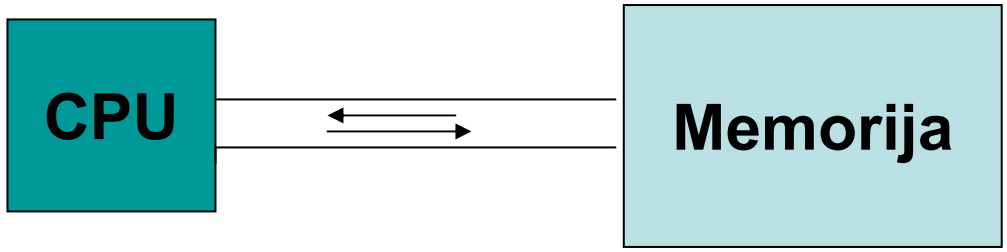
9. Priručne memorije

1. Svojstva i organizacija dinamičkog RAM-a
2. Memorijska hijerarhija
3. Organizacija priručne memorije
4. Odabir parametara, performansa
5. Izvedbeni detalji

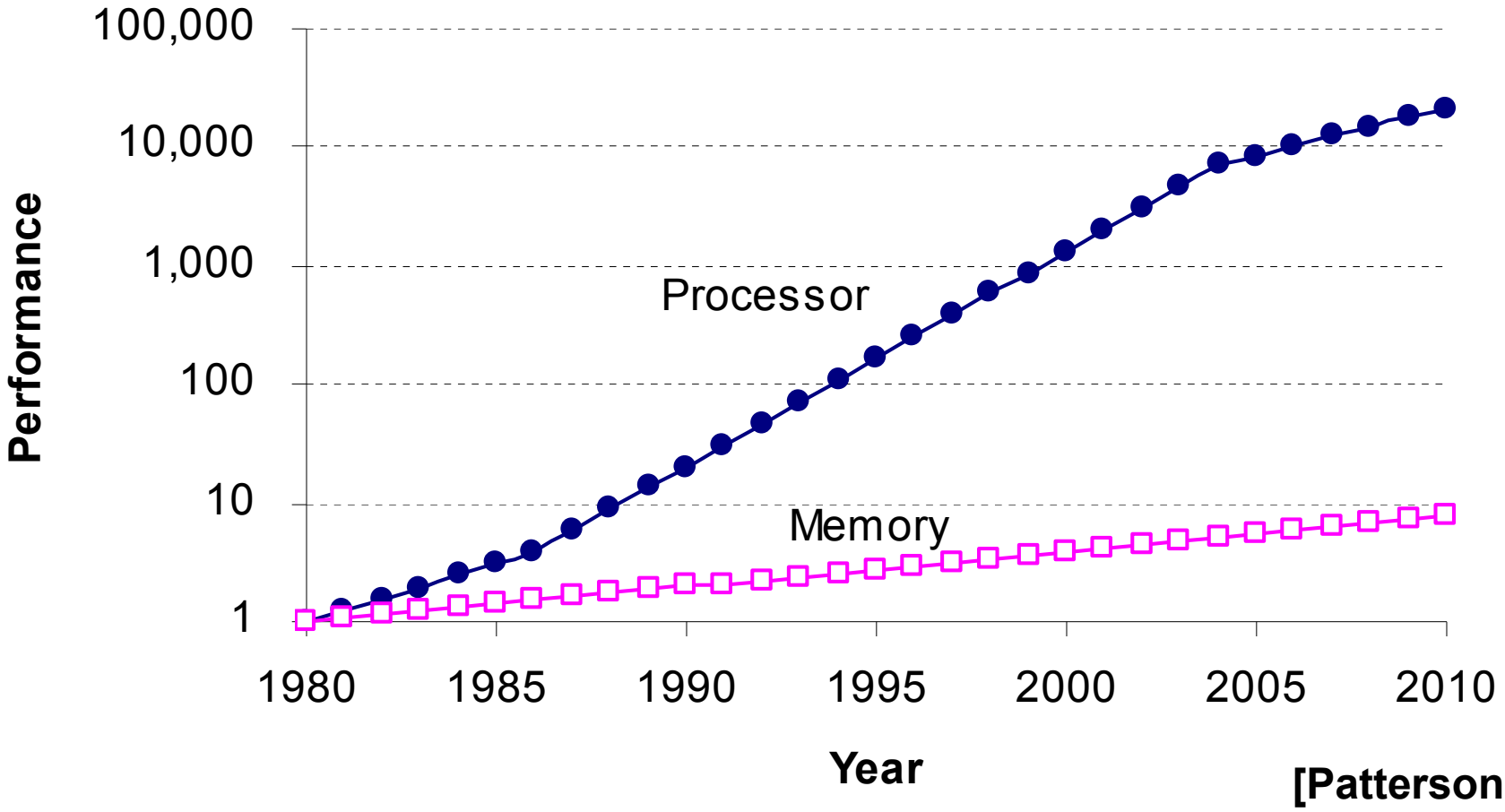
Komunikacija s memorijom usko grlo performanse



Obrada mnogo brža od **latencije** memorije potrebnog kapaciteta:
• 0.5 ns (zbrajanje 32b, P4@2GHz) vs. 50 ns (DDR2-800)
• za sada razmatramo **radnu** (glavnu) **memoriju**, RAM

Propusnost u prosjeku dovoljna (korištenjem sofisticiranih tehnika):
•potrebna propusnost: (takt / CPI) × (1 + m) × riječ ≈ 4GB/s
– takt = 2 GHz, *prosječna* riječ = 3
– m = 30% (učestalost memorijskih instrukcija)
• 20% grananje, 50% aritmetika
– CPI ∈ (1, 6), uzmimo srednju vrijednost CPI=2 (P4, SPECint2000)
(CPIthmax (P4)=0.33)
•~4 GB/s (P4, vidi gore) vs. 6.4 GB/s (DDR2-800: 64b×800 transfera/s)
•obratiti pažnju na to da ovdje nismo razmatrali grafiku!

Nesrazmjer latencije memorije i slijedne performanse procesora i dalje raste...



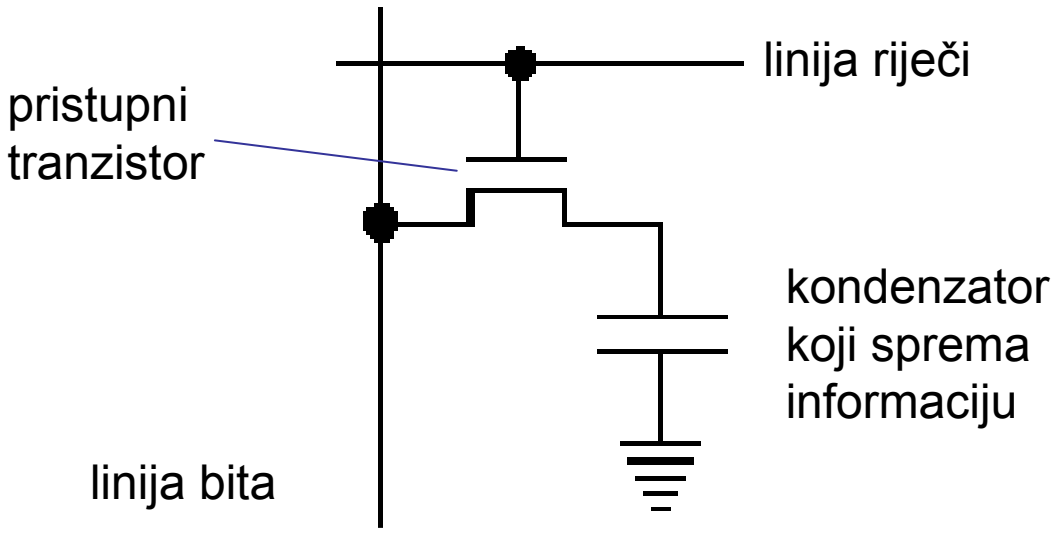
[Patterson08]

10. Priručne memorije

1. **Svojstva i organizacija dinamičkog RAM-a**
2. Memorijska hijerarhija
3. Organizacija priručne memorije
4. Odabir parametara, performansa
5. Izvedbeni detalji

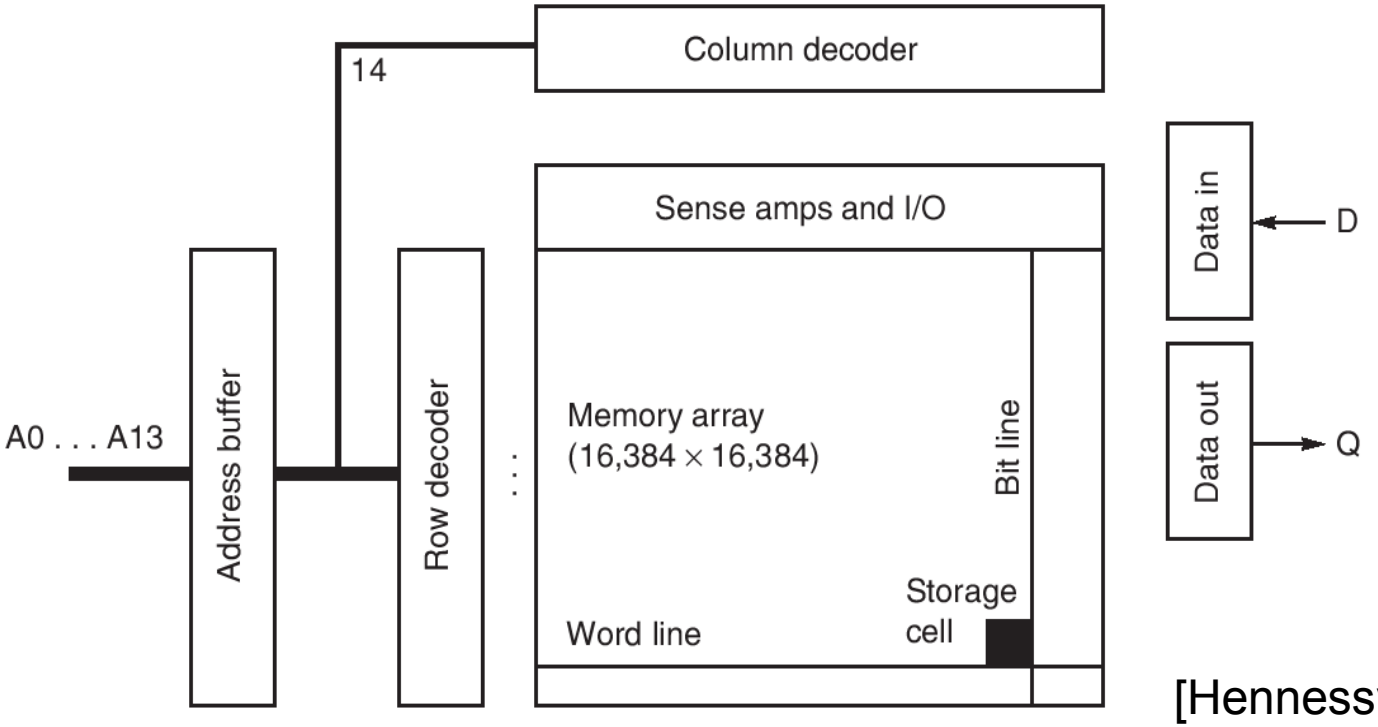
Memorijske tehnologije (RAM)

- magnetni bubnjevi (1950)
- feritne jezgre (1960, pristup 1μs)
- poluvodičke memorije (DRAM, 1970, Intel)
  - DRAM: tehnologija izbora za **glavnu** memoriju
  - veličina 1T ćelije DRAM-a odgovara veličini tranzistora
  - DRAM vs SRAM: 10× gušći, 100× jeftiniji, 40× sporiji



Struktura DRAM memorije

- informacija smještena u kvadratnom polju 1T ćelija
  - optimalne veličine dekodera i duljine prospoja

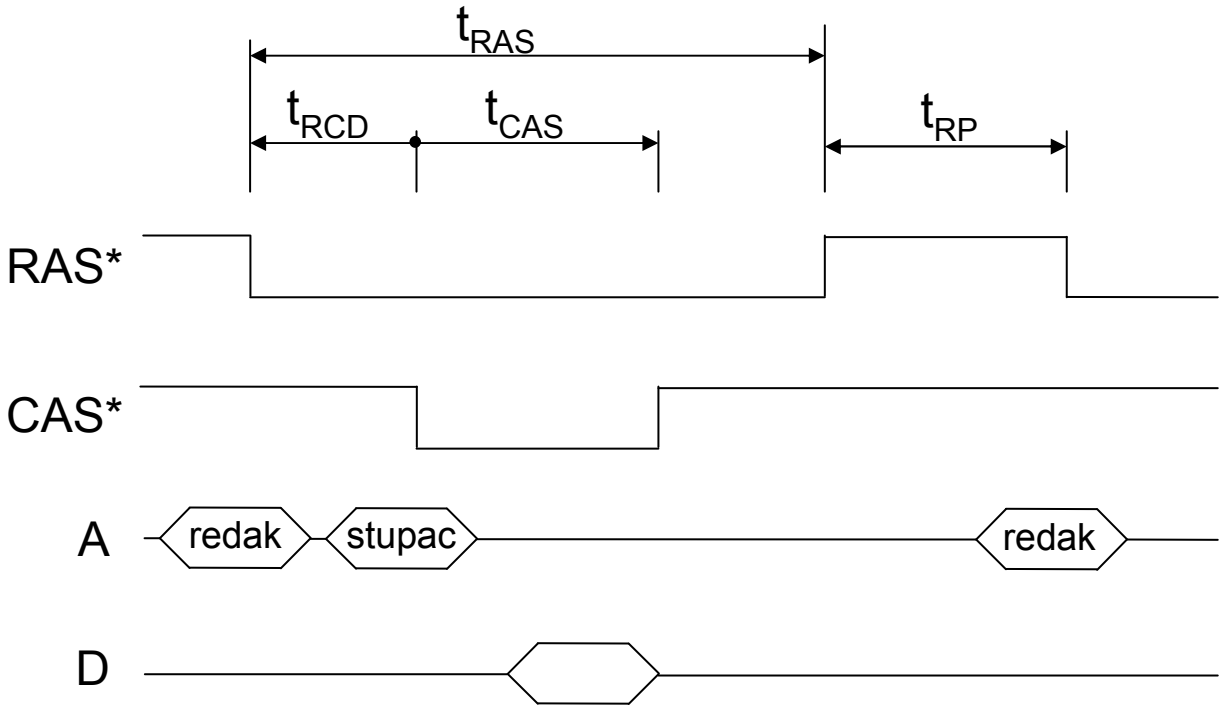


[Hennessy07]

