Konzistentnost i replikacija procesa

Konzistentnost i replikacija

- **Replikacija** podataka se vrši radi **povećanja pouzdanosti** ili **unapređenja performansi**
- Kada postoje replike, jedan od glavnih problema je održati ih **konzistentnim**
  - kada se jedna kopija ažurira, moramo osigurati da se i sve ostale kopije ažuriraju. U suprotnom, replike više neće biti identične
- **Motivacija za replikaciju**:

1. **Povećanje pouzdanosti**: replicirani fajl sistem može da nastavi da radi i nakon pada neke replike, pruža se i bolja zaštita od narušavanja podataka
2. **Unapređenje performansi**: skaliranje u smislu veličine ili geografske pokrivenosti, kada veliki broj procesa želi da pristupi podacima kojima upravlja jedan server replikacijom servera i preraspodelom opterećenja mogu se unaprediti performanse, **cena replikacije – problemi sa održavanjem konzistentnosti**

Replikacija kao tehnika za skaliranje

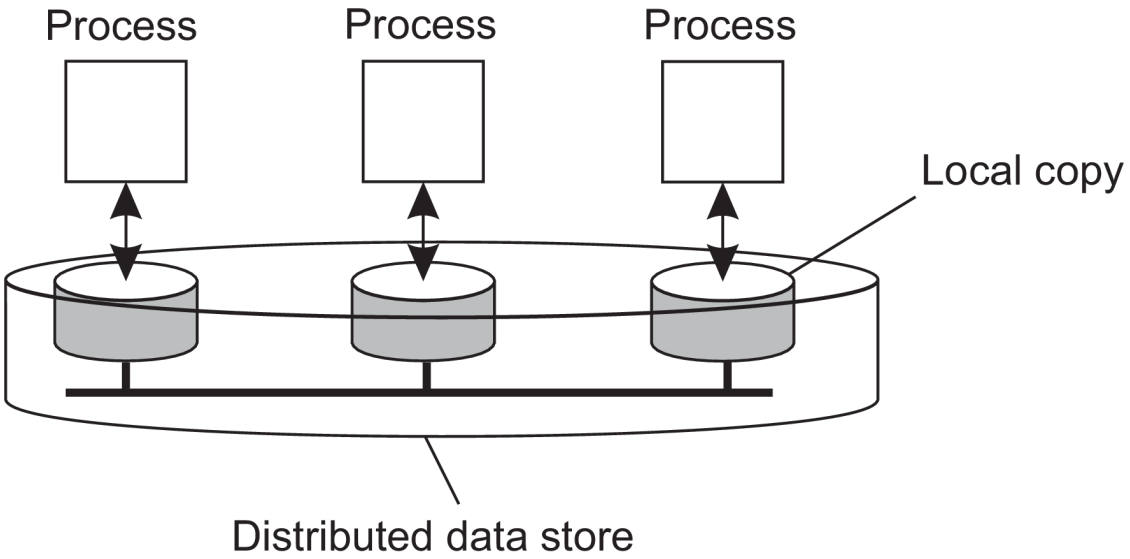
- Replikacija i keširanje su široko upotrebljavane tehnike za skaliranje čime se **unapređuju performanse** – **podaci se čuvaju** što je moguće **bliže procesima**, što može dovesti do **povećanih zahteva za propusnim opsegom**
- Još ozbiljniji problem je **očuvanje konzistentnosti replika** prilikom skaliranja – ažuriranje jedne kopije propagira se na sve kopije
- **Sinhrona replikacija** – ažuriranje se na svim kopijama izvodi kao jedinstvena **atomična operacija** ili transakcija, uvođenje atomičnosti unosi komplikacije
- Kako bi se održala konzistentnost, neophodno je da se **sve potencijalno konfliktne operacije izvedu svuda u istom redosledu**
  - **čitaj-piši** (engl. *read-write*) **konflikt**: operacije čitanja i pisanja se izvršavaju konkurentno
  - **piši-piši** (engl. *write-write*) **konflikt**: dve konkurentne operacije pisanja

Replikacija kao tehnika za skaliranje

- Replike mogu biti u situaciji da treba da odluče o globalnom redosledu operacija korišćenjem Lamportovih vremenskih otisaka – ovakva **globalna sinhronizacija** zahteva dosta vremena za komunikaciju
- Replikacijom i keširanjem se rešava skaliranje, ali dolazi do problema sa performansama ako konzistentnost zahteva globalnu sinhronizaciju – rešenje **relaksacija konzistentnosti** kako bi se izbegla globalna sinhronizacija
- **Vrsta relaksacije** zavisi od **šablona pristupa i ažuriranja repliciranih podataka**
- **Modeli konzistentnosti usmereni na podatke**: **kontinualna konzistentnost**, **sekvencijalna konzistentnost**, kauzalna konzistentnost, eventualna konzistentnost
- **Modeli konzistentnosti usmereni na klijente**: monotona čitanja, monotoni upisi, čitaj svoje upise, upisi prate čitanja

Modeli konzistentnosti usmereni na podatke

- **Model konzistentnosti** je **ugovor** između (distribuiranog) **skladišta podataka** i **procesa**, u kome **skladište podataka precizno specificira** šta su **rezultati operacija čitanja i upisa** u **konkurentnom okruženju**
- Skladište podataka (engl. *data store*) je distribuirana kolekcija memorijskih medijuma:



Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Kontinualna konzistentnost

- Kod **kontinualne konzistentnosti** (engl. *continuous consistency*) [Yu and Vahdat 2002] cilj je **postaviti granice** za **numeričku devijaciju** replika, **devijaciju ustajalosti** (engl. *staleness*) i **devijaciju u uređenju** operacija
- **Numerička devijacija** odnosi se na vrednost za koju replike mogu da se razlikuju, razlikujemo apsolutnu i relativnu. Ovaj vid devijacije značajno zavisi od primene, ali se može koristiti npr. kod replikacije akcija na berzi
- **Devijacija ustajalosti** odnosi se na vreme u kome se replika još uvek smatra konzistentnom, uprkos tome što su se ažuriranja desila pre nekog vremena (npr. keširanje na vebu, vremenska prognoza)
- **Devijacija u uređenju operacija** odnosi se na maksimalni broj privremenih (engl. *tentative*) upisa koji mogu da postoje na bilo kom serveru, bez da je izvršena sinhronizacija sa ostalim replikama servera
- Jedinica za konzistentnost – **conit** određuje jedinični podatak nad kojim se meri konzistentnost

Sekvencijalna konzistentnost

- Model **proširuje kontinualnu konzistentnost** u smislu da, kada privremena ažuriranja na replikama treba da se komituju, replike moraju da se dogovore o globalnom tj. konzistentnom uređenju datih ažuriranja
- Model predložio Lamport 1979, sekvencijalna konzistentnost suštinski **pruža semantiku** koju **programeri očekuju u konkurentnom programiranju**: **svi vide sve operacije upisa u istom redosledu**
- **Rezultat je uvek kao da su sve operacije svih procesa izvršene u određenom sekvencijalnom redosledu, i operacije u svakom pojedinačnom procesu se pojavljuju u ovoj sekvenci u redosledu specificiranom odgovarajućim programom**

(a) sekvencijalno konzistentno skladište

P1:	W(x)a			
P2:		W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)b	R(x)a

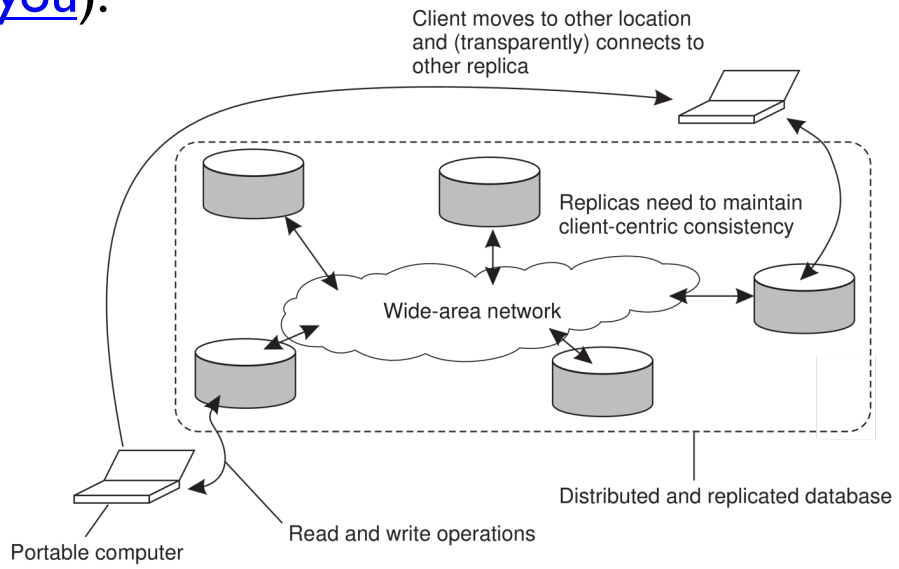
(b) skladište koje nije sekvencijalno konzistentno

P1:	W(x)a			
P2:		W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Modeli konzistentnosti usmereni na klijenta

- Posmatramo distribuiranu bazu kojoj pristupamo putem laptopa kao front end-a, na lokaciji A uradimo ažuriranje i pređemo na lokaciju B, ako se promeni server kome pristupamo mogu se javiti nekonzistentnosti
- Za klijenta je važno da se ulazi koje je ažurirao u A, u B pojavljuju u istom stanju kao što ih je ostavio u A – tada se baza klijentu čini konzistentnom
- Princip mobilnog korisnika koji pristupa različitim replikama distribuirane baze (primer Bayou):



Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Modeli konzistentnosti usmereni na klijenta

- U **modele konzistentnosti usmerene na klijente** spadaju:
  - **monotona čitanja** (engl. *monotonic reads*): ako proces pročita vrednost podatka *x*, svaka sukcesivna operacija čitanja *x* od strane tog procesa će uvek vratiti istu ili noviju vrednost. **Primeri:** automatsko čitanje ažuriranja ličnog kalendara sa različitih servera. Monotona čitanja garantuju da korisnik vidi sva ažuriranja, bez obzira sa kog servera se dešava automatsko čitanje; čitanje (bez modifikacije) pristižućeg mejla u pokretu. Svaki put kada se povežete na različiti e-mejl server, taj server (u najmanju ruku) pribavi sva ažuriranja sa servera koji ste prethodno koristili
  - **monotoni upisi** (engl. *monotonic writes*): operacija upisa podatka *x* od strane nekog procesa se kompletira pre bilo koje sukcesivne operacije upisa *x* od strane istog procesa. **Primeri:** ažuriranje programa na serveru *S2* i osiguravanje da se sve komponente od kojih prevođenje i linkovanje zavisi takođe nalaze na *S2*; održavanje verzija repliciranih fajlova svugde u ispravnom redosledu (propagacija prethodne verzije serveru gde je instalirana najnovija verzija)

Modeli konzistentnosti usmereni na klijenta

- U **modele konzistentnosti usmerene na klijente** spadaju:
  - **čitaj svoje upise** (engl. *read your writes*): efekti operacije upisa podataka *x* od strane nekog procesa će uvek biti vidljivi od strane sukcesivne operacije čitanja *x* od strane istog procesa. **Primer**: ažuriranje veb stranice i garantovanje da će veb pretraživač prikazati najnoviju verziju, a ne keširanu kopiju
  - **upisi prate čitanja** (engl. *writes follow reads*): operacija upisa podatka *x* od strane nekog procesa, koja prati prethodnu operaciju čitanja *x* od strane istog procesa, se garantovano dešava nad istom ili novijom vrednošću *x* od one koja je pročitana. **Primer:** reakcije na postavljene članke se vide samo ako pratite taj post (operacija čitanja "povlači" odgovarajuće operacije upisa)

Upravljanje replikama

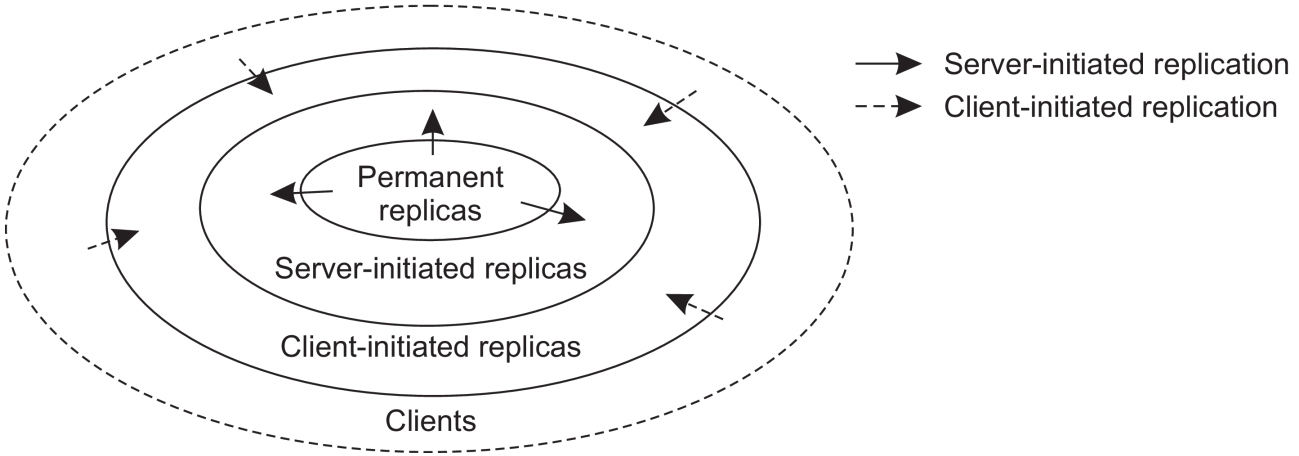
- Ključna odluka u svim distribuiranim sistemima koji podržavaju replikaciju je **gde, kada i ko postavlja replike** i, potom, koji se **mehanizmi** koriste kako bi se **replike očuvale konzistentnim**
- Problem postavljanja replika se sastoji od dva potproblema – postavljanje **replika servera** (engl. *replica servers*) i postavljanja **sadržaja** (engl. *content*)
  - **postavljanje replika servera** se bavi pronalaženjem najboljih lokacija za postavljanje servera na kome se čuva (deo) skladišta podataka, postoji veći broj algoritama za ovu namenu
  - **postavljanje sadržaja** tiče se pronalaženja najboljih servera za držanje sadržaja

Postavljanje replika servera

- Treba pronaći *K* najboljih mesta od *N* mogućih lokacija:
  - izaberi najbolju lokaciju od $N - K$ za koju je prosečna udaljenost do klijenata minimalna. Onda izaberi sledeći najbolji server (prva odabrana lokacija minimizira srednju distancu do svih klijenata). Algoritam je skup za izračunavanje – $O(N^2)$
  - izaberi *K*-ti najveći autonomni sistem i postavi server na najbolje povezanom domaćinu. Algoritam je skup za izračunavanje – $O(N^2)$
  - [Szymaniak et al. 2006] postavi čvorove u *d*-dimenzionalni geometrijski prostor, gde distanca odgovara latenciji. Identifikuj *K* regija sa najvećom gustinom i postavi server u svaku od njih. Algoritam je jeftin za izračunavanje – $O(N \times \max\{\log(N), K\})$, omogućava postavljanje replika servera u realnom vremenu

Replikacija sadržaja

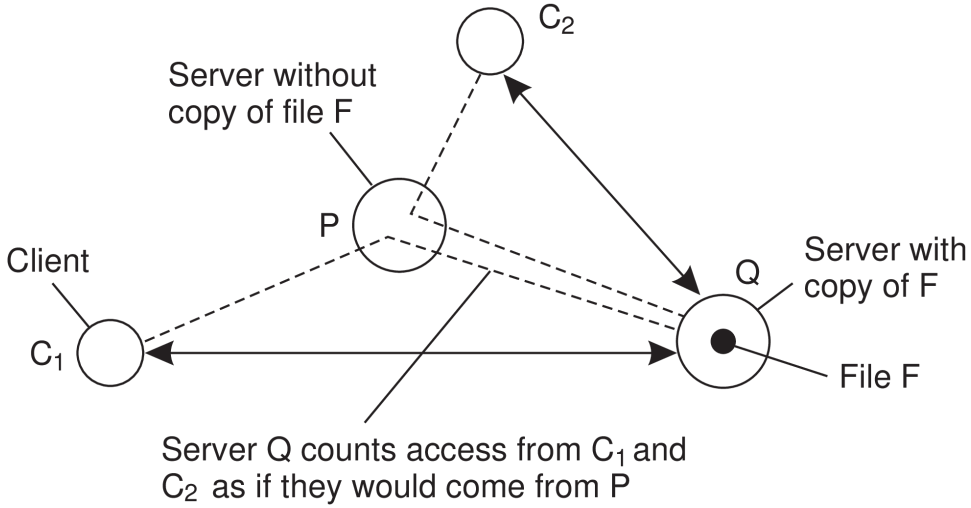
- Proces je u stanju da bude domaćin tj. da hostuje repliku objekta ili podatka:
  - **permanentna replika**: proces/mašina uvek poseduje repliku, npr. *mirror site*
  - **replika inicirana od servera** (engl. *server-initiated replica*): proces koji može dinamički hostovati repliku na zahtev drugog servera u skladištu podataka
  - **replika inicirana od klijenta** (engl. *client-initiated replica*): proces koji može dinamički hostovati repliku na zahtev klijenta (**klijentski keš**)



Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Replike inicirane od servera

- Dinamička replikacija kod web hostinga – broje se zahtevi za pristup od različitih klijenata [Rabinovich et al. 1999]:



- Vodi se evidencija o broju pristupa po fajlu, agregirana uzimanjem u obzir servera najbližeg klijentima koji šalju zahteve:
  - Ako broj pristupa padne ispod praga D ⇒ odbaci (engl. drop) fajl
  - Ako broj pristupa pređe prag R ⇒ repliciraj fajl
  - Ako je broj pristupa između D i R ⇒ migriraj fajl

Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Otpornost na otkaze i konsenzus algoritmi

Otpornost na otkaze

- Karakteristika distribuiranih sistema koja ih značajno razlikuje od sistema sa jednom mašinom je mogućnost **parcijalnog kvara** (engl. *partial failure*) – deo sistema može otkazati dok ostatak nastavlja da ispravno radi
- Važno je konstruisati distribuirane sisteme tako da se mogu **automatski oporaviti** od parcijalnih kvarova bez ozbiljnog uticaja na performanse
- Od distribuiranog sistema se očekuje da bude **otporan na otkaze** (engl. *fault tolerant*), otpornost se unapređuje **uvođenjem redundantnosti**
- Osnovna tehnika: organizacija procesa u **procesne grupe** radi postizanja otpornosti na otkaze, **više identičnih procesa sarađuje pružajući privid jedinstvenog logičkog procesa** tako da, ako **jedan ili više njih otkažu klijent to ne primećuje**; posebno važno pitanje u procesnim grupama je postizanje **konsenzusa** o tome koju od operacija zatraženih od klijenta treba izvršiti – rešavaju ga **konsenzus algoritmi** (npr. Paxos, Raft, PBFT, HotStuff)

Zavisnost (engl. *dependability*)

- **Komponenta** pruža **servise klijentima**. Kako bi pružila usluge, komponenta može zahtevati usluge **od drugih komponenti**, tj. **zavisiti** (engl. *depend*) od njih
- Komponenta C_i **zavisi** od neke druge komponente C_j ako **tačnost rada C_i zavisi od tačnosti rada C_j** (komponente mogu biti procesi ili kanali)
- Zahtevi povezani sa zavisnošću:

Zahtev	Opis
dostupnost (engl. *availability*)	sposobnost, tj. pripravnost za upotrebu – verovatnoća u procentima
pouzdanost (engl. *reliability*)	kontinualnost pružanja usluga bez kvara
sigurnost (engl. *safety*)	kada sistem privremeno ne radi ispravno, ne dolazi do katastrofalnih događaja
održivost (engl. *maintainability*)	lakoća oporavka sistema nakon kvara, lako održiv sistem ima tipično i visoku dostupnost

Pouzdanost i dostupnost

- **Pouzdanost** $R(t)$ komponente C predstavlja uslovnu verovatnoću da je C radila ispravno tokom vremena $[0, t)$, pod pretpostavkom da je C radila ispravno u trenutku $T = 0$. Tradicionalne **metrike**:
  - **srednje vreme do kvara** komponente (engl. *Mean Time To Failure* – **MTTF**) – vreme ispravnog rada komponente do njenog kvara
  - **srednje vreme do opravke** komponente (engl. *Mean Time To Repair* – **MTTR**) – vreme potrebno za opravku komponente nakon kvara
  - **srednje vreme između otkaza** (engl. *Mean Time Between Failures* – **MTBF**) jednako je zbiru MTTF i MTTR
- **Dostupnost** $A(t)$ komponente C predstavlja srednji udeo vremena u kome je C radila ispravno tokom intervala $[0, t]$
  - dugoročna dostupnost $A(\infty)$, formula $A = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF+MTTR}$
- **Pouzdanost** i **dostupnost** imaju smisla samo ako imamo **tačan opis** šta se u stvari podrazumeva pod **kvarom** (engl. *failure*)

Kvar, greška, otkaz

Pojam	Opis	Primer
kvar (engl. *failure*)	komponenta ne radi u skladu sa svojom specifikacijom	program koji je prestao da radi
greška (engl. *error*)	deo komponente koji može dovesti do kvara	bag u programskom kodu
otkaz (engl. *fault*) • **prolazni** (engl. *transient*) • **povremeni** (engl. *intermittent*) • **trajni** (engl. *permanent*)	uzrok greške, krivica za kvar • desi se jednom i nestane, operacije se ponavlja bez otkaza • otkaz se pojavljuje i nestaje spontano i ponavlja se • otkaz postoji dok se komponenta koje je otkazala ne zameni	loš programer • ptica i predajnik • labav kontakt • softverski bag, pregoreli čip

Upravljanje otkazima

- Sistem u kvaru ne pruža adekvatno servise za koje je projektovan. Postoji više načina za upravljanje otkazima u distribuiranim sistemima:

Pojam	Opis	Primer
prevencija otkaza (engl. *fault prevention*)	prevencija pojavljivanja otkaza	ne zaposliti loše programere
tolerisanje otkaza (engl. *fault tolerance*)	napraviti komponentu tako da može maskirati pojavu otkaza	napraviti svaku komponentu od strane dva nezavisna programera
uklanjanje otkaza (engl. *fault removal*)	redukovati prisustvo, broj i ozbiljnost otkaza	dati otkaz lošim programerima ☺
predviđanje otkaza (engl. *fault forecasting*)	proceniti trenutno prisustvo, buduću incidencu i posledice otkaza	proceniti u kom procentu firma angažuje loše programere

Modeli kvarova

Pojam	Opis ponašanja servera (procesa)
kvar usled pada (engl. *crash failure*)	proces pada, ali radi ispravno do trenutka pada, npr. padovi OS
kvar usled propusta (engl. *omission failure*) • propust prijema (*receive-omission*) • propust slanja (*send-omission*)	proces ne odgovara na dolazeće zahteve • ne prihvata dolazeće poruke • ne šalje poruke
kvar usled lošeg tajminga (engl. *timing failure*)	odgovor procesa se dešava izvan određenog vremenskog intervala, npr. video striming
kvar usled pogrešnog odgovora (engl. *response failure*) • otkaz vrednosti • otkaz prelaza stanja (*state-transition*)	odgovor procesa je netačan • vrednost odgovora je netačna • postoji devijacija od ispravne kontrole toka
proizvoljni (vizantijski) kvar (engl. *arbitrary failure*)	proces može proizvesti proizvoljne odgovore u proizvoljnim trenucima

Pouzdanost i sigurnost

- **Proizvoljni (vizantijski) kvarovi** se često **poistovećuju sa malicioznim** procesima, mada uglavnom nije moguće pouzdano utvrditi da li je neka akcija bila benigna ili maliciozna. S toga se pravi sledeća razlika:
  - **kvar usled propuštanja** (engl. *omission failure*) – komponenta ne preduzme akciju koju je trebala preduzeti
  - **kvar usled delovanja** (engl. *comission failure*) – komponenta preduzme akciju koju nije trebala preduzeti
- **Namerni** (engl. *deliberate*) **kvarovi**, bili oni usled propuštanja ili delovanja, su tipično bezbednosni problem
- Pravljenje razlike između namernih i slučajnih kvarova u opštem slučaju nije moguće

Zaustavljajući kvarovi

- Proces *P* ne registruje više nikakvu aktivnost nekog drugog procesa *Q* – kako *P* može zaključiti da li se *Q* zaustavio (engl. *halt*) ili je došlo do kvara usled propusta ili tajminga?
- Kako bi odgovorili na prethodno pitanje, pravimo razliku između **dva tipa distribuiranih sistema**:
  - **asinhroni sistemi**, kod kojih **nema nikakvih pretpostavki** o brzini izvršavanja procesa ili dužini trajanja slanja poruka, i kod kojih se **ne mogu pouzdano detektovati kvarovi usled pada**, i
  - **sinhroni sistemi**, kod kojih su brzina izvršavanja procesa i dužina trajanja isporuke poruka **vremenski ograničeni**, i kod kojih je **moguće pouzdano detektovati kvarove usled propusta ili tajminga**
  - **u praksi imamo parcijalno sinhrone sisteme**, koji se **najveći deo vremena** ponašaju kao **sinhroni**, ali kod kojih **nema vremenskih ograničenja** za periode u kome se ponašaju kao **asinhroni** – s toga možemo koristiti tajmaut kako bi zaključili da li se proces srušio, ali ponekad taj zaključak može biti pogrešan – **uglavnom** možemo **pouzdano detektovati kvarove usled pada**

Kvarovi usled pada

- **Kvarovi usled pada** mogu se **klasifikovati** na sledeći način, počevši od najmanje ozbiljnih (proces *P* pokušava da detektuje da li je proces *Q* pao):

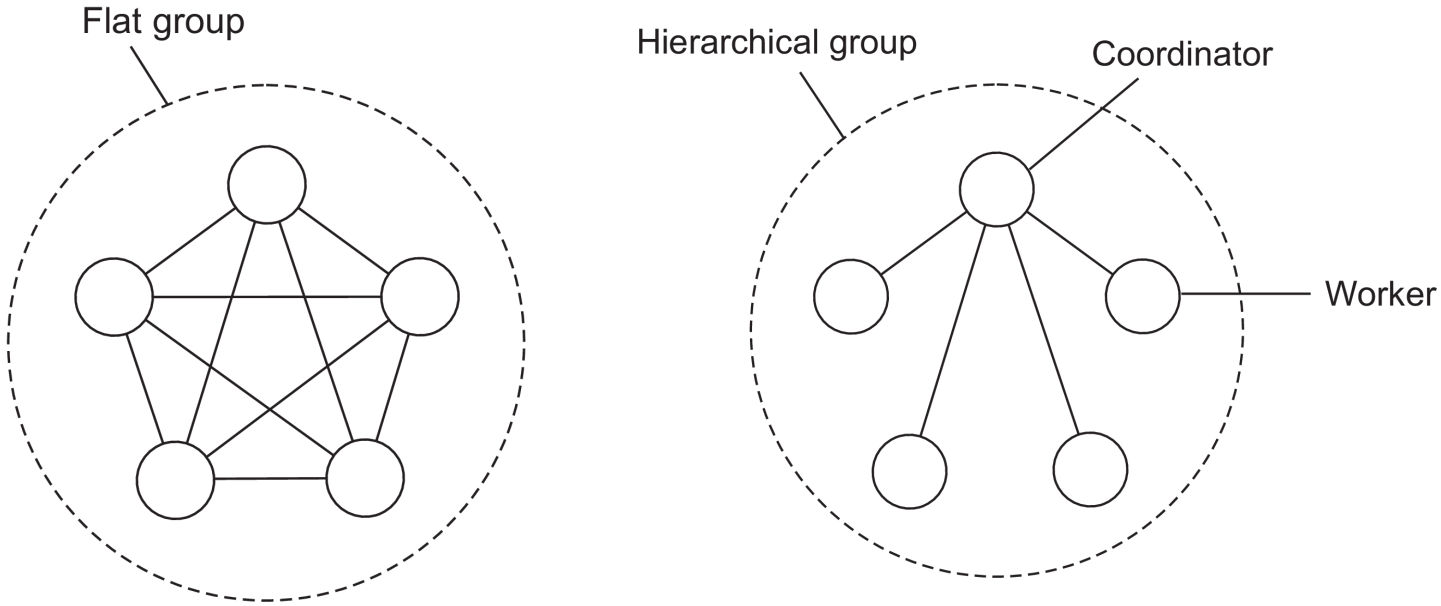
Tip pada	Opis
zaustavljajući kvar (engl. *fail-stop*)	kvar usled pada koji se može pouzdano detektovati, podrazumeva ispravne komunikacione linkove, *P* može postaviti granicu na vreme u kome očekuje odgovor od *Q*
bučni kvar (engl. *fail-noisy*)	kvar usled pada, *P* može samo na kraju (eventualno) doći do ispravnog zaključka da je *Q* pao (postoji neko unapred nepoznato vreme u kome je detekcija pada nepouzdana)
tihi kvar (engl. *fail-silent*)	kvarovi usled pada ili propusta, klijenti ne mogu da znaju gde je kvar
sigurni kvar (engl. *fail-safe*)	proizvoljni, ali benigni kvarovi (ne mogu naškoditi)
proizvoljni kvar (engl. *fail-arbitrary*)	proizvoljni maliciozni kvarovi koji mogu biti neprimetni za ostale procese

Redundantnost za maskiranje kvarova

- **Ključna tehnika** za **maskiranje otkaza** je **redundantnost** (engl. *redundancy*)
- Tipovi redundantnosti:
  - **informaciona redundantnost**, kod koje se dodaju bitovi podacima kako bi greške mogle da se isprave; npr. primena Hamingovog koda za kodiranje podataka koji se šalju kroz komunikacioni kanal sa šumom
  - **vremenska redundantnost**, kod koje se akcije izvode onoliko puta koliko je neophodno, tj. sistem se projektuje tako da se akcija ponavlja ako se desi greška, tipično se koristi za rešavanje prolaznih ili povremenih kvarova; npr. transakcije, ponovno slanje zahteva serveru ako nema očekivanog odgovora
  - **fizička redundantnost**, kod koje se dodaje oprema ili procesi kako bi sistem bio otporan kao celina na gubitak ili kvar pojedinih komponenti, **replikacijom procesa** može se postići **visok stepen otpornosti** na otkaze, intenzivno korišćena u distribuiranim sistemima

Elastičnost (engl. *resilience*) procesa

- **Zaštita od procesa** koji su **otkazali** realizuje se kroz **replikaciju procesa**, organizacijom više procesa u **procesnu grupu** (engl. *process group*) koja može biti **dinamička**, proces se može pridružiti ili napustiti grupu i može biti član više grupa istovremeno
- Razlikuju se **ravne** (engl. *flat*) i **hijerarhijske** (engl. *hierarchical*) **grupe**:



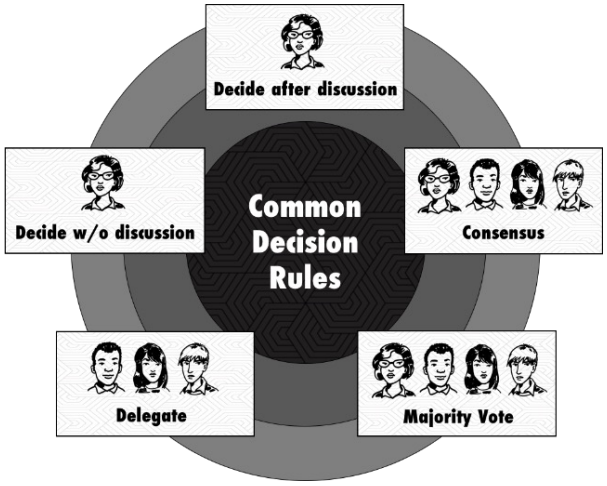
Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Grupe i maskiranje kvarova

- **Grupa** se naziva **tolerantnom na *k* otkaza** (engl. *k-fault tolerant group*) kada može maskirati bilo kojih *k* konkurentnih otkaza njenih članova (*k* se naziva **stepenom tolerantnosti na otkaze**)
- Koliko **velika** treba biti **grupa** da bi bila **tolerantna na *k* otkaza**?
  - kod **zaustavljajućih kvarova** (engl. *halting failures*), bilo da su u pitanju kvarovi usled pada, propusta ili tajminga, pošto **nijedan član grupe neće proizvesti netačan rezultat** za grupu sa ***k* + 1 članova**, **dovoljan je i jedan član**
  - kod **proizvoljnih kvarova** (engl. *arbitrary failures*), neophodno je **2*k* + 1 članova** tako da se tačan rezultat može dobiti kroz većinsko glasanje
- Važne pretpostavke:
  - svi članovi su identični
  - svi članovi obrađuju naredbe u istom redosledu
- Kao rezultat, možemo biti **sigurni da svi procesi rade tačno istu stvar**

Konsenzus

- **Mnogobrojni klijenti** mogu **poslati naredbe procesnoj grupi** koja se ponaša kao **jedinstven, visoko robustan proces**
- Model radi pod **važnom pretpostavkom**:
  - **U procesnoj grupi otpornoj na otkaze, svaki ispravan** (engl. *nonfaulty*) **proces izvršava iste naredbe i u istom redosledu kao i svi ostali ispravni procesi**
  - **Neophodno je da ispravni procesi postignu konsenzus o tome koja je sledeća naredba za izvršavanje**, ako nema kvarova postizanje konsenzusa je lako

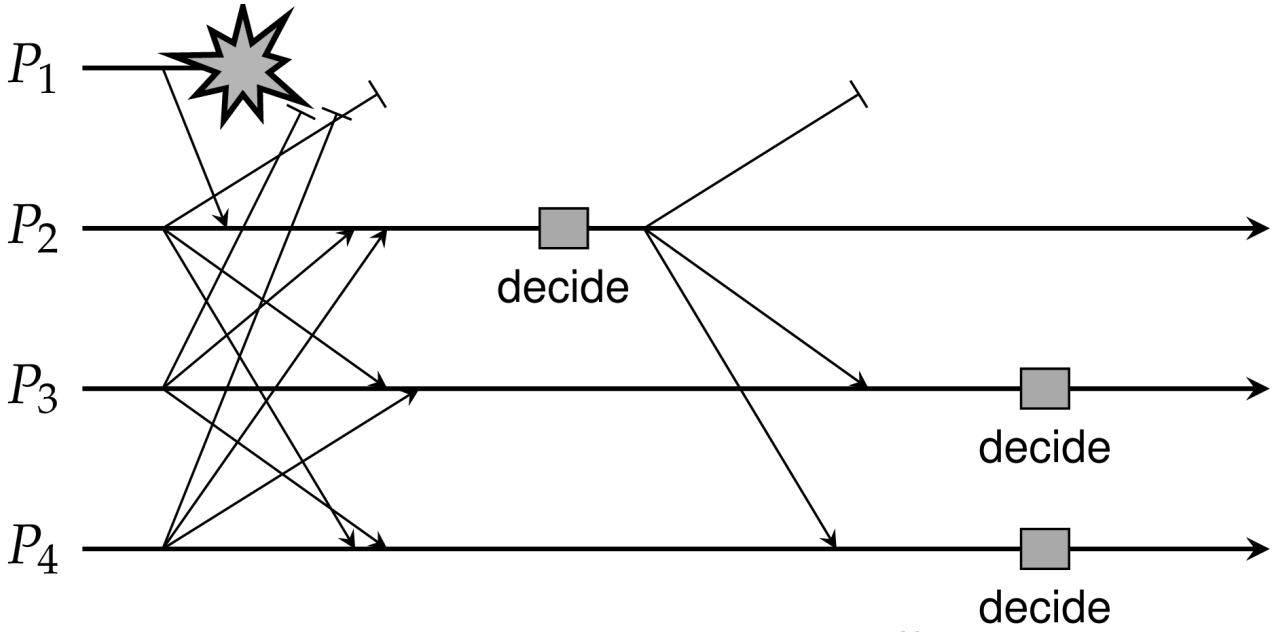


Izvor: https://www.kisspng.com/png-consensus-decision-making-information-brand-manage-3110117/preview.html

Konsenzus zasnovan na plavljenju

- **Konsenzus zasnovan na plavljenju** (engl. *flooding-based consensus*)
  - Model sistema (Cachin et al. 2011):
    - procesna grupa **P** = $\{P_1, P_2, \ldots, P_n\}$
    - semantika zaustavljajućeg kvara (fail-stop), tj. sistem je pouzdan sa detekcijom pada
    - klijent kontaktira P_i i traži od njega da izvrši komandu
    - svaki P_i održava listu predloženih komandi
- **Osnovni algoritam** (zasnovan na rundama):
  - u rundi r, P_i objavljuje multikastom skup njemu poznatih komandi C_i^r svim drugima procesima
  - na kraju runde r, svaki P_i spaja sve primljene naredbe u novi skup C_i^{r+1}
  - sledeća komanda cmd_i bira se putem **globalno deljene deterministčke funkcije**: $cmd_i \leftarrow select(C_i^{r+1})$

Primer: konsenzus zasnovan na plavljenju



- P_1 je pao tokom runde r, P_2 je primio sve predložene komande od svih procesa i donosi odluku
- P_3 je možda detektovao da se P_1 srušio, ali ne zna da li je P_2 primio bilo išta, tj. P_3 ne može znati da li ima iste informacije kao P_2, te s toga ne može doneti odluku (isto važi i za P_4) u rundi r, ali će na osnovu odluke P_2 u narednoj rundi $r + 1$ i P_3 i P_4 moći da donesu odluke

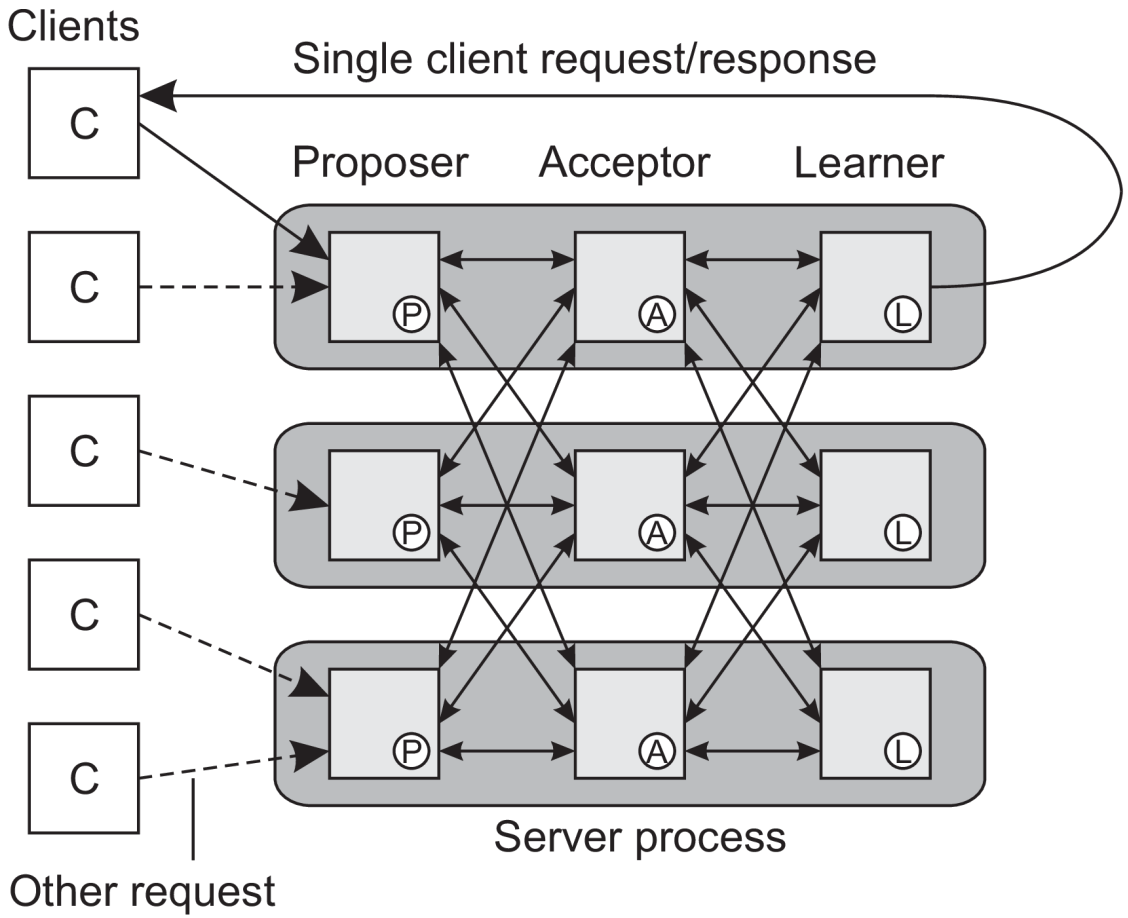
Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Paxos algoritam

- Algoritam zasnovan na plavljenju nije realističan zbog zaustavljajućeg (*fail-stop*) modela kvarova. Realističnije je pretpostaviti **bučni** (*fail-noisy*) **model** u kome procesi **eventualno detektuju** da se neki proces srušio
- **Lamport** je 1989. predložio **Paxos algoritam** u tehničkom izveštaju, tek 1998. objavljen u formi rada, pretpostavlja **klijent-server konfiguraciju**
- (**Slabe i realistične**) **pretpostavke** pod kojima radi Paxos algoritam:
  - distribuirani sistem je **parcijalno sinhron** (može biti i asinhron)
  - **komunikacija** između procesa može biti **nepouzdana**, što znači da poruke mogu biti izgubljene, duplicirane ili preuređene
  - **poruke** koju su **oštećene** mogu biti **detektovane** (i s toga nadalje ignorisane)
  - sve **operacije su determinističke**: jednom kada se krenulo sa izvršavanjem, tačno je poznato šta će se desiti
  - procesi mogu imati **kvarove usled pada**, ali ne i **proizvoljne kvarove**
  - procesi **ne komuniciraju tajno** (engl. *do not collude*)

Paxos algoritam

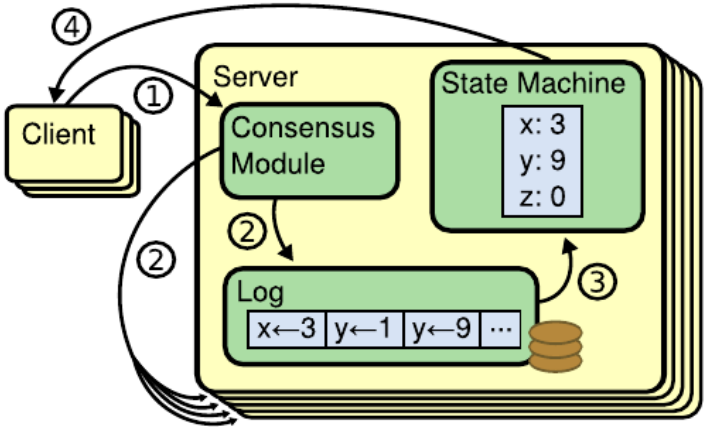
- **Organizacija Paxos algoritma** u različite **logičke procese**:

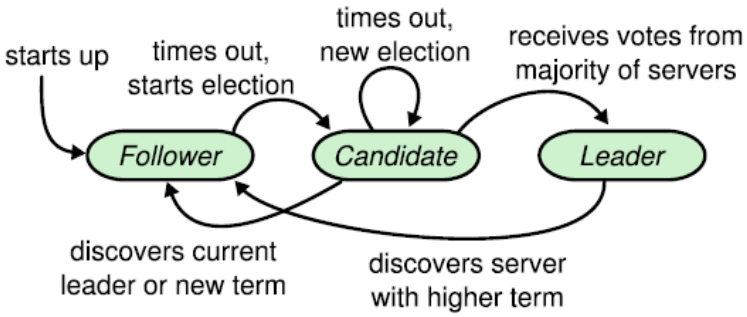


Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Raft algoritam

- **Raft konsenzus algoritam** je predložen 2014. (Ongaro i Ousterhout: https://ramcloud.stanford.edu/wiki/download/attachments/11370504/raft.pdf) kao **razumljivija alternativa Paxos**-u (http://thesecretlivesofdata.com/raft/, https://raft.github.io), **formalno dokazana sigurnost** (klijent uvek dobija tačan odgovor)
  - Raft pruža **generički način za distribuciju konačnih automata** u sistemu, osiguravajući pri tom da se **svaki čvor** u sistemu **slaže o redosledu prelaza između stanja**
  - Raft **razdvaja ključne elemente konsenzusa**, kao što su **izbor lidera**, **replikacija logova** i **sigurnost**, realizuje jači stepen koherentnosti kako bi redukovao broj stanja koja se moraju razmatrati

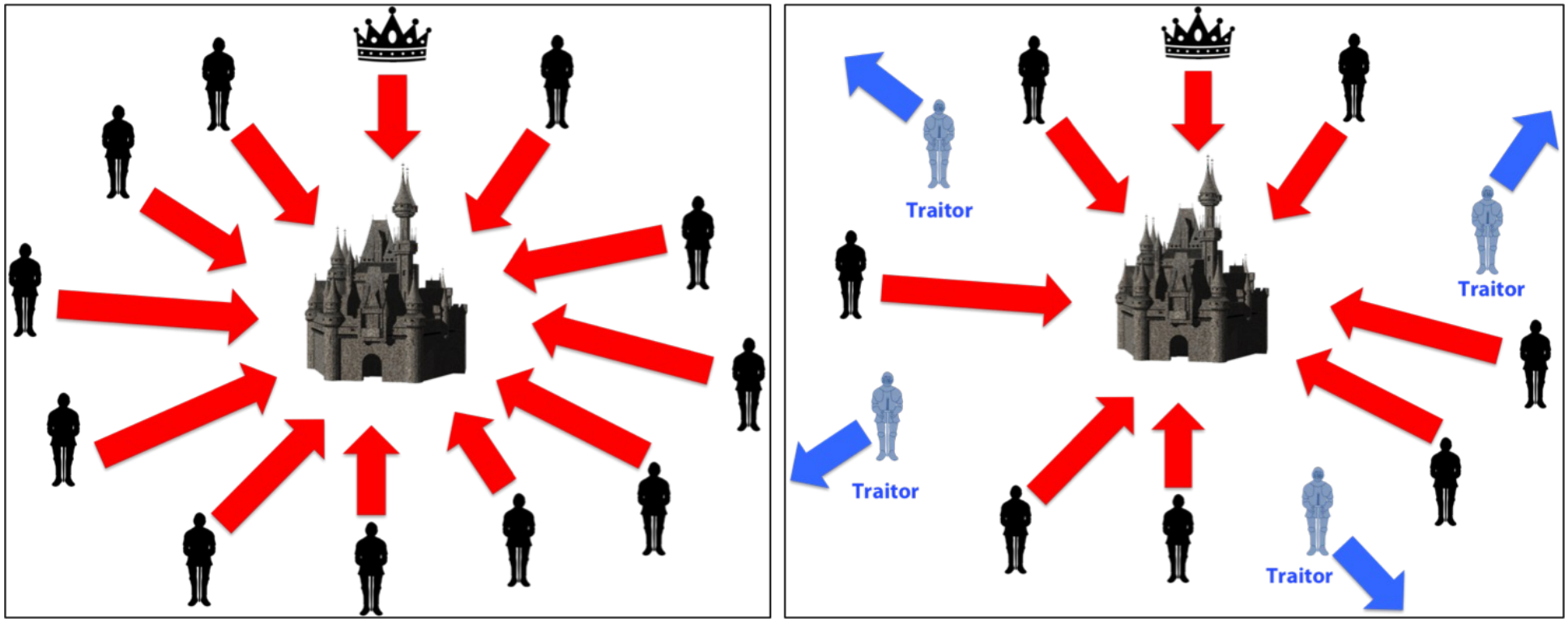




Izvor: https://container-solutions.com/raft-explained-part-23-overview-core-protocol/

Konsenzus sa proizvoljnim otkazima

- **Problem vizantijskih generala** (engl. *The Byzantine Generals Problem*) – Lamport, Shostak, Pease (1982) (http://lamport.azurewebsites.net/pubs/byz.pdf):

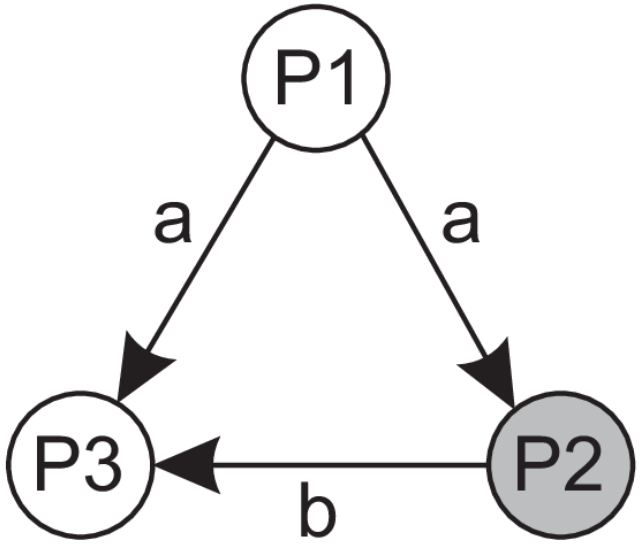


Coordinated Attack Leading to Victory **Uncoordinated Attack Leading to Defeat**

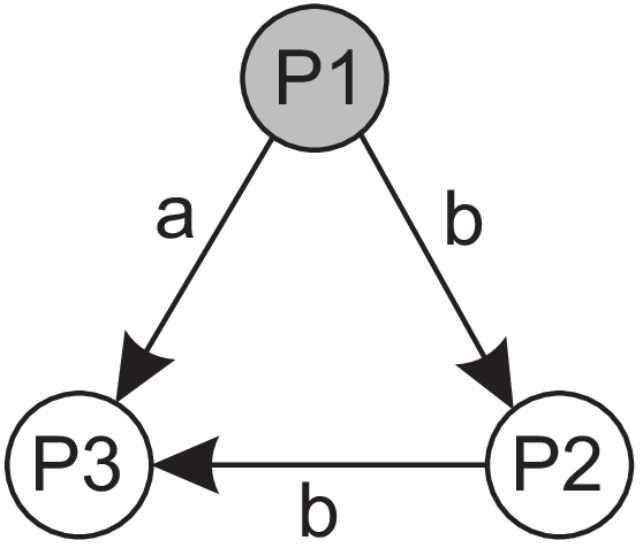
Izvor: https://medium.com/@DebrajG/how-the-byzantine-general-sacked-the-castle-a-look-into-blockchain-370fe637502c

Konsenzus sa proizvoljnim otkazima

- Posmatra se procesna grupa kod koje komunikacija između procesa nije konzistentna:
  - nepravilno prosleđivanje poruka
  - saopštavanje različitih stvari različitim procesima
- Primer procesa u repliciranoj grupi koji se ponaša nekonzistentno:



(a) nepravilno prosleđivanje poruka



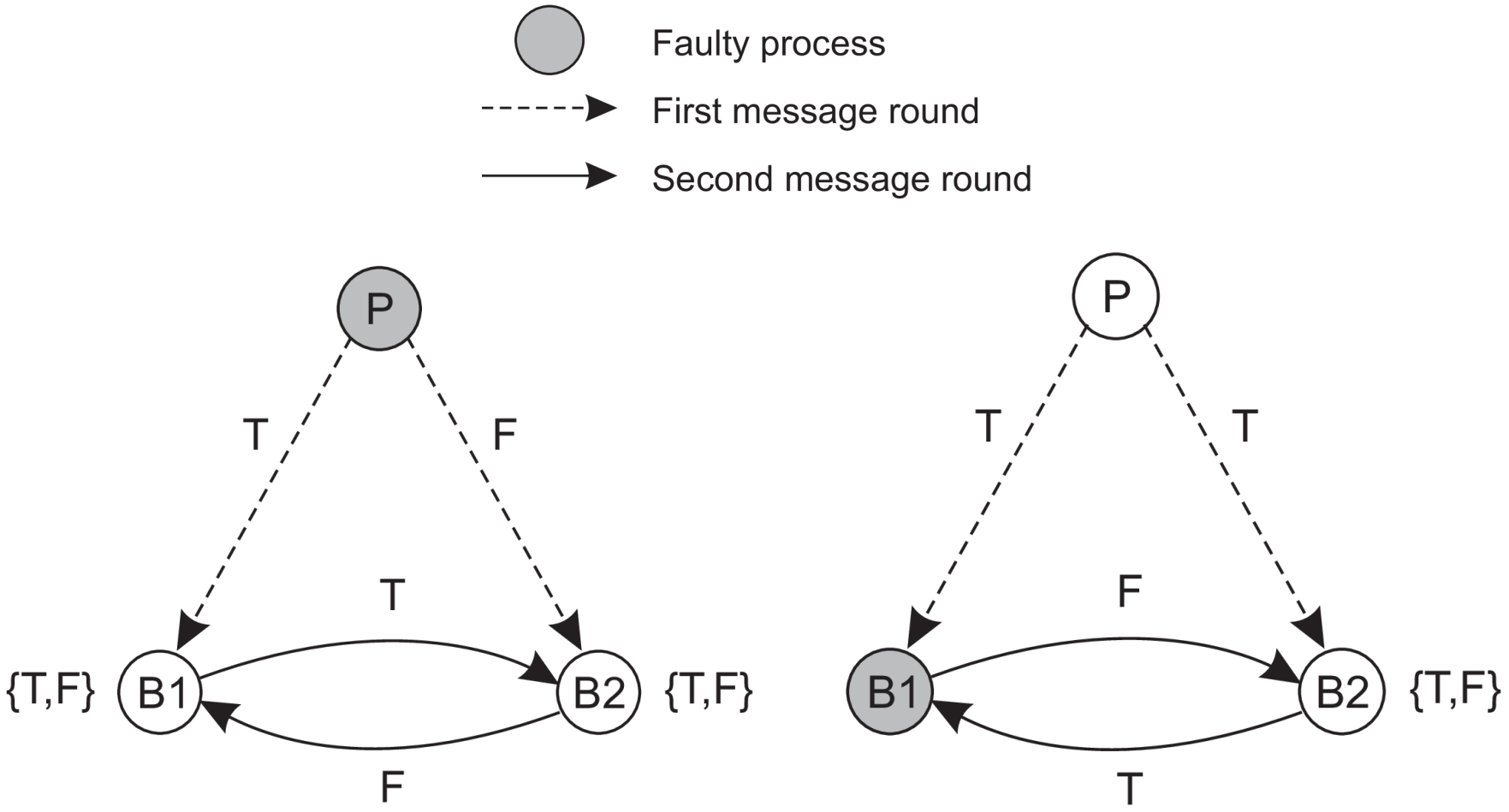
(b) saopštavanje različitih stvari različitim procesima

Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Konsenzus sa proizvoljnim otkazima

- Model sistema:
  - posmatramo **grupu** sa n članova, **primarni proces** P i **n - 1 bekapa** $B_1, B_2, \ldots, B_{n-1}$
  - klijent šalje $v \in \{T,F\}$ primarnom procesu P
  - poruke se mogu izgubiti, ali se takvo dešavanje može i detektovati
  - poruke ne mogu biti oštećene u toj meri da se ne mogu detektovati
  - primalac poruke može pouzdano detektovati njenog pošiljaoca
- Zahtevi za **vizantijski dogovor** (engl. ***Byzantine agreement***):
  - **BA1**: Svaki ispravni bekap proces čuva istu vrednost
  - **BA2**: Ako je primarni proces ispravan, onda svaki ispravan bekap proces čuva tačno ono što je primarni proces poslao
- Posledice:
  - primarni proces otkazao ⇒ BA1 kaže da bekapi mogu čuvati istu vrednost koja je različita od originalno poslate od klijenta i s toga pogrešna
  - primarni proces nije otkazao ⇒ zadovoljavanje BA2 implicira da je zahtev BA1 zadovoljen

Zašto 3*k* procesa nije dovoljno?



Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Zašto je $3k+1$ procesa dovoljno?