Gábor Dénes
Foiskola

ELOADÁS VÁZLATOK

OPERÁCIÓS RENDSZEREK

105

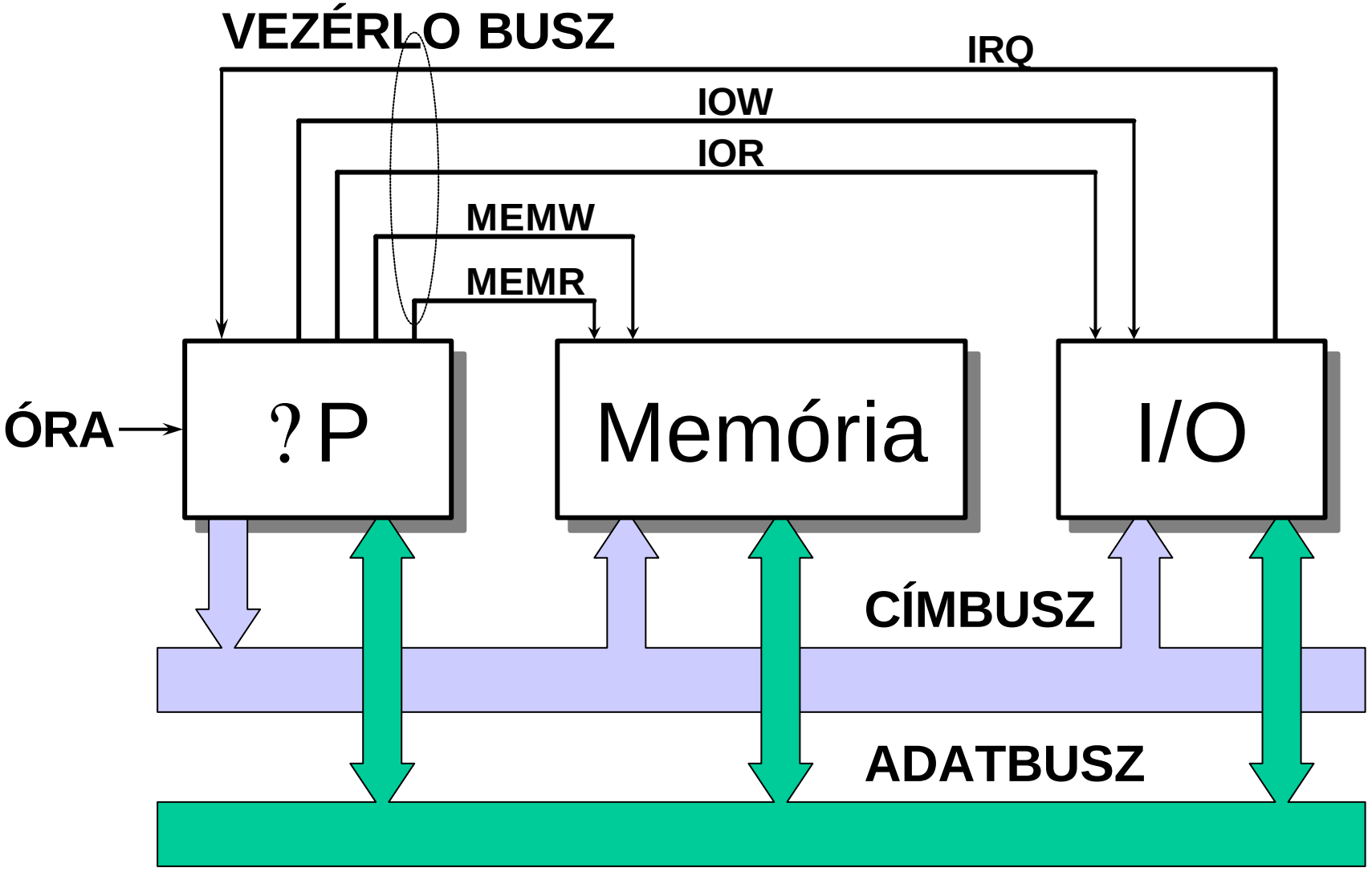
Vezetotanár:
KNAPP GÁBOR

2001/2002
tavasz

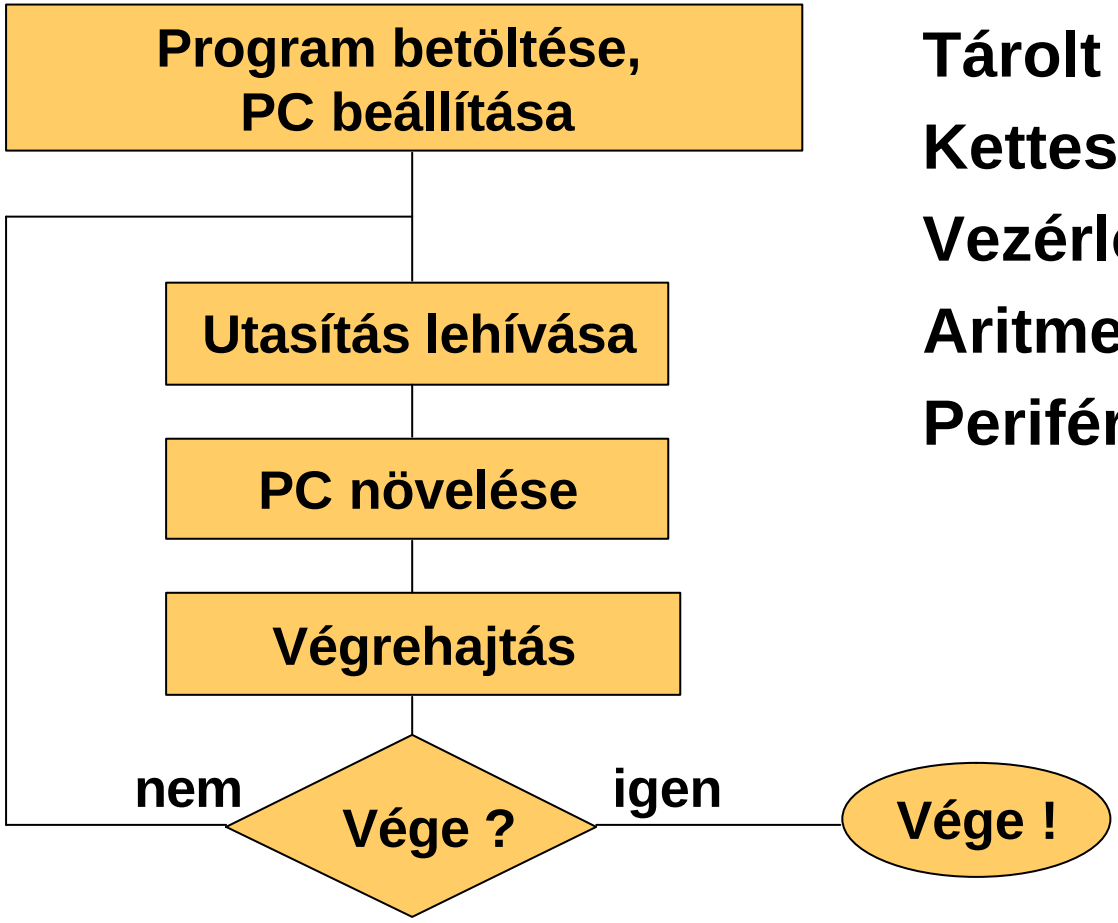
Bevezetés

- A számítógépek felépítése (ismétlés)
- Múlt, jelen, jövo

- Alapfogalmak
  - Folyamatok, Eroforrások
  - Az operációs rendszerek felépítése, feladata


VEZÉRLŐ BUSZ
IRQ
IOW
IOR
MEMW
MEMR
ÓRA
?P
Memória
I/O
CÍMBUSZ
ADATBUSZ

Neumann-ciklus, elvek



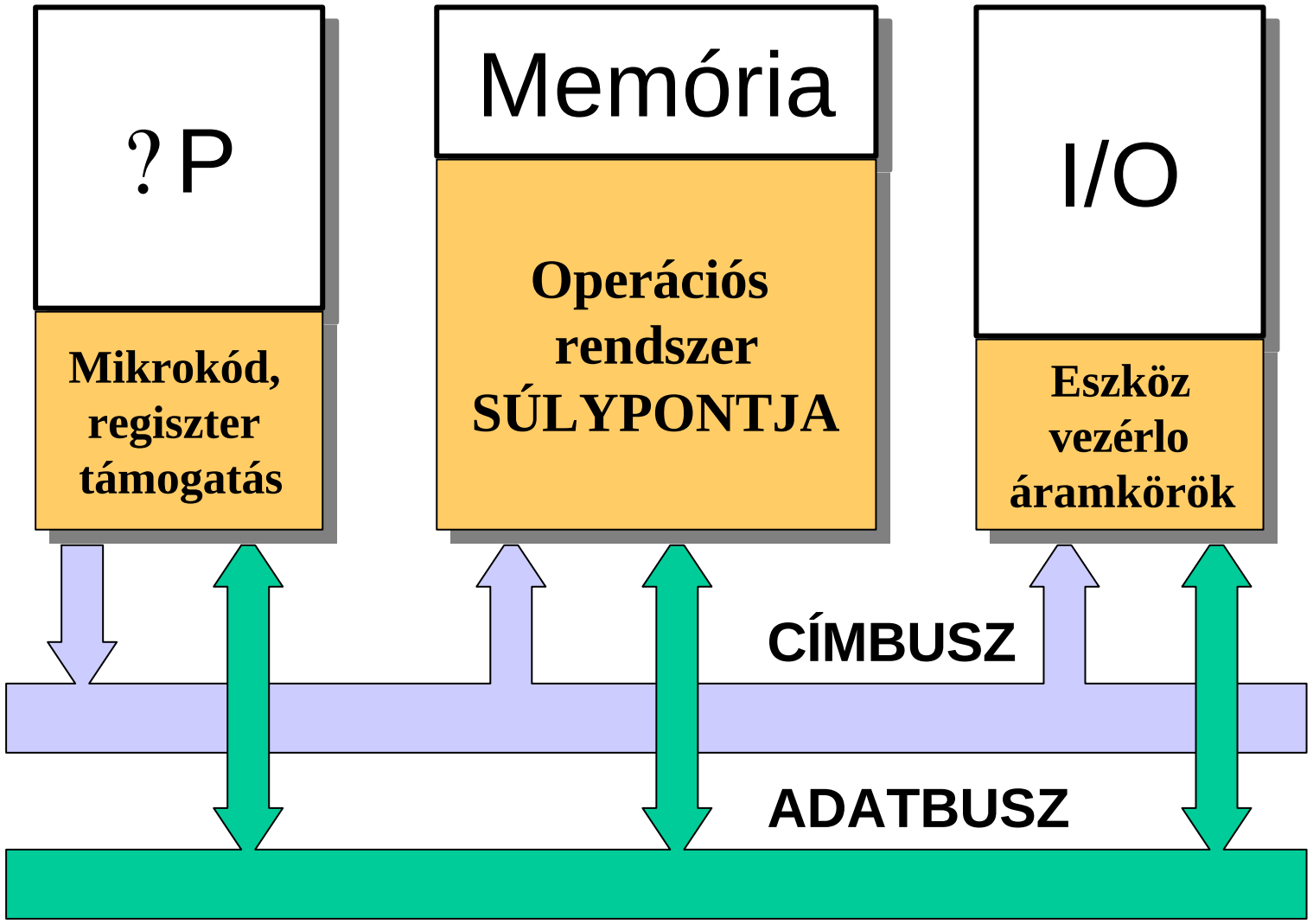
Tárolt program

Kettes számrendszer

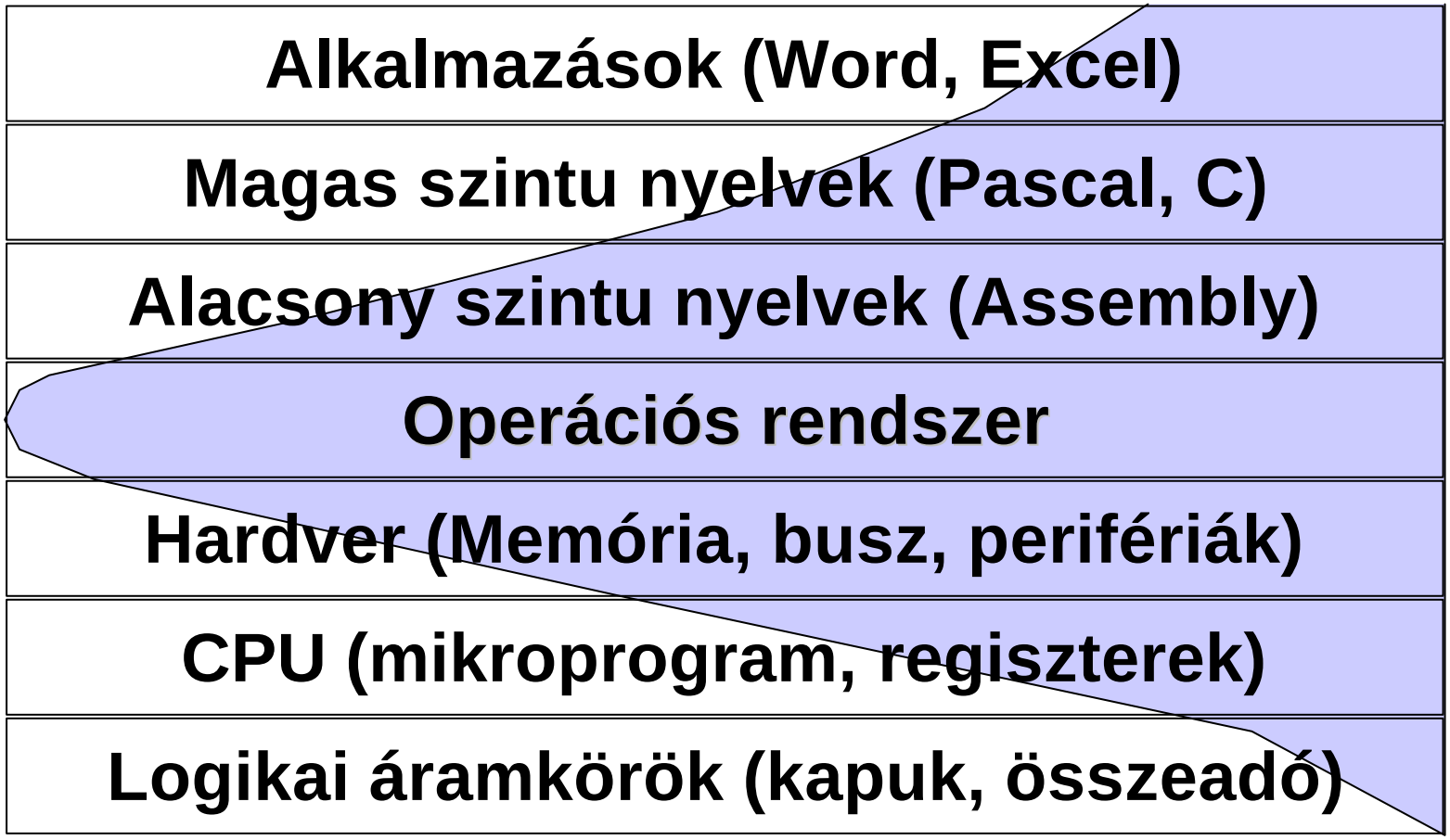
Vezérloegység

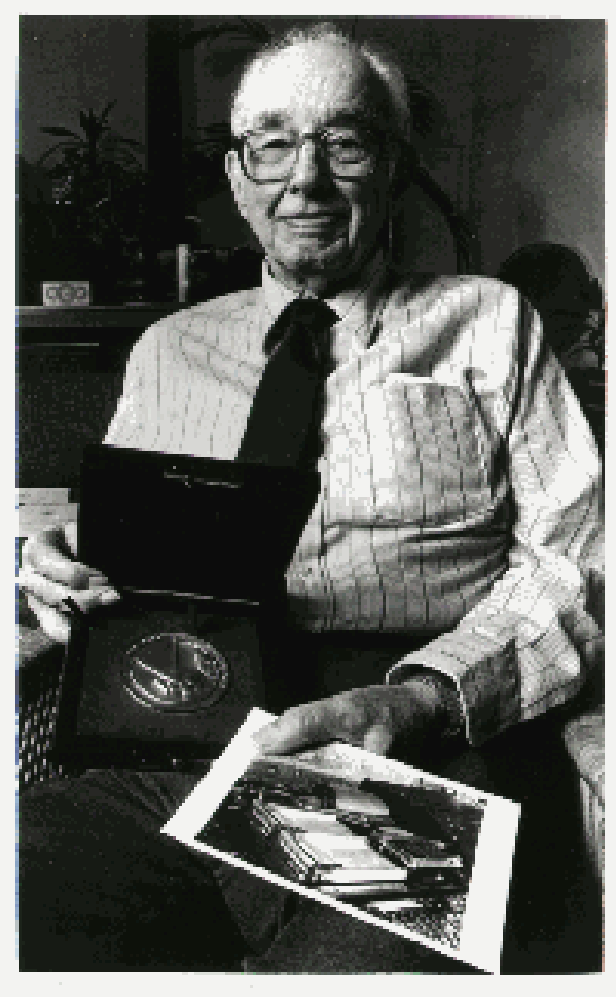
Aritmetikai-Logikai egység

Perifériák


?P
Memória
I/O
Mikrokód, regiszter támogatás
Operációs rendszer SÚLYPONTJA
Eszköz vezérlo áramkörök
CÍMBUSZ
ADATBUSZ

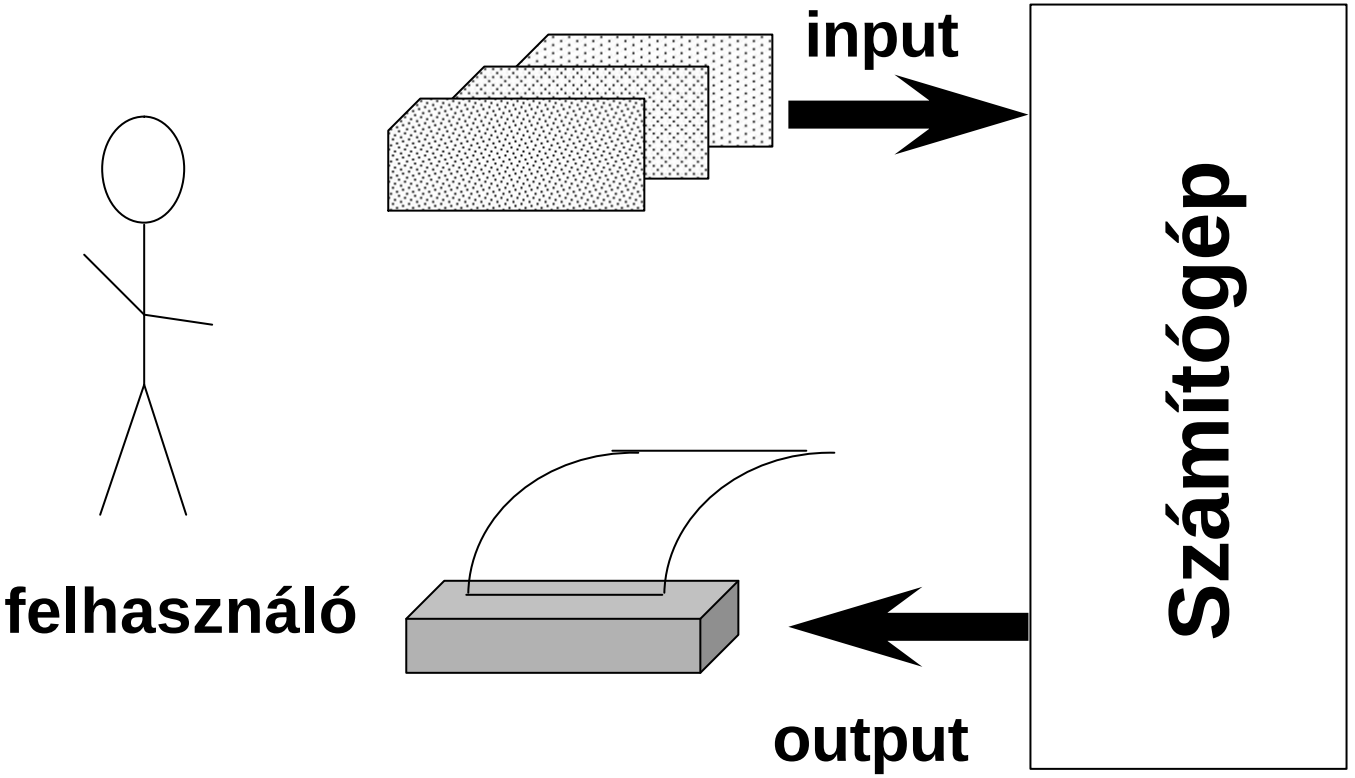
A logikai modell



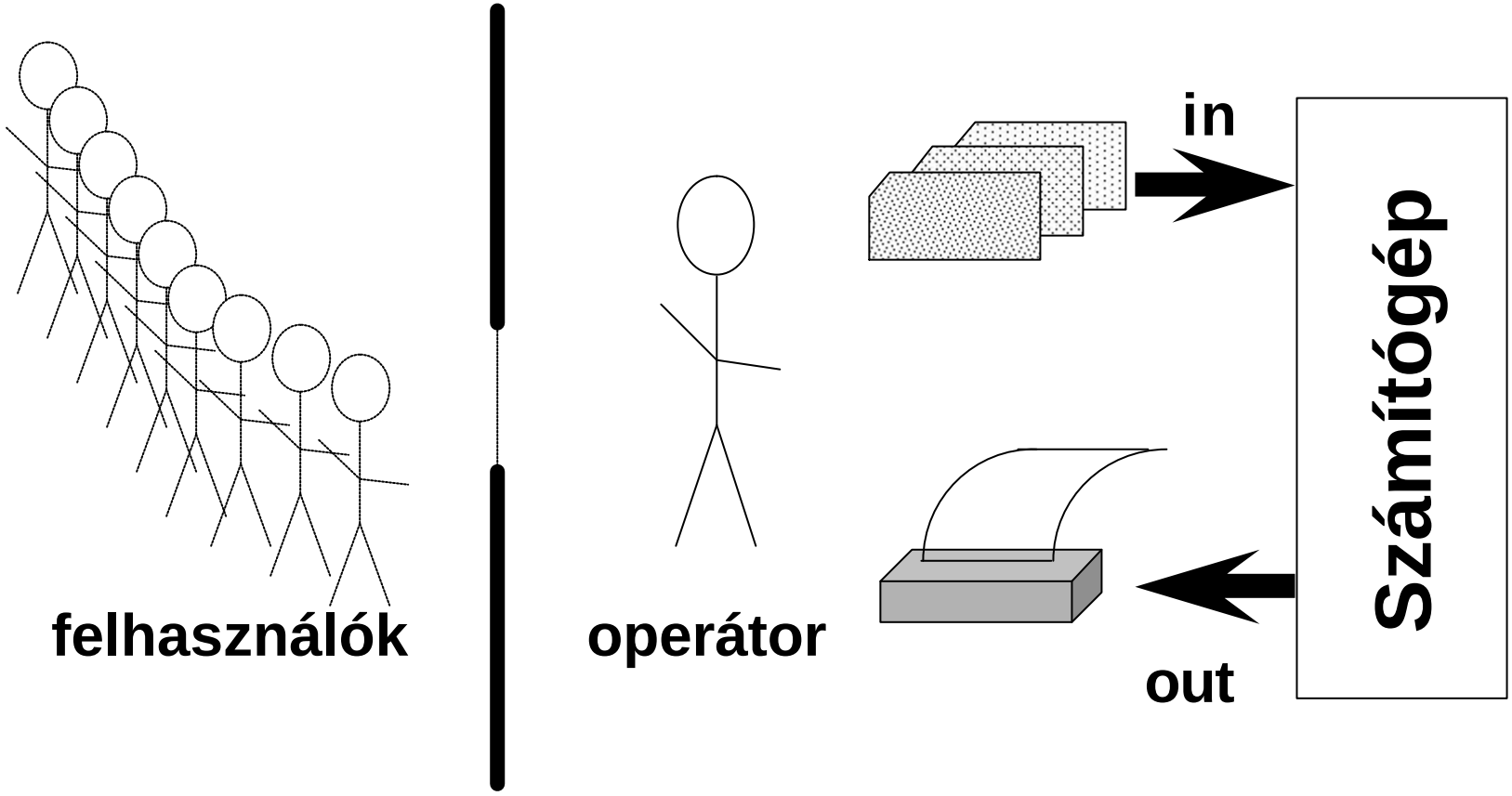


John Vincent Atanasoff
(1903-1995)

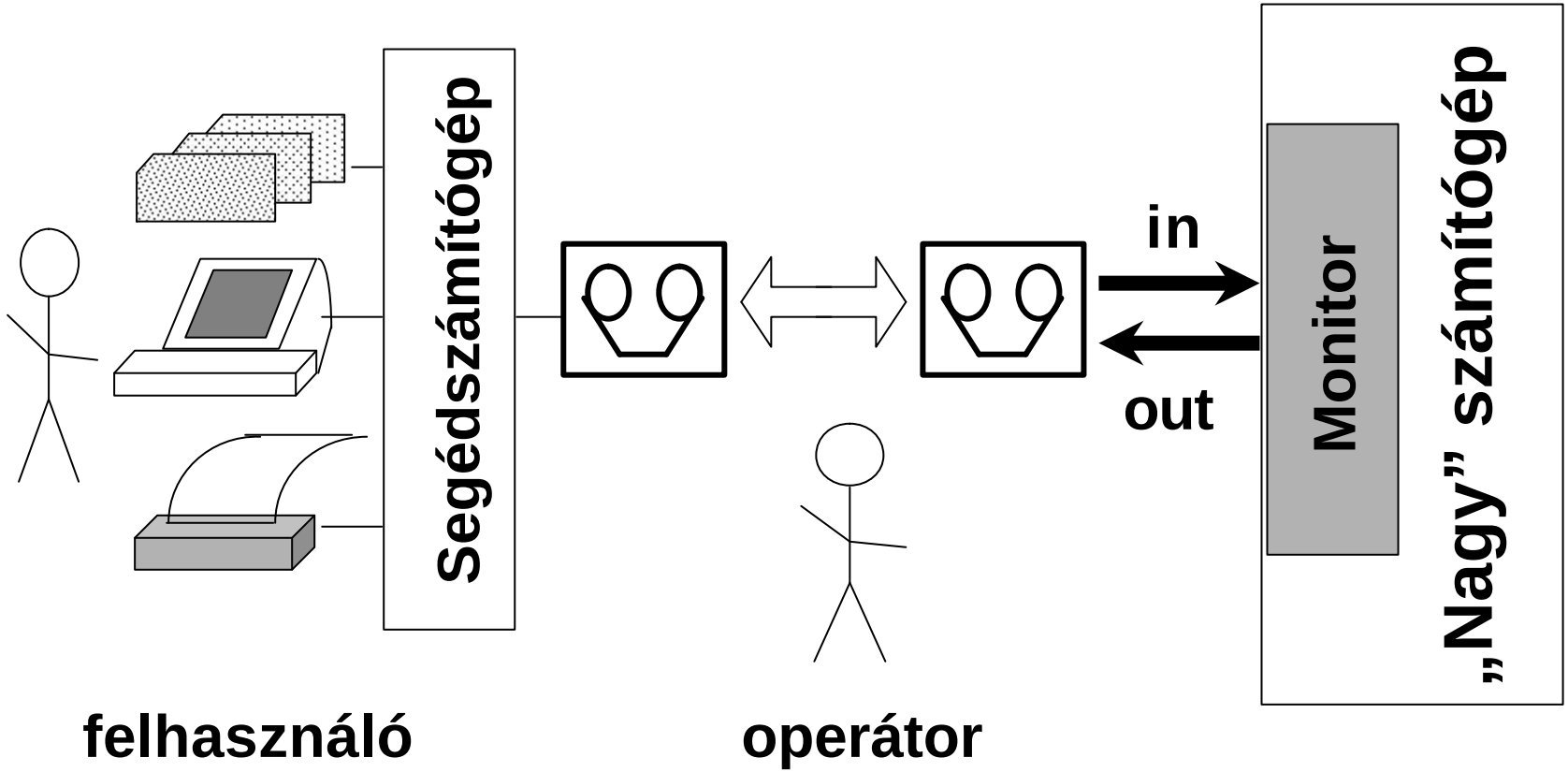
„Open shop”



„Closed shop”



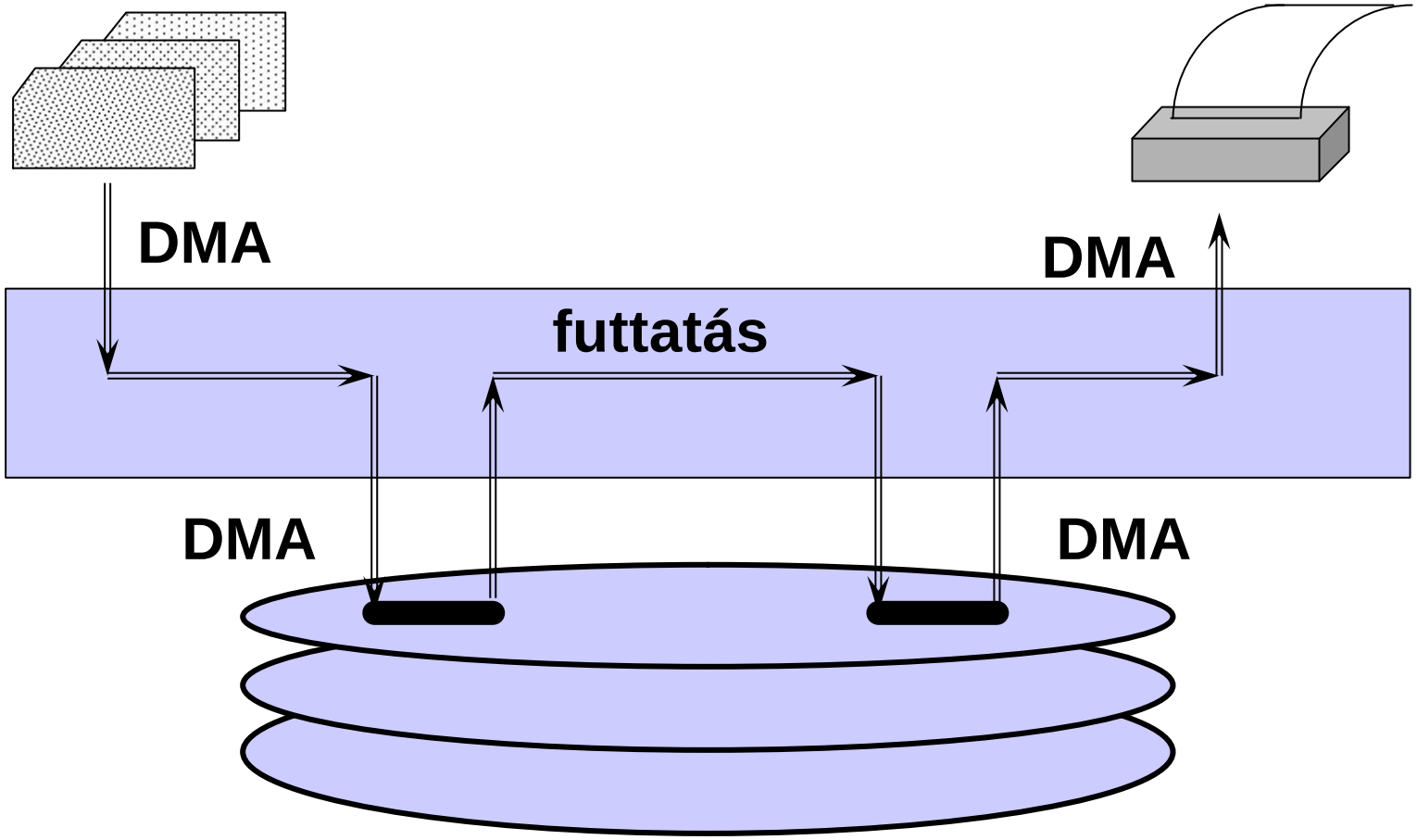
Mágnesszalag alkalmazása



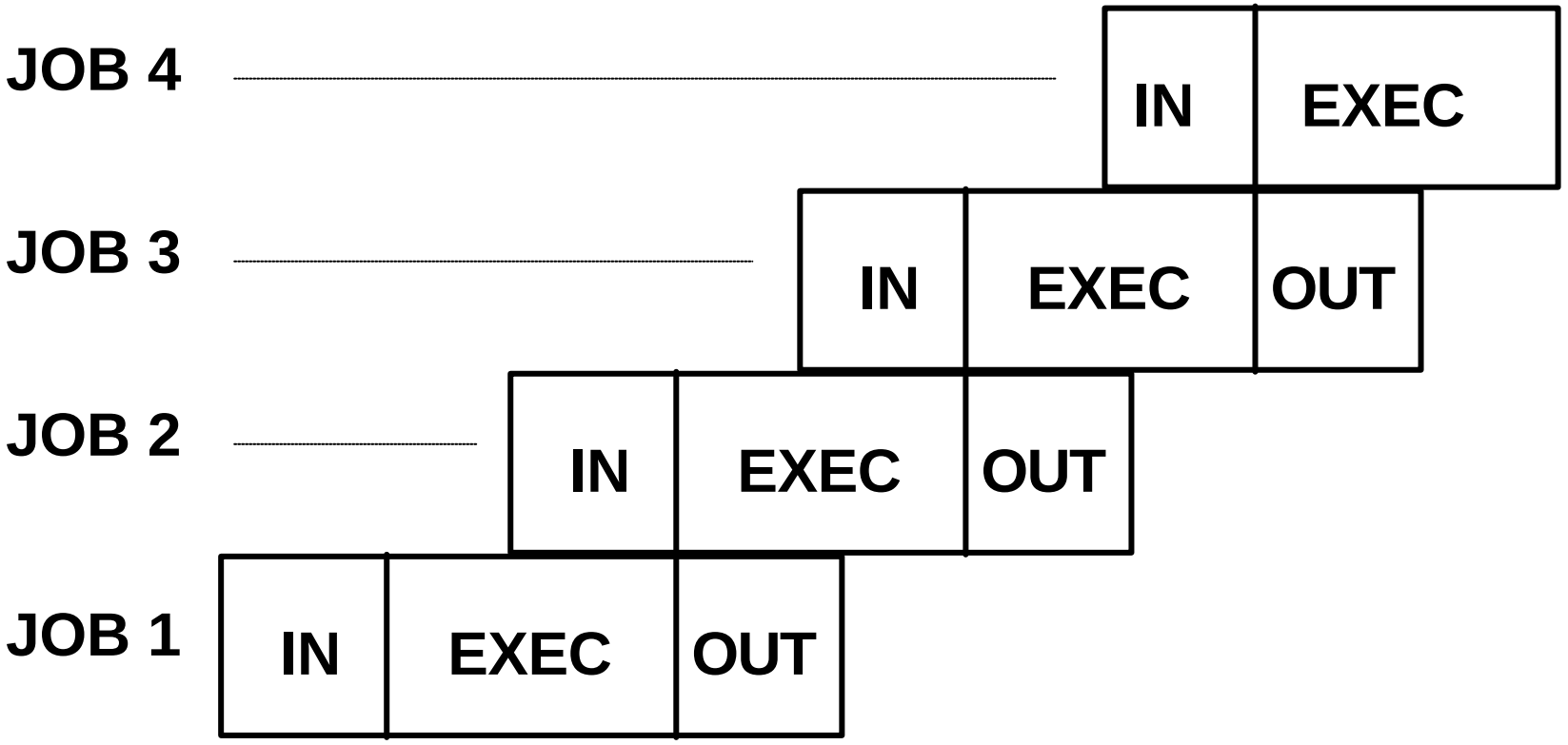
Parancsnyelv (pl. JCL)

```
$ JOB OK     # a munka neve
$ FTN        # a FORTRAN fordító indítása
-
-            # FORTRAN programsorok
-
$ LOAD       # a lefordított program betöltése
$ RUN          futtatás
-
-            # a program bemeno adatai
-
$ END        # a munka vége
```

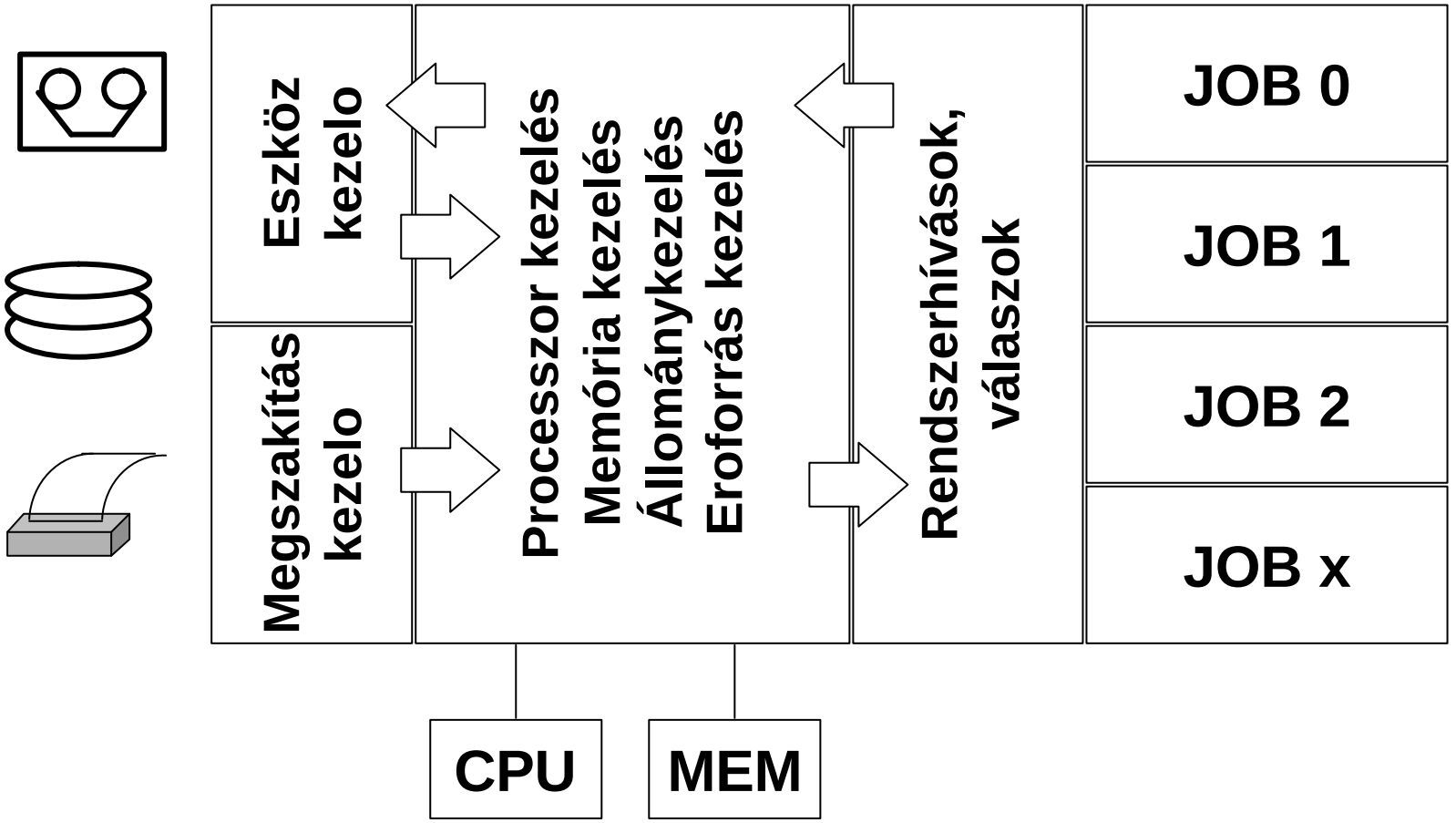
„Spooling” rendszer



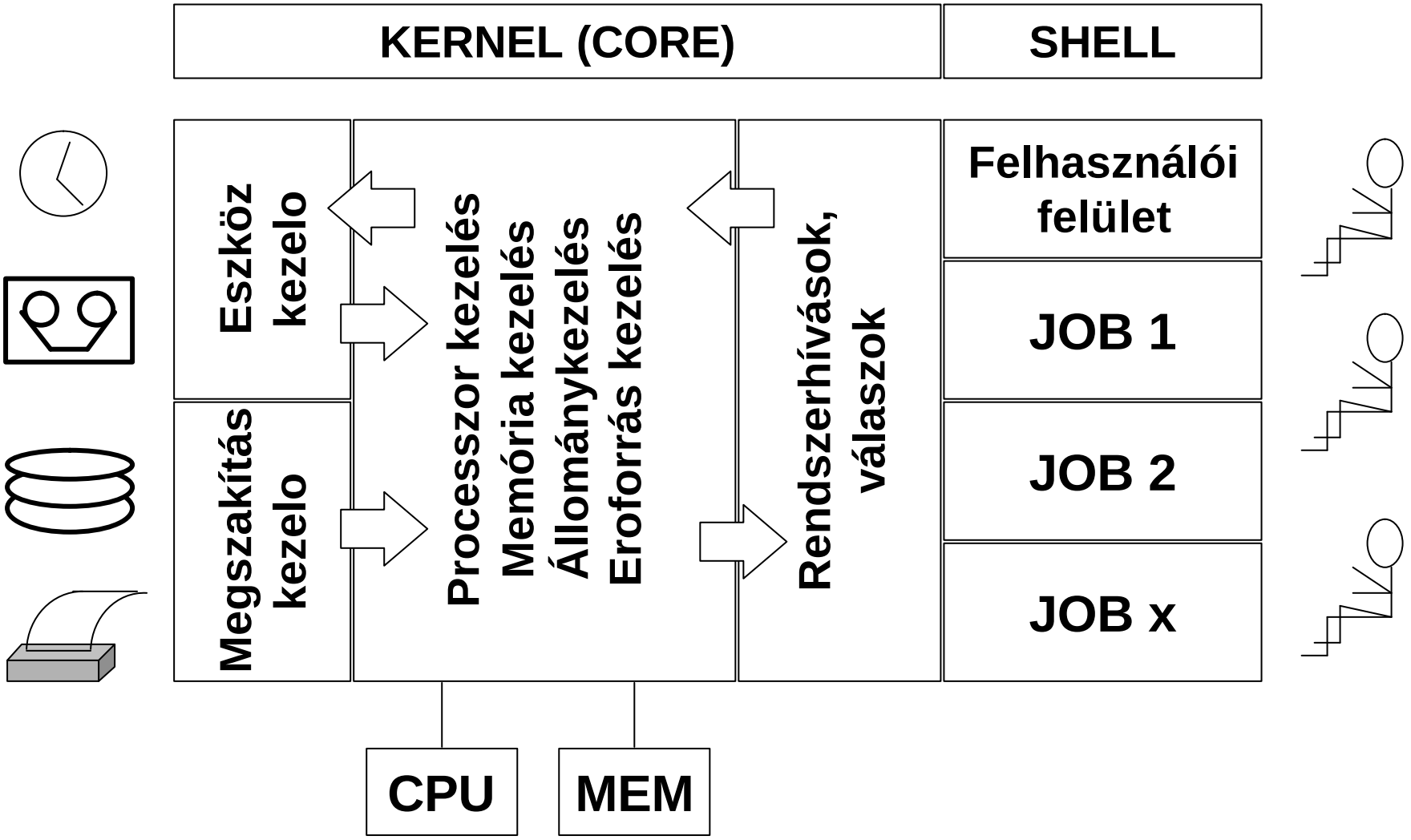
Átlapoló rendszer (pipelined)



Multiprogramozás



Idoosztásos, interaktív rendszerek



Operációs rendszerek feladatai

Eszközkezelés

Megszakítás kezelés

Rendszerhívások, kiszolgálása

Eroforrás kezelés

Memória kezelés

Állomány- és lemezkezelés

(Felhasználói felület biztosítása)

(Program fejlesztési támogatás)

Az interaktív rendszerek új szempontjai

„Emberi” válaszido

– Idoosztás (time sharing)