Vremenski dijagram pristupa DRAM-u

DRAM ciklus:

- aktiviranje retka (dekodiranje, pojačavanje i spremanje, t_RCD)
- pristup stupcu (odabir bitova retka, čitanje ili pisanje, t_CAS)
- prednabijanje linija bitova (potrebno prije novog aktiviranja, t_RP)



Understanding DRAM operation,
IBM Applications Note, 1996

Pristupi za povećanje propusnosti DRAM-a (1)

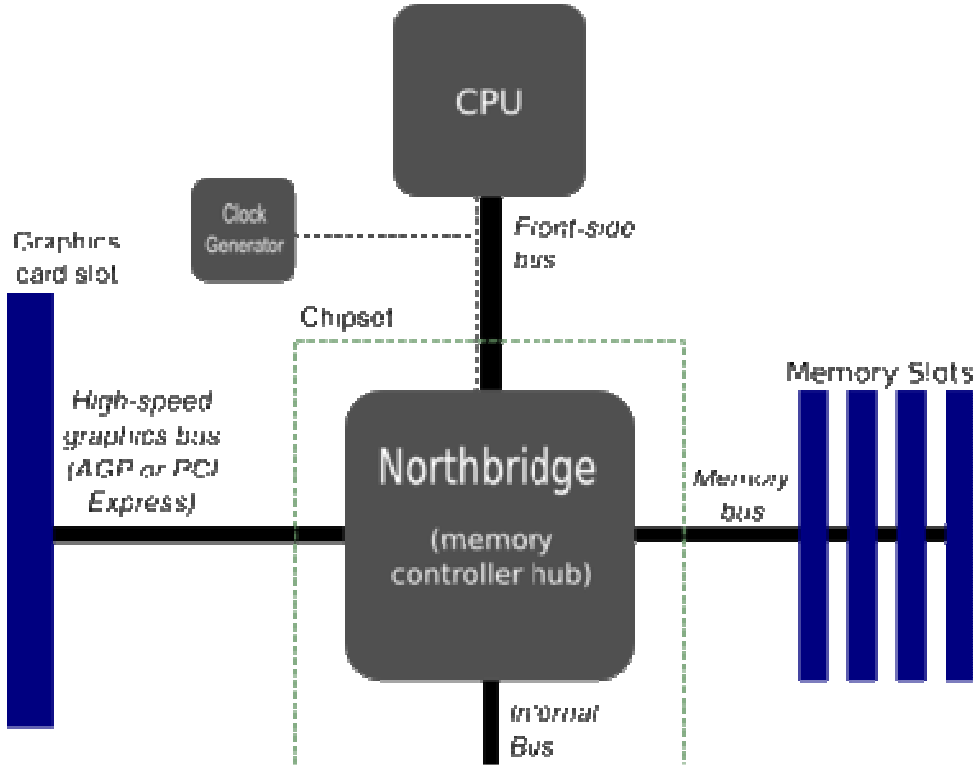
1. brzi pristup retku (fast page mode)
   - redak ostaje selektiran, bitovi stupca izlaze u ritmu t_CAS
   - jeftin način brzog pristupa susjednim podatcima

2. paralelna organizacija memorijskog modula
   - oblikovanje memorijskog modula od više pojedinačnih sklopova
   - svaki sklop odgovoran za smještanje dijela riječi
   - usporedan pristup prednost, uz veću cijenu i potrošak energije
   - tipični kompromis: 4 ili 8 bitova na svakom sklopu

Pristupi za povećanje propusnosti DRAM-a (2)

3. sinkroni sabirnički protokol (SDRAM)
   - preklapanje iščitavanja podataka i pristupa retku
     - prilikom svakog pristupa zapamtiti cijeli redak u dediciranom spremniku
     - tijekom prijenosa uzastopnih podataka iz dediciranog spremnika započeti pristup novom retku
     - sabirnica mora biti sinkrona (imati signal vremenskog vođenja)
   - **protočni** protokol pristupa memoriji:
     - naredbe se izdaju prije dovršavanja prethodne operacije!
     - dok se pribavljenih N bitova u grupama upućuju na vanjsku sabirnicu, traje pristup novom retku/stupcu
   - ostvaruje se brži **grupni** prijenos
     - podatkovna sabirnica širine 64 bita
     - grupni (burst) prijenos: više uzastopnih 64-bitnih podatka
     - pojedinačni prijenos jednako brz ili sporiji
     - prikladno za servisiranje priručnih memorija i DMA
   - nedostatak je složeno upravljanje
     - posao **memorijskog pristupnog sklopa** (MCH, northbridge)
     - ako propusnost nije kritična, bolje koristiti asinkroni protokol

Uloga memorijskog pristupnog sklopa (memory controller, northbridge)



[Wikipedia]

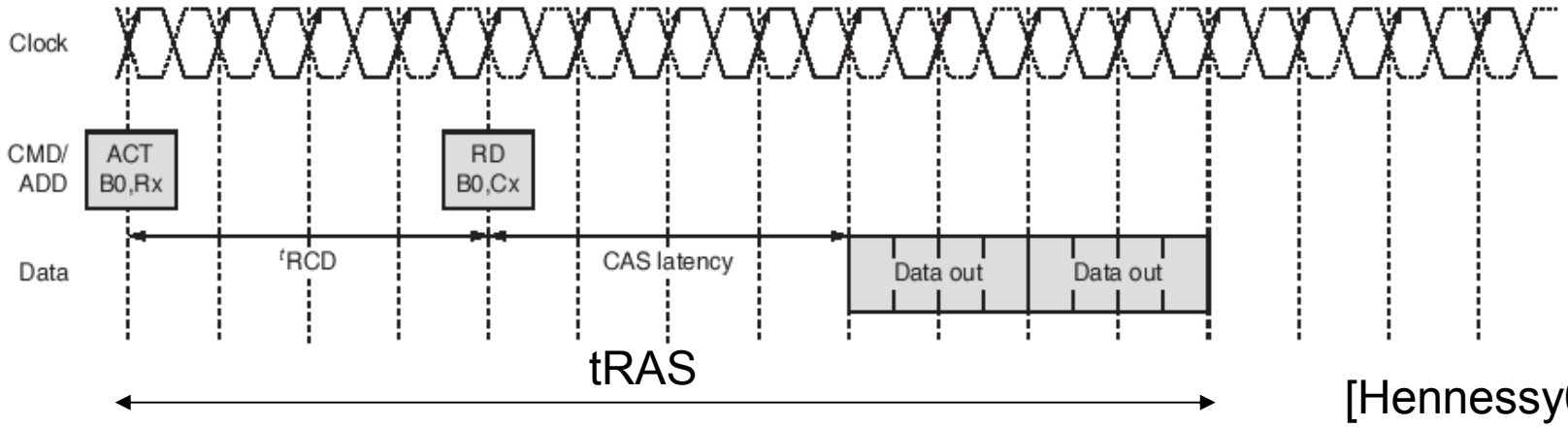
Pristupi za povećanje propusnosti DRAM-a (3)

4. preplitanje (interleaving)

- ideja: smještati susjedne podatke u različite sklopove
  - i-ti logički bit smjestiti u polje (i mod N)
  - usporedno adresirati svih N polja
  - pristigle podatke slati na vanjsku sabirnicu u ritmu t_CAS/N
- preplitanje na razini pojedinačnog sklopa:
  - danas metoda izbora (DDR: N=2, DDR2: N=4, DDR3: N=8)
  - sklopovi sadrže više polja (bank) s prepletenim podatcima
  - npr (64 Mb): umjesto 1 polja 8192×8192 imamo 4 polja 4096×4096
  - propusnost na vanjskoj sabirnici modula
    N× veća od memorijske frekvencije!
- preplitanje na razini matične ploče (dual channel):
  - najbolji rezultati uz odgovarajuće memorijske module
  - manje praktično, rjeđe korišteno

Prepleteni sinkroni dinamički RAM (SDRAM DDR)

- SDRAM DDR2: industrijski standard (www.jedec.org)
- preplitanje ×4: min. 4 uzastopna čitanja/pisanja po 64 bita
  DDR2-800: DRAM 200MHz, BUS 400Mhz (DDR), FSB 800 MHz
- latencija po fazama izražena u ciklusima **vanjske** memorijske sabirnice (CAS-RCD-RP-RAS) na slici: t_CAS=4, t_RCD=4
- DDR2-800: tipično 5-5-5-15, odnosno 12.5ns-12.5ns-12.5ns
  - memorijska frekvencija iznosi 400 MHz ⇒ T = 2.5 ns
  - ukupno vrijeme **slučajnog** pristupa: t= tRAS + tRP = (5+15)T = 50 ns



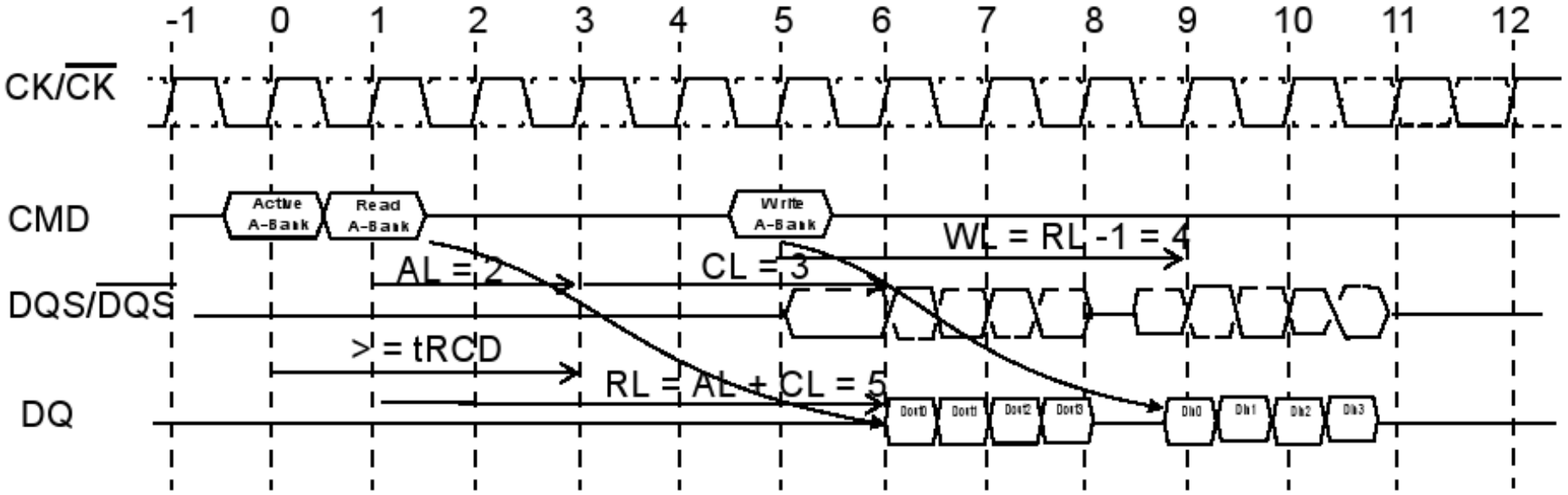
[Hennessy07]

Primjeri:

- npr DDR2-800:
  - vršna propusnost: 800 Mprijenosa/s (6400MB/s $\Rightarrow$ PC-6400)
  - istovremeno se prozivaju 4 polja na 200MHz
  - latencija sukladna memoriji DDR-400!
  - DIMM modul: 240 izvoda, tipično 8 sklopova po 8(+ECC) bita
- npr DDR3-X:
  - osmerostruko preplitanje
  - ukupna propusnost 8$\times$ veća od propusnosti pojedinačnog polja
  - latencija odgovara SDRAM memoriji na taktu X/8
- npr, DDR2-800 vs DDR3-800
  - koji modul ima bolju propusnost odnosno latenciju?

SDRAM DDR2 standard (JEDEC):

- AL – additive latency
- CL – CAS latency
- RL,WL – read/write latency
- BL – burst length



[AL = 2 and CL = 3, RL = (AL + CL) = 5, WL = (RL - 1) = 4, BL = 4]

[http://www.jedec.org/download/search/JESD79-2E.pdf]

DRAM memorija, sažetak

- usko grlo performanse zbog velike latencije
  - 50 ns memorijske latencije **naprema** 0.5 ns takta CPU
  - više od 100 ciklusa latencije u najgorem slučaju!
  - u najboljem slučaju oko 25 ciklusa latencije (slijedni pristup, t_CAS)

- sofisticiranom organizacijom (1-4) postiže se veća propusnost uz jednaku latenciju slučajnog pristupa

- svaka instrukcija referencira memoriju 1.3 puta
  - 1× dohvat instrukcije + 30% memorijskih instrukcija
  - da bismo podržali izdavanje instrukcije u svakom taktu, trebalo bi nam više od 100 memorijskih pristupa u svakom trenutku (!!)

