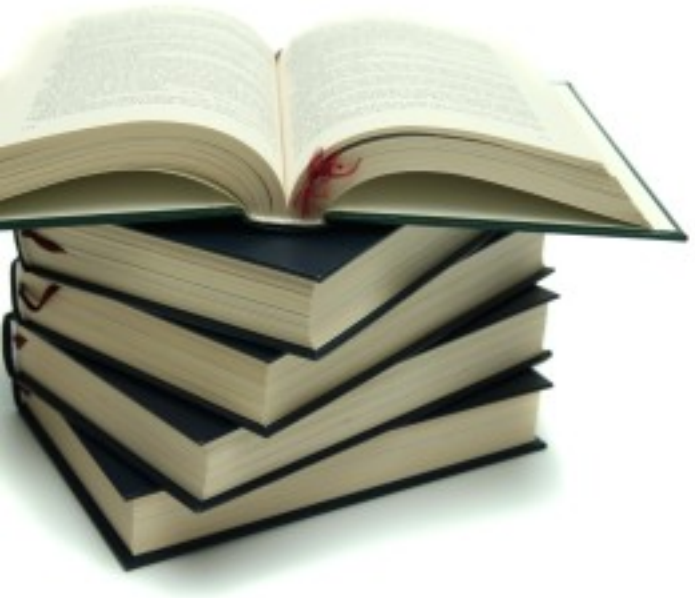
page crunch

Cursul 10



10

Implementarea sistemelor de fișiere

11 mai 2010

- OSC
  - Capitolul 11 – File-System Implementation
    - Secțiunile 11.1 - 11.8
- MOS
  - Capitolul 6 – File Systems
    - Secțiunile 6.3, 6.4

Cuprins

- Structura sistemului de fișiere
- Alocarea spațiului pe disc
- Implementarea directoarelor și link-urilor
- Gestiunea spațiului liber
- Considerente de performanță
- Consistența sistemelor de fișiere
- Sisteme de fișire jurnalizate
- Sistemul de fișiere MINIX

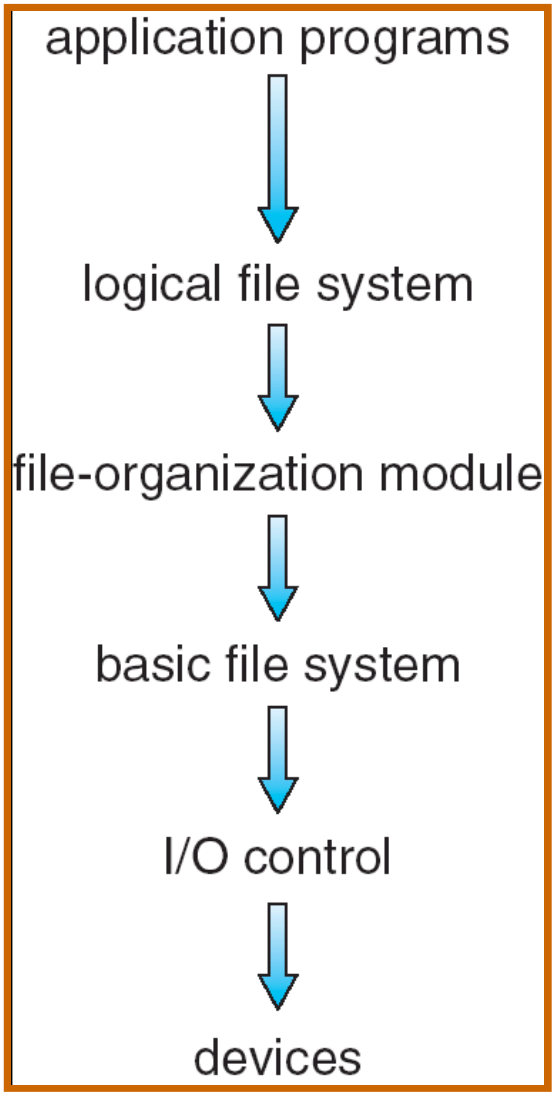
Sistemul de fișiere

- Perspectiva utilizatorului
  - ierarhie de directoare și fișiere
  - nume, drepturi de acces, utilizator, grup, timpi de acces
- Perspectiva sistemului de operare
  - structuri și algoritmi de stocare eficientă și scalabilă a informației
  - suport persistent

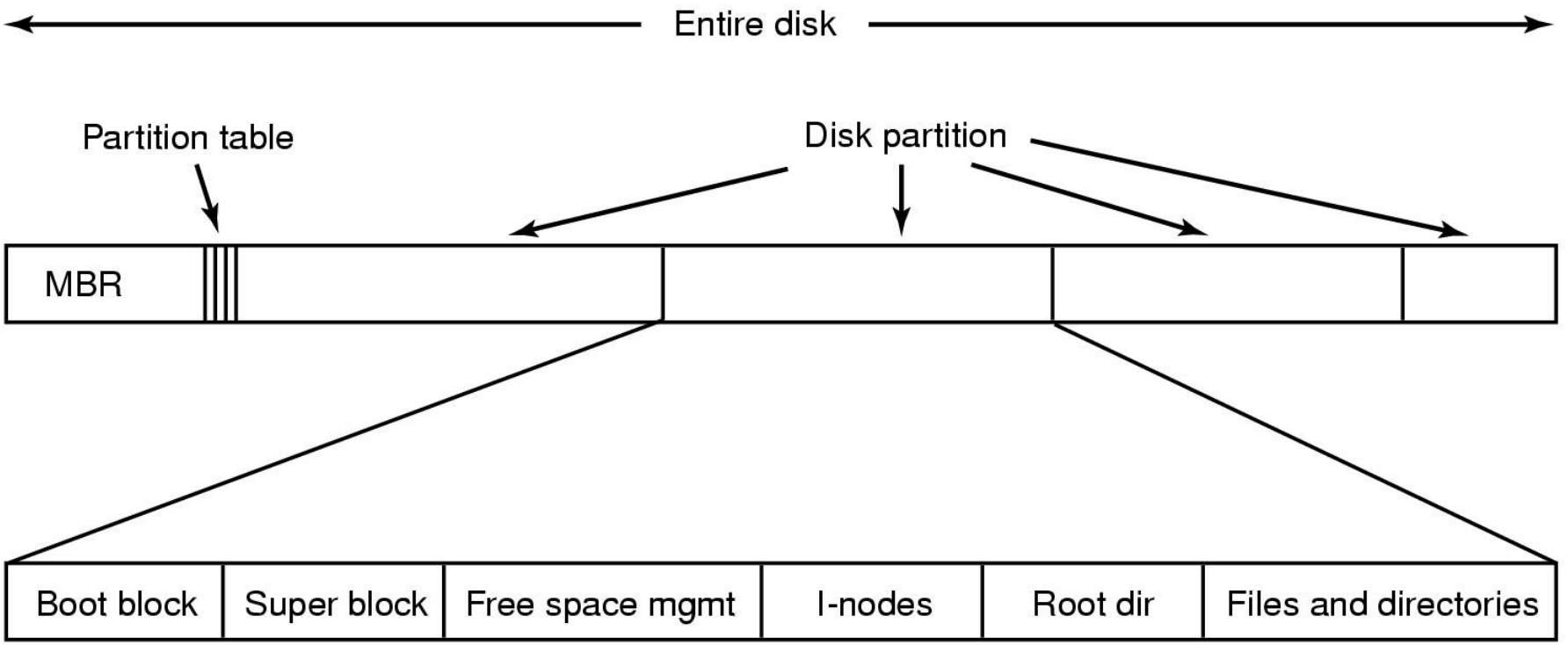
Implementarea unui sistem de fișiere

- Mod de organizare a datelor pe un suport persistent
- Două roluri
  - oferirea unei interfețe facile utilizatorului
    - fișiere, directoare, ierarhie
    - atribute (drepturi, nume, timpi)
  - algoritmi de alocare și organizare a informației pe suportul fizic
    - translatare cereri de la utilizator în sectoare

