HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK

MŰEGYETEM 1782

Budapest,
2025.02.20.

SZÁMÍTÓGÉP ARCHITEKTÚRÁK

Háttértárak

Horváth Gábor, Belső Zoltán

BME Hálózati Rendszerek és Szolgáltatások Tanszék
ghorvath@hit.bme.hu, belso@hit.bme.hu

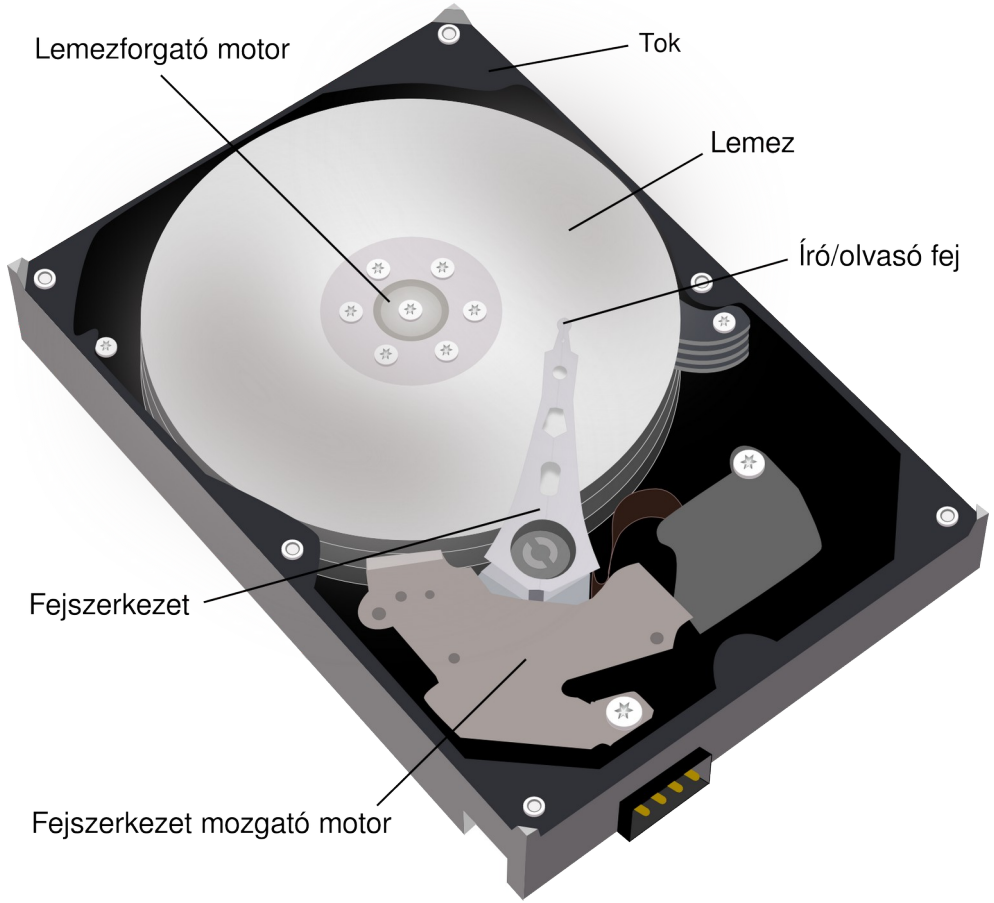
HÁTTÉRTÁRAK

- Minden számítógép részét képezik
- Teljesítményük a rendszer egészének teljesítményére kihat
- Fontos szerepet játszanak a virtuális tárkezelésben
- Amiről tanulunk:
  - HDD: **merevlemez alapú háttértárak**
  - SSD: **félvezető alapú háttértárak**
  - Van még más is:
    - Optikai lemezek: sok, a HDD-nél látott megoldás visszaköszön
    - Pen-drive: alapja ugyanaz a flash memória, ami az SSD-ben van
    - Stb.

Adattárolás merevlemezen (HDD)

FORGÓLEMEZES ADATTÁROLÓK

- Részei:
  - Lemez(ek)
    - 1 vagy több adathordozó felület
    - Ezen tároljuk az adatot
      → mágneses elven
  - A fej
    - Olvassa/írja az adatokat
- Adatok elérése:
  1) A fejet a kívánt radiális távolságba toljuk
     - Ez a **seek**
  2) A lemezt a kívánt pozícióba forgatjuk
  3) Megtörténik az írás/olvasás
     - HDD → mágneses elven
     - CD/DVD/BR → optikai elven



- https://youtu.be/3owqvmMf6No

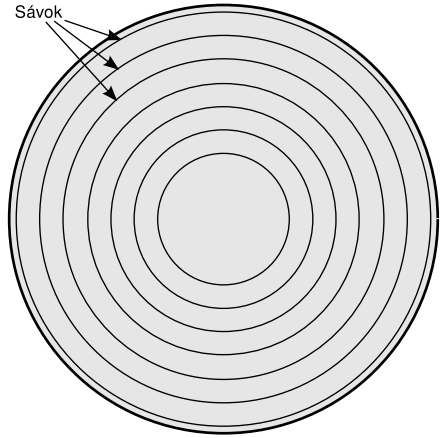
AZ ADATOK SZERVEZÉSE

- Domináns megoldás: fix méretű blokkok
  → **szektor** (tipikusan 512 bájt)
- Hivatkozás a szektorokra: lineáris címmel (0-tól)
  - **Logical Block Addressing (LBA)**
  - Leképzés a lemezek geometriájára → a diszk dolga

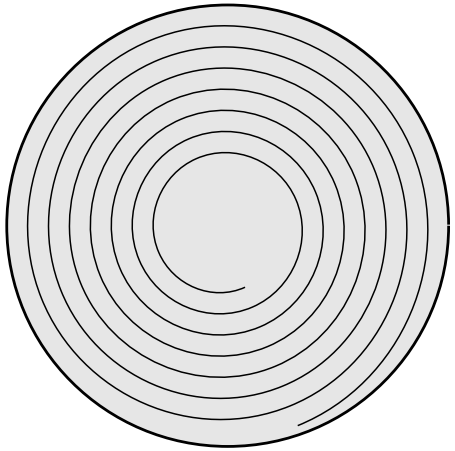
Szektorok elhelyezkedése:

- Merevlemezen:
  - Koncentrikus körökben
    → neve: **sáv**
  - Forgás: állandó szögsebességgel
    CAV: Constant Angular Velocity
- Optikai lemezeken:
  - Spirálisan
  - Forgás: állandó lineáris sebességgel a spirál mentén
    CLV: Constant Linear Velocity

Merevlemezek:



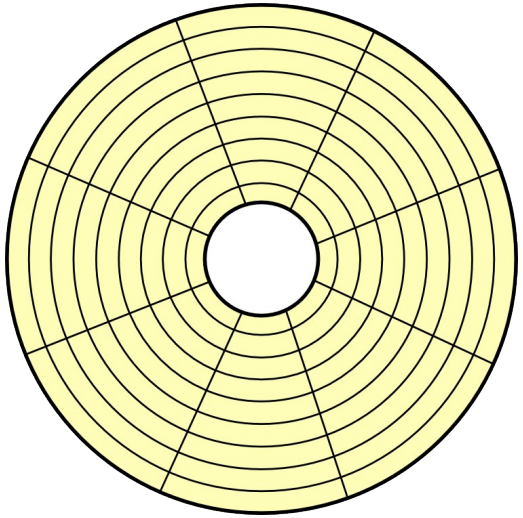
Optikai lemezek:



