Međusobno isključivanje

read/write promenljive

Algoritam sa ceduljicama (Bakery Algorithm)

Ovaj algoritam je sličan sa read-modify-write algoritmom

Postoji red čekanja:

Proces na početku (glavi) reda je u kritičnoj sekciji

Nov proces se ubacuje na kraj (rep)

Skica algoritma

Ulaz:

```
t = tail;
tail = tail + 1;
Wait until t == head;
```

Kritična sekcija

Izlaz:

```
head = head + 1;
```

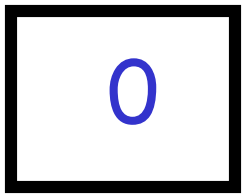
(***tail*** i ***head*** su deljene promenljive,
t je lokalna promenljiva)

Problem: ovaj deo koda se ne ponaša korektno

Ulaz:

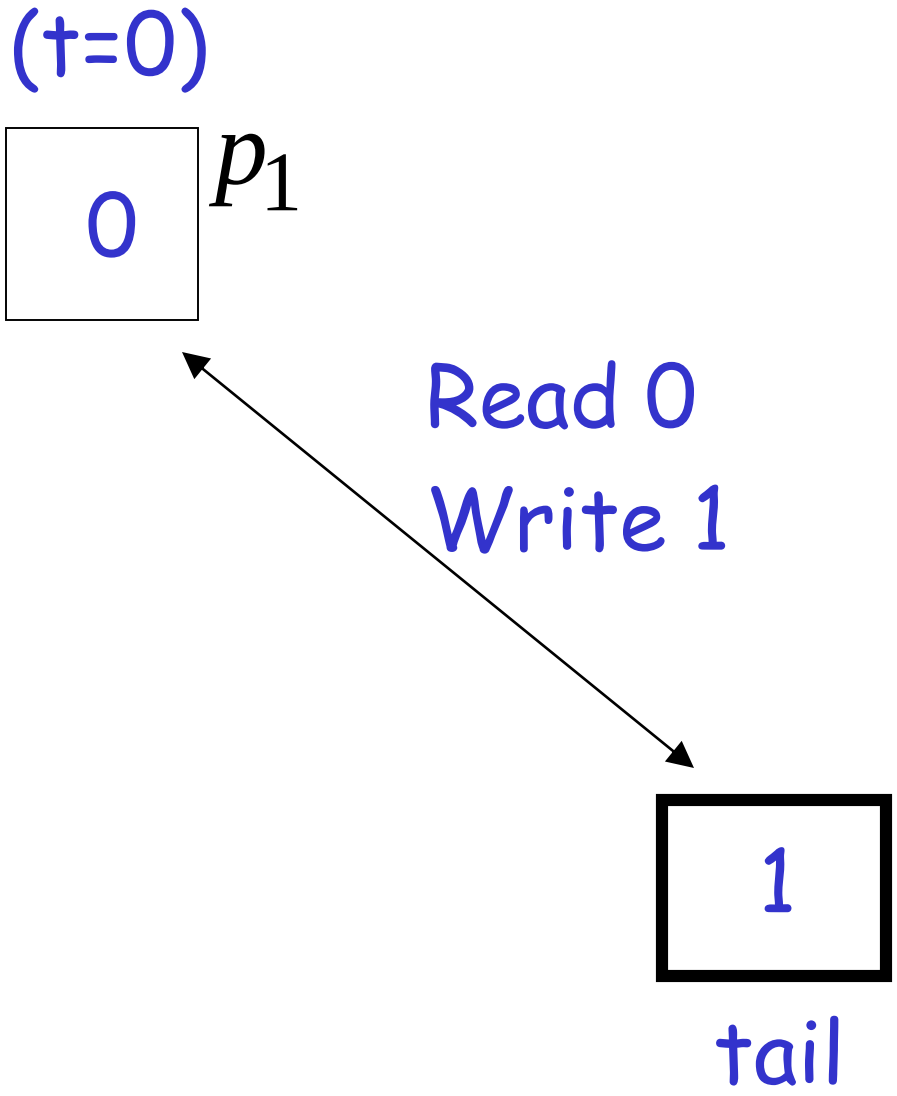
```
t = tail;          //read tail
tail = tail + 1;   //write tail
```

Dobar scenario

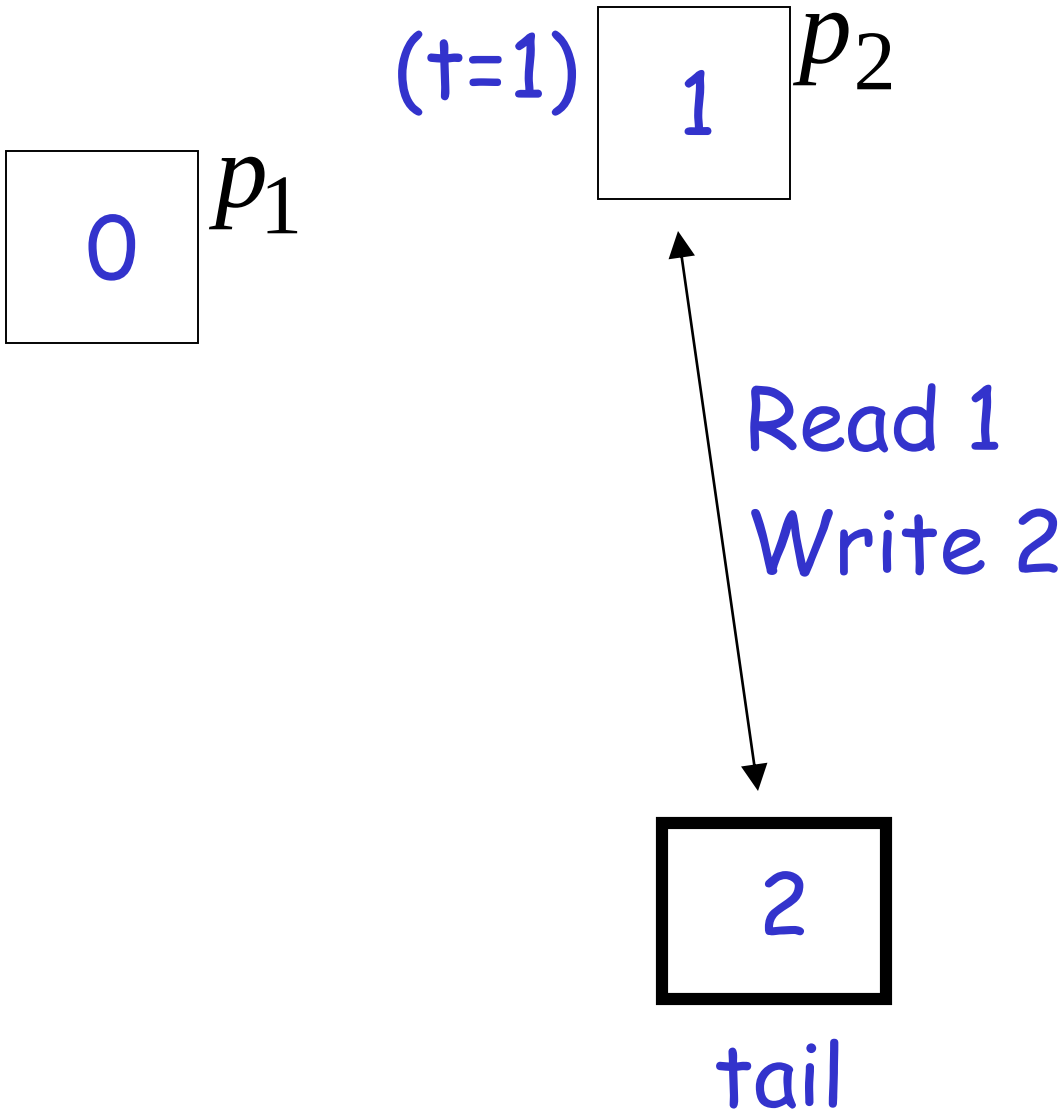


tail

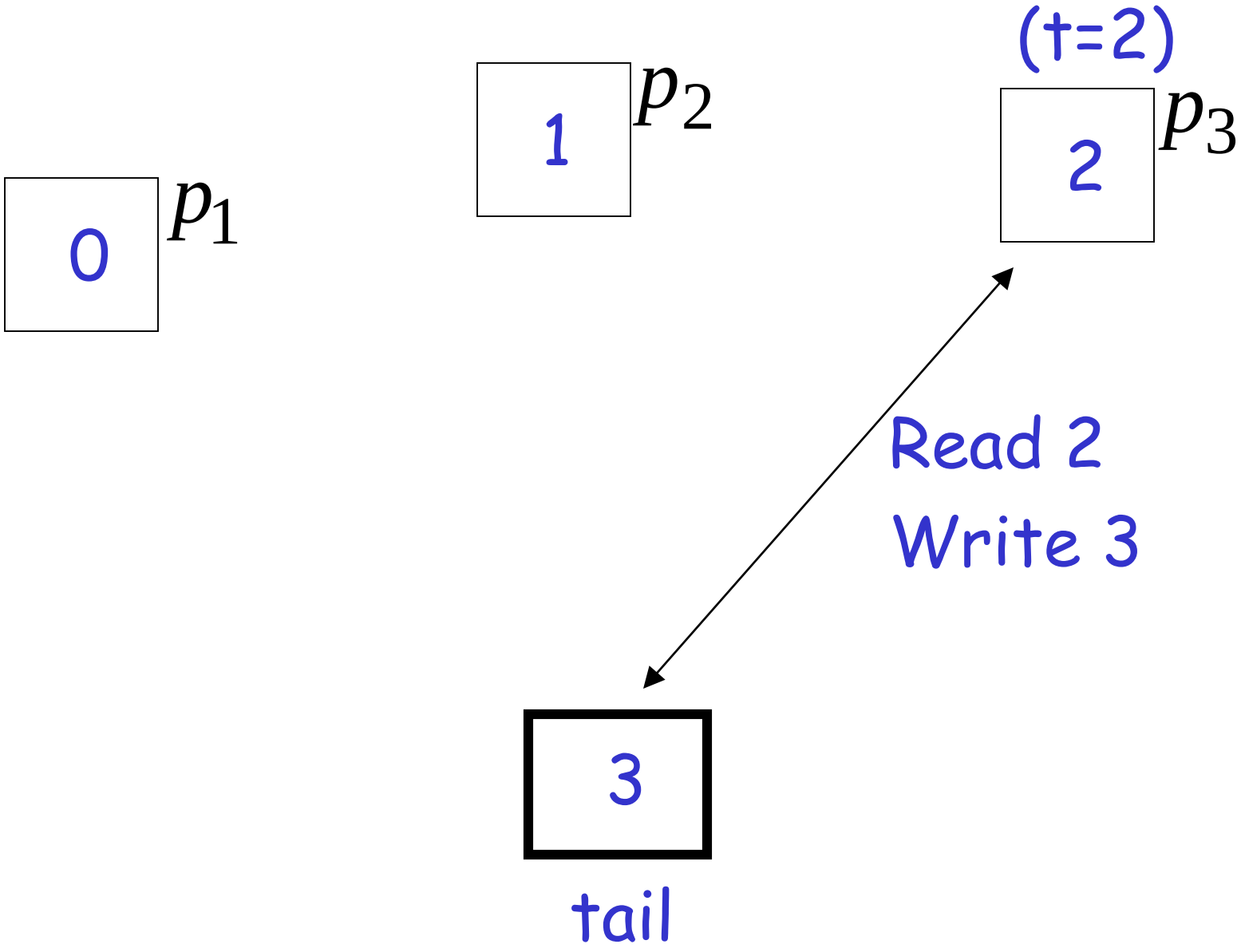
Dobar scenario



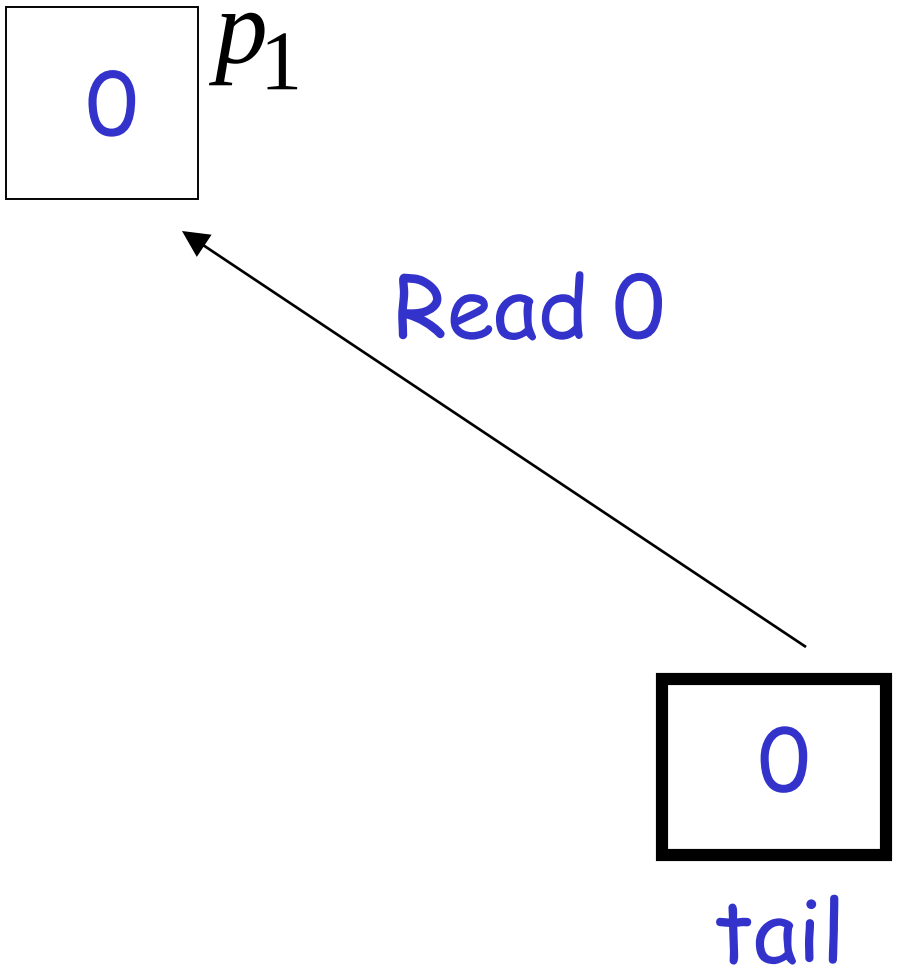
Dobar scenario



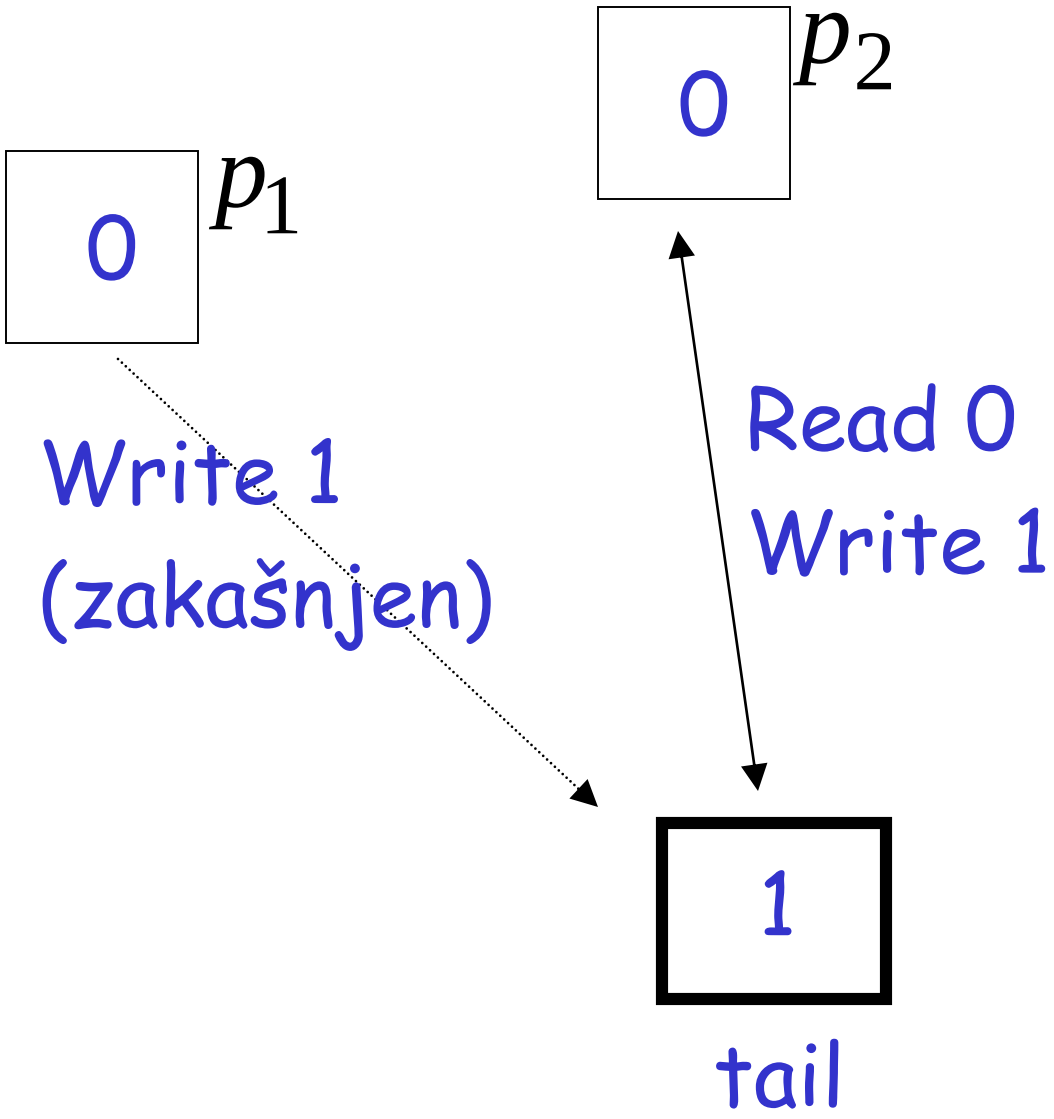
Dobar scenario



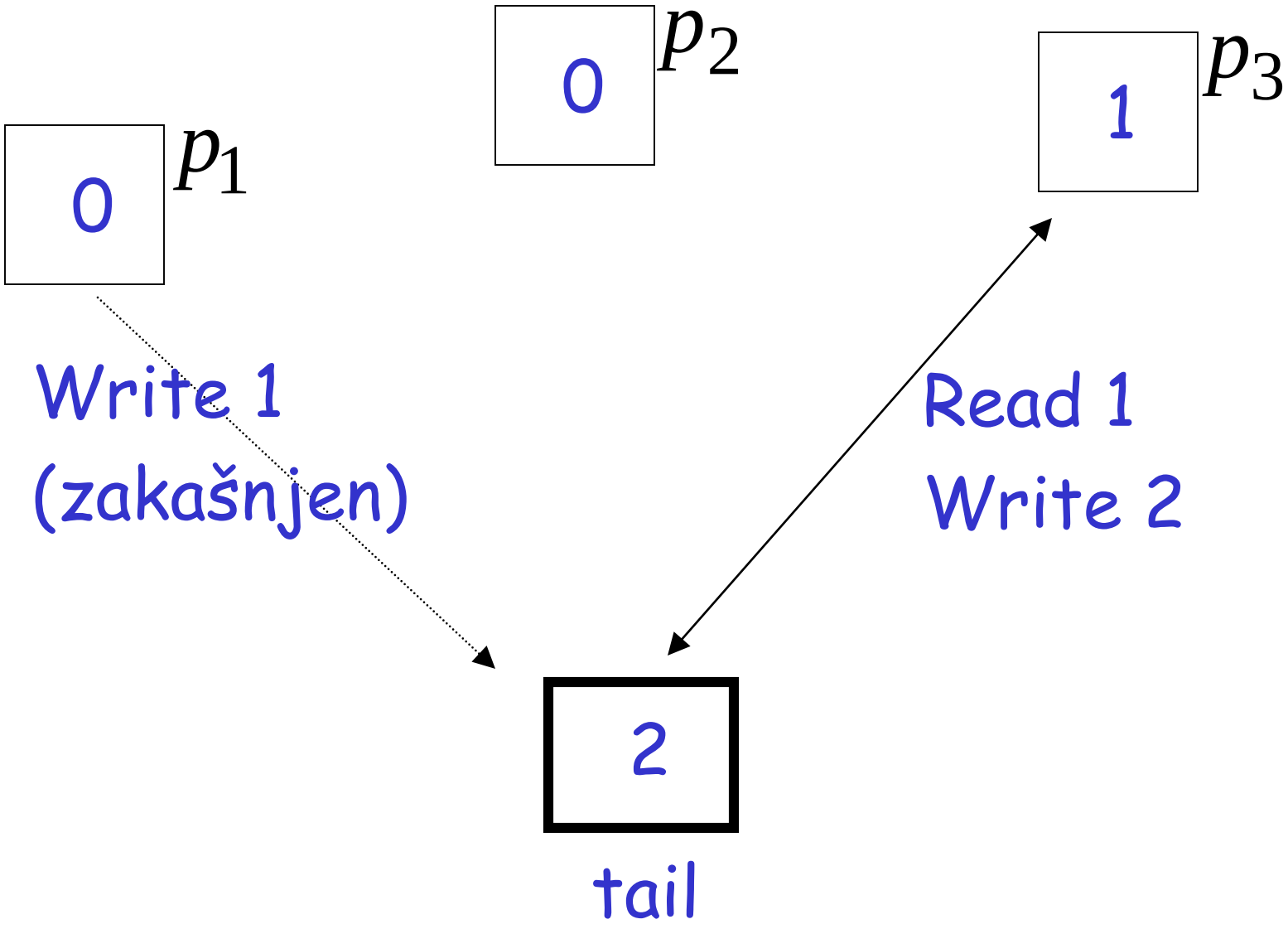
Loš scenario



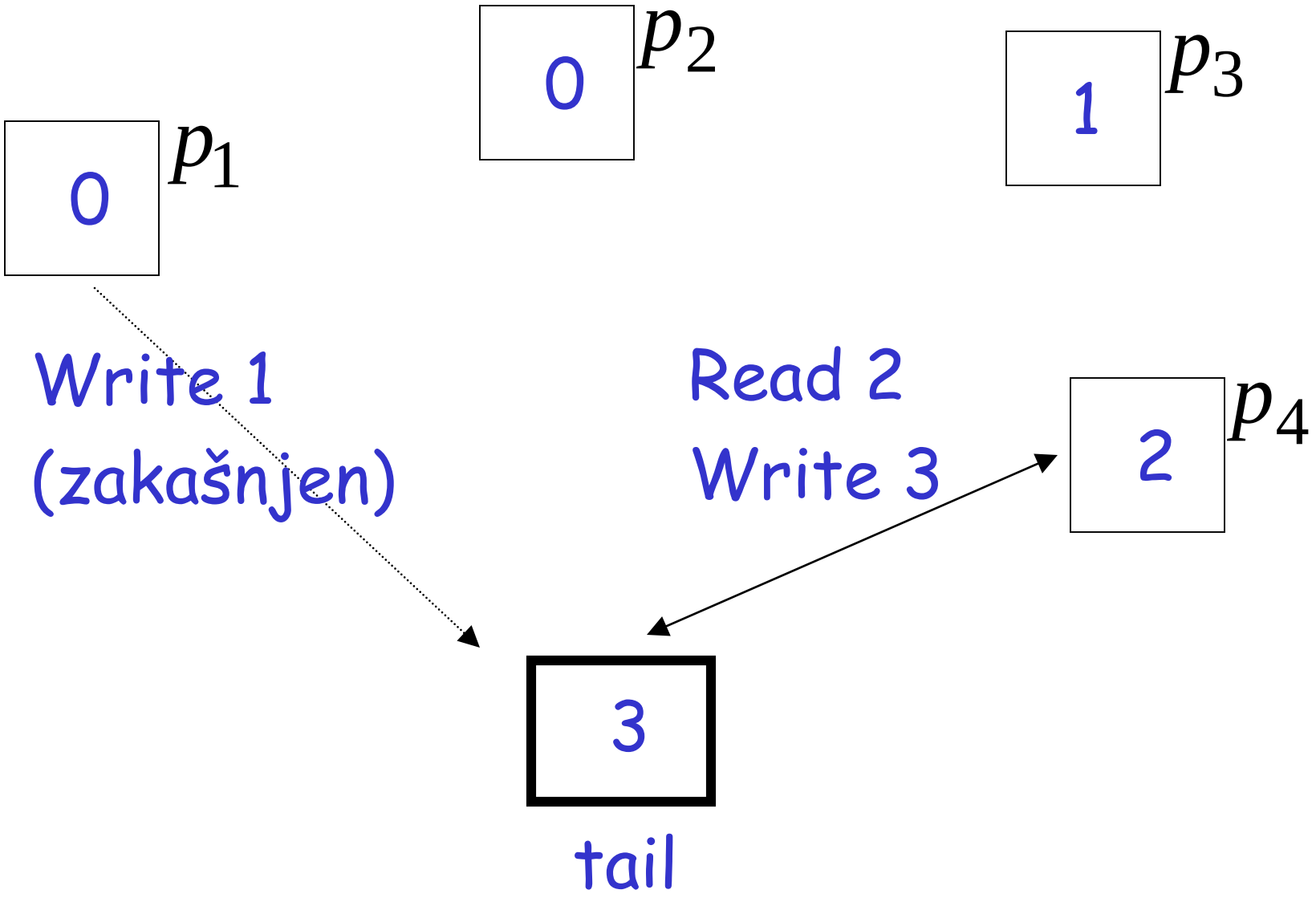
Loš scenario



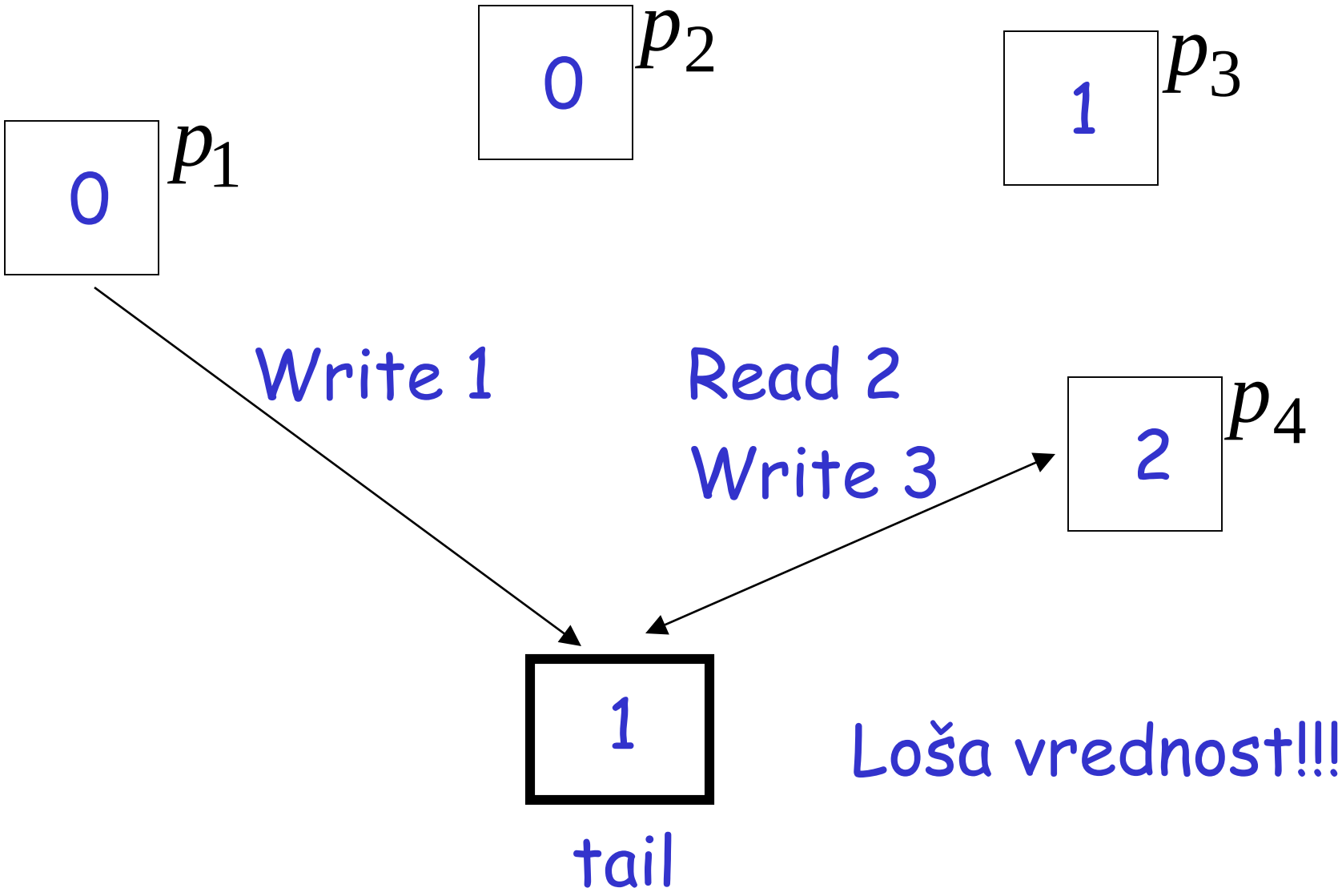
Loš scenario



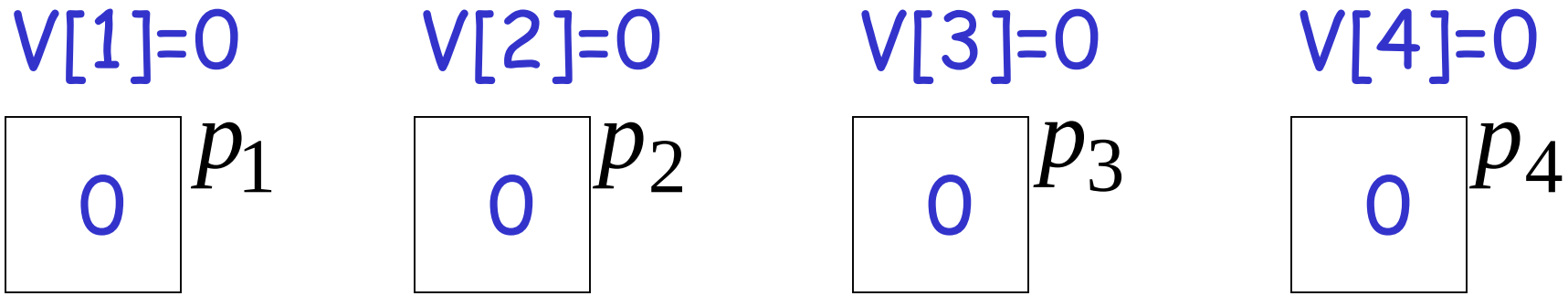
Loš scenario



Loš scenario

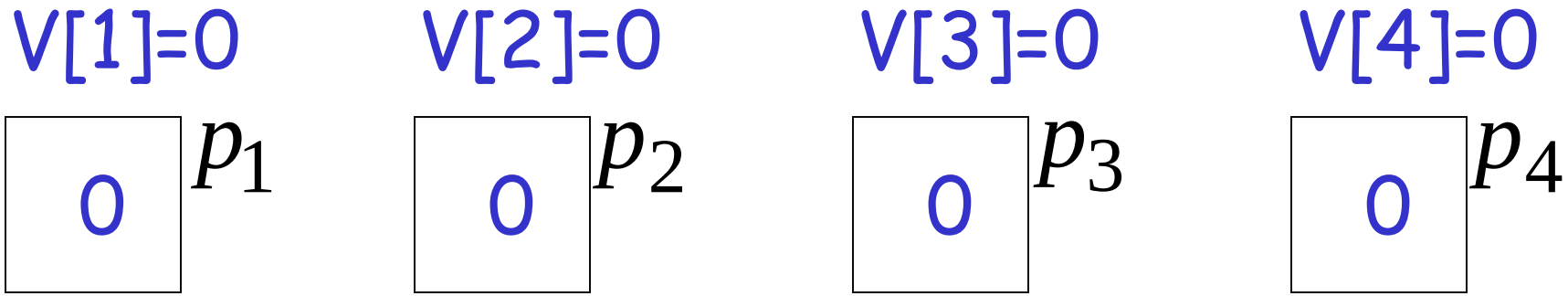


Rešenje: distribuirano brojanje



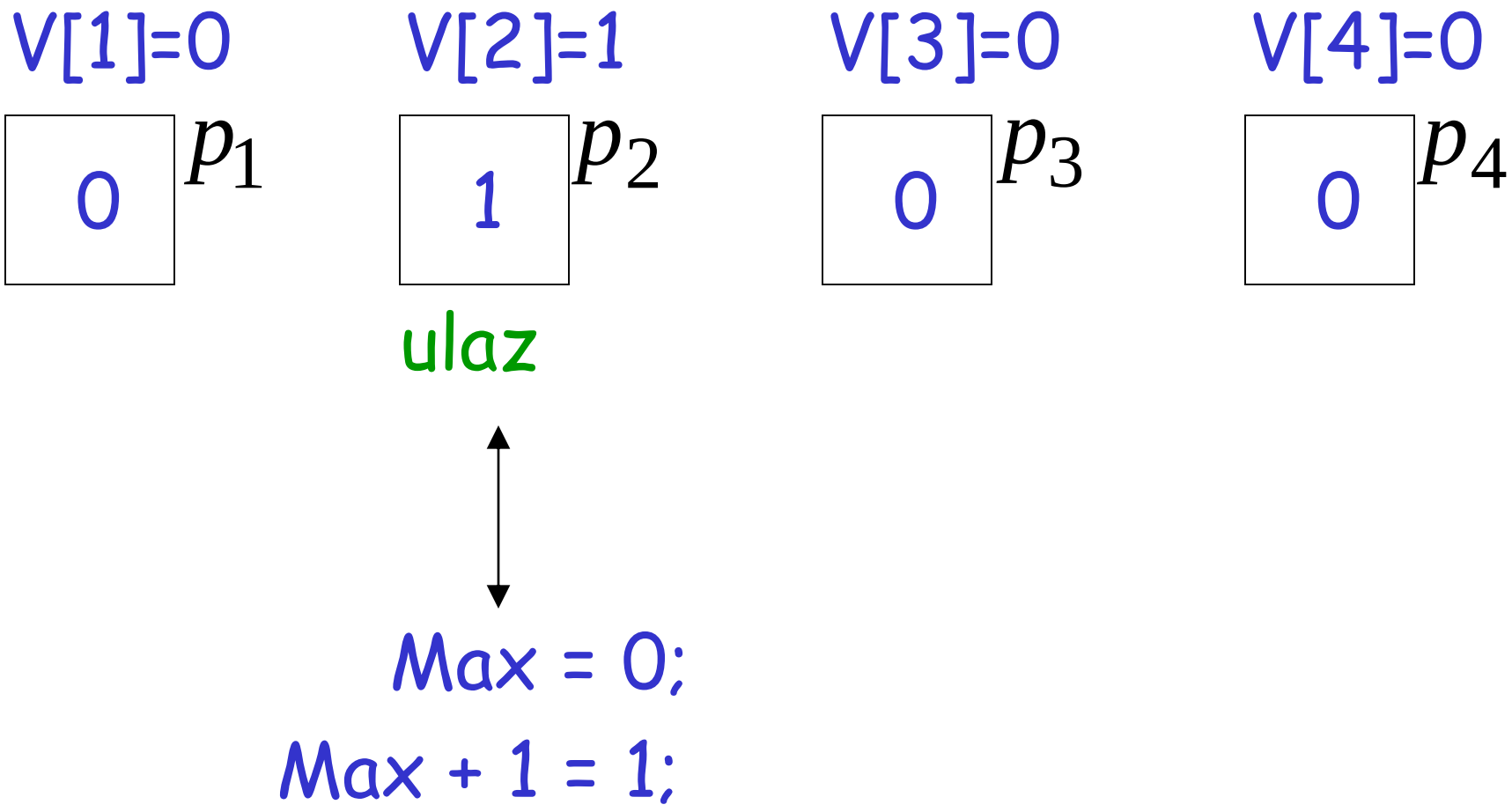
Treba nam niz deljenih promenljivih
v[1], v[2], …, v[n]

Proces p_i ima vrednost v[i]



U ulaznom kodu, proces
čita vrednosti svih drugih procesa.

Nova vrednost je maximum + 1


V[1]=0
V[2]=1
V[3]=0
V[4]=0
0
p_1
1
p_2
0
p_3
0
p_4
ulaz
Max = 0;
Max + 1 = 1;

V[1]=0

0 p_1

V[2]=1

1 p_2

ulaz

V[3]=0

0 p_3

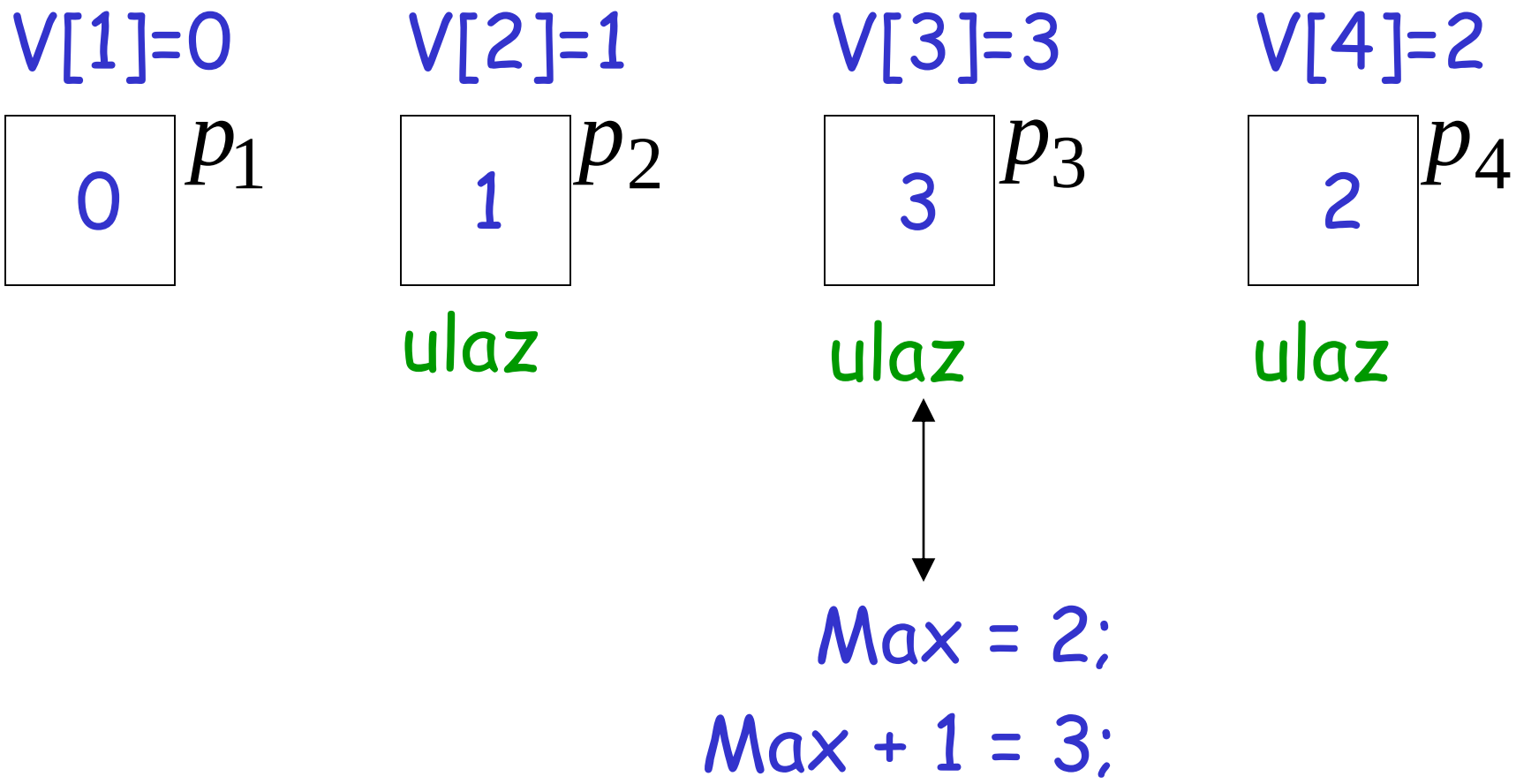
V[4]=2

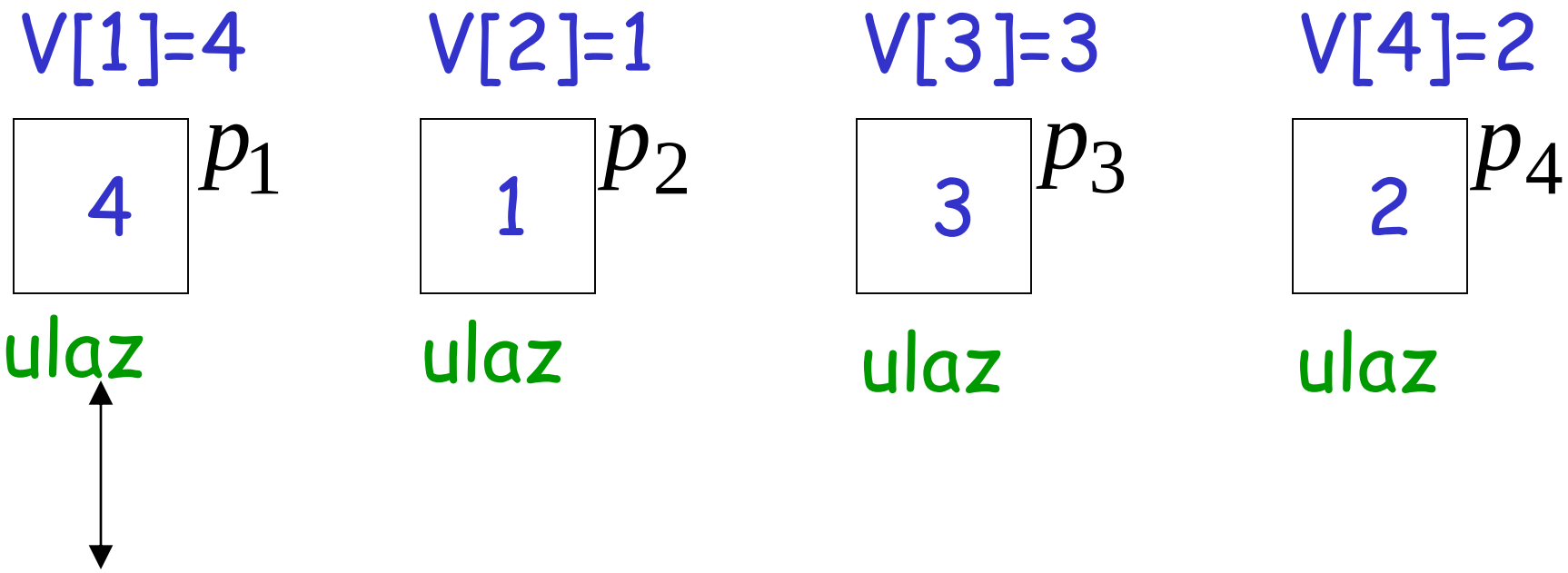
2 p_4

ulaz

Max = 1;

Max + 1 = 2;

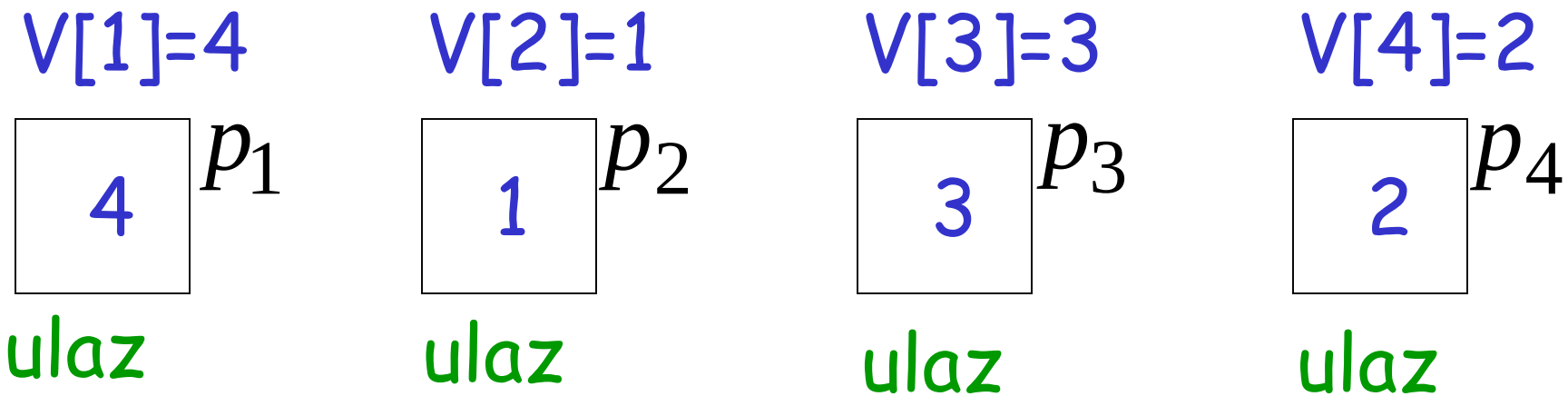

V[1]=0
0
p_1
V[2]=1
1
p_2
ulaz
V[3]=3
3
p_3
ulaz
V[4]=2
2
p_4
ulaz
Max = 2;
Max + 1 = 3;



Max = 3;

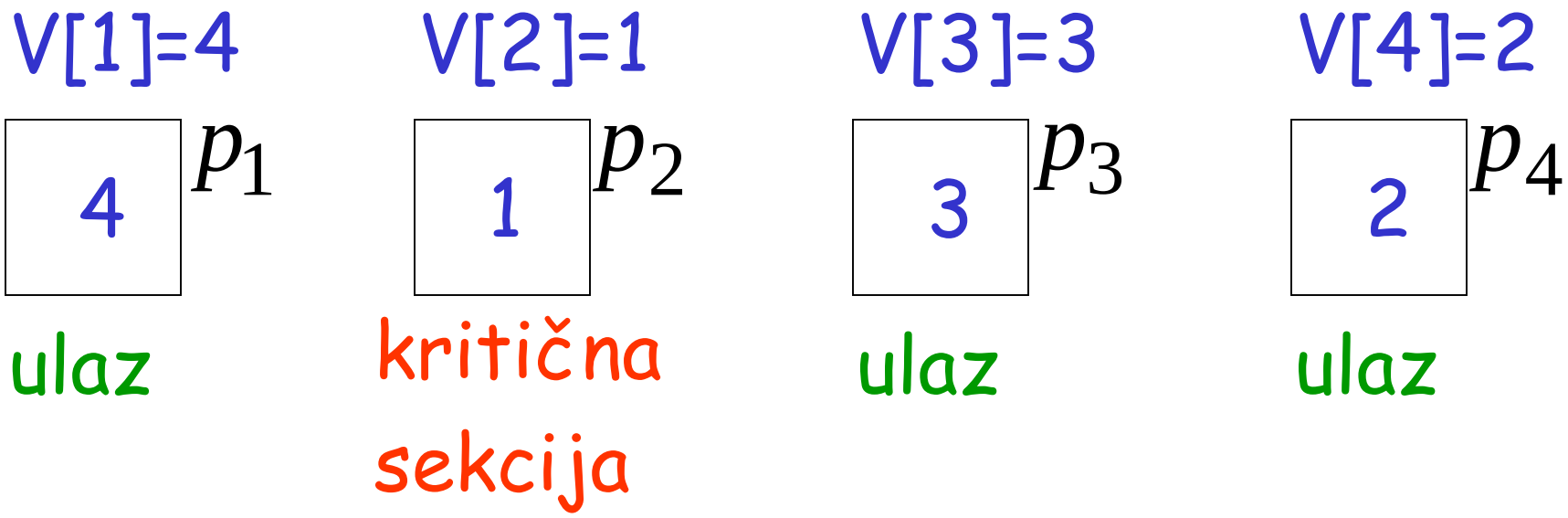
Max + 1 = 4;

Svaki proces dobija jedinstvenu vrednost (jedinstvena pozicija u distribuiranom redu)



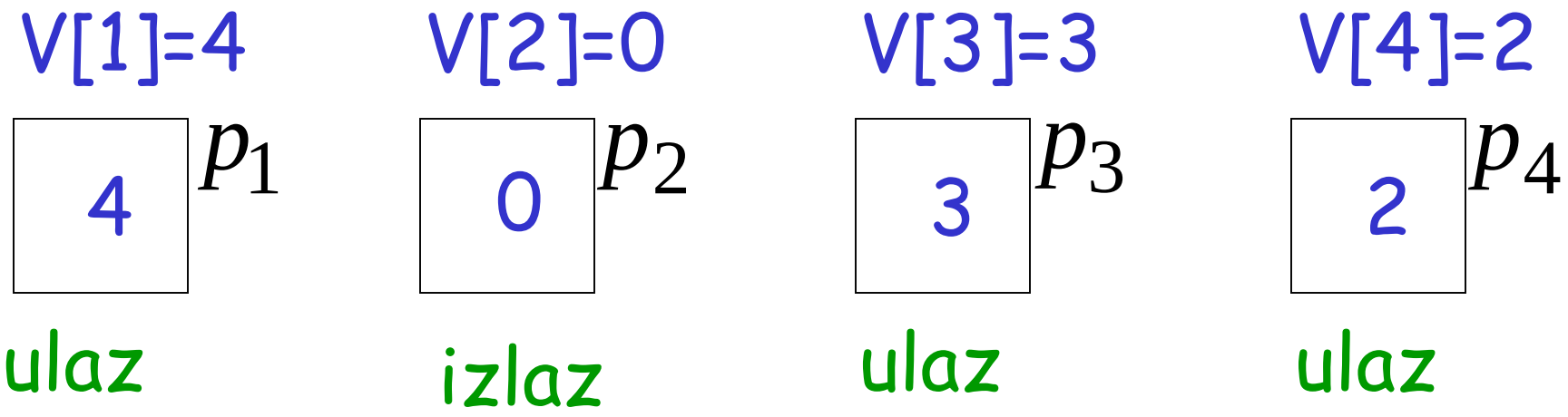
Zatim procesi porede svoje vrednosti sa svim drugim vrednostima.

Najmanja vrednost ulazi u kritičnu sekciju
(različita od 0)

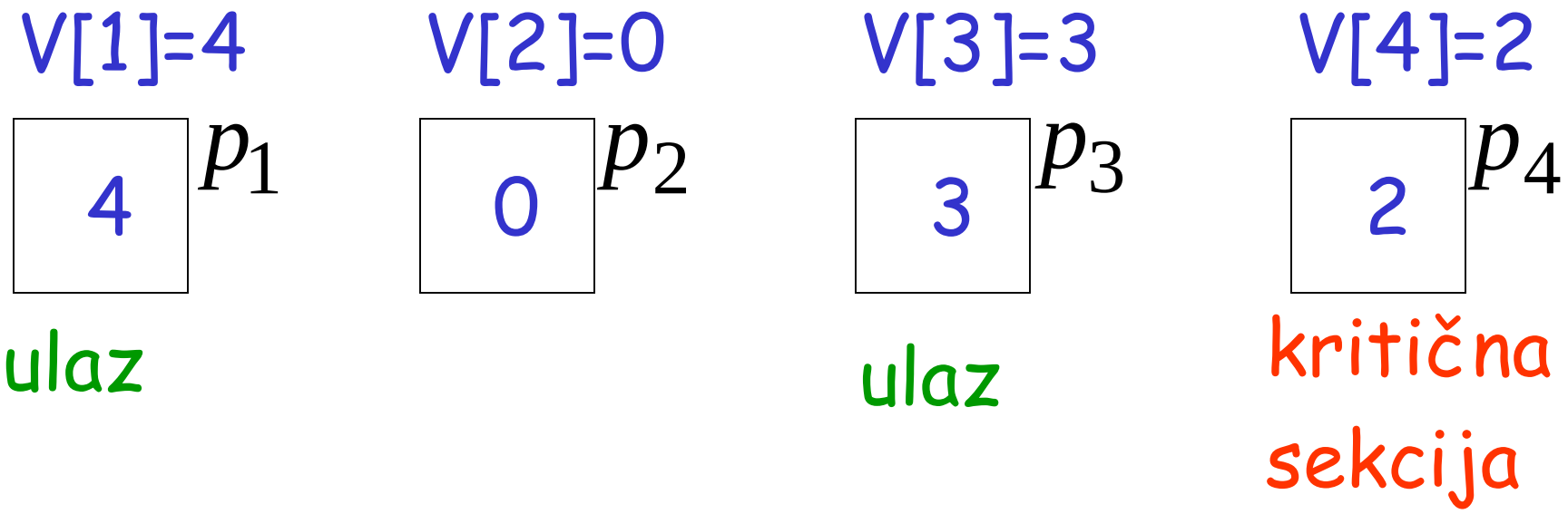


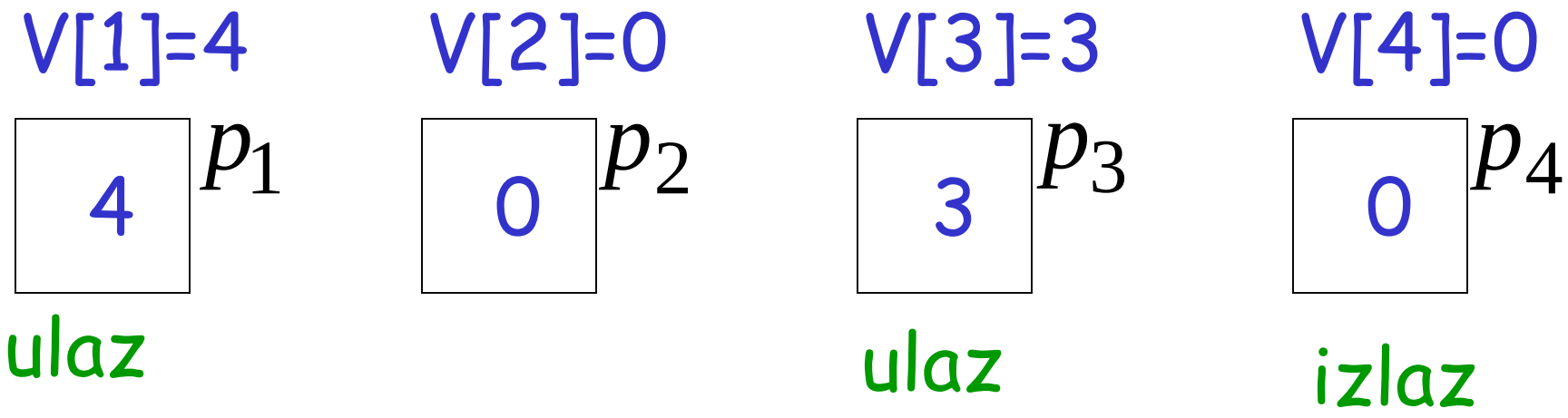
p_2 čita sve vrednosti

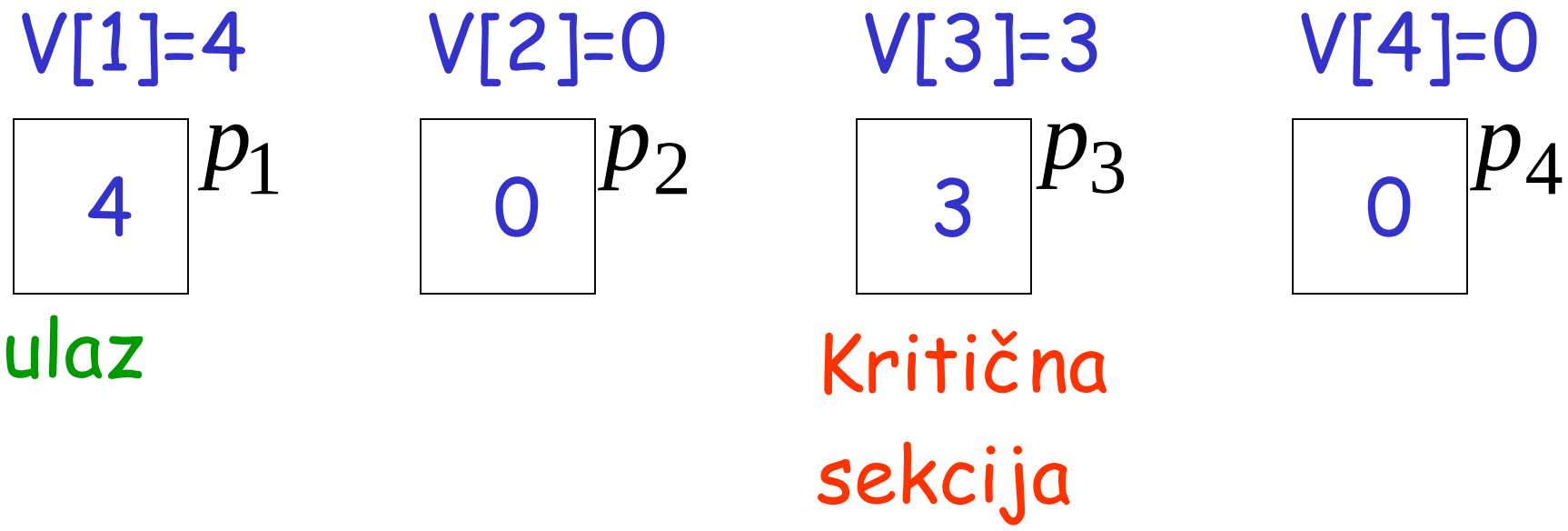
p_2 shvata da ima najmanju vrednost



p_2 postavlja svoju prom na 0

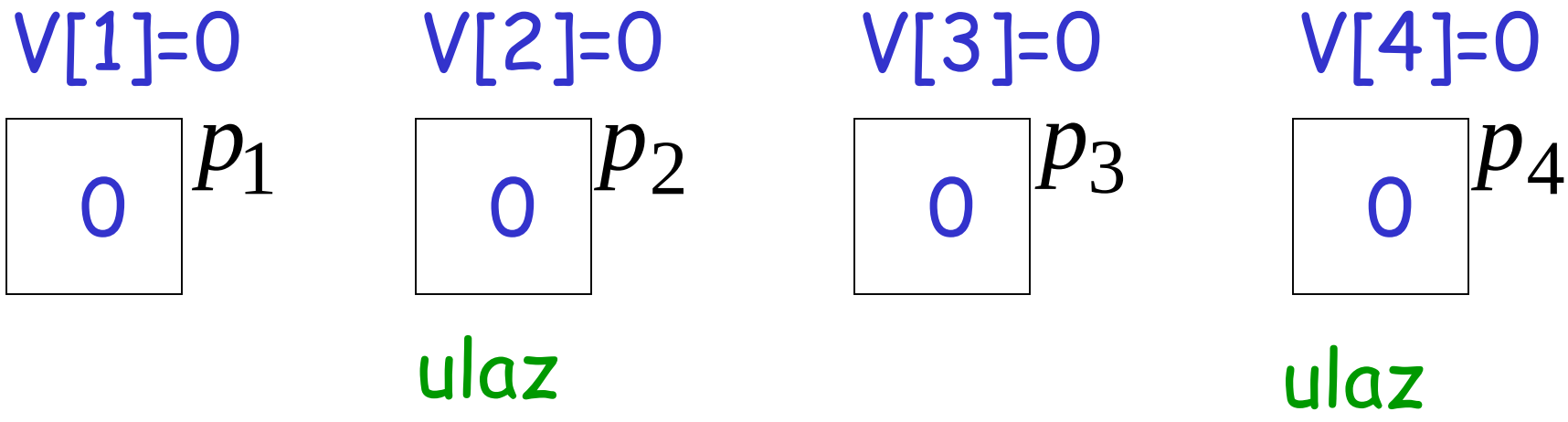

V[1]=4
4
p_1
ulaz
V[2]=0
0
p_2
V[3]=3
3
p_3
ulaz
V[4]=2
2
p_4
kritična
sekcija


V[1]=4
V[2]=0
V[3]=3
V[4]=0
4
p_1
0
p_2
3
p_3
0
p_4
ulaz
ulaz
izlaz

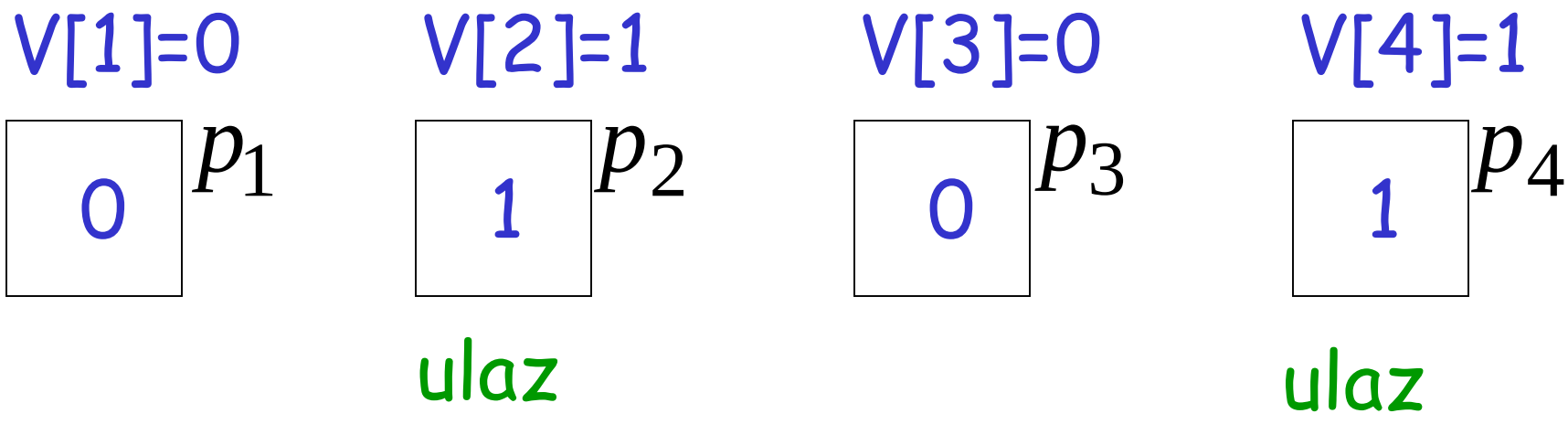


I tako dalje……

Problem: Kada dva procesa uđu
u istom trenutku,
oni mogu da izaberu istu vred.

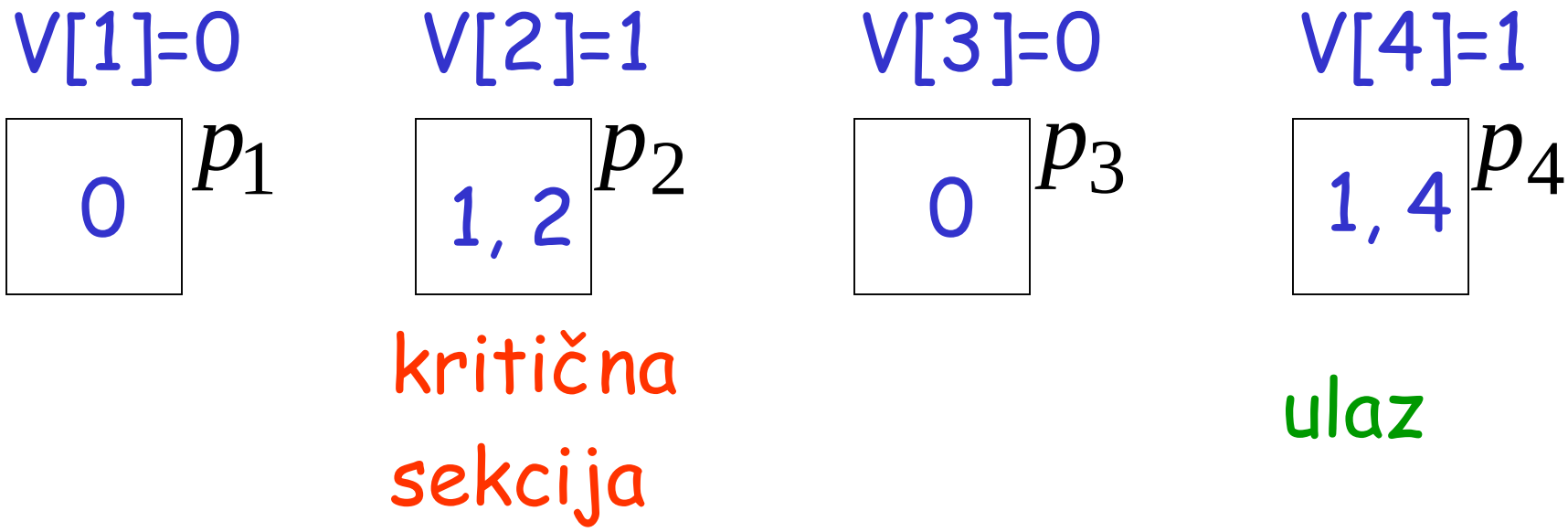


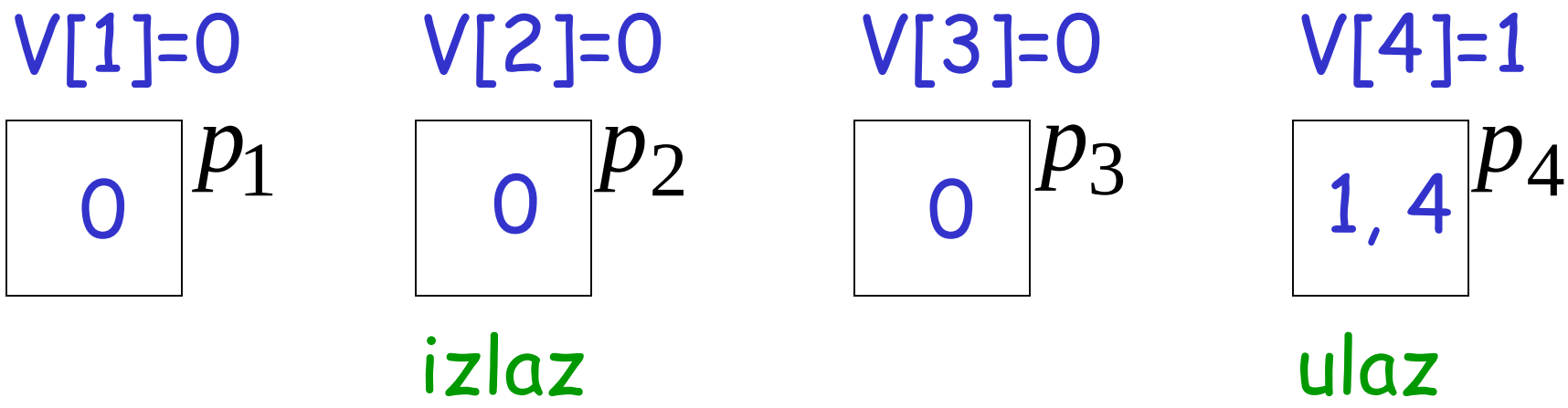
Max vrednosti koje oni pročitaju su iste



Rešenje: koristiti ID za razbijanje simetrije

(najmanji ID pobeđuje)




V[1]=0
V[2]=0
V[3]=0
V[4]=1
0
0
0
1, 4
p_1
p_2
p_3
p_4
izlaz
ulaz

Ceo algoritam sa ceduljicama

Proces i: V[i] = 0; choosing[i] = false;

Ulaz:
```
choosing[i] = true;
V[i] = max(V[1], V[2], …, V[n])+1;
choosing[i] = false;
for (k = 1; k <= n; k++)
    Wait until choosing[k] == false;
    Wait until V[k] == 0 or
                    (V[k],k) > (V[i],i)
```
Kritična sekcija

Izlaz: V[i] = 0;

Prednosti algoritma sa ceduljicama:

- Koristi Read/Write promenljive
- Zadovoljava uslov da nema trajnog zaključavanja

Nedostaci:

- Koristi n deljenih prom. za n procesa
  (u stvari, ne postoji rešenje sa manje prom.)
- Vrednosti mogu da rastu neograničeno
  (voleli bi da nađemo neki algoritam sa ograničenim vrednostima)

Međusobno isključivanje za 2 procesa

	proces visokog priori.	proces niskog priori.
Ulaz:	Want[0] = 1; Wait until want[1]== 0;	1: Want[1] = 0; Wait until want[0] ==0; Want[1] = 1; if (want[0] == 1) goto 1;
	Kritična sekcija	Kritična sekcija
Izlaz:	Want[0] = 0;	Want[1] = 0;

Dobro: Koristi samo ograničene vred. prom.

Problem: Proces niskog prio. može se traj.zak.

Algoritam istog prioriteta

	Proces 0	Proces 1

Ulaz:	1: Want[0] = 0;
	Wait until (want[1]== 0
	or Priority == 0);
	Want[0] = 1;
	if priority == 1 then
	if Want[1] == 1 then
	goto Line 1
	Else wait until Want[1]==0;
	Critical section
		1: Want[1] = 0;
		Wait until (want[0]== 0
		or Priority == 1);
		Want[1] = 1;
		if priority == 0 then
		if Want[0] == 1 then
		goto Line 1
		Else wait until Want[0]==0;
		Critical section

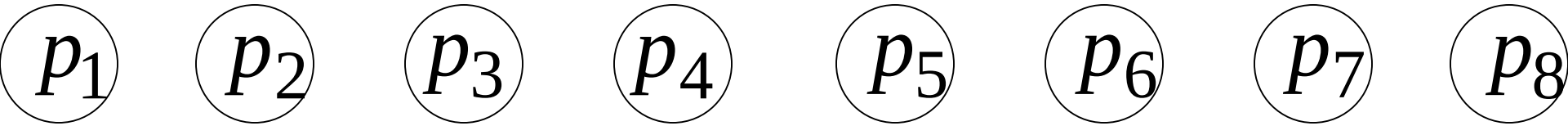
Izlaz:	Priority = 1; Want[0] = 0;	Priority = 0; Want[1] = 0;

Dobro: Koristi ograničene vred. prom.

Nema trajnog zaključ. (lockout)

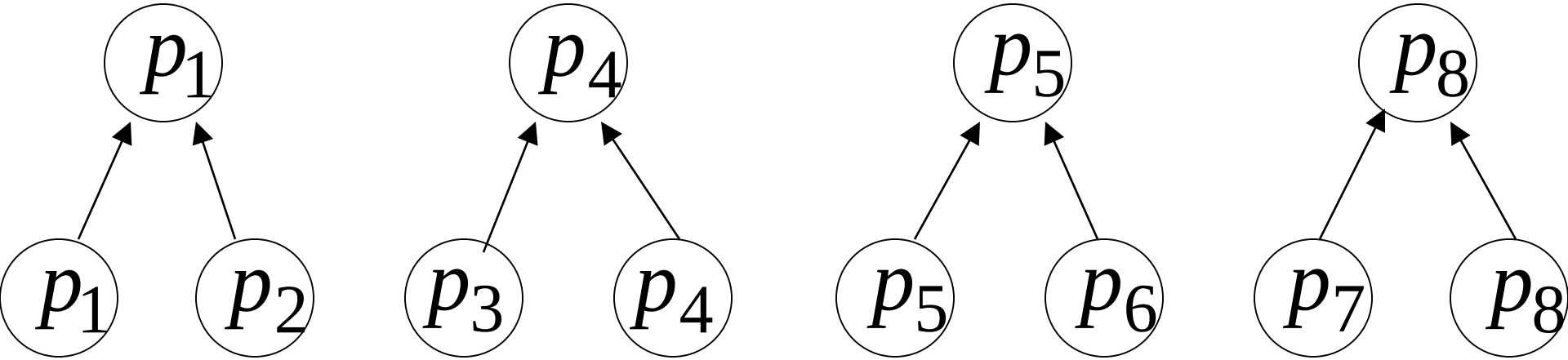
Algoritam sa turnirom (Tournament)

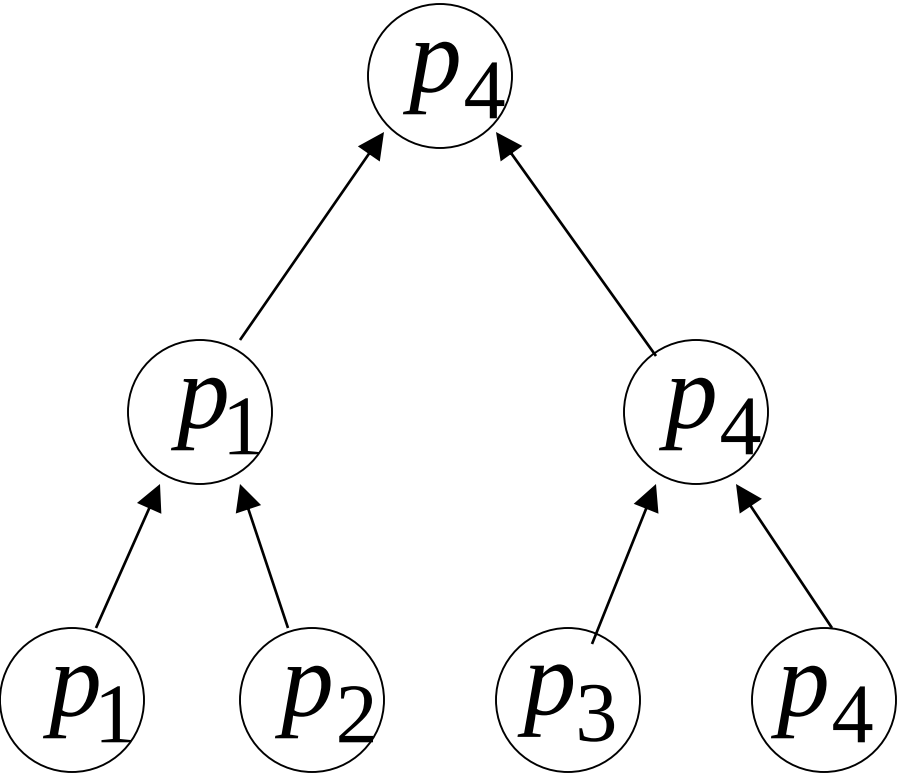
Možemo realizovati algoritam turnira za međusobno isključivanje, koristeći algorit. jednakog prioriteta po parovima

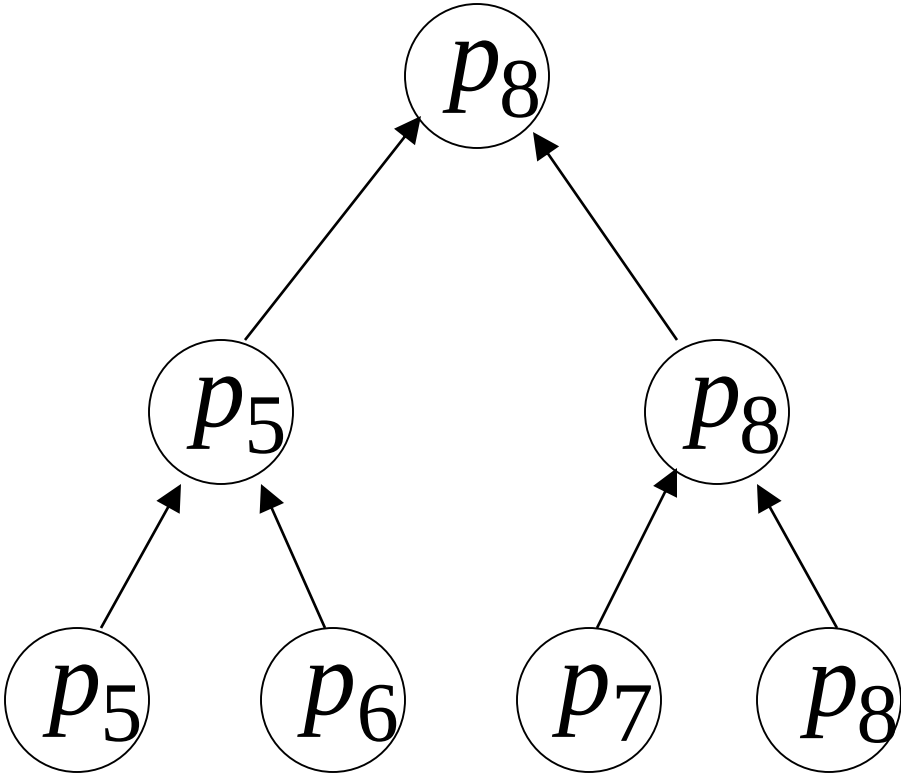


Međusobno isključivanje za parove procesa

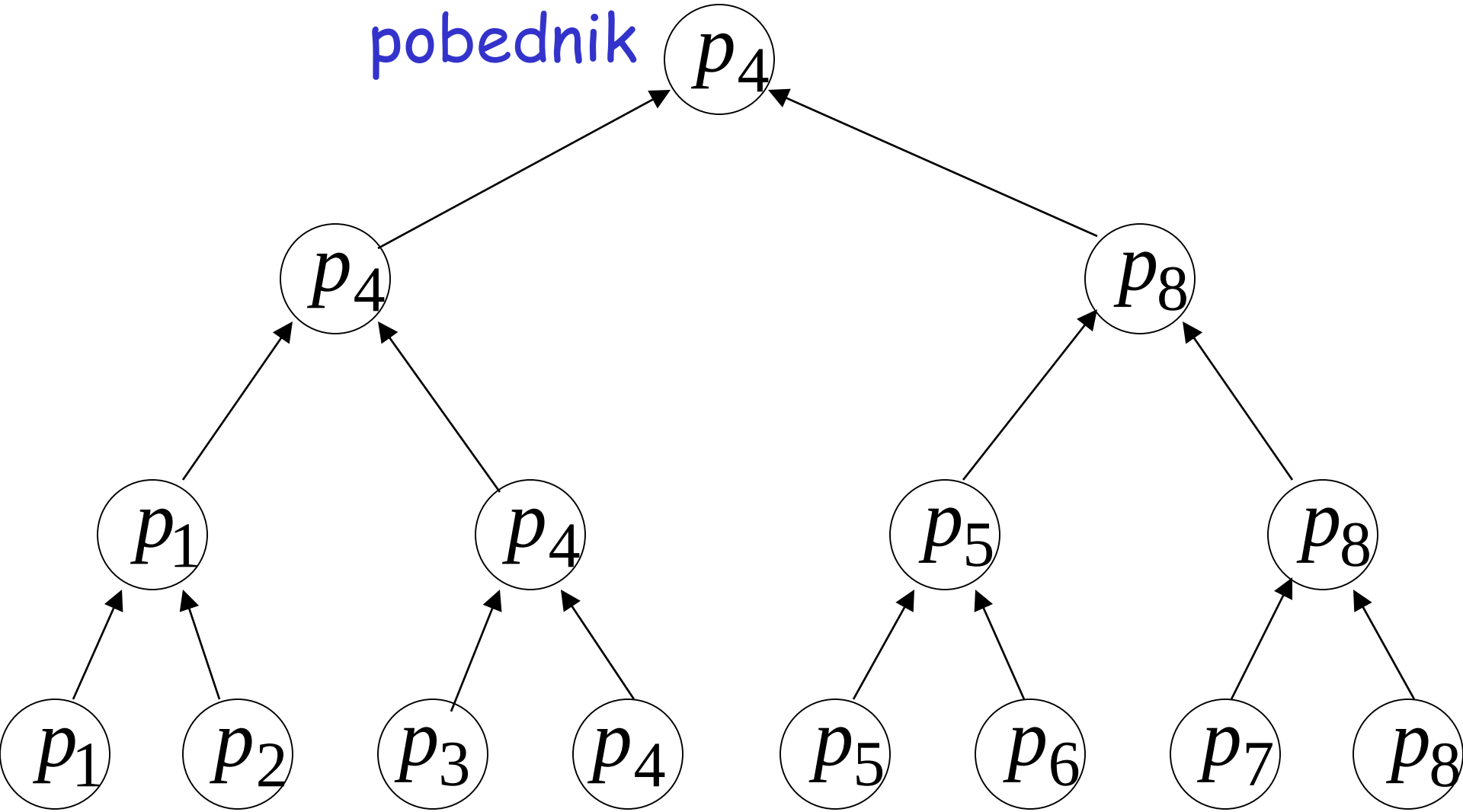
pobednik




p_4
p_1
p_4
p_1
p_2
p_3
p_4


p_8
p_5
p_8
p_5
p_6
p_7
p_8

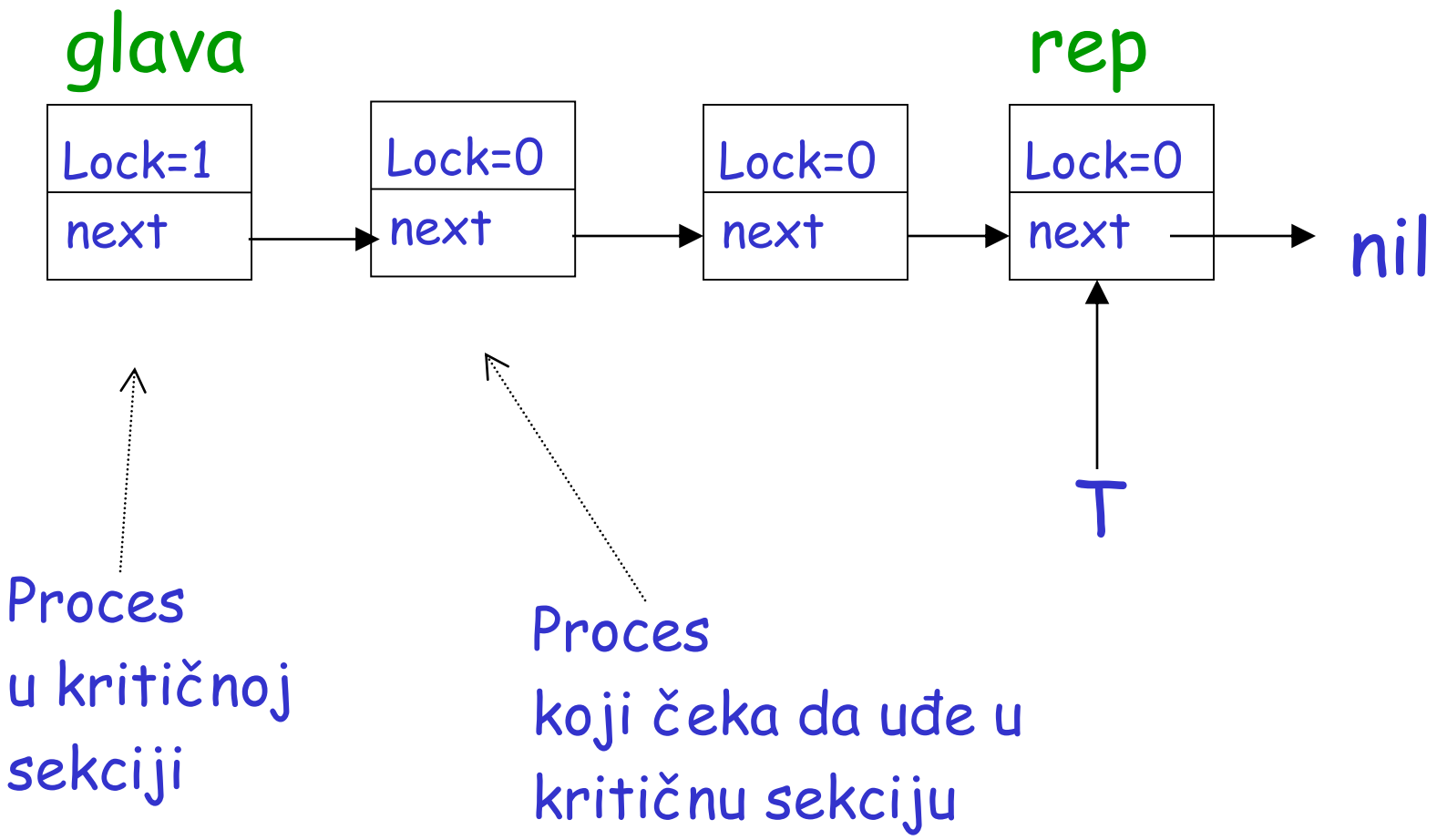
Kritična sekcija

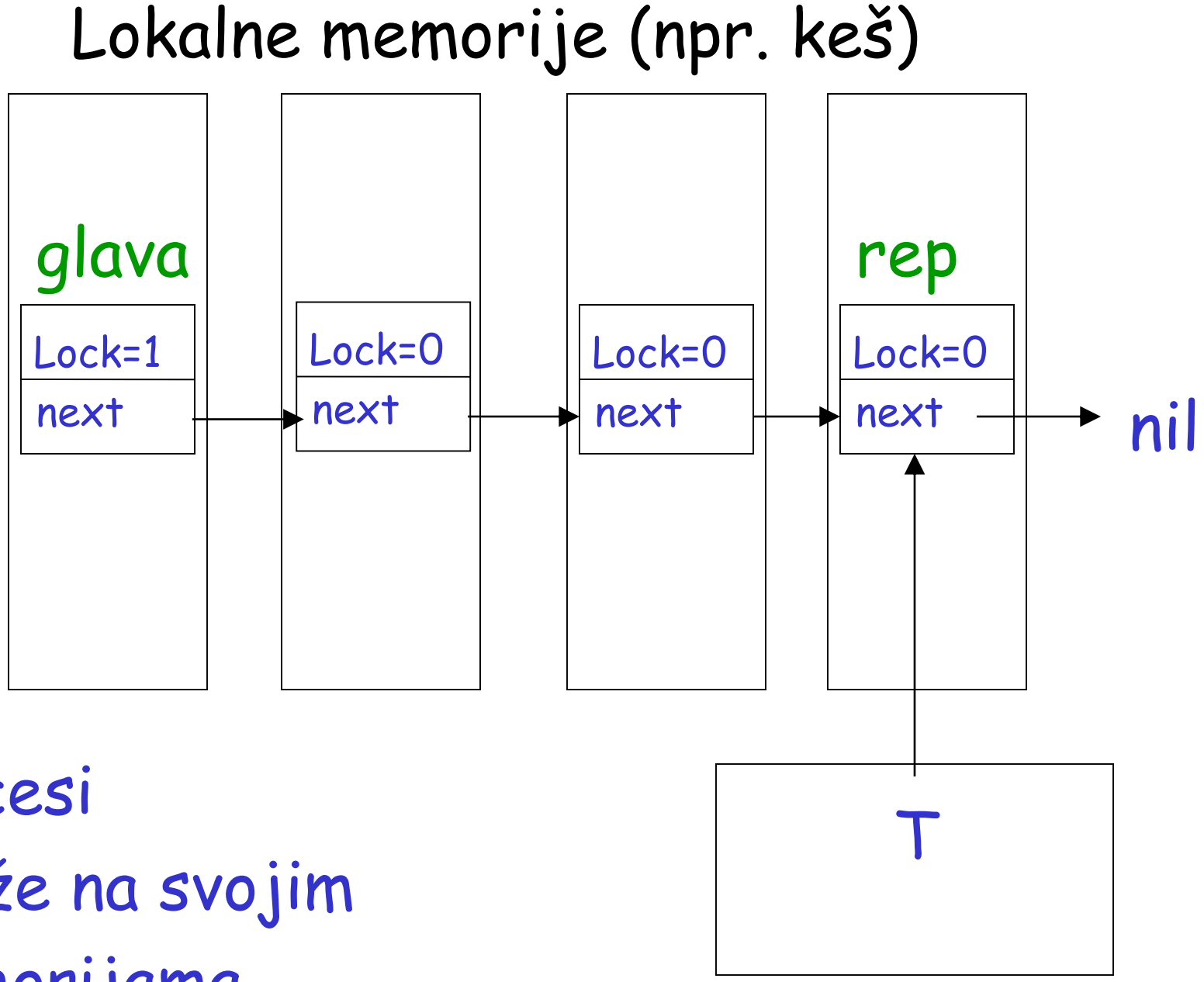


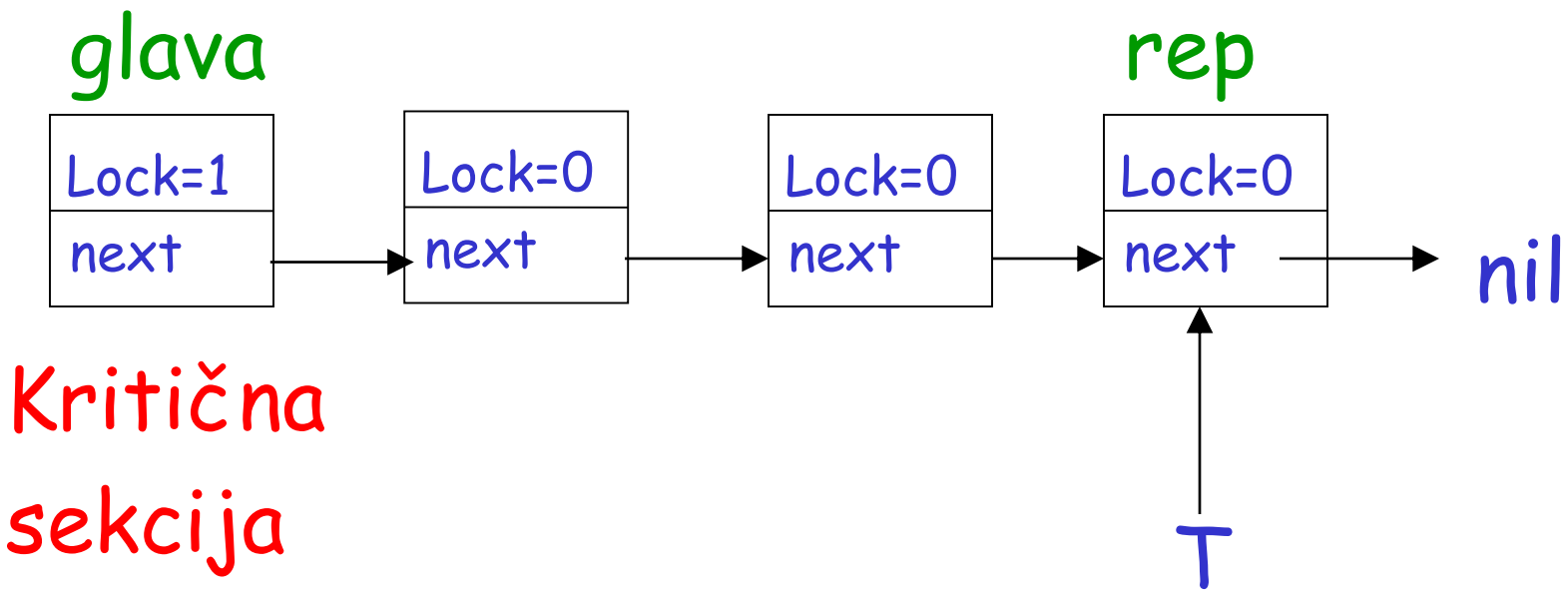
Prednosti algoritma sa turnirom

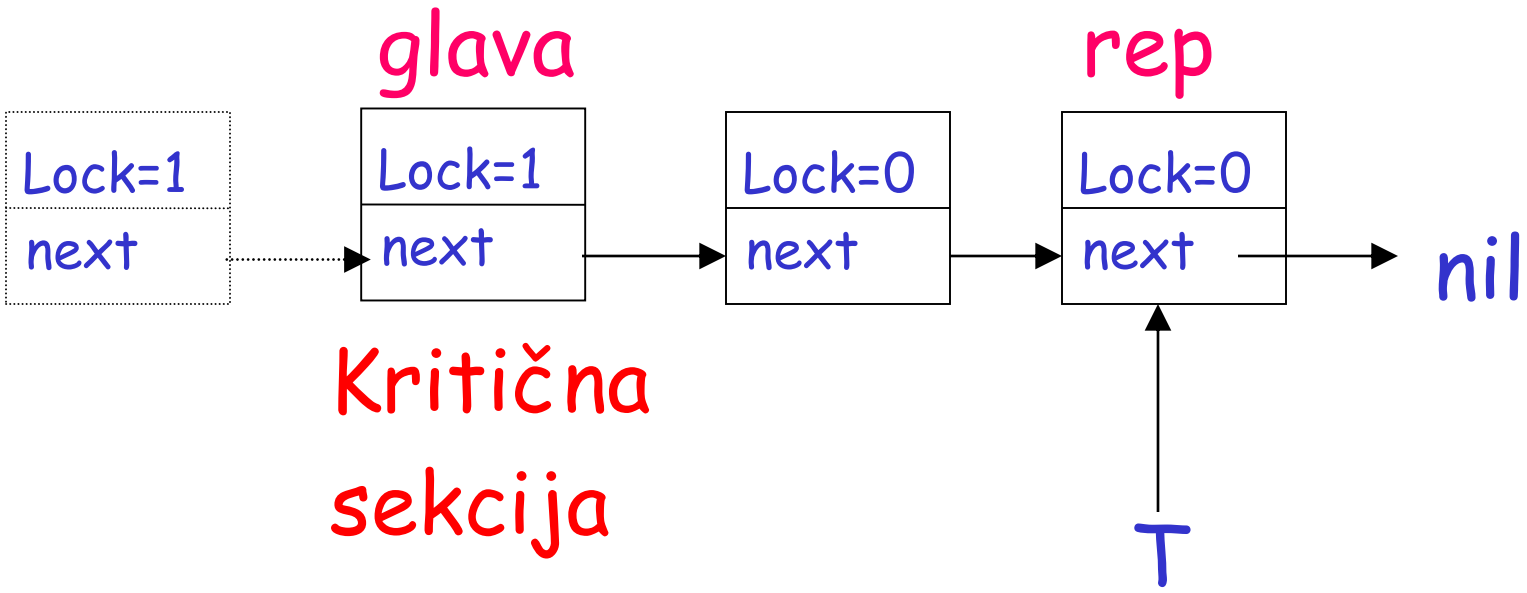
- O(n) promenljivih
- Ograničene vred. promenljivih
- Nema trajnog zaključavanja
  (pošto za međuso. isključivanje
  para nema trajnog zaključavanja)

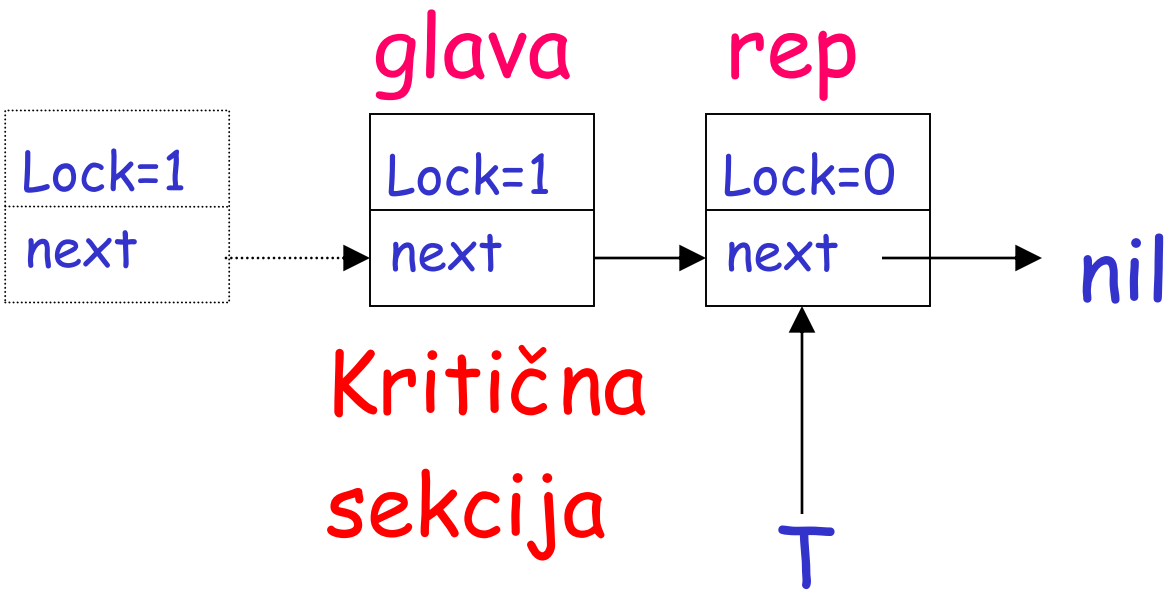
MCS algoritam međusobnog isključivanja

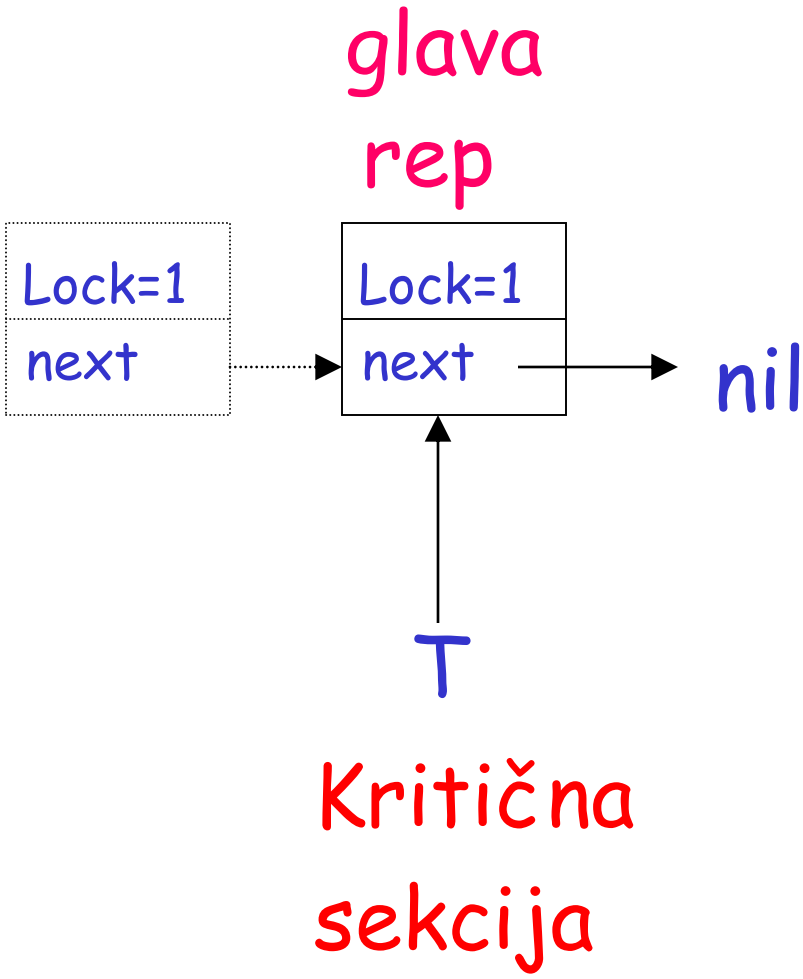



Lokalne memorije (npr. keš)
glava
rep
Lock=1
next
Lock=0
next
Lock=0
next
Lock=0
next
nil
T
Procesi
Kruže na svojim
memorijama
Globalna deljena mem.


glava
rep
Lock=1
next
Lock=0
next
Lock=0
next
Lock=0
next
nil
Kritična
sekcija
T


glava
rep
Lock=1
next
Lock=1
next
Lock=0
next
Lock=0
next
nil
Kritična
sekcija
T


glava
rep
Lock=1
next
Lock=1
next
Lock=0
next
nil
Kritična
sekcija
T


glava
rep
Lock=1
next
Lock=1
next
nil
T
Kritična
sekcija

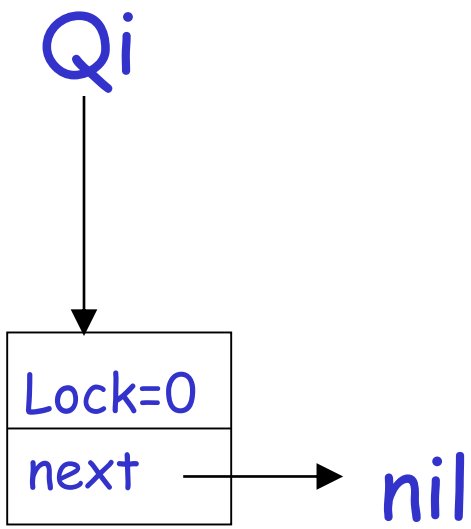

nil
T

Ulazni kod za procesor i:

```
Qi = pokaz. na nov queuenode; // Qi, *Qi je u lokalnoj mem.
Qi->Lock = 0;
Qi->Next = nil;
Ti = Swap(T, Qi); //Ti je u lokalnoj mem.
If Ti ≠ nil then
    Ti -> next = Qi;
else
    Qi->Lock =1;    //on je na glavi reda
Wait until Qi->lock = 1;
```

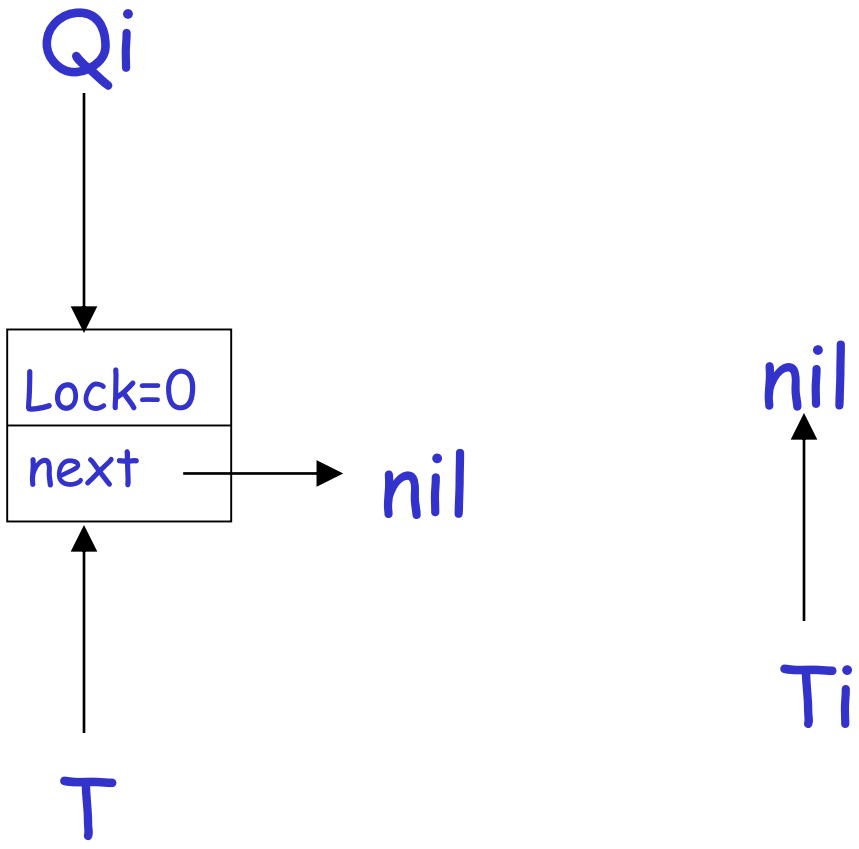
Atomska operacija

```
Swap(T,Qi)
{ x = T;          //čitaj T
  T = Qi;         // zameni T i Qi
   return x;      // vrati staru vred. od T
}
```

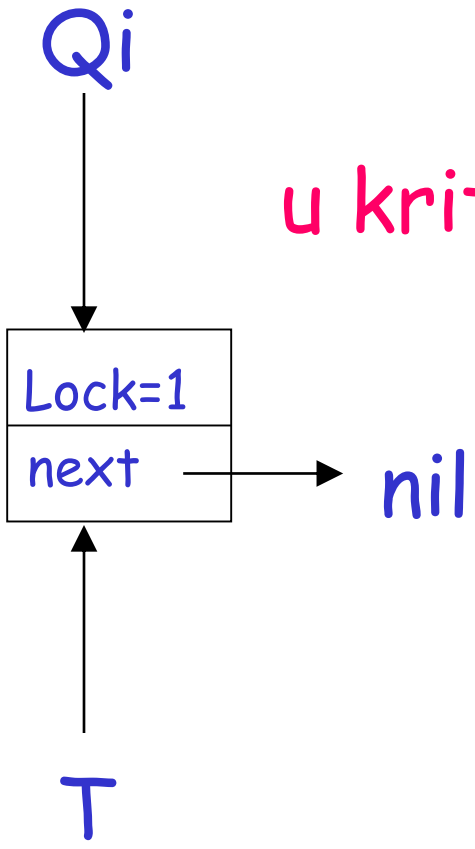


Prazan red

Qi je napravljen

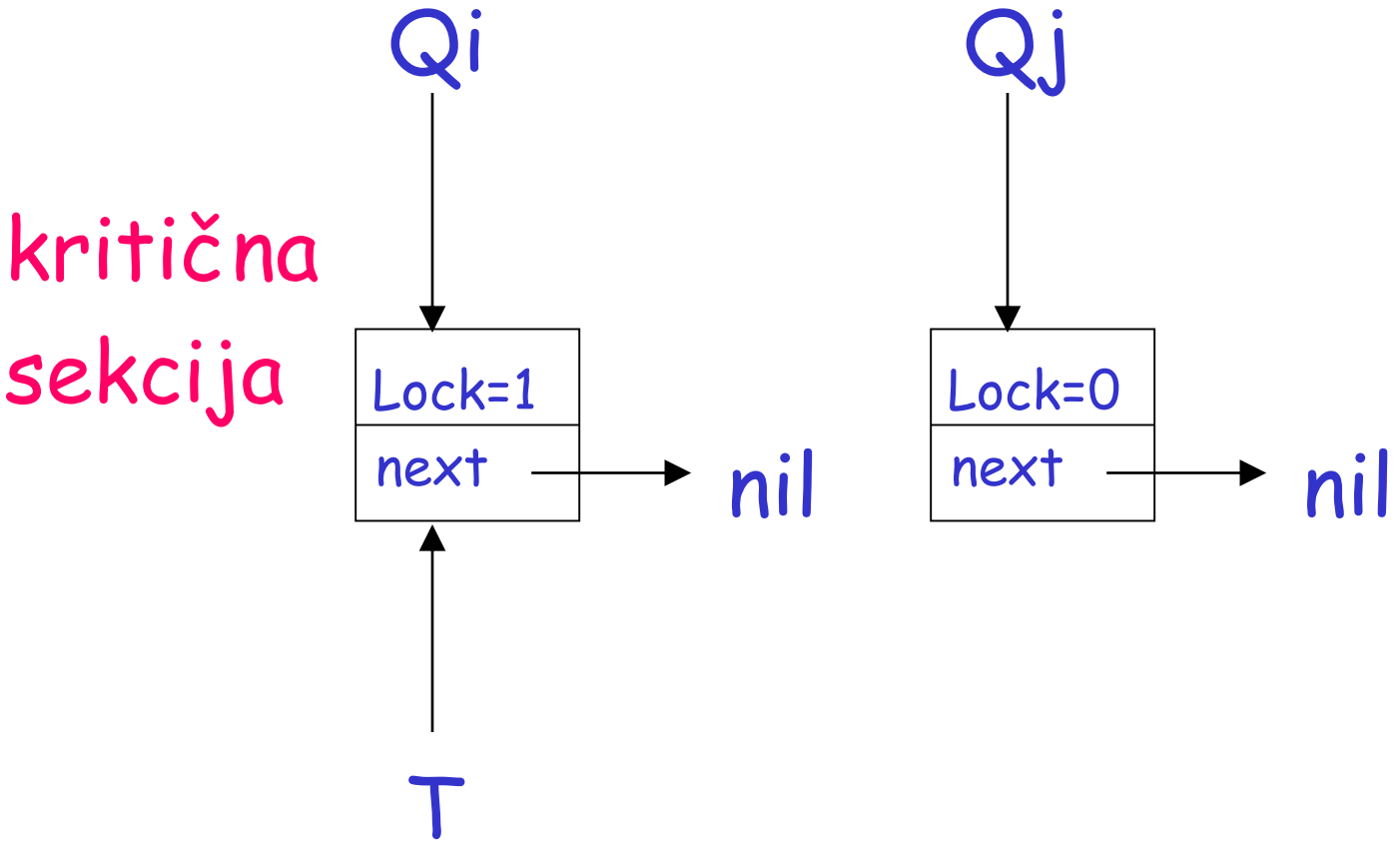


Posle operacije zamene (swap)

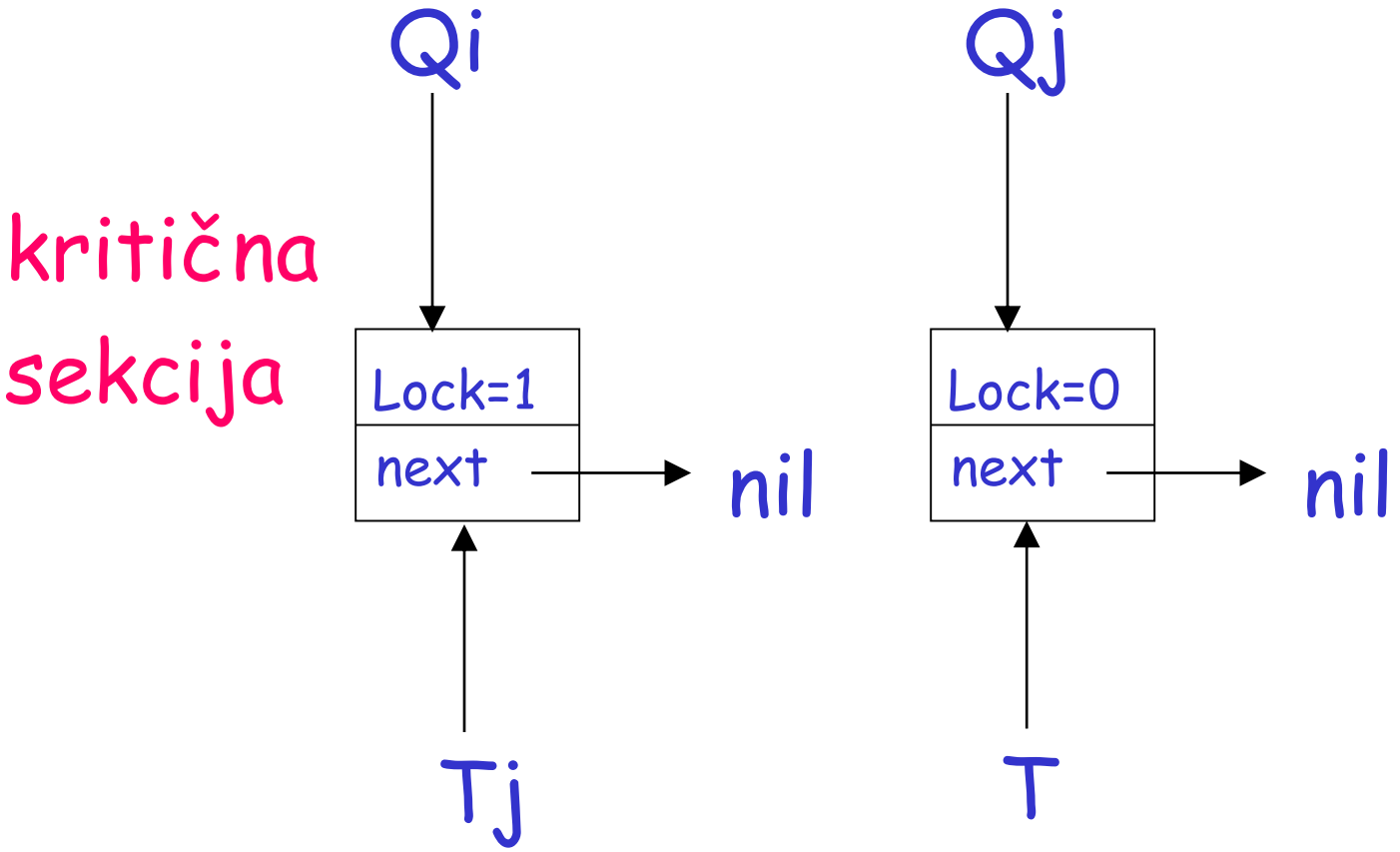


u kritičnoj sekciji

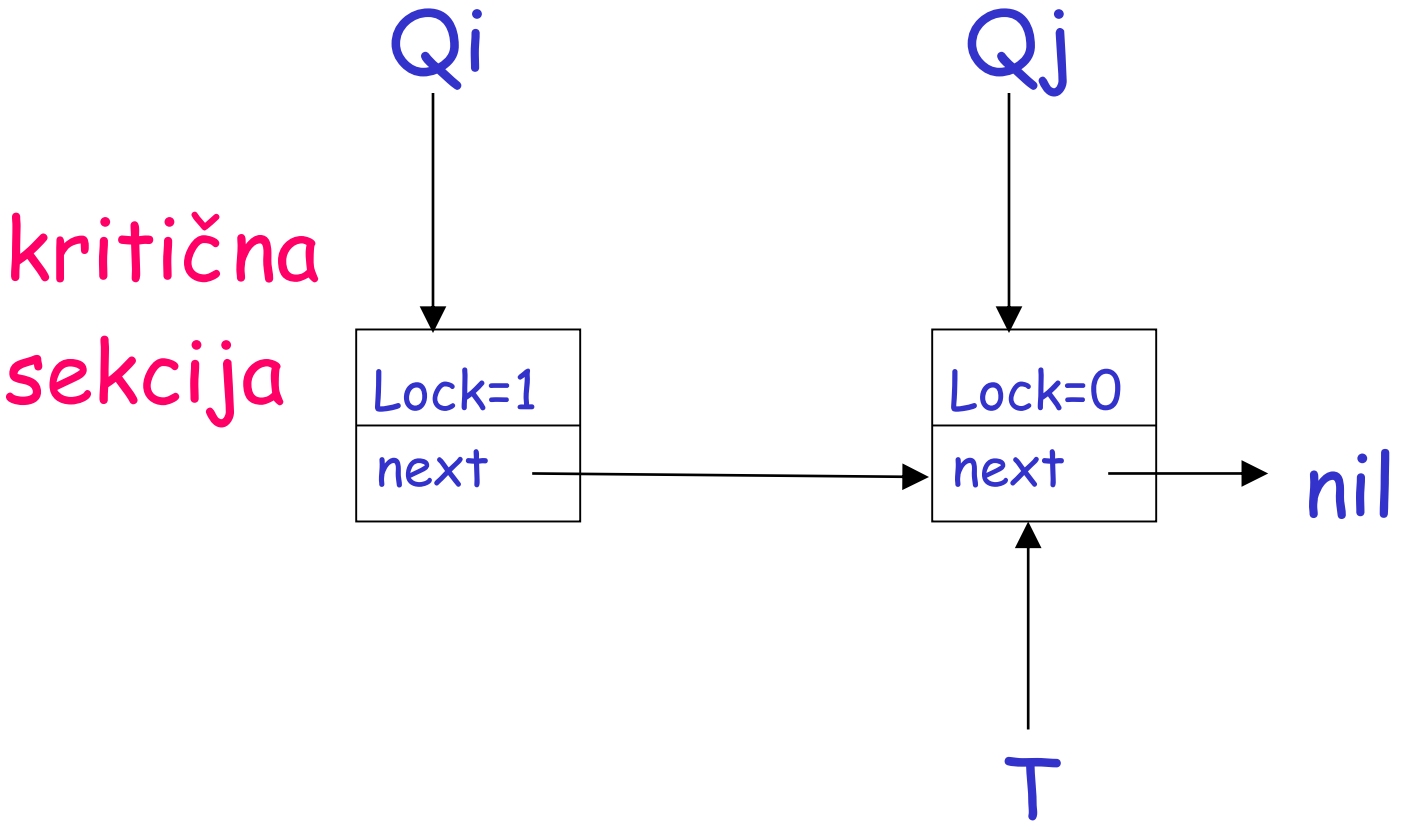
Posle iskaza if



Stiže proces j

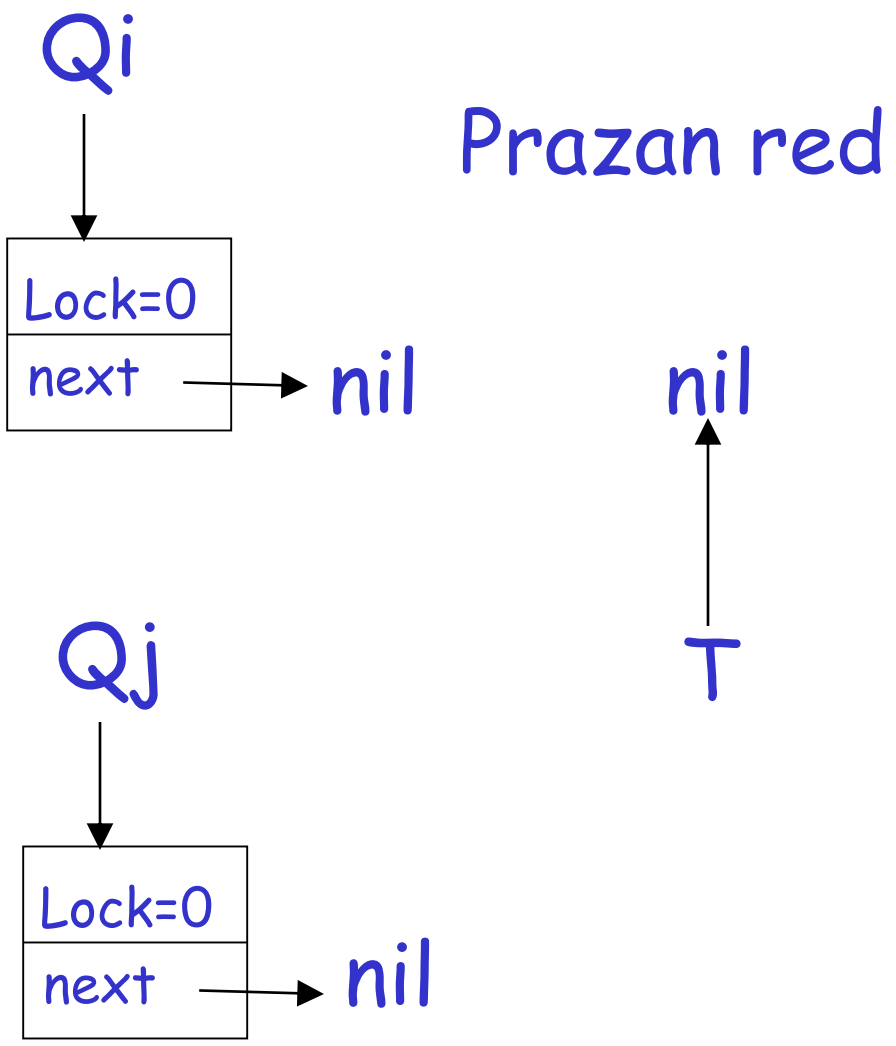


Posle operacije zamene

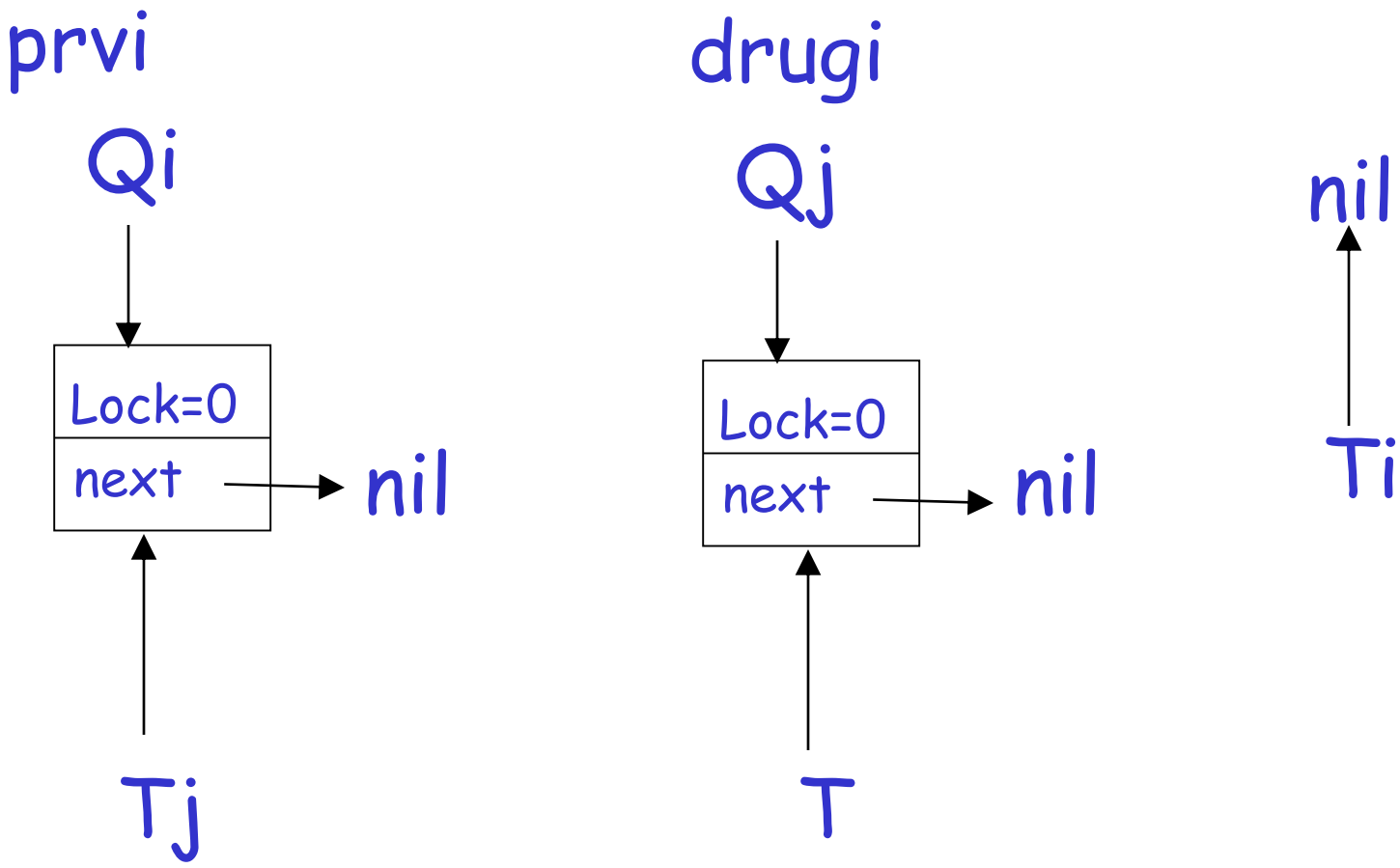


Posle operacije u iskazu if

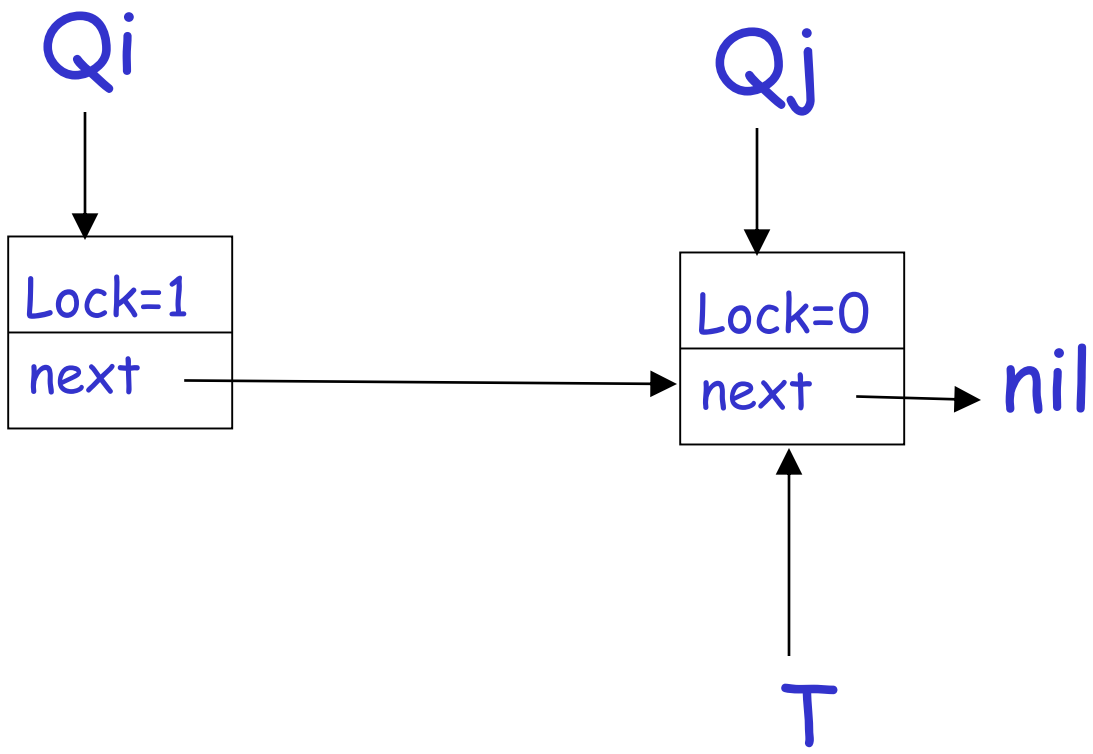
Procesi i i j
stižu simultano



Izvršenje operacije zamene dodeljuje redosled



kritična
sekcija

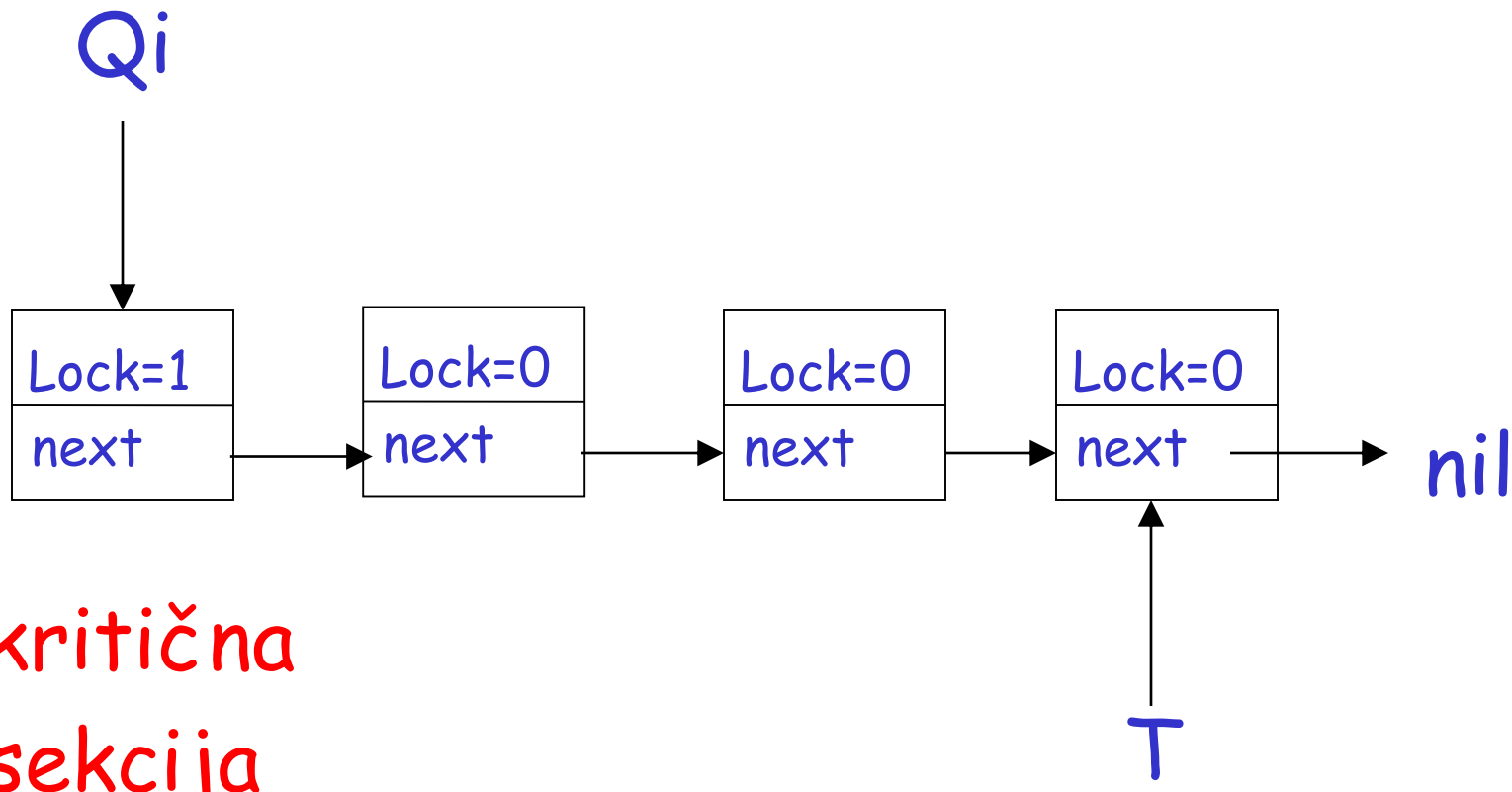


Izlazni kod za procesor i:

```
Ti = Compare&Swap(T, Qi, nil); //Ti je u lokalnoj mem.
If Ti ≠ Qi then
    wait until (Qi->next ≠ nil)
    Ti -> next->lock = 1;
Delete Qi and *Qi
```

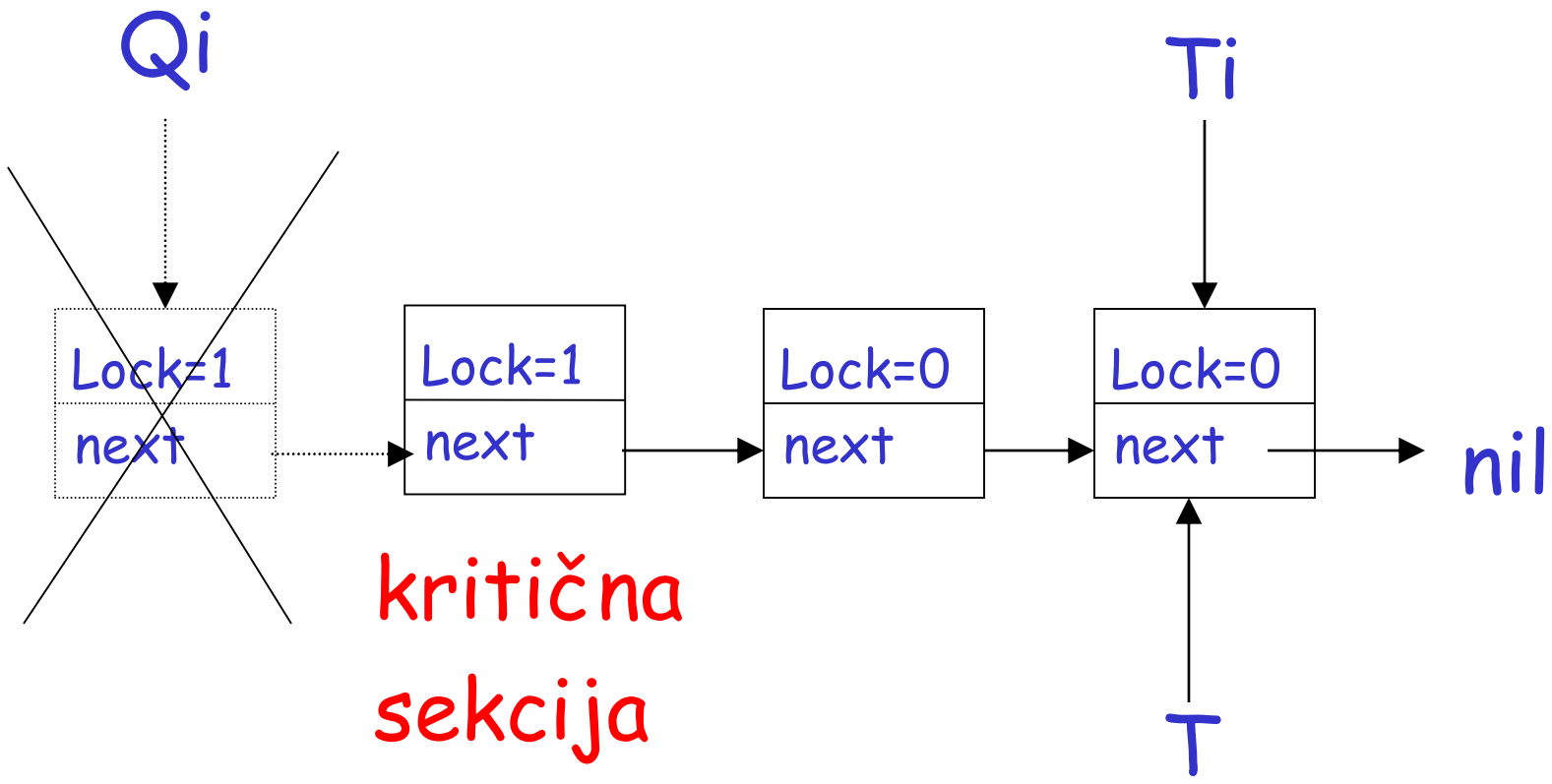
Atomska operacija

```
Compare&Swap(T,Qi,nil)
{ x = T;              //čitaj vrednost T
  If T == Qi then
       T = nil;     //zameni T sa nil aku su T i Qi jednaki
  return x;         // vrati staru vrednost T
}
```

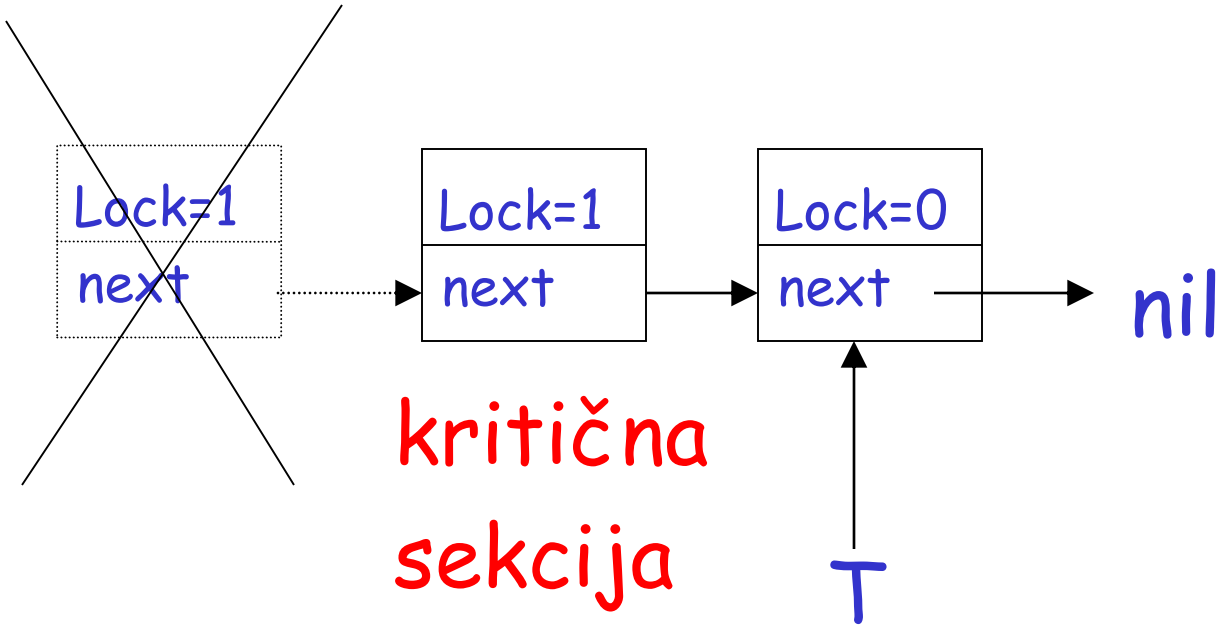


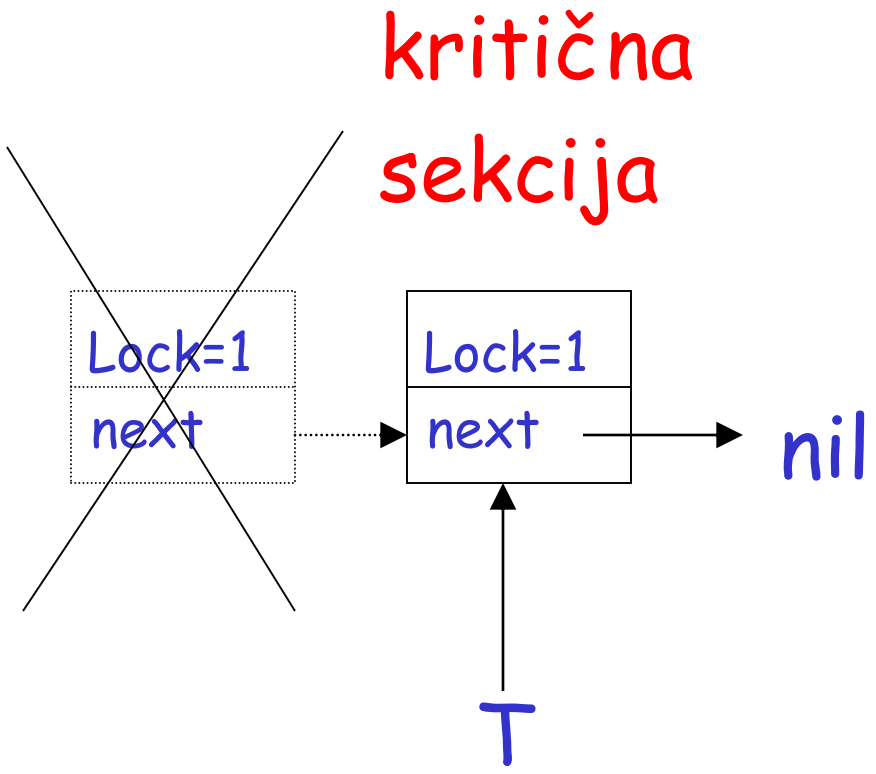
kritična sekcija

pre compare&swap



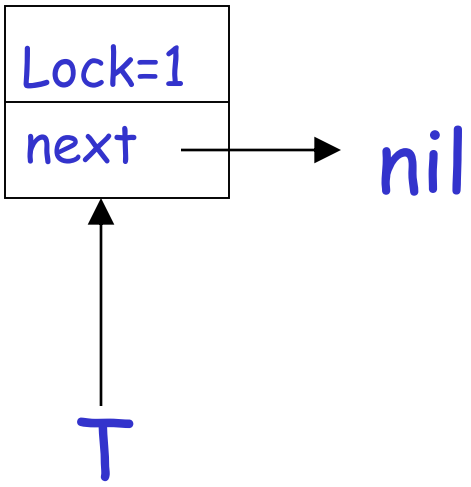
posle compare&swap


Lock=1
next
Lock=1
next
Lock=0
next
nil
kritična
sekcija
T


kritična
sekcija
Lock=1
next
Lock=1
next
nil
T

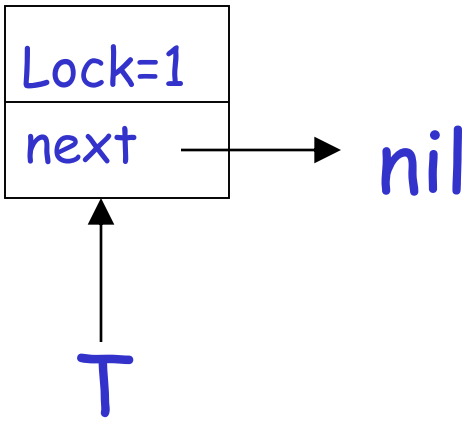
Ekstremni slučaj

kritična
sekcija



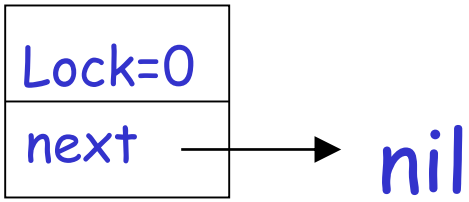
Ekstremni slučaj

kritična sekcija



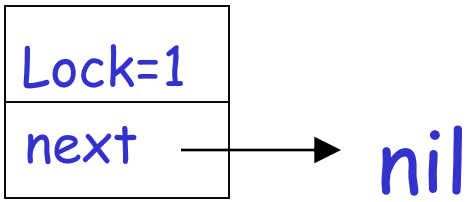
će izvršiti compare&swap

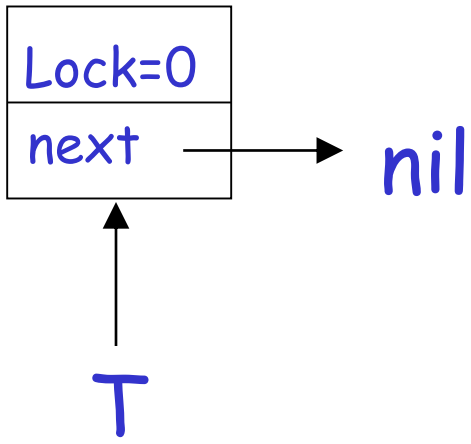
Nov čvor će bit umetnut



će izvršiti swap

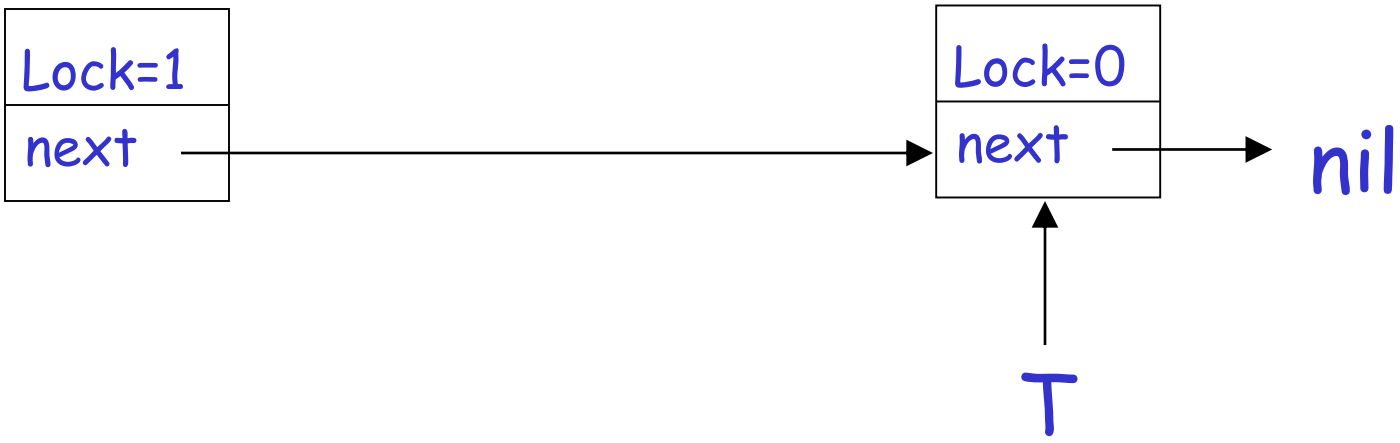
Slučaj 1: swap se izvrši prvi



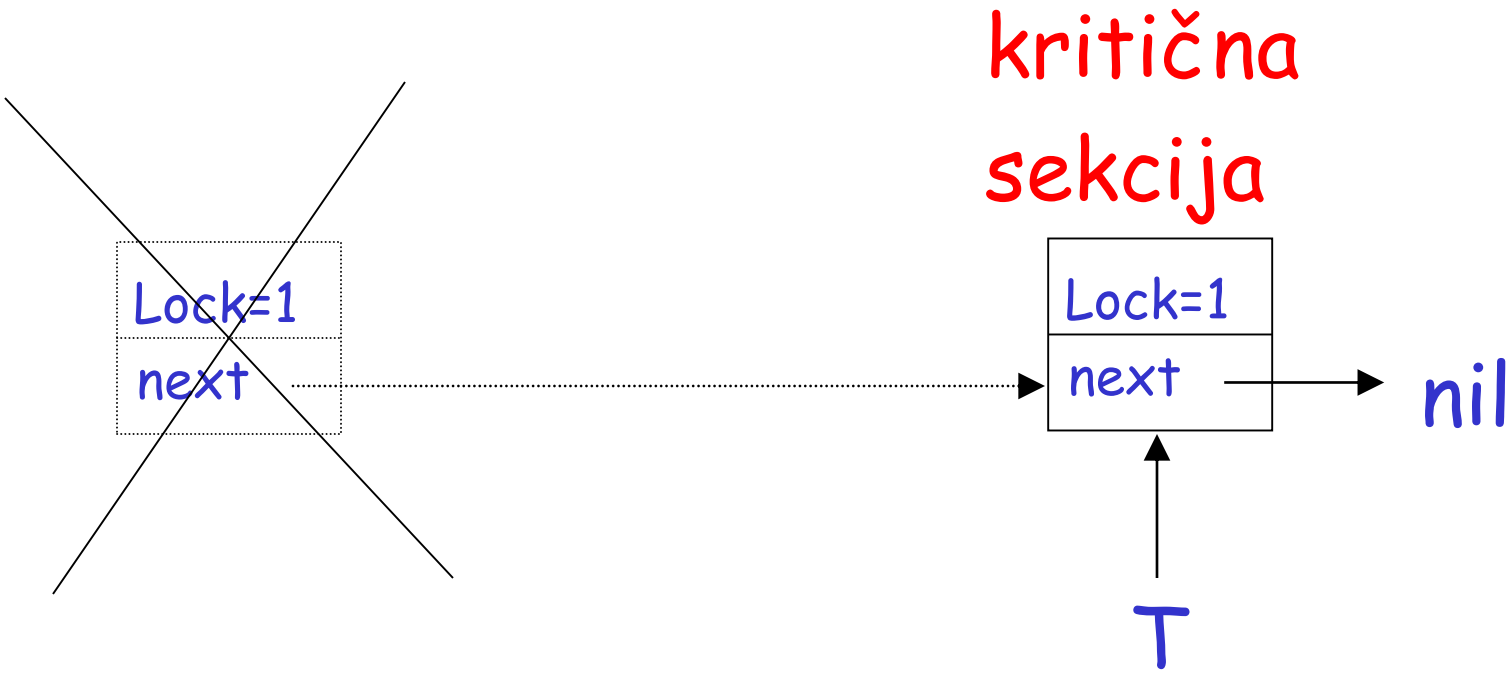


Čeka dok
next ne pokaže
na nešto

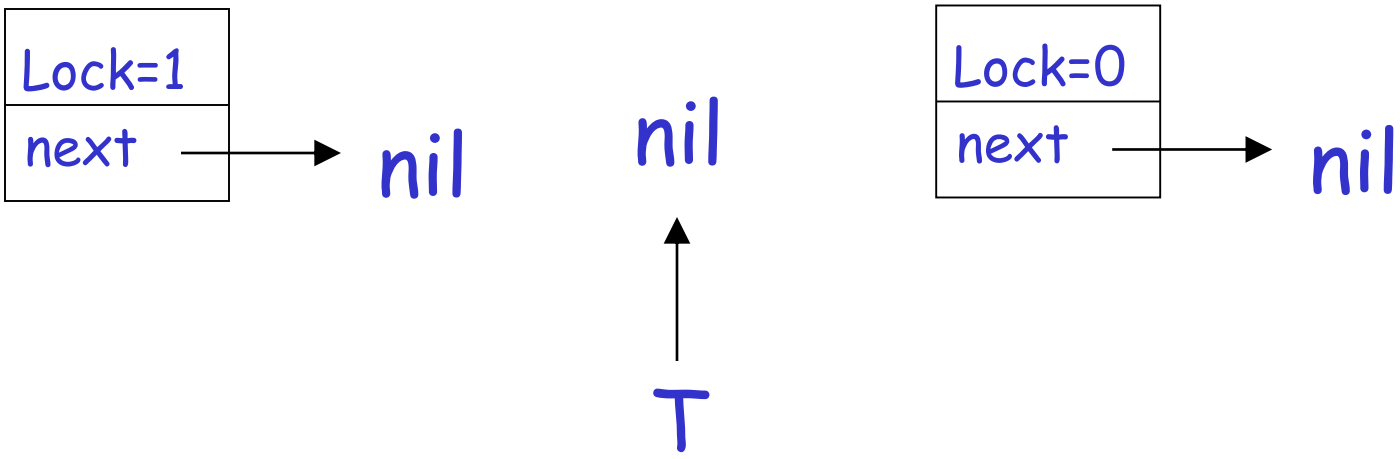
Slučaj 1: swap se izvrši prvi



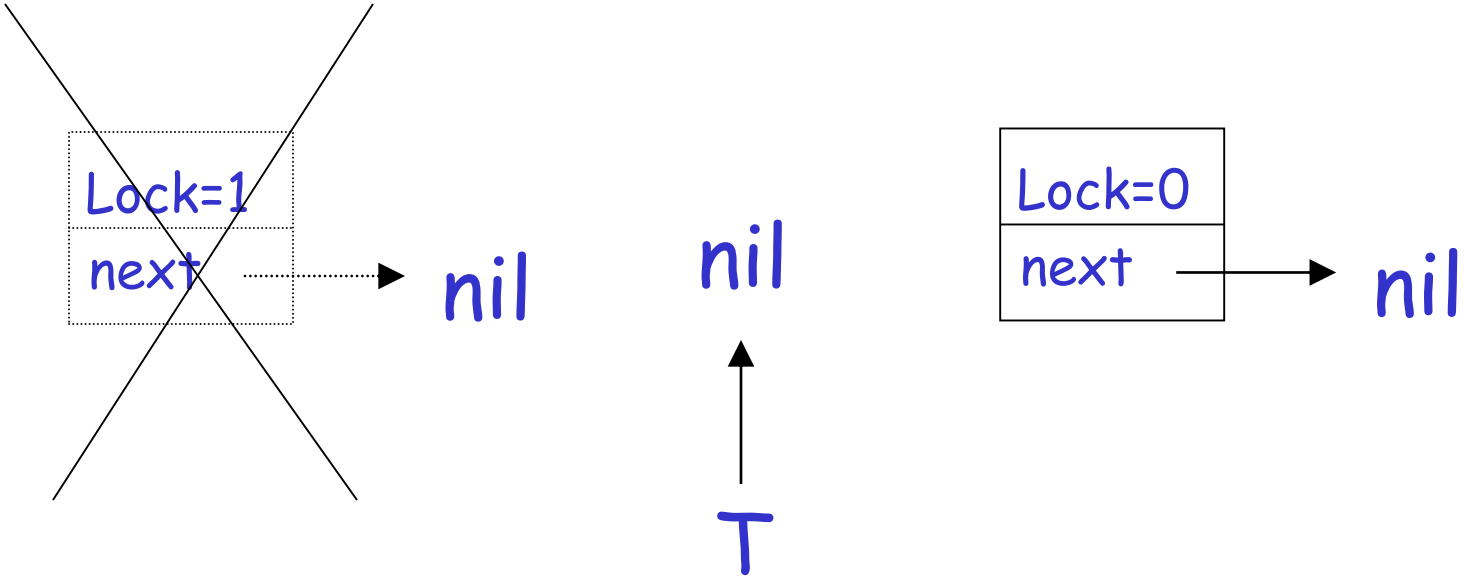
Slučaj 1: swap se izvrši prvi



Slučaj 2: compare&swap se izvrši prvi



Slučaj 2: compare&swap se izvrši prvi



Slučaj 2: compare&swap se izvrši prvi