# Sisteme de Operare

## Administrarea informațiilor : Sisteme de fișiere

**Cristian Vidrașcu**

https://profs.info.uaic.ro/~vidrascu

# Cuprins

- Conceptele de fișier și de sistem de fișiere
- Interfața sistemului de fișiere
- Implementarea sistemului de fișiere

# Conceptul de fișier /1

- **Fișier:** zonă de stocare contiguă d.p.d.v. logic
  (nu neapărăt contiguă și d.p.d.v. fizic)
- **Conținutul unui fișier:**
  – date (numerice, de tip caracter, binare)
  – program
- **Structura unui fișier:**
  – nestructurat (i.e., o secvență de octeți oarecare)
  – structură simplă de tip înregistrare (i.e., linii / înregistrări, de lungime fixă sau variabilă)
  – structură complexă (e.g. document formatat, fișier executabil cu încărcare relocabilă, etc.)
  – Cine decide structura (i.e., modul de interpretare a conţinutului) ? programul (UNIX) vs. sistemul de operare (Windows, OS/2)

# Conceptul de fișier /2

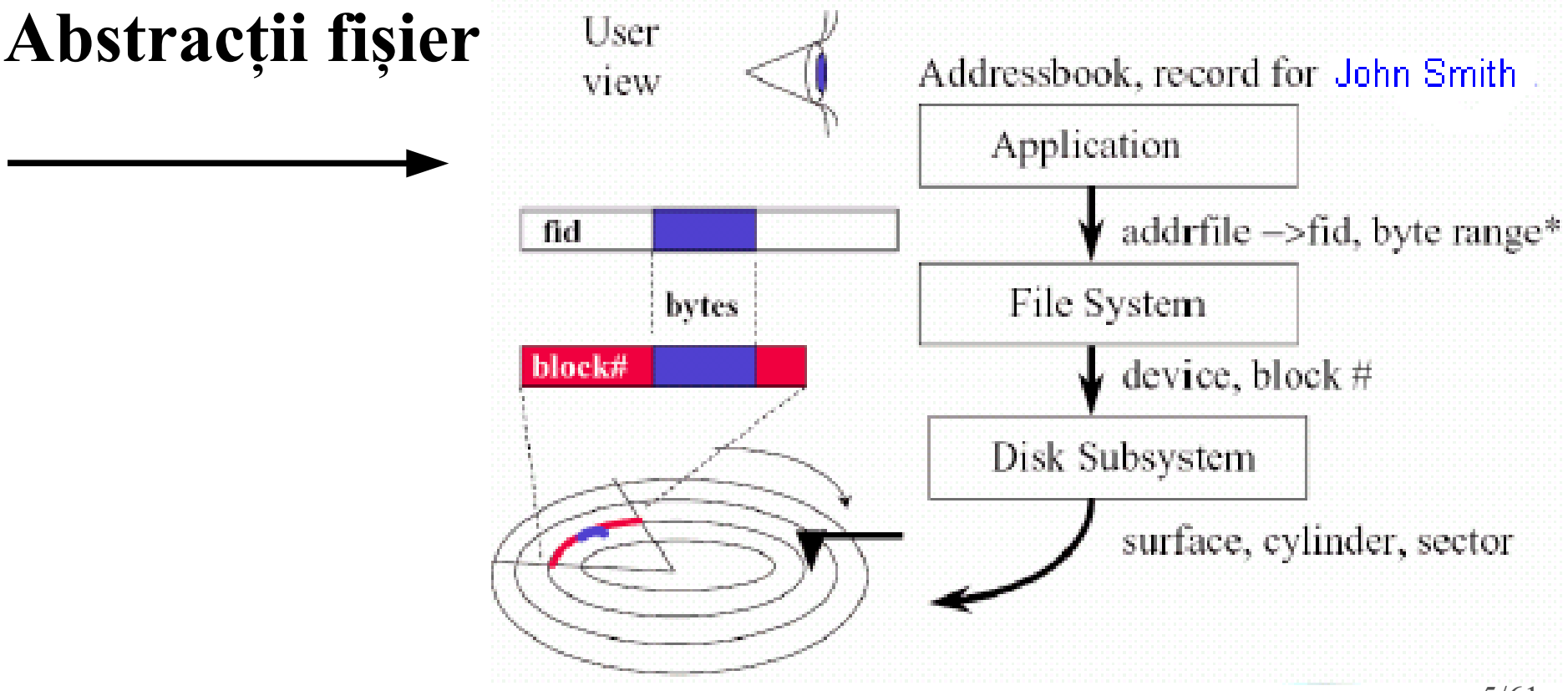
- **Structura unui fișier (cont.)**
  - Cine decide structura (i.e. modul de interpretare a conţinutului) ? programul (UNIX) vs. sistemul de operare (Windows, OS/2)

În al doilea caz (introdus o dată cu trecerea la interfețe GUI), decizia se ia pe baza **extensiei**, i.e. sufixul din numele fișierului:

| 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 |

# Conceptul de fișier /3

- **Rolul fișierelor:**
  - Persistență – date cu viață lungă, pentru posteritate
    - medii de stocare nevolatile
    - nume cu înțeles semantic (d.p.d.v. al utilizatorului)
- **Abstracții fișier**



# Atributele fişierului (metadate)

- Nume – singura informație păstrată în formă inteligibilă uman
- Tip – necesar pentru sistemele ce suportă diverse tipuri
- Locație – pointer la locația unde este stocat fișierul pe disc (i.e., perifericul de stocare al sistemului de fișiere din care face parte acel fișier)
- Dimensiune – dimensiunea curentă a fișierului
- Protecție – controlează cine poate citi, scrie, executa, ... fișierul
- Timp, dată și identificatorul de utilizator – informații pentru protecție, securitate și monitorizarea utilizării
- Informațiile despre fișiere sunt păstrate în structura de directoare, ce este menținută pe disc

# Operații asupra fișierelor

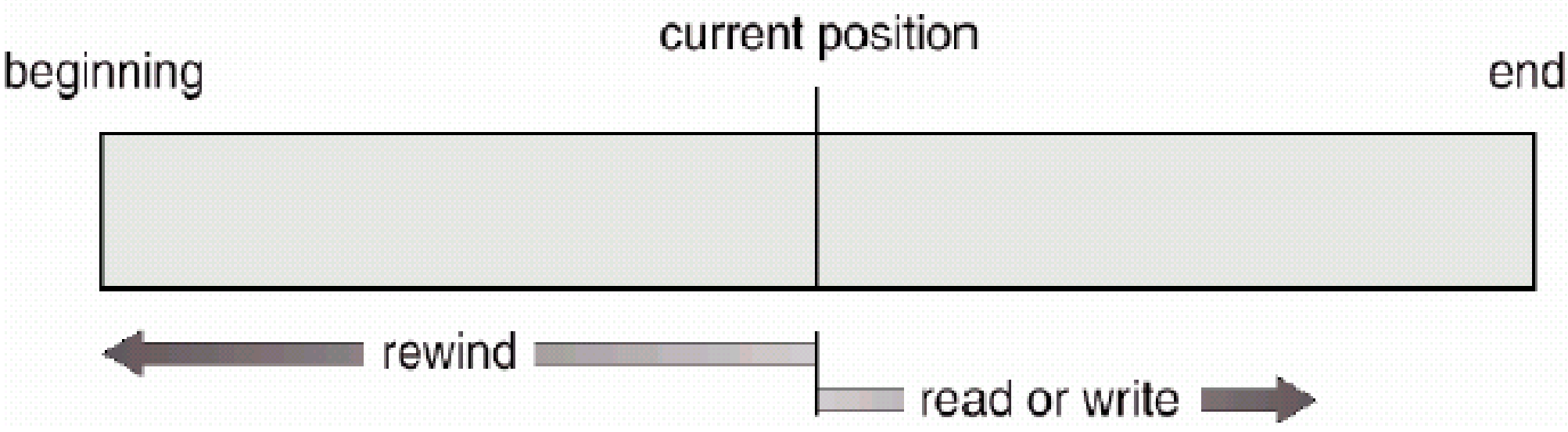
- Creare – creat() sau open(..., O_CREAT ...)
- Citire – read()
- Scriere – write()
- Repoziționare în fișier (i.e., căutare în fișier) – lseek()
- Ștergere – unlink()
- Trunchiere – truncate() sau ftruncate()
- Deschidere (i.e., căutarea în structura de directoare de pe disc a intrării fișierului și copierea conținutului intrării în memorie) – open()
- Închidere (i.e., mutarea conținutului intrării fișierului din memorie în structura de directoare de pe disc) – close()

# Accesul la fișiere /1

- Acces secvențial – acces de la început spre sfârșit, posibilitate de întoarcere la început (*rewind* ) pentru reluarea parcurgerii secvențiale; se pot face citiri și scrieri, dar nu ambele simultan în timpul aceleași deschideri – *metafora benzii magnetice*
- Acces direct (acces aleatoriu) – acces după numărul (i.e., adresa) înregistrării; se pot face citiri și scrieri în orice combinații dorite – *metafora discului de vinyl*
- Acces indexat – acces direct prin conținut, folosind diverse tehnici (e.g. fișiere de index, fișiere multilistă, *hashing*, B-arbori, ș.a.)

# Accesul la fișiere /2

- Acces secvențial



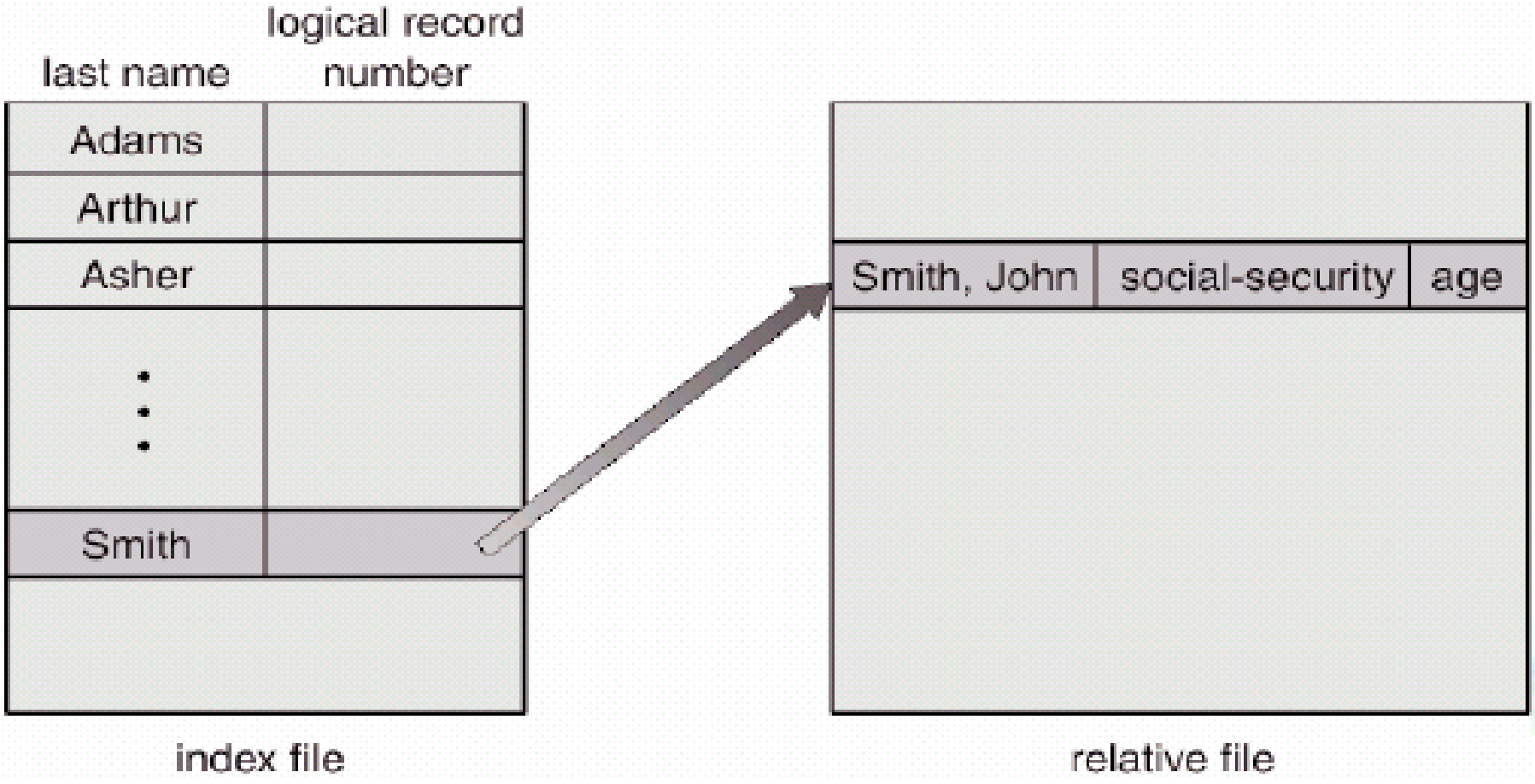
# Accesul la fișiere /3

- Simularea accesului secvențial pe un fișier cu acces direct

| acces secvențial | implementarea cu acces direct |
|---|---|
| reset | cp = 0; |
| read next | read cp;<br>cp = cp + 1; |
| write next | write cp;<br>cp = cp + 1; |

