OPERATIVNI SISTEMI

VI - Upravljanje memorijom

VI - Upravljanje memorijom

S A D R Ž A J

6.1 - Uvod

- **memorija** predstavlja niz memorijskih reči od kojih svaka ima jedinstvenu adresu
- prilikom izvršavanja procesa
  - procesor na osnovu vrednosti programskog brojača čita instrukcije iz memorije
  - instrukcije u toku izvršenja mogu zahtevati
    - čitanje podataka sa drugih lokacija
    - upis podataka na druge memorijske lokacije
- razni OS koriste različite metode upravljanja memorijom
  - dok su neke jednostavne, druge su poput straničenja i segmentacije, vrlo komplikovane
  - svaka od njih ima svoje prednosti i nedostatke
  - izbor metode u mnogome zavisi i od postojeće arhitekture procesora

6.1 - Uvod

- ciljevi koje je potrebno postići na nivou sloja upravljanja memorijom su
  - alokacija memorije - dodela memorije procesima
  - razdvajanja fizičkog i logičkog adresnog prostora programa i vezivanje adresa
    - prevođenje relativnih relokatibilnih adresa u fiksne
  - logička organizacija memorije - razdvajanje neizmenljivih segmenata (modula i procedura) od segmenata s promenljivim sadržajem, tj. podacima, čime se postiže
    - nezavisan rad na segmentima koji čine program
    - zaštita segmenata sa malim premašenjem, u vidu listi za kontrolu pristupa segmentima (u višeprocesnom okruženju, procesi ne smeju menjati vrednosti drugih memorijskih lokacija bez dozvole)
    - deljenje segmenata između većeg broja procesa (u slučaju da proces ima dozvolu da izmeni vrednost memorijskih lokacija koje koristi drugi proces, tj. postoje zajedničke lokacije)

6.1 - Uvod

- ciljevi koje je potrebno postići na nivou sloja upravljanja memorijom su *(nastavak)*
  - relokacija, koja obuhvata
    - sažimanje tj. defragmentaciju radne memorije (vezivanje diskontinuelnih memorijskih prostora u jedan kontinuelan nefragmentisan prostor)
    - swap, suspendovanje procesa njegovim smeštanjem na disk
  - podrška za dinamičko punjenje memorije programom i dinamičko vezivanje

6.1 - Uvod

Dodeljivanje memorije

- pored korisničkih procesa u radnu memoriju se smešta i rezidentni deo OS (jezgro)
- da bi u višeprocesnom okruženju obezbedio stabilan i pouzdan rad sistema, neophodno je što efikasnije dodeliti različite delove memorije
- memorija se deli na najmanje dve particije
  - jedna (najčešće niži deo) je namenjena rezidentnom delu OS (*kernel space*)
  - druga (viši delovi) je namenjena korisničkim procesima (*user space*)
- u korisničkom adresnom prostoru se nalazi više procesa koji formiraju red čekanja na procesor
- glavni problem pri upravljanju memorijom jeste dodela slobodne memorije procesima koji su u ulaznom redu (alokacija memorije)

6.1 - Uvod

Dodeljivanje memorije

- tehnike za dodelu memorije procesima grubo se mogu podeliti na dve vrste
  - kontinualna alokacija (*contiguous allocation*)
    - i logički i fizički adresni prostor procesa sastoje se od kontinualnih nizova memorijskih reči, pri čemu memorijske particije po veličini mogu biti jednake ili različite
  - diskontinualna alokacija (*discontiguous allocation*)
    - fizički adresni prostor procesa nije realizovan kao kontinualan niz memorijskih adresa
    - obuhvata metode straničenja, segmentacije i straničenja sa segmentacijom

6.1 - Uvod

Vezivanje adresa

- program se nalazi na disku kao binarna izvršna datoteka (*binary executable*)
- da bi se moagao izvršiti program sa diska se mora učitati u memoriju, unutar adresnog prostora novostvorenog procesa
- proces se u toku izvršavanja može više puta pomerati na relaciji disk – memorija
- kolekcija procesa na disku, koja čeka povratak u memoriju i nastavak izvršenja, naziva se ulazni red (*input queue*)
- kod sistema sa omogućenim multiprogramiranjem, veći broj procesa deli operativnu memoriju računara
  - programer **ne može unapred odrediti fiksne** memorijske lokacije za smeštaj programa
  - zato koristi **simboličke adrese**

6.1 - Uvod

Vezivanje adresa

- simboličke adrese se pretvaraju u apsolutne na sledeći način
  - u izvornom kodu programa imamo na primer **simboličku promenljivu** *count*
  - kompajler prevodi ovu simboličku adresu u **relativnu**, odnosno relokatibilnu
    - promenljiva *count* se npr. vezuje (*bind*) za lokaciju na adresi 14 **u odnosu na početak modula**
  - punilac (*loader*) OS-a prevodi relativne adrese u **fiksne** prilikom učitavanja programa u memoriju
    - punilac pretvara relokatibilnu adresu u apsolutnu adresu (npr. 74014)
  - **vezivanje adresa** (*binding)* je mapiranje iz jednog adresnog prostora u drugi

