

Operációs rendszerek I.

Tárkezelés

Várkonyiné Kóczy Annamária

Professzor

Informatika Tanszék

[\(koczy@ujs.sk\)](mailto:(koczy@ujs.sk))

varkonyi-koczy@uni-obuda.hu

Felhasznált irodalom:

- Kóczy-Kondorosi (szerk.): Operációs rendszerek mérnöki megközelítésben
- Tanenbaum: Modern Operating Systems 2nd. Ed.
- Silberschatz, Galvin, Gagne: Operating System Concepts

6. Tárkezelés

- Bevezetés
- A program címeinek kötése
- Társzervezési elvek
- Egy- és többpartíciós rendszerek
- Szegmens- és lapszervezés

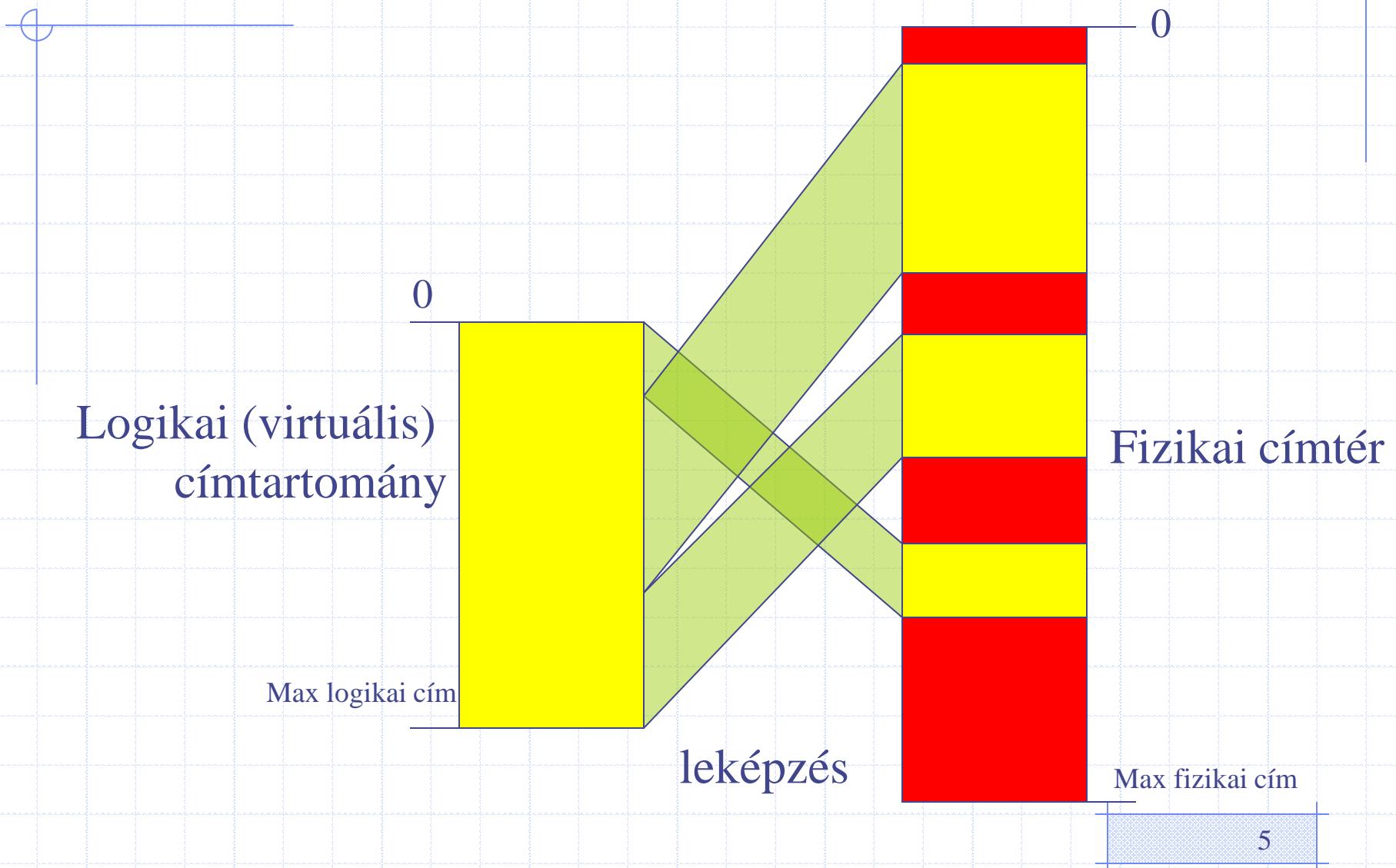
Bevezetés

- A központi tár (*main storage, memory*)
 - Szervezése és
 - kezeléseaz OS tervezését, implementálását és teljesítményét befolyásoló egyik legfontosabb tényező.
- A multiprogramozás igénye és a programmációk növekedése a *valós tár* kezelésén túl megkövetelte a *virtuális tár* kezelésének hardver és szoftver technikáit.

6.1. A program címeinek kötése

- Logikai címtartomány:
 - Folytonos címtartomány
 - 0-tól kezdődik
 - Lineárisan nő
 - a maximális értékig.
- Fizikai címtartomány: a gyakorlatban
 - nem 0 fizikai címtől kezdve történik a programok végrehajtása,
 - sokszor nem folytonos memóriaterület áll rendelkezésre, allokálás nem lineárisan
- A két tartomány között a megfeleltetés a leképzés (mapping).

A címleképzés



A címek kötésének lehetőségei

statikus

dinamikus

- Fordítás közben (compile time)
 - A fordítóprogram a program és az adatterület elemeihez abszolút címet rendel. Merev technika, elsősorban ROM-ban lévő programok esetében alkalmazzák.
- Szerkesztés közben (link time)
 - A függetlenül lefordított modulok saját logikai címtartományt használnak. A linker feladata, hogy az összes modult - egymás mögé - elhelyezze a fizikai tárba, valamint feloldja a modulok kereszthivatkozásait.
- Betöltés közben (load time)
 - A fordító áthelyezhető kódot generál, ennek a címhivatkozásait a betöltő program az aktuális címkiosztás szerint módosítja.
- Futás közben (run time)
 - A program csak logikai címeket tartalmaz, speciális hardver elemek határozzák meg a címet az utasítás végrehajtásakor.

A címek kötésének lehetőségei



A logikai és fizikai címek kapcsolata

- Logikai (virtuális) cím:
 - az a memóriacím, amit a CPU generál
- Fizikai cím:
 - a memória valós címe
- A logikai és fizikai címek
 - megegyeznek: statikus címkötés esetén
 - különböznek: dinamikus címkötés esetén

A memória-menedzsment egység (MMU)

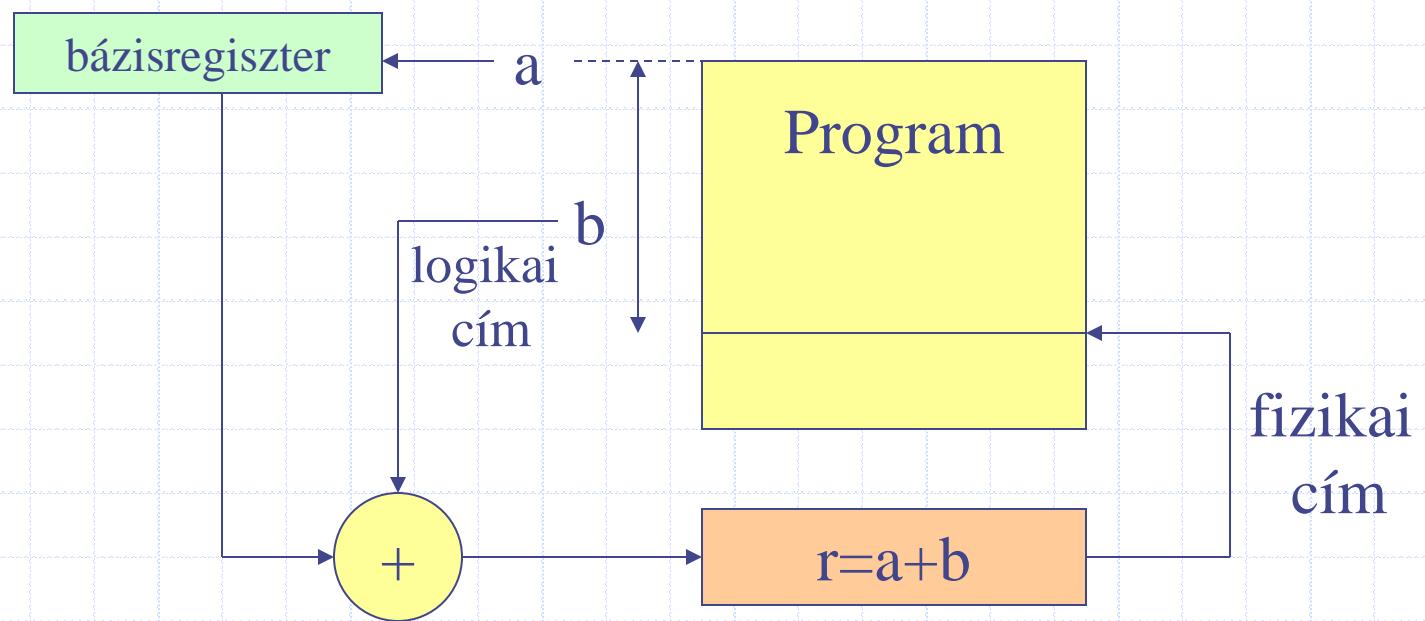
- Memory-Management Unit, MMU:
 - az a hardver egység, amely a virtuális címeket fizikai címmé alakítja
- Megjegyzés: a felhasználói programok csak virtuális címekkel operálnak, soha nem látják a valós fizikai címeket!

6.2 A dinamikus logikai-fizikai címleképzés alapvető módszerei

- Bázis-relatív címzés
- Utasításszámláló-relatív címzés

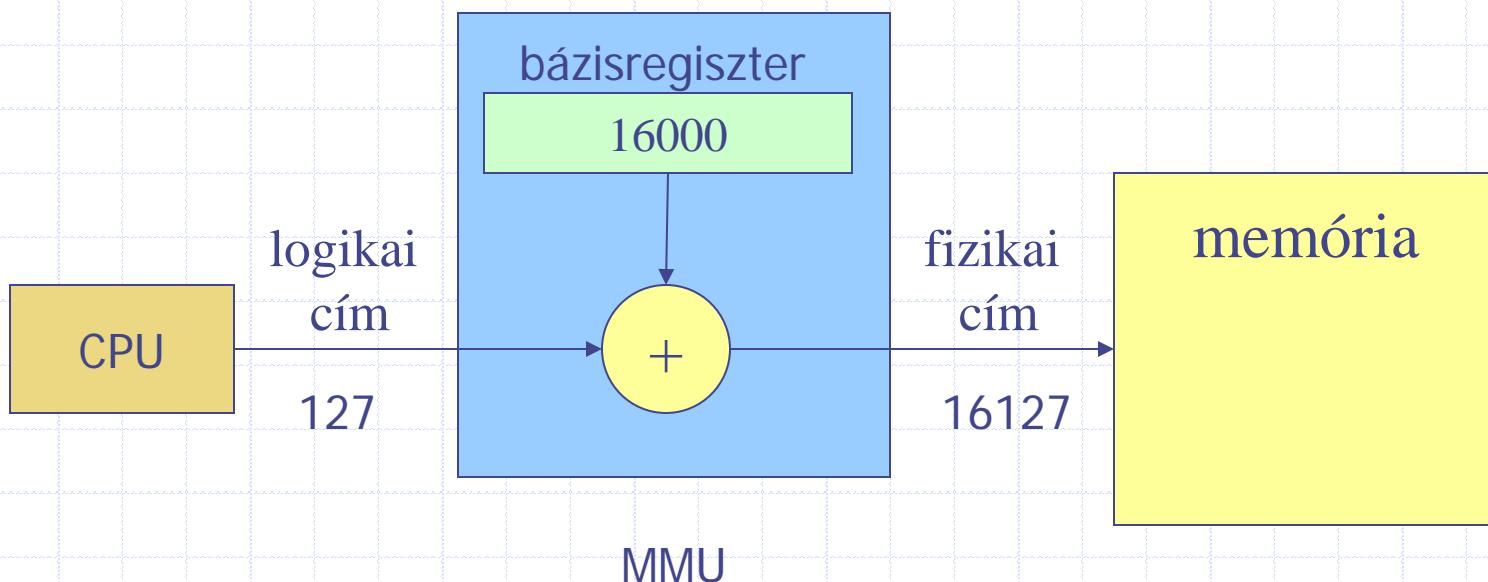
Bázis-relatív címzés

- A program tetszőleges helyre betölthető
- A bázisregisztert a betöltési kezdőcímre állítva a program végrehajtható.



