RAČUNALNIŠKA ARHITEKTURA

9 Pomnilniška hierarhija

9 Pomnilniška hierarhija - cilji:

- Osnovno razumevanje :
  - Lokalnosti pomnilniških dostopov
  - Pomena in delovanja pomnilniške hierarhije
- Razumevanje predpomnilnikov:
  - Vpliva na hitrost računanja
  - Podmnožica vsebine gl. pomnilnika
- Razumevanje navideznega pomnilnika

9 Pomnilniška hierarhija

- Delovanje pomnilniške hierarhije
  - Štirinivojska pomnilniška hierarhija – povprečni čas dostopa kot ga vidi CPE
  - Primer: Vpliv verjetnosti zgrešitve v glavnem pomnilniku na povprečni dostopni čas pri trinivojski pomnilniški hierarhiji

9.1 Lokalnost pomnilniških dostopov

- Princip lokalnosti pomnilniških dostopov je eden najpomembnejših pojavov, ki ga opazimo pri delovanju von Neumannovega računalnika.

- Programi bolj pogosto uporabljajo ukaze in operande, ki so v pomnilniku blizu trenutno uporabljanim.

- Programi pogosto več kot enkrat uporabijo iste ukaze in operande.

- Tipičen program 90% časa uporablja samo 10% ukazov.
- Prostorska lokalnost
- Časovna lokalnost
- Lokalnost pomnilniških dostopov omogoča, da glavni pomnilnik nadomestimo s pomnilniško hierarhijo

9.2 Pomnilniška hierarhija

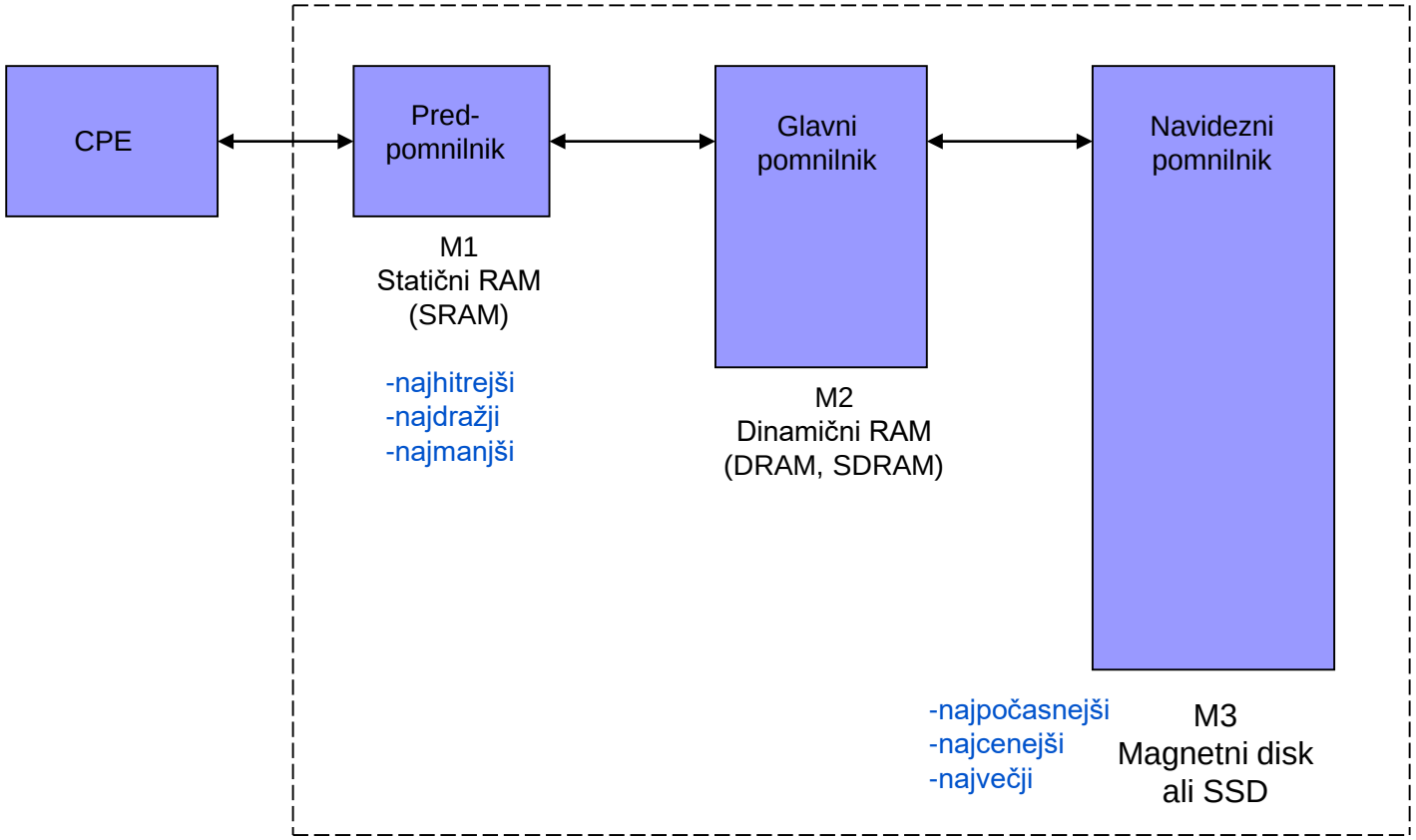
- Želja programerjev:
  - čim hitrejši in
  - čim večji pomnilnik

- Pomnilniška hierarhija, ki jo sestavlja več ločenih pomnilnikov z različnimi lastnostmi, omogoča realizacijo te iluzije:
  - Predpomnilnik (lahko več nivojev predpomnilnikov)
  - Glavni pomnilnik
  - Navidezni pomnilnik

- Uspešno delovanje pomnilniške hierarhije je možno zaradi že omenjene lokalnosti pomnilniških dostopov.

- Pomnilniško hierarhijo torej sestavlja več ločenih pomnilnikov z različnimi lastnostmi:
  - Prvi v hierarhiji pomnilnik M1 (najbližji CPE) je najhitrejši, najdražji in najmanjši.
  - Zadnji v hierarhiji pomnilnik Mn (najbolj oddaljen od CPE) je najcenejši, največji in najpočasnejši.
- Cilj pomnilniške hierarhije je, da je velik, počasen in cenen pomnilnik Mn videti kot hiter in drag pomnilnik M1.

Primer trinivojske pomnilniške hierarhije



CPE vidi pomnilniško hierarhijo kot glavni pomnilnik kot je definiran v von Neumannovem modelu

- Pravilo delovanja hierarhije je, da je pomnilniški prostor na nivoju i podmnožica prostora na nivoju i+1.

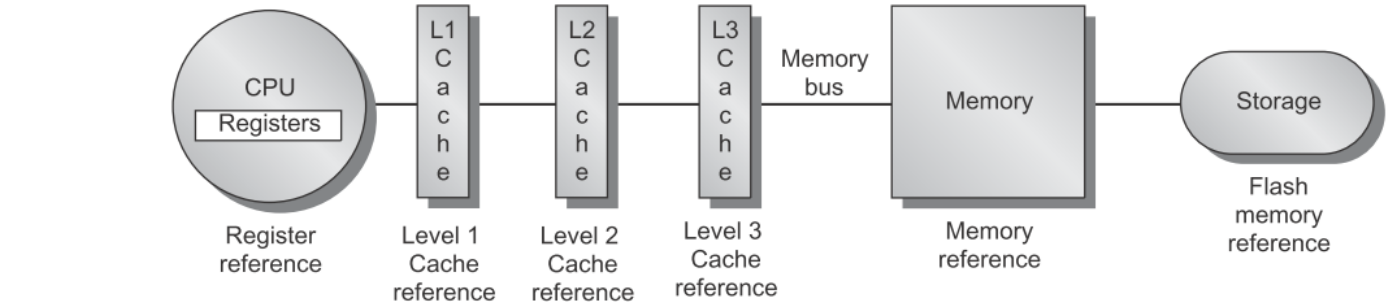
- Če informacije do katere želi CPE ni v M1, se mora prenesti iz M2 v M1. Če je ni tudi v M2, se najprej prenese iz M3 v M2 in nato iz M2 v M1.

- Prenašanje iz enega nivoja na sosednji nivo se izvaja avtomatsko, brez sodelovanja programerja.

- Za CPE je trinivojska pomnilniška hierarhija videti kot glavni pomnilnik velikosti M3, s hitrostjo ki je blizu hitrosti M1.

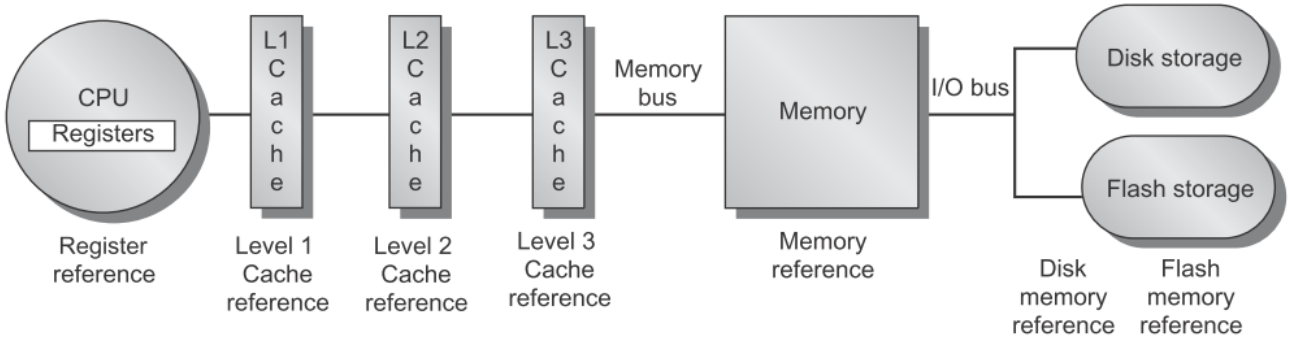
- Pomnilniška hierarhija bi bila brez lokalnosti pomnilniških dostopov neuporabna.

Pomnilniške tehnologije v pomnilniški hierarhiji [Patt]



Laptop	Size:	1000 bytes	64 KB	256 KB	4-8 MB	4–16 GB	256 GB-1 TB
	Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	50-100 uS
Desktop	Size:	2000 bytes	64 KB	256 KB	8-32 MB	8–64 GB	256 GB-2 TB
	Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	50-100 uS

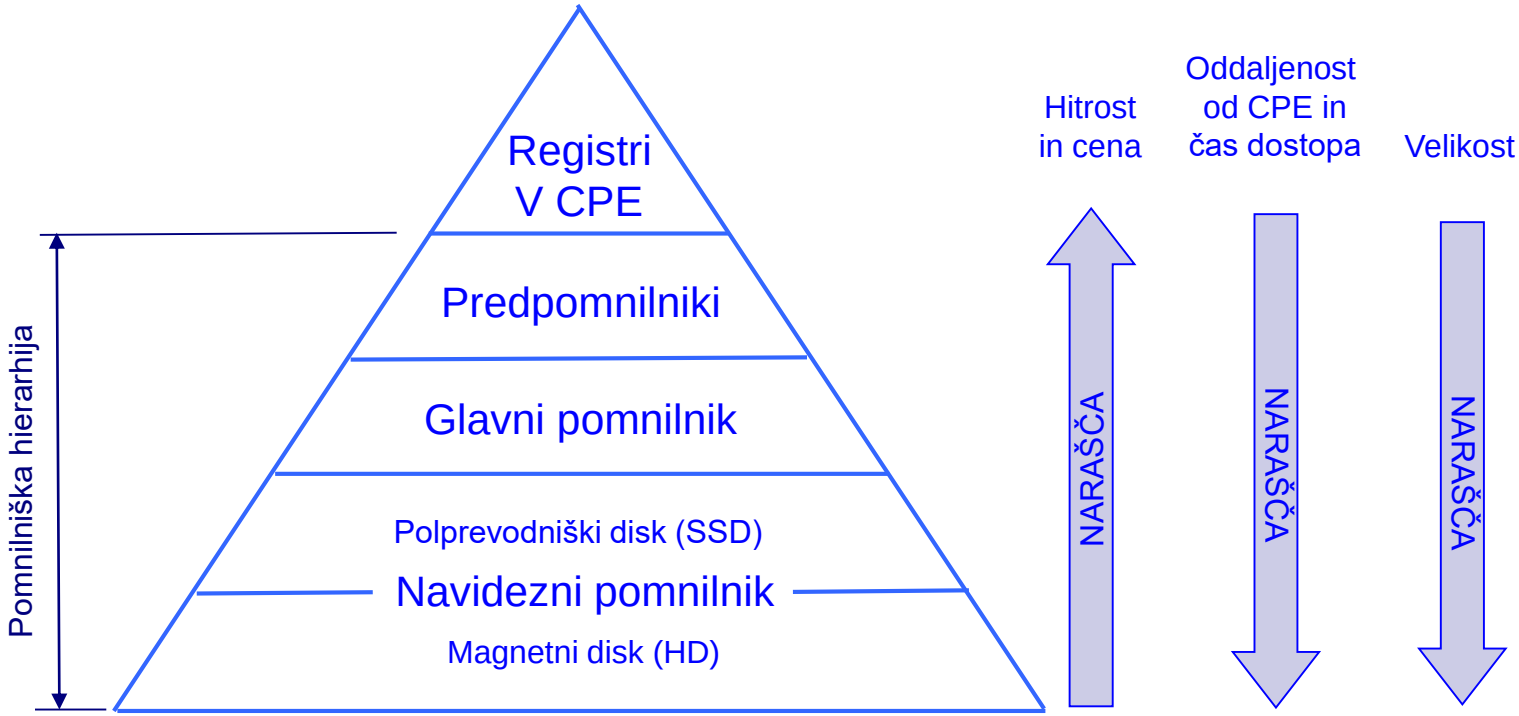
(B) Memory hierarchy for a laptop or a desktop



Size:	4000 bytes	64 KB	256 KB	16-64 MB	32–256 GB	16–64 TB	1-16 TB
Speed:	200 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	5–10 ms	100-200 us

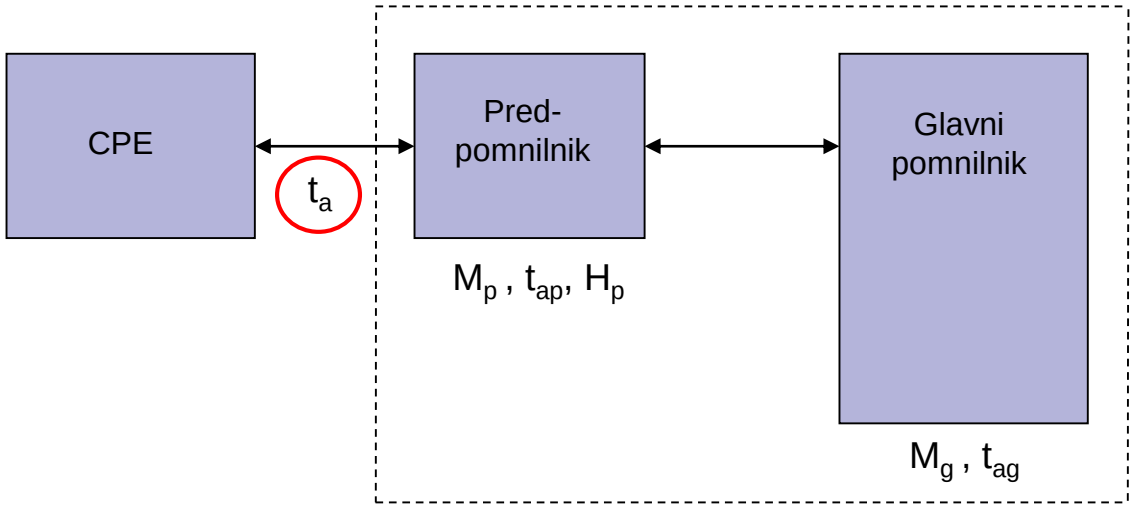
(C) Memory hierarchy for server

Pomnilniki, ki sestavljajo pomnilniško hierarhijo



9.3 Predpomnilnik

- Predpomnilnik (cache) je majhen hiter pomnilnik (SRAM) med CPE in glavnim pomnilnikom.
- Uporaba predpomnilnika v pomnilniški hierarhiji ustvari iluzijo hitrega pomnilnika, ki je hitrejši kot glavni pomnilnik.
- Vsebina predpomnilnika: podmnožica vsebine glavnega pomnilnika.
- CPE s pomnilniškim naslovom vedno dostopa do predpomnilnika.



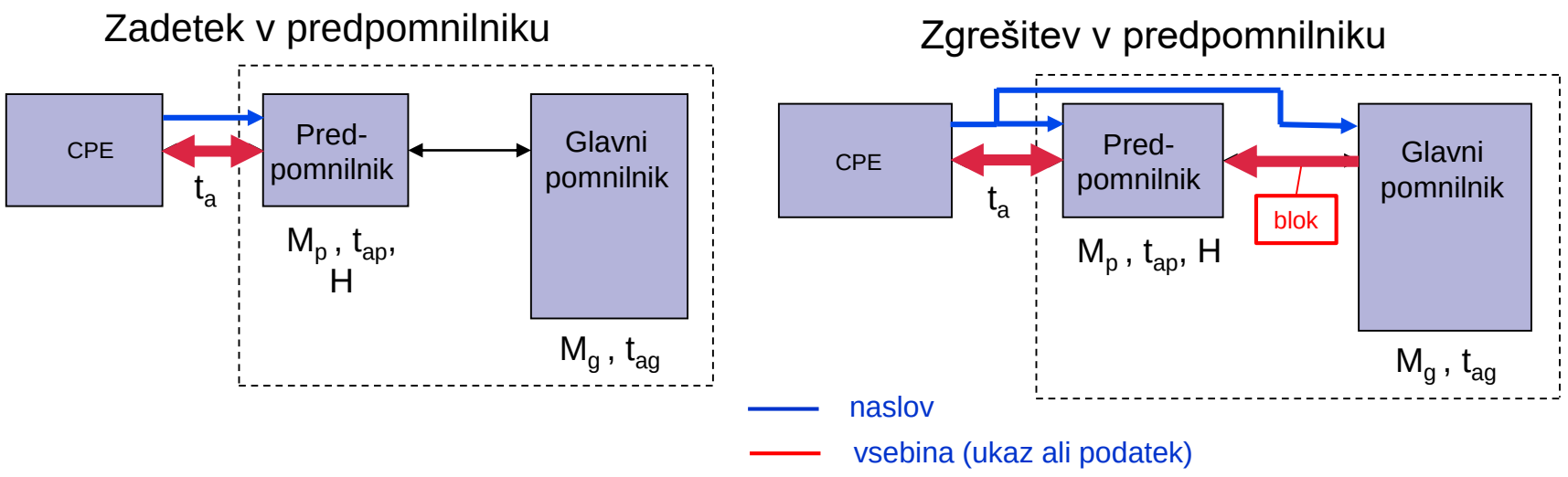
Dvonivojska pomnilniška hierarhija

M_p predpomnilnik
t_{ap} čas dostopa do predpomnilnika [ns]
H_p verjetnost zadetka v predpomnilniku [%]

M_g glavni pomnilnik
t_{ag} čas dostopa do glavnega pomnilnika [ns]
(verjetnost zadetka v glavnem pomnilniku je 100%)

t_a povprečni čas dostopa do celotne hierarhije, kot ga vidi CPE [ns]

- Pri dostopu CPE do informacije (ukaz, operand) v predpomnilniku sta dve možnosti:
  - **Zadetek** (hit), če je naslov (in vsebina s tega naslova) v predpomnilniku ⇒ čas dostopa je t_{ap}
  - **Zgrešitev** (miss), če naslova (in vsebine) ni v predpomnilniku ⇒ čas dostopa je $t_{ap} + t_{ag}$



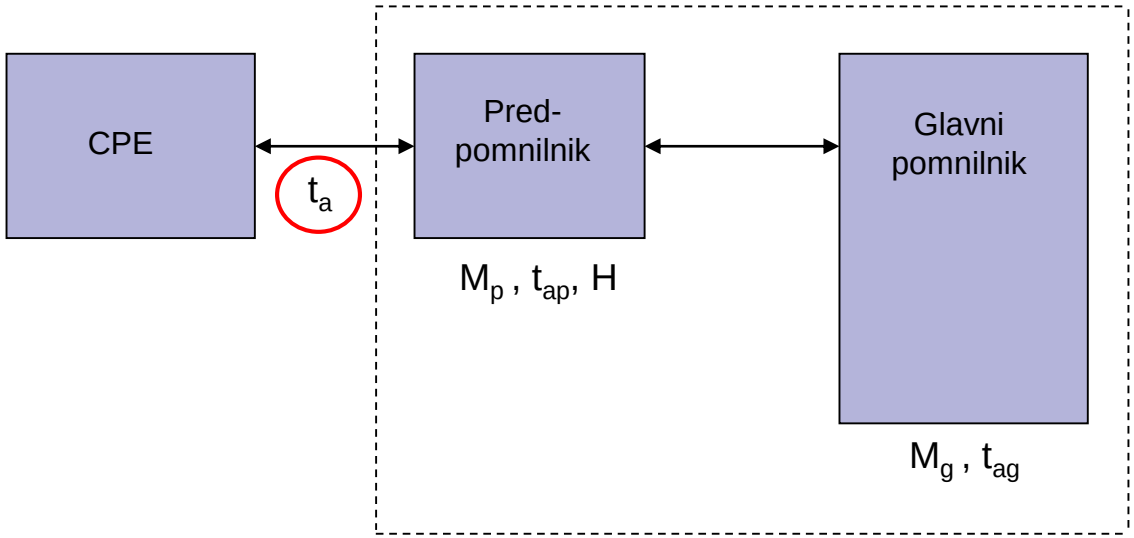
- Uspešnost delovanja predpomnilnika merimo:
  - Z verjetnostjo zadetka *H* $$H = \frac{N_p}{N} = \frac{N_p}{N_g + N_p}$$

    N - število vseh dostopov do predpomnilnika ($N = N_g + N_p$)

    N_p - število zadetkov (želena informacija je v predpomnilniku)

    N_g - število zgrešitev (želene informacije ni v predpomnilniku, potreben je prenos informacije iz glavnega pomnilnika v predpomnilnik)

  - Ali z verjetnostjo zgrešitve, ki je *1 - H* (želimo čim manjšo)
- V predpomnilniku je verjetnost zadetka H > 0,9 (90%)
- V primeru zgrešitve je potreben dostop do glavnega pomnilnika.



- Povprečni dostopni čas t_a, kot ga vidi CPE, je:

$$t_a = H\,t_{ap} + (1-H)(t_{ap} + t_{ag})$$

$$t_a = t_{ap} + (1-H)t_{ag}$$

- Pri računanju je treba upoštevati dve posebnosti predpomnilnikov:
  - Med glavnim pomnilnikom in predpomnilnikom se vedno prenaša **predpomnilniški blok**, kar je več sosednjih pomnilniških besed (bajtov)
  - Čas v računalniku merimo z urinimi periodami
- Čas t_{ap} za dostop do informacije v predpomnilniku je na nivoju L1 pri večini računalnikov od ene do nekaj urinih period.
- Pri zgrešitvi $\Rightarrow$ čas za dostop do glavnega pomnilnika in prenos bloka v predpomnilnik imenujemo **zgrešitvena kazen t_B.**

- Zgrešitvena kazen t_B je čas, ki se ob zgrešitvi (verjetnost za zgrešitev je $1 - H$) prišteje k času dostopa do predpomnilnika.
- Zgrešitvena kazen je tipično med 10 in 100 urinimi periodami.
- Če ima računalnik tudi predpomnilnik na nivoju L2, je zgrešitvena kazen precej manjša, ker je predpomnilnik L2 hitrejši kot glavni pomnilnik.

- Povprečni čas dostopa t_a je z upoštevanjem zgrešitvene kazni:

$$t_a = t_{ap} + (1 - H)t_B$$

t_{ap} – dostopni čas do predpomnilnika

(1 - H) - verjetnost zgrešitve v predpomnilniku

t_B – zgrešitvena kazen (čas dostopa do glavnega pomnilnika + čas za prenos bloka v predpomnilnik)

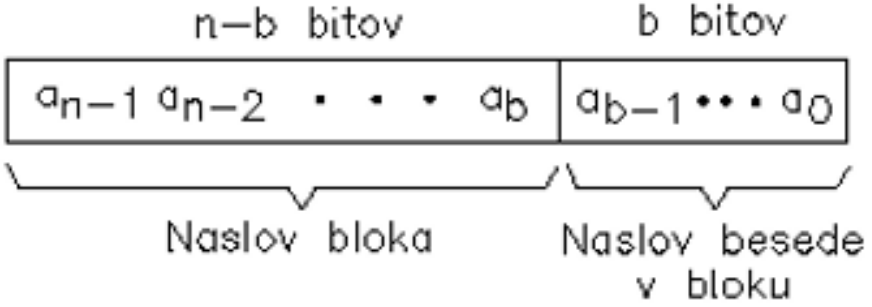
- Če sta časa t_{ap} in t_B v urinih periodah, je seveda tudi rezultat t_a v urinih periodah.

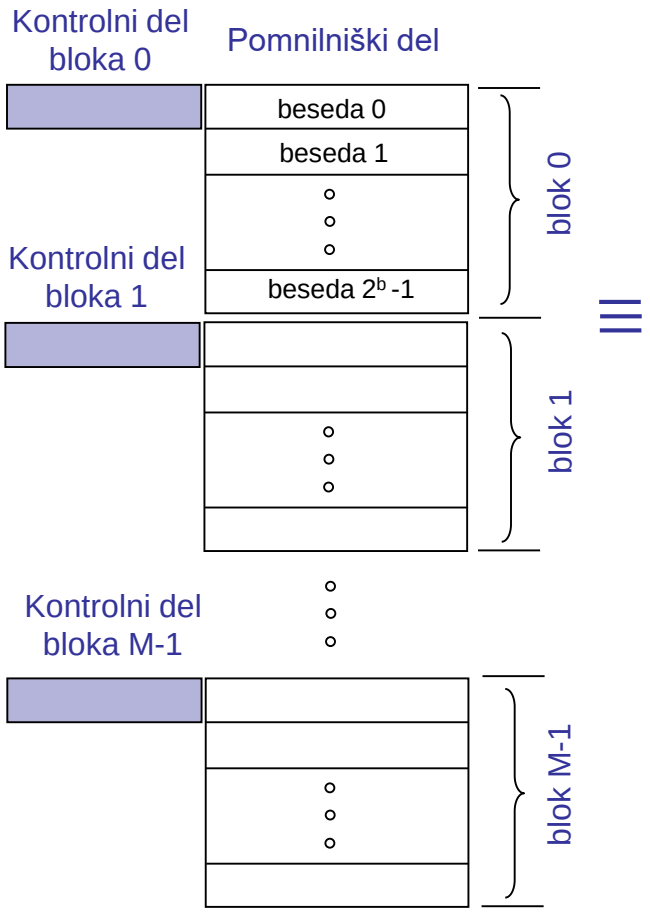
- Povprečni dostopni čas v sekundah (t_{CPE} je trajanje ene urine periode v sekundah) :

$$t_a \text{ [s]} = t_a \text{ [urine periode]} * t_{CPE} \text{ [s]}$$

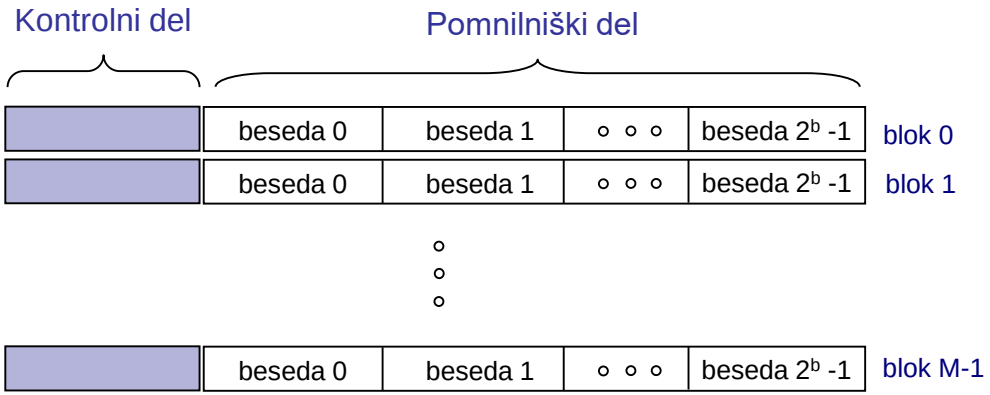
- Vsebina v predpomnilniku se spreminja ⇒
  - V predpomnilnik se prenašajo bloki iz glavnega pomnilnika
  - in naslovi teh blokov (številke blokov iz glavnega pomnilnika)
- Zato je vsak predpomnilnik sestavljen iz dveh delov:
  - **Pomnilniškega dela**, ki je razdeljen na bloke ali predpomnilniške vrstice
  - **Kontrolnega dela**, ki ga sestavljajo kontrolne besede. Vsakemu bloku v pomnilniškem delu pripada ena kontrolna beseda, ki vsebuje naslov bloka (številko bloka v glavnem pomnilniku), ki je v pomnilniškem delu.