6.1 - Uvod

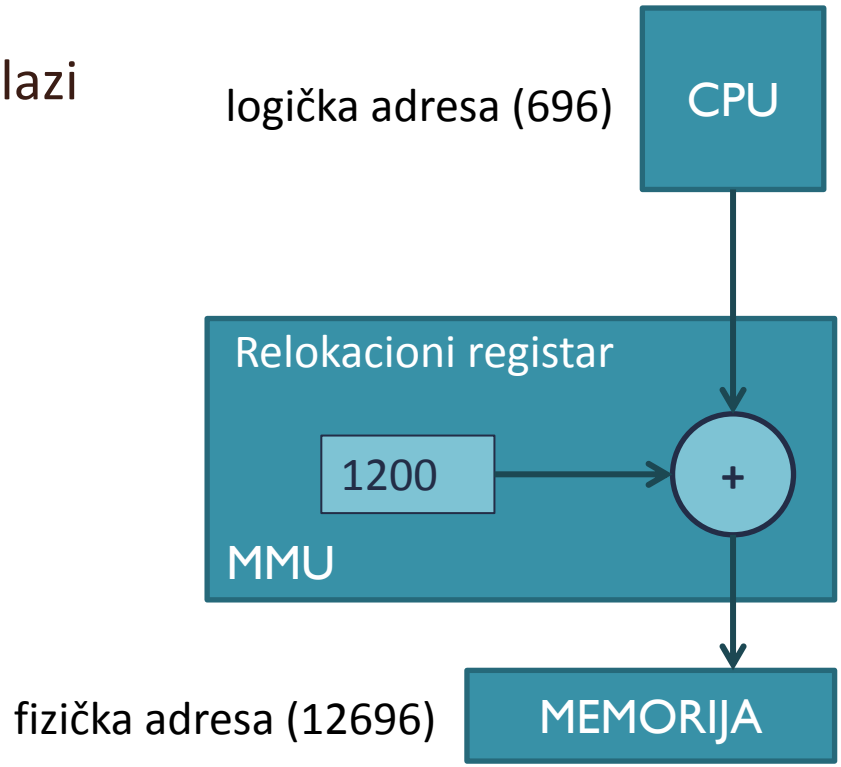
Logički i fizički adresni prostor

- **logička adresa** je adresa koju generiše procesorka instrukcija
  - u fazi izvršavanja programa naziva se **virtuelna adresa**
- **fizička adresa** je adresa same memorijske jedinice
- logičke i fizičke adrese su iste u fazi prevođenja i učitavanja programa ali se razlikuju u fazi njegovog izvršavanja
- **logički (virtualni) adresni prostor** je skup svih logičkih adresa koje generiše program
- **fizički adresni prostor** je skup svih fizičkih adresa koje njima odgovaraju
- jedinica za upravljanje memorijom - **MMU** (engl. *Memory Management Unit)* je hardverski uređaj koji mapira virtuelni adresni prostor u fizički

6.1 - Uvod

Logički i fizički adresni prostor

- u nastavku je prikazana najprostija MMU šema koja koristi **relokacioni registar**
  - relokacioni registar definiše adresu **fizičkog početka programa**
  - fizička adresa = logička adresa koju generiše program + vrednost relokacionog registra
  - logički adresni prostor koji se nalazi u opsegu [0, max] se mapira u opseg [R+0, R+max]
  - R je vrednost relokacionog registra (fizička adresa početka programa)
  - vrednost max je gornja granica



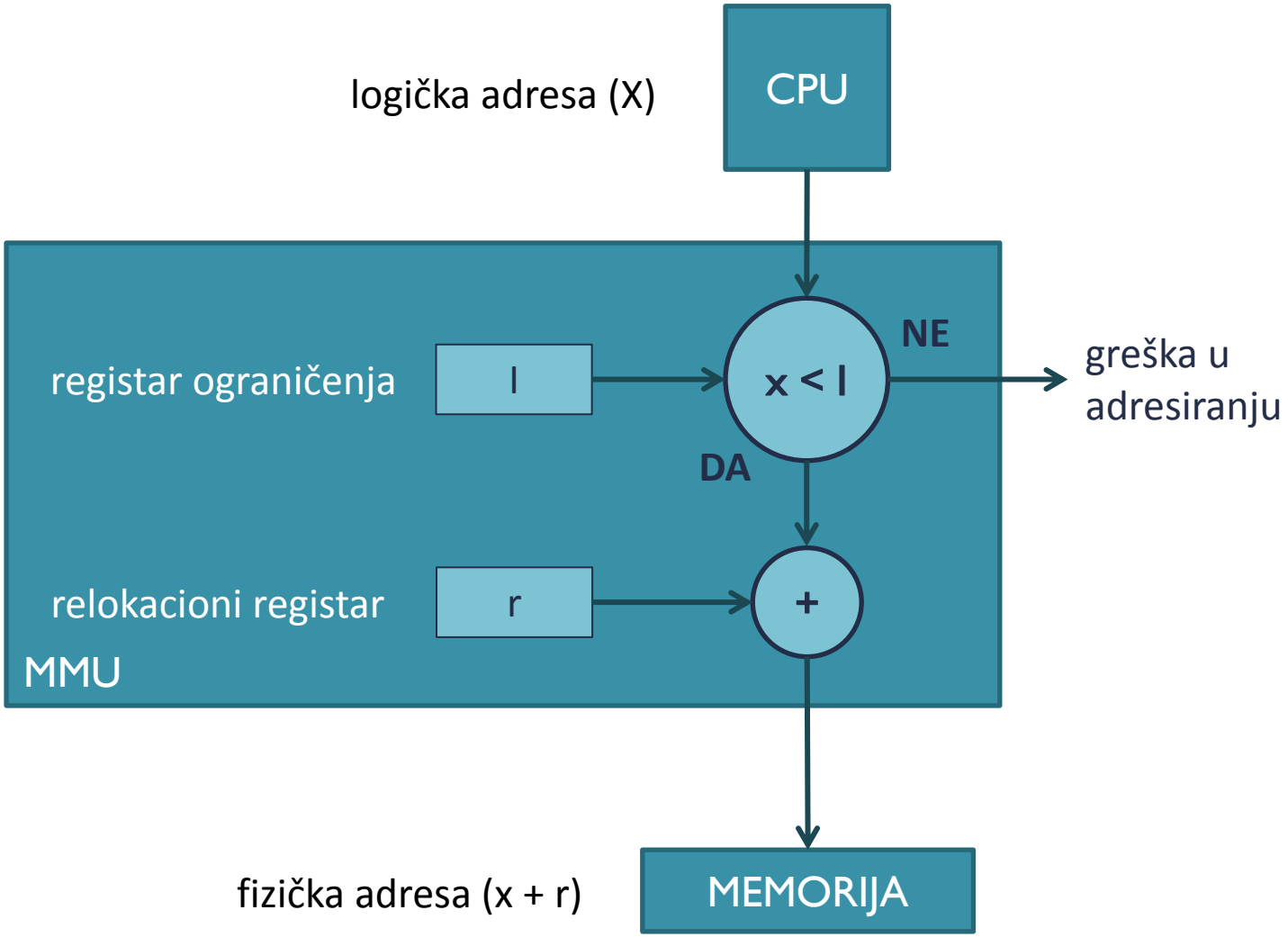
6.1 - Uvod

Zaštita memorije

- zaštita memorije obuhvata
  - zaštitu OS od korisničkih procesa
  - međusobnu zaštitu korisničkih procesa po pitanju pristupa memorijskim sekcijama
- može se realizivati pomoću dva registra
  - **relokacioni registar** - sadrži najnižu adresu procesa
  - **registar ograničenja** - sadrži najveći opseg logičkih adresa procesa
- MMU mapira svaku logičku adresu procesa dinamički
  - proverava da li je logička adresa manja od vrednosti limit registra
  - ako jeste, dodaje vrednost relokacionog registra
- relokacioni i limit registar su dva registra procesora
  - pune se onda kada proces dobija procesor na izvršenje
  - dispečer čita vrednosti ova dva registra iz konteksta procesa i postavlja ih prilikom aktiviranja procesa

6.1 - Uvod

Zaštita memorije



6.1 - Uvod

Razmena (*swap)*

- proces se prilikom izvršavanja mora nalaziti u operativnoj memoriji
- proces se može **privremeno prebaciti iz memorije na disk,** kako bi se oslobodila memorija
  - oslobođena memorija puni se drugim procesom
  - posle izvesnog vremena, proces se može **vratiti sa diska u memoriju** kako bi nastavio izvršavanje
- koristi se u **prioritetnim šemama za raspoređivanje procesa**
  - procesi visokog prioriteta se čuvaju u memoriji
  - procesi niskog prioriteta se upisuju na disk i čekaju da se oslobodi memorija
- ova varijanta razmenjivanja naziva se *roll out, roll in*
- proces koji se razmenjuje mora biti potpuno oslobođen aktivnosti, ne sme biti u stanjima READY ili WAIT

