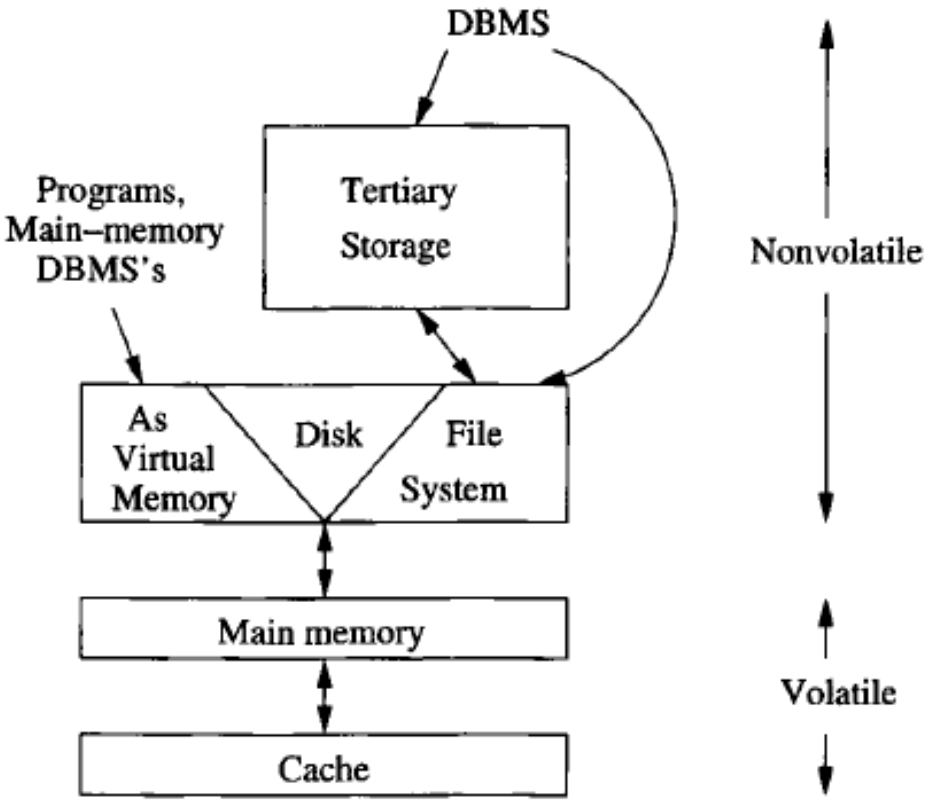
Adattárolás

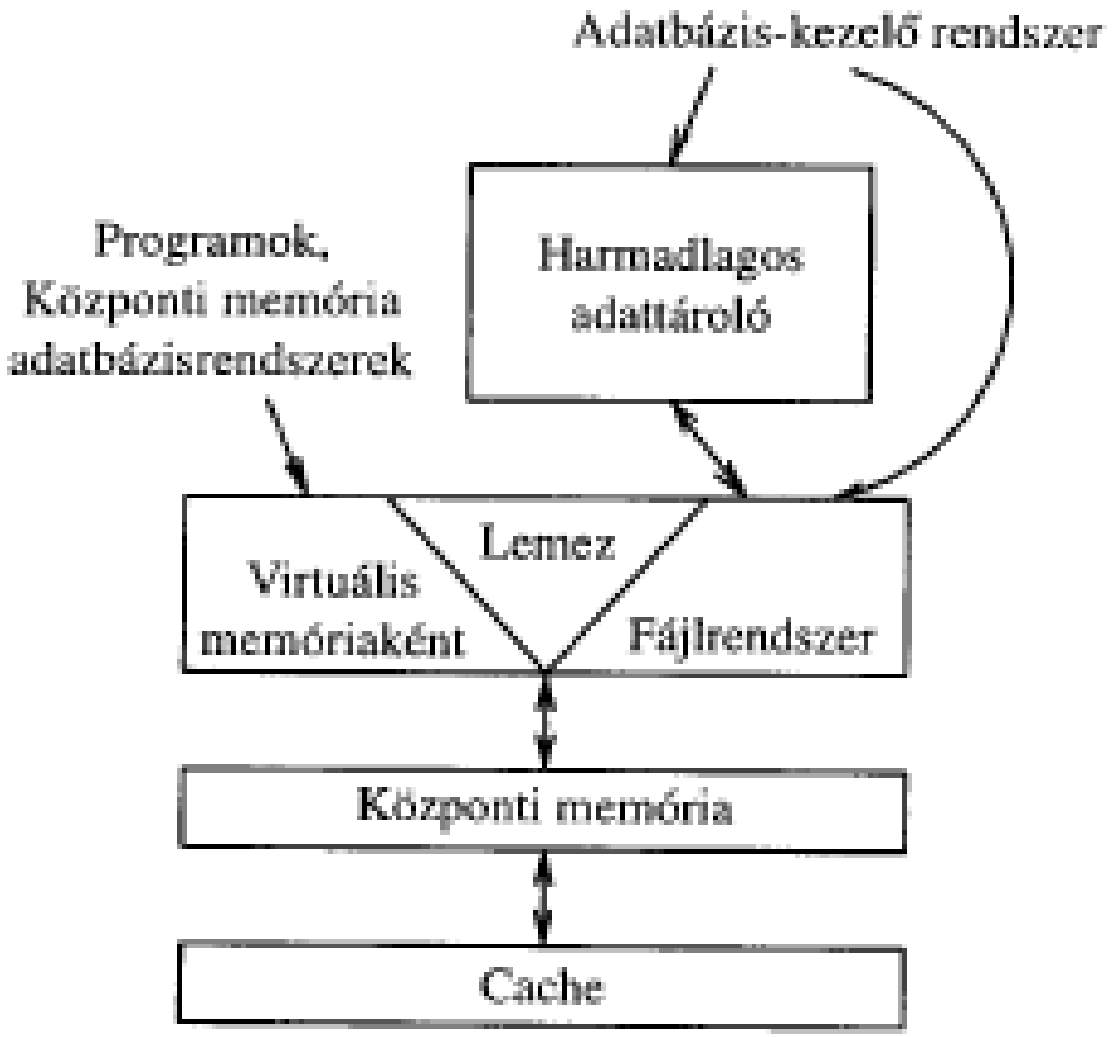
Adatbázisok 2.

(Méretek)

- $2^{10} \sim 10^3$: kilo
- $2^{20} \sim 10^6$: mega
- $2^{30} \sim 10^9$: giga
- $2^{40} \sim 10^{12}$: terra
- $2^{50} \sim 10^{15}$: peta.
- https://en.wikipedia.org/wiki/Petabyte

Memóriahierarchia





2.1. ábra. *A memóriahierarchia*

Cache

- Egy átlagos gép cache mérete 1Mb (2008).
- A cache gyakran két szintre osztható:
  - A beépített cache (on-board cache) ugyanazon a chipen található, mint a processzor.
  - A második szintű cache egy másik chipen helyezkedik el.
- A processzor néhány nanomásodperc alatt éri el a cache-ben tárolt adatot.

A maradék

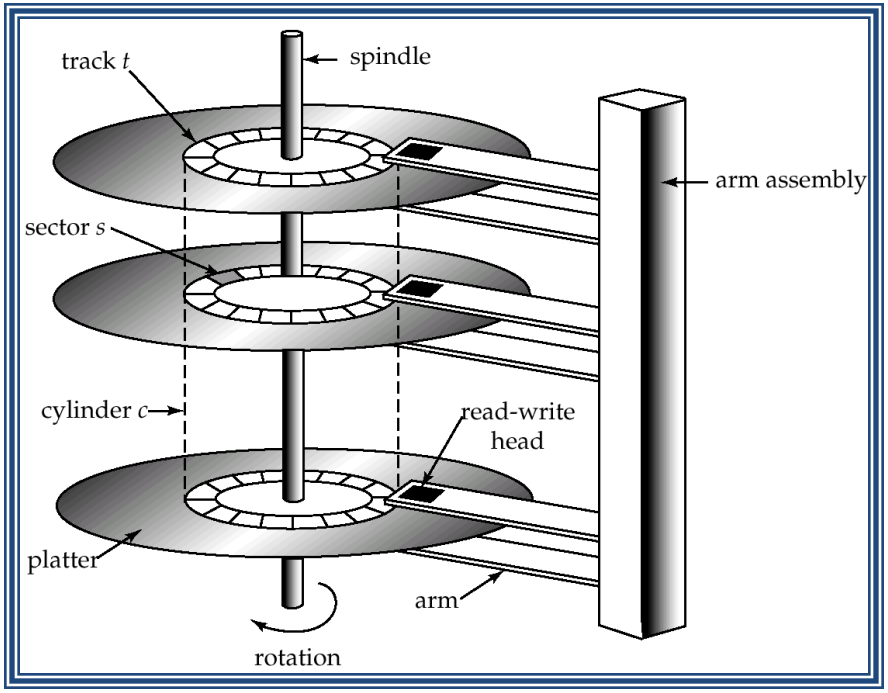
- Központi memória.
- Virtuális memória: megcímzett memóriaterület, 32 bites rendszerekben 2^{32} méretű (4 Gb)
  - 64 bites rendszerekben 2^{64} lehet a címterület.
- Másodlagos tároló.
- Harmadlagos tároló.
- Felejtő (volatile) és nem felejtő tárolás.

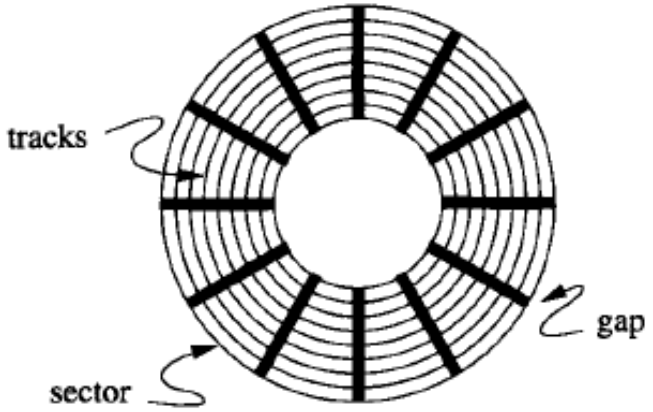
Adatok mozgatása

- Egymást követő szintek között mozgatjuk az adatokat.
- A központi memória és másodlagos tároló közötti adatmozgatás legkisebb egysége a blokk (4-64 kb).
- I/O művelet: lemezírás vagy olvasás.
- A központi memória folytonos területét, mely egy blokkot tárol, puffernek nevezzük.
- Egy I/O művelet körülbelül 10 miliszekundomot igényel.
- A cache-be egy cache sor (cache line) kerül, általában 32 egymást követő bájt.

