Implementace souborového systému

diskový blok, sektor
disková oblast (*disk partition*) posloupnost po sobě následujících očíslovaných diskových bloků stejné velikosti

System V File System, *s5fs* (1978)
první univerzální implementace souborového sytému

jednoduchý návrh

boot blok, super blok, tabulka iuzlů, datové bloky

superblok jeden

jedna tabulka diskových iuzlů a jedna oblast datových bloků

přidělování diskových iuzlů náhodné, tj. iuzly souborů téhož adresáře nejsou seskupeny

přidělování diskových bloků suboptimální, jenom při vytvoření souborového systému v diskové oblasti je seznam volných bloků konfigurován rotačně po sobě

seznam (omezený) volných iuzlů v superbloku, po jejich vyčerpání, čtení bloků s iuzly a zjišťování volných iuzlů

seznam volných datových bloků jako seznam volných bloků s čísly volných bloků

BSD Fast File System, *ffs* (1984)

stejná funkcionalita

hlavní přínos v rozvržení disku

kromě rozdělení disku na oblasti, které obsahují souborové systémy, rozděluje oblast dále na skupiny malého počtu válců (*cylinder group*), které obsahují bloky se souvisejícími iuzly a datovými bloky

fragmentace bloků, blok může být rozdělen na 1, 2, 4, 8 fragmentů s nejmenší velikostí rovnající se velikosti sektoru

bloky souboru jsou uloženy v diskových blocích, kromě posledního bloku, který může obsahovat jeden nebo více po sobě následujících fragmentů

UNIX File System - UFS

SunOS 2.0, Solaris vychází z BSD FFS

Second Extended Filesystem – Ext2

obrázky z Bovet D.P., Cesati M.: Understanding the LINUX KERNEL

první verze operačního sytému Linux vycházely ze souborového sytému operačního systému Minix

později byl vytvořen Extended Filesystem – Ext FS
a v roce 1994 byl uveden Ext2

efektivnost Ext2

- při vytváření souborového systému je možné specifikovat velikost bloku (1024 až 4096 bytů), jestliže očekáváme soubory s několika tisíci bytů volíme velikost 1024, čím snižujeme interní fragmentaci (průměrně není využito půl bloku), na druhé straně veliké bloky pro rozsáhlé soubory snižují počet diskových operací

- pro diskovou oblast můžeme zadat povolený počet iuzlů podle očekávaného počtu souborů, což maximalizuje využití diskového prostoru

- souborový systém je rozdělen na skupiny bloků, každá skupina obsahuje bloky na sousedních stopách

- souborový systém předem přiřadí (*preallocates*) bloky obyčejným souborům, tj. předtím než jsou skutečně požadovány, při zvětšení souboru jsou tak dispozici sousedící diskové bloky

- podporuje rychlé symbolické odkazy, má-li symbolický odkaz 60 znaků nebo méně, je uložen v iuzlu

robustnost a flexibilita

- aktualizace souborů je navržena s ohledem na minimalizaci škody v případě havárie diskového systému, například při vytváření nového odkazu na soubor, napřed se zvýší počet odkazů v iuzlu a nové jméno se uloží do příslušného adresáře až následně

- podporuje automatickou kontrolu konzistenci souborového sytému, při zavádění systému, po předdefinovaném počtu připojení souborových systémů nebo po uplynutí předdefinovaného času od poslední kontroly

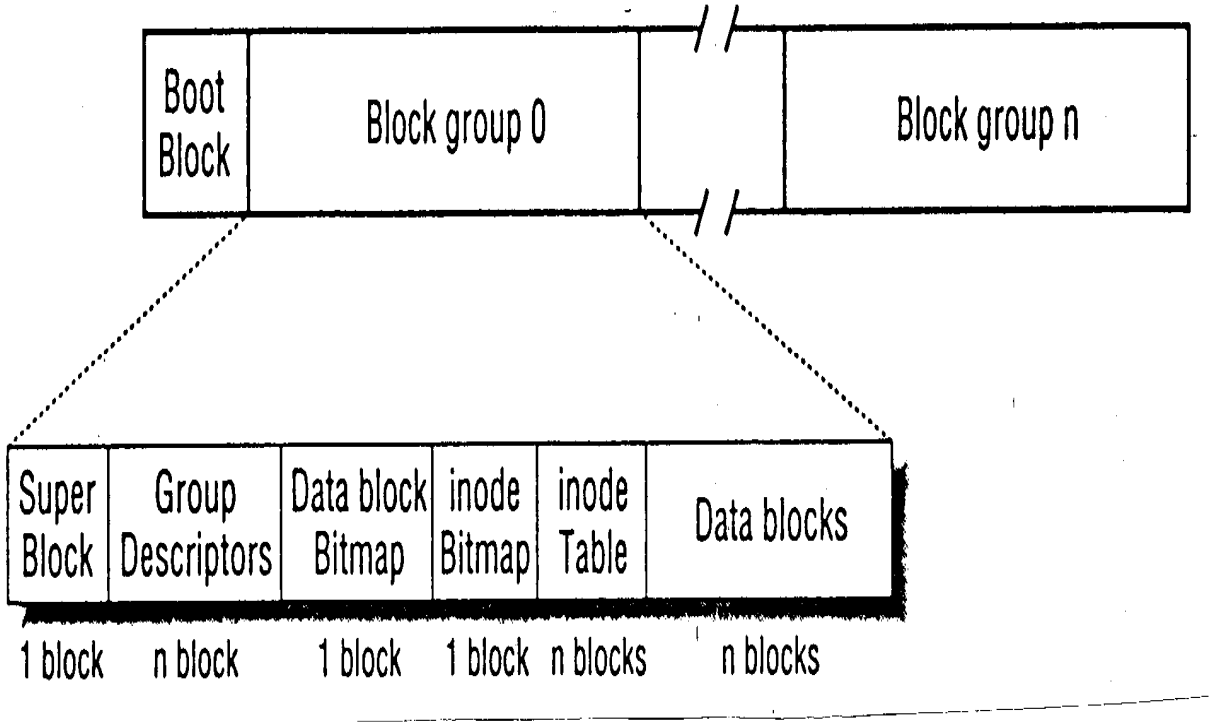
- podpora neměnných souborů

- podpora SVR4 i BSD sémantiky pro GID nového souboru, v SVR4 má nový soubor GID procesu, který ho vytvořil, v BSD nový soubor získá GID podle adresáře, ve kterém je vytvořen

