HÁLÓZATI RENDSZEREK ÉS SZOLGÁLTATÁSOK TANSZÉK

MŰEGYETEM 1782

Budapest,
2025.03.17.

SZÁMÍTÓGÉP ARCHITEKTÚRÁK

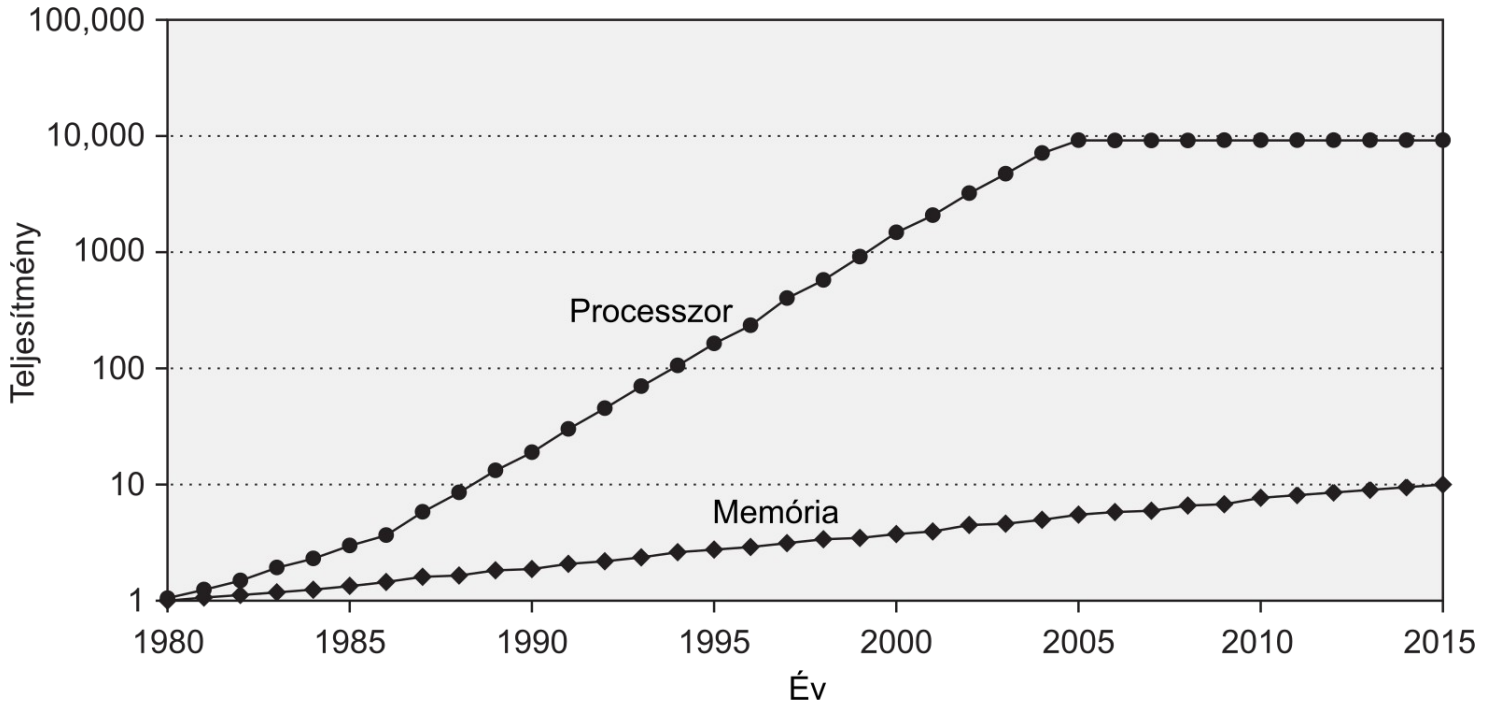
Cache memória

Horváth Gábor, Belső Zoltán

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

MÁR MEGINT A MEMÓRIA...

- Mindenről a memória tehet.
- Mert lassú.



- A virtuális tárkezelés tetézi a bajt
  - 1 memóriabeli objektum elérése → több memóriaművelet

REMÉNYSUGÁR

- A programok nem véletlenszerűen nyúlkálnak a memóriába
- Sokszor speciális mintázatot követnek
  → kihasználható!
- Lokalitási elvek:
  - **Időbeli**: egyszer hozzányúltunk, többször is hozzá fogunk
  - **Térbeli**: ha hozzányúltunk, a környezetéhez is hozzá fogunk
  - **Algoritmikus**: speciális adatszerkezet kiszámítható bejárása
- Példa:
  - Médialejátszás:
    - Térbeli lokalitás: igen, időbeli lokalitás: nem
  - Ciklusszervezés:
    - A ciklusmag kódjára teljesül a térbeli és időbeli lokalitás is