# Accesul la fișiere /4

- Exemplu de acces indexat – fișiere index și relative



# Clasificări ale fișierelor

- Clasificarea după posibilitatea de tipărire (afișare) ASCII:
  – fișiere text
  – fișiere binare
- Clasificarea după suportul pe care sunt rezidente:
  – fișiere pe disc magnetic (HDD, FD, ZIP, ...) sau tambur magnetic
  – fișiere pe bandă magnetică sau casetă magnetică
  – fișiere pe suport optic (CD, DVD)
  – fișiere pe suport memorii NAND (stick USB, card SD/MMC/... , SSD, etc.)
  – fișiere pe imprimantă sau plotter
  – fișiere pe cartele perforate sau bandă de hârtie perforată
  – fișiere pe ecran, fișiere tastatură, ș.a.
- Suportul influențează operațiile și modurile de acces posibile

  E.g. suportul disc acceptă accesul direct, suportul bandă magnetică acceptă doar accesul secvențial; fișierele pe cartele sau tastatură acceptă doar citiri, iar cele pe imprimantă, plotter sau ecran acceptă numai scrieri

# Sisteme de fișiere/1

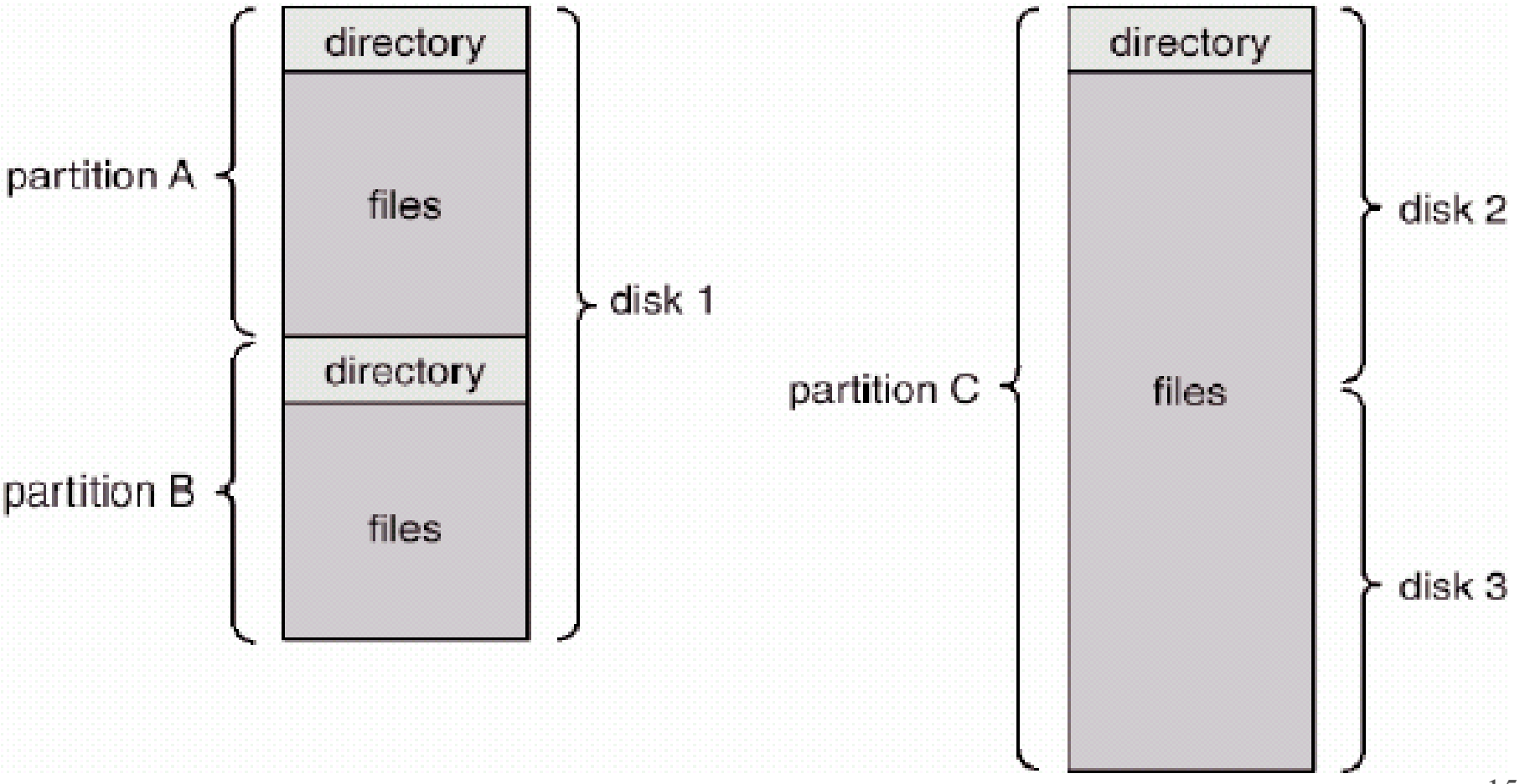
- Definiție: un **sistem de fișiere** este o colecție oarecare de fișiere, împreună cu structura de directoare în care sunt acestea organizate
- Există mai multe **tipuri** de sisteme de fișiere de uz general, e.g. *NTFS* (specific Windows) sau *ext2fs*/*ext3fs*/*ext4fs*, *btrfs*, *zfs*, ș.a. (specifice Linux/UNIX), ce diferă prin structurile de date folosite pentru stocarea lor și prin modul de implementare a operațiilor asupra fișierelor și directoarelor.
- În plus, există și diverse tipuri de sisteme de fișiere de uz particularizat, e.g. tmpfs (RAM disk), objfs (kernel debugging), procfs (interfață structuri procese), ș.a.
- **Cluster**: unitatea de măsură pentru alocarea spațiului necesar memorării unui fișier, la nivelul *file-system*-ului. Cu alte cuvinte, dimensiunea unui fișier (i.e. informația utilă stocată în el) este un atribut exprimat în octeți, însă spațiul ocupat de conținutul fișierului pe disc este un multiplu de clustere.
- *Dimensiunea unui cluster*: o putere de forma $2^n$, cu $n=0,1,2,\ldots$, în termeni de **sectoare** (sau *blocuri-disc*, i.e. unitatea de stocare la nivelul *discului*)

# Sisteme de fișiere/2

- Pentru persistență, sistemele de fișiere se păstrează pe ***dispozitive de stocare nevolatilă*** (e.g. disc HDD, stick USB, CD/DVD, disc SSD, etc.), în entități numite **volume**.
- Un volum poate fi (mai multe detalii – în cursul despre perifericele de stocare):
- – o ***partiție*** *a unui disc* (schema clasică, MBR/GPT, ce permite stocarea mai multor volume *per* disc)
- – un *disc* ***întreg*** (e.g. stick USB, CD/DVD, etc.)
  – un ***ansamblu*** *de mai multe discuri* (ansamblu ce stochează un singur sistem de fișiere, prin tehnici precum RAID)
- Formatarea logică a unui volum = "crearea sistemului de fișiere" rezident pe acel volum, prin inițializarea (pe disc) a structurilor de date specifice tipului acelui sistem de fișiere; operația are ca parametri: tipul sistemului de fișiere, numele volumului creat (i.e., o etichetă) și dimensiunea clusterului (definită în termeni de blocuri disc)

# Sisteme de fișiere/3

- Un exemplu de organizare a sistemului de fișiere



# Interfața sistemului de fișiere

- Funcții
- Structura de directoare
- Organizare
- Partajare
- Montare
- Protecție

# Funcțiile sistemului de fișiere

- (Subsistemul de directoare) Mapează numele de fișiere în identificatori de fișiere prin primitiva open (sau creat). Creează structurile de date din nucleu
- Menține structura de nume (unlink, mkdir, rmdir)
- Determină plasarea fișierelor (și a metadatelor) pe disc, în termeni de clustere. Gestionează alocarea clusterelor (1 cluster = $2^n$ sectoare consecutive). Gestionează clusterele eronate (i.e., cele ce conțin *bad sectors*)
- Gestionează apelurile de sistem read și write
- Inițiază operațiile I/O pentru transferul blocurilor dinspre disc spre RAM, respectiv dinspre RAM spre disc
- Gestionează tampoanele *cache* pentru disc

# Structura de directoare /1

- **Director** – o colecție de fișiere, inclusiv sub-directoare (referite prin structuri de date specializate ce conțin informații despre acestea)
- Atât fișierele propriu-zise, cât și structura de directoare sunt păstrate pe disc
- Copiile de siguranță ale acestora sunt păstrate (de obicei) pe benzi, CD-ROM-uri, ș.a.
- La început: director unic (e.g. CP/M, SIRIS – '60), iar apoi director pe 2 nivele (e.g. RSX – '70-'80, un S.O. în timp real pentru DEC PDP-11)
- În prezent: directoare cu structura arborescentă sau cu structură de graf aciclic

# Structura de directoare /2

- Cum este organizată structura de directoare?
  - Listă înlănţuită
  - Vector sortat
  - Tabelă hash

(a se vedea la Implementarea sistemului de fișiere)

# Operații asupra directoarelor

- Căutarea unui fișier
- Crearea unui fișier
- Ștergerea unui fișier
- Redenumirea unui fișier
- Listarea unui director
- Traversarea sistemului de fișiere
- e.g. salvarea/restaurarea unui sistem de fișiere

# Numirea fișierelor

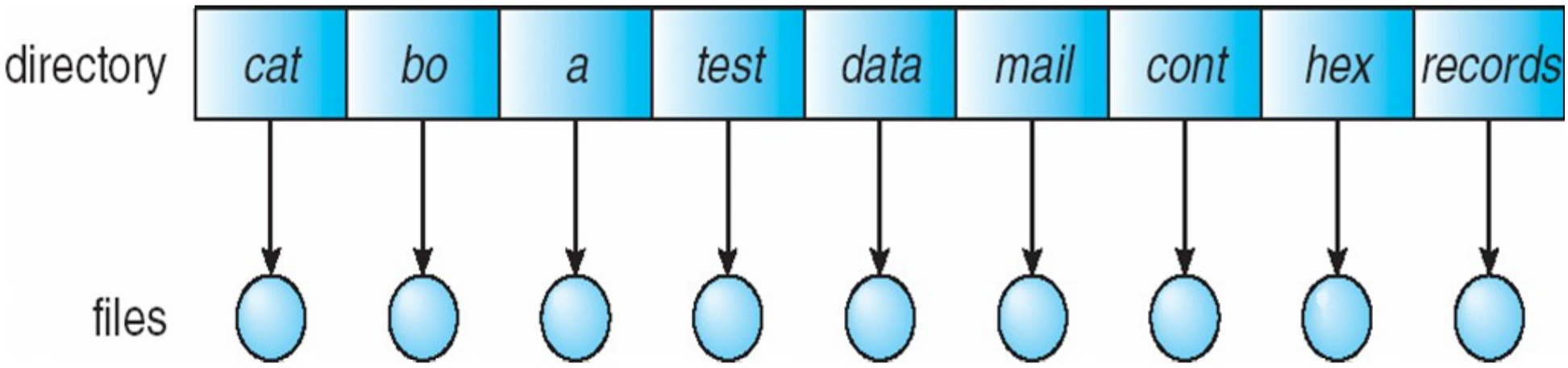
- Scopuri:
  - Pentru a identifica fișierele (la deschidere, SO-ul mapează numele de fișiere în obiecte de tip fișier utilizate intern pentru accesul la fișiere)
  - Pentru a permite utilizatorilor să-și aleagă propriile nume de fișiere fără probleme de conflict de nume
  - Ușurința de utilizare: nume scurte, grupări de fișiere

# Organizarea directoarelor /1

- Organizarea (logică a) directoarelor, pentru a obține:
  - **Eficiență** – localizarea rapidă a unui fișier
  - **Nume** – numele sunt utile pentru utilizatori
    - Doi utilizatori pot avea același nume pentru fișiere diferite
    - Același fișier poate avea mai multe nume diferite
  - **Grupare** – gruparea logică a fișierelor după proprietăți (e.g. toate programele C, toate jocurile, etc.)

