

10

Implementarea sistemelor de fișiere

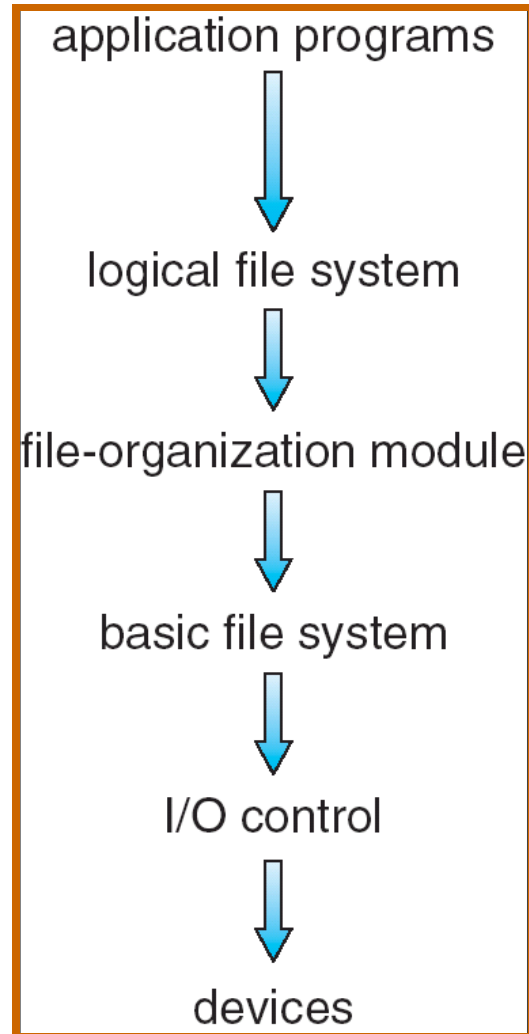
13 mai 2009

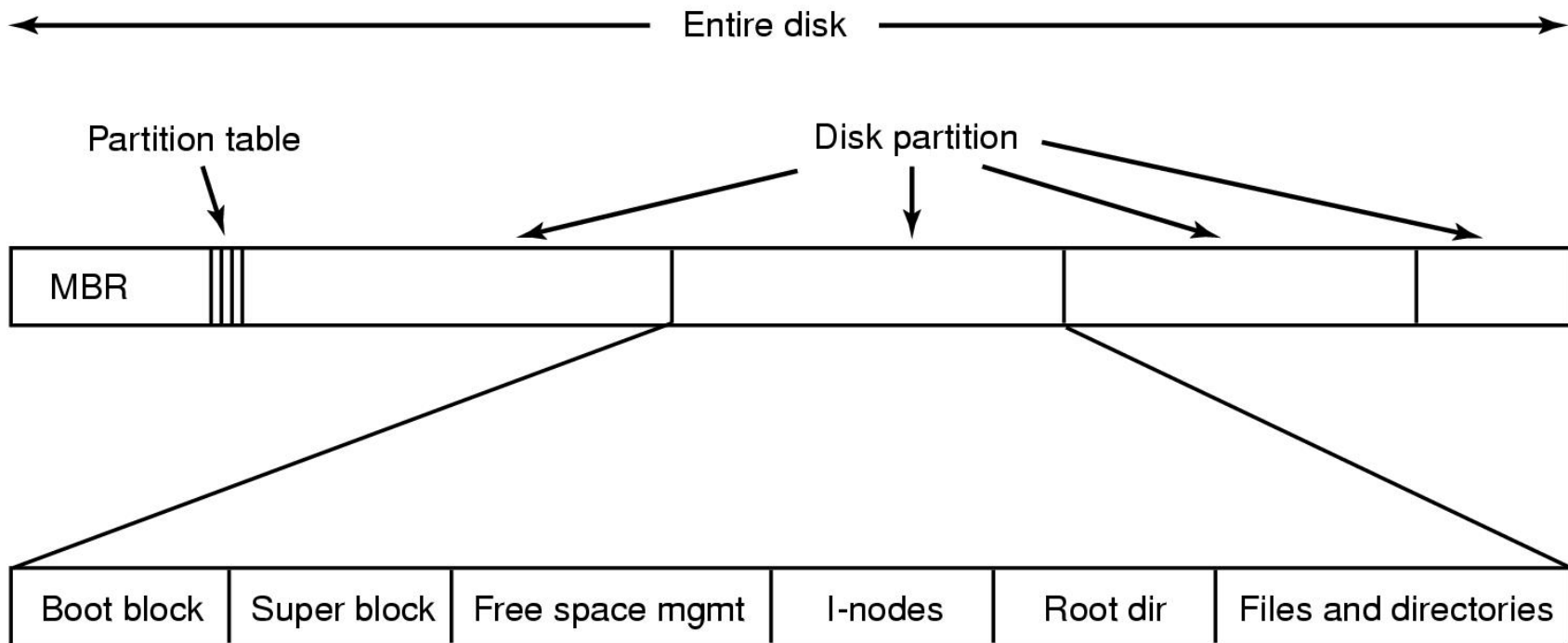
- OSC
 - Capitolul 11 – File-System Implementation
 - Secțiunile 11.1 - 11.8
- MOS
 - Capitolul 6 – File Systems
 - Secțiunile 6.3, 6.4

- 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 fișiere jurnalizate
- Sistemul de fișiere MINIX