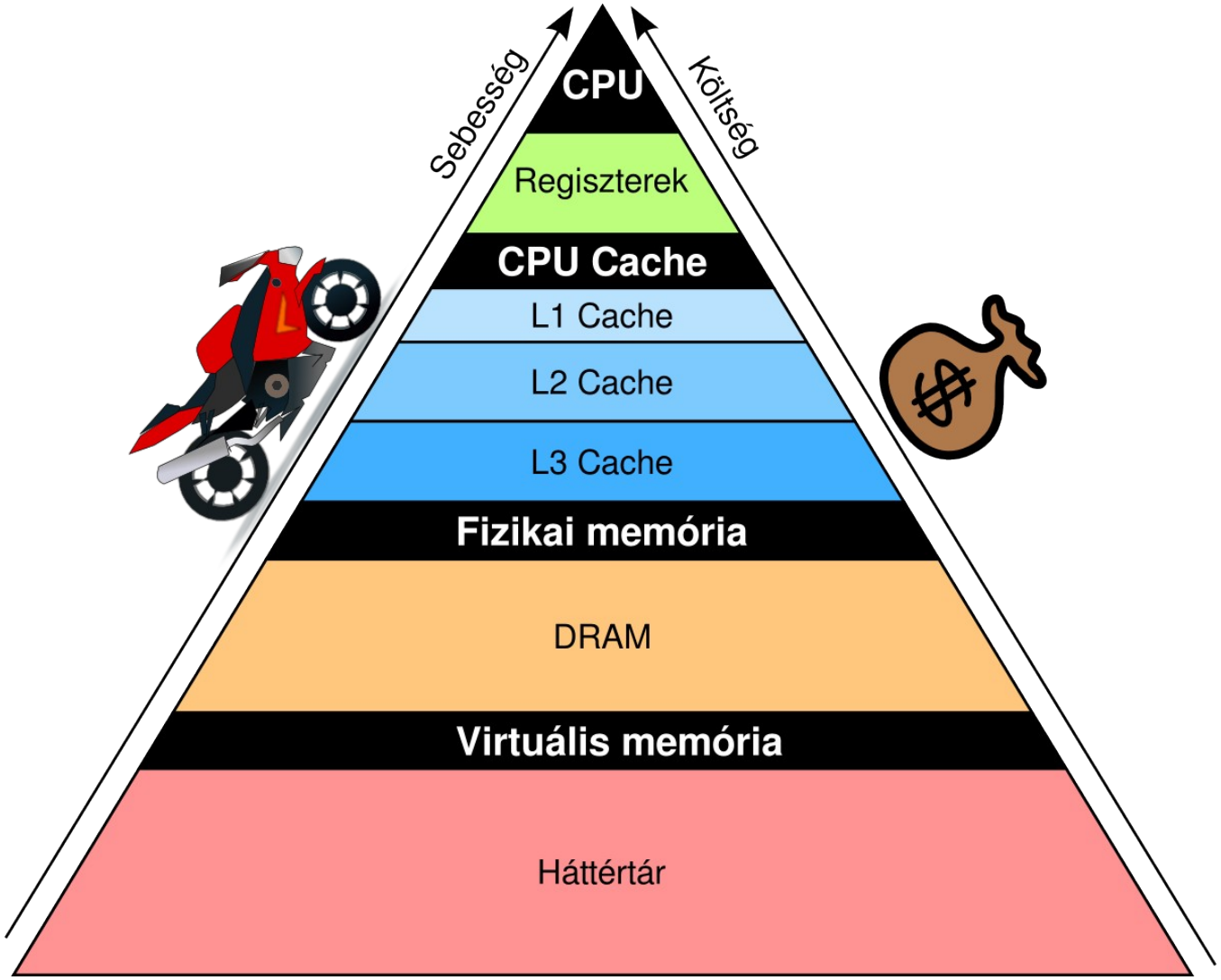
TÁRHIERARCHIA

- Ha van lokalitás:
  - vigyük a gyakran használt adatokat gyorsabb memóriába, közel a CPU-hoz
- Miért, nem minden memória egyformán lassú?
  - Van lassabb is: HDD
  - Az SRAM gyorsabb, mint a DRAM
  - Kisebb memória → nagyobb sebesség (lásd: jelterjedési idő)

Tár típusa	Elérési idő	Ár/GB
SRAM	0.5 – 2.5 ns	$500 – $1000
DRAM	50 – 70 ns	$10 – $50
Flash	5 000 – 50 000 ns	$0.75 – $1.0
HDD	$5 – 20 \cdot 10^6$ ns	$0.05 – $0.1

(2012-as adatok)

TÁRHIERARCHIA



CACHE MEGVALÓSÍTÁSOK

- Címzési mód szerint:
  - Transzparens:
    - A cím tartomány egy részének másolata a gyors memóriában van
  - Nem transzparens:
    - A címtartomány egy része alatt a gyors memória van
- Menedzsment szerint:
  - Implicit menedzsment:
    - A gyors memória tartalmát a hardver kezeli
  - Explicit menedzsment:
    - A gyors memória tartalmát a futó alkalmazás kezeli

CACHE MEGVALÓSÍTÁSOK

	Címzési mód	Menedzsment
Transzparens cache	Transzparens	Implicit
Szoftver-menedzselt cache	Transzparens	Explicit
Önszervező scratch-pad	Nem transzparens	Implicit
Scratch-pad memória	Nem transzparens	Explicit

- **Transzparens cache: klasszikus CPU cache**
- Scratch-pad memória: DSP-k, mikrokontrollerek, PlayStation 3
- Szoftver-menedzselt cache:
  - Nincs cache találat → szoftver feladata a cache update
- Önszervező scratch-pad
  - Egzotikus megoldás

CACHE MEGVALÓSÍTÁSOK

- Amiről ma tanulunk:
  - **Cache szervezés:**
    - Adatok hatékony tárolása a cache-ben
  - **Cache tartalom menedzsment:**
    - Mikor tegyünk a cache-be egy adatot
    - Ki tegyünk ki onnan, ha tele van

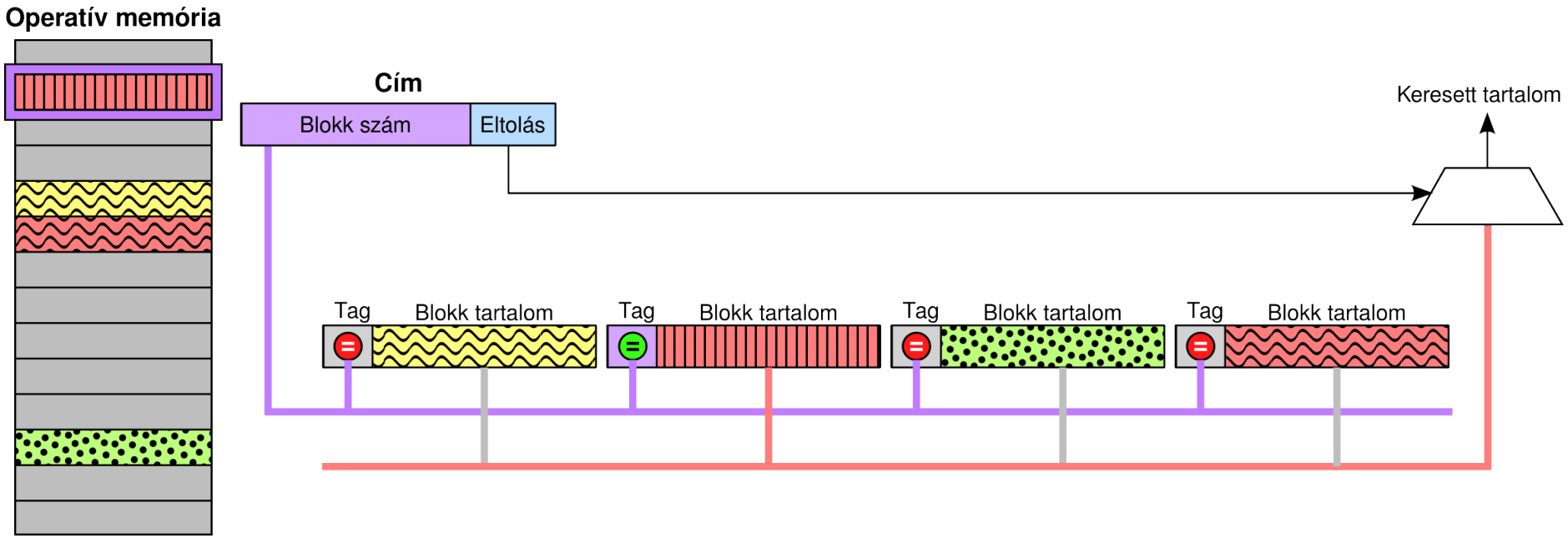
Cache szervezés

TRANSZPARENS CACHE SZERVEZÉS

- Tárolási egység: ***blokk*** (= cache line)
  - Blokkok mérete = 2^L
  - Címek alsó L bitje: blokkon belüli eltolás
  - Felső bitek: blokk sorszáma
- A cache minden blokkja mellé tárolt járulékos információ:
  - Cache ***tag*** (az op. memória hányas blokkja ez?)
  - ***Valid*** bit: ha =1, ez a cache blokk érvényes
  - ***Dirty*** bit: ha =1, erre a cache blokkra történt írás, mióta itt van
- A cache szervezés alap kérdése:
  - Hogy tároljuk a blokkokat a cache-ben
    - Hogy gyorsan kereshető legyen
    - Hogy egyszerű (gyors és olcsó) legyen

