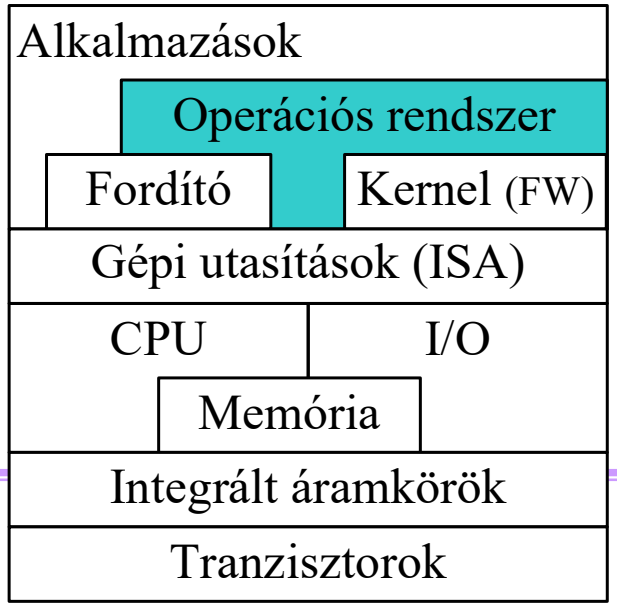
INFORMATIKA I.

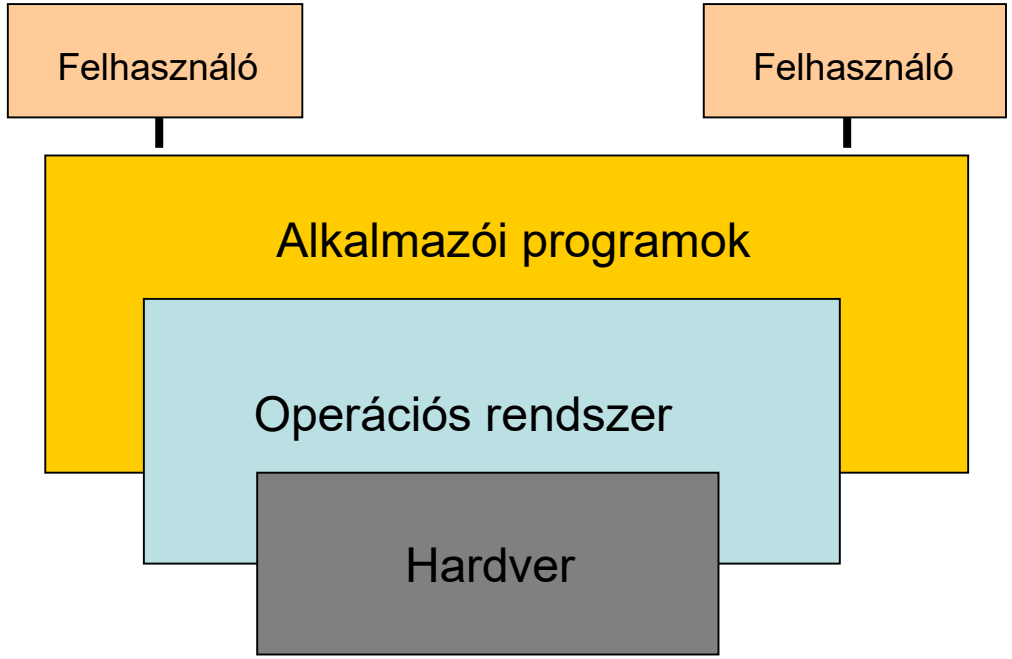
BMEVIIIAB04

Operációs rendszerek



Operációs rendszerek

Számítógép rendszer



Operációs rendszer
kapcsolat a hardver és a felhasználó között

Cél
Hatékony hardver kihasználás
A felhasználó kényelme

Multiprogramozás – Megoldandó feladatok

Melyik JOB fusson

CPU ütemezés

Egyszerre több program lehet a memóriában

Memória gazdálkodás

A periféria használatot koordinálni kell

Periféria kezelés, ütemezés

Együttfutási problémák kezelése

Szinkronizálás

Üközések, patthelyzet

Holtpont kezelés

Védelmi kérdések

JOB ↔ JOB
JOB → operációs rendszer

Memória gazdálkodás

Kérdések

Hogyan helyezkednek el a programok a memóriában

Hogyan használják a folyamatok a memóriát

Hogyan támogatja ezt az operációs rendszer

Memória (hardver)

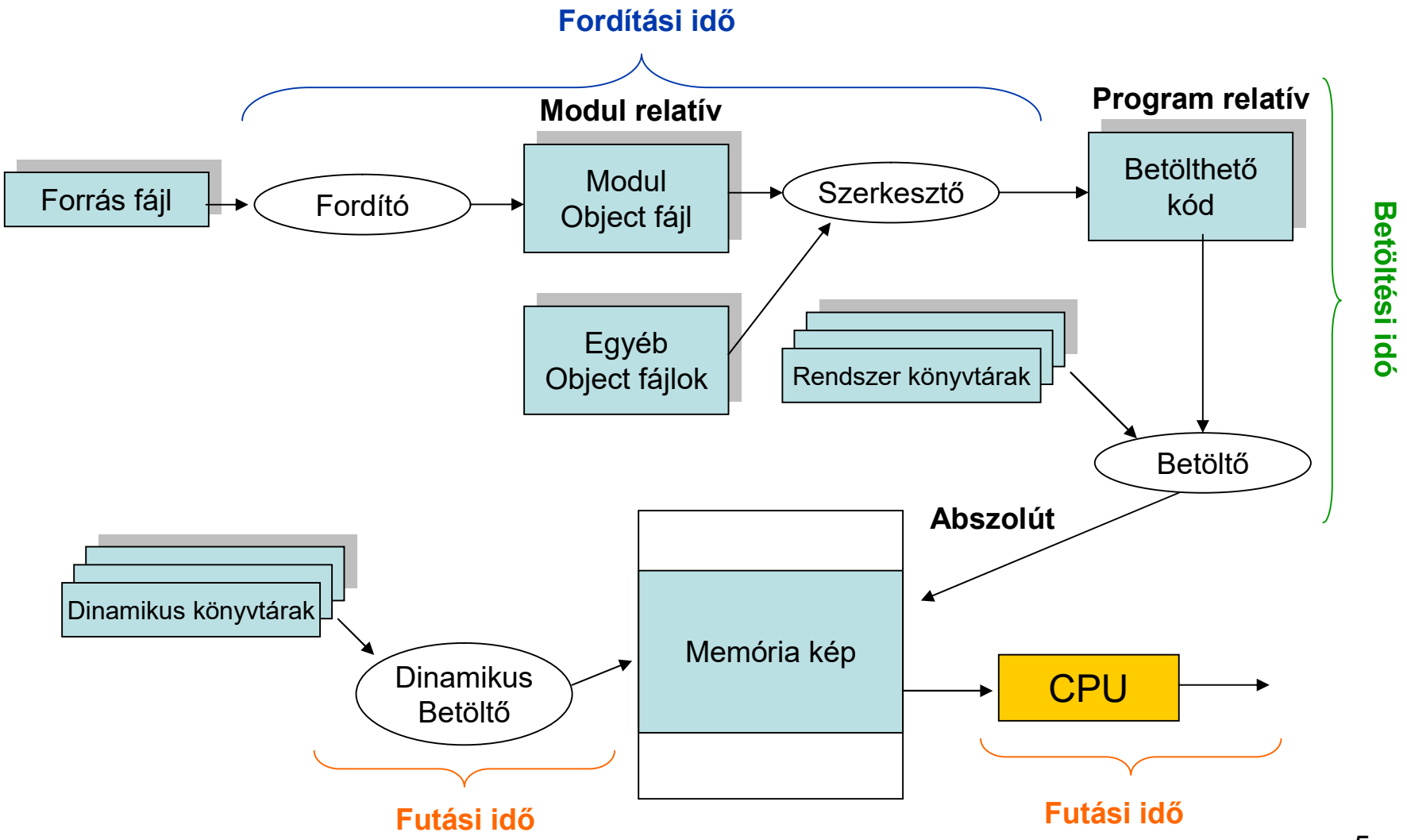
Lineárisan címezhető byte tömb

Elérése több ciklust vehet igénybe

Cache alkalmazásával a hozzáférés gyorsítható

A helyes működés érdekében védelmi mechanizmusok kellenek

Címek kialakulása a programban



Címek kialakulása a programban

Forrás → szimbolikus címek

↳ Fordítás → relokálható címek : modul relatív

↳ Szerkesztés → relokálható címek : program relatív

↳ Betöltés → abszolút címek

Logikai cím: A CPU által kiadott cím
Fizikai cím: A memória eléréséhez használt cím

