Operační systémy

Systémy souborů II

Jan Trdlička

České vysoké učení technické v Praze, Fakulta informačních technologií
Katedra počítačových systémů

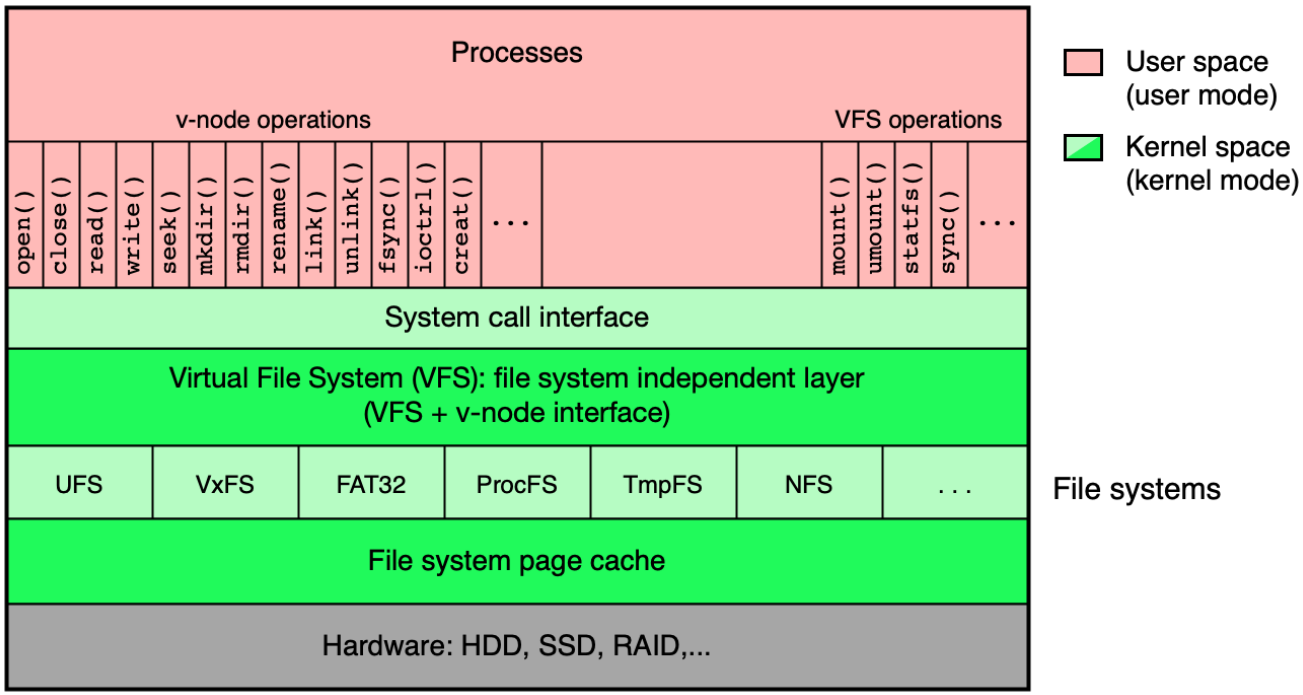
https://courses.fit.cvut.cz/BI-OSY

Obsah přednášky

1. Implementace FS v OS
   - Virtuální FS/Installable FS
   - Filesystem in userspace (FUSE)
   - Block cache/Page cache
   - Directory Name Look-up Cache(DNLC)
   - Block Read Ahead
   - Žurnálované FS
2. Moderní FS
   - B stromy a B+ stromy
   - Vlastnosti FS
     - ZFS (Zettabyte FS)
     - BTRFS (B-tree FS)

Implementace FS v OS

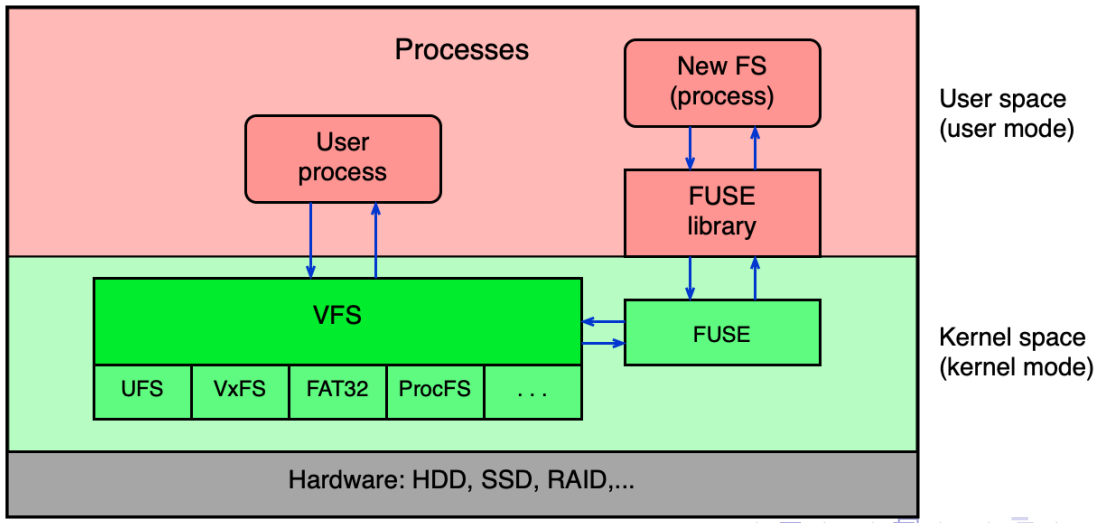
- **Virtual File System (VFS)**
  - Framework v rámci, kterého jsou v OS unixového typu implementovány konkrétní systémy souborů.
  - Vrstva jádra OS, která představuje rozhraní mezi procesy a jednotlivými implementacemi konkrétních FS.



- **Installable File System (IFS)**
  - API v IBM OS/2 a v MS Windows, které umožňuje OS rozpoznat a nahrát ovladač pro konkrétní FS.

Implementace FS v OS

- **Filesystem in Userspace (FUSE)**
  - Rozhraní v OS unixového typu, které umožňuje neprivilegovaným uživatelů vytvářet své vlastní FS bez nutnosti modifikovat jádro OS.
  - FUSE obsahuje
    - ★ modul jádra OS: samotná implementace FUSE,
    - ★ knihovna: rozhraní k modulu.
  - Nový FS implementovaný pomocí FUSE
    - ★ Implementován jako proces, ve kterém se musíme definovat standardní operace nad novým FS.
    - ★ Běžný uživatel může manipulovat s tímto FS pomocí standardních knihovních funkcí/příkazů.
  - Vhodný pro implementaci pseudo FS (FS, který existuje pouze v hlavní paměti) a data uložit prostřednictvím existujících FS.



Implementace FS v OS

- Přístup na disk je mnohem pomalejší než přístup do hlavní paměti
  ⇒ OS optimalizuje přístup do FS různými způsoby
  - Využívá volnou hlavní paměť jako skrytou paměť (cache) pro
    - ★ datové bloky FS (např. Block cache/Page cache),
    - ★ metadata FS (např. Directory Name Look-up Cache).
  - Používá přednačítání datových bloků FS (např. Block Read Ahead) na základě principu prostorové lokality při přístupu k datům.
- **Block cache/Page cache**
  - Velikost datových bloků FS je obvykle navržena tak, aby byla násobkem velikosti stránek ve virtuální stránkované paměti.
  - Block cache/Page cache je množina nedávno používaných datových bloků z FS (obsah souborů/adresářů) uložená v hlavní paměti z důvodu zlepšení výkonu FS.
  - OS si udržuje informaci o všech blocích načtených do hlavní paměti.
  - Když proces čte obsah adresáře/souboru, OS hledá bloky (obsah) nejdříve v hlavní paměti, pokud zde nejsou, pak je načte z FS.
  - Když proces modifikuje obsah adresáře/souboru, pak existují dva přístupy
    - ★ write-behind cache: modifikace je zapsána ihned do datového bloku v hlavní paměti a se zpožděním do FS (běžně používané u FS na interních datových úložištích),
    - ★ write-through cache: modifikace je současně zapsána do hlavní paměti i do FS (běžně používané u FS na přenosných datových úložištích typu flash-disk).

Implementace FS v OS

- **Příklad: Page cache v Solarisu**
  - Informaci o aktuálním obsazení fyzické můžeme získat v OS Solaris pomocí modulárního debagreru jádra prostřednictvím příkazu `mdb -k`.

```
root@solaris:~> echo "::memstat" | mdb -k
Page Summary                Pages                Bytes  %Tot
----------------- ----------------  ----------------  ----
Kernel                     115198             449.9M   22%
ZFS Metadata                18159              70.9M    3%
ZFS File Data              135738             530.2M   26%
Anon                       102981             402.2M   20%
Exec and libs                3869              15.1M    1%
Page cache                  26831             104.8M    5%
Free (cachelist)                3                12k    0%
Free (freelist)            101709             397.3M   19%
Total                      524159               1.9G
```

  - V systému jsou používány dva systémy souborů: ZFS a UFS.
  - Ve výpisu vidíme následující informace
    - Kernel: paměti alokovaná jádru OS.
    - ZFS Metadata: skrytá paměť systému souborů ZFS pro metadata
    - ZFS File Data: skrytá paměť systému souborů ZFS pro datové bloky.
    - Anon: anonymní paměť (stránky procesů, které nesouvisí s FS, např. zásobníky, haldy,...).
    - Exec and libs: bloků binárních programů a knihoven.
    - Page cache: skrytá paměť systému souborů UFS pro datové bloky.
    - Free (cachelist): volné stránky s použitelným obsahem (např. soubory, které aktuálně nejsou nikým používány, ale v budoucnu by opět mohly být použity,...).
    - Free (freelist): volné stránky bez použitelného obsahu (např. zásobníky, haldy ukončených procesů,...).

Implementace FS v OS

- **DNLC (Directory Name Look-up Cache)**
  - Skrytá paměť v OS Solaris, která obsahuje jména nedávno používaných souborů/adresářů a jejich v-nody (datova struktura VFS).
  - Při následujícím přístupu k souboru/adresáři se použije informace z DNLC, nikoliv z datových bloků FS.
- **Příklad: DNLC v Solarisu**
  - Pokud vytvoříme adreáře `A/B/C/D/E/F/G/H/I/J/K/L` a restartujeme OS, aby se vyčistila DNLC

```
user@solaris:~$ mkdir -p A/B/C/D/E/F/G/H/I/J/K/L

user@solaris:~$ sudo reboot
```

  - Pak můžeme zjistit vliv DNLC na rychlost přístupu k souborů/adresářům.

```
user@solaris:~$ time ls -l A/B/C/D/E/F/G/H/I/J/K/L
total 0

real    0m0.015s
user    0m0.001s
sys     0m0.012s

user@solaris:~$ time ls -l A/B/C/D/E/F/G/H/I/J/K/L
total 0

real    0m0.002s
user    0m0.000s
sys     0m0.001s
```

Implementace FS v OS

- **Přednačítání datových bloků FS (Block Read Ahead)**
  - Předpokládá se, že pro přístup k datům platí princip prostorové lokality.
  - Při požadavku čtení konkrétního datového bloku se z FS do paměti načte současně tento blok a několik následujících bloků.
  - Tímto přístupem lze zlepšit např. sekvenční čtení nebo využití datového úložiště typu RAID.
- **Příklad: Block Read Ahead v UFS**
  - UFS má velikost bloku FS fixně nastavenou na 8 kB a odpovídá velikosti stránky na CPU SPARC.
  - Parametr UFS `maxcontig` definuje, kolik bloků se bude číst/zapisovat při jedné operaci čtení/zápisu.
  - Příkaz `fstyp` v OS Solaris vypíše aktuální hodnoty parametrů UFS.

```
root@solaris:~> fstyp -v  /dev/rdsk/c2t1d0s2  | head
ufs
magic   11954   format  dynamic time    Thu May  9 16:21:46 2019
sblkno  16      cblkno  24      iblkno  32      dblkno  760
sbsize  2048    cgsize  8192    cgoffset 64     cgmask  0xffffffc0
ncg     213     size    10461696        blocks  10303204
bsize   8192    shift   13      mask    0xffffe000
fsize   1024    shift   10      mask    0xfffffc00
frag    8       shift   3       fsbtodb 1
minfree 1%      maxbpg  2048    optim   time
maxcontig 32    rotdelay 0ms    rps     120
...
```

  - Novou hodnotu parametru `maxcontig` může administrátor nastavit příkazem `tunefs`.

```
root@solaris:~> tunefs -a 128  /dev/rdsk/c2t1d0s2  | head
```

Implementace FS v OS

- **Ochrana dat při pádu systém/výpadku napájení**
  - Vytvoření/modifikace souboru/adresáře se obvykle skládá z několika kroků.
    - ★ Modifikace struktur související s volným prostorem (i-nody, datové bloky,...).
    - ★ Modifikace obsahu adresáře, ve kterém se soubor/adresář má vytvořit.
    - ★ Modifikace metadat (položky adresáře, i-nodu/položky v MFT,...).
    - ★ Modifikace dat v datových blocích.
  - Pokud všechny tuto kroky nejsou dokončeny, FS může zůstat v nekonzistentním stavu.
  - Při následné opravě FS (např. pomocí příkazu `fsck` v OS unixového typu) můžeme o některá data přijít.
- **Žunrálovaný FS (Journaling FS)**
  - Snaží se ochránit FS před nekonzistencí a ztrátou dat.
  - FS si alokuje na disku speciální oblast "journal", do které si v předstihu zapíše záznam o změnách, které se budou následně provádět. Potom, co se změny úspěšně zapíší do FS, je záznam z "journalu" odstraněn.
  - V případě havarie, po zotavení systému se "přehrají" záznamy z "journalu" a FS vrátí do konzistentního stavu.
  - "Journaling" je implementován ve většině současných FS (např. UFS, EXT2/3/4, NTFS,...).
  - Z důvodu výkonu se do "journálu" většinou ukládají pouze metadata ale nikoliv obsah datových bloků.
  - "Journal" s nemusí vždy nacházet na stejném disku jako samotný FS, ale může být umístěn např. na rychlejším SSD.

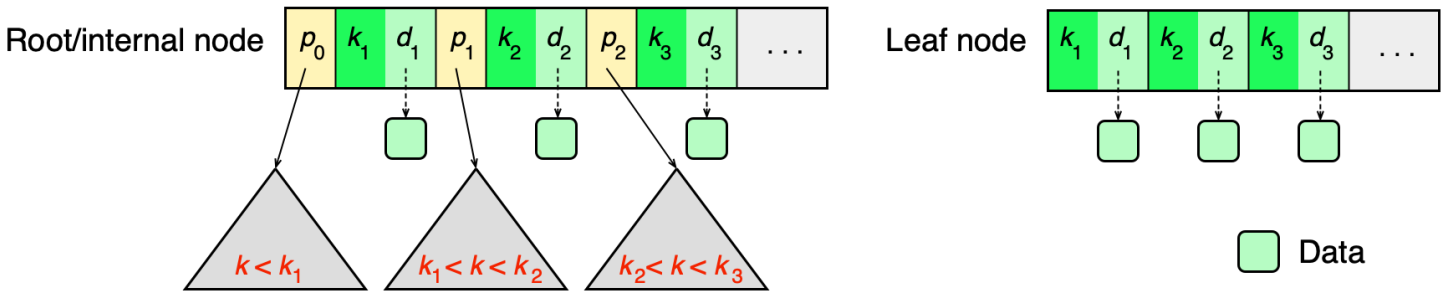
Moderní FS

- Většina klasických FS (např. UFS, EXT2/3/4, FAT32, NTFS,...) byla navržena v 80. a 90. letech minulého století.
- Díky vývoji datových úložišť (zvláště RAID, SSH), navýšení diskových kapacit a CPU výkonu systémů začaly vznikat moderní FS jako např. ZFS (Zettabyte FS), BTRFS (B-tree FS), APFS (Apple FS),...
- Mezi jejich důležité vlastnosti patří
  - Při implementaci FS se používají B stromy/B+ stromy.
  - Většinou reprezentují kombinaci SW RAIDu a FS.
  - Integrita dat (FS je neustále v konzistentním stavu).
    - ★ Redundance pomocí RAID ⇒ ochrana proti výpadkům HW komponent.
    - ★ Copy-on-write transactional object model ⇒ ochrana dat proti výpadku napájení,....
    - ★ Kontrolní součty dat/metadat ⇒ detekce a oprava dat/metadat.
  - Podpora efektivního vytváření "clonů" a "snapshotů".
  - Podpora šifrování dat.
  - Podpora kompresi dat a deduplikaci datových bloků.
  - Jednodušší administrace FS (obvykle se vystačí pouze s několika příkazy, např. `zpool` a `zfs` v ZFS).

