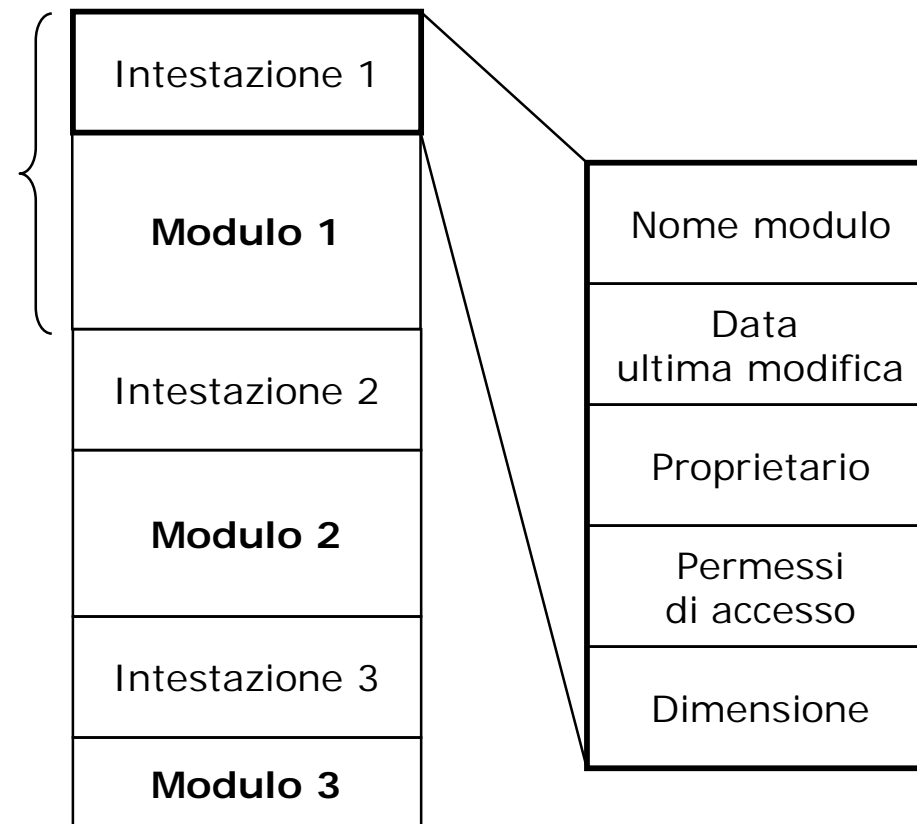
- Il FS può trattare diversi tipi di *file*
  - Classificazione distinta da quella dell'utente!
  - **File regolari** (*regular*)
    - Sui quali l'utente può operare normalmente
      - Contenuto ASCII (testo) o binario (eseguibile)
  - **File catalogo** (*directory*)
    - Tramite i quali il FS permette di descrivere l'organizzazione di gruppi di *file*
  - **File speciali**
    - Con i quali il FS rappresenta logicamente dispositivi orientati a carattere (p.es.: terminale) o a blocco (p.es.: disco)



# File binari in UNIX e GNU/Linux



Struttura di un *file* eseguibile



Struttura di un *file* archivio  
(**tar** : *tape archive*)

# Operazioni ammesse – 1

- **Creazione**
  - *File* inizialmente vuoto; inizializzazione attributi
- **Apertura**
  - Deve precedere il 1° uso; predispone le informazioni utili all'accesso
- **Cerca posizione (*seek*)**
  - Solo per accesso casuale
- **Cambia nome**
  - *Rename* (può implicare spostamento nella struttura logica del FS)
- **Distruzione**
  - Rilascio della memoria occupata
- **Chiusura**
  - Rilascio delle strutture di controllo usate per l'accesso ed il salvataggio dei dati
- **Lettura. Scrittura**
  - *Read, write, append*
- **Trova attributi** (per *make*)
- **Modifica attributi** (permessi)

Azioni più complesse (p.es.: copia) si ottengono tramite combinazione delle operazioni di base

