

UVOD U DISTRIBUIRANE TRANSAKCIJE



◆ ŠTA JE TRANSAKCIJA?

- Kolekcija operacija (čitanja i pisanja) koje želimo zajedno da obavimo nad bazom podataka u cilju obavljanja nekog posla definisanog na višem nivou (aplikativnom)

◆ TRANSAKCIJE U SQL-u

- Počinju sa BEGIN
- Završavaju se sa COMMIT ili ROLLBACK (ABORT)
 - Ako se završe sa COMMIT, baza ili sačuva sve promene ili radi ROLLBACK
 - Ako se završe sa ROLLBACK, sve promene se odbacuju i podaci se vraćaju u prvobitno stanje

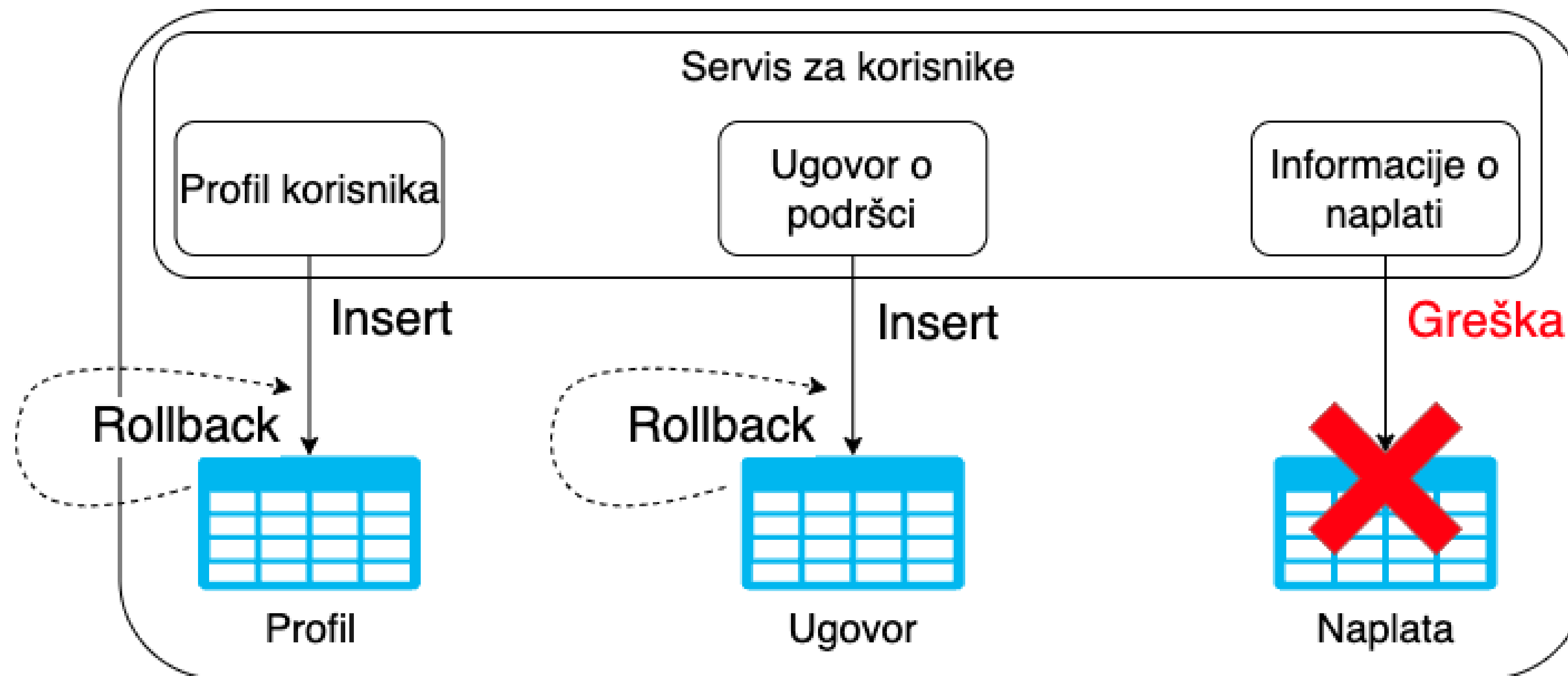


SVOJSTVA TRANSAKCIJA

3

◆ KOJA SVOJSTVA TRANSAKCIJA TREBA DA POSEDUJE?

- A - Atomičnost skupa operacija (Atomicity)
- C - Konzistentnost podataka (Consistency)
- I - Izolovanost operacija (Isolation)
- D - Izdržljivost podataka (Durability)





DISTRIBUIRANI SISTEMI

5

◆ ŠTA JE DISTRIBUIRANI SISTEM?

- Distribuirani sistem je skup nezavisnih računara koji korisnicima izgleda kao koherentan sistem

◆ DVA ASPEKTA

- Sistemi rade na više servera gde broj servera može da varira od svega par na stotine
- Sistemi upravljaju podacima što ih čini inherentno *stateful* sistemima



◆ ŠTA JE CAP TEOREMA?

- Pri pravljenju distribuiranih sistema (ne samo baza podataka) kao najvažniji ciljevi ističu se tri svojstva:
 - Konzistentnost (Consistency)
 - Dostupnost (Availability)
 - Tolerancija razdvajanja (Partition tolerance)
- Erik Bruer (Eric Brewer) uvodi pretpostavku koja kaže da se, u prisustvu razdvajanja mreže (Partition), mogu garantovati najviše dva od tri svojstva za bilo koji zajednički sistem podataka
- CAP nije izbor “šta želimo”, već “šta žrtvujemo kada se dogodi razdvajanje mreže”
- Ne bira se C ili A stalno — biraju se samo kada mreža pukne



CAP TEOREMA

7

◆ KONZISTENTNOST

- *Konzistentan sistem ili funkcioniše kao celina ili ne funkcioniše uopšte*
- *Sva čitanja, na svim čvorovima, moraju da daju isti rezultat (u odnosu na redosled operacija)*
- *Rezultat operacije (čitanja) nikada ne zavisi od čvora na kome se izvršava*

◆ TIPIČNA POSLEDICA

- *Ako se sistem ne može da dogovori koji je podatak tačan radije neće odgovoriti nego da vrati pogrešan podatak*



◆ DOSTUPNOST

- *Sistem uvek pokušava da odgovori (čak i kada nije siguran da li je odgovor globalno ispravan)*
- Dostupnost se praktično definiše kao odziv sistema u nekim garantovanim granicama
- Sistem obično nije dostupan upravo kada je potreban
- U vreme kada je dostupnost najpotrebnija, onda je i najteže ostvariva, zato što je tada sistem najviše opterećen
- Ako je sistem dostupan kada nije potreban, to nema značaja

◆ TIPIČNA POSLEDICA

- Ako čvor ima podatke, vratiće ih čak i ako nisu sinhronizovani sa ostatkom sistema

◆ TOLERANCIJA RAZDVOJENOSTI

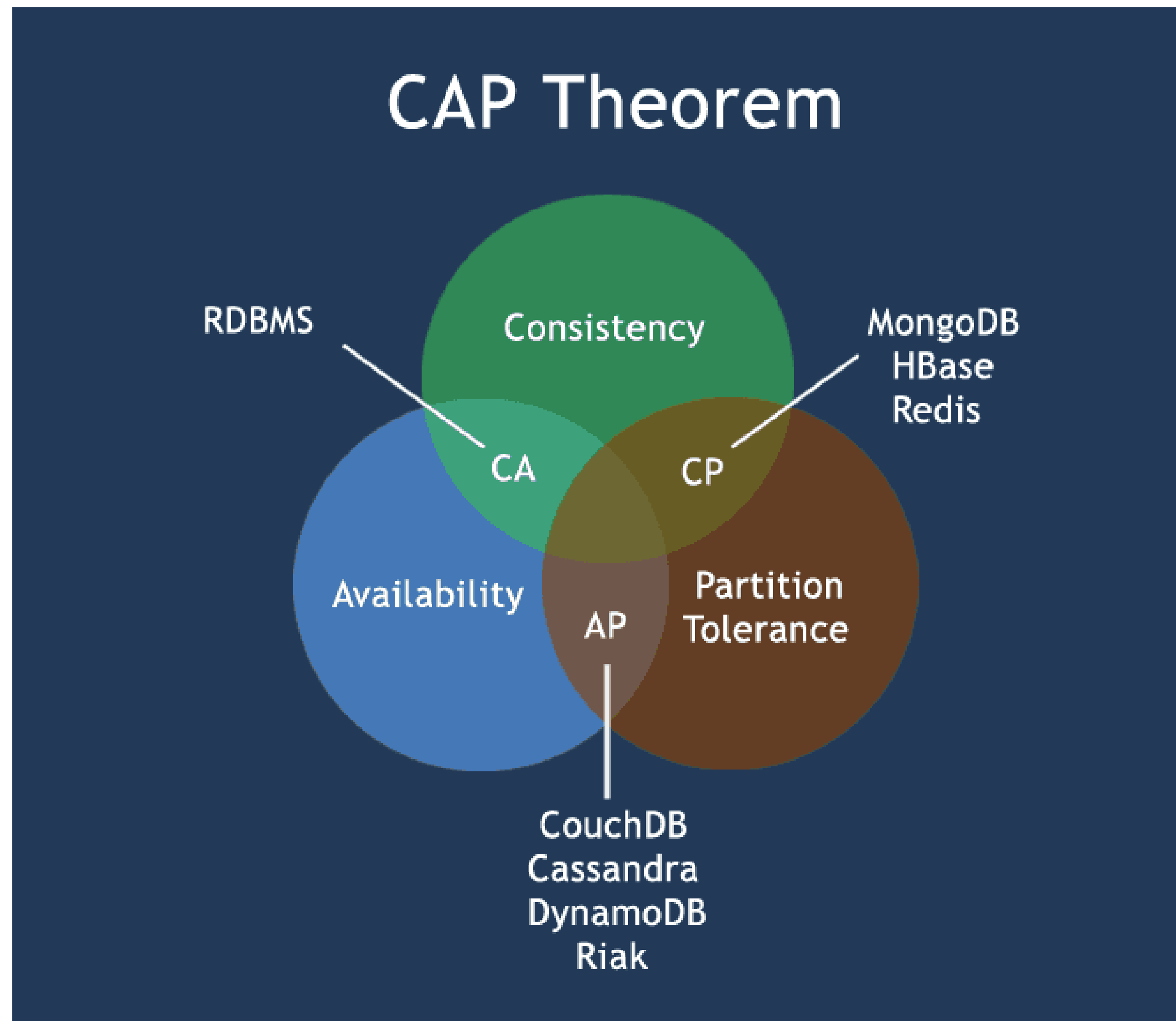
- *Nijedan skup problema, osim potpunog otkazivanja, ne sme da proizvede neispravan odziv sistema*
- Sistem mora da prihvata delimične otkaze komunikacije i da nastavlja ispravan rad (u ovom kontekstu se razmatra ispravnost odziva)
- Povremeni prekidi komunikacije među čvorovima su neizbežni
- Razdvojenost (partition) je stanje komunikacione mreže u kome su delovi sistema (obično usled kvara) podeljeni na particije (dva ili više razdvojenih skupova čvorova) između kojih ne postoji komunikacija