Felhasználói felület

– Kiegészített parancsnyelv

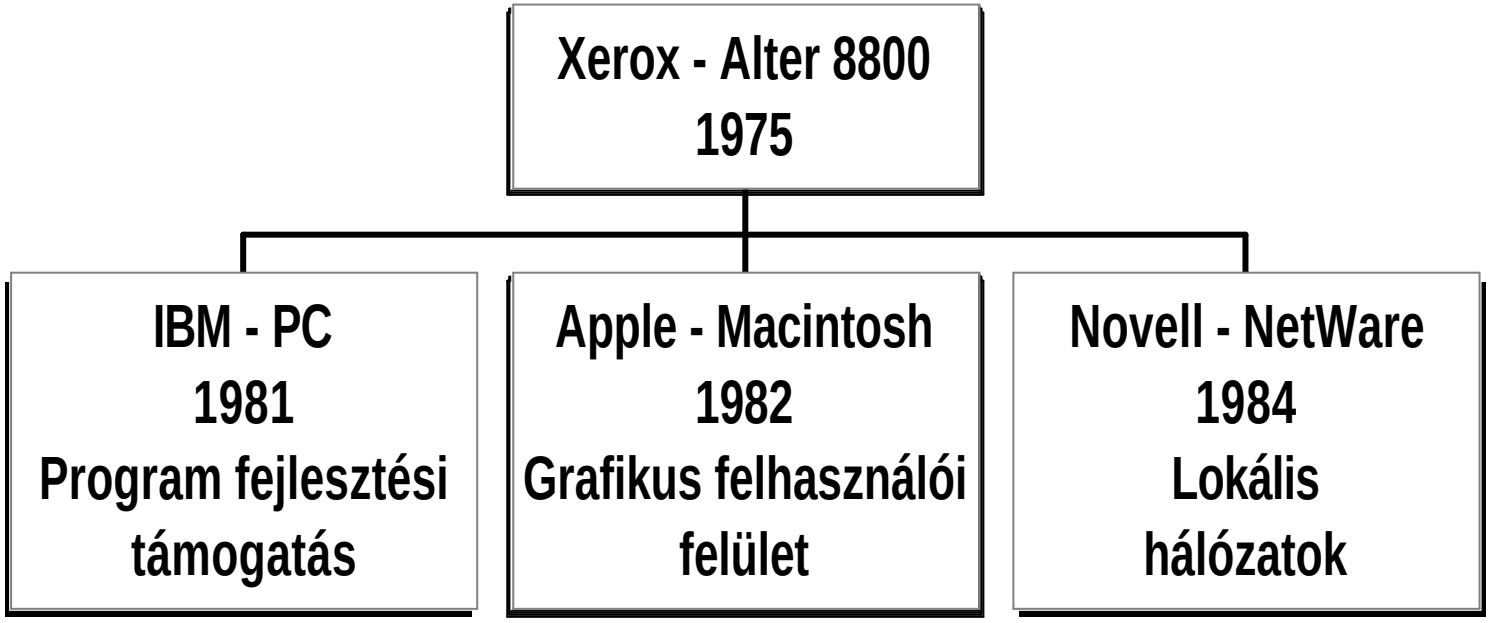
– Barátságos felület

Felhasználói adminisztráció

– azonosítás, naplózás

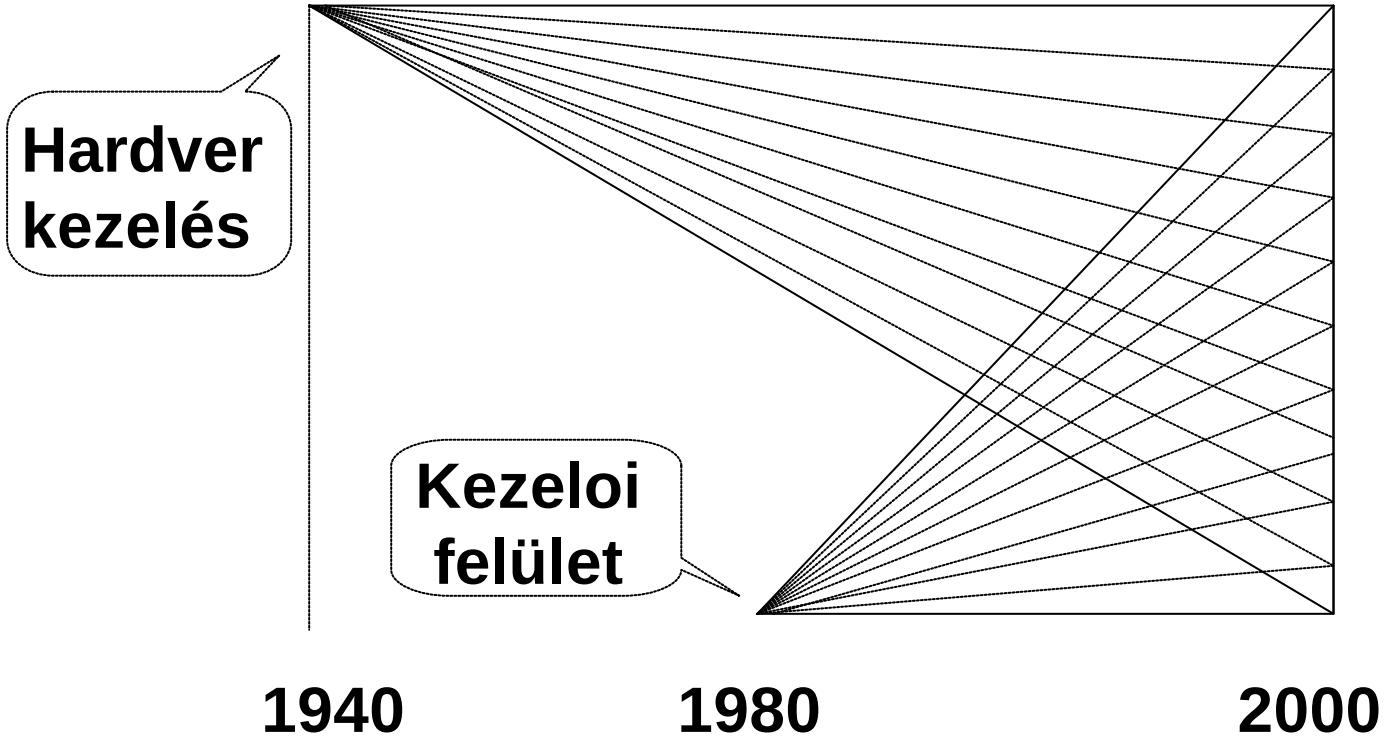
– védelem

Személyi számítógépek

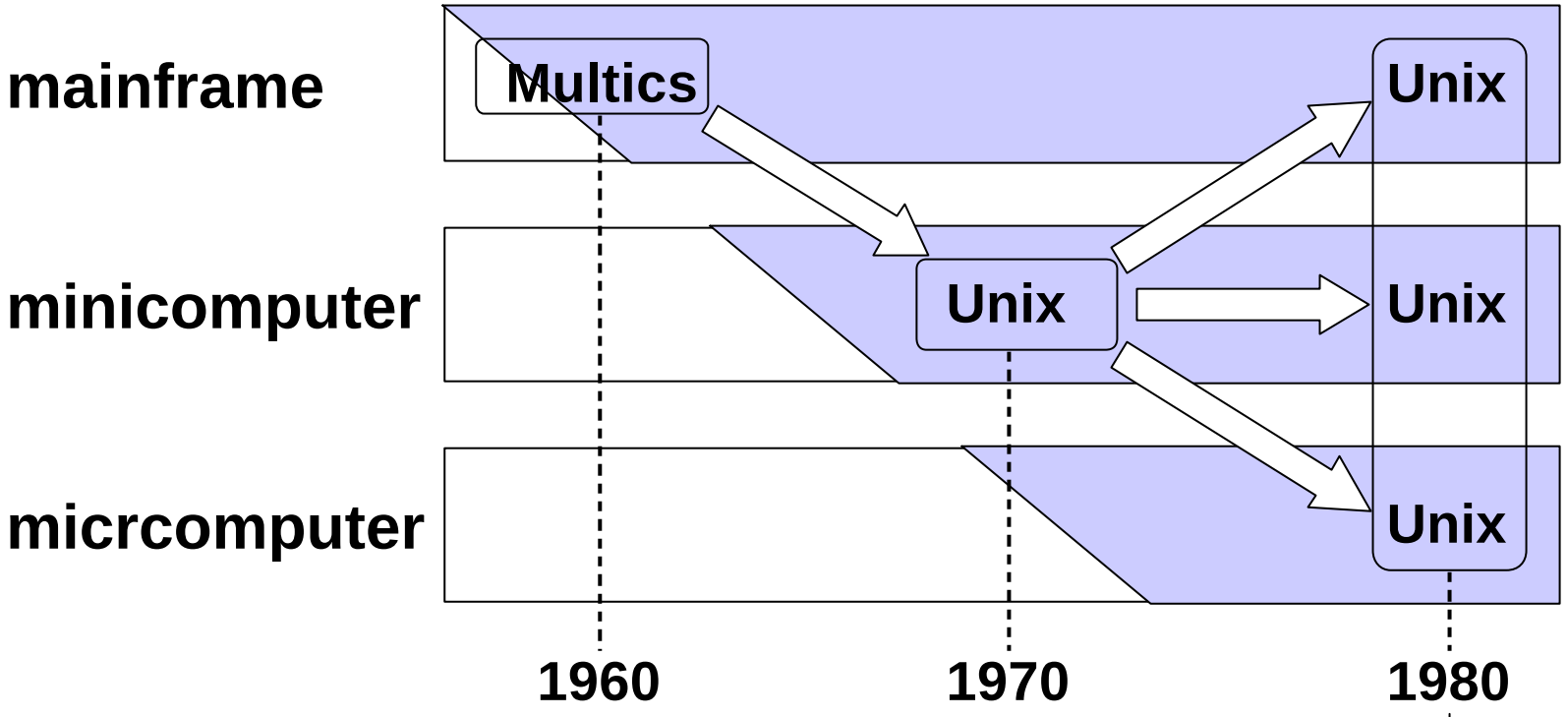


Legfontosabb a felhasználó !

Tendencia



A Unix jelentosége



A közös os
Alkalmazkodó
Szabványteremto

Többprocesszoros rendszerek

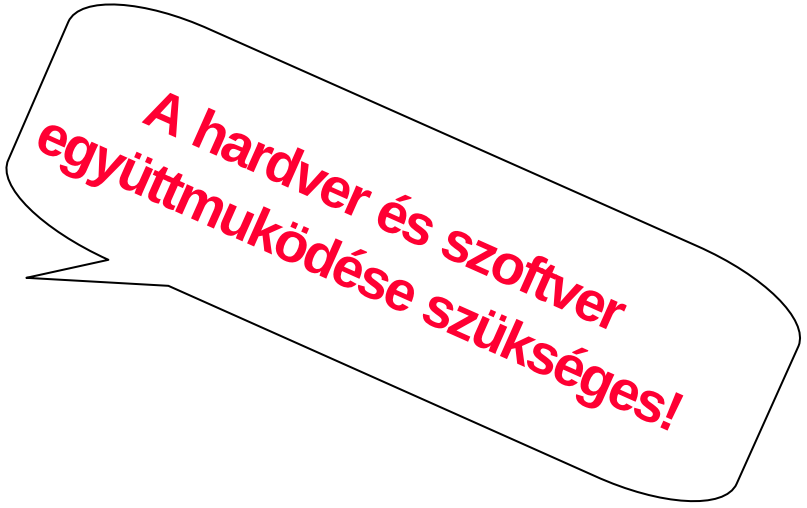
Nagyobb átbocsátó képesség

Eroforrás megtakarítás

Megbízhatóság növekedése (Fault tolerant system)

Szimmetrikus (SMP)

Aszimmetrikus



Elosztott rendszerek (hálózatok)

Rugalmasság

Eroforrásmegosztás

Sebességnövekedés

Megbízhatóság

Kommunikáció

Folyamatok (task, process)

1. „Életre kelt” programok

2. Olyan programok, amelyeknek van folyamatleíró táblája

Folyamatleíró blokk (PCB, TSS)

Folyamat azonosítója

Programszámláló állása

Folyamat állapota

Regiszterek tartalma

Memóriaterület adatai

Perifériák, állományok állapota

Eroforrások

Minden, ami egy folyamat végre-hajtásához szükséges
(memória, processzor,perifériák állományok stb.)

Eroforrások típusai

Megszakítható (preemptív)	Nem megszakítható (non-preemptív)
A folyamatoktól az eroforrás a folyamat vagy eroforrás károsodása nélkül elveheto	Az eroforrás használat félbeszakítása esetén a folyamat vagy eroforrás sérülhet!

Operációs rendszer (Eroforrás szemlélet)

A folyamatok egy olyan csoportja, amely a felhasználói folyamatok között elosztja az eroforrásokat

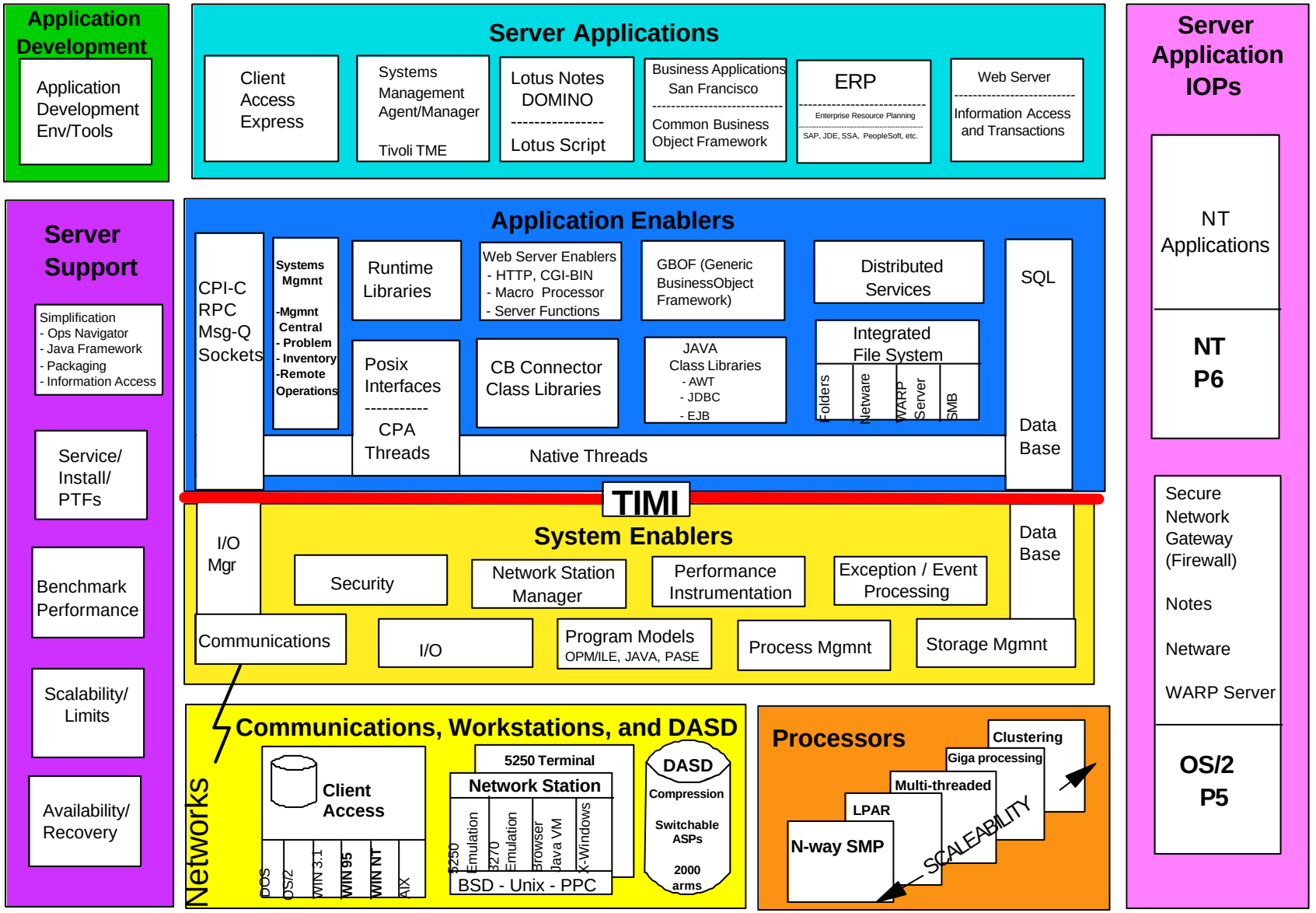
Operációs rendszer (Felhasználói szemlélet)

A folyamatok egy olyan csoportja, amely megkíméli a felhasználókat a hardver kezelés nehézségeitol, kellemesebb alkalmazói környezetet biztosít

Az operációs rendszer szerkezete

<table>
<tr><td colspan="3">Felhasználói programok</td></tr>
<tr><td colspan="3">Program készítési támogatás</td></tr>
<tr><td colspan="3">Felhasználói folyamatok kiszolgálása</td></tr>
<tr><td colspan="2">Rendszerhívások</td><td>Válaszok</td></tr>
<tr><td colspan="2" rowspan="3">Rendszermag
(KERNEL)</td><td>Processzorkezelés</td></tr>
<tr><td>Memóriakezelés</td></tr>
<tr><td>Állománykezelés</td></tr>
<tr><td colspan="2">Eszközmeghajtók</td><td>Megszakítás kezelés</td></tr>
<tr><td colspan="3">Hardver</td></tr>
</table>

AS/400e server Function and Structure



Megszakítások fajtái

Megszakítás (Interrupt)

- A perifériák jelzése a processzor számára (pl adatátvitel vége)

Kivételek (Exception)

- A processzormuveletek során keletkezo hibák esetén (pl rossz címszámítás, osztás 0-val)

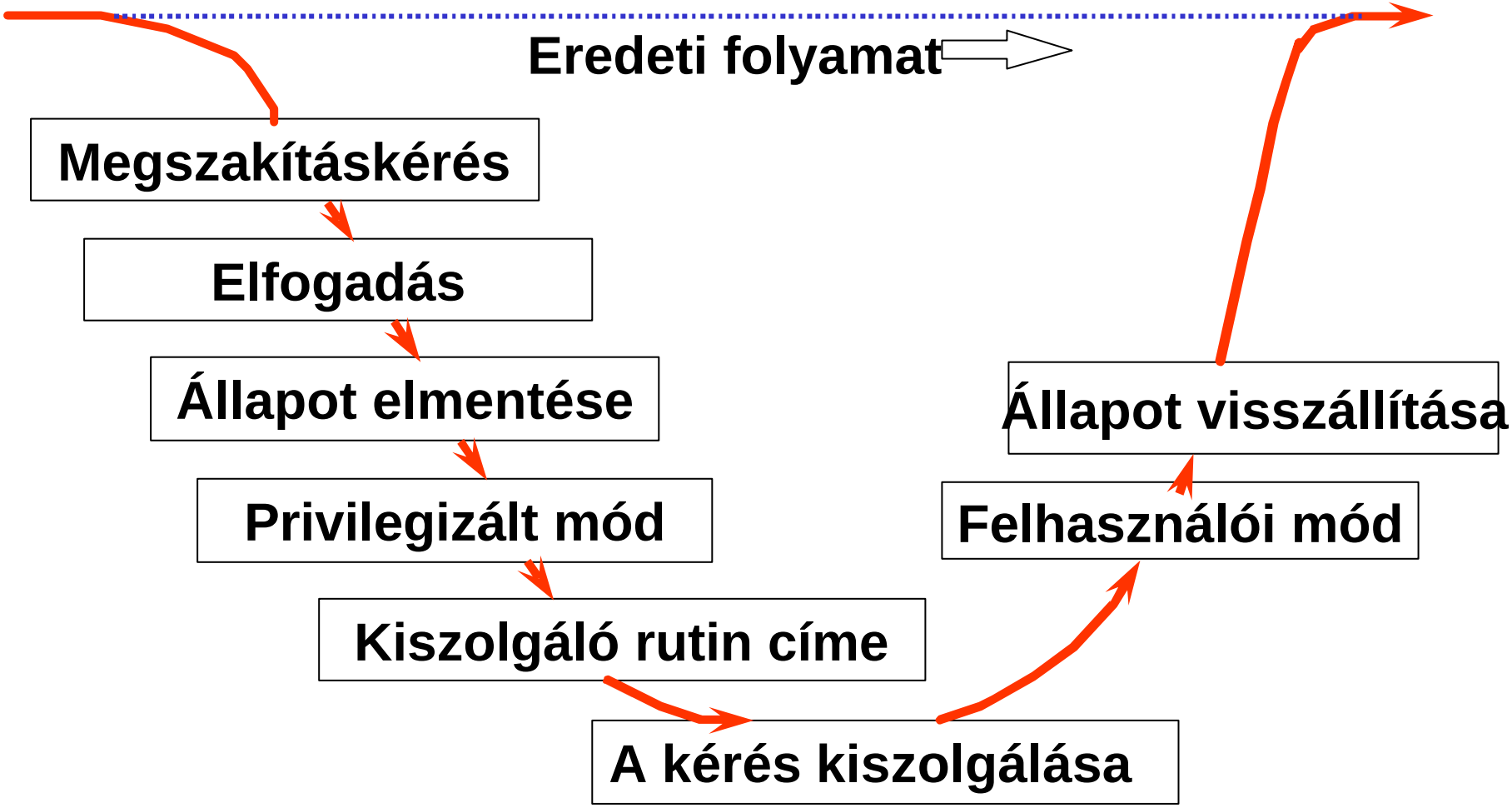
Nem maszkolható megszakítások (NMI)

- súlyos hardver (pl RAM, tápfeszültség) hiba

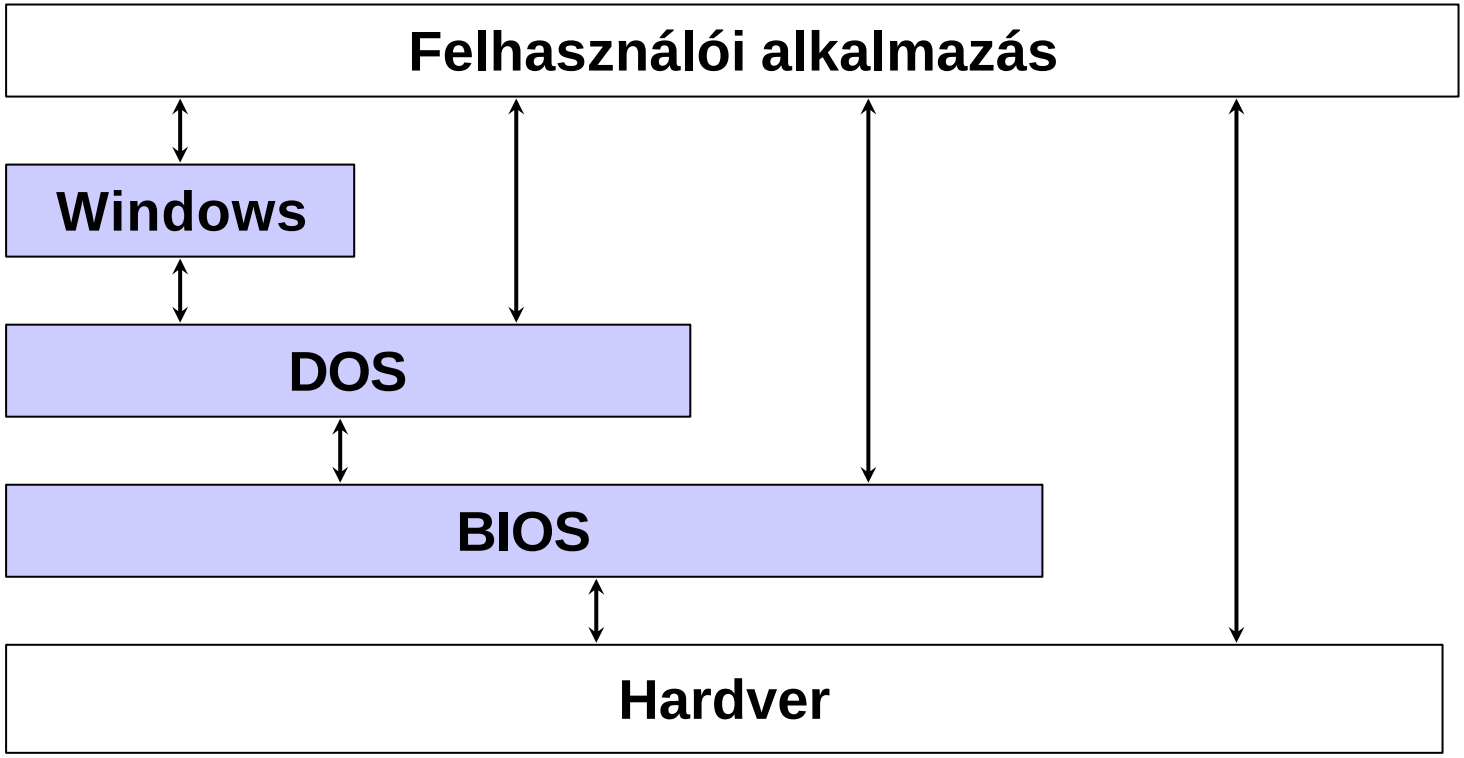
Csapda (Szoftver megszakítás, Trap)

- a felhasználói folyamatoktól érkezo rendszerhívások

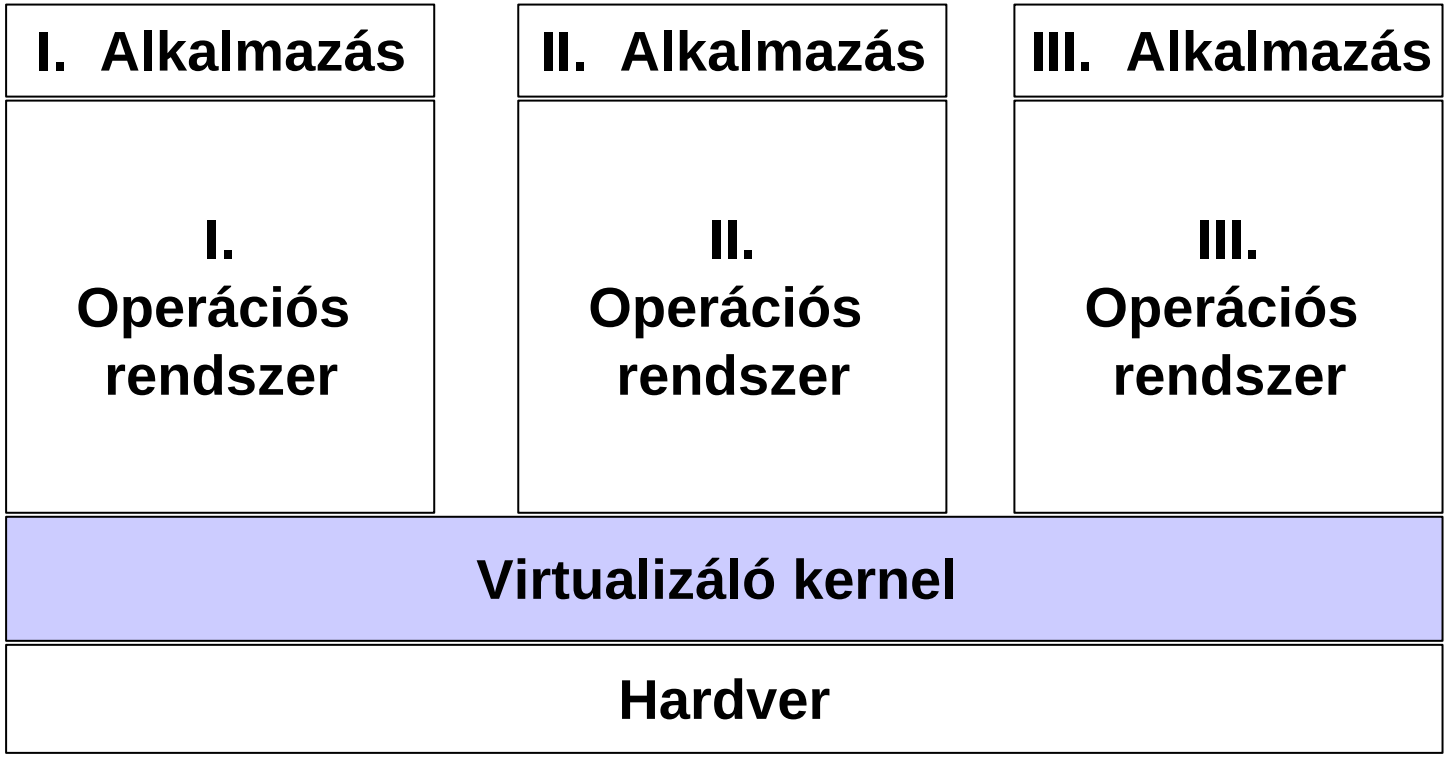
Megszakítás-kezelés lépései



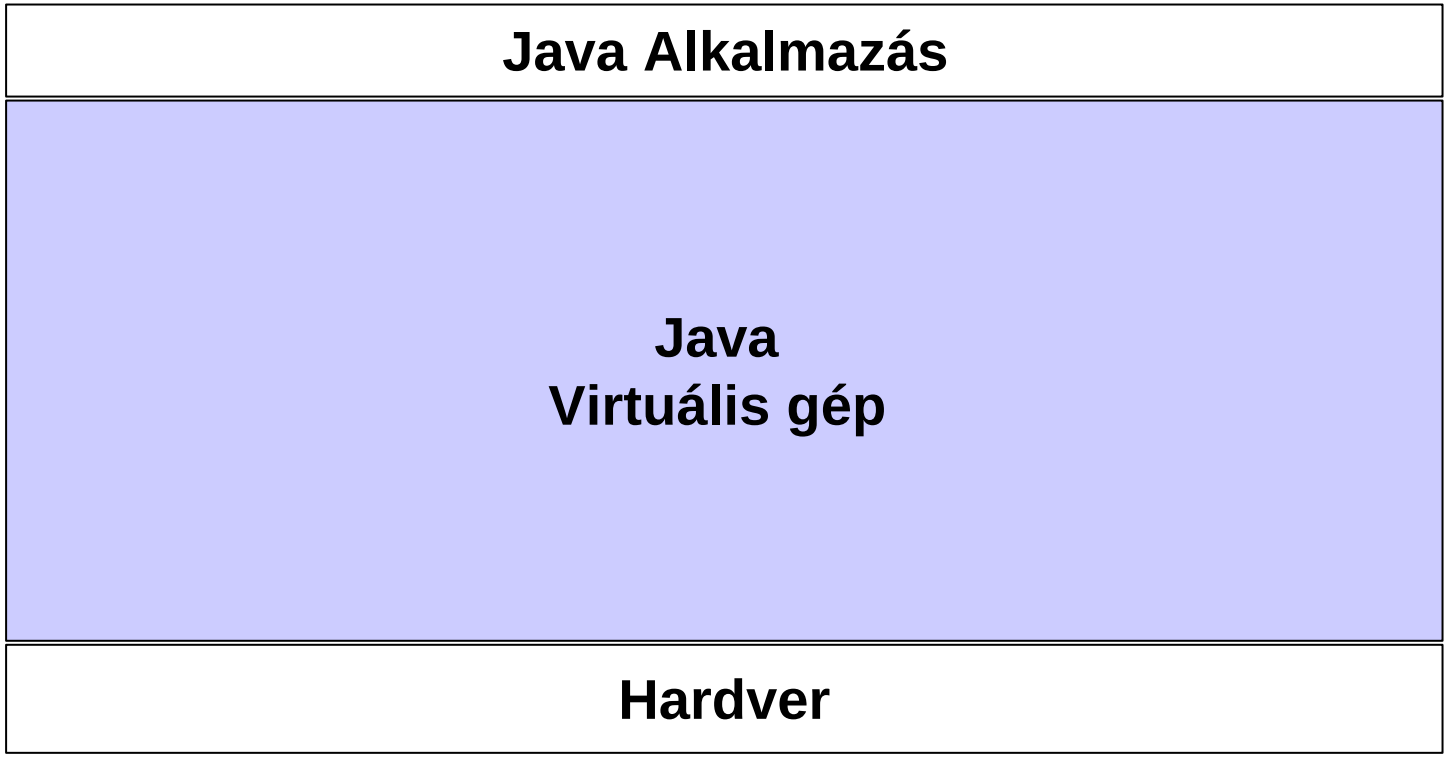
Példa: DOS+Windows



Virtuális gépek: Az IBM VM



Virtuális gépek: JAVA



Összefoglalás

Folyamatok, PCB fogalma

Eroforrás fogalma, alaptípusai

Operációs rendszerek funkciói

Kommunikáció:

- Megszakítások
- Rendszerhívások

A felhasználói felület

- **A felhasználó és a rendszermag**
- **A programozói felület**
- **Segédprogramok, alrendszerek**
- **Jellemzok**

A felhasználói felület részei

A programindítás eszközei

A rendszermag szolgáltatásainak elérése

A programozói felület

Alapveto segédprogramok, alrendszerek

A perifériák beállítási lehetoségei

„Kézi” beállítások	**Config.sys**
Automatikus beállítások	**Plug and Play**
Félautomata beállítások	**Scan for Devices**

A memória beállítási lehetoségei

Bufferek száma, mérete (hálózat, lemez)

Leíró táblák száma (pl. FCB)

Lemezgyorsító tár (cache)

Felhasználási módok (EMS, HMA,XMS)

A programozói felület

API (Application Programming Interface)

SDK (Software Development Kit)

Rendszerhívások = Függvények

DOS - kb. 100 alapveto függvény

Windows - kb. 1000 magasszintu fv.

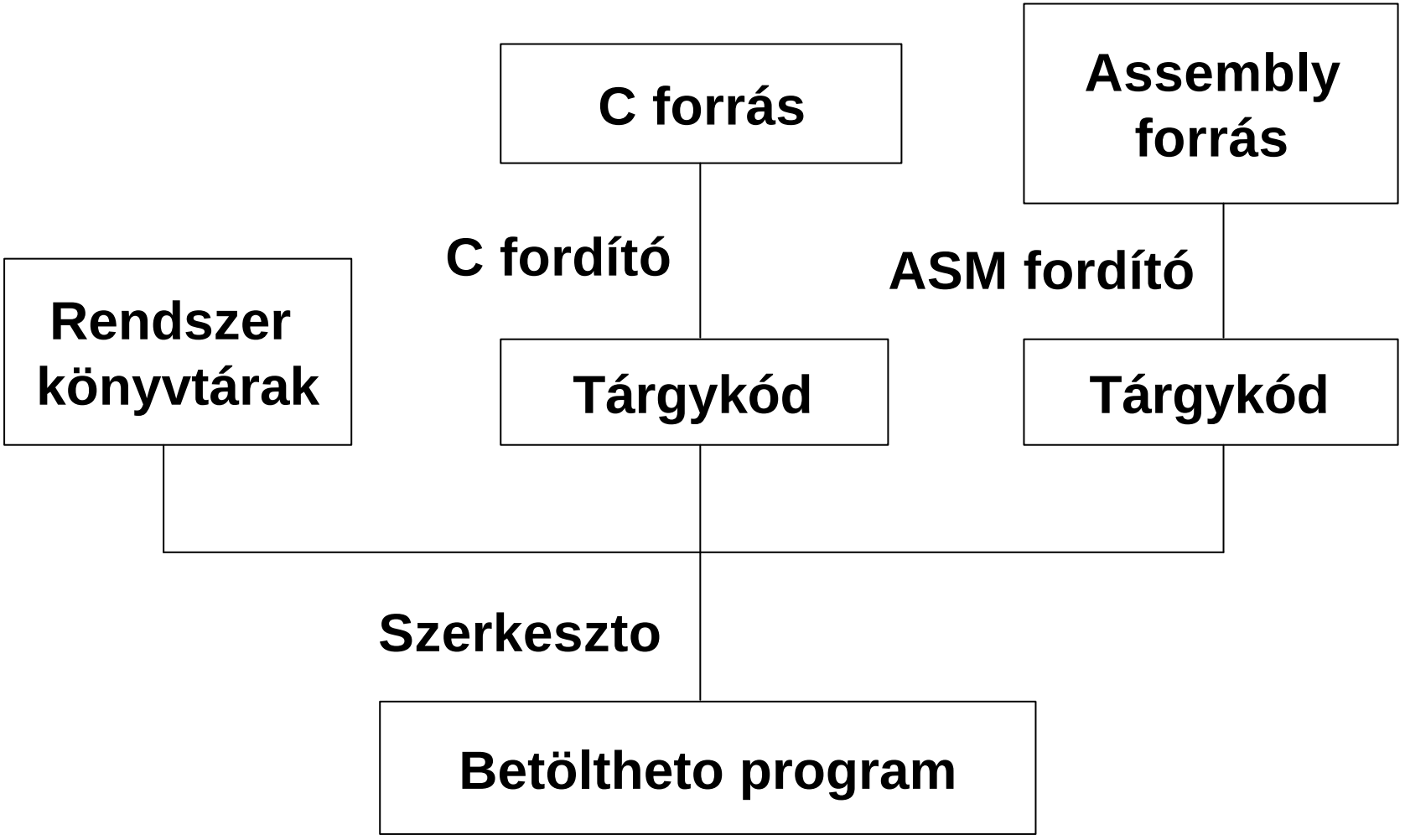
Programkészítés

Forráskód (algoritmus)

Tárgykód (könyvtárak)

Betöltheto kód (program)

Programok készítése



Parancsértelmezo

Elnevezések

(A lényegesnek tartott funkció szerint)

- Shell (DOS-Shell, Bourne-shell)
- Command interpreter (command.com)
- Monitor (Novell)
- Manager (Program, Fájl, Nyomtató)

A parancsértelmezo feladatai

Programindítás

Fájlkezelés

Programkörnyezet beállítása

Folyamatok futásának ellenorzése

Vezérlési szerkezetek

Programindítás

Közvetlen indítás (név beírása, load, run)

Közvetett elérés

Keresési útvonalak

Láncolt (kötegelt) futtatás (*.bat, *.ncf)

Automatikus programbetöltés

A program környezete

Paraméterek

Kapcsolók

Átirányítási adatok

Környezeti változók

Alrendszerek

Állománykezelo

Programfejlesztoi

Adatbázis kezelo

Kommunikációs

Felhasználói felület jellemzoi

Könnyu legyen megtanulni

Méretezheto legyen

Lehessen visszavonni

Törölni lehessen a muveleteket

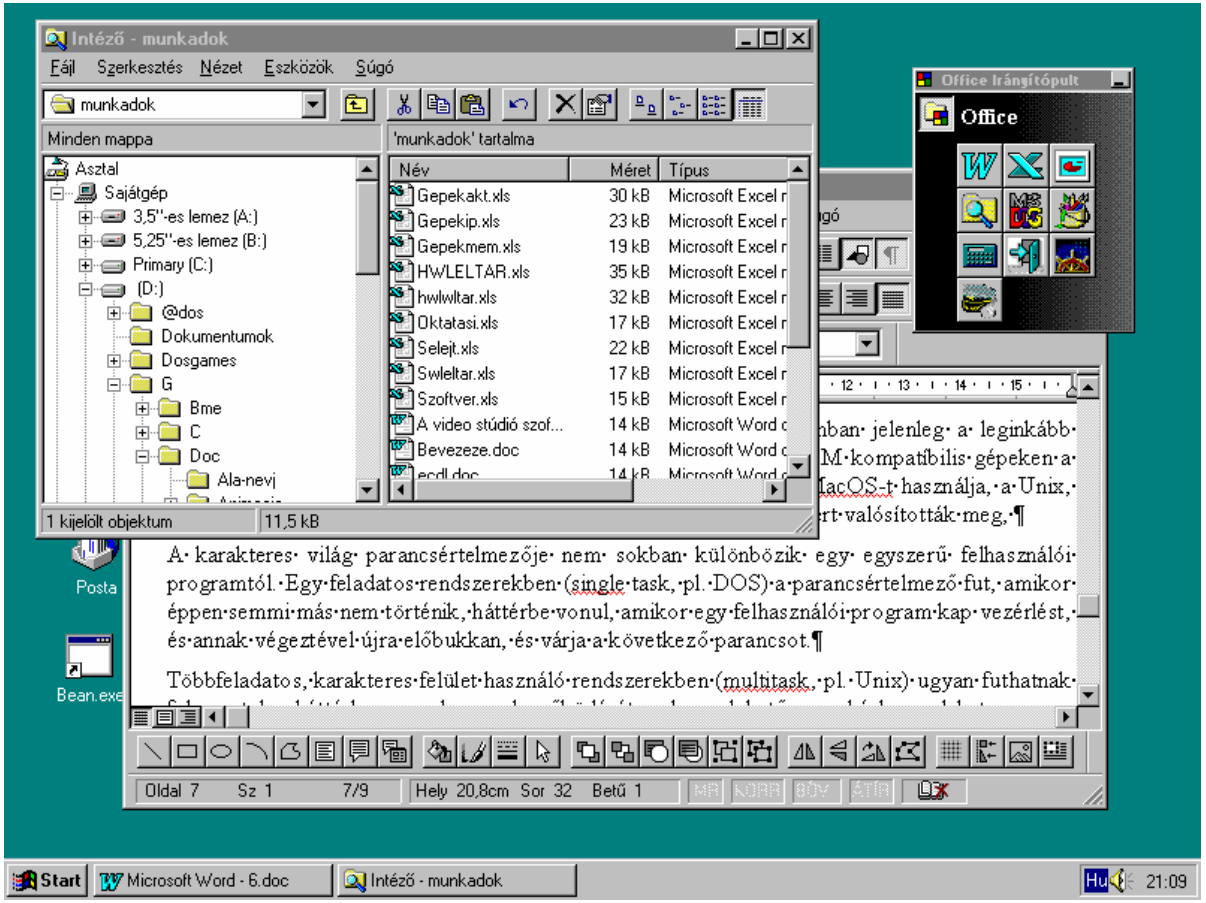
Többszintu súgó rendszer

Hasonlítson az élo nyelvhez

Minden parancsra legyen válasz

Hasonló funkciók hasonló módon

A Windows95 néhány ablaka



Ablakozó rendszerek

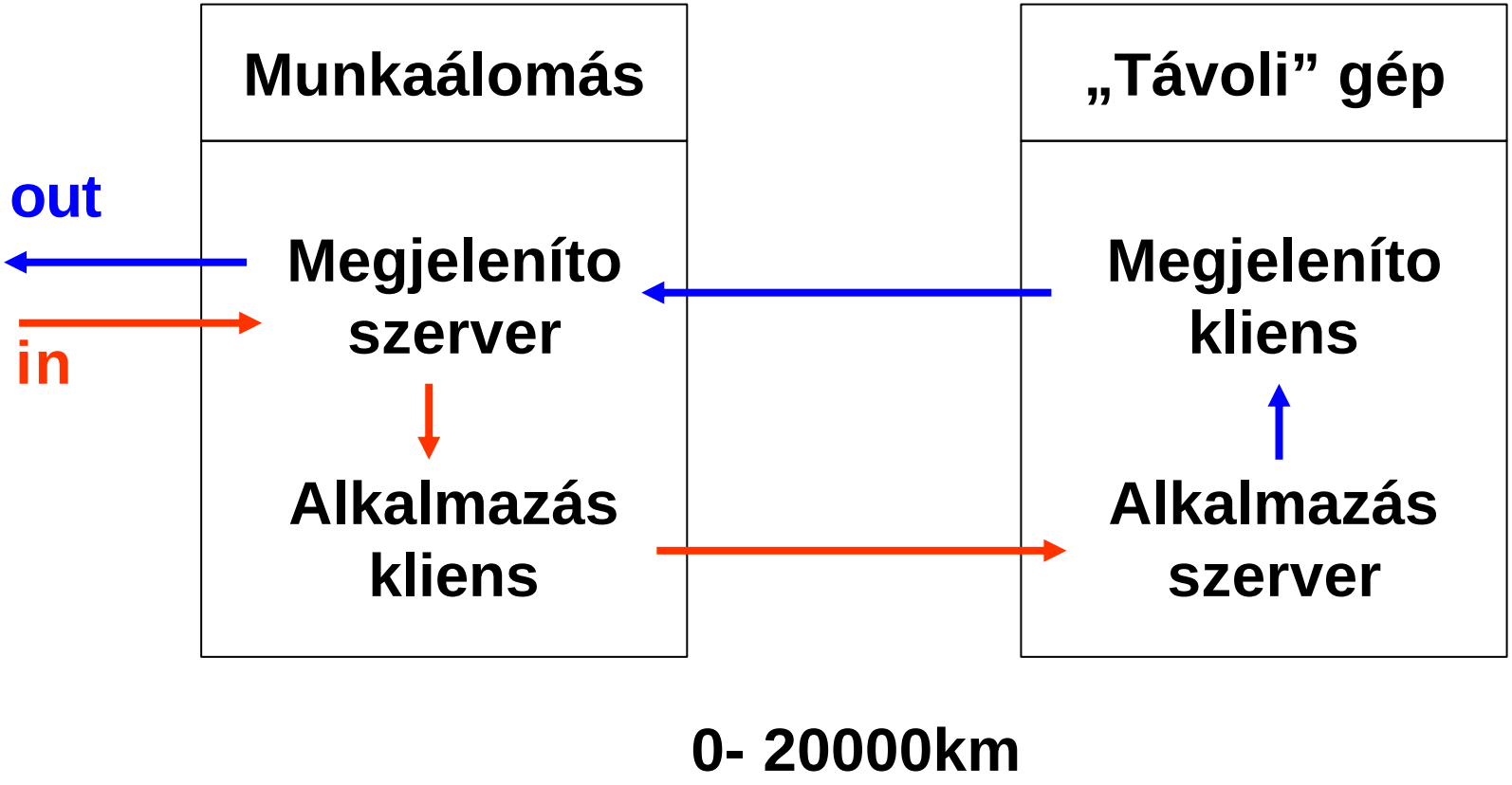
Követelmények

- A multitask környezet miatt megoldandó az események címzettjeinek felismerése (aktív ablak, fókusz)
- Eszközfüggetlen muködés biztosítandó
- Az adatforgalom a kliens és szerver között csökkentendo

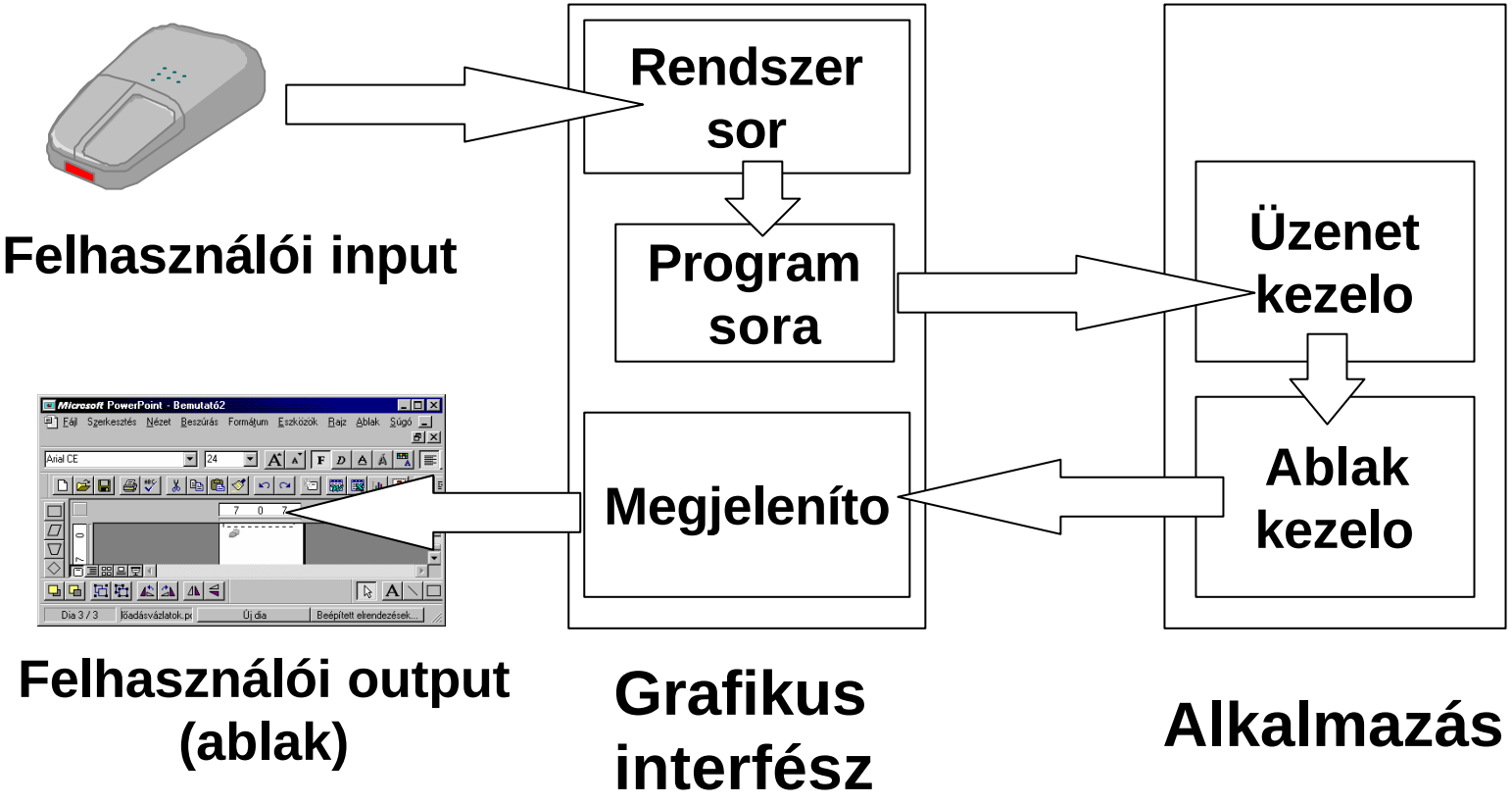
Megoldás

- Szerver-kliens architektúra
- X-szerver, X-kliens, X.11, X-window

X-window architektúra



Üzenetvezérlés



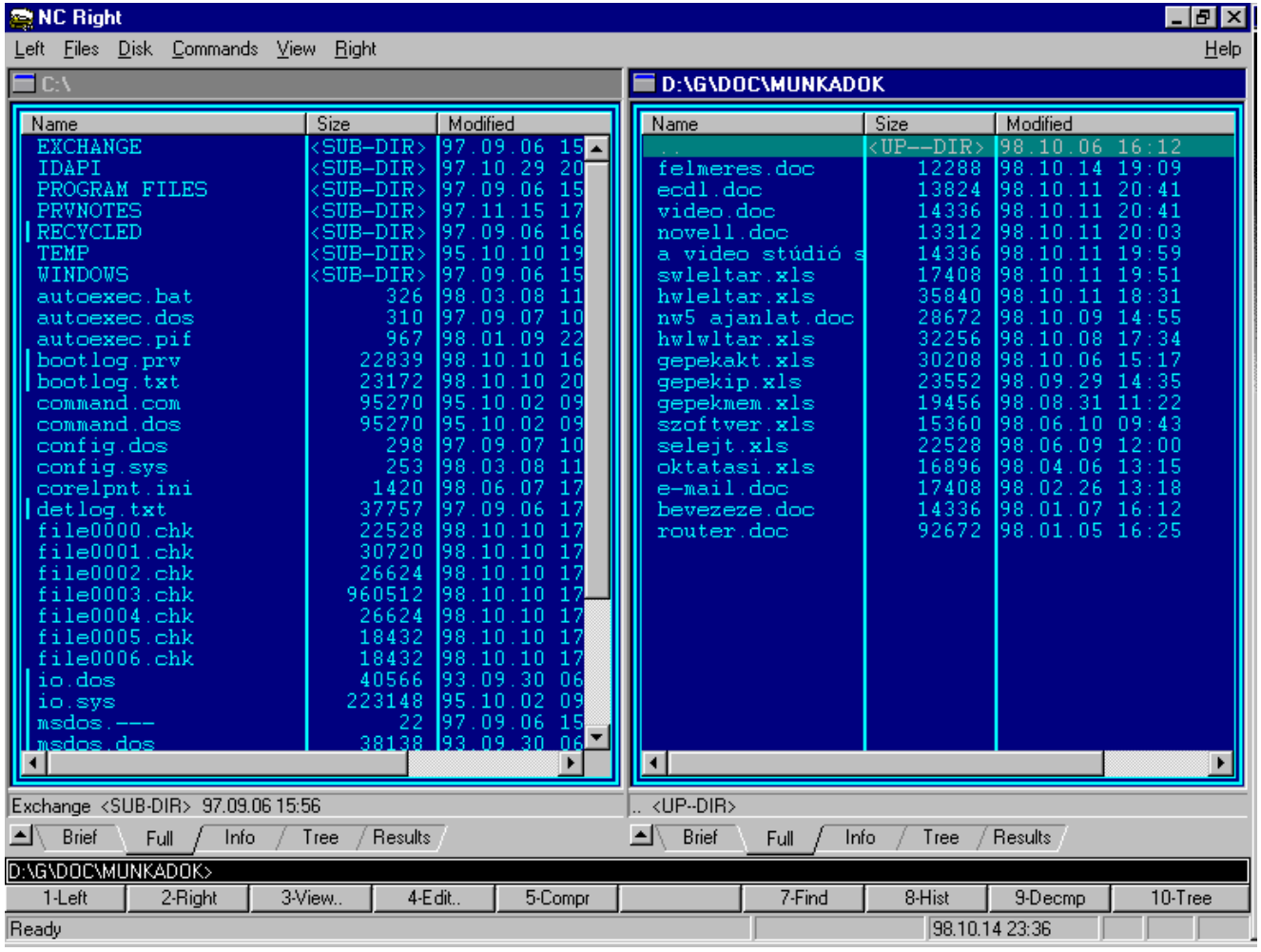
Eszközfüggetlen muködés

– A megjeleníto szerver az alkalmazás szabványos, eszközfüggetlen üzenetei alapján állítja össze a megjelenítendo képet
– Az alkalmazásnak nem kell ismernie a munkaállomás perifériáit, a megjelenítésrol a helyi szerver gondoskodik (GUI)
– A GUI az alapveto objektumokat maga állítja elo az alkalmazástól kapott paraméterek alapján

Adatforgalom csökkentése

- Nem mozognak az átviteli csatornán nagyméretu képernyotartalmak, csak az ezek összeállítására vonatkozó utasítások
- A GUI az alapveto objektumokat maga állítja elo az alkalmazástól kapott paraméterek alapján
- Az ideiglenesen nem használt felületek tartalma tárolódik az X-szerveren
- Lehetoség szerint csak a változásokra vonatkozó utasítások mozogjanak

A Norton Commander



Összefoglalás

Felhasználói felületek

- célja
- beállítási lehetoségek
- programindítás

Programkészítés lépései

Grafikus, üzenetvezérelt felületek

Fájlok, katalógusok

- **Fájlnevek, jellemzok**
- **Közvetett hivatkozások**
- **Katalógus szerkezetek**
- **Hozzáférési jogok**
- **Fájlok elhelyezése**
- **Muveletek állományokkal, katalógusokkal**

File = Állomány

**Adatok egy olyan csoportja,
melyre együttesen,
egy névvel hivatkozhatunk**

Fájltípusok

Ideiglenes rendszerállományok
(pl. virtuális memória)

Adminisztratív állományok
(pl. katalógusok)

Felhasználói állományok

Tipikus fájlnevek

DOS	**EZEREGY.DOC**
UNIX	**Az.Ezeregy.Ejszaka.Mesei.DOC**
Win 95 (hosszú)	**Az ezeregy éjszaka meséi.doc**
Win 95 (rövid)	**AZEZER~1.DOC**
VMS	**EZEREGY.DOC;4**

Helyettesíto karakterek

	JÓ	NEM JÓ
LEVI?.TXT	LEVI1.TXT LEVI2.TXT	LEVI10.TXT
*NI.DOC	DANI.DOC ZOKNI.DOC	DANO.DOC
[TD]OBOZ *(unix!)*	TOBOZ DOBOZ	KOBOZ doboz

Fájl jellemzok

Fáljnév

Méret

Utolsó módosítás ideje

Attribútumok

[Hozzáférési jogok]

Fizikai elhelyezkedés

Jogosultságok típusai

Olvasás (Read - R)

Írás (Write - W)

Létrehozás (Create - C)

Végrehajtás (eXecute - X)

Törlés (Erase - E)

Jellemzok módosítása (Modify - M)

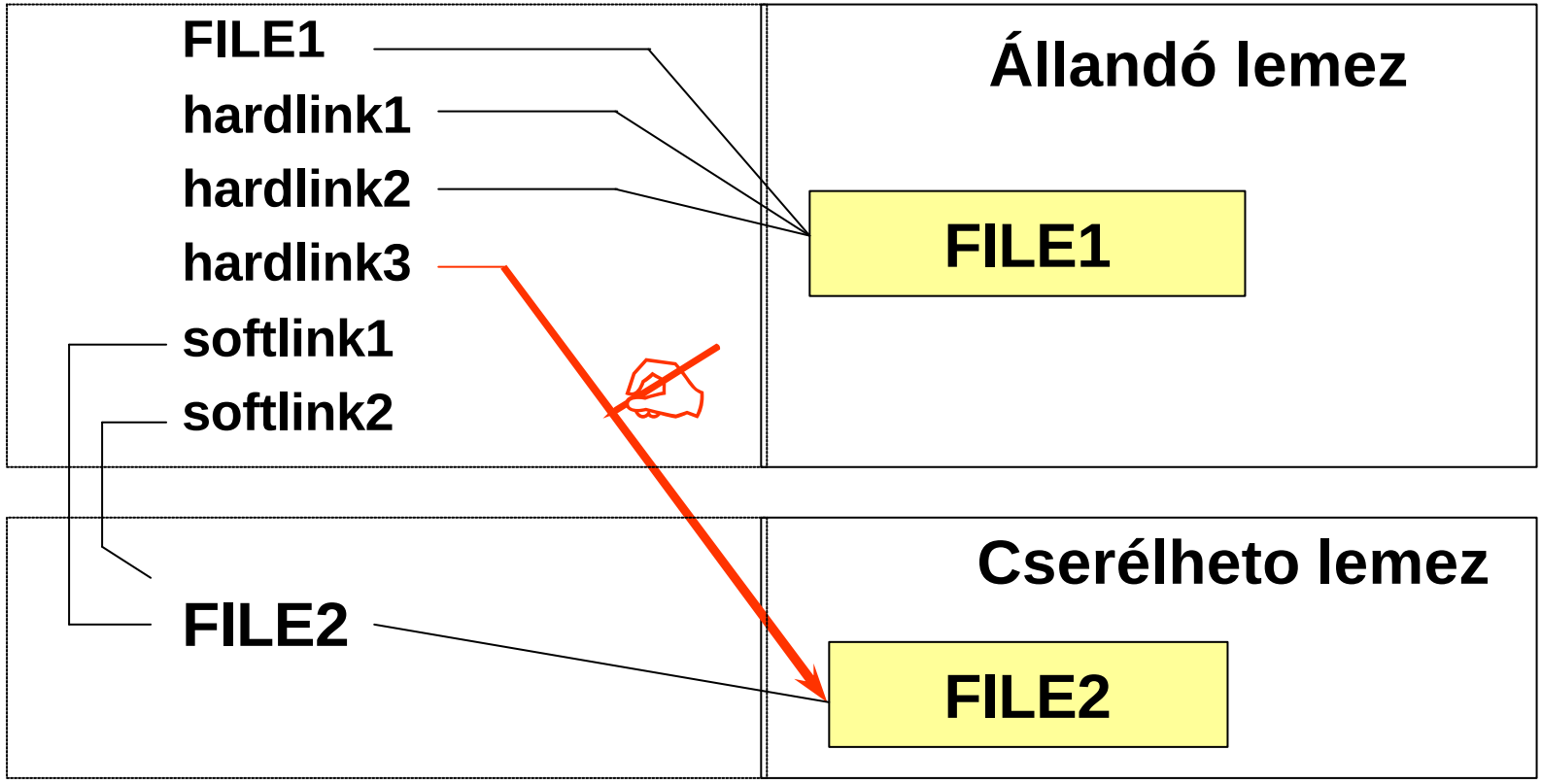
Hozzáférés módosítása (Access control - A)

Katalógus = Directory

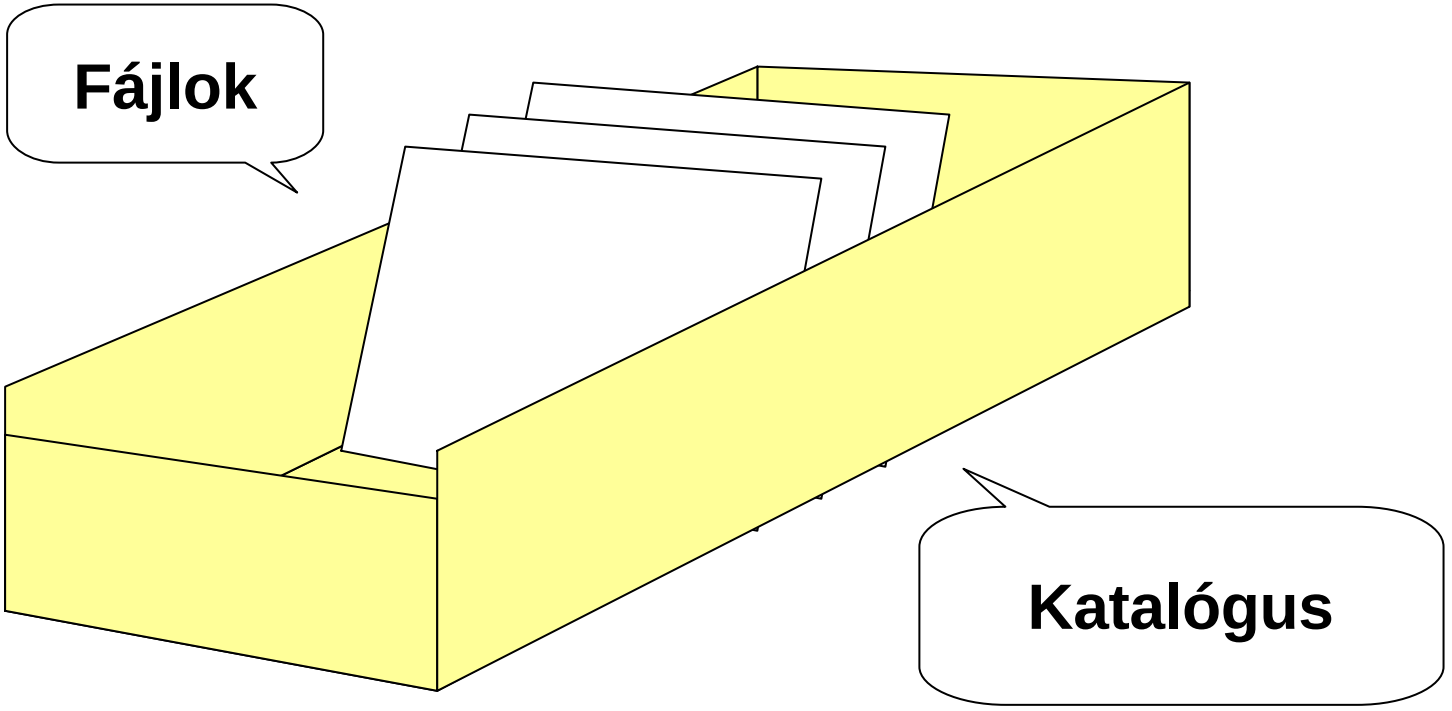
(“*könyvtár*”)

Olyan speciális állomány, melynek tartalma a fájlok nevét és jellemzoit tartalmazó rekordok listája

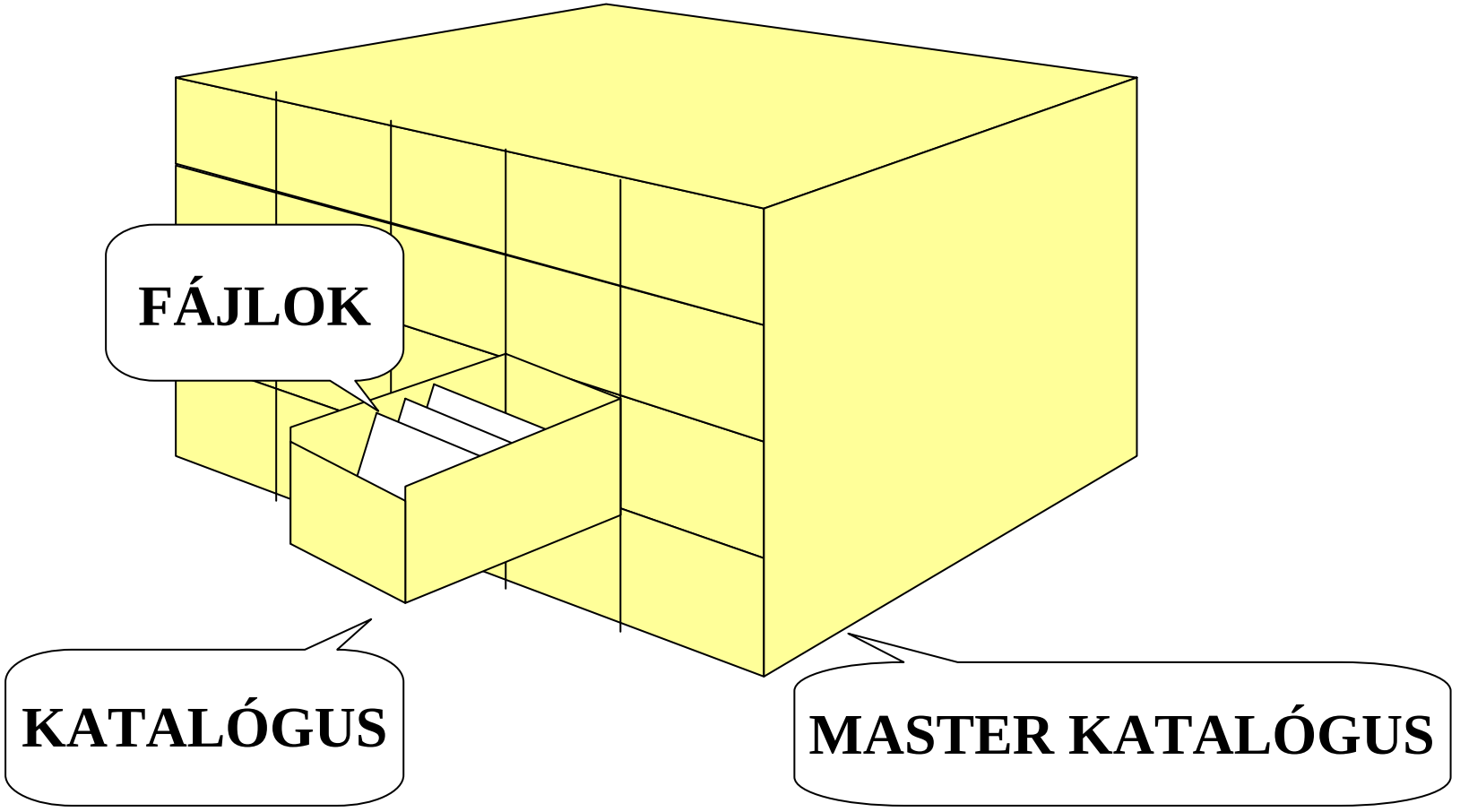
Láncolás


FILE1
hardlink1
hardlink2
hardlink3
softlink1
softlink2
Állandó lemez
FILE1
FILE2
Cserélheto lemez
FILE2

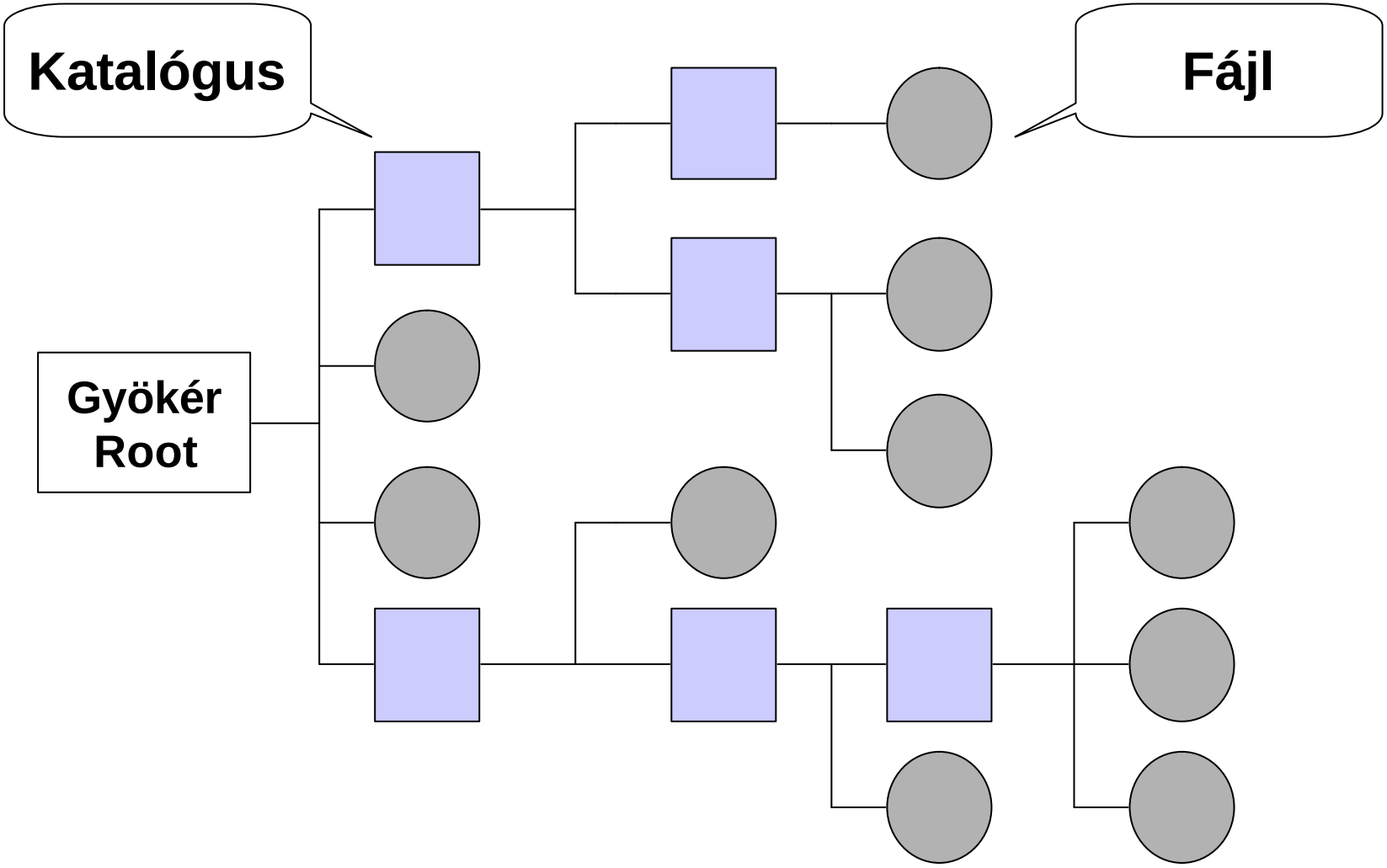
Egyszintu katalógus



Kétszintu katalógus



Fa struktúra



Címzés a fájlrendszerben

Abszolút címzés

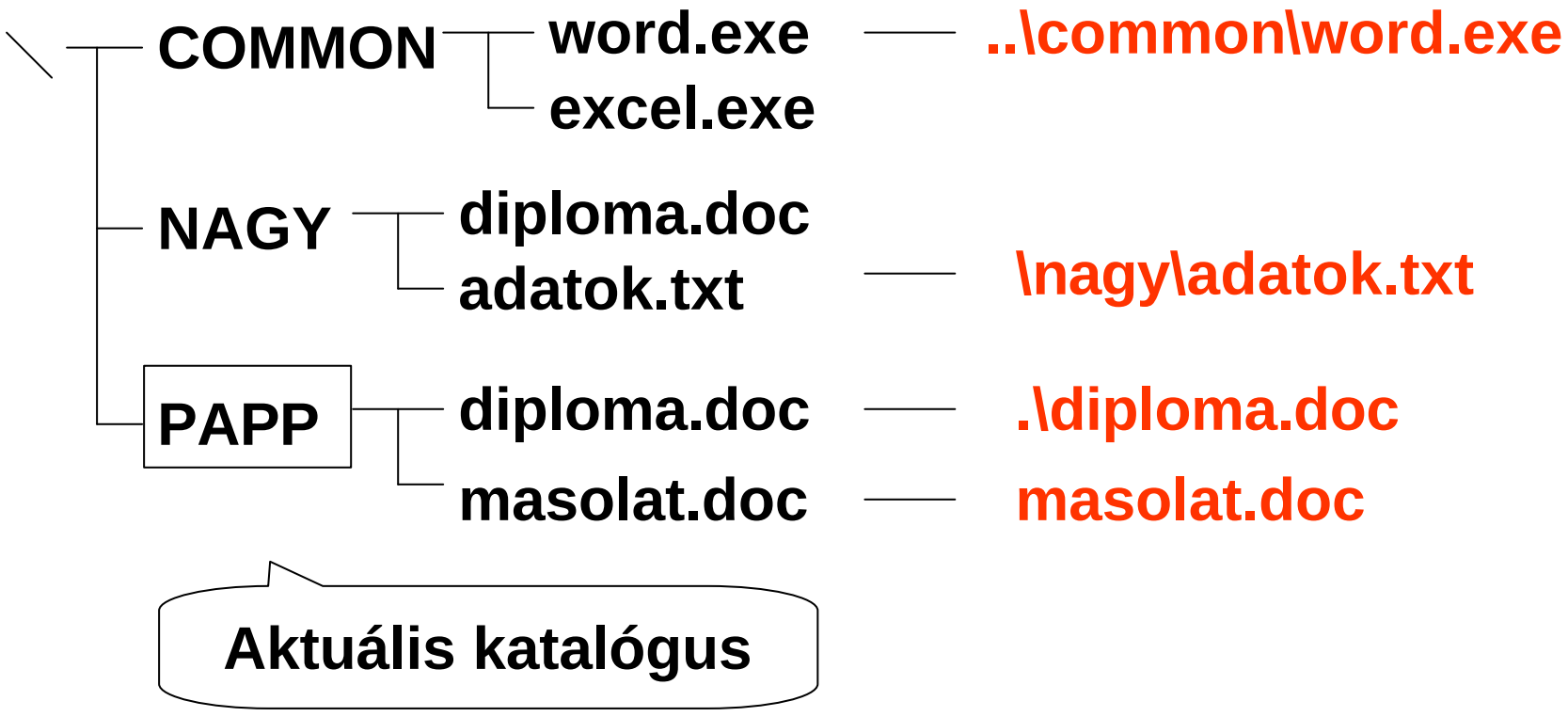
– **kiindulópont a gyökér, a cím ’\’-rel kezdodik**

Relatív címzés

– **kiindulópont az aktuális katalógus**

– **’..’ a szülo, ’.’ az aktuális katalógus neve**

Példák fájl elérésekre



Fájlok elhelyezkedése

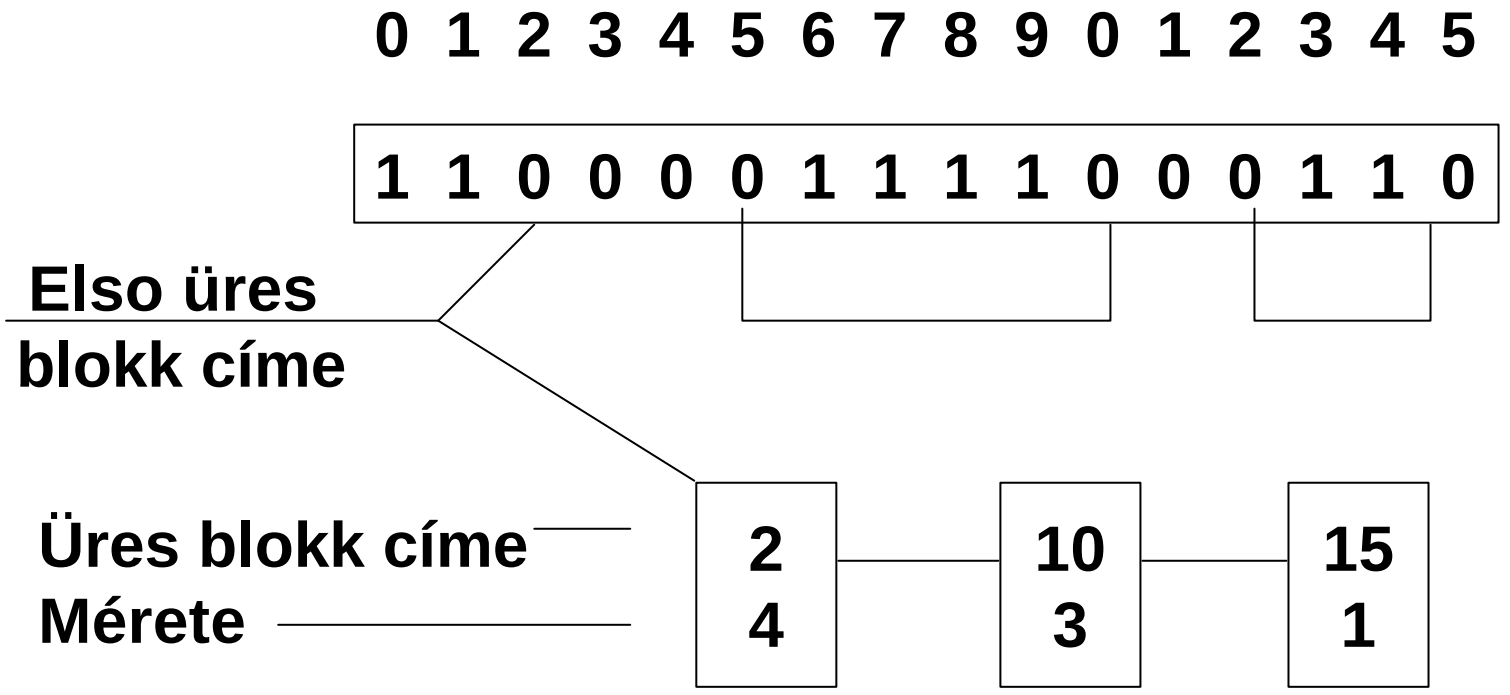
Szabad helyek nyilvántartása

Folytonos: first, worst, best

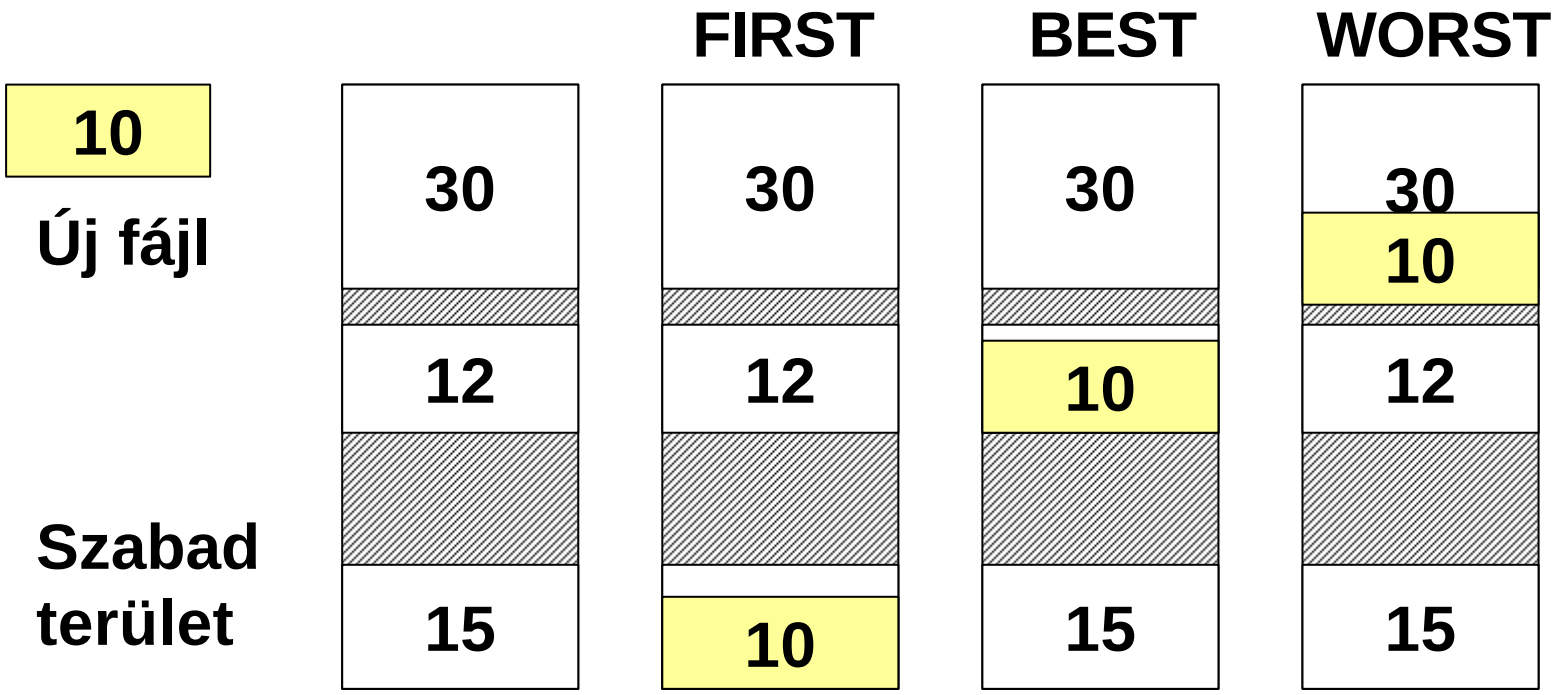
Láncolt: FAT

Indexelt: INODE

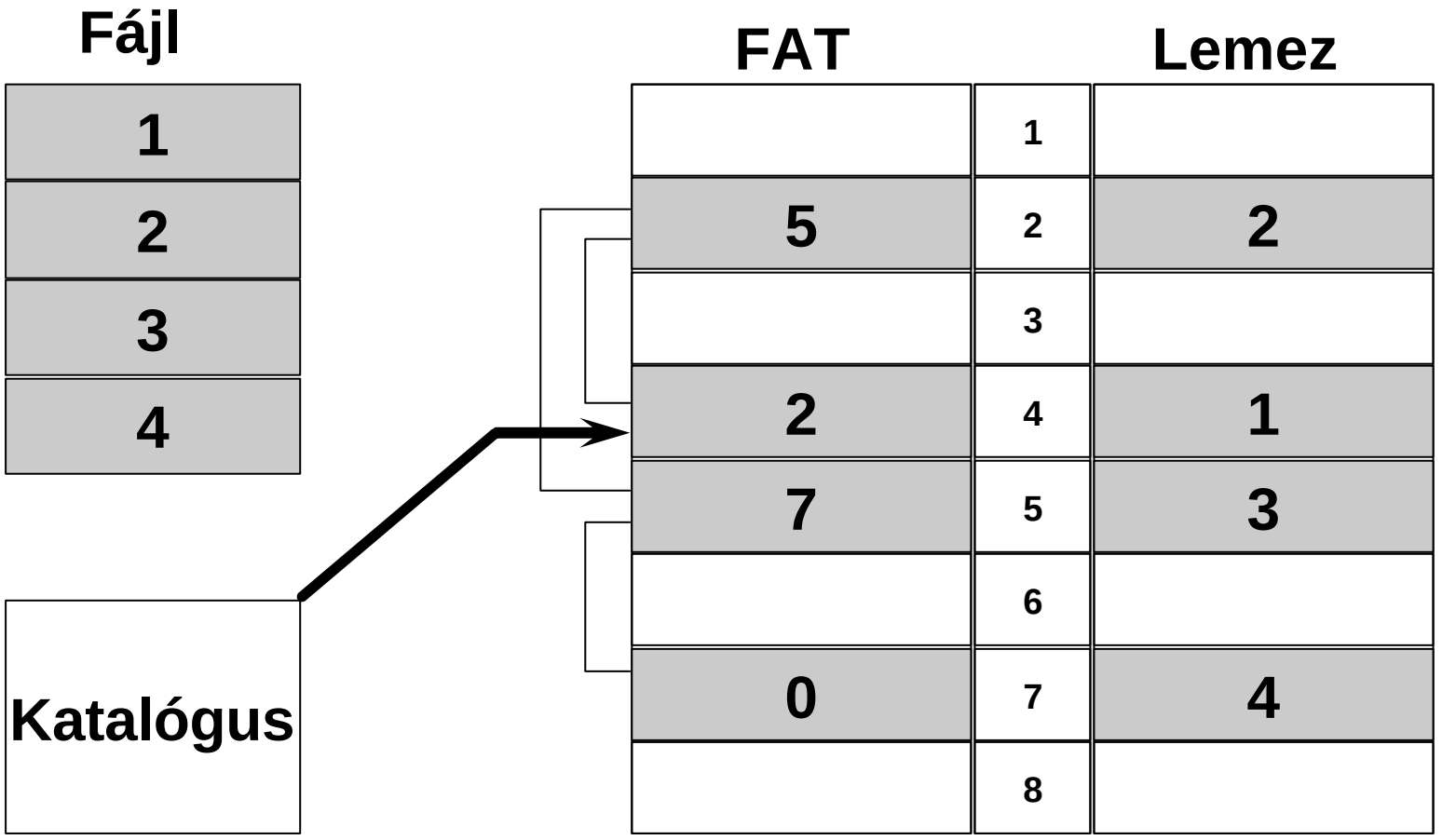
Szabad helyek nyilvántartása



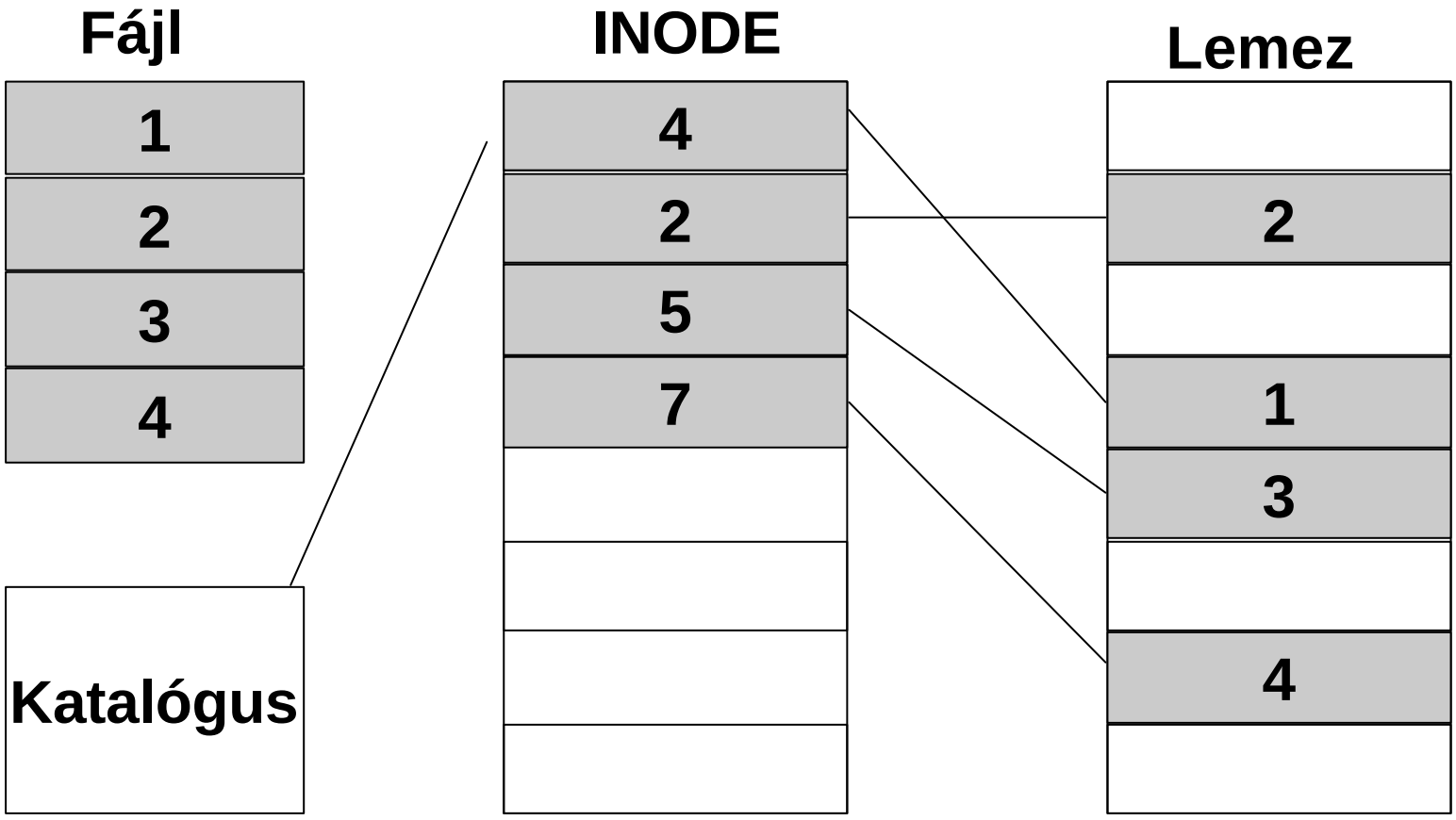
Folytonos elhelyezés



Láncolt elhelyezés



Indextábla alkalmazása



Muveletek állományokkal

Létrehozás

Keresés katalógusban

Írás, olvasás

- **megnyitás**
- **pozicionálás**
- **írás-olvasás**
- **lezárás**

Törlés

Összefoglalás

Fájlok szerepe, jellemzoi

Elnevezések,

- célja
- beállítási lehetoségek
- programindítás

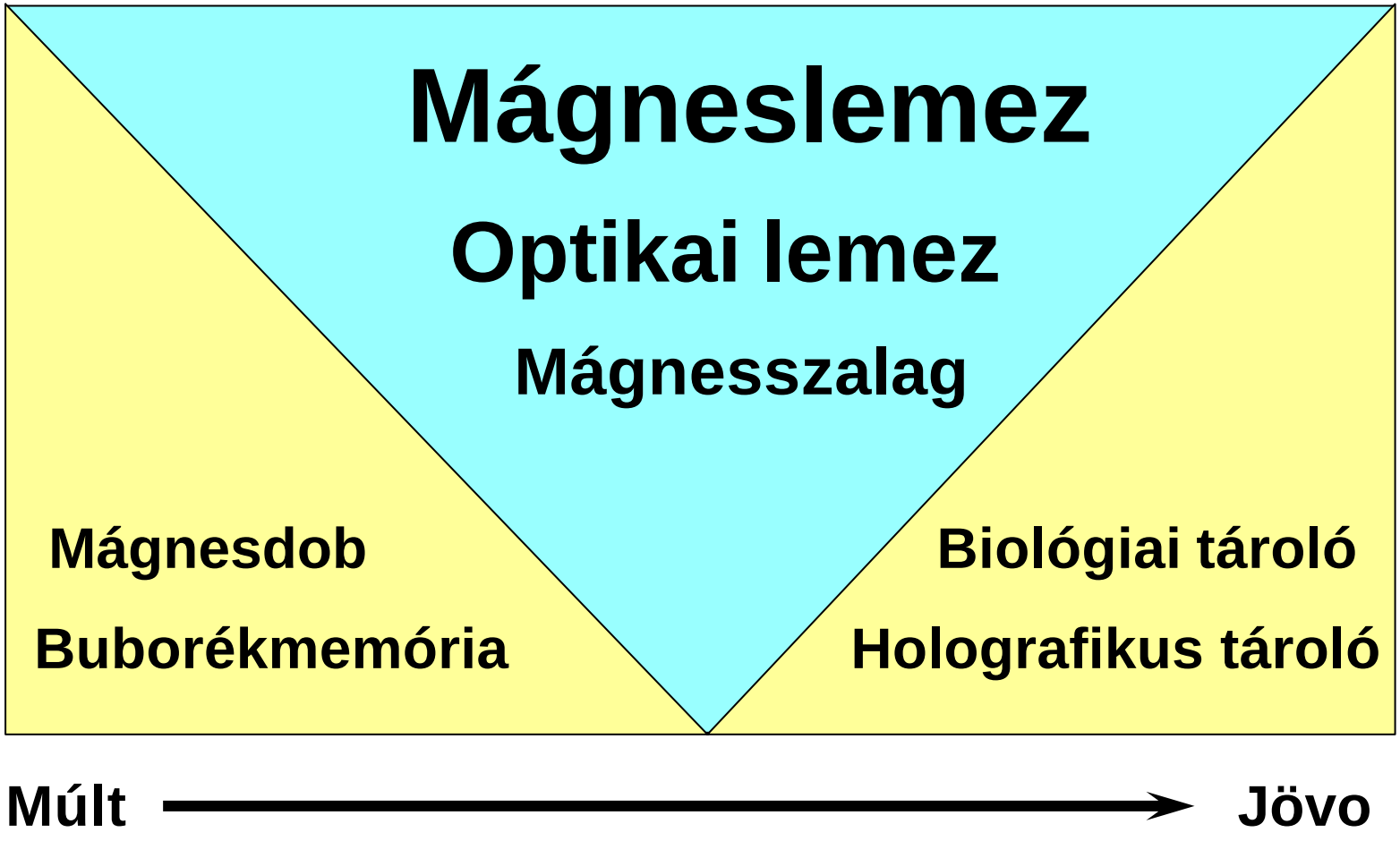
Programkészítés lépései

Grafikus, üzenetvezérelt felületek

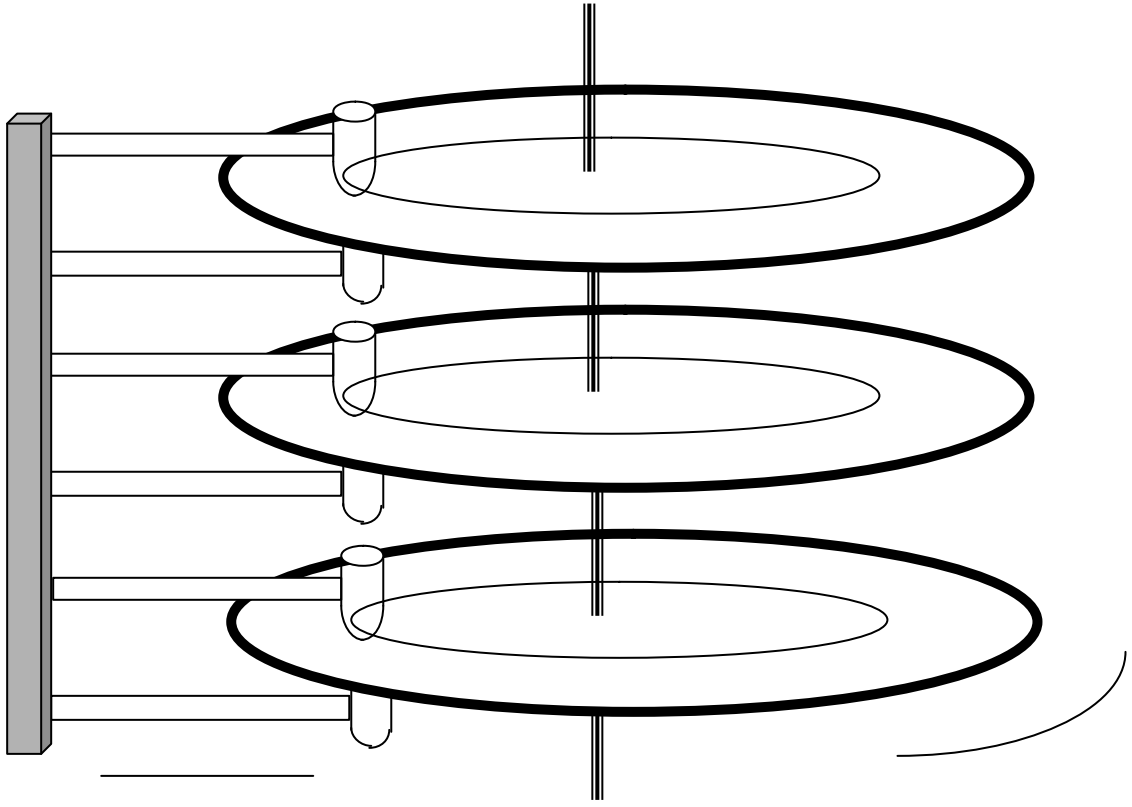
Lemezkezelés

- **Háttértárolók felépítése**
  - Szalag, Lemez, CD
- **Fizikai lemezkezelés**
  - Ütemezés, Címszámítás, Adatátvitel
- **Az adattárolás optimalizálása**
  - Blokkméret
  - Tömörítés, redundancia