# Operazioni ammesse – 2

- **Sessione d'uso di un *file***
  - Si può accedere in uso solo a un *file* già aperto
  - All'apertura del *file* il S/O ne predispone uno specifico strumento di accesso (*handle*)
  - Dopo l'uso il *file* dovrà essere chiuso
  - UNIX (→ GNU/Linux) mantiene una tabella dei *file* aperti a due livelli
    - **Livello I:** informazioni sul *file* comuni a “famiglie” di processi
    - **Livello II:** dati specifici del particolare processo

# Esempio d'uso con “chiamate di sistema”

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc,
        char *argv[]){

    FILE *fp;
    char dato;

    if (argc != 2) {
        printf("Nome del file?");
        exit(1);
    }

    // continua ...
```

1/2

```
    if ((fp = fopen(argv[1], "w"))
        == NULL){
        printf("File non aperto.\n");
        exit(1);
    }
    do {
        dato = getchar();
        if (EOF == putc(dato, fp)) {
            printf("Errore di lettura.\n");
            break;
        };
    } while (dato != 'c');
    fclose(fp);
}
```

2/2

# File mappati in memoria

- Il S/O può mappare un *file* in memoria virtuale
  - Il *file* continua a risiedere in memoria secondaria
  - All'indirizzo di ogni suo dato corrisponde un indirizzo di memoria virtuale (base + *offset*)
    - Con segmentazione si **potrebbe** avere {*file* = segmento} potendo così usare lo stesso *offset* per entrambi
  - Le operazioni su *file* avvengono in memoria principale
    - Chiamata di indirizzo → *page fault* → caricamento → operazione → rimpiazzo di pagina → salvataggio in memoria secondaria
  - A fine sessione **tutte** le modifiche effettuate in memoria primaria devono essere riportate in memoria secondaria
- Riduce gli accessi a disco ma comporta problemi con la condivisione e con i *file* di enorme dimensione
  - Dove trovare la versione corrente dei dati: RAM o disco?

# Struttura di *directory* – 1

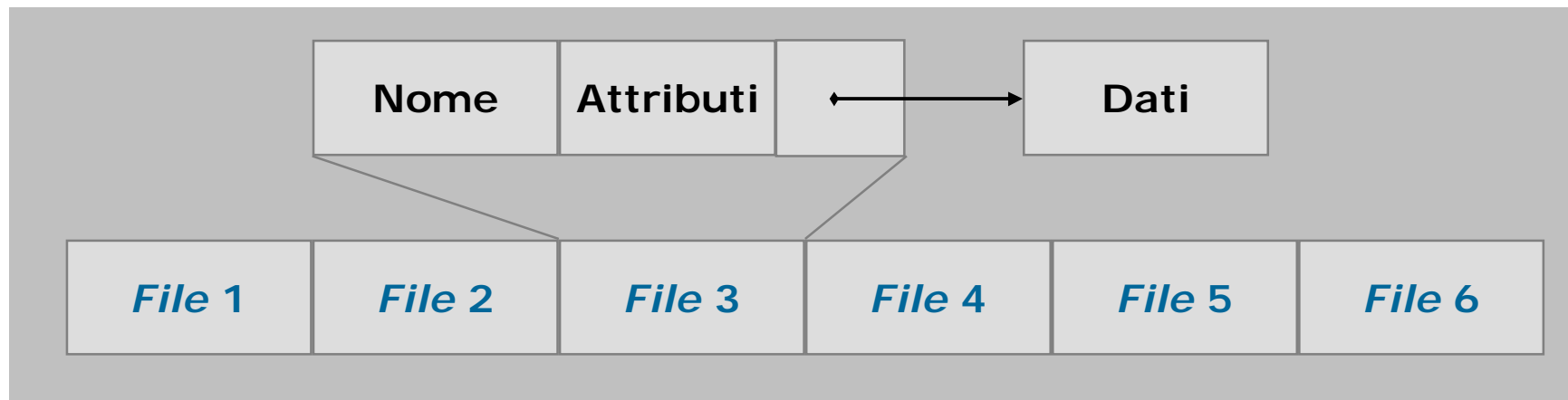
- Ogni FS usa *directory* (catalogo) o *folder* (cartella) per tener traccia dei suoi *file* regolari
- Le *directory* possono essere classificate rispetto all'organizzazione di *file* che esse consentono
  - A livello singolo
  - A due livelli
  - Ad albero
  - A grafo aperto
  - A grafo generalizzato (con cicli)

# Struttura di *directory* – 2

- Requisiti fondamentali a livello utente
  - **Efficienza**
    - Realizzare un *file* deve essere semplice
