ELTE | IK
INFORMATIKAI KAR

Adatbázisok 2

Fizikai tárolás

Az előadás célja

- Fizikai adattárolás (HDD, SSD)
- Pufferkezelés
- Fájlok és blokkok felépítése
- Rekordok felépítése

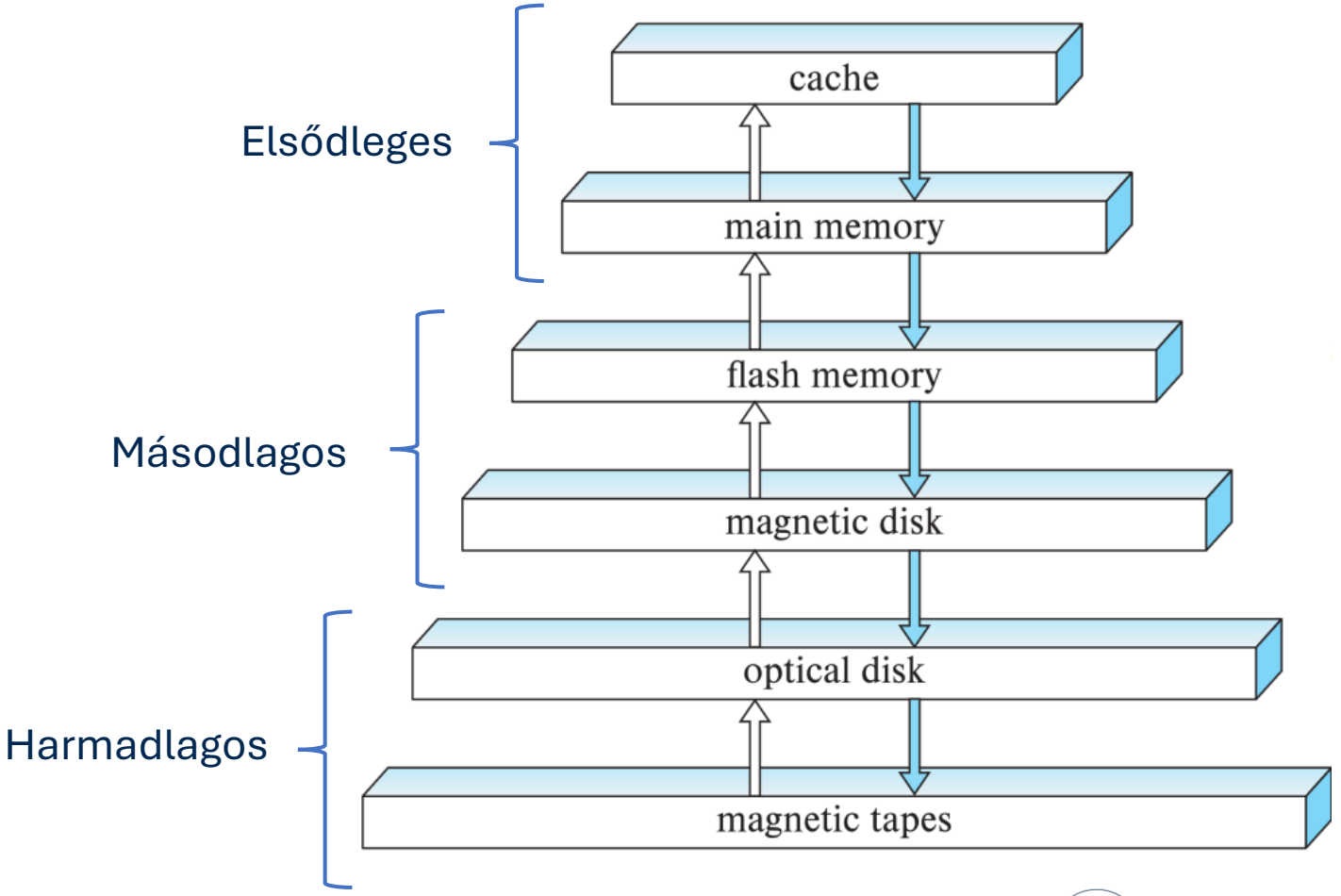
Memóriahierarchia

Gyorsabb,

drágább

Lassabb,

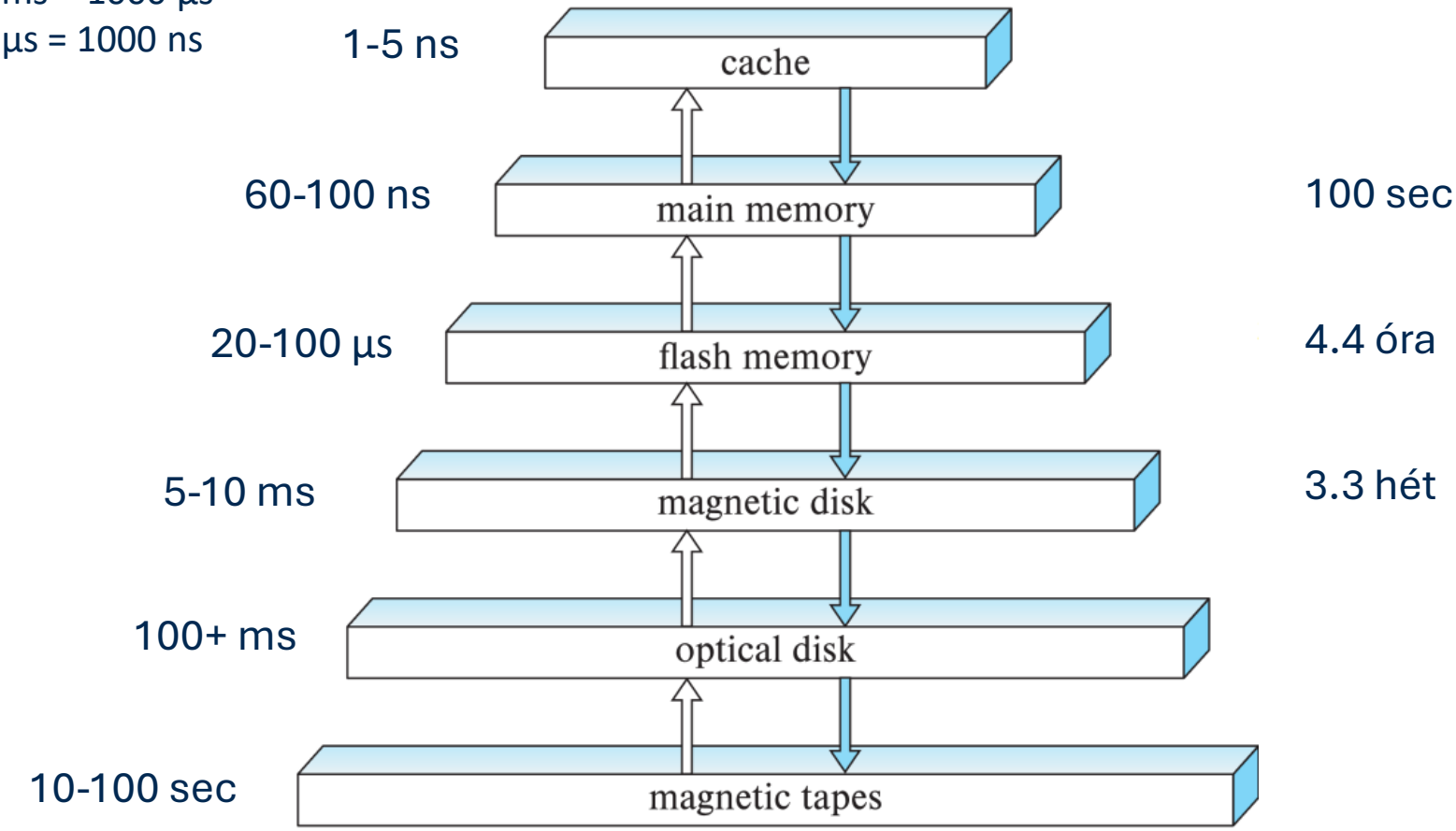
olcsóbb



Memóriahierarchia – elérési idők

1 sec = 1000 ms
1 ms = 1000 µs
1 µs = 1000 ns

Tfh. 1ns = 1sec



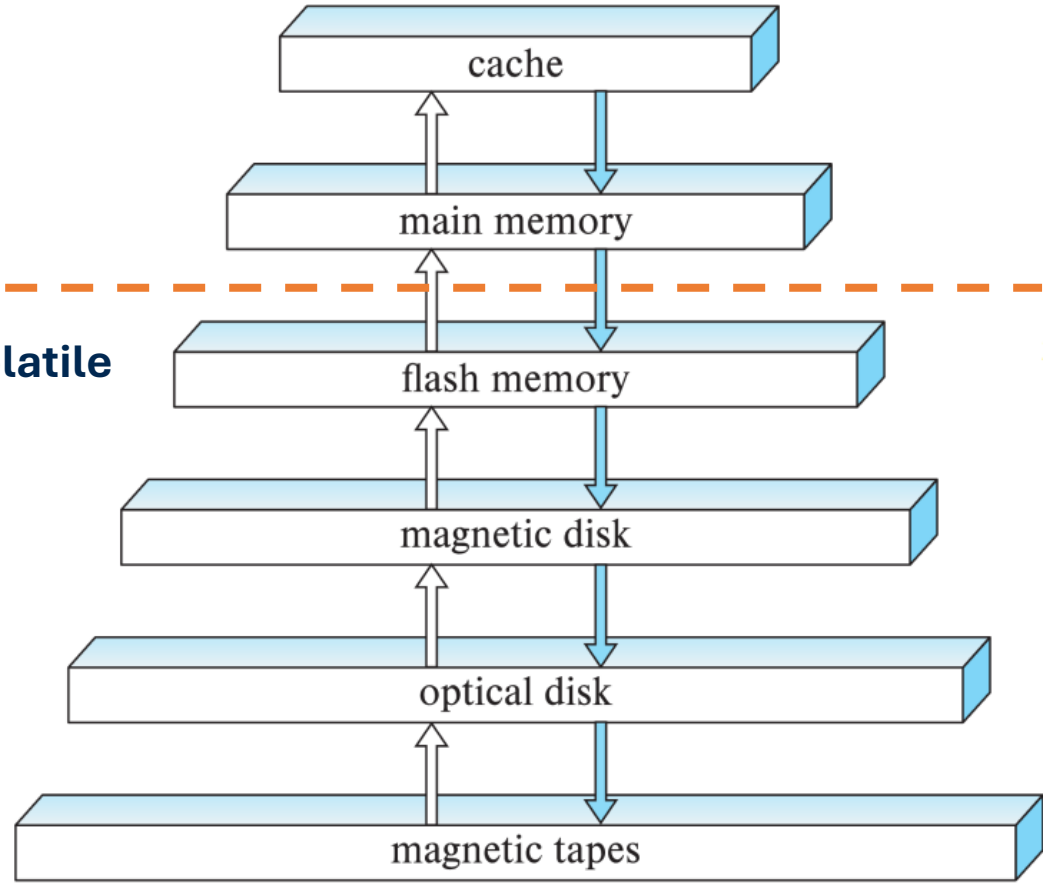
Memóriahierarchia - felejtés

Felejtő - volatile

(random elérés)