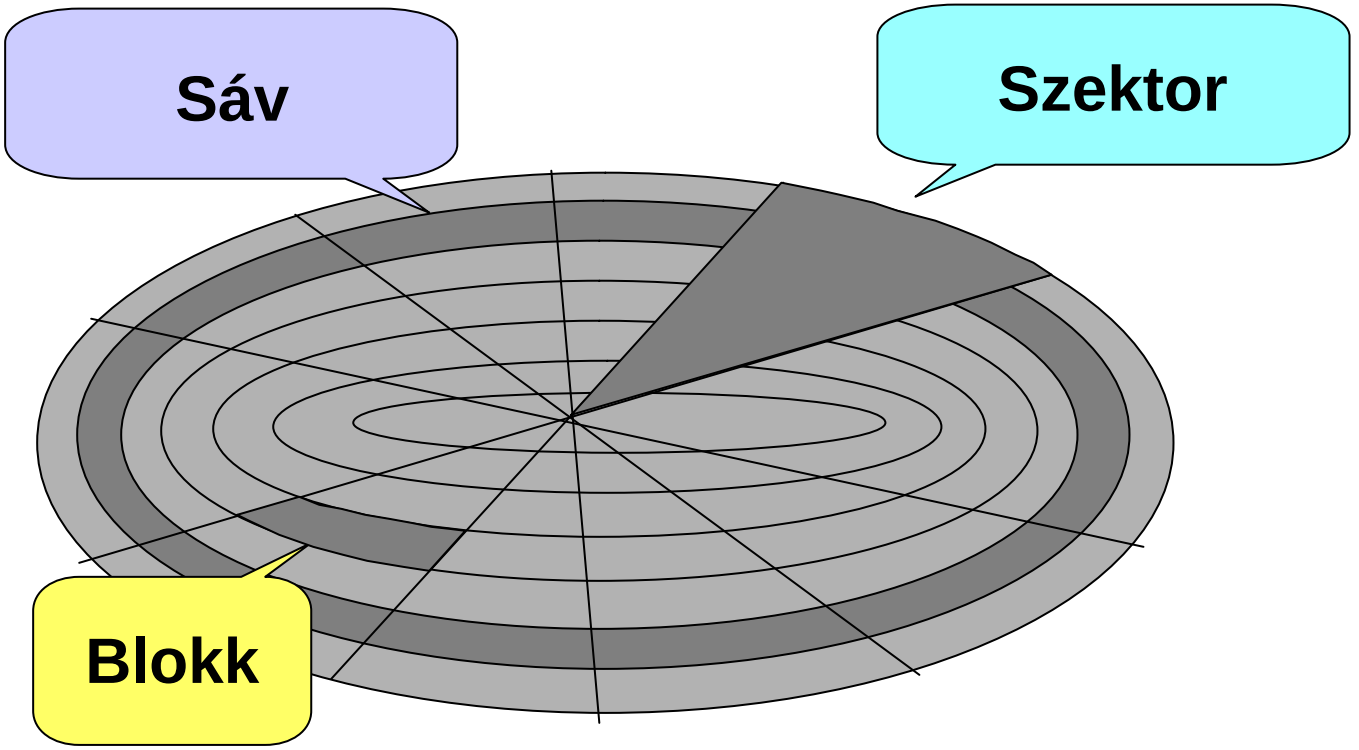
Háttértárak



Merevlemezek felépítése



Lemezfelületek felosztása



Mágneslemezek jellemzoi

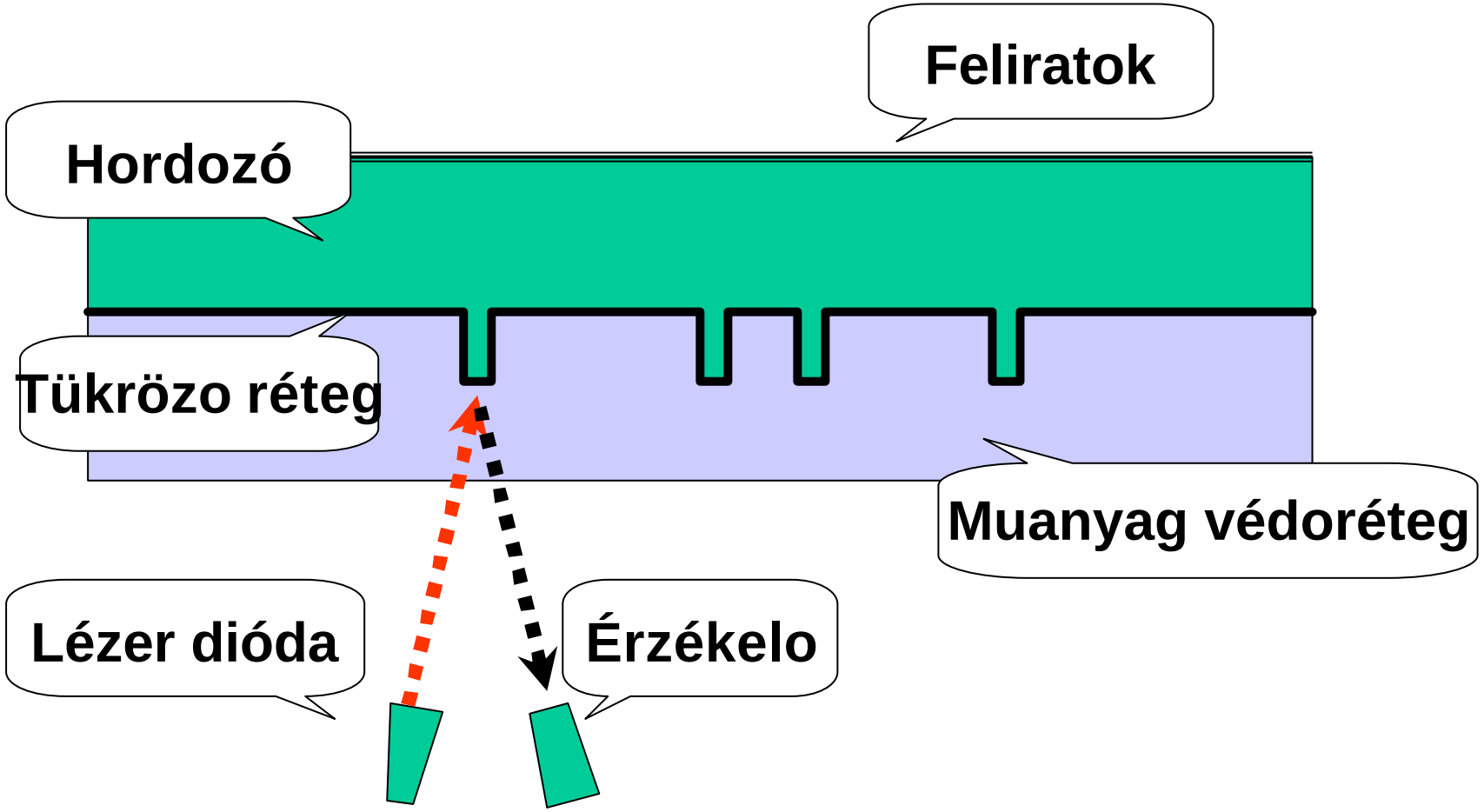
Technikai jellemzoi

- **Kapacitás: 1-180GB**
- **Elérési ido: 5-10 ms**
- **Adatátviteli sebesség: 2-20-(100-1000)Mb/s**

Alkalmazása

- **Virtuális memória**
- **Programok tárolása**
- **Adatok tárolása**

Optikai lemezek felépítése



Optikai lemezek jellemzoi

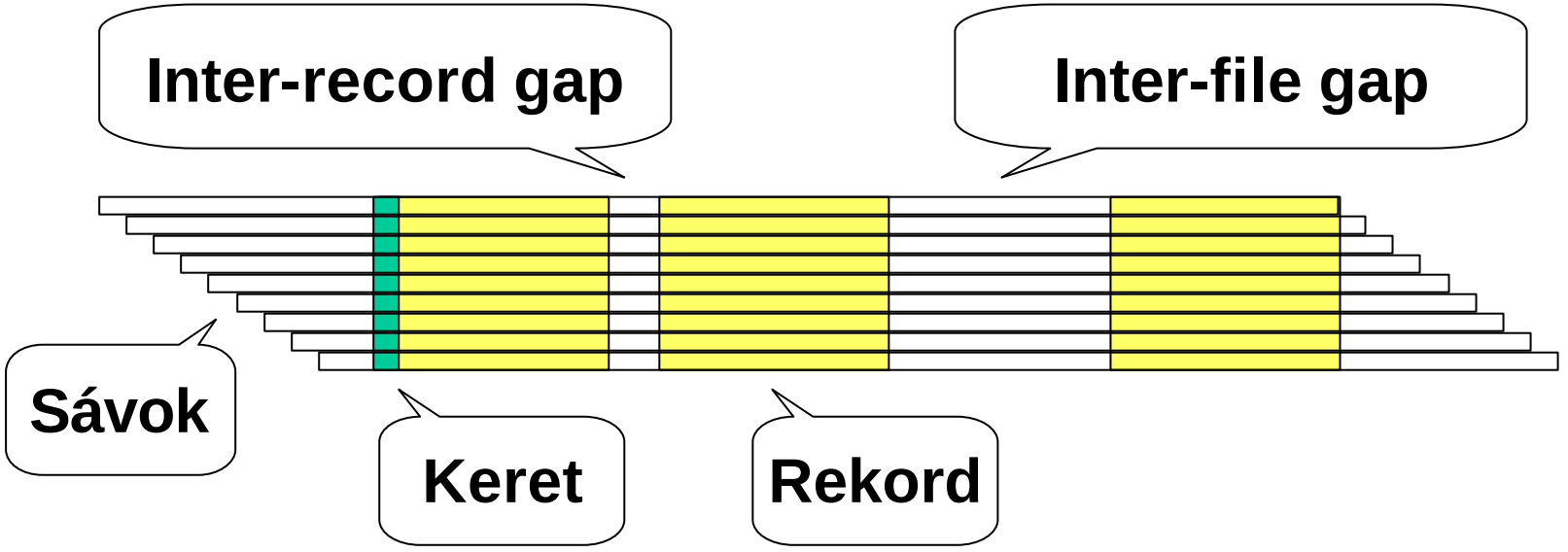
Technikai jellemzoi

- **Kapacitás: 650MB-14GB**
- **Elérési ido: 300 ms-tól**
- **Adatátviteli sebesség: 150 kb/s-tól**

Számítástechnikai alkalmazása

- **Archiválás**
- **Program kereskedelem**
- **Nagy adatbázisok, lexikonok**

Mágnesszalagok felépítése



Kapacitás: 2-16 GB
Keresési ido: 0-5 perc
Adatátviteli ido: 2-10 Mb/sec
Újra felhasználható !

Mágnesszalagok jellemzoi

Technikai jellemzoi

- **Kapacitás: 2-100 GB**
- **Keresési ido: 0-5 perc**
- **Adatátviteli ido: 2-10 Mb/sec**
- **Újra felhasználható !**

Alkalmazása

- **Nagy mennyiségu, összefüggo adat**
- **Archiválás, adatmentés**
- **Nagy tömegu adat átvitele**
- **Sok adat átmeneti tárolása**

Háttértárak összehasonlítása

	Kapacitás	Elérés	Átvitel	Ár/MB
HDD-IDE	2..72 GB	8 ms	100 MB/s	2000 Ft/GB
HDD-SCSI	9..72 GB	5 ms	80 MB/s	8000 Ft/GB
HDD-FC	30..72 GB	10 ms	400 MB/s	3000 Ft/GB
CD-ROM	650 MB	100 ms	6 MB/s	300 Ft/GB
CD-RW	650 MB	100 ms	6 MB/s	500 Ft/GB
DVD-ROM	8/16 GB	100 ns	16 Mbps	n.a.
DVD-RAM	2,5/5 GB	10 ms	2 MB/s	2000 Ft/GB
Streamer	4..20 GB	n.a.	1 MB/s	1000 Ft/GB
DAT	2 GB	n.a.	2 MB/s	200 Ft/GB
DLT	20/40 GB	n.a.	10 MB/s	1200 Ft/GB

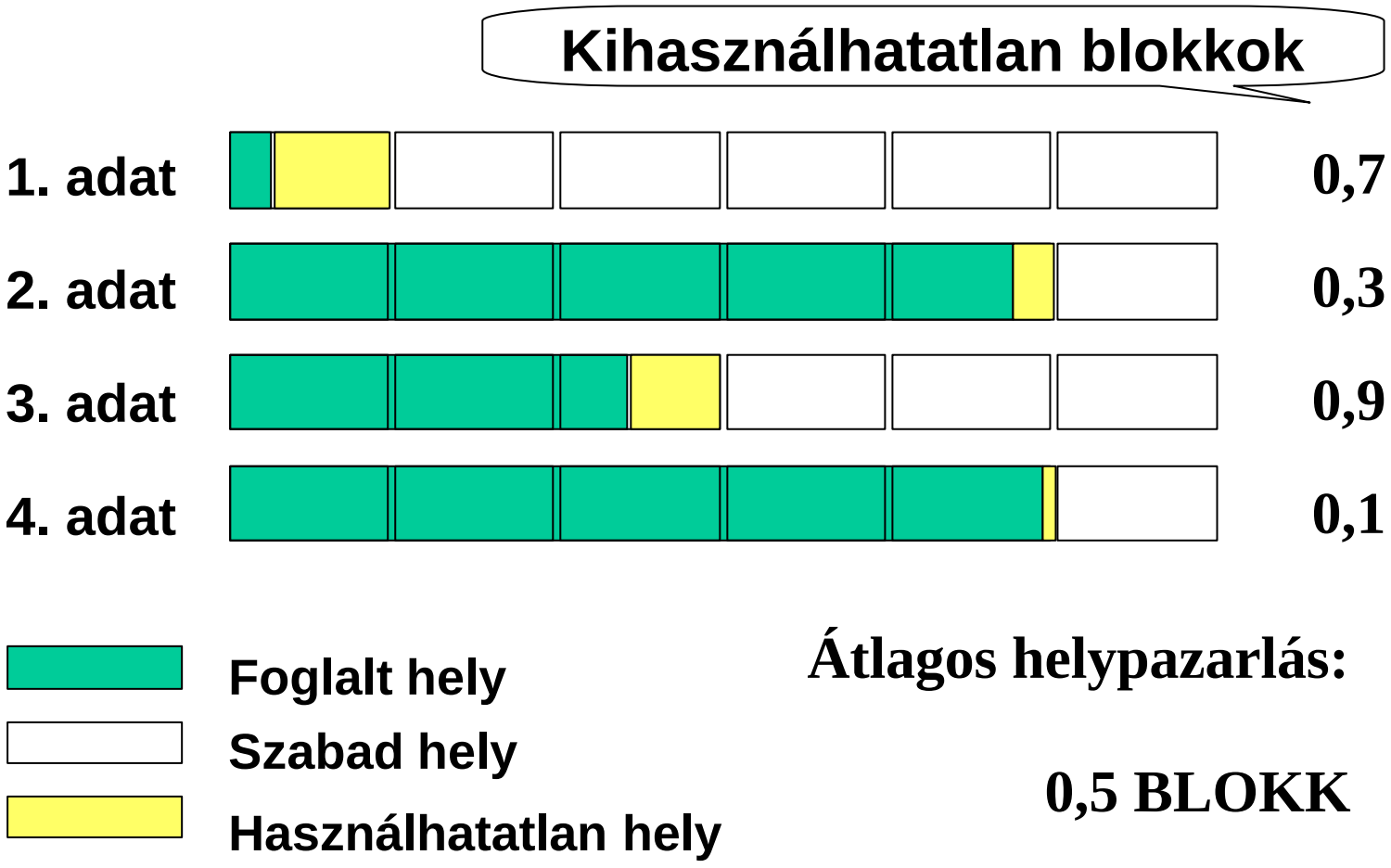
Mágneslemezek

Blokkméret

Tömörítés

Adatvédelem

Helykihasználás



Foglaltsági tábla

A lemez kapacitása 1 GB = 2^{30} bájt

Blokkméret	Foglaltsági tábla mérete	
512 = 2^9 bájt	$2^{30}/2^9/2^3 = 2^{18}$	256 kB
2048 = 2^{11} bájt	$2^{30}/2^{11}/2^3 = 2^{16}$	64 kB
64k = 2^{16} bájt	$2^{30}/2^{16}/2^3 = 2^{11}$	2 kB

Blokkok elhelyezkedése

1. lemez	**1**	**21**	**5**	**25**	**9**	**29**	**13**	**33**	**17**
2. lemez	**2**	**22**	**6**	**26**	**10**	**30**	**14**	**34**	**18**
3. lemez	**3**	**23**	**7**	**27**	**11**	**31**	**15**	**35**	**19**
4. lemez	**4**	**24**	**8**	**28**	**12**	**32**	**16**	**36**	**20**

4 lemez, 9 szektor, 1:2 interleave

Tömörítési eljárások

Kisebb helyfoglalás ⟷ **Nagyobb számításigény**
Gyorsabb adatátvitel **Kisebb adatbiztonság**

Futás hossz kódolás: Sok azonos karakter esetén

Különbségi kódolás: Lassan változó minta esetén

Huffmann-kódolás: Erosen eltéro gyakoriságú karakterek esetén

Gyakorlatban: **Microsoft Drive Space / Double Space**
Novell NetWare Compressed Volume

Huffmann-kódolás

Eredeti szöveg: **KEREKES SZEKEREK MENNEK**

Statisztika, kódolás:

8 db E	**00**
5 db K	**01**
2 db R	**10**
2 db S	**1100**
2 db N	**1101**
2 db space	**1110**
1 db M	**11110000**
1 db Z	**11110001**

Hatékonyság:

Eredeti szöveg:
184 bit

Kódolt szöveg
70 bit

Adatbiztonságot javító módszerek

Adatszintu védelem

- **paritásbit - egyetlen bithiba**
- **hibajavító kód - független hibák**
- **CRC - összefüggo hibák**

Eszközszintu védelem

- **Lemeztükrözés**
- **Lemez megkettozése**
- **RAID - adatok redundáns elosztása**

Az adatátvitelhez szükséges adatok

Eszköz típusa

Eszköz sorszáma

Adat kezdocíme az eszközön

Adat kezdocíme a memóriában

Adat mennyisége

Átvitel iránya: Írás vagy olvasás

Visszatérési folyamat

A lemez által várt adatok

Fej sorszáma

Szektor sorszáma

Cilinder sorszáma

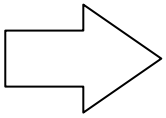
Adat kezdocíme a memóriában

Adat mennyisége

Átvitel iránya: Írás vagy olvasás

Címtranszformáció

Blokk logikai címe

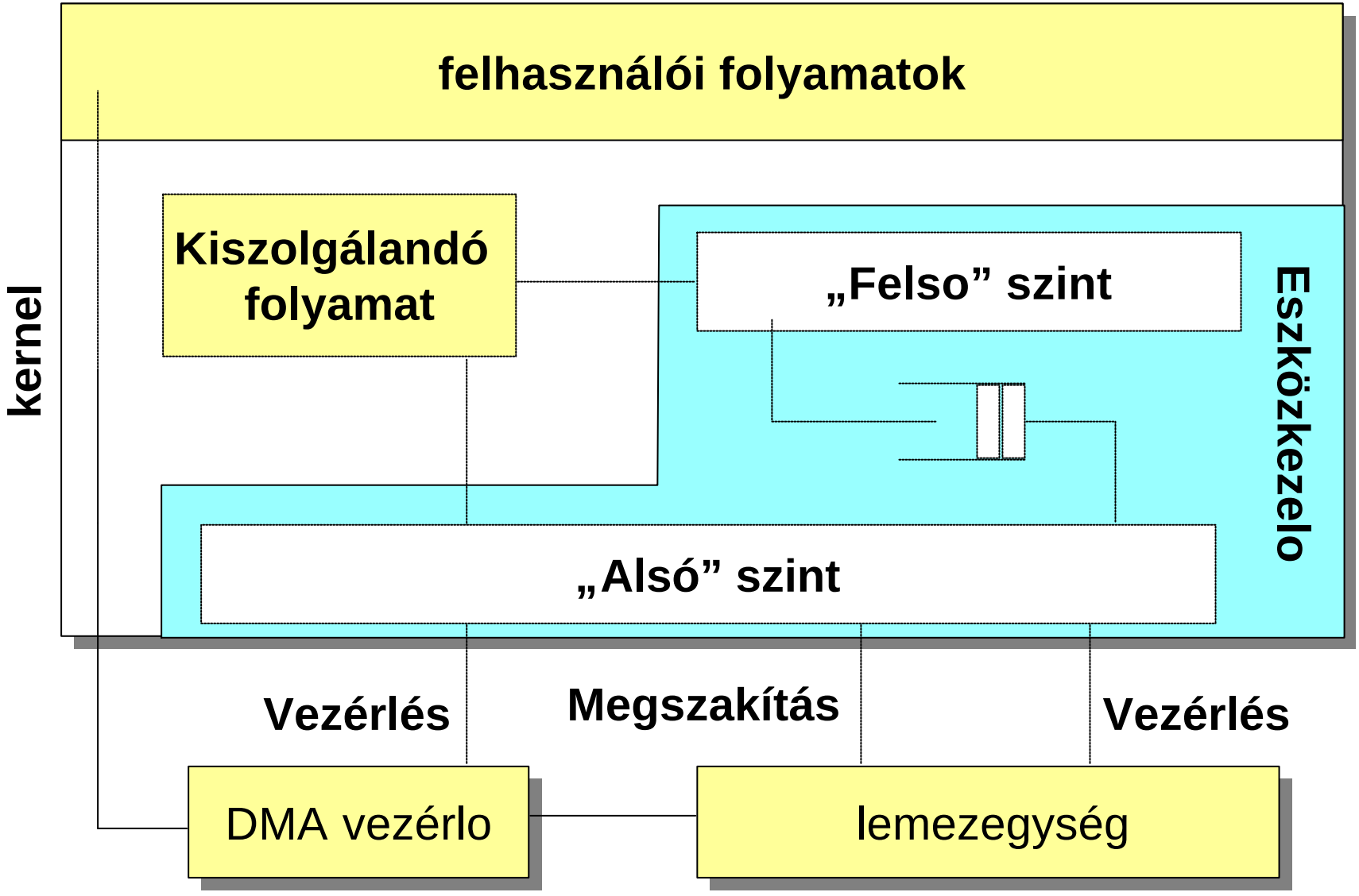


h - Fej sorszáma
s - Szektor sorszáma
c - Cilinder sorszáma

b = h * C* S + c * S + s

C - cilinderek száma
S - szektorok száma

Ennek az inverze kell !!!


felhasználói folyamatok
kernel
Kiszolgálandó folyamat
„Felso” szint
Eszközkezelo
„Alsó” szint
Vezérlés
Megszakítás
Vezérlés
DMA vezérlo
lemezegység

Lemezütemezési algoritmusok

Algoritmus		Várakozási ido	Várakozási ido szórása
Sorrendi	FCFS	nagy	kicsi
Legrövidebb ideju	SSTF	kicsi	nagy
Pásztázó	SCAN	közepes	közepes
Egyirányú pásztázó	C-SCAN	közepes	kicsi

Adatátviteli módok

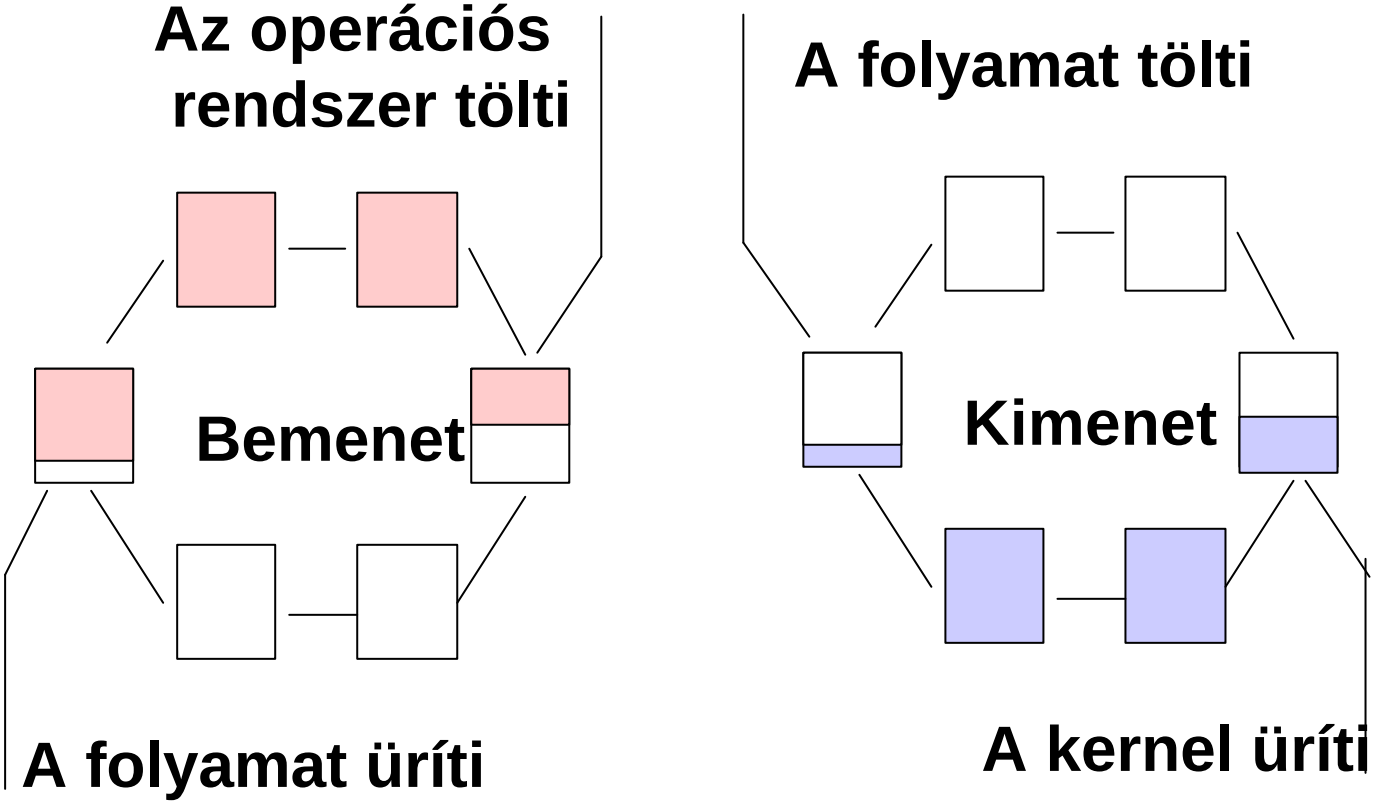
Szinkron: kiszolgálás közben a folyamat várakozik

Aszinkron: a folyamat fut tovább

Lemezgyorsító (Disk cache)

Körkörös átmeneti tár (Buffer pool)

Körkörös átmeneti tárolók



Összefoglalás

- **Fizikai lemezkezelés**
  - Ütemezés: FCFS, SSTF, Scan változatok
  - Logikai-fizikai cím konverzió
  - Blokkméret optimalizálás
- **Tömörítés**
  - RLE, DE, Huffmann
- **Adatvédelem**
  - Szoftver: paritás, CRC
  - Hardver: tükrözés, RAID

Eroforráskezelés

Eroforrások

- Eroforrás foglalási gráf
- Holtpont, Kiéheztetés

Holtpont kezelo stratégiák

- Megelozés
- Felszámolás
- Közösen használt eroforrások problémái

Eroforráskezelés alapfogalmai

- **Feladat**
  - **a számítógéprendszer eroforrásainak elosztása (a futó folyamatok igényei alapján)**
  - **illetve az eroforrások használatáért vívott versenyhelyzetek kezeléséról**
- **Céljai:**
  - **a rendszer muködését gazdaságossá tenni**
  - **elkerülni a holtponthelyzetek kialakulását és/vagy felszámolni a kialakult holtponthelyzeteket**

Eroforrások csoportosítása

hardver eroforrások (központi processzor, memóriák, I/O csatornák, perifériák)

szoftver eroforrások (programok, adatállományok - egyre fontosabbak)

“hagyományos” eroforrások (nyomtatók, szövegszerkesztok)

az op. r. által létrehozott eroforrások (lapok, pufferek, floppy blokkok, adatállományok, tartalomjegyzékek)

elveheto eroforrások

nem elveheto eroforrások

Gazdaságosság

Valamilyen költségfüggvény minimalizálása

- kihasználtság
- válaszido

Egyéb szempontok (ellentmondóak)

- átlagos átfutási ido minimalizálása
- a lassú válaszok számának minimalizálása
- a (hardver) kihasználtság maximalizálása
- maximális kihasználtság adott válaszidokorlát mellett

Tendencia: a kihasználtság háttérbe szorul (HW ára csökken!) a válaszidovel szemben

Az eroforrások lefoglalása

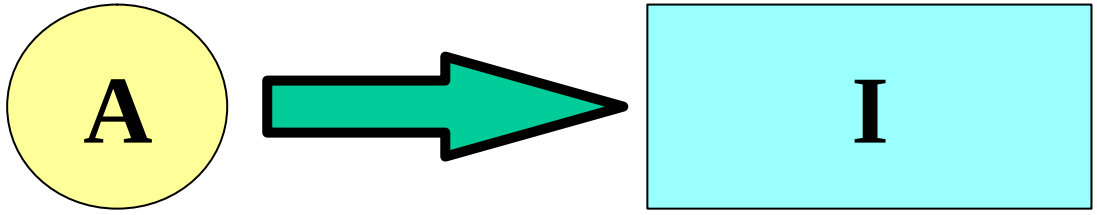
Statikus lefoglalás

– a folyamat indulása elott lefoglalja az összes szükséges eroforrást
  - **pazarló**
  - **kiéheztetés**
  - **ha egyszer elindult, eroforrás korlát miatt nem áll le**

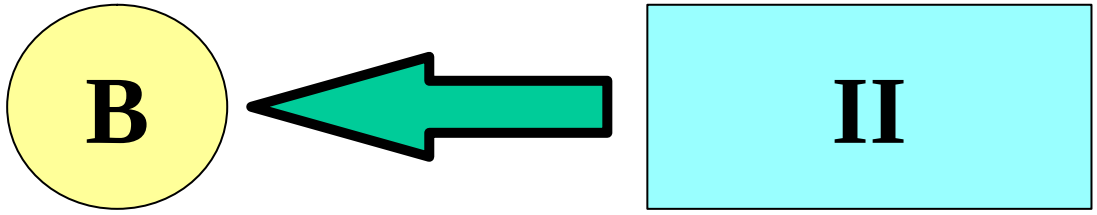
Dinamikus lefoglalás

– a folyamat csak akkor igényel eroforrást, amikor éppen szüksége van rá
  - **hamar elindulhat egy folyamat, de versenyzés miatt lassabban fut(hat)**
  - **teljesítoképesség, átbocsátóképesség no**
  - **nagy probléma: holtpont**

Állapot átmeneti gráf

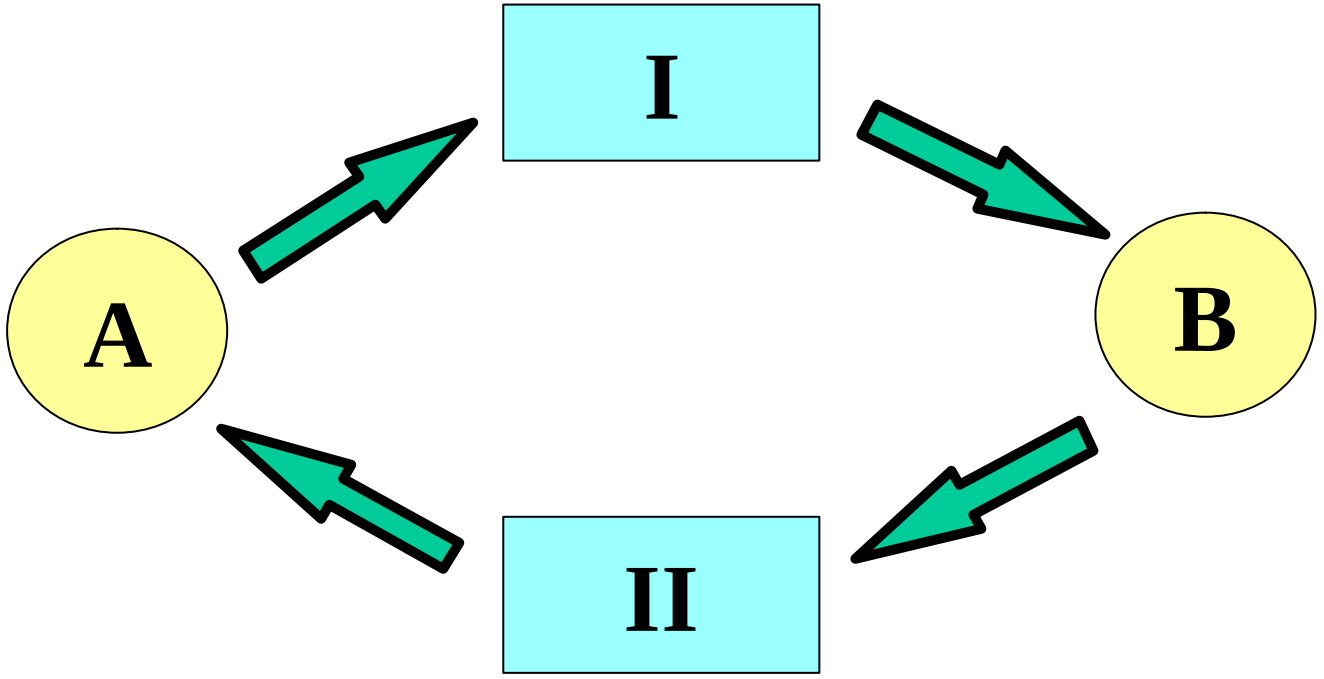


Az „A” folyamat igényli a „I” eroforrást



A „B” folyamat birtokolja a „II” eroforrást

Holtponti állapotgráf



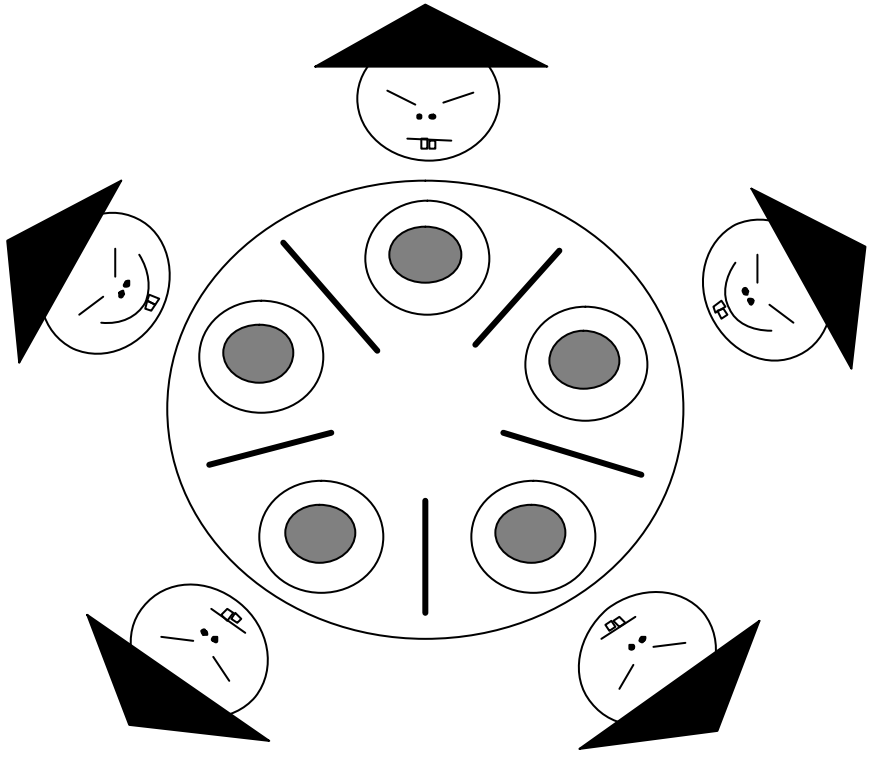
HOLTPONT - Deadlock

**Több folyamat egy olyan eroforrás felszabadulására vár,
amit csak egy ugyancsak várakozó folyamat tudna eloidézni**

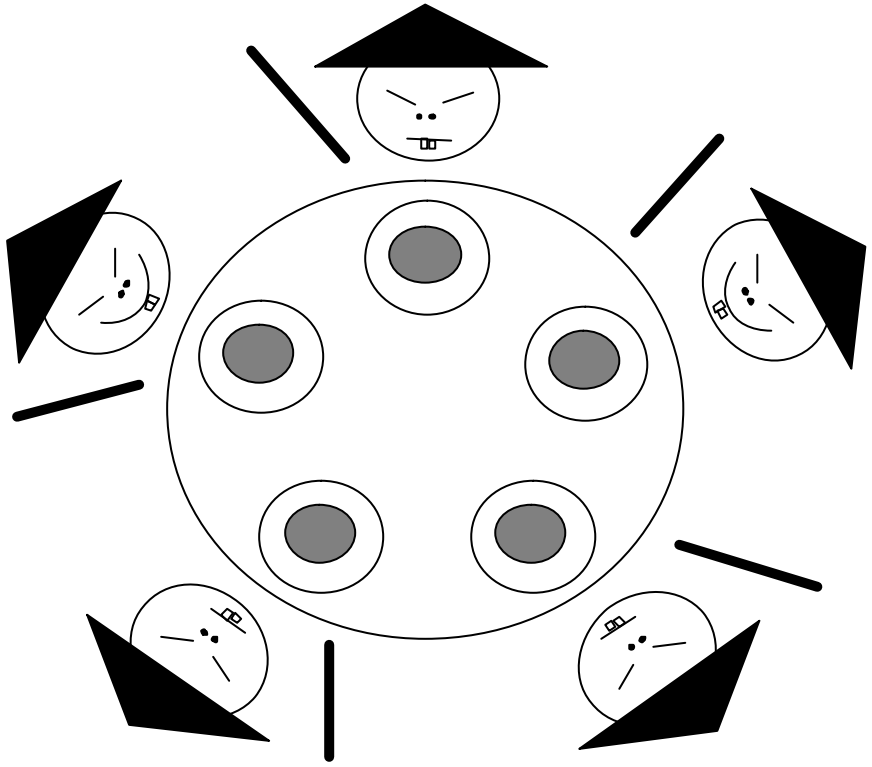
KIÉHEZTETÉS - Starvation

Egy folyamat - az eroforráskezelo stratégiája miatt - beláthatatlan ideig nem jut eroforráshoz

Vacsorázó bölcsek - példa

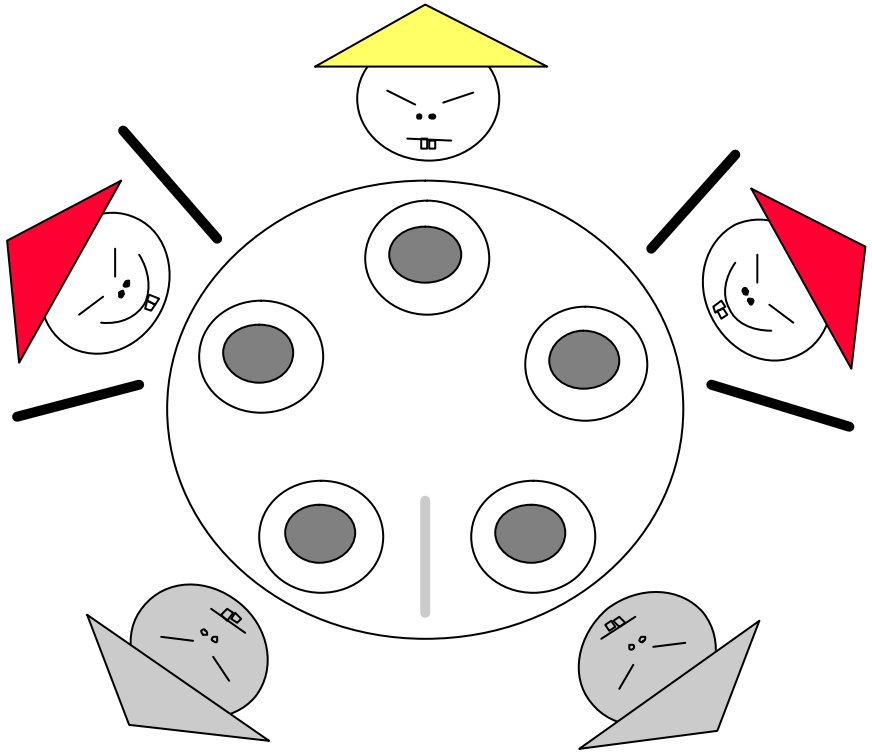


Vacsorázó bölcsek - Holtpont



Egyikük sem rendelkezik az ÖSSZES szükséges eroforrással

Vacsorázó bölcsek - Kiéheztetés



Összesen ugyan van elegendo eroforrás, de szerencsétlen esetben egyesek mégis éheznek

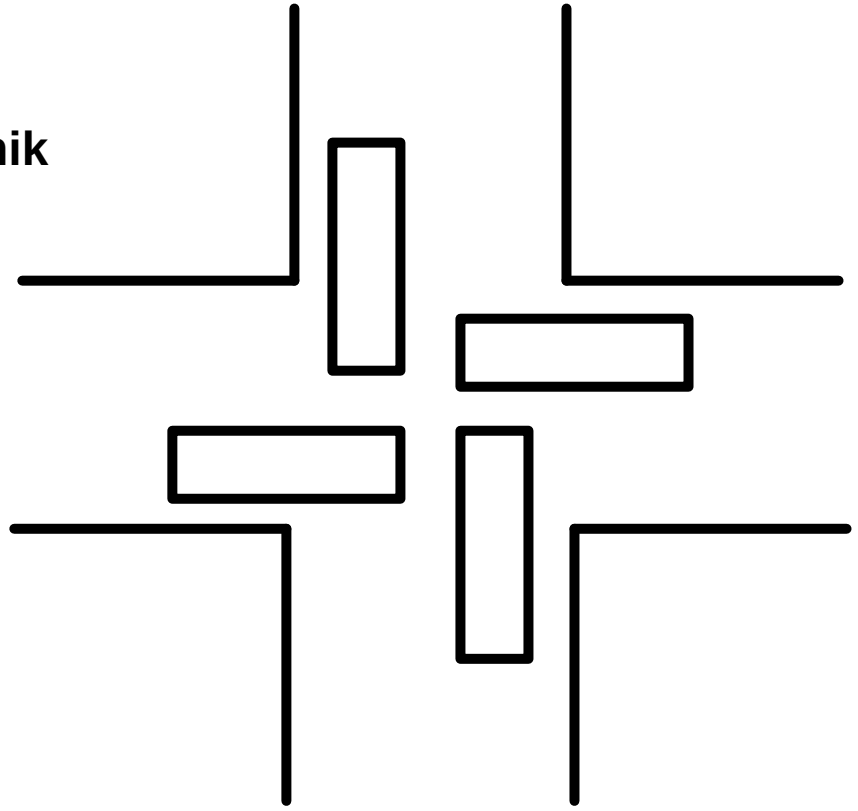
Holtpontkialkulás lehetoségének feltételei

1. **Kölcsönös kizárás van**
2. **Várakozás közben lekötés történik**
3. **Rablás nincs**
4. **Ciklikus várakozás van**

A feltételeknek EGYSZERRE kell teljesülniük a holtpont kialakulásához, vagyis HA LEGALÁBB AZ EGYIK FELTÉTEL NEM TELJESÜL, NEM ALAKULHAT KI HOLTPONT!

Holtpontkialkulás lehetoségének feltételei - illusztráció

1. **Kölcsönös kizárás van**
2. **Várakozás közben lekötés történik**
3. **Rablás nincs**
4. **Ciklikus várakozás van**



Egy „igazi” holtpont

Holtpont kezelési stratégiák

- **A holtpont megelozése**
  - A négy feltétel (legalább) egyikét nem engedjük teljesülni
  - **NEM MINDIG LEHETSÉGES**
    (a korlátozó feltételek miatt)
- **A kialakult holtpont felismerése és megszüntetése**

A megelozés “olcsóbb”

A holtpont megelozése

Kölcsönös kizárás

Ezzel a feltétellel nem tudunk mit kezdeni, vannak olyan eroforrások, melyeket egyszerre csak egy folyamat használhat (pl. nyomtató)

A holtpont megelozése

Várakozás közbeni lekötés

Egy folyamat csak akkor igényelhet újabb eroforrást, ha nincs más lekötött eroforrása

Megoldások:

- Statikus eroforrás-kezelés
- Ha egy folyamat eroforrást igényel, el kell engedje az összes korábban lefoglalt eroforrását

Hátrányok:

- rossz az eroforrások kihasználtsága
- kiéheztetés

A holtpont megelozése

Eroforrásrablás

1. **Ha egy folyamat egy olyan eroforrást igényel, amit nem tud megkapni azonnal (vagyis várakoznia kell), akkor elvesszük tole az ÖSSZES lefoglalva tartott eroforrását, akkor folytatódhat, ha visszaszerezte az összes régit és megszerezte az újat is**
2. **Ha egy folyamat olyan eroforrást igényel, amit más VÁRAKOZÓ folyamat foglal, ezt attól elveszi; ha az igényelt eroforrást egy nem várakozó folyamat használja, akkor az IGÉNYLO folyamat kerül várakozó állapotba és TOLE rabolhatnak a többiek**

E módszerek CSAK az elveheto eroforrásokra alkalmazhatók

A holtpont megelozése

Ciklikus várakozás 1.

1. módszer

Minden eroforráshoz egy (a többitol különbözo) sorszámot rendelünk

A folyamatok az eroforrásokat csak azok sorszámaik szerint növekvo sorrendjében igényelhetik (ezért célszeru, ha a sorszámokat az eroforrások természetes felhasználási sorrendje szerint osztjuk ki)

Másképp fogalmazva: ha egy folyamat egy bizonyos sorszámú eroforrást igényel, fel kell szabadítania az összes általa lefoglalt, az igényeltnél NAGYOBB sorszámú eroforrást

Hátrány: csökken a rendszer áteresztoképessége

A holtpont megelozése

Ciklikus várakozás 2.

2. módszer

Az operációs rendszer csak akkor engedi meg egy új eroforrás lefoglalását, ha ennek teljesítése után a rendszer ún. BIZTONSÁGOS állapotban marad

Biztonságos állapot: az összes folyamat eroforrás igénye valamilyen sorrendben kielégítheto

Nem biztonságos állapot:

holtpont kialakulhat (nem biztos, hogy kialakul!)

Hátrányok:

Olyan pluszinformáció szükséges (a maximális igény), amely sokszor nem tudható elore

Bonyolult algoritmus

Holtpont megelozo stratégiák

Egyetlen foglalási lehetoség

- **Csak az a folyamat foglalhat eroforrást, amelyik egyetlen egy fölött sem rendelkezik**

- **Elony:**
  - **nincs holtpont**

Hátrány:

- Rossz eroforrás kihasználás,
- kiéheztetés veszélye

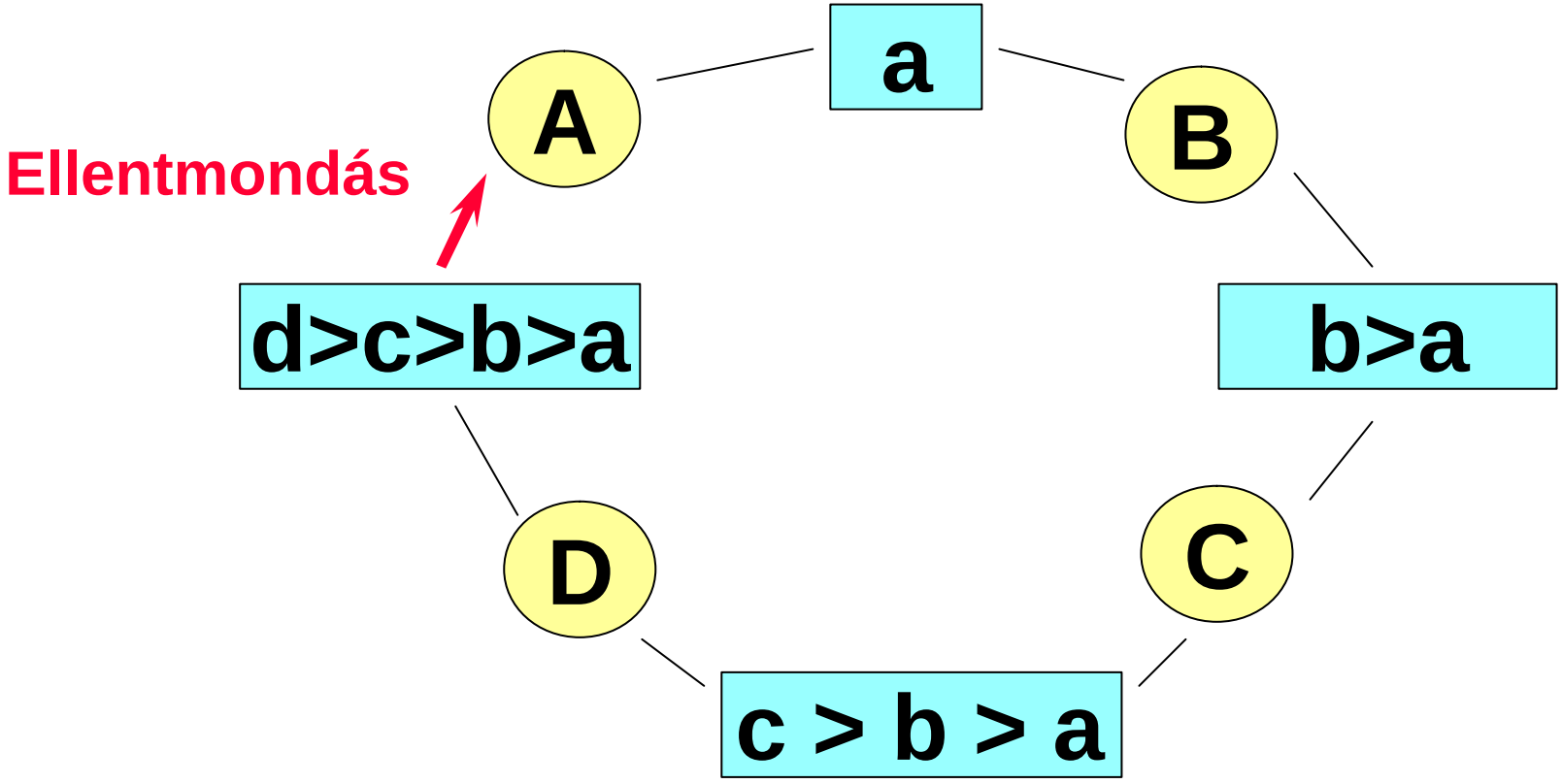
Holtpont megelozo stratégiák

Rangsor szerinti foglalás

- **Egy folyamat csak olyan osztályból igényelhet eroforrást, melynek sorszáma magasabb, mint a már birtokolt eroforrások sorszáma**

• **Elony:** – **Nincs holtpont**	**Hátrány:** – **Pazarló** – **Önkényes**

Rangsor szerinti foglalás bizonyítás



Holtpont megelozo stratégiák

Bankár algoritmus

- **Sohase elégítsünk ki egy igényt, ha az bizonytalan állapotot eredményez**

Elony:

- **nincs holtpont**
- **az elozoeknél hatékonyabb**

Hátrány:

- **Sok számolást igényel**
- **Elozetes feltevéseken alapul**

Bankár algoritmus - példa A.1

Aktuális állapot:

Összesen 12 db eroforrás

A és B folyamatok

	Foglal	Max. igény	Várható igény
A	**4**	**6**	**2**
B	**4**	**11**	**7**
Szabad	**4**		

Várakozó „C” folyamat - BEENGEDHETO ?

C	**2**	**8**	**6**

Bankár algoritmus - példa A.2.

Tegyük föl, hogy beengedjük...

	Foglal	Max. igény	Várható igény
A	4	6	2
B	4	11	7
C	2	8	6
Szabad	2		

Az „A” folyamat lefuthat, mivel várható igénye nem nagyobb a szabad eroforrások számánál

Bankár algoritmus - példa A.3.

Az „A” lefutott, foglalt eroforrásai felszabadultak…

	Foglal	Max. igény	Várható igény
B	**4**	**11**	**7**
C	**2**	**8**	**6**
Szabad	**6**		

Az „C” folyamat lefuthat, mivel várható igénye nem nagyobb a szabad eroforrások számánál

Bankár algoritmus - példa A.4.

Az „C” lefutott, foglalt eroforrásai felszabadultak...

	Foglal	Max. igény	Várható igény
B	**4**	**11**	**7**
Szabad	**8**		

Az „B” folyamat lefuthat, mivel várható igénye nem nagyobb a szabad eroforrások számánál

Az állapot a „C” beengedése után is biztonságos

ENGEDJÜK BE !

Bankár algoritmus - példa B.1

Aktuális állapot:

Összesen 12 db eroforrás

A és B folyamatok

	Foglal	Max. igény	Várható igény
A	**4**	**6**	**2**
B	**4**	**11**	**7**
Szabad	**4**		

Várakozó „C” folyamat - BEENGEDHETO ?

C	**2**	**9**	**7**

Bankár algoritmus - példa B.2.

Tegyük föl, hogy beengedjük...

	Foglal	Max. igény	Várható igény
A	4	6	2
B	4	11	7
C	2	9	7
Szabad	2		

Az „A” folyamat lefuthat, mivel várható igénye nem nagyobb a szabad eroforrások számánál

Bankár algoritmus - példa B.3.

Az „A” lefutott, foglalt eroforrásai felszabadultak…

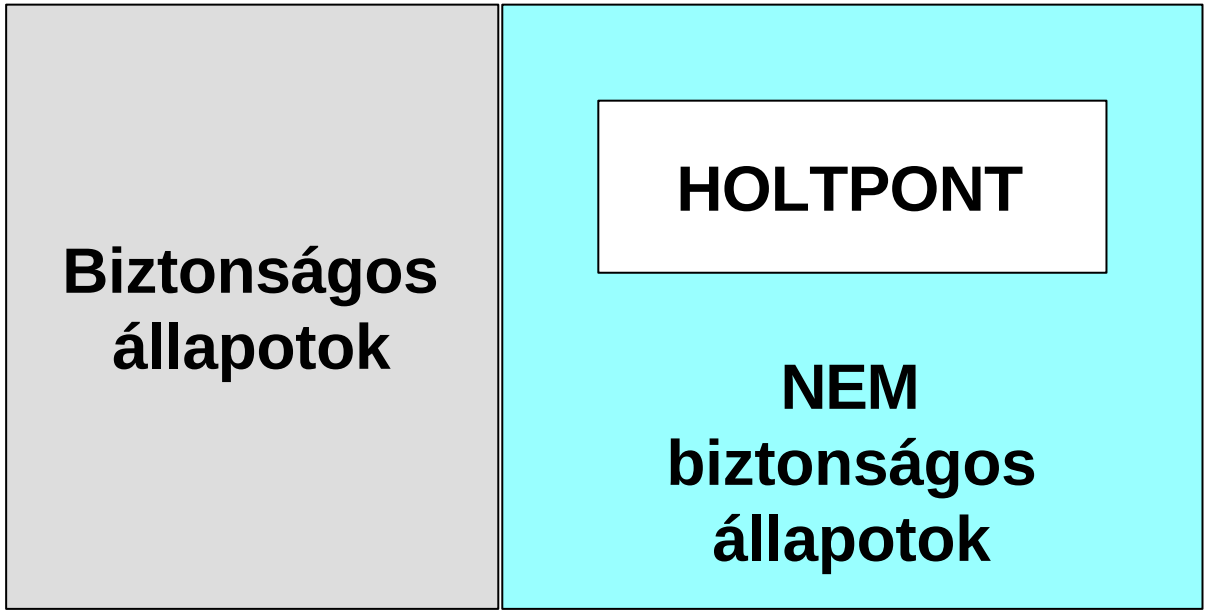
	Foglal	Max. igény	Várható igény
B	4	11	7
C	2	8	7
Szabad	6		

Az állapot „C” beengedése után

NEM BIZTONSÁGOS

Biztonságos - nem biztonságos

Egy rendszer biztonságos, ha létezik olyan sorrend, mely szerint haladva a folyamatok igénye kielégítheto



Bankár algoritmus

1. **Felírjuk a folyamatok által maximálisan igényelt eroforrások számait tartalmazó mátrixot (MAX. IGÉNY)**
2. **Felírjuk a folyamatok által lefoglalva tartott eroforrások számait tartalmazó mátrixot (FOGLAL)**
3. **A MAX. IGÉNYbol FOGLALt kivonva kapjuk a még kielégítetlen igényeket leíró mátrixot (IGÉNY)**
4. **Eroforrás fajtánként összeadjuk a lefoglalva tartott eroforrások számait, majd ezeket kivonva az egyes eroforrások összdarabszámából kapjuk a pillanatnyilag rendelkezésre álló eroforrás készletet (KÉSZLET)**

Bankár algoritmus - folytatás

5. Megnézzük, hogy a KÉSZLETbol kielégítheto-e valamelyik folyamat igénye

6a. Ha nincs ilyen folyamat, a rendszer NEM BIZTONSÁGOS állapotban van

6b. Ha van ilyen folyamat, kielégítjük igényét

7. A kiválasztott folyamat a lefutása után felszabadítja az összes általa használt eroforrást, azaz KÉSZLET új értékét megkapjuk, ha elozo értékéhez hozzáadjuk a folyamat által eredetileg lefoglalva tartott eroforrások számát (FOGLAL megfelelo sorát)

8. Visszamegyünk az 5. lépésre

Bankár algoritmus - Több eroforrás

A feladat megfogalmazása

Egy rendszerben az alábbi eroforrások vannak:

E1: 10 darab E2: 5 darab E3: 7 darab

A rendszerben 5 folyamat van: F1, F2, F3, F4, F5

Biztonságos-e holtpontmentesség szempontjából a következo állapot?

	MAX. IGÉNY			FOGLAL		
	E1	E2	E3	E1	E2	E3
F1	7	5	3	0	1	0
F2	3	2	2	3	0	2
F3	9	0	2	3	0	2
F4	2	2	2	2	1	1
F5	4	3	3	0	0	2

Bankár algoritmus - Több eroforrás

IGÉNY meghatározása

[IGÉNY] = [MAX.IGÉNY] - [FOGLAL]

	MAX. IGÉNY		
	E1	E2	E3
F1	7	5	3
F2	3	2	2
F3	9	0	2
F4	2	2	2
F5	4	3	3

	FOGLAL		
	E1	E2	E3
F1	0	1	0
F2	3	0	2
F3	3	0	2
F4	2	1	1
F5	0	0	2

	IGÉNY		
	E1	E2	E3
F1	7	4	3
F2	0	2	0
F3	6	0	0
F4	0	1	1
F5	4	3	1

Bankár algoritmus - Több eroforrás

KÉSZLET meghatározása

(minden eroforrásra)

? EROFORRÁS

- ? FOGLAL

KÉSZLET

	FOGLAL E1	E2	E3
F1	0	1	0
F2	3	0	2
F3	3	0	2
F4	2	1	1
F5	0	0	2
? FOGLAL	8	2	7

E1: 10 darab
E2: 5 darab
E3: 7 darab

	E1	E2	E3
? EROFORRÁS	10	5	7
? FOGLAL	8	2	7
KÉSZLET	2	3	0

Bankár algoritmus - Több eroforrás

IGÉNY kielégítése - 1. lépés

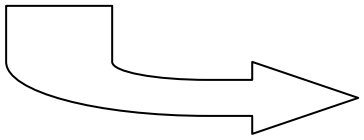
	IGÉNY		
	E1	E2	E3
F1	7	4	3
F2	0	2	0
F3	6	0	0
F4	0	1	1
F5	4	3	1

Kielégítheto

Felszabadul

	FOGLAL		
	E1	E2	E3
F1	0	1	0
F2	3	0	2
F3	3	0	2
F4	2	1	1
F5	0	0	2

KÉSZLET : (2, 3, 0)



ÚJ KÉSZLET : (5, 3, 2)

Bankár algoritmus - Több eroforrás

IGÉNY kielégítése - 2. lépés

	IGÉNY		
	E1	E2	E3
F1	7	4	3
F2	0	2	0
F3	6	0	0
F4	0	1	1
F5	4	3	1

	FOGLAL		
	E1	E2	E3
F1	0	1	0
F2	3	0	2
F3	3	0	2
F4	2	1	1
F5	0	0	2

Kielégítheto

Felszabadul

KÉSZLET : (5, 3, 2)

ÚJ KÉSZLET : (5, 3, 4)

Bankár algoritmus - Több eroforrás

IGÉNY kielégítése - 3. lépés

	IGÉNY		
	E1	E2	E3
F1	7	4	3
F2	0	2	0
F3	6	0	0
F4	0	1	1
F5	4	3	1

	FOGLAL		
	E1	E2	E3
F1	0	1	0
F2	3	0	2
F3	3	0	2
F4	2	1	1
F5	0	0	2

Kielégítheto

Felszabadul

KÉSZLET : (5, 3, 4)

ÚJ KÉSZLET : (7, 4, 5)

Bankár algoritmus - Több eroforrás

IGÉNY kielégítése - 4. lépés

	IGÉNY		
	E1	E2	E3
F1	7	4	3
F2	0	2	0
F3	6	0	0
F4	0	1	1
F5	4	3	1

Kielégítheto

Felszabadul

	FOGLAL		
	E1	E2	E3
F1	0	1	0
F2	3	0	2
F3	3	0	2
F4	2	1	1
F5	0	0	2

KÉSZLET : (7, 4, 5)

ÚJ KÉSZLET : (7, 5, 5)

Bankár algoritmus - Több eroforrás

IGÉNY kielégítése - 5. lépés

IGÉNY	E1	E2	E3
F1	7	4	3
F2	0	2	0
F3	**6**	**0**	**0**
F4	0	1	1
F5	4	3	1

FOGLAL	E1	E2	E3
F1	0	1	0
F2	3	0	2
F3	**3**	**0**	**2**
F4	2	1	1
F5	0	0	2

Kielégítheto

Felszabadul

KÉSZLET : (7, 5, 3)

ÚJ KÉSZLET : (10, 5, 7)

Bankár algoritmus - Több eroforrás

BIZTONSÁGOS !

Vagyis találtunk legalább egy (ebben a példában több is van) sorrendet, amelyben a folyamatok eroforrás igénye kielégítheto:

F2, F5, F4, F1, F3

Tehát a rendszer

BIZTONSÁGOS ÁLLAPOTBAN van.

A holtpont felismerése

Folyamatos adatgyujtés az eroforrások szétosztásáról és a ki nem elégített igényekrol

Ezen adatokból a holtponthelyzetet detektálni képes algoritmus ismételt futtatása

- ha egy igényt nem lehet azonnal kielégíteni
  - gyakran kell futtatni, sok plusz ido
- elore megbecsült idonként - ezalatt holtpont (akár több is) kialakulhat
  - nagy felélesztési veszteség

Holtpontból való felélesztés

Megszüntetjük a holtponti helyzetben lévo folyamatokat - nagy veszteség

Finomítás: EGYENKÉNT szüntetünk meg holtponti helyzetben lévo folyamatokat, amíg a holtpont fennáll - kisebb veszteség

Holtpontból való felélesztés

Áldozatkijelölési szempontok

Melyikkel hány eroforrást nyerek

Hány további eroforrást igényel még

Mennyi már elhasznált CPU idot ill. I/O munkát vesztek

Mennyi van még hátra a futásából

Ismételheto / nem ismételheto folyamat-e

A folyamat prioritása

Megszüntetése hány további folyamatot érint

Holtpontból való felélesztés

Eroforrás rablás

- **A holtponti helyzetben lévo folyamatoktól eroforrásokat rablok el**
- **Veszteség, ha nem elveheto eroforrást kell elvenni**
- **Áldozatkijelölés az elozohöz hasonló szempontok szerint**

Holtpontból való felélesztés

Sokszor elég, ha a folyamatot nem szüntetjük meg teljesen, hanem egy elozo állapotából folytatjuk - ellenorzo pontok (check point)

- nagy tárigény
- sok plusz idot igényel a folyamatok állapotának periodikus mentése

Ugyanaz a folyamat nem választható akárhányszor áldozatul (kiéheztetés)

Az eroforrás-kezelés problémái

ALULSZABÁLYOZÁS - nem tudjuk kihasználni a HW lehetoségeit (nem muködtet párhuzamosan eszközöket, feleslegesen várakoztat stb.)

TÚLSZABÁLYOZÁS - túl sok adminisztrációt végez és ezzel csökkenti a hasznos idot és tárméretet

Közösen használt eroforrások

Termelo / fogyasztó probléma

A közös adatterületet (KÖZÖS EROFORRÁS) egyszerre csak egy folyamat használhatja (KÖLCSÖNÖS KIZÁRÁS)

Közösen használt eroforrások

Vezérlés: Szemafor

Kölcsönös kizárás igénye nemcsak közös memória esetén lép fel; pl. nyomtató közös használata

Mielott a folyamat használni kezdené a közös eroforrást, ellenoriznie kell, hogy az szabad-e. (Ezt az adott közös eroforráshoz rendelt szemafor jelzi.) CSAK akkor kezdheti el használni, ha a szemafor szabadot jelzett, ellenkezo esetben várakoznia kell!

Közösen használt eroforrások

Termelo és fogyasztó programja

1. a szemafor olvasása
2. a beolvasott érték vizsgálata
3. ha szabad: a szemafor foglaltra állítása
4. ha foglalt: vissza 1-re
5. az eroforrás használata (írás a közös memóriába)
6. a szemafor szabadra állítása

1. a szemafor olvasása
2. a beolvasott érték vizsgálata
3. ha szabad: a szemafor foglaltra állítása
4. ha foglalt: vissza 1-re
5. az eroforrás használata (olvasás a közös memóriából)
6. a szemafor szabadra állítása

Közösen használt eroforrások

A program közös elemei

1. a szemafor olvasása
2. a beolvasott érték vizsgálata
3. ha szabad: a szemafor foglaltra állítása
4. ha foglalt: vissza 1-re
5. az eroforrás használata (írás a közös memóriába)
6. a szemafor szabadra állítása

1. a szemafor olvasása
2. a beolvasott érték vizsgálata
3. ha szabad: a szemafor foglaltra állítása
4. ha foglalt: vissza 1-re
5. az eroforrás használata (olvasás a közös memóriából)
6. a szemafor szabadra állítása

P és V primitívek

Primitív: megszakíthatatlan (oszthatatlan) muvelet

P primitív: FOGLALTTÁ ÁLLÍTÁS

- **1. a szemafor olvasása**
- **2. a beolvasott érték vizsgálata**
- **3. ha szabad: a szemafor foglaltra állítása**
- **4. ha foglalt: vissza 1-re**

V primitív: SZABADDÁ ÁLLÍTÁS

- **A szemafor szabadra állítása**

Közösen használt eroforrások

Termelo és fogyasztó programja -Primitívekkel

P(S);

ÍRÁS A MEMÓRIÁBA

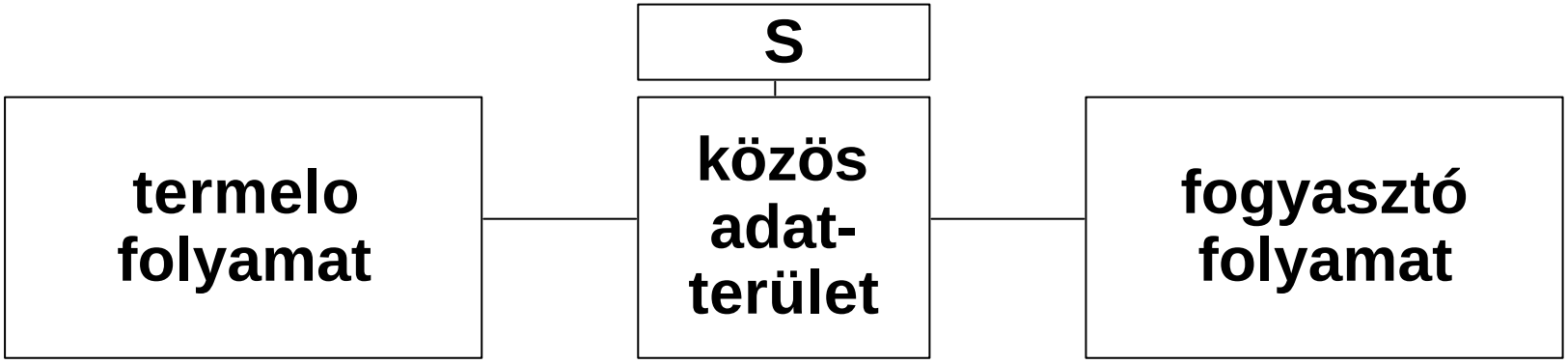
V(S);

P(S);

OLV. A MEMÓRIÁBÓL

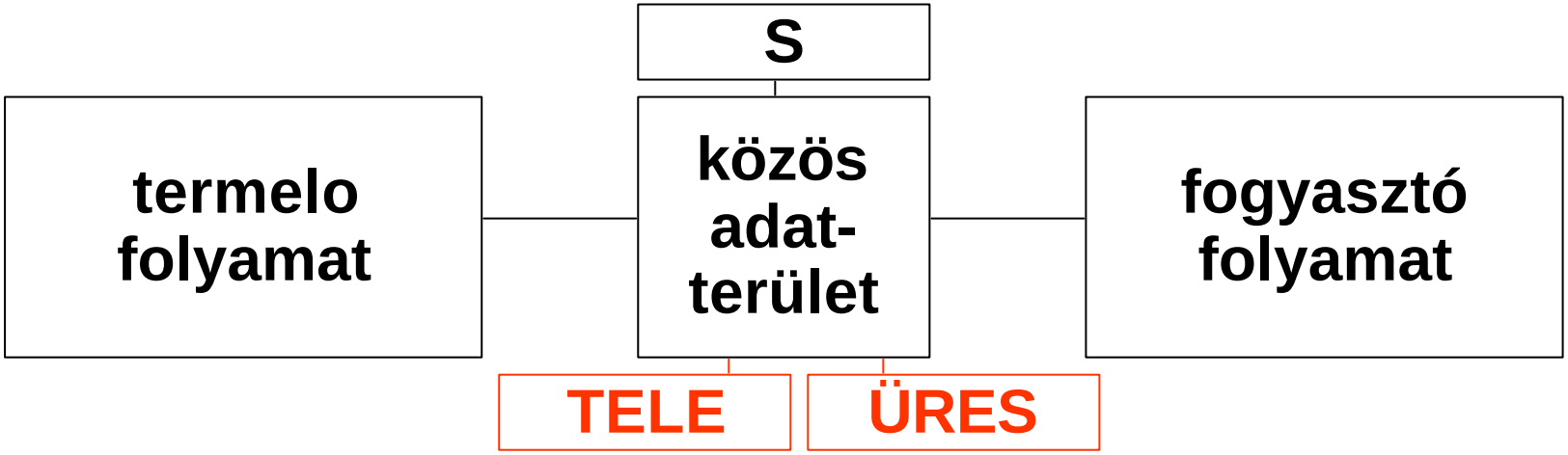
V(S);

Postaláda kezelés



Postaláda: olyan közös adatterület, ahová EGYNÉL TÖBB (pl. N db.) üzenet írható

Postaláda kezelés



Postaláda: olyan közös adatterület, ahová EGYNÉL TÖBB (pl. N db.) üzenet írható

Újabb szemaforok a vezérléshez:

- TELE
- ÜRES

Postaláda kezelés

Postaláda: olyan közös adatterület, ahová EGYNÉL TÖBB (pl. N db.) üzenet írható

3 db. szemafor kell a vezérléséhez

- S: a kölcsönös kizárást megvalósító szemafor (bináris; 0=foglalt; 1=szabad; kezdeti értéke: szabad)
- TELE: a tele helyek száma (nem bináris; értéke 0 és N között lehet; kezdeti értéke:0)
- ÜRES: az üres helyek száma (nem bináris; értéke 0 és N között lehet; kezdeti értéke:N)

P és V primitívek

Primitív: megszakíthatatlan (oszthatatlan) muvelet

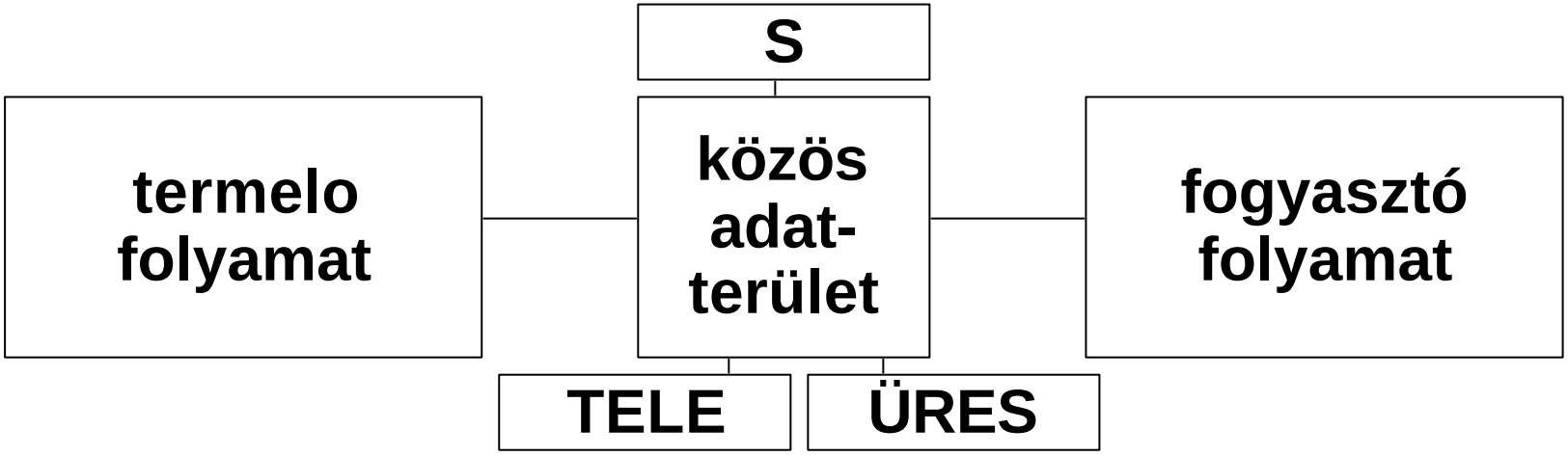
P primitív: a paraméterül kapott szemafor értékének EGGYEL CSÖKKENTÉSE (bináris szemafor esetén ez a FOGLALTTÁ ÁLLÍTÁS)

V primitív: a paraméterül kapott szemafor értékének EGGYEL NÖVELÉSE (bináris szemafor esetén ez a SZABADDÁ ÁLLÍTÁS)

Feltétel:

- SZABAD = 1
- FOGLALT = 0

Postaláda kezelés programja



P(ÜRES);
P(S);
ÍRÁS A MEMÓRIÁBA
V(S);
V(TELE);

P(TELE);
P(S);
OLV. A MEMÓRIÁBÓL
V(S);
V(ÜRES);

Összefoglalás

Az eroforráskezelés alapfogalmai

Holtpont, kiéheztetés

Holtpont feltételei, megelozése

- **Egyetlen foglalási lehetoség**
- **Rangsor szerinti foglalás**
- **BANKÁR ALGORITMUS**

Holtpont felismerése, felszámolása

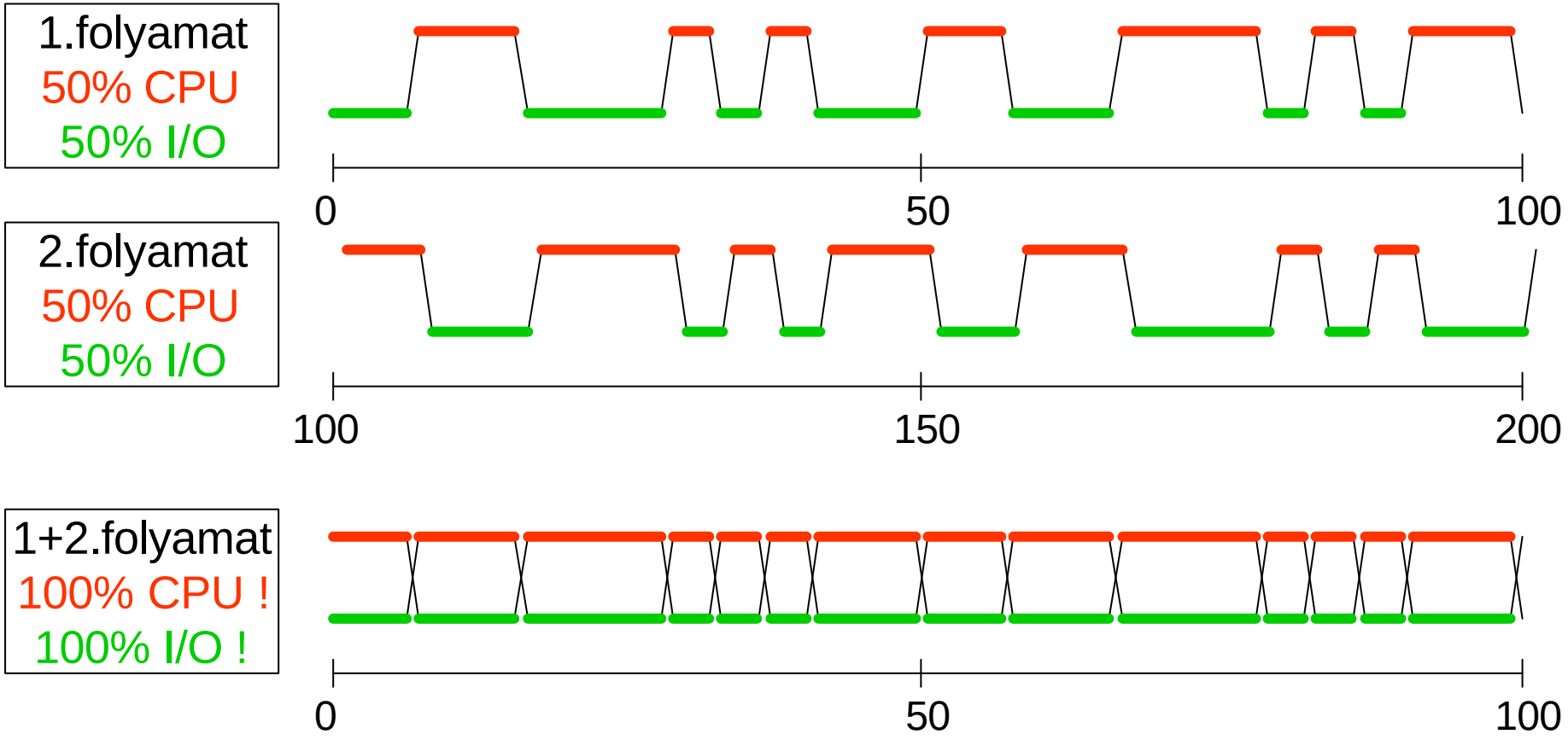
Közösen használt eroforrások

- **Termelo-fogyasztó probléma**
- **Postaláda kezelés**
- **Szemaforok, primitívek**

Folyamat- és processzorkezelés

- Folyamatok létrehozása
- Muveletek folyamatokkal
- Speciális állapotok
- Processzor ütemezés

Processzorkezelés - bevezeto



Az összes futási ido (majdnem) a felére csökken

A CPU és az I/O eszköz kihasználtsága 100%

DE plusz tevékenységeket kell végezni

Operációs rendszer (Eroforrás szemlélet)

A folyamatok egy olyan csoportja, amely a felhasználói folyamatok között elosztja az eroforrásokat

Folyamatok

1. „Életre kelt” programok
2. Olyan programok, amelyeknek van folyamatleíró táblája

Folyamatleíró/vezérlo blokk

(Process Control Block - PCB)

Egyértelmuen azonosítja a folyamatot, minden információ megtalálható benne, ami az operációs rendszer számára szükséges

- A folyamat azonosítója, szülo, gyerekek
- A folyamat állapota (új, futásra kész, aktív, várakozó stb.)
- A PC + regiszterek tartalma (a következo utasítás címe)
- Eroforrások állapota (ki nem elégített I/O igények, a folyamathoz rendelt I/O eszközök, megnyitott állományok)
- CPU ütemezési információ (prioritás, CPU igény)
- Elszámolási adatok, statisztikák (rendszerben eltöltött ido)
- Memóriaterület adatai, címtranszformációs táblák

Várakozási sor (queue)

A folyamatok közül egyidejuleg csak egy futhat, tehát a többinek valamire várnia kell (I/O eszköz, CPU, memória stb.), tehát egy várakozási sorba kell állnia.

Funkció: A folyamatok (PCB-k) egy olyan sora, melyben minden elem utal az ot követo elemre

Megvalósítás: láncolt adatstruktúra (lista)

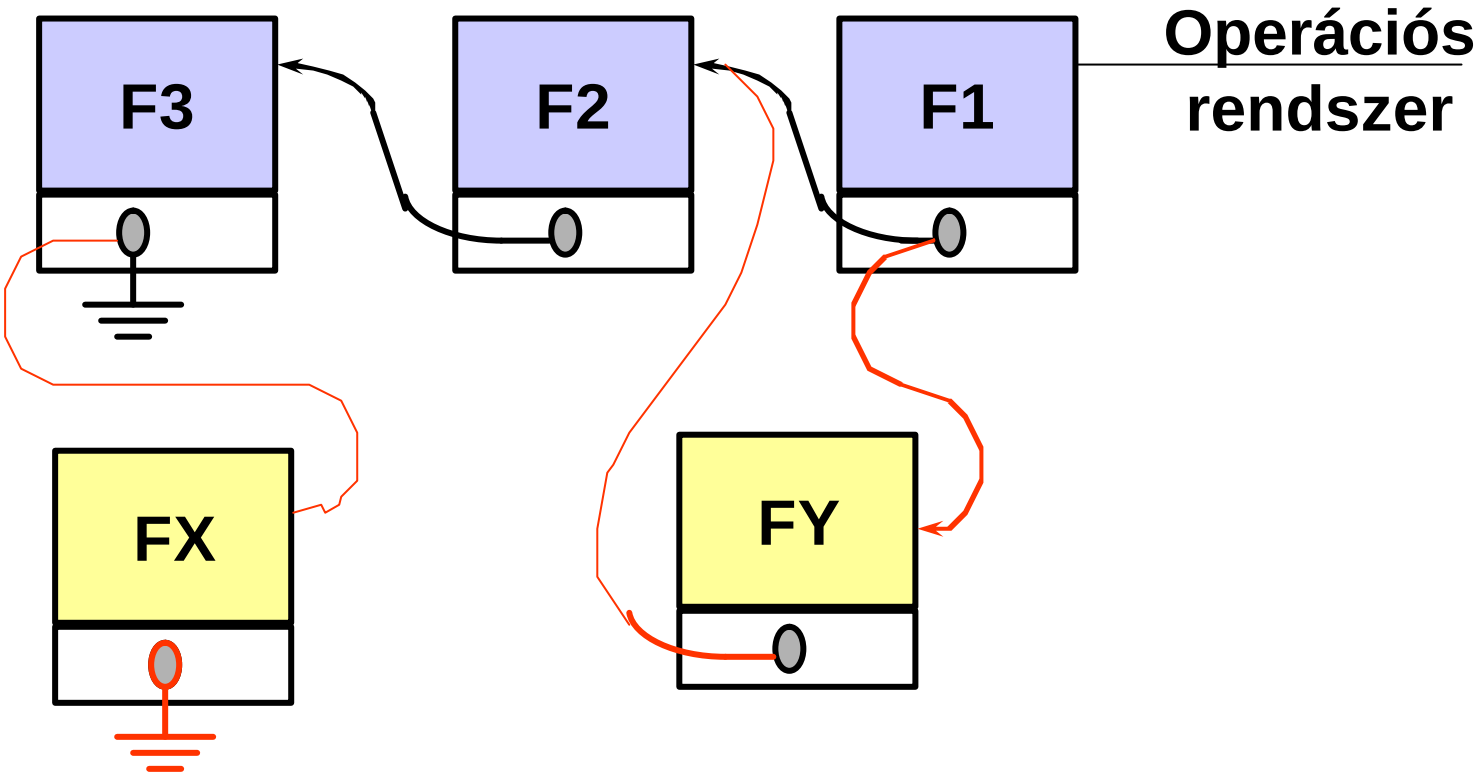
Tipikus: FIFO (First In First Out)

Muveletek: hozzáfuzés, beékelés

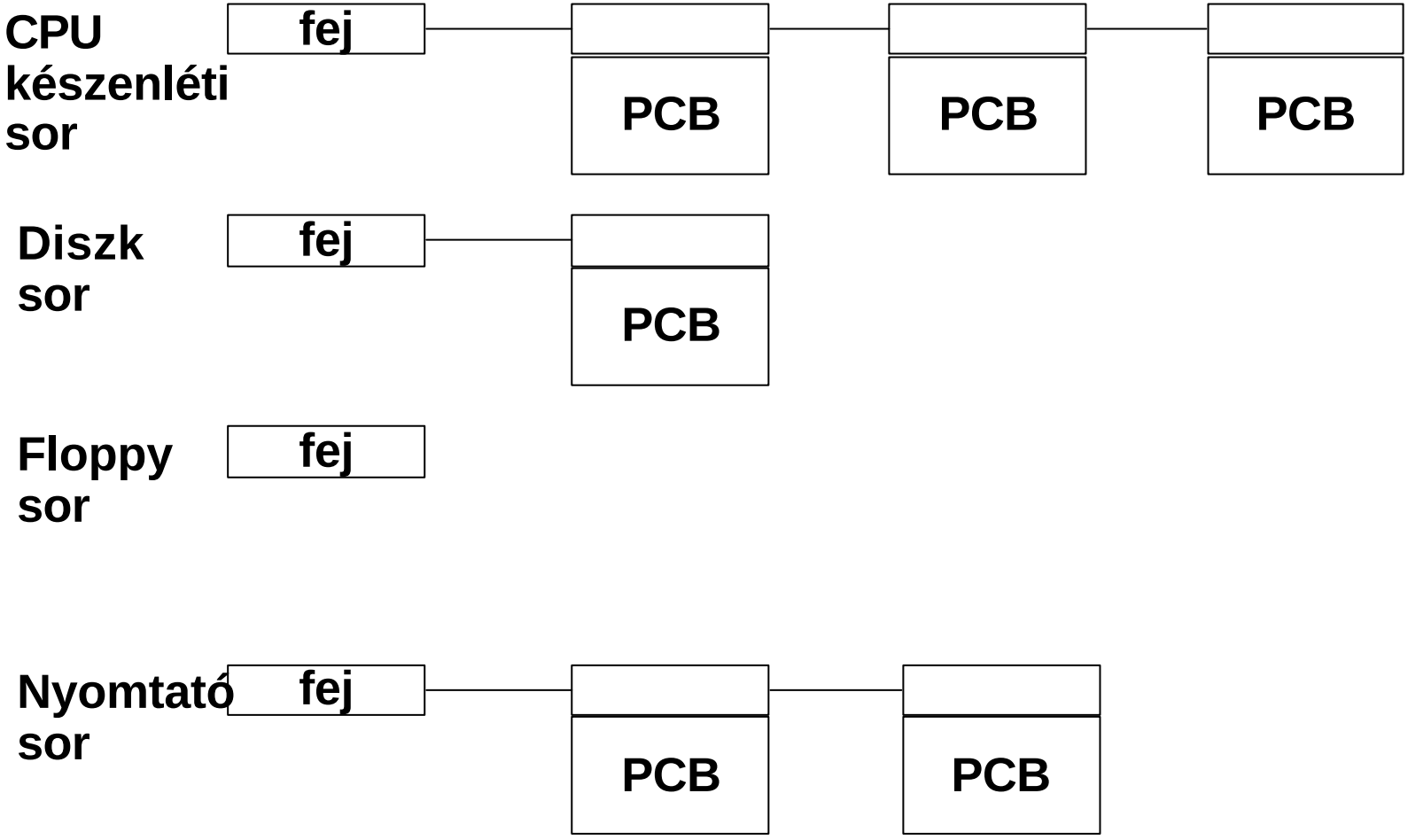
Lista deklarációja

```
type
 PCB = record
        ProgramCounter: integer;
        ProcessorState: integer;
        Registers: array 1:16 of integer
        {egyéb adatok}
     end;
 PCBpointer = ^PCB;
 Lista = record
        process: PCB;
        NextProcess: PCBpointer;
     end;
```

Lista (sor) bovítése



PCB-k használata - példa

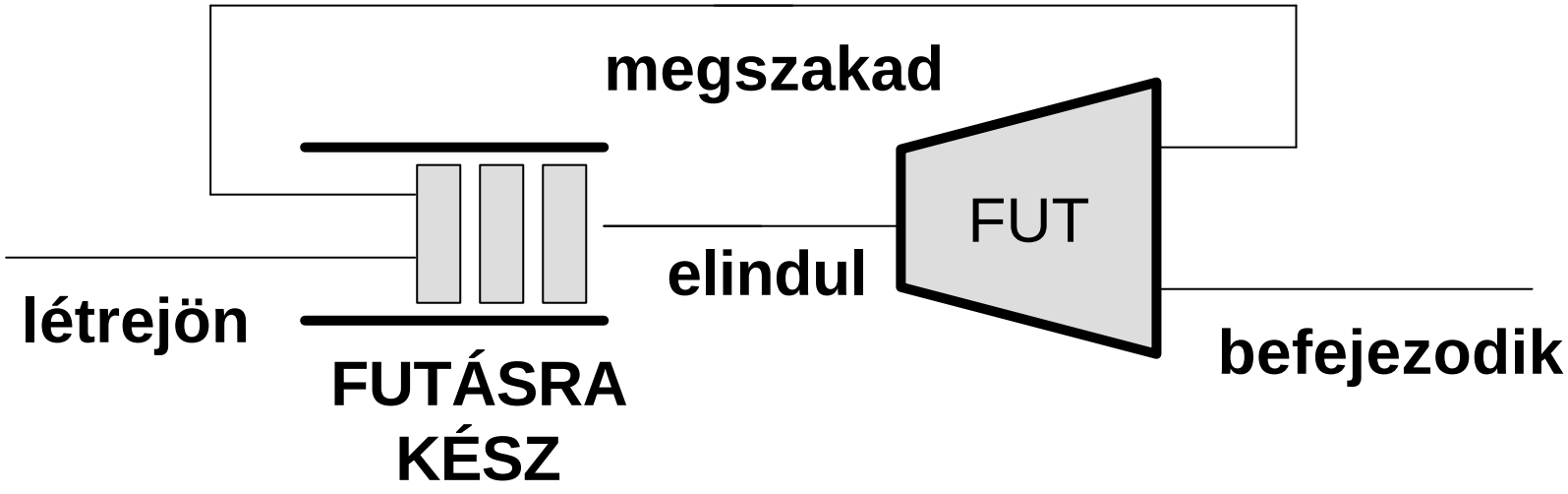


Folyamatok állapotai

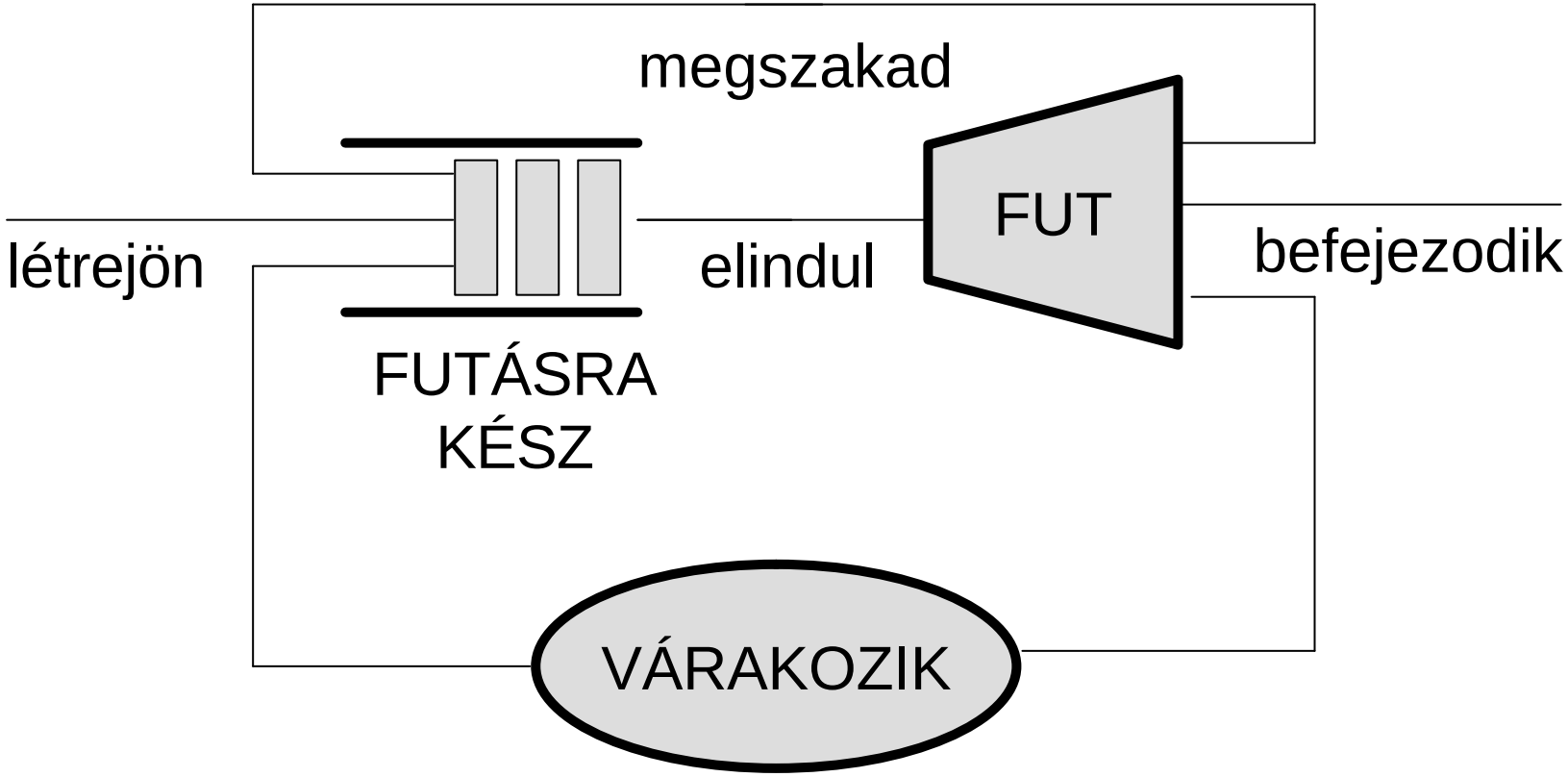
A várakozási sorok megfelelnek a folyamatok állapotainak.

- **FUTÁSRA KÉSZ** (már csak a CPU hiányzik)
  - Megszakított (pl. kivétel, I/O muvelet)
- **FUT**
- **VÁRAKOZIK** (további eroforrásra)
  - Holtponti (ha reménytelenül vár valamire)
- **FELFÜGGESZTETT** (eroforrás nélkül, csak PCB)

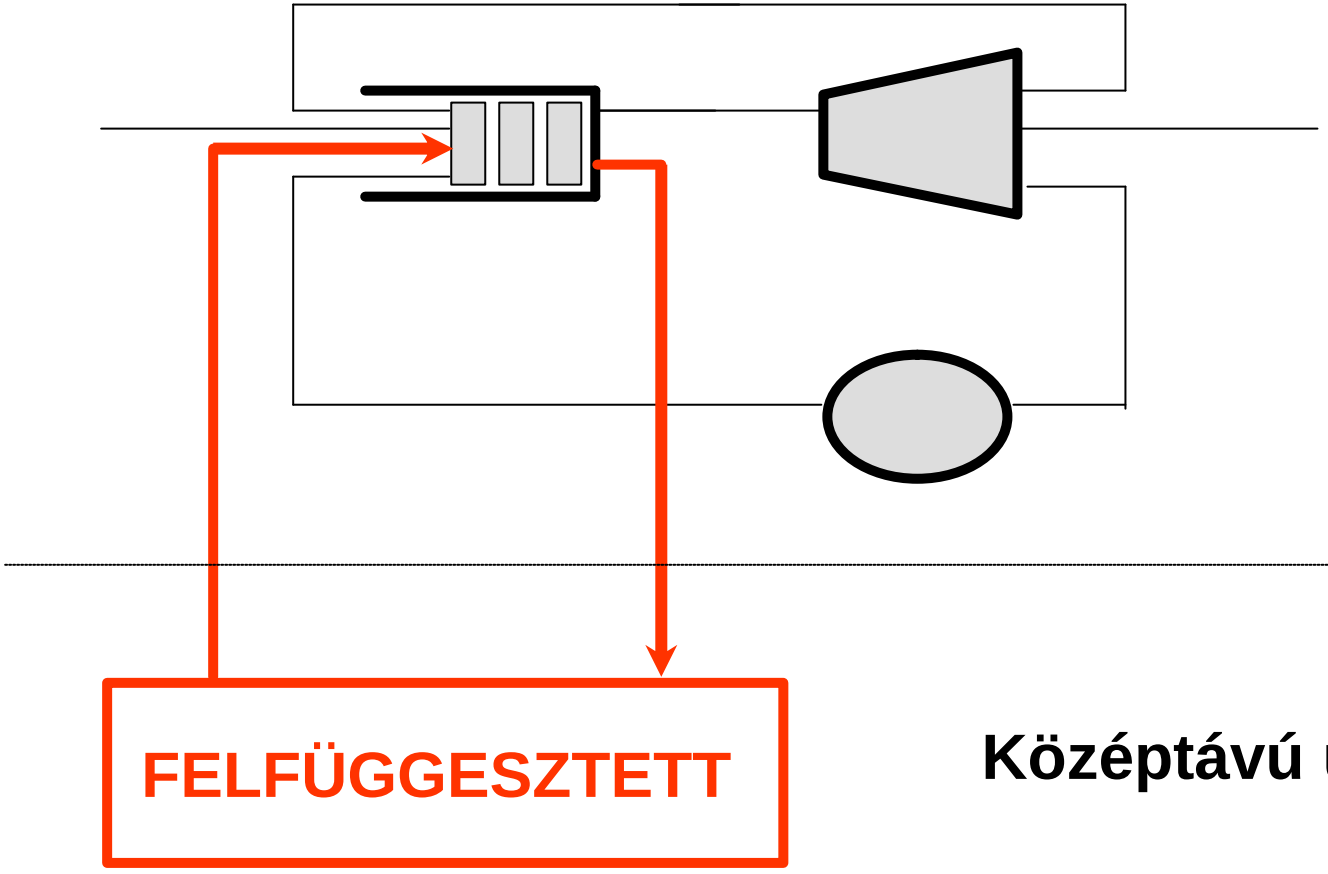
Folyamatok állapotai I.



Folyamatok állapotai II.

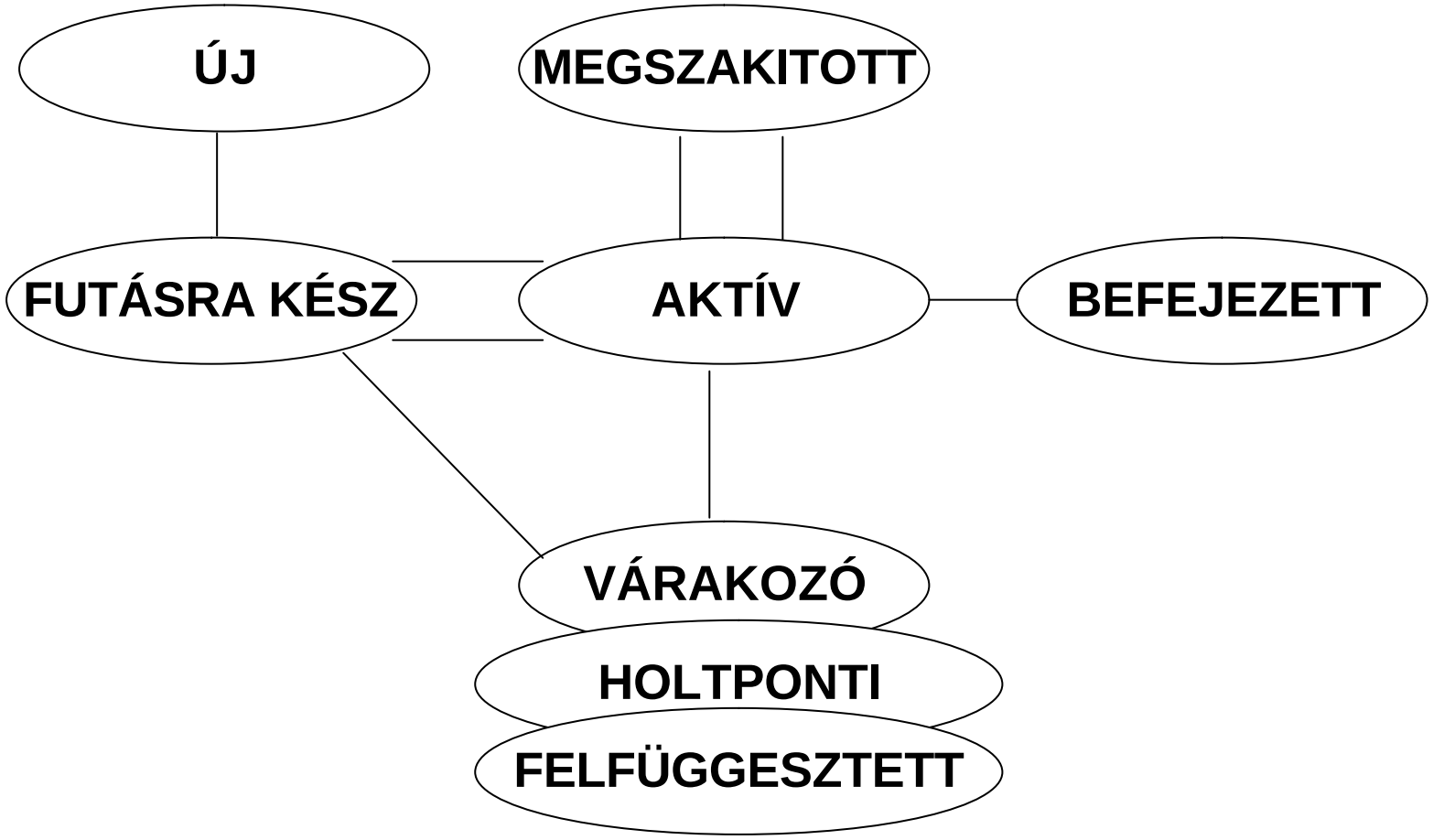


Folyamatok állapotai III.

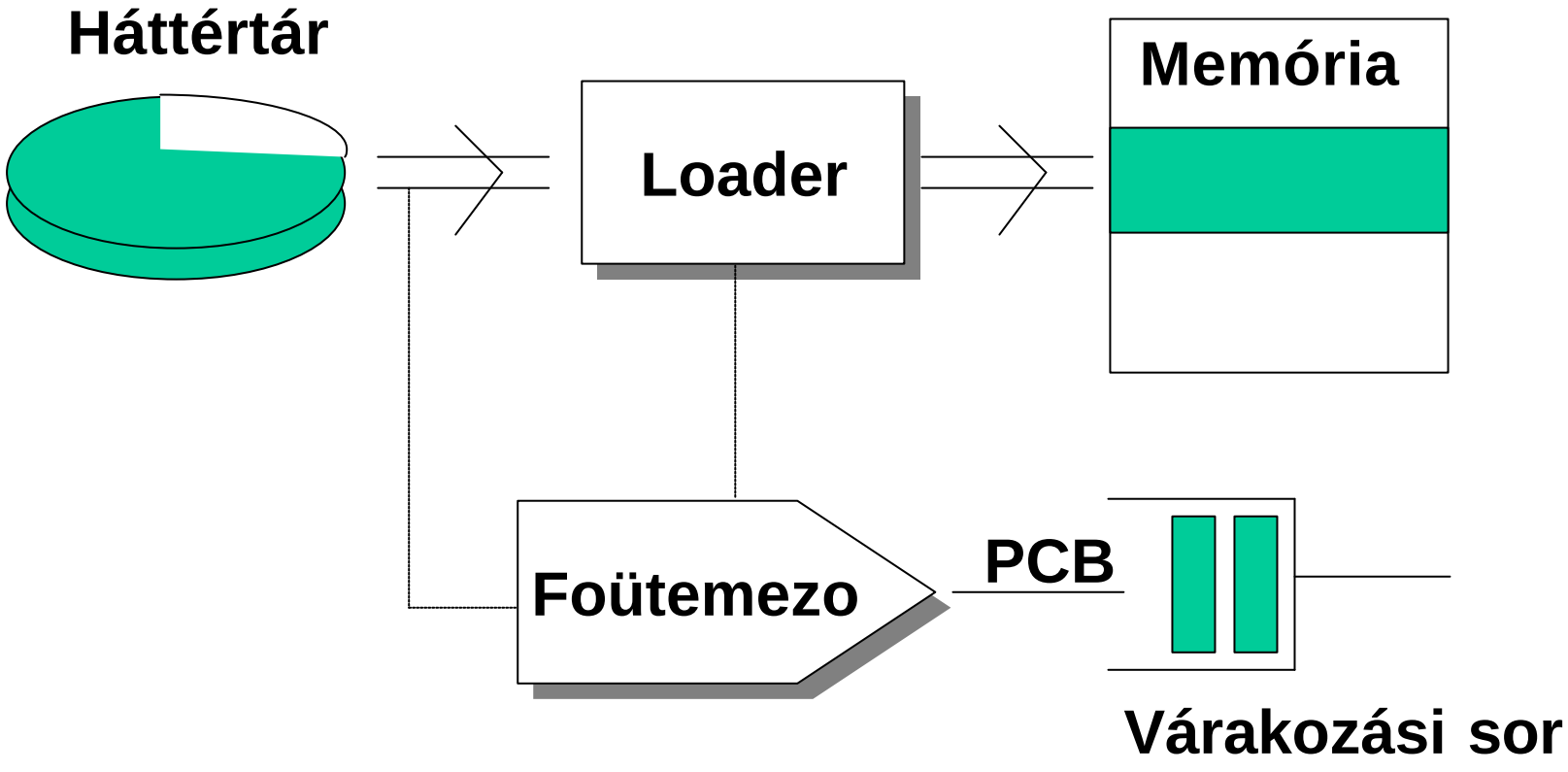


Középtávú ütemezo

A folyamatok állapotai



A Foütemezo



PROGRAM — FOLYAMAT

Magas szintu ütemezo

A diszken várakozó folyamatok közül “enged be” újat a processzor várakozási sorába

A kiválasztott folyamathoz elkészíti a PCB-t

Viszonylag ritkán fut - nem sebesség kritikus

Interaktív rendszereknél háttérbe szorul

Optimális munkakeveréket töltsön be

- CPU orientált folyamatok (viszonylag ritka)
- I/O orientált folyamatok

Közbenso szintu ütemezo

Nagyobb rendszerekben található meg

Olyan szempontokat figyel, amelyre az alacsony szintu ütemezonek nincs ideje

- pl. folyamatok futási sebességének figyelése
  - kiéheztetést tudja detektálni
  - észreveheti, ha egy folyamat aránytalanul lassan fut
- a megbomlott egyensúly helyreállítása
  - folyamatok prioritásának megváltoztatása
  - folyamatok ideiglenes / végleges felfüggesztése

“Globális optimum”

Alacsony szintu ütemezo

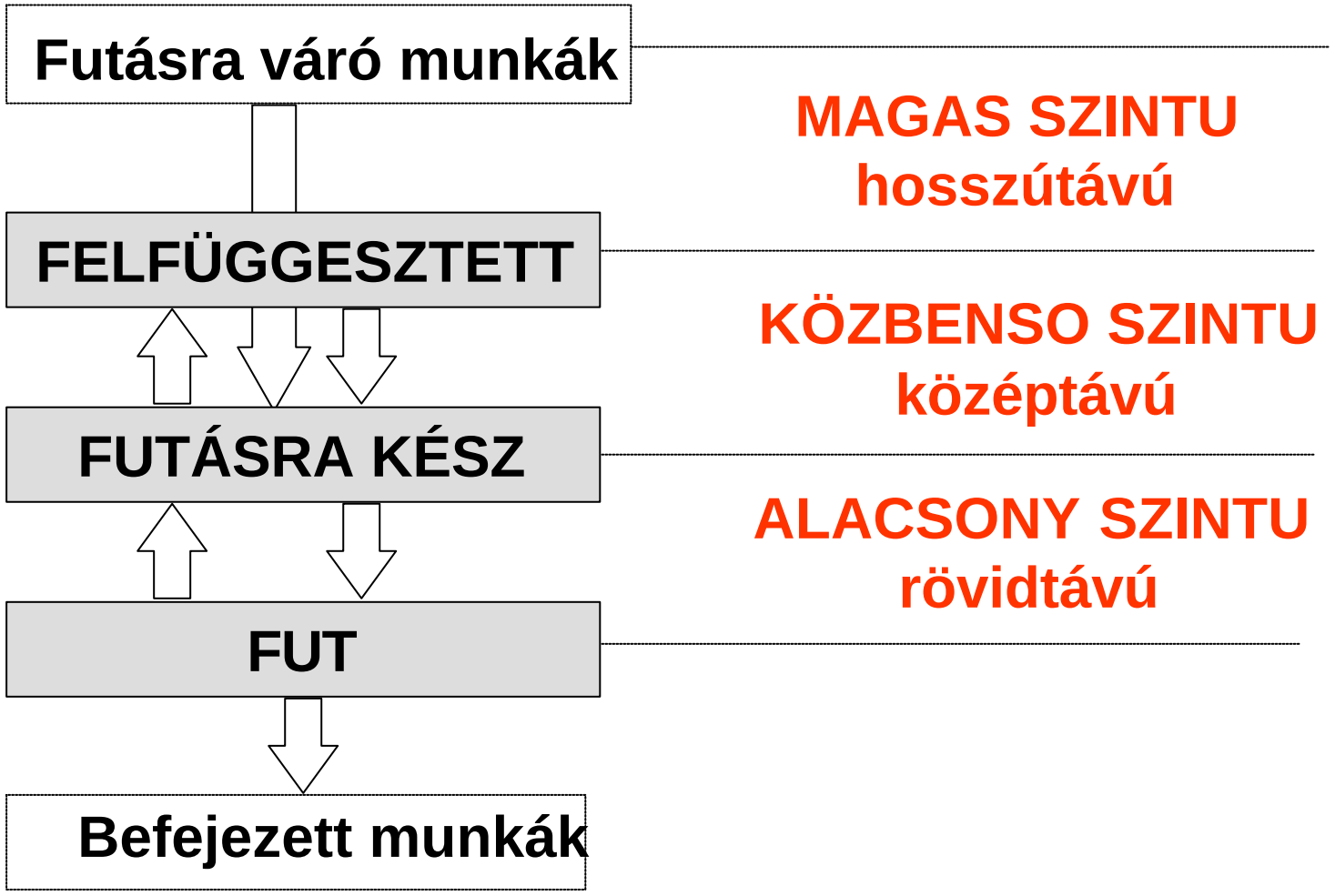
A CPU-ra várakozó folyamatok közül választja ki azt, amelyik a következo idoben használja a CPU-t

Aktivizálja a kiválasztott folyamatot (regiszterek visszaállítása a PCB-bol, vezérlésátadás) - diszpécser

Gyakran fut - sebességkritikus (különösen idoosztásos és valósideju rendszereknél)

Gyors kell legyen => viszonylag kevés szempontot vehet figyelembe a választáshoz (“lokális optimum”)

Ütemezési szintek



Ütemezési statisztikák

$T_{ÉRKEZÉS}$	**A folyamat a futásra kész sorba érkezik**
T_{KEZDET}	**A futás kezdoidopontja**
$T_{BEFEJEZÉS}$	**A folyamat elhagyja a rendszert**
$T_{IGÉNY}$	**A folyamat processzorido igénye**

$$T_{VÁRAKOZÁS} = T_{BEFEJEZÉS} - T_{ÉRKEZÉS} - T_{IGÉNY}$$

$$T_{VÁLASZ} = T_{KEZDET} - T_{ÉRKEZÉS}$$

$$T_{ÁTFUTÁS} = T_{BEFEJEZÉS} - T_{KEZDET}$$

Az ütemezési módszerek minosítése

Szimuláció mintavétel alapján

- Egy rendszer statisztikai adatait gyujtjük, ezek az adatsorok szolgálnak mintasorozatként

Szimuláció valószínuségi modell alapján

- Az összegyujtött adatok alapján valószínuségi modellt állítunk fel (Poisson-eloszlás), ennek alapján generálunk mintasorozatot

Analitikus számítások

Tapasztalat

Ütemezési algoritmusok

Optimalizálási szempontok

CPU kihasználtság

A rendszer átbocsátóképessége (pl. a befejezett feladatok száma óránként)

Fordulási ido (feladat leadása és befejezése közti ido)

Várakozási ido (a CPU-ra várakozási sorban töltött ido - tkp. EZT befolyásolják az ütemezési algoritmusok)

Válaszido (interaktív rendszereknél nagyon fontos)

Elobb jött - elobb fut

First Come First Served - FCFS

A folyamatok érkezési sorrendjükben kapják meg a processzort

Elony:

- a legegyszerubb stratégia, biztos

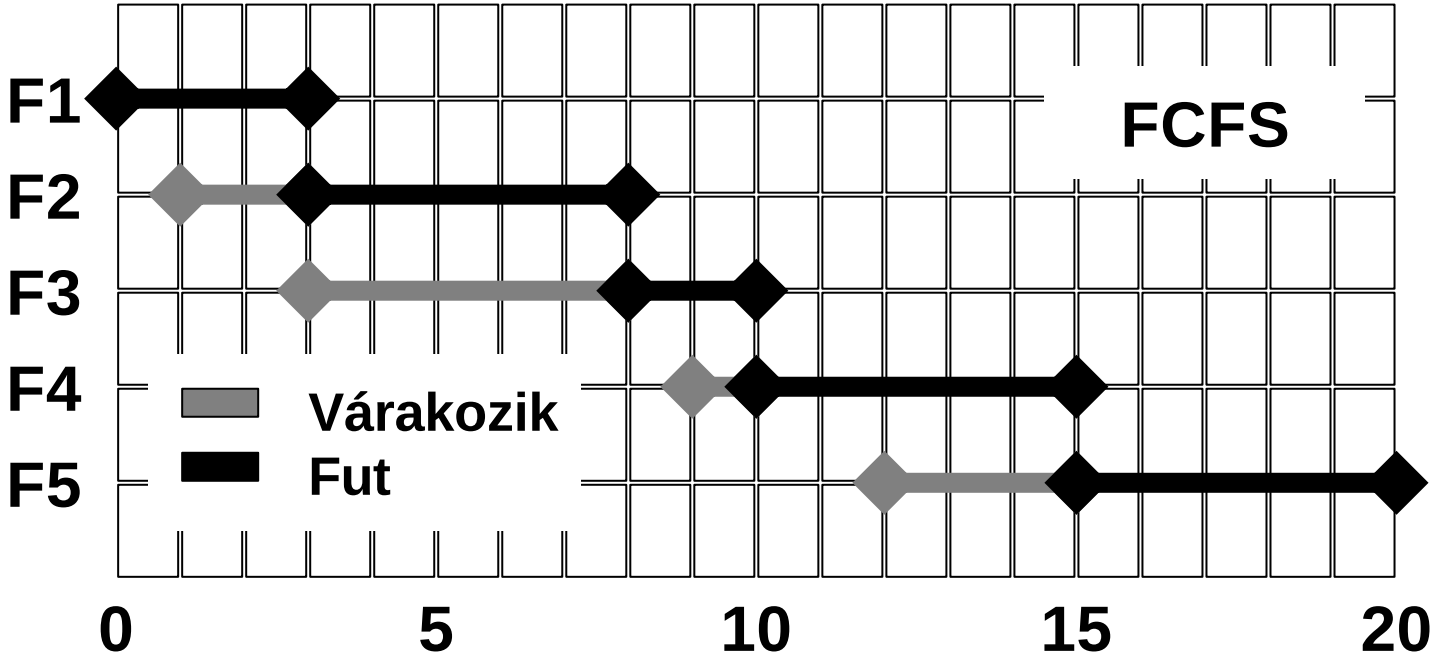
Hátrány:

- a folyamatok várakozási, fordulási ideje nagymértékben függ a folyamatok érkezési sorrendjétol
- „lassú kamion” - effektus, csorda hatás
- nagy várakozási ido, rossz hatékonyság

FCFS példa

	Érkezés	Igény	Kezdés	Befejezés	Várakozás
F1	0	3	0	3	0
F2	1	5	3	8	2
F3	3	2	8	10	5
F4	9	5	10	15	1
F5	12	5	15	20	3
			Átlagos ido		11/5

FCFS idozítés



Átlagos várakozási ido számítás - FCFS

Határozza meg az alábbi terhelés esetén az átlagos várakozási ido értékét!

PR.	ÉRK. IDO	CPU IGÉNY
F1	0	14
F2	7	8
F3	11	36
F4	20	10

Átlagos várakozási ido számítás - FCFS

PR.	ÉRK. IDO	CPU IGÉNY
P1	0	14
P2	7	8
P3	11	36
P4	20	10

PR.	ÉRK. IDO	CPU IGÉNY	KEZD. IDOPONT	BEF.	VÁRAKOZÁSI IDO (KEZD. - ÉRK.)
P1	0	14	0	14	0
P2	7	8	14	22	7
P3	11	36	22	58	11
P4	20	10	58	68	38
					56

ÁTLAGOS VÁRAKOZÁSI IDO: 56 / 4 = 14

A legrövidebb elonyben

Shortest Job First - SJF

A CPU-t egy folyamat befejezodése után a legrövidebbnek adja oda (ha több ilyen van, FCFS szerint választ közülük), általában kötegelt rendszereknél van ilyen

Elony:

- a legrövidebb az átlagos várakozási ido

Hátrány:

- KIÉHEZTETÉS (hosszú folyamatokkal mostohán bánik)
- Tudni kell ELORE a folyamat hosszát
  - kötegelt rendszereknél programozói becslés
  - idoosztásos rendszereknél statisztikai becslés
  - mi történjen, ha a becslés rossz?

Preemptív / nem preemptív algoritmusok

Az SJF-nek és a prioritásos algoritmusoknak vannak preemptív formái is

Preempció: ha egy rövidebb (SJF) vagy egy magasabb prioritású (prioritásos algoritmus) folyamat érkezik, az MEGSZAKÍTJA az éppen futó folyamatot

Elony: kisebb átlagos várakozási idot ad (SJF), illetve jobban preferálja a magasabb prioritású folyamatokat (prioritásos algoritmus)

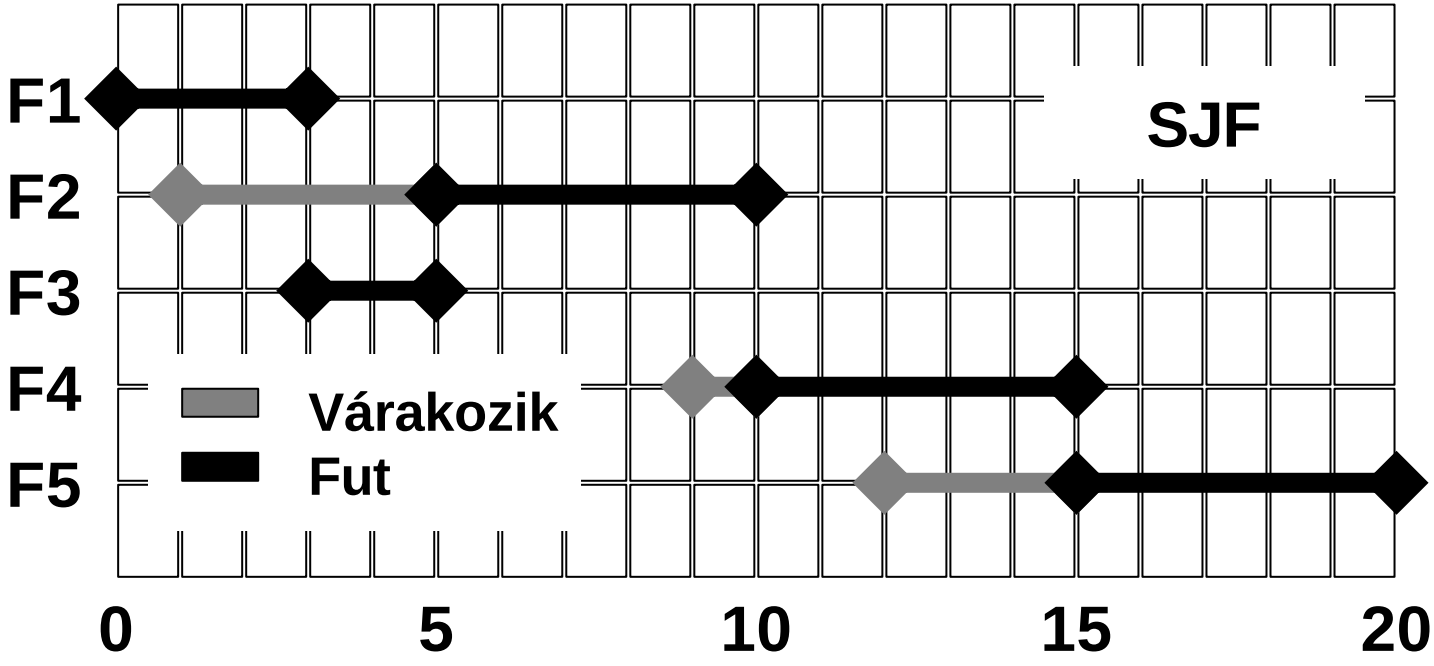
Hátrány: a megszakításkor környezetváltásra van szükség

SJF - példa

	Érkezés	Igény
F1	0	3
F2	1	5
F3	3	2
F4	9	5
F5	12	5

Kezdés	Befejezés	Várakozás
0	3	0
5	10	4
3	5	0
10	15	1
15	20	3
Átlagos ido		8/5

SJF idozítés



Átlagos várakozási ido számítás - SJF

Határozza meg az alábbi terhelés esetén az átlagos várakozási ido értékét!

PR.	ÉRK. IDO	CPU IGÉNY
F1	0	14
F2	7	8
F3	11	36
F4	20	10

Átlagos várakozási ido számítás - SJF

PR.	ÉRK. IDO	CPU IGÉNY
P1	0	14
P2	7	8
P3	11	36
P4	20	10

PR.	ÉRK. IDO	CPU IGÉNY	KEZD. IDOPONT	BEF.	VÁR. IDO	VÁRÓ PROC.	LEG-RÖVIDEBB
P1	0	14	0	14	0	P2(8), P3(36)	P2
P2	7	8	14	22	7	P3(36), P4(10)	P4
P4	20	10	22	32	2	P3(36)	P3
P3	11	36	32	68	21	-	-
					30		

ÁTLAGOS VÁRAKOZÁSI IDO: 30 / 4 = 7,5

Körbenjáró

Round Robin - RR

A folyamatokat egy zárt körbe szervezzük, és minden folyamat egy elore rögzített IDOSZELET - re kapja meg a processzort, majd visszaáll a sor végére

Tipikusan az interaktív rendszerek stratégiája

Elony:

- egyszeru algoritmus, kis *válaszido*
- nincs kiéheztetés, demokratikus

Hátrány:

- az idoszelet lejártakor a folyamat állapotát el kell menteni - idoveszteség

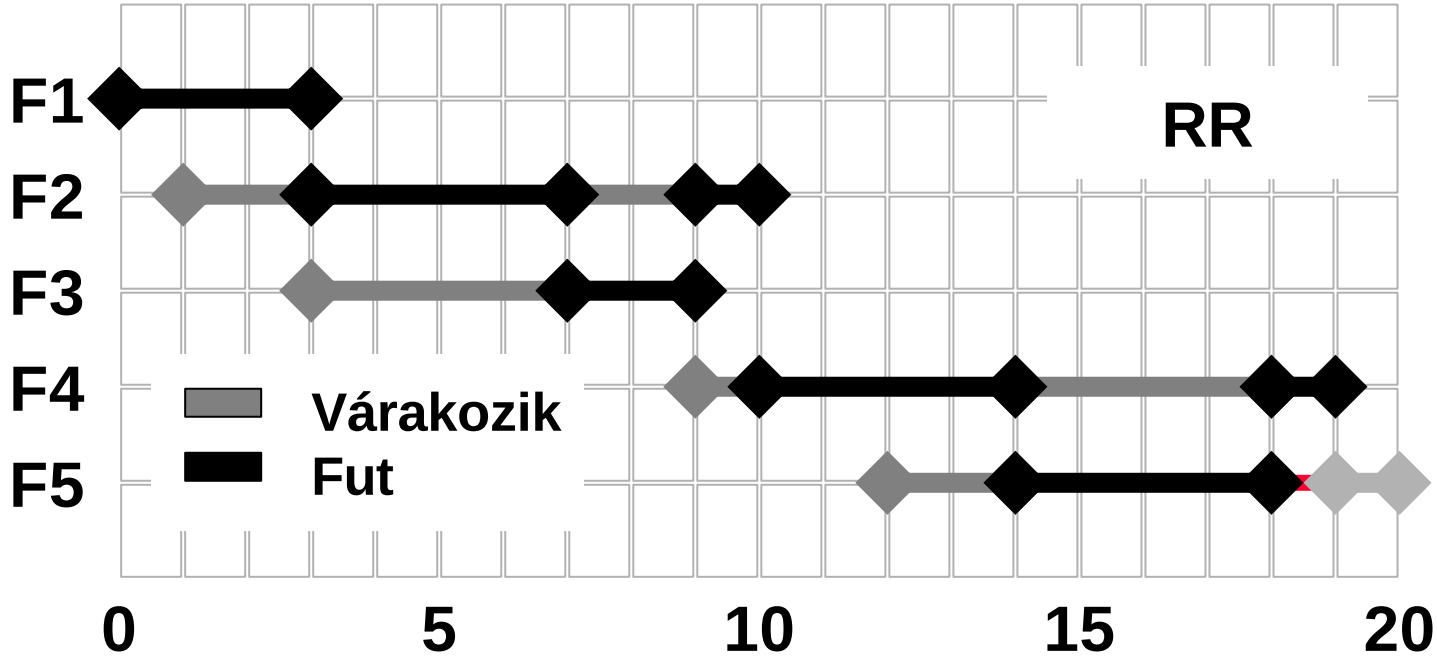
RR - példa

	Érkezés	Igény
F1	0	3
F2	1	5
F3	3	2
F4	9	5
F5	12	5

Befejezés	Várakozás
3	0
10	4
9	4
19	5
20	3
Átlagos ido	16/5

Idoszelet = 4 egység

RR idozítés



Átlagos várakozási ido számítás - RR

Határozza meg az alábbi terhelés esetén az átlagos várakozási ido értékét, ha az idoszelet 10!

PR.	ÉRK. IDO	CPU IGÉNY
F1	0	14
F2	7	8
F3	11	36
F4	20	10

Átlagos várakozási ido számítás - RR

PR.	ÉRK. IDO	CPU IGÉNY
P1	0	14
P2	7	8
P3	11	36
P4	20	10

ÁTLAGOS VÁRAKOZÁSI IDO: 44 / 4 = 11

PR.	ÉRK. IDO	CPU IGÉNY	KEZD. IDOPONT	BEF.	VÁR. IDO	MARAD. IDO	VÁRÓ PROC.
P1	0	14	0	10	0	4	P2, P1
P2	7	8	10	18	3	-	P1, P3
P1*	(10)	4	18	22	8	-	P3, P4
P3	11	36	22	32	11	26	P4, P3
P4	20	10	32	42	12	-	P3
P3*	(32)	26	42	52	10	16	P3
P3*	(52)	16	52	62	0	6	P3
P3*	(62)	6	62	68	0	-	-
					44		

Prioritásos módszerek

Minden folyamathoz egy prioritási értéket rendelünk, és a CPU-t a legmagasabb prioritású folyamat kapja meg

- külso prioritás - a folyamat “fontosságától“ függo érték
- belso prioritás - az operációs rendszer határozza meg pl. eroforrás igény alapján

Elony:

- a prioritásokkal sokféle szempont érvényesítheto

Hátrány:

- KIÉHEZTETÉS (kis prioritású folyamaté)
- Megszüntetése: öregedés (prioritás növelése adott idonként)

Összefoglalás

Folyamatok állapotai, a PCB

Várakozási sorok (queue)

Ütemezési szintek

magas szintu, közbenso szintu, alacsony szintu

Ütemezési stratégiák

FCFS, SJF, RR

Memóriakezelés

- Valóságos tárkezelés
- Virtuális memória
- Tárvédelem, szegmentálás
- Gyorstárak, Tároló hierarchia

Abszolút címzésu rendszerek

a tárat két részre osztják

- operációs rendszer rezidens része
- felhasználó területe

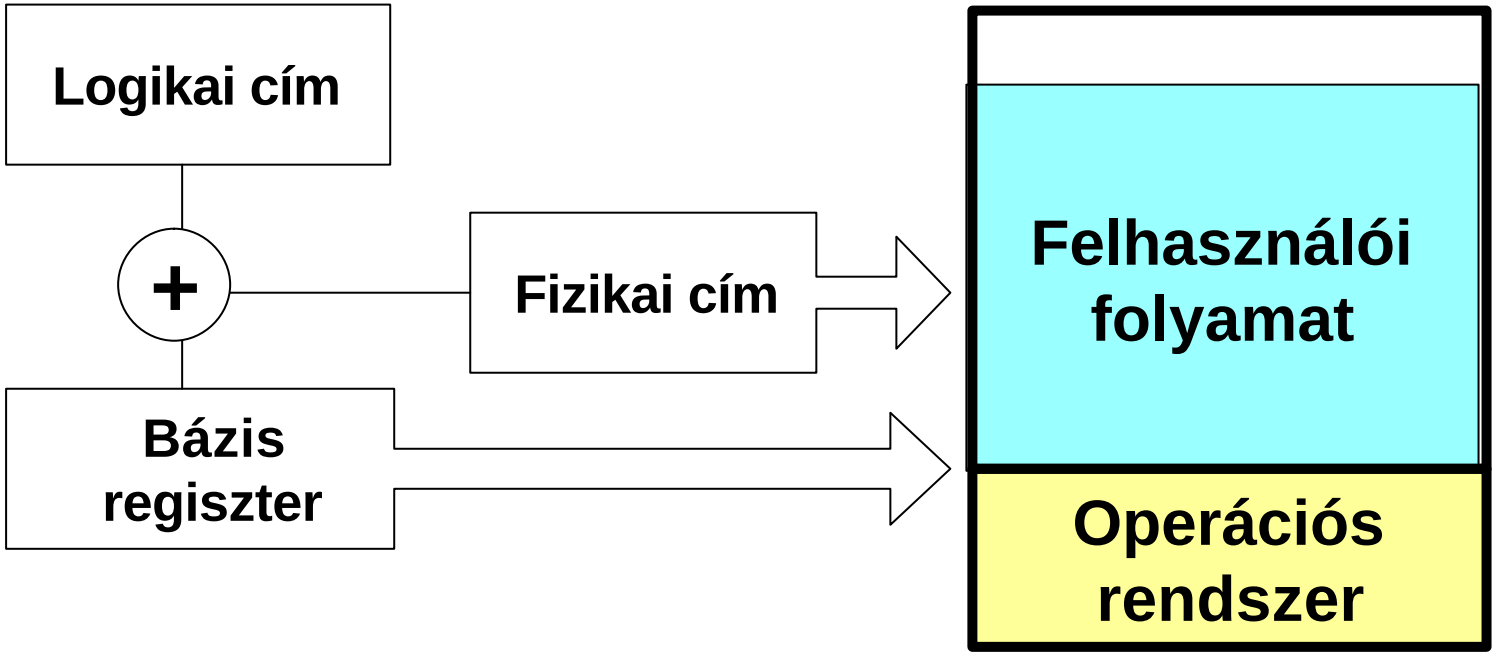
a program helye már írásakor ismert

az utasítások közvetlenül címezhetik a teljes tárat

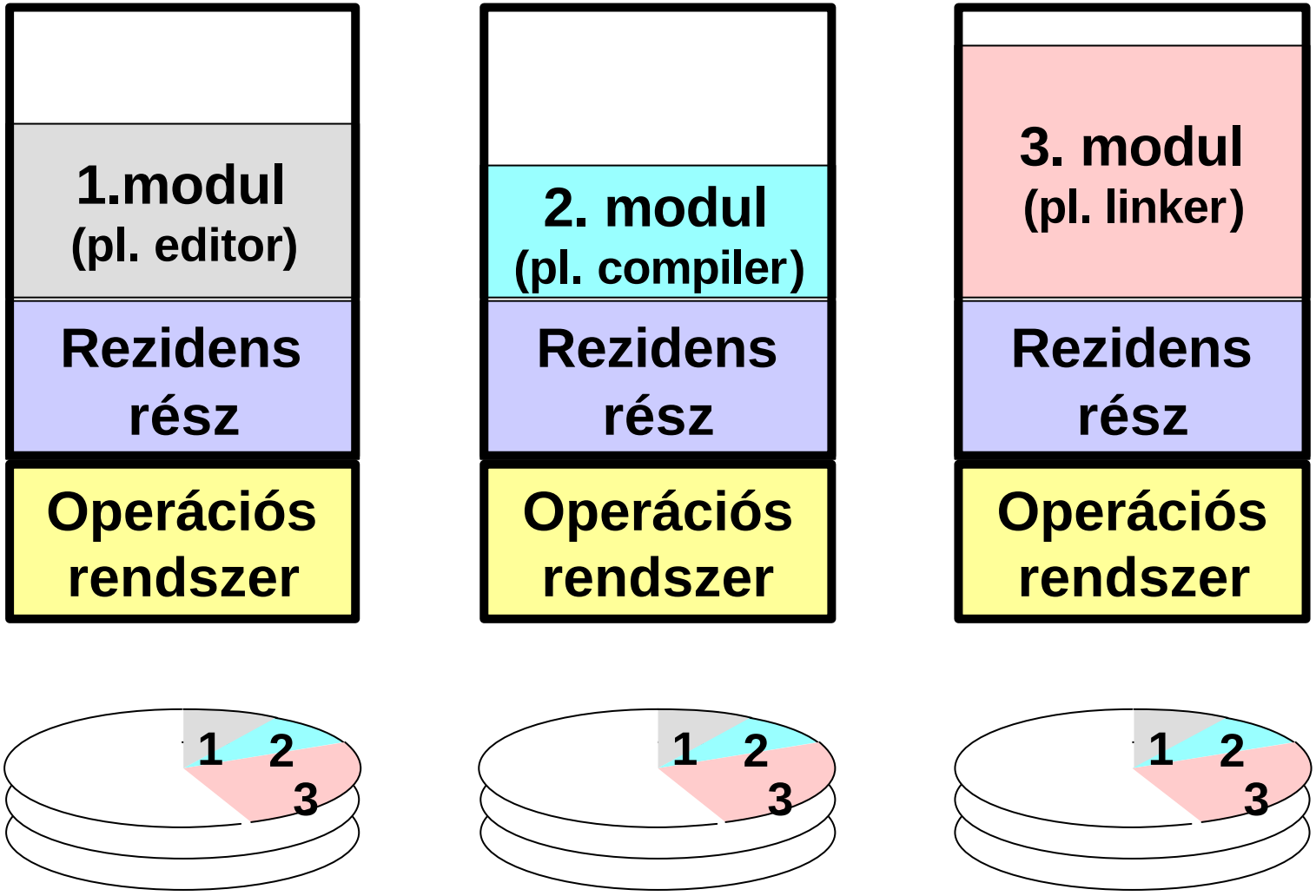
rugalmatlan (mi van, ha az operációs rendszer mérete no?)

multiprogramozásra alkalmatlan

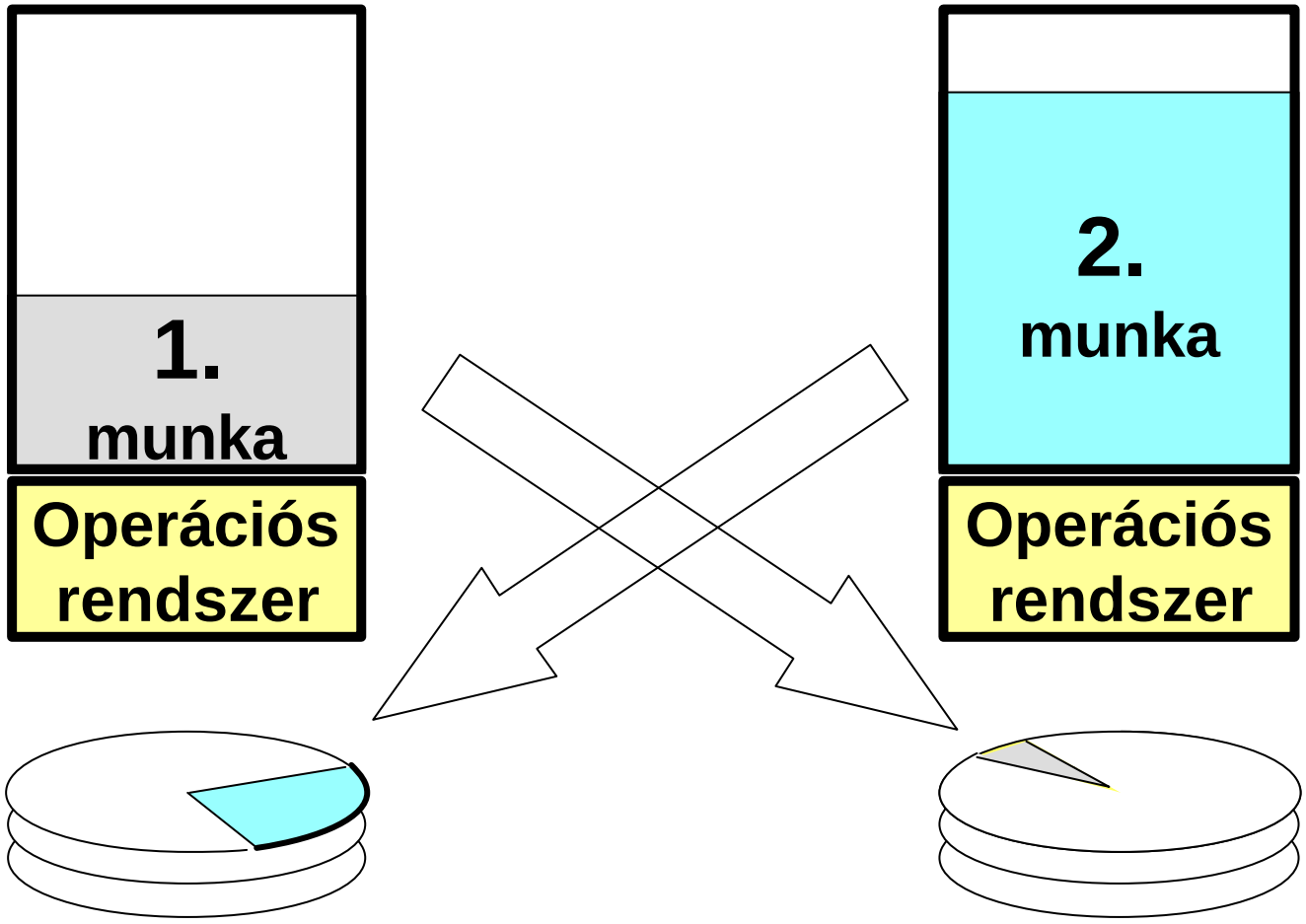
Áthelyezheto címzés



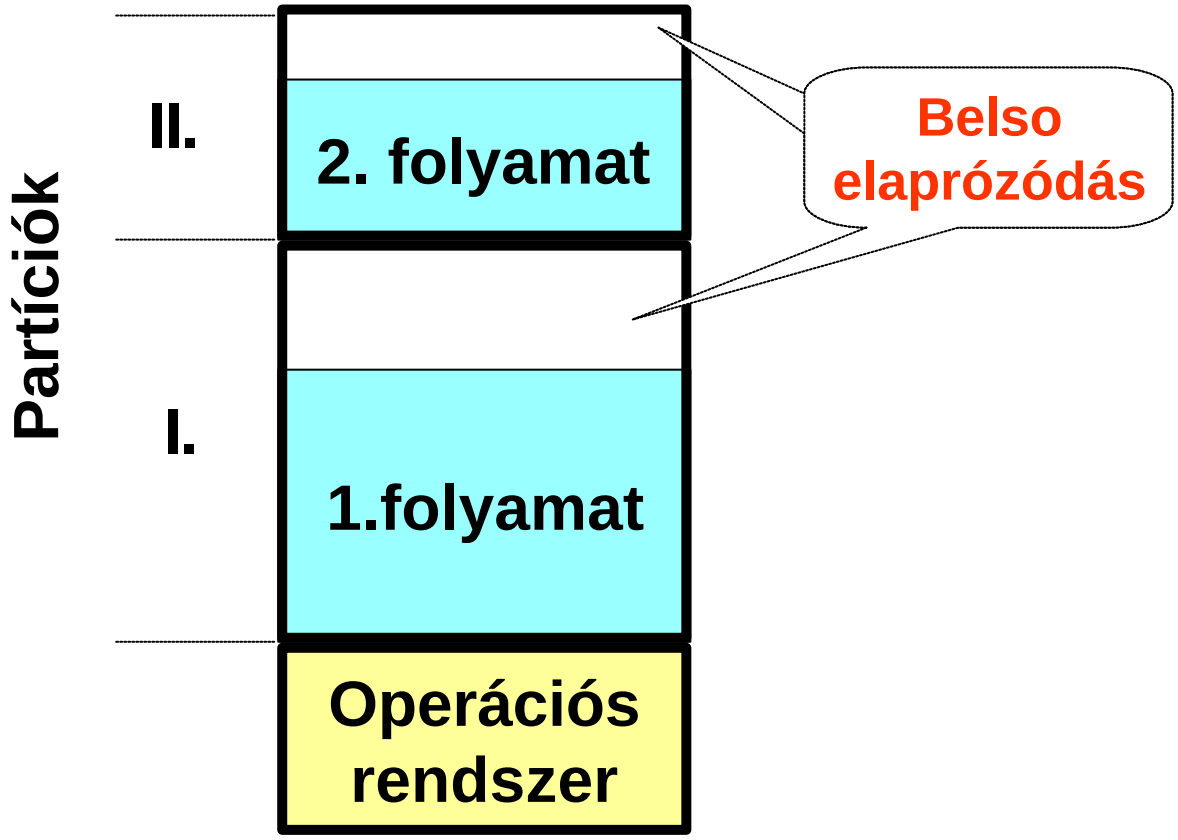
Átlapoló (overlay) technika



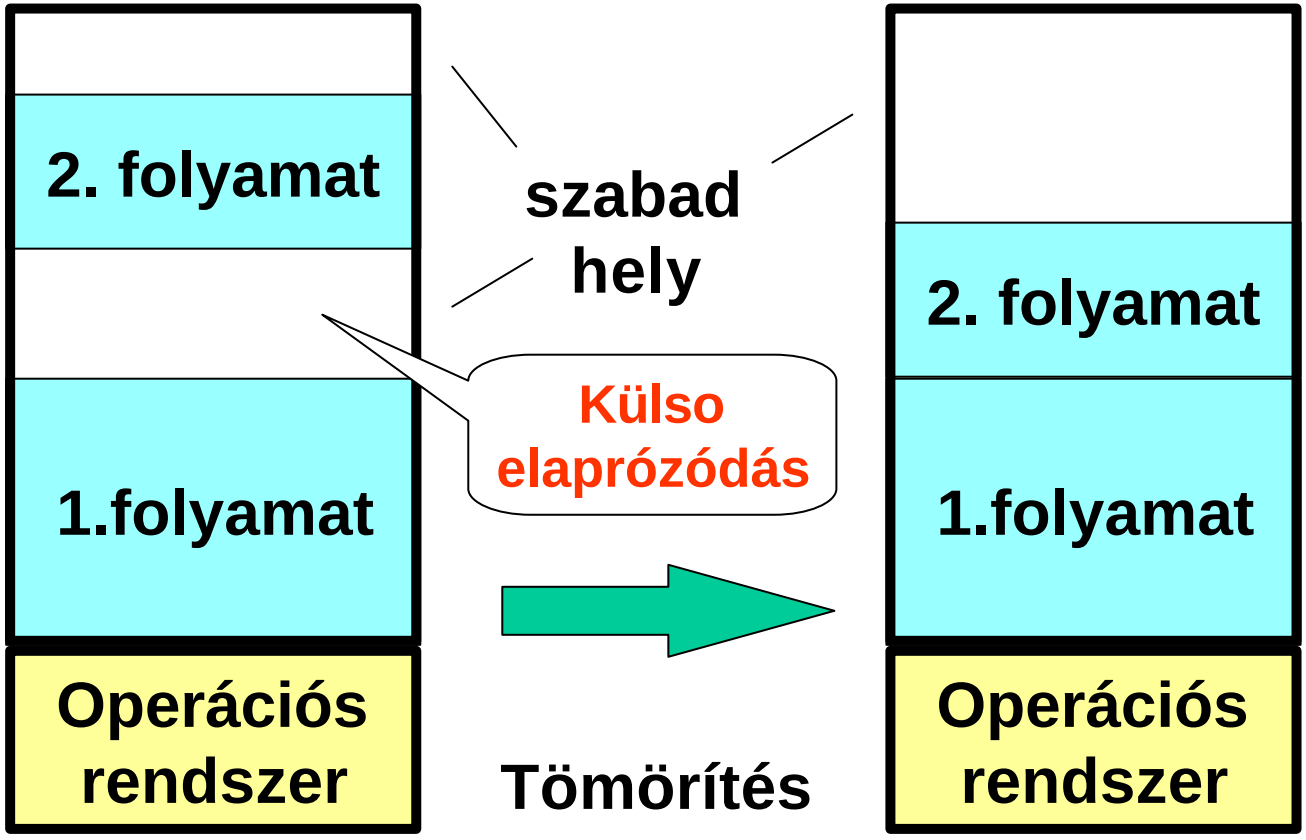
Tárcsere (swapping)



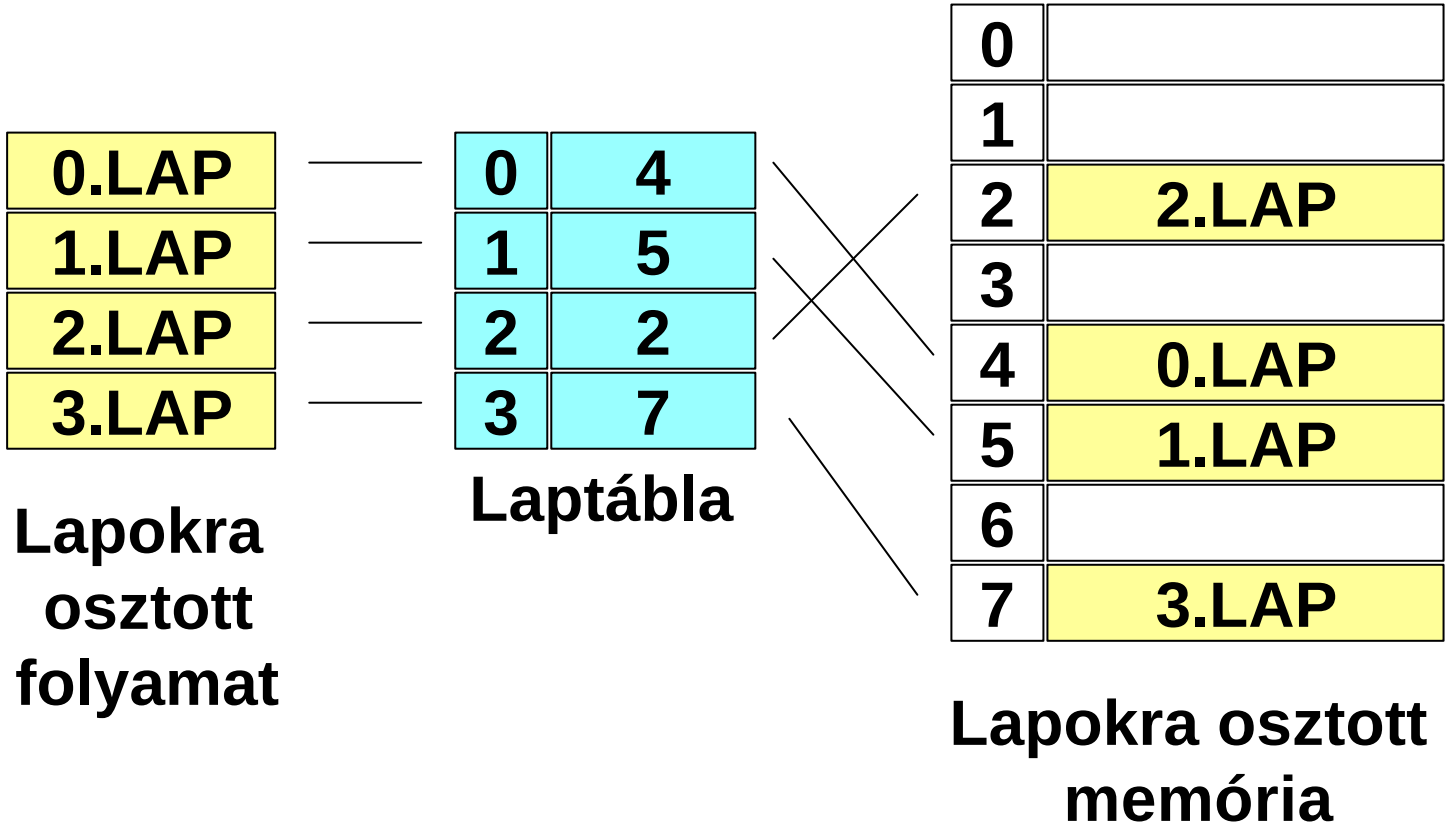
Particionált rendszer



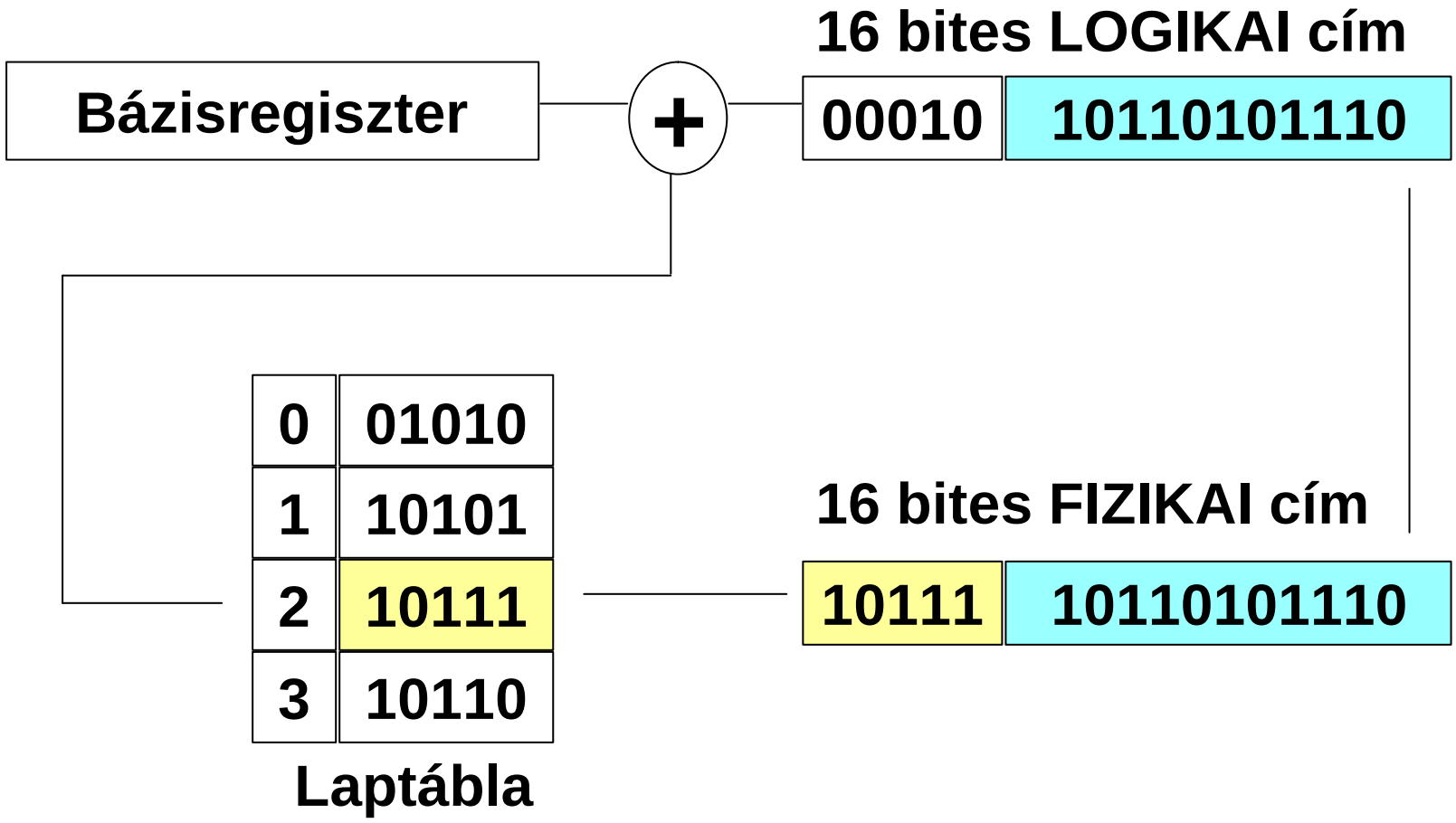
Rugalmas partíciók



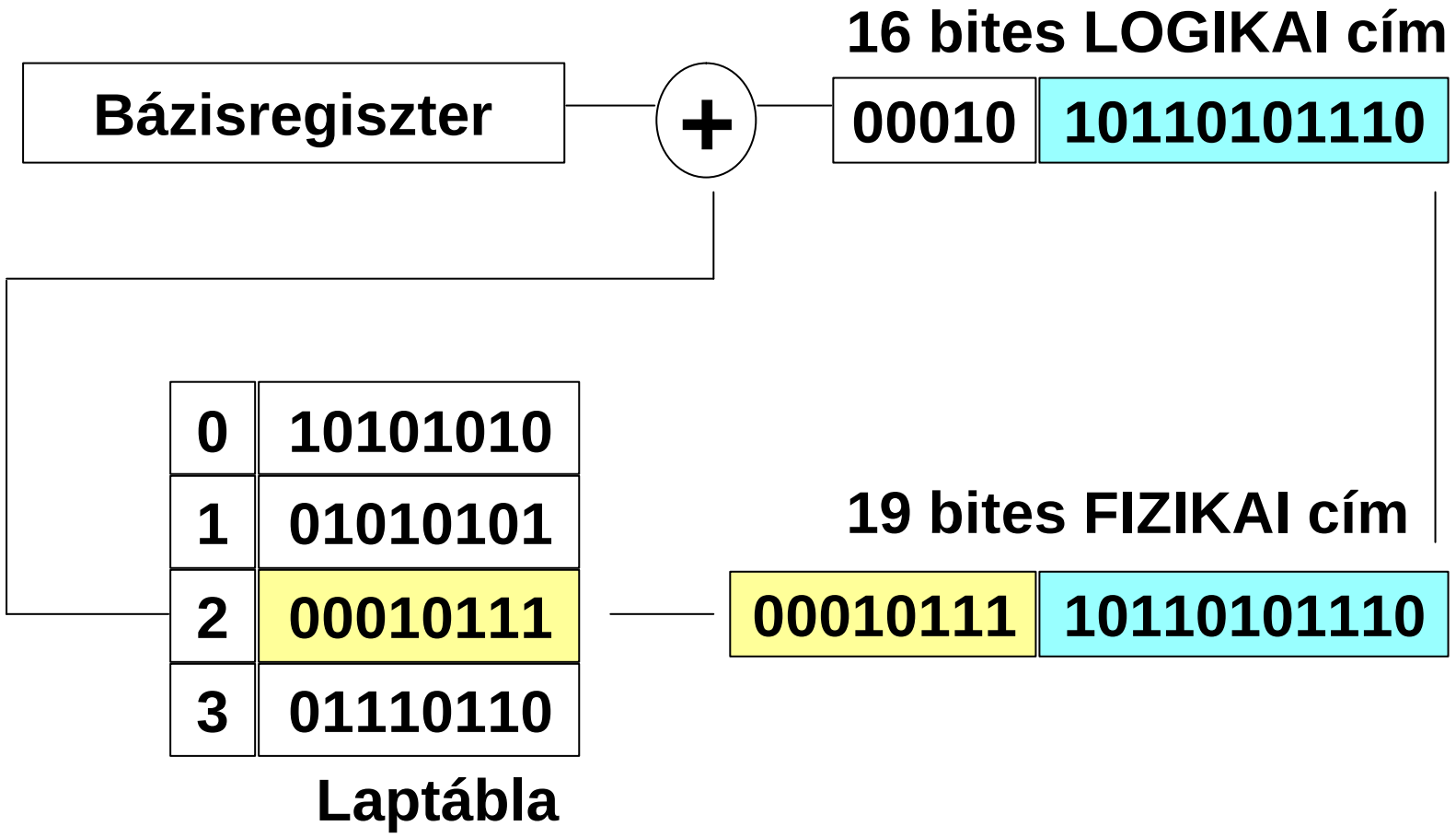
Lapozási technika



Fizika cím számítása lapozásnál



Memóriabovítés laptábla segítségével



Elozmények

Áthelyezheto: nem mindig ugyanoda töltodik

Átlapoló (overlay): nem minden rész kell egyszerre

Tárcsere (swapping): memóriakép a háttértáron

Lapozás: nem folytonos elhelyezés

Virtuális tárkezelés

A címtartomány háttértáron (virtuális tár, virtuális / logikai címtartomány)

Az operatív memória mérete sokkal kisebb a háttértárénál

A virtuális címtartomány csak egy kis része fér be az operatív tárba, ahonnan futtatni tudunk

általában nincs szükség minden programrészletre

Elonyök:

- ma már a mikroprocesszorok címtartománya is túl nagy ahhoz, hogy teljesen kiépítsük
- multiprogramozás során nem kell bent tárolni a teljes folyamatokat => több folyamat futhat párhuzamosan
- gyorsabb a betöltés és a felfüggesztés

Lapszervezésu virtuális tár

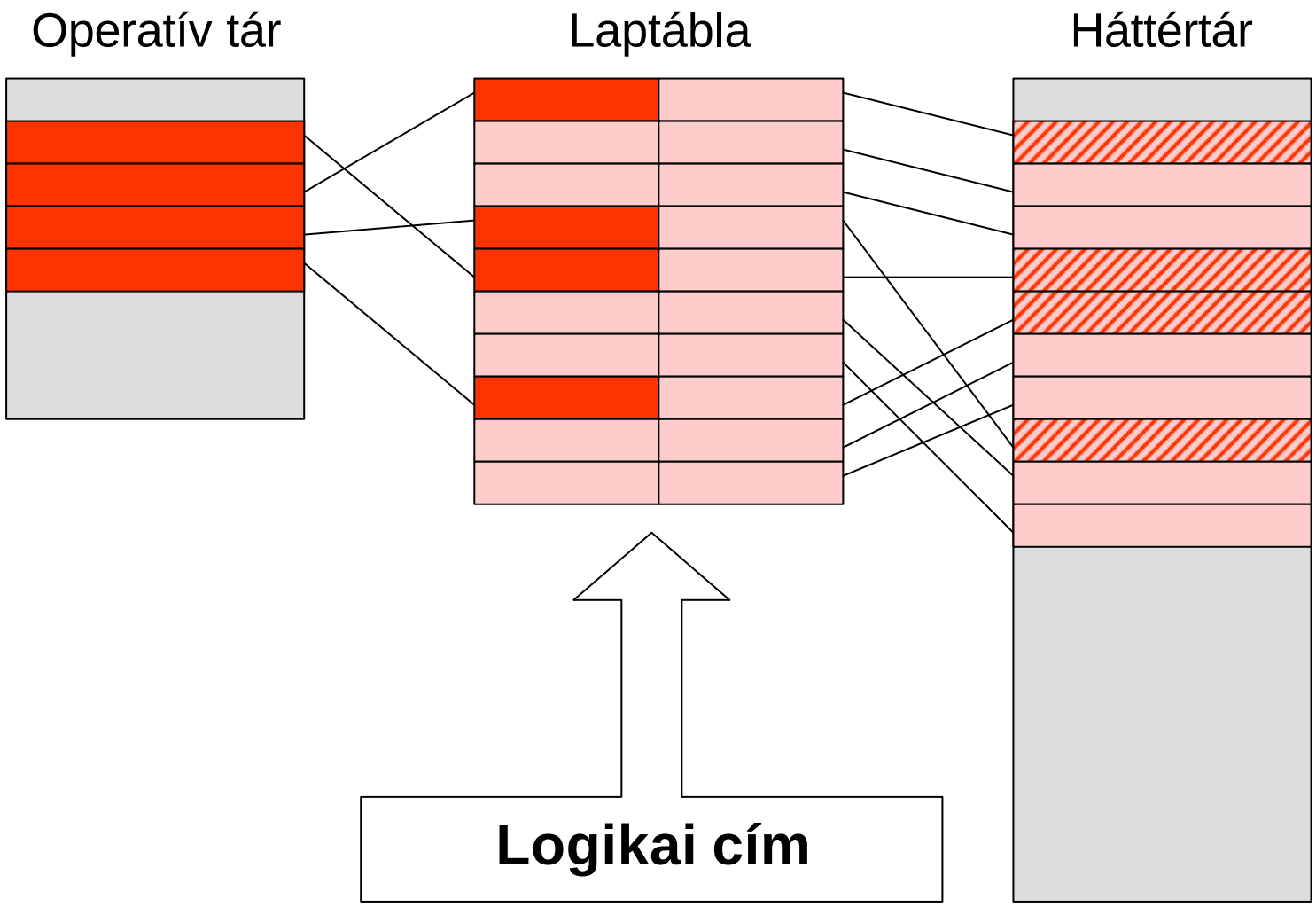
Felosztjuk a virtuális és az operatív tárat is egyforma méretu (kis, néhány kByte-os) egységekre, ún. LAPOKRA

Az operatív memóriában csak az éppen szükséges lapokat tartjuk bent

Laptábla tartja nyilván, hogy mely lapok vannak bent az operatív tárban

A processzor által kiadott (logikai) címet át kell alakítani operatív tárbeli (fizikai) címmé

Virtuális tárkezelés



Lényeg, elonyök

A folyamat lapjai közül nem mindegyik van a memóriában

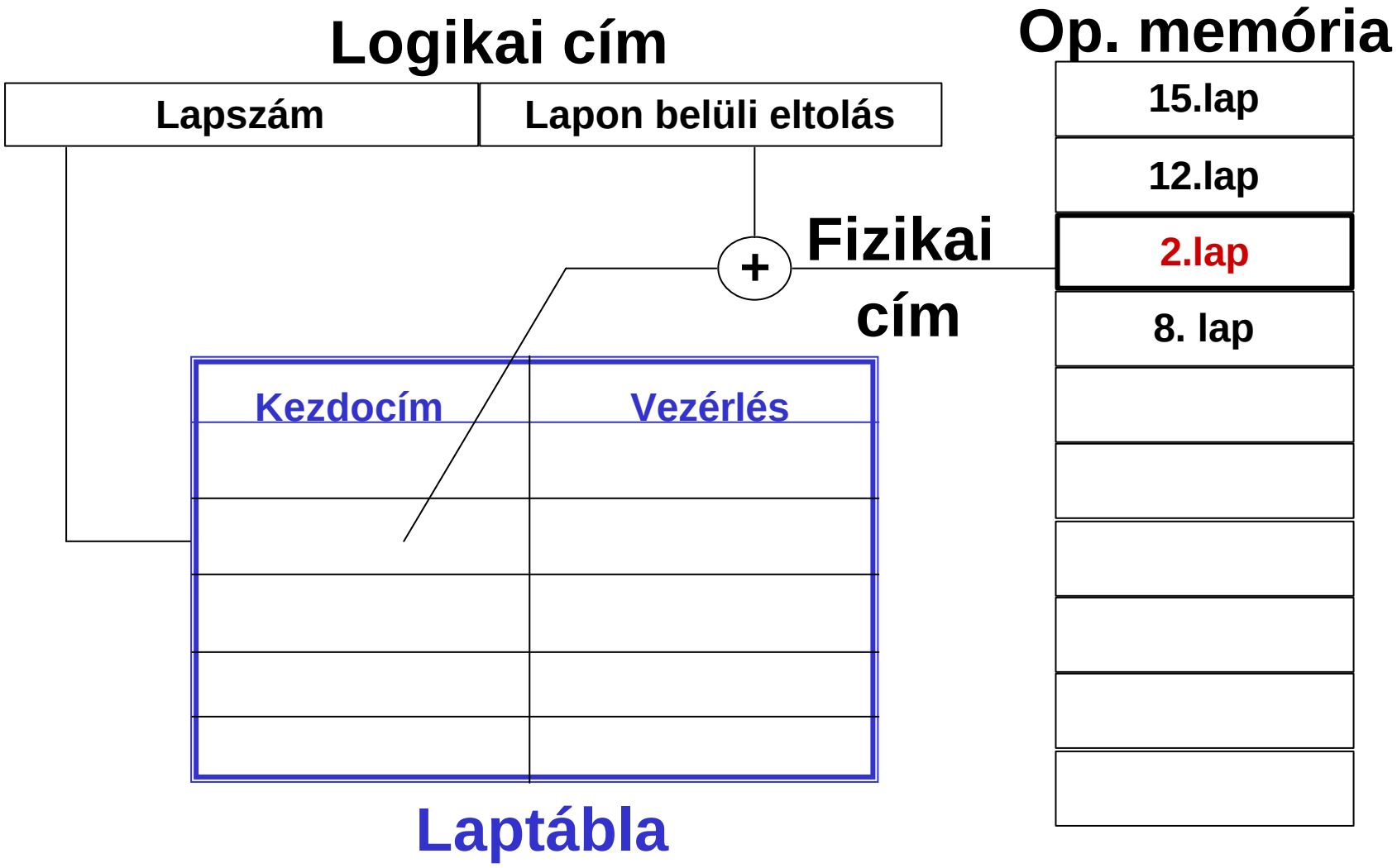
a többi a háttértáron várakozik

csak a folyamat által igényeltek kerülnek be (igény szerinti lapozás)

Elony:

- **a valós memóriánál nagyobb program**
- **több folyamat futhat egyszerre**
- **gyorsabb a betöltodés is**

Címszámítás lapszervezésu virtuális tár esetén



Laphiba (page fault)

Ha a folyamat olyan memóriacímre hivatkozik, ami olyan lapon található, amely nincs betöltve az operatív memóriába

Nem igazi hiba! Virtuális tárkezelésnél természetes!

Szükséges lépések:

- Ellenorizzük, hogy az adott folyamat használhatja-e az adott címet
- A lapot behozzuk a háttértárról az operatív tárba (természetesen elotte egy lapot lehet, hogy ki kell menteni) és módosítjuk a laptáblát
- Meg kell ismételni annak az utasításnak a végrehajtását, ahol a laphiba fellépett

A lapkezelés algoritmusa

A laptábla-vezérlés mezojének tartalma:

Bent van-e a lap az operatív memóriában (present)

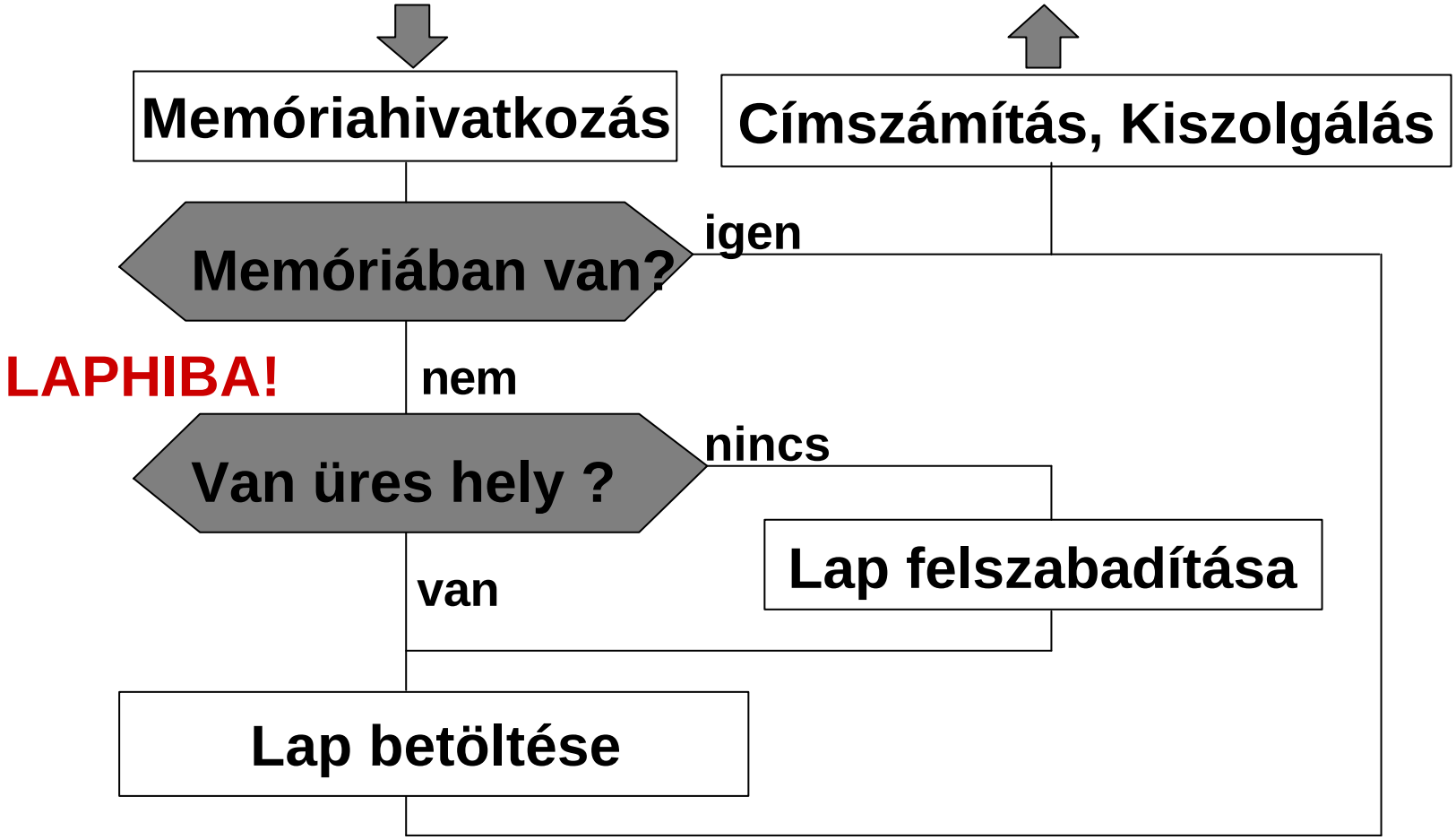
Módosítottuk-e (dirty)

– programlap / adatlap

Információ a lapcseréhez (pl. milyen régóta van bent az adott lap)

Védelmi információ

A virtuális memóriakezelés



Lapozási stratégiák
FIFO - Elobb be, elobb ki

A behozott lapok számait egy FIFO sorban tároljuk

Laphiba esetén a FIFO elején álló lapot cseréljük le

A behozott lap sorszámát a FIFO végére írjuk

Elony: **egyszeru**

Hátrány: **sok laphibát okoz**

FIFO - elso példa

Azt a lapot kell lecserélni, amely

- **legrégebben van a tárban**

Igényelt lap	6	8	3	8	6	0	3	6	3	5	3
1.lap	6	6	6			0		0		0	3
2.lap		8	8			8		6		6	6
3.lap			3			3		3		5	5
	*	*	*			*		*		*	*

Laphibák száma: 3 + 4

FIFO - második példa

5	**4**	**3**	**2**	**4**	**5**	**1**	**7**	**4**	**5**	**4**	**3**	**6**	**7**	**3**	**4**	**5**	**4**	**3**	**7**
5	**5**	**5**	**5**				**1**	**1**	**1**	**1**		**3**	**3**	**3**		**3**	**5**		**5**
-	**4**	**4**	**4**				**4**	**7**	**7**	**7**		**7**	**6**	**6**		**6**	**6**		**3**
-	**-**	**3**	**3**				**3**	**3**	**4**	**4**		**4**	**4**	**7**		**7**	**7**		**7**
-	**-**	**-**	**2**				**2**	**2**	**2**	**5**		**5**	**5**	**5**		**4**	**4**		**4**

FIFO:

5 4 3 2 1 7 4 5 3 6 7 4 5 3

Laphibák száma: 4 + 10

OPT - optimális stratégia

Az új lapot mindig annak a helyére hozzuk be, amelyre a legkésobb fogunk (újra)hivatkozni

Elony: **ez adja a minimális laphiba számot**

Hátrány: **a gyakorlatban megvalósíthatatlan, mert nem tudhatjuk elore a hivatkozások sorrendjét**

Megvalósíthatatlansága miatt csak az egyéb stratégiák jóságának vizsgálatához való referenciaként használják

OPT - elso példa

Azt a lapot kell lecserélni, melyre

- **a legkésobb lesz szükség**

Igényelt lap	6	8	3	8	6	0	3	6	3	5	3
1.lap	6	6	6			6				6	
2.lap		8	8			0				5	
3.lap			3			3				3	
	*	*	*			*				*	

Laphibák száma: 3 + 2

OPT - második példa

5	4	3	2	4	5	1	7	4	5	4	3	6	7	3	4	5	4	3	7
5	5	5	5			5	5					6				5			
-	4	4	4			4	4					4				4			
-	-	3	3			3	3					3				3			
-	-	-	2			1	7					7				7			

Laphibák száma: 4 + 4(!)

LRU - legrégebben használt

Az optimális stratégia (egyik) közelítése

Az új lapot mindig annak a helyére hozzuk be, amelyre a legrégebben hivatkoztunk (a lokalitási elv alapján erre lesz a legkevésbé valószínuleg szükség a későbbiek során)

Elony: **viszonylag jól közelíti az optimálist**

Hátrány: **kiegészíto HW-t igényel**

lassú

LRU - elso példa

Azt a lapot kell lecserélni, melyhez

- **a legrégebben fordult a folyamat**

Igényelt lap	6	8	3	8	6	0	3	6	3	5	3
1.lap	6	6	6			6	6			6	
2.lap		8	8			8	3			3	
3.lap			3			0	0			5	
	*	*	*			*	*			*	

Laphibák száma: 3 + 3

LRU - második példa

5	4	3	2	4	5	1	7	4	5	4	3	6	7	3	4	5	4	3	7
5	5	5	5				5	5					5	5	7				7
-	4	4	4				4	4					4	4	4				4
-	-	3	3				1	1					3	3	3				3
-	-	-	2				2	7					7	6	6				5

Laphibák száma: 4 + 6

Second Chance (SC) - második esély

Minden laphoz tartozik egy ”hivatkozás” bit (H) is
Ha a lapra hivatkozunk, ezt “1”-be állítjuk
(tehát behozáskor is!)

Lapcsere esetén azt a lapot vizsgáljuk, ami a FIFO elején van

- **ha H=0, ez lesz az áldozat**
- **ha H=1, nullázzuk a bitet, és a lap a sor végére áll, és újabb áldozatot keresünk**

Hátrány: HW-t igényel és elég bonyolult is

Elony: sokkal gyorsabb mint az LRU

SC - példa

5	4	3	2	4	5	1	1	1	1	1	7	3	4	4	5	5	5
5_1	5_1	5_1	5_1			5_0	5_0	5_0	5_0	1_1	1_1	1_1	1_1	1_1	1_0	1_0	1_0
-	4_1	4_1	4_1			4_1	4_0	4_0	4_0	4_0	7_1	7_1	7_1	7_1	7_1	7_0	7_1
-	-	3_1	3_1			3_1	3_1	3_0	3_0	3_0	3_0	3_1	3_0	3_0	3_0	3_0	5_1
-	-	-	2_1			2_1	2_1	2_1	2_0	2_0	2_0	2_0	2_0	4_1	4_1	4_1	4_1
*	*	*	*			!	!	!	!	*	*	!	!	*	!	!	*

Laphibák száma: 4 + 4

Not Used Recently (NUR) - Mostanában nem használt

Az LRU közelítése

Az operációs rendszer azok közül a lapok közül választja ki a lecserélendot, amelyek

az elozo lapcsere óta

NEM MÓDOSULTAK
és/vagy
NEM HIVATKOZTAK RÁJUK

A címszámítás gyorsítása

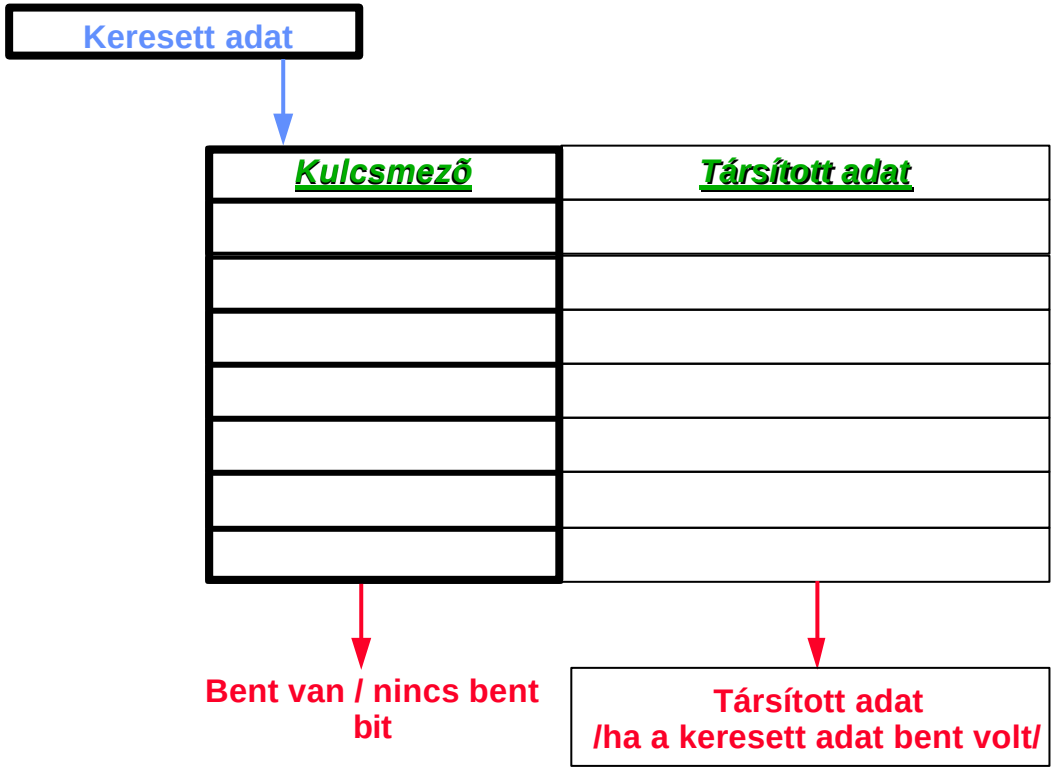
A virtuális tárkezelésnél a címszámítás sebessége több mint a felével csökken

Gyorsítás igénye

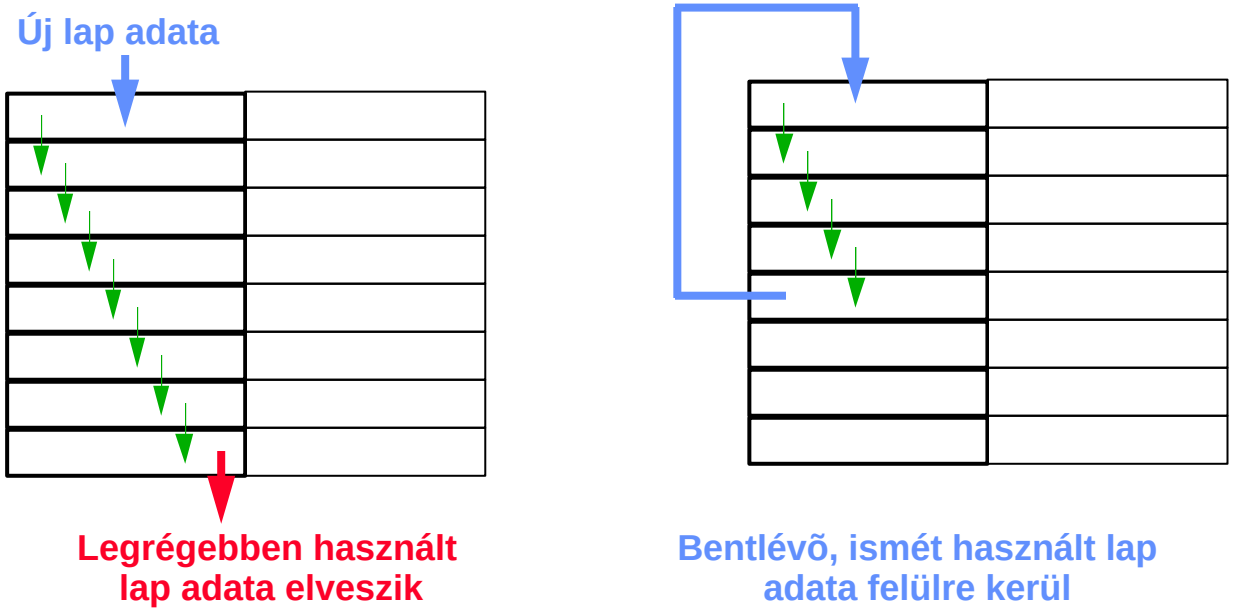
Az utoljára használt néhány lap címét egy speciális gyorstárban (TLB) tárolják (asszociatív memória)

A laptáblán keresztüli címszámítással párhuzamosan keresünk a TLB-ben is, ha ott megtaláljuk, leállítjuk a laptáblán keresztüli keresést

Asszociatív memória



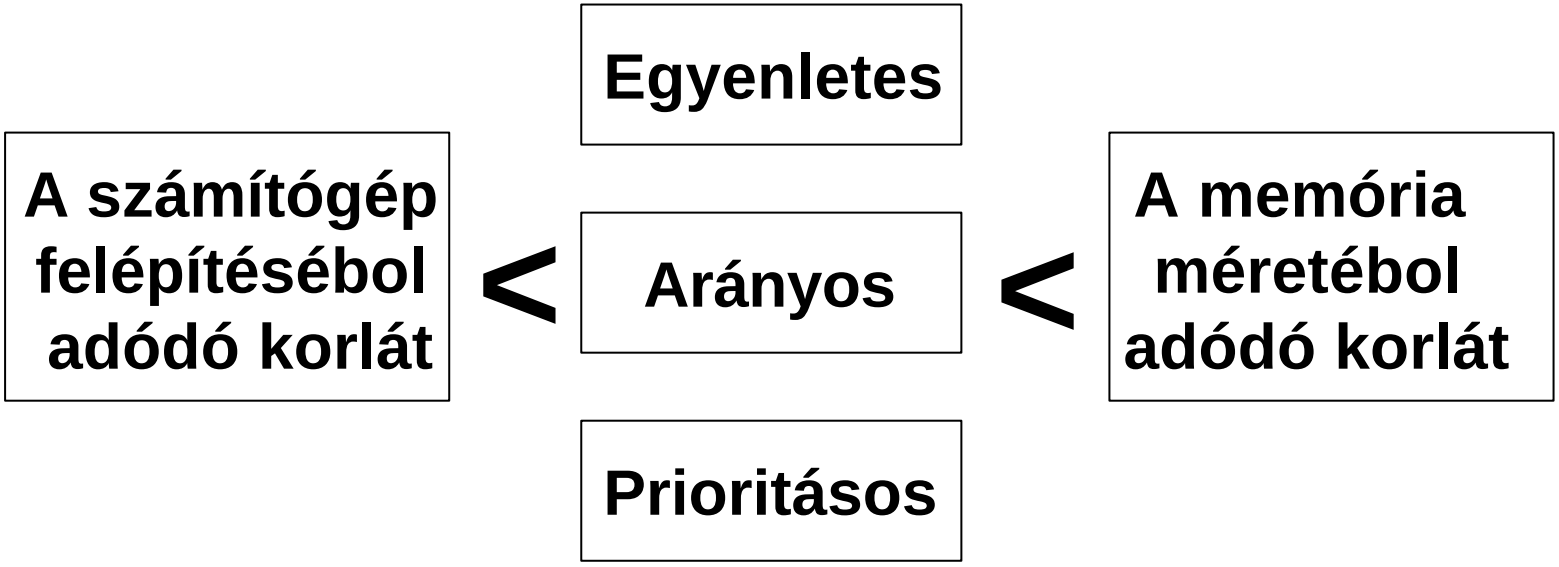
A TLB muködése



a.)

b.)

Keretek számának meghatározása



Lokális stratégia: A lapok száma futás alatt állandó
Globális stratégia: A lapok száma igény szerint változik

Vergodés

Egy folyamat több idot tölt lapozással, mint hasznos tevékenységgel (kevés lapja van)

Elkerülése:

Lokális lapkiosztási algoritmus: Minden folyamatnak rögzített darabszámú lapja van. Így, ha egy folyamat vergodik, a többi (majdnem) szabadon futhat

Munkahalmaz: viszonylag lassan változik; az op. r. olyan lapozási stratégiát követ, hogy minden folyamatnak igyekszik bent tartani a munkahalmazát; bonyolult SW és HW igény

Laphiba-gyakoriság figyelése: kevés laphibánál a folyamatnak túl sok kerete van, ezért elveszünk tole egyet, ill. sok laphibánál kevés kerete van, tehát adunk neki még egyet

Védelem

Szegmentálás: bontsuk a programjainkat LOGIKAI egységekre (lapozás: fizikai egységekre tördelés)

Ezzel megoldható

1. **Egy folyamaton belül a logikai egységek védelme egymástól**
2. **A folyamatok védelme egymástól (DE biztosítva az esetleges együttmuködés lehetoségét)**
3. **Az operációs rendszer védelme a felhasználóktól**

Logikai egységek védelme

Különítsük el a folyamat LOGIKAI egységeit!

- kód-, adat-, veremszegmens

Minden utasításnak legyen alapértelmezett szegmense!

- JUMP kódszegmens;
- MOVE adatszegmens;
- PUSH stack szegmens,

Az utasítás végrehajtásakor a HW ellenorizze, hogy tényleg adott típusú szegmensben van-e az operandus.

Tárolni kell a szegmens típusát, kezdocímét és HOSSZÁT

(Az univerzalitás ellen hat: “prefix utasítások”)

Folyamatok védelme egymástól

Lokális leíró tábla (LDT)

Legyen minden egyes folyamatnak SAJÁT szegmens leíró táblája, amelyben CSAK az általa elérheto szegmensek vannak feltüntetve, így a memória MÁS területeit egyáltalán nem használhatja az adott folyamat!

Globális/Interrupt leíró tábla (GDT, IDT)

A közösen használt szegmensek leíróit helyezzük el az összes együttmuködni szándékozó folyamat leíró táblájában, DE esetleg különbözo jogokkal!

Az operációs rendszer védelme

Prioritási szintek bevezetése:

Egy adott szegmenst csak az a folyamat használhat, melynek prioritása minimum megegyezik a szegmens prioritásával

- szegmensek prioritási szintje: a vezérlés mezoben
- folyamat prioritási szintje: a KÓDSZEGMENSÉHEZ rendelt prioritási érték

(Természetesen az operációs rendszer folyamatai kerülnek a magas prioritású szintekre, míg a felhasználói folyamatok az alacsonyakra)

Tárvédelem

Folyamatok logikai egységeinek védelme egymástól	**Szegmensleíró tábla, szegmenshossz mezo**
Folyamatok védelme egymástól	**Szegmensleíró tábla létezése**
Operációs rendszer védelme	**Szegmensleíró tábla védelmi (prioritás) mezoi**

Lapozás vs. Szegmentálás

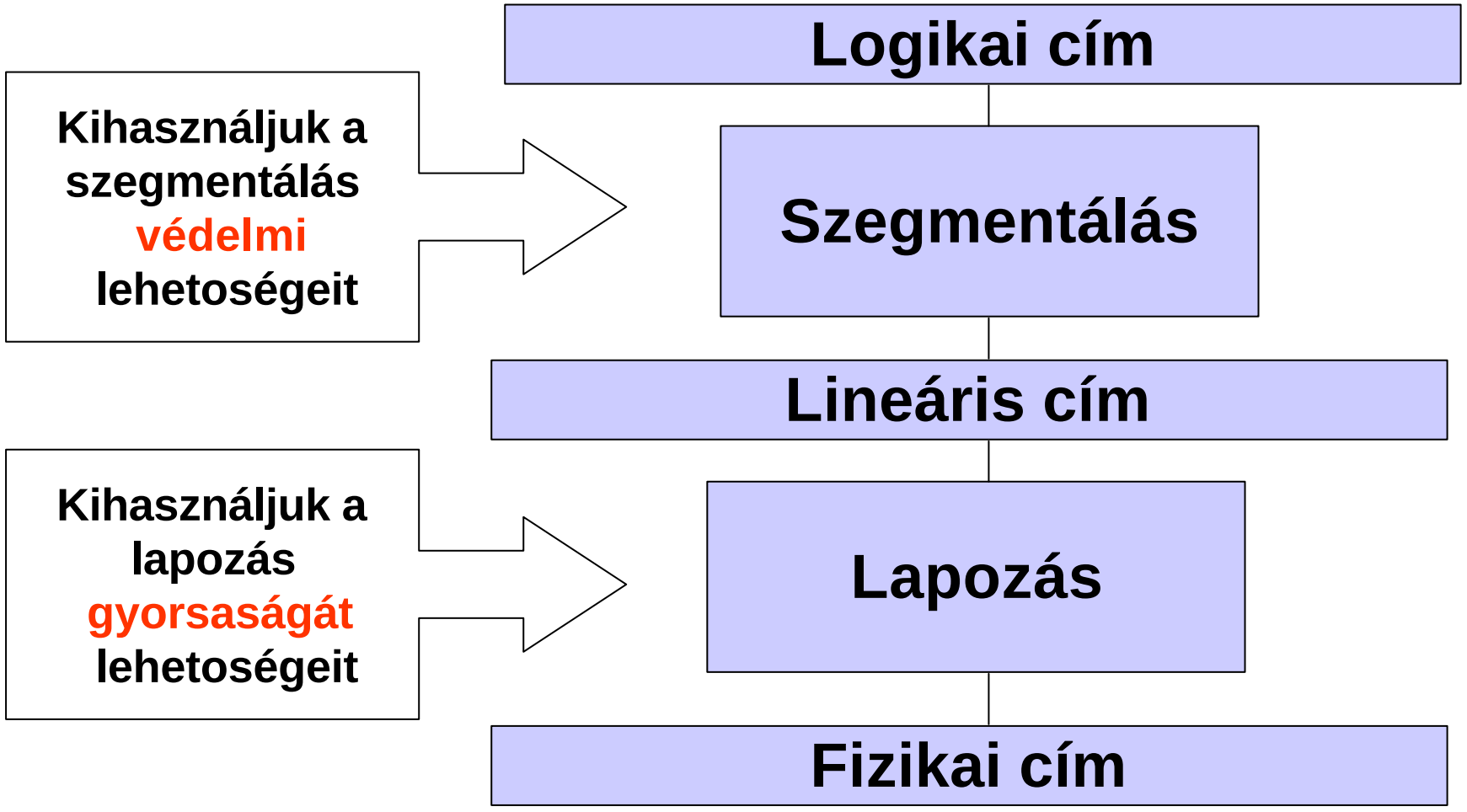
Lapozás:

- Fizikai darabolás eredménye
- Egyforma méretu lapok
- **Gyors, hardver által végezheto lapcsere**
- **A védelemnek nem sok értelme van**

Szegmentálás

- Logikai darabolás eredménye
- Különbözo méretu szegmensek
- **Lassabb betöltés/mentés**
- **Védelemhez jól használható**

Lapozott szegmentálás



Lapozott szegmentálás - példa

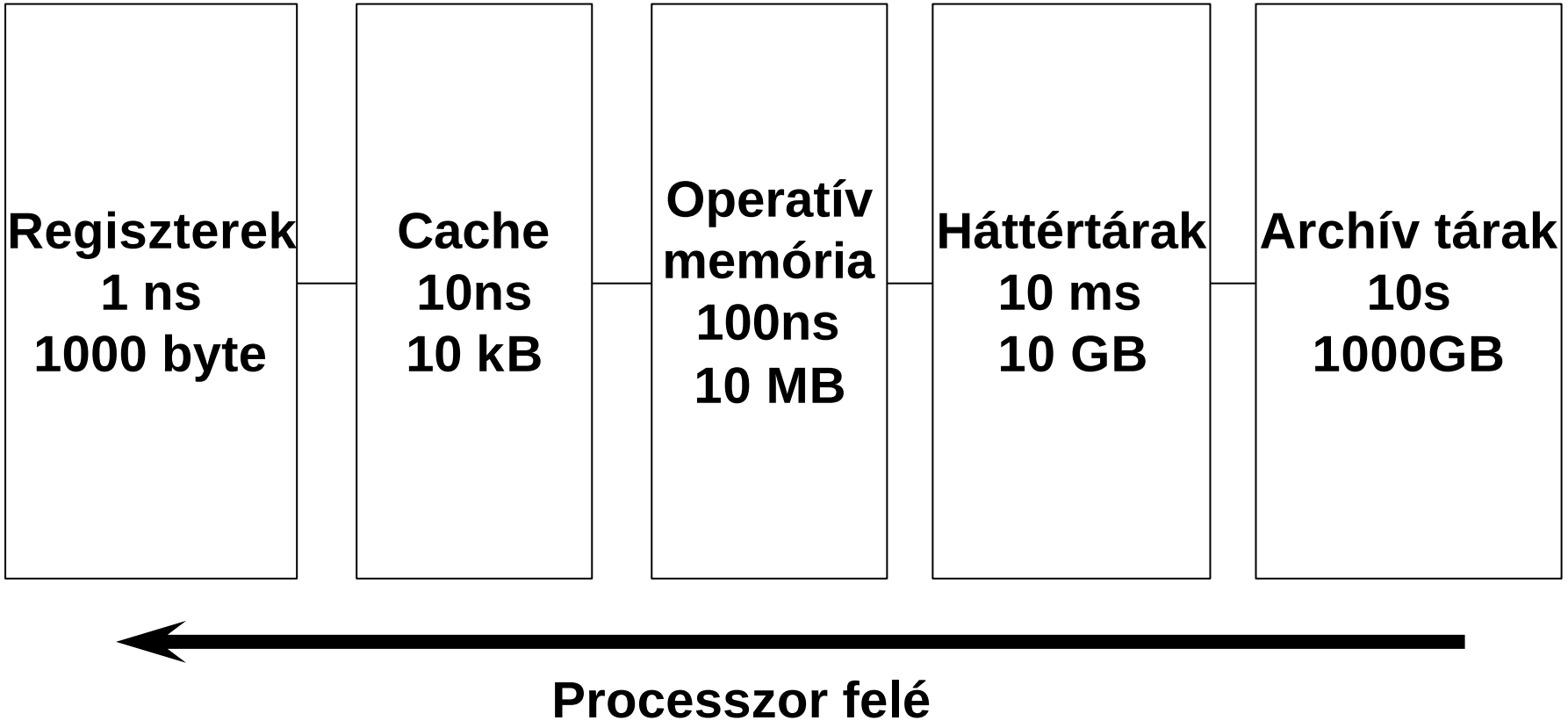
pl. JMP *valahova* (szegmens: CS, offset: *valahova*

- **A folyamat által adott cím a LOGIKAI cím**
- **A szegmensleíró tábla CS-hez tartozó sorában ellenorizzük, hogy a folyamat jogosult-e használni a szegmenst, illetve, hogy a valahova cím benne van-e a tartományban.**
- **Kiszámítjuk a szegmens:offset alapján a LINEÁRIS címet (pl. szegmens kezdocím+offset)**
- **A lineáris cím a lapozó egység bemenete, felosztódik laptábla címre (pl. 6 bit -> 4kB) és lapon belüli offset-re, majd a laptábla által tartalmazott fizikai lapcímet összeillesztve az eltolással, megkapjuk a FIZIKAI címet.**

Gyorsítótár (cache) - elv

A leggyakrabban használt adatok legyenek a leghamarabb elérheto helyen

Tároló hierarchia



Összefoglalás

Valóságos tárkezelés

- Overlay, swapping, particionálás, lapozás

Virtuális tárkezelés

- Laphiba, lapkezelés, címszámítás
- Lapcsere stratégiák: FIFO, OPT, LRU, SC, NUR
- A címszámítás gyorsítása

Tárvédelem

- A logikai egységek védelme
- A folyamatok védelme egymástól
- Az operációs rendszer védelme

Párhuzamos programozás

Párhuzamosság, szinkronizálás, konkurencia

Precedenciagráf, leíró szerkezetek

- **Fork-join**
- **Parbegin-parend szerkezet**
- **Szemaforok**

Párhuzamosság, szinkronizálás, konkurencia

u1: **x:=12;**

u2: **y:=x/4;**

u3: **a:=x+y;**

u4: **b:=x-y;**

u5: **c:=a*b;**

u6: **d:=x+1;**

u7: **e:=c+d;**

Párhuzamosság, szinkronizálás, konkurencia

u1: **x:=12;**

u2: **y:=x/4;**

u3: **a:=x+y;**

u4: **b:=x-y;**

u5: **c:=a*b;**

u6: **d:=x+1;**

u7: **e:=c+d;**

Bizonyos utasítások csak egymás után (szekvenciálisan) hajthatók végre, pl. u1 és u2, hiszen u2 használja u1 eredményét

Párhuzamosság, szinkronizálás, konkurencia

```
u1:   x:=12;
u2:   y:=x/4;
u3:   a:=x+y;
u4:   b:=x-y;
u5:   c:=a*b;
u6:   d:=x+1;
u7:   e:=c+d;
```

Bizonyos utasítások csak egymás után (szekvenciálisan) hajthatók végre, pl. u1 és u2, hiszen u2 használja u1 eredményét

DE: pl. u3 és u4 egyszerre, egymással párhuzamosan is végrehajtható, hiszen nem függenek egymástól

Precedenciagráf

Csomópontok: UTASÍTÁSOK

Élek: Abban a csomópontban lévo utasítás, ahonnan egy él indul, MEG KELL, HOGY ELOZZE azt az utasítást, amely abban a csomópontban található, ahová az él mutat

Szerkesszük meg a precedenciagráfot!

u1: x:=12;

u2: y:=x/4;

u3: a:=x+y;

u4: b:=x-y;

u5: c:=a*b;

u6: d:=x+1;

u7: e:=c+d;

Precedenciagráf

ha több él találkozik egy csomópontban: szinkronizálás!

u1: x:=12;

u2: y:=x/4;

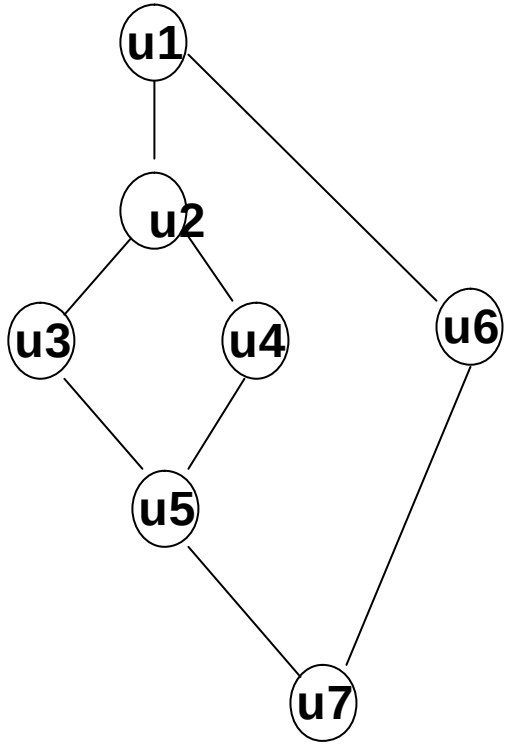
u3: a:=x+y;

u4: b:=x-y;

u5: c:=a*b;

u6: d:=x+1;

u7: e:=c+d;



A precedenciagráfok szemléletesen tükrözik a párhuzamosítási lehetoségeket, de programozásra közvetlenül nem használhatók

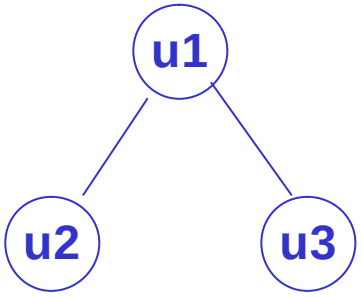
1. **megoldás: fork / join utasításpár**
2. **megoldás: parbegin/parend utasításpár**

Fork / join utasításpár

fork utasítás: a végrehajtás két egymással párhuzamosan végrehajtható ágra szakad, az egyik ág közvetlenül a fork utasítás után, a másik ág a megadott címkénél található

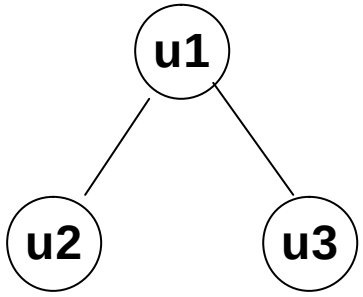
Fork / join utasításpár

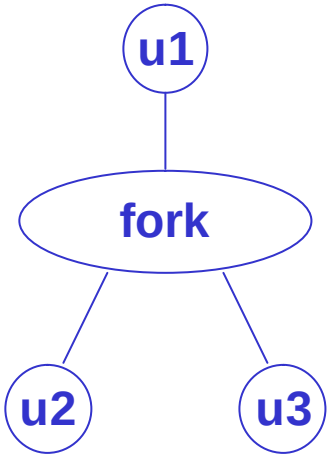
fork utasítás: a végrehajtás két egymással párhuzamosan végrehajtható ágra szakad, az egyik ág közvetlenül a fork utasítás után, a másik ág a megadott címkénél található



Fork / join utasításpár

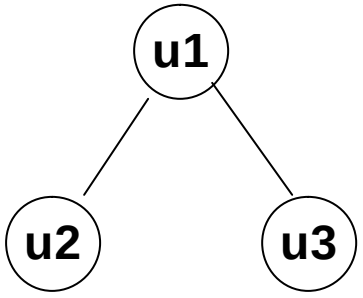
fork utasítás: a végrehajtás két egymással párhuzamosan végrehajtható ágra szakad, az egyik ág közvetlenül a fork utasítás után, a másik ág a megadott címkénél található

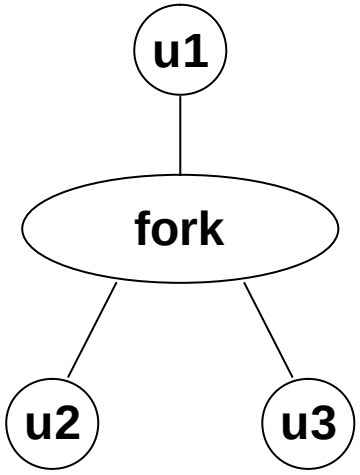




Fork / join utasításpár

fork utasítás: a végrehajtás két egymással párhuzamosan végrehajtható ágra szakad, az egyik ág közvetlenül a fork utasítás után, a másik ág a megadott címkénél található






```
        u1;
        fork   L;
        u2;
        ...
L:      u3;
```

Fork / join utasításpár

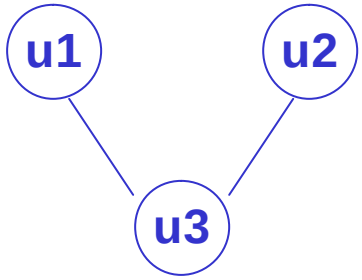
join utasítás: két vagy több ág egyesítése;

- az ágak “bevárják egymást” (szinkronizáció) és csak a legutoljára “érkezo” folytatódik, a többi “meghal”
- az egyesítendo ágak számát egy számláló mutatja

Fork / join utasításpár

join utasítás: két vagy több ág egyesítése;

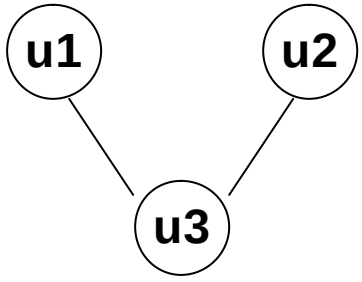
- az ágak “bevárják egymást” (szinkronizáció) és csak a legutoljára “érkezo” folytatódik, a többi “meghal”
- az egyesítendo ágak számát egy számláló mutatja

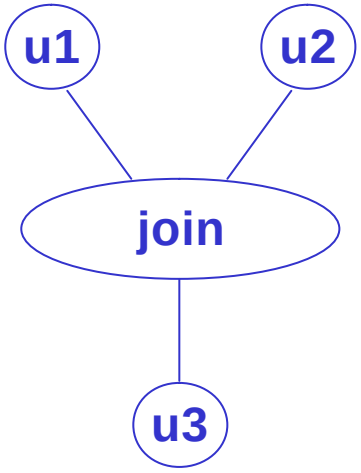


Fork / join utasításpár

join utasítás: két vagy több ág egyesítése;

- az ágak “bevárják egymást” (szinkronizáció) és csak a legutoljára “érkezo” folytatódik, a többi “meghal”
- az egyesítendo ágak számát egy számláló mutatja

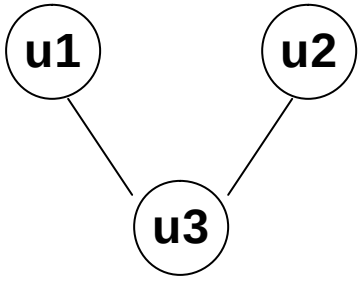


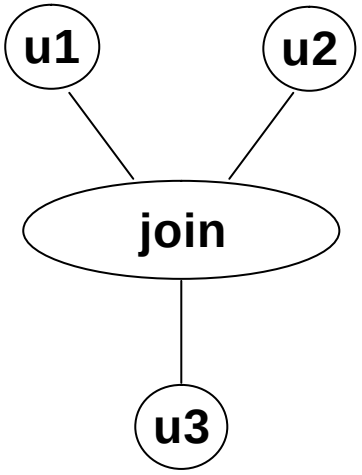


Fork / join utasításpár

join utasítás: két vagy több ág egyesítése;

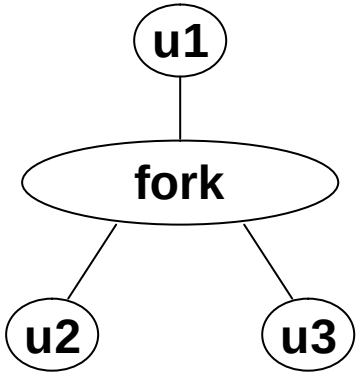
- az ágak “bevárják egymást” (szinkronizáció) és csak a legutoljára “érkezo” folytatódik, a többi “meghal”
- az egyesítendo ágak számát egy számláló mutatja



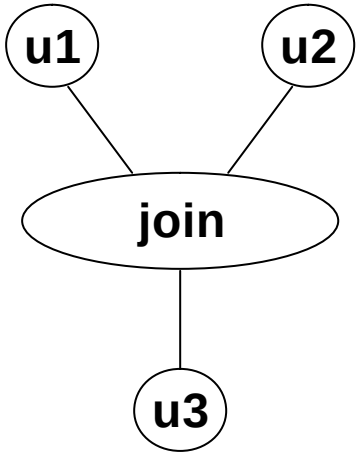



```
    száml:=2;
    u1;
    goto L;
    u2;
L: join száml;
    u3;
```

Fork / join utasításpár

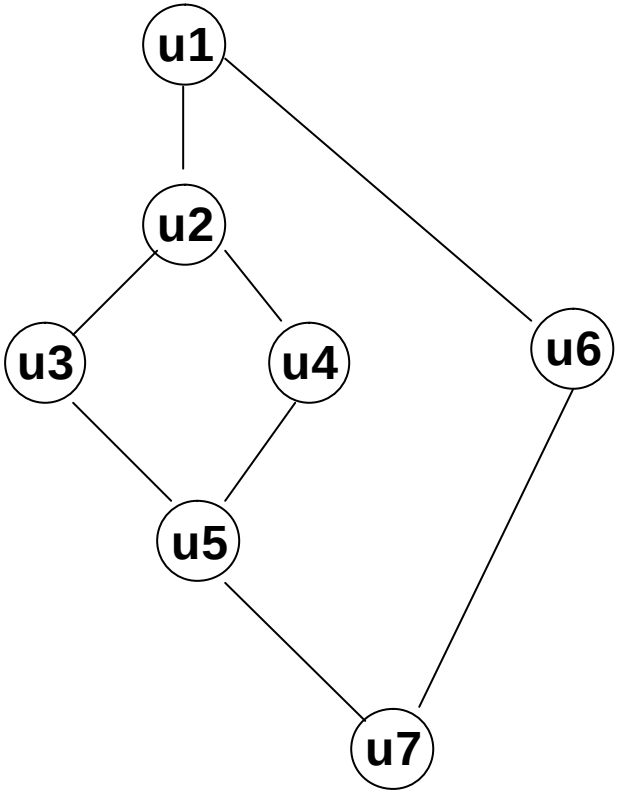


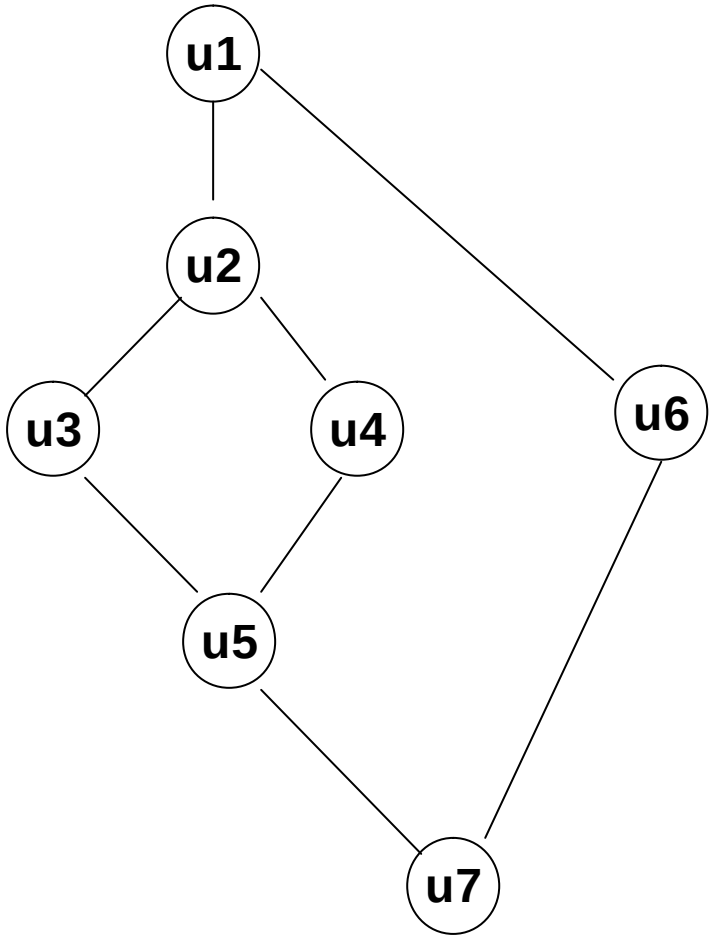

```
        u1;
        fork   L;
        u2;
        ...
L:      u3;
```




```
    száml:=2;
    u1;
    goto L;
    u2;
L: join száml;
    u3;
```

Példa a fork/join utasítás használatára






```
        száml2 := 2;
        száml1 := 2;
        u1;
        fork    L1;
        u2;
        fork    L2;
        u3;
        goto    L3;
L2: u4;
L3: join száml1;
        u5;
        goto    L4;
L1: u6;
L4: join száml2;
        u7;
```

Parbegin / parend utasításpár

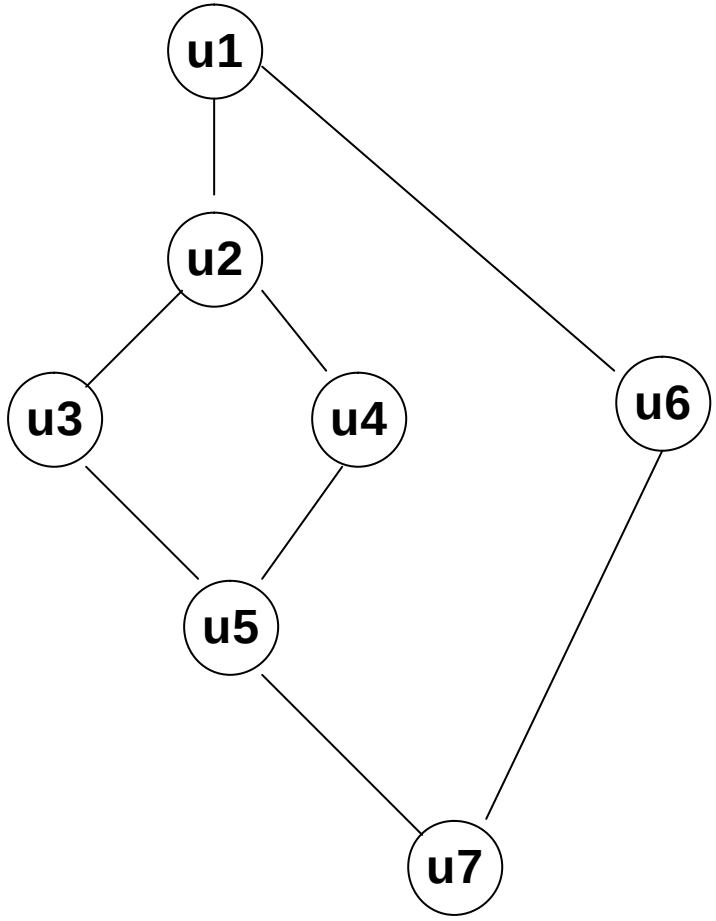
A fork / join utasítások használatával áttekinthetetlen programstruktúra jön létre, a sok “goto” utasítás miatt

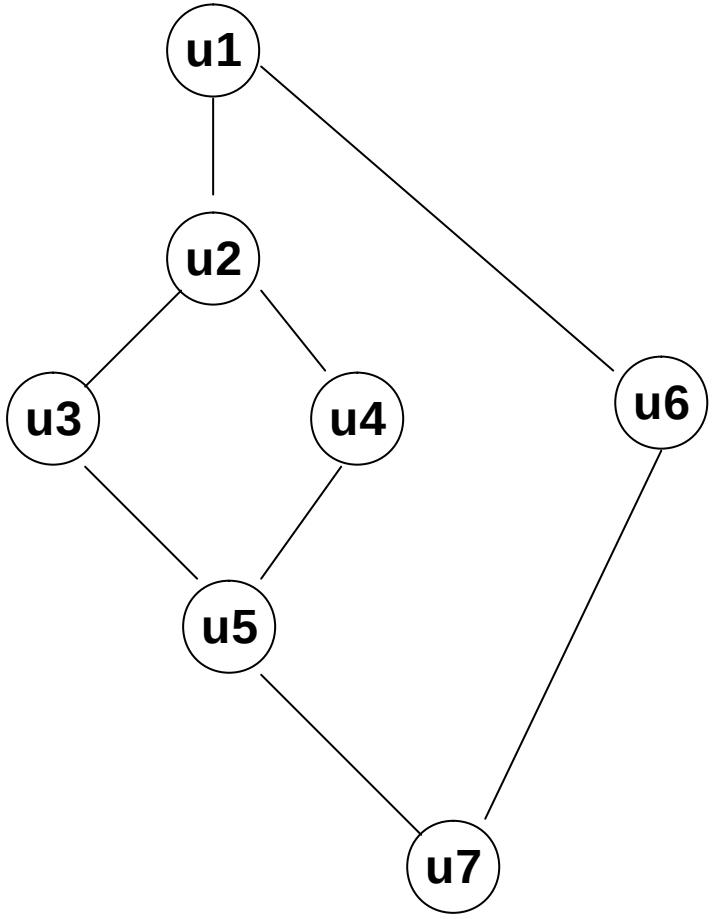
Megoldás: parbegin / parend utasítások

parbegin u1; u2; .... parend;

Jelentése: u1, u2, ... egymással párhuzamosan végrehajtható utasítások

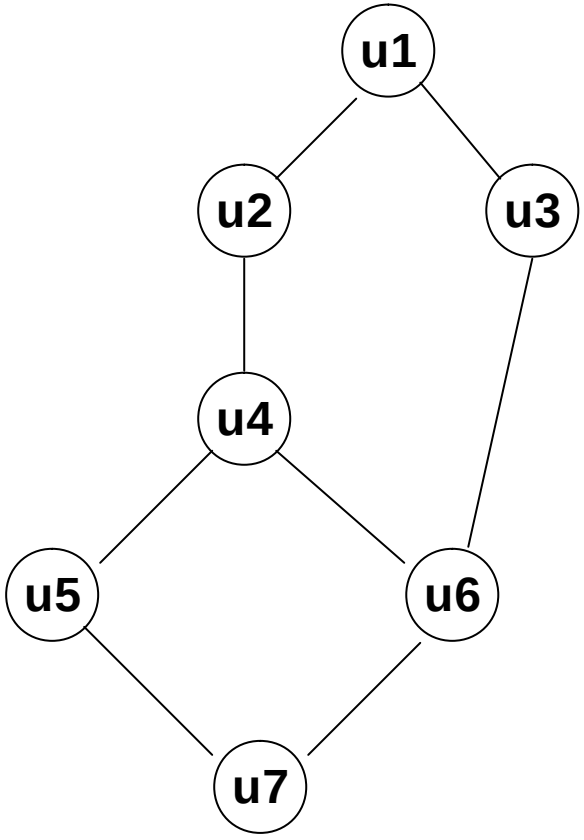
Példa a parbegin/parend utasítás használatára






```
u1;
parbegin
    u6;
    begin
        u2;
        parbegin
            u3;
            u4;
        parend;
        u5;
    end;
parend;
u7;
```

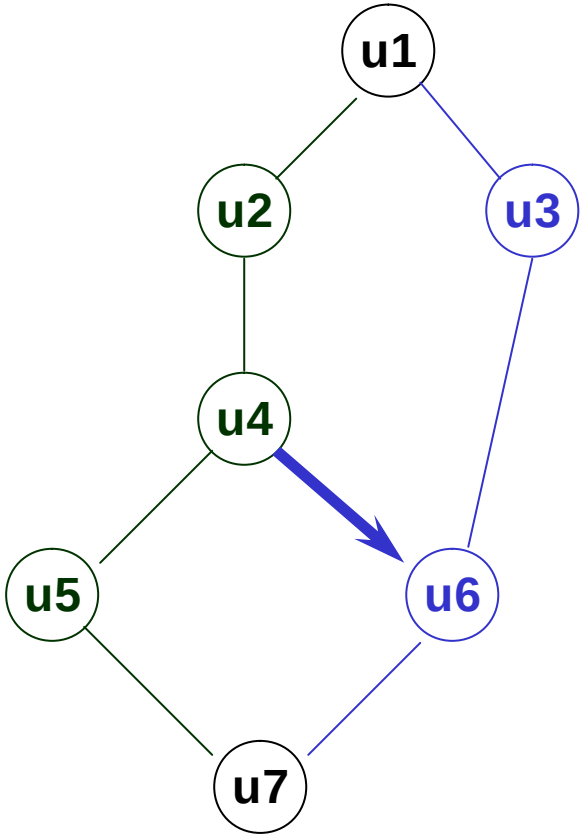
Parbegin / parend utasításpár



A parbegin/parend utasításpár nem univerzális

Ez a precedenciagráf nem írható le segítségükkel

Parbegin / parend utasításpár



A parbegin/parend utasításpár nem univerzális

Ez a precedenciagráf nem írható le segítségükkel

A precedenciagráf, a fork/join és a parbegin/parend összehasonlítása

- **A legszemléletesebb a precedenciagráf, de programozásra nem alkalmas**
- **A fork/join segítségével minden leírható, ami precedenciagráffal, de áttekinthetetlen programot eredményez**
- **A parbegin/parend áttekintheto programot ad, de “kevesebbet tud” mint a precedenciagráf és a fork/join. Ahhoz, hogy egyenrangú legyen velük, még további támogatás kell (szemaforok bevezetése)**

A nagy adminisztrációs igény miatt a párhuzamosítás általában nem utasításszintu, hanem folyamat szintu

Folyamatok dinamikus létrehozása

- **szülo / gyerek folyamatok**

Végrehajtás:

- **A szülo a gyerekkel párhuzamosan fut**
- **A szülo felfüggesztodik és megvárja minden gyerekének befejezodését, csak utána fut tovább**

Eroforrás használat:

- **A közös változókat osztottan használják**
- **A gyerek a szülo változóinak csak egy részét kapja (UNIX: változókat nem, csak az állomány hozzáférési jogokat; kommunikáció speciális egységen (pipe))**

Folyamatok törlése

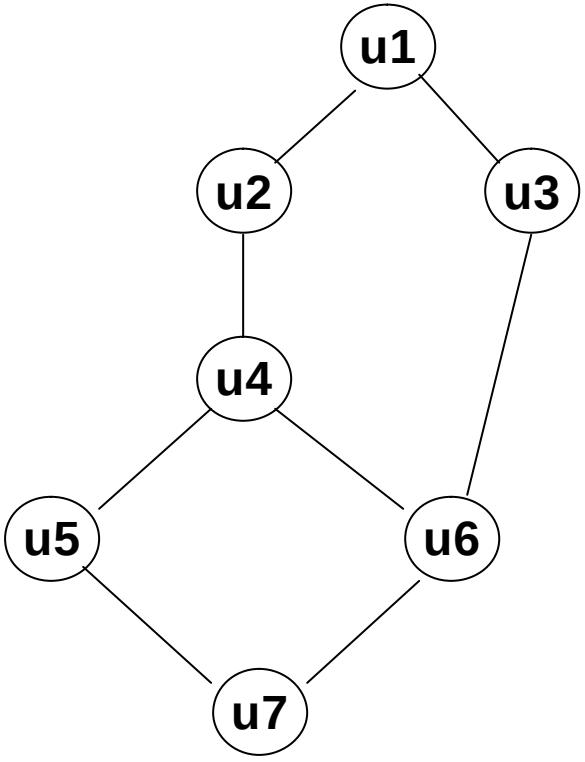
Folyamatok törlése: “KILL foly.azonosító” utasítással

- folyamat létrehozásakor egyedi azonosítót kell rendelnünk hozzá

Általában csak a szülo törölheti a gyerekét

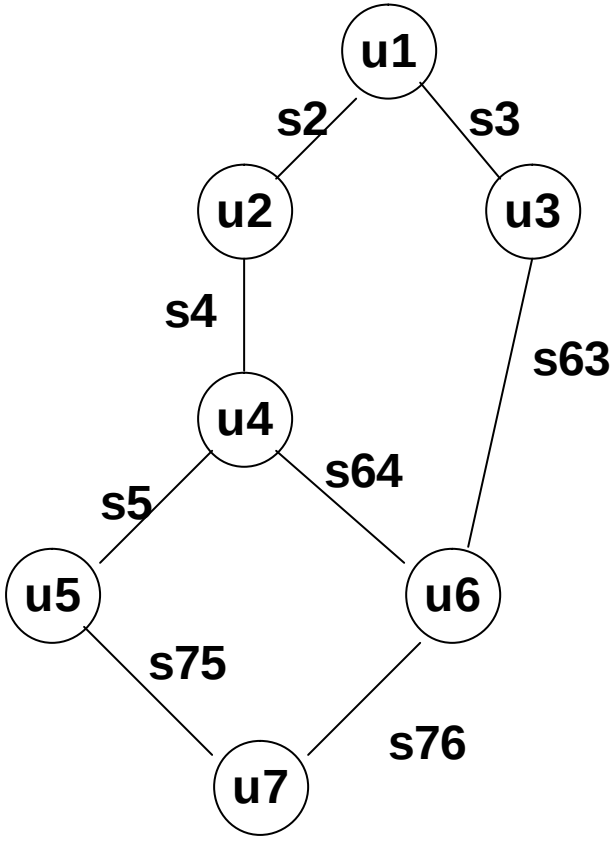
Általában egy folyamat törlése maga után vonja összes gyerekének törlését is

A parbegin / parend általánosítása szemaforokkal



- Tisztán parbegin/parend utasításokkal nem írható le
- Rendeljünk minden élhez egy-egy átmenetet engedélyezo szemafort
- E szemaforok kezdeti értéke “foglalt” kell legyen

A parbegin / parend általánosítása szemaforokkal




```
parbegin
    begin u1; V(s2); V(s3); end;
    begin P(s2); u2; V(s4); end;
    begin P(s3); u3; V(s63); end;
    begin P(s4); u4; V(s5); V(s64); end;
    begin P(s5); u5; V(s75); end;
    begin P(s64); P(s63); u6; V(s76);
      end;
    begin P(s75); P(s76); u7; end;
parend;
```

A parbegin / parend általánosítása szemaforokkal

A szemaforokkal kiegészített megoldás lassabb, mint a tisztán parbegin/parend utasításokat tartalmazó megoldás, ezért csak akkor használjuk, ha feltétlen szükséges!

Összefoglalás

Párhuzamos folyamatok szinkronizálása

Precedenciagráf, leíró szerkezetek

A fork-join utasításpár - elonyök, hátrányok

A parbegin-parend szerkezet - elonyök, hátrányok

A parbegin-parend szerkezet általánosítása