

Raspodijeljeni sustavi

Pitanja i odgovori iz ispita, rokova i pitanja za provjeru znanja

Verzija 0.1

6.2.2013.

Sadržaj

Teorijski zadaci	2
Prvi ciklus.....	2
Drugi ciklus	11
Računski i praktični zadaci.....	18
Prvi ciklus.....	18
Drugi ciklus	27

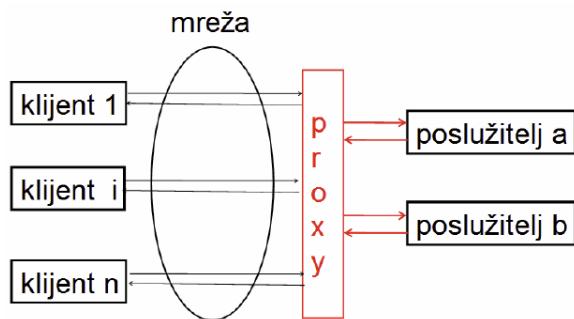
Teorijski zadaci

Prvi ciklus

Objasnite pojam konkurencijske transparentnosti raspodijeljenih sustava.

Za raspodijeljeni sustav kažemo da posjeduje konkurencijsku transparentnost ukoliko on omogućava da više različitih korisnika istodobno (konkurentno) rabi isto sredstvo, a da pri tome oni sami toga nisu niti svjesni.

Skicirajte i objasnite ulogu zastupnika poslužitelja (*proxy*). Ukoliko prepostavimo da zastupnik poslužitelja obavlja zadaću priručne pohrane (*cache*), što će se dogoditi pri ponovnom zahtjevu za istim podatkom?



Zastupnik poslužitelja (*proxy*) posreduje između klijenata i poslužitelja tako da od klijenata prikriva broj i lokaciju poslužitelja te način na koji su povezani (tj. omogućava replikacijsku transparentnost). Ukoliko je vrijeme između ta dva zahtjeva za istim podatkom relativno kratko, zastupnik poslužitelja će imati kopiju tog podatka pohranjenu u priručnom spremištu te će ju isporučiti klijentu bez ponovnog kontaktiranja poslužitelja.

Objasnite razliku između perzistentne i tranzijentne komunikacije procesa? Navedite po jedan primjer za obje vrste komunikacije.

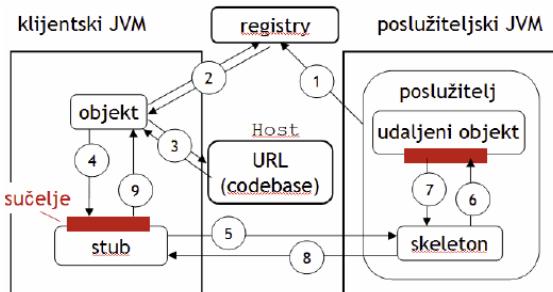
Kod **perzistentne** komunikacije procesa jamči se isporuka poruke ukoliko pošiljatelj i primatelj nisu istodobno dostupni i to tako da se poruka pohranjuje u sustavu i isporučuje primatelju kada to bude moguće. Kod **tranzijentne** komunikacije ne jamči se isporuka poruke ukoliko pošiljatelj i primatelj nisu istodobno dostupni.

Primjeri za perzistentnu komunikaciju su **objavi-preplati** i **komunikacija porukama**. Primjeri za tranzijentnu komunikaciju su socket **TCP** i **UDP**.

U tablici su prikazane aktivnosti na poslužiteljskoj strani kod socketa TCP. Da bi redoslijed postao ispravan, popunite tablicu odgovarajućim rednim brojevima aktivnosti.

1. socket()
2. bind()
3. listen()
4. accept()
5. read()
6. write()
7. close()

Skicirajte model pozivanja udaljene metode Java RMI (Remote Method Invocation), uz pretpostavku da se klasa stub učitava dinamički. Navedite redoslijed koraka u komunikaciji koji je potreban da bi klijent pozvao metodu dostupnu na poslužitelju.



Koraci u komunikaciji su sljedeći:

1. Poslužitelj definira codebase udaljenog objekta i registrira ga pod odabranim imenom.
2. Klijent (ili objekt na klijentu) od registrya traži i dobiva referencu na udaljeni objekt koristeći registrirano ime.
3. Klijent traži i dobiva klasu stuba koristeći codebase.
4. Klijent poziva metodu stuba dostupnu na klijentskom računalu.
5. Stub serijalizira parametre i šalje ih skeletonu.
6. Skeleton deserijalizira parametre i poziva metodu udaljenog objekta.
7. Udaljeni objekt vraća rezultat izvođenja metode skeletonu.
8. Skeleton serijalizira rezultat i šalje ga stubu.
9. Stub deserijalizira rezultat i dostavlja ga klijentu.
10. Stub deserijalizira rezultat i dostavlja ga klijentu.

Objasnite sličnosti i razlike u obilježjima komunikacije između dva komunikacijska modela podržana s JMS (Java Messaging Service)?

JMS podržava komunikaciju porukama i model objavi-preplati. Obje vrste komunikacije su vremenski neovisne zato što pošiljatelj i primatelj ne moraju istovremeno biti dostupni da bi se komunikacija mogla ostvariti. Kod komunikacije porukama pošiljatelj mora znati identifikator odredišta, dok je kod modela objavi-preplati komunikacija anonimna. Komunikacija je perzistentna i asinkrona u oba slučaja. Komunikacija se pokreće na načelu pull kod komunikacije porukama, a na načelu push kod modela objavi-preplati.

Usporedite grozd i splet računala s obzirom na kategorije i značajke koje su prikazane u tablici. Za svaku od ponuđenih kategorija, u stupac „Splet ili grozd“ upišite splet ukoliko ta značajka karakterizira splet računala ili grozd ukoliko ona karakterizira grozd računala.

Kategorija	Značajka	Splet ili grozd
Aplikacije	Izvođenje računalno zahtjevnih aplikacija	Grozd
	Dijeljenje raznovrsnih sredstava u globalnoj mreži	Splet
Tehnologija	Primjena vlasničkih i standardiziranih tehnologija	Grozd
	Primjena standardiziranih tehnologija	Splet
Geografska raspodijeljenost	Elementi sustava globalno raspodijeljeni	Splet
	Elementi sustava na bliskoj geografskoj udaljenosti	Grozd
Upravljanje	Središnje upravljanje sredstvima sustava	Grozd
	Više administrativnih domena za upravljanje sustavom	Splet
Povezanost	Labavo povezane strukture	Splet
	Čvrsto povezane strukture	Grozd
Prilagodljivost	Zatvorena okolina	Grozd
	Otvorena okolina	Splet

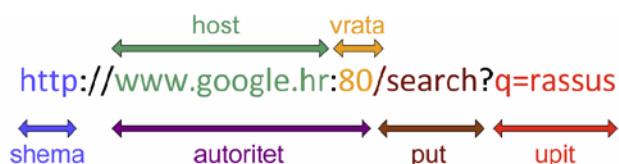
Kako se izvodi pretraživanje kod strukturiranih, a kako kod nestrukturiranih sustava sustava P2P (peer-to-peer)? Koji od ovih sustava su skalabilni i zašto?

U nestrukturiranim sustavima P2P, podatak je pohranjen na peeru koji ga kreira, a njegova kopija može biti pohranjena i na nekim drugim peerovima u mreži. Zbog toga se u ovim sustavima pretraživanje izvodi preplavljanjem mreže i slučajnim izborom (random walk). Kod strukturiranih sustava P2P, podatak je pohranjen na peeru koji je zadužen za ključ tog podatka. Pretraživanje se provodi traženjem podatka po njegovom ključu. Strukturirani sustavi su skalabilni zato što se kod njih pretraživanje odvija u $\log(n)$ koraka, gdje je n broj peerova u mreži.

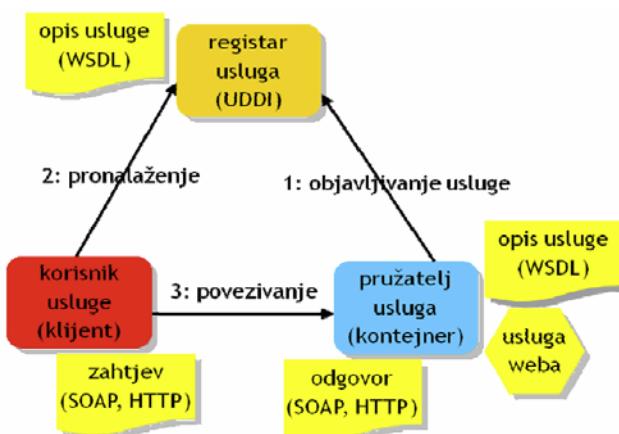
Objasnite razliku između programskog agenta i pokretnog programskog agenta. Što je to agentska platforma i koje tri vrste funkcija ona sadrži?

Za razliku od programskog agenta koji je statican, pokretni programski agent može samostalno migrirati u mreži. Agentska platforma je osnovna programska oprema koja agentu omogućuje pokretljivost. Ona se sastoji od sljedeća tri sloja funkcija: agentske, sigurnosne i komunikacijske funkcije.

Označite karakteristične dijelove URI-ja na sljedećem primjeru.



Skicirajte arhitekturu i slijed operacija za korištenje usluge weba. Ukratko objasnite redoslijed operacija pri korištenju usluga weba.



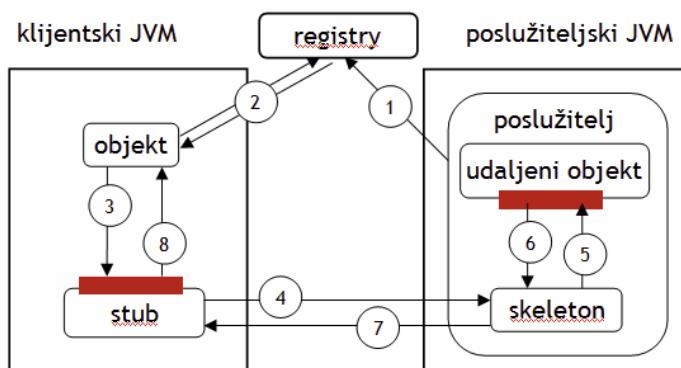
Redoslijed operacija pri korištenju usluge weba je sljedeći:

1. **Objavljivanje** usluge: pružatelj usluge vrši registraciju usluge u Registru usluga koristeći WSDL opis usluge.
2. **Pronalaženje** usluge: korisnik usluge (klijent) pronalazi odgovarajuću uslugu z registru usluga.
3. **Povezivanje**: temeljem specifikacije u WSDL opisu usluge, klijent šalje zahtjev za uslugom, te od usluge dobiva odgovor. Ovisno o usluzi, zahtjev i odgovor mogu biti upućeni korištenjem protokola SOAP ili HTTP.

Usporedite model klijent-poslužitelj s uslugom weba prema značajkama koje su navedene u sljedećoj tablici. Za svaku od ponuđenih kategorija, u stupac „KP ili WS“ upišite KP ukoliko ta značajka karakterizira model klijent-poslužitelj ili WS ukoliko ona karakterizira uslugu weba.

Kategorija	Značajka	KP ili WS
Primjena	Unutar tvrtke	KP
	Između tvrtki	WS
Programski jezici	Neovisnost o programskom jeziku	WS
	Ograničen skup programskih jezika	KP
Komunikacija između komponenti	Proceduralno	KP
	Razmjenom poruka	WS
Transportni mehanizmi	Različiti transportni mehanizmi	WS
	Jedan transportni mehanizam	KP
Povezanost	Slabo povezane strukture	WS
	Čvrsto povezane strukture	KP
Učinkovitost procesiranja	Velika	KP
	Manja	WS

Skicirajte model pozivanja udaljene metode Java RMI (Remote Method Invocation). Navedite korake u komunikaciji potrebne da bi klijent pozvao metodu dostupnu na poslužitelju, uz pretpostavku da je klasa stub već instalirana na klijentskoj strani.



1. Poslužitelj registrira udaljeni objekt pod odabranim imenom.
2. Klijent od *registrya* traži referencu na udaljeni objekt koristeći registrirano ime.
3. Klijent poziva metodu *stuba* dostupnu na klijentskom računalu.
4. *Stub* serijalizira parametre i šalje ih *skeletonu*.
5. *Skeleton* deserijalizira parametre i poziva metodu udaljenog objekta.
6. Udaljeni objekt vraća rezultat izvođenja metode *skeletonu*.
7. *Skeleton* serijalizira rezultat i šalje ga *stubu*.
8. *Stub* deserijalizira rezultat i dostavlja ga klijentu.

Navedite vrste funkcija koje sadrži agentska platforma.

Čemu služi protokol SOAP?

*** određuje format poruke, komunikaciju web usluga

Navedite i ukratko objasnite vrste usluga weba.

*** REST
-http , GET,POST,DELETE
RPC
-prijenos pohranjenih procedura
-WSDL,SOAP
PORUKE
XML, WSDL,SOAP

Skicirajte i objasnite primjer komunikacije porukama između dva procesa/objekta (primatelja i pošiljatelja). Kakva je komunikacija porukama s obzirom na vremensku ovisnost primatelja i pošiljatelja?



U komunikaciji između pošiljatelja i primatelja rep sudjeluje kao posrednik. Pošiljatelju se u načelu garantira isporuka poruke u primateljev rep, ali ne i isporuka poruke primatelju. Primatelj može pročitati poruku iz repa u bilo kojem budućem trenutku. Stoga su pošiljatelj i primatelj poruke vremenski neovisni.

U tablicama su prikazane metode na klijentskoj i poslužiteljskoj strani socketa TCP. Upišite ispravan redoslijed izvođenja metoda u tablice.

Klijent	Poslužitelj
socket()	1. socket()
connect()	2. bind()
write()	3. listen()
read()	4. accept()
close()	5. read()
*	6. write()
*	7. close()

Objasnite pojam migracijske transparentnosti raspodijeljenih sustava na primjeru raspodijeljenog sustava weba.

Skicirajte i objasnite razliku između komunikacije procesa temeljene na načelu *pull* i *push*.

Objasnite razliku u obilježjima komunikacije između modela objavi-preplati i komunikacije porukama?

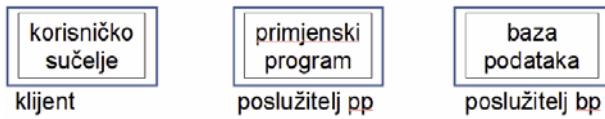
Objasnite pojam skalabilnosti raspodijeljenog sustava.

Raspodijeljeni sustav je skalabilan ukoliko posjeduje sposobnost prilagodbe povećanom broju korisnika i sredstava, njihovoj rasprostranjenosti te načinu upravljanja sustavom.

Objasnite pojam migracijske transparentnosti raspodijeljenog sustava.

Za raspodijeljeni sustav kažemo da posjeduje migracijsku transparentnost ukoliko on prikriva promjenu lokacije nekog sredstva na način da ta promjena ne utječe na način pristupa tom sredstvu.

Skicirajte trorednu arhitekturu klijent-poslužitelj te na proizvolnjom primjeru aplikacije objasnite ulogu svake razine u cjelokupnoj arhitekturi.



Primjer su aplikacije weba, gdje klijentski program koji se izvodi na klijentskom računalu nikada ne pristupa direktno bazi podataka, već posredno preko aplikacije weba. Klijentski program prikazuje korisničko sučelje i komunicira s aplikacijom weba koja obavlja cjelokupnu logiku usluge i pristupa potrebnim podacima.

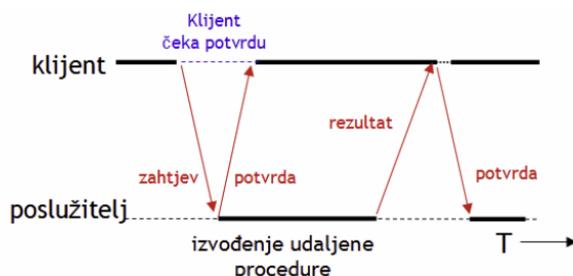
Objasnite razliku između sinkrone i asinkrone komunikacije.

Dok je kod sinkrone komunikacije pošiljatelj blokiran nakon slanja poruke sve do primitka potvrde o isporuci, kod asinkrone komunikacije pošiljatelj nije blokiran te nastavlja procesiranje odmah nakon slanja.

Navedite obilježja komunikacije socketom UDP.

Ova komunikacija se temelji na modelu klijent-poslužitelj, gdje oba moraju istovremeno biti aktivna da bi se komunikacija mogla ostvariti. Komunikacije je tranzientna i asinkrona, a može se koristiti za implementaciju komunikacije na načelu *pull* ili *push*.

Skicirajte tijek komunikacije između klijenta i poslužitelja te objasnite odgođeni sinkroni poziv udaljene procedure RPC (Remote Procedure Call).



Kod odgođenog sinkronog poziva udaljene procedure, klijent nije blokiran dok čeka rezultat izvođenja, već nastavlja s radom nakon uspješnog primitka potvrde. Kasnije mu poslužitelj šalje rezultat koristeći drugi asinkroni poziv udaljene procedure.

Objasnite opći format poruka protokola HTTP. Navedite kako glasi potpun i apsolutan URI koji identificira resurs zatražen u zahtjevu, ako prva 2 retka HTTP zahtjeva sadrže sljedeće podatke:

GET /predmet/rassus HTTP/1.1

Host: www.fer.hr

Opći format poruka protokola HTTP sastoji se od početnog retka, polja zaglavja te tijela poruke. Potpun i apsolutan URI je <http://www.fer.hr/predmet/rassus>.

Korisnik nakon ispunjavanja obrasca na Web-u odabire opciju Submit, čime pošalje podatke Web-poslužitelju na adresu www.tel.fer.hr/obrazac/accept korištenjem protokola HTTP verzije 1.1. Kojim se HTTP zahtjevom šalju podaci poslužitelju i kako je definiran prvi redak zahtjeva?

Podaci se šalju zahtjevom POST.

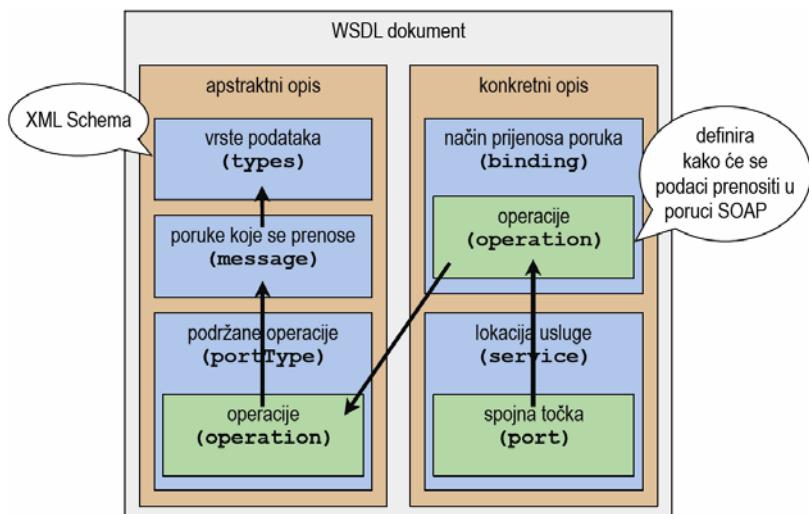
Prvi redak je definiran na sljedeći način:

POST /obrazac/accept HTTP 1.1.

Navedite dva osnovna načina rada protokola SOAP i objasnite kako se poruka SOAP šalje pomoću protokola HTTP.

Dva osnovna načina rada koje podržava protokol SOAP su poziv udaljene procedure te razmjena dokumenata i poruka. Poruka SOAP, koja je pisana jezikom XML, se sastoji od zaglavlja i tijela. Prilikom slanja poruke SOAP protokolom HTTP, i zaglavlje i tijelo poruke SOAP se nalaze u tijelu poruke HTTP.

Objasnite i skicirajte sadržaj apstraktnog i konkretnog opisa u strukturi dokumenta WSDL.



Apstraktni opis:

1. *types*: definira vrste podataka neovisne o platformi i jeziku (koristi se XML Schema),
2. *message*: definiraju ulazne i izlazne poruke koje se mogu koristiti kao parametri usluge,
3. *operation*: predstavlja jednu operaciju/metodu/proceduru koja je definirana u usluzi, a sastoji se od definicija ulaznih, izlaznih i iznimnih poruka koje se mogu razmjenjivati korištenjem ove operacije i
4. *portType*: koristi poruke (pod 2) da bi opisao sve operacije koje pruža usluga.

Konkretni opis:

1. *binding*: definira kako je konkretna implementacija povezana s operacijama u apstraktnom opisu i definira format u kojem će se poruke prenositi (protokol i elemente) i
2. *service*: definira URI gdje je usluga isporučena tj. na kojoj adresi se može pozvati usluga (taj URI je definiran u spojnoj točci).

Objasnite svojstvo slabe povezanosti usluga kod uslužno orijentirane arhitekture.

Svojstvo slabe povezanosti usluga kod uslužno orijentirane arhitekture (SOA – Service Oriented Architecture) odnosi se na dizajn programske izvedbe usluga. U sustavu SOA, usluge trebaju biti izvedene tako da promjena u jednoj usluzi ne zahtijeva promjenu neke druge usluge. Pri tome, svaka usluga može i dalje nesmetano koristiti neku drugu uslugu. Npr. algoritam koji se koristi u usluzi se može promijeniti bez znanja drugih usluga i druge usluge ju mogu nesmetano koristiti. Bitno je da se sučelja opisana WSDL-om ne promijene.

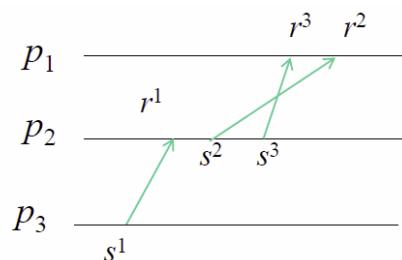
Objasnite za koje je od sljedeća tri svojstva raspodijeljenih sustava značajna komunikacijska složenost algoritama: a) replikacijska transparentnost b) skalabilnost c) otvorenost.

Komunikacijska složenost algoritama je važna za **skalabilnost** raspodijeljenog sustava jer na temelju komunikacijske složenosti možemo zaključiti kako raste generirani promet raspodijeljenog sustava s rastom tog sustava. **Primjer:** komunikacija grupe procesa.

Objasnite model komunikacijskog kanala koji se temelji na uzročnoj slijednosti.

Uzročna slijednost (*causal ordering*) osigurava da uzročno povezani događaji slanja dviju poruka istom primatelju rezultiraju primanjem u slijedu kojim su poslani.

Objasnite zašto za sljedeći primjer vrijedi CO ili vrijedi non-CO?



Za primjer vrijedi non-CO zato što proces p_1 prima poruke od p_2 drugačijim redoslijedom od redoslijeda slanja, a pri tome je slanje poruke 3 uvjetovano slanjem poruke 2.

Navedite i objasnite operacije koje implementira programska infrastruktura dijeljenog podatkovnog prostora.

- **write(t)** – dodaj tuple t u raspodijeljeni podatkovni prostor
- **read (s) -> t** – vraća tuple t koji odgovara predlošku s
- **take (s) -> t** – vraća tuple t koji odgovara predlošku s i briše ga iz podatkovnog prostora

Prepostavite da se sjedište weba sastoji od 2 poslužitelja priključena na Internet preko posrednika (proxy). Navedite i objasnite svojstva ovog raspodijeljenog sustava.

Ovaj raspodijeljeni sustav karakteriziraju:

- 1) replikacijska transparentnost – zato što korisnik nije svjestan koji poslužitelj ga je zapravo poslužio,
- 2) otpornost na kvarove – jer se kvar jednog poslužitelja može prikriti od korisnika i
- 3) skalabilnost – zato što ovaj sustav posjeduje sposobnost prilagodbe povedanom broju korisnika.

Ovaj sustav podržava i lokacijsku i migracijsku transparentnost (putem sustava DNS).

Objasnite razliku između web-aplikacija temeljenih na CGI (Common Gateway Interface) i poslužiteljskim skriptama.

CGI (*Common Gateway Interface*) je jednostavno sučelje za pokretanje eksternih programa iz web-poslužitelja na platformski i programski neovisan način. Kod svakog zahtjeva se pokreće novi proces, a podaci između poslužitelja i procesa šalju se preko varijabli okoline i tokova podataka. Nakon svake obrade proces se gasi. Nedostatak CGI-a je što se kod svakog zahtjeva pokreće novi proces i nakon obrade gasi što je zahtjevno za resurse (procesorsko vrijeme i memorija) pa kod velikog broja zahtjeva na poslužitelju to znatno utječe na performanse.

Poslužiteljske skripte (engl. *server side scripts*), dinamički generiraju HTML-dokumente poput CGI-ja, ali je razlika u tome što se za svaki zahtjev ne pokreće novi proces i na taj način se štede resursi.

***** Objasnite tri načina rada posredničkog (*proxy*) poslužitelja**

Proxy poslužitelj posreduje između klijenta i poslužitelja. Svrha – filtriranje informacija, privremeno spremište, anonimiziranje klijenta, bilježenje korištenja mreže, pregledavanje sadržaja.

- **Forward proxy** (lokalna mreža klijenta) – smanjenje korištenja mreže, filtriranje i kontrola sadržaja
- **Open proxy** (javna mreža) – anonimiziranje klijenta
- **Reverse proxy** (lokalna mreža poslužitelja) – enkripcija, uravnoteženje opterećenja, sigurnosne postavke

***** Objasnite svojstva sigurnosti, idempotentnosti i *cacheable* za metode protokola HTTP i označite ih u tablici.**

- Sigurna (safe) – bez posljedica za podatke
- Indempotentna – može se izvršavati više puta
- Cacheable – odgovor se može privremeno spremiti

	<i>sigurnost</i>	<i>idempotentnost</i>	<i>cacheable</i>
GET	DA	DA	DA
PUT		DA	
DELETE		DA	
HEAD	DA	DA	
POST			

Drugi ciklus

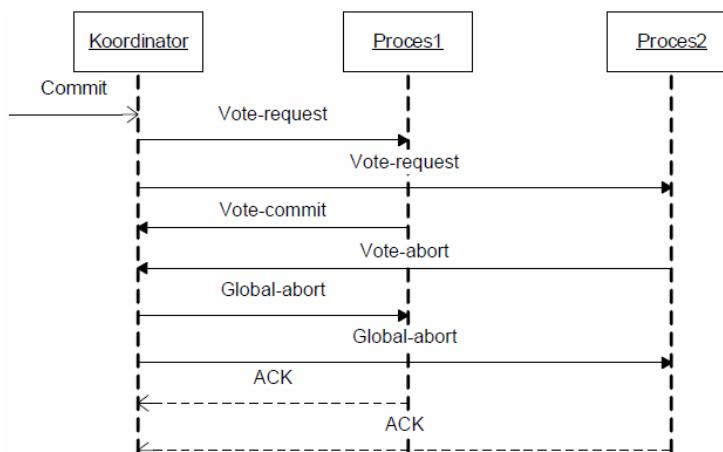
Koje su vrste osnovnih sredstava sadržane u spletu računala? Navedite barem jednu uslugu potrebnu za dijeljene sredstava u spletu računala.

Vrste osnovnih sredstava u spletu računala su:

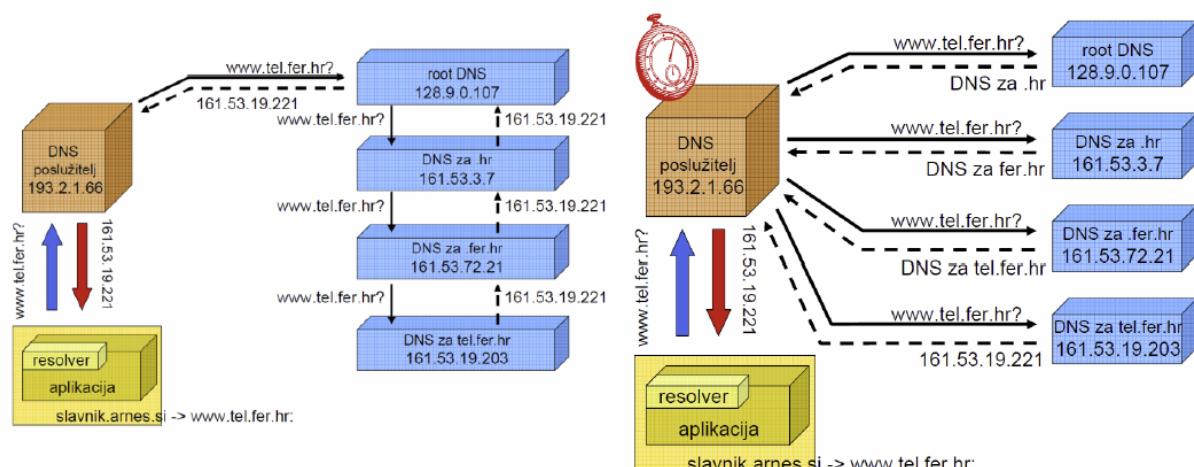
- računalna sredstva (computational resources),
- spremnički prostor (storage systems),
- katalozi sredstava (resource catalogues),
- mrežna sredstva (network resources) i
- osjetila i aktuatori (sensors and actuators).

Jedna od usluga za dijeljenje sredstava je imenička usluga.

Slika prikazuje koordinatora i dva procesa koji koriste protokol **two-phase commit** za raspodijeljeno izvršavanje operacije commit koju prima koordinator. Pretpostavite da proces 1 prihvata izvršavanje navedene operacije dok ju proces 2 odbija izvršiti te da nema gubitaka u komunikaciji pri prijenosu poruka. Nadopunite slijedni dijagram prikazan slikom.



Nadopunite skicu te objasnite razliku između rekurzivnog i iterativnog načina rada DNS poslužitelja. Pretpostavite da je u oba slučaja priručno spremište (cache) prazno.



Raspodijeljeni sustavi – pitanja i odgovori iz ispita, rokova i pitanja za provjeru znanja

Kod **rekurzivnog načina rada**, poslužitelj vrada ili grešku ili odgovor, ali nikad ne vrada pokazivače na druge DNS poslužitelje.

Kod **iterativnog načina rada** poslužitelj odgovara samo na osnovu informacije koju posjeduje. To znači da vrada grešku, odgovor ili pokazivač na drugi, „bliži“ poslužitelj.

Skicirajte i objasnite slojevitu arhitekturu spletka računala.

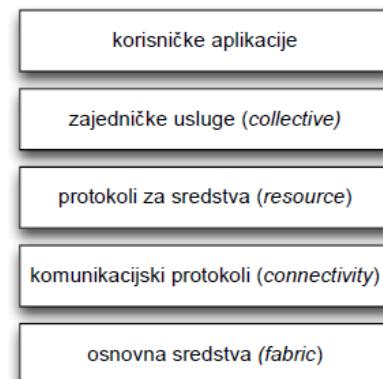
Sloj osnovnih sredstava upravlja osnovnim funkcionalnostima spletka računala. On implementira lokalne operacije koje su specifične svakom sredstvu po vrsti i implementaciji te omogućuje korištenje tih operacija na višim slojevima.

Sloj komunikacijskih protokola definira protokole komunikacije i autentifikacije koji su potrebni za transakcije u spletu.

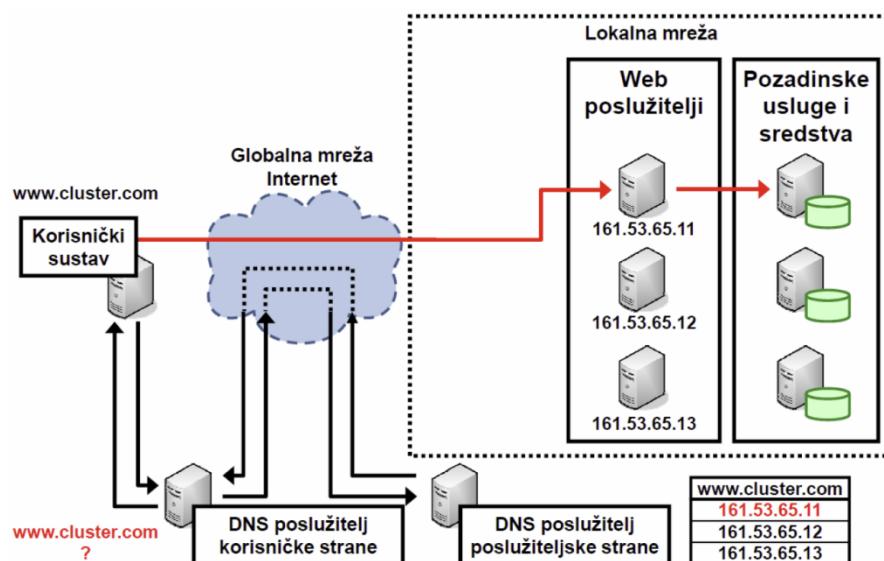
Sloj protokola za sredstva omogućuje korisniku interakciju s udaljenim resursima i uslugama. On definira protokole za sigurno pregovaranje, pokretanje, praćenje, kontrolu, obračun (engl. accounting) i naplatu dijeljenih operacija i individualnih resursa.

Sloj zajedničkih usluga definira protokole i usluge koji su zaduženi za upravljanje grupom sredstava, a ne za pojedino sredstvo.

Na vrhu se nalazi **sloj korisničkih aplikacija**.

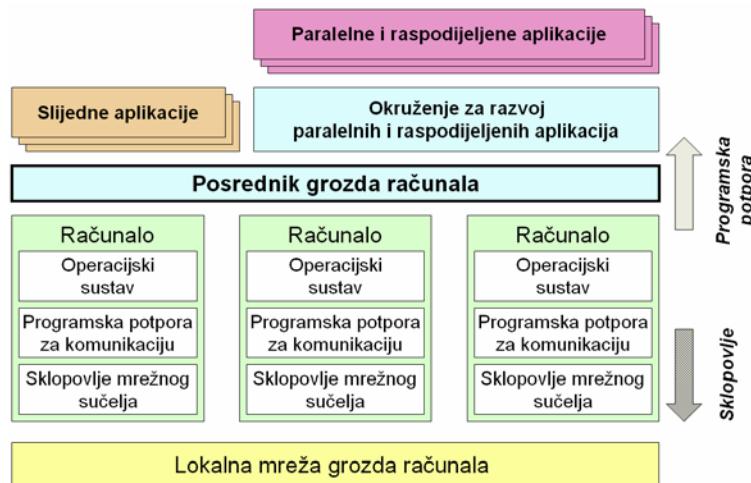


Nadopunite skicu i objasnite primjer ostvarivanja razmјernog rasta sustava primjenom metode DNS poslužitelja.



Kod ostvarivanja razmјernog rasta sustava primjenom metode DNS poslužitelja, korisnički sustav upućuje zahtjev korisničkom DNS poslužitelju. On taj zahtjev proslijedi poslužiteljskom DNS poslužitelju. Poslužiteljski DNS poslužitelj odabire jedno od postojećih odredišnih računala te njegovu IP adresu šalje kao odgovor. Korisnički sustav prima odgovor s adresom odredišnog računala i upućuje zahtjev dodijeljenom poslužitelju.

Grozovi računala zasnovani su na višeslojnoj organizaciji. Skicirajte slojevitu strukturu grozda računala i objasnite slojeve koji se izvode programski.



Sve računala u grozdu su međusobno neovisna, a svako posjeduje operacijski sustav s programskom potporom za komunikaciju. Za izvođenje aplikacija, osim samih računala, potreban je i posrednički sustav grozda računala čiju osnovu čine podsistemi za upravljanje poslovima, za nadgledanje rada i dijeljenje spremničkog prostora te razvojno okruženje. Ukoliko su aplikacije koje se izvode paralelne i raspodijeljene, potrebno je koristiti i odgovarajuće okružje za razvoj takvih aplikacija.

Objasnite što je replika podatka, a što je nekonistentnost replike podatka.

Replika podatka je jedna kopija podatkovnog objekta u raspodijeljenoj okolini. Nekonistentnost replika podataka se javlja kada dvije ili više replika u raspodijeljenoj okolini u nekom trenutku u vremenu se nalaze u različitim stanjima.

Navedite i opišite značajke tri osnovna razreda replika podataka u raspodijeljenim sustavima.

U raspodijeljenim sustavima koriste se sljedeća tri razreda replika:

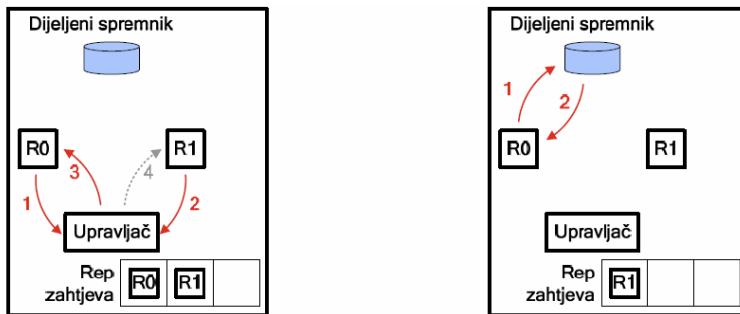
1. Trajne replike: Osnovni primjeri podataka koji su trajno postavljeni na poslužiteljima. Njihove postavke su statičke i upravljane od strane vlasnika podataka. Nad njima se najčešće ostvaruju operacije čitana podataka i poslužuju primjenom grozdova poslužitelja i replika poslužitelja.

2. Poslužiteljske replike: Poslužitelji podataka stvaraju nove replike trajnih replika i raspoređuju ih na dostupne poslužitelje u mreži. Odabir i raspoređivanje replika ostvaruje se u stvarnom vremenu tijekom rada poslužitelja.

3. Korisničke replike: Postupak repliciranja iniciran je odstrane korisnika putem korisničkih programa. Korisnički programi u lokalne spremnike spremaju pribavljene podatke. Potrebno je održavati konzistentnost lokalnog spremnika s poslužitelje od kojeg su podatci dohvaćeni.

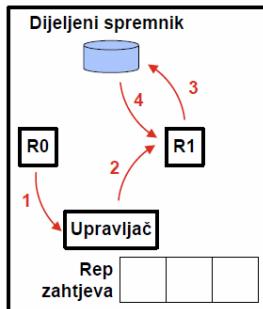
Opišite postupak međusobnog isključivanja dvaju procesa (R0 i R1) primjenom središnjeg upravljača s repom čekanja tako da nacrtate redoslijed operacija i objasnite ih. Nakon zauzimanja dijeljenog spremnika, proces provodi jednu operaciju čitanja ili pisanja nad dijeljenim spremnikom.

Raspodijeljeni sustavi – pitanja i odgovori iz ispita, rokova i pitanja za provjeru znanja

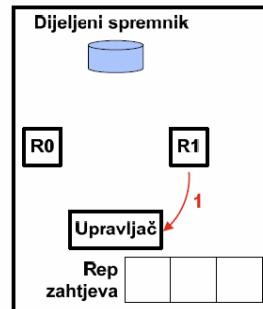


- | | |
|---|--|
| 1 – RO šalje zahtjev za zauzimanje sredstva, zahtjev se sprema u rep | 1 – RO priprema zahtjev za zauzimanje sredstva |
| 2 – R1 šalje zahtjev za zauzimanje sredstva, zahtjev se stavlja u rep | 2 – RO priprema zahtjev za zauzimanje sredstva |
| 3 – Kako je zahtjev od R0 stigao prije, upravljač šalje potvrdu R0 i uklanja njegov zahtjev iz repa | |

- 1 – R0 provodi operaciju pisanja
 - 2 – R0 prima potvrdu

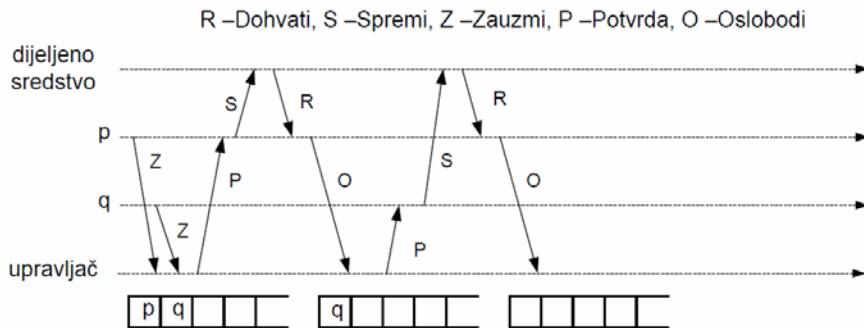


- 1 – RO šalje poruku Upravljaču i otpušta pristup
 - 2 – Upravljač šalje poruku dojave R1 te mu dodjeljuje pristup dijeljenom spremniku. Iz repa zahtjeva uklanja se zahtjev od R1
 - 3 – R1 provodi operaciju pisanja
 - 4 – R1 prima potvrdu



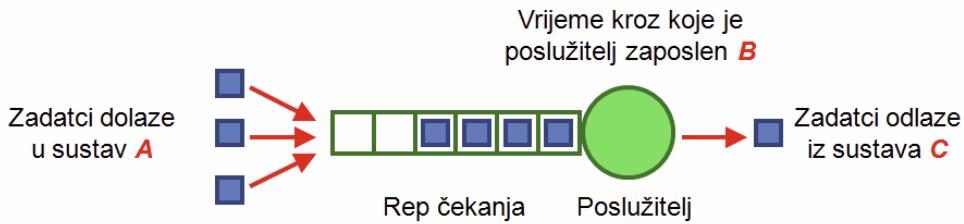
- ## 1 – R1 šalje poruku Upravljaču i otpušta pristup

Noviji dijagram ($R0=p$, $R1=q$):



- Proces p šalje zahtjev za zauzimanje sredstva, zahtjev se sprema u rep
 - Proces q šalje zahtjev za zauzimanje sredstva, zahtjev se stavlja u rep
 - Kako je zahtjev od R0 stigao prije, upravljač šalje potvrdu R0 i uklanja njegov zahtjev iz repa
 - Proces p provodi operaciju pisanja
 - Proces p prima potvrdu
 - Proces p šalje poruku Upravljaču i otpušta pristup
 - Upravljač šalje poruku dojave procesu q te mu dodjeljuje pristup dijeljenom spremniku. Iz repa zahtjeva uklanja se zahtjev od procesa p
 - Proces q provodi operaciju pisanja
 - Proces q prima potvrdu
 - Proces q šalje poruku Upravljaču i otpušta pristup

Prikažite elemente osnovnog modela repa čekanja. Koje su osnovne veličine, a koje izvedene u modelu repa čekanja? Kako je definirano stacionarno stanje sustava?



Osnovne veličine modela su vrijeme promatranja (T), broj dolazaka (A), broj odlazaka (C) i vrijeme zaposlenosti poslužitelja (B).

Izvedene veličine modela su ulazni ritam ($L = A/T$), izlazni ritam ($X = C/T$), srednje vrijeme posluživanja ($S = B/C$) i zaposlenost poslužitelja ($U = B/T$).

U stacionarnom stanju sustava je $X = L$.

Objasnite razliku između ispada sustava i neispravnosti u sustavu.

Ispad sustava je stanje sustava koje se detektira kroz nemogućnost korištenja jedne ili više njegovih usluga. Posljedica je neispravnosti te ukazuje na nju. Neispravnost je nedostatak u programskom kodu, oblikovanju sustava ili komunikacijskom kanalu koji može uzrokovati ispad sustava.

Prepostavite da se u grupi procesa s ispadima poštuje načelo virtualne sinkronosti. Proces p šalje poruku m grupi procesa G. Tijekom isporuke poruke m dogodi se ispad procesa p. Što se događa isporukom poruke m?

Isporuka se prekida, procesi ignoriraju primljenu poruku.

Prepostavite da procesi P1 i P2 šalju poruke koje se isporučuju procesima P3 i P4 prema tablici. Navedite koju vrstu pouzdane komunikacije podržava grupa procesa P1, P2, P3 i P4?

Proces P1	Proces P2	Proces P3	Proces P4
šalje m11	šalje m21	prima m11	prima m11
šalje m12	šalje m22	prima m21	prima m12
šalje m13		prima m22	prima m21
		prima m12	prima m13
		prima m13	prima m22

Ova grupa procesa podržava **pouzdani FIFO multicast**.

Navedite obilježja pouzdane komunikacije grupe procesa pod nazivom atomic multicast.

Atomic multicast je pouzdani virtualno sinkroni multicast s potpuno uređenim slikom poruka.

Objasnite razliku protokola *three-phase commit* u odnosu na *two-phase commit*.

Three-phase commit je sličan protokolu *two-phase commit* osim što koordinator nakon odluke za izvođenje operacije šalje poruku PREPARE_COMMIT na koju procesi odgovaraju s READY_COMMIT. Nakon što primi poruku READY_COMMIT od svih procesa, koordinator šalje GLOBAL_COMMIT.

U spletu računala se koriste 3 komunikacijska sloja: primjenski sloj, sloj prividne mreže i transportni sloj. Ukratko opišite primjenski sloj.

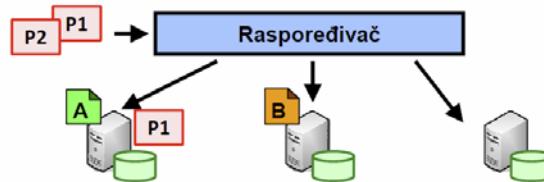
Primjenski sloj je sloj za upravljanje sredstvima u sletu računala koji ostvaruje funkcionalnosti za upravljanje prijenosom velike količine podataka, korištenje udaljenih sredstava na način kao da su lokalno dostupna te izgradnju imenika sredstava u prividnoj mreži spletu računala.

Navedite i opišite osnovne elemente za uspostavu sigurnosti u spletovima računala.

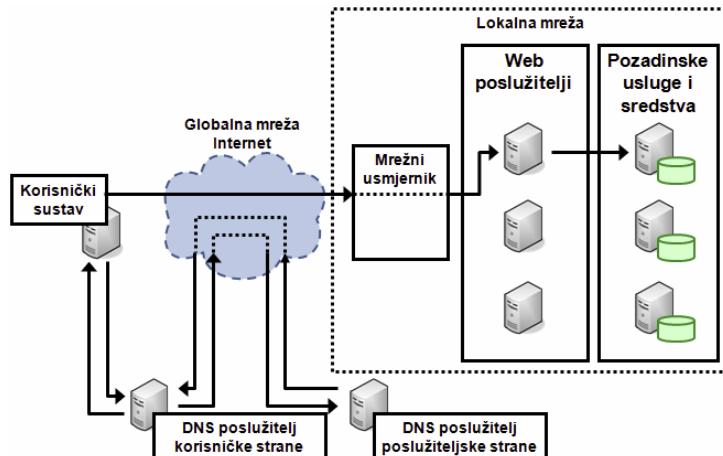
Autentikacija je provjera identiteta korisnika i sredstava u spletu računala. **Autorizacija** je provjera i uspostava kontrole pristupa sredstvima u sustavu spletu računala. Enkripcija podataka je primjena strukture i sadržaja podataka s ciljem zaštite informacija koje se razmjenjuju između sudionika komunikacije.

Na primjeru opišite značajke raspoređivanja zasnovanog na korištenju prostorne lokalnosti.

Kod prostorne lokalnosti, poslovi se raspoređuju na čvorove koji sadrže podatke potrebne za izvođenje posla. Drugim riječima, poslovi se približavaju podacima.

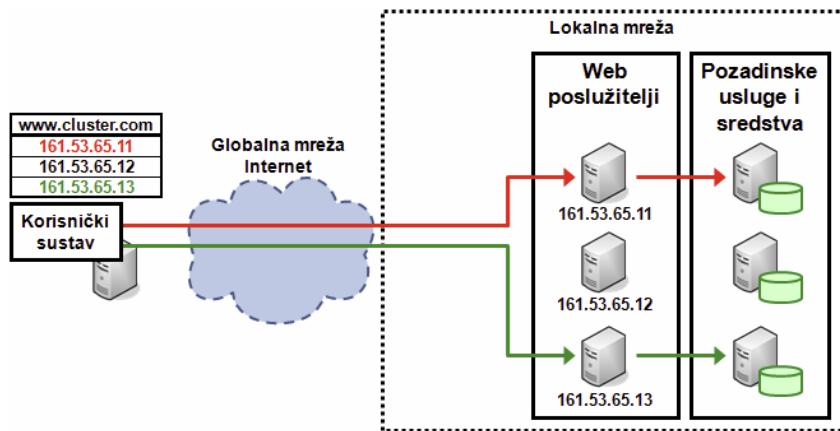


Prikažite i opišite elemente modela grozda računala.



- Korisnički sustav je aplikacija kojom korisnik ostvaruje pristup i koristi sredstva i usluge na grozdu računala.
- DNS poslužitelj korisničke strane je poslužitelj pomoću kojeg korisnički sustav razlučuje adrese udaljenih računala na Internetu.
- DNS poslužitelj poslužiteljske strane je poslužitelj koji razlučuje adresu poslužitelja u lokalnoj mreži.
- Mrežni usmjernik je uređaj koji prihvata, analizira i usmjerava pristigne zahtjeve.
- Web poslužitelji i pozadinska sredstva i usluge su osnovni elementi grozda računala.

Prikažite primjer ostvarivanja razmjernog rasta sustava primjenom metode *prosljeđivanje zahtjeva na strani korisnika*.



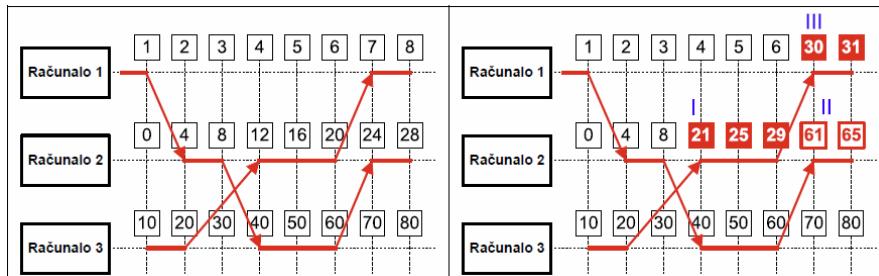
Objasnite načine na koje se indeks dokumenata može podijeliti između čvorova u raspodijeljenim tražilicama.

Indeks dokumenata se dijeli prema dokumentima ili riječima. U prvom slučaju je svaki čvor zadužen za podskup dokumenata iz kolekcije za koji izgrađuje invertirani indeks, a pri rješavanju upita kontaktira se svaki čvor koji generira odgovor na temelju vlastite kolekcije. U drugom slučaju je svaki čvor zadužen za dio rječnika kompletne kolekcije. Procesira je upita je jednostavnije nego u prethodnom slučaju, no problem je kreiranje i održavanje ovakvog raspodijeljenog indeksa.

Računski i praktični zadaci

Prvi ciklus

Za slijed razmijenjenih poruka između tri računala prikazan slikom, uspostavite globalni tijek vremena primjenom skalarnih oznaka logičkog vremena. Navedite i objasnite trenutke u kojima se ostvaruje korekcija lokalnih satnih mehanizama.

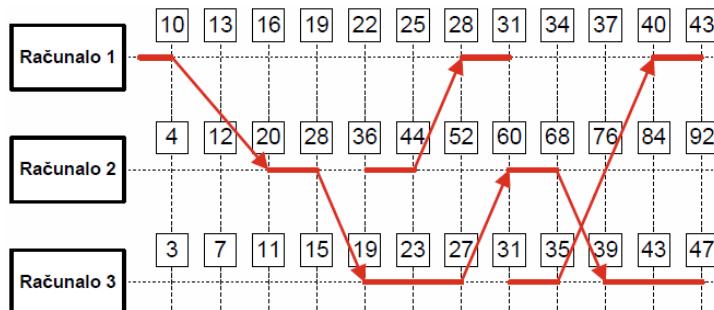
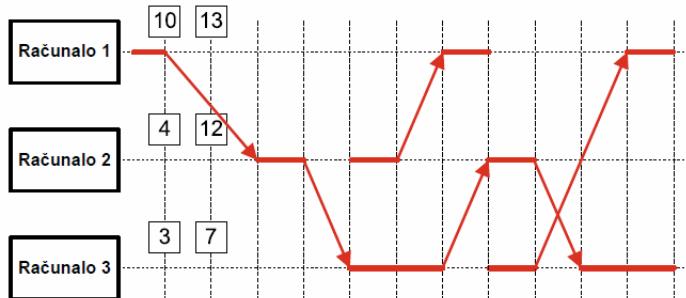


Trenutak I: Računalo 2 prima poruku od računala 3 s oznakom vremena $T_p=20$ koja je veća od lokalne oznake vremena $T_L= 12$. Lokalni sat se pomiče na vrijednost $T_p+1=21$.

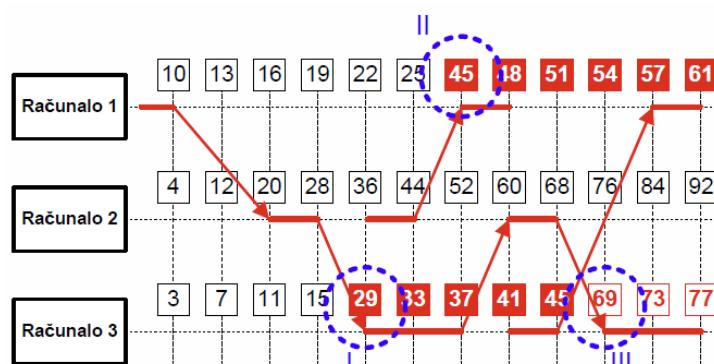
Trenutak II: Računalo 2 prima poruku od računala 3 s oznakom vremena $T_p=60$ koja je veća od lokalne oznake vremena $T_L= 33$. Lokalni sat se pomiče na vrijednost $T_p+1=61$.

Trenutak III: Računalo 1 prima poruku od računala 2 s oznakom vremena $T_p=29$ koja je veća od lokalne oznake vremena $T_L= 7$. Lokalni sat se pomiče na vrijednost $T_p+1=30$.

Za slijed razmijenjenih poruka između tri računala prikazan slikom, uspostavite globalni tijek vremena primjenom skalarnih oznaka logičkog vremena. Navedite i objasnite trenutke u kojima se ostvaruje korekcija lokalnih satnih mehanizama.



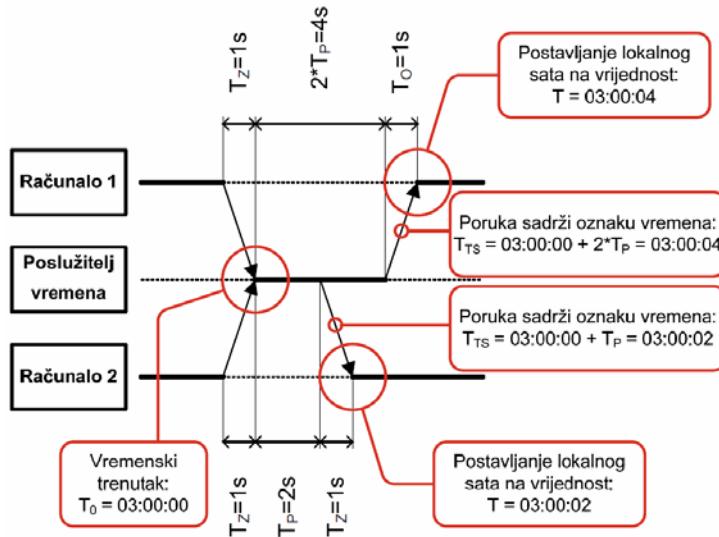
Trenutak I: Računalo 3 prima poruku od računala 2 s oznakom vremena $T_P=28$ koja je veća od lokalne oznake vremena $T_L=19$. Lokalni sat se pomiče na vrijednost $T_P+1=29$.



Trenutak II: Računalo 1 prima poruku od računala 2 s oznakom vremena $T_P=44$ koja je veća od lokalne oznake vremena $T_L=28$. Lokalni sat se pomiče na vrijednost $T_P+1=45$.

Trenutak III: Računalo 3 prima poruku od računala 2 s oznakom vremena $T_P=68$ koja je veća od lokalne oznake vremena $T_L=49$. Lokalni sat se pomiče na vrijednost $T_P+1=69$.

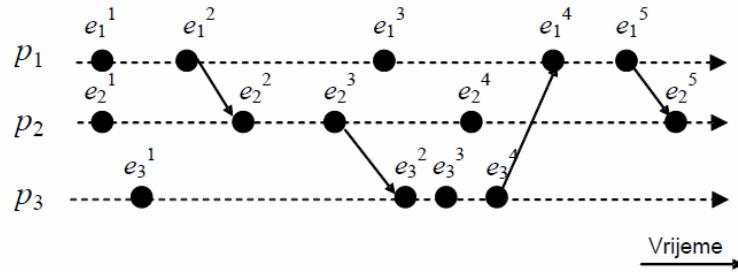
U raspodijeljenoj okolini nalazi se poslužitelj vremena i dva računala koja koriste poslužitelj primjenom algoritma Christian kako bi međusobno uskladili vrijednosti satnih mehanizama. Poruke zahtjeva i odgovora od svakog računala do poslužitelja putuju $T_Z = T_0 = 1\text{s}$ dok je vrijeme obrade poruke $T_P = 2\text{s}$. U slučaju da poslužitelj istodobno primi dva ili više zahtjeva, poslužitelj slijedno obrađuje primljene zahtjeve. U trenutku 3:00:00 poslužitelj prima poruku zahtjeva od računala 2 te nakon 0 sekundi prima poruku zahtjeva od računala 1. Ukoliko pretpostavimo da svi satovi imaju isti takt, kolika je razlika vremena između sva tri računala u raspodijeljenoj okolini nakon završetka izvođenja algoritma za oba računala? Dopunite vremenski dijagram karakterističnim točkama.



Nakon što računalo 2 prima poruku dogovora s oznakom vremena 03:00:02, postavlja lokalni satni mehanizam na vrijednost 03:00:02. Nakon što računalo 1 prima poruku dogovora s oznakom vremena 03:00:04, postavlja lokalni satni mehanizam na vrijednost 03:00:04. Nakon što računalo 1 prima poruku odgovora i ostvari uskladišvanje lokalnog sata, satni mehanizmi računala imaju sljedeće vrijednosti i odstupanja u odnosu na ostala računala:

$$\begin{aligned}
 R_1: & \quad 03:00:04, \quad \Delta R_{12}: \quad 00:00:00, \quad \Delta R_{1P}: \quad + 00:00:01 \\
 R_P: & \quad 03:00:05, \quad \Delta R_{P1}: \quad - 00:00:01, \quad \Delta R_{P2}: \quad - 00:00:01 \\
 R_2: & \quad 03:00:04, \quad \Delta R_{21}: \quad 00:00:00, \quad \Delta R_{2P}: \quad + 00:00:01
 \end{aligned}$$

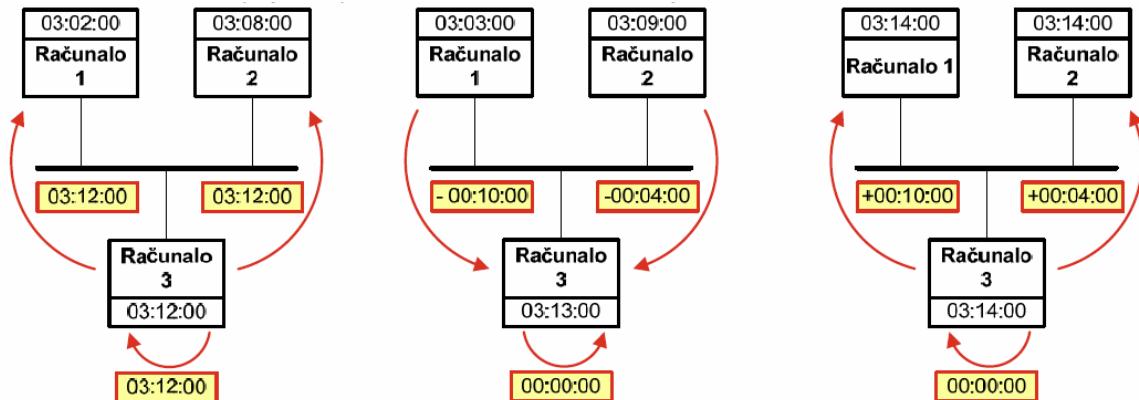
Na temelju primjera procesa sa slike objasnite jesu li sljedeći parovi događaja uzročno povezani ili nisu? a) e_1^3 i e_2^2 b) e_2^2 i e_1^5



a) Događaji e_1^3 i e_2^2 su neovisni zato što nisu slijedni događaji na istom procesu, između njih ne postoji slanje i primanje poruke te između njih ne postoji tranzitivna uzročnost.

b) Između događaja e_2^2 i e_1^5 postoji uzročna povezanost preko tranzitivne uzročnosti $e_2^2 \rightarrow e_2^3 \wedge e_2^3 \rightarrow e_3^2 \wedge e_3^2 \rightarrow e_3^3 \wedge e_3^3 \rightarrow e_3^4 \wedge e_3^4 \rightarrow e_1^4 \wedge e_1^4 \rightarrow e_1^5$.

Prikažite i objasnite korake algoritma Berkeley za uskladivanje satnih mehanizama tri računala u raspodijeljenoj okolini. Računala imaju sljedeće vrijednosti satova T1=03:02:00, T2=03:08:00 i T3=03:12:00. Upravitelj je treće računalo. Pretpostavite da prijenos poruke između 2 računala traje 1 minutu.

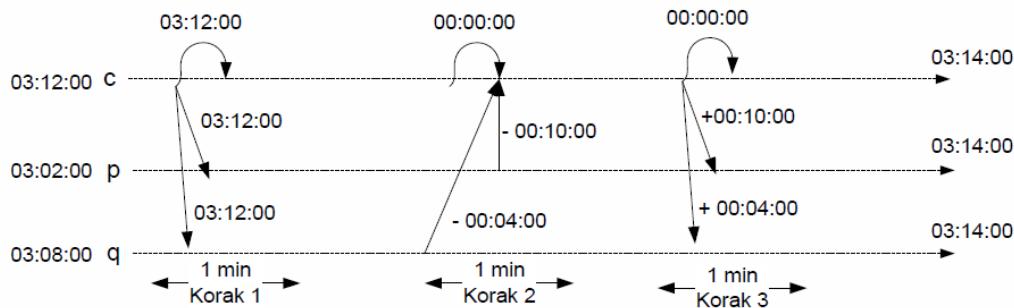


Upravitelj šalje poruku s trenutnim vremenom svim računalima.

Poslane poruke putuju 1 minuto i nakon primitka poruka, računala odgovaraju s porukom koja sadrži razliku lokalnog vremena u odnosu na primljeno vrijeme.

Nakon primitka poruka odgovora, upravitelj šalje poruke zahtjeva koje sadrže vremenski pomak za svako računalo. Poruke zahtjeva putuju 1 minuto te nakon primitka poruke zahtjeva, svako računalo uskladuje lokalni satni mehanizam.

Noviji dijagram za isti zadatak:



Pet procesa postavljenih na različita računala u raspodijeljenoj okolini ostvaruje međusobno isključivanje primjenom prstena. Vrijeme prijenosa poruke zahtjeva i odgovora pri pristupu dijeljenom sredstvu jednako je 3 ms, vrijeme obrade poruke zahtjeva na sredstvu je 5 ms, vrijeme prijenosa tokena između dva susjedna procesa u prstenu je 2 ms. Kada primi token, proces može maksimalno jednom ostvariti pristup dijeljenom sredstvu prije nego što proslijedi token idućem susjedu. Prikažite strukturu prstena i naznačite navedena vremena na slici. Koje je minimalno, a koje maksimalno vrijeme čekanja bilo kojeg procesa u prstenu za pristup dijeljenom sredstvu.

Minimalno vrijeme čekanja na pristup

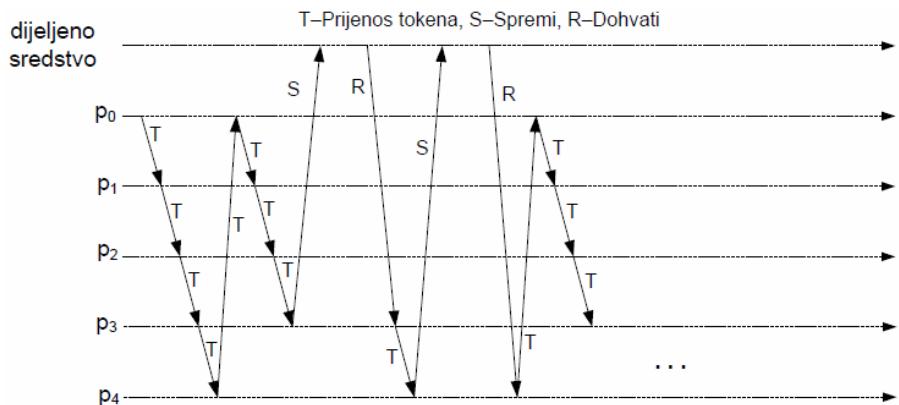
U najboljem slučaju, proces koji želi ostvariti pristup čeka $T=0$ sekundi. Naime, taj slučaj nastupa kada proces uđe u stanje u kojem želi ostvariti pristup sredstvu netom prije nego što je primio token.

Maksimalno vrijeme čekanja na pristup

U najgorem slučaju, proces ulazi u stanje u kojem želi ostvariti pristup sredstvu netom nakon što je proslijedio token svojem susjedu. U tom slučaju, proces mora čekati da svi ostali procesi prime token i ostvare pristup dijeljenom sredstvu. Maksimalno vrijeme čekanja u tom slučaju iznosi

$$T = 5 * T_T + 4 * (T_Z + T_O + T_P) = 10 + 44 = 54 \text{ ms.}$$

Noviji dijagram za isti zadatak:

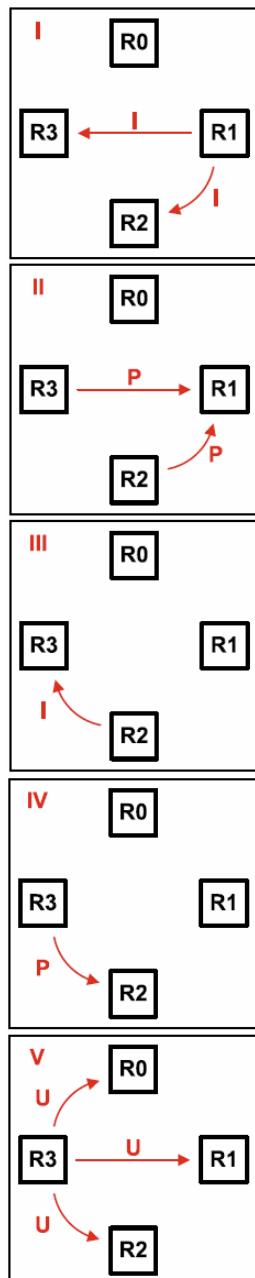


Min. vrijeme - U najboljem slučaju, proces koji želi ostvariti pristup čeka $T=0$ sekundi. Naime, taj slučaj nastupa kada proces uđe u stanje u kojem želi ostvariti pristup sredstvu netom prije nego što je primio token.

Max. vrijeme - U najgorem slučaju, proces ulazi u stanje u kojem želi ostvariti pristup sredstvu netom nakon što je proslijedio token svojem susjedu. U tom slučaju, proces mora čekati da svi ostali procesi prime token i ostvare pristup dijeljenom sredstvu. Maksimalno vrijeme čekanja u tom slučaju iznosi

$$T = 5 * T_T + 4 * (T_Z + T_O + T_P) = 10 + 44 = 54 \text{ ms.}$$

Prikažite postupak određivanja upravitelja između četiri računala (R_0 , R_1 , R_2 , R_3) u raspodijeljenoj okolini. Postupak odlučivanja započinje računalom R_1 .



Postupak izbora započinje računalom R_1 slanjem poruke *izbor* (I) svim računalima većeg rednog broja od računala R_1 . Poruka izbora poslana je računalima R_2 i R_3 .

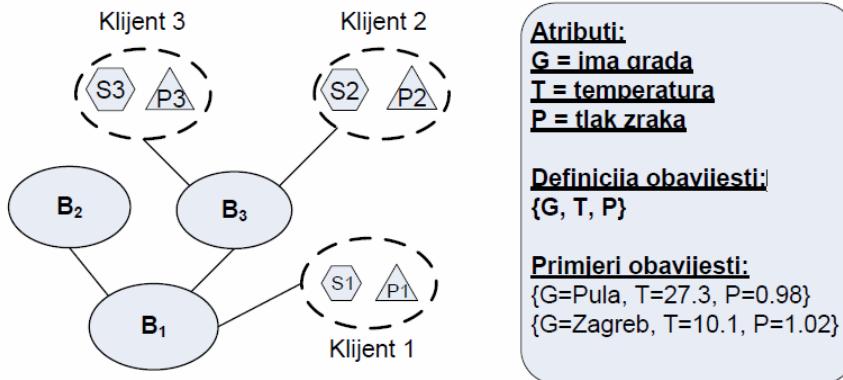
Računala R_2 i R_3 primaju poruku izbora. Obzirom da su oba računala aktivna, oba računala šalju poruku *potvrda* (P) računalu R_1 .

Obzirom da je računalo R_3 računalo s najvećim rednim brojem, računalo R_3 niti jednom računalu ne šalje poruku *izbor*. Međutim, računalo R_1 nije računalo s najvećim rednim brojem te stoga šalje poruku *izbor* (I) računalu R_3 .

Računalo R_3 prima poruku *izbor* (I) od računala R_2 i šalje poruku *potvrda* (P) računalu R_2 .

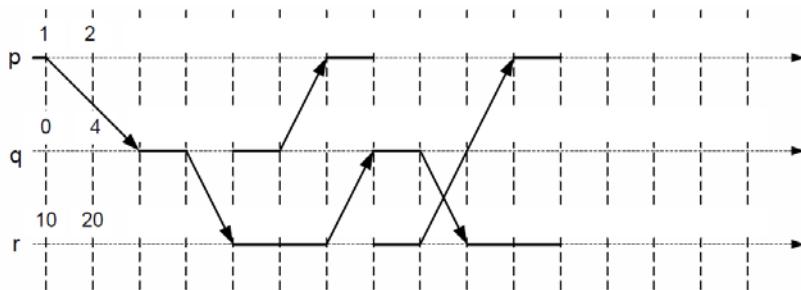
Obzirom da je računalo R_3 računalo s najvećim rednim brojem, računalo R_3 postaje novi upravitelj te šalje poruku *upravitelj* (U) svim računalima (R_0 , R_1 , R_2).

Raspodijeljeni sustav objavi-preplati, u kojem se koristi algoritam preplavljivanja obavijestima, sastoji se od 3 posrednika i 3 klijenta kako je prikazano slikom. Svaki klijent u sustavu ima ulogu preplatnika i objavljuvaca. Odgovorite na sljedeća pitanja:

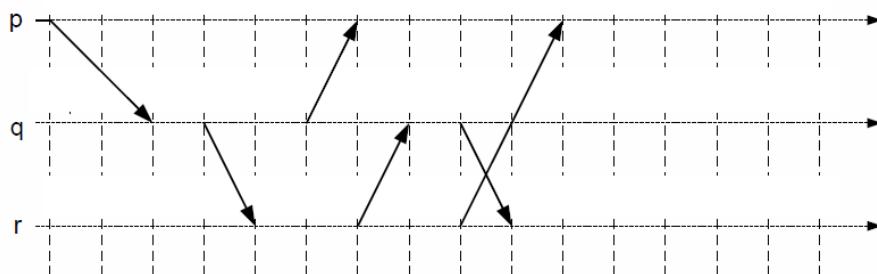


- a) U trenutku t1 **klijent 1** generira preplatu $s1=\{G=Zagreb, T<15.5, P>0.98\}$. Napišite oznake svih posrednika na kojima se pohranjuje ova preplata. **Pretplata se pohranjuje na posredniku B1.**
- b) U trenutku t2>t1 **klijent 2** generira preplatu $s2=s1$. Napišite oznake svih posrednika na kojima se pohranjuje ova preplata. **Pretplata se pohranjuje na posredniku B3.**
- c) U trenutku t3>t2 **klijent 3** generira obavijest $p1=\{G=Zagreb, T=-2.2, P=1.01\}$. Objasnite točan redoslijed kojim de se ova obavijest proširiti sustavom i biti isporučena zainteresiranim klijentima.
P3->B3(->S2)->B1(->S1)->B2

Za slijed razmijenjenih poruka između tri računala prikazan slikom, uspostavite globalni tijek vremena primjenom skalarnih oznaka logičkog vremena. Navedite i objasnite trenutke u kojima se ostvaruje korekcija lokalnih satnih mehanizama.



Ukratko objasnite razliku između skalarnih i vektorskih oznaka vremena prilikom sinkronizacije procesa u vremenu te navedite prednosti i nedostatke jedne i druge metode. Za slijed razmijenjenih poruka između tri računala prikazan slikom, uspostavite tijek akcija primjenom vektorskih oznaka logičkog vremena.

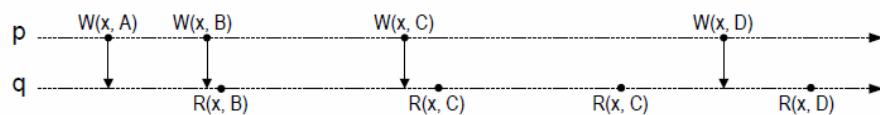


Drugi ciklus

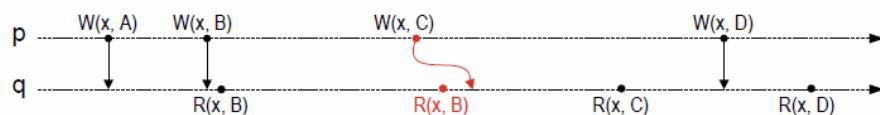
Objasnite što je stroga konzistentnost operacija u raspodijeljenim sustavima? Na primjeru procesa p i q prikažite slijed operacija čitanja i pisanja koji je a) u skladu i b) nije u skladu s načelima stroge konzistentnosti.

Model nalaže da su sve operacije pisanja trenutno vidljive svim ostalim procesima u raspodijeljenoj okolini. Naime, bez obzira koliko je kratko vremena prošlo od posljednje operacije pisanja, ako se provede operacija čitanja na bilo kojem računalu rezultat će biti upravo sadržaj zapisan u prethodnoj operaciji pisanja.

a)



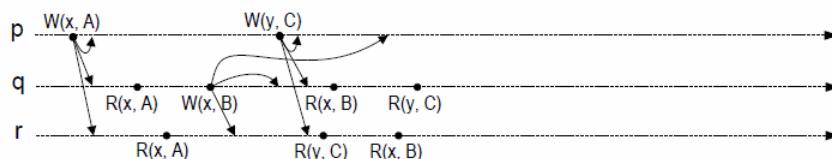
b)



Objasnite što je povezana konzistentnost operacija u raspodijeljenim sustavima? Na primjeru procesa p, q i r prikažite slijed operacija čitanja i pisanja koji je a) u skladu i b) nije u skladu s načelima povezane konzistentnosti.

Redoslijed izvođenja povezanih operacija pisanja vidljiv je svim procesima na jednak način, dok redoslijed izvođenja operacija pisanja koje nisu povezane svakom procesu može biti prikazan na drugačiji način.

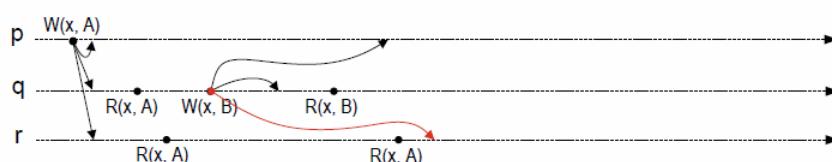
a)



Čitanje q: A->B->C

Čitanje r: A->C->B

b)

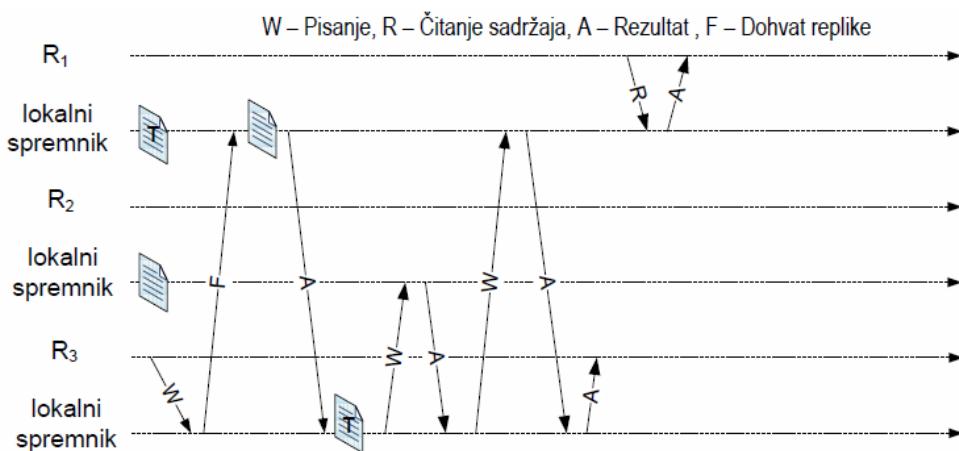


Čitanje q: A->B

Čitanje r: A->A

Raspodijeljeni sustav uključuje tri računala (R_0, R_1, R_2) s lokalnim spremnicima. U lokalnom spremniku računala R_1 nalazi se trajna replika dokumenta, dok se u lokalnom spremniku računala R_2 nalazi obična replika dokumenta. Korisnik putem računala R_3 provodi operaciju pisanja nad dokumentom primjenom postupka lokalnog obnavljanja stanja replike. Skicirajte i objasnite korake postupka.

R_3 upućuje zahtjev za provođenje operacije pisanja lokalnom spremniku sustava replika. Obzirom da to računalo u lokalnom spremniku ne sadrži trajnu repliku, računalo dohvaća trajnu repliku s R_1 u svoj lokalni spremnik. Replika na R_1 se pretvara u običnu repliku. Nakon prenošenja trajne replike u lokalni spremnik od R_3 , obnavlja se njeno stanje i prosljeđuje se ostalim računalima u sustavu. Nakon što ta računala pošalju potvrdu, R_3 prosljeđuje potvrdu korisniku koji je započeo operaciju pisanja. Ostali korisnici zatim imaju mogućnost provođenja operacije čitanja nad najbližom replikom u sustavu replika.



U sustavu replika koji se sastoji od glavnog poslužitelja i n=4 podjednako opterećena pomoćna poslužitelja, odredite metodu održavanja konzistentnosti replika za koju će prosječno mrežno (prometno) opterećenje poslužitelja L biti najmanje. Pri tome prepostavite da korisnike isključivo poslužuju pomoći poslužitelji, da je prosječna frekvencija upita $f_u=5$ upita/s, prosječna frekvencija promjena $f_p=1$ promjena/min te da su prosječne veličine upita/odgovora, operacija za promjenu sadržaja i replika $l_p=1\text{ kB}$, $l_o=50\text{ kB}$ i $l_r=100\text{ kB}$. Usporedite dobivena opterećenja s centraliziranim slučajem kada korisnike poslužuje glavni poslužitelj.

PUSH metoda s prosljeđivanjem obavijesti o promjenama: Nakon svake promjene, glavni poslužitelj šalje svakom pomoćnom poslužitelju obavijest o promjenama. Za prvi sljedeći korisnički zahtjev, svaki od pomoćnih poslužitelja šalje glavnom poslužitelju upit za novom replikom na koji mu glavni poslužitelj odgovara novom verzijom replike.

$$L_1 = n \cdot f_p \cdot l_p + n \cdot f_p \cdot l_p + n \cdot f_p \cdot l_r = n \cdot f_p \cdot (2 \cdot l_p + l_r) = 6,8 \text{ kB/s}$$

PUSH metoda s prosljeđivanjem operacija za promjenu sadržaja: Nakon svake promjene, glavni poslužitelj šalje svakom pomoćnom poslužitelju obavijest o operacijama koje je potrebno izvršiti nad prethodnom verzijom replike da bi se ona dovela u konzistentno stanje.

$$L_2 = n \cdot f_p \cdot l_o = 3,33 \text{ kB/s}$$

PUSH metoda s prosljeđivanjem cijelog kupa sadržaja replika: Nakon svake promjene, glavni poslužitelj šalje novu verziju replike svakom pomoćnom poslužitelju.

$$L_3 = n \cdot f_p \cdot l_r = 6,67 \text{ kB/s}$$

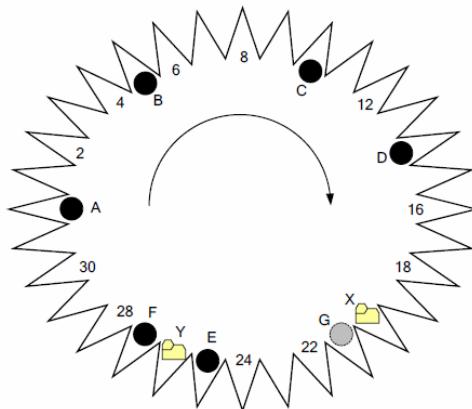
PULL metoda: Nakon svakog upita, pomoći poslužitelj provjerava kod glavnog poslužitelja je li došlo do promjene stanja replike koju pohranjuje lokalno. U $1/(n \cdot f_p)$ od $1/f_u$ slučajeva će poslužitelj odgovoriti novom verzijom replike, a u $1/(f_u - n \cdot f_p)$ od $1/f_u$ slučajeva će odgovoriti porukom da nije došlo do promjene.

$$L_4 = f_u \cdot l_p + n \cdot f_p \cdot l_r + (f_u - n \cdot f_p) \cdot l_p = 16,6 \text{ kB/s}$$

Centralizirani slučaj: Glavni poslužitelj odgovara replikom na svaki korisnički upit.

$$L_5 = f_u \cdot (l_p + l_r) = 505 \text{ kB/s}$$

Na slici je prikazana mreža Chord koja se sastoji od 6 čvorova (A, B, C, D, E i F) i koristi prostor identifikatora duljine N=32 (dovoljno je m=5 bita za kodiranje). Ukoliko je H1(A)=0, H1(B)=5, H1(C)=10, H1(D)=14, H1(E)=25 i H1(F)=27, odgovorite na sljedeća pitanja:



1. Popunite tablice usmjerenja čvorova A i F.

Routing table A(0)

1, 5
2, 5
4, 5
8, 10
16, 25

Routing table F(27)

28, 0
29, 0
31, 0
3, 5
11, 14

2. Na kojem će se čvoru pohraniti podatak X s ključem H2(X)=20?

Na čvoru E(25).

3. Odredite slijed čvorova preko kojih se usmjerava upit od čvora A s ciljem pronađaska podatka Y s ključem H2(Y)=26.

A-E-F

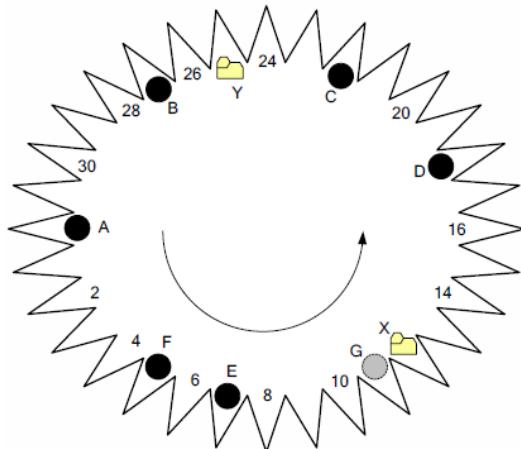
4. Dodan je novi čvor G (H1(G)=21) u mrežu.

Što će se promijeniti u tablici usmjerenja čvora A?

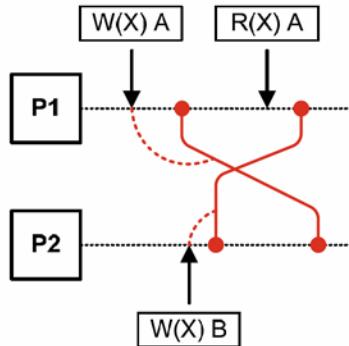
Umjesto 16, 25 pisat će 16, 21.

Na slici je prikazana mreža **Chord** koja se sastoji od 6 čvorova (**A**, **B**, **C**, **D**, **E** i **F**) i koristi prostor identifikatora veličine $N=32$ (dovoljno je $m=5$ bita za kodiranje). Ako su čvorovima pridijeljeni sljedeći ključevi je $H_1(A)=0$, $H_1(B)=27$, $H_1(C)=22$, $H_1(D)=18$, $H_1(E)=7$ i $H_1(F)=5$, odgovorite na sljedeća pitanja:

1. Popunite tablice usmjeravanja čvorova **A** i **F**.
2. Na kojem de se čvoru pohraniti podatak **X** s ključem $H_2(X)=12$?
3. Odredite slijed čvorova preko kojih se usmjerava upit od čvora **A** s ciljem pronalaska podatka **Y** s ključem $H_2(Y)=25$.
4. Dodan je novi čvor **G** ($H_1(G)=11$) u mrežu. Što de se promijeniti u tablici usmjeravanja čvora **A**?
5. Popunite tablicu usmjeravanja čvora **G**.

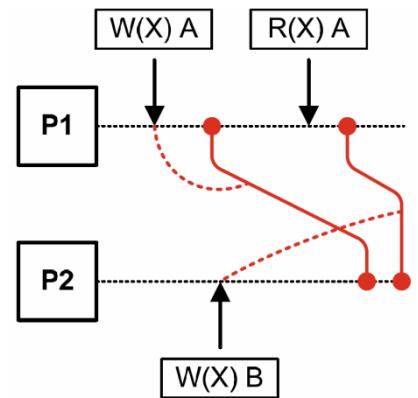


Na slici je prikazan redoslijed izvođenja operacija dvaju procesa. Objasnite poštije li prikazani slijed izvođenja operacija slijednu konzistentnost? Ako da, prikazani primjer promjenite tako da narušite slijednu nekonzistentnost izvođenja operacija. U suprotnom prikazani primjer promjenite tako da ostvarite slijednu nekonzistentnost izvođenja operacija. Obrazložite predloženo rješenje.

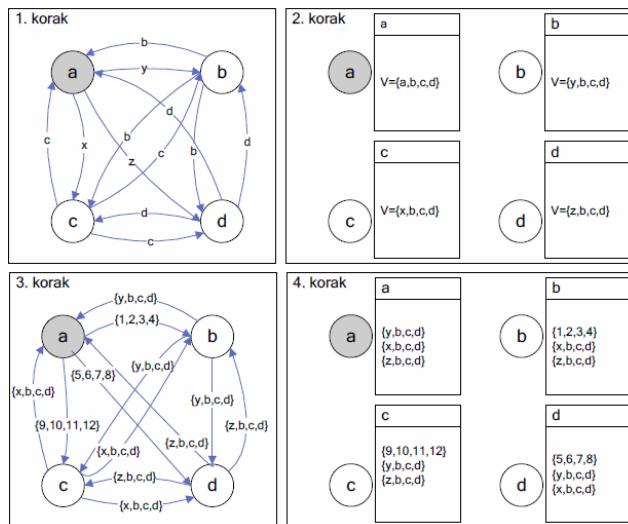


Slijedna konzistentnost nalaže da je redoslijed izvođenja operacija nebitan. Međutim, svi procesi moraju na jednak način vidjeti redoslijed izvođenja operacija u vremenu. U prikazanom primjeru nije ostvarena slijedna konzistentnost izvođenja operacija. Naime, proces P1 vidi da se na lokaciju X prvo zapisuje vrijednost A, te zatim vrijednost B, dok proces P2 vidi da se na lokaciju X prvo zapisuje vrijednost B a zatim vrijednost A.

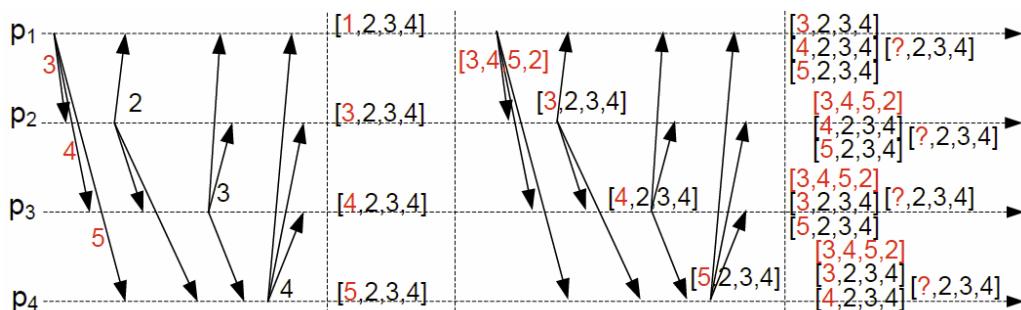
Da bi se ostvarila slijedna konzistentnost, potrebno je osigurati da svi procesi vide na jednak način redoslijed izmjena sadržaja lokacije X u vremenu. Primjerice, odgađanjem učinaka provođenja operacije pisanja na lokaciju X sadržaja B koju provodi proces P2, na način prikazan na slici .



U grupi od 4 procesa s identifikatorima a, b, c i d proces a je neispravan (prepostavite bizantski ispad). Grupa procesa želi postići sporazum o identifikatorima ostalih procesa grupe. U koracima 1 i 3 procesi međusobno razmjenjuju podatke, a u koracima 2 i 4 prikupljaju i analiziraju primljene podatke. Nacrtajte na slici koje podatke procesi razmjenjuju u koracima 1 i 3 (nacrtajte strelice i razmijenjene podatke uz svaku strelicu), a za korake 2 i 4 navedite podatke (napišite ih u pripremljene kućice) koje pojedini proces ima na raspolaganju radi donošenja odluke o sporazumu.



U grupi od 4 procesa (p_1, p_2, p_3 i p_4) proces p_1 je neispravan (prepostavite bizantski ispad). Grupa procesa želi postići sporazum o identifikatorima ostalih procesa grupe. U koracima 1 i 3 procesi međusobno razmjenjuju podatke, a u koracima 2 i 4 prikupljaju i analiziraju primljene podatke. Nacrtajte na slici podatke koje procesi razmjenjuju u koracima 1 i 3, a za korake 2 i 4 navedite podatke koje pojedini proces ima na raspolaganju radi donošenja odluke o sporazumu.



Sustav sadrži 3 serijske procesne jedinice s prosječnim vremenima posluživanja 1 s, 2 s i 3 s. Koliko de biti vrijeme zadržavanja u sustavu uz ulazni ritam zahtjeva od 0,1 z/s? Koliki će biti prosječni broj zahtjeva u sustavu?

Prosječna vremena posluživanja su $S_1 = 1 \text{ s/z}$, $S_2 = 2 \text{ s/z}$, $S_3 = 3 \text{ s/z}$.

U stabilnom stanju sustava je $X = L$.

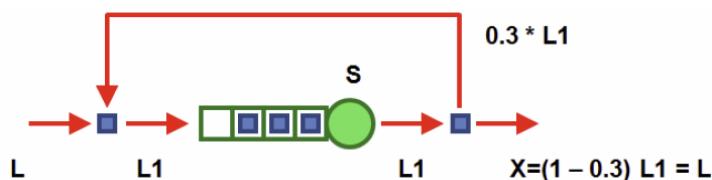
Propusnost sustava je $X = 0.1 \text{ z/s}$.

Vremena zadržavanja je $R_N = S_N / (1 - X * S_N)$.

$R_1 = 1.11 \text{ s}$, $R_2 = 2.5 \text{ s}$, $R_3 = 4.29 \text{ s}$

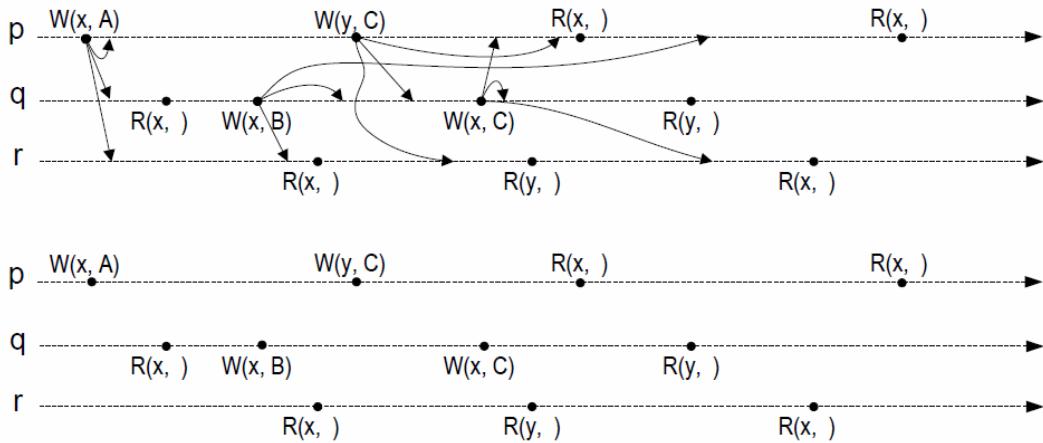
Prosječni broj zahtjeva u repu je $Q = X * (R_1 + R_2 + R_3) = 0.1 \text{ z/s} * (1.11 + 2.5 + 4.29) = 0.79 \text{ z}$.

Paketi dolaze u komunikacijski kanal s učestalošću 0,5 paketa u sekundi i zahtijevaju 0,75 sekundi za obradu. Za 30 % paketa dogodi se pogreška pri prijenosu i takvi paketi se umeću u rep za ponovno slanje. Koliko vremena paket prosječno provede u kanalu?