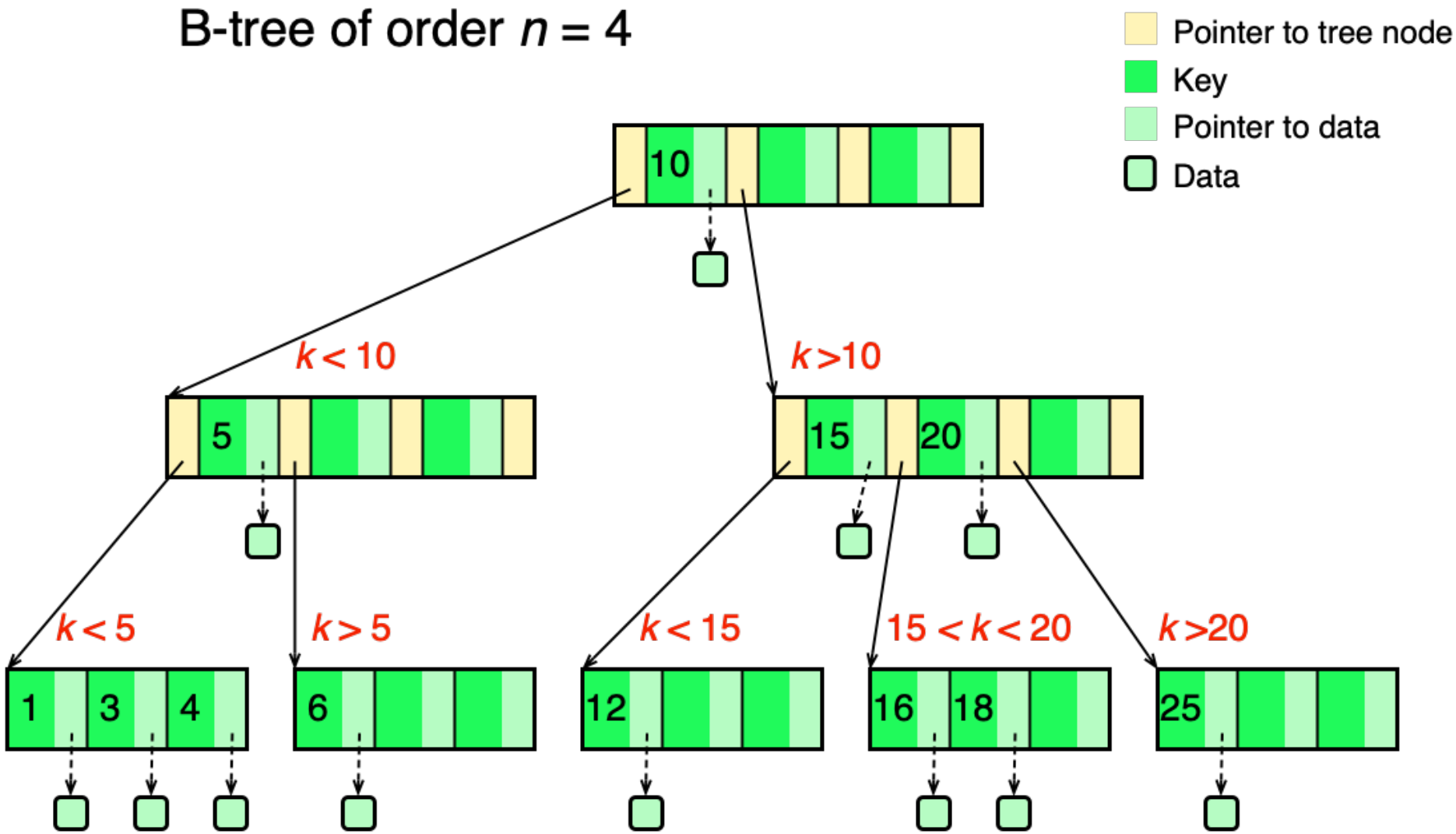
Moderní FS

- **B strom řádu n**
  - Kořen stromu
    - Má nejméně 2 a nejvíce n potomků, pokud není listem.
    - Obsahuje nejméně 0 a nejvýše $n - 1$ položek.
  - Vnitřní uzel stromu
    - Má nejméně $\lceil n/2 \rceil$ a nejvýše n potomků.
    - Obsahuje nejméně $\lceil n/2 \rceil - 1$ a nejvýše $n - 1$ položek.
  - List stromu
    - Všechny cesty od kořene k listům jsou stejně dlouhé.
    - Obsahuje nejméně $\lceil n/2 \rceil - 1$ a nejvýše $n - 1$ položek.
  - Data v uzlech stromu tvoří uspořádanou posloupnost
    - $p_0, [k_1, d_1], p_1, [k_2, d_2], p_2, \ldots$ (kořen/vnitřní uzel),
    - $[k_1, d_1], [k_2, d_2], \ldots$ (list stromu),

    kde k_i označují klíčové atributy ukládaných dat d_i, takové, že pro ně jde zavést relace uspořádání $<$ a p_i označují odkazy na podstromy. Pro každé k_i a k_j, kde $i < j$ je $k_i < k_j$.
  - Pro každé k v podstromě odpovídajícím odkazu p_{i-1} platí $k < k_i$.
  - Pro každé k v podstromě odpovídajícím odkazu p_i platí $k > k_i$.

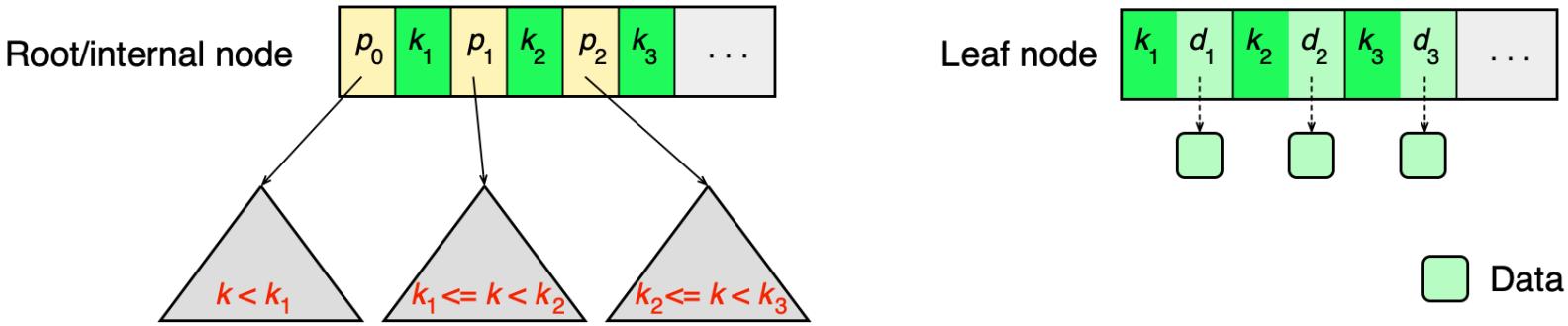


Moderní FS

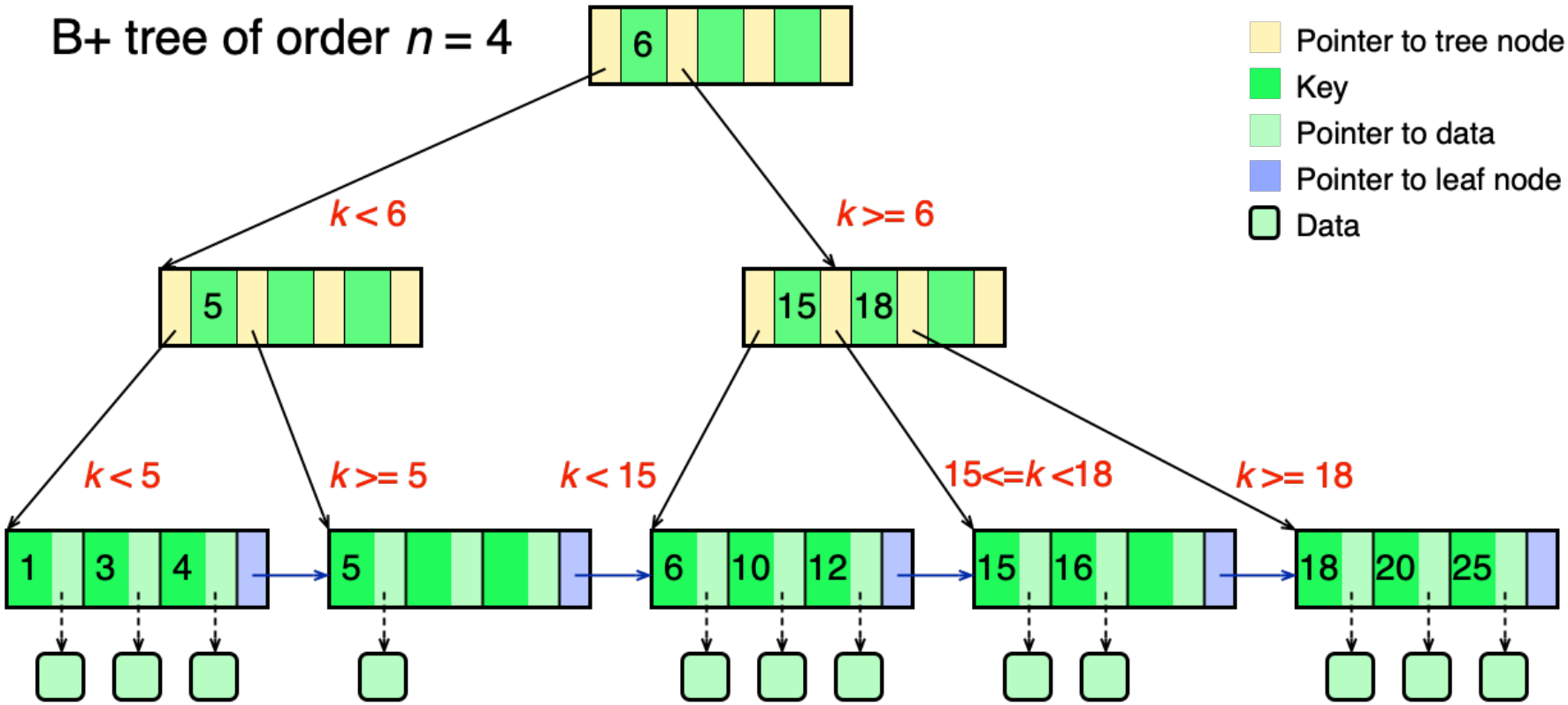

B-tree of order $n = 4$
Pointer to tree node
Key
Pointer to data
Data
10
$k < 10$
$k > 10$
5
15
20
$k < 5$
$k > 5$
$k < 15$
$15 < k < 20$
$k > 20$
1
3
4
6
12
16
18
25

Moderní FS

- **B+ stromy**
  - Hlavní rozdíl od B stromů je, že v kořenu/vnitřních uzlech jsou uloženy pouze klíče a odkazy na podstromy a v listech stromu jsou uloženy klíče s daty. Ostatní vlastnosti má stejné jako zmiňovaný B-strom.
  - B+ strom se ve skutečnosti realizuje tak, že je vždy ve všech listech uložen kromě vlastních klíčů a dat také odkaz (ukazatel) na následujícího sourozence.



Moderní FS

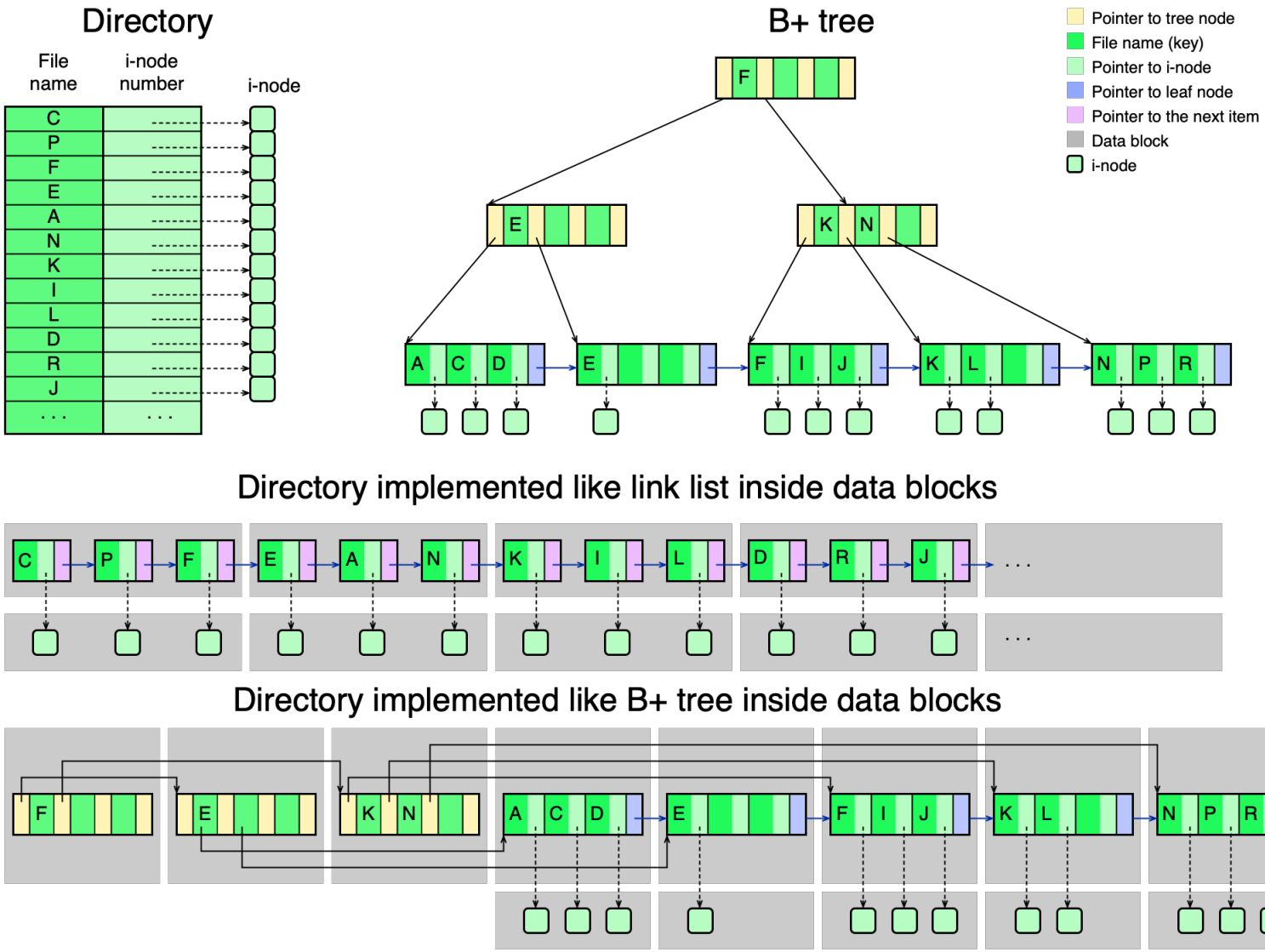

B+ tree of order n = 4
6
5
15
18
k < 6
k >= 6
k < 5
k >= 5
k < 15
15<=k <18
k >= 18
1
3
4
5
6
10
12
15
16
18
20
25
Pointer to tree node
Key
Pointer to data
Pointer to leaf node
Data

Moderní FS

- **Vlastnosti B stromů/B+ stromů**
  - Vložení/smazání/hledání prvku má časovou složitost $O(\log(N))$, kde N je počet záznamů ve stromě.
  - V uzlech je uloženo více klíčů
    $\Rightarrow$ uzel může být uložen ve skryté paměti L1/L2/L3 jako blok, v hlavní paměti jako stránka, na disku/FS jako sektor/datový blok.
  - Uzly s klíči mohou být nahrány do paměti, zatímco data zůstávají na disku.

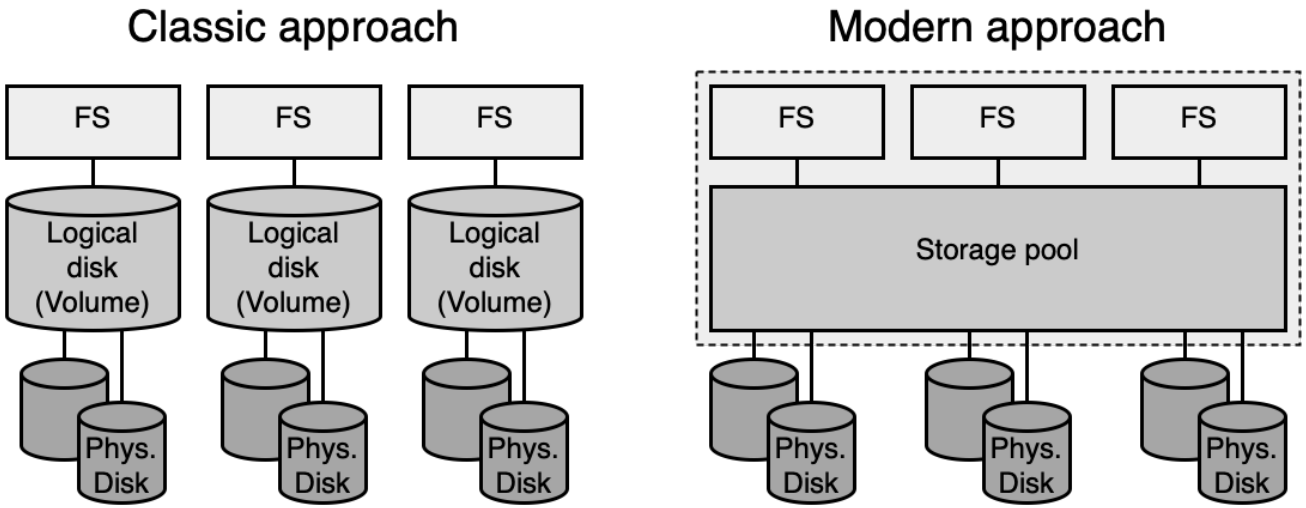
Moderní FS

- **Příklad: Použití B+ stromů při implementaci adresářů**



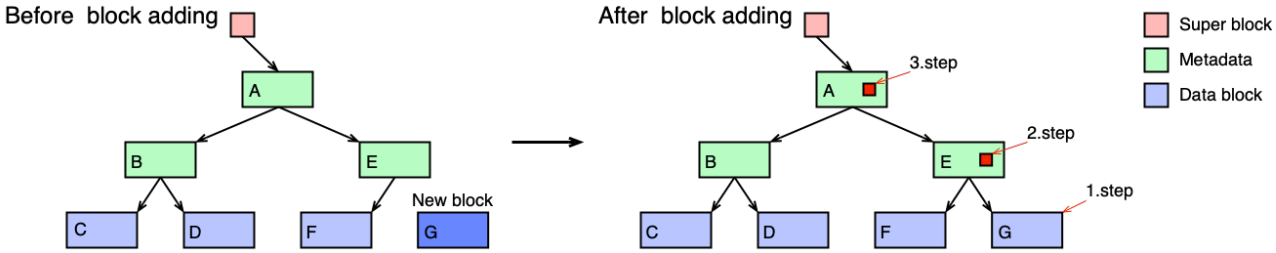
Moderní FS

- **Kombinace SW RAIDu a FS**
  - Klasický přístup
    - ★ SW RAID a FS jsou oddělené (dvě nezávislé SW vrstvy).
    - ★ V SW RAIDu se vytvoří z fyzických disků logický disk (Volume) s příslušnými vlastnostmi a v něm se následně vytvoří příslušný FS.
  - Moderní přístup
    - ★ SW RAID a FS jsou implementovány jako celek (jedna SW vrstva).
    - ★ Fyzické disky se zařadí do "poolu", který představuje konkrétní typ RAIDu.
    - ★ V rámci poolu se pak vytváří jednotlivé FS.
      ⇒ Efektivnější využívání a sdílení kapacity fyzických disků.
      ⇒ Jednodušší administrace (zvětšování/zmenšování jednotlivých FS).

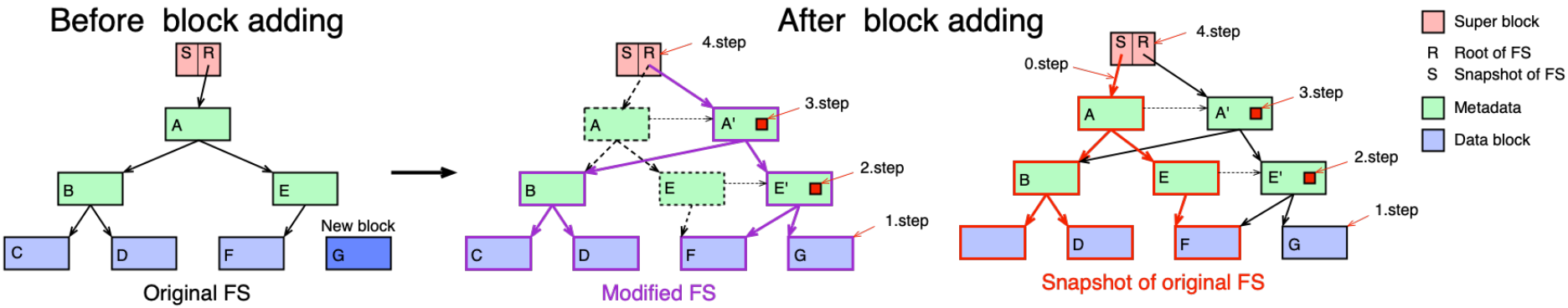


Moderní FS

- **Integrita dat**
  - Klasický přístup
    - ★ Při změně dat/metadat ve FS se původní data přepíší novou hodnotu.
    - ★ Změna probíhá v několika krocích (např. zvětšení obsahu souboru o blok G ⇒ zápis bloku G, zápis metadat A a E).
    - ★ Pokud všechny změny neproběhnou, pak FS skončí v nekonzistentním stavu.



  - Copy-On-Wrire (COW) přístup
    - ★ Původní data se nepřepisují, ale vytvoří se kopie, která se teprve modifikuje.
    - ★ Po provedení všech změn se atomicky přepíše ukazatel v Super bloku ⇒ FS je neustále konzistentní.

