Operációs rendszerek I.

Virtuális tárkezelés

Várkonyiné Kóczy Annamária
Professzor
Informatika Tanszék
(koczya@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**

7. Virtuális tárkezelés

- Bevezetés
- A virtuális tárkezelés általános elvei
- Lapcsere stratégiák
- Folyamatok lapigénye, lapok allokációja
- Egyéb tervezési szempontok

Bevezetés

- Olyan szervezési elvek, algoritmusok összessége, amelyek biztosítják, hogy a rendszer folyamatai logikai címtartományainak csak egy - *a folyamat futásához szükséges* - része legyen a központi tárban.
- Korábbi „helytakarékos módszerek”:
  - késleltetett betöltés,
  - tárcserék, stb.

  Ezek nem voltak általános megoldások.

7.1. A virtuális tárkezelés általános elvei

- Motiváció
- A megvalósítás elve
- Hardver feltételek
- Futási sebesség
- Betöltendő lapok kiválasztása

Motiváció 1

- A programok nem használják ki a teljes címtartományukat
  - tartalmaznak ritkán használt kódrészleteket (pl. hibakezelő rutinok)
  - a statikus adatszerkezetek általában túlméretezettek (statikus vektorok táblák stb.).
  - a program futásához egy időben nem kell minden részlet (overlay).
  - Időben egymáshoz közeli utasítások és adatok általában a térben is egymáshoz közel helyezkednek el (lokalitás).

Motiváció 2

- Nem célszerű a egész programot a tárban tartani:
  - programok méretét nem korlátozza a tár nagysága, a program nagyobb lehet mint a ténylegesen meglévő memória mérete.
  - a memóriában tartott folyamatok száma növelhető (nő a multiprogramozás foka, javítható a CPU kihasználtság és az átbocsátó képesség)
  - a programok betöltéséhez, háttértárba mentéséhez kevesebb I/O művelet kell.

A megvalósítás elve 1.

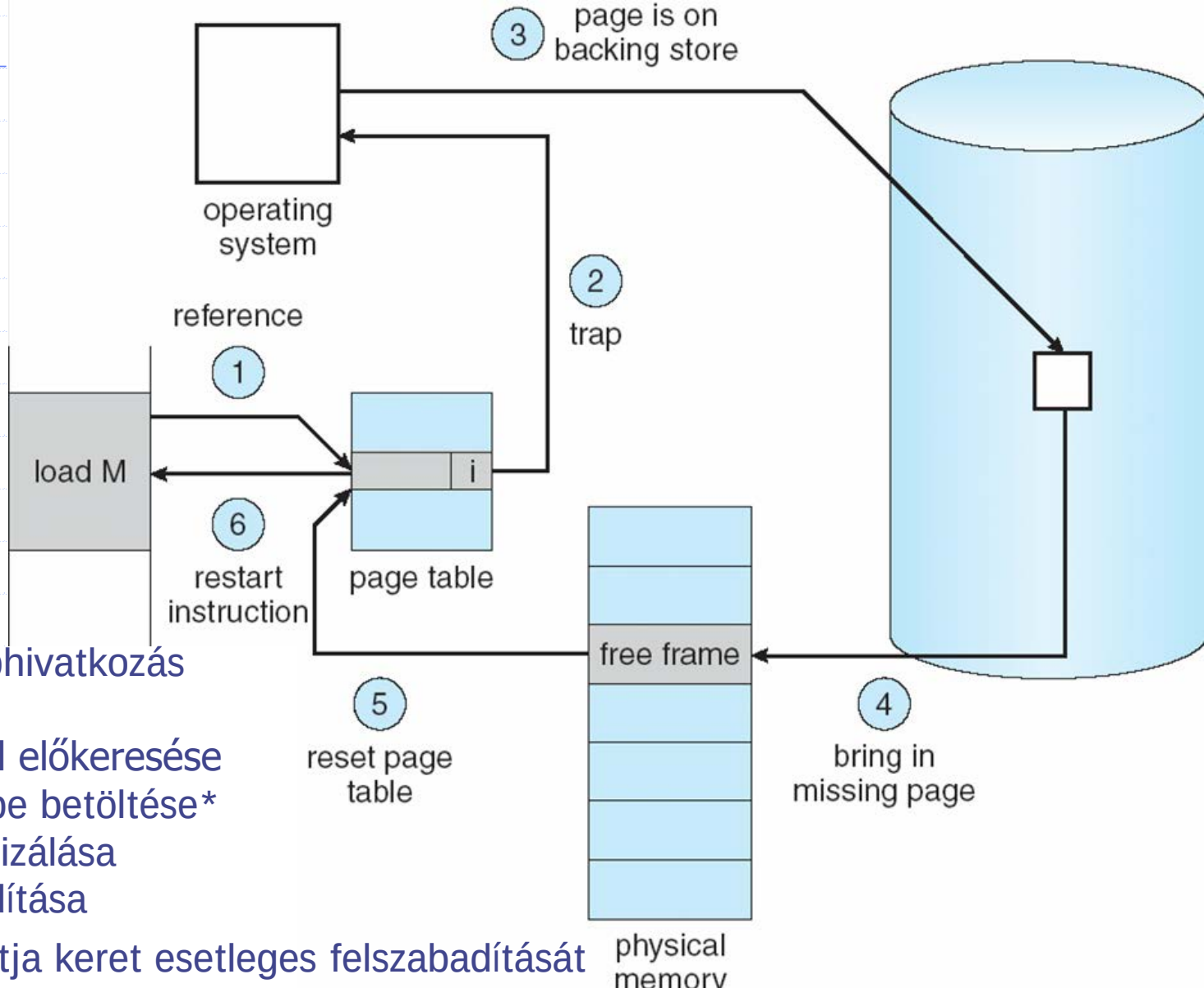
- Amikor a folyamat *érvénytelen*, a memóriában nem található címre hivatkozik, hardver megszakítást okoz, ezt az OS kezeli, és behozza a kívánt blokkot.
- Lépései:
- Az OS megszakítást kezelő programrésze kapja meg a vezérlést (1,2)
  - elmenti a folyamat környezetét
  - elágazik a megfelelő kiszolgáló rutinra
  - eldönti, hogy a megszakítás nem programhiba-e (pl. kicímzés)

A megvalósítás elve 2.

- A kívánt blokk beolvasása a tárba (4,5)
  - Az OS a blokknak helyet keres a tárban; ha nincs szabad terület, akkor fel kell szabadítani egy megfelelő méretű címtartományt, ennek tartalmát esetleg a háttértárra mentve
  - Beolvassa a kívánt blokkot

  Megjegyzés: A perifériás műveletek sok időt vesznek igénybe, ki kell várni az eszköz felszabadulását, az előkészítő műveletet (fejmozgás) és az átvitelt. A rendszer jobb kihasználtsága érdekében a megszakított folyamatot az OS várakozó állapotba helyezi, és egyéb folyamatot indít el. Amikor a kívánt blokk bekerül a tárba, a folyamat futás kész lesz.
- A folyamat újra végrehajtja a megszakított utasítást (6)

A megvalósítás elve 3.



1. Érvénytelen laphivatkozás
2. Megszakítás
3. Lap háttértárról előkeresése
4. Lap üres keretbe betöltése*
5. Laptábla aktualizálása
6. Utasítás újraindítása

* Az ábra nem mutatja keret esetleges felszabadítását

Példa: A rendszer állapota

0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H

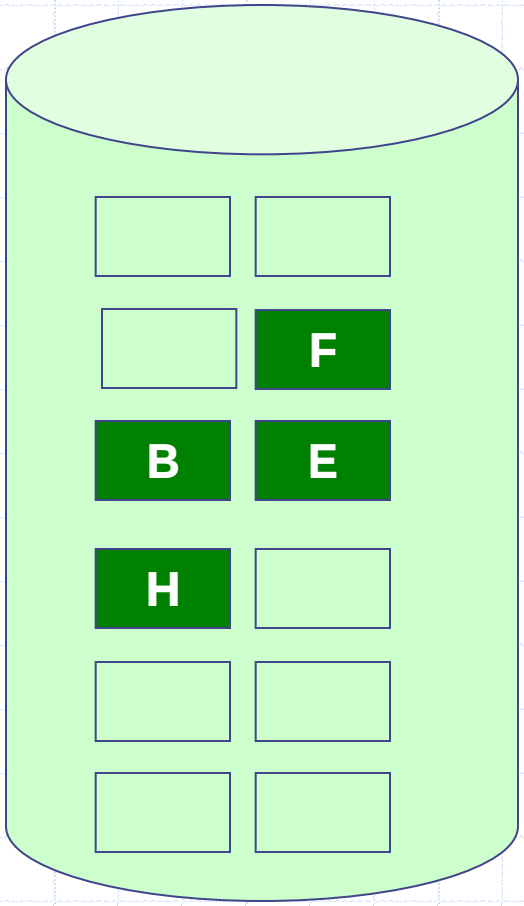
Logikai címtér

	p	p'	v/i
0	1	v
1		i
2	6	v
3	4	v
4		i
5		i
6	9	v
7		i

laptábla

0	
1	A
2	
3	
4	D
5	
6	C
7	
8	
9	G
10	
11	
12	

fizikai memória



háttértár

Az érvénytelen frame-indexek helyén a háttértár megfelelő pontját azonosító címek találhatók

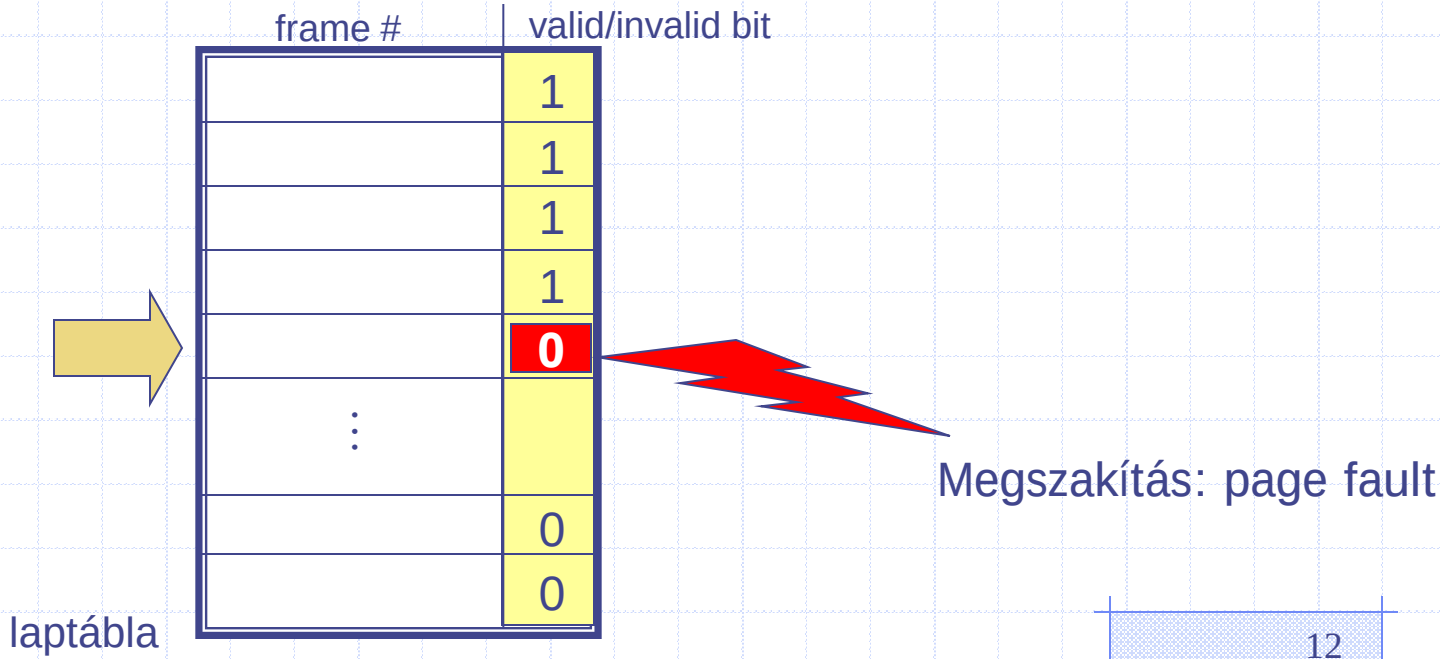
Hardver feltételek

- Az érvénytelen címhivatkozás megszakítást okozzon
- A megszakított utasítás újraindítható legyen.

Megjegyzés: A korszerű rendszerek lap- vagy kombinált szervezésűek, ezért itt virtuális tár kezelés mindig lapok mozgatásán alapul.

Az érvénytelen címhivatkozás kezelése

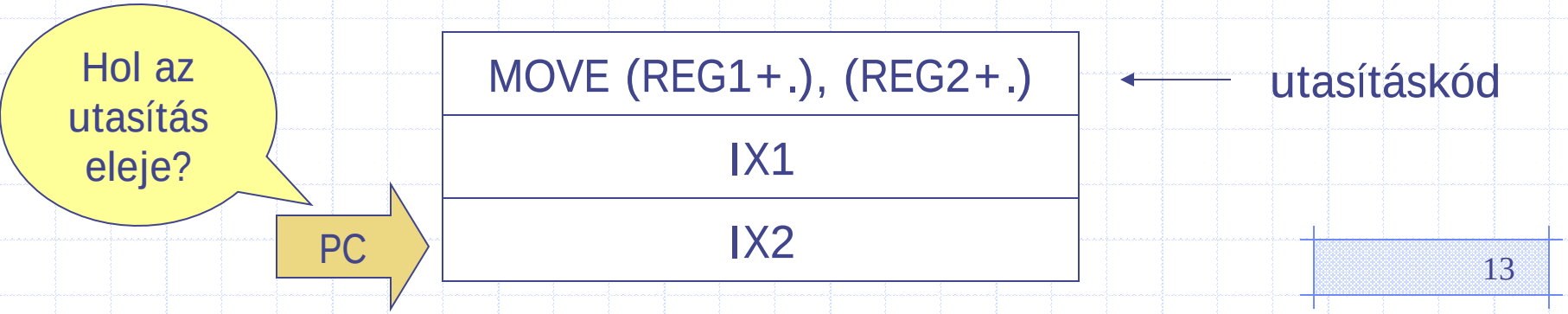
- Emlékeztető: a laptábla mindig tartalmaz egy érvényesség bitet:
  - valid: a lap a tárban van
  - invalid: a lap nincs a tárban



Utasítások újraindíthatósága 1.

- A laphiba lekezelése után az utolsó utasítást újra kell indítani...
  - az utasítás elejétől. Ez több szavas utasítások esetén gond lehet. (Mire a megszakítás bekövetkezett, a PC már rég nem az utasítás elejére mutat...)
  - Pl.: 3 szavas utasítás

    MOVE (REG1+IX1), (REG2+IX2)



Utasítások újraindíthatósága 2.

- A laphiba lekezelése után az utolsó utasítást újra kell indítani…
  - de az esetleges mellékhatásokat az újraindítás előtt meg kell szüntetni.
  - Pl.: auto inkremens utasítás végrehajtása során az inc bekövetkezett-e már? Ha igen, akkor dec kell az újraindítás előtt.
- Csak HW támogatással lehet hatékonyan megoldani:
  - rejtett PC másolat (utasítás elejét mutatja)
  - auto inc/dec flagek jelzik, hogy végrehajtódott-e már

Laphiba hatása a folyamatok futásának sebességére

- Effektív hozzáférési idő =

  (1 - *p*) * memória hozzáférési idő + *p* * laphiba idő

- *p* a laphiba gyakorisága
- Mivel a laphiba kiszolgálása 5 nagyságrenddel nagyobb lehet, így *p*-nek kicsinek kell lennie.

Példa

Effektív hozzáférési idő (EAT) =

(1 - *p*) * memória hozzáférési idő + *p* * laphiba idő

- memória hozzáférési idő = 1 µs
- Egy lap kiírási/beolvasási ideje (swap in/out): 10ms
- A helyettesítendő lapok 50%-a módosult, tehát ezeket a háttértárra ki is kell írni (swap out).
  → átlagos swap idő: 15ms

- EAT = (1 – *p*) x 1 µs + *p* x 15000 µs
  ~ 15000*p* (µs)

Alapvető kérdések

- A betöltendő blokk (lap) kiválasztása (*fetch*)
- A behozott blokk a valós tárba hova kerüljön (*placement*)
- Ha nincs hely, akkor melyik blokkot tegyük ki (*replacement strategy*)
- Hogyan gazdálkodjunk a fizikai tárral, azaz egy folyamat számára hány lapot biztosítsunk.

A betöltendő lap kiválasztása 1.

Igény szerinti lapozás (demand pages)

Előnyei:

- egyszerű a lapot kiválasztani, a tárba csak a biztosan szükséges lapok kerülnek be

Hátrányai:

- Új lapokra való hivatkozás mindig laphibát okoz.

A betöltendő lap kiválasztása 2.

Előretekintő lapozás (anticipatory paging)

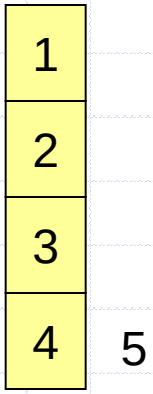
- Az OS megpróbálja kitalálni, hogy a folyamatnak a jövőben melyik lapokra lesz szüksége és azokat "szabad idejében" betölti.
  - Ha a jóslás gyors és pontos, akkor a futási sebesség jelentősen felgyorsul
  - Ha a döntés hibás, a felesleges lapok foglalják a tárat.
- A memória ára jelentősen csökken, így a mérete nő, a hibás döntés ára (a felesleges tárfoglalás) egyre kisebb.
- Az előretekintő lapozás egyre népszerűbb.

7.2. Lapcsere stratégiák

- Akkor lenne **optimális**, ha azt a lapot választaná ki, amelyre legtovább nem lesz szükség.
- A lapcserét nagyban gyorsítja, ha a mentést csak akkor végezzük el, ha az a lap a betöltés óta módosult. A hardver minden lap mellett nyilvántartja, hogy a lapra írtak-e a betöltés óta (*modified* vagy *dirty* bit).
- Algoritmusok:
  - Véletlen kiválasztás
  - Legrégebbi lap (FIFO)
  - Újabb esély
  - Óra algoritmus
  - Legrégebben nem használt lap
  - Legkevésbé használt lap
  - Mostanában nem használt lap

Példa: optimális algoritmus

- Algoritmus: azt a lapot helyettesítjük, amelyre még a leghosszabb ideig nem lesz szükség.
- Laphivatkozások (referencia-string):
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 4 keret (folyamatonként ennyi lap lehet egyszerre a memóriában)



4

5

6 laphiba

Időben előre néz, használat ideje számít

Ezt használjuk referenciának az algoritmusok teljesítmény-mérésekor

Véletlen kiválasztás

- Egyszerű, buta algoritmus, csak elvi jelentőségű, nem használják.

Legrégebbi lap (FIFO)

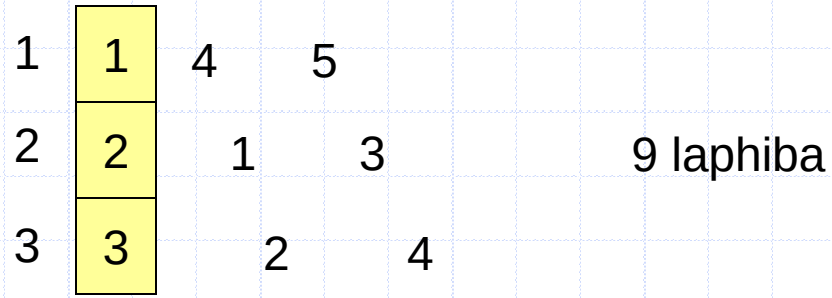
- A tárban lévő legrégebbi lapot cseréli le. Megvalósítása egyszerű FIFO listával történik.
- Hibája:
  - Olyan lapot is kitesz, amelyet gyakran használnak
  - Felléphet egy érdekes jelenség. Ha növeljük a folyamatokhoz tartozó lapok számát, a laphibák száma esetenként nem csökken, hanem nő! (Béládi-anomália)

Időben hátra néz, behozatal ideje számít

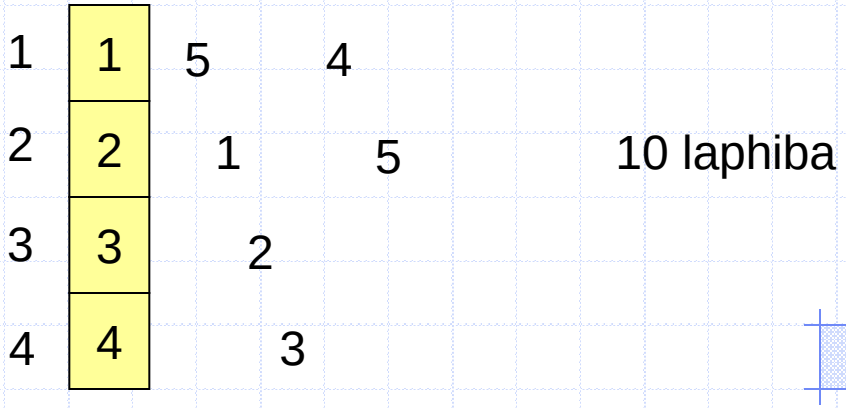
Béládi László magyar származású kutató, az IBM virtuális tárkezelésének kidolgozója.

Példa: FIFO

- Laphivatkozások (referencia-string): 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 keret (folyamatonként ennyi lap lehet egyszerre a memóriában)



- 4 keret



A Béládi-anomália

Laphivatkozások	0	1	2	3	0	1	4	0	1	2	3	4	0	1
FIFO 3 lap kerettel	0	0	0	3	3	3	4	4	4	4	4	4	0	0
		1	1	1	0	0	0	0	0	2	2	2	2	1
			2	2	2	1	1	1	1	1	3	3	3	3
Laphibák	*	*	*	*	*	*	*			*	*		*	*
FIFO 4 lap kerettel	0	0	0	0	0	0	4	4	4	4	3	3	3	3
		1	1	1	1	1	1	0	0	0	0	4	4	4
			2	2	2	2	2	2	1	1	1	1	0	0
				3	3	3	3	3	3	2	2	2	2	1
Laphibák	*	*	*	*			*	*	*	*	*	*	*	*

Újabb esély (Second Chance)

- A FIFO egy változata
- Minden lap tartalmaz egy *R* hivatkozás bitet, ami kezdetben törölve van.
- A lapra hivatkozáskor *R*=1.
- Lapcsere esetén:
  - Ha a sor elején levő lapon R=0, akkor csere.
  - Ha a sor elején levő lapon R=1, akkor az *R* bitet töröljük és a lapot a FIFO végére rakjuk. A következő (most már első) lappal próbálkozunk tovább.
- Kiküszöböli a FIFO fő hibáját.

Időben hátra néz, újrahasználat ténye számít

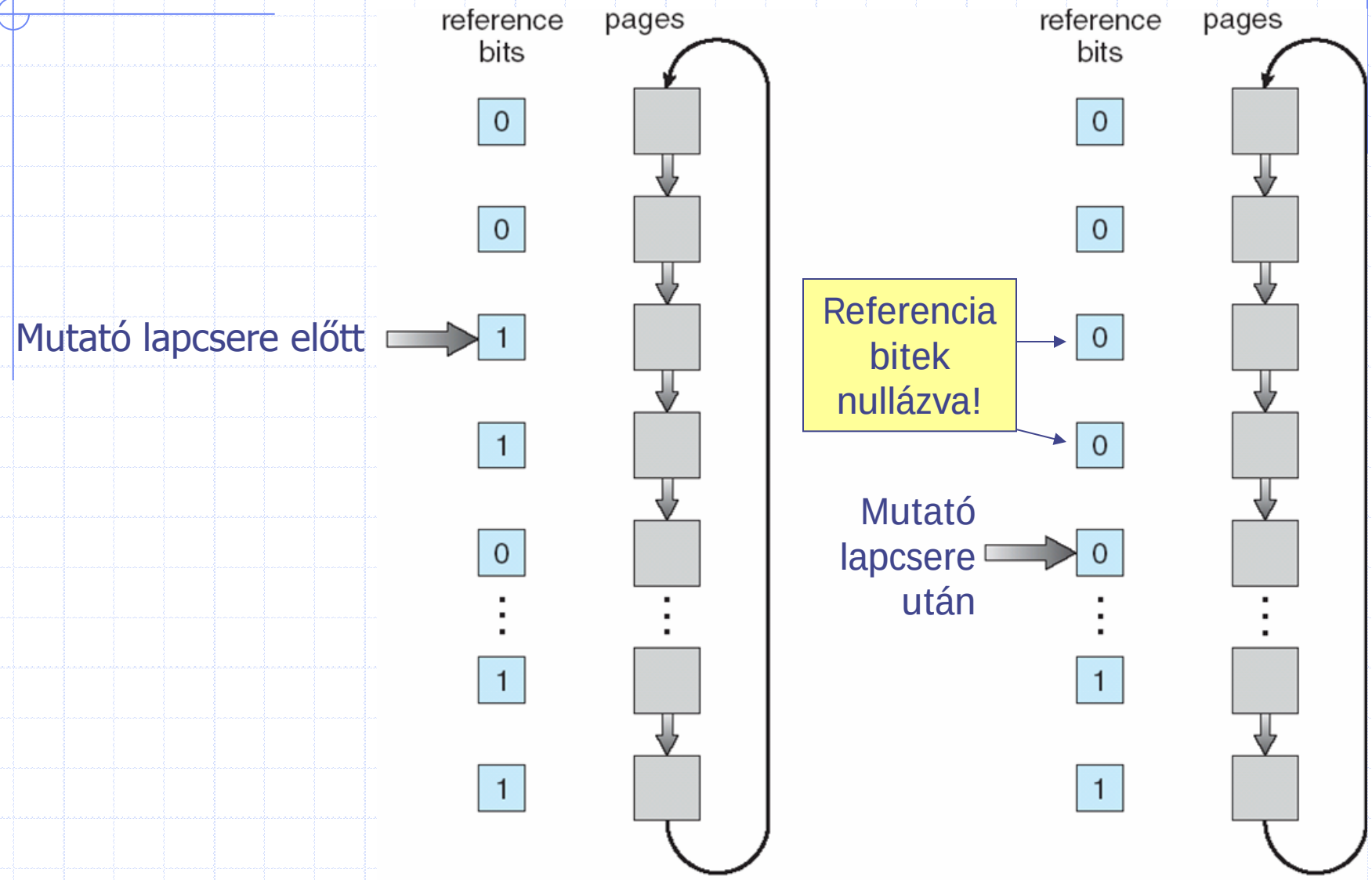
Újabb esély (Second Chance)

	0	1	2	3	0	1	4	0	1	2	3	4
	0	1	2	3	3	3	4	4	4	2	3	4
		0	1	2	2	2	1	1	1*	4	1	3
			0	1	1	1*	0	0*	0*	1*	0	1
				0	0*	0*	3	3	3	0*	2	0
	p	p	p	p			p			p	p	p

Óra algoritmus (Clock)

- Az újabb esély algoritmus másik implementációja
- A lapok körkörös láncban vannak felfűzve a betöltés sorrendjében.
- Lapcsere előtt az algoritmus megvizsgálja az *R* bitet: ha egynek találja, akkor nem veszi ki a lapot, törli az *R*-t és a mutatót továbblépteti.

Példa: Újabb esély/óra algoritmus



Legrégebben nem használt lap (Least Recently Used, LRU)

i.lap

idő

- Azt a lapot választjuk, amelyre a leghosszabb ideje nem hivatkoztak.
- A múltbeli információk alapján próbál előrelátni, az optimális algoritmust közelíteni.
- Szimulációs eredmények alapján jó teljesítmény, de nehéz implementálni.

Időben hátra néz, használat ideje számít

Legrégebben nem használt lap (Least Recently Used, LRU)

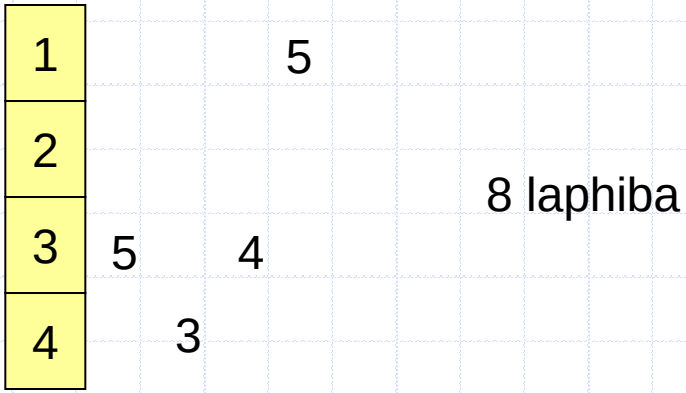
	0	1	2	3	0	1	4	0	1	2	3	4
	0	1	2	3	0	1	4	0	1	2	3	4
		0	1	2	3	0	1	4	0	1	2	3
			0	1	2	3	0	1	4	0	1	2
				0	1	2	3	3	3	4	0	1
	p	p	p	p			p			p	p	p

LRU implementáció

- Számlálóval
  - A lapra történő hivatkozáskor feljegyezzük annak idejét
  - A helyettesítendő lap kiválasztáskor a tárolt időpontok közül keressük a legrégebbit.
- Láncolt listával
  - A lapok egy láncolt listában vannak.
  - A frissen behozott lap a lista elejére kerül.
  - Hivatkozáskor a lista elejére kerül a lap. A lista végén van a legrégebben nem használt lap.
  - Nem kell hosszadalmas keresés.
- Csak hardver támogatással lehetne jól implementálni.

Példa: LRU

- Laphivatkozások (referencia-string): 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 4 keret (folyamatonként ennyi lap lehet egyszerre a memóriában)



Legkevésbé használt lap (Least Frequently Used, LFU)

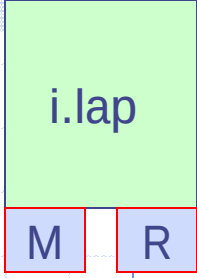
i.lap

CT | R

- Leggyakrabban használt lapok a memóriában.
- Implementálás:
  - Lap hivatkozásakor *R* bebillentése
  - Periodikusan minden lapnál egy számlálót növel, ha *R*=1, majd törli az *R*-t.
  - Lapcsere esetén a legkisebb számláló-értékű lapot dobja ki.
- Hátrány:
  - A valamikor sokat használt lapok a memóriában maradnak (öregítéssel lehet segíteni ezen)
  - A frissen betöltött lapok könnyen kiesnek (frissen behozott lapok befagyasztása, *page locking*)

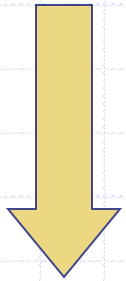
Időben hátra néz,
használat mennyisége számít

Mostanában nem használt lapok (Not Recently Used, NRU)



- Hivatkozott (*R*) és módosított (*M*) bitek használata.
- OS időközönként törli az *R* bitet, *M* bitet viszont őrizni kell (lap törlése információ veszteséghez vezetne).

- Négy prioritás
  1. nem hivatkozott, nem módosult
  2. nem hivatkozott, módosított
  3. hivatkozott, nem módosult
  4. hivatkozott, módosult



- A módszer a legkisebb prioritásból választ véletlenszerűen.

Időben hátra néz, használat ténye számít

Egyéb változatok

- Szabad lapok fenntartása, ha a kiválasztott lap módosul, a szabad lapok közül adunk.
- Majd később írjuk ki a lapot, amely a szabadok közé kerül.
- A módosult lapokat henyélés közben írjuk ki.

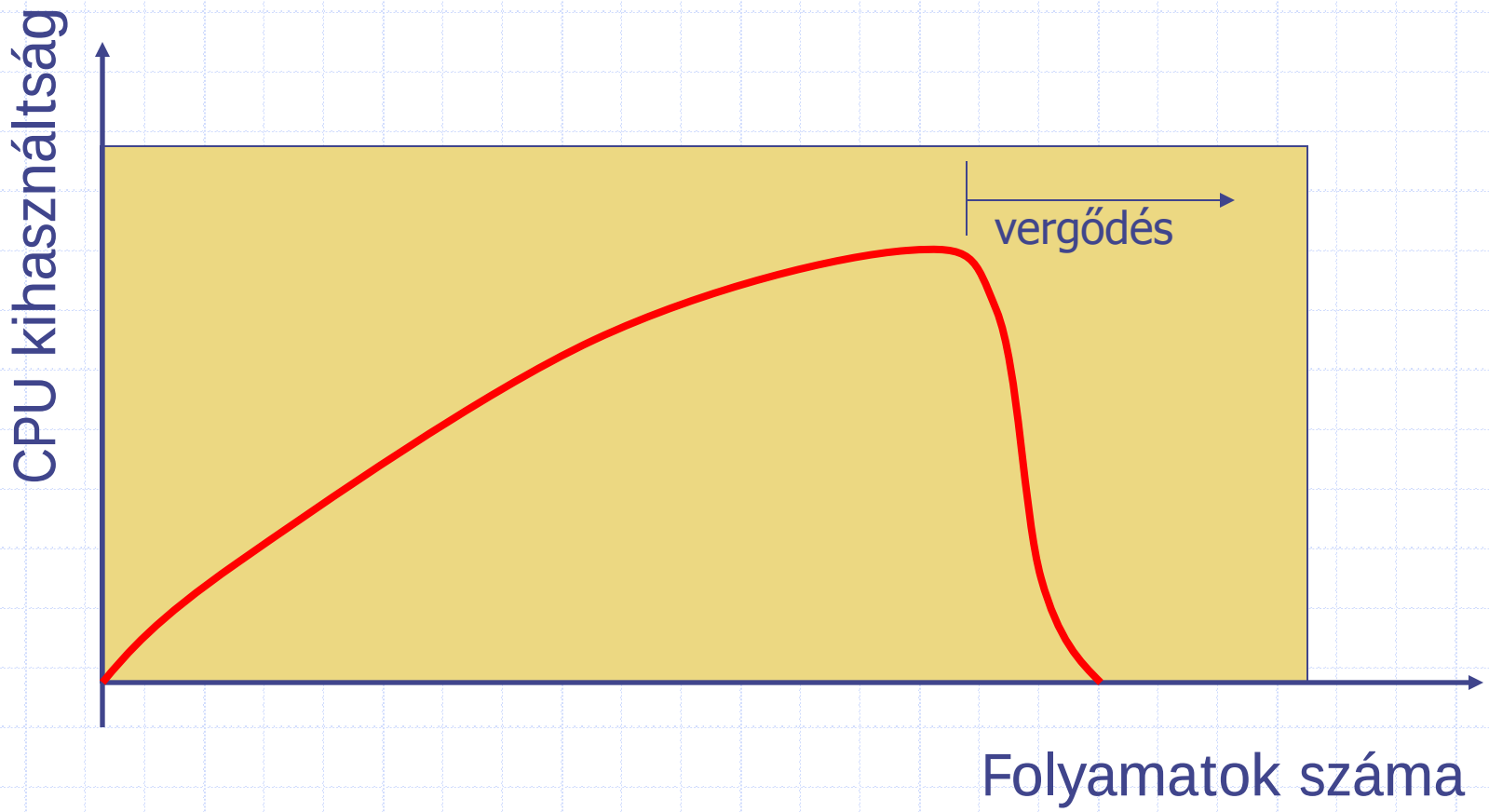
7.3. Folyamatok lapigénye, lapok allokációja

- Folyamat szempontjából az a jó, ha minél több lapja van bent a tárban.
- Minimális lapigény számítható a CPU utasításokból. Maximális (optimális) nehezen határozható meg.

Vergődés (thrashing)

- Ha egy folyamat vagy rendszer több időt tölt lapozással, mint amennyit hasznosan dolgozik.
- Oka:
  - Folyamat: kevés lap van a tárban, gyakran hivatkozik a háttértáron lévőkre
  - Rendszer: Túl sok a folyamat, ezek egymás elől lopkodják el a lapokat, mindegyik folyamat vergődni kezd.
- Elkerülése: a tárban lévő folyamatoknak biztosítani kell a futáshoz szükséges optimális számú lapot.

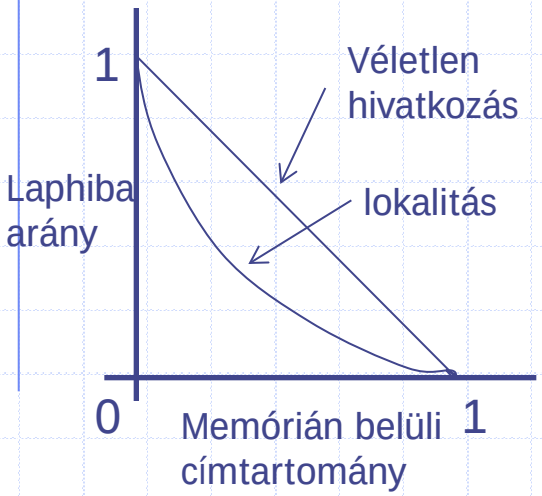
Vergődés



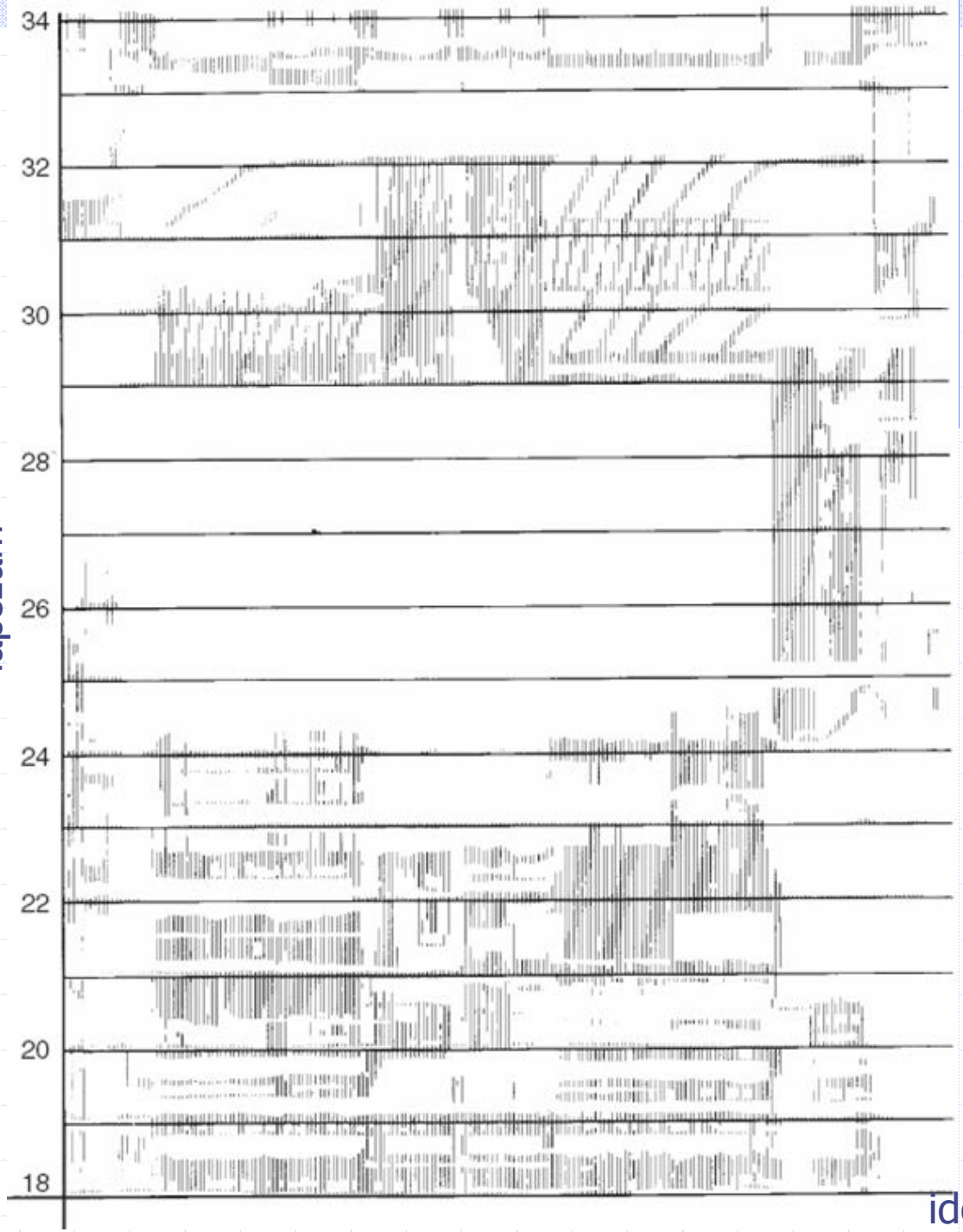
Lokalitás

- Statisztikai tulajdonság: a folyamatok egy időintervallumban csak címtartományuk szűk részét használják.
- időbeli:
  - a hivatkozott címre a közeljövőben nagy valószínűséggel újra hivatkozni fog (ciklusok, eljárások, verem változók, stb.)
- térbeli:
  - hivatkozott címek egymás melletti címre történnek (soros kód, tömbkezelés).

Lokalitás



Memória-hozzáférési térkép

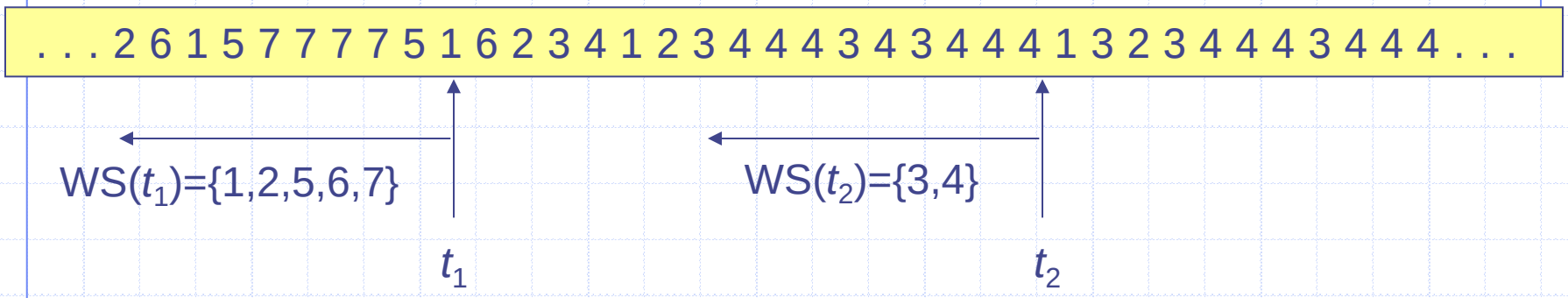


A munkahalmaz modell

- A munkahalmaz (*working set*, WS) az elmúlt Δ időben (munkahalmaz ablak) hivatkozott lapok halmaza.
- A lokalitás miatt a folyamatnak nagy valószínűséggel ezekre a lapokra lesz szüksége a közeljövőben.
- Munkahalmaz mérete folyamatonként és időben is változik (lásd a következő példát).
- OS célja minden aktív folyamat számára biztosítani a munkahalmazt.
- Pontos méréséhez bonyolult hardver kellene, ezért fix időintervallumonkénti mintavételezéssel oldják meg.

Munkahalmaz mérése

Példa (Δ = 10):



Mérése (példa):

- A WS mérete: Δ=10000.
- Újabb referencia bitek használata (pl. laponként 2 bit)
- Időközönként (pl. 5000 laphivatkozásonként) a referenciabitek vizsgálata/törlése.
  - Ha valamely bit 1, akkor a lapra történt hivatkozás az utóbbi 5000, vagy az előtte lévő 5000 laphivatkozás alatt → lap része a WS-nek.

A munkahalmaz mérete

Fontos kérdés: mekkora legyen Δ ?

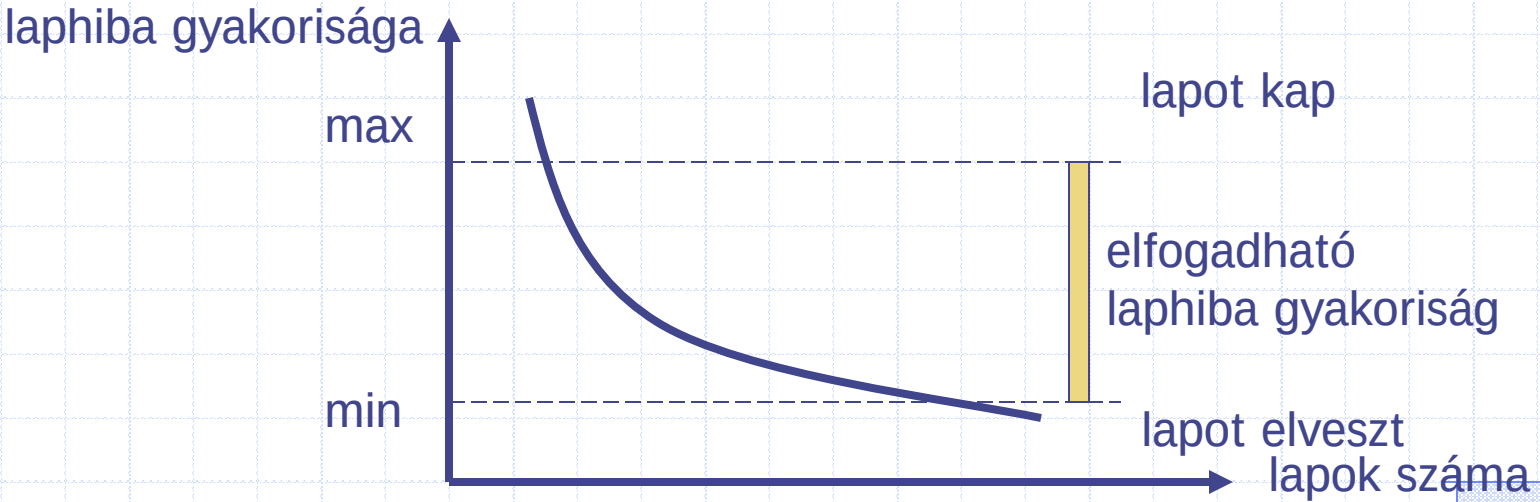
- Ha túl kicsi:
  - nem tartalmazza az egész lokalitást
- Ha túl nagy:
  - több lokalitást is tartalmaz (feleslegesen)

A munkahalmaz és a vergődés kapcsolata

- WSS_i: az i-ik folyamat munkahalmazának mérete
  - A példában: $WSS(t_1)=5$, $WSS(t_2)=2$
- A rendszer teljes lapigénye: $D=\Sigma WSS_i$
- A rendszerbeli lapkeretek száma: m
- Vergődés akkor lép fel, ha D>m
  - (vagyis a lokalitások összes mérete nagyobb, mint a rendelkezésre álló memória mérete)

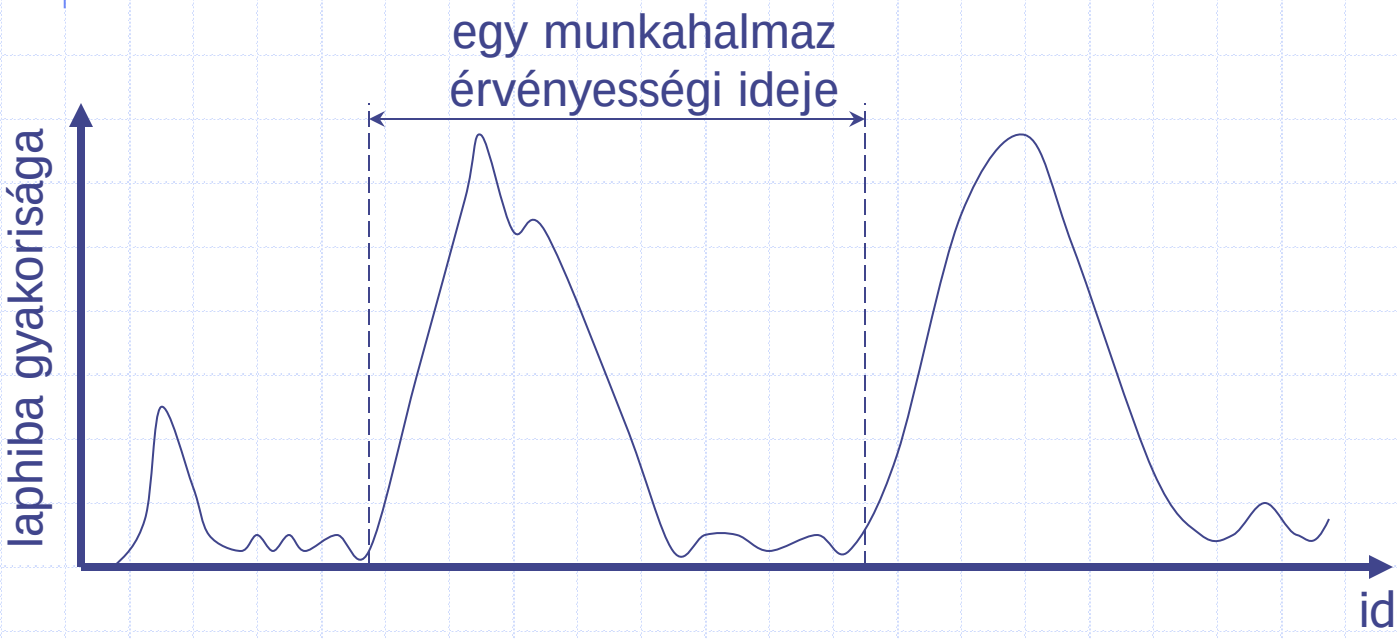
Laphiba gyakoriság figyelése

- A munkahalmaz modellnél egyszerűbb módszer a vergődés elkerülésére
- Stratégia:
  - ha folyamatnak sok a laphibája, akkor újabb lapot kell adni neki,
  - ha kevés, akkor más folyamatok rovására túl sok tartozik hozzá.
- Az OS a laphiba gyakoriságát vizsgálja.
  - Ha a laphibák gyakorisága a maximum felett van, akkor lapot kap
  - Ha a laphibák gyakorisága a minimum alatt van, akkor lapot veszünk tőle
  - Csak akkor indítunk el újabb folyamatot, ha van elég szabad lap.
  - Szükség esetén folyamatokat fel lehet függeszteni, lapjait ki lehet osztani.



A laphiba gyakoriság és a munkahalmaz kapcsolata

- Ha a munkahalmaz (kb.) a memóriában van, akkor kicsi a laphibák gyakorisága
- A munkahalmaz változásakor megnő a laphibák gyakorisága.
- A laphiba gyakoriság tipikus alakulása:



7.4. Egyéb tervezési szempontok

- A legfontosabb tervezési szempontok:
  - Lapcsere stratégia kiválasztása (7.2)
  - Lapok allokációja (7.3)
- Egyéb fontos szempontok:
  - Előre lapozás
  - Lapok mérete
  - Asszociációs memória fedése
  - Programozási trükkök
  - Lapok befagyasztása
  - ...

Előre lapozás

- Folyamat indításakor, illetve aktiválásakor az összes lapja a háttértáron van.
- Érdemes a várható munkahalmazt behozni a tárba.
- Akkor hasznos, ha az előre lapozás találati aránya nagy, különben pazarlás.
- Példa:
  - munkahalmaz modell használata esetén a felfüggesztett folyamat teljes munkahalmazát aktiválás esetén előre be lehet lapozni.

A lapméret hatása

- Lapméret (tipikusan: 4kByte – 4Mbyte)
- Lap méretének növelése:
  - laptábla mérete csökken
  - perifériás átviteli idő csökken (nagyobb blokkot relatíve gyorsabb átvinni, mert kevesebb az keresési idő/adminisztráció)
- Lap méretének csökkenése:
  - a lokalitás jobban érvényre jut (kisebb a munkahalmaz)
  - kisebb belső tördelődés

Az asszociációs memória fedése

- Az asszociációs memória (TLB) találati aránya fontos jellemző (lsd. 6.5.2).
- Másik fontos jellemző a TLB *fedése*. Ez azt mutatja meg, hogy a TLB-n keresztül mekkora memóriát lehet elérni.
- A TLB fedés értéke nyilvánvalóan:

  TLB bejegyzések száma x lapméret

Programozási trükkök

- Programírás
  - több dimenziós tömbök tárbeli elhelyezésének megfelelő bejárása
  - egyszerre használt változó egymás mellé
  - egymást hívó eljárások egymás mellé
  - célszerű kerülni a nagy ugrásokat (lokalitást mutató adatszerkezetek preferálása)
- Fordítás
  - eljárások egymás mellé helyezése
  - kód és az adatterület szétválasztása (kód nem változik, a lapcserénél nem kell kiírni)

Példa: a programstruktúra hatása

A lapméret legyen 128 bájt.

`int[128,128] data;`

- A fordító soronként tárol: data[0][0], data[0][1], ..., data[0][127], data[1][0], data[1][1], ..., data[127][127]
  → minden sor egy lapon tárolódik

- **1. program**

```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

128 x 128 =
16,384 laphiba

- **2. program**

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

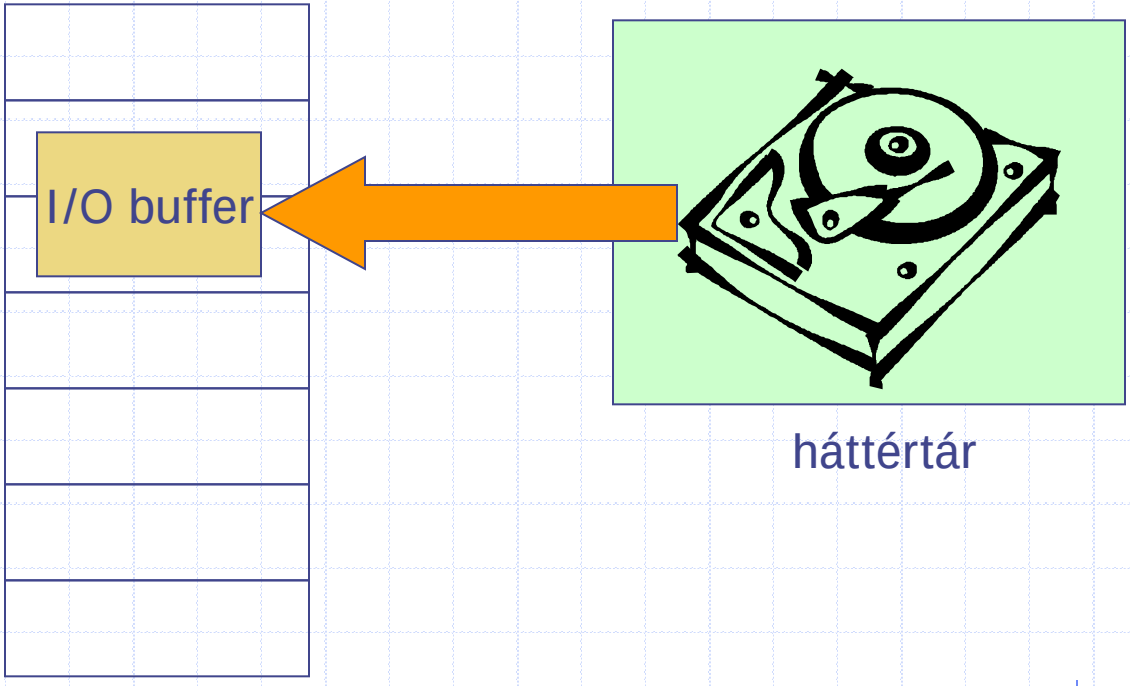
128 laphiba

Lapok tárba fagyasztása

- Az elindított perifériás műveleteknél a kijelölt címtartományt az átvitel idejére érdemes befagyasztani.
- A beemelt lapokat az első hivatkozásig érdemes befagyasztani.
- Tipikus példák:
  - OS magjának befagyasztása
  - I/O lapok befagyasztása
- Megoldás módja:
  - Minden laphoz „*lock*” bit

Példa: tárba fagyasztás I/O alatt

- Háttértárról beolvasás pufferbe (segédprocesszor)
- Ha kész, a CPU megszakítást kap
- Művelet közben a lapnak a memóriában kell lenni!



Copy on write (COW)

- Folyamatok indításának hatékony módszere:
- A gyermek és szülő folyamatok kezdetben azonos memóriaterületet használnak
- Ha a közös lapot valamely folyamat módosítja, akkor történik meg a lap másolása

Áttekintés

Mikor foglalkozik az OS lapkezeléssel?

1. Folyamat létrehozása
   - Programméret meghatározása
   - Laptábla létrehozása
2. Folyamat végrehajtása
   - Új folyamat esetén MMU reset
   - TLB kiürítése
3. Laphiba
   - A hibát okozó virtuális cím meghatározása
   - Lap kivitele (felszabadítás), szükséges lap behozatala
4. Folyamat terminálása
   - A laptábla és a lapok felszabadítása

7.5. Esettanulmányok

- Windows XP
- Solaris
- Linux

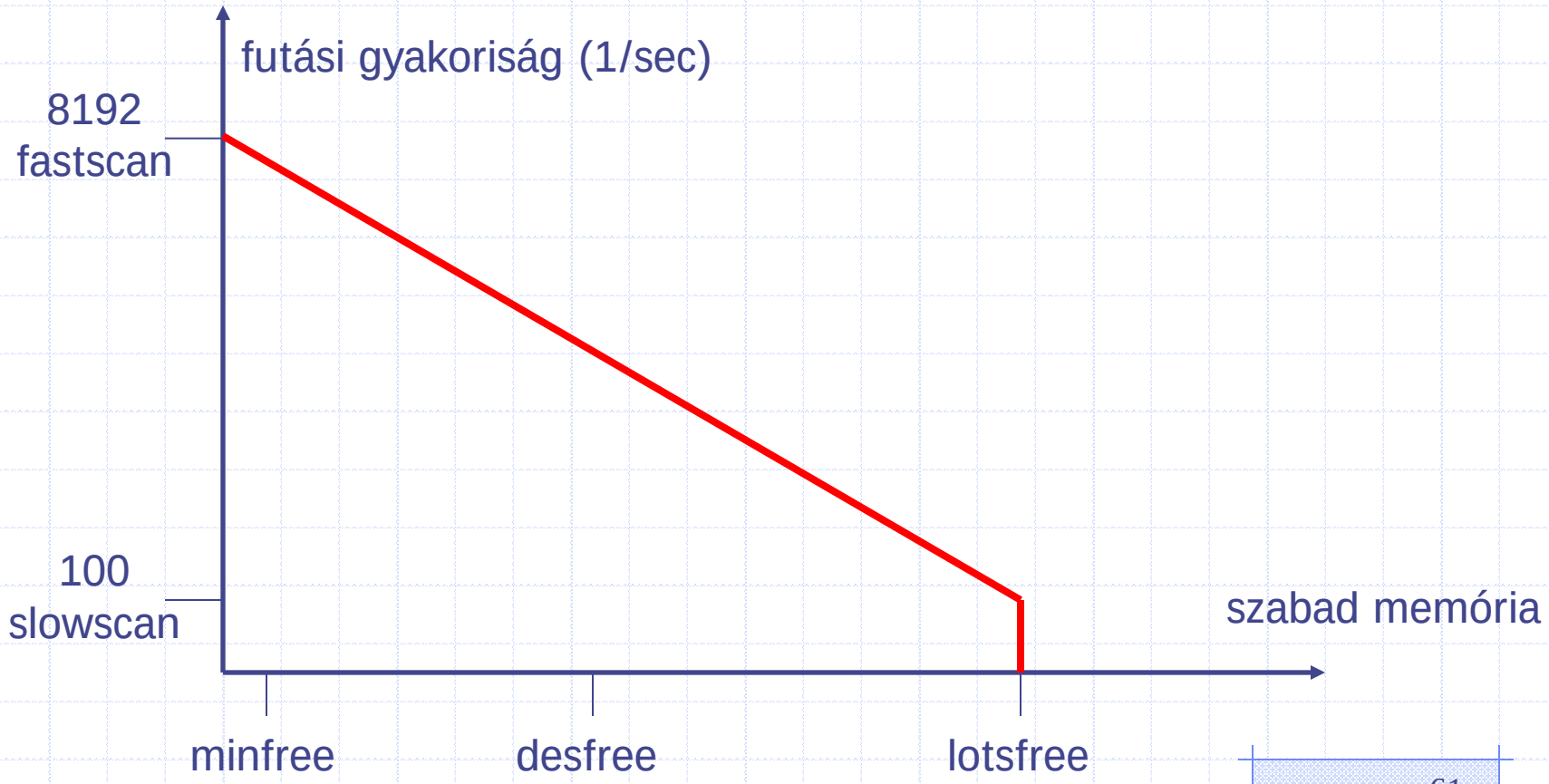
Windows XP

- Igény szerinti lapozás *klaszterezés*sel módosítva. A klaszterezés során a laphibát okozó lapot és annak környezetét olvassuk be.
- Minden folyamatnak van két paramétere:
  - *working set minimum*
  - *working set maximum*
- A *working set minimum* a memóriában tartott lapok minimális, garantált számát tartalmazza (pl. 50).
- Egy folyamat kaphat a *working set maximum* által meghatározott számú lapot (pl. 345).
- Ha a rendszer szabad memóriája egy küszöbérték alá csökken, akkor egy automatikus munkahalmaz vágás (automatic working set trimming) hajtódik végre.
- Az automatikus munkahalmaz vágás során a folyamatoktól elvesszük a *working set minimum* feletti lapokat.
- Lapcsere algoritmus:
  - Egyprocesszoros x86 típusú rendszereken: óra algoritmus.
  - Alpha, többprocesszoros x86: módosított FIFO

Solaris

- Szabad lapokat tart fenn, ebből kapnak a laphibát okozó folyamatok. A jó működéshez elegendő mennyiségű szabad memória kell.
- *Lotsfree* – küszöbérték. Ha a szabad memória ez alá csökken, elindul a **pageout** folyamat.
- *Desfree* – küszöbérték. Ha a szabad memória ez alá csökken, a pagout gyakrabban fut, folyamatok swappelése megkezdődik (a kiswappelt folyamat valamennyi lapja felszabadítódik).
- *Minfree* – küszöbérték. Ha a szabad memória ez alá csökken, akkor a pagout minden új lap kérésekor lefut.
- A lapozást a pagout folyamat hajtja végre egy módosított óra algoritmussal.
- A *scanrate* paraméter határozza meg pagout futási gyakoriságát a lassútól (*slowscan*) a gyorsig (*fastscan*).

Solaris


futási gyakoriság (1/sec)
8192
fastscan
100
slowscan
szabad memória
minfree
desfree
lotsfree

Linux

- Szabad lapok listája, ebből ad laphiba esetén.
- *kswapd*: page démon. Ellenőrzi, hogy van-e elég szabad lap (1/sec).
  - Kevés szabad lap esetén felszabadításba kezd, egyre fokozódó intenzitással (először a paging cache-ből, majd a kevéssé használt osztott memóriából, végül a felhasználóktól).
  - Óra-algoritmus variációja, de nem FIFO szerint keres, hanem virtuális cím szerint (lokalitás elve!).
  - Kidobandó lap kezelése:
    - „tiszta”: azonnal kidob
    - „piszkos” és van már háttértáron korábbi másolata: kiírásra ütemez
    - „piszkos” és nincs a háttértáron korábbi másolata: paging cache-be kerül
- *bdflush*: periodikusan ellenőrzi, hogy a lapok mekkora része piszkos. Ha túl sok, elkezdi kiírni őket.