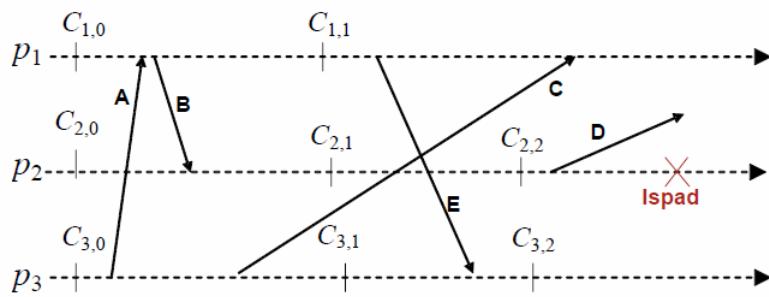


- Broj pristiglih paketa u sekundi je $L = 0.5 \text{ p/s}$.
- Prosječno vrijeme obrade paketa je $S = 0.75 \text{ s/p}$.
- Vjerojatnost pogreške paketa pri prijenosu je $p = 0.3$.
- $L_1 = p L_1 + L \Rightarrow L_1 = L / (1 - p)$
- $L_1 = L / (1 - p) = 0.5 / 0.7 = 0.714 \text{ p/s}$
- Prosječna zaposlenost kanala je $U = L_1 * S = 0.714 \text{ p/s} * 0.75 \text{ s/p} = 0.536$ (53.6 %).
- Srednje vrijeme čekanja u repu je $W = S * U / (1 - U) = 0.866 \text{ s/p}$.
- Srednje vrijeme zadržavanja paketa u kanalu je $R_1 = W + S = 0.866 \text{ s/p} + 0.75 \text{ s/p} = 1.616 \text{ s/p}$.
- Prosječno vrijeme koje paket provede u kanalu je $R = R_1 / (1 - p) = 2.31$.

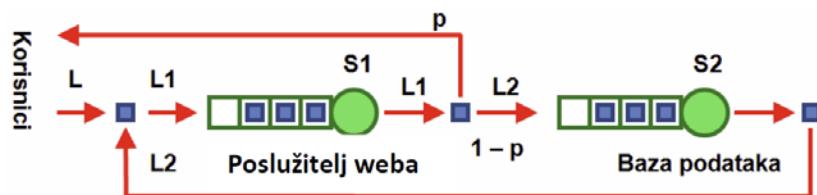
Na slici je prikazan redoslijed izvođenja operacija triju procesa. Objasnite poštuje li prikazani slijed izvođenja operacija konzistentnost redoslijeda? Ako da, prikazani primjer promijenite tako da narušite konzistentnost redoslijeda izvođenja operacija. U suprotnom prikazani primjer promijenite tako da ostvarite konzistentnost redoslijeda izvođenja operacija. Obrazložite predloženo rješenje.



Sustav na slici koristi strategiju oporavka unazad, a svaki proces bilježi kontrolne točke neovisno. Navedite konzistentno stanje u koje je sustav potrebno vratiti nakon ispada. Što će se dogoditi s porukama C i D nakon što se sustav vrati u konzistentno stanje i započne s dalnjim izvođenjem.

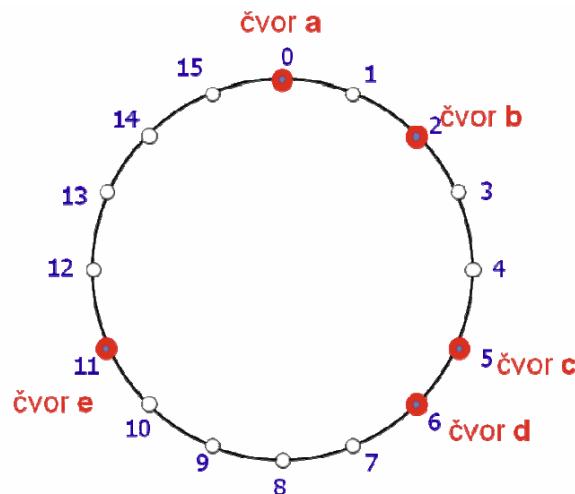


Web poslužitelj opisan je modelom zasnovanim na repovima koji je prikazan na slici. Vjerojatnost usmjeravanja zahtjeva prema korisniku (p) iznosi 0.9. Ako obrada zahtjeva na poslužitelju weba u prosjeku traje 0.01 sekundu, dok obrada zahtjeva na bazi podataka u prosjeku traje 0.1 sekundu. Koliko je vrijeme zadržavanja u sustavu uz ulazni ritam dolazaka zahtjeva u iznosu od 1 zahtjev u sekundi.



Opišite značajke raspoređivanja zasnovanog na korištenju prostorne lokalnosti i vremenske lokalnosti. Skicirajte primjer obje vrste raspoređivanja.

U strukturiranom P2P sustavu hash tablica ključeva koji omogućuju pronalaženje podataka raspodijeljena je na 5 čvorova (a, b, c, d, e) u prostoru identifikatora veličine $N = 16$, kojima su dodijeljeni identifikatori prema funkciji H1 kako slijedi:
 Čvor a: 0, Čvor b: 2, Čvor c: 5, Čvor d: 6 i Čvor e: 11.



Ukoliko se neki podatak pridijeli funkcijom H2 identifikatoru 8, objasnite na koji će se čvor taj podatak pohraniti?

Kako ne postoji čvor s identifikatorom 8, podatak će se pohraniti na prvi sljedeći čvor, a to je čvor e s identifikatorom 11.

Disk za trajno spremanje podataka ispunjava 50 zahtjeva u sekundi. Srednje vrijeme obrade zahtjeva operacija pisanja i čitanja je 10 ms. Disk ima prosječno 1 zahtjev u repu. Koliko je prosječno vrijeme čekanja na obradu zahtjeva?

Srednje vrijeme obrade zahtjeva je $S = 10 \text{ ms/z}$.

Propusnost sustava je $X = 50 \text{ z/s}$.

Broj zahtjeva u repu je $Q = 1 \text{ z}$.

Vrijeme zadržavanja zahtjeva u sustavu je $R = Q / X = (1 \text{ z}) / (50 \text{ z/s}) = 20 \text{ ms}$.

Vrijeme zadržavanja R uključuje vrijeme čekanja u repu (W) i vrijeme obrade zahtjeva (S): $R = W + S$.

Vrijeme čekanja na obradu je $W = R - S = 20 \text{ ms} - 10 \text{ ms} = 10 \text{ ms}$.

Web aplikacija uključuje podršku korisnicima putem chat usluge. Kupci sami odabiru jedan od 10 repova čekanja. Mjerenja pokazuju da zahtjevi prosječno dolaze 3 upita u minuti te da svaki kupac prosječno čeka 3 minute u repu i prosječno provodi 2 minute u konverzaciji. Koliko je srednje vrijeme zadržavanja kupaca za zadani sustav?

Prosječno vrijeme posluživanja je $S = 2 \text{ min/z}$.

Broj pristiglih zahtjeva u jednom repu je $L = 3 \text{ z/min}$.

U stabilnom stanju sustava $X = L$.

Pronađeno je $U = S L = (2 \text{ min/z}) (3 \text{ z/min}) = 6$.

Faktor iskorištenja je $ro = U / N = 6 / 10 = 0,6$.

Srednje vrijeme zadržavanja korisnika u sustavu je $R = S / (1 - ro) = 2 / (1 - 0,6) = 5 \text{ min}$.

Srednje vrijeme čekanja u repu je $W = R - S = 5 \text{ min} - 2 \text{ min} = 3 \text{ min}$.

Prepostavite da grupa procesa treba postidi sporazum. U slučaju da su dva procesa grupe u stanju bizantskog ispada, koji je minimalni ukupni broj procesa u grupi za postizanje sporazuma?

$$k = 2, N = 3k+1 = 7.$$

1 Uvod u raspodijeljene sustave

1.1. Objasnite ulogu programskog posredničkog sloja za raspodijeljene sustave.

- Programski posrednički sloj omogućava povezivanje, komunikaciju i suradnju aplikacija, sustava i uređaja te omogućuje interakciju programa na aplikacijskoj razini. Nalazi se između operacijskog sustava i aplikacijskih programa u programskoj arhitekturi sustava. Cilj je olakšati korisnicima i aplikacijama pristup i korištenje udaljenih sredstava na kontroliran i učinkovit način.

1.2. Usporedite migracijsku i relokacijsku transparentnost raspodijeljenog sustava. Koje svojstvo (ne)zadovoljava web poslužitelj i zašto?

- Migracijska transparentnost: prikrivanje promjene lokacije sredstva na način da ta promjena ne utječe na način pristupa sredstvu
- Relokacijska transparentnost: prikrivanje premještanja / kretanja sredstva
- Web poslužitelj zadovoljava migracijsku transparentnost. Mijenjanjem lokacije i/ili mrežnog sučelja mijenja se mrežna adresa, ali se ne mijenja simbolička adresa. Web poslužitelj ne zadovoljava relokacijsku transparentnost jer je poslužitelj stacionaran i ne kreće se tijekom pružanja usluge.

1.3. Objasnite pojam skalabilnosti raspodijeljenog sustava.

- Raspodijeljeni sustav je skalabilan ukoliko posjeduje sposobnost prilagodbe povećanom broju korisnika i sredstava, njihovoj rasprostranjenosti te načinu upravljanja sustavom.

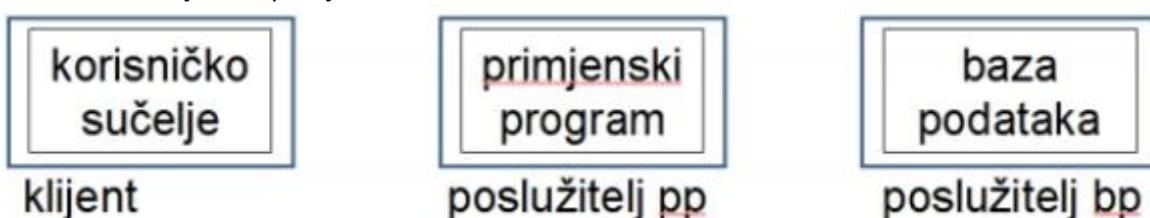
1.4. Kojim aspektima transparentnosti pridonosi sustav imenovanja domena (DNS)?

- Migracijska transparentnost, konkurencijska transparentnost i lokacijska transparentnost.

1.5. Napravite analizu transparentnosti raspodijeljenog sustava elektroničke pošte.

- Primjer gmaila:
Zadovoljava lokacijsku transparentnost jer korisnik pristupa usluzi preko simboličke adrese.
Zadovoljava konkurencijsku transparentnost jer uslugu koristi puno korisnika, a individualan korisnik ima dojam kao da je on jedini. Transparentnost pristupa, prikrivaju se razlike u pristupu sredstvu korisnicima s različitim OS-ima (npr. Mac, MS Windows itd.).
Transparentnost na kvar i replikacijska transparentnost, kopije podataka se čuvaju na drugim poslužiteljima za slučaj kada se jedan poslužitelj pokvari, da drugi može preuzeti posao bez da korisnik primjeti da je došlo do kvara.

1.6. Prikažite i objasnite primjer troredne arhitekture weba.



- Primjer su aplikacije weba, gdje korisnički program koji se izvodi na klijentskom računalu nikad ne pristupa direktno bazi podataka, već posredno preko aplikacije weba. Klijentski program prikazuje korisničko sučelje i komunicira s aplikacijom weba koja obavlja cijelokupnu logiku usluge i pristupa potrebним podacima.

- 1.7. Kakvi bi se problemi pojavili kad bi se više repliciranih poslužitelja weba priključilo na mrežu izravno, a ne putem zastupnika (proxy)?
- Korisnik ne bi znao kojem poslužitelju pristupa i sa svakim sljedećim pristupanjem ne bi bilo zajamčeno pristupanje sadržaju kojem je prethodno pristupio.

- 1.8. Kakvi su tržišni udjeli web-poslužitelja i web-preglednika?

- Wtf?

- 1.9. Što je vrijeme odziva za poslužitelja weba?

- Vrijeme odziva je vrijeme koje je potrebno da poslužitelj procesira zahtjev i da odgovor stigne do korisnika.

2 Procesi i komunikacija

- 2.1. Objasnite na primjeru razliku između perzistentne i tranzijentne komunikacije

-Primjer perzistentne komunikacije bi bio komunikacija porukama. Korisnik 1 šalje poruku korisniku 2 i sustav jamči da će korisnik 2 dobiti tu poruku, bez obzira je li mu upaljen ili ugašen uređaj. Primjer tranzijentne komunikacije je chat sustav baziran na UDP-u gdje korisnik 1 šalje poruku korisniku 2, ali se ne jamči isporuka poruke ako korisnik 2 nije aktivan.

- 2.2. Objasnite razliku između sinkrone i asinkrone komunikacije.

-Kod sinkrone komunikacije pošiljatelj je blokiran nakon slanja poruke sve do primitka potvrde o isporuci ili sve do primitka odgovora od poslužitelja, ovisno o implementaciji. Kod asinkrone komunikacije pošiljatelj nije blokiran te nastavlja procesiranje odmah nakon slanja.

- 2.3. Objasnite zašto tranzijentna sinkrona komunikacija potencijalno pati od problema vezanih uz skalabilnost.

-Uzmimo na primjer da imamo jednog korisnika. On pošalje poruku poslužitelju i budući da je komunikacija sinkrona korisnik je blokiran, ne radi ništa dok ne primi odgovor. Ali, budući da je komunikacija tranzijentna moguće je da poslužitelj nije dobio poruku i korisnik tada ulazi u beskonačnu petlju čekanja. Pri povećanju skalabilnosti se povećava broj korisnika kojima sustav ne radi ispravno.

- 2.4. Navedite prednosti koje ima operacija slanja zahtjeva koja je neblokirajuća u odnosu na blokirajuću operaciju.

-Korisnik koji šalje zahtjev nije blokiran i može nastaviti s radom dok čeka zahtjev. Za to vrijeme može poslati nove zahtjeve drugim poslužiteljima ili može napraviti neki unutarnji posao.

- 2.5. Ima li smisla ograničiti broj dretvi za obradu korisničkih zahtjeva na višedretvenom poslužitelju? Zašto?

-Valjda ima smisla jer se ostatak dretvi koristi za npr. obradu unutarnjih poslova.

- 2.6. Je li poslužitelj koji održava TCP/IP konekciju prema klijentu stateful ili stateless?

-Poslužitelj je stateful jer postoje zahtjevi unutar TCP/IP protokola koji mijenjaju stanje konekcije, npr. ako se korisnik želi odspojiti, poslati će zahtjev za raskid konekcije.

2.7. Navedite prednosti konkurentnog poslužitelja u odnosu na iterativni poslužitelj.

-Prednost je što konkurentni poslužitelj može posluživati više korisnika odjednom i samim time se povećava efikasnost sustava.

3 Sloj raspodijeljenog sustava za komunikaciju među procesima

3.3 Poslužitelj je implementiran pomoću socketa TCP s ograničenjem veličine repa za dolazne zahtjeve na instanci klase ServerSocket (tzv. backlog) na 10. Što se događa s novim zahtjevima ako je prilikom dolaska novog zahtjeva u tom repu već na čekanju 10 zahtjeva?

-Zahtjevi se odbacuju.

3.4. Na koji je način moguće ograničiti broj paralelnih konekcija prema višedretvenom poslužitelju kao što je npr.UpperCaseServer?

- Wtf?

3.5. U tablicama su prikazane metode na klijentskoj i poslužiteljskoj strani socketa TCP. Upišite ispravan redoslijed izvođenja metoda u tablice.

Klijent	Klijent
Socket()	Socket()
Write()	Connect()
Read()	Write()
Close()	Read()
Connect()	Close()

Mnemonička metoda ili šalabahter: SCWRC

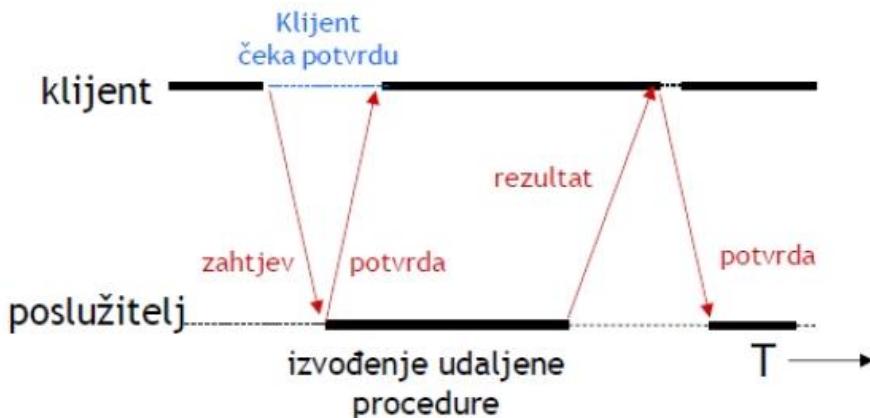
Poslužitelj	Poslužitelj
Listen()	Socket()
Socket()	Bind()
Accept()	Listen()
Write()	Accept()
Read()	Socket()
Close()	Read()
Bind()	Write()
	Close()

Mnemonička metoda ili šalabahter: SBLASRWC

3.6. Može li se pomoću UDP-a implementirati protokol za pouzdanu komunikaciju između klijenta i poslužitelja? Ako može, na koji način?

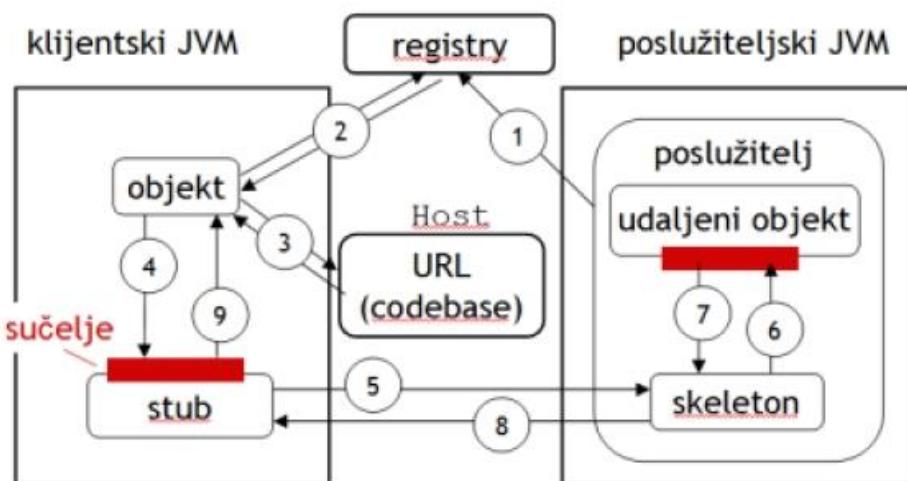
-Može se implementirati pomoću slanja potvrda za svaki UDP paket. Ako korisnik 1 nije dobio potvrdu od korisnika 2 za primitak poslanog UDP paketa nakon određenog vremena, korisnik 1 isponova šalje taj određeni UDP paket.

3.7. Skicirajte tijek komunikacije između klijenta i poslužitelja te objasnite odgođeni sinkroni poziv udaljene procedure RPC (Remote Procedure Call).



- Kod odgođenog sinkronog poziva udaljene procedure, klijent nije blokiran dok čeka rezultat izvođenja, već nastavlja s radom nakon uspješnog primitka potvrde. Kasnije mu poslužitelj šalje rezultat koristeći drugi asinkroni poziv udaljene procedure.

3.8. Skicirajte model pozivanja udaljene metode Java RMI (Remote Method Invocation). Navedite korake u komunikaciji potrebne da bi klijent pozvao metodu dostupnu na poslužitelju, uz pretpostavku da klasa stub već postoji i dostupna je na klijentskoj strani.



- 1) Poslužitelj definira codebase udaljenog objekta i registrira ga pod odabranim imenom.
- 2) Klijent od registra traži referencu na udaljeni objekt, i dobiva je.
- 3) Klijent traži i dobiva klasu stub koristeći codebase.
- 4) Klijent poziva metodu stuba dostupnu na klijentskom računalu.
- 5) Stub serijalizira parametre i šalje ih skeletonu.
- 6) Skeleton deserijalizira parametre i poziva metodu udaljenog objekta.
- 7) Udaljeni objekt vraća rezultat izvođenja metode skeletonu.
- 8) Skeleton serijalizira rezultat i šalje ga stubu.

9) Stub deserijalizira rezultat i dostavlja ga klijentu.

3.9. Ima li smisla implementirati perzistentnu asinkronu komunikaciju pomoću RMI-ja? Zašto?

- wtf?

3.10 Objasnite razliku između transportne adrese i adrese repa u sustavima za komunikaciju porukama?

- Svaki rep ima jedinstveno ime (tzv. adresa repa) neovisno o samoj transportnoj adresi te je potrebna usluga koja povezuje ime repa s transportnom adresom (analogno DNS-u). Adresiranje se izvodi najčešće na nivou sustava, a svaki rep ima jedinstveni identifikator i stoga komunikacija porukama nije anonimna. Pošiljatelj i primatelj ne moraju znati ništa jedan o drugome, no moraju znati adresu repa i format poruke kako bi mogli komunicirati.

3.11 Skicirajte i objasnite primjer komunikacije porukama između dva procesa/objekta (primatelja i pošiljatelja). Kakva je komunikacija porukama s obzirom na vremensku ovisnost primatelja i pošiljatelja?



- U komunikaciji između pošiljatelja i primatelja rep sudjeluje kao posrednik. Pošiljatelju se u načelu garantira isporuka poruke u primateljev rep, ali ne i isporuka poruke primatelju. Primatelj može pročitati poruku iz repa u bilo kojem budućem trenutku.

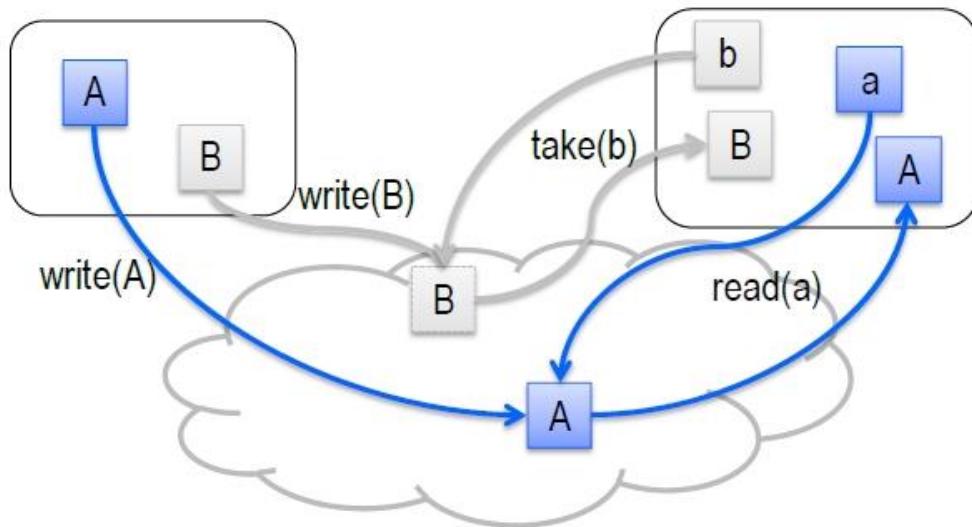
- Primatelj ne mora biti aktivan za vrijeme kada pošiljatelj želi poslati poruku primatelju.

3.12. Navedite i objasnite operacije koje implementira programska infrastruktura dijeljenog podatkovnog prostora.

- write (t): dodaj tuple t u raspodijeljeni podatkovni prostor

- read (s) -> t: vraća tuple t koji odgovara predlošku s

- take (s) -> t: vraća tuple t koji odgovara predlošku s i briše ga iz podatkovnog prostora



3.13. Navedite sličnosti i razlike komunikacije na načelu objavi-preplati i dijeljenog podatkovnog prostora.

- Obje vrste komunikacije su vremenski neovisne zato što pošiljatelj i primatelj ne moraju istovremeno biti dostupni da bi se komunikacija mogla ostvariti. Kod komunikacije porukama pošiljatelj mora znati identifikator odredišta, dok je kod modela objavi-preplati komunikacija anonimna. Komunikacija je perzistentna i asinkrona u oba slučaja. Komunikacija se pokreće na načelu pull kod komunikacije porukama, a na načelu push kod modela objavi-preplati.

3.14. Usporedite preplatu u sustavima objavi-preplati i predložak u sustavima s dijeljenim podatkovnim prostorom. Može li se koristeći navedenu međupremu realizirati tzv. anonimnu komunikaciju (pošiljatelj ne zna adresu primatelja) i zašto?

-Može se kombinirati posrednik iz sustava objavi-preplati s dijeljenim podatkovnim prostorom. Komunikacija u sustavu dijeljenog podatkovnog prostora je anonimna jer se temelji na sadržaju podataka. Dakle, posrednik iz sustava objavi-preplati komunicira s krajnjim korisnicima na temelju dijeljenog podatkovnog prostora.

3.15. Gdje se filtriraju obavijesti u raspodijeljenom sustavu objavi-preplati koji koristi preplavljivanje obavijestima?

-Vrši se u krajnjem posredniku prije isporuke obavijesti lokalnim preplatnicima.

4 Arhitekture web-aplikacija

4.1. Navedite i objasnite najčešće korištene metode protokola HTTP/1.1.

- OPTIONS – služi za informiranje i dohvaćanje mogućnosti resursa i poslužitelja,
- GET – koristi se za dohvaćanje resursa (najčešće u uporabi),
- HEAD – koristi se za dohvaćanje podataka o resursu (npr. veličina, provjera postojanja),
- POST – služi za aktiviranje resursa (npr. slanje podataka obrazaca),
- PUT – postavljanje resursa (npr. promjena podataka u web-uslugama),

- DELETE – brisanje resursa (npr. kod web-usluga),
- TRACE – koristi se za dijagnostiku i
- CONNECT – predviđeno za buduću uporabu.

4.2. Objasnite namjenu jezika CSS. Zašto se pojavila potreba za definiranjem takvog jezika?

- Jezik CSS (*Cascading Style Sheets*) je jezik za definiranje pravila prikaza sadržaja HTML-ovog dokumenta, a nastao je iz potrebe odvajanja sadržaja i prikaza dokumenta.

4.3. Korisnik nakon ispunjavanja obrasca na Web-u odabire opciju *Submit*, čime pošalje podatke Web-poslužitelju na adresu www.tel.fer.hr/obrazac/accept korištenjem protokola HTTP verzije 1.1. Kojim se HTTP zahtjevom šalju podaci poslužitelju i kako je definiran prvi redak zahtjeva?

- Podaci se šalju POST HTTP zahtjevom. Prvi redak je definiran na sljedeći način: POST /obrazac/accept HTTP/1.1

4.4. Objasnite opći format poruka protokola HTTP. Navedite kako glasi potpun i apsolutan URI koji identificira resurs zatražen u zahtjevu, ako prva 2 retka HTTP zahtjeva sadrže sljedeće podatke:

GET /predmet/rassus HTTP/1.1

Host: www.fer.unizg.hr

- Poruka se sastoji od sljedećih elemenata: Početnog retka koji je obvezan, zaglavljiva koja se sastoje od polja (neka polja su obvezna), prazan redak i tijela poruke. Obvezni dijelovi su početni redak, dio zaglavljiva s poljem Host i prazan redak.
- <http://www.fer.unizg.hr/predmet/rassus>

4.5. Objasnite ulogu posredničkog poslužitelja s priručnim spremištem koji se nalazi u mreži između web-preglednika i web-poslužitelja. Gdje se sve može nalaziti u mreži?

- Posrednički poslužitelj s priručnim spremištem služi kao privremeno spremište. S njim se ubrzava posluživanje i manje je se tereti mreža između posredničkog poslužitelja i poslužitelja klijenta. Može se nalaziti u lokalnoj mreži klijenta, javnoj mreži ili u lokalnoj mreži poslužitelja.

4.6. Navedite i objasnite prednosti korištenja posredničkog poslužitelja s priručnim spremištem za korisnika, izvorni poslužitelj i komunikacijsku mrežu.

- Prednosti s korisničke strane su brže učitavanje sadržaja koji je spremljen u priručno spremište i smanjuje se korištenje veze između lokalne mreže i ostatka mreže.
- Prednosti s izvornog poslužitelja je privremeno spremanje izračunatih podataka: posrednički poslužitelj može svježe resurse spremiti u svoje priručno spremište i njih slati kao odgovor te na taj način rasteretiti poslužitelja usluge.
- Prednost za komunikacijsku mrežu je smanjenje korištenih mrežnih resursa.

4.7. Prepostavite da se sjedište weba sastoji od 2 poslužitelja priključena na Internet preko posrednika (*proxy*). Navedite i objasnite svojstva ovog raspodijeljenog sustava.

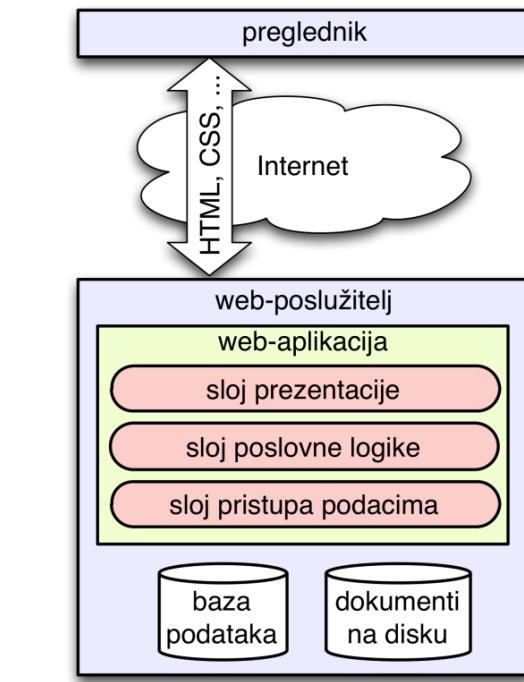
- Zastupnik poslužitelja (*proxy*) posreduje između klijenata i poslužitelja tako da od klijenata prikriva broj i lokaciju poslužitelja te način na koji su povezani (tj. omogućava replikacijsku transparentnost). Također, proxy pruža mogućnost rasterećenja rada poslužitelja tako da nadolazeće zahtjeve preusmjeri na onog poslužitelja koji ja najmanje zaposlen, kriptira ili komprimira podatke i privremeno sprema izračunate podatke i njih šalje kao odgovor, što naravno rasterećuje poslužitelja usluge. Ujedno, može dozvoliti pristup samo nekim uslugama i na taj način poslužitelj usluge ne mora biti toliko siguran jer ga štiti posrednički poslužitelj koji može osiguravati SSL komunikaciju između klijenta i posredničkog poslužitelja, a komunikacija između posredničkog poslužitelja i poslužitelja usluga ne treba biti zaštićena.

4.8. Objasnite razliku između web-aplikacija temeljenih na CGI (Common Gateway Interface) i poslužiteljskim skriptama.

- CGI (Common Gateway Interface) je jednostavno sučelje za pokretanje eksternih programa iz web-poslužitelja na platformski i programski neovisan način. Kod svakog zahtjeva se pokreće novi proces, a podaci između poslužitelja i procesa šalju se preko varijabli okoline i tokova podataka. Nakon svake obrade proces se gasi.
- Poslužiteljske skripte isto dinamički generiraju HTML-dokumente, ali je razlika u tome što se za svaki zahtjev ne pokreće novi proces i na taj način se štede resursi.

4.9. Skicirajte i objasnite slojevitu arhitekturu web-aplikacija.

- Web-aplikacije su obično organizirane u 3 sloja:
 - sloj prezentacije – služi za prikaz informacija (GUI, HTML, klikovi mišem, ...) i za obradu HTTP-zahtjeva;
 - sloj poslovne (domenske) logike – obrađuje podatke koje je dobio od sloja prezentacije. To je glavni dio sustava koji radi ono za što je sustav namijenjen;
 - sloj pristupa podacima – komunicira s bazom podataka i drugim komunikacijskim sustavima te se brine o transakcijama i o pohrani podataka.

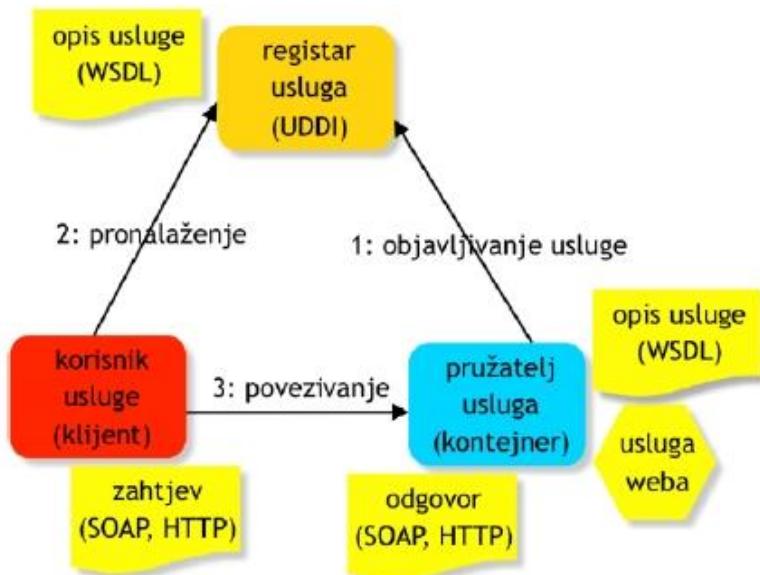


4.10. Koja je prednost korištenja tehnike dugog prozivanja poslužitelja za web-aplikacije?

Prednost je što se u slučajevima kada nema puno poruka ne opterećuje mreža s klijentovim zahtjevima i poslužiteljevim praznim odgovorima. Nego klijent pošalje jedan zahtjev i poslužitelj mu odgovori s porukom tek kada ta poruka stigne od drugog klijenta.

5 Web usluge

5.1. Skicirajte i objasnite arhitekturu web-usluge te interakcije između komponenti web-usluge.



Kada se na pružatelju usluge instalira web-usluga, pružatelj ju prvotno registrira u registru usluga. Koristeći protokol UDDI pružatelj objavi uslugu i registru šalje opis usluge u WSDL-u. Kada klijent želi pronaći uslugu, prvo registru pošalje upit za pronalaženje, a registar mu odgovara opisom usluge koja odgovara zadanim kriterijima. Nakon toga se klijent može povezati s pružateljem usluge jer ima opis usluge. Klijent pomoću protokola HTTP i SOAP šalje zahtjev za pozivom usluge, pružatelj usluge pokreće izvođenje usluge i šalje natrag rezultat u odgovoru HTTP i SOAP protokolima.

5.2. Navedite i ukratko objasnite tri vrste web-usluga.

1. Poziv udaljene procedure – usko povezuje klijenta i poslužitelja jer je usluga usko vezana za programski jezik jer se za svaki programski jezik koristi alat koji automatski iz programskog jezika na svoj način definira elementa WSDL-a.
2. Usluge temeljene na dokumentima / porukama – ne povezuje jako klijenta i poslužitelja odnosno njihove implementacijske jezike jer je fokus na ugovorima koji su propisani WSDL-om.
3. Usluge temeljene na stanju resursa - koristi protokol HTTP pa je sučelje dobro definirano, fokus je na interakciji s resursima koji imaju stanje, a ne na porukama ili operacijama.

5.3. Prikažite arhitekturu i objasnite korištenje usluge Weba.

Valjda isto kao 5.1.?

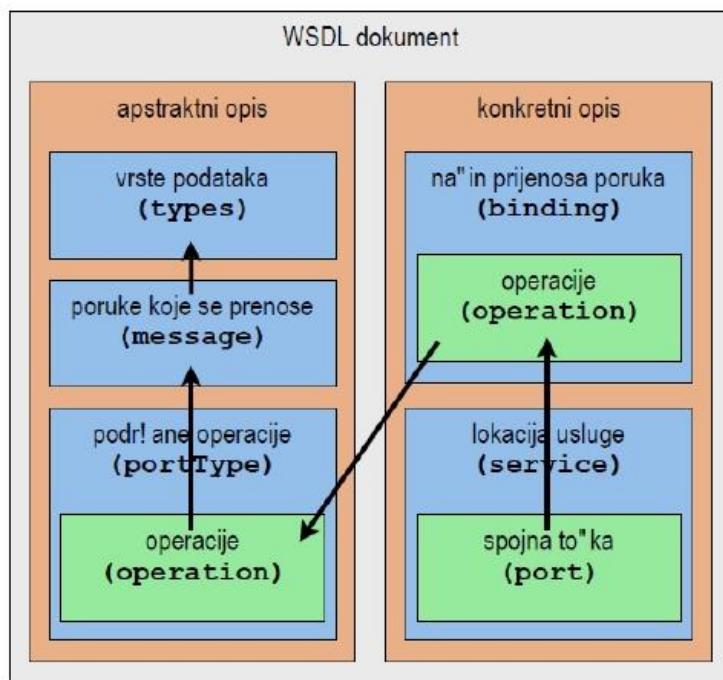
5.4. Navedite dva osnovna načina rada protokola SOAP i objasnite kako se poruka SOAP šalje pomoću protokola HTTP.

Dva osnovna načina rada protokola SOAP su:

- 1) Poziv udaljene procedure – služi za prijenos serijaliziranih parametara i rezultata. Korist ovog načina rada je dobro definirano sučelje i tipovi podataka, te prilagodni kod može biti generiran automatski.
- 2) Razmjena dokumenata / poruka – koriste se XML dokumenti za razmjenu poruka.

5.5. Navedite i objasnite primjenu dokumenta u WSDL-u. Od kojih se dijelova sastoji takav dokument?

Dокумент u WSDL-u se koristi za opis web-usluge. Dokument se sastoji od apstraktnog i konkretnog dijela.



Types – definira vrste podataka neovisne o platformi i jeziku

- 2) Message – definiraju ulazne i izlazne poruke koje se mogu koristiti kao parametri usluga
- 3) Operation – predstavlja jednu operaciju/metodu/proceduru koja je definirana u usluzi, a sastoji se od definicija ulaznih, izlaznih i iznimnih poruka koje se mogu razmjenjivati
- 4) Binding – definira kako je konkretna implementacija povezana s operacijama u apstraktnom opisu i definira formt u kojem de se poruke prenosi
- 5) Service – definirani URI preko kojeg se usluga može pozvati

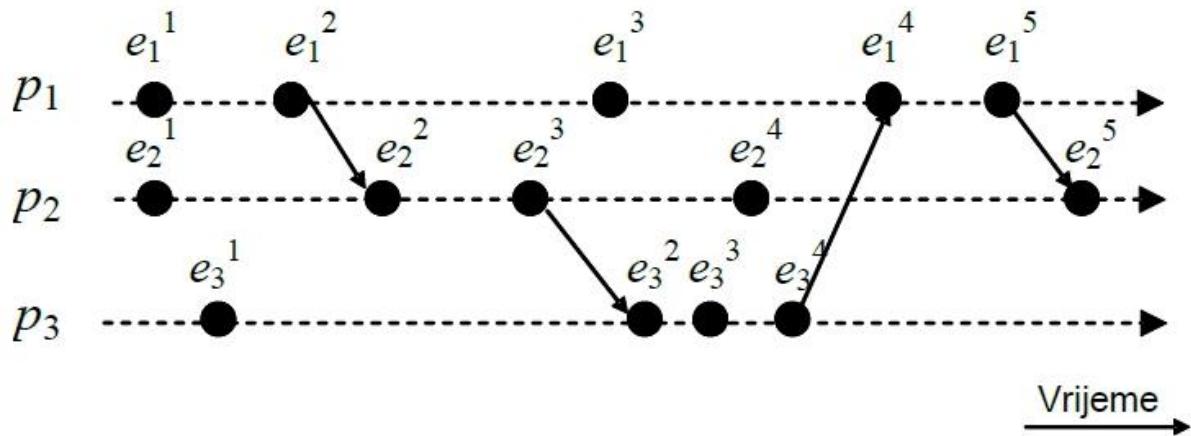
5.6. Objasnite svojstvo idempotentnosti za web-usluge temeljene na REST-u.

Metoda koja ima svojstvo idempotentnosti se može pozvati više puta, ali rezultat će uvijek biti isti.

6. Model raspodijeljenog sustava

- 6.1. Za koje je svojstvo raspodijeljenih sustava značajna komunikacijska složenost algoritama? Zašto?
- Za skalabilnost sustava. Na temelju komunikacijske složenosti možemo zaključiti kako raste generirani promet raspodijeljenog sustava s rastom tog sustava.

- 6.2. Na temelju primjera procesa sa slike objasnite jesu li sljedeći parovi događaja uzročno povezani ili nisu? a) e_1^3 i e_2^2 i b) e_2^2 i e_1^5 .

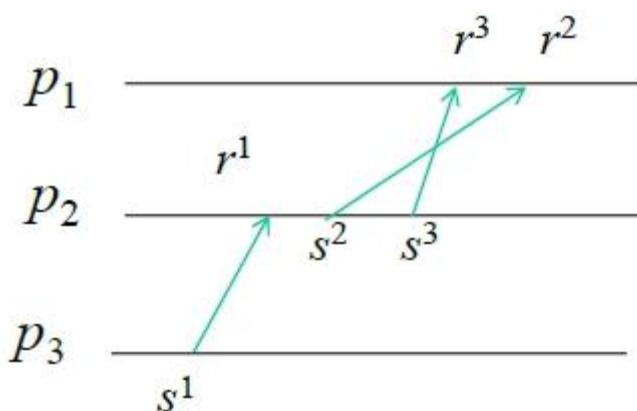


- a) $e_1^3 \rightarrow e_2^2$, zato što:
 $e_1^2 \rightarrow e_2^2$, e_1^2 se dogodio prije e_2^2 , ali e_1^3 se dogodio nakon e_1^2 stoga ne postoji tranzitivna ovisnost i ne e_2^2 i e_1^3 ne razmjenjuju poruke.
- b) $e_2^2 \rightarrow e_1^5$?
 $e_2^2 \rightarrow e_2^3$, $e_2^3 \rightarrow e_3^2$, $e_3^2 \rightarrow e_3^3$, $e_3^3 \rightarrow e_3^4$, $e_3^4 \rightarrow e_1^4$, $e_1^4 \rightarrow e_1^5$. Preko tranzitivnosti $e_2^2 \rightarrow e_1^5$

6.3. Objasnite model komunikacijskog kanala koji se temelji na uzročnoj slijednosti.

Vrijedi li za sljedeći primjer CO ili non-CO i zašto?

- Kanal koji osigurava uzročnu slijednost (causal ordering, CO) osigurava da uzročno povezani događaji slanja dviju poruka istom primatelju rezultiraju primanjem u slijedu kojim su poslati.
-

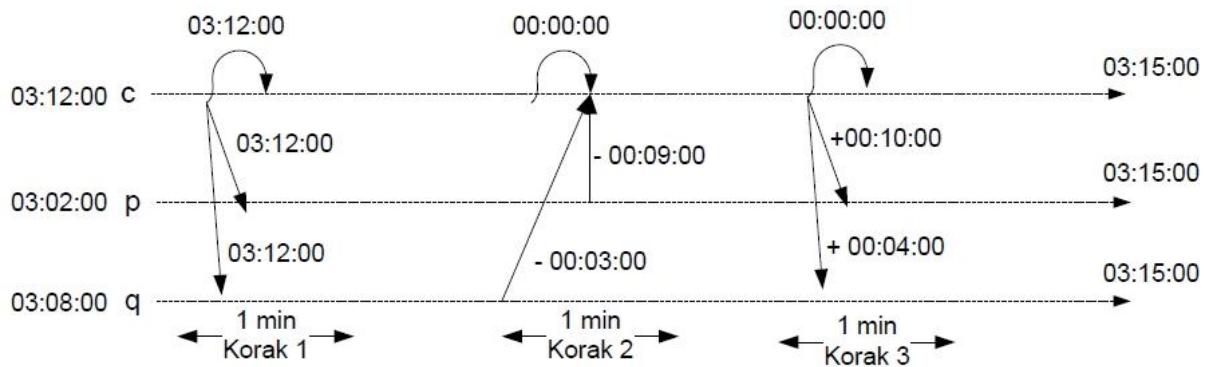


- Non-CO. Proces 2 šalje poruke procesu 1 slijedom s_2 , s_3 . Proces 1 umjesto da te poruke dobije istim slijedom kojim su poslane, on ih dobije obrnutim redoslijedom.

7 Sinkronizacija procesa u vremenu

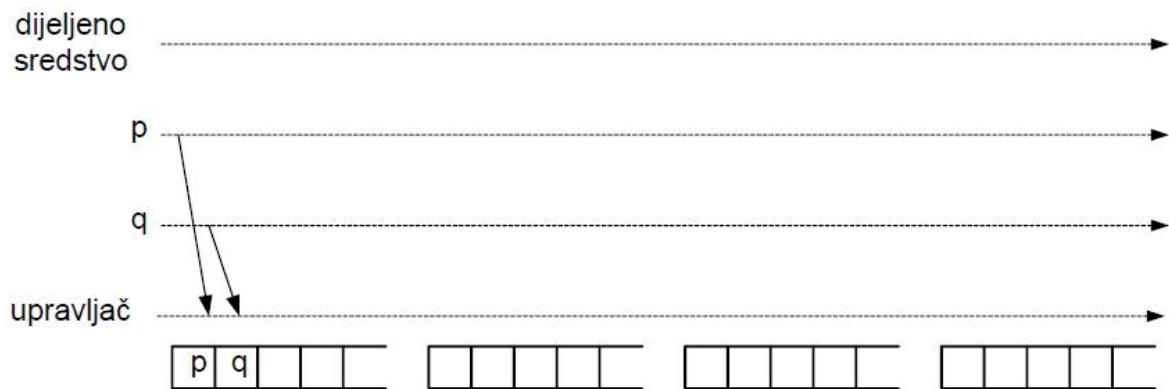
7.1. Prikažite i objasnite korake algoritma Berkeley za usklađivanje satnih mehanizama tri računala u raspodijeljenoj okolini. Računala imaju sljedeće vrijednosti satova $T(p)=03:02:00$, $T(q)=03:08:00$ i

$T(c)=03:12:00$. Upravitelj je treće računalo. Pretpostavite da prijenos poruke između 2 računala traje 1 minuta i da upravitelj koristi svoje lokalno vrijeme kao zajedničko pri usklađivanju satnih mehanizama.

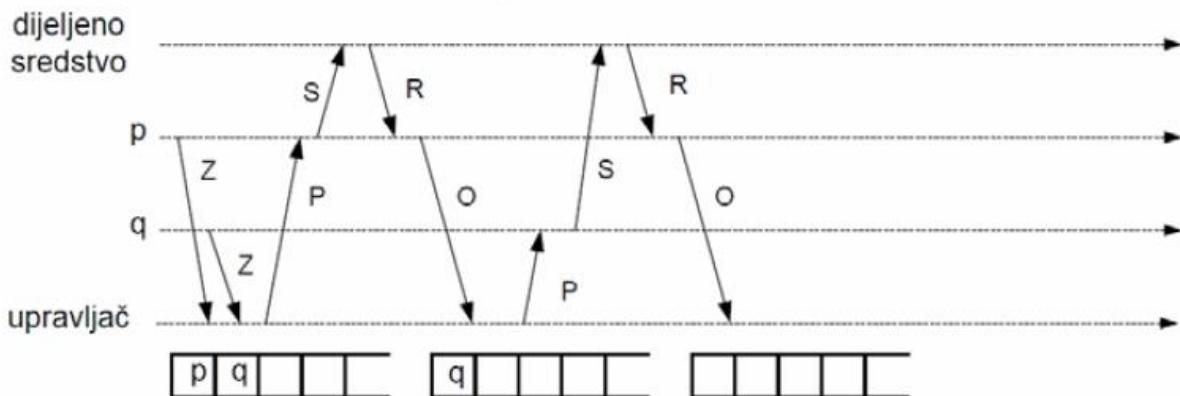


7.2. Opišite postupak međusobnog isključivanja dvaju procesa (p i q) primjenom središnjeg upravljača s repom čekanja tako da nacrtate redoslijed operacija i objasnite ih. Nakon zauzimanja dijeljenog spremnika, proces provodi jednu operaciju čitanja ili pisanja nad dijeljenim spremnikom.

R –Dohvati, S –Spremi, Z –Zauzmi, P –Potvrda, O –Oslobodi



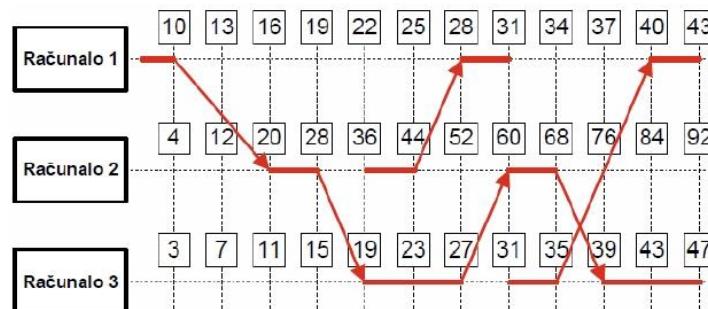
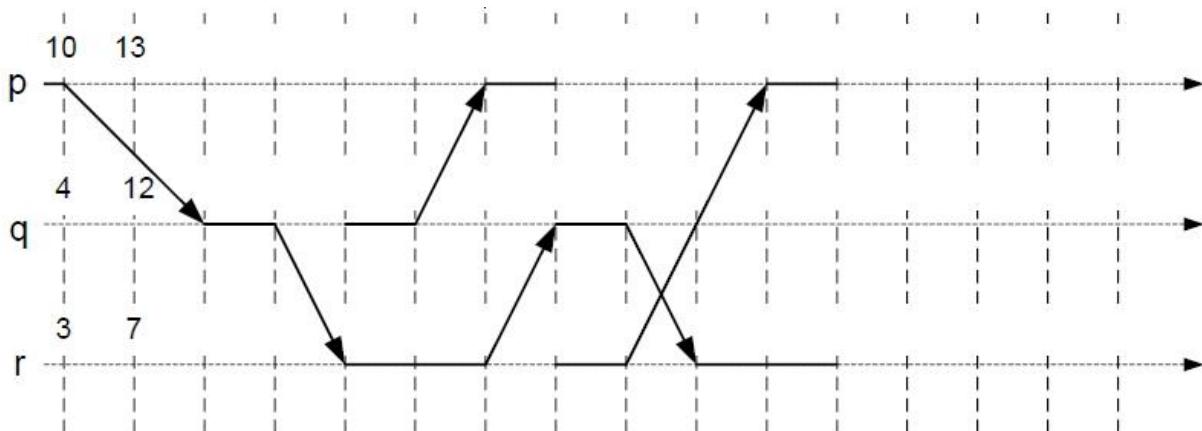
R –Dohvati, S –Spremi, Z –Zauzmi, P –Potvrda, O –Oslobodi



- Proces p šalje zahtjev za zauzimanje sredstva, zahtjev se sprema u rep
- Proces q šalje zahtjev za zauzimanje sredstva, zahtjev se stavlja u rep
- Kako je zahtjev od R0 stigao prije, upravljač šalje potvrdu R0 i uklanja njegov zahtjev iz repa
- Proces p provodi operaciju pisanja

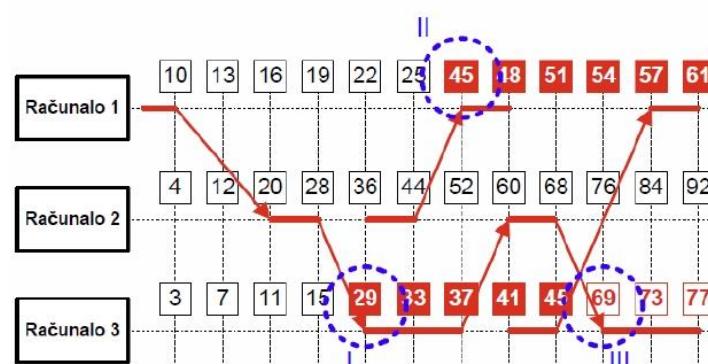
- Proces p prima potvrdu
- Proces p šalje poruku Upravljaču i otpušta pristup
- Upravljač šalje poruku dojave procesu q te mu dodjeljuje pristup dijeljenom spremniku. Iz repa zahtjeva uklanja se zahtjev od procesa p
- Proces q provodi operaciju pisanja
- Proces q prima potvrdu
- Proces q šalje poruku Upravljaču i otpušta pristup

7.3. Za slijed razmjene poruka između tri računala prikazan na slici uspostavite globalni tijek vremena primjenom skalarnih oznaka logičkog vremena. Navedite i opišite trenutke u kojima se ostvaruje korekcija lokalnih satnih mehanizama.



Trenutak I: Računalo 3 prima poruku od računala 2 s oznakom vremena $T_p=28$ koja je veća od lokalne oznake vremena $T_L=19$. Lokalni sat se pomiče na vrijednost $T_p+1=29$.

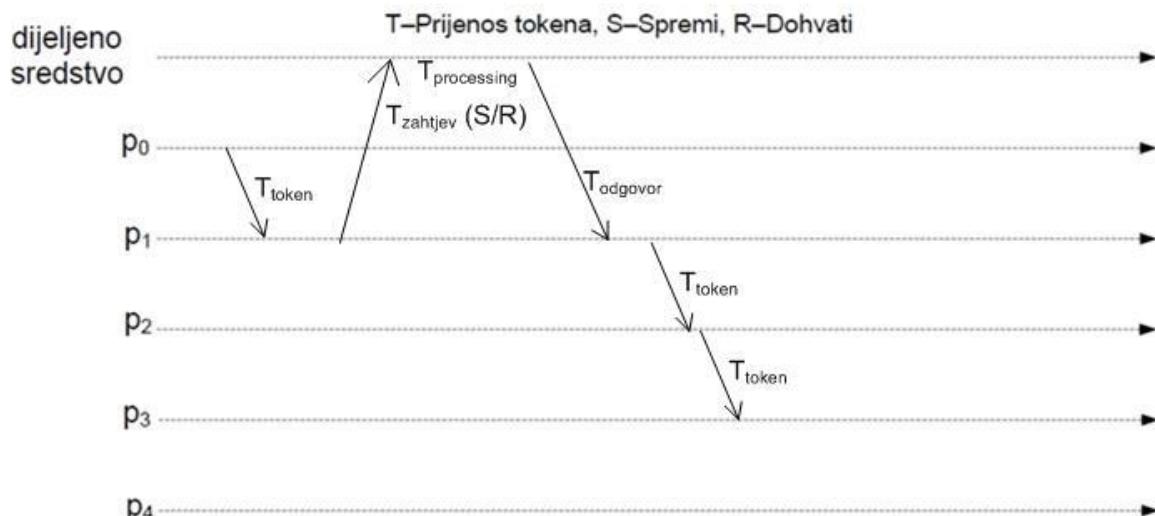
Trenutak II: Računalo 1 prima poruku od računala 2 s oznakom vremena $T_p=44$ koja je veća od lokalne oznake vremena $T_L=28$. Lokalni sat se pomiče na vrijednost $T_p+1=45$.



Trenutak III: Računalo 3 prima poruku od računala 2 s oznakom vremena $T_p=68$ koja je veća od lokalne oznake vremena $T_L=49$. Lokalni sat se pomiče na vrijednost $T_p+1=69$.

7.4. Pet procesa postavljenih na različita računala u raspodijeljenoj okolini ostvaruje međusobno isključivanje primjenom prstena. Vrijeme prijenosa poruke zahtjeva i odgovora pri pristupu dijeljenom sredstvu jednako je 3 ms, vrijeme obrade poruke zahtjeva na sredstvu je 5 ms, vrijeme prijenosa tokena između dva susjedna procesa u prstenu je 2 ms. Kada primi token,

proces može maksimalno jednom ostvariti pristup dijeljenom sredstvu prije nego što proslijedi token idućem susjedu. Naznačite navedena vremena na dijagramu. Koje je minimalno, a koje maksimalno vrijeme čekanja bilo kojeg procesa u prstenu za pristup dijeljenom sredstvu.



Min. vrijeme - U najboljem slučaju, proces koji želi ostvariti pristup čeka $T=0$ sekundi. Naime, taj slučaj nastupa kada proces uđe u stanje u kojem želi ostvariti pristup sredstvu netom prije nego što je primio token.

Max. vrijeme - U najgorem slučaju, proces ulazi u stanje u kojem želi ostvariti pristup sredstvu netom nakon što je proslijedio token svojem susjedu. U tom slučaju, proces mora čekati da svi ostali procesi prime token i ostvare pristup dijeljenom sredstvu. Maksimalno vrijeme čekanja u tom slučaju iznosi $T = 5 * T_T + 4 * (T_z + T_o + T_p) = 10 + 44 = 54$ ms.

8 Konzistentnost i replikacija podataka

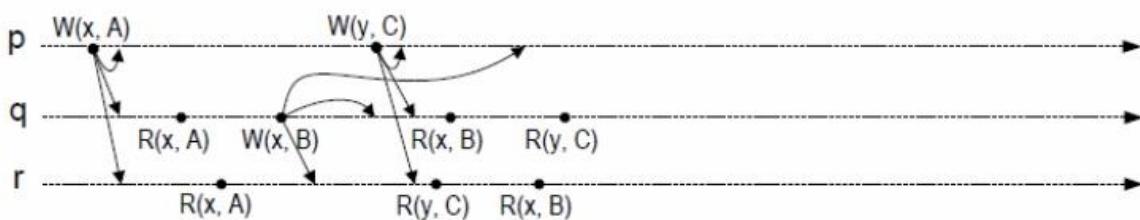
8.1. Objasnite što je replika podatka, a što je nekonzistentnost replike podatka.

Replika podatka je jedna kopija podatkovnog objekta u raspodijeljenoj okolini. Nekonzistentnost replika podataka se javlja kada dvije ili više replika u raspodijeljenoj okolini u nekom trenutku u vremenu se nalaze u različitim stanjima.

8.2. Objasnite što je povezana konzistentnost operacija u raspodijeljenim sustavima? Na primjeru procesa p, q i r prikažite slijed operacija čitanja i pisanja koji je a) u skladu i b) nije u skladu s načelima povezane konzistentnosti.

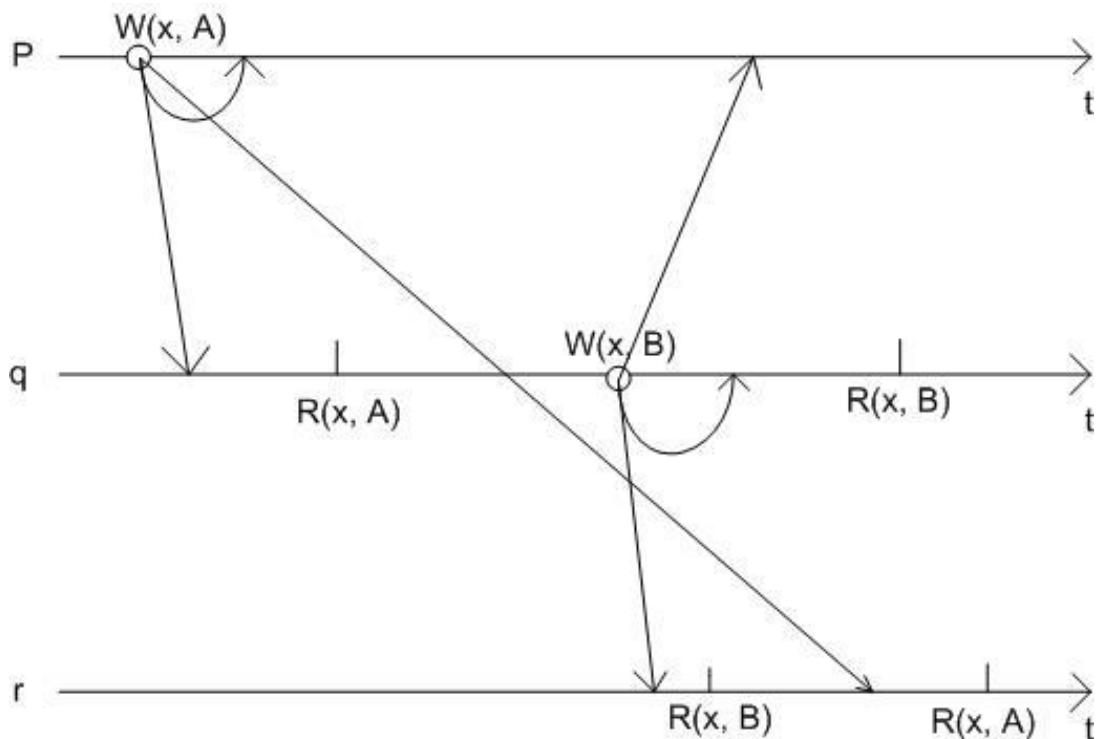
Model povezane konzistentnosti zahtjeva da povezane operacije svi procesi moraju vidjeti u istom redoslijedu. Redoslijed izvođenja povezanih operacija upisivanja vidljiv je svim procesima na jednak način, dok redoslijed izvođenja operacija upisivanja koje nisu povezane, svakom procesu može biti prikazan na drugičiji način.

A)



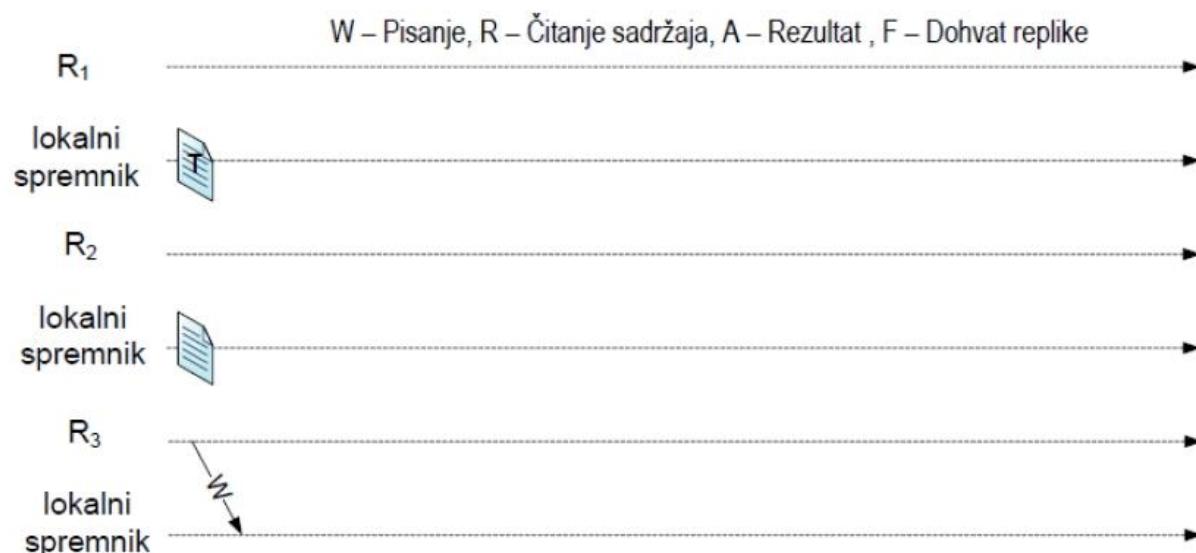
Proces p zapisuje A na mjesto x, zatim proces q čita A s lokacije x i nakon toga na lokaciju x zapisuje B. Vidi se da su operacije $R(x, A)$ i $W(x, B)$ povezane jer očitana vrijednost A može utjecati na izračun vrijednosti B. Operacije $W(x, B)$ i $W(y, C)$ nisu povezane jer upisuju dva različita podatka na dvije različite lokacije. Dakle, svi procesi moraju vidjeti slijed povezanih operacija u istom redoslijedu tj. mora se prvo izvršiti $R(x, A)$ pa onda $W(x, B)$. Vidimo da se na procesu q tako i izvršava, ali vidimo da je taj redoslijed i na procesu r.

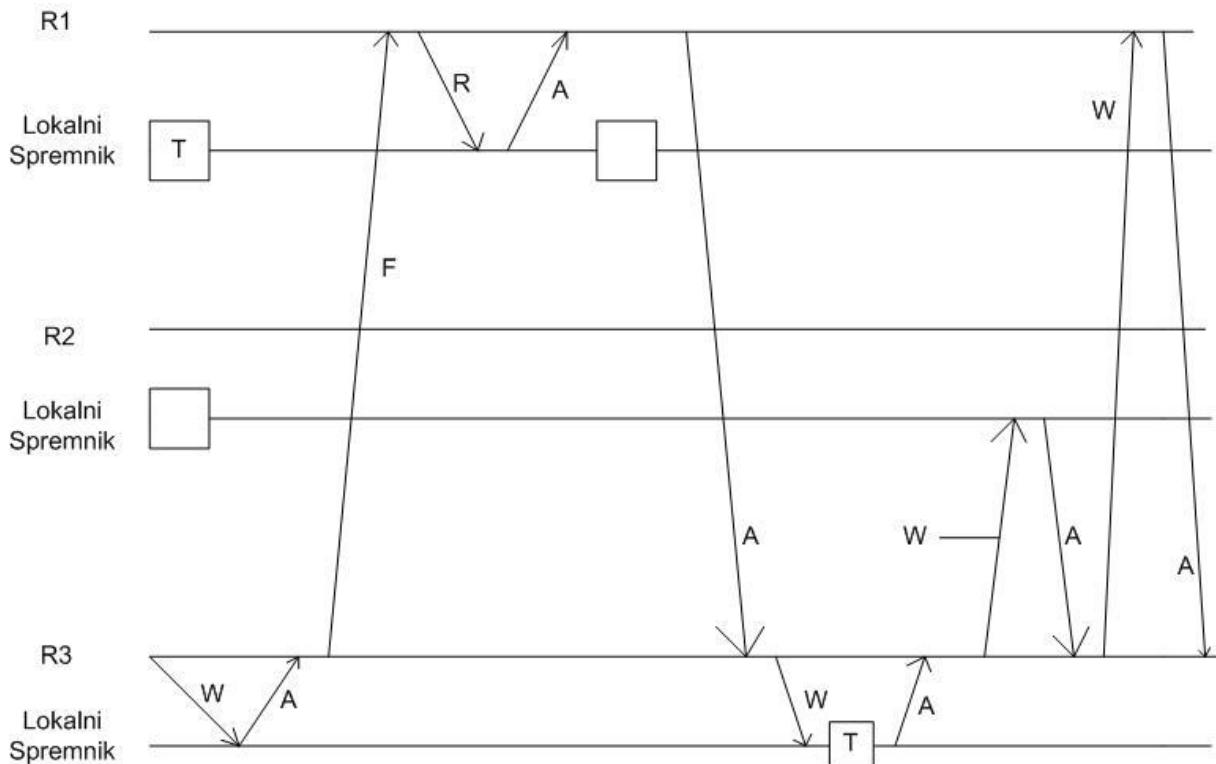
B)



Proces p zapisuje na lokaciju x, podatak A. Zatim proces q čita A s lokacije x i nakon toga zapisuje na lokaciju x podatak B. Vidimo da su operacije $R(x, A)$ i $W(x, B)$ povezane i svaki proces bi trebao vidjeti taj slijed operacija. No, vidimo da proces r prvo zapisuje B na lokaciju x, zatim čita A s lokacije x i zato ovaj primjer ne zadovoljava povezanu konzistentnost.

8.3. Raspodijeljeni sustav uključuje tri računala (R_0, R_1, R_2) s lokalnim spremnicima. U lokalnom spremniku računala R_1 nalazi se trajna replika dokumenta, dok se u lokalnom spremniku računala R_2 nalazi obična replika dokumenta. Korisnik putem računala R_3 provodi operaciju pisanja nad dokumentom primjenom postupka lokalnog obnavljanja stanja replike. Skicirajte i objasnite korake postupka.





Korisnik na računalu R3 želi pristupiti promijeniti saržaj podatka (replike). Prvo pristupa svom lokalnom spremniku da provjeri je li se tamo nalazi podatak. Nakon što utvrdi da se ne nalazi od glavnog poslužitelja zatraži repliku. Glavni poslužitelj (R1) dohvati repliku iz svog lokalnog spremnika i proslijedi ju korisniku. Korisnik lokalno obavlja izmjenu replike i on sam postaje glavni poslužitelj. Nakon izmjene obavještava ostale poslužitelje s novom verzijom replike.

8.4. U sustavu replika koji se sastoji od glavnog poslužitelja i $n=4$ podjednako opterećena pomoćna poslužitelja, odredite metodu održavanja konzistentnosti replika za koju će prosječno mrežno (prometno) opterećenje poslužitelja L biti najmanje. Pri tome prepostavite da korisnike isključivo poslužuju pomoći poslužitelji, da je prosječna frekvencija upita $f_u=5$ upita/s, prosječna frekvencija promjena $f_p=1$ promjena/min te da su prosječne veličine upita/odgovora, operacija za promjenu sadržaja i replika $I_p=1kB$, $I_o=50 kB$ i $I_r=100 kB$. Usporedite dobivena opterećenja s centraliziranim slučajem kada korisnike poslužuje glavni poslužitelj.

PUSH metoda s proslijeđivanjem obavijesti o promjenama: Nakon svake promjene, glavni poslužitelj šalje svakom pomoćnom poslužitelju obavijest o promjenama. Za prvi sljedeći korisnički zahtjev, svaki od pomoćnih poslužitelja šalje glavnom poslužitelju upit za novom replikom na koji mu glavni poslužitelj odgovara novom verzijom replike.

$$L_1 = n \cdot f_p \cdot I_p + n \cdot f_p \cdot I_p + n \cdot f_p \cdot I_r = n \cdot f_p \cdot (2 \cdot I_p + I_r) = 6,8 kB/s$$

PUSH metoda s proslijeđivanjem operacija za promjenu sadržaja: Nakon svake promjene, glavni poslužitelj šalje svakom pomoćnom poslužitelju obavijest o operacijama koje je potrebno izvršiti nad prethodnom verzijom replike da bi se ona dovela u konzistentno stanje.

$$L_2 = n \cdot f_p \cdot I_o = 3,33 kB/s$$

PUSH metoda s proslijedivanjem cjelokupnog sadržaja replika: Nakon svake promjene, glavni poslužitelj šalje novu verziju replike svakom pomoćnom poslužitelju.

$$L_3 = n \cdot f_p \cdot l_r = 6,67 \text{ kB/s}$$

PULL metoda: Nakon svakog upita, pomoćni poslužitelj provjerava kod glavnog poslužitelja je li došlo do promjene stanja replike koju pohranjuje lokalno. U $1/(n \cdot f_p)$ od $1/f_u$ slučajeva će poslužitelj odgovoriti novom verzijom replike, a u $1/(f_u - n \cdot f_p)$ od $1/f_u$ slučajeva će odgovoriti porukom da nije došlo do promjene.

$$L_4 = f_u \cdot l_p + n \cdot f_p \cdot l_r + (f_u - n \cdot f_p) \cdot l_p = 16,6 \text{ kB/s}$$

Centralizirani slučaj: Glavni poslužitelj odgovara replikom na svaki korisnički upit.

$$L_5 = f_u \cdot (l_p + l_r) = 505 \text{ kB/s}$$

$$\text{Min } \{L_1, L_2, L_3, L_4, L_5\} = L_2.$$

9 Otpornost na neispravnosti

9.1. Objasnite razliku između ispada sustava i neispravnosti u sustavu.

-Ispad sustava je stanje sustava koje se detektira kroz nemogudnost korištenja jedne ili više njegovih usluga. Posljedica je neispravnosti te ukazuje na nju. Neispravnost je nedostatak u programskom kodu, oblikovanju sustava ili komunikacijskom kanalu koji može uzrokovati ispad sustava.

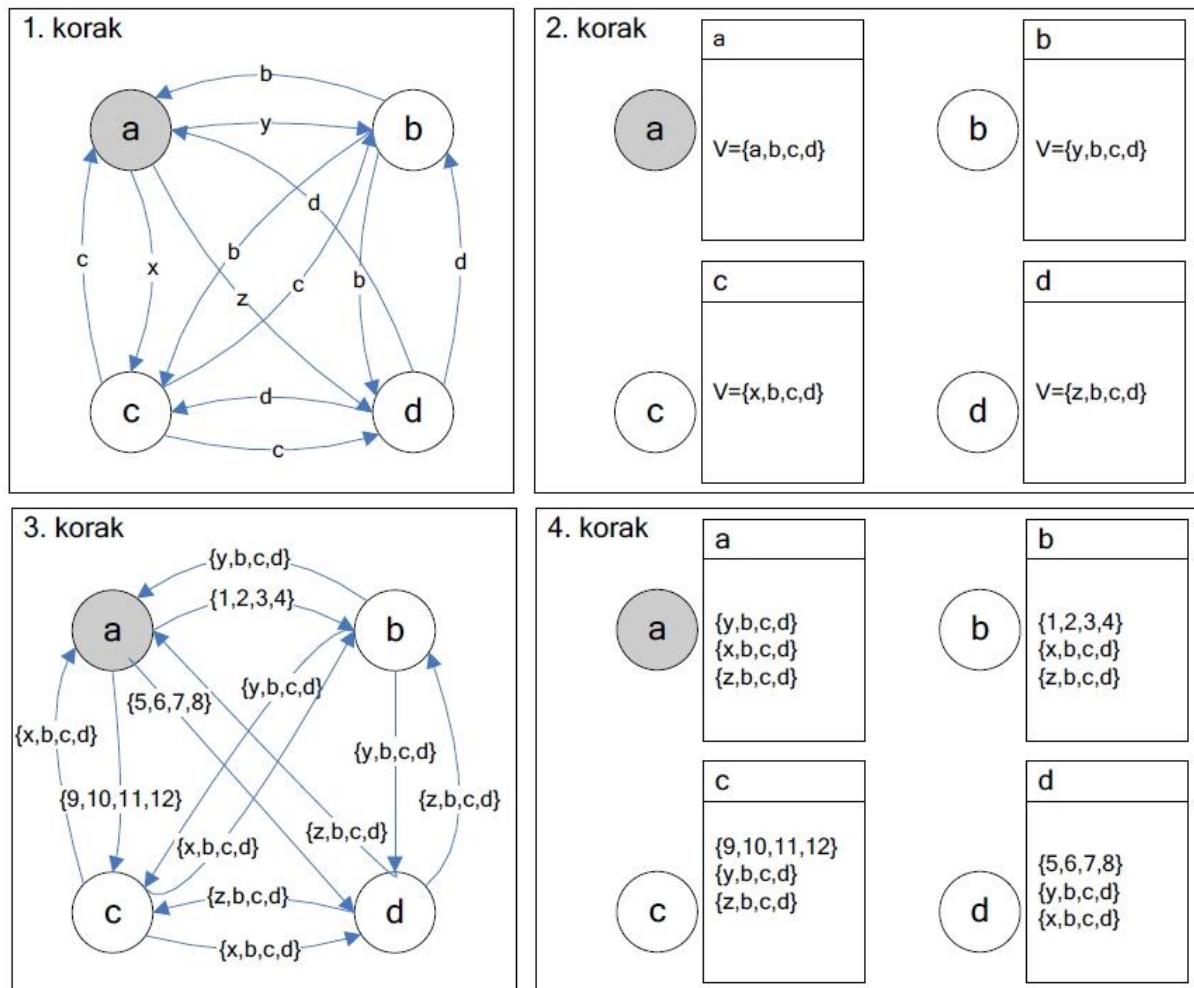
9.2. Prepostavite da grupa procesa treba postići sporazum. U slučaju da su dva procesa grupe u stanju bizantskog ispada, koji je minimalni ukupni broj procesa u grupi za postizanje sporazuma?

$$-N = 3k+1 \text{ gdje je } k \text{ broj procesa u bizantskom ispadu. } N = 3 \cdot 2 + 1 = 7$$

9.3. Objasnite razliku protokola three-phase commit u odnosu na two-phase commit.

-*Three-phase commit* protokol ima jedno stanje više od *two-phase commit* protokola. To stanje je PRECOMMIT. U *two-phase commit* protokolu postoji problem blokirajućeg stanja. Stanje PRECOMMIT riješava taj problem. Koordinator nakon odluke za izvođenje operacije šalje poruku PREPARE_COMMIT na koju procesi odgovaraju s READY_COMMIT. Nakon što primi poruku READY_COMMIT od svih procesa, koordinator šalje GLOBAL_COMMIT. Ako koordinator ispadne, procesi se međusobno mogu dogovoriti koja im je sljedeća akcija i time je riješen problem blokirajućeg stanja.

9.4. U grupi od 4 procesa (p_1, p_2, p_3 i p_4) proces p_1 je neispravan (prepostavite bizantski ispad). Grupa procesa želi postići sporazum o identifikatorima ostalih procesa grupe. U koracima 1 i 3 procesi međusobno razmjenjuju podatke, a u koracima 2 i 4 prikupljaju i analiziraju primljene podatke. Nacrtajte na slici podatke koje procesi razmjenjuju u koracima 1 i 3, a za korake 2 i 4 navedite podatke koje pojedini proces ima na raspolaganju radi donošenja odluke o sporazumu.



10 Vrednovanje performansi raspodijeljenih sustava

10.1 Disk za trajno spremanje podataka ispunjava 50 zahtjeva u sekundi. Srednje vrijeme obrade zahtjeva operacija pisanja i čitanja je 10 ms. Disk ima prosječno 1 zahtjev u repu. Koliko je prosječno vrijeme čekanja na obradu zahtjeva?

Propusnost sustava je $X = 50 \text{ z/s}$.

Srednje vrijeme obrade zahtjeva je $S = 10 \text{ ms/z}$.

Broj zahtjeva u repu je $Q = 1 \text{ z}$.

Vrijeme zadržavanja zahtjeva u sustavu je $R = Q / X = (1 \text{ z}) / (50 \text{ z/s}) = 20 \text{ ms}$.

Vrijeme zadržavanja R uključuje vrijeme čekanja u repu (W) i vrijeme obrade zahtjeva (S): $R = W+S$.

Vrijeme čekanja na obradu je $W = R - S = 20 \text{ ms} - 10 \text{ ms} = 10 \text{ ms}$.

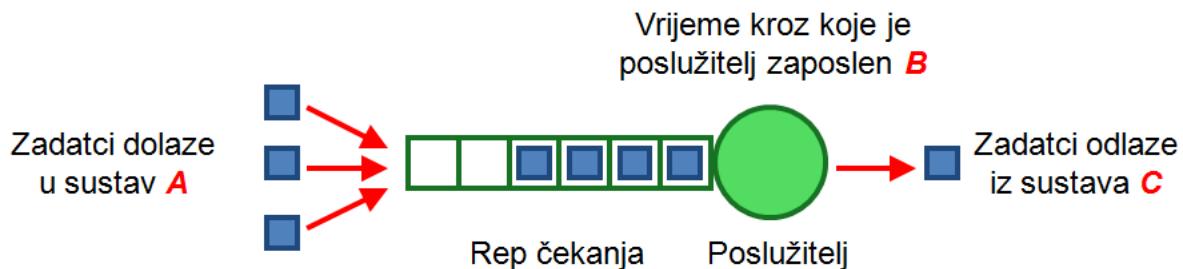
10.2 Web aplikacija uključuje podršku korisnicima putem chat usluge. Kupci sami odabiru jedan od 10 repova čekanja. Mjerenja pokazuju da zahtjevi prosječno dolaze 3 upita u minuti te da svaki kupac prosječno čeka 3 minute u repu i prosječno provodi 2 minute u konverzaciji. Koliko je srednje vrijeme zadržavanja kupaca za zadani sustav?

$$T_w = 3 \text{ min}$$

$$T_s = 2 \text{ min}$$

$$T = T_s + T_w = 5 \text{ min}$$

10.3 Prikažite elemente osnovnog modela repa čekanja. Koje su osnovne veličine, a koje izvedene u modelu repa čekanja? Kako je definirano stacionarno stanje sustava?

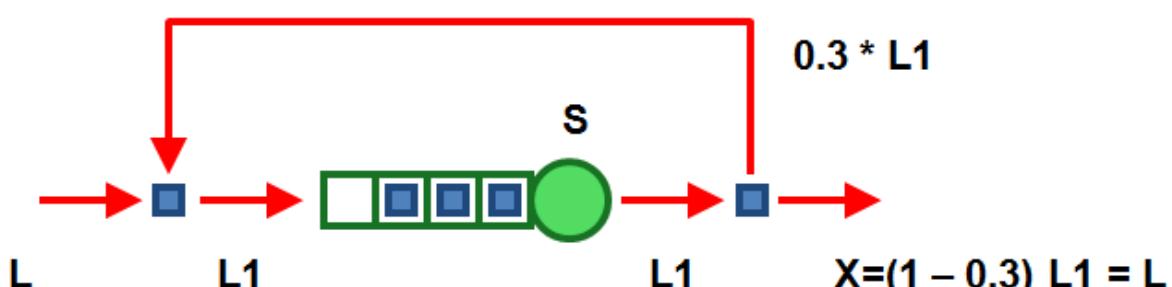


Osnovne veličine modela su vrijeme promatranja (T), broj dolazaka (A), broj odlazaka (C) i vrijeme zaposlenosti poslužitelja (B).

Izvedene veličine modela su ulazni ritam ($L=A/T$), izlazni ritam ($X=C/T$), srednje vrijeme posluživanja ($S=B/C$) i zaposlenost poslužitelja ($U=B/T$).

U stacionarnom stanju sustava je $X = L$.

10.4 Upiti dolaze na poslužitelj s učestalošću od 12 upita u sekundi te zahtijevaju 0,75 sekundi za obradu. Za 30 % paketa dogodi se pogreška pri obradi te se oni moraju ponovno obraditi. Izračunajte koliko vremena paket prosječno provede u sustavu?



$$\text{Broj pristiglih upita u sekundi } L = 0.5 \text{ p/s}$$

$$\text{Prosječno vrijeme obrade upita } S = 0.75 \text{ s/p}$$

$$\text{Vjerovatnost pogreške pri obradi } p = 0.3$$

$$L_1 = p L_1 + L \Rightarrow L_1 = L / (1 - p)$$

$$L_1 = L / (1 - p) = 0.5 / 0.7 = 0.714 \text{ p/s}$$

Prosječna zaposlenost poslužitelja

$$U = L_1 * S = 0.714 \text{ p/s} * 0.75 \text{ s/p} = 0.536 (53.6\%)$$

Srednje vrijeme čekanja u repu

$$W = S * U / (1 - U) = 0.866 \text{ s/p}$$

Srednje vrijeme zadržavanja na poslužitelju

$$R_1 = W + S = 0.866 \text{ s/p} + 0.75 \text{ s/p} = 1.616 \text{ s/p}$$

Prosječno vrijeme zadržavanja u sustavu $R = R_1 / (1 - p) = 2.31$

11 Sustavi s ravnopravnim sudionicima

11.1. Usporedite svojstva centraliziranih i decentraliziranih raspodijeljenih sustava na primjeru.

-Centralizirani raspodijeljeni sustavi temelje se na modelu klijent-poslužitelj. U sustavu postoji koordinator koji prihvata zahteve i raspodjeljuje ih među ostalim procesima u sustavu. Osnovna prednost centraliziranih sustava u smislu performansi sustava je vrlo kratko vrijeme odziva sustava. No, njegova mana je jedinstvena točka ispada sustava (koordinator) i velik broj računala potreban za izvršavanje posla. Primjer takvog sustava je web tražilica. S obzirom da su kolekcije dokumenata prilično velike, potreban je i veliki broj računala za održavanje njihovog indeksa. Npr. za 100 TB tekstualnih dokumenata generira se indeks veličine 25 TB za čije održavanje treba oko 3.000 računala. Stoga je potrebna infrastruktura izrazito složena i skupa, a generira i izrazito visoke troškove održavanja.

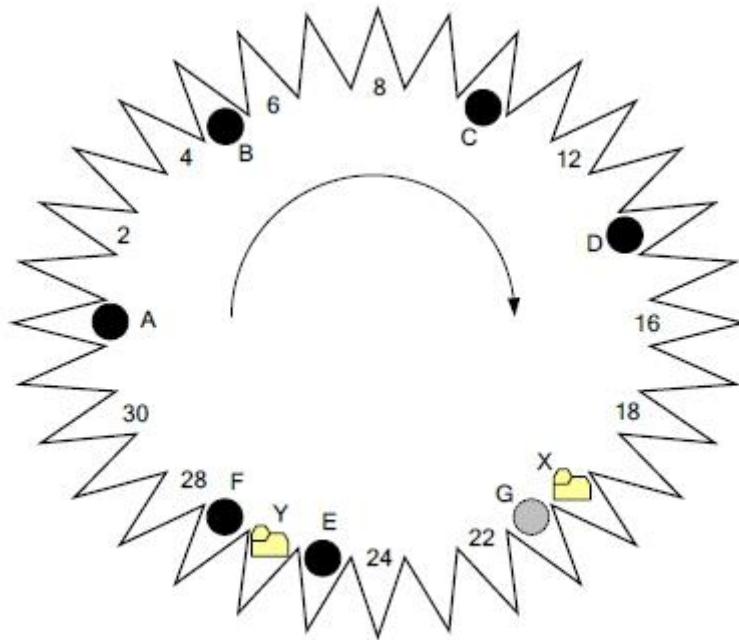
Gnutella je primjer potpuno decentraliziranog sustava koji pretražuje susjedne čvorove s ciljem pronađaska željene datoteke. Sustav se sastoji od peerova koji su međusobno ravnopravni tj. korisnici su ujedno i poslužitelji i klijenti. Svi peerovi sudjeluju u procesu pretraživanja (ne postoji centralizirani indeks), a posebno je pogodno rješenje za pronađenje datoteke koje su replicirane na velikom broju peerova. Ne postoji posebna infrastruktura niti potreba za održavanjem sustava, a time niti jedinstvena točka ispada. Glavni nedostatak ovog rješenja je velika količina generiranog mrežnog prometa pri pretraživanju, a sustav ne može garantirati pronađazak tražene datoteke.

11.2. Kako se izvodi pretraživanje kod strukturiranih, a kako kod nestukturiranih sustava sustava P2P (*peer-to-peer*)? Koji od ovih sustava su skalabilni i zašto?

-U nestukturiranim sustavima P2P, podatak je pohranjen na peeru koji ga kreira, a njegova kopija može biti pohranjena i na nekim drugim peerovima u mreži. Zbog toga se u ovim sustavima pretraživanje izvodi preplavljanjem mreže i slučajnim izborom (*random walk*). Kod strukturiranih sustava P2P, podatak je pohranjen na peeru koji je zadužen za ključ tog podatka. Pretraživanje se provodi traženjem podatka po njegovom ključu. Strukturirani sustavi su skalabilni zato što se kod njih pretraživanje odvija u $\log(n)$ koraka, gdje je n broj peerova u mreži.

11.3. Na slici je prikazana mreža Chord koja se sastoji od 6 čvorova (A, B, C, D, E i F) i koristi prostor identifikatora duljine $N=32$ (dovoljno je $m=5$ bita za kodiranje). Ukoliko je $H_1(A)=0$, $H_1(B)=5$, $H_1(C)=10$, $H_1(D)=14$, $H_1(E)=25$ i $H_1(F)=27$, odgovorite na sljedeća pitanja:

- a) Definirajte tablice usmjeravanja na čvorovima A i F.
- b) Na kojem će se čvoru pohraniti podatak X s ključem $H_2(X)=20$?
- c) Odredite slijed čvorova preko kojih se usmjerava upit od čvora A s ciljem pronađaska podatka Y s ključem $H_2(Y)=26$.
- d) Dodan je novi čvor G ($H_1(G)=21$) u mrežu. Što će se promijeniti u tablici usmjeravanja čvora A?



a)

Čvor A:

$i = 0, 1, 2, 3, 4$ (i ide od 0 do $m-1$)

$(k+2^i)$

$k = H(A) = 0$

$i = 1$

$(0+1) = 1$, nema čvora na jedan. Prvi sljedeći je 5 tj. čvor B.

$(0+2) = 2$, nema čvora na 2. Prvi sljedeći je 5 tj. čvor B.

$(0+4) = 4$, nema čvora na 4. Prvi sljedeći je 5 tj. čvor B.

$(0+8) = 8$, nema čvora na 8. Prvi sljedeći je 10 tj. čvor C.

$(0+16) = 16$, nema čvora na 16. Prvi sljedeći je 25 tj. čvor E.

Tablica usmjerenja čvora A:

0	5
1	5
2	5
3	10
4	25

Čvor F

$i = 0, 1, 2, 3, 4$ (i ide od 0 do $m-1$)

$(k+2^i)$

$k = H(F) = 27$

$i = 1$

$(27+1) = 28$, nema čvora na 28. Prvi sljedeći je 0 tj. čvor A.

$(27+2) = 29$, nema čvora na 28. Prvi sljedeći je 0 tj. čvor A.

$(27+4) = 31$, nema čvora na 31. Prvi sljedeći je 0 tj. čvor A.

$(27+8) = 35 = (\text{aritmetika modulo } 31) = 32 + 3 = 0+3$, nema čvora na 3. Prvi sljedeći je 5 tj. čvor B.

$(27+16) = 43 = 31 + 11$, nema čvora na 11. Prvi sljedeći je 14 tj. čvor D.

Tablica usmjerenja čvora F:

0	0
1	0
2	0
3	5
4	14

b)

$H_2(X) = 20$

Nema čvora na 20, prvi sljedeći je 25. Podatak X će se pohraniti na čvoru E.

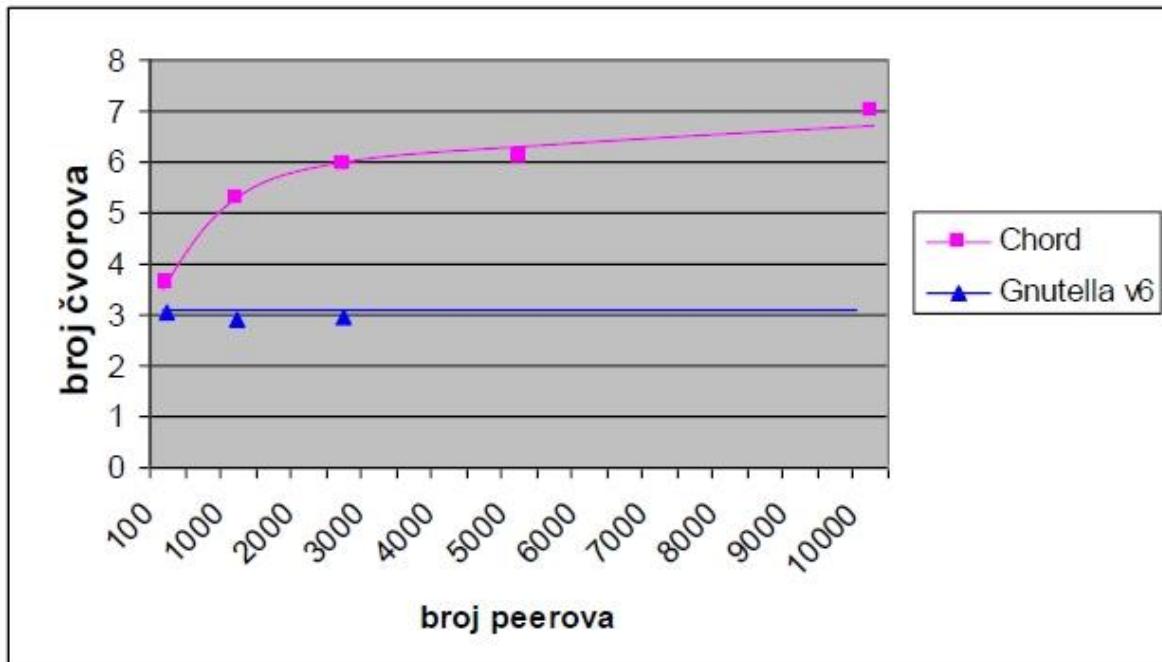
c)

A čvor: $\text{lookup}(26) \rightarrow A$ gleda u tablicu usmjerenja i najveći broj kojeg ima je 25. Proslijeđuje upit njemu. Čvor 25 proslijeđuje upit čvoru zaduženom za 26, a to je njegov sljedbenik 27.

d)

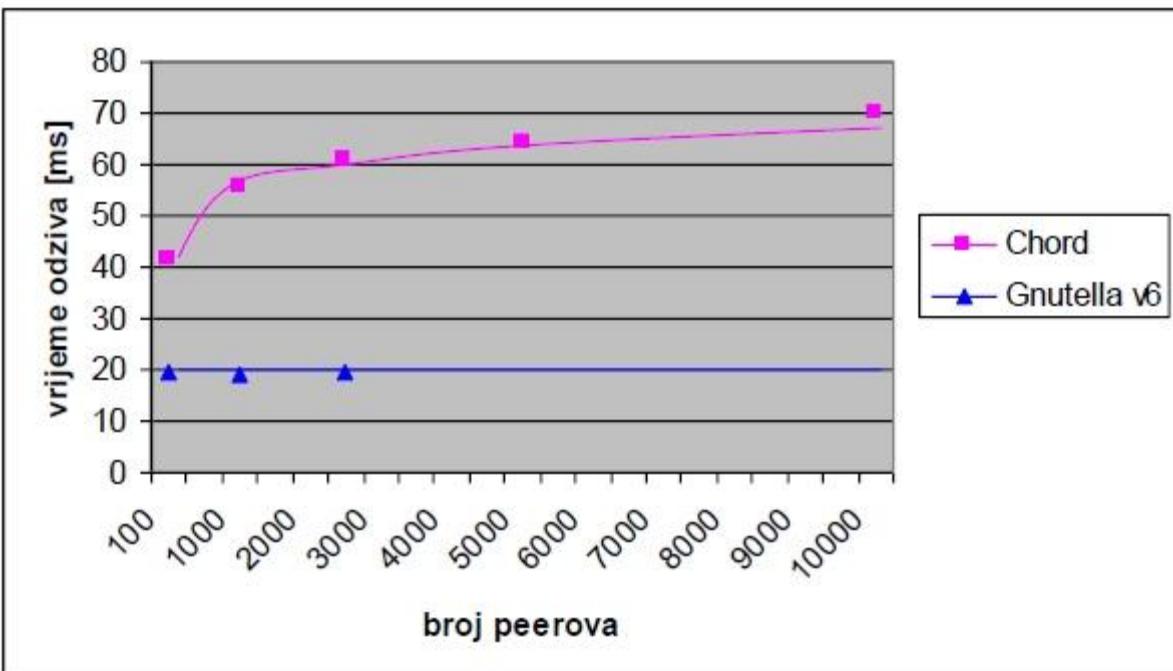
Zadnji redak tablice usmjerenja čvora A će se promijeniti iz 25 u 21.

