



# **Sistem Operasi 10**

**“File System dan Security”**

**Antonius Rachmat C, S.Kom,  
M.Cs**



# Konsep File

- File adalah kumpulan informasi yang **berhubungan** dan tersimpan dalam **secondary storage**
- Tipe:
  - Data (character, numeric, binary)
  - Program - binary
  - Direktori – logika
    - Di Linux dalam bentuk file: /home/anton (d)
  - Device – logika
    - Di Linux dalam bentuk file: /dev/sda1
- Sifat: persistence, big size, dan sharability




# File

- Simple record structure
  - Baris (lines)
    - Fixed length
    - Variable length
- Complex Structures
  - Formatted document
    - RTF, HTML
- Yang mengatur:
  - Operating system
  - System Program




# Struktur File

- Sistem operasi membutuhkan **struktur file** tertentu untuk menjalankan/ mengakses suatu file.
  - Semua sistem operasi diharuskan mampu mengenal sedikitnya **satu** jenis struktur file.
  - Jika sistem operasi mengenal semakin banyak struktur file, maka semakin **luas** aplikasi yang dapat dijalankan namun ukuran sistem operasi semakin **membengkak**.
  - Sebaliknya, jika semakin sedikit struktur file, maka sistem operasi hanya dapat menjalankan aplikasi dalam jumlah yang sedikit pula.
- 




# Atribut File

- **Name** – disimpan dalam human readable name
  - **Identifier** – unique tag (number) dalam file system
  - **Type** – dibutuhkan oleh sistem (ex: .txt)
  - **Location** – pointer to file location di harddisk
  - **Size** – current file size
  - **Protection** – controls siapa yang reading, writing, executing
  - **Time, date, and user identification** – data untuk protection, security, and usage monitoring
  - Information about files are kept in the **directory structure**, which is maintained on the disk
- 



# Operasi File

- **Create:** menciptakan file, size=0
  - **Write:** menulis file dari posisi tertentu
  - **Read:** baca file dari posisi tertentu
  - **Delete:** hapus file
  - **Truncate:** menghapus isi, mempertahankan atribut, kec file length, size=0, space released
  - **Seek :** mencari suatu data di posisi tertentu dari posisi tertentu
- 



# Open Files

- Ketika terjadi open file, data yang harus dimaintenance:
  - **File pointer**: pointer ke lokasi read/write terakhir, per process yang membuka file
  - **File-open count**: counter dari berapa kali sebuah file dibuka – untuk membuang data dari tabel open-file ketika proses terakhir menutup nya.
    - Misal: 1 jika dibuka, 0 jika ditutup
  - **Lokasi disk** tempat penyimpanan file: berisi cache dari informasi akses data.
  - **Access rights**: hak akses per proses file



# Open File Locking

- Dilakukan oleh **OS** dan **program**
- Terjadi ketika ada akses ke file
- Ada 2 kemungkinan:
  - **Mandatory** – access is denied ketika ada perintah/request
  - **Advisory** – processes dapat menemukan **status of locks** dan memutuskan yang dilakukan nya sendiri.





# File Types – Name, Extension

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information



# Access Methods

- **Sequential Access**

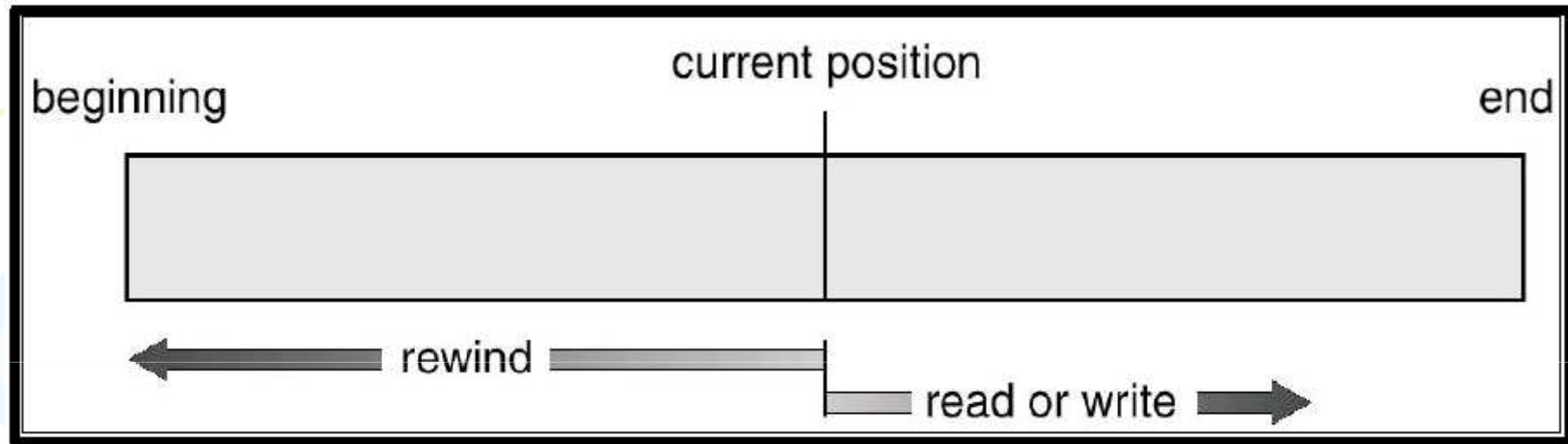
read next  
write next  
reset  
no read after last write (rewrite)

- **Direct Access**

read posisi  $n$   
write posisi  $n$   
set position to  $n$   
    read next  
    write next  
rewrite  $n$

$n$  = relative block number

# Simulation of Sequential Access on Direct-access File



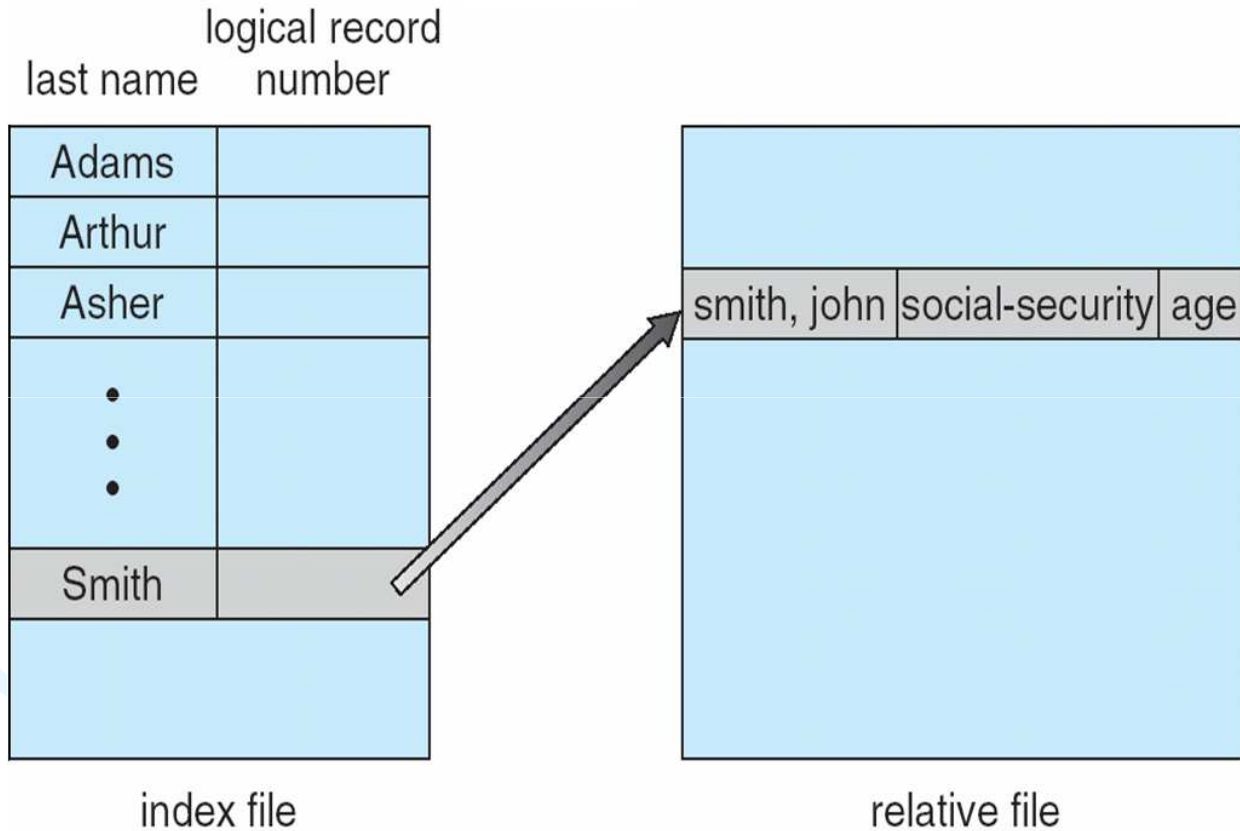
sequential access	implementation for direct access
<i>reset</i>	$cp = 0;$
<i>read next</i>	$read\ cp;$ $cp = cp + 1;$
<i>write next</i>	$write\ cp;$ $cp = cp + 1;$