Niveluri de utilizare/implementare



Structura sistemului de fișiere

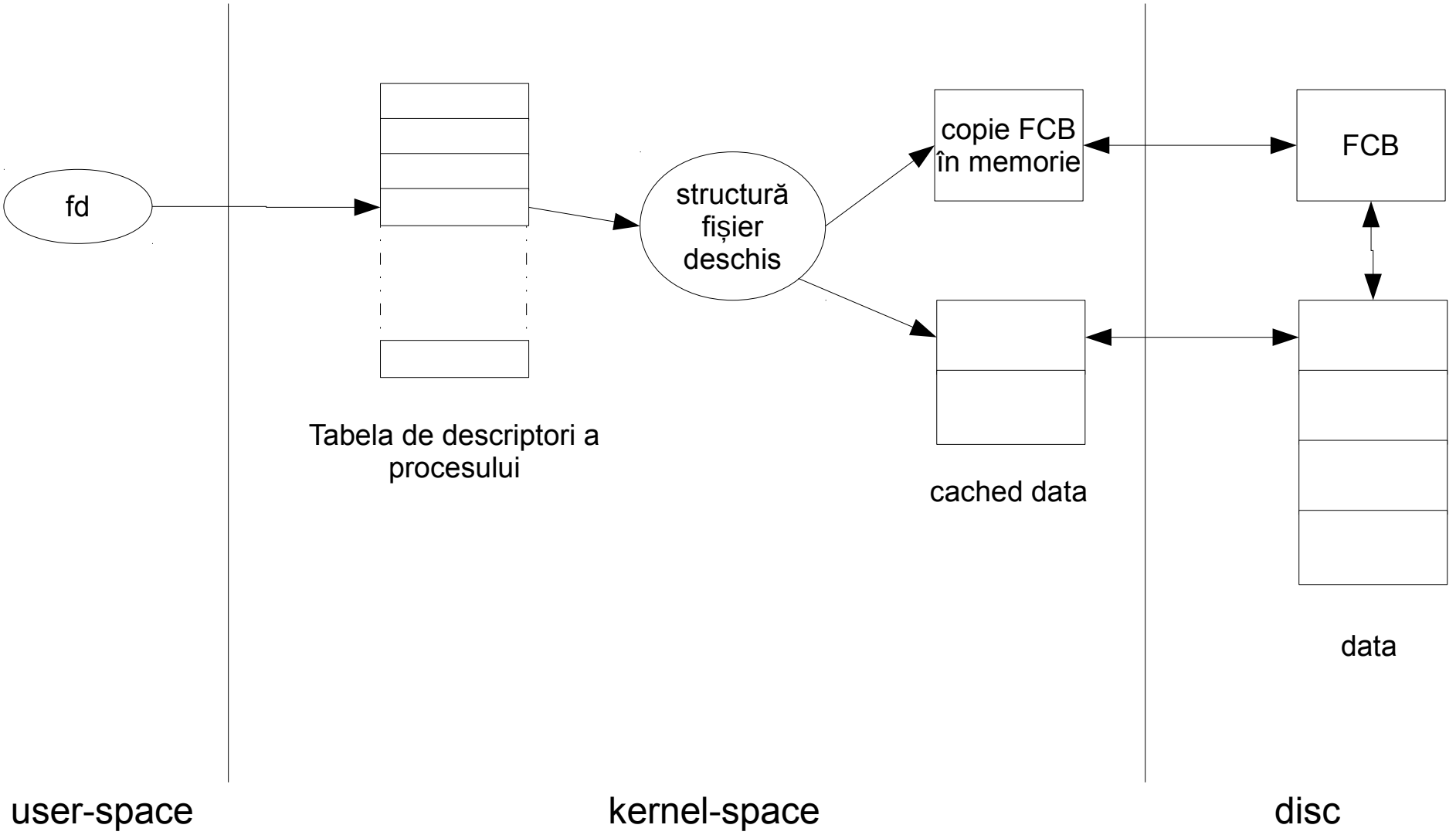


Fișierul

- Entitatea de bază a unui sistem de fișiere
- Abstractizează datele/informațiile
- În Unix, totul e un fișier; dacă nu e fișier, atunci e proces

- Un fișier este descris de date și metadate
  - date – blocurile de date efective
  - metadate (informație despre date) – structură specializată
    - File Control Block (i-node în Unix)

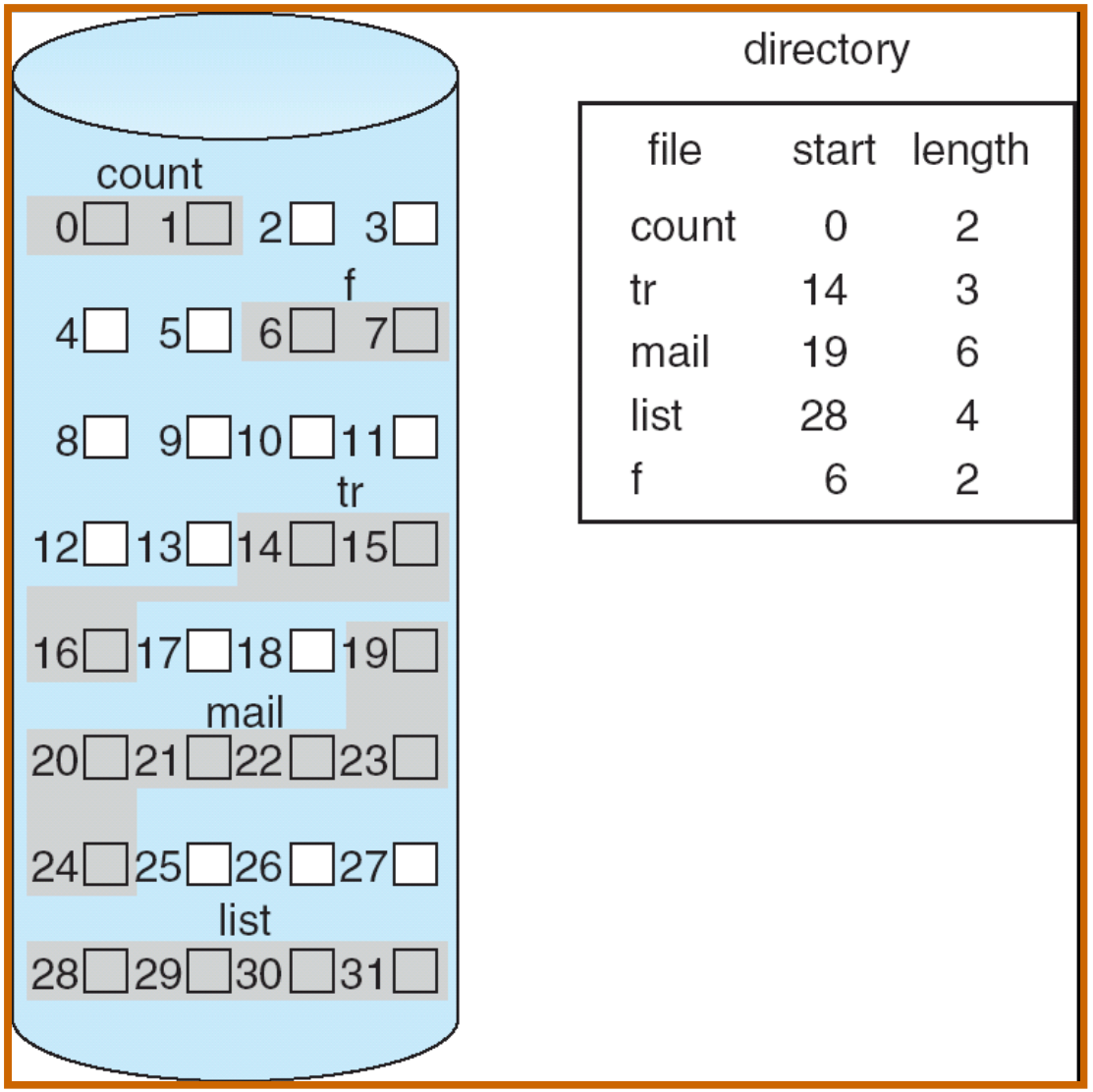
De la descriptor la FCB



Alocarea spațiului pe disc

- Mecanismul prin care SO gestionează spațiul liber
  - alocare contiguă
  - alocare cu liste
  - alocare indexată
- Alocare la nivel de bloc
  - multiplu de sector
  - submultiplu de pagină
  - (512, 1024, 2048, 4096 octeți)