11.4. Komentirajte rezultate eksperimenta kojim se ispituje svojstvo skalabilnosti protokola Gnutella i Chord u statičnom scenaruju. Kako objašnjavate krivulje kojima se prikazuje prosječan broj čvorova po upitu i prosječno vrijeme odziva?



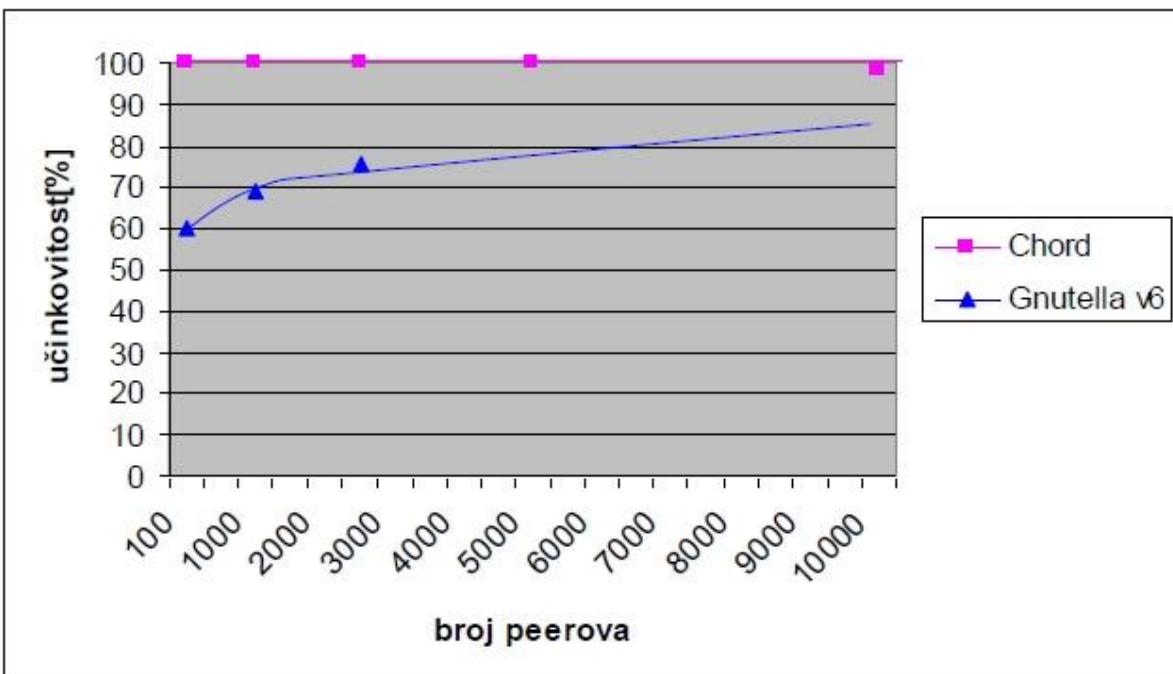
Slika 11.15. Prosječan broj kontaktiranih čvorova po upitu

Vidljivo je da s povećanjem broja peerova u mreži Chord raste broj skokova i vrijeme odziva. Taj porast nije linearan, već se uočava logaritamski porast, što dokazuje da je složenost pretraživanja kod Chorda $O(\log n)$ pri čemu je n broj peerova u mreži. Učinkovitost pretraživanja, tj. postotak uspješno odgovorenih upita je velika što pokazuje slika 11.17.



Slika 11.16. Prosječno vrijeme odziva

Iz slike 11.15. i 11.16. vidljivo je da je kod Gnutelle v6 gotovo jednako vrijeme odziva i broj skokova bez obzira na broj peerova u sustavu, međutim učinkovitost pronaleta podataka raste s povećanjem broja peerova zbog veće povezanosti peerova u mreži što je uočljivo iz slike 11.17.



Slika 11.17. Postotak uspješno odgovorenih upita

12 Grozdovi i spletovi računala

12.1. Skicirajte i ukratko objasnite slojevitu arhitekturu spleta računala.



-**Sloj osnovnih sredstava** upravlja osnovnim funkcionalnostima spleta računala. On implementira lokalne operacije koje su specifične svakom sredstvu po vrsti i implementaciji te omogućuje korištenje tih operacija na višim slojevima.

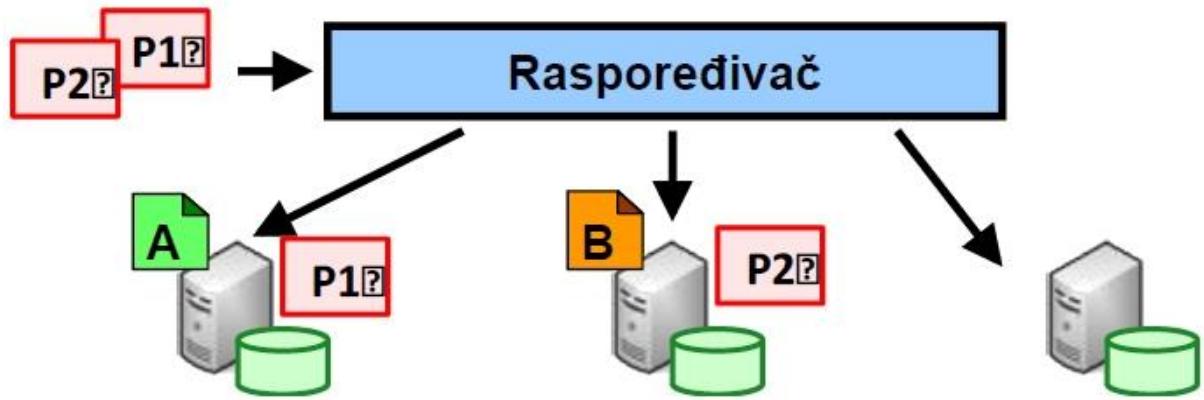
Sloj komunikacijskih protokola definira protokole komunikacije i autentikacije koji su potrebni za transakcije u spletu.

Sloj protokola za sredstva omogućuje korisniku interakciju s udaljenim resursima i uslugama. On definira protokole za sigurno pregovaranje, pokretanje, praćenje, kontrolu, obračun (engl. accounting) i naplatu dijeljenih operacija i individualnih resursa.

Sloj zajedničkih usluga definira protokole i usluge koji su zaduženi za upravljanje grupom sredstava, a ne za pojedino sredstvo.

Na vrhu se nalazi **sloj korisničkih aplikacija**.

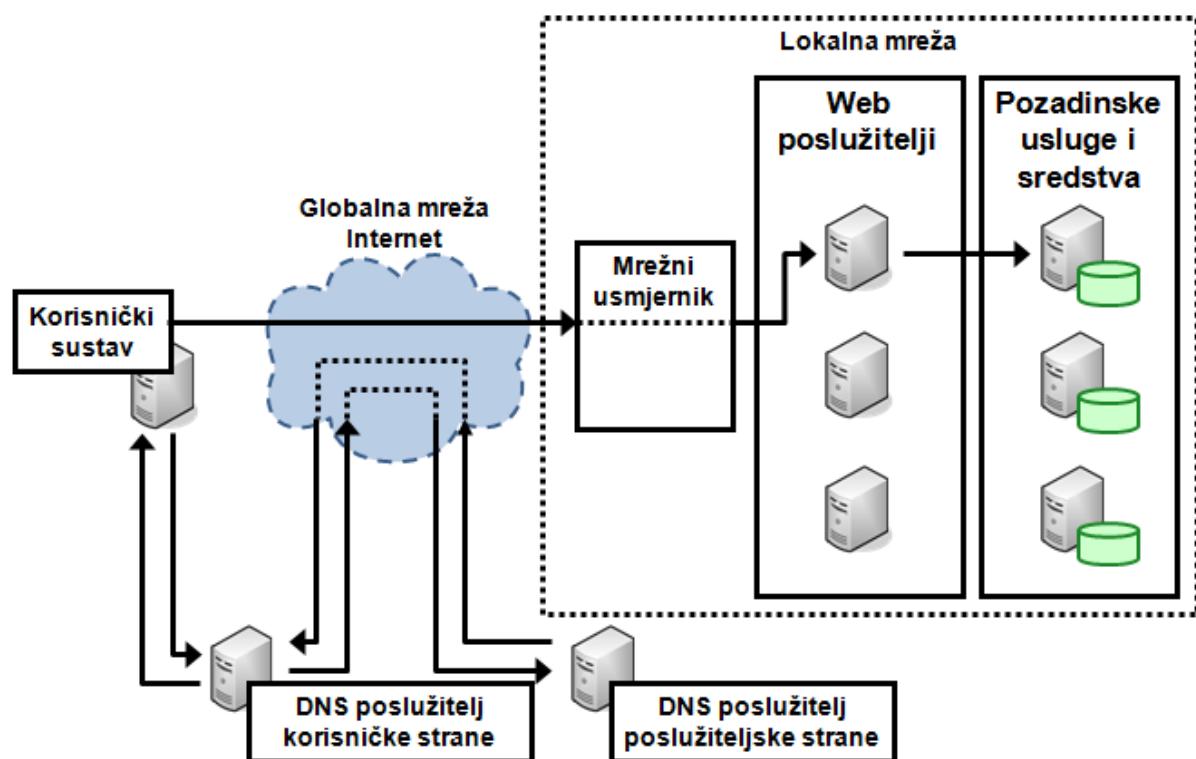
12.2. Na primjeru opišite značajke raspoređivanja zasnovanog na korištenju prostorne lokalnosti.



Slika 12.7. Primjer prostorne lokalnosti

-Kod prostorne lokalnosti, poslovi se raspoređuju na čvorove koji sadrže podatke potrebne za izvođenje posla. Drugim riječima, poslovi se približavaju podacima. Na primjeru sa slike, proces P1 koristi podatke A. Proces P1 se šalje na obradu na čvor na kojem su smješteni podaci A, a proces P2 se šalje na čvor na kojem su smješteni podaci B.

12.3. Prikažite i opišite elemente modela grozda računala.



Korisnički sustav je aplikacija kojom korisnik ostvaruje pristup i koristi sredstva i usluge na grozdu računala.

DNS poslužitelj korisničke strane je poslužitelj pomoću kojeg korisnički sustav razlučuje adrese udaljenih računala na Internetu.

DNS poslužitelj poslužiteljske strane je poslužitelj koji razlučuje adrese poslužitelja u lokalnoj mreži.

Mrežni usmjernik je uređaj koji prihvata, analizira i usmjerava pristigle zahtjeve.

Web poslužitelji i pozadinska sredstva i usluge su osnovni elementi grozda računala.



Diplomski studij

Informacijska i
komunikacijska tehnologija
Telekomunikacije i
informatika

Računarstvo
Računarska znanost
Programsko inženjerstvo i
informacijski sustavi

Raspodijeljeni sustavi

Pitanja za provjeru znanja s odgovorima
1. blok predavanja

Ak.g. 2011./2012.

Napomena:

Preporučena literatura su bilješke s predavanja.

Zadatak Objasnite pojam skalabilnosti raspodijeljenog sustava.

1.1

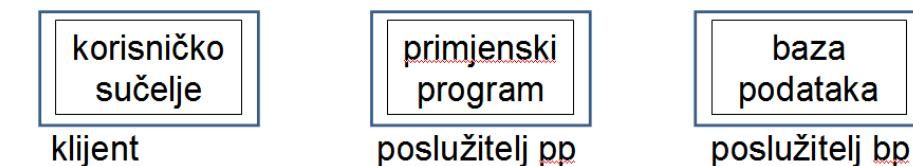
Raspodijeljeni sustav je skalabilan ukoliko posjeduje sposobnost prilagodbe povećanom broju korisnika i sredstava, njihovoj rasprostranjenosti te načinu upravljanja sustavom.

Zadatak Objasnite pojam migracijske transparentnosti raspodijeljenog sustava.

1.2

Za raspodijeljeni sustav kažemo da posjeduje migracijsku transparentnost ukoliko on prikriva promjenu lokacije nekog sredstva na način da ta promjena ne utječe na način pristupa tom sredstvu.

Zadatak Skicirajte trorednu arhitekturu klijent-poslužitelj te na proizvoljnom primjeru aplikacije objasnite ulogu svake razine u cijelokupnoj arhitekturi.



Primjer su aplikacije weba, gdje klijentski program koji se izvodi na klijentskom računalu nikada ne pristupa direktno bazi podataka, već posredno preko aplikacije weba. Klijentski program prikazuje korisničko sučelje i komunicira s aplikacijom weba koja obavlja cijelokupnu logiku usluge i pristupa potrebnim podacima.

Zadatak Objasnite razliku između sinkrone i asinkrone komunikacije.

1.4

Dok je kod sinkrone komunikacije pošiljatelj blokiran nakon slanja poruke sve do primitka potvrde o isporuci, kod asinkrone komunikacije pošiljatelj nije blokiran te nastavlja procesiranje odmah nakon slanja.

Zadatak Navedite obilježja komunikacije *socketom UDP*.

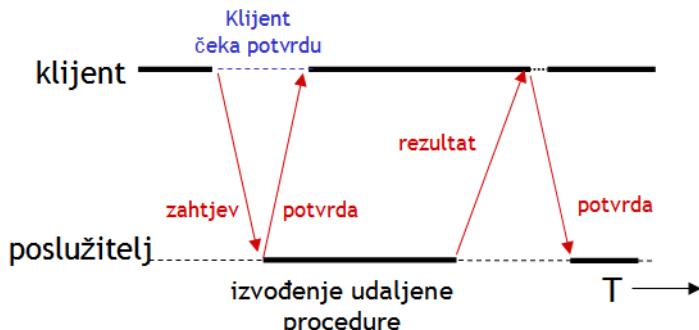
2.1

Ova komunikacija se temelji na modelu klijent-poslužitelj, gdje oba moraju istovremeno biti aktivna da bi se komunikacija mogla ostvariti. Komunikacije je tranzientna i asinkrona, a može se koristiti za implementaciju komunikacije na načelu *pull* ili *push*.

Zadatak U tablicama su prikazane metode na klijentskoj i poslužiteljskoj strani *socketa TCP*.
2.2 Upišite ispravan redoslijed izvođenja metoda u tablice.

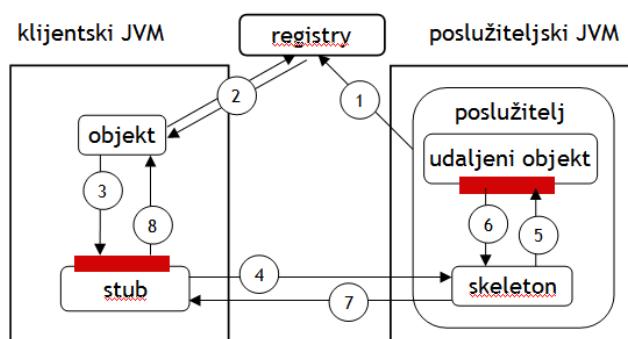
Klijent		Poslužitelj	
1	socket()	3	listen()
3	write()	1	socket()
4	read()	4	accept()
5	close()	6	write()
2	connect()	5	read()
		7	close()
		2	bind()

Zadatak 2.3 Skicirajte tijek komunikacije između klijenta i poslužitelja te objasnite odgođeni sinkroni poziv udaljene procedure RPC (*Remote Procedure Call*).



Kod odgođenog sinkronog poziva udaljene procedure, klijent nije blokiran dok čeka rezultat izvođenja, već nastavlja s radom nakon uspješnog primitka potvrde. Kasnije mu poslužitelj šalje rezultat koristeći drugi asinkroni poziv udaljene procedure.

Zadatak 2.4 Skicirajte model pozivanja udaljene metode Java RMI (*Remote Method Invocation*). Navedite korake u komunikaciji potrebne da bi klijent pozvao metodu dostupnu na poslužitelju, uz pretpostavku da je klasa stub već instalirana na klijentskoj strani



Koraci u komunikaciji su sljedeći:

1. Poslužitelj registrira udaljeni objekt pod odabranim imenom.
2. Klijent od *registrya* traži referencu na udaljeni objekt koristeći registrirano ime.
3. Klijent poziva metodu *stuba* dostupnu na klijentskom računalu.
4. *Stub* serijalizira parametre i šalje ih *skeletonu*.
5. *Skeleton* deserijalizira parametre i poziva metodu udaljenog objekta.
6. Udaljeni objekt vraća rezultat izvođenja metode *skeletonu*.
7. *Skeleton* serijalizira rezultat i šalje ga *stubu*.
8. *Stub* deserijalizira rezultat i dostavlja ga klijentu.

Zadatak 3.1 Skicirajte i objasnite primjer komunikacije porukama između dva procesa/objekta (primatelja i pošiljatelja). Kakva je komunikacija porukama s obzirom na vremensku ovisnost primatelja i pošiljatelja?



U komunikaciji između pošiljatelja i primatelja rep sudjeluje kao posrednik. Pošiljatelju se u načelu

garantira isporuka poruke u primateljev rep, ali ne i isporuka poruke primatelju. Primatelj može pročitati poruku iz repa u bilo kojem budućem trenutku. Stoga su pošiljatelj i primatelj poruke vremenski neovisni.

Zadatak 3.2 Objasnite sličnost i razlike u obilježjima komunikacije između dva komunikacijska modela podržana s JMS (*Java Messaging Service*)?

JMS podržava komunikaciju porukama i model objavi-preplati. Obje vrste komunikacije su vremenski neovisne zato što pošiljatelj i primatelj ne moraju istovremeno biti dostupni. Kod komunikacije porukama pošiljatelj mora znati identifikator odredišta, dok je kod modela objavi-preplati komunikacija anonimna. Komunikacija je perzistentna i asinkrona u oba slučaja. Komunikacija se pokreće na načelu *pull* kod komunikacije porukama, a na načelu *push* kod modela objavi-preplati.

Zadatak 3.3 Navedite i objasnite operacije koje implementira programska infrastruktura dijeljenog podatkovnog prostora.

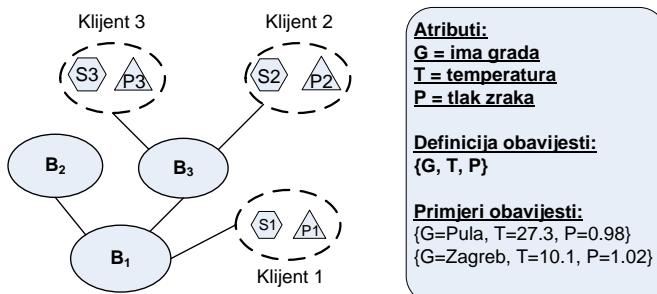
write(t)	-	dodaj tuple t u raspodijeljeni podatkovni prostor
read (s) -> t	-	vraća tuple t koji odgovara predlošku s
take (s) -> t	-	vraća tuple t koji odgovara predlošku s i briše ga iz podatkovnog prostora

Zadatak 3.3 Objasnite opći format poruka protokola HTTP. Navedite kako glasi potpun i apsolutan URI koji identificira resurs zatražen u zahtjevu, ako prva 2 retka HTTP zahtjeva sadrže sljedeće podatke:

```
GET /predmet/rassus HTTP/1.1
Host: www.fer.hr
```

Opći format poruka protokola HTTP sastoji se od početnog retka, polja zaglavja te tijela poruke. Potpun i apsolutan URL je <http://www.fer.hr/predmet/rassus>.

Zadatak 3.4 Raspodijeljeni sustav objavi-preplati, u kojem se koristi **algoritam preplavljivanja obavijestima**, sastoji se od 3 posrednika i 3 klijenta kako je prikazano slikom. Svaki klijent u sustavu ima ulogu pretplatnika i objavljevica. Odgovorite na sljedeća pitanja:



- U trenutku t1 **klijent 1** generira pretplatu $s1=\{G=Zagreb, T<15.5, P>0.98\}$. Napišite oznake svih posrednika na kojima se pohranjuje ova pretplata. Pretplata se pohranjuje na posredniku B1.
- U trenutku t2>t1 **klijent 2** generira pretplatu $s2=s1$. Napišite oznake svih posrednika na kojima se pohranjuje ova pretplata. Pretplata se pohranjuje na posredniku B3.
- U trenutku t3>t2 **klijent 3** generira obavijest $p1=\{G=Zagreb, T=-2.2, P=1.01\}$. Objasnite točan redoslijed kojim će se ova obavijest proširiti sustavom i biti isporučena zainteresiranim klijentima.
 $P3 \rightarrow B3 \rightarrow B1 \rightarrow B2$

Zadatak Korisnik nakon ispunjavanja obrasca na Web-u odabire opciju Submit, čime pošalje podatke Web-poslužitelju na adresu www.tel.fer.hr/obrazac/accept korištenjem protokola HTTP verzije 1.1. Kojim se HTTP zahtjevom šalju podaci poslužitelju i kako je definiran prvi redak zahtjeva?

Podaci se šalju zahtjevom POST. Prvi redak je definiran na sljedeći način:
POST /obrazac/accept HTTP 1.1 .

Zadatak Objasnite opći format poruka protokola HTTP. Navedite kako glasi potpun i apsolutan URI koji identificira resurs zatražen u zahtjevu, ako prva 2 retka HTTP zahtjeva sadrže sljedeće podatke:

```
GET /predmet/rassus HTTP/1.1
Host: www.fer.hr
```

Opći format poruka protokola HTTP sastoji se od početnog retka, polja zaglavla te tijela poruke. Potpun i apsolutan URI je <http://www.fer.hr/predmet/rassus>.

Zadatak Pretpostavite da se sjedište weba sastoji od 2 poslužitelja priključena na Internet preko posrednika (*proxy*). Navedite i objasnite svojstva ovog raspodijeljenog sustava.

Ovaj raspodijeljeni sustav karakteriziraju:

- 1) replikacijska transparentnost – zato što korisnik nije svjestan koji poslužitelj ga je zapravo poslužio,
- 2) otpornost na kvarove – jer se kvar jednog poslužitelja može prikriti od korisnika i
- 3) skalabilnost – zato što ovaj sustav posjeduje sposobnost prilagodbe povećanom broju korisnika.

Ovaj sustav podržava i lokacijsku i migracijsku transparentnost (putem sustava DNS).

Zadatak Objasnite razliku između web-aplikacija temeljenih na CGI (Common Gateway Interface) i poslužiteljskim skriptama.

CGI (*Common Gateway Interface*) je jednostavno sučelje za pokretanje eksternih programa iz web-poslužitelja na platformski i programski neovisan način. Kod svakog zahtjeva se pokreće novi proces, a podaci između poslužitelja i procesa šalju se preko varijabli okoline i tokova podataka. Nakon svake obrade proces se gasi. Nedostatak CGI-a je što se kod svakog zahtjeva pokreće novi proces i nakon obrade gasi što je zahtjevno za resurse (procesorsko vrijeme i memorija) pa kod velikog broja zahtjeva na poslužitelju to znatno utječe na performanse.

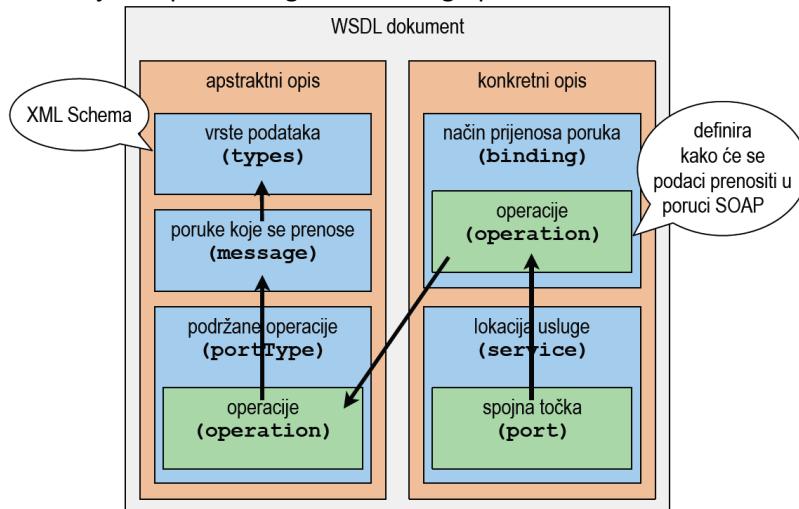
Poslužiteljske skripte (engl. *server side scripts*), dinamički generiraju HTML-dokumente poput CGI-ja, ali je razlika u tome što se za svaki zahtjev ne pokreće novi proces i na taj način se štede resursi.

Zadatak Navedite dva osnovna načina rada protokola SOAP i objasnite kako se poruka SOAP šalje pomoću protokola HTTP.

Dva osnovna načina rada koje podržava protokol SOAP su poziv udaljene procedure te razmjena dokumenata i poruka. Poruka SOAP, koja je pisana jezikom XML, se sastoji od zaglavljia i tijela. Prilikom slanja poruke SOAP protokolom HTTP, i zaglavljie i tijelo poruke SOAP se nalaze u tijelu poruke HTTP.

Zadatak 5.2 Objasnite i skicirajte sadržaj apstraktnog i konkretnog opisa u strukturi dokumenta WSDL.

Dokument WSDL se sastoji od apstraktnog i konkretnog opisa kao što se vidi na slici.



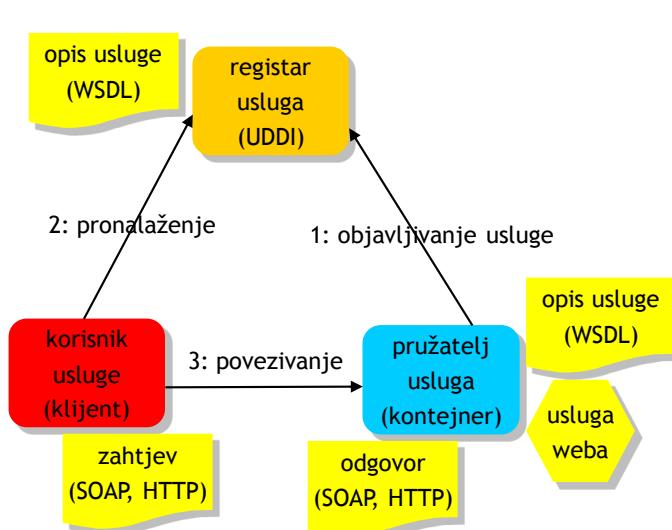
Apstraktan opis u WSDL-u se sastoji od 4 elementa:

1. **types:** definira vrste podataka neovisne o platformi i jeziku (koristi se XML Schema),
2. **message:** definiraju ulazne i izlazne poruke koje se mogu koristiti kao parametri usluge,
3. **operation:** predstavlja jednu operaciju/metodu/proceduru koja je definirana u usluzi, a sastoji se od definicija ulaznih, izlaznih i iznimnih poruka koje se mogu razmjenjivati korištenjem ove operacije i
4. **portType:** koristi poruke (pod 2) da bi opisao sve operacije koje pruža usluga.

Konkretni opis se sastoji od 2 dijela:

1. **binding:** definira kako je konkretna implementacija povezana s operacijama u apstraktnom opisu i definira format u kojem će se poruke prenositi (protokol i elemente) i
2. **service:** definira URI gdje je usluga isporučena tj. na kojoj adresi se može pozvati usluga (taj URI je definiran u spojnoj točci).

Zadatak 5.3 Prikažite arhitekturu i objasnite korištenje usluge Weba.



U arhitekturi usluge Weba postoje 3 uloge: pružatelj usluge, registar usluga i korisnik usluge. Pružatelj usluge je vlasnik usluge Weba i zadužen je za njen smještaj. Klijent je stranka zainteresirana za uslugu weba, a registar usluga omogućava pretraživanje registriranih usluga weba po njihovim opisima.

Tipično korištenje usluge Weba sastoji se od tri operacije: objavljivanje, pronalaženje i povezivanje. Pružatelj usluge registrira uslugu i njen opis (WSDL) objavljivanjem usluge u registru usluga. Korisnik usluge putem registra usluga pronalazi traženu uslugu Weba. U zadnjem koraku korisnik usluge poziva metodu usluge Weba i dobiva rezultat njenog izvođenja putem poruka SOAP i protokola HTTP.

Zadatak Objasnite svojstvo slabe povezanosti usluga kod uslužno orientirane arhitekture.

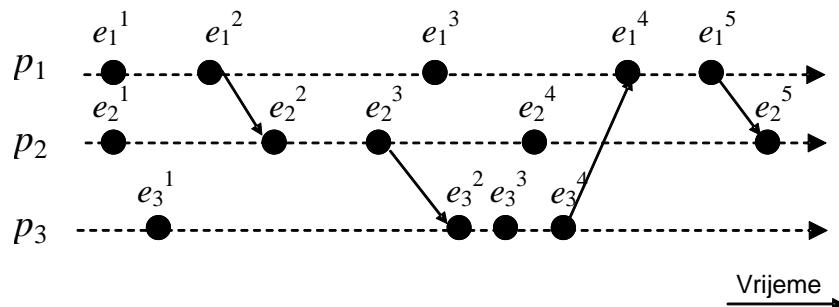
5.4

Svojstvo slabe povezanosti usluga kod uslužno orientirane arhitekture (SOA – Service Oriented Architecture) odnosi se na dizajn programske izvedbe usluga. U sustavu SOA, usluge trebaju biti izvedene tako da promjena u jednoj usluzi ne zahtijeva promjenu neke druge usluge. Pri tome, svaka usluga može i dalje nesmetano koristiti neku drugu uslugu. Npr. algoritam koji se koristi u usluzi se može promijeniti bez znanja drugih usluga i druge usluge ju mogu nesmetano koristiti. Bitno je da se sučelja opisana WSDL-om ne promijene.

Zadatak Objasnite za koje je od sljedeća tri svojstva raspodijeljenih sustava značajna komunikacijska složenost algoritama: a) replikacijska transparentnost b) skalabilnost c) otvorenost.

Komunikacijska složenost algoritma je važna za skalabilnost raspodijeljenog sustava jer na temelju komunikacijske složenosti možemo zaključiti kako raste generirani promet raspodijeljenog sustava s rastom tog sustava. Primjer: komunikacija grupe procesa.

Zadatak Na temelju primjera procesa sa slike **objasnite** jesu li sljedeći parovi događaja uzročno povezani ili nisu? a) e_1^3 i e_2^2 i b) e_2^2 i e_1^5 .



- a) Događaji e_1^3 i e_2^2 su neovisni zato što nisu slijedni događaji na istom procesu, između njih ne postoji slanje i primanje poruke te između njih ne postoji tranzitivna uzročnost.
- b) Između događaja e_2^2 i e_1^5 postoji uzročna povezanost preko tranzitivne uzročnosti $e_2^2 \rightarrow e_2^3 \wedge e_2^3 \rightarrow e_3^2 \wedge e_3^2 \rightarrow e_3^3 \wedge e_3^3 \rightarrow e_3^4 \wedge e_3^4 \rightarrow e_1^4 \wedge e_1^4 \rightarrow e_1^5$.

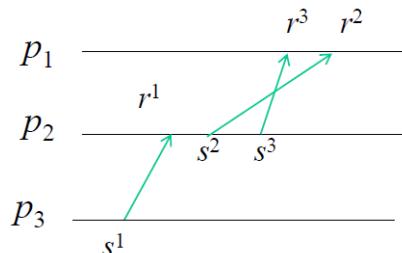
Zadatak Objasnite model komunikacijskog kanala koji se temelji na uzročnoj slijednosti.

6.3

Uzročna slijednost (causal ordering) osigurava da uzročno povezani događaji slanja dviju poruka istom primatelju rezultiraju primanjem u slijedu kojim su poslati.

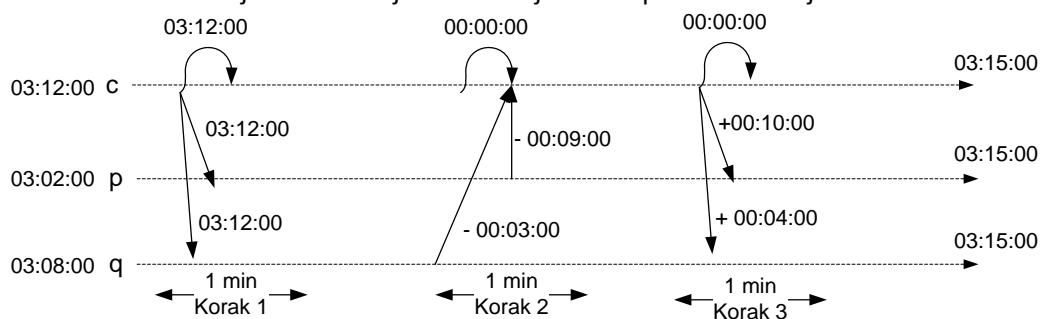
Zadatak Objasnite zašto za sljedeći primjer vrijedi CO ili vrijedi non-CO?

6.4



Za primjer vrijedi non-CO zato što proces p_1 prima poruke od p_2 drugačijim redoslijedom od redoslijeda slanja, a pri tome je slanje poruke 3 uvjetovano slanjem poruke 2.

Zadatak Prikažite i objasnите korake algoritma Berkeley za usklađivanje satnih mehanizama tri računala u raspodijeljenoj okolini. Računala imaju sljedeće vrijednosti satova $T(p)=03:02:00$, $T(q)=03:08:00$ i $T(c)=03:12:00$. Upravitelj je treće računalo. Pretpostavite da prijenos poruke između 2 računala traje 1 minuta i da upravitelj koristi svoje lokalno vrijeme kao zajedničko pri usklađivanju satnih mehanizama.



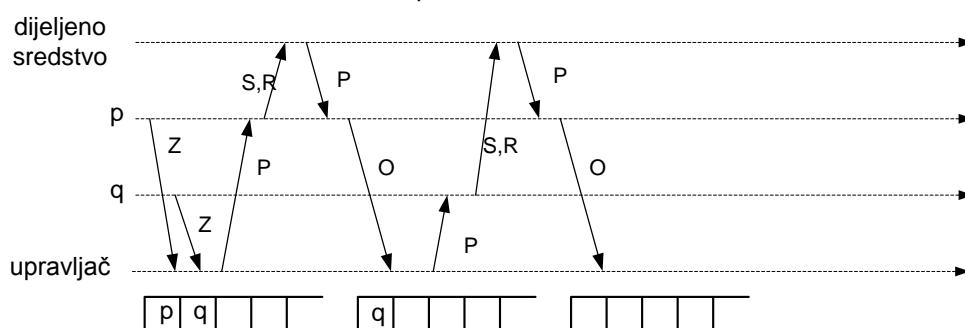
Korak 1:
Upravitelj šalje poruku s trenutnim vremenom svim računalima.

Korak 2:
Poslane poruke putuju 1 minuto i nakon primitka poruka, računala odgovaraju s porukom koja sadrži razliku lokalnog vremena u odnosu na primljeno vrijeme.

Korak 3:
Nakon primitka poruka odgovora, upravitelj šalje poruke zahtjeva koje sadrže vremenski pomak za svako računalo. Poruke zahtjeva putuju 1 minuto te nakon primitka poruke zahtjeva, svako računalo usklađuje lokalni satni mehanizam.

Zadatak Opišite postupak međusobnog isključivanja dvaju procesa (p i q) primjenom središnjeg upravljača s repom čekanja tako da nacrtate redoslijed operacija i objasnite ih. Nakon zauzimanja dijeljenog spremnika, proces provodi jednu operaciju čitanja ili pisanja nad dijeljenim spremnikom.

R –Dohvati, S –Spremi, Z –Zauzmi, P –Potvrda, O –Oslobodi

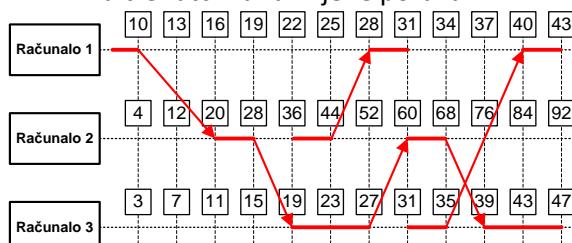


- | | |
|---|---|
| Proces p šalje zahtjev za zauzimanje sredstva, zahtjev se sprema u rep | Proces p provodi operaciju pisanja |
| Proces q šalje zahtjev za zauzimanje sredstva, zahtjev se stavlja u rep | Proces p prima potvrdu |
| Kako je zahtjev od RO stigao prije, upravljач šalje potvrdu RO i uklanja njegov zahtjev iz repa | Proces p šalje poruku Upravljaču i otpušta pristup |
| | Upravljač šalje poruku dojave procesu q te mu dodjeljuje pristup dijeljenom spremniku. Iz repa zahtjeva uklanja se zahtjev od procesa p |
| | Proces q provodi operaciju pisanja |
| | Proces q prima potvrdu |
| | Proces q šalje poruku Upravljaču i otpušta pristup |

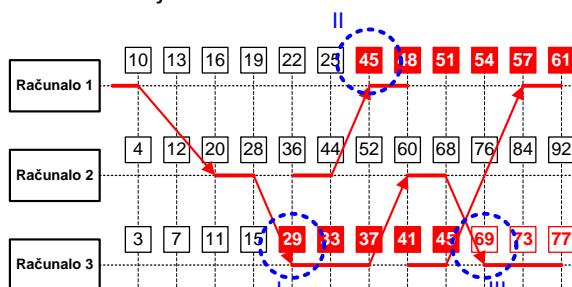
Zadatak
7.3

Za slijed razmjene poruka između tri računala prikazan na slici uspostavite globalni tijek vremena primjenom skalarnih oznaka logičkog vremena. Navedite i opišite trenutke u kojima se ostvaruje korekcija lokalnih satnih mehanizama.

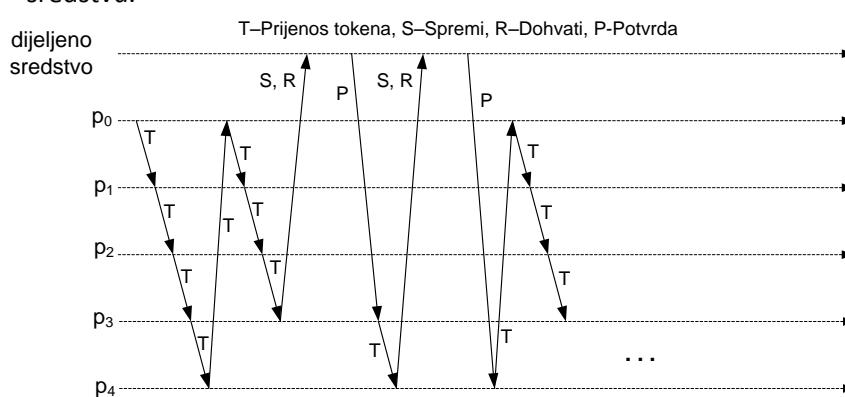
- Određivanje vrijednosti satnih mehanizama u trenutcima razmjene poruka



- Primjena skalarnih oznaka vremena


Zadatak
7.4

Pet procesa postavljenih na različita računala u raspodijeljenoj okolini ostvaruje međusobno isključivanje primjenom prstena. Vrijeme prijenosa poruke zahtjeva i odgovora pri pristupu dijeljenom sredstvu jednako je 3 ms, vrijeme obrade poruke zahtjeva na sredstvu je 5 ms, vrijeme prijenosa tokena između dva susjedna procesa u prstenu je 2 ms. Kada primi token, proces može maksimalno jednom ostvariti pristup dijeljenom sredstvu prije nego što prosljedi token idućem susjedu. Prikažite naznačite navedena vremena na dijagramu. Koje je minimalno, a koje maksimalno vrijeme čekanja bilo kojeg procesa u prstenu za pristup dijeljenom sredstvu.



Min. vrijeme - U najboljem slučaju, proces koji želi ostvariti pristup čeka $T=0$ sekundi. Naime, taj slučaj nastupa kada proces uđe u stanje u kojem želi ostvariti pristup sredstvu netom prije nego što je primio token. **Max. vrijeme** - U najgorem slučaju, proces ulazi u stanje u kojem želi ostvariti pristup sredstvu netom nakon što je proslijedio token svojem susjedu. U tom slučaju, proces mora čekati da svi ostali procesi prime token i ostvare pristup dijeljenom sredstvu. Maksimalno vrijeme čekanja u tom slučaju iznosi $T = 5 * T_T + 4 * (T_z + T_o + T_p) = 10 + 44 = 54$ ms.



SVEUČILIŠTE U ZAGREBU



Diplomski studij

Računarstvo

Znanost o mrežama

Programsko inženjerstvo i
informacijski sustavi

Računalno inženjerstvo

**Ostali (slobodni izborni
predmet)**

Raspodijeljeni sustavi

1. Raspodijeljene arhitekture programskih
sustava. Centralizirana i decentralizirana
rješenja.

Ak. god. 2022./2023.

Sadržaj predavanja

- **Definicija, obilježja i vrste raspodijeljenih sustava**
- Zahtjevi na raspodijeljene sustave: otvorenost, transparentnost, skalabilnost i kvaliteta usluge
- Arhitektura raspodijeljenih sustava
- Primjeri modela raspodijeljene obrade
- Studijski primjeri:
 - Raspodijeljeni sustav weba
 - Internet stvari

Definicija raspodijeljenog sustava (1)

Andrew S. Tanenbaum:

- “Skup neovisnih računala koji korisniku izgleda kao jedan cjeloviti sustav.”

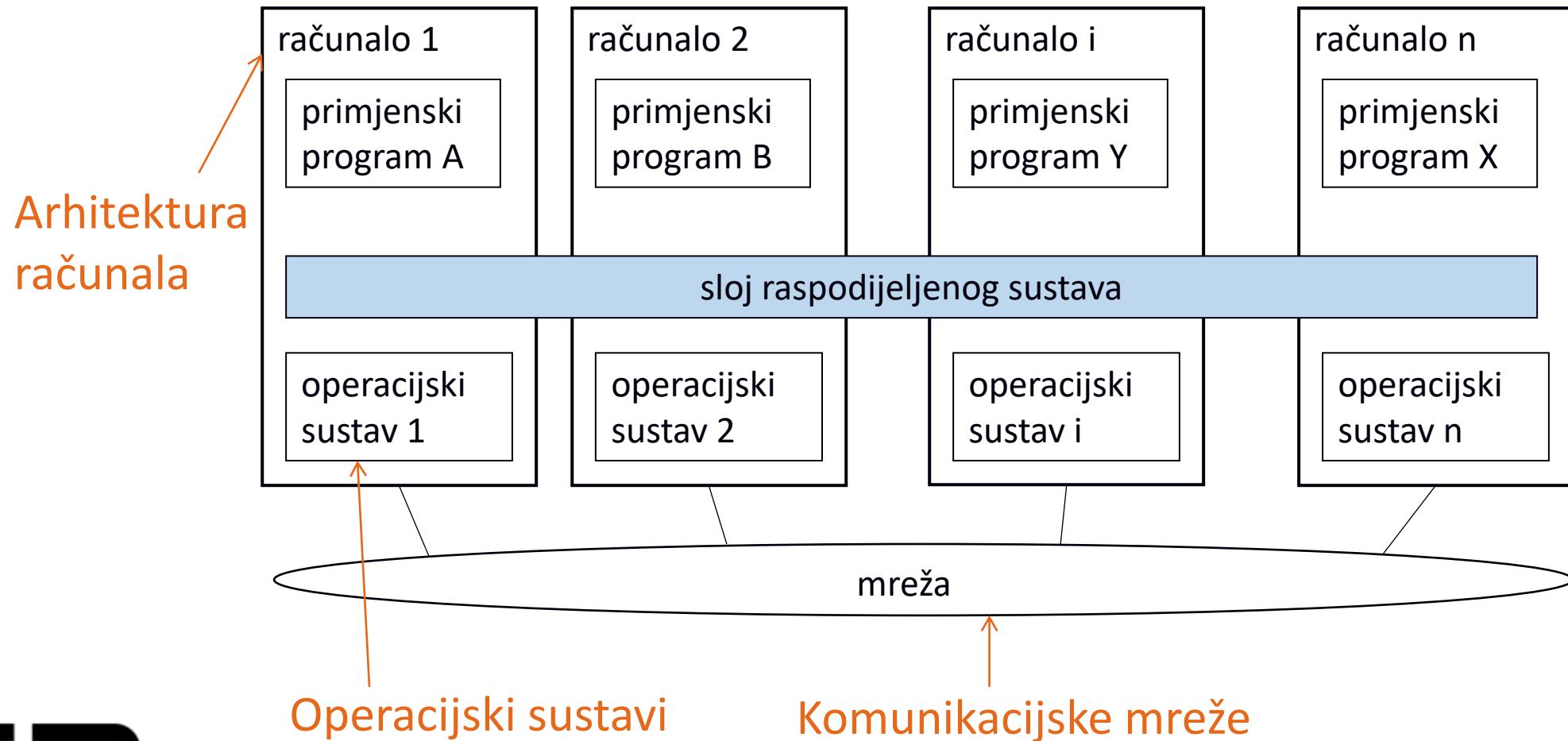
George Coulouris:

- “Sustav u kojem programske i sklopovske komponente umreženih računala komuniciraju i usklađuju svoje aktivnosti isključivo razmjenom poruka.”

Leslie Lamport:

- “Sustav u kojem kvar računala za koje uopće ne znate da postoji može učiniti vaše računalo neupotrebljivim.”

Definicija raspodijeljenog sustava (2)



Sloj raspodijeljenog sustava

Programski posrednički sloj raspodijeljenog sustava, međuoprema
(engl. *middleware*)

- prikriva činjenicu da su procesi i sredstva (resursi) raspodijeljeni na više umreženih računala
- omogućuje povezivanje i suradnju aplikacija, sustava i uređaja
- omogućuje interakciju programa na aplikacijskoj razini

Razlozi za raspodijeljene sustave

Inherentna raspodijeljenost:

- korisnika, uređaja, stvari, informacija, sredstava, ...

Funkcijsko odvajanje:

- različite namjene, različite mogućnosti, različite uloge (korisnik – davatelj usluge, proizvođač – potrošač,)

Opterećenje:

- mogućnost raspodjele i uravnoteženja

Pouzdanost i raspoloživost:

- ostvarene s više komponenata na različitim mjestima

Cijena, troškovi, održavanje...

Vrste raspodijeljenih sustava (1)

Raspodijeljeni računalni sustavi:

- grozd računala (engl. *cluster*)



Računalni klaster Isabella

<http://www.srce.unizg.hr/isabella>

135 računalnih čvorova s 3100 procesorskih jezgri i 12 grafičkih procesora

- splet računala (engl. *grid*)



CRO NGI, Hrvatska nacionalna grid infrastruktura

<https://www.srce.unizg.hr/cro-ngi>

1.868 procesorskih jezgri, 36 grafičkih procesora i 205 TB podatkovnog prostora

- računalni oblak (engl. *cloud*)



<https://www.srce.unizg.hr/hr-zoo>

Vrste raspodijeljenih sustava (2)

Sustavi za pružanje informacijskih i komunikacijskih usluga

- usluge (Skype, Facebook, MS Teams, Gmail, Twitter...)

Raspodijeljeni poslovni i transakcijski sustavi:

- poslovni i transakcijski sustavi (Amazon online shop, Bitcoin, ...)

Internet stvari (engl. *Internet of Things*, IoT)

- povezivanje stvari/uređaja na Internet – fizičkih i virtualnih objekata

Internet svega (engl. *Internet of Everything*, IoE)

- inteligentno povezivanje ljudi, procesa, podataka i stvari

Obilježja raspodijeljenog sustava

Paralelne i konkurentne aktivnosti:

- autonomne komponente sustava istodobno izvode više aktivnosti

Komunikacija razmjenom poruka:

- komponente sustava razmjenjuju podatke porukama, ne dohvaćaju ih iz zajedničke memorije

Dijeljenje sredstava:

- zajedničkim sredstvima pristupa više komponenata sustava

Nema globalnog stanja:

- niti jedan proces ne zna stanje svih ostalih procesa u svim komponentama sustava

Nema globalnog vremenskog takta:

- ograničena mogućnost vremenskog usklađivanja

Temeljni teorijski modeli

Modeli raspodijeljenih sustava sastoje se od procesa koji međusobno komuniciraju i razmjenjuju poruke putem komunikacijskog kanala

Temeljni formalizmi:

- **Komunikacijski i interakcijski model:** procesi, komunikacija, vremenska usklađenost odvijanja i komunikacije procesa
- **Model kvara:** kvarovi i njihov utjecaj na odvijanje i komunikaciju procesa
- Model sigurnosti: prijetnje odvijanju i komunikaciji procesa te mjere zaštite (ne obrađuje se u okviru predmeta Raspodijeljeni sustavi)

Sadržaj predavanja

- Definicija, obilježja i vrste raspodijeljenih sustava
- **Zahtjevi na raspodijeljene sustave: otvorenost, transparentnost, skalabilnost i kvaliteta usluge**
- Arhitektura raspodijeljenih sustava
- Primjeri modela raspodijeljene obrade
- Studijski primjeri: raspodijeljeni sustav weba, Internet stvari

Zahtjevi na raspodijeljene sustave

- **Otvorenost**
 - otvoreni sustav (engl. *open system*): pruža usluge sukladno normiranim pravilima te definiranoj sintaksi i semantici
- **Transparentnost**
 - prikrivanje odabranih značajki raspodijeljenog sustava
- **Skalabilnost**
 - sposobnost razmjerne prilagodbe veličini (broj korisnika – količina sredstva), rasprostranjenosti (lokalno, regionalno, globalno, ...) i načinu upravljanja (jedna ili više administrativnih domena)
- **Kvaliteta usluge**
 - performance (npr. vrijeme odziva), raspoloživost/pouzdanost, trošak

Otvorenost

Norma ili standard je specifikacija koja je:

- široko prihvaćena u industriji (*de facto standard*) ili zastupana od normizacijskog tijela (*de jure standard*),
- dobro definirana,
- neutralna, tj. vlasnički neovisna i
- javno dostupna.

Otvorenost je preduvjet za:

- međudjelovanje (engl. *interoperability*)
- prenosivost (engl. *portability*)
- proširljivost (engl. *extensibility*)

Transparentnost (1)

Transparentnost pristupa (engl. *access transparency*)

- prikrivanje razlika u pristupu sredstvima i predočavanju podataka (različite arhitekture računala, različiti operacijski sustavi, različite baze podataka, ...)

Lokacijska transparentnost (engl. *location transparency*)

- prikrivanje lokacije sredstva: položaj sredstva u sustavu ne treba biti i nije poznat korisniku
- primjer: poslužitelj www.fer.unizg.hr čiju lokaciju (IP-adresu) zna DNS-poslužitelj

Transparentnost (2)

Migracijska transparentnost (engl. *migration transparency*)

- prikrivanje promjene lokacije: promjena lokacije sredstva ne utječe na način pristupa sredstvu

Relokacijska transparentnost (engl. *relocation transparency*)

- prikrivanje premještanja sredstva tijekom njegove uporabe: sredstvu se može pristupiti i može se upotrebljavati tijekom njegove relokacije, tj. premještanja

Replikacijska transparentnost (engl. *replication transparency*)

- prikrivanje više istovrsnih sredstava ili više preslika nekog sredstva (sve replike nude istu funkcionalnost)

Transparentnost (3)

Konkurencijska transparentnost (engl. *concurrency transparency*)

- prikrivanje istodobne uporabe istog resursa od strane više korisnika: zajednička/dijeljena uporaba sredstva uz očuvanje konzistentnosti

Transparentnost na kvar (engl. *failure transparency*)

- prikrivanje kvara: otkrivanje kvara i obnavljanje sustava nakon kvara nije uočljivo korisnicima
- problem otkrivanja kvara: veliko opterećenje može se očitovati kao kvar (npr. nema odgovora u očekivanom vremenu)

Skalabilnost (1)

Kako bi se uz promjenu broja korisnika održale performance sustava uz prihvatljive troškove treba osigurati:

- više (istovrsnih) dijelova koliko?
- prostorno raspodijeljenih gdje?
- koji komuniciraju kako?

Primjeri neskalabilnih rješenja

- centralizirana usluga: jedan poslužitelj za sve korisnike
- centralizirani podaci: jedan poslužitelj sa svim korisničkim podacima
- centralizirani algoritam: svi podaci o sustavu poznati na glavnom procesu

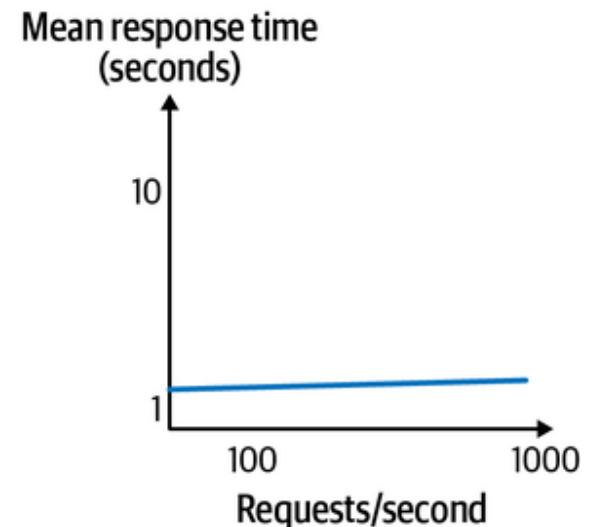
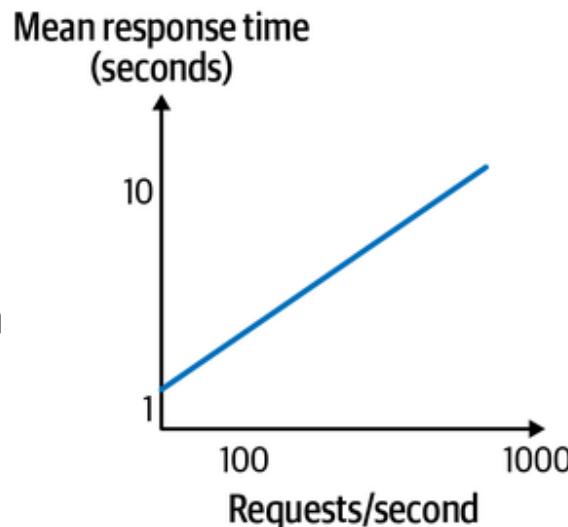
Mogu li se centralizirana rješenja nositi s povećanim brojem korisnika?

- Za motivaciju pogledati <https://www.internetlivestats.com/>

Skalabilnost (2)

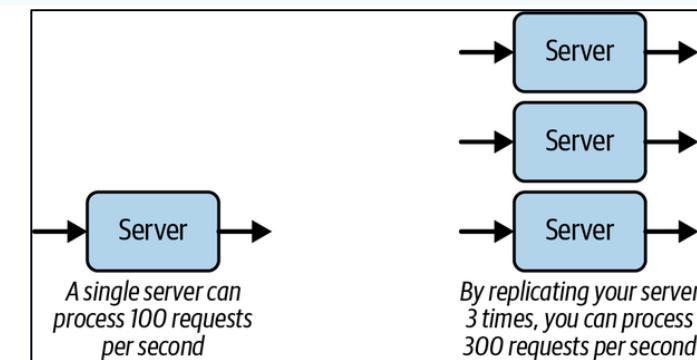
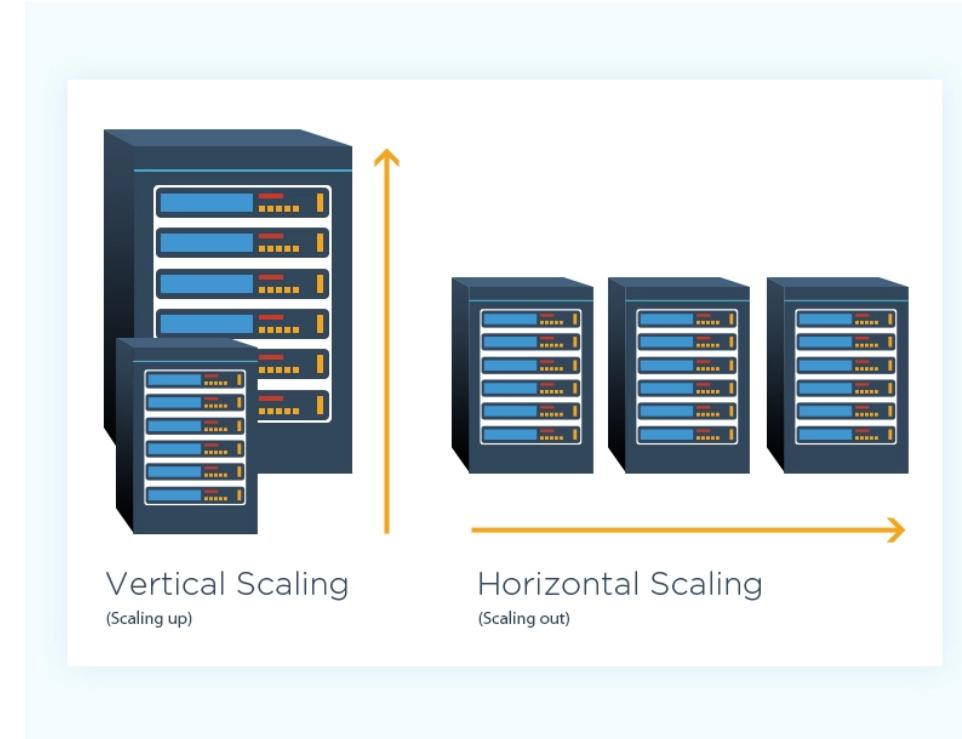
Tehnike koje omogućuju skalabilnost sustava

- Prikrivanje kašnjenja u komunikaciji
 - “radi nešto korisno dok čekaš odgovor” - asinkrona komunikacija
- Komponentno oblikovanje
 - više komponenti koji omogućuju funkcionalnost sustava (npr. raspodijeljena baza podataka, sustav imenovanja domena (DNS))
 - optimizacija implementacije pojedine komponente i cijelog sustava



Skalabilnost (3)

- Replikacija
 - replika = istovjetna kopija dijela sustava (funkcionalnosti) ili podatka
 - **horizontalno skaliranje, tzv. scale out** (radi povećanja propusnosti sustava)
 - problem: konzistentnost originala i kopije, upravljanje skupinom replika



povećanje propusnosti (broj obrađenih zahtjeva u sekundi)

Kvaliteta usluge

Kvaliteta usluge (engl. *Quality of Service, QoS*)

- skupni naziv za nefunkcijska obilježja sustava, od kojih su posebno važna sljedeća:
 - **vrijeme odziva** (engl. *response time*): vremenski period od slanja zahtjeva do primitka odgovora
 - **propusnost** (engl. *throughput*): mjeri promet na poslužitelju ili usluzi, a izražava se brojem zahtjeva u sekundi ili bit/s.
 - **raspoloživost** (engl. *availability*): vjerojatnost da je usluga dostupna u trenutku t i da generira odgovor na korisnički zahtjev.

Iskustvena kvaliteta (engl. *Quality of Experience, QoE*)

- mjeri korisnikovog zadovoljstva uslugom sustava

Oblikovanje raspodijeljenih sustava

Definiranje zahtjeva, potrebno odgovoriti na sljedeća pitanja

- Koje funkcijalne zahtjeve treba ostvariti – **ŠTO** sustav treba raditi?
- Kakve nefunkcijalne zahtjeve treba ostvariti – **KAKO** sustav treba raditi (kakva se kvaliteta usluge zahtijeva)?
- Temelji li se sustav na otvorenim rješenjima?
- Kakav je stupanj transparentnosti potreban i kako utječe na složenost, performance i troškove sustava?
- Kakva je skalabilnost sustava potrebna s motrišta veličine, rasprostranjenosti i upravljanja?

Sadržaj predavanja

- Definicija, obilježja i vrste raspodijeljenih sustava
- Zahtjevi na raspodijeljene sustave: otvorenost, transparentnost, skalabilnost i kvaliteta usluge
- **Arhitektura raspodijeljenih sustava**
- Primjeri modela raspodijeljene obrade
- Studijski primjeri: raspodijeljeni sustav weba, Internet stvari

Arhitektura raspodijeljenih sustava

Programska arhitektura:

- logička organizacija sustava: programske komponente sustava, njihova organizacija i interakcija

(centralizirana arhitektura ili decentralizirana arhitektura)

Sustavska arhitektura (engl. *deployment*):

- smještaj programskih komponenti na raspoložive računalne resurse

Kako predočiti raspodijeljeni sustav?

Slojevita arhitektura

- u središtu pozornosti aplikacijski sloj (sloj primjene)
- aplikacijski programi i procesi te usluge koje im pružaju niži slojevi

Arhitektura temeljena na komponentama

- npr. mikrousluga: komponenta sustava s dobro definiranim sučeljem
- mehanizam komunikacije, usklađivanja i suradnje mikroservisa

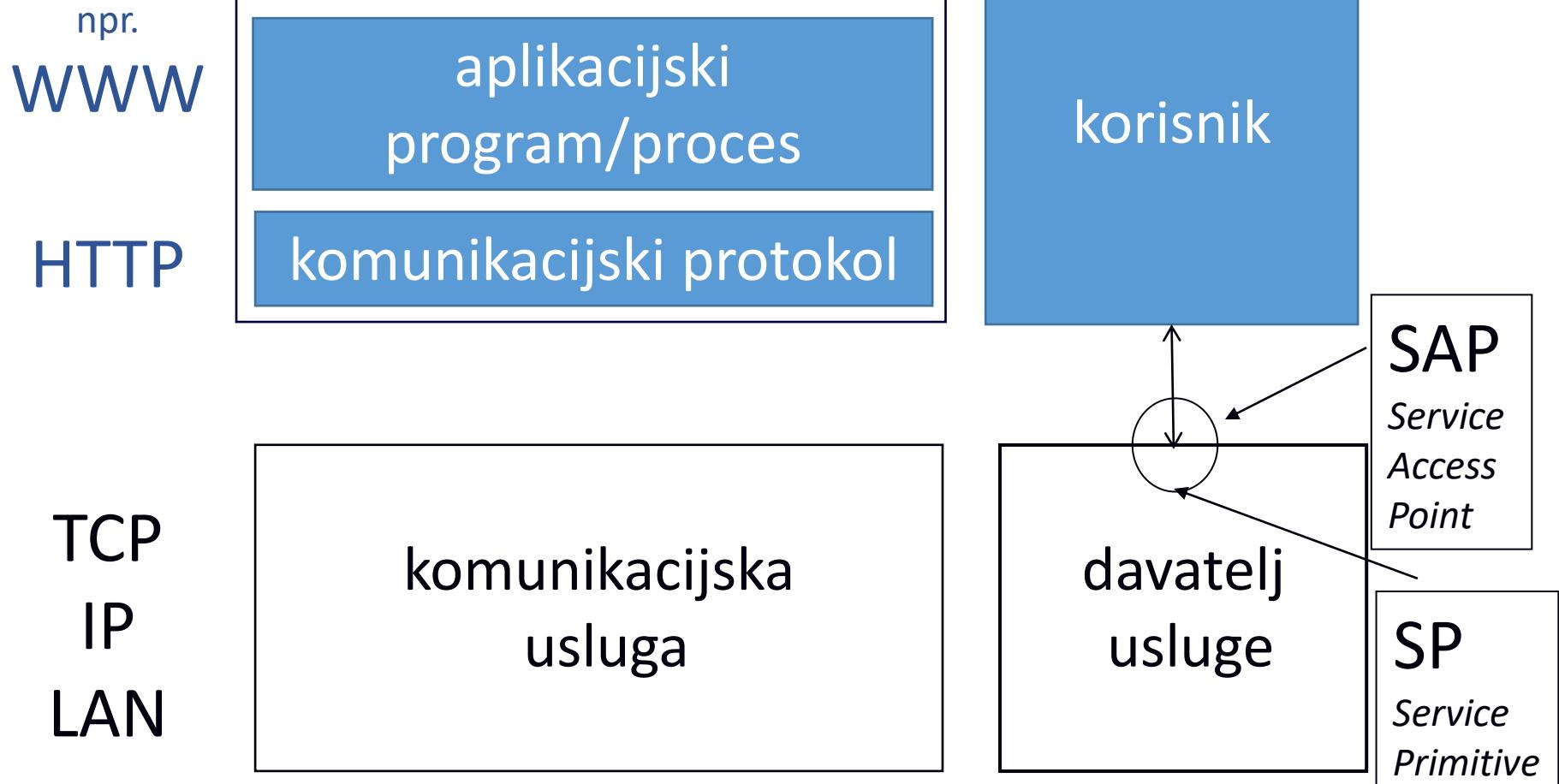
Arhitektura temeljena na podacima

- procesi komuniciraju putem zajedničkog, dijeljenog (raspodijeljenog) repozitorija

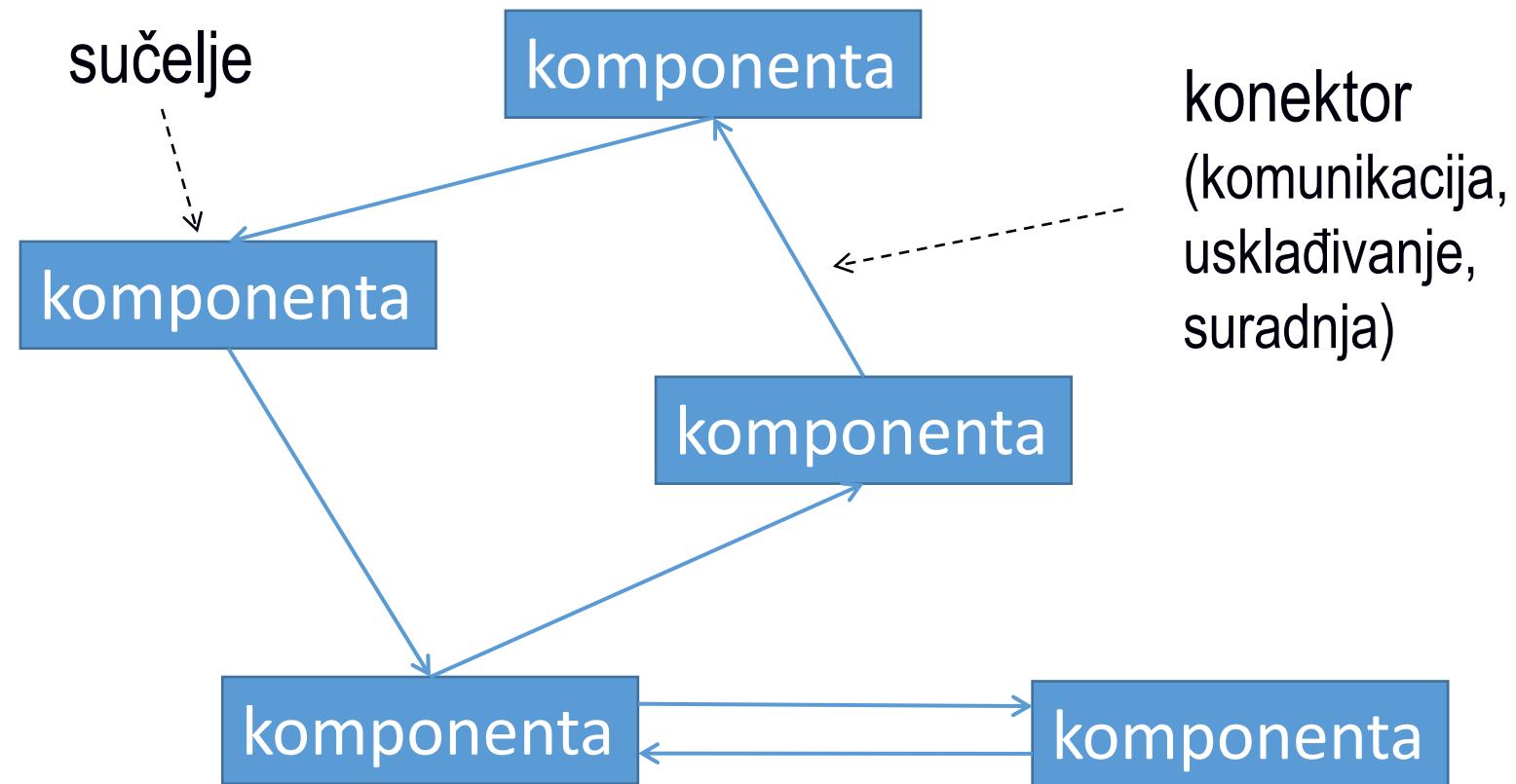
Arhitektura temeljena na događajima

- procesi komuniciraju razmjenom tzv. događaja koji prenose informacije

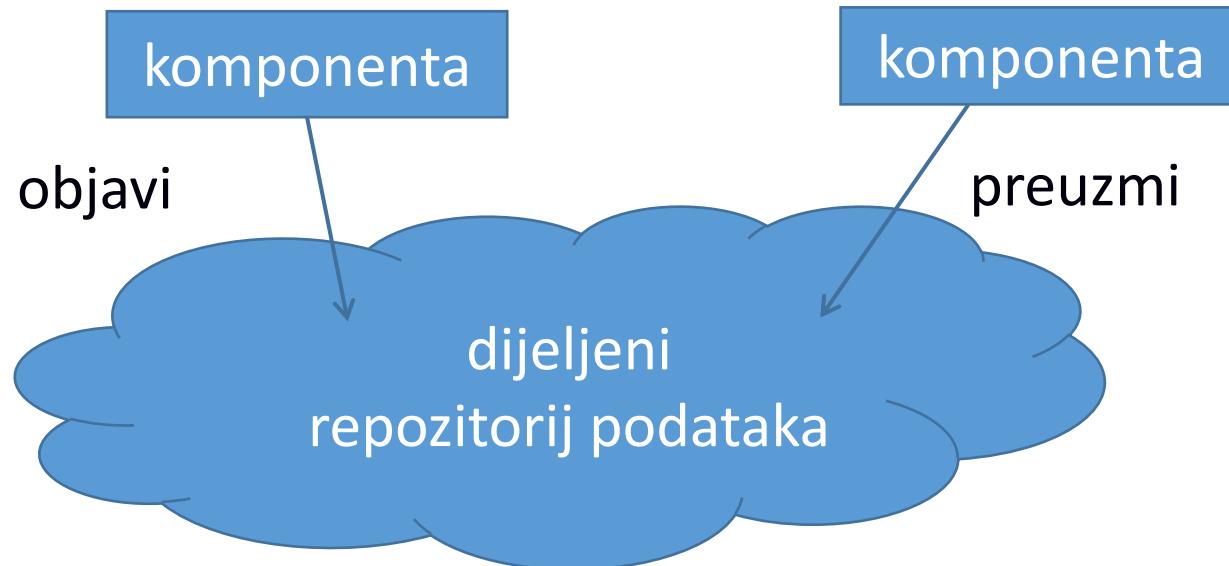
Slojevita arhitektura



Arhitektura temeljena na komponentama

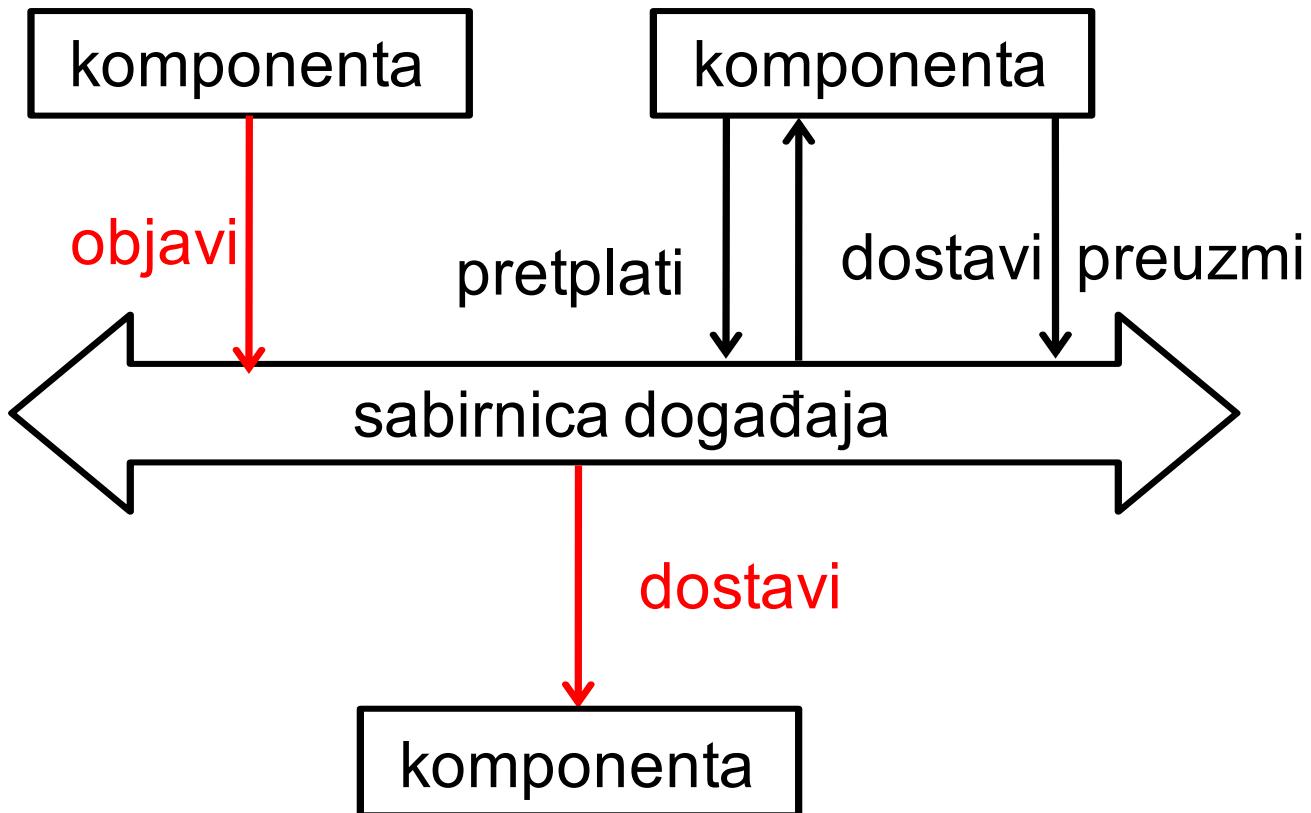


Arhitektura temeljena na podacima



Komponenta sustava upisuje (objavljuje) podatak u dijeljeni repozitorij koji omogućuje čitanje (preuzimanje) podatka drugim komponentama.

Arhitektura temeljena na događajima



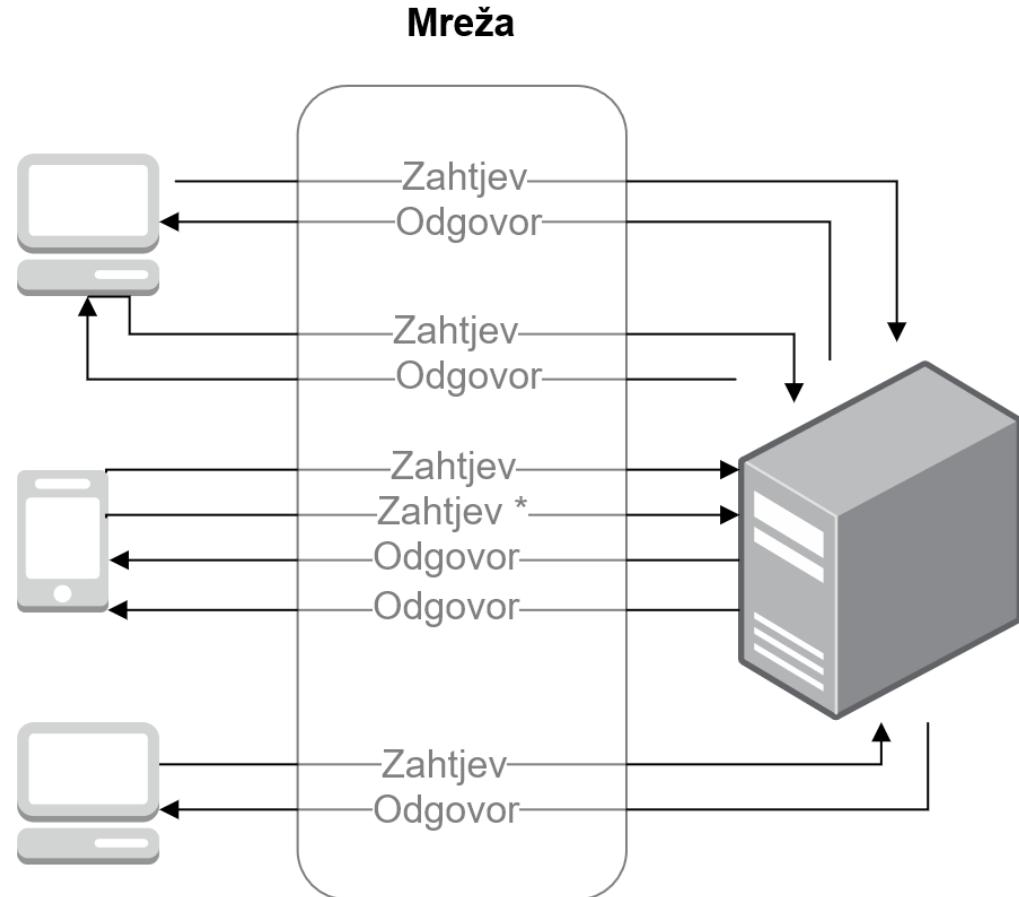
Komponenta se može preplatiti na podatak koji joj je potreban. Kad neka komponenta objavi podatak (događaj!), to će se dostaviti (događaj!) komponenti koja je na njega preplaćena.

Sadržaj predavanja

- Definicija, obilježja i vrste raspodijeljenih sustava
- Zahtjevi na raspodijeljene sisteme: otvorenost, transparentnost, skalabilnost i kvaliteta usluge
- Arhitektura raspodijeljenih sustava
- **Primjeri modela raspodijeljene obrade**
- Studijski primjeri: raspodijeljeni sustav weba, Internet stvari

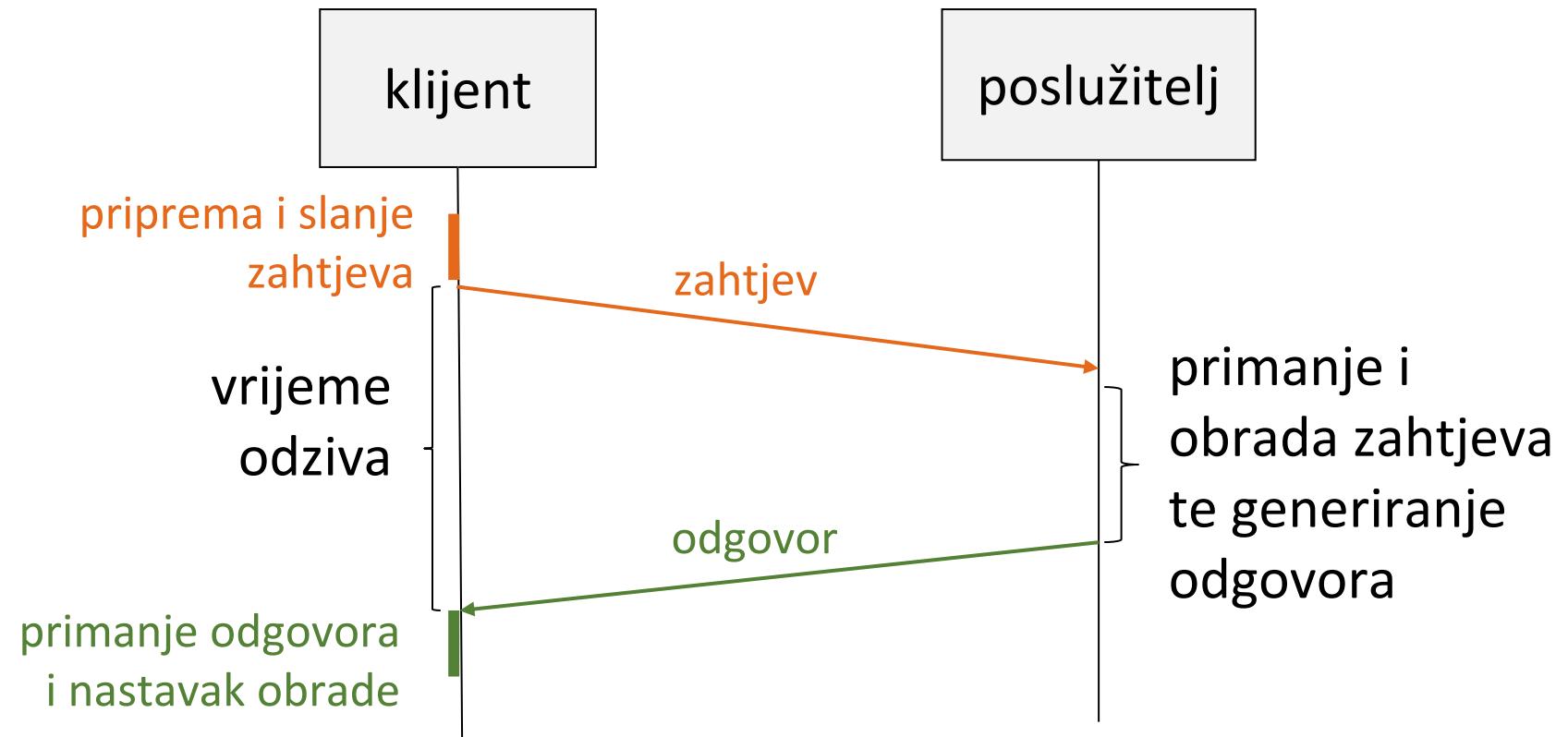
Model klijent – poslužitelj (1)

- Klijent: traži uslugu (zahtjev)
- Poslužitelj: pruža uslugu (odgovor) za više/mnogo klijenta



Model klijent – poslužitelj (2)

- Klijent šalje zahtjev i čeka odgovor
- Poslužitelj: prihvaca i obrađuje zahtjev te vraća odgovor

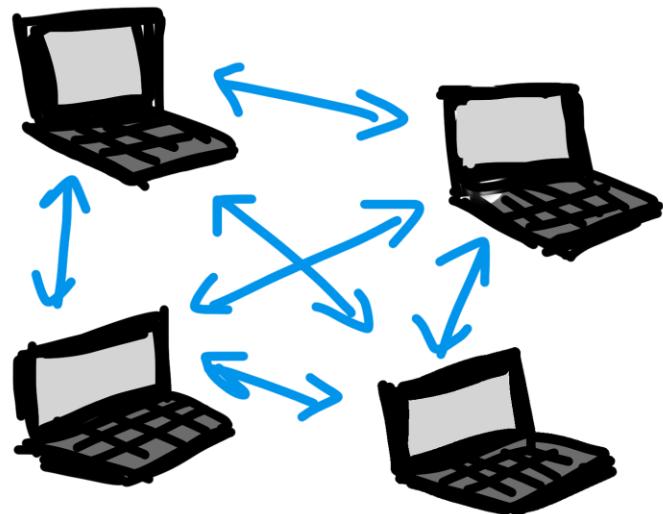


Model ravnopravnih sudionika (1)

- ravnopravni sudionik (engl. *peer*) je onaj koji može obaviti i funkciju poslužitelja i funkciju klijenta
- ravnopravni sudionici međusobno komuniciraju (engl. *Peer-to-Peer, P2P*) tako da se na aplikacijskom sloju povezuju u “prekrivajuću mrežu” (engl. *overlay network*) nad stvarnom mrežnom topologijom
- svaki čvor “plaća” sudjelovanje u mreži nudeći dio vlastitih sredstava ostalim čvorovima
- model ravnopravnih sudionika potencijalno nudi neograničena sredstva u velikim mrežama s puno čvorova

Model ravnopravnih sudionika (2)

Peer-to-peer, P2P



Decentralizirani raspodijeljeni sustav

- nema centralizirane koordinacije među *peerovima*
- ne postoji jedna točka ispada

Samoorganizirajuća mreža čvorova

- *peerovi* su međusobno neovisni, ulaze i izlaze iz sustava po volji

Programski agenti i premještanje programa

- **Programski agent** (engl. *software agent*): program koji obavlja neki posao za svog korisnika ili vlasnika, a raspolaže svojstvima kao što su **inteligencija, samostalnost, reaktivnost, proaktivnost** itd.
- **Migracija programa** (engl. *code migration*): razmjena programa između umreženih čvorova:
 - migracija procesa s jednog računala na drugo zbog (proširenja) funkcionalnosti, (uravnoteženja) opterećenja, (uvođenja) konkurentnosti,
- **Pokretni agent** (engl. *mobile agent*): programski agent koji predočuje korisnika u mreži i za njega obavlja neki posao krećući se samostalno između čvorova u mreži;

Sadržaj predavanja

- Definicija, obilježja i vrste raspodijeljenih sustava
- Zahtjevi na raspodijeljene sustave: otvorenost, transparentnost, skalabilnost i kvaliteta usluge
- Arhitektura raspodijeljenih sustava
- Primjeri modela raspodijeljene obrade
- **Studijski primjer: raspodijeljeni sustav weba**

Osnovne postavke weba

- Internetska aplikacija:
 - komunikacijski protokol aplikacijskog sloja - HTTP
 - jezik za označavanje HTML
 - standardi: *World Wide Web Consortium* (www.w3.org)
- Model klijent – poslužitelj
- Otvoreni sustav s transparentnim pristupom i konkurencijskom transparentnosti
- Transformacija weba:
 - “korisnik čita sadržaj” → “korisnik stvara sadržaj” (Web 2.0)
 - “korisnik odabire sadržaj” → “korisnik traži uslugu”: web-usluga (engl. *Web Service*)



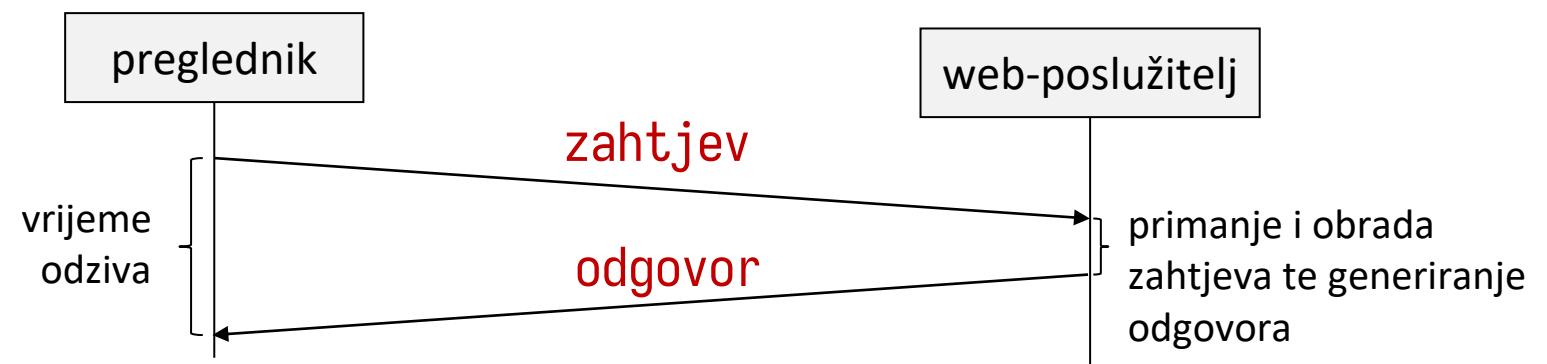
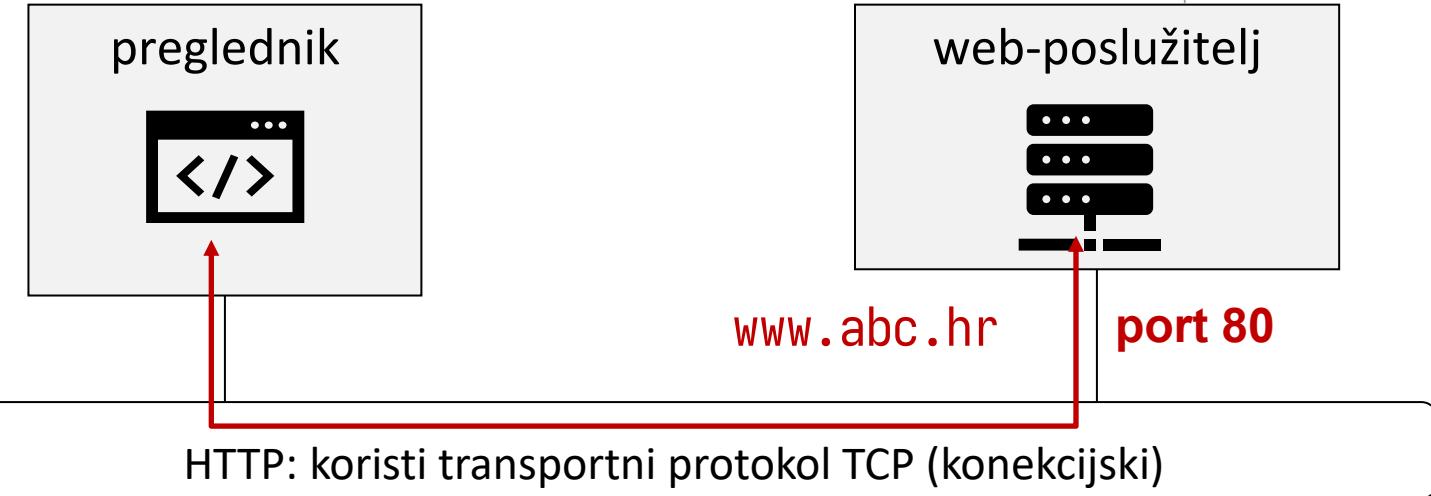
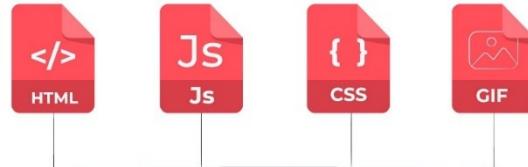
Hypertext Transfer Protocol (HTTP)

HTTP je **protokol** na aplikacijskom sloju

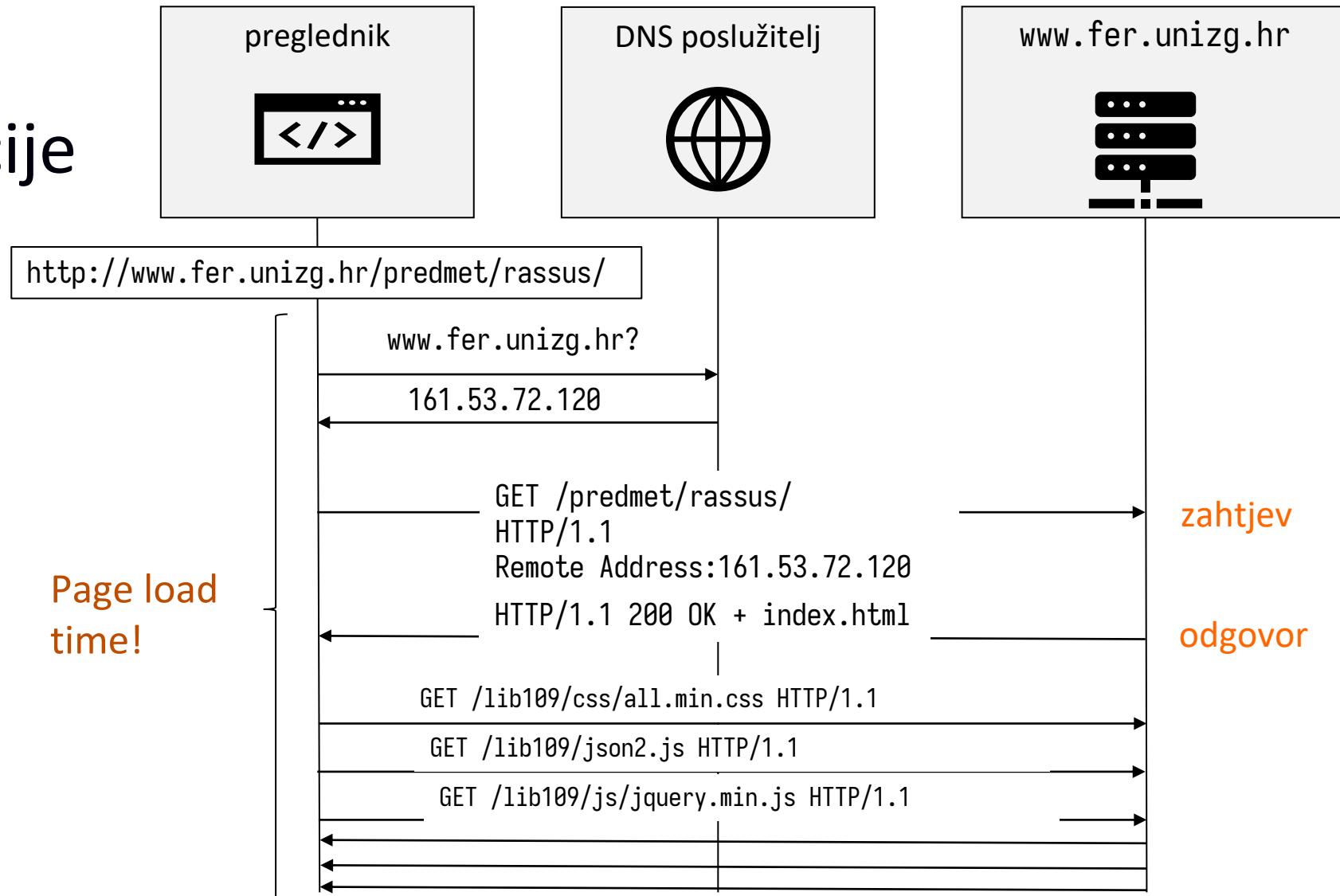
- protokol definira format i sadržaj poruka (zahtjeva i odgovora) te očekivano „ponašanje” poslužitelja, tj. pravila za generiranje odgovora na primljeni zahtjev
- **Zahtjev**: definira operaciju (tzv. metodu), oznaku resursa, verziju protokola, itd.
- **Odgovor**: rezultat (uspjeh, neuspjeh, pogreška,..) opisan statusnim kôdom i sadržaj resursa (npr. datoteka HTML, CSS, JPEG...)
- Poslužitelj ne čuva stanje između dviju konverzacija s istim klijentom jer je HTTP **protokol bez očuvanja stanja (engl. *stateless protocol*)**

Podsjetimo se kako web izgleda ...

resursi smješteni na poslužitelju



Primjer komunikacije



Transparentnost web-poslužitelja (1)

Lokacijska transparentnost:

- postiže se simboličkim imenima koja se u sustavu imenovanja domena (DNS) prevode u lokacije poslužitelja (mrežne adrese):
 - korisnik rabi simbolička imena, položaj poslužitelja (sjedišta weba) kao i bilo kojeg resursa ne treba biti i nije poznat korisniku

Migracijska transparentnost:

- ne mijenja se simboličko ime, već se samo mijenja lokacija poslužitelja (mrežna adresa) u DNS-u

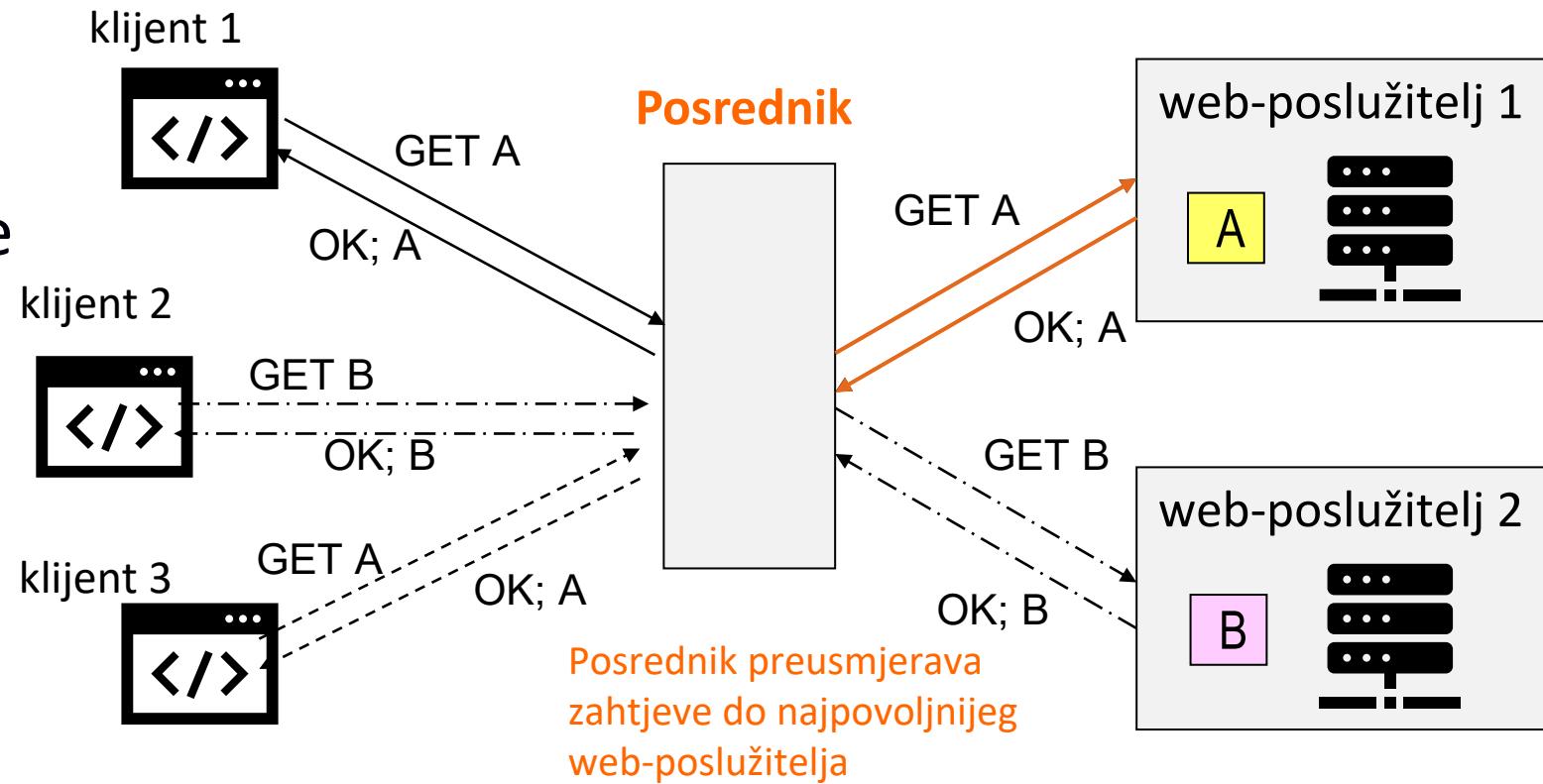
Relokacijska transparentnost:

- ne zahtijeva se, poslužitelj je stacionaran i ne kreće se tijekom pružanja usluge

Transparentnost web-poslužitelja (2)

Replikacijska transparentnost 1:

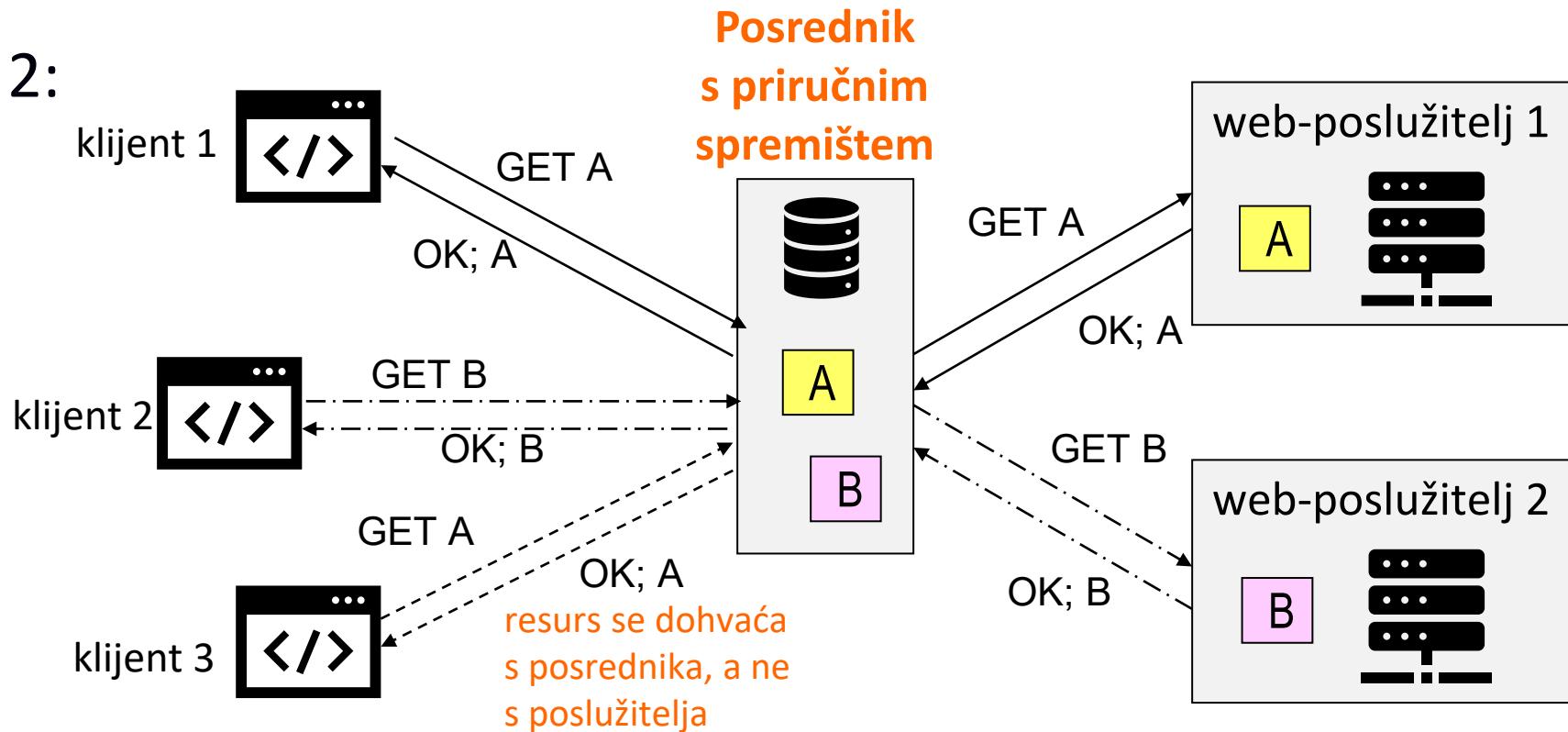
- Poslužiteljima se dostupa posredstvom posrednika (engl. *proxy*)



Transparentnost web-poslužitelja (3)

Replikacijska
transparentnost 2:

- Uvođenje
priručnog
spremišta



Sadržaj predavanja

- Definicija, obilježja i vrste raspodijeljenih sustava
- Zahtjevi na raspodijeljene sisteme: otvorenost, transparentnost, skalabilnost i kvaliteta usluge
- Arhitektura raspodijeljenih sustava
- Primjeri modela raspodijeljene obrade
- **Studijski primjer: Internet stvari**

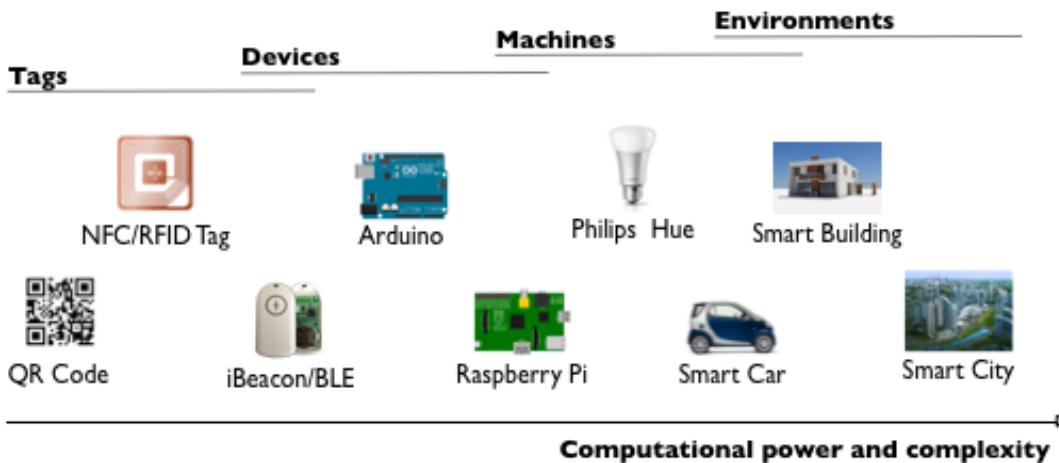
Uredaji i "stvari" postaju dio Interneta

Internet Connected
Object (ICO)



Što je "stvar"?