# Direct Access & Index sequential

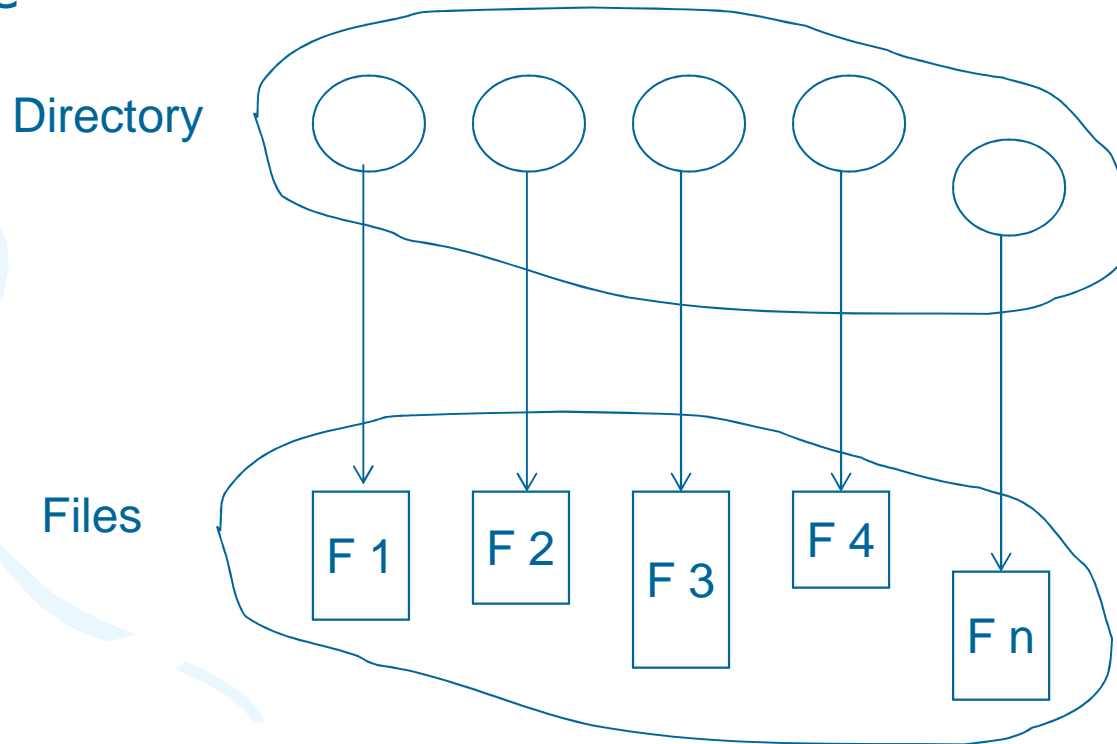
- Direct access: sangat berguna untuk pengaksesan langsung informasi dalam jumlah besar.
  - Contoh : database
- Index sequential: file juga dapat dilihat sebagai sederetan blok yang ber**indeks**
  - Untuk mencari suatu bagian dari file, pertama-tama cari **indeksnya**, kemudian dengan ***pointer*** tersebut kita mengakses file secara langsung, lalu mencari bagian dari file yang diinginkan.

# Example of Index and Relative Files



# Directory Structure

- Kumpulan node yang berisi informasi tentang semua file



directory structure dan file-file berada dalam disk



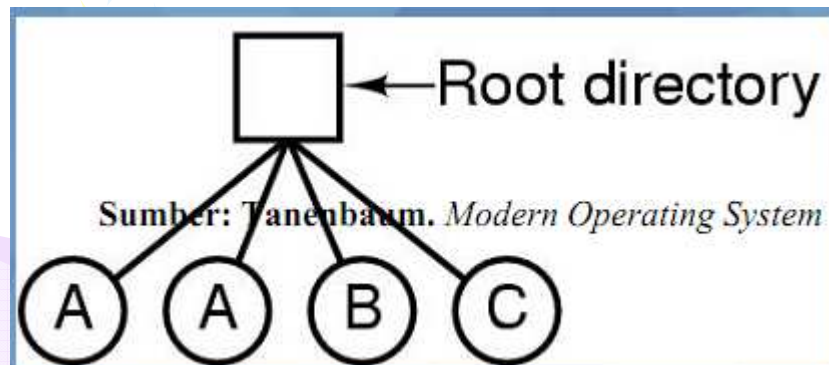
# Direktori

- **Operasi** terhadap direktori:
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system
- **Struktur** Direktori
  - Single-Level Directory
  - Two-Level Directory
  - Tree-Structured Directory
  - *Acyclic-Graph Directory*
  - *General-Graph Directory*

# Single Level Directory

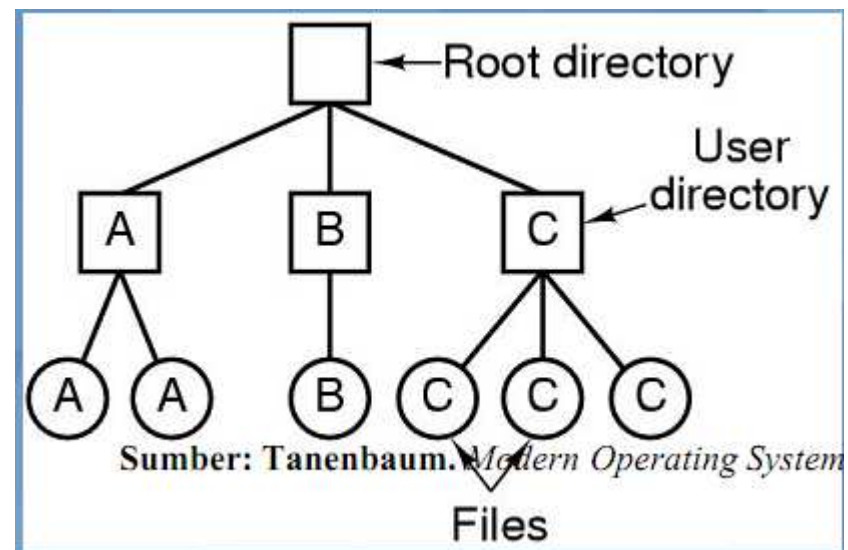
## Single Level Directory:

- Semua file terdapat dalam direktori yang sama
- Tiap file memiliki nama yang unik



## Two Level Directory:

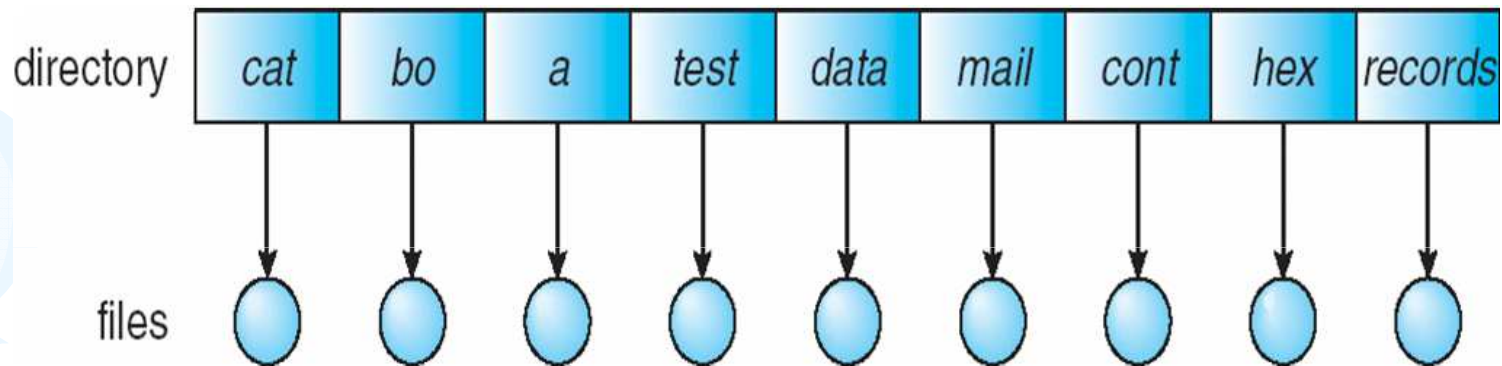
- Membuat direktori yang terpisah untuk tiap user
- Terdapat User File Directory (UFD) dan Master File Directory (MFD)
- Bila beberapa user ingin mengerjakan tugas secara bersama dan ingin mengakses file user lain





