HÁLÓZATI RENDSZEREK ÉS SZOLGÁLTATÁSOK TANSZÉK

MŰEGYETEM 1782

Budapest,
2025.03.11.

SZÁMÍTÓGÉP ARCHITEKTÚRÁK

A virtuális memória

Horváth Gábor, Belső Zoltán

BME Hálózati Rendszerek és Szolgáltatások Tanszék
ghorvath@hit.bme.hu, belso@hit.bme.hu

VIRTUÁLIS TÁRKEZELÉS

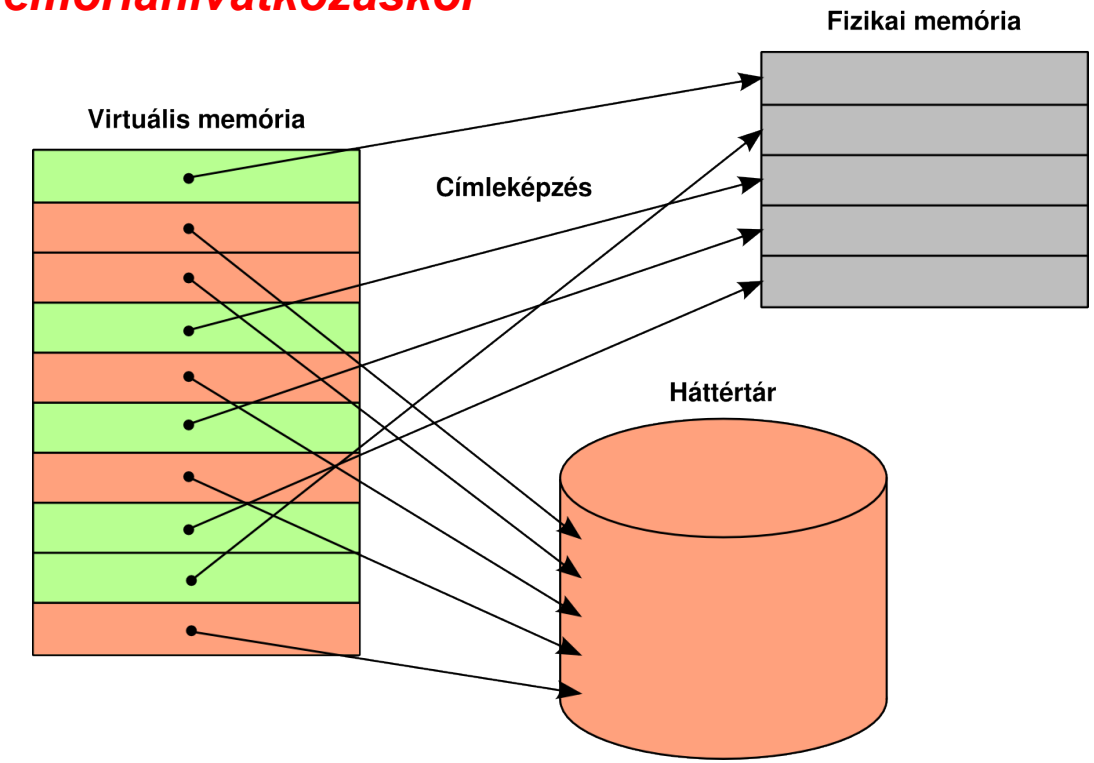
- Motiváció:
  - Multitaszking környezet
    - Taszkok jönnek-mennek
    - A programunknak jutó memóriaméret változó
  - Olcsó 64 bites processzorok
    - Ebből pl. az AMD64 48 bitet használ: 256 TB
    - Ennyi memóriát nem tudunk hozzáilleszteni

- A CPU minden programnak a teljes címtartományt felajánlja
  - 0-tól → a címtartomány tetejéig
  - Ne legyen gond a memóriaméret
- Ezt a „virtuális” memóriát kell leképezni a fizikaiba

  → ***Virtuális tárkezelés***

VIRTUÁLIS TÁRKEZELÉS

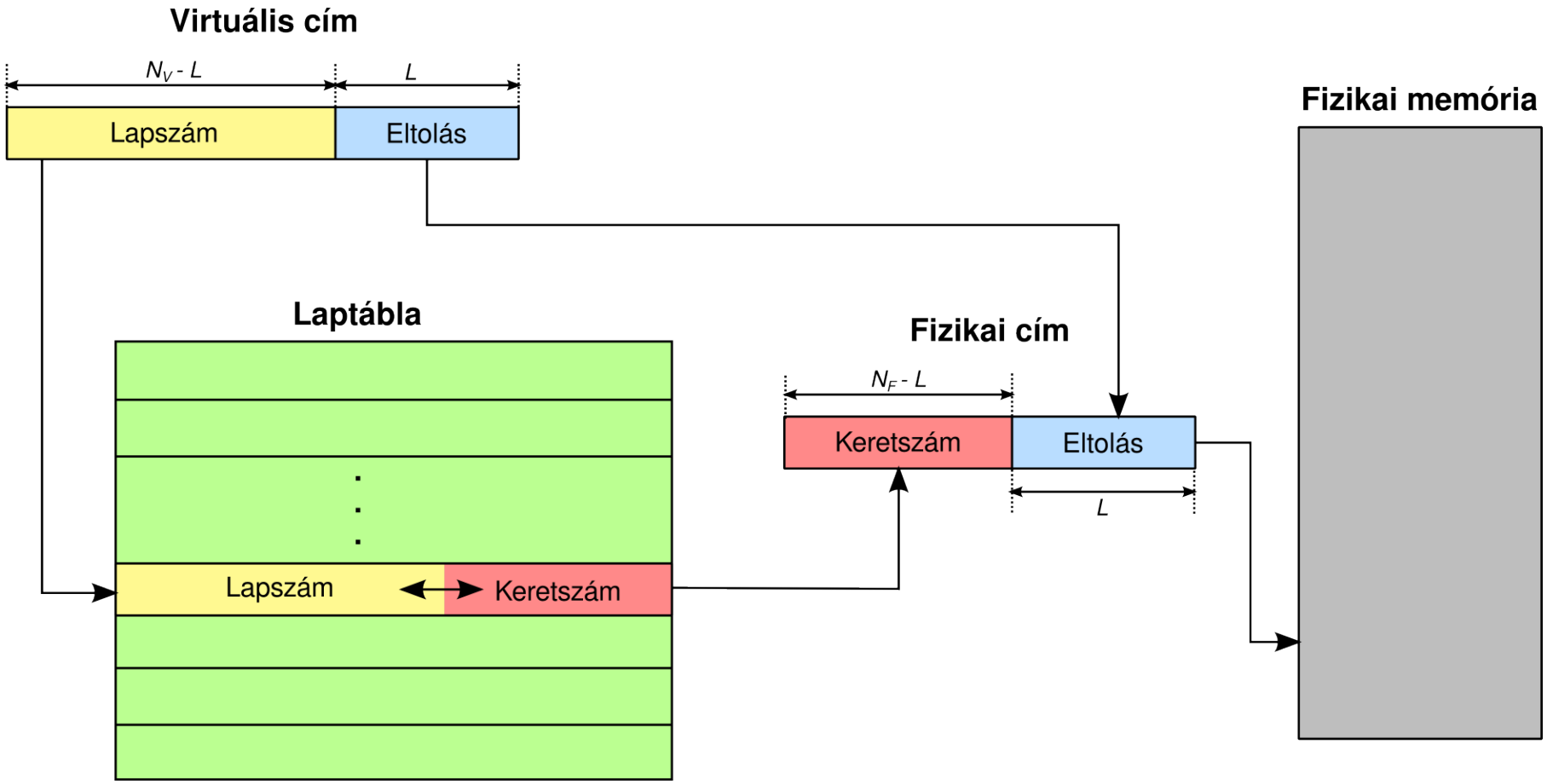
- Programok: ***Virtuális címeket*** használnak
- CPU lábain/buszon: ***Fizikai címek*** jelennek meg
- Virtuális → fizikai cím leképzés: ***Címfordítás***
  - Aki csinálja: ***MMU***
  - Amikor csinálja: ***minden memóriahivatkozáskor***
  - Egysége:
    - ***Lap*** (fix)
    - vagy ***szegmens*** (változó méretű)
- Ha nem fér mindenki a fizikai memóriába → háttértárra kerül (swap-elés)