# Organizarea directoarelor /2

- Director unic pentru toți utilizatorii:

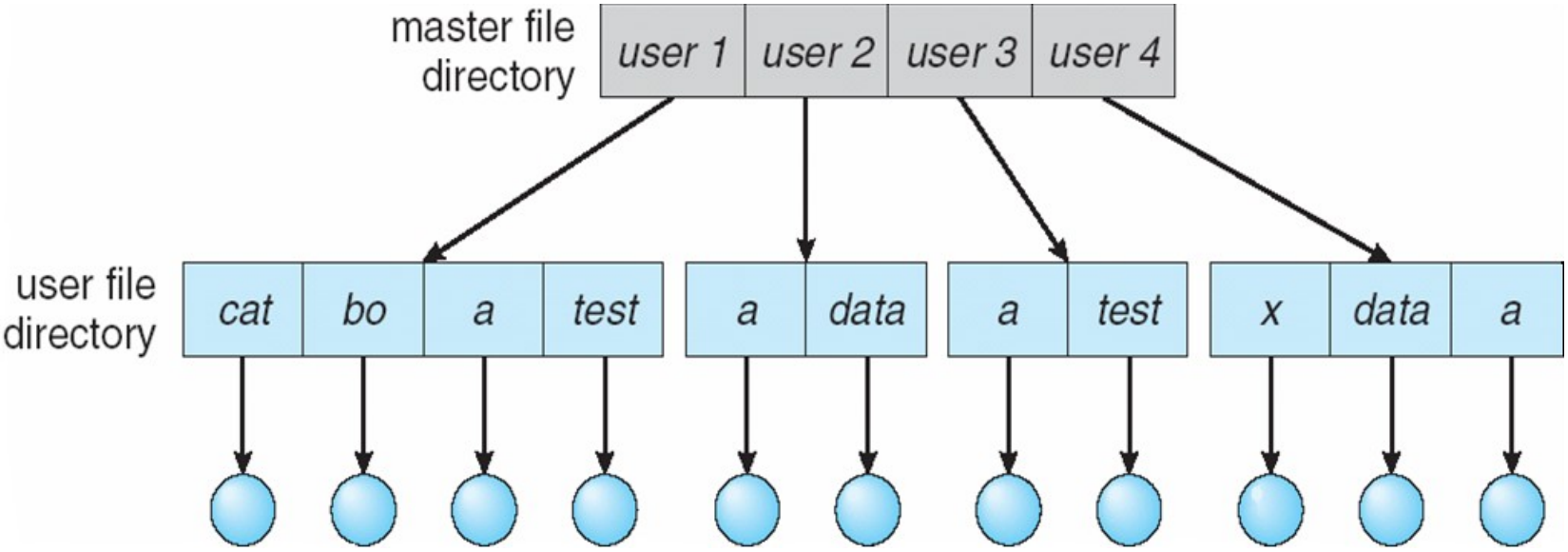



Probleme (dezavantaje):

– conflict de nume
– imposibilitate de grupare

# Organizarea directoarelor /3

- Director separat pentru fiecare utilizator:




Avantaje:
– cale (*path name*) – căutare eficientă
– același nume de fișier pentru utilizatori diferiți

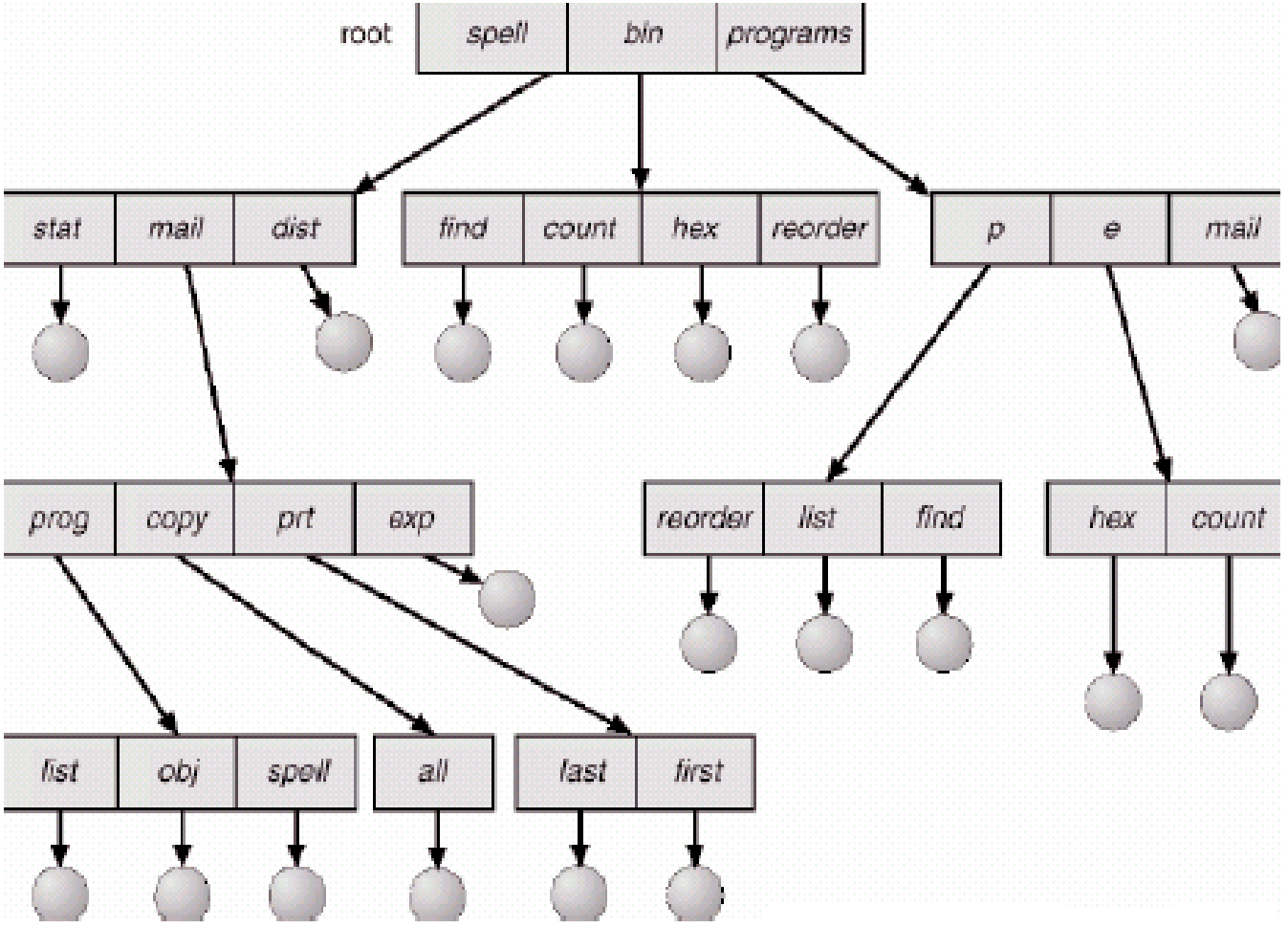
Dezavantaje:
– imposibilitatea de grupare

# Organizarea directoarelor /4

- Directoare cu structură arborescentă:

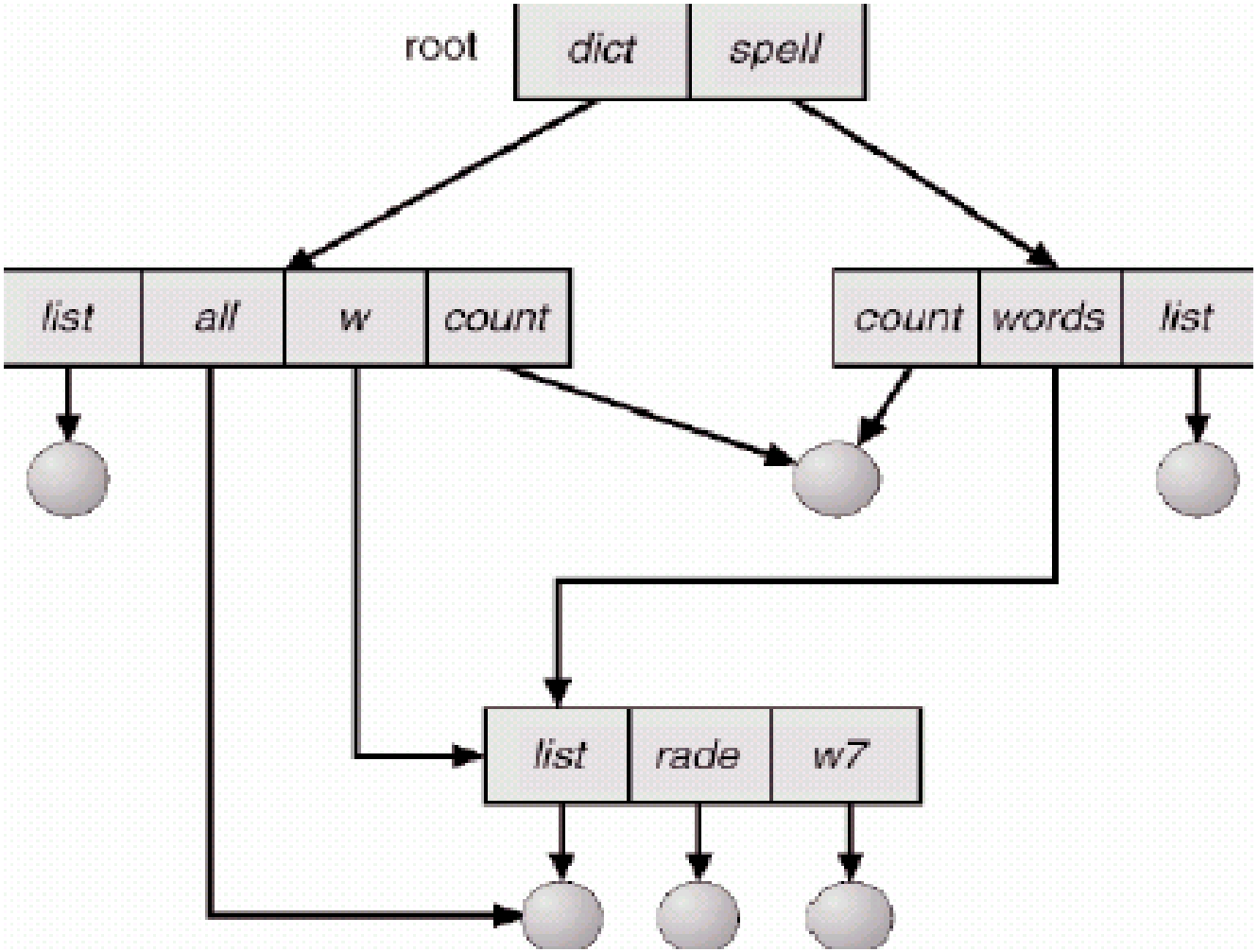
Avantaje:
– cale (*path name*)
– căutare eficientă
– posibilitatea de grupare a fișierelor
– director curent (de lucru) =>
specificarea de căi **relative** vs **absolute**