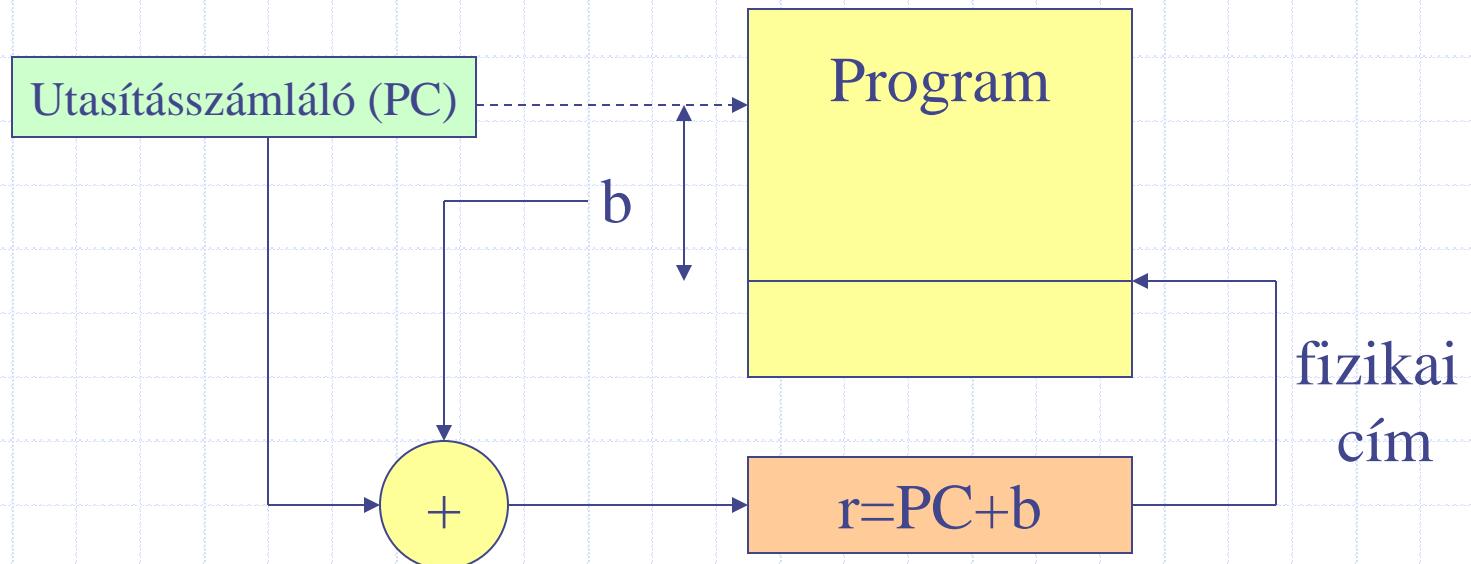
Bázis-relatív címzés példa

- A program tetszőleges helyre betölthető
- A bázisregisztert a betöltési kezdőcímre állítva a program végrehajtható.



Utasításszámláló relatív címzés

- Pozíció-független kód:
 - csak pozíció-független virtuális címeket tartalmaz



6.3. Programok végrehajtása kisebb tárterületen

- Az eddigi módszereknél a fizikai memória mérete behatárolta a futtatható kód méretét (hiszen az egész programnak és a kapcsolódó adatoknak egyszerre a memóriában kellett lennie)
- Ötlet: nem kell az egész programnak a memóriában lennie!

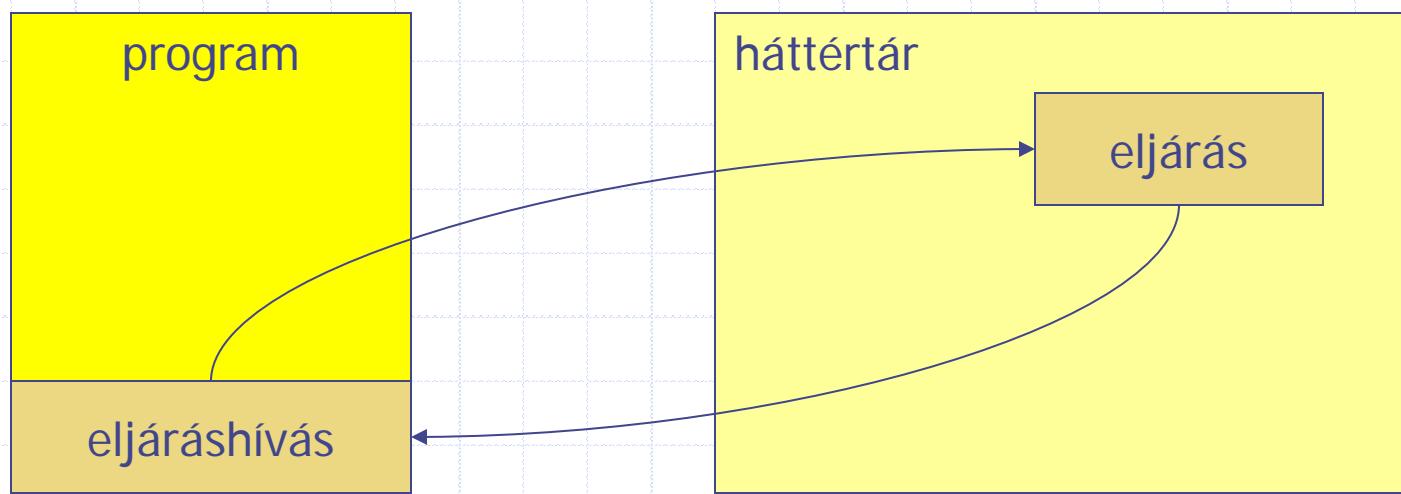
Késleltetett betöltés

- Futás közben nincs az egész program a tárban. Szükség esetén töltődnek be egyes programrészek.
 - dinamikus betöltés
 - *dynamic loading*
 - dinamikus kapcsolatszerkesztés
 - *dynamic linking*
 - átfedő programrészek
 - *overlay*

Dinamikus betöltés (dynamic loading)

- A programhoz tartozó egyes eljárások a háttértáron vannak, ha valamelyikre szükség van, azt egy speciális programrészlet ezt betölti.
- Programozó szervezi meg, az OS nem nyújt támogatást.
- Pl. ritkán használt részek, hibakezelés, stb.
- Lépései:
 1. Hívás
 2. Ellenőrzés: memóriában van?
 3. Speciális programrész betölti a memóriába, vezérlést átadja

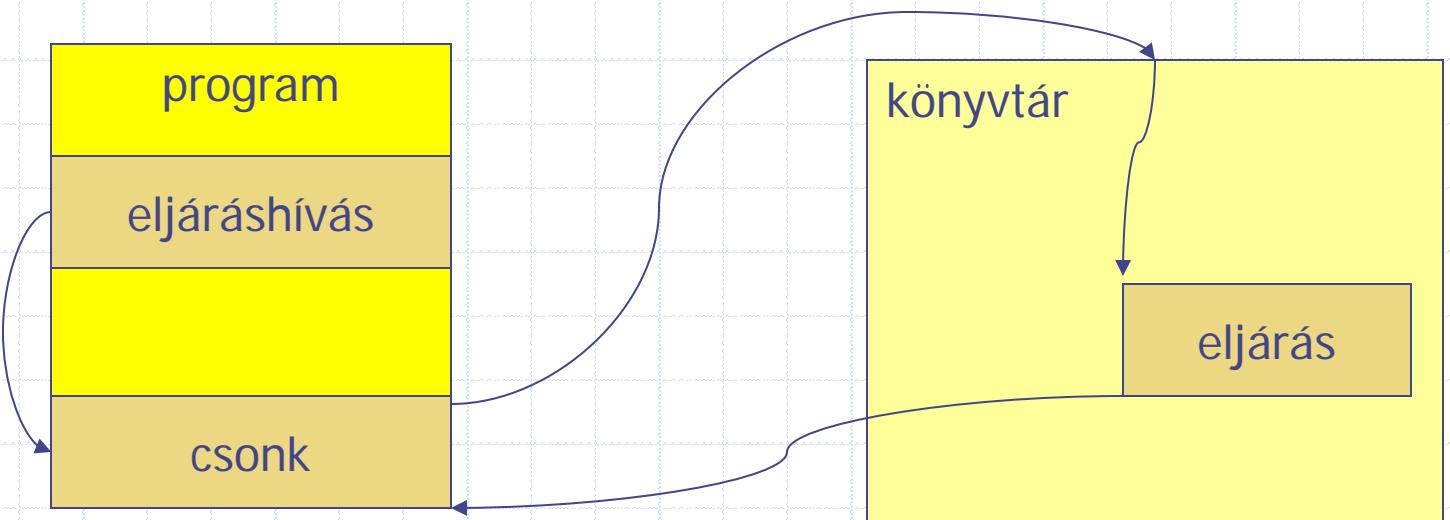
Dinamikus betöltés (dynamic loading)



Dinamikusan betöltött könyvtárak (dynamic linking)

- A dinamikus betöltés változata, az OS támogatásával.
- A programban használt rendszerkönyvtárak eljárásai helyett csak egy csonk (*stub*) kerül a programba
- A csonk tartalmaz egy hivatkozást
 - a könyvtárra
 - és az azon belüli eljárásra.
- Az csonk első meghívásakor az OS az eljárást betölti a tárba.
- A következő hívások már az eljárást hívják meg időveszteség nélkül.

Dinamikusan betöltött könyvtárak (dynamic linking)



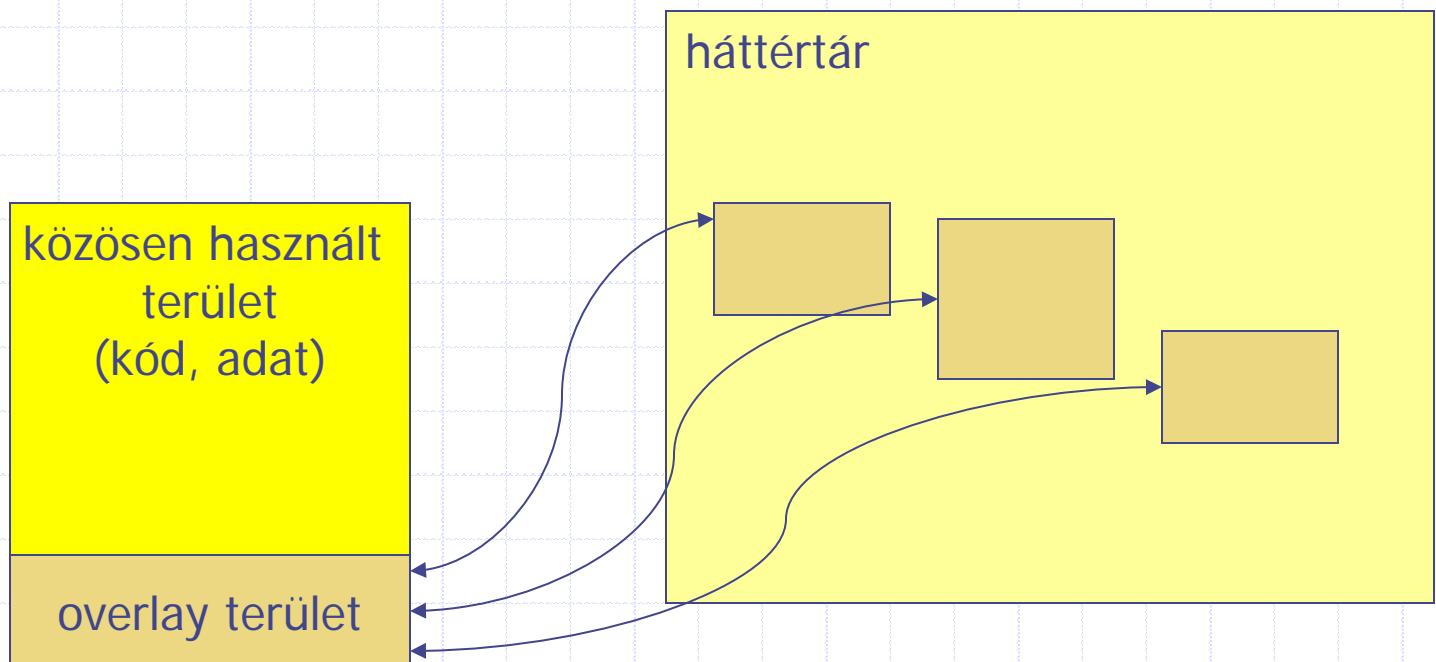
Dinamikusan betöltött könyvtárak tulajdonságai

- Előny:
 - csökken a tárfelhasználás
 - hibás eljárás javításakor nem kell az összes programot újrafordítani
- Hátrány:
 - Verzió-kontrol nehéz: „dll pokol”
- Pl.:
 - Windows dll
 - Unix shared library

Átfedő programrészek (overlay)

- A program részekre bontása:
 - közös adat- és programrészek (nem változik)
 - olyan átfedő részek, amelyek közül egy időben csak egyre van szükség.
- Az átfedő részeket egyesével töltjük be a memóriába.
- Nincs szükség OS támogatásra.
- Az átfedés számára fenntartott tárterület a legnagyobb átfedő programrész hosszával egyenlő.