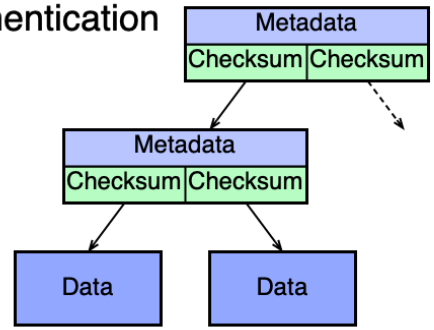


Moderní FS

- **Různé typy poškození dat**
  1. Bit corruption: Původní obsah sektoru/datového bloku byl modifikován (např. elektromagnetickým zářením,...). Data byla zapsána do sektoru/bloku, který je poškozený (obsah bloku se liší od zapisovaných data).
  2. Lost writes: Disková operace "write" se neprovedla, ale úspěšné dokončení bylo potvrzeno.
  3. Misdirected writes: Data byla zapsána do špatného datového bloku.
  4. Torn writes: Data byla zapsána pouze částečně, ale úplné dokončení bylo potvrzeno.
- **Zabezpečení dat**
  - Klasický přístup
    - ★ Klasické FS se spoléhají pouze na kontrolní součty (ECC), které zabezpečují obsah sektoru. V lepším případě používají ještě kontrolní součty v rámci FS na zabezpečení obsahu datových bloků.
    - ★ Kontrolní součty jsou typicky umístěné společně s daty v sektoru/bloku.
      ⇒ Detekuje/opraví pouze poškození dat typu 1.
  - ZFS data authentication
    - ★ Kontrolní součty jsou umístěné v rodičovských metadatech (Merkle stromu).
      ⇒ Data a kontrolní součty jsou oddělené. Detekuje/řeší poškození dat typu1-4.

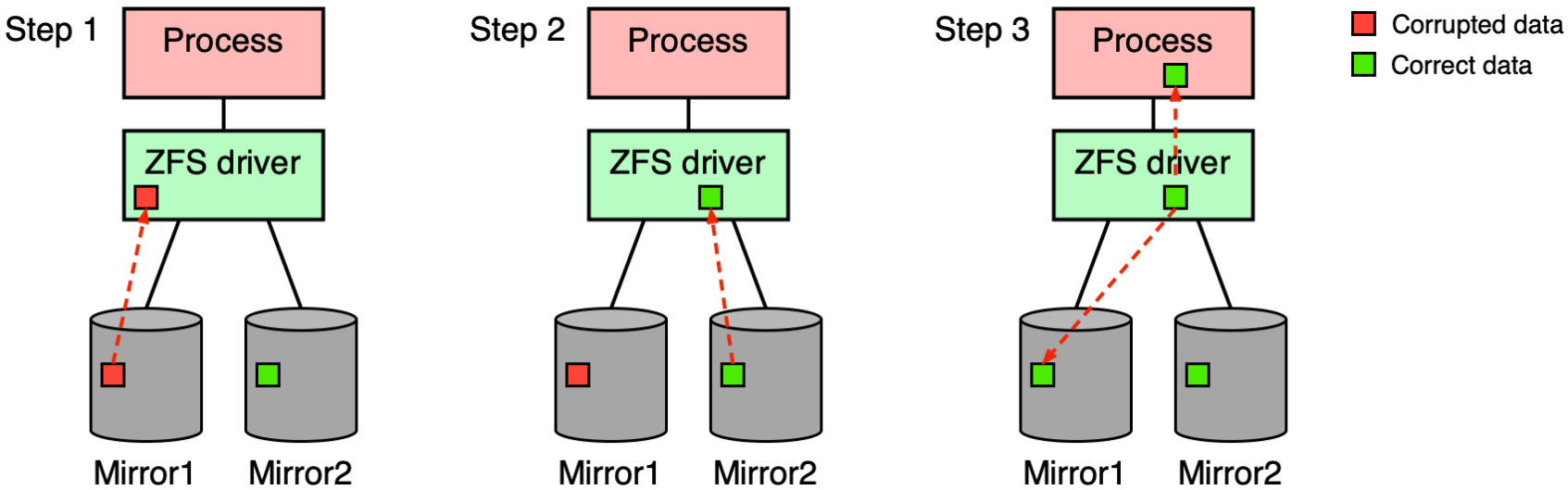
Sector checksums

ZFS data authentication



Moderní FS

- **Automatická oprava dat (Self-healing data)**
  - Díky kontrolním součtům je FS schopný sám detekovat při přístupu k datům chybu a díky redundanci dat v RAIDu ji následně i sám opravit.
- **Příklad: Detekce a oprava chyby v ZFS - RAID 1 (zrcadlení).**
  1. Ovladač ZFS načte datový blok z prvního zrcadla a detekuje ho jako poškozený.
  2. Ovladač ZFS načte datový blok z druhého zrcadla a detekuje ho jako správný.
  3. Ovladač ZFS předá správná data procesu a současně opraví poškozená data na prvním zrcadle.



Použité zdroje

1. A. S. Tanenbaum, H. Bos: *Modern Operating Systems (4th edition)*, Pearson, 2014.
2. W. Stallings: *Operating Systems: Internals and Design Principles (9th edition)*, Pearson, 2017.
3. A. Silberschatz, P. B. Galvin, G. Gagne: *Operating System Concepts (9th edition)*, Wiley, 2012.
4. R. McDougall, J. Mauro: *Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture (2nd edition)*, Prentice Hall, 2006.
5. *To FUSE or Not to FUSE: Performance of User-Space File Systems.* [Online]. Available: https://www.usenix.org/system/files/conference/fast17/fast17-vangoor.pdf.[Accessed:8-Mayr-2019].
6. J. Bonwick, M. Ahrens, V. Henson, M. Maybee, M. Shellenbaum: *The Zettabyte File System*.[Online]. Available: https://www.academia.edu/20291242/Zfs_overview [Accessed:8-Mayr-2019].