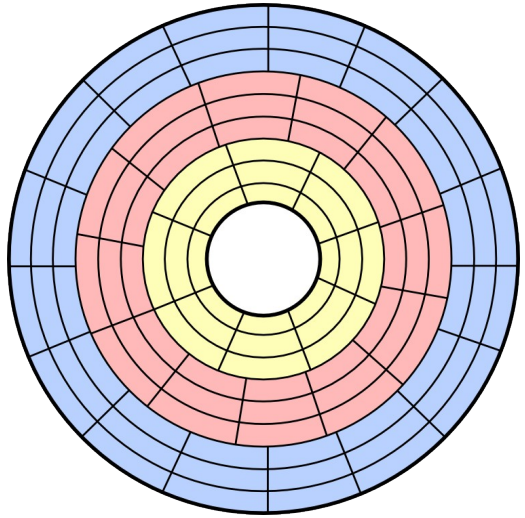
ZÓNA RENDSZERŰ ADATTÁROLÁS

- **Ha a sávonkénti szektorok száma fix** az egész diszkre:
  - Minden szektort ugyanannyi ideig tart elolvasni
  - A belső szektorokban nagyobb az adatsűrűség → a külsőkben pazarló
- **Ha az adatsűrűség fix** az egész diszkre:
  - Kifelé haladva minden sávban egyre több szektor lesz
  - Külső sávok felé haladva egyre gyorsabban kell olvasni
  - Sávonként olvasási sebességet váltani → nem megoldható
- Arany középút: nem váltunk sebességet minden sávban, csak zónánként
  → **zóna rendszerű adattárolás (ZBR, Zoned Bit Recording)**
  - Egy zónán belül minden sávban ugyanannyi szektor van
  - Egy zónán belül fix az adatleolvasási sebesség

ZBR nélkül:

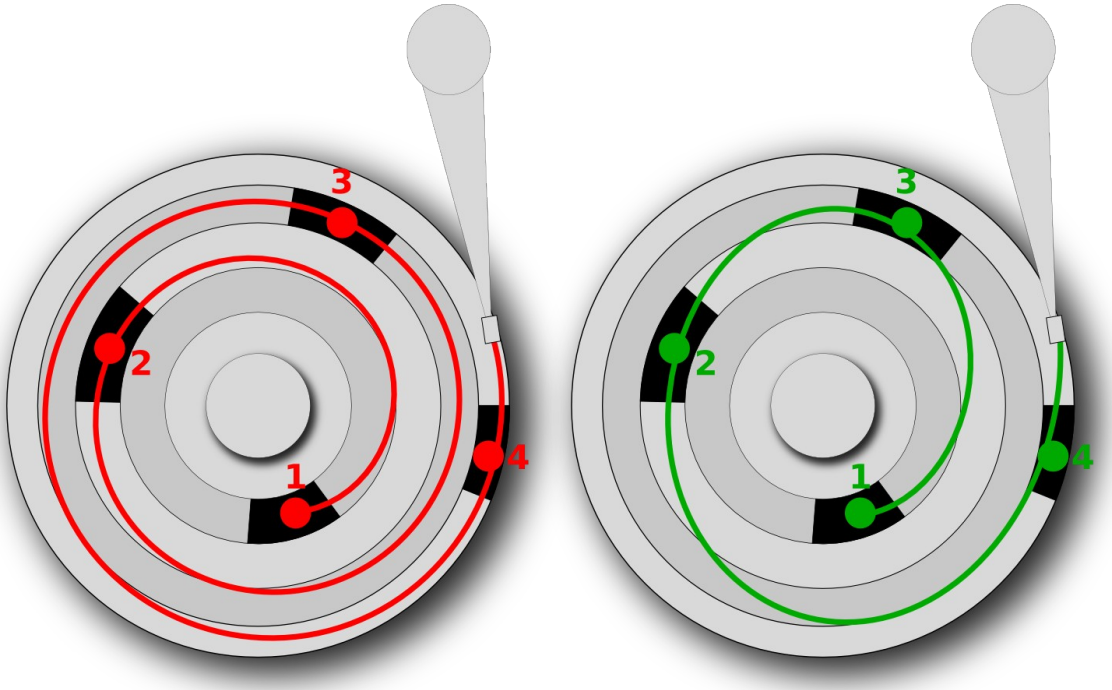


ZBR-rel:



MEREVLEMEZEK TELJESÍTMÉNYE

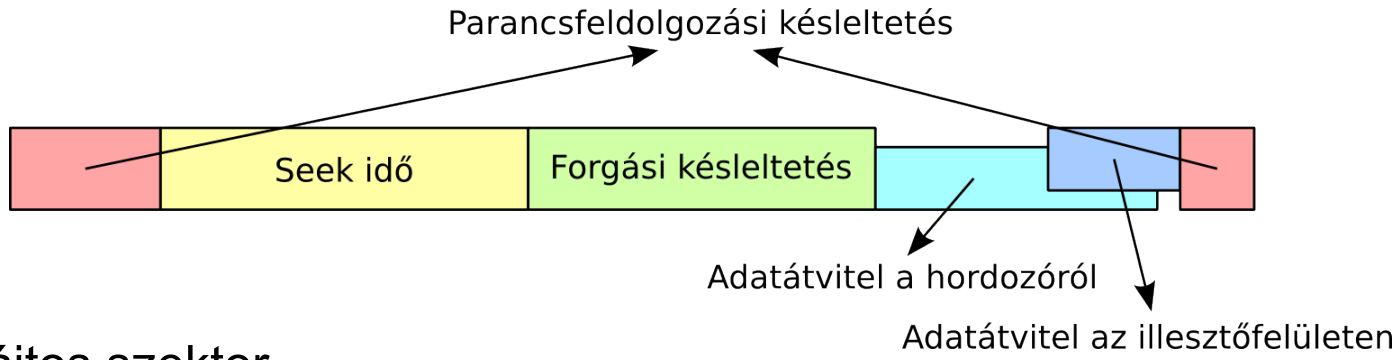
- Szeretnénk egy szektorsorozatot átvinni
  - A kérés a kiszolgálása előtt egy bufferbe kerül, ott várja a sorát
    - Régen: op. rendszer menedzselte
    - Ma: diszk menedzseli → **command queueing**
      - A számára kényelmes (és gyors) sorrendben szolgálja ki őket



Kiszolgálás érkezési sorrendben Kiszolgálási sorrend optimalizálása

MEREVLEMEZEK TELJESÍTMÉNYE

- **Kiszolgálási idő**
  - 5 komponensből áll:
- Példa:
  - Forgási sebesség: 10.000 fordulat/perc
  - Átlag seek idő: 4.5 ms,
  - Parancsfeldolgozási késleltetés: 0.3 ms
  - A keresett sávban legyen 600 db 512 bájtos szektor
  - Illesztőfelület sebessége legyen 100 MB/s



- Mennyi ideig tart egy 4 kB-os adatátvitel?
  - Forgási késleltetés: 1 fordulat: 6 ms.
    → Véletlen időpontban átlag 3 ms-ot kell várni, hogy a keresett adat fölé kerüljön a fej
  - Adat leolvasása: 4 kB = 8 szektor. 1 szektor átvitele: 6 ms / 600 = 0.01 ms.
  - Adat átvitele az illesztőfelületen: átlapolható!
    Leolvasási idő + az utolsó szektor átvitele: 512 bájt / 100 MB/s = 0.005 ms
  - Összesen: 0.3 + **4.5** + **3** + 0.08 + 0.005 = **7.885 ms**
- A forgási és a seek idő dominál!
- **Érdemes sok adatot egyszerre átvinni!** Seek és forgási idő csak egyszer adódik hozzá!