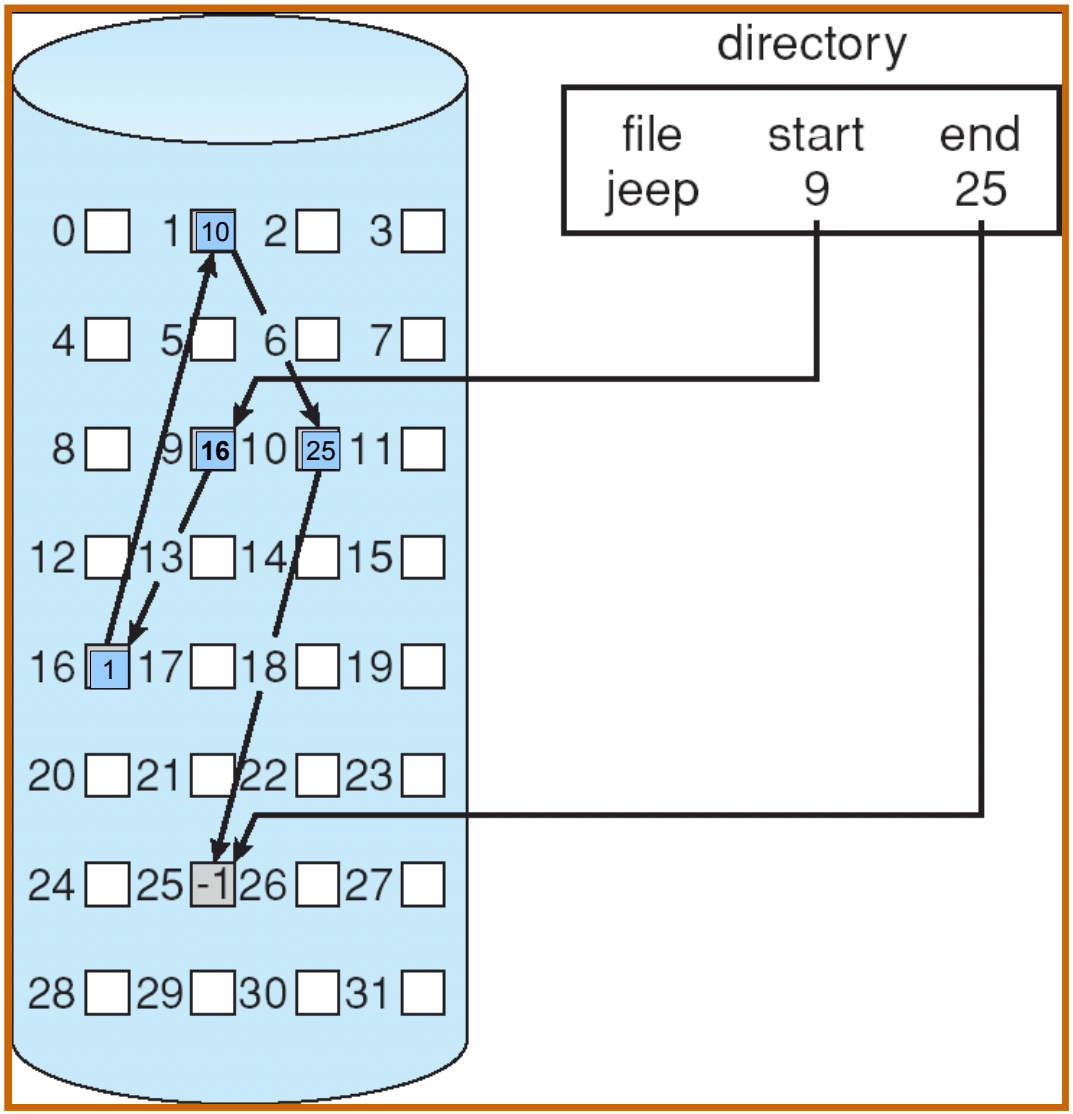
Alocare contiguă



Alocare contiguă (2)

- Stocarea fișierelor ca o secvență continuă pe disc
- Avantaje
  - identificare simplă a blocurilor
    - blocul de start
    - dimensiunea fișierului în blocuri
  - viteză mare la citire
- Dezavantaje
  - fragmentare externă
  - trebuie specificată dimensiunea fișierului la crearea acestuia

Alocare cu liste



Alocare cu liste (2)

- Un bloc conține pointer către următorul bloc
- Avantaje
  - nu mai există fragmentare externă
  - este necesar doar primul bloc pentru a localiza fișierul pe disc
- Dezavantaje
  - timp de acces ridicat pentru ultimele blocuri

Tabela de alocare a fișierelor

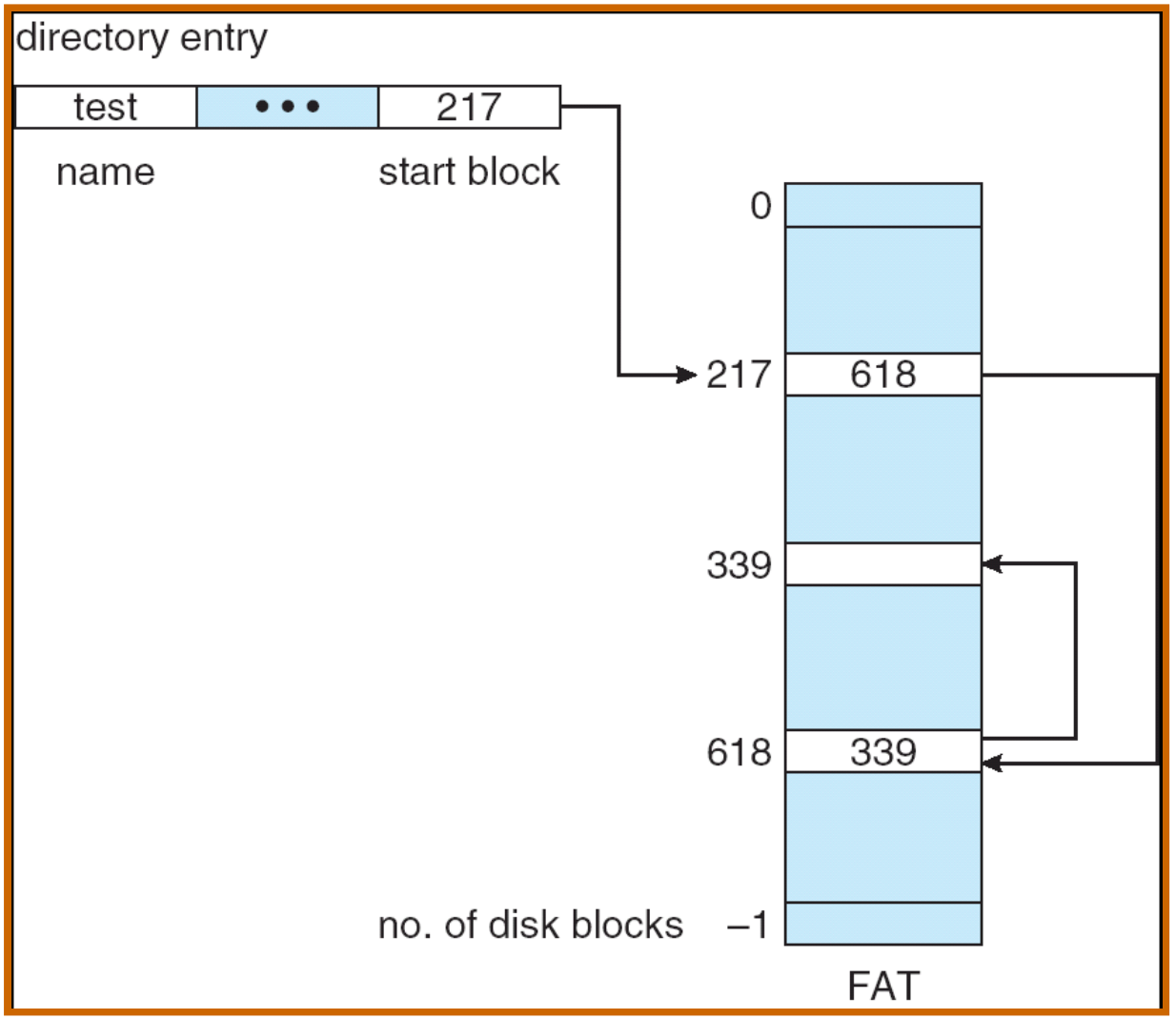
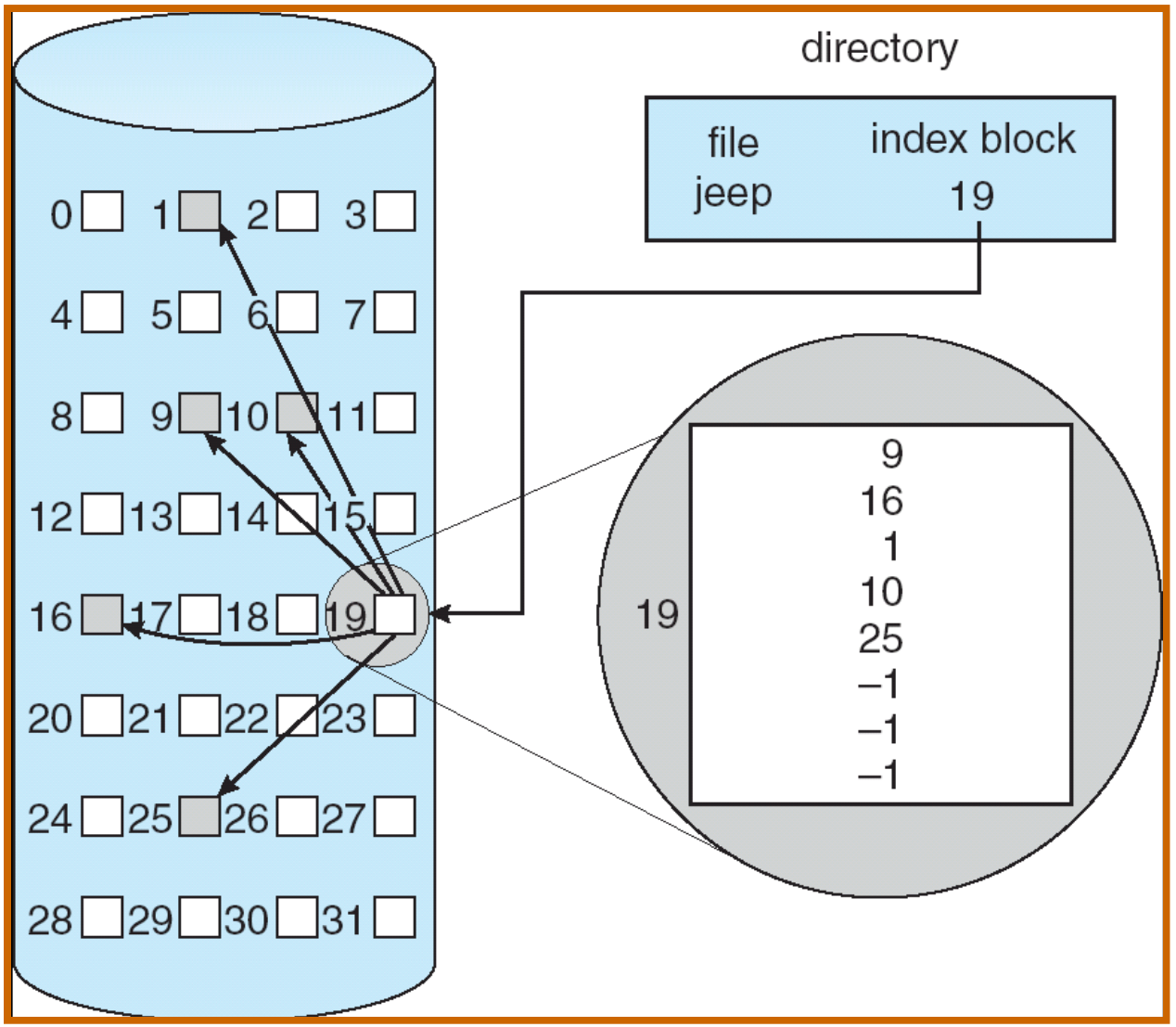