Átfedő programrészek (overlay)



6.4. Memóriaallokációs elvek

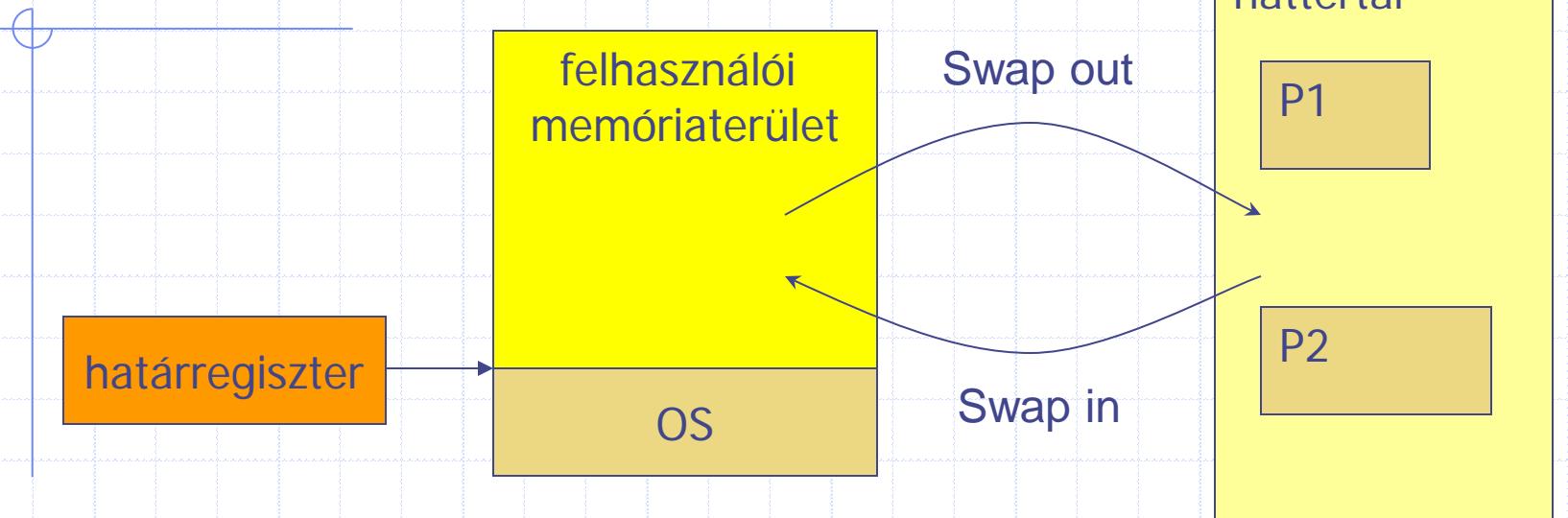
- 6.4.1. Egypartíciós rendszer
- 6.4.2. Többpartíciós rendszer
- 6.4.3. Tárcsere (swap)
- 6.5. Korszerű módszerek
 - 6.5.1. Szegmens szervezés
 - 6.5.2. Lap szervezés

6.4.1. Egypartíciós rendszerek

Memóriaszervezés:

- Védett területek:
 - OS
 - speciális tárterületek
 - megszakítás vektorok
 - periféria címtartományok
- Felhasználói terület:
 - A nem védett területen felüli cím-tartományt csak egy folyamat használja.
 - A program az első szabad címre töltődik
- Ha az OS-nek szüksége van memóriára, akkor
 - vagy áthelyezi a programot,
 - vagy a nem használt területről allokál.

Egypartíciós rendszer



Védelem: **felhasználói** és **rendszer** mód

- Az OS területének védelmére elég egy regiszter, amely a program legkisebb címét tartalmazza
- Felhasználói mód:
 - Futás közben egy hardver figyeli, hogy minden hivatkozás a tárolt cím felett legyen.
- Rendszer mód:
 - Rendszerhíváskor a védelem kikapcsol, az OS az egész címtartományt eléri.

6.4.2. Többpartíciós rendszerek

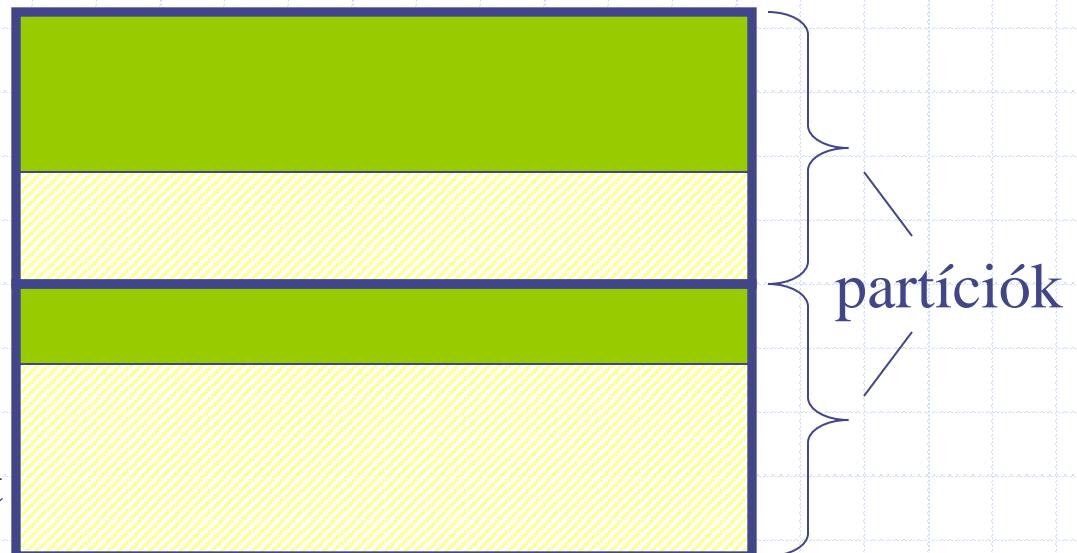
- A multiprogramozás elve megköveteli, hogy több folyamat legyen egy időben a tárban.
- Lehetséges megoldások:
 - Fix partíciós rendszerek:
 - az OS feletti tárterületet partíciókra bontják
 - a határok nem változnak.
 - Változó partíció méretű partíció
 - a program igényeinek megfelelő partícióméret

Belső tördelődés

- Belső tördelődés (internal fragmentation):
 - A folyamatok nem használják ki a rendelkezésére bocsátott partíciót.

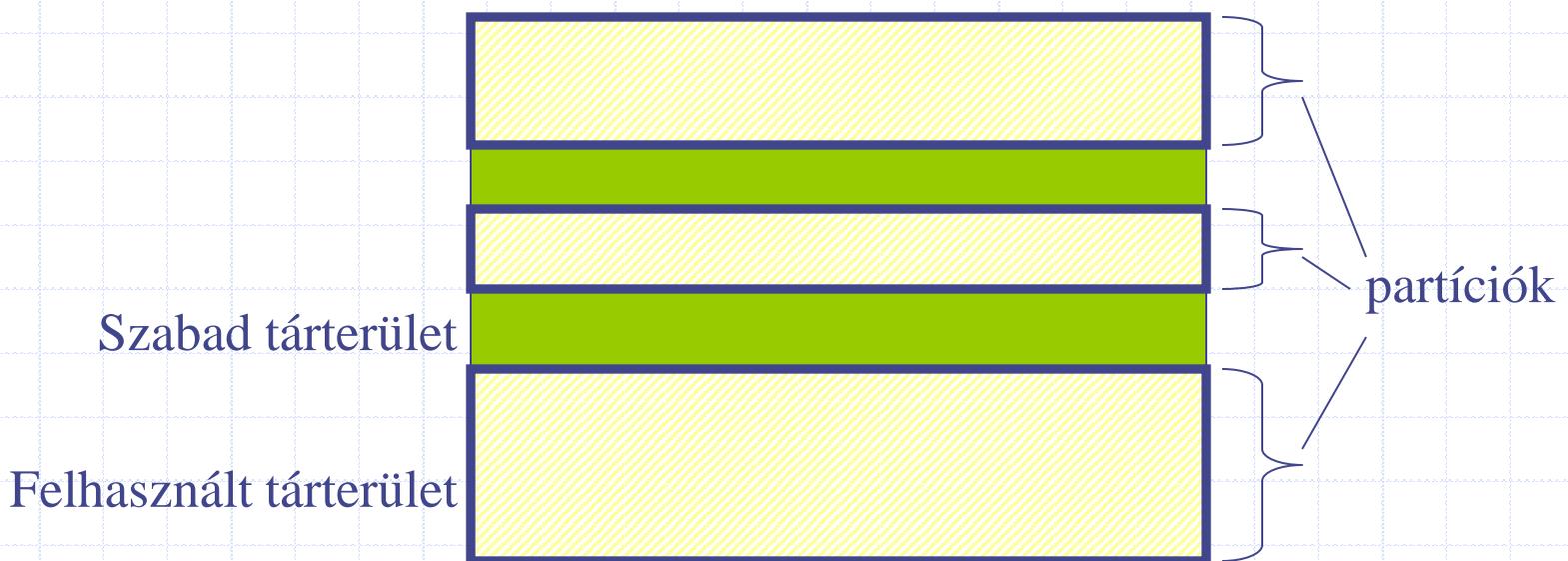
Szabad tárterület

Felhasznált tárterület



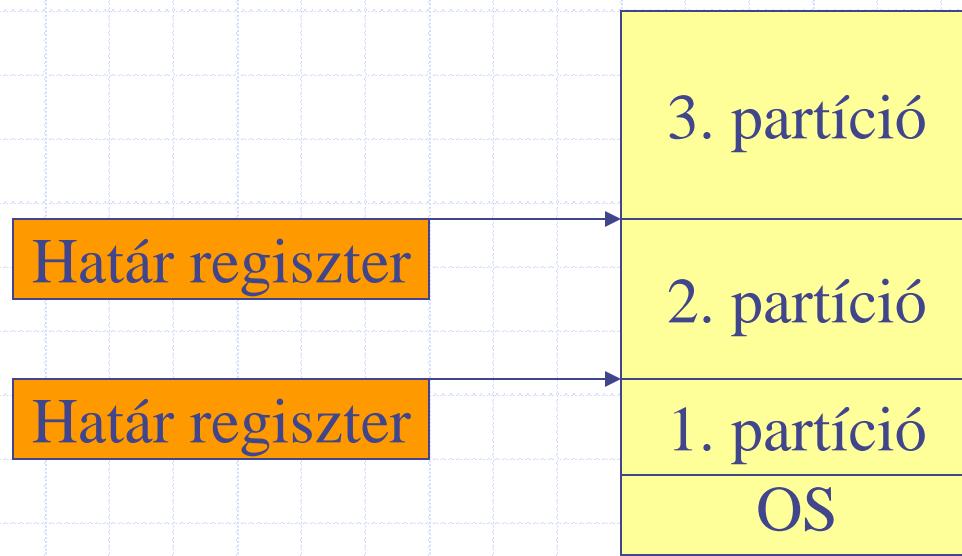
Külső tördelődés

- Külső tördelődés (external fragmentation):
 - A szabad memória kis, egymással nem szomszédos részekre oszlik



Fix partíciós rendszerek

- Az OS feletti tárterületet fix partíciókra bontják.
- A határok nem változnak.
- Rossz hatékonyság: *belső tördelődés*
- Védelem: határ-regiszterek



Változó partíciós rendszerek

- A partíció a program igényeinek megfelelő méretű
- Problémák: Szabad terület tördelődése
 - Egy folyamat lefutásakor a használt memória felszabadul.
 - Az OS nyilvántartja ezeket a területeket, az egymás melletti szabad területeket automatikusan összevonja.
 - Sokszor ezen területek nem szomszédosak, így a szabad memória kis részekre oszlik (*külső tördelődés*).
- Belső tördelődés nincs, hiszen csak a szükséges memóriát kapják meg a folyamatok.