Zgradba predpomnilnika



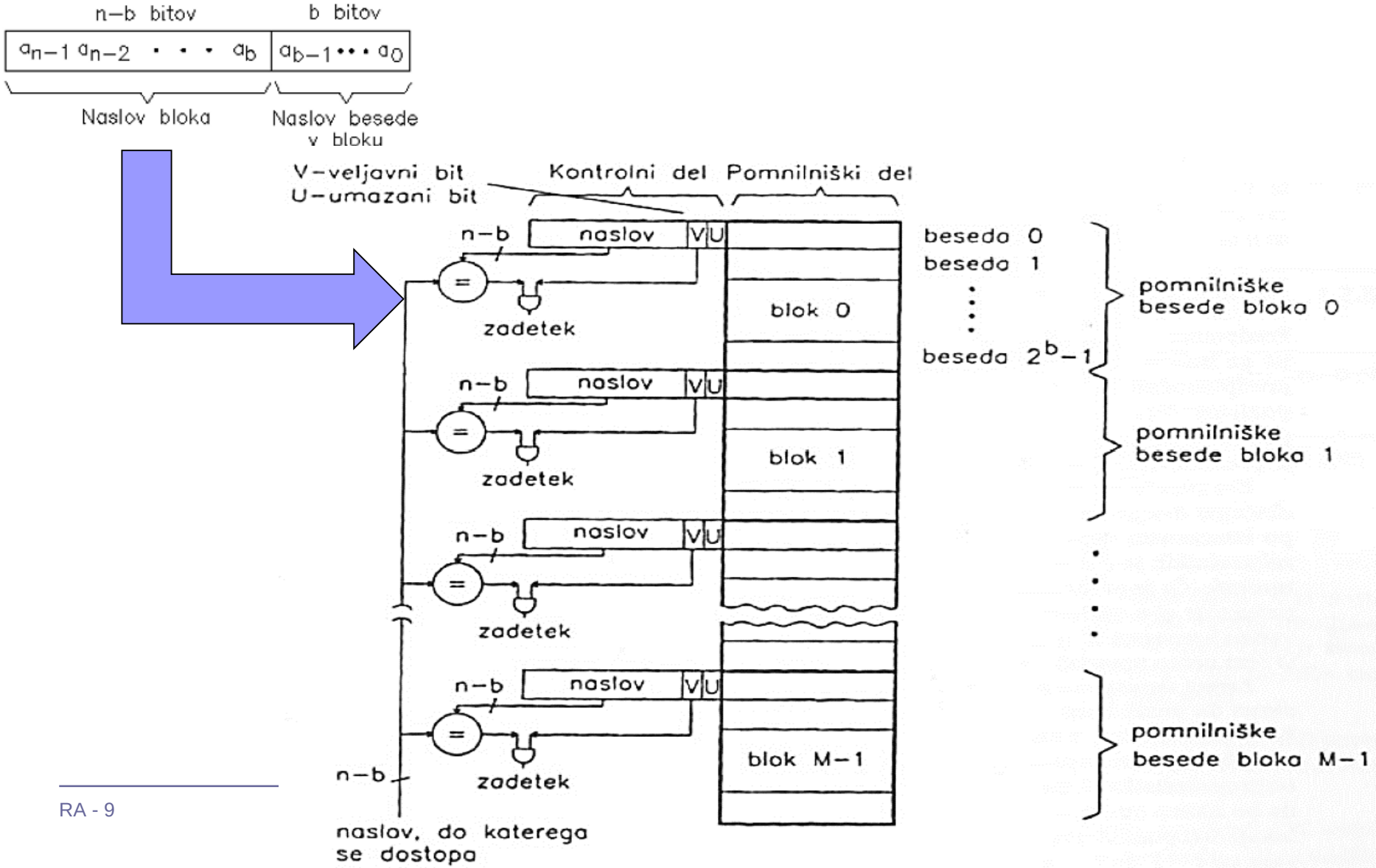


=



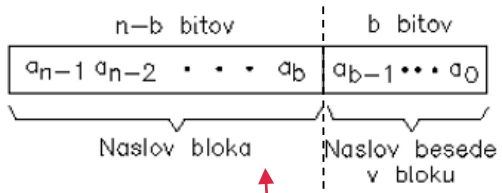
Blok ali predpomnilniška vrstica = 2^b besed

Zgradba in delovanje predpomnilnika



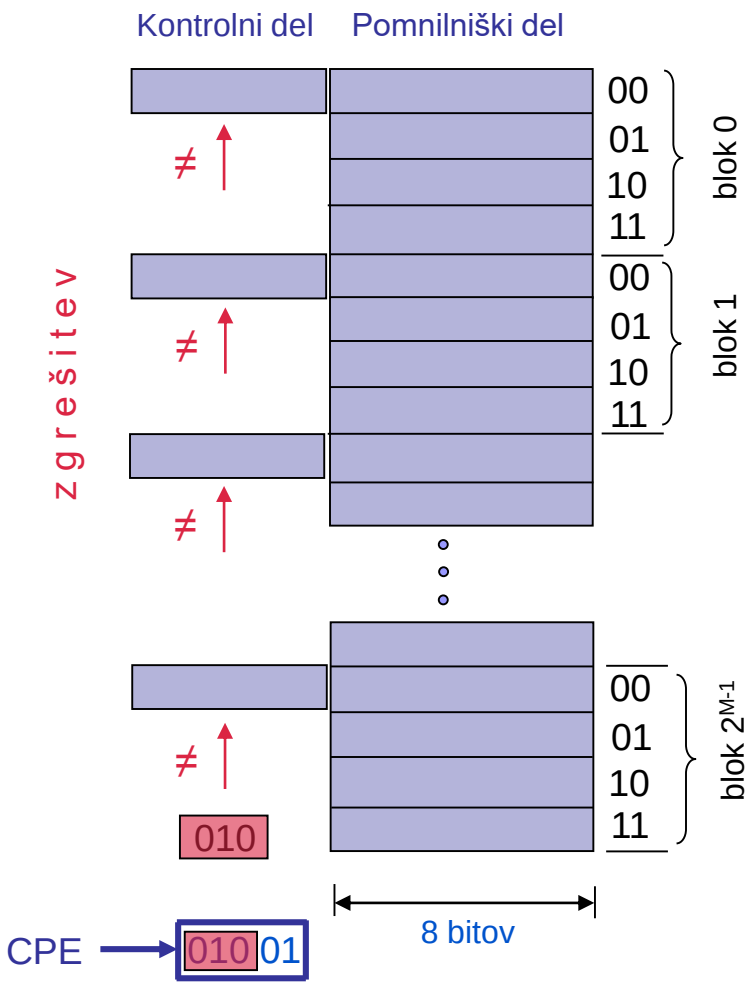
- Blok (ali predpomnilniška vrstica) je nekaj sosednjih pomnilniških besed (pomnilniška beseda je običajno 1 bajt).
- Velikost bloka ($B = 2^b$) je tipično 4 do 512 pomnilniških besed.
- Med glavnim pomnilnikom in predpomnilnikom se **vedno prenaša cel blok.**
- Ko se blok iz glavnega pomnilnika prenese v prosti blok v predpomnilniku se:
  - Vsebina bloka se prenese v pomnilniški del bloka v predpomnilniku
  - Naslov (številka) bloka se prenese v kontrolni del bloka v predpomnilniku

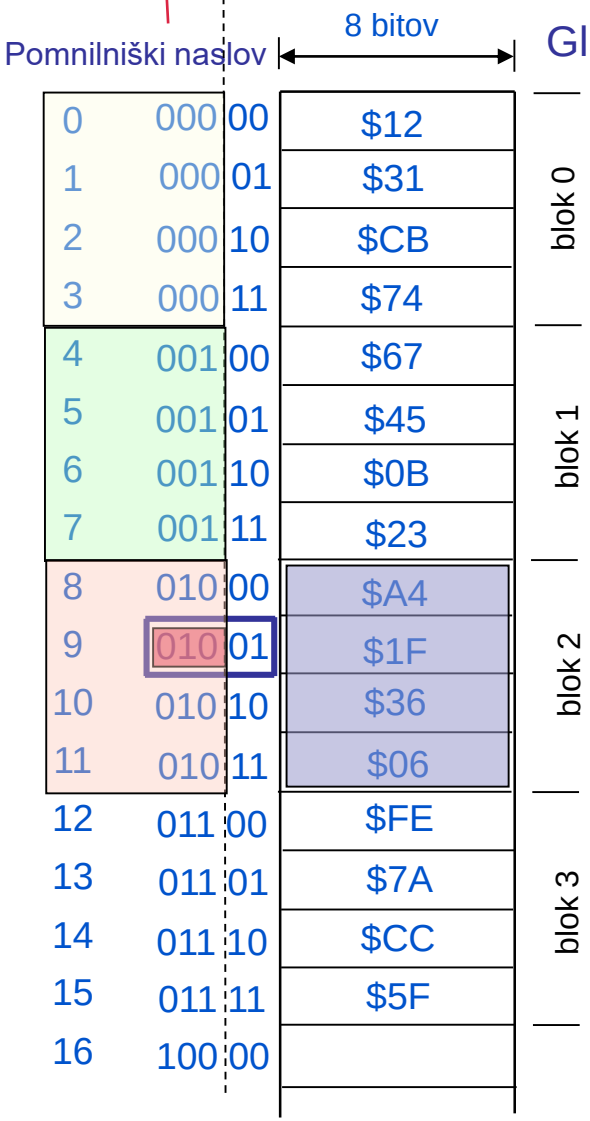
- Primer delovanja predpomnilnika:
  - Predpostavimo:
    - da procesor zaporedoma dostopa do pomnilniških besed z naslednjimi desetiškimi pomnilniškimi naslovi:
    - 9, 10, 11, 2, 3, 9, 10, 11, 2, … ;
    - da ima predpomnilnik bloke velikosti 4 bajte ($B = 2^2 = 4$) in je na začetku prazen;
    - da je pomnilniški naslov je dolg 5 bitov.
    - Potem zgornji trije biti pomnilniškega naslova določajo številko bloka, spodnja dva bita pomnilniškega naslova pa besedo (bajt) v bloku ($2^2 = 4$)



CPE dostopa do pomn. naslova:

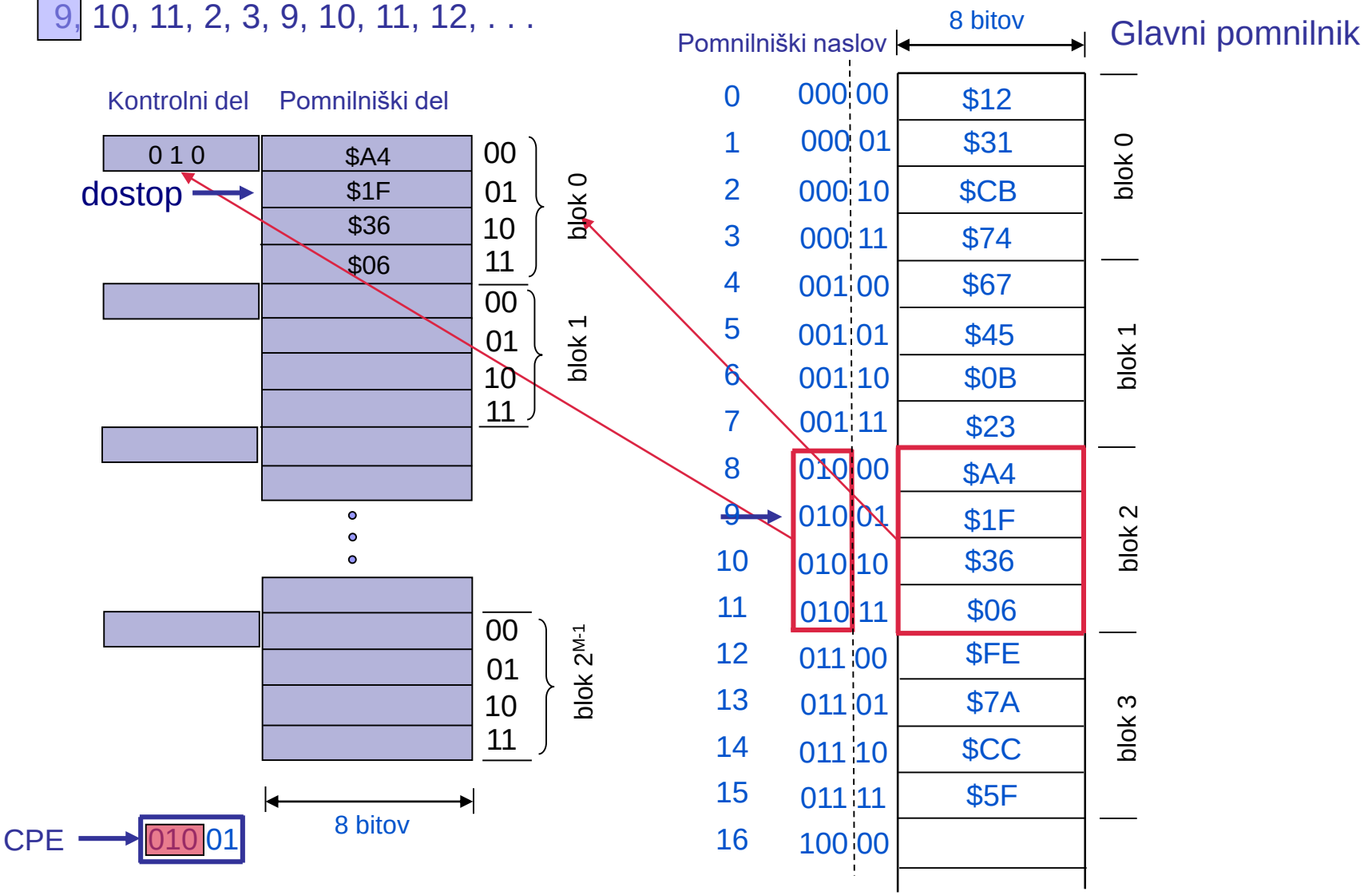
9, 10, 11, 2, 3, 9, 10, 11, 12, . . .





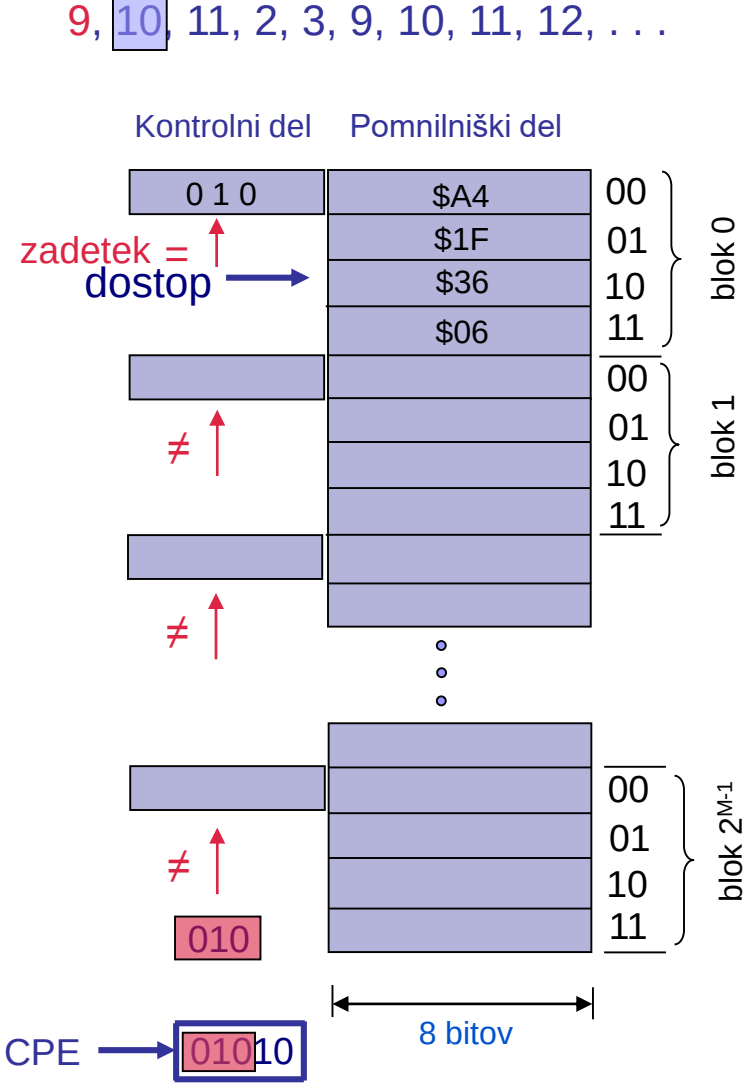
CPE dostopa do pomn. naslova:

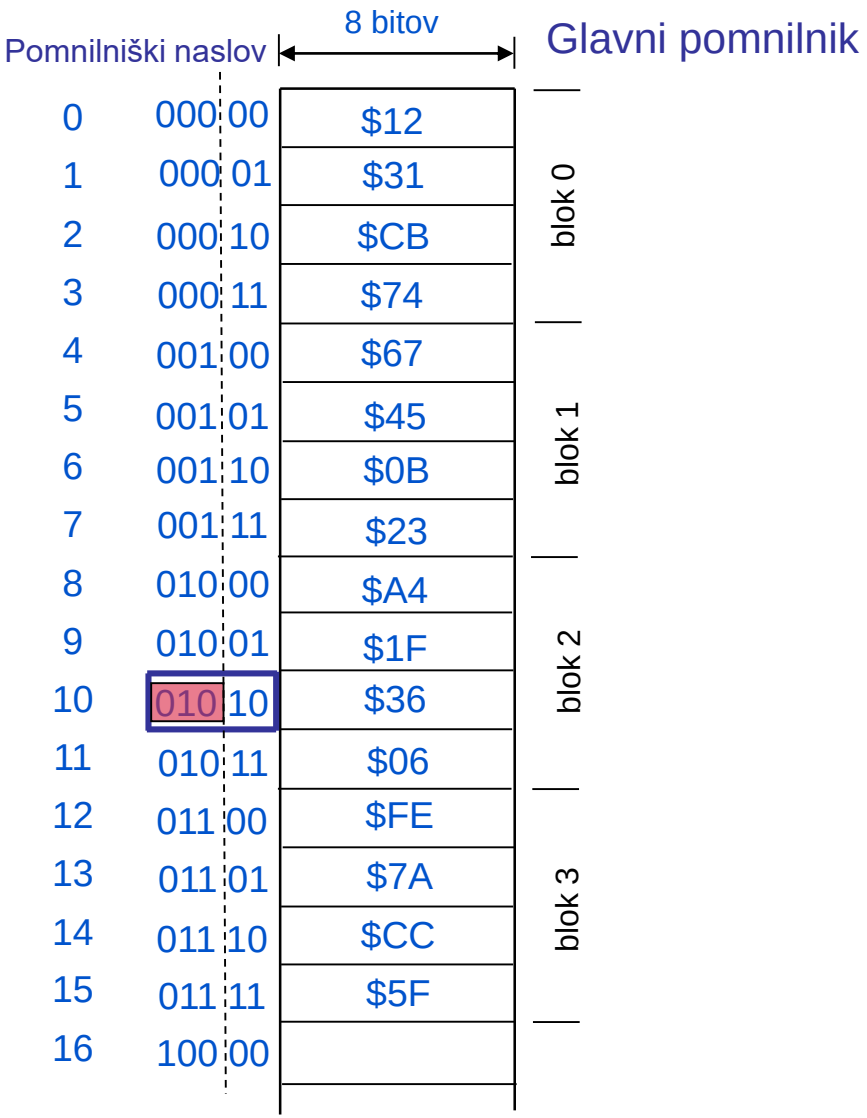
9, 10, 11, 2, 3, 9, 10, 11, 12, . . .



CPE dostopa do pomn. naslova:

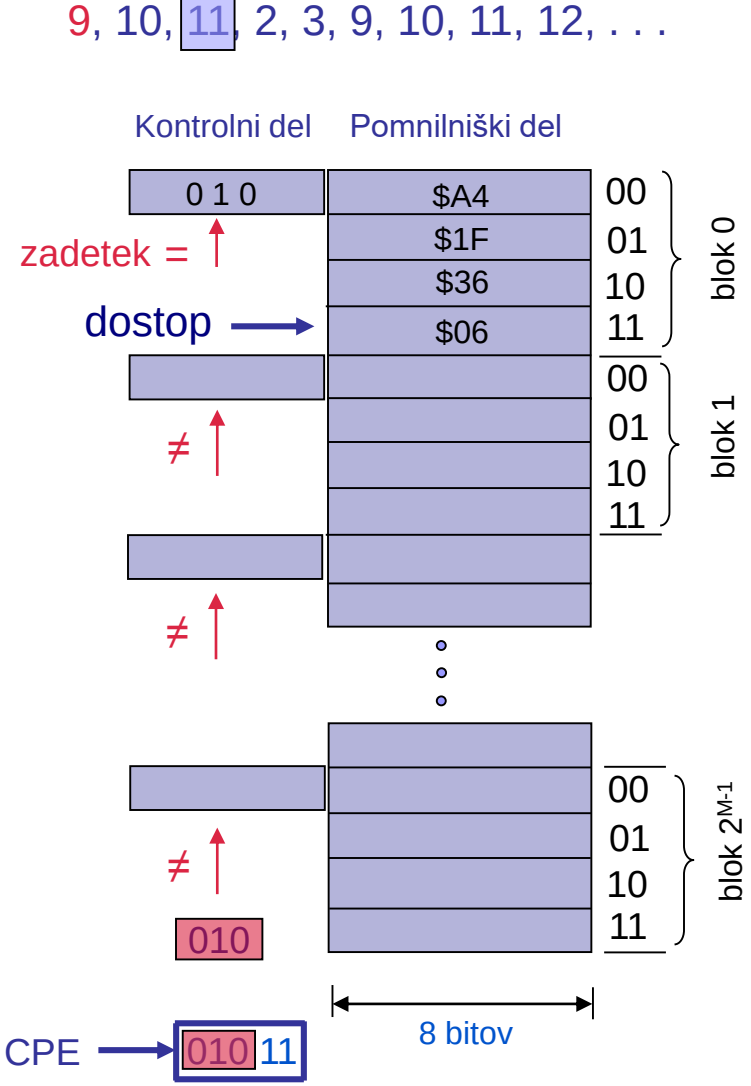
9, 10, 11, 2, 3, 9, 10, 11, 12, . . .

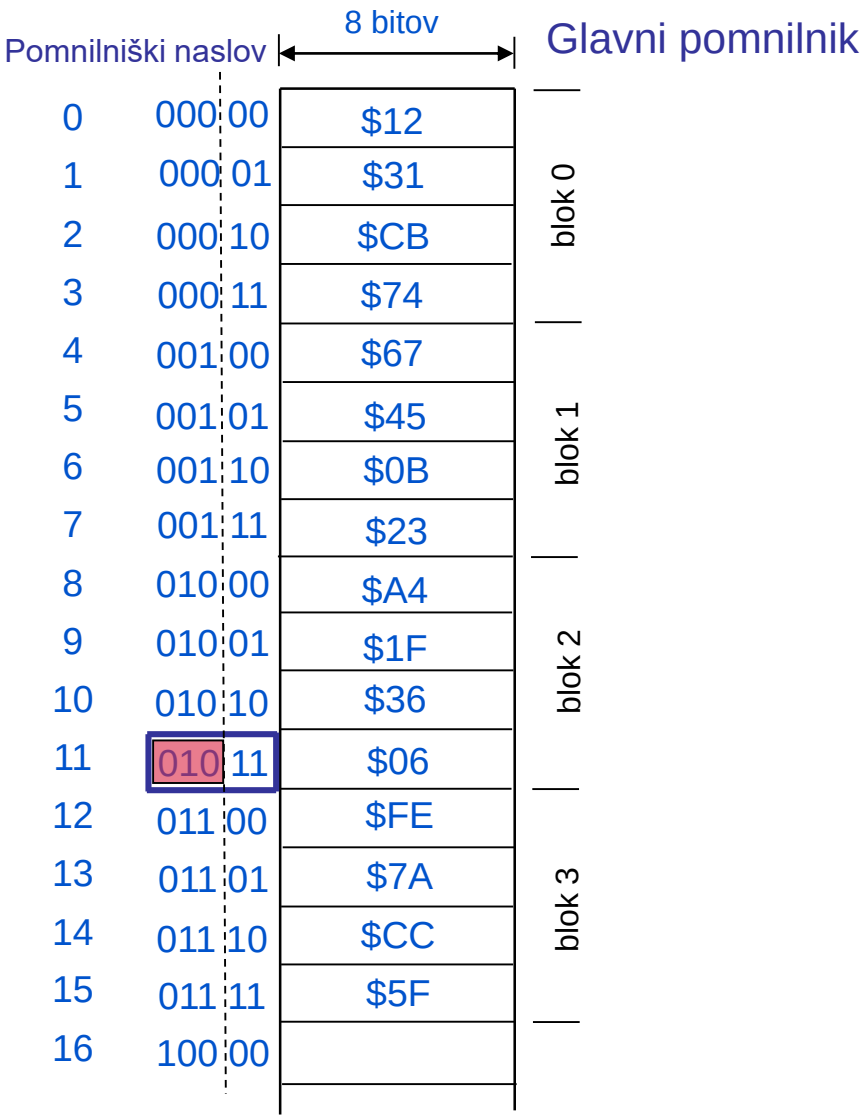




CPE dostopa do pomn. naslova:

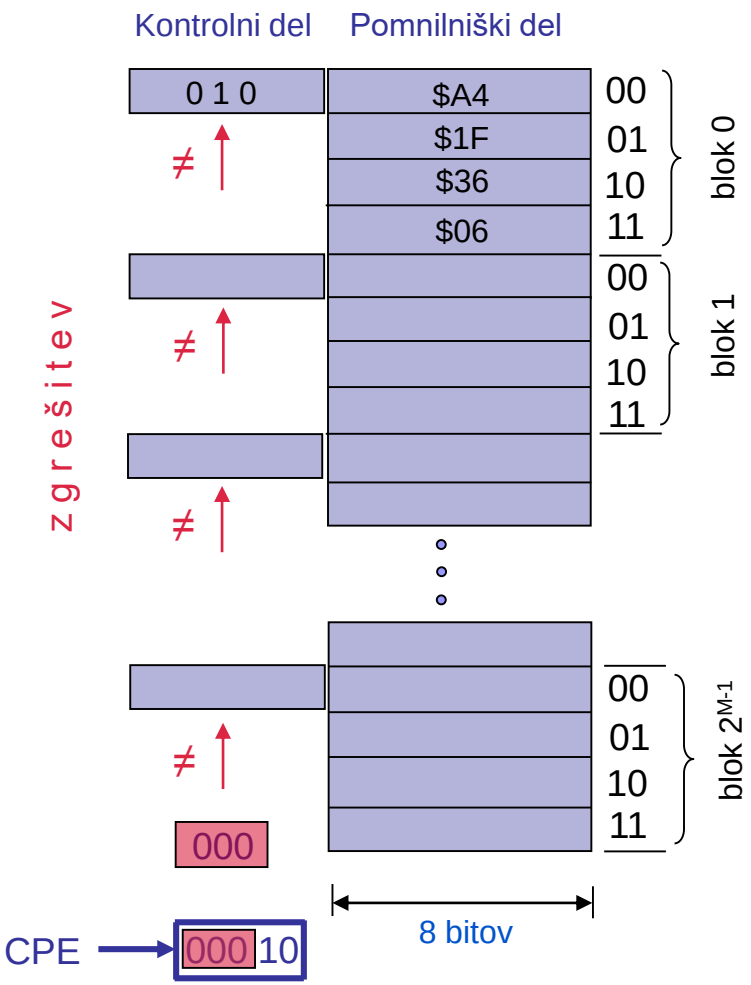
9, 10, 11, 2, 3, 9, 10, 11, 12, . . .