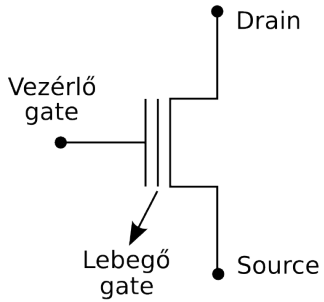
MEREVLEMEZEK TELJESÍTMÉNYE

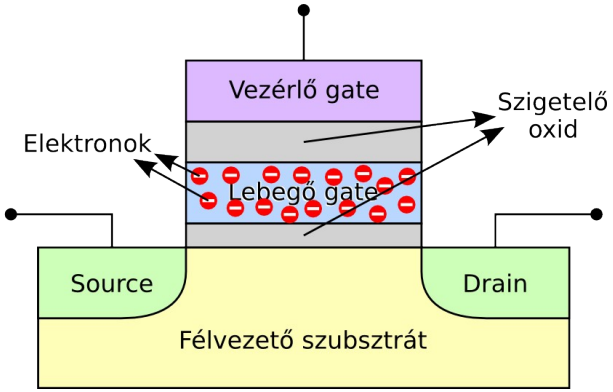
- **Átviteli sebesség (throughput)**
  - Kétféle megadás:
    - Kérések számában mérve: IOPS: [kiszolgált kérések száma/sec]
    - Adatsebességben mérve: IOPS x átlag átvitt adatmennyiség [byte/sec]
  - Kis blokkmérettel mérve: **véletlen átviteli sebesség**
    - Seek idő, forgási késleltetés dominál
  - Nagy, folytonos blokkméretekkel mérve: **folytonos átviteli sebesség**
    - Az adatleolvasási idő dominál
- Példa:
  - Véletlen (4 kB blokkméret):
    - Előbb kijött: 7.885 ms/kérés → 126.823 IOPS/s ≈ 0.52 MB/s
      → Nagyon lassú!
  - Folytonos (16 MB blokkméret):
    - Előbbihez hasonlóan: 335.485 ms/kérés → 2.98 IOPS/s ≈ 50 MB/s
      → Sokkal gyorsabb!

Félvezető alapú háttértárak (SSD)

EGY BIT TÁROLÁSA

- Flash memória alapja a lebegő gate-es tranzisztor
  - A lebegő gate-be elektronok zárhatók
    - Benne is maradnak (táp nélkül)
    - Megváltoztatják a tranzisztor viselkedését
- Adatok reprezentálása:
  - Egy tranzisztor 1 bitet tárol (egyelőre)
  - Vannak elektronok: **0**, nincsenek: **1**
- Váltás 1-ből 0-ba: **programozás**
  - Elektronokat tömünk a lebegő gate-be
- Váltás 0-ból 1-be: **törlés**
  - Kiszivattyúzzuk az elektronokat a lebegő gate-ből
- Mindkét művelet nagy stresszt jelent a tranzisztornak!
- **A programozás és törlés mellékhatása:**
  - A lebegő gate-be és onnan ki mászkáló elektronok beragadhatnak a szigetelőbe (örökre)
  - A szigetelőréteg egyre kevésbé fog szigetelni
  - **Öregedés!**



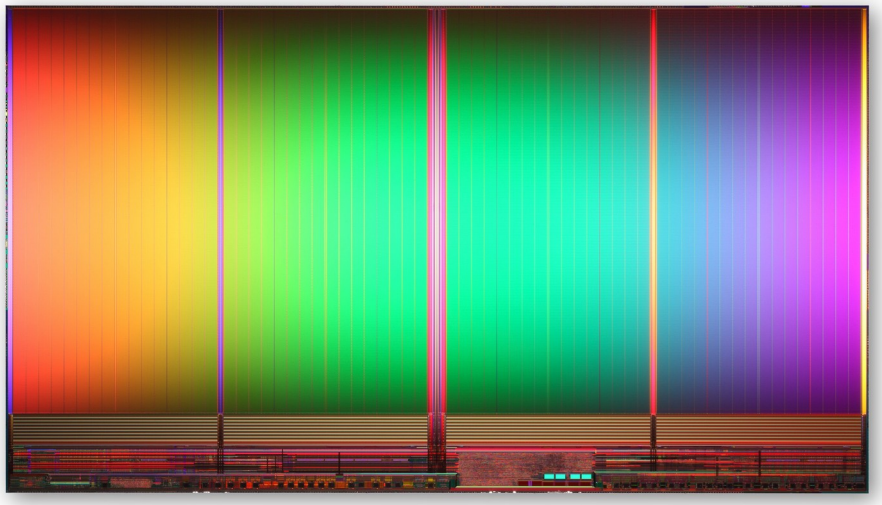


NÉHÁNY BIT TÁROLÁSA

- Eddig 2 töltöttségi állapot volt: nincs töltés → 1, van → 0
- Miért ne használnánk több töltöttségi szintet?
  - ***N* bit tárolásához 2^N töltöttségi szintet kell megkülönböztetni**
    - SLC flash: 1 tranzisztor → 1 bit
    - MLC flash: 1 tranzisztor → több (tipikusan 2) bit
  - Újabban a gyártók máshogy értelmezik az MLC-t:
    - SLC: N=1 – leggyorsabb, legdrágább
    - MLC: N=2
    - TLC: N=3
    - QLC: N=4 – leglassabb, legolcsóbb
- Minél nagyobb N, annál gyorsabban öregszik! Elviselt törlési ciklusok száma:
  - SLC: kb. 100.000
  - MLC: kb. 10.000
  - TLC: kb. 1.000 – 3.000
  - QLC: kb. 100 – 750 (!)
  - Példa: Intel SSD 670p 1TB, QLC, 144 rétegű, TBW = 320TB (Terabytes Written)

SOK BIT TÁROLÁSA

- A tranzisztorokat 2D rácsba rendezzük
- De hogyan?
- **NOR flash:**
  - Byte szinten címezhető
  - Utasításmemóriaként használható (BIOS, firmware, stb.)
- **NAND flash:**
  - Kezdetektől fogva háttértárnak szánták
  - Olvasás és írás egységei: **lapok**
    - Lap: a tárolómező egy sorának bitjei
  - Az egész tárolómező neve: **blokk**
  - **Törölni csak *teljes blokkot* lehet, lapokat egyesével *nem***
  - Tárolási hierarchia:
    - 1 lebegő gate-es tranzisztor: 1 – 3 bitet tárol
    - 1 lap: 512 byte – 8 kB
    - 1 blokk: 128 – 256 lap
    - 1 tárolósík: 1024 blokk
    - 1 szilíciumlapka: 1 – 4 tárolósík
    - 1 tok: 1 – 4 szilíciumlapka