    - Trovare un *file* deve essere facile e veloce
  - **Libertà di denominazione**
    - Più utenti devono poter ciascuno usare lo stesso nome per un *file* loro proprio
    - Lo stesso *file* deve poter essere “chiamato” con nomi diversi da utenti diversi
  - **Libertà di raggruppamento**
    - Creare gruppi logici di *file* sulla base di proprietà significative per l'utente

# Struttura di *directory* – 3

- **Directory a livello singolo**
  - Tutti i *file* sono elencati su un'unica lista lineare (“*root directory*” ?), ciascuno con il proprio nome
    - I nomi dei *file* devono pertanto essere **unici**
  - Semplice da capire e da realizzare
  - File facili da trovare
  - Gestione onerosa all'aumentare dei *file*

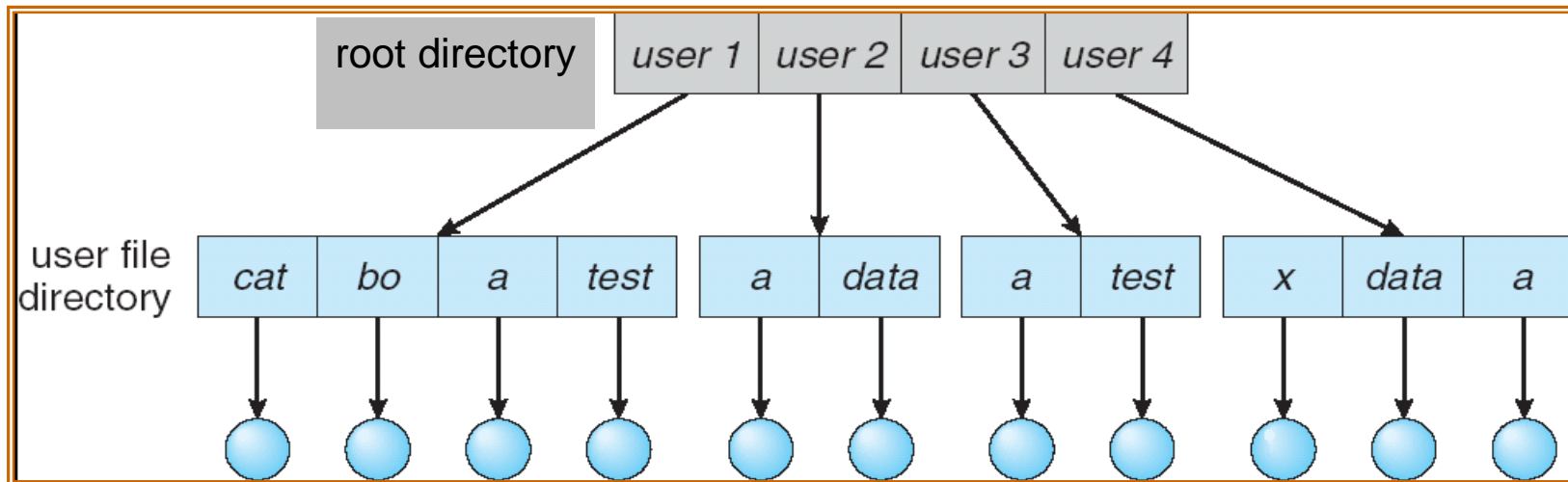




# Struttura di *directory* – 4

- ***Directory* a due livelli**
  - Una *Root Directory* contiene una *User File Directory* (UFD) per ciascun utente di sistema
  - L'utente registrato può vedere **solo** la propria UFD
    - Le UFD di altri solo se esplicitamente autorizzato
    - Buona soluzione per isolare utenti in sistemi multiprogrammati
  - I *file* sono localizzati tramite percorso (*path name*)