Dezavantaje:
– imposibilitatea de partajare (prin *alias*-uri)



# Organizarea directoarelor /5

- Directoare cu structură de graf aciclic:

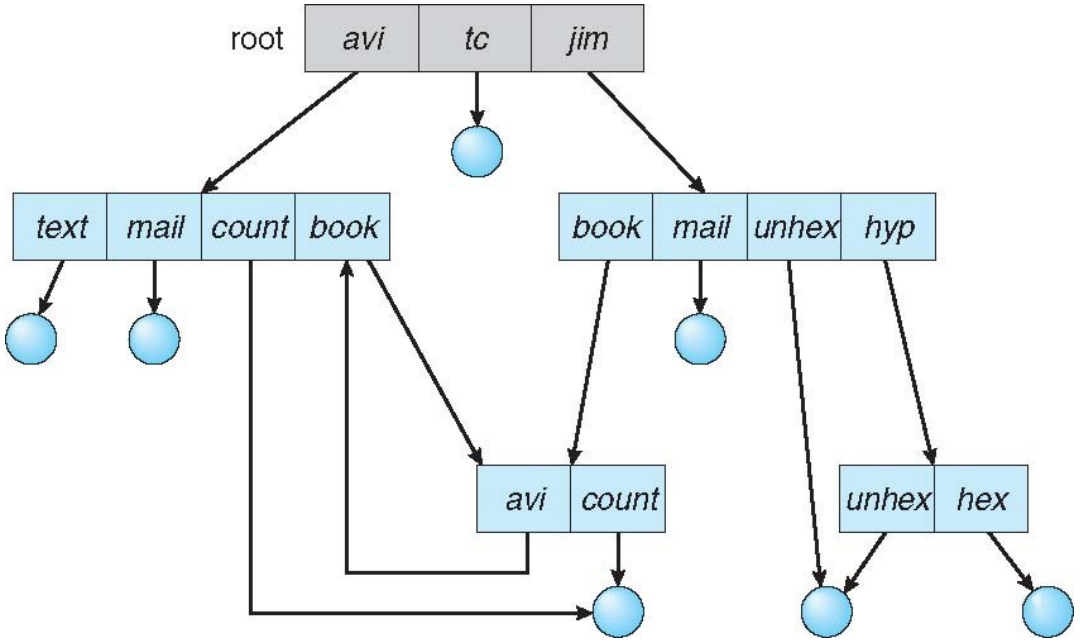


# Organizarea directoarelor /6

- Directoare cu structură de graf aciclic (cont.)
  - Pot exista subdirectoare și fișiere partajate
  - Pot avea două nume diferite
  - *linking* hard / soft – link() , symlink()
  - **Problemă:** ștergerea unui fișier partajat
  - **Soluții:**
    - Back-pointers, pentru a putea șterge toate referințele
      Problemă: înregistrări cu lungime variabilă
    - Soluția cu contor de referințe păstrat în intrarea fișierului

# Organizarea directoarelor /7

- Directoare cu structură de graf general:
  - **Problemă:** cum garantăm inexistența circuitelor?
  - **Soluții:**
    - Permitem *link*-uri doar către fișiere, nu și către subdirectoare
    - De fiecare dată când se adaugă un nou *link*, să se utilizeze un algoritm de detecție a circuitelor pentru a decide dacă să se permită acea adăugare sau nu
    - *Garbage collection*

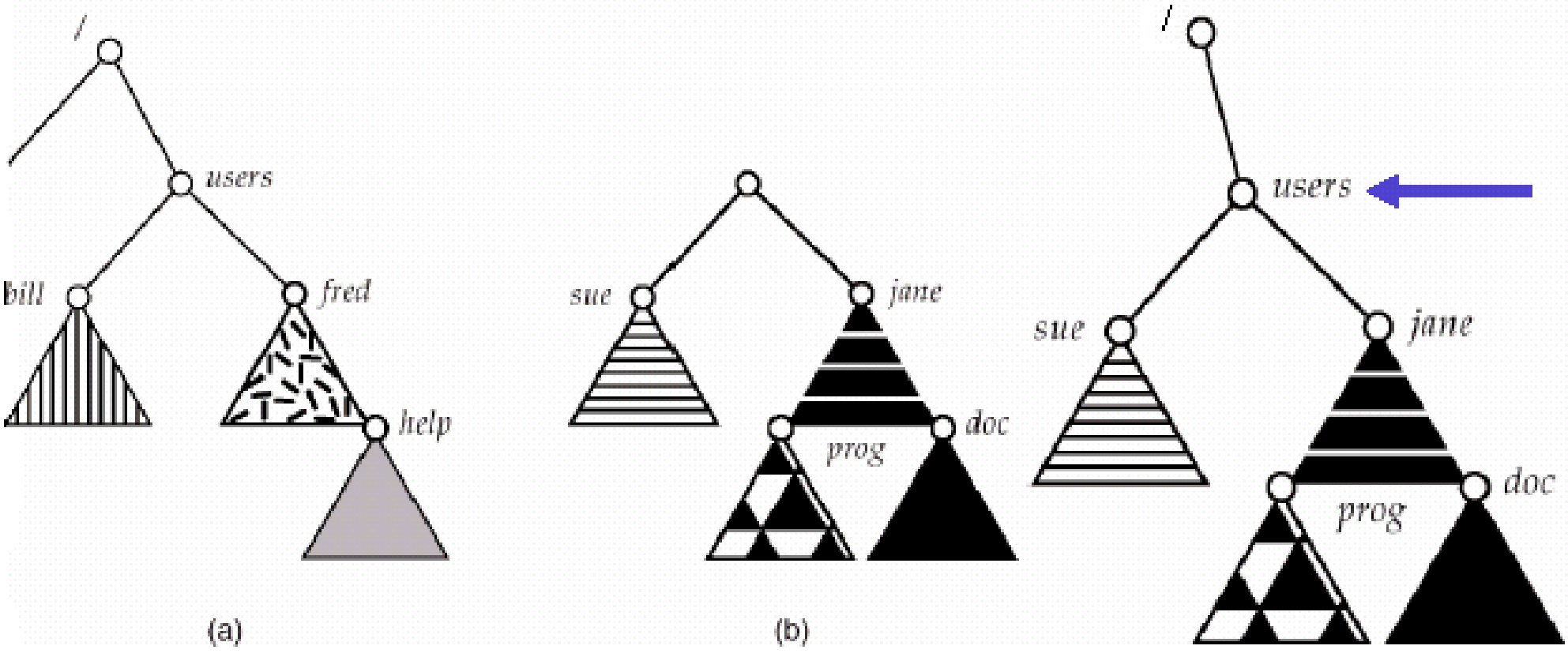


# Montarea /1

- Un sistem de fișiere trebuie să fie **montat** înainte de a putea fi accesat
- Un sistem de fișiere nemontat se montează la un **punct de montare** (un director de montare)
- Auto-montarea (la introducerea unei dischete, CD, stick USB, ș.a.)
- În UNIX/Linux: mount() , umount()

# Montarea /2

- Exemplu:




(a) Sistemul existent (b) Partiție nemontată Punctul de montare

# Partajarea fișierelor

- Partajarea fișierelor în sisteme multi-utilizator este o facilitate de dorit
- Partajarea se poate face printr-o schemă de protecție
- În sisteme distribuite, fișierele pot fi partajate prin intermediul rețelei
- Protocoale și tehnologii de rețea folosite:
  - NFS (Networked File System), ce utilizează RPC – SUN
  - CIFS: protocolul standard din Windows pentru partajarea fișierelor
  - Active Directory-ul din rețele Windows
  - Yellow Pages → NIS → NIS+ → LDAP, folosite în rețele UNIX/Linux
  - sistemul DNS

# Protecția fișierelor

- Proprietarul (creatorul) unui fișier trebuie să poată controla:
  – ce operații pot fi făcute asupra fișierului
  – și de către cine
- Drepturi de acces:
  – Read
  – Write (doar modificare, nu și append sau delete)
  – Execute
  – Append (adăugare de articole noi la sfârșitul fișierului)
  – Delete
  – List

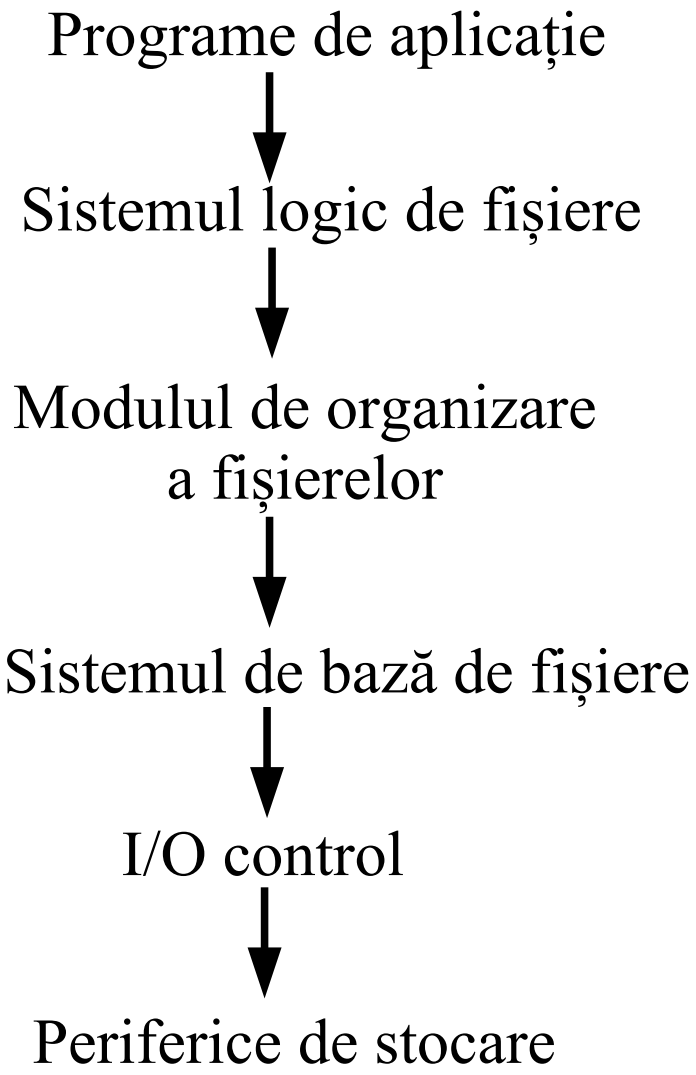
# Implementarea sistemului de fișiere

- Structura sistemului de fișiere
- Implementarea fișierelor și directoarelor
- Metode de alocare
- Gestiunea spațiului liber
- Eficiența și performanța
- Recuperare
- Sisteme de fișiere jurnalizate
- Networked File System (NFS)
- Sisteme de fișiere "*next-generation*"