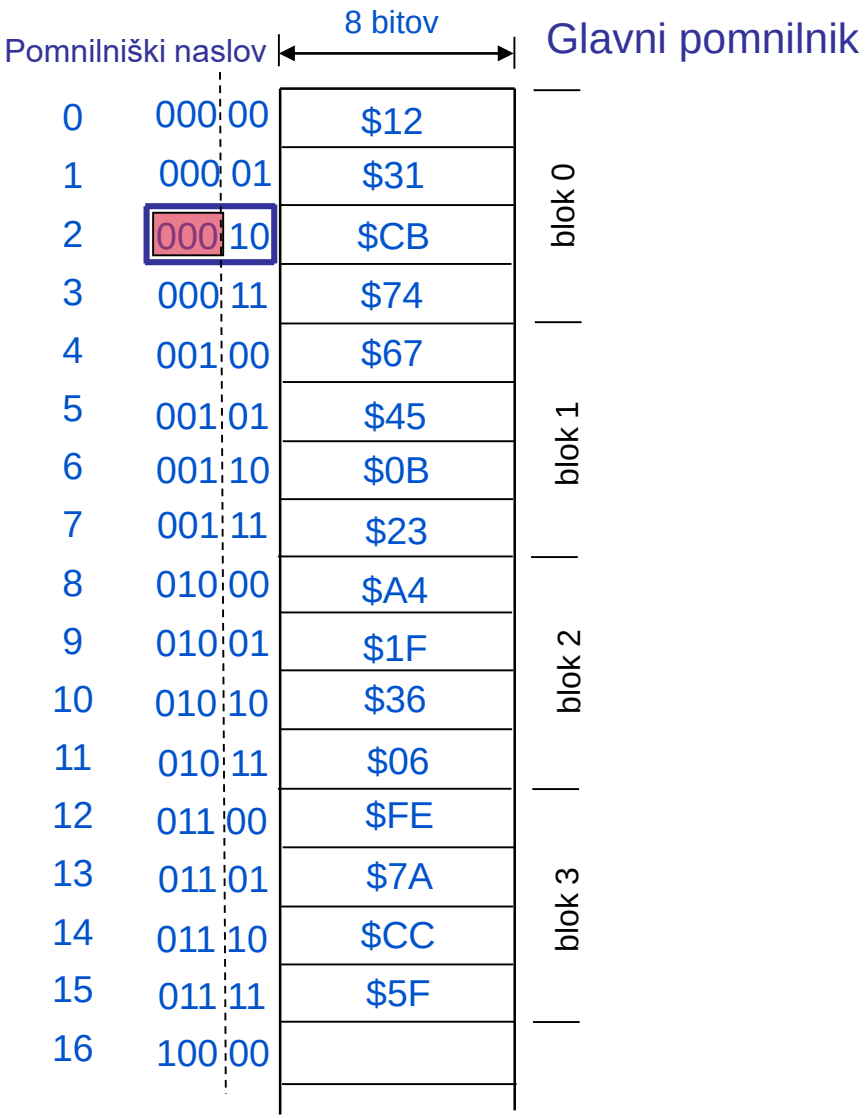




CPE dostopa do pomn. naslova:

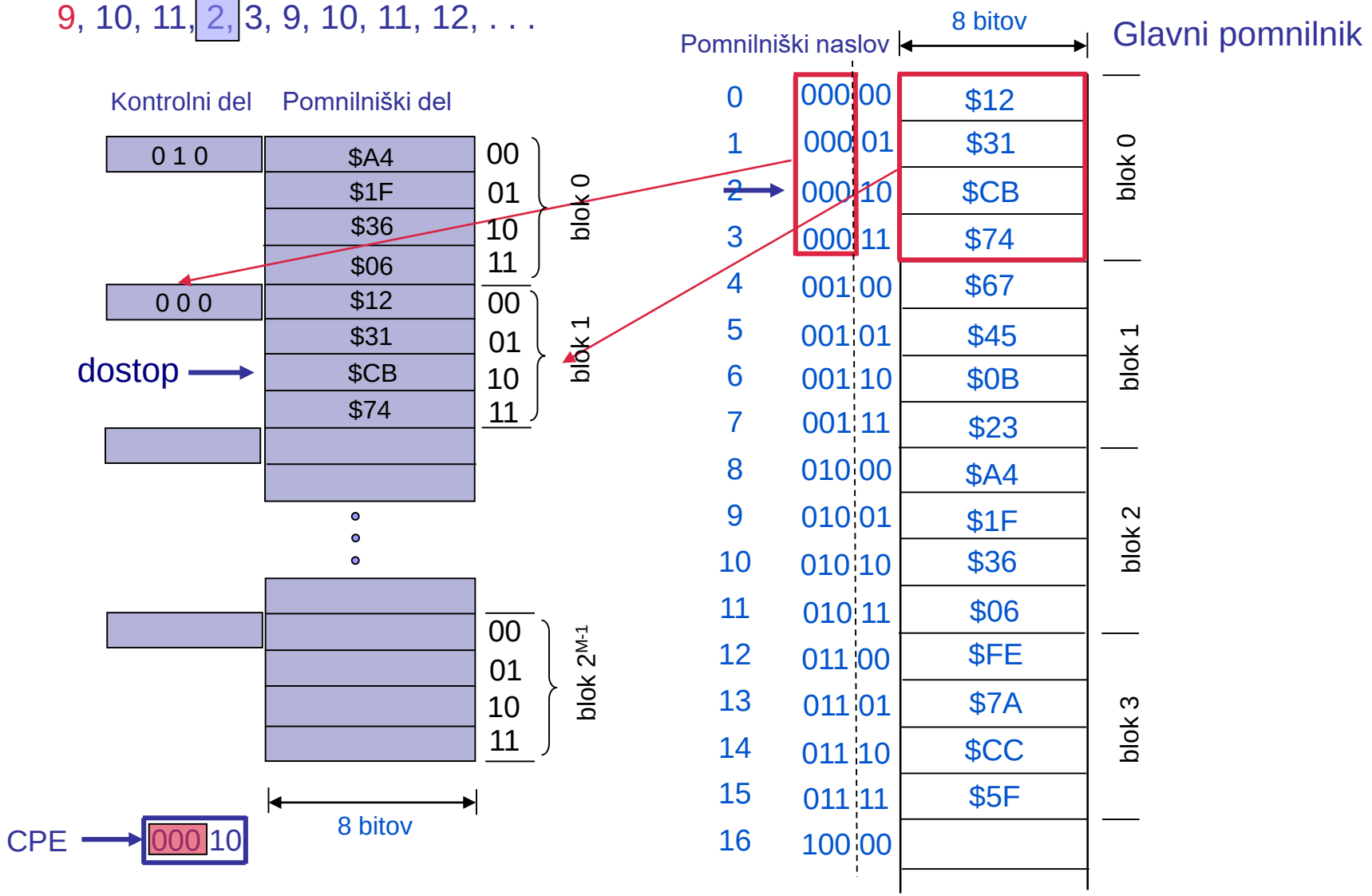
9, 10, 11, 2, 3, 9, 10, 11, 12, . . .





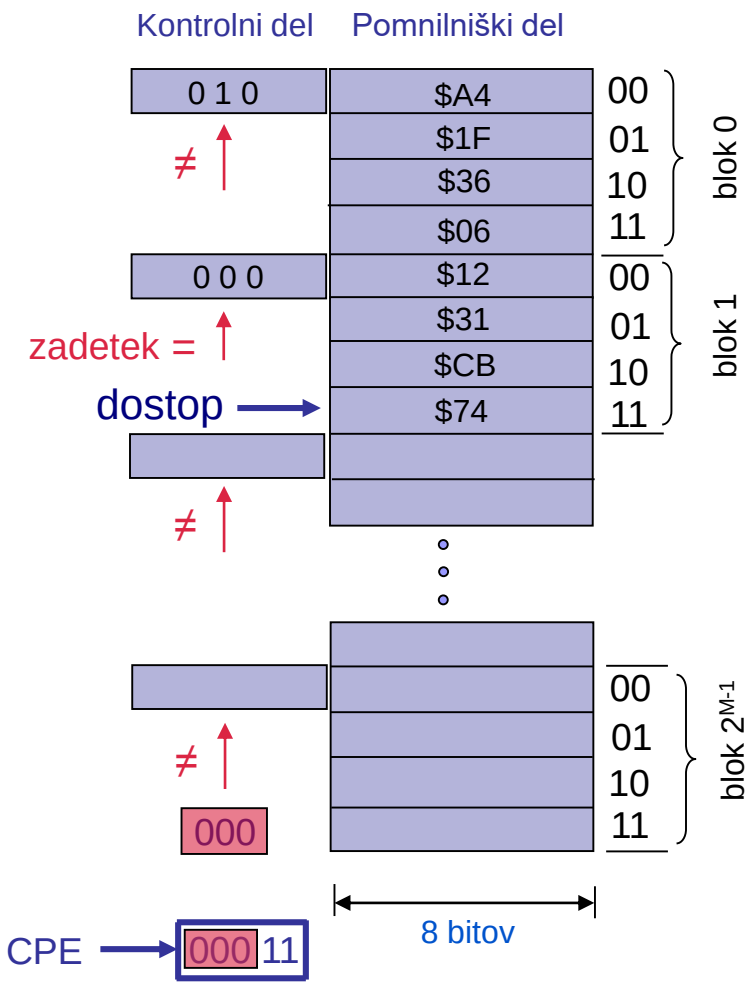
CPE dostopa do pomn. naslova:

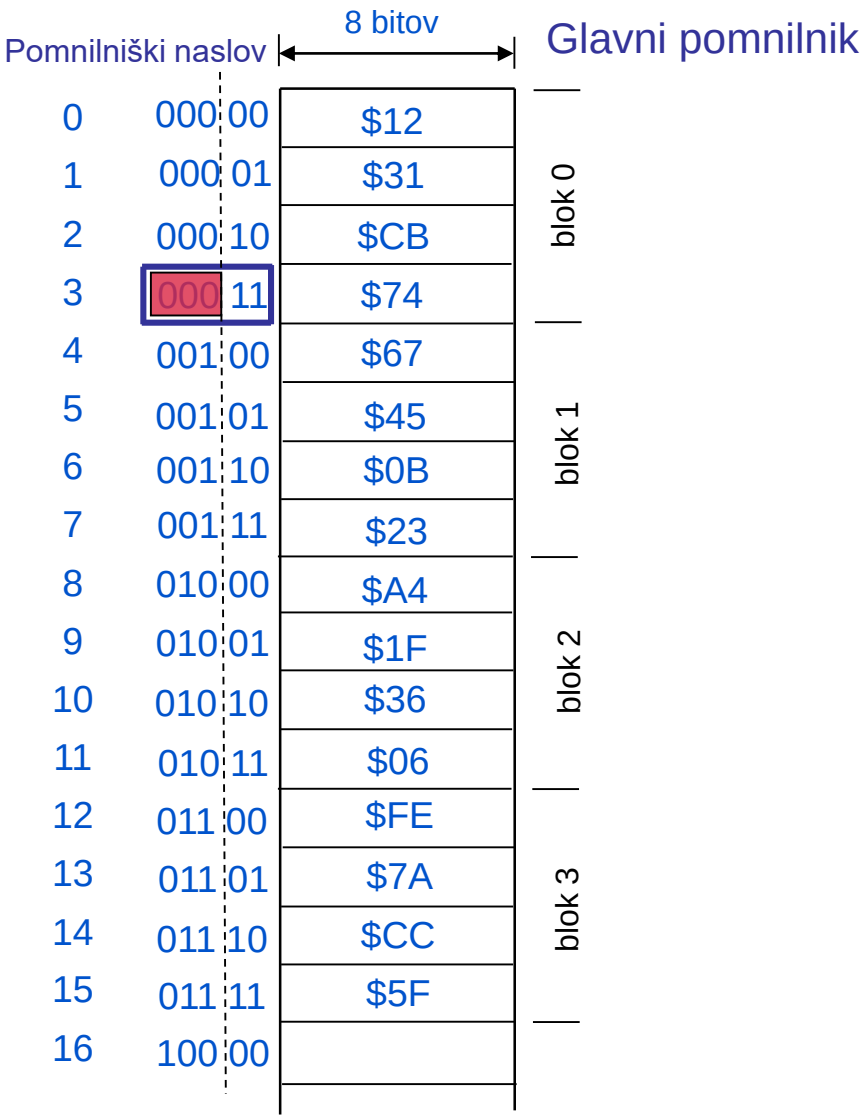
9, 10, 11, 2, 3, 9, 10, 11, 12, . . .



CPE dostopa do pomn. naslova:

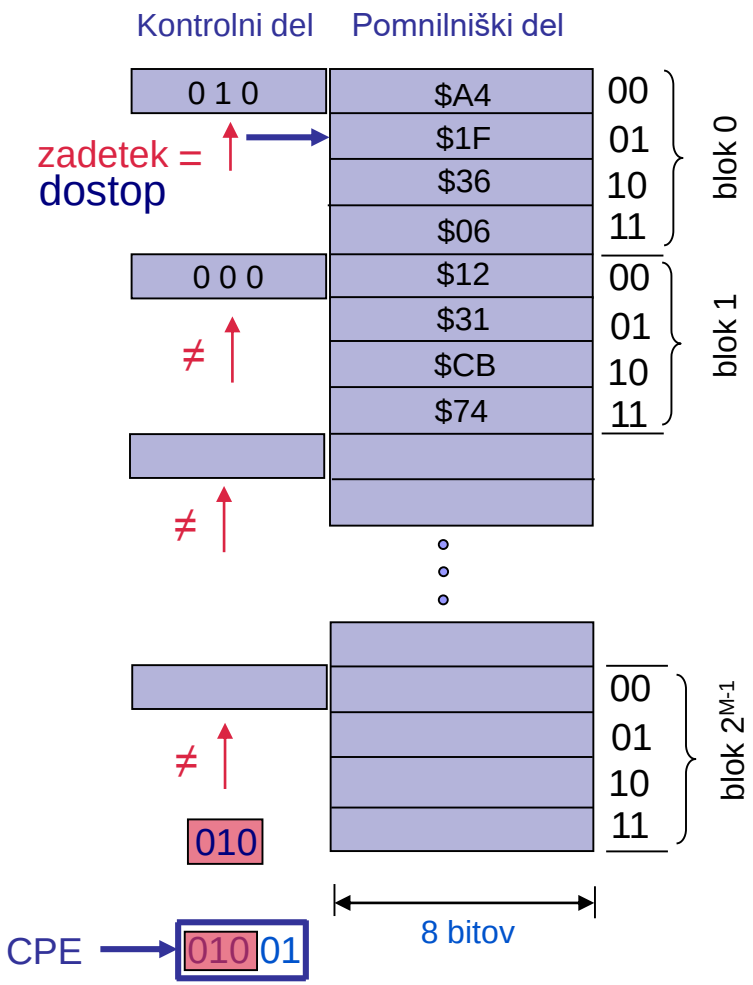
9, 10, 11, 2, 3, 9, 10, 11, 12, . . .

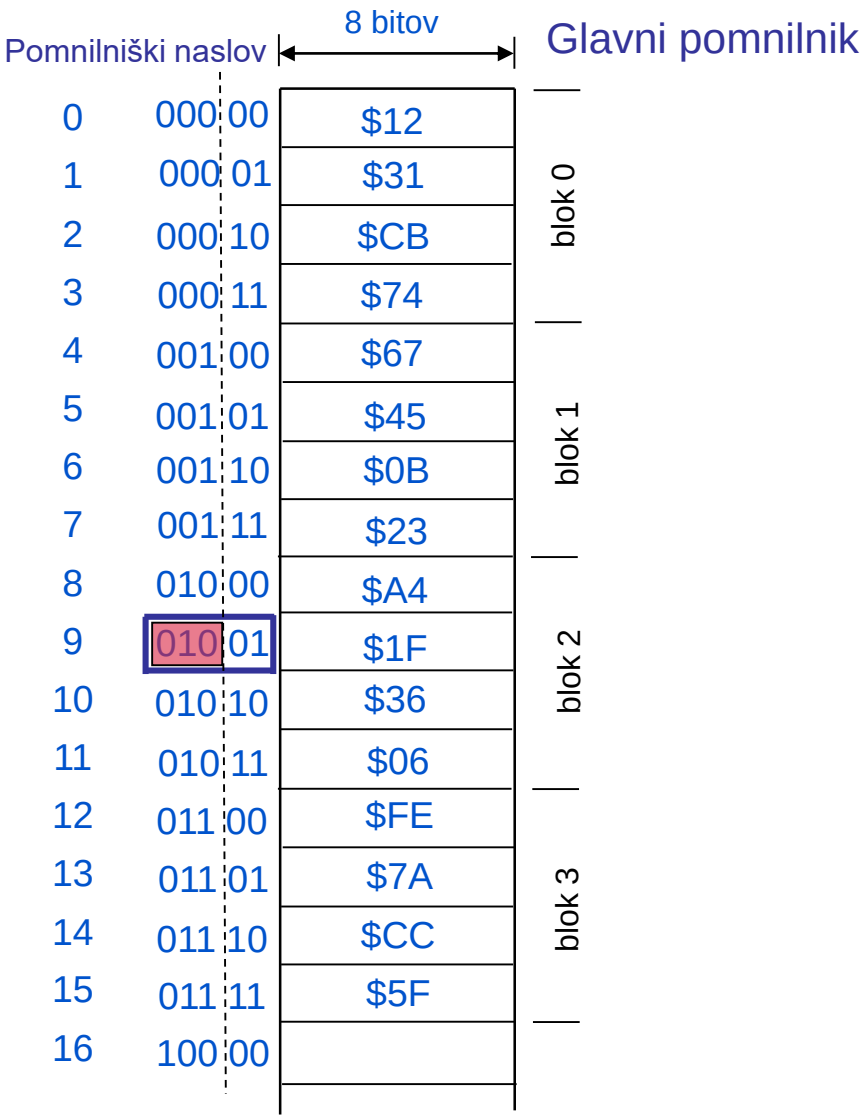


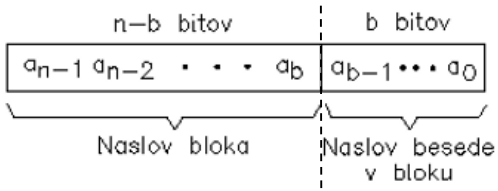


CPE dostopa do pomn. naslova:

9, 10, 11, 2, 3, 9, 10, 11, 12, . . .



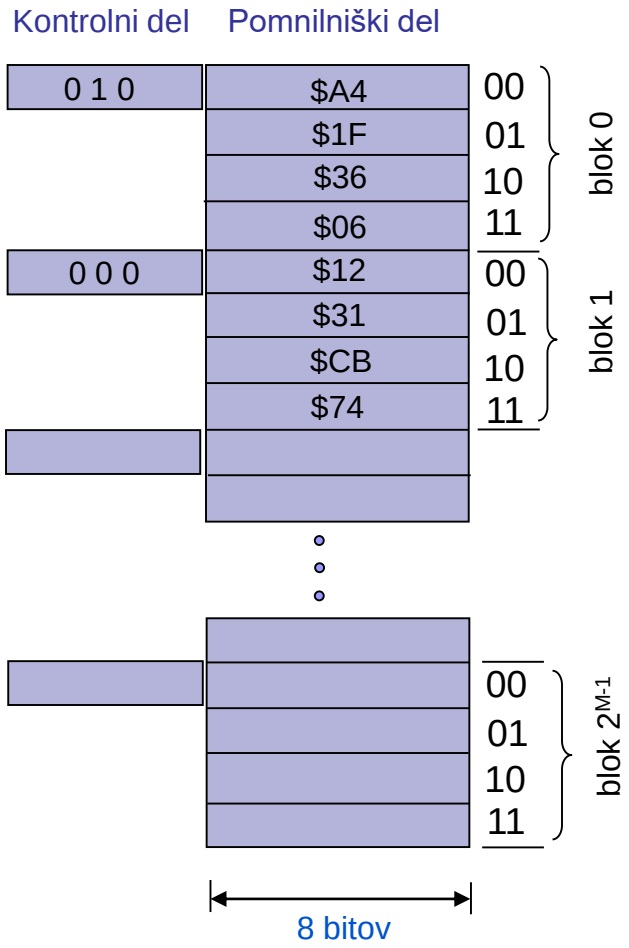


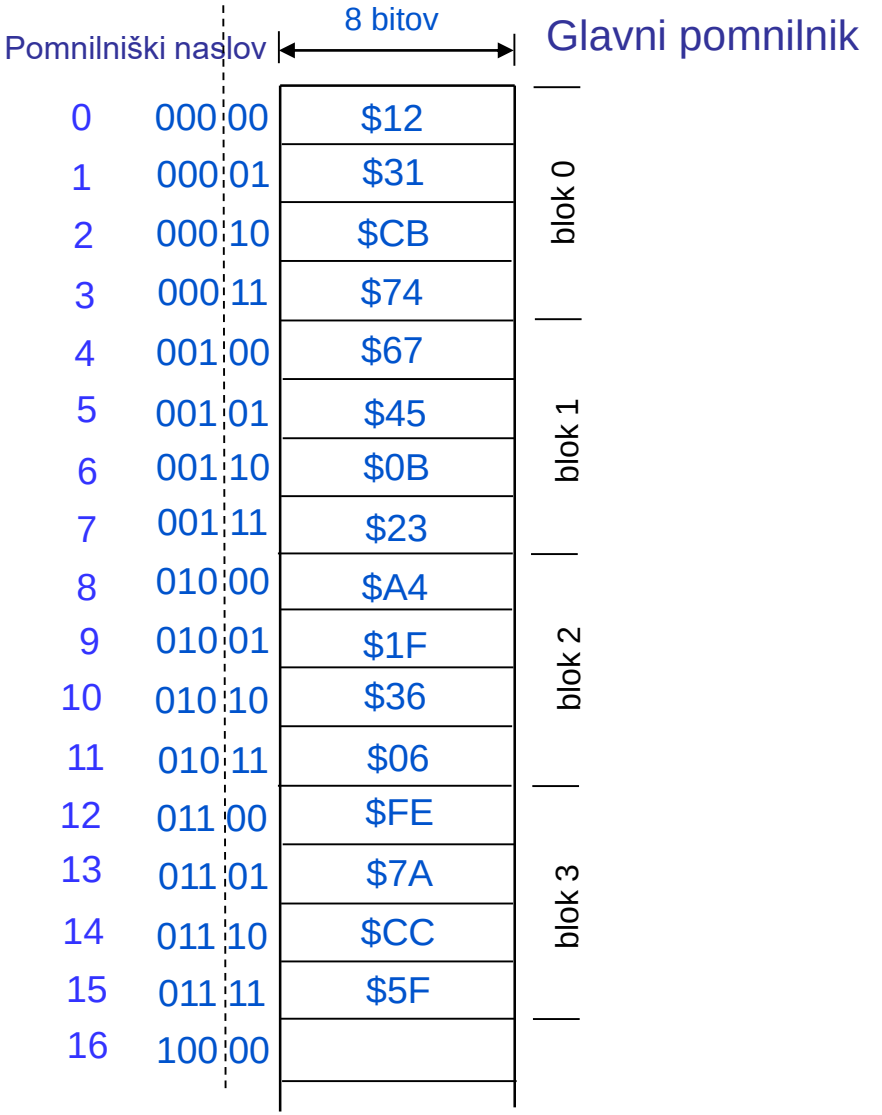


CPE dostopa do pomn. naslova:

9, 10, 11, 2, 3, 9, 10, 11, 2, . . .

zgrešitvi



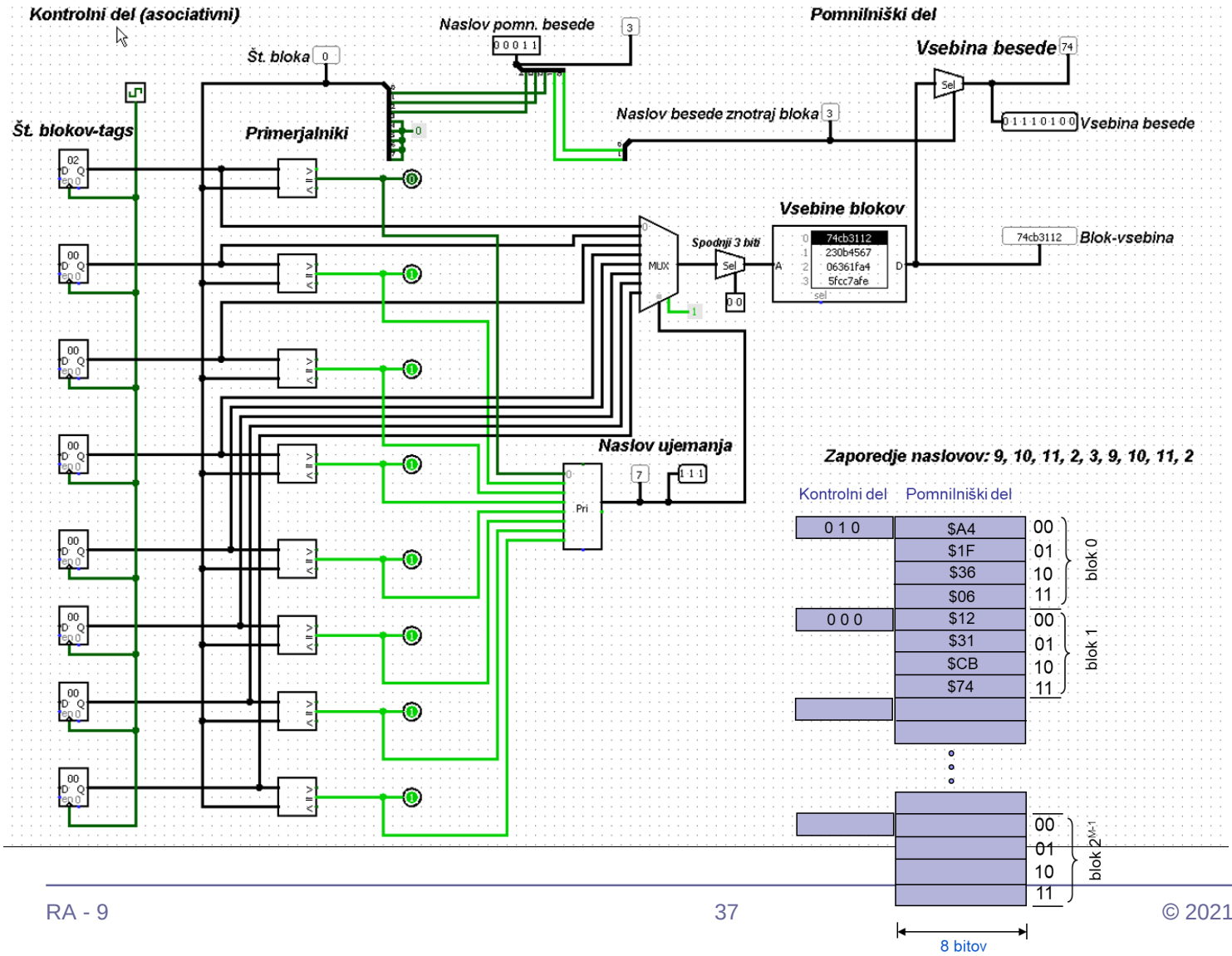


CPE dostopa do pomn. naslova:

