Upravljanje distribuiranim procesima

Operativni sistemi 2

Januar 2014. god.

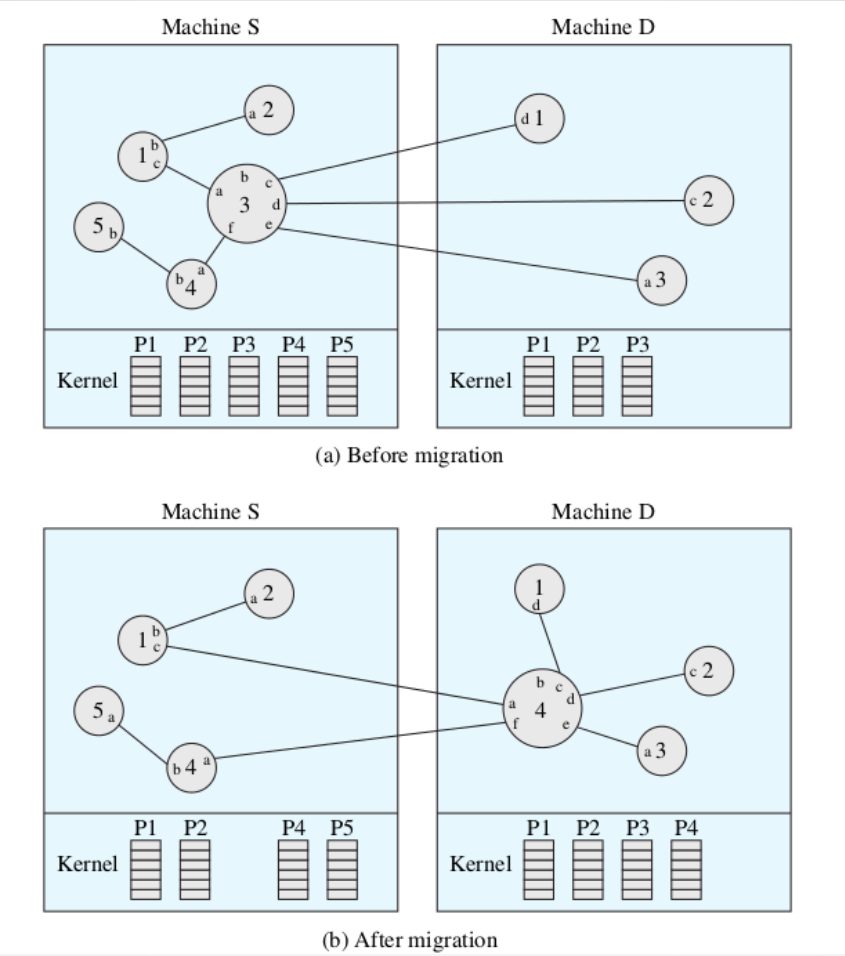
Migracija procesa

- **Migracija procesa** je prenos dovoljne količine stanja nekog procesa tako da se taj proces može izvršavati na drugom računaru.
- Komplikovanija implementacija nego što se u početku očekivalo.
- **Motivacija**
  - Deljenje opterećenja
  - Performanse komunikacija
  - Raspoloživost
  - Upošljavanje posebnih mogućnosti hardvera/softvera

Mehanizmi migracije procesa

- **Ko inicira migraciju?** Najčešće modul kernela koji je zadužen za monitoring i komunikaciju.
- **Šta se tačno migrira?** Uništiti proces na jedom, a pokrenuti ga na drugom računaru, ali uz očuvanje svih veza (poruka, signala). Premeštanje PCB-a je jednostavno, dok problemi nastaju kada treba da se premesti adresni prostor, otvoreni fajlovi itd.
  - *Doslovno (sve)*. Loše performanse.
  - *Kopiranje unapred*. Proces nastavlja da se izvršava dok se kopira
  - *Doslovno (prljavo)*. Prenose se samo stranice koje su u RAM-u i koje su promenjene. Ostale stranice na zahtev.
  - *Kopiranje na zahtev*. Varijanta prethodnog.
  - *Oslobađanje memorije*. Stranice procesa na izvorišnom računaru se povlače iz RAM-a tako što se modifikovane stranice snimaju na disk.
- **Odluka zavisi od odgovora na sledeća pitanja:**
  - Da li proces prelazi privremeno ili za stalno?
  - Da li se migriraju pojedinačne niti ili celoviti procesi?
  - Da li se uvek isplati premeštati sve otvorene fajlove?
- **Šta se dešava sa porukama i signalima?**
  - Potrebno je obezbediti mehanizam za privremeno skladištenje.

Mehanizmi migracije procesa



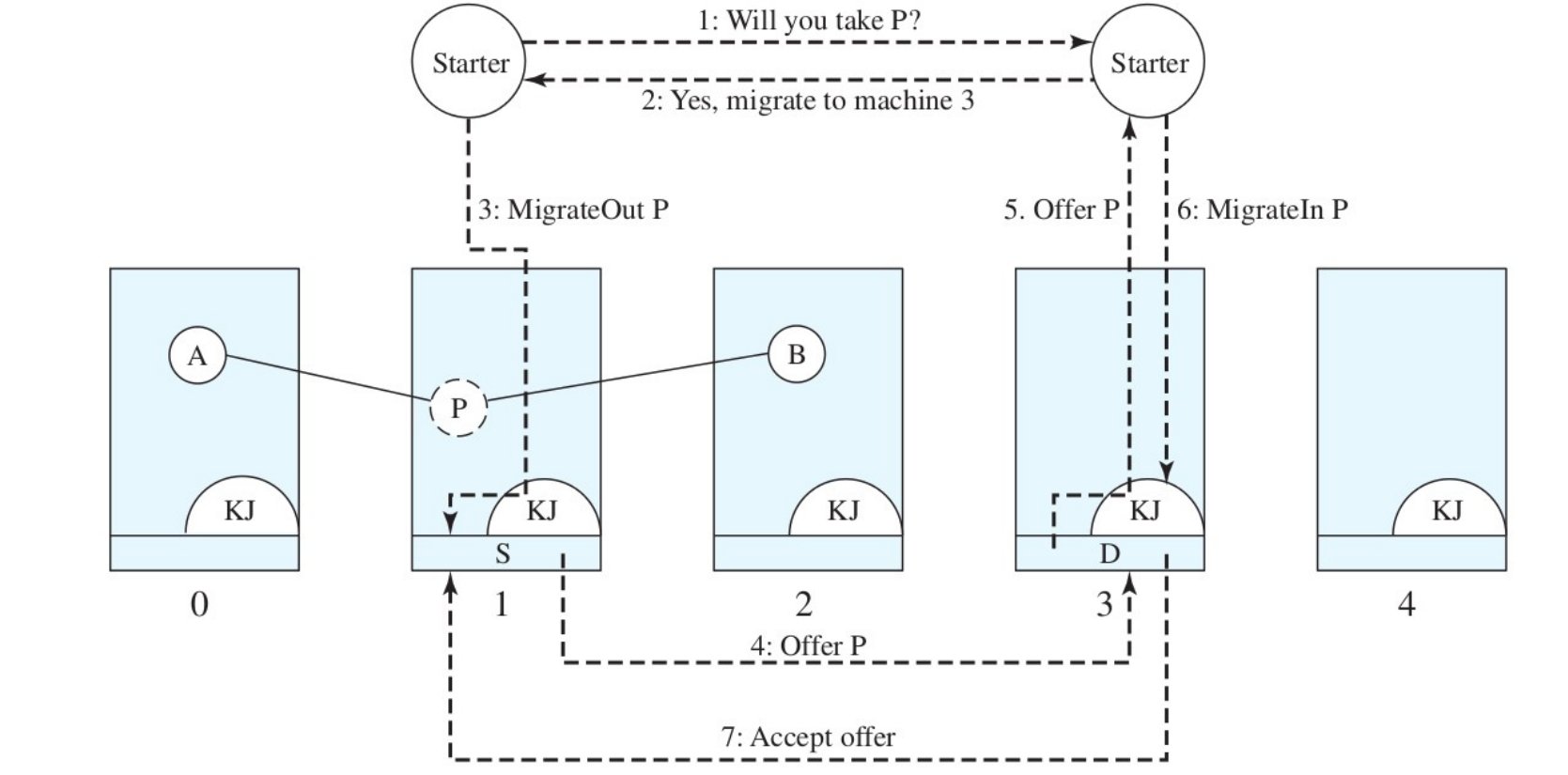
- **Non-preemptive migracija** – pre nego što se proces pokrene
- **Preemptive migracija** – u toku izvršavanja procesa

Primer slučaja upotrebe – IBM AIX

- Proces odlučuje da se migrira, bira odredišni računar i šalje poruku koja nosi deo slike procesa i info o otvorenim fajlovima.
- Na odredištu se otvara proces potomak sa tim informacijama
- Kopira se tekst segment, podaci, stek, argumenti, ...
- Prvi proces se izveštava o uspešnoj migraciji i potom uništava sam sebe.

Dogovaranje migracije - *Charlotte*

- Program **Starter** upravlja migracijom. U tačno određenim intervalima prima statističke informacije od jezgara.



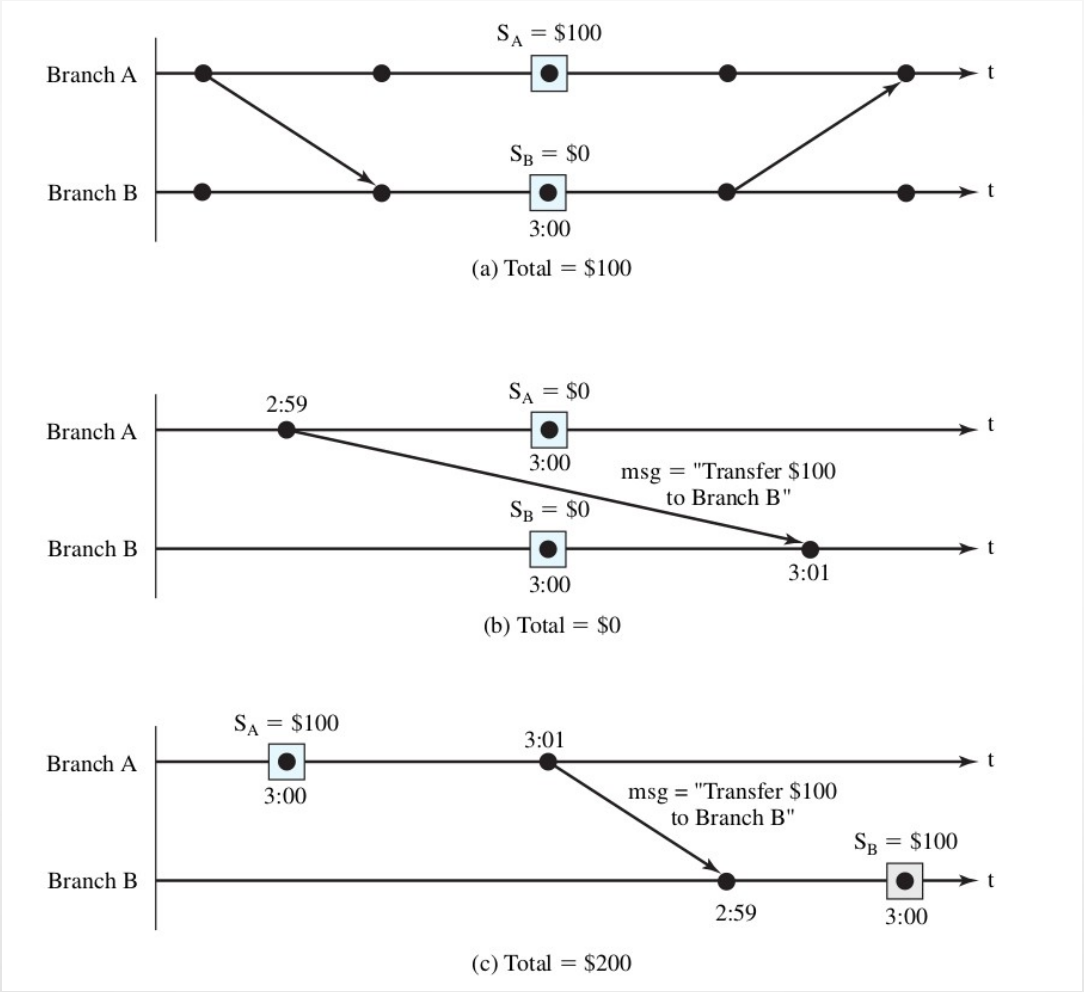
Izbacivanje – primer *Sprite OS*

Ponekad je potrebno izbaciti migrirane procese na stari host kako bi se obezbedilo odgovarajuće vreme odziva isl.

- Monitoring prati trenutno opterećenje hostova
- Proces koji se izbacuje migrira se nazad na domaći host
- Svi procesi obeleženi za izbacivanje momentalno se prekidaju
- Celokupni adresni prostor se prebacuje na domaći čvor

Distribuirana globalna stanja

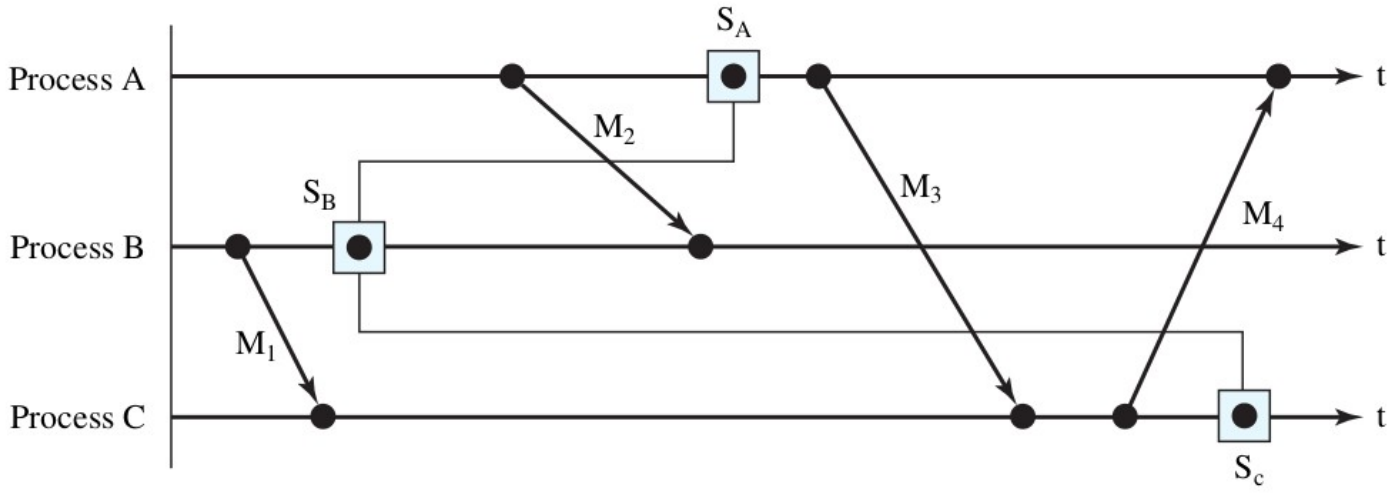
- **Aksiom 1**: Nemoguće je da jedan OS zna stanja svih procesa u distribuiranom sistemu.



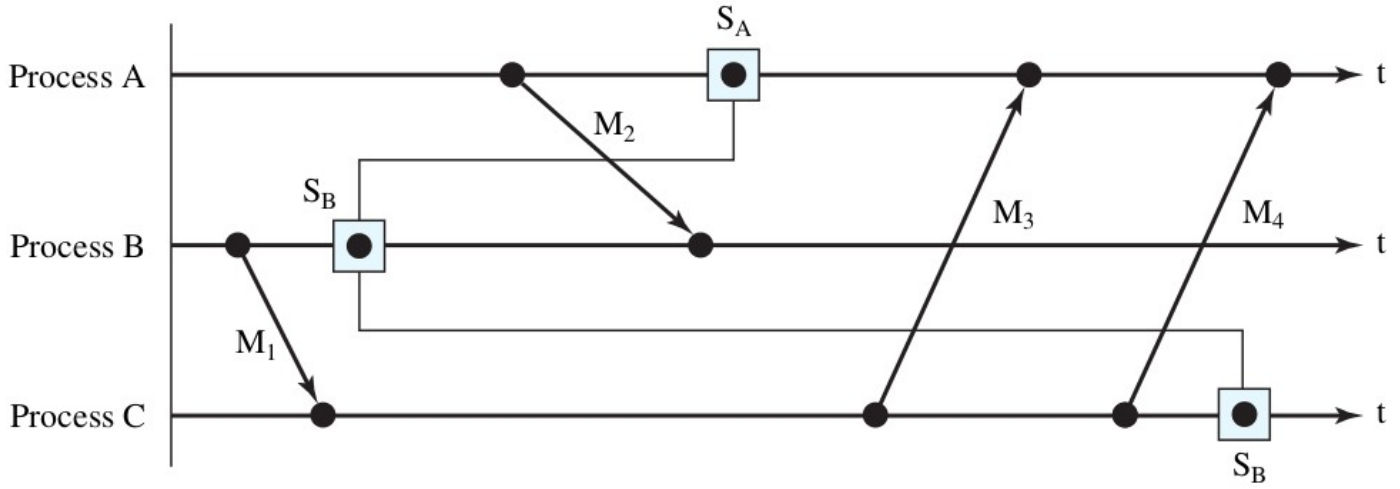
Distribuirana globalna stanja - pojmovi

- **Kanal** – Jednosmerni put poruka između 2 procesa
- **Stanje** – Niz poruka koje su poslate ili primljene duž kanala
- **Trenutni snimak** – Beleži stanje procesa
- **Globalno stanje** – Zajedničko stanje svih procesa
- **Distribuirani trenutni snimak** – Skup trenutnih snimaka za svaki proces

Konzistentno i nekonzistentno stanje



(a) Inconsistent global state

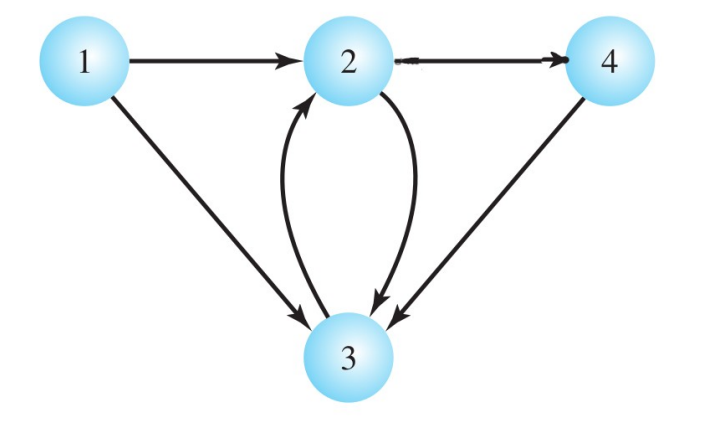


(b) Consistent global state

Algoritam distribuiranog trenutnog snimka

- Specijalna kontrolna poruka – **marker**
- Proces *p* nakon prijema markera, od procesa *q* radi sledeće (atomski):
  - *p* zapisuje svoje sopstveno stanje S_p
  - *p* zapisuje stanje ulaznog kanala od procesa *q* kao da je prazno
  - *p* prenosi marker svim susedima duž svih izlaznih kanala
- Kada p primi marker, npr. od procesa *r*, on zapisuje stanje kanala od *r* ka *p* kao niz poruka koje je primio od zapisa S_p do trenutka prijema markera od *r*.
- Algoritam se završava nakon što *p* primi marker duž svih svojih ulaznih linija.

Algoritam distribuiranog trenutnog snimka



Process 1
Outgoing channels
2 sent 1,2,3,4,5,6
3 sent 1,2,3,4,5,6
Incoming channels

Process 3
Outgoing channels
2 sent 1,2,3,4,5,6,7,8
Incoming channels
1 received 1,2,3 stored 4,5,6
2 received 1,2,3 stored 4
4 received 1,2,3

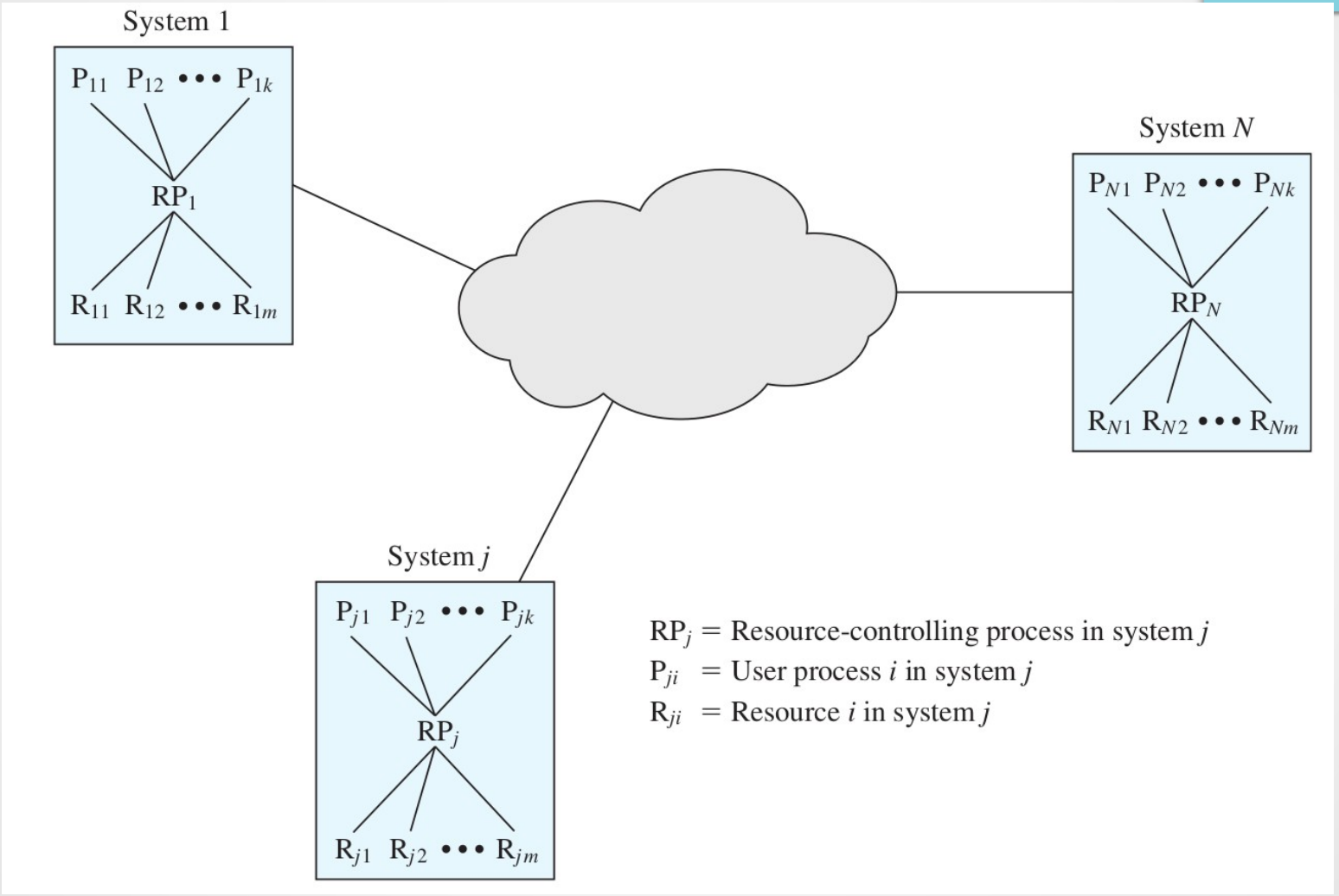
Process 2
Outgoing channels
3 sent 1,2,3,4
4 sent 1,2,3,4
Incoming channels
1 received 1,2,3,4 stored 5,6
3 received 1,2,3,4,5,6,7,8

Process 4
Outgoing channels
3 sent 1,2,3
Incoming channels
2 received 1,2 stored 3,4

Algoritam distribuiranog trenutnog snimka

- Bilo koji proces može pokrenuti algoritam slanjem markera. Čak i više njih istovremeno.
- Algoritam se okončava u konačnom vremenu
- Algoritam je distribuiranog tipa
- Nakon završetka, konzistentno globalno stanje se skuplja na jednom čvoru
- Ne utiče na ostale distribuirane algoritme

Distribuirano međusobno isključenje



Distribuirano međusobno isključenje

- Samo jedan proces sme biti u svojoj kritičnoj sekciji
- Proces koji se zaustavlja u svojoj sekciji koja nije kritična ne sme da spreči ostale da uđu u svoje k.s.
- Proces ne sme beskonačno dugo da čeka ulazak u svoju k.s.
- Kada nijedan proces nije u k.s. bilo kom procesu koji zateva ulaz u k.s. mora se dozvoliti da uđe bez čekanja
- Unapred nisu poznate relativne brzine procesa i broj procesora
- Proces unutar svoje k.s. ostaje konačno dugo

Distribuirano međusobno isključenje

- **Centralizovano rešenje**
  - Jednostavno
- **Distribuirano rešenje**
  - Svi čvorovi imaju približno jednak nivo informacija
  - Svaki čvor ima samo delimičnu sliku
  - Svi čvorovi su podjednako odgovorni za odluku
  - Otkaz čvora nema za posledicu pad čitavog sistema
  - Ne postoji zajednički generator takta za vremensku sinhronizaciju

Redosled događaja u distribuiranom sistemu

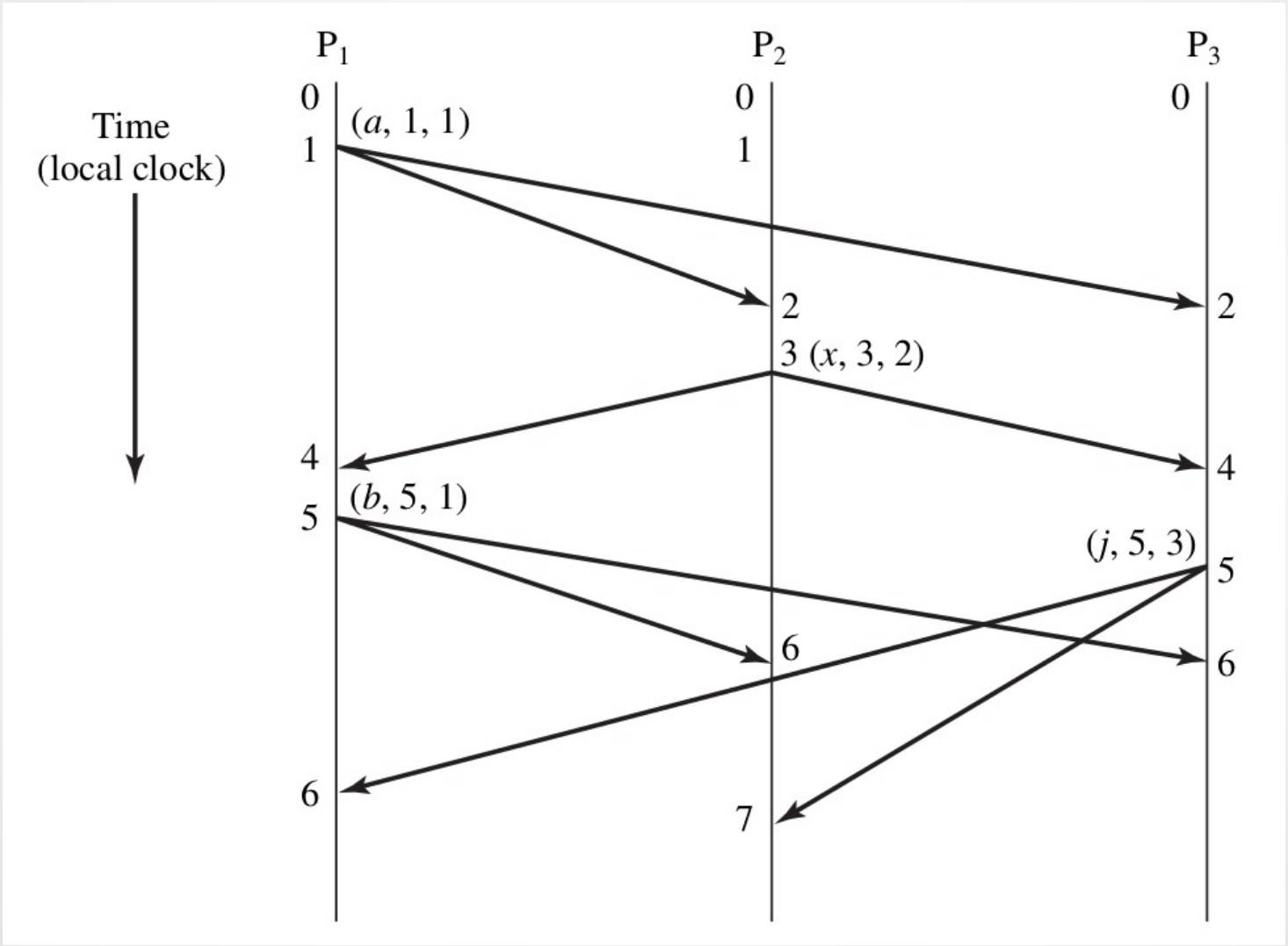
- Pod događajem se smatra slanje poruke iz nekog procesa
- Svaki proces ima svoj brojač (generator takta) C_i
- Poruka se šalje u obliku **(m, T_i, i)**
  (sadržaj, *timestamp*, oznaka računara)
- Kada se poruka primi prijemni sistem ***j*** postavlja vrednost svog generatora takta na

  $C_i = 1+\max(T_i, C_j)$

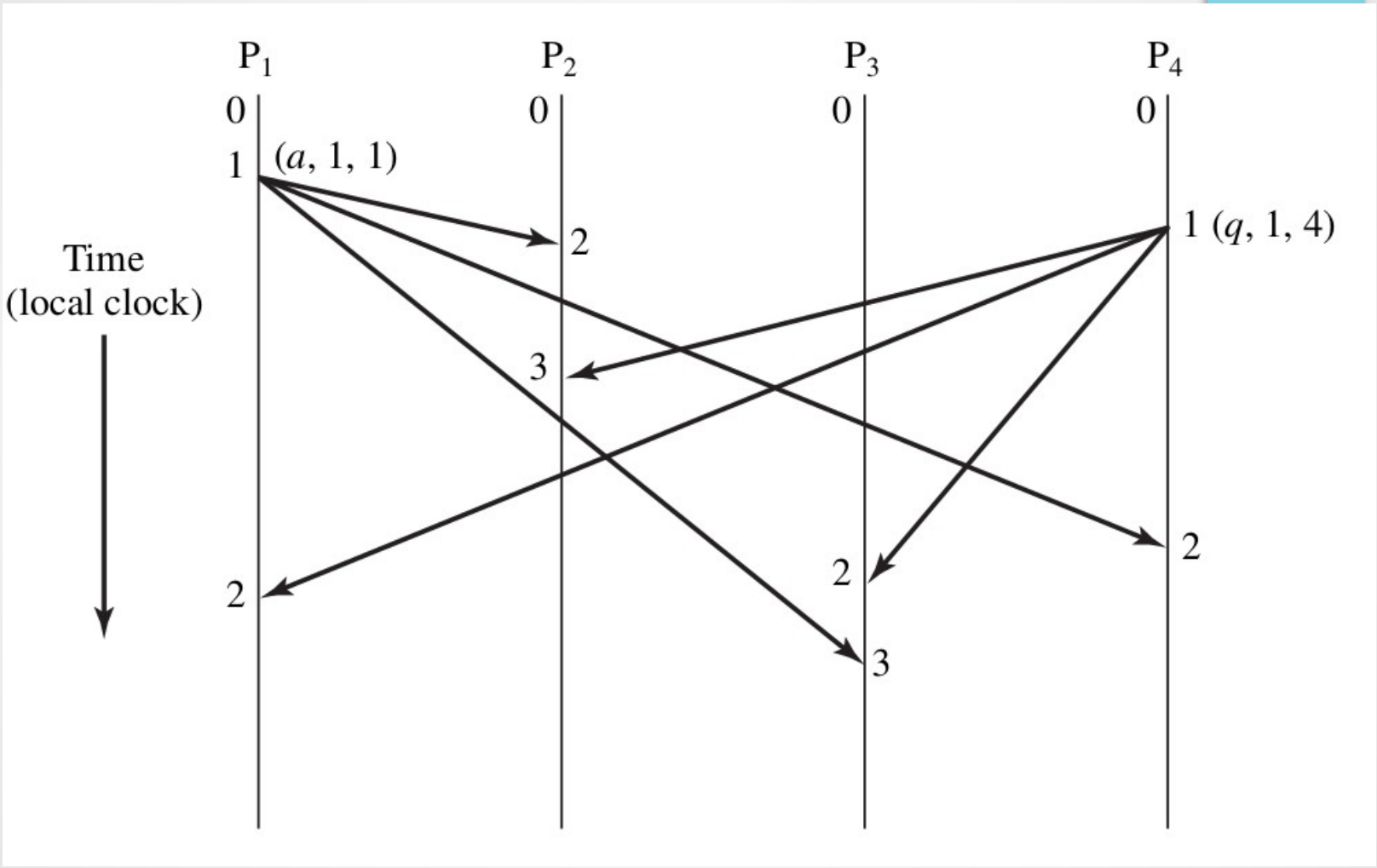
- Za poruku ***x*** sa računara ***i*** se kaže da prethodi poruci ***y*** sa računara ***j*** ako važi

  $T_i<T_j$ ili $T_i=T_j$ i $i<j$

Algoritam vremenskog žiga – Primer 1



Algoritam vremenskog žiga– Primer 2



Koji je redosled primljenih poruka u procesima P_2 i P_3?

Lamportov algoritam distribuiranog reda čekanja
Pretpostavke modela

1. Distribuirani sistem se sastoji od ***N*** čvorova. Svaki čvor sadrži proces koji zahteva uzajamno isključenje u ime svih ostalih procesa
2. Poruke se primaju u redosledu u kome su poslate
3. Sve poruke se isporučuju ispravno
4. Mreža je potpuno povezana, nema posrednika

- **Svaki čvor mora da ima kopiju istog reda čekanja**!
- Neophodno je da proces primi poruke od svih čvorova kako bi bio siguran da nema poslatih poruka koje su još na putu (primer sa slike sa prethodnog slajda)
- Svaki čvor ima svoj red čekanja (niz) čiji član ***q[j]*** sadrži poruku od procesa P_j.
- Poruke mogu biti:
  - ***(Zahtev,T_i,i)*** - Zahtev da proces P_i koristi neki resurs
  - ***(Odgovor,T_j,j)*** - Proces P_j dozvoljava pristup resursu kojim upravlja
  - ***(Prepuštanje,T_k,k)*** - Proces P_k prepušta resurs koji mu je prethodno dodeljen

Lamportov algoritam dist. reda čekanja
Verzija 1

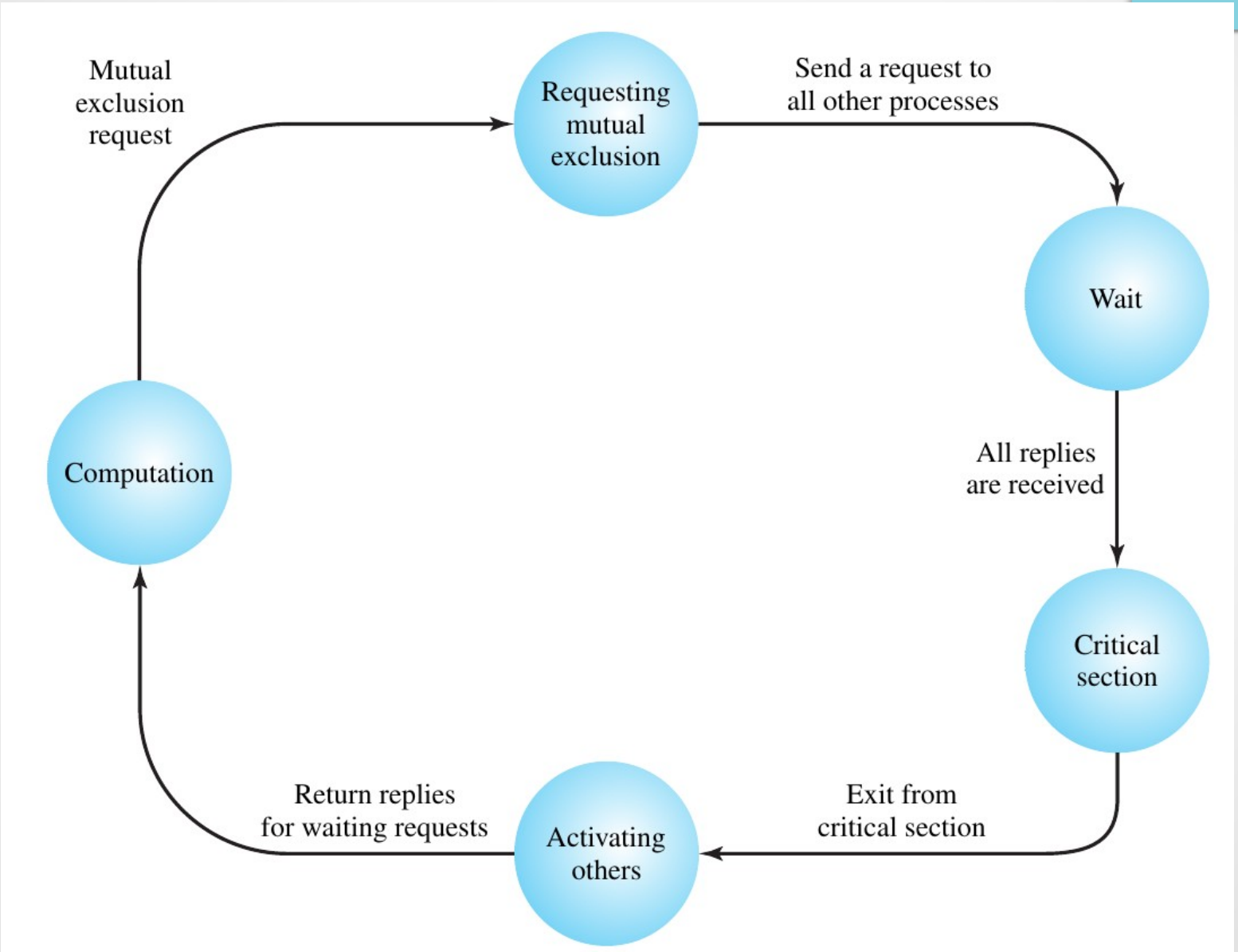
1. Kada proces P_i zahteva pristup do nekog resursa, izdaje zahtev ***(Zahtev,T_i,i)***. Ovaj zahtev smešta kao ***q[i]*** i šalje poruku svim ostalim procesima.
2. Kada proces P_j primi poruku ***(Zahtev,T_i,i)***, smešta je u svoj niz na mesto ***q[j]***. Ako P_j ne sadrži zahtev, odmah šalje ***(Odgovor,T_j,j)***.
3. P_i može da pristupi resursu ako su ispunjena oba uslova:
   - Sopstvena poruka sa zahtevom je najranija poruka sa zahtevom u ***q***. Pošto su sve poruke poređane istim redosledom, ovim pravilom se dopušta da samo jedan proces ima pristup resursu.
   - Sve ostale poruke u ***q*** su kasnije u odnosu na poruku ***q[i]***. Ovo garantuje da proces zna za sve zahteve koji prethode njegovom zahtevu.
4. P_i prepušta resurs izdavanjem poruke ***(Prepuštanje,T_i,i)***. Poruka se kopira u ***q[i]*** i šalje ostalima.
5. *Kada* P_i primi poruku ***(Prepuštanje,T_j,j)***,*ona* se kopira u ***q[j]***.
6. *Kada* P_i primi poruku ***(Odgovor,T_j,j)***,*ona* se kopira u ***q[j]***.

Ovim algoritmom se postiže uzajamno isključenje, pravednost (po redosledu vremenskih žigova), nema uzajamnog blokiranja i nema praznog hoda (slanjem poruke *Prepuštanje*).

Lamportov algoritam dist. reda čekanja Verzija 2

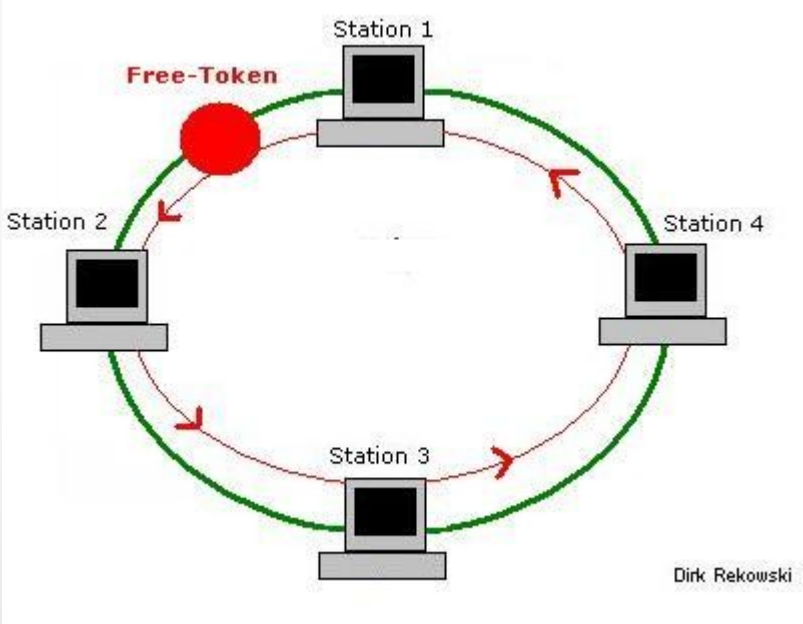
1. Kada proces P_i zahteva pristup do nekog resursa, izdaje zahtev ***(Zahtev,T_i,i)***. *Ovaj zahtev smešta kao* ***q[i]*** i šalje poruku svim ostalim procesima.
2. Kada proces P_j primi poruku ***(Zahtev,T_i,i)***, on radi po sledećim pravilima*:*
   - *Ako je* P_j u svojoj k.s. odlaže slanje odgovora.
   - Ako P_j ne čeka da uđe u svoju k.s. šalje poruku ***(Odgovor,T_j,j)***.
   - Ako P_j čeka da uđe u svoju k.s. i ako pristigla poruka kasni u odnosu na njegov sopstveni zahtev, on poruku smešta u ***q[i]*** i odlaže slanje odgovora.
   - Ako P_j čeka da uđe u svoju k.s. i ako pristigla poruka prethodi njegovom sopstvenom zahtevu, on poruku smešta u ***q[i]*** i šalje ***(Odgovor,T_j,j)***.
3. P_i može da pristupi resursu kada primi poruke sa odgovorom od svih ostalih procesa.
4. Kada P_i napusti svoju k.s, on prepušta resurs tako što šalje poruke sa odgovorom svim procesima koji imaju zahteve na čekanju.

Lamportov algoritam distribuiranog reda čekanja



Uzajamno isključenje sa prosleđivanjem tokena

- Samo proces koji ima token može da uđe u svoju kritičnu sekciju
- Token je, u stvari, niz tokena u čiji je *k*-ti element upisan vremenski žig sa vremenom kada je token poslednji put bio kod procesa P_k.
- Svi procesi održavaju niz sa zahtevima u čiji je *j*-ti element upisan vremenski žig sa vremenom poslednjeg zahteva od procesa P_j.



Uzajamno isključenje sa prosleđivanjem tokena

```
if (!token_present) {
    clock++;                                    /* Prelude */
    broadcast (Request, clock, i);
    wait (access, token);
    token_present = true;
}
token_held = true;
<critical section>;

token[i] = clock;                               /* Postlude */
token_held = false;
for (int j = i + 1; j < n; j++) {
    if (request(j) > token[j] && token_present) {
         token_present = false;
         send (access, token[j]);
    }
}
```

(a) First part

```
if (received (Request, k, j)) {
    request (j) = max(request(j), k);
    if (token_present && !token_held)
      <text of postlude>;
}
```

(b) Second part

```
Notation
    send (j, access, token)        end message of type
                                   access, with token, by
                                   process j
    broadcast (request, clock, i)  send message from
                                   process i of type
                                   request, with time-
                                   stamp clock, to all
                                   other processes
    received (request, t, j)       receive message from
                                   process j of type
                                   request, with time-
                                   stamp t
```