- Objekt iz fizičkog svijeta ili virtualnog digitalnog svijeta (virtualni objekt)
 - ima jedinstveni identifikator i povezan je na Internet (direktno ili putem posrednika), *Internet Connected Object* (ICO)
 - senzor: opažanje okoline, potencijalno kontinuirano generira podatke
 - aktuator: može izvršiti određene funkcije



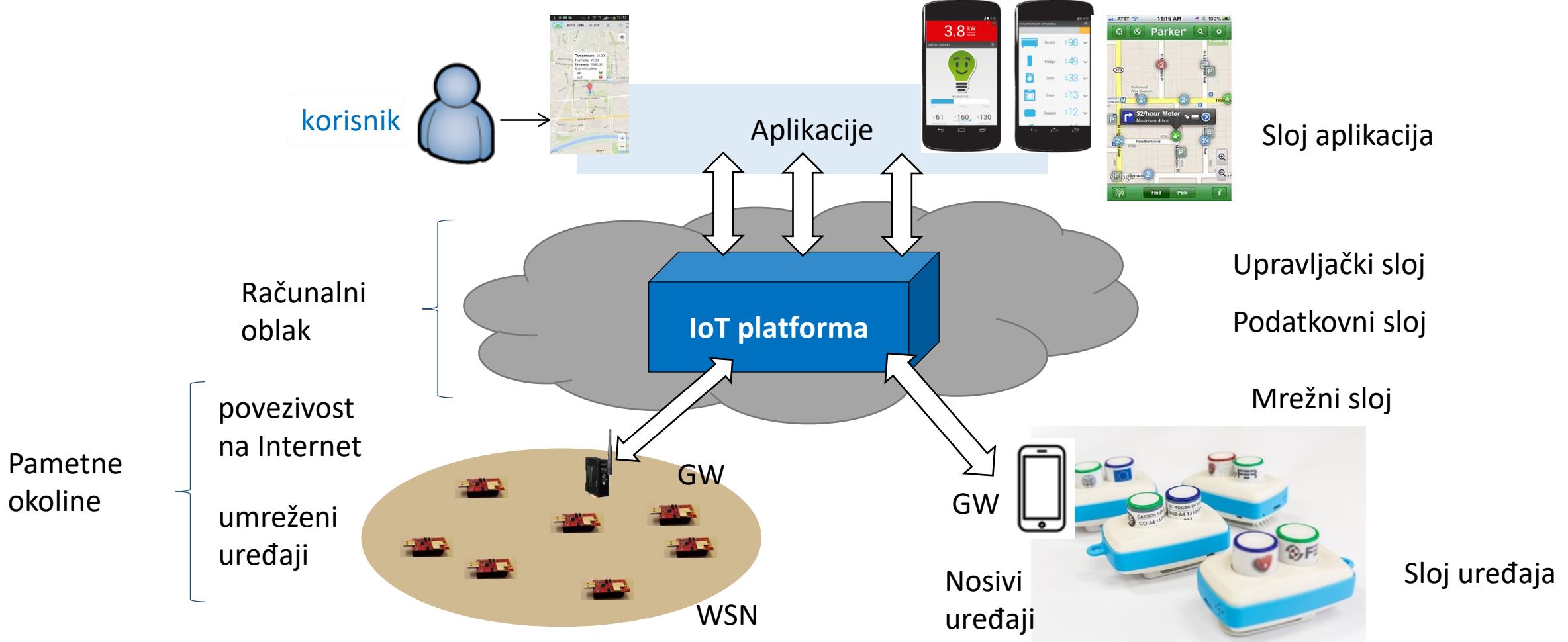
Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

Internet of Things (IoT): definicija

ITU-T Recommendation Y.2060, 06/2012:

- *A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) Things based on, existing and evolving, interoperable information and communication technologies.*
 - *Through the exploitation of identification, data capture, processing and communication capabilities, the IoT makes full use of things to offer services to all kinds of applications, whilst ensuring that security and privacy requirements are fulfilled.*
 - *In a broad perspective, the IoT can be perceived as a vision with technological and societal implications.*

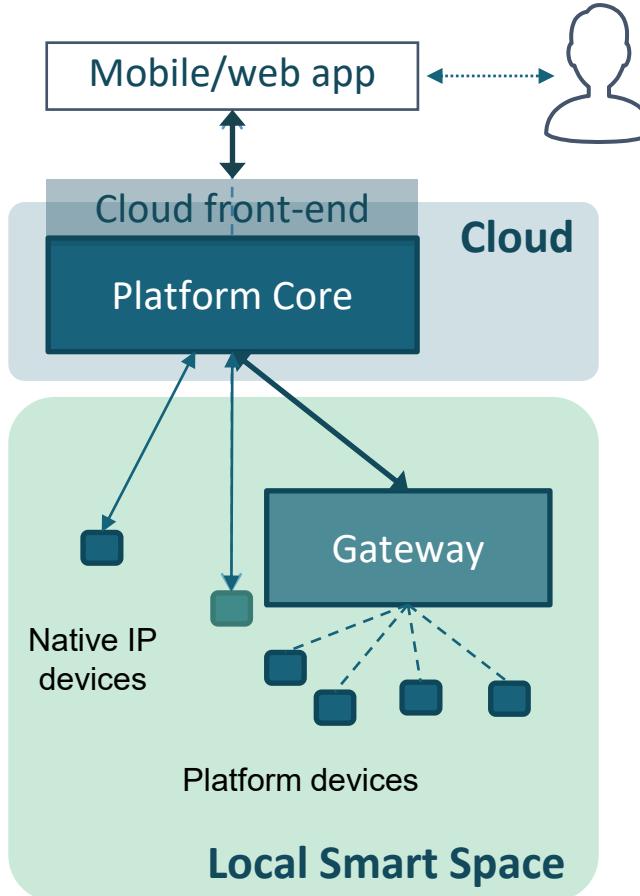
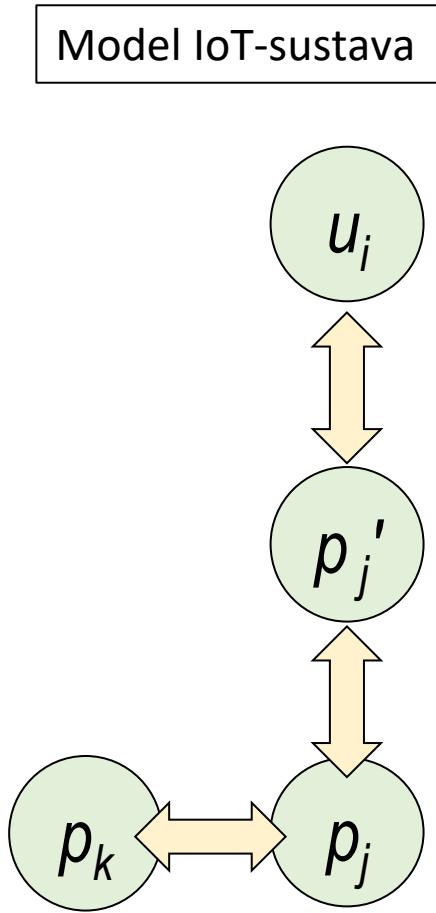
Pojednostavljena arhitektura Interneta stvari



Kako integrirati "stvari" i ponuditi aplikacije korisnicima?

- Pomoću programskih platformi (**IoT-platforma**) koje integriraju i upravljaju uređajima
 - raspodijeljeni sustav velikih razmjera
 - uređaji: često imaju vrlo ograničene resurse te su povezani na Internet putem prilaznog uređaja (engl. *gateway*)
 - potrebno je objediniti i na jedinstveni način zapisati podatke primljene iz različitih izvora
 - potreba za obradom velike količine podataka (često u stvarnom vremenu)
 - *Web of Things*: koncept koji povezuje uređaje direktno na WWW (tehnologije vezane uz protokol HTTP)

IoT-rješenje je složeni raspodijeljeni sustav



Virtualni entitet predstavlja stvarni uređaj

- programska platforma održava metapodatke o uređajima
- pohranjuje senzorska očitanja, stanja aktuatora, obrađuje podatke

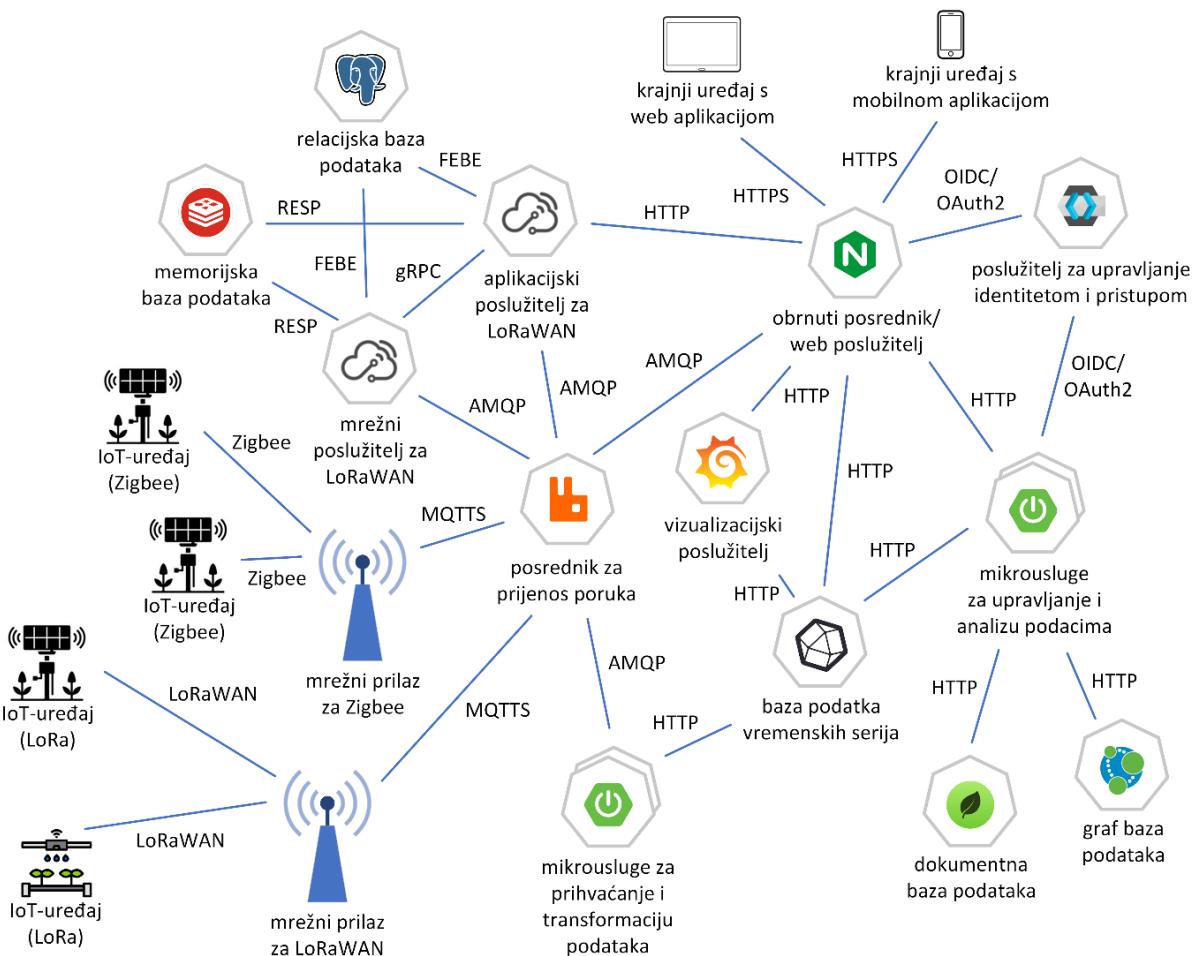
Interoperabilna IoT-platforma



<http://www.iot.fer.hr/>



Arhitektura IoT-platforme



Zadaci (1)

1. Čime je definirana otvorenost weba?
2. Kojim aspektima transparentnosti pridonosi sustav imenovanja domena (DNS)?
3. Kako replikacija pridonosi otpornosti na kvarove i skalabilnosti?
4. Kakvi bi se problemi pojavili kada bi se više repliciranih poslužitelja weba priključilo na mrežu izravno, a ne posredstvom zastupnika (*proxy*)?
5. Što sve utječe na vrijeme odziva poslužitelja weba?

Zadaci (2)

6. Na primjeru weba objasnite razliku između vertikalnog i horizontalnog skaliranja sustava.
7. Zašto se koriste perzistentne konekcije u HTTP-u?
8. Kako se definira uvjetni GET i kako funkcionira? Kakve prednosti donosi za performance HTTP-a?
9. Objasnite zašto današnja rješenja za IoT predstavljaju raspodijeljene sustave velikih razmjera?



SVEUČILIŠTE U ZAGREBU



**Diplomski studij
Računarstvo**

Znanost o mrežama
Programsko inženjerstvo i
informacijski sustavi
Računalno inženjerstvo
**Ostali (slobodni izborni
predmet)**

Raspodijeljeni sustavi

2. Procesi i komunikacija: model klijent-poslužitelj

Ak. god. 2022./2023.

Creative Commons



- slobodno smijete:

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
 - **prerađivati** djelo



- pod sljedećim uvjetima:

- **imenovanje**: morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
 - **nekomercijalno**: ovo djelo ne smijete koristiti u komercijalne svrhe.
 - **dijeli pod istim uvjetima**: ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.



U slučaju daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.

Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.

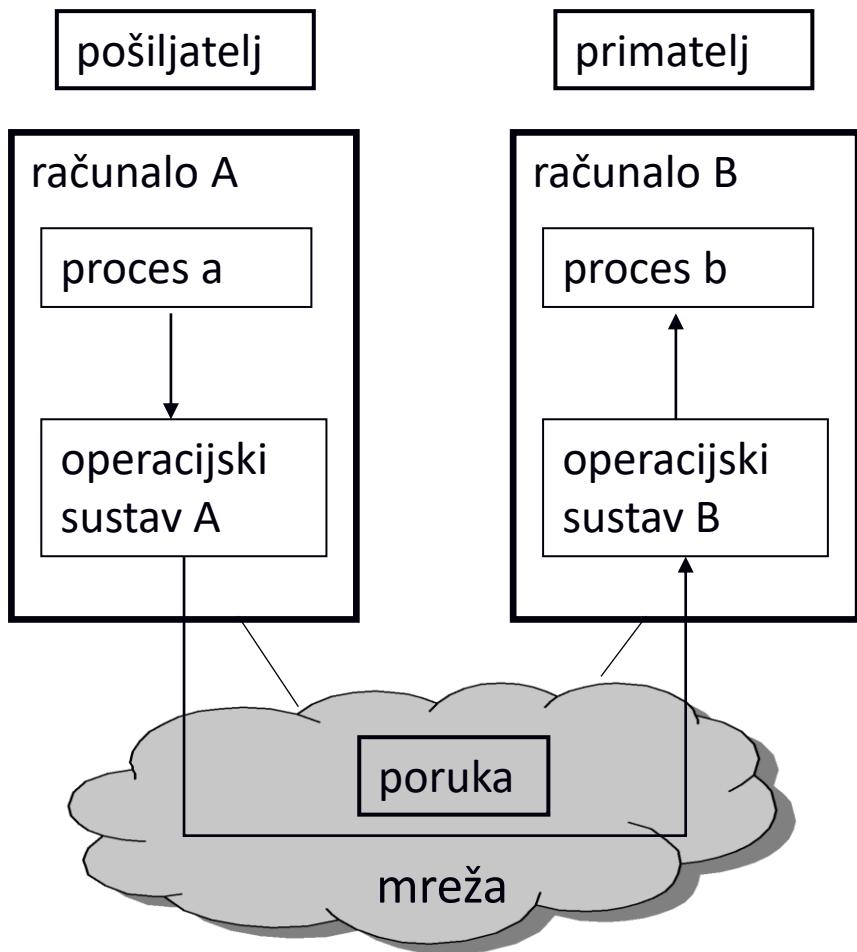
Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.

Tekst licence preuzet je s <http://creativecommons.org/>

Sadržaj predavanja

- Osnovni model komunikacije u raspodijeljenom okruženju
 - obilježja komunikacije
 - obilježja procesa
- Sloj raspodijeljenog sustava za komunikaciju među procesima
 - komunikacija korištenjem priključnica (Socket API)
 - primjeri TCP/UDP klijenta i poslužitelja
 - oblikovanje višedretvenog poslužitelja
 - poziv udaljene procedure (*Remote Procedure Call - RPC*) / poziv udaljene metode (*Remote Method Invocation - RMI*)
 - Java RMI
 - gRPC

Osnovni model komunikacije



- **Procesi**

- izvode se na različitim računalima, autonomni su

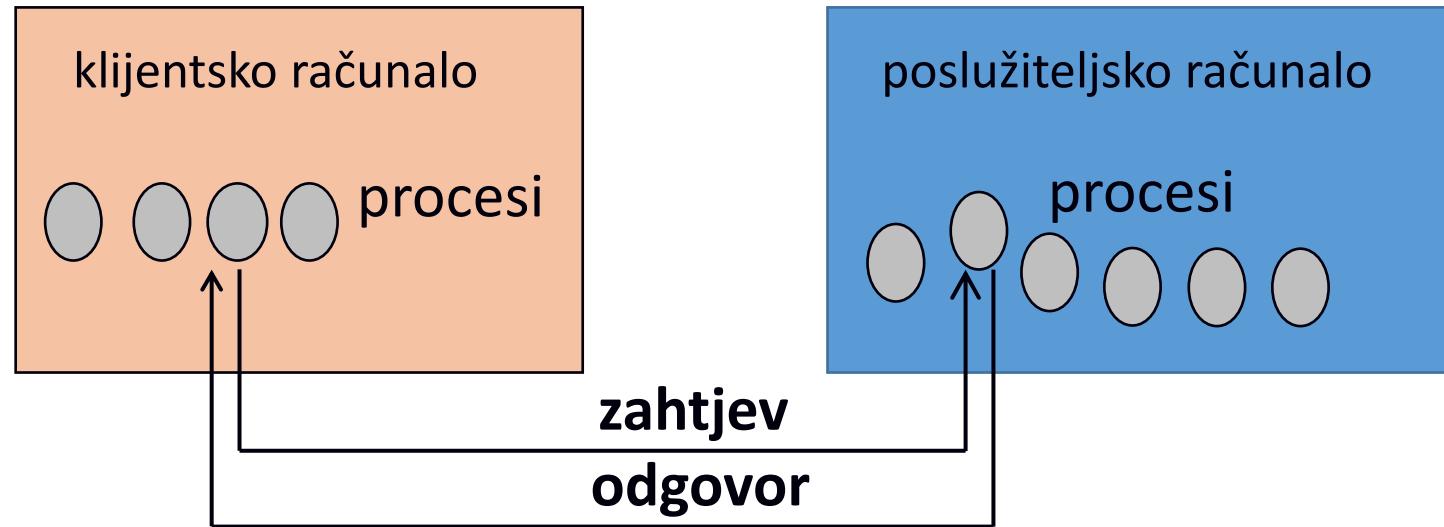
- **Komunikacija**

- proslijedivanje poruka (engl. *message passing*), tj. razmjena poruka na mrežnom sloju

- Međuprocesna komunikacija

- engl. *interprocess communication* (IPC)
- potrebno je osigurati vremensku usklađenost procesa

Prisjetimo se modela klijent-poslužitelj



- KLIJENT

- zahtjeva uslugu
- šalje zahtjev poslužitelju i čeka odgovor

- POSLUŽITELJ

- nudi usluge
- prima i obrađuje dolazne zahtjeve te šalje odgovor klijentima

Obilježja komunikacije

- **koneksijska**
 - procesi eksplisitno kreiraju konekciju prije razmjene podataka, postoje kontrolne poruke za uspostavu konekcije
- **bezkoneksijska**
 - sve poruke prenose podatke, nema kontrolnih poruka za uspostavu konekcije među procesima
- **perzistentna komunikacija**
 - garantira isporuku poruke, poruka se pohranjuje u sustavu i isporučuje primatelju kada je to moguće
- **tranzijentna komunikacija**
 - nepouzdana, garantira isporuku poruke samo ako su pošiljatelj i primatelj poruke istovremeno dostupni

Obilježja komunikacije

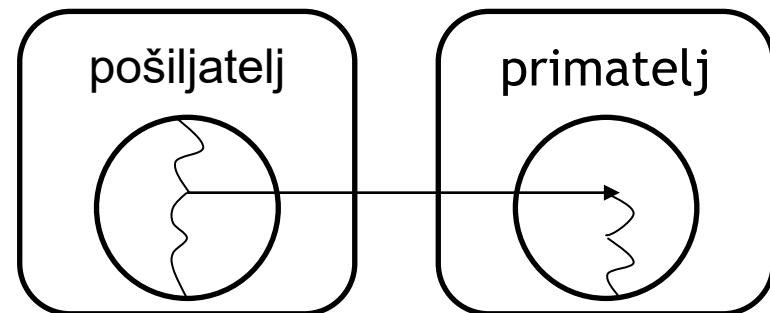
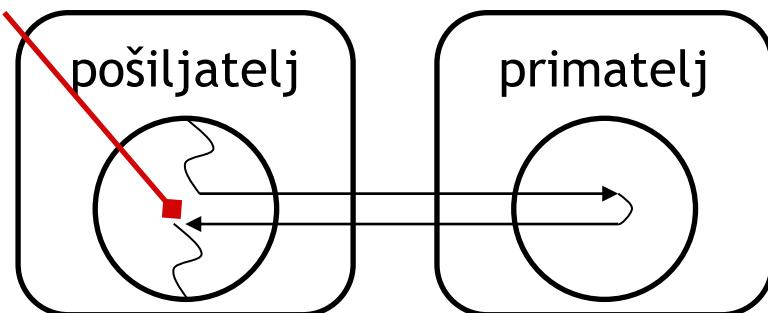
- **sinkrona komunikacija**

- blokira pošiljatelja do primitka potvrde od strane primatelja

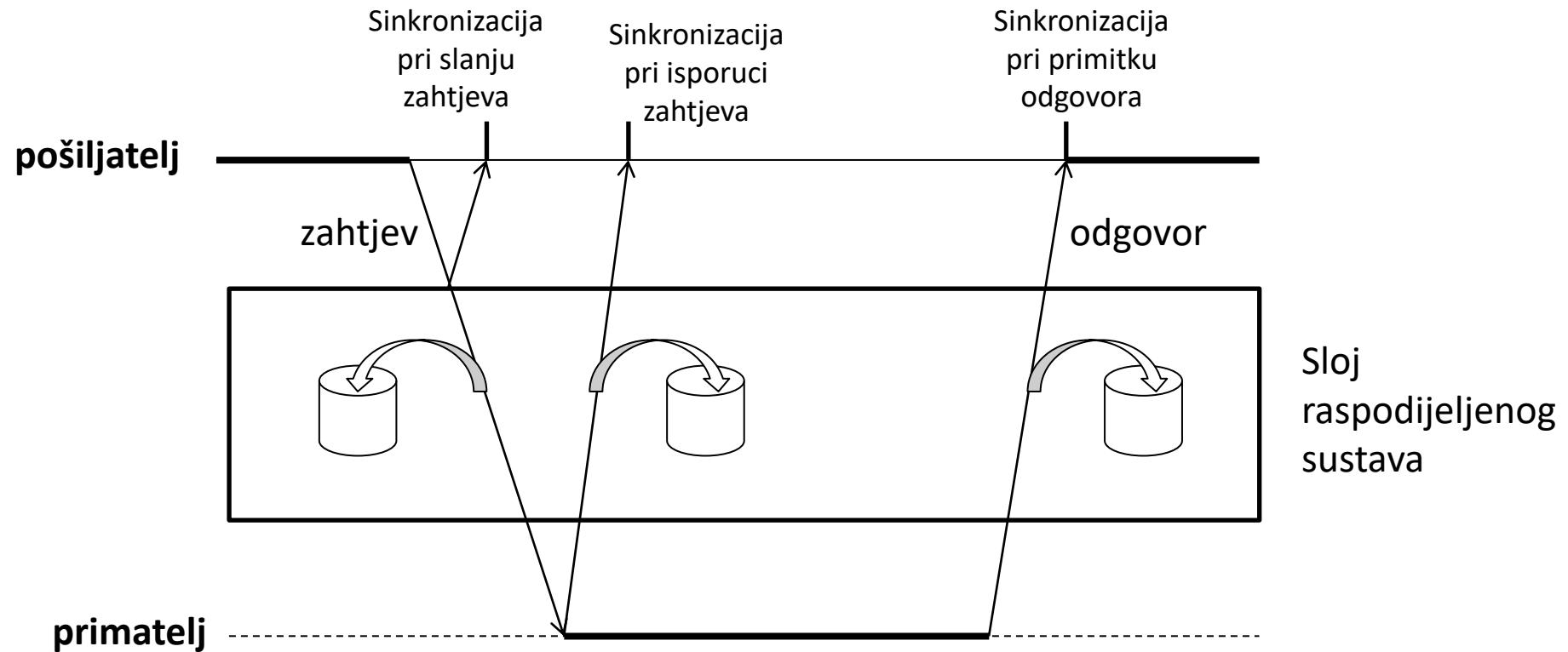
- **asinkrona komunikacija**

- omogućuje pošiljatelju nastavak obrade odmah nakon slanja poruke

blokiranje

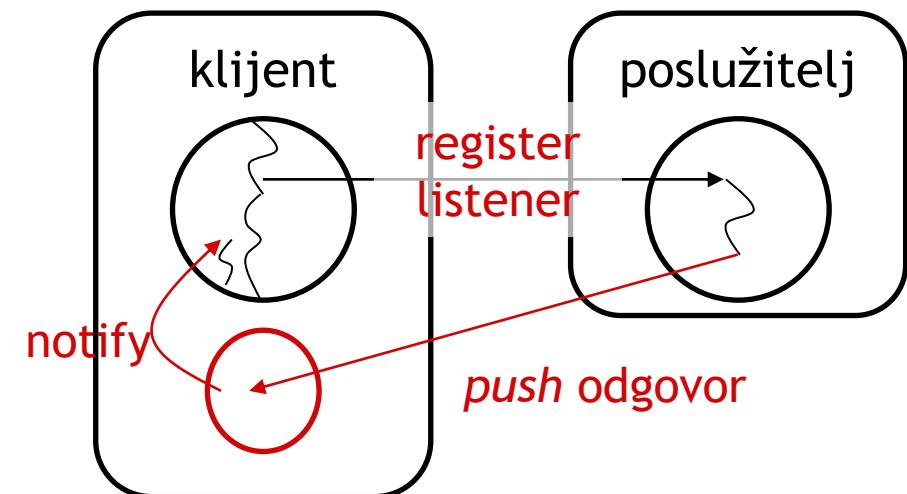
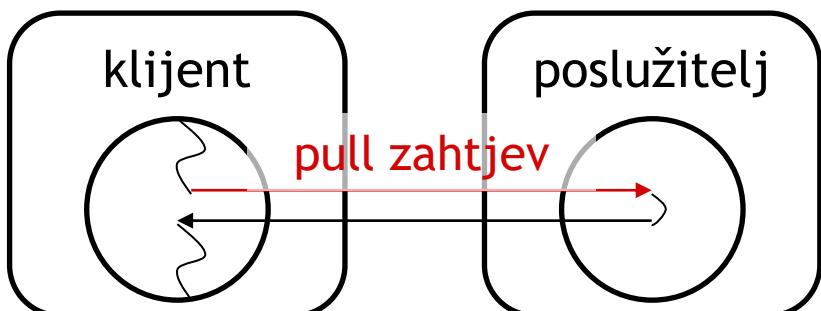


Točke sinkronizacije



Obilježja komunikacije

- komunikacija na načelu *pull* ili *push*
 - *pull* – “klasični” model zahtjev-odgovor
 - *push* – klijent registrira zahtjev i “sluša” odgovor, poslužitelj šalje odgovor nakon što završi obradu zahtjeva



Procesi

- Definira se kao program u izvođenju (prisjetimo se operacijskih sustava)
- Višedretvenost je važna za učinkovitu implementaciju raspodijelih procesa
 - omogućuje održavanje više logičkih konekcija s jednim procesom
 - višedretveni poslužitelj može paralelno obradivati korisničke zahtjeve
 - višedretveni klijent može nastaviti s obradom dok čeka odgovor poslužitelja
(primjer: Web preglednik)

Obilježja procesa

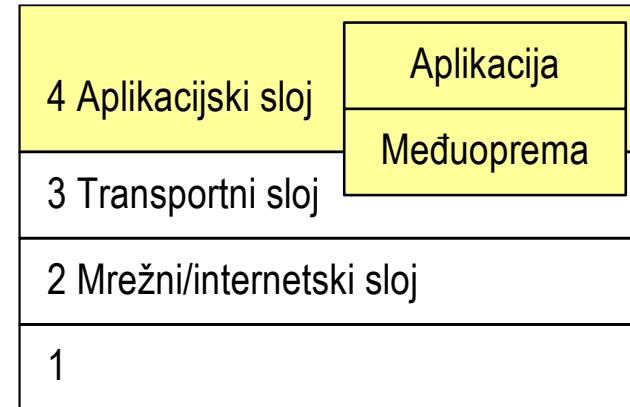
- **vremenska (ne)ovisnost**
 - vremenski ovisni procesi moraju biti istovremeno aktivni za realizaciju komunikacije
 - vremenski neovisni procesi mogu komunicirati i ako nisu istovremeno aktivni
- **ovisnost o referenci “sugovornika”**
 - proces je ovisan o referenci “sugovornika” ako mora znati jedinstveni identifikator (adresu) udaljenog procesa s kojim želi komunicirati
 - proces može biti i neovisan o referenci, tj. ne mora znati jedinstveni identifikator udaljenog procesa

Sadržaj predavanja

- Osnovni komunikacijski model
 - obilježja komunikacije
 - obilježja procesa
- **Sloj raspodijeljenog sustava za komunikaciju među procesima**
 - komunikacija korištenjem priključnica (Socket API)
 - primjeri TCP/UDP klijenta i poslužitelja
 - oblikovanje višedretvenog poslužitelja
 - poziv udaljene procedure (*Remote Procedure Call - RPC*) / poziv udaljene metode (*Remote Method Invocation - RMI*)
 - Java RMI
 - gRPC

Sloj raspodijeljenog sustava za komunikaciju među procesima

- vrsta programskog posredničkog sloja (međuopreme)
- implementira komunikacijske protokole za raspodijeljene procese na višem nivou apstrakcije od transportnog sloja
- omogućuje jednostavniji razvoj raspodijeljenih aplikacija, sakriva kompleksnost i heterogenost nižih slojeva



Sloj raspodijeljenog sustava za komunikaciju među procesima

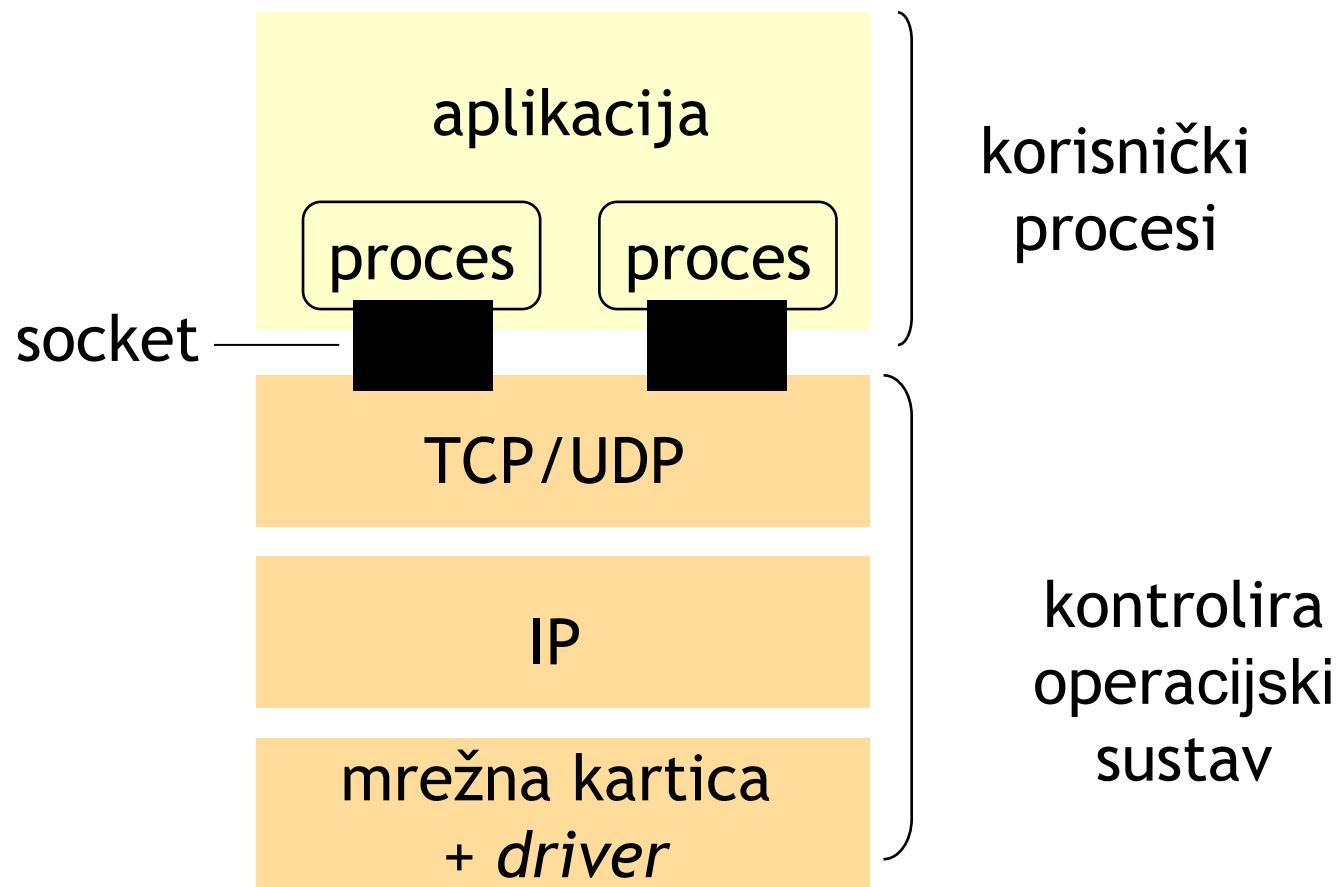
- Postojeća rješenja za komunikaciju raspodijeljenih procesa
 1. komunikacija korištenjem priključnica (*socket API*)
 2. poziv udaljene procedure (*remote procedure call, RPC*)
 3. raspodijeljeni objekti - poziv udaljene metode (*remote method invocation, RMI*)
 4. komunikacija razmjenom poruka (*message-oriented interaction*)
 5. model objavi-preplati (*publish/subscribe*)
- U nastavku analiziramo prva 3 rješenja koja se temelje na modelu klijent-poslužitelj

Komunikacija korištenjem priključnica

Socket API

- koristi funkcionalnost transportnog sloja
 - TCP – konekcijski protokol, pouzdan prijenos podataka
 - UDP – prijenos nezavisnih paketa (*datagrami*), nepouzdan prijenos
- priključnica (engl. *socket*)
 - pristupna točka preko koje aplikacija šalje podatke u mrežu i iz koje čita primljene podatke
 - viši nivo apstrakcije nad komunikacijskom točkom koju operacijski sustav koristi za pristup transportnom sloju
 - veže se uz vrata (engl. *port*) koja jednoznačno određuju aplikaciju kojoj su poruke namijenjene

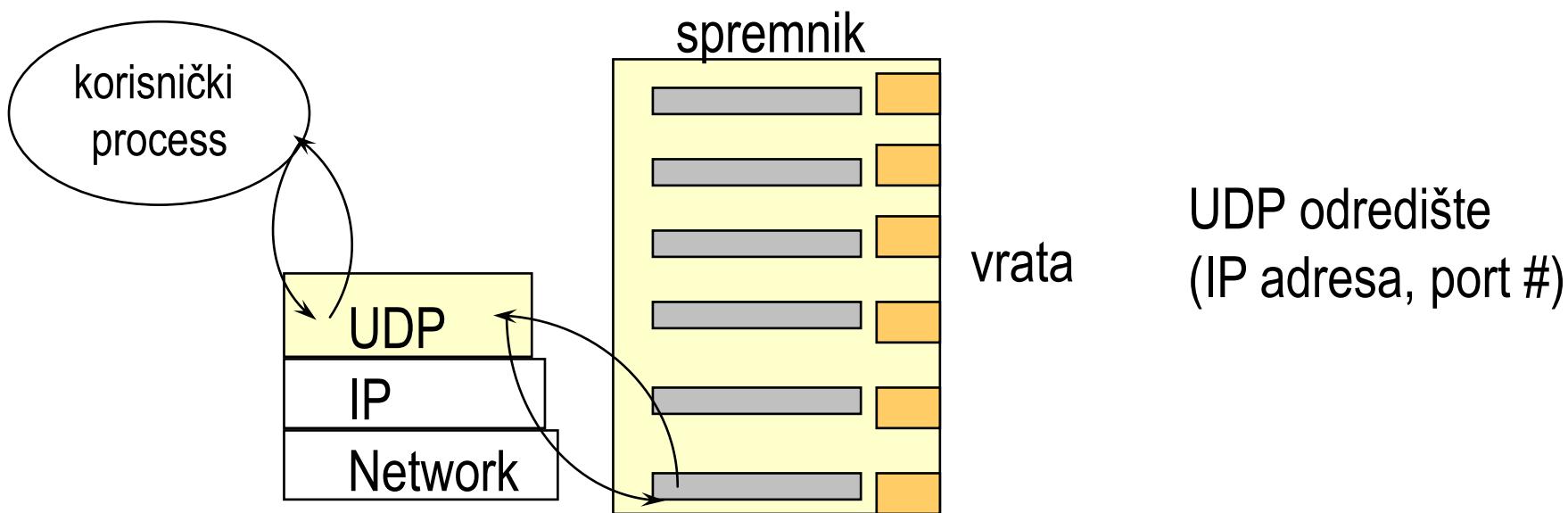
Komunikacija pomoću *socketa*



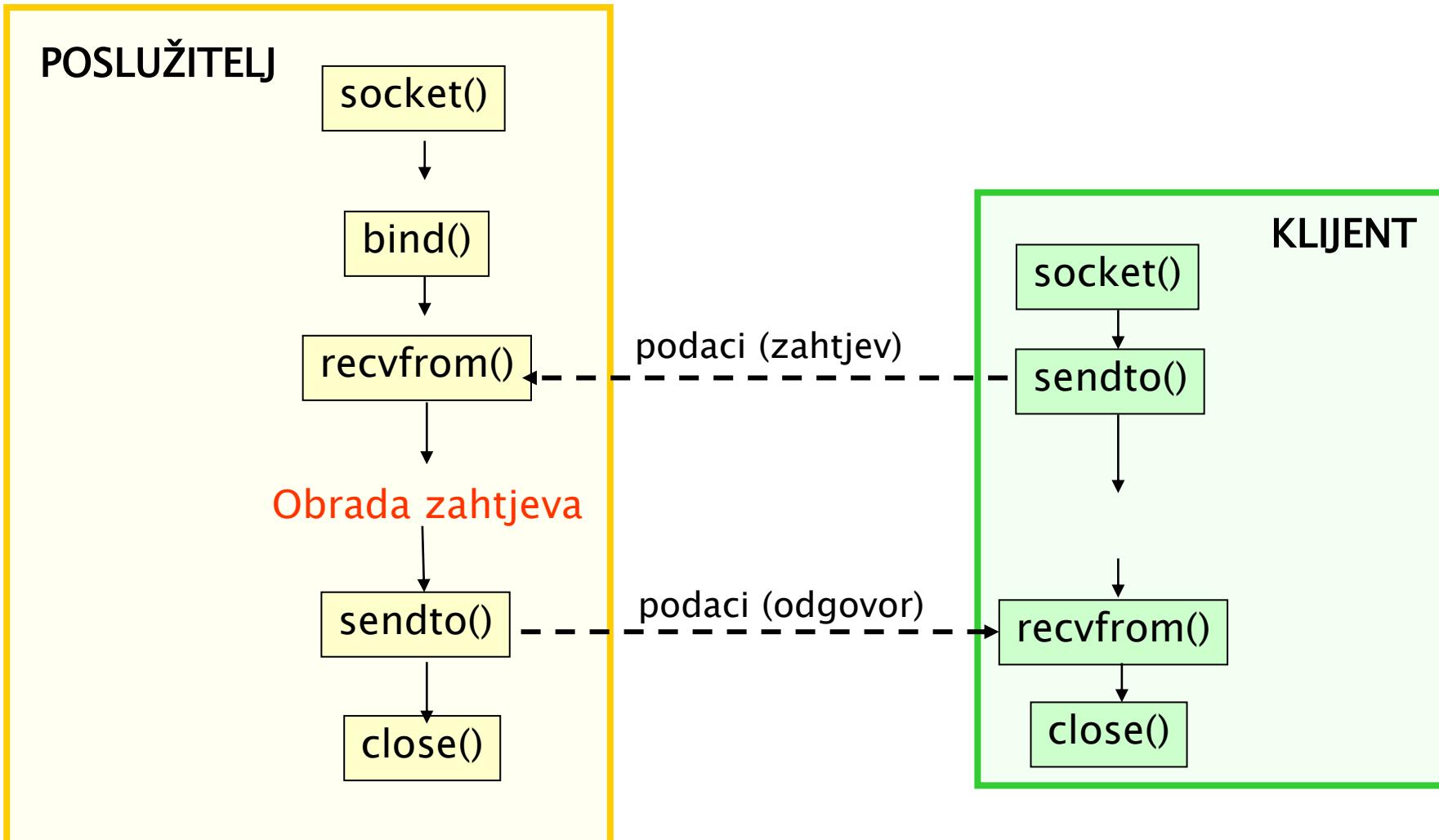
Transportni protokol UDP

User Datagram Protocol (UDP)

- komunikacija se odvija preko vrata (engl. *portova*) koje dodjeljuje operacijski sustav na strani klijenta, na strani poslužitelja se koriste “dobro poznata vrata”



Komunikacija pomoću socketa *UDP*



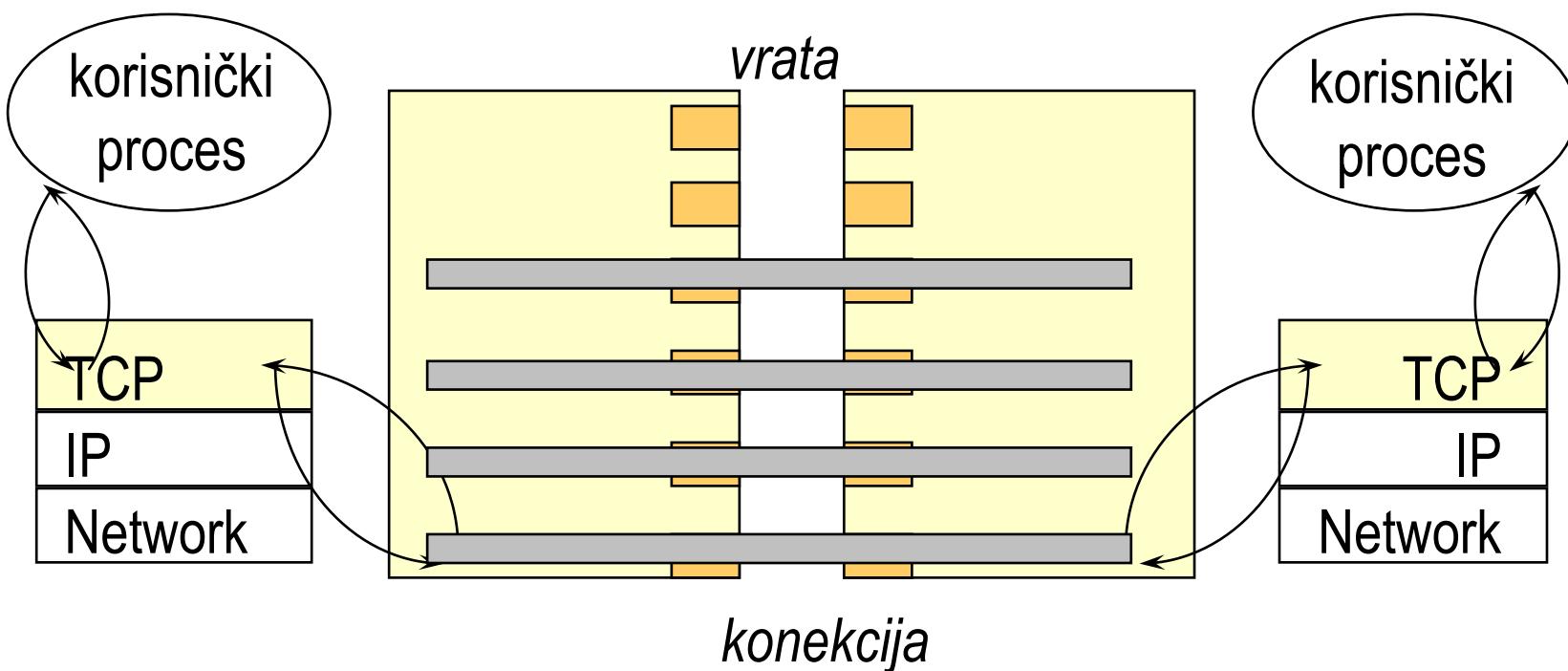
Obilježja socketa UDP

- model klijent-poslužitelj
- vremenska ovisnost procesa
 - poslužitelj mora biti aktivan za primanje datagrama
- klijent mora znati identifikator poslužitelja
- tranzijentna komunikacija
- asinkrona komunikacija
 - klijent šalje datagram i nastavlja obradu, nema blokiranja pošiljatelja
- nepouzdana komunikacija
- može se koristiti za implementaciju komunikacije na načelu *pull* i *push*

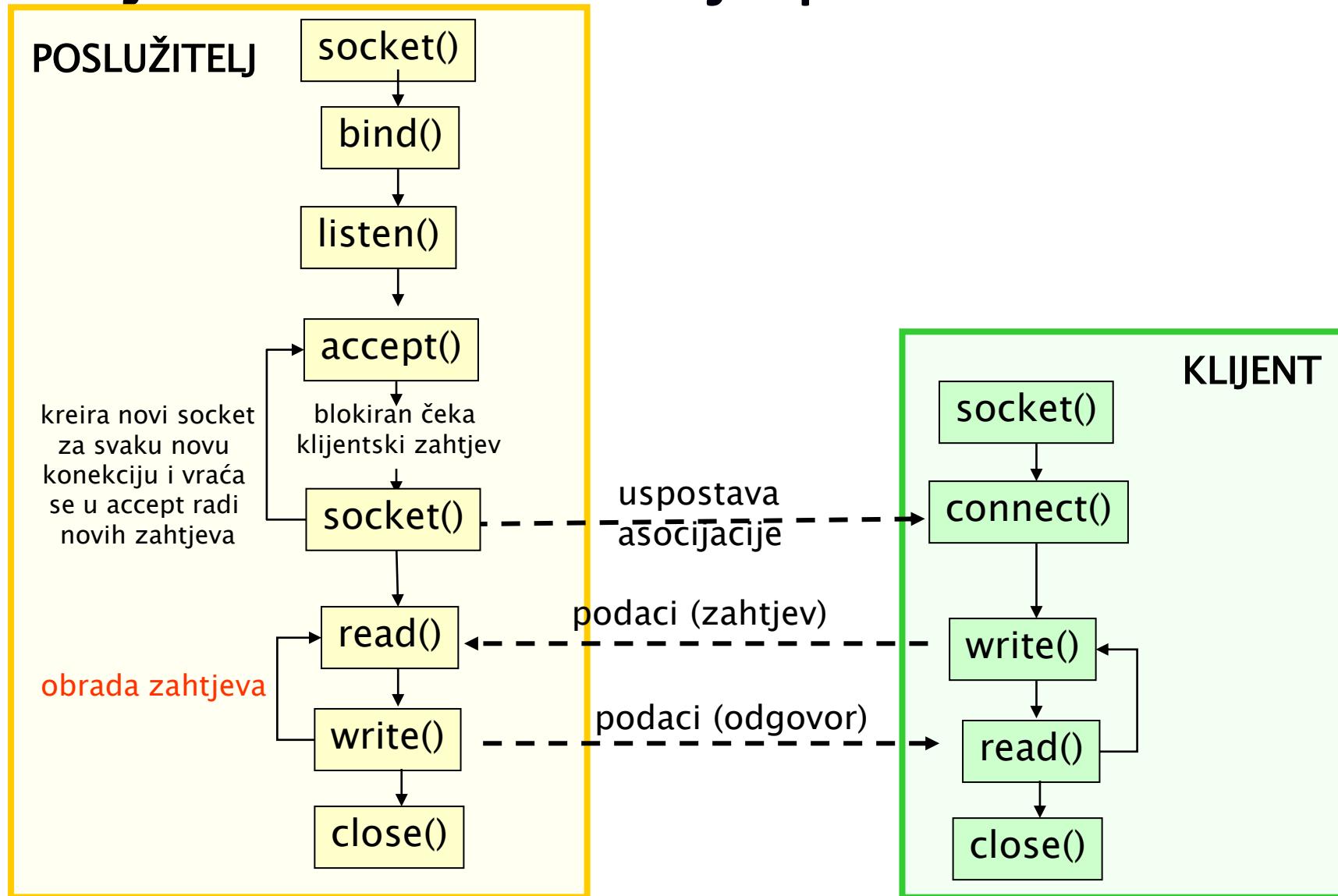
Transportni protocol TCP

Transmission Control Protocol (TCP)

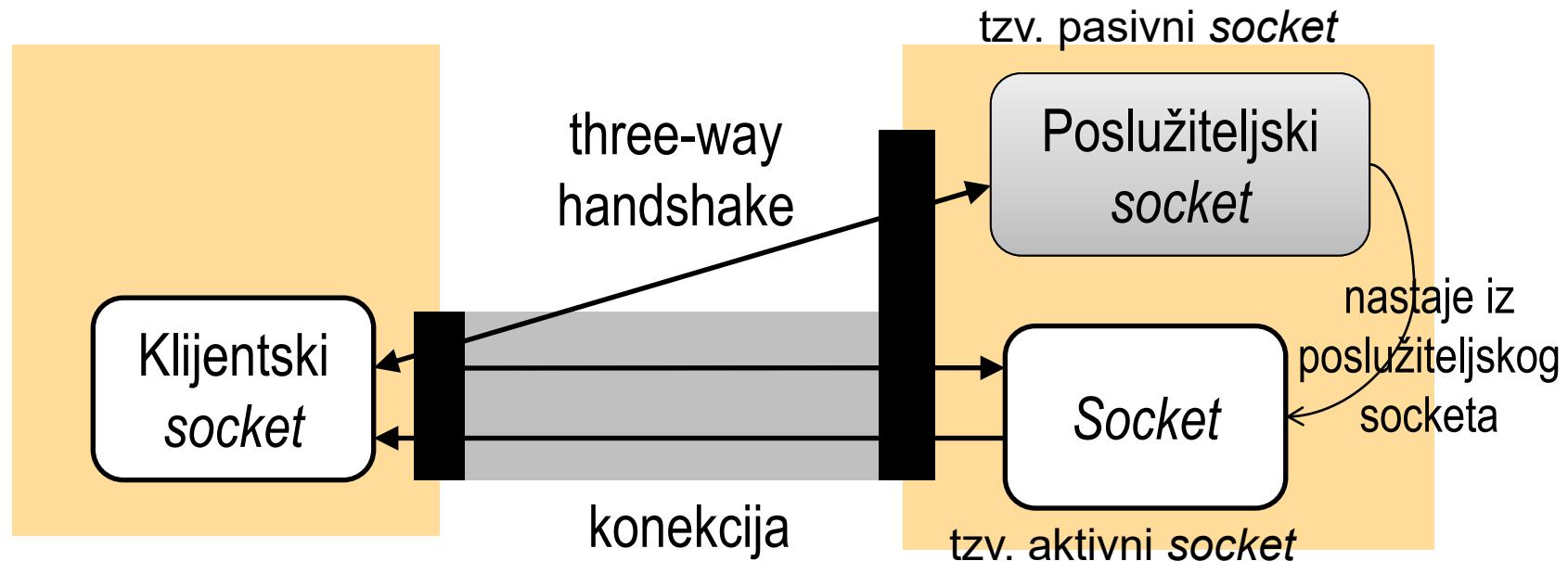
- konekcija između dvije krajnje točke koje se moraju dogovoriti o uspostavi konekcije



Konekcijska komunikacija pomoću socketa TCP



Konkurentni korisnički zahtjevi



- ◆ za svaki novi korisnički zahtjev kreira se **novi socket** (s dva *buffer-a*, *in* i *out*) koji se veže uz **konekciju** <poslužiteljska IP adresa i broj vrata (broj vrata ostaje isti kao za poslužiteljski socket), klijentska IP adresa i broj vrata>
- ◆ originalni poslužiteljski *socket* mora konstantno biti u stanju “osluškivanja”

Obilježja socketa TCP

- model klijent-poslužitelj
- vremenska ovisnost
 - klijent i poslužitelj moraju biti istovremeno dostupni
- klijent mora znati identifikator poslužitelja
- tranzijentna komunikacija
- sinkrona komunikacija
 - klijent šalje zahtjev za kreiranje konekcije i proces je blokiran do uspostave konekcije
- pokretanje komunikacije na načelu *pull*

Sadržaj predavanja

- Osnovni komunikacijski model
 - obilježja komunikacije
 - obilježja procesa
- Sloj raspodijeljenog sustava za komunikaciju među procesima
 - komunikacija korištenjem priključnica (Socket API)
 - primjeri TCP/UDP klijenta i poslužitelja
 - oblikovanje višedretvenog poslužitelja
 - poziv udaljene procedure (*Remote Procedure Call - RPC*) / poziv udaljene metode (*Remote Method Invocation - RMI*)
 - Java RMI
 - gRPC

UDP: implementacija poslužitelja

1. Kreirati socket poslužitelja:

```
DatagramSocket serverSocket;  
serverSocket = new DatagramSocket( PORT );
```

2. Kreirati paket (prazan, priprema za primanje):

```
byte[] rcvBuf = new byte[256];  
DatagramPacket packet =  
    new DatagramPacket(rcvBuf, rcvBuf.length);
```

3. Čekati korisnički paket (blokira proces do klijentskog zahtjeva!):

```
serverSocket.receive( packet );
```

4. Obrada pristiglog paketa i po potrebi odgovor klijentu

5. Zatvoriti socket (gasi poslužitelja):

```
serverSocket.close();
```

UDP: implementacija klijenta

1. Kreirati socket:

```
DatagramSocket clientSocket;  
clientSocket = new DatagramSocket();
```

2. Kreirati paket i napuniti ga podacima:

```
byte[] sendBuf = new byte[256];  
DatagramPacket packet =  
    new DatagramPacket(sendBuf, sendBuf.length, destAddress,  
destPort);
```

3. Slanje paketa:

```
clientSocket.send( packet );
```

4. Po potrebi obrada i čekanje odgovora

5. Zatvoriti socket:

```
clientSocket.close();
```

TCP: implementacija poslužitelja

1. Kreirati socket poslužitelja:

```
ServerSocket serverSocket;  
serverSocket = new ServerSocket( PORT );
```

2. Čekati korisnički zahtjev (blokira proces do klijentskog zahtjeva!!!) i kreirati kopiju originalnog socketa:

```
Socket copySocket = serverSocket.accept();
```

3. Kreirati I/O stream za komunikaciju s klijentom

```
DataInputStream is = new DataInputStream( copySocket.getInputStream() );  
DataOutputStream os = new DataOutputStream( copySocket.getOutputStream() );
```

4. Komunikacija s klijentom

5. Zatvoriti kopiju socketa:

```
copySocket.close();
```

6. Zatvoriti poslužiteljski socket:

```
serverSocket.close();
```

TCP: implementacija klijenta

1. Kreirati klijentski socket:

```
clientSocket = new Socket( address, port );
```

2. Kreirati I/O stream za komunikaciju s poslužiteljem:

```
is = new DataInputStream( clientSocket.getInputStream() );
os = new DataOutputStream( clientSocket.getOutputStream() );
```

3. Komunikacija s poslužiteljem:

- //Receive data from server:

```
String line = is.readLine();
```
- //Send data to server:

```
os.writeBytes("Hello\n");
```

4. Zatvoriti socket:

```
clientSocket.close();
```

Paket java.net

- API specification

<https://docs.oracle.com/javase/8/docs/api/java/net/package-summary.html>

- Osnovne klase

- Socket, ServerSocket, URL, URLConnection, (koriste TCP)
 - DatagramPacket, DatagramSocket, MulticastSocket (koriste UDP)

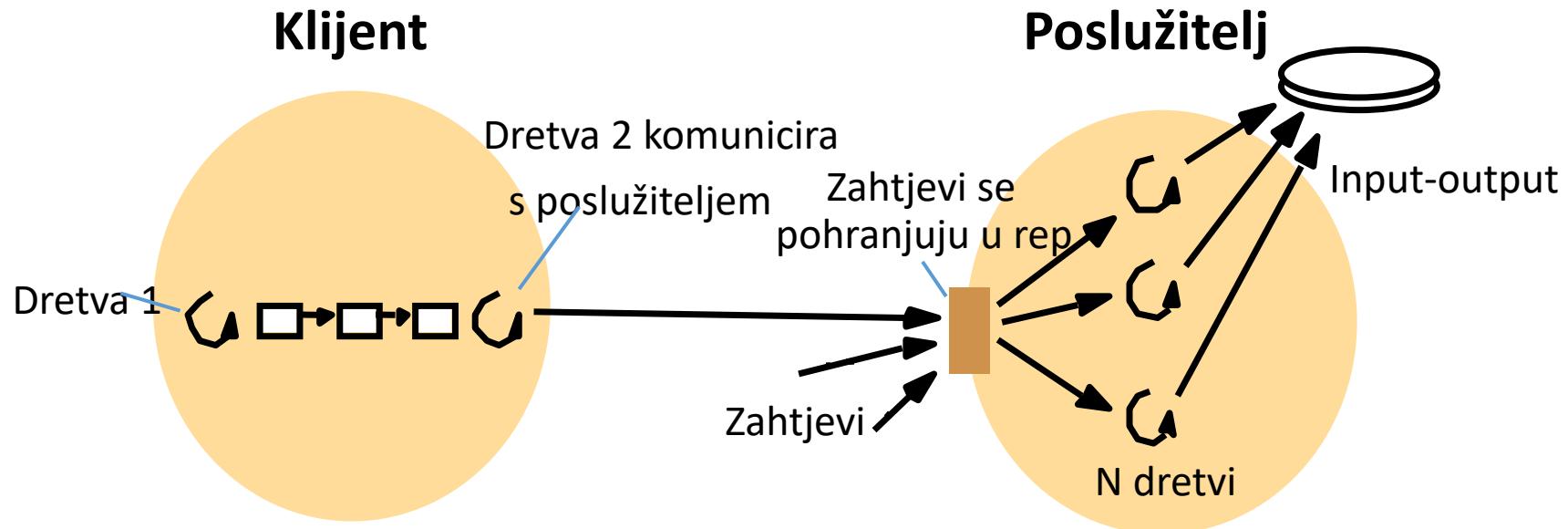
- Java Networking Tutorial

<http://docs.oracle.com/javase/tutorial/networking/>

Sadržaj predavanja

- Osnovni komunikacijski model
 - obilježja komunikacije
 - obilježja procesa
- Sloj raspodijeljenog sustava za komunikaciju među procesima
 - komunikacija korištenjem priključnica (Socket API)
 - primjeri TCP/UDP klijenta i poslužitelja
 - **oblikovanje višedretvenog poslužitelja**
 - poziv udaljene procedure (*Remote Procedure Call - RPC*) / poziv udaljene metode (*Remote Method Invocation - RMI*)
 - Java RMI
 - gRPC

Višedretveni poslužitelj i klijenti



Uobičajene zadaće na strani klijenta:

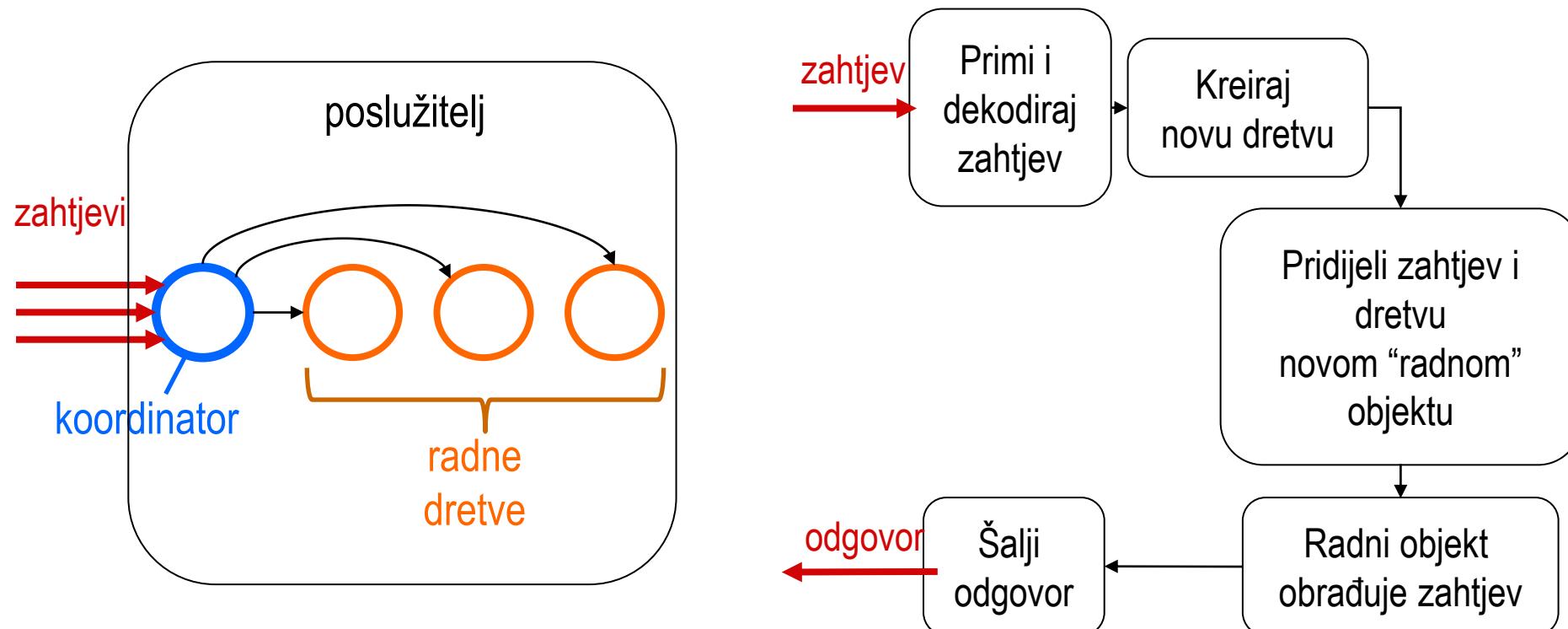
- korisničko sučelje,
- otvaranje mrežne konekcije i primanje podataka

Uobičajene zadaće na strani poslužitelja:

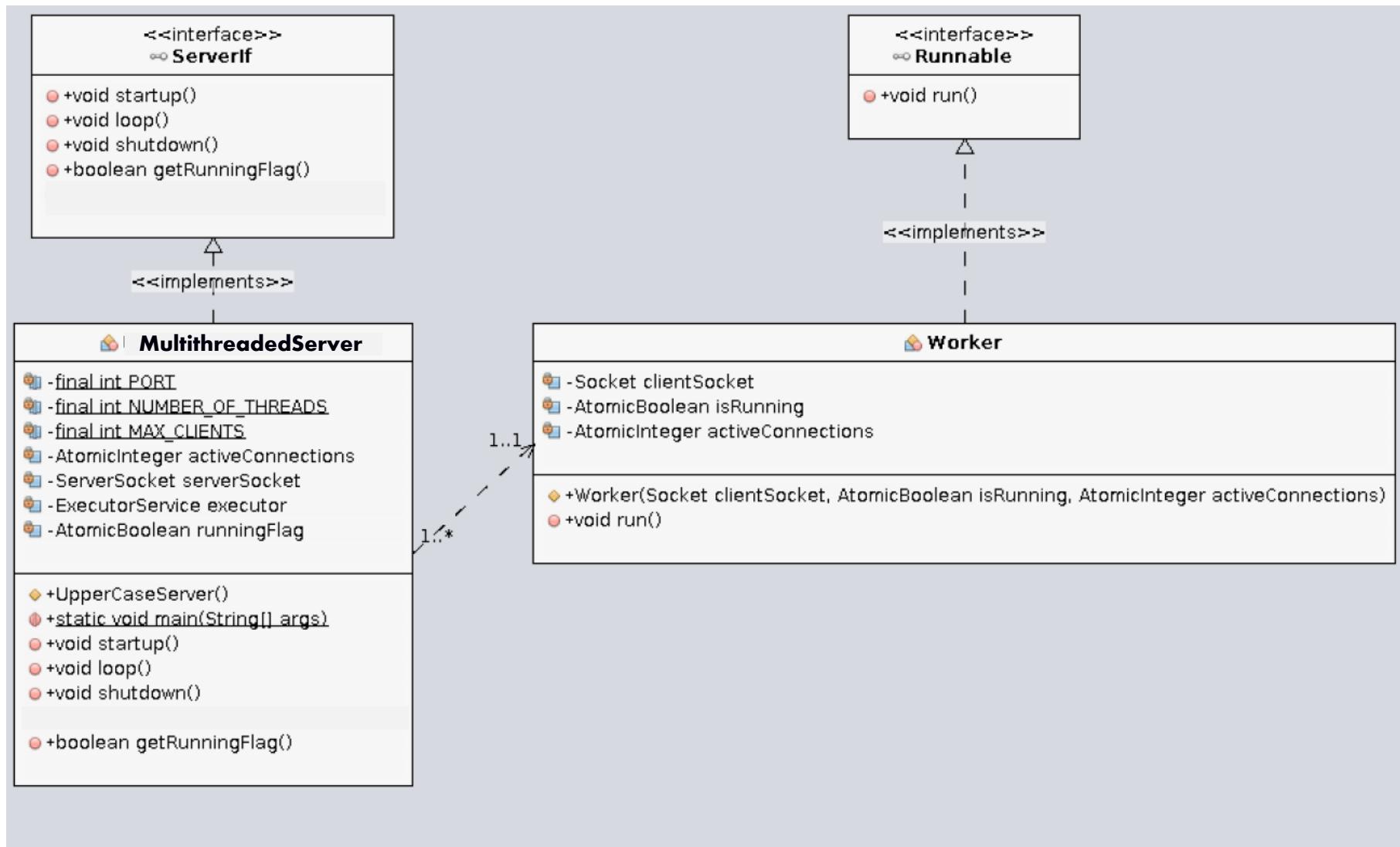
- primanje konkurentnih klijentskih zahtjeva
- složena obrada podataka
- rad s diskom/bazom podataka

Višedretveni poslužitelj

Model koordinator/radna dretva (*dispatcher/worker model*)



Primjer višedretvenog poslužitelja



Sučelje višedretvenog poslužitelja

```
public interface ServerIf {
    // Server startup. Starts all services offered by the server.
    public void startup();

    // Server loops when in running mode. The server must be active
    // to accept client requests.
    public void loop();

    // Server shutdown. Shuts down all services started during
    // startup.
    public void shutdown();

    // Gets the running flag that indicates server running status.
    // @return running flag
    public boolean getRunningFlag();
}
```

Poslužitelj (1)

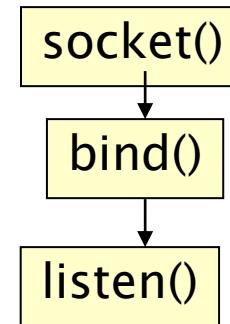
```
public class MultithreadedServer implements ServerIf {  
  
    private static final int PORT = 10002; // server port  
    private static final int NUMBER_OF_THREADS = 4;  
    //Max queue length for incoming connection requests.  
    private static final int BACKLOG = 10;  
  
    private final AtomicInteger activeConnections;  
    private ServerSocket serverSocket;  
    private final ExecutorService executor;  
    private final AtomicBoolean runningFlag;  
  
    ...
```

Poslužitelj (2)

```
...
public MultithreadedServer () {
    activeConnections = new AtomicInteger(0);
    executor = Executors.newFixedThreadPool(NUMBER_OF_THREADS);
    runningFlag = new AtomicBoolean(false);
}
public static void main(String[] args) {
    ServerIF server = new MultithreadedServer ();
    //start all required services
    server.startup();
    // run the main loop to accept client requests
    server.loop()
    //initiate shutdown when such request is received
    server.shutdown();
}
...
```

Poslužitelj (3)

```
//Starts all required server services.  
  
@Override  
  
public void startup() {  
    // create a server socket, bind it to the specified port  
    // on the local host and set the backlog for  
    // client requests  
    try {  
        this.serverSocket = new ServerSocket(PORT, BACKLOG);  
        // set timeout to avoid blocking  
        serverSocket.setSoTimeout(500);  
        runningFlag.set(true);  
        System.out.println("Server is ready!");  
    } catch (SocketException e1) {  
        System.err.println("Exception caught when setting server socket timeout: " +  
e1);  
    } catch (IOException ex) {  
        System.err.println("Exception caught when opening or setting the server  
socket: " + ex);  
    }  
}  
}...
```



Poslužitelj (4)

```
// The main loop for accepting client requests.  
  
@Override  
public void loop() {  
    while(runningFlag.get()) {  
        try{// create a new socket, accept and listen for a connection made to this socket  
            Socket clientSocket = serverSocket.accept(); accept()  
            // execute a tcp request handler in a new thread  
            Runnable worker = new Worker(clientSocket, runningFlag, activeConnections);  
            executor.execute(worker);  
            activeConnections.getAndIncrement();  
        } catch(SocketTimeoutException ste) {  
            // do nothing, check runningFlag  
        } catch(IOExceptionex) {  
            System.err.println("Exception caught when waiting for a connection: " + ex);  
        }  
    }  
}
```

Poslužitelj (5)

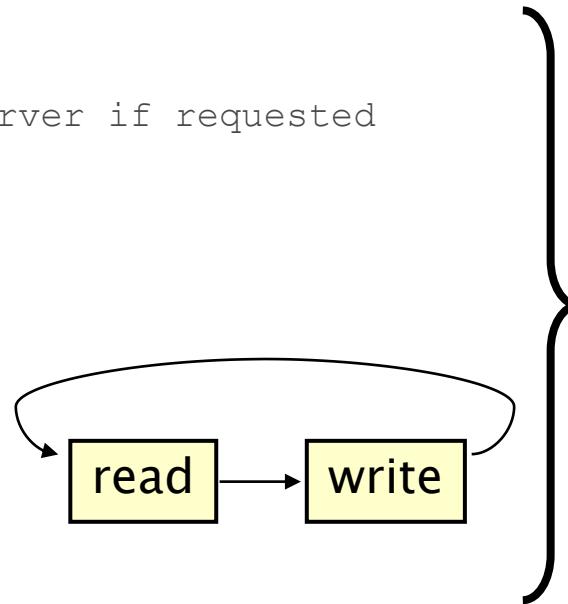
```
@Override  
public void shutdown() {  
    while( activeConnections.get() > 0 ) {  
        System.out.println( "WARNING: There are still active  
                           connections" );  
        try { Thread.sleep( 5000 );  
        } catch( java.lang.InterruptedException e ) {}  
    }  
    if( activeConnections.get() == 0 ) {  
        System.out.println( "Starting server shutdown." );  
        try { serverSocket.close(); close()  
        } catch (IOException e) {  
            System.err.println("Exception caught when closing the server socket: " + e);  
        } finally { executor.shutdown();  
        }  
        System.out.println("Server has been shutdown.");  
    }  
}
```

Worker (1)

```
public class Worker implements Runnable {  
    private final Socket clientSocket;  
    private final AtomicBoolean isRunning;  
    private final AtomicInteger activeConnections;  
    public Worker(Socket clientSocket, AtomicBoolean isRunning, AtomicInteger activeConnections)  
    {  
        this.clientSocket = clientSocket;  
        this.isRunning = isRunning;  
        this.activeConnections = activeConnections;  
    }  
    @Override  
    public void run() {  
        try ( //create a new BufferedReader from an existing InputStream  
            BufferedReader inFromClient = new BufferedReader(new  
                InputStreamReader(clientSocket.getInputStream()));  
            //create a PrintWriter from an existing OutputStream  
            PrintWriter outToClient = new PrintWriter(new  
                OutputStreamWriter(clientSocket.getOutputStream(), true);  
        )  
    }
```

Worker (2)

```
String receivedString;  
// read a few lines of text  
  
while ((receivedString=inFromClient.readLine()) != null {  
    System.out.println("Server received:" + receivedString);  
    if (receivedString.contains("shutdown")) {//shutdown the server if requested  
        outToClient.println("Initiating server shutdown!");  
        isRunning.set(false);  
        activeConnections.getAndDecrement();  
        return;  
    }  
    String stringToSend = receivedString.toUpperCase();  
    // send a String then terminate the line and flush  
    outToClient.println(stringToSend);  
    System.out.println("Server sent: " + stringToSend);  
}  
activeConnections.getAndDecrement();  
} catch (IOException ex) {  
    System.err.println("Exception caught when trying to read or send data: " + ex);
```

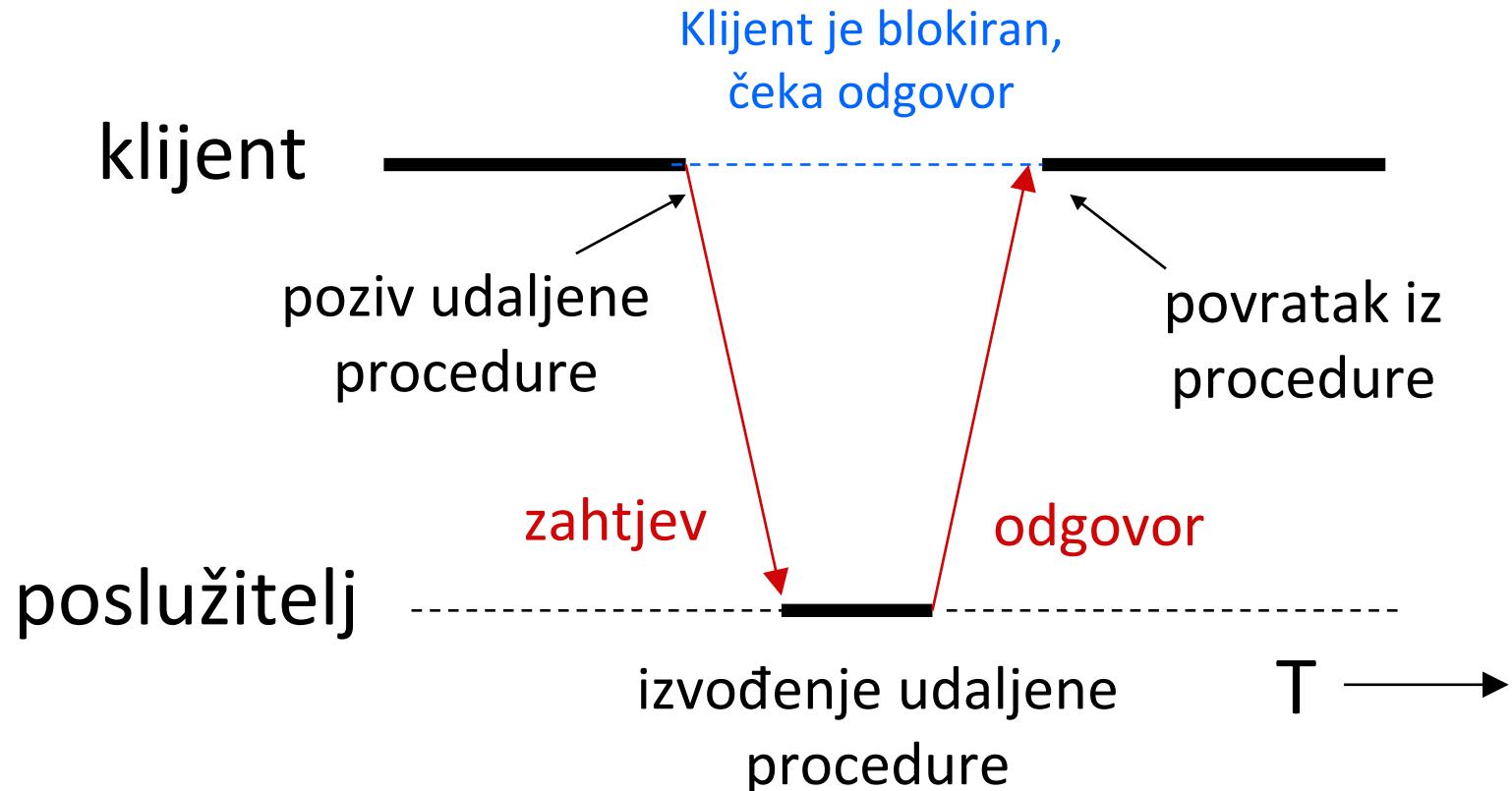


Sadržaj predavanja

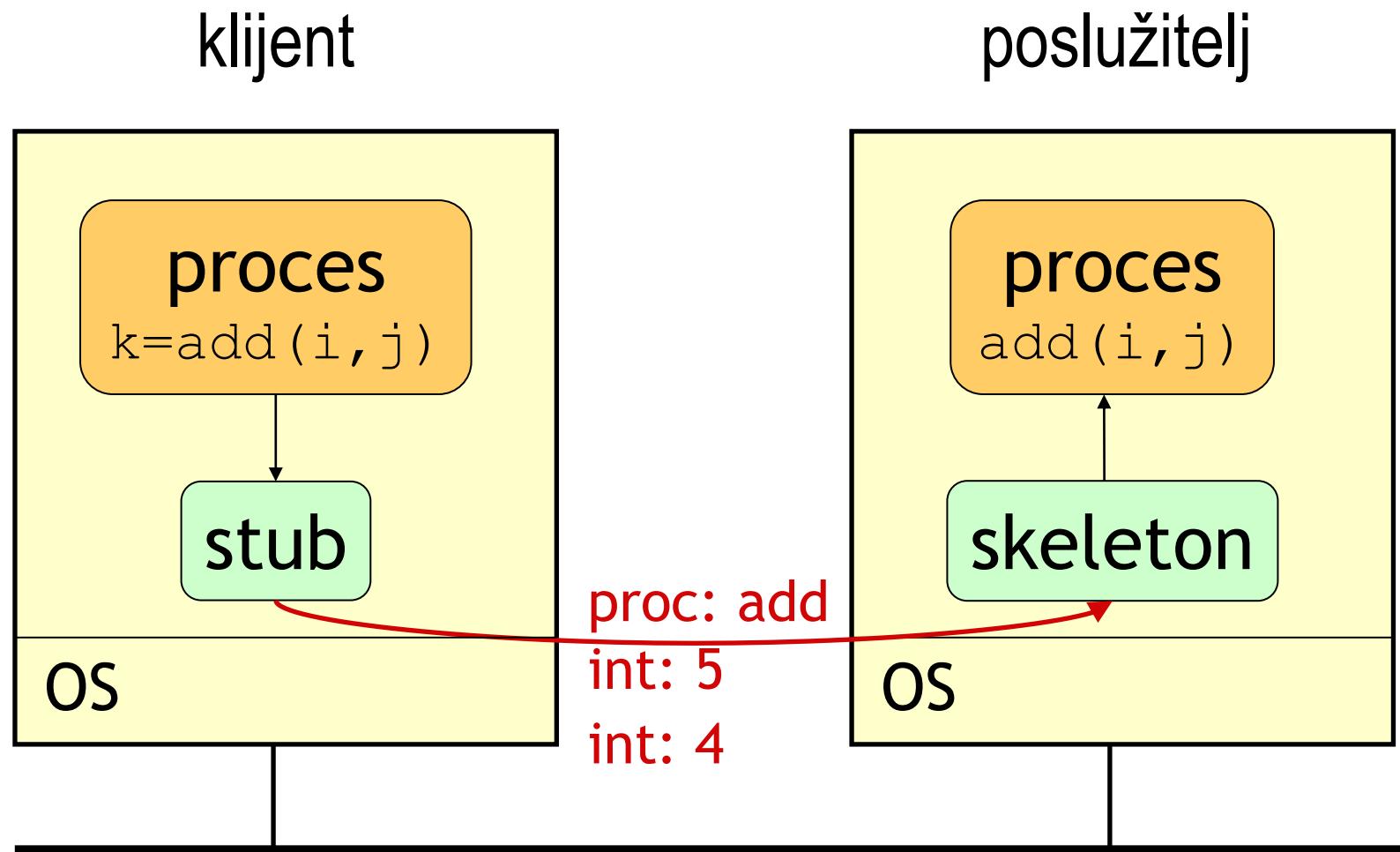
- Osnovni komunikacijski model
 - obilježja komunikacije
 - obilježja procesa
- Sloj raspodijeljenog sustava za komunikaciju među procesima
 - komunikacija korištenjem priključnica (Socket API)
 - primjeri TCP/UDP klijenta i poslužitelja
 - oblikovanje višedretvenog poslužitelja
 - poziv udaljene procedure (*Remote Procedure Call - RPC*) / poziv udaljene metode (*Remote Method Invocation - RMI*)
 - Java RMI
 - gRPC

Poziv udaljene procedure (RPC)

- Omogućuje procesima pozivanje i izvođenje procedura na udaljenom računalu.



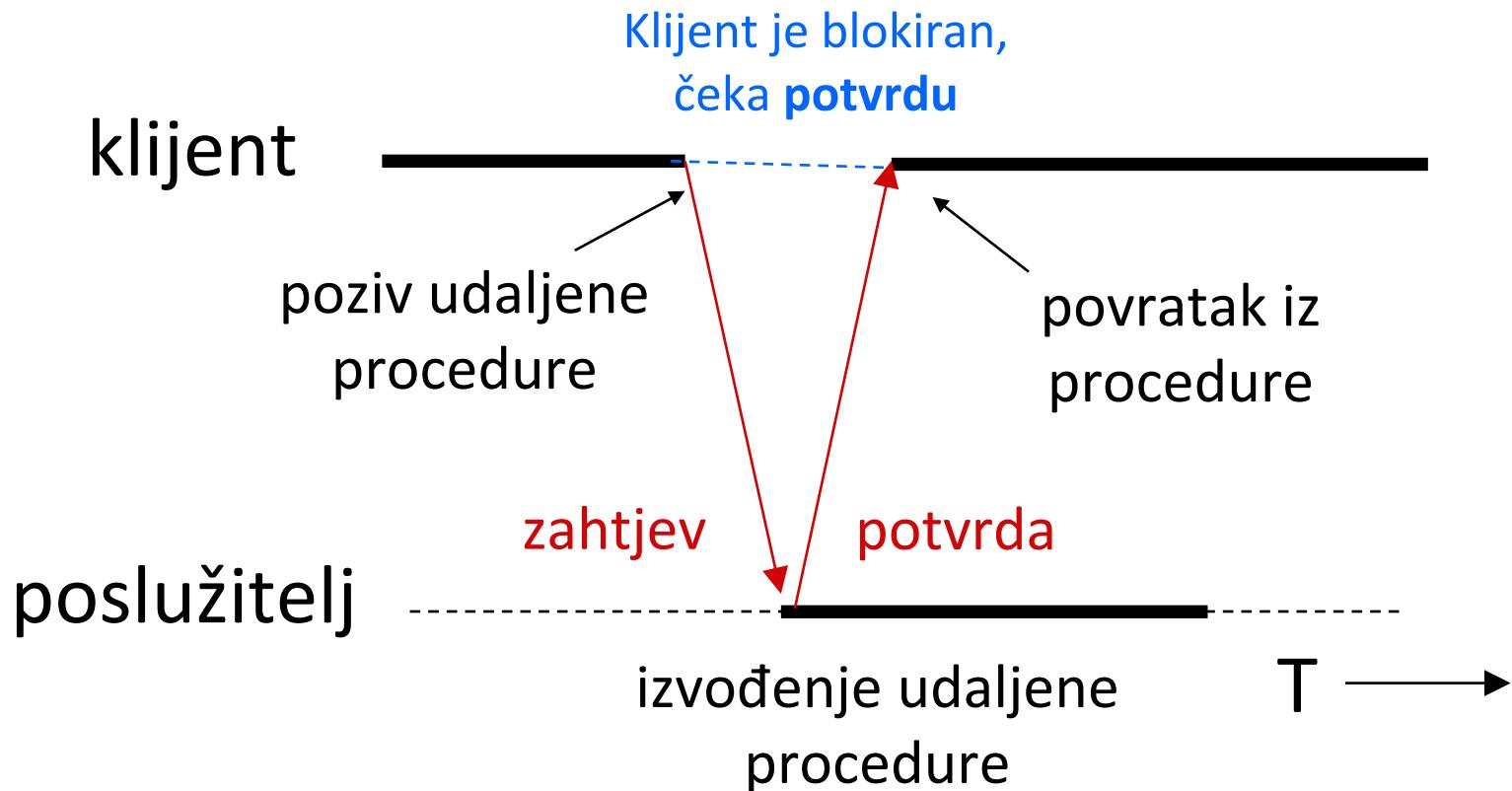
Izvođenje RPC-a



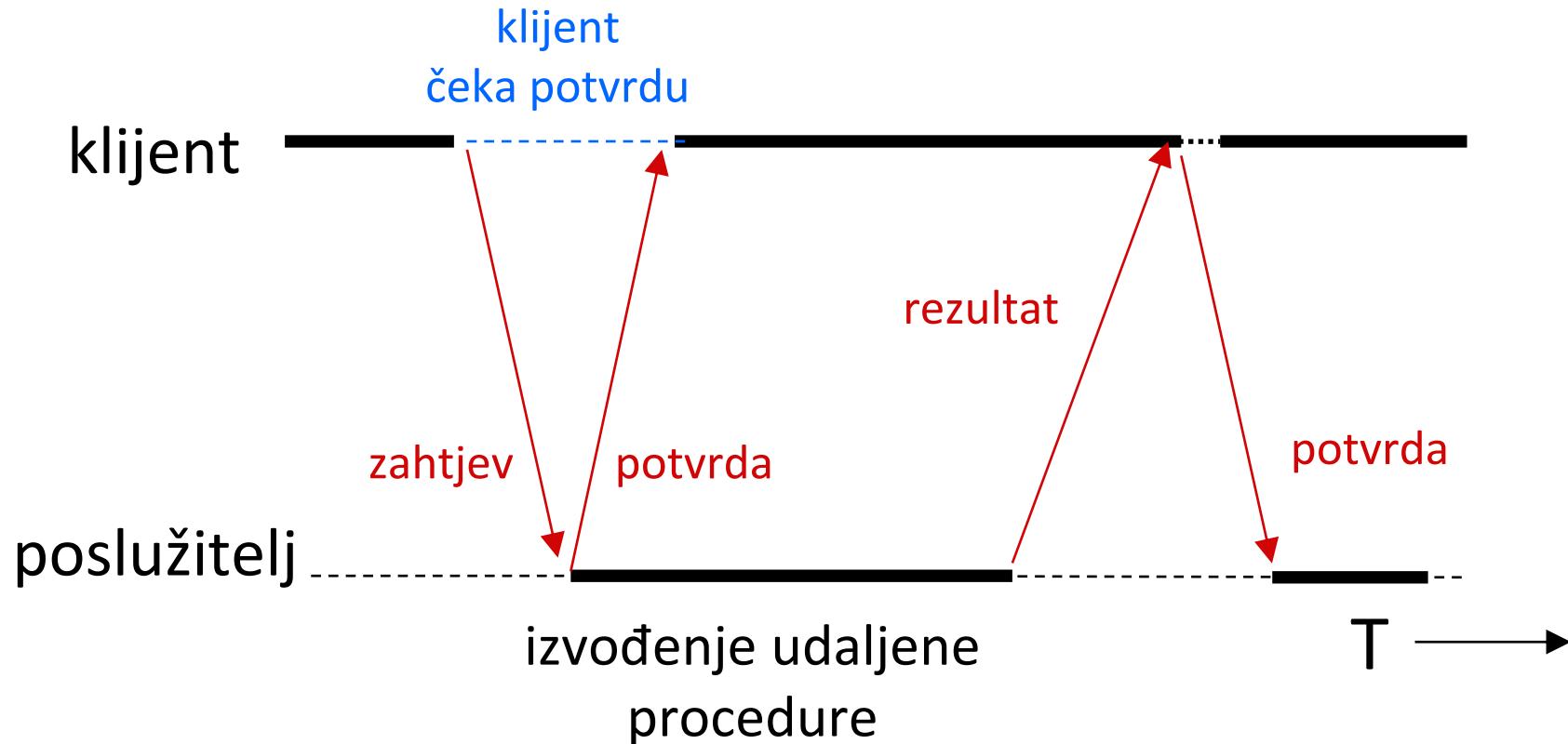
Prenošenje parametara

- *Marshaling* – “pakiranje” parametara ili rezultata u poruku
- *Unmarshaling* – čitanje parametara ili rezultata iz poruke
- Prenošenje vrijednosti parametra
 - Navodi se tip (npr. int, char, long) i vrijednost
 - Različiti OS-ovi često koriste različite prikaze znakova
- Prenošenje parametara koristeći reference
 - Referenca ima smisla samo u adresnom prostoru procesa koji je koristi!
 - Kako prenijeti string na udaljeno računalo?
 - nije moguće koristiti referencu na string!
 - kopiranje cijelog stringa i “pakiranje” u poruku

Asinkroni RPC



Odgodjeni sinkroni RPC



Poziv udaljene metode

Remote Method Invocation (RMI)

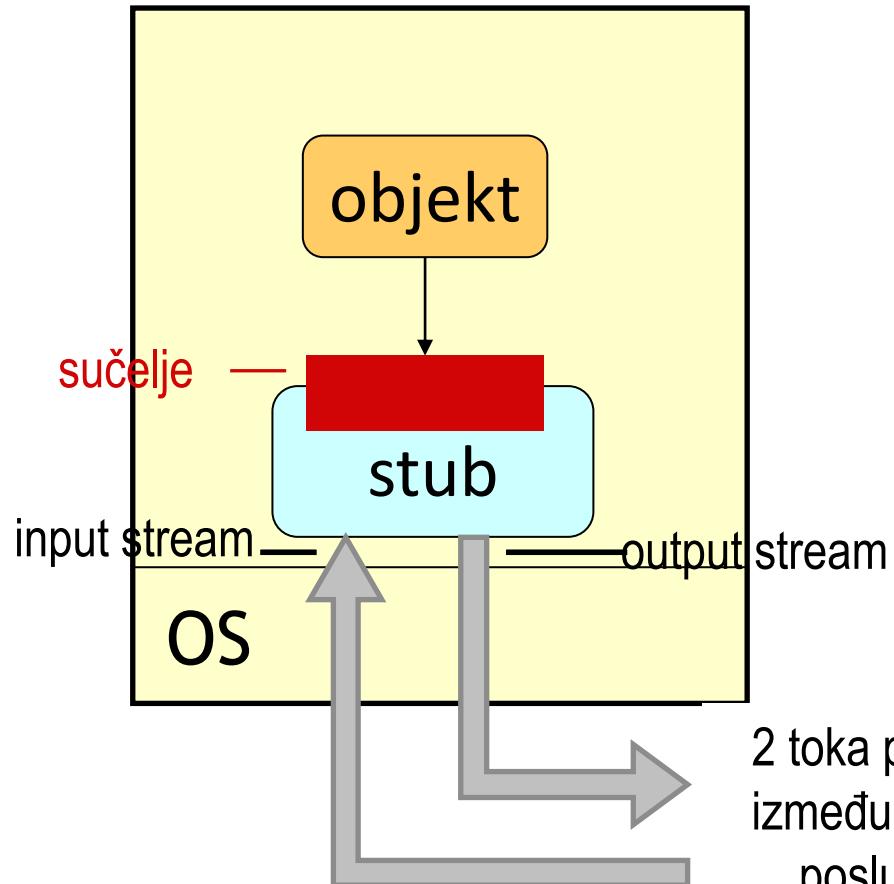
- “nasljednik” poziva udaljene procedure, poziva se metoda udaljenog objekta
- udaljeni objekt
 - proširenje osnovnog objektnog modela za raspodijeljenu okolinu
 - odvajanje sučelja i implementacije objekta
- objekt (klijent) poziva metodu udaljenog objekta (poslužitelja) na transparentan način
 - identično pozivu metode lokalnog objekta

Udaljeni objekti

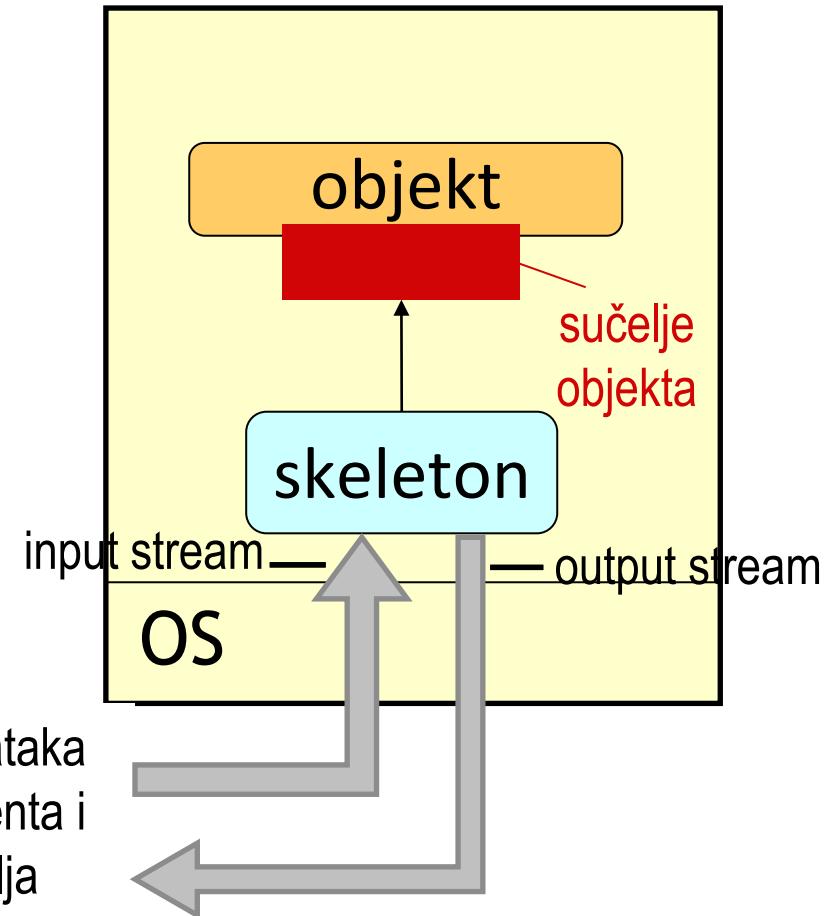
- Postoje reference na lokalne i udaljene objekte
- Svaki udaljeni objekt ima globalno jedinstven identifikator
 - npr. [ref: [endpoint:[161.53.19.24:1251](local),objID:[0]]]]
- Potrebna je usluga za registriranje i pronalaženje udaljenih objekata (*directory service*)

Izvođenje RMI-a

klijent



poslužitelj



2 toka podataka
između klijenta i
poslužitelja

Obilježja RPC/RMI

- model klijent-poslužitelj
- vremenska ovisnost klijenta i poslužitelja
- klijent mora znati identifikator poslužitelja
- tranzijentna komunikacija
- sinkrona komunikacija
 - klijent je blokiran dok ne primi odgovor od strane poslužitelja
- pokretanje komunikacije na načelu *pull*

Sadržaj predavanja

- Osnovni komunikacijski model
 - obilježja komunikacije
 - obilježja procesa
- Sloj raspodijeljenog sustava za komunikaciju među procesima
 - komunikacija korištenjem priključnica (Socket API)
 - primjeri TCP/UDP klijenta i poslužitelja
 - oblikovanje višedretvenog poslužitelja
 - poziv udaljene procedure (*Remote Procedure Call - RPC*) / poziv udaljene metode (*Remote Method Invocation - RMI*)
 - Java RMI
 - gRPC

Java RMI

Java Remote Method Invocation

Sunovo rješenje za komunikaciju udaljenih objekata na načelu poziva udaljene procedure/metode

Oblikovan isključivo za programski jezik Java: omogućuje jednostavniju komunikaciju objekata koji se izvode u različitim JVM (*Java Virtual Machine*)

Implementacija koristi TCP kao transportni protokol

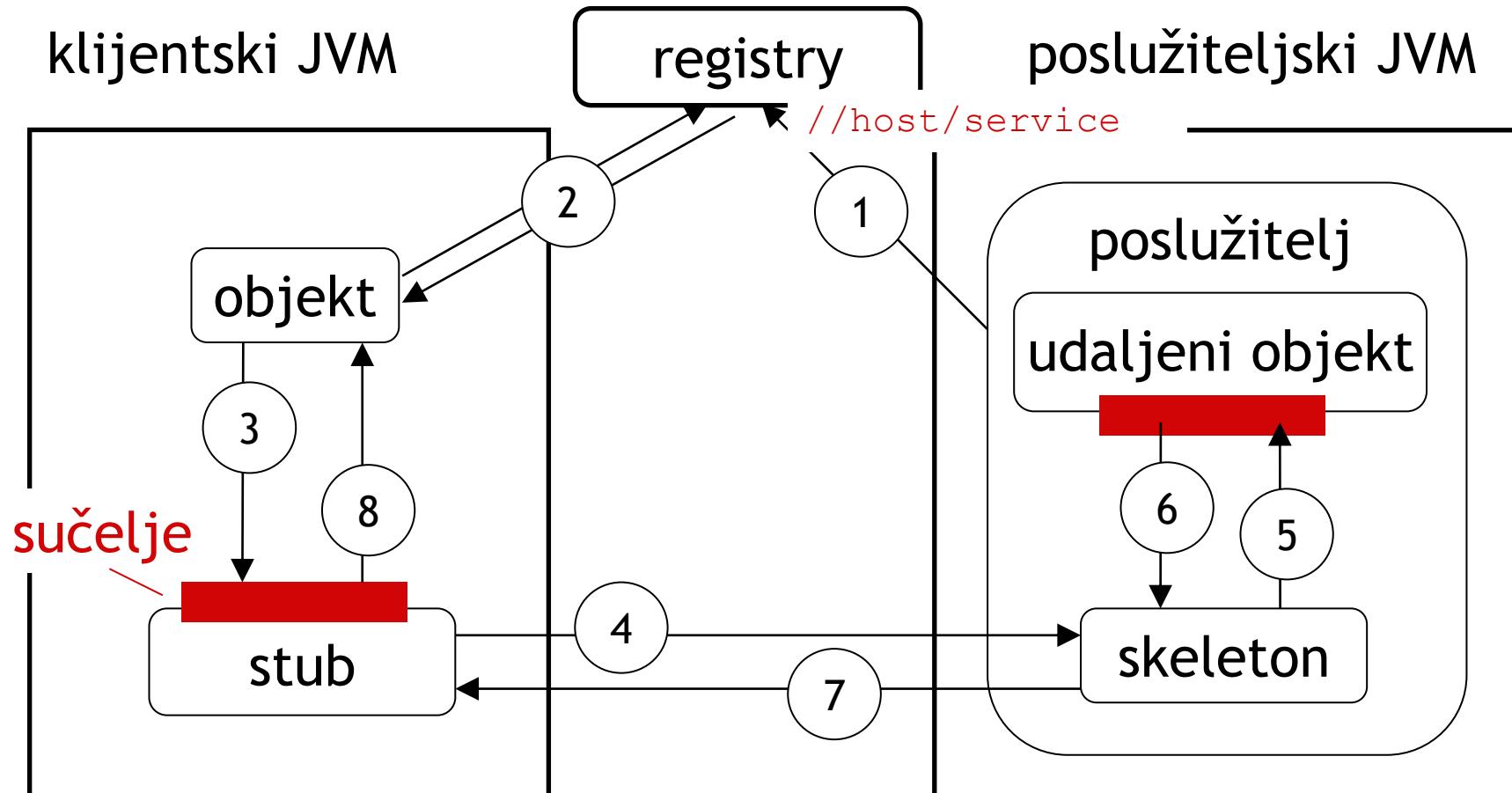
Javni model objekta

- transparentnost pristupa udaljenim objektima
 - referenca na udaljeni objekt istovjetna je referenci na lokalni objekt, no moraju implementirati sučelje `java.rmi.Remote`
- sučelja udaljenog objekta omogućuju komunikaciju s udaljenim objektom
- sučelje udaljenog objekta implementira *stub (proxy)* u adresnom prostoru klijentskog računala
- klase *stub* i *skeleton* generiraju se iz implementacije, a ne iz sučelja udaljenog objekta

Prenošenje parametara u daljenoj metodi

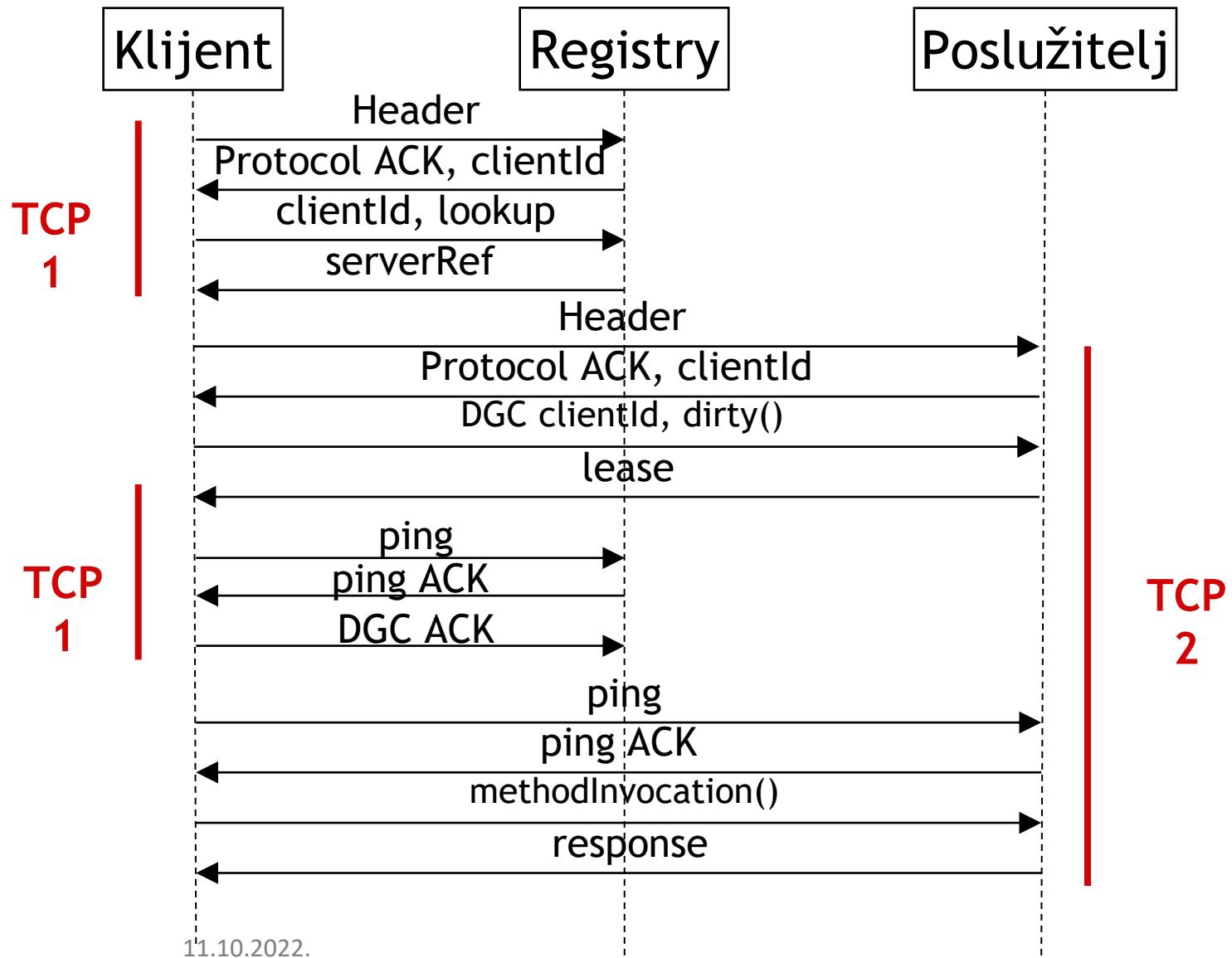
- lokalni objekti moraju se serijalizirati i prenosi se njihova vrijednost (*pass by value*)
 - implementiraju sučelje Serializable
- udaljeni se objekti prenose koristeći referencu (*pass by reference*)
 - implementiraju sučelje java.rmi.Remote i pravilno su eksportirani UnicastRemoteObject.exportObject()
 - referenca = adresa računala + port + identifikator udaljenog objekta
 - referenca udaljenog objekta je jedinstvena u raspodijeljenom sustavu

Protokol Java RMI (1)

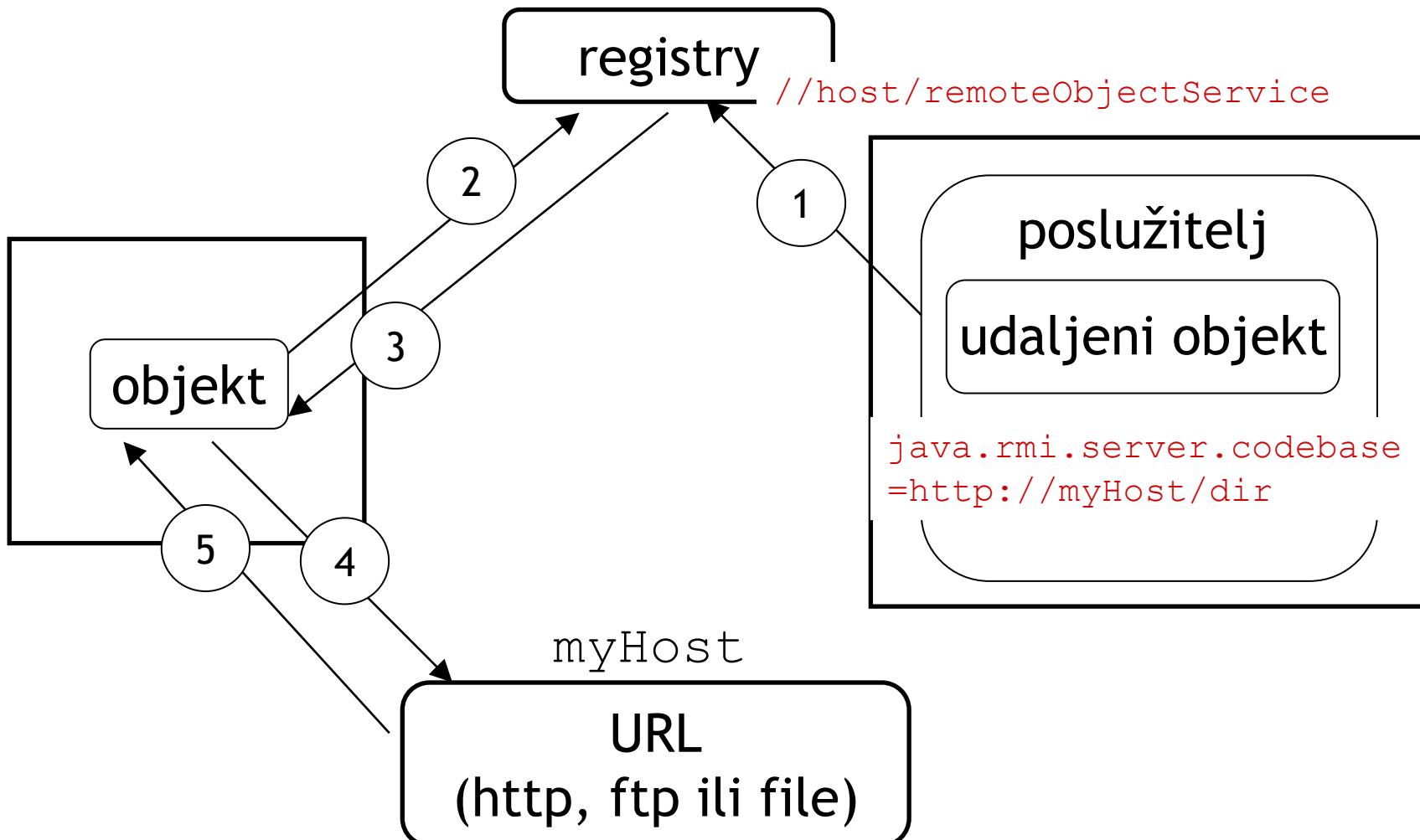


Prepostavka: stub postoji na strani klijenta

Protokol Java RMI (2)



Dinamičko učitavanje klase stuba



Primjer RMI sučelja

```
import java.rmi.RemoteException;
import java.rmi.Remote;

/**
 * Remote object offers the service of converting a string
 * to upper case.
 */
public interface Uppercase extends Remote {

    public String uppercase
        (String originalString) throws RemoteException;

}
```

Primjer RMI poslužitelja (1)

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;

public class UpperCaseImpl extends UnicastRemoteObject
    implements UpperCase {
    private static final String rmiUrl = "rmi://localhost:1099/UpperCase4U";
    public UpperCaseImpl() throws RemoteException {
        super();
    }
    public String toUpperCase( String originalString )
        throws RemoteException {
        return( originalString.toUpperCase() );
    }
}
```

Primjer RMI poslužitelja (2)

```
...
public static void main(String[] args) {
    try {
        if (System.getSecurityManager() == null)
            System.setSecurityManager(
                new RMISecurityManager());
        UpperCaseImpl serverObject = new UpperCaseImpl();
        Naming.rebind(rmiUrl, serverObject);
        System.out.println("UpperCase object bound to " + rmiUrl);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Primjer RMI klijenta (1)

```
import java.rmi.RemoteException;
import java.rmi.NotBoundException;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;

public class UpperCaseClient {
    private static final String rmiUrl = "rmi://localhost:1099/UpperCase4U";
    private UpperCase uc = null;
    public UpperCaseClient() {
        try { uc = (UpperCase) Naming.lookup( rmiUrl );
            System.out.println( "Found remote object " + uc.toString() );
        } catch( Exception e ) {
            e.printStackTrace();
        }
    }
}
```

Primjer RMI klijenta (2)

```
...
public static void main(String[] args) {
    if (System.getSecurityManager() == null)
        System.setSecurityManager(new RMISecurityManager());
    UpperCaseClient client = new UpperCaseClient();
    try {
        String any = new String( "Any string..." );
        System.out.println( "Sending\t" + any );
        System.out.println("Received\t"
                           + client.uc.toUpperCase(any));
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    System.exit(0);
} }
```

Obilježja Java RMI-a

- pozitivna svojstva
 - visok nivo transparentnosti
 - poziv udaljene metode ima jednaku sintaksu pozivu lokalne metode
 - podržava dinamičko učitavanje klasa
 - jednostavna i brza implementacija raspodijeljenog sustava
 - jednostavniji i čitljiviji kod programa
- negativna svojstva
 - performanse: poziv udaljene metode je puno sporiji od poziva metode lokalnog objekta, čak i ako su udaljeni objekt i klijent na istom računalu (TCP + dizajn protokola s velikim brojem ping paketa)

Paket java.rmi

- **API specification**
<http://docs.oracle.com/javase/8/docs/api/java/rmi/package-summary.html>
- **The Java Tutorials, Trail: RMI**
<http://docs.oracle.com/javase/tutorial/rmi/>
- **Java Remote Method Invocation - Distributed Computing for Java**
<http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html>

Sadržaj predavanja

- Osnovni komunikacijski model
 - obilježja komunikacije
 - obilježja procesa
- Sloj raspodijeljenog sustava za komunikaciju među procesima
 - komunikacija korištenjem priključnica (Socket API)
 - primjeri TCP/UDP klijenta i poslužitelja
 - oblikovanje višedretvenog poslužitelja
 - poziv udaljene procedure (*Remote Procedure Call - RPC*) / poziv udaljene metode (*Remote Method Invocation - RMI*)
 - Java RMI
 - gRPC

gRPC

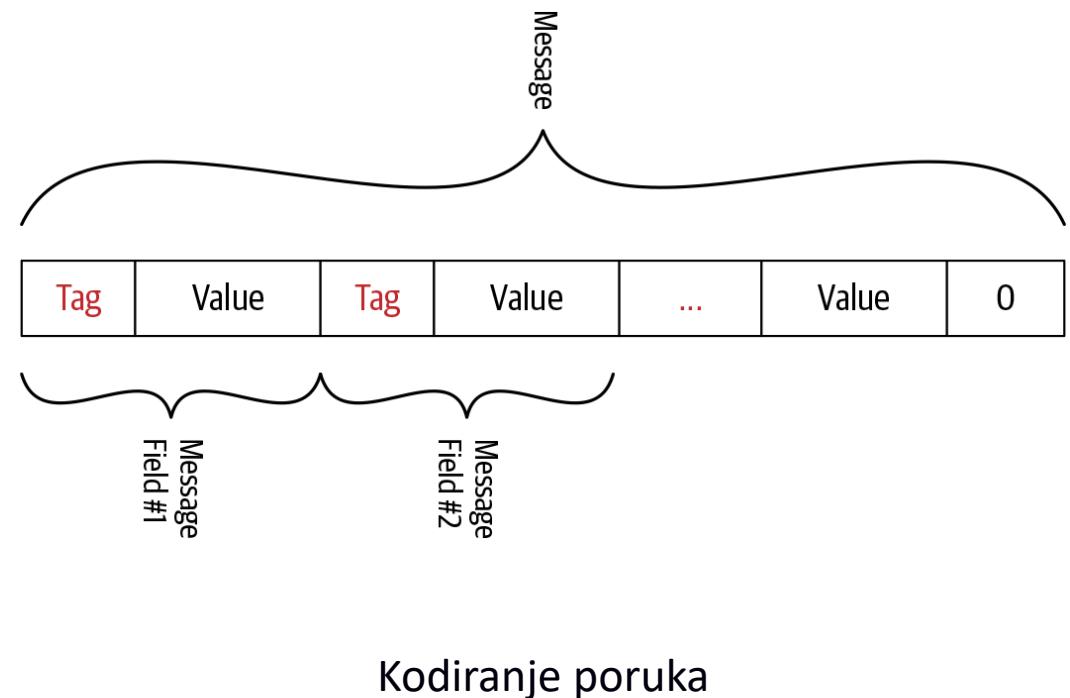
<https://grpc.io/>

- implementacija RPC-a **otvorenog koda** nastala na temelju Googleovog projekta Stubby
- podržava niz jezika: Java, Go, C++, Python...
- koristi posebnu implementaciju za prijenos i serijalizaciju podataka Protocol Buffers (HTTP/2)

- Od 2015.: *open source RPC framework*
- “*... a modern, bandwidth and CPU efficient, low latency way to create massively distributed systems that span data centers*”
- Danas ima raširenu uporabu: Netflix, Square, Lyft, Docker, Cisco, CoreOS
- Cloud Native Computing Foundation (CNCF), [https://www.cncf.io/projects/\(incubating\)](https://www.cncf.io/projects/(incubating))

Protocol buffers

- IDL za definiranje sučelja: piše se u tekstualnu datoteku .proto (koristeći jednostavni format za definiranje RPC metoda i njihovih parametara)
- Protokol je neovisan o programskom jeziku
- Vrlo učinkovit mehanizam serijalizacije podataka, binarno kodiranje (poruke su značajno manje i jednostavnije za računalnu obradu od JSON-a)



Primjer ProductInfo.proto

```
syntax = "proto3";

package ecommerce;

service ProductInfo {

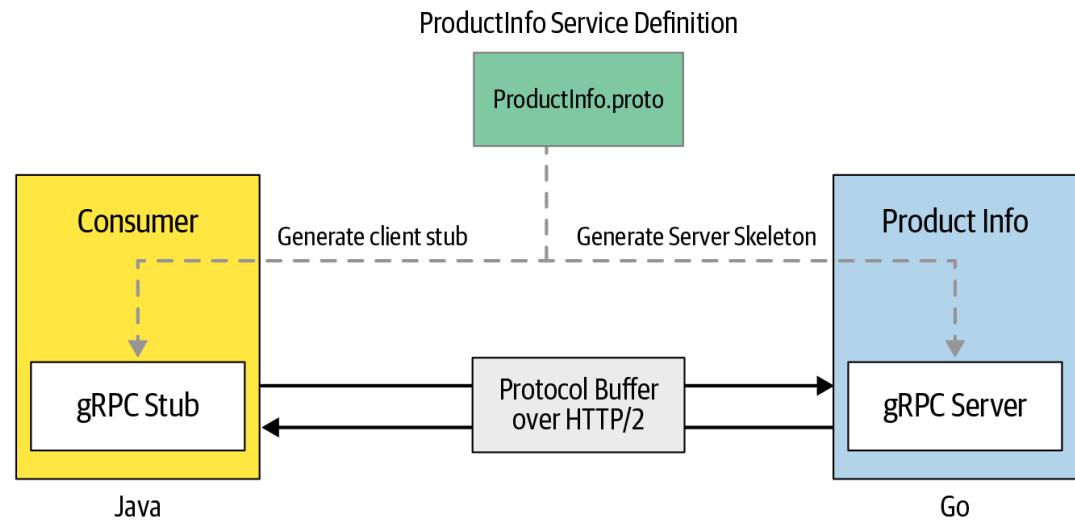
    rpc addProduct(Product) returns (ProductID);

    rpc getProduct(ProductID) returns (Product);
}

message Product {
    string id = 1;
    string name = 2;
    string description = 3;
}

message ProductID {
    string value = 1;
}
```

Koristi se kompjajler *protoc* za generiranje poslužiteljskog i klijentskog koda



Primjer gRPC for UpperCase

```
syntax = "proto3";
option java_multiple_files = true;
option java_package = "hr.fer.tel.rassus.examples";
package hr.fer.tel.rassus;

// The uppercase service definition.
service Uppercase {
    // Sends back a message converted to upper case
    rpc RequestUppercase (Message) returns (Message) {}
}

// Definition of request and response message
message Message {
    string payload = 1;
}
```

Primjer definicije
sučelja servisa
Uppercase koje definira
samo jednu metodu
RPC RequestUppercase

Koraci prilikom implementacije i izvođenja

Poslužitelj

1. generiraj skeleton servisa na temelju .proto opisa
2. dodaj logiku generiranim metodama (*override*)
3. pokreni poslužitelja koji će kontinuirano primati korisničke zahtjeve

Klijent

1. kreiraj konekciju (kanal) prema udaljenom poslužitelju
2. poveži klijentski stub uz tu konekciju
3. pozovi udaljenu metodu na poslužitelju, a implementacija Protocol Buffers će se pobrinuti da prenese podatke u jednom i drugom smjeru

Uppercase service

```
public class UppercaseService extends UppercaseGrpc.UppercaseImplBase {  
    private static final Logger logger =  
        Logger.getLogger(UppercaseService.class.getName()); // Generira plug-in  
    @Override  
    public void requestUppercase( // Prima obavijesti iz stremna poruka (observable pattern), koristi se za slanje i primanje poruka  
        Message request, StreamObserver<Message> responseObserver ) {  
        logger.info("Got a new message: " + request.getPayload());  
        // Create response  
        Message response =  
            Message.newBuilder().setPayload(request.getPayload().toUpperCase()).build();  
        // Send response  
        responseObserver.onNext( response ); // Šalje odgovor klijentu  
        logger.info("Responding with: " + response.getPayload());  
        // Send a notification of successful stream completion.  
        responseObserver.onCompleted(); // Zatvori stream  
    } }
```

gRPC poslužitelj (1/2)

```
public class SimpleUnaryRPCServer {  
    private static final Logger logger = Logger.getLogger(SimpleUnaryRPCServer.class.getName());  
    private Server server;  
    private final UppercaseService service;  
    private final int port;  
  
    public SimpleUnaryRPCServer(UppercaseService service, int port) {  
        this.service = service;  
        this.port = port;  
    }  
    public void start() throws IOException {  
        // Register the service  
        server = ServerBuilder.forPort(port).addService(service).build().start();  
        logger.info("Server started on " + port);  
        // Clean shutdown of server in case of JVM shutdown  
        Runtime.getRuntime().addShutdownHook(new Thread(() -> { System.err.println("Shutting down gRPC server since JVM is  
shutting down");  
            try {  
                SimpleUnaryRPCServer.this.stop();  
            } catch (InterruptedException e) {  
                e.printStackTrace(System.err);  
            }  
            System.err.println("Server shut down");  
        }));  
    }...  
}
```

Kreira instancu poslužitelja koji osluškuje na definiranom portu

gRPC poslužitelj (2/2)

```
public void stop() throws InterruptedException {
    if (server != null) {
        server.shutdown().awaitTermination(30, TimeUnit.SECONDS);
    }
}

public void blockUntilShutdown() throws InterruptedException {
    if (server != null) {
        server.awaitTermination();
    }
}

public static void main(String[] args) throws IOException, InterruptedException {
    final SimpleUnaryRPCServer server = new SimpleUnaryRPCServer(new UppercaseService(),
3000);
    server.start();
    server.blockUntilShutdown();
}
```

Radna dretva poslužitelja se zadržava sve dok ne stigne zahtjev za gašenjem poslužitelja.

gRPC klijent (1/2)

```
public class SimpleUnaryRPCCClient {  
    private static final Logger logger =  
        Logger.getLogger(SimpleUnaryRPCCClient.class.getName());  
    private final ManagedChannel channel;  
    private final UppercaseGrpc.UppercaseBlockingStub uppercaseBlockingStub;  
  
    public SimpleUnaryRPCCClient(String host, int port) {  
        this.channel = ManagedChannelBuilder.forAddress(host, port).usePlaintext().build();  
        uppercaseBlockingStub = UppercaseGrpc.newBlockingStub(channel);  
    }  
    public void stop() throws InterruptedException {  
        // Initiates an orderly shutdown in which preexisting calls continue but new calls are  
        // immediately cancelled. Waits for the channel to become terminated, giving up if the  
        // timeout is reached.  
        channel.shutdown().awaitTermination(5, TimeUnit.SECONDS);  
    } ...  
}
```

Kreira gRPC kanal prema transportnoj adresi poslužitelja.

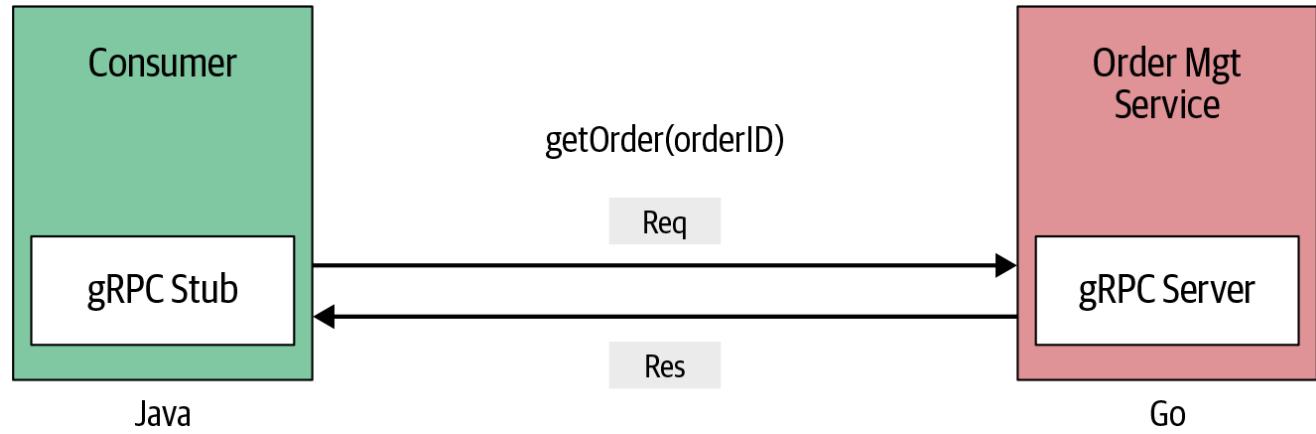
Kreira klijentski stub (blokirajući) pomoću kanala.

gRPC klijent (2/2)

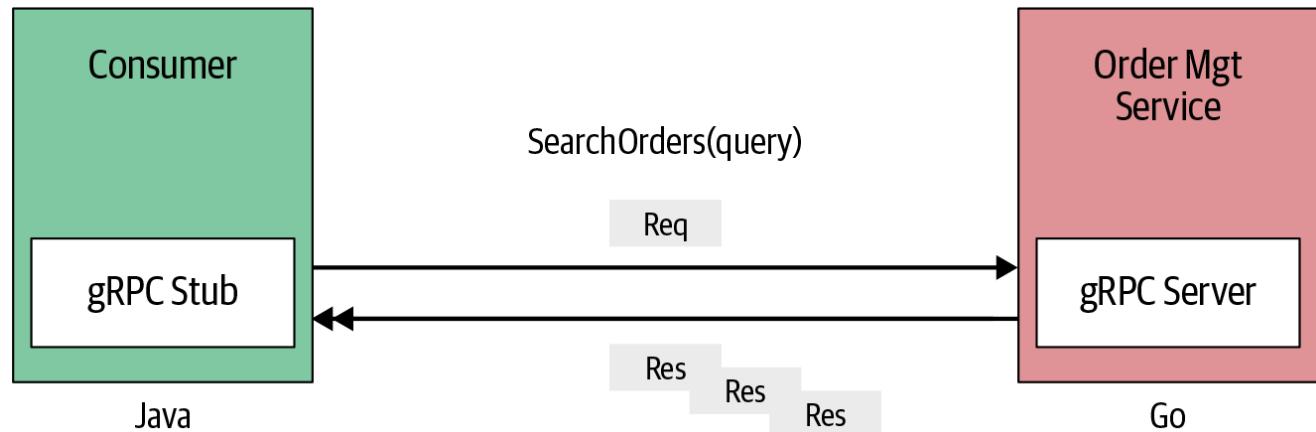
```
public void requestUppercase() {  
    final String payload = "message";  
    Message request = Message.newBuilder().setPayload(payload).build();  
    logger.info("Sending: " + request.getPayload());  
    try {  
        Message response = uppercaseBlockingStub.requestUppercase(request);  
        logger.info("Received: " + response.getPayload());  
    } catch (StatusRuntimeException e) {  
        logger.info("RPC failed: " + e.getMessage());  
    }  
}  
  
public static void main(String[] args) throws InterruptedException {  
    SimpleUnaryRPCClient client = new SimpleUnaryRPCClient("127.0.0.1", 3000);  
    client.requestUppercase();  
    client.stop();  
}
```

Mogući oblici komunikacije (1/2)

- Simple RPC (Unary RPC)

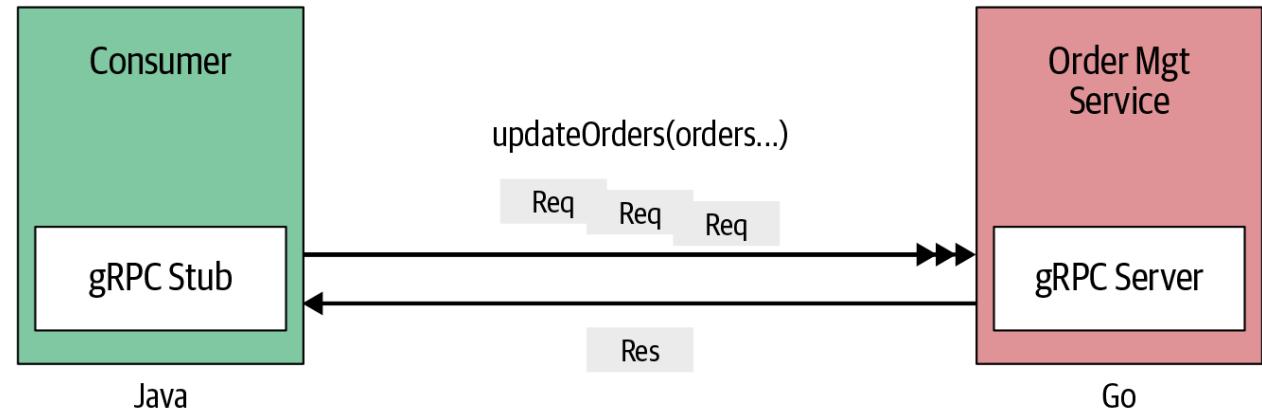


- Server-Streaming RPC

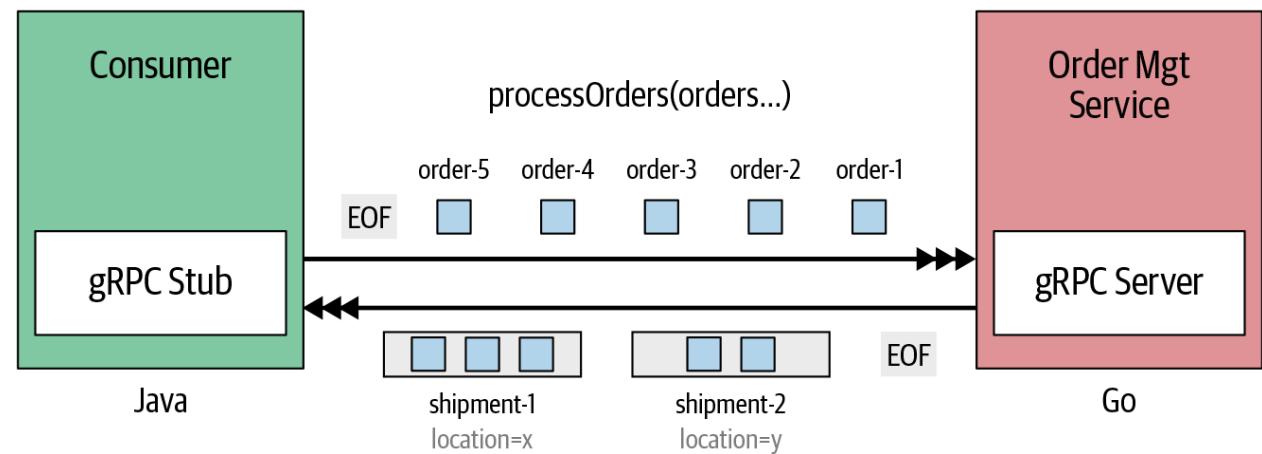


Mogući oblici komunikacije (2/2)

- **Client-Streaming RPC**



- **Bidirectional-Streaming RPC**



Pitanja za učenje i ponavljanje

- Objasnite zašto tranzientna sinkrona komunikacija potencijalno pati od problema vezanih uz skalabilnost.
- Može li se pomoću UDP-a implementirati protokol za pouzdanu komunikaciju između klijenta i poslužitelja? Ako može, na koji način?
- Poslužitelj je implementiran pomoću socketa TCP na portu 10000 s ograničenjem NUMBER_OF_THREADS=2. Objasnite detaljno operacije prilikom dolaska prvog klijentskog zahtjeva na poslužitelj. Što se događa kada stigne drugi, pa treći klijentski zahtjev, a prve dvije konekcije su još uvijek aktivne? Koliko socketa je vezano uz port 10000?
- Koliko byte-a se može maksimalno zapisati u UDP datagram?
- Usporedite gRPC i RESTful servise u pogledu performansi.

Literatura

- Maarten van Steen, Andrew S. Tanenbaum (2017.), *Distributed Systems 3rd edition*, Createspace Independent Publishing Platform poglavlja 4.2 (RPC) i 4.3 (samo dio o Socketima)
- G. Coulouris, J. Dollimore, T. Kindberg: *Distributed Systems: Concepts and Design*, 5th edition, Addison-Wesley, 2012 poglavlja 4.1, 4.2 i 4.3. (bez 4.3.1 CORBA i 4.3.3 XML) i poglavlje 5



SVEUČILIŠTE U ZAGREBU



Diplomski studij

Računarstvo

Raspodijeljeni sustavi

3. Arhitekture web-aplikacija,
tehnologije weba

Ak. god. 2022./2023.

Creative Commons



- slobodno smijete:

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
 - **prerađivati** djelo



- pod sljedećim uvjetima:

- **imenovanje**: morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
 - **nekomercijalno**: ovo djelo ne smijete koristiti u komercijalne svrhe.
 - **dijeli pod istim uvjetima**: ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.



U slučaju daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.

Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.

Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.

Tekst licence preuzet je s <http://creativecommons.org/>

Sadržaj predavanja

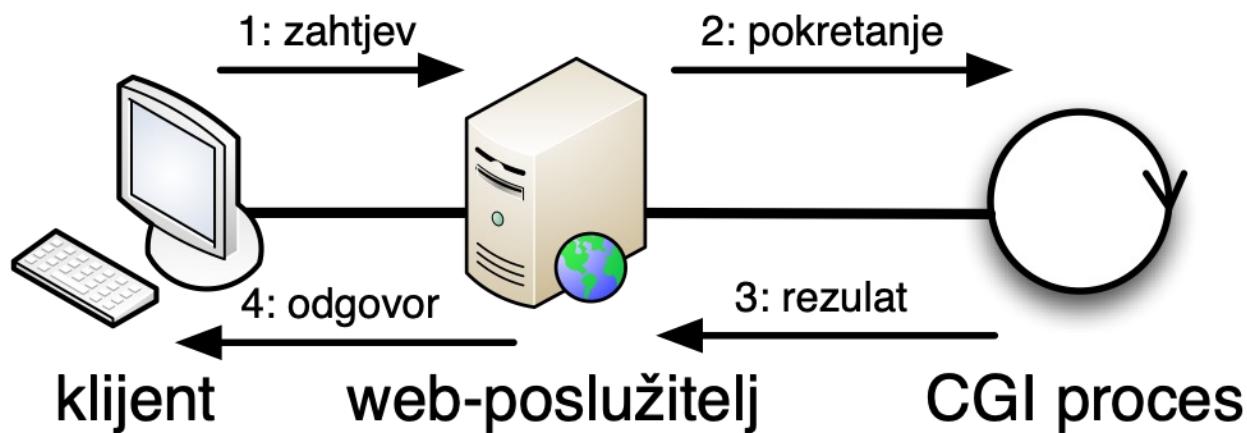
- arhitekture web-aplikacija
- AJAX
- uvod u Web 2.0
- Web-usluge
- jezici i protokoli: SOAP, WSDL, UDDI
- web-usluge temeljene na RPC-u, temeljene na dokumentima
- Web-usluge temeljene prijenosu prikaza stanja resursa (REST)
- Svojstva metoda protokola HTTP
- Model zrelosti web-usluga i relevantni standardi
- Implementacija REST-a u Springu

Web-aplikacije

- Definicija:
 - "*Web applications are stored on web servers, and use tools like databases, JavaScript (or Ajax or Silverlight), and PHP (or ASP.Net) to deliver experiences beyond the standard web page or web form.*"
- dvije vrste:
 - koje izgledaju kao normalne web-stranice (npr. portali)
 - koje izgledaju kako normalne aplikacije - bogato korisničko sučelje (npr. Google Mail)
- koriste tehnologije weba:
 - HTML, CSS, JavaScript, PHP, ASP, JSP, Ruby on Rails, Java Servlets, Cold Fusion, ...

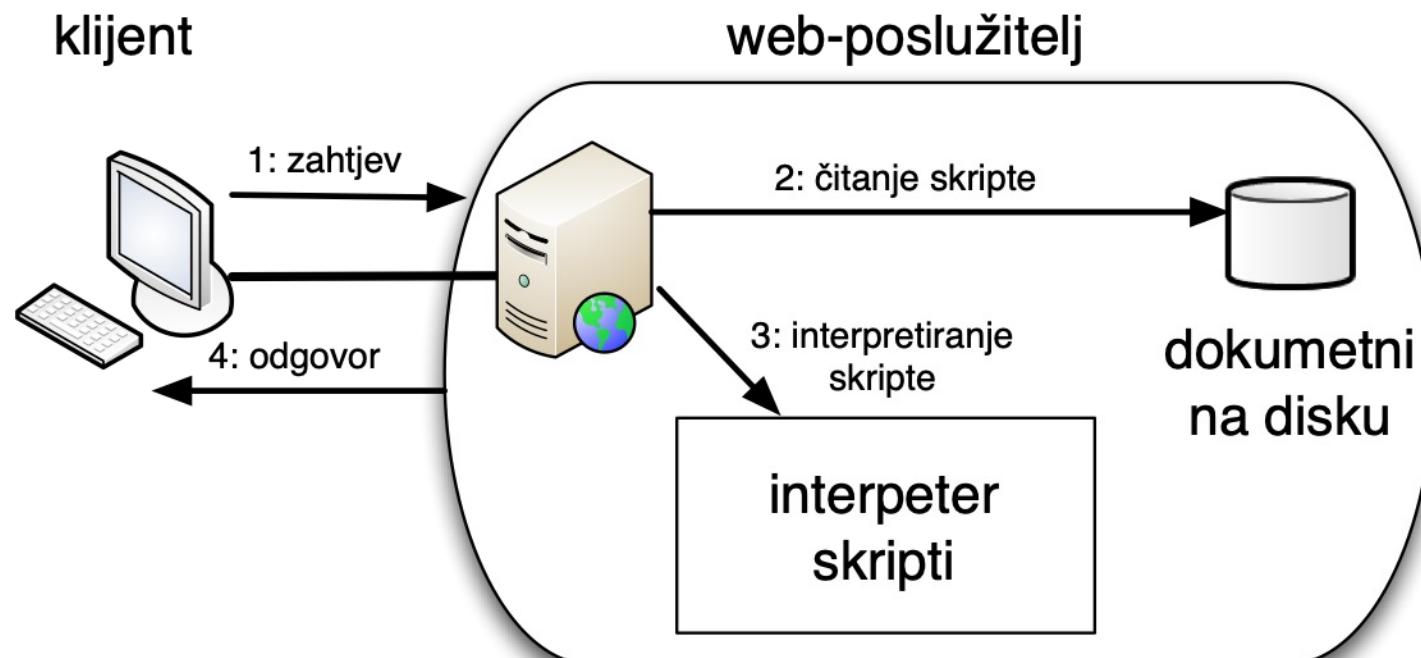
Web-aplikacije - CGI

- CGI - *Common Gateway Interface* - RFC 3875
- kod svakog zahtjeva se pokreće proces
- podaci između poslužitelja i procesa se šalju preko varijabli okoline i tokova podataka
- nakon svake obrade proces se gasi
- Bash, Perl i ostali skriptni jezici



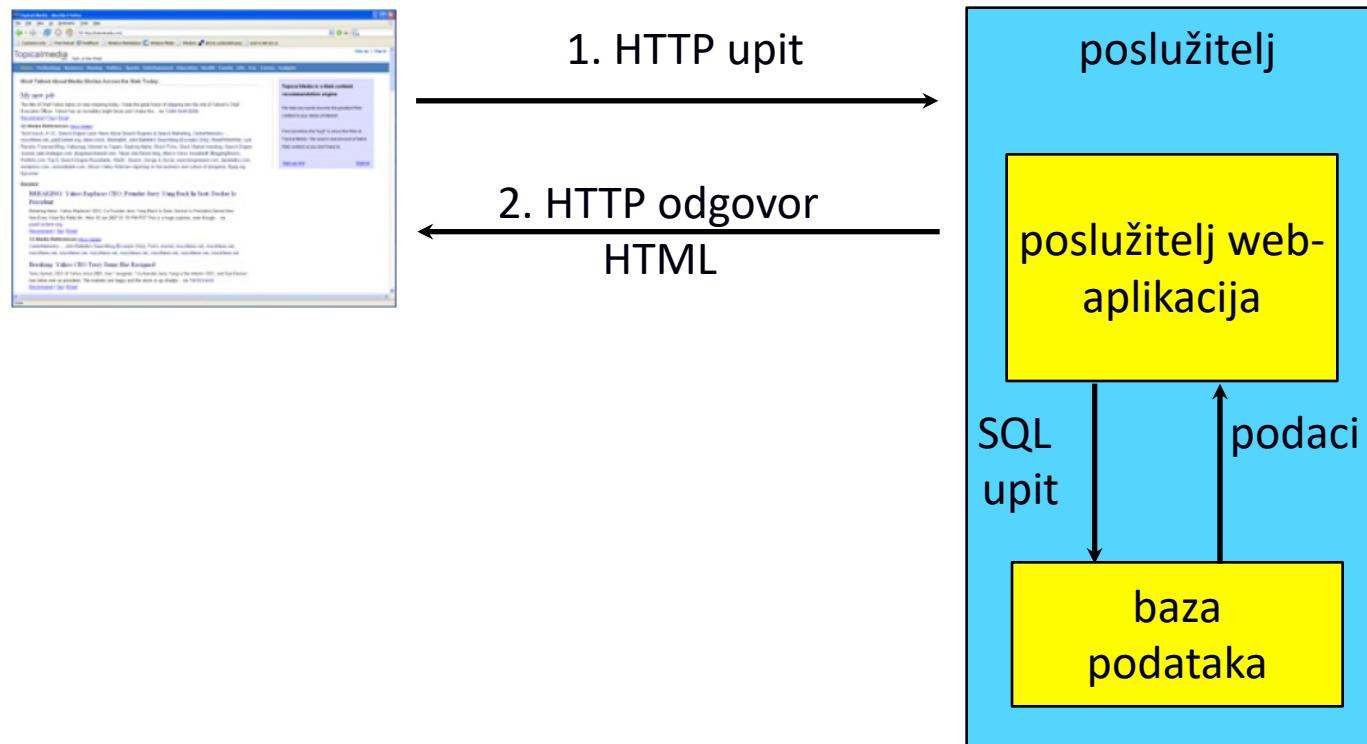
Web-aplikacije - poslužiteljske skripte

- poslužiteljske skripte - *server side scripts*
- dinamički se generira HTML-dokument (iz skripte)
- primjeri: PHP, ASP, JSP, Django, Ruby on Rails, ...



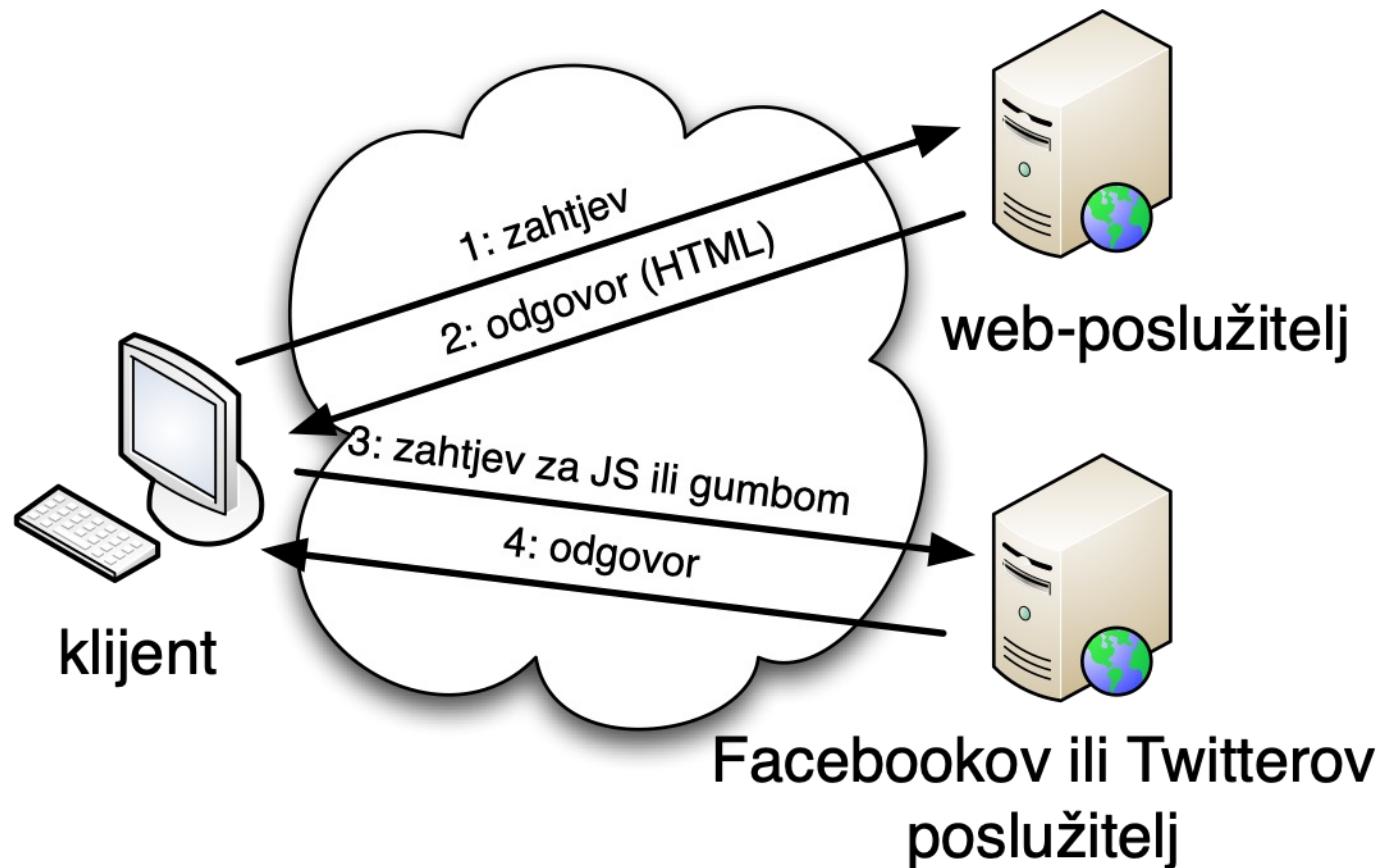
Tipična web-aplikacija na jednom računalu

- razvijateljska konfiguracija
- dobro za mali broj zahtjeva u produkciji



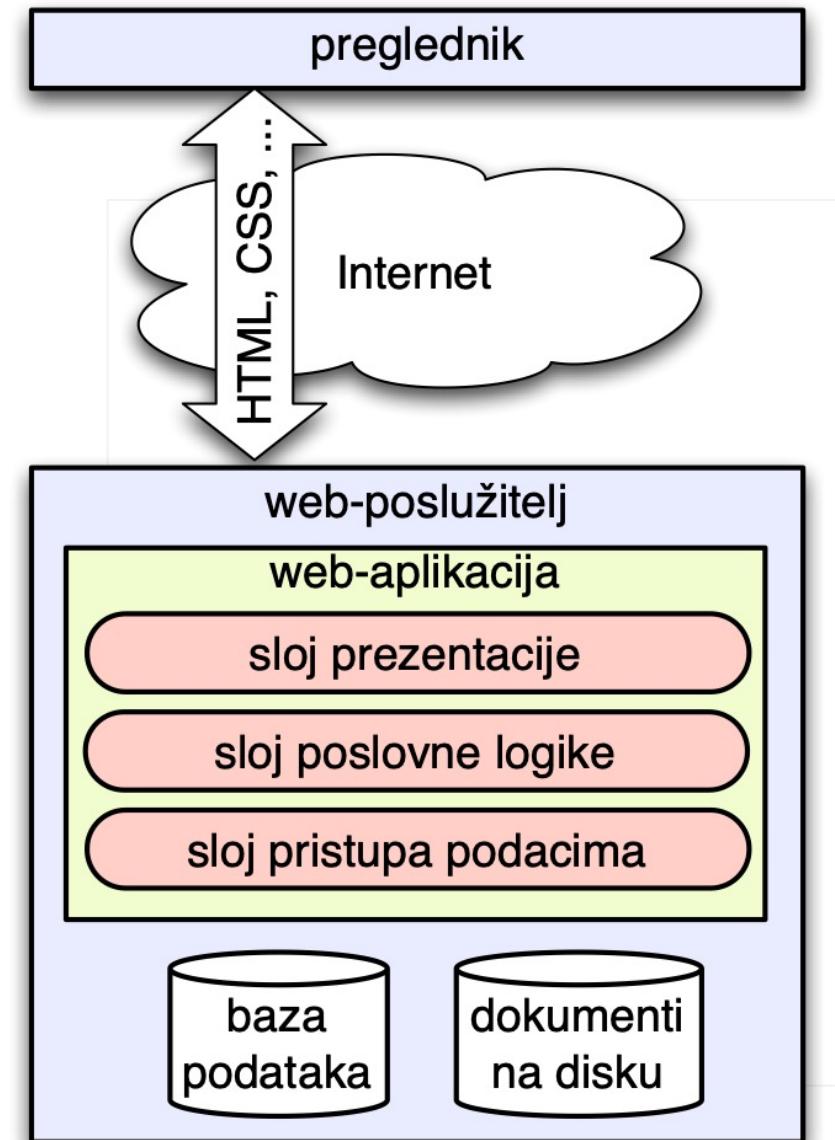
Web-aplikacija integrirana s drugim sjedištima

- gumbi za objavljivanje na drugim stranicama (npr. Facebook ili Twitter)



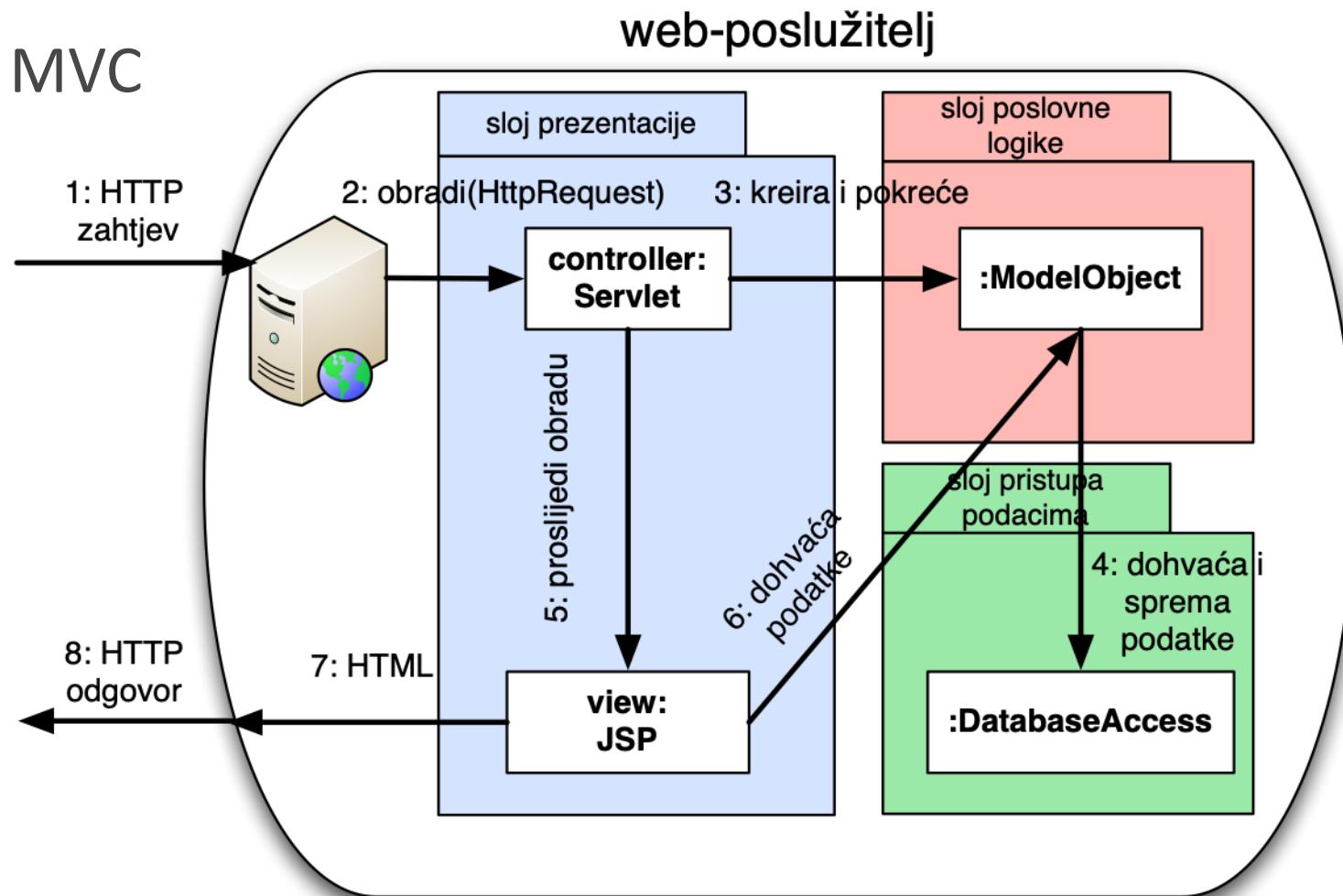
Arhitektura web-aplikacija

- višeslojna arhitektura
- sloj prezentacije
 - prikaz informacija: GUI, HTML, klikovi mišem, ...
 - obrada HTTP zahtjeva
- sloj poslovne (domenske) logike
 - obrada podataka
 - glavni dio sustava koji radi ono za što je sustav namijenjen
- sloj pristupa podacima
 - komunicira s bazom podataka i drugim komunikacijskim sustavima
 - brine se o transakcijama
 - brine se za pohranu podataka



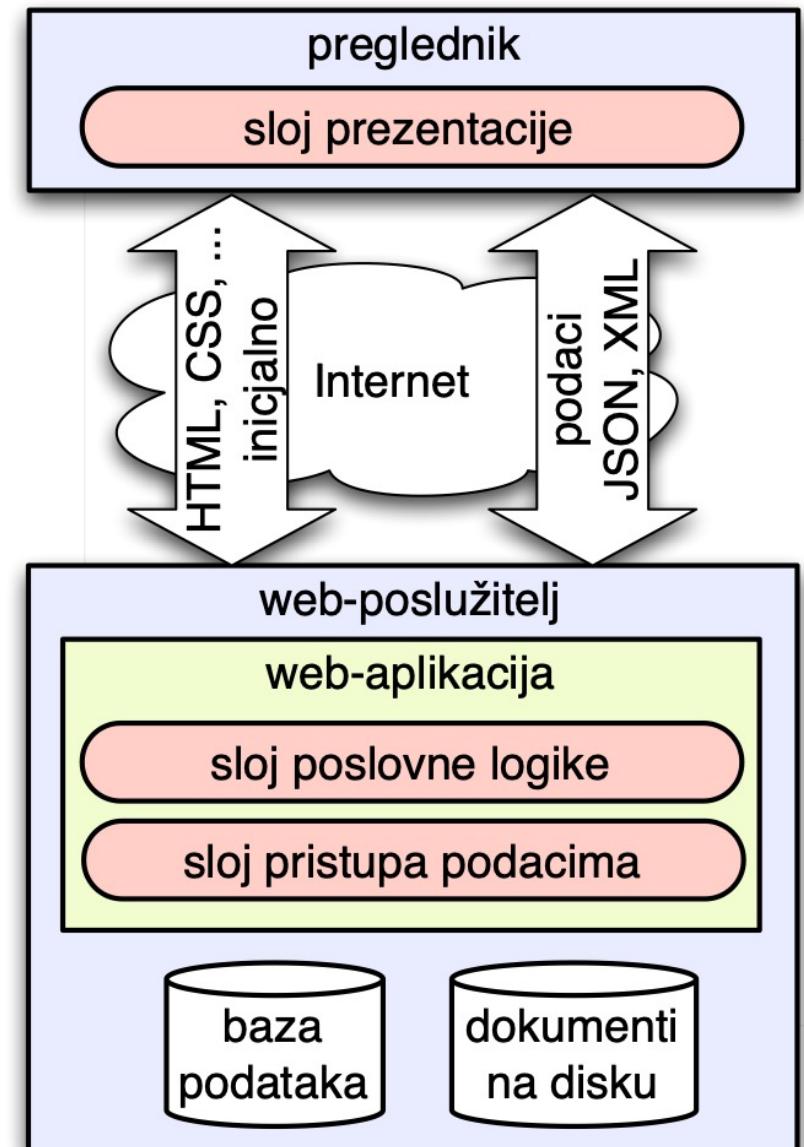
Arhitektura web-aplikacija - MVC u Javi

- *Model View Controller - MVC*



Nova arhitektura web-aplikacija

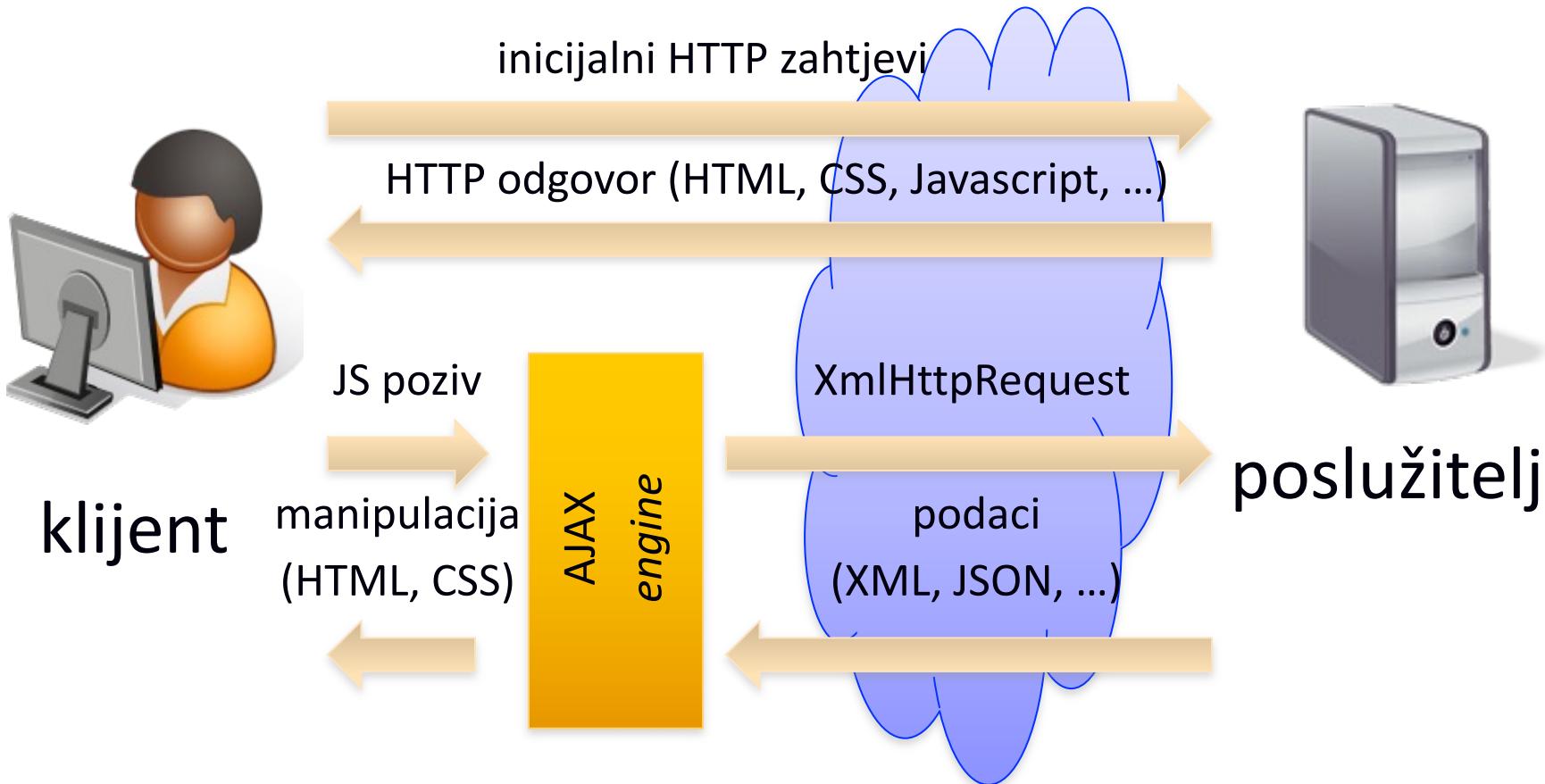
- inicijalno pokretanje
 - HTML, CSS, JavaScript, ...
- za vrijeme korištenja aplikacije
 - podaci putuju u obliku:
 - JSON - JavaScript Object Notation ili
 - XML
 - koristi se AJAX - [Asynchronous JavaScript](#) and XML



Skripte na klijentu - dio sloja prezentacije

- uključene u HTML ili u posebnoj datoteci koja je povezana
- obično se koristi:
 - **JavaScript** (Netscape), JScript (Microsoft), ECMAScript
- ECMAScript – standardiziran
 - specifikacije ECMA-262 i ISO/IEC 16262
 - svojstva: dinamički, slabo povezan, objektni, funkcijski
 - nema veze s jezikom Java osim imena JavaScript
- svrha:
 - dinamički elementi
 - interakcija s korisnikom
 - provjera obrazaca
 - komunikacija s poslužiteljem (AJAX)
 - ...

AJAX (Asynchronous JavaScript and XML)



- iz JavaScripta poslati upit na poslužitelj
- odgovor nije nova stranica već podatak (u formatu **JSON**, XML ili ...)
- dobije se na dinamičnosti web-stranice

Poznate knjižnice u JavaScriptu

- popis se stalno mijenja i stalno raste
 - <https://www.javascripting.com>
- DOM manipulation:
 - React (<https://facebook.github.io/react/>)
 - Gatsby (<https://www.gatsbyjs.com>)
 - Chakra Ui (<https://chakra-ui.com>)
- Korisničko sučelje (*User Interface*):
 - Ant Designl (<http://ant.design/>)
 - Material Ui (<https://material-ui.com/>)
 - Github Readme Stats (<http://github.com/anuraghazra/github-readme-stats>)
- Općenito:
 - React Native (<http://facebook.github.io/react-native/>)
 - Next.js (<https://zeit.co/blog/next>)
 - Electron (<https://electron.atom.io/>)
 - Ant Design (<https://ant.design>)

Web 2.0

- pojam nastao u O'Reilly Media (2003.)
- ideja: Internet kao platforma poput operacijskog sustava
- potiče inovacije i sastavljanje funkcionalnih dijelova iz neovisnih izvora
- korisnici postaju središte
 - interaktivnost, sudjelovanje, objavljivanje
- višemedijski sadržaji (glazba, video, lokacije, slike, karte, ...)

Usporedba weba 1.0 i 2.0

Web 1.0

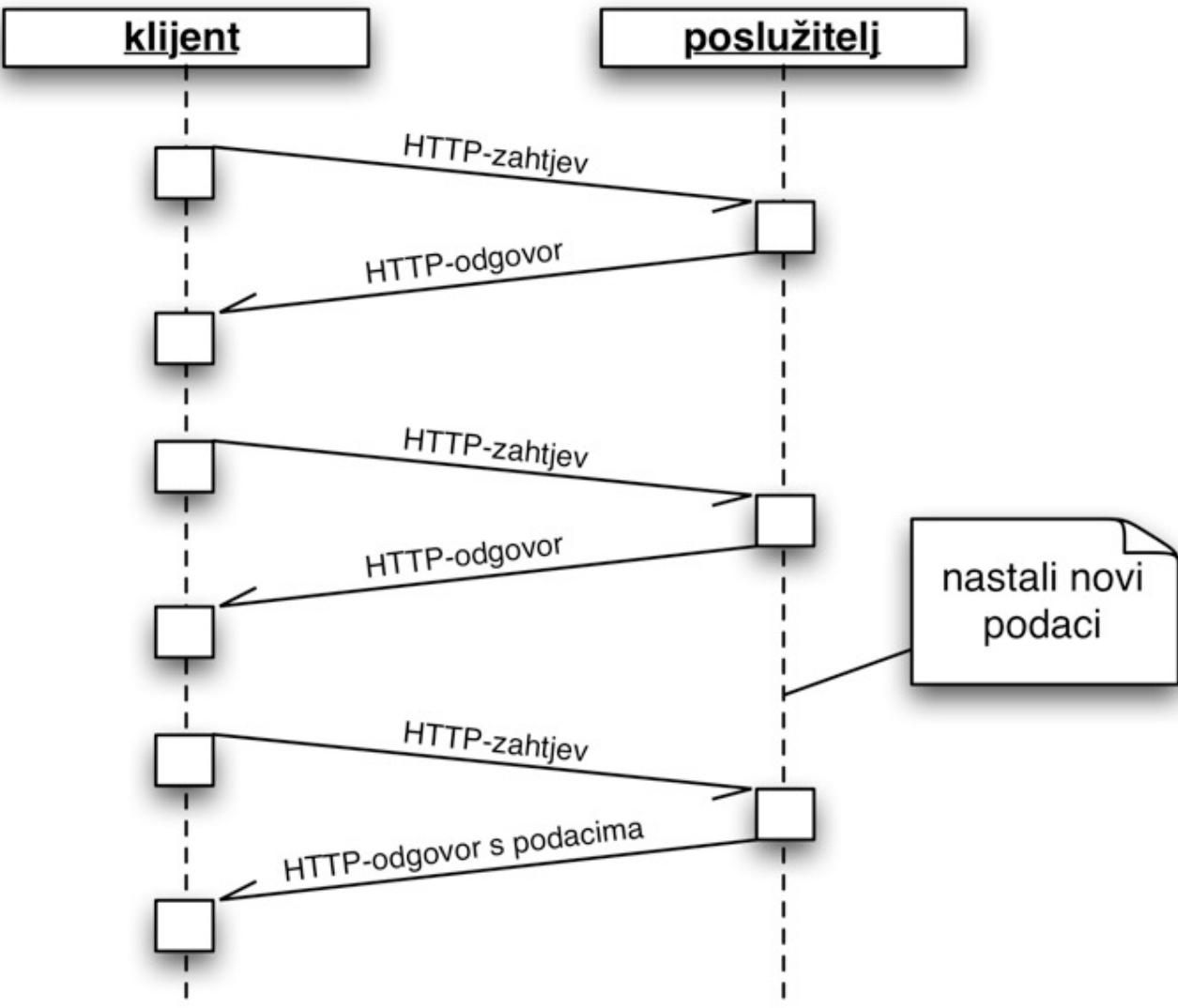
- čitanje informacija
- klijent-poslužitelj
- HTML
- portali
- parsiranje informacija
- posjedovanje
- modemska veza i HW
- direktoriji
- tipična tvrtka: Netscape

Web 2.0

- pisanje informacija
- P2P (osobe i aplikacije)
- XML, JSON, HTML5
- usluge (*services*)
- REST API sučelja, RSS
- dijeljenje
- širokopojasna veza i SW
- oznake (*tag*)
- tipične tvrtke: Google, Facebook, Twitter, ...

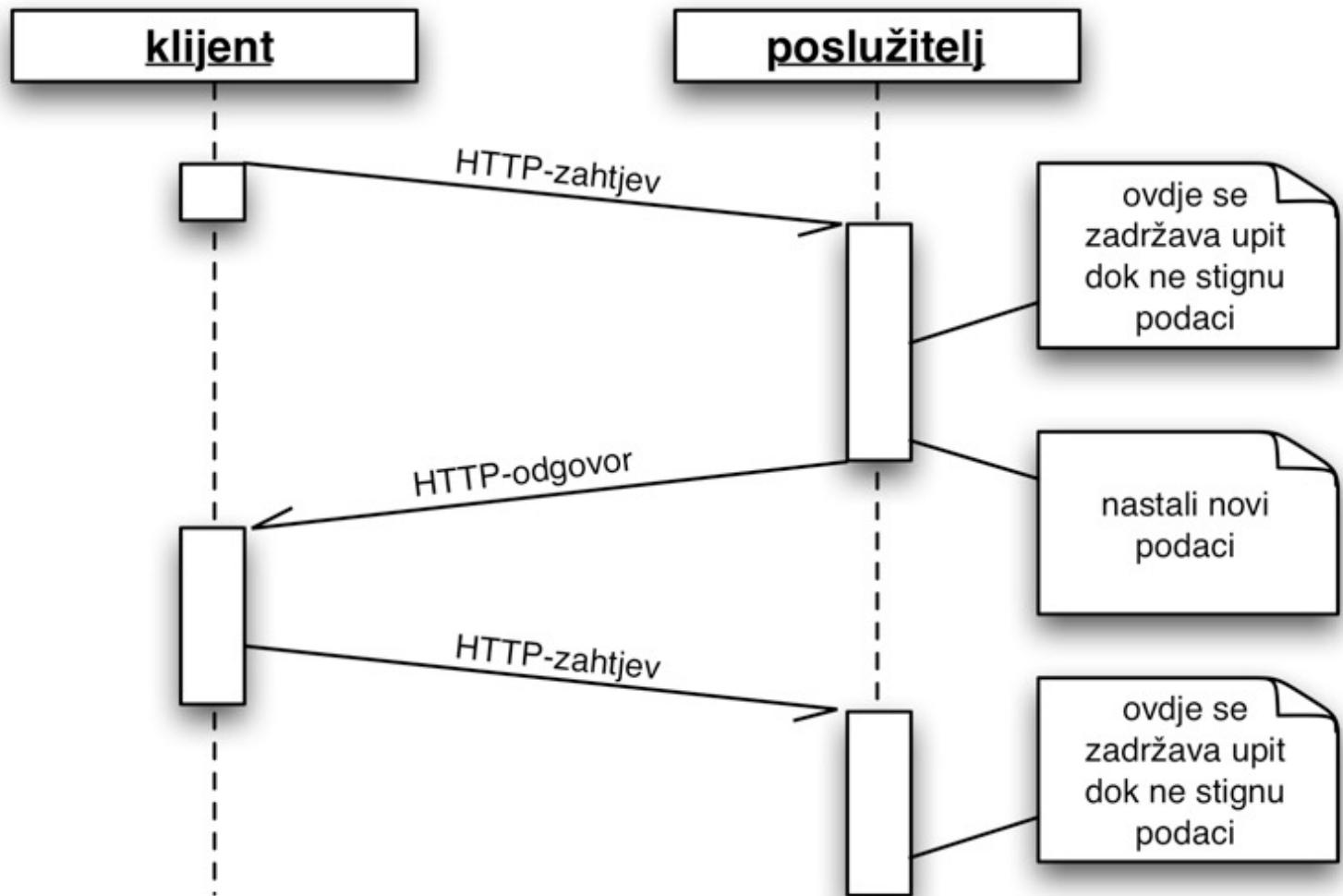
Slanje događaja klijentu iz preglednika (1) - prozivanje

- prozivanje poslužitelja (*poll*)



Slanje događaja klijentu iz preglednika (2) – dugo prozivanje

- dugo prozivanje poslužitelja
(long poll)



Slanje događaja klijentu iz preglednika (3) - SSE

- **Server-Sent Events** - SSE
- definirano standardom [Server-Sent Events](#) iz 2015.
- omogućuje slanje događaja klijentu kroz otvorenu konekciju
- klijent šalje zaglavljje Accept: text/event-stream
- poslužitelj kroz istu konekciju šalje tekst
 - svaka poruka je odijeljena praznim retkom
 - poruka ima polja koja su slična zaglavljima HTTP-a:
 - *event* - vrsta događaja (opcionalno)
 - *data* - podaci (obavezno)
 - *id* - identifikator paketa (opcionalno)
 - *retry* - vrijeme ponovnog spajanja u milisekundama (opcionalno)
- [članak - SSE, PHP primjer](#)

Primjer poslanih podataka:
data: Prva poruka

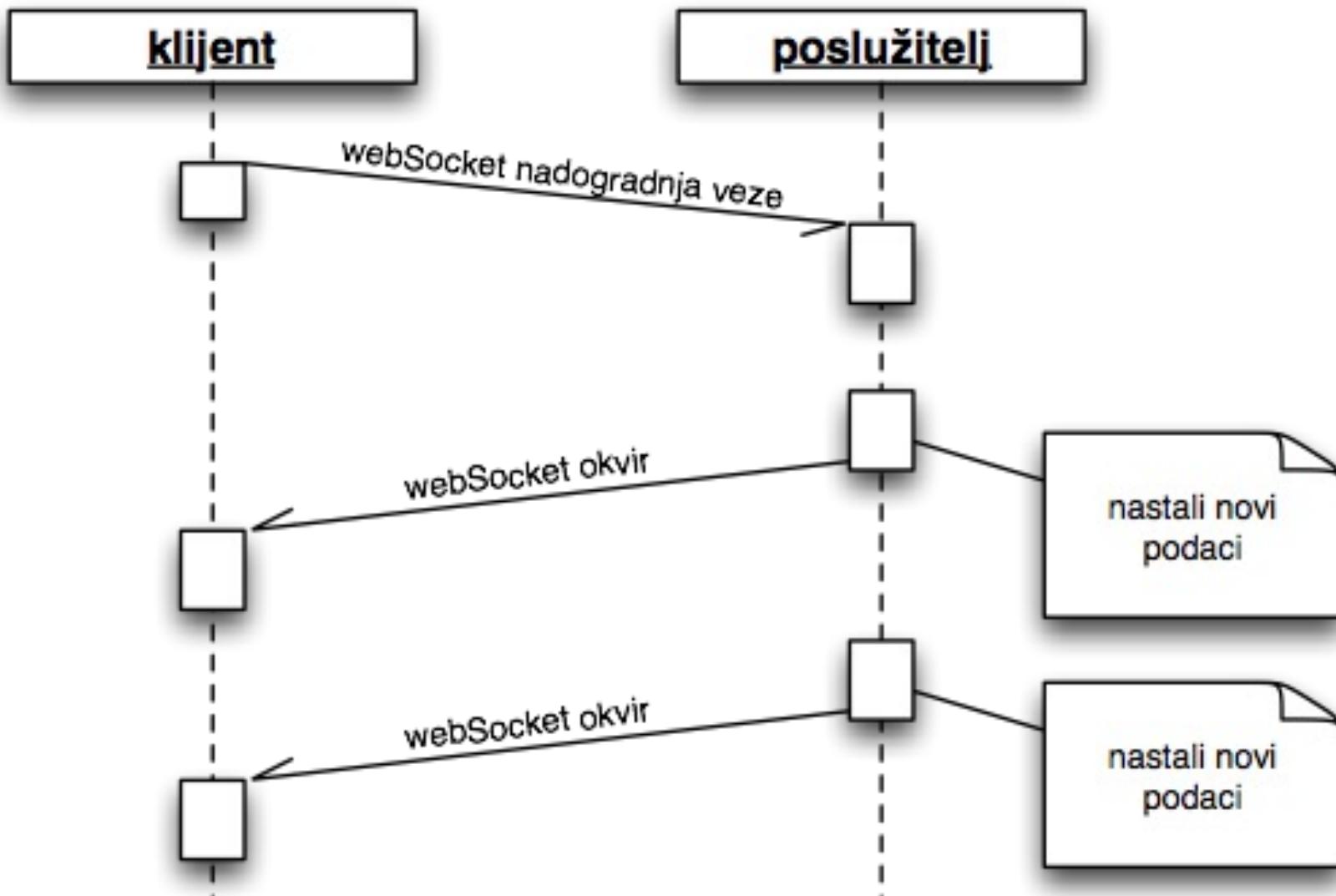
data: Druga poruka
data: 2. red 2. poruke

data: Treća poruka

Slanje događaja klijentu iz preglednika (4) - WS

- The **WebSocket (WS)** Protocol - RFC 6455 - prosinac 2011.
- omogućuje komunikaciju nalik TCP-ovskoj komunikaciji, ali iz internetskih preglednika
- podržavaju full-duplex, istovremeno je moguće primati i slati podatke
- inicijalna uspostava veze je kompatibilna s HTTP-om da bi isti poslužitelji mogli na istim vratima primati i HTTP-ovske zahtjeve i Web Socket zahtjeve
- nakon toga slijedi razmjena podataka u okrvirima
- URL shema: ws: (80) ili wss: (443) - ista vrata kao i HTTP
- koristi se: igre, kolaborativne aplikacije, financijske aplikacije, feed na društvenim mrežama, strujanje video sadržaja, ...

Web Socket - komunikacija

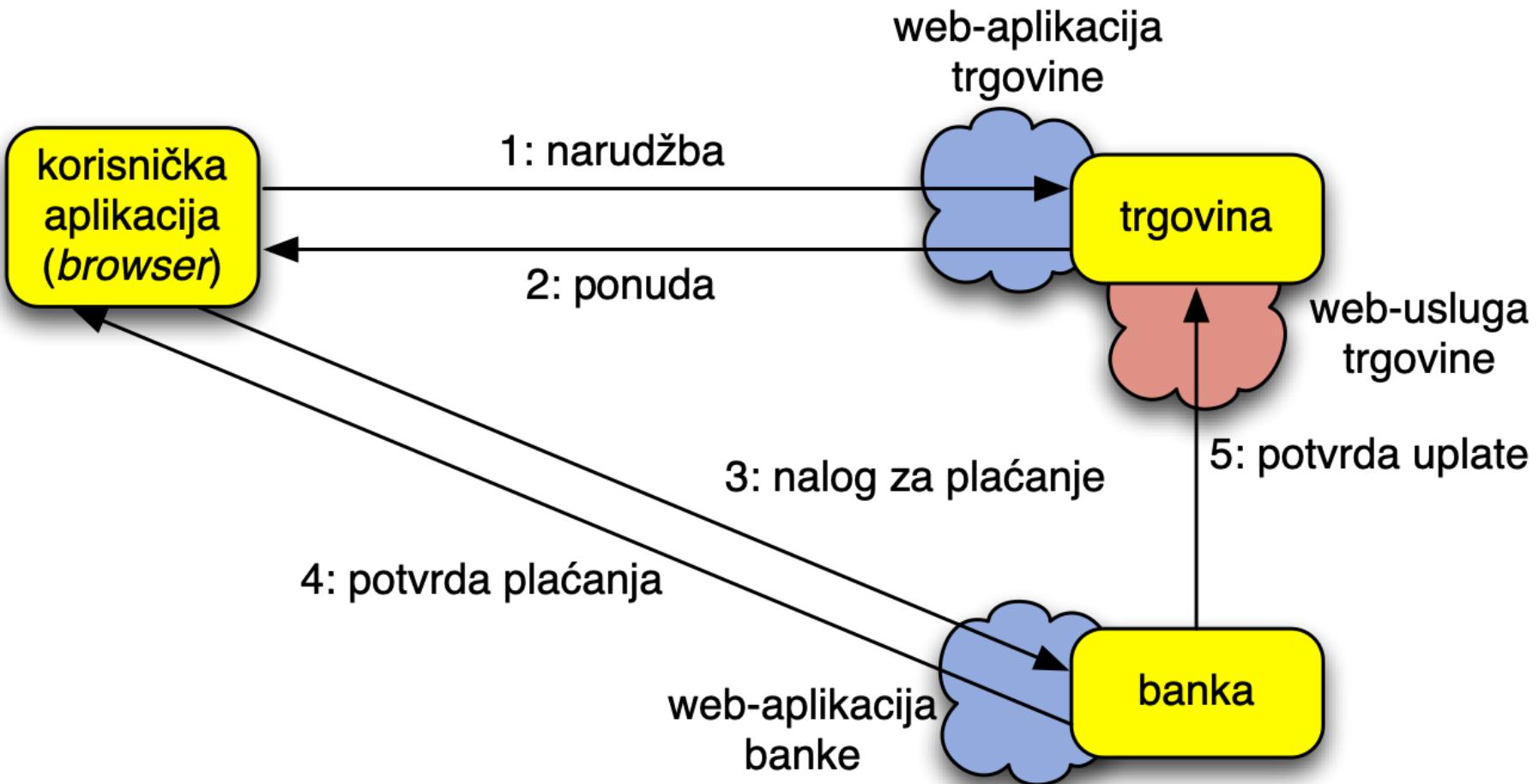


Web Socket - sadržaj

- 3 vrste sadržaja poruke:
 - tekstualni
 - binarni
 - ping-pong
 - ping-pong služi za provjeru dostupnosti druge strane
 - na poruku ping druga strana mora čim prije odgovoriti porukom pong jednakog sadržaja (maksimalna dužina sadržaja ping i pong poruka je 125 okteta)
 - na više primljenih ping poruka odgovara se samo jednom pong porukom, a samoinicijativnu pong poruku se ignorira
- podprotokoli
 - WebSocket ne definira aplikacijski protokol
 - Previše "posla" za programera kod parsiranja poruka
 - Možemo koristiti aplikacijske protokole preko WebSocketsa
 - dogovor prilikom pregovaranja
 - npr. STOMP, WAMP, MQTT, XMPP, ...

Web-usluge

Motivacija: scenarij kupovine



Uvod u web-usluge

- “obični” Web tipično koriste ljudi za pribavljanje informacija
- tehnologija Web Services (WS) omogućuje programima lakše korištenje Weba
 - umjesto RPC-a/Java RMI-a, CORBA-e, DCOM-a, itd.
- svojstva:
 - komunikacija se temelji na XML-u
 - koristi već postojeću internetsku infrastrukturu i protokole
 - usluge dostupne putem Interneta
 - omogućuje integraciju između različitih aplikacija
 - ne ovisi o programskom jeziku ili zatvorenoj tehnologiji jedne tvrtke
 - omogućuje otkrivanje usluga koje nude te aplikacije
 - usluge su slabo povezane
 - temelji se na industrijskim standardima

Što je web-usluga?

«Web-usluga je aplikacija identificirana URL-jem, čija se sučelja i veze mogu definirati, opisati i otkriti XML-om i koja podržava direktnu interakciju s drugim aplikacijama putem internetskih protokola koristeći poruke temeljene na XML-u»

(www.w3.org)

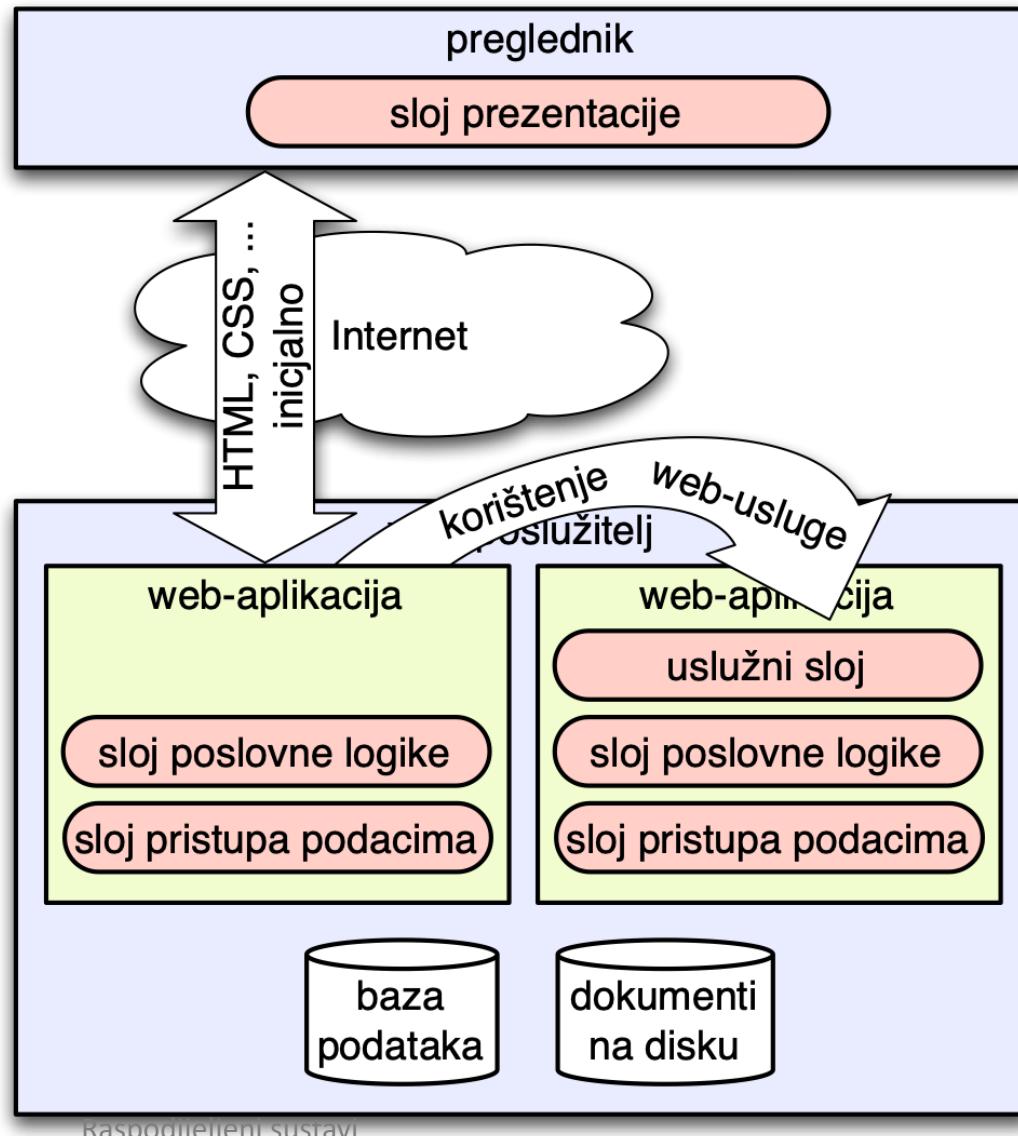
«Tehnologija web-usluga predstavlja novu vrstu web-aplikacija. To su samostalne, samoopisujuće aplikacije građene od modula, a koje se mogu objaviti, otkriti i pobuditi putem Weba. Web-usluge obavljaju funkcije koje mogu biti bilo što, od jednostavnih zahtjeva do komplikiranih poslovnih transakcija...»

(IBM's tutorial, www.xml.com)

- web-usluga je program koji:
 - je identificiran URI-jem
 - komunicira s klijentskim programima putem Weba
 - ima sučelje (API) opisano standardima web-usluga
 - omogućuje korištenje neovisno o platformama i programskim jezicima

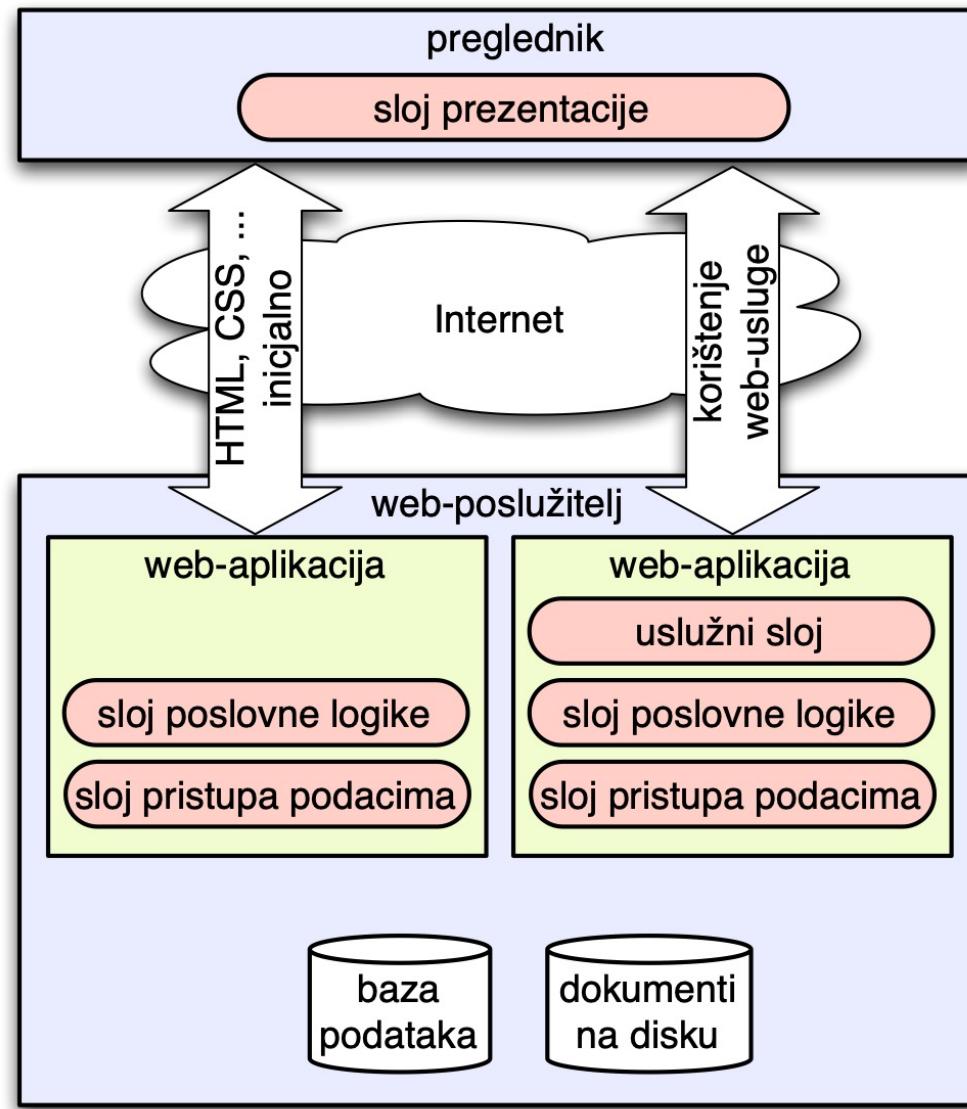
Web-aplikacija koja koristi web-usluge (1)

- web-aplikacija koristi web-uslugu
- web-usluga ne mora biti na istom računalu (u principu nije)



Web-aplikacija koja koristi web-usluge (2)

- klijent koristi web-uslugu (npr. klijent je aplikacija na pametnom telefonu ili preglednik koji koristi JavaScript za korištenje usluge)
- ovdje se obično radi o web-usluzi koja se temelji na REST-u



Vrste web-usluga

- **Usluge temeljene na udaljenim procedurama (RPC)**
 - rade kao poziv udaljenih procedura
 - koriste protokol SOAP i specifikaciju WSDL
 - na poslužitelju se poziva metoda u objektu
 - podaci su povezani s metodama koje se pozivaju
- **Usluge temeljene na dokumentima/porukama**
 - definiraju se poruke koje se razmjenjuju (XML Schema, WSDL)
 - koriste protokol SOAP i specifikaciju WSDL
 - nisu jako povezane
 - nije bitna implementacija, već samo podaci
- **Usluge temeljene na prijenosu prikaza stanja resursa (REST)**
 - RESTful (*Representational state transfer*) ili REST-usluge
 - temelje se na protokolu HTTP
 - koriste metode protokola: GET, PUT, DELETE, POST, PATCH

Tehnologije

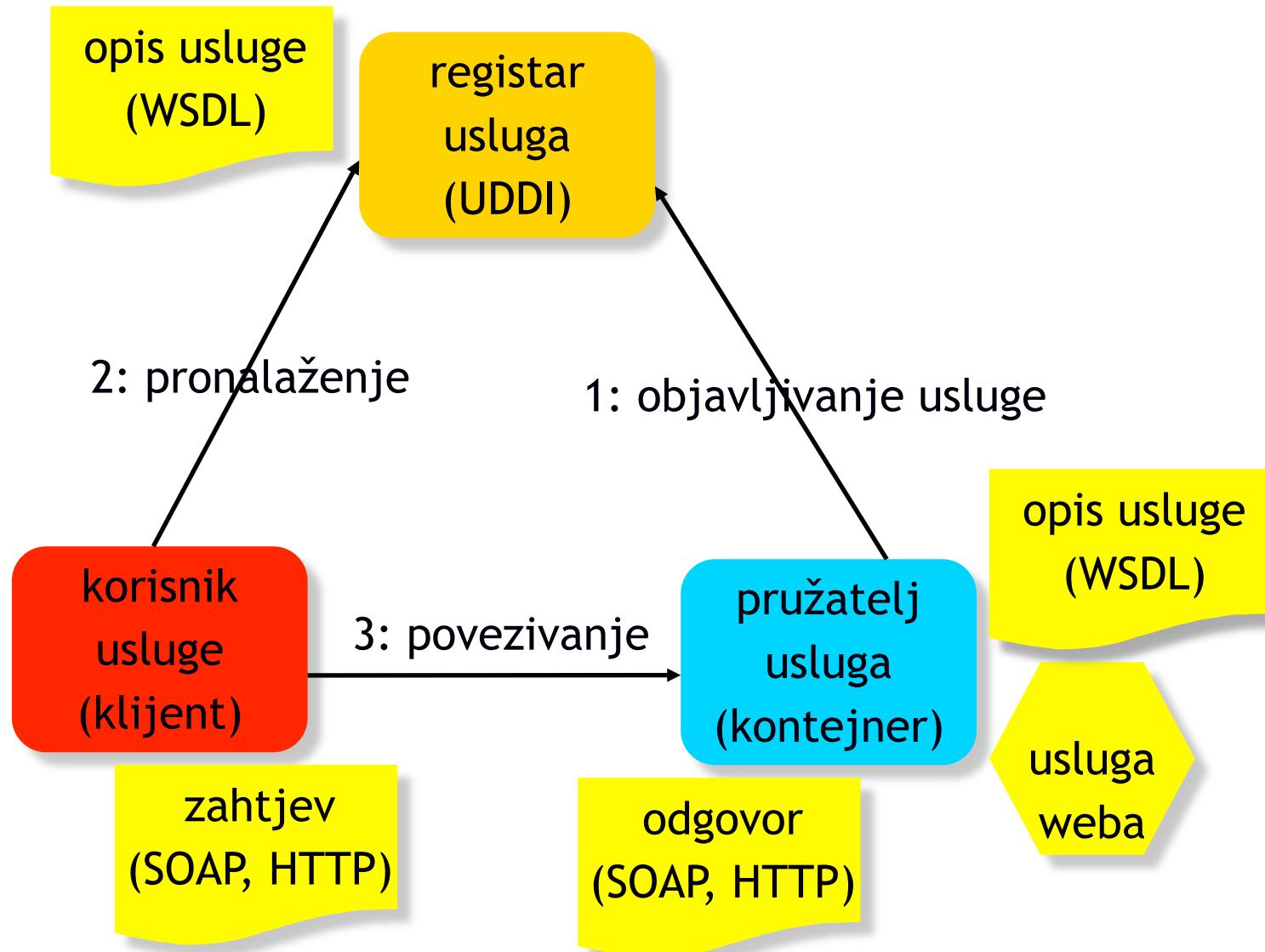
Tehnologije web-usluga

- WSDL (*Web Services Definition Language*)
 - opisuje uslugu
- SOAP (*Simple Object Access Protocol*)
 - format poruke
- UDDI (*Universal Description, Discovery and Integration*)
 - za otkrivanje usluga

Druga generacija web-usluga (WS-*)

- WS-Coordination
 - protokol za koordinaciju distribuiranih aplikacija
- WS-Transaction
- BPEL4WS (Business Process Execution Language)
 - jezik za formalnu specifikaciju poslovnih procesa i interakcijskih protokola
- WS-Security
 - sigurnosni protokol (TLS, integritet, privatnost, ...)
- WS-ReliableMessaging
- WS-Policy
- WS-Attachments
- WS-Addressing
 - adresiranje usluga i poruka

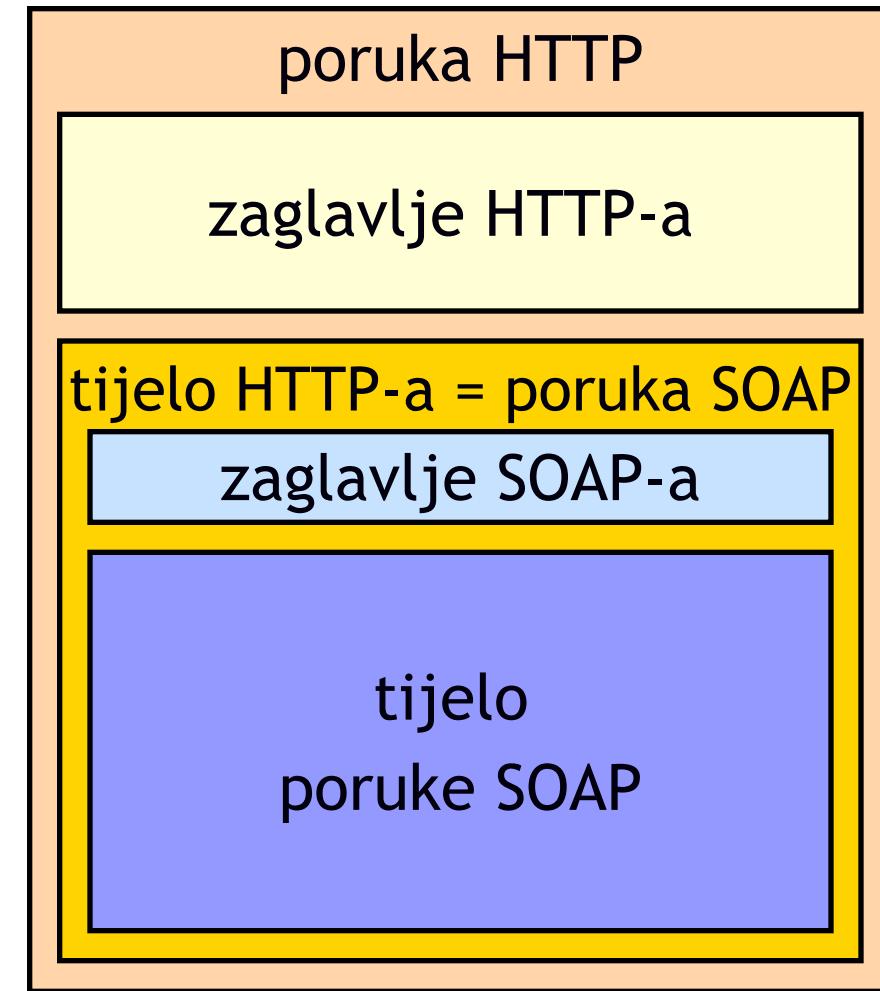
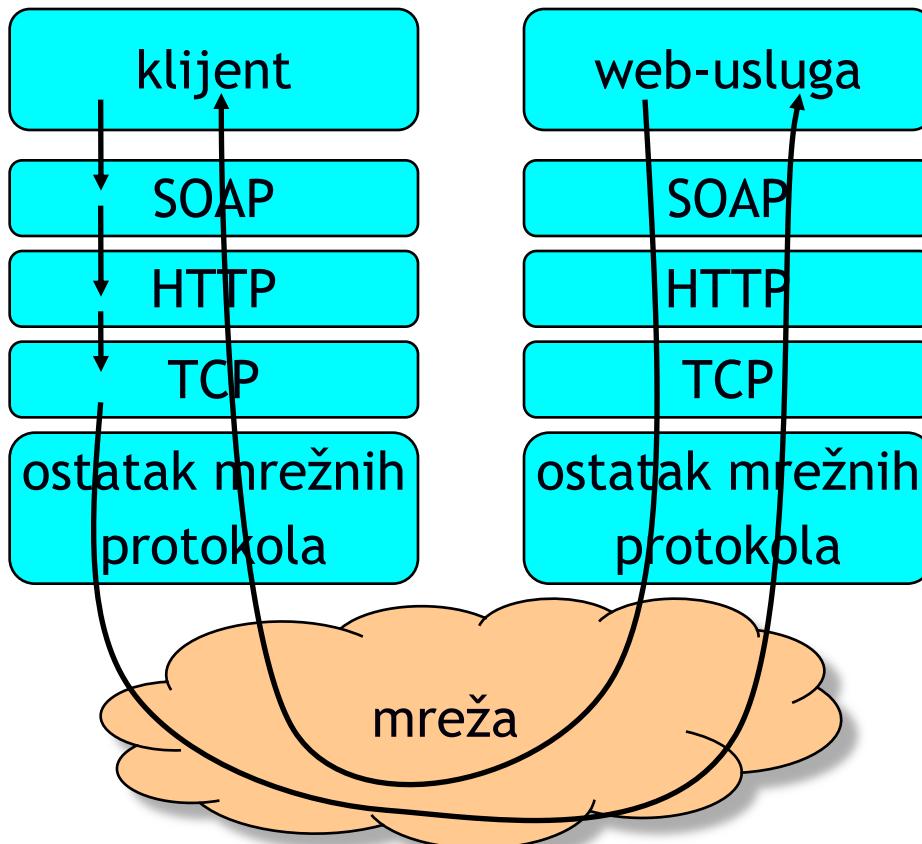
Arhitektura i korištenje



SOAP (*Simple Object Access Protocol*)

- omogućuje komunikaciju s web-uslugom
- dva osnovna načina rada:
 - poziv udaljenih procedura (RPC)
 - slično kao Corba, DCOM, Java RMI
 - služi za prijenos serijaliziranih parametara i rezultata
 - posljedica:
 - dobro definirana sučelja i tipovi podataka
 - prilagodni kod može biti generiran automatski
 - razmjena dokumenata/poruka
 - sadrži XML dokument
 - fleksibilnije u odnosu na RPC
 - XSLT i XQuery se koristi za prilagodbu dokumenata
 - lakše se koriste uzorci razmjene poruka
 - polako se uključuje semantički web (ontologije, procesiranje i sl.)
- specifikacija: <http://www.w3.org/TR/soap/>

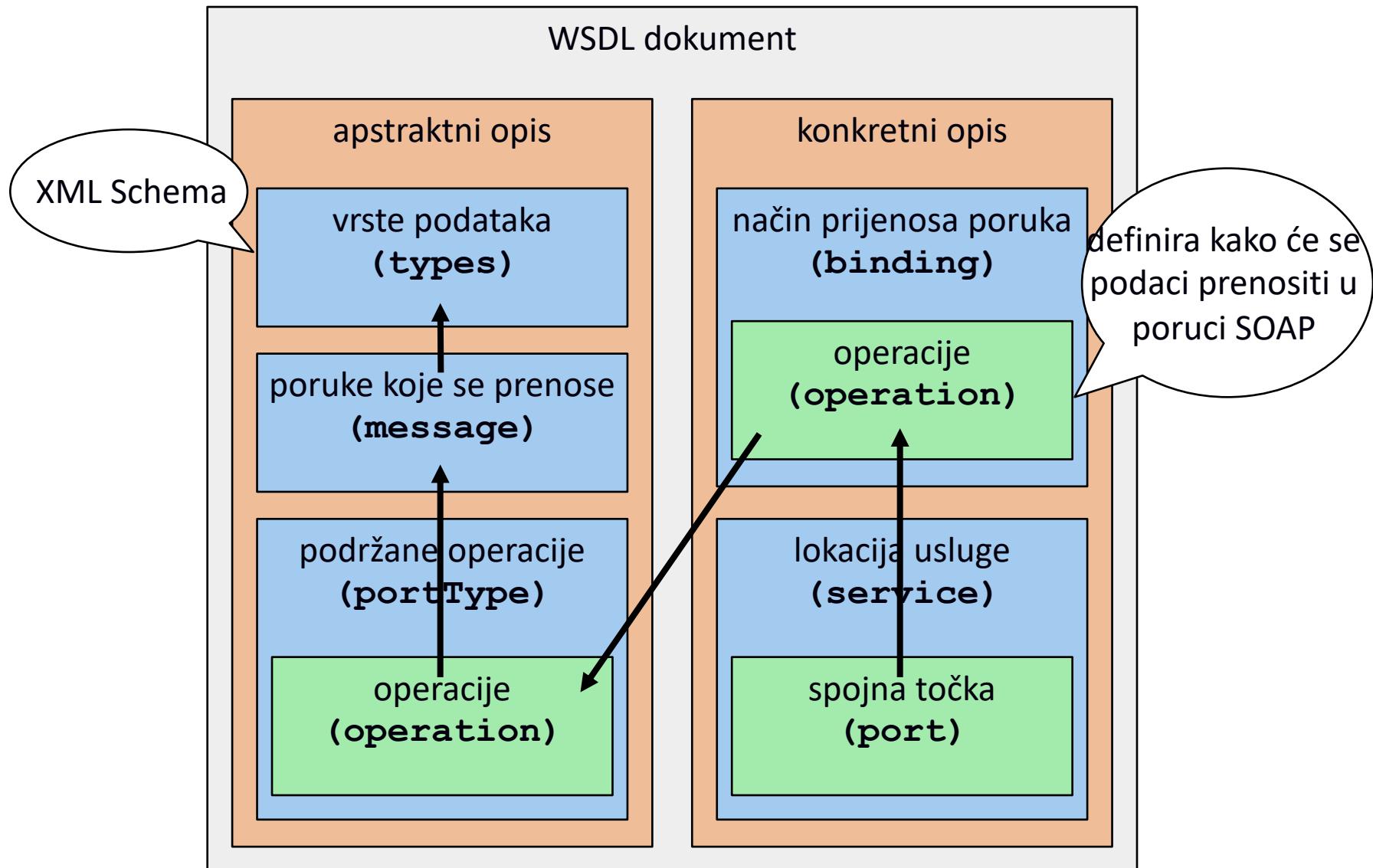
Prijenos poruke SOAP



WSDL (Web Services Description Language)

- jezik za opis web-usluga
- web-usluga je opisana skupom komunikacijskih krajnjih točaka (*ports*)
- krajnja točka se sastoji od dva dijela:
 - apstraktne definicije operacija i poruka
 - specifikacije mrežnog protokola i pojedine krajnje točke te formata poruke
- opisuje komunikacijske detalje između klijenta i usluge
 - strojevi (računala) mogu pročitati WSDL
 - mogu pozvati uslugu definiranu WSDL-om
- jedan je od mehanizama koji omogućuje da se usluga može otkriti pomoću registra
- specifikacija: <https://www.w3.org/TR/wsdl>

Struktura WSDL-a



Elementi WSDL-a (1)

- **definitions**
 - osnovni element WSDL dokumenta
- **types**
 - opis podataka pomoću XML Scheme
 - nepotreban ako se koriste osnovni podaci iz Scheme
- **message**
 - opis jednosmjerne poruke
 - definira ime poruke
 - poruka se sastoji od dijelova (**part**)
 - svaki dio se referencira na vrste podataka iz dijela **types**
- **portType**
 - definira operacije
 - svaka operacija se sastoji od poruka (reference na poruke)
 - poruke koji mogu biti:
 - parametri (ulazne poruke)
 - vrijednosti koje se vraćaju (rezultati)

Elementi WSDL-a (2)

- **binding**
 - definira kako će se poruke iz operacije prenositi
 - definira koje transportne protokole će koristiti (HTTP GET, HTTP POST, SOAP, SMTP)
 - stil definira vrstu čitave poruke:
 - **rpc** - zahtjev će imati omotač u kojem će pisati naziv funkcije koja se poziva
 - **document** - zahtjev i odgovori će imati “obične” XML dokumente
 - poruke mogu koristiti dvije vrste pakiranja poruka:
 - **literal** - onako kako definira Schema
 - **encoded** - kodirano pravilima u SOAP-u
- **services**
 - definira lokaciju usluge (URL)
- **import**
 - koristi se za uključivanje drugih WSDL ili XML Schema dokumenata

Pronalaženje usluga

- UDDI (*Universal Description, Discovery and Integration*)
- osigurava platformu za otkrivanje usluga na Internetu
- sastoji se od 3 dijela:
 - imenik (*White pages*)
 - adrese, kontakti i identifikatori
 - poslovni imenik (*Yellow pages*)
 - kategorizacija područja, usluga i proizvoda te lokacije
 - tehničke informacije (*Green pages*)
 - sadrži tehničke informacije o uslugama
- nema centralnog registra
- više informacija: <http://uddi.xml.org/uddi-org>
- standard: <https://www.oasis-open.org/standards#uddiv3.0.2>

Web-usluge temeljene na RPC-u

- napravimo novi projekt (npr. NetBeans)
- stvorimo novu web-uslugu i stvorimo operacije/metode kao u primjeru dolje
- alat sam generira WSDL i iz WDSL-a klijenta

```
@WebService()  
public class MyService {  
    @WebMethod(operationName = "add")  
    public int add(@WebParam(name = "x") int x,  
                  @WebParam(name = "y") int y)  
    {  
        return x+y;  
    }  
  
    @WebMethod(operationName = "toLowerCase")  
    public String toLowerCase(  
        @WebParam(name = "text")  
        String text)  
    {  
        return text.toLowerCase();  
    }  
}
```

Usluge temeljene na dokumentima

- Usluge temeljene na dokumentima
 - razmjenjuju se dokumenti
 - dogovor o dokumentima (XML Schema)
 - obično su asinkrone
 - parametri kod povezivanja u WSDL-u su: Document-literal
- Postupak izrade:
 1. definiranje XML Scheme dokumenata
 - obično u posebnoj datoteci
 2. definiranje WSDL-a
 - uključuje XML Scheme
 3. iz WSDL-a se mogu generirati:
 - kostur usluge
 - primer klijenta

Web-usluge temeljene prijenosu prikaza stanja resursa (REST)

- REST (*Representational State Transfer*)
- pojmovi: *The REST Way* ili *RESTful services*
- pojam je skovao Roy Fielding u svojoj doktorskoj disertaciji
- nije standard već arhitekturni stil
- sve se temelji na resursima koji su predstavljeni URL-ovima:
 - `http://localhost:8080/RassusRest/rest/persons` - popis svih korisnika
 - `http://localhost:8080/RassusRest/rest/persons/1` - korisnik s identifikatorom 1
- koristi protokol: HTTP (GET, POST, PUT, DELETE, PATCH)
- koristi podatke: JSON, XML, sirovi (npr. za slike, video)
- bez stanja (*stateless*), priručni spremnik (*cache*)

Format JSON (Javascript Object Notation)

- podatak - par: **ime/vrijednost**

"firstName": "Ivana"

- vrijednosti mogu biti:

- broj, *string*, *boolean* (`true`, `false`), polje, objekt, null

- objekt su parovi u vitičastim zagradama, npr.:

{ "firstName" : "Ivana", "lastName" : "Podnar Žarko" }

- ime u objektu mora biti jedinstveno

- polje su vrijednosti u uglatim zagradama, npr.:

```
[ { "name": "Ignac Lovrek", ... },  
  { "name": "Ivana Podnar Žarko", ... },  
  ... ]
```

Primjer podataka u JSON-u i XML-u

- JSON:

```
{  
  "firstName" : "Ivana",  
  "lastName" : "Podnar Žarko",  
  "room" : "C7-12",  
  "phone" : "261",  
  "_links" : {  
    "self" : {  
      "href" : "http://localhost:8080/persons/2  
    }  
  }  
}
```

- XML:

```
<person>  
  <firstName>Ivana</firstName>  
  <lastName>Podnar Žarko</lastName>  
  <room>C7-12</room>  
  <phone>261</phone>  
  <links>  
    <link>  
      <rel>self</rel>  
      <href>http://localhost:8080/persons/2</href>  
    </link>  
  </links>  
</person>
```

Usporedba JSON-a i XML-a

- JSON
 - za
 - format
 - ugrađen u preglednike
 - protiv
 - nema ugrađene poveznice (koristi se kao tekst)
 - nema mogućnosti definiranja sheme
 - ograničen skup vrsta podataka
 - problem binarnih podataka
- XML
 - za
 - mogu se zapisati složene strukture podataka
 - ima standardne poveznice
 - dobar skup alata za transformaciju i obradu
 - protiv
 - problem binarnih podataka
 - loš za strukture bez redoslijeda
 - ograničenja DTD-a
 - postoji puno standarda koji ga komplikiraju
 - brzina procesiranja

Primjer zahtjeva i odgovora (1)

HTTP GET <http://localhost:8080/persons>

```
[  
  {  
    "id" : 1,  
    "name" : "Ignac Lovrek",  
  },  
  {  
    "id" : 2,  
    "name" : "Ivana Podnar Žarko",  
  },  
  {  
    "id" : 3,  
    "name" : "Mario Kušek",  
  },  
  {  
    "id" : 4,  
    "name" : "Krešimir Pripužić",  
  },  
  ...  
]
```

Primjer zahtjeva i odgovora (2)

HTTP GET <http://localhost:8080/persons/2>

```
{  
    "firstName" : "Ivana",  
    "lastName" : "Podnar Žarko",  
    "phone" : "261",  
    "room" : "C7-12"  
}
```

Svojstva metoda protokola HTTP

- Svojstva:
 - Sigurna (*safe*) - bez posljedica za podatke
 - *Idempotentna* - može se izvršavati više puta
 - Može se privremeno spremiti odgovor (*cachable*)
- Metode:
 - GET - sigurna, idempotentna, može se privremeno spremiti
 - PUT - idempotentna
 - DELETE - idempotentna
 - HEAD - sigurna, idempotentna
 - POST - ništa
 - PATCH - ništa, ali se može napraviti da je idempotentna (za više vidi [ovdje](#)) što je česti slučaj u praksi

Tipično korištenje usluga REST

- za *Create, Read, Update and Delete* (CRUD)

HTTP	CRUD
POST	Create, (Overwrite/Replace)
GET	Read
PUT	Update, (Create, Delete)
DELETE	Delete
PATCH	Partial update

- primjer gotove usluge: *Twitter*

Dizajniranje usluge

- paziti na mrežu:
 - što i koliko se podataka prenosi
 - koliko zahtjeva i odgovora imamo da bismo nešto prikazali na klijentu
- napraviti URL za svaki resurs
- definirati metode za svaki resurs
- stavljati poveznice u resursima
 - ne vraćati čitavu strukturu
- specificirati format za svaki resurs
- povezati usluge tako da sve kreće preko jednog URL-a
 - HATEOAS - *Hypermedia as the Engine of Application State* (3. razina zrelosti web-usluga)
- preko vrsta medija dogovorati verzije usluga (API-ja)

Zadatak

- Napraviti uslugu koja služi za evidenciju plaćenih računa
- U sustavu imamo osobe
- Svaka osoba ima osobne podatke
- Osobe se mogu stvoriti, obrisati i promijeniti im podatke
- Svaka osoba ima račune koje je platila za pojedini mjesec
- Kada je neki račun unesen više ga nije moguće mijenjati niti obrisati
- Rješenje projekta i klijenata koji ga koriste se nalazi na:
<https://gitlab.tel.fer.hr/spring>

Tablica dizajna usluge

resurs	metode	šalje	svrha
/persons	POST	osoba	stvara novu osobu
	GET		vraća popis osoba
/persons/{id}	GET		vraća osobu s ID-om
	DELETE		briše osobu (brišu se i svi računi te osobe)
	PUT	osoba	mijenja podatke o osobi
	PATCH	dio osobe	mijenja samo podatka koji su poslati
/persons/{id}/bills	POST	račun	stvara novi račun za osobu s ID-om
	GET		vraća popis računa te osobe
/bills/{bid}	GET		vraća račun s ID-om bid

Primjer upita i odgovora - stvaranje resursa

- POST /persons

- sadržaj zahtjeva:

```
{  
    "firstName": "Jura",  
    "lastName": "Jurić",  
    "address": "Unska 3, 1000 Zagreb",  
    "phone": "+385 1 6129 999",  
    "email": "jura.juric@fer.hr"  
}
```

- odgovor:

201 Created

Location: <http://localhost:8080/persons/4>

Primjer upita i odgovora - lista osoba

- GET /persons
- odgovor:

200 OK

```
[  
  {  
    "id" : 1,  
    "name" : "Ignac Lovrek",  
  },  
  {  
    "id" : 2,  
    "name" : "Ivana Podnar Žarko",  
  },  
  {  
    "id" : 3,  
    "name" : "Mario Kušek",  
  },  
  {  
    "id" : 4,  
    "name" : "Krešimir Pripužić",  
  },  
  ...  
]
```

Primjer upita i odgovora - dohvaćanje resursa

- GET /persons/2

- odgovor:

```
{  
    "firstName" : "Ivana",  
    "lastName" : "Podnar Žarko",  
    "address": "Unska 3, 10000 Zagreb",  
    "phone": "+385 1 6129 999",  
    "email: "ivana.podnar@fer.hr"  
}
```

Primjer upita i odgovora - dohvaćanje nepostojećeg

- GET /persons/100
- odgovor:

404 Not Found

Primjer upita i odgovora - promjena resursa

- PUT /persons/2
 - {
 - "firstName" : "Ivana",
 - "lastName" : "Podnar Žarko",
 - "address": "Unska 3, 10000 Zagreb",
 - "phone" : "+385 1 6129 761",
 - "email: "ivana.podnar@fer.hr"
 - }

- odgovor:

204 No Content

Primjer upita i odgovora - stvaranje resursa

- PUT /persons/200

```
{  
    "firstName" : "Ana",  
    "lastName" : "Anić",  
    "address": "Unska 3, 10000 Zagreb",  
    "phone": "+385 1 6129 999",  
    "email: "ana.anic@fer.hr"  
}
```

200 se ignorira

- odgovor:

201 Created

Location: <http://localhost:8080/persons/5>

Primjer upita i odgovora - promjena dijela resursa

- PATCH /persons/2

```
{  
    "phone" : "+385 1 6129 761"  
}
```

- odgovor:

200 OK

```
{  
    "firstName" : "Ivana",  
    "lastName" : "Podnar Žarko",  
    "address": "Unska 3, 10000 Zagreb",  
    "phone" : "+385 1 6129 761",  
    "email: "ivana.podnar@fer.hr"  
}
```

- ovo je idempotentno

Primjer upita i odgovora - promjena nepostojećeg resursa

- PATCH /persons/200
- ```
{
 "phone" : "+385 1 6129 761"
}
```

- odgovor:

404 Not Found

# Primjer upita i odgovora - osoba koja postoji

- DELETE /persons/3
- odgovor:

204 No Content

# Primjer upita i odgovora - osoba koja ne postoji

- DELETE /persons/300
- odgovor:

404 Not Found

# Primjer upita i odgovora - stvaranje resursa

- POST /persons/2/bills

- sadržaj zahtjeva:

```
{
 "month": 3,
 "year": 2019,
 "amount": 250
}
```

- odgovor:

201 Created

Location: <http://localhost:8080/bills/1>

# Primjer upita i odgovora - lista računa

- GET /persons/2/bills
- odgovor:

200 OK

```
[
 {
 "id": 1,
 "peroid": "2019-3"
 },
 {
 "id": 3,
 "peroid": "2019-4"
 },
 ...
]
```

# Primjer upita i odgovora - jedan račun

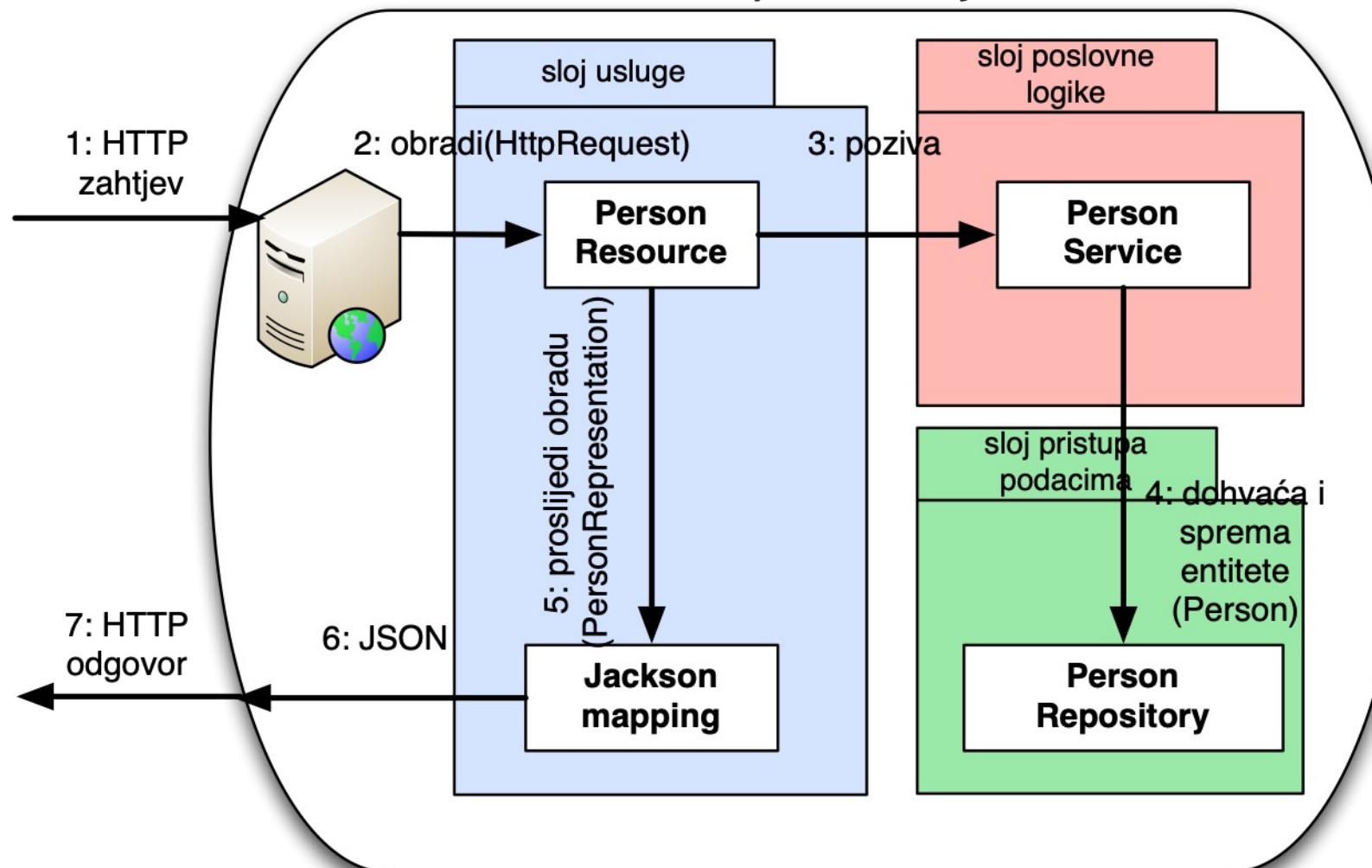
- GET /bills/1
- odgovor:

200 OK

```
{
 "month": 3,
 "year": 2019,
 "amount": 250
}
```

# Arhitektura web-aplikacije (REST)

web-poslužitelj



# Spring Framework - <http://spring.io>



- prva verzija 2003. koju je napravio Rod Johnson
- radni okvir za lakši razvoj aplikacija
- bavi se konfiguracijom objekata u sustavu (*IoC – inversion of control*)
  - upravlja poslovnim objektima kao običnim objektima (*POJO – plain old java objects*)
  - brine se za kreiranje objekata
  - povezuje kreirane objekte (*IoC, wiring up, DI - dependency injection*)
  - upravlja njihovim životnim ciklusom
- složene veze između objekata se definiraju u XML-u ili pomoću bilješki (*annotation*)
- odvaja poslovnu logiku od mehanizama za ispravan rad sustava (transakcije, logiranje, ...)
- vrlo je složen za početnika jer ima puno stvari ugrađeno

# Spring Boot (trenutna verzija 2.7.4)

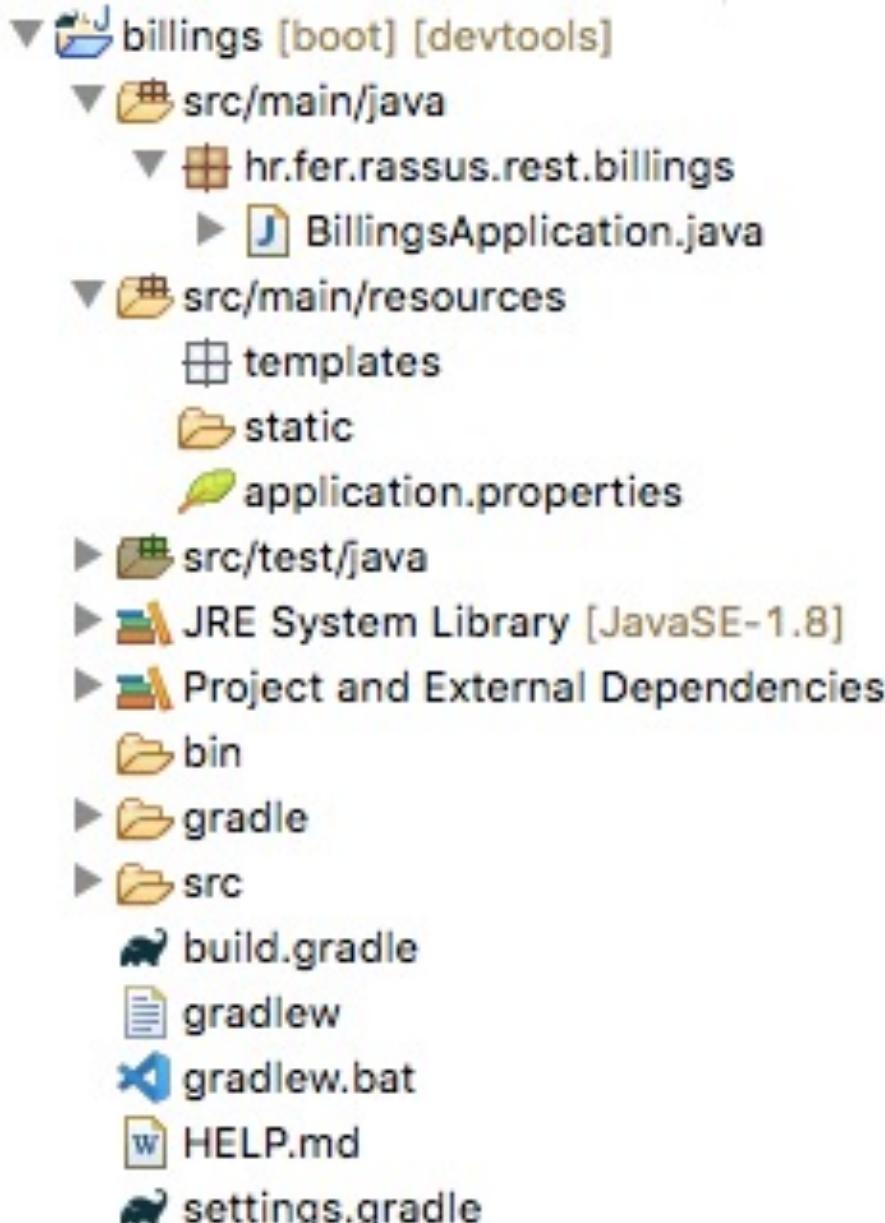
- jedan od podprojekata Springa
  - <http://projects.spring.io/spring-boot/>
- pojednostavljuje korištenje Springa, pogotovo stvaranje novog projekta
- podržava:
  - automatsku konfiguraciju
  - pretraživanja klasa na putu (*path*)
- aplikacija ima manje koda
- kod web-aplikacija - omogućuje izradu samostalnih aplikacija
  - web-poslužitelj zapakiran u jar
  - jednostavnije instaliranje
  - aplikacija spremna za produkcijsku okolinu
- [primjeri projekata](#), [Spring framework guru tutorials](#)
- **Video materijali:**
  - Spring predavanja za prediplomski projekt - [youtube](#),
  - Vještina RUAZOSA - meduza ([1. dio](#), [2. dio](#))

# Stvaranje Spring projekta

- otvoriti stranicu <https://start.spring.io>
- klik na "Switch to the full version"
- popuniti:
  - Group: hr.fer.rassus.rest
  - Artifact: billings
  - Name: billings
  - Packaging: Jar
  - Java Version: 17 (može i novija verzija)
  - Language: Java
- odabratи: Web, HATEOAS, DevTools, (Lombok?)
- klik na Generate Project - napravi zip koji preglednik skine
- trebamo u IDE-u otvoriti projekt u koji smo raspakirali

# Struktura projekta

- pogledati:
  - BillingsApplication
    - klasa koja se pokreće
  - build.gradle
    - skripta za "građenje"
  - application.properties
    - vanjska konfiguracija



# klasa LecturesApplication

```
@SpringBootApplication
public class BillingsApplication {

 // metoda koja sve pokreće
 public static void main(String[] args) {
 SpringApplication.run(BillingsApplication.class, args);
 }
}
```

# Beanovi i DI (*Dependency Injection*)

- Objekte koje stvara i s kojima upravlja Springov kontejner zovu se *Springovi beanovi* (skraćeno samo *bean*)
- DI - *dependency injection*
  - objekti definiraju ovisnosti o drugim objektima
  - mora imati ili: atribute, setere ili konstruktor kroz koji se ovisnosti postavljaju
  - kontejner onda ubacuje ovisnosti kada stvara baenove
  - nepotrebne posebne metode za instanciranje i postavljanje ovisnih objekata da bi objekti normalno funkcionali
- doseg beanova - `@Scope("tip")`
  - **singleton (podrazumijevano)** - jedna instanca u JVM-u
  - prototip - novi objekt svaki put kada se zahtjeva bean
  - zahtjev (*request*) - kod svakog zahtjeva se stvara novi bean
  - sjednica (*session*) - za svakog korisnika jedan bean
  - globalna sjednica - koristi se kod portleta

# Ubrizgavanje beanova

1. preko atributa (*field*) - ne preporuča se

```
@Autowired
private NekiBean b;
```

2. preko konstruktora

```
private NekiBean b;

public XApplication(NekiBean b) {
 this.b = b;
}
```

3. preko setera

```
private NekiBean b;

@Autowired
public void setB(NekiBean b) {
 this.b = b;
}
```

# Definiranje beanova

- dva načina:
  - u konfiguracijskoj klasi (npr. SimpleApplication) definirati metodu koja je označena da vraća bean (@Bean)
  - označiti klasu s @Component
    - mehanizam pretraživanja puta može pronaći takvu klasu
    - podvrste:
      - @Controller - predstavlja kontroler u Web MVC-u
      - @Service - predstavlja namjeru da je to usluga
      - @Repository - predstavlja komponentu koja pristupa podacima
      - @RestController - predstavlja kontroler u Web MVC-u koja vraća podatke koji se serializiraju u JSON ili XML

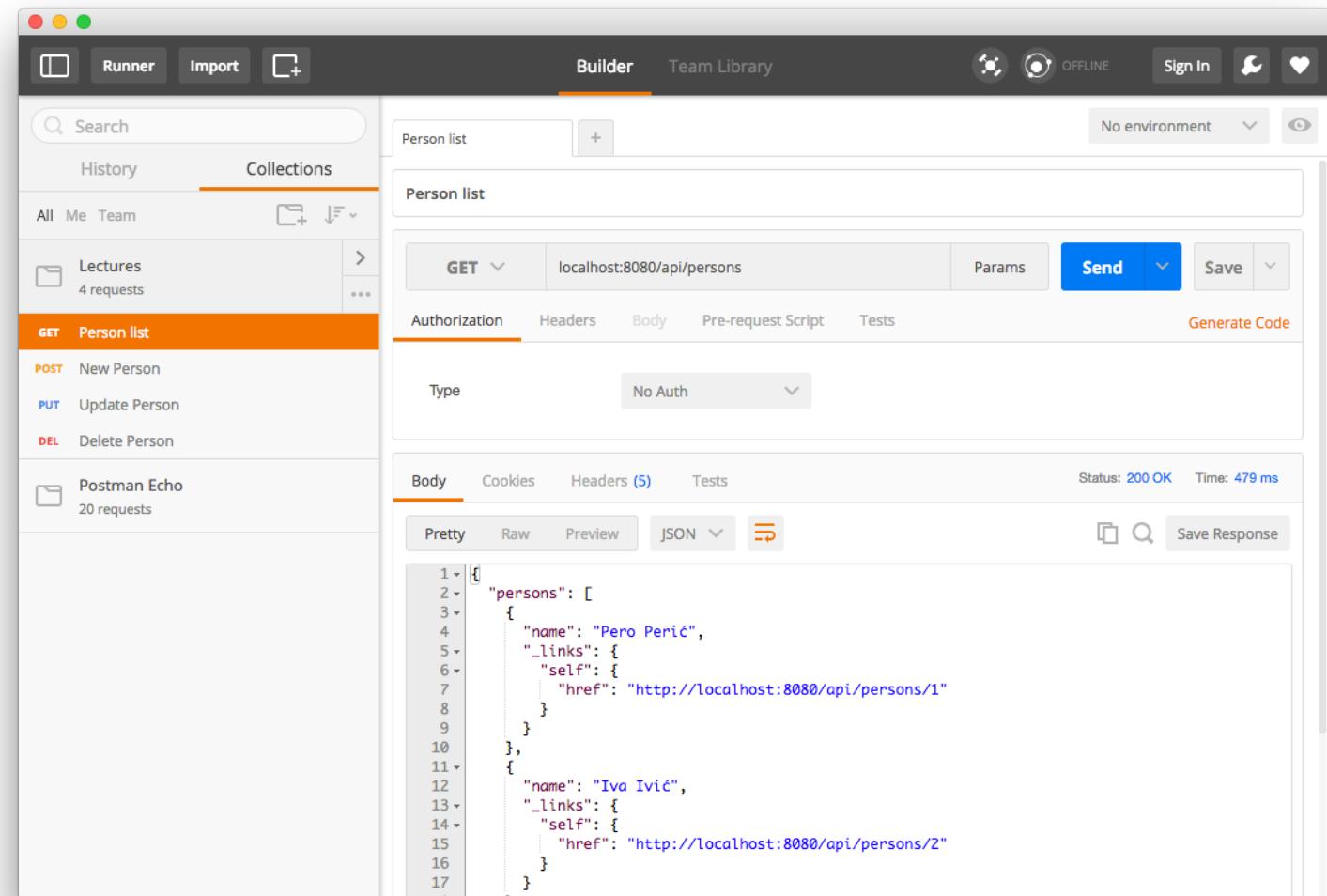
# REST kontroler - *bean*

```
@RestController
public class PersonResourceController {

 @GetMapping("/persons")
 public String getPersonsList() {
 return "Pero i Ana";
 }
}
```

# Primjer dohvaćanja

- Chrome extensions  
(<https://chrome.google.com/webstore/search/rest>):
  - Advanced Rest Client ili
  - Postman
- u terminalu:
  - curl
  - HTTPie



# REST kontroler - *bean* (2)

```
@RestController
public class PersonResourceController {

 @GetMapping("/persons")
 public String getPersonsList() {
 return "Pero i Ana";
 }

 @GetMapping("/persons/{id}")
 public String getPerson(@PathParam("id") String id) {
 return "Ana";
 }
}
```

# REST kontroler - bean (3)

```
@RestController
public class PersonResourceController {

 @GetMapping("/persons")
 public String getPersonsList() {
 return "Pero i Ana";
 }

 @GetMapping("/persons/{id}")
 public PersonRepresentation getPerson(@PathParam("id") String id) {
 return new PersonRepresentation("Ana", "Anić",
 "Unska 3, 10000 Zagreb", "+385 1 6129 999", "ana.anic@fer.hr");
 }
}

public class PersonRepresentation {
 private String firstName, lastName, address, phone, email;

 // getters, setters, konstruktori (prazan, atributi), toString
}
```

# PersonService

```
@Service
public class PersonService {

 private int pidCounter = 0;
 private Map<Integer, Person> persons = new HashMap<>();

 public Collection<Person> getPersons() {
 return persons.values();
 }
}
```

# PersonResourceController

```
@RestController
public class PersonResourceController {

 private PersonService personService;

 public PersonResourceController(PersonService personService) {
 this.personService = personService;
 }

 @GetMapping("/persons")
 public Collection<ShortPersonRepresentation> getPersonsList() {
 return personService.getPersons().stream()
 .map(p -> PersonAssembler.toShortPersonRepresentation(p))
 .collect(Collectors.toList());
 }

 ...
}
```

# ShortPersonRepresentation

```
public class ShortPersonRepresentation {
 private int id;
 private String name;

 // setters/getters/konstruktori
}
```

# PersonAssembler

```
public class PersonAssembler {

 public static ShortPersonRepresentation toShortPersonRepresentation(
 Person person) {
 return new ShortPersonRepresentation(
 person.getId(),
 person.getFirstName() + " " + person.getLastName());
 }

 public static PersonRepresentation toPersonRepresentation(Person person)
 { ... }

 public static Person toPerson(PersonRepresentation personRepresentation)
 { ... }

 public static Person toPerson(int id,
 PersonRepresentation personRepresentation) { ... }

 public static void updatePersonForNotNullValues(Person person,
 PersonRepresentation personRepresentation) { ... }
}
```

# Dohvaćanje jedne osobe

```
@RestController
public class PersonResourceController {
 ...
 @GetMapping("/persons/{id}")
 public ResponseEntity<PersonRepresentation> getPerson(
 @PathVariable("id") Integer id)
 {
 Person person = personService.getPerson(id);
 if(person != null) {
 return ResponseEntity.ok(
 PersonAssembler.toPersonRepresentation(person));
 }
 return ResponseEntity.notFound().build();
 }
}
```

# Kreiranje nove osobe

```
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.methodOn;
...
@RestController
@RequestMapping("/persons")
public class PersonResourceController {
...
 @PostMapping()
 public ResponseEntity<?> newPerson(
 @RequestBody PersonRepresentation personRepresentation) {
 Person person = PersonAssembler.toPerson(personRepresentation);
 personService.newPerson(person);

 return ResponseEntity
 .noContent()
 .location(linkTo(
 methodOn(this.getClass()).getPerson(person.getId())).toUri())
 .build();
 }
...
}
```

# Model zrelosti web-usluga

- model je napravio Leonard Richardson
  - <http://www.crummy.com/writing/speaking/2008-QCon/act3.html>
- Razine:
  - 0
    - pozivanje usluga je većinom pozivanje udaljenih procedura
    - jedan URI jedna HTTP metoda (većinom XML-RPC i SOAP)
  - 1
    - koriste više resursa, ali su nazivi metoda i parametara enkodirani u URL
    - više URI-ja, jedna HTTP meotda (GET)
  - 2
    - koriste više resursa i HTTP metoda te statusne kodove
    - ne koriste svoju shemu (vrste podataka koji se prenose)
    - primjer Amazon S3
  - 3
    - isto kao razina 2, ali koristi hipermedijske vrste
    - resurs opisuje svoje mogućnosti i veze

# Standardi, prijedlozi ...

- URI Template - ožujak 2012., IETF, [RFC 6570](#)
  - standard za ekspanziju varijabli iz URI-ja
- JSON Hypertext Application Language (HAL) - svibanj 2016., IETF, v08
  - [draft-kelly-json-hal-08](#)
  - prijedlog višemedijskih tipova za reprezentaciju resursa i njihovih relacija
- Application-Level Profile Semantics (ALPS) - svibanj, 2021., IETF, v07
  - [draft-amundsen-richardson-foster-alps-07](#)
  - format podataka za opis aplikacije i njihove semantike
- [OpenAPI inicijativa \(konzorcij\)](#)
  - osnovana 2016. u sklopu Linux Foundationa
  - OpenAPI specification (bivši Swagger 2.0 spec. 3.0.3)
  - aktualna verzija [v3.1.0](#)



# Pitanja za ponavljanje

- Koje su dvije vrste web-aplikacija i koja je razlika između njih?
- Kakva je arhitektura web-aplikacije i svrha svakog sloja?
- Koja je razlika između nove i stare arhitekture web-aplikacija?
- Čemu služe skripte na klijentu?
- Što je i kako radi AJAX?
- Usporedite dva načina slanja događaja s poslužitelja (web-aplikacije) klijentu (preglednik).
- Što su i čemu služe web-usluge?
- Objasnite dva načina kako web-aplikacija koristi web-usluge.
- Koja je razlika između Weba 1.0 i Weba 2.0?
- Koje su 3 vrste web-usluga i razlike između njih?

# Pitanja za ponavljanje

- Što je SOAP i čemu služi?
- Što je WSDL i čemu služi?
- Kakva je struktura WSDL-a u čemu služe pojedini elementi?
- Što je UDDI i čemu služi?
- Koja je razlika između web-usluga RPC i temeljenih na dokumentima?
- Kakve su web-usluge temeljene prijenosu prikaza stanja resursa i što ih karakterizira?
- Koje su razlike između JSON-a i XML-a?
- Koja su svojstva metoda protokola HTTP te ih objasnite?
- Objasnite model zrelosti web-usluga.
- Na što treba paziti kod dizajniranja REST usluge?
- Objasniti postupak dizajniranja REST usluge?



SVEUČILIŠTE U ZAGREBU



**Diplomski studij  
Računarstvo**

Znanost o mrežama  
Programsko inženjerstvo i  
informacijski sustavi  
Računalno inženjerstvo  
**Ostali (slobodni izborni  
predmet)**

# Raspodijeljeni sustavi

## 4. Formalni model raspodijeljenog sustava i primjeri raspodijeljenih algoritama

Ak. god. 2022./2023.

# Creative Commons



- slobodno smijete:

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
  - **prerađivati** djelo



- pod sljedećim uvjetima:

- **imenovanje**: morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
  - **nekomercijalno**: ovo djelo ne smijete koristiti u komercijalne svrhe.
  - **dijeli pod istim uvjetima**: ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.



*U slučaju daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.*

*Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.*

*Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.*

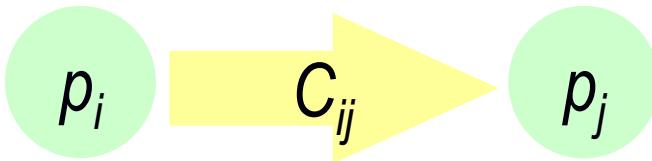
*Tekst licence preuzet je s <http://creativecommons.org/>*

# Sadržaj predavanja

- Model raspodijeljenog sustava
  - model raspodijeljenog izvođenja
  - uzročna ovisnost događaja
  - globalno stanje raspodijeljenog sustava
  - raspodijeljeni algoritam
- Proširenje osnovnog modela raspodijeljenog sustava
  - sinkroni model
  - asinkroni model

# Osnovni model raspodijeljenog sustava

- skup autonomnih procesa  $p_1, p_2, \dots, p_n$
- $C_{ij}$  – kanal koji povezuje procese  $p_i$  i  $p_j$
- $m_{ij}$  – poruka od  $p_i$  za  $p_j$

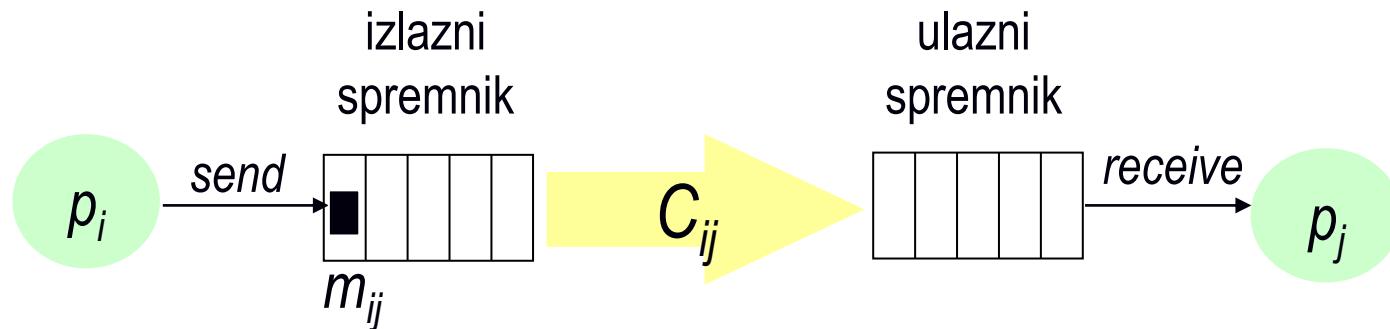


# Svojstva

- Izvođenje procesa i prijenos poruka su asinkroni
- Procesi ne dijele zajednički memorijski prostor
- Pri komunikaciji procesa neminovno se javlja kašnjenje
- Procesi ne koriste jedinstveni globalni sat

# Komunikacija procesa

- procesi međusobno komuniciraju razmjenom poruka (*message passing*) preko komunikacijskog medija (komunikacijske mreže)

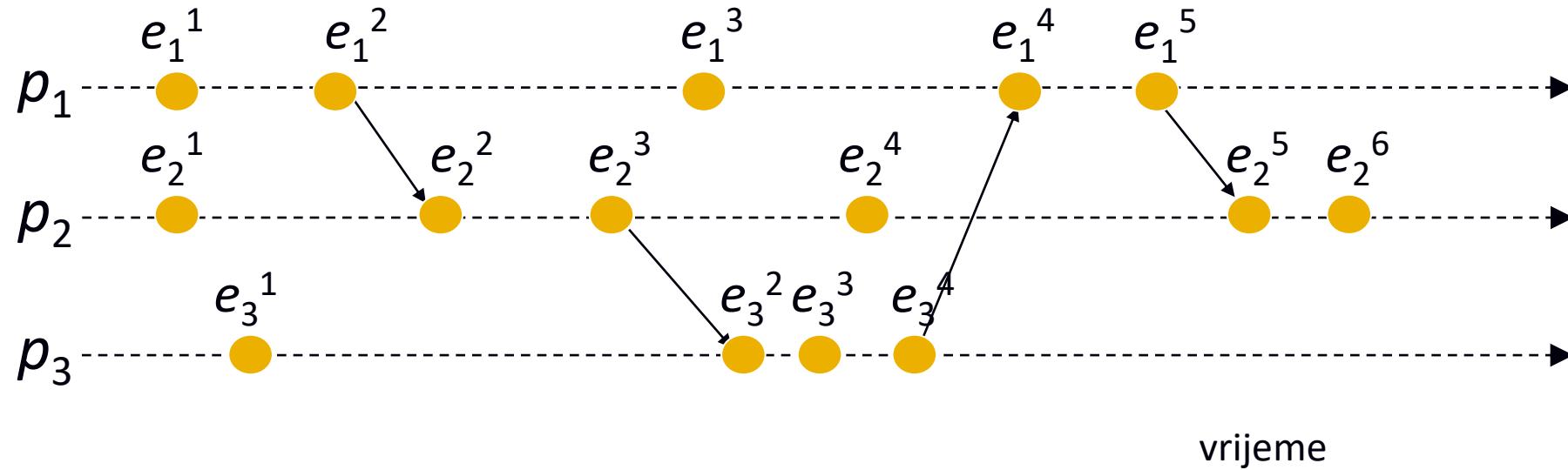


- procesi koriste operatore *send* i *receive*
- send*: pohranjuje poruku u izlazni spremnik i priprema za prijenos preko kanala
- receive*: čita poruku iz dolaznog spremnika i proslijeđuje procesu

# Model raspodijeljenog izvođenja

- Izvođenje procesa: slijedno izvođenje akcija procesa
- **Akcije** se modeliraju sljedećim događajima:
  - unutarnji događaj
  - slanje poruke
  - primanje poruke
- Događaj mijenja stanje procesa i komunikacijskog kanala
- Slijed događaja na procesu  $p_i$ :
$$e_i^1, e_i^2, e_i^3, \dots, e_i^x$$
$$(e_i^2 \text{ se dogodio prije } e_i^3)$$

# Primjer raspodijeljenog izvođenja



# Uzročna ovisnost događaja (1)

- Uzročna relacija ovisnosti događaja (oznaka:  $\rightarrow$ )
  - izražava uzročnu ovisnost između dva događaja tijekom raspodijeljenog izvođenja, uzročnost može biti direktna ili tranzitivna
- $e_i^x \rightarrow e_i^y$ 
  - događaj  $e_i^x$  je izvršen na procesu  $p_i$  prije događaja  $e_i^y$  te su oni uzročno povezani ( $e_i^x$  se nužno dogodio prije  $e_i^y$ )
- $send(m) \rightarrow_{msg} receive(m)$ 
  - uzročna ovisnost vezana uz slanje i primanje poruke, da bi poruka bila primljena, mora prethodno nužno biti poslana na kanal
- $e_i^x \rightarrow e_k^z \wedge e_k^z \rightarrow e_j^y \Rightarrow e_i^x \rightarrow e_j^y$ 
  - primjer tranzitivne uzročnosti događaja izvršenih na 3 različita procesa

# Uzročna ovisnost događaja (2)

- Kada su 2 događaja uzročno ovisna?

$$e_i^x \rightarrow e_j^y \Leftrightarrow \begin{cases} e_i^x \rightarrow e_j^y, (i = j) \wedge (x < y) & \text{slijedni događaji na istom procesu} \\ e_i^x \rightarrow_{msg} e_j^y & \text{slanje i primanje poruke } msg \\ e_i^x \rightarrow e_k^z \wedge e_k^z \rightarrow e_j^y & \text{tranzitivna uzročnost} \end{cases}$$

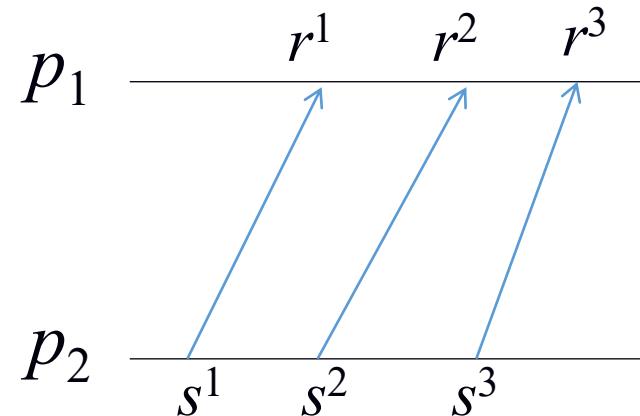
# Uzročna neovisnost događaja

- Uzročna relacija neovisnosti dvaju događaja (oznaka:  $\not\rightarrow$ )
  - označava neovisnost dvaju događaja tijekom raspodijeljenog izvođenja
- $e_i \not\rightarrow e_j$ 
  - događaj  $e_j$  nije ovisan o događaju  $e_i$
- Vrijede sljedeća pravila
  1. za 2 događaja  $e_i$  i  $e_j$ ,  $e_i \not\rightarrow e_j \Rightarrow e_j \not\rightarrow e_i$
  2. za 2 događaja  $e_i$  i  $e_j$ ,  $e_i \rightarrow e_j \Rightarrow e_j \not\rightarrow e_i$
  3. ako za 2 događaja  $e_i$  i  $e_j$ , vrijedi  $e_i \not\rightarrow e_j$  i  $e_j \not\rightarrow e_i$ , onda su  $e_i$  i  $e_j$  konkurenti događaji i to možemo napisati na sljedeći način  $e_i \parallel e_j$

# Model komunikacijskog kanala (1/3)

## FIFO (first-in, first-out)

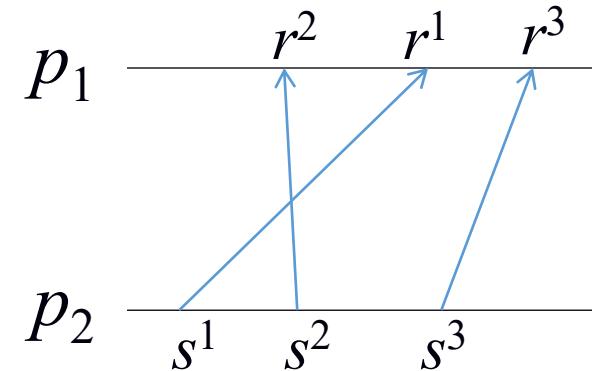
- kanal čuva slijednost poruka,  
ponaša se kao rep



FIFO

## non-FIFO

- kanal ne čuva slijednost poruka,  
ponaša se kao skup

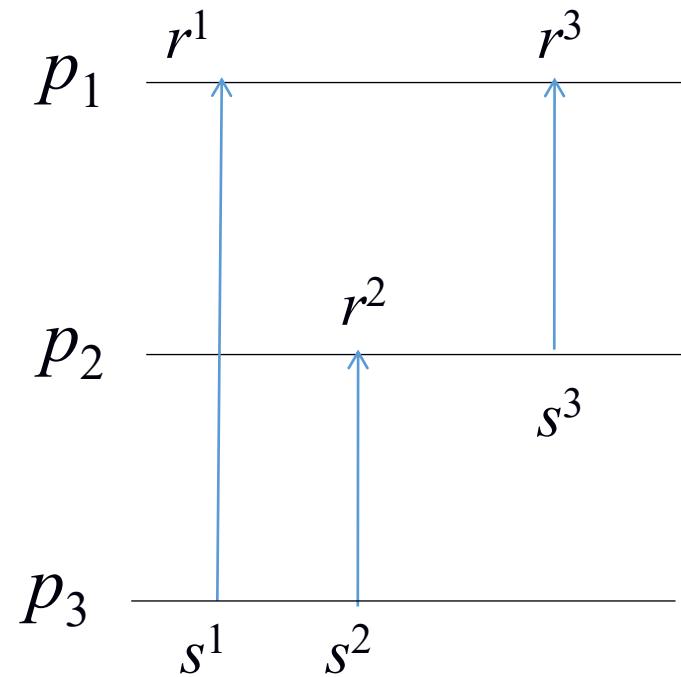


non-FIFO

# Model komunikacijskog kanala (2/3)

## sinkrona slijednost

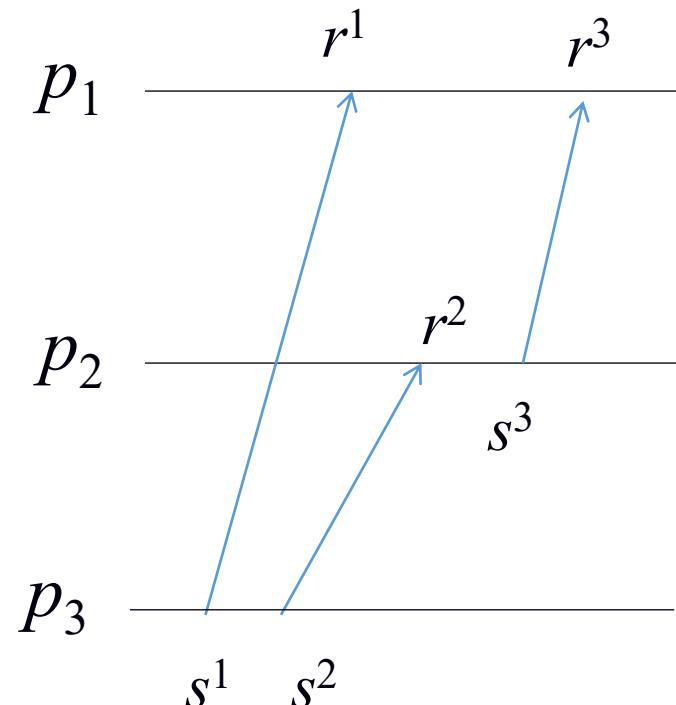
- slanje i primanje poruke događa se istovremeno



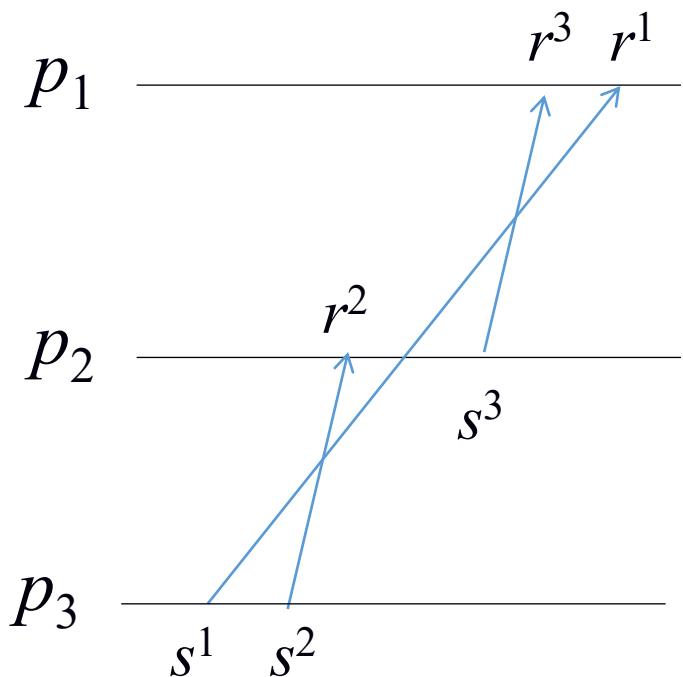
# Model komunikacijskog kanala (3/3)

uzročna slijednost (*causal ordering*, CO)

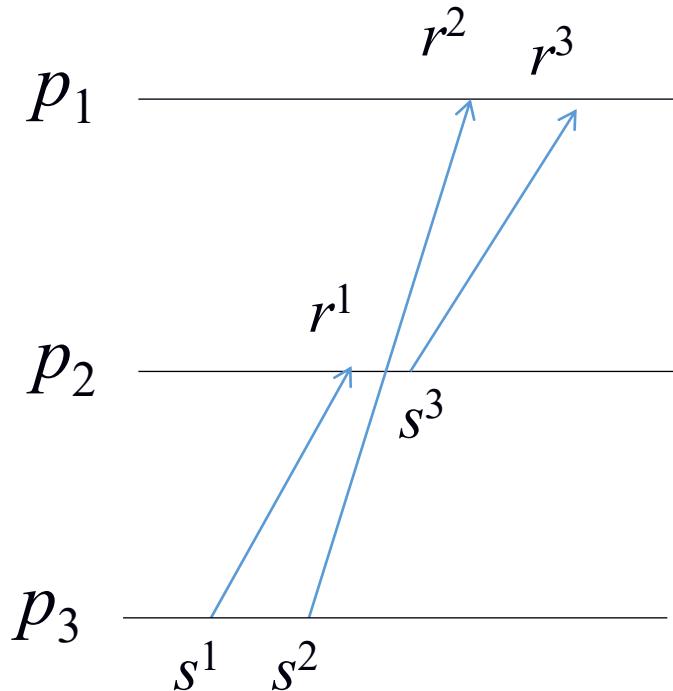
- osigurava da uzročno povezani događaji slanja dviju poruka istom primatelju rezultiraju primanjem u slijedu kojim su poslati
  - ako su dva događaja slanja poruke istom primatelju uzročno povezana (kao  $s^1$  i  $s^3$ ) onda primanje poruka mora slijediti redoslijed slanja ( $r^1$  se mora dogoditi prije  $r^3$  kako bi bilo zadovoljeno svojstvo CO)
- koristan za razvoj distribuiranih algoritama, pojednostavljuje razvoj jer ima ugrađeni mehanizam „sinkronizacije“
  - npr. replicirana baza podataka, svaki proces koji osvježava repliku mora primiti zahtjeve za update u istom slijedu (važno zbog konzistentnosti baze)



# Primjer izvođenja non-CO i CO



**non-CO**  
 $s^1 \rightarrow s^3$ , jer na  $p_1$  se dogodilo  $r^3 \rightarrow r^1$



**CO**  
 $s^1 \rightarrow s^2$ ,  $s^1 \rightarrow s^3$  ali odredišta poruka se razlikuju, a kada analiziramo  $r^2$  i  $r^3$ , poruke slanja  $s^2$  i  $s^3$  su neovisne pa je njihov redoslijed irelevantan

# Globalno stanje raspodijeljenog sustava

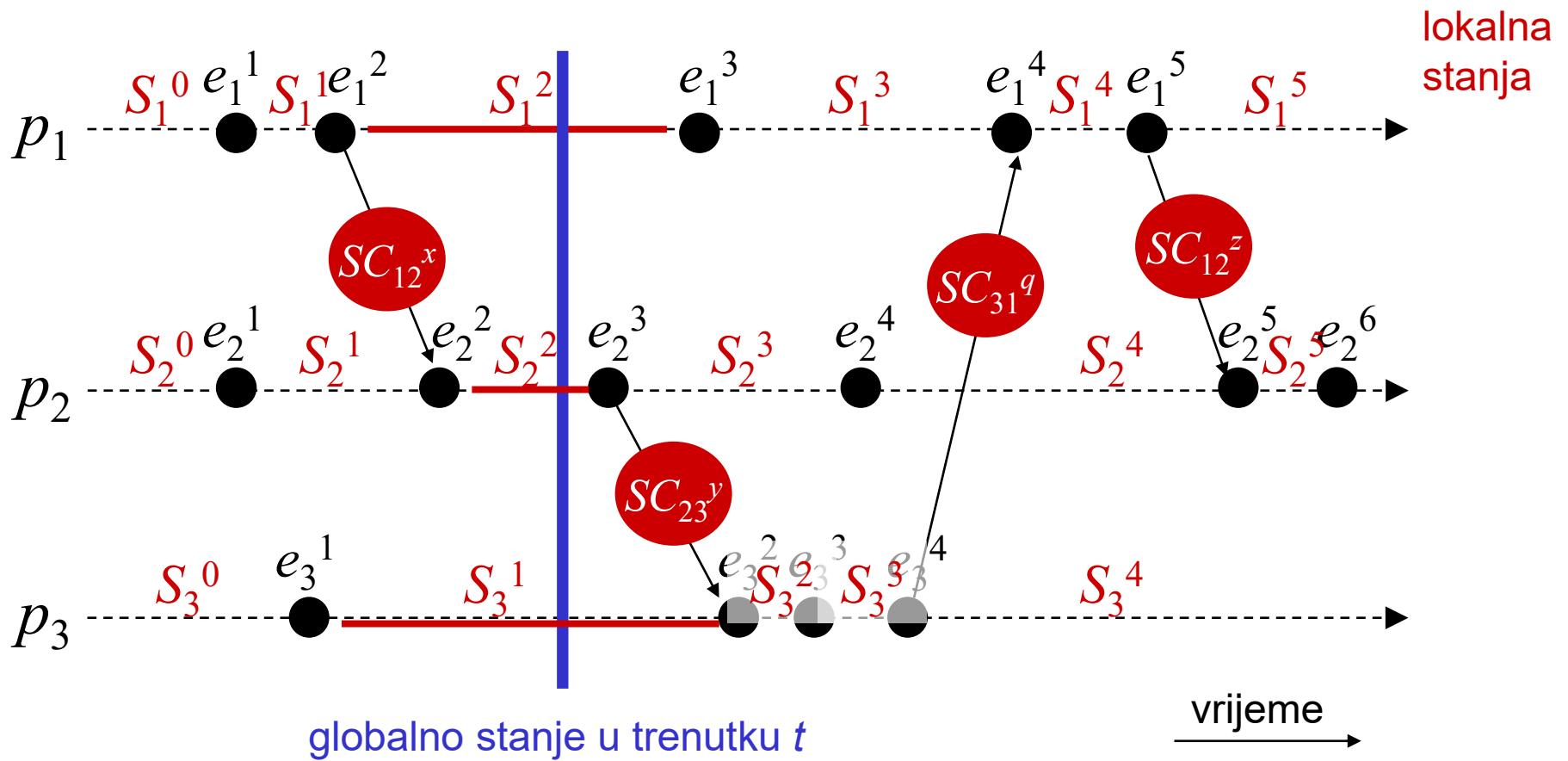
## Lokalno stanje procesa ili kanala

- Stanje procesa određeno je stanjem lokalne memorije i izvođenjem unutarnjih događaja, lokalno stanje procesa je potpuno privatno (ne može ga mijenjati drugi proces)
- Stanje kanala određeno je skupom primljenih i poslanih poruka

## Globalno stanje

- Određeno (trenutnim) lokanim stanjima svih procesa i kanala, kontinuirano se mijenja uslijed akcija tj. događaja na procesima i kanalima
- Izvođenje događaja mijenja lokano stanje procesa/kanala te istovremeno mijenja i globalno stanje raspodijeljenog sustava

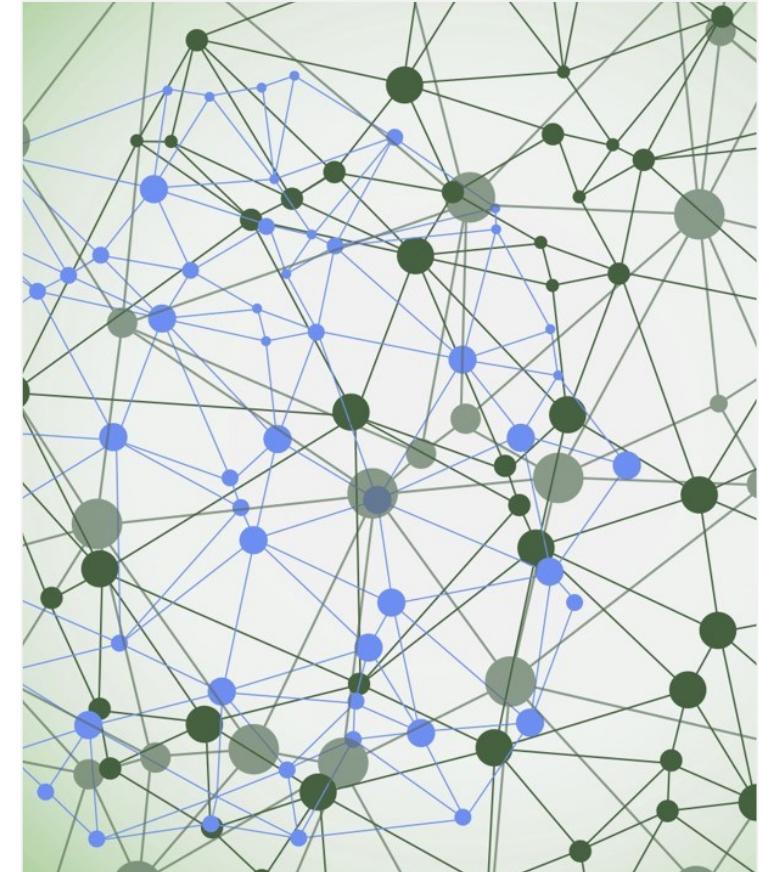
# Primjer lokalnog/globalnog stanja



$$GS(t) = \{S_1^2, S_2^2, S_3^1, SC_{12}^x, SC_{23}^y, SC_{31}^q\}$$

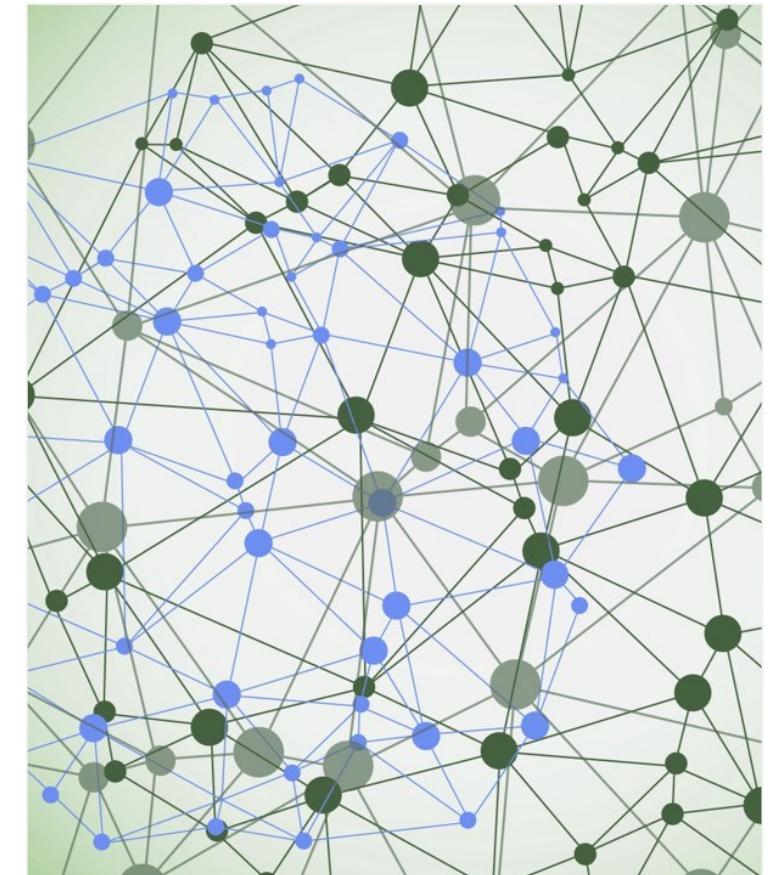
# Raspodijeljeni (distribuirani) algoritam

- Algoritam koji se izvodi u raspodijeljenoj okolini na 2 ili više procesa koji komuniciraju razmjenom poruka
- Definira akcije koje izvodi svaki proces sustava, uključujući prijenos poruka između procesa. Poruke se prenose radi prijenosa informacija i omogućuju koordinaciju aktivnosti procesa koja vodi zajedničkom cilju – implementacija funkcionalnosti koju nudi raspodijeljeni algoritam.



# Raspodijeljeni (distribuirani) algoritam

- izvodi se na većem broju računala povezanih mrežom
- nije jednostavno odrediti početak i kraj izvođenja algoritma, koliko procesa izvodi raspodijeljeni algoritam u svakom trenutku, globalno stanje sustava
- treba biti otporan na ispade procesa jer je to djelomični ispad sustava
- **komunikacija složenost:** brojimo poruke koje se generiraju tijekom izvođenja algoritma



# Sadržaj predavanja

- Model raspodijeljenog sustava
  - model raspodijeljenog izvođenja
  - uzročna ovisnost događaja
  - globalno stanje raspodijeljenog sustava
  - raspodijeljeni algoritam
- Proširenje osnovnog modela raspodijeljenog sustava
  - sinkroni model
  - asinkroni model

# Proširenje osnovnog modela (sinkroni i asinkroni)

## Sinkroni model

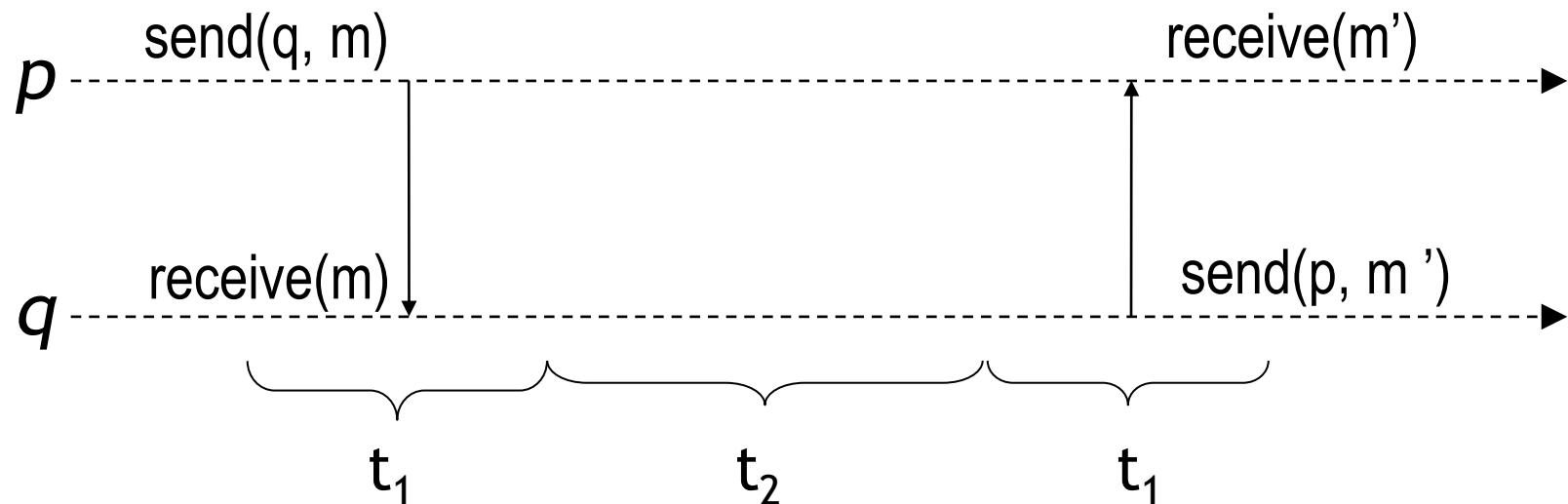
- prepostavka: svi procesi raspodijeljenog sustava izvode događaje (tj. korake) istovremeno
- pojednostavljenje koje nije realno za raspodijeljene sustave, ali može biti korisno za njihovo razumijevanje i analizu

## Asinkroni model

- prepostavka: procesi izvode događaje u proizvolnjom slijedu
- postoji neodređenost vezana uz slijed događaja
- realna situacija

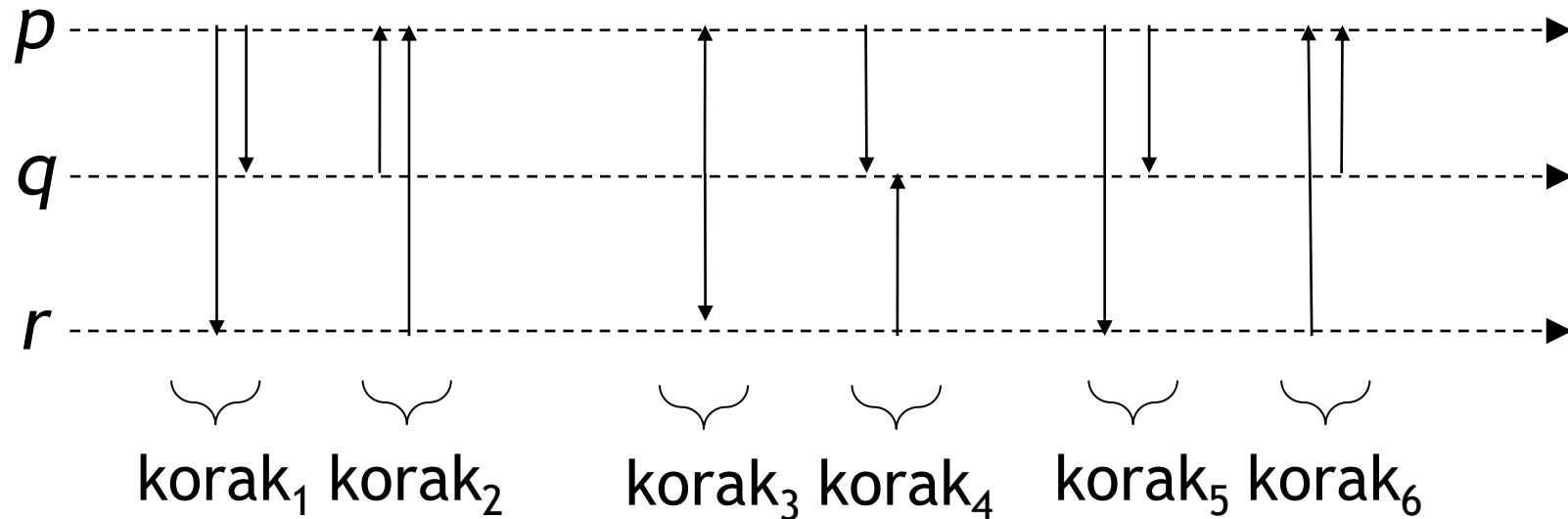
# Sinkroni model

- Poznata je gornja vremenska granica za
  - trajanje prijenosa poruke kanalom ( $t_1$ ) i izvođenje prijelaza nekog procesa ( $t_2$ )
- Pretpostavka
  - procesi imaju potpuno sinkronizirana lokalna vremena



# Primjer sinkrone komunikacije

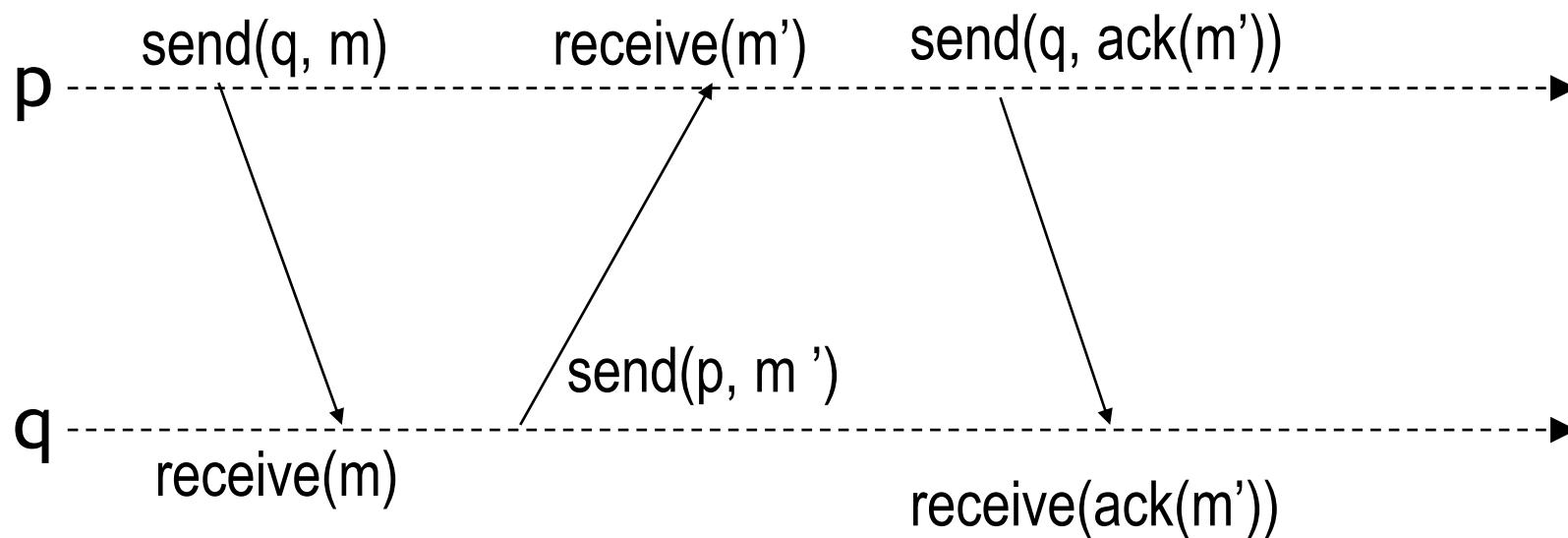
- izvođenje algoritma u sinkronom sustavu organizirano je u koracima
  - pošalji poruke procesima u sustavu
  - primi poruke od drugih procesa u sustavu
  - izvođenje prijelaza: promijeni stanje na temelju primljenih poruka



# Asinkroni model

- Ne postoji gornja vremenska granica za
  - izvođenje prijelaza nekog procesa (no trajanje prijelaza je uvijek konačno)
  - trajanje prijenosa poruke kanalom
- Pretpostavka
  - procesi **nemaju** sinkronizirana lokalna vremena
- Realni slučaj koji ćemo najčešće razmatrati, znatno komplificira model i analizu distribuiranog algoritma raspodijeljenog sustava

# Primjer asinkrone komunikacije

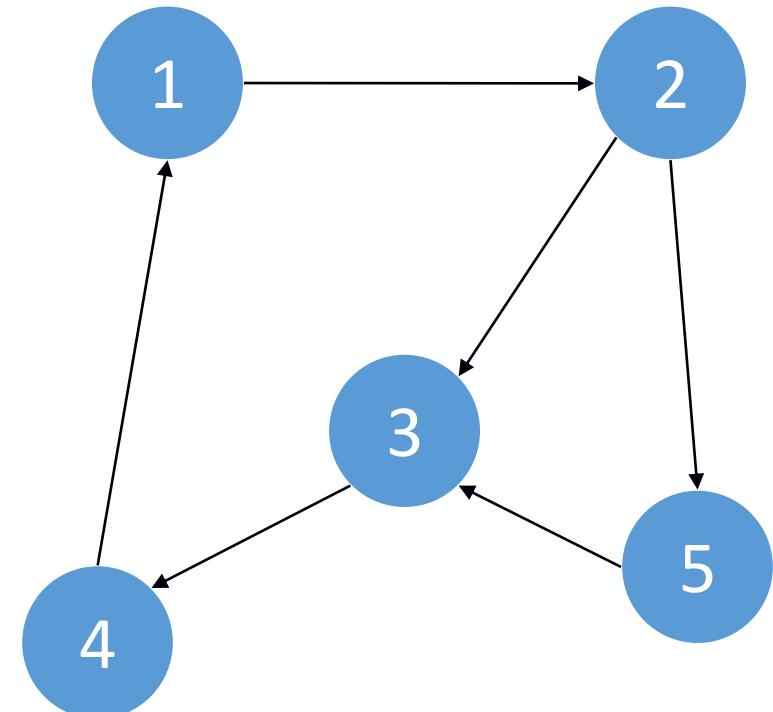


nepouzdani komunikacijski medij, potrebno je modelirati vjerojatnost gubitka poruke na kanalu

# Sinkroni model raspodijeljenog sustava

# Sinkroni model

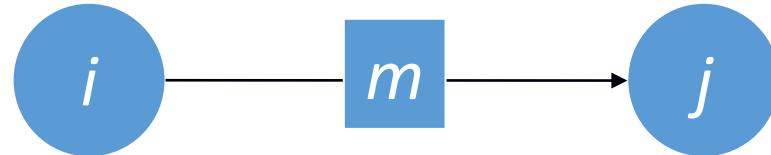
- usmjereni graf  $G = (V, E)$
- $v_i \in V$ , čvor modelira **proces**
- $e_j \in E$ , grana modelira **kanal**
- $M$  je skup poruka, *null* ako na kanalu nema poruka
- $out-nbrs_i$  – izlazni susjedi
- $in-nbrs_i$  – ulazni susjedi
- $distance(i, j)$  – najkraći put između  $i$  i  $j$  u  $G$ , ( $i, j \in V$ )
- $diameter(G)$  – max  $distance(i, j)$  za sve parove  $(i, j)$



# Model procesa

- svaki se proces vezan uz čvor  $v_i \in V$  modelira kao uređena četvorka:  $(states_i, start_i, msgs_i, trans_i)$
- $\underline{states_i}$  – skup mogućih stanja procesa
- $start_i$  – skup početnih stanja
  - $start_i \subset states_i$ ,  $start_i \neq \emptyset$
- $msgs_i$  – funkcija za generiranje poruka
  - određuje izlaznu poruku za svakog susjeda na temelju trenutnog stanja procesa
  - $states_i \times out-nbrs_i \rightarrow M_i \subset M \cup \{\text{null}\}$
- $trans_i$  – funkcija prijelaza, određuje sljedeće stanje na temelju trenutnog stanja i primljenih poruka od ulaznih susjeda

# Model kanala

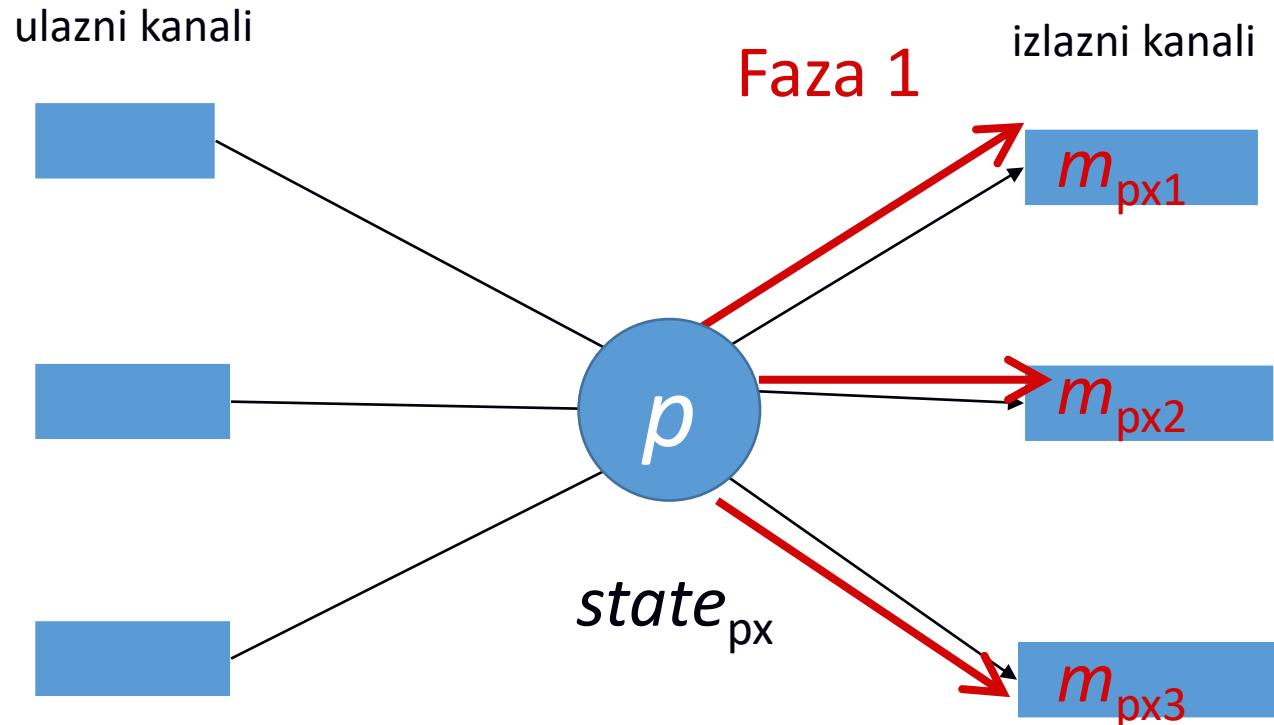


- modelira ga grana između para čvorova  $(i, j)$  iz  $G$
- može primiti poruku  $m$  iz definiranog skupa poruka  $M$  ili  $null$
- $null$  označava praznu poruku

# Izvođenje sinkronog modela

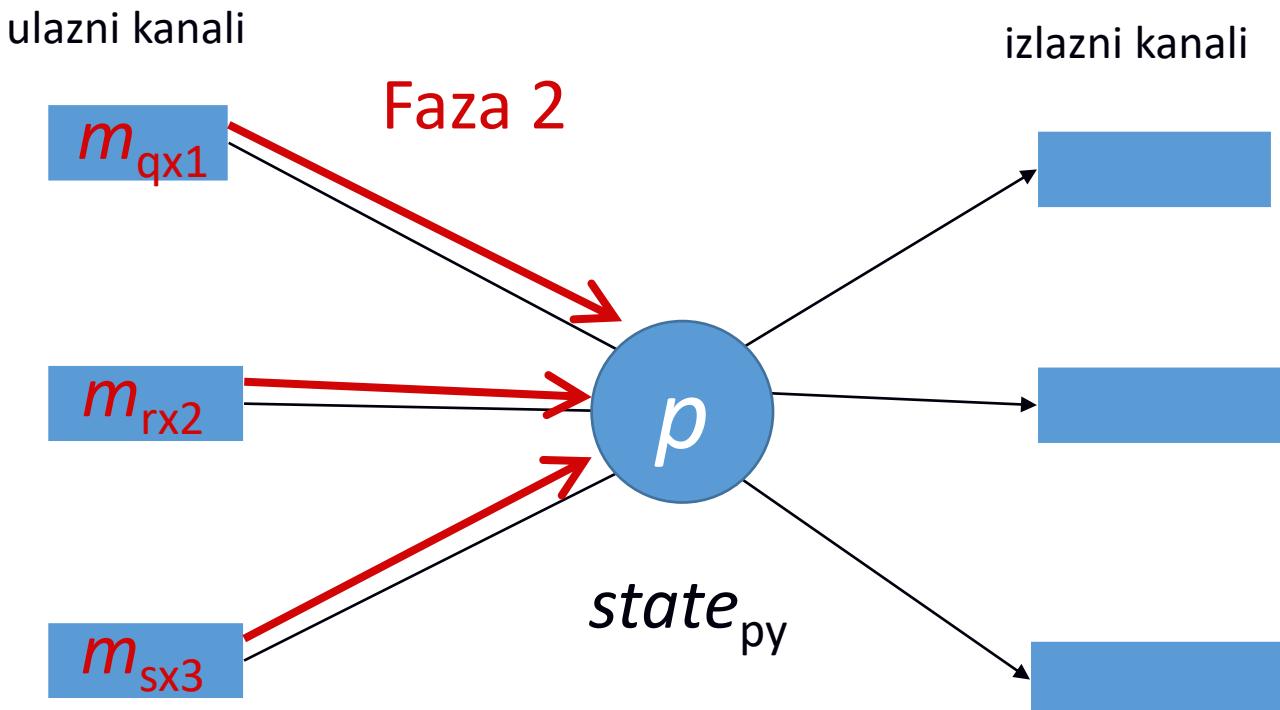
- Algoritmi za sinkrone modele se izvode u **koracima** (*round*). Inicijalno su svi procesi u proizvoljnom početnom stanju i svi su kanali prazni. Nakon toga se izvode koraci.
- Korak se sastoji od 2 faze
  - **Faza 1:** Za svaki proces primjeni funkciju za generiranje poruka (*msg*), a na temelju trenutnog stanja. Generiraj poruke koje će biti poslane izlaznim susjedima i postavi te poruke na izlazne kanale.
  - **Faza 2:** Primjeni funkciju prijelaza (*trans*) koja će na temelju trenutnog stanja i primljenih poruka odrediti sljedeće stanje procesa. Briši sve poruke na kanalima.

# Izvođenje (faza 1)



na temelju trenutnog stanja generiraj  
poruke i postavi ih na izlazne kanale

# Izvođenje (faza 2)



primijeni funkciju prijelaza koja na temelju  
primljenih poruka određuje sljedeće stanje  
procesa

# Formalni model izvođenja

- Beskonačni slijed:  
 $C_0, M_1, N_1, C_1, M_2, N_2, C_2, \dots$
- $C_k$  - stanje svih procesa nakon  $k$  koraka
- $M_k$  – poslane poruke na svim kanalima nakon  $k$  koraka
- $N_k$  – primljene poruke na svim kanalima nakon  $k$  koraka
- $M_k \neq N_k$  – ako dođe do ispada na nekom kanalu, inače je u sustavima bez gubitaka poruka  $M_k = N_k$  za svaki  $k$

# Složenost sinkronog algoritma

- vremenska složenost
  - mjeri se brojem izvedenih koraka (*rounds, r*) koji dovodi do završnog stanja algoritma tj. do stanja u kome su svi procesi zaustavljeni ili kada se više ne proizvode novi izlazi
- komunikacijska složenost
  - mjeri se broj kreiranih i poslanih poruka na kanalima

(Pri određivanju mjere složenosti uvijek se analizira najgori mogući scenarij izvođenja algoritma!)

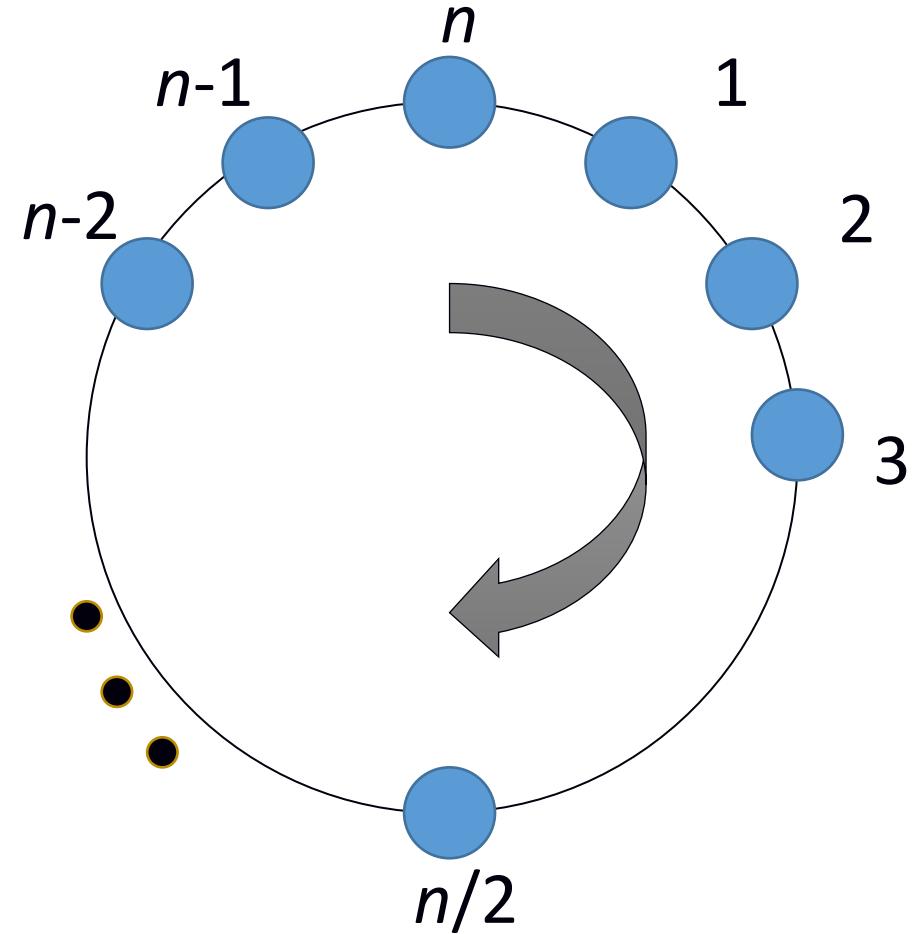
(Usporedite s mjerama složenosti algoritama, vremenska i prostorna)

# Primjeri sinkronog modela

# Problem 1: Odabir vođe u sinkronom prstenu

## Definicija problema

- Izabrati jedinstvenog “vođu” među procesima u mreži
- U bilo kojem koraku samo 1 proces može postati vođa i promijeniti status u *leader*
- Pretpostavka jednostavne mreže od  $n$  čvorova
  - *token ring*
  - svaki čvor je označen brojem od 1 do  $n$



# Dodatni zahtjevi

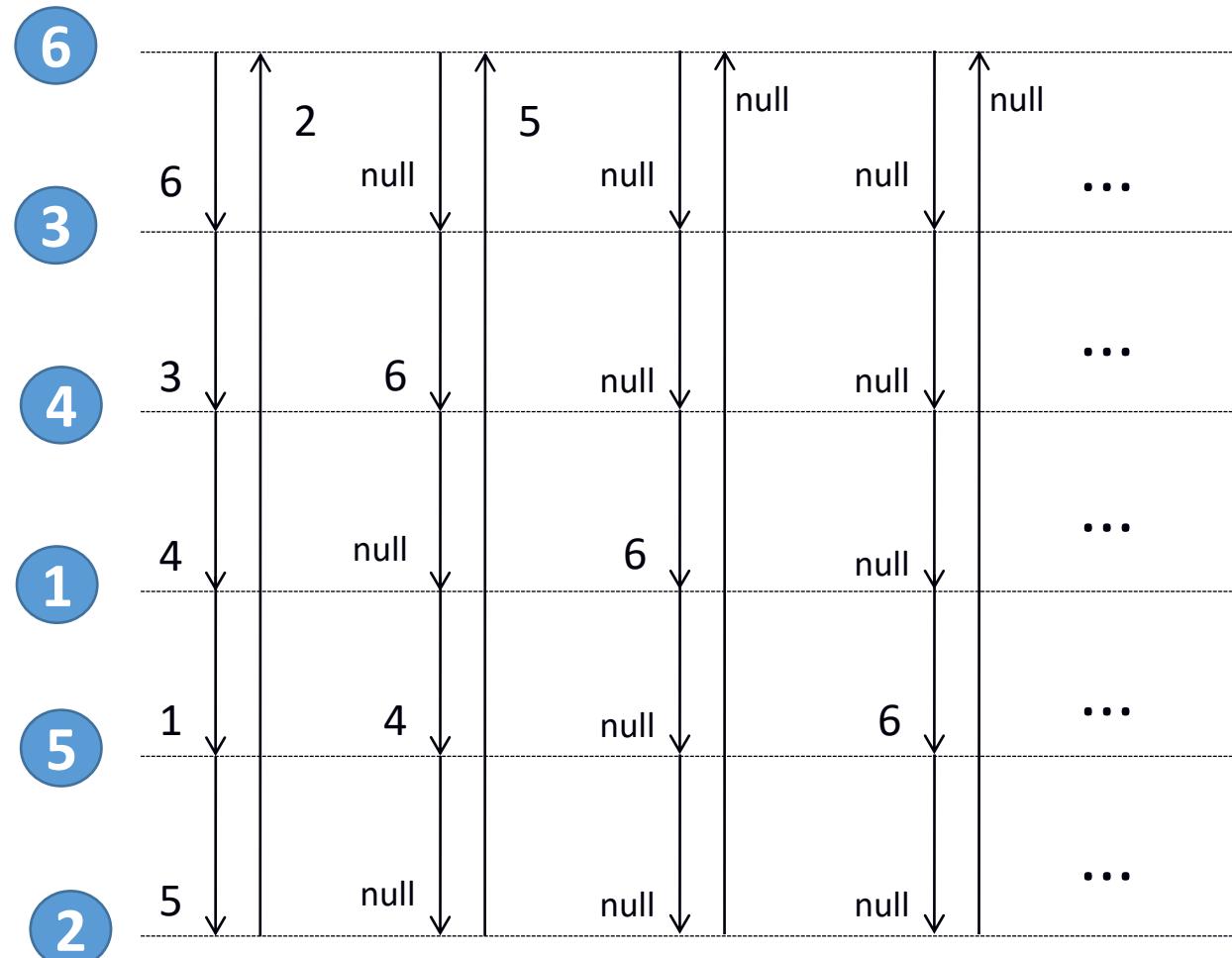
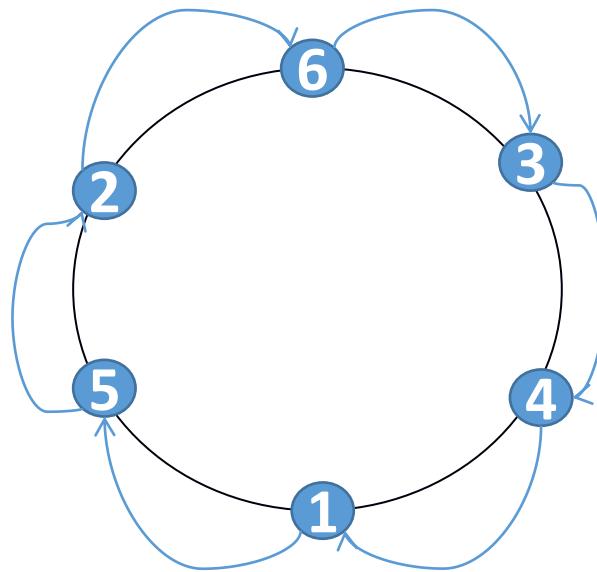
- Svi procesi su identični osim po identifikatoru
  - svaki ima jedinstven identifikator (UID - *Unique ID* )
  - UID nisu sljedbenici u prstenu
  - UID se mogu međusobno uspoređivati
  - (to omogućuje odabir vođe, inače bi svi procesi bili jednaki i vođa se ne bi mogao odabrati)
- Procesi znaju svoje susjede (ulazni ili izlazni)
- Broj procesa u prstenu ( $n$ ) može biti poznat ili nepoznat svim procesima

# Osnovni algoritam za odabir vođe

- Pretpostavke
  - Jednosmjerna komunikacija među procesima u prstenu (usmjereni graf u smjeru kazaljke na satu)
  - Procesi ne znaju veličinu prstena  $n$
  - Svaki proces ima jedinstveni identifikator UID iz skupa prirodnih brojeva, UID se procesu dodjeljuje na slučajan način
- Vođa je proces s najvećim UID
- Skica algoritma:

Svaki proces inicijalno šalje svoj UID susjedu. Kada proces primi UID, ako je taj veći od njegovog UID-a prosljeđuje ga dalje, ako je primljeni UID malji od njegovog UID-a primljeni UID se odbacuje, a ako je primljeni UID jednak njegovom UID-u proces objavljuje sebe kao vođu

# Primjer prstena i algoritma za odabir vode



# Formalni model osnovnog algoritma

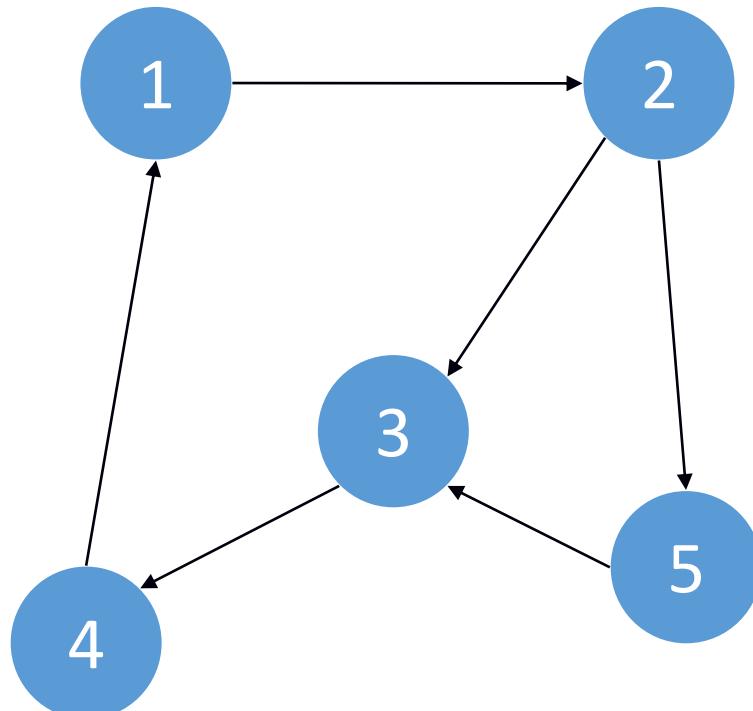
- $M$  – skup poruka čini skup svih UID
- Za svaki proces  $i$ :
  - $states_i = (u, send, status)$ 
    - $u$  – identifikator, inicijalno UID za  $i$
    - $send$  – identifikator ili null, inicijalno UID za  $i$
    - $status \in \{unknown, leader\}$ , inicijalno  $unknown$
  - $start_i = (\text{UID procesa } i, \text{UID procesa } i, unknown)$
  - $msgs_i$  – poslati vrijednost varijable  $send$  sljedećem procesu
  - $trans_i$  –
    - receive  $v$
    - $send := \text{null}$  (pobriši poruke na kanalima)
    - if  $v > u$  then  $send := v$
    - if  $v = u$  then  $status := leader$
    - if  $v < u$  then *do nothing*

# Složenost algoritma

- Vremenska
  - s obzirom da algoritam završava u slučaju kada čvor s najvećim UID ponovo primi vlastitu poruku, potrebno je  $n$  koraka da ta poruka stigne do vođe u prstenu s  $n$  čvorova
  - samo će proces koji je vođa znati da je algoritam završen (primio je poruku identičnu vlastitom UID), stoga može poslati posebnu poruku (*halt*) s obavijesti da je vođa izabran – potrebno je  $2n$  koraka za pronađak vođe i slanje poruka zaustavljanja
- Komunikacijska
  - u mreži se generira  $O(n^2)$  poruka - pri svakom koraku svaki čvor potencijalno generira novu poruku ( $n^2 = n$  poruka po koraku  $\times n$  koraka do završetka algoritma)
  - Ako analiziramo max broj generiranih poruka uzimajući u obzir *null* poruke, onda je to za mrežu s padajućim UID u smjeru kazaljke na satu kada se za svaki korak generira  $n+(n-1)+(n-2)+\dots+1$ , što je ukupno  $n*(n+1)/2$  poruka, što je  $O(n^2)$

## Problem 2: Odabir vođe u usmjerenoj mreži

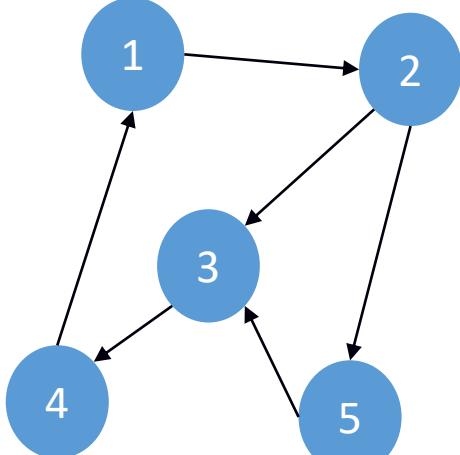
- Povezana usmjerena mreža, za svaki par čvorova postoji konačan  $distance(i, j)$
- Svaki čvor ima jedinstveni identifikator UID
- Izabratи vođу међу procesима у мрежи
- Samo 1 proces mijenja status u *leader*
- $diameter(G) = \max distance(v_i, v_j)$  за sve парове  $(v_i, v_j)$  из  $G$



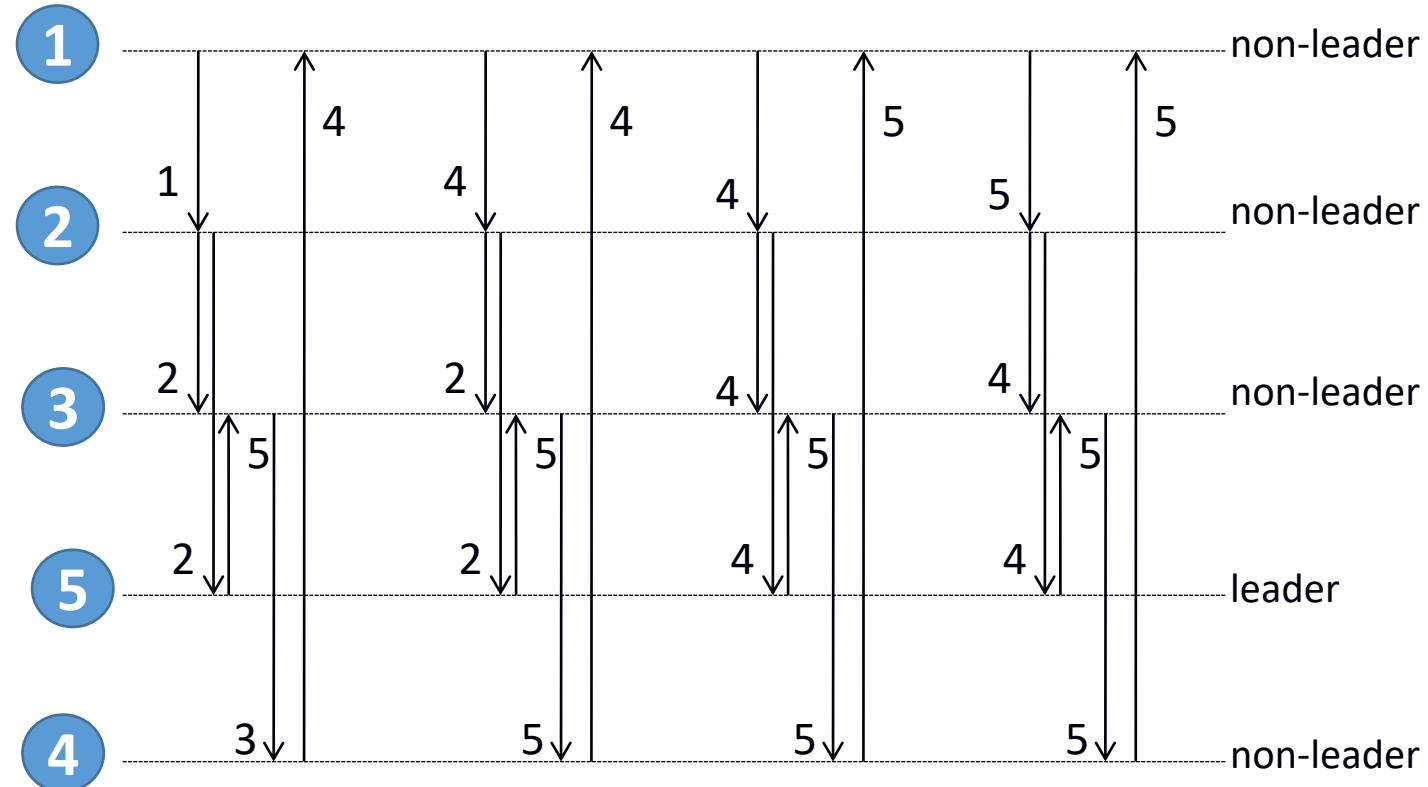
# Rješenje problema 2: Algoritam preplavljanja

- *Simple Flooding Algorithm (SFA)*
- Prepostavke
  - Svaki proces ima UID iz skupa prirodnih brojeva
  - Svaki proces zna  $diameter(G)$  – (**pažnja: ovo je globalno znanje!**)
- Skica algoritma
  - Svaki proces bilježi max primljeni UID (inicijalno je to vlastiti UID). U svakom koraku proces šalje tu maksimalnu vrijednost na izlaznim kanalima svim susjedima. Nakon  $diameter(G)$  koraka ako je maksimalna vrijednost jednaka vlastitom UID, proces se proglašava vođom, a u suprotnom nije vođa.

# Primjer algoritma za odabir vođe u usmjerenoj mreži



diameter = 4 jer je  
distance (5,2) = 4



# Formalni model algoritma preplavljanja

- $states_i = (u, max\text{-}uid, status, rounds)$   
 $u$  – UID, inicialno UID za  $i$   
 $max\text{-}uid$  – UID, inicialno UID za  $i$   
 $status \in \{unknown, leader, non-leader\}$ , inicialno *unknown*  
 $rounds$  – cijeli broj, inicialno 0
- $msgs_i$  – if  $rounds < diameter$  then  
send  $max\text{-}uid$  to all  $j \in out\text{-}nbrs$
- $trans_i$  –  $rounds := rounds + 1$   
receive set of UIDs  $U$  from neighbors  
 $max\text{-}uid := \max(\{max\text{-}uid\} \cup u)$   
if  $rounds = diameter$  then  
if  $max\text{-}uid = u$  then  $status := leader$   
else  $status := non-leader$

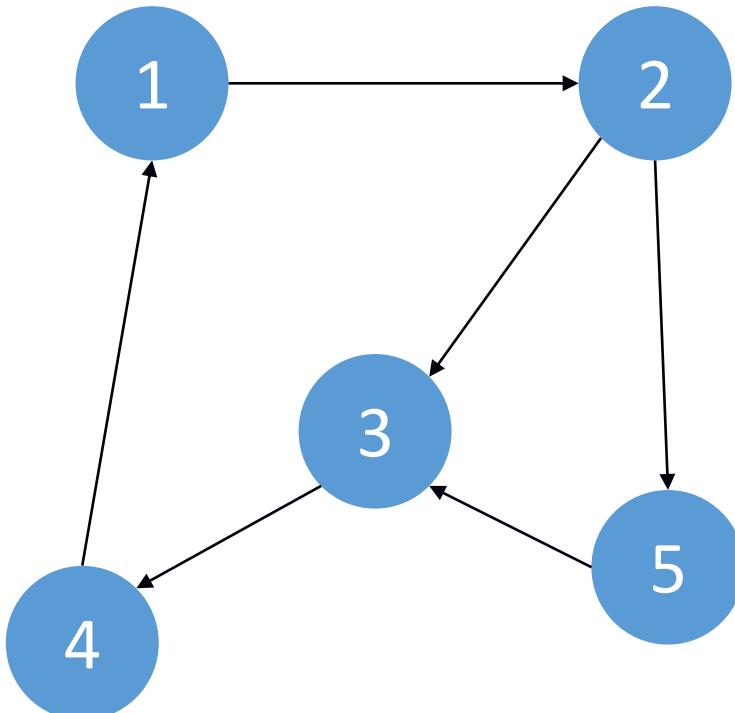
# Složenost

- Vremenska
  - Određena vrijednošću  $diameter(G)$
- Komunikacijska
  - broj poruka = diameter  $\cdot |E|$ , gdje je  $|E|$  broj usmjerenih grana grafa
  - poruka se šalje na svaku granu za svaki korak algoritma
  - jednostavna optimizacija koja smanjuje broj poruka – proces šalje *max-uid* susjedima samo ako se vrijednost *max-uid* promijeni

# Asinkroni model raspodijeljenog sustava

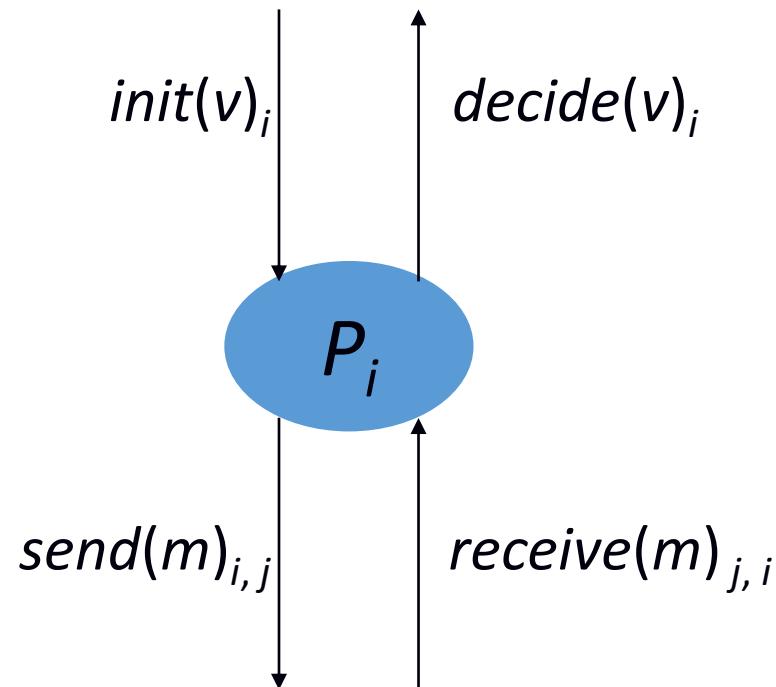
# Asinkroni model

- ♦ usmjereni graf  $G = (V, E)$
- ♦  $v_i \in V$ , čvor modelira proces
- ♦  $e_j \in E$ , grana modelira kanal
- ♦  $out-nbrs_i$  – izlazni susjedi
- ♦  $in-nbrs_i$  – ulazni susjedi
- ♦ asinkronost izvođenja procesa i komunikacije (razlika u odnosu na sinkroni model)
- ♦ svaki proces i svaki kanal se modeliraju I/O automatom



# Ulazno/izlazni (I/O) automat

- formalni model za asinkrone sustave
- I/O automat modelira komponentu raspodijeljenog sustava koja je u interakciji s ostalim komponentama
- prijelazi su vezani uz **događaje**
- događaji mogu biti *ulazni*, *izlazni* ili *unutarnji*



primjer procesa u asinkronom raspodijeljenom sustavu

# Formalna definicija I/O automata

I/O automat  $A$  se sastoji od sljedećih komponenti:

- $\text{sig}(A)$  – signatura  
 $\text{sig}(A) = \{ \text{in}(A), \text{out}(A), \text{int}(A) \}$  – opis ulaznih, izlaznih i unutarnjih događaja)
- $\text{states}(A)$  – skup stanja automata
- $\text{start}(A)$  – skup početnih stanja,  $\text{start}(A) \neq \emptyset$
- $\text{trans}(A)$  – funkcija prijelaza,  
npr.  $(s, \pi, s')$  –  $s$  i  $s'$  su stanja, a  $\pi$  je događaj
  - za svako stanje  $s$  i svaki ulazni događaj  $\pi$  postoji prijelaz  $(s, \pi, s') \in \text{trans}(A)$

# Izvođenje automata

- automat  $A$  se izvodi kao konačan ili beskonačan slijed stanja i događaja, npr.

$s_0, \pi_0, s_1, \pi_1, s_2, \pi_2, s_2, \dots \pi_k, s_k, \dots$

- $(s_k, \pi_k, s_{k+1}) \in \text{trans}(A)$ , za svaki  $k \geq 0$

# Primjer: automat kanala FIFO (1)



- $sig(C_{i,j}) = (send(m)_{i,j}, receive(m)_{i,j}, 0), m \in M$
- states:
  - *queue*, a FIFO queue
- trans:
  - $send(m)_{i,j}$  – dodaj  $m$  u *queue*
  - $receive(m)_{i,j}$  – preduvjet:  $m$  je 1. element iz *queue*, posljedica: briši  $m$  iz *queue*

# Primjer: automat kanala FIFO (2)

primjeri izvođenja – tj. slijed događaja (engl. *trace*)

- [null],  $send(1)_{i,j}$ , [1],  $receive(1)_{i,j}$ , [null],  $send(2)_{i,j}$ , [2],  $receive(2)_{i,j}$ , [null]
- [null],  $send(1)_{i,j}$ , [1],  $send(1)_{i,j}$ , [11],  $send(1)_{i,j}$ , [111]...

Za “*trace*” su važna sljedeća 2 svojstva:

- A **safety property** is often interpreted as saying that some particular “bad” thing never happens.
- A **liveness property** is often informally understood as saying that some particular “good” thing eventually happens.

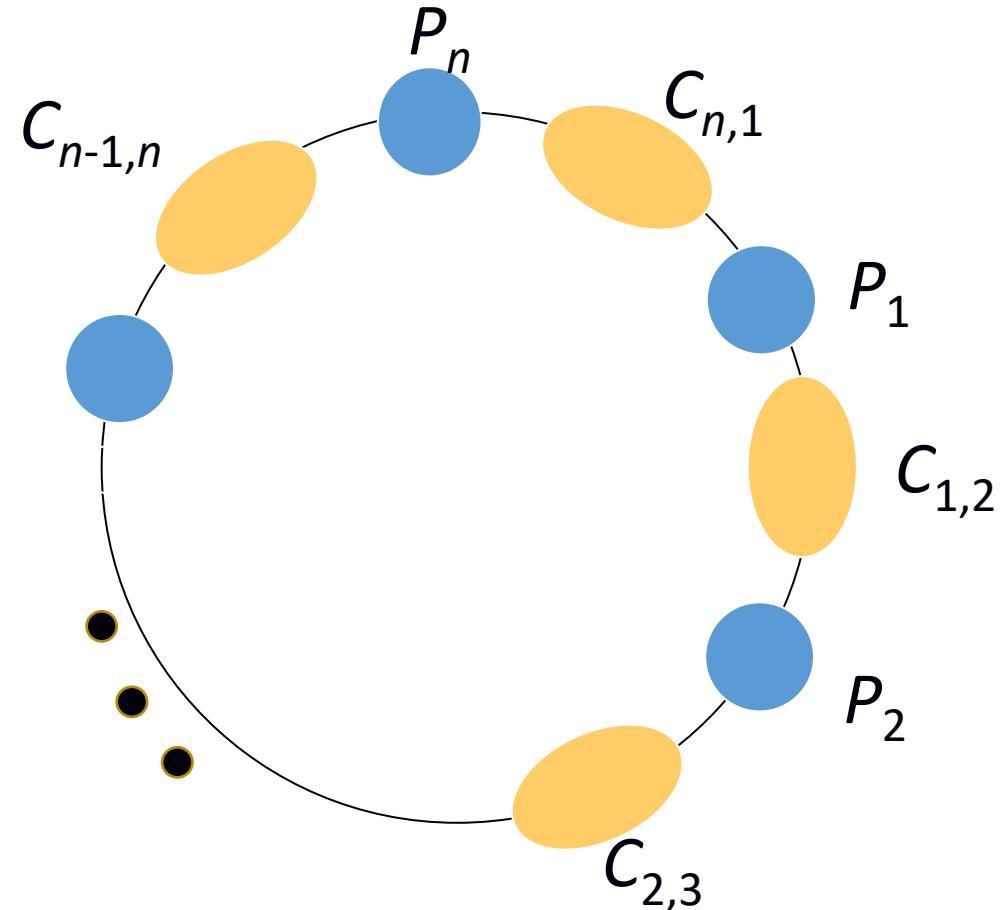
(prema N. Lynch)

# Primjeri asinkronog modela

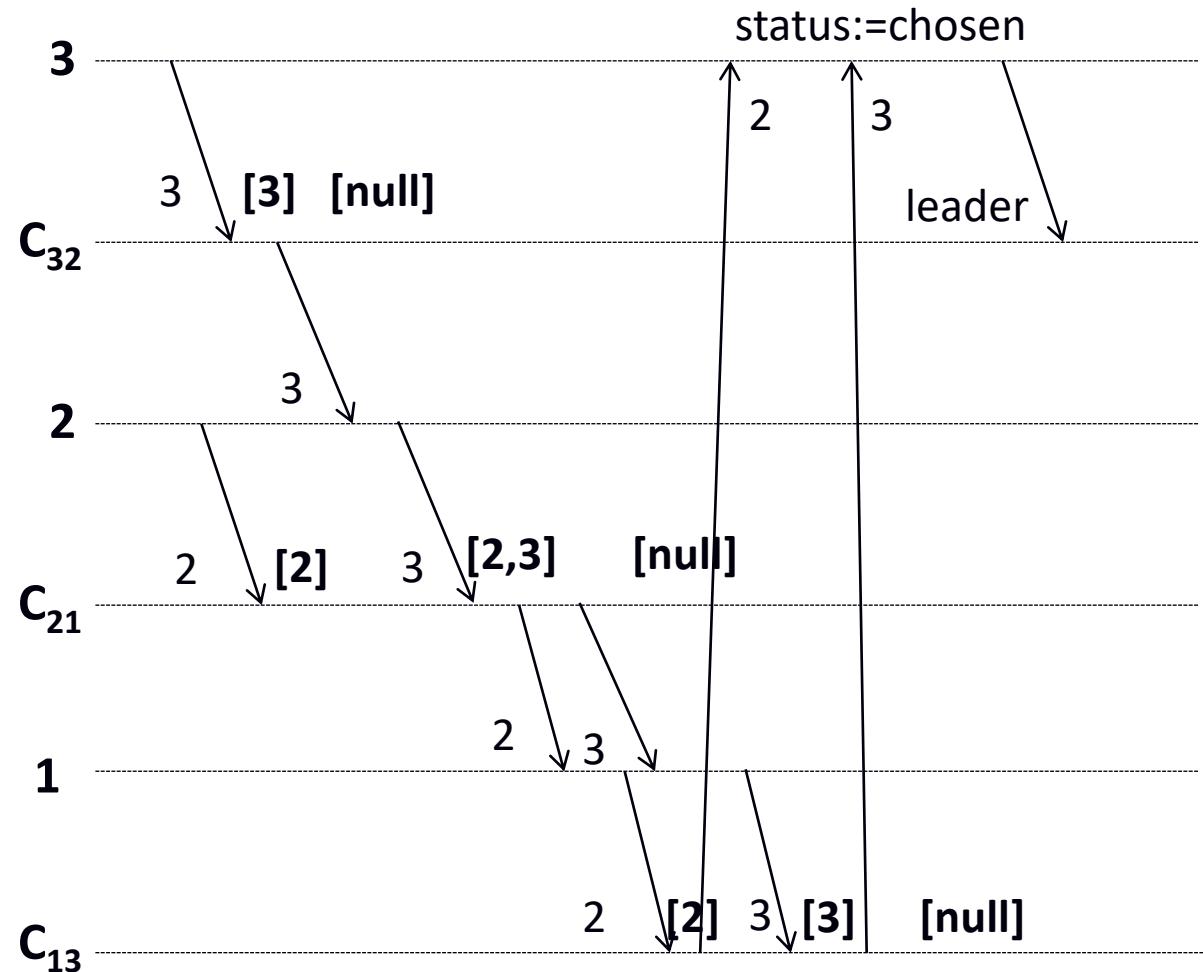
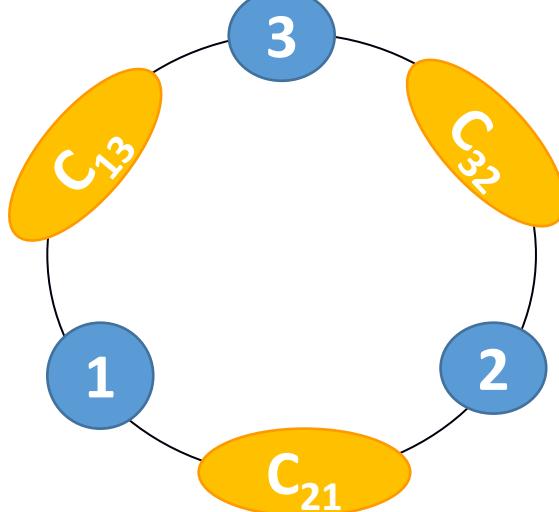
# Odabir vođe u asinkronoj mreži

## Definicija problema

- izabrati “vođu” među procesima u mreži
- samo 1 proces mijenja status u *leader*
- adaptacija sinkronog algoritma – svaki proces ima ulazni spremnik koji može primiti maksimalno  $n$  poruka (poruke se mogu gomilati zbog asinkronosti komunikacije)
- procesi: modelirani I/O automatom
- kanali: prepostavka je pouzdani FIFO



# Primjer asinkronog prstena i algoritma za odabir vođe



# Osnovni algoritam za asinkroni model

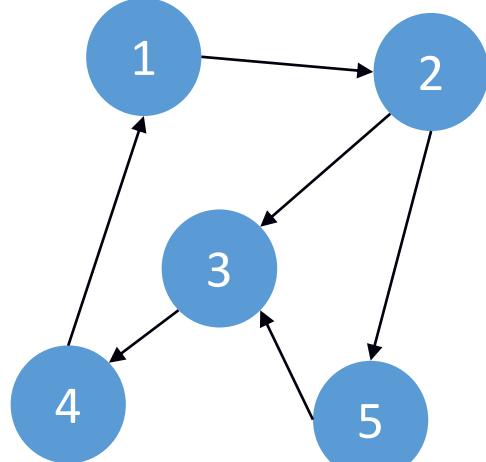
Definicija automata procesa  $P_i$

- input:  $receive(v)_{i-1,i}$ ,  $v$  je UID
- output:  $send(v)_{i,i+1}$ ;  $leader_i$
- $states_i$ :
  - $u$  – UID, inicijalno UID za  $i$
  - $send$  – FIFO queue UID-ova veličine  $n$ , inicijalno sadrži UID za  $i$
  - $status \in \{unknown, chosen, reported\}$ , inicijalno  $unknown$
- trans:
  - $send(v)_{i,i+1}$  – preduvjet:  $v$  je 1. element iz  $send$ , posljedica: briši  $v$  iz  $send$
  - $leader_i$  – preduvjet:  $status = chosen$ , posljedica:  $status := reported$
  - $receive(v)_{i-1,i}$ 
    - if  $v > u$ : add  $v$  to  $send$
    - if  $v = u$ : then  $status := chosen$
    - if  $v < u$ : do nothing

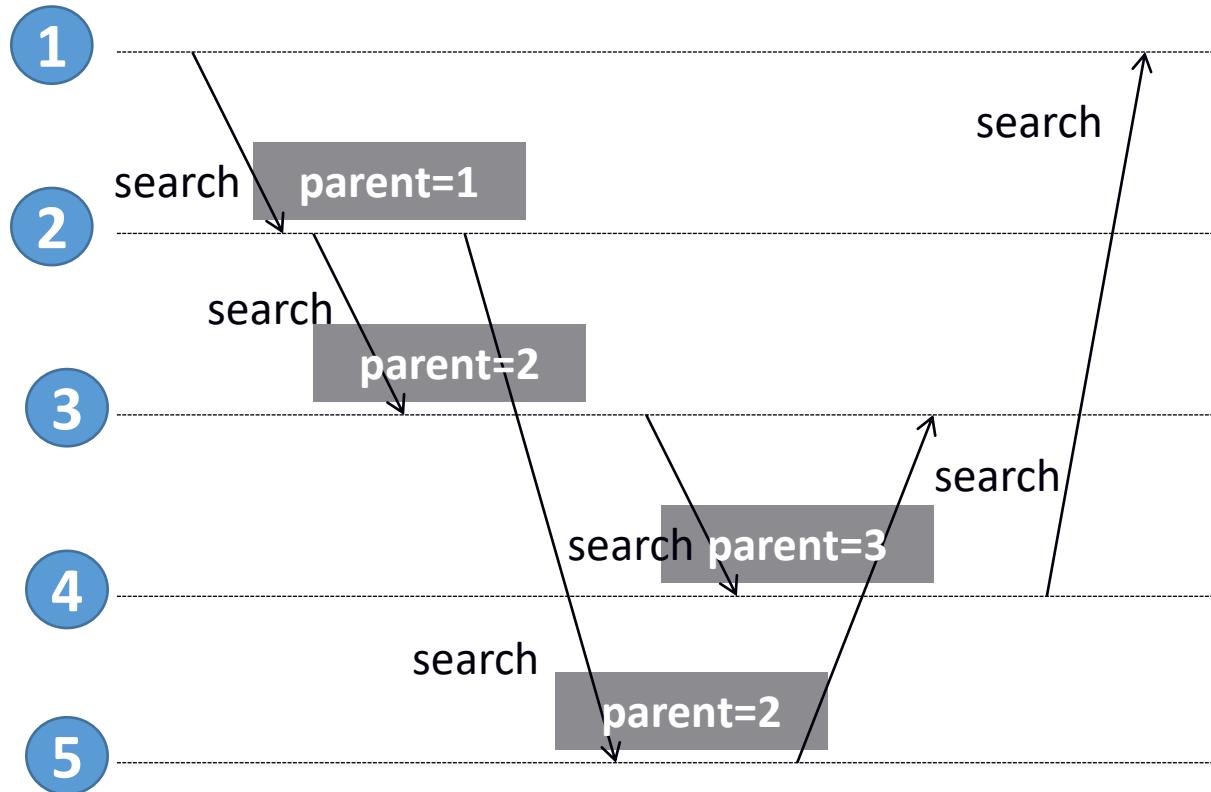
# Kreiranje stabla u asinkronoj mreži

- kreiranje stabla s definiranim korijenskim čvorom  $i_0$
- mreža je modelirana grafom  $G(V, E)$  koji je usmjeren i povezan
- procesi ne znaju dijametar mreže
- cilj: svaki proces treba odrediti prethodnika (*parent*)
- Skica algoritma *AsynchSpanningTree*
  - Inicijalno je  $i_0$  označen.  $i_0$  šalje *search* svim izlaznim susjedima. Kada proces primi *search* taj proces postaje označen, odabire jedan od susjeda od kojih je primio poruku za *parent* i šalje *search* svim svojim susjedima.

# Primjer algoritma *AsynchSpanningTree*



$$i_0 = 1$$



# AsynchSpanningTree

Definicija automata procesa  $P_i$

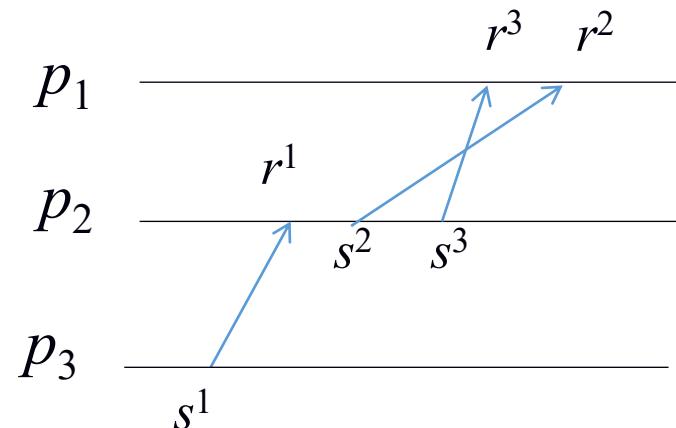
- input:  $receive("search")_{j,i}$ ,  $j \in nbrs$
- output:  $send("search")_{i,j}$ ,  $j \in nbrs$ ;  $parent(j)_i$ ,  $j \in nbrs$
- states:
  - $parent \in nbrs \cup \{null\}$ , inicijalno  $null$
  - $reported$  – boolean, inicijalno  $false$
  - za svaki  $j \in nbrs$  postoji  
 $send(j) \in \{search, null\}$ , inicijalno  $search$  ako je  $i = i_0$  inače  $null$
- trans:
  - $send("search")_{i,j}$  – preduvjet:  $send(j) = search$ , posljedica:  $send(j) := null$
  - $parent(j)_i$  – preduvjet:  $parent = j$ ,  $chosen = false$ , posljedica:  $reported := true$
  - $receive("search")_{j,i}$ 
    - if  $i \neq i_0$  and  $parent = null$   
 $parent := j$
    - for all  $k \in nbrs \setminus \{j\}$   
 $send(k) := search$

# Literatura

- A. D. Kshemkalyani, M. Singhal: *Distributed Computing: Principles, Algorithms, and Systems*, *Cambridge University Press*, 2008.  
poglavlja 2.1-2.4
- N. Lynch: *Distributed Algorithms*, *Morgan Kaufmann Publishers Inc.* 1996.
  - poglavlje 2: Modelling I: Synchronous Network Model
  - poglavlje 3: Leader Election in a Synchronous Ring (osnovni algoritam do 3.4)
  - poglavlje 8: Modelling II: Asynchronous System Model
  - poglavlje 15.1: Leader Election in a Ring

# Pitanja za učenje i ponavljanje

1. Za koje je svojstvo raspodijeljenih sustava značajna komunikacijska složenost algoritama? Zašto?
  - a) replikacijska transparentnost
  - b) skalabilnost
  - c) otvorenost
2. Objasnite model komunikacijskog kanala koji se temelji na uzročnoj slijednosti. Vrijedi li za primjer na slici CO ili non-CO i zašto?
3. Proučite formalnu definiciju algoritma *AsynchSpanningTree* na slajdu 60.





SVEUČILIŠTE U ZAGREBU



**Diplomski studij**  
**Računarstvo**  
Znanost o mrežama  
Programsko inženjerstvo i  
informacijski sustavi  
Računalno inženjerstvo  
**Ostali (slobodni izborni  
predmet)**

# Raspodijeljeni sustavi

**5. Procesi i komunikacija:**  
komunikacija porukama, model objavi-  
preplati, dijeljeni podatkovni prostor

Ak. god. 2022./2023.

# Creative Commons



- slobodno smijete:

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
  - **prerađivati** djelo



- pod sljedećim uvjetima:

- **imenovanje**: morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
  - **nekomercijalno**: ovo djelo ne smijete koristiti u komercijalne svrhe.
  - **dijeli pod istim uvjetima**: ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.



*U slučaju daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.*

*Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.*

*Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.*

*Tekst licence preuzet je s <http://creativecommons.org/>*

# „Neizravna” komunikacija

- engl. *indirect communication*
- komunikacija među procesima raspodijeljenog sustava **putem posrednika** bez direktne interakcije pošiljatelja i primatelja
- područja primjene
  - pokretne mreže i okoline
  - tokovi podataka (npr. financijski sustavi)
  - aplikacije u području Interneta stvari (senzori kontinuirano generiraju podatke)
- osigurava **prostornu i vremensku neovisnost procesa** (engl. *space uncoupling and time uncoupling*)

# Sadržaj predavanja

- Komunikacija porukama
- Model objavi-preplati
  - Primjeri programske opreme za komunikaciju porukama: JMS, AMQP, Kafka
- Dijeljeni podatkovni prostor

# Komunikacija porukama

engl. *message-queuing systems, Message-Oriented Middleware (MOM)*

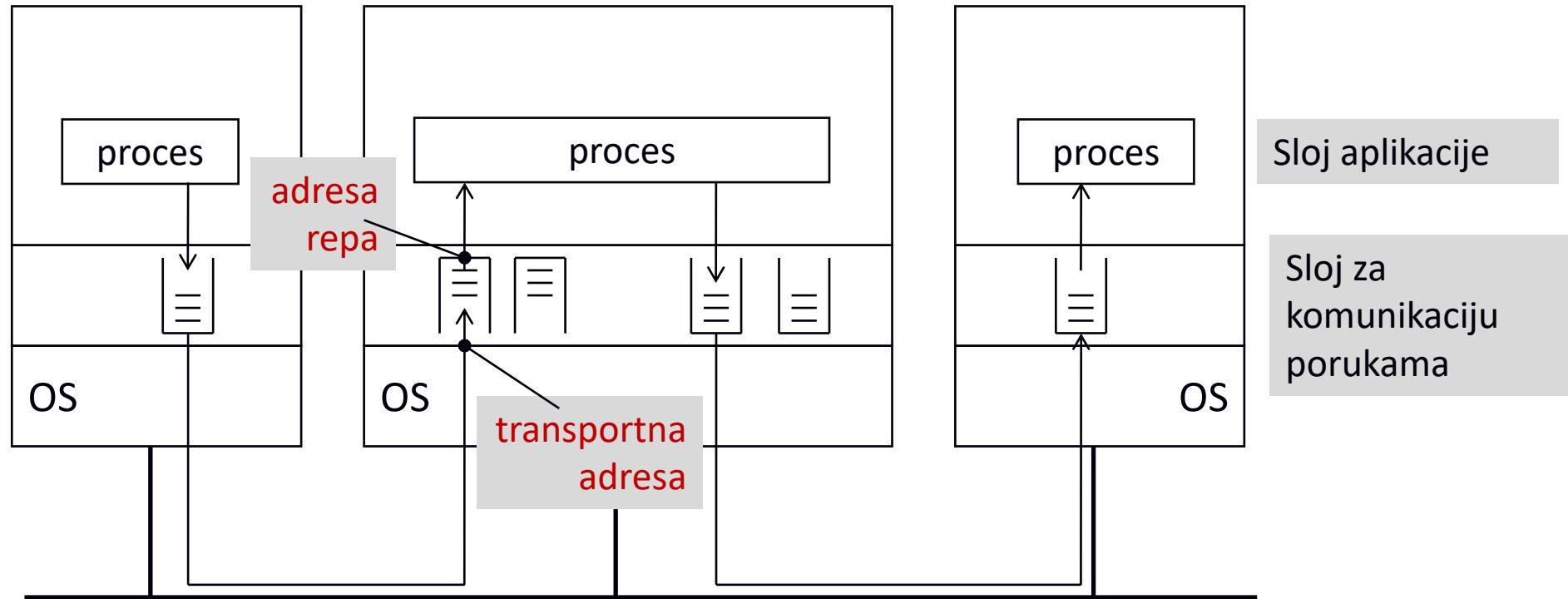
- Procesi/objekti komuniciraju razmjenjujući poruke.
- U komunikaciji sudjeluju izvor (pošiljatelj poruke) i odredište (primatelj poruke).
- Izvor šalje poruku, poruka se pohranjuje u rep koji je pridijeljen odredištu.
- Odredište čita poruku iz repa.
- Poruke sadrže podatke, važna je adresa određnog repa.
- Adresiranje se izvodi najčešće na nivou sustava, svaki rep ima jedinstven identifikator u sustavu.

# Izvođenje komunikacije porukama



- **put** – dodaj poruku u rep
- **get** – pročitaj poruku iz repa, primatelj je blokiran ako je rep prazan
- **poll** – provjeri postoji li poruke u repu i pročitaj prvu poruku ako takva postoji, primatelj nije blokiran

# Arhitektura sustava za komunikaciju porukama



# Obilježja komunikacije porukama

- vremenska neovisnost
  - primatelji i pošiljatelji ne moraju istovremeno biti aktivni, poruka se spremi u rep
- pošiljatelj mora znati identifikator odredišta, tj. njegovog repa
- komunikacija je **perzistentna**
- asinkrona komunikacija
  - pošiljatelj šalje poruku i nastavlja obradu neovisno o odgovoru od strane primatelja
- pokretanje komunikacije na načelu *pull*
  - primatelj provjerava postoji li poruka u repu

# Komunikacija je moguća i na načelu *push*

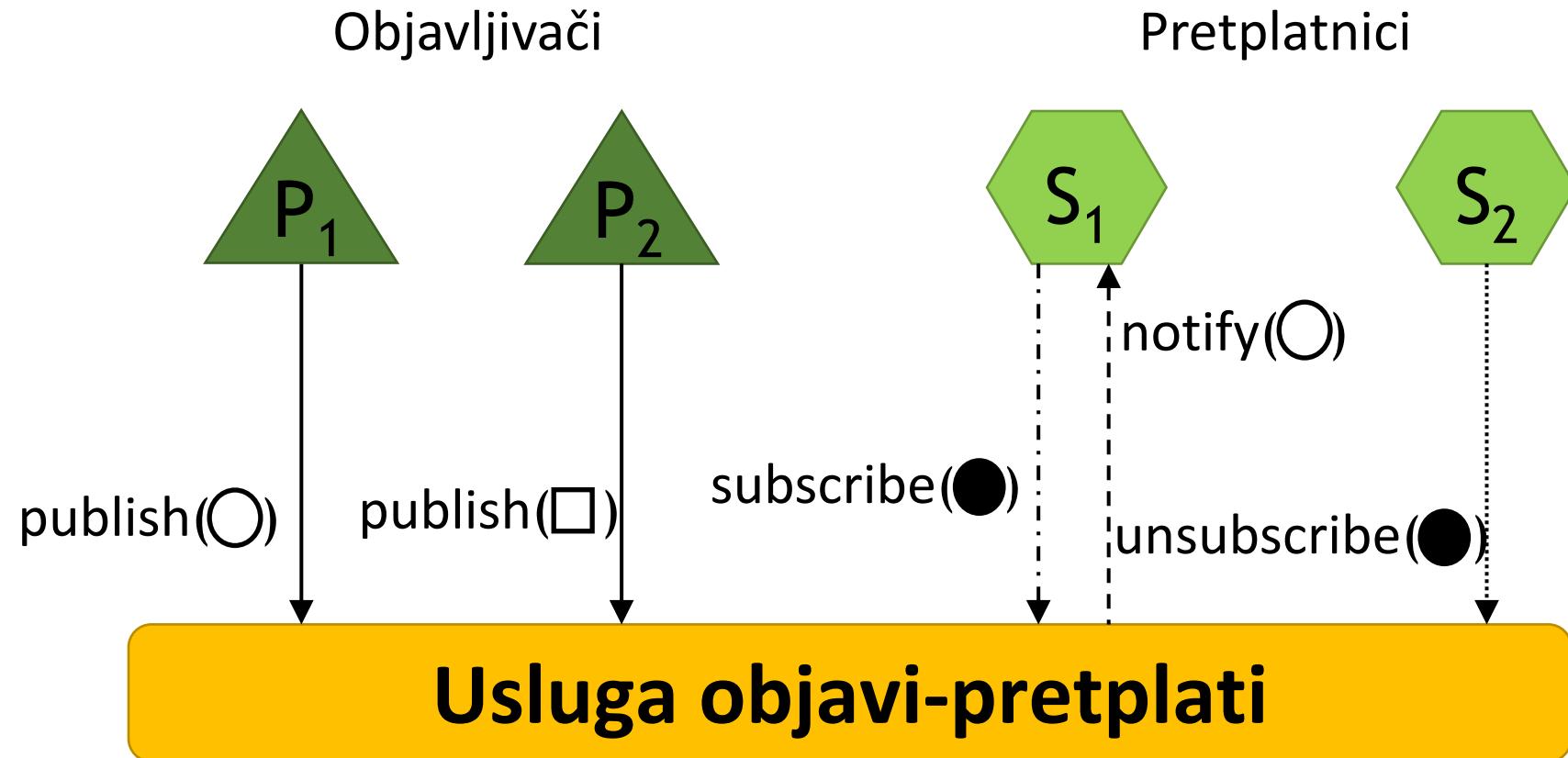


- **notify** – aktivna isporuka poruke iz repa primatelju po primitku poruke (na strani primateljskog procesa nužan je *listener thread*)

# Sadržaj predavanja

- Komunikacija porukama
- **Model objavi-preplati**
  - Primjeri programske opreme za komunikaciju porukama: JMS, AMQP, Kafka
- Dijeljeni podatkovni prostor

# Interakcija objavi-preplati



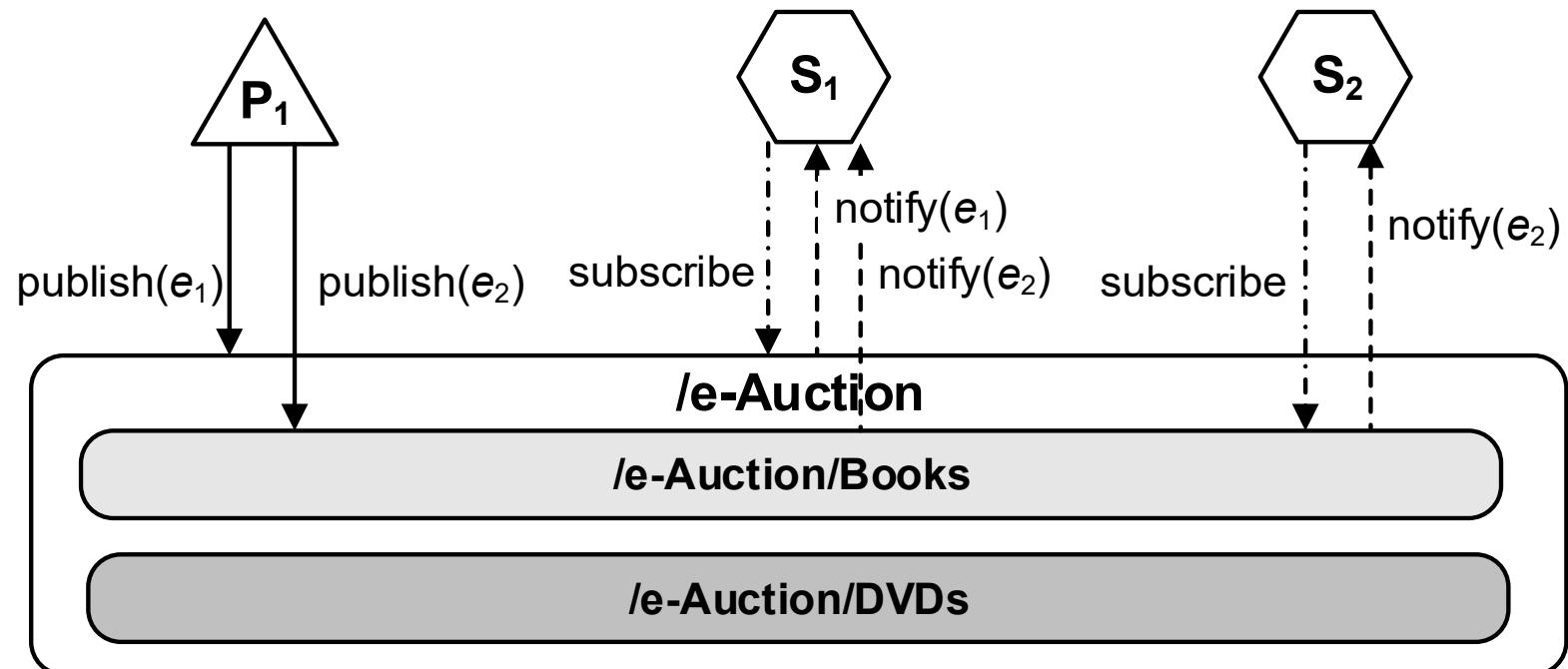
# Osnovni pojmovi

- objavljavači (*publishers*)
  - definiraju obavijesti (*notifications*)
- pretplatnici (*subscribers*)
  - pretplatama (*subscriptions*) i odjavama pretplata (*unsubscriptions*) izražavaju namjeru primanja određenog skupa obavijesti
- usluga objavi-pretplati:
  - sustav za obradu događaja (*event service – ES*)
  - obrađuje i pohranjuje primljene obavijesti/pretpiske/odjave pretplata
  - isporučuje obavijesti pretplatnicima prema njihovim aktivnim pretplatama
  - omogućuje persistenciju komunikaciju između objavljavača i pretplatnika

# Pretplate

- „kontinuirani upiti”
- pretplata na kanal/temu (engl. *topic-based subscription*)
  - kanal – logička veza između izvora i odredišta koja služi za tematsko grupiranje obavijesti (npr. vrijeme, sport, itd.)
  - hijerarhijski odnos kanala (npr. vrijeme u Europi, Hrvatskoj, Zagrebu)
- pretplata na sadržaj (engl. *content-based subscription*)
  - pretplata se definira ovisno o svojstvima i sadržaju obavijesti (skup atributa i vrijednosti)

# Pretplata na kanal



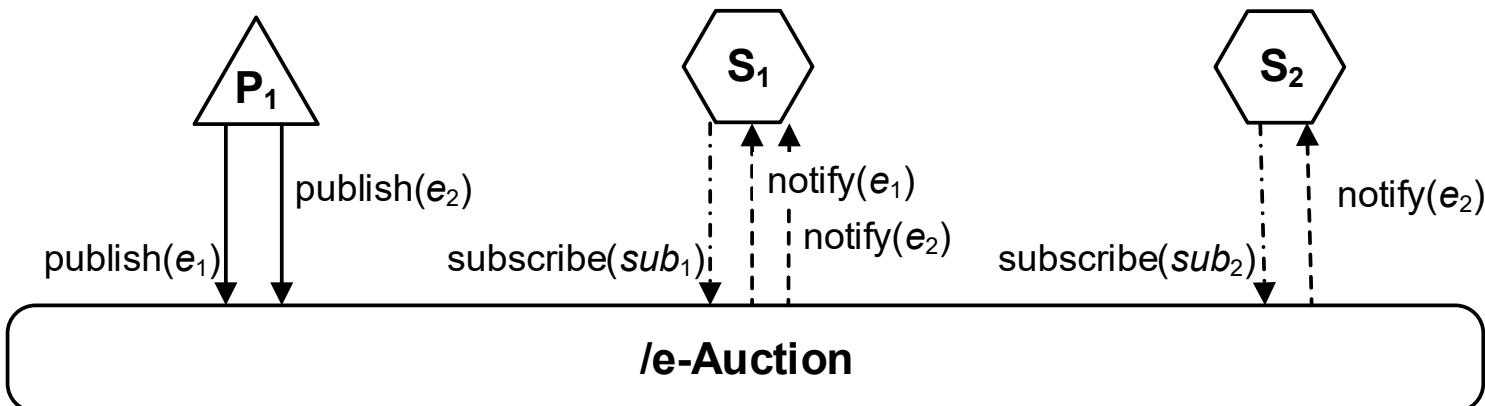
# Pretplata na sadržaj

```
e1 = (category = "books"
 & author = "D. Adams"
 & title = "The Hitchhiker's Guide through the Galaxy"
 & price = 9.99 EUR)
```

```
e2 = (category = "books"
 & author = "J.R.R. Tolkien"
 & title = "The Lord of the Rings"
 & price = 19.99 EUR)
```

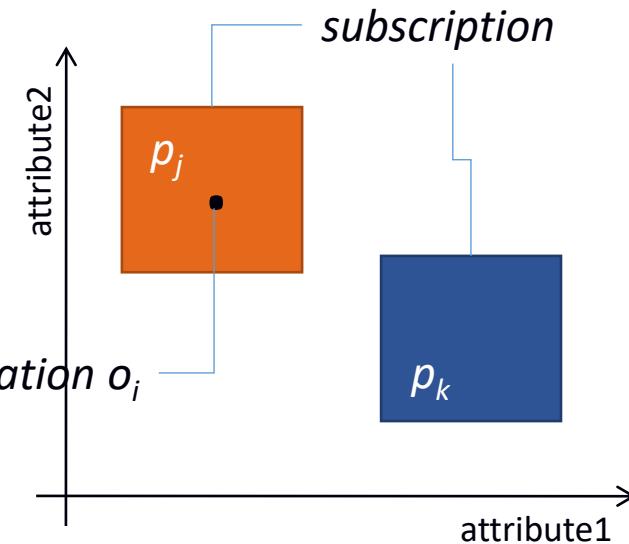
```
sub1 = (category == "books"
 & price < 20 EUR)
```

```
sub2 = (category == "books" &
 author == "J.R.R. Tolkien"
 & price < 20 EUR)
```



# Primjer obavijesti/preplate (strukturirani podaci)

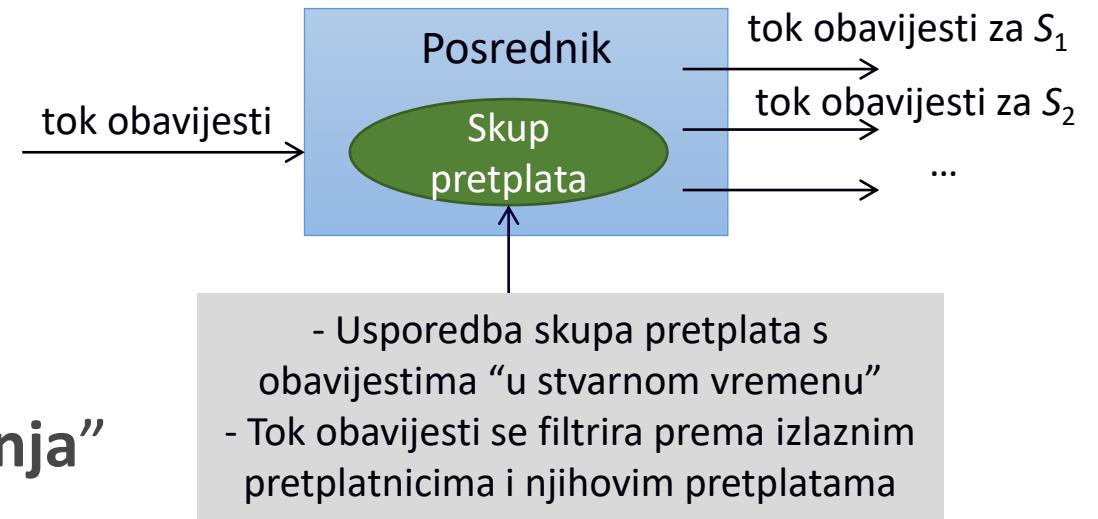
- **obavijest** je najčešće točka u višedimenzionalnom prostoru
  - npr. očitanje senzora, cijena dionice, oglas, vijest
  - objavljivači kontinuirano objavljaju nove obavijesti (često ograničene valjanosti)
- **preplata** je potprostor višedimenzionalnog prostora
  - definira se kao Booleova funkcija nad parom (obavijest, preplata)
  - za preplatu kažemo da **prekriva** obavijest kada obavijest zadovoljava uvjete preplate, tj.  $f(o_i, p_j) = T$



Poseban implementacijski izazov:  
učinkovita usporedba objave sa  
skupom preplate jer je u stvarnom  
vremenu potrebno odrediti  
podskup preplate koje prekrivaju  
obavijest kako bi se isporučila svim  
zainteresiranim preplatnicima

# Usporedba obavijesti sa skupom pretplatama

- Sustav objavi-preplati održava skup pretplata koje se uspoređuju s novoobjavljenom obavijesti



- Usporedba ispituje svojstvo **“prekrivanja”** obavijesti pretplatom

- Pretplata “prekriva” obavijest kada obavijest zadovoljava sve uvjete definirane pretplatom
- Pretplata  $[a < 10, b \leq 20]$  prekriva obavijest  $[a=5, b=20]$ , ali ne prekriva obavijest  $[a=5, b=25]$
- Pretplata  $[\text{sveučilište==Zagreb, fakultet==FER}]$  prekriva obavijest  $[\text{sveučilište=Zagreb, fakultet=FER, vijest=Proslavljen dan FER-a!}]$

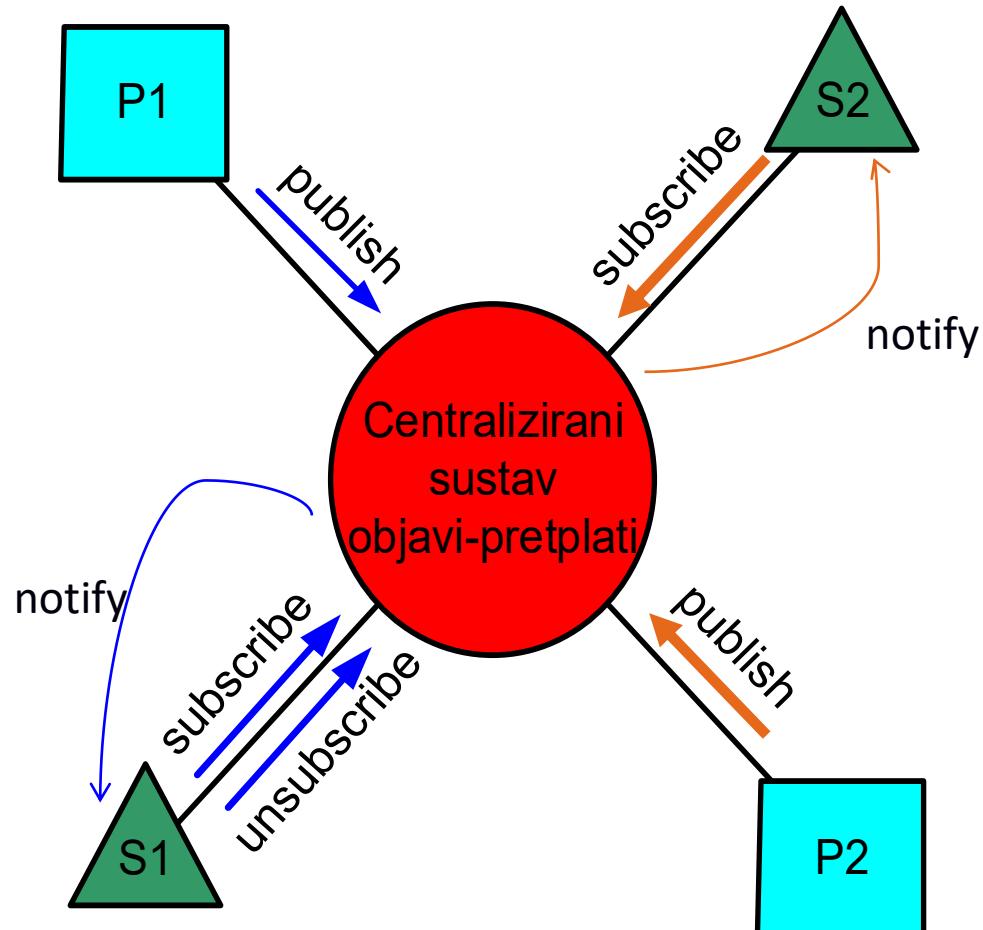
# Kvaliteta usluge za komunikaciju porukama

- Vezana uz garanciju isporuke poruke
  - najviše jednom (*at-most-once*) – ne postoje mehanizmi koji osiguravaju isporuku poruke u slučaju ispada
  - barem jednom (*at-least-once*) – postoje mehanizmi koji će u slučaju ispada ponoviti operaciju, moguće je da će primatelj primiti poruku više puta
  - sigurno jednom (*exactly once*) – primatelj će primiti poruku samo jednom
- Poruke mogu biti perzistentne (imaju vremenski definiran period valjanosti) i neperzistentne poruke („vrijede” u trenutku u kome su definirane)

# Arhitektura usluge objavi-pretplati

- Centralizirana
  - svi objavljavači i pretplatnici razmjenjuju obavijesti i definiraju preplate preko jednog poslužitelja posrednika
  - poslužitelj pohranjuje sve preplate i prosljeđuje obavijesti
- Raspodijeljena
  - skup poslužitelja, svaki je poslužitelj zadužen za objavljavače i pretplatnike u svojoj domeni
  - algoritmi za usmjerenje informacija o preplatama i usmjerenje obavijesti

# Centralizirana arhitektura

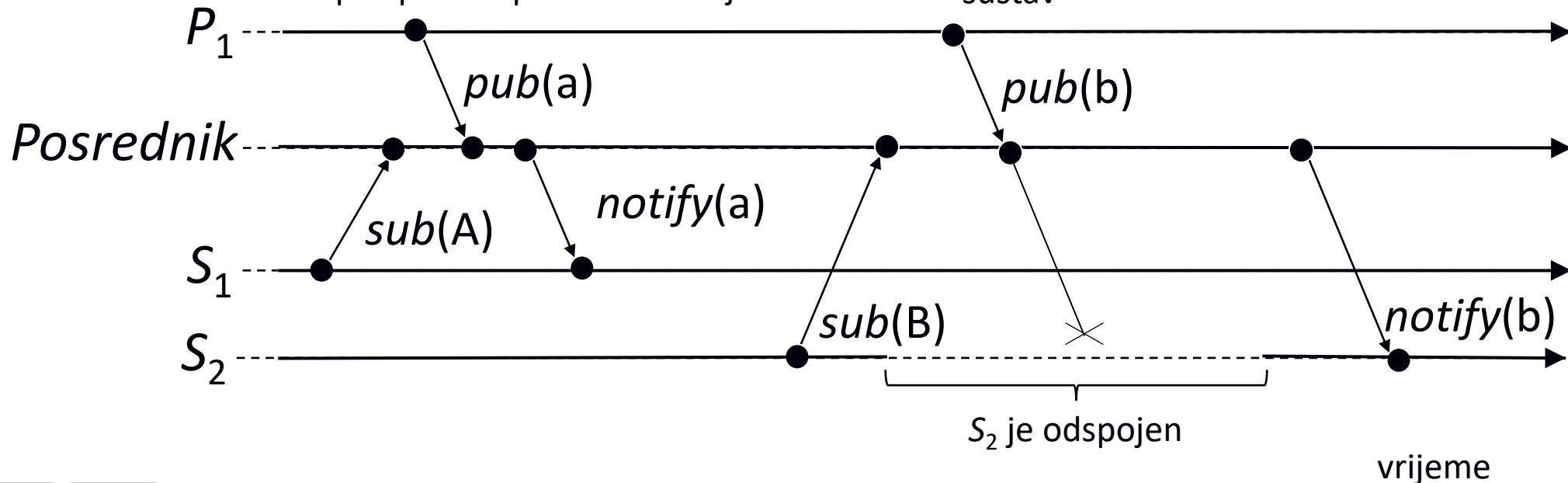


# Primjer raspodijeljenog izvođenja

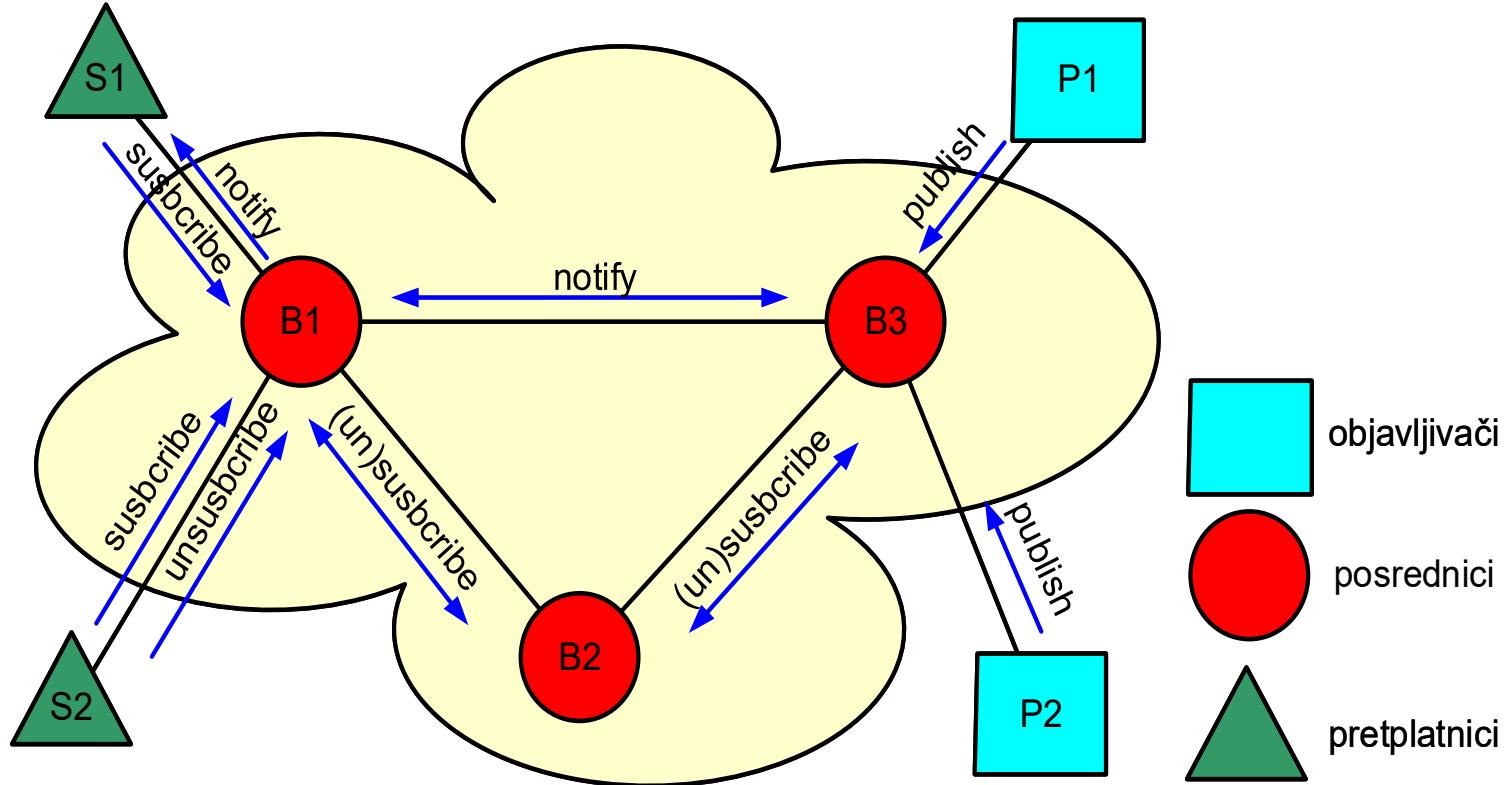
(asinkroni model, centralizirano)

primjer tipičnog slijeda događaja,  
 $sub \rightarrow pub \rightarrow notify$   
kada pretplata A prekriva obavijest a

primjer isporuke **perzistentne obavijesti**,  
B je prezistentna pretplata (*durable subscription*),  
pa posrednik čuva *matching* obavijesti koje ne može  
isporučiti i isporučuje ih kada se  $S_2$  ponovno spoji u  
sustav



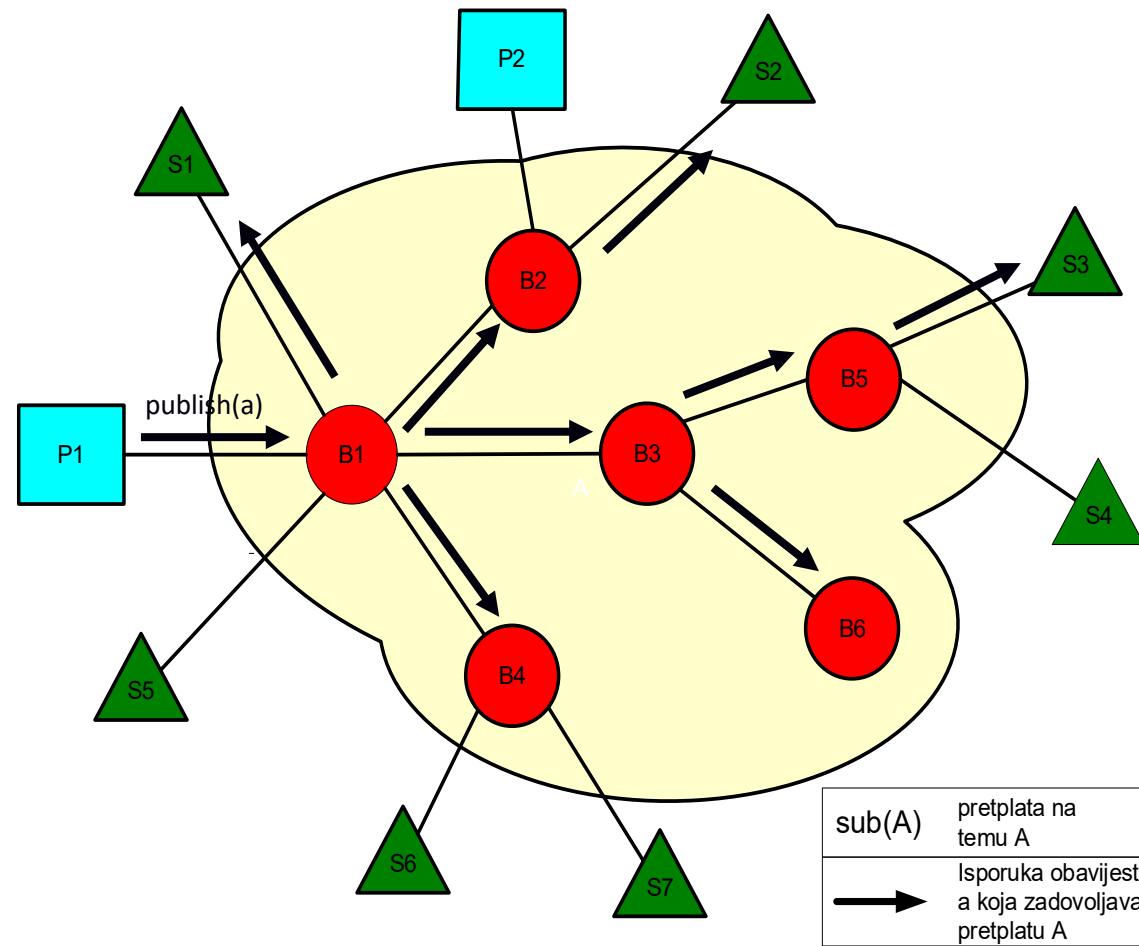
# Raspodijeljena arhitektura



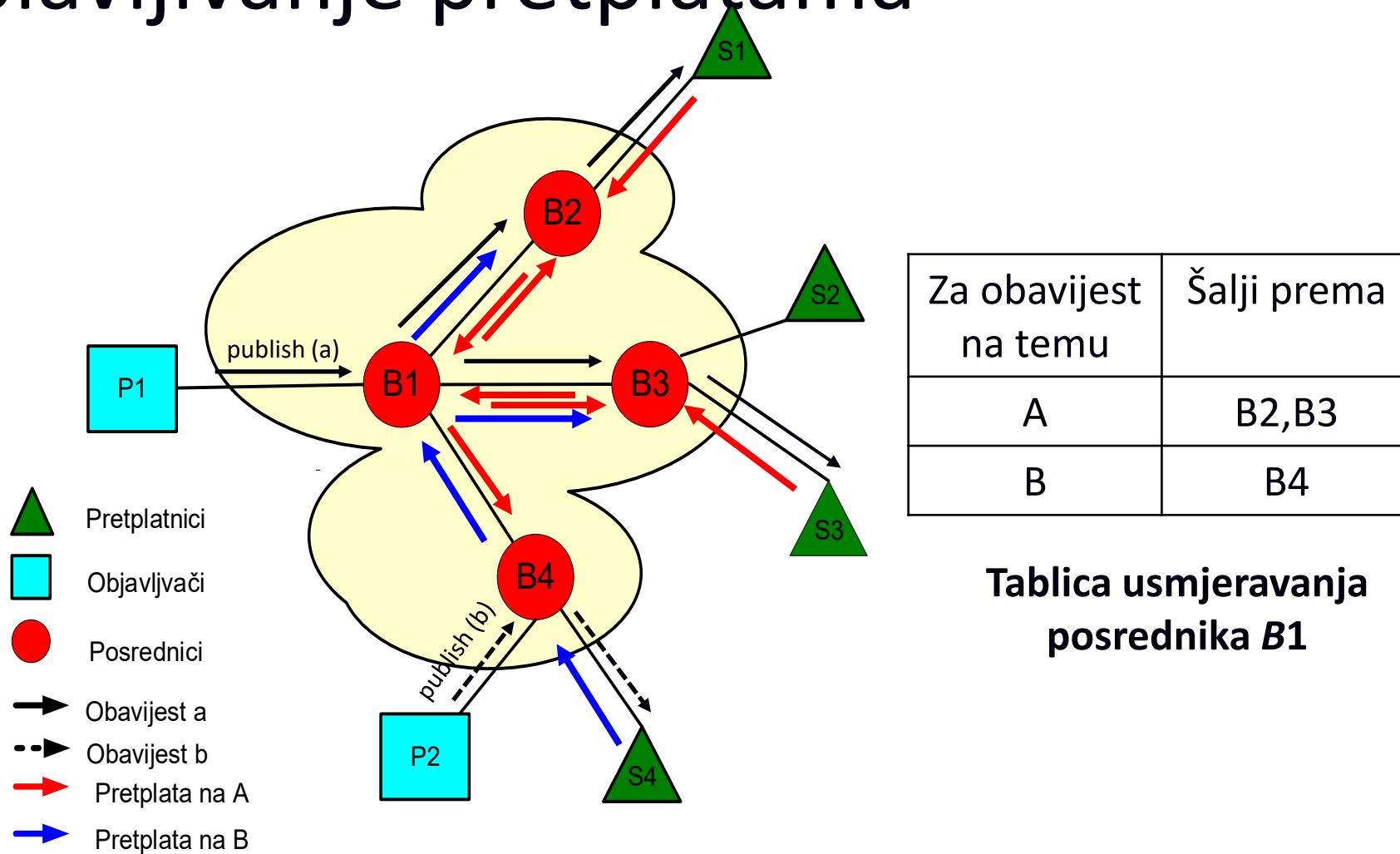
# Osnovna načela usmjeravanja

- preplavljanje
  - svaka primljena poruka (obavijest, pretplata ili odjava pretplate) prosljeđuje se svim susjedima osim onome od koga je poruka primljena
  - posrednik posjeduje tablicu usmjeravanja koja sadrži informacije o svim susjednim posrednicima i lokalnim pretplatnicima
- filtriranje poruka
  - filtriranje poruka se izvodi usporedbom obavijesti s aktivnim pretplatama koje definiraju svojstva obavijesti za koje je pretplatnik zainteresiran
  - osnovni cilj je isporuka samo onih obavijesti koje pretplatnika zanimaju
  - omogućuje i smanjenje prometa u mreži posrednika zbog sprječavanja širenja obavijesti "nezainteresiranim" posrednicima

# Preplavljanje obavijestima



# Preplavljanje pretplatama



# Obilježja modela objavi-preplati

- **vremenska neovisnost**
  - objavljavač i pretplatnici ne moraju istovremeno biti aktivni, posrednik pohranjuje poruku
- objavljavač ne mora znati identifikator pretplatnika (**anonimnost**), o tome se brine posrednik – **prostorna neovisnost**
- komunikacija je **perzistentna**
- **asinkrona komunikacija**
  - objavljavač šalje poruku i nastavlja obradu neovisno o odgovoru od strane odredišta – **vremenska neovisnost**
- pokretanje komunikacije na načelu **push**
  - objavljavač šalje poruku posredniku koji je prosljeđuje pretplatnicima bez prethodnog eksplicitnog zahtjeva

# Obilježja modela objavi-preplati (2)

- personalizacija primljenog sadržaja
  - filtriranje objavljenih poruka prema pretplatama
- proširivost sustava
  - dodavanje novog objavljivača ili pretplatnika ne utječe na ostale strane u komunikaciji
- skalabilnost
  - raspodijeljena arhitektura

# Sadržaj predavanja

- Komunikacija porukama
- Model objavi-preplati
  - Primjeri programske opreme za komunikaciju porukama: JMS, AMQP, Kafka
- Dijeljeni podatkovni prostor

# JMS

## Java Message Service

JMS 2.0, Java Community Process, 21.05.2013.