Logikai címtartomány $\leq$ Fizikai címtartomány

Logikai cím → fizikai cím : Mapping

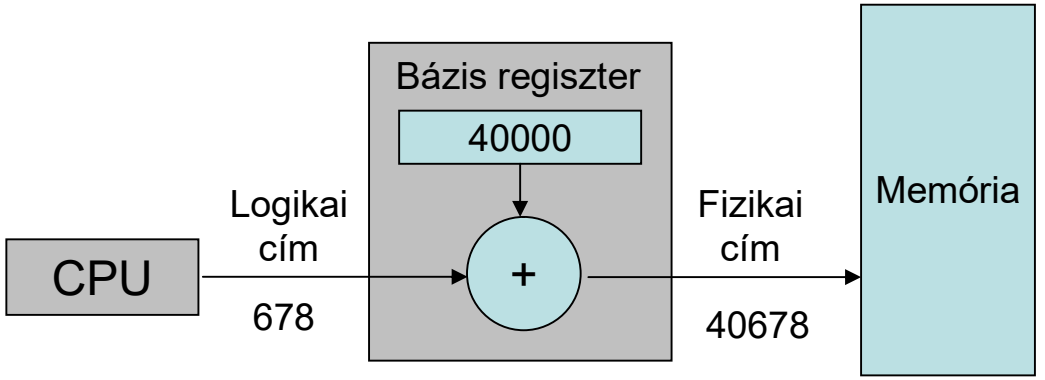
Relokálás (áthelyezés)

Statikus

- Előkészítési fázisban
- Legkésőbb a betöltéskor
- Szoftver

Dinamikus

- Végrehajtáskor
- Hardver: bázisregiszter

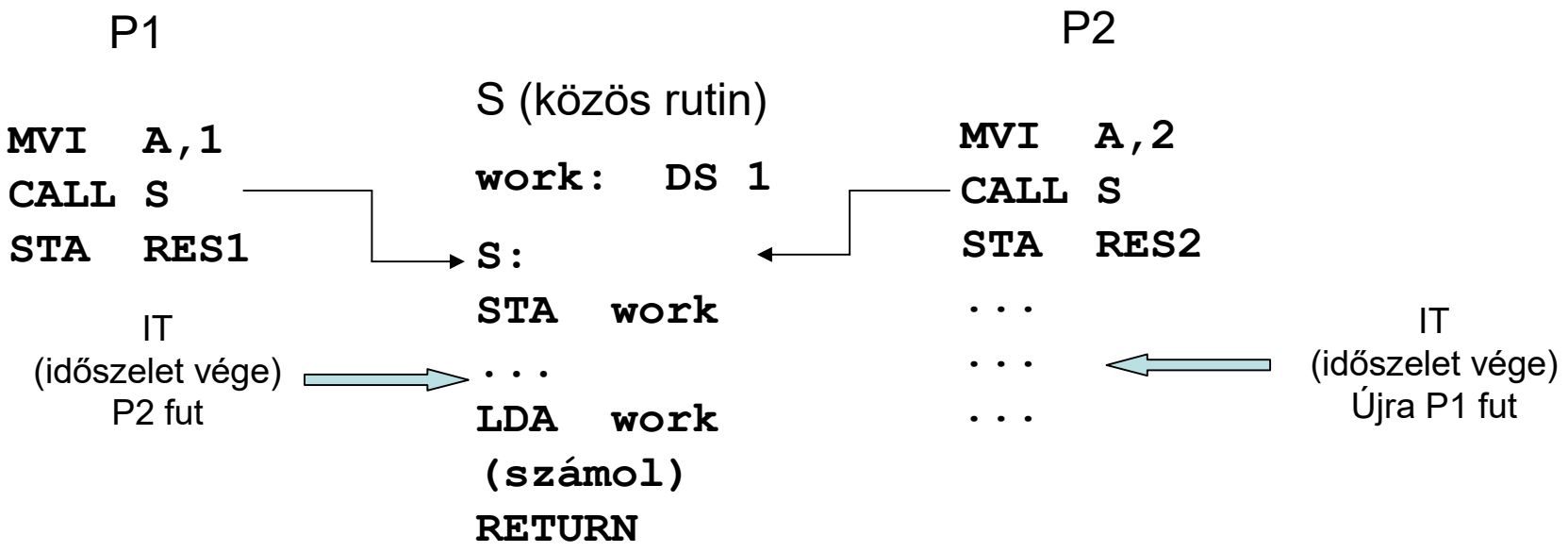


Áthelyezhető program → tetszőleges címre betölthető

Csak relatív hivatkozásokat tartalmazhat
Kód: relatív JMP, CALL
Adat: PC relatív adatcímzés?

Újrahívhatóság

Egy szubrutin újrahívható, ha visszatérése előtt újrahívva minden hívó számára helyes eredményt ad

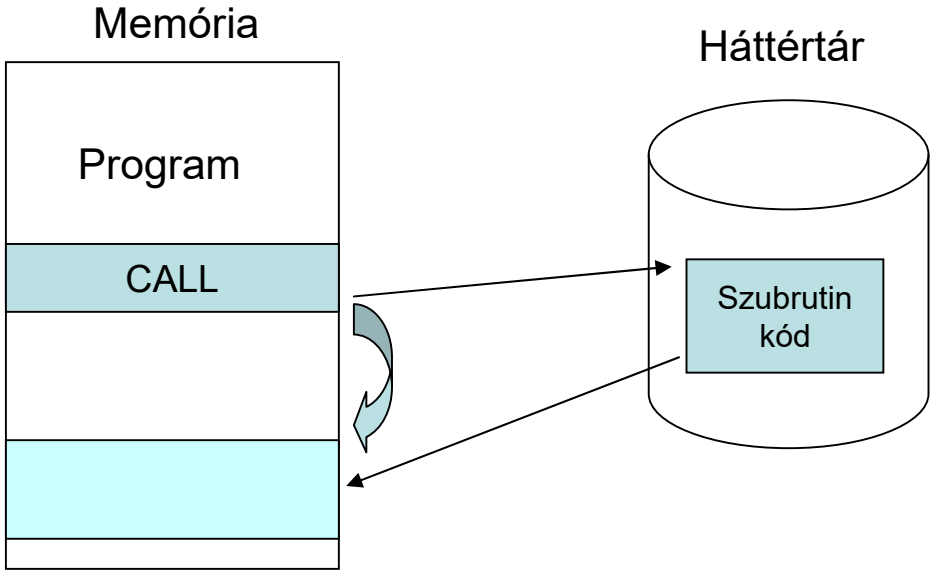


P1 nem a saját értékéből számolt eredményt kapja vissza → Nem újrahívható
Munkarekesz a hívó területén
→ S a hívó területére tud címezni

- Bázisrelatív címzés
- A munkarekesz a stacken van (minden folyamatnak saját stack-je van)

Késleltetett betöltés

Indításkor csak a program egy része töltődik be
Dinamikus betöltés
a szubrutin csak a meghívásakor kerül be a memóriába
→ áthelyezhető kód kell



Dinamikus könyvtárbetöltés

A rendszerkönyvtárak helyett csak egy csonk (stub) van a programban

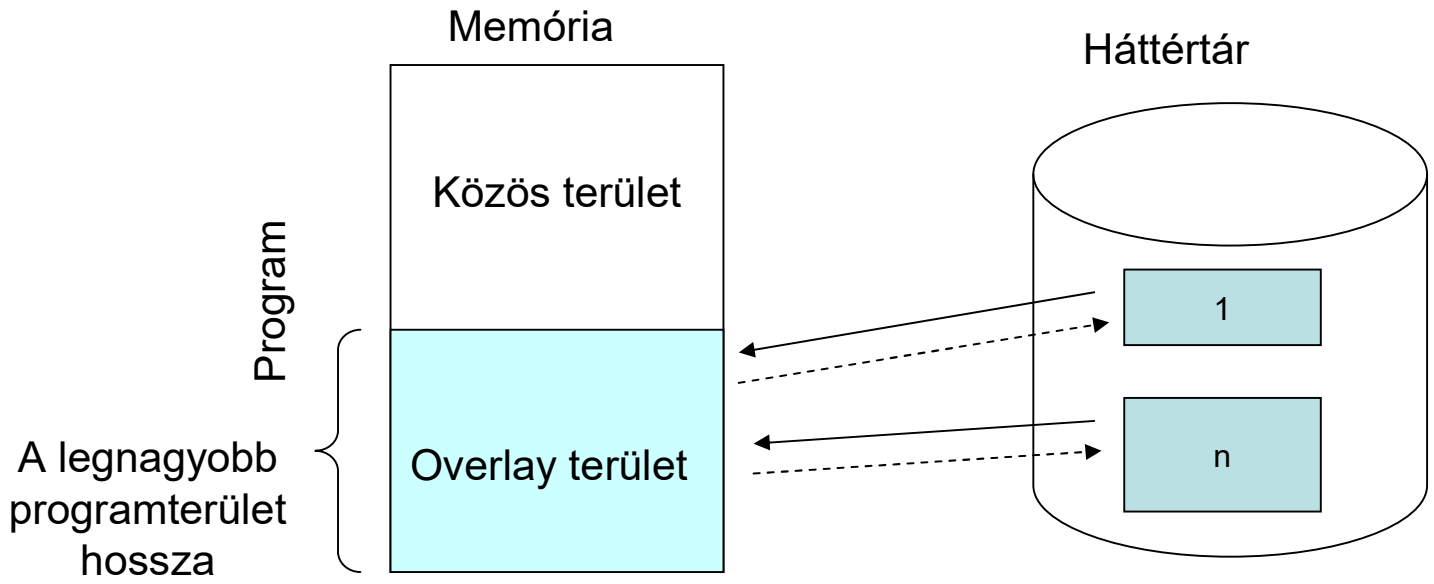
Az első hívásnál a rendszer betölti a könyvtárat
A további hívásoknál már nincs időveszteség

Lehetőség van arra, hogy több folyamat is használja ugyan azt a könyvtárat

→ a könyvtári szubrutinoknak újrahívhatónak kell lenniük

Overlay technika (átfedő programrészek)

A program egy részére folyamatosan szükség van
más részekre egyszerre soha nincs szükség



A felhasználónak kell szervezni, támogatás csak a betöltéshez

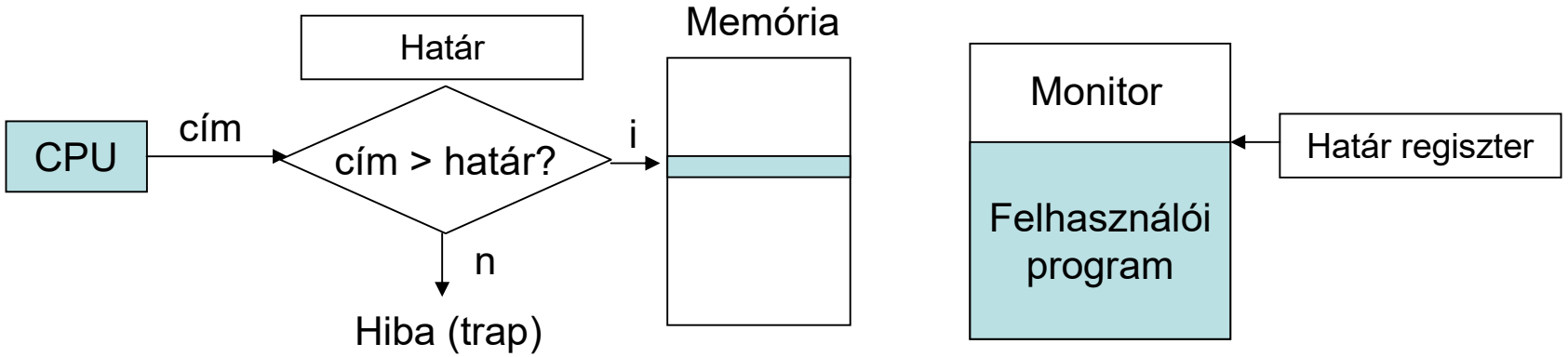
Programok elhelyezése

Csupasz gép

A teljes memóriával a felhasználó gazdálkodik
Nincs kötöttség – nincs szolgáltatás

Rezidens monitor

Védelem: futási időben



Védelmi regiszter:
fix cím – rugalmatlan
regiszter – átírás ellen védelem kell

Relokálás: statikus, betöltéskor

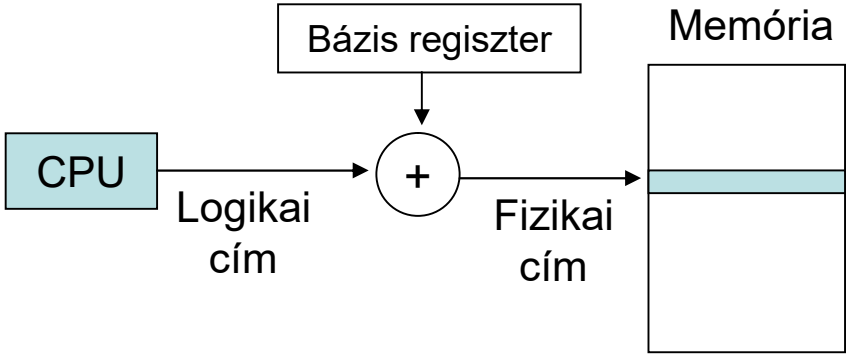
Programok elhelyezése

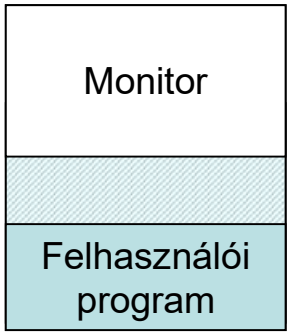
Tranziens monitor

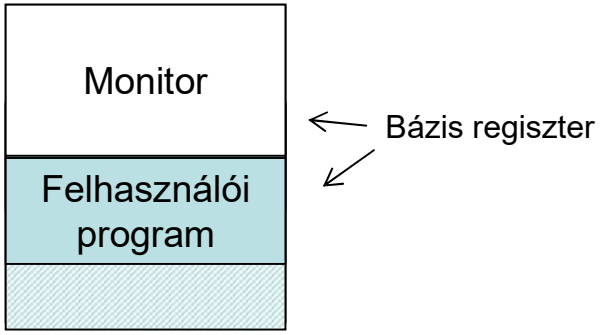
Működés közben a monitor mérete változik

- A felhasználói program a memória végére kerül
  Statikus relokálás

- Dinamikus relokálás
  Hardver címmódosítás







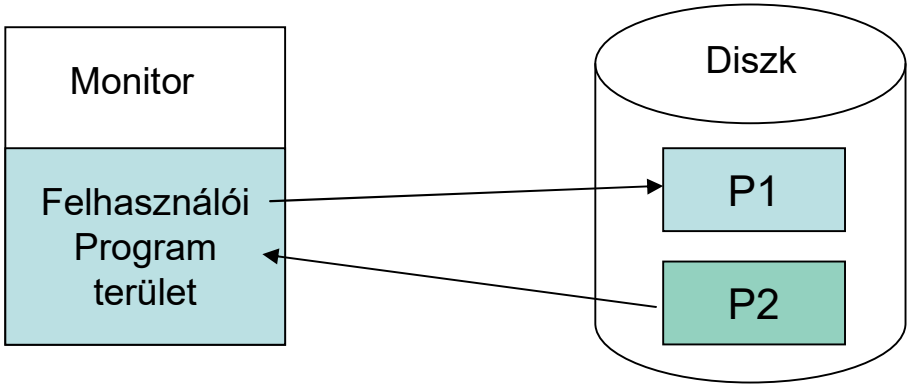
Ha a monitor növekszik, a programot el kell mozdítani
System módban a teljes memória elérhető

Programok elhelyezése – Multiprogramozás

Egy program akkor tud futni, ha teljes egészében bent van a memóriában

Egypartíciós rendszer – Swapping

Gyors háttértár használata



Háttértár:
- Hozzáférési idő: ~ 10 ms
- Adatátviteli idő: ~ 100 ns/byte

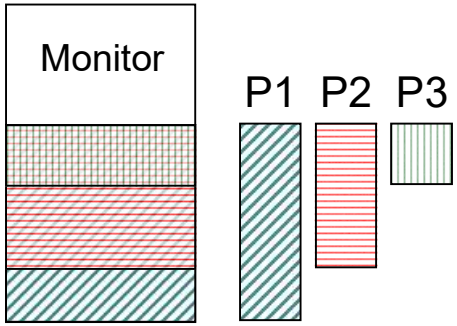
Gyorsítás

Háttértár paraméterek javítása

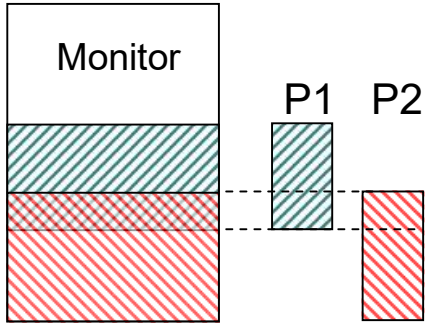
Programok elhelyezése

Gyorsítás

Hagymahéj algoritmus

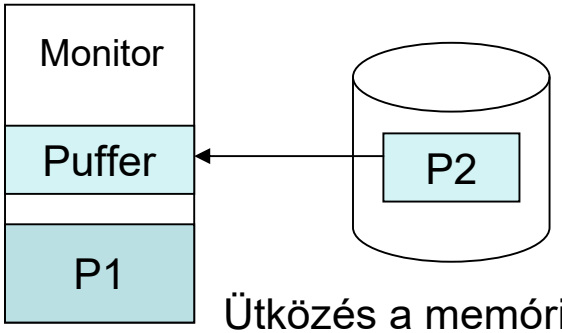


Minimális átlapolás



Ami nem fedi át egymást, azt nem másoljuk

Swapping átlapolása

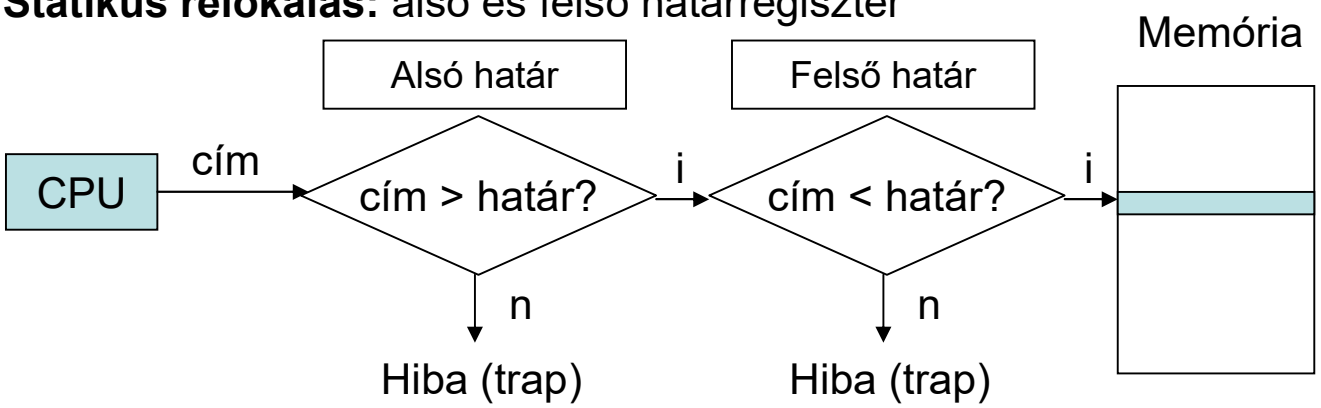


Ütközés a memória használaton
I/O ↔ CPU

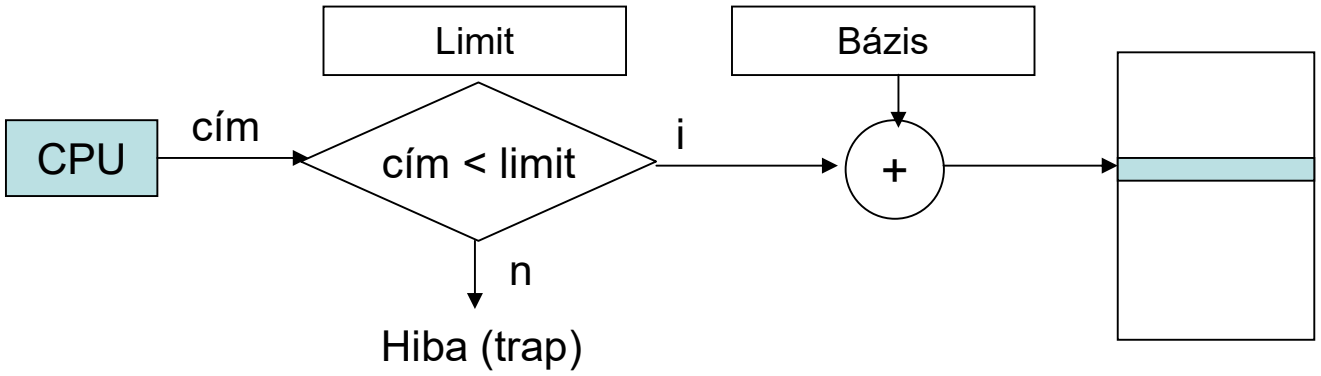
Programok elhelyezése - Multiprogramozás

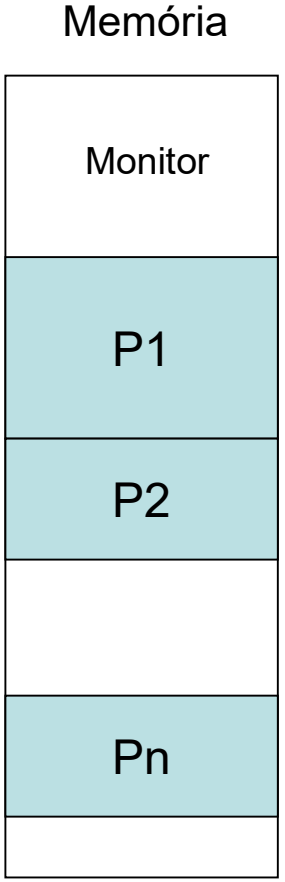
Többpartíciós rendszer

Statikus relokálás: alsó és felső határregiszter



Dinamikus relokálás: limit és bázis regiszter





Programok elhelyezése

Többpartíciós rendszer - Rögzített partíciók

Partíciónként csak egy folyamat
→ Multiprogramozhatóság foka = partíciók száma

Mekkora legyen egy partíció mérete?

- Külső fragmentáció
  van szabad partíció de a mérete nem elegendő a folyamat számára

- Belső fragmentáció
  Kihasználatlan területek maradnak a partícióban

Programok elhelyezése

Többpartíciós rendszer – Rugalmas partíciók

A memóriát a folyamat igénye szerint osztjuk
→ dinamikus tárkiosztás

Tárhelyet kell keresni
A kilépő folyamat területét fel kell szabadítani

Töredezettség (de belső fragmentáció nincs)

Töredezettség megszüntetése → dinamikus relokálás szükséges
periodikusan
eseményhez rögzítetten

Programok elhelyezése

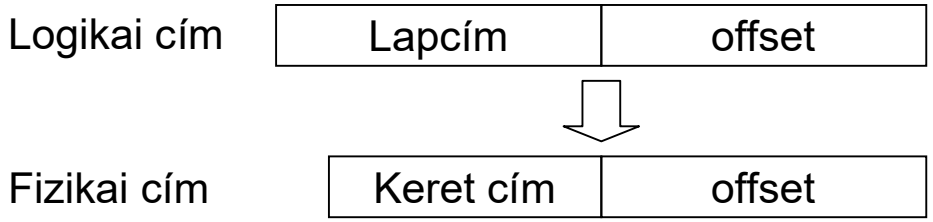
Többpartíciós rendszer – Rugalmas partíciók

Szabad tárhely keresése

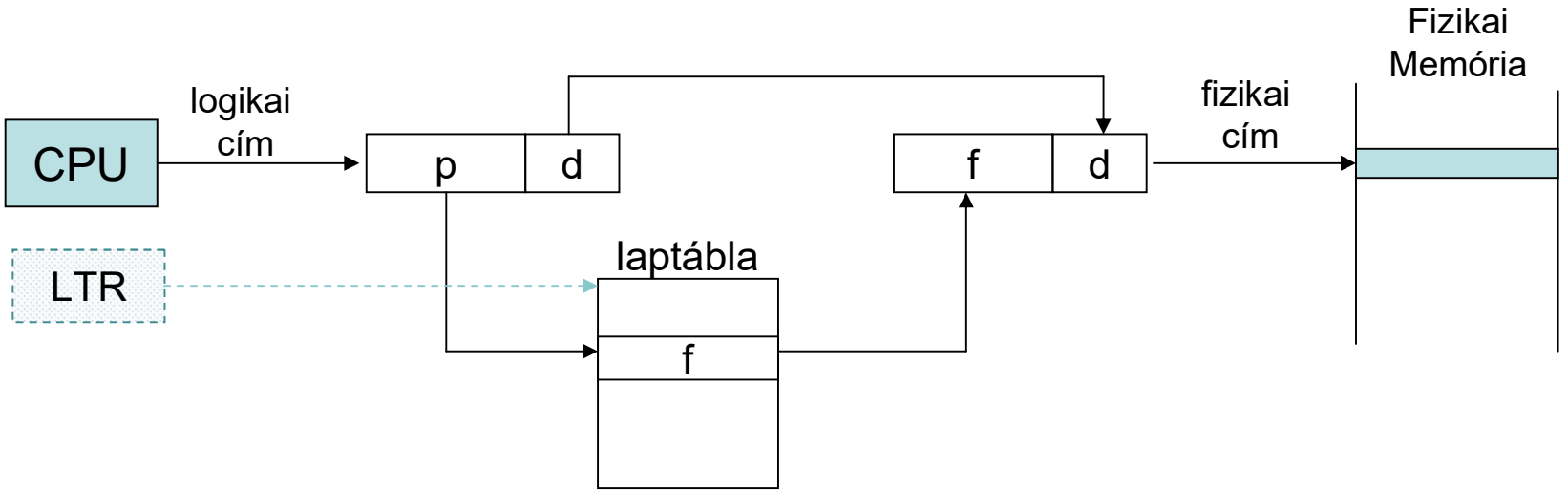
- **First-fit:** A legelső megfelelő méret
  Gyors
  Kihasználtság: 30%
- **Next-fit:** Keresés indítása az utoljára lefoglalt helyről
- **Best-fit:** A legkevesebb hulladékot eredményező hely megkeresése
  Lassabb
  A teljes területet végig kell nézni, hacsak a szabad területek nyilvántartása nem nagyság szerint rendezett
- **Worst-fit:** A legnagyobb helyet keressük
  Lassabb
  A teljes területet végig kell nézni, hacsak a szabad területek nyilvántartása nem nagyság szerint rendezett

Programok elhelyezése

Lapszervezés

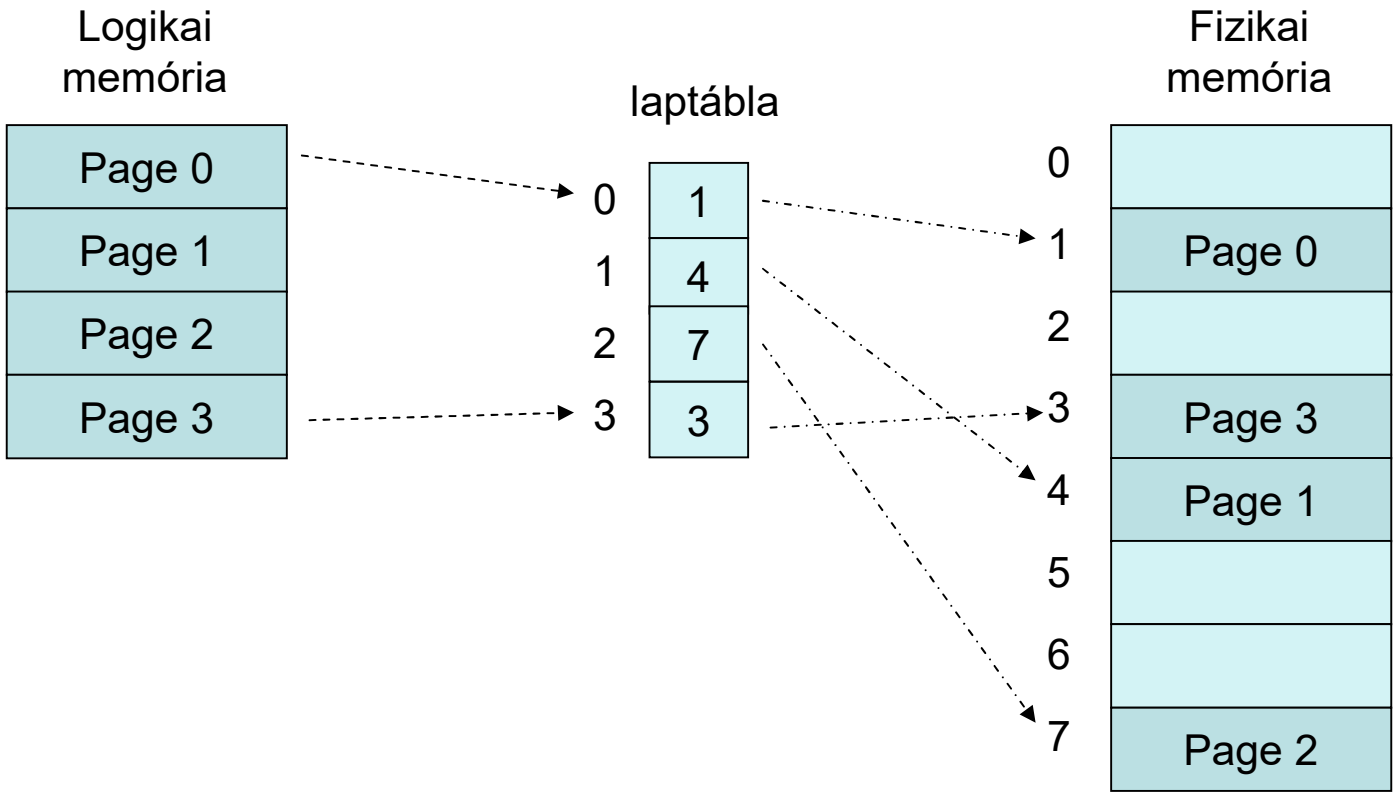


lap (page) méret = keret (frame) méret



Programok elhelyezése

Lapszervezés



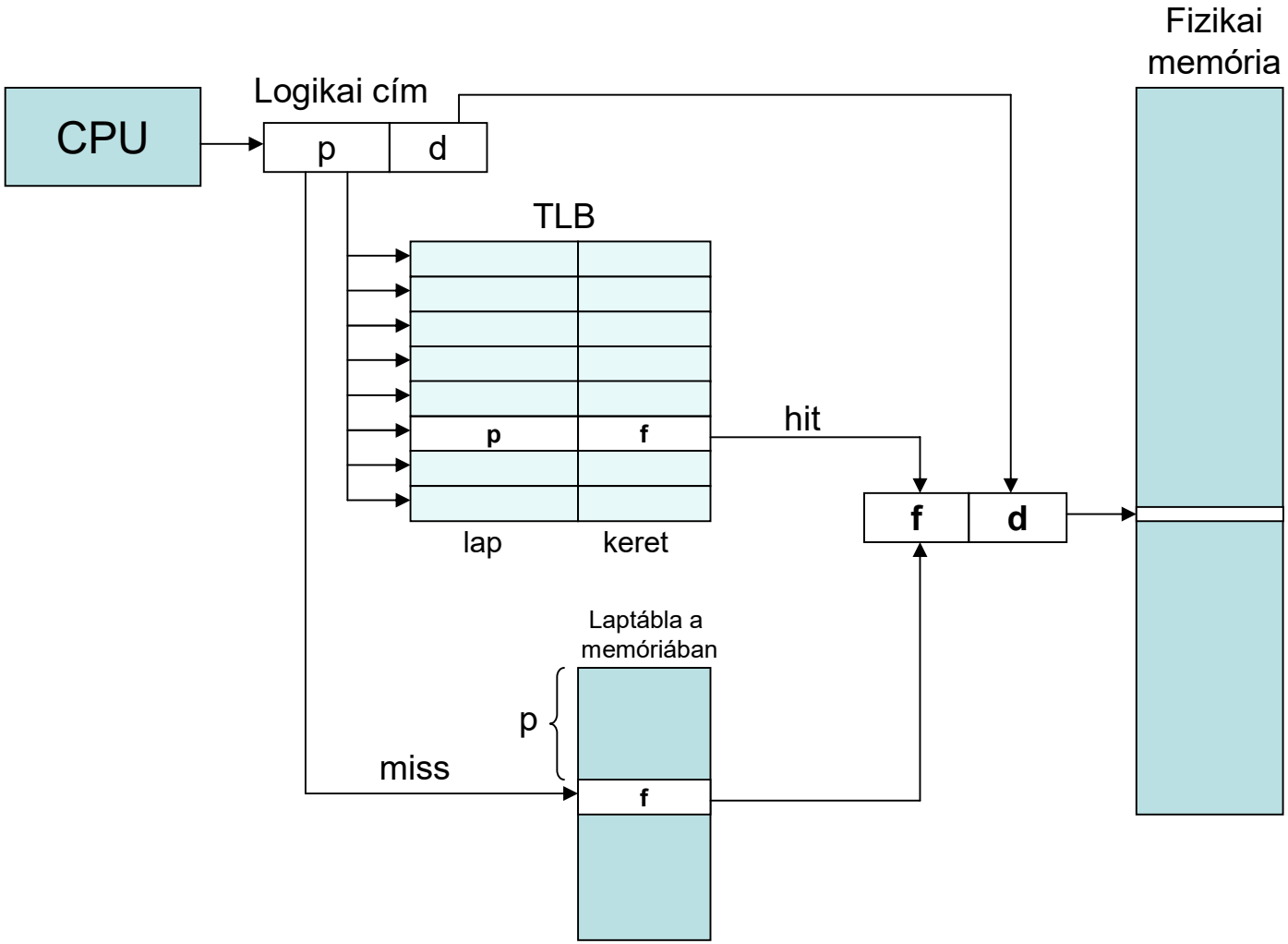
Programok elhelyezése

Lapszervezés

Laptábla implementálása

- Regiszter tömb
  - Gyors, de sok kell belőle
  - Privilegizált utasításokkal kezelendő
- Memóriában tárolt laptábla
  - Laptábla bázis regiszter
  - Folyamat átkapcsoláskor csak ezt kell átírni
  - Minden memóriaművelet 2 memóriaciklus
  - → cache kell
    - asszociatív cache
- Asszociatív tár
  - 8 – 16 rekesz: 80 – 90 % hit rate

Laptábla cache – TLB (Translation Lookaside Buffer)



Programok elhelyezése

Lapszervezés

Laptábla implementálása

Extra információk:

- Csak olvasható (konstans, kód)
- Írható/olvasható (adat)
- Csak végrehajtható (kód)
- Bejegyzés érvényes

Lapméret – belső fragmentáció: ½ frame folyamatonként

kis lapméret

jobb memóriakihasználtság

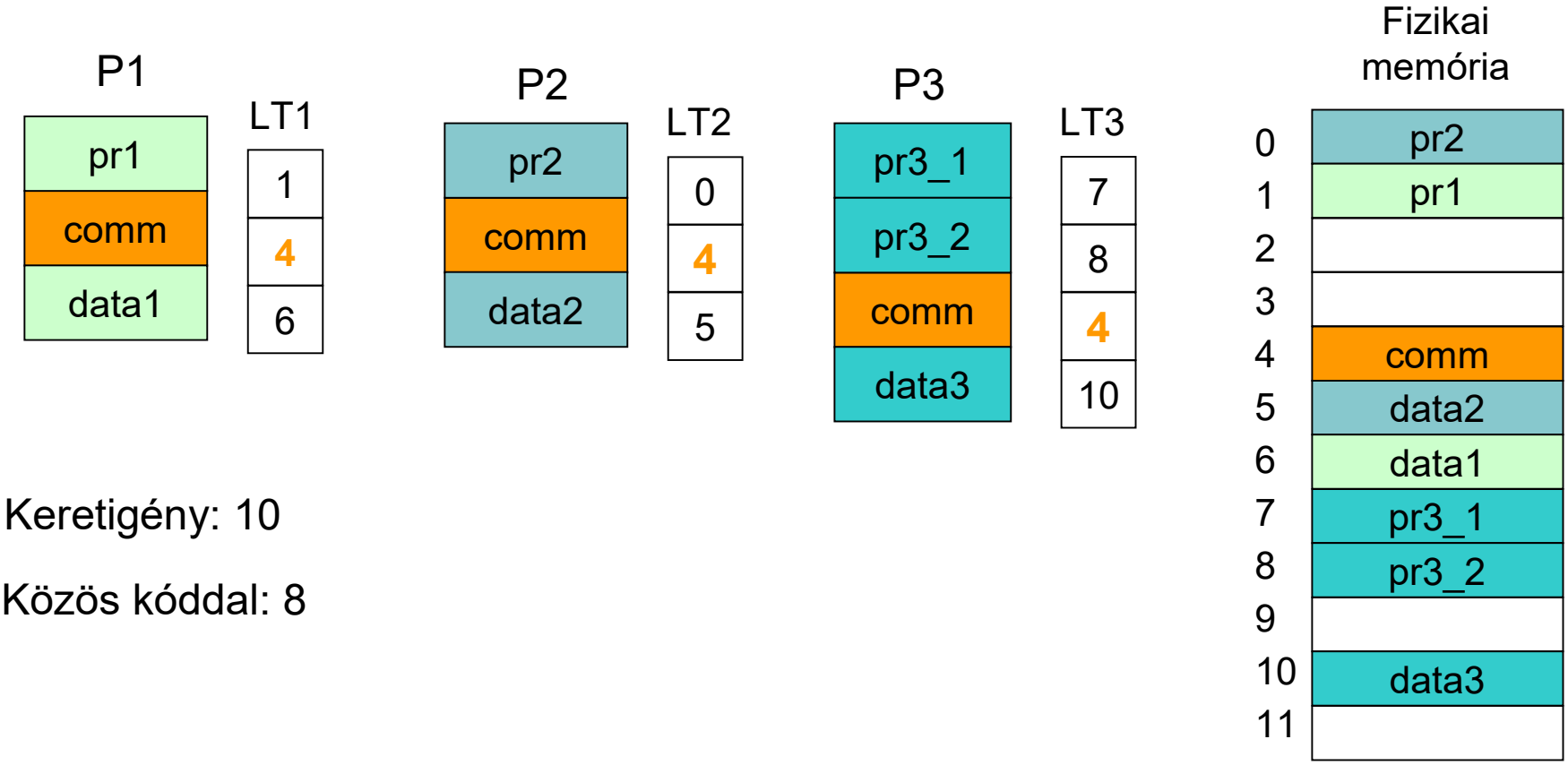
nagyobb méretű laptábla

Folyamat mérete: 134 186 byte		
Lapméret:	2048 byte (2^{11} byte)	16368 (2^{14} byte)
	65 lap + 1066 byte	8 lap + 3242
Belső fragmentáció:	2 048 – 1 066 = 982 byte	16368 – 3242 = 13126
Folyamatok maximális mérete: 1 Mbyte	512 lap	128 lap
Fizikai memória: 1 Gbyte (2^{30} byte)	$2^{11} * 2^{19} \rightarrow 2^{19}$ keret	$2^{14} * 2^{16} \rightarrow 2^{16}$ keret
Laptábla szélessége (extra bitek nélkül):	19 bit (4 byte)	16 bit (2 byte)
Laptábla maximális mérete:	4 * 512 = 2048 byte	2 * 128 = 256 byte

Programok elhelyezése

Lapszervezés

Közös programrészek – több laptáblában is lehetnek azonos bejegyzések



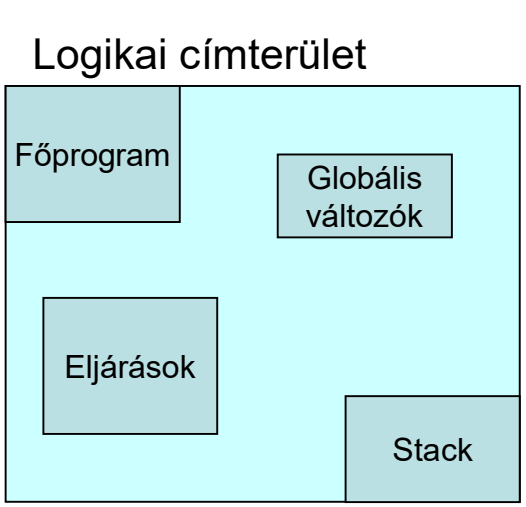
Keretigény: 10

Közös kóddal: 8

Programok elhelyezése

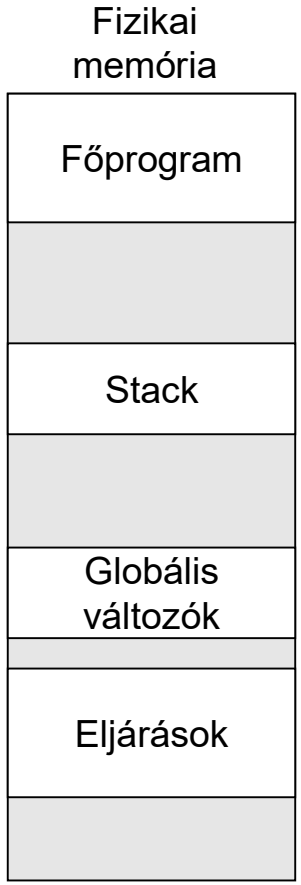
Szegmentálás

Főprogram
Eljárások
Függvények
Konstansok
Lokális változók
Globális változók
Stack
Standard C könyvtár



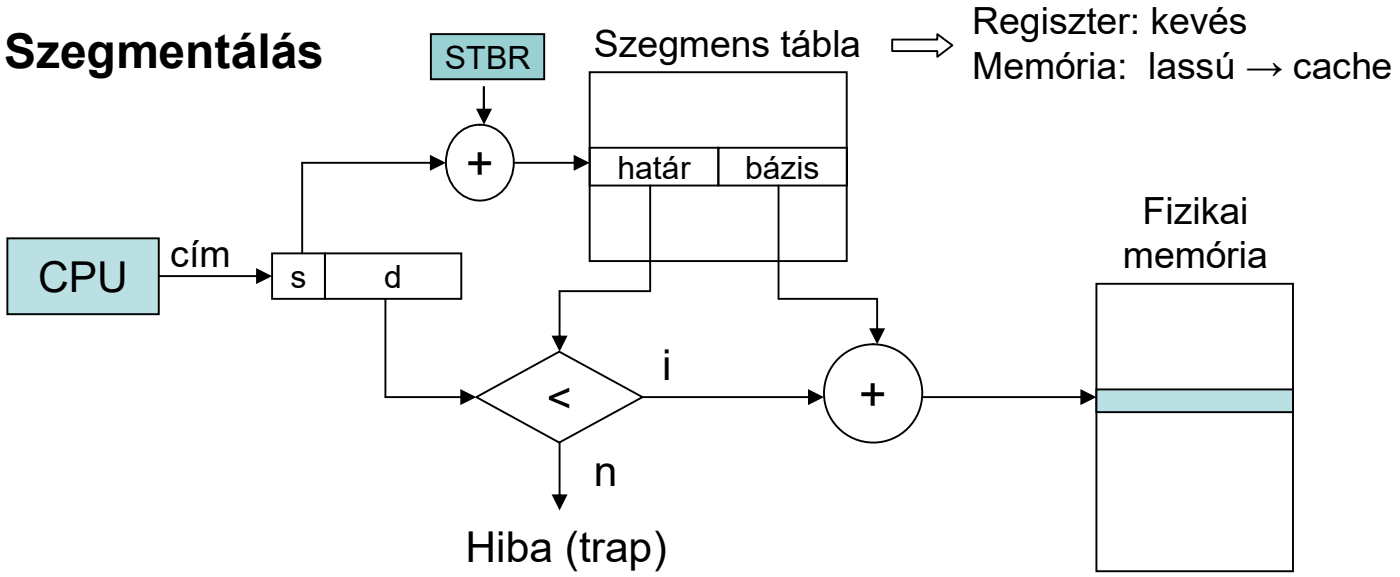
Logikai cím: <szegmens szám, offset>

Szegmens tábla
- bázis
- határ



Programok elhelyezése

Szegmentálás



Védelem

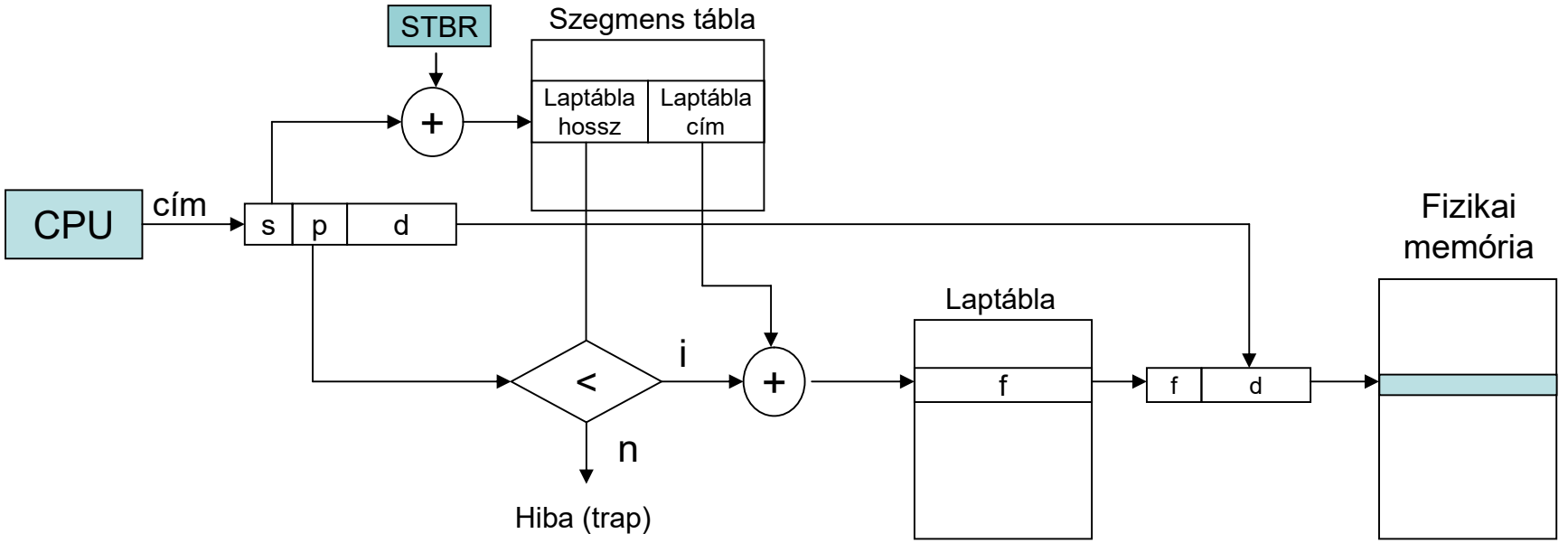
Szemantikai egységenként definiálható
Tömböket szegmensbe helyezve: automatikus túlfutás védelem
A szegmensek megosztottan használhatók

Fragmentáció

Hasonló a rugalmas partícióhoz

Programok elhelyezése

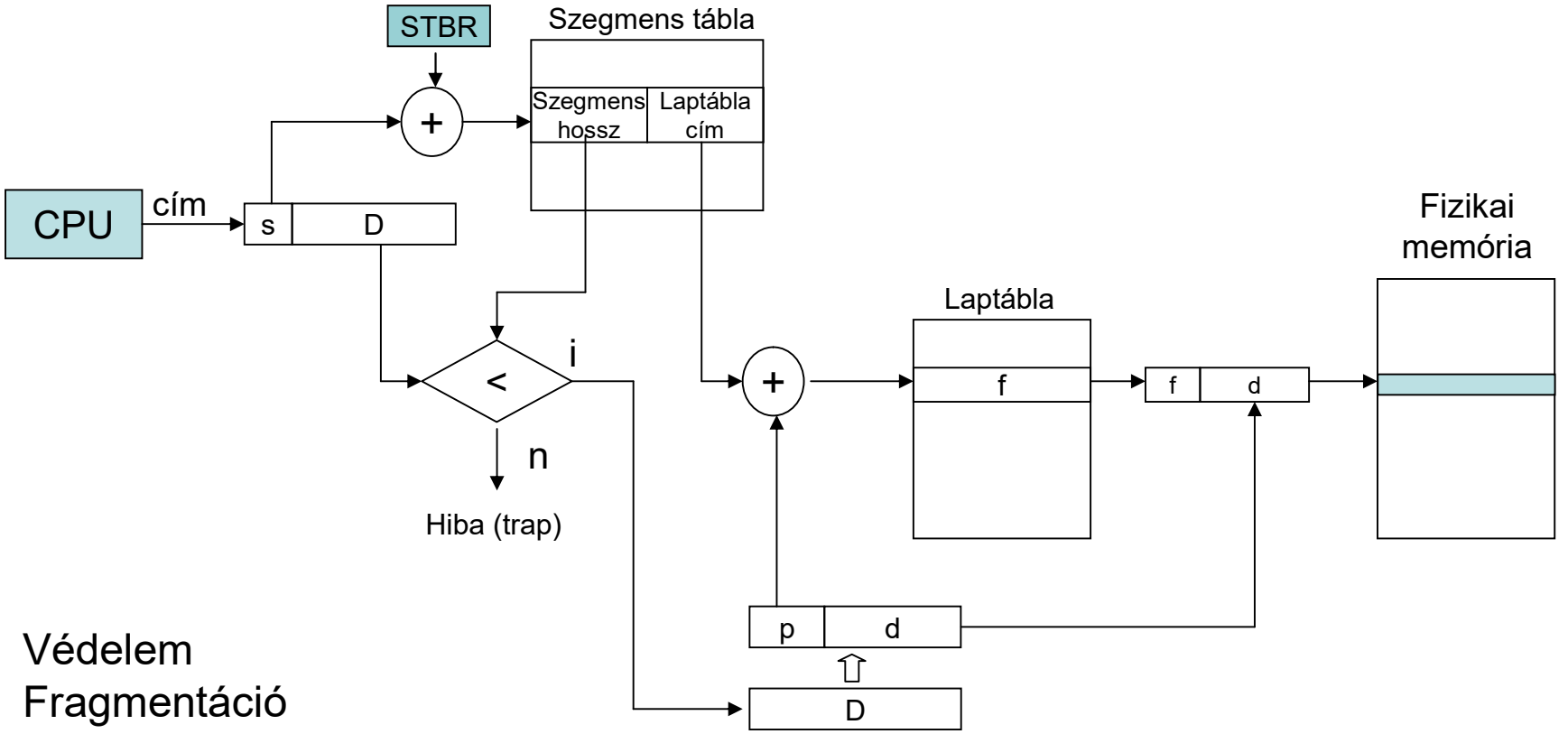
Szegmentált lapcímzés



Védelem
Fragmentáció

Programok elhelyezése

Lapozott szegmentálás



Védelem
Fragmentáció

Programok elhelyezése

IA-32: kétszintű laptábla szegmentálással