Osnovni koncepti operacijskih sustava

Upravljanje spremnikom

Upravljanje spremnikom

- Radni (glavni/središnji) spremnik, ***RAM*** (engl. Random Access Memory – spremnik s izravnim pristupom)
  - radni spremnik je (privremeni) spremnik podataka barem nekoliko puta brži od ostalih spremnika podataka u računalnom sustavu (npr. od tvrdog diska)
  - ostali spremnici se koriste za:
    - trajno spremanje podataka (disk, cd-rom, …) za datoteke
    - kao pomoćni spremnici (kada je radni spremnik premali)
- *Sve* u računalnom sustavu prolazi kroz glavni spremnik:
  - podaci i instrukcije operacijskog sustava moraju se najprije učitati u radni spremnik da bi se izvodili
  - da bi se programi pokrenuli, prethodno se moraju “učitati”
  - podaci za rad programa, kao i njegovi rezultati prolaze kroz radni spremnik
  - priručni spremnici za sporije uređaje/spremnike također se smještaju u radni spremnik

Spremnik za podatke i instrukcije OS-a

- Mora biti zaštićen od ostalih (korisničkih) programa
- Treba biti dostupan samo OS-u (tj. njegovoj jezgri)
- Poželjno je da se cijelo vrijeme rada nalazi u radnom spremniku, zato jer:
  - se vrlo često koristi (u jezgrinim funkcijama)
  - jezgrine funkcije moraju biti brze
    - za vrijeme izvođenja jezgrinih funkcija prekidanje je zabranjeno
    - ako je za izvođenje jezgrine funkcije potrebno dohvatiti dodatne podatke sa sporijih medija to će značajno produljiti izvođenje jezgrinih funkcija

- Osnovne strukture podataka OS-a:
  - upravljanje procesima/dretvama, spremnikom, UI napravama, mrežni podsustav, datotečni podsustav (i pripadni priručni spremnici)

Smještaj programa u radni spremnik

- Program koji se učita u radni spremnik postaje ***proces***

- Spremnički prostor koji proces koristi dijeli se na:
  - segment za instrukcije
  - segment za podatke (varijable, gomilu)
  - stog

- Za upravljanje procesima potreban je ***opisnik procesa***
  - potrebni podaci: ID procesa, prioritet, način raspoređivanja, korišteni spremnički prostor, opisnici datoteka, priručni spremnici za UI naprave, …

- Procesi se moraju biti zaštićeni od drugih procesa

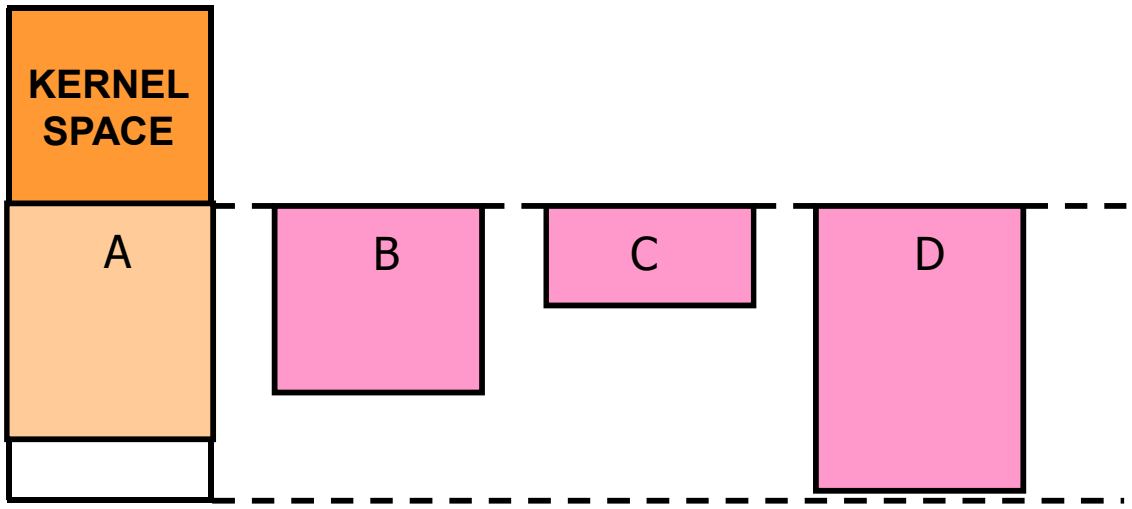
Problemi upravljanja spremnikom

- Kako organizirati spremnički prosto?
  - Gdje staviti jezgru/progame/podatke/isntrukcije/stog/…
- Kako štititi segmente spremnika od nedozvoljenog (namjernog ili nenamjernog) pristupa?
  - od neovlaštene dretve
  - od drugog procesa
- Što ako program ne stane u spremnik (nema dovoljno spremničkog prostora)?
- Kako sklopovlje može pomoći upravljanju spremnikom? Koji su zahtjevi?

- "Dinamičko upravljanje spremnikom" – operacije **`malloc`**/**`free`** (ili **`new`**/**`delete`**) ?
  - ove operacije se ovdje ne razmatraju!

Upravljanje u “najjednostavnijim sustavima”

- “Jednostavni sustavi”:
  - samo jedan program (“OS” je ugrađen u program)
  - više programa koji se samo slijedno mogu izvoditi (jedan po jedan – nikada dva ili više istovremeno)
    - primjeri: ručna računala, mobiteli, …
- Mogući način upravljanja za ovakve sustave
  - OS učitati u jedan dio spremnika (podaci i instrukcije)
  - program (ili programe) u drugi dio
    - ako ima više programa, u ovom ih segmentu izmjenjivati (učitavati jedan po jedan iz pomoćnog spremnika)



Upravljanje u “najjednostavnijim sustavima”

- Kada jedan program završi, drugi se učita na njegovo mjesto te započinje s radom; itd.

- Na sličan način, uz postojanje većeg pomoćnog spremnika može se ostvariti i **višeprogramski** rad (više programa “paralelno radi” i prisutno je u sustavu)
  - pri zamjeni programa, program koji je u radnom spremniku treba pohraniti na **pomoćni**, te s njega učitati drugi “**velika**” zamjena konteksta (ne samo sadržaja registara) značajno dulje traje (pomoćni spremnik je **sporiji** !!!)