Tabela de alocare a fișierelor (2)

- O zonă separată (FAT) conține pointerii către blocurile fișierului
- O tabelă FAT de dimesiune redusă poate fi încarcată în memorie
- Altfel, la deschiderea fișierului se pot citi din FAT o parte a blocurilor de date ale fișierului în memorie
  - se reduce timpul de acces

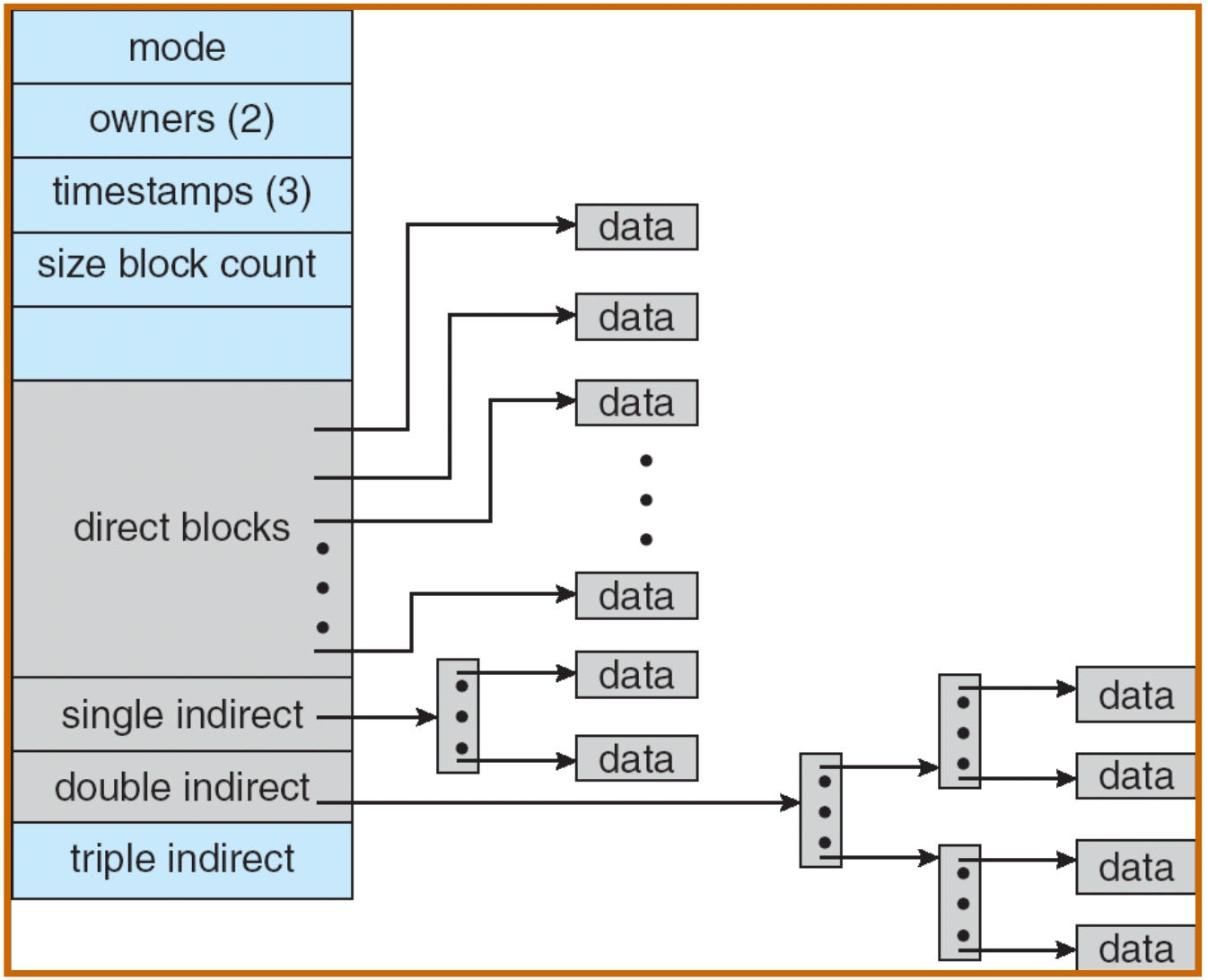
Alocare indexată



Alocare indexată (2)

- Un index block (i-node) pentru fiecare fișier
- i-node-ul mapează blocurile logice ale fișierului în blocuri fizice
- Avantaje
  - nu există fragmentare externă
  - timp de acces bun
- Dezavantaje
  - numărul limitat de intrări în tabela de indexare
  - ultimele intrări conțin pointeri către alte tabele de indecși
    - latența crește la accesarea blocurile „distante”

i-node


mode
owners (2)
timestamps (3)
size block count
direct blocks
single indirect
double indirect
triple indirect
data
data
data
data
data
data
data
data
data
data