# Single-Level Directory

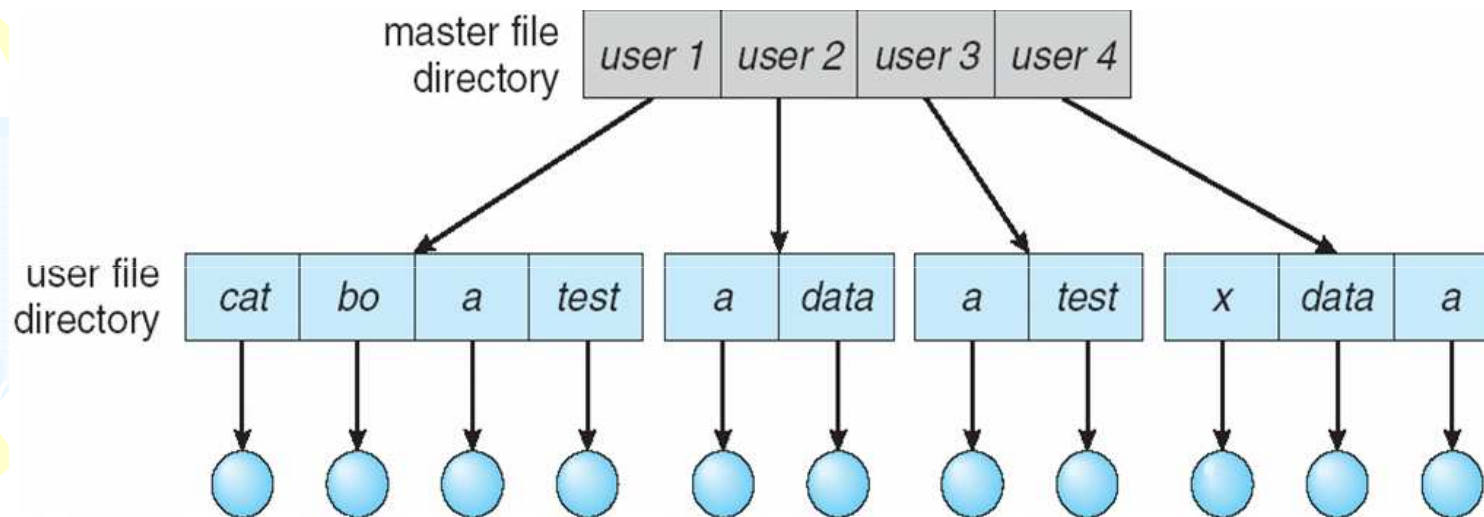
- A single directory for all users



- Naming problem
- Grouping problem

# Two-Level Directory

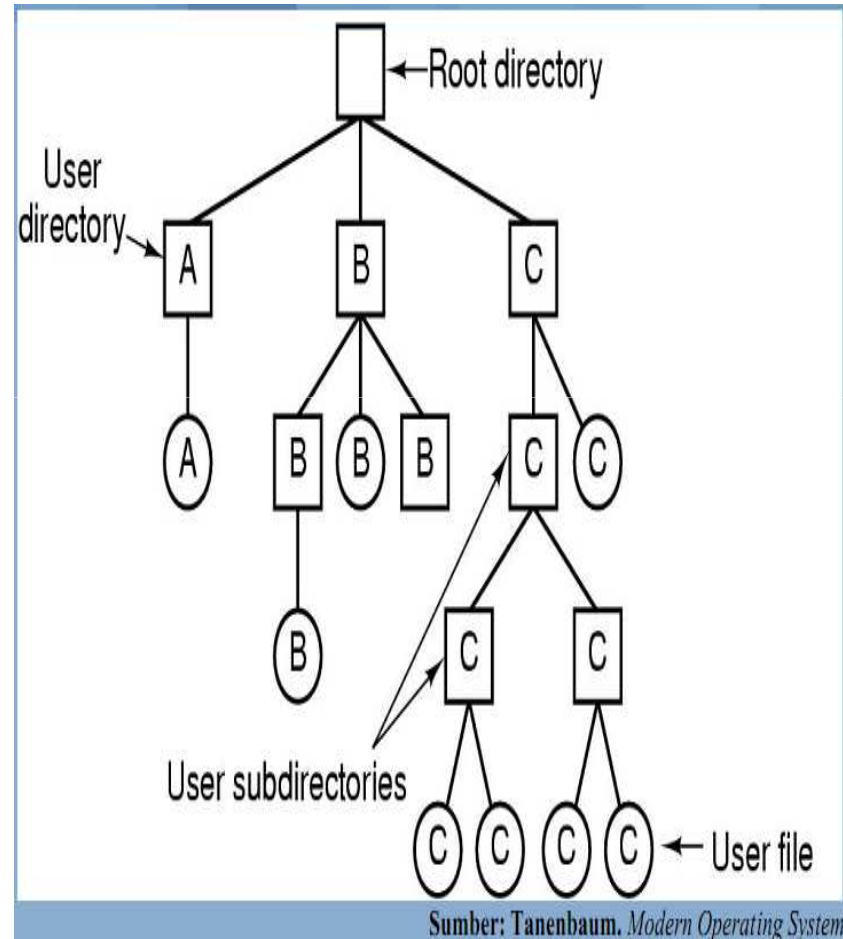
- Separate directory for each user



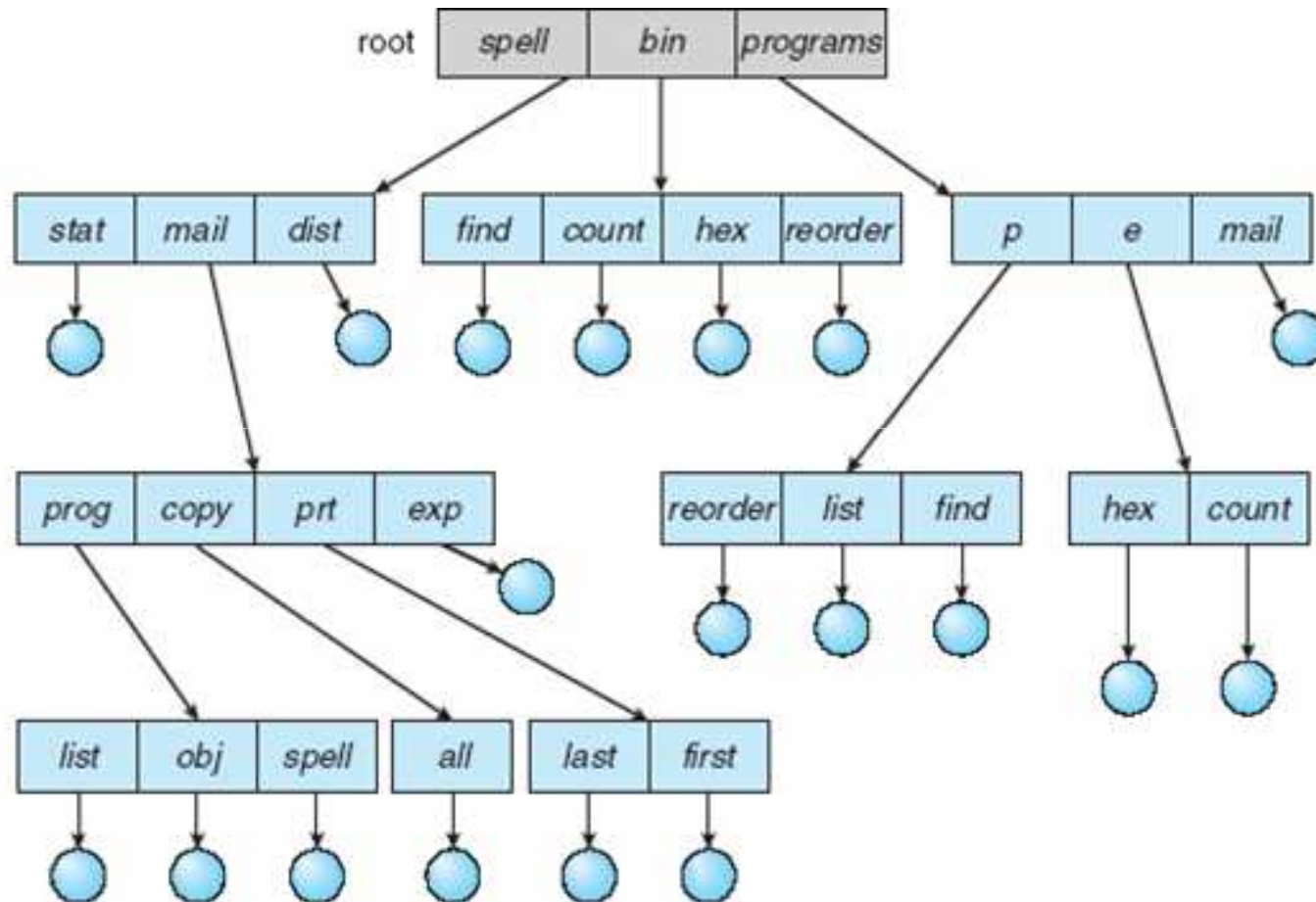
- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

# Tree structured directory

- Tiap direktori dapat mengandung file dan subdirektori
- Path (absolut path) adalah urutan direktori yang berasal dari MFD (master file directory)
- Working dir. (relative path) adalah path yang berasal dari current directory
- Current directory adalah direktori yang baru-baru ini digunakan
- Contoh absolut path : /C/C/C/C