- Ostali sustavi (složeniji) moraju imati mehanizme da u spremniku drže više od jednog programa – inače je učinkovitost značajno smanjena

Zahtjevi prema upravljanju spremnikom

- OS (jezgra) i neki programi moraju biti prisutni cijelo vrijeme u spremniku
- Više od jednog programa treba istovremeno biti prisutno u spremniku
  - ako je potrebno, “velika promjena konteksta” može se obaviti “u pozadini” (npr. preko DMA jedinice) tako da za to vrijeme drugi programi mogu iskoristiti procesor
- Procesi trebaju biti razdvojeni – zaštićeni jedan od drugoga
  - dretve istog procesa dijele adresni prostor – tj. sve unutar procesa
  - dretve različitih procesa trebaju biti odvojene

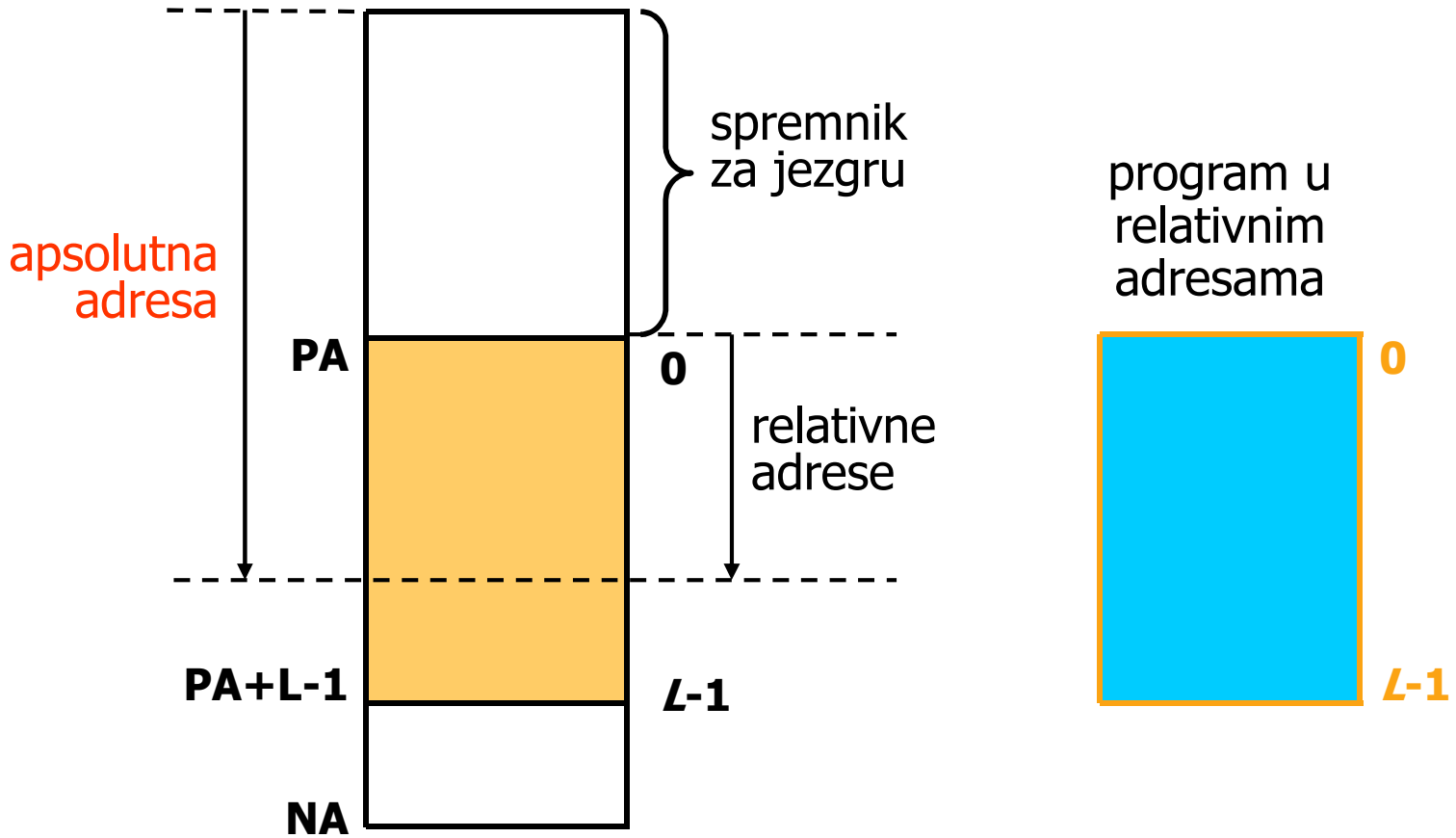
Dodatni zahtjevi za upravljanje spremnikom

- Postupci koji će omogućiti izvođenje i programa koji ne stanu u radni spremnik
- Fragmentacija treba biti što manja
- Što manji sklopovski zahtjevi
- Upravljanje spremnikom treba biti skriveno od programa i programera – ne bi trebalo tražiti posebne tehnike u programima kojima se prilagođava načinu upravljanja spremnikom
  - izuzeci su posebni slučajevi kada se traži optimalna učinkovitost ili minimalno kašnjenje program, kada poznavanje načina rada i posebnog sučelja može biti od koristi

Osnovni principi upravljanja spremnikom

- **Statičko upravljanje spremnikom**
  - podjela spremnika u particije (po jedna za svaki program)
  - može se koristiti sklopovska potpora (za zaštitu)
- **Dinamičko upravljanje spremnikom**
  - dinamički se određuje spremnički prostor za pojedini novi proces
  - *zahtijeva sklopovsku potporu*
- **Straničenje** (“virtualni spremnik”)
  - programi se dijele na stranice, spremnik na okvire
    - veličina jedne stranice jednaka je veličini jednog okvira
  - stranice se učitavaju u okvire
  - pretvorba logičkih adresa (koje koriste procesi, *relativne adrese*) u fizičke adrese (koje koristi sabirnica, *apsolutne adrese*) obavlja se korištenjem sklopovske potpore (MMU – Memory management unit)

Relativne (logičke) i apsolutne (fizičke) adrese



Primjer programa u relativnim i apsolutnim adresama

program (na disku, prije učitavanja, u relativnim adresama)

```
  0  (početak)
     .
 20  LDR R0, (100)
 24  LDR R1, (104)
 28  ADD R2, R0, R1
 32  STR R2, (120)
 34  B 80
     .
     .
 80  CMP R0, R3
     .
     .
100  DD 5
104  DD 7
     .
120  DD 0
```

proces (učitan na početnu adresu = 1000, apsolutne adrese)

```
1000  (početak)
      .
1020  LDR R0, (1100)
1024  LDR R1, (1104)
1028  ADD R2, R0, R1
1032  STR R2, (1120)
1034  B 1080
      .
      .
1080  CMP R0, R3
      .
      .
1100  DD 5
1104  DD 7
      .
1120  DD 0
      .
1500  (vrh stoga)
```

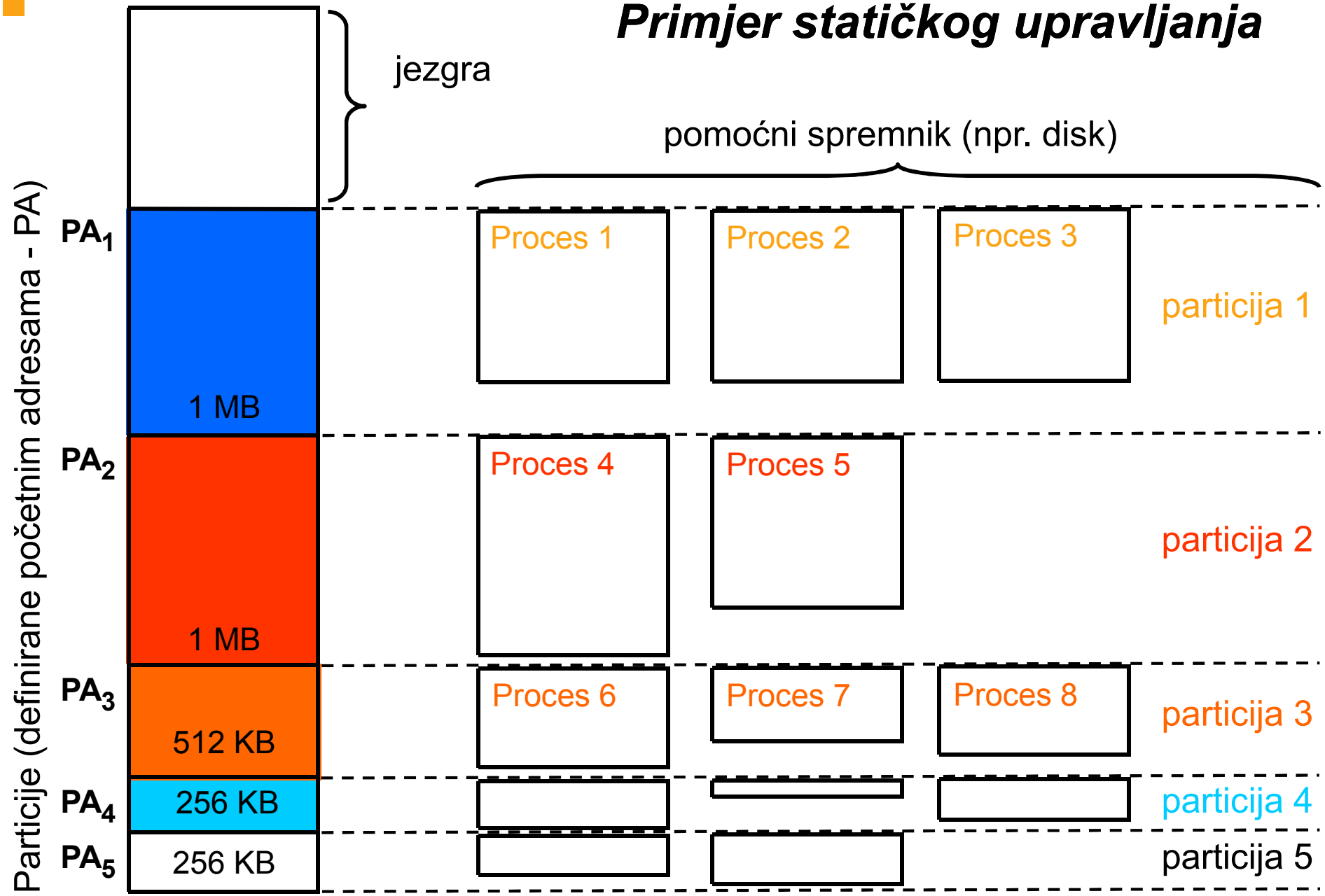
proces (učitan negdje u spremnik, ali još uvijek u relativnim adresama)

```
  0  (početak)
     .
 20  LDR R0, (100)
 24  LDR R1, (104)
 28  ADD R2, R0, R1
 32  STR R2, (120)
 34  B 80
     .
     .
 80  CMP R0, R3
     .
     .
100  DD 5
104  DD 7
     .
120  DD 0
     .
500  (vrh stoga)
```

Statičko upravljanje spremnikom - particije

- Spremnik je podijeljen u particije fiksnih veličina
  - različite veličine za različite programe
- Za svaku particiju pripremljeno je nekoliko programa (na pomoćnom spremniku)
  - ili samo jedan program koji se u tu particiju učitava
- Kada program koji se nalazi u particiji završi ili se blokira:
  - pokreće se zamjena programa drugim programom pripremljenim na pomoćnom spremniku za istu particiju
    - npr. korištenjem DMA sklopa
  - aktivira se drugi program koji se već nalazi u drugoj particiji (i pripravan je za izvođenje – u redu pripravnih)
    - procesor ne čeka na učitavanje programa s pomoćnog spremnika već izvodi program koji se nalazi u radnom spremniku, ali u nekoj drugoj particiji
- Nikakvo dodatno sklopovlje nije potrebno
  - ali bi se moglo iskoristiti ako postoji: za zaštitu particija

Primjer statičkog upravljanja



Nedostaci statičkog upravljanja spremnikom

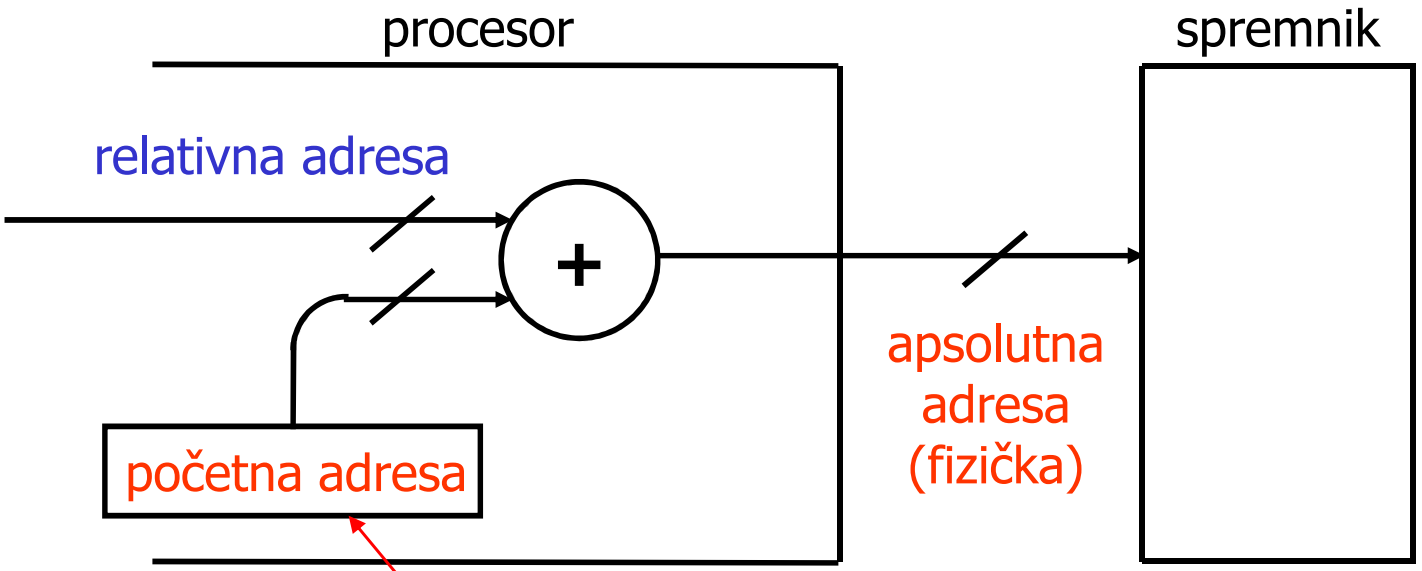
- Nema zaštite između: procesa ⇔ procesa ⇔ jezgre
  - ako postoji sklopovska potpora može ju se iskoristiti

- Fragmentacija
  - *interna* – neki programi neće koristiti cijelu particiju za koju su pripremljeni
  - *externa* – svi procesi pridjeljeni isto particiji u nekom trenutku mogu biti blokirani te iako u sustavu ima drugih pripravnih programa na pomoćnom spremniku, oni se ne mogu učitati u tu particiju jer za nju nisu pripremljeni (već su pripremljeni za neku drugu)!

- Nije moguće izvoditi program koji cijeli ne stane u neku particiju (kada ni najveća particija nije dovoljno velika)

Dinamičko upravljanje spremnikom

- Programi ostaju (učitavaju se u spremnik) koristeći relativne adrese
- Potrebna je sklopovska podrška za pretvorbu adresa
  - *zbrajalo* koje će zbrajati *početnu adresu* (PA) sa *relativnom adresom* koja se generira u programu



- Procesi se mogu učitavati bilo gdje u spremniku
  - dovoljno je u *bazni registar* učitati *početnu adresu*

Svojstva dinamičkog upravljanja spremnikom

- Bolje od statičkog
  - manja unutarnja i vanjska fragmentacija
  - programi ostaju u relativnim adresama

- Uz dodatne komparatore adresa dobiva se mogućnost zaštite (proces ⇔ proces i proces ⇔ jezgra)

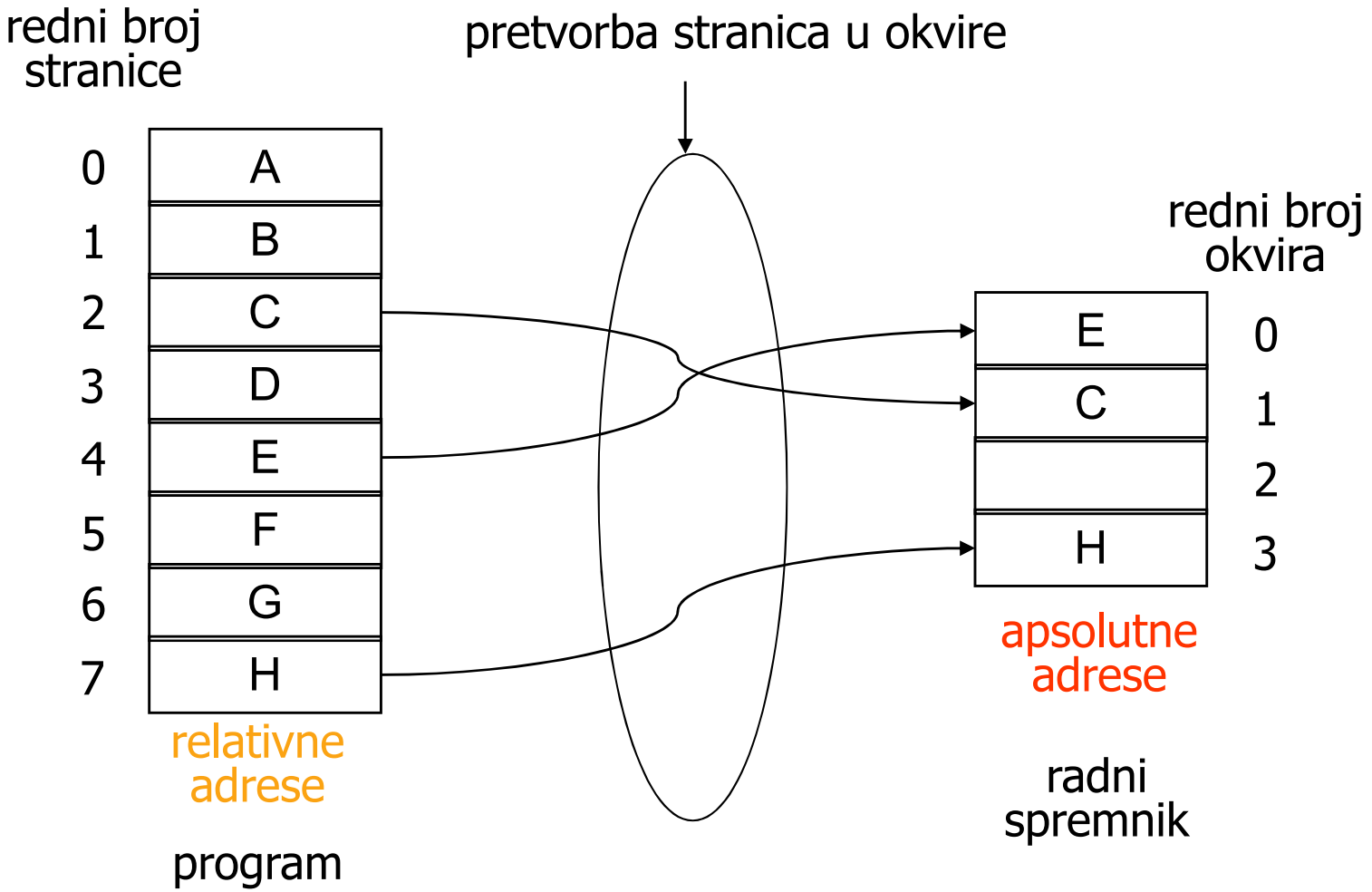
- Nedostaci algoritma
  - fragmentacija:
    - u dinamičkom okruženju, gdje se programi pokreću, izvode, završavaju, pokreću novi, itd. pojavljuje se fragmentacija slobodnih segmenata zbog koje neki novi programi se ne mogu učitati ni u najveći prazan segment

  - i dalje se ne može izvoditi program koji ne stane u radni spremnik

Straničenje

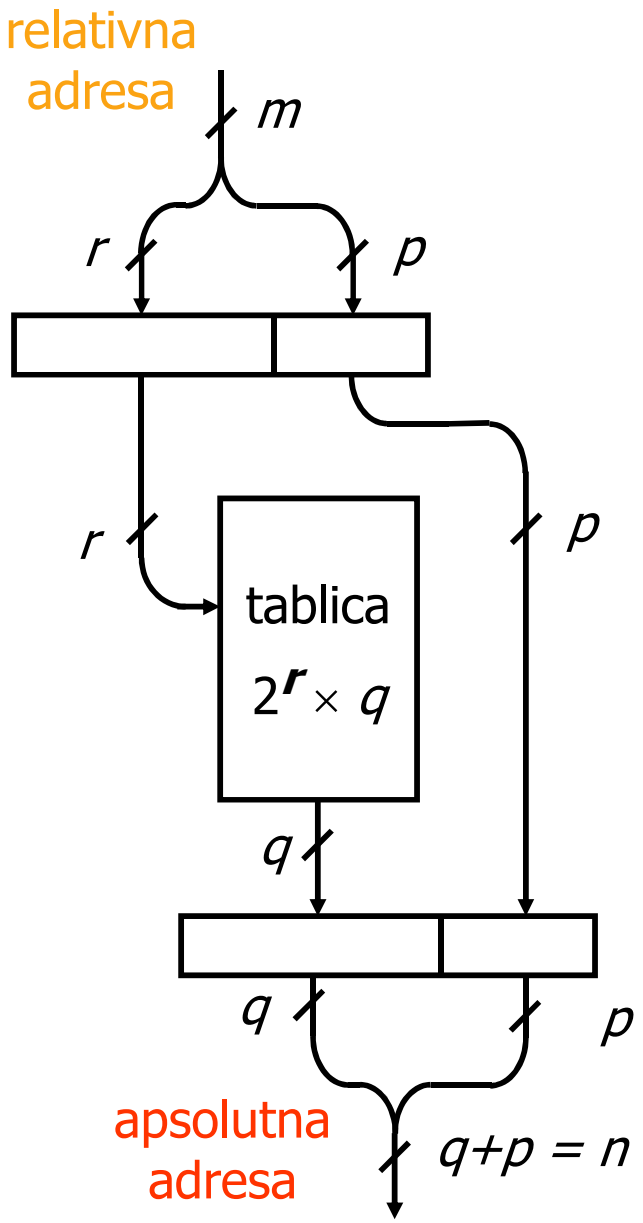
- Upravljanje spremnikom nije jednostavno!
- Za bolje načine upravljanja potrebno je složenije sklopovlje
- Osnovne ideje za ostvarenje:
  - podijeliti programe u dijelove – stranice
  - podijeliti spremnički prostor u dijelove - okvire
    - veličina jedne stranice odgovara veličini jednog okvira
  - u spremnik (okvire) učitavati samo trenutno potrebne dijelove programa (procesa)
    - program pripremiti na pomoćnom spremniku (proces)
    - učitavati one stranice koje se trenutno traže
    - one koje se ne koriste trenutno mogu se i “izbaciti” iz radnog spremnika da bi se napravilo mjesta za one koje su potrebne
    - ako ima dovoljno radnog spremnika učitava se sve
  - programi ostaju u relativnim adresama
  - zaštita se ostvaruje mehanizmom pretvorba relativnih u apsolutne adrese

Osnovni koncept straničenja

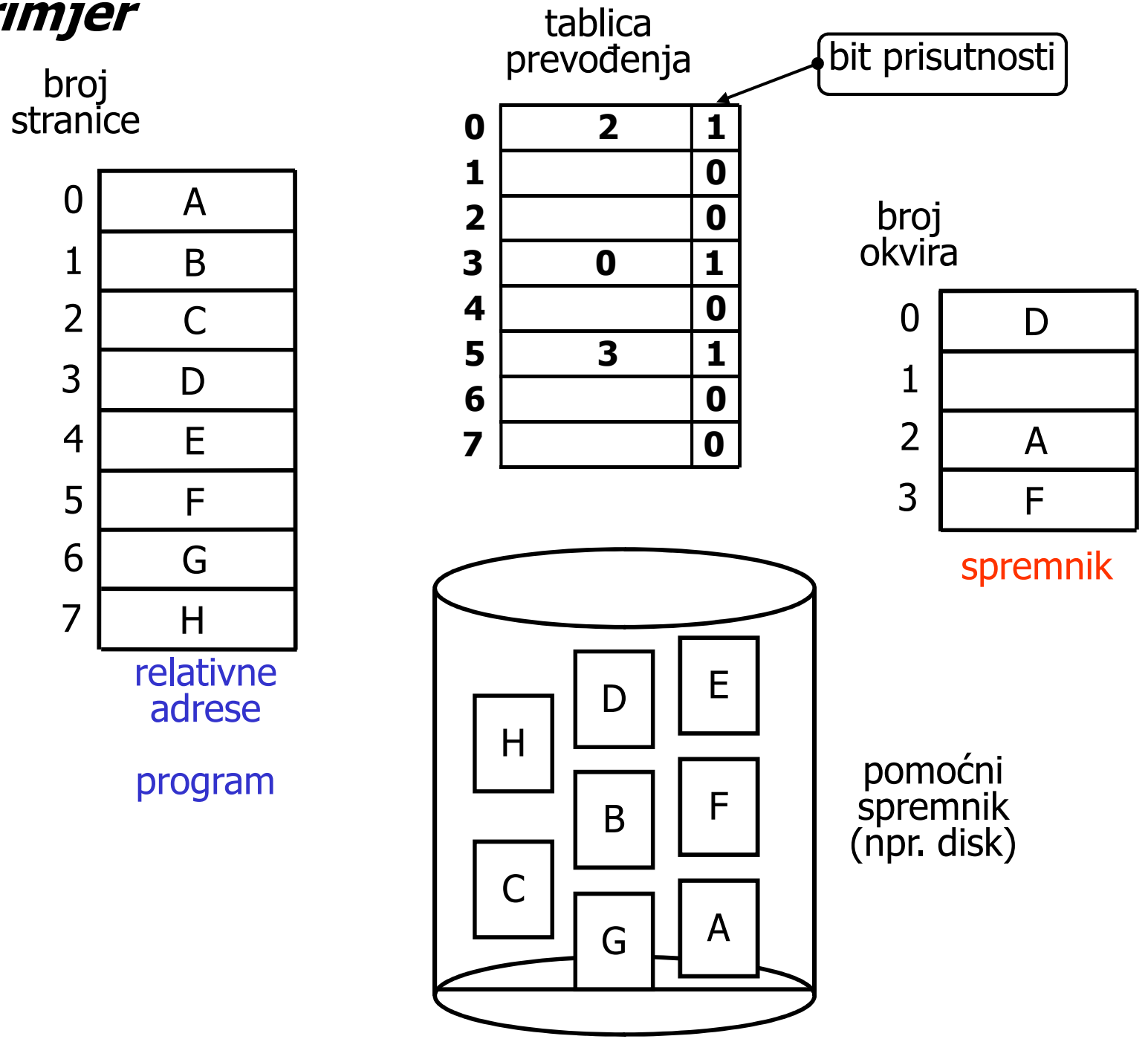


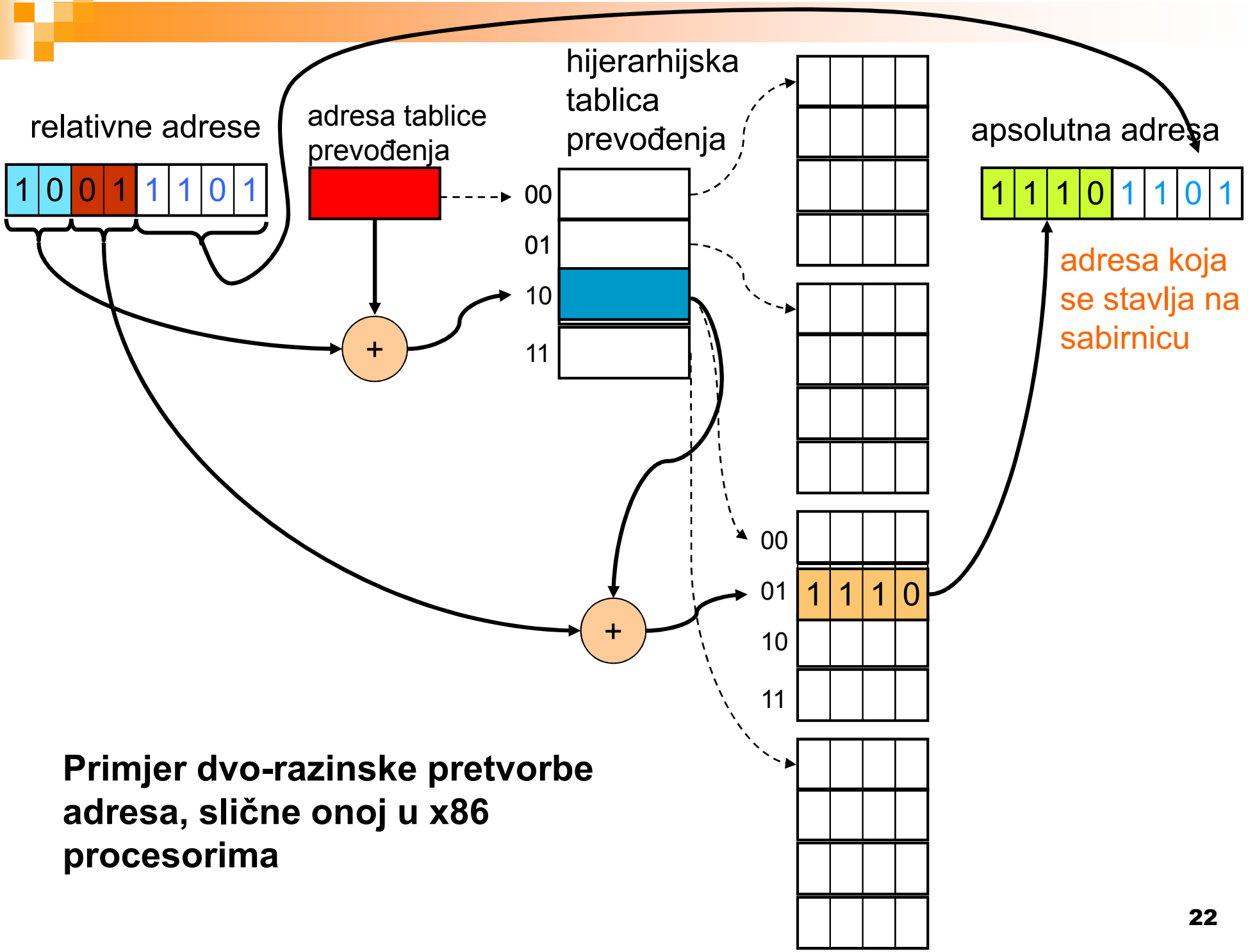
Pretvorba adresa

- Relativne u apsolutne
  - širina relativne adrese: *m* bita
  - širina apsolutne adrese: *n* bita
  - *m* može biti različito od *n*
- Relativna adresa se sastoji od:
  - rednog broja stranice: *r* bita
  - adrese unutar stranice: *p* bita
- Pretvorba adresa = određivanje rednog broja okvira koji sadrži zadanu stranicu
  - koristi se ***tablica prevođenja***
  - pretvorba se obavlja ***sklopovski***



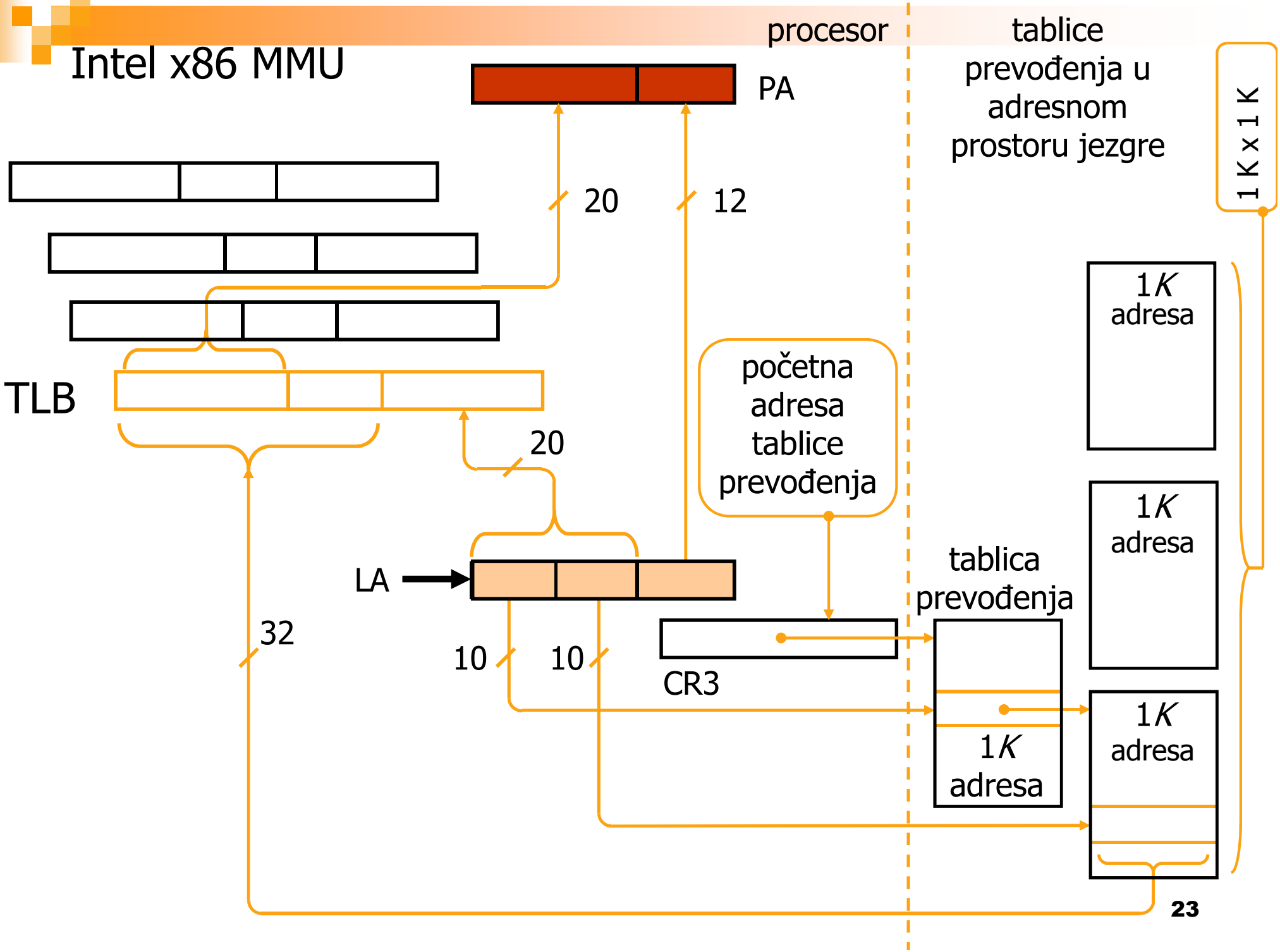
Primjer


broj
stranice
0 A
1 B
2 C
3 D
4 E
5 F
6 G
7 H
relativne
adrese
program
tablica
prevođenja
bit prisutnosti
0 2 1
1 0
2 0
3 0 1
4 0
5 3 1
6 0
7 0
broj
okvira
0 D
1
2 A
3 F
spremnik
H
D
E
B
F
C
G
A
pomoćni
spremnik
(npr. disk)



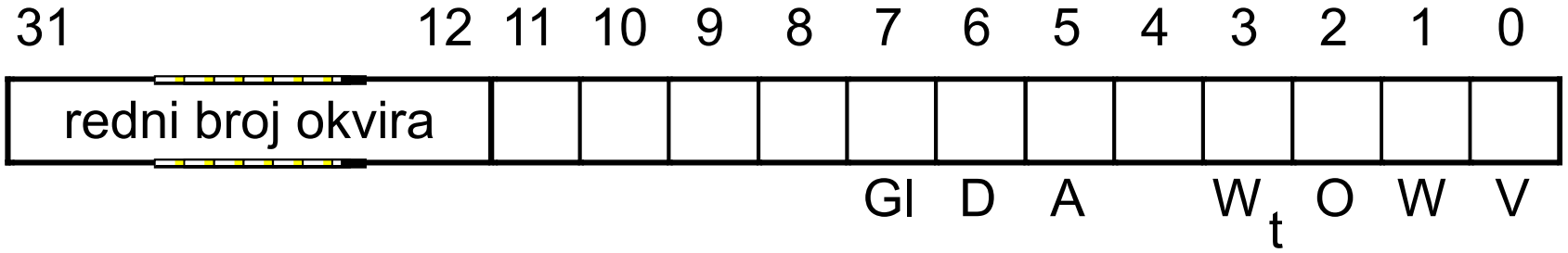
Primjer dvo-razinske pretvorbe adresa, slične onoj u x86 procesorima

Intel x86 MMU



Tablica prevođenja

- Za svaku stranicu postoji zapis u tablici prevođenja:
  - redni broj okvira u kojem se stranica nalazi
  - razne zastavice, npr. za x86:



Zastavice:

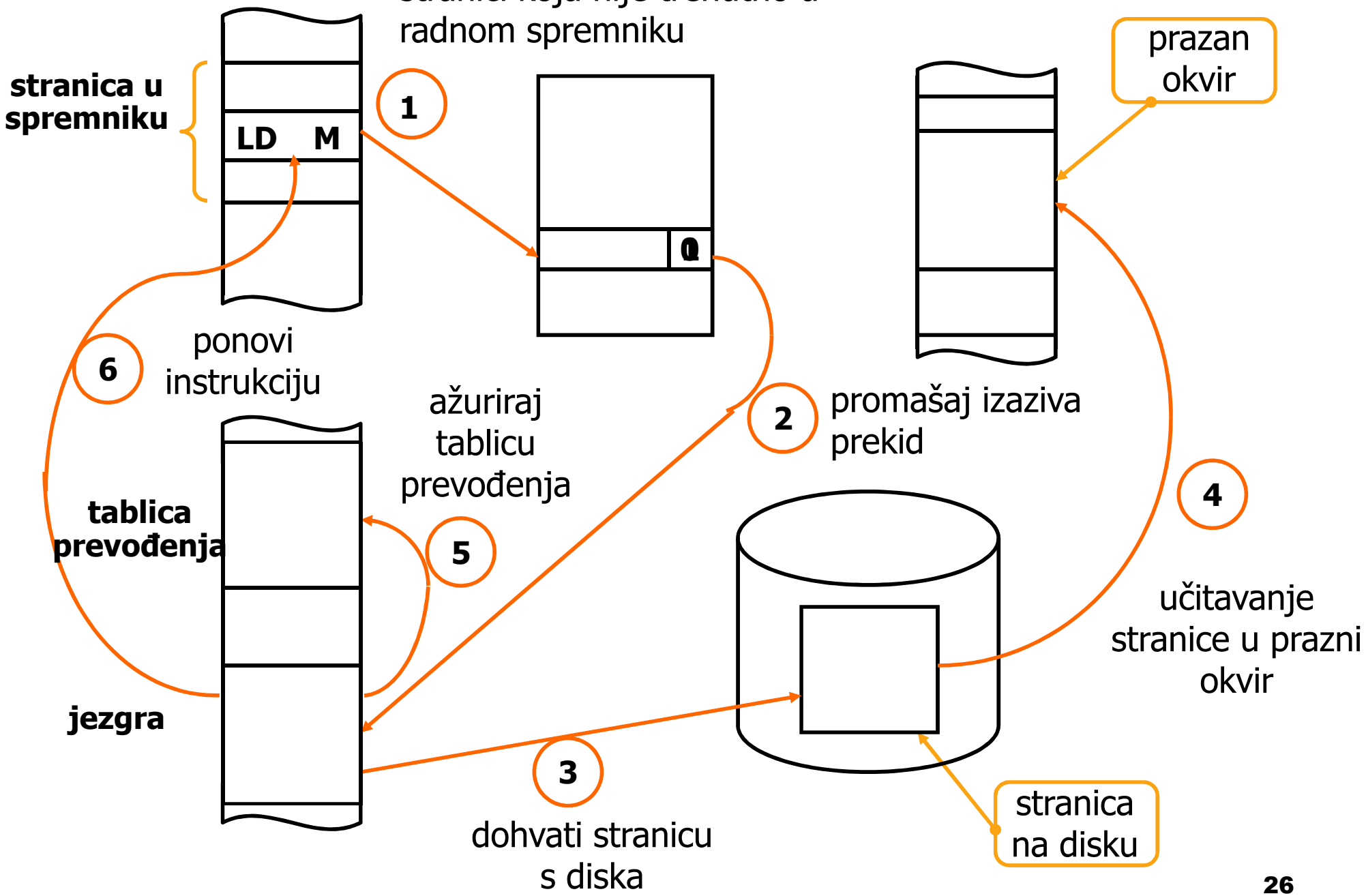
V	bit prisutnosti	A	stranica je korištena
W	zaštita od promjene	D	"prljava" (dirty)
O	za OS	Gl	globalna stranica
W_t	"write through"		

- Osim zapisa u tablici prevođenja, za svaku stranicu je potrebno zapisati i gdje se ona nalazi na pomoćnom spremniku

“Promašaj” stranice

- Kada se tijekom izvođenja programa generira adresa stranice koja se trenutno ne nalazi u radnom spremniku, tada se događa “promašaj”
- Sklopovlje za podršku straničenju će na promašaj reagirati prekidom
  - u obradi prekida (jezgrina funkcija) odrediti će se koja je stranica tražena te će se ona dohvatiti
  - po dohvatu stranice, instrukcija koja je izazvala prekid se mora ponoviti
- Straničenje može značajno smanjiti učinkovitost, ako se promašaju često događaju (dohvat stranice s diska se mjeri u desecima milisekundi)

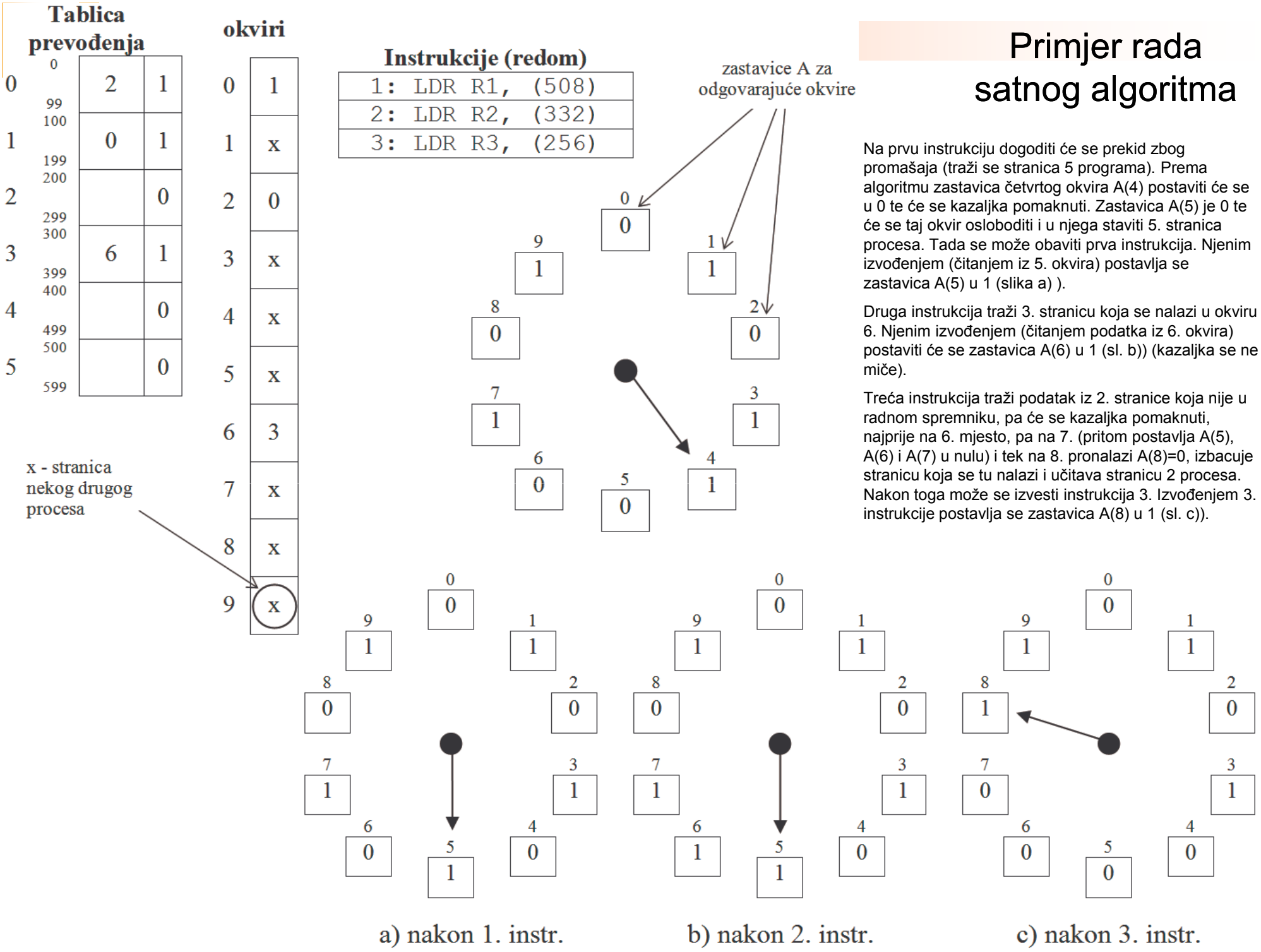
- **Straničenje na zahtijev**
  - stranice se učitavaju tek po promašaju (po zahtjevu)


generirana adresa M pripada stranici koja nije trenutno u radnom spremniku
stranica u spremniku
LD M
1
Q
prazan okvir
6
ponovi instrukciju
ažuriraj tablicu prevođenja
2
promašaj izaziva prekid
tablica prevođenja
5
4
učitavanje stranice u prazni okvir
jezgra
3
dohvati stranicu s diska
stranica na disku

Zamjena stranica

- Kada su svi okviri popunjeni i dogodi se promašaj što napraviti?
  - očito je potrebno izbaciti neku stranicu iz nekog okvira!
  - kako odabrati stranicu za izbacivanje?
- Koristi inačica LRU algoritma (*least recently used*)
  - izbaciti stranicu koja se već duže nije koristila
    - statistički se ta stranica niti neće još neko vrijeme koristiti
  - *satni algoritam* (*clock algorithm*, *second chance algorithm*)
    - okviri, tj. opisnici stranica koje se u njima nalaze, se kružno obilaze - promatra se **zastavica A** koja označava je li stranica korištena (engl. *accessed*)
      - ako je A == 0 stranica se izbacuje iz okvira i u njega se može staviti druga stranica
      - ako je A == 1, postavlja se A = 0 i pomiče se na idući okvir (trenutna stranica ostaje u okviru, daje joj se još jedna prilika obzirom da se nedavno koristila)

Primjer rada satnog algoritma



Na prvu instrukciju dogoditi će se prekid zbog promašaja (traži se stranica 5 programa). Prema algoritmu zastavica četvrtog okvira A(4) postaviti će se u 0 te će se kazaljka pomaknuti. Zastavica A(5) je 0 te će se taj okvir osloboditi i u njega staviti 5. stranica procesa. Tada se može obaviti prva instrukcija. Njenim izvođenjem (čitanjem iz 5. okvira) postavlja se zastavica A(5) u 1 (slika a)).

Druga instrukcija traži 3. stranicu koja se nalazi u okviru 6. Njenim izvođenjem (čitanjem podatka iz 6. okvira) postaviti će se zastavica A(6) u 1 (sl. b)) (kazaljka se ne miče).

Treća instrukcija traži podatak iz 2. stranice koja nije u radnom spremniku, pa će se kazaljka pomaknuti, najprije na 6. mjesto, pa na 7. (pritom postavlja A(5), A(6) i A(7) u nulu) i tek na 8. pronalazi A(8)=0, izbacuje stranicu koja se tu nalazi i učitava stranicu 2 procesa. Nakon toga može se izvesti instrukcija 3. Izvođenjem 3. instrukcije postavlja se zastavica A(8) u 1 (sl. c)).

Straničenje - sažetak

- Potrebna je sklopovska potpora
- Pretvorba adresa (relativne→apsolutne) obavlja se sklopovski (uz podršku OS-a u obradi prekida)
- Prednosti straničenja naspram ostalih metoda:
  - nema fragmentacije
  - zaštita procesa (procesi su odjeljeni, svaki u svom adresnom prostoru)
  - veliki programi (i veći od raspoloživog radnog spremnika) mogu se izvoditi (mehanizmom *straničenja na zahtjev*)
- Nedostaci:
  - zahtijevaju složenu sklopovsku potporu!
  - usporenje rada (kada ima puno “promašaja”)
- Svi današnji operacijski sustavi (opće namjene) podržavaju straničenje
  - izuzeci su RT i ugrađeni sustavi (iako se i tamo sporadično koriste)

Preporuke za programere (i straničenje)

- Teoretski program nije posebno potrebno pripremati
  - upravljanje spremnikom straničenjem je transparentno za program – u potpunosti upravljano sklopovljem i OS-om

- Ipak, saznanje da se straničenje koristi te za njega prikladno oblikovanje programa može značajno povećati učinkovitost sustava
  - "promašaji" su vrlo skupi (vremenski) – treba ih nastojati izbjegavati
  - osnovni princip je jednostavan: slijedno koristiti spremničke lokacije – izbjegavati algoritme koji nasumično pristupaju spremniku:
    - **koristiti princip vremenske i prostorne lokalnosti**
    - isti princip će učinkovitije iskoristiti i sve mehanizme priručnih spremnika, od priručnog spremnika diska do procesora (L1)

Kada se programira za RT

- Koristiti sučelje (API) za zaključavanje pojedinih stranica
- Npr. POSIX sučelje (zapravo samo preporuke OS-u):
  - zaključaj segment:
    - `int mlock (const void * addr, size_t len);`

    `http://www.opengroup.org/onlinepubs/9699919799/functions/mlock.html`
  - zaključaj proces:
    - `int mlockall (int flags);`

    `http://www.opengroup.org/onlinepubs/9699919799/functions/mlockall.html`
- Npr. Win32
  - `VirtualLock (lpAddress, dwSize);`

    `http://msdn.microsoft.com/en-us/library/aa366895(VS.85).aspx`
  - `SetProcessWorkingSetSize (hProcess, Min, Max)`

    `http://msdn.microsoft.com/en-us/library/ms686234(v=VS.85).aspx`