Másodlagos tároló

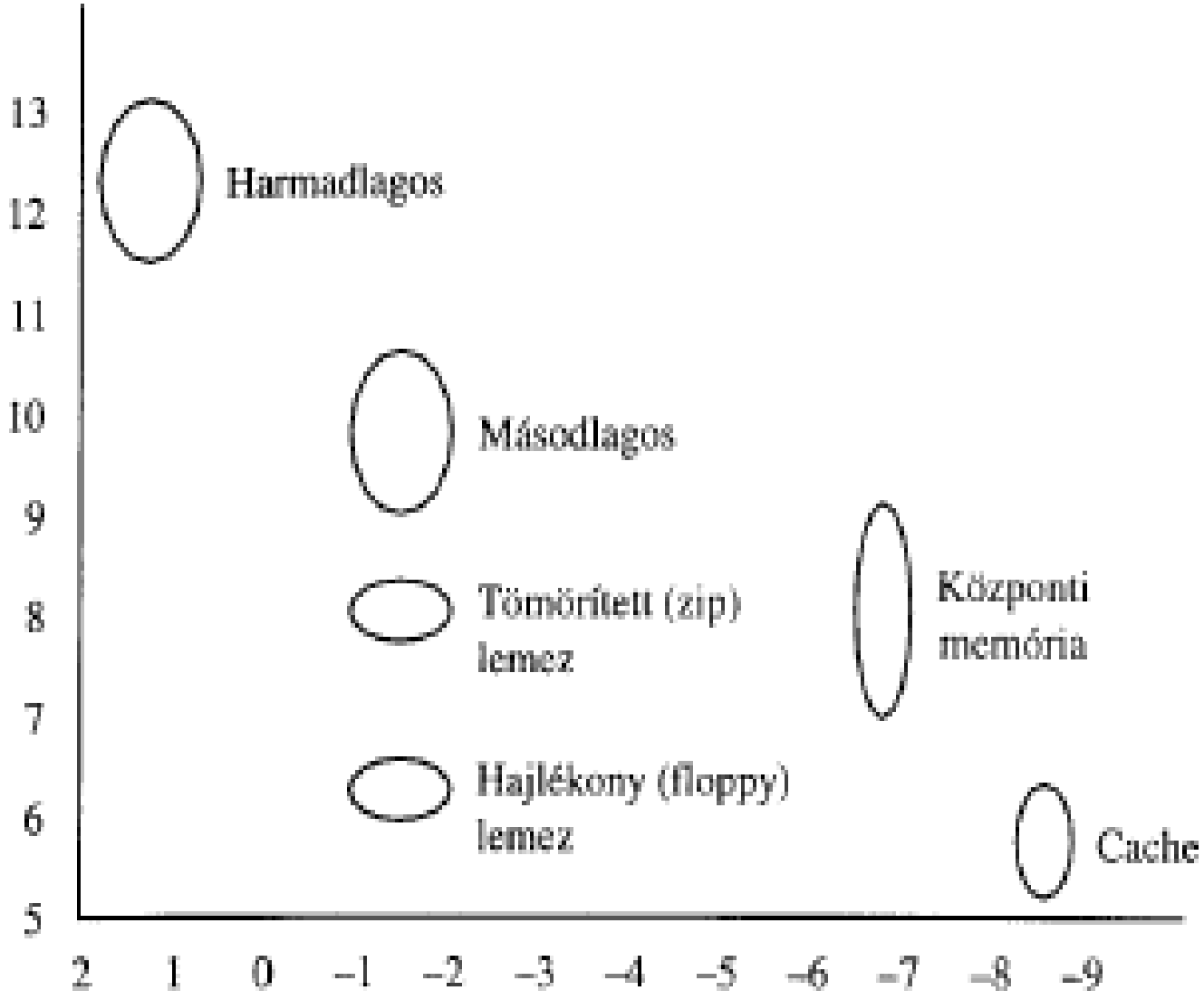
- Lemezgyűjtemény (disk assembly)
- Fejszerelvény (head / arm assembly)
- Tányér (platter)
- Sáv (track)
- Szektor (sector)
- Hézag (gap)
- Cilinder (cylinder)