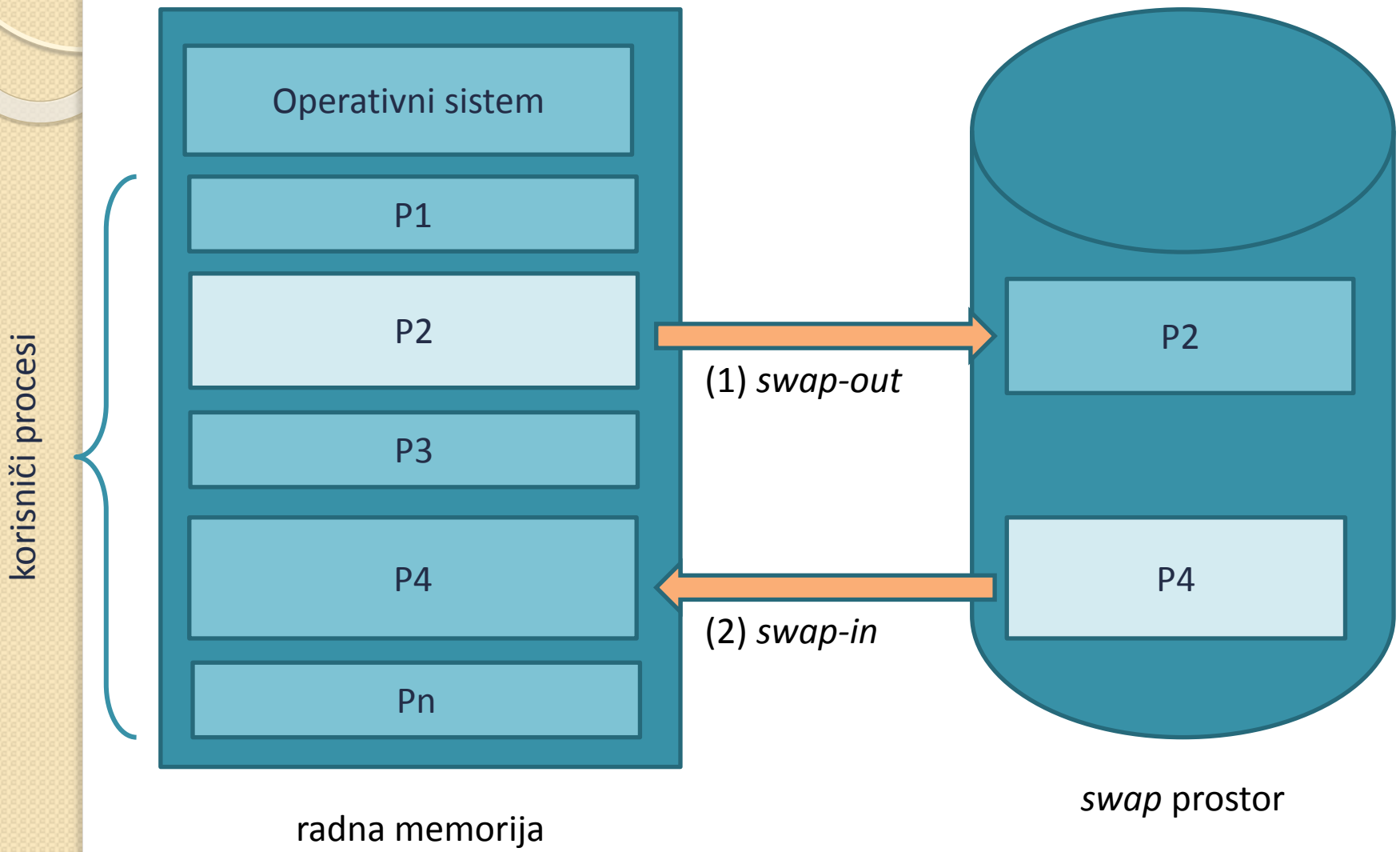
6.1 - Uvod

Razmena (*swap)*

- tehnika razmene zahteva postojanje 3 komponente
  - prostor na disku (*swap space*) na koji će se smeštati uspavani procesi
  - mehanizam s*wap-out* koji prebacuje proces iz memorije na disk
  - mehanizam *swap-in* koji vraća uspavani proces sa diska u memoriju
- najveći deo vremena u razmeni otpada na transfer podataka između memorije i diska
  - trajanje jedne razmene zavisi od
    - količine podataka za transfer
    - tipa i karakteristika sekundarnih memorija i pratećeg hardvera
- to vreme je znatno veće u odnosu na performanse savremenih procesora
- zbog toga se ne preporučuje često korišćenje tehnike razmenjivanja

6.1 - Uvod

Razmena (*swap*)



6.2 - Programerske tehnike upravljanja memorijom

- prilikom projektovanja programa koji zbog veličine ne mogu stati celi u memoriju, programeri koriste odgovarajuće tehnike kojima se postiže prividno povećanje i bolje iskorišćenje memorijskog prostora

Dinamičko punjenje memorije programom

- suština **dinamičkog punjenja** (*dynamic loading)* je u sledećem
  - u memoriju se smeštaju samo neophodni delovi programa
  - odgovarajuće rutine se učitavaju u memoriju samo kada ih program pozove
- da bi to bilo moguće, sve rutine programa se čuvaju na disku u **relokatibilnom formatu**
- kada se rutina pozove iz programa
  - proverava se da li je ona već u memoriji
  - ako nije, poziva se punilac da je učita u memoriju

6.2 - Programerske tehnike upravljanja memorijom

Tehnika preklapanja (*overlay*)

- **primer**
- program čine dve relativno nezavisne komponente koje se izvršavaju jedna za drugom
- programer može da podeliti izvorni kôd programa na dva dela
- veličine programskih komponenti
  - prvi deo programa (80kb)
  - drugi deo programa (70kb)
  - zajednički podaci (20kb),
  - zajedničke rutine (30kb)
- za izvršavanje programa potrebno je **najmanje 200kb slobodne memorije**
  - to znači da program ne može da se izvršava na hipotetičkom sistemu sa 150kb memorije

6.2 - Programerske tehnike upravljanja memorijom

Tehnika preklapanja (*overlay*)

- **primer** (*nastavak*)
- pošto su komponente nezavisne
  - prilikom izvršavanja prvog dela programa, kôd drugog dela ne mora biti učitan u memoriju
  - drugi deo programa se može izvršavati ukoliko prvi deo nije prisutan u memoriji
- možemo da definišemo dve *overlay* komponente
  - **overlay A**: zajednički podaci i rutine + kôd za prvi deo programa
    - ukupno 130KB
  - **overlay B**: zajednički podaci i rutine + kôd za drugi deo programa
    - ukupno 120KB
- programu se mora dodati i *overlay* drajver koji upravlja *overlay* tehnikom
  - pretpostavimo da on zauzima 10KB memorije

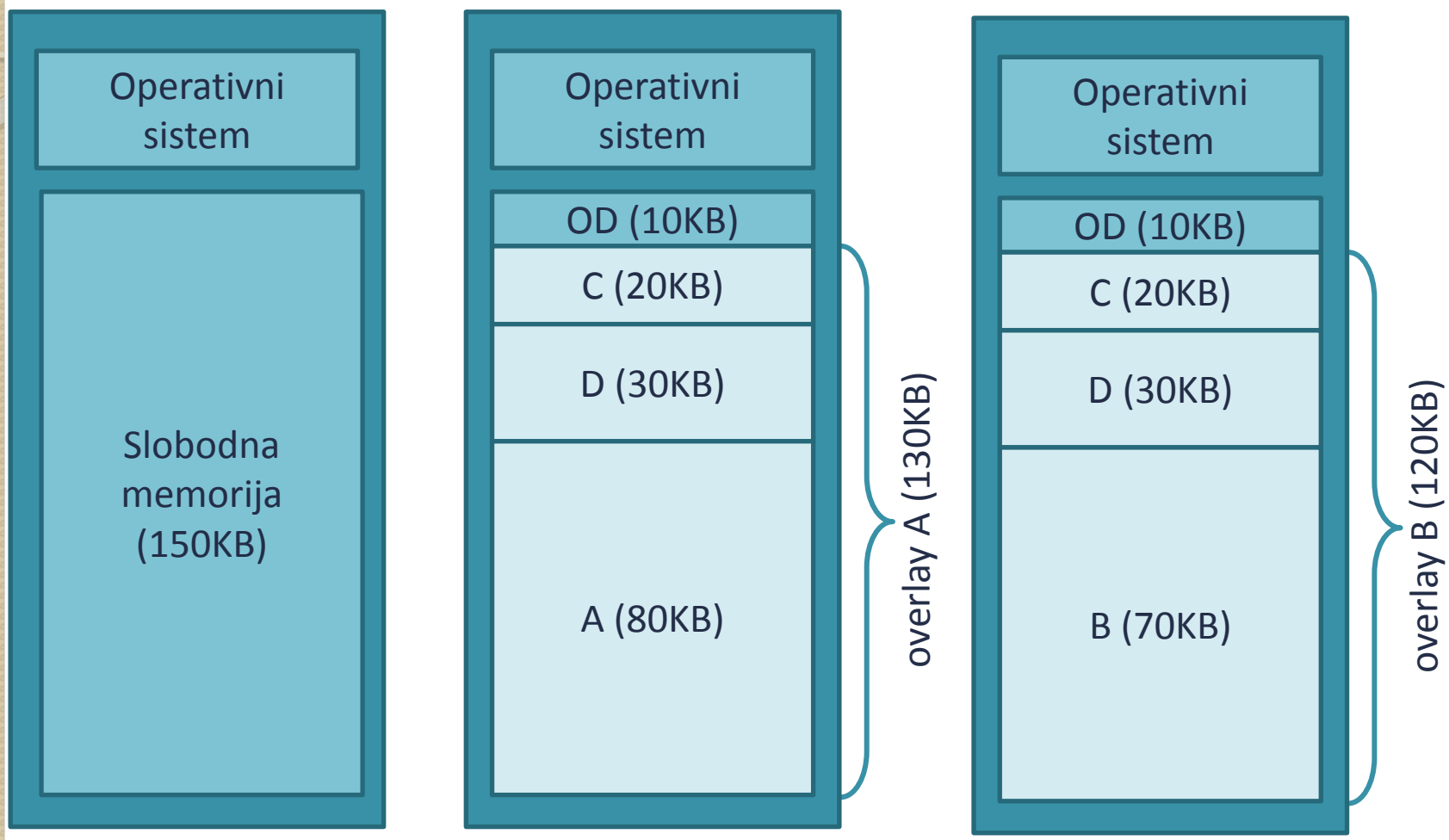
6.2 - Programerske tehnike upravljanja memorijom

Tehnika preklapanja (*overlay*)

- **primer** (*nastavak*)
- program se izvršavana sledeći način
  - prva faza: *overlay* drajver + *overlay* A.
    - zauzima 140KB i može da se izvrši na sistemu sa 150KB
  - kada *overlay* A završi svoje, drajver će učitati u memoriju *overlay* B preko njega
  - druga faza: *overlay* drajver + *overlay* b
    - zauzima 130KB i može da se izvrši na sistemu sa 150KB memorije
- tehnika preklapanja usporava rad samog programa, jer se delovi programa učitavaju u memoriju u više iteracija
- *overlay* drajveri su uključeni u neke programske jezike (Borland Pascal, Clipper), tako da primena tehnike preklapanja ne zahteva posebnu podršku OS
- konkretnu strukturu *overlay* delova definiše programer jer najbolje poznaje svoj program

6.2 - Programerske tehnike upravljanja memorijom

Tehnika preklapanja (*overlay*)



A - prvi deo programa
B - drugi deo programa

C - zajednički podaci
B - zajedničke rutine

OD - overlay drajver

6.3 - Kontinuelno dodeljivanje memorije

- postoje dva osnovna načina da se izvrši dodeljivanje (alokacija) memorije
- **kontinualno dodeljivanje memorije**
  - logički i fizički adresni prostor procesa sastoje se od kontinualnog niza memorijskih adresa
  - svaki proces dobija jedan kontinualni deo memorije
  - metode
    - multiprogramiranje sa fiksnim particijama
    - multiprogramiranje sa particijama promenljive veličine
- **diskontinualno dodeljivanje memorije**
  - fizički adresni prostor procesa nije realizovan kao kontinualan niz memorijskih adresa
  - metode
    - straničenje
    - segmentacija
    - straničenje sa segmentacijom

6.3 - Kontinuelno dodeljivanje memorije

Multiprogramiranje sa fiksnim participama

- podrazumeva podelu cele fizičke memorije na više delova **fiksne veličine**
  - u jednom delu se može naći samo jedan proces
  - stepen multiprogramiranja je jednak broju memorijskih particija
- novi proces se smešta **u najmanju moguću particiju**
- problem može nastati u vidu **interne fragmentacije**
  - delovi memorije koji su veći od veličine procesa potpuno neiskorišćeni
- svi procesi se stavljaju u **red čekanja** (*input queue)* koji može biti
  - jedinstven za ceo sistem
    - ovde je problem veliki neiskorišćen prostor (mali procesi u velikim participama)
  - poseban za svaku particiju
    - problem: veći broj malih procesa može čekati u redu za male particije
    - velike particije ostaju **neiskorišćene**
    - u tom slučaju ima dovoljno memorije, ali se ne koristi

6.3 - Kontinuelno dodeljivanje memorije

Multiprogramiranje sa participama promenljive veličine

- memorija se deli dinamički
- **šupljina** *(hole)* je slobodan kontinualni deo memorije
  - šupljine raznih veličina su razbacane po memoriji
  - kada proces naiđe u sistem traži se šupljina dovoljno velika da zadovolji proces
  - sav prostor koji proces ne zauzme od cele šupljine, predstavlja novu šupljinu u koju se može smestiti drugi proces
- operativni sistem vodi evidenciju o
  - participama koje su dodeljene procesima
  - slobodnim šupljinama
- metode vođenja evidencije
  - bit mape
  - povezane liste
  - sistem udruženih parova

6.3 - Kontinuelno dodeljivanje memorije

Multiprogramiranje sa particijama promenljive veličine

- **Bit mape**
- memorija se deli na delove **iste veličine**
- svakom delu dodeljujemo po jedan bit
  - 1 označava da je taj deo zauzet
  - 0 da je slobodan
  - taj niz nula i jedinica predstavlja **bit mapu** (*bitmap)* **memorije**
- pitanje: koje veličine treba da budu ovi delovi memorije?
  - ako su delovi manji, bit mapa je veća
    - veličina delova 32 bita → za bit mapu koristimo 1/33 ukupne memorije
    - veličina delova 16 bita → za bit mapu koristimo 1/17 od ukupne memorije
    - problem je što je pretraživanje slobodne memorije (niz nula) sporo
  - ako su delovi veći, memorija je loše iskorišćena
    - uzmimo za primer delove veličine 4KB
    - ako proces zauzme 1KB od nekog dela, 3KB ostaje neiskorišćeno
    - 3KB ne može biti alocirano od strane drugog procesa jer je formalno zauzeto

6.3 - Kontinuelno dodeljivanje memorije

Multiprogramiranje sa participama promenljive veličine

- **Povezane liste** (*linked lists)*
- grade se od slogova sledeće strukture
  - **tip memorije**
    - P znači da se radi o procesu
    - H da se radi o slobodnoj memoriji
  - **početna adresa** dela memorije koju opisuje dati slog
  - **dužina** opisane memorije
  - **pokazivač** na sledeći slog
- upravljanje memorijom se odvija na sledeći način
  - prilikom **zauzimanja memorije** u povezanoj listi se traži rupa (slog tipa H) dovoljne veličine
    - ako se nađe rupa odgovarajuće veličine, umesto H se upisuje P
    - ako je rupa veća ubacuje se i novi čvor tipa H
  - kada proces završi sa radom, **oslobođa se zauzeta memorija**

6.3 - Kontinuelno dodeljivanje memorije

Multiprogramiranje sa particijama promenljive veličine

- **Sistem udruženih parova** (buddy system)
- koristi se po jedna lista za blokove slobodne memorije veličine 2^n bajtova (1, 2, 4, ...)
- **primer**
  - imamo 1MB memorije
  - treba nam 21 lista (2^0=1, 2^2= 2 ... 2^{20}=1MB)
- na početku rada, cela memorija je prazna
  - u listi za rupe veličine 1MB imamo jedan slog
  - ostale liste su prazne
- proces A veličine 70KB nailazi u sistem
  - može se smestiti u particiju veličine najmanje 128KB (mora biti stepen od 2)
  - lista koja sadrži particije te veličine je **prazna**

6.3 - Kontinuelno dodeljivanje memorije

Multiprogramiranje sa particijama promenljive veličine

- **Sistem udruženih parova** (buddy system)
- **primer** (*nastavak*)
  - particija veličine 1MB deli se na dva dela od 512KB
    - ovi delovi se zovu **drugovi** (*buddy)*
  - zatim se prvi deo deli na dva dela od 256KB, pa još jednom, na dva dela od 128KB
  - u prvu particiju od 128KB **smešta se proces**
- nakon smeštanja procesa lista od 128Kb sadrži dva čvora
  - jedan je zauzet - P (128KB) {proces A:70KB, neiskorišćeno:58KB}
  - drugi je slobodan - H (128KB)
- liste od 256KB i 512KB sadrže po jedan slobodan čvor
  - H (256KB) i H (512KB)
- nedostatak ovog sistema je **unutrašnja fragmentacija**
  - proces veličine 70KB zauzima čitavu particiju od 128KB
  - 58KB memorije ostaje neiskorišćeno (unutrašnja fragmentacija)

6.3 - Kontinuelno dodeljivanje memorije

Multiprogramiranje sa particijama promenljive veličine

- **Sistem udruženih parova** (buddy system)
- **primer** (*nastavak*)
- proces B veličine 35KB nailazi u sistem
  - može se smestiti u particiju veličine 64KB
  - lista za takve particije je prazna
  - slobodna particija od 128KB deli na dva dela veličine 64KB
  - proces B se smešta u jednu particiju, dok druga ostaje prazna
- sada imamo
  - po jedan čvor u listama za particije veličine 128KB, 256KB i 512KB
  - dva čvora u listi za particije veličine 64KB
    - P (128KB) {proces A:70KB, neiskorišćeno:58KB}
    - P (64KB) {proces B:35KB, neiskorišćeno:29KB}
    - H (64KB)
    - H (256KB)
    - H (512KB)

6.3 - Kontinuelno dodeljivanje memorije

Multiprogramiranje sa particijama promenljive veličine

- **Sistem udruženih parova** (buddy system)
- **primer** (*nastavak*)
- proces C veličine 80KB nailazi u sistem
- može se smestiti u particiju veličine 128KB
- lista za takve particije je prazna
- slobodna particija od 256KB deli na dva dela veličine 128KB
- proces C se smešta u jednu particiju, dok druga ostaje prazna
- sada imamo
  - P (128KB) {proces A:70KB, neiskorišćeno:58KB}
  - P (64KB) {proces B:35KB, neiskorišćeno:29KB}
  - H (64KB)
  - P (128KB) {proces C:80KB, neiskorišćeno:48KB}
  - H (128KB)
  - H (512KB)

6.3 - Kontinuelno dodeljivanje memorije

Multiprogramiranje sa particijama promenljive veličine

- **Sistem udruženih parova** (buddy system)
- **primer** (*nastavak*)
- stanje memorije nakon dodele memorije procesima A, B i C
  - tri procesa ukupne veličine 70+35+80=185KB
  - zauzimaju 128+64+128=320KB memorije
  - 135KB ostaje neiskorišćeno kao posledica unutrašnje fragmentacije
- **susedne rupe iste veličine** predstavljaju drugove
  - susedni procesi to nisu
- kada neki proces završi rad, particija u kojoj se proces nalazio postaje rupa
  - rupa se spaja sa susedom (svojim drugom ukoliko on postoji)
  - time se formira duplo veća rupa
  - veličina novoformirane rupe mora da bude stepen broja dva
    - ukoliko to nije slučaj, spajanje se ne vrši

6.3 - Kontinuelno dodeljivanje memorije

Algoritmi za izbor slobodne particije

- po pravilu, postoji više šupljina koje se mogu iskoristiti za smeštaj procesa, a u tom kontekstu postoji i više algoritama za alokaciju šupljina
- **prvo podudaranje** (*first-fit)*
  - pretraživanje se odvija ili od početka liste ili se nastavlja prethodno pretraživanje (*next-fit*)
  - pretraga staje kada se nađe prva odgovarajuća šupljina bez obzira što ima i boljih rešenja
  - vreme pretraživanja je minimalno, jer se ne pretražuje cela lista
- **najbolje podudaranje** (*best-fit)*
  - procesu se **dodeljuje najmanja šupljina** koja je dovoljno velika za smeštaj procesa
  - pretražuje se cela lista **(intenzivna pretraga)** osim ako nije uređena po veličini
  - od postojeće šupljine nakon smeštaja procesa **ostaje najmanja moguća šupljina**

6.3 - Kontinuelno dodeljivanje memorije

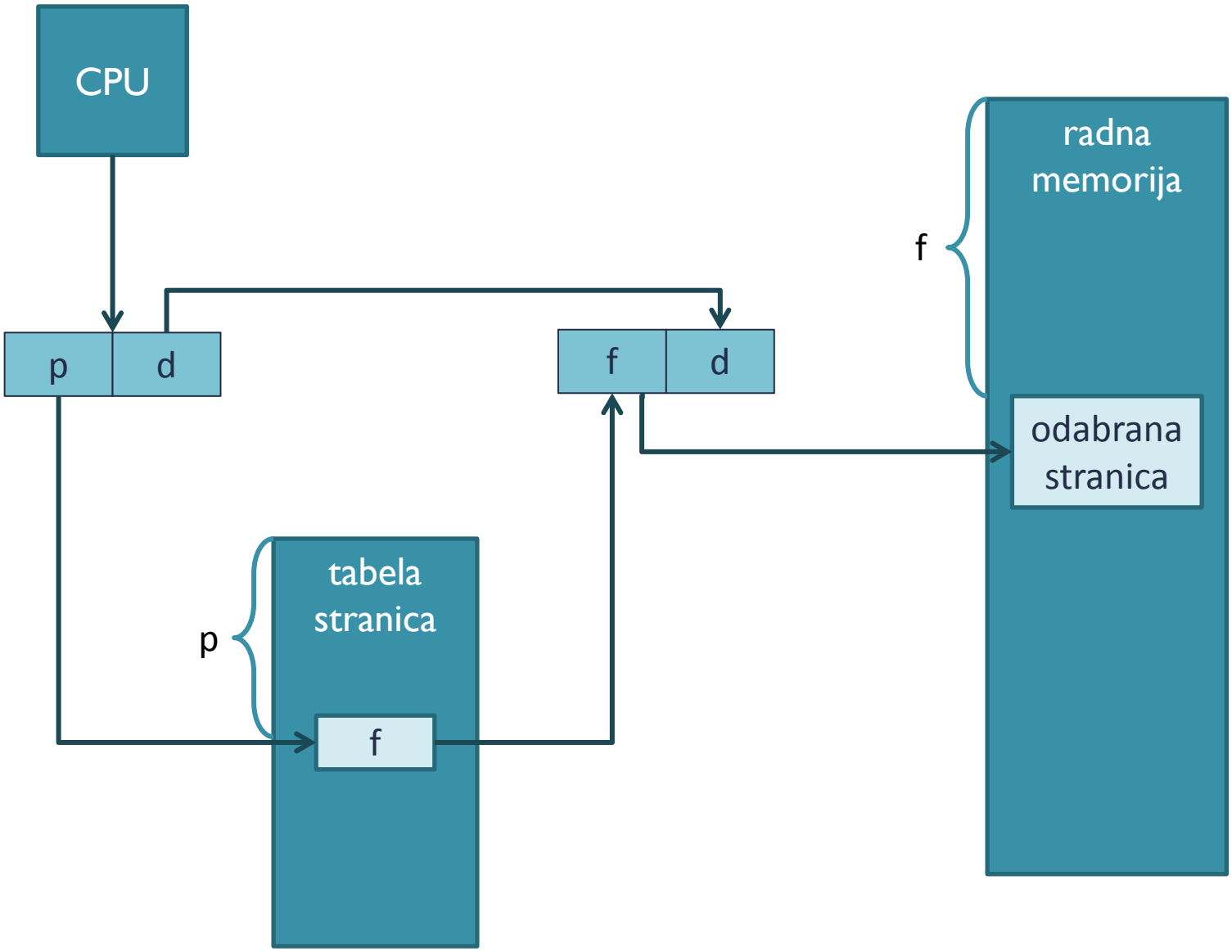
Fragmentacija memorije

- prilikom dodele memorije javljaju se dve vrste fragmentacije
- **unutrašnja (interna) fragmentacija**
  - procesu je dodeljena memorijska particija veća od memorije koju zahteva proces
  - preostala memorija je neupotrebljiva
- **spoljašnja (eksterna) fragmentacija**
  - ukupan memorijski prostor može da zadovolji zahtev procesa ali **nije kontinualan**
- eksterna fragmentacija se može smanjiti **sažimanjem memorije**
  - sadržaj memorije se ispremešta
  - dobijaju se veće šupljine, odnosno kontinualan prazan prostor
  - problemi sažimanja memorije
    - sistem mora da prekida procese i da ih privremeno prebacuje na disk
    - to je vremenski zahtevno
    - dovodi do degradacije performansi sistema

6.4 - Straničenje

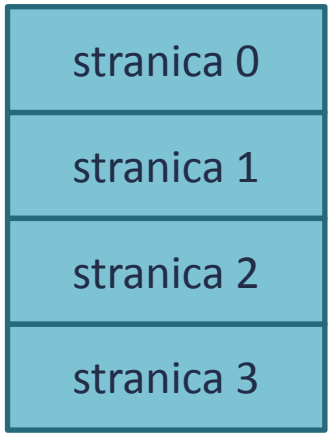
- straničenje je metoda sa **hardverskom podrškom na nivou procesora**
- fizička memorija se deli na blokove fiksne veličine: **okvire** (*page frames)*
- logički adresni prostor se deli na blokove istih veličina: **stranice** (*pages)*
  - veličine stranica su po pravilu stepen broja 2
- svakoj logičkoj stranici odgovara jedna fizička - f
- korespodencija između njih se čuva u **tabeli stranica** (*page table)*
  - kontinualni logički prostor procesa može da se razbaca po fizičkoj memoriji
- svaka logička adresa koju generiše procesor se deli na dva dela
  - **broj stranice** (*page number)* - p
    - koristi se kao indeks u tabeli stranica koja sadrži baznu adresu okvira
    - bazna adresa predstavlja viši deo adrese
  - **pomeraj unutar stranice** (engl *page offset)* **-** d
    - definiše položaj u odnosu na samu stranicu

6.4 - Straničenje



6.4 - Straničenje

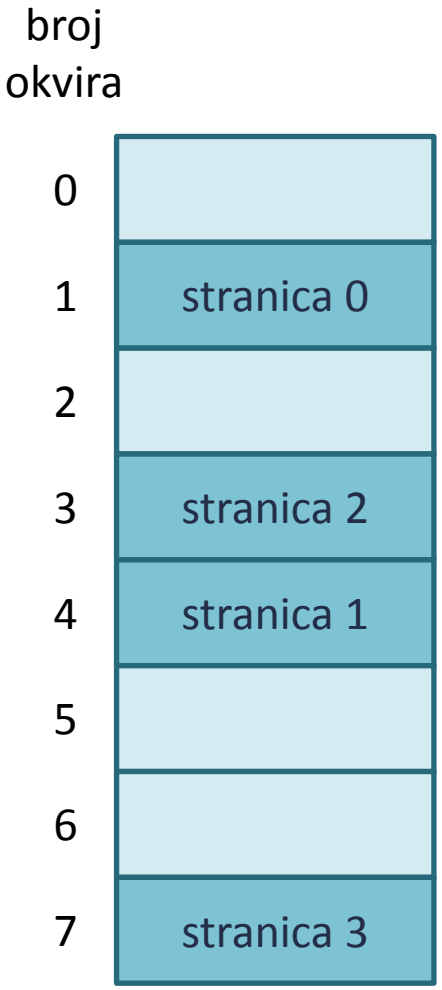
- **primer**
  - imamo memoriju veličine 32B
  - definišimo 8 okvira veličine 4B
  - korisnički proces zauzima 4 logičke stranice
    - stranica 0 - logičke adrese 0-3
    - stranica 1 - logičke adrese 4-7
    - stranica 2 - logičke adrese 8-11
    - stranica 3 - logičke adrese 12-15



logički adresni prostor

p	f
0	1
1	4
2	3
3	7

tabela stranica



logički adresni prostor

6.4 - Straničenje

p	d	adresa	vrednosti
0	0	0	a
	1	1	b
	2	2	c
	3	3	d
1	0	4	e
	1	5	f
	2	6	g
	3	7	h
2	0	8	i
	1	9	j
	2	10	k
	3	11	l
3	0	12	m
	1	13	n
	2	14	o
	3	15	p

logički adresni prostor

p	f
0	5
1	6
2	1
3	2

tabela stranica

mapiranje logičke adrese 5

- log. adresa 5 je logička stranica (p)1 sa pomerajem (d)1 sa vrednošću f
- na osnovu tabele stranica, to je okvir (f) 6 sa pomerajem (d) 1
- fizička adresa je prema tome (6*4)+1=25, gde je
  6 - redni broj okvira (f)
  4 - broj stavki u jednoj stranici

f	d	adr.	vred.
0	0	0	
	1	1	
	2	2	
	3	3	
1	0	4	i
	1	5	j
	2	6	k
	3	7	l
2	0	8	m
	1	9	n
	2	10	o
	3	11	p
3	0	12	
	1	13	
	2	14	
	3	15	
4	0	16	
	1	17	
	2	18	
	3	19	
5	0	20	a
	1	21	b
	2	22	c
	3	23	d
6	0	24	e
	1	25	f
	2	26	g
	3	27	h
7	0	28	
	1	29	
	2	30	
	3	31	

fizički adresni prostor

6.4 - Straničenje

Zaštita memorije

- zaštita memorije postiže se uvođenjem novih bitova koji imaju specijalno značenje
- ovi bitovi se upisuju u dodatne opcione kolone u tabeli stranica za svaku stranicu koja se u njoj nalazi
- **bit validnosti**
  - **"v"** ukazuje da je vrednost u tabeli stranica važeća, ispravna i da se okvir može koristiti za mapiranje
  - **"i"** ukazuje da te stranice su u tabeli proglašene kao nevažeće (*invalid bit)*
    - to se ponekad koristi da se nekom programu zabrani pristup određenim logičkim adresama
- **bit za kotrolu upisa**
  - **"rw"** određuje da je stranica sa punim pristupom
  - **"ro"** označava da se stranica može samo čitati

6.5 - Segmentacija

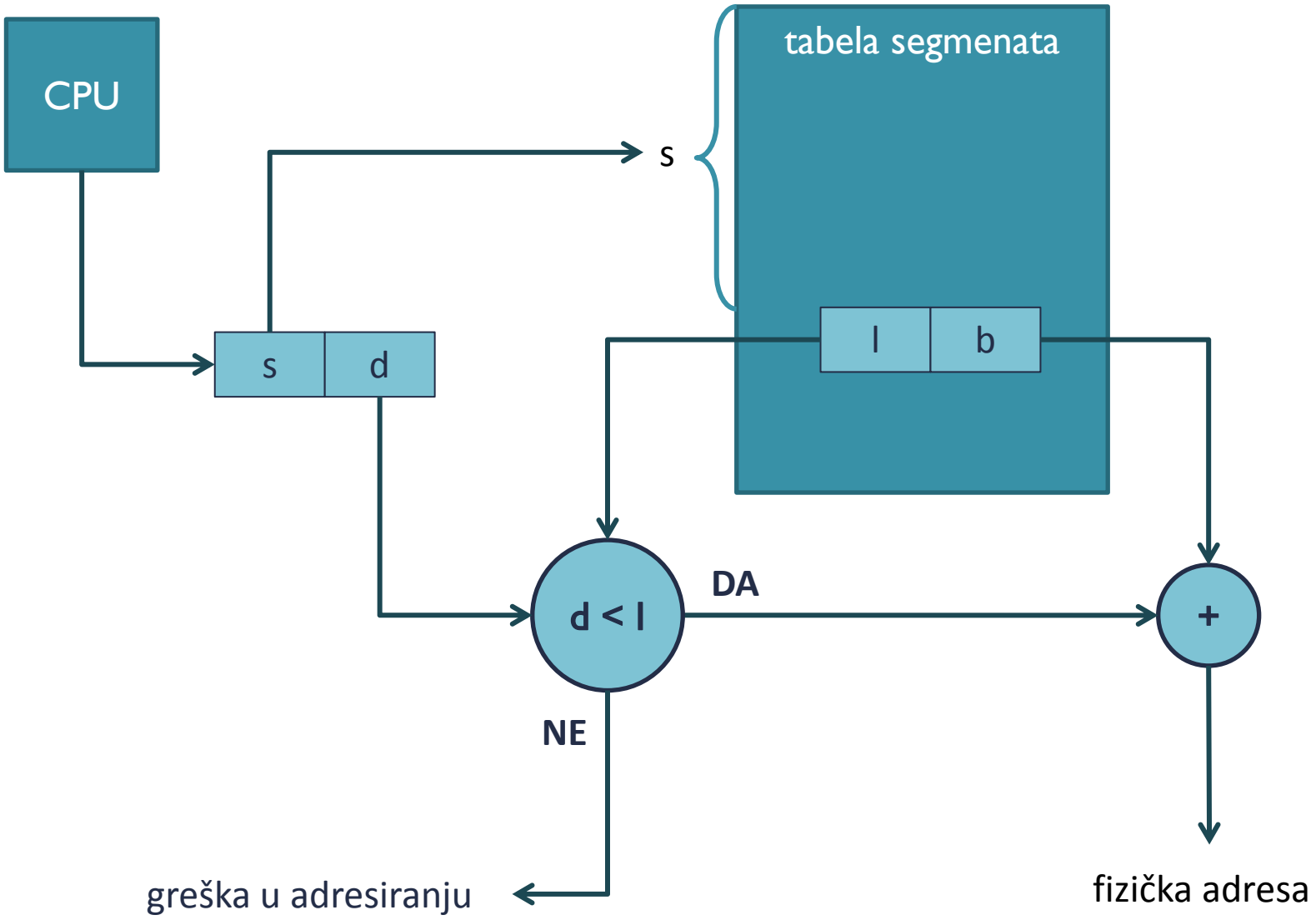
- logički adresni prostor se sastoji od kolekcije segmenata
- svaki segment ima jedinstveno ime i dužinu
- logička adresa se se sastoji od dva dela
  - **imena segmenta** (umesto imena obično se zadaje broj koji predstavlja ID segmenta)
  - **pomeraja unutar segmenta**
- kod segmenatacije je prisutna **eksterna fragmentacija**
- slobodna memorija se ne može iskoristiti za smeštaj segmenta ukoliko ne postoji dovoljno velika šupljina
- problem se može umanjiti **sažimanjem memorije**

6.5 - Segmentacija

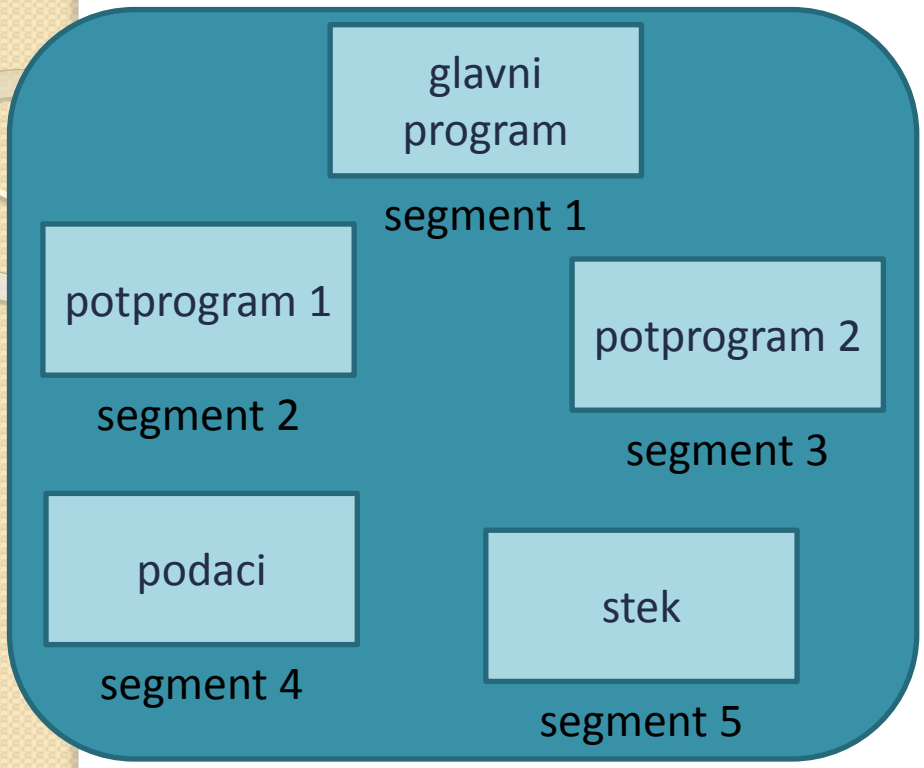
Hardverska podrška

- kod metode segmentacije korisničke adrese su dvodimezionalne
- mapiranje u fizičke adrese se obavlja preko **tabele segmenata**
- svaki ulaz u tabeli segmenata sadrži dva parametra
  - **baznu adresu segmenta** (početna fizičku adresu segmenta u memoriji)
  - **limit (ograničenje) segmenta** (dužina segmenta)
- mapiranje logičke adrese u fizičku se vrši na sledeći način
  - **broj segmenta** (s) se koristi kao **ulaz u tabelu segmenta**
  - iz tabele se čitaju **bazna adresa** segmenta (b) i **limit** (l)
  - ako je pomeraj veći od limita (d > l)
    - imamo grešku u adresiranju
  - ako je pomeraj manji od limita (d < l):
    - fizička adresa = bazna adresa + pomeraj

6.5 - Segmentacija



6.5 - Segmentacija

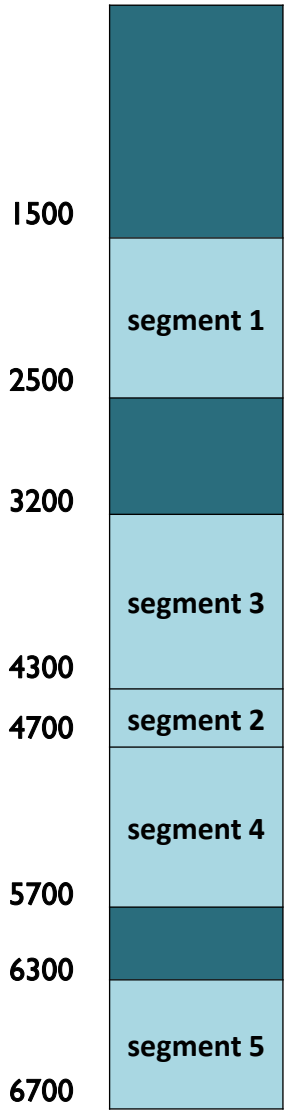


Segment	Bazna adresa	Ograničenje
1	1500	1000
2	4300	400
3	3200	1100
4	4700	1000
5	6300	400

tabela segmenata

Primer

segmentacija 5 segmenata čije su definicije date u tabeli segmenata



fizički adresni prostor