- 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

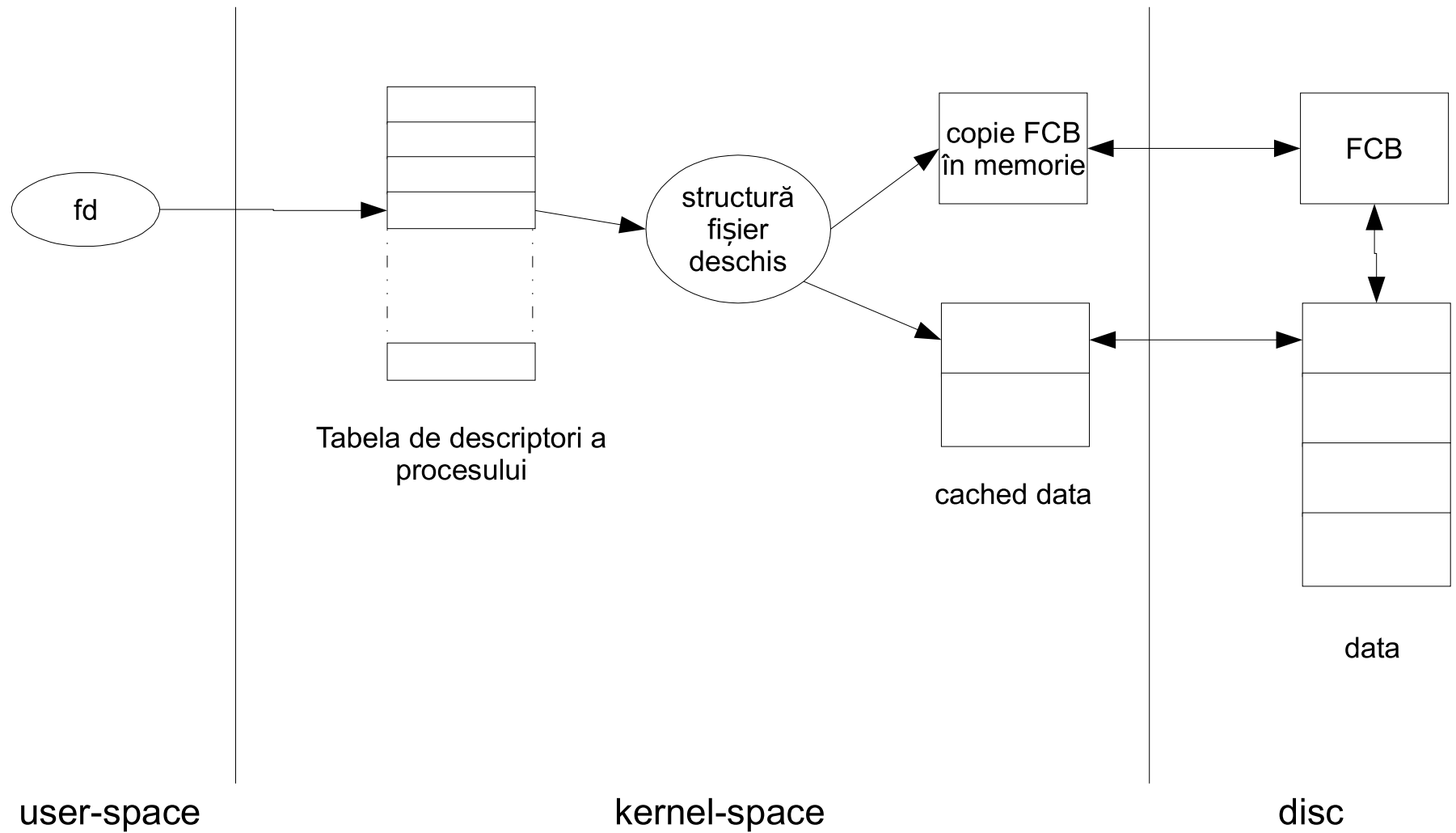
- Mod de organizare a datelor pe un suport persistent
- Două roluri
 - oferirea unei interfețe facile utilizatorului
 - fișiere, directoare, ierarhie
 - atribute (drepturi, nume, timpi)
 - algoritmi de alocare și organizare a informației pe suportul fizic
 - translatore cereri de la utilizator în sectoare





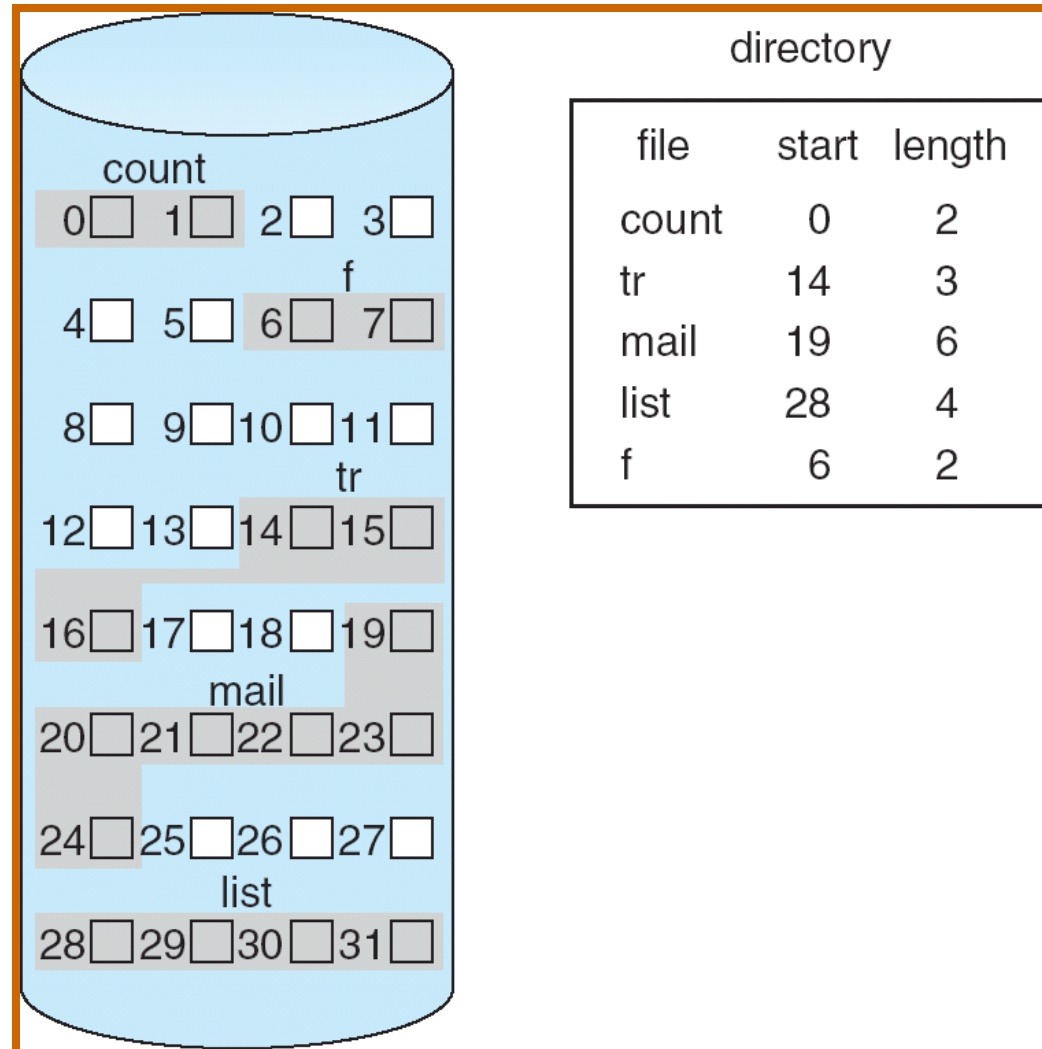
- Entitatea de bază a unui sistem de fișiere
- Abstractizează datele/informațiile
- În Unix, totul e un fișier; dacă nu e fișier, atunci e proces

- 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)

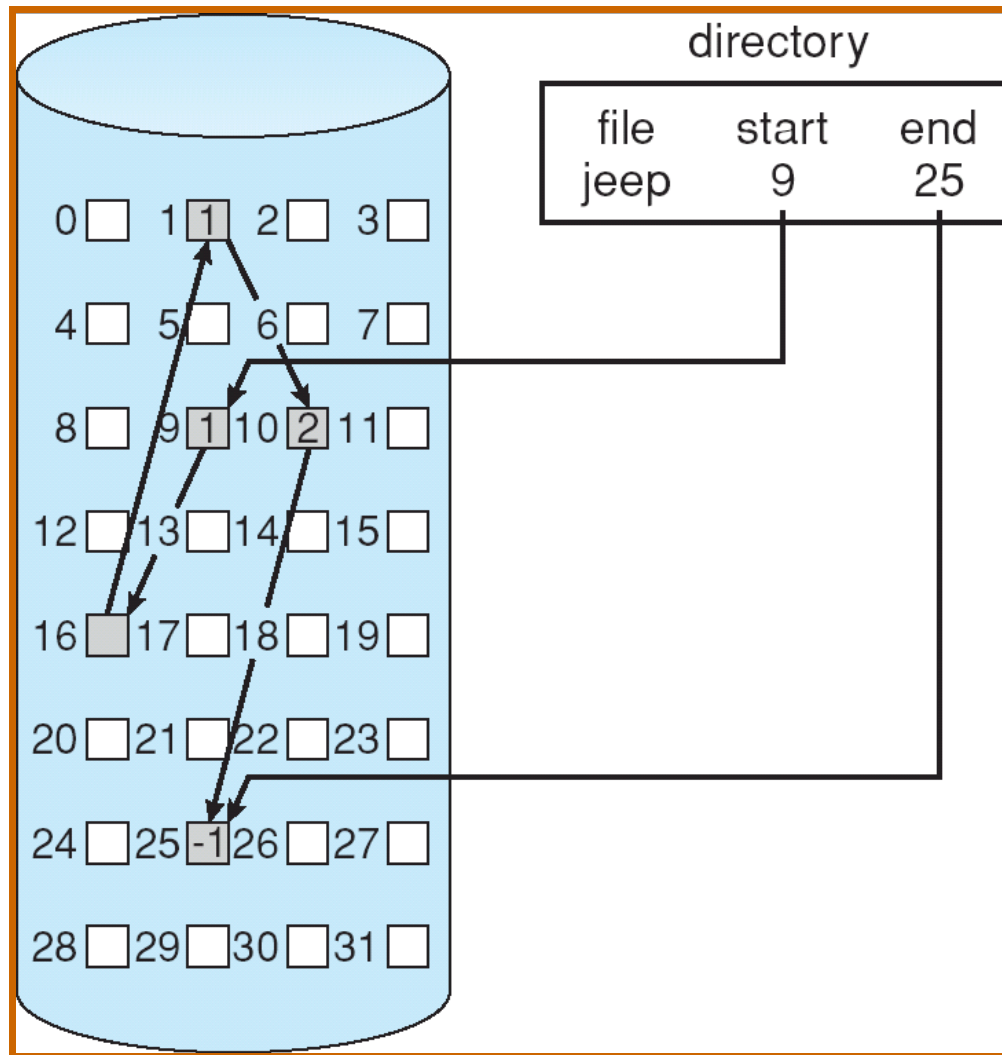


- Mecanismul prin care SO gestionează spațiul liber
 - alocare contiguă
 - alocare cu liste
 - alocare indexată

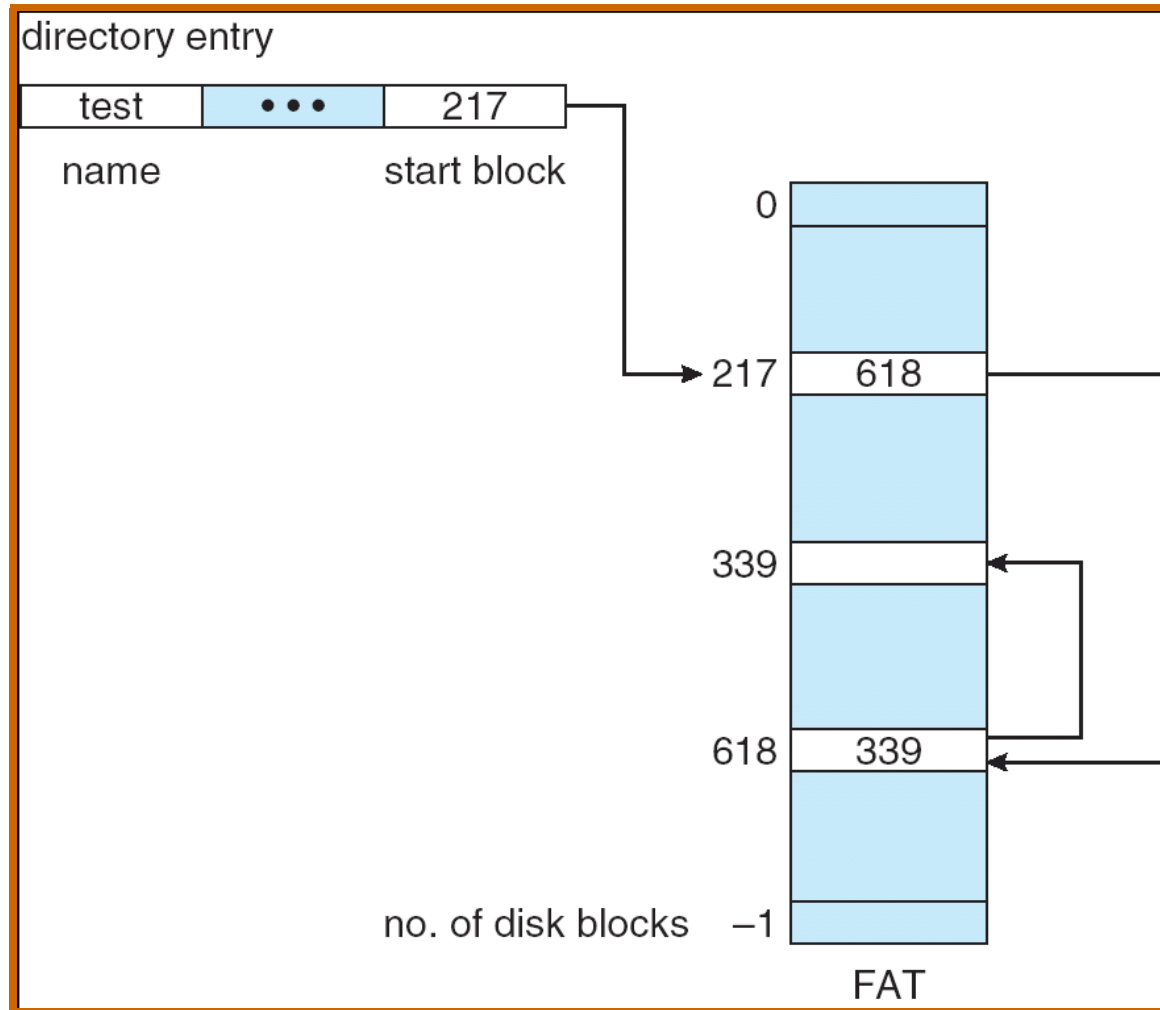
- Alocare la nivel de bloc
 - multiplu de sector
 - submultiplu de pagină
 - (512, 1024, 2048, 4092 octeți)



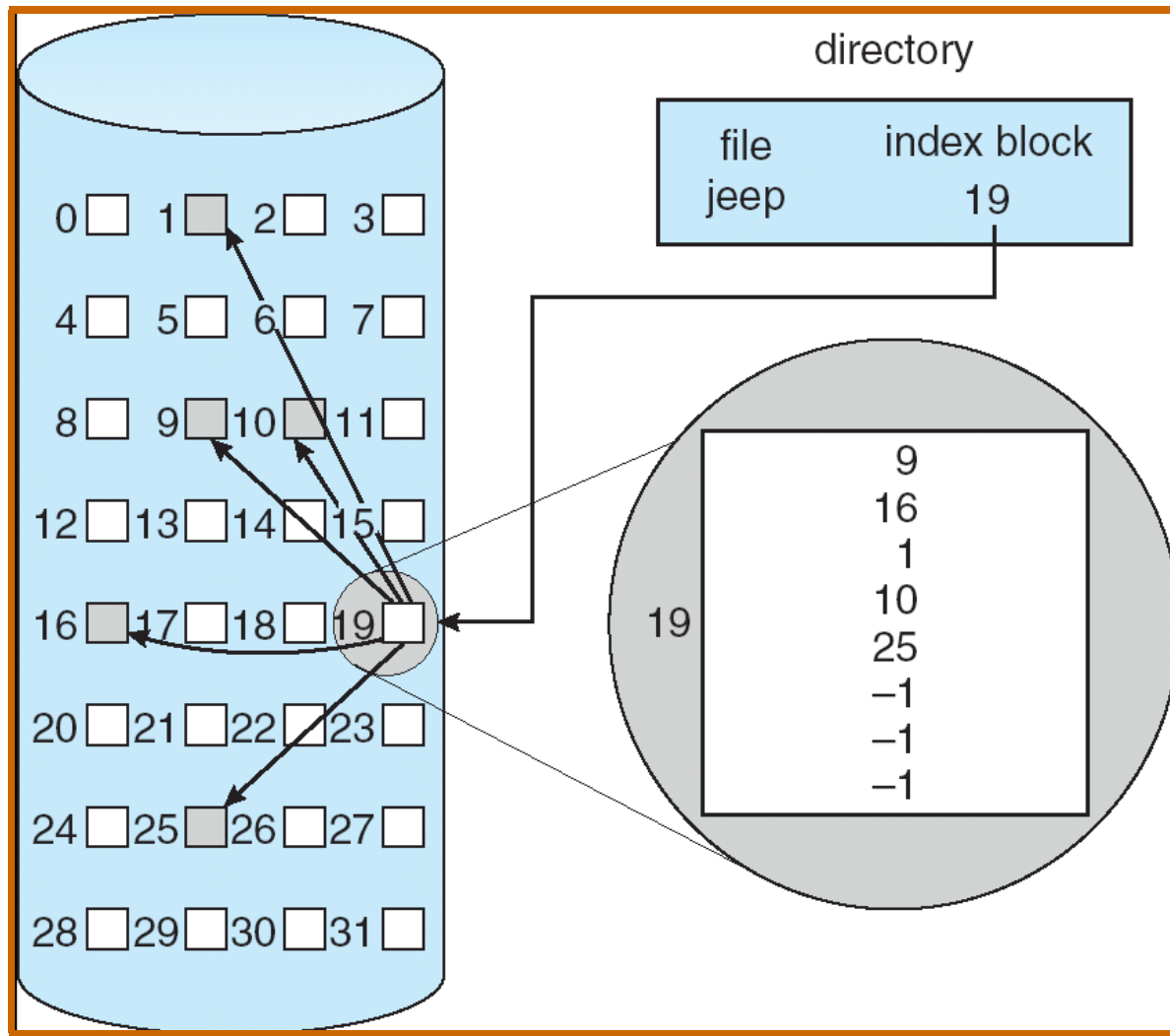
- Stocarea fișierelor ca o secvență continuă pe disc
- Avantaje
 - identificare simplă a blocurilor
 - blocul de start
 - dimensiunea fișierului în blocuri
 - viteză mare la citire
- Dezavantaje
 - fragmentare externă
 - trebuie specificată dimensiunea fișierului la crearea acestuia



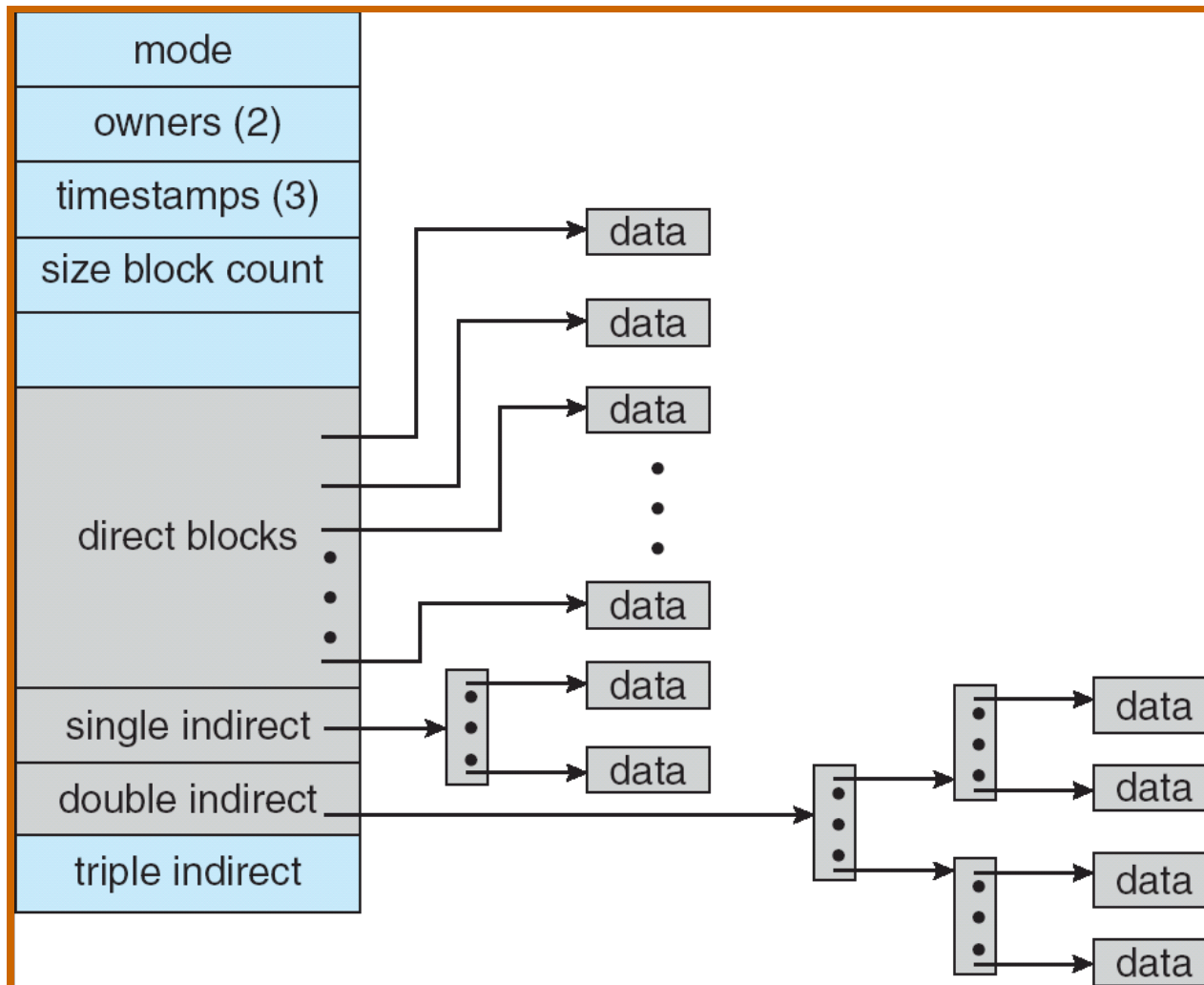
- Un bloc conține pointer către următorul bloc
- Avantaje
 - nu mai există fragmentare externă
 - este necesar doar primul bloc pentru a localiza fișierul pe disc
- Dezavantaje
 - timp de acces ridicat pentru ultimele blocuri



- O zonă separată (FAT) conține pointerii către blocurile fișierului
- O tabelă FAT de dimensiune redusă poate fi încărcată 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



- 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 indexare
 - latența crește la accesarea blocurilor „distante”



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

- 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
```

	inode	rec_len	file_type	name_len	name
0	21	12	1	2	· \0 \0 \0
12	22	12	2	2	· · \0 \0
24	53	16	5	2	h o m e 1 \0 \0 \0
40	67	28	3	2	u s r \0
52	0	16	7	1	o l d f i l e \0
68	34	12	4	2	s b i n

- 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

```
$ 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
```

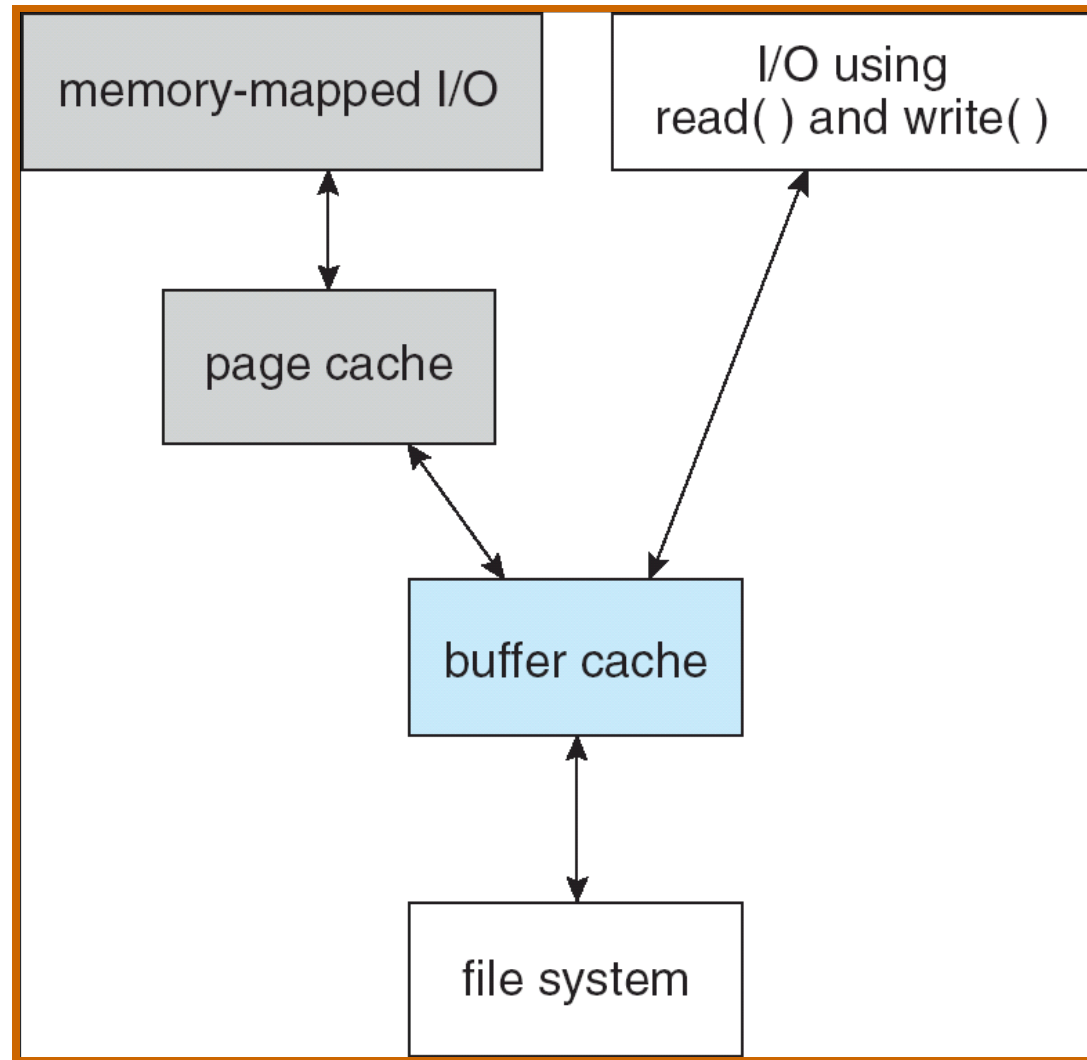

- Vector de biți
 - 1 blocul este liber
 - 0 blocul este ocupat
- Liste înlănțuite
 - primul bloc liber conține un pointer către al doilea, etc.
 - capătul listei este ținut în memorie
 - versiune îmbunătățită - se mențin tabele de blocuri libere
 - optimizare - se mențin zone contigue în tabele

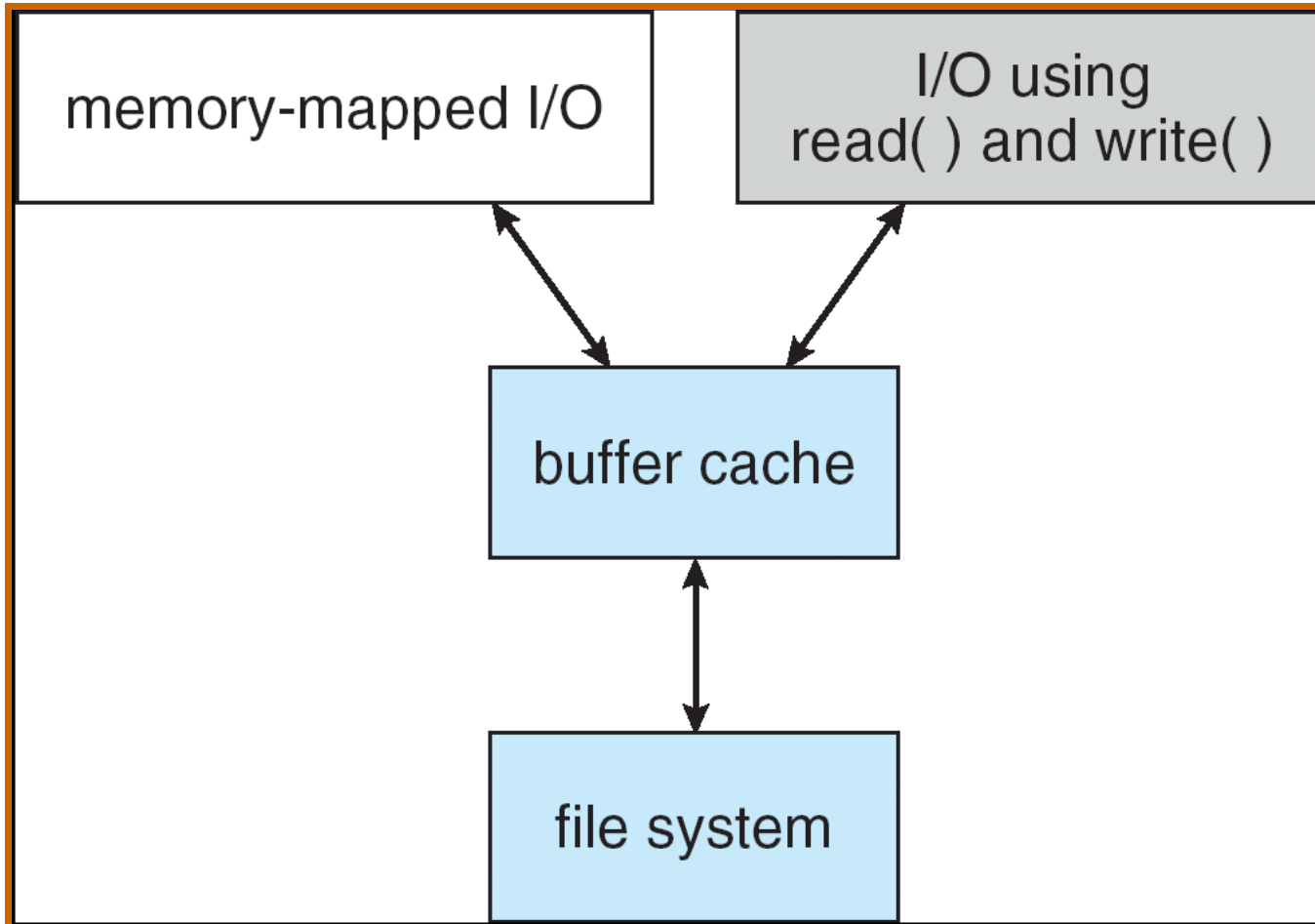
- Spațiul necesar pentru lista/vectorul de biți de blocuri libere pentru:
 - un disc de 16GB
 - blocuri de 1KB
 - adresa unui bloc pe 32 de biți
- Răspuns
 - listă: $2^{24} : (1024 : 4 - 1) = 2^{24} : 255 = 65794$ de blocuri
 - vector: $2^{24} : (8 * 1024) = 2^{11} = 2048$ de blocuri

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

- 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

- 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





- Cache-ul pentru i-noduri: icache
 - attributele fișierelor se păstrează în inode-uri
 - acces rapid pentru comenzi de tipul ls

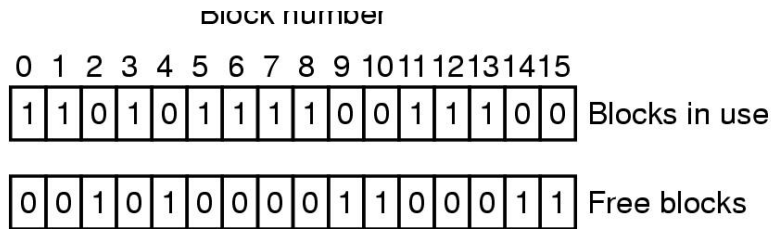
- Cache-ul pentru rezolvarea numelor în inode-uri: dcache
 - rezolvarea de nume este costisitoare
 - posibilă trecere prin mai multe blocuri
 - folosire tabele hash

- Backup-uri
 - 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

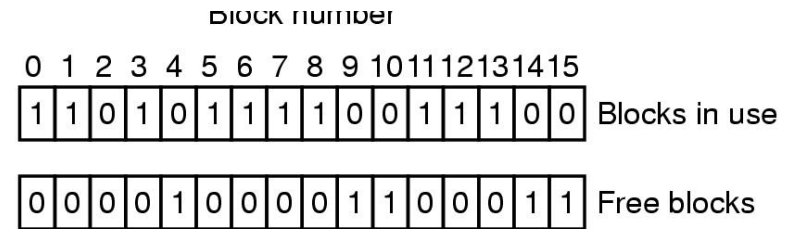
- Incosistențe
 - La blocurile de date
 - nu se pot detecta
 - nu sunt grave pentru consistența întregului sistem de fișiere
 - La blocurile auxiliare (metadata)
 - se pot detecta
 - pot avea efecte grave asupra întregului sistem de fișiere
- La boot-are, dacă sistemul de fișiere nu a fost demontat,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

- 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

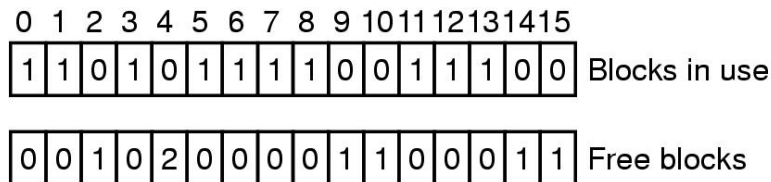
- Se repară inconsistențele
 - blocuri
 - un bloc să fie contorizat o singură dată numai într-una din tabele
 - dacă nu, se repară inconsistența
 - fișiere
 - contorul de utilizare în i-node-ul fișierului nu corespunde contorului din tabelă -> se forțează în i-node valoarea din tabelă



(a)

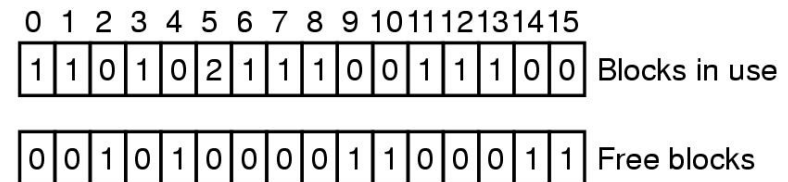


(b)



(a) SF consistent

(b) bloc pierdut

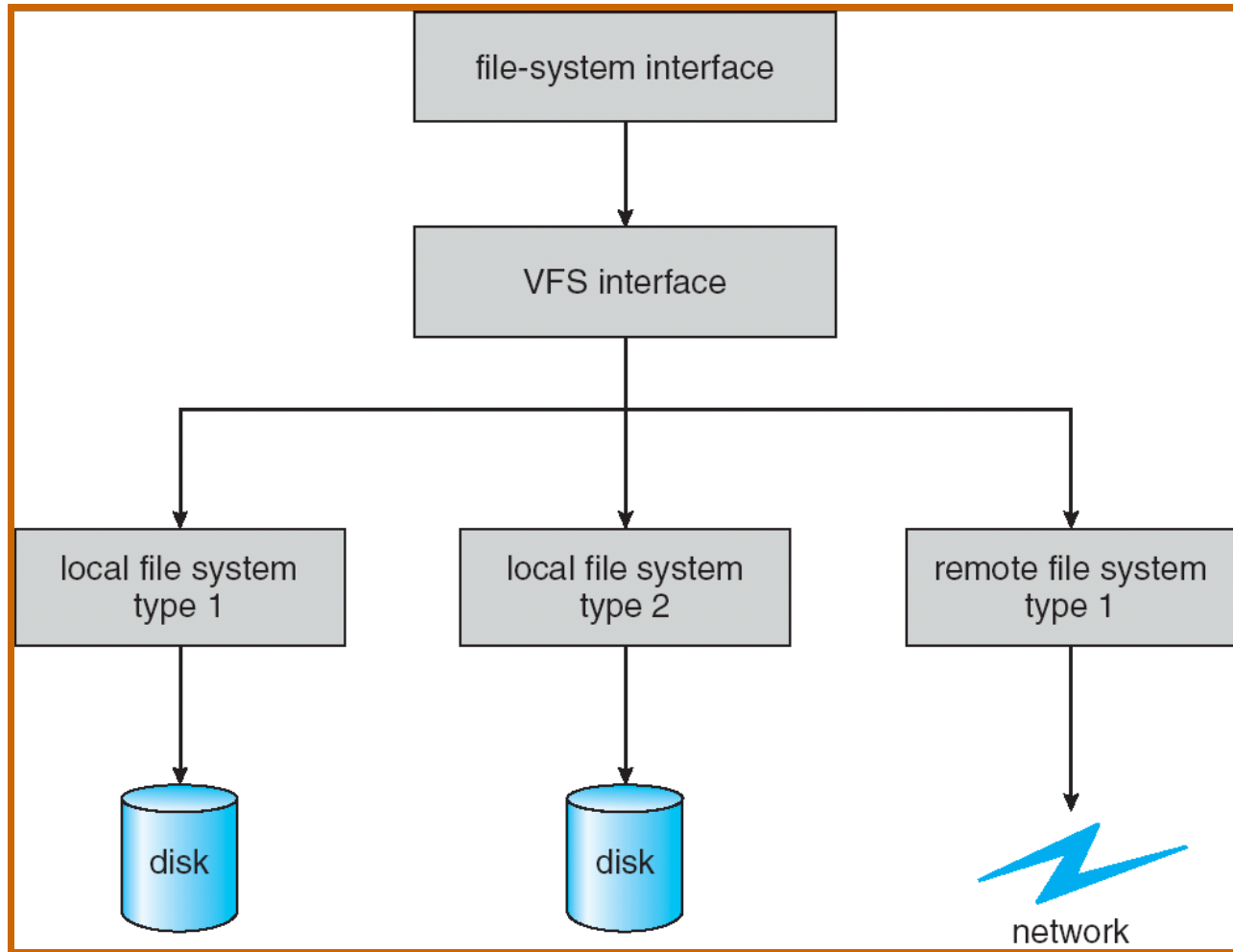


(c) bloc duplicat in lista de blocuri libere

(d) bloc de date duplicat

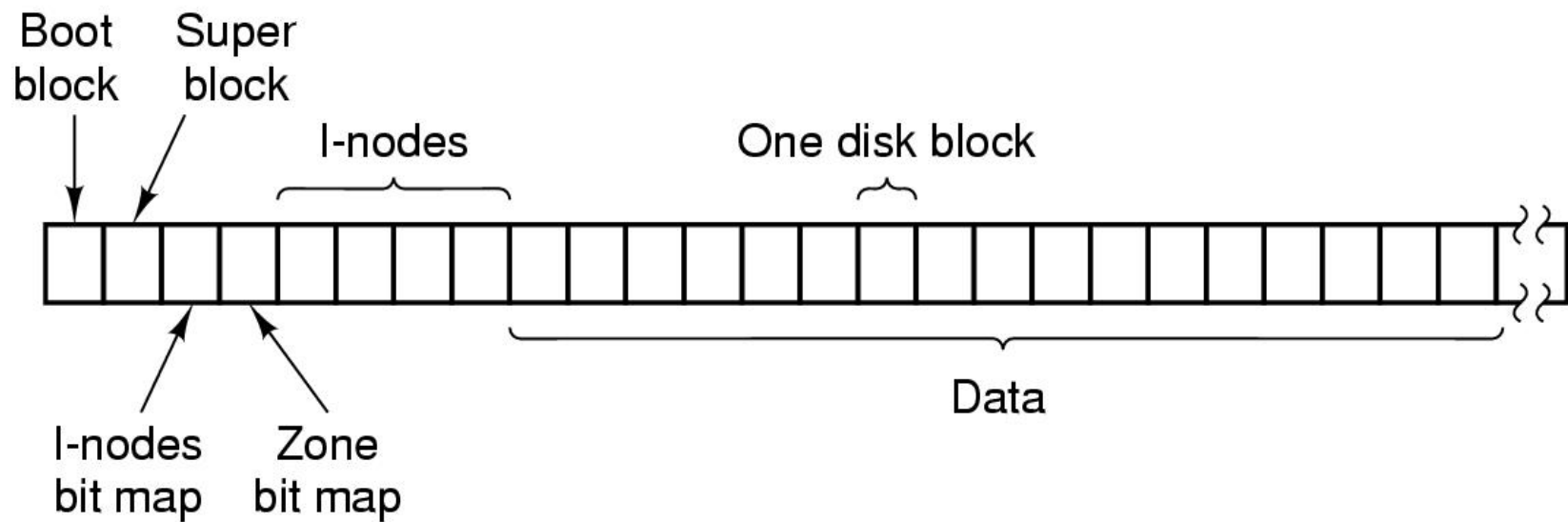
- Journaling file systems (ext3, ReiserFS, JFS)
- Operaţiunile sunt scrise într-un jurnal
 - înainte de efectuare sunt sumarizate în jurnal în forma unor tranzacţii
 - după încheiere tranzacţia este ştearsă din jurnal
- De obicei se foloseşte “metadata-only journaling”

- 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



- 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

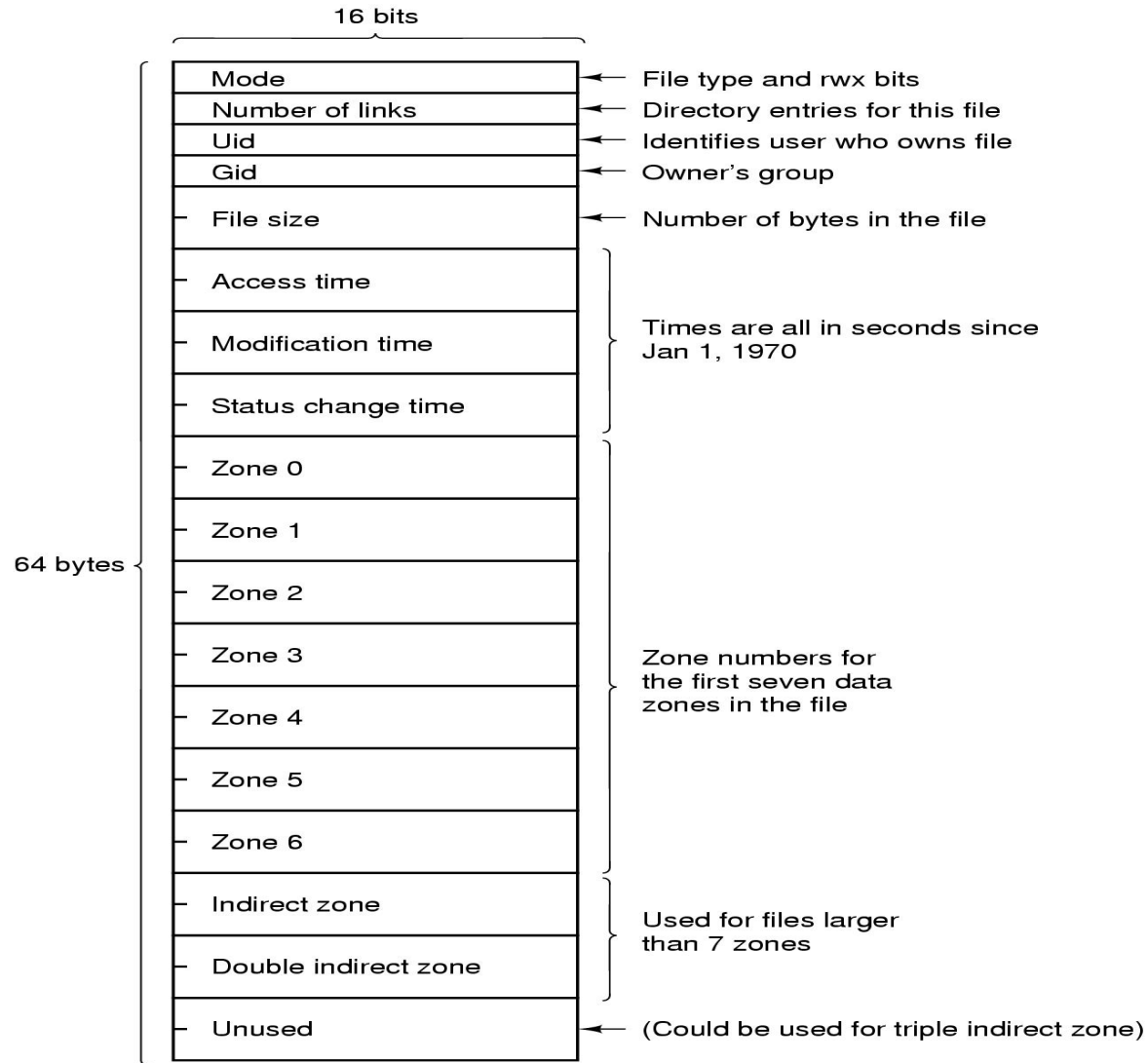
- Variantă simplificată a unui sistem de fișiere UNIX
- Folosit inițial de Linux
 - înlocuit de ext (apoi de ext2, ext3)
- Simplu (ca și FAT)
 - folosit în sisteme embedded



Present on disk and in memory	Number of nodes
	Number of zones (V1)
	Number of i-node bit map blocks
	Number of zone bit map blocks
	First data zone
	\log_2 (block/zone)
	Maximum file size
	Magic number
	Padding
	Number of zones (V2)
Present in memory but not on disk	Pointer to i-node for root of mounted file system
	Pointer to i-node mounted upon
	I-nodes/block
	Device number
	Read-only flag
	Big-endian FS flag
	FS version
	Direct zones/i-node
	Indirect zones/indirect block
	First free bit in i-node bit map
First free bit in zone bit map	

- 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

- 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



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

- 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

- 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?

- Un sistem de fișiere Unix-like dispune de următoarele structuri:
 - dentry = [30 octeți nume + 2 octeți număr inode]
 - inode = [metada + 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?

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