# Tree-Structured Directories



# Tree-Structured Directories (Cont)

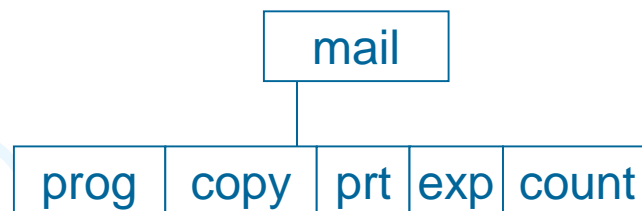
- Menciptakan sebuah file bisa dilakukan pada current directory
- Delete a file

`rm <file-name>`

- Creating a new subdirectory is done in current directory

`mkdir <dir-name>`

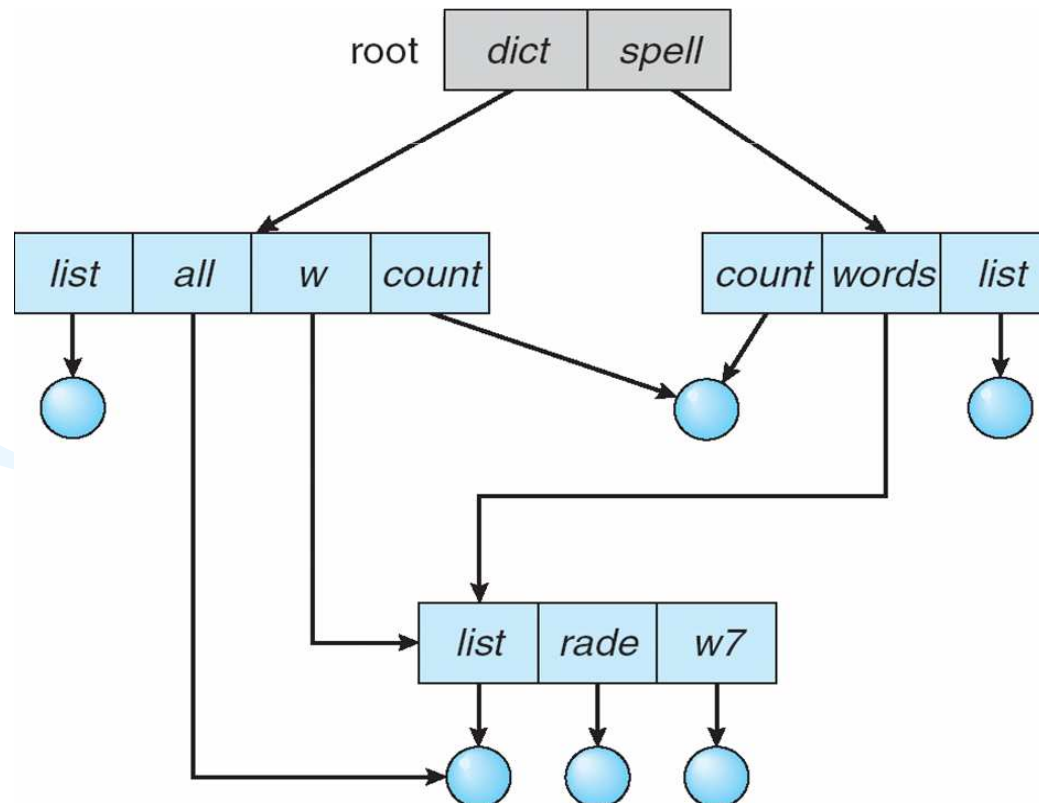
Example: if in current directory `/mail`  
`mkdir count`



Deleting "mail"  $\Rightarrow$  deleting the entire subtree rooted by "mail"

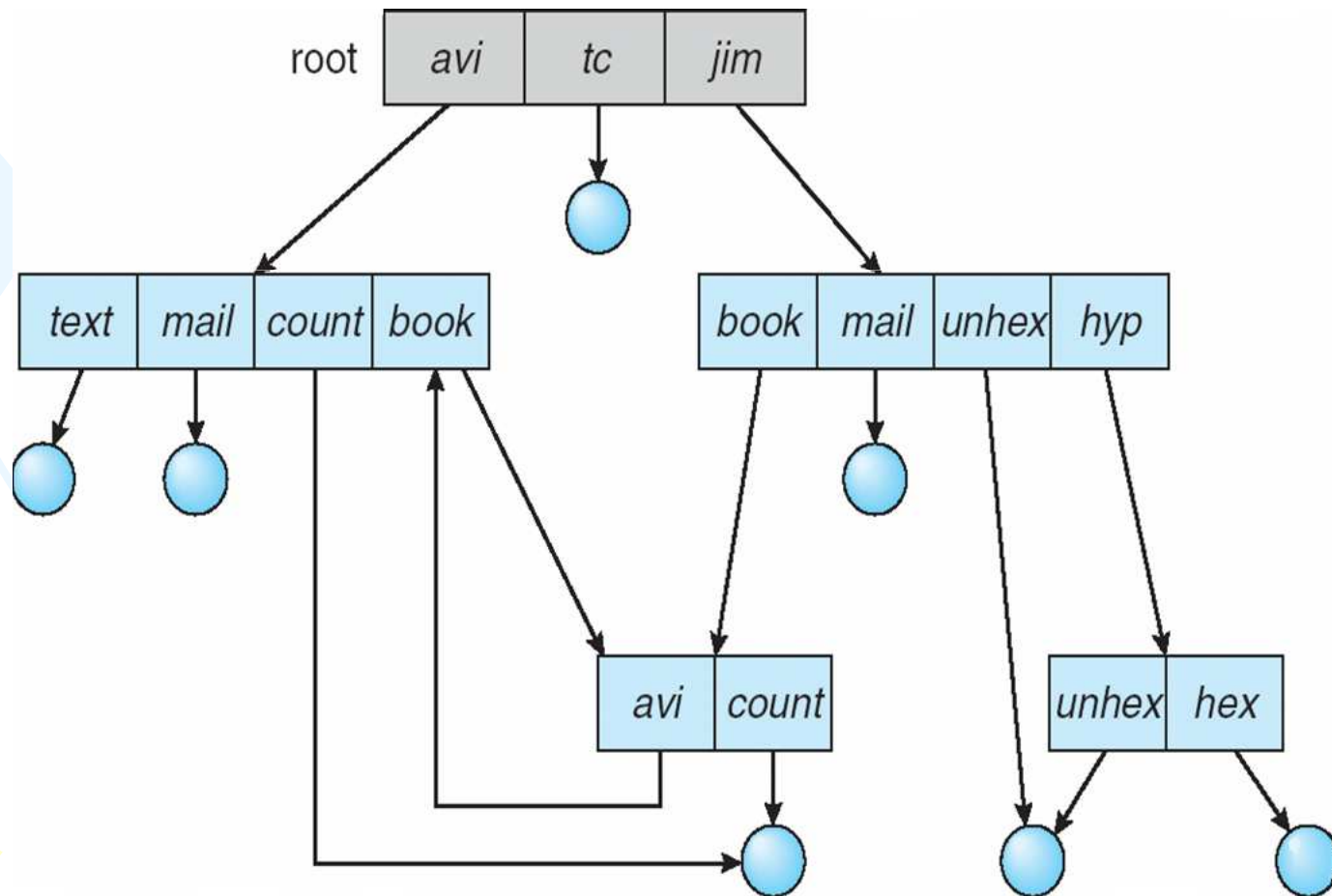
# Acyclic-Graph Directories

- Have shared subdirectories and files
  - Satu file dapat memiliki banyak *absolut path* yang berbeda
- Masalah: Penghapusan *dangling pointer*