Nem felejtő – non-volatile

(szekvenciális elérés)

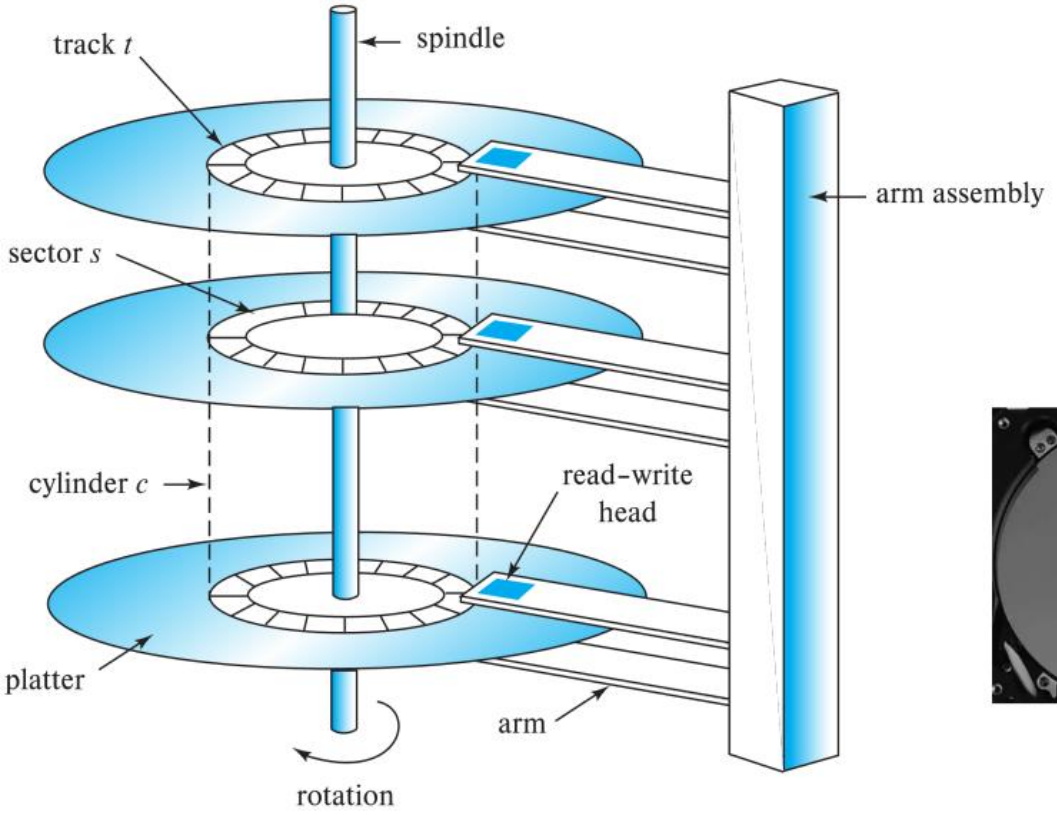


Megjegyzések

- Feltételezzük, hogy az adatbázis rendelkezik valamilyen nem felejtő tárolóval, így a rendszernek adatokat kell mozgatnia a memória és a tároló között.
- Egy lemezblokk olvasása ~10 ms – ennyi idő alatt egy modern processzor 100 millió műveletet is végrehajthat. Ezért adatbázis szempontból a lemez I/O költségek minimalizálása a legfontosabb.
- A nagy adatközpontok másodlagos tárolóként HDD-t és SSD-t is használnak. [1,2]

[1] https://blog.westerndigital.com/a-balancing-act-hdds-and-ssds-in-modern-data-centers/
[2] https://blocksandfiles.com/2024/12/20/the-future-of-the-hdd-is-the-ssd/

HDD felépítése



Sematikus ábra

Valós lemez

HDD teljesítmény

- Az elérés három komponensből tevődik össze:
    1. Keresési idő (seek time) – amíg a fej a megfelelő sávhoz ér.
        - Az átlagos keresési idő fele akkora, mint a legrosszabb esetben.
        - 4-10 ms egy átlagos lemezen.
    2. Forgási késés (rotational latency) – amíg a fej a megfelelő blokkhoz ér.
        - Az átlagos forgási késés ideje fele akkora, mint a legrosszabb eseteben.
        - 4-11 ms egy átlagos lemezen (5400-15000 rpm).
    3. Átviteli idő (data-transfer rate) – az adatok olvasásának az ideje.
        - 80-200 MB/s

HDD vs SSD teljesítmény

Jellemző	HDD	SSD
Szekvenciális olvasás	80-200 MB/s	3000+ MB/s
Késleltetés	ms	µs
Véletlenszerű IOPS*	50-200	40 000 (350 000 párhuzamosan)

- IOPS – Input/Output Operations Per Second

*4 KB blokkok esetén

HDD vs SSD

- HDD előnyök:
  - A tárhely/érték arány az SSD-hez viszonyítva ~1:6.
  - Nagyobb gyártáskapacitás (jelenleg).
- SSD előnyök:
  - Sokkal nagyobb szekvenciális írási és olvasási sebesség és random IOPS.
  - Nem olyan nagy a különbség a szekvenciális és véletlenszerű elérés között (4 KB blokkok esetén ~1,5x).
  - Energia felhasználás szempontjából hatékonyabb.
  - Nincsenek mozgó alkatrészek.

Következtetések

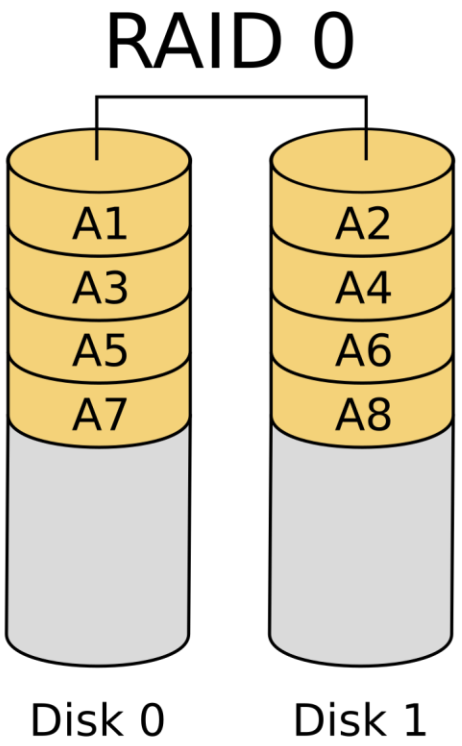
- Mivel a szekvenciális olvasás gyorsabb, mint a random, ezért arra fogunk törekedni, hogy egymáshoz közel tároljuk az adatokat fizikailag.
- A lemez IO költséges (HDD és SSD esetén is), ezért minimalizálni szeretnénk az ilyet műveleteket.
- A következőkben a másodlagos tárolóra lemezként hivatkozunk, legyen az SSD vagy HDD.

Lemezhibák és teljesítmény

- **Mi történik ha megsérül a lemez?**
- Egyetlen lemez helyett használjunk többet.
- Javíthatjuk az IO időt és a megbízhatóságot.
- RAID – Redundant Arrays of Indepentent Disks (független lemezek redundáns tömbje)
- 7 szint + 2 kombinált (több szintű)

RAID 0. szint

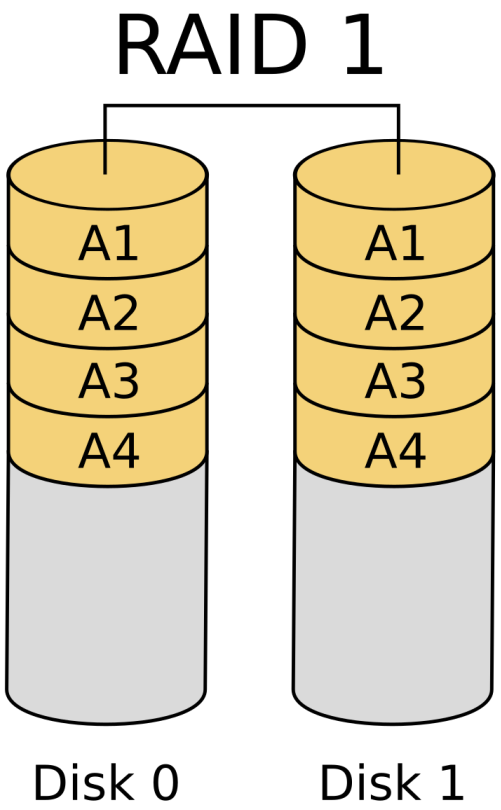
- Blokkszintű csíkozás (striping).
- Nincs redundancia.
- A lemez IO teljesítményt javítja.