◆ VAŽNA IMPLIKACIJA

- Ako sistem nije tolerantan na razdvajanje, on nije distribuiran sistem ili se oslanja na savršenu mrežu (što ne postoji)
- *Zato se realnim distribuiranim sistemima, P se podrazumeva.*



CAP TEOREMA





CAP TEOREMA

Izbor	Šta sistem radi
CP	Blokira zahteve → čuva konzistentnost
AP	Odgovara zahtevima → žrtvuje konzistentnost
CA	Moguće samo ako nema razdvajanja mreže (teorijski ili single-node)



◆ **KOMPROMISI**

- Pri projektovanju (ili konfigurisanju) distribuiranog sistema neophodno je napraviti neki kompromis:
 - odbacivanje konzistentnosti
 - odbacivanje raspoloživosti
 - odbacivanje tolerancije razdvojenosti
 - odbacivanje više strogo definisanih uslova
 - zasnivanje sistema na drugačijem skupu uslova



KOMPROMISI

13

◆ ODBACIVANJE KONZISTENTNOSTI

- Dopušta se da isti upit daje različite rezultate na različitim čvorovima
- Ova opcija se koristi kada su razlike u podacima prihvatljivije od niske dostupnosti i ako je cena nekonzistentnosti prihvatljiva
 - Društvene mreže (broj lajkova)
 - Analitika
 - Keširani prikazi podataka
- Konzistentnost je jedan od osnovnih uslova za uspešan rad baza podataka

◆ ODBACIVANJE DOSTUPNOSTI

- U slučaju razdvojenosti se ne garantuje vreme odziva
- Problem se redukuje pažljivim projektovanjem sistema, odnosno uspostavljanjem što niže sprege među čvorovima
- Sistem koji nije raspoloživ je praktično neupotrebljiv
- Zbog toga se ovaj pristup koristi relativno retko
 - Sistemi za koordinaciju
 - Konfiguracioni servisi
 - Sistemi gde greška ima visoku cenu (npr. finansije)
- Ako je konzistentnost primarna i ako je tolerancija razdvojenosti nezaobilazna, onda se teži niskoj spregnutosti među čvorovima

◆ ODBACIVANJE TOLERANCIJE RAZDVOJENOSTI

- Pristaje se da sistem ne radi u slučaju razdvojenosti
- Jedan način prevazilaženja je da sve bude na jednoj mašini (tada je particionisanje besmisleno)
- Alternativa je da se višestrukim umrežavanjem razdvojenost (a time i otkaz sistema) svede na najmanju moguću meru (redudancija)
- Mana ovog pristupa je da su obe alternative veoma skupe
- Imajući u vidu da se distriburana rešenja koriste samo onda kada su neophodna zbog obima podataka i poslova, ovaj vid kompromisa se retko pravi

◆ DRUGAČIJI SKUP USLOVA

- Dostupnost i tolerancija razdvojenosti nisu deo ACID garancija transakcije, tako da je možda moguće da se od njih odustane da bi se sačuvao integritet baze podataka
- Međutim, to možda nije najbolji izbor u svim okruženjima jer ograničava sposobnost sistema da se skalira i bude visoko dostupan
- U mnogim okruženjima, dostupnost i tolerancija razdvojenosti su važniji od konzistentnosti
- Da bi se garantovalo ACID ponašanje u transakcijama, objekti (npr. delovi baze podataka) moraju biti zaključani tako da svi vide konzistentne podatke, što uključuje da drugi entiteti moraju da čekaju dok ti podaci ne budu dosledni i otključani
- Zaključavanje dobro funkcioniše ali je to teško efikasno uraditi na velikim sistemima sa velikim količinama podataka
- Umesto toga treba razmotriti korišćenje keširanih podataka
 - Rizik je da prekršimo „C“ i „I“ u ACID - dve odvojene transakcije mogu da vide različite prikaze istih podataka

◆ DRUGAČIJI SKUP USLOVA

- Alternativa strogim zahtevima ACID-a je BASE:
 - *Basic Availability* - ne garantuje se raspoloživost odgovora, već samo sistema; to znači da ako korisnik ne može da se dobije odgovor na zahtev, bar će se dobiti obaveštenje o tome
 - *Soft-state* - stanje sistema može da se menja čak i kada nije u toku nijedna transakcija, na primer radi postizanja konzistentnosti
 - *Eventual consistency* - žrtvuju se garantovana stalna konzistentnost i izolovanost transakcija zarad dostupnosti; sistem će u nekom trenutku (*eventually*) postati konzistentan ali će raditi i davati odgovore (potencijalno različite) do tada

◆ ŠTA SU DISTRIBUIRANE TRANSAKCIJE?

- Transakcije koje za izvršavanje zahtevaju pristup podacima koji se nalaze na različitim lokacijama (čvorovima)
- Transakcije koje se prožimaju kroz više različitih sistema zovu se i globalne transakcije
- Globalne transakcije se sastoje iz lokalnih transakcija koje se izvršavaju na pojedinačnim čvorovima zbog čega je potrebna koordinacija

◆ ZAŠTO SE IZUČAVAJU?

- Ne moraju se svi potrebni podaci naći u jednom čvoru
- Određena svojstva je teže implementirati
- Mehanizmi koji se koriste za rešavanje problema na jednom čvoru nisu dovoljni



◆ ZAŠTO IH JE TEŠKO IMPLEMENTIRATI?

- Atomičnost
 - Različiti delovi podataka iz transakcije se možda nalaze na drugim lokacijama
 - Kako da obezbedimo da je sve ili ništa izvršeno?
- Konzistentnost
 - Otkaz može da utiče samo na jedan deo transakcije
- Izolovanost
 - *Commit* mora da se desi “simultano” na svim lokacijama
- Izdržljivost
 - Isti problemi kao i kod “običnih” transakcija



◆ KOJI SU GLAVNI PROBLEMI?

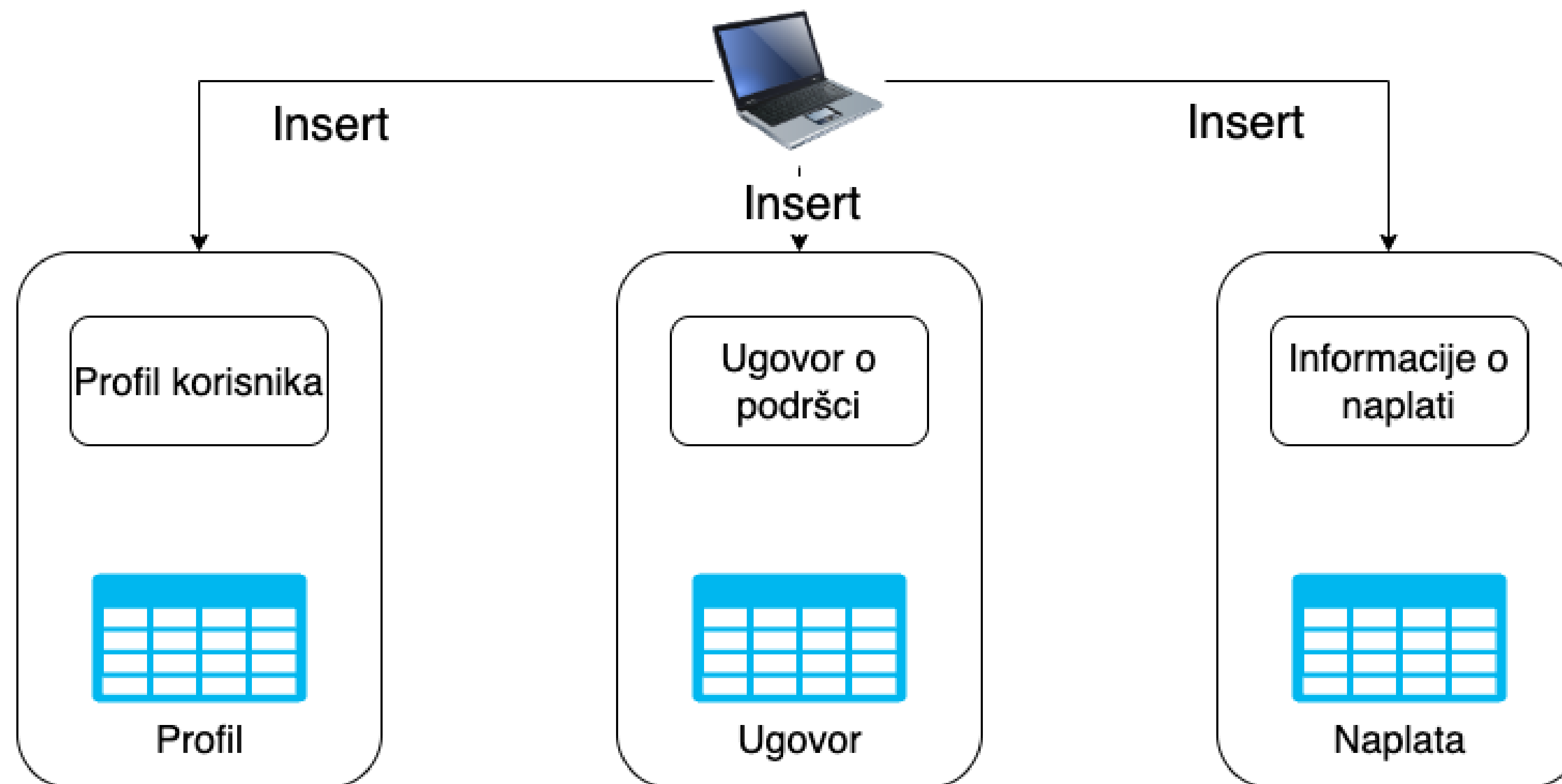
- *Commit*
 - Standardne tehnike čuvaju svojstva transakcija kada se desi *commit*
 - Distribuirani sistemi moraju da imaju *commit* protokol da bi se znalo kada se *commit* desio
- Otkazi
 - Standardne tehnike podržavaju izdržljivost u slučaju *commit* ili *abort* operacije
 - Šta se dešava ako čvor koji učestvuje u transakciji otkaže tokom *commit*-a?



◆ KOJI SU GLAVNI PROBLEMI?

- Ako bi se poslala *commit* naredba svakom čvoru, transakcije bi se izvršavale nezavisno
- Lako se može desiti da se transakcija uspešno izvrši na nekim čvorovima, dok na drugim izvrši neuspešno čime se narušava osobina atomičnosti:
 - Neki čvorovi mogu da detektuju narušavanje ograničenja ili konflikt i time započnu *abort*, dok drugi čvorovi mogu da izvrše transakciju bez problema
 - Neki zahtevi za *commit* mogu da se izgube u mrežnoj komunikaciji i ti zahtevi mogu da rezultuju *timeout*-om
 - Neki čvorovi mogu da otkažu pre nego što se transakcija u potpunosti izvrši i pokrene se *rollback*

DISTRIBUIRANE TRANSAKCIJE

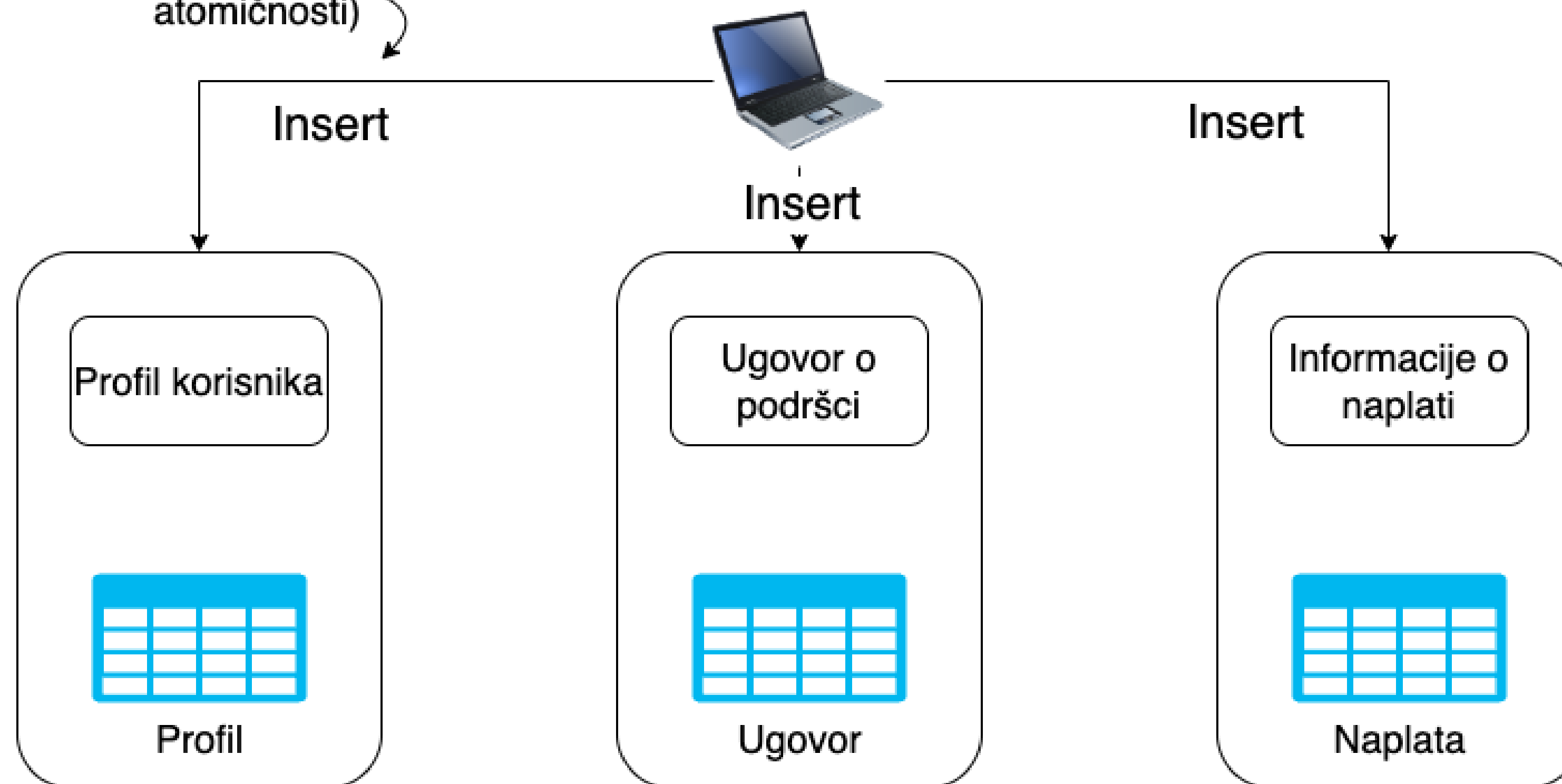




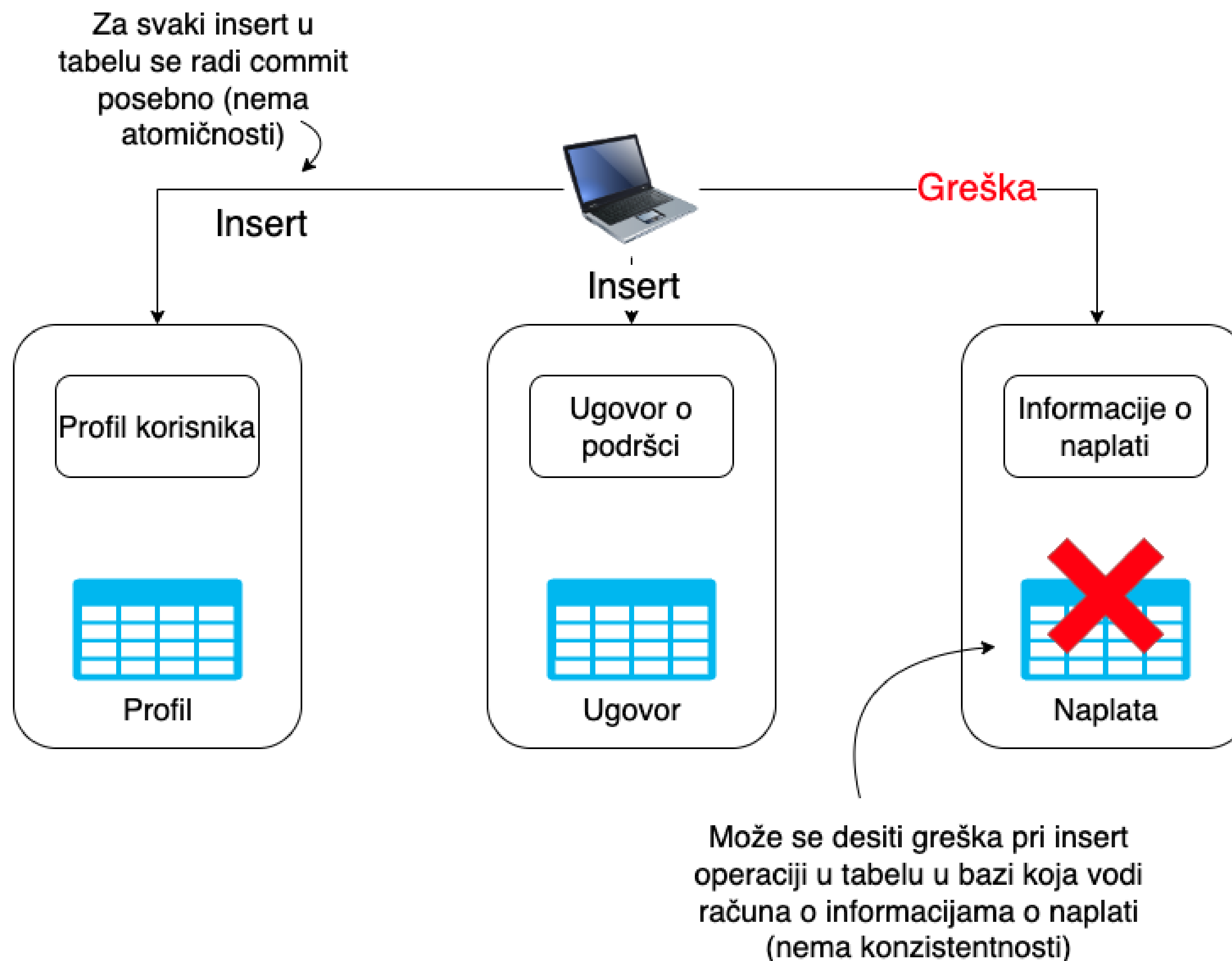
DISTRIBUIRANE TRANSAKCIJE

23

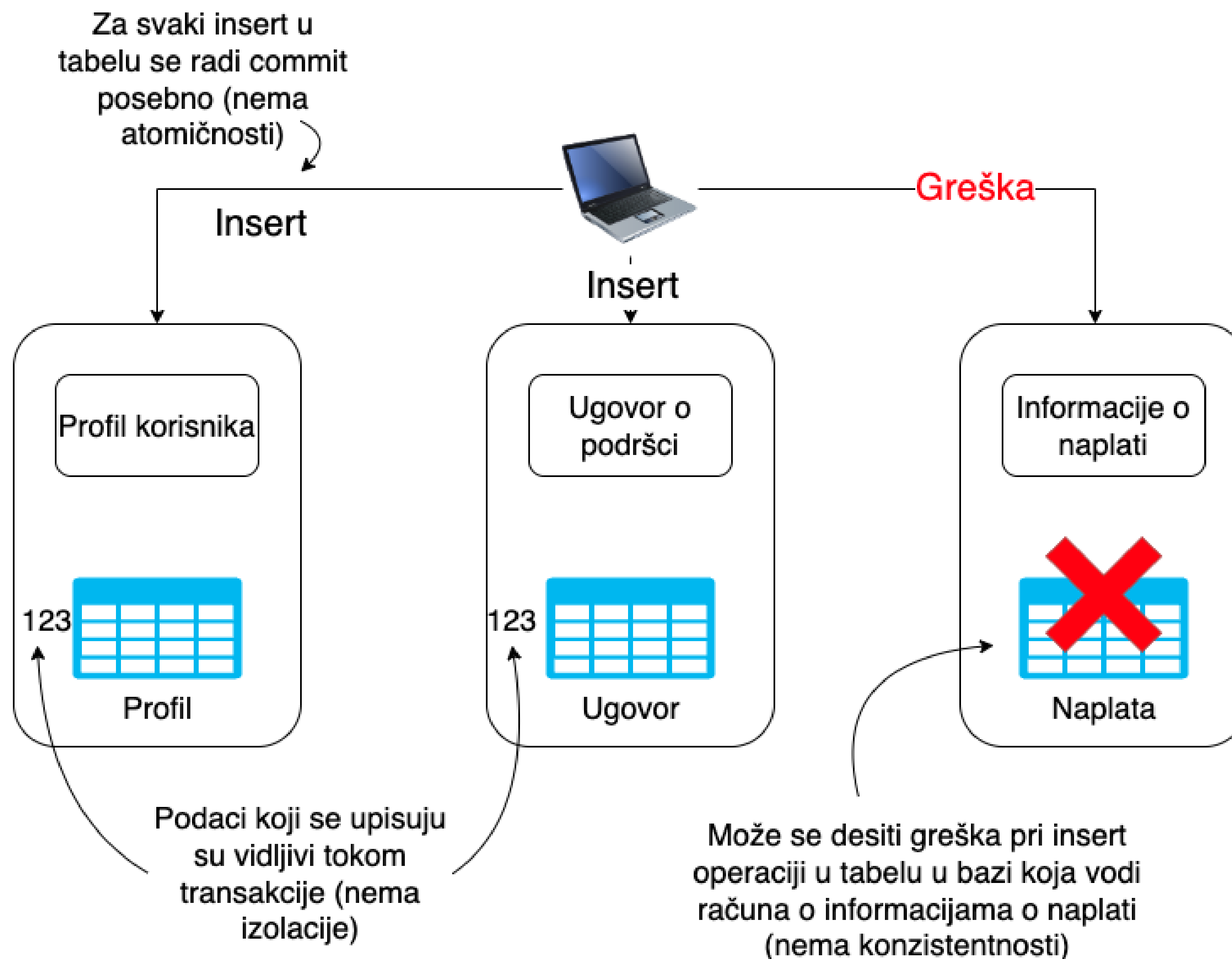
Za svaki insert u tabelu se radi commit posebno (nema atomičnosti)



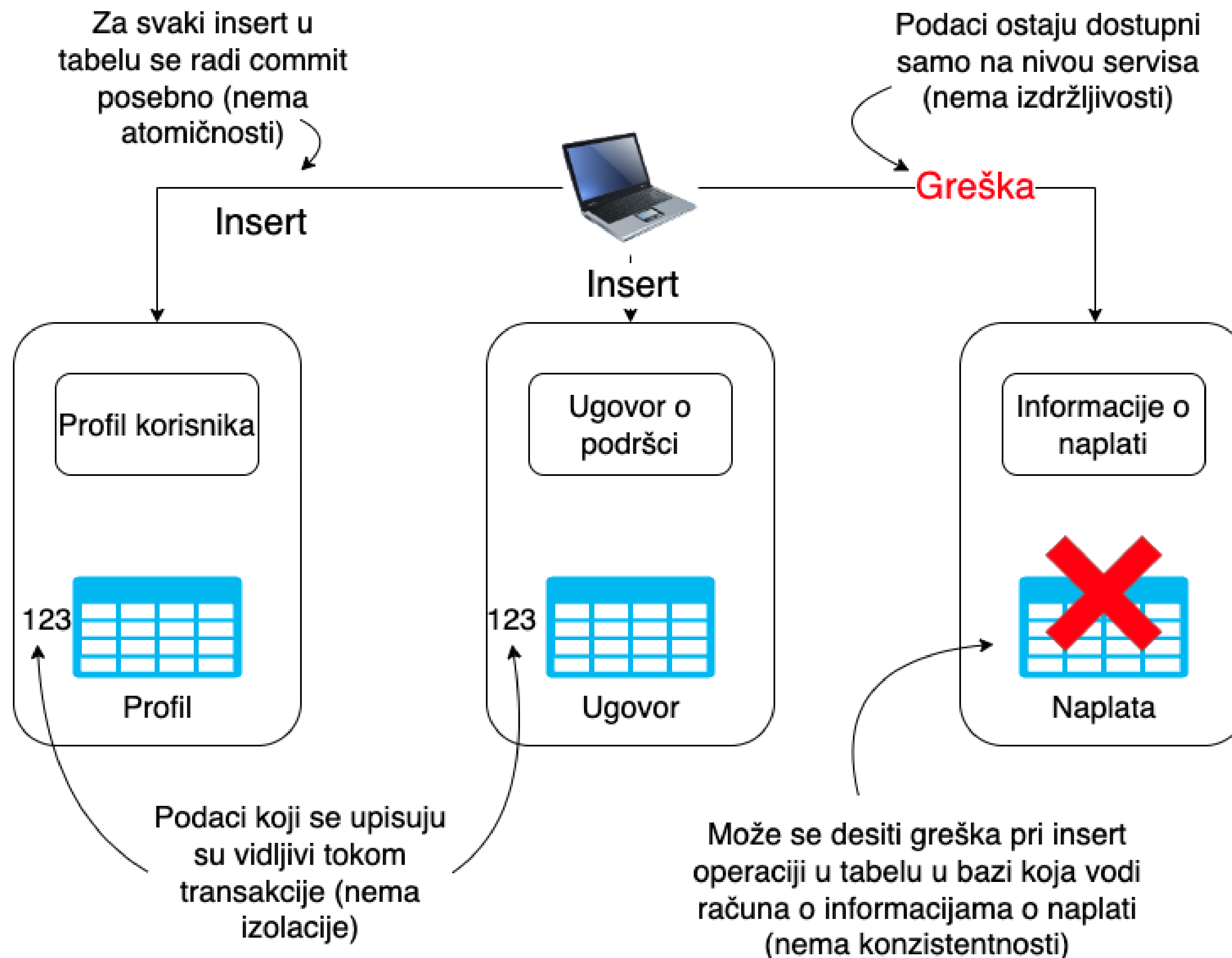
DISTRIBUIRANE TRANSAKCIJE



DISTRIBUIRANE TRANSAKCIJE



DISTRIBUIRANE TRANSAKCIJE





TWO-PHASE COMMIT

27

◆ ŠTA JE 2PC?

- Algoritam za postizanje atomičnog *commit*-a transakcija na više čvorova
- Blokirajući je algoritam
- Sastoji se iz dve faze:
 - Faza pripreme – proverava se da li transakcija može da se izvrši
 - *Commit* faza – radi se *commit* transakcije
- Uvodi dodatnu komponentu koja koordiniše fazama – koordinator ili transakcioni menadžer



TWO-PHASE COMMIT

28

◆ ŠTA JE 2PC?

- 2PC započinje tako što aplikacija čita i/ili piše podatke na više čvorova
- Kada je aplikacija spremna za *commit* izmena, koordinator kreće u prvu fazu i šalje *prepare* zahtev svakom čvoru
 - Svaki čvor proverava da li može da izvrši transakciju, zapisuje odluku u log, zadržava zaključavanja i odgovara sa DA ili NE
- U Commit fazi:
 - Ako svi čvorovi odgovore sa DA, koordinator šalje *commit* zahtev svim čvorovima u drugoj fazi
 - Ako bar jedan čvor odgovori sa NE, koordinator šalje *abort* zahtev svim čvorovima u drugoj fazi

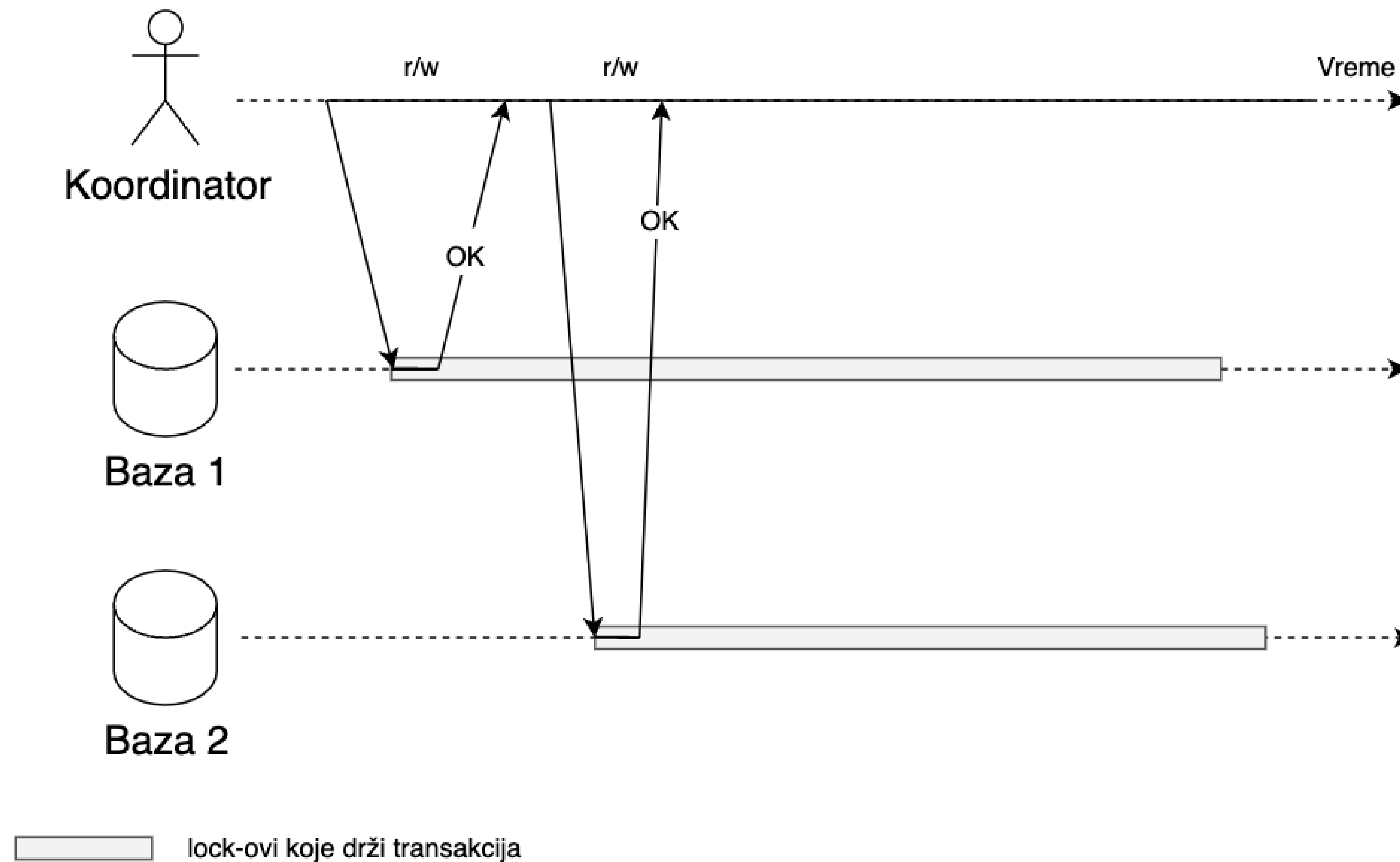


TWO-PHASE COMMIT

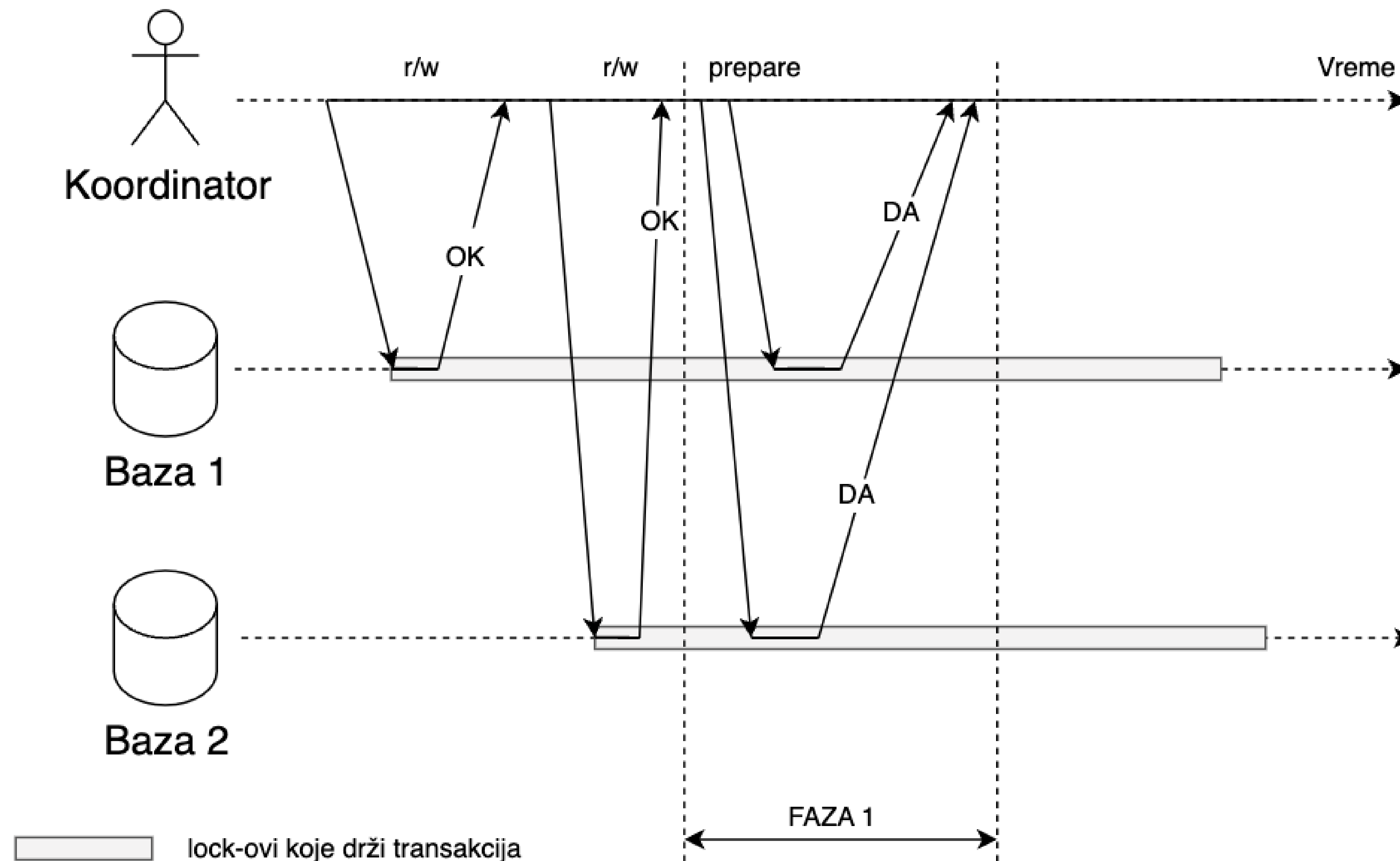
29



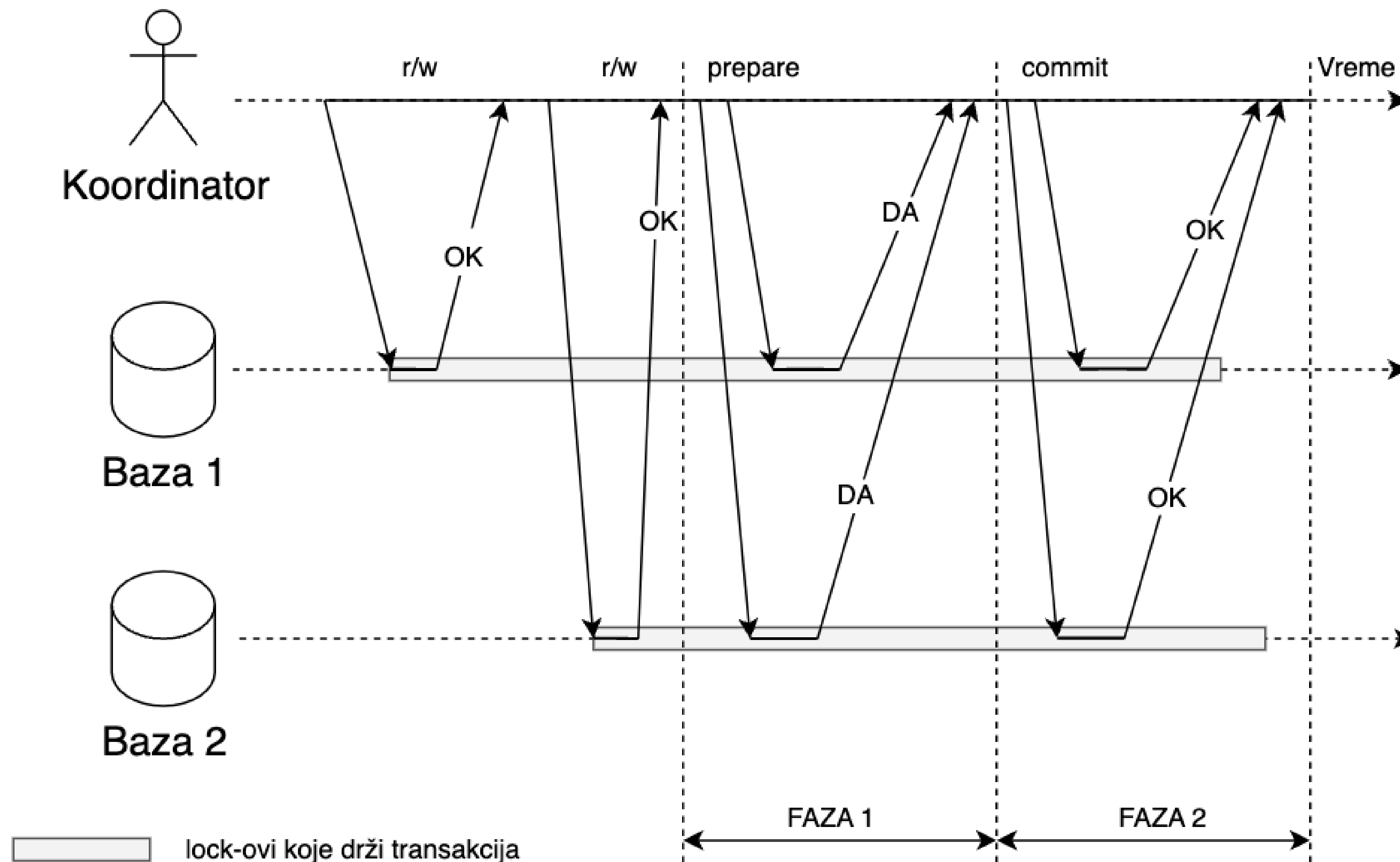
TWO-PHASE COMMIT



TWO-PHASE COMMIT



TWO-PHASE COMMIT





TWO-PHASE COMMIT

33

◆ KOJI SU GLAVNI PROBLEMI?

- Ako čvor (učesnik u transakciji) postane nedostupan u fazi pripreme:
 - Koordinator ne dobija uniformno DA i radi se *abort*
- Ako čvor (učesnik u transakciji) postane nedostupan u *commit* fazi:
 - Kada se oporavi, može da pita koordinatora da li da radi *commit* (*redo*) ili *abort*
 - Zahteva da oba stanja (i *commit* i *abort*) budu zapamćena u logu (*durability*) pre slanja odgovora
- Ako koordinator postane nedostupan pre faze pripreme učesnici mogu bezbedno da odustanu od transakcije
- Ako čvorovi dobiju *prepare* zahtev i daju odgovor DA, moraju da čekaju odgovor od koordinatora da čuju da li je transakcija komitovana ili otkazana (mogu zauvek da ostanu blokirani)
 - Ako koordinator postane nedostupan u ovom trenutku čvorovi čekaju da se oporavi (transakcije moraju ostati izlovene)
 - Zato koordinator treba da zapiše sve odluke o *commit*-u ili *abort*-u u transakcioni log pre slanja odluke čvorovima kako bi posle oporavka mogao da prosledi adekvatnu informaciju



TWO-PHASE COMMIT

34

◆ PREDNOSTI

- Jednostavno za razumevanje i implementaciju
- Podržava ACID

◆ MANE

- Mehanizmi zaključavanja i duge konekcije tokom transakcija dovode do opadanja performansi – blokirajući protokol
- Jako sprezanje između svih učesnika u komunikaciji
- Slaba otpornost na greške u komunikaciji
- Koordinator je *Single Point of Failure* (SPoF)

◆ ŠTA AKO SE TRANSAKCIJE NE IZVRŠE NA SVIM ČVOROVIMA?

- Svi čvorovi se moraju dogovoriti o ishodu – ili će svi raditi *abort* (ako nešto pođe po zlu) ili će raditi *commit* (ako sve prođe kako treba)
- Jedan ili više čvorova će predložiti vrednosti, a konsenzus algoritam će se odlučiti za jednu od njih
- Takav algoritam treba da zadovolji nekoliko svojstava:
 - Uniformni dogovor – nijedna dva čvora ne mogu dogovoriti različit ishod
 - Integritet – nijedan čvor ne može da odlučuje dvaput
 - Validnost – ako je neki čvor izabrao vrednost, znači da je neki čvor tu vrednost morao predložiti
 - Terminacija – svaki čvor koji ne otkaže, odlučuje o nekoj vrednosti
- Poznati algoritmi za konsenzus:
 - Viewstamped Replication (VSR)
 - Paxos
 - Raft
 - Zab



THREE-PHASE COMMIT

36

◆ ŠTA JE 3PC?

- Varijacija koja definiše gornju granicu vremena za koju transakcija treba da se komituje ili otkaže
- Pruža rešenje za blokirajuću prirodu 2PC
- *Commit* faza se dekomponuje na dve potfaze:
 - Priprema za *commit* - koordinator šalje poruku svim čvorovima kada dobije DA u prvoj fazi
 - Ako čvor dobije *prepare-to-commit* poruku, priprema sve za *commit* (resurse i *lock-ove*), šalje odgovor da je spreman i čeka
 - Ako čvor dobije *abort* poruku ili istekne period čekanja poruke od koordinatora, radi se *abort* transakcije (puštaju se svi *lock-ovi* nad resursima)
 - *Commit* faza – ista kao i u 2PC, ako dobije *prepare-to-commit* odgovor od svih čvorova šalje *commit* poruku, u suprotnom radi *abort*



THREE-PHASE COMMIT

37

◆ ŠTA JE 3PC?

- Čvorovi u fazi pripreme za *commit* imaju informaciju šta je konsenzus tako da ako koordinator otkaže, mogu da pošalju informaciju rezervnom koordinatoru kakva je bila odluka (*commit* ili *abort*)
- Ako je svaki čvor dobio *prepare-to-commit* poruku, koordinator može da traži *commit*
- Ako su samo neki čvorovi dobili *prepare-to-commit* poruku, koordinator zna da je jednoglasna odluka trebala da bude *commit* i može ponovo da izda zahtev
- Ako nijedan čvor nije dobio *prepare-to-commit* poruku, koordinator može da restartuje protokol



THREE-PHASE COMMIT

38

◆ ZAŠTO SE NE KORISTI U PRAKSI?

- Zahteva delimično sinhronu mrežu gde se zna granica kašnjenja poruka (timeout-i moraju biti pouzdani)
- U realnim sistemima kašnjenje \neq otkaz
- Osetljiv na pogrešno podešene timeout-e
- Veća složenost od 2PC
- Mali praktični dobitak

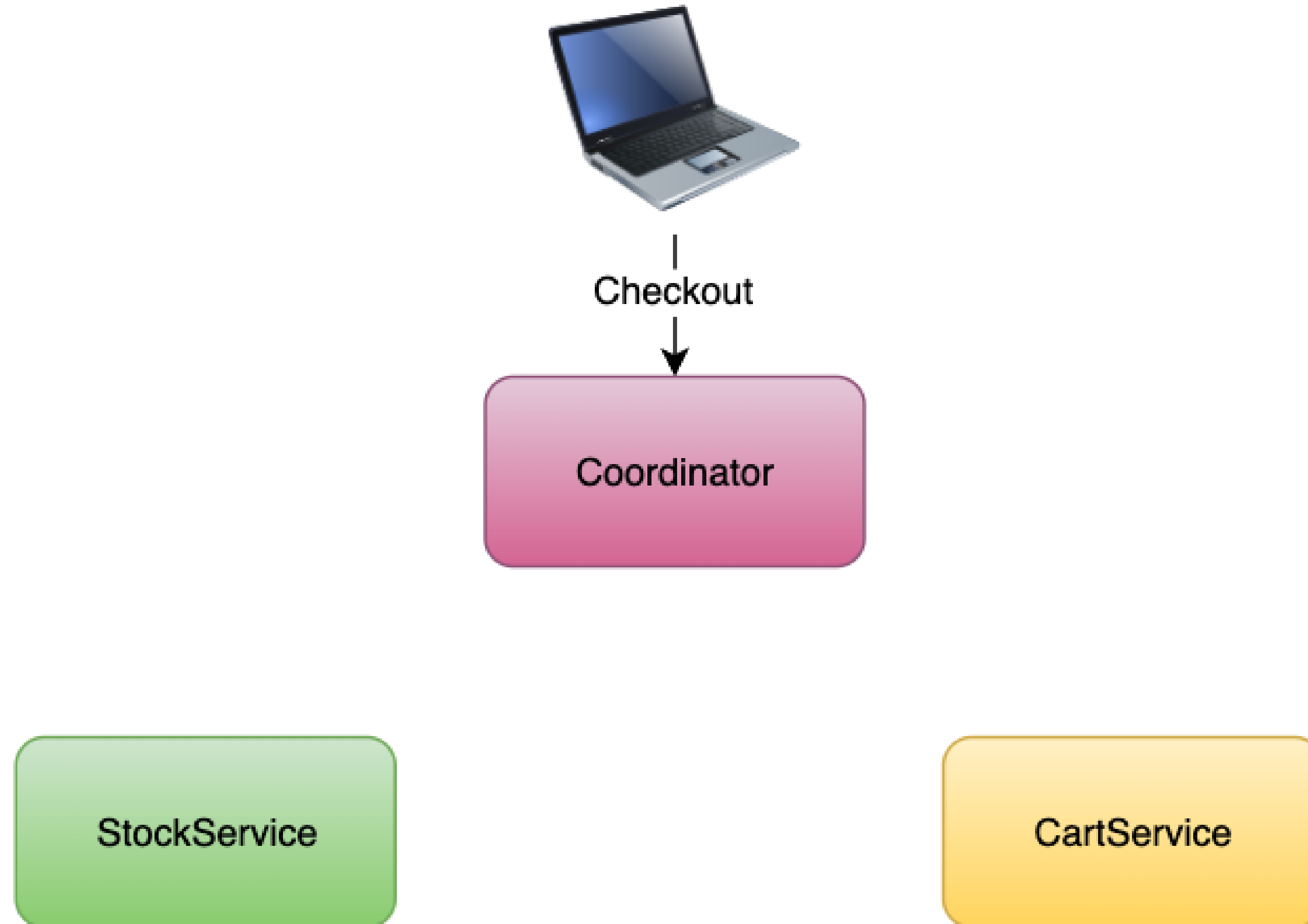
◆ ŠTA JE SAGA?

- Pristup koji transakciju razdvaja na lanac više lokalnih podtransakcija (koraka)
- Saga koordinator transakcija izvršava korake u lancu transakcija
- Svakoj operaciji transakcije u lancu transakcija odgovara operacija reverzibilne transakcije
- Ako korak propadne, šalje se kompenzaciona operacija (*undo*)
- Koordinacija koraka se sprovodi na dva načina:
 - Orkestrirana saga – centralni koordinator poziva sledeci korak
 - Koreografsana saga – svaki servis poziva sledeći
- Popularan pristup za rukovanje transakcijama, pogotovo u mikroservisnoj arhitekturi



ORKESTRIRANA SAGA

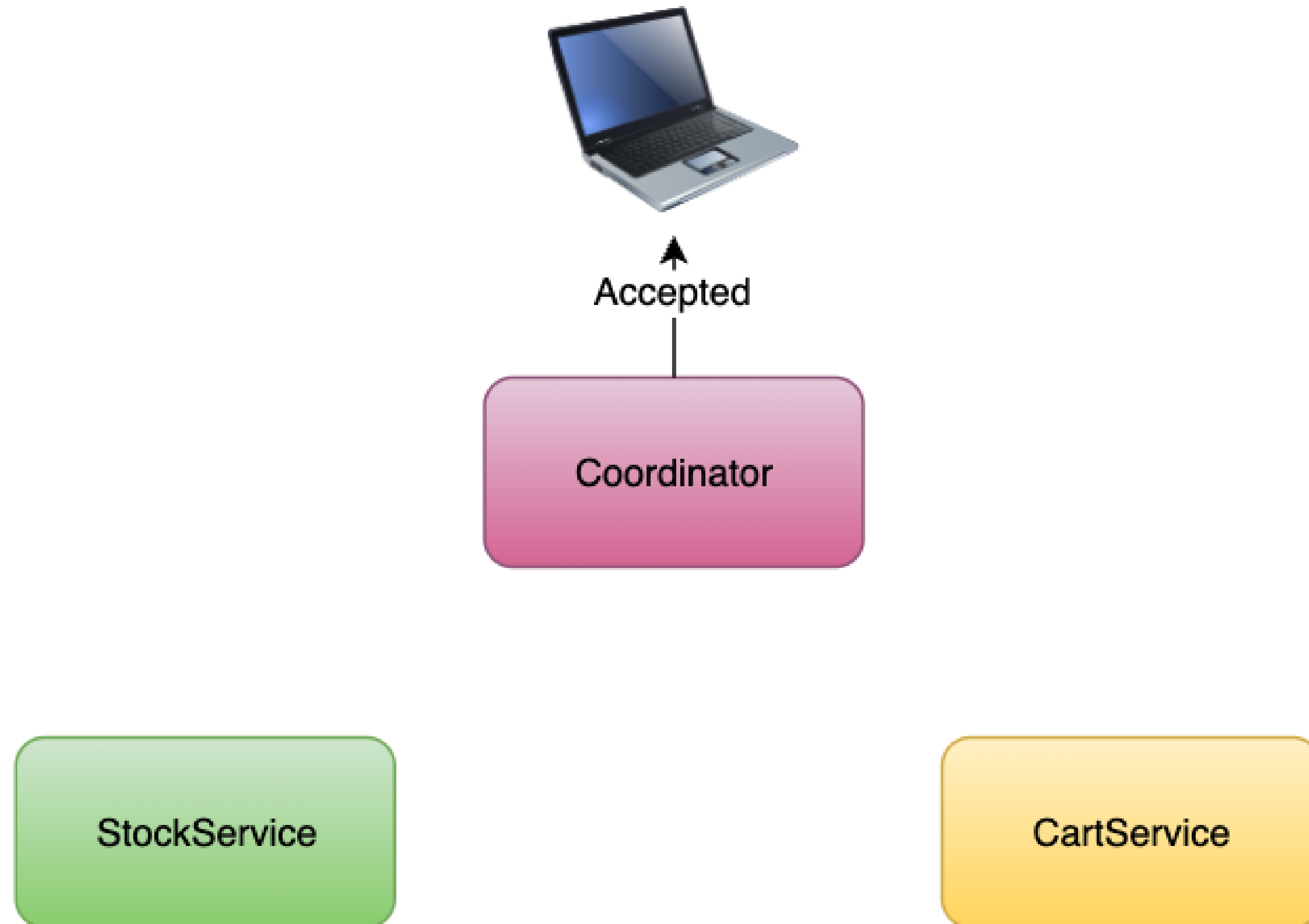
40





ORKESTRIRANA SAGA

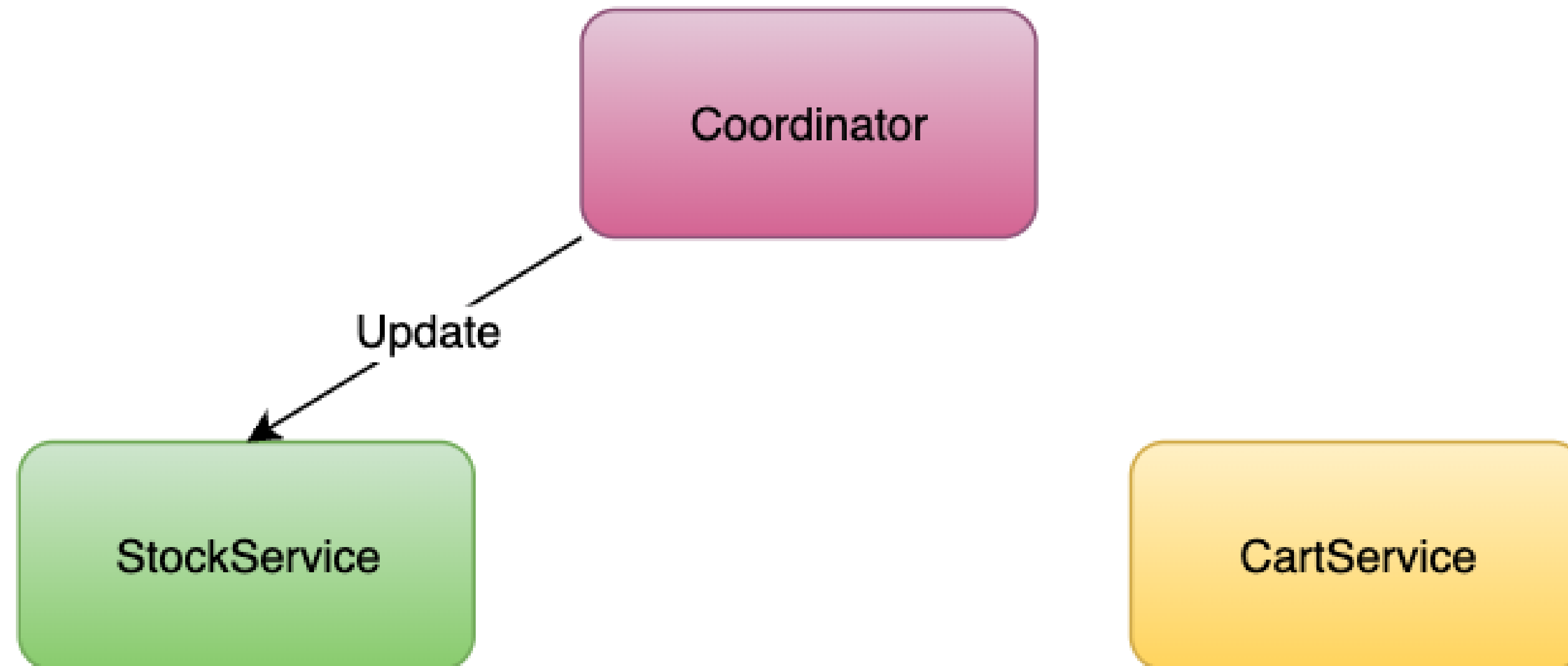
41





ORKESTRIRANA SAGA

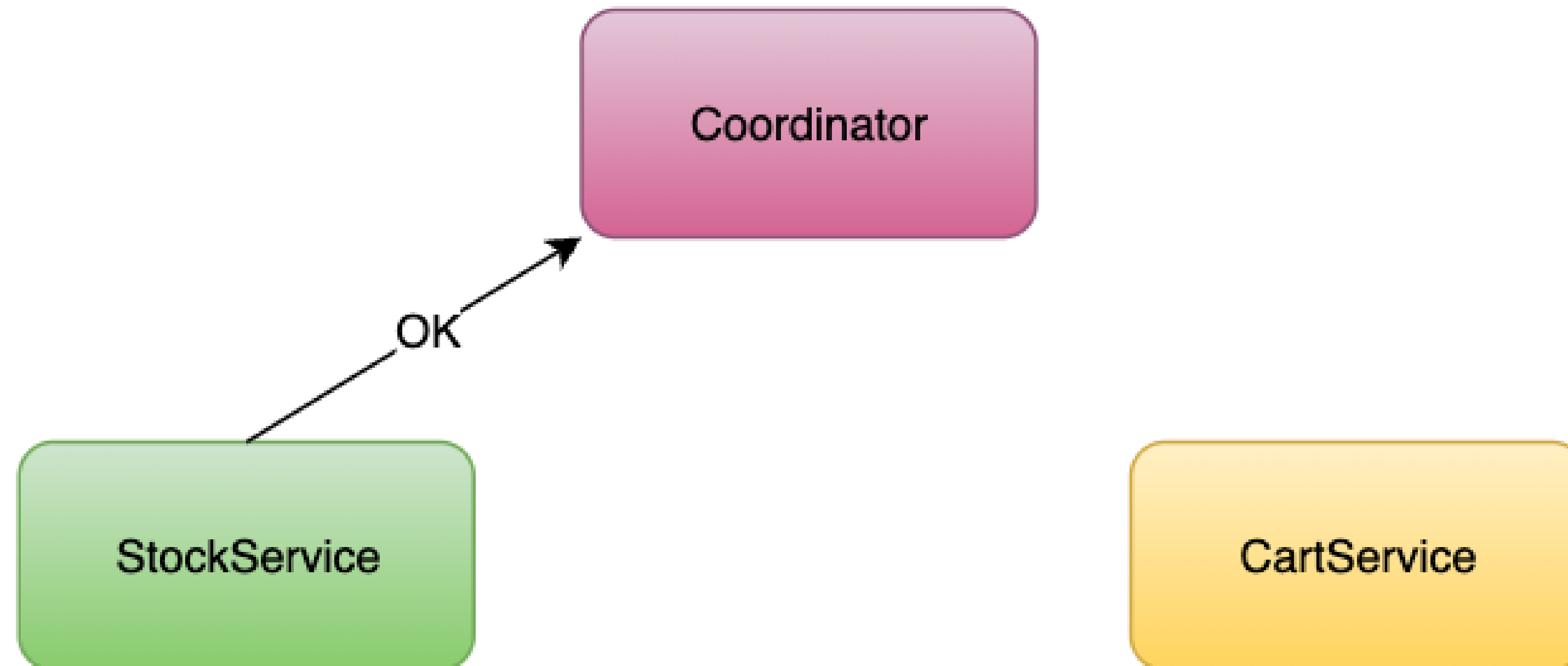
42





ORKESTRIRANA SAGA

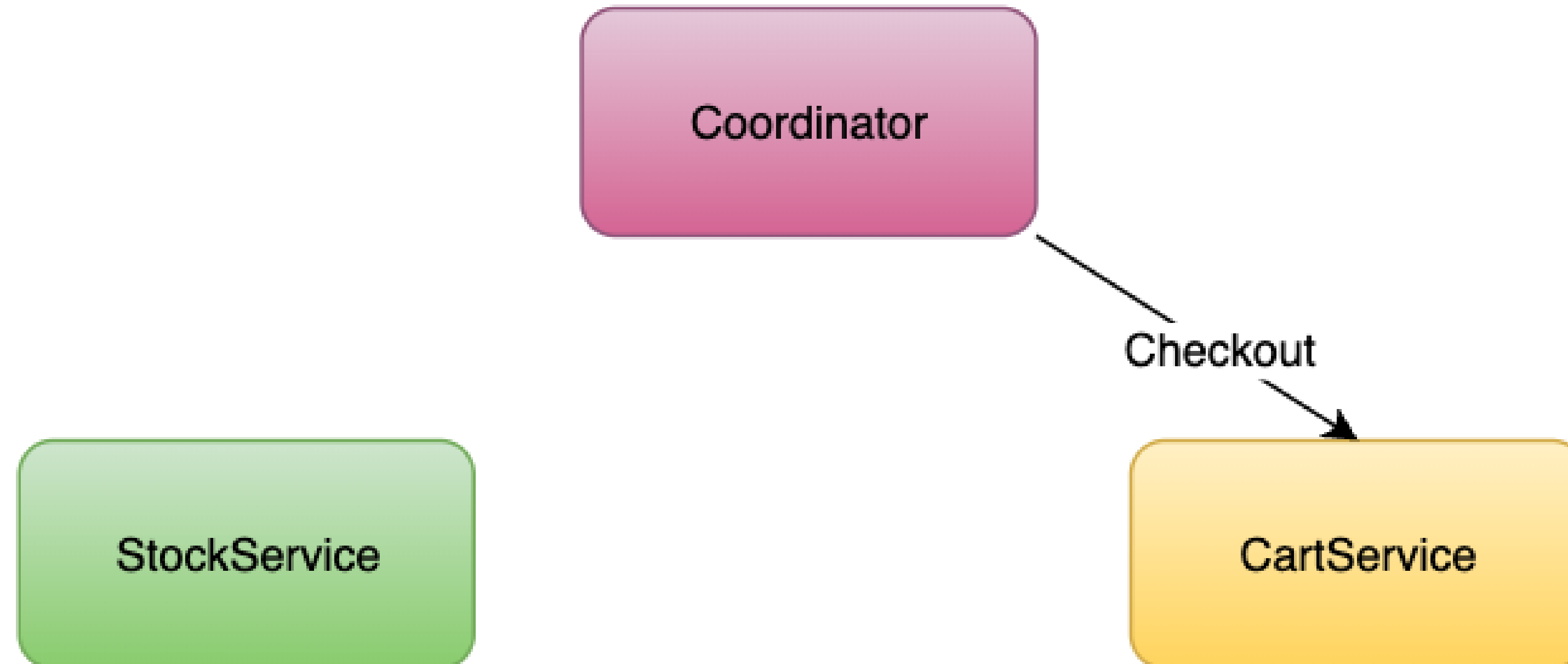
43





ORKESTRIRANA SAGA

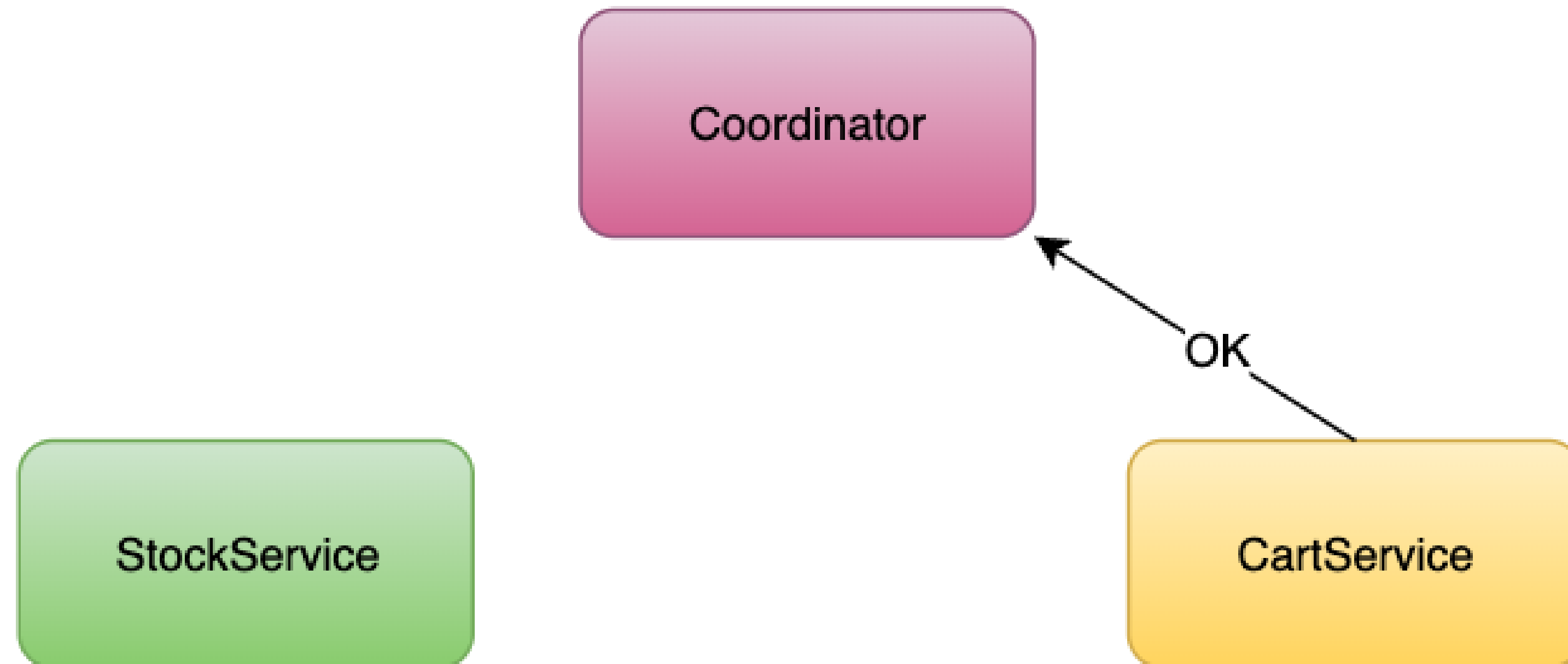
44





ORKESTRIRANA SAGA

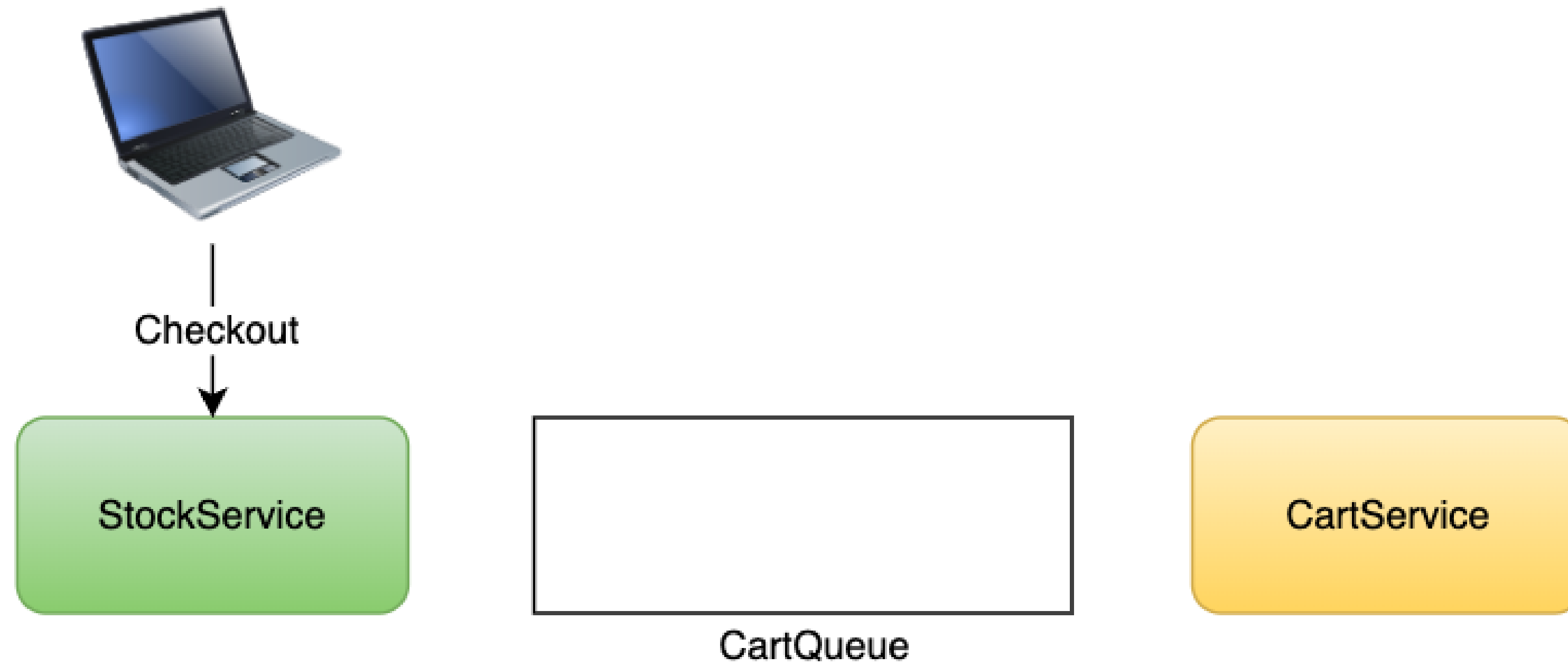
45





KOREOGRAFISANA SAGA

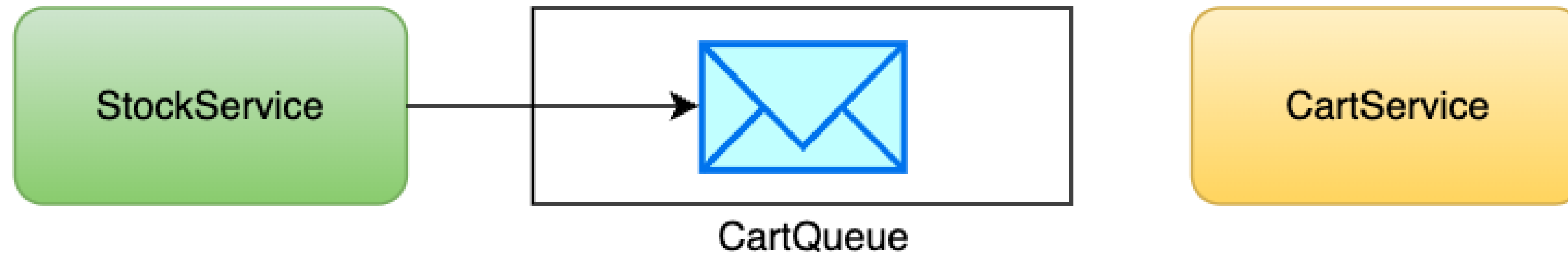
46





KOREOGRAPHISANA SAGA

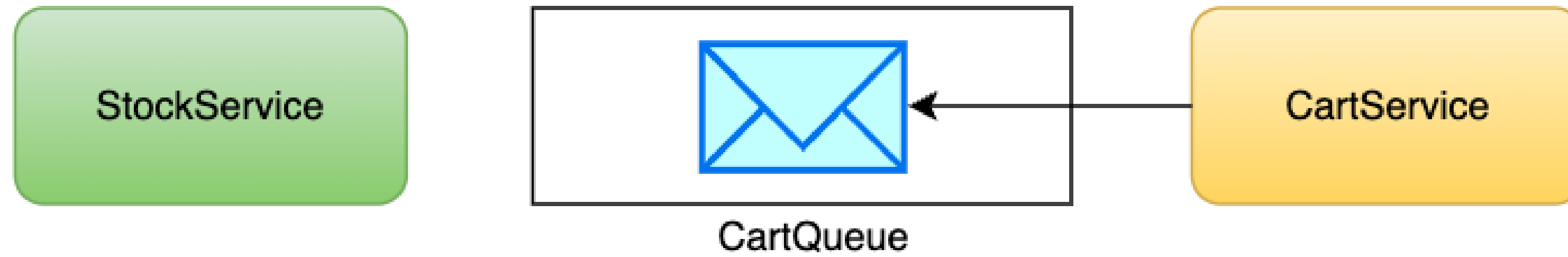
47





KOREOGRAFISANA SAGA

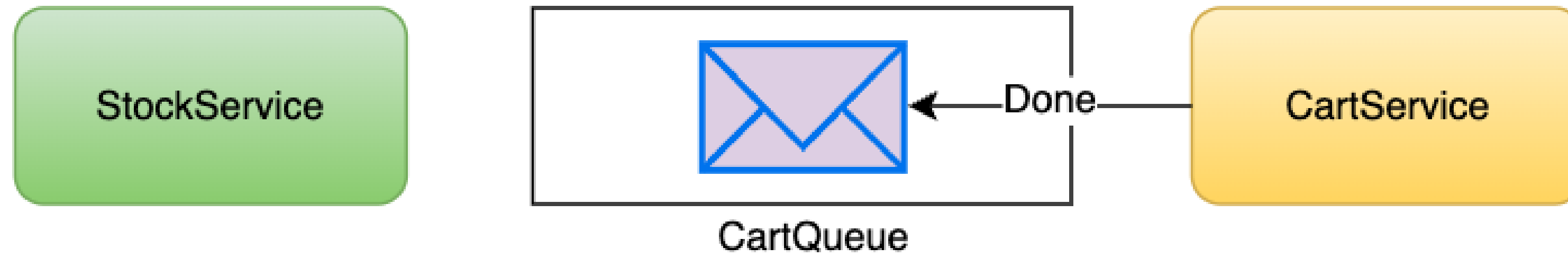
48





KOREOGRAFISANA SAGA

49



◆ ŠTA JE SAGA?

- Saga transakcije garantuju tri osobine:
 - Atomičnost - koordinator može da obezbedi da sve lokalne transakcije se izvrše ili se efekti ponište kompenzacijama
 - Konzistentnost - saga transakcije obezbeđuju *eventual consistency*
 - Izdržljivost - Saga je zasnovana na lokalnim transakcijama pa se ova osobina može obezbediti
- Saga ne garantuje izolaciju transakcija
 - Lokalna transakcija će biti vidljiva drugim transakcijama nakon što se izvrši
 - Ako su druge transakcije promenile podatke koji su uspešno poslati, operacija kompenzacije može propasti



◆ ŠTA JE SAGA?

- Saga transakcije zahtevaju da dizajn i implementacija budu u skladu sa tri svojstva:
 - Podrška za praznu kompenzaciju – učesnici u transakciji mogu dobiti redosled operacija za kompenzaciju pre obavljanja originalnih operacija zbog problema sa mrežom; u ovom slučaju je potrebna nulta naknada tj. da se ignorišu sva ažuriranja iz operacija za kompenzaciju
 - Održavanje idempotentnosti – originalne operacije i operacije kompenzacije mogu se više puta pokretati uvek sa istim ishodom
 - Sprečavanje suspenzije resursa - ako originalna operacija stigne kasnije od operacije kompenzacije zbog problema sa mrežom, originalna operacija mora biti odbačena kako ne bi došlo do suspenzije resursa

◆ PREDNOSTI

- Slaba sprega između servisa/baza
- Dobar pristup za duge transakcije
- Dobar pristup za sisteme sa velikim razlikama u performansama pisanja

◆ MANE

- *Eventual consistency* osobinu je teško zadovoljiti zbog kompleksnog dizajna i implementacije
- Izolacija nije obezbeđena što može dovesti do nekonzistentnosti podataka
- Redosled poruka je veoma bitan



2PC VS. SAGA

53

Osobina	2PC	Saga
Tip transakcije	Globalna, distribuirana	Lanac lokalnih transakcija
Atomičnost	Stroga (svi ili nijedan)	Logička (kompenzacije)
Izolacija	Da (lock-ovi)	Ne
Blokiranje	Da (blokirajući)	Ne
Otpornost na otkaze	Slaba (koordinator)	Dobra
Konzistentnost	Jaka, trenutna	Eventual consistency
Skalabilnost	Loša	Dobra
CAP ponašanje	CP	AP / BASE
Tipična upotreba	Klasične DB	Mikroservisi



◆ ŠTA SU MANE DISTRIBUIRANIH TRANSAKCIJA?

- Ako koordinator nema replike (već postoji samo jedna instanca) postaje *Single Point of Failure* (SPoF)
 - Njegov otkaz uzrokuje da drugi serveri budu blokirani lock-ovima koje drže za aktuelne transakcije
- Mnoge aplikacije su danas *stateless* i podatke čuvaju u bazi podataka, ali kada je koordinator deo sistema, u njegovom logu se čuvaju informacije o transakcijama i aplikacije time prestaju da budu *stateless*
- Pošto implementacije distribuiranih transakcija treba da budu kompatibilne sa različitim sistemima, moraju da poznaju različite sisteme da bi se mogle adekvatno koristiti (možda ne mogu da detektuju *deadlock* ili ne rade sa nekim nivoima izolacije)



REFERENCE

55

- ◆ **SILBERSCHATZ A, KORTH H, SUDARSHAN S. DATABASE SYSTEM CONCEPTS.** <https://db-book.com/>
- ◆ **BREWER E. TOWARDS ROBUST DISTRIBUTED SYSTEM.**
https://sites.cs.ucsb.edu/~rich/class/cs293b-cloud/papers/Brewer_podc_keynote_2000.pdf
- ◆ **KLEPPMANN M. DESIGNING DATA-INTENSIVE APPLICATIONS** <https://bit.ly/30gFSz3>
- ◆ **KLEPPMANN M. A CRITIQUE OF THE CAP THEOREM.** <https://arxiv.org/abs/1509.05393>
- ◆ **PRITCHETT D (EBAY). BASE: AN ACID ALTERNATIVE.** <https://queue.acm.org/detail.cfm?id=1394128>
- ◆ **GRAOVAC J. PROJEKTOVANJE BAZA PODATAKA.**
http://poincare.matf.bg.ac.rs/~jgraovac/courses/projbp/2016_2017/projbp_skripta.pdf
- ◆ **GARCIA-MOLINA H, SALEM K. SAGAS.**
<https://www.cs.cornell.edu/andru/cs711/2002fa/reading/sagas.pdf>
- ◆ **FORD N, RICHARDS M, SADALAGE P, DEGHANI Z. SOFTWARE ARCHITECTURE: THE HARD PARTS.**
<https://amzn.to/3tu3glj>

**KOJA SU VAŠA
PITANJA?**