CÍMFORDÍTÁS

- A virtuális címtartományt fix méretű ***lap***okra particionáljuk
- A fizikai címtartományt ugyanekkora ***keret***ekre osztjuk
- Méretek:
  - Lapok mérete = 2^L
  - Alsó L bit: lapon belüli eltolás
  - Felső bitek: virt. címeknél lapsorszám, fiz. címeknél keretsorszám
- Lap ↔ keret összerendeléseket tárolja: ***laptábla***
- Összerendeléshez kell:
  - Lap sorszáma
  - Keret sorszáma
  - Védelmi információ (írható/olvasható)
  - Vezérlő bitek:
    - Valid
    - Dirty

CÍMFORDÍTÁS



Laptábla implementációk

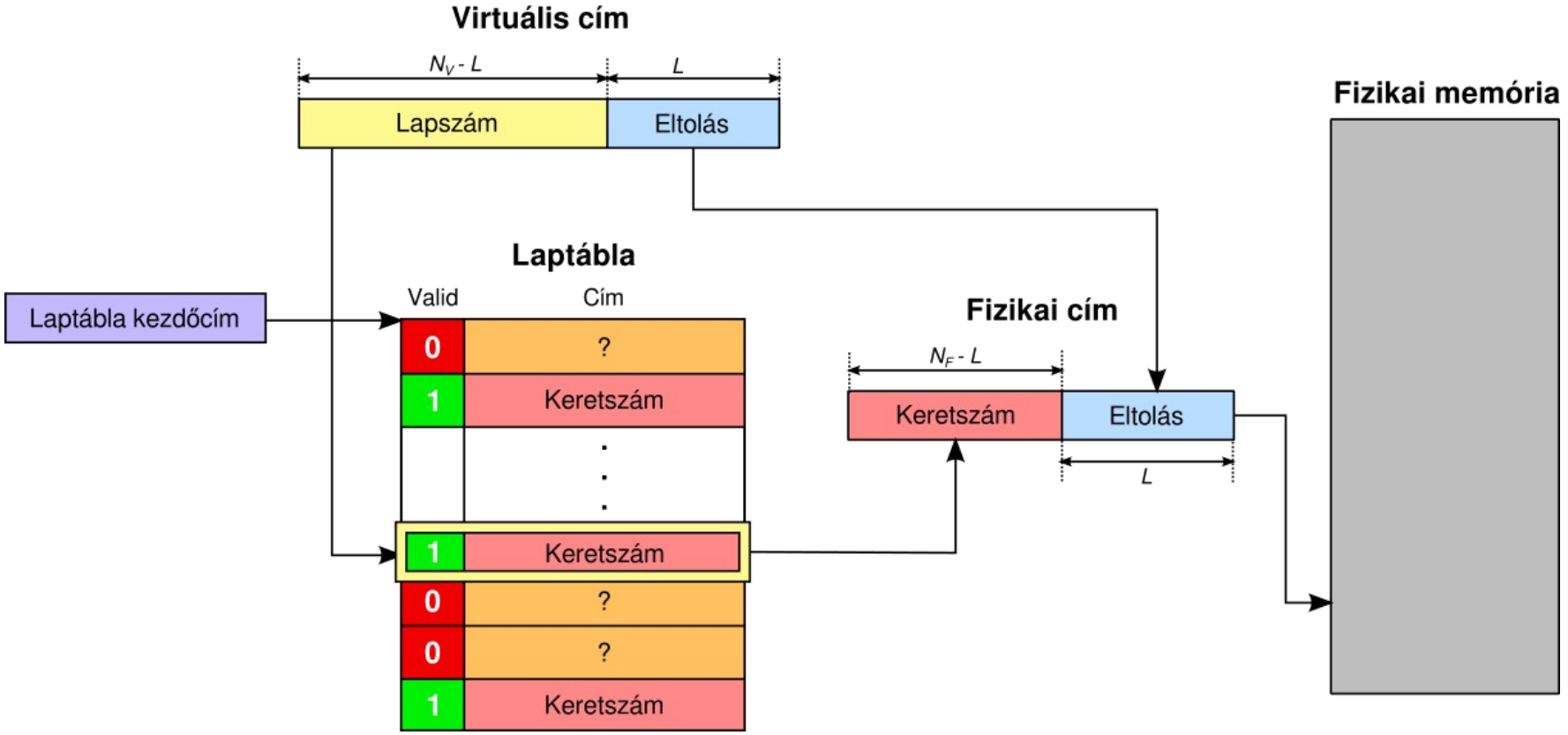
LAPTÁBLA IMPLEMENTÁCIÓK

- Cél:
  - Címfordítás legyen minél gyorsabb
    → virtuális cím szerinti keresés legyen gyors
    → minél kevesebbszer kelljen a memóriához nyúlni
  - Laptábla legyen a lehető legkisebb

- Eszköz:
  - Speciális adatszerkezetek
    - Egyszintű laptábla
    - Többszintű (hierarchikus) laptábla
    - Inverz laptábla
    - (Virtualizált laptábla)

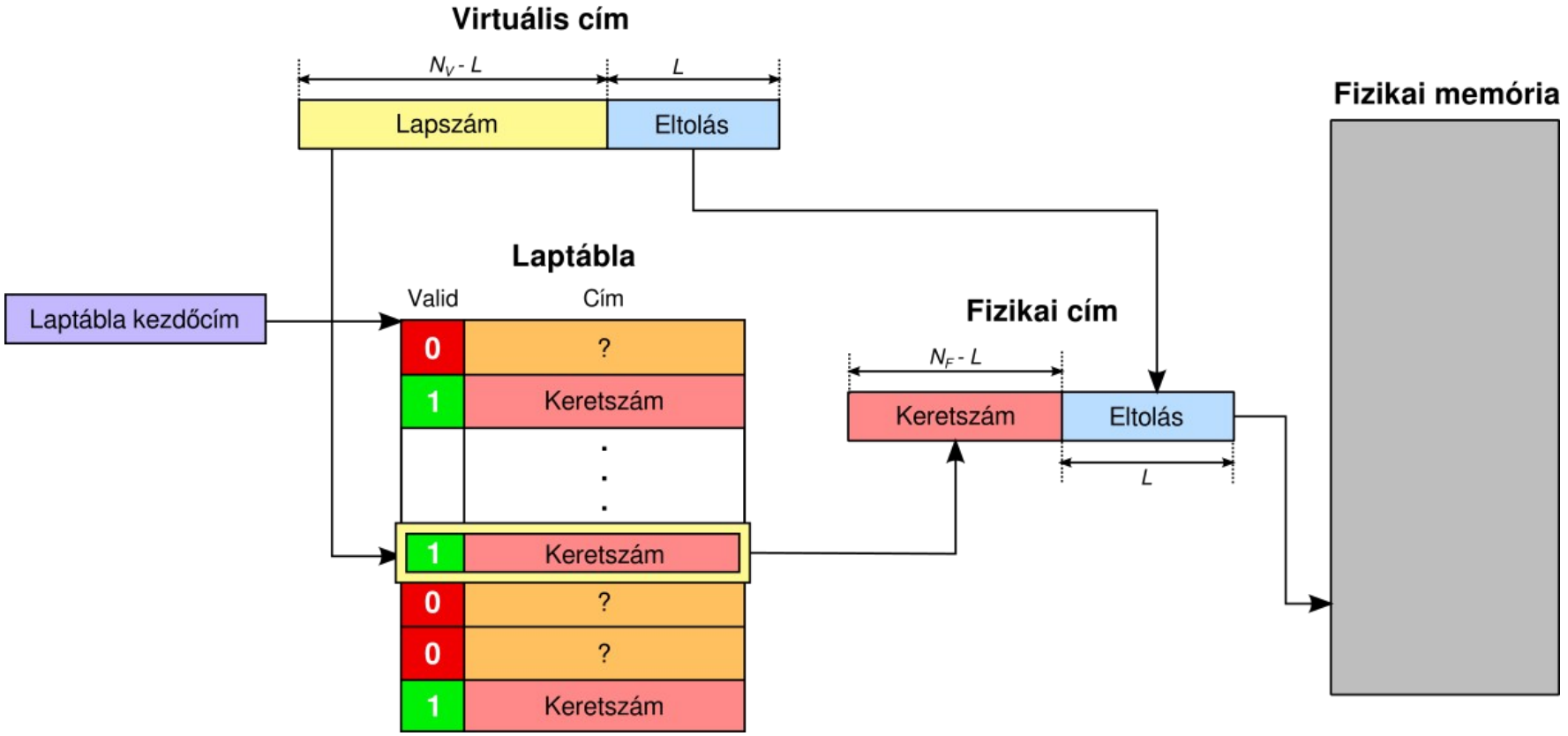
EGYSZINTŰ LAPTÁBLA

- Laptáblabejegyzések tömbje
- i. bejegyzés: i. laphoz tartozó keret



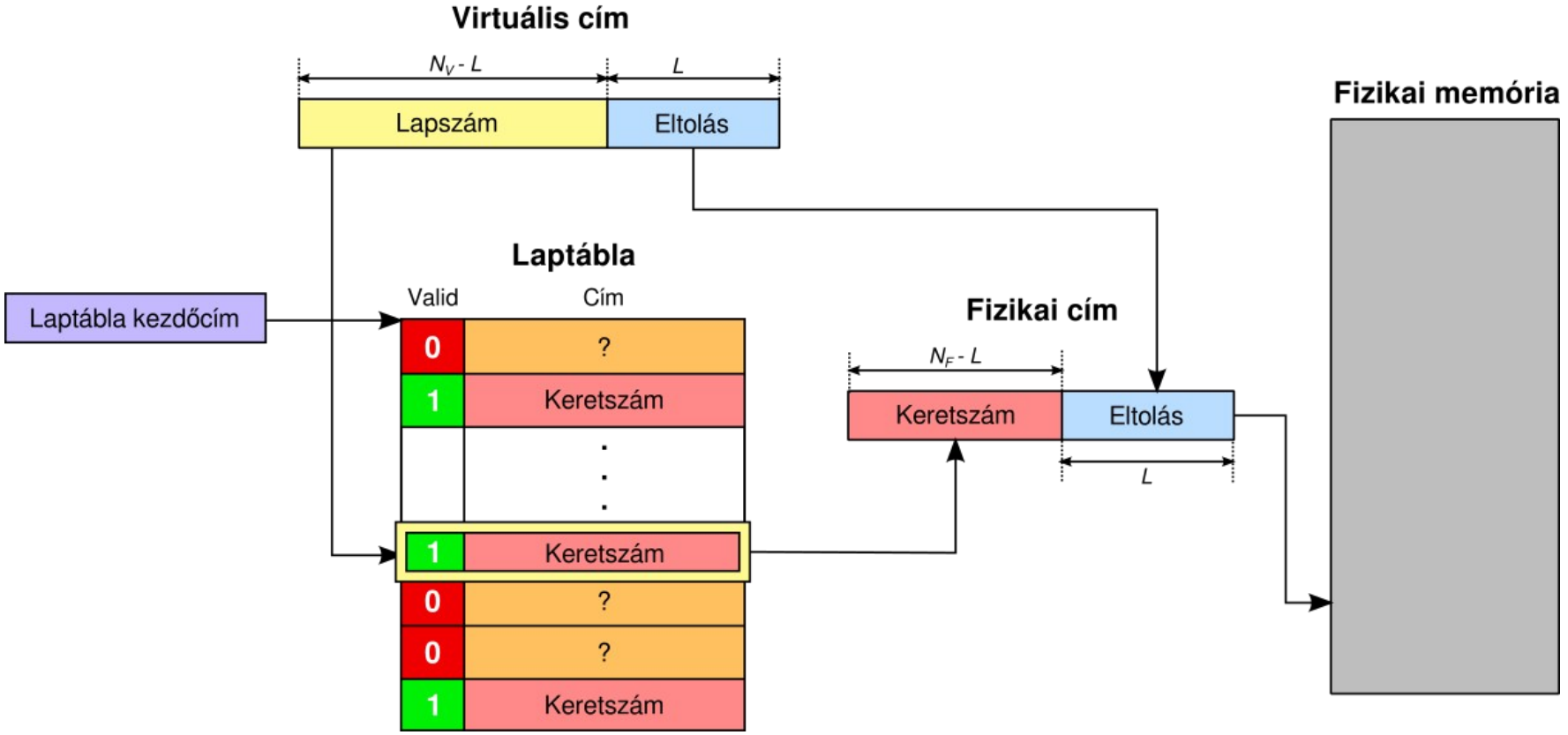
EGYSZINTŰ LAPTÁBLA

- Keresés: gyors!
- Bejegyzés megtalálása: pontosan 1 memóriaművelet!



EGYSZINTŰ LAPTÁBLA

- Bejegyzés mérete: kicsi
  - Nem kell tárolni a virtuális címet → az pont az index
  - Nem kell külön tárolni a háttértáron való elhelyezkedést



EGYSZINTŰ LAPTÁBLA

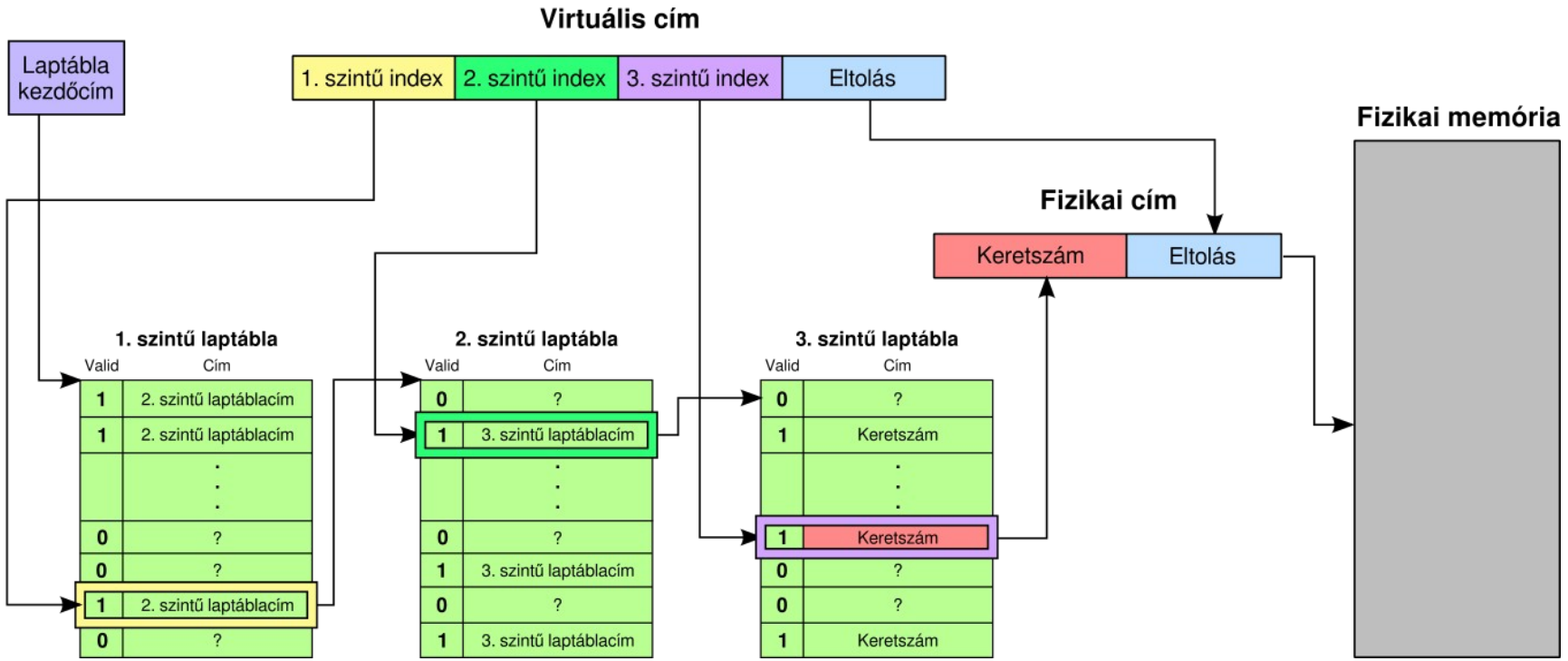
- Gyors, kevés memória-hozzáférés kell, kicsi a bejegyzés...
  ...miért nem használ mindenki ilyet?

A teljes laptáblának bent kell lennie a memóriában!

- Gyors kalkuláció:
  - 32 bites esetben, 4 kB lapokkal:
    - Lapméret: 12 bit, lapok száma: 32-12 = 20 bit, 1 mega-lap
    - 1 bejegyzés: 4 bájt elég, laptábla mérete: 4 MB
  - 64 bites esetben, 4 kB lapokkal:
    - Lapméret: 12 bit, lapok száma: 64-12 = 52 bit, 2^{52} db lap
    - 1 bejegyzés: 8 bájt, laptábla mérete: $8 * 2^{52}$ = 32 PB

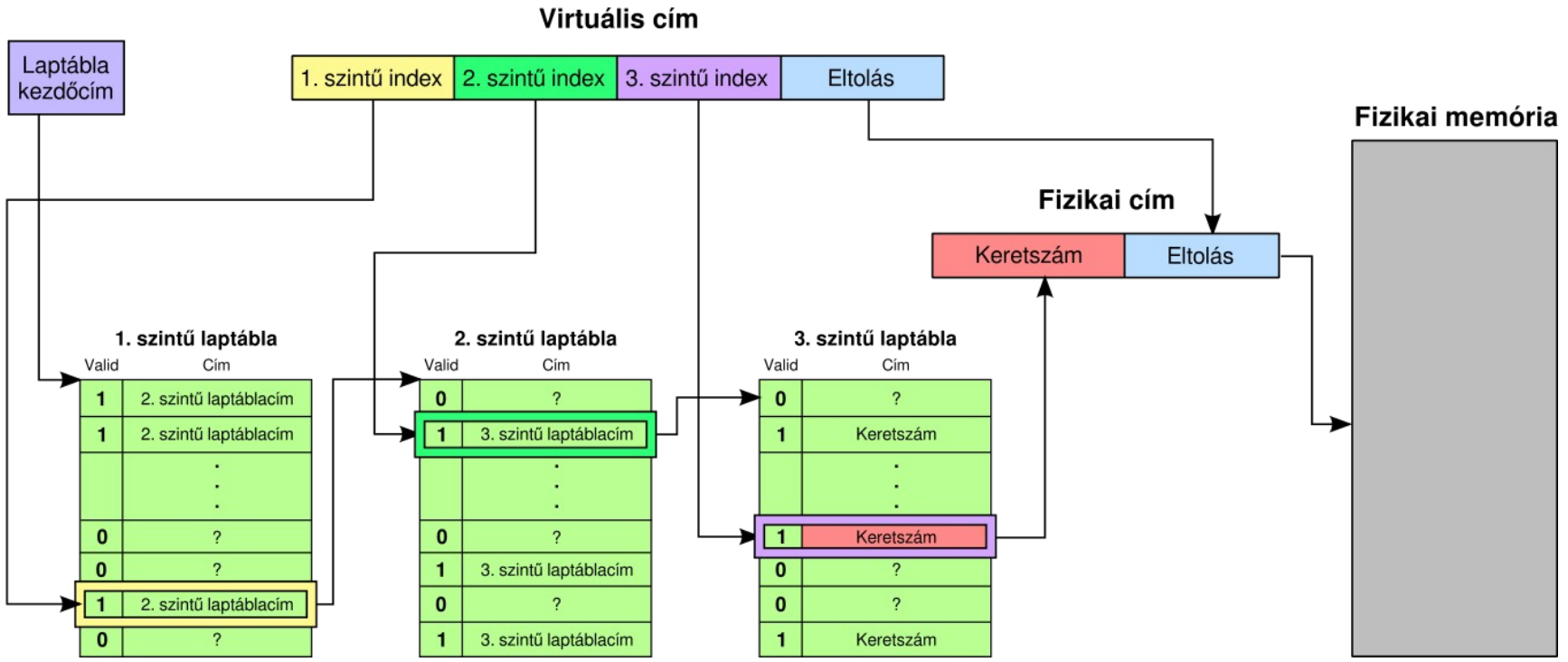
HIERARCHIKUS LAPTÁBLA

- Ötlet: magát a laptáblát is lapokra bontjuk
- A laptábla lapok elhelyezkedését egy másik laptáblában tároljuk, azokét egy harmadikban, stb.

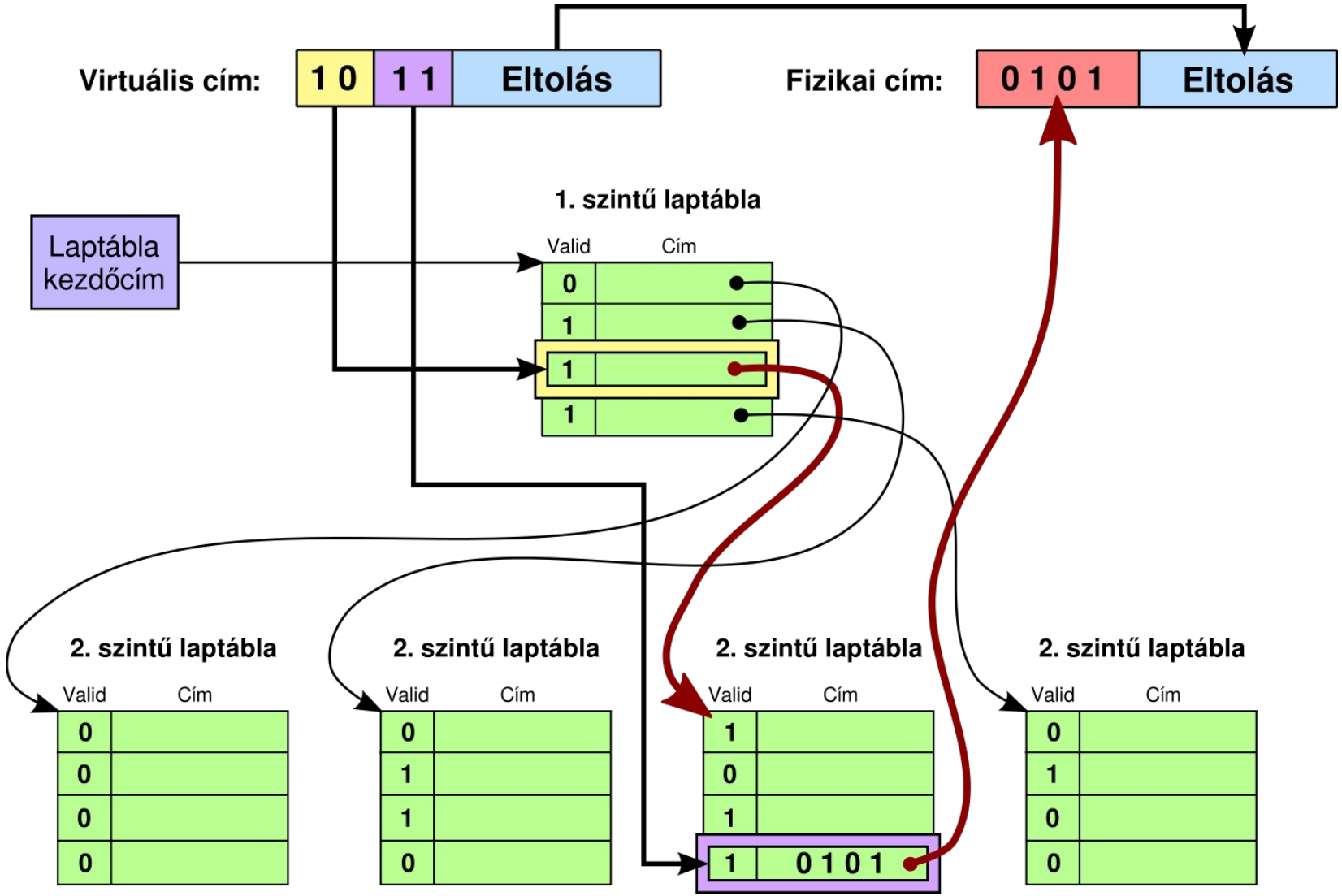


HIERARCHIKUS LAPTÁBLA

- Csak azt tartjuk a memóriában, ami tényleg kell
- Több memóriaművelet kell a címfordításhoz → lassú!



KÉTSZINTŰ LAPTÁBLA, PÉLDA



HIERARCHIKUS LAPTÁBLA

- Gyors kalkuláció:
  - 32 bites esetben, 4 kB lapokkal:
    - Lapméret: 12 bit, 1 bejegyzés: 4 bájt elég
    - Egy lapra 1024 bejegyzés fér → 10 bittel indexelhető
    - Kell 1024 laptábla lap, és még egy lap, ami ezekre mutat
    - 32 bites cím = 10 + 10 + 12 bites részek
      → 2 szintű laptábla kell
  - 64 bites esetben, 4 kB lapokkal:
    - Lapméret: 12 bit, 1 bejegyzés: 8 bájt
    - Egy lapra 512 bejegyzés fér → 9 bittel indexelhető
    - 64 bites cím = 7 + 9 + 9 + 9 + 9 + 9 + 12 bites részek
      → 6 szintű laptábla kell!
      6 memóriaművelet / címfordítás !!!
- X86, ARM

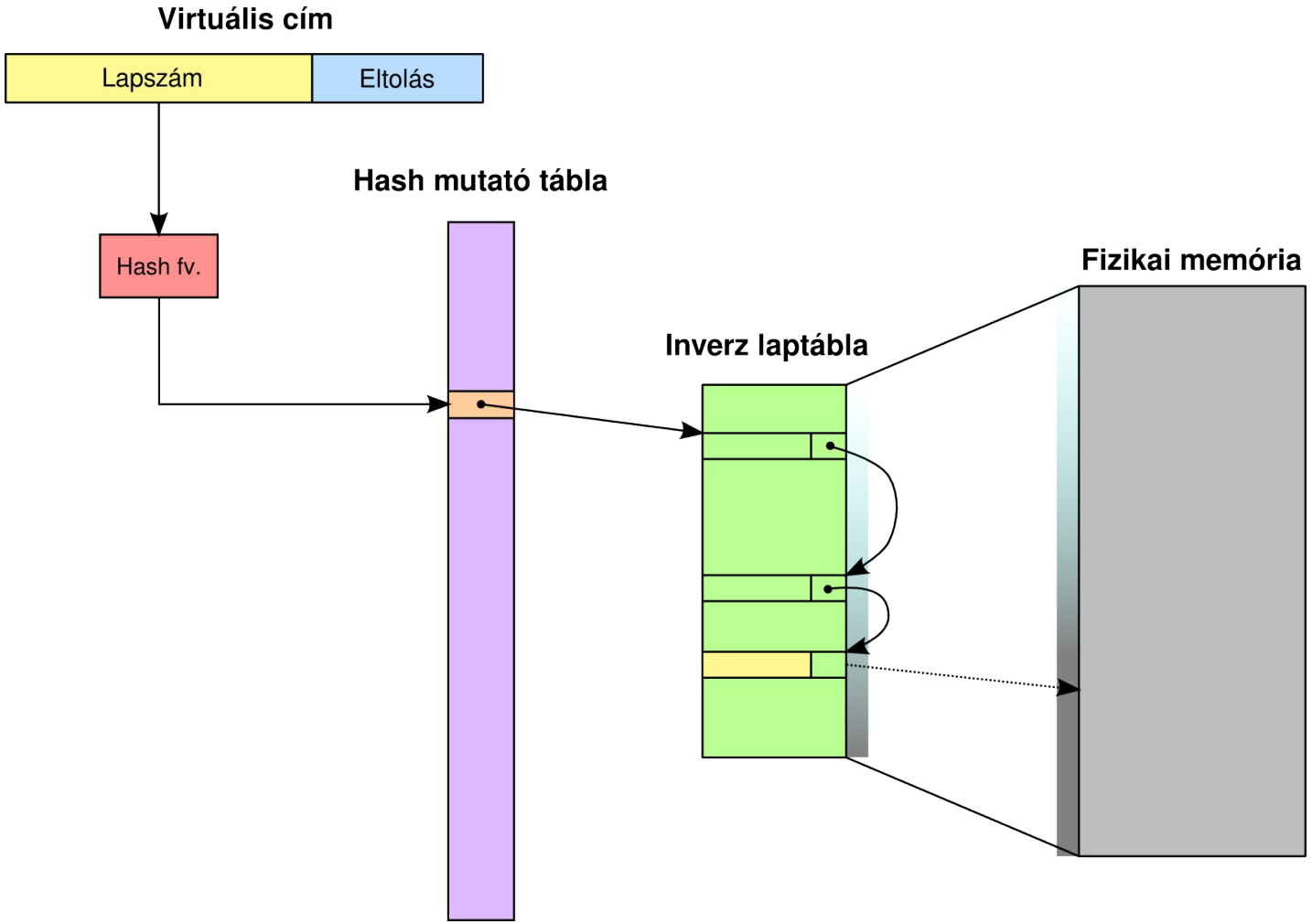
INVERZ LAPTÁBLA

- Miért is voltak nagyok az eddigi laptáblák?
  - Összerendelés: i. bejegyzés = i. laphoz tartozó keret
  - Elemszám = lapok száma
- Fordított gondolkodás:
  - Összerendelés: i. bejegyzés = i. keretben lévő lap száma
  - Elemszám = keretek száma

  → ***Inverz laptábla***
- Mit nyerünk?
  - Kicsi lesz a laptábla (fizikai memória méretével arányos)
- Mit vesztünk?
  - Adott lapsorszámhoz tartozó keretet keressük
  - A tömbünk indexei keretsorszámok, a bejegyzés *tartalma* a lap száma

  → Tartalom szerint kell keresnünk!
- Tartalom szerinti kereséshez: hash függvény!

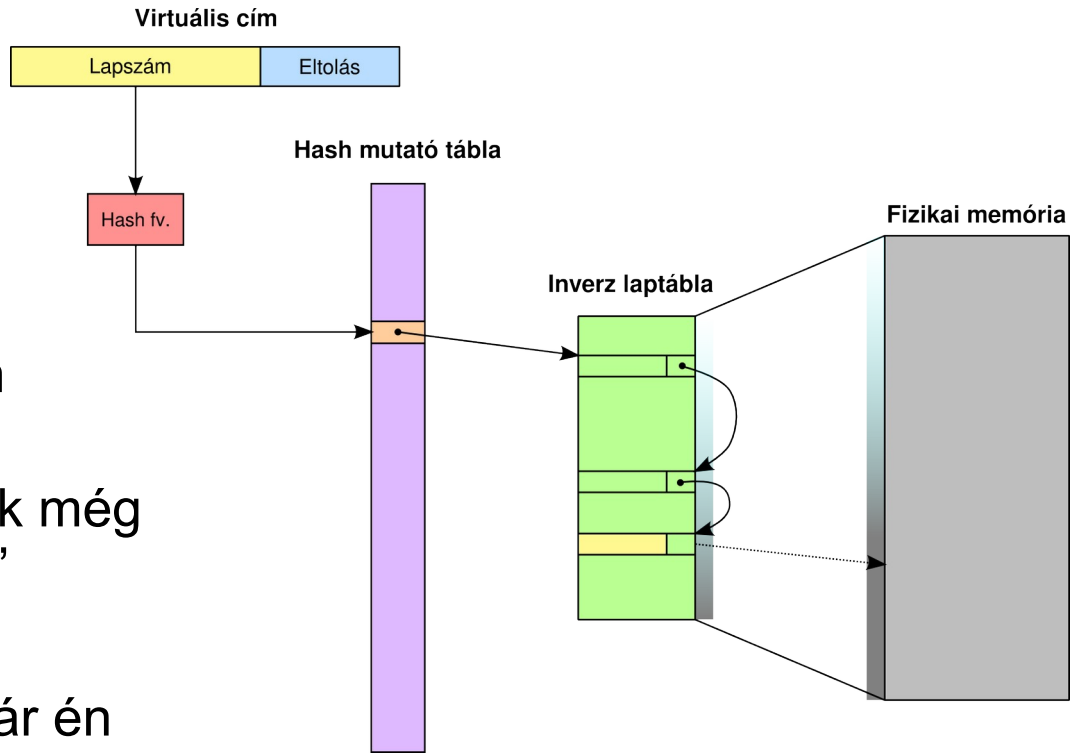
INVERZ LAPTÁBLA



- PI. jó hash függvény: Lapsorszám alsó bitjei

INVERZ LAPTÁBLA

- Hash mutató táblában:
  - Pointer:
    „Hol lehetnek az ilyen lapok a laptáblában?”
- Laptáblában:
  - „Ebbe a bejegyzésbe én vagyok írva?”
  - „Nem? Pech. Hol vannak még hozzám hasonló lapok?”
  - „Követem a pointert”
  - „Ebbe a bejegyzésbe már én vagyok írva?”
  - „Nem. Nem baj, megyek tovább. Talán itt?”
  - „Igen? Szuper. Hányadik bejegyzésen állok? 329?”
  - „Akkor a fizikai memória 329-es keretében vagyok elhelyezve!”



INVERZ LAPTÁBLA

- Mitől függ a memóriaműveletek száma?
  - Először a hash mutató táblát kell kiolvasni
  - + ahány lépést kell tenni a laptábla láncolt listájában
- És az mitől függ?

  → Attól, hogy mennyire van tele a memória
- Normál terhelés (nincs tele a memória)

  → 1 memóriaművelet / címfordítás (+1: hash mutató tábla)
- Nagy terhelés (megtelt a memória)

  → >1 memóriaművelet / címfordítás (+1: hash mutató tábla)

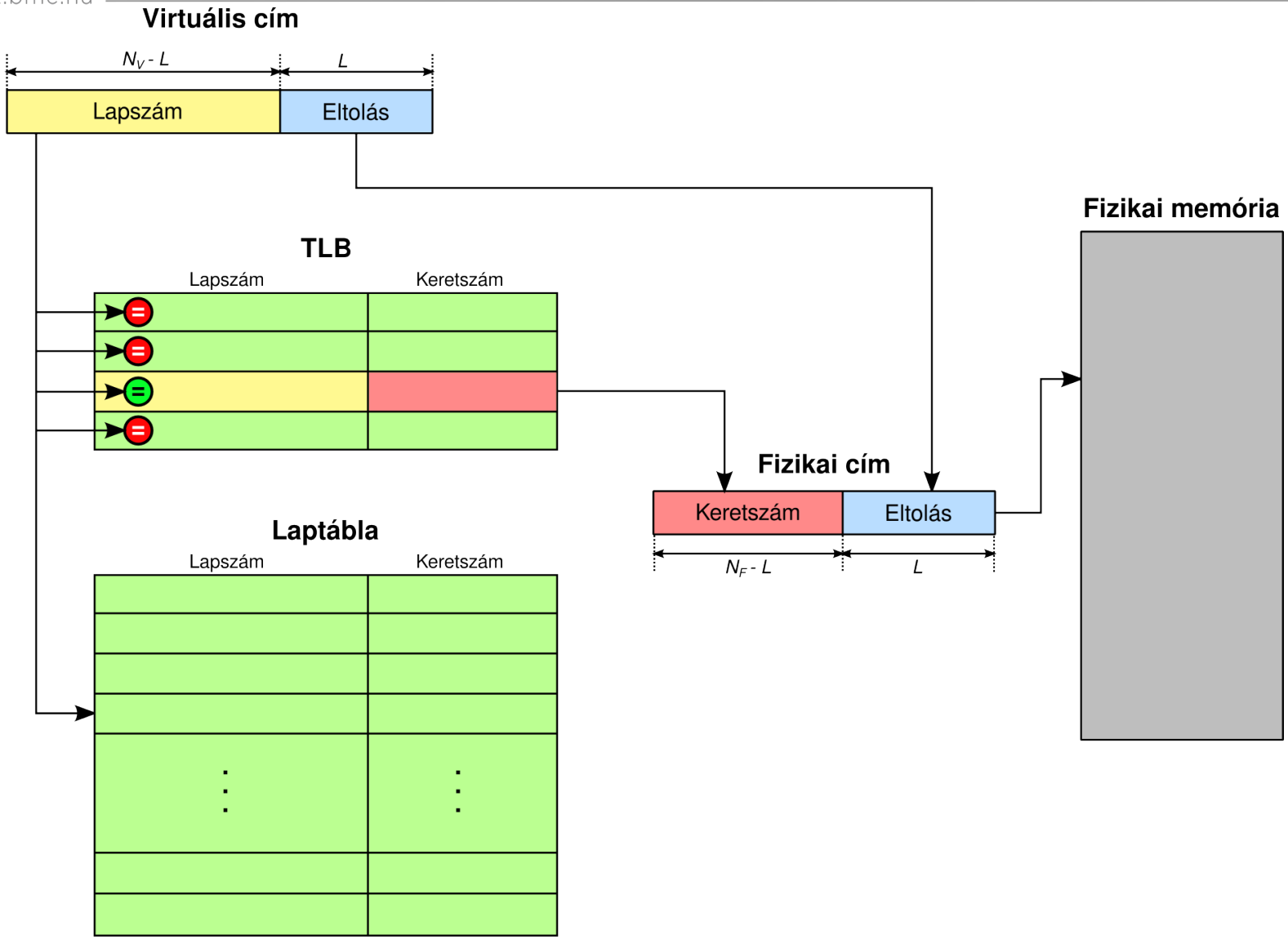
  De akkor már úgyis mindegy (swap-eléstől lassú a rendszer)
- PA-RISC, PowerPC

TLB

TLB

- Memória-hozzáférés menete:
  - Címfordítás: Fizikai cím előállítása a virtuálisból
  - Tényleges adat olvasás/írás a fizikai címről/címre
- A laptábla is a fizikai memóriában van!
- **1 memóriaművelet a programban**
  → ≥ 2 memóriaművelet a valóságban !!
- Nem elég lassú a memória e nélkül is?
- De.
- Reménység: lokalitás
- ***TLB***: Translation Lookaside Buffer
  - A gyakran használt virtuális ↔ fizikai összerendelések cache-e
  - Címfordításkor először a TLB-ben keres

CÍMFORDÍTÁS TLB-VEL



TLB JELLEMZŐI

- **TLB lefedettség**: a TLB bejegyzések által lefedett címtartomány
  - Minél nagyobb, annál ritkábban kell a lassú laptáblához nyúlni
- TLB megvalósítása: **tartalom szerint címezhető memória**

  → Nagy a felülete és sokat fogyaszt

  → Szűkös a tárolási kapacitása :(

TLB méret:	16 – 512 bejegyzés
Találat ideje:	0.5 – 1 órajel
TLB hiba esetén címfordítási idő:	10 – 100 órajel
TLB hibaarány:	0.01% – 1%

HARDVER MENEDZSELT TLB

- Címfordítás: **CPU/MMU**...
  - először a TLB-ből próbálkozik,
  - TLB hiba esetén: ***bejárja a laptáblát***
    - Valid = 1 esetén: TLB update, kész.
    - Valid = 0 esetén: Laphiba! → operációs rendszert hív, majd újra próbálkozik
- Laphiba kezelése: **Az operációs rendszer**...
  - megkapja a keresett lap számát
  - saját nyilvántartása alapján betölti a diszkről
  - ...ha ott van!
    Ha még senki nem használta a lapot, létrehoz egy üreset
  - elhelyezi a fizikai memóriában (esetleg kidob onnan valakit)
  - ***frissíti a laptáblát***
- Példa: x86, x86-64, ARM, POWER

SZOFTVER MENEDZSELT TLB

- Címfordítás: **CPU/MMU**...
  - először a TLB-ből próbálkozik,
  - TLB hiba esetén: ***operációs rendszert hív***, majd újra próbálkozik
- TLB hiba kezelése: **Az operációs rendszer**...
  - megkapja a keresett lap számát
  - bejárja a saját laptábláját:
    - Valid = 1: beírja a CPU TLB-be a lap ↔ keret összerendelést
    - Valid = 0: mint eddig (swap-elés, laptábla frissítés)
  - ***Mindenképp frissíti a TLB-t***
- Előnye:
  - Nincs hardver megkötés, op. rendszer frissítéssel jöhet hatékonyabb laptábla
  - Bonyolultabb adatszerkezetek is használhatók
- Hátránya:
  - Sokkal lassabb címfordítás (TLB hiba esetén)
  - Példa: SPARC, Alpha, MIPS, félig a PA-RISC

MÉRETEZÉSI KÉRDÉSEK

- Mekkora lapokat használjunk?
  - Nagyot, mert
    - Nagyobb a TLB lefedettség, kevesebb a laphiba
    - Diszk háttértár nagyot és kicsit kb. egyforma ideig tölt be
      - Akkor már vigyünk be minél nagyobb darabot
  - Kicsit, mert
    - A lapon csak az van, ami kell.
    - És ami nem kell, ne foglalja már a drága, kicsi memóriát
  - Gyakorlatban: 4 – 8 kB

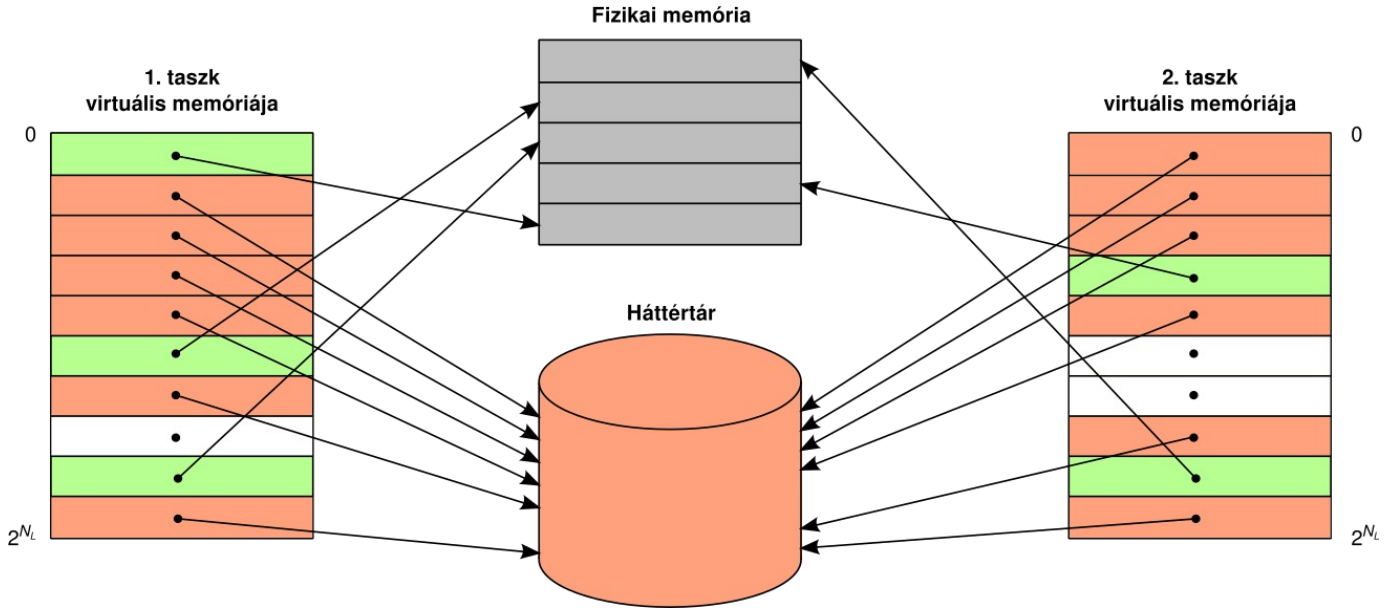
Virtuális memória multitaszking környezetben

CÍMTÉR ELKÜLÖNÍTÉS

- Cél: minden taszk kapja meg a teljes címtartományt (pl. 0 → 4 GB)
- Előnyök:
  - **Állandó futási környezet**
    - Könnyebb szoftverfejlesztés
    - Mert fordításkor ismert
      - a belépési pont
      - a globális változók címe
      - a függvények címe
      - stb.
  - **Védelmet ad**
    - Nem tudja elérni más taszkok címterét!
- Megoldás:
  - Taszkonként külön laptábla

TASZKONKÉNTI LAPTÁBLA

- Minden taszk megkapja a teljes címtartományt
- Megoldás:
  - Taszkonként külön laptábla
  - Taszkváltáskor:
    - Laptábla kezdőpointer lecserélése
    - TLB invalidálás!



- Lehetnek osztott címtartományok, ugyanazokkal a keretekkel
  - Szükséges az op. rendszer meghívásához!

CÍMTÉR AZONOSÍTÓ

- Gond: TLB ürítés taszkváltáskor
  - Míg fel nem telik újra, lassúak a memóriaműveletek
- Megoldás:
  - **Címtér azonosító (ASID)**

    → taszk azonosító
  - TLB bejegyzés mezői: lap, keret, ASID
  - Működés:
    - Op. rendszer taszk váltáskor egy spec. regiszterbe írja az aktuális ASID-ot
    - A CPU csak az erre vonatkozó TLB bejegyzéseket használja
    - A TLB több taszk laptábla bejegyzéseit is tudja tárolni

      → nem kell TLB invalidálás taszkváltáskor

HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK