

Hardver alapok

(VIII BA01)

Virtuális tárkezelés

Pilászy György gpilaszy@iit.bme.hu

2018/19. tanév őszi félév

Alapvető megoldandó problémák:

Kicsi a logikailag megcímezhető tárkapacitás

- A teljes program, vagy adatállomány nem fér el a „címtérben”

→ *Bővíteni kell a fizikailag megcímezhető memóriát*

Nem lehet, vagy nem célszerű a megcímezhető tárat teljes egészében realizálni

- Adott program, vagy adatállomány nem fér el a fizikailag megvalósított memóriában
- Adott programnak futnia kell különböző fizikai memóriával rendelkező gépeken is

→ *Virtuális tárkezelés*

Támogatást kell nyújtani az operációs rendszernek a tárkezeléshez és a védelem megoldásához

Néhány fogalom...

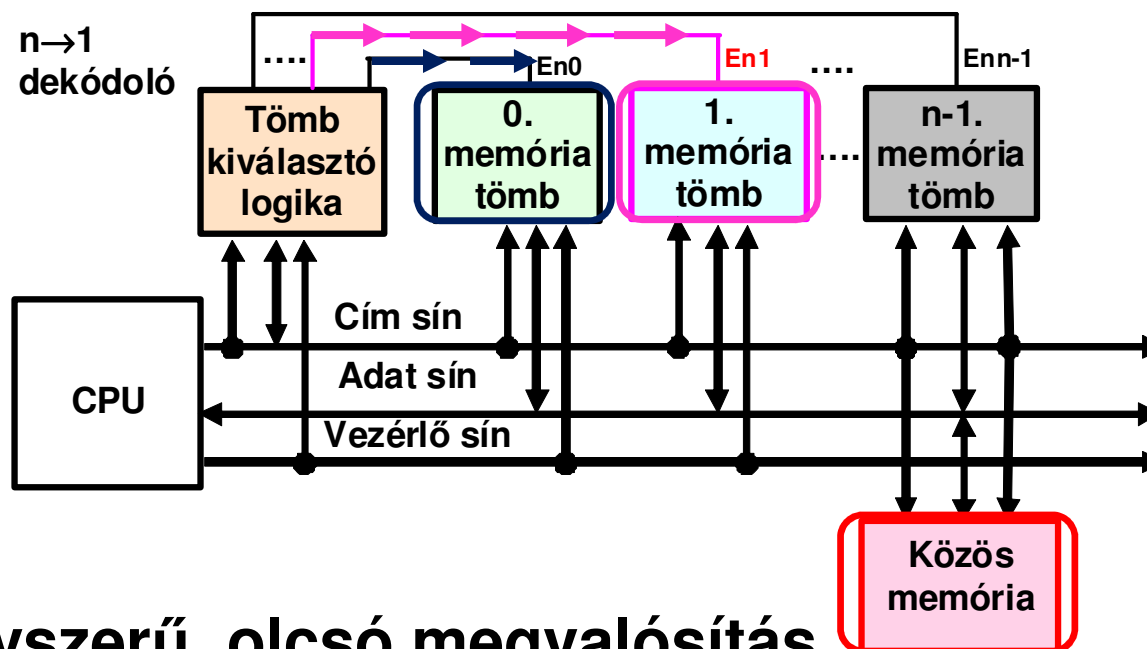
Fizikai címtartomány: a processzor címvezetékeivel megcímezhető memóriatartomány vagy a rendszerben ténylegesen beépített memória tartomány (ez lehet kisebb, mint a maximálisan megcímezhető)

Logikai címtartomány: a processzor által támogatott címszámítási mechanizmus során keletkező címtartomány (ez lehet nagyobb, mint a fizikailag címezhető tartomány)

Kis logikai és fizikai címkapacitás

Tömbkapcsolásos memóriabővítés

Egy I/Oként működő dekódoló n-ből egy tömböt engedélyez



Előnyök:

- Nagyon egyszerű, olcsó megvalósítás
- Gyors (az adott tömb OUT utasítással átkapcsolható)

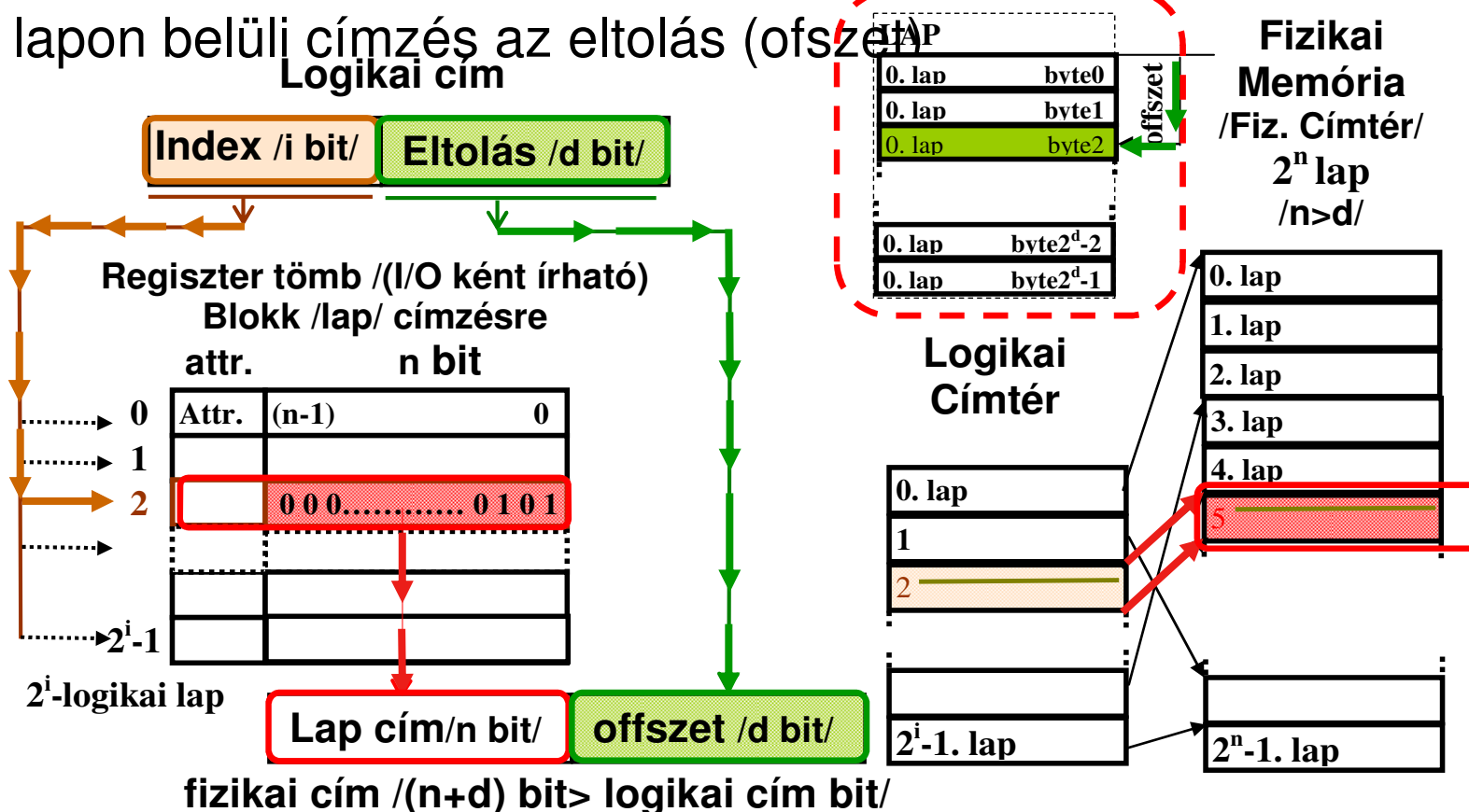
Hátrányok:

- Durva, merev memória felosztás
- Átkapcsoláshoz, IT kezeléshez használt rész mindig aktív

Kis logikai címkapacitás

Indexelt leképezés (Külső regisztertömb alkalmazásával)

- Logikai cím fizikai cím összerendelés egy regiszter tartalmával
- I/O-ként írható regisztertömb
- Rögzített méretű blokkok (lapok) → *egyszerre 2^i lap aktív*
- A lapon belüli címzés az eltolás (offset)