# Structura sistemului de fişiere

- Structura fișierului
  – Unitatea logică de stocare a informațiilor (date sau programe)
  – O colecție de informații înrudite prin înțelesul lor semantic
- Sistemul de fișiere se păstrează pe memoria secundară (i.e., periferice de stocare – discuri: HDD, SSD; FD, CD/DVD, ș.a.)
- Sistem de fișiere organizat ierarhic, pe nivele
- Blocul de control al fișierului – **FCB** (*File Control Block*) – structura de date, stocată pe disc, ce conține anumite informații despre un fișier
- Deschiderea unui fișier pune la dispoziție un *handle* de referință la FCB, utilizat pentru accesele la fișier

# Sistem de fişiere pe nivele



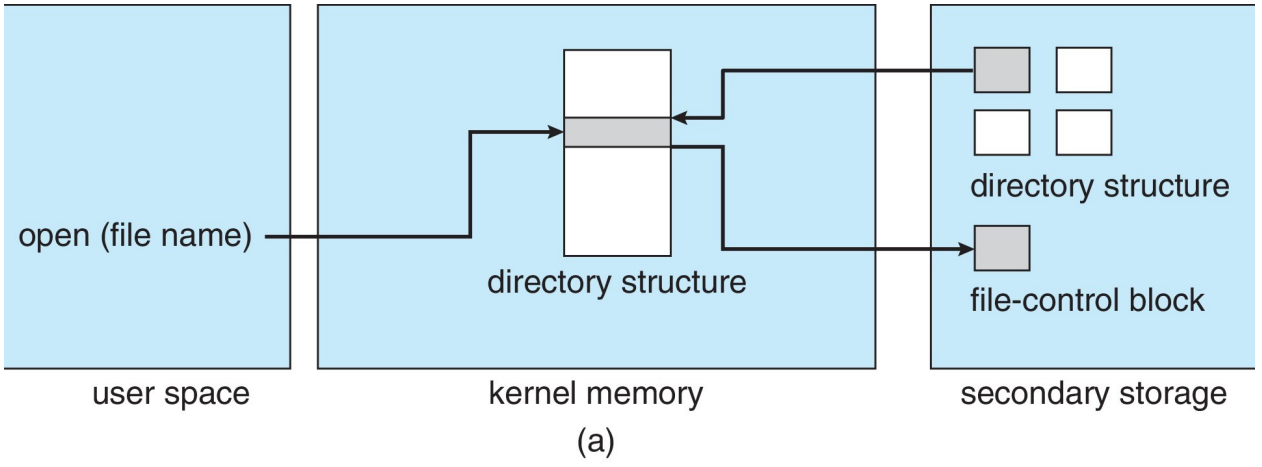

| permisiuni de acces la fișier |
| --- |
| data (creării, actualizării, accesării) |
| proprietar, grup, ACL |
| dimensiunea fișierului |
| blocurile de date |

FCB-ul unui fișier

# Structuri interne

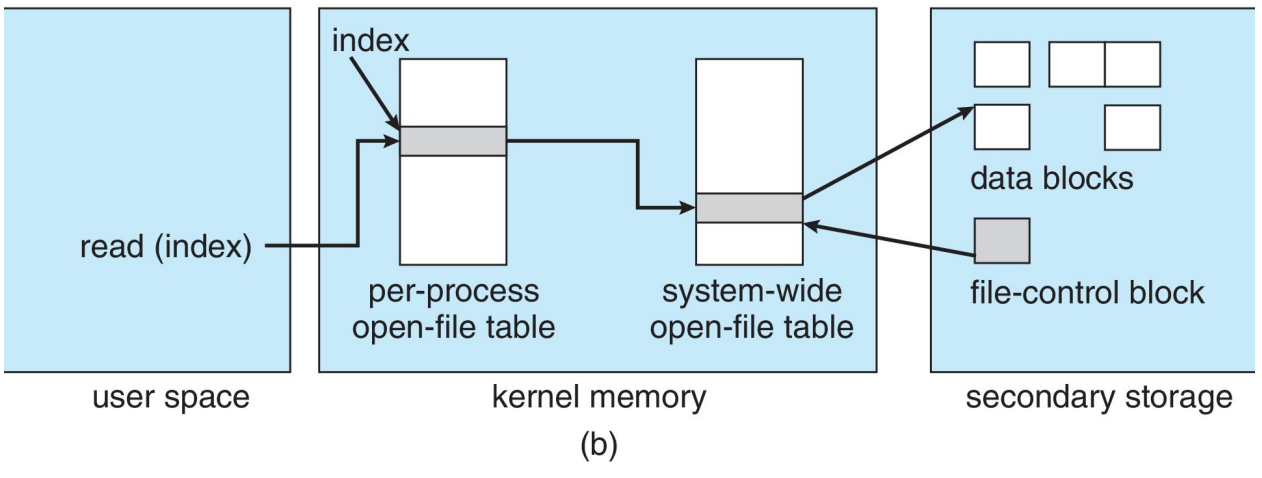
Structurile sistemului de fișiere necesare, furnizate de SO-uri:

- fig. (a): deschiderea
- fig. (b): citirea (similar pentru scriere)




Structuri de date în memorie:

- tabela de montare a sistemelor de fișiere
- tabela de fișiere deschise per proces
- tabela globală de fișiere deschise

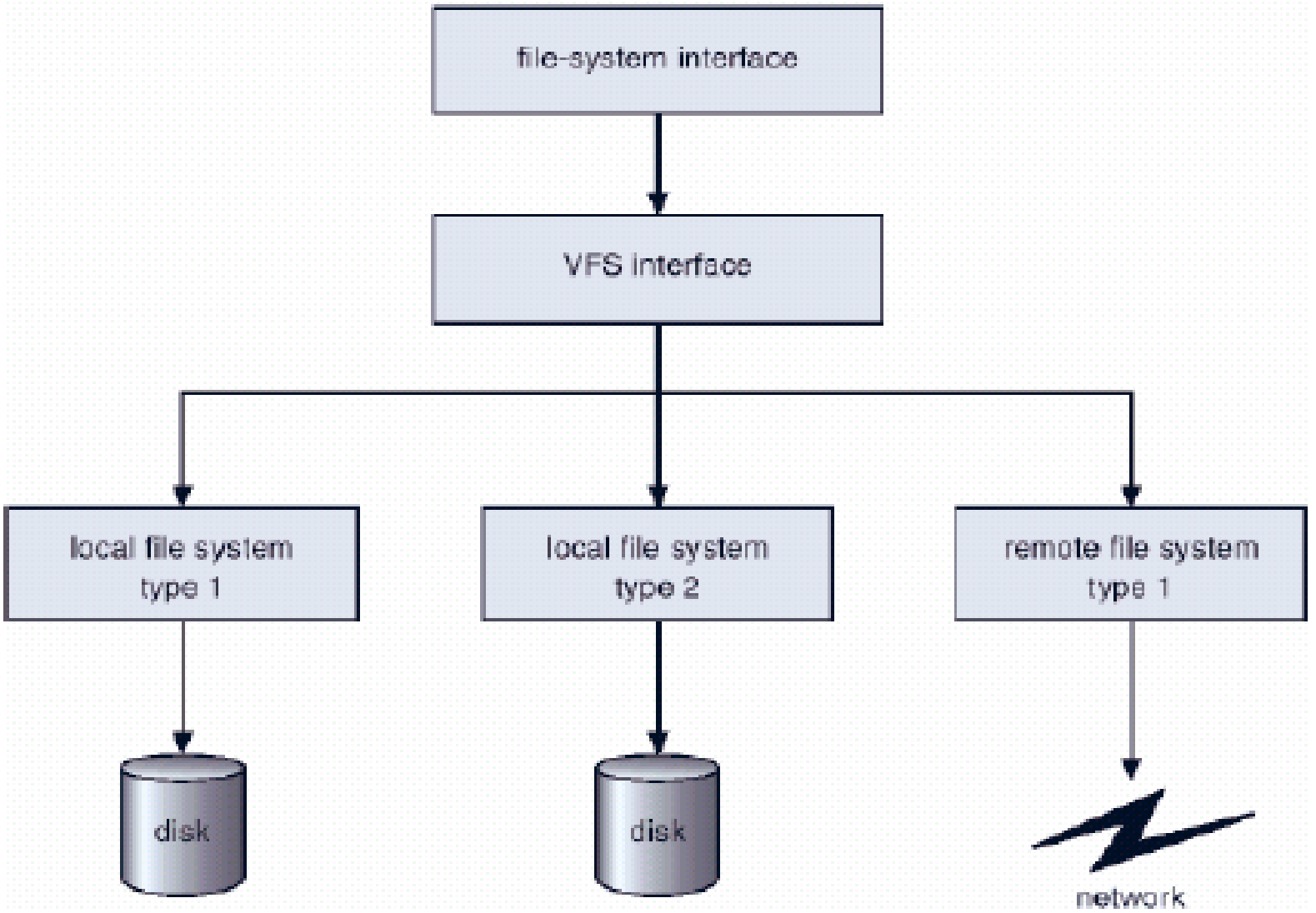


# Sisteme de fișiere virtuale /1

- VFS furnizează o modalitate orientată-obiect de implementare a sistemelor de fișiere
- VFS permite utilizarea aceleași interfețe a apelurilor de sistem pentru operații I/O (i.e., API-ul I/O din POSIX) pentru diferite tipuri de sisteme de fișiere
  (e.g. ext3fs, ext4fs, vfat, fat16, hpfs, ntfs, nfs, ...)
- VFS este o componentă din nucleul Linux

# Sisteme de fișiere virtuale /2

- Schema unui VFS:



# Implementarea directoarelor

- Listă liniară de nume de fișiere cu pointeri către blocurile de date
  - simplu de programat
  - consumator de timp la execuție
- Tabelă hash – listă liniară cu structură de date hash
  - micșorează timpul de căutare în director
  - coliziuni – situații în care două nume de fișiere au prin funcția hash aceeași locație
  - dimensiune fixată
- B+ tree

# Metode de alocare /1

- Metoda de alocare se referă la modul în care blocurile disc sunt alocate pentru fișiere (i.e., pentru a stoca conținutul propriu-zis al fișierului, nu și metadatele sale – acestea sunt stocate în FCB-ul fișierului)
  - **Alocare contiguă**
  - **Alocare înlănțuită**
  - **Alocare indexată**

# Metode de alocare /2

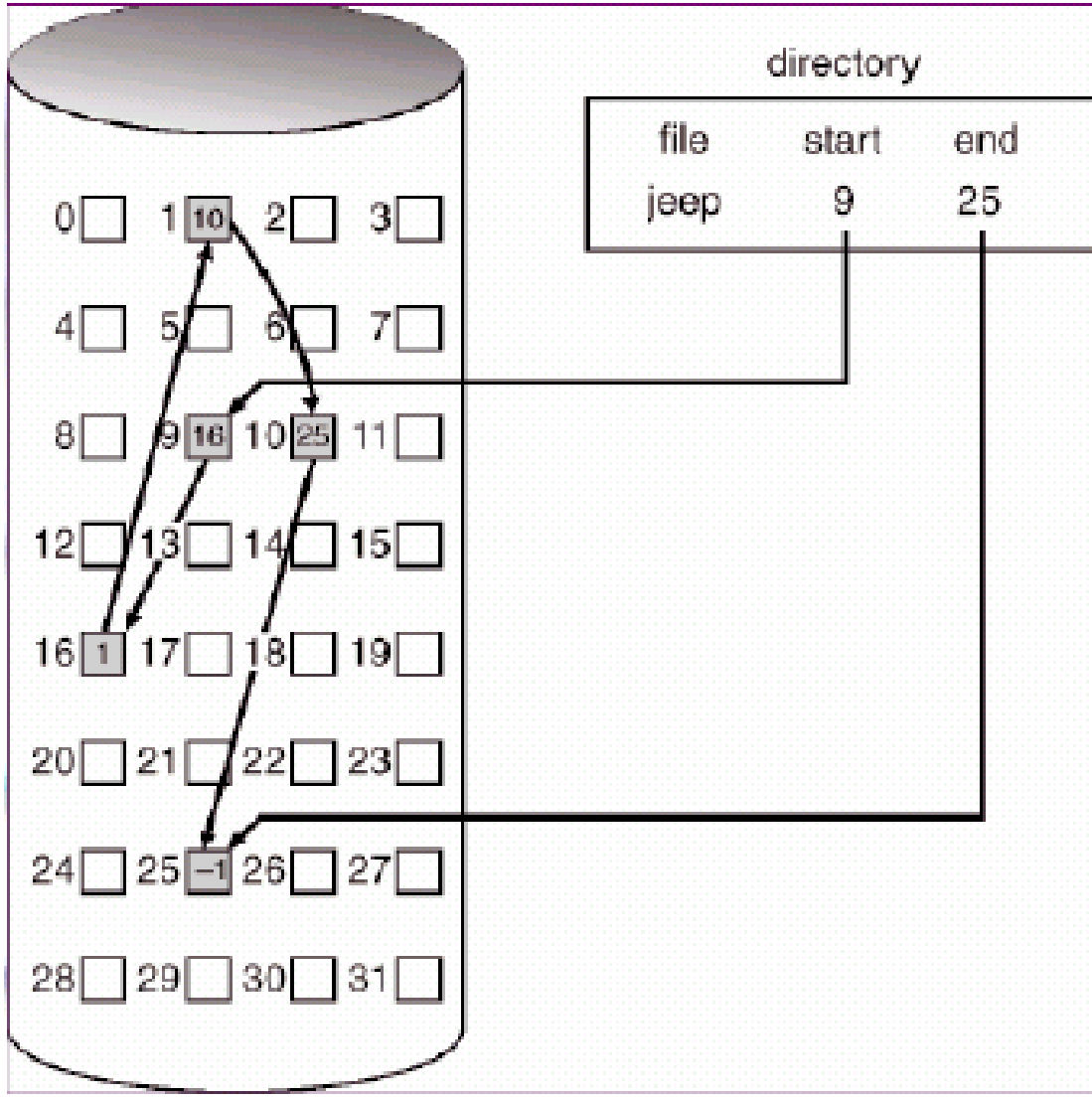
- **Alocarea contiguă**
  - Fiecare fișier ocupă o mulțime contiguă de blocuri pe disc
  - Simplă – sunt necesare doar locația de start (numărul blocului de început) și lungimea (numărul de blocuri ocupate)
  - Acces aleatoriu (direct)
  - Risipă de spațiu (problema alocării dinamice a spațiului de stocare)
  - Fișierele nu pot crește în dimensiune
  - Alocarea bazată pe extindere – Veritas File System

# Metode de alocare /3

- **Alocarea înlănțuită**
  - Fiecare fișier este o listă liniară simplu înlănțuită de blocuri disc; fiecare bloc conține o parte din datele fișierului și o referință (un pointer) către următorul bloc
  - Blocurile pot fi "împrăștiate" oriunde pe disc (nu mai sunt într-o zonă contiguă)
  - Blocuri adiționale sunt alocate pe măsură ce conținutul fișierului crește în dimensiune
  - Accesul secvențial nu ridică nici o problemă, în schimb accesul direct nu se poate face eficient
  - Coruperea lanțului de pointeri provoacă probleme majore
  - Tabela de alocare a fișierelor (FAT) – alocare utilizată de sistemul de fișiere FAT (= sistemul de fișiere principal din MS-DOS, Windows 9x și OS/2 < 3.0)

# Metode de alocare /4

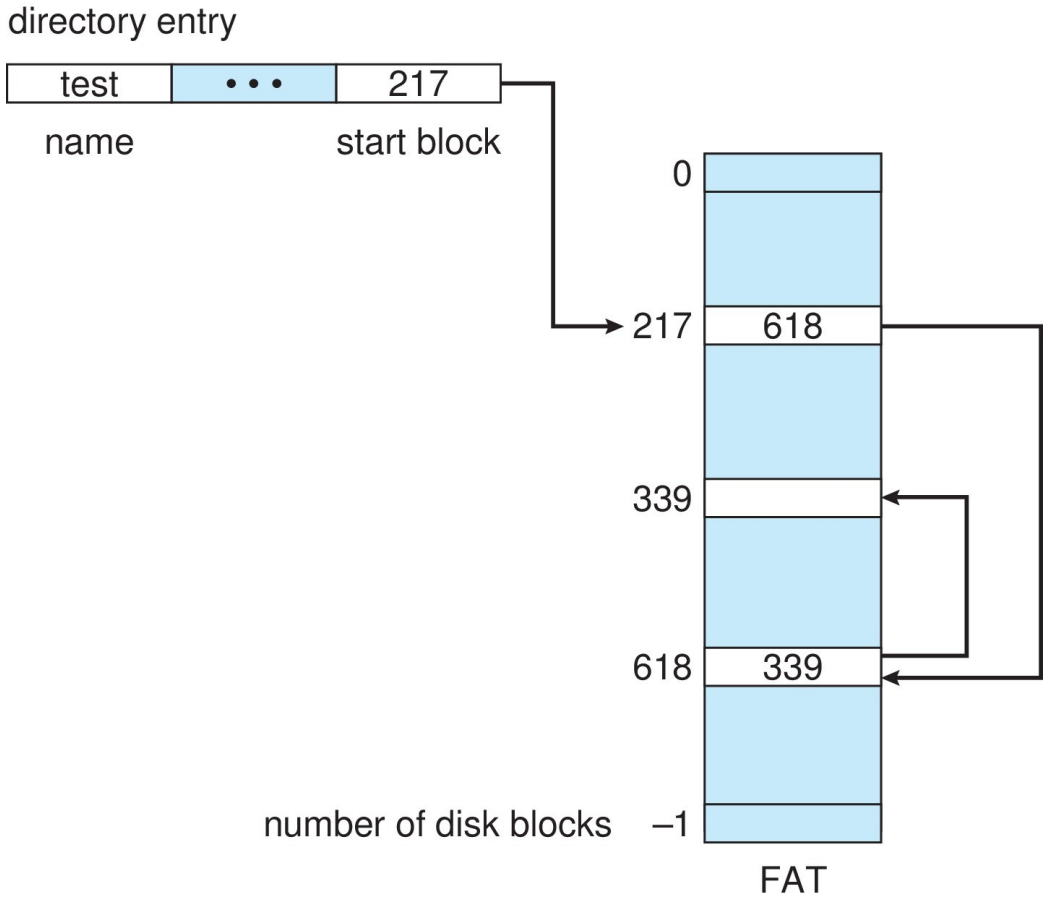
- **Alocarea înlănțuită**



# Metode de alocare /5

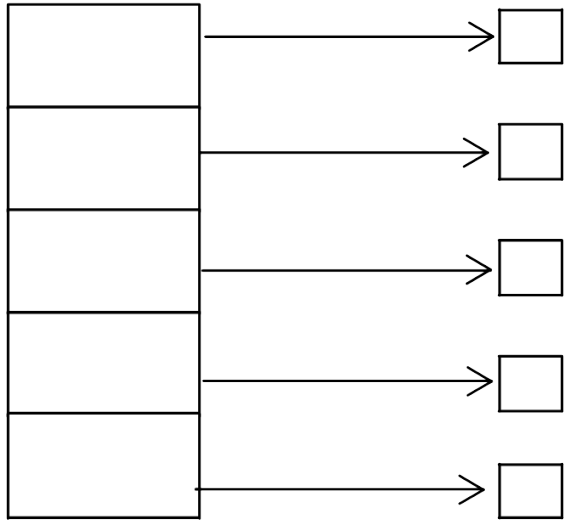
- **Alocarea înlănțuită**

File Allocation Table (FAT)



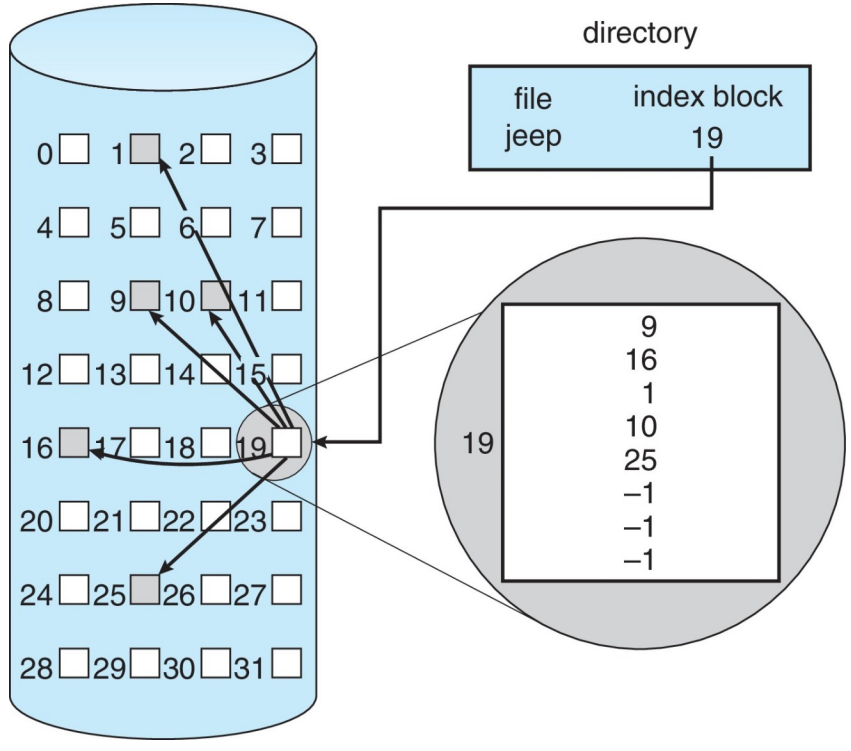
# Metode de alocare /6

- **Alocarea indexată**
  - Pune laolaltă toți pointerii într-un bloc de index (alocarea indexată grupează referințele împreună și le asociază cu un fișier particular)
  - Se utilizează o tabelă de indecși



# Metode de alocare /7

- **Alocarea indexată (cont.)**
  - Acces aleatoriu (direct)
  - Acces dinamic fără fragmentare externă (optimizat pentru situația în care în sistemul de fișiere sunt multe fișiere mici)
  - *overhead*-ul implicat de accesul suplimentar la blocul de index
  - i-noduri – alocare utilizată de familia SO-urilor UNIX, inclusiv Linux și Mac OS X

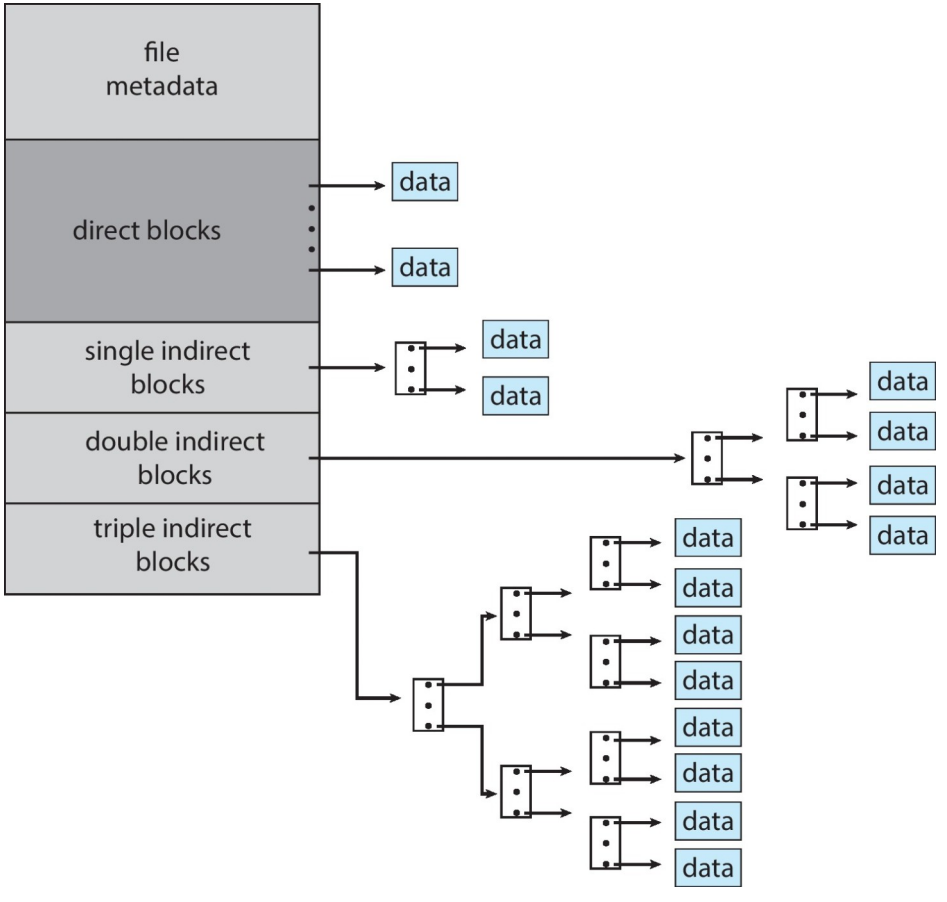


# Metode de alocare /8

- **Alocarea indexată (cont.)**
  - i-noduri – alocare utilizată de Linux, Mac OS X și celelalte UNIX-uri

    FCB-ul (i.e., i-nodul) unui fișier conține toate atributele, inclusiv locația – astfel:
    - 12 adrese directe
    - 1 adresă de simplă indirectare
    - 1 adresă de dublă indirectare
    - 1 adresă de triplă indirectare

    *File pointers* (adresele): 32-bits vs 64 bits (ZFS suportă 128-bits)

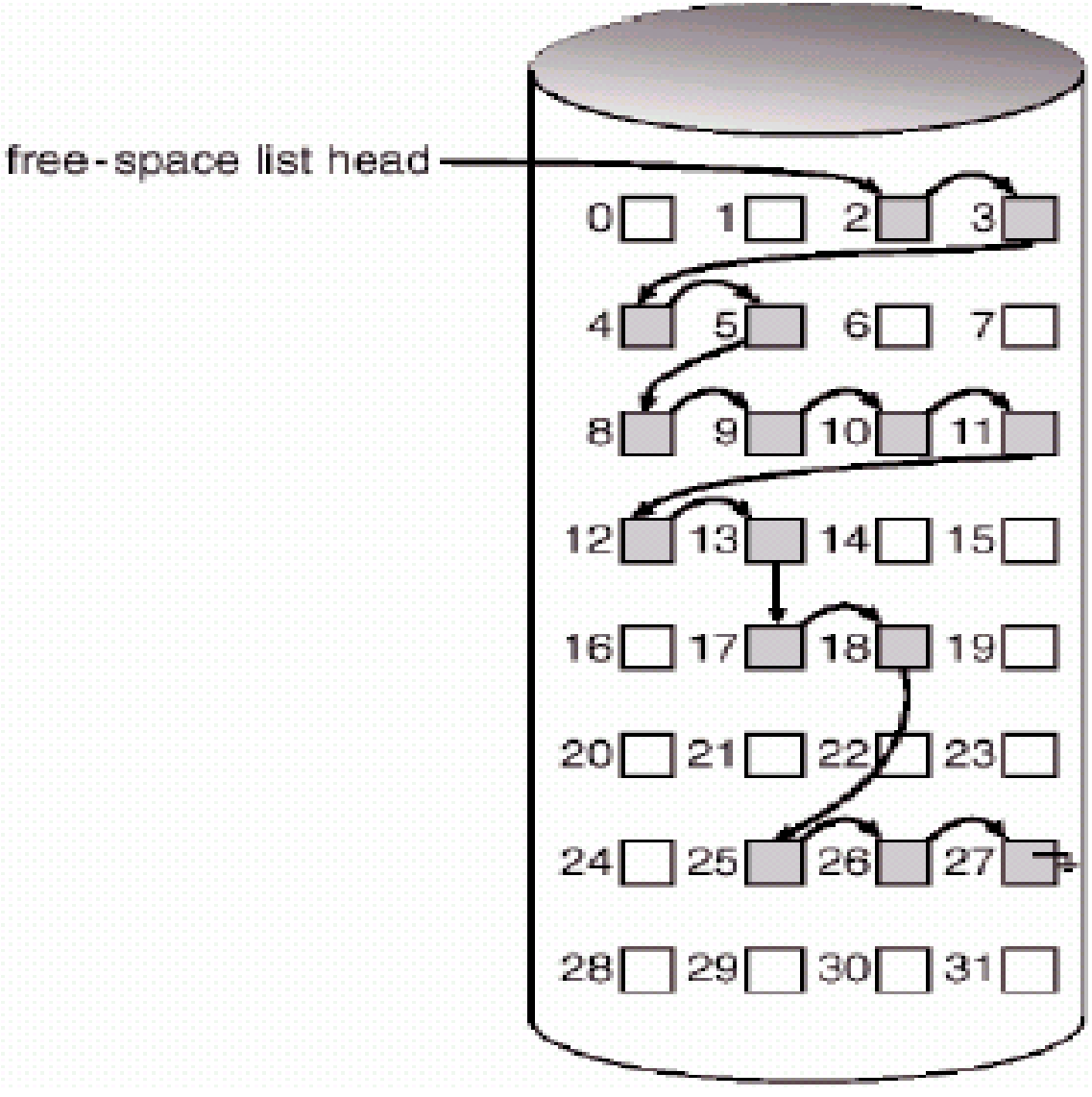


# Gestiunea spațiului liber /1

- **Metode folosite pentru evidența blocurilor libere:**
  - Vectorul de biți
    - Instrucțiuni mașină pentru găsirea primului bit setat
    - Pentru eficiență, este nevoie de păstrarea vectorului de biți în memorie, nu pe disc
    - Exemplu: Mac OS
  - Lista înlănțuită : exemplu pe următorul slide
  - Gruparea : exemplu peste două slide-uri
  - Păstrarea unei liste cu zonele libere contigue – perechi de forma (numărul blocului liber de început, dimensiune)

