

# Distribuirana Sinhronizacija

- **Distribuirana sinhronizacija procesa**
- **Međusobno Isključenje**
- **Atomatske Transakcije**
- **Konkurentne Atomske Transakcije**
- **Upravljanje Zastojima**
- **Election Algoritmi**

# Distribuirana Sinhronizacija Procesa

## ■ Happened-before relacija ( označena kao $\rightarrow$ )

➤ Ako su **A** i **B**

➤ događaji istog procesa,

➤ i

➤ **A** je izvršen pre **B**,

➤ onda **A**  $\rightarrow$  **B**

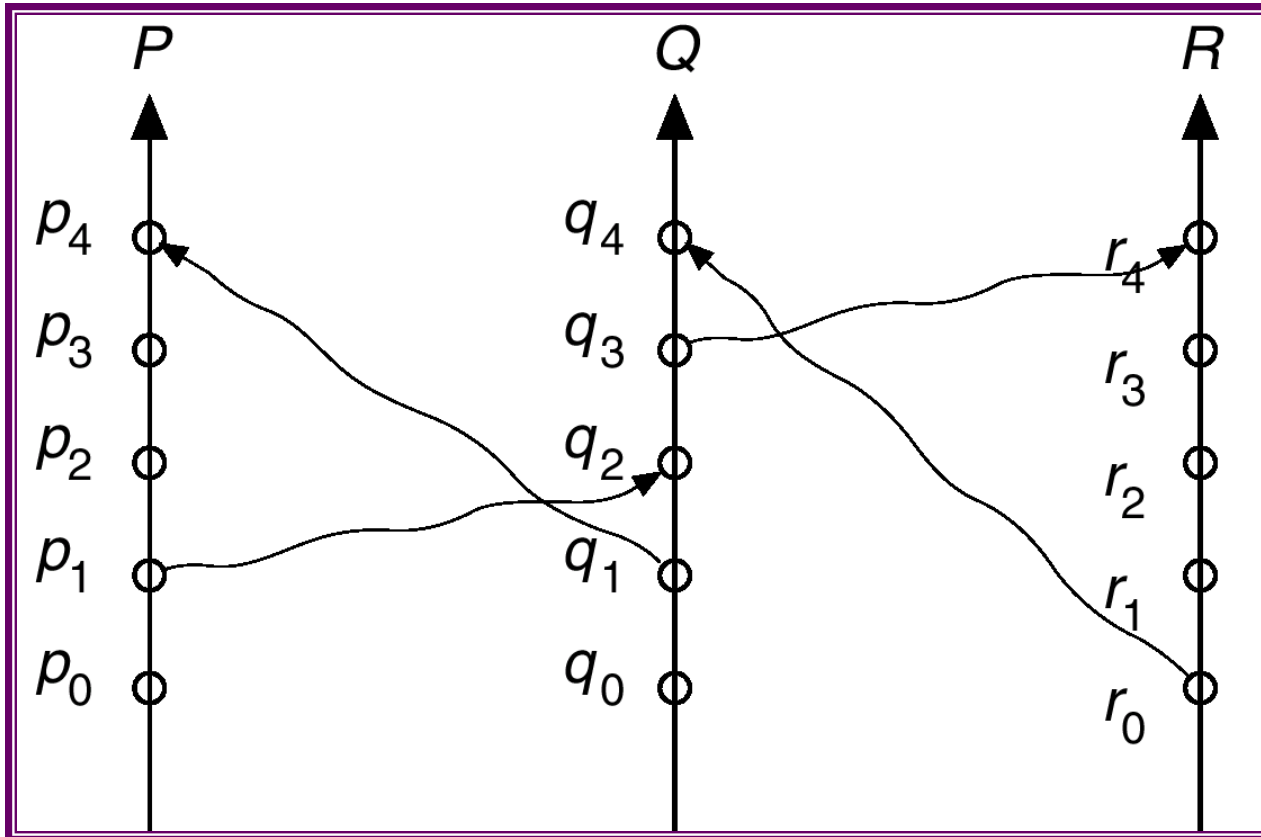
➤ Ako je **A**:događaj slanje poruke jednog procesa

➤ **B** je događaj primanja poruke od strane drugog procesa,

➤ onda **A**  $\rightarrow$  **B**

➤ Ako **A**  $\rightarrow$  **B** i **B**  $\rightarrow$  **C** onda **A**  $\rightarrow$  **C**

# Povezano Vreme za Tri Konkurentna Procesa



# Implementacija →

- Dodeljivanje **vremenske oznake (timestamp)** svakom sistemskom događaju
- Zahteva da svaki par događaja  $A$  i  $B$ ,
  - ☞ ako  $A \rightarrow B$ ,
  - ☞ onda
  - ☞ vremenska oznaka za  $A$  je manja od vremenske oznake za  $B$ .
- Unutar svakog procesa  $P_i$ 
  - ☞ se dodeljuje **logički časovnik, LC**
- **Logički časovnik** može biti implementiran:
  - ☞ kao **jednostavan brojač**
  - ☞ koji se inkrementira
  - ☞ između
  - ☞ bilo koja **dva uspela događaja**
  - ☞ izvršena unutar procesa
  - ☞ (na završetak prvog)

# Implementacija →

- Proces pokreće svoj **logički sat**
  - ☞ **kada primi poruku**
  - ☞ **čija vremenska oznaka TS je veća**
  - ☞ **od trenutne vrednosti njegovog logičkog sata**
- Ako su vremenske oznake dva događaja **A i B iste**,
  - ☞ onda su događaji **konkurentni**
- Možemo da koristimo **identifikacione brojeve procesa**
  - ☞ da prekinemo veze
  - ☞ i
  - ☞ da stvorimo totalno uređenje (**total ordering**)

# Distributed Mutual Exclusion (DME)

## ■ Pretpostavke:

- ☞ Sistem se sastoji od **n procesa**;
  - 📄 svaki proces  $P_i$  se izvršava
  - 📄 na **različitom procesoru**
- ☞ Svaki proces ima kritičnu sekciju
  - 📄 koji zahteva međusobno isključenje

## ■ Uslov

- ☞ Ako  $P_i$ , izvršava svoj kritičnu sekciju
- ☞ onda
- ☞ ni jedan proces  $P_j$  nemože da izvršava svoju kritičnu sekciju

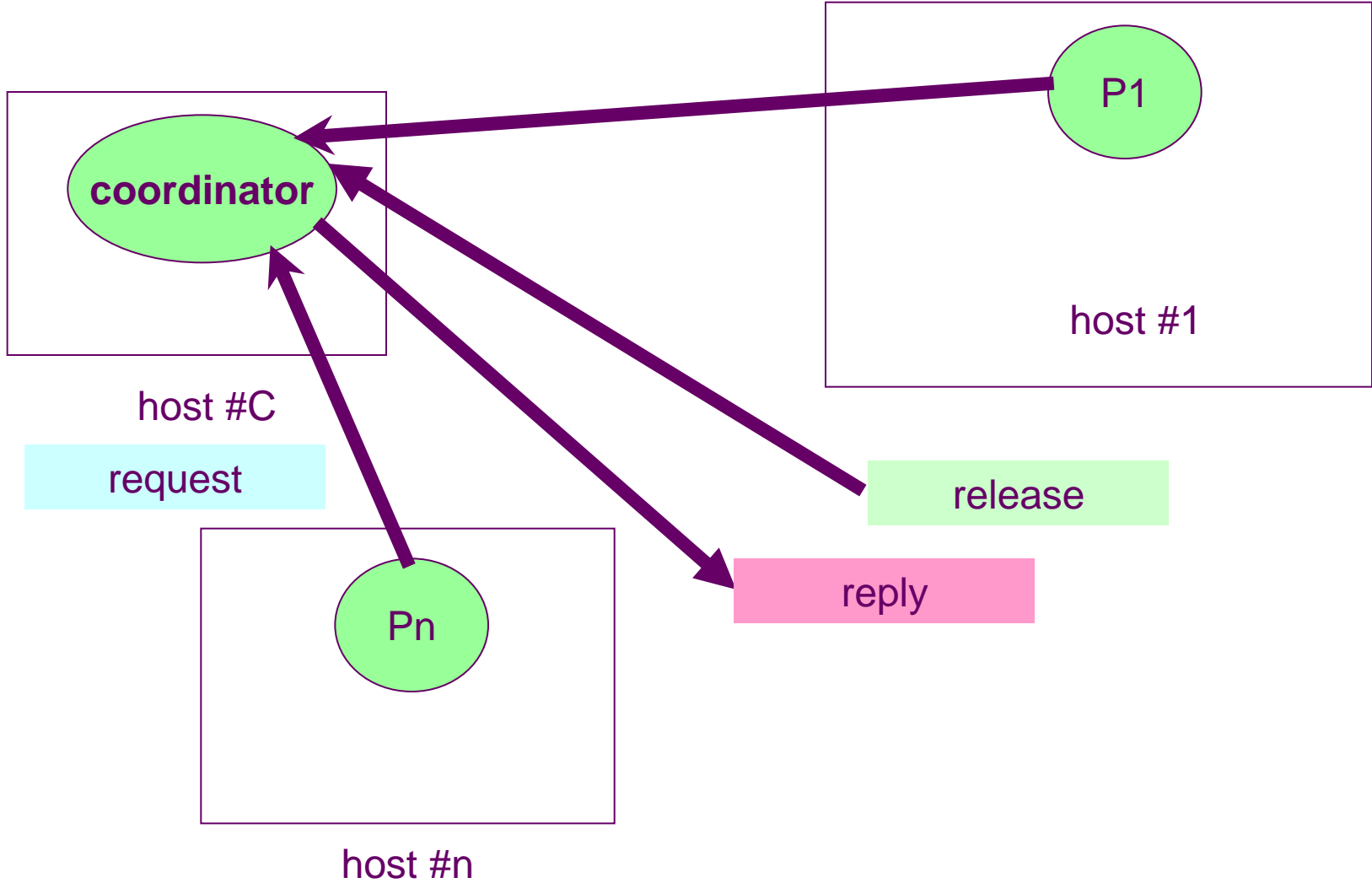
## ■ dva algoritma za **DME**

- ☞ Centralizovano rešenje (centralized approach)
- ☞ Distribuirano rešenje (distributed approach)

# DME: Centralizovano rešenje

- **Jedan od procesa** unutar sistema
  - ☞ je izabran
  - ☞ **da koordinira pristup** kritičnim sekcijama
- **1. Proces**
  - ☞ koji želi da uđe u svoj kritični deo
  - ☞ šalje *request* poruku koordinatoru.
- **2. Koordinator odlučuje**
  - ☞ koji proces može da uđe u kritičnu sekciju,
  - ☞ šalje poruku tom procesu *reply* poruku.
- **3. Kada proces primi**
  - ☞ *reply* poruku od koordinatora,
  - ☞ on ulazi u svoju kritičnu sekciju.
- **4. Nakon izlaska iz kritičnog dela,**
  - ☞ proces šalje
  - ☞ *release* poruku koordinatoru
  - ☞ nastavlja sa izvršavanjem
- **Ova šema zahteva tri poruke po ulasku u kritičan deo:**
  - ☞ **request**
  - ☞ **reply**
  - ☞ **release**

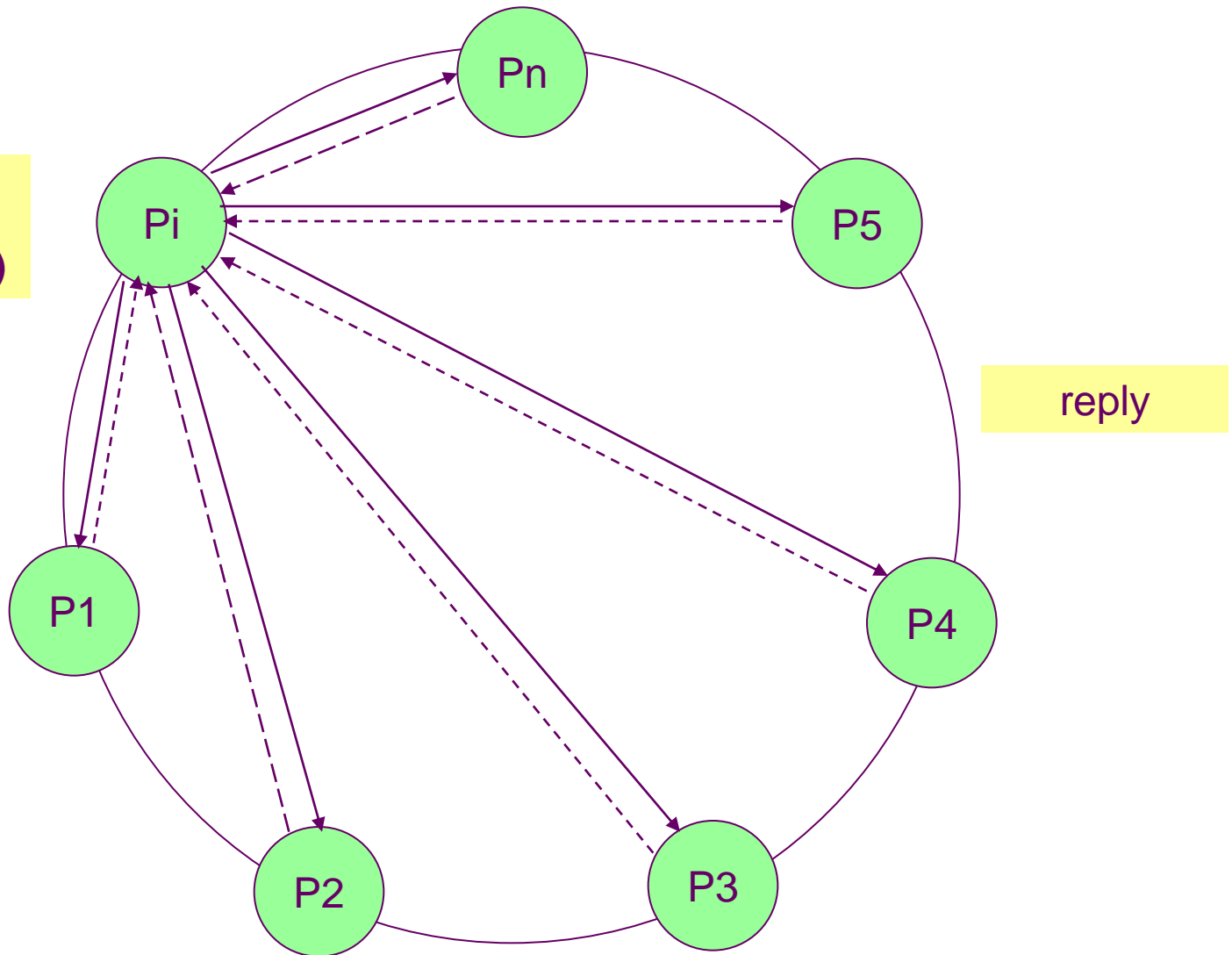
# DME: Centralizovano rešenje



# DME: Puno Distribuirano Rešenje

- Kada proces  $P_i$  želi da uđe u svoju kritičnu sekciju,
  - ☞ on stvara new vremensku oznaku,  $TS$
  - ☞ šalje poruku **request** ( $P_i$ ,  $TS$ )
  - ☞ svim drugim procesima u sistemu.
- Kada proces  $P_j$  primi **request** poruku,
  - ☞ može odmah da odgovori
  - ☞ ili
  - ☞ može da odloži slanje odgovora.
- Kada proces  $P_i$  primi **reply** poruku
  - ☞ od svih ostalih procesa u sistemu,
  - ☞ može da uđe u svoju kritičnu sekciju
- Nakon izlaska iz kritične sekcije,
  - ☞ proces šalje **reply** poruku

# DME: Puno Distribuirano Rešenje



# DME: Puno Distribuirano Rešenje(Odluka)

## ■ Odluka da li proces $P_j$

- ☞ odgovara istog trenutka na **request**( $P_i$ , **TS**) poruku
- ☞ ili
- ☞ da odloži svoj odgovor se zasniva na tri faktora:

## ■ 1. Out of CS, do not want-in

- ☞ Ako  $P_j$  ne želi da uđe u svoju kritičnu sekciju,
- ☞ on šalje odgovor odmah ka  $P_i$

## ■ 2. In-CS

- ☞ Ako se  $P_j$  nalazi u svojoj kritičnoj sekciji,
- ☞ onda on odlaže odgovor ka  $P_i$ .

## ■ 3. Out of CS, want-in

- ☞ Ako  $P_j$  želi da uđe u svoju kritičnu sekciju
- ☞ ali još nije ušao,
- ☞ onda poredi svoju vremensku oznaku sa vremenskom oznakom **TS**.
  - 📄 ako je njegova vremenska oznaka veća od **TS**,
  - 📄 onda šalje odgovor istog trenutka ka  $P_i$  ( $P_i$  je prvi tražio)
  - 📄 U suprotnom, odgovor se odlaže

# Poželjno Ponašanje Punog Distribuiranog Rešenja

- **Oslobodeno od zastoja (Deadlock)**
- **Oslobodeno od zakucavanja (starvation)**
  - ☞ nakon ulaska u kritičnu sekciju
  - ☞ se raspoređuje
  - ☞ prema rasporedu vremenskih oznaka
- **Raspored vremenskih oznaka osigurava**
  - ☞ ti procesi se opslužuju FIFO način:
  - ☞ prvi ušao, prvi opslužen (FIFO)
- **broj poruka** po ulasku u kritičnu sekciju je

$$2 \times (n - 1).$$

Ovo je minimalan broj

- ☞ **zahtevanih poruka za ulazak u kritičnu sekciju**
- ☞ **kad se proces ponaša nezavisno i konkurentno**

# Tri Neželjene Posledice

- **1. Proces mora da zna**
- **identitet svih ostalih procesa u sistemu,**
  - ☞ što čini
  - ☞ dinamičko dodavanje i uklanjanje procesa
  - ☞ još komplikovanijim
- **2. Ako jedan proces otkáže,**
  - ☞ onda cela šema propada.
  - ☞ Ovo može da bude prevaziđeno
  - ☞ konstantnim monitoring-om
  - ☞ stanja svih procesa u sistemu.
- **3. Procesi koji nisu ušli u svoje kritične delove**
  - ☞ moraju se često pauzirati (zbog slanja poruka)
  - ☞ da bi osigurali ostale procese
  - ☞ one koji nameravaju da uđu u kritičan deo.
- **Ovaj protokol je dakle zadovoljavajući**
  - ☞ za male,
  - ☞ stabilne setove kooperativnih procesa.

# Atomske Transakcije

## ■ **Transakcija =**

- ☞ Kolekcija instrukcija
- ☞ koja čini jednu logičku funkciju

## ■ **U sistemima baza podataka**

- ☞ beleže čitanje
- ☞ beleže upisivanje

## ■ **Kraj transakcije = status**

- ☞ izvršeno - commit (uspešna transakcija)
- ☞ prekinuto - abort

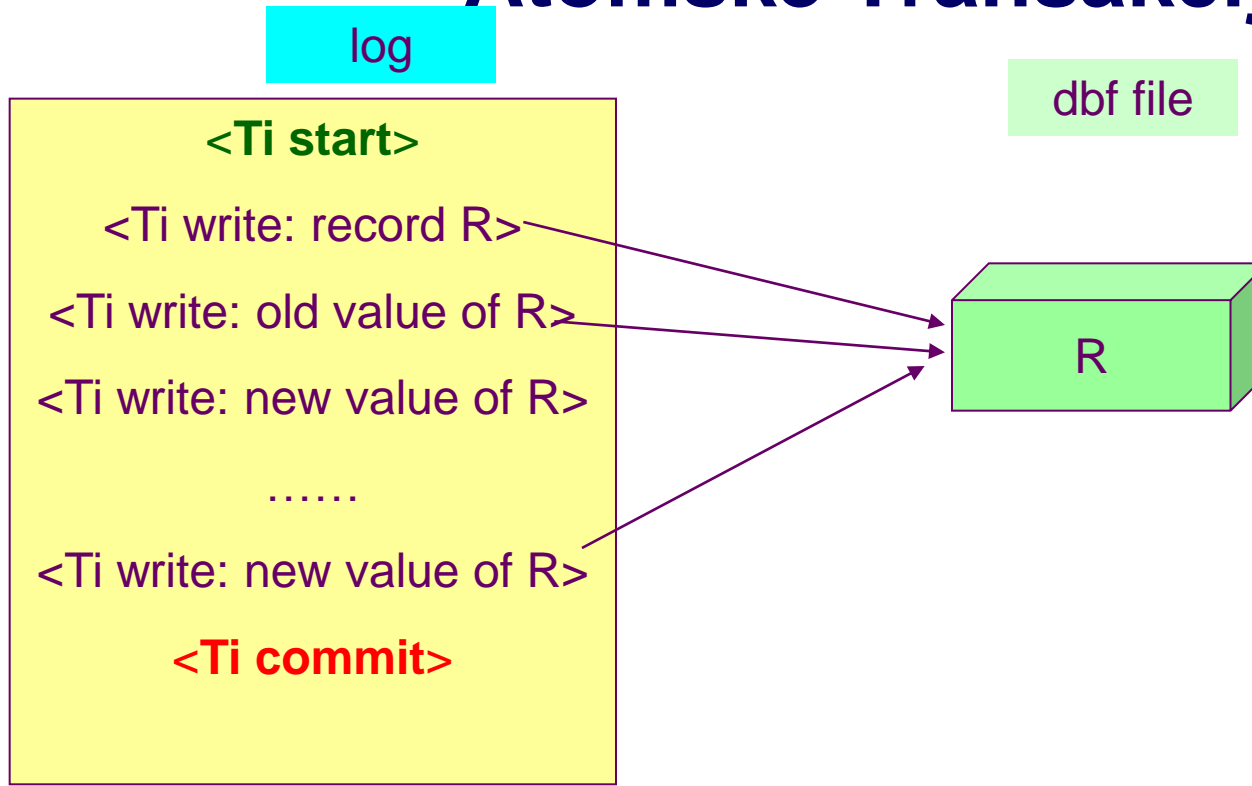
## ■ **Atomska transakcija**

- ☞ u slučaju greške =>T treba da se vrati na početak (roll-back)

## ■ **log implementacija**

- ☞ log zapis za T sadrži:
  - 📄 ime T
  - 📄 ime zapisa za upis (R)
  - 📄 stare vrednosti R (pre upisivanja)
  - 📄 nove vrednosti R (nakon upisivanja)

# Atomske Transakcije-Log



## ■ Nakon pada sistema => Atomska transakcija

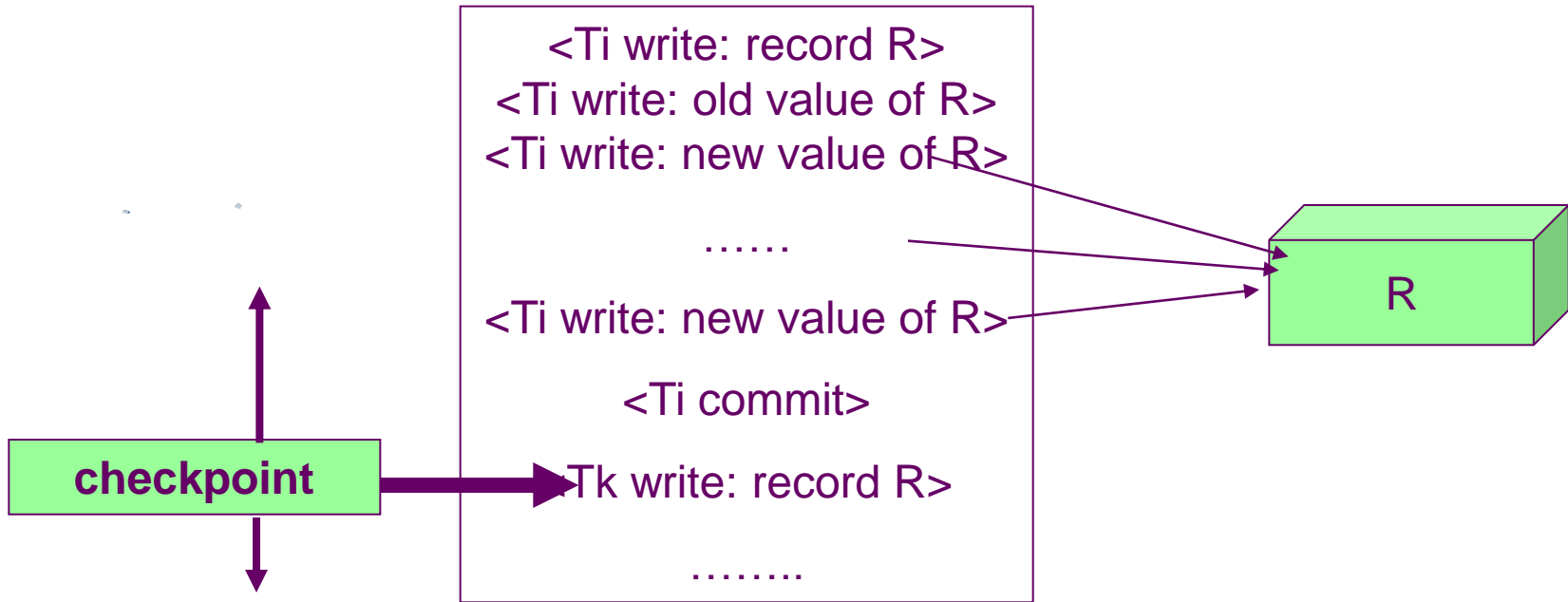
☞ **undo(T)** if not exist **<commit T>**

📄 vraća na stanje R pre Ti (**roll-back**)

☞ **redo (T)** if exist **<commit T>**

📄 upisuje poslednje vrednosti R preko T da bi zapisao R

# Atomske Transakcije - Checkpoint



■ **Checkpoint = sve uspele transakcije**

■ **nakon pada -> traži checkpoint**

☞ **za sve Tk sa <Tcommit>**

☞ radi **redo(Tk)**

☞ **za sve Tk bez <Tcommit>**

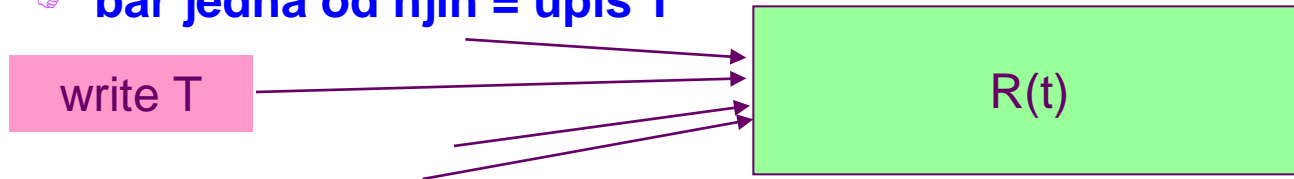
☞ radi **undo(Tk)**

# Konkurentne Atomske Transakcije

- **Conflict mode** transakcija
- **minimum dve** transakcije  $T_i, T_j$

☞ za isto  $R$ ,

☞ bar jedna od njih = upis  $T$



- **Dve šeme:**
  - ☞ **lock - protocoli za zaključavanje**
  - ☞ **TS redosled izvršavanja**

# Lock Protokol

## ■ T će da zaključa (lock) R pre korišćenja

☞ zahtev za lock

☞ koristi R

📄 nakon što je odobren lock

☞ Otključavanje R (release a lock)

## ■ Shared lock (samo za čitanje)

☞ posle odobrenog zaključavanja,

📄 Ti će da koristi R, **samo za čitanje**

☞ nekoliko deljenih zaključavanja istovremeno

## ■ Exclusive lock (za upisivanje)

☞ posle odobrenog zaključavanja,

📄 Ti može da upisuje u R

☞ **samo jedno ekskluzivno zaključavanje istovremeno**

# TS ordering

- **log based + TS**
- **TS (timestamp protokol obezbeđuje)**
  - ☞ da konfliktno čitanje i upis obave u TS poretku,
  - ☞ koja isključuje preklapanje konflikta
- **TS funkcioniše na sledeći način:**
- **Svakoj transakciji dodeljujemo vremensku oznaku TS,**
  - ☞ a to je vreme kada počinje se izvršava.
- **Takođe svakom zapisu dodeljujemo dva vremenska parametra:**
  - ☞ **W-timestamp(Q)**
    - ☞ predstavlja vreme poslednje transakcije
    - ☞ koja je uspešno obavila upis u Q
  - ☞ **R-timestamp(Q)**
    - ☞ prestavlja vreme poslednje transakcije
    - ☞ koja je uspešno obavila čitanje iz Q
- **Ova dva parametra se stalno ažuziraju**

# TS protocol-reading case

- transakcija  $T_i$  pošalje zahtev  $\text{read}(Q)$ .
- Moguće su dve situacije:
  1.  $\text{TS}(T_i) \geq \text{W-TS}(Q)$ 
    - ☞ tada je zahtev korektan
    - ☞ čitanje se obavlja iz  $Q$
    - ☞ a posle toga se ažurira  $\text{R-timestamp}(Q)$
  2.  $\text{TS}(T_i) < \text{W-TS}(Q)$ ,
    - ☞  $T$  traži vrednost  $Q$  iz nekog prošlog vremena,
    - ☞ a vrednost je prepisana
    - ☞ čitanje iz  $Q$  odbaci
    - ☞ cita se iz log zapisa
    - ☞ Ili **roll-back za sve koje su pogresne**

# TS protocol-writing case

- transakcija  $T_i$  pošalje zahtev  $write(Q)$ .
- Moguće su 3 situacije:
  - 1.  $TS(T_i) < R-TS(Q)$ 
    - ☞ pokušava da se upiše nešto
    - ☞ što je već trebalo da bude pročitano.
    - ☞ roll-back za onu  $T_k$  koja je procitala pogresno, restart za  $T_k$
  - 2.  $TS(T_i) < W-TS(Q)$ ,
    - ☞ tada transakcija pokušava da upiše
    - ☞ staru vrednost za  $Q$
    - ☞ upis ide u log
    - ☞ ili roll-back za pogresne, restart
  - 3. U svim ostalim slučajevima
    - ☞  $TS(T_i) > R-TS(Q)$  i  $TS(T_i) > W-TS(Q)$
    - ☞ upis u  $Q$  se obavlja

# Timestamping

- **Stvara jedinstvenu vremensku oznaku u distribuiranoj šemi:**

- ☞ **Svaki sajt generiše**

- 📄 **jedinstvenu lokalnu vremensku oznaku.**

- ☞ **globalna jedinstvena vremenska oznaka se koristi**

- 📄 **vezivanjem**

- 📄 **the **unique local timestamp****

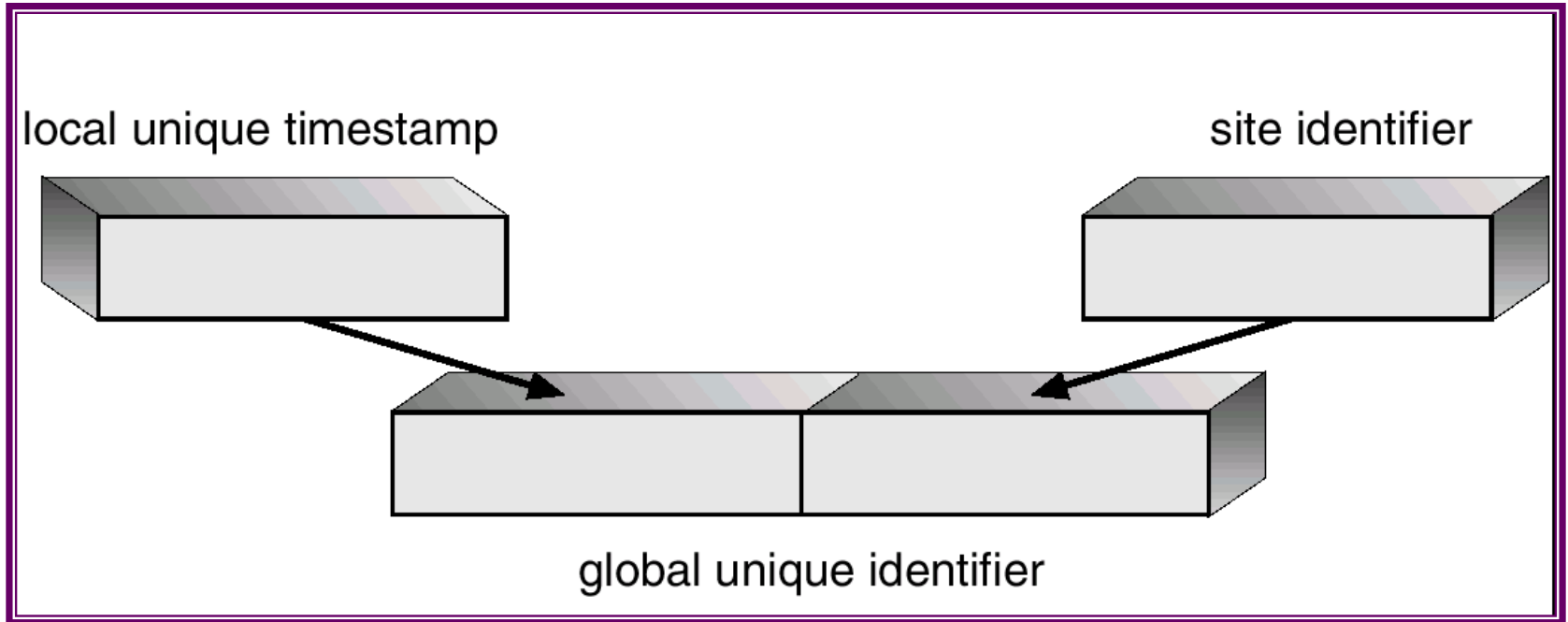
- 📄 **with the **unique site identifier****

- ☞ **Use a logical clock defined within each site**

- 📄 **to ensure**

- 📄 **the fair generation of timestamps.**

# Generation of Unique Timestamps



# Vremanska Oznaka – Šema Rasporeda

- Ova šema kombinuje
  - ☞ centralizovanu timestamp metodu
  - ☞ sa 2PC protokolom
  - ☞ **da obezbedi poredak**
  - ☞ **bez velikog broja kaskadnih roll-back operacije.**
- Baferuju (log) se različiti zahtevi za čitanje i upis
- a onda se izvršavaju preko timestamp poretka, koji glasi:
- **reading:**
- **record x**      $T_i \{read(x)\}$ 
  - ☞ tada **read(x) mora biti odloženo**
  - ☞ ako postoji transakcija  $T_j \{write(x)\}$  i ako je  $TS(T_j) < TS(T_i)$
- **writing:**
- **record x**      $T_i \{write(x)\}$ 
  - ☞ tada **write(x) mora biti odloženo**
  - ☞ ako postoji transakcija  $T_j \{read(x)\}$  i ako je  $TS(T_j) < TS(T_i)$

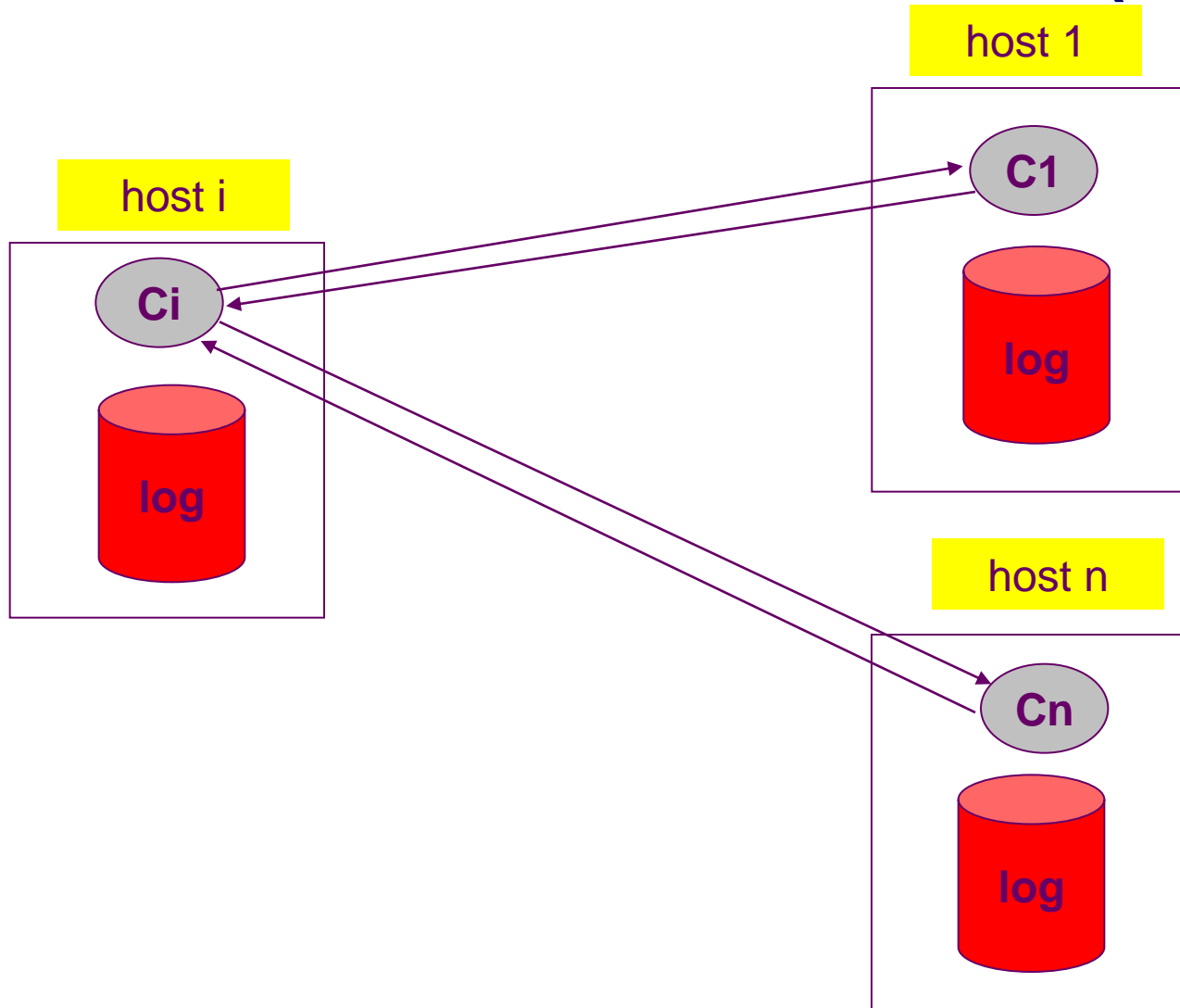
# Atomičnost Distribuiranih Sistema

- Osiguravanje atomičnosti u distribuiranim sistemima
- zahteva **koordinatora transakcije**
- koji je **odgovoran za sledeće:**
  - ☞ **startovanje izvršenja transakcije**
  - ☞ **deljenje transakcije na više podtransakcija**
  - ☞ **distribucija tih podtransakcija**
    - 📄 **na odgovarajuća mesta za izvršenje**
  - ☞ **Koordiniranje završetkom transakcije**
  - ☞ **čiji rezultat može da bude:**
    - 📄 **transakcija izvršena na svim mestima**
    - ili
    - 📄 **prekinuta na svim mestima**

# Two-Phase Commit Protocol (2PC)

- Preuzima **fail-stop** model
- Izvršenje protokola se inicijalizuje
  - ☞ od strane koordinatora
- Kada je protokol inicijalizovan,
  - ☞ transakcija može da se izvršava
  - ☞ na nekim lokalnim mestima
- Protokol obuhvata
  - ☞ sva lokalna mesta
  - ☞ na kojima se izvršava transakcija
- Primer:
  - ☞ Neka **T** bude transakcija
  - ☞ inicijalizovana na **mestu  $S_i$**  i
  - ☞ neka koordinator transakcije na  **$S_i$  bude  $C_i$**

# Two-Phase Commit Protocol (2PC)



# Faza 1: Dodavanje Odluka

- $C_i$  dodaje **<prepare T>** zapis
  - ☞ za log
- $C_i$  šalje **<prepare T>** poruku
  - ☞ Svim mestima.
- **Mesto K:**
  - ☞ Kada sajt primi **<prepare T>** poruku,
  - ☞ upravljač(menager) transakcije odlučuje
  - ☞ da li može da izvrši transakciju.
  - ☞ **Ako ne može (neće da radi):**
    - 📄 dodaje **<no T>** zapis u log
    - 📄 odgovara  $C_i$  sa **<abort T>** porukom.
  - ☞ **Ako može (hoće da radi):**
    - 📄 dodaje **<ready T>** zapis u log.
    - 📄 izbacuje sve log zapise za **T** u stabilnu memoriju-log
    - 📄 šalje **<ready T>**poruku do  $C_i$

# Faza 1 (Nastavak)

## ■ Koordinator sakuplja odgovore

- ➡ 1. Ako su svi odgovori “**ready**”, => odluka je commit (**izvrši**)
- ➡ 2. Bar jedan odgovor “**abort**”, => odluka je **prekini**
- ➡ 3. Bar jedan učesnik neuspe da odgovori
- ➡ unutar time out perioda, => odluka je **prekini**

# Faza 2: Zapisivanje Odluke u Bazu

- **Koordinator dodaje zapis o odluci**

- ☞ **<abort T>**

- ☞ ili

- ☞ **<commit T>**

- **u svoj log**

- **izbacuje zapis u stabilnu memoriju-log**

- **Kada zapis stigne do stabilne memorije**

- ☞ on je neopoziv

- ☞ (čak i ako se desi neuspeh prilikom upisa)

- **Koordinator šalje poruku**

- ☞ **svakom učesniku**

- ☞ **obaveštavajući ga**

- ☞ **o odluci (commit ili abort).**

- **Učesnici izvode određenu radnju, lokalno.**

# Faza 2: Zapisivanje Odluke u Bazu

## ■ abort T

- ☞ koordinator šalje poruku
- ☞ svakom učesniku
- ☞ obaveštavajući ga o odluci (**abort**)

## ■ commit T

- ☞ koordinator šalje poruku
- ☞ svakom učesniku
- ☞ obaveštavajući ga o odluci (**commit**)

## ■ Učesnici izvode određenu radnju lokalno

- ☞ nakon završetka,
- ☞ učesnik šalje poruku
- ☞ **<acknowledge T>** koordinatoru Ci

## ■ Nakon svega, Ci dodaje zapis **<complete T>**

# Manipulisanje Neuspesima u 2PC – Site Failure

- **Site failed and recover, look at the log**
- Log sadrži **<commit  $T$ >** zapis.
  - ☞ U ovom slučaju, sajt izvršava **redo( $T$ )**.
- Log sadrži **<abort  $T$ >** zapis.
  - ☞ U ovom slučaju, sajte izvršava **undo( $T$ )**.
- Log sadrži **<ready  $T$ >** zapis; konsultuje  $C_i$ 
  - ☞ Ako je  $C_i$  pao-otkazao,
  - ☞ sajt šalje **<query-status  $T$ >** poruku drugim sajtovima
- **Log ne sadrži kontrolne zapise** koji se tiču  $T$ 
  - ☞ U ovom slučaju,
  - ☞ sajt izvršava **undo( $T$ )**

# Manipulisanje Neuspesima u 2PC – Koordinator failed $C_i$

- Koordinator failed  $C_i$
- Čekanje na oporavak  $C_i$
- Ako aktivno mesto (sajt)
  - ☞ sadrži <commit  $T$ > zapis u svom logu
  - ☞  $T$  mora da se izvrši
- Svi aktivni sajtovi
  - ☞ imaju <ready  $T$ > zapis u svojim logovima,
  - ☞ ali bez dodatnih kontrolnih zapisa.
  - ☞ u ovom slučaju moramo da sačekamo koordinatora da se oporavi.
    - 📄 **Blocking** problem:
    - 📄  $T$  je blokiran do oporavka sajta  $S_i$
- Ako nema oporavka  $C_i$ 
  - ☞ sve transakcije izvršavaju **undo()**

# Locking Protokoli

- mogu da koriste
- **2PC locking protokol**
- u distribuiranom okruženju
- menjanjem načina na koji je lock menadžer implementiran
  
- **Šema bez replikacije**
  - ☞ samo jedna replika
  - ☞ svaki sajt održava lokalni **lock menadžer**
  
- **LM upravlja lock i unlock zahtevima**
  - ☞ za one stavke podataka
  - ☞ koje su smeštene na tom sajtu
- **Jednostavna implementacija obuhvata**
  - ☞ dva transfera poruka za manipulisanje lock zahtevima,
    - 📄 lock request, lock acknowledge
  - ☞ jedan transfer poruka za manipulisanje unlock zahtevima.
  
  - ☞ Manipulisanje zastojima je još komplikovanije.

# Locking Protokoli - Šeme sa Replikacijama

- **Metod sa jednim koordinatorom**
- **Metod sa više koordinatora**

# Metod sa Jednim Koordinatorom

- **Centralizovani lock menadžer**
- **Jedan lock menadžer je smešten na jedan izabrani sajt,**
  - ☞ svi lock i unlock zahtevi
  - ☞ čine taj sajt.
- **Jednostavna implementacija**
- **Jednostavna manipulacija zastojsima**
- **mogućnost uskog grla**
- **Podložan gubitku sinhronizacije procesa**
  - ☞ ako jedan sajt otkáže
- **Moguć pad sistema**

# Metod sa Više Koordinatora

- distribuira lock meadžer funkciju
- preko
- više sajtova.
  
- **Majority protocol** – Protokol većine
  
- **Biased protocol** – Pristrasni protokol
  
- **Primary copy** – Primarna kopija

# Majority Protocol

- **LM na svakom sajtu**
  - ☞ koji kontroliše svoje lokalne podatke i njihove replike
- **kada transakcija trazi lock(Q)**
  - ☞ koji repliciran na n različitih sajtova
  - ☞ transakcija mora da pošaje lock zahtev
  - ☞ **za više od  $n/2$  sajtova**
- **Svaki sajt odlučuje**
  - ☞ hoće li da ispuni lock zahtev
  - ☞ a transakcija je dobiti dozvolu
  - ☞ ako većina dozvoljava
- **Dobra osobina ove šeme je**
  - ☞ što se poštuje većina
  - ☞ izbegava centralizovanu kontrola
- **Međutim postoje 2 loše osobine:**
  - ☞ **komplikovanija je za realizaciju**
  - ☞ **zastoji se mnogo teže kontrolišu**

# Biased Protocol

- Ovaj metod je sličan većinskom protokolu
  - ☞ ovde postoji LM za svaki sajt,
- **ali deljivi i ekskluzivni lock se opslužuju drugačije**
  - ☞ lock zahtevi za deljivi lock, favorizuju
  - ☞ u odnosu na zahteve za ekskluzivni lock.
- **Deljivi lock** zahtevi (svaki zahtev za čitanje) su takvi
  - ☞ da kada transakcija traži lock Q zapisa,
  - ☞ on se **upućuje samo jednom sajtu** koji sadrži repliku od Q
- **Ekskluzivni lock** zahtevi (svaki zahtev za upis) su takvi
  - ☞ da kada transakcija traži lock Q zapisa,
  - ☞ on se **upućuje svim sajtovima** koji sadrži repliku od Q
- Šema ima prednosti u odnosu na većinski protokol
  - ☞ ima manje kašnjenje (overhead) na operacije čitanja
    - 📄 u odnosu na većinski protokol, jer su lock zahtevi deljivi
  - ☞ ali za ciklus upisa je lošiji
    - 📄 jer mora da se obrati svakom sajtu koji sadrži repliku

# Primarna Kopija

- U slučaju replika,
  - ☞ jedan od sajtova koji sadrzi **repliku datoteke**
  - ☞ ce se proglasiti za **njegovu primarnu repliku** (primary site).
- Uvek se lock zahtev prosleđuje
  - ☞ samo za primarnu repliku (tom sajtu),
  - ☞ čiji će sajt obezbediti konkurentnu lock kontrolu
- Slično jednoj **LM metodi**
- Ali ako primarni sajt otkáže,
  - ☞ tada će Q zapis da budu neraspoloživ,
  - ☞ bez obzira što postoji ostale replike na različitim računarima.

# Sprečavanje Zastoja

- Raspoređivanje resursa - **sprečavanje zastoja:**
- definisanje globalnog raspoređivanja resursa sistema
  - ☞ **Zadavanje jedinstvenog broja svim resursima sistema**
  - ☞ **Proces može da zahteva resurs sa jedinstvenim brojem**
  - ☞ **samo ako već ne poseduje resurs sa jedinstvenim brojem većim od traženog**
  - ☞ **Jednostavno za implementaciju; zahteva male troškove**
- **Banker's algoritam:**
- **označava jedan proces u sistemu**
- **kao proces koji sadrži informaciju**
- **neophodnu za sprovođenje Banker's algoritma**
  - ☞ **takođe se lako implementira**
  - ☞ **ali može da zahteva mnogo više troškova**

# Timestamped Deadlock-Šema Prevenzije

- Svaki proces  $P_i$  je označen
  - ☞ jedinstvenim brojem prioriteta
- Brojevi prioriteta se koriste za odlučivanje da li
  - ☞ proces  $P_i$  treba da sačeka proces  $P_j$ ;
  - ☞ inače za  $P_i$  se obavi roll-back
- Šema sprečava deadlock ako:
  - ☞ Za svaku ivicu  $P_i \rightarrow P_j$
  - ☞ u wait for grafiku,
  - ☞  $P_i$  ima veći prioritet od  $P_j$ .
- U tom slučaju krug je nemoguć

# Wait-Die Šema

- Zasnovana na **non-preemptive** tehnici
- Ako  $P_i$  zahteva resurs
- koji trenutno drži  $P_j$ ,
  - ☞  $P_i$  je **dozvoljeno da čeka** samo
  - ☞ ako ima **manju vremensku oznaku** nego  $P_j$
  - ☞ ( $P_i$  je stariji od  $P_j$ )
- Inače, za  $P_i$  se obavi **roll-back** (umire)
- Primer: Pretpostavimo da procesi  $P_1$ ,  $P_2$ , i  $P_3$
- imaju vremenske oznake **5**, **10**, i **15** redom.
  - ☞ ako  $P_1$  zahteva resurs koji drži  $P_2$ , onda će  $P_1$  da čeka.
  - ☞ ako  $P_3$  zahteva resurs koji drži  $P_2$ , onda će  $P_3$  biti roll-backed

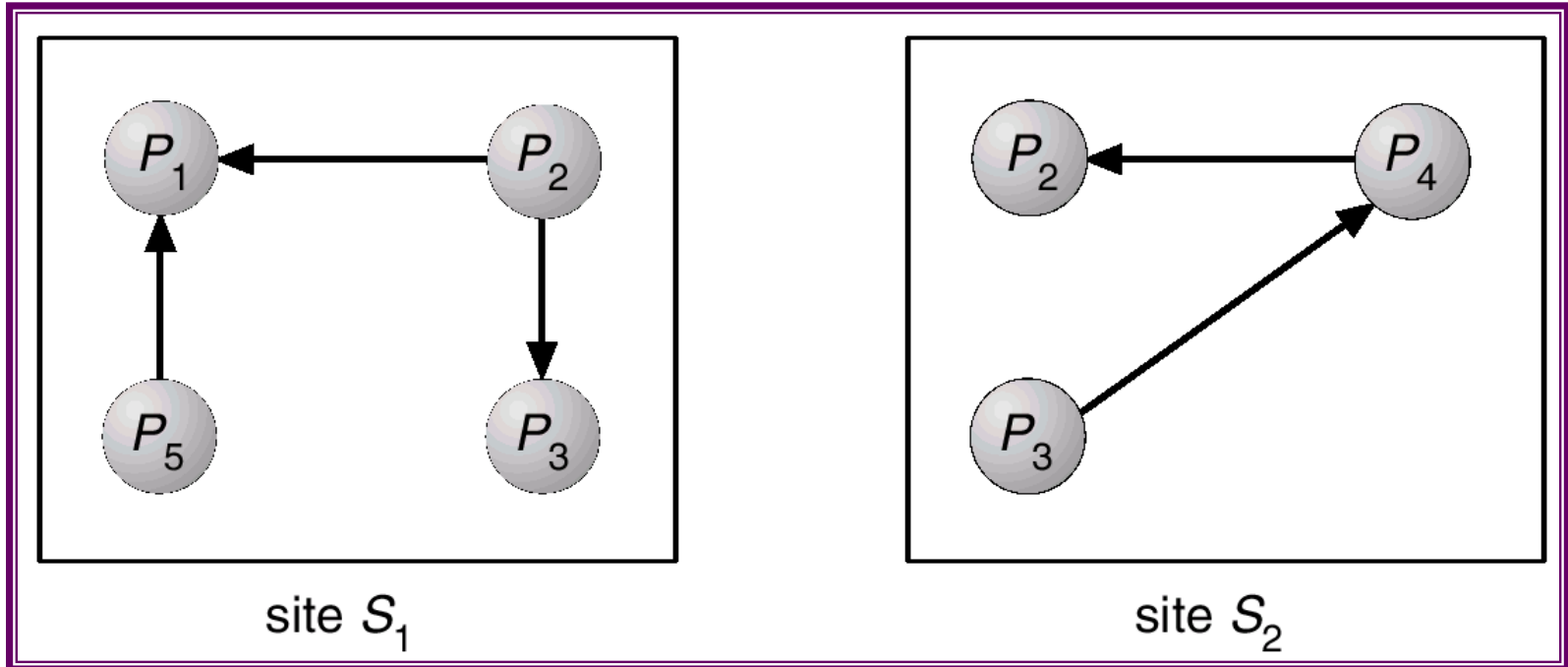
# Would-Wait Šema

- Zasnovana na **preemptive** tehnicima; kopija wait-die sistema
- Ako  $P_i$  zahteva resurs
- koji drži  $P_j$ ,
  - ☞  $P_i$  je dozvoljeno da čeka samo
  - ☞ ako ima veću vremensku oznaku nego  $P_j$
  - ☞ ( $P_i$  je mlađi od  $P_j$ )
- Inače za  $P_j$  se obavlja **roll-back** ( $P_j$  je **preempt** od strane  $P_i$ ).
- Primer: Pretpostavimo da procesi  $P_1$ ,  $P_2$ , i  $P_3$
- imaju vremenske oznake **5**, **10**, i **15** redom.
  - ☞ ako  $P_1$  zahteva resurs koji drži  $P_2$ ,
    - 📄 onda će resurs biti oduzet od  $P_2$
    - 📄  $P_2$  će biti **roll-back**
  - ☞ ako  $P_3$  zahteva resurs koji drži  $P_2$ , onda će  $P_3$  čekati

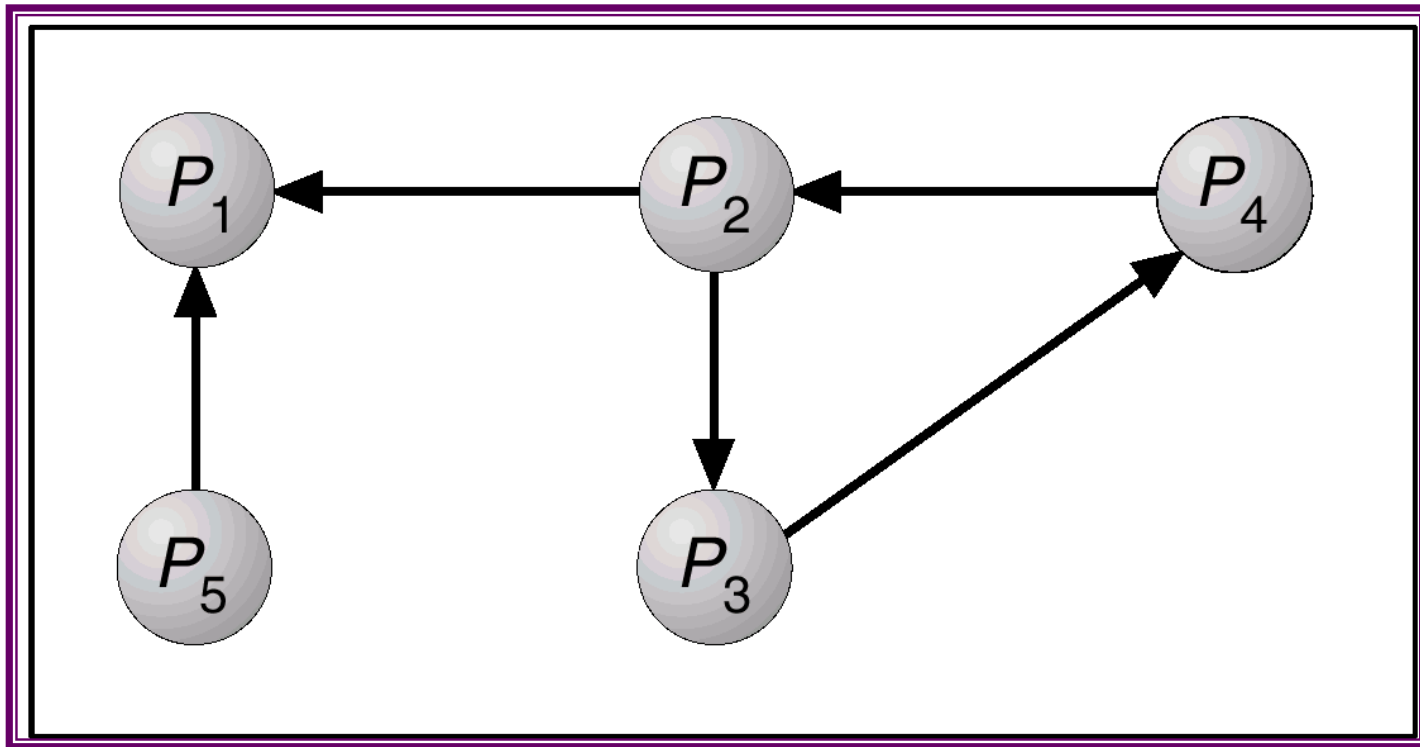
# Detekcija zastoja

- **wait for graf**
- **Centralizovano rešenje**
- **Puno distribuirano rešenje**

# Dva Lokalna Wait-For Grafika



# Globalni Wait-For Grafik



# Detekcija Zakočenja – Centralizovano Rešenje

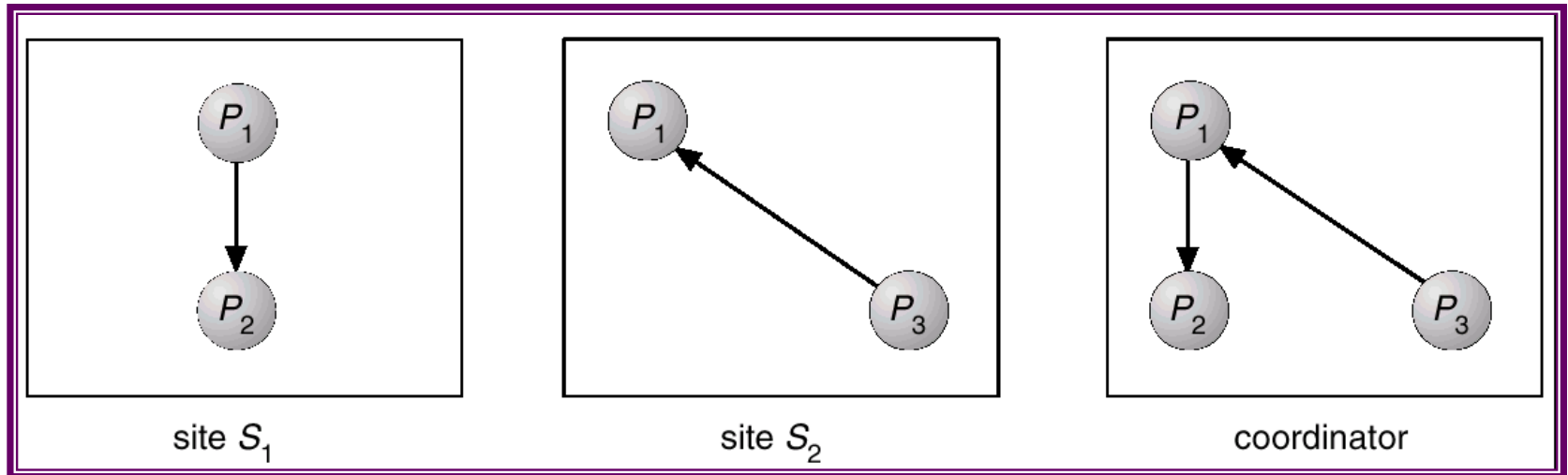
- Svaki sajt čuva *lokalni wait-for graf*
- čvorovi grafa odgovaraju
  - ☞ svim procesima
  - ☞ od kojih bilo ko može da drži ili zahteva
  - ☞ bilo koji resurs koji je lociran na tom sajtu
- **Globalni wait-for graf** je onaj koji se kreira
  - ☞ u *jednom procesu koordinacije*;
  - ☞ ovaj grafik je unija svih lokalnih wait-for grafika

# Detekcija zastoja – Centralizovano Rešenje

- Postoje **tri različite opcije** (tačke u vremenu)
- kada **wait-for graf** može stvori:
  1. Kad god se ubaci ili izbaci nova ivica u neki od lokalnih wait-for grafa
  2. Periodično, kada se neki broj promena dogodio u wait-for graph
  3. Kad god koordinator treba da pozove cycle-detection algoritam..