  - I programmi di sistema possono essere copiati su tutte le UFD, oppure (meglio) posti in una *directory* di sistema condivisa e lì localizzati mediante cammini di ricerca predefiniti (*search path*)

# Struttura di *directory* – 5



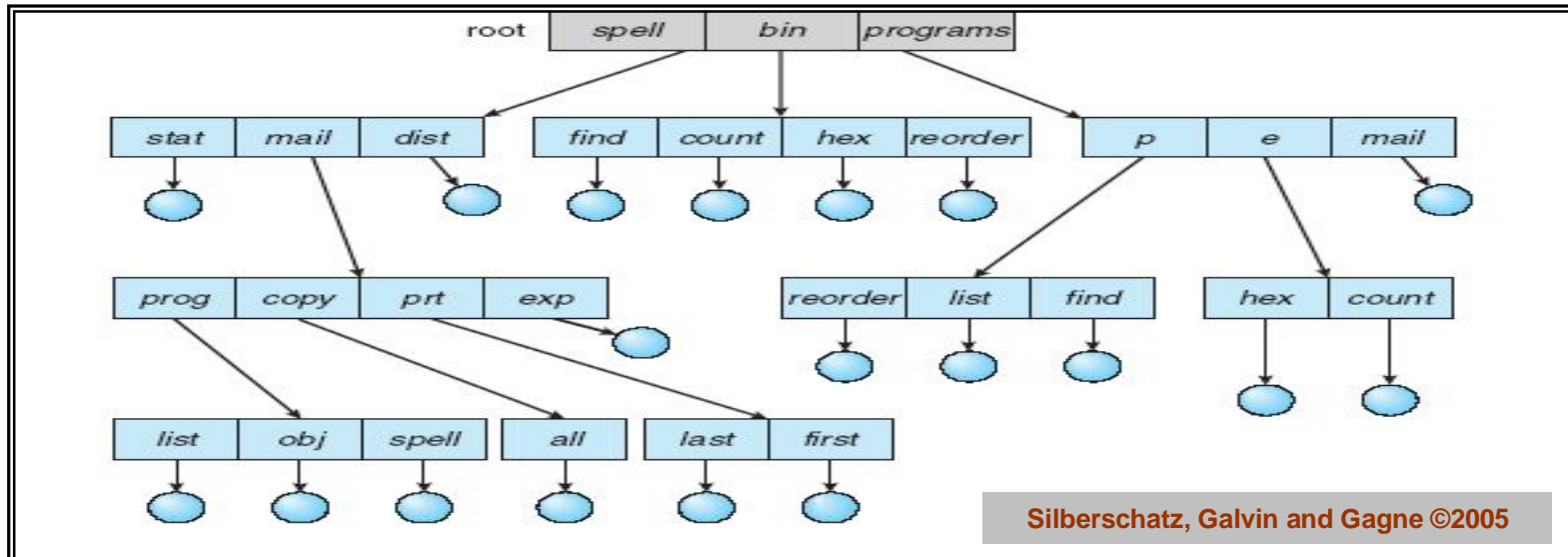
Silberschatz, Galvin and Gagne ©2005

- Requisiti parzialmente soddisfatti
  - Efficienza di ricerca
  - Libertà di denominazione
    - Ma non di riferimenti multipli allo stesso *file*
- Requisiti **non** soddisfatti
  - Libertà di raggruppamento

# Struttura di *directory* – 6

- ***Directory* ad albero**
  - Numero arbitrario di livelli
  - Il livello superiore viene detto radice (*root*)
  - Ogni *directory* può contenere *file* regolari o *directory* di livello inferiore
  - Ogni utente ha la sua *directory* corrente che può cambiare con comandi di sistema
  - Se non si specifica il cammino (*path*) si assume come riferimento la *directory* corrente
  - Il cammino può essere **assoluto**
    - Espresso rispetto alla radice del FS
  - Oppure **relativo**
    - Rispetto alla posizione corrente

# Struttura di *directory* – 7



- Requisiti **parzialmente** soddisfatti
  - Ricerca efficiente
  - Libertà di denominazione
    - Ma non di riferimenti multipli allo stesso *file*
  - Libertà di raggruppamento

# Esempio di *directory* ad albero

(/ per UNIX/GNU/Linux, \ per MS Windows)

Livello corrente:

*directory* **verdi** = . (*dot*)

Livello superiore

(*directory* padre) = ..

Livello inferiore

(*directory* figlio) = ./ (*slash*)

Il *file* **A1** identificato come

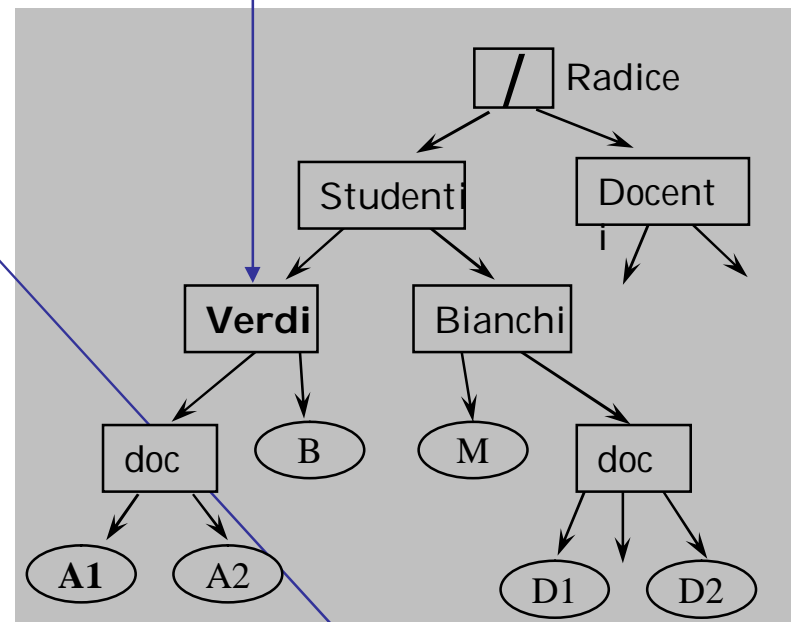
[./]**doc/A1** (cammino **relativo**)

**/studenti/Verdi/doc/A1** (cammino **assoluto**)

Il *file* **D1** di un altro ramo (purché condiviso)

../**studenti/Bianchi/doc/D1** (relativo)

**/studenti/Bianchi/doc/D1** (assoluto)

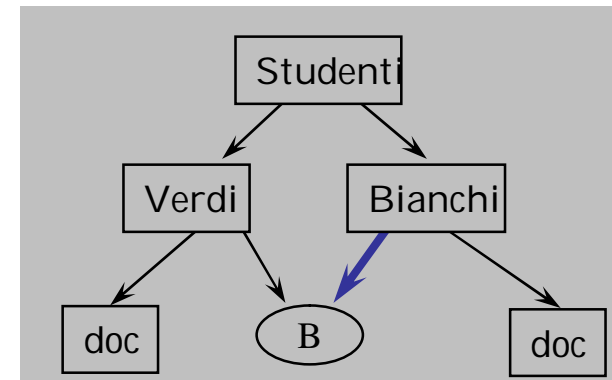


Contenuto  
iniziale di **cd**

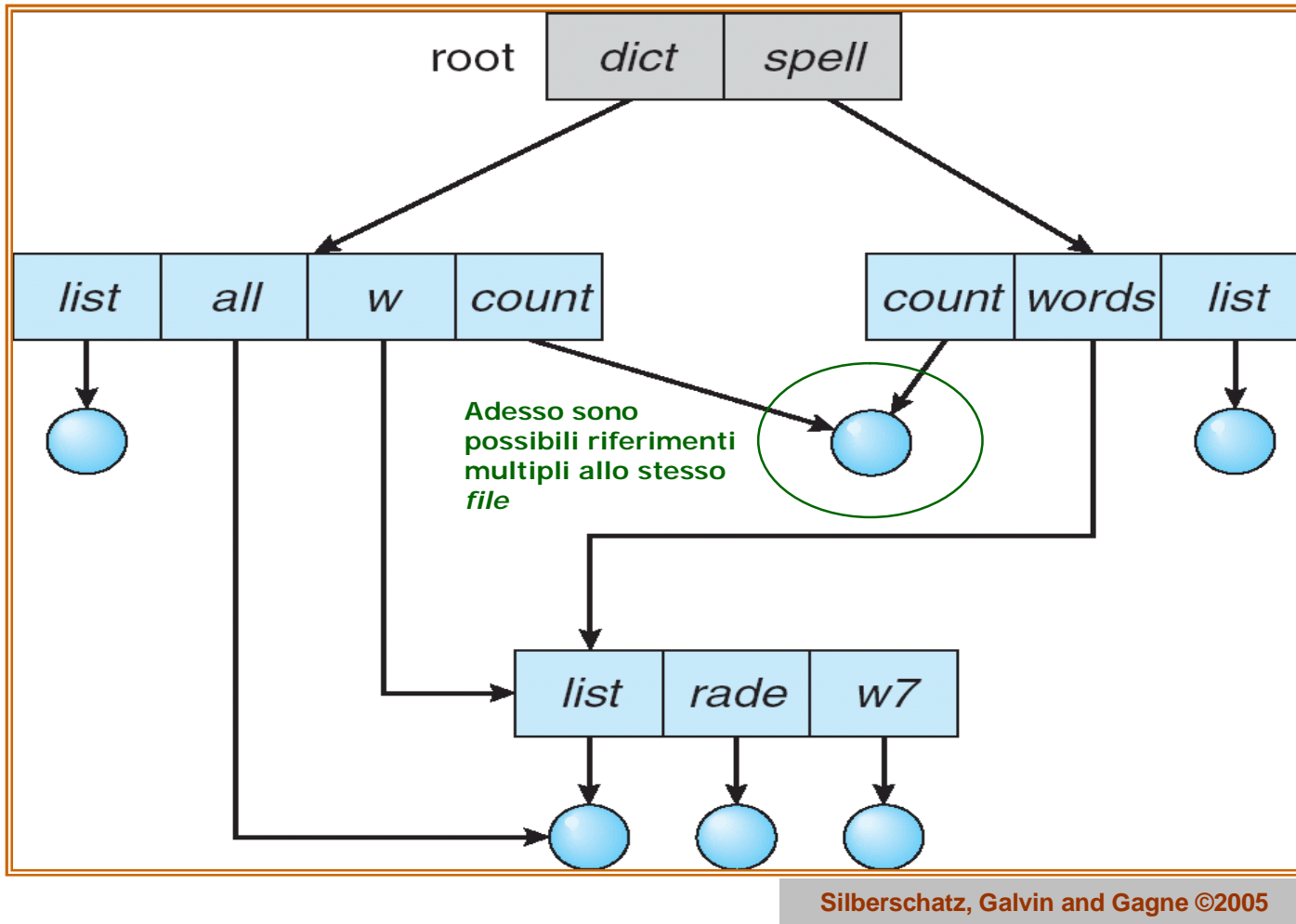
# Struttura di *directory* – 8

- **Directory a grafo**

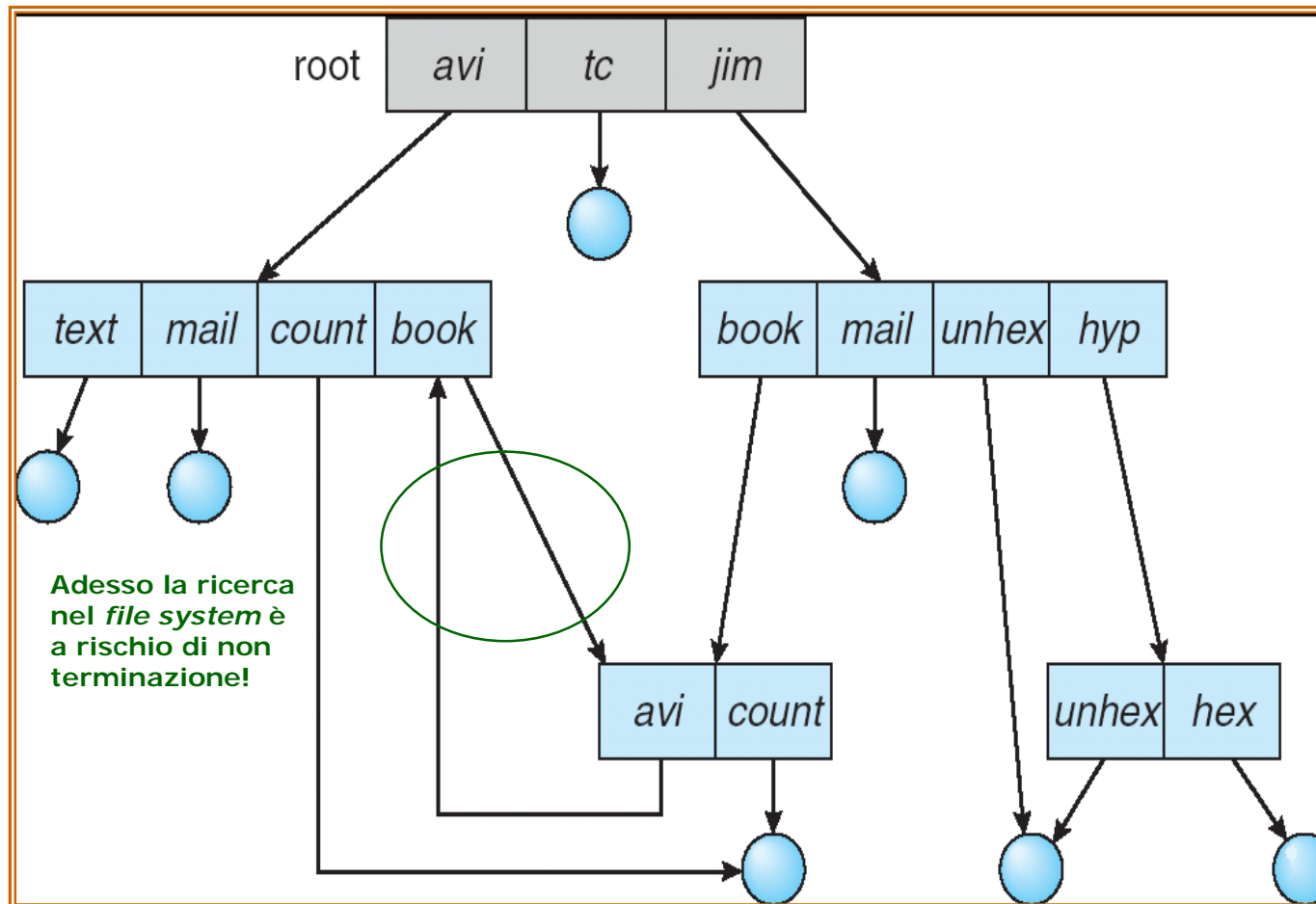
- Aciclico oppure ciclico (generalizzato)
  - L'albero diventa grafo consentendo allo stesso *file* di appartenere simultaneamente a più *directory*
  - UNIX e GNU/Linux utilizzano collegamenti simbolici (*link*) tra il nome reale del *file* e la sua presenza virtuale
  - La forma generalizzata consente collegamenti ciclici e dunque riferimenti circolari
  - Un S/O potrebbe duplicare gli identificatori di accesso al *file* (*handle*) → nomi distinti