- Előnyök:
 - Egyszerű, gyors logikai $\rightarrow$ fizikai címátalakítás
 - Rugalmas (*programok a tárolóban szabadon, laponként mozgathatók*)
- Hátrányok:
 - Bonyolultabb, mint a tömbkapcsoló
 - Közösen használt részeknek (*pl. I/O, IT-k*) mindig aktívnak kell lennie
 - (*pl.: 0. regiszter helyett konstans 0 cím*)

Bázisregiszter alkalmazása

Fizikai cím = (bázisregiszter tartalom + logikai cím)

Egyszerű, ha a CPU tartalmaz ilyen speciális regisztert

Kevés regiszter esetén durva felosztás

Felhasználói szintű Overlay szervezés

- *A felhasználói program háttértárolóról egy megadott memóriaszegmensre rendre új programrészletet, vagy adatot tölt*
Nehézkes, a hiba elleni védelem nehezen megoldható

Esettanulmány

- *i8086/88-nál a CPU tartalmaz bázisregisztereket (szegmensregiszterek)*
- *átmenet a logikai cím és a fizikai tárbővítés között címtartomány 1MB*
8086/88 PC-nél gyakran alkalmazott megoldás
 - *a memória bővítésére - a tömbkapcsolás és az overlay együttes alkalmazása, **védelem!***

Virtuális tárkezelés

A processzor teljes címtartományát (logikai cím) kiterjeszti a háttértárolóra is → virtuális címtér

- *Megoldandó feladatok*

- ☐ Minden virtuális blokkhoz tartozzon egy leíró /OM-ben/

- ☐ Virtuális címtér ↔ fizikai címtér transzformáció /összerendelés/

- ☐ Memória kezelő egység (MMU) → HW és/vagy SW

- Átviteli igény jelzése az operációs rendszer /OR/ felé

- OR betölti az információt a háttértárolóról

- Leíró tábla kitöltés → OR. → SW

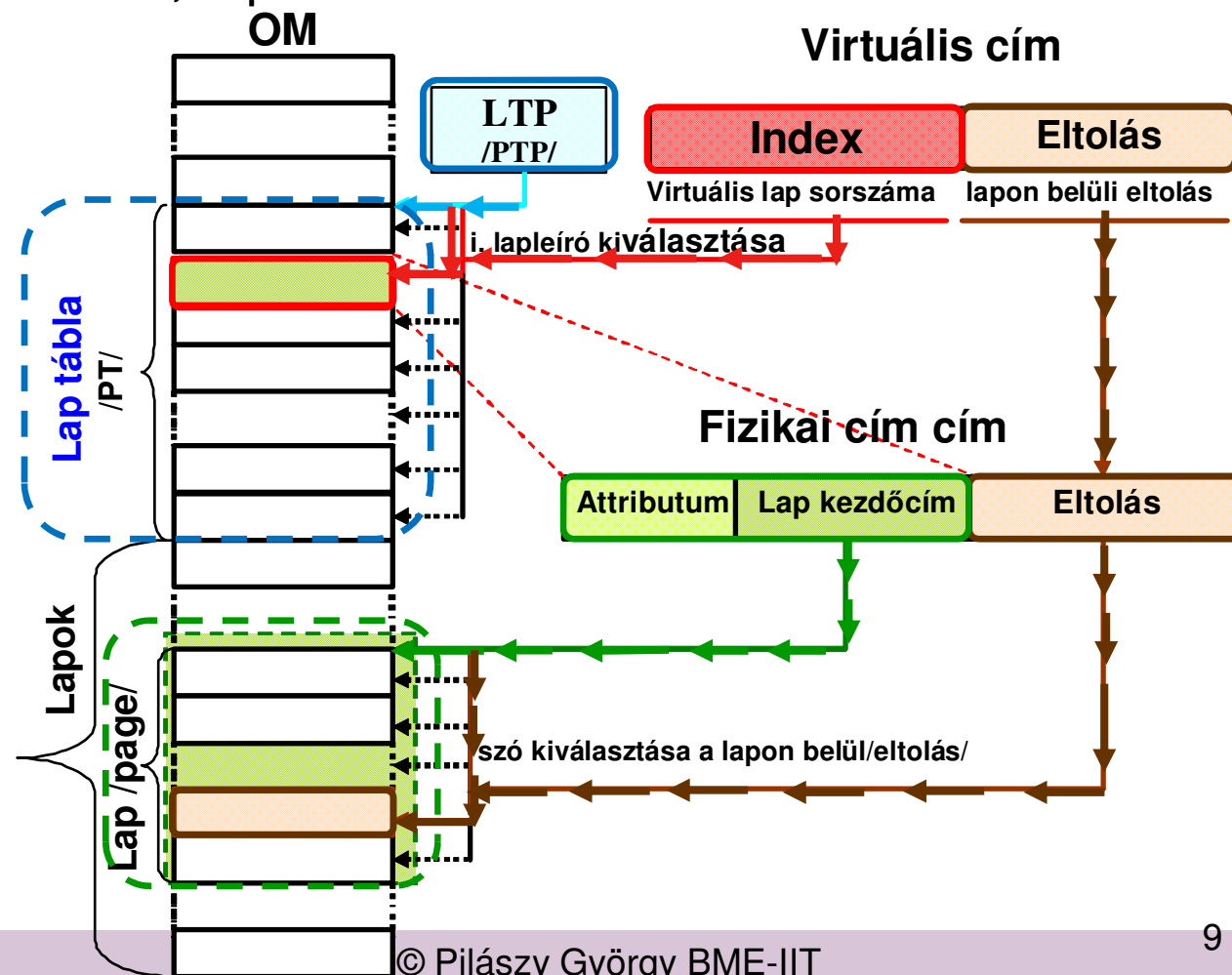
- Címtranszformáció MMU HW

- ☐ Felhasználó felé transzparens+védelem

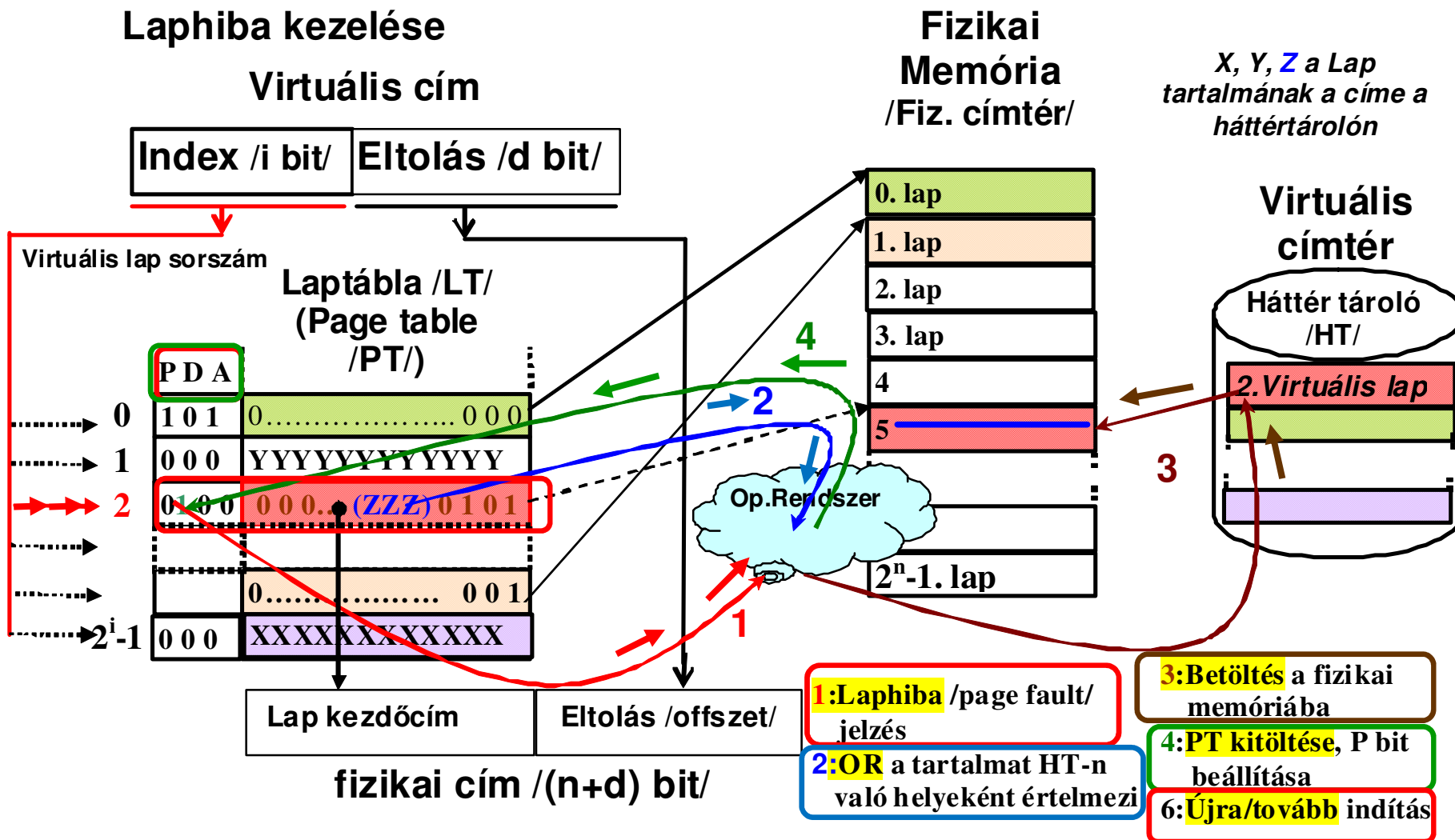
- ☐ Kialakítás : Lap/szegmens - szervezésű virtuális tárkezelés

Lapszervezésű virtuális tárkezelés

- ❑ Fix méretű lapok
- ❑ Transzformációs tábla /laptábla/ az OM-ben
- ❑ Lapeleíró: lap kezdőcím, lapra vonatkozó attribútumok

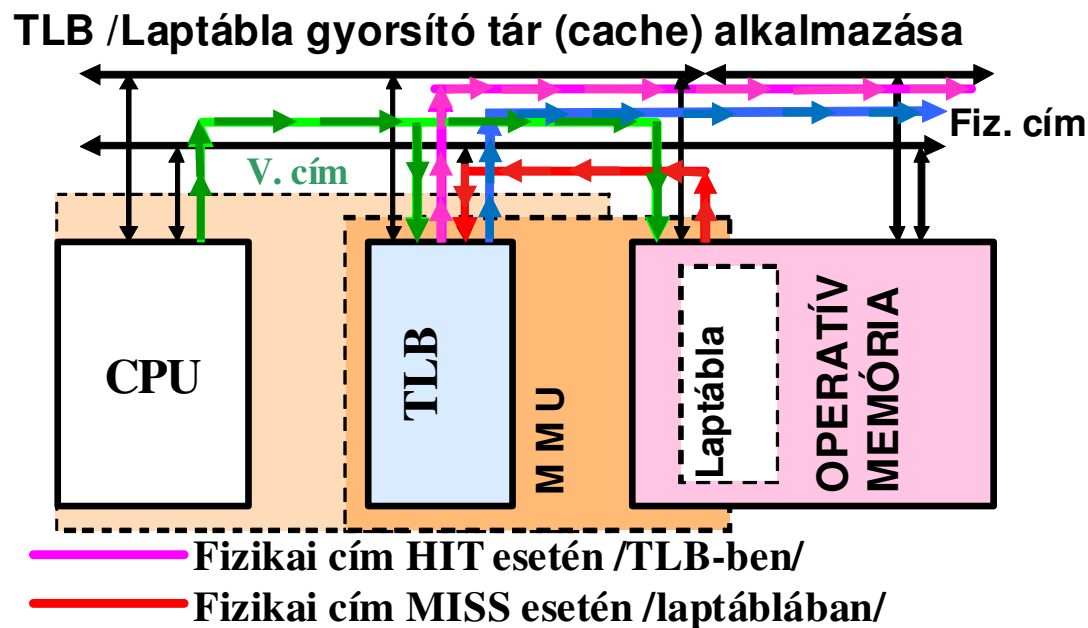


Lap hiba (*page fault*) kezelése



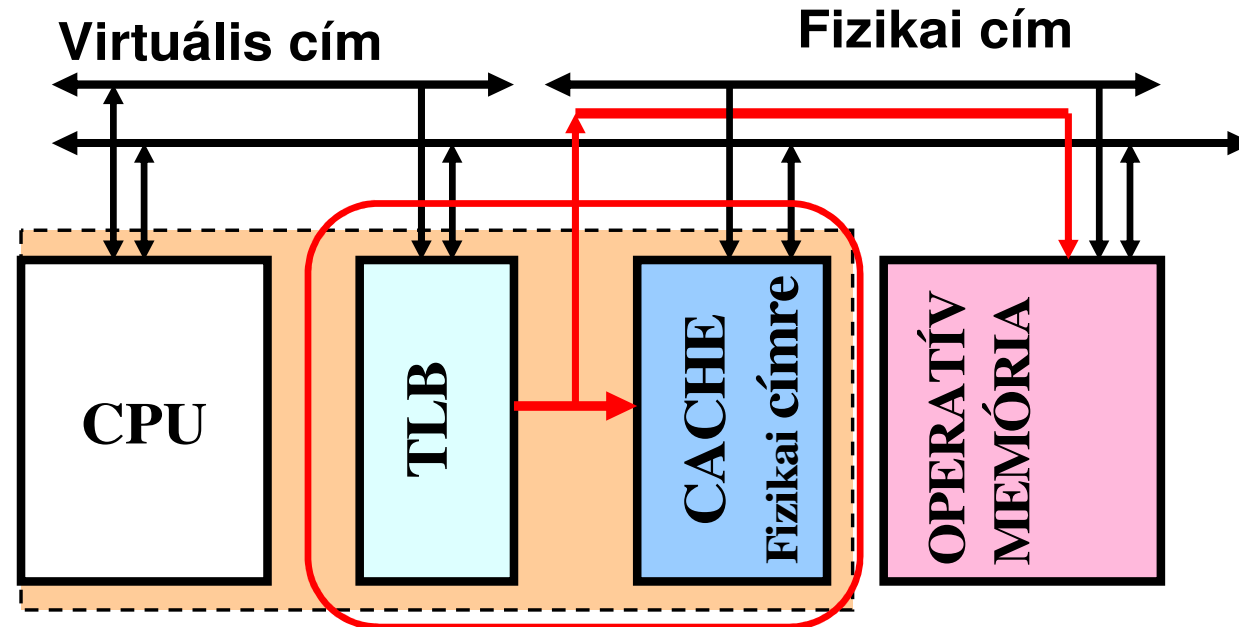
Megoldandó problémák

- ❑ Sebesség /ne kelljen mindig többször az OM-hez fordulni/
 - *Laptábla tartalmát egy erre a célra kiépített gyorsító tárban (TLB Translation Lookaside Buffer) (cache) tartani*



TLB hatása a CACHE működésére

CACHE a TLB után

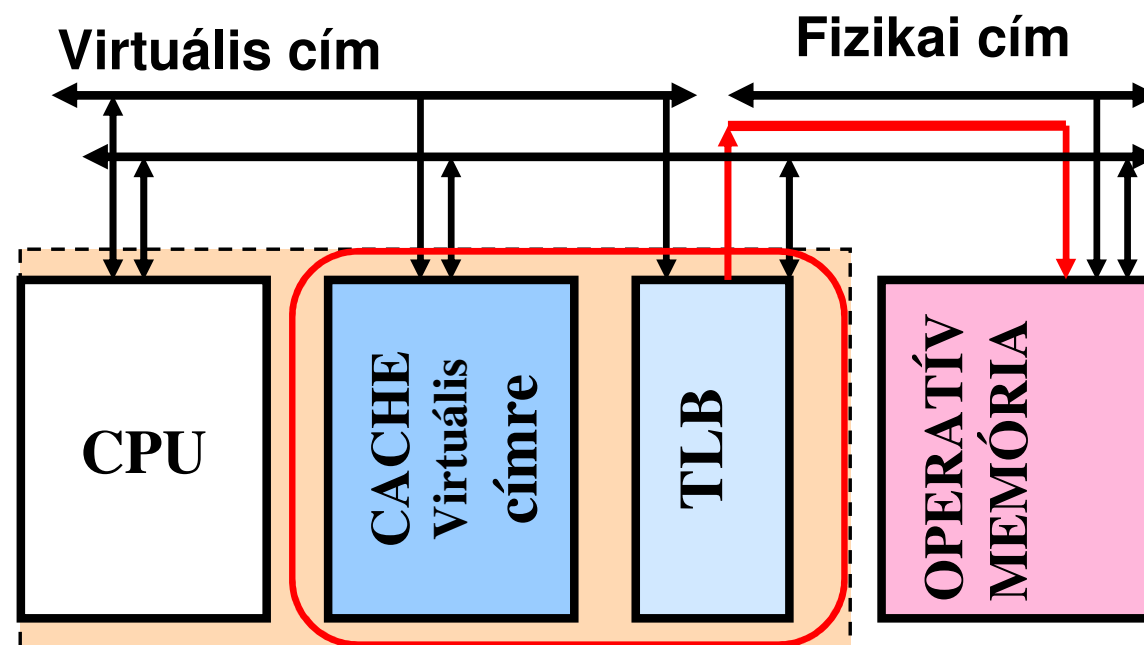


A CACHE tartalom keresés csak a TLB keresés után indul

→ Lassabb

→ Egyszerűbb megvalósítás

CACHE a TLB előtt



A CACHE-ben és a TLB-ben egyszerre keres

→ Gyors

→ **probléma:** ha V1 és V2 ugyanarra a fizikai címre mutat →

CACHE-ben V1 és V2 példány van (melyiket írtuk át - adat **egyezés** ?!)

Megoldandó problémák

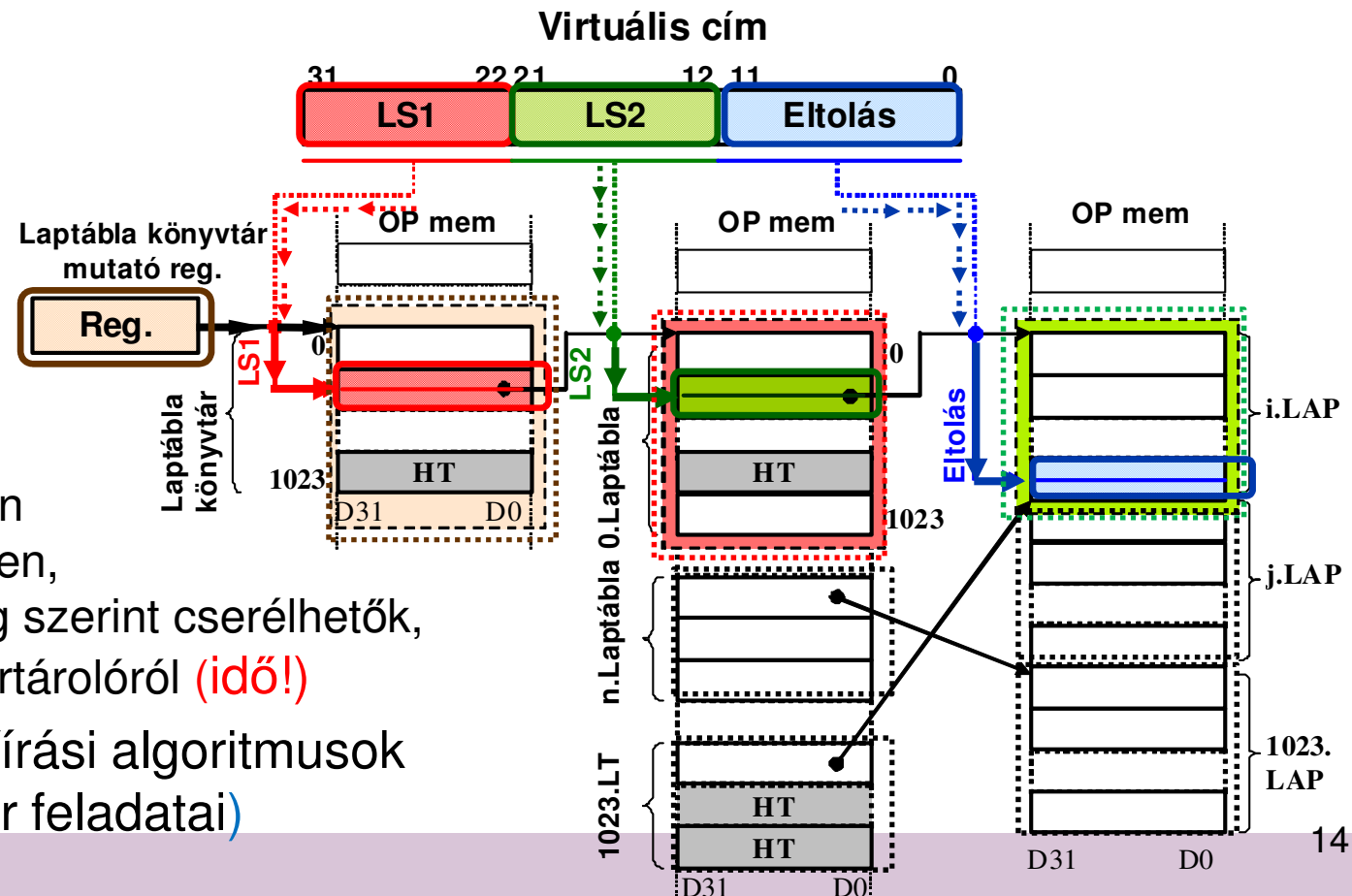
❑ Laptábla mérete

Minden virtuális laphoz tartozik egy leíró

(pl.: lapméret 4KB, a leíró 4byte → 4GB címtérhez $\Sigma 4MB$ lenne az egylépcsős laptábla mérete!)

■ Több lépcsős laptábla → Laptábla könyvtár → laptábla → lap

Kétlépcsős laptábla szervezés



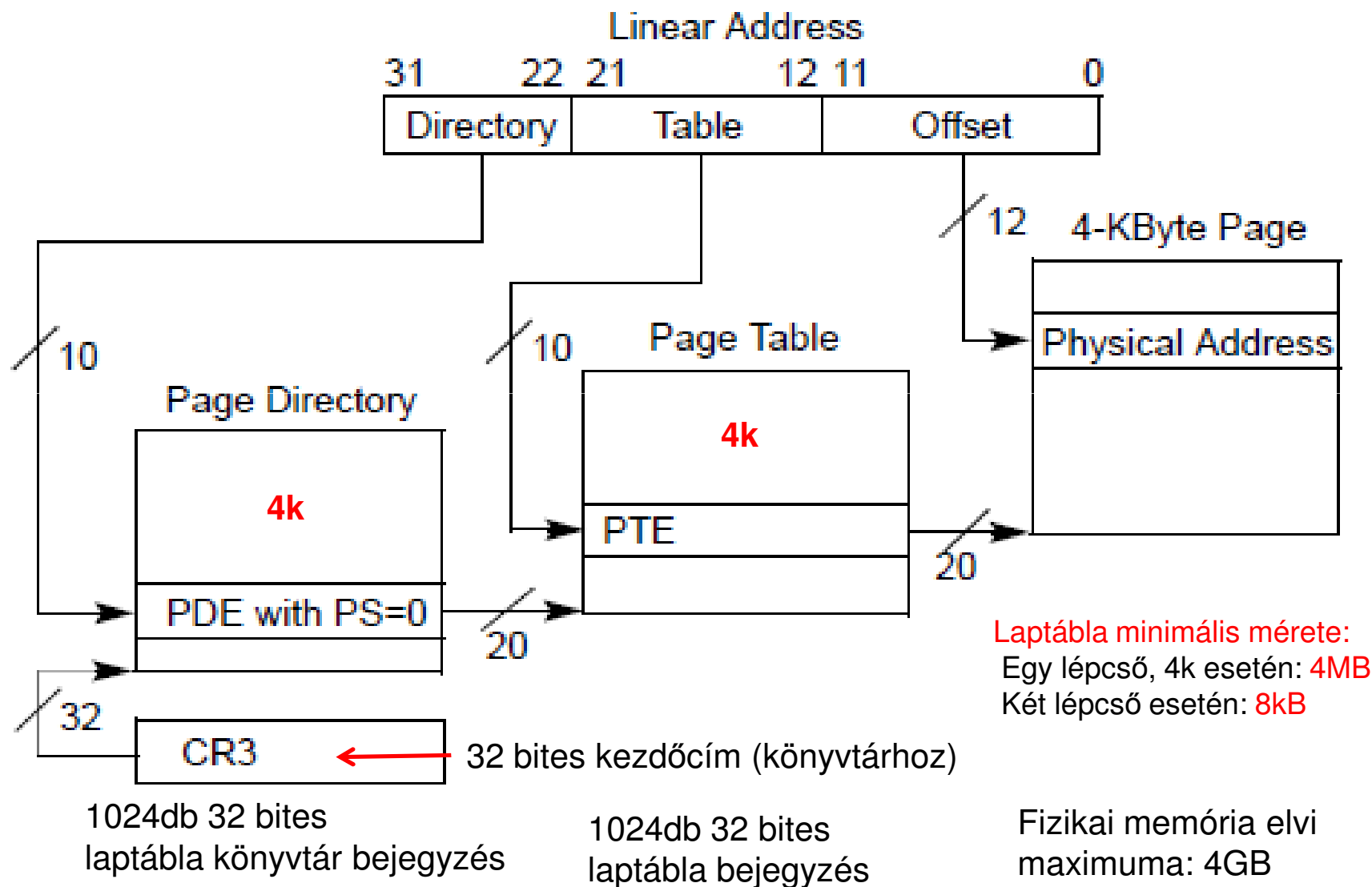
Csak a könyvtár van
állandóan az OM-ben,
a laptáblák szükség szerint cserélhetők,
behozhatók a háttértárolóról (idő!)

Behozatali/csere/írási algoritmusok
(→ Op. Rendszer feladatai)

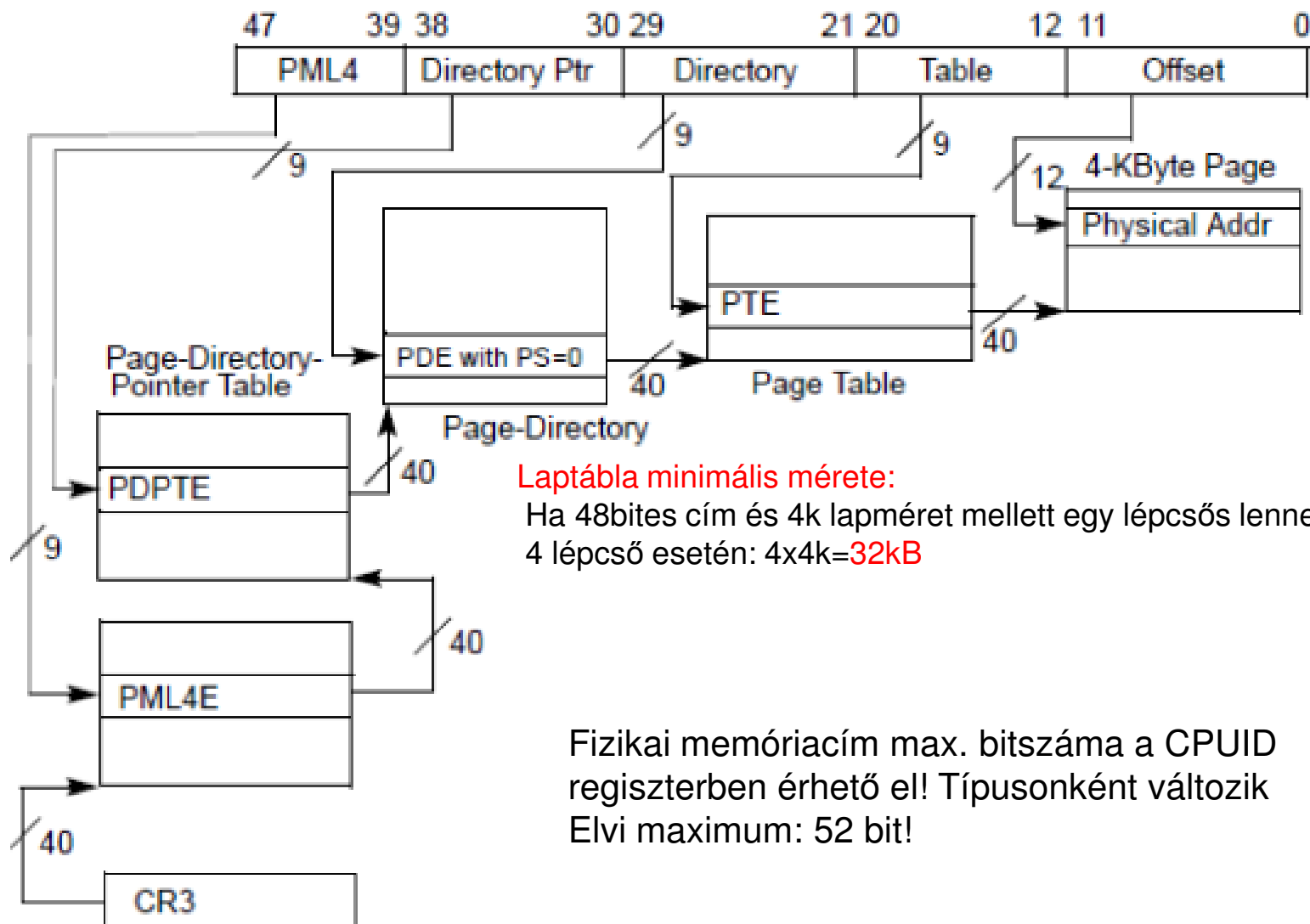
Konkrét példák az Intel64-ből

- Többlépcsős laptábla szervezés támogatása
- Laptábla könyvtárak és lapleírók az OM egy részét elfoglalják
- Választható lapméretek: 4k, 2M, 4M, 1G
- Ezek vegyesen is használhatók
- Leíró táblák kitöltését és kezelését nem bízuk a felhasználóra → processzorban hardveres támogatás kell
- A táblák mérete (memória igénye) független a fizikailag kiépített memóriától
- A teljes struktúra gyökerének mutatója a CR3 regiszterben

4k lapméret, 2 lépcső 32 bites lineáris cím



4k lapméret, 4 lépcső 48 bites lineáris cím



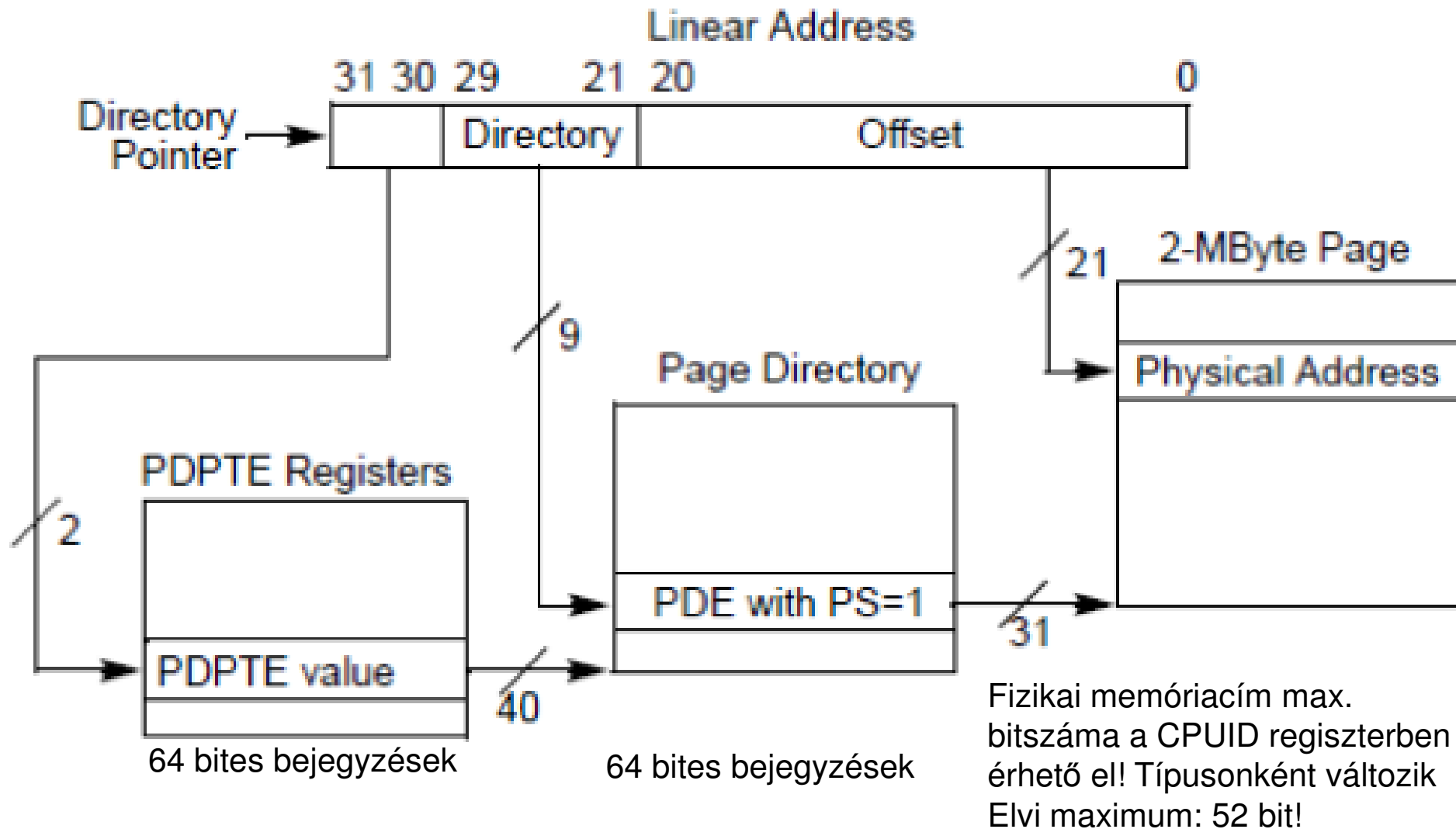
Laptábla minimális mérete:

Ha 48 bites cím és 4k lapméret mellett egy lépcsős lenne: **512GB!**

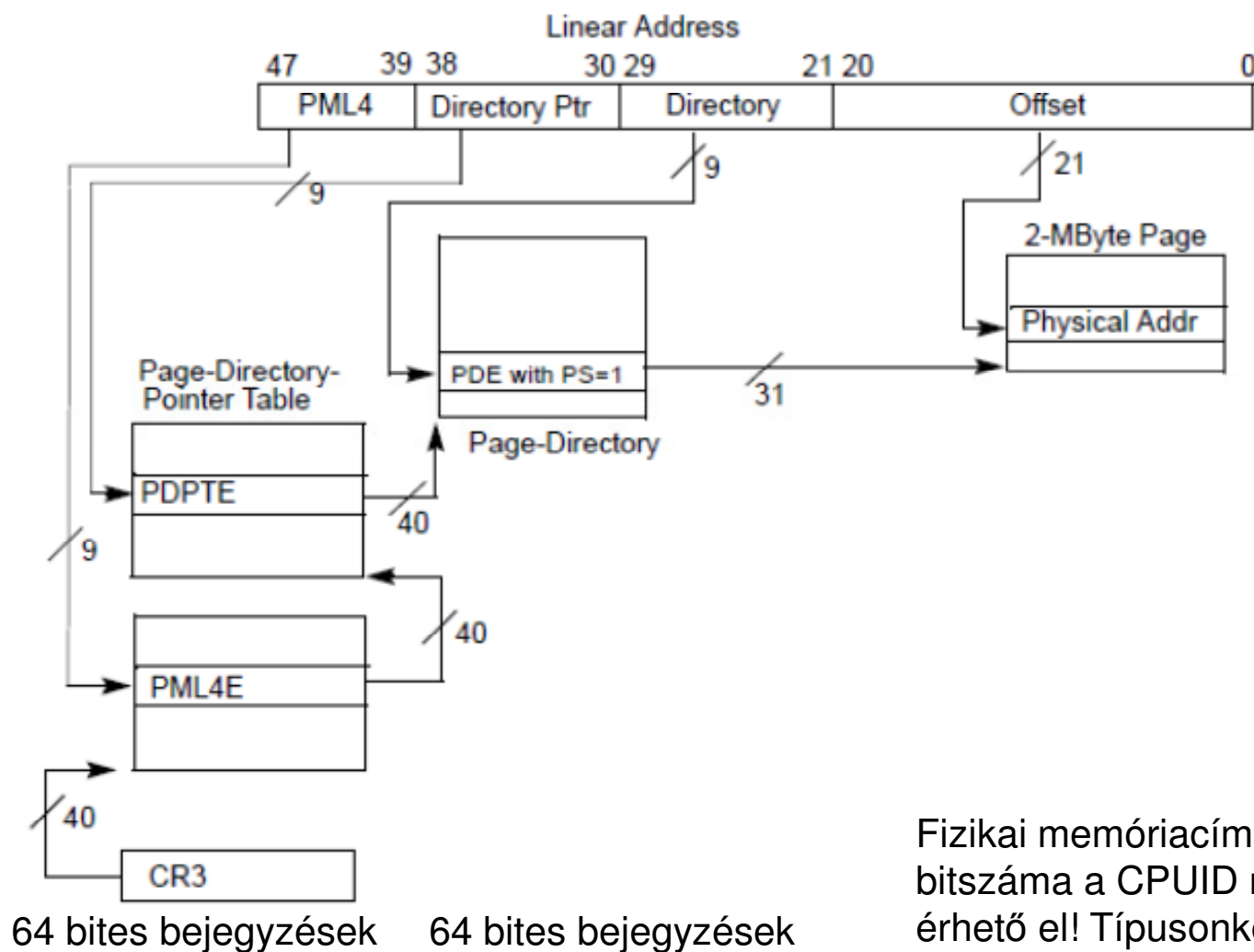
4 lépcső esetén: $4 \times 4k = \mathbf{32kB}$

Fizikai memóriacím max. bitszáma a CPUID regiszterben érhető el! Típusonként változik
Elvi maximum: 52 bit!

2M lapméret, 2 lépcső

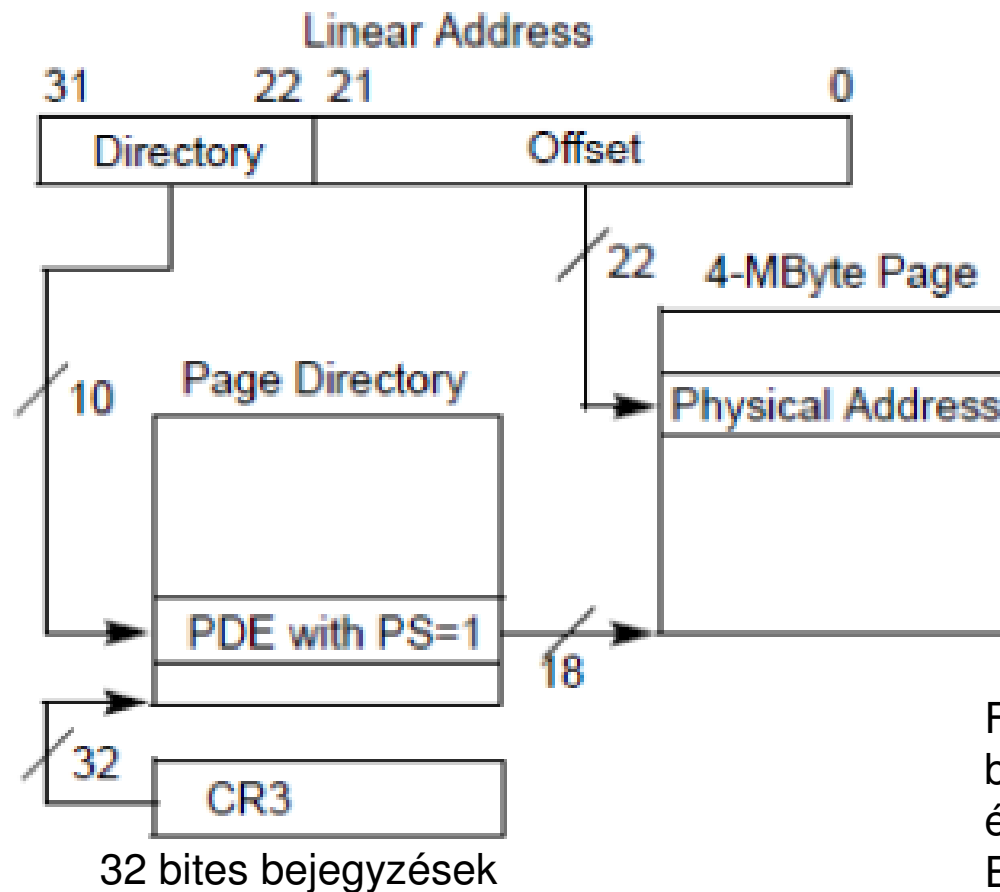


2M lapméret, 48 bites cím



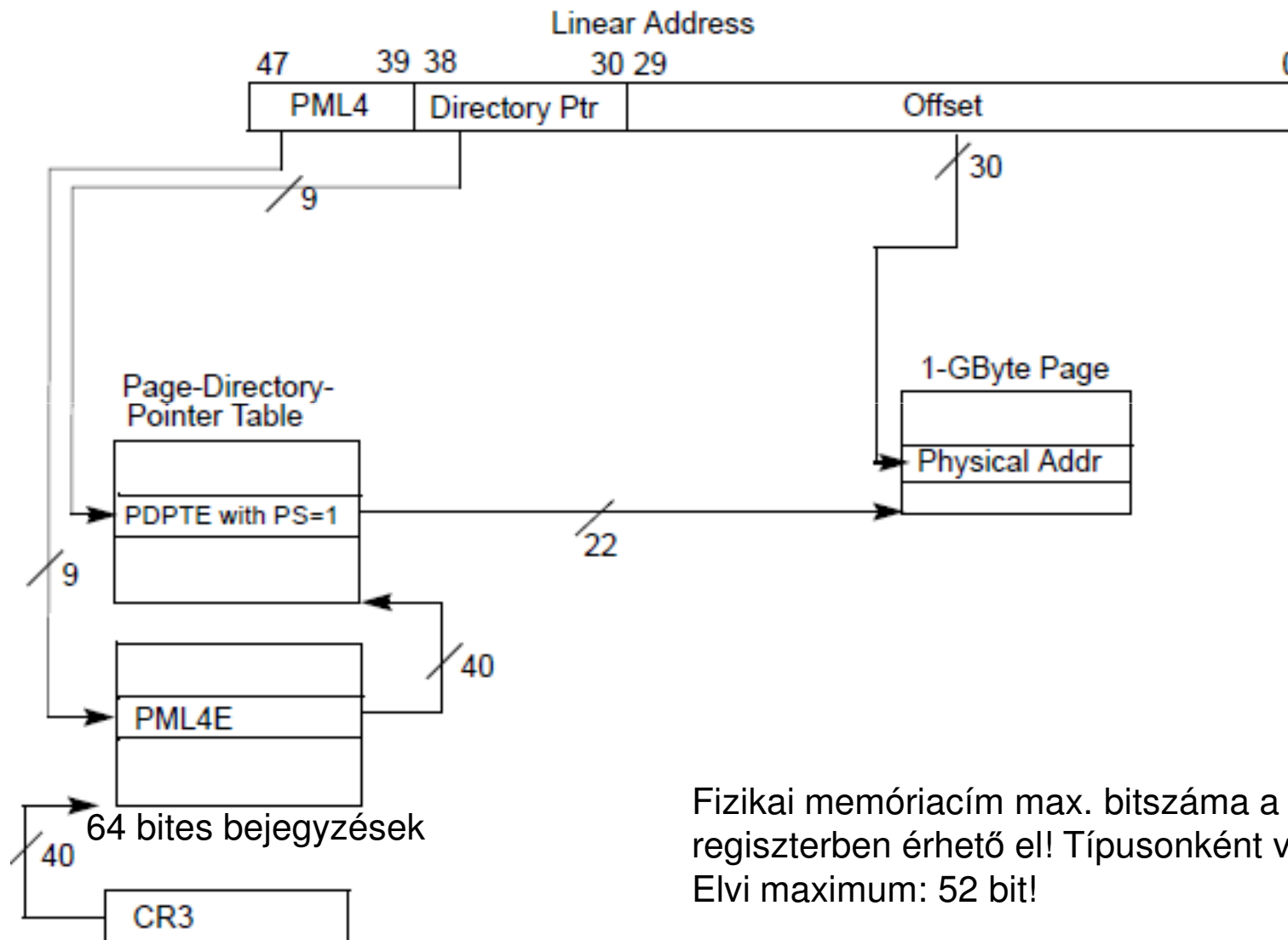
Fizikai memóriacím max.
bitszáma a CPUID regiszterben
érhető el! Típusonként változik
Elvi maximum: 52 bit!

4M lapméret, 1 lépcső



Fizikai memóriacím max.
bitszáma a CPUID regiszterben
érhető el! Típusonként változik
Elvi maximum: 40 bit!

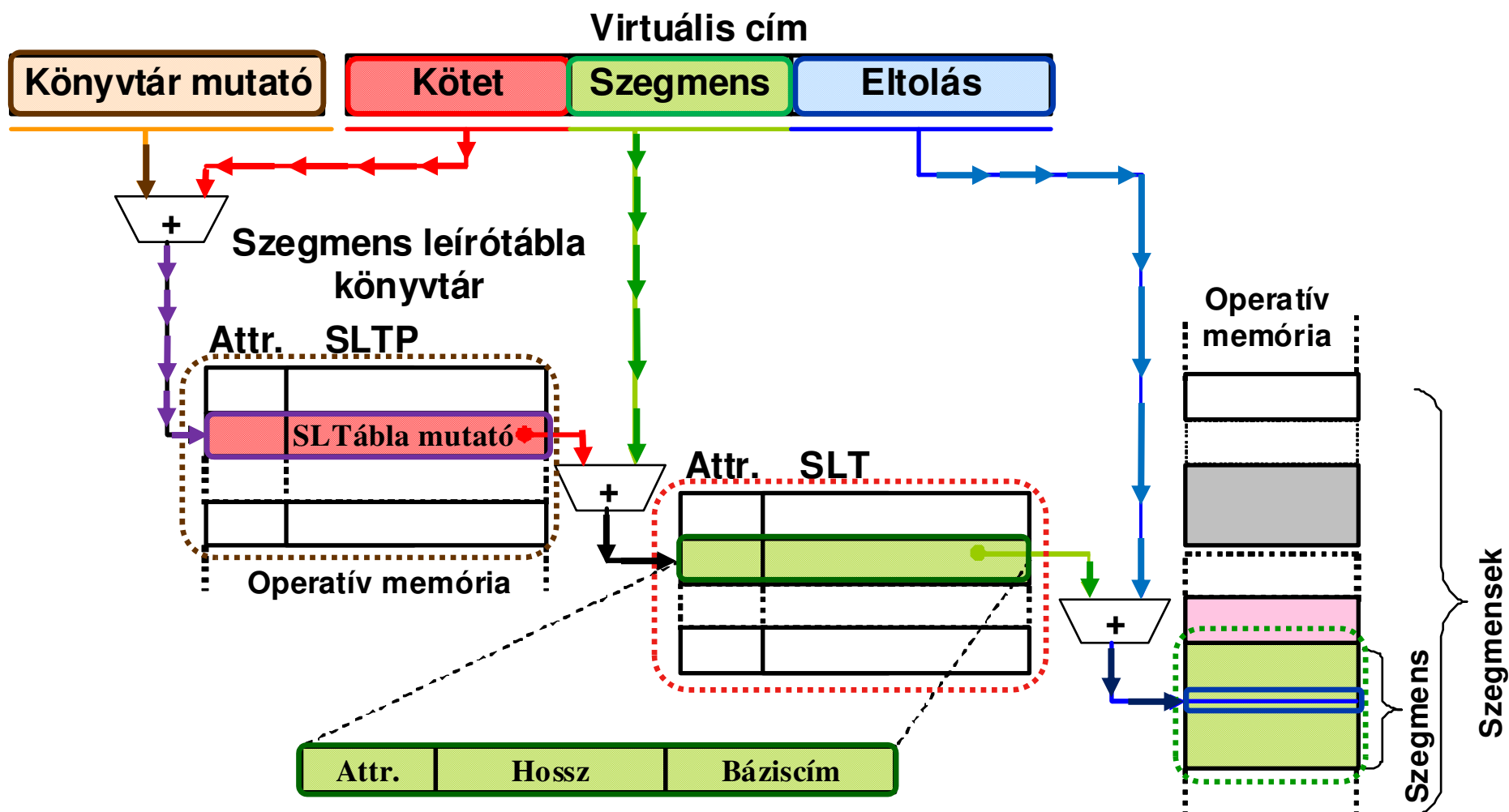
1 GB lapméret



Fizikai memóriacím max. bitszáma a CPUID regiszterben érhető el! Típusonként változik
Elvi maximum: 52 bit!

- ❑ A logikai objektumokhoz /pl.: *program, adat, egyéb*/ változó méretű memóriarészeket /*szegmenseket*/ használunk
- ❑ Címfordítás hasonló, mint a lapozásnál
 - Egy szegmens összefüggő /*folytonos*/ fizikai tárterületet igényel
 - Szegmens hosszát is meg kell adni →**védelem!**
 - Címszámítás aritmetikai összeadás →**bonyolultabb hw.**
- ❑ Egy felhasználót /alkalmazást/ a hozzá tartozó teljes, őt leíró környezetével /***context***/ -*itt kötet*- együtt kezeljük

/kétlépcsős/



Szegmensszervezés

Előnyök:

- Jobban illeszkedik a „programozási szemlélethez”
- Hatásos védelem alakítható ki
- A felhasználó számára transzparens

Hátrányok:

- Az eltérő méretek miatt memória szétesés /fragmentation/ jöhet létre
- Bonyolultabb szegmens–csere algoritmusok kellenek

Lapszervezés

Előnyök:

- *A fix lapméret jól illeszthető a háttértár szektor méretéhez*
- *Elkerülhető a memória „szétesés”*
- *A címtranszformáció nem igényel aritmetikát*
- *Nagy objektumoknál (pl.: program memória szegmens) a fizikai tárban **nem kell** folytonos tárterület →*
 - *Jól kihasználható a fizikai tárterület*
 - *A felhasználó számára transzparens*

Hátrányok:

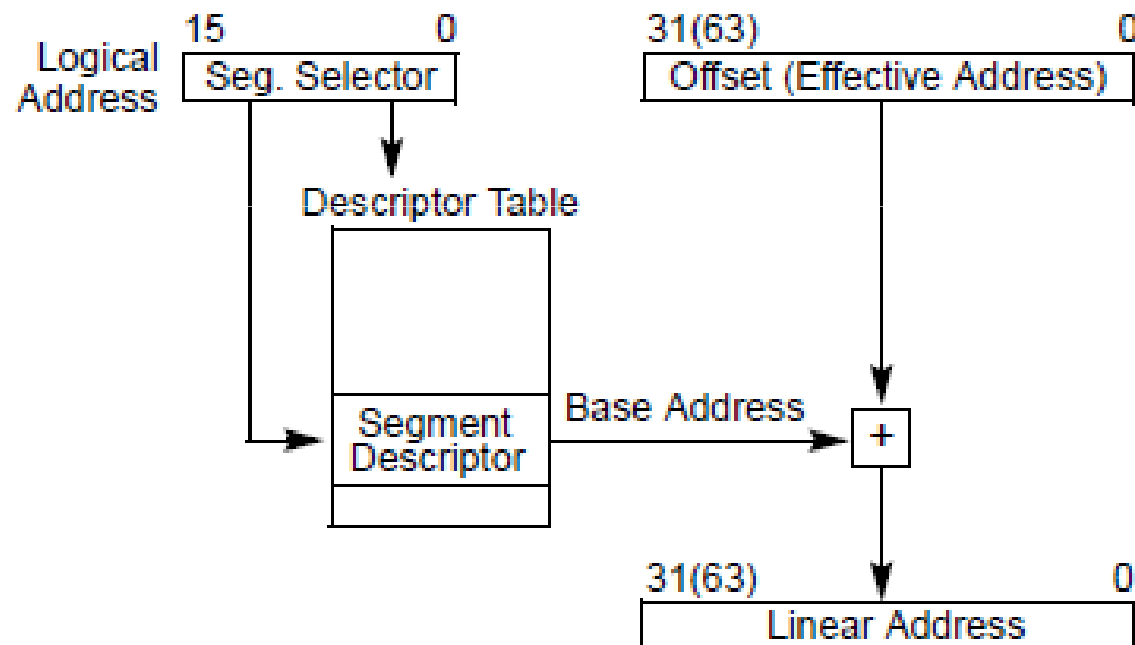
- *Ha csak lapozás van, csökken a védelem lehetősége*
- *Belső szétesés (pl.: lapméret 4096 byte, kell 5000 byte)*

Szegmentált lapszervezésű virtuális tárkezelés

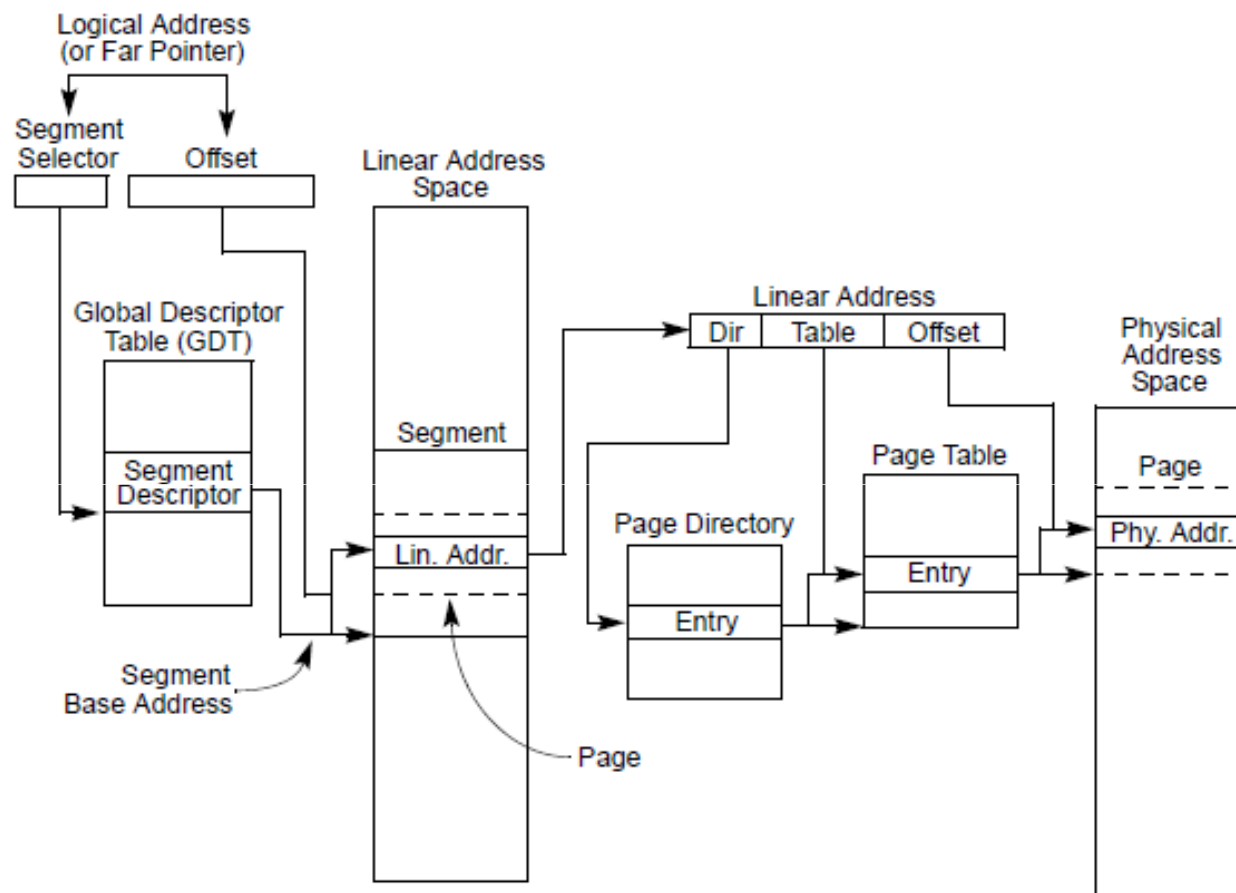
A két módszert egyesíti → (lásd Intel64 processzoroknál)

Szegmentálás, logikai cím → lineáris cím (Intel64)

Logikai cím: 16bit szegmens + 32 bit eltolás, de az eredmény is 32 bites
 Vagy cím: 16bit szegmens + 64 bit eltolás, de az eredmény is 64 bites
 (64 bites módban nincs fizikailag implementálva mint a 64 bit!)



Szegmentálás+lapszervezés (Intel64)



← Szegmentálás → ← Lapszervezés →

- Tömbkapcsolásos memória bővítés
- Indexregiszteres memória bővítés
- Bázisregiszter alkalmazása
- Felhasználói szintű overlay
- Virtuális tárkezelés
- Szegmens szervezés
- Lapszervezés
- Szegmentált lapszervezés