

DISTRIBUIRANI ALGORITMI I SISTEMI

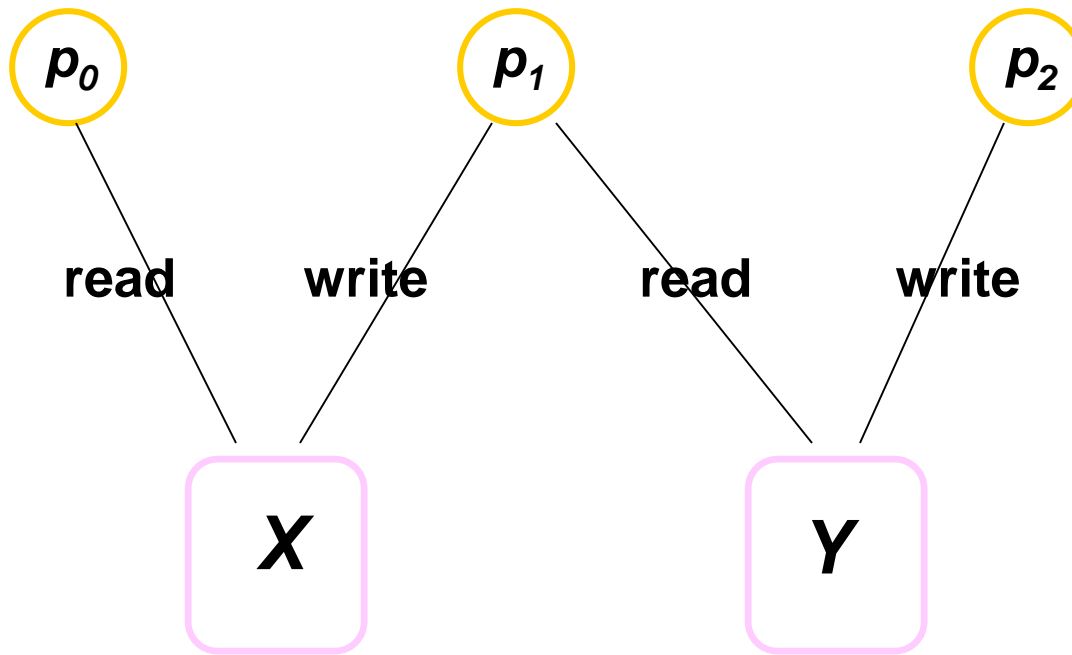
Model deljenja memorije (SM = Shared Memory)

2

- Procesori komuniciraju preko skupa deljenih promenljivih, umesto slanjem poruka.
- Svaka deljena promenljiva ima **tip**, koji definiše skup operacija koje se mogu izvoditi *atomski*.

Primer modela deljene memorije

3



Model deljene memorije

4

- Razlike u odnosu na model sa slanjem poruka:
 - nema *inbuf* i *outbuf* komponenti stanja
 - konfiguracija uključuje vrednosti za sve deljene promenljive
 - Jedini tip događaja je korak procesorskog računanja
 - Izvršenje je **prihvatljivo** ako svaki procesor izvodi beskonačan broj koraka

Korak računanja u modelu deljenje memorije

5

- Kada procesor p_i izvodi korak:
 - ▣ stanje p_i u staroj konfiguraciji određuje kojim deljenim promenljivama treba pristupiti i sa kojim operacijama
 - ▣ operacija se obavi: vrednost deljene prom. u novoj konfiguraciji se menja u skladu sa semantikom operacije
 - ▣ stanje p_i u novoj konfiguraciji se menja na osnovu starog stanja i rezultata operacije

Opservacije na modelu deljene memorije

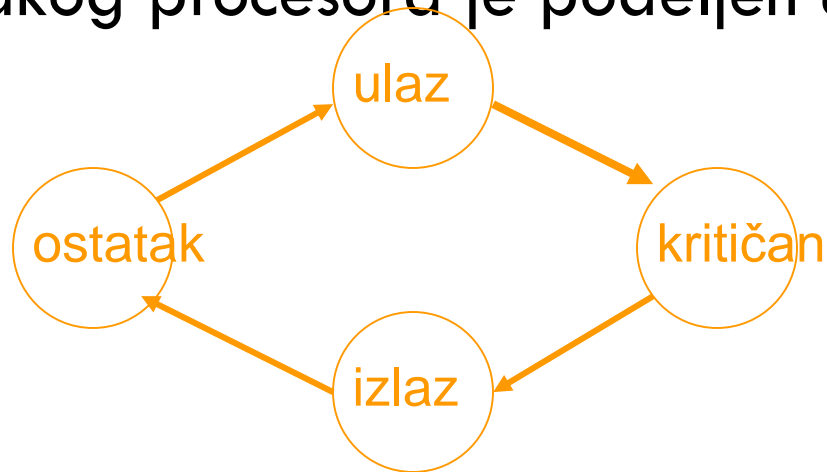
6

- Pristupi deljenim promenljivama tokom koraka računanja modeluju se kao ternutni (atomski), jedan pristup po koraku
- Definicija prihvatljivog izvršenja implicira
 - ▣ asinhronizam
 - ▣ nema otkaza

Problem međusobnog isključivanja (Mutex)

7

- Kod svakog procesora je podeljen u 4 sekcije:



- ▣ **ulaz:** sinhronizacija sa drugima radi međusobnog isključivanja prilikom pristupa...
- ▣ **kritičan:** koristi neki resurs; iza toga, uđi u...
- ▣ **izlaz:** raščišćavanje; iza toga, uđi u...
- ▣ **ostatak:** bez korišćenja resursa


Algoritmi međusobnog isključivanja

8

- *Algoritam međusobnog isključivanja* specificira kod za sekcije ulaza i izlaza radi obezbeđivanja:
 - ▣ **međusobnog isključivanja:** najviše 1 procesor je u svojoj kritičnoj sekciji u bilo kom trenutku, i
 - ▣ neka vrstu uslova „životnosti“ ili „napredka“ (progress). Postoji 3 uobičajena uslova koji se razmatraju...

Mutex uslovi napredka

9

- 
- **nema međusobnog blokiranja:** ako je procesor u svojoj ulaznoj sekciji u nekom trenutku, onda je kasnije neki procesor u svojoj kritičnoj sekciji
 - **nema trajnog zaključavanja:** ako je procesor u svojoj ulaznoj sekciji u nekom trenutku, onda je kasnije *isti* procesor u svojoj kritičnoj sekciji
 - **ograničeno čekanje:** kao predhodno + dok je procesor u svojoj ulaznoj sekciji, drugi procesori ulaze u kritičnu sekciju ograničen broj puta.
 - Ovi uslovi su navedeni u rastućem redosledu snage.

Algoritmi međusobnog isključivanja

10

- U kodu za ulazne i izlazne sekcije su dozvoljene sledeće pretpostavke:
 - ▣ ni jedan procesor ne ostaje u svojoj kritičnoj sekciji zauvek
 - ▣ deljenim prom., koje se koriste u ulaznim i izlaznim sekcijama, ne pristupa se u kritičnoj sekciji i sekciji ostatka

Mere složenosti za Mutex

11

- Važna mera složenosti za mutex algoritme je potrebna količina deljenog prostora.
- Veličina prostora zavisi od toga:
 - ▣ koliko je snažan tip deljenih promenljivih
 - ▣ koliko je snažan uslov napredka koji treba zadovoljiti (nema blokiranja ili nema trajnog zaključavanja ili ograničeno čekanje)

Mutex rezultati korišćenjem RMW

12

- Kad se koriste moćne deljene promenljive "read-modify-write" tipa

broj SM stanja	gornja granica	donja granica
nema međusobnog blokiranja	2 (test&set alg)	2 (očigledno)
nema trajnog zaklju. (bez pamćenja)	$n/2 + c$ (Burns et al.)	$\sqrt{(2n)} \quad (n/2)$ (Burns et al.)
ograničeno čekanje	n^2 (queue alg.)	n (Burns & Lynch)

Mutex rezultati korišćenjem Read/Write

13

- Kad se koriste read/write deljene promenljive

broj različitih prom.	gornja granica	donja granica
nema međusobnog blokiranja		n (Burns & Lynch)
nema trajnog zaključavanja	$3n$ bool prom. (Tournament alg.)	
ograničeno čekanje	$2n$ neograničeno (Bakery alg.)	

Test-and-Set deljena promenljiva

14

- **test-and-set** promenljiva V uzima 2 vred., 0 ili 1, i podržava 2 (atomske) operacije:

- **test&set(V):**

```
temp := V
V := 1
return temp
```

- **reset(V):**

```
V := 0
```

Mutex algoritam koji koristi Test&Set

15

- kod za ulaznu sekciju:

```
repeat  
    t := test&set(V)  
until (t = 0)
```

Alternativna sintaksna konstrukcija je:

```
wait until test&set(V) = 0
```

- kod za izlaznu sekciju:

```
reset(V)
```

Međusobno isključivanje je osigurano

16

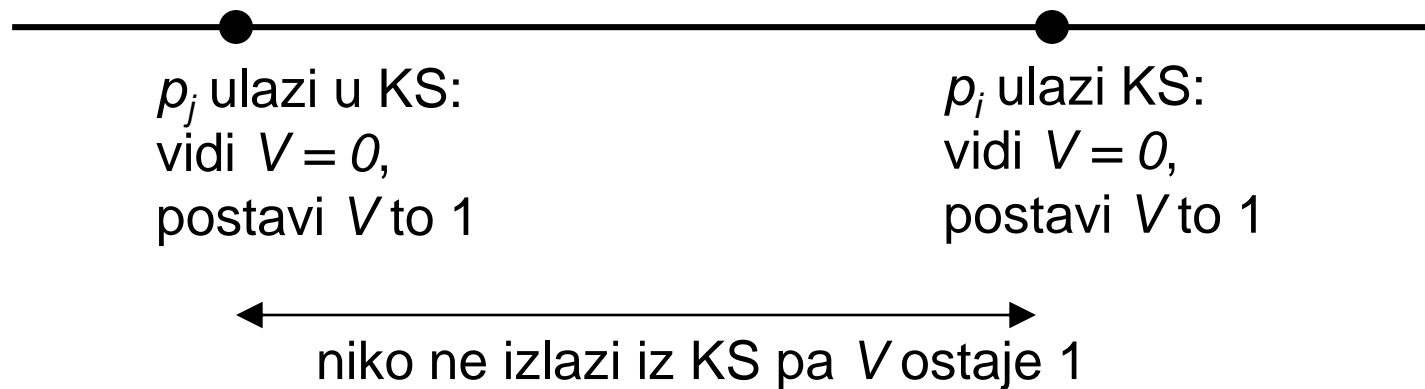
- Pred. da nije. Razmotri prvo narušavanje, kada neki p_i uđe u KS a drugi p_j je već u KS



Međusobno isključivanje je osigurano

16

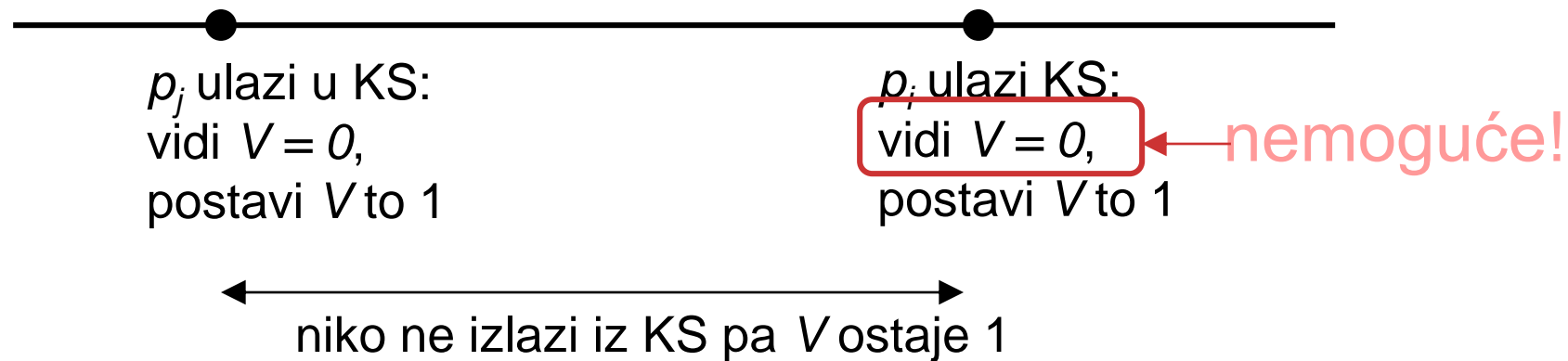
- Pred. da nije. Razmotri prvo narušavanje, kada neki p_i uđe u KS a drugi p_j je već u KS



Međusobno isključivanje je osigurano

16

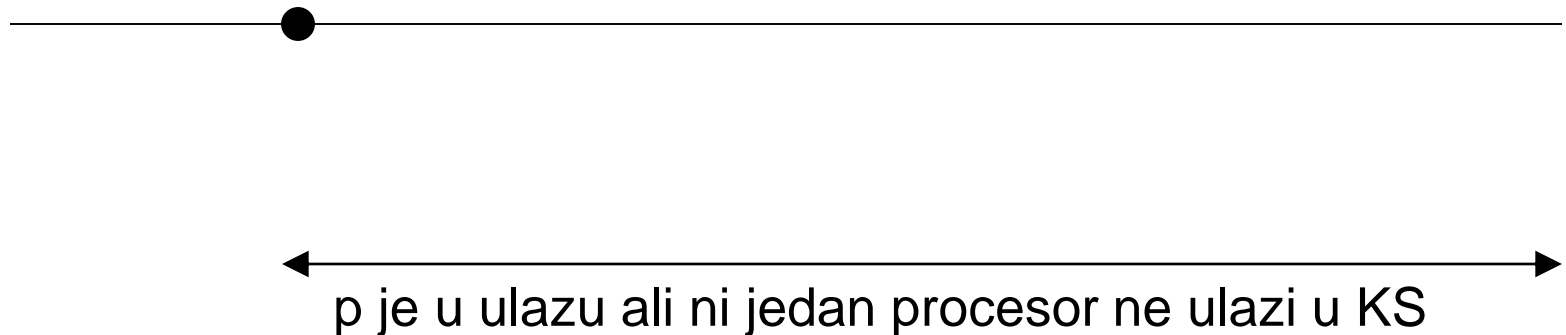
- Pred. da nije. Razmotri prvo narušavanje, kada neki p_i uđe u KS a drugi p_j je već u KS



Nema međusobnog blokiranja

17

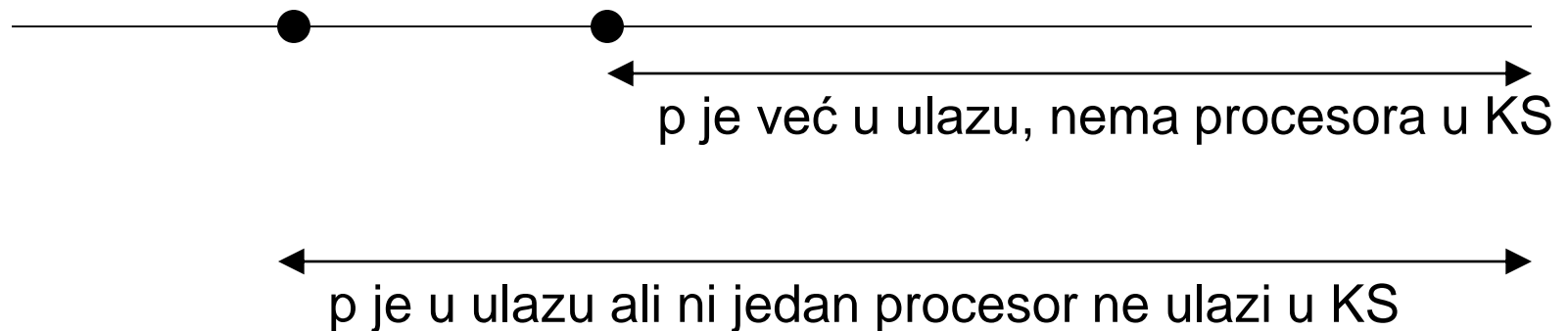
- *Tvrdnja:* $V = 0$ akko ni jedan procesor nije u KS.
 - ▣ Dokaz je indukcijom po događajima u izvršenju, uz oslonac na činjenicu da međusob. isključivanje važi.
- Predpost. postoji trenutak posle kad je proc. p u svojoj ulaznoj sekciji ali ni jedan proc. nikada ne ulazi u KS.



Nema međusobnog blokiranja

17

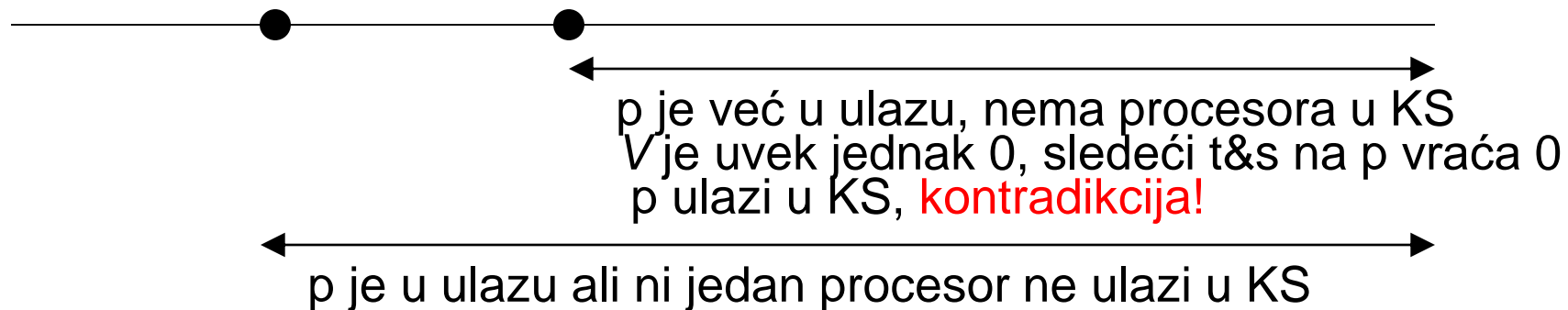
- *Tvrdnja:* $V = 0$ akko ni jedan procesor nije u KS.
 - Dokaz je indukcijom po događajima u izvršenju, uz oslonac na činjenicu da međusob. isključivanje važi.
- Predpost. postoji trenutak posle kad je proc. p u svojoj ulaznoj sekciji ali ni jedan proc. nikada ne ulazi u KS.



Nema međusobnog blokiranja

17

- *Tvrdnja:* $V = 0$ akko ni jedan procesor nije u KS.
 - Dokaz je indukcijom po događajima u izvršenju, uz oslonac na činjenicu da međusob. isključivanje važi.
- Predpost. postoji trenutak posle kad je proc. p u svojoj ulaznoj sekciji ali ni jedan proc. nikada ne ulazi u KS.



Da li ima trajnog zaključavanja?

18

- Neki procesor bi mogao da uvek pobeđuje u test&set utakmici, i da „izgladnjuje“ druge.
- Uslov odsustva trajnog zaključavanja nije ispunjen.
- Sledi da ni uslov ograničenog čekanja nije ispunjen.

Read-Modify-Write deljena promenljiva

19

- Stanje ove vrste promenljive može biti bilo šta i bilo koje veličine.
- Promenljiva V podržava (atomske) operacije
 - ▣ $\text{rmw}(V, f)$, gde je f bilo koja funkcija

```
temp := V
V := f(V)
return temp
```
- Ovaj tip promenljive je toliko jak da nema smisla imati više promenljivih (sa teorijskog stanovišta).

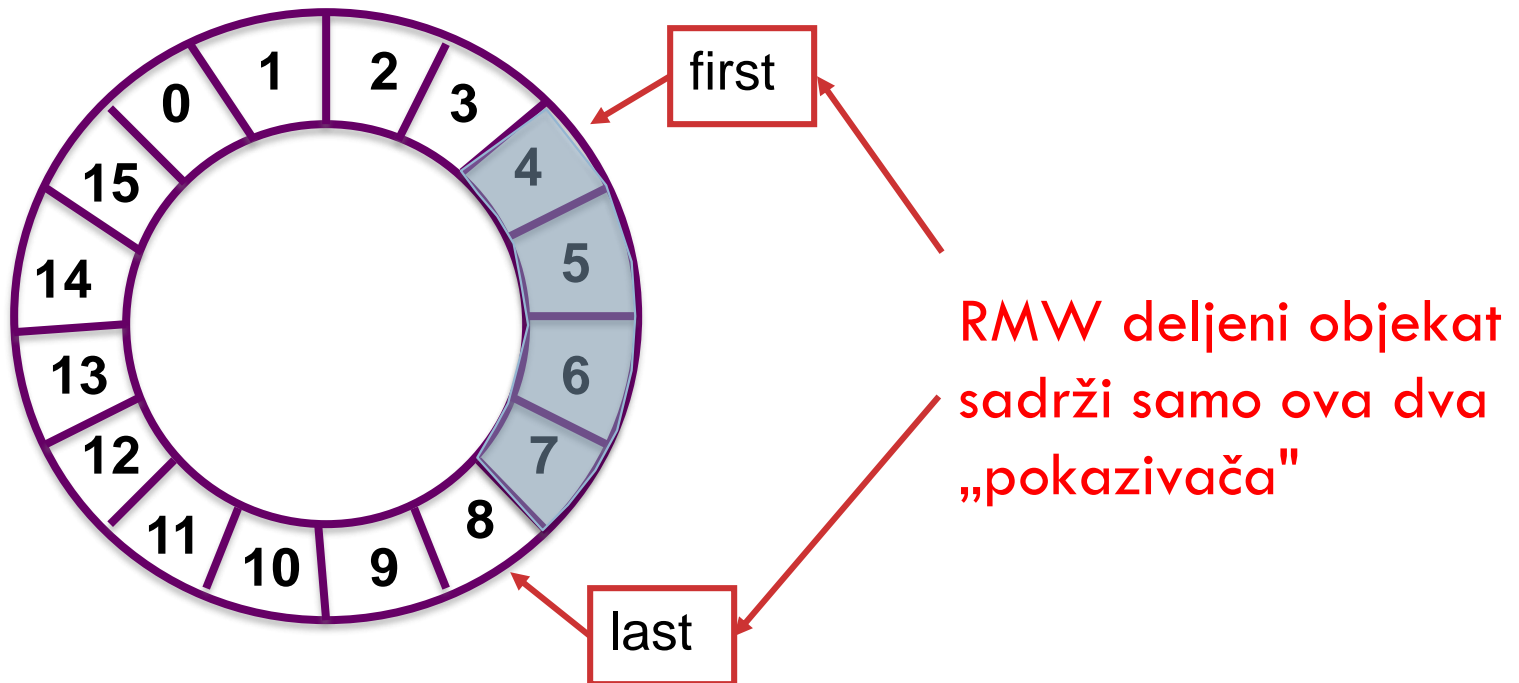
Mutex algoritam koji koristi RMW

20

- Konceptualno, lista procesora koji čekaju je smeštena u deljeni kružni red dužine n
- Svaki procesor pamti svoju lokaciju u redu u svom lokalnom stanju (umesto da se ova informacija drži u deljenoj promenljivoj)
- Deljena RMW prom. V čuva zapis o aktivnom delu reda pomoću indeksa *first* i *last*, to su indeksi reda (između 0 i $n-1$)
 - tako V ima dve komponente, *first* i *last*

Konceptualna struktura podataka

21



Mutex algoritam koji koristi RMW

22

□ Kod za ulaznu sekciju:

```
// povećaj last da bi ulančao sebe  
position := rmw(V, (V.first, V.last+1))  
// čekaj dok first ne dobije ovu vrednost  
repeat  
    queue := rmw(V, V)  
until (queue.first = position.last)
```

□ Kod za izlaznu sekciju:

```
// povećaj first da bi izlančao sebe  
rmw(V, (V.first+1, V.last))
```

Skica dokaza korektnosti

23

□ **Međusobno isključivanje:**

- Samo procesor na vrhu reda (*V.first*) može ući u KS, i samo jedan procesor je na vrhu u bilo kom trenutku.

□ ***n*-ograničeno čekanje:**

- FIFO redosled ulančavanja, i činjenica da ni jedan procesor ne ostaje u KS zauvek, daje ovaj rezultat.

Veličina potrebnog prostora

24

- Deljena RMW promenljiva V ima dve komponente u svom stanju, *first* i *last*.
- Obe su celi brojevi u opsegu od 0 do $n-1$, dakle n različitih vrednosti.
- Ukupan broj različitih stanja za V je onda n^2 .
- Sledi da je potrebna veličina V u bitima $2 \cdot \log_2 n$.

Kruženje (Spinning)

25

- Nedostatak RMW algoritma sa redom čekanja: procesor u ulaznoj sekciji u više prolaza pristupa **istoj** deljenoj promenljivoj
 - ▣ tzv. **kruženje**
- Kruženje više procesora na istoj deljenoj prom. može biti vrlo vremenski neefikasno ne nekim arhitekturama
- Promeni algoritam sa redom čekanja tako da svaki procesor kruži na drugoj **različitoj** deljenoj prom.

RMW Mutex algoritam sa zasebnim kruženjem (1 / 2)

26

Deljene RMW promenljive:

- *Last* : odgovara indeksu *last* iz org. alg.
 - ▣ kruži od 0 do $n-1$
 - ▣ vodi zapis o indeksu koji se dodeljuje sledećem procesoru da započne čekanje
 - ▣ inicijalno je 0

RMW Mutex algoritam sa zasebnim kruženjem (2/2)

27

Deljene RMW promenljive (nastavak):

- $Flags[0..n-1]$: niz binarnih promenljivih
 - ▣ to su prom. na kojima procesori kruže
 - ▣ treba osigurati da procesori ne kruže na istim promenljivama u istom trenutku
 - ▣ inicijalno $Flags[0] = 1$ (proc „ima bravu“) i $Flags[i] = 0$ (proc „mora da čeka“) za $i > 0$

Pregled algoritma

28

□ ulazna sekcija:

- uzmi sledeći indeks iz *Last* i smesti ga u lokalnu promenljivu *myPlace*
 - povećaj *Last* (sa zamotavanjem – eng. wrap-around)
- kruži na *Flags[myPlace]* dok ne postane 1 (znači proc „ima bravu“ i može ući u KS)
- postavi *Flags[myPlace]* na 0 („nema bravu“)

□ izlazna sekcija:

- postavi *Flags[myPlace+1]* na 1 (tj., daj prioritet sledećem proc)
 - koristiti modulo aritmetiku za zamotavanje

Pitanje

29

- Da li deljene promenljive *Last* i *Flags* moraju da budu RMW promenljive?

Pitanje

29

- Da li deljene promenljive *Last* i *Flags* moraju da budu RMW promenljive?
- **Odgovor:** RMW semantika (atomsko čitanje i ažuriranje promenljive) je potrebna za *Last*, da dva procesora ne bi dobili isti indeks u preklopljenim vremenskim intervalima.

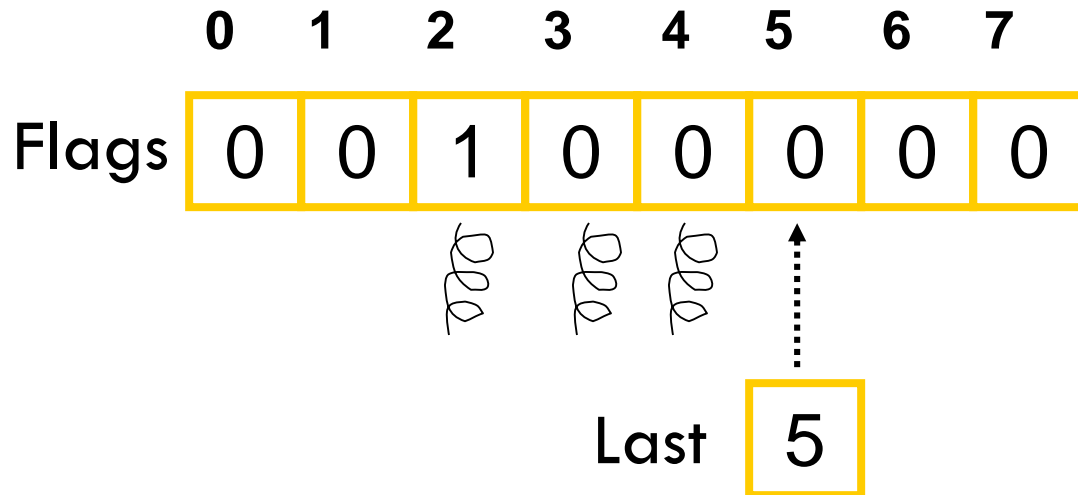
Invarijante algoritma

30

1. Najviše jedan element *Flags* ima vred. 1 („ima bravu“)
2. Ako ni jedan element *Flags* nema vred. 1, onda je neki procesor u KS.
3. Ako je $Flags[k] = 1$, onda je tačno
($Last - k$) mod n procesora u ulaznoj sekciji, kruže na $Flags[i]$, za $i = k, (k+1) \bmod n, \dots, (Last-1) \bmod n$.

Primer invarijanti

31



$k = 2$ i $Last = 5$.

Znači $5 - 2 = 3$ procesora su u ulazu,
kruže na $Flags[2]$, $Flags[3]$, $Flags[4]$

Korektnost

32

- Ove 3 invarijante se mogu koristiti da se dokaže:
 - ▣ Da je zadovoljeno međusobno isključivanje.
 - ▣ Da je zadovoljeno n -ograničeno čekanje.

Donja granica broja memorijskih stanja (1 / 4)

33

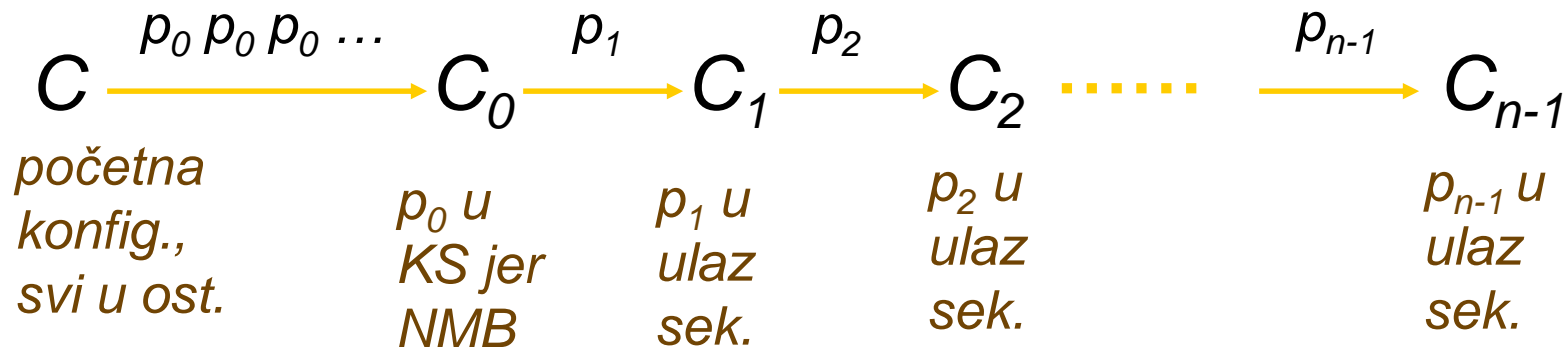
Teorema (4.4): bilo koji mutex algoritam sa k -ograničenim čekanjem (i bez među. blokiranja) koristi barem n stanja deljene memorije.

Dokaz: predpost. u cilju kontradikcije da postoji algoritam koji koristi manje od n stanja deljene memorije.

Donja granica broja memorijskih stanja (2/4)

34

- Razmotrimo izvršenje ovog algoritma:

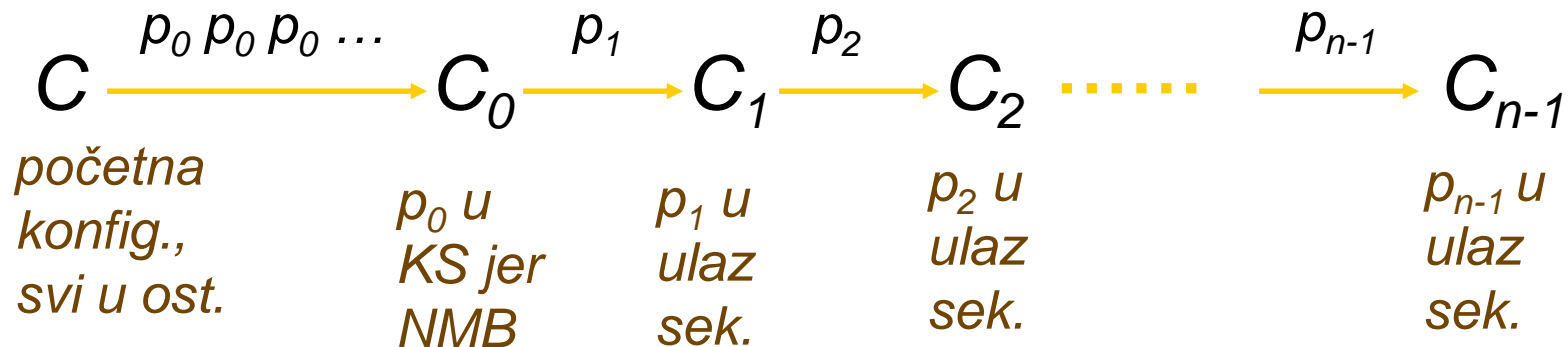


- Postoje i i j takvi da C_i i C_j imaju isto stanje deljene memorije.

Donja granica broja memorijskih stanja (2/4)

34

- Razmotrimo izvršenje ovog algoritma:



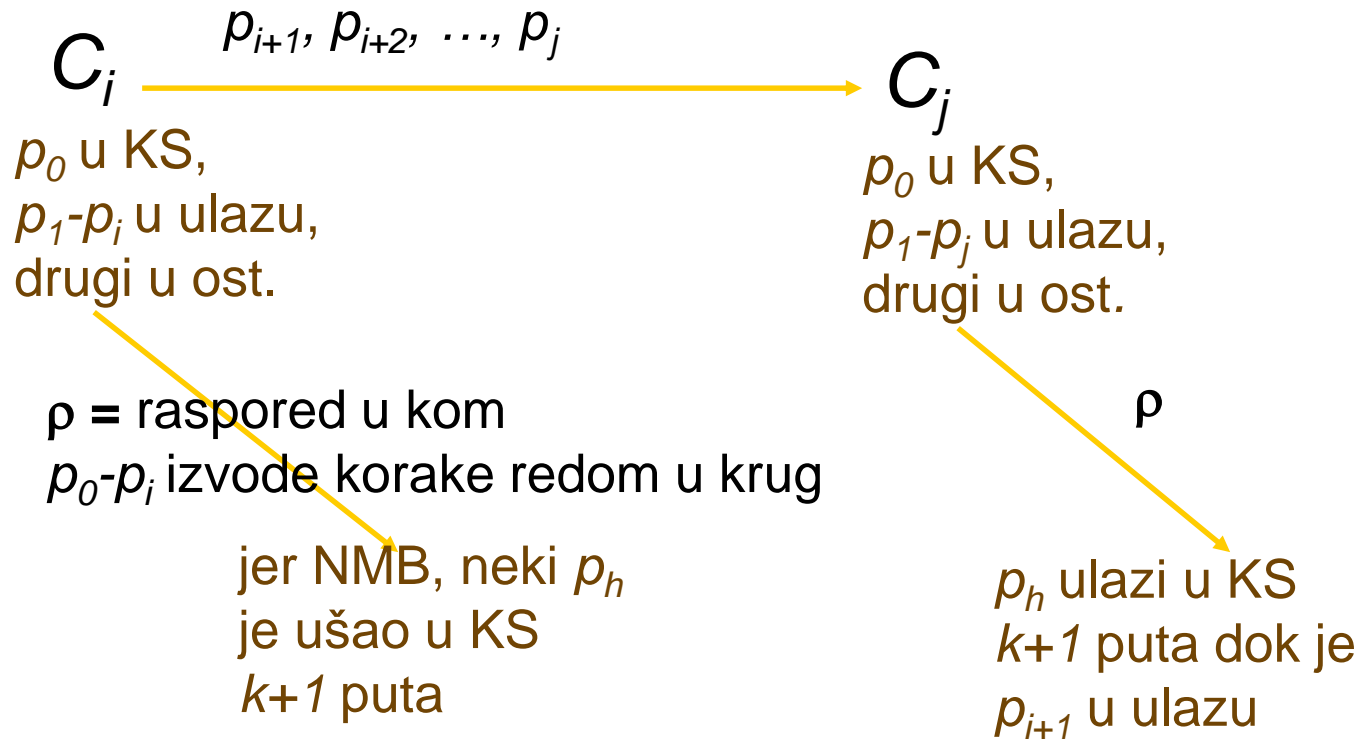
- Postoje i i j takvi da C_i i C_j imaju isto stanje deljene memorije.

zašto?

Donja granica broja memorijskih stanja (3/4)

35

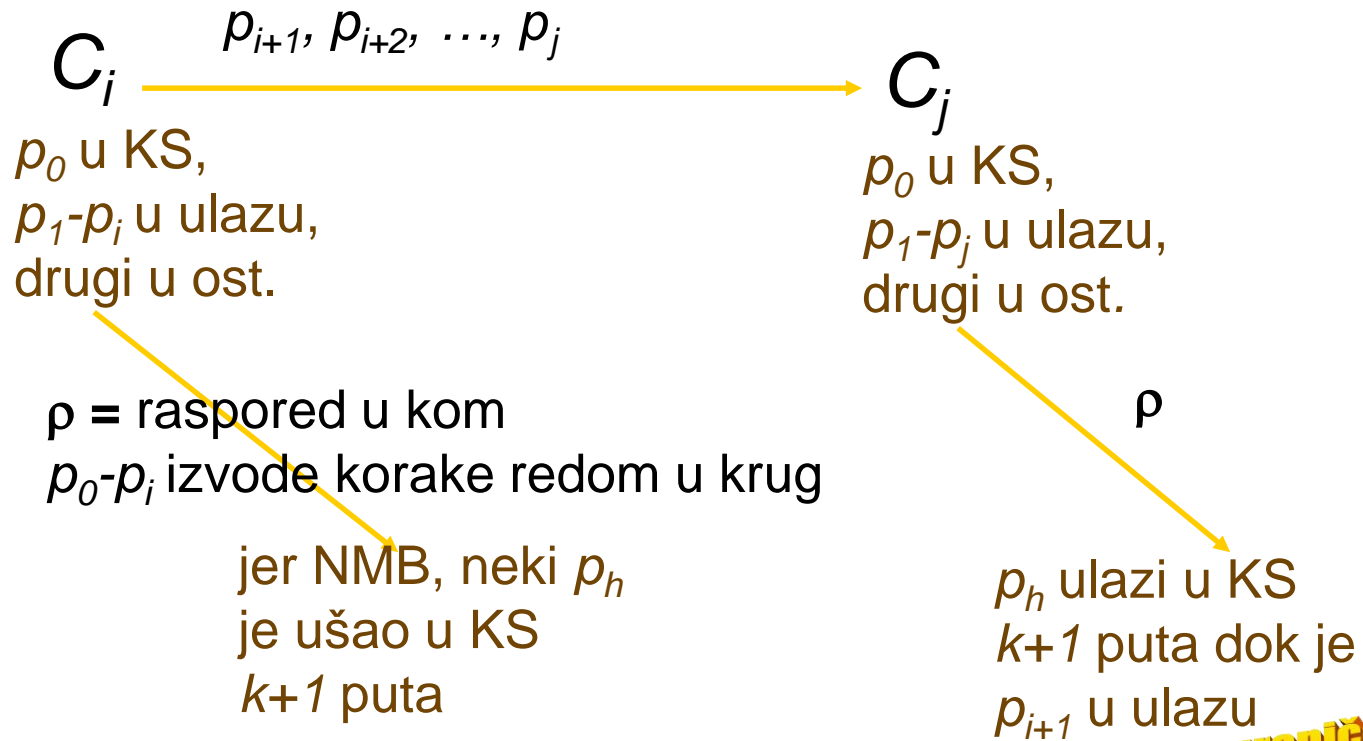
Stanje deljene memorije je isto u C_i kao u C_j



Donja granica broja memorijskih stanja (3/4)

35

Stanje deljene memorije je isto u C_i kao u C_j



**Kotradikcija k-ograničenog
čekanja!**

Donja granica broja memorijskih stanja

(4/4)

36

- Ali zašto p_h radi istu stvar dok izvršava sekvencu koraka u ρ kad kreće iz C_j kao kad kreće iz C_i ?
- Svi procesori p_0, \dots, p_i rade istu stvar zato što:
 - ▣ oni su u istim stanjima u te dve konfiguracije
 - ▣ stanje deljene memorije je isto u te dve konfiguracije
 - ▣ jedine razlike između C_i i C_j su (možda) stanja od p_{i+1}, \dots, p_j a ti procesori ne izvode nikakve korake u ρ

Diskusija donje granice

37

- Donja granica na n samo pokazuje broj mem. stanja i samo važi za algoritme koji garantuju ograničeno čekanje u svakom izvršenju.
- Predpost. da oslabimo uslov napredka da samo nema trajnog zaključavanja u svakom izvršenju: tada granica postaje $n/2$ različitih stanja deljene memorije.
- Ako uslov napredka oslabimo da samo nema međusob. blok. u svakom izvršenju, onda je granica samo 2.

„Pobeđivanje“ donje granice randomizacijom

38

- Alternativan način da se oslabe zahtevi je da se odustane od uslova životnosti u svakom izvršenju
- **Probabilistički** uslov da nema neograničenog blokiranja: svaki proc ima verovatnoću koja nije 0 da će uspeti svaki put kad uđe u ulaznu sekciju.
- Za to postoji algoritam koji koristi $O(1)$ stanja deljene memorije.