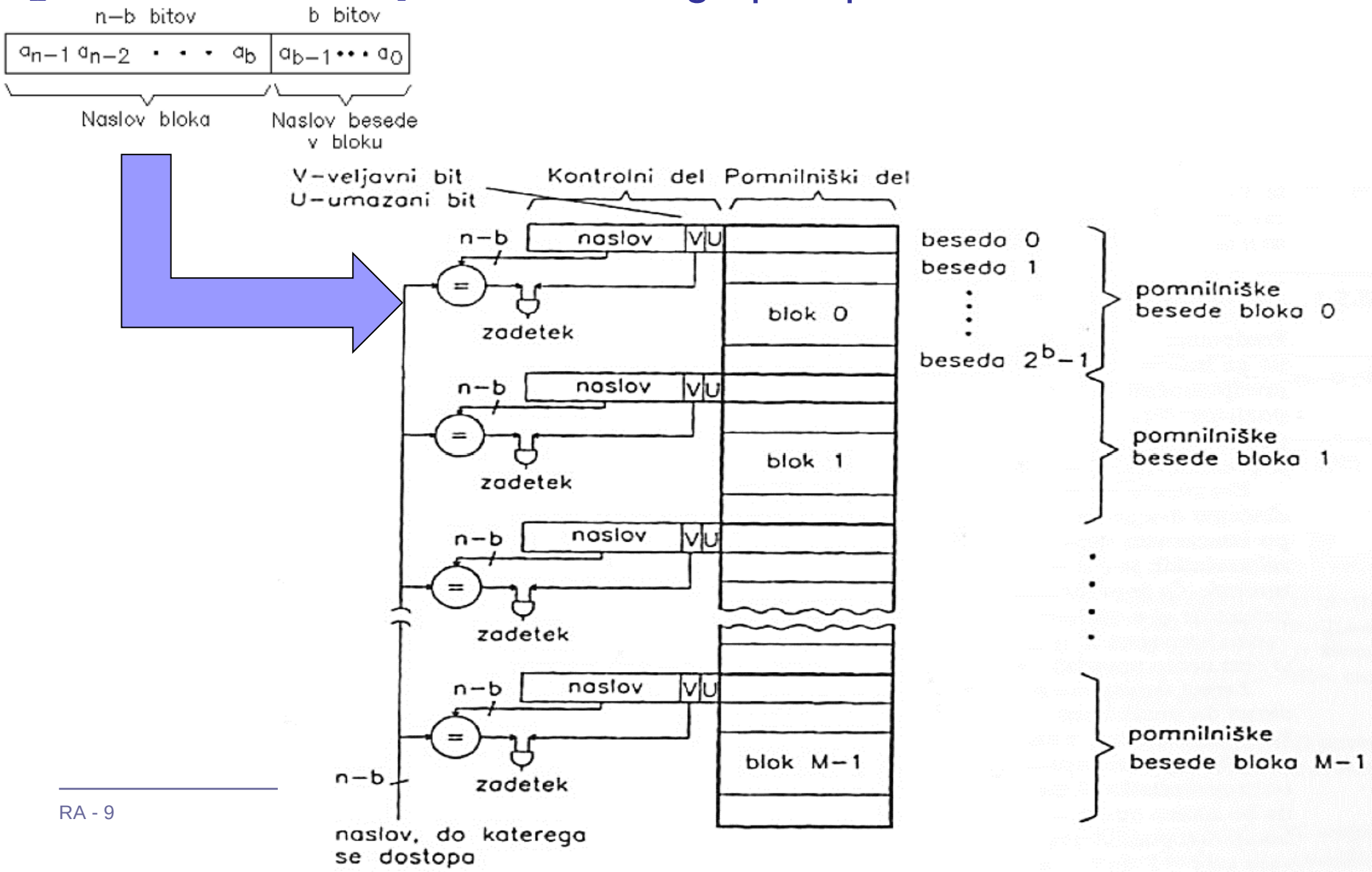
9, 10, 11, 2, 3, 9, 10, 11, 2, . . .

zgrešitvi

Enostaven model predpomnilnika v Logisimu



Zgradba in delovanje asociativnega predpomnilnika



Dostop do operanda v pomnilniku :

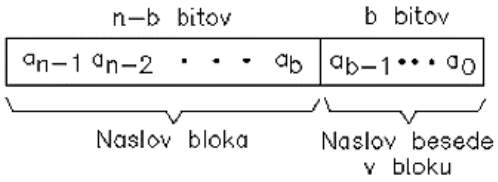
- Zadetek v predpomnilniku (verjetnost H):
  - CPE dostopa do operanda v predpomnilniku (bere ali piše)
- Zgrešitev v predpomnilniku (verjetnost 1 – H):
  - Preslikava bloka iz glavnega pomnilnika v predpomnilnik ali
  - zamenjava bloka – če je predpomnilnik poln, se eden od blokov v predpomnilniku shrani nazaj v glavni pomnilnik (ali je to vedno potrebno?), na njegovo mesto pa se preslika nov blok iz glavnega pomnilnika
  - CPE dostopa do operanda v predpomnilniku

Vrste predpomnilnikov glede na omejitve pri preslikavi blokov

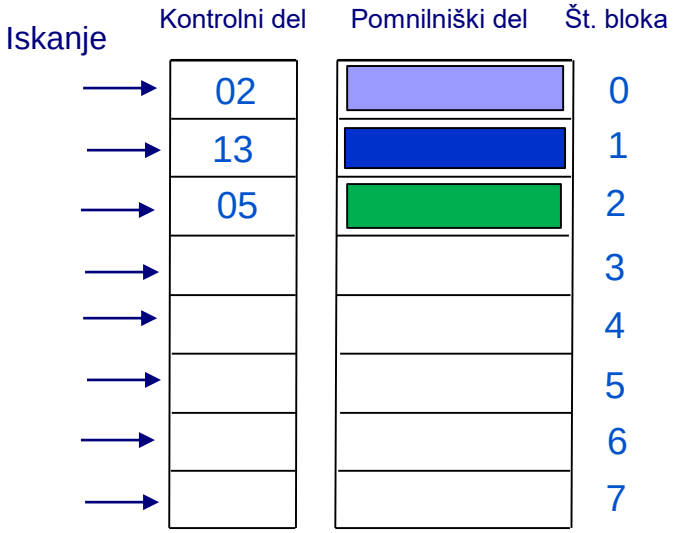
- Iskanje bloka v predpomnilniku mora biti hitro.
- Če to ni mogoče, je treba pri preslikavi bloka iz glavnega pomnilnika v predpomnilnik vpeljati omejitve.
- Glede na strogost omejitev pri preslikavi bloka iz glavnega pomnilnika v predpomnilnik, razlikujemo tri vrste predpomnilnikov:
  - Čisti asociativni predpomnilnik
    - (brez omejitev pri preslikavi bloka v predpomnilnik)
  - Set-asociativni predpomnilnik
    - (blok se lahko preslika samo v točno določen set, znotraj seta pa brez omejitev)
  - Direktni predpomnilnik
    - (blok se lahko preslika samo v točno določen blok)

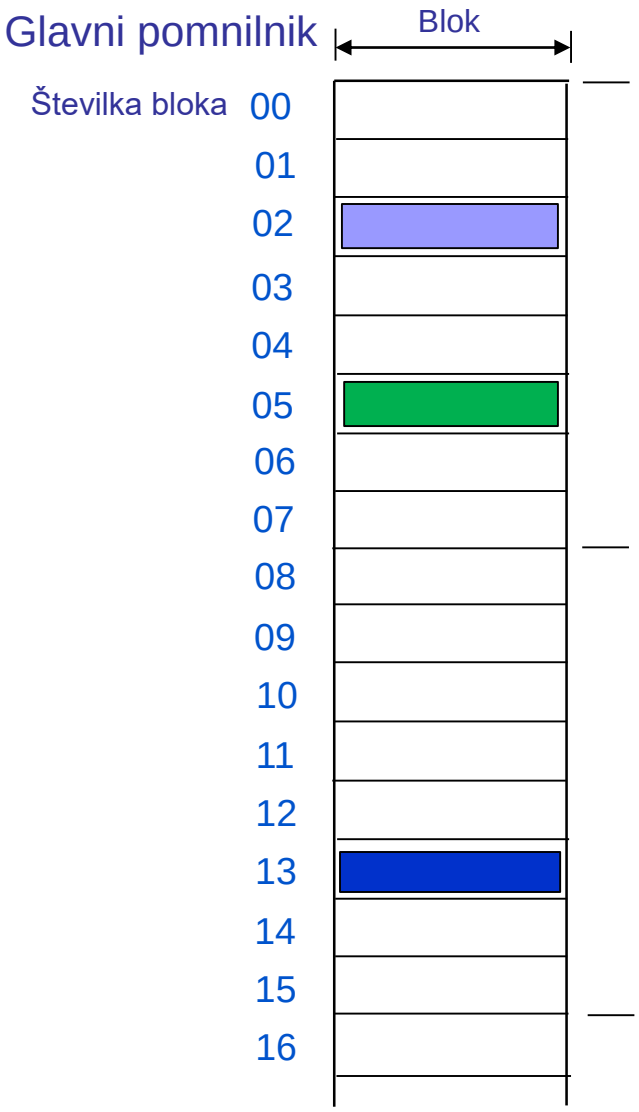
Čisti asociativni predpomnilnik

- Pomnilniški del predpomnilnika je statični RAM (velja za vse tri vrste predpomnilnikov)
- Kontrolni del predpomnilnika je asociativni pomnilnik, ki omogoča hitro iskanje številke bloka po celem kontrolnem delu predpomnilnika
- Ker je iskanje hitro po celem predpomnilniku, se lahko blok iz glavnega pomnilnika preslika v predpomnilnik kamorkoli v katerikoli blok.
- Ker je z današnjo tehnologijo velikost asociativnega pomnilnika omejena, so čisti asociativni predpomnilniki redki in veliki le do nekaj 100 blokov.
- Če želimo velik predpomnilnik, je rešitev set-asociativni ali direktni predpomnilnik.



Čisti asociativni predpomnilnik velikosti 8 blokov



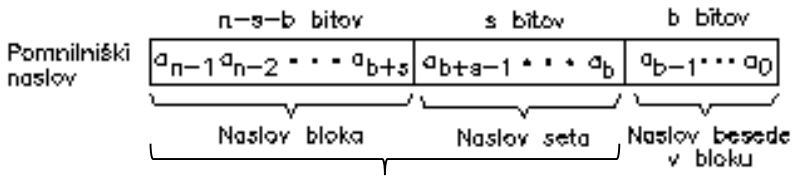


Blok iz glavnega pomnilnika se lahko preslika v katerikoli blok predpomnilnika brez omejitev, ker je iskanje po asociativnem kontrolnem delu hitro.

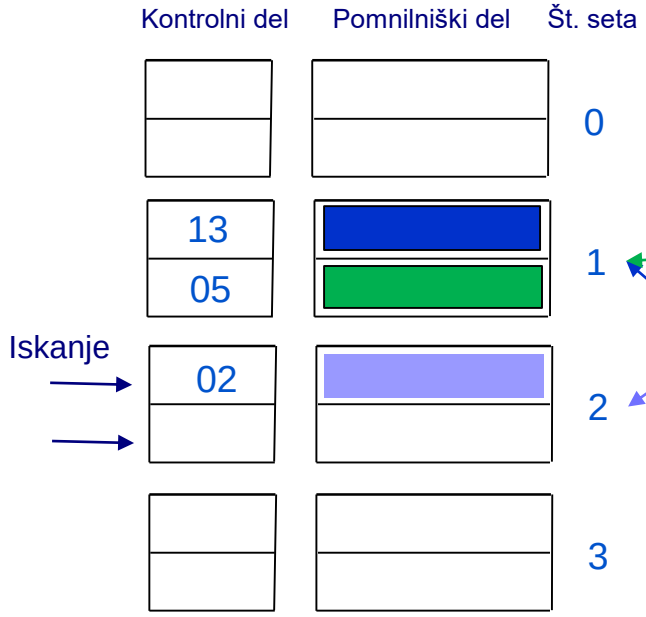
Set-asociativni predpomnilnik

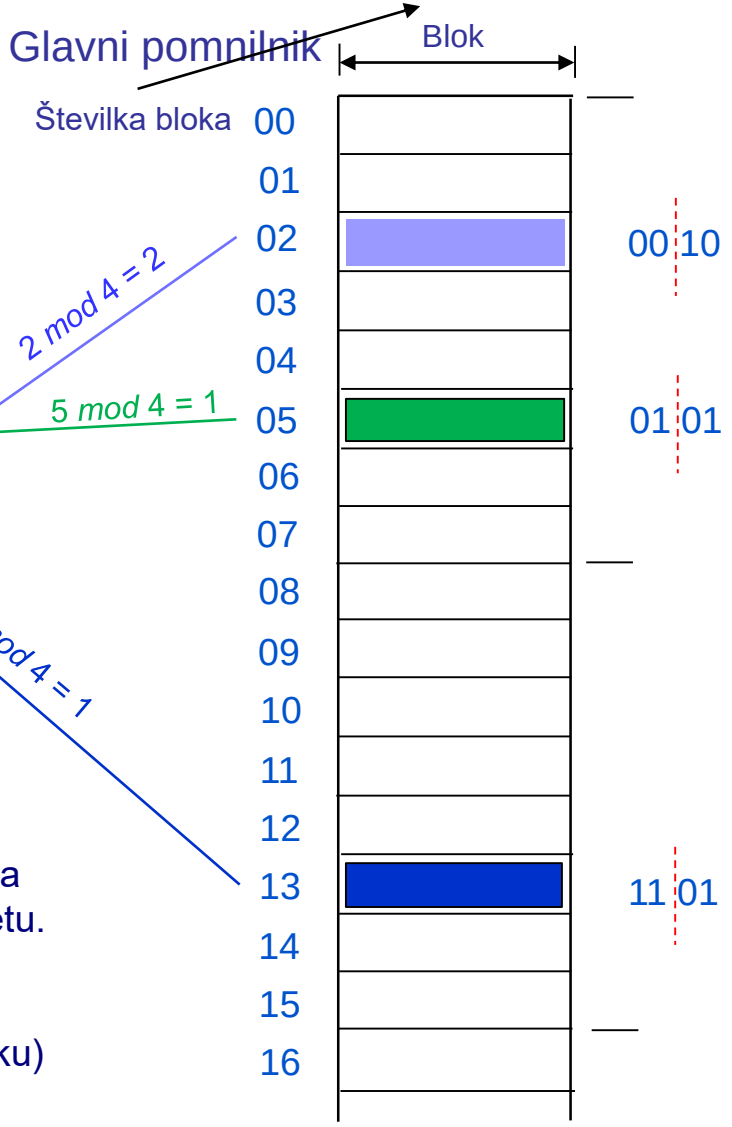
- Celoten predpomnilnik je razdeljen na več delov, ki jih imenujemo seti.
- Vsak set je majhen čisti asociativni predpomnilnik.
- Iskanje bloka znotraj seta je hitro (asociativni kontrolni del), iskanje v katerem setu je blok pa ne.
- Zato se določen blok iz glavnega pomnilnika lahko preslika samo v točno določen set (da ga ni potrebno iskati med seti), znotraj seta pa kamorkoli.
- Število blokov v setu imenujemo stopnja asociativnosti E.
- Večja kot je stopnja asociativnosti, večja je verjetnost zadetka.

Preslikava bloka pri set-asociativnem predpomnilniku



Set-asociativni predpomnilnik
velikost 8 blokov, razdeljen na 4 sete.
2 bloka v setu (= stopnja asociativnosti E=2)





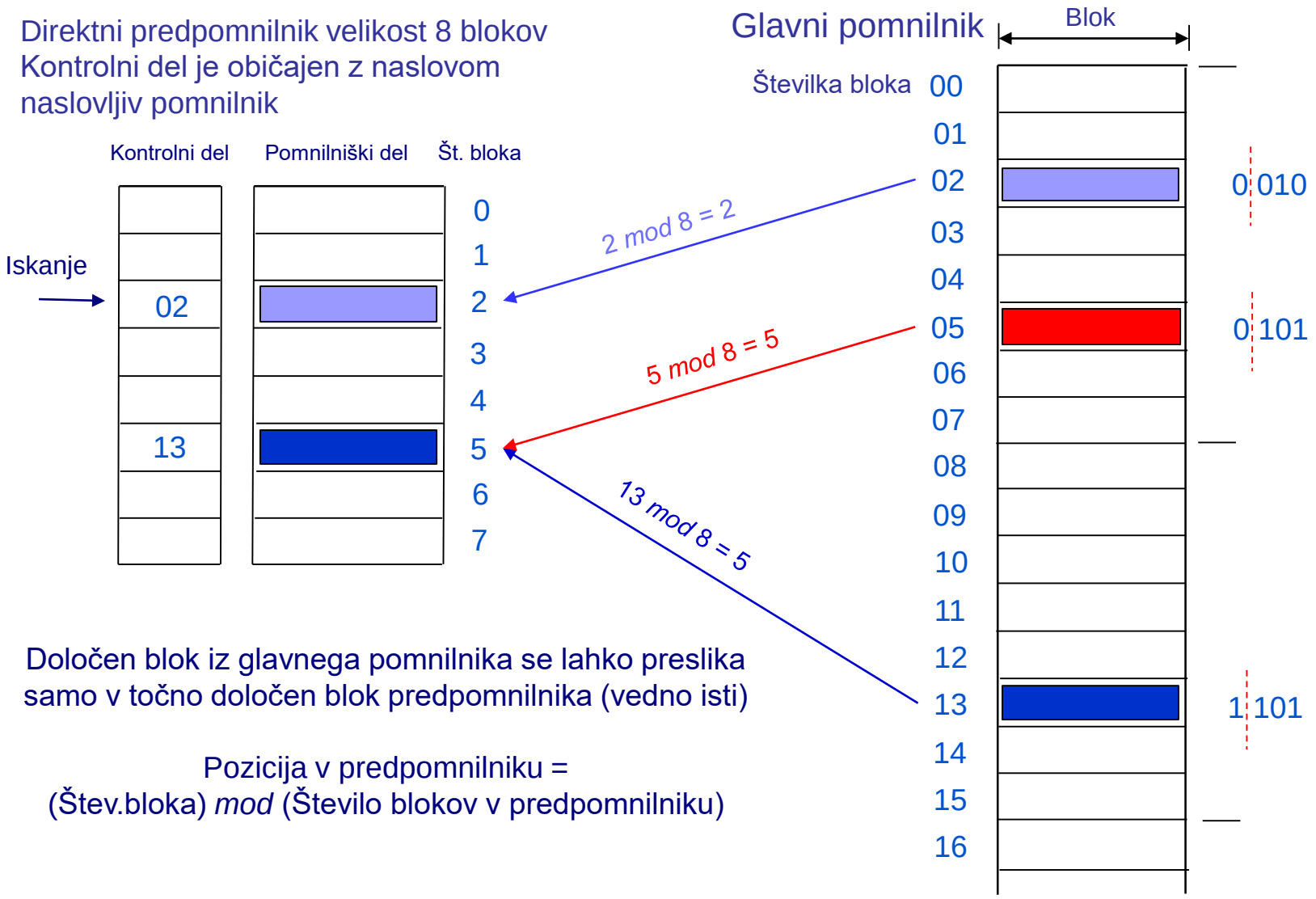
Določen blok iz glavnega pomnilnika se lahko preslika samo v točno določen set, vendar v katerikoli blok v setu.

Številka seta =
(Štev.bloka in seta) *mod* (Število setov v predpomnilniku)

- Direktni predpomnilnik
  - Celoten kontrolni del predpomnilnika je običajen z naslovom naslovljiv pomnilnik – statični RAM
  - Zato je iskanje bloka nemogoče (prepočasno).
  - Določen blok iz glavnega pomnilnika se zato lahko preslika samo v točno določen blok predpomnilnika, da iskanje ni potrebno.
  - Če je v predpomnilniku blok v katerega se mora preslikati blok iz glavnega pomnilnika poln, je potrebna zamenjava bloka.
  - Verjetnost zadetka je zato v direktnem predpomnilniku v primerjavi z enako velikim set-asociativnim predpomnilnikom precej manjša..

Preslikava bloka pri direktnem predpomnilniku

Direktni predpomnilnik velikost 8 blokov
Kontrolni del je običajen z naslovom naslovljiv pomnilnik

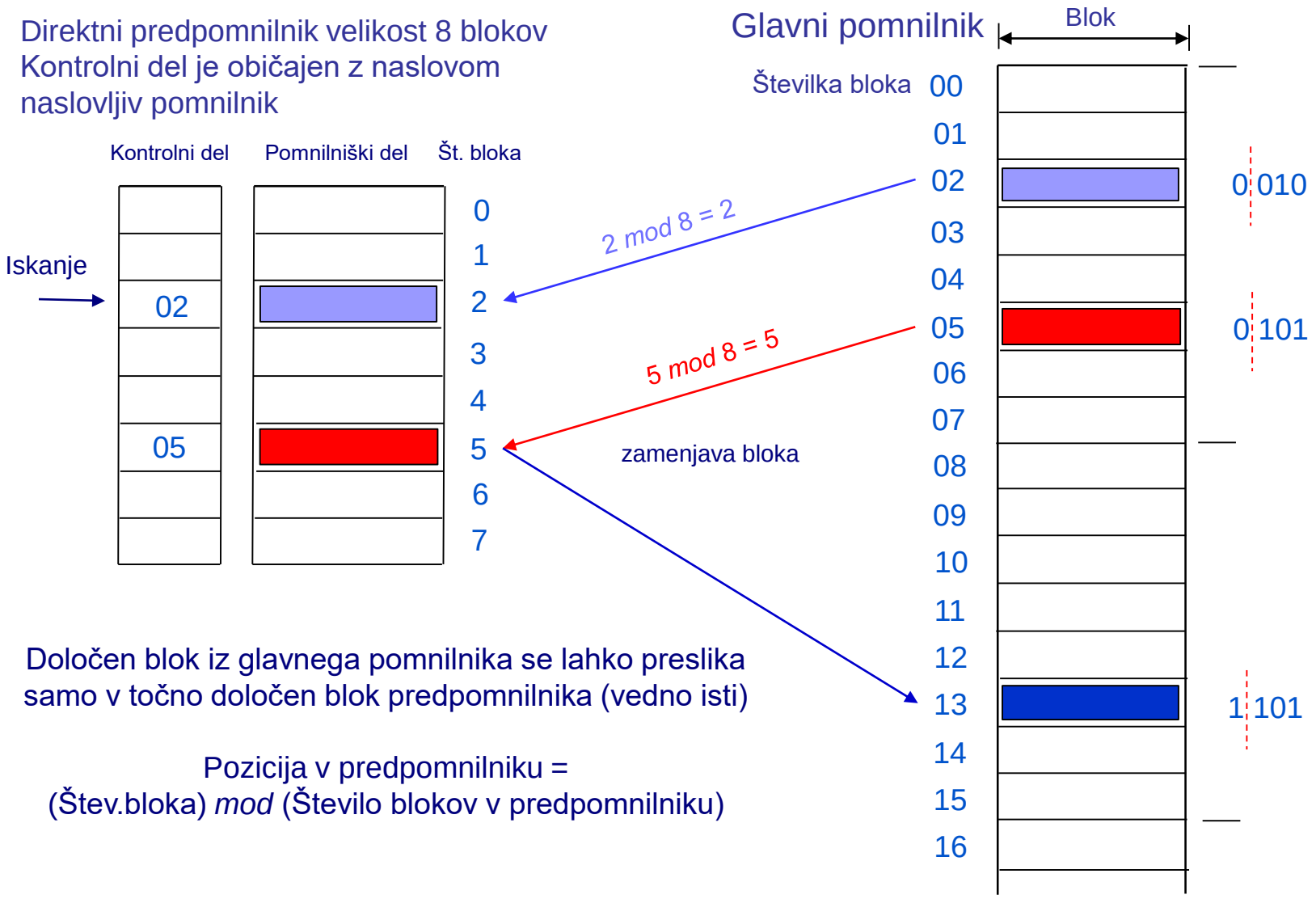


Določen blok iz glavnega pomnilnika se lahko preslika samo v točno določen blok predpomnilnika (vedno isti)

Pozicija v predpomnilniku =
(Štev.bloka) *mod* (Število blokov v predpomnilniku)

Preslikava bloka pri direktnem predpomnilniku

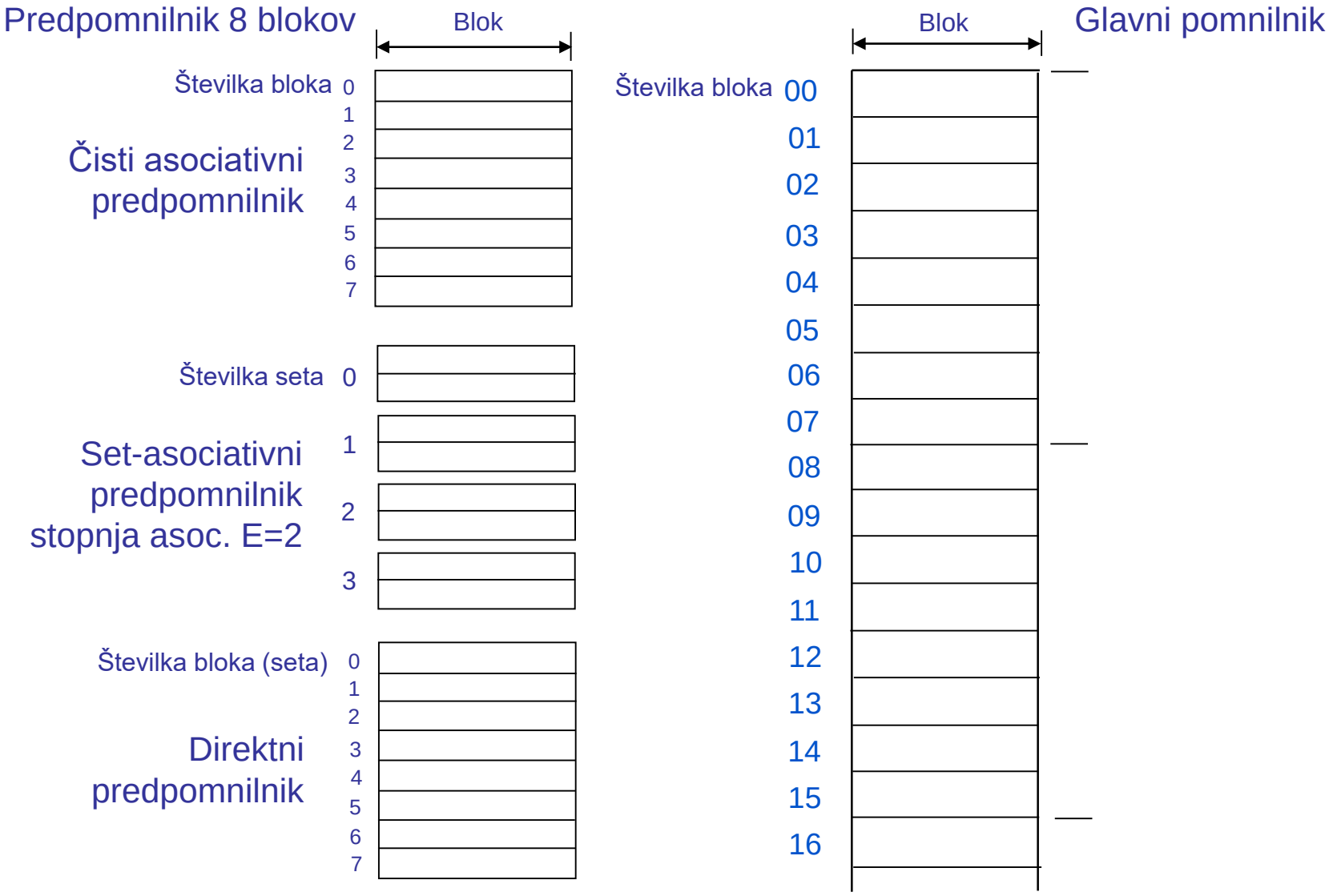
Direktni predpomnilnik velikost 8 blokov
Kontrolni del je običajen z naslovom naslovljiv pomnilnik