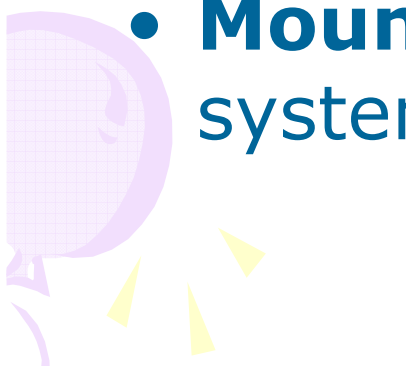
# General Graph Directory

- Berbentuk **link**.
- *Garbage collector*



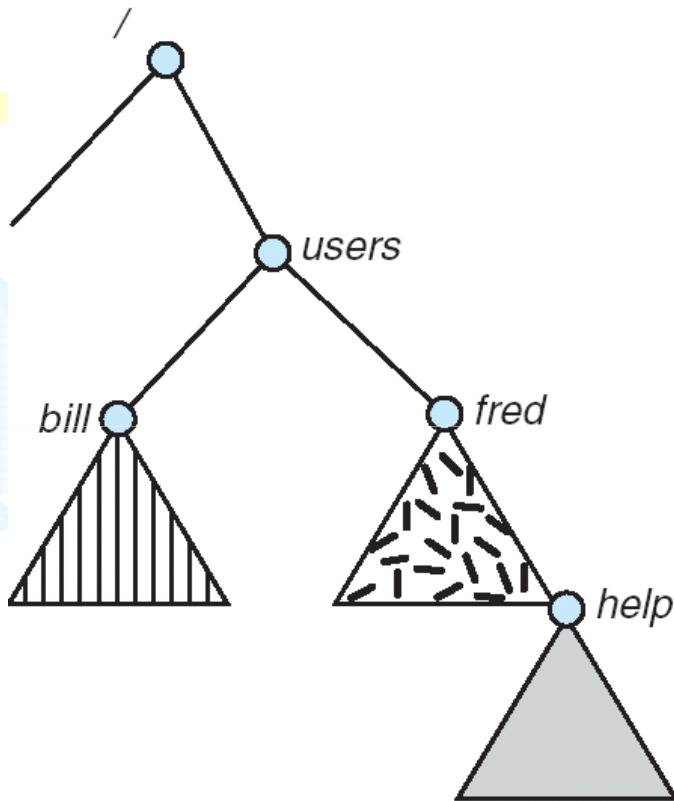


# File System Mounting

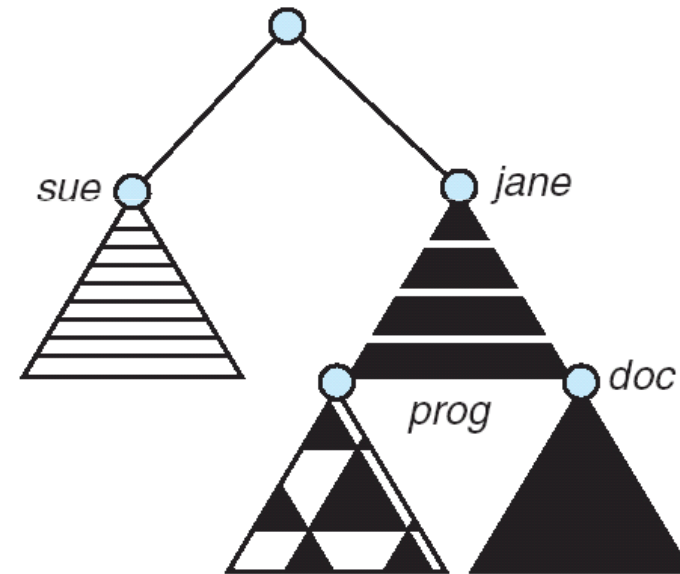
- Sebuah sistem berkas sebelum dapat digunakan harus di-**mount** terlebih dahulu.
  - **Mounting**: proses paling awal sebelum membuka sebuah direktori, yaitu dengan membuat sebuah direktori baru yang menjadi sub-tree dari tempat file system tsb diletakkan
  - **Mount point**: direktori kosong tempat file system yang akan di-mount diletakkan.
- 
- 



# (a) Existing. (b) Unmounted Partition

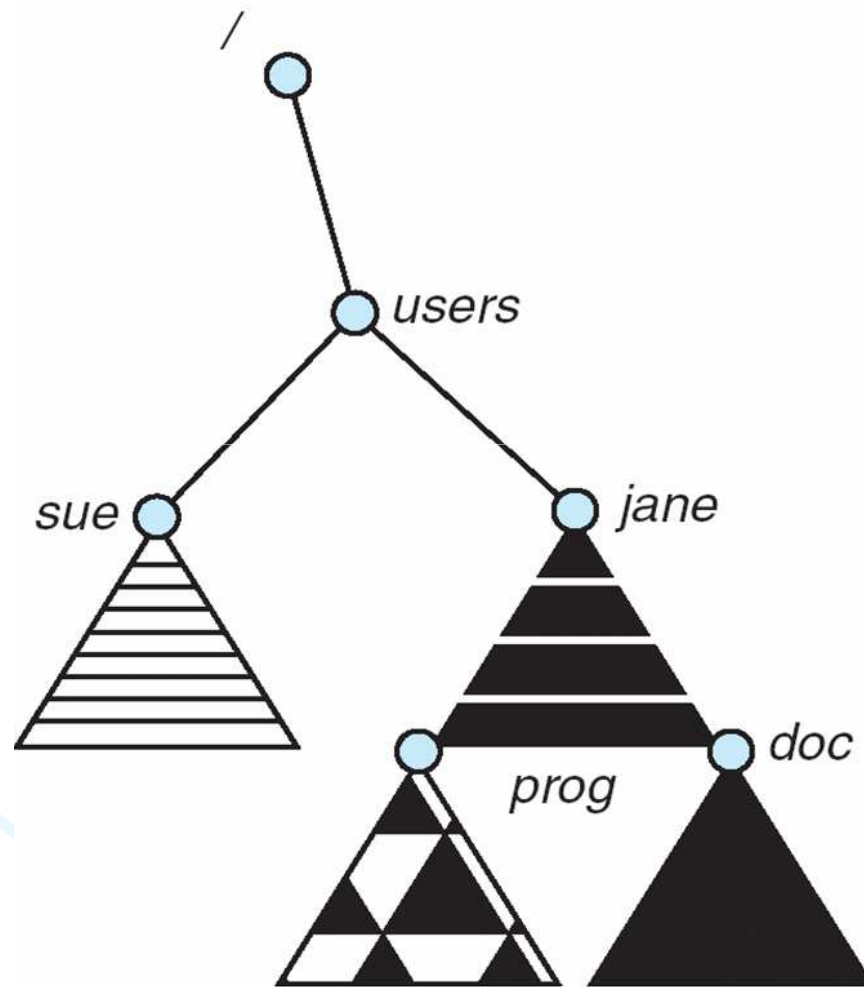


(a)



(b)

# Mount Point





# Partisi dan Mounting

- Root partition di-mount pada saat **boot time**
- Partisi yang lain di-mount secara otomatis atau manual (tergantung sistem operasi)
  - Otomatis: diletakkan di **/etc/fstab**
- Windows
  - setiap partisi yang di-mount ditandai dengan huruf dan colon dan back slash (:\\)
- UNIX
  - file system dapat di-mount di semua direktori





# File Sharing

- File sharing mendukung sebuah sistem operasi yang **user-oriented**.
- Berhubungan dengan **permission**.
- **Multiple user** bisa mengakses file yang sama.
- On distributed systems, files may be shared **across a network**
- Pada Multiple users:
  - **Owner ID**: user yang bisa mengganti atribut, membuka akses, dan mengontrol sebuah file atau direktori.
  - **Group ID**: sekelompok user yang men-share akses sebuah file.
  - **Universe**: umum
  - Tiap user memiliki user ID masing-masing yang unik.



# Protection

- File owner/creator harus dikontrol:
    - Apa yang dilakukan,
    - Oleh siapa
  - Tujuan proteksi:
    - Menjaga aman dari kerusakan fisik (*reliability*).
    - Menjaga dari akses yang tidak diijinkan (*protection*).
  - Types of access:
    - **Read**
    - **Write**
    - **Execute**
    - **Append**
    - **Delete**
    - **List**
- 
- 



# Protection

- Menggunakan ACL

- rw-rwxr-- john staff 100 Oct 20 22:12 journal

- Klasifikasi users dalam mengakses suatu file:

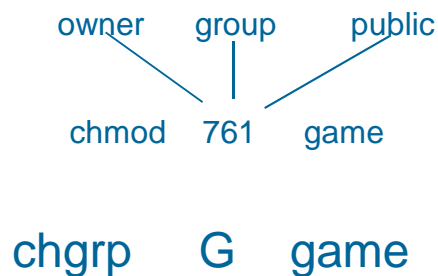
- **Owner:** User yang menciptakan file tsb.
  - **Group:** Sekelompok users yang saling berbagi file dan tergabung dalam sebuah kelompok kerja.
  - **Universe:** Semua users yang saling terhubung dalam sistem.

