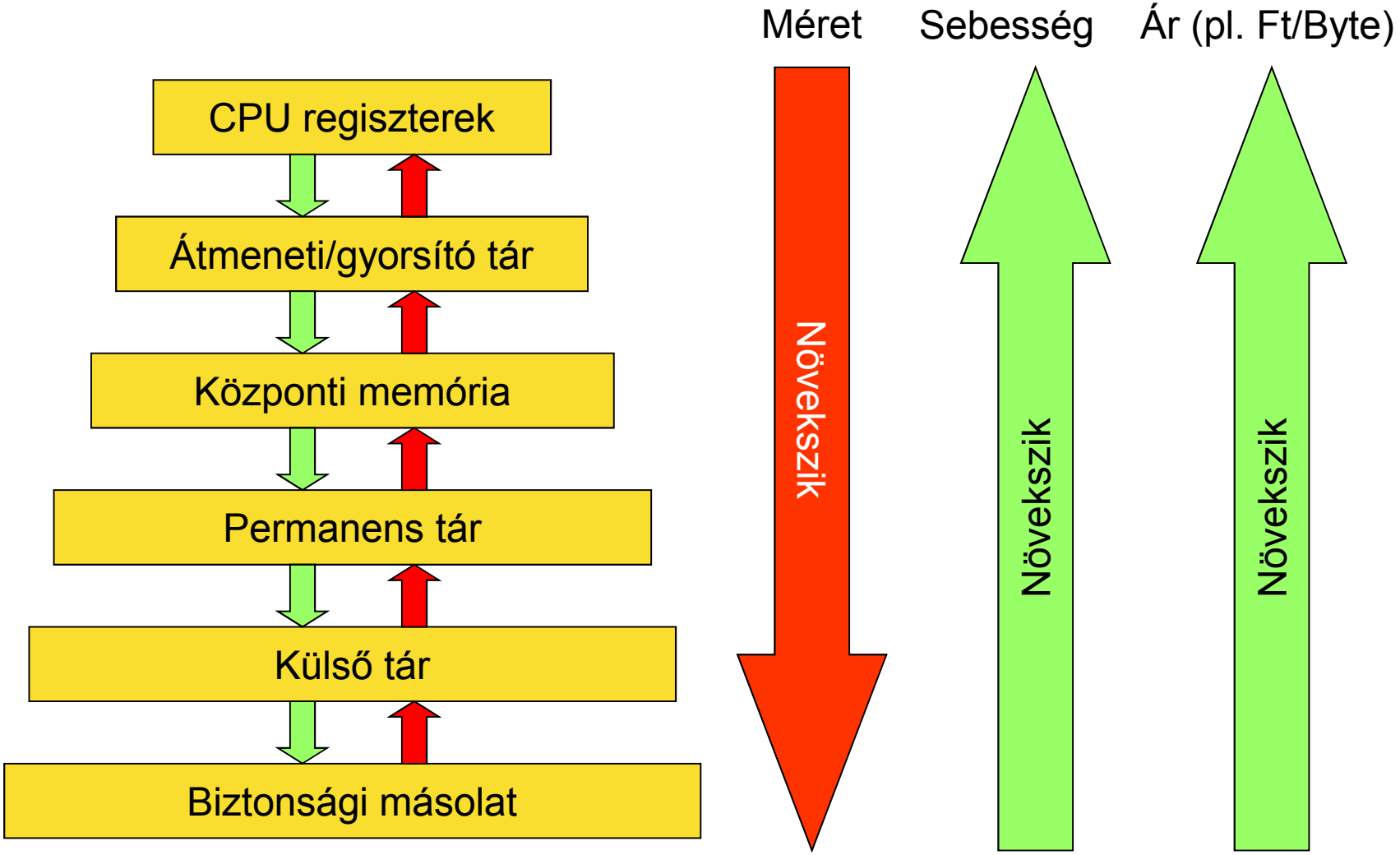
Memóriakezelés

dr. Kovácsházy Tamás

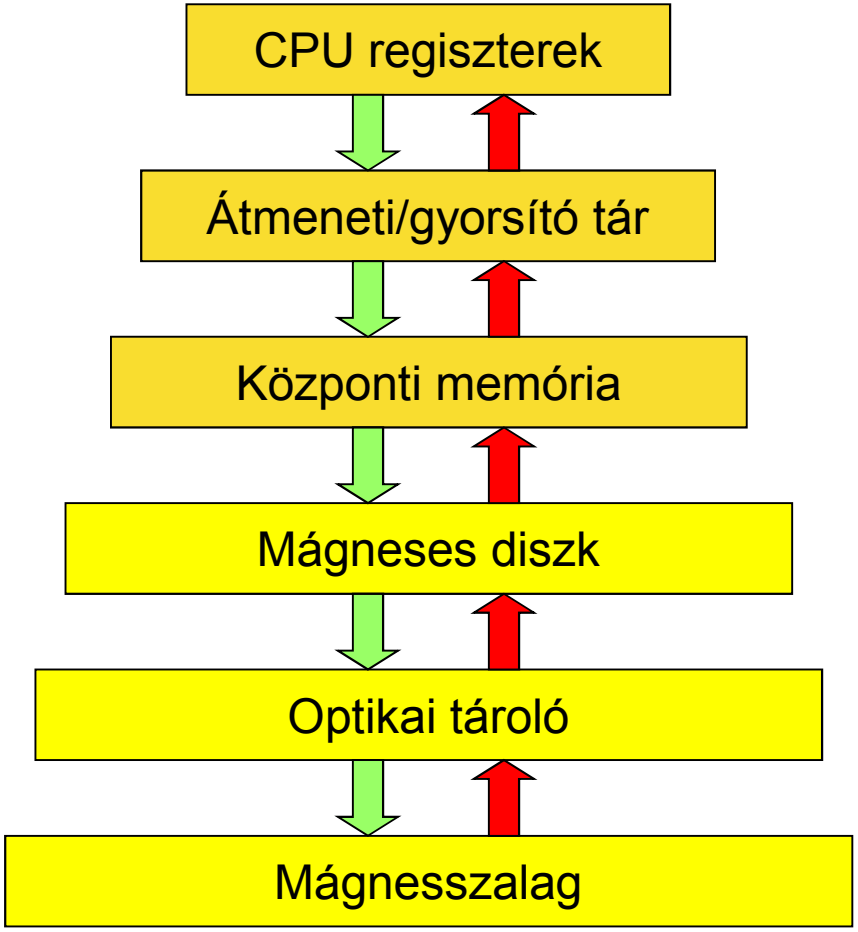
8. anyagrész,
Memóriakezelés

Méréstechnika és
Információs Rendszerek
Tanszék

Tárolóeszközök hierarchia ma



Tárolóeszközök hierarchia korábban



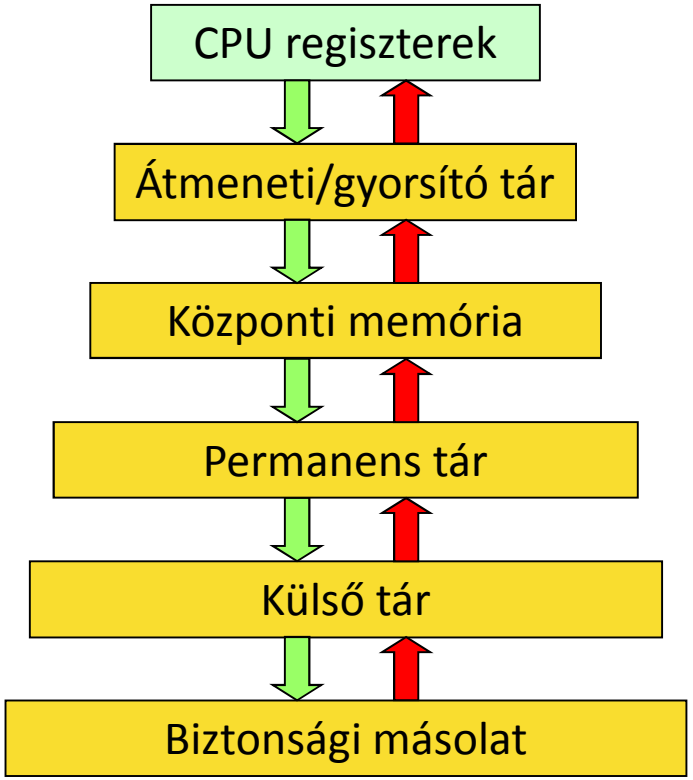
Flash memória (Pendrive, SSD) miatt nem igaz már!

Eltűnőben, Internet, hordozható HDD, pendrive!

Eltűnőben, HDD alapú mentés!

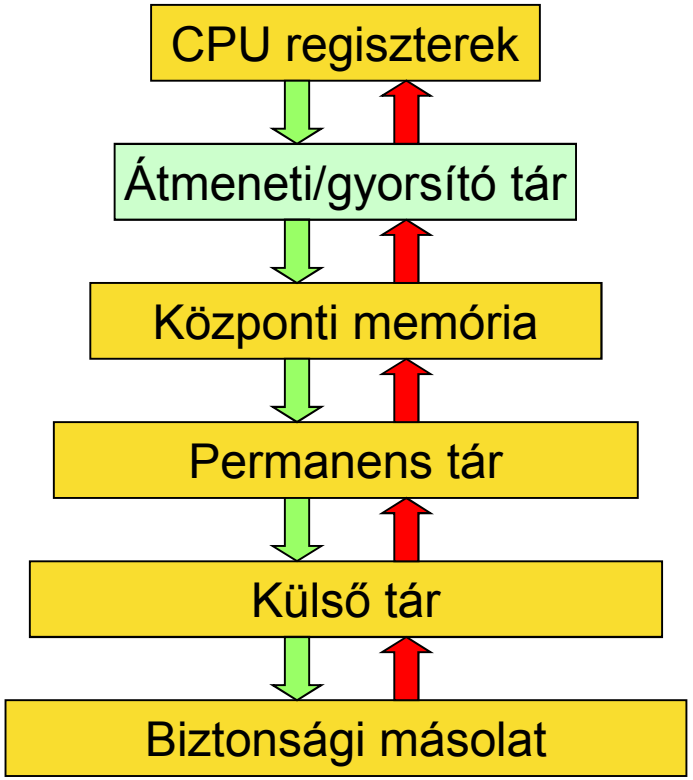
CPU regiszterek

- Típus: jellegzetesen D tároló.
  - 10-100 gépi szó.
    - x86 architektúrában kis számú.
  - Nem általános felhasználású egy része (PC, szegmens reg., stb.).
- Sebesség:
  - Az utasítás végrehajtása alatt akár többször elérhető.
    - Pl. Összeadás (ADD) utasítás.
- Ár:
  - Nehezen értelmezhető, architektúra függő...
- Felejtő memória (tápfeszültség)



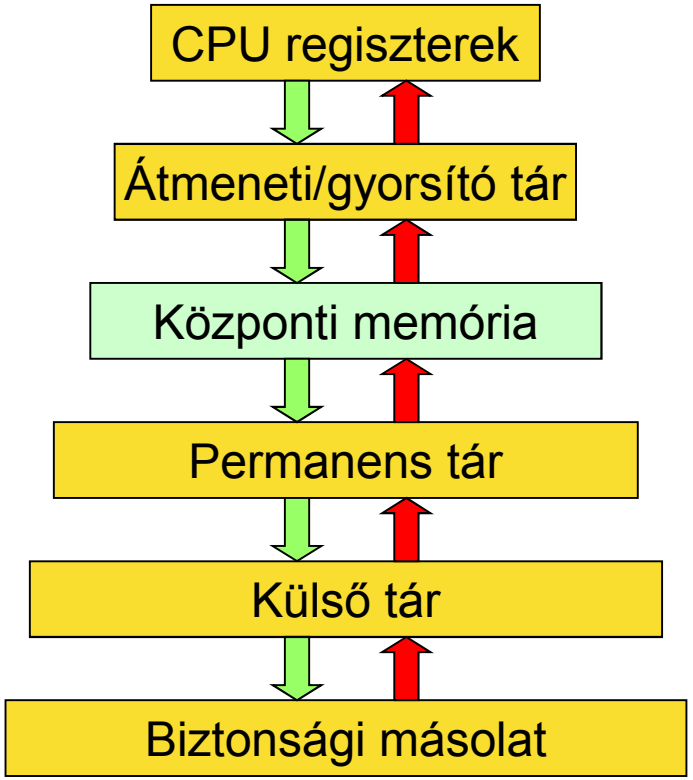
Átmeneti/gyorsító tár (Cache)

- Típusa: jellegzetesen SRAM
  - Többnyire többszintű.
  - 1. szint 64-128 Kbyte (I+D).
  - 2. szint 1-8 Mbyte.
  - 3. szint 4-32 Mbyte (ha van).
- Sebesség:
  - Sávszélesség: n*10 Gbyte/s.
  - 1. szint: Egy vagy néhány órajel ciklus.
  - 2. és 3. szint: 10 – n*10 órajel ciklus.
- Ár:
  - Magas, az SRAM cella nagy félvezető területet foglal.
- Felejtő memória (tápfesz. nélkül)



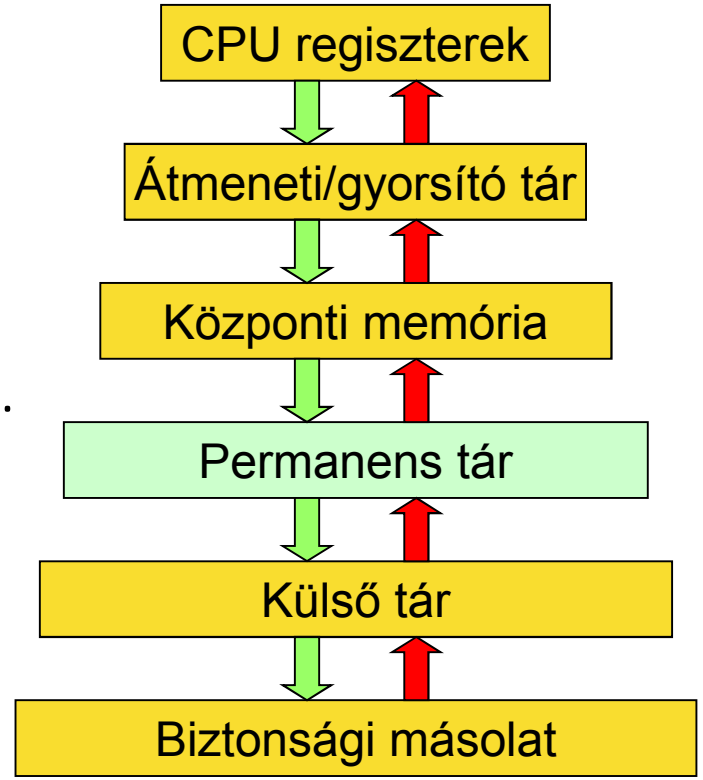
Központi (fizikai) memória

- Típusa: jellegzetesen DRAM
- n*10 Mbyte – n*100 Gbyte
- Sebesség:
  - Memory wall (és okai).
  - Max. és random access más.
    - A véletlen hozzáférés lassabb
  - Max.: n*1 Gbyte/s – 10 Gbyte/s.
  - Késleltetés: n*10 ns (worst case)
- Ár:
  - Erősen sebesség, méret, technológia, és piac függő.
- Felejtő memória: tápfesz. + idő
  - Frissíteni kell...