Mogu da se **dogode nepotrebni rollback-ovi** kao rezultat *false cycles*

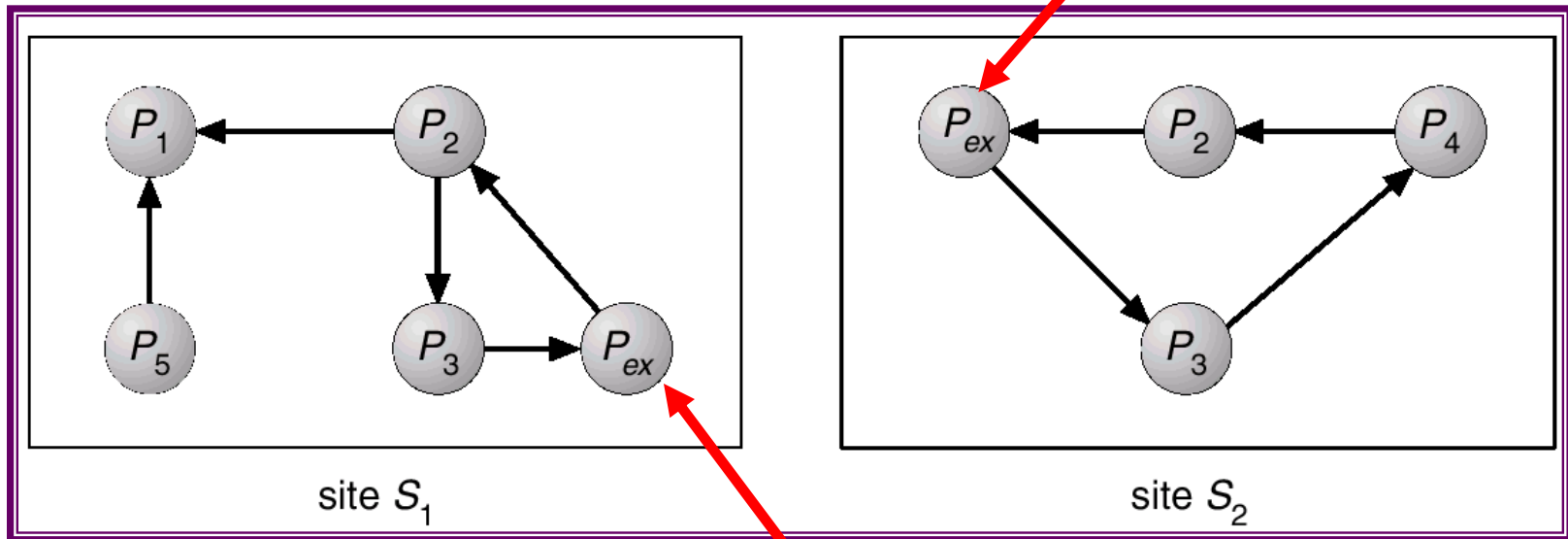
# Lokalni i Globalni Wait-For Grafici



# Potpuno Distribuirano Rešenje

- **Svi kontroleri dele podjednako**
  - ☞ odgovornost detektovanje zakočenja
- **Svaki sajt obrazuje wait-for graph**
  - ☞ koji predstavlja deo ukupnog grafika.
- **Uvodimo dodatni čvor  $P_{ex}$** 
  - ☞ za svaki lokalni wait-for graf
- **Ako lokalni wait-for grafik sadrži krug**
  - ☞ koji ne sadrži čvor  $P_{ex}$ ,
  - ☞ onda je sistem u stanju zakočenja
- **Ako ciklus sadrži cvor  $P_{ex}$  to podrazmeva**
  - ☞ mogućnost zakočenja
- **Da bi se saznalo da li postoji zakočenje,**
  - ☞ **distributed deadlock-detection algoritam**
  - ☞ mora da se pozove

# Uvećani Lokalni Wait-For Grafovi



# Election Algoritmi

## ■ Determine

- ☞ Gde nova kopija koordinatora

- ☞ treba da bude restart

## ■ Preuzma jedinstveni broj prioriteta

- ☞ koji je vezan za svaki aktivni proces u sistemu,

- ☞ broj prioriteta od procesa  $P_i$ , je  $i$ .

## ■ Preuzima jedan-na-jedan korespodenciju

- ☞ između procesa i sajtova.

## ■ Koordinator je uvek proces

- ☞ sa najvećim brojem prioriteta.

## ■ Kada koordinator neuspe,

- ☞ algoritam mora da izabere

- ☞ onaj aktivni proces sa najvećim brojem prioriteta.

## ■ Dva algoritma,

- ☞ **bully algorithm i ring algorithm,**

- ☞ mogu da se koriste pri izboru novog koordinatora u slučaju neuspeha.

# Bully Algoritam

- **Primenljiv kod sistema**
  - ☞ gde svaki proces može da pošalje poruku
  - ☞ svakom drugom procesu u sistemu.
- **Ako proces  $P_i$  šalje zahtev**
  - ☞ na koji koordinator nije odgovorio
  - ☞ unutar vremenskog intervala  $T$ ,
  - ☞ pretpostavlja da je koordinator pao;
  - ☞  $P_i$  pokušava da izabere samog sebe
  - ☞ kao novog koordinatora.
- **$P_i$  šalje election (biranje) poruku**
  - ☞ svakom procesu sa većim brojem prioriteta,
  - ☞  $P_i$  zatim čeka sa bilo koji od ovih procesa
  - ☞ odgovori unutar intervala  $T$ .

# Bully Algoritam (Nastavak)

- **Ako nema odgovora unutar  $T$ ,**
  - ☞ pretpostavlja da su svi procesi sa brojevima većim od  $i$  pali;
  - ☞  $P_i$  postavlja sebe kao novog koordinatora.
- **Ako primi odgovor,**
  - ☞  $P_i$  započne vremenski interval  $T'$ ,
  - ☞ čeka da primi poruku da je proces sa većim brojem prioritata izabran.
- **Ako nije poslata poruka unutar  $T'$ ,**
  - ☞ pretpostavlja da proces sa višim brojem nije uspeo;
  - ☞  $P_i$  će da restartuje algoritam

# Bully Algoritam (Nastavak)

- Ako  $P_i$  nije koordinator,
  - ☞ onda, bilo kad za vreme izvršenja,
  - ☞  $P_i$  može da primi jednu od sledeće dve poruke od procesa  $P_j$ .
  - ☞ 1.  $P_j$  je novi koordinator ( $j > i$ ).
  - ☞ 2.  $P_j$  započeo biranje ( $j > i$ ).
    - 📄  $P_i$ , sends a response to  $P_j$  and
    - 📄 započinje sopstveni election algoritam,
    - 📄 ako taj  $P_i$  nije već inicijalizovao takav izbor.
- Nakon što se neuspeli proces povрати,
  - ☞ odmah počinje izvršenje istog algoritma.
- ako nema aktivnog procesa sa većim brojevima,
  - ☞ povraćeni proces izbacuje sve procese sa manjim brojevima
  - ☞ da bi postao koordinator procesa,
  - ☞ čak i ako već postoji koordinator sa manjim brojem.

# Ring Algoritam

- **Primenjiv na sistemima koji su organizovani kao prsten (ring) (logički ili fizički).**
- **Pretpostavlja da su linkovi sjedinjeni,**
  - ☞  $i$
  - ☞ da procesi šalju svoje poruke
  - ☞ svom desnom susedu.
- **Svaki proces sadrži *aktivnu listu*,**
  - ☞ u kojoj se nalaze svi brojevi prioriteta
  - ☞ svih aktivnih procesa u sistemu
  - ☞ kada se algoritam završi.
- **Ako proces  $P_i$  detektuje neuspeh koordinatora,**
  - ☞ Stvara novu aktivnu listu koja je na početku prazna.
  - ☞ zatim šalje poruku *elect( $i$ )* svom desnom susedu,
  - ☞  $i$
  - ☞ dodaje broj  $i$  svojoj aktivnoj listi.

# Ring Algoritam (Nastavak)

## ■ Ako $P_i$ primi poruku $\text{elect}(j)$

☞ od procesa sa leve strane,

☞ mora da odgovori na jedan od tri načina:

1. Ako je ovo prva  $\text{elect}$  koja je viđena ili poslata,  $P_i$  stvara novu aktivnu listu sa brojevima  $i$  i  $j$ .  
zatim šalje poruku  $\text{elect}(i)$ ,  
a posle nje poruku  $\text{elect}(j)$ .
2. Ako  $i \neq j$ ,  
onda aktivna lista za  $P_i$  sada sadrži brojeve svih aktivnih procesa u sistemu.  
 $P_i$  može sada da determinira najveći broj u aktivnu listu  
da bi identifikovao novi koordinator proces.
3. Ako  $i = j$ ,  
onda  $P_i$  prima poruku  $\text{elect}(i)$ .  
Aktivna lista za  $P_i$  sadrži sve aktivne procese u sistemu.  
 $P_i$  može sada da determinira novi koordinator proces.