# Access Lists and Groups

- Mode of access: r=read, w=write, x=execute (masing2 3bit)
- Three classes of users

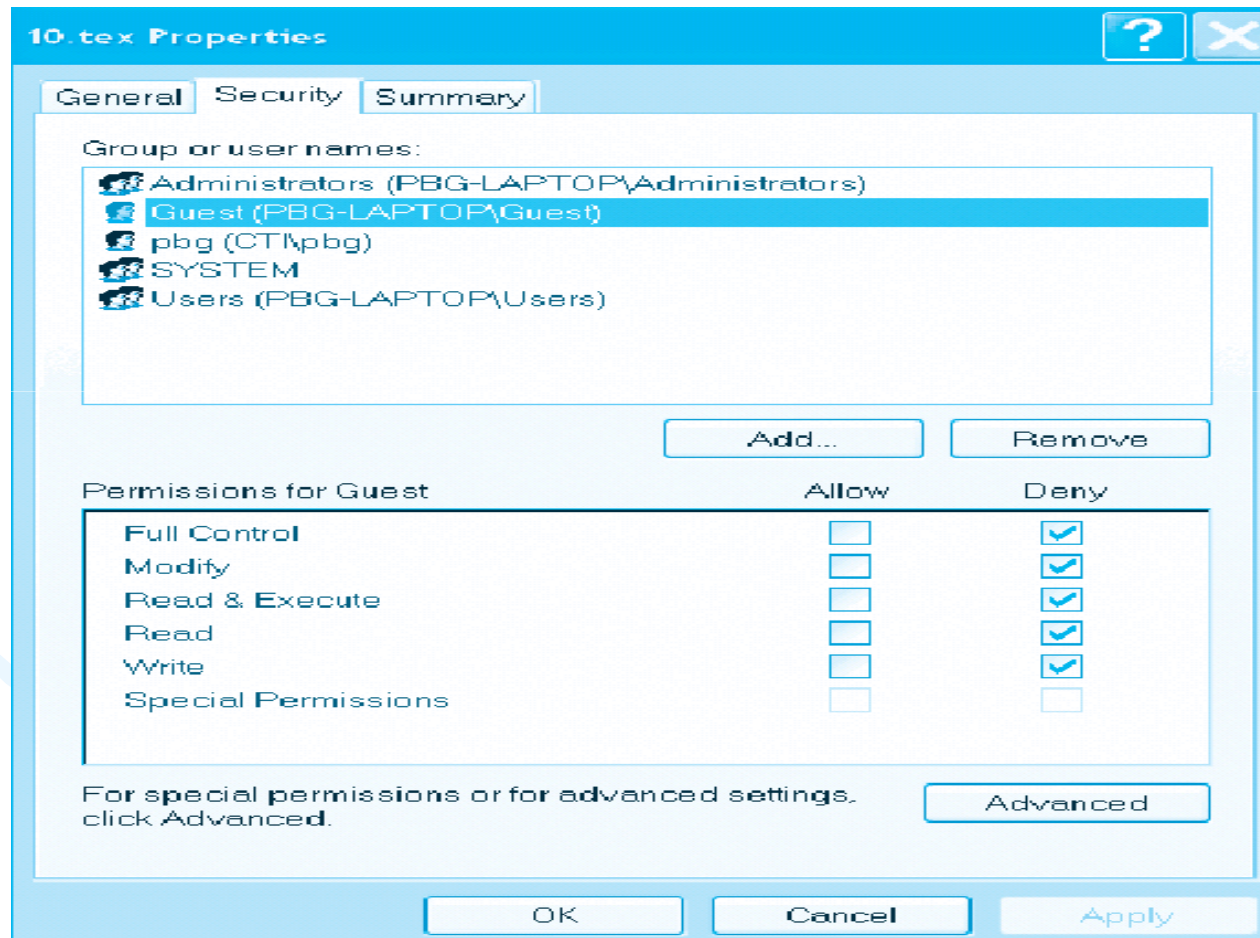
a) <b>owner access</b>	7	⇒	RWX 1 1 1
b) <b>group access</b>	6	⇒	RWX 1 1 0
c) <b>public access</b>	1	⇒	RWX 0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file

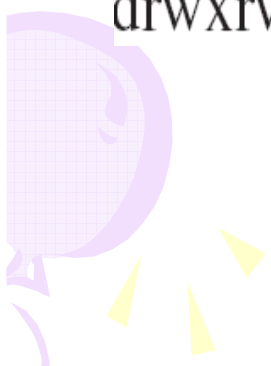

# Windows XP Access-control List Management







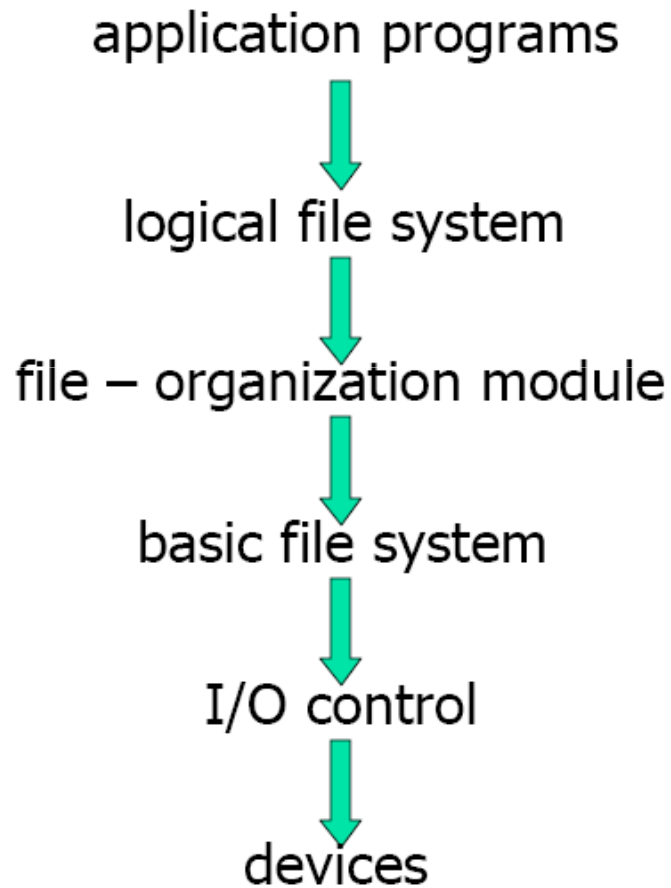
# A Sample UNIX Directory Listing



-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx-----	5 pbg	staff	512	Jul 8 09:33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx-----	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/



# Organisasi File System



# Hal yg berhubungan dengan File System

- *I/O control (driver device dan interrupt handler)*
  - **Device driver** adalah perantara komunikasi antara sistem operasi dengan perangkat keras
- *Basic file system*
  - Mengeluarkan **perintah** generic ke *device driver* untuk baca dan tulis pada suatu block dalam disk
- *File-organization module*
  - **Informasi** tentang *logical address* dan *physical address* dari file tersebut, mengatur juga sisa disk dengan melacak alamat yang belum dialokasikan dan menyediakan alamat tersebut saat user ingin menulis file ke dalam disk
- *Logical file system*
  - tingkat ini berisi **informasi** tentang simbol nama file, struktur dari direktori, proteksi dan sekuriti dari file tersebut



# Implementasi File System

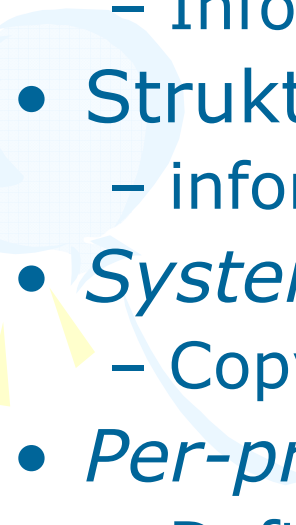

## **Struktur *On-disk***

- *Boot control block*
  - informasi sistem file pada sistem operasi
- *Partition block control*
  - spesifikasi partisi yang dimiliki
- Struktur direktori
  - mengatur file-file dalam direktori
- FCB (File Control Block)
  - detail-detail mengenai file yang spesifik



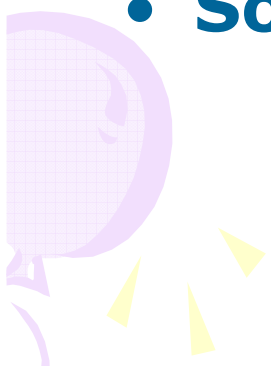
# Implementasi File System

## **Struktur In-Memory:**

- *Table partition*
    - Informasi semua partisi yang di-mount
  - Struktur direktori (LRU-stack)
    - informasi direktori yang paling sering diakses
  - *System wide open file table*
    - Copy-an dari FCB
  - *Per-process open file table*
    - Daftar pointer yang menunjuk access point dalam system wide open file table
- 
- 

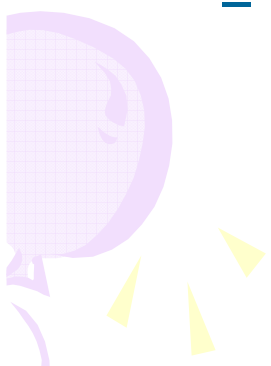


# Implementasi Direktori

- Asumsi direktori di Linux = File
  - Linier List: metoda paling sederhana: nama file dihubungkan dengan **pointer** ke data block
  - **Proses**: mencari (tidak ada nama file yang sama), tambah file baru pada akhir direktori, hapus (mencari file dalam direktori dan melepaskan tempat yang dialokasikan)
  - **Kelemahan**: *linear search* untuk mencari sebuah file, sehingga implementasi yang **lambat** pada cara aksesnya
  - **Solusi** yang mungkin: double linked list
- 




# Hash Table

- **Linear list** menyimpan direktori, sedangkan struktur data **hash** juga digunakan untuk penyimpanan
  - **Proses:** Hash table mengambil nilai yang dihitung dengan **function** dari nama file dan mengembalikan sebuah **pointer** ke nama file yang ada di linear list
  - **Kesulitan:** ukuran tetap dan ketergantungan dari fungsi hash dengan ukuran hash table
  - **Solusi: chained-overflow linked list**
    - setiap hash table mempunyai linked list dari nilai individual dan kita dapat mengatasi *collision* dengan menambah tempat pada linked list tersebut
- 