    - Questo però rende più difficile assicurare la coerenza del *file*



# Struttura a grafo aciclico



# Struttura a grafo generalizzato



Silberschatz, Galvin and Gagne ©2005



# Struttura di *directory* – 9

- ***Hard link***
  - Un puntatore diretto a un *file* **regolare** viene inserito in una *directory* a esso remota
    - Che deve risiedere nello **stesso** FS del *file*
  - Questo crea **2 vie d'accesso distinte** dirette a uno stesso *file*
- ***Symbolic (soft) link***
  - Viene creato un *file* **speciale** il cui contenuto è il cammino del *file* originario
    - Il *file* originario può avere qualunque tipo e risiedere anche in un FS remoto
  - Questo riferimento mantiene **1 sola via d'accesso** al *file* originario

# Operazioni su *directory* GNU/Linux

Azione	Nome comando	Chiamata di sistema
Crea <i>directory</i>	<code>mkdir</code> →	Create
Cancella <i>directory</i>	<code>rmdir</code> →	Delete
Cambia nome a <i>directory</i>	<code>mv</code> →	Rename
Apri, chiudi, leggi <i>directory</i>	→	Opendir, Closedir, Readdir
Crea collegamento a <i>file</i>	<code>ln</code> →	Link
Rimuovi collegamento a <i>file</i>	<code>rm</code> →	Unlink