10. Priručne memorije

1. Svojstva i organizacija dinamičkog RAM-a
2. **Memorijska hijerarhija**
3. Organizacija priručne memorije
4. Odabir parametara, performansa
5. Izvedbeni detalji

Prostorna i vremenska **lokalnost** pristupa

- **vremenska lokalnost**: korištene lokacije će se vjerojatno koristiti i u budućnosti
- **prostorna lokalnost**: lokacije blizu korištenih lokacija će se vjerojatno također koristiti

Lokalnost pristupa u praksi:

- programska memorija: petlje, potprogrami
- podatkovna memorija: lokalne varijable (stog), članovi objekta, polja, konstante

Korištenje lokalnosti pristupa je **velika ideja**

Radni skup kao mjera **lokalnosti** procesa

Lokalnost programa izražavamo radnim skupom (working set)

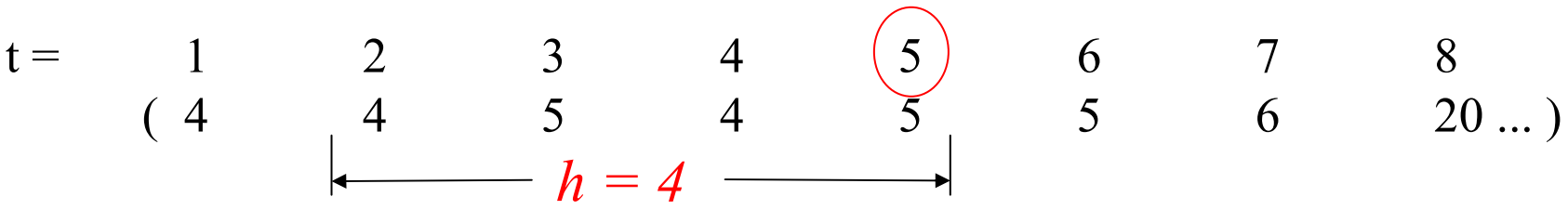
- neka je **slijed naslovljavanja** stranica (blokova od 4KB)

  $R = (r_1, r_2, ..., r_k, ...)$
- tada je radni skup WS(t, h) skup stranica koje su referencirane u posljednjih **h** memorijskih pristupa, počevši od trenutka **t**:

$$WS(t, h) = U\{r_i\},\ i = t\text{-}h+1,\ t\text{-}h+2,\ ...\ ,\ t$$

Primjer:

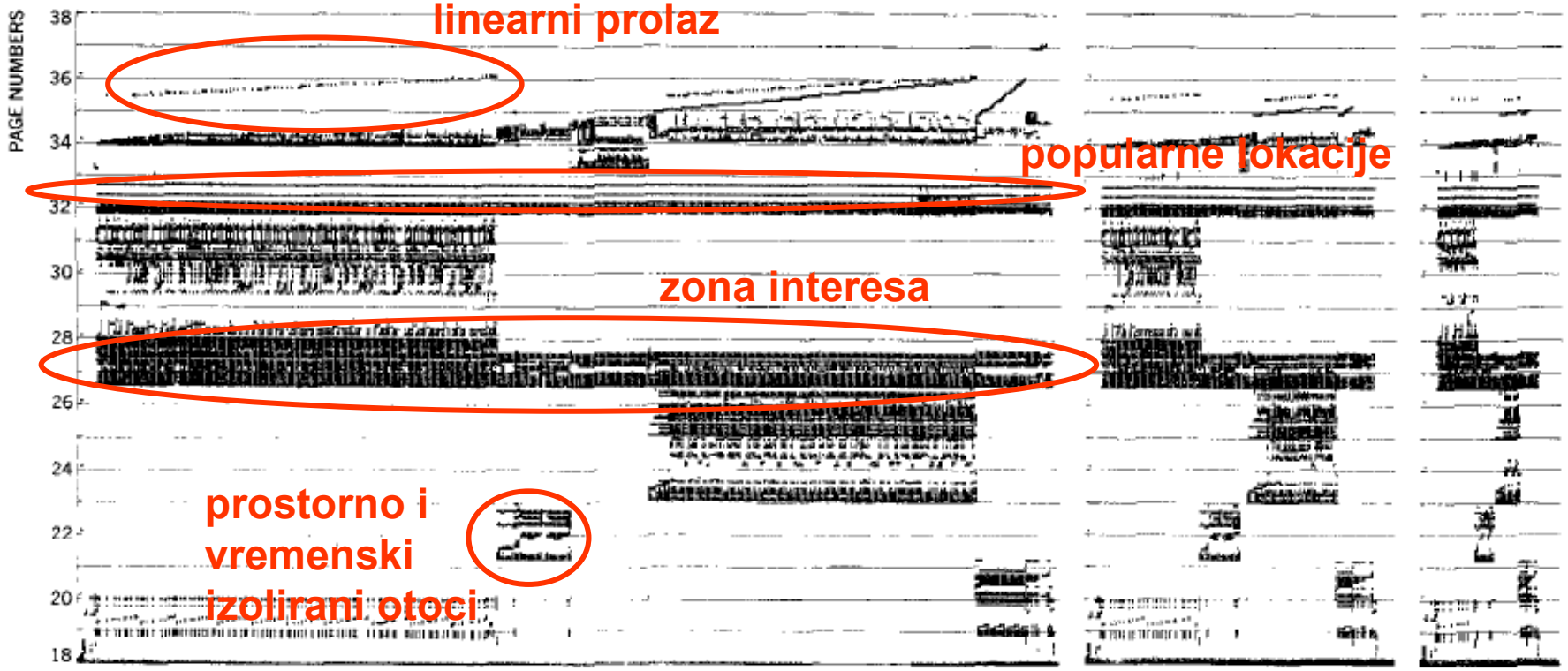
- neka je slijed naslovljavanja stranica R = (4, 4, 5, 4, 5, 5, 6, 20, 20, ...)
- tada je WS(*5, 4*) = {4, 5}



Obrasci korištenja memorije

- pristupi memoriji tijekom prevođenja programa u Fortranu
- apscisa: vrijeme; ordinata: radni skup
- u praksi, vremenska i prostorna lokalnost dobro izraženi

Figure 2 Memory usage during separate compilations



[Hatfield71IBMSJ]

Memorijska hijerarhija

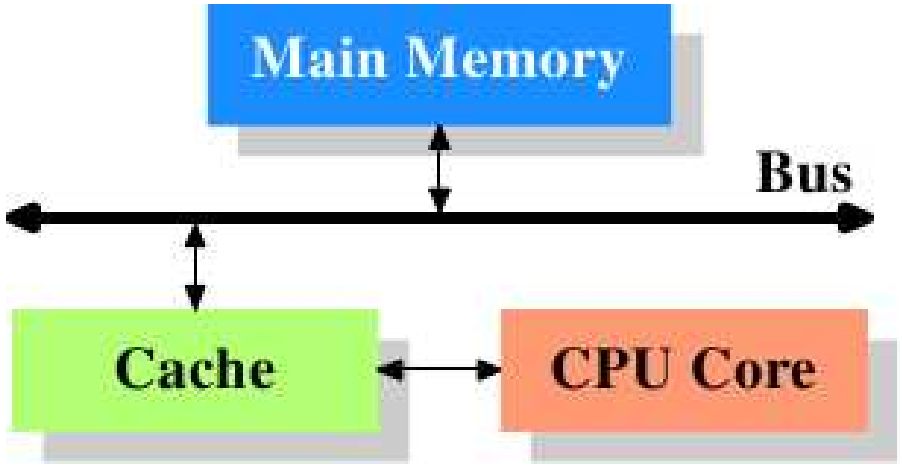
- **ideja**: smanjiti prosječnu latenciju korištenjem lokalnih kopija "popularnih" podataka (radnog skupa) u bržoj memoriji
- niže razine imaju veći kapacitet, veću latenciju i manju cijenu
- prividno, računalo ima kapacitet diska, a brzinu registara (da li je to besplatan ručak? ovisi o inženjerskim plaćama...)

zaporni sklopovi protočne strukture	.05 ns ($1/5 \times \Delta T_{CPU}$), 100B
registri (SRAM)	.25 ns ($1 \times \Delta T_{CPU}$), 500B
L1 cache (SRAM)	1 ns ($4 \times \Delta T_{CPU}$), 64kB
RAM (DRAM)	50 ns, 1 GB
diskovi	10 ms, 1 TB

Osnove priručnih memorija (PM, cache)

- cache: mala brza memorija, blizu procesora

- kad se referencira podatak:
  - ako je kopija podatka u priručnoj memoriji, vrati nju
  - inače:
    - ako je potrebno, izbaci nešto iz priručne memorije
    - dohvati podatak iz glavne memorije (dohvati i susjede)



[drepper07lwn]

Važna pitanja

- kamo smjestiti koju memorijsku lokaciju?
  (RAM>PM $\Rightarrow$ više lokacija RAM-a mora se moći preslikati u istu lokaciju cachea!)
- kako saznati da li je tražena adresa u priručnoj memoriji?
- kako brzo pristupiti cacheiranoj kopiji podatka?
- koje lokacije izbaciti van kad se javi potreba?
- kada upisati promijenjeni podatak natrag u glavnu memoriju?

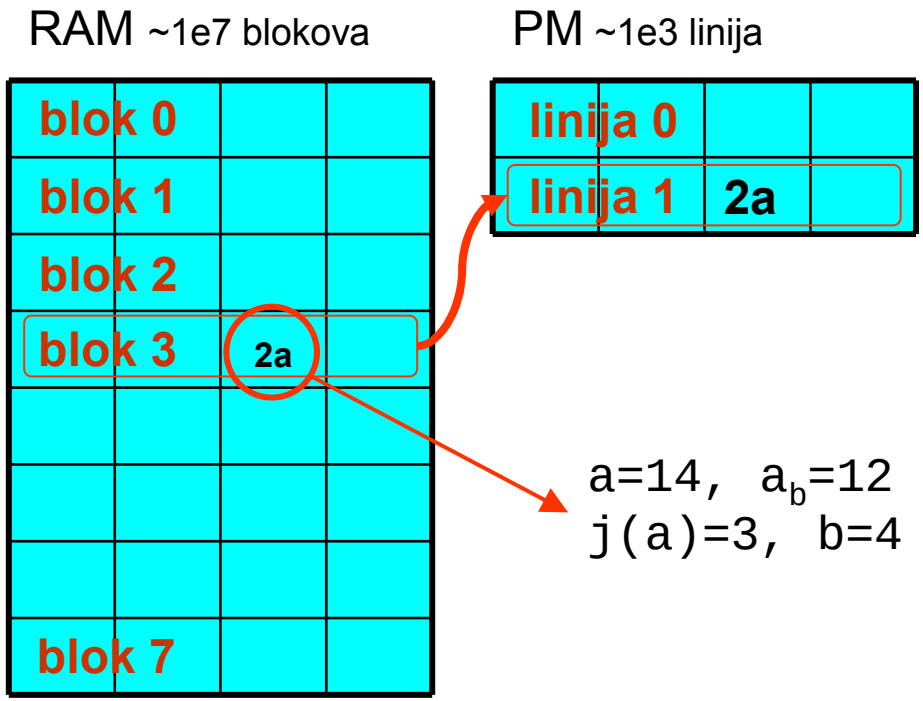
10. Priručne memorije

1. Svojstva i organizacija dinamičkog RAM-a
2. Memorijska hijerarhija
3. **Organizacija priručne memorije**
4. Odabir parametara, performansa
5. Izvedbeni detalji

Osnovna organizacija PM

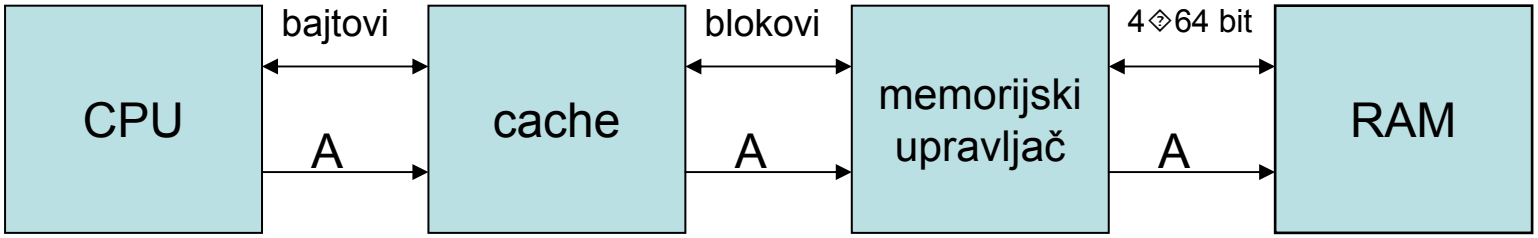
- PM smješta kopije **poravnatih blokova** podataka iz RAM-a
  - veličina bloka **b** izražena u bajtovima je potencija broja 2
  - adresa bloka a_b poravnata u odnosu na veličinu bloka:

    a_b mod **b = 0**



- Svaki bajt memorije pripada točno jednom bloku
  - adresa *a* pripada bloku s indeksom **j**(*a*)= $\lfloor a/\mathbf{b} \rfloor$ ("najveće cijelo")
  - adresa bloka koji sadrži *a* je $a_b(a)=\mathbf{j}(a)\cdot\mathbf{b}$
  - npr (**b**=4, *a* = **14**): ,
    - $\mathbf{j}(a)=\lfloor a/\mathbf{b} \rfloor = \lfloor 3.5 \rfloor = \mathbf{3}$,
    - $a_b(a)=\mathbf{j}\cdot\mathbf{b} = 3\cdot 4 = \mathbf{12}$
  - prihvatno mjesto za tako poravnate blokove nazivamo **linijom** priručne memorije (cache line)

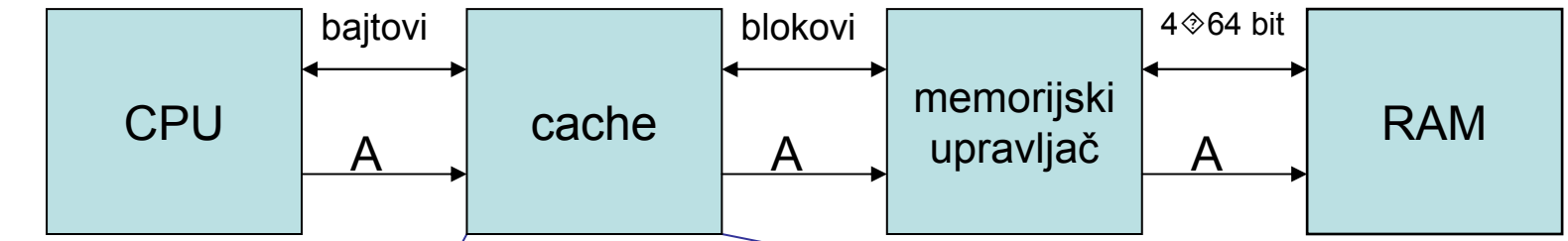
Osnovna organizacija PM (2)

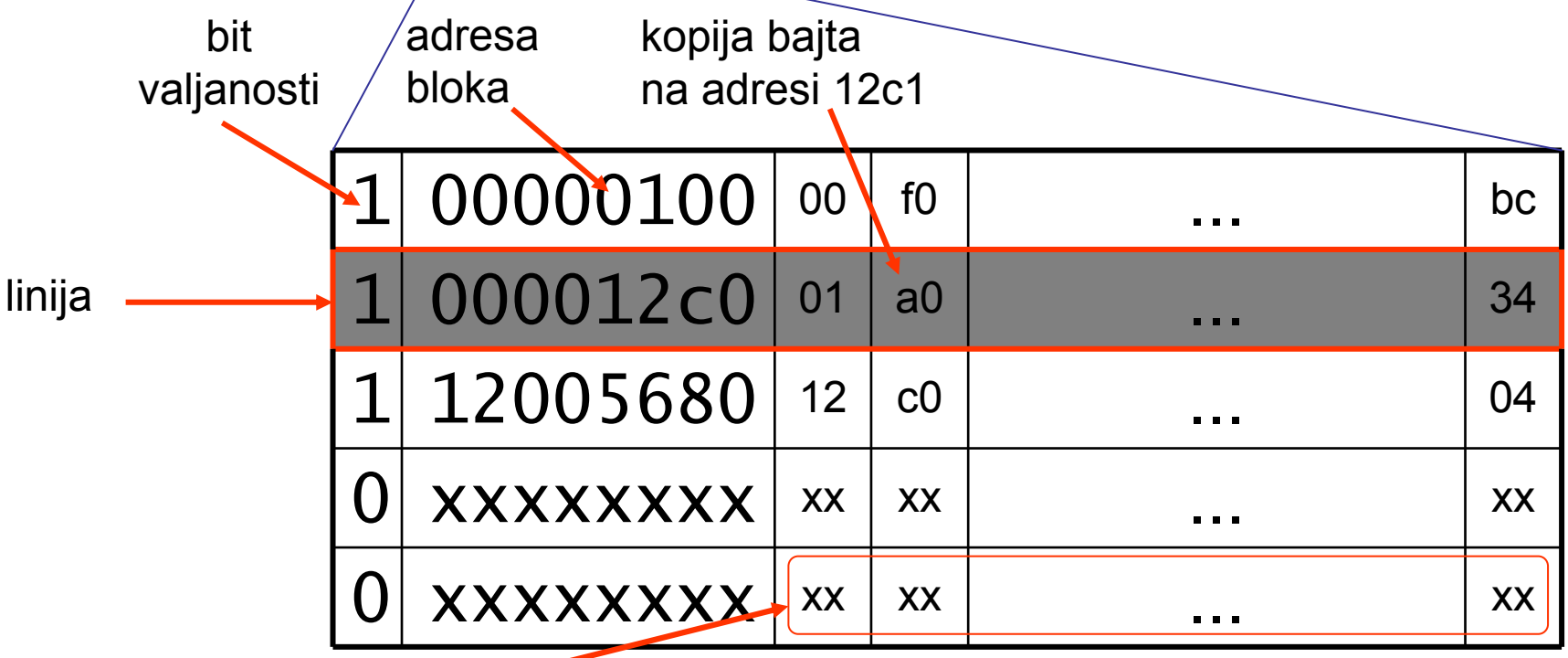


- veličina linije osnovni parametar priručne memorije
  - osnovna struktura PM: broj linija **n** i veličina linije **b**
  - kapacitet priručne memorije tada je: **s** = **n**·**b**
  - linija je kvant prijenosa iz glavne memorije u priručnu (time favoriziramo prostornu lokalnost)

- poravnanje blokova osigurava
  - brzi transfer iz glavne memorije (ili prema njoj)
  - brzi pristup cacheiranim podatcima (o tome ćemo pričati)

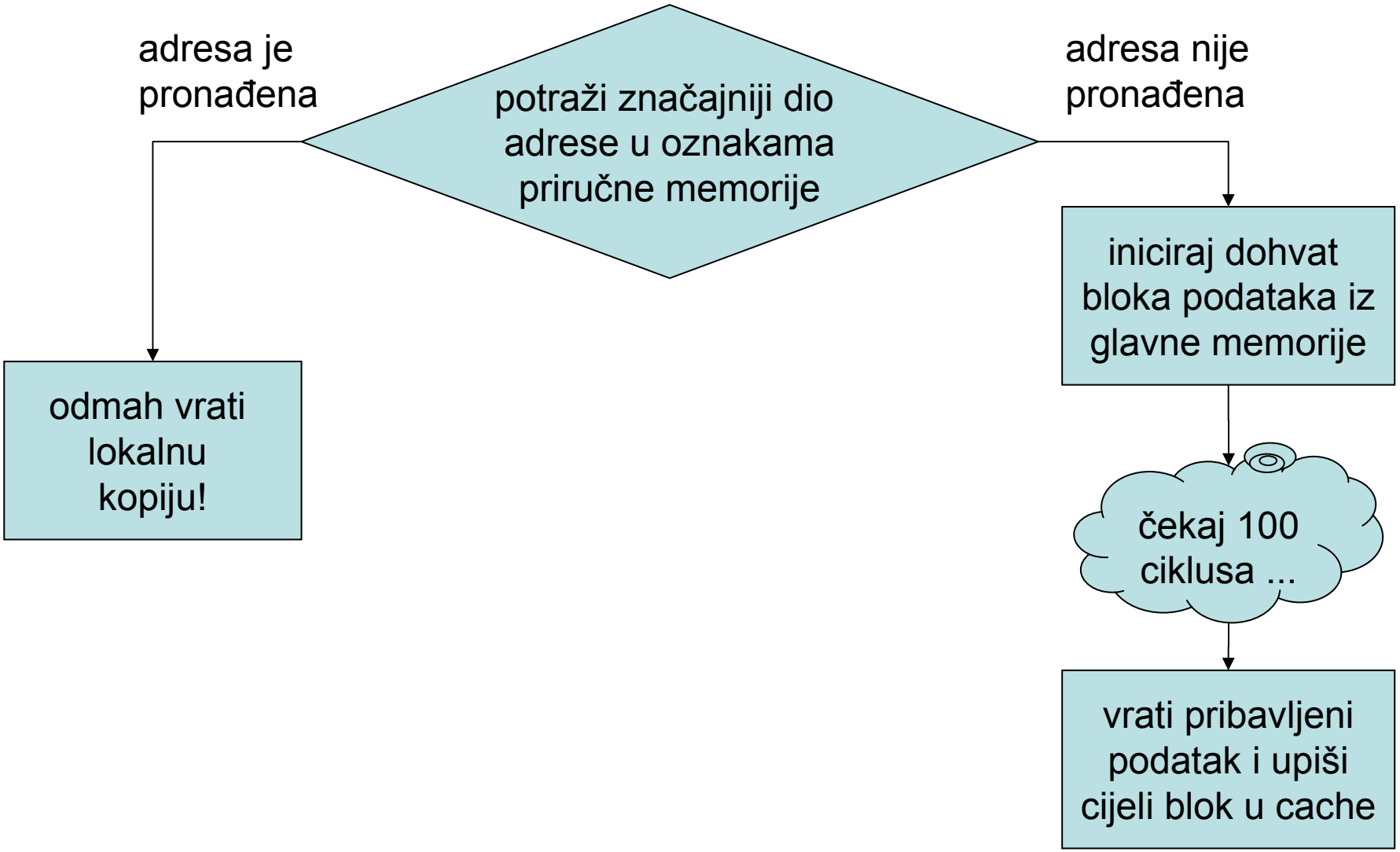
Osnovna organizacija PM (3)





poravnati **blok** podataka (veličina bloka **b** tipično 32B ili 64B)

Čitanje priručne memorije



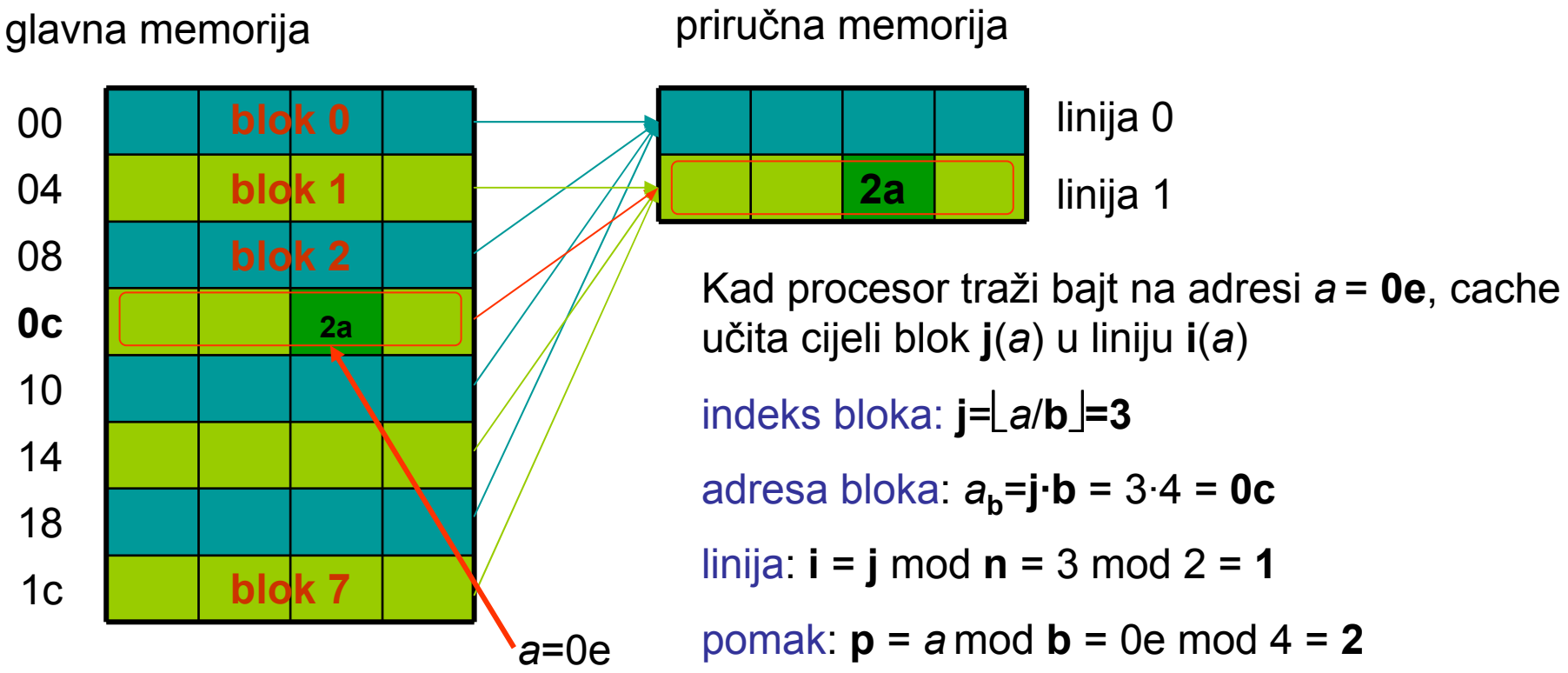
Oblikovanje preslikavanja adresa → linija

- koji memorijski blokovi se mogu upisati u koju liniju?

- moguće izvedbe:
  - izravno preslikavanje
    (blok se preslikava u točno određenu liniju)
  - potpuno asocijativno preslikavanje
    (blok se može preslikati u bilo koju liniju)
  - skupno asocijativno preslikavanje
    (blok se može preslikati u neku od **a** linija)
    - npr, **a**=4: četveroelementno asocijativno preslikavanje
      (4-way associative mapping)
    - npr, **a**=1: izravno preslikavanje
    - npr, **a**=**n**: potpuno asocijativno preslikavanje

Izvedba **izravnog** preslikavanja

- svaki memorijski blok može se preslikati u **samo jednu** liniju
- **princip**: blokove linijama dodjeljivati po modulu **n**
- **i**-ta linija PM prima blokove s rednim brojevima {**j** mod **n** = **i**}
- olakšano provjeravanje i traženje: kopija može biti samo na jednom mjestu (najlakša implementacija)



Kad procesor traži bajt na adresi *a* = **0e**, cache učita cijeli blok **j**(*a*) u liniju **i**(*a*)

indeks bloka: $\mathbf{j}=\lfloor a/\mathbf{b} \rfloor=\mathbf{3}$

adresa bloka: $a_b=\mathbf{j}\cdot\mathbf{b} = 3\cdot 4 = \mathbf{0c}$

linija: **i** = **j** mod **n** = 3 mod 2 = **1**

pomak: **p** = *a* mod **b** = 0e mod 4 = **2**

Detalji izravnog preslikavanja

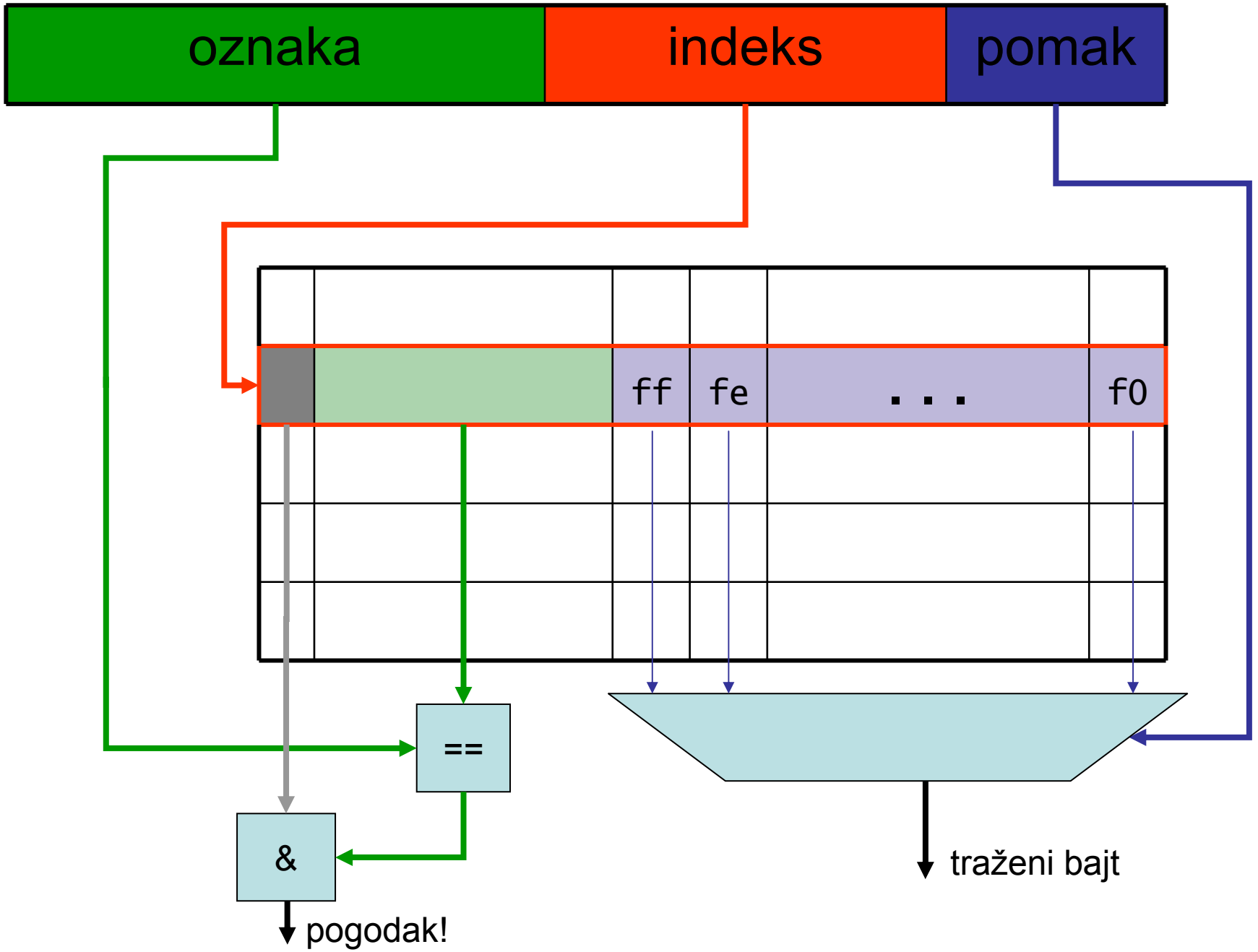
- ukoliko odaberemo “lijepe” **b** i **n,** implementacija iznimno laka
- podijelimo bitove adrese koju generira procesor u tri polja:

  a =oooooooooooooooooiiiiiiiiiipppp

- polje **pomaka** (*offset*) **p**(*a*) = *a* mod **b**
  - adresira bajt u bloku (odnosno liniji)
- polje **indeksa** linije **i**(*a*) = **j**(*a*) mod **n**, **j**(*a*)=$\lfloor a/\mathbf{b} \rfloor$
  - određuje prihvatnu liniju u priručnoj memoriji
- polje **oznake** (*tag*) **o**(*a*) = $\lfloor a / (\mathbf{n} \cdot \mathbf{b}) \rfloor$
  - koristi se za identifikaciju bloka u priručnoj memoriji
- vrijedi: **b** = $2^{w(p)}$, **n** = $2^{w(i)}$

- konkretni primjer je za cache s izravnim preslikavanjem od 1024 linije po 16B (16kB)

PM s izravnim preslikavanjem



Zadatak: neka je zadano:

- PM s izravnim preslikavanjem **s**=16kB, **b**=16B, 32b adresa
- slijed pristupa: 0x00000014, 0x0000001C, 0x00000034, 0x00008014

Odrediti koji će pristupi rezultirati pogotkom!

- pretpostaviti bajtne pristupe početno praznom cacheu

Struktura adrese:

- w(**p**)=$\log_2$**b**=4
- **n**=**s**/**b**=16kB/16B=1024
- w(**i**)=$\log_2$(**n**)=10; w(**o**)=32-14=18
- **adresa:** oooooooooooooooooiiiiiiiiiipppp

Analiza pristupa:

0x00000014 : linija 1, pomak 4 (promašaj)
0x0000001C : linija 1, pomak C (pogodak)
0x00000034 : linija 3, pomak 4 (promašaj)
0x00008014 : linija 1, pomak 4 (promašaj s promjenom)

0x00000030?
0x0000001C?
0x00008020?

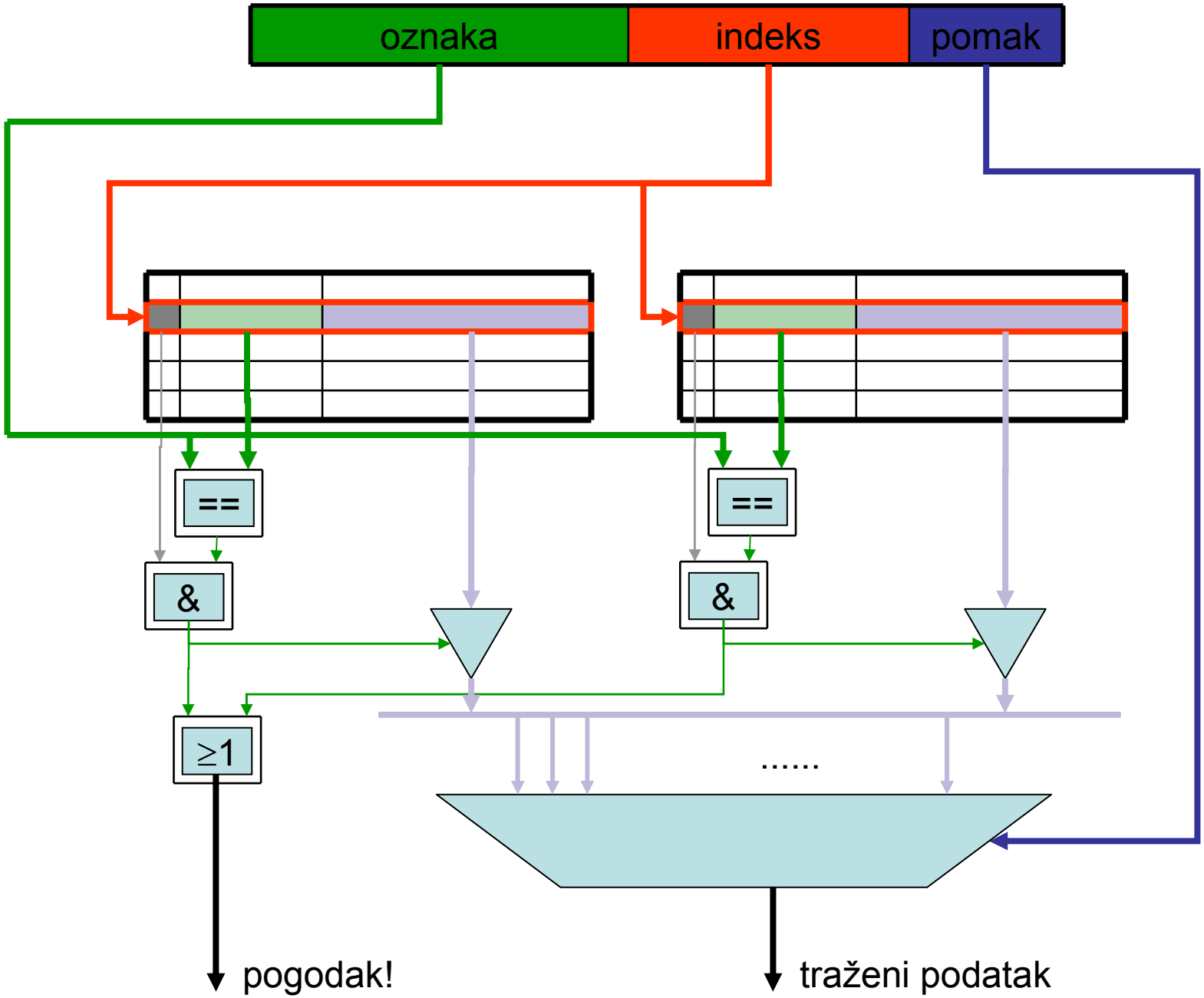
Asocijativno preslikavanje

- Izravno preslikavanje je lako izvesti ali ima nedostataka
  - pristupi s istim indeksom problematični (0x00080014 → 0x0000001c)
  - opetovano pražnjenje i punjenje iste linije

- lijek: priručne memorije sa skupnom asocijativnošću **a**
  - indeks sada adresira skup od **a** linija priručne memorije!
  - odabir između **a** adresiranih linija na temelju oznake
  - u odnosu na izravno preslikavanje s istim brojem linija, indeks je uži za $\log_2(\mathbf{a})$ bitova!
  - povećanje asocijativnosti usporava cache (jednostavnije strukture brže!)

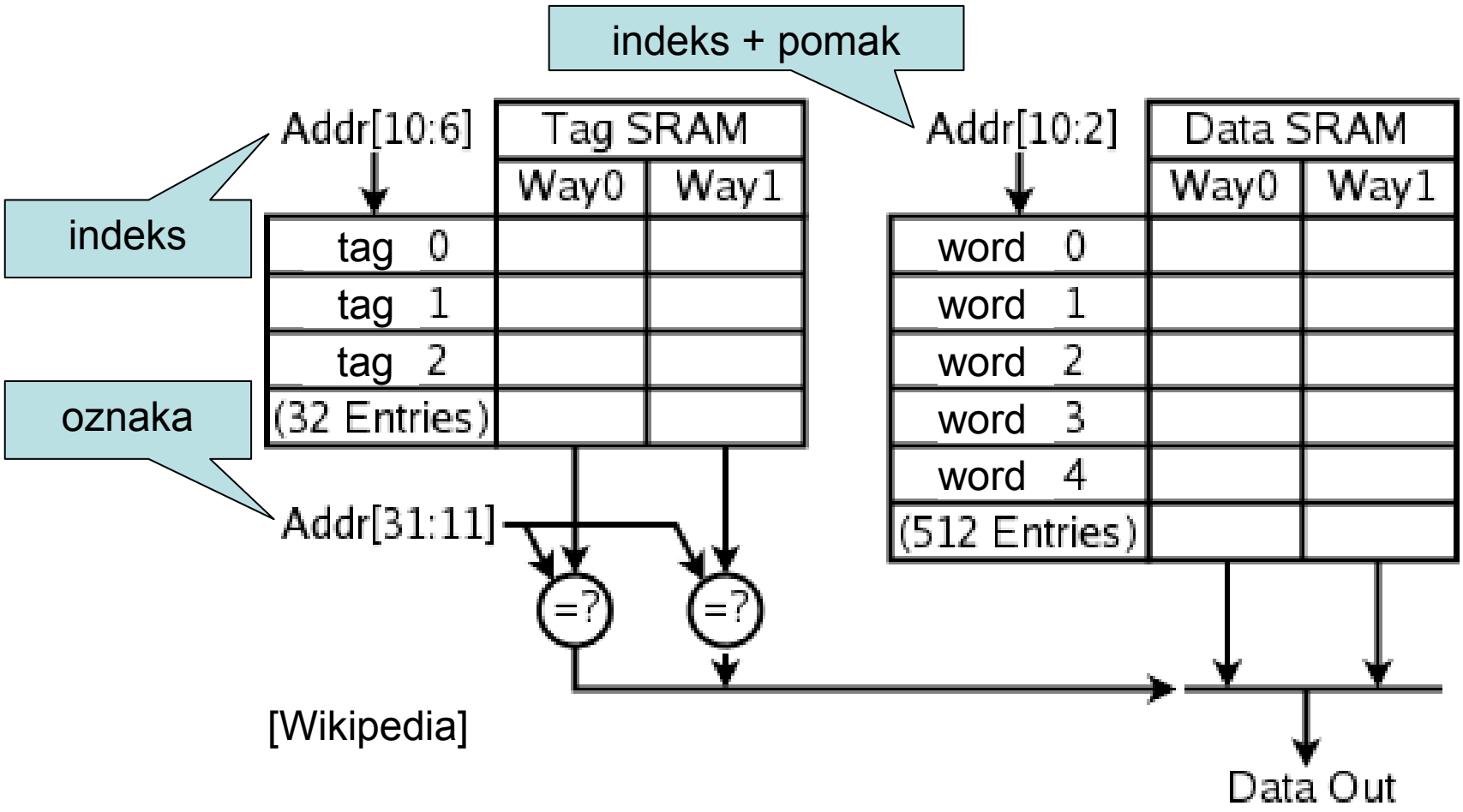
- npr, 32-bitna adresa, 4kB cache, linija od 64B:
  - ukupno **n**=64 linije (4096B/64B) po **b**=64 bajtova
  - izravno preslikavanje: pomak(6b), indeks(6), oznaka(20)
  - 2× asocijativno preslikavanje: pomak(6b), indeks(5), oznaka(21)
  - 4× asocijativno preslikavanje: pomak(6b), indeks(4), oznaka(22)
  - potpuno asocijativno preslikavanje: pomak(6b), indeks(0), oznaka(26)

Asocijativno preslikavanje, a=2



4kB cache, 2× asocijativan, 64-bajtna linija, 32-bitni pristup

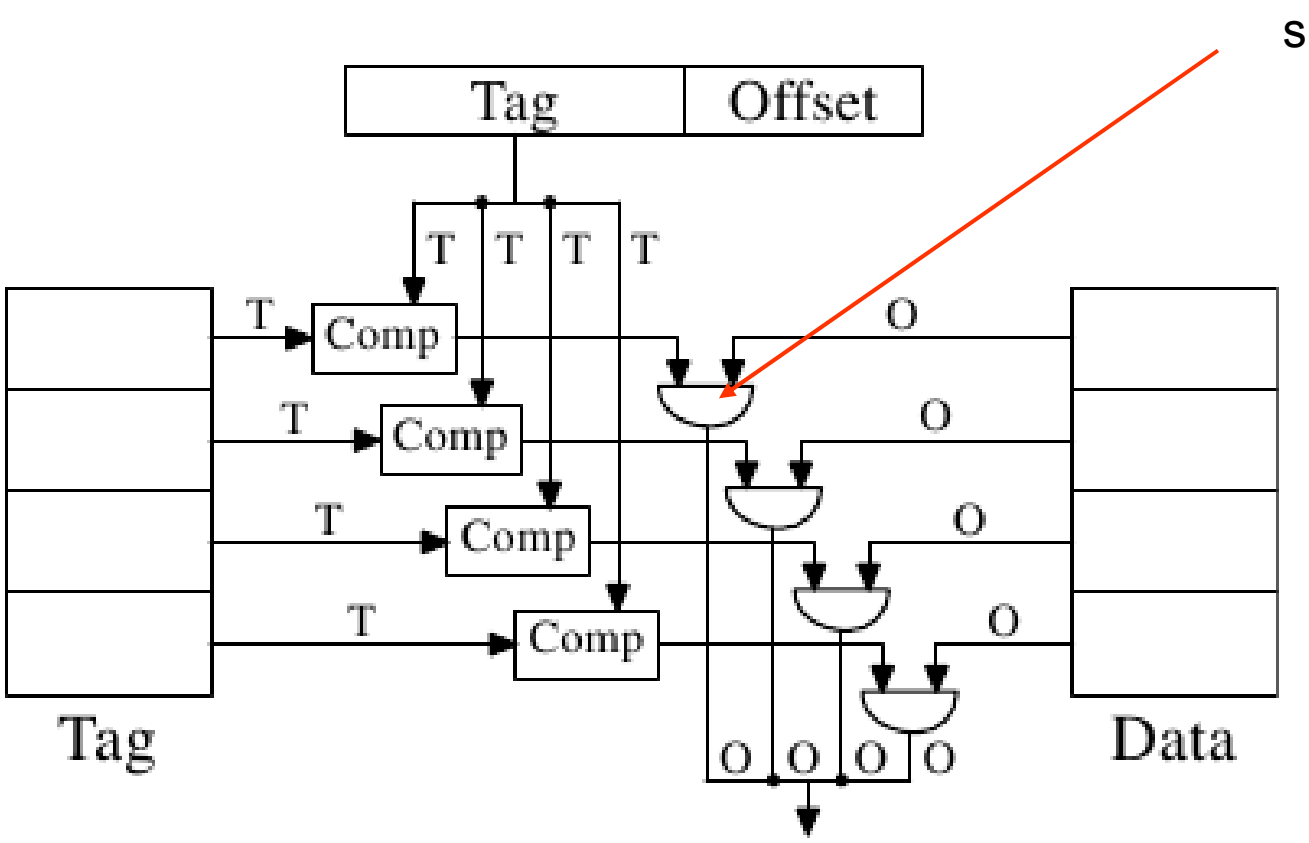
- 4 bita za **pomak** 32-bitne riječi, 5 bitova za **indeks skupa**, 21 bit za **oznaku** adrese (ukupno 30 korisnih bitova adrese)
- u izvedbi, odvaja se **identifikacijski** dio od **podatkovnog** dijela PM
- identifikacijski dio (tablicu oznaka) izravno adresira polje indeksa
- podatkovni dio adresiraju kombinacija indeksa skupa i pomaka (5+4=9 bitova)
- eventualni odabir pribavljenih riječi obavljaju izlazni sklopovi s tri stanja



[Wikipedia]

Potpuno asocijativno preslikavanje

- oznake **svih** linija PM uspoređuju se istovremeno!
- nema polja indeksa: **a**=**n**, w(i)=0
- samo za male PM s vrlo skupim promašajima
- npr, za n=4:



sklop s tri stanja!

[drepper07lwn]

Zadano: 2× asocijativna PM s bajtnom zrnatošću, kapacitet 64 kB, linija od 64 B, 32-bitna adresa, 2 servisna bita (**V**alid,**D**irty).

Odrediti ukupni broj bitova linije PM.

Struktura adrese:

- w(**p**)=$\log_2$**b**=6
- **n**=**s**/**b**=64kB/64B=1024
- w(**i**)=$\log_2$(**n**/**a**)=9;
- w(**o**)=32-9-6=17
- **adresa:** oooooooooooooooooiiiiiiiiipppppp

Struktura linije:

- oznaka (17b), servisni bitovi (2b), podatci (64B)
- ukupno: 17+2+64*8=531b

Organizacija priručne memorije, **međusažetak**

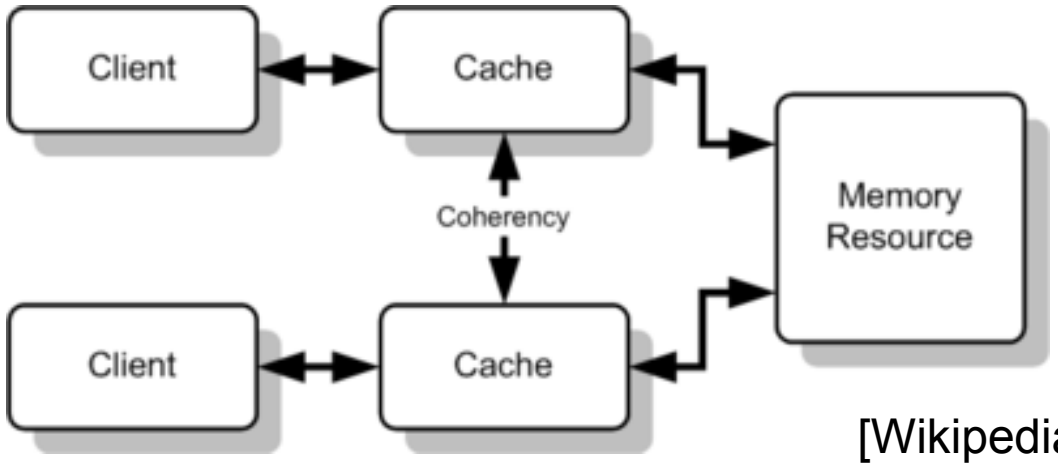
- koristi se lokalnost pristupa kako bi se maksimirala performansa u **čestim** slučajevima
- transparentni prijenos blokova RAM-a u linije PM
- preslikavanje indeksiranjem ili asociranjem (izravno, višeelementno ili potpuno asocijativno)

- još nismo rekli:
  - kada upisati promijenjenu kopiju natrag u glavnu memoriju?
  - koje lokacije izbaciti van kad se javi potreba?
  - kako dimenzionirati cache (**n**, **b**, **a**)?

Što napraviti nakon pisanja u priručnu memoriju?

1. novu vrijednost odmah proslijediti u memoriju (write-through)
   - najsigurniji pristup, ali najveći pritisak na memoriju

2. odgoditi upis (write-back)
   - upisati novu vrijednost samo u cache
   - dodati bit promjene ('dirty' bit) koji pamti da je kopija promijenjena
   - OS upisuje promijenjene linije pri promjeni konteksta ili UI operaciji
   - koherencija je osjetljivo pitanje, posebno kod MP sustava!



[Wikipedia]

Algoritmi zamjene blokova

- kod izravnog preslikavanja, sve je **jasno**:
  - novi blok se upisuje na jedino mjesto, prethodni stanar se izbacuje (ako ga ima)

- kod skupno asocijativnih memorija moramo odabrati blok za izbacivanje
  - LRU: blok koji je najdavnije korišten se izbacuje
    - relativno dobri rezultati (vremenska lokalnost)
    - skupa implementacija za više od 2× asocijativnost
  - NMRU: izbacuje se blok koji nije posljednji korišten
    - jeftina aproksimacija LRU
    - (pseudo) slučajan odabir izbačenog bloka
  - FIFO, random, ...

Zadatak: odrediti pogotke cachea u sustavu s 4-bitnim adresama:

- parametri cachea: a=2, n=4, b=1, LRU
- pristupi: 0, 2, 0, 1, 4, 0, 2, 3, 5, 4

Struktura adrese:

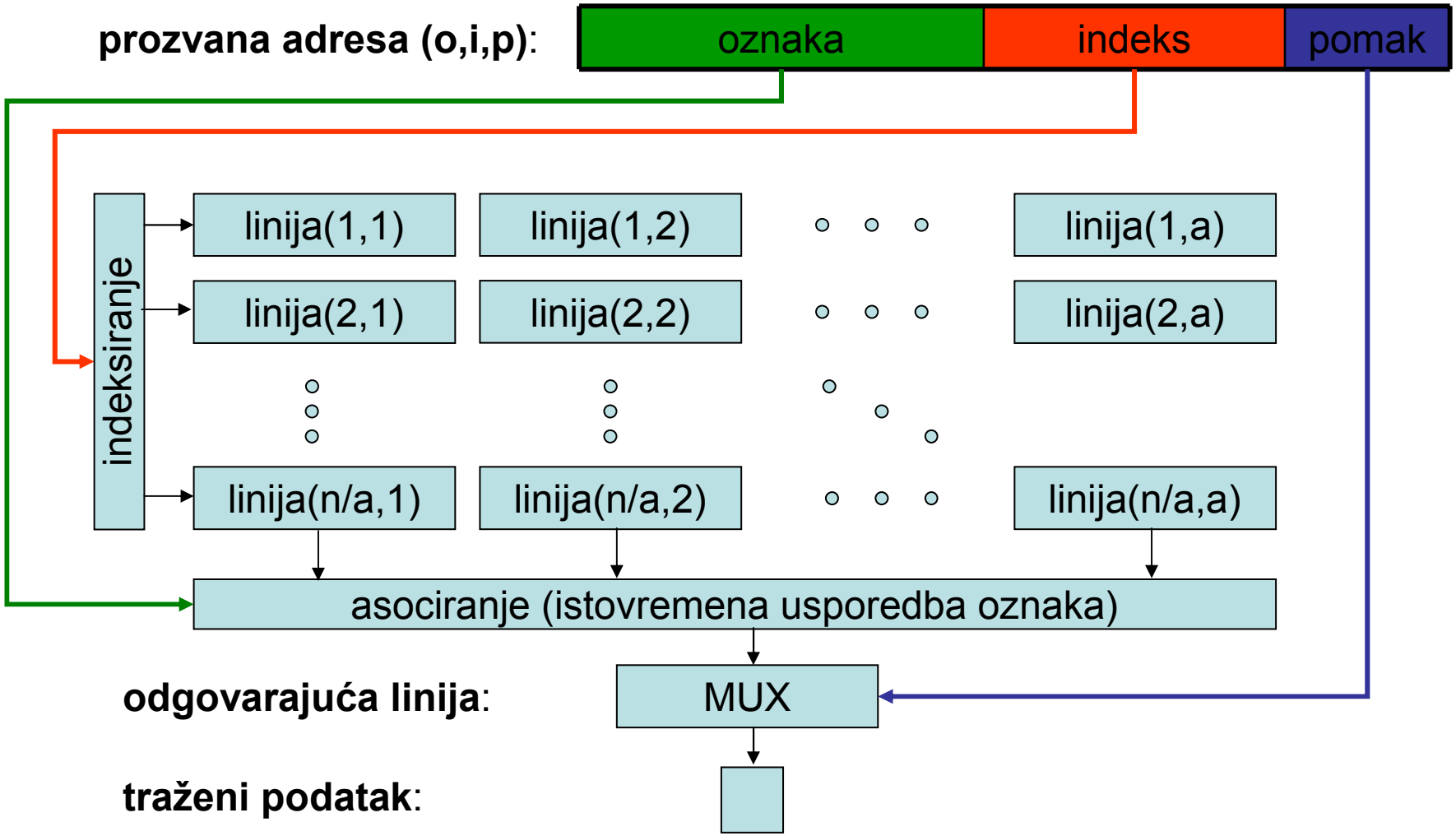
- adresa: oooi
- dvije "linije" za parne, te dvije linije za neparne adrese

Rješenje:

0: promašaj
2: promašaj
0: pogodak
1: promašaj
4: promašaj (izbacuje 2)
0: pogodak
2: promašaj (izbacuje 4)
3: promašaj
5: promašaj (izbacuje 1)
4: promašaj (izbacuje 0)

Sažetak parametara organizacije PM (**n**, **b**, **a**):

- broj linija **n**, broj bajta po liniji **b**, asocijativnost **a**, veličina **s** = **n** × **b**
- indeksiranjem odabiremo 1 od **n/a** skupova linija
- asociranjem odabiremo 1 od **a** linija u skupu

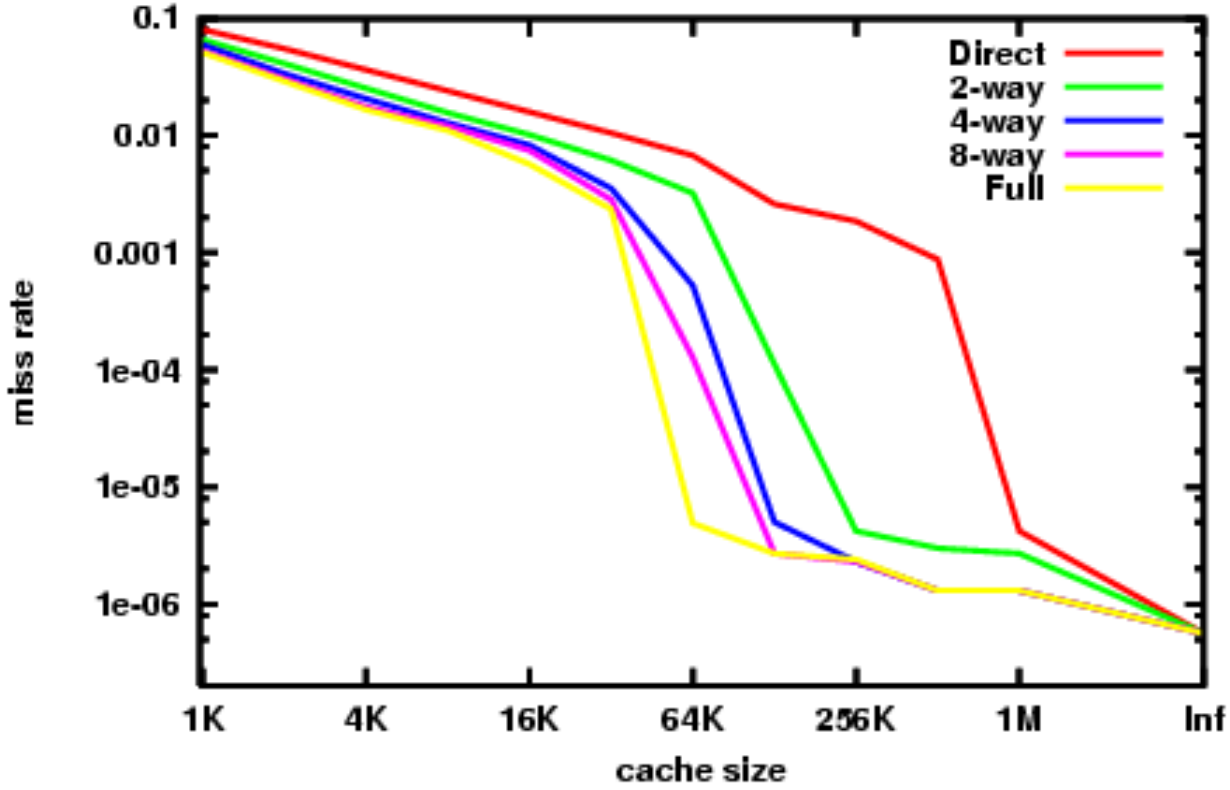


10. Priručne memorije

1. Svojstva i organizacija dinamičkog RAM-a
2. Memorijska hijerarhija
3. Organizacija priručne memorije
4. **Odabir parametara, performansa**
5. Izvedbeni detalji

Kako odabrati veličinu cachea i asocijativnost (**n**, **a**)?

- u mnogome ovisi o raspoloživoj tehnologiji
- ključni su eksperimentalni podatci pribavljeni analizom izvođenja reprezentativnih programa (podatci na slici za SPECint2000)



[Wikipedia]

Analiza promašaja (**3C**)

- **c**ompulsory misses (nezaobilazni):
  - učitavanje svih podataka koji su potrebni za izvođenje programa
  - ne ovise o veličini priručne memorije
  - desni dio grafa pokazuje koliko ih ima (oko 1e-6)

- **c**apacity misses (zbog ograničenog broja linija):
  - dobar pokazatelj je graf za potpuno asocijativno preslikavanje
  - graf pokazuje da je radni skup (working set) između 32kB i 64kB

- **c**onflict misses (zbog neidealne organizacije):
  - mogu se podijeliti na promašaje uslijed **ograničenog preslikavanja** i neprikladnog **algoritma zamjene**
  - pokazatelj promašaja uslijed ograničenog preslikavanja je usporedba s potpuno asocijativnim preslikavanjem

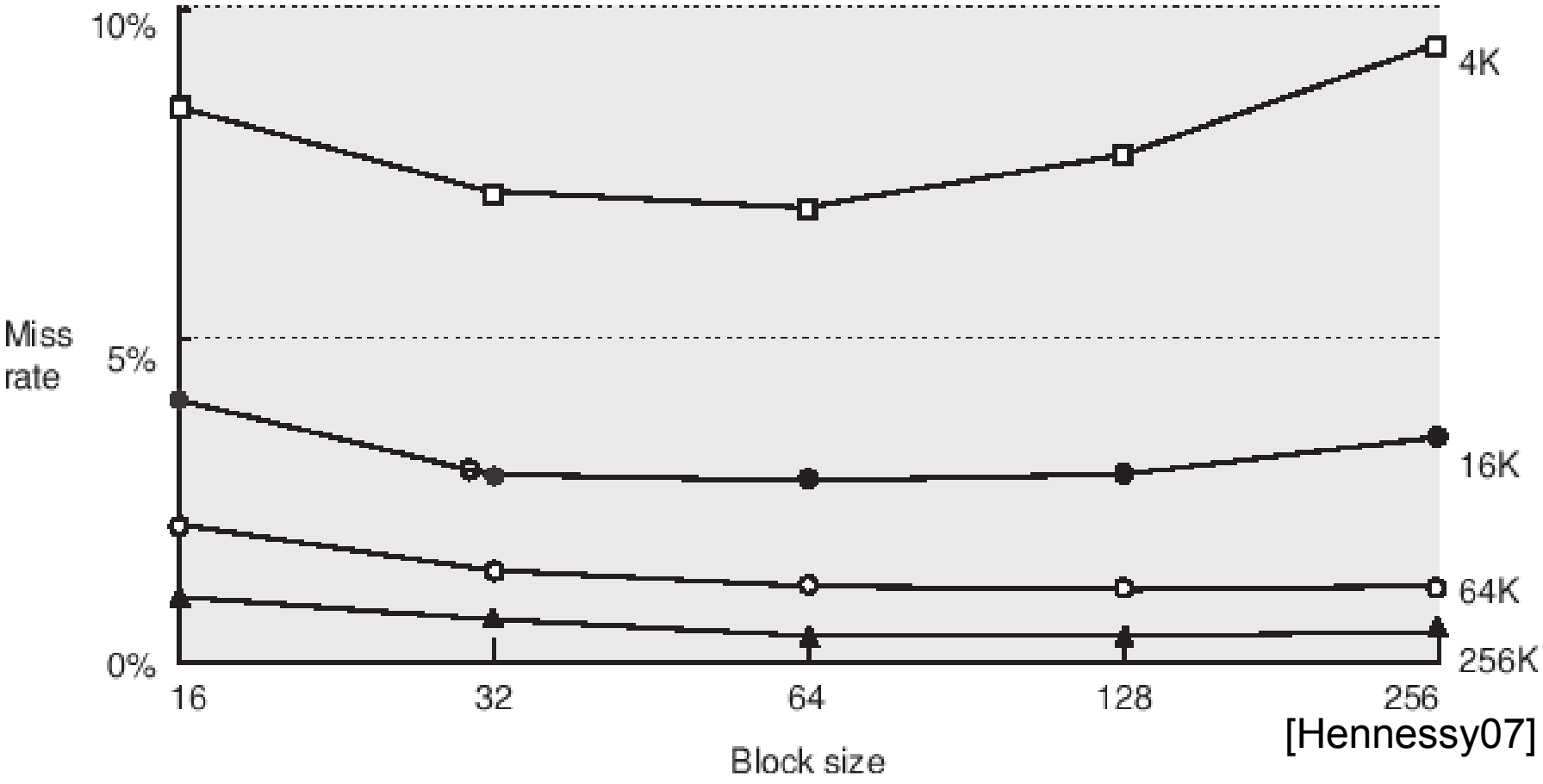
- što smo naučili?
  - za velike i male priručne memorije, izravno preslikavanje prihvatljiv izbor (ali u višeprogramskom kontekstu višestruka asocijativnost je dobra ideja)
  - za srednje priručne memorije povećanje asocijativnosti nužna
  - veličina L1 cachea 32kB – 64kB
  - nema smisla imati priručnu memoriju veću od 1M (L2, L3, po korisniku)

Kako odabrati veličinu linije?

- linija ne smije biti premala...
  - ne iskorištavamo prostornu lokalnost
  - važno za slijedne pristupe (instrukcije, elementi polja)
- ... ali ni prevelika
  - veća cijena promašaja (tražili 1B, dobili prijenos 64B)
  - manje linija $\Rightarrow$ više promašaja, pogotovo u malom cacheu
- recept?
  - optimirati vrijeme pristupa prema modelu
  - najjednostavniji model: prosječno vrijeme pristupa memoriji
    - AMAT: average memory access time
    - t_{AVG} = t(pogodak)·υ(pogodak)+t(promašaj)·υ(promašaj)
    - $t_{AVG} \approx$ t(pogodak) + t(promašaj)·υ(promašaj)
    - model nije prikladan za procesore s dinamičkim raspoređivanjem!
  - u praksi, 16B, 32B i 64B su najčešći odabiri

Kako odabrati veličinu linije (2)?

- Ovisnost postotka promašaja za programe iz SPECint92:
  - najbolji rezultati za linije od 64B
  - za velike linije, učestalost promašaja raste
  - maksimum se pomiče udesno za velike priručne memorije
  - pažnja, velika PM $\Rightarrow$ manji υ(promašaj), ali i veći t(pogodak)!



[Hennessy07]

Prosječno vrijeme pristupa (primjer)

- Average memory access time (AMAT)
  - t_{AVG} = t(pogodak) + ν(promašaj) × t(promašaj)

- Zadano:
  - Period procesorskog takta T=1ns
  - t(pogodak) = 1 T (poznato, PM prati procesor)
  - t(promašaj) = 100 T (tipična vrijednost, ovisi o memoriji)
  - ν(promašaj) = 1% (izmjereno)

- t_{AVG} = 1 T + 0.01 × 100 T = 2 T
  - efektivno vrijeme pristupa je dva ciklusa

Parametri priručne memorije, sažetak:

- veličina memorije **s** (kompromis u odnosu na brzinu!)
- veličina linije **b,** broj linija **n=s/b**
- asocijativnost **a**
- algoritam zamjene (za **a**>1)
- strategija upisa (wb, **wt**)
- L2, L3 (u nastavku predavanja)?

Donekle objektivan pristup dimenzioniranju:

- optimizacija prosječnog vremena pristupa
- čimbenici: budžet, tehnologija (veličina i složenost vs brzina), ciljani programi, ...

Utjecaj PM na performansu računala

- Komponente procesorskog vremena:
  - normalno izvođenje programa
    - uključujući pogotke cachea
  - memorijski zastoji
    - uglavnom uslijed promašaja PM

- memorijski zastoj po instrukciji:

  $T_{MZ} = \nu(\text{promašaja}) \times t(\text{promašaja})$

- Utjecaj na CPI:

  $CPI = CPI_{OSNOVNI} + \nu(\text{promašaja}) \times t(\text{promašaja}) / T_{CPU}$

Utjecaj PM na performansu (primjer)

- Zadano
  - promašaji instrukcijske PM: ν(PIPM) = 2%
  - promašaji podatkovne PM: ν(PPPM) = 4%
  - cijena promašaja: c = 100 ciklusa
  - osnovni CPI (idealna PM): $CPI_O = 2$
  - učestalost memorijskih instrukcija: ν(MI) = 36%

- Broj **c**iklusa **z**astoja uslijed promašaja, **p**o **i**nstrukciji:
  - instrukcijska PM: $CZPI_{IPM} = 1 \times 0.02 \times 100 = 2$
  - podatkovna PM: $CZPI_{PPM} = 0.36 \times 0.04 \times 100 = 1.44$

- Stvarni CPI = 2 + 2 + 1.44 = 5.44
  - usporenje uslijed neidealne PM =2.72$\times$!

Utjecaj PM na performansu, sažetak

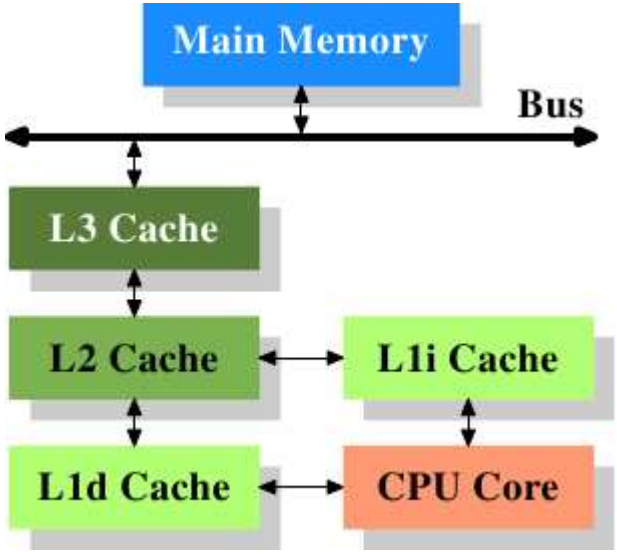
- memorijski zastoji mogu značajno smanjiti efektivni CPI
- performansa procesora raste brže od latencije memorije
  $\Rightarrow$ utjecaj promašaja na performansu raste
- svojstva priručne memorije ne mogu se zanemariti pri evaluiranju performanse sustava

10. Priručne memorije

1. Svojstva i organizacija dinamičkog RAM-a
2. Memorijska hijerarhija
3. Organizacija priručne memorije
4. Odabir parametara, performansa
5. **Izvedbeni detalji**

Višerazinska priručna memorija

- Primarna PM (L1) spojena izravno na CPU (malena, brza)
- Sekundarna PM (L2) servisira L1 promašaje
  - veća, sporija, znatno brža od RAM-a
- RAM servisira L2 promašaje



[drepper07lwn]

- Sofisticirana računala imaju i PM L3
  - u višeprocesorskom sustavu, L3 obično dijele svi procesori
  - asocijativnost L3 mora biti veća od broja procesora!

Višerazinska priručna memorija (primjer)

- zadano:
  - CPI_O=1, T=250 ps
  - ν(promašaj, L1) = 2%
  - t(promašaj) = 100 ns
- samo s PM L1:
  - c(promašaj) = 100 ns / 0.25 ns = 400 ΔT
  - $CPI_{L1} = CPI_O + 400 \times 2\% =$ **9**
- svojstva L2
  - t(pogodak,L2)=5 ns (20 ΔT)
  - ν(promašaj, L2) = 0.5%
- CPI s L1 i L2

  zastoji L1 zastoji L2

  - $CPI_{L1+L2} = CPI_O + 2\% \times 20\ \Delta T + 0.5\% \times 400\ \Delta T$
  - $CPI_{L1+L2} = 1 +$ **0.4** $+$ **2** $=$ **3.4**

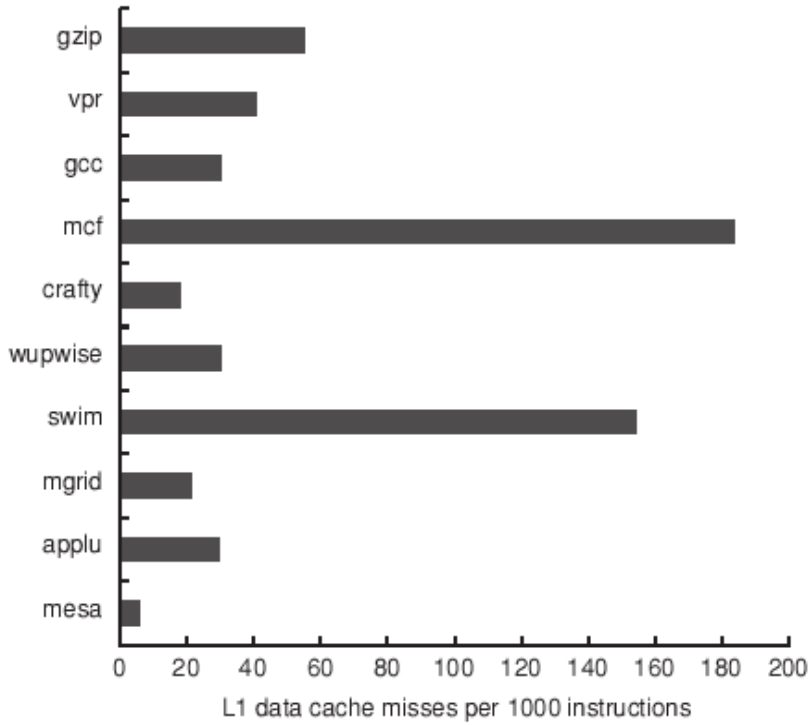
Višerazinska priručna memorija (sažetak)

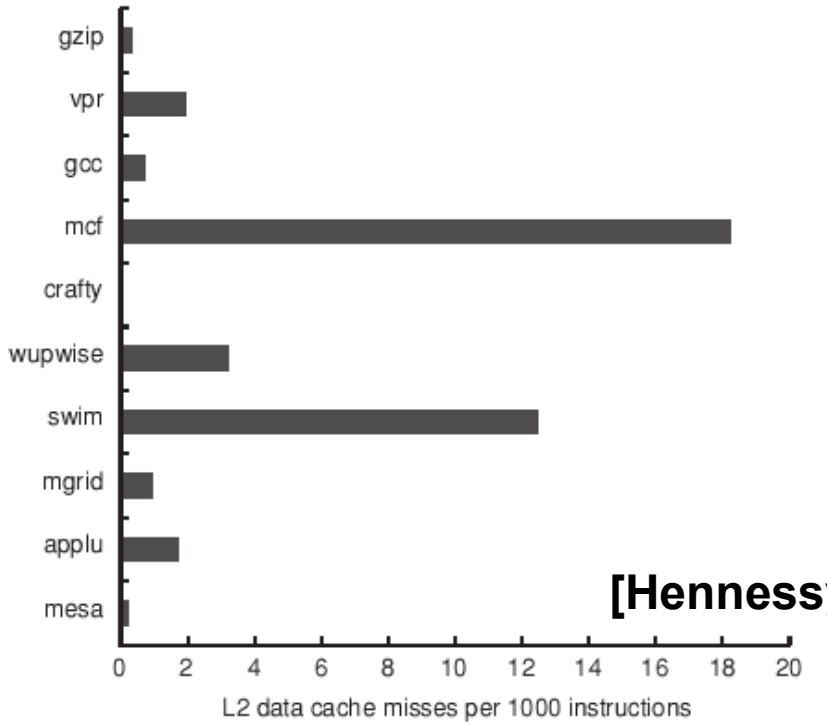
- PM L1: vrijeme pogotka prilagoditi taktu procesora

- PM L2: minimizirati učestalost promašaja
  - vrijeme pogotka manje interesantno

- Zaključci
  - PM L1 tipično manja od jedinstvene PM
  - linija L1 tipično manja od linije L2

Promašaji PM podataka (P4, specCPU2000)





[Hennessy07]

U istom eksperimentu, promašaji PMI zanemarivi (0.2-0.6 %)

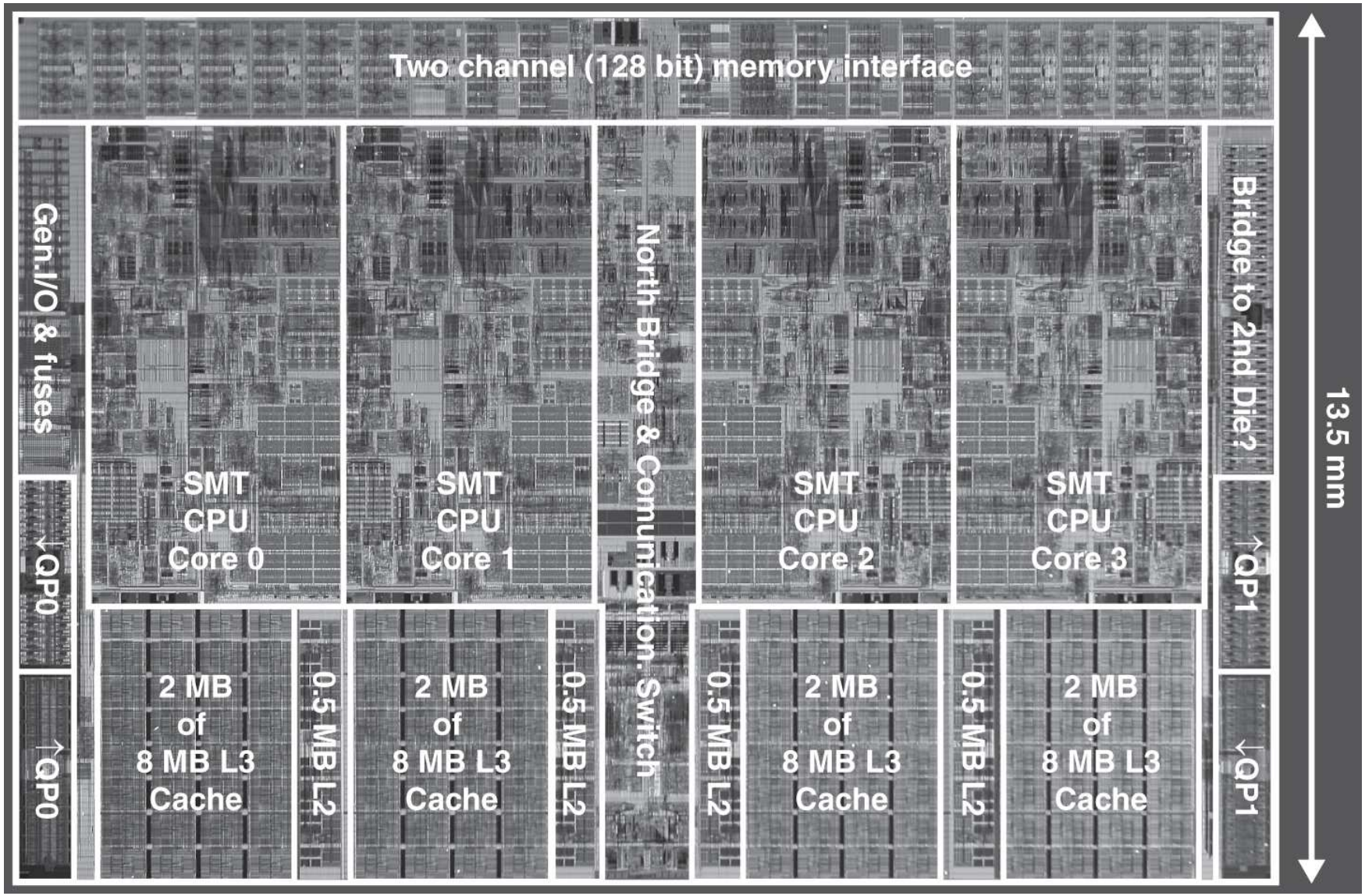
Priručne memorije, primjeri

- Intel Pentium 4 (Prescott, 2004):
  - I$ 12 kμop (4×a), D$: 16 kB (8×a)
  - L2: 2 MB, (8×a)

- Ultra Sparc IV+ (2005)
  - I$ 64 kB (64B, 4×a), D$: 64 kB (32B, 4×a)
  - L2: 2 MB (64B, 4×a)
  - L3: 32 MB (64B, 4×a)

- AMD Athlon 64 (2005)
  - I$ 64 kB (64B, 2×a), D$: 64 kB (64B, 2×a)
  - L2: 1 MB (8×a)

Intel Core i7: I$ 32kB, D$ 32kB, L2 512kB



[Patterson08]

PM u kontekstu dinamičkog raspoređivanja

- Procesori s dinamičkim raspoređivanjem mogu raditi nešto korisno tijekom promašaja cachea!
  - instrukcije load/store čekaju PM u memorijskoj funkcijskoj jedinici
  - ovisne instrukcije čekaju u rezervacijskim redovima
  - neovisne instrukcije se nastavljaju izvoditi!
  - neki procesori omogućavaju više istovremenih pristupa PM!

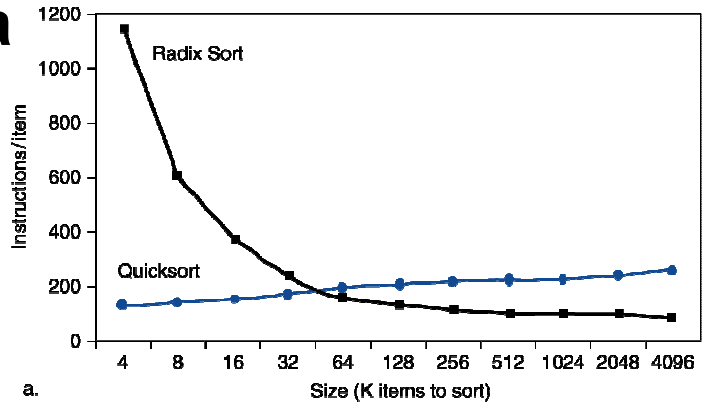
- cijenu promašaja PM teško analizirati
  - učinak promašaja ovisi o strukturi programa
  - jednostavna procjena prosječnog trajanja pristupa memoriji (engl. AMAT) nije relevantna
  - do relevantnijih procjena može se doći simulacijom rada računalnog sustava

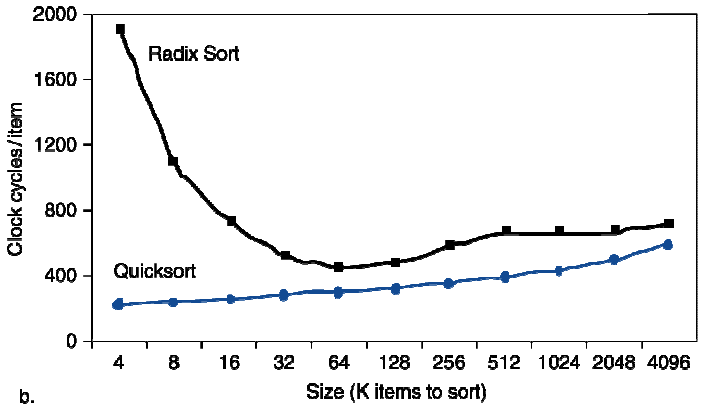
PM u kontekstu zahtjevnih programa

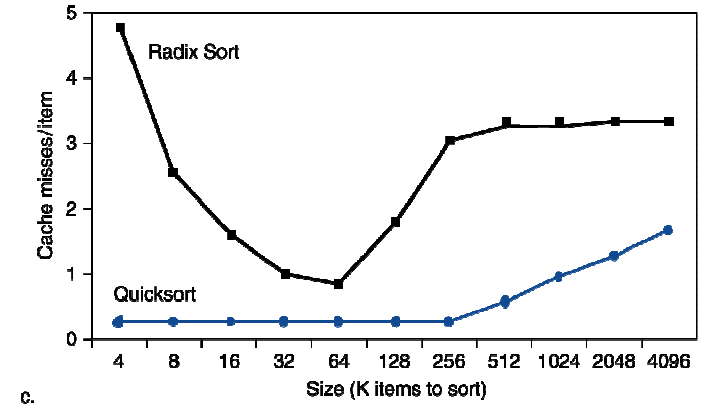
- promašaji ovise o redosljedu pristupa!
  - algoritamska prednost u O-notaciji može se istopiti uslijed suboptimalnog redosljeda pristupa podatcima
  - moderni prevoditelji mogu pomoći!

- prilagoditi strukture podataka liniji PM L1?
  - moderni procesori pružaju mogućnost dinamičkog prilagođavanja programa strukturi memorijskog sustava
  - x86: instrukcija cpuid!

- kakvu PM ima moje računalo?
  - Linux: hardinfo, x86info, cpuid
  - Windows: System Information Viewer







[Patterson08]

Priručne memorije, **sažetak**

- glavna ideja: ubrzati **najčešći** slučaj korištenjem **lokalnosti** pristupa
  - lokalnost pristupa: u zadanom vremenskom intervalu, programi koriste relativno mali dio ukupnog memorijskog prostora
  - brze memorije su malene, velike memorije su spore
  - memorijska hijerarhija nam često donosi najbolje od oba svijeta!

- koncept **cacheiranja** često se koristi u računarstvu:
  - datotečni sustav, preglednici weba, baze podataka
  - općeniti pristup: zapamtiti rezultat skupe operacije i koristiti ga pri naknadnim pozivima

- dimenzioniranje priručne memorije optimiranjem modela izvođenja
  - procjena prosječnog vremena pristupa
  - ovisi o najčešćim programima, tehnologiji, budžetu, ...