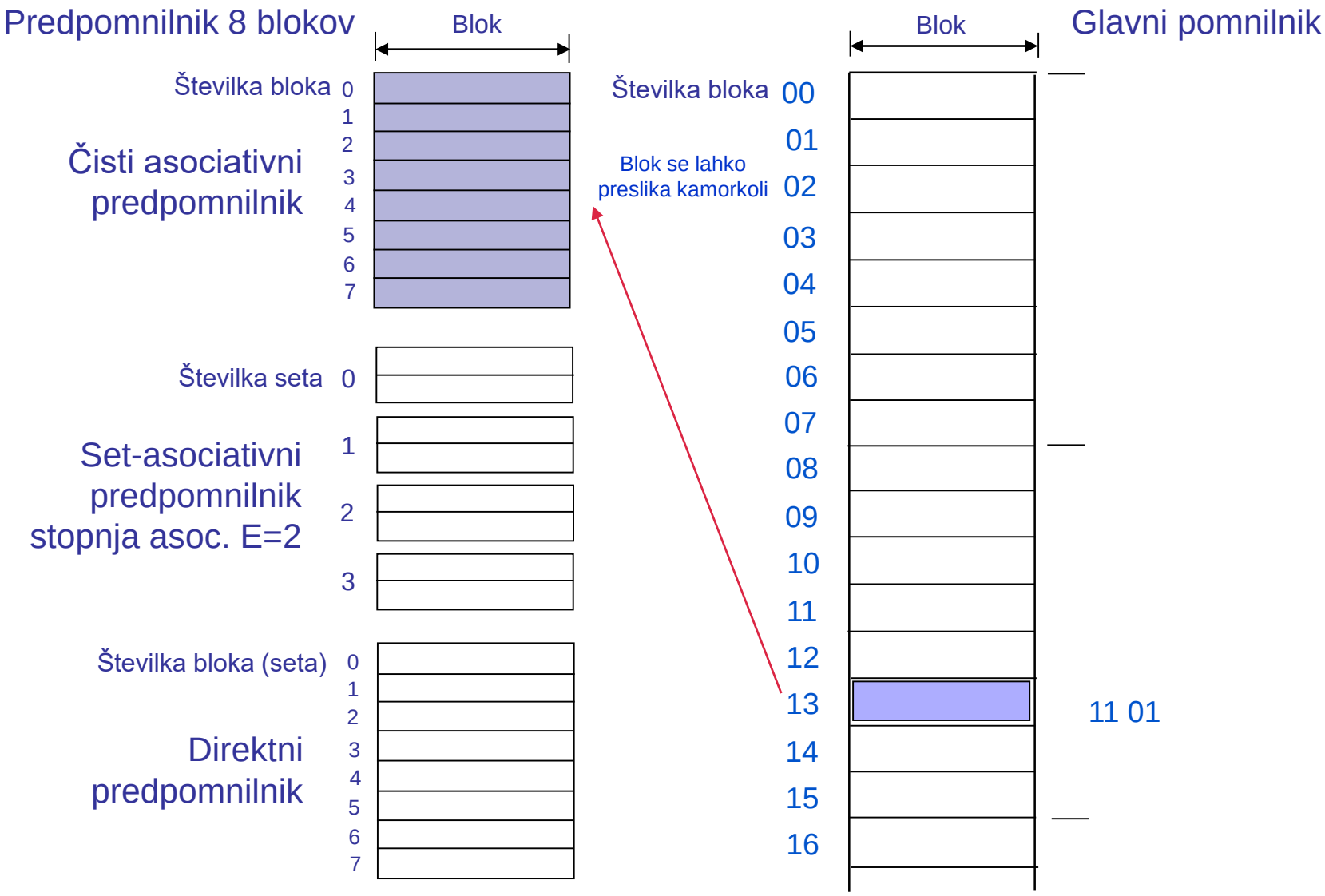


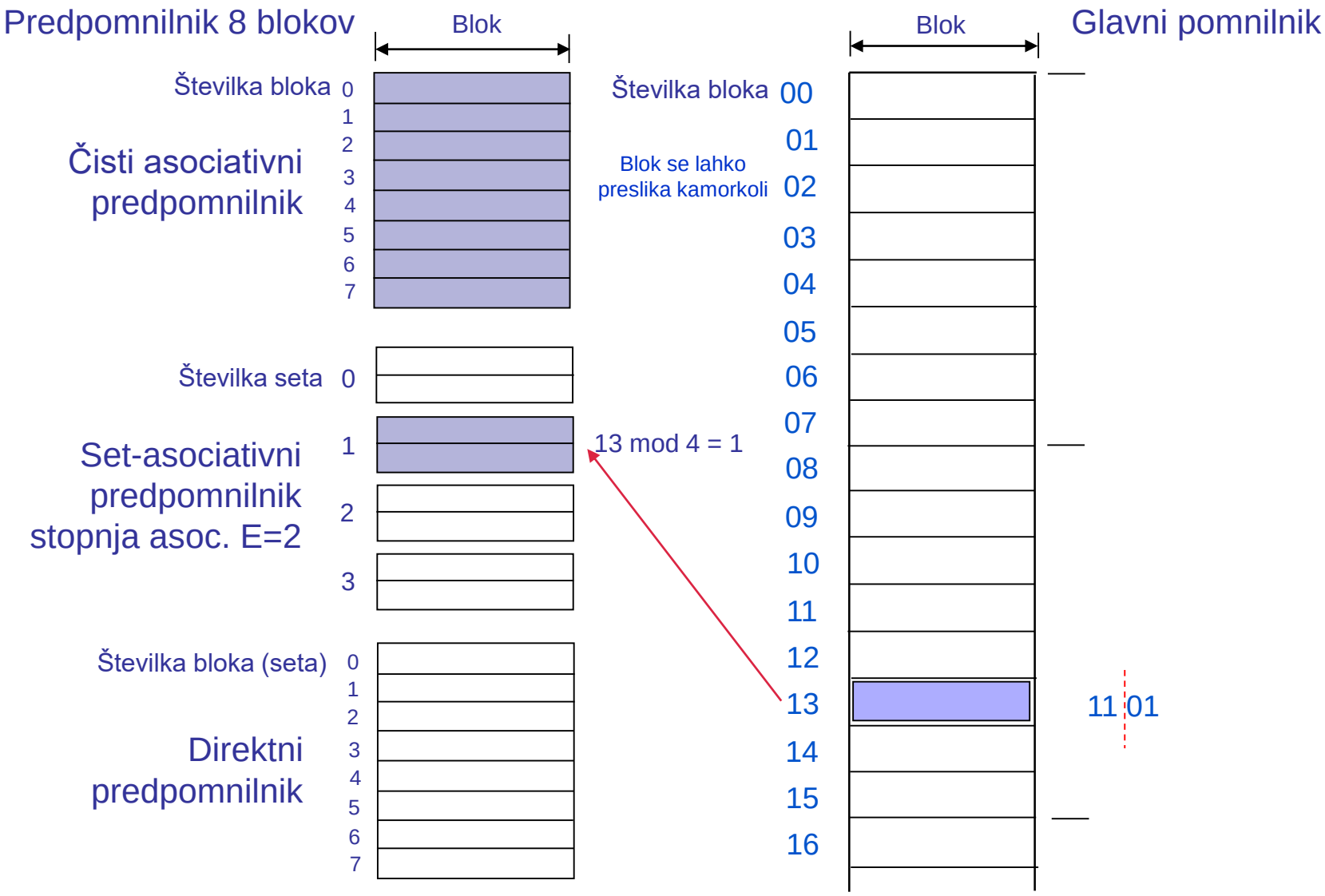
Določen blok iz glavnega pomnilnika se lahko preslika samo v točno določen blok predpomnilnika (vedno isti)

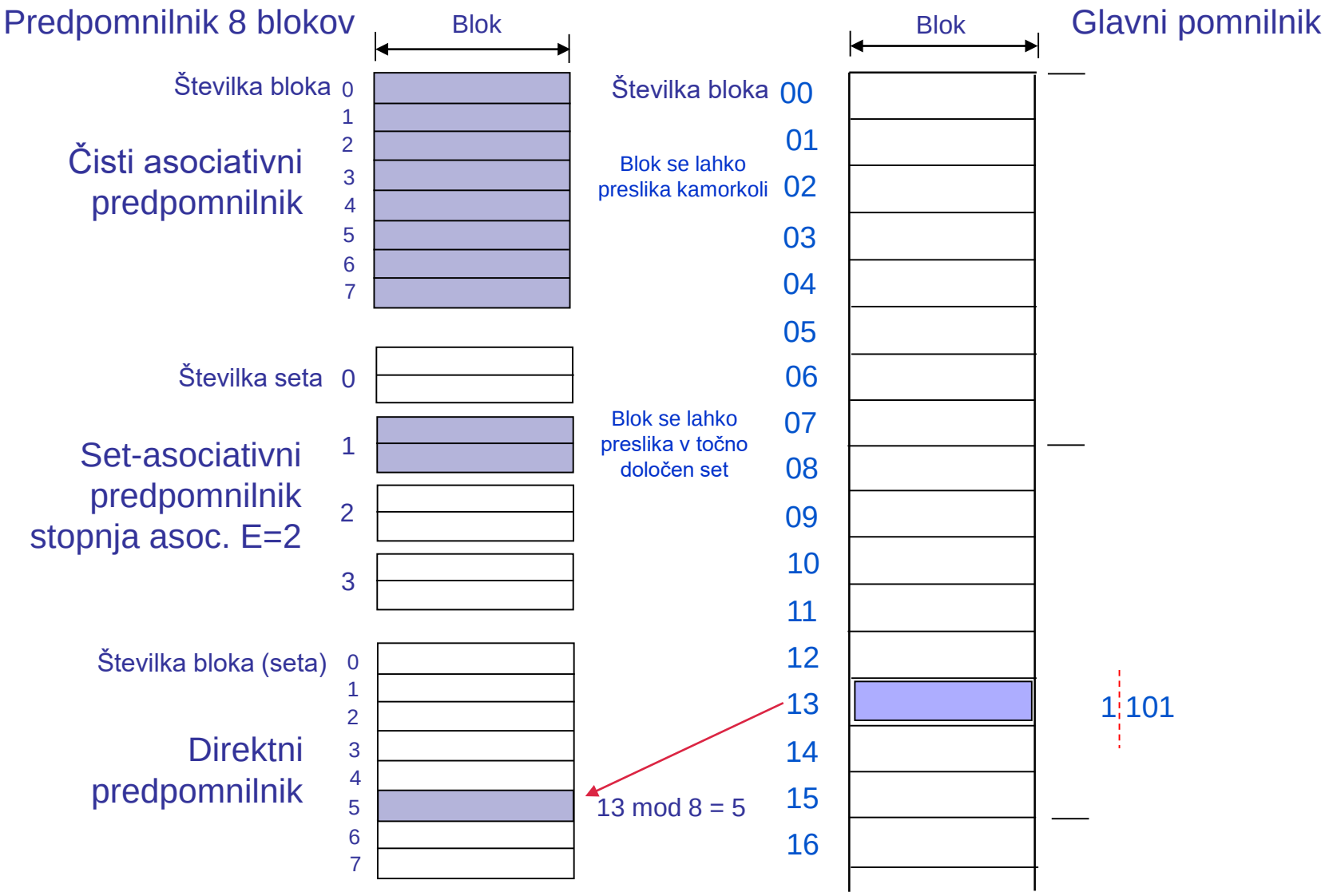
Pozicija v predpomnilniku =
(Štev.bloka) *mod* (Število blokov v predpomnilniku)

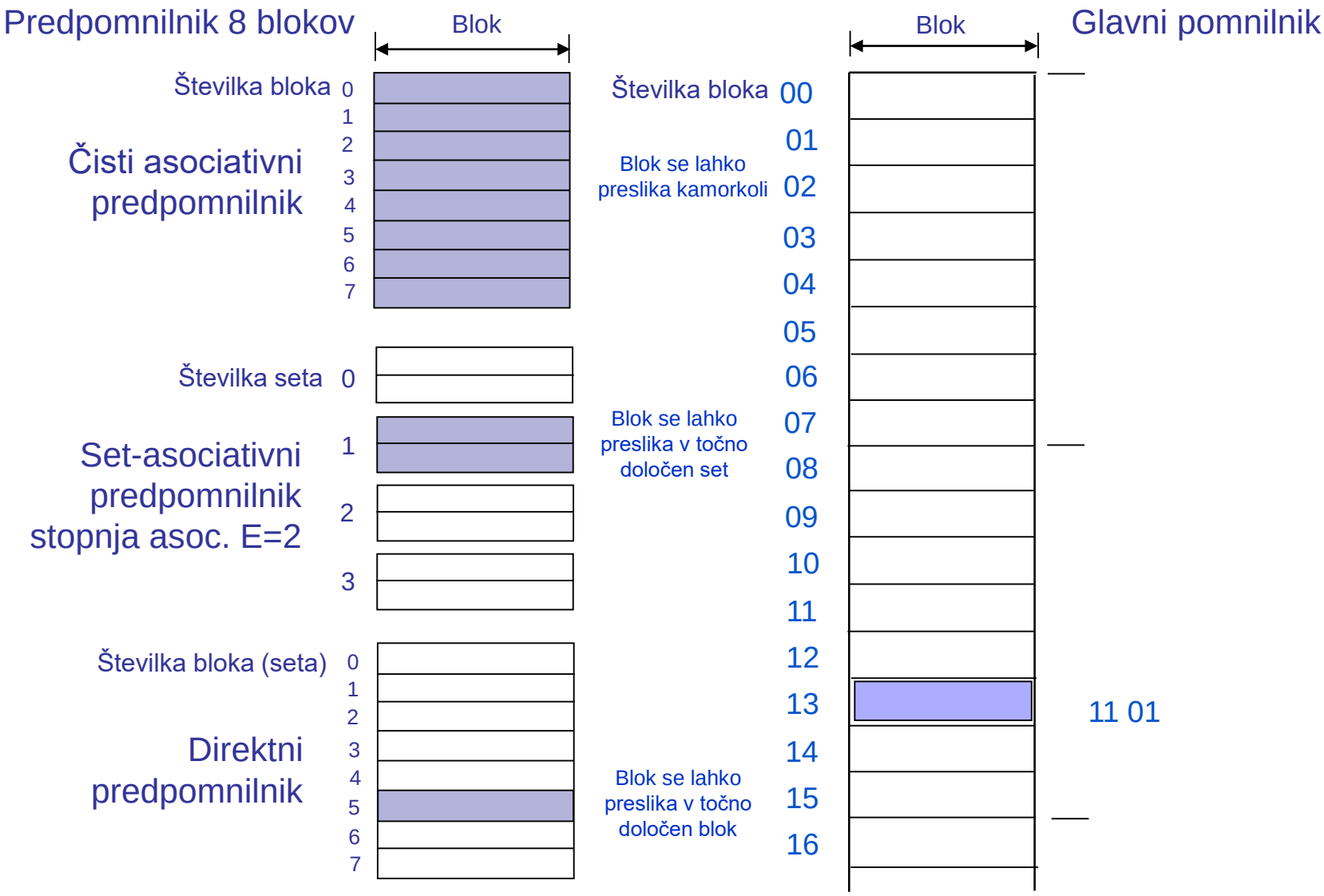
Predpomnilnik - preslikava bloka v predpomnilnik pri različnih vrstah predpomnilnikov









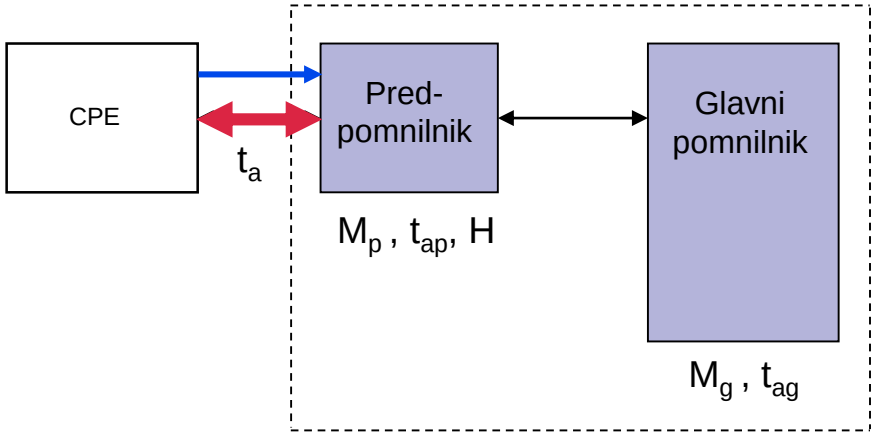


Vpliv predpomnilnika na hitrost delovanja CPE

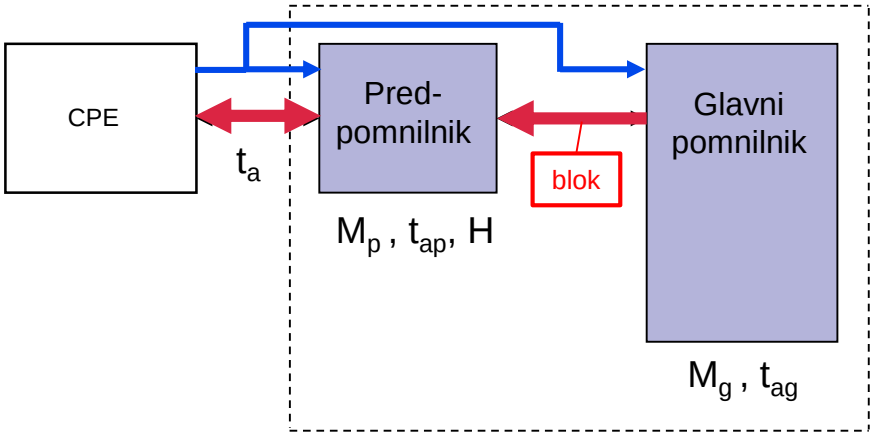
- Dostop do predpomnilnika:
  - Zadetek:
    - branje - običajno 1 urina perioda,
    - pisanje - branje bloka,

      - spreminjanje vsebine,

      - pisanje bloka nazaj - tipično ena urina perioda več.



- Zgrešitev:
  - dostop do glavnega pomnilnika,
  - prenos bloka do predpomnilnika,
  - pisanje bloka v predpomnilnik,
  - sledi branje ali pisanje kot pri zadetku,
  - če je predpomnilnik poln, je potrebna še zamenjava bloka.



Za vse te operacije pri zgrešitvi je potrebnih od 10 do 100 urinih period (zgrešitvena kazen).

- Predpomnilniške zgrešitve zmanjšujejo hitrost delovanja CPE, oziroma povečujejo CPI.

- Idealni CPI (CPI_I) – brez upoštevanja zgrešitev v predpomnilniku

- Realni CPI (CPI_R) – z upoštevanjem zgrešitev v predpomnilniku

- Realni CPI z upoštevanjem zgrešitev v predpomnilniku je:

$$CPI_R = CPI_I + M_I(1-H) * Zgrešitvena\_kazen$$

CPI_R - realni CPI
CPI_I - idealni CPI brez zgrešitev v predpomnilniku
M_I - povprečno število pomn. dostopov na ukaz

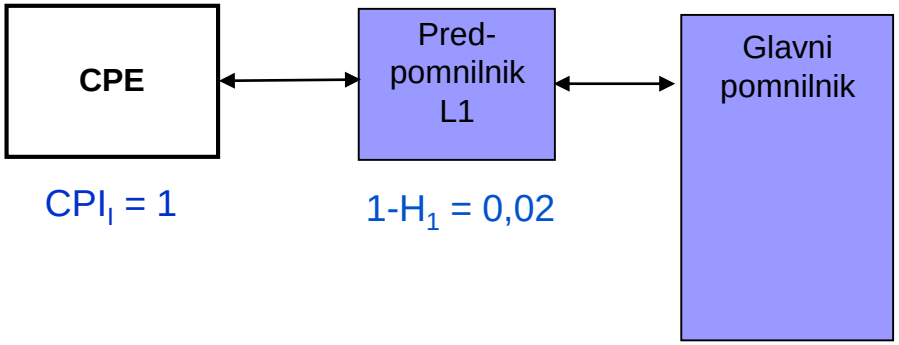
- Realni čas izvajanja programa z *N* ukazi pa je:

$$CPE_{čas} = N * CPI_R * t_{CPE}$$

Primer: Vpliv predpomnilnika L2 na hitrost CPE

- Procesor ima idealni $CPI_I = 1$, če v ukaznem predpomnilniku L1 ni zgrešitev
- Frekvenca ure procesorja $f_{CPE} = 4$ GHz
- Verjetnost zgrešitve v predpomnilniku L1 je 2%
- Zgrešitvena kazen je 100 ns (čas za prenos bloka iz glavnega pomnilnika)
- Če v hierarhijo dodamo še predpomnilnik L2 z zgrešitveno kaznijo 5 ns (čas za prenos bloka t_{B2}), je globalna verjetnost zgrešitve v L2 0,5% (splošna verjetnost, da bo potreben dostop do glavnega pomnilnika)
- Kolikokrat hitrejše je delovanje procesorja, če v hierarhijo dodamo predpomnilnik drugega nivoja L2?

Dvonivojska pomnilniška hierarhija (brez L2)



Zgrešitvena kazen t_{B2} (čas za prenos bloka iz glavnega pomnilnika v predpomnilnik)

t_{B2}= 100 ns = 400 [up]
[up] - urine periode

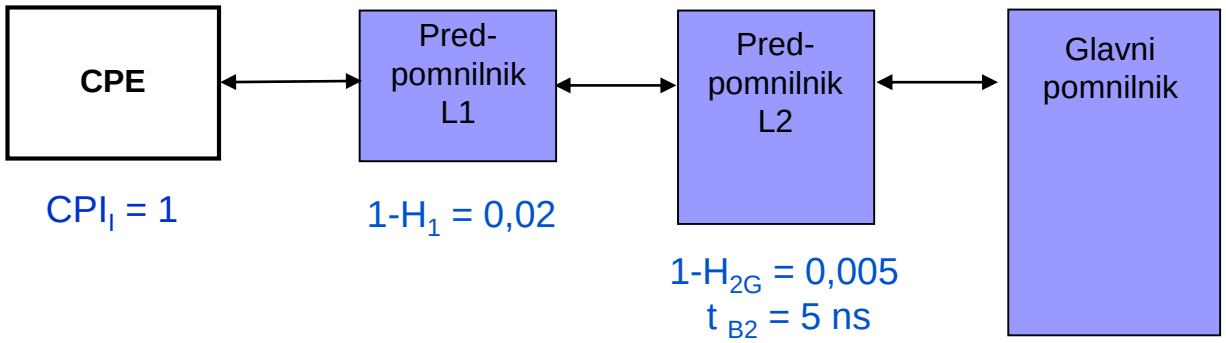
$$t_{CPE} = \frac{1}{f_{CPE}} = \frac{1}{4 \cdot 10^9}[s] = 0{,}25 \cdot 10^{-9}[s] = 0{,}25[ns]$$ Čas trajanja ene urine periode

$$t_B = \frac{100[ns]}{0{,}25[\frac{ns}{up}]} = 400[up]$$

$$CPI_R(L1) = CPI_I + (1 - H_1) \cdot t_{B2} = 1[up] + 0{,}02 \cdot 400[up] = 9[up]$$

Zaradi zgrešitev v predpomnilniku se CPI iz 1 poveča na 9 urinih period

Trinivojska pomnilniška hierarhija



Zgrešitvena kazen t_{B3} (čas za prenos bloka iz glavnega pomnilnika v predpomnilnik)

t_{B3}= 100 ns = 400 [up]
[up] - urine periode

$1-H_{2G}$ predstavlja globalno verjetnost zgrešitve glede na vse pomn. dostope in vključuje lokalni verjetnosti zgrešitve L1 in L2
(v glavni pomnilnik dostopamo, ko se zgodita obe zgrešitvi)

$$t_{B2} = \frac{5[ns]}{0{,}25[\frac{ns}{up}]} = 20[up]$$

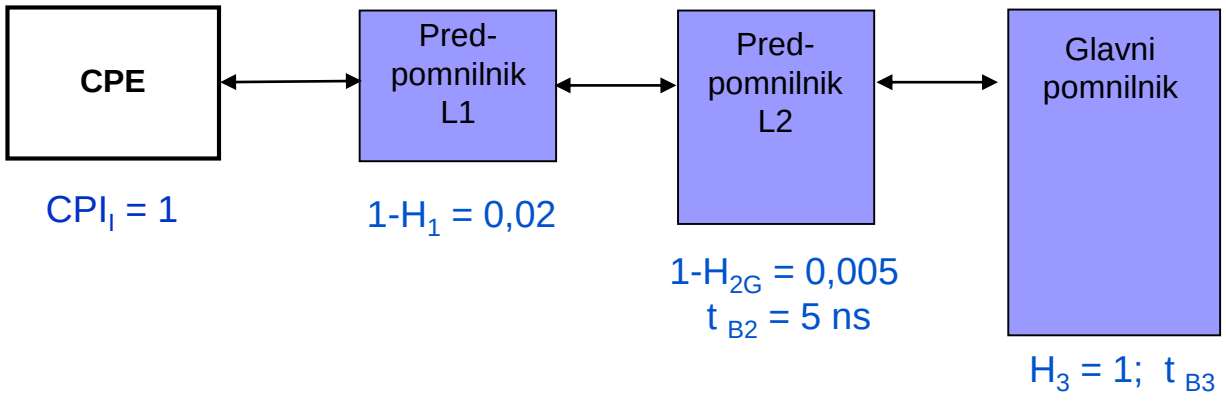
Čas za prenos bloka iz L2 v L1

$$CPI_R(L1, L2) = CPI_I + (1 - H_1) \cdot t_{B2} + (1 - H_{2G}) \cdot t_{B3} =$$
$$= 1[up] + 0{,}02 \cdot 20[up] + 0{,}005 \cdot 400[up] = 1 + 0{,}4 + 2 = 3{,}4[up]$$

$$Pohitritev = \frac{CPI_R(L1)}{CPI_R(L1, L2)} = \frac{9}{3{,}4} = 2{,}6$$

Če dodamo predpomnilnik L2, se hitrost izvajanja ukazov 2,6-krat poveča

Trinivojska pomnilniška hierarhija



Zgrešitvena kazen t_{B3} (čas za prenos bloka iz glavnega pomnilnika v predpomnilnik)

t_{B3}= 100 ns = 400 [up]
[up] - urine periode

1-H_{2G} predstavlja globalno verjetnost zgrešitve glede na vse pomn. dostope in vključuje lokalni verjetnosti zgrešitve L1 in L2
(v glavni pomnilnik dostopamo, ko se zgodita obe zgrešitvi)

Primerjava izračuna s pomočjo lokalne ali globalne verjetnosti

$$CPI_R(L1,L2) = CPI_I + (1-H_1)\cdot(t_{B2} + (1-H_{2L})\cdot t_{B3}) =$$
$$= CPI_I + (1-H_1)\cdot t_{B2} + (1-H_1)(1-H_{2L})\cdot t_{B3}$$
$$= CPI_I + (1-H_1)\cdot t_{B2} + (1-H_{2G})\cdot t_{B3}$$
$$= 1[up] + 0,02\cdot 20[up] + 0,005\cdot 400[up] = 1 + 0,4 + 2 = 3,4[up]$$

V predpomnilniku L1 sta lokalna in globalna verjetnost zgrešitve enaki, ker vsi dostopi pridejo v L1 PP.

V predpomnilniku L2 pa je lokalna verjetnost zgrešitve *1-H_{2L}* izražena glede na dostope samo v PP L2, globalna pa glede na vse pomn. dostope..

Pri večnivojskih hierarhijah so globalne verjetnosti običajno bolj uporabne, saj vključujejo tudi vpliv predhodnih nivojev (lokalne se nanašajo samo na določen nivo)

Primer izračuna vpliva predpomnilnika L2 na hitrost CPE : lokalne in globalne verjetnosti

Globalne :

$$CPI_R = CPI_I + (1 - H_1) * t_{BL2} + (1 - H_{2G}) * t_{BG}$$

$$CPI_R = 1 + 0.02*20 + 0.005*400 = 3.4\ t_{CPE}$$

Lokalne :

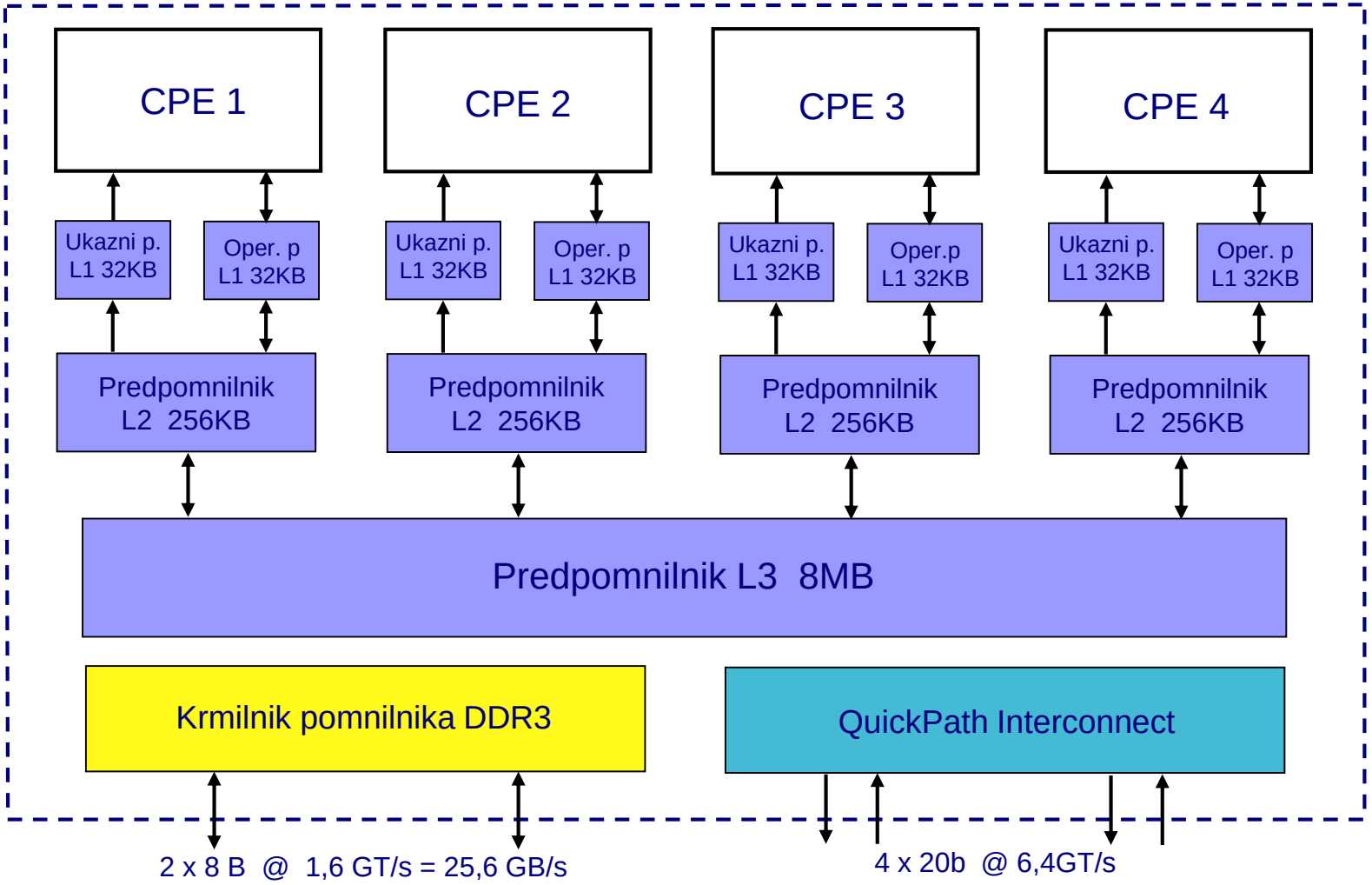
$H_{1L} = H_1$;

$(1-H_{2L})=(1-H_{2G})/(1-H_1)=0.005/0.02 = 0.25$

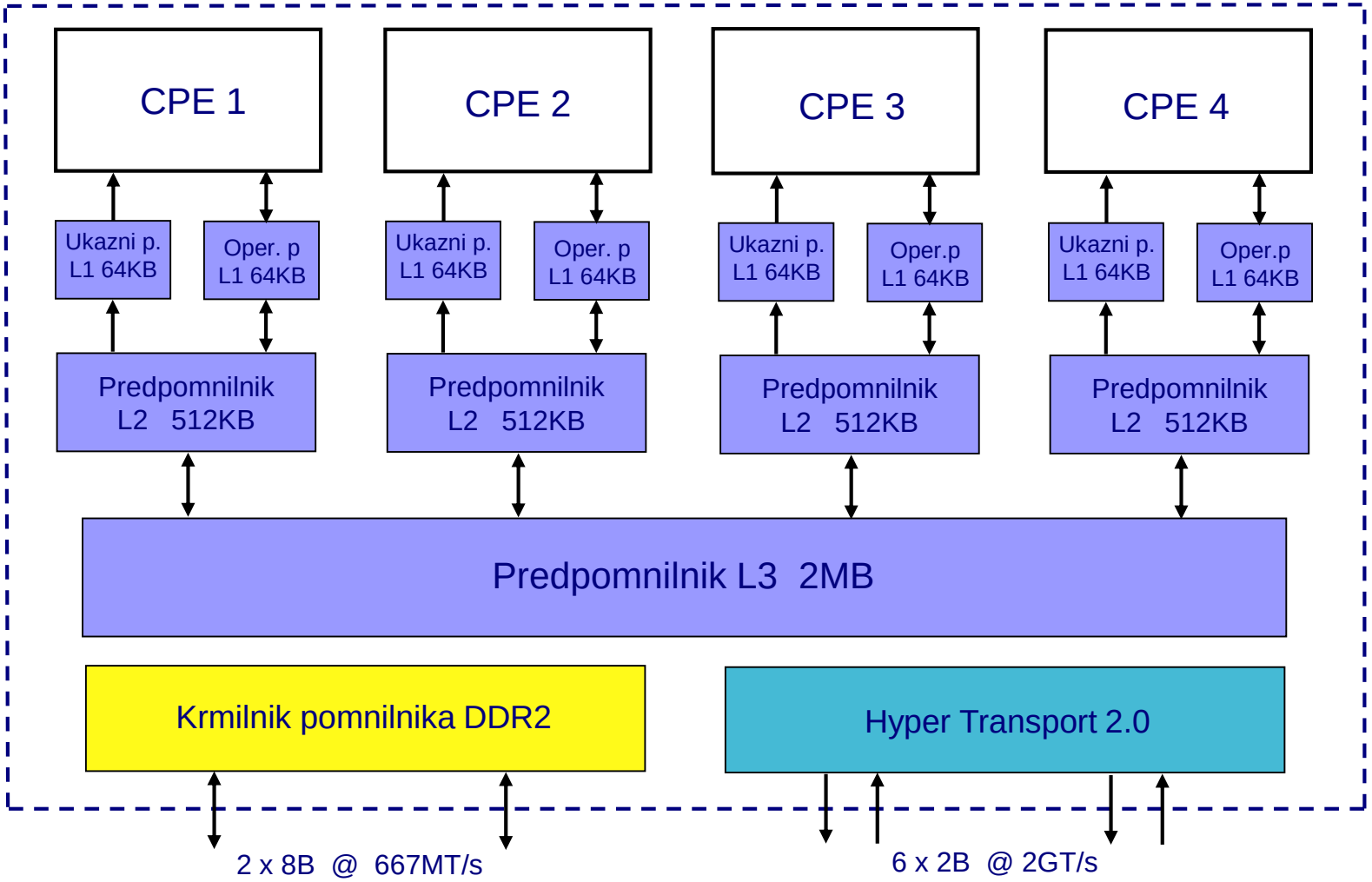
$$CPI_R = CPI_I + (1 - H_1) * (t_{BL2} + (1 - H_{2L}) * t_{BG})$$

$$CPI_R = 1 + 0.02* (20 + 0.25*400) = 3.4\ t_{CPE}$$

Zgradba 4-jedrnega procesorja Intel Core i7 (Haswell)



Zgradba 4-jedrnega procesorja AMD Opteron (Barcelona)



Loop Interchange

Some programs have nested loops that access data in memory in nonsequential order. Simply exchanging the nesting of the loops can make the code access the data in the order in which they are stored. Assuming the arrays do not fit in the cache, this technique reduces misses by improving spatial locality; reordering maximizes use of data in a cache block before they are discarded. For example, if `x` is a two-dimensional array of size [5000,100] allocated so that `x[i,j]` and `x[i,j+1]` are adjacent (an order called row major because the array is laid out by rows), then the two pieces of the following code show how the accesses can be optimized:

```
/* Before */
for (j = 0; j < 100; j = j + 1)
      for (i = 0; i < 5000; i = i + 1)
            x[i][j] = 2 * x[i][j];
/* After */
for (i = 0; i < 5000; i = i + 1)
      for (j = 0; j < 100; j = j + 1)
            x[i][j] = 2 * x[i][j];
```

The original code would skip through memory in strides of 100 words, while the revised version accesses all the words in one cache block before going to the next block. This optimization improves cache performance without affecting the number of instructions executed.

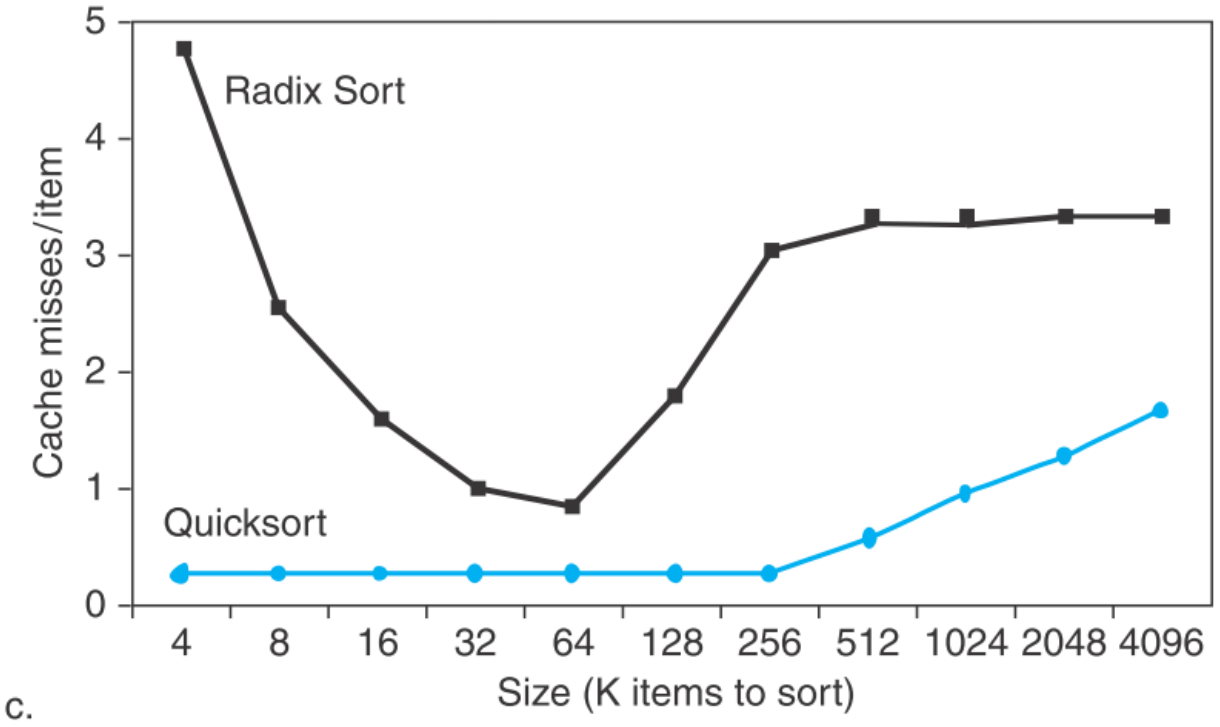
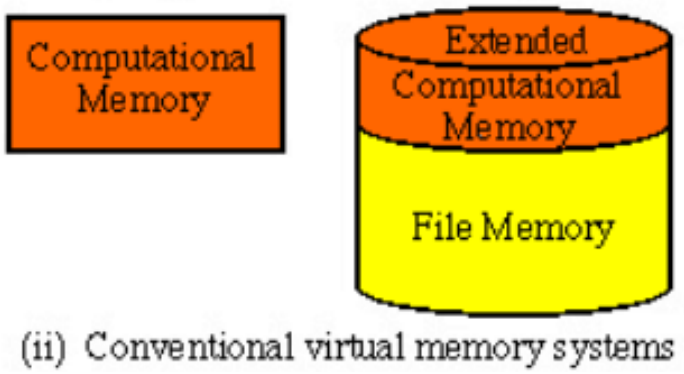


FIGURE 5.19 Comparing Quicksort and Radix Sort by (a) instructions executed per item sorted, (b) time per item sorted, and (c) cache misses per item sorted. These data are from a paper by LaMarca and Ladner [1996]. Due to such results, new versions of Radix Sort have been invented that take memory hierarchy into account, to regain its algorithmic advantages (see Section 5.15). The basic idea of cache optimizations is to use all the data in a block repeatedly before they are replaced on a miss.

9.4 Navidezni pomnilnik

- Navidezni pomnilnik (virtual memory) $\Rightarrow$ prostor v pomožnem pomnilniku (SSD ali magnetni disk), ki je za uporabnika videti kot glavni pomnilnik.
- Dostop do pomožnega pomnilnika poteka z V/I ukazi, oziroma z V/I programi.
- Prenosi med glavnim in navideznim pomnilnikom so za uporabnika nevidni ($\Rightarrow$ navidezni pomnilnik)
- Potrebna je dodatna logika v CPE in programska oprema

- Navidezni pomnilnik ima danes večina računalnikov, razlog ni več samo premajhen glavni pomnilnik kot včasih, temveč tudi:
  - Veliko nižja cena pomnilnika na pomožnem pomnilniku.
  - Enostavna rešitev pozicijske neodvisnosti programov.
  - Zaščita pomnilnika.



- Prostor na pomožnem pomnilniku:
  - Prostor za navidezni pomnilnik.
  - Prostor za shranjevanje datotek (običajno precej večji).

- Čas dostopa in prenosa informacije (= zgrešitvena kazen) iz pomožnega pomnilnika v glavni pomnilnik je zelo dolg.

- Rešitve za zmanjšanje vpliva zelo velike zgrešitvene kazni pri navideznem pomnilniku:

  - Bloki morajo biti veliki (4KB, 8KB, do 64KB in več)

  - Vsak blok se lahko preslika v poljuben blok glavnega pomnilnika

  - Zamenjava blokov se ob zgrešitvah opravi programsko in ne strojno kot pri predpomnilniku

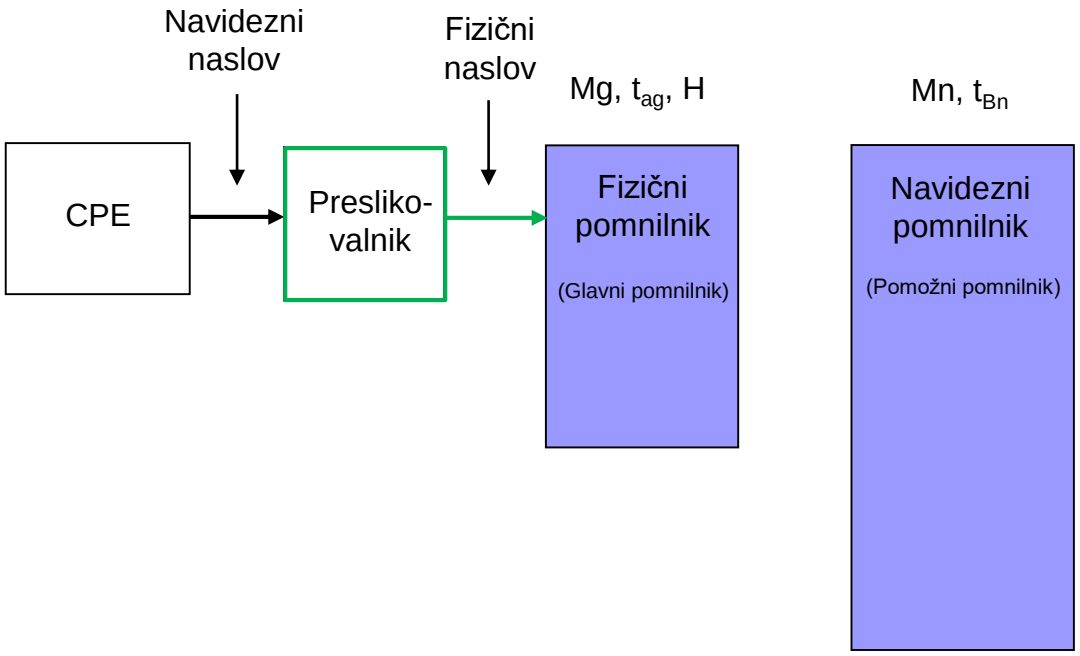
- Pomnilniški naslov iz CPE = **navidezni naslov** (ker se nanaša na navidezni pomnilnik).
- V povezavi z navideznim pomnilnikom imenujemo glavni pomnilnik **fizični pomnilnik.**
- Naslov, ki se nanaša na glavni pomnilnik = **fizični naslov**

- Pri vsakem pomnilniškem dostopu:
  Navidezni naslov → preslikava → fizični naslov

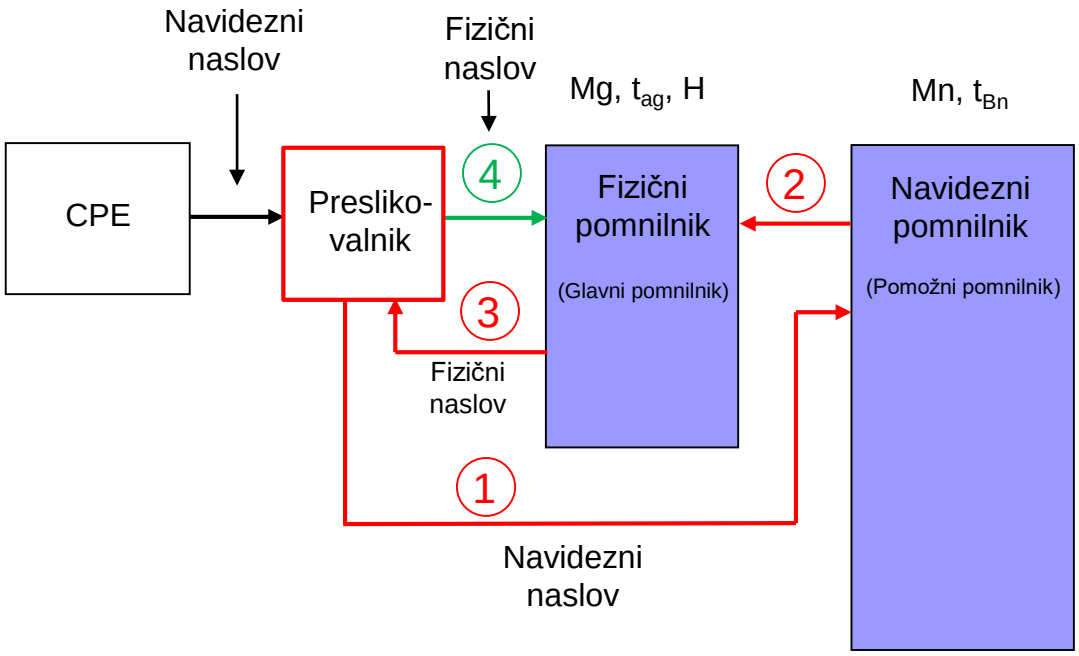
- Fizični naslov obstaja, če je v glavnem (fizičnem) pomnilniku zadetek.

- Pri večini računalnikov se fizični naslov (ne navidezni) uporabi za dostop do predpomnilnika.

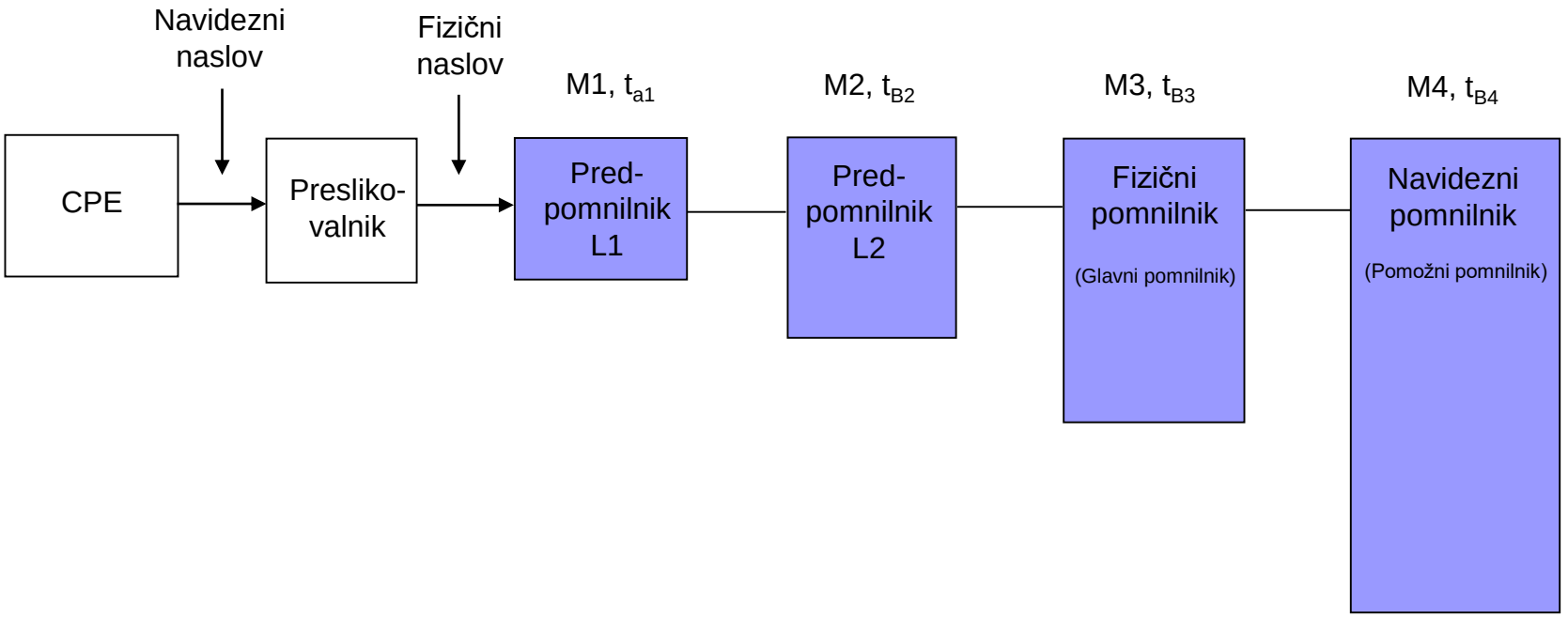
Naslovljena informacija je v fizičnem pomnilniku – zadetek
Verjetnost zadetka *H*



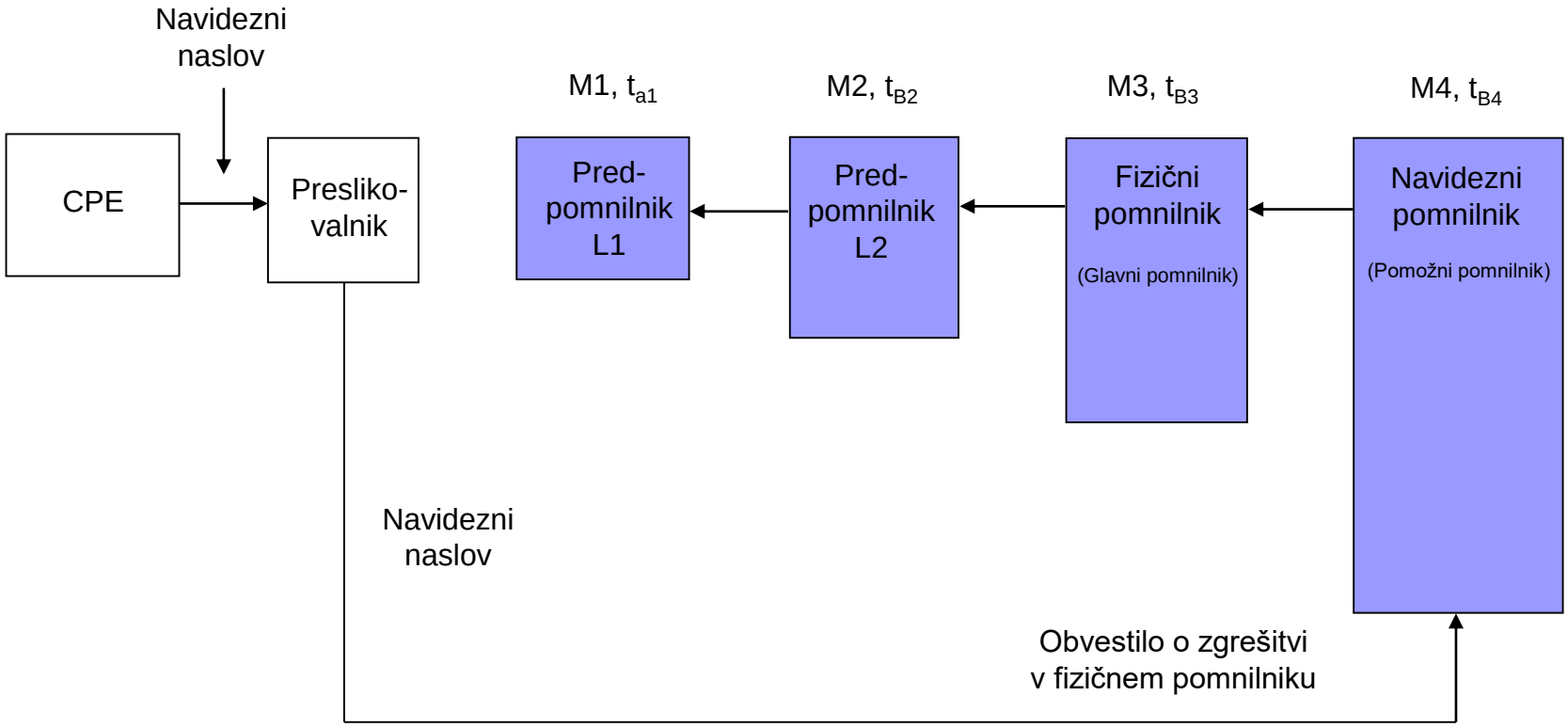
Naslovljena informacije ni v fizičnem pomnilniku – zgrešitev
Verjetnost zgrešitve *1-H*



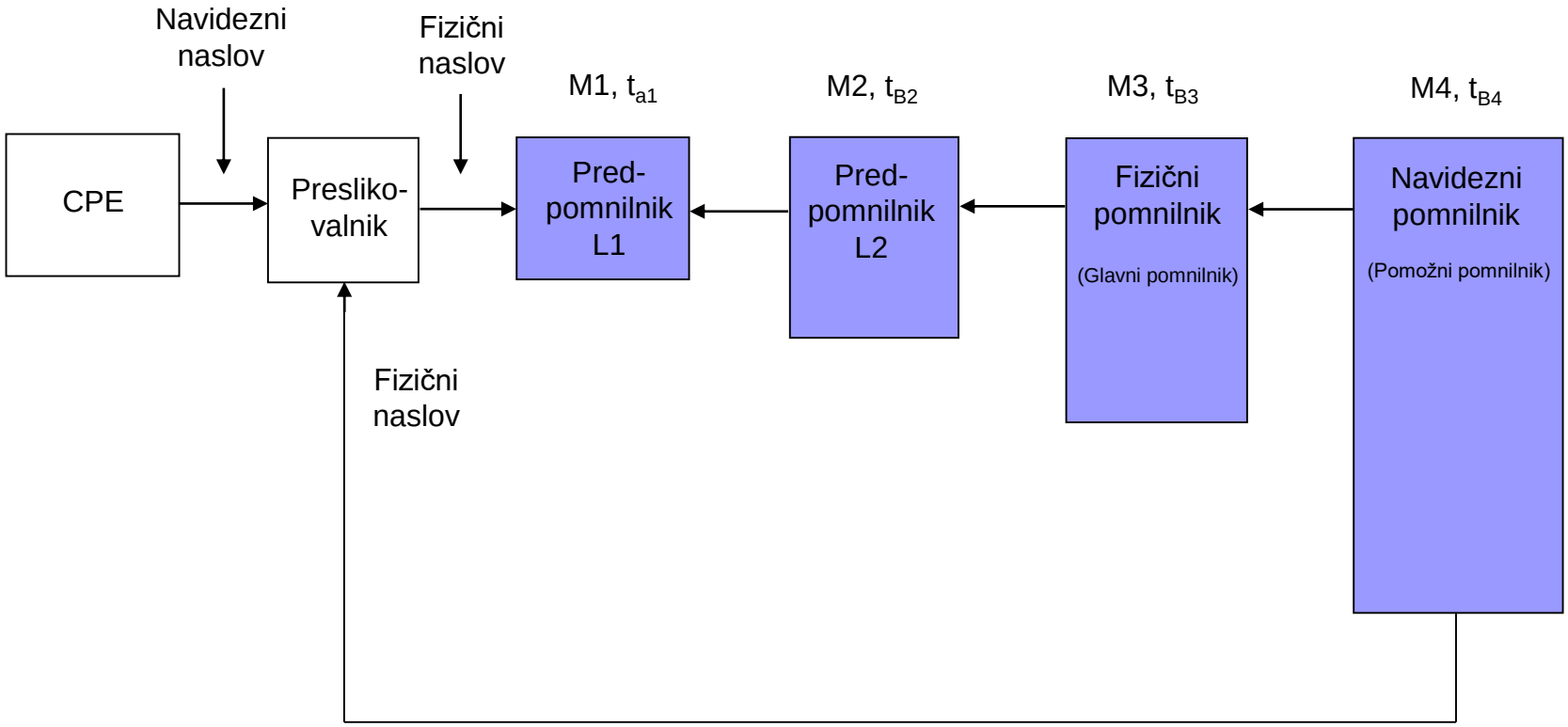
Celotna hierarhija
Naslovljena informacija je v fizičnem pomnilniku - zadetek



Celotna hierarhija
Naslovljene informacije ni v fizičnem pomnilniku - zgrešitev



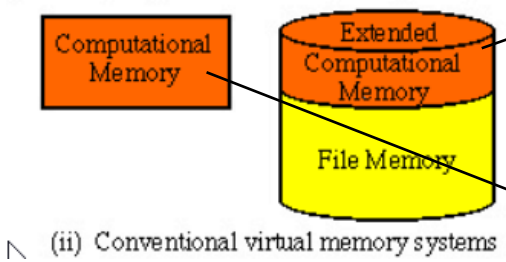
Celotna hierarhija
Naslovljene informacije ni v fizičnem pomnilniku - zgrešitev



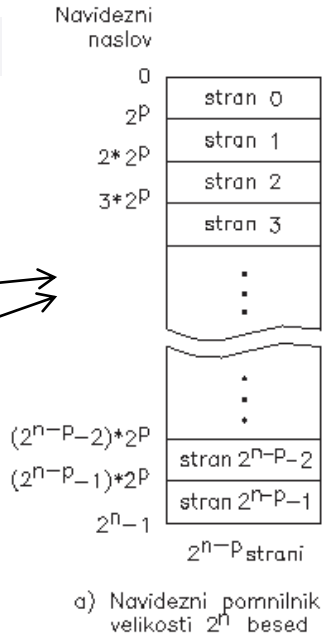
- Preslikovalna funkcija se vzpostavi programsko (operacijski sistem)
- Pri vklopu računalnika je zato preslikovanje navideznih naslovov v fizične potrebno izklopiti (ker še ne deluje).
- Preslikovanje je možno izklopiti tudi kadarkoli, v tem primeru velja: navidezni naslov = fizični naslov

Navidezni pomnilnik z ostranjevanjem (paging)

- Pomožni pomnilnik ⇒ razdeljen na strani (pages):
  - strani ⇒ bloki enakih velikosti.