Directory entry

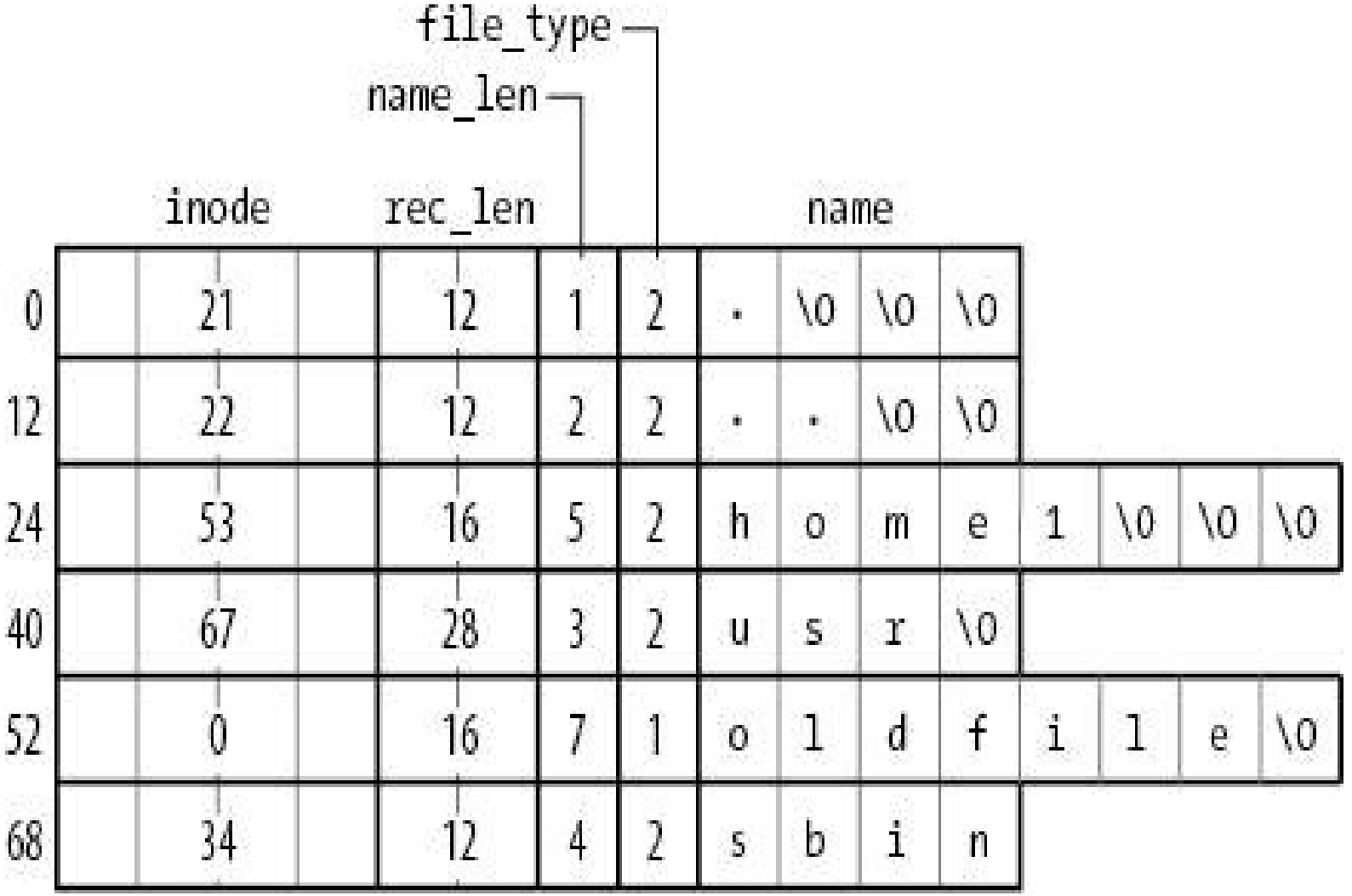
- Directory entry (dentry) - numele unui fișiere
  - struct dirent (POSIX)
  - WIN32_FIND_DATA (Win32)
- Pe disc, structura dentry conține
  - numele fișierului
  - un identificator al FCB (i-node number)
- În sistemele Unix numele (dentry) este separat de FCB (i-node)

Directorul

- Un tip de fișier care conține structuri dentry
- Dispune de un FCB (i-node, intrare în tabela FAT)
- Intrările . și .. sunt speciale
- Dimensiunea este dată structurile dentry conținute

```
$ ls -ld /
drwxr-xr-x  24 root root  4096 2008-05-06 17:28 /
$ ls -ld /etc
drwxr-xr-x 126 root root 12288 2008-05-11 00:10 /etc
```

Implementare de director în ext2



Link-uri

- Hard links
  - două structuri dentry care referă același i-node
  - nu pot fi folosite între sisteme de fișiere diferite; de ce? (Hint: ce conține un dentry?)
- Sym[bolic] links (soft links)
  - i-node specializat
  - i-node-ul conține numele fișierului referit
    - pe ext2 numele se stochează în inode dacă are sub 60 de caractere

Link-uri (2)

```
$ touch a
$ ln a b
$ ls -l -i
71682 -rw-r--r-- 2 razvan razvan 0 2008-05-11 00:26 a
71682 -rw-r--r-- 2 razvan razvan 0 2008-05-11 00:26 b
$ ln -s a c
$ ls -l -i
71682 -rw-r--r-- 2 razvan razvan 0 2008-05-11 00:26 a
71682 -rw-r--r-- 2 razvan razvan 0 2008-05-11 00:26 b
71683 lrwxrwxrwx 1 razvan razvan 1 2008-05-11 00:27 c -> a
$ rm b
$ ls -l -i
71682 -rw-r--r-- 1 razvan razvan 0 2008-05-11 00:26 a
71683 lrwxrwxrwx 1 razvan razvan 1 2008-05-11 00:27 c -> a
```

Gestiunea spațiului liber

- Vector de biți
  - 1 blocul este liber
  - 0 blocul este ocupat
- Liste înlănțuite
  - primul bloc liber conține un pointer către al doilea, etc.
  - capătul listei este ținut în memorie
  - versiune îmbunătățită - se mențin tabele de blocuri libere
  - optimizare - se mențin zone contigue în tabele

Gestiunea spațiului liber (2)

- Spațiul necesar pentru lista/vectorul de biți de blocuri libere pentru:
  - un disc de 16GB
  - blocuri de 1KB
  - adresa unui bloc pe 32 de biți
- Răspuns
  - listă: 2^24 : (1024 : 4 -1) = 2^24 : 255 = 65794 de blocuri
  - vector: 2^24 : (8 * 1024) = 2^11=2048 de blocuri

Considerente de performanță

- Sporirea performanțele sistemelor de fișiere
  - caching
  - free-behind/read-ahead
  - reducerea timpului de seek (vezi și planificarea operațiilor de I/O)

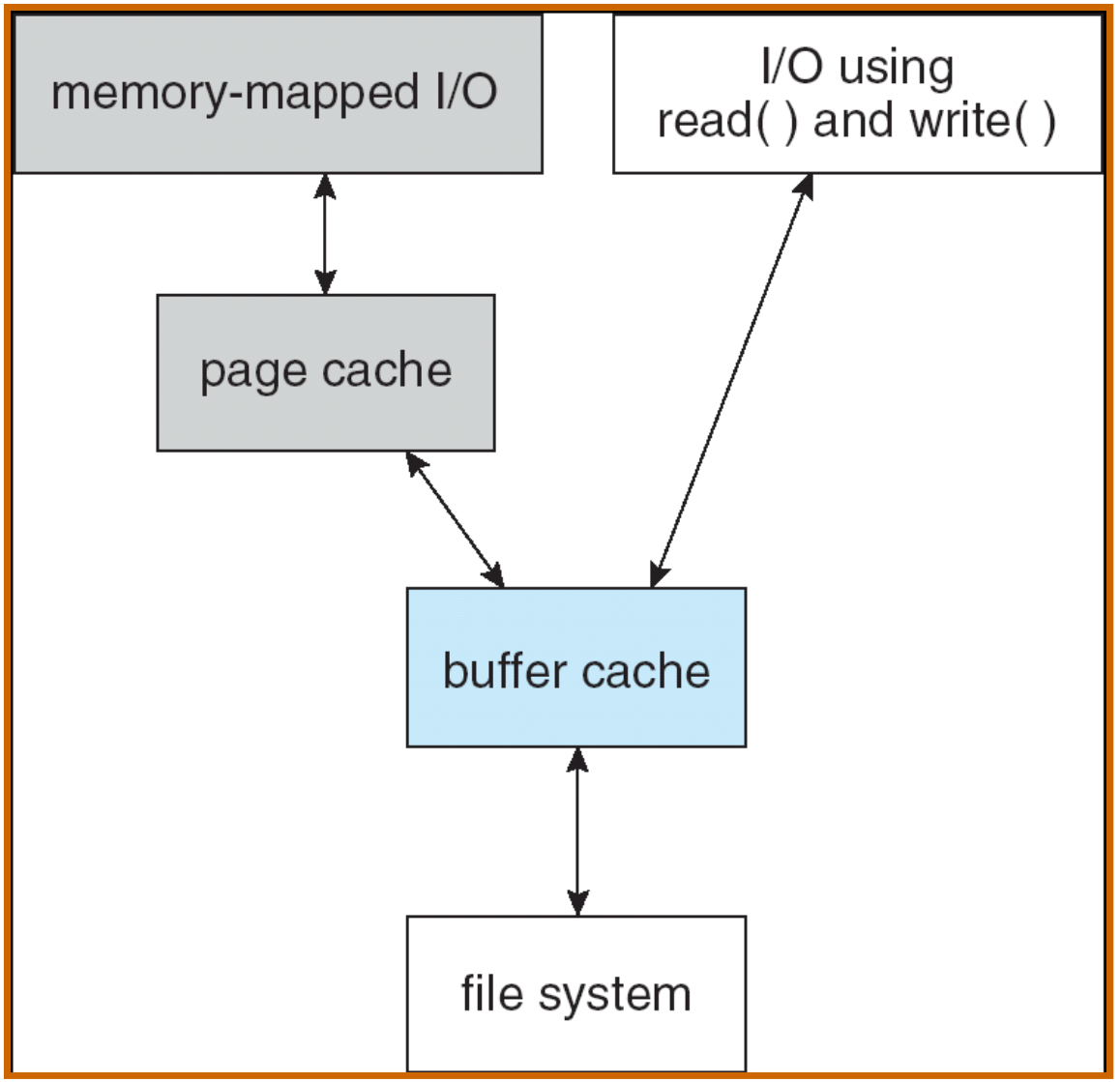
Page cache

- Folosește pagini în loc de blocuri
- Folosit de memory-mapped files
- Apelurile read/write folosesc buffer cache
- Pentru flush (scrierea pe disc a paginilor modificate)
  - msync(2) pentru page cache
  - fsync(2) pentru buffer cache
  - este posibil ca discul să aibă un cache intern la scriere

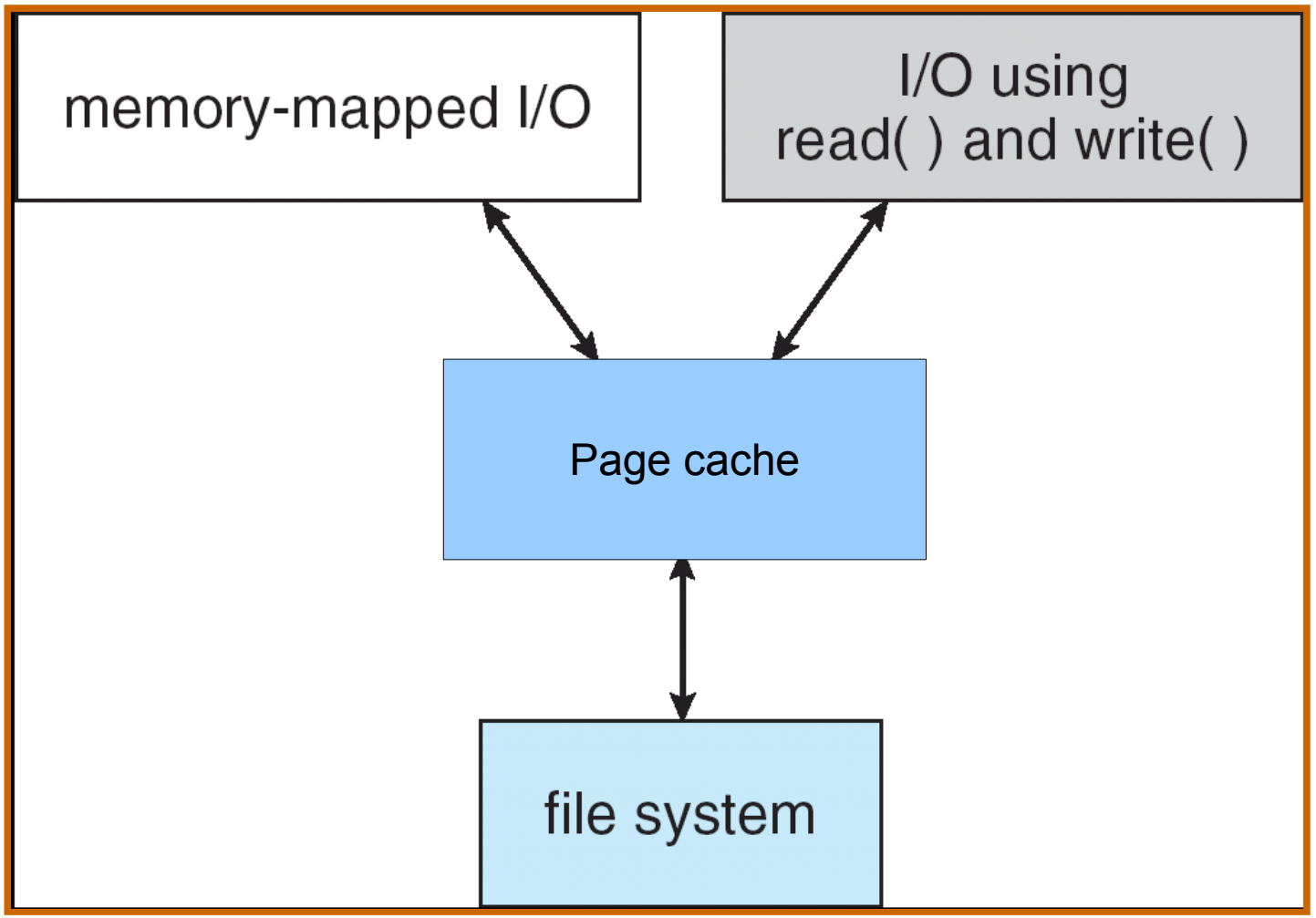
Buffer cache

- Menține în memorie blocurile recent citite de pe disc
  - dimensiunea relativ mare
    - folosire tabele hash
  - algoritmi de înlocuire: FIFO, LRU, second chance
  - LRU este mult mai ușor (eficient) de implementat
    - referirea unei pagini se face mult mai rar

Non-unified buffer cache



Unified buffer/page cache



Alte cache-uri

- Cache-ul pentru i-noduri: icache
  - atributele fișierelor se păstrează în inode-uri
  - acces rapid pentru comenzi de tipul ls
- Cache-ul pentru rezolvarea numelor în inode-uri: dcache
  - rezolvarea de nume este costisitoare
    - posibilă trecere prin mai multe blocuri
  - folosire tabele hash

Backup-uri

- Backup-uri
  - Sunt toate informațiile salvate?
    - Da => full backup
    - Nu, numai cele care nu au fost salvate de la ultimul backup => incremental backup
  - Sunt salvate toate blocurile sistemului de fișiere?
    - Da => physical dump; nu se pot face backup-uri incrementale
    - Nu => logical dump

Consistența unui sistem de fișiere

- Incosistențe
  - La blocurile de date
    - nu se pot detecta
    - nu sunt grave pentru consistența întregului sistem de fișiere
  - La blocurile auxiliare (metadate)
    - se pot detecta
    - pot avea efecte grave asupra întregului sistem de fișiere
- La boot-are, dacă sistemul de fișiere nu a fost demontat, se verifică consistența acestuia

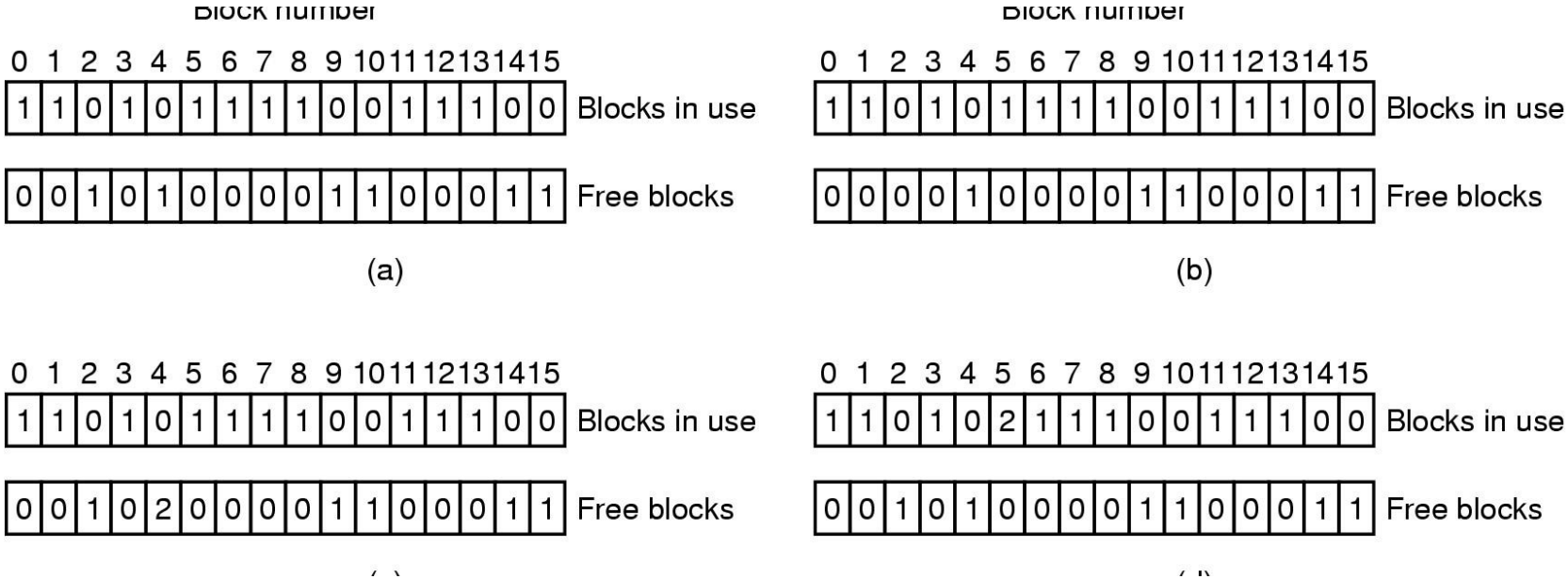
fsck/scandisk

- Teste de consistență pentru blocuri
  - două tabele (blocuri libere și blocuri utilizate)
  - se parcurg i-node-urile și tabelele de indecși și se incrementează corespunzător în tabela de blocuri utilizate
  - se parcurge lista de blocuri libere și se incrementează corespunzător în tabela de blocuri libere
- Test de consistență pentru fișiere
  - se creează o tabelă cu numărul de referiri ale fișierelor
  - se parcurg intrările de directoare și se incrementează corespunzător în tabela de referiri ale fișierelor

fsck/scandisk (2)

- Se repară inconsistențele
  - blocuri
    - un bloc să fie contorizat o singură dată numai într-una din tabele
    - dacă nu, se repară inconsistența
  - fișiere
    - contorul de utilizare în i-node-ul fișierului nu corespunde contorului din tabelă -> se forțează în i-node valoarea din tabelă

fsck/scandisk (3)



(a) SF consistent

(b) bloc pierdut

(c) bloc duplicat in lista de blocuri libere

(d) bloc de date duplicat

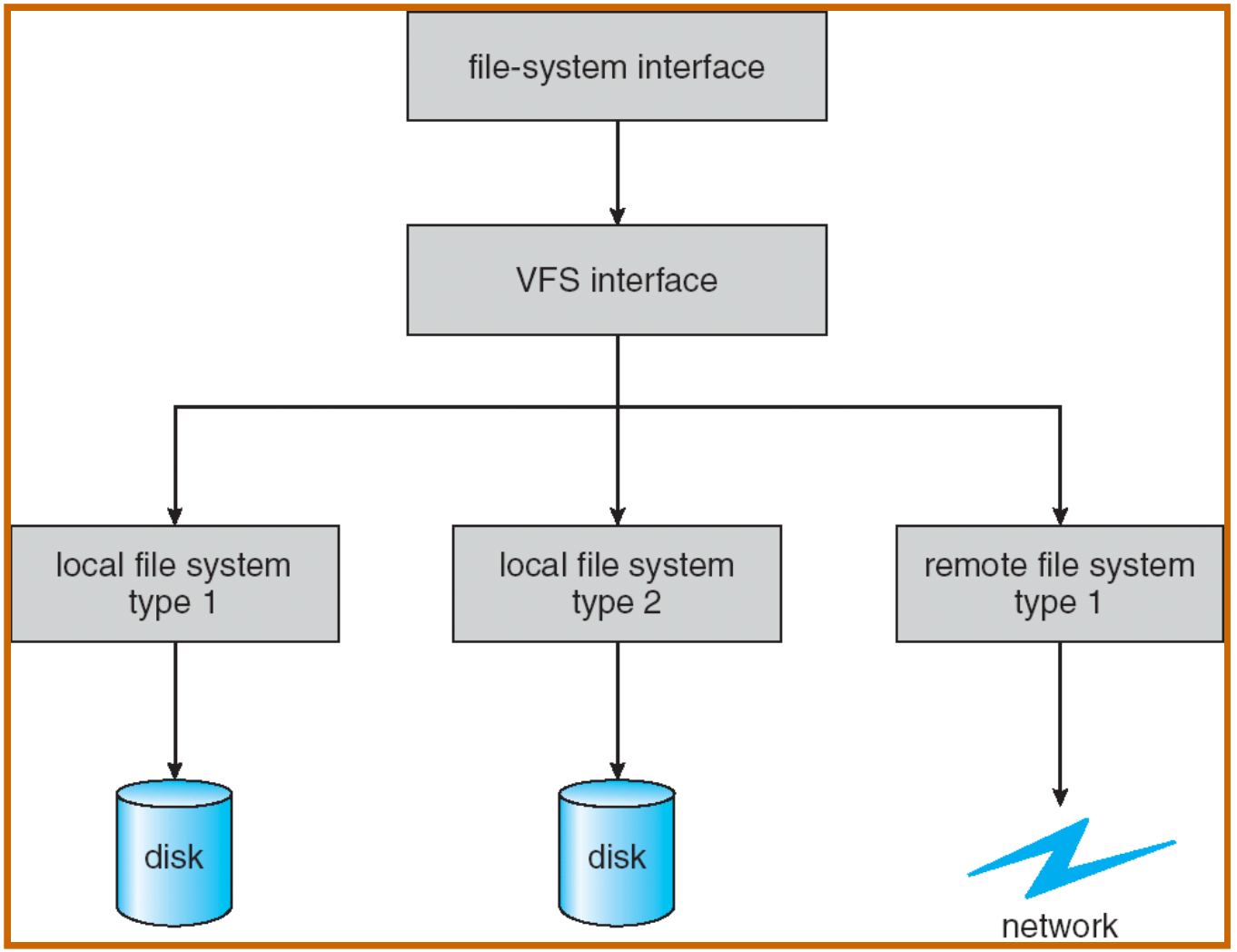
Sisteme de fișiere jurnalizate

- Journaling file systems (ext3, ReiserFS, JFS)
- Operațiile sunt scrise într-un jurnal
  - înainte de efectuare sunt sumarizate în jurnal în forma unor tranzacții
  - după încheiere tranzacția este ștearsă din jurnal
- De obicei se folosește “metadata-only journaling”

Sisteme de fișiere jurnalizate (2)

- La montarea sistemului de fișiere se consultă jurnalul și se efectuează operațiile din tranzacții (*replay/rollback*)
- Avantaje
  - reduce timpul de verificare a consistenței
  - reduce posibilitatea de a pierde date în urma unui crash
- Dezavantaje
  - încetinește sistemul de fișiere

VFS


file-system interface
VFS interface
local file system type 1
local file system type 2
remote file system type 1
disk
disk
network

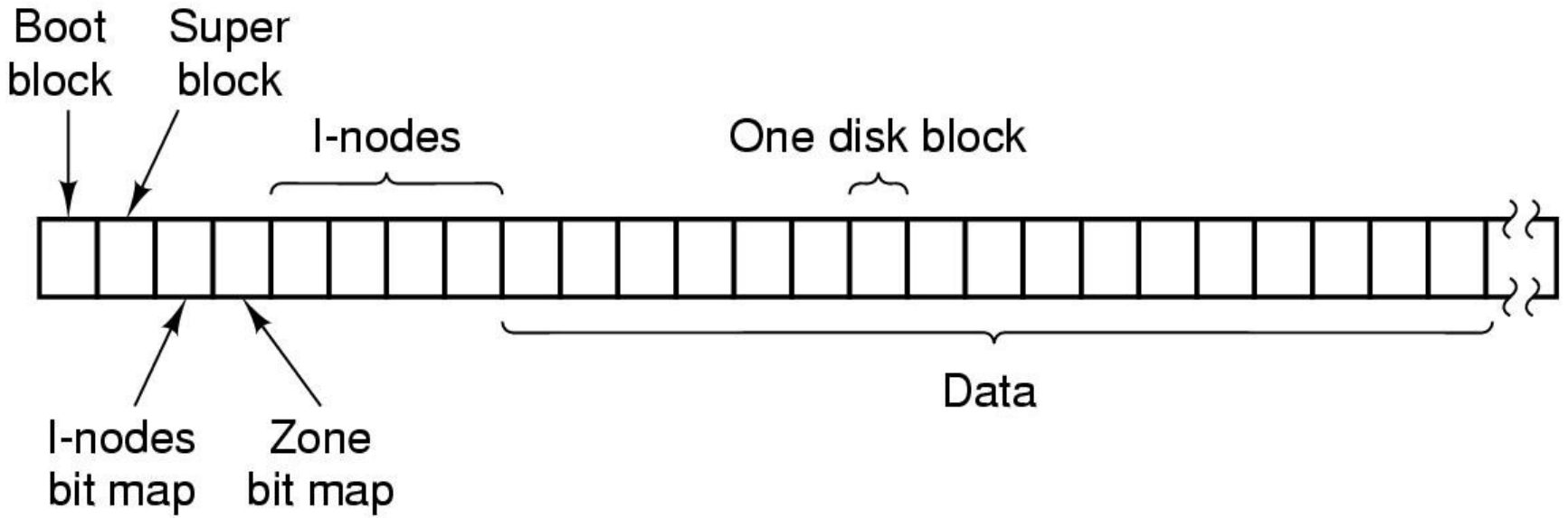
VFS (2)

- Virtual File System
- Operațiile generice ale SF sunt separate de implementare
- vnode = entitate ce identifică în mod unic un fișier din sistem
  - identifică tipul sistemului de fișiere și activează operațiile specifice
- VFS în Linux, IFS în Windows

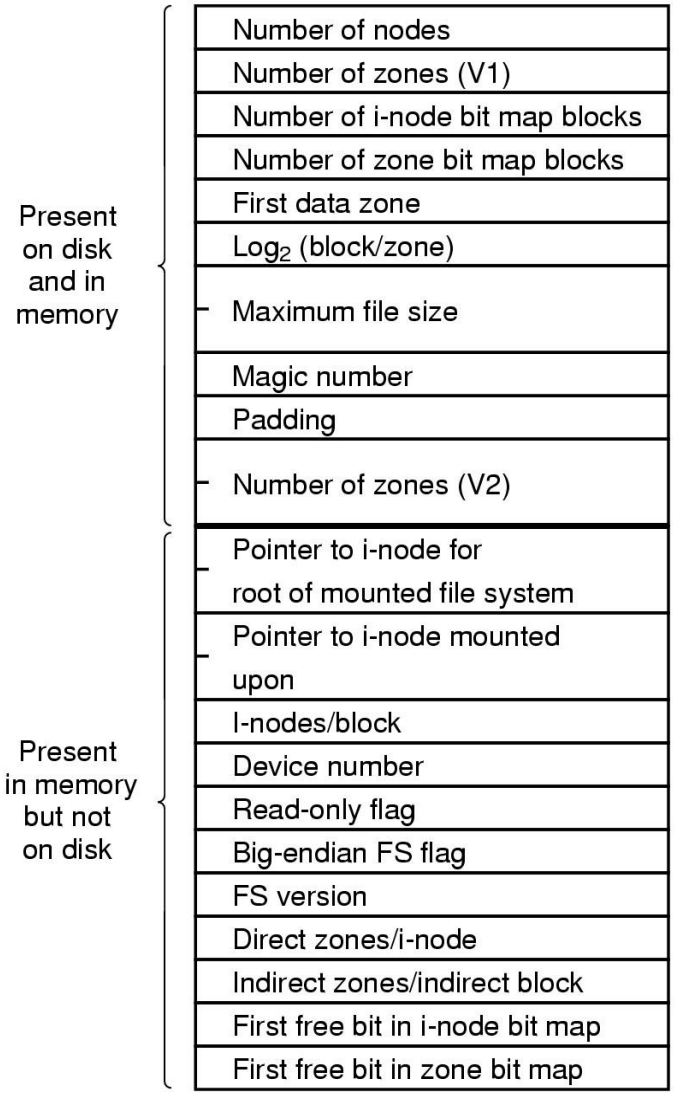
Sistemul de fișiere MINIX

- Variantă simplificată a unui sistem de fișiere UNIX
- Folosit inițial de Linux
  - înlocuit de ext (apoi de ext2, ext3)
- Simplu (ca și FAT)
  - folosit în sisteme embedded

Structura MINIX FS



Superblocul



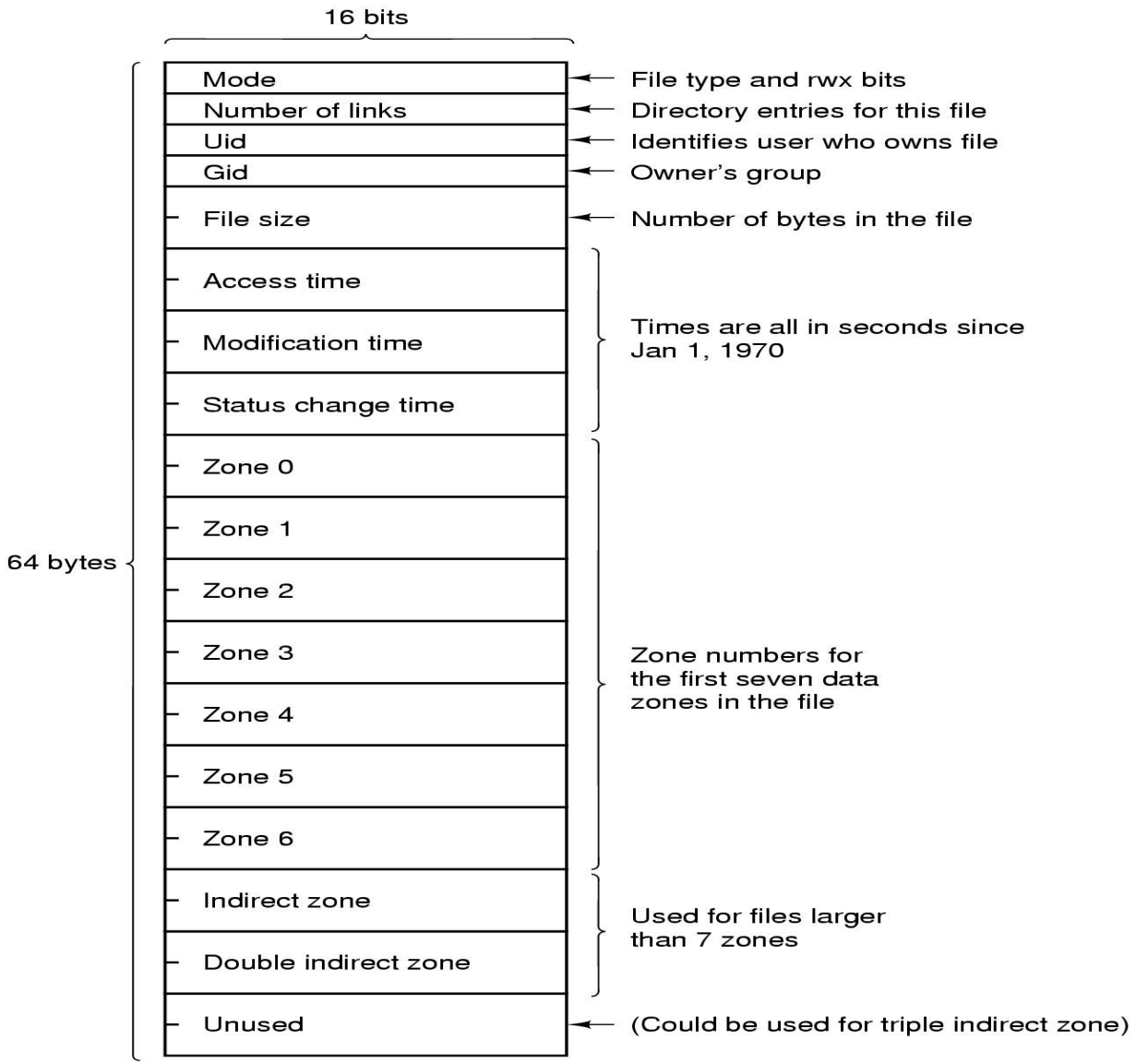
Inode bitmap

- Vector de biți pentru a identifica inode-urile libere și cele ocupate
- Intrarea 0 nu este folosită din motive dictate de implementare
  - funcția care caută un inode liber întoarce 0 dacă nu mai există inode-uri libere

Zone bitmap

- O zonă = un număr de blocuri (puteri ale lui 2) folosite pentru alocarea datelor unui fișier
- Menține zonele libere cu un vector de biți
- Zone pentru a ține blocurile aceluiași fișier la un loc
  - reducerea timpului de căutare (seek)
- Probleme de securitate
  - date reziduale în zonă

Inode-ul


16 bits
64 bytes
Mode
Number of links
Uid
Gid
File size
Access time
Modification time
Status change time
Zone 0
Zone 1
Zone 2
Zone 3
Zone 4
Zone 5
Zone 6
Indirect zone
Double indirect zone
Unused
File type and rwx bits
Directory entries for this file
Identifies user who owns file
Owner's group
Number of bytes in the file
Times are all in seconds since Jan 1, 1970
Zone numbers for the first seven data zones in the file
Used for files larger than 7 zones
(Could be used for triple indirect zone)

Structura directoarelor

- Un director conține un vector de intrări de director (dentry)
- O intrare de director este formată din
  - numarul inode-ului
  - numele fișierului (16 / 32 de octeti V1 / V2)

Cuvinte cheie

- sistem de fișiere
- date/metadate
- FCB
- FAT
- i-node
- dentry
- hard link
- symbolic link
- page cache
- buffer cache
- icache/dcache
- consistența SF
- fsck/scandisk
- jurnalizare
- VFS
- superbloc
- bitmap

Exercițiu 1

- Un sistem de fișiere Unix-like dispune de următoarele structuri:
  - dentry = [30 octeți nume + 2 octeți număr inode]
  - inode = [metadata + 7 pointeri + 1 pointer indirectare]
  - bloc de date = 1024 octeți
  - adresa unui bloc de date = 32 de biți
  - superbloc = 512 octeți
- Presupunând un disc cu dimensiune suficient de mare, câte fișiere se pot crea pe acest sistem de fișiere?

Exercițiu 2

- Un sistem de fișiere Unix-like dispune de următoarele structuri:
  - dentry = [30 octeți nume + 2 octeți număr inode]
  - inode = [metadata + 7 pointeri + 1 pointer indirectare]
  - bloc de date = 1024 octeți
  - adresa unui bloc de date = 32 de biți
  - superbloc = 512 octeti
- Presupunând un disc cu dimensiune suficient de mare, câte legături simbolice, respectiv legături hard pot fi create?

Exercițiu 3

- Pentru un sistem de fișiere de tip MINIX, câte blocuri de date și metadate ocupă un fișier de dimensiune 1MB, dacă
  - dimensiunea unui block de date este de 1024 octeți
  - pointerii din zonele indirectate au 32 de biți

Întrebări

?