Kép: By en:User:Cburnett - This W3C-unspecified vector image was created with Inkscape ., CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1509075

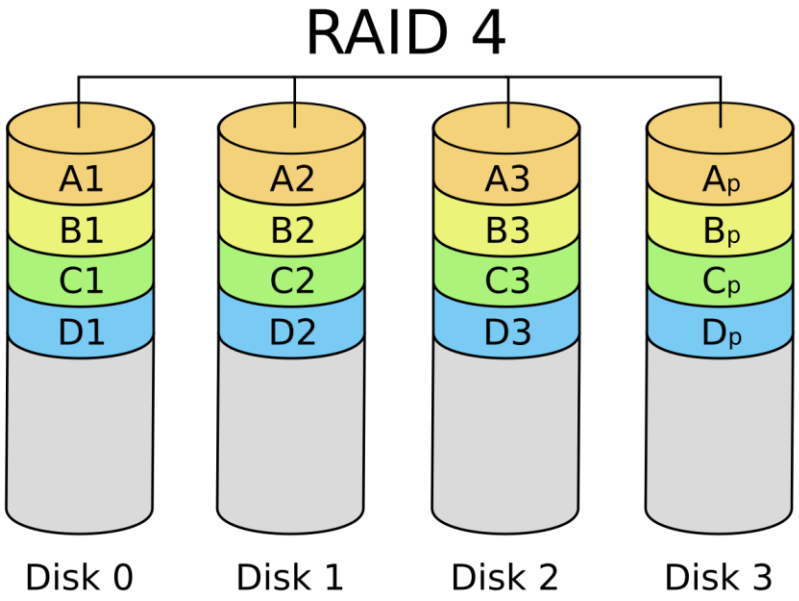
RAID 1. szint

- Tükrözés – minden lemezünkről legyen egy másolat.
- Párhuzamos olvasás.
- Redundáns, hibatűrő.
- Drága (a megvásárolt táhelynek csak a felét használhatjuk).



RAID 2-4. szintek

- Ezeket a szinteket a gyakorlatban már nem használják.
- RAID 2. szint – bit szintű csíkozás (Hamming-kód a helyreállításhoz)
- RAID 3. szint – bájt szintű csíkozás (külön paritáslemez)
- RAID 4. szint – blokk szintű csíkozás (külön paritáslemez)



Paritásblokkok (RAID 4. esetén)

- Egyszerű eset: 4 lemez, lemezenként 1 blokk, amelyek 8 bitből állnak.
- Ha írjuk valamelyik lemezt, akkor a paritáslemezt is írni kell.

1. lemez (A)	2. lemez (B)	3. lemez (C)	A XOR B XOR C
1	1	0	0
1	0	0	1
1	1	1	1
1	0	1	0
0	1	1	0
0	0	0	0
0	1	0	1
0	0	0	0

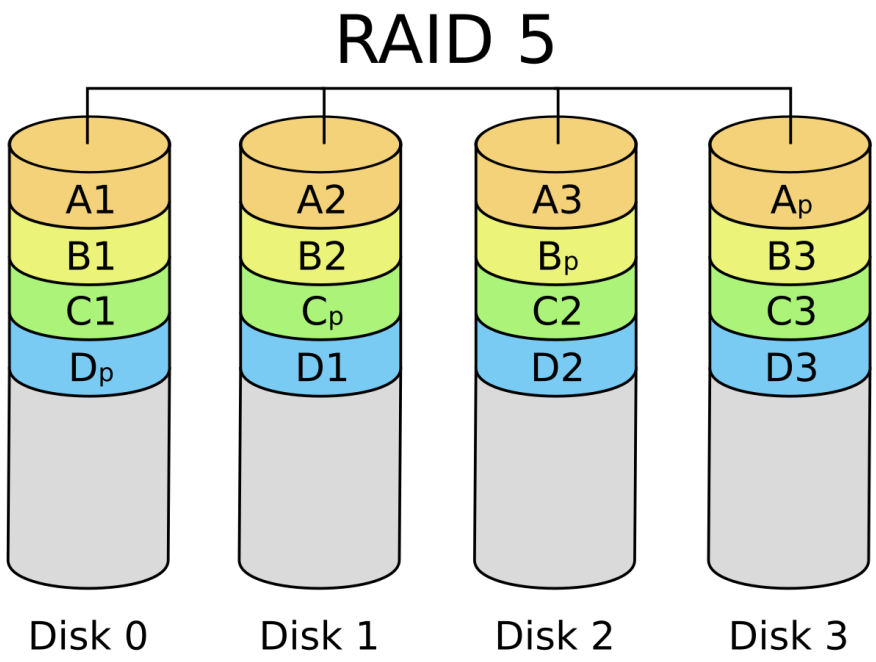
Helyreállítás

- Bármelyik bit az összes többi lemez megfelelő helyén álló bitekre alkalmazott XOR (kizáró vagy) műveletek eredménye (vagy a bitek mod 2 szerinti összege).
- Mi a baj ezzel a megközelítéssel? A paritás lemezt gyakrabban kell írni.

1. lemez (A)	2. lemez (B)	3. lemez (C)	A XOR B XOR C
1	0	?	0
0	?	1	1
0	?	1	0
?	0	0	0
?	1	0	1
1	1	1	?
1	1	0	?
0	0	?	0

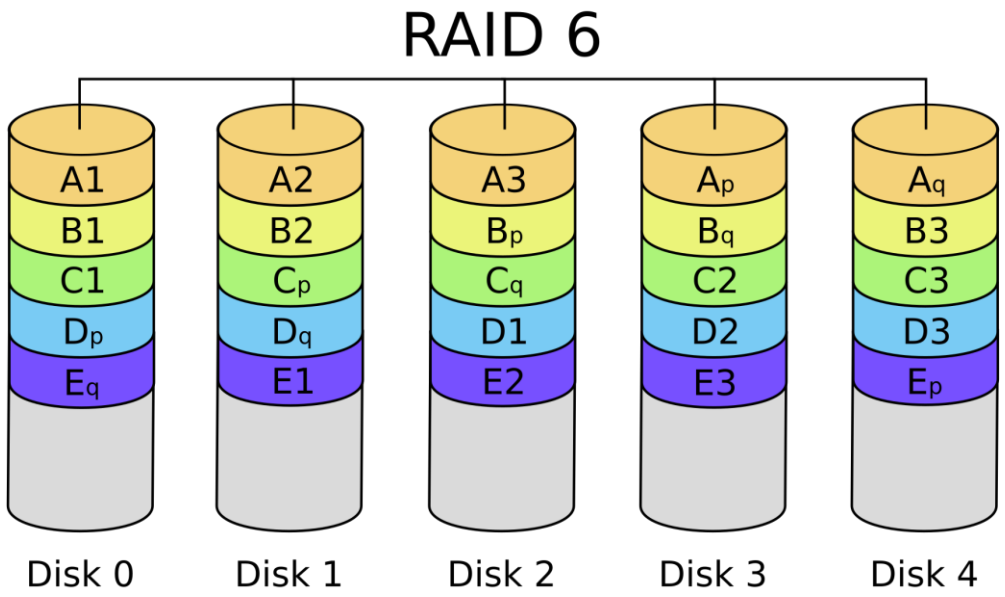
RAID 5. szint

- A paritásblokkokat osszuk szét a lemezek között.
- Így hasonló lesz a lemezek IO terhelése.
- Legalább 3 lemezre van szükség.



RAID 6. szint

- Két paritást használ (P és Q általában), ezért két lemez sérülése esetén is képes a helyreállításra.
- Az egyik (P) általában egyszerű paritást használ (mint a korábbiak).
- A másik (Q) valamilyen más megközelítést, pl. Reed-Solomon kód.



Több szintű RAID

- RAID 01 (0+1)
  - Először csíkozás, aztán tükrözés.
- RAID 10 (1+0)
  - Először tükrözés, aztán csíkozás.

RAID szempontok

- Mit kell figyelembe vennünk választáskor?
  - Költségkeret
  - Teljesítmény
  - Hibatűrés
- RAID 0 csak akkor van használatban ha a hibatűrés nem fontos (pl. más forrásból gyorsan helyre lehet állítani)

RAID választás

- Napjainkban a **RAID 6** már népszerűbb megoldás, mint a RAID 5.
- A teljesítmény kritikus rendszereknél a **RAID 1+0** az egyik legjobb választás.

Szint	Használat	Előny	Hátrány
RAID 5	Kisebb rendszerek	Jó tárhelykihasználás	Csak egy lemez kiesés tolerálható
RAID 6	Nagyobb adattárházak	Jó hibatűrés (két lemez esetén is)	Lassabb írás
RAID 1+0	Gyors, kritikus rendszerek	Gyors írás/olvasás	A tárhely felét elveszítjük

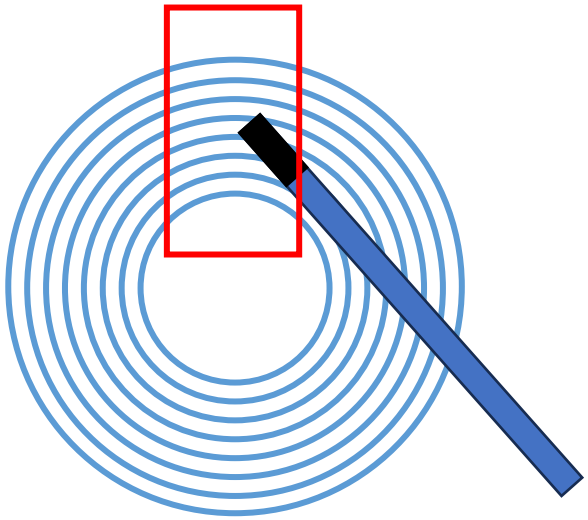
Blokkolvasások optimalizálása

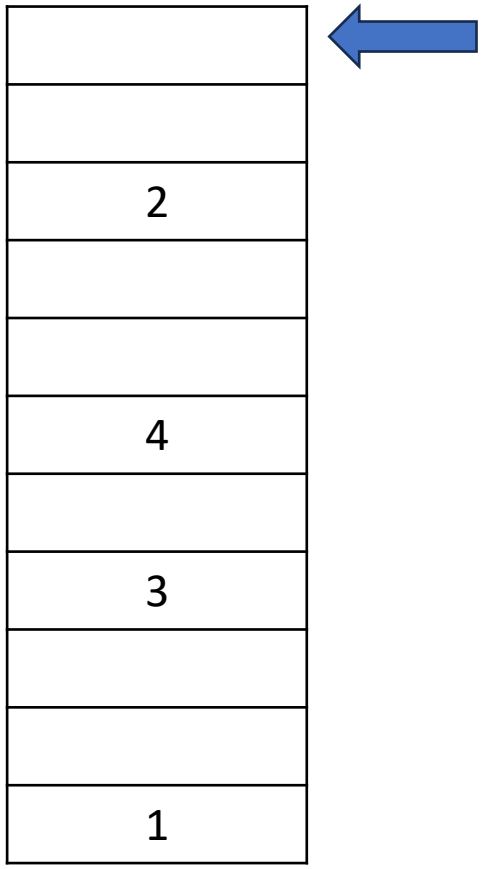
- Pufferelés (buffering)
  - Megpróbáljuk kitalálni milyen adatokra lehet szükség a későbbiekben és előre beolvassuk azokat.
- Előre olvasás (read-ahead)
  - A lemezen a szükséges blokkokon kívül a soron következőket is beolvassuk (szekvenciálisan).
- Lemez ütemezése (disk-arm-scheduling)
  - A kar mozgását optimalizáljuk (pl. lift algoritmus – elevator vagy scan algorithm)

Lift algoritmus: Peter J. Denning. 1967. Effects of scheduling on file memory operations. In Proceedings of the April 18-20, 1967, spring joint computer conference (AFIPS '67 (Spring)). Association for Computing Machinery, New York, NY, USA, 9–21. https://doi.org/10.1145/1465482.1465485

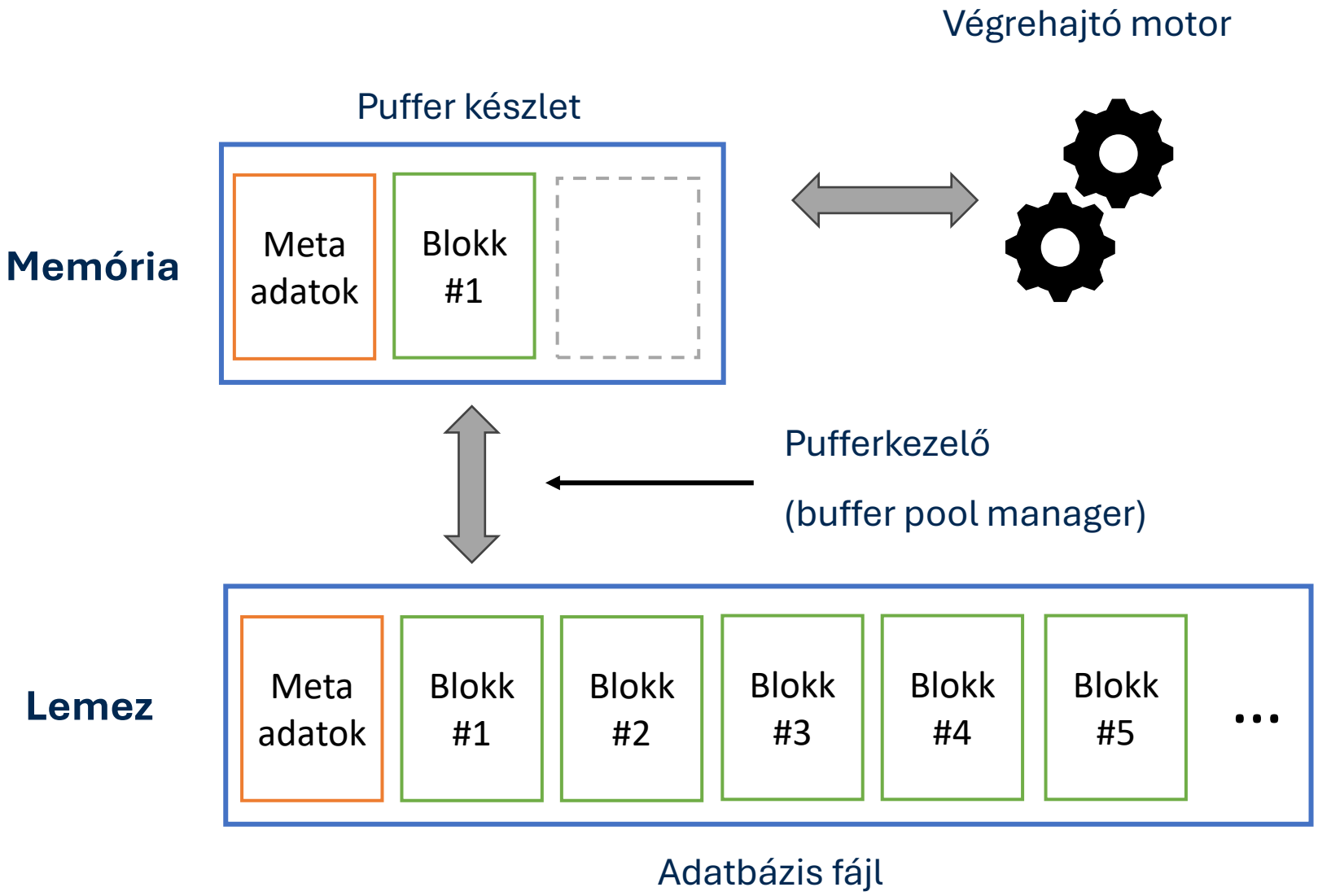
Lift algoritmus

- Gondoljunk úgy a fejre, mint egy liftre, amely a külső és belső sávok (cilinderek) között mozog.
- Mindig elindul egy irányba és megáll ha valahol olvasnia kell. A végén visszafordul.
- Egyszerű megközelítés: 10+8+5+2=25 lépés
- Lift algoritmussal: 10 lépés (fentről le)



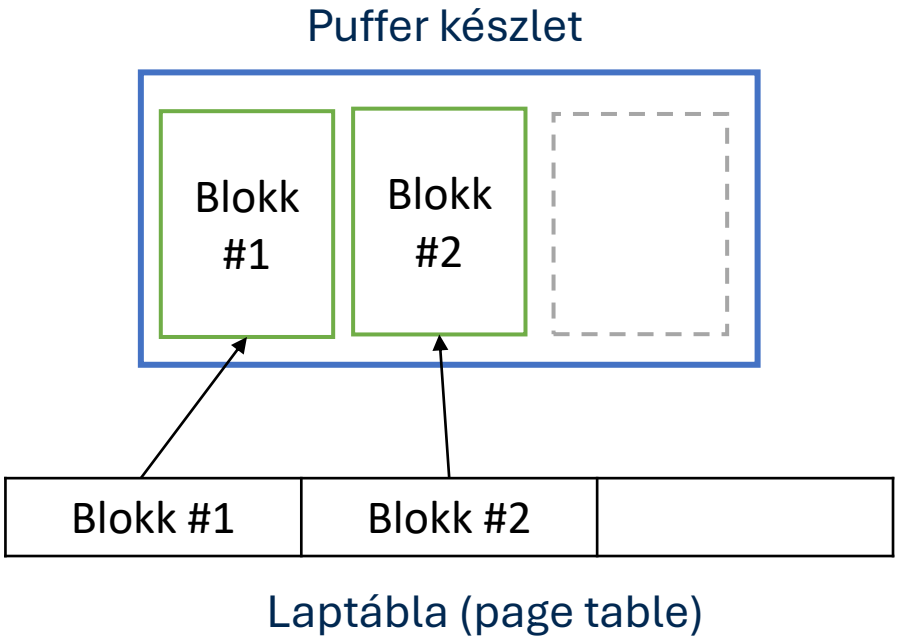


Pufferkészlet (buffer pool)



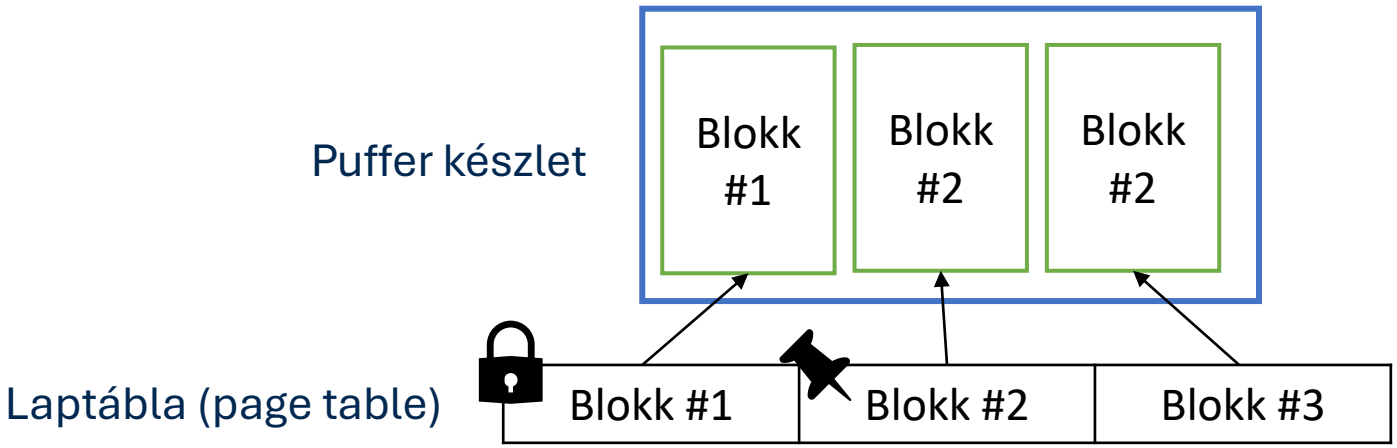
Pufferkészlet

- A puffer egy memória-terület, amelyet az ABKR használ az adatok ideiglenes tárolására. Célja, hogy gyorsítsa az adatokhoz való hozzáférést.
- A puffert a pufferkezelő rendszer kezeli.
- A laptábla a pufferkészletben lévő lapok állapotáról tartalmaz metaadatokat.



Laptábla

- A laptábla általában egy fix méretű hash-tábla.
- Egyéb adatok mellett tartalmazza a következőket:
  - Feltűzés (pin) számláló – ha valaki használja, akkor rögzítjük a blokkot, hogy ne lehessen kidobni a pufferből.
  - Dirty flag – piszkos puffer vagy sem (kiíratlan módosítások).
  - Hozzáférési információk – milyen zárak vannak érvényben (erről később)



Pufferkezelő működése

- Tegyük fel, hogy a végrehajtó motornak szüksége van egy blokkra.
- Ha a memóriában van, akkor használhatja (buffer hit)
- Ha nincs a memóriában, akkor a pufferkezelő beolvassa a lemezről a pufferkészlet egy üres helyére (buffer miss).
- **Kérdés**: mi történik akkor, ha nincs üres hely a pufferkészletben?

Lapozási (lapcsere) algoritmusok

- Ha egy blokk piszkos, akkor a lapcsere előtt ki kell írnunk a lemezre.
- **Least Recently Used (LRU)**
  - Minden blokkhoz tartozik egy időbélyeg (timestamp), amely azt mondja meg, hogy mikor volt utoljára használva.
  - Lapcsere esetén a legrégebben használt lap helyére tesszük az újat.
- **Azonnali eldobás (toss-immediate)** – ha feldolgoztunk egy lapot, azonnal dobjuk el. Mikor lehet ez hasznos?
- **Most Recently Used (MRE)**
  - Itt is szükségünk van egy időbélyegre. Az LRU-val ellentétben lapcsere esetén a legutóbb használt blokk helyére tesszük az újat.

Lapozási (lapcsere) algoritmusok

- **LRU-K**
  - Tároljuk a legutolsó K hozzáférés időbélyegét.
  - Lapcsere esetén a K-adik időbélyegek közül válasszuk a legrégebbit.
  - Figyelembe veszi nem csak a frissességet, de a gyakoriságot is.
- **LRU-K közelítése** (pl. MySQL)
  - Két listát tart számon: fiatal és idős (young list, old list).
  - Az új blokkok mindig az idős lista elejére lesznek beszúrva (idővel hátrébb kerülnek).
  - Ha egy blokkot az idős listából újra elérnek, akkor a fiatal lista elejére kerül.
  - A fiatal lista végéről a blokkok az idős listába kerülnek, az idős lista végéről kiírjuk őket (lapcsere).

További megközelítések

- **Lokalizáció** (pl. PostgreSQL)
  - Egy tranzakcióhoz fix számú puffer helyet rendelünk, és csak azokkal dolgozhat.
- **Fontossági információk**
  - Egyes blokkok „fontosabbak” lehetnek, mint mások (pl. index gyökérblokk)
  - Ezeket érdemes mindig a pufferben tartani.
- **Piszkos puffer figyelembe vétele**
  - Ha egy blokk módosítva lett, akkor a lapcsere előtt ezt a blokkot ki kell írni a lemezre és csak utána lehet lecserélni. A kiírás időigényes, ezért érdemes lehet helyette más lapot lecserélni.

További megközelítések

- **Több pufferkészlet**
  - Az adatbázis több pufferkészletet is fent tart (pl. külön puffer az adatoknak és az indexeknek).
- **Scan sharing** (szinkronizált szkennelés)
  - Különböző lekérdezések használhatják ugyanazt a kurzort az adatok olvasásához (PostgreSQL, MSSQL)
  - Oracle-ben csak teljesen azonos lekérdezéseknél megengedett.
- **Pufferkészlet megkerülése** (buffer pool bypass)
  - Szekvenciális szkennelés esetén a blokkokat nem helyezzük a pufferbe.

Adatbázis fájlok

- Az adatbázisok egy vagy több fájlban tárolják az adatokat (és metaadatokat).
- Az OS nem tud semmit a fájlok tartalmáról, teljesen az ABKR kezeli őket.
- A különböző rendszerek általában valamilyen saját fejlesztésű formátumot használnak.
- A fájlok blokkok (block) vagy más néven lapok (page) kollekciója.

Adatbázis blokkok

- Egy blokk (vagy lap) egy fix méretű logikai adategység.
- A blokkok tartalmazzák a rekordokat, metaadatokat, indexebejegyzéseket stb.
- Általában egy blokk csak egy táblához kapcsolódó rekordokat tartalmaz.
    - Ellenpélda: klaszteren tárolt táblák (clustered tables).
- A blokkoknak van azonosítója:
    - Nem mindig egyedi az egész adatbázist tekintve.
    - Oracle-ben file_id+block_id azonosítja egyértelműen.

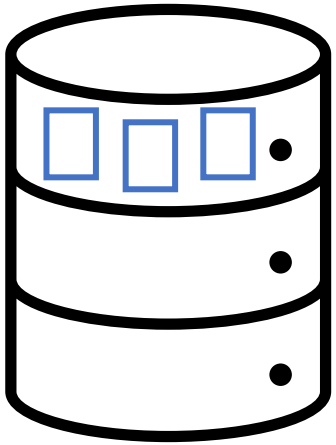
Különböző blokk fogalmak

- Hardver blokk (általában 4 KB)
  - Olyan adategység, amit az eszköz atomi módon tud írni/olvasni.
- OS blokk
  - A hardver blokk többszöröse lehet (általában 4 KB)
- **Adatbázis blokk** – innentől ezt értjük majd blokk alatt
  - A hardver blokk többszöröse lehet (512B – 32KB)
  - Az ABKR-en konfigurálható (egyes rendszerekben táblánként is eltérhet)

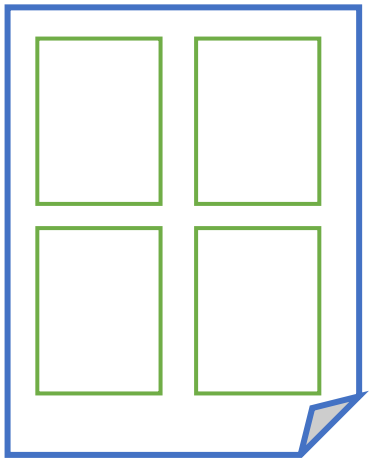
Alapértelmezett blokk méret	**ABKR**
4KB	SQLite, IBM Db2
8KB	Oracle, MS SQL Server

Adattárolás felülnézet

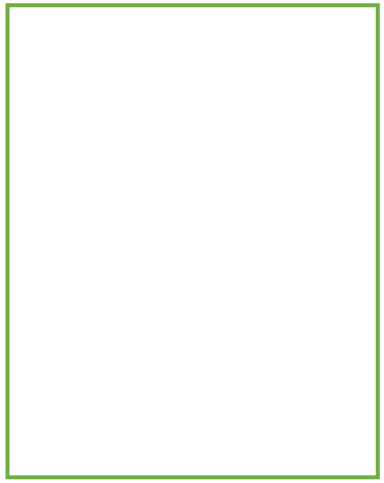
- A lemezen egy vagy több fájl van.
- A fájlok blokkokból állnak
- A blokkok tartalmazzák az adatokat – még ne foglalkozzunk azzal, hogy mi van a blokkokban.



Lemez



Egy fájl



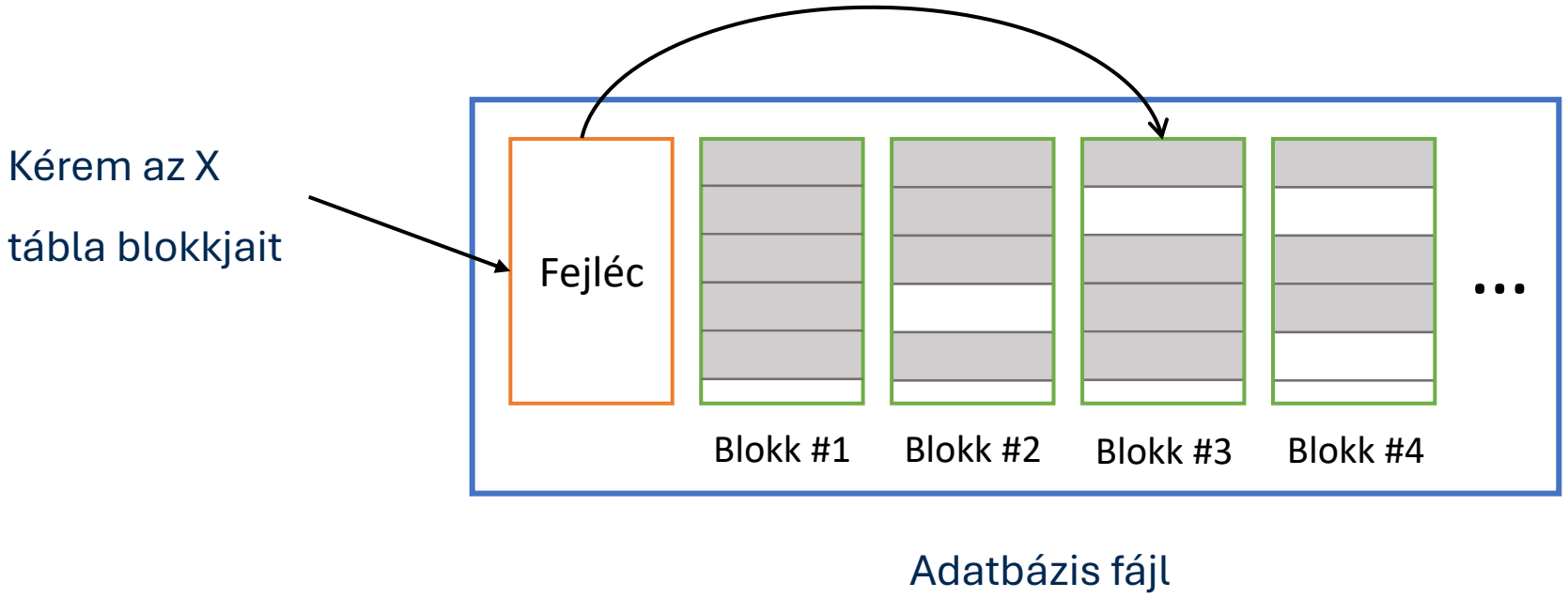
Egy blokk

Fájlszervezés

- A blokkokokat egy fájlon belül különböző módokon szervezhetjük:
  - Kupac szervezés (heap organization) - legnépszerűbb
  - Rendezett fájlok (sorted file organization) – régebbi megközelítés
  - Hasított fájlok (hashing file organization)
  - Fa-struktúra alapú szervezés
- A fájl mindig tartalmaz egy fejlécet, amely a fájl tartalmához kapcsolódó metaadatokat tárol:
  - Fájl mérete
  - Blokkméret (a fájlon belül)
  - DBMS verzió
  - Séma információk (a fájlban tárolt táblákról)
  - Ellenőrző összeg (checksum)

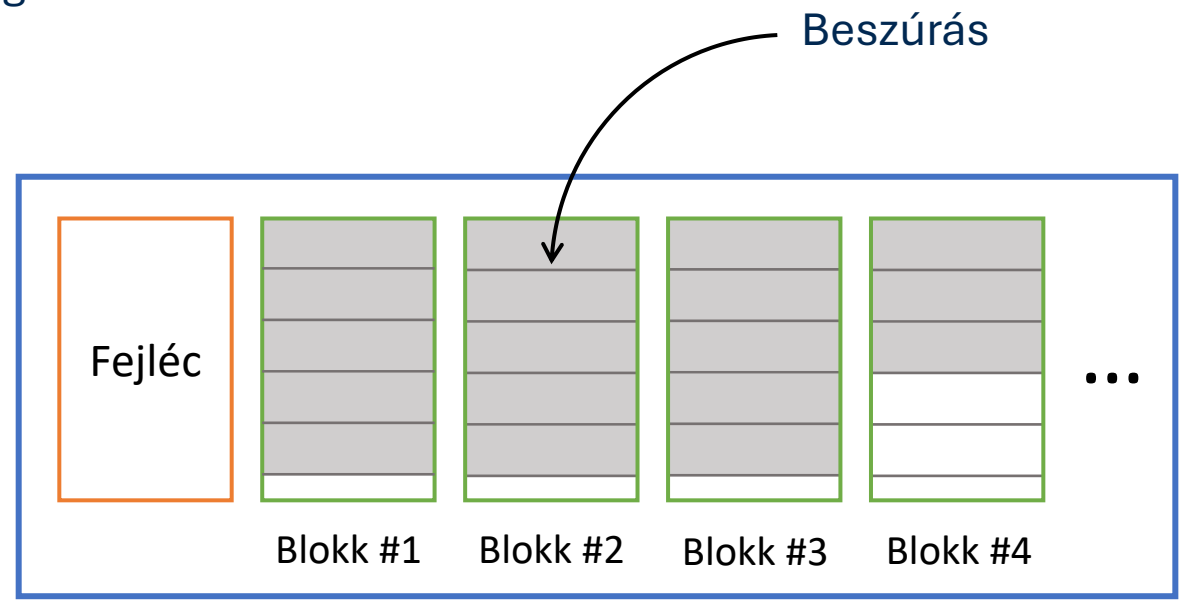
Kupac fájlszervezés

- Rendezetlen kollekciója a blokkoknak.
- Hatékony beszúrás.
- Lassú keresés:
  - Egyenlőségi keresés: átlagosan a blokkok felét kell beolvasni.
  - Intervallum keresés: minden blokkot be kell olvasni.



Rendezett fájlszervezés

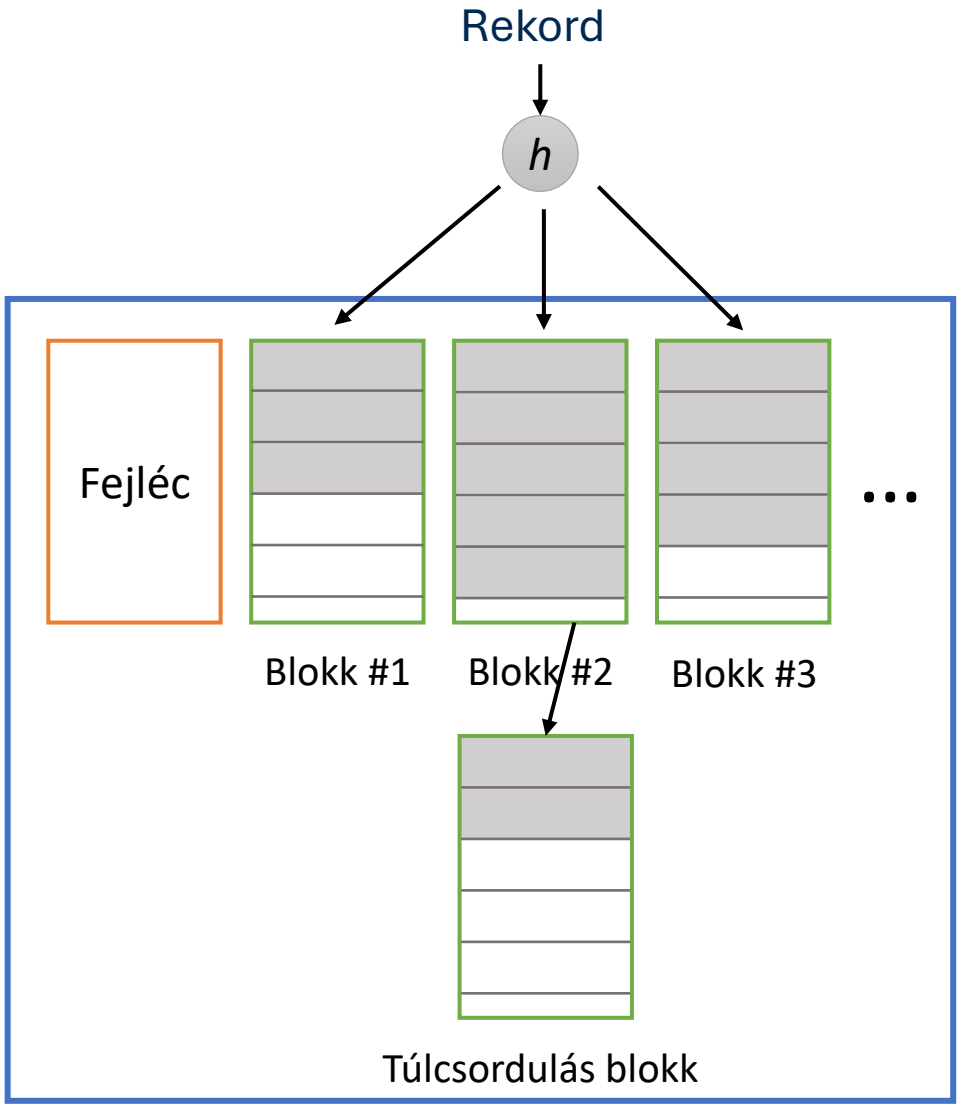
- Egy adott mező szerint rendezetten tároljuk a rekordokat és blokkokat.
- Lassú a beszúrás és a törlés.
- Gyors logaritmikus keresés.



Adatbázis fájl

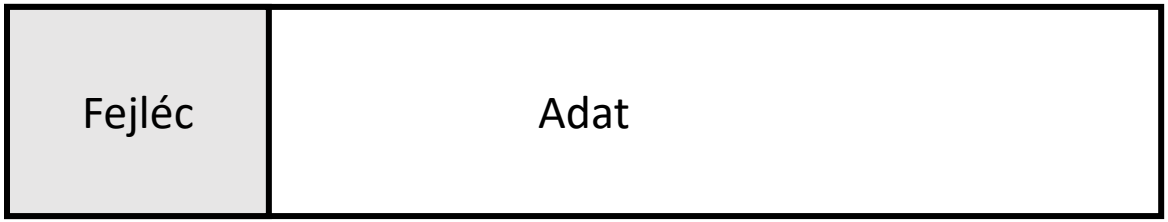
Hasított fájlszervezés

- Egy hasító függvény (*h*) kosarakba osztja a rekordokat.
- Hatékony beszúrás, törlés, egyenlőség keresés.
- Nem támogatja az intervallumos keresést.



Rekordok felépítése

- Egy rekord tartalmaz egy fejlécet és az adatokat (értékek).
- Fejléc tartalma lehet:
  - Rekordséma (vagy egy mutató a sémára)
  - Rekord azonosító
  - A rekord hossza
  - Időbélyegzők (pl. mikor volt módosítva)
  - stb.



Egy rekord vázlatos felépítése

Rögzített (fix) hosszúságú rekordok

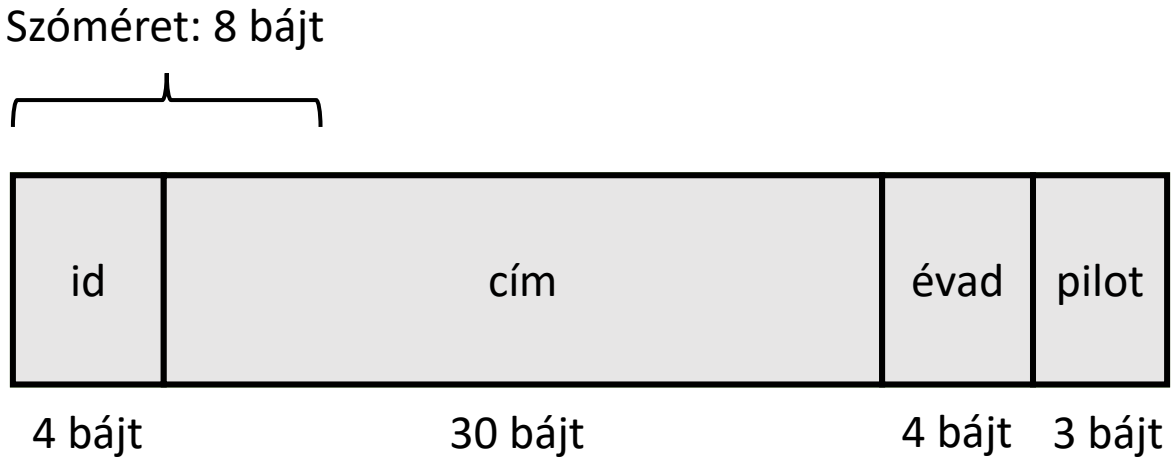
- Nézzük meg, hogyan néznek ki egy tábla rekordjai.

```
CREATE TABLE sorozatok (
    id INT PRIMARY KEY,
    cím CHAR(30),
    évad INT,
    pilot DATE
);
```

Részletesebben az adattípusokról (Oracle):
https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Data-Types.html

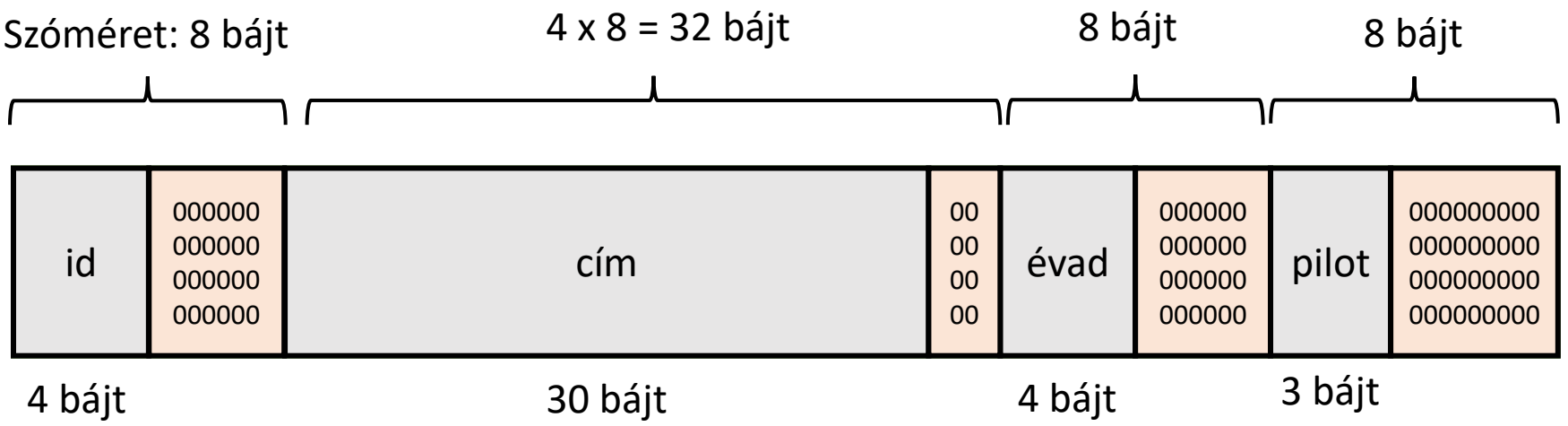
Memóriahozzáférés igazítás (word-aligment)

- Az adatokat a CPU „szóméretéhez” érdemes igazítani.
- Ez általában 4 (32 bit - ARM) vagy 8 bájt (64 bit – x64).
- Arra kell törekednünk, hogy a mezők a szóméret többszörösén kezdődjenek.



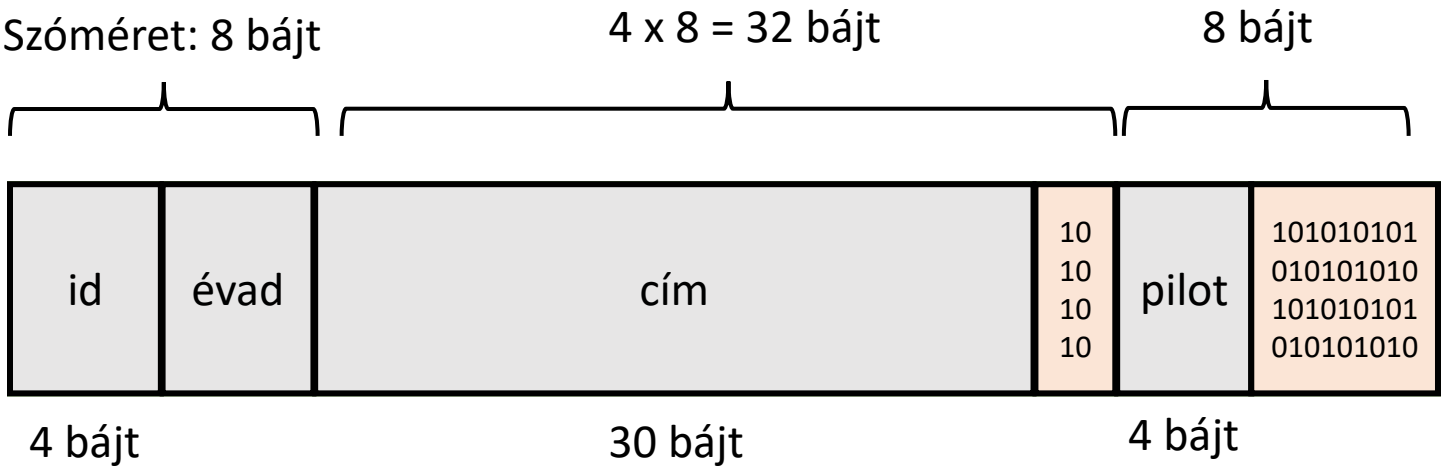
Word-aligment: felkerekítés

- A legegyszerűbb megoldás:
  - Minden mező hosszát kerekítsük fel a legközelebbi számra (amely a szóméret többszöröse)
- Ez a legnépszerűbb megoldás is (pl. Postgres ezt használja)



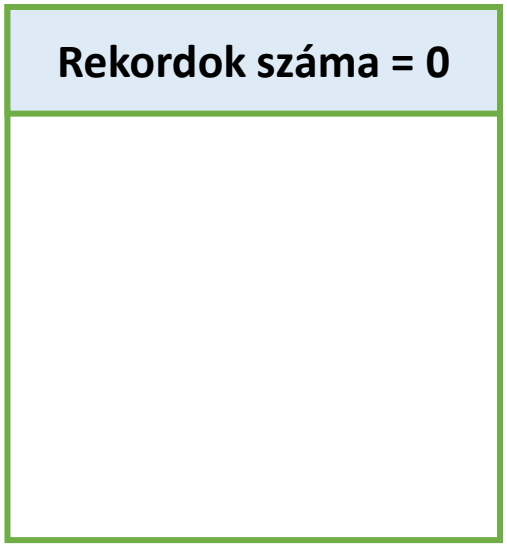
Word-aligment: átrendezés

- Ha meg tudjuk tenni, rendezzük át a mezők sorrendjét úgy, hogy a szóméretthez igazodjon.
- Pl: az *évad* kerüljön az *id* után.



Rögzített hosszú rekordok blokkokba pakolása

- Tartsuk nyilván a rekordok számát.
- Ha jön egy új rekord csak szúrjuk be, ahová tudjuk



Egy blokk vázlatos felépítése

Egyszerű megközelítés

1. lépés: beszúrtunk 3 rekordot a blokkba egymás után.
2. lépés: töröltük a 2. rekordot.
3. lépés: beszúrtunk egy újabb rekordot.

1. lépés 2. lépés 3. lépés

Egyszerű megközelítés - megjegyzések

- A törlés kezelése az egyszerű esetben (különböző megközelítések):
  - Hagyjuk üresen a helyet, ha jön új rekord szúrjuk be oda (előző slide).
  - Ha törlünk egy rekordot középről, az utána lévő rekordokat mozgassuk egyel lentebb.
  - Ha törlünk egy rekordot középről, akkor az utolsó rekordot tegyük be a helyére.
- Nem tárolunk fél rekordot a blokkokban, azaz ha egy rekordnak csak egy része férne el, akkor más blokkba tesszük. Így a blokkokban gyakran marad egy kis üres hely.
- Az egyszerű megközelítésnek van egy fontos **hiányossága**:
  - Csak rögzített hosszúságú rekordok esetén működik jól!

Változó hosszúságú mezők

- Pl. Oracle-ben: CHAR és VARCHAR2
- Például tároljuk az „ELTE” szót. Fix hosszúságnál a maradék részt kitöltjük speciális töltelékkarakterekkel:
  - CHAR(8) – [”E”, ”L”, ”T”, ”E”, ”⊥”, ” ⊥”, ” ⊥”, ” ⊥”]
- Változó hosszúságnál két lehetőségünk is van.
  - Nullkarakterrel jelöljük a végét: [”E”, ”L”, ”T”, ”E”, ”Ø”]
  - A karaktersorozat előtt tároljuk a hosszát: [4, ”E”, ”L”, ”T”, ”E”]
- **Kérdés**: melyik változatát érdemes használni a változó hosszúságú adatok tárolásnak?

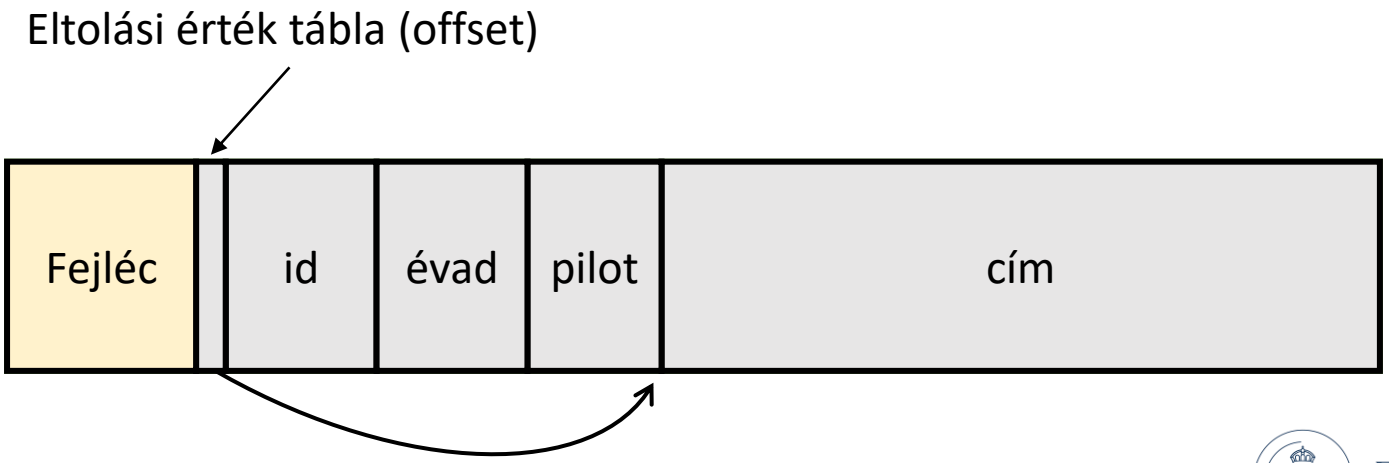
Változó hosszúságú mezők

- Az ABKR-ek általában mindig a második megoldást választják (hossz tárolása).
- Előnyök:
  - A karaktersorozat hosszát könnyen meg lehet határozni.
  - Bináris adatokat is lehet így tárolni.

Változó hosszúságú mezők

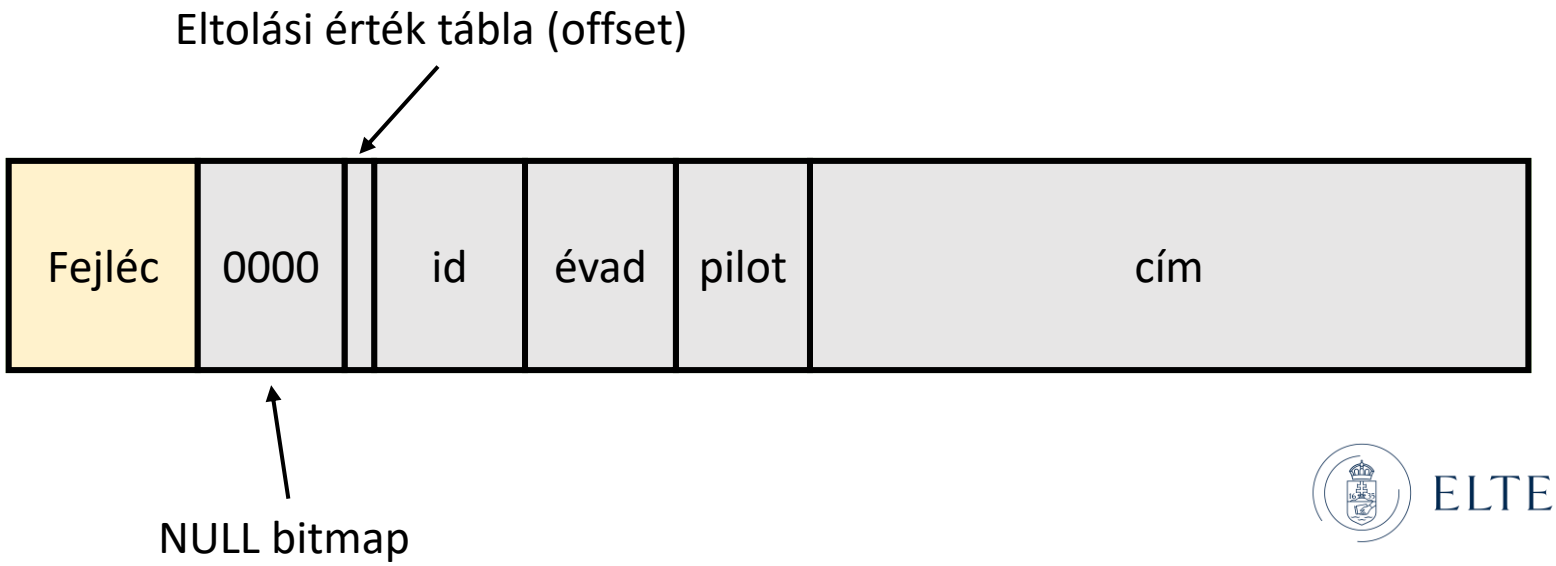
- CHAR helyett használjunk VARCHAR2-őt.
- A fix hosszúságú értékek előre kerülnek.
- Az eltolási érték tábla minden változó hosszúságú mezőhöz tartalmaz egy mutatót.

```
CREATE TABLE sorozatok (
    id INT PRIMARY KEY,
    cím VARCHAR2(30),
    évad INT,
    pilot DATE
);
```



Null értékek jelölése

- Egyszerű megközelítés: használjunk egy speciális értéket, ami a NULL-t jelzi.
  - INT esetén pl. INT32_MIN
  - Probléma: később ezt nem szabad használni.
  - Főleg oszlop-orientált rendszerek használják.
- **Általában használt módszer:** NULL bitmap használata.

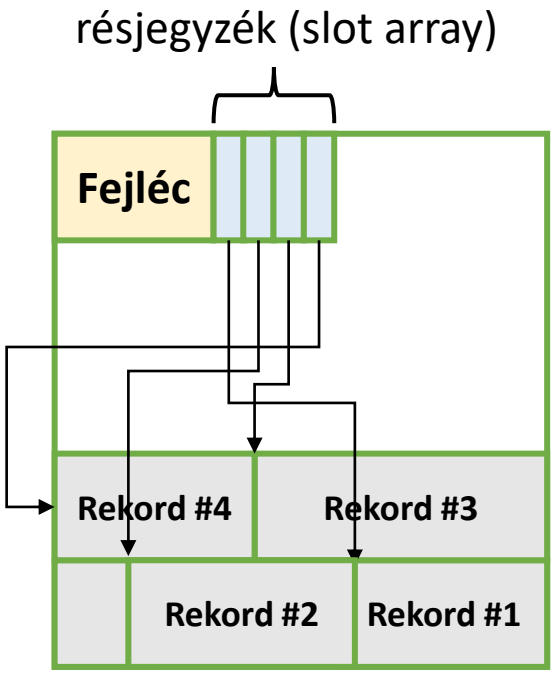


Nagy értékek tárolása

- Mi történik akkor, ha az egyik mezőben olyan nagy értéket szeretnénk tárolni, ami nem fér el egy blokkban?
- **Belső tárolás** (internal): Egy határig (Oracle: 4000 bájt) képes a rendszer belsősleg tárolni. Ilyenkor speciális LOB (Large OBject) blokkokba kerülnek a nagy adatok, a rekordban pedig mutató van elhelyezve.
- **Külső tárolás** (external): Egy bizonyos méretnél minden rendszer „külsőleg” tárolja a fájlokat, ilyenkor ismét egy mutató kerül a rekordba, amivel elérhetjük ezt a külső fájl. Ezek nem részei az adatbázis tranzakcióknak, az operációs rendszer kezeli őket (pl. BFILE Oracle-ben).

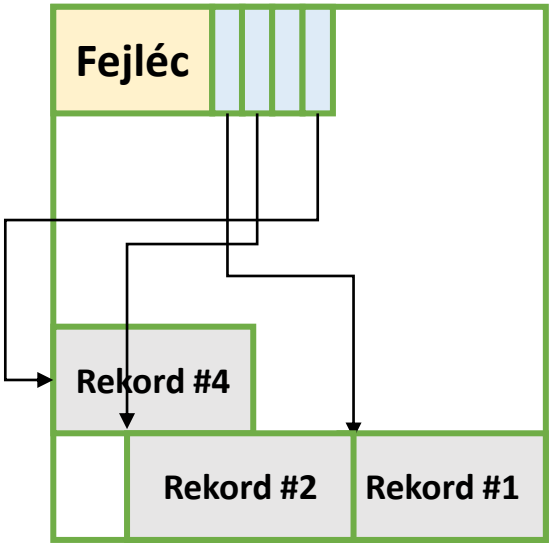
Változó hosszúságú mezők a blokkokban

- Miért nem jó az egyszerű megközelítés ebben az esetben?
  - Törlés után üresen maradt helyek hossza változó lehet.
  - Beszúráskor elég nagy helyet kell keresni.
- **Megoldás**: réses lapszerkezet (slotted pages)

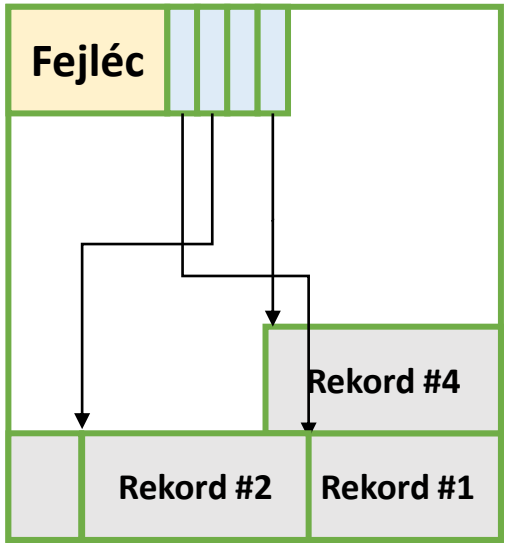


Slotted pages - törlés

- Törléskor a slot-ot kiürítjük, a rekordot töröljük.
- Mit csináljunk az üres hellyel?
  - Nem csinálunk semmit (Postgres) vs. átpakoljuk a rekordokat (Oracle).
- **Kérdés**: miért nem jó átpakolni?



Töröljük a 3. rekordot



Átrendezés

Rekord azonosítók

- Az ABKR-ek hozzárendelnek minden rekordhoz egy egyedi azonosítót.
- Tartalmazza:
  - Fájl azonosító
  - Blokk azonosító
  - Rekord azonosító a blokkon belül
- Ezt az ID-t nem tároljuk a rekordokban (általában), a rekord pozíciójából következik.
  - Pl. SQLite ezt használja elsődleges kulcsként (rejtve).
- Azonosítók különböző rendszerekben:
  - Oracle: ROWID (10 bájt)
  - SQLite: ROWID (8 bájt)
  - PostgreSQL: CTID (6 bájt)
- Az alkalmazásoknak nem szabad ezekre hivatkozni (pl. áthelyezés miatt)

Rendszerkatalógusok

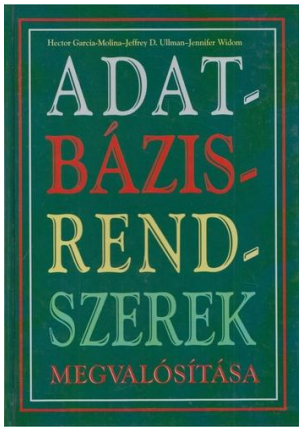
- A relációs ABKR-ek táblákban tárolnak metaadatokat az adatbázisban található objektumokról:
  - Relációk és attribútumaik nevei, típusai
  - Nézetek neve és definíciója
  - Felhasználók, jogosultságok
  - Statisztikák az adatokról – később fontos lesz!
  - Fizikai fájlszervezéshez kapcsolódó információk:
    - Pl. melyik fájlban találhatók egy reláció sorai
- A gyakorlatokon ezeket részletesen megnézzük.

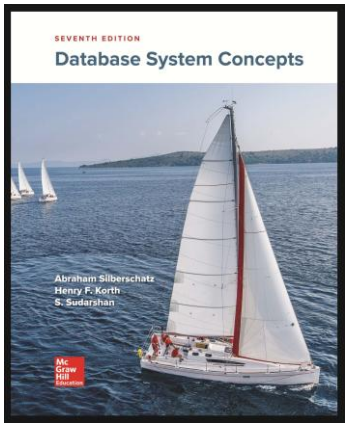
Tankönyv fejezetek

- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom:

  Adatbázisrendszerek megvalósítása

  - 2. fejezet: Adattárolás (46-109. oldalak)
  - 3. fejezet: Adatelemek ábrázolása (111-152. oldalak)
- Silberschatz, Korth, & Sudarshan: **Database System Concepts**
  - Chapter 12. Physical Storage Systems
  - Chapter 13. Data Storage Structures





Összegzés

- A lemez alapú ABKR-ek rendelkeznek valamilyen nem-felejtő másodlagos tárolóval (pl. HDD, SSD).
- A pufferkezelő a memória és a lemez között mozgatja a blokkokat.
- A lemezen fájlokban tároljuk az adatbázis blokkokat.
- Az adatbázis blokkok tartalmazzák a rekordokat és különböző metaadatokat.