# Contoh implementasi Linux

- / : direktori root
  - /bin : perintah biner yang esensial
  - /boot : file statis dari *boot loader*
  - /dev : *device files*
  - /etc : konfigurasi sistem *host-specific*
  - /lib : *shared libraries essential* dan *modul kernel*
  - /mnt : *mount point* untuk me-mount suatu file system sementara
  - /opt : tambahan paket software application
  - /sbin : sistem binary esensial
  - /tmp : tempat file-file sementara
  - /usr : berisi file untuk user tertentu
  - /var : berisi variabel data
- 





# Alokasi Blok File System

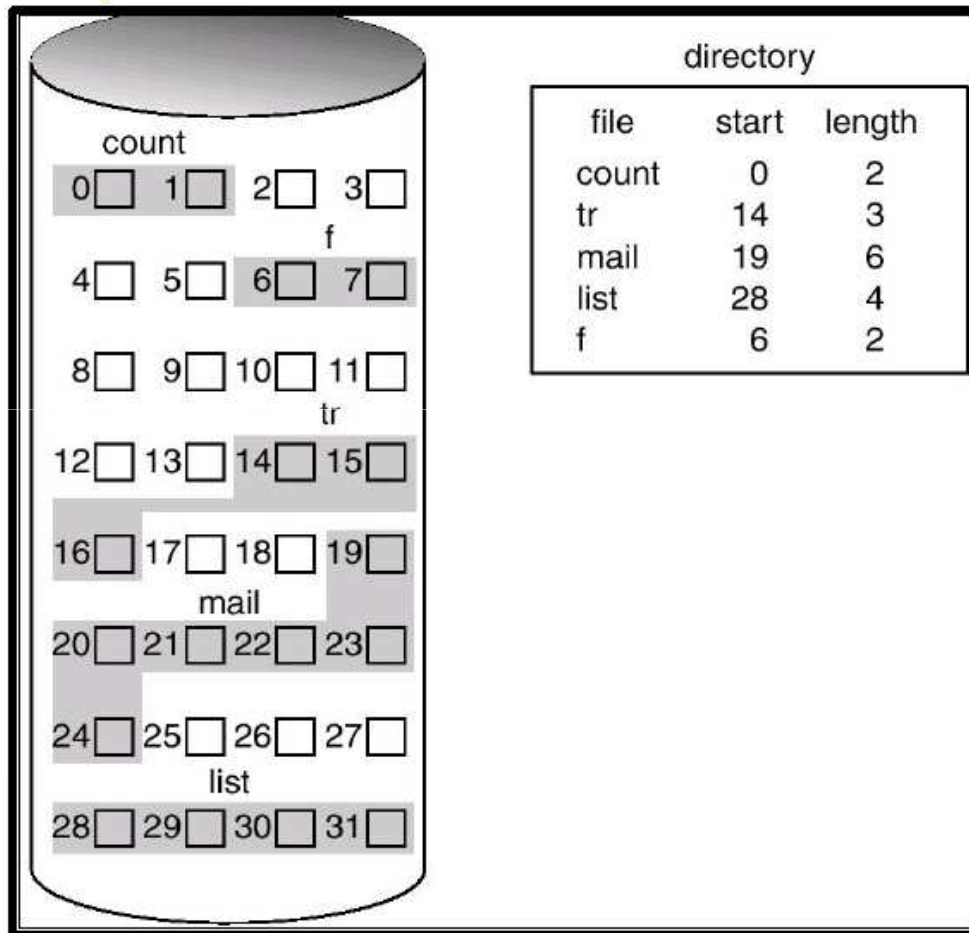
- Untuk mengalokasikan file agar dapat **diakses** dengan cepat dan disk dapat dimanfaatkan secara efektif
- Metode yang sering digunakan ialah:
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation



# Contiguous allocation

- Sebuah file didefinisikan oleh **alamat disk** (mendefinisikan urutan linier dari disk) dan **panjangnya** (dalam satuan blok) dari blok **pertama**
- Contiguous allocation mendukung pengaksesan secara **sekuensial** dan juga pengaksesan secara **langsung**

# Contiguous allocation



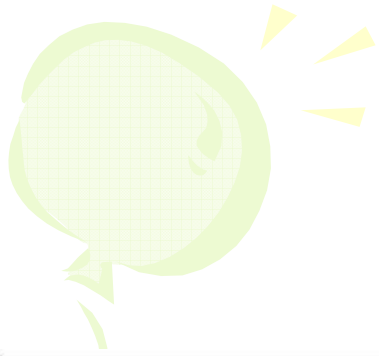
## MASALAH:

- Mencari ruang untuk file baru
- *External fragmentation*
- Menentukan berapa banyak ruang yang dibutuhkan untuk suatu file (harus dihitung)

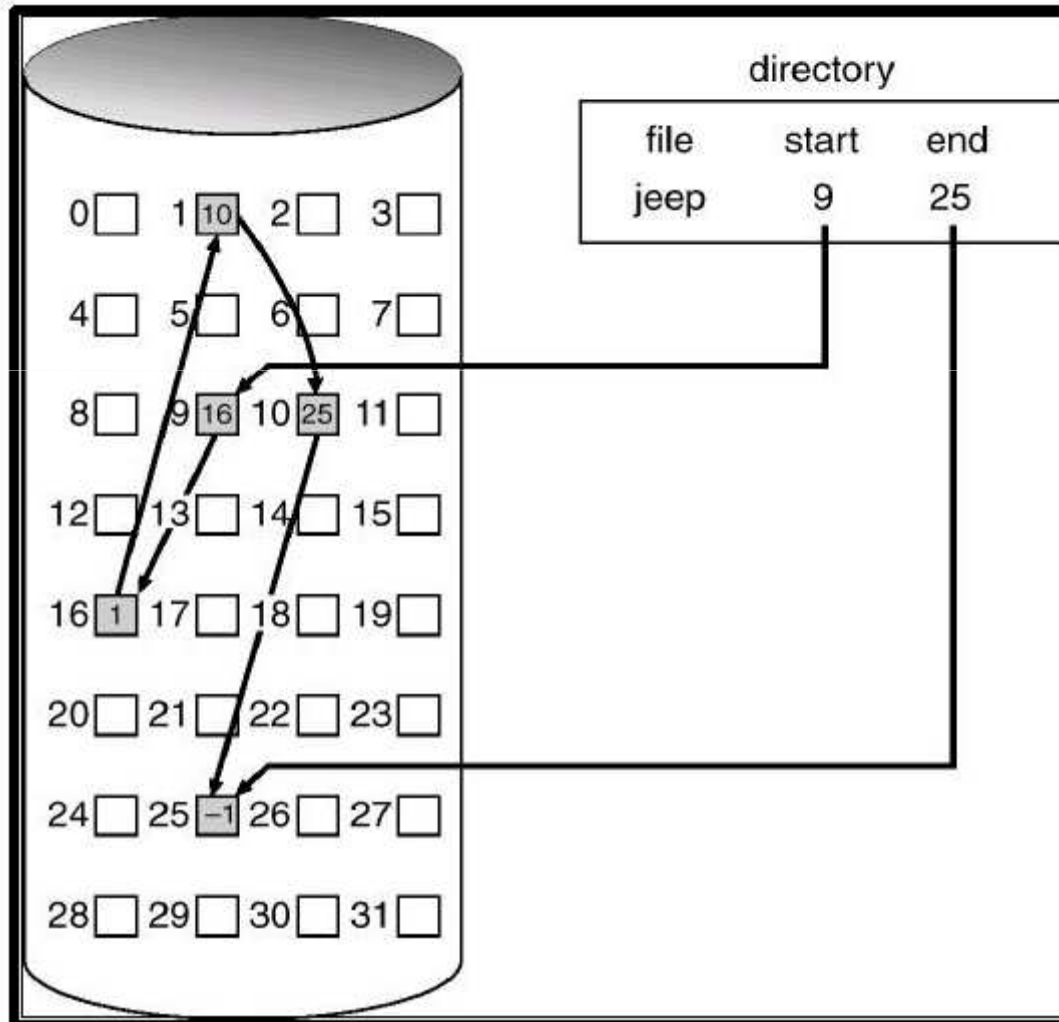


# Linked Allocation

- Direktori mengandung sebuah **pointer** untuk blok pertama dan blok terakhir dari sebuah file
- Setiap blok mengandung sebuah pointer untuk ke blok selanjutnya
  - tidak dapat di buat oleh user
- Efektif saat file diakses secara **sequential**
- **Kelemahan:** jika linked semakin panjang dan terpisah-pisah pencarian lambat



# Linked Allocation



## MASALAH:

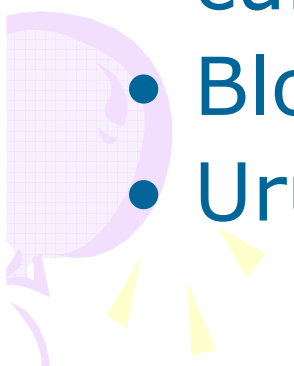
- Tidak efisien saat file diakses secara langsung
- Lambat
- Pointer membutuhkan ruang
- File berikutnya bergantung dengan file sebelumnya (dalam pointer)



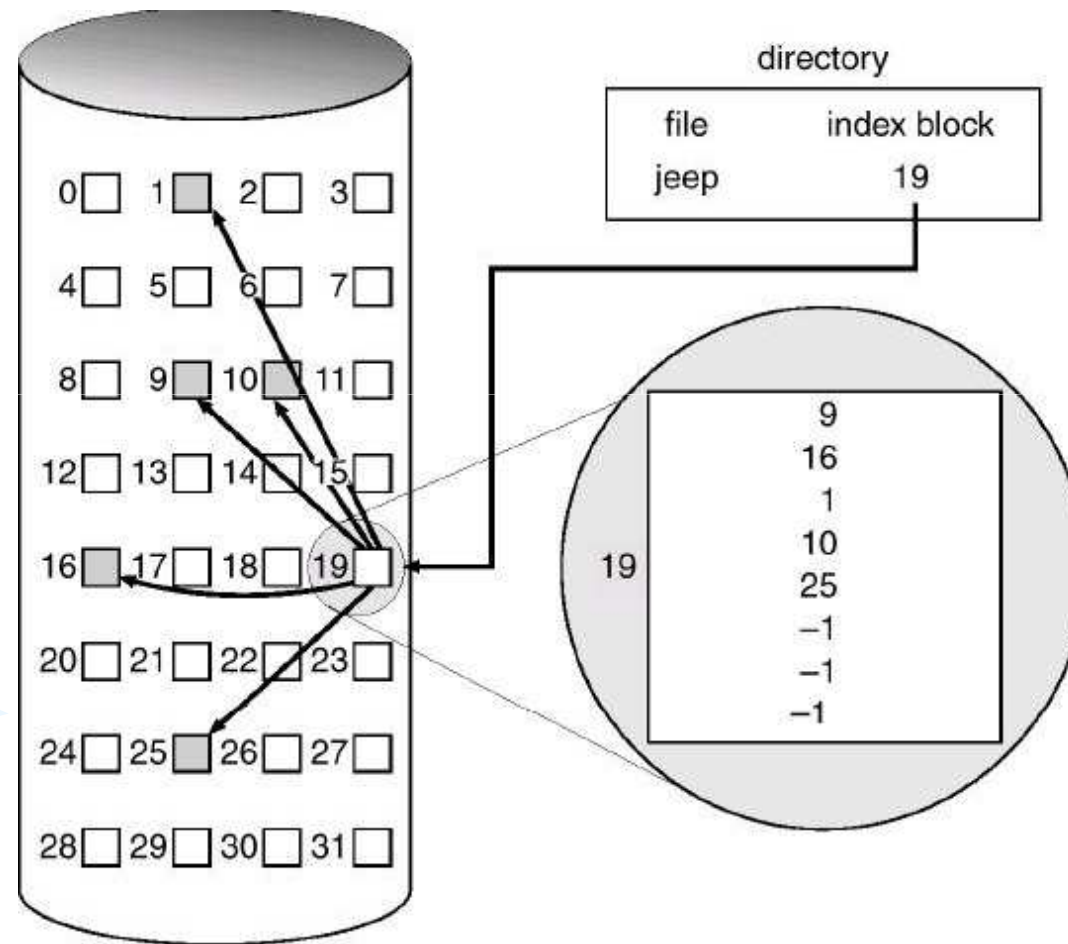
# Indexed Allocation

- Pointer digabungkan didalam suatu blok yang dinamakan **blok indeks**
- Setiap file memiliki blok indeks masing-masing

## MASALAH:

- Jika blok indeks **terlalu kecil**, maka tidak akan bisa memuat pointer yang cukup untuk sebuah file yang besar
  - Blok indeks membutuhkan **tempat**
  - Urutan pointer **berpengaruh**
- 

# Indexed Allocation



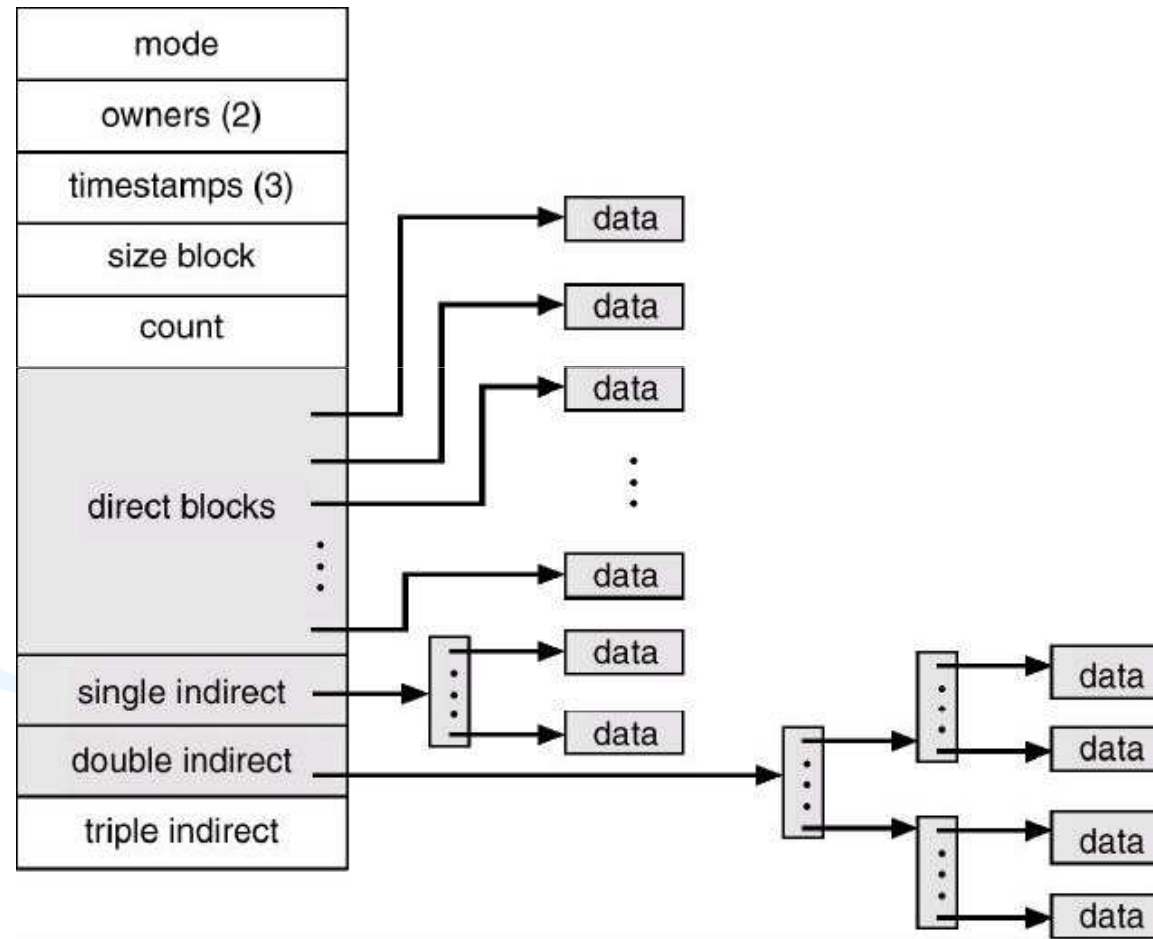


# Implementasi indexed allocation

- Linked scheme:
  - Mekanisme ini dapat menghubungkan beberapa blok indeks dengan **pointer**
  - Jika **pointer** tidak muat dalam satu blok indeks, maka pointer terakhir dari blok indeks ini menunjukkan blok indeks yang memuat pointer selanjutnya
  - Sifatnya langsung
- Multilevel index scheme:
  - Blok indeks pada level pertama akan menunjukkan blok-blok indeks pada level kedua yang akan menunjuk ke alamat data
  - Ini dapat diteruskan ke level ketiga atau level keempat tergantung dari jumlah data yang dibutuhkan
  - Sifatnya tidak langsung


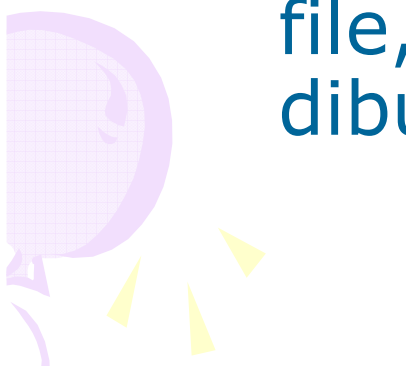


# Implementasi indexed allocation





# Kinerja dari Metode Alokasi

- Countiguous allocation:
    - Efisien untuk file kecil
    - Mendukung akses file secara langsung
  - Linked allocation
    - Mendukung akses file secara sequential
  - Indexed allocation
    - Tergantung dari struktur index, ukuran file, dan posisi dari blok yang dibutuhkan
- 
- 



# NEXT

- Sistem Input Output