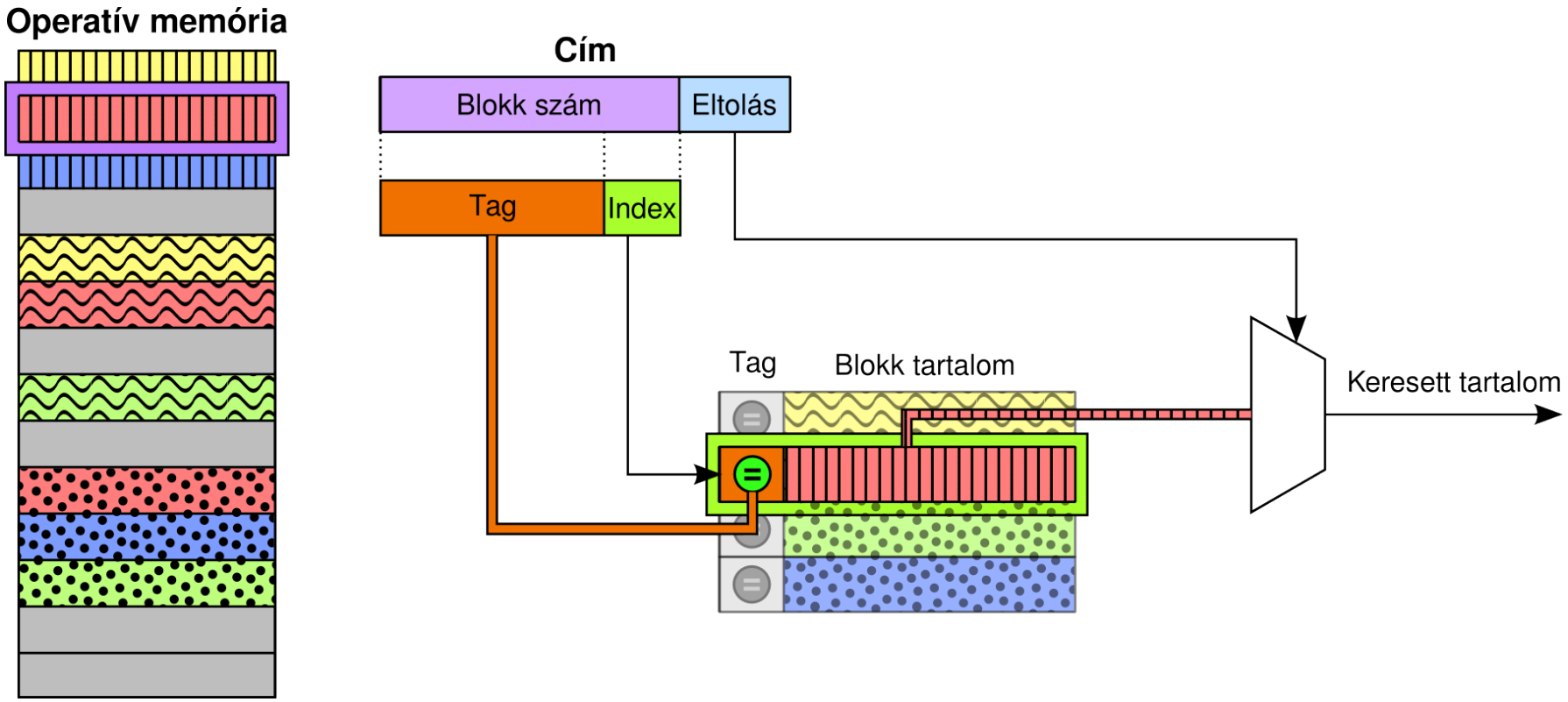
TELJESEN ASSZOCIATÍV SZERVEZÉS

- A blokkok a cache-ben bárhová elhelyezhetők
- Cache tag: ez a blokk az operatív memória hányas blokkja
- Sokat fogyaszt:
  - Keresés: cím blokk száma és az ***összes*** cache tag komparálása
  - Komparátorok szélessége: blokkszám bitszélessége



DIREKT LEKÉPZÉS

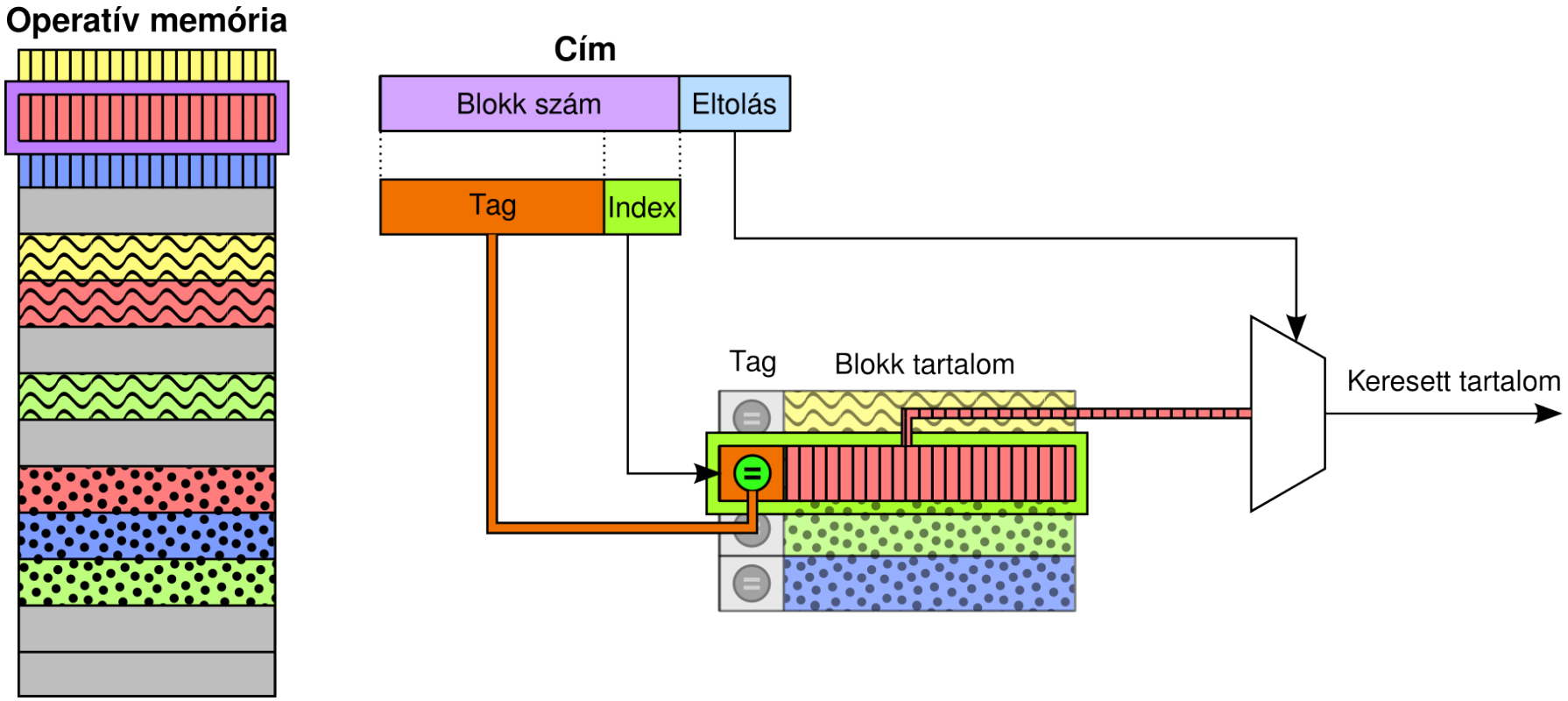
- Minden memóriabeli blokk csak egyetlen helyre kerülhet a cache-ben
- Hogy hova, azt a memóriabeli blokkszám egyértelműen eldönti
  - Pl. a blokkszám alsó bitjei alapján



DIREKT LEKÉPZÉS

- Pl.: cache 4 blokkot tud tárolni: 1 sárgát, 1 pirosat, 1 kéket, 1 zöldet
- A memóriában a blokkok ebben a sorrendben követik egymást

  → szín = blokkszám alsó 2 bitje



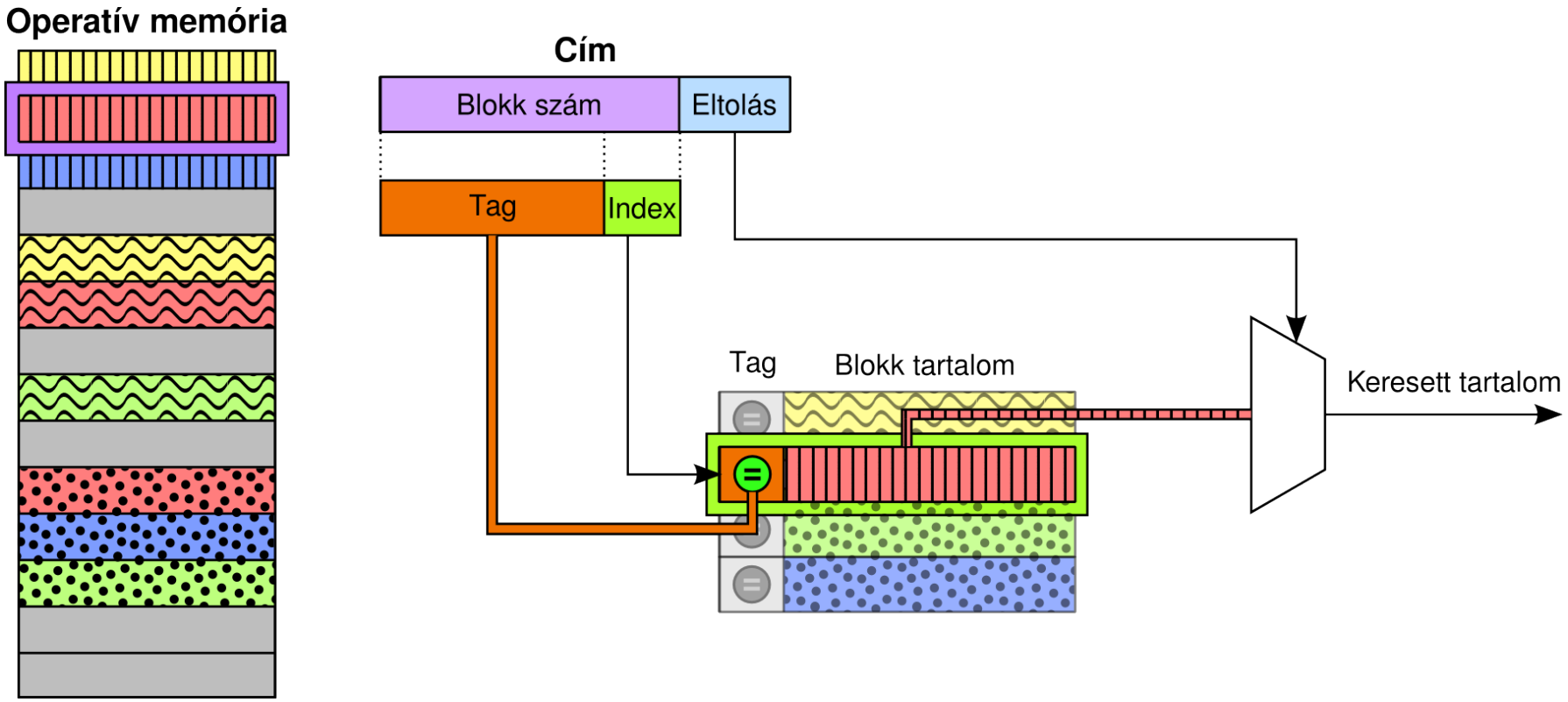
DIREKT LEKÉPZÉS

- Keresés:

  1) Indexelés: A cím blokkszámából a szín tudható: piros (ez az ***index***)

  2) Komparálás: A cache piros rekeszében ez a blokk van?

  - Egyetlen komparátor megy, és az is keskenyebb! (***Tag*** hosszú)



ÉRTÉKELÉS

- Teljesen asszociatív:
  - Szabad blokk elhelyezés
  - Sok, széles komparátor működik benne

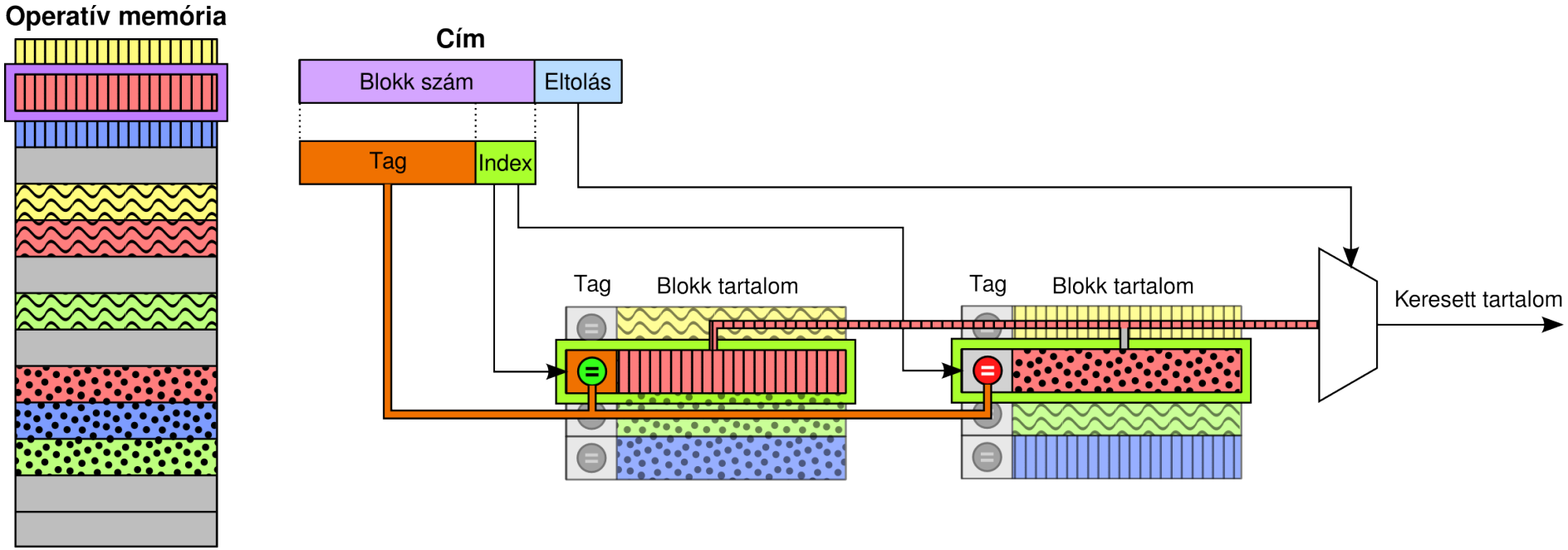
    → komplexitás, fogyasztás
- Direkt leképzés:
  - Korlátozott blokk elhelyezés:

    → Versenyhelyzet: szuboptimális működés

    Pl. a program csak piros blokkokkal dolgozik
  - Csak 1, keskenyebb komparátor dolgozik

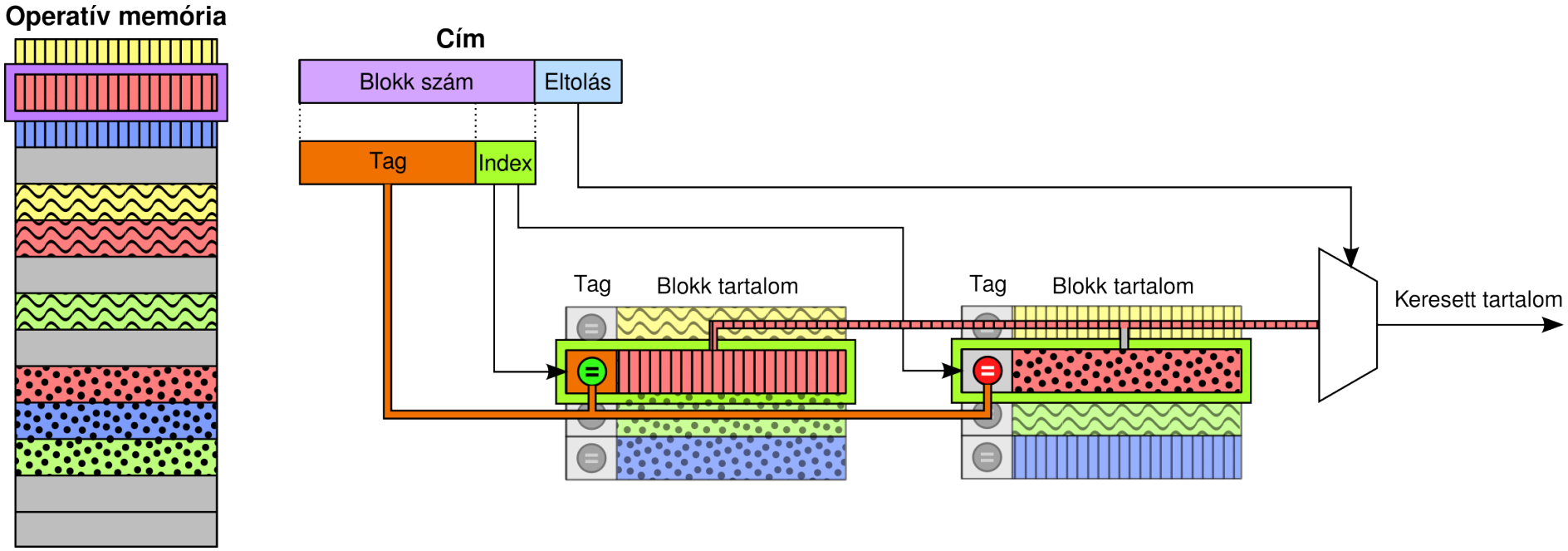
N-UTAS ASSZOCIATÍV SZERVEZÉS

- Kombinálja a korábbi két megoldást
- A blokkszám alsó bitjei meghatározzák a blokk helyét
  - De nem egyértelműen!
  - Az alsó bitek kijelölnek egy ***halmazt***, ahol a blokk lehet
  - A halmaz n cache blokkból áll, ezek bármelyikében lehet a keresett blokk



N-UTAS ASSZOCIATÍV SZERVEZÉS

- Keresés:
  1) **Indexelés**: A cím blokkszámából a szín tudható: piros (ez az ***index***)
  2) **Komparálás**: A cache n piros rekesze közül melyikben van ez a blokk?
     - n komparátor megy, keskenyebb szélességgel (***Tag*** hosszú)



ÉRTÉKELÉS

- n-utas asszociatív szervezés:
  - Korlátozott blokk elhelyezés, n lehetőséggel
    → ritkábban van versenyhelyzet
  - Csak n komparátor dolgozik
    → moderált komplexitás és fogyasztás
- Tipikus n értékek:
  n=2...16

	Core i7 (Kaby Lake)		AMD Ryzen		ARM Cortex A53		ARM Cortex A75	
	n=	méret	n=	méret	n=	méret	n=	méret
L1	8	32 KB	4/8	64/32 KB	2/4	16-64 KB	4	64 KB
L2	4	256 KB	8	512 KB	16	128-2048 KB	8	256-512 KB
L3	16	2 MB/mag	16	2 MB/mag	-	-	16	1-4 MB

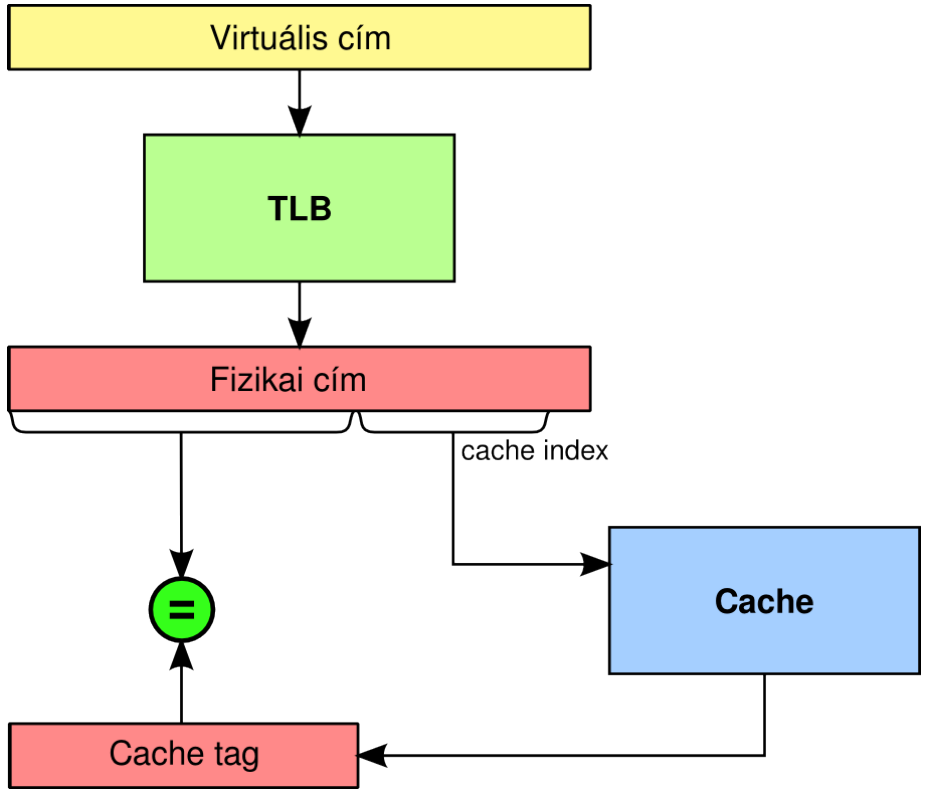
Cache és a virtuális memória

CACHE + VIRTUÁLIS MEMÓRIA

- Mit tároljon a cache?
  - Fizikai memória blokkjait?
  - Virtuális memória blokkjait?
- Természetes választás:
  - Fizikai memória blokkjait!
    - Mert kisebb → kevesebb blokk van benne → a cache tag és a komparátorok keskenyebbek lehetnek
    - Mert a tárhierarchiában a fizikai memória van alatta