Lemeztárolók

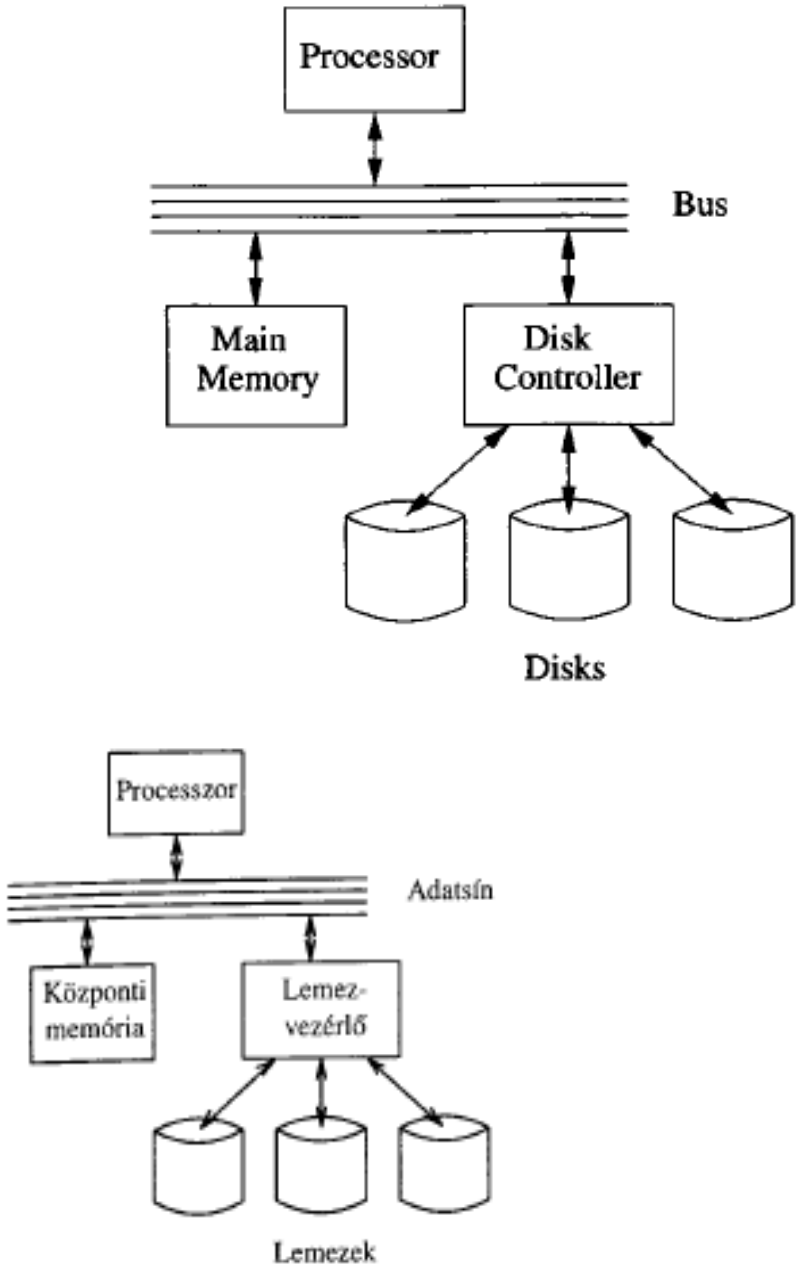
- Megatron 747 lemez jellemzői (2008):
  - 8 tányér (összesen 16 felület)
  - 2^{16}, azaz 65536 sáv egy-egy felületen
  - sávonként átlagosan 2^8 szektor
  - a szektorokban 2^{12}, azaz 4096 bájt (4 KB).
- Egy lemezre tehát 2^{40} bájt fér. =1 TB
- $= 16*2^{16}*2^{8}*2^{12} = 2^{40}$



2.3. ábra. *Az elérési idő és a kapacitás összevetése a memóriahierarchia különböző szintjein*

Lemezvezérlő

- A vezérli fejszerelvény mozgató mechanikus szerkezetet. A fejeket egy adott cilinder fölé pozícionálja.
- Egy-egy sávon belül a megfelelő szektor megtalálásáért is felelős.
- A kiválasztott szektor és a központi memória között mozgatja az adatokat.
- Saját memóriájában egy teljes sáv tartalmát is tárolhatja.



2.6. ábra. *Egy egyszerű számítógépes rendszer vázlatos felépítése*

Lemezhozzáférés

- A megfelelő cilinder megtalálásához szükséges idő a keresési idő (seek time).
- A megfelelő szektorhoz történő forgatás ideje a rotációs késés (rotational latency).
- A szükséges szektorok és hézagok fej alatti áthaladási ideje az átviteli idő (transfer time).

Feladat

- A lemez forgási sebessége 7200 fordulat/perc.
- A fejszerelvény mozgásánál az elindulás és leállás összesen 1 ms-t igényel. 4000 cilinderrel való elmozdulás további 1ms költséggel jár.
- A hézagok 10%-t foglalják el egy-egy sáv területének.
- Egy 16 kilobájtos blokk beolvasása minimum, maximum és átlagosan mennyi időt igényel?

Minimális idő

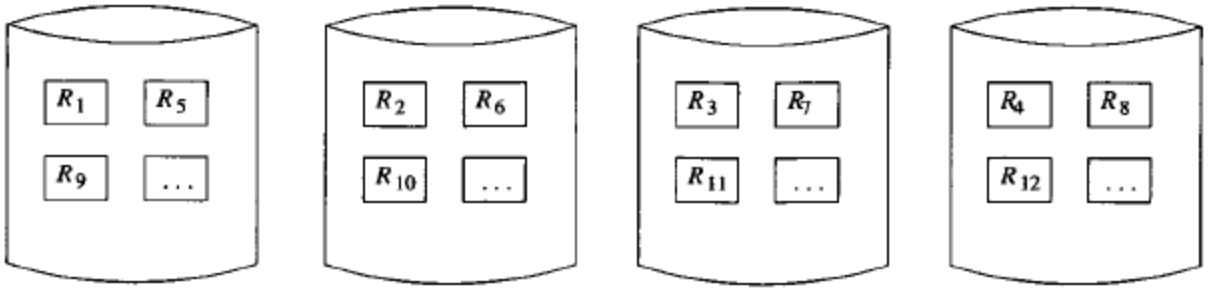
- Egy szektorban 4 kilobájt tárolható, tehát összesen négy szektort kell beolvasni, a fej alatt a köztük lévő három hézag is áthalad.
- 1 fordulat 60000/7200 = 8.33 ms.
- 1 sávon 256 szektor és hézag található.
- A 3 hézag összesen (36/256)*3 foknyit jelent egy 360 fokos fordulatnál. A szektorok (324/256)*4 fokot jelentenek: ez összesen 5,48 fok.
- Az eredmény: (5.48/360)*8.33 = 0,13 ms.

Maximális és átlagos idő

- Maximum. Az összes cilinderen történő áthaladáshoz szükséges idő: 1+(65536/4000) = 17,38 ms.
- Legrosszabb esetben épp áthaladtunk a legelső szükséges blokk fölött.
- Az eredmény: 17,38+8,33+0,13 = 25,84 ms.
- Átlag. Meg lehet mutatni, hogy egyenletes eloszlást feltételezve a fejek pozíciójára vonatkozóan átlagosan a cilinderek 1/3-n kell keresztülhaladni.
- Az átlagos keresési idő tehát: 1+(65536/3/4000) = 6,46 ms.
- Az eredmény: 6,46+8,33/2+0,13 = 10,76 ms.

A hozzáférés idejének javítása I.

- Cilinderes szervezés: azokat az adatokat, amelyeket várhatóan egyszerre kell majd elérni (pl. egy reláció) egy cilinderen tároljuk.
- Növelhetjük a lemezek számát is.
  - Nagy mennyiségű adatmozgatás esetén n lemez hozzáadása körülbelül olyan sebességnövekedést ér el, mintha n-szeresére gyorsulna egy lemez esetén a működés.



A hozzáférés idejének javítása II.

- Lemezek tükrözése.
  - Ez egyrészt véd az adatok meghibásodása ellen, másrészt n lemez esetén n blokkot tudunk beolvasni párhuzamosan.

Lift algoritmus

- Tegyük fel, a lemezvezérlő dönthet, az igényeket milyen sorrendben válaszolja meg.
- Az algoritmus során a fejek folyamatosan pásztázzák a lemezeket a legbelső cilindertől a legkülsőig és fordítva.
- Amint olyan cilinderhez érnek, ahol egy vagy több használandó blokk szerepel, a lemezvezérlő végrehajtja a műveleteket és ugyanabba az irányba halad tovább.

Feladat (2.4.4)

- Tegyük fel a 8000., 24000. és az 56000. cilinderekről egy-egy blokkot szeretnénk beolvasni. A fejek a 8000. cilinderen állnak épp.
- Emellett a 10., 20., 30. ms-ban további kérések érkeznek egy-egy blokk beolvasására a 16000., 64000. és 40000. cilinderről.
- Az egy blokkra jutó átviteli idő: 0,13 ms. Az átlagos rotációs késés 4,17 ms, átlagos keresés 6,46. keresési idő becslése: (a cilinderek/sávok sorszámának különbsége/4000)+1 ms.
- Mennyi ideig tart kiszolgálni az iménti kéréseket a lift algoritmussal?
- Mennyi ideig tart, ha mindig a legkorábban érkezett kérést szeretnénk kiszolgálni?

Megoldás

- Az átlagos rotációs késés 4,17 ms. Így ha egy keresett cilinderhez érkezünk 4,17 + 0,13 = 4,3 ms időt igényel a blokk beolvasása.

Cylinder of request	First time available
8000	0
24000	0
56000	0
16000	10
64000	20
40000	30

Lift algoritmus

Cylinder of request	Time completed
8000	4.3
24000	13.6
56000	26.9
64000	34.2
40000	45.5
16000	56.8

Legkorábbi kérés kiszolgálása

Cylinder of request	Time completed
8000	4.3
24000	13.6
56000	26.9
16000	42.2
64000	59.5
40000	70.8

Korai beolvasás, nagyléptékű pufferezés

- Bizonyos esetekben előre tudhatjuk, hogy mely blokkokra lesz szükség a közeljövőben, ilyenkor előre elkérhetjük ezeket, így javulhat a cilinderek elérésének az ütemezése.
- Analóg módon a kiírást is késleltethetjük addig, amíg már várhatóan nem lesz szükség a kiírandó blokkra, blokkokra.
- Ha sáv vagy cilinder méretű output puffereket használatával a rotációs késést (esetenként a keresési időt is) megspórolhatjuk.

Lemezhibák

- Ideiglenes meghibásodás
  - ellenőrző összegeket szoktak használni annak ellenőrzésére, hogy helyesen olvastuk, írtuk-e egy szektor tartalmát.
- Eszközhiba: egy vagy több bit végérvényesen elromlik.
- Íráshiba: az írás nem sikerül, az eredeti tartalmat nem lehet visszanyerni.
- Lemezhiba: a teljes lemez tönkremegy.

Ellenőrző összegek

- Legegyszerűbb formájában, ha a szektor bitjeinek páratlan sok 1-es található az ellenőrző bit 1, különben 0.
- Ha a bitekből álló halmazhoz hozzávesszük az iménti paritásbitet, a kapott halmazban mindig páros sok 1-es szerepel.
- Ha két bit is elromlik, már nem feltétlen nyújt védelmet.
- Ha n független paritásbitet használunk, akkor annak valószínűsége, hogy nem vesszük észre a hibát, $1/2^n$ körüli értékre csökkenthető.

Stabil tárolás

- A szektorokból párokat képzünk. Minden pár (X_L, X_R) egy X szektortartalmat reprezentál.
- Írás:
  - először X_L értékét írjuk. Ha hibás az írás, X_R alapján helyreállítható az eredeti tartalom. Addig ismételjük X_L írását, amíg helyes nem lesz az eredmény.
  - Az előbbi műveletet megismételjük X_R-re.
- Olvasás:
  - olvassuk be X_L értékeit. Ha túl sokszor kapunk hibaüzenetet, olvassuk be X_R értékeit helyette.

1. szintű RAID

- RAID: független lemezek redundáns tömbje (Redundant Arrays of Indenpendent Disks).
- A RAID alkalmazása során adatlemezekkel és redundáns lemezekkel dolgozunk.
- Az 1. szintű RAID szimplán a lemezek tükrözését jelenti.
- Ebben az esetben az adat- és redundáns lemezek száma azonos.

4. szintű RAID

- Egyetlen redundáns lemezt használunk.
- Az adatlemezek és a redundáns lemez szerkezete ugyanolyan.
- A redundáns lemez i. blokkjának j. bitje az adatlemezekhez tartozó i. blokkok j. bitjeinek paritásbitje.
- Példa: 1. lemez: 11110000
  2. lemez: 10101010
  3. lemez: 00111000
  4. lemez: 01100010 (ő redundás)

Olvasás

- Mindenféle változtatás nélkül működik.
- Egyes esetekben időt is spórolhatunk.
  - Tegyük fel az 1. lemeznek épp egy blokkját olvassuk, de egy másik blokk tartalmára is szükség lenne.
  - A maradék három lemez segítségével ezt könnyen megkaphatjuk.
- Példa: 2. lemez: 10101010
  3. lemez: 00111000
  4. lemez: 01100010
  1. lemez: 11110000

Írás

- Ha valamelyik adatlemezen módosítjuk egy blokk tartalmát, a redundáns lemez megfelelő blokkját is módosítani kell.
- Példa: 1. lemez: 11110000
  2. lemez: 10101010
  3. lemez: 00111000.
  - 2. második lemez blokkját 11001100-ra változtatjuk.
  - Az eredeti és új blokk tartalmának modulo-2 összege: 01100110, azaz a 2., 3., 6., 7. helyeken kell változtatni a paritásblokkot.
  - Ez egyszerűen megtehető. Az iménti vektort modulo-2 hozzáadjuk a paritásblokk tartalmához.

Hibajavítás

- Bármelyik lemez tönkremegy, a többi lemez alapján helyreállítható.
- Példa: 1. lemez: 11110000

  2. lemez: ????????

  3. lemez: 00111000

  4. lemez: 01100010

  A hiányzó blokk: 10101010.

Vektorok modulo-2 összegzése

- Kommutatív: $x \oplus y = y \oplus x$.
- Asszociatív: $(x \oplus y) \oplus z = x \oplus (y \oplus z)$.
- A csupa nulla vektor egységelem:
  - $x \oplus \underline{0} = \underline{0} \oplus x = x$.
- $\oplus$ saját maga inverze: $x \oplus x = \underline{0}$.
  - Következmény: ha $x \oplus y = z$, akkor $y = x \oplus z$.

5. szintű RAID

- A 4. szintű RAID használatánál n adatlemez esetén, a redundáns lemez írásainak száma n-szerese az egy-egy adatlemezre jutó írások számának, ha azok egyenletesen oszlanak el.
- 5. szintű RAID: minden lemez esetén bizonyos sávok lesznek redundánsak.
  - n + 1. lemezünk van, melyeket 0-tól n-ig számozunk.
  - A j. lemez i. sávja redundáns, ha i / (n + 1) maradéka j.

6. szintű RAID

	Adat				Redundáns		
Lemezek száma	1	2	3	4	5	6	7
	1	1	1	0	1	0	0
	1	1	0	1	0	1	0
	1	0	1	1	0	0	1

- Minden lehetséges oszlop megjelenik, kivéve a csupa 0 oszlopot.
- Azok a lemezek, amelyekhez egy-egy sorban 1-es tartozik 4. szintű RAID-et alkotnak.
  - Például: az 5. lemez bitjeit úgy kapjuk, hogy az 1., 2., 3. lemez megfelelő bitjeit modulo-2 összeadjuk é. í. t.

Hibajavítás

- Olvasás és írás hasonlóan működik, mint a 4. szintű RAID esetében.
- Ebben az esetben viszont két tönkrement lemezt is ki tudunk javítani.
  - A két lemezhez tartozó oszlop legalább az egyik sornál különbözik.
  - Az a lemez, ahol az előbbi sorban 1-es szerepel, azon lemezek segítségével, melyeknél a sorban szintén 1-es szerepel, helyreállítható, hiszen ezek 4. szintű RAID-et alkotnak.
  - A másik lemeznél ezután egy olyan sort veszünk, ahol a neki megfelelő oszlopban 1-es szerepel, és a sorban szintén 1-essel szereplő lemezek segítségével helyreállítjuk ezt is.

Hibajavítás II.

- Példa:

  1. lemez: 11110000

  2. lemez: ????????

  3. lemez: 00111000

  4. lemez: 01000001

  5. lemez: ????????

  6. lemez: 10111110

  7. lemez: 10001001

Adat				Redundáns		
1	2	3	4	5	6	7
1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

- Itt a 2. lemez az 1., 4., 6. lemez alapján helyreállítható.
- Az 5. lemez ez után az 1., 2., 3. lemezek alapján szintén helyreállítható.

Észrevétel

- Az előbbi példa általánosítható.
- $2^r - 1$ lemez esetén r lemez használandó redundáns lemezként, a maradék $2^r - 1 - r$ adatlemezként.