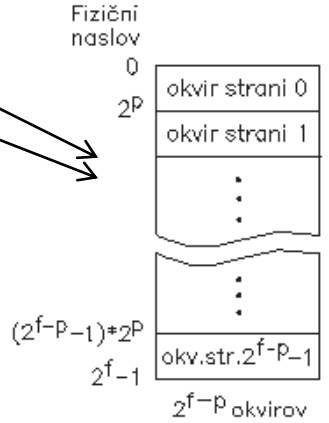


(ii) Conventional virtual memory systems



a) Navidezni pomnilnik velikosti 2^n besed

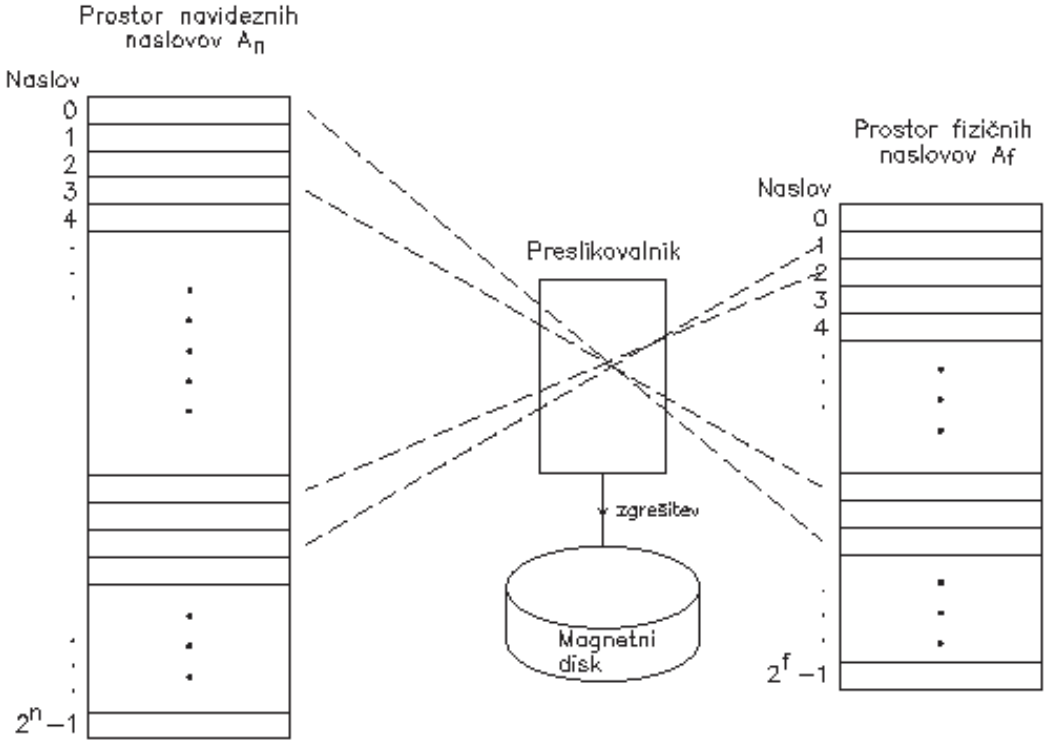
- Glavni pomnilnik ⇒ razdeljen na okvirje strani (page frames):
  - okvirji strani ⇒ enako veliki bloki kot v pomožnem pomnilniku.



b) Fizični pomnilnik velikosti 2^f besed

- Število strani v navideznem pomnilniku je običajno veliko večje kot število okvirov v glavnem pomnilniku:
  - iluzija o praktično neomejeno velikem pomnilniku.

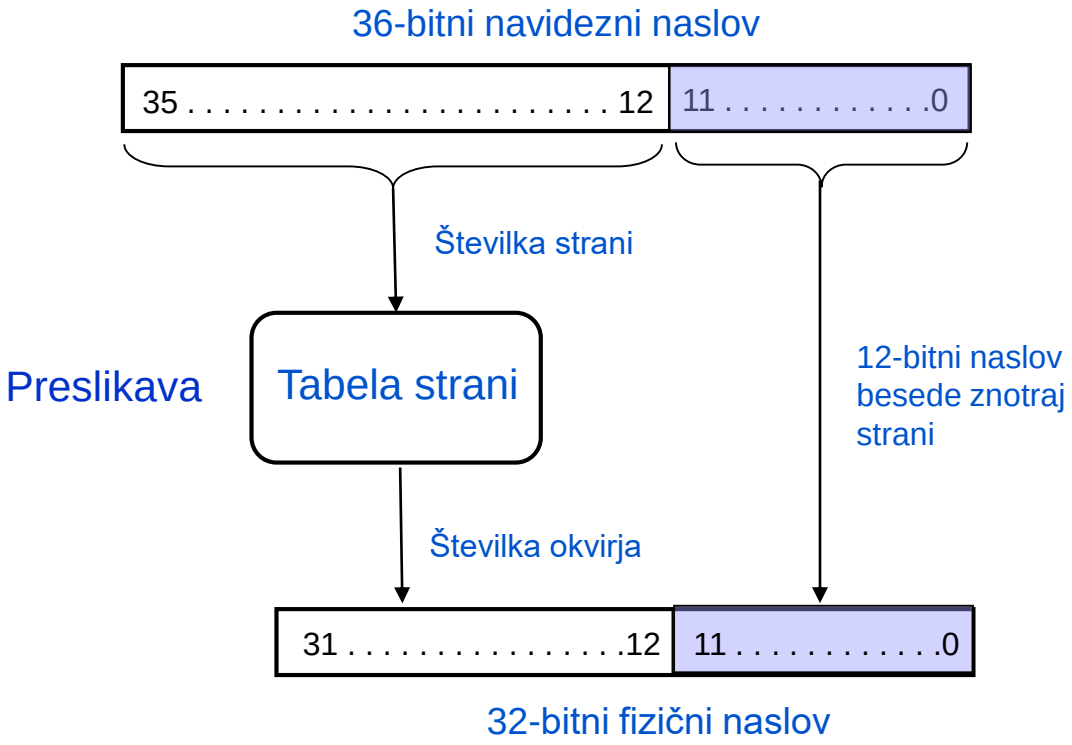
- Vsako stran iz navideznega je možno prenesti v poljuben okvir fizičnega pomnilnika.
- Za uporabnika je delitev pomnilniškega prostora na strani nevidna.



- Preslikava navideznega naslova (naslov strani) v fizični naslov (naslov okvirja) poteka preko tabele strani. ->

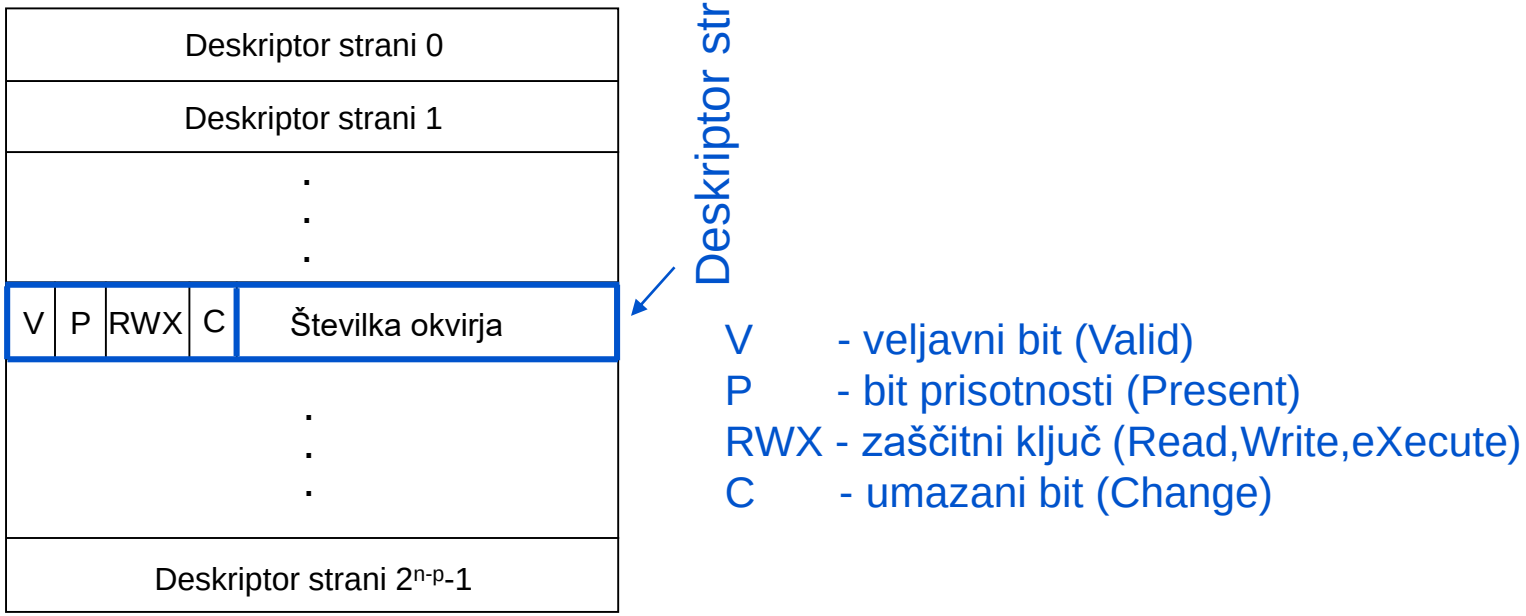
Primer preslikovanja navideznih naslovov v fizične pri ostranjevanju:

Velikost strani (in okvirja) 4 KB ($\Rightarrow 2^{12}$ B)
Navidezni naslov 36 bitov ($\Rightarrow$ Navidezni pomnilnik največ 2^{36} B = 64 GB)
Fizični naslov 32 bitov ($\Rightarrow$ Fizični pomnilnik največ 2^{32} B = 4 GB)



Zgradba tabele strani

Velikost navideznega pomnilnika 2^n Bajtov (pri n=36 $\Rightarrow$ navidezni pomnilnik = 64 GB)
Velikost strani 2^p Bajtov (pri p=12 $\Rightarrow$ velikost strani = 4 KB)
Število strani v nav. pomnilniku = 2^{n-p} ($2^{36-12} = 2^{24}$ = 16 M strani (M = 2^{20}))
Število deskriptorjev v tabeli = število strani = 16 M

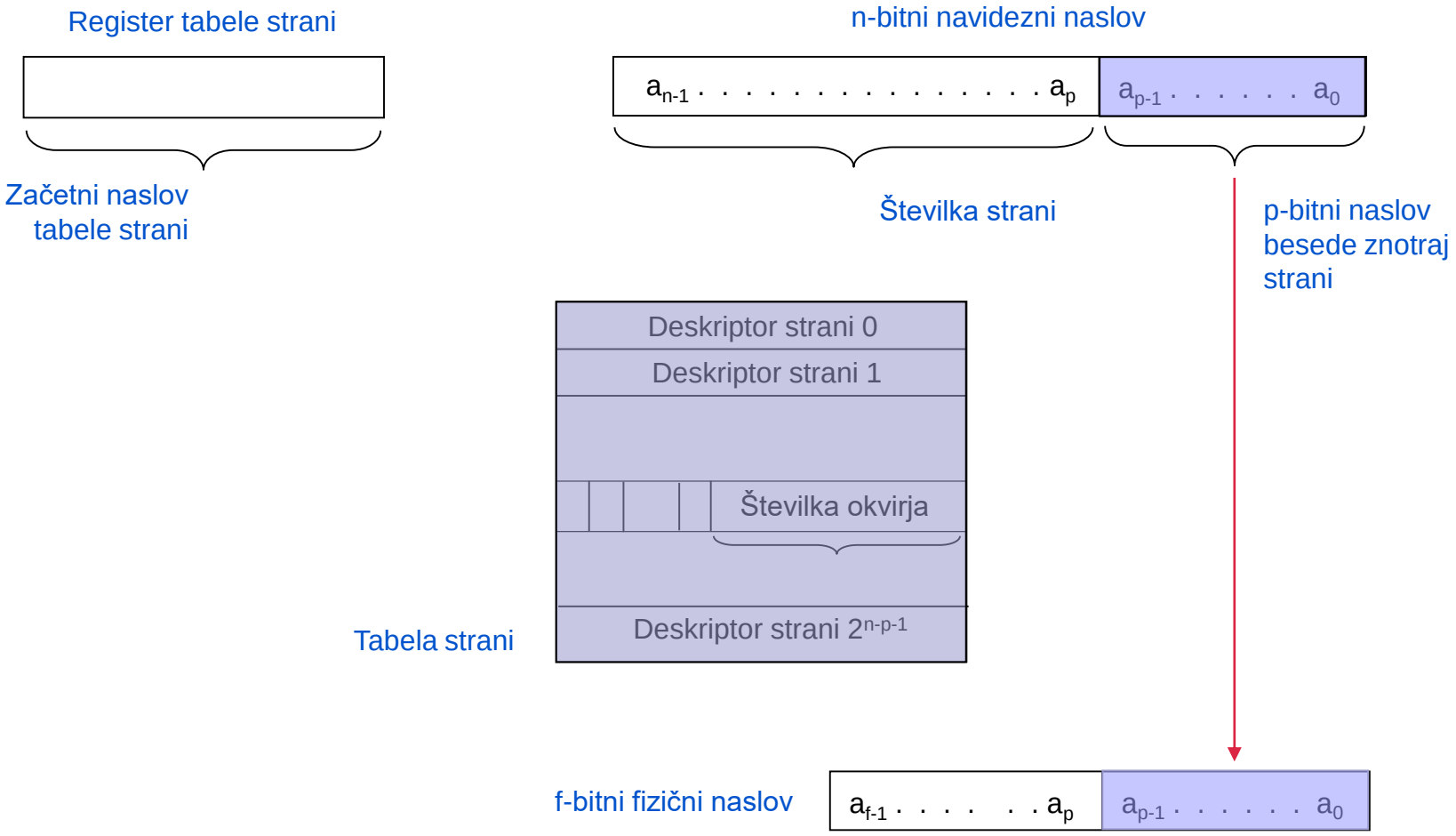


- Deskriptor strani (page descriptor) $\Rightarrow$ polje v tabeli strani, ki opisuje določeno stran.

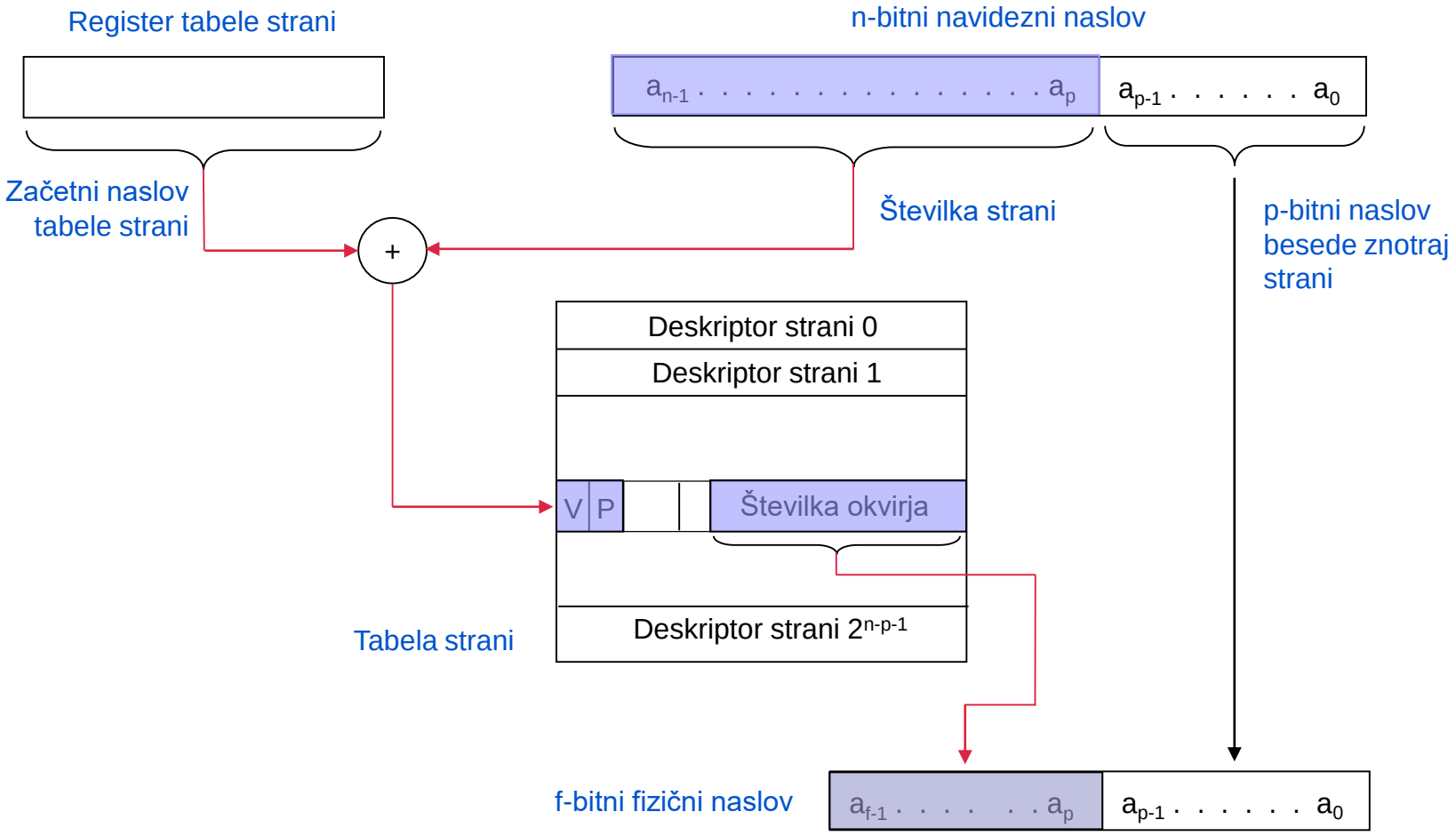
- Število deskriptorjev v tabeli je enako številu strani v navideznem pomnilniku.

- Tabela strani je običajno v glavnem pomnilniku.

Preslikovanje navideznih naslovov v fizične pri ostranjevanju

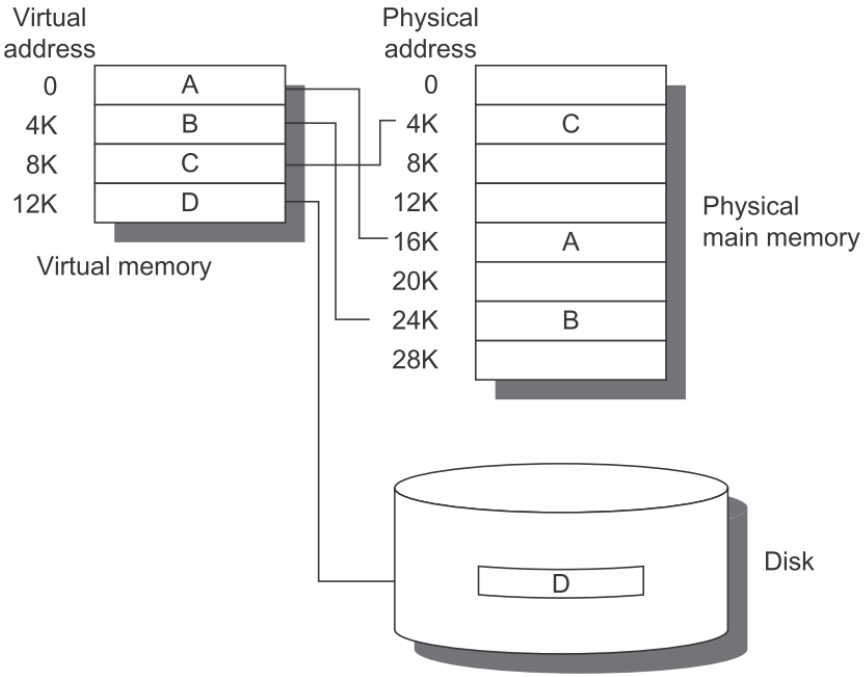


Preslikovanje navideznih naslovov v fizične pri ostranjevanju



- Linearno preslikovanje – navidezni naslovni prostor je linearen. Pri računanju z navideznimi naslovi ni nobenih omejitev, kot če ne bi imeli navideznega pomnilnika.
- Delitev pomnilniškega prostora na strani je za uporabnika nevidna – običajnim programerjem sploh ni treba vedeti za obstoj strani.
- Ena sama tabela strani $\Rightarrow$ Enonivojska preslikava

- Operacijski sistem za vsak program, ki se izvaja, vzpostavi svojo tabelo strani. Ob preklopu na drug program se zamenja vsebina registra, ki kaže na začetek tabele strani.



- Stanje programa je določeno s tabelo strani, programskim števcem in registri (= proces).
- S tabelo strani je določen naslovni prostor, ki ga določen proces lahko uporablja.

- Tabele strani očitno v pomnilniku zasedejo veliko prostora
- Tabelo strani razdelimo na več nivojev $\Rightarrow$ več-nivojska preslikava
- Prednost: zmanjša se prostor, ki ga zasedajo tabele strani v glavnem pomnilniku.
- Največkrat se uporablja dve ali trinivojska preslikava preko dveh ali treh nivojev tabel strani.

- Operacijski sistem dodeljuje glavni (fizični) pomnilnik procesom in skrbi za ažuriranje tabel strani.

- Navidezni pomnilnik omogoča uporabo glavnega pomnilnika večim procesom tako, da je:

  - pomnilniški prostor enega procesa zaščiten pred drugimi procesi.

Napake strani

- Napaka strani (page fault): če navidezne strani ni v nobenem od okvirjev v glavnem pomnilniku (P-bit deskriptorja strani = 0), se sproži past za napako strani.

- Past za napako strani $\Rightarrow$ izvajati se začne servisni program, ki:
  - poišče stran v navideznem pomnilniku (na disku),
  - določi v kateri okvir v glavnem pomnilniku se bo stran preslikala in jo prenese,
  - ažurira deskriptor te strani v tabeli strani.

- Ko operacijski sistem kreira proces, na disku običajno ustvari prostor za vse strani procesa (swap space).

- Istočasno ustvari tudi podatkovno strukturo, ki za vsako stran vsebuje informacijo, kje je shranjena na disku.

Primerjava realizacije navideznega pomnilnika

	Intel Core i7 (Nehalem)	ARM Cortex-A8 (32-bitni)	ARM Cortex-A53 (64-bitni)
Navidezni naslov	48 bitov	32 bitov	48 bitov
Fizični naslov	44 bitov	32 bitov	44 bitov
Velikost strani	4 KB, 2 MB, 4 MB	4, 16, 64 KB; 1, 16 MB	4, 16, 64 KB; 1, 2 MB; 1 GB

Strategije in algoritmi

- Delovanje navideznega pomnilnika vodi operacijski sistem, s ciljem doseči največjo izkoriščenost računalnika.

- Kot velika izkoriščenost se večinoma smatra, da se dana množica programov izvrši v najkrajšem možnem času.

- Na izkoriščenost računalnika vpliva izbira pravil, ki določajo:
  - Koliko okvirov strani naj ima v glavnem pomnilniku nek program.
  - Kdaj, katere in koliko strani naj se prenese iz pomožnega v glavni pomnilnik.
  - Katere strani naj se prenesejo iz glavnega pomnilnika nazaj v pomožni pomnilnik.

- Ta pravila se imenujejo **dodeljevalna, polnilna in zamenjevalna strategija**.

- Pri navideznem pomnilniku so strategije realizirane programsko, pri predpomnilniku pa strojno.

- Vse tri strategije izvajajo algoritmi s skupnim imenom **upravljanje s pomnilnikom** (memory management).

Pohitritev preslikovanja

- Pri preslikavi navideznega naslova v fizični naslov
  - je potreben dostop do tabele strani
  - tabele so shranjene v glavnem pomnilniku ali celo v navideznem pomnilniku
- Pri vsakem dostopu do pomnilnika sta zato potrebna dva dostopa do glavnega pomnilnika (če je preslikava enonivojska):
  - 1. dostop do deskriptorja v tabeli strani v glavnem pomnilniku
  - 2. dostop do želene besede na fizičnem naslovu v glavnem pomnilniku

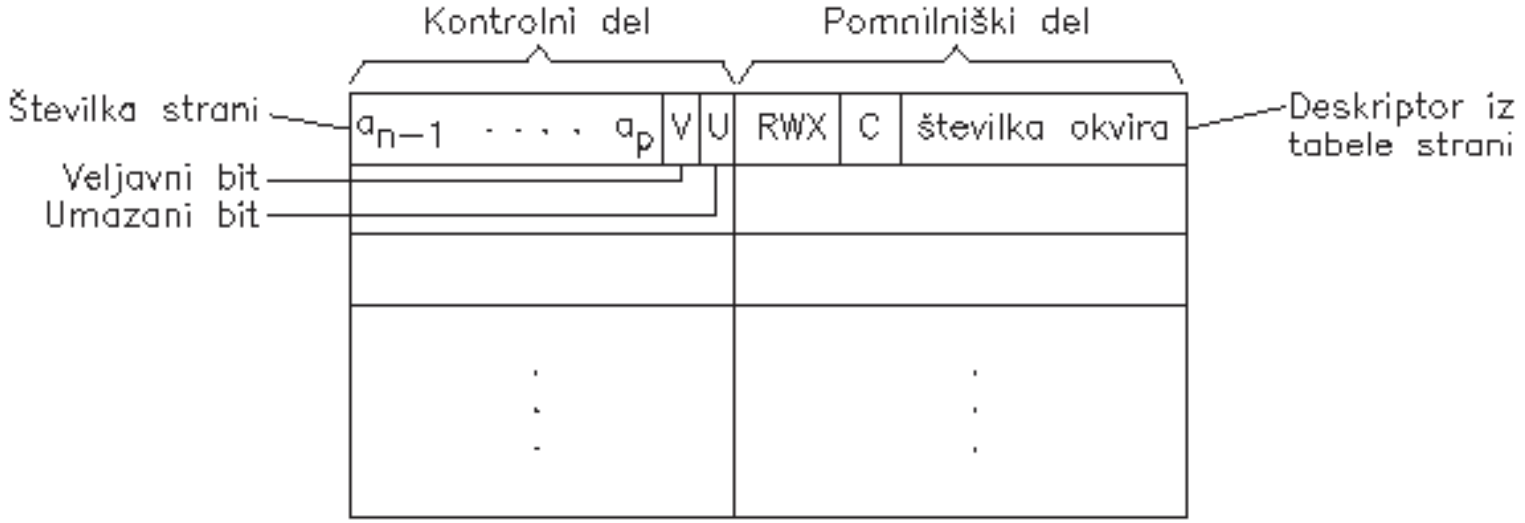
- Pri več-nivojski preslikavi se poveča na 3 do 4 dostope do glavnega pomnilnika.
- Prepočasno!
- Rešitev: Predpomnilnik v CPE, ki vsebuje nekaj nazadnje uporabljenih deskriptorjev (nikoli operandov ali ukazov).

Preslikovalni predpomnilnik (translation cache)

- **TLB** (Translation Lookaside Buffer)

- Dolžina bloka v preslikovalnem predpomnilniku je enaka dolžini deskriptorja, v kontrolnem delu pa je številka strani, ki ji deskriptor pripada.

- Visoko verjetnost zadetka (99% do 99,9%) se lahko doseže že z nekaj deskriptorji, zato je preslikovalni predpomnilnik lahko majhen in je večkrat čisti asociativni.



- Pri zadetku v preslikovalnem predpomnilniku dostop do tabele v glavnem pomnilniku ni potreben.

- Pri Harvardski arhitekturi (ukazni in operandni predpomnilnik), sta potrebna tudi dva preslikovalna predpomnilnika (ukazni in operandni – ITLB in DTLB).

9.5 Delovanje pomnilniške hierarhije

- Pomnilniška hierarhija je iz CPE videti kot en sam pomnilnik:
  - S hitrostjo, ki je blizu hitrosti predpomnilnika (pomnilnika, ki je najbližji CPE).
  - Z velikostjo navideznega pomnilnika na pomožnem pomnilniku (zadnjega v hierarhiji).

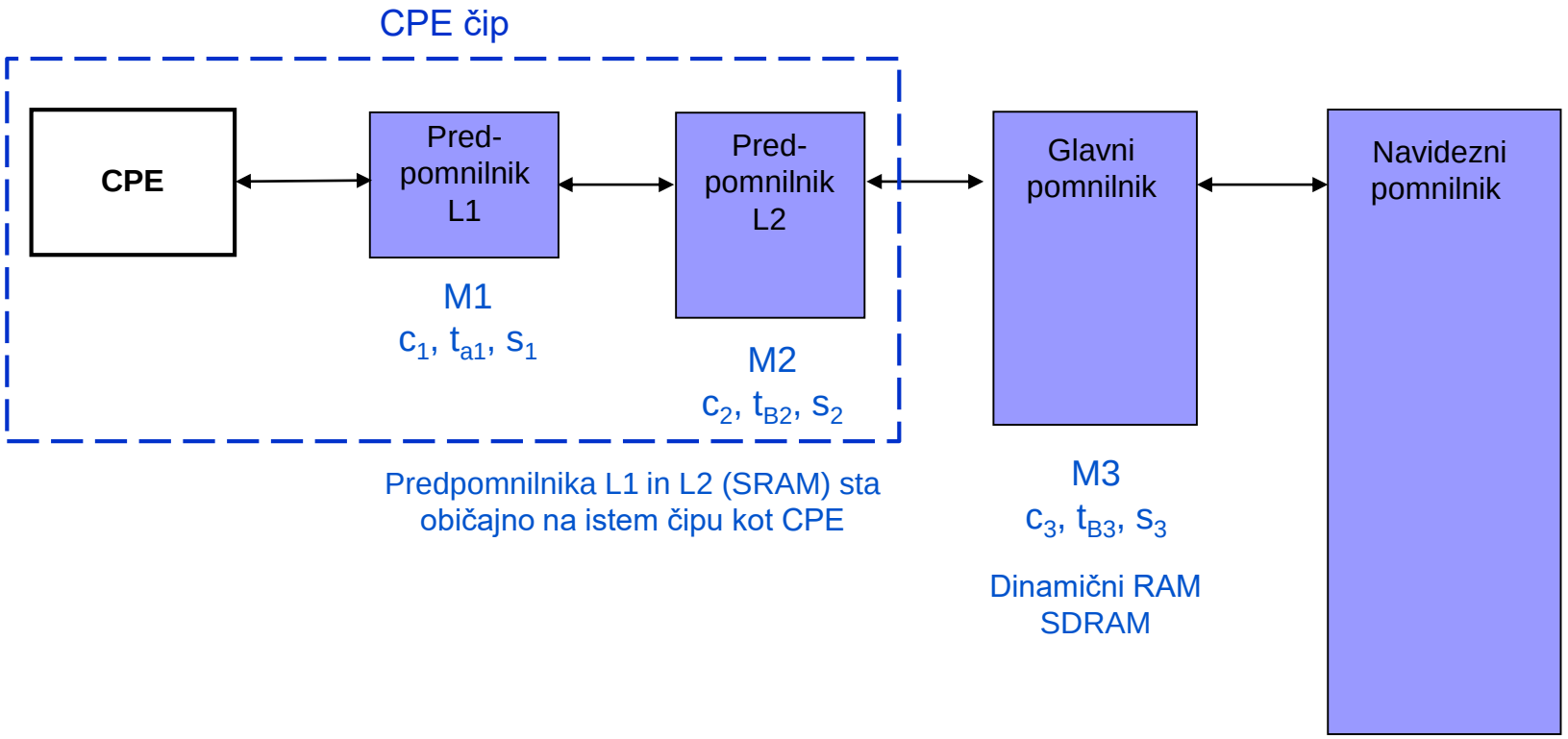
- Pomnilniška hierarhija se od enonivojskega pomnilnika razlikuje v naslednjih lastnostih:
  - Čas dostopa ni enak za vse pomnilniške naslove, odvisen je od pomnilniškega nivoja na katerem je trenutno iskana pomnilniška beseda.
  - Za določen pomnilniški dostop ne moremo predvideti njegovega trajanja, znana je samo statistično določena povprečna vrednost časa dostopa.

- CPE pošlje v pomnilniško hierarhijo vedno naslov, ki se nanaša na pomnilnik M_n (zadnji v hierarhiji), vendar to ne pomeni, da bo dostop v resnici opravljen v M_n.
  - Če je informacija, do katere želi CPE dostop, v M_1 ($\Rightarrow$ zadetek), se opravi dostop do M_1
  - Če informacije ni v M_1 ($\Rightarrow$ zgrešitev), se v M_1 prenese iz M_2
  - Če želene informacije ni tudi v M_2, se v M_2 prenese iz M_3
  - . . .
  - Pri vsakem dostopu je zahtevana informacija vedno v pomnilniku M_n na zadnjem nivoju

Primerjava lastnosti predpomnilnika in navideznega pomnilnika z ostranjevanjem

	Predpomnilnik	Navidezni pomnilnik
Dostop	Predpomnilniška vrstica (blok)	Stran (okvir strani)
Blok	16B do 128B	4KB do 16KB (lahko tudi nekaj MB)
Verjetnost zgrešitve (1-H)	0,1% do 10% za L1	< 0,0001% (za glavni pomn.)
Zadetek	nekaj urinih period	~ 10 do 100 urinih period
Zgrešitvena kazen	~ 10 do 100 urinih ciklov	~ 10M urinih ciklov
Zamenjava bloka	Strojno (hardware)	Programsko (software)

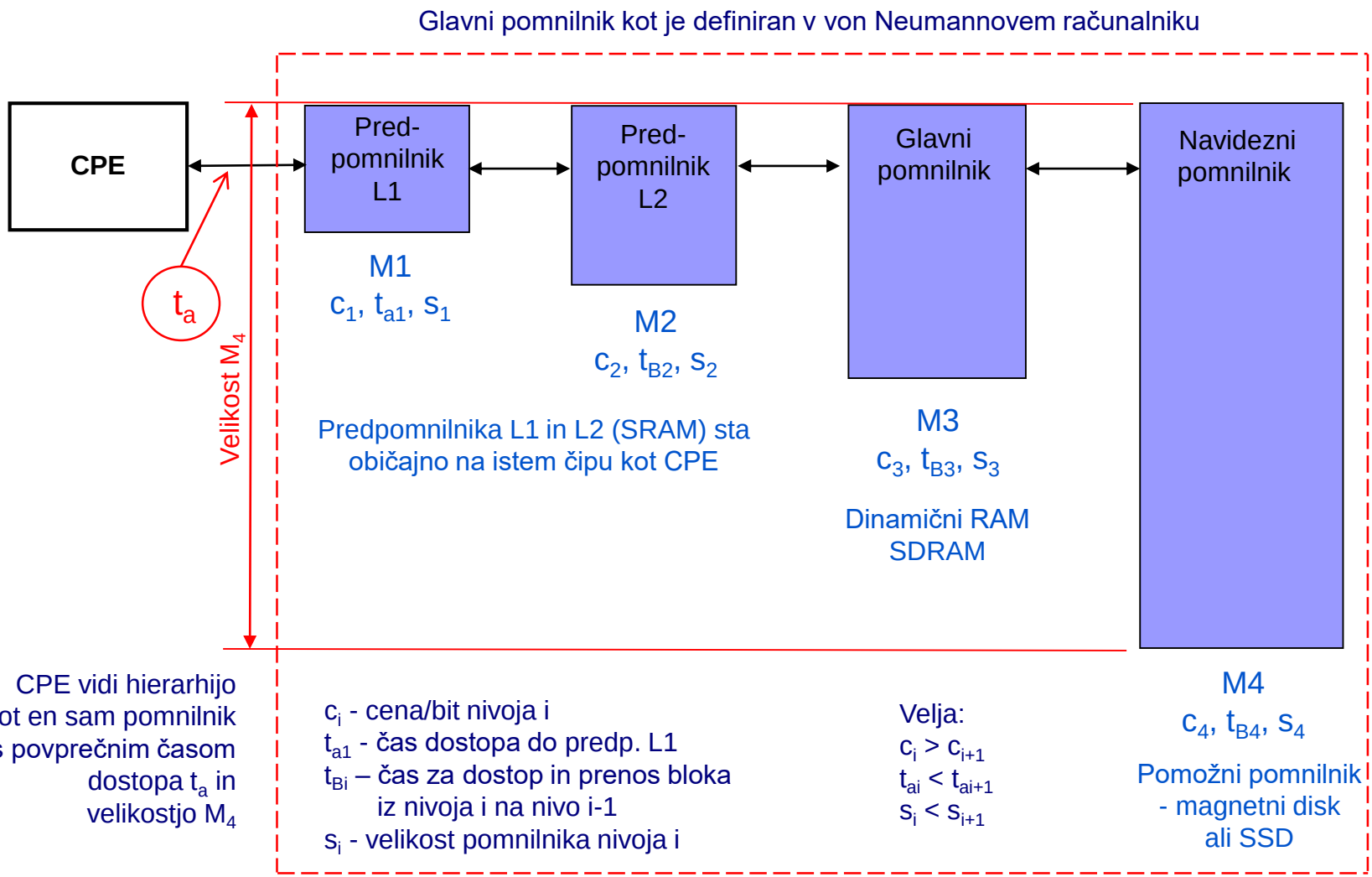
Štirinivojska pomnilniška hierarhija



c_i – cena/bit nivoja i
t_{a1} – čas dostopa do predpomnilnika L1
t_{Bi} – čas za dostop in prenos bloka iz nivoja i na nivo i-1
s_i – velikost pomnilnika nivoja i

Velja:
$c_i > c_{i+1}$
$t_{ai} < t_{ai+1}$
$s_i < s_{i+1}$

Štirinivojska pomnilniška hierarhija

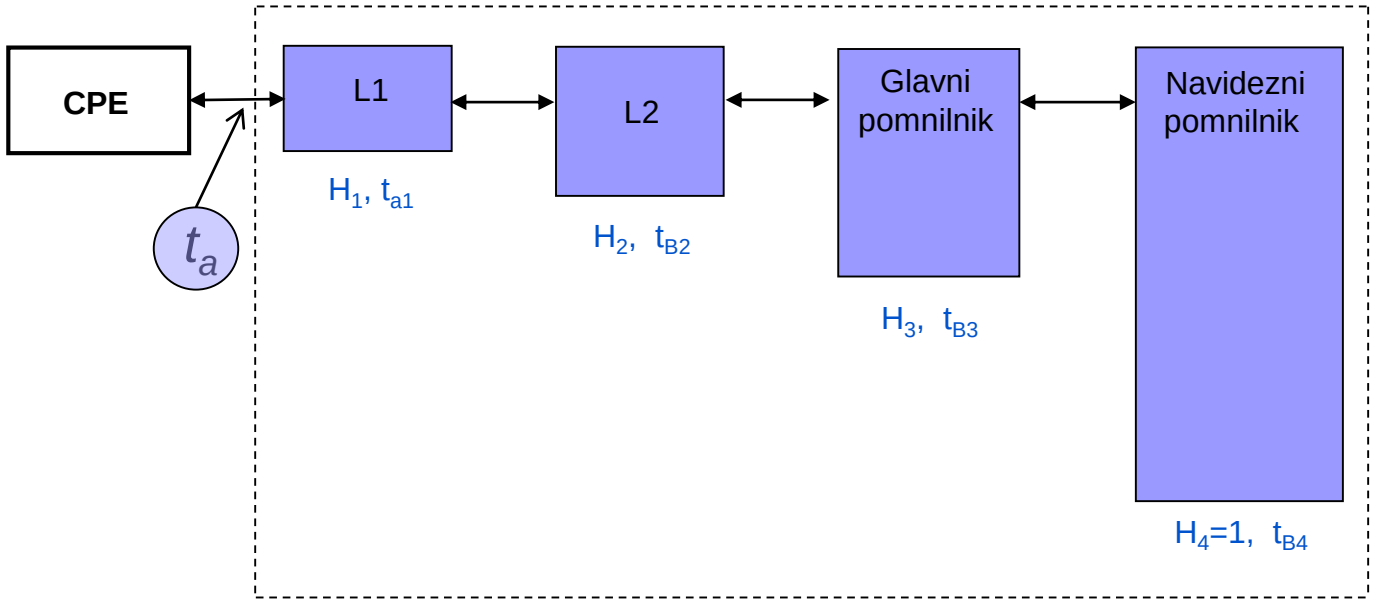


Velja: Če je informacija na nivoju *i,* je zagotovo tudi na *(i+1).*

- $H_i \Rightarrow$ (globalna) verjetnost, da je pri poljubnem dostopu do pomnilniške hierarhije informacija na nivoju *i* .
- $(1 - H_i) \Rightarrow$ (globalna) verjetnost, da pri poljubnem dostopu želene informacije ni na nivoju *i* .
- Povprečni čas dostopa t_a do *n*-nivojske pomnilniške hierarhije, kot ga vidi CPE, je:

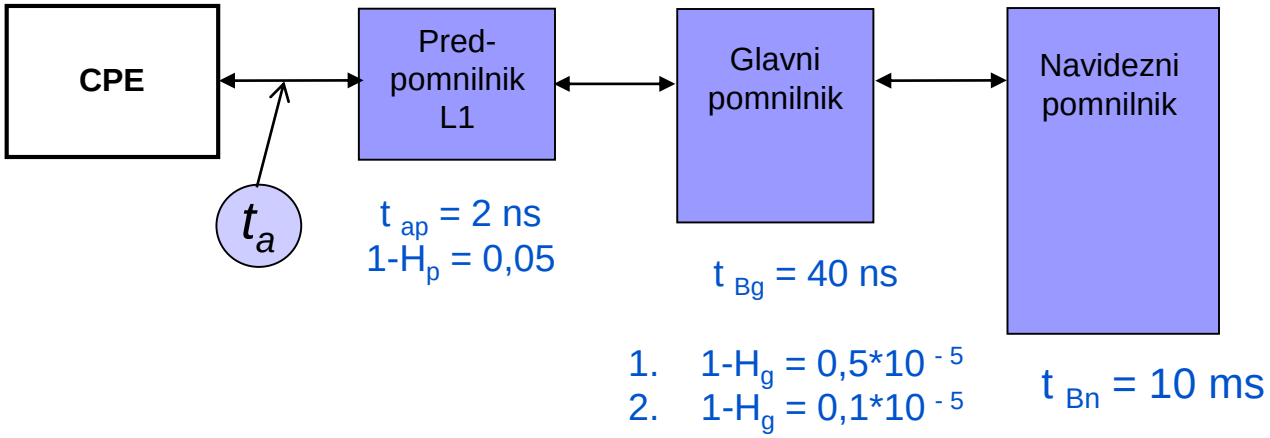
$$t_a = t_{a1} + (1 - H_1)t_{B2} + \ldots + (1 - H_{i-1})t_{Bi} + \ldots + (1 - H_{n-1})t_{Bn}$$

- Za štirinivojsko pomnilniško hierarhijo velja:



$$t_a = t_{a1} + (1 - H_1)t_{B2} + (1 - H_2)t_{B3} + (1 - H_3)t_{B4}$$

Primer: Vpliv verjetnosti zgrešitve v glavnem pomnilniku na povprečni dostopni čas pri trinivojski pomnilniški hierarhiji



t_{ap} – čas dostopa do predpomnilnika L1
H_p – verjetnost zadetka v predpomnilniku L1 ($1-H_p$ – verjetnost zgrešitve v L1)
t_{Bg} – čas za dostop do glavnega pomnilnika in prenos bloka iz glavnega pomnilnika v L1
H_g – verjetnost zadetka v glavnem pomnilniku ($1-H_g$ – verjetnost zgrešitve v glavnem pomnilniku)
t_{Bn} – čas za dostop do navideznega pomnilnika in prenos bloka iz navideznega pomnilnika v glavni pomnilnik
t_a – povprečni dostopni čas do celotne hierarhije, kot ga vidi CPE

1. Naj bo verjetnost zadetka v glavnem pomnilniku $H_g = 0{,}999995 = 99{,}9995\%$ torej je verjetnost zgrešitve v glavnem pomnilniku $1\text{-}H_g = 1 - 0{,}999995 = 0{,}000005 = 0{,}0005\%$ ali $1\text{-}H_g = 0{,}5*10^{-5}$

   $t_{ap} = 2$ ns; $1\text{-}H_p = 0{,}05$; $t_{Bg} = 40$ ns; $t_{Bn} = 10$ ms

$$t_a = t_{ap} + (1 - H_p) \cdot t_{Bg} + (1 - H_g) \cdot t_{Bn} =$$
$$= 2 \cdot 10^{-9}[s] + 0{,}05 \cdot 40 \cdot 10^{-9}[s] + 0{,}5 \cdot 10^{-5} \cdot 10 \cdot 10^{-3}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 5 \cdot 10^{-8}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 50 \cdot 10^{-9}[s] = 54 \cdot 10^{-9}[ns] = 54[ns]$$

1. Naj bo verjetnost zadetka v glavnem pomnilniku $H_g = 0{,}999995 = 99{,}9995\%$ torej je verjetnost zgrešitve v glavnem pomnilniku $1\text{-}H_g = 1 - 0{,}999995 = 0{,}000005 = 0{,}0005\%$ ali $1\text{-}H_g = 0{,}5*10^{-5}$

   $t_{ap} = 2$ ns; $1\text{-}H_p = 0{,}05$; $t_{Bg} = 40$ ns; $t_{Bn} = 10$ ms

$$t_a = t_{ap} + (1 - H_p) \cdot t_{Bg} + (1 - H_g) \cdot t_{Bn} =$$
$$= 2 \cdot 10^{-9}[s] + 0{,}05 \cdot 40 \cdot 10^{-9}[s] + 0{,}5 \cdot 10^{-5} \cdot 10 \cdot 10^{-3}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 5 \cdot 10^{-8}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 50 \cdot 10^{-9}[s] = 54 \cdot 10^{-9}[ns] = 54[ns]$$

Povprečni dostopni čas do celotne hierarhije je v tem primeru 54 ns, kar je slabše kot dostopni čas do glavnega pomnilnika (40 ns). Taka pomnilniška hierarhija rešuje problem velikosti pomnilnika, poslabša pa dostopni čas in je zato popolnoma neuporabna. Rešitev je povečanje verjetnosti zadetka v glavnem pomnilniku.

2. Če se verjetnost zadetka v glavnem pomnilniku poveča iz 99,9995% na 99,9999% $\Rightarrow H_g = 0{,}999999 = 99{,}9999\%$

   torej bo verjetnost zgrešitve v glavnem pomnilniku $1\text{-}H_g = 0{,}000001 = 0{,}0001\%$ ali $1\text{-}H_g = 0{,}1*10^{-5}$ (v prejšnjem primeru $1\text{-}H_g = 0{,}5*10^{-5}$)

   ostali podatki pa ostanejo nespremenjeni:
   $t_{ap} = 2$ ns; $1\text{-}H_p = 0{,}05$; $t_{Bg} = 40$ ns; $t_{Bn} = 10$ ms

$$t_a = t_{ap} + (1 - H_p) \cdot t_{Bg} + (1 - H_g) \cdot t_{Bn} =$$
$$= 2 \cdot 10^{-9}[s] + 0{,}05 \cdot 40 \cdot 10^{-9}[s] + 0{,}1 \cdot 10^{-5} \cdot 10 \cdot 10^{-3}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 1 \cdot 10^{-8}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 10 \cdot 10^{-9}[s] = 14 \cdot 10^{-9}[ns] = 14[ns]$$

2. Če se verjetnost zadetka v glavnem pomnilniku poveča iz 99,9995% na 99,9999% $\Rightarrow H_g = 0{,}999999 = 99{,}9999\%$

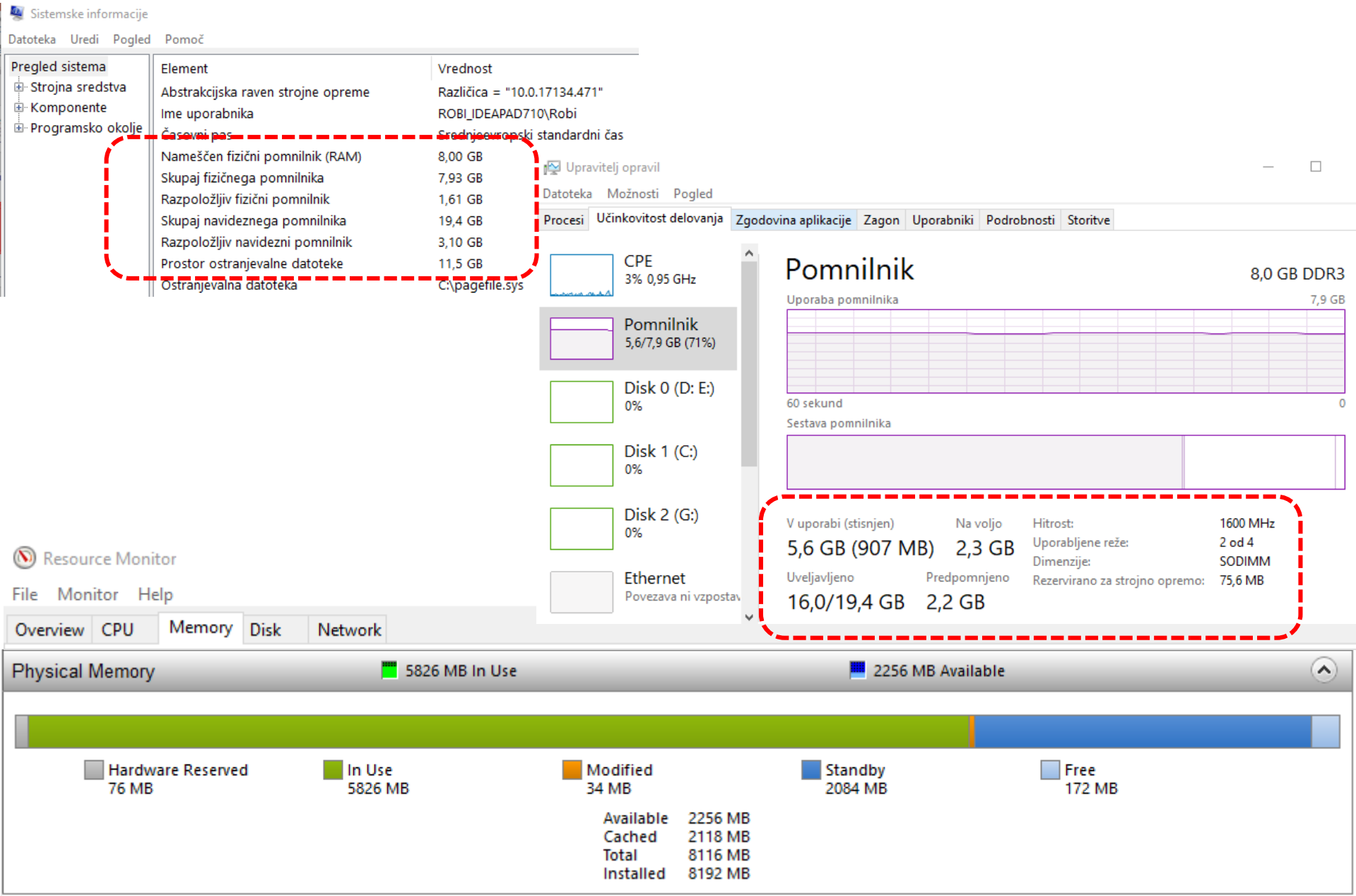
torej bo verjetnost zgrešitve v glavnem pomnilniku $1\text{-}H_g = 0{,}000001 = 0{,}0001\%$ ali $1\text{-}H_g = 0{,}1*10^{-5}$ (v prejšnjem primeru $1\text{-}H_g = 0{,}5*10^{-5}$)

ostali podatki pa ostanejo nespremenjeni:
$t_{ap} = 2$ ns; $1\text{-}H_p = 0{,}05$; $t_{Bg} = 40$ ns; $t_{Bn} = 10$ ms

$$t_a = t_{ap} + (1 - H_p) \cdot t_{Bg} + (1 - H_g) \cdot t_{Bn} =$$
$$= 2 \cdot 10^{-9}[s] + 0{,}05 \cdot 40 \cdot 10^{-9}[s] + 0{,}1 \cdot 10^{-5} \cdot 10 \cdot 10^{-3}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 1 \cdot 10^{-8}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 10 \cdot 10^{-9}[s] = 14 \cdot 10^{-9}[ns] = 14[ns]$$

Če se verjetnost zgrešitve v glavnem pomnilniku zmanjša iz $0{,}5*10^{-5}$ na $0{,}1*10^{-5}$ (verjetnost zadetka se poveča), se povprečni dostopni čas zmanjša iz 54 ns na 14 ns.

Primer: Navidezni pomnilnik Win10



- Hvala za pozornost in veliko uspeha na izpitu !
- Spletne strani: http://ucilnica.fri.uni-lj.si
  http://www.fri.uni-lj.si/
- Moj e-naslov: rozman@fri.uni-lj.si
- Literatura:
  - Dušan Kodek: ARHITEKTURA IN ORGANIZACIJA RAČUNALNIŠKIH SISTEMOV, Bi-TIM, 2008
  - David A. Patterson, John L. Hennesy: COMPUTER ORGANIZATION AND DESIGN, ARM Edition, Morgan Kaufmann, Elsevier, 2017
  - Andrew S. Tanenbaum: STRUCTURED COMPUTER ORGANIZATION, Sixth Edition, Pearson Prentice Hall, 2013
  - Prosojnice na http://ucilnica.fri.uni-lj.si