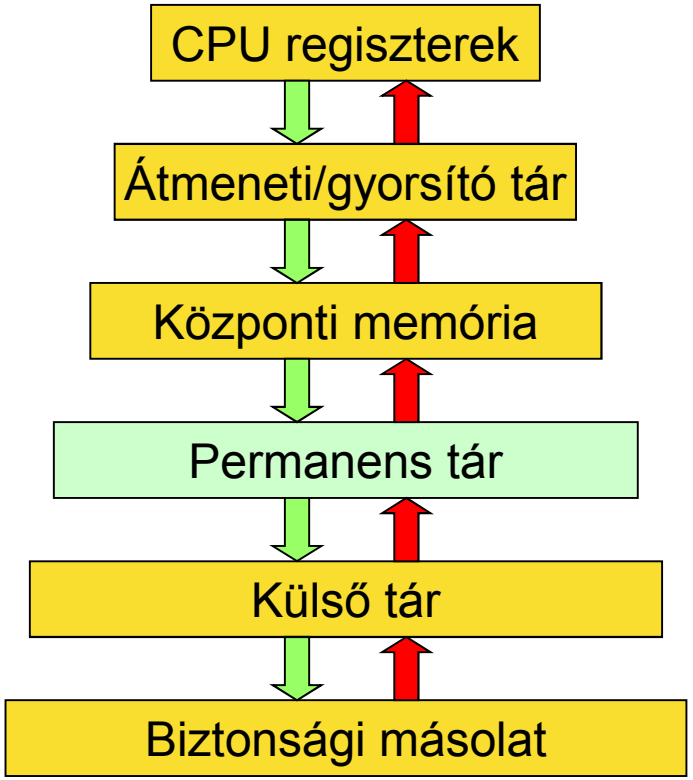
Permanens/háttér tár (HDD, Flash) 1.

- Típusa:
  - HDD (mágneses diszk).
  - Flash memória (pendrive, SSD, stb.).
  - Méret:n*1 Mbyte (Flash) – n*100 Tbyte.
- Fájl alapú elérés
  - Blokk elérésű eszköz.
  - Fájl-ként látjuk a tartalmat, kivéve NOR Flash).
- Sebesség:
  - Max. és random access más.
    - Véletlen hozzáférés más HDD esetén
  - Olvasás és írás eltérő.
  - Max.: n*10 Mbyte/s (olcsó pendrive) – n*1 Gbyte/s (RAID).
  - Késleltetés: n*10 ns (Flash) – kb. 10 ms



Permanens/háttér tár (HDD, Flash) 2.

- Ár: Erősen sebesség, méret technológia, és piac függő.
- Nem felejtő de meghibásodhat:
  - HDD: MTBF
    - Leggyengébb láncszem (mechanikai elemek)
  - Flash: Véges alkalommal írható (wear)
    - Nincs tapasztalat, hogy valójában mit bír ki
- Hogyan hat ez az Operációs rendszerre?
  - A meghibásodás nem feltétlenül jelentkezik azonnal (rejtett hiba)
  - Az adatvesztés többnyire katasztrofális következményekkel jár
    - OS-hez tartozó fájl: Az OS nem indul, vagy egyes részei nem működnek (okozhat további hibákat)
    - Alkalmazáshoz vagy adathoz tartozó fájl



További lépések

- Központi memória kezelése.
  - Fizikai, logikai, és virtuális memória fogalma.
  - Ezeknek a megfeleltetése és kezelése.
- Permanens tár kezelése.
  - A virtuális tárkezelésen keresztül kapcsolódik majd a központi memória kezeléséhez.
  - A permanens tár kezelése, szerkezete, partíció, fájlrendszer, fájl fogalma.
  - RAID, NAS, SAN.

Memória

- A CPU használja:
  - Utasítások betöltése.
  - Adatok olvasása és írása.
- Egymás után következő memória műveletek folyamáról van szó.
  - Olvasások és írások adott sorrendben.
  - Ahogy ezt a program kód előírja.
- Folyamat támogatás (szeparáció).
- Tekintsünk el a átmeneti tártól (cache)!
  - Csak a hozzáférés sebességét befolyásolja.

Logikai és fizikai cím

- Logikai cím (logical address):
  - A központi egység (CPU) generálja a folyamat futása közben.
  - Logikai címtartomány (logical address space): Egy adott folyamathoz tartozó logikai címek összessége.
- Fizikai cím (physical address):
  - Egy adott memória elem címe, ahogy az a fizikai memória buszon megadásra kerül a memória vezérlő által.
  - Fizikai címtartomány (physical address space): Egy adott folyamathoz tartozó fizikai címek összessége.
- Ha ezek eltérnek:
  - A logikai címet virtuális címnek (virtual address) hívjuk.
  - Ebben az esetben a futási idejű leképzést az MMU valósítja meg (volt már róla szó).

Címképzés (address binding) 1.

- Mikor történik meg a logikai és a fizikai címek közötti leképzés?
- Fordítási idejű (compile/link time)
  - Ismert a betöltés helye.
  - Abszolút címzés.
  - A program fizikai címeket tartalmaz.
  - Pl. Egyszerű beágyazott rendszerek firmware-je, BIOS/EFI és OS kernel legalapvetőbb része, DOS *.com programok.
- Betöltési idejű (load time)
  - Áthelyezhető kód (relocatable code).
  - Betöltés során oldódik fel a leképzés.
  - A program fizikai címeket használ.

Futási idejű címképzés (execution time) 2.

- Futási időben is változhat a leképzés.
  - A folyamat számára transzparens módon más fizikai címterületre kerülhet.
  - Speciális HW szükséges ehhez (MMU).
    - Megfelelő sebességű végrehajtáshoz.
  - A folyamat nem látja (láthatja), hogy milyen fizikai címeken található a memória, amit elér.
    - Mindig logikai/virtuális címeket használ.
- A fizikai címekhez kötött logikai címek használata alapvető a modern operációs rendszerek működése szempontjából.

Dinamikus betöltés (Dynamic loading)

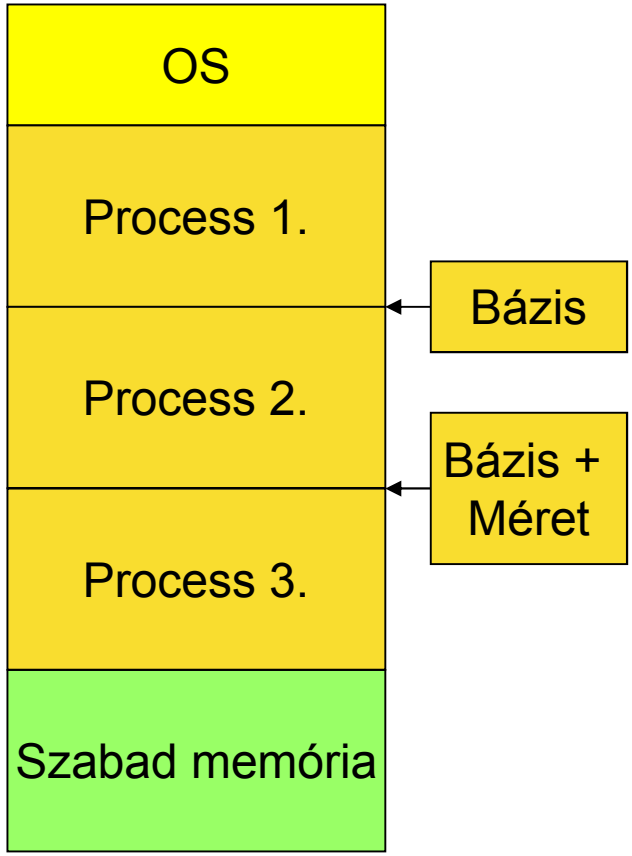
- Bizonyos funkciók nem töltődnek be, amíg nem használják őket.
  - Gyors program indulás.
  - Alacsonyabb memória használat.
  - A program feladata a dinamikus betöltés megvalósítása.
    - Az operációs rendszer nem tud róla, nem támogatja azt.

Dinamikus kapcsolatszerkesztés

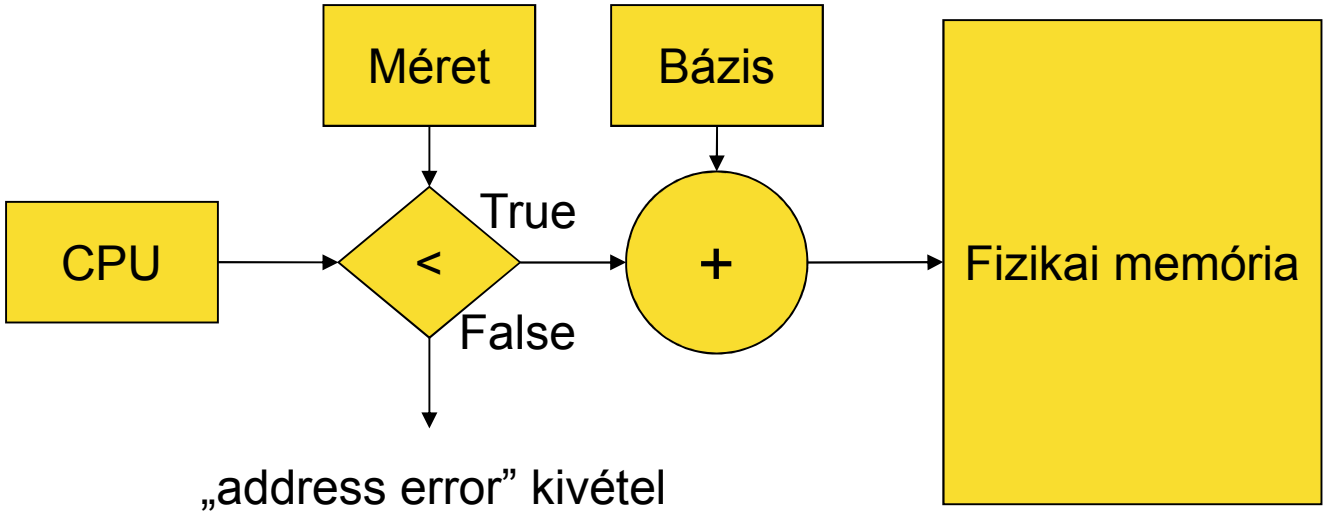
- Dynamic linking
- A dinamikus betöltés operációs rendszer támogatással.
  - Dynamically linked/loaded library (Windows *.dll).
  - Shared Object (UNIX/Linux *.so).
  - A dinamikusan beszerkesztett programkönyvtárak több program számára is elérhetőek (code sharing).
- Megvalósítás:
  - A program csak egy csonkot (stub) tartalmaz.
  - A csonk feladata a dll/so megtalálása vagy betöltése az OS felhasználásával.
  - Egy adott funkciójú dll/so több eltérő verzióban is jelen lehet a rendszerben.
    - Melyik töltődik be?
    - UNIX: LD_LIBRARY_PATH környezeti változó

Alapmegoldás (Változó méretű partíciók)

- Minden egyes folyamat egy összefüggő, statikus, előre kiválasztott fizikai címtartományba kerül.
  - Báziscímtől kezdődik (Bázis).
  - Adott méretű (Méret).
  - Bázisrelatív címzése
- A folyamatnak ebbe a memória területbe kell beférnie élettartama során.
- Nem túl hatékony megoldás, de első iterációnak jó.



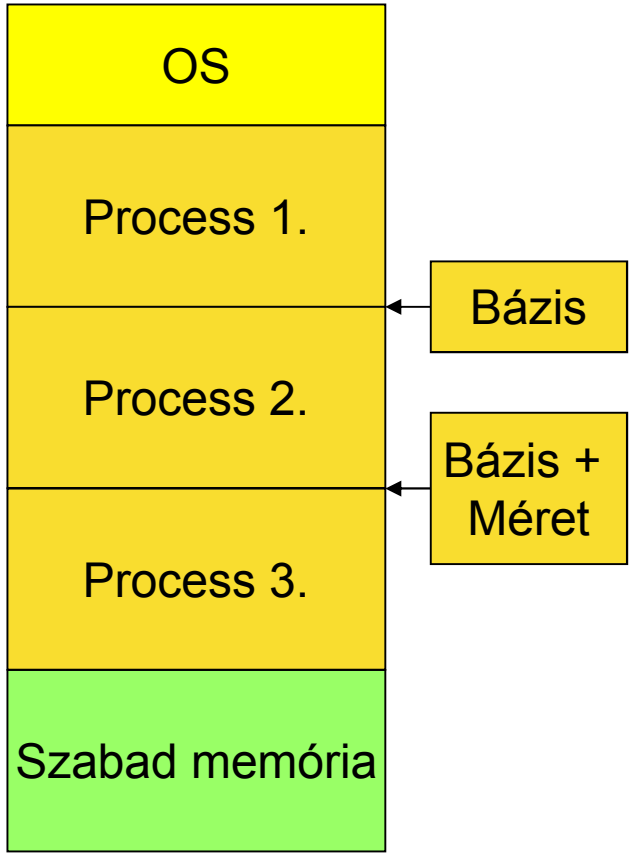
Egyszerű megvalósítás



- Implementációhoz szükséges:
  - Folyamatonként 2 védett (kernel) módban állítható regiszter.
  - 1 összeadás, és egy vizsgálat.
- Ha a folyamat kicímez a rendelkezésre álló memóriából, akkor kivétellel jelezzük a CPU-nak.
  - Azt kernel módban kezeli a CPU.

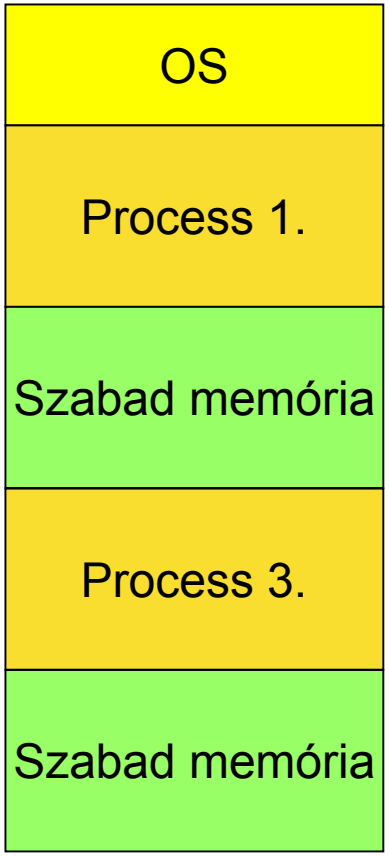
Problémák 1.

- Külső tördelődés (external fragmentation):
  - Process 2. kilép.



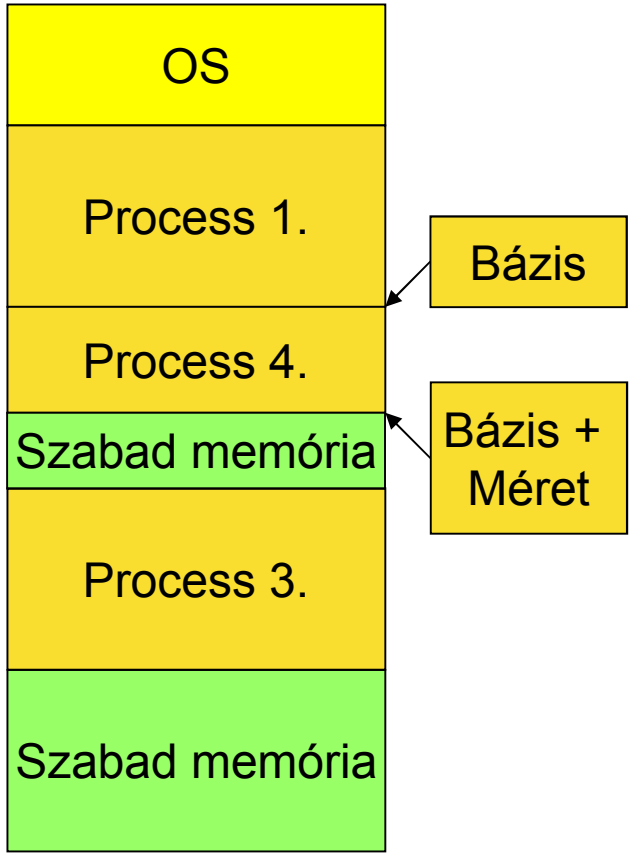
Problémák 1.

- Külső tördelődés (external fragmentation):
  - Process 2. kilép.
  - A helye felszabadul.



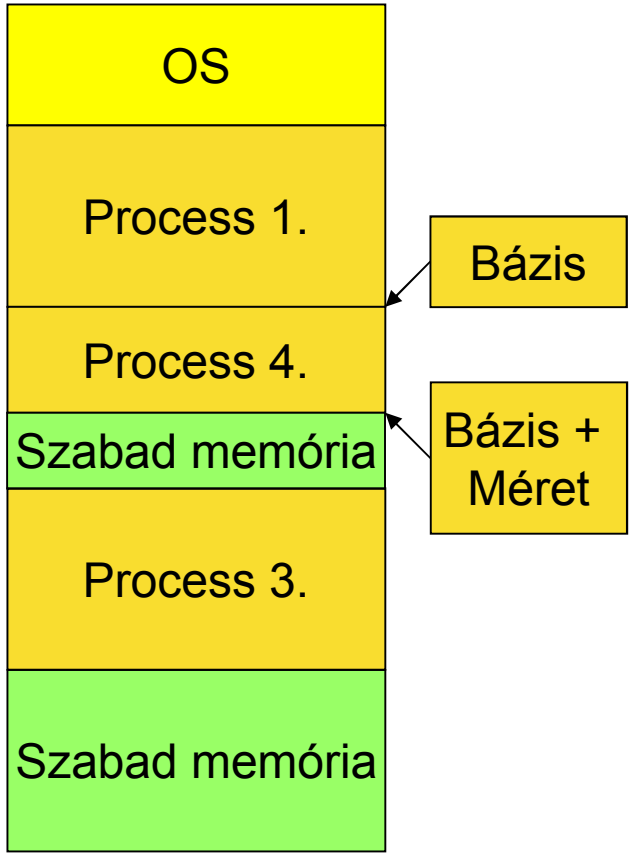
Problémák 1.

- Külső tördelődés (external fragmentation):
  - Process 2. kilép.
  - A helye felszabadul.
  - Helyére egy kisebb Process 4. töltődik.



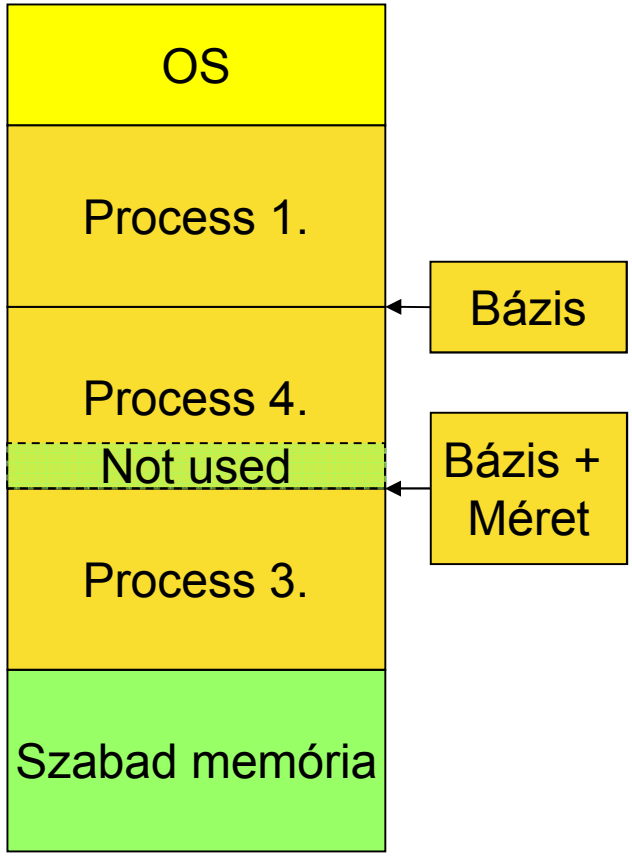
Problémák 1.

- Külső tördelődés (external fragmentation):
  - Process 2. kilép.
  - A helye felszabadul.
  - Helyére egy kisebb Process 4. töltődik.
  - Process 4. és Process 3. közötti memória terület nem használható mérete miatt.
    - Nem fér bele folyamat.
    - Lényegében elveszik az OS számára.



Problémák 2.

- Belső tördelődés (internal fragmentation):
  - Ugyan az a folyamat, mint a külső tördelődés, de...
  - Process 4. és Process 3. közötti kisméretű memória terület túl kicsi lehet.
    - Odaadjuk Process 4-nek.
    - Nem érdemes nyilvántartani.
  - A folyamatok által lefoglalt területen belül biztosan van nem használt (elvesző) memória terület



Problémák 3.

- A programok által igényelt memória dinamikusan változik, nehéz egy felső korlátot adni.
- A programok rendelkeznek az alábbi tulajdonságokkal:
  - A programkód kis része használja az adatok kis részét az idő nagy részében (lokalitás).
  - Egyes részek soha nem kerülnek felhasználásra.
    - Pl. vizsgálatok szerint az MS Office funkcióinak 90%-át a felhasználók 90% soha nem használja.
    - Dinamikus kapcsolatszerkesztés, csak a ténylegesen használt kódrészleteket töltjük be, rendszerszinten egy példányban.

Foglalási (allokációs) stratégiák

- Első megfelelő (First fit):
  - A tár elejéről indulva az első elégséges méretű területet allokálja.
- Következő megfelelő (Next fit):
  - Nem a tár elején kezdi a keresést, hanem a legutolsóra allokált terület után.
- Legjobban megfelelő (Best fit):
  - A legkisebb szabad hellyel járó helyre tesszük be.
  - Minimális külső tördelődés.
- A legrosszabban illeszkedő (Worst fit):
  - A legnagyobb szabad hellyel járó helyre tesszük be.
  - Talán a maradék helyre még befér majd valami.

Foglalási stratégiák értékelése

- A külső tördelődést befolyásolják.
- „Első/Következő/Legjobban megfelelő” kb. a teljes memória terület 33%-át, a használt memória 50%-át nem tudja allokálni, az kihasználatlanul marad.
- A „Legrosszabban illeszkedő” algoritmus esetén a teljes memória 50%-a kihasználatlan marad.
- „Első/Következő megfelelő” a legkevésbé erőforrás igényes, a másik kettő a teljes lista végignézését igényli.
- Ilyen példa lehet a ZH vagy a vizsga sorban.

Példa

- Magyarázza el a változó méretű memória partíciók lefoglalásánál használt
- a, first fit
- b, next fit
- c, best fit
- d, worst fit
- algoritmusokat. Egy rendszerben az adott pillanatban 23K, 64K, 10K, 80K, 12K, 50K és 40K méretű szabad területek vannak. Hogyan fog a fenti négy algoritmus sorrendben a 65K, 21K, 48K, 13K, 62K méretű memória igénynek helyet foglalni?

Megoldás „első megfelelő” algoritmusra

- Foglalások: 65K, 21K, 48K, 13K, 62K
- Szabad helyek:

0. 23K, 64K, 10K, 80K, 12K, 50K és 40K (65K foglal)
1. 23K, 64K, 10K, 15K, 12K, 50K és 40K (21K foglal)
2. 2K, 64K, 10K, 15K, 12K, 50K és 40K (48K foglal)
3. 2K, 16K, 10K, 15K, 12K, 50K és 40K (13K foglal)
4. 2K, 3K, 10K, 15K, 12K, 50K és 40K (62K foglal)

Nem lehetséges, nincs szabad terület...

Megoldás „legjobban megfelelő” algoritmusra

- Foglalások: 65K, 21K, 48K, 13K, 62K
- Szabad helyek:

0. 23K, 64K, 10K, 80K, 12K, 50K és 40K (65K foglal)
1. 23K, 64K, 10K, 15K, 12K, 50K és 40K (21K foglal)
2. 2K , 64K, 10K, 15K, 12K, 50K és 40K (48K foglal)
3. 2K , 64K, 10K, 15K, 12K, 2K és 40K (13K foglal)
4. 2K , 64K, 10K, 2K, 12K, 2K és 40K (62K foglal)
5. 2K , 2K, 10K, 2K, 12K, 2K és 40K

Az igények kielégíthetőek

Szabad helyek tömörítése

- Compaction, Garbage collection.
- Megoldja a külső és belső tördelődést.
  - Átrendezi a futó folyamatokat az optimális memória használat elérésére.
  - Egy, összefüggő szabad területet hoz létre.
- Nagyon erőforrás igényes.
  - Át kell állítani a bázis címeket.
  - A fizikai memóriát másolni kell.

További lehetőségek

- A fizikai memória kezelését általánosabban és hatékonyabban kell megvalósítani.
  - Különösen kritikus volt ez a korai (70-es évek) számítógépekre jellemző n*10 kByte fizikai memóriákkal.
  - De a mai n*1Gbyte méretű memóriák esetén sem tűnt el a probléma.
- Tárcsere (swapping)
- Szegmens szervezés
- Lapszervezés

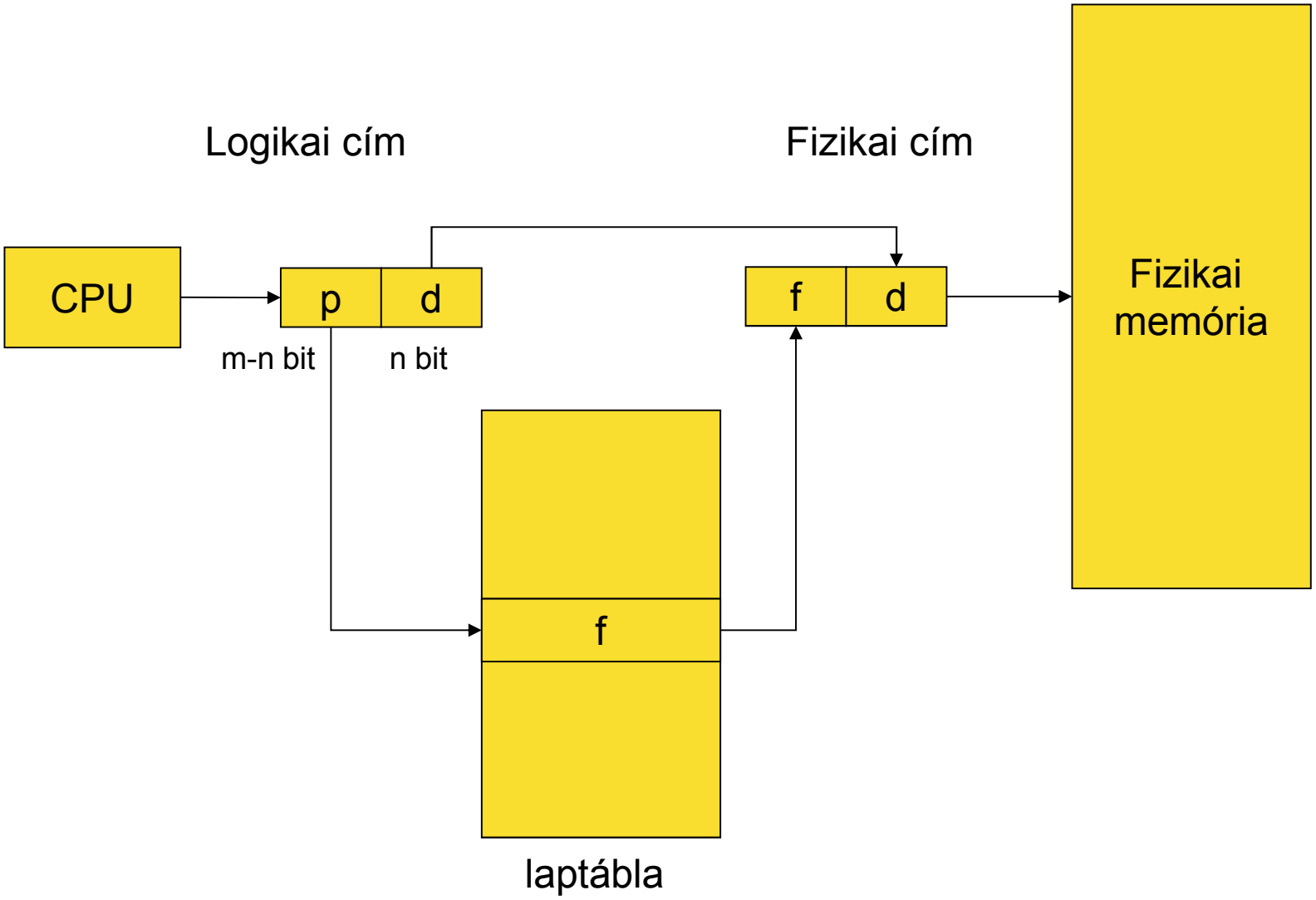
Tárcsere (swapping)

- A teljes folyamat háttértárolóra is kiírható:
  - Ha nincs futó állapotban és nincs folyamatban lévő I/O művelet (DMA sem a területre).
    - Kivéve az OS területén lefoglalt pufferekbe/pufferekből végzett I/O.
  - Ha a címképzés futási időben történik:
    - Akár még más fizikai címre is kerülhet a visszatöltött folyamat.
  - A tárcserével összekapcsolt kontextus váltás nagyon időigényes (a háttértár lassú a memóriához képest).
    - A teljes folyamatot ki kell írni majd visszaolvasni!

Lapszervezés (paging)

- A folyamathoz tartozó fizikai memória terület lehet nem folytonos is.
- A fizikai memóriát keretekre (frame) osztjuk.
- A logikai memóriát lapokra (page) osztjuk.
- Minden egyes logikai címet két részre oszthatunk.
  1. Lap szám (page number, p).
  2. Lap eltolás (page offset, d).
- A lap szám a laptábla indexelésére szolgál
  - A laptábla az egyes lapokhoz tartozó fizikai báziscímet tárolja (f).
- A keret tábla (frame table) tartja nyilván az üres kereteket.

Címtranszformáció laptáblával



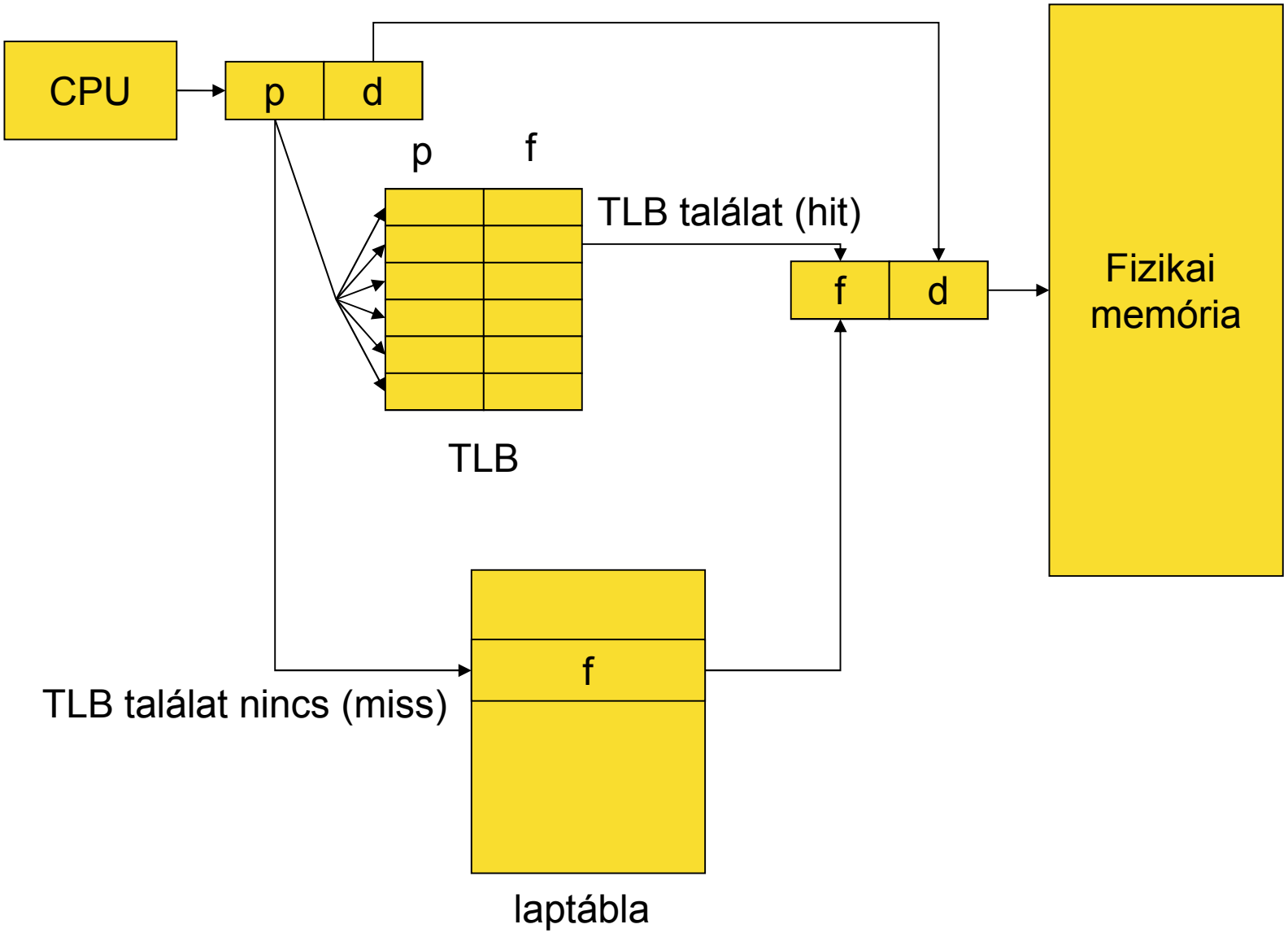
Lapszervezés tulajdonságai

- A leképzést hardver végzi
- Szeparáció megvalósul (folyamat létrehozható).
  - A folyamat által látott logikai címtartomány, és a ténylegesen használt fizikai címtartományok teljesen elkülönülnek.
  - A folyamatok nem férnek hozzá egymás lapjaihoz.
  - Az operációs rendszer kezelheti a lap- és kerettáblát (kernel mode).
- Külső tördelődés nincs.
- Belső tördelődés átlagosan fél lapnyi (kis lapméret lenne jó).
- A modern gépekben sok memória van (nagy lapméret lenne jó):
  - Mert egyszerűsödne az adminisztráció.
  - Kisebb lehetne a lap- és kerettábla (kevesebb bejegyzés).

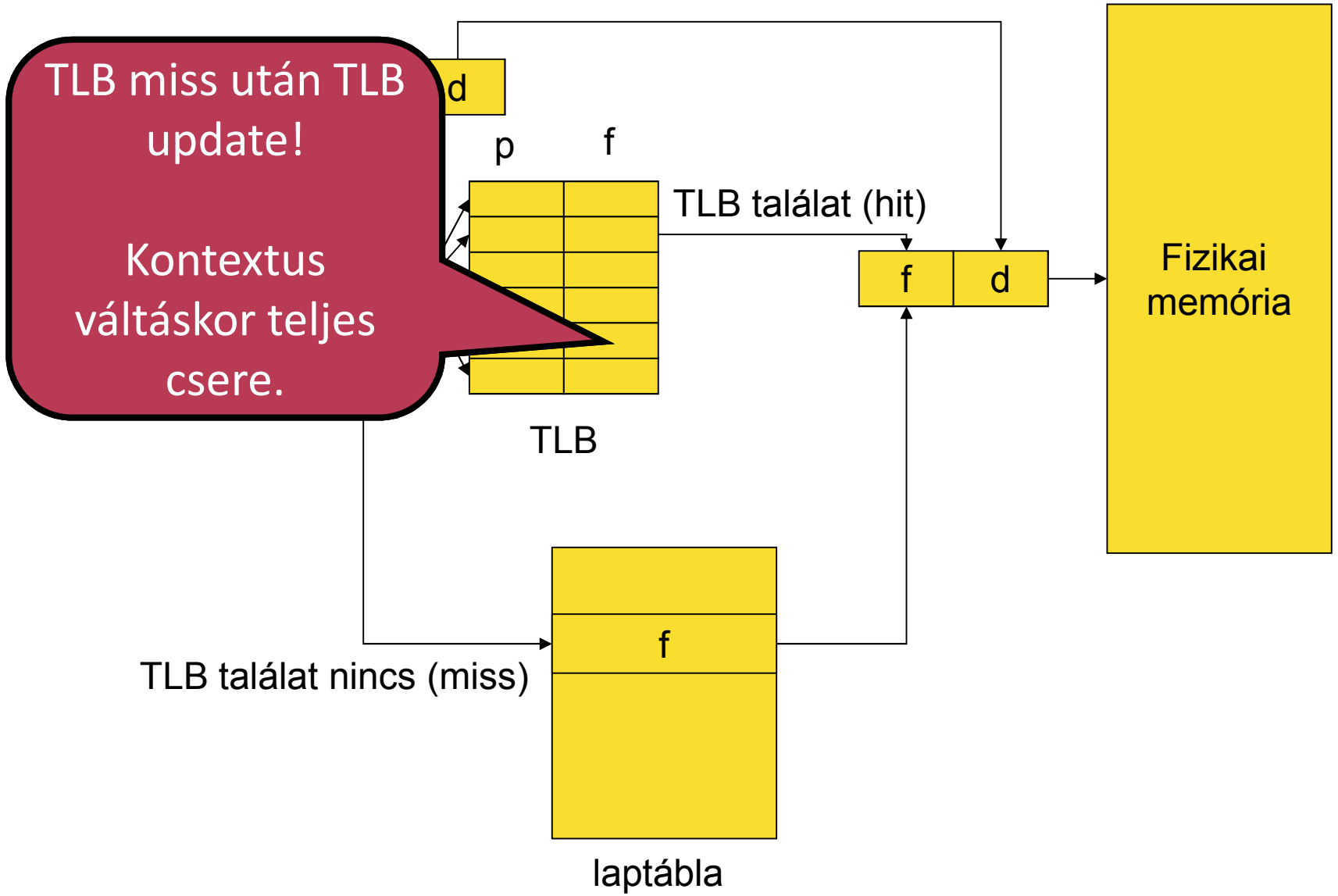
Keresés a laptáblában.

- A lap- és kerettábla nagy lehet a modern OS-ekben:
  - 4Kbyte-os lapok esetén 32 bites címzésnél 2^{20} bejegyzés van a laptáblában, ami 4MByte memóriát igényel minimum.
  - Többféle lapméret támogatása (a folyamat memória igényének megfelelően).
- Megoldás:
  - Többszintű laptáblák (hierarchical paging).
  - Hash-elt laptáblák (hashed page table)
  - Inverted page table
- A keresés a laptáblában lassú:
  - 2x annyi idő (laptábla indexelés és olvasás majd adatelérés).
  - Asszociatív lekérdezés (Translation look-aside buffer, TLB) kombinálva a laptábla használatával.

Asszociatív leképzés



Asszociatív leképzés



TLB hatása

- A kérdés a találati arány (hit ratio)?
  - A TLB méretétől és a végrehajtott kódtól függ.
    - Hány lapra és milyen sorrendben hivatkozunk a lapokra.
    - A lapok milyen arányban férnek be a TLB-be, és milyen a TLB frissítési algoritmusa (új bejegyzés milyen régi bejegyzés helyére kerül).
  - Tipikusan 80-90% körül van.
- Hatása a teljesítményre: AMD Phenom (2007)
  - A TLB hibás, a BIOS-ból kikapcsolható, eltérés:
    - 20% teljes (memória benchmark is), 14% alkalmazás
    - Memória benchmarkok esetén akár 50% is.
    - http://techreport.com/articles.x/13741/4

Egyéb információk a laptáblában

- Kiegészítő bitek a laptáblában:
  - Valid/invalid bit
    - Benne van-e az adott lap a fizikai memóriában.
    - Ha nem, be kell hozni.
  - Read/read-write bit, execute bit, stb.
    - Végrehajtható memória műveleteket ad meg.
    - Pl. No Execute : Buffer túlcsordulás ellen az adatokra.
  - Referenced/Used bit
    - Memóriaművelet esetén az MMU beállítja
- A HW (MMU) beállítja/ellenőrzi, ha a szabályokkal ellentétes használatot talál, akkor kivételt generál, amit az OS kezel.
- Az OS is felhasználja ezeket.

Lapok megosztása

- Folyamatok memória tekintetében szeparálva vannak egymástól.
- De teljesítmény okokból nem teljes a szeparáció.
- Kontrollált (OS által) módon megosztható a memória:
  - Közös kód lapok (read only, pl. DLL/SO).
  - Copy on Write (COW):
    - Szülő-gyermek kapcsolatban lévő folyamatok esetén.
    - A két folyamat először egy közös fizikai lapot használ.
    - Ha azt bármelyik írni próbálja, akkor a gyermeknek létrejön egy saját lap, az írási kísérlet előtti tartalommal.
  - Közösen olvasható és írható lapok.
    - UNIX System V Shared memory.
- HW (MMU) támogatás szükséges a megvalósításukhoz!

Megjegyzés

- A magyar nyelvű könyv 173. oldalán az újrahívhatósággal (reentrant code) kapcsolatos megjegyzés sajnos hibás.
  - Az itt megadott feltételek szükségesek az újrahívhatósághoz, de nem elégségesek.
    - Az újrahívhatóság elsősorban a függvény/metódus által használt adatszerkezetekre vonatkozik (kölcsönös kizárás).
  - Itt tulajdonképpen a könyv önmódosító kódról beszél (annak a használatát zárja ki).
  - Az ilyen osztott kód lapok mindig read-only hozzáféréssel kell hogy legyenek belapozva.
  - Ha nem, az súlyos biztonsági rés a rendszerben!!!
    - „Arbitrary code injection” and „privilege escalation”.

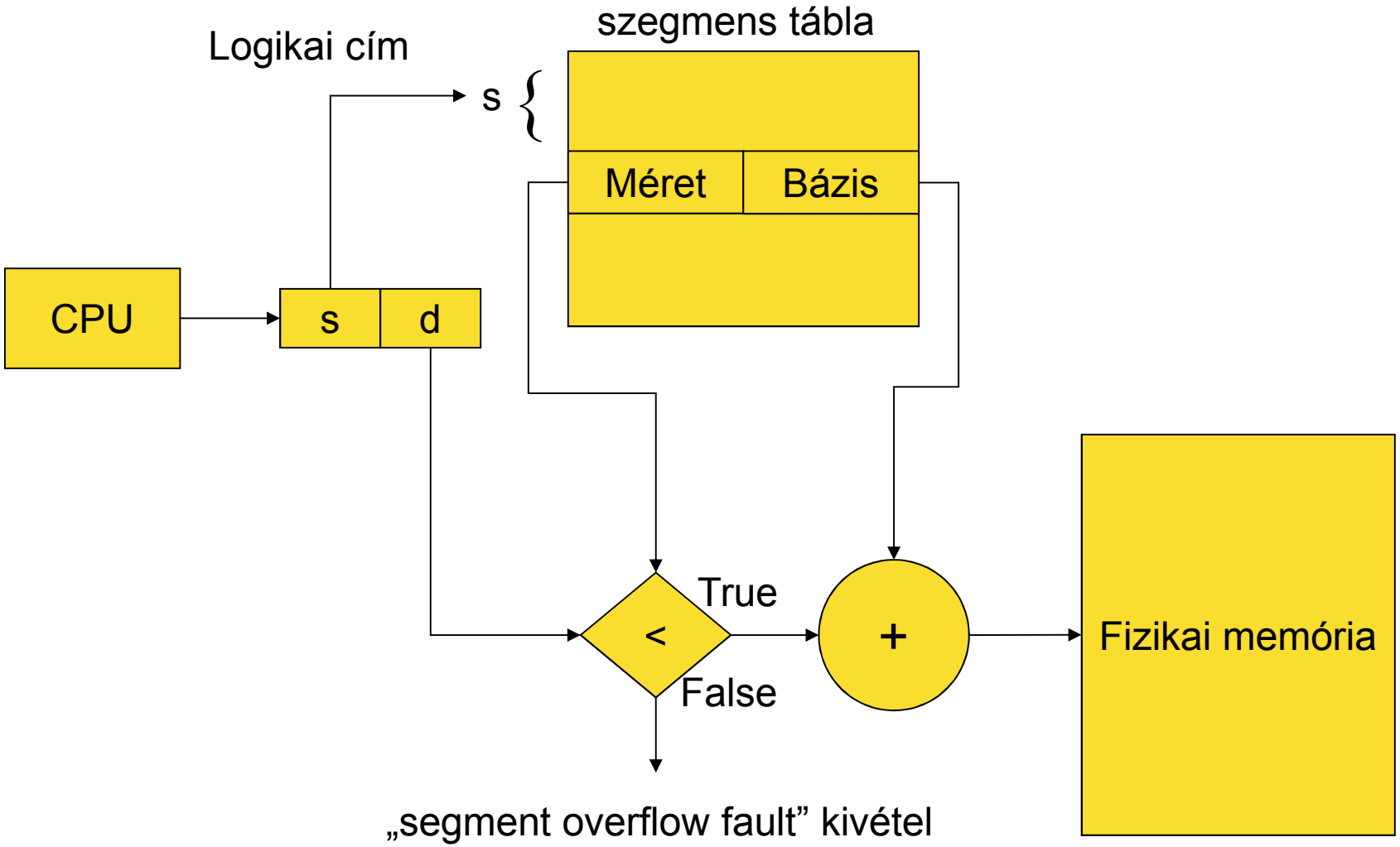
Szegmensszervezésű memória 1.

- A programozó:
  - Adat, kód, bizonyos programkönyvtárak, stack, heap, stb.
  - Nem egy összefüggő lineáris memória területben gondolkozik...
- A szegmensszervezésű (segmentation) memória ennek az elképzelésnek felel meg.
  - A logikai címtartomány szegmensekre van osztva.
  - A programozó egy szegmenst (segment name/ID) és azon belül egy szegmens ofszetet (segment offset) ad meg.
    - A lapozásnál egy címet ad meg, és azt a HW bontja ketté!
    - Itt két részt ad meg, és azt a HW rakja össze!

Szegmensszervezésű memória 2.

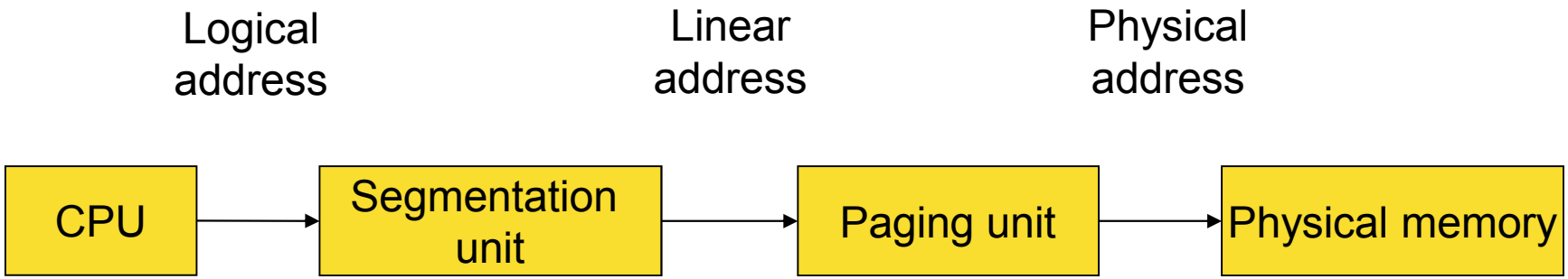
- A szegmensek mérete adott, azt is tárolni kell!
  - Adott szegmensen kívüli címzés esetén „segment overflow fault” kivétel.
- A szegmensek folytonosan tárolhatók, belső tördelődés nincs (meg lehet csinálni tördelődés nélkül).
- A szegmenseket a fordító és linker állítja össze, és adja meg a programot betöltő loader-nek.

Megvalósítás



Szegmens és lapszervezés együtt

- Egyes rendszerek (pl. PC) mind a kettőt támogatják.
- Az angol nyelvű könyvben a PC architektúra részletesen bemutatásra kerül (nem foglalkozunk vele idő hiányában).
- A szegmensszervezést minimálisan használják a x86 HW-ét támogató OS-ek.
  - Pl. Linux esetén 6 szegmens: kernel kód/adat, user kód/adat, Task state segment, default local descriptor table.
- A lapozás viszont alapvető CPU szolgáltatás.
  - x86 HW esetén 4KByte (2 szintű tábla) és 4Mbyte-os lapok (1 szintű tábla).
  - A Linux 3 szintű laptáblát használ, ebből a középső üres x86 HW esetén.



Virtuális tárkezelés (Virtual memory)

- Korábban beszéltünk virtuális címről (logikai cím ≠ fizikai cím).
  - Itt nem erről van szó.
- Ezt sokszor össze szokták keverni a swapping-gel.
  - Itt erről sincs szó.
- A korábban ismertetett memória menedzsment módszerek alapján kidolgozott komplex memória menedzsment megoldás a virtuális tárkezelés.

Alapgondolat, megfigyelések

- A folyamatok fizikai memóriában történő végrehajtása:
  - Szükségesnek és célszerűnek tűnik.
  - Ugyanakkor komoly és kellemetlen következményekkel jár.

1. A teljes folyamat nem szükséges annak a végrehajtásához, többnyire az aktuális utasítás számláló „környezete”, és az éppen használt adatszerkezetek elégségesek (lokalitás).
2. A folyamatok kódrészleteinek nagy részét soha nem hajtjuk végre vagy nagyon ritkán hajtjuk végre (error handling, software/feature bloat).
3. A folyamatok elindulásához nem szükséges a teljes program tényleges betöltése.
4. Egyes kódrészletek, erőforrások megoszthatóak a folyamatok között (pl. ugyan azt a kódot, adatot használják).
5. A párhuzamosan futó folyamatok egyes, már használt kódrészleteire sokáig vagy akár soha többé nincs szükségünk, míg más folyamatoknak nem elég a memória.

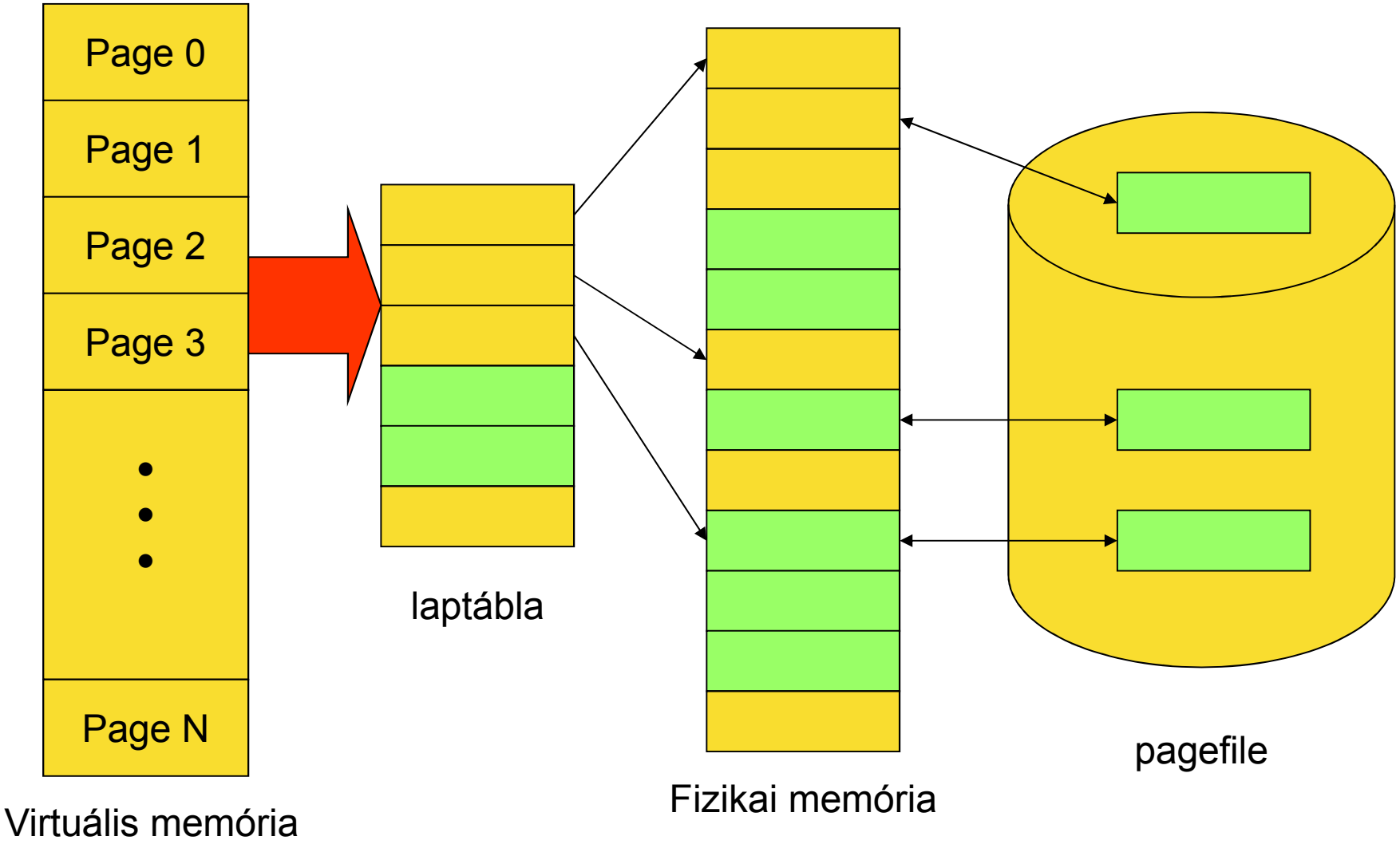
Elvárások

- Jó lenne nagyobb memóriát használó folyamatokat futtatni, mint amennyi fizikai memória rendelkezésre áll.
  - Virtuális memória, a programozónak nem kell foglalkoznia a rendelkezésre álló memóriával.
  - Persze ez egy architektúra függő határig lehetséges.
  - Vannak következményei: komplexitás és sebesség.
- Ha a folyamataink csak a tényleg szükséges fizikai memóriát tartják a fizikai memóriában, több folyamatot tudunk egy időben betölteni.
- A programok gyorsabban induljanak el, csak a szükséges kódrészleteket töltse be az operációs rendszer.
- Képesek legyenek osztozni a közös kód és adatszerkezeteken, erőforrásokon.

Virtuális tárkezelés

- Az alapja a lapozás.
  - A folytonos virtuális címteret egy tábla (memory map, page table, laptábla) képzi le.
- Nem direkt módon fizikai memóriára történik a leképzés, hanem:
  - Részben fizikai memóriára.
  - Részben a permanens táron (HDD, Flash tár) kialakított speciális területre.
    - Pagefile (Windows), vagy swap file (UNIX/Linux), a UNIX/Linux esetén a név történelmi örökség, nem swapping-ről van szó.
- A folyamat egy összefüggő virtuális címteret lát!

Virtuális tárkezelés ábra

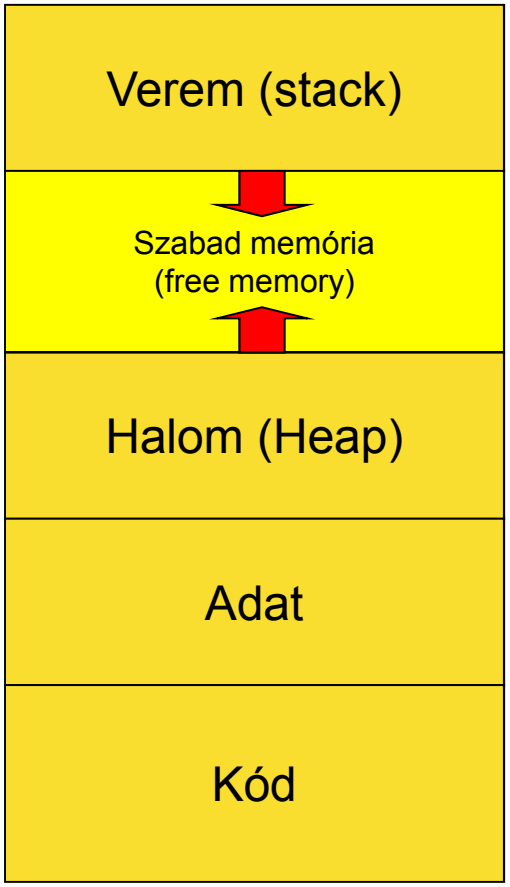


Egyéb tulajdonságok

- Folyamatok megoszthatnak memóriaterületeket olvasás vagy akár írás és olvasás hozzáféréssel.
  - Hozzáférési jogosultságok.
  - Az ilyen memória területek több folyamat virtuális címtartományába vannak belapozva.
- Módosítás nyilvántartása (modified/dirty bit):
  - Minden laphoz tartozik egy HW által kezelt bit (pl. a laptáblában).
    - Betöltéskor törlik, módosításkor beállítják.
- Hivatkozások nyilvántartása (referenced/used bit):
  - OS adott időnként és/vagy adott eseményekre törli.
  - Használat esetén beállítják.

Következmények

- A folyamat virtuálisan összefüggő memóriát lát (virtuális memória).
- Valójában az összefüggő memóriaterület ritka (sparse address space).
  - Nagy része mögött nincs fizikai memória vagy pagefile bejegyzés.
  - Ha az szükséges, akkor dinamikusan kerül mögé fizikai memória, amit a folyamat elérhet.



Folyamat a memóriában

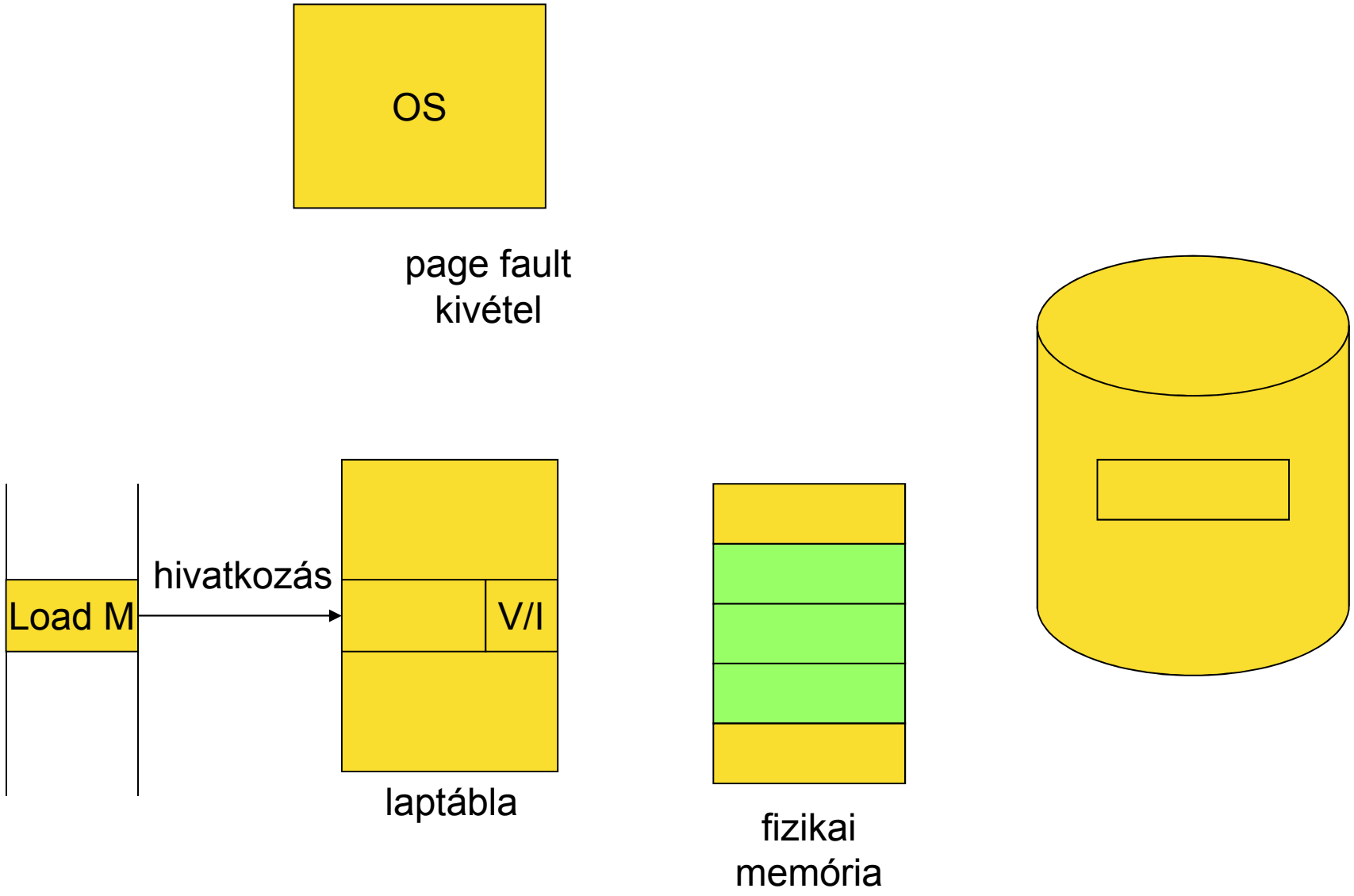
Működés

- Ha egy éppen használt (pl. egy utasításban) virtuális memória lapon található cím bent van a fizikai memóriában, akkor a kód végrehajtható.
- Ha nincs benn (valid/invalid bit)?
  - Laphiba („page fault”) kivételt generál az MMU.
    - Érvényes, de éppen nem fizikai memóriában lévő lap.
    - Nem hiba, normális működés!
  - Az operációs rendszer ezt kezeli, lehetőségek:
    - HDD-re ki van írva: be kell hozni a pagefile-ból.
    - Soha nem lett betöltve: be kell tölteni.
    - Kérdés: Hová, főleg ha tele van a fizikai memória?
  - Az OS visszaadhatja a vezérlést a megszakított folyamatnak.
  - A hozzárendelés során a folyamat passzívan várakozik.

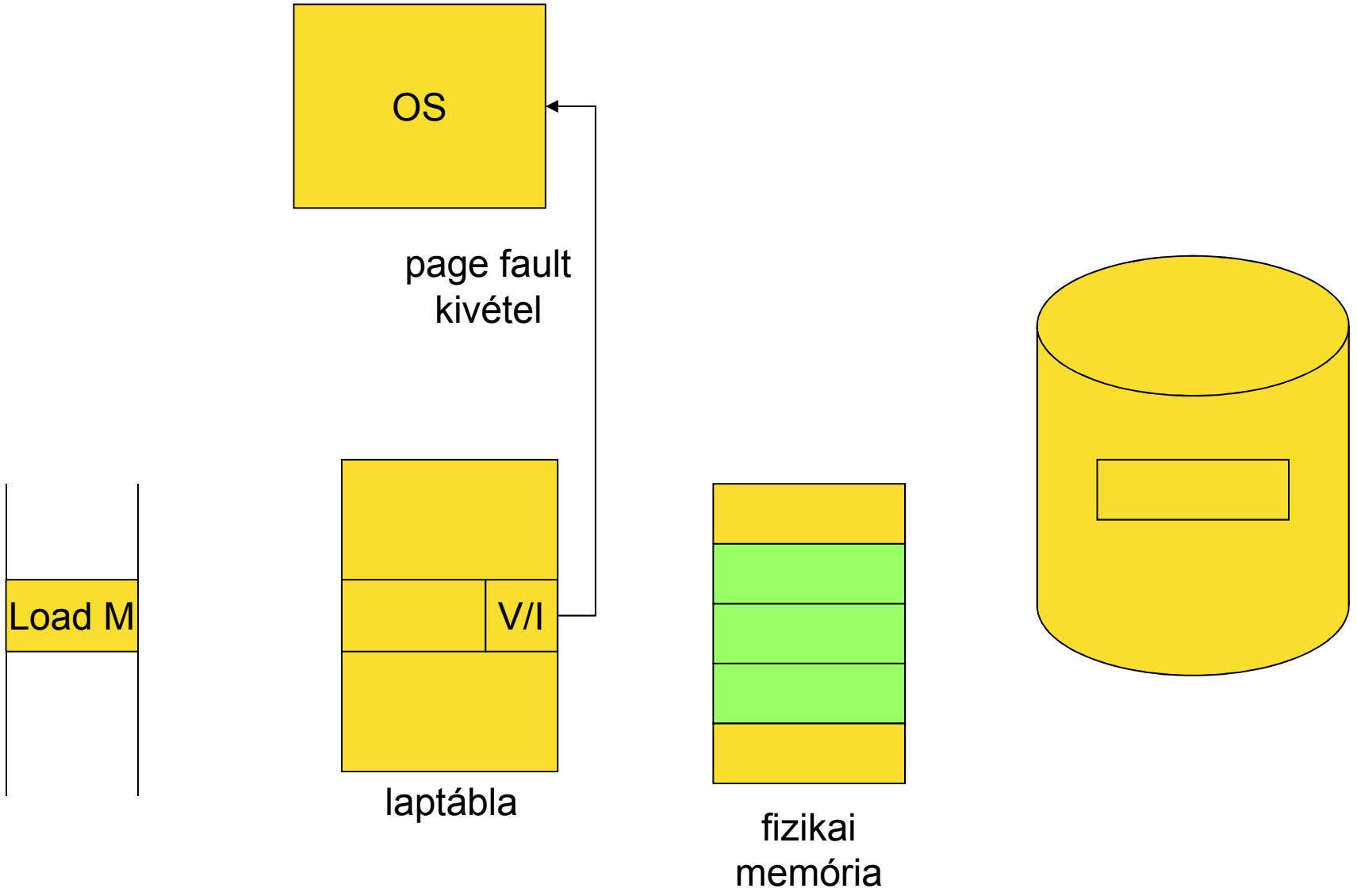
Lapozási stratégiák (fetch strategy)

- Igény szerinti lapozás (demand paging):
  - Csak laphiba esetén, és csak a laphibát megszüntető lapot hozza be a fizikai memóriába.
  - Csak a szükséges lapok vannak a fizikai memóriában.
  - Új lapra vonatkozó hivatkozás mindig hosszú várakozást eredményez (be kell hozni).
- Előretekintő lapozás (anticipatory paging):
  - Előre tekintve (becslés) az OS megpróbálja kitalálni, hogy mely lapokra lesz szükség, és azokat is behozza.
  - Feltétel: szabad erőforrások (CPU, HDD, fizikai memória).
  - Ha a behozott lapokra tényleg szükség lesz (sikeres a becslés), akkor csökken a laphibák száma.
  - Ha nem, akkor feleslegesen használunk erőforrásokat.

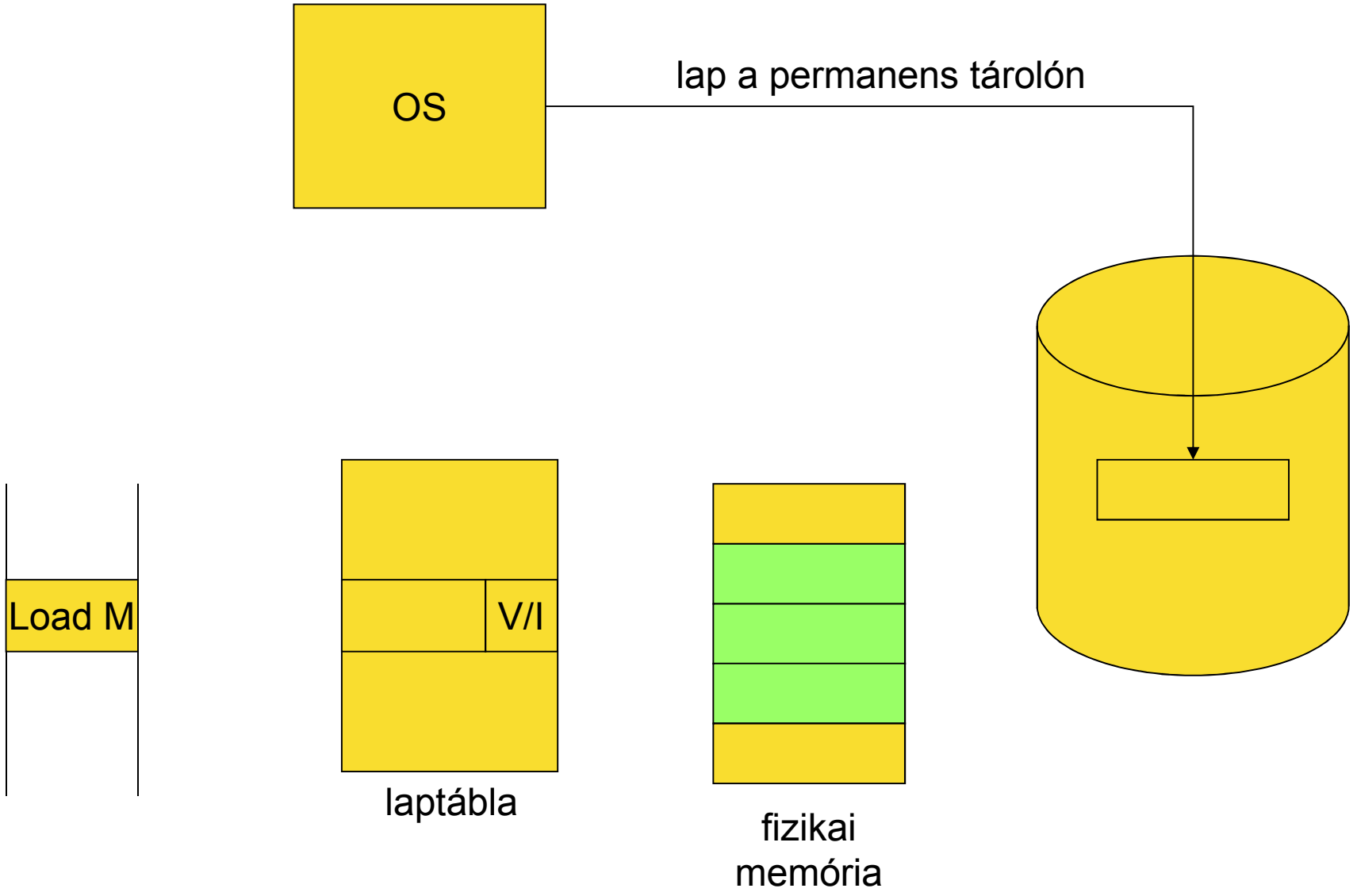
Működés laphiba esetén 1.



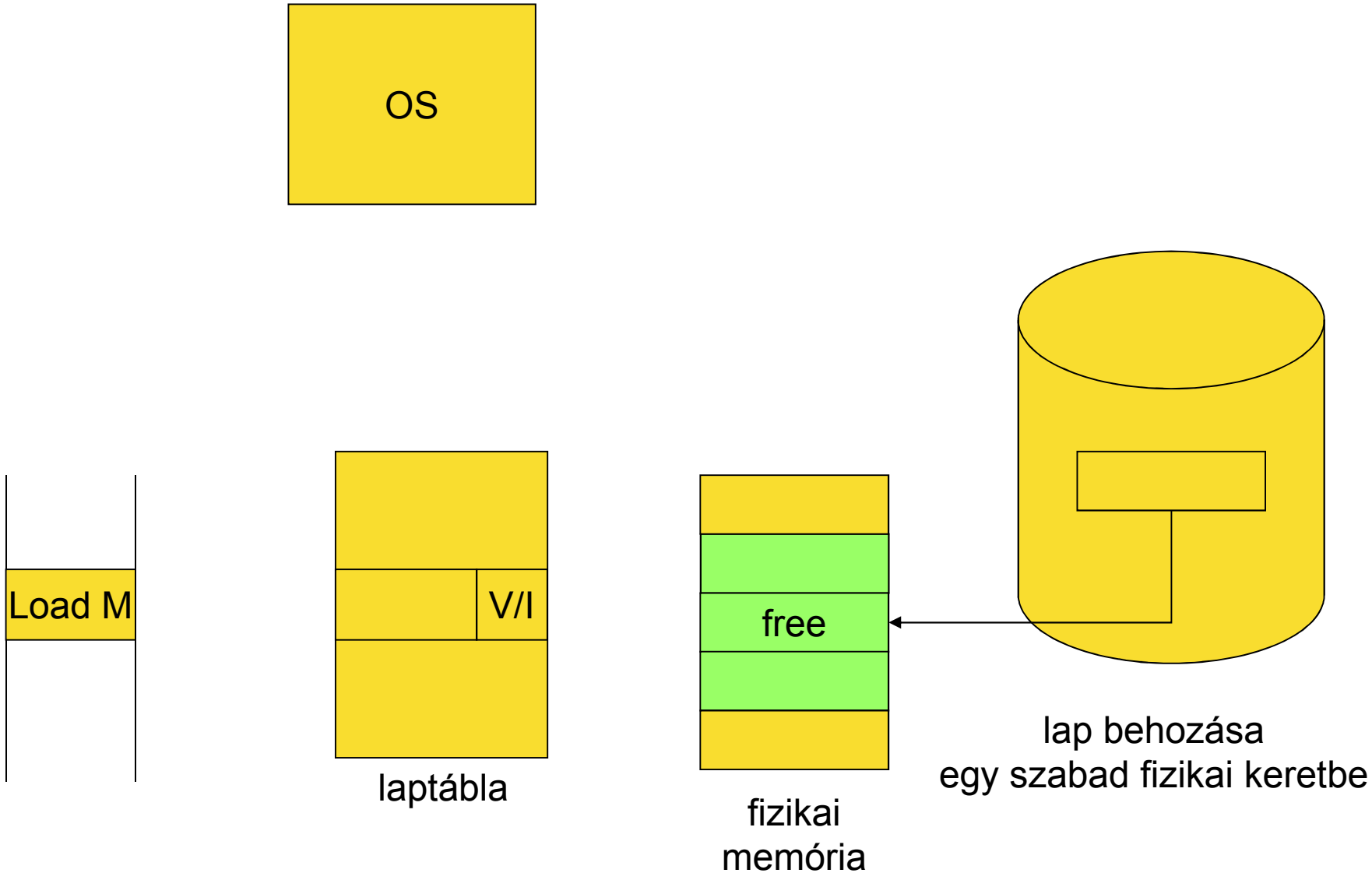
Működés laphiba esetén 2.



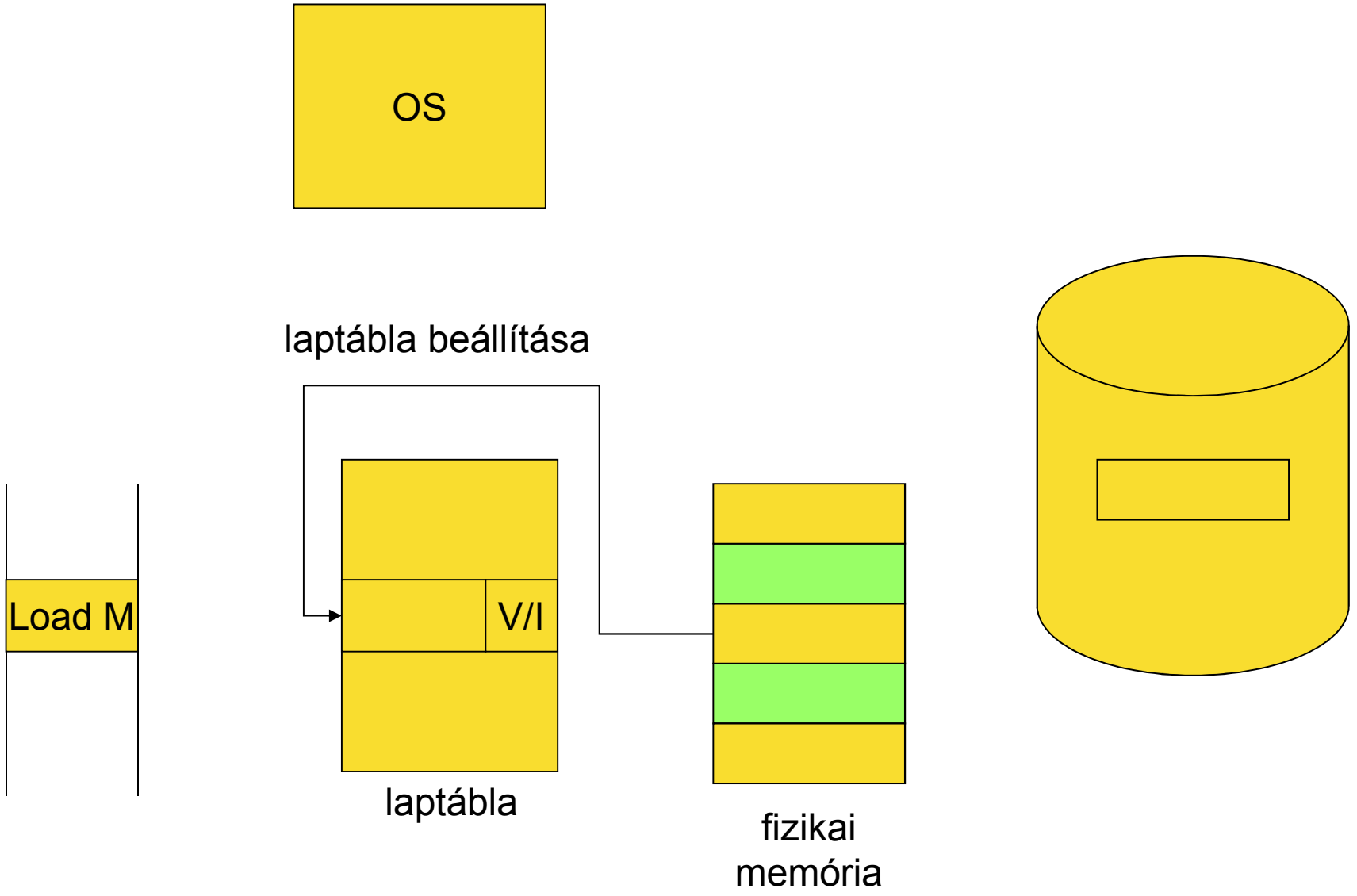
Működés laphiba esetén 3.



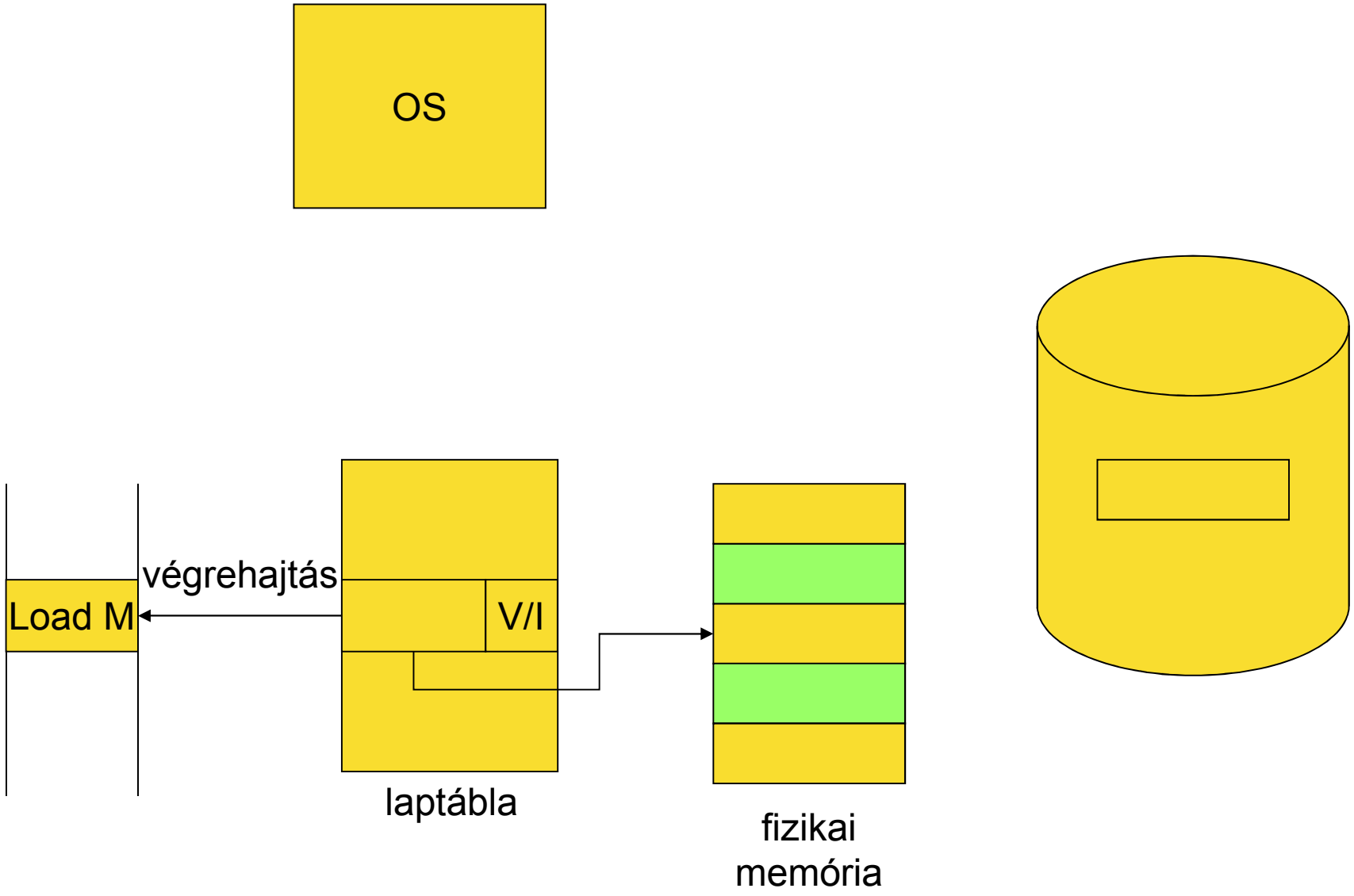
Működés laphiba esetén 4.



Működés laphiba esetén 5.



Működés laphiba esetén 6.



Lapcsere stratégiák (replacement strategies)

- Tele van a fizikai memória, és laphiba történik.
  - Ki kell választani a permanens tárra kikerülő lapot.
    - Áldozat kiválasztás (victim selection).
  - Ennek a helyére kerül be a lap a permanens tárról.
- Algoritmusok:
  - Optimális algoritmus.
  - Legrégebbi lap (FIFO).
  - Újabb esély algoritmus (Second chance).
  - Legrégebben nem használt (Least recently used, LRU).
  - Legkevésbé használt (Least frequently used, LFU).
  - Utóbbi időben nem használt (Not recently used, NRU).

Optimális algoritmus

- Előre néz, és teljes információval rendelkezik a jövőben használt lapokról.
  - Nem realizálható, de jó összehasonlítási alap.

FIFO

- A behozott lapokat egy FIFO-ba rendezi, és mindig a legrégebben behozott lapot cseréli le.
  - Egyszerű, a múlt alapján dönt, hátranéz.
  - Azokat a lapokat is lecseréli, amelyeket a folyamatok gyakran használnak.
    - Nem nézi a tényleges használatot.
    - Csak a behozás sorrendjét.
  - Bélády anomália (Bélády László, BME hallgató volt, 56-ban hagyta el az országot, az IBM-nél dolgozott):
    - Növeljük a folyamatnak adott memória keretek számát (fizikai memória).
    - Nő a laphibák gyakorisága.
    - Anomális: Nem úgy működik, ahogy várnánk...

Újabb esély algoritmus, SC

- A sor elején lévő lapot csak akkor cserélik le, ha arra nem hivatkoztak (referenced/used bit).
  - A referenced bitet az MMU állítja be, ha a lapot használják!
  - A reference bit-et törli, ha az be van állítva:
    - Ezért újabb esély a neve.
    - Ezek után a lapot a sor végére rakja (FIFO jellegű a sor ebben az esetben is).
    - Egyébként végtelen ciklusba kerülhetne!
  - Bonyolultabb, mint a FIFO, de alapvetően egyszerű algoritmus.
  - Hátra néz, és a behozás sorrendje, és a használat alapján dönt.
    - Figyelembe veszi a program lokalitását.

Legrégebben nem használt, LRU

- Megoldás:
  - Bonyolult, de jól közelíti az optimális algoritmust.
    - A lokalitás miatt jó a közelítés.
  - Hátrafelé néz.
  - Megvalósítás:
    - Számláló: Minden laphoz egy „last used” timestamp kerül.
    - Láncolt lista: A lista végére kerül a legutoljára használt.
    - Kétdimenziós tömb: NxN-es mátrix, ahol N a lapok száma.
  - Sokszor a közelítéseit szokták használni.

Legkevésbé használt, LFU

- A közelmúltban gyakran használt lapokat a lokalitás miatt nagy valószínűséggel újra fogjuk használni.
  - A ritkán használtakat kis valószínűséggel fogjuk használni.
  - Az R bit értékét hozzáadja időnként egy laphoz tartozó számlálóhoz, és törli az R bitet.
  - A kisebb számláló értéket tartalmazó lapokat cseréljük le.
  - Az algoritmus nem felejt...
  - Az algoritmus a frissen behozott lapokat fogja lecserélni (0 vagy kicsi számláló érték).
    - Azokat be kell fagyasztani a memóriába egy időre.

Utóbbi időben nem használt, NRU

- A hivatkozott és módosított biteket is használja.
- R törölhető, M-et viszont meg kell őrizni.
- Prioritást rendel a lapokhoz R és M alapján.
  - 0. prioritás: R=0, M=0 (legalacsonyabb)
  - 1. prioritás: R=0, M=1
  - 2. prioritás: R=1, M=0
  - 3. prioritás: R=1, M=1 (legmagasabb)
- Mindig a legkisebb prioritású csoportból választ, ahol még van lap.

Lapcsere szempontok

- Globális lapcsere: A teljes fizikai memória potenciálisan lecserélhető.
- Lokális lapcsere: A folyamat által használt fizikai memória lapok között történik a csere.
- Lapok tárba fagyasztása (lock bit):
  - I/O műveletek hivatkoznak rá.
    - Ott fizikai címet kell használnunk!
    - Lehet kernel szinten pufferelni, és akkor ez is megkerülhető.
  - LFU algoritmus frissen behozott lapjai (nem voltak használva, elsőrangú áldozatok).

Példa

- Az algoritmusokkal ismerkedjünk meg egy példán keresztül.
- Ilyen is lehet ZH-án vagy vizsgán.

Példa

Egy igény szerinti lapozást használó rendszerben 3 vagy 4 fizikai memórialap áll egy folyamat rendelkezésére. A futás folyamán sorban a következő virtuális lapokra történik hivatkozás:

0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4, 0, 1

Hány laphiba következik be a rendszerben a következő algoritmusok esetén, ha kezdetben a 3 vagy 4 lap üres?

- Legrégebbi lap (FIFO) algoritmus alkalmazásánál 3 vagy 4 fizikai memória lap esetén
- Legrégebben nem használt (LRU) algoritmus alkalmazásánál 3 és 4 fizikai memória lap esetén
- Újabb esély (SC) algoritmus alkalmazásánál 3 és 4 fizikai memória lap esetén

- Röviden magyarázza meg az eredményeket!

FIFO megoldás 3 memória kerettel

Laphivatkozások	0	1	2	3	0	1	4	0	1	2	3	4	0	1
FIFO 3 memória kerettel	0	1	2	3	0	1	4	4	4	2	3	3	0	1
		0	1	2	3	0	1	1	1	4	2	2	3	0
			0	1	2	3	0	0	0	1	4	4	2	3
Laphibák	i	i	i	i	i	i	i			i	i		i	i

FIFO megoldás 4 memória kerettel

Laphivatkozások	0	1	2	3	0	1	4	0	1	2	3	4	0	1
FIFO 4 memória kerettel	0	1	2	3	3	3	4	0	1	2	3	4	0	1
		0	1	2	2	2	3	4	0	1	2	3	4	0
			0	1	1	1	2	3	4	0	1	2	3	4
				0	0	0	1	2	3	4	0	1	2	3
Laphibák	i	i	i	i			i	i	i	i	i	i	i	i

LRU megoldás 3 memória kerettel

Laphivatkozások		0	1	2	3	0	1	4	0	1	2	3	4	0	1
LRU 3 memória kerettel		0 1	0 2	0 3	3 1	3 2	3 3	4 1	4 2	4 3	2 1	2 2	2 3	0 1	0 2
			1 1	1 2	1 3	0 1	0 2	0 3	0 1	0 2	0 3	3 1	3 2	3 3	1 1
				2 1	2 2	2 3	1 1	1 2	1 3	1 1	1 2	1 3	4 1	4 2	4 3
Laphibák			i	i	i	i	i	i	i			i	i	i	i	i

LRU megoldás 4 memória kerettel

Laphivatkozások	0	1	2	3	0	1	4	0	1	2	3	4	0	1
LRU 4 memória kerettel	0 1	0 2	0 3	0 4	0 1	0 2	0 3	0 1	0 2	0 3	0 4	4 1	4 2	4 3
		1 1	1 2	1 3	1 4	1 1	1 2	1 3	1 1	1 2	1 3	1 4	0 1	0 2
			2 1	2 2	2 3	2 4	4 1	4 2	4 3	4 4	3 1	3 2	3 3	3 4
				3 1	3 2	3 3	3 4	3 5	3 6	2 1	2 2	2 3	2 4	1 1
Laphibák	i	i	i	i			i			i	i	i	i	i

SC megoldás 3 memória kerettel

Laphivatkozások	0	1	2	3	0	1	4	0	1	2	3	4	0	1
SC 3 memória kerettel	0 y	1 y	2 y	3 y	0 y	1 y	4 y	4 y	4 y	2 y	3 y	3 y	0 y	1 y
		0 y	1 y	2 n	3 y	0 y	1 n	1 n	1 y	4 n	2 y	2 y	3 n	0 y
			0 y	1 n	2 n	3 y	0 n	0 y	0 y	1 n	4 n	4 y	2 n	3 n
Laphibák	i	i	i	i	i	i	i			i	i		i	i

Igény szerinti lapozás, behozza és hozzá is fér!

SC megoldás 4 memória kerettel

Laphivatkozások	0	1	2	3	0	1	4	0	1	2	3	4	0	1
SC 4 memória kerettel	0 y	1 y	2 y	3 y	3 y	3 y	4 y	0 y	1 y	2 y	3 y	4 y	0 y	1 y
		0 y	1 y	2 y	2 y	2 y	3 n	4 y	0 y	1 y	2 n	3 y	4 y	0 y
			0 y	1 y	1 y	1 y	2 n	3 n	4 y	0 y	1 n	2 n	3 y	4 y
				0 y	0 y	0 y	1 n	2 n	3 n	4 y	0 n	1 n	2 n	3 y
Laphibák		i	i	i	i			i	i	i	i	i	i	i	i

Példa értékelése

- FIFO és SC 3 és 4 memória kerettel:
  - 4 keretre rosszabb, mint 3 keretre.
  - Bélády anomália.
- LRU 3 keretre rosszabb, mint FIFO v. SC 3 keretre:
  - A lapsorozatra nem teljesül a lokalitás feltétel, ezért rosszabb.
- 4 keret esetén a legjobb az LRU.
- Statisztikai vizsgálatok alapján lehet értékelni az algoritmusokat.
  - Ezek a számok úgy lettek kitalálva, hogy megjelenjenek a jellegzetességek.

Virtuális memória teljesítménye 1.

- Fizikai memória sebessége:
  - n*1 Gbyte/s adatátviteli sebesség.
  - n*10 ns késleltetés.
  - Ha cache-elve van, akkor még gyorsabb...
- Permanens tároló sebessége:
  - Tipikusan 100 Mbyte adatátviteli sebesség, de random elérés esetén és sok párhuzamos felhasználó esetén nagyságrendekkel lassabb.
  - 10 ms legrosszabb hozzáférési idő (fejmozgás).
  - Flash esetén az írás (törlés) lassú és problémás.
    - Hamar tönkremegy (pagefile-t sokat írja a rendszer)
  - Több nagyságrend a különbség.

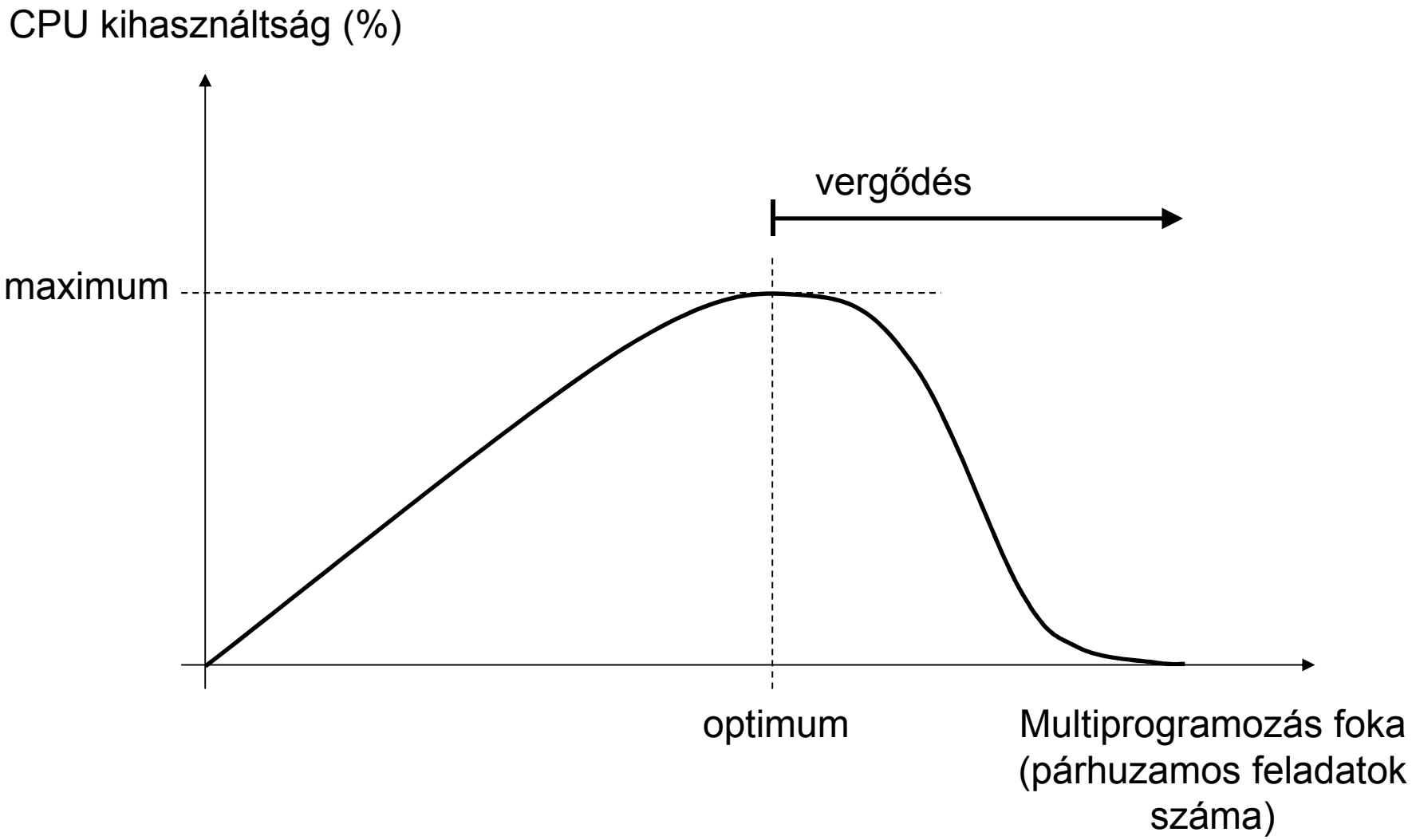
Virtuális memória teljesítménye 2.

- Ha egy lap nincs bent a fizikai memóriában:
  - Több nagyságrenddel lassabb a permanens tárolón lévő lapok elérése (betöltése), mint egy fizikai memóriában lévő lap elérése.
- Az elérhető sebességet alapvetően a laphibák gyakorisága befolyásolja.
  - A gyakori laphibák a rendszer teljesítményét drasztikusan csökkenthetik...
  - Sikeres előretekintő lapozás javíthat a teljesítményen.

Vergődés (Thrashing)

- Hogyan válasszuk meg, hogy egy folyamathoz hány memória keretet rendeljünk a fizikai memóriában?
  - Kevés: Nagyszámú/állandó laphiba (trashing).
  - Sok: Más feladatnak nem marad memória.
- DEF: A gyakori laphibák által okozott rendszer teljesítmény csökkenést vergődésnek (trashing) nevezzük.
  - A laphiba kezelése során újabb laphiba jelenik meg.
  - A laphibák felgyűlnek a háttértár várakozási sorában.
  - A CPU a laphibák megoldására vár.
  - A hosszú távú ütemező (ha van), ezt I/O intenzív folyamatként is értelmezheti, újabb folyamatokat beengedve a rendszerbe...
    - A helyzet még rosszabbá válhat.

Vergődés (ábra)



Vergődés elkerülése

- Cél: alacsony laphiba gyakoriság (Page Fault Frequency, PFF).
- Egy laphiba kezelése során ne jöjjön létre újabb laphiba:
  - A háttértár várakozási sora csak csökkenhet...
- Lokális lapcsere stratégia:
  - A folyamatok nem tudják egymástól elvenni a fizikai memória kereteket.
  - Nem terjedhet át a hatás más feladatokra.
  - Csökkenti a problémát, de nem oldja meg.
- Hány keretre van szüksége a fizikai memóriában egy folyamatnak a hatékony futáshoz?

Lokalitás

- Statisztika: Egy időintervallumban a folyamatok a címtartományuk csak egy kis részét használják.
  - Időbeli.
  - Térbéli.
- Vergődés:
  - Megfelelő számú fizikai memória keret allokálása.
    - Nincs vergődés.
    - Laphibák a lokalitás váltásakor.
  - Ennél kisebb: vergődés

Munkahalmaz (Working-set)

- A lokalitáson alapul.
- A folyamat azon lapjainak halmaza, amelyekre egy időintervallumban (munkahalmaz-ablak) a folyamat hivatkozik.
- A munkahalmaz alapján megadható az adott folyamat munkahalmaz mérete (WSS).
- A futó folyamatok teljes fizikai memória keret igénye számítható (D).

$$D = \sum WSS_i$$

Alkalmazása

- Az OS méri a WSS-t folyamatonként.
  - Ha van szabad memória keret:
    - Akkor az igények kielégíthetőek.
    - Új folyamatok engedhetőek be a rendszerbe.
  - Ha nincs szabad memóriakeret:
    - Akkor ki kell választani egy „áldozat” folyamatot.
    - Azt fel kell függeszteni (suspend, tényleges swap out).
    - Az áldozat folyamat fizikai memória kereteit fel lehet használni.
- A vergődés elkerülhető a multiprogramozás fokát közel optimálisan tartva.
- A gyakorlatban erőforrás igényes a megvalósítása:
  - Más megoldást kell találni...

PFF alapú optimalizáció

- Vergődés: magas PFF érték...
- Folyamatonként egyszerűen mérhető a PFF.
  - Alacsony: túl sok fizikai memória keret.
  - Magas: túl kevés memória keret.
- Felső és alsó PFF határérték megadása.
  - Felső határérték túllépés: kap egy fizikai memória keretet.
  - Alsó határérték túllépése: egy fizikai memória keret elvonása.
    - Esetleg csak akkor, ha nincs szabad fizikai memória keret a rendszerben.
    - Ha nincs fizikai memória: egy folyamat felfüggesztése

Beágyazott rendszerek

- A beágyazott rendszerekben nagyon eltérő a memória kezelése (nem foglalkozunk vele).
  - A beágyazott rendszerek CPU-inak egy részében nincs MMU (vagy nem használják):
    - Lapozás, virtuális memória kezelés, szeparáció, stb. eleve ki van zárva.
  - Ha van MMU, és használják:
    - A háttértár korlátos, a pagefile-nak nincs hely sem.
    - Pl. virtuális memória alkalmazása valós idejű rendszerekben gyakorlatilag kizárható/értelmetlen.
      - Laphiba esetén a végrehajtás idejére hogyan adható felső korlát?
    - Elsősorban a folyamatok szeparációja és virtualizáció a feladat.