FIZIKAILAG INDEXELT CACHE FIZIKAI TAG-EL

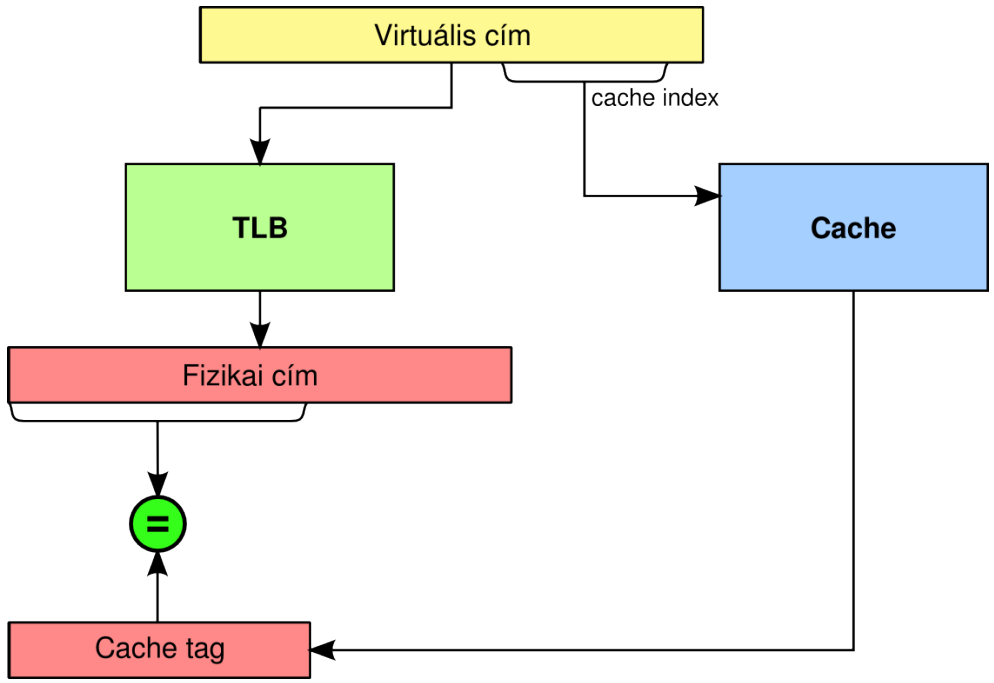
- Minden memóriaművelet:
  1) Címfordítás
  2) Cache indexelés
     (halmaz kiválasztása)
  3) Komparálás
     (halmazon belül hol lehet)



- Ez 3 lépés.
- Nem lehetne gyorsabban?

VIRTUÁLISAN INDEXELT CACHE FIZIKAI TAG-EL

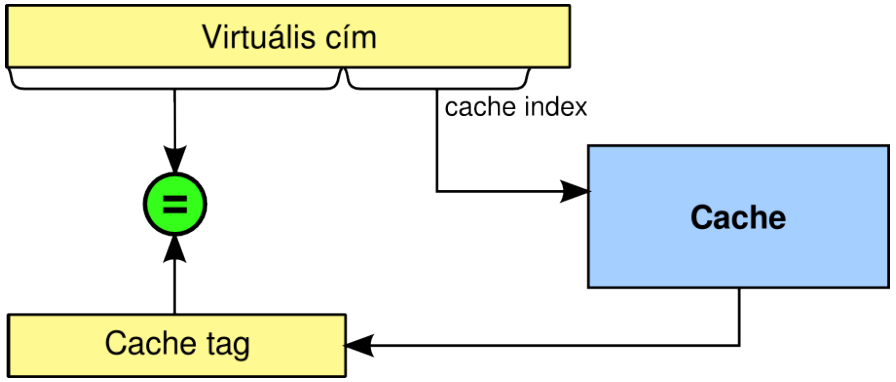
- Cache indexelés (halmaz kiválasztás) a virtuális címmel!
- Minden memóriaművelet:
  1) Egyidejűleg:
     - Címfordítás
     - Cache indexelés (halmaz kiválasztása)
  2) Komparálás (halmazon belül hol lehet)



- Már csak 2 lépés.
- Nem lehetne még egyszerűbben?

VIRTUÁLISAN INDEXELT CACHE VIRTUÁLIS TAG-EL

- Az egész cache virtuális címekre épül
- Minden memóriaművelet:
  1) Cache indexelés
     (halmaz kiválasztása)
  2) Komparálás
     (halmazon belül hol lehet)



- Címfordítás csak cache hiba esetén kell!
- Hátrány:
  - Szélesebb a tag. (Mert a virtuális cím szélesebb, kivéve PAE)
  - Ha egy keretet több lappal lehet elérni, többször lesz bent a cache-ben

Cache tartalom menedzsment

TARTALOM MENEDZSMENT

- Már tudjuk, milyen megoldások vannak
  - Az adatok tárolására:
    - Teljesen asszociatív: hatékony, de drága
    - Direkt leképzés: egyszerű, de nem olyan hatékony
    - N-utas asszociatív: arany középút
  - A virtuális és fizikai címek használatára:
    - Érdemes a virtuális címmel indexelni
    - Jó ötlet lehet a virtuális címből képezni a tag-et
- Jön: hogy menedzseljük a cache tartalmát?
  - **Kit tegyünk bele?**
  - **Mikor tegyük bele?**
  - **Kit rakjunk ki?**
- Implicit / explicit

ADATOK BETÖLTÉSE A CACHE-BE

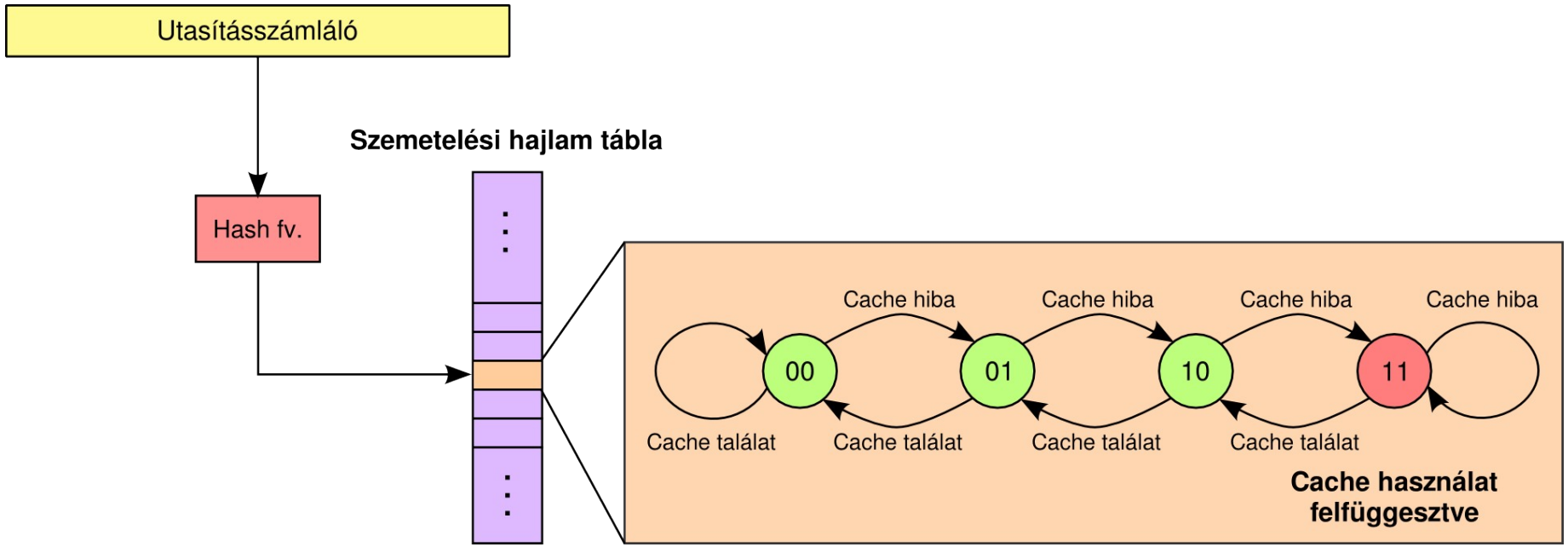
- Mikor tegyünk be egy memória blokkot a cache-be?
  - Soha
  - Amikor a futó program először hivatkozik rá
  - Jó előre, mielőtt még használnánk, jól jön az még
- **Soha**: cache szemetelés ellen véd
  - pl. médialejátszás: kijátszik egy képet, soha többet nem kell.
    → nem érdemes cache-be betölteni a képet
- **Első hivatkozáskor**: elsőre sokáig tart (memória → cache átvitel), de később gyors lesz
- **Idő előtti betöltés** (prefetch): spekulációt igényel

CACHE SZEMETELÉS MEGELŐZÉSE

- Cache szemét: az a blokk, amire a bekerülés és a kirakás között nem volt hivatkozás
  - Kár volt behozni, kiszoríthatott hasznos blokkokat
- Explicit megoldás: spéci hardver utasítások:
  - Adatmozgatás a cache megkerülésével
  - Szinte minden architektúrán:
    - **x86**: MOVNTI, MOVNTQ
      - SSE regiszterből memóriába, a cache kikerülésével
    - **PowerPC**: LVXL memóriából vektor regiszterbe olvas.
      - A blokk cache-be kerül, de megjelölik, ezt dobják ki elsőnek, ha kell a hely. (LVX: ugyanez, jelölés nélkül)
    - **PA-RISC**: Sok utasítás kódjában van egy bit:
      - Spatial Locality Cache Control Hint
    - **Itanium**: Adatmozgatásnál „.nt” opció
      - jelezhető, hogy az adat nem felel meg az időbeni lokalitásnak
    - **Alpha**: ECB utasítás (Evict Data Cache Block)
      - Jelezhető, hogy a blokk nem lesz mostanában hivatkozva

CACHE SZEMETELÉS MEGELŐZÉSE

- Implicit megoldások: a CPU próbálja detektálni a szemetelő utasításokat
- Ezernyi algoritmus. Példa: Rivers' algoritmus
- Ha egy utasítás sok cache hibát okoz → CPU eltiltja a cache-től



IDŐ ELŐTTI BETÖLTÉS (PREFETCH)

- Kulcsfontosságú funkció
- Cél: a CPU működése ne akadjon meg cache hiba miatt
- Ezért már be kell hozni minden adatot, mielőtt még először meghivatkoznák
- ***Utasítás cache*** esetén egyszerű:
  - Az nem ugró utasítások sorban követik egymást
    → kiszámítható
  - A feltétel nélküli ugrások ugrási címe ismert
    → kiszámítható
  - A feltételes ugrások nagyon jól becsülhetők
    → egész jól kiszámítható

IDŐ ELŐTTI BETÖLTÉS (PREFETCH)

- ***Adat-cache*** esetén nehéz:
  - Az adatok hivatkozásának mintázata nehezen kiszámítható
  - Ha a becslés:
    - Túl óvatos: nem lesz bent az adat, mikorra kell
      → sok cache hiba
    - Túl agresszív: feleslegesen hoz be szükségtelen blokkokat
      → kiszorít gyakran használtakat → sok cache hiba
  - Explicit támogatás:
    - Speciális utasításokkal
  - Implicit támogatás:
    - CPU spekulál

IDŐ ELŐTTI BETÖLTÉS (PREFETCH)

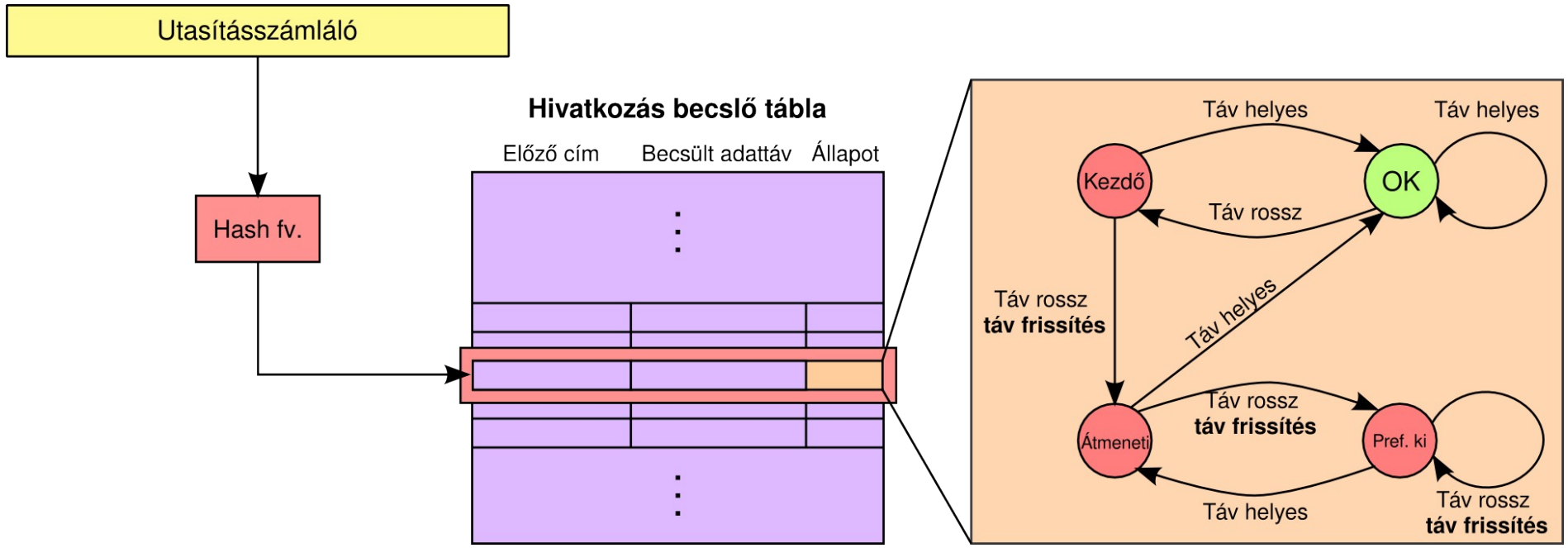
- Explicit prefetch utasítások:
  - **x86**: PREFETCHT0/1/2, PREFETCHNTA
    - behoznak egy blokkot a cache-be
    - PREFETCHNTA: a több utas cache első blokkjába teszi a behozott adatot → elkerülhető a szemetelés is!
  - **PowerPC**: DCBT, DCBTST
    - egy blokk cache-be töltése olvasásra/írásra
  - **PA-RISC**: LDD, LDW
    - egy blokk cache-be töltése írásra/olvasásra
  - **Itanium**: lfetch
    - utasítás egy blokk cache-be töltésére
  - **Alpha**: Ha egy normál adatmozgató utasításnál célként az R31 regiszter van megjelölve, akkor azt a processzor egy idő előtti betöltés kérésnek értelmezi

- GCC fordító platformfüggetlen megoldása:

```
__builtin_prefetch (mutató);
```

IDŐ ELŐTTI BETÖLTÉS (PREFETCH)

- Implicit algoritmus:
  - Fix távolságra lévő adatok lineáris bejárására:
    - X, X + táv, X + 2*táv, X + 3*táv, ...
  - Automatikusan detektálja az egymást követő címek távolságát
- = "Intel Smart Memory Access IP Prefetcher" a Core i7-ben



CSERESTRATÉGIA

- Eldöntöttük, kit akarunk behozni, és mikor
- De hova tegyük?

  → a cache üzemi állapota az, hogy tele van
- Valakit ki kell dobni.
- Lehetséges jelöltek száma = a cache asszociativitása
  - Direkt leképzés esetén nincs választási lehetőség:

    Az új blokk csak egy helyre kerülhet. Aki ott volt, repül.
- Lehetséges algoritmusok:
  - Véletlen választás
  - Körbenforgó
  - **Legrégebben használt (LRU)**
  - Nem a legutóbb használt
  - Legritkábban hivatkozott

ÍRÁSI MŰVELETEK

- Miért kell csínján bánni az írással?
  - Memória osztott erőforrás
    - Több processzorra közös
    - Perifériák is használják (DMA)
  - Ha módosítunk egy cache-blokkot, a memória tartalma nem lesz naprakész!
- Stratégiák a memóriatartalom frissítésére:
  - **Write-through**
    → cache-beli írást rögtön átvezeti a memóriába
  - **Write-back**
    → csak akkor vezeti át, amikor kikerül a cache-ből
- A cache gyors, a memória lassú. Bírja a tempót a sok visszaírással?
  - Az erre szánt blokk egy **write-buffer**-be kerül
  - Ahogy bírja (lassan), írja a memóriába
  - Ha valakinek kell valami, először itt kell keresni

TÖBBSZINTŰ CACHE MEMÓRIÁK

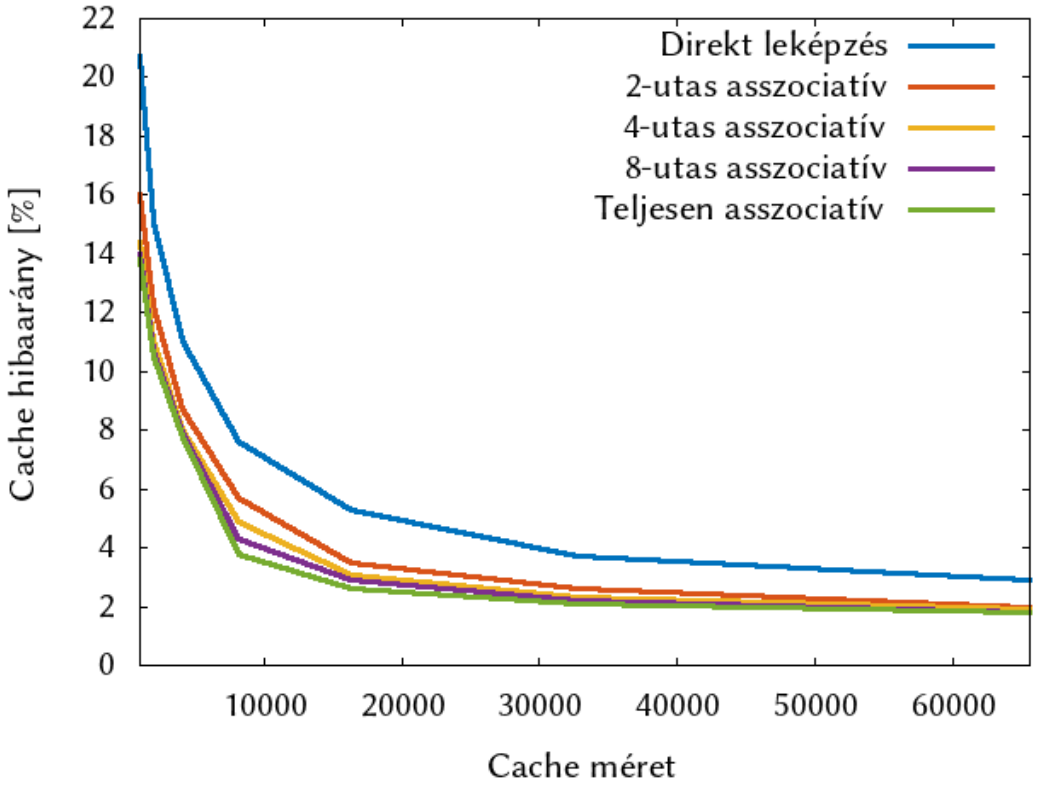
- Kisebb cache → kisebb késleltetés
- Több szintet használnak
  - Méret: nő
  - Sebesség: csökken
  - Ha nincs L1 találat, L2-t nézi, majd L3-at, stb.
  - Más lehet a blokkméret, szervezés, menedzsment, stb.
- Más a cél:
  - **L1 cache célja: minél kisebb késleltetés**
    - Kis méret, alacsony asszociativitás
  - **Ln (n>1) cache célja: cache hiba-arány minimalizálása**
    - Nagyobb méret, magas asszociativitás

Cache memória a gyakorlatban

CACHE SZERVEZÉSEK ÖSSZEHASONLÍTÁSA

- A mérés tárgya: `gcc -o hellow hellow.c` cache viselkedése
- Eszköz: valgrind csomag cachegrind eszköze

```
valgrind --tool=cachegrind --D1=4096,2,64 gcc -o hellow hellow.c
```



GYAKORLATI CACHE PARAMÉTEREK

- Szervezések:

	L1 cache	L2 cache	L3 cache
Raspberry Pi	(16kB; 4; 32)	(128kB; 4; 32)	–
RK3188	(32kB; 4; 32)	(512kB; ?; 32)	–
Pentium 4	(8kB; 4; 64)	(512kB; 8; 128)	–
Core i7-2600	(32kB; 8; 64)	(256kB; 8; 64)	(8MB; 16; 64)

- Menedzsment:

	Címek	Írás	Blokkcsere
Raspberry Pi	Virt. ind, fiz. tag	WB/WT	Random/Round robin
RK3188	Ut: VF / Adat: FF	WB/WT	Random/Round robin
Pentium 4	Virt. ind, fiz. tag	WB/WT	Pszeudo-LRU
Core i7-2600	Virt. ind, fiz. tag	WB	Pszeudo-LRU

HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK