

# DISTRIBUIRANI ALGORITMI I SISTEMI

# Tipovi podataka izvan registara

2

- Registri podržavaju *read* i *write* operacije
- Moguće su WF simulacije jedne vrste registara pomoću druge vrste
  - ▣ različiti brojevi vrednosti, čitača, pisača
- Šta je sa (WF) simuliranjem značajno različitih vrsta tipova podataka pomoću registara?
- Generalno, šta je sa (WF) simuliranjem objekta tipa  $X$  pomoću objekata tipa  $Y$  ?

# Ključan uvid

3

- Sposobnost objekata tipa  $Y$  da mogu simulirati objekte tipa  $X$  je povezana sa sposobnošću tih tipova podataka da reše konsenzus!
- Fokusiramo se na sisteme koji su:
  - asinhroni
  - sa deljenom memorijom
  - oslobođeni-čekanja (WF)

# Primer FIFO reda

4

- Sekvencijalna specifikacija FIFO reda:
  - ▣ operacija sa pozivom  $enq(x)$  i odgovorom  $ack$
  - ▣ operacija sa pozivom  $deq$  i odgovorom  $return(x)$
  - ▣ sekvenca operacija je prihvatljiva akko svaki  $deq$  vraća najstariju ulančanu vred koja još uvek nije izlančana (vraća  $\perp$  ako je red prazan)

# Algoritam konsenzusa za $n = 2$ sa korišćenjem FIFO reda

5

Inicijalno  $Q = [0]$  i  $Prefer[i] = \perp$

$Prefer[i] :=$  ulaz od  $p_i$

$val := deq(Q)$

**if**  $val = 0$  **then**

    odluči se za ulaz od  $p_i$

**else**

$temp := Prefer[1 - i]$

    odluči se za  $temp$

# Algoritam konsenzusa za $n = 2$ sa korišćenjem FIFO reda

5

Inicijalno  $Q = [0]$  i  $Prefer[i] = \perp$

*jedan deljeni FIFO red  
i dva deljena registra*

$Prefer[i] :=$  ulaz od  $p_i$

$val := deq(Q)$

**if**  $val = 0$  **then**

odluči se za ulaz od  $p_i$

**else**

$temp := Prefer[1 - i]$

odluči se za  $temp$

# Algoritam konsenzusa za $n = 2$ sa korišćenjem FIFO reda

5

Inicijalno  $Q = [0]$  i  $Prefer[i] = \perp$

*jedan deljeni FIFO red  
i dva deljena registra*

$Prefer[i] :=$  ulaz od  $p_i$

*upiši moj ulaz u moj registar*

$val := deq(Q)$

**if**  $val = 0$  **then**

odluči se za ulaz od  $p_i$

**else**

$temp := Prefer[1 - i]$

odluči se za  $temp$

# Algoritam konsenzusa za $n = 2$ sa korišćenjem FIFO reda

5

Inicijalno  $Q = [0]$  i  $Prefer[i] = \perp$

*jedan deljeni FIFO red  
i dva deljena registra*

$Prefer[i] :=$  ulaz od  $p_i$

*upiši moj ulaz u moj registar*

$val := deq(Q)$

**if**  $val = 0$  **then**

odluči se za ulaz od  $p_i$

*koristi deljeni red za arbitražu  
između 2 proc: prvi koji  
izlanča početnu 0-lu pobeđuje,  
vrednost odluke je njegov ulaz*

**else**

$temp := Prefer[1 - i]$

odluči se za  $temp$



# Algoritam konsenzusa za $n = 2$ sa korišćenjem FIFO reda

5

Inicijalno  $Q = [0]$  i  $Prefer[i] = \perp$

*jedan deljeni FIFO red  
i dva deljena registra*

$Prefer[i] :=$  ulaz od  $p_i$

*upiši moj ulaz u moj registar*

$val := deq(Q)$

**if**  $val = 0$  **then**

odluči se za ulaz od  $p_i$

*koristi deljeni red za arbitražu  
između 2 proc: prvi koji  
izlanča početnu 0-lu pobeđuje,  
vrednost odluke je njegov ulaz*

**else**

$temp := Prefer[1 - i]$

odluči se za  $temp$

*gubitnik dobija vred  
odluke iz registra  
drugog proc*

# Implikacije algoritma konsenzusa sa korišćenjem FIFO reda

6

- Prepodst. da želimo da simuliramo na WF način FIFO red koristeći read/write registre
- Da li je to moguće?
- Ne! Ako bi bilo moguće, mogli bi da rešimo konsenzus:
  - ▣ simuliraj FIFO red koristeći registre
  - ▣ koristi simulirani red i predhodni algoritam za rešenje konsenzusa

# Proširenje algoritma za više proc?

7

- Da li možemo koristiti FIFO redove za rešenje konsenzusa za više od 2 proc?
- Sposobnost atomskog izlančavanja vred je bila ključ za algoritam za 2 proc:
  - ▣ jedan proc. saznaje da je pobednik
  - ▣ drugi saznaje da je gubitnik
- Nije jasno kako treba rukovati sa 3 proc.
- Predpost. da imamo različit tip podataka:

# Specifikacija Compare & Swap

8

`compare&swap(X : adresa deljene mem,`

`old: vred,`

`new: vred)`

`previous := X // previous je lok. prom.`

`if previous = old then X := new`

`return previous`

*dešava se atomski*

**X**

~~old~~  
new

# Algoritam konsenzusa sa korišćenjem Compare & Swap

9

Initially  $First = \perp$

$val := compare\&swap(First, \perp, \text{moj ulaz})$

**if**  $val = \perp$  **then**

    odluči se za svoj ulaz

**else**

    odluči se za  $val$

# Algoritam konsenzusa sa korišćenjem Compare & Swap

9

Initially  $First = \perp$

jedan deljen C&S objekt

$val := compare\&swap(First, \perp, \text{moj ulaz})$

**if**  $val = \perp$  **then**

odluči se za svoj ulaz

**else**

odluči se za  $val$

# Algoritam konsenzusa sa korišćenjem Compare & Swap

9

Initially  $First = \perp$

jedan deljen C&S objekt

if  $First = \perp$  then stavi moj ulaz

$val := compare\&swap(First, \perp, \text{moj ulaz})$

**if**  $val = \perp$  **then**

odluči se za svoj ulaz

**else**

odluči se za  $val$

# Algoritam konsenzusa sa korišćenjem Compare & Swap

9

Initially  $First = \perp$

jedan deljen C&S objekt

if  $First = \perp$  then stavi moj ulaz

$val := compare\&swap(First, \perp, \text{moj ulaz})$

**if**  $val = \perp$  **then**

odluči se za svoj ulaz

**else**

odluči se za  $val$

istovremeno odredi da li  
si pobednik i vrednost za  
koju treba da se odluče svi  
gubitnici



# Nemogućnost konsenzusa 3-Proc pomoću FIFO reda

10

**Teorema (11.3):** WF konsenzus pomoću FIFO redova i registara nije moguć ako je  $n > 2$

**Dokaz:** Iste strukture kao za registre

Ključna razlika je kad se razmatra situacija u kojoj je:

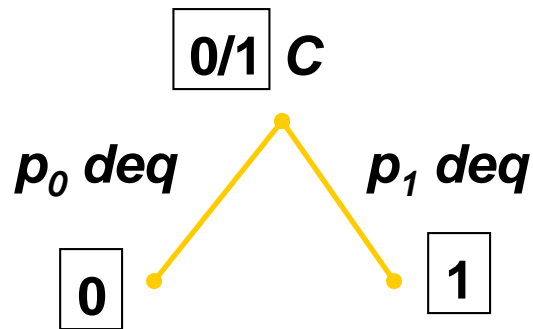
- $C$  bivalentna
- $p_0(C)$  0-valentna i  $p_1(C)$  je 1-valentna

# Nemogućnost konsenzusa 3-Proc pomoću FIFO reda

11

- $p_0$  i  $p_1$  moraju da pristupaju istom FIFO redu

*Sluč. 1*: Oba koraka su *deq*

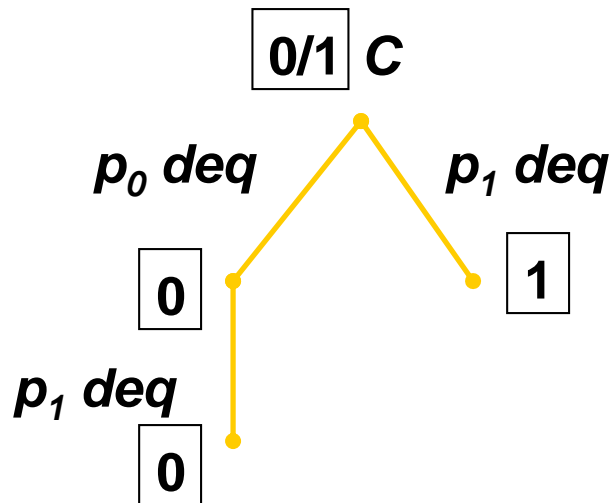


# Nemogućnost konsenzusa 3-Proc pomoću FIFO reda

11

- $p_0$  i  $p_1$  moraju da pristupaju istom FIFO redu

*Sluč. 1*: Oba koraka su *deq*

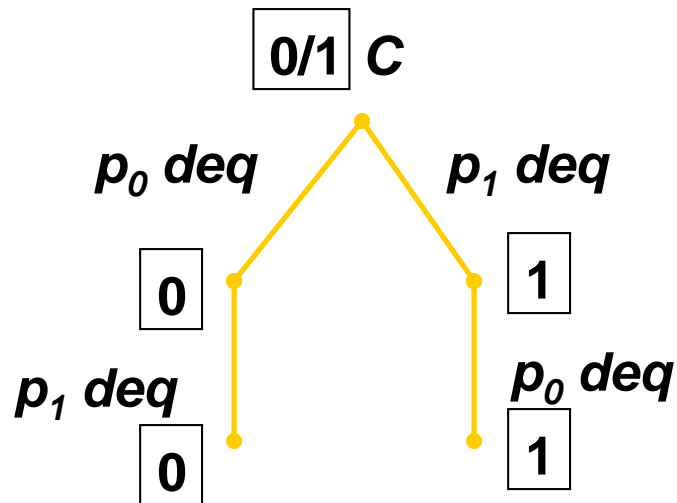


# Nemogućnost konsenzusa 3-Proc pomoću FIFO reda

11

- $p_0$  i  $p_1$  moraju da pristupaju istom FIFO redu

*Sluč. 1*: Oba koraka su *deq*

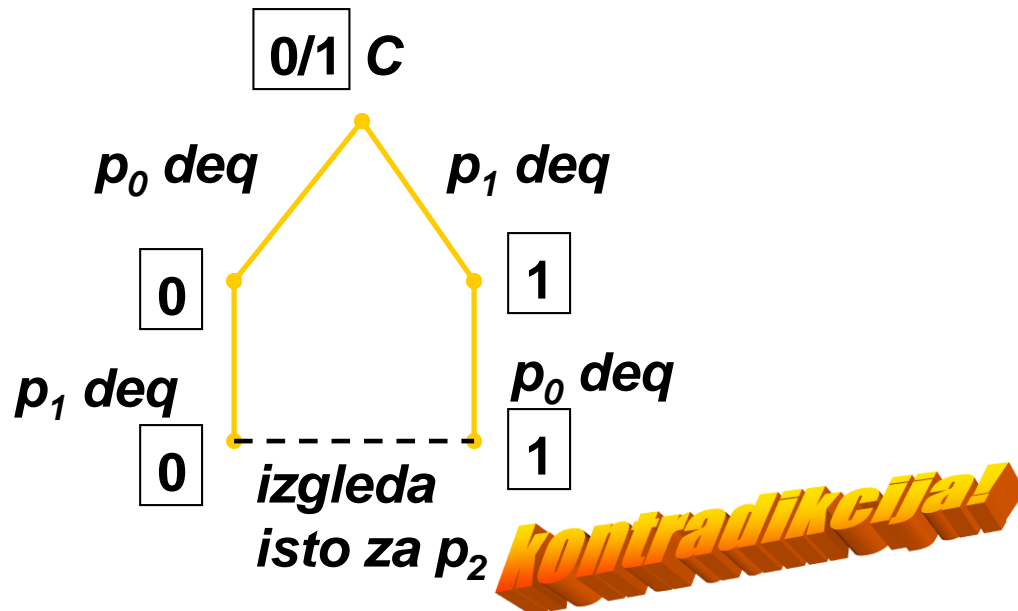


# Nemogućnost konsenzusa 3-Proc pomoću FIFO reda

11

- $p_0$  i  $p_1$  moraju da pristupaju istom FIFO redu

*Sluč. 1*: Oba koraka su *deq*

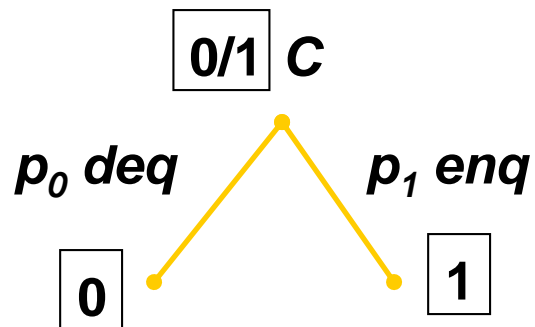


# Dokaz nemogućnosti

12

*Sluč. 2:*  $p_0 \text{ deq}$  i  $p_1 \text{ enq}$

*Sluč. 2.1:* Red nije prazan u  $C$

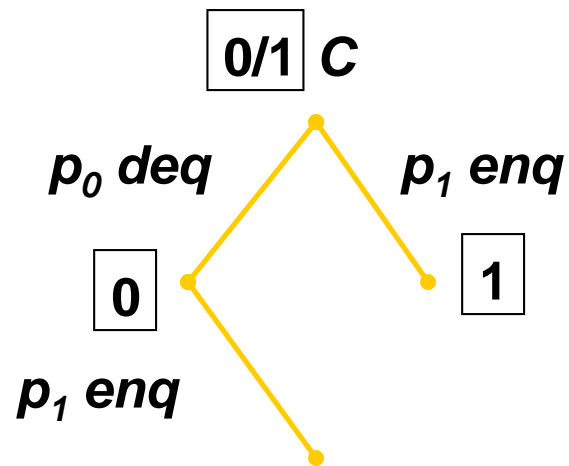


# Dokaz nemogućnosti

12

Sluč. 2:  $p_0 \text{ deq}$  i  $p_1 \text{ enq}$

Sluč. 2.1: Red nije prazan u C

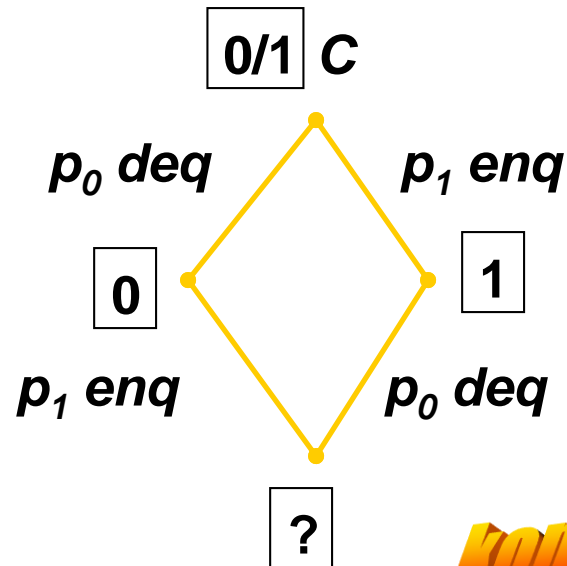


# Dokaz nemogućnosti

12

Sluč. 2:  $p_0 \text{ deq}$  i  $p_1 \text{ enq}$

Sluč. 2.1: Red nije prazan u C



**kontradikcija!**

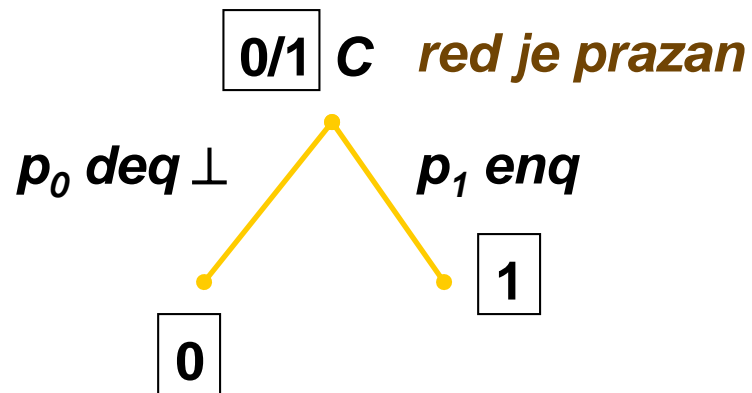


# Dokaz nemogućnosti

13

Sluč. 2:  $p_0 \text{ deq}$  i  $p_1 \text{ enq}$

Sluč. 2.2: Red je prazan u C

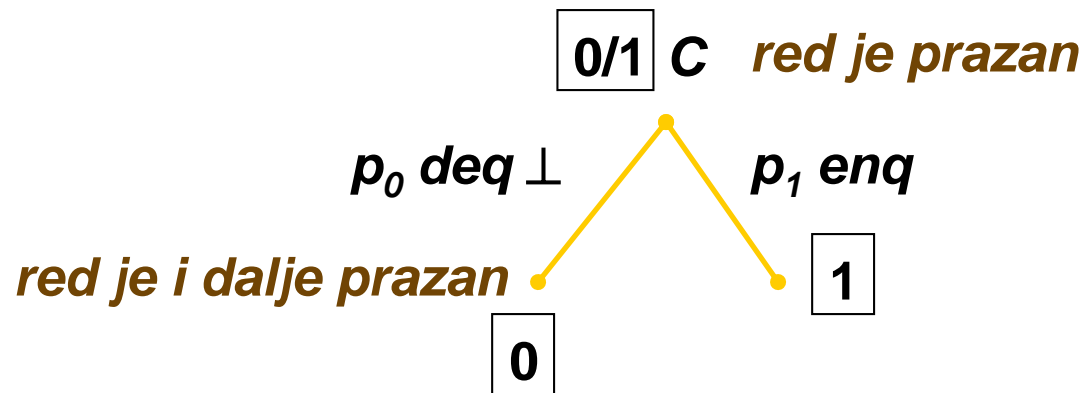


# Dokaz nemogućnosti

13

Sluč. 2:  $p_0 \text{ deq}$  i  $p_1 \text{ enq}$

Sluč. 2.2: Red je prazan u C

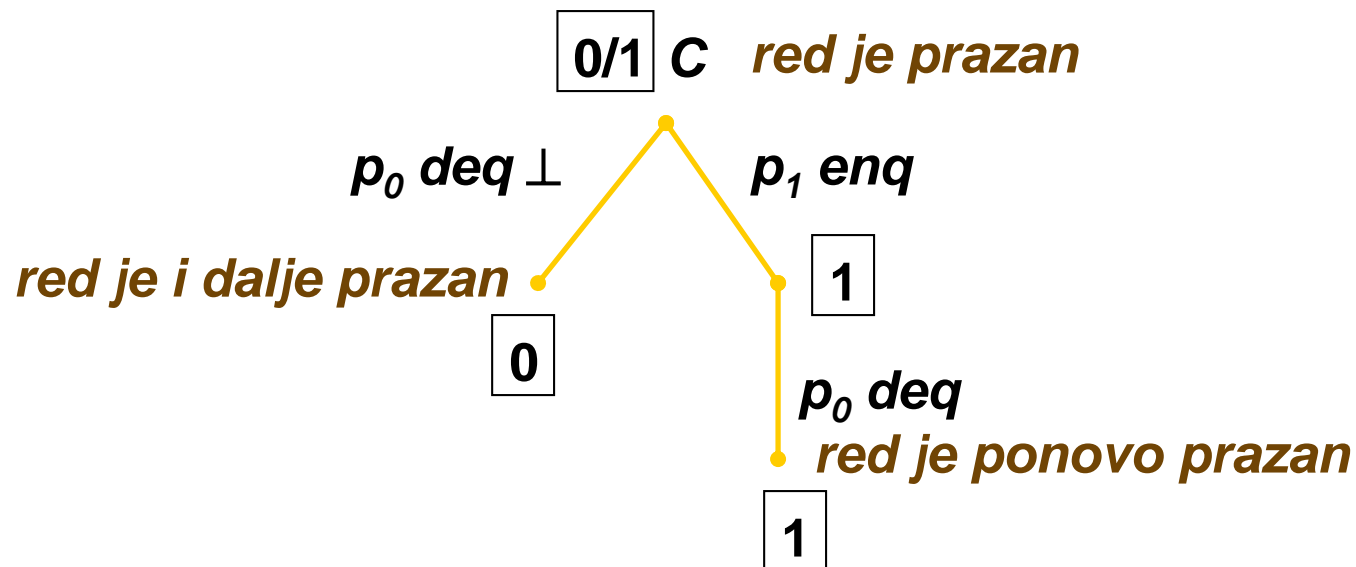


# Dokaz nemogućnosti

13

Sluč. 2:  $p_0$  deq i  $p_1$  enq

Sluč. 2.2: Red je prazan u C

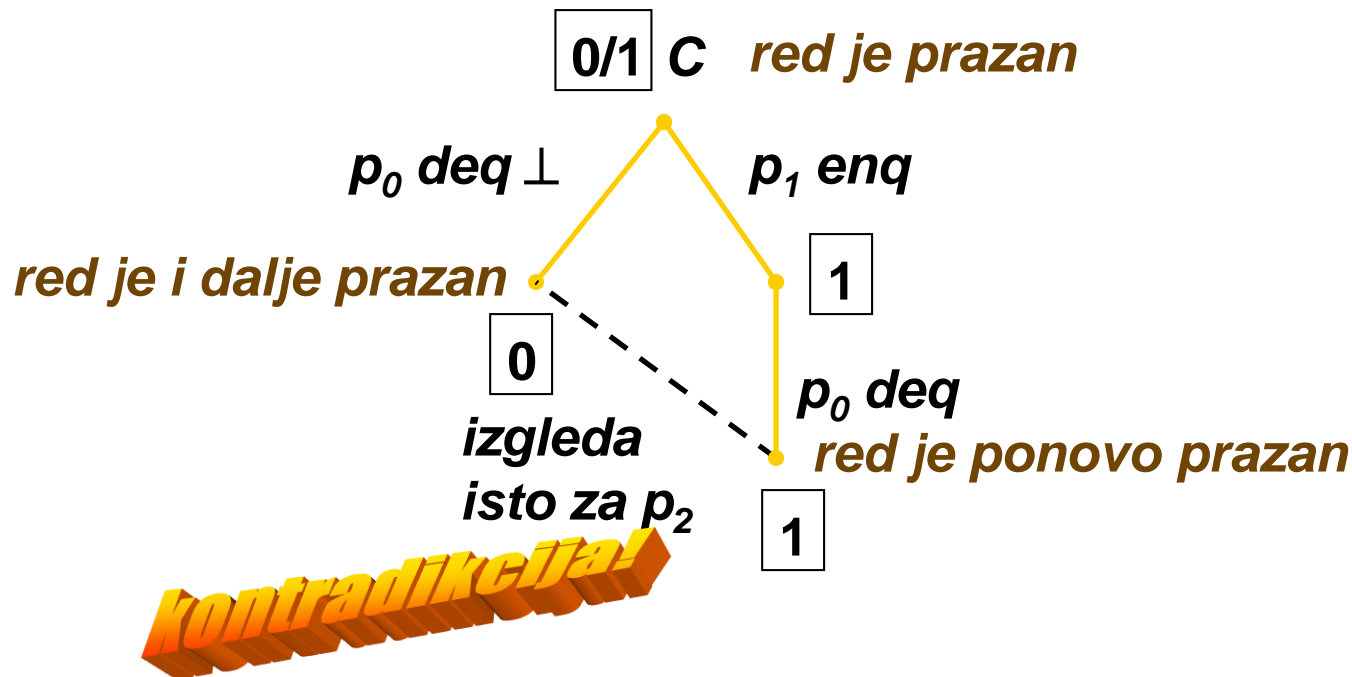


# Dokaz nemogućnosti

13

Sluč. 2:  $p_0 \text{ deq}$  i  $p_1 \text{ enq}$

Sluč. 2.2: Red je prazan u C



# Dokaz nemogućnosti

14

*Sluč. 3:* Oba  $p_0$  i  $p_1$  enq (na istom redu).

Domaći: Pogledati dokaz u knjizi.

Zaključak: ne postoji WF algoritam sa korišćenjem  
FIFO redova i registara za konsenzus za 3 proc.

# Implikacije

17

- Predpost. da hoćemo da simuliramo C&S objekt na WF način pomoću FIFO redova (i registara)
- Da li je to moguće?
- Ne ako je  $n > 2!$  Da je to moguće, mogli bi da rešimo konsenzus pomoću FIFO redova (i reg.):
  - ▣ simuliraj C&S objekt pomoću FIFO redova (i reg.)
  - ▣ koristi simulirani C&S objekt i c&s algoritam za rešenje konsenzusa

# Generalizacija ovih argumenata

18

- Predhodni rezultati u vezi FIFO redova i C&S objekata sugerišu kriterijum za određivanje da li postoje WF simulacije:
- na osnovu sposobnosti tipova podataka da reše konsenzus za određeni broj proc.

# Broj konsenzusa

19

Tip podataka  $X$  ima **broj konsenzusa**  $n$  ako je  $n$  najveći broj proc. za koje se konsenzus može rešiti samo pomoću objekata tipa  $X$  i read/write registara

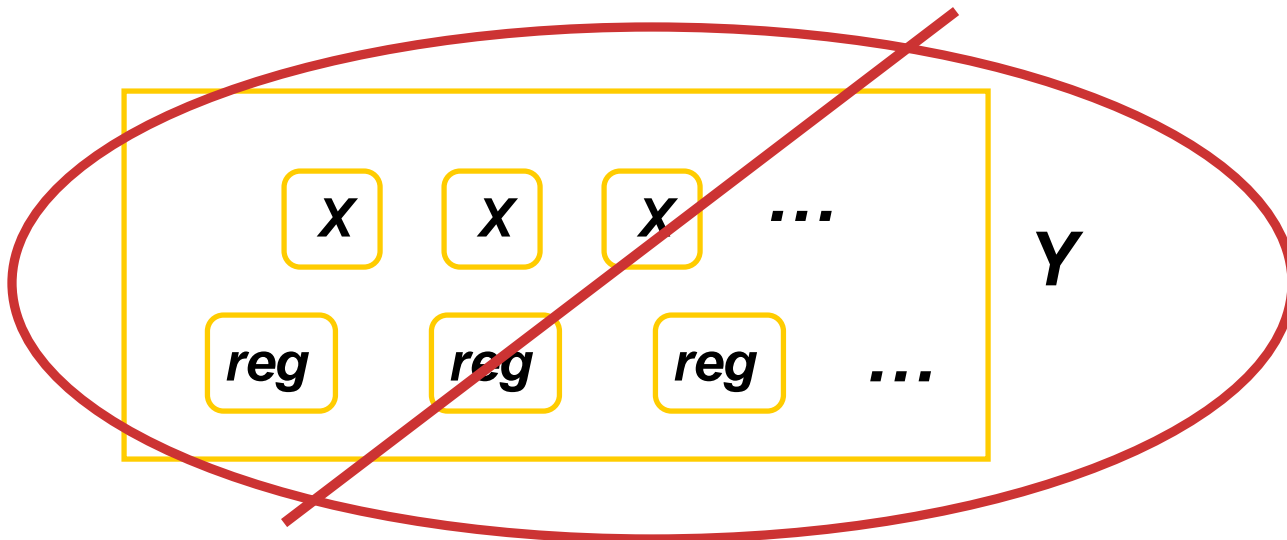
<b>tip data</b>	<b>broj konsenzusa</b>
read/write registar	1
FIFO red	2
compare&swap	$\infty$



# Korišćenje brojeva konsenzusa

20

**Teorema (11.5):** Ako tip pod.  $X$  ima broj konsenzusa  $m$  i tip pod.  $Y$  ima broj konsenzusa  $n$  sa  $n > m$ , onda ne postoji WF simulacija objekta tipa  $Y$  pomoću objekata tipa  $X$  i read/write registara u sistemu sa više od  $m$  proc



# Korišćenje brojeva konsenzusa

21

- Dokaz:** Predpost. radi kontradikcije da postoji WF simulacija  $S$  od  $Y$  pomoću  $X$  i registara u sistemu sa  $k$  proc, gde je  $m < k \leq n$
- Konstruišimo algoritam konsenzusa za  $k > m$  proc pomoću objekata tipa  $X$  (i registara):
    - ▣ Koristi  $S$  za simulaciju objekata tipa  $Y$  pomoću objekata tipa  $X$  (i registara)
    - ▣ Koristi simulirane objekte tipa  $Y$  (i registre) u algoritmu konsenzusa za  $k$  proc, koji postoji jer je  $CN(Y) = n$

**kontradikcija za  $CN(X) < k$**

# Posledice

22

- Ne postoji WF simulacija bilo kog objekta sa br. konsenzusa  $> 1$  samo pomoću read/write registara
- Ne postoji WF simulacija bilo kog objekta sa br. konsenzusa  $> 2$  samo pomoću FIFO redova i read/write registara

# Univerzalnost

23

- Razmotrimo sad pozitivne rezultate u vezi broja konsenzusa
- Tip podataka je **univerzalan** ako objekti tog tipa (zajedno sa read/write registrima) mogu da simuliraju bilo koji tip podataka na WF način
- **Teorema:** Ako tip podataka  $X$  ima broj konsenzusa  $n$ , onda je on univerzalan u sistemu sa najviše  $n$  proc

# Dokazivanje univerzalnosti

24

1. Opiši algoritam koji simulira bilo koji tip podataka
  - ▣ koristi C&S (umesto bilo kog objekta sa brojem konsenzusa  $n$ )
  - ▣ simulacija je samo **neblokirajuća**, slabija od WF
2. Promeni da koristi bilo koji objekt sa brojem konsenzusa  $n$
3. Promeni da bude WF
4. Promeni da se ograniči korišćena delj. mem.

# Neblokirajuća (Non-blocking, NB)

25

- Odnos NB prema WF je analogan sa odnosom nema međusobnog blokiranja prema nema izgladnjivanja (no-lockout) kod međusobnog isključivanja
- **Neblokirajuća** simulacija: u bilo kojoj tački izvršenja, ako je bar jedna operacija nerešena (odgovor još nije spreman), onda postoji konačna sekvenca koraka od strane jednog proc koja završava *jednu* od nerešenih operacija
- Ne garantuje da će se svaka nerešena operacija na kraju završiti

# Univerzalna konstrukcija

26

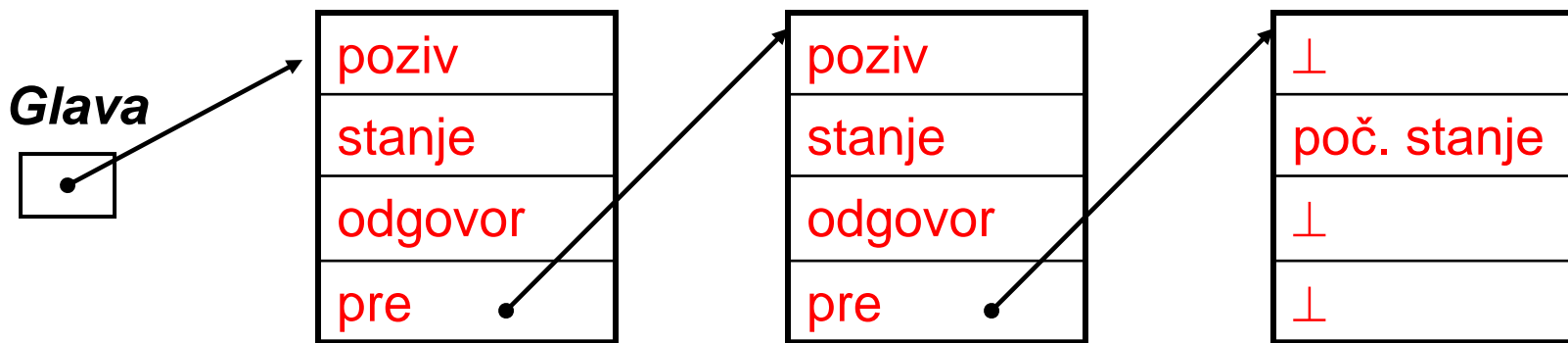
- Čuvaj istoriju operacija koje su bile primenjene na simulirani objekt kao deljenu listu
- Da bi primenio operaciju na simulirani objekt, pozivajući proc. mora da umetne odgovarajući „čvor“ u tu listu:
  - ▣ zgodno je staviti najnoviji čvor na *glavu* (početak) liste
- C&S objekt se koristi za ažuriranje glave te liste

# Detalji povezane liste

27

Svaki čvor povezane lista ima:

- poziv (operacija koja se poziva)
- novo stanje simuliranog objekta
- odgovor za operaciju
- pokazivač na predhodni čvor (predhodnu op)





# Simulacija

28

Na početku Head pokazuje na čvor kotva

- ▣ predstavlja početno stanje simuliranog objekta

Kad se desi poziv inv:

- ▣ dodeli novi čvor liste u deljenoj memoriji, na koji pokazuje lokalna promenljiva point

- ▣ `point.inv := inv`

- ▣ repeat

- ▣ `h := Head // h je lokalna prom`

zavisi od simuliranog  
tipa podataka

- ▣ `point.state, point.response := apply(inv,h.state)`

- ▣ `point.before := h`

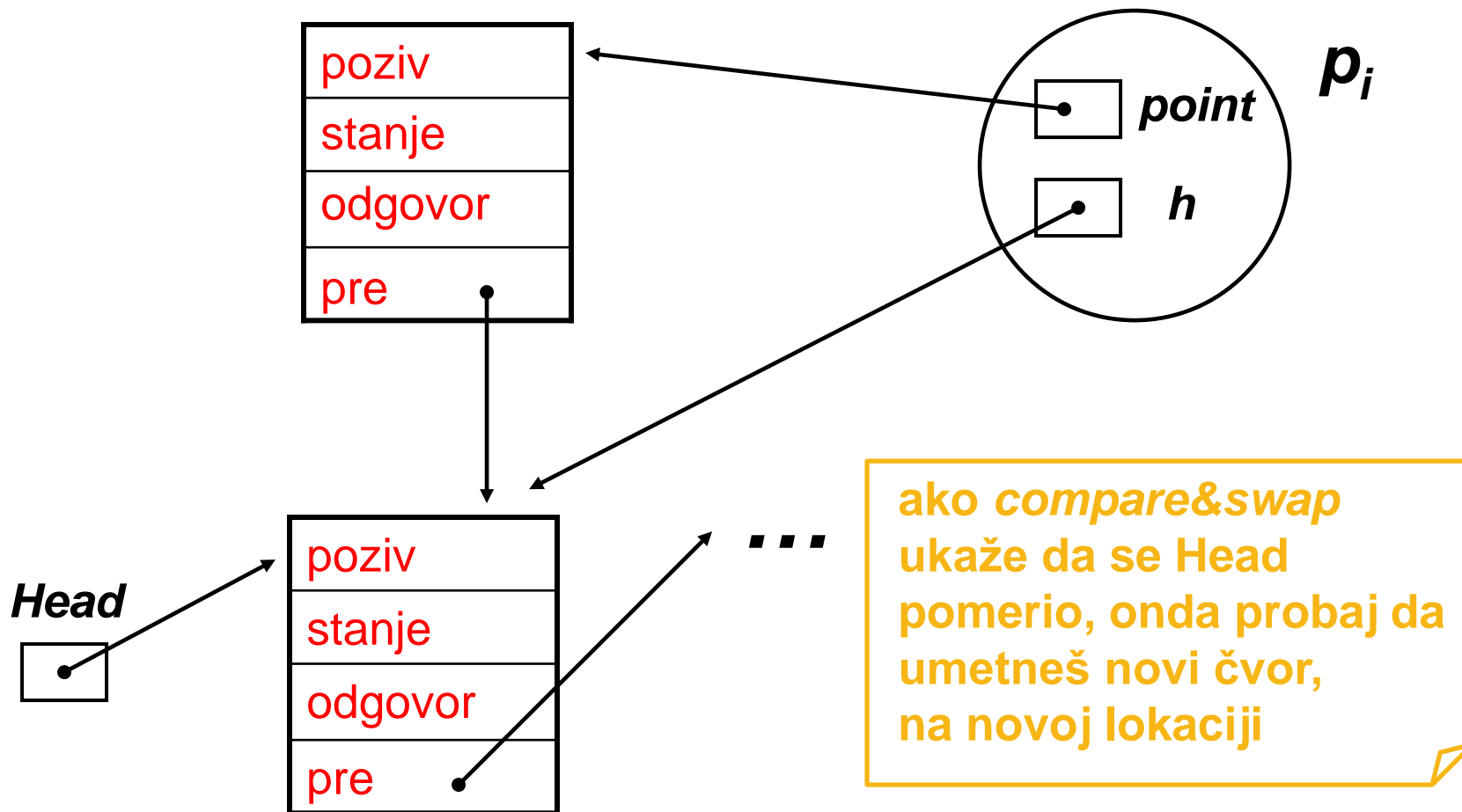
- ▣ until `compare&swap(Head,h,point) = h`

- ▣ omogući izlaz određen sa `point.response`

ako *Head* i dalje poka.  
na isti čvor kao *h*,  
onda usmeri *Head*  
da pokaže na nov čvor

# Slika simulacije

29



# Ojačanja algoritma

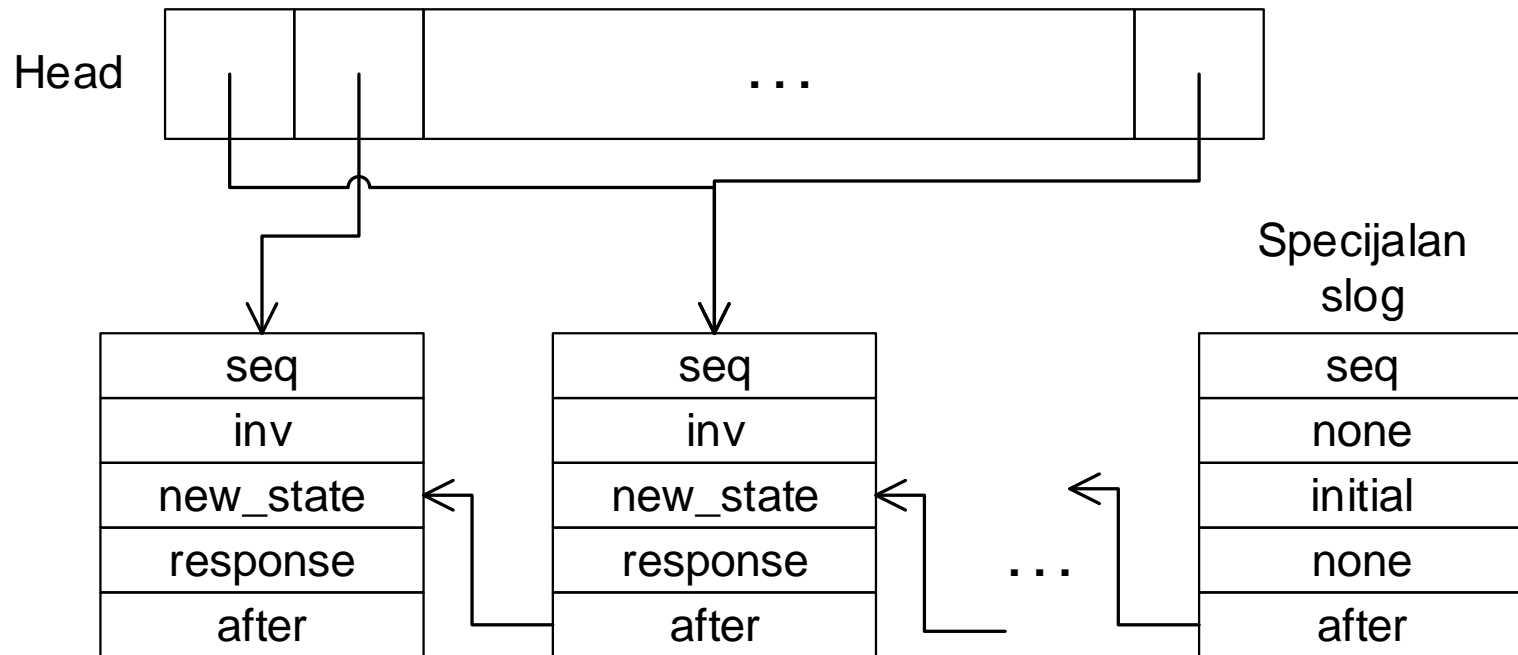
30

- Da bi zamenili C&S objekt sa bilo kojim objektom sa brojem konsenzusa  $n$  (broj proc):
  - ▣ definiši **objekt konsenzusa** (verziju tipa podataka za problem konsenzusa)
  - ▣ zaobiđi poteškoću da se objekt konsenzusa može koristiti samo jednom, dodajući objekt konsenzusa u svaki čvor povezane liste

# NB sim. sa obj. konsenzusa $CN = n$

44

Head[i] je pokazivač za proc  $p_i$



Polje after je objekt sa  $CN = n$

Procesori se bore da je ulančaju na vrh liste

# Dalja ojačanja algoritma

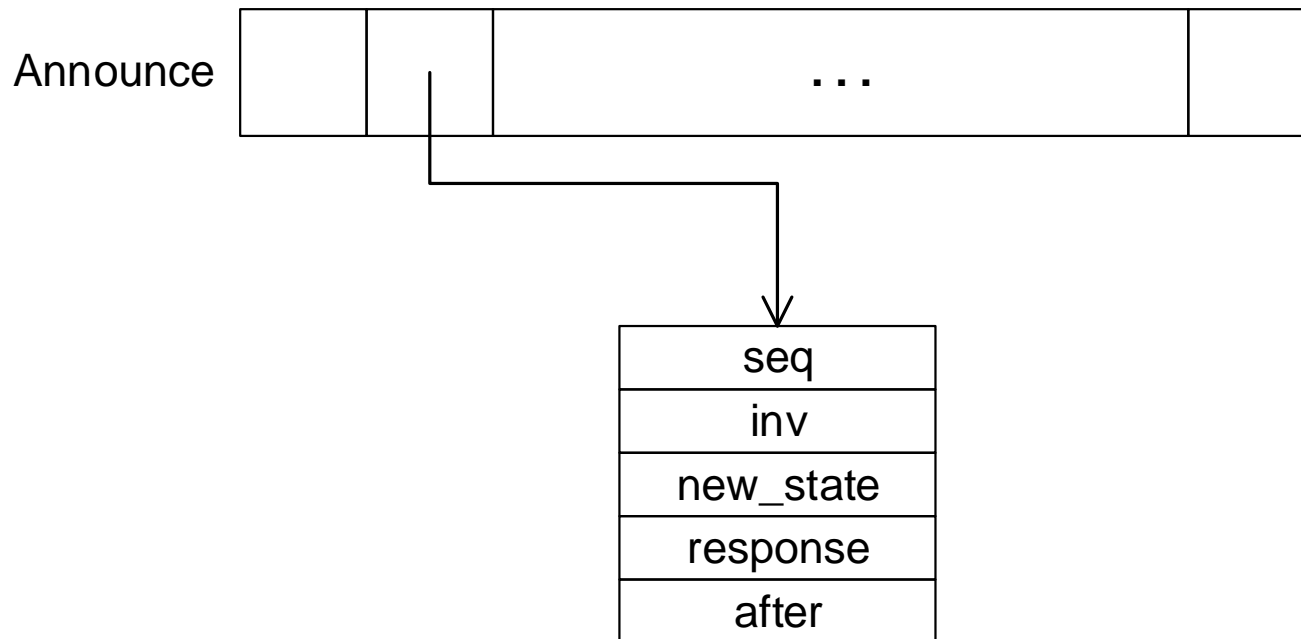
31

- Da bi dobili WF implementaciju, koristiti ideju **pomaganja**: proc pomažu jedan drugom da završe nezavršene operacije (ne samo svoje sopstvene)
- Da bi smanjili veličinu liste (da ne raste bez ograničenja), potrebno je voditi evidenciju o čvorovima liste koji se mogu reciklirati
- Da bi podržali nedeterminističke operacije, polja `new_state` i `response` smeštamo u dodatni objekt konsenzusa

# WF Simulacija (1 / 2)

46

Announce[i] je pokazivač za proc  $p_i$



Prioritet ima proc  $p_i$ ,  $i = (\text{seqmax} + 1) \% n$

Ako čeka, svi mu pomažu, inače pomažu sebi

# WF Simulacija (2/2)

47

**Algoritam 11.5.** WF univerzalni algoritam korišćenjem objekata konsenzusa:

Kod za procesor  $p_i$ ,  $0 \leq i \leq n - 1$ .

Na početku  $Head[j]$  i  $Announce[j]$  pokazuje na specijalan slog, za  $0 \leq j \leq n - 1$

```
01: kad se desi inv:                               // poziv operacije, sa parametrima
02: Announce[i] := alloc(opr)                       // alociraj novi opr slog
03: Announce[i].inv := inv, Announce[i].seq := 0 // postavi polja inv i seq
04: for j := 0 to  $n - 1$  do // pronađi slog sa najvećim rednim brojem
05:   if Head[j].seq > Head[i].seq then Head[i] := Head[j]
06: while Announce[i].seq = 0
07:   priority := (Head[i].seq + 1) mod  $n$  // indeks procesora sa prioriteto
08:   if Announce[priority].seq = 0 // proveriti da li je potrebna pomoć
09:     then point := Announce[priority] // da, izaberi slog drugoga
10:     else point := Announce[i] // ne, izaberi svoj slog
11:   win := decide(Head[i].after, point) // pokušaj da umetneš izabran slog
12:   win.new_state, win.response := apply(win.inv, Head[i].new_state)
13:   win.seq := Head[i].seq + 1
14:   Head[i] := win // ukaži na slog na vrhu liste
15: omogući izlaz na koji ukazuje point.response // odgovor operacije
```

# Efekat randomizacije

32

- Predpost. da oslabimo uslov životnosti za deljenu mem. čije sekvence oper. se mogu linearizovati:
  - ▣ operacije se moraju završiti sa velikom verovatnoćom
- Sad se randomizirani algoritam konsenzusa može koristiti za simulaciju bilo kog tipa podataka pomoću bilo kog drugog tipa podataka, uključujući read/write registre
- Tj., nestaje hijerarhija