

Distribuirani sistemi

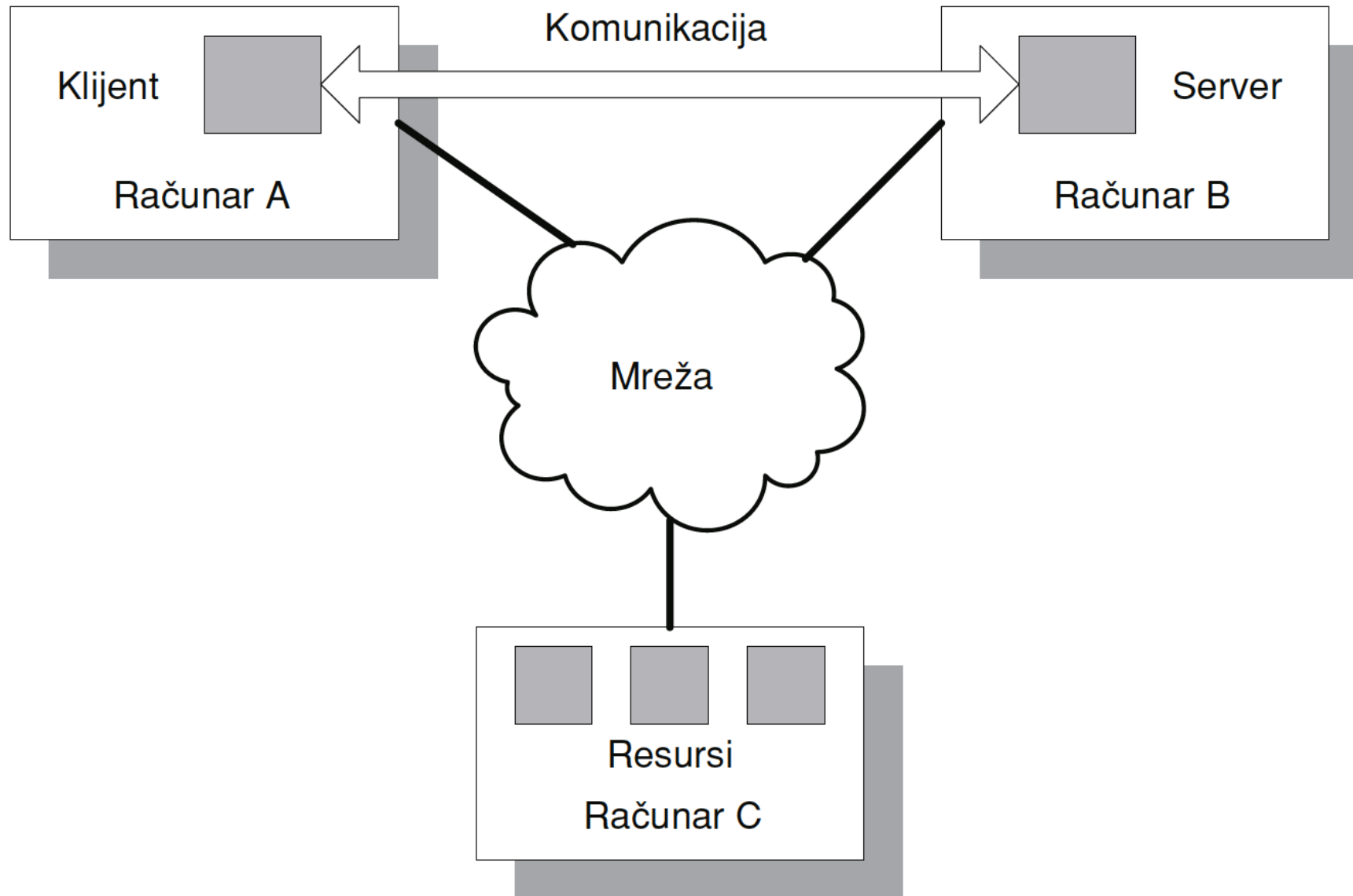
Nemanja Maček

- Uvod u distribuirane sisteme
- Tipovi mrežno-orijentisanih operativnih sistema
- Sinhronizacija procesa u distribuiranim sistemima
- Atomske transakcije u distribuiranim uslovima
- Upravljanje zastojsima u distribuiranim uslovima

- **Distribuirani sistem** (engl. *distributed system*) je kolekcija procesora odnosno računara koji ne dele zajedničku memoriju i sistemski časovnik.
- U DS-u:
 - svaki procesor, odnosno računar ima sopstvenu lokalnu memoriju,
 - međusobna komunikacija se ostvaruje putem računarske mreže LAN ili WAN tipa.
- **Osnovna namena** DS-a je da obezbedi efikasno i pogodno **deljenje resursa**.
- Najprostiji slučaj:
 - Distribuirani sistem se sastoji od servera, klijenata i sekundarnih memorija rasprostranjenih na raznim mestima.
- DS obezbeđuju korisnicima:
 - visoke performance,
 - deljivost podataka,
 - visoku pouzdanost.

- Osim podataka, datoteka i štampača, **distribuiraju se i procesi.**
- Zato se moraju obezbediti mehanizmi:
 - sinhronizacije procesa,
 - komunikacije između procesa,
 - rešavanja problema zastoja.
- Napomena: u slučaju DS-a mogu se pojaviti problemi koji nisu karakteristični za centralizovane OS!

- Procesori, odnosno računarski sistemi u DS-u se nazivaju:
 - sajt (engl. *site*)
 - čvor (engl. *node*)
 - računar (engl. *computer, machine, host*).
- U ovom izlaganju će se uglavnom koristiti termin **sajt**.
- Sa stanovišta jednog procesora:
 - drugi procesori i njihovi resursi su **udaljeni resursi** (engl. *remote resources*),
 - njegovi sopstveni resursi su **lokalni resursi** (engl. *local resources*).



- DS mogu biti realizovani kao:
 - **klijent-server sistemi**, koji se sastoje od **servera** (podataka, za izračunavanje, za štampu, itd.) i **klijenata** (računari koji koriste njihove usluge) ili
 - **ravnopravni računarski sistemi** koji po mreži dele resurse.

Karakteristike distribuiranih sistema

- **Transparentnost.**
 - DS korisniku treba da izgleda kao konvencionalni centralizovani sistem.
- **Otpornost na greške.**
 - DS treba da nastavi funkcionisanje u slučaju bilo kog otkaza.
 - DS mora da detektuje otkaz i da pronađe sajt koji će zameniti onoga ko je “ispao iz igre”.
- **Skalabilnost.**
 - Sa povećanjem zahteva DS treba lako da prihvati dodavanje novih računara i resursa.
- **Deljenje resursa.**
 - DS sistemi obezbeđuju mehanizme za deljenje datoteka, obradu informacija u distribuiranim bazama podataka, deljenje štampača i specijalizovanog hardvera.
- **Ubrzavanje izračunavanja.**
 - Može da se obavi takozvanim deljenjem opterećenja (engl. *load sharing*).
 - Proces se izdela na celine koje se obrađuju na posebnim računarima u mreži.
 - Nakon toga, rezultati parcijalnih izračunavanja se spajaju.

- U distribuiranim sistemima moguće su različite vrste hardverskih otkaza:
 - **otkaz sajta,**
 - **otkaz linka,**
 - **gubitak poruke.**
- Kako se može ustanoviti **da li je otkazao link ili host?**
 - Pretpostavka: sajtovi A i B imaju fizički link koji funkcioniše.
 - U fiksnim intervalima oba sajta šalju poruke tipa **<I-am-up>**.
 - Sajt A ne prima ovu poruku u propisnom intervalu i pretpostavlja da je nešto otkazalo.
 - Ako i dalje nema poruke sajt A šalje poruku **<Are-you-up>**.
 - Ako i dalje nema odgovora sajt A pošalje istu poruku **preko druge putanje.**
 - Ukoliko se **sajt B** javi zaključuje se da je **link otkazao.**
 - Šta možemo da zaključimo ako odgovora nema ni preko druge putanje?

- Ako i preko druge putanje nema odgovora tada se doazi do **nepreciznog zaključaka**:
 - host B otkazao,
 - prvi link otkazao,
 - drugi link otkazao ili
 - poruka je izgubljena.
- Dakle:
 - jedan od ove četiri vrste otkaza se dogodio, ali
 - ne može se ustanoviti tačno koji.

- Nakon detekcije greške otpočinje procedura za rekonfiguraciju sistema:
 - Ako je **direktni link između A i B otkazao**:
 - svaki sajt u sistemu mora da dobije tu informaciju,
 - razlog je ažuriranje tabela rutiranja.
 - Ako se pretpostavi da je **host otkazao**:
 - svaki sajt u sistemu mora da dobije tu informaciju;
 - razlog je sprečavanje pokušaja ostvarivanja konekcije sa tim sajtom.
- Oporavljeni link ili host se moraju **ponovo vratiti u sistem**.
- Šta se u tom slučaju čini?
 - Oporavljen link – link se unosi u tabele za rutiranje.
 - Oporavljen sajt – svi sajtovi se obaveštavaju da je prethodno otkazani host ponovo u funkciji.

Tipovi mrežno-orijentisanih operativnih sistema

- Mrežno-orijentisani OS se prema svojim karakteristikama mogu podeliti na:
 - mrežne OS,
 - distribuirane OS.

- Mrežni OS obezbeđuju takvu okolinu u kojoj korisnik može pristupiti udaljenim resursima na dva načina:
 - procedurom prijavljivanja na udaljeni računar (engl. **remote login**), kao što je SSH, Remote Desktop, itd.,
 - transferom datoteka sa udaljene mašine na sopstvenu (engl. **remote file transfer**), kao što je SCP.

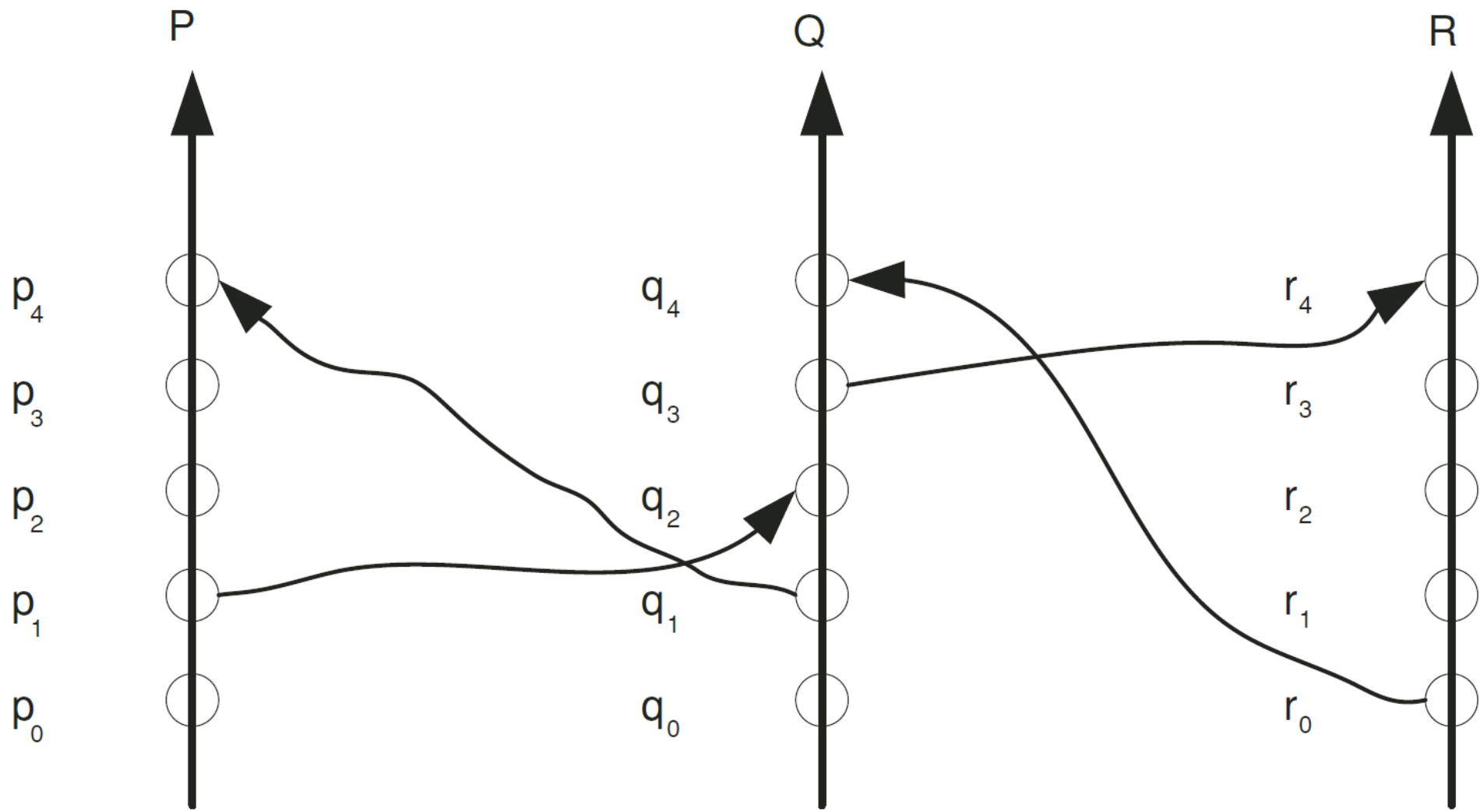
- U distribuiranim OS korisnik **pristupa udaljenim resursima kao da su lokalni.**
- Distribuirane OS karakterišu tri vrste migracija: migracija podataka, izračunavanja i procesa.
- **Migracija podataka.**
- **Migracija izračunavanja.**
 - Primer: proces inicirata obradu na više udaljenih strana a zatim sakuplja rezultate.
 - Kako se može odvijati komunikacija? RPC pozivima, slanjem poruka, itd.
- **Migracija procesa.**
 - Proširenje migracije izračunavanja.
 - Ideja: ceo proces ili neki njegovi delovi mogu se izvršavati na drugim računarima.
 - Pogodnosti:
 - balansirano opterećenje (engl. *load balancing*),
 - ubrzavanje obrade (engl. *computation speedup*),
 - izvršavanje na namenskom hardveru sa namenskim softverskom (preferenciranje),
 - brži pristup podacima.

Distribuirana sinhronizacija procesa

- Problem:
 - Veliki broj aplikacija zahteva da se poštuje redosled događaja.
 - U DS na različitim računarima postoje **različiti časovnici**.
 - Dva procesa koja se izvršavaju na različitim računarima ne mogu da odrede tačan redosled događaja.
- Kako rešavamo problem?

Distribuirana sinhronizacija procesa

- U distribuiranim sistemima uvodi se relacija ***happened-before***.
- Ova relacija pruža samo delimičnu informaciju o **redosledu događaja**.
- Relaciju *happened-before* označićemo sa \rightarrow .
- Ako je $A \rightarrow B$ tada se se A mora izvršiti pre B.
- Kada važi $A \rightarrow B$?
 - A i B su događaji istog procesa pri čemu je događaj A izvršen pre događaja B.
 - Događaj A je slanje poruke a događaj B primanje te poruke pri čemu različiti procesi šalju i primaju poruku.
- Šta je karakteristično za relaciju *happened-before*?
 - Ako je $A \rightarrow B$ i $B \rightarrow C$ tada je $A \rightarrow C$ (implikacija).
- Dva događaja A i B koji nisu povezana relacijom *happened-before* se izvršavaju **konkurentno**.



Distribuirana sinhronizacija procesa

- Sa prethodne slike vidimo:
 - Događaji $p_1 \rightarrow q_2$, $r_0 \rightarrow q_4$, $q_3 \rightarrow r_4$ itd. su povezani relacijom *happened-before*.
 - $p_1 \rightarrow r_4$ je implikacija ($p_1 \rightarrow q_2$ i $q_3 \rightarrow r_4$).
 - Neki događaji se odvijaju konkurentno (q_0, p_2), (r_0, q_3), (r_0, p_3).
 - U slučaju konkurentnih događaja ne znamo koji se dogodio prvi.

Distribuirana sinhronizacija procesa

- Redosled događaja mora se odrediti **bez fizičkih časovnika**.
- Zato ćemo svakom fizičkom događaju dodeliti **vremensku oznaku TS** (engl. *timestamp*).
- Definisaćemo relaciju *happened-before* za svaki par događaja A i B na sledeći način:
 - Ako je $A \rightarrow B$ tada je $TS(A) < TS(B)$.
- Vrednosti TS mogu se dodeliti procesima na osnovu **logičkog časovnika LC** (engl. *logical clock*).
 - Za svaki proces P_i definiše se LC_i kao **brojač događaja**.
 - Brojač događaja je monotono rastuća nenegativna vrednost.
 - Ako se događaj A desi pre događaja B tada je $LC_i(A) < LC_i(B)$.
- Šta je problem?

Distribuirana sinhronizacija procesa

- Problem se javlja u slučaju dva procesa koji se izvršavaju na **dva različita računara** sa različitim procesorima i različitim taktovima.
 - Proces P_1 šalje poruku procesu P_2 na drugoj mašini (dogadjaj A).
 - Proces P_2 prima poruku (dogadjaj B).
 - Neka je $LC_1(A) = 200$ i $LC_2(B)=195$.
 - Situacija je moguća ali ovo nije saglasno sa onim što smo hteli.
 - Ako je $A \rightarrow B$ tada $LC_1(A)$ mora biti manji od $LC_2(B)$.
- Kako se problem rešava?
 - Uvođenjem korekcije LC vrednosti procesa primaoca poruke u slučaju ovakve nesaglasnosti.
 - Proces primalac koriguje svoju LC vrednost tako što uzima LC vrednost procesa pošiljaoca uvećanu za 1.
 - U konkretnom slučaju $LC_2(B)$ mora da se koriguje i da postane $LC_2(B)=201$.

- Pretpostavimo da se sistem sastoji od n procesa i da se svaki izvršava na različitom procesoru.
- Postoje tri načina za rešavanje problema međusobnog isključenja:
 - centralizovani pristup,
 - slanje žetona,
 - puni distribuirani pristup.

- **Centralizovani pristup.**
 - Proces odredjen da odobrava ulaz u kritičnu sekciju naziva se **koordinator**.
 - Opslužuje red čekanja za ulazak u kritičnu sekciju (KS).
 - U slučaju otkaza mora se zameniti drugim.

- **Slanje žetona.**
 - Zasniva se na specijalnoj poruci – **žetonu** (engl. *token*) koji kruži između svih procesa u sistemu.
 - Procesi obrazuju logički krug.
 - Proces koji je dobio žeton, ima pravo da uđe u svoju KS.
 - Ako proces ne želi da uđe u svoju kritičnu sekciju prosleđuje ga dalje.
 - Ako je kruženje žetona jednosmerno, nema zakucavanja.
 - Mogući problem: otkaz nekog procesa.
 - Nakon toga se mora formirati novi logički krug procesa.

- **Puni distribuirani pristup.**
 - **Proces P_i** koji želi da uđe u svoju KS generiše novi *timestamp* TS i šalje svim ostalim procesima poruku *request* (P_i , TS).
 - **Proces P_j** nakon primanja poruke *request*:
 - Odlaze slanje poruke *reply* za kasnije ukoliko se nalazi u KS.
 - Odgovara porukom *reply* ako nije u KS i nema nameru da uđe u nju.
 - Ako proces P_j ima nameru da uđe u KS:
 - Uporediće TS sa procesom koji traži dozvolu.
 - Ako je $TS(P_j) > TS(P_i)$ poslaće *reply* poruku
 - U suprotnom se slanje poruke odlaze.
 - Proces P_i može ući u svoju KS kad primi *reply* poruke od svih drugih procesa u sistemu.
 - Kada napusti svoju KS proces šalje *reply* poruku svim procesima koji su mu se obratili.
 - Šta je problem ove šeme?
 - Proces mora da poznaje sve procese u sistemu i da prati dodavanje i uklanjanje procesa iz sistema.

- **Transakcija** je kolekcija instrukcija koje obavljaju jednu logičku funkciju.
- Transakcija obuhvata čitanje ili upis podataka.
- Transakcija se završava se sa dve moguće operacije:
 - **Commit**: označava da je transakcija obavila svoje izvršavanje uspešno
 - **Abort**: označava da je transakcija završila svoje izvršavanje neuspešno (greška).
- Od transakcija se traži da se obavljaju **atomski**.
 - Transakcija se ili izvršava ili se ne izvršava.
 - Stanje sistema u slučaju greške ili prekida transakcije mora biti svedeno na stanje sistema pre izvršavanja transakcije.
 - To se naziva **povratak unazad** (engl. *rollback*).

- Da bi se obezbedila atomska priroda transakcije mora se voditi evidencija o svim koracima upisa!
- To se obavlja preko **dnevnika transakcija** smeštene na stabilnoj memoriji.
- Svaki **zapis u dnevniku** opisuje jednu operaciju upisa u transakciji i sadrži sledeća polja:
 - ime transakcije,
 - ime polja za upis,
 - stare podatke koji predstavljaju vrednost podataka pre operacije upisa,
 - nove podatke koji predstavljaju vrednost nakon operacije upisa.

- Pre nego što transakcija *Ti* počne izvršavanje u dnevnik se upiše zapis **<Ti starts>**.
- U toku svake transakcije pre svake operacije upisa mora se upisati jedan zapis u dnevnik.
- Poslednji zapis u dnevniku je **<Ti commit>**.
- Transakcija *Ti* izvršena na ovaj način **može se poništiti!**
- Kako?

- Uvode se dve operacije za rad sa transakcijama koje koriste dnevnik:
 - ***undo (Ti)*** koja vraća sve vrednosti koje je promenila transakcija na stare vrednosti
 - ***redo (Ti)*** koja postavlja sve vrednosti koje je transakcija promenila na nove vrednosti.
- Ako se dogodi otkaz sistema zbog havarije *log* tehika omogućava dve situacije:
 - **Ako ne postoji zapis <Ti commit>**
 - Transakcija *Ti* se nije završila.
 - Izvršava se operacija *undo (Ti)*.
 - **Ako postoji zapis <Ti commit>**
 - Transakcija *Ti* se završila.
 - Treba je imati u oporavljenom sistemu.
 - Izvršava se operacija *redo (Ti)*.

- Kako se rad sa transakcijama može učiniti efikasnijim u slučaju otkaza i regeneracije sistema?
- U dnevnik se uvodi **kontrolna tačka** (oznaka *checkpoint*).
- **Kontrolna tačka** određuje sve transakcije završene na stabilnom medijumu.
 - Sve što se u dnevniku nalazi pre kontrolne tačke je u redu.
 - Sve što se nalazi iza nje treba obraditi na sledeći način:
 - Operacija *redo* (*Tk*) izvršava se za svaku *Tk* koja sadrži zapis *<Tk commit>*.
 - Operacija *undo* (*Tk*) izvršava se za svaku *Tk* koja ne sadrži zapis *<Tk commit>*.

Konkurentne atomske transakcije

- Svaka transakcija mora da bude atomska.
- Transakcije se izvršavaju konkurentno sa svojim procesima.
- Pravilan redosled izvršenja može se obezbediti izvršavanjem svake transakcije **u celosti unutar kritične sekcije**.
 - Ovaj način izvršavanja je isuviše restriktivan.
 - Serijsko raspoređivanje ne može dovesti do konfliktnih odnosa.
 - Povlači strogo serijsko nekonkurentno raspoređivanje transakcija i umanjuje performanse.
- Uvode se specijalne tehnike kojima se:
 - reguliše konkurentnost transakcija,
 - sprečavaju konflikti.

Konkurentne atomske transakcije

- **Konkurentne operacije** u transakcijama mogu biti u:
 - konfliktnom odnosu,
 - nekonfliktnom odnosu.
- Konkurentne operacije su **u konfliktnom odnosu** ukoliko se:
 - odnose na isti zapis,
 - pri tome je bar jedna operacija upis.
- Ukoliko transakcije preklapaju svoja izvršenja:
 - performanse se povećavaju,
 - mogući su konfliktni odnosi između operacija.
- **Konfliktne operacije ne smeju se izvršavati istovremeno!**
- Transakcije se mogu preklapati samo ako se izbegnu konflikti između operacija što se postiže:
 - protokolom za zaključavanje,
 - poredkom izvršavanja na bazi vremenske oznake (TS poredak).

- Transakcija T_i pre pristupa zapisu Q mora da zatraži odgovarajuće **pravo zaključavanja**.
- Transakcija pristupa zapisu **isključivo nakon zaključavanja**.
- Zapis se može zaključati na dva načina: deljivo i nedeljivo.
 - **Deljivo** (engl. *shared lock*).
 - Transakcija koja dobije pravo na deljivo zaključavanje zapisa Q :
 - Dobija pravo čitanja zapisa Q
 - Ne dobija pravo upisa zapisa Q .
 - **Ekskluzivno** (engl. *exclusive lock*).
 - Transakcija koja dobije pravo na ekskluzivno zaključavanje zapisa Q taj zapis može čitati i upisati.

Poredak izvršavanja na bazi vremenske oznake (TS poredak)

- Sinhronizacija između procesa može se ostvariti pomoću **protokola zasnovanog na vremenskim oznakama**.
- Svakoj transakciji dodeljujemo **TS** kao **vreme kada transakcija počinje da se izvršava**.
- Takođe svakom zapisu Q dodeljujemo dva vremenska parametra:
 - **W-timestamp** (Q) – vreme poslednje transakcije koja je uspešno obavila upis u Q .
 - **R-timestamp** (Q) – vreme poslednje transakcije koja je uspešno obavila čitanje iz Q .
- Ova dva parametara se stalno ažuriraju.
- TS protokol obezbeđuje da konfliktno čitanje i upis obave u TS poretku koji isključuje preklapanje konflikta.

Poredak izvršavanja na bazi vremenske oznake (TS poredak)

- Pretpostavimo da transakcija T_i pošalje zahtev *read(Q)*.
- Moguće su dve situacije:
 - $TS(T_i) \geq W\text{-timestamp}(Q)$
 - Zahtev je korektan.
 - Čitanje se obavlja.
 - Nakon toga se ažurira $R\text{-timestamp}(Q)$.
 - $TS(T_i) < W\text{-timestamp}(Q)$
 - Transakcija traži vrednost Q iz prošlosti.
 - Ta vrednost je prepisana.
 - Čitanje se odbacuje.
 - Vrednost se može dobiti primenom metode *rollback* na transakciju T_i .

Poredak izvršavanja na bazi vremenske oznake (TS poredak)

- Pretpostavimo da je transakcija T_i pošalje zahtev ***write(Q)***.
- Moguće su tri situacije:
 - $TS(T_i) < R\text{-timestamp}(Q)$
 - Transakcija pokušava da upiše nešto što je već trebalo da bude pročitano
 - Zahtev je nekorektan pa se upis odbacuje
 - Na transakciju T_i se primenjuje metoda *rollback*.
 - $TS(T_i) < W\text{-timestamp}(Q)$
 - Transakcija pokušava da upiše staru vrednost za Q
 - Upis se odbacuje
 - Na transakciju T_i se primenjuje metoda *rollback*.
 - U svim ostalim slučajevima upis se obavlja.
- Napomena: transakcija za koju se obavlja *rollback* dobija novu TS vrednost i restartuje se.

Konkurentne atomske transakcije u distribuiranim uslovima

- Mehanizmi koje smo definisali za sinhronizaciju procesa moraju se prilagoditi ili redefinisati za distribuiranu okolinu!
- Svaki sajt mora da ima sopstveni mehanizam za upravljanje transakcijama koji pored sinhronizacije obavezno vodi i dnevnik koji pomaže u slučaju opravaka od otkaza.
- Kao i u slučaju centralizovane okoline definisaćemo protokole za zaključavanje i *timestamp* metodu.

- **Slučaj bez replikacije datoteka.**
 - Svaki sajt ima svoj mehanizam za zaključavanje i otključavanje podataka (engl. **lock manager**).
 - Transakcija koja želi da zaključa zapis Q na sajtu S_i šalje **poruku o zaključavanju**: $lock(S_i, Q)$.
 - *Lock manager* sajta S_i :
 - Poslaće transakciji poruku da joj je zahtev ispunjen ako ima uslove da zaključa Q .
 - Kada transakcija odradi svoje slanjem poruke $unlock(S_i, Q)$ otključaće zapis Q .
 - U protivnom transakcija mora da čeka.
- **Slučaj sa više replika datoteke u sistemu.**
 - Protokoli za zaključavanje se mogu realizovati na više načina:
 - protokol za zaključavanje sa jednim koordinatorom,
 - protokol većine,
 - pristrasni protokol,
 - primarna kopija.

Timestamp protokoli u distribuiranim uslovima

- Svaka transakcija ima **vremensku oznaku** (TS) koja omogućava serijalizaciju izvršavanja.
- Jedinствена TS vrednost može se generisati **centralizovanom** ili **distribuiranom metodom**.
 - Dodela TS vrednosti je centralizovana ukoliko **jedan sajt generiše i dodeljuje TS vrednosti** na osnovu vrednosti svog logičkog brojača ili sopstvenog časovnika.
 - Dodela TS vrednosti je distribuirana ukoliko se jedinствена TS vrednost formira udruživanjem **jedinствene lokalne TS vrednosti** koju svaki sajt generiše putem svog logičkog brojača i **identifikatora računara**.
 - Poredak udruživanja je bitan.
 - Problem različitih brzina računara rešava se sinhronizacijom dva logička brojača pri slanju poruka između sajtova.
 - Primer:
 - Kada jedan sajt poseti drugi sajt sa manjim *LC*, tada se *LC* bržeg sajta sinhroniše, odnosno postavlja na *LC* vrednost sporijeg uvećanu za 1.

Šema poretka vremenskih oznaka

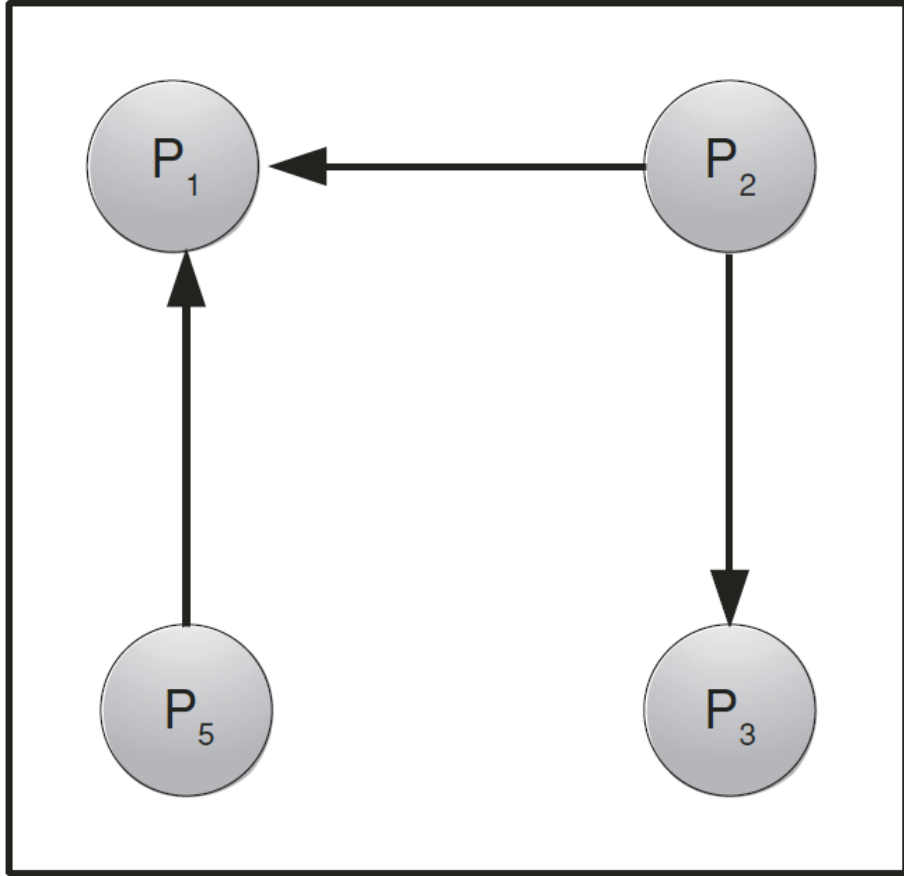
- Ova šema kombinuje centralizovanu *timestamp* metodu sa 2PC protokolom u cilju obezbeđivanja poretka bez velikog broja kaskadnih *roll-back* operacija.
- Različiti zahtevi za čitanje i upis se baferuju, a zatim izvršavaju u poretku definisanom na sledeći način:
 - Zahtev transakcije T_i za **čitanje podatka x** mora biti odložen ako postoji transakcija T_j koja treba da izvrši upis podatka x a pri tome je $TS(T_j) < TS(T_i)$.
 - Zahtev transakcije T_i za **upis podatka x** mora biti odložen ukoliko postoji transakcija T_j koja treba da izvrši čitanje ili upis podatka x a pri tome je $TS(T_j) < TS(T_i)$.

Upravljanje zastojava u distribuiranim uslovima

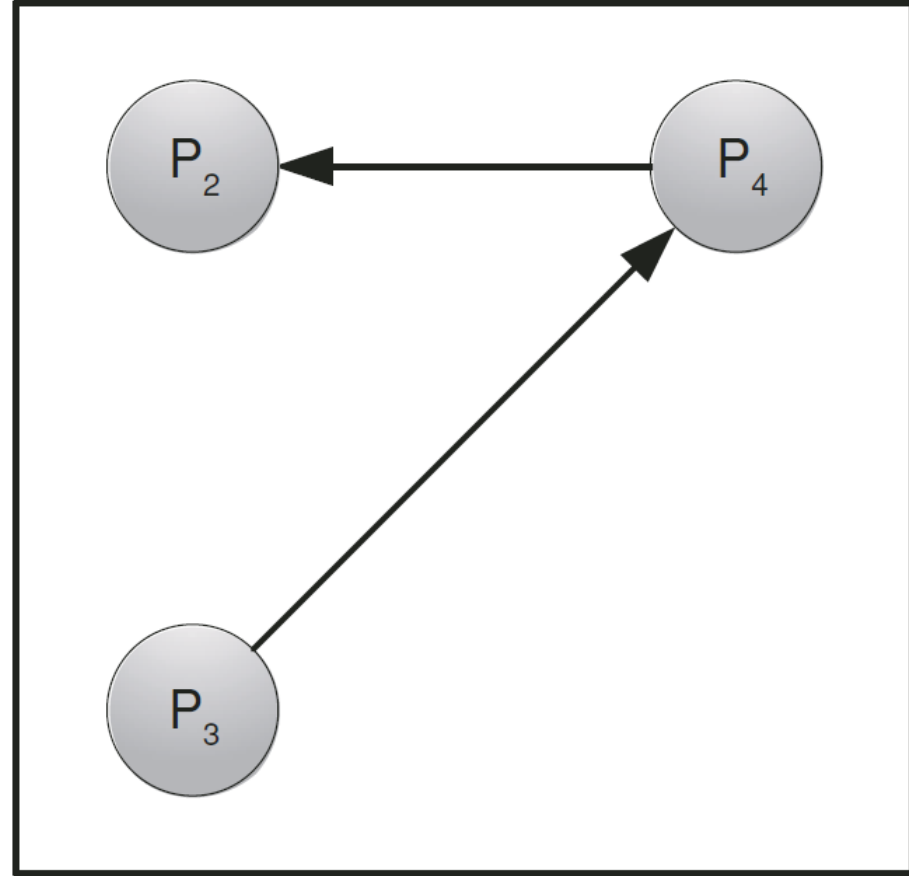
- **Prevenција zastoja.**
 - **Metoda *wait-die*.**
 - Nema pretpražnjenja (nasilnog oduzimanja resursa).
 - Stariji proces čeka na mlađeg da otpusti resurs.
 - Proces P_i koji traži resurs dodeljen procesu P_j čekaće da proces P_j taj resurs otpusti samo ako je $TS(P_i) > TS(P_j)$.
 - U protivnom proces P_i se podvrgava *rollback* operaciji.
 - **Metoda *wound-wait*.**
 - Suprotna metodi *wait-die*.
 - Zasnovana je na tehnici pretpražnjenja.
 - Proces P_i će čekati na resurs dodeljen procesu P_j samo ako je P_i mlađi: $TS(P_i) < TS(P_j)$.
 - U protivnom, P_j se podvrgava *rollback* operaciji, a resurs mu se oduzima i predaje procesu P_i .
 - Stariji proces nikada ne čeka na mlađe.

Upravljanje zastojava u distribuiranim uslovima

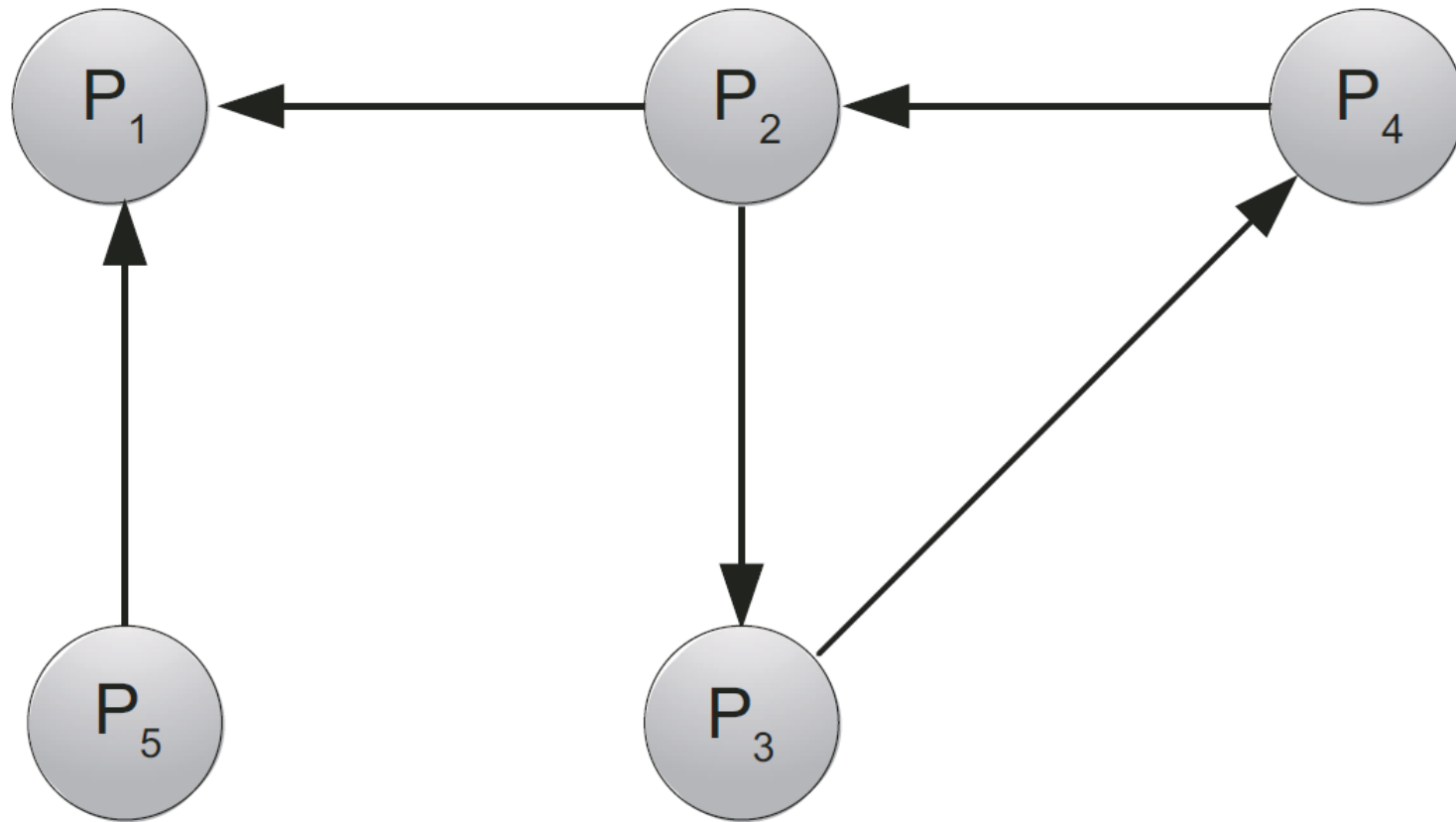
- **Detekcija zastoja.**
 - ***Wait-for grafovi* sajtova** spajaju se u jedan **zajednički graf**.
 - Detekcija zastoja u DS je složena procedura i može se obavljati:
 - **Centralizovano.**
 - Problem zastoja rešava jedan sajt na kome se izvršava proces **koordinator za detekciju zastoja** (*deadlock detection coordinator*).
 - Taj proces treba da proceni da li je DS u stanju zastoja ili ne.
 - Ovaj proces traži od ostalih sajtova da mu dostave svoj *wait-for* graf kako bi na osnovu toga konstruisao zajednički *wait-for* graf.
 - **Distribuirano.**
 - Svi sajtovi su zaduženi za detekciju zastoja.
 - Svaki sajt konstruiše svoj modifikovani *wait-for* graf uključujući Pex čvor (eksterni čvor).
 - Kada primeti mogućnost zastoja, sajt šalje poruku kritičnom sajtu sa kojim se može naći u stanju zastoja da oba sajta dodatno provere da li je sistem u zastoju.



Računar R₁



Računar R₂



1. B. Đorđević, D. Pleskonjić, N. Maček (2005): Operativni sistemi: teorija, praksa i rešeni zadaci. Mikro knjiga, Beograd.
2. R. Popović, I. Branović, M. Šarac (2011): Operativni sistemi. Univerzitet Singidunum, Beograd.

Pitanja su dobrodošla.