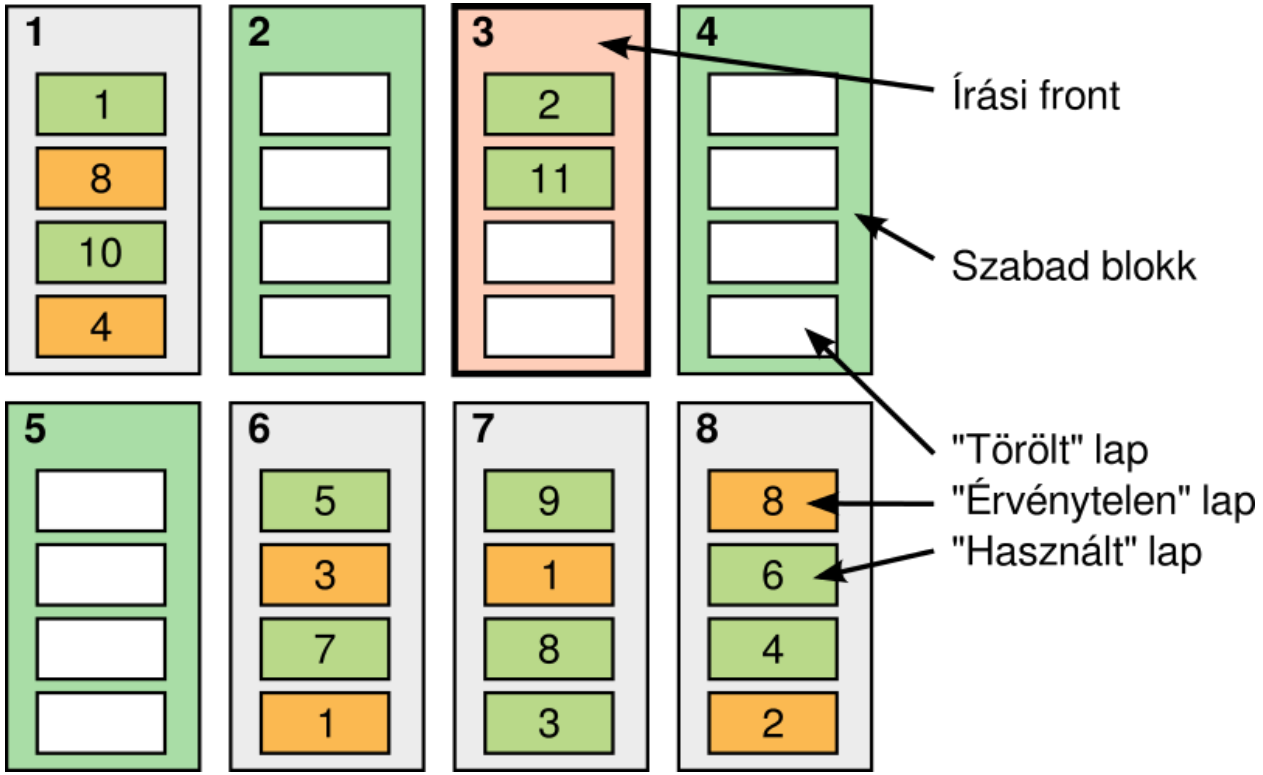
Az Intel 8 GB-os 2-bites MLC NAND flash lapkája

SZEKTORSZINTŰ MŰVELETEK

- **Olvasás**
  - LBA címéből → fizikai elhelyezkedés, kiolvasás, kész.
- **Írás**
  - HDD-nél: Ha egy fájlt többször mentünk, HDD felülírja a régit
  - SSD-nél: Nem tud lapot felülírni!!!
  - Minden új változatot új, szűz helyre tesz, a régit érvénytelennek jelöli
  - Új szűz lapok generálása: ha egy teljes blokkon már nincs hasznos adat, törli (akár 2 MB!)
  - Minden laphoz állapotot tárol:
    - „Használt”
    - „Érvénytelen”
    - „Törölt”

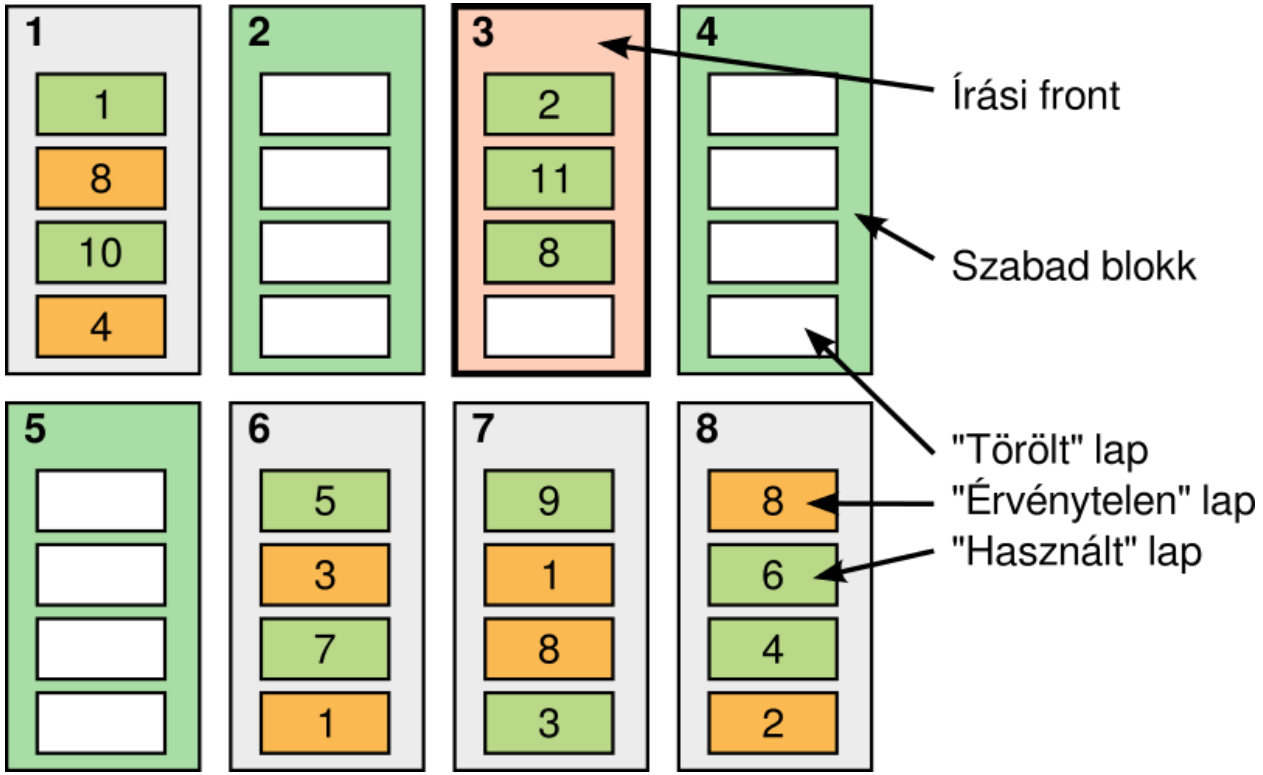
LAPÍRÁS MENEDZSMENT

- Minden írási művelet az **írási front**ra vonatkozik
- Ha az írási front megtelik, újat választ a törölt blokkok közül
- Példa: írási kérések a 8, 12, 1 LBA címekre



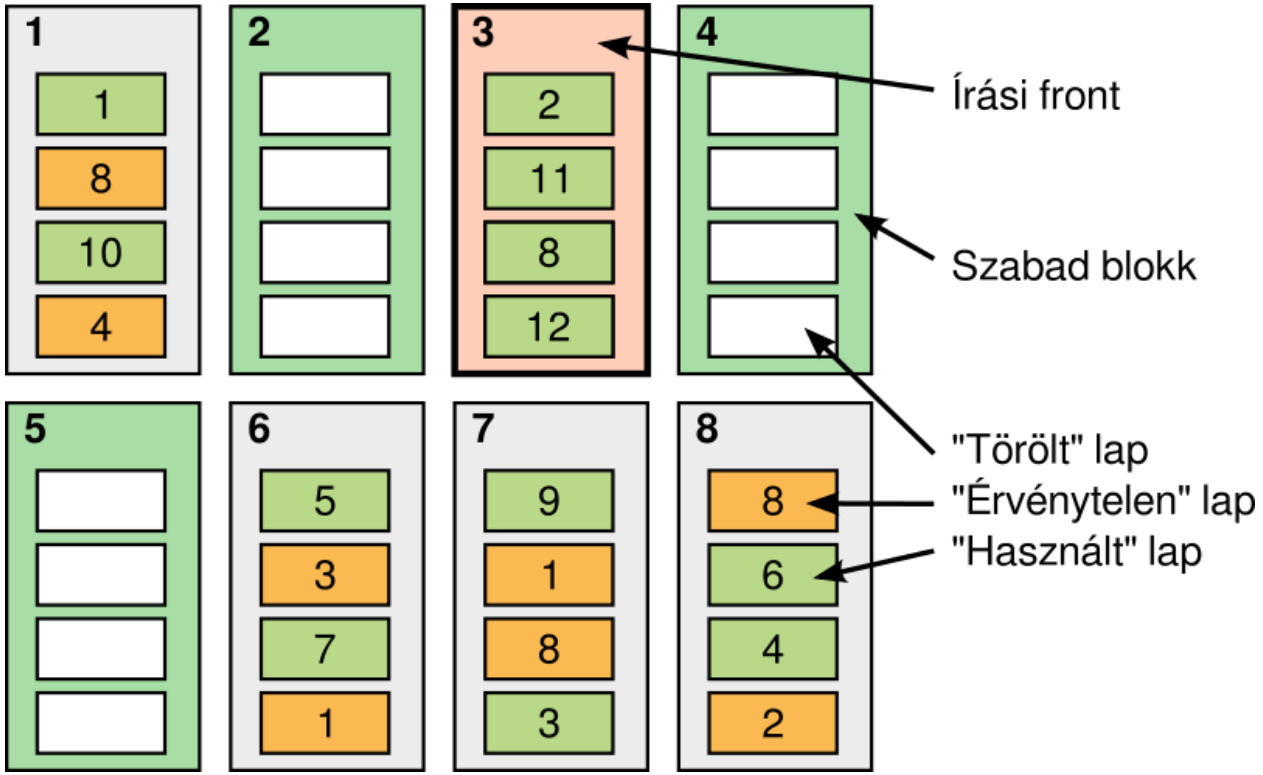
LAPÍRÁS MENEDZSMENT

- Minden írási művelet az **írási front**ra vonatkozik
- Ha az írási front megtelik, újat választ a törölt blokkok közül
- Példa: írási kérések a 8, 12, 1 LBA címekre



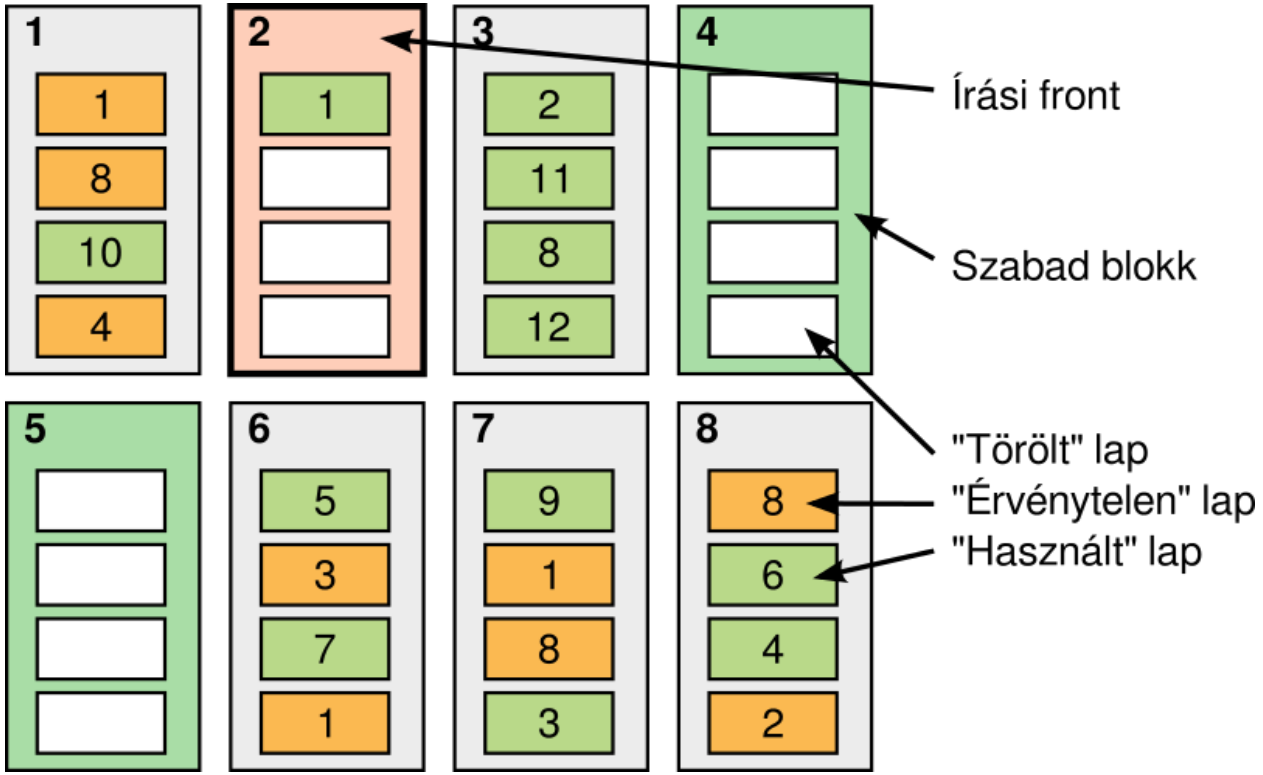
LAPÍRÁS MENEDZSMENT

- Minden írási művelet az **írási front**ra vonatkozik
- Ha az írási front megtelik, újat választ a törölt blokkok közül
- Példa: írási kérések a 8, 12, 1 LBA címekre



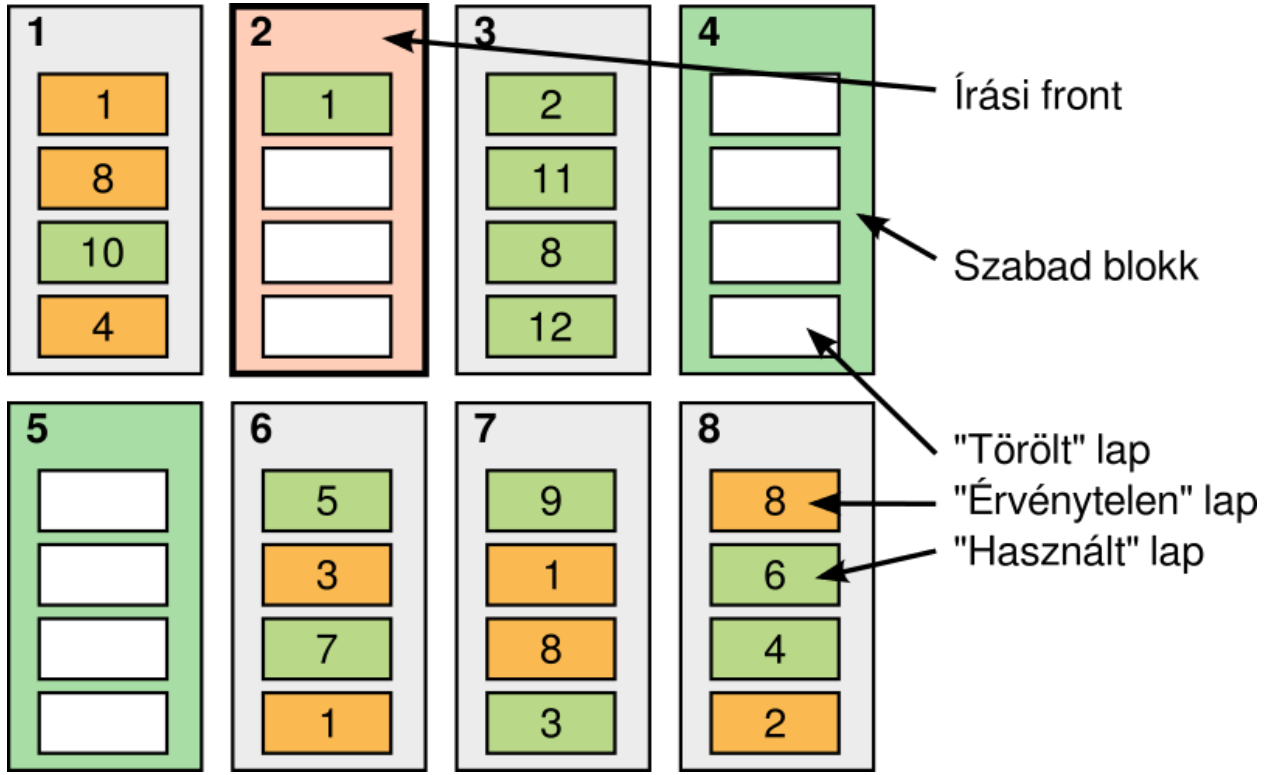
LAPÍRÁS MENEDZSMENT

- Minden írási művelet az **írási front**ra vonatkozik
- Ha az írási front megtelik, újat választ a törölt blokkok közül
- Példa: írási kérések a 8, 12, 1 LBA címekre



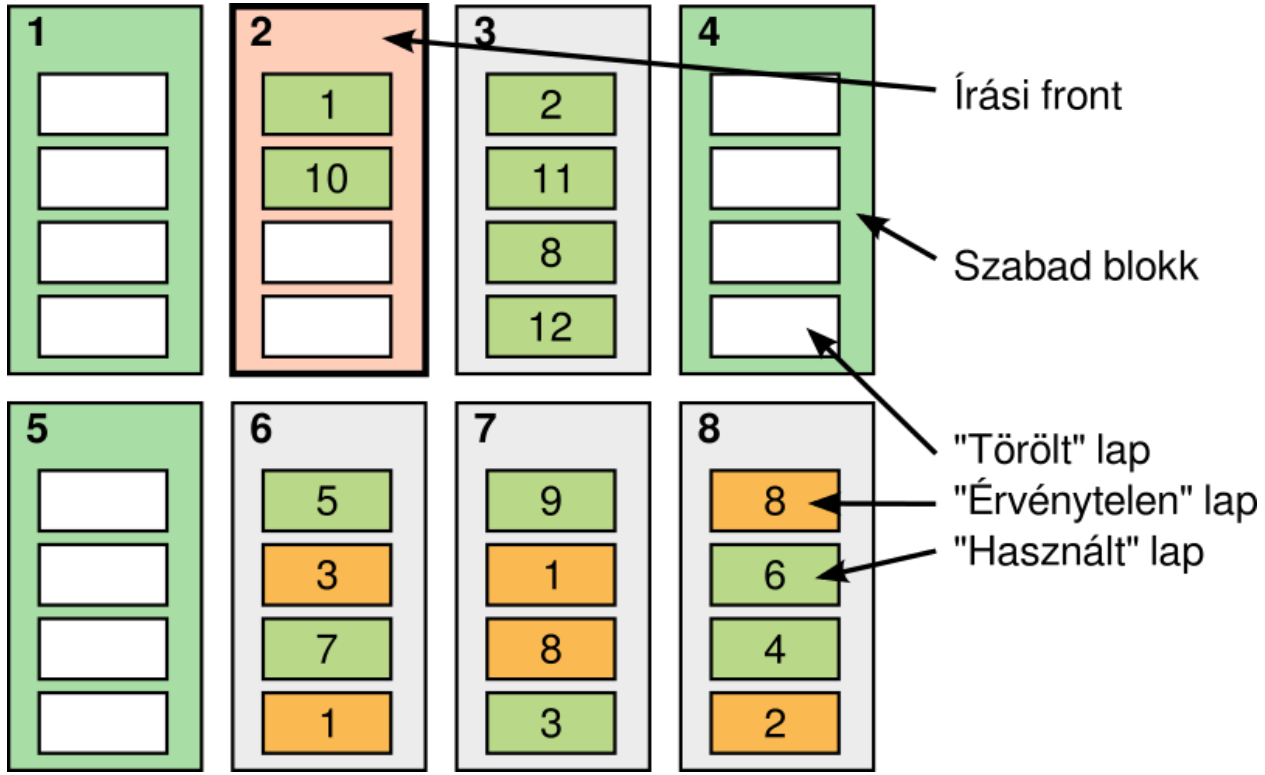
LAPÍRÁS MENEDZSMENT

- Ha fogynak törölt blokkok → **szemétgyűjtés**
  - Kinéz egy blokkot
  - „Használt” lapjait → írási frontra
  - Törli az egész blokkot
- Példa:



LAPÍRÁS MENEDZSMENT

- Ha fogynak törölt blokkok → **szemétgyűjtés**
  - Kinéz egy blokkot
  - „Használt” lapjait → írási frontra
  - Törli az egész blokkot
- Példa:



LAPÍRÁS MENEDZSMENT

- A működés következménye:
  - **Az SSD idővel lassul!**
  - Mert amikor új, nem kell szemetet gyűjtenie
- Korszerű SSD-k megoldása:
  - *Ráérő idejükben szemetet gyűjtögetnek*
    - Csínján kell bánni a lapok rendezgetésével – koptatja az SSD-t!
    - Csak akkor megy, ha van ráérő idő. Ha nincs (pl. terhelt szerver), drasztikus a lassulás.
  - *Kopás-kiegyenlítés*
    - Minden blokk egyformán kopjon (statikus és változékony egyaránt)
  - *Adattömörítés*
    - Ha az adatokat tömörítjük, kevesebbet kell írni (lassul az öregedés)
  - *Több írási kérés bevárása*
    - Hátha jön 1-2 kérés, ami ugyanarra a lapra vonatkozik
    - Ezek egyetlen írási művelettel végrehajthatók
- Működés következménye: *Többletírás*
  - A meghajtó többet ír, és **jobban kopik**, mint amennyit mi kezdeményezünk!!!

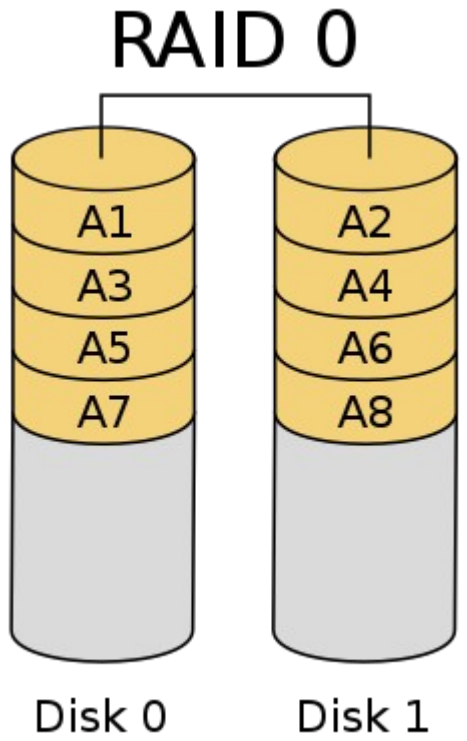
Tárolótömbök

MOTIVÁCIÓ

- SLED: Single Large Expensive Disk – Nagy tárolókapacitás, drágán
- Ötlet: Használjunk sok, olcsó diszket, és vonjuk őket össze logikailag!
- Probléma: hibatűrés
  - MTTF: Mean Time To Failure
  - Pl. Ha 1 diszkre MTTF = 100.000 óra, akkor 100 diszkes tömbre 100.000/100 = 1000 óra
    - Azaz: átlagosan 1000 óránként meghibásodik egy diszk
- Megoldás: redundancia
  - Legyen kisebb a hasznos kapacitás, mint a teljes → a különbségből a hibatűrést javítjuk
- Ez a RAID – Redundant Array of { Inexpensive | Independent } Disks
- A redundancia mikéntje alapján:
  - **RAID 0**, **RAID 1**, **RAID 2**, **RAID 3**, **RAID 4**, **RAID 5**, **RAID 6**
  - Ezek kombinációi: RAID 10, RAID 0+1

RAID 0

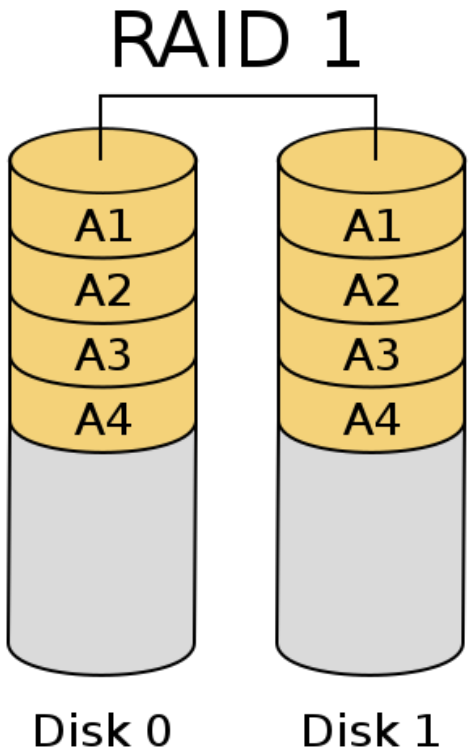
- RAID 0: a diszkek összefogása a kapacitás növelése érdekében, redundancia nélkül
- ...de nem akárhogy: striping
  → Blokkok felváltva a diszkek között
- Előnyök:
  - Nagy tárolókapacitás
    (felkínált kapacitás = diszkek kapacitásának összege)
  - Nagy átviteli sebesség – terhelés megoszlik a diszkek között
  - Egyszerű megvalósítás
- Hátrányok:
  - Alacsony rendelkezésre állás (MTTF)
  - Már 1 diszk hibája is katasztrofális



RAID 1

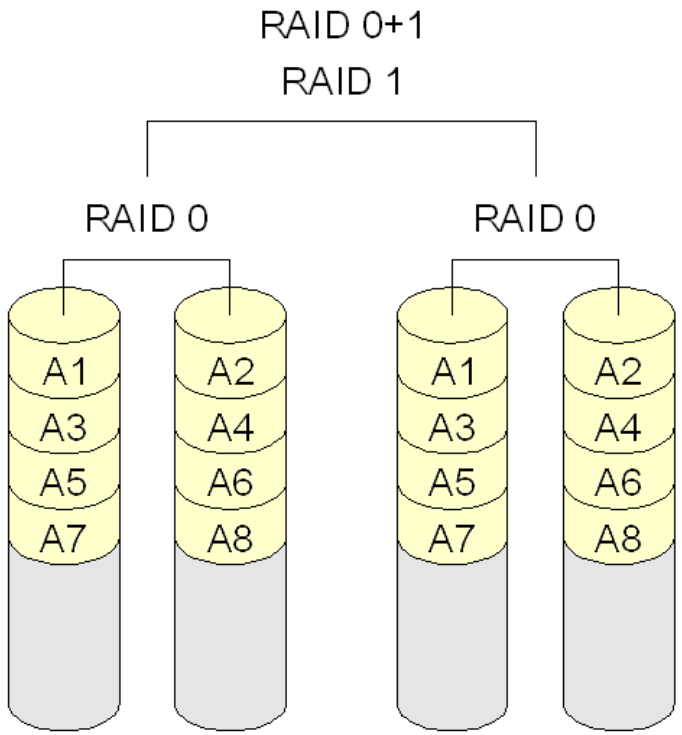
- RAID 1: adatok tükrözése több diszken
- Előnyök:
  - Olvasási műveletek: N-szeres átviteli sebesség
  - Írási műveletek: mint 1 diszk (nem gyorsabb)
  - Egyszerű
- Hátrányok:
  - Nagy (N-szeres) overhead

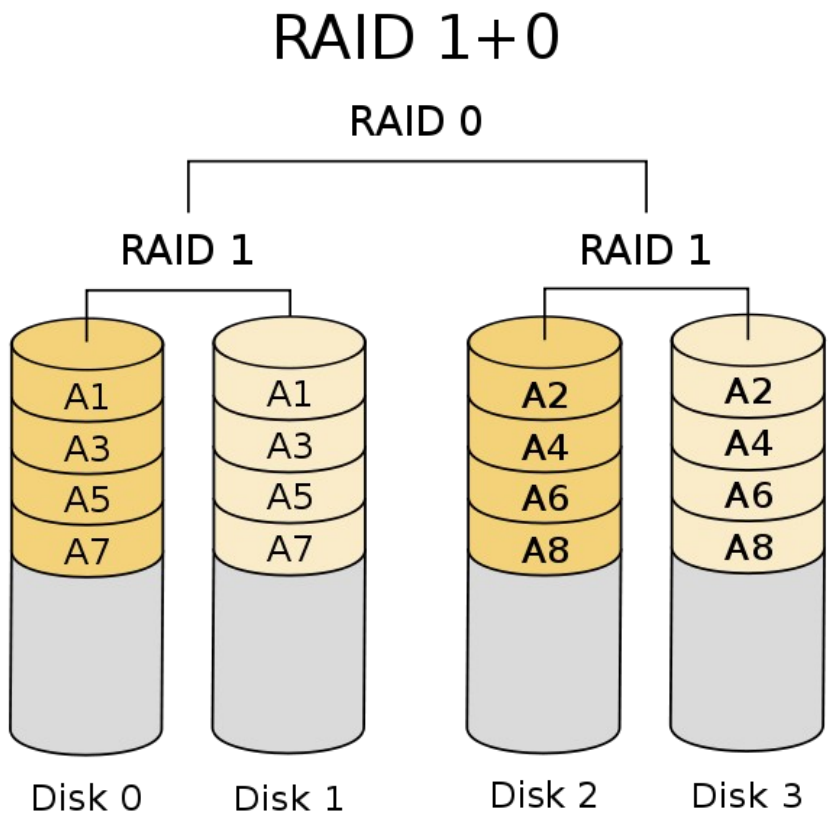
    → N diszkkapacitás = 1 felkínált kapacitás



RAID 0 ÉS RAID 1 KOMBINÁCIÓI

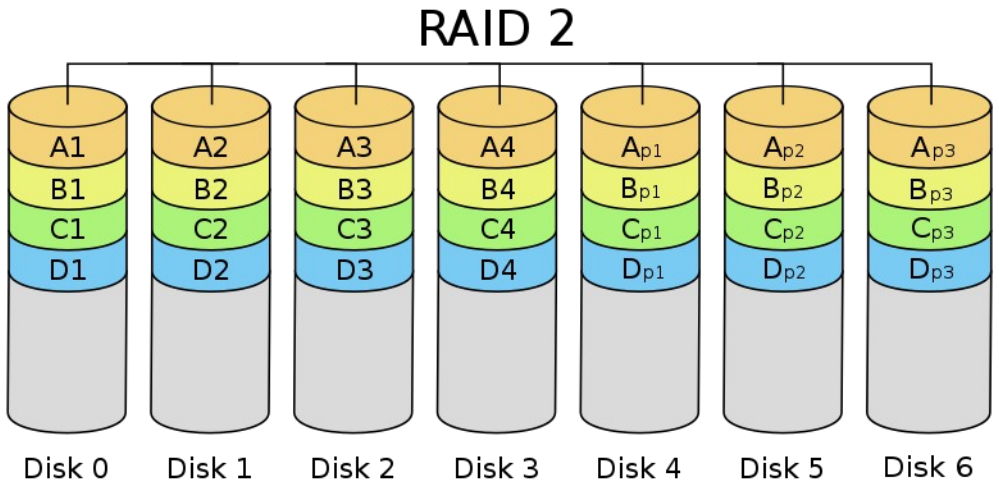
- RAID 0+1 és RAID 1+0 (=10)
- RAID 1+0 jobb, mert meghibásodáskor csak a RAID 1 rész érintett

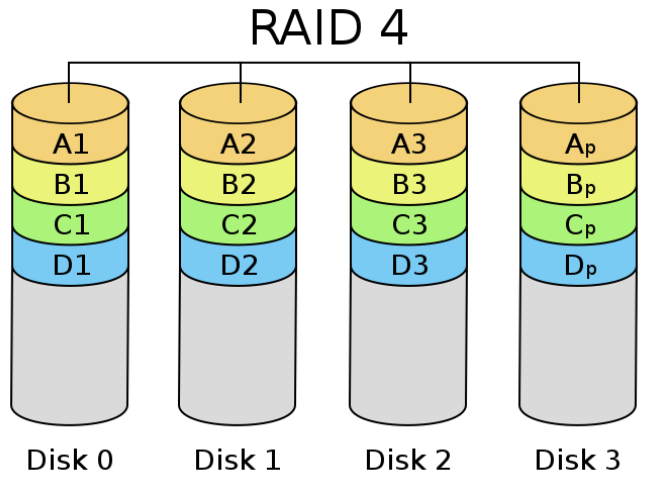




HIBAJAVÍTÁS KISEBB REDUNDANCIÁVAL

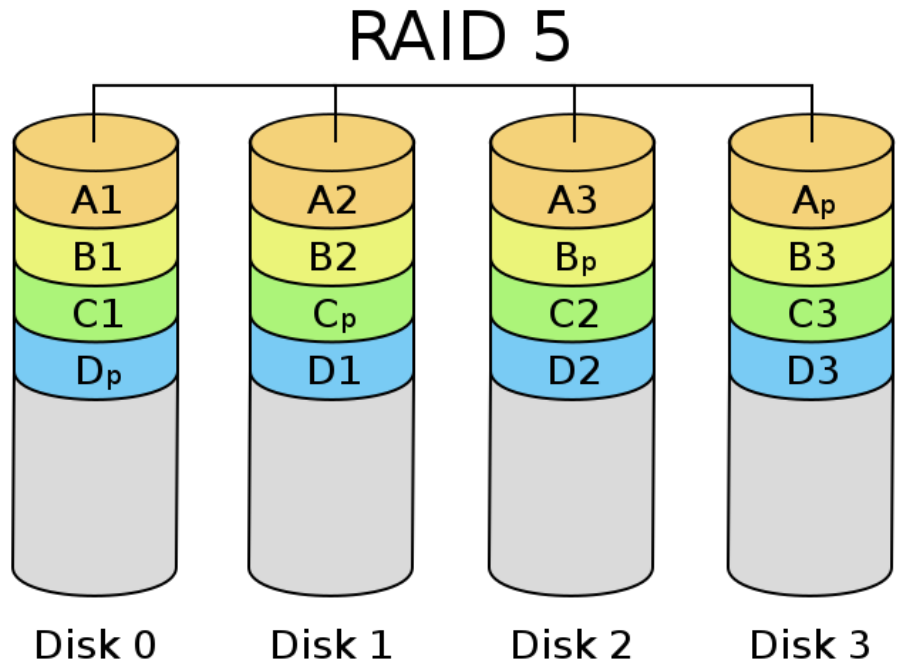
- Tükrözés túl pazarló
- Vannak jó hibajavító kódok (lásd: információelmélet)
- Pl. Hamming kód ilyen
  - Pl. (7, 4) kódolással 4 bites üzenetekből 7 bit lesz, bármelyik bit kijavítható
  - Ennél kisebb overhead is elérhető
  - RAID-2 alapja
    - de nincs kereskedelmi implementáció
- Paritásbit:
  - Nem alkalmas hibajavításra, csak 1 hiba jelzésére
  - Hacsak nem tudjuk a hiba helyét!
  - Mert akkor a paritásbit 1 hibát javítani is tud
  - Márpedig a diszkek szólnak, ha sikertelen egy művelet
- RAID-3 és 4
  - Dedikált paritásdiszk (szűk keresztmetszet)
  - Különböző adategységek





RAID 5

- RAID 5: hibajavítás paritásbittel
- N blokknyi adatot N+1 diszken tároljuk → a +1 a paritás
- Hibajavítás:
  - Tudható, ha egy diszk meghibásodik
  - Az összes blokk visszaállítható a többi diszkről
    - Akár a paritás sérült, akár bármely másik
- Előnyök:
  - Paritásinformáció eloszlik a diszkek között
    - Nem egy diszket terhel → szűk keresztmetszet lenne
  - Olvasási műveletek: nagy átviteli sebesség
    - N diszk dolgozik párhuzamosan
- Hátrányok:
  - Komplex hardveres implementáció
  - Adatok felülírása: lassabb
    - Felülírunk – minden diszket olvassuk – paritást számolunk – paritást írunk
  - Visszaállítás közbeni újabb hiba ellen nem véd
    - Márpedig nagy diszkek esetén valós veszély

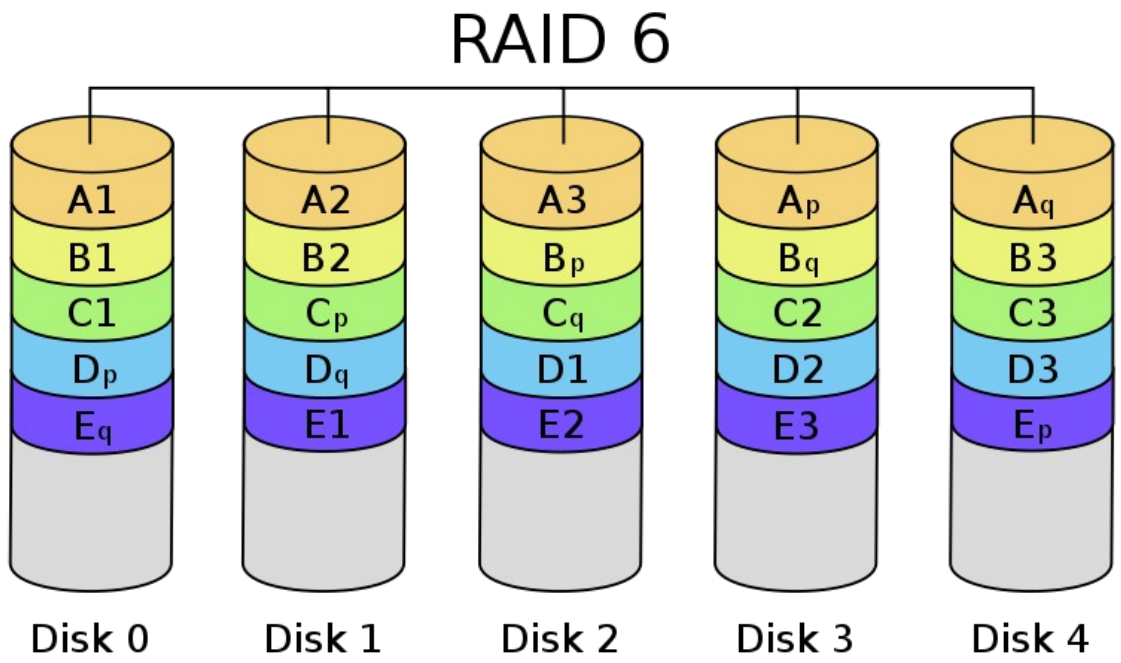


RAID 6

- RAID 6: mint RAID-5, de két paritásbittel
- N blokknyi adatot N+2 diszken tároljuk → a +2 a paritás

- Előnyök:
  - Olvasási műveletek: nagy átviteli sebesség (diszkek párhuzamos elérése)
  - Két diszk meghibásodása ellen is véd
    - Visszaállítás alatt is lesz védelem

- Hátrányok:
  - Nagyobb overhead
  - Felülírás: még lassabb (lásd: RAID-5)
  - Komplexitás



VEZÉRLŐK

- Áramkimaradás írás közben
  - RAID Write Hole Effect: Inkonzisztens állapot – nem helyreállítható a tömb
  - Hardveres RAID vezérlő megoldása:
    - Írási cache akkumulátorral védve
- Hot Swap
  - Diszk csere a rendszer leállítása nélkül
- Hot Spare
  - Bekötött tartalék diszk a RAID tömbben
  - Hiba esetén azonnal indul a helyreállítás
    - Nem kell megvárni az operátort

HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK