DISTRIBUIRANI ALGORITMI I SISTEMI

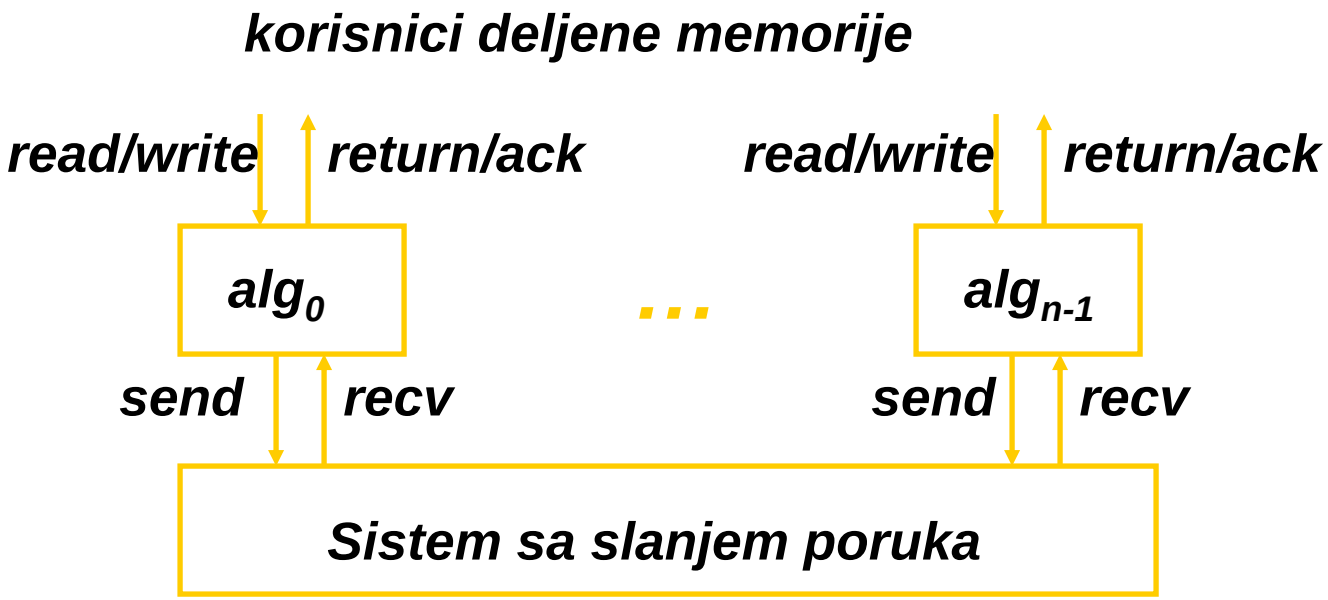
Iz kursa CSCE 668
Proleće 2014

Autor izvorne prezentacije:
Prof. Jennifer Welch

Distribuirana deljena memorija

- Jedan model međuprocesne komunikacije
- Obezbeđuje iluziju deljenih promenljivih na vrhu sistema za slanje poruka
- Deljena memorija se obično smatra pogodnijom platformom od sistema za slanje poruka
- Formalno, to je simulacija modela deljene memorije na vrhu modela sa slanjem poruka
- Posmatraćemo poseban slučaj:
  - nema otkaza
  - simuliraju se samo promenljive sa operacijama čitaj/piši

Simulacija


korisnici deljene memorije
read/write
return/ack
read/write
return/ack
alg0
...
algn-1
send
recv
send
recv
Sistem sa slanjem poruka

Problemi deljene memorije

- Proces poziva operaciju deljene memorije (čitaj ili piši) u nekom trenutku
- Simulacioni algoritam na nekom čvoru izvršava neki kod, uz moguću razmenu poruka
- Na kraju simulacioni algoritam informiše proces o rezultatu pozvane operacije
- Znači operacije deljene memorije nisu trenutne!
  - Operacije (pozvane od raznih procesa) se mogu preklapati
- Koje vrednosti treba da vrate operacije koje se preklapaju sa drugim operacijama?
  - to definiše **uslov memorijske konzistentnosti**

Sekvencijalne specifikacije

- Svaki deljeni objekt ima **sekvencijalnu spec.**: specificira ponašanje objekta u *odsustvu* konkurencije
- Objekt podržava **operacije**:
  - pozive
  - odgovarajuće odgovore
- Skup sekvenci operacija koje su **legalne**

Sekvencijalna specifikacija za R/W registre

- Svaka operacija ima dva dela, poziv i odgovor
- Operacija read ima poziv $read_i(X)$ i odgovor $return_i(X,v)$ (indeks *i* odgovara procesu)
- Operacija write ima poziv $write_i(X,v)$ i odgovor $ack_i(X)$ (indeks *i* odgovara procesu)
- Sekvenca operacija je legalna ako i samo ako svako čitanje vraća vrednost poslednjeg prethodnog upisa
- Npr.: $[write_0(X,3)\ ack_0(X)]\ [read_1(X)\ return_1(X,3)]$

Uslovi memorijske konzistentnosti

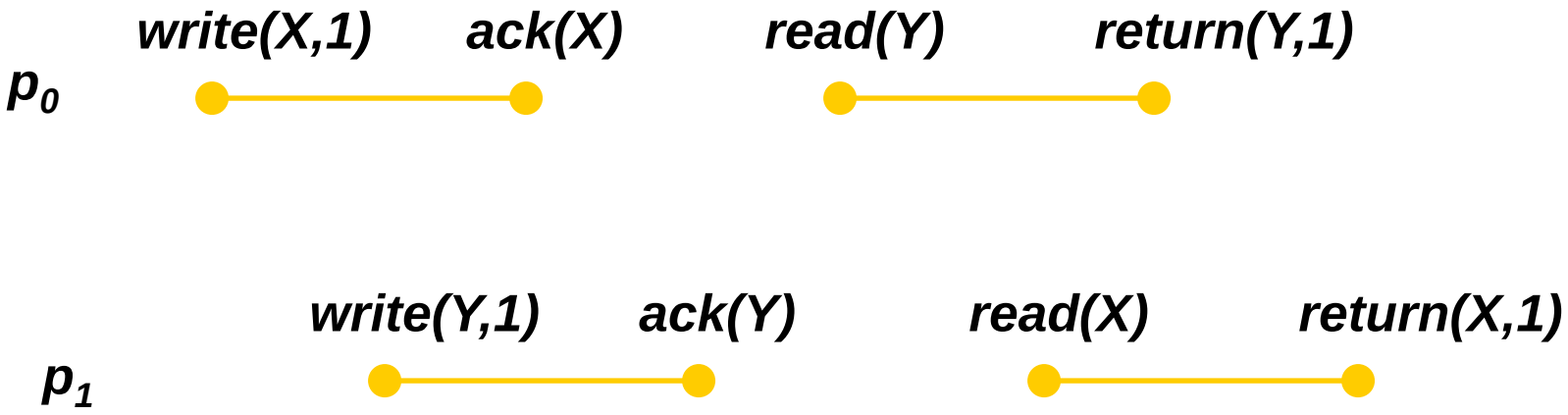
- Uslovi konzistentnosti povezuju sekvencijalnu specifikaciju sa onim što se dešava u prisustvu konkurencije
- Proučićemo dva dobro-poznata uslova:
  - mogućnost linearizacije (linearizability)
  - sekvencijalna konzistentnost
- Razmotićemo samo registre za čitanje/upis, u slučaju kada nema otkaza

Definicija mogućnosti linearizacije (Linearizability)

- Neka je σ sekvenca poziva i odgovora za skup operacija
  - poziv ne mora biti odmah praćen svojim odgovorom, može imati konkurentne, preklapajuće operacije
- σ se **može linearizovati** ako postoji permutacija π svih *operacija* u σ (gde je svaki poziv odmah praćen svojim odgovorom), koja zadovoljava:
  - $\pi | X$ je legalna (zadovoljava sekve. spec.) za sve prom. X, i
  - ako se odgovor za operaciju O_1 desio u σ pre poziva operacije O_2, onda se O_1 desila u π pre O_2 (π respektuje redosled realnog vremena neprekloplјenih operacija u σ)

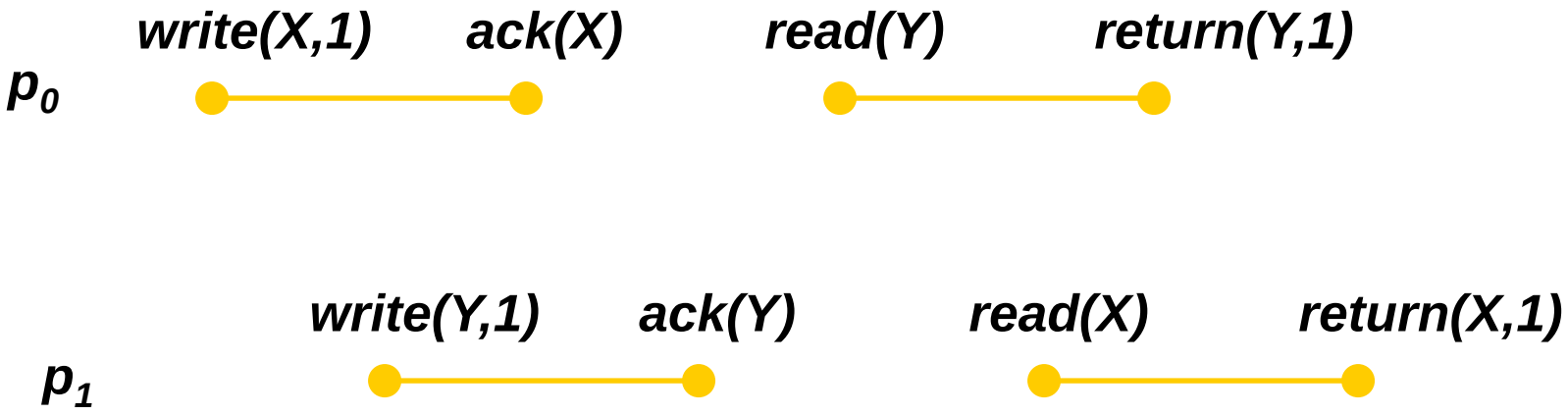
Primeri mogućnosti linearizacije

Neka postoje dve deljene promenljive, *X* i *Y*, obe inicijalno 0



Primeri mogućnosti linearizacije

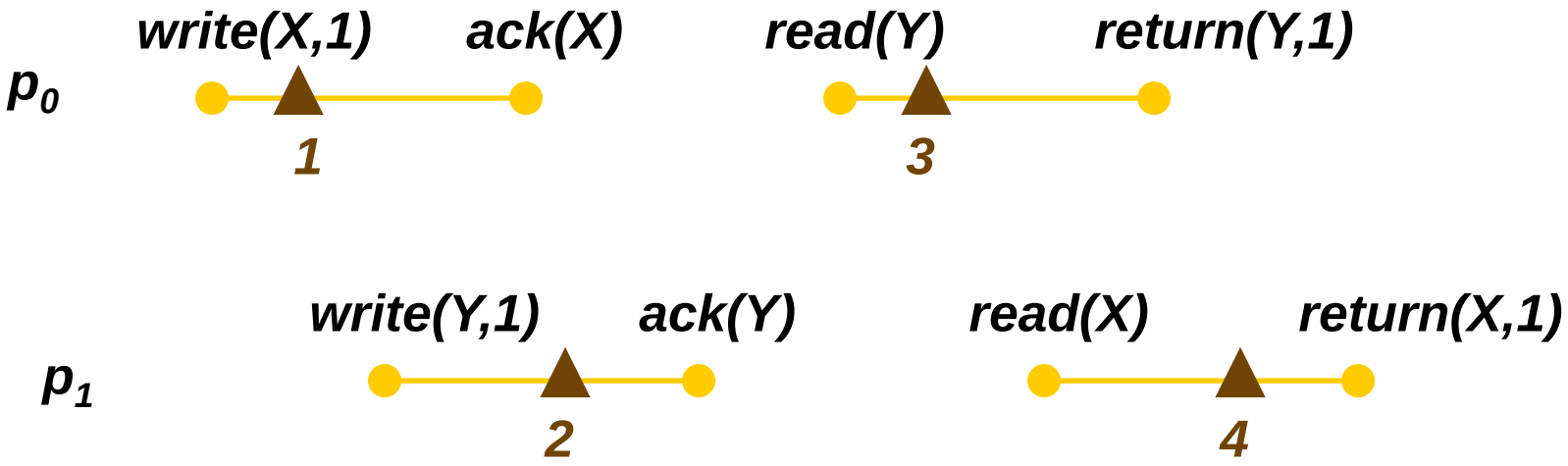
Neka postoje dve deljene promenljive, *X* i *Y*, obe inicijalno 0



Da li se ova sekvenca može linearizovati?

Primeri mogućnosti linearizacije

Neka postoje dve deljene promenljive, *X* i *Y*, obe inicijalno 0

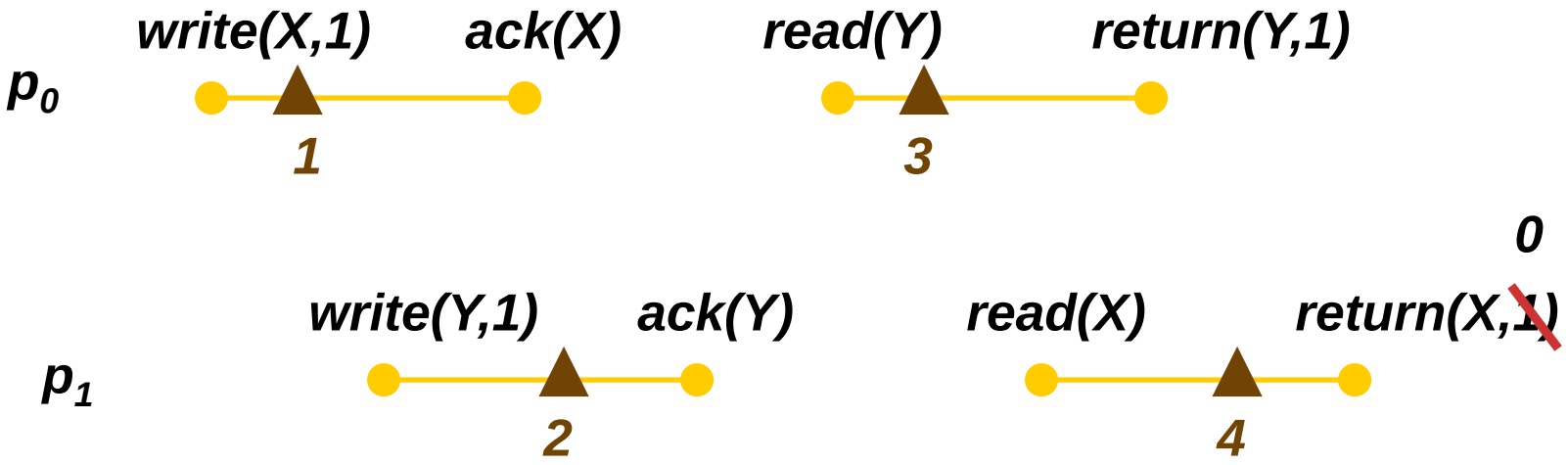


Da li se ova sekvenca može linearizovati?

Da – braon trouglovi.

Primeri mogućnosti linearizacije

Neka postoje dve deljene promenljive, *X* i *Y*, obe inicijalno 0



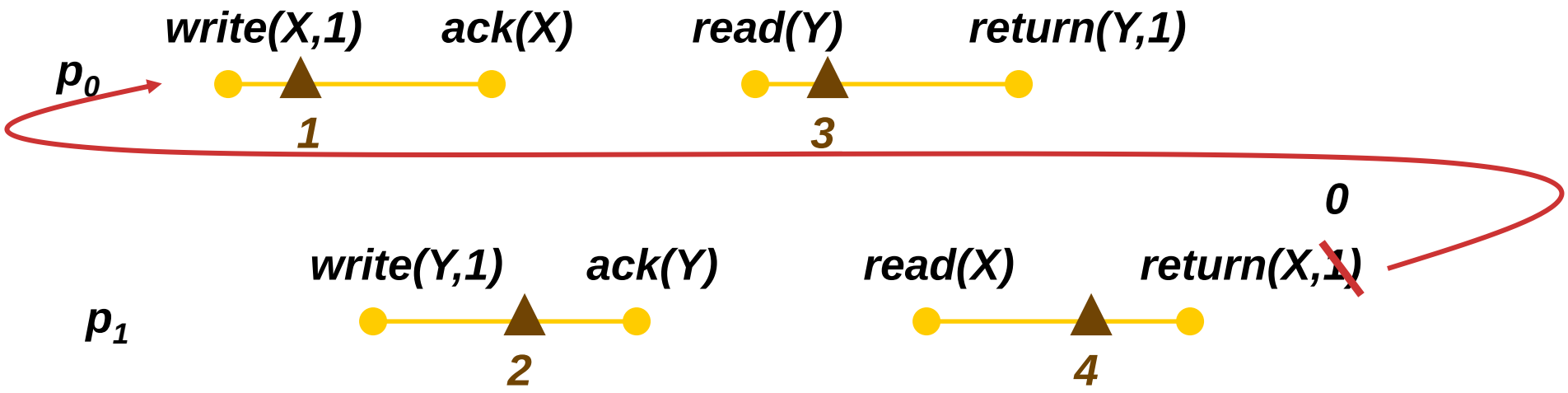
Da li se ova sekvenca može linearizovati?

Da – braon trouglovi.

Šta ako read od p_1 vrati 0?

Primeri mogućnosti linearizacije

Neka postoje dve deljene promenljive, *X* i *Y*, obe inicijalno 0



Da li se ova sekvenca može linearizovati?

Da – braon trouglovi.

Šta ako read od p_1 vrati 0?

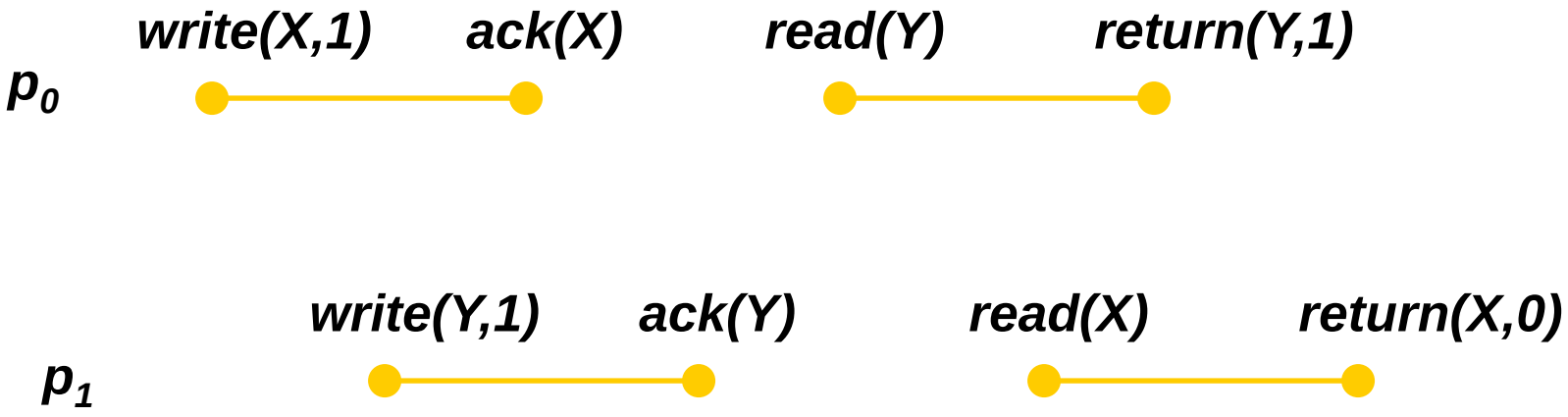
Ne – vidi strelicu.

Definicija sekve. konzistentnosti (Sequential Consistency)

- Neka je σ sekvenca poziva i odgovora za neki skup operacija
- σ je **sekve. konzistentna** ako postoji permutacija π svih operacija u σ takva da:
  - $\pi | X$ je legalna (zadovoljava sekve. spec.) za sve prom. X, i
  - ako se odgovor za operaciju O_1 desio u σ pre poziva operacije O_2 *u istom procesu*, onda se O_1 desila u π pre O_2 (π respektuje redosled realnog vremena operacija *od strane istog procesa* u σ)

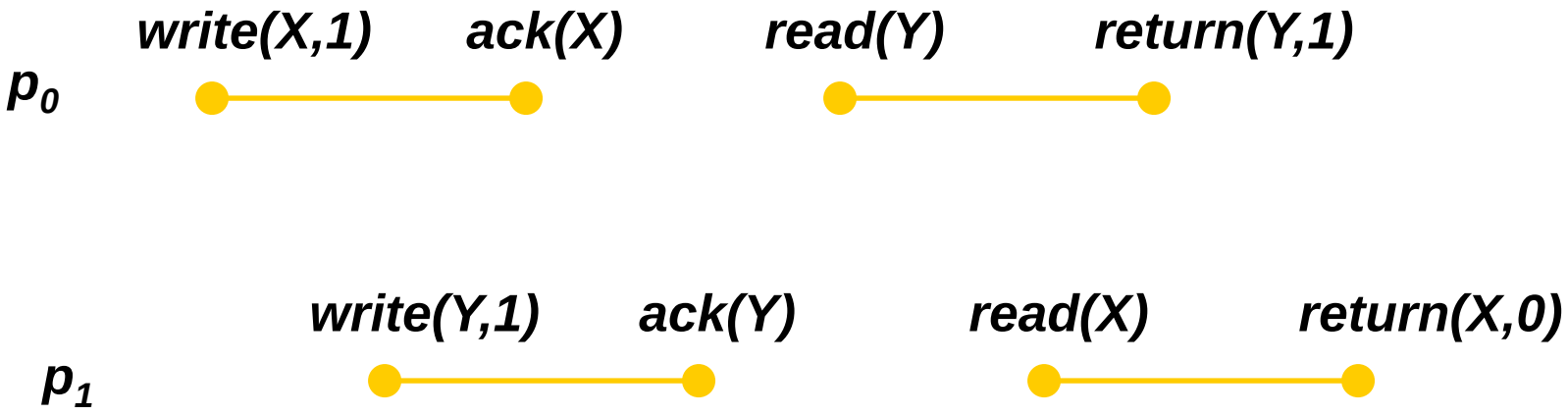
Primeri sekvencijalne konzistentnosti

Neka postoje dve deljene promenljive, *X* i *Y*, obe inicijalno 0



Primeri sekvencijalne konzistentnosti

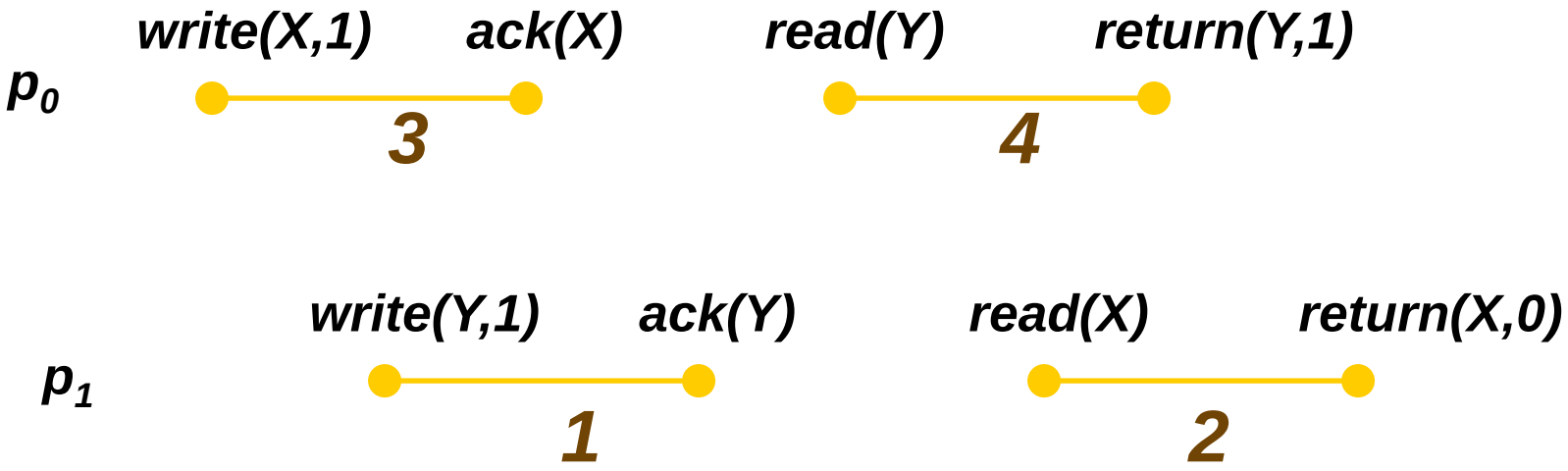
Neka postoje dve deljene promenljive, *X* i *Y*, obe inicijalno 0



Da li je ova sekvenca sekve. konzistenta?

Primeri sekvencijalne konzistentnosti

Neka postoje dve deljene promenljive, *X* i *Y*, obe inicijalno 0

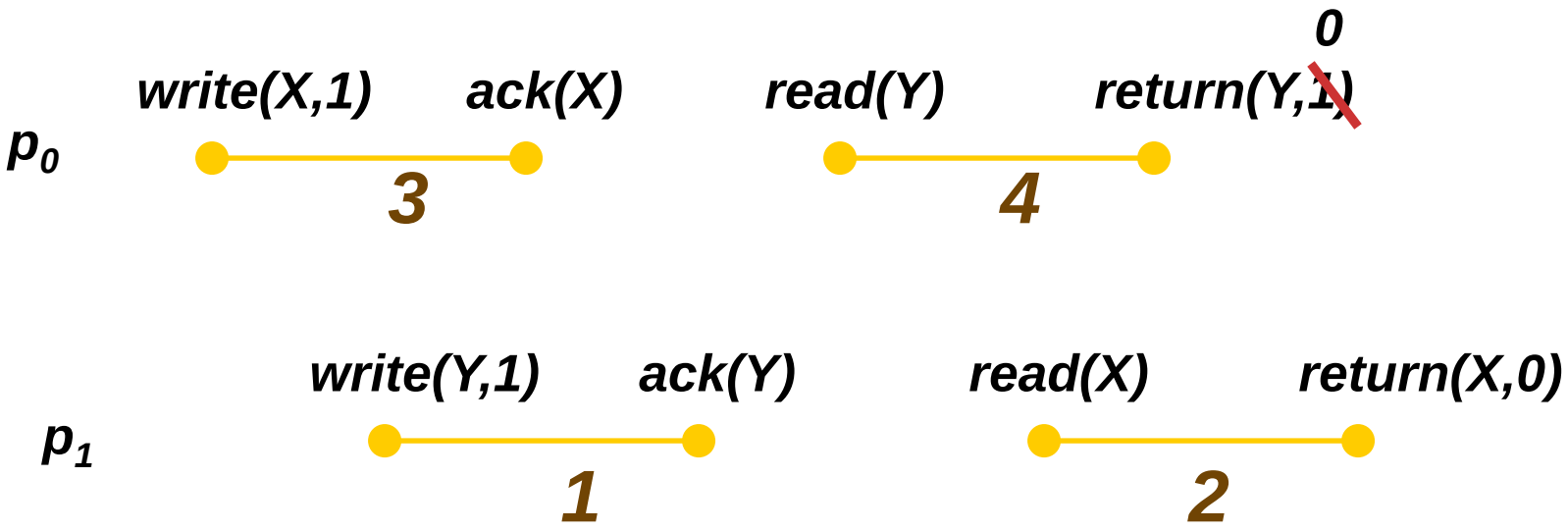


Da li je ova sekvenca sekve. konzistenta?

Da – braon brojevi.

Primeri sekvencijalne konzistentnosti

Neka postoje dve deljene promenljive, *X* i *Y*, obe inicijalno 0

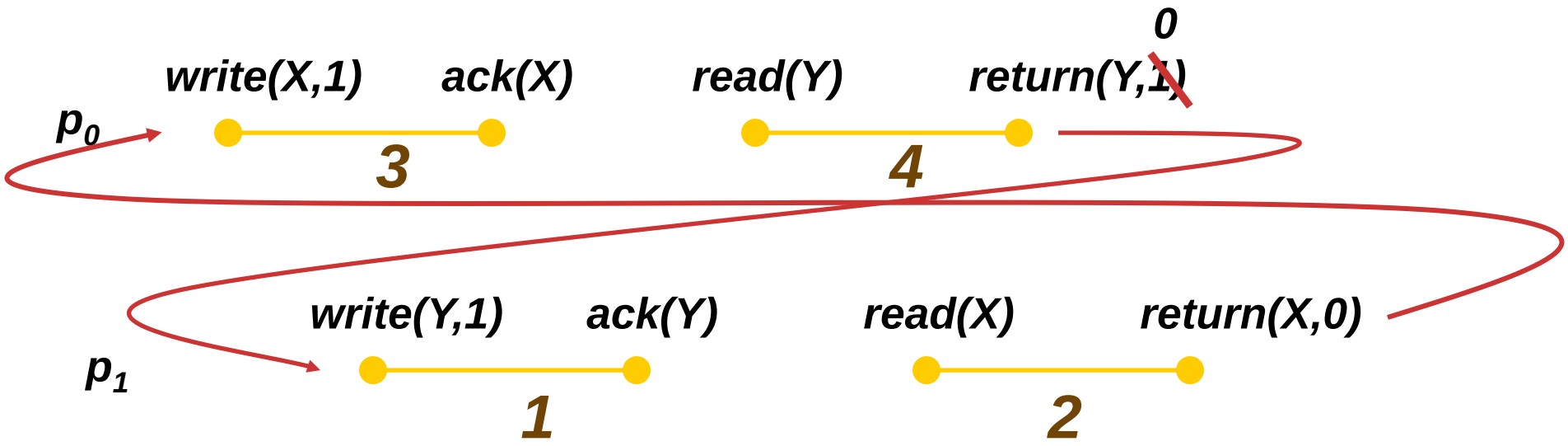


Da li je ova sekvenca sekve. konzistenta? Da – braon brojevi.

Šta ako read od p_1 vrati 0?

Primeri sekvencijalne konzistentnosti

Neka postoje dve deljene promenljive, *X* i *Y*, obe inicijalno 0



Da li je ova sekvenca sekve. konzistenta? Da – braon brojevi.

Šta ako read od p_1 vrati 0? Ne – vidi strelice.

Specifikacija deljene memorije čije sekvence se mogu linearizovati

- Ulazi su pozivi operacija deljenih objekata
- Izlazi su odgovori iz deljenih objekata
- Sekvenca σ je u dopuštenom skupu ako i samo ako:
  - *Korektna interakcija:* svaki proc. ima naizmenične pozive i odgovarajuće odzive
  - *Životnost:* svaki poziv ima odgovarajući odgovor
  - *Mogućnost linearizacije:* σ se može linearizovati

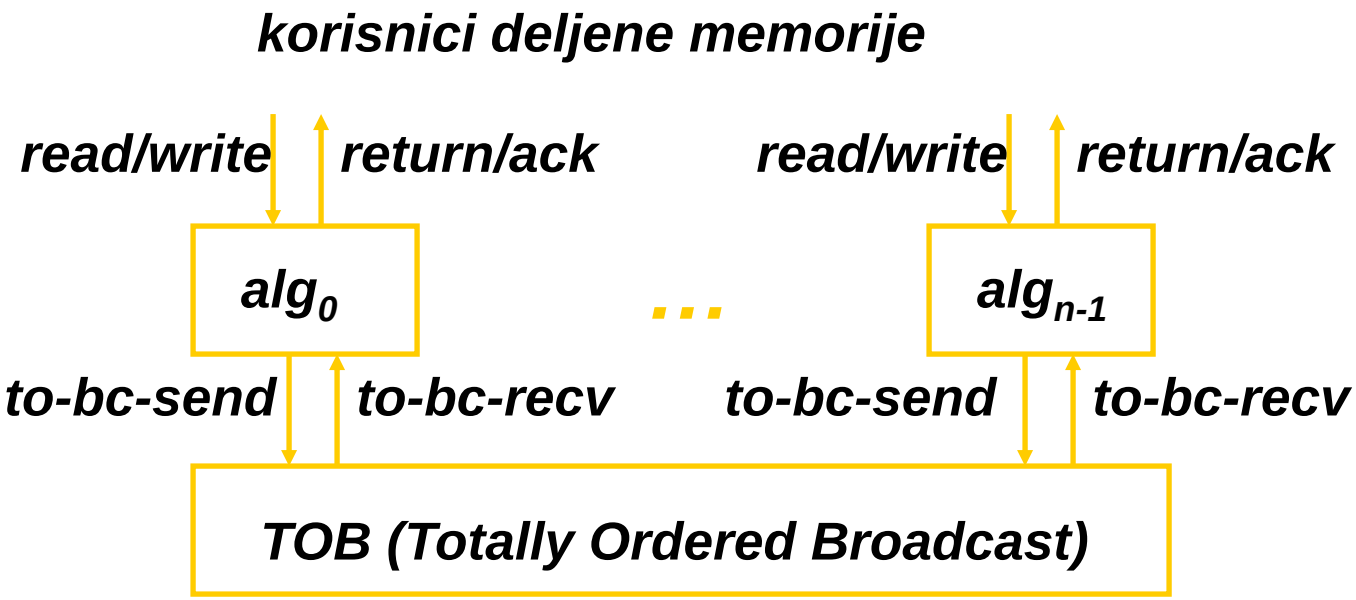
Specifikacija deljene memorije koja je sekvencijalno konzistentna

- Ulazi su pozivi operacija deljenih objekata
- Izlazi su odgovori iz deljenih objekata
- Sekvenca σ je u dopuštenom skupu ako i samo ako:
  - *Korektna interakcija:* svaki proc. ima naizmenične pozive i odgovarajuće odzive
  - *Životnost:* svaki poziv ima odgovarajući odgovor
  - *Sekvencijalna konzistentnost:* σ je sekvencijalno konzistentna

Algoritam deljene memorije čije sekvence se mogu linearizovati

- Koristi slanje poruke svima (broadcast) u skladu sa totalnim redosledom (total order)
- Svaki proc održava repliku za svaku deljenu promenljivu
- Kad stigne zahtev za čitanje:
  - pošalji bcast por. koja sadrži zahtev
  - kad stigne sopstvena bcast por., vrati vrednost lokalne replike
- Kad stigne zahtev za pisanje:
  - pošalji bcast por. koja sadrži zahtev
  - nakon prijema, svaki proc ažurira svoju repliku
  - kad stigne sopstvena bcast por., odgovori sa ack

Simulacija


korisnici deljene memorije
read/write
return/ack
read/write
return/ack
alg0
...
algn-1
to-bc-send
to-bc-recv
to-bc-send
to-bc-recv
TOB (Totally Ordered Broadcast)

Korektnost ovog algoritma

- Neka je α bilo koje prihvatljivo izvršenje algoritma za koje važi:
  - slanje poruka svima u skladu sa totalnim redosledom radi ispravno
  - korisnička interakcija je ispravna (naizmenični pozivi i odgovori)
- Pokazati da σ (restrikcija α na događaje na vrhu interfejsa) zadovoljava Životnost i Mogućnost linearizacije

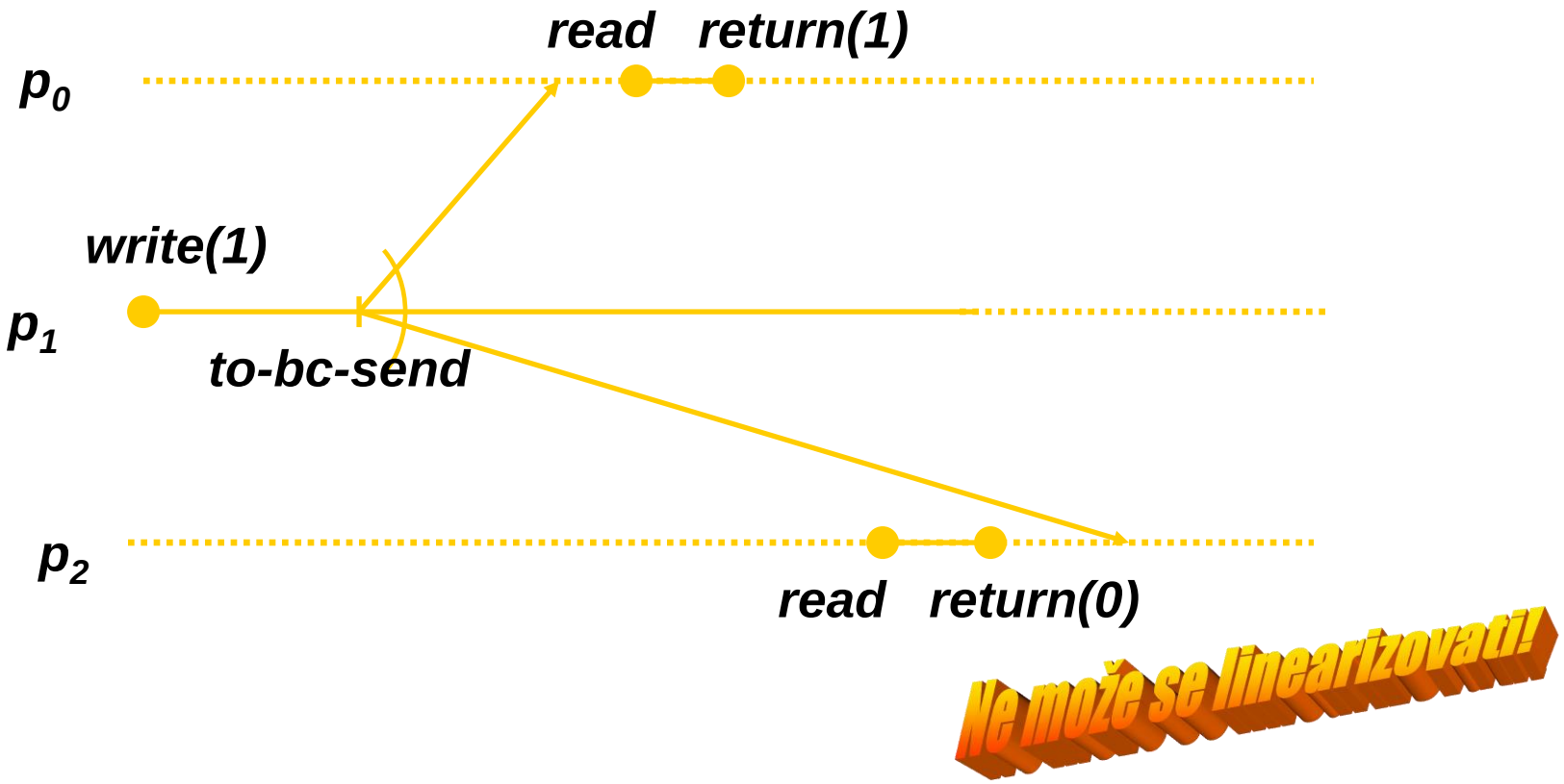
Korektnost ovog algoritma

- Životnost (svaki poziv ima svoj odgovor): po svojstvu životnosti TOB (Totally Ordered Broadcast)
- Mogućnost linearizacije: Definišimo permutaciju operacija π da bude u redosledu u kom su odgovarajuće bcast por. primljene:
  - π je legalna: jer su sve operacije konzistentno poređane po TOB.
  - π respektuje redosled operacije u realnom vremenu: ako se O_1 završi pre nego O_2 počne, bcast od O_1 se obavlja pre bcast od O_2

Zašto je potrebno slanje svima (Bcast) kod operacije čitanja?

- Slanje svima kod read operacije ne izaziva promene replika, samo kasni odgovore na read op
- Zašto je ono ipak potrebno?
- Pogledajmo šta se dešava ako ga uklonimo

Zašto je read bcast neophodan?



Algoritam za sekvencijanu konzistentnost (SC algoritam)

- Kao predhodni algoritam, ali bez bcast za čitanja:
- Koristi TOB
- Svaki procesor održava repliku svake deljene promenljive
- Kad stigne zahtev za čitanje:
  - odmah vrati vrednost lokalne replike
- Kad stigne zahtev za pisanje:
  - pošalji bcast por. koja sadrži zahtev
  - nakon prijema, svaki proc ažurira svoju repliku
  - kad stigne sopstvena bcast por., odgovori sa ack

Korektnost SC algoritma

Lema (8.3): Lokalne kopije svakog proc. uzimaju sve vrednosti iz write operacija, u istom redosledu, koji očuvava redosled neprekloplјenih write operacija

- implicira da je redosled write op po proc. očuvan

Lema (8.4): Ako p_i piše u Y i kasnije čita X, onda p_i-jevo ažuriranje njegove lokalne kopije Y (unutar write op) predhodi njegovom čitanju svoje lokalne kopije X (unutar read op)

Korektnost SC algoritma

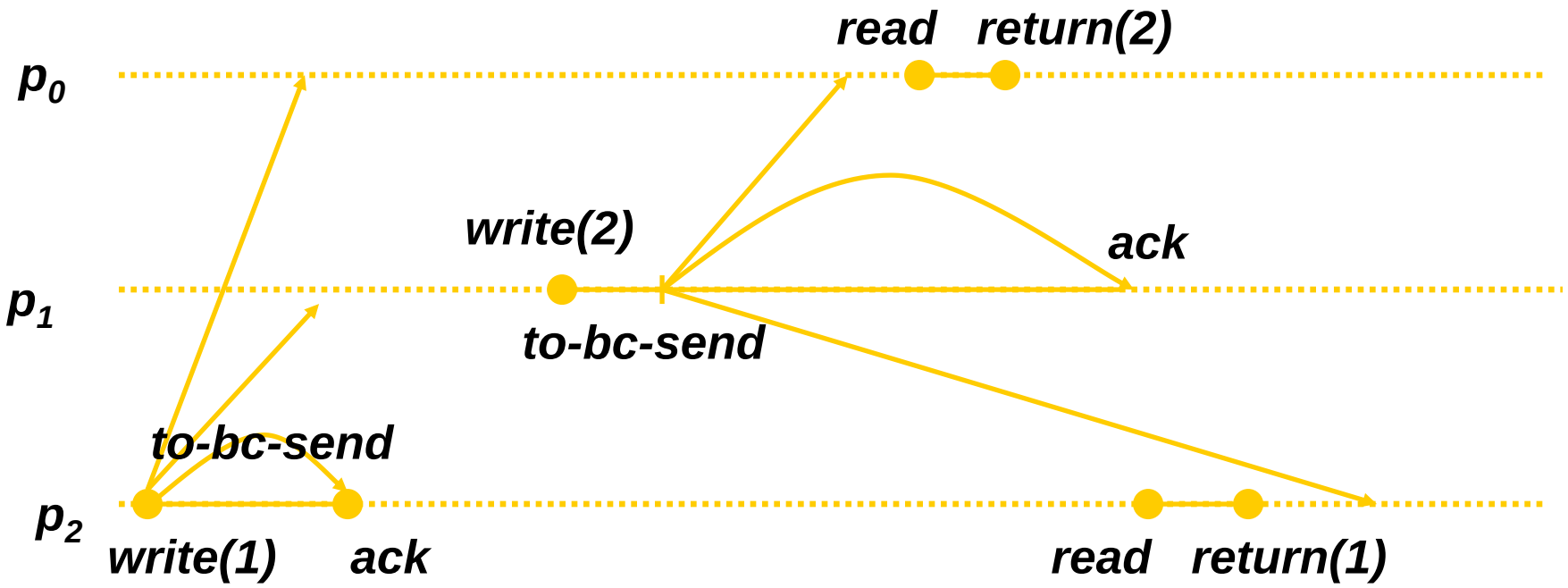
(Teorema 8.5) Zašto važi SC?

- Za bilo koje prihvatljivo izvršenje α, moramo doći do permutacije π operacija deljene mem. koja je:
  - legalna i
  - respektuje redosled operacija po proc.

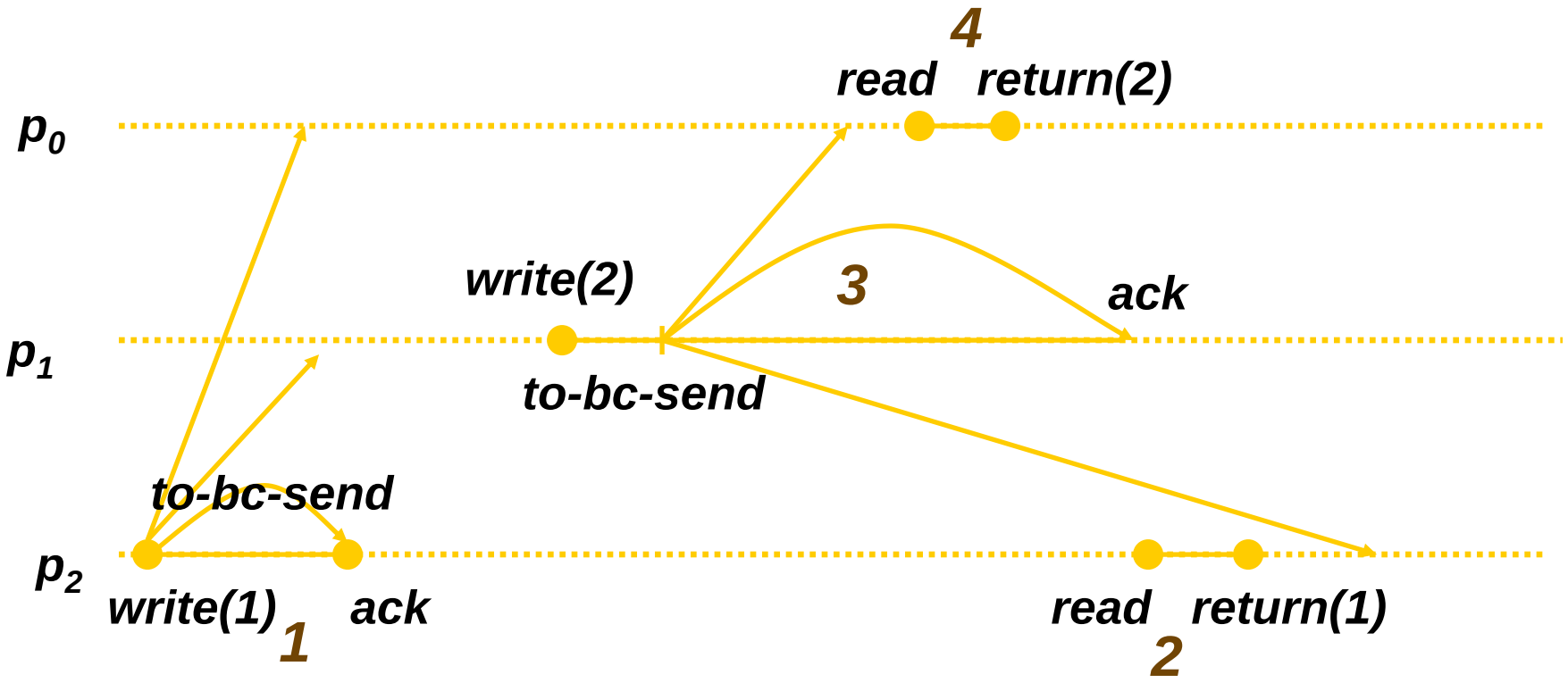
Permutacija π

- Ubacite sve write op u π u njihovom TOB redosledu
- Razmotrimo svaki read *R* u α u redosledu poziva:
  - neka je *R* čitanje *X* od p_i
  - stavi *R* u π odmah iza poslednje od ove 2 op:
    1. operacija od p_i koja neposredno predhodi *R* u α, i
    2. Write op „iz koje *R* čita“ (izaziva poslednje ažuriranje p_i -jeve lokalne kopije *X*, koje predhodi odgovoru za *R*)

Primer permutacije



Primer permutacije



permutacija je data sa braon brojevima

Performansa SC algoritma

- Read operacije su implementirane „lokalno“, i ne zahtevaju međuprocesnu komunikaciju
- Zato se read op može posmatrati kao „brza“: vreme između poziva i odgovora je vreme lokane obrade
- Vreme za write op je vreme isporuke jedne to-bcast por (zavisi od implementacije to-bcast)

Alternativni SC algoritam

- Postoji alternativni alg koji ima obrnutu performansu:
  - write op je lokalna/brza (iako se šalju bcast-ovi, ne čeka se da oni budu primljeni)
  - read op može zahtevati da neki bcast-ovi budu primljeni
- Kao i predhodni SC algoritam, ni ovaj ne obezbeđuje mogućnost linearizacije sekvence op deljene memorije