Szabad területek tömörítése

compaction, garbage collection

- A külső tördelődés bizonyos fok után lehetetlenné teszi újabb folyamat elindítását (elég a szabad terület, de a leghosszabb összefüggő szabad terület nem elég a folyamatnak).
- Megoldás: a szabad helyek tömörítése.
- Tömörítő algoritmusok:
 - időigényes (nem biztos hogy megéri futtatni, esetleg jobban járunk, ha megvárjuk, míg néhány folyamat befejeződik).
 - HW támogatást igényel.
 - váratlanul lehet szükség rá (ezért pl. egy interaktív rendszer válaszideje hirtelen megnőhet)
 - a tárterületek mozgatását körültekintően kell végrehajtani (az áthelyezési információk megőrzése, stb.)

Memóriaterületek lefoglalása

Stratégiák a külső tördelődés csökkentésére.

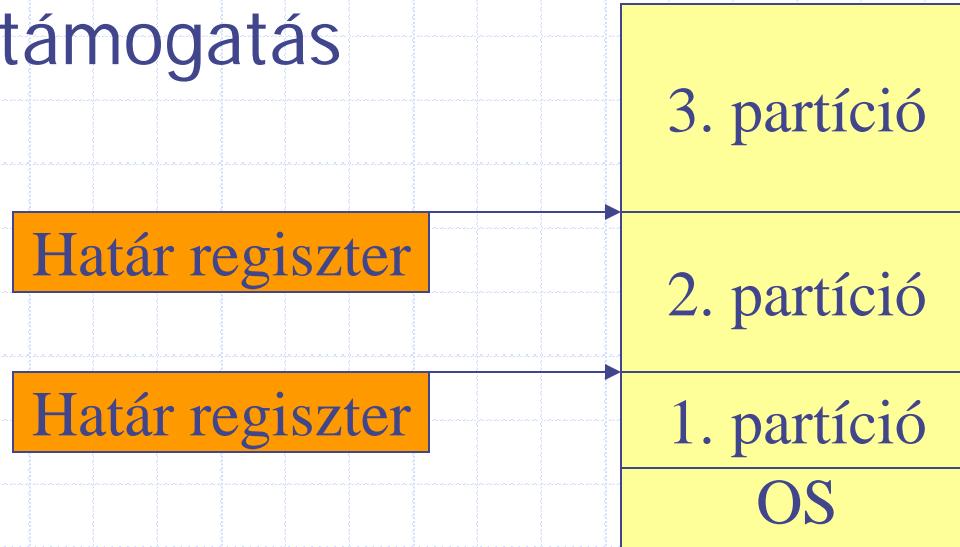
Az OS a szabad területek közül a következőképpen választhat:

- legjobban megfelelő (*best fit*)
 - legkisebb még elegendő méretű
- első megfelelő (*first fit*)
 - a kereséssel a tár elejétől indulunk, az első megfelelő méretűt lefoglaljuk.
- következő megfelelő (*next fit*)
 - a kereséssel az utoljára lefoglalt tartomány végéről indulunk, az első megfelelő méretűt lefoglaljuk.
 - Igen gyors algoritmus, a memória 30%-a marad kihasználatlan
 - (50%-os szabály, mivel a folyamatok által foglalt memória fele nincs kihasználva).
- legrosszabban illeszkedő (*worst fit*)
 - a legnagyobb szabad területből foglaljuk le, abban bízva, hogy a nagy darabból fennmaradó terület más folyamat számára még elegendő lesz.

Hatókonyság szempontjából a legrosszabban illeszkedő a leggyengébb, a többi nagyjából azonos.

Védelem

- Multiprogramozott rendszerekben nemcsak az OS, hanem más folyamatok területeit is védeni kell.
- 2 regiszter kell, amelyek a címtartomány határait tartalmazzák
- hardver támogatás



6.4.3. Tárcsere (swap)

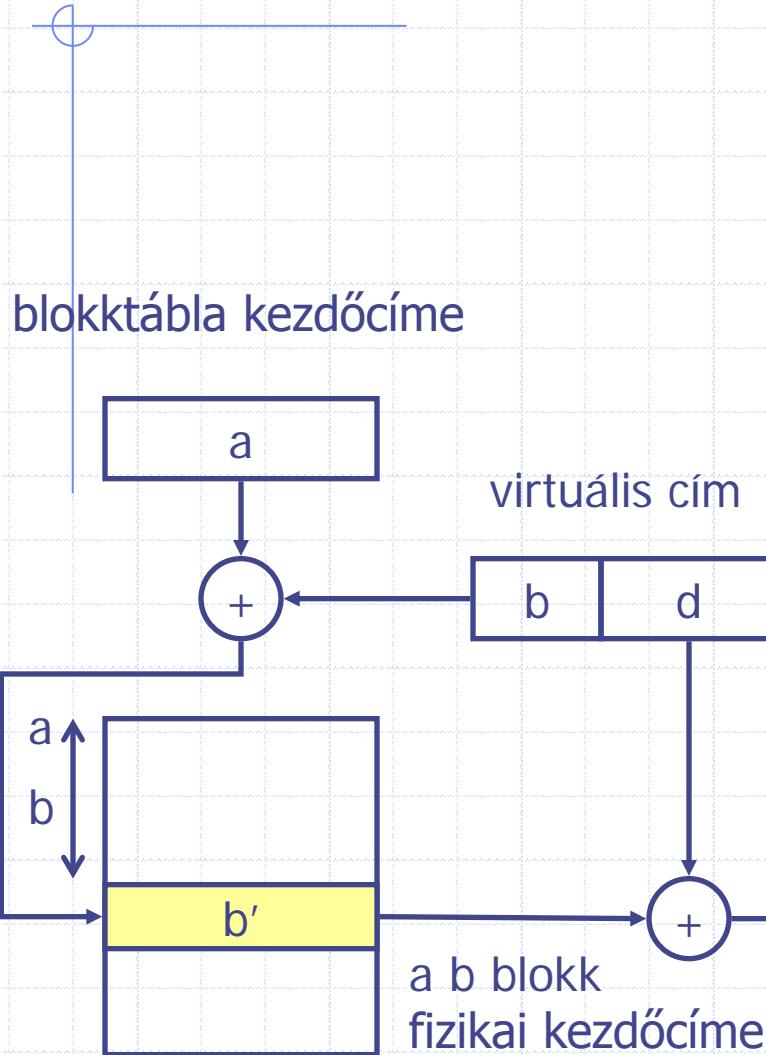
- A tárcsere során az OS egy folyamat [teljes] tárterületét a háttértárra másolja, így szabadítva fel területet más folyamatok számára.
- A perifériás átvitel miatt időigényes (sokkal több idő, mint egy környezetváltás, CPU ütemezéskor ezt figyelembe kell venni).
- Problémák:
 - Melyik folyamatot tegyük ki háttértárra?
 - Figyelembe kell venni a folyamat állapotát, prioritását, futási, várakozási idejét, vagy a lefoglalt partíció nagyságát.
 - Melyik folyamatot mikor hozzunk be a háttértárról?
 - Az előző szempontok itt is érvényesek. Ügyelni kell a kiéheztetés veszélyére.
 - Kerülendő a folyamatok felesleges pakolgatása.

6.5. A tárcsere korszerű módszerei

Hardver támogatás

- Mesterséges folytonosság (*artificial continuity*)
 - Virtuális címtartományban folytonos program a valóságban nem az.
- A futó folyamatoknak nem az egész címtartománya van a központi tárban.
 - A nem érvényes hivatkozás (nincs a központi tárban) esetén automatikus programrészlet betöltés (OS – hardver együttműködés).
- Közös vonás a *futás közbeni címleképzés*.
 - Csak hardver támogatással viselkedik megfelelő sebességgel.

Futás közbeni címleképzés



- Virtuális cím: $\langle b, d \rangle$
 - b : blokkcím,
 - d : eltolás, *displacement*
- A transzformáció a blokktábla segítségével megy végbe.
- minden folyamatnak saját blokktáblája van.
- A folyamatok virtuális címtartománya fedi egymást, de a fizikai címtartomány természetesen nem.

6.5.1. Szegmens szervezés

- A logikai címtartományban a program memóriája nem egybefüggő terület, hanem önmagukban folytonos blokkok (**szegmensek**) halmaza.
- A szegmens *logikai egység*. Pl.:
 - főprogram,
 - eljárások, függvények, módszerek, objektumok,
 - lokális változók, globális változók,
 - stack, szimbólumtábla, stb.
- A címtranszformáció az előző általános modellnek megfelelő (lásd előző ábra!).
 - Blokktábla → szegmenstábla
 - Cím: <szegmenscím, eltolás>

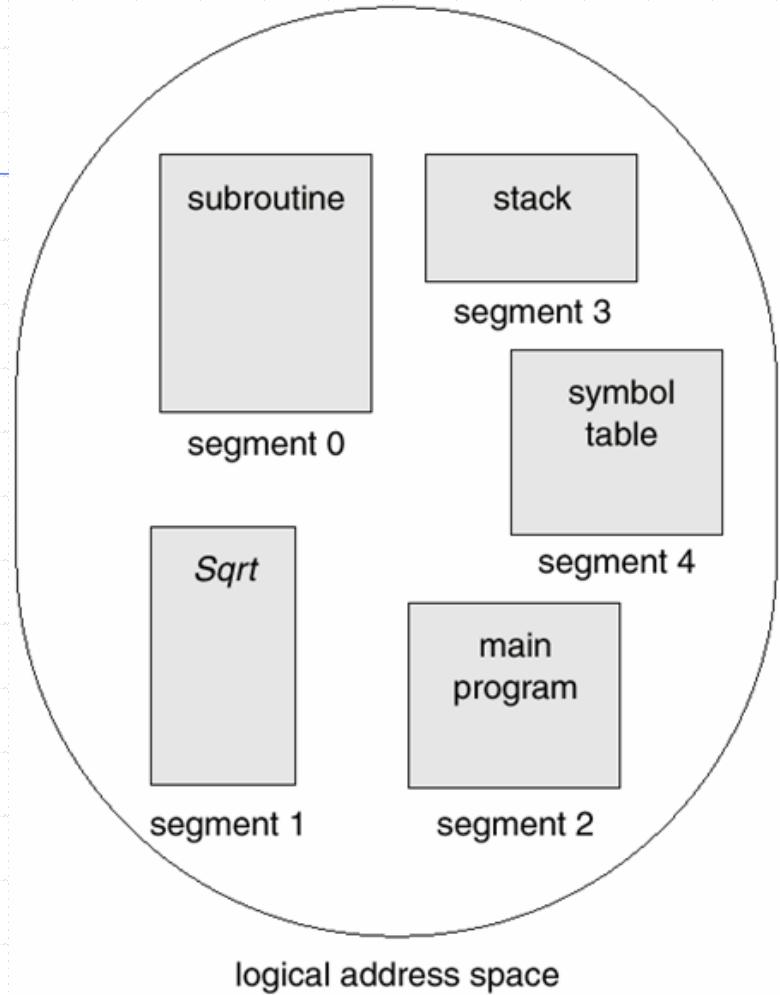
Példa: szegmensek Linux alatt

- Kevés szegmens, hogy hordozhatóbb legyen.
- 6 szegmens:
 - *Kernel kód*
 - *Kernel adat*
 - *User kód* (az összes user folyamat osztozik rajta)
 - *User adat* (ugyancsak osztott használat)
 - *Task-state* (folyamatok hardver kontextusai)
 - *LDT* (ritkán használt)

Szegmens szervezés védelme

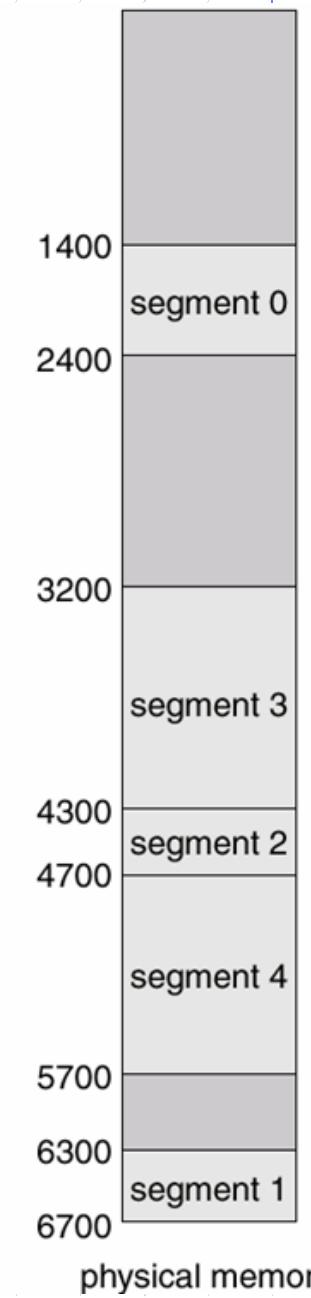
- A folyamat nem címezhet ki a saját szegmenséből
 - szegmensen belüli cím \leq szegmens mérete,
 - különben megszakítás (segment overflow fault).
 - A szegmenstábla tárolja a szegmens méretét (limit).
- Hozzáférés ellenőrzés
 - olvasási jog (szegmens területét olvashatja)
 - írási jog (szegmens területére írhat, az ott lévő értékeket módosíthatja)
 - végrehajtási jog (a szegmensben gépi utasítások vannak, azokat a folyamat végrehajthatja)
 - Jogosultság megsértése, megszakítást generál (segment protection fault).

Példa



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



Osztott szegmenshasználat

- Közös utasítások használata:
 - több folyamat azonos programot futtat
 - kevesebb memóriahasználat
 - lehet teljes program is, de rendszerkönyvtár is.
- Közös adatterület
- Folyamatok közötti kommunikáció.
- Megvalósítás:
 - A folyamatok szegmenstáblájában valamelyik szegmensnél azonos fizikai cím van.
 - A jogosultságok természetesen különbözőek, biztosítani kell a kölcsönös kizárást.

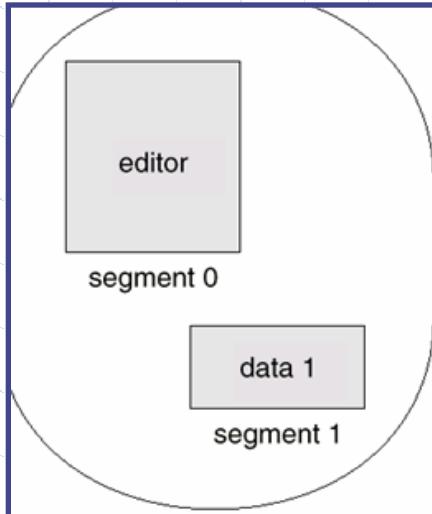
Példa

Szövegszerkesztő

- kódszegmens
- adatszegmens

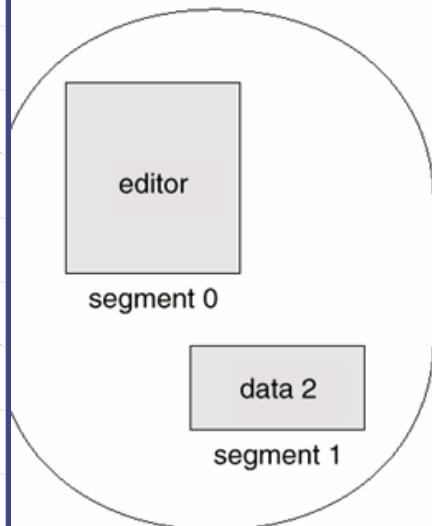
Két példány:

- kódszegmens
osztott használata



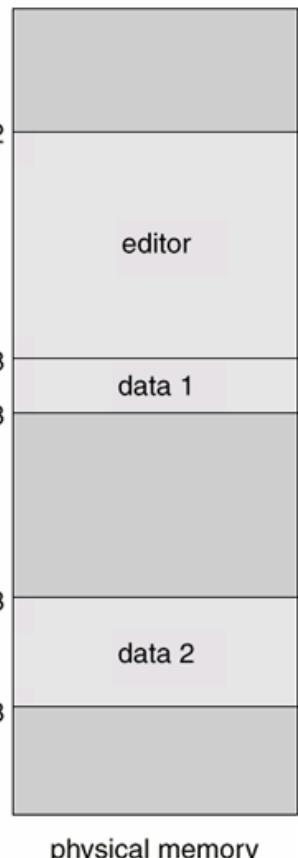
	limit	base
0	25286	43062
1	4425	68348

segment table
process P_1



	limit	base
0	25286	43062
1	8850	90003

segment table
process P_2



Címtranszformáció háttértáron lévő szegmens esetén

Szegmenstábla tartalmaz még:

- benntartózkodási bitet (residency bit).
 - Jelentése: a szegmens a memóriában van-e.
- Információt, hogy hol van a háttértáron
- Ha a benntartózkodási bit hamis:
 - hiányzó szegmens hiba (*missing segment fault*) lép fel,
 - amit az OS kezel le.
- Szegmenshasználat hátránya:
 - nagy külső tördelődés, mivel nem azonos méretűek a blokkok.

Szegmenstabla felépítése

Blokktábla/szegmenstabla

szegmensek	szegmens cím	szegmens hossz	RB	helye a háttértáron	R	W	X

Residency bit

Hozzáférési jogosultságok
(Read/Write/Execute)

6.5.2. Lapszervezés

- Azonos méretű blokkok (lap, page) használata
- Megszünteti a külső tördelődést.
- A belső tördelődés elkerülése miatt kis lapokat érdemes választani.
- A lapok mérete mindenig 2 hatvány.

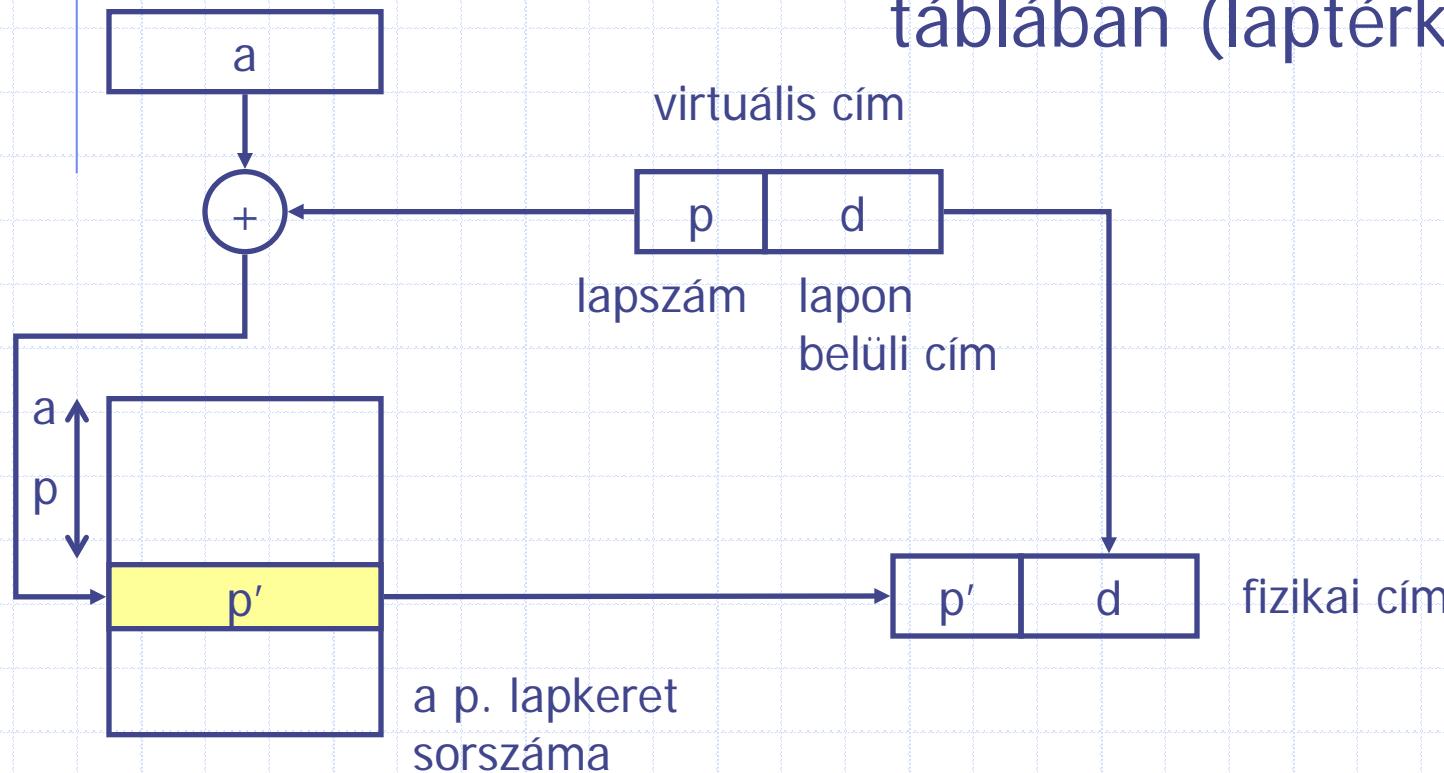
Címtranszformáció

- A 2 hatvány miatt az alsó biteken lehet tárolni az eltolást:
 - $\langle b, d \rangle$, ahol **b** most 2 hatványa:
 - Pl.: 1011011000000, az utolsó nullák helyére beírható a **d**
- Címtranszformáció módszerei:
 - Közvetlen leképzés
 - Asszociatív leképzés
 - Kombinált technikák

Közvetlen leképzés

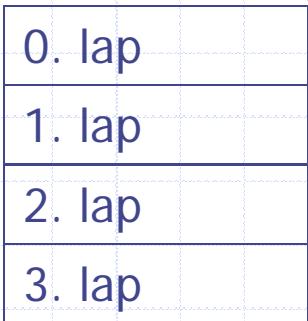
- Egyszintű laptábla:
A folyamathoz tartozó minden lap fizikai címe egy táblában (laptérkép) van.

laptábla kezdőcíme



Példa

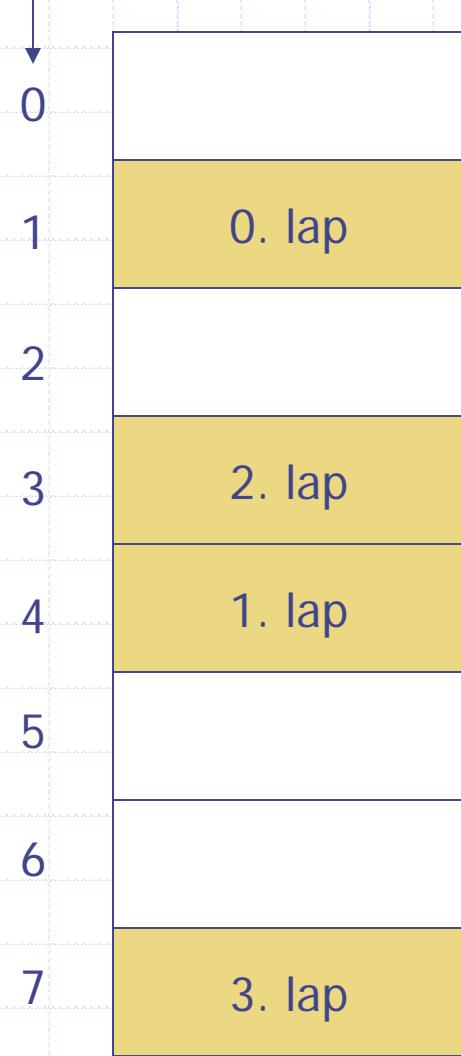
p' : lap-keretek (*nem kezdőcímek!*)



Logikai címtér

p	p'
0	1
1	4
2	3
3	7

laptábla



fizikai memória

Megjegyzés: a laptáblának ilyen kialakítás esetén természetesen nem kell tárolnia p értékeit, hiszen p maga a táblán belüli cím.

Példa

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

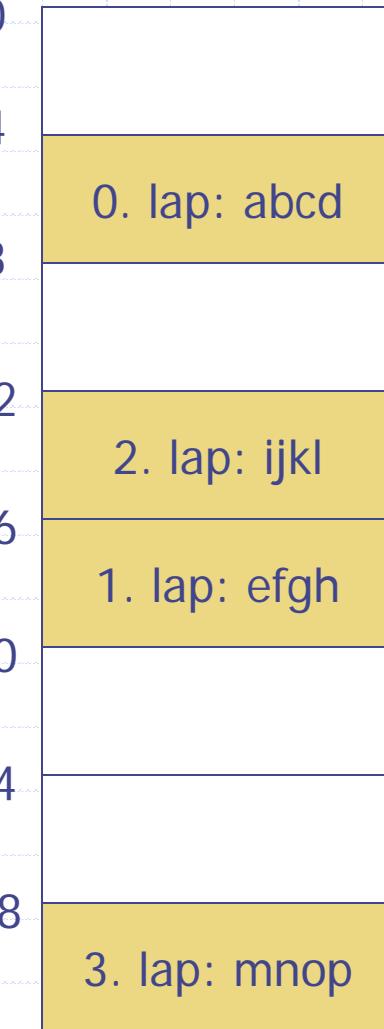
Logikai címtér

Lapok fizikai kezdőcímei → 0
(p'^*N)

} lapmáret $N = 4$

p	p'
0	1
1	4
2	3
3	7

laptábla



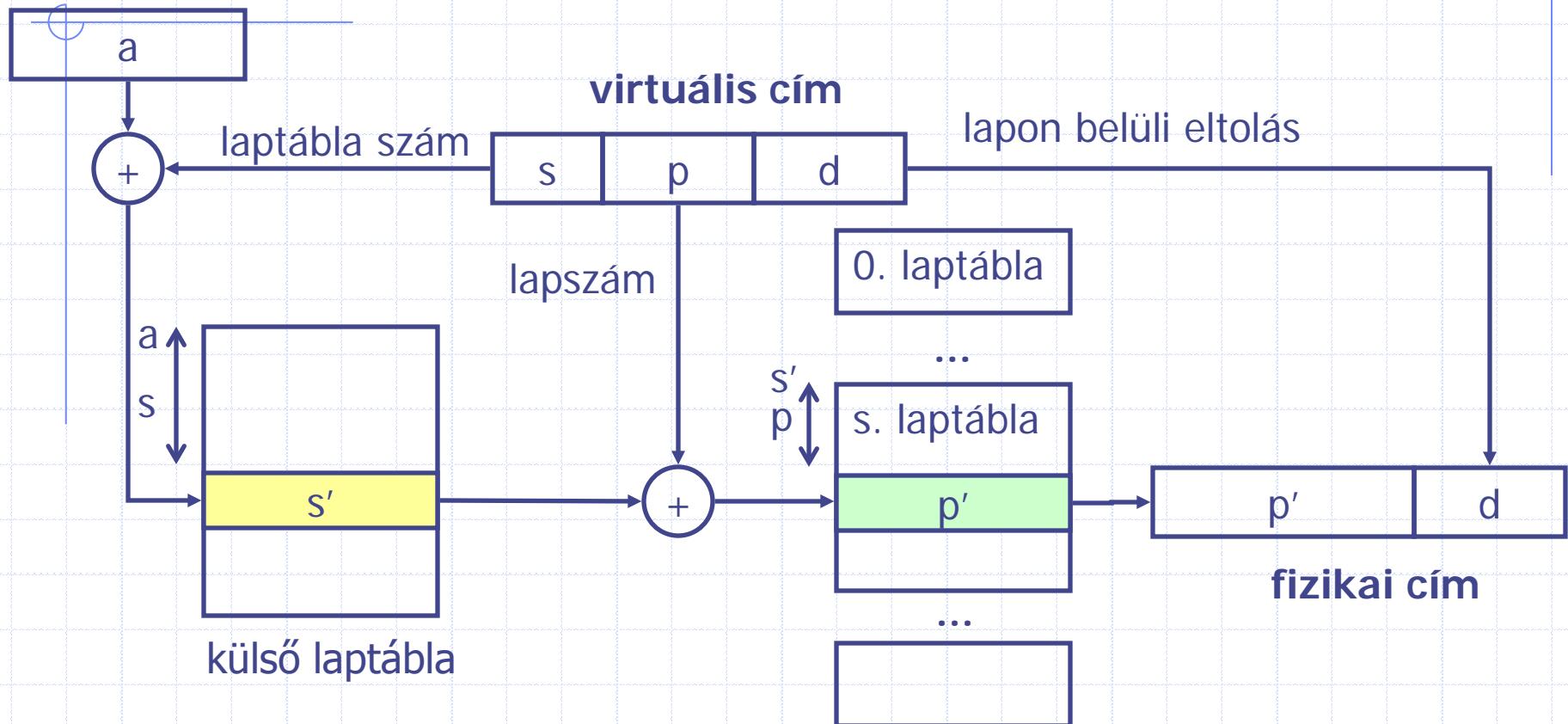
fizikai memória

Egyszintű laptábla problémái

- Laptábla mérete nagy lehet, mivel a lapok száma sok.
- Nehéz gyors elérésű tárban tartani.
- Pl.:
 - 32 bites címtér (2^{32})
 - 4 Kbites lapok (2^{12})
 - 2^{20} ($\approx 10^6$) db bejegyzés
- Megoldás: laptábla tördelése → többszintű laptábla.
- Felfogható a laptábla lapozásának is.

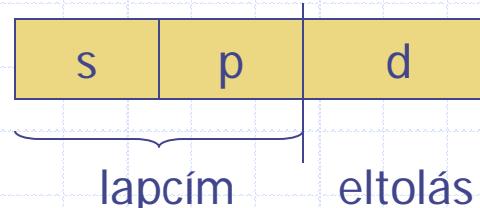
Többszintű laptábla

külső laptábla kezdőcíme

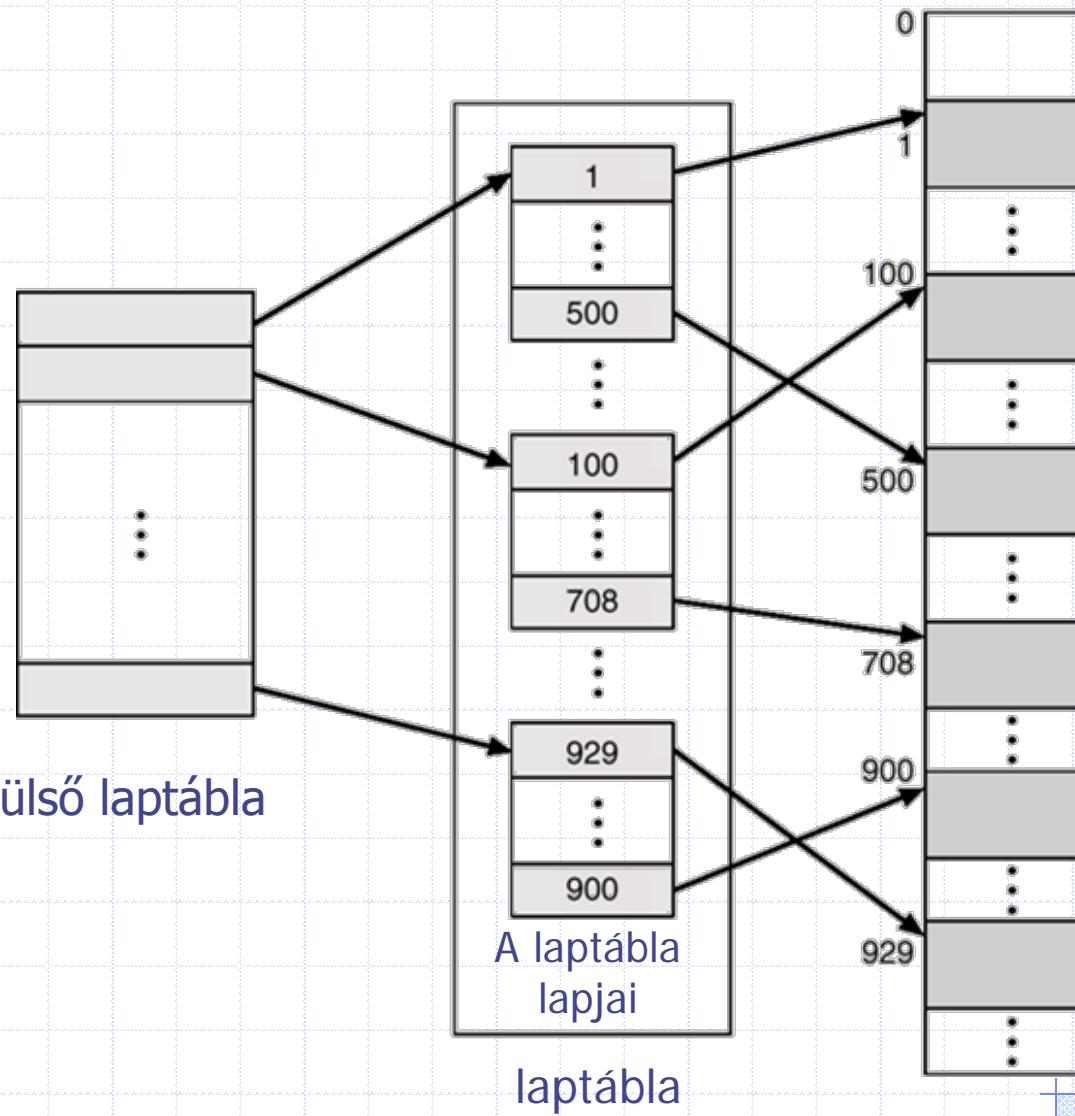


Tipikus példa (32 bites architektúrán):

- 10 bites laptábla-szám (s)
- 10 bites lapszám (p)
- 12 bites eltolás (d)



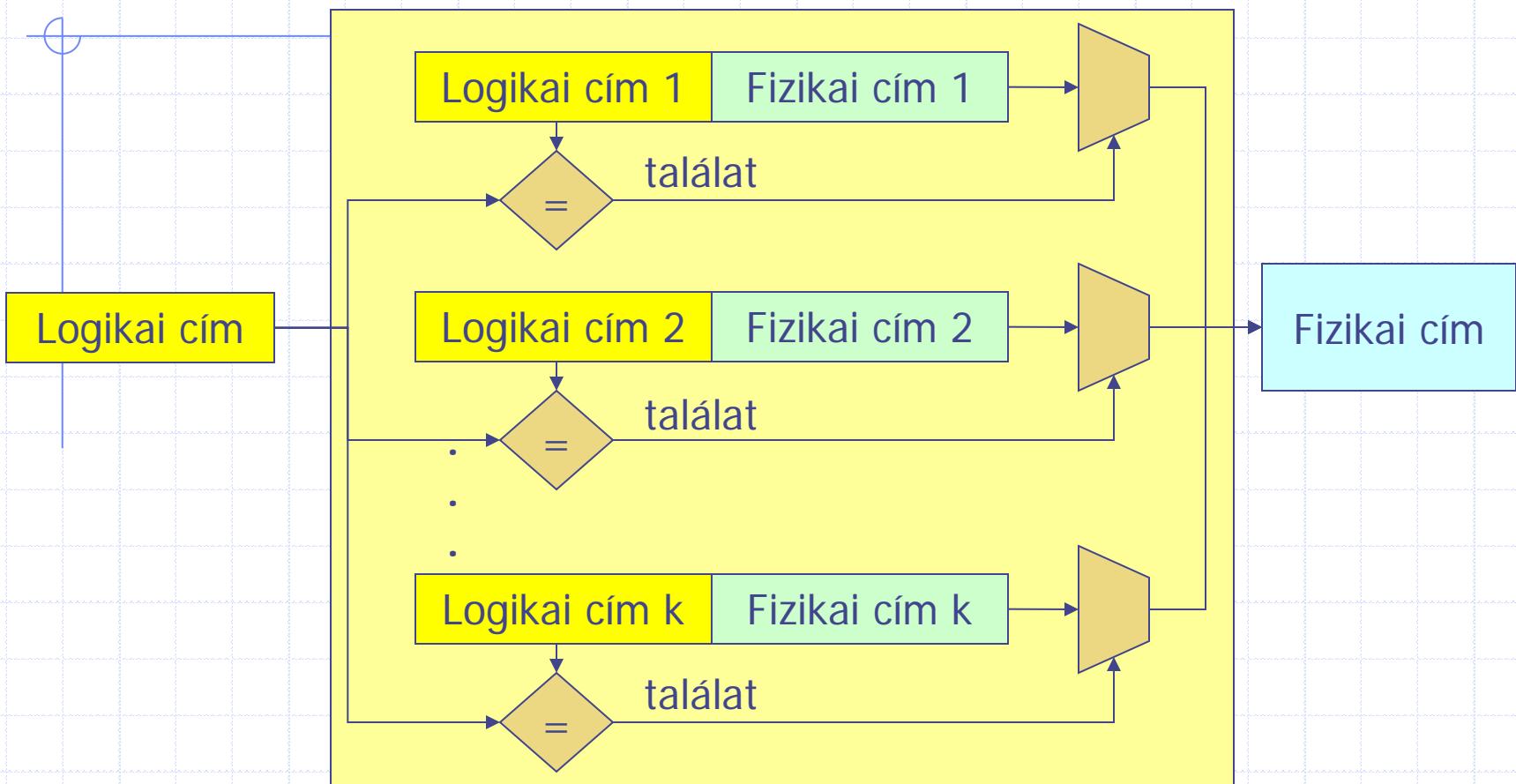
Többszintű laptábla példa



Asszociatív leképezés

- Speciális gyors elérésű tár (asszociatív tár) segíti a címzést
 - translation look-aside buffer (TLB)
- A laptábla gyorsító tárban a várhatóan gyakran használt lapok címét tároljuk.
- A tár mérete itt sem elég nagy.
- A gyakorlatban a kombinált technikák (laptábla + gyorsítótár) használhatók.

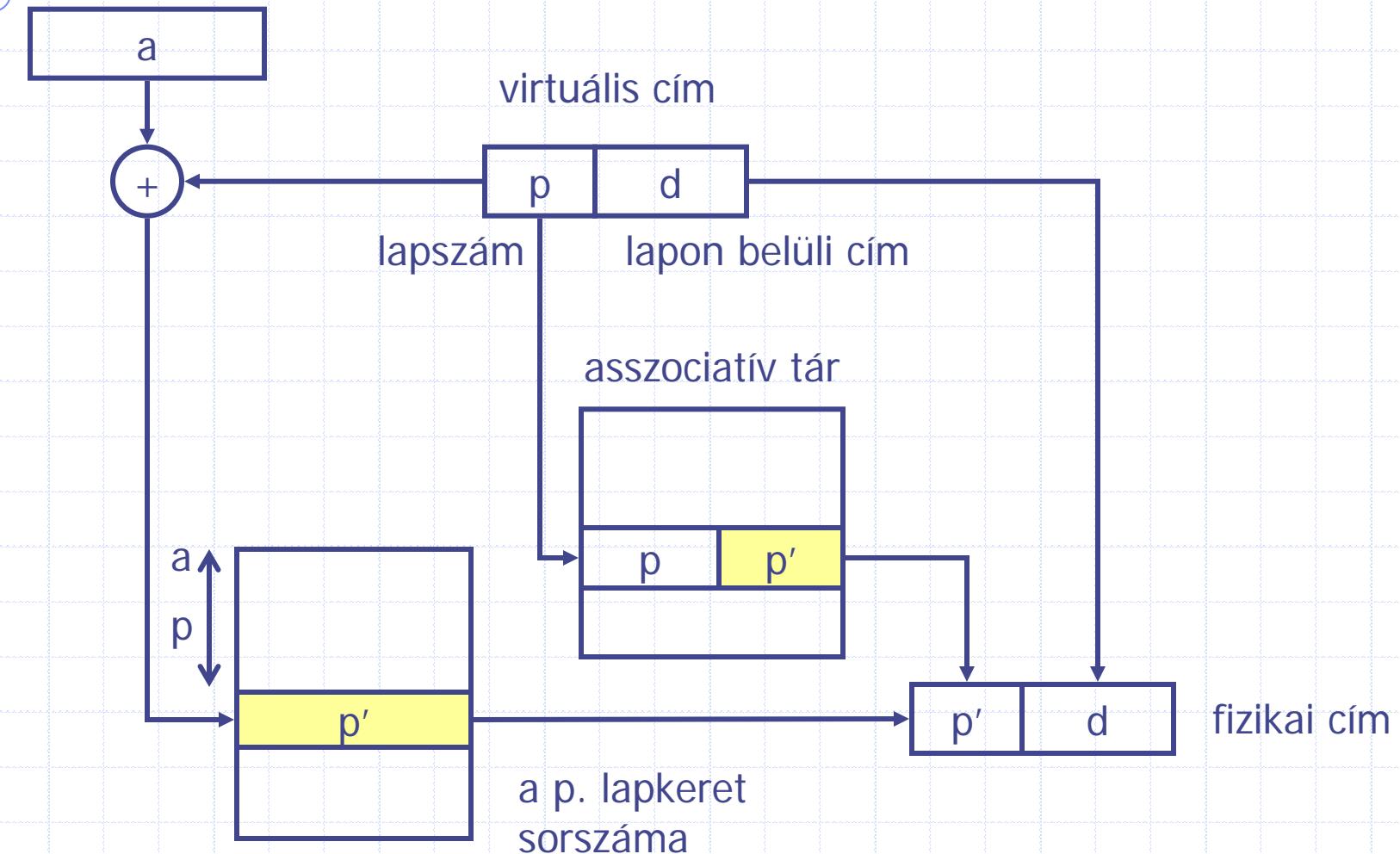
Az asszociatív tár működése



- Keresés: párhuzamosan az összes tárolt logikai cím alapján.
- Találat esetén a találatnak megfelelő fizikai cím kerül a kimenetre.
- Drága → kis kapacitású

Kombinált technika

laptábla kezdőcíme



Kombinált technikák

- A fizikai lapcím keresése egyszerre kezdődik minden asszociatív tárban, minden direkt laptáblában.
- Ha az asszociatív tárban van találat, akkor a direkt keresés leáll.
- Az asszociatív tárban lévő lapokat frissíteni kell, környezetváltás esetén az asszociatív laptáblát is cserélni kell.
- Egy adott időszak alatt csak a teljes címtartomány kis része van kihasználva, így a találati arány elég magas lehet (80-99%).

Túlcímzés elleni védelem

- Lapon belüli túlcímzés ellen nem kell védeni, hiszen minden lapon belül kiadható cím megfelelő
 - kivétel: utolsó lap...
- A lapok érvényességét egy bit jelzi (lásd a következő példát).
- Címtranszformáció a háttértáron lévő lap esetén:
 - Az érvényesség bit jelzi, hogy a memóriában van-e a lap. Ha nincs (invalid bit), akkor lehet, hogy a háttértáron van. Utóbbi esetben a tábla a háttértáron való elhelyezkedés információját tárolja.

Példa

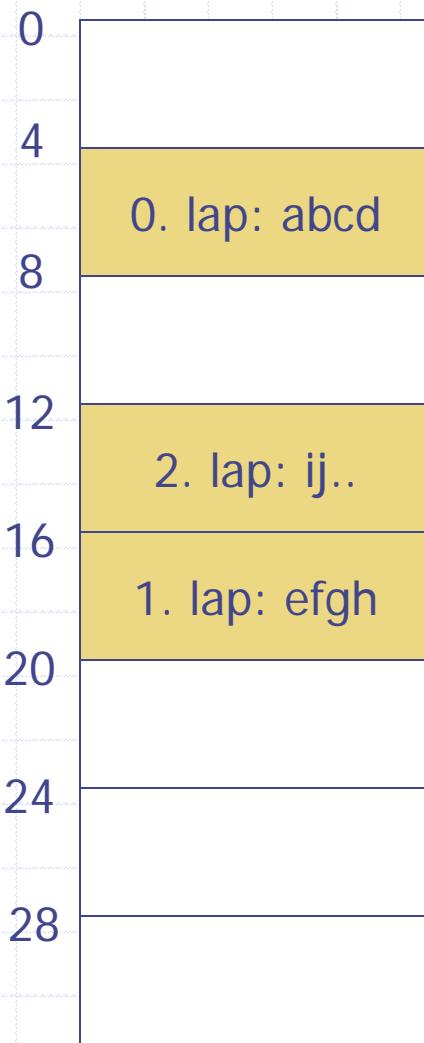
0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	
11	
12	
13	
14	
15	

Logikai címtér

valid/invalid bit

0	1	v
1	4	v
2	3	v
3	0	i

laptábla



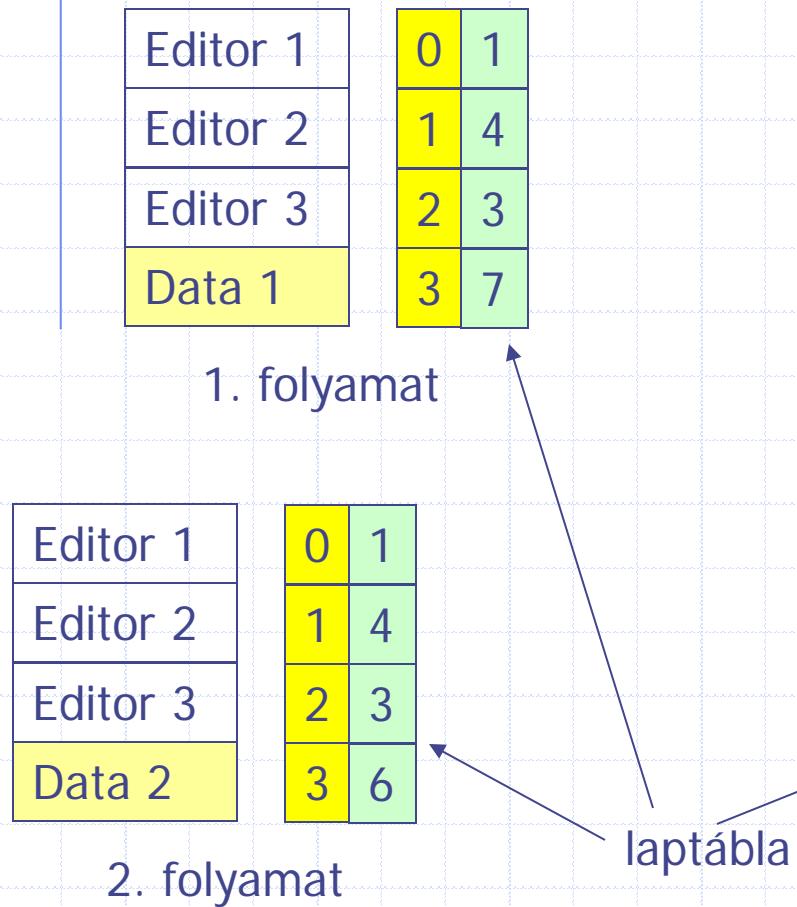
fizikai memória

Osztott laphasználat

- Hasonló az osztott szegmenshasználathoz, több folyamat lapterkép táblája azonos fizikai címekre hivatkozhat.
- Pl.:
 - közösen használt kód. Ilyenkor az OS gondoskodik a védelemről:
 - Szövegszerkesztő
 - 3 lap kód
 - 1 lap adat
 - Ha 20 példány fut, akkor a memóriaigény
 - $4 \times 20 = 80$ lap (nem osztott laphasználat)
 - $1 \times 20 + 3 = 23$ lap (osztott laphasználat)

Példa

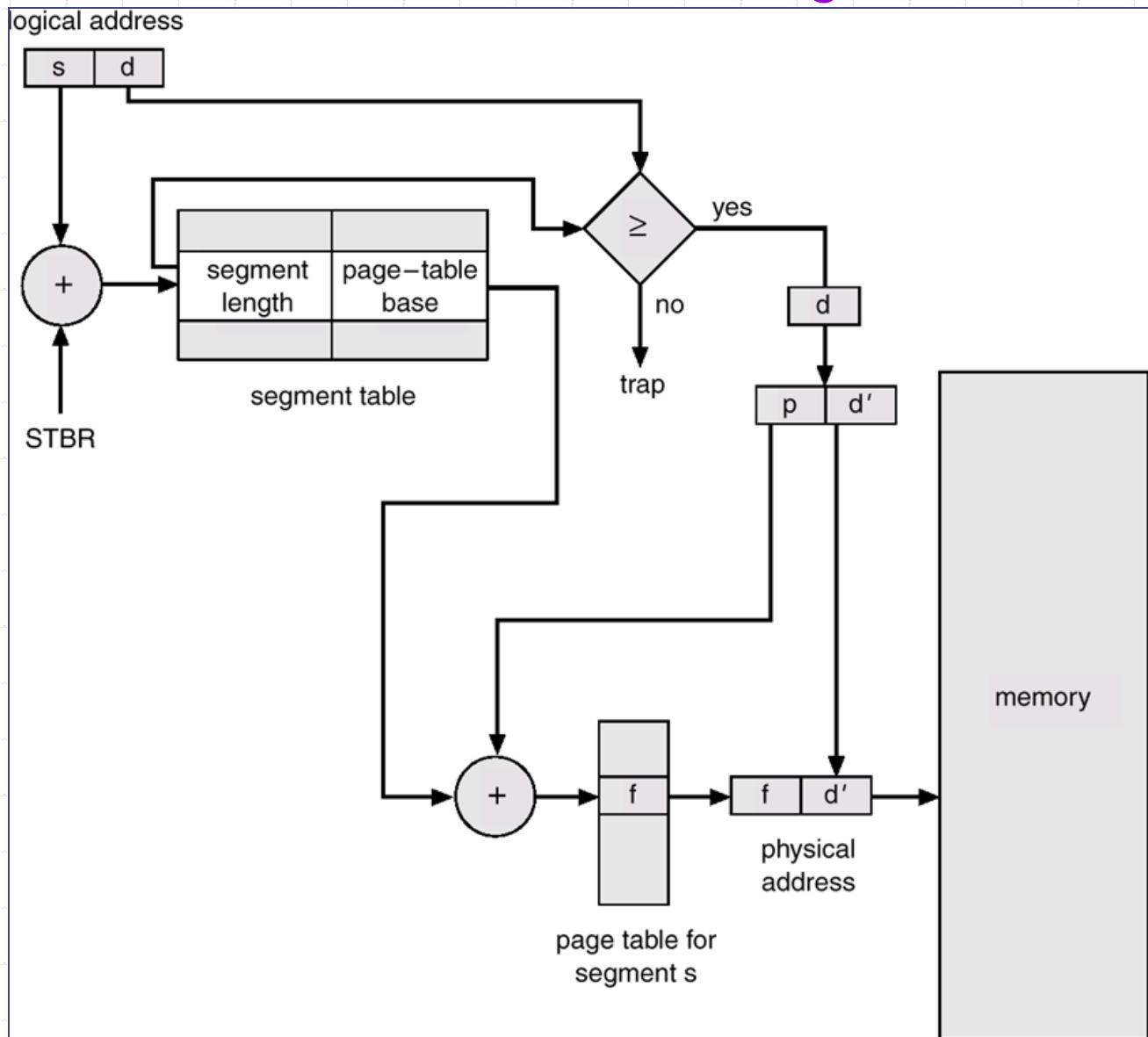
- Editor: 3 lap kód, 1 lap adat, 3 példány



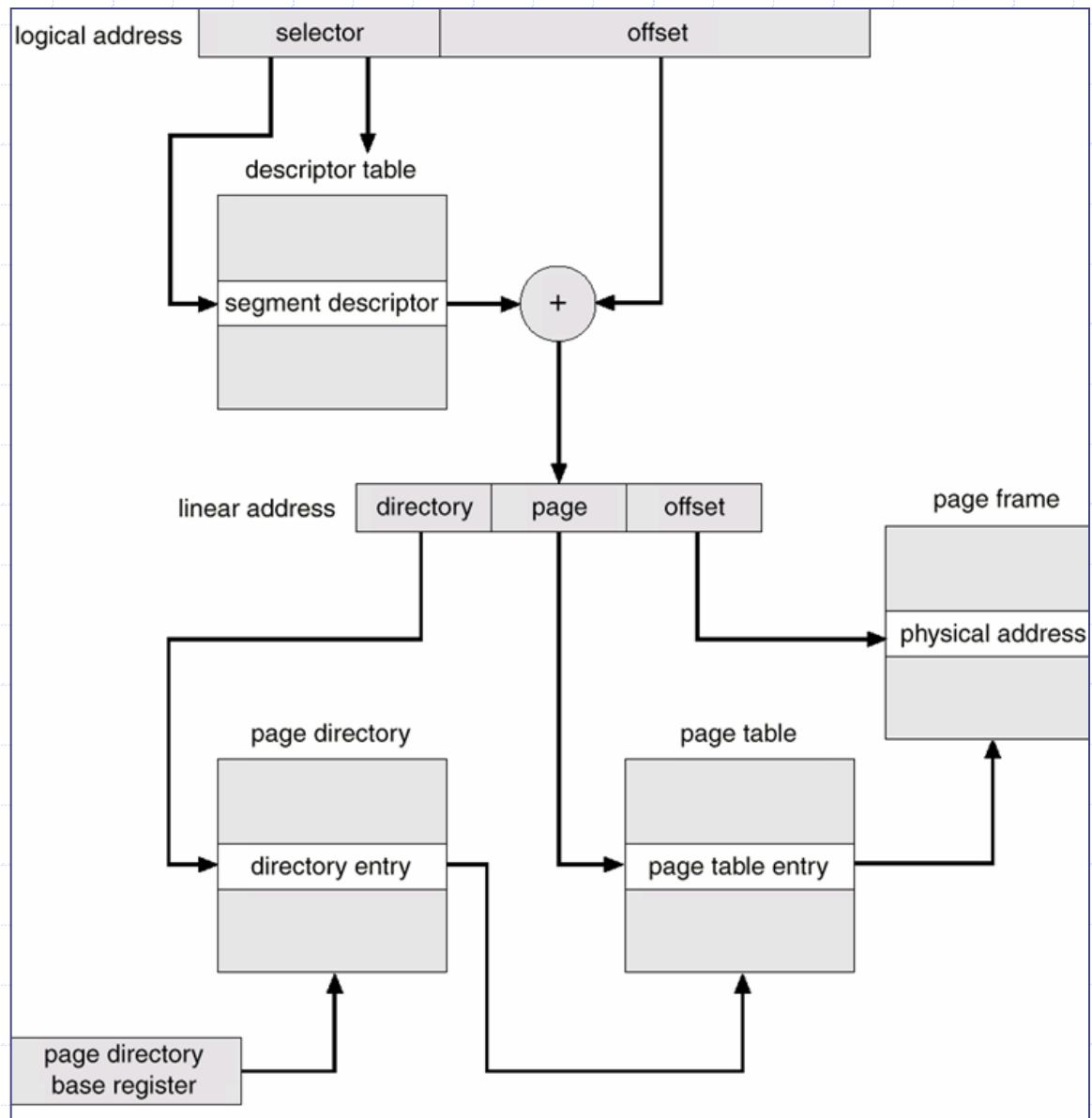
6.5.3. Kombinált szegmens- és lapszervezés

- Egyesíti a két technika előnyeit.
 - Lap szervezés: nincs külső tördelődés, nem kell a teljes szegmensnek a tárban lennie.
 - Szegmens szervezés: tükrözi a folyamat logikai társzerkezetét, hozzáférési jogosultság megoldható.
- Címtranszformáció: lényegében egy kétszintű táblakezelés.
 - Első szint: laptábla címeket tartalmazó szegmenstábla
 - Második szint: szegmensenként egy-egy fizikai lapcímeket tartalmazó laptábla.
- A cím három részre tagozódik (szegmenscím, lapcím, lapon belüli eltolás).
- Hozzáférési jogok ellenőrzése a szegmens szervezésének megfelelően történik.
- Osztott tárhazsnálat a szegmens szervezésének megfelelően történik.

Példa: MULTIX operációs rendszer címtranszformációja

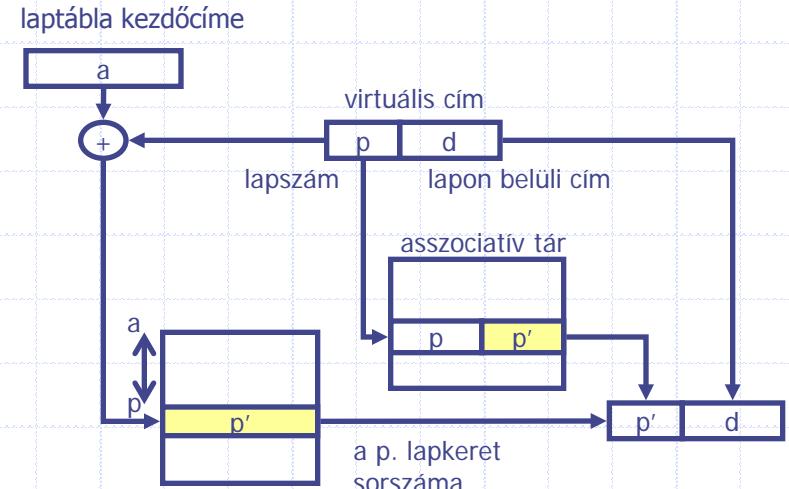


Példa: Intel 386 címtranszformációja



Példa: Átlagos hozzáférési idő

- Kombinált (laptáblát és asszociációs memóriát is tartalmazó) rendszerben a memória-hozzáférés ideje 100ns. Az asszociációs memória 20ns alatt érhető el. A találati arány 90%. Mekkora az átlagos elérési idő?



Válasz:

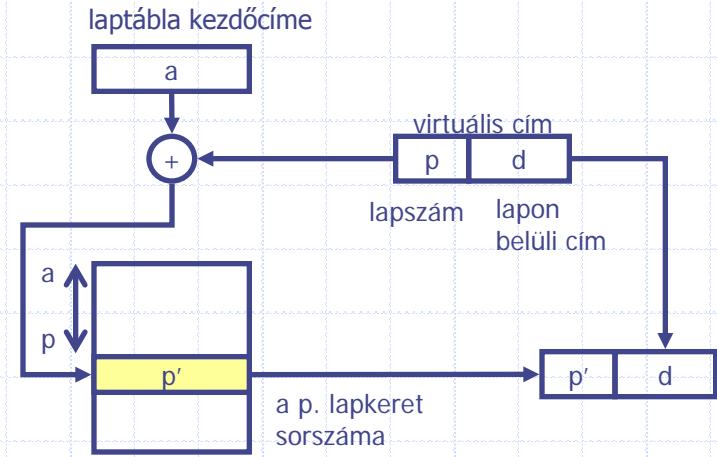
Fizikai cím elérése asszoc. memóriával: $20\text{ns} + 100\text{ns} = 120\text{ns}$

Fizikai cím elérése asszoc. memória nélkül: $100\text{ns} + 100\text{ns} = 200\text{ns}$

Átlagos elérési idő: $0.9 * 120\text{ns} + 0.1 * 200\text{ns} = 128\text{ns}$

Példa: Optimális lapméret

- Egy rendszerben az átlagos folyamat mérete $s = 1\text{MB}$. Egy laptábla bejegyzés mérete $e = 8\text{ byte}$. Mekkora a lapok ideális mérete (p)?



Válasz:

Az „elpazarolt” memória mérete (K):

- Laptábla bejegyzések: s/p^*e
- Az utolsó lap átlag fele üres: $p/2$

$$K = s/p^*e + p/2$$

$$\text{Optimum: } dK/dp = 0 = -se/p^2 + 1/2 \rightarrow p = \sqrt{2se} = 4\text{kB}$$