diskové datové struktury

první blok každé diskové oblasti obsahuje zaváděcí program (*boot sector*)

zbytek je rozdělen na skupiny bloků s rozvržením podle obrázku



blok skupiny bloků může obsahovat

- kopii superbloku
- kopii skupiny deskriptorů skupiny bloků
- bitovou mapu datových bloků
- bitovou mapu iuzlů
- bloky iuzlů
- datové bloky souborů

o blocích, které nejsou datové říkáme, že obsahují metadata

pokud blok neobsahuje data souboru ani metadata nazývá se volný

jádro používá superblok a deskriptory skupin bloků jenom ze skupiny bloků 0, ostatní jsou udržovány jako kopie a můžou být použity v případě havárie souborového systému

počet skupin bloků v diskové oblasti je zdola omezen požadavkem, aby se bitová mapa bloků skupiny vešla do jednoho bloku

velikost *s* skupiny bloků může být nejvíc *8*b* bloků, kde *b* je velikost bloku v bytech $s < 8*b$

je-li *o* velikost diskové oblasti v blocích počet skupin bloků je $o/s \geq o/(8*b)$

čím menší blok, tím menší skupina bloků a pro danou velikost diskové oblasti větší počet skupin bloků

počet skupin bloků je přibližně $o/(8*b)$

příklad

o = 8 GB, b = 4KB
bitová mapa opisuje využití 32K bloků
oblast obsahuje 8 GB / 4 KB = 2M bloků
je třeba nejmíň 2M / 32K = $2^{11} / 2^5$ = 64 skupin bloků

superblok

obsahuje údaje pro správu celého souborového systému

položky

`s_inodes_count`
uchovává počet iuzlů v souboru

`s_blocks_count`
uchovává počet bloků v souborovém systému Ext2

`s_log_block_size`
vyjadřuje velikost bloku uvedením dvojkového exponentu pro počet 1024 bytových jednotek, tedy pro blok velikosti 1024 bytů (slabik, bajtů) obsahuje 0, pro 2048 bytové bloky obsahuje 1, atd.

`s_log_frag_size`
`= s_log_block_size` verze 2.2, 2.4 nepodporují fragmentaci bloku

`s_blocks_per_group`
`s_frags_per_group`
`s_inodes_per_group`
uchovávají počet bloků, fragmentů a iuzlů ve skupině bloků

`s_def_resuid`
`s_def_resgid`
rezervované bloky, umožňující administrátorovi pracovat se souborovým systémem i když pro ostatní uživatele nejsou již žádné volné bloky

s_mnt_count
s_max_mnt_count
s_lastcheck
s_checkinterval
řídí vykonání kontroly konzistence souborového systému - *fsck*
s_state
uchovává stav souborového systému
0 - je připojen nebo nebyl bezchybně odpojený, např. při havárii systému
1 - byl bezchybně odpojený
2 - obsahuje chyby
umožňuje vykonat kontrolu konzistentnosti při zavádění systému

deskriptor skupiny bloků

záznam velikosti 32 bytů, posledních 14 nevyužito, obdoba superbloku pro skupinu bloků

bg_block_bitmap
bg_inode bitmap
čísla bloků s bitovými mapami bloků skupiny a iuzlů

bg_inode_table
číslo bloku s prvním iuzlem tabulky iuzlů

bg_free_blocks_count
bg_free_inodes_count
bg_used_dirs_count
počet volných bloků a iuzlů v skupině bloků a počet adresářů v skupině bloků, slouží pro přidělování bloků

bitové mapy

posloupnost bitů, ve které hodnota 0 specifikuje, že odpovídající blok nebo iuzel je volný a hodnota 1 specifikuje, že je používán

každá bitová mapa musí být uchována v jednom bloku velikosti 1024, 2048 nebo 4096 bytů, jedna bitová mapa opisuje 8192, 16 384, nebo 32 768 bloků

tabulka (pole, seznam) iuzlů

každý iuzel má velikost 128 bytů, blok o velikosti 1024 tedy obsahuje 8 iuzlů, ...

počet_iuzlů_v_bloku* = s_log_block_size / *128

počet bloků obsazených iuzly ve skupině je

s_inodes_per_group / *počet_iuzlů_v_bloku*

mnoho položek iuzlu na disku odpovídá položkám iuzlu ve VFS – typ, přístupová práva, vlastník, velikost, časy přístupu a změny, ...

další zohledňují specifika implementace Ext2, většinou uložení souboru v diskových blocích

položky iuzlu

`i_size`
délka souboru v bytech

`i_blocks`
počet přidělených datových bloků v jednotkách 512 bytů

obecně
`i_size != 512 * i_blocks`

soubor velikosti jeden byte obsadí celý blok (bez fragmnetace)

soubor může obsahovat mezery a potom může být
`i_size > 512 * i_blocks`

`i_block`
pole velikosti **`EXT2_N_BLOCKS`**, obvykle 15, ukazatelů použitých na určení datových bloků souboru

diskový iuzel se určí z VFS čísla iuzlu (na disku číslo iuzlu není uloženo na disku)

pořadí_skupiny_bloků =
číslo_iuzlu / **`s_inodes_per_group`**

pořadí_iuzlu_v_tabulce =
číslo_iuzlu % **`s_inodes_per_group`**

použití diskových bloků různými typy souborů

obyčejné soubory

po vytvoření nebo po zkrácení (**`truncate()`**) obyčejný soubor neobsazuje žádné bloky, dále jsou mu přidělovány když je potřebuje

adresář

datové bloky adresáře obsahují záznamy typu **`ext2_dir_entry_2`** s položkami

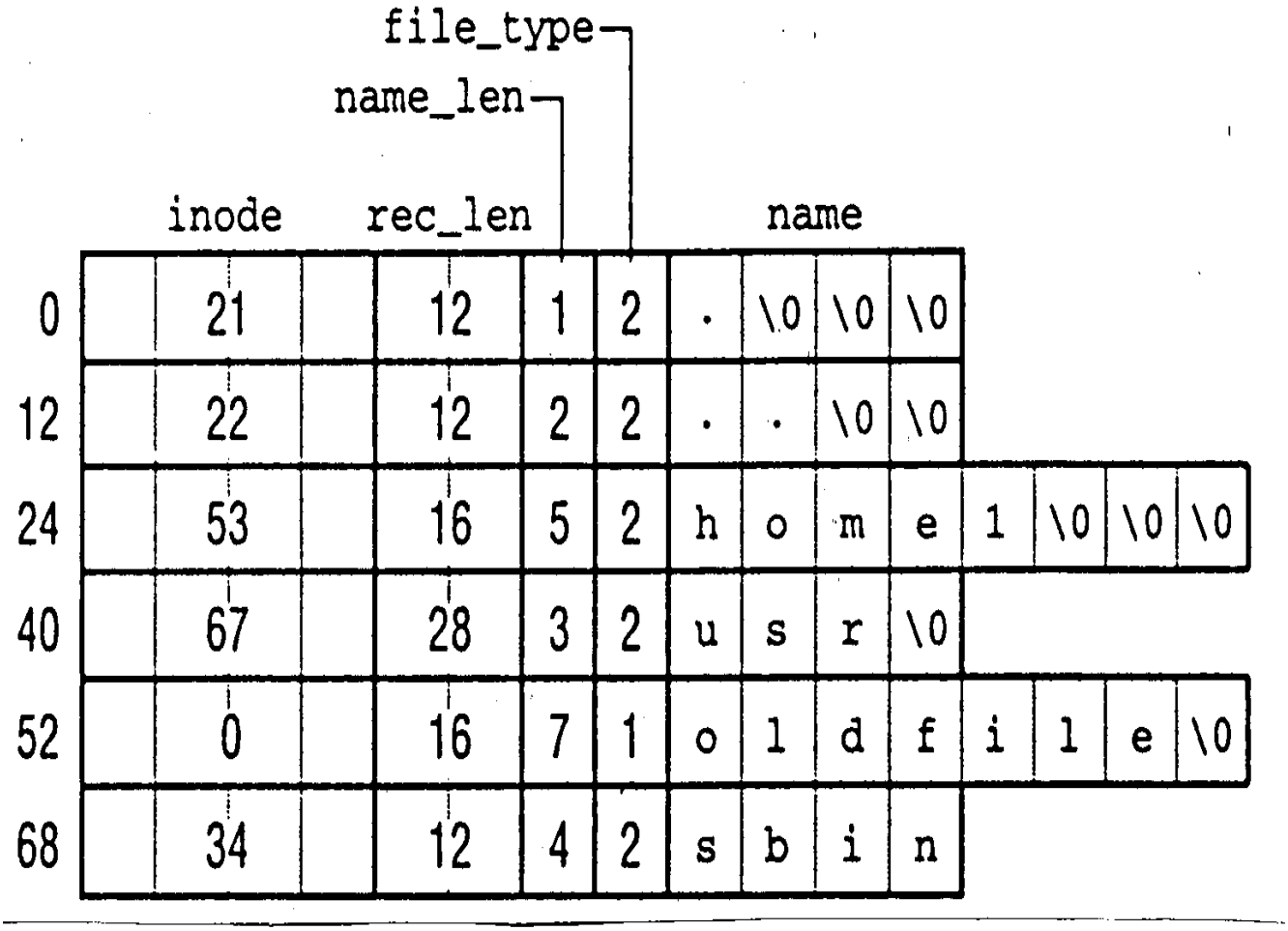
`inode`
: číslo iuzlu

`rec_len`
: délka položky adresáře, slouží na určení začátku následující položky

`name_len`
: skutečná délka jména souboru

`file_type`
: typ souboru
0 – neznámy
1 – obyčejný soubor
2 – adresář
3 – znakové zařízení
4 – blokové zařízení
5 – pojmenovaná roura
6 – soket
7 – symbolický odkaz

`name`
: pole proměnné délky, maximálně **`EXT2_NAME_LEN`** znaků, obvykle 255, vždycky však násobek 4, je-li třeba doplní se znaky \0



v Ext2 adresáři na obrázku byl zrušen soubor ***oldfile***

symbolické odkazy

pole **`i_block`** má typicky 15 prvků velikosti 4 byty pro určení uložení dat souboru, není-li cesta delší než 60 bytů bude uložena přímo v iuzlu a nejsou potřeba žádné datové bloky, jinak je potřebný jeden datový blok

soubory typu zařízení, roura, soket

nevyžadují datové bloky

datové struktury v paměti

většina informací uložených v diskových datových strukturách je při jejich používání kopírována do mezipaměti v hlavní paměti

typ	uložení v mezipaměti
superblok	stále v mezipaměti
deskriptor skupiny	stále v mezipaměti
bitová mapa bloků	pevné omezení
bitová mapa iuzlů	pevné omezení
iuzel	dynamické
datový blok	dynamické

data, která jsou často aktualizována jsou v mezipaměti stále

ve vyrovnávacích pamětech bloků (*buffer cache*) se uchovává pevný počet bitových map bloků a iuzlů, nejstarší jsou přepsány na disk když jejich počet překročí omezení

iuzly a datové bloky jsou uchovávány ve vyrovnávacích pamětech bloků dokud je sdružený objekt používán, po skončení používání můžou být přepsány na disk

specifické informace Ext2 v objektu superblok

po připojení souborového systému typu Ext2 položka **`u`** objektu superblok je naplněna specifickou informací obsahující

- většinu položek diskového superbloku
- mezipaměť bitových map bloků, kterou tvoří dvojice polí **`s_block_bitmap`** a **`s_block_bitmap_number`**
- mezipaměť bitových map iuzlů, kterou tvoří dvojice polí **`s_inode_bitmap`** a **`s_inode_bitmap_number`**
- ukazatelé na hlavičku a vyrovnávací paměť bloku s diskovým superblokem
- počet deskriptorů skupiny bloků, které můžou být umístněny do bloku

...

specifické informace Ext2 v objektu iuzel

při inicializaci objektu iuzel je položka **`u`** naplněna specifickou informací obsahující

- většinu položek diskového iuzlu, které nejsou ve všeobecném objektu iuzel
- velikostí fragmentu a počtem fragmentů (nevyužito)
- index skupiny bloků, do které iuzel patří
- položky **`i_alloc_block`** a **`i_alloc_count`** využité pro přidělení bloků předem

mezipaměti bitových map

vyrovnávací paměť bloku obsahující superblok připojeného souborového systému Ext2 se uvolní jenom když se souborový systém odpojí

všechny bitové mapy vzhledem na jejich počet není možné stále uchovávat v hlavní paměti

příklad

velikost disku 4 GB
velikost bloku 1 KB

bitová mapa v jednom bloku opisuje $8*2^{10}$ bloků = 8K

*nejmenší počet skupin bloků je 4 GB / (8K *1KB) = 512*

velikost bitových map bloků a iuzlů v jedné skupině bloků je 2 KB

na mezipaměť všech bitových map je třeba 1 MB paměti

Ext2 používá mezipaměti bitových map bloků a bitových map iuzlů, každou o velikosti **`EXT2_MAX_GROUP_LOADED`** (obvykle 8)

uchovávají naposledy použité bitové mapy

mezipaměť implementuje dvojice polí velikosti **`EXT2_MAX_GROUP_LOADED`**

`s_inode_bitmap` obsahuje čísla skupin bloků, kterých bitová mapa iuzlů se nachází v mezipaměti
`s_inode_bitmap_number` obsahuje ukazatele na hlavičky vyrovnávacích pamětí bloků

je-li počet skupin bloků v Ext2 diskové oblasti menší nebo rovný **`EXT2_MAX_GROUP_LOADED`**, index v poli mezipaměti se rovná indexu skupiny bloků

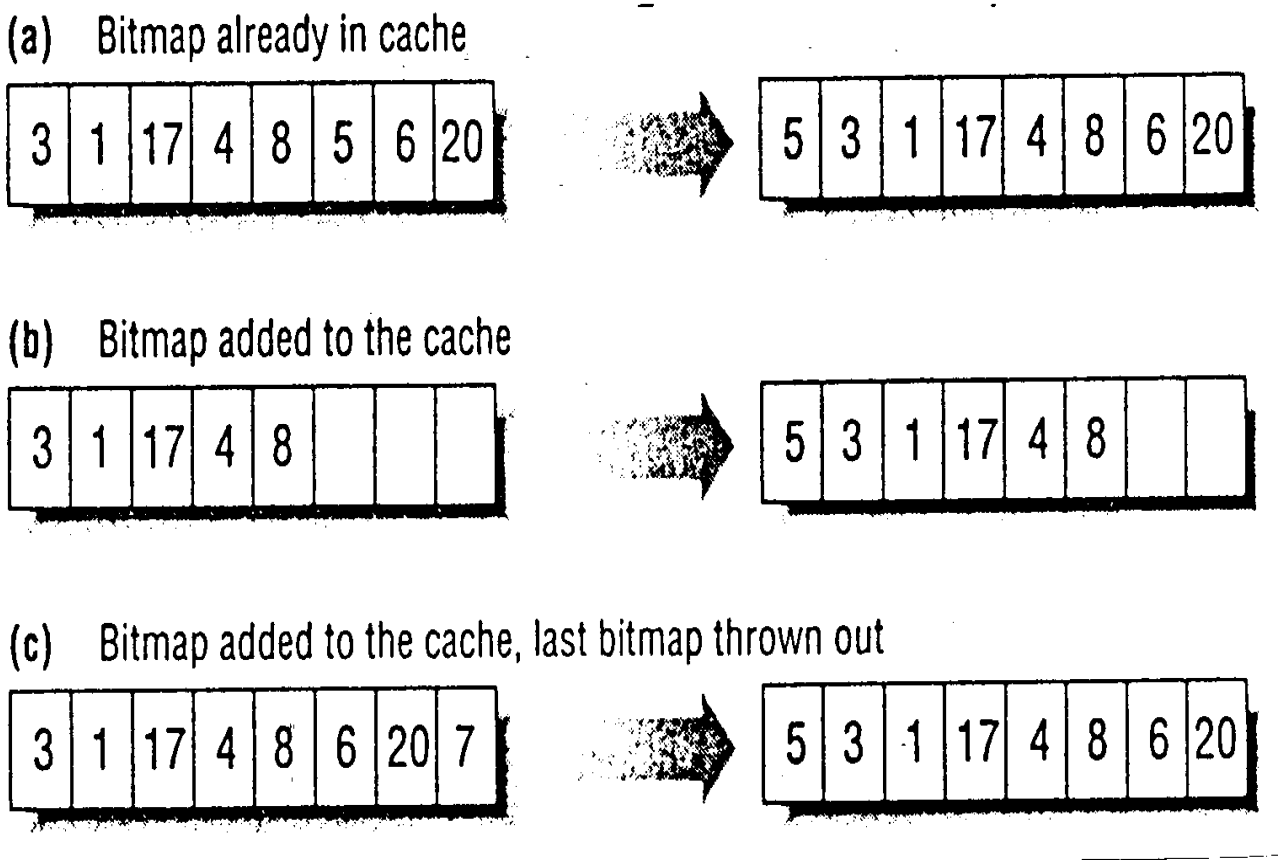
příklad

velikost diskové oblasti 1 GB
velikost bloku 4 KB
*nejmenší počet skupin bloků 1GB / (8 * 4 K * 4 KB) = 8*

jinak se používá strategie LRU (Least Recently Used) a požadovaná bitová mapa (index její skupiny bloků) je umístněn na začátek

příklad

na obrázku jsou tři možné případy, kdy je požadována bitová mapa v skupině bloků s indexem 5



správa diskového prostoru

přidělování diskových iuzlů

je-li nový iuzel adresářem je snahou, aby jsme udrželi stejnoměrné rozložení adresářů v částečně zaplněných skupinách bloků

přidělí se iuzel ze skupiny bloků, která má největší počet volných bloků ze všech skupin bloků s počtem volných iuzlů větším než je průměrný počet volných bloků

není-li adresářem, označme ***i*** skupinu bloků, ve které je rodičovský adresář, hledá se volný iuzel postupně v log*(počet_skupin_bloků)* skupinách bloků s indexy

i mod (počet_skupin_bloků),
i+1 mod (počet_skupin_bloků),
i+1+2 mod (počet_skupin_bloků),
i+1+2+4 mod (počet_skupin_bloků)

nenalezne-li se volný iuzel prohledávají se skupiny bloků sekvenčně od první skupiny bloků

adresování datových bloků

neprázdné obyčejné soubory sestávají z několika datových bloků

tyto bloky můžeme označovat pořadím v souboru, tj. čísly bloků souboru, nebo čísly diskových bloků souborového systému

obecně přistupujeme k souboru od pozice *f* a nutno určit, ve kterém diskovém bloku se byte na této pozici v souboru nachází *číslo bloku souboru = f / velikost bloku*

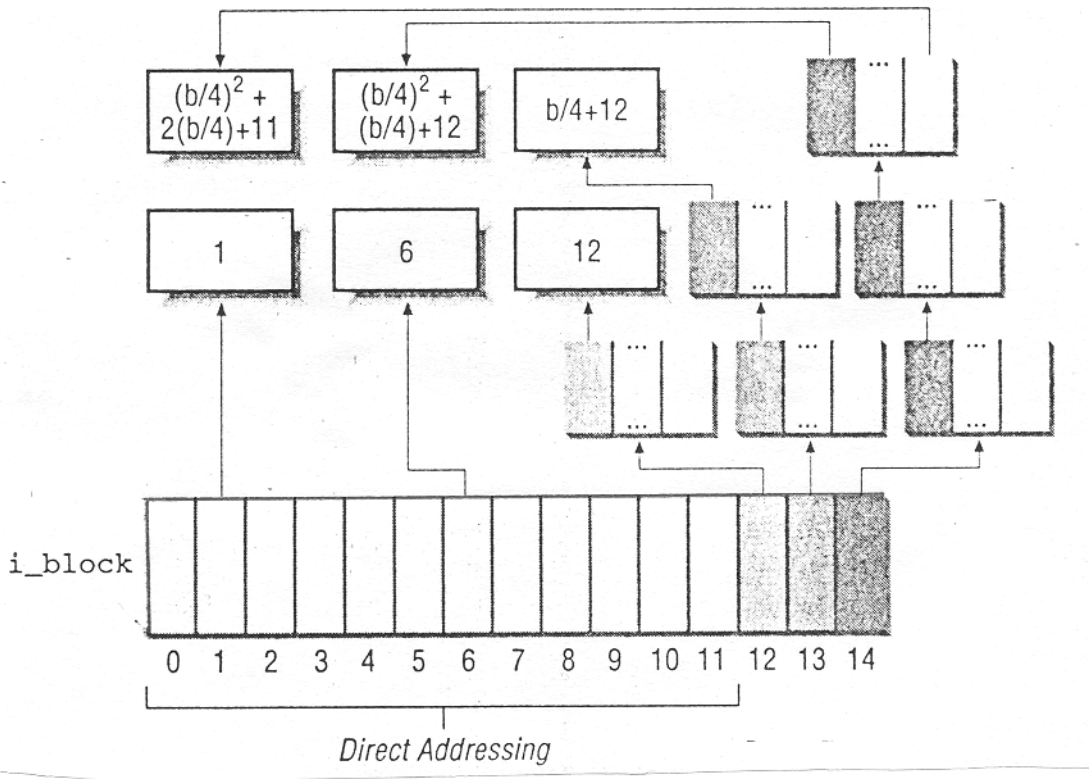
číslo bloku souboru se převede na číslo diskového bloku, ve kterém se nachází, využitím pole **`i_block`** v diskovém iuzlu

- prvních 12 prvků obsahuje čísla diskových bloků, které obsahují prvních 12 bloků souboru, které mají čísla bloků souboru od 0 do 11

- prvek s indexem 12 obsahuje číslo diskového bloku, obsahující další pole čísel diskových bloků, je-li velikost bloku *b* a číslo diskového bloku je umístněno ve 4 bytech, potom jsou to čísla diskových bloků dalších bloků souboru, které mají čísla od 12 do *b/4 + 11*

- prvek s indexem 13 obsahuje číslo diskového bloku, který obsahuje čísla diskových s čísly diskových bloků pro následující bloky souboru

- prvek s indexem 14 obsahuje číslo diskového bloku, ...


$(b/4)^2 + 2(b/4)+11$
$(b/4)^2 + (b/4)+12$
b/4+12
1
6
12
i_block
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Direct Addressing

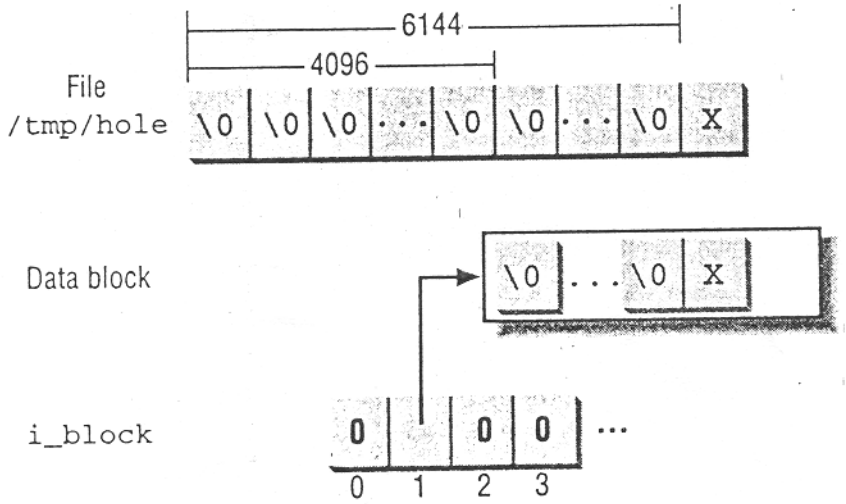
mezery v souborech (*file holes*)

aplikace využívající rozptýlení (hash) v souborech vytvářejí v souboru mezery, pro bloky souboru, které jsou v mezeře není třeba přidělit diskové bloky, ty si přidělí až se bude do bloku skutečně zapisovat

```
echo -n "X" | dd of=/tmp/hole bs=1024 seek=6
```

vytvoří soubor o velikosti 6145 znaků se znakem X na poslední pozici

je-li velikost bloku 4 KB položka **`i_size`** odpovídajícího iuzlu má hodnotu 6145 a položka **`i_blocks`** má hodnotu 8 v jednotkách 512 B



přidělování datových bloků

s cílem snížit fragmentaci souboru Ext2 zkouší přidělit nový blok nejblíže k posledně přidělenému bloku

jeli neúspěšný, hledá nový blok v skupině bloků, ve které je iuzel souboru

naposledy hledá v jiné skupině bloků

souborový systém Ext2 přiděluje datové bloky předem

`i_prealloc_block_count` uchovává počet předem přidělěných ještě nepoužitých bloků a **`i_prealloc_block`** uchovává číslo diskového bloku, který bude další použitý

pro přidělení nového bloku se nejdřív určí cíl

- obsahují-li přidělovaný diskový blok a předcházející přidělený diskový blok po sobě jdoucí bloky souboru, cíl je číslo diskového bloku předcházejícího přiděleného bloku + 1

- jinak, byl-li přidělen aspoň jeden blok, je cíl číslo diskového bloku už přiděleného bloku souboru, který v souboru předchází blok, kterému se má diskový blok přidělit

- jinak je cíl číslo prvního bloku skupiny bloků, ve které se nachází iuzel souboru

je-li cíl mezi předem přidělenými bloky, je přidělen

jinak předem přidělené bloky jsou vráceny a hledá se volný blok následovně

- je-li cíl volný, přidělí se

- je-li obsazen, kontroluje se, je-li volný jeden z následujících 64 bloků

- nenašel-li se, hledá se od skupiny bloků obsahující cíl ve všech skupinách bloků

    - skupina nejméně osmi sousedních volných bloků
    - volný blok

před skončením přidělení se zkusí předem přidělit až osm sousedních volných bloků

Ext3 souborový systém

- kompatibilní s Ext2

datové struktury v podstatě stejné s Ext2

odpojený Ext3 je možné připojit jako Ext2 a opačně

- žurnálový systém

diskové operace jsou zaznamenávány v diskové oblasti nazývané **žurnál**

změny souborového systému se vykonávají ve dvou krocích

- kopie zapisovaných bloků se uloží do žurnálu
- commit pro žurnál

- bloky se zapíší do souborového systému
- commit pro souborový systém
- bloky jsou odstraněny ze žurnálu

po chybě

- nastala-li před commit pro žurnál, bloky v žurnálu nejsou nebo nejsou úplné, jsou ignorovány *fsck*, systém zůstane konzistentní

- nastala-li po commit pro žurnál, *fsck* přepíše bloky ze žurnálu do souborového systému

žurnálové systémy ReiserFX, XFS, JFS zaznamenávají jenom operace ovlivňující metadata, systém zůstane konzistentní, data mohou být pokažena

Ext3 módy:

Journal
data i metadata jsou zaznamenávána do žunrálu
nejbezpečnější a nejpomalejší

Ordered
data jsou zapsána do souborového systému před commit jeho metatad

Writeback
zaznamenávají se jenom změny metadat
nejrychlejší mód

Ext4 souborový systém

- kompatibilní s Ext3

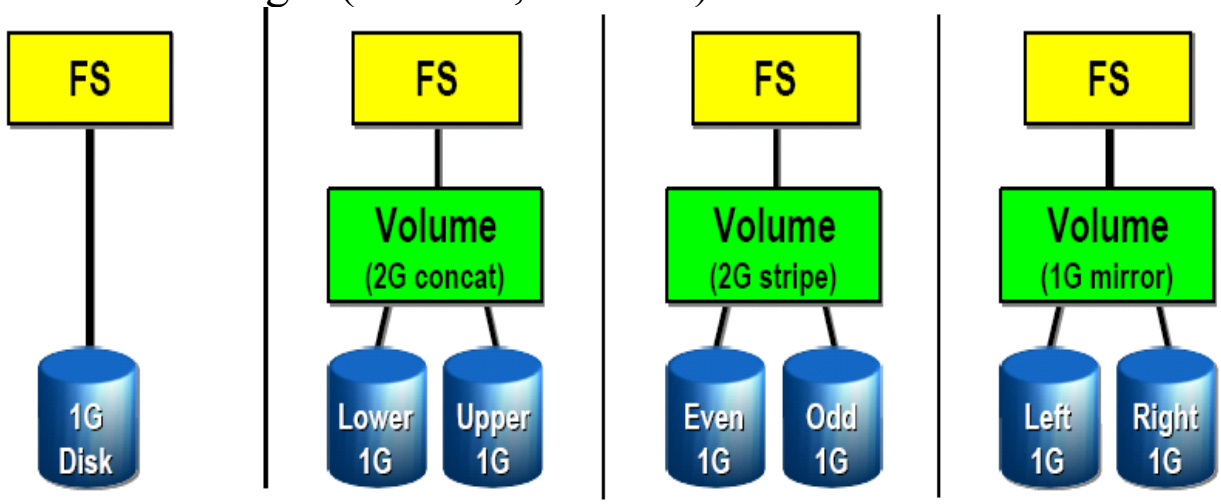
Avantika Mathur, Mingming Cao, Suparna Bhattacharya:
The new ext4 filesystem: current status and future plans

http://ols.108.redhat.com/2007/Reprints/mathur-Reprint.pdf

ZFS (2005) OpenSolaris

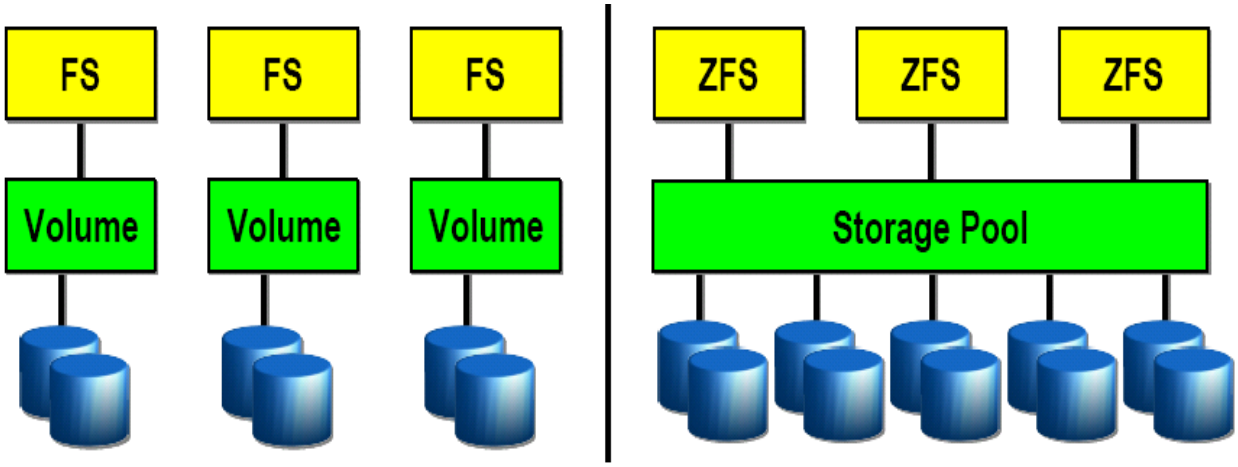
Zdroj: Bonwick J., Moore B.: ZFS The Last Word In File Systems
http://www.opensolaris.org/os/community/zfs/docs/zfs_last.pdf, 10.12.2008

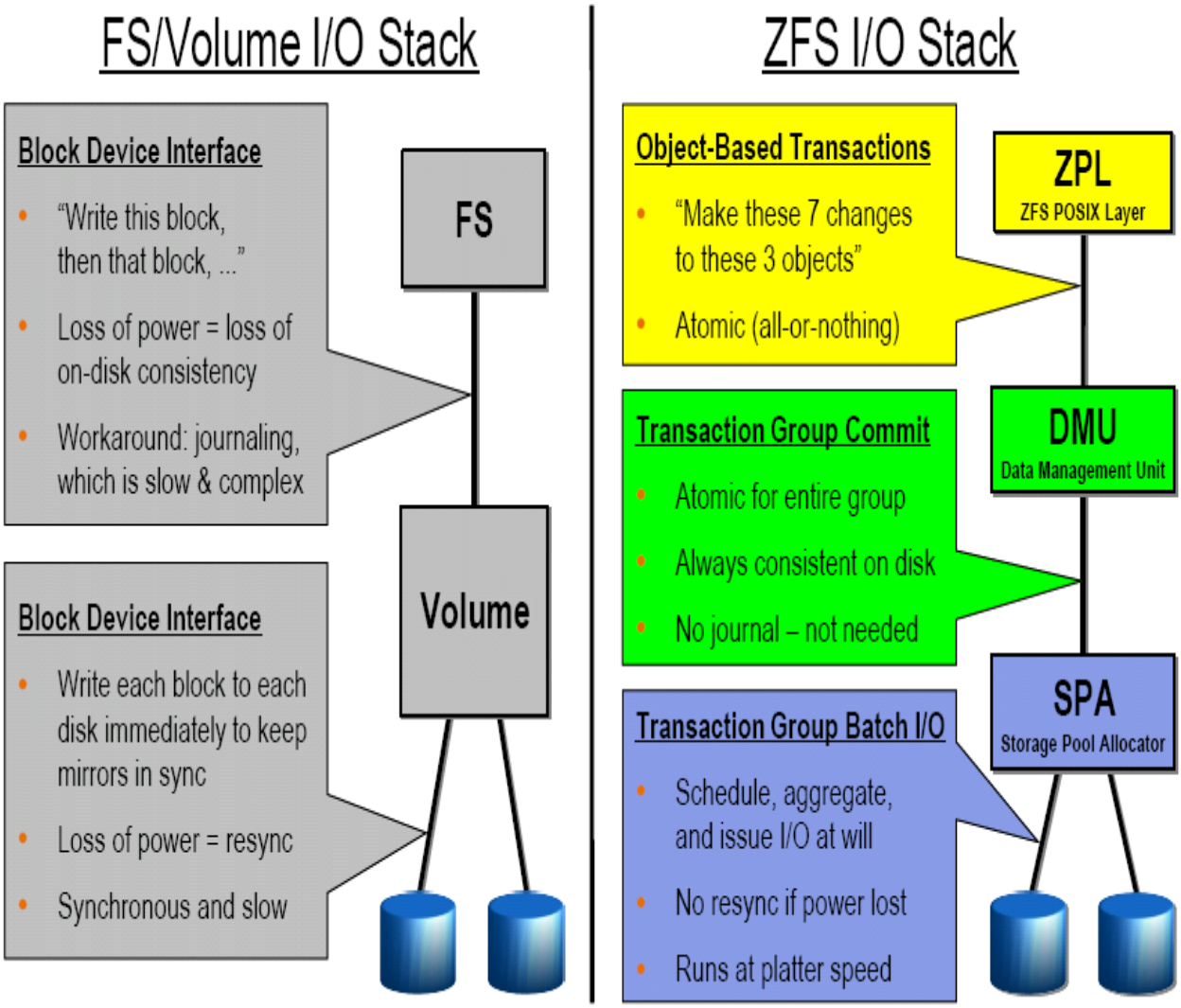
Volume Manager (RAID-0, RAID-1)



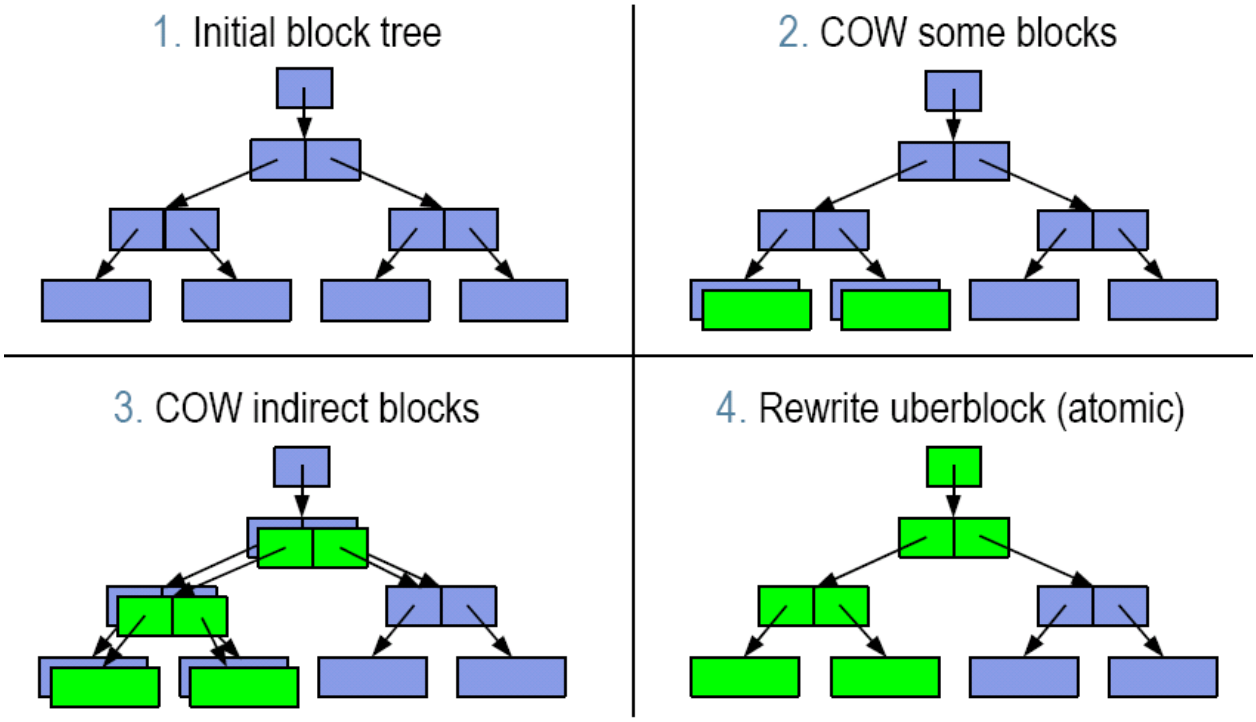
Svazek **Pool** (RAID-Z)

virtuální disk malloc/free




FS/Volume I/O Stack
Block Device Interface
"Write this block, then that block, ..."
Loss of power = loss of on-disk consistency
Workaround: journaling, which is slow & complex
FS
Volume
Block Device Interface
Write each block to each disk immediately to keep mirrors in sync
Loss of power = resync
Synchronous and slow
ZFS I/O Stack
Object-Based Transactions
"Make these 7 changes to these 3 objects"
Atomic (all-or-nothing)
ZPL
ZFS POSIX Layer
Transaction Group Commit
Atomic for entire group
Always consistent on disk
No journal – not needed
DMU
Data Management Unit
Transaction Group Batch I/O
Schedule, aggregate, and issue I/O at will
No resync if power lost
Runs at platter speed
SPA
Storage Pool Allocator

Copy on Write transakce



Snapshot (clone)