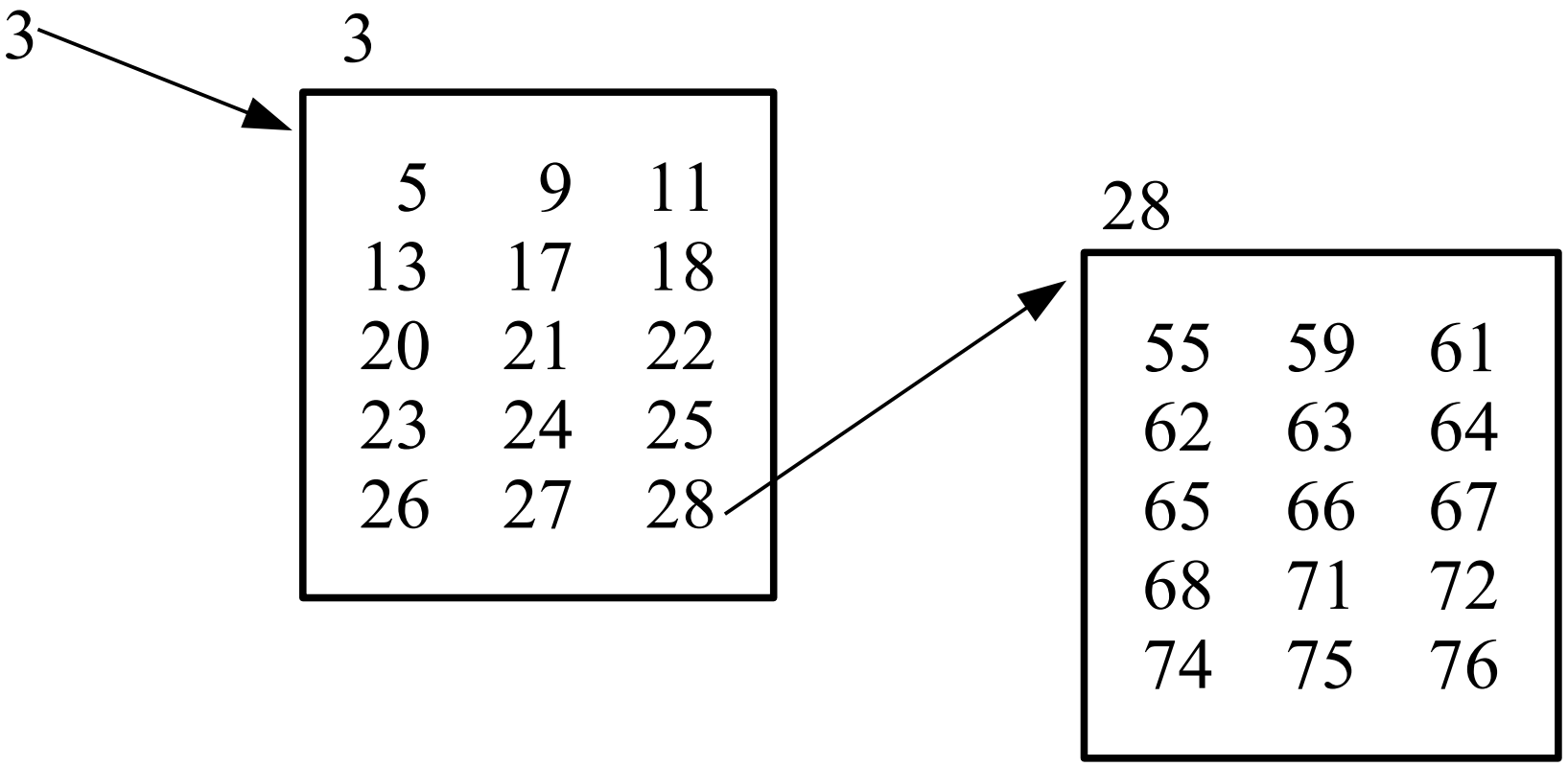
# Gestiunea spațiului liber /2

- Lista înlănțuită a spațiului liber pe disc



# Gestiunea spațiului liber /3

- Gruparea spațiului liber pe disc

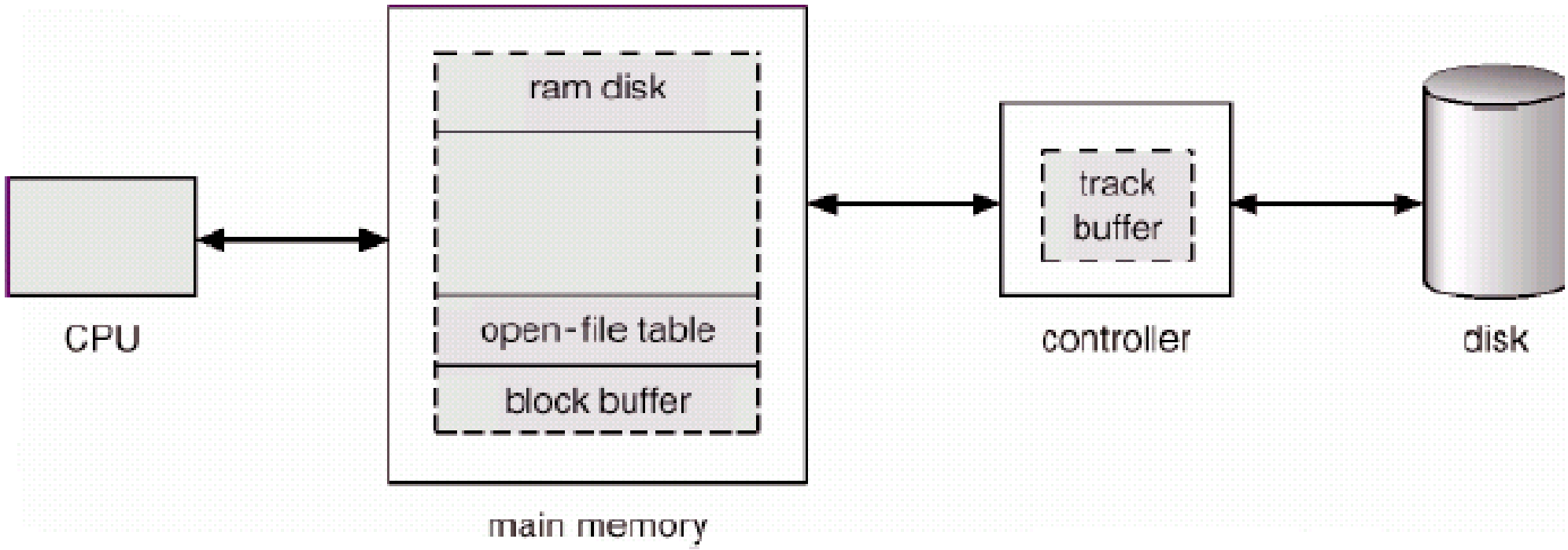


# Eficiența și performanța /1

- **Eficiența** – depinde de:
  - Algoritmii de alocare a discului și pentru directoare
  - Tipurile de date păstrate în intrarea fișierului din director
- **Performanța** – tehnici de îmbunătățire folosite:
  - Disk cache – o secțiune separată a memoriei principale utilizată ca memorie *cache* a blocurilor de disc utilizate frecvent
  - Free-behind și read-ahead – tehnici de optimizare a accesului secvențial
  - Dedicarea unei secțiuni a memoriei principale drept disk virtual (numit uneori și RAM disk)

# Eficiența și performanța /2

- Localizarea *cache*-ului de disc



# Recuperare

- Verificări de consistență – compară datele din structura de directoare cu blocurile de date de pe disc și încearcă să remedieze inconsistențele găsite; dezavantaj: poate dura mult timp. Unealta *consistency checker*: fsck (în UNIX) sau chkdsk (în Windows)
- Folosirea unor utilitare de sistem pentru **backup**-ul datelor de pe disc pe un alt periferic de stocare (disc extern, CD/DVD, bandă magnetică, ș.a.), operație ce trebuie realizată periodic!
  + Recuperarea unor fișiere "pierdute" / unui disc "pierdut" prin **restaurarea** datelor de pe un backup realizat anterior!
- O altă soluție interesantă este furnizată de sistemele de fișiere **jurnalizate**, respectiv, mai nou, de sistemele de fișiere "*next-generation*"

# Sisteme de fişiere jurnalizate

- Sistemele de fișiere **jurnalizate** înregistrează fiecare actualizare a sistemului de fișiere ca pe o **tranzacție**
- Toate tranzacțiile sunt scrise într-un **jurnal** (*log*). O tranzacție este considerată **comisă** atunci când este scrisă în jurnal. Totuși, se poate ca sistemul de fișiere să nu fi fost încă actualizat
- Tranzacțiile din jurnal sunt operate asincron în sistemul de fișiere. După ce sistemul de fișiere este modificat, tranzacția este ștearsă din jurnal
- Dacă sistemul "crapă", toate tranzacțiile rămase în jurnal trebuie să fie operate după repornirea sistemului
- Exemple: NTFS (Windows) sau *ext3fs*/*ext4fs* (Linux)

# Networked File System

- NFS = o implementare a firmei Sun Microsystems (în SO-urile Solaris, SunOS) și o specificație a unui sistem software pentru accesarea fișierelor de la distanță în rețele LAN (sau WAN)
- Stațiile de lucru interconectate prin intermediul rețelei sunt privite ca o mulțime de mașini independente, cu sisteme de fișiere independente, ce permit partajarea fișierelor între aceste sisteme de fișiere, într-o manieră transparentă pentru utilizator (folosind protocoale de montare, RPC, UDP/IP, și modelul client-server)
- Specificațiile NFS sunt independente de hardware, sistem de operare și tehnologia de rețea

# Sisteme de fişiere "*next-generation*"

- Sistemele de fişiere "*next-generation*" încearcă să ofere *toleranţă la tipuri de erori* ce nu sunt prevenite de tehnologiile anterioare: sistemele de fişiere jurnalizate (ce oferă protecţie doar la nivelul structurii *file-system*-ului, nu şi pentru conţinutul fişierelor) sau tehnicile RAID (ce oferă protecţie doar în anumite situaţii, dar nu şi contra "bit rot")
- "Bit-rot" = "the silent corruption of data on disk; one at a time, from time to time, a random bit here or there gets flipped."
  Referinţă: https://arstechnica.com/information-technology/2014/01/bitrot-and-atomic-cows-inside-next-gen-filesystems/
- Exemple: ZFS (Solaris), btrfs (Linux) sau, mai nou, ReFS (Windows) şi APFS (for Apple's devices)
- Protecţia contra "bit-rot": ZFS stochează *checksums* pentru toate datele şi metadatele unui volum, şi astfel poate detecta (şi corecta) coruperea acestora
- Alte facilităţi: operaţii *COW* atomice, *snapshots*, *built-in handling of disk failures and redundancy*, *integration of RAID functionality*, ş.a.

# Bibliografie

- **Bibliografie obligatorie**

  capitolele despre *sisteme de fișiere* din

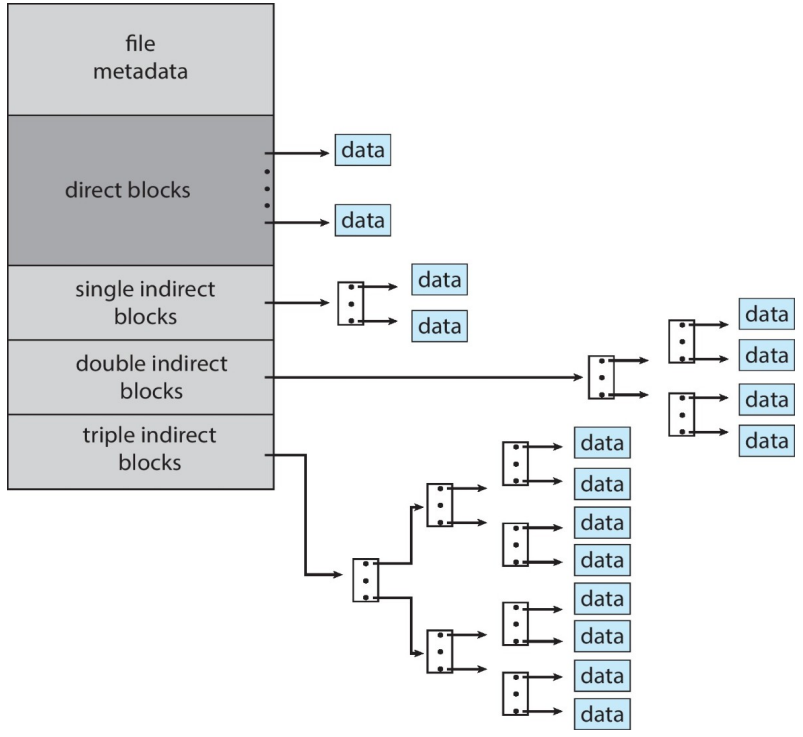
  – Silberschatz : "*Operating System Concepts*"
  (cap.13 și cap.14 din [OSC10])

  sau

  – Tanenbaum : "*Modern Operating Systems*"
  (cap.4 din [MOS4])

# Exerciții de seminar

- Aplicații la: Sisteme de fișiere – structurile de date FCB
  - **Enunț**: Se dă un sistem de fișiere UNIX ce are caracteristicile următoare: sector size = 4KB, file pointers = 32bits.
    Un fișier stocat în acest sistem are lungimea L=1.073.761.780 octeți.
    Care este "amprenta" sa pe disc, i.e. cât spațiu ocupă fișierul pe disc (exprimat în **unități de alocare** a discului, i.e. *blocuri-disc* (sau *sectoare*, alt termen folosit pentru același concept)) ?
  - **Rezolvare**: ?




*Indicație*:

L = 1.073.761.780 = (12*4096) + (1024*4096) + (261.112*4096+3572)

Iar 261.112+1 = 254*1024 + 1017

# Exerciții de seminar

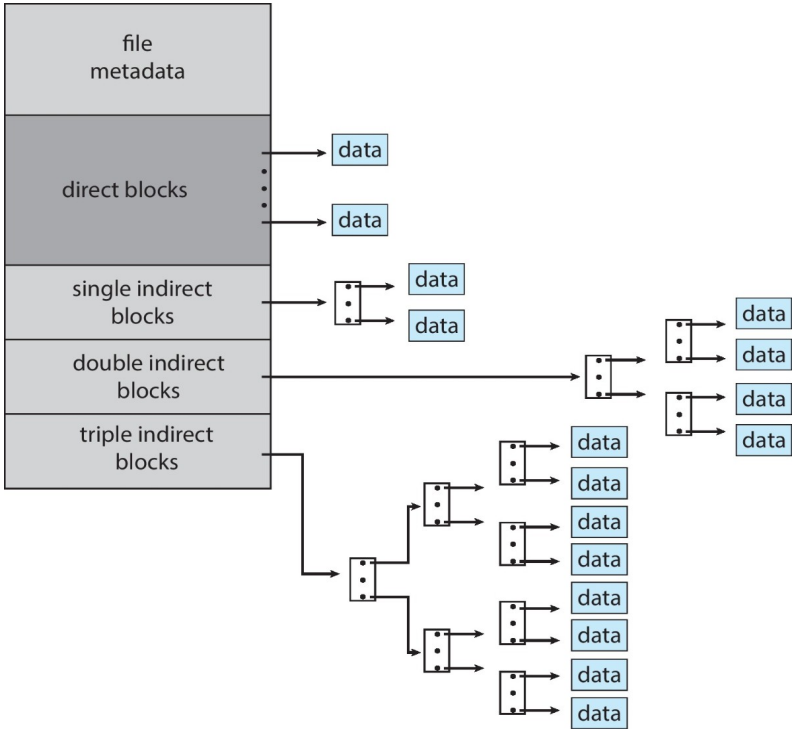
- Aplicații la: Sisteme de fișiere – structurile de date FCB
  - **Enunț**: Se dă un sistem de fișiere UNIX ce are caracteristicile următoare: sector size = 4KB, file pointers = 32bits.
    Un fișier stocat în acest sistem are lungimea L=1.073.761.780 octeți.
    Care este "amprenta" sa pe disc, i.e. cât spațiu ocupă fișierul pe disc?
  - **Rezolvare**:

Un *bloc de date* conține 4096 de octeți (4KB).
Iar un *bloc de indecși* conține 4KB/32bits = 1024 adrese de blocuri-disc (și ocupă pe disc tot 4096 octeți!).

*A se vedea formula de la indicație* => numărul total de blocuri de date pentru a stoca doar conținutul fișierului este: 12 blocuri directe + 1024 de blocuri de simplă indirectare (plus 1 bloc de indecși; primul arbore este complet alocat!) + 261.113 de blocuri de dublă indirectare (al doilea arbore este parțial alocat: 1 bloc de indecși în rădăcina sa, plus doar 255 blocuri de indecși pe nivelul doi). Arborele terțiar este nul.

Total spațiu ocupat pe disc : (12+1024+261.113 blocuri de date) + (1+1+255 blocuri de indecși) = 262.406 de blocuri.

La care se mai adaugă spațiul necesar pentru FCB: sizeof (i-nod).



# Sumar

- Conceptele de fișier și de sistem de fișiere
- Interfața sistemului de fișiere
- Implementarea sistemului de fișiere

Întrebări ?