<https://java.net/projects/jms-spec/pages/JMS20FinalRelease>

Specifikacija otvorenog protokola za komunikaciju porukama i komunikaciju na načelu objavi-preplati.

JMS API definira skup sučelja i pripadajuću semantiku koja omogućuje programima pisanim u Javi komunikaciju razmjenom poruka i na načelu objavi-preplati.

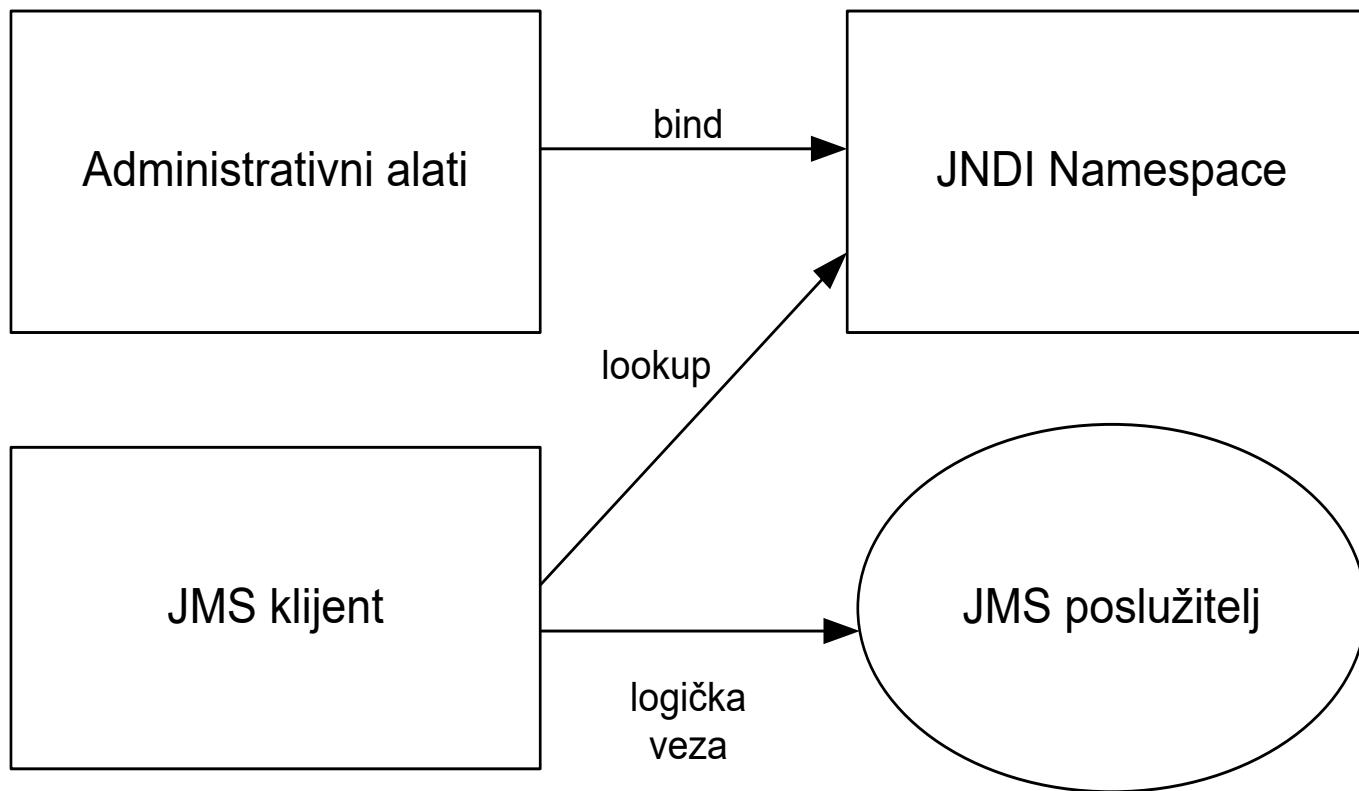
Popularne implementacije: Apache ActiveMQ, IBM WebSphereMQ, HornetQ, OpenJMS

# Arhitektura JMS-a (1)

- JMS poslužitelj
  - sustav za razmjenu poruka koji implementira JMS sučelja i nudi administrativne i kontrolne usluge
- Klijent
  - bilo koji objekt, proces ili aplikacija koja stvara ili konzumira poruke
- Poruka (*message*)
  - objekt koji se sastoji od zaglavljiva koje prenosi identifikacijske i adresne informacije i tijela koje prenosi podatke
- Odredište (*destination*)
  - objekt koji sadrži informacije o odredištu poruke

# Arhitektura JMS-a (2)

JNDI  
(Java Naming and Directory Interface)

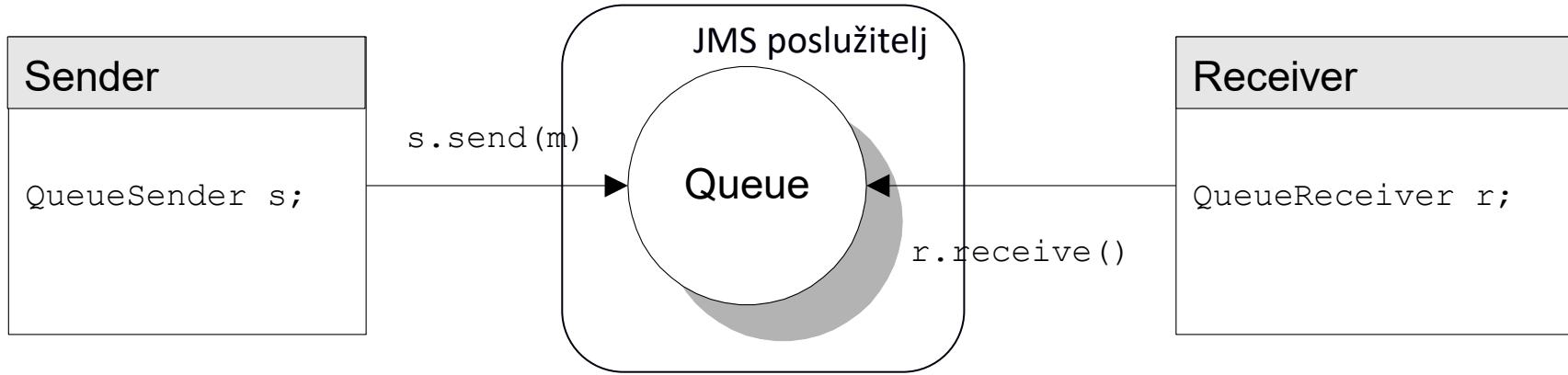


# Modeli JMS-a

JMS implementira sljedeće modele za komunikaciju porukama i obavijestima

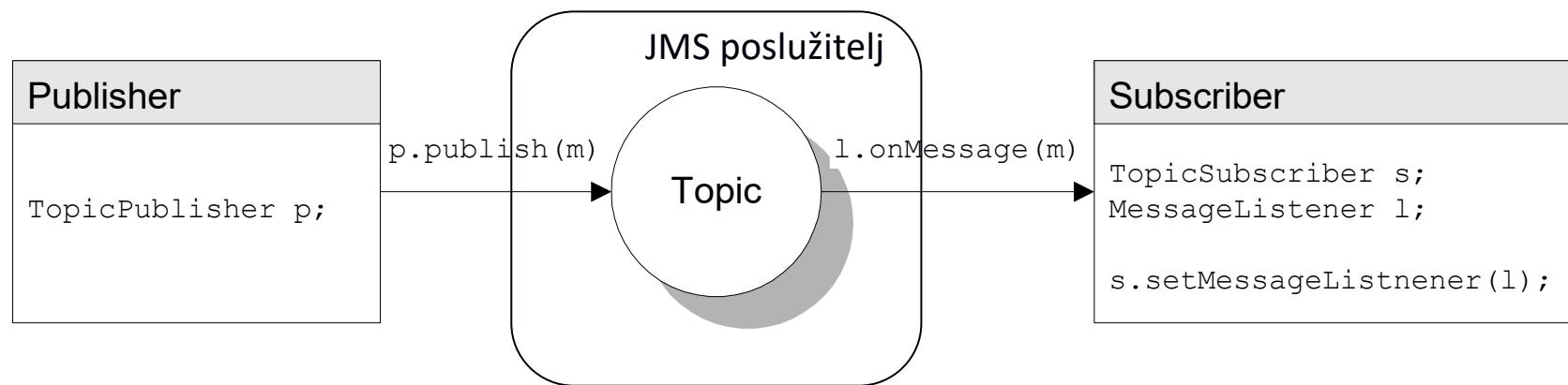
- *Point-to-point*
  - komunikacija porukama, jedna poruka za jedno odredište
- *Publish/subscribe*
  - objavi-preplati, jedna poruka za skup zainteresiranih pretplatnika

# Point-to-point



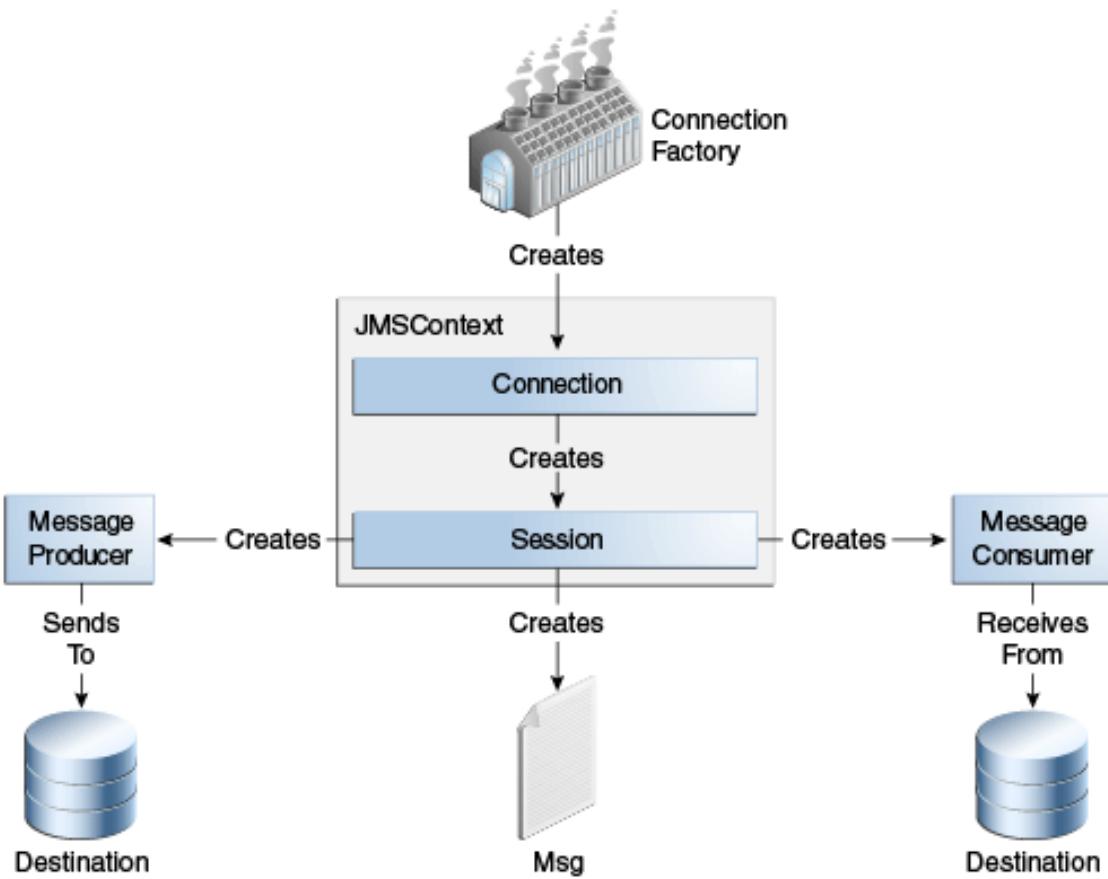
1. Klijent s koji šalje poruku m poziva `s.send(m)`. Poruka se sprema u repu.
2. Klijent koji prihvata poruku mora provjeriti postoji li poruka u repu . Poziva `r.receive()`.
3. Poruka se briše iz repa i šalje klijentu.

# Publish/subscribe



1. Tijekom inicijalizacije pretplatnik registrira instancu klase koja implementira sučelje `MessageListener` pozivajući `s.setMessageListener(l)`. Topic pamti sve preplate.
2. Izvor objavljuje poruku `m` sa `p.publish(m)`.
3. Topic isporučuje poruku pretplatniku pozivajući `l.onMessage(m)`.

# Programski model JMS API-ja



Izvor: **The Java EE 7 Tutorial**  
Poglavlje 45: Java Message Service Concepts

# Sučelja JMS-a (1)

|                   |                        |                        |
|-------------------|------------------------|------------------------|
| Nad-sučelje       | Point-to-point         | Publish/subscribe      |
| Destination       | Queue                  | Topic                  |
| ConnectionFactory | QueueConnectionFactory | TopicConnectionFactory |
| Connection        | QueueConnection        | TopicConnection        |

- ◆ **Destination**
  - administrirani objekt
  - predstavlja odredište - identitet ili adresu repa/teme.
- ◆ **ConnectionFactory**
  - administrirani objekt koji sadrži konfiguracijske parametre
  - klijenti ga koriste za stvaranje objekta *Connection*.
- ◆ **Connection**
  - predstavlja aktivnu konekciju prema JMS poslužitelju
  - klijenti ga koriste za stvaranje sjednice (*Session*).

# Sučelja JMS-a (2)

|                 |                |                   |
|-----------------|----------------|-------------------|
| Nad-sučelje     | Point-to-point | Publish/subscribe |
| Session         | QueueSession   | TopicSession      |
| MessageProducer | QueueSender    | TopicPublisher    |
| MessageConsumer | QueueReceiver  | TopicSubscriber   |

- ◆ **Session**
  - dretva u kojoj se primaju odnosno šalju poruke
  - klijenti koriste sesiju da stvore jedan ili više *MessageProducer* ili *MessageConsumer* objekata
- ◆ **MessageProducer**
  - objekt za slanje poruka odredištu
- ◆ **MessageConsumer**
  - objekt za primanje poruka koje su poslane odredištu

# Poruke JMS-a

- zaglavje
  - skup definiranih polja koja sadrže vrijednosti koje identificiraju i usmjeravaju poruku
- svojstva poruke
  - opcionalni parovi ime-vrijednost, a vrijednost može biti *boolean*, *byte*, *short*, *int*, *long*, *float*, *double* ili *String*
- tijelo poruke
  - *TextMessage* sadrži *java.lang.String*. (npr. za slanje XML dokumenata)
  - *StreamMessage* za niz Javinih primitiva.
  - *MapMessage* kada tijelo sadrži skup parova ime-vrijednost.
  - *ObjectMessage* sadrži Java objekt.
  - *ByteMessage* za tijelo koje sadrži niz neinterpretiranih *byte*-ova.

# Literatura: JMS

## The Java EE 7 Tutorial

- Chapter 45: Java Message Service Concepts

<https://docs.oracle.com/javaee/7/tutorial/jms-concepts.htm>

- What's New in JMS 2.0, Part One: Ease of Use

<http://www.oracle.com/technetwork/articles/java/jms20-1947669.html>

- What's New in JMS 2.0, Part Two—New Messaging Features

<http://www.oracle.com/technetwork/articles/java/jms2messaging-1954190.html>

- Enterprise Integration Patterns

<http://www.enterpriseintegrationpatterns.com/patterns/messaging/toc.htm>

!

# Primjer 1: JMS

## 1. Perform a JNDI lookup of the ConnectionFactory and Queue:

```
/* Create a JNDI API InitialContext object if none exists yet. */
Context jndiContext = null;
try {
 jndiContext = new InitialContext();
} catch (NamingException e) {
 System.out.println("Could not create JNDI API " + "context: " + e.toString());
 System.exit(1);
}

/* Look up connection factory and destination. If either does not exist, exit. */
QueueConnectionFactory connectionFactory = null;
Queue queue = null;
try {
 connectionFactory = (QueueConnectionFactory)
 jndiContext.lookup("jms/QueueConnectionFactory");
 queue = (Queue) jndiContext.lookup("queue");
}
catch (Exception e) {
 System.out.println("JNDI API lookup failed: " + e.toString());
 e.printStackTrace();
 System.exit(1);
}
```

# Queue Sender (2)

## 2. Create a Connection and a Session:

```
QueueConnection connection =
 connectionFactory.createQueueConnection();
QueueSession session = connection.createQueueSession(false,
 Session.AUTO_ACKNOWLEDGE);
```

## 3. Create a QueueSender and a TextMessage:

```
QueueSender sender = session.createSender(queue);
TextMessage message = session.createTextMessage();
```

# Queue Sender (3)

4. Send one or more messages to the queue:

```
for (int i = 0; i < NUM_MSGS; i++) {
 message.setText("This is message " + (i + 1));
 System.out.println("Sending message: " +
 message.getText());
 sender.send(message);
}
```

5. Send an empty control message to indicate the end of the message stream. Sending an empty message of no specified type is a convenient way to indicate to the consumer that the final message has arrived.

```
sender.send(session.createMessage());
```

# Queue Sender (4)

6. Close the connection in a finally block, automatically closing the session and QueueSender:

```
} finally {
 if (connection != null) {
 try {
 connection.close();
 } catch (JMSEException e) {}
 }
}
```

# Queue Receiver (1)

1. Performs a JNDI lookup of the ConnectionFactory and Queue.
2. Creates a Connection and a Session.
3. Creates a QueueReceiver:

```
QueueReceiver receiver = session.createReceiver(queue);
```

4. Starts the connection, causing message delivery to begin:

```
connection.start();
```

# Queue Receiver (2)

5. Receives the messages sent to the destination until the end-of-message-stream control message is received:

```
while (true) {
 Message m = receiver.receive();
 if (m != null) {
 if (m instanceof TextMessage) {
 message = (TextMessage) m;
 System.out.println("Reading message: " +
 message.getText());
 } else {
 break;
 }
 }
}
```

- Since the control message is not a TextMessage, the receiving program terminates the while loop and stops receiving messages after the control message arrives.
6. Closes the connection in a finally block, automatically closing the session and QueueReceiver.

# TopicPublisher

1. Perform a JNDI lookup of the TopicConnectionFactory and Topic.
2. Create a TopicConnection and a TopicSession.
3. Create a TopicPublisher and a TextMessage.
4. Send one or more messages to the topic.
5. Send an empty control message to indicate the end of the message stream.
6. Close the connection in a finally block, automatically closing the session and TopicPublisher.

# TopicSubscriber (1)

1. Perform a JNDI lookup of the TopicConnectionFactory and Topic.
2. Create a TopicConnection and a TopicSession.
3. Create a TopicSubscriber.
4. Create an instance of the TextListener class and registers it as the message listener for the TopicSubscriber:

```
listener = new TextListener();
subscriber.setMessageListener(listener);
```

5. Start the connection, causing message delivery to begin.

# TopicSubscriber (2)

6. Listen for the messages published to the topic, stopping when the user types the character q or Q:

```
System.out.println("To end program, type Q or q, " +
 "then <return>");

InputStreamReader = new InputStreamReader(System.in);
while (!((answer == 'q') || (answer == 'Q'))) {
 try {
 answer = (char) inputStreamReader.read();
 } catch (IOException e) {
 System.out.println("I/O exception: "
 + e.toString());
 }
}
```

7. Close the connection, which automatically closes the session and TopicSubscriber.

# Message Listener

1. When a message arrives, the `onMessage` method is called automatically.
2. The `onMessage` method converts the incoming message to a `TextMessage` and displays its content. If the message is not a text message, it reports this fact:

```
public void onMessage(Message message) {
 TextMessage msg = null;

 try {
 if (message instanceof TextMessage) {
 msg = (TextMessage) message;
 System.out.println("Reading message: " +
 msg.getText());
 } else {
 System.out.println("Message is not a " +
 "TextMessage");
 }
 } catch (JMSEException e) {
 System.out.println("JMSEException in onMessage(): " +
 e.toString());
 } catch (Throwable t) {
 System.out.println("Exception in onMessage(): " +
 t.getMessage());
 }
}
```

# AMQP

## Advanced Message Queuing Protocol

- Specifikacija otvorenog protokola za komunikaciju porukama i komunikaciju na načelu objavi-preplati.
- Popularan protokol i raširena primjena zbog niza implementacija: RabbitMQ, OpenAMQ, StortMQ, Apache QPid...

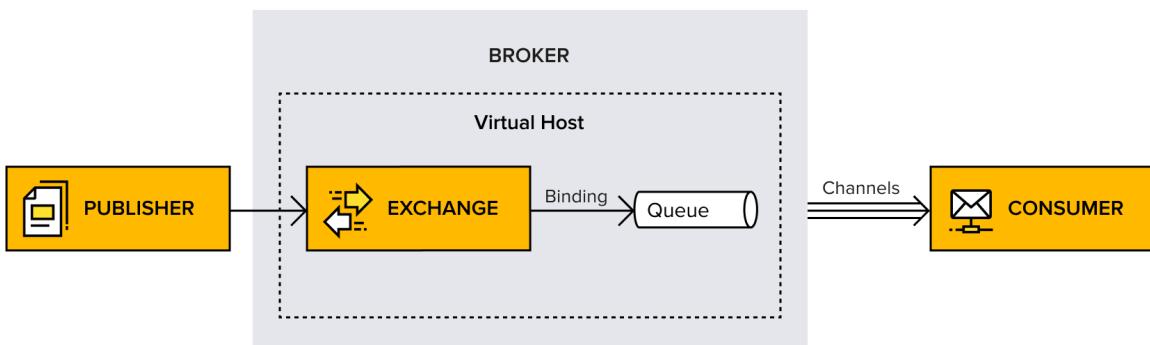
Version 1.0, OASIS Standard, 29.10.2012.

<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html>

Version 0.91, specifikacija [AMQP WG](#) iz 2008.

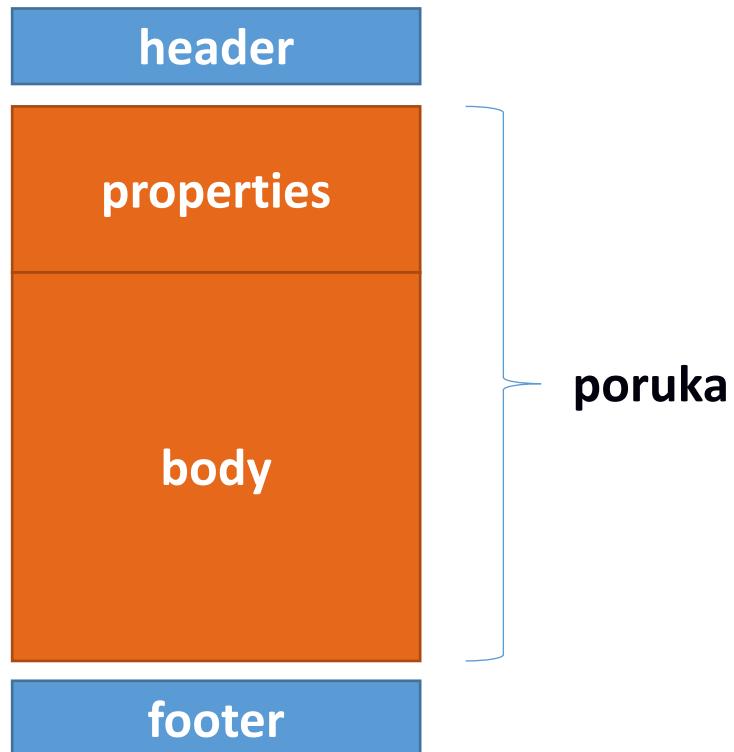
<http://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf>.

# Osnovni koncepti AMQP-a (v0.91)



- *Exchange*: entitet unutar posrednika koji usmjerava poruke u repove
- *Virtual host*: logički kontejner posrednika, odvaja različite aplikacije koje koriste istu instancu RabbitMQ posrednika
- *Channel*: virtualna veza unutar TCP konekcije koja povezuje posrednika s klijentom
- *Binding*: virtualna poveznica između *exchange-a* i repa
- *Publisher* (ili *producer*): objavljuje poruke na *exchange*
- *Consumer*: vezan je uz rep (*queue*) i definira *binding*

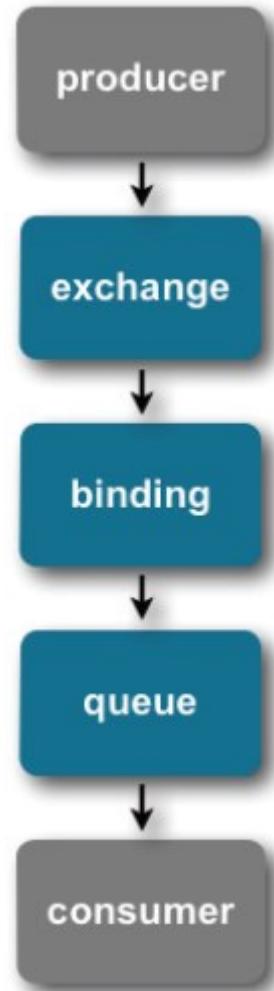
# AMQP poruke



- Mreža ne smije mijenjati poruku
- Header i footer se mogu mijenjati u mreži
- Svaka poruka dobiva jedinstveni ID
- Na čvoru smije postojati samo jedna kopija poruke
- Body type: byte message

# Kako se dostavljaju poruke?

- *Producer*: šalje poruke u *exchange* i dodaje *routing key* uz poruku
- Exchange je povezan s repom putem poveznice (*binding*)
- Binding definira *consumer* (*consumer-driven messaging*), a specificira kakve poruke trebaju biti usmjerene iz *exchangea* do repa
- Consumer je vezan uz rep i prima poruke iz repa
- Uspoređuje se *routing key* i *binding*, ako je uvjet zadovoljen, poruka se isporučuje repu s definiranim *bindingom*



# RabbitMQ

- *open source message queuing software*
- pisan u programskom jeziku Erlang
- implementira AMQP v0.91, postoji [plugin](#) za AMQP v1.0
- podržani protokoli
  - <https://www.rabbitmq.com/protocols.html>

STANDALONE

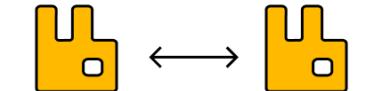


RabbitMQ  
Broker

CLUSTER



RabbitMQ  
Broker



FEDERATION



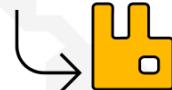
RabbitMQ  
Broker



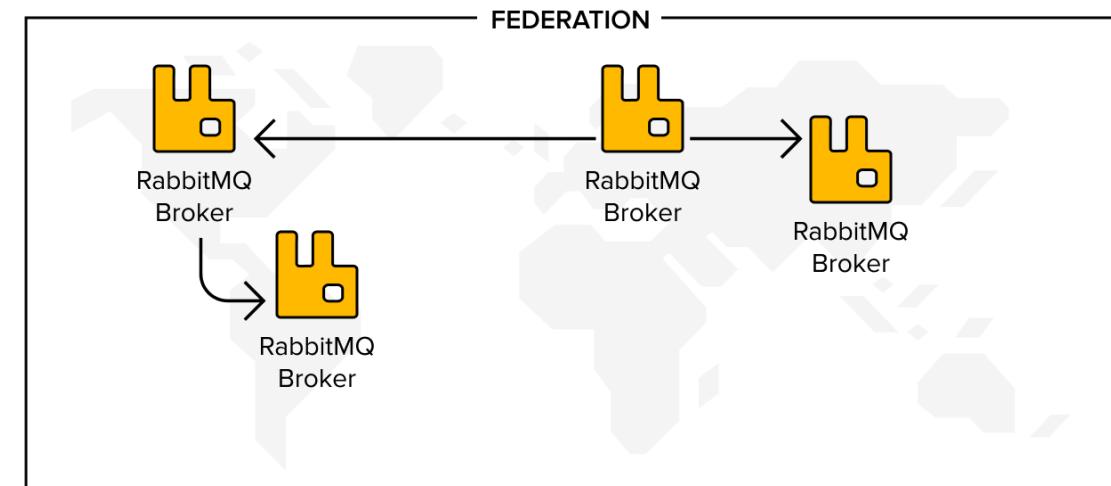
RabbitMQ  
Broker



RabbitMQ  
Broker



RabbitMQ  
Broker

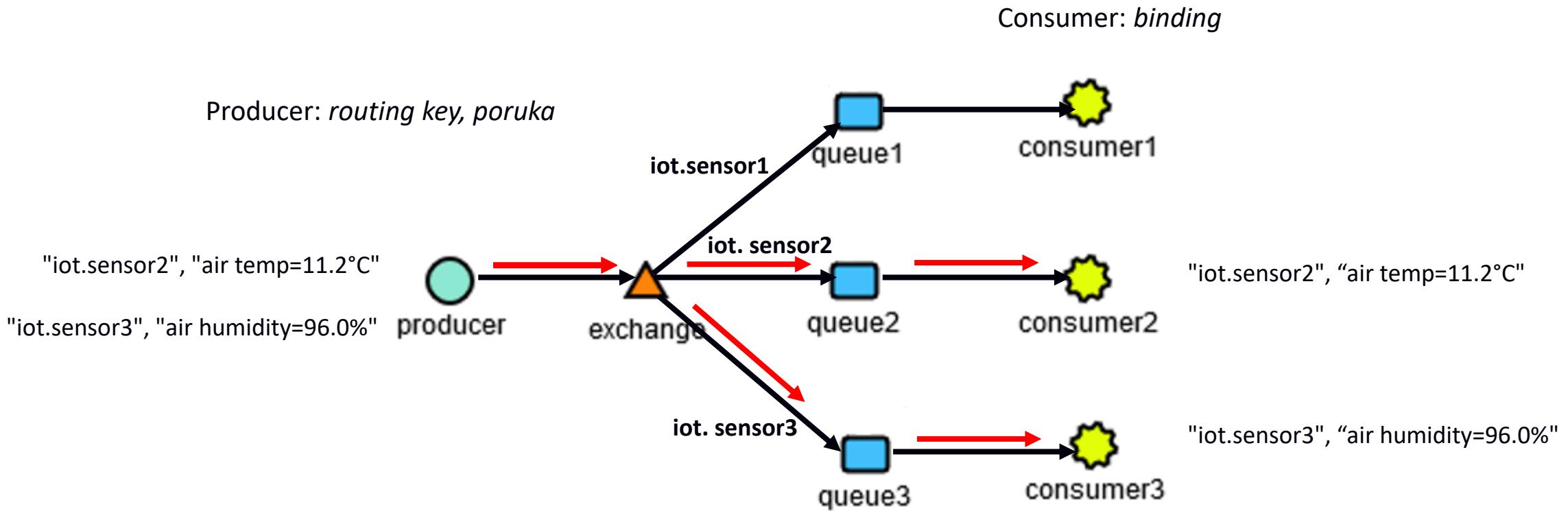


Različite topologije i organizacije posrednika

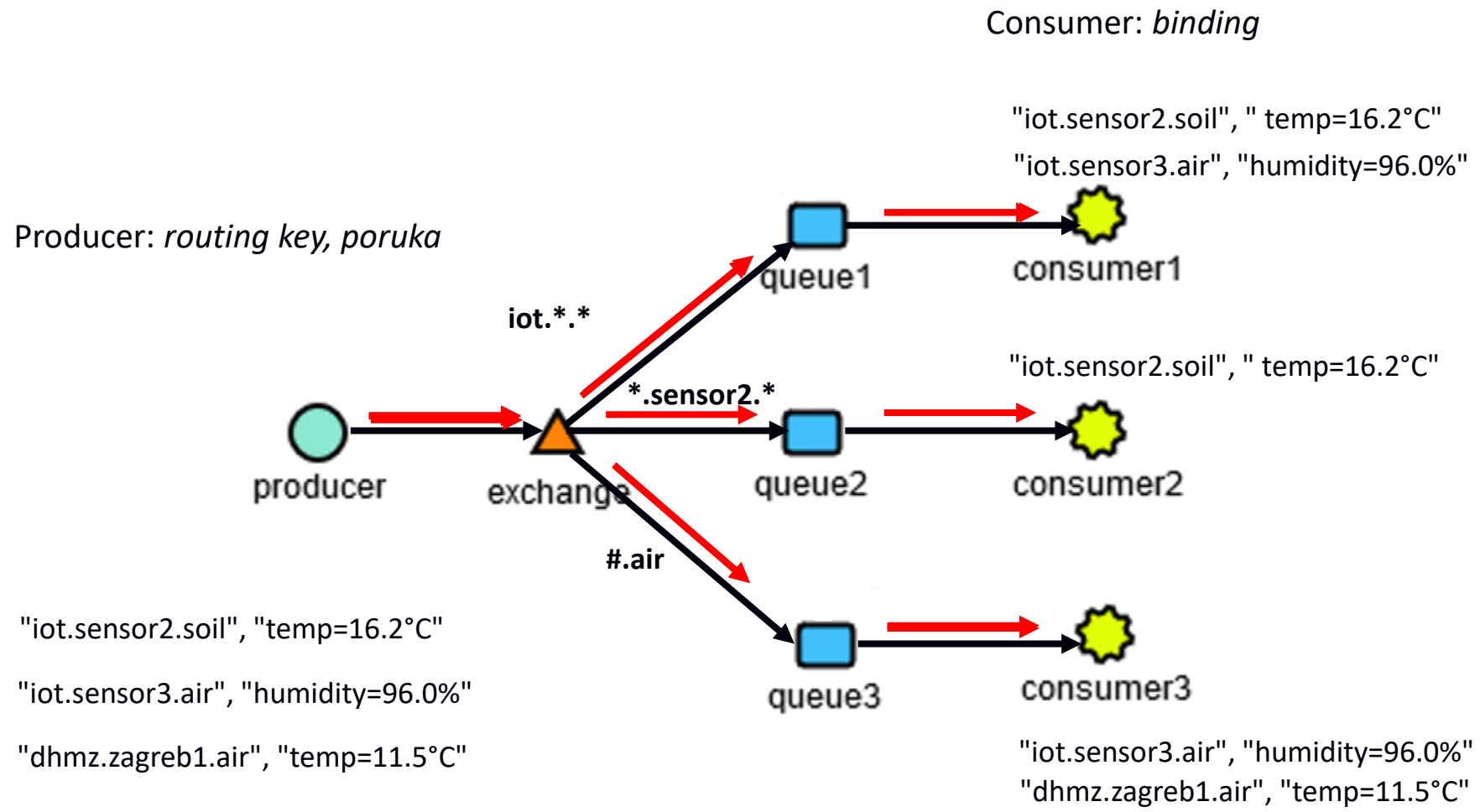
# RabbitMQ: različiti komunikacijski modeli

- *Direct Exchange*: odgovara *point-to-point* modelu JMS-a
  - isporučuje poruku u rep samo ako *routing key*-a poruke zadovoljava uvjet *binding*-a vezanog uz rep
  - poruka može biti isporučena u više repova ako više *binding*-a odgovara *routing key*-u
- *Fanout i Topic Exchange*: odgovara *publish/subscribe* modelu JMS-a
  - *Fanout Exchange*: šalje sve poruke na sve repove spojene na *exchange*
  - *Topic Exchange*: omogućuje filtriranje poruka i definiranje *binding*-a uporabom specijalnih znakova # i \*

# RabbitMQ: Direct Exchange



# RabbitMQ: Topic Exchange



# Primjer 2: AMQP - Producer

```
private final static String EXCHANGE_NAME = "MyExchange";

public static void main(String[] args) throws Exception {
 ConnectionFactory factory = new ConnectionFactory();
 factory.setHost("localhost");
 Connection connection = factory.newConnection();
 Channel channel = connection.createChannel();

 channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.TOPIC);
 String routingKey = "iot.sensor2.soil";
 String message = "temp 16.2 C";
 channel.basicPublish(EXCHANGE_NAME, routingKey, null, message.getBytes());

 channel.close();
 connection.close();
}
```

RabbitMQ: producer šalje poruku na exchange pod nazivom "MyExchange", veže *routing key* uz poruku

# AMQP - Consumer

Consumer definira rep i povezuje ga s *exchangeom*, kada binding odgovara *routing key*-u, poruka se isporučuje

```
...
channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.TOPIC);
String queueName = channel.queueDeclare().getQueue();

String binding = "*.sensor2.*";
channel.queueBind(queueName, EXCHANGE_NAME, binding);

Consumer consumer = new DefaultConsumer(channel) {
 @Override
 public void handleDelivery(String consumerTag, Envelope envelope,
 AMQP.BasicProperties properties, byte[] body) throws IOException{
 String message = new String(body, "UTF-8");
 System.out.println("Received: " + message);
 }
};
channel.basicConsume(queueName, true, consumer);
```

# Literatura: AMQP

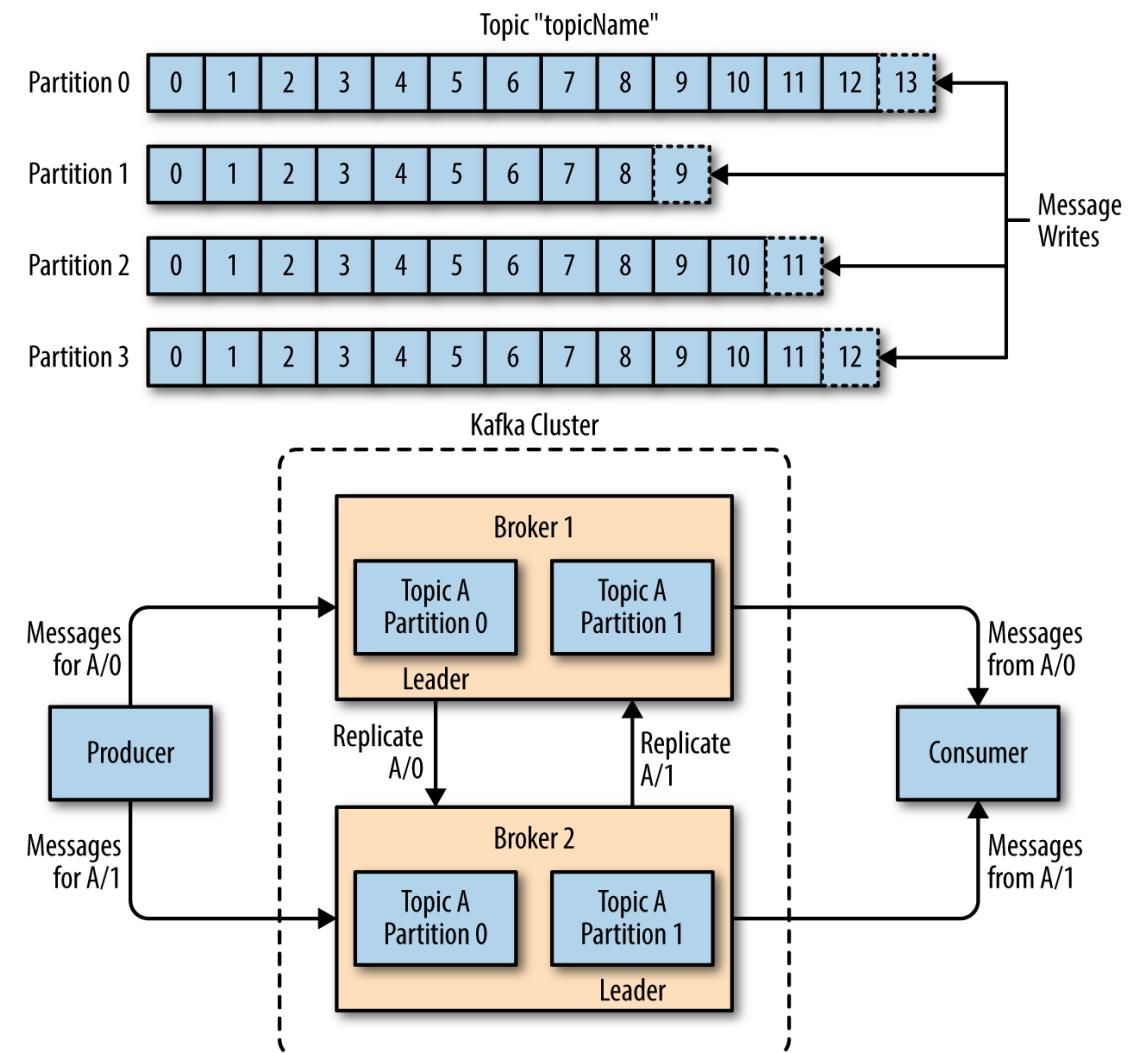
- RabbitMQ Tutorials, <https://www.rabbitmq.com/getstarted.html>
- RabbitMQ Simulator, <http://tryrabbitmq.com/>
- Mark Richards: Understanding the Differences between AMQP & JMS, 2011  
<http://www.wmrichards.com/amqp.pdf>
- Spring messaging with RabbitMQ  
<https://spring.io/guides/gs/messaging-rabbitmq/>

# Apache Kafka

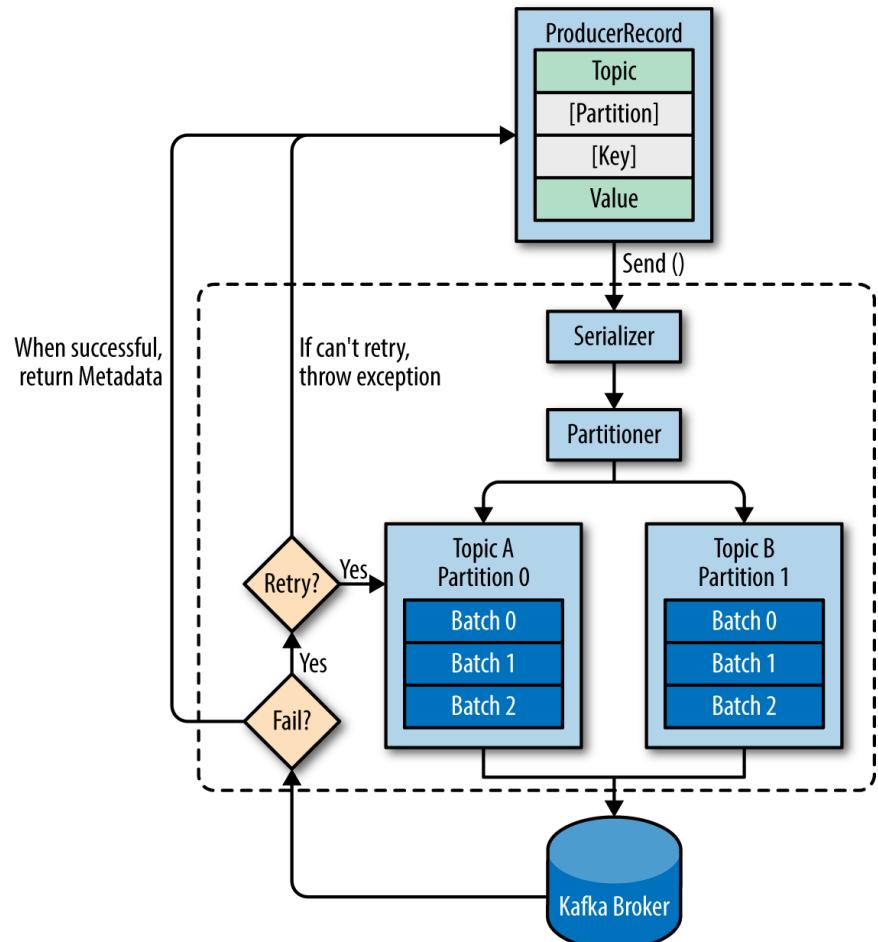
- *streaming platform*: sustav koji omogućuje objavljivanje i pretplatu na tokove podataka, njihovo pohranjivanje i obradu
- Platforma inicijalno razvijena u LinkedIn-u za “*user activity tracking*”
- Obrada „velikih“ tokova podataka, veliki broj objavljivača i pretplatnika, skalabilnost je na prvom mjestu dizajna sustava
- *Data retention*: omogućuje pohranu i čuvanje podataka iz toka, podaci se pohranjuju na disk
- Može se konfigurirati Kafka klaster koji koristi više brokera, a i više klastera koji pokrivaju veći broj podatkovnih centara

# Terminologija

- *Message*: (key, value), key pridjeljuje poruku particiji, ali ne mora biti definiran
- *Batch*: niz poruka koje se objavljuju na isti *topic* i particiju, može se koristiti kompresija
- *Topic*: sadrži više particija
- *Partition*: uređeni niz poruka
- *Producer*: objavljuje poruke na particiju
- *Consumer*: čita poruke iz particije
- Broker
- Cluster



# Kafka Producer



```
Properties kafkaProps = new Properties();
kafkaProps.put("bootstrap.servers",
"broker1:9092,broker2:9092");
kafkaProps.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
kafkaProps.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
producer = new KafkaProducer<String,
String>(kafkaProps);
```

Novi producer s nužnim parametrima

# Slanje poruke

1. *Fire-and-forget*: poruka se šalje bez potvrde primitka (ack=0)
2. *Synchronous send*: producer je uвijek asinkron (шалје се порука, а метода send() враћа Future object). У овом случају се користи метода get() која чека информацију да је send() био успјешан
3. *Asynchronous send*: pozива методу send() с посебном callback функцијом која прима одговор од брокера у будућности о томе је ли метода send() успјешно извршена или не

```
ProducerRecord<String, String>
record =
 new ProducerRecord<>(Topic,
key, value);
try {
 producer.send(record);
} catch (Exception e) {
 e.printStackTrace();
}
```

# Consumer

```
Properties props = new Properties();

props.put("bootstrap.servers",
"broker1:9092,broker2:9092");

props.put("group.id", "CountryCounter");

props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");

props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");

KafkaConsumer<String, String> consumer =
new KafkaConsumer<String, String>(props);
```

- Jednostavna pretplata na topic  
consumer.subscribe(Collections.singletonList (Topic))
- Asynchronous Commit  
Duration timeout =  
Duration.ofMillis(100);  
while (true) {  
 ConsumerRecords<String, String>  
 records = consumer.poll(timeout);  
 for (ConsumerRecord<String, String>  
 record : records) {  
 System.out.printf("topic = %s,  
partition = %s, offset = %d, customer =  
%s, country = %s\n", record.topic(),  
record.partition(), record.offset(),  
record.key(), record.value());  
 }  
 consumer.commitAsync(); 1  
}

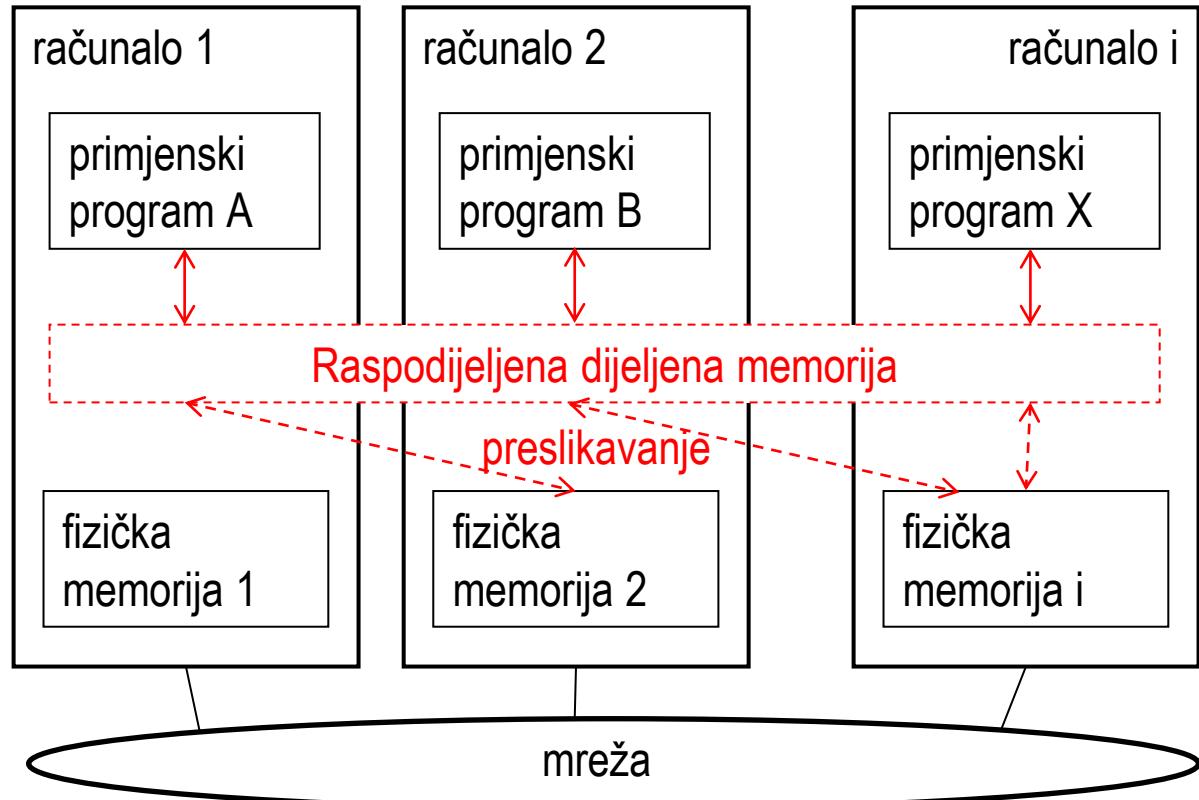
# Literatura

- **Kafka: The Definitive Guide, 2nd Edition, By Gwen Shapira, Todd Palino, Rajini Sivaram, Krit Petty, PUBLISHED BY:O'Reilly Media, Inc. PUBLICATION DATE:November 2021, PRINT LENGTH:455 pages**
- <http://kafka.apache.org/downloads> (koristimo verziju 2.8.1)

# Sadržaj predavanja

- Komunikacija porukama
- Model objavi-preplati
  - Primjeri programske opreme za komunikaciju porukama: JMS, AMQP, Kafka
- Dijeljeni podatkovni prostor

# Raspodijeljena dijeljena memorija



- Posrednički sloj koji nudi transparentan pristup dijeljenoj memoriji računala bez zajedničke fizičke memorije

*Distributed Shared Memory (DSM)*

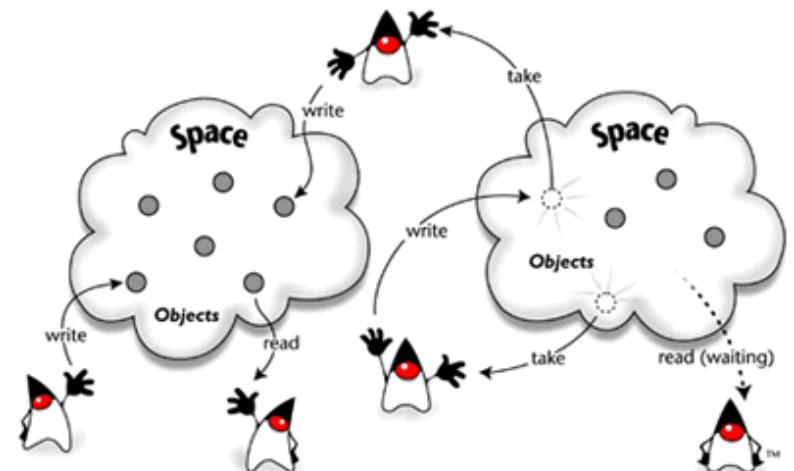
- omogućuje transparentan pristup fizičkoj memoriji na drugim računalima (primjenski program ima dojam da pristupa vlastitoj fizičkoj memoriji)
- upravlja replikama podataka, na računalu se čuvaju lokalne kopije podataka kojima je nedavno pristupao primjenski program X

# Dijeljeni podatkovni prostor

engl. *shared data/tuple spaces*

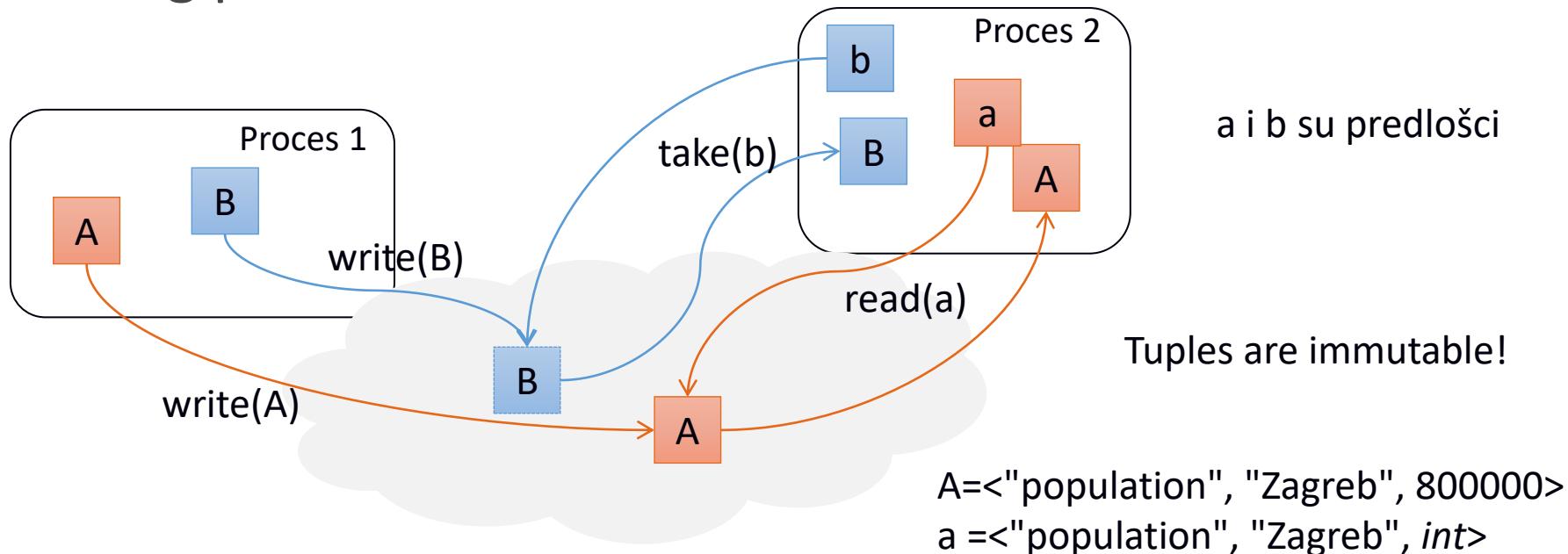
- arhitektura temeljna na podacima (*content-addressable memory*)
- procesi mogu dodati, čitati i “izvaditi” *tuple* iz zajedničkog dijeljenog podatkovnog prostora (*tuple space*)
- *tuple*: slijed podataka, za svaki je definiran tip
- primjeri: Linda, JavaSpaces,  
TSpaces

Izvor: "JavaSpaces Principles, Patterns, and Practice"  
<http://java.sun.com/developer/Books/JavaSpaces/introduction.html>

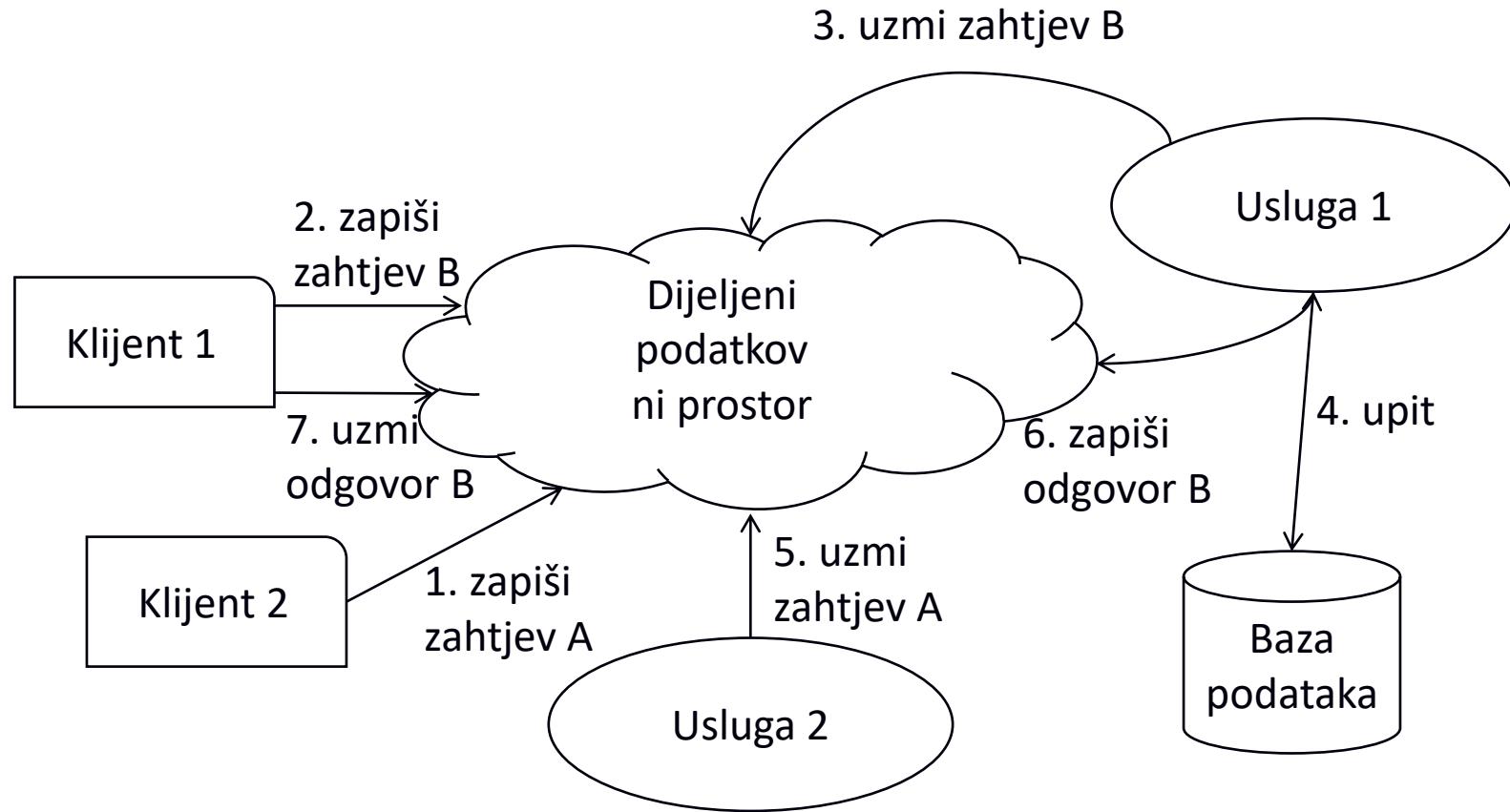


# Operacije

- **write (A)** – dodaj tuple A u raspodijeljeni podatkovni prostor
- **read (a) → A** – vraća tuple A koji odgovara predlošku a
- **take (b) → B** – vraća tuple B koji odgovara predlošku b i biće ga iz podatkovnog prostora



# Primjer aplikacije



# Obilježja dijeljenog podatkovnog prostora

- **vremenska neovisnost**
  - procesi ne moraju istovremeno biti aktivni radi komunikacije, dijeljeni podatkovni prostor pohranjuje poruku
- **anonimna komunikacija** (temelji se na sadržaju podataka)
- komunikacija je **perzistentna**
- **asinkrona komunikacija**
  - proces dodaje podatak u podatkovni prostor i nastavlja obradu
- pokretanje komunikacije na načelu ***pull***
  - proces eksplicitno šalje zahtjev za čitanje podatka iz dijeljenog podatkovnog prostora

# Pitanja za učenje i ponavljanje

- Objasnite značenje vremenske i prostorne neovisnosti za komunikaciju procesa. Navedite jesu li komunikacija porukama i komunikacija na načelu objavi-preplati vremenski i prostorno ovisne ili neovisne.
- Navedite sličnosti i razlike komunikacije na načelu objavi-preplati i dijeljenog podatkovnog prostora.
- Usporedite preplatu u sustavima objavi-preplati i predložak u sustavima s dijeljenim podatkovnim prostorom. Zašto je moguće realizirati tzv. vremenski i prostorno neovisnu komunikaciju?
- Gdje se filtriraju obavijesti u raspodijeljenom sustavu objavi-preplati koji koristi preplavljivanje obavijestima?
- Zašto za raspodijeljeni sustav objavi-preplati koji koristi preplavljivanje preplatama kažemo da filtrira obavijesti na samom ulazu u mrežu posrednika?
- Skicirajte primjer raspodijeljenog izvođenja sustava objavi-preplati s 3 posrednika gdje je  $P_1$  spojen na  $B_1$ ,  $S_1$  na  $B_2$ , a  $S_3$  na  $B_3$  za slijed događaja sa slajda 22.

# Literatura

1. G. Coulouris, J. Dollimore, T. Kindberg: *Distributed Systems: Concepts and Design*, 5th edition, Addison-Wesley, 2012  
poglavlje 6
2. Maarten van Steen, Andrew S. Tanenbaum (2017.), *Distributed Systems 3<sup>rd</sup> edition*, Createspace Independent Publishing Platform  
poglavlje 4.3 (bez dijela o Socketima)



SVEUČILIŠTE U ZAGREBU



**Diplomski studij**  
**Računarstvo**  
Znanost o mrežama  
Programsko inženjerstvo i  
informacijski sustavi  
Računalno inženjerstvo  
**Ostali (slobodni izborni  
predmet)**

# Raspodijeljeni sustavi

## 6. Sinkronizacija procesa u vremenu

Ak. god. 2022./2023.

# Creative Commons



- slobodno smijete:

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
  - **prerađivati** djelo



- pod sljedećim uvjetima:

- **imenovanje**: morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
  - **nekomercijalno**: ovo djelo ne smijete koristiti u komercijalne svrhe.
  - **dijeli pod istim uvjetima**: ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.



*U slučaju daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.*

*Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.*

*Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.*

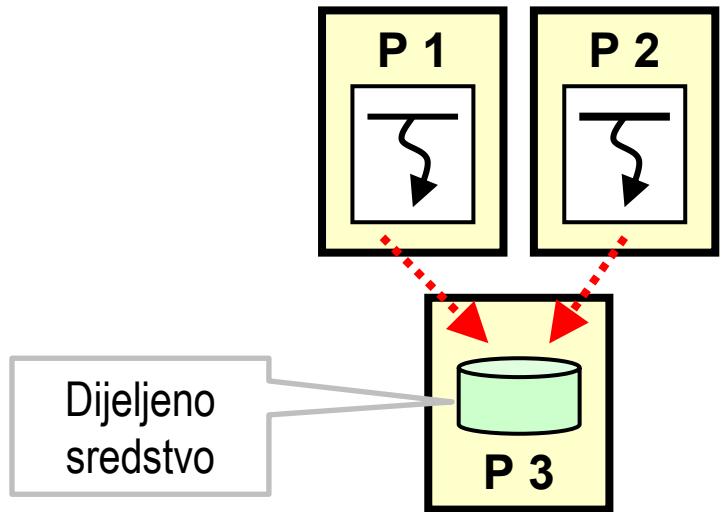
*Tekst licence preuzet je s <http://creativecommons.org/>*

# Sadržaj predavanja

- Motivacija: potreba za sinkronizacijom procesa u raspodijeljenoj okolini
- Primjena sata u jednoprocesorskoj okolini
- Primjena sata u raspodijeljenoj okolini
- Sinkronizacija tijeka izvođenja procesa
- Međusobno isključivanje procesa

# Potreba za sinkronizacijom procesa (1/4)

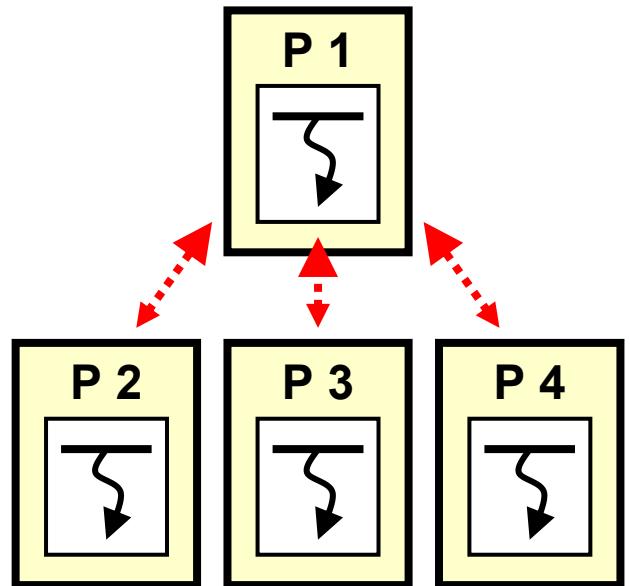
- Uporaba **dijeljenog sredstva** u raspodijeljenoj okolini
  - Procesi istodobno pristupaju dijeljenom sredstvu
  - Potrebno je ostvariti pristup dijeljenom sredstvu na međusobno isključiv način
  - Raspodijeljeni procesi moraju postići dogovor o redoslijedu pristupa sredstvu



# Potreba za sinkronizacijom procesa (2/4)

- **Nadgledanje i upravljanje** nad izvođenjem poslova u raspodijeljenoj okolini

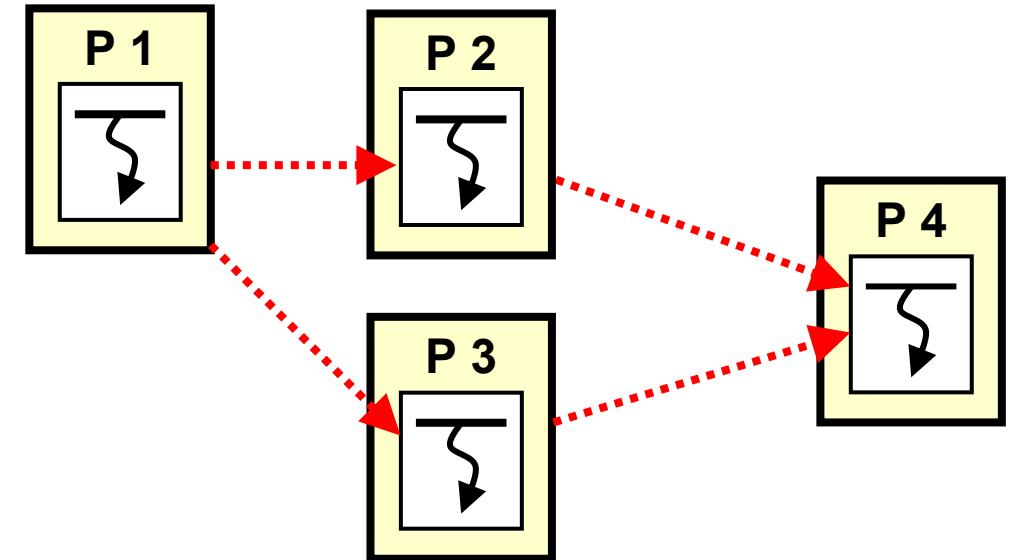
- Odabir upravljačkog procesa (sjetite se primjera s predavanja o modeliranju raspodijeljenog sustava)
- Upravljački proces nadzire i određuje aktivnosti radnih procesa u raspodijeljenoj okolini



# Potreba za sinkronizacijom procesa (3/4)

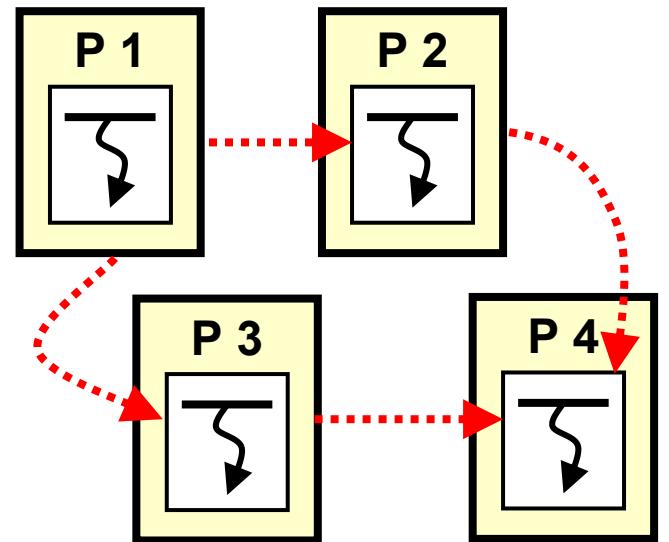
- Usuglašavanje **vremenskog redoslijeda izvođenja akcija**

- Potrebno je omogućiti vremenski tijek izvođenja akcija na procesima u raspodijeljenoj okolini ako postoji međuvisnost među procesima



# Potreba za sinkronizacijom procesa (4/4)

- Uspostava **suradnje skupa procesa** u raspodijeljenoj okolini
  - Ostvarivanje vremenski i **prostorno** usklađenog raspodijeljenog tijeka izvođenja (proširenje 3. primjera)
  - Primjer: P2P sustav za dijeljenje datoteka

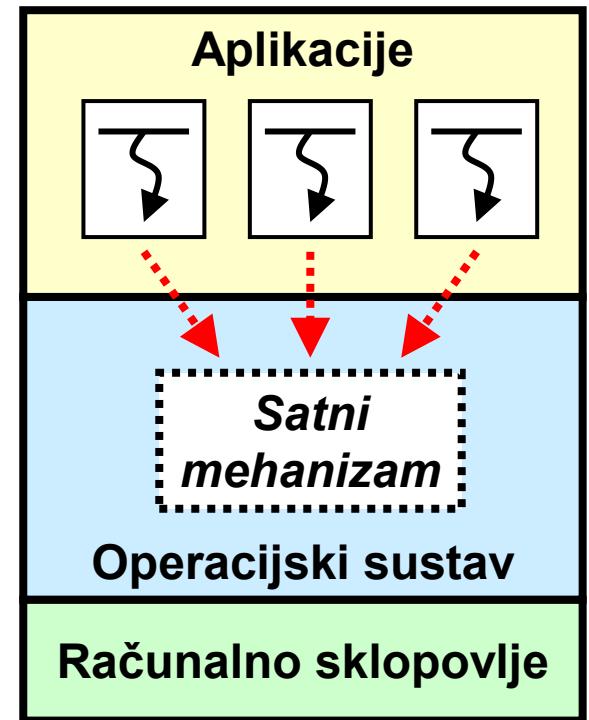


# Sadržaj predavanja

- Motivacija: potreba za sinkronizacijom procesa u raspodijeljenoj okolini
- Primjena sata u jednoprocesorskoj okolini
- Primjena sata u raspodijeljenoj okolini
- Sinkronizacija tijeka izvođenja procesa
- Međusobno isključivanje procesa

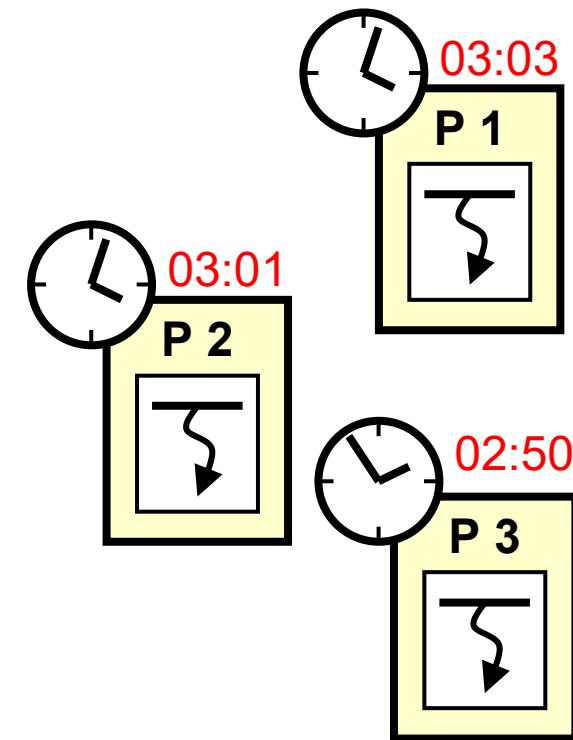
# Primjena sata u jednoprocesorskoj okolini

- Podsjetimo se: satni mehanizam operacijskog sustava
  - Izведен uporabom kristala kvarca, osciliraju pod naponom zbog piezoelektričkog efekta
- Aplikacije
  - Procesi koriste i upravljaju mehanizmom sata
  - Primjena programskih knjižnica za uporabu satnog mehanizma
- Zatvorena okolina
  - Predvidiva vremena izvođenja procesa
  - Jednostavnija sinkronizacija procesa u vremenu



# Fizički i logički sat

- Svako računalo ima vlastiti satni mehanizam
  - Satovi nisu usklađeni
  - Satovi imaju različiti takt
  - Satovi imaju različita odstupanja (pogrešku)
- Usuglašavanje vremena
  - Fizički sat u raspodijeljenoj okolini
  - Logički sat u raspodijeljenoj okolini

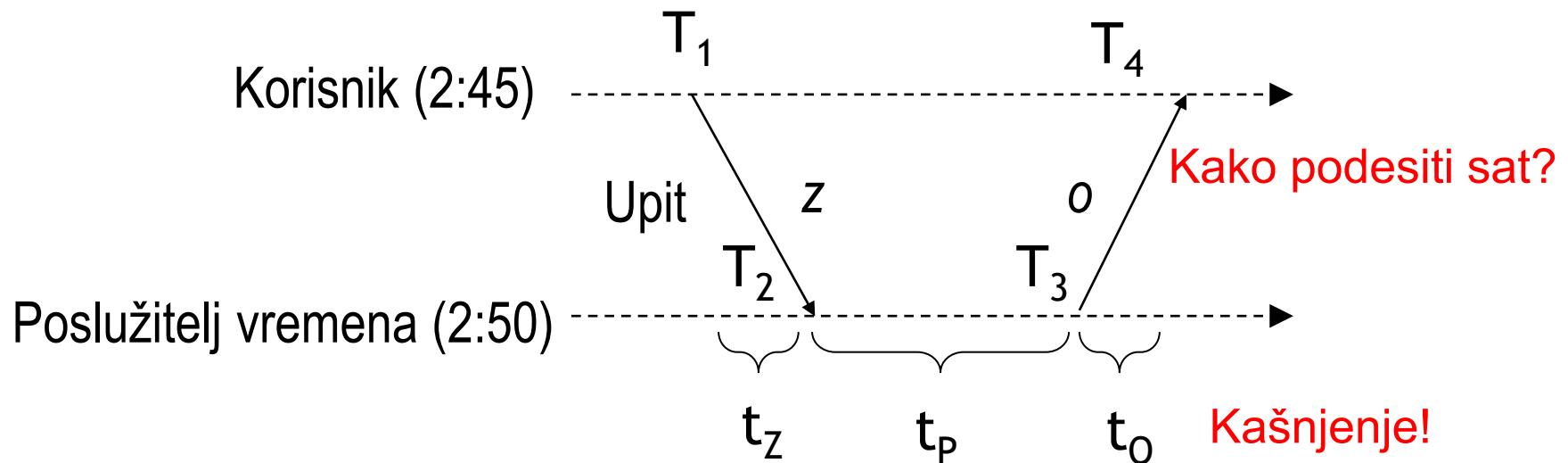


# Fizički sat u raspodijeljenoj okolini

- **Cristianov algoritam**
  - Razvio ga je F. Cristian (1989)
  - Primjena poslužitelja s točnim vremenom, sinkronizacija prema vanjskom izvoru
  - Dohvaćanje informacije o vremenu prema potrebi
- **Algoritam Berkeley**
  - Razvili su ga R. Gusella i S. Zatti na University of California, Berkeley (1989)
  - Primjena upravitelja vremena, sinkronizacija unutar skupine procesa
  - Periodičko odašiljanje informacije o vremenu

# Cristianov algoritam (1/2)

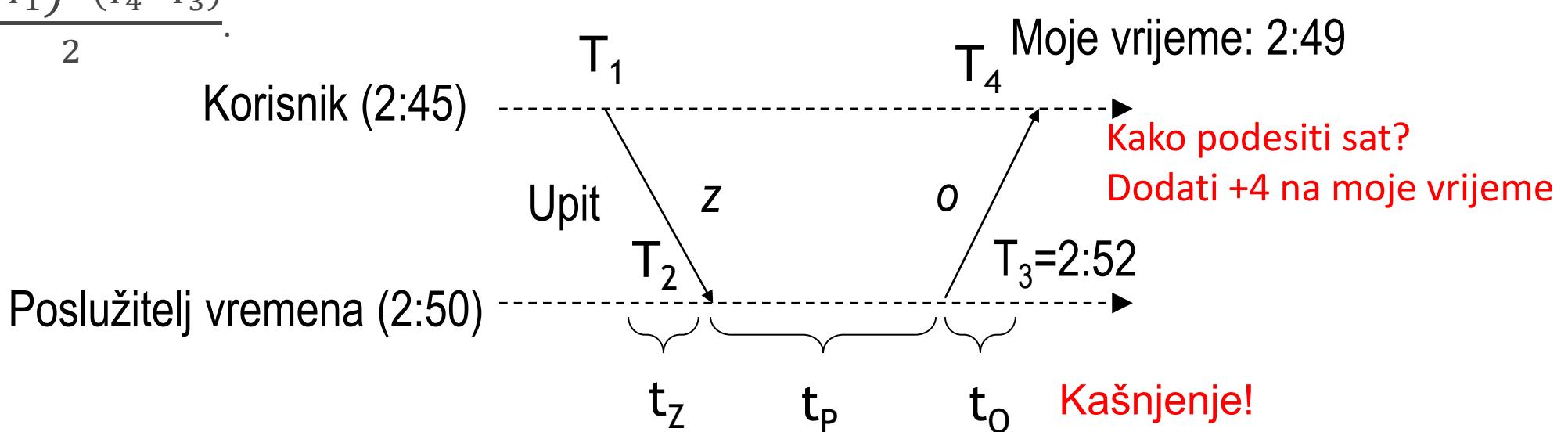
- Primjena poslužitelja vremena
- Koraci algoritma
  - 1) Korisnički proces upućuje zahtjev za dohvat vremena ( $z$ )
  - 2) Poslužitelj vremena prima i obrađuje zahtjev te šalje trenutno vrijeme ( $o$ )



# Cristianov algoritam (1/2)

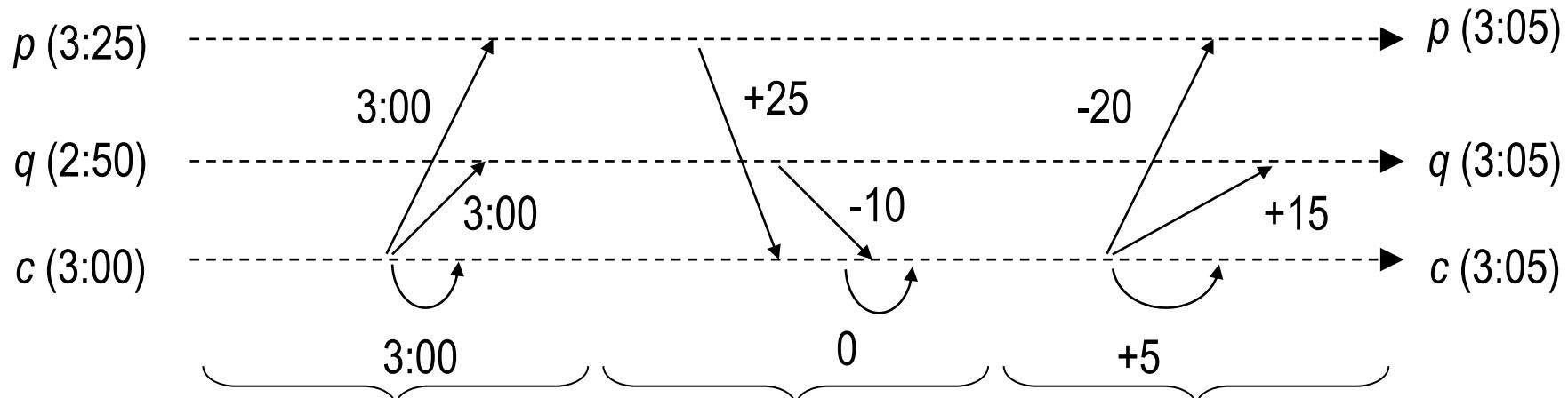
3. Odgovor sadrži  $T_2$  i  $T_3$  na poslužitelju u trenutku slanja odgovora.
4. Klijent na osnovi vremenskih trenutaka koje je primio u poruci i izmjerениh vremenskih trenutaka  $T_1$  i  $T_4$  pomiče svoje lokalno vrijeme za

$$\theta = T_3 + t_O - T_4 \approx T_3 + \frac{t_Z + t_O}{2} - T_4 = T_3 + \frac{(T_4 - T_1) - (T_3 - T_2)}{2} - T_4 = \frac{(T_2 - T_1) - (T_4 - T_3)}{2}.$$



# Algoritam Berkeley

- Primjena upravitelja vremena
- Koraci algoritma
  - 1) Upravljački proces  $c$  šalje vrijeme procesima  $p, q, c$
  - 2) Procesi  $p, q, c$  šalju razliku vremena upravljačkom procesu  $c$
  - 3) Upravljački proces  $c$  šalje pomak procesima  $p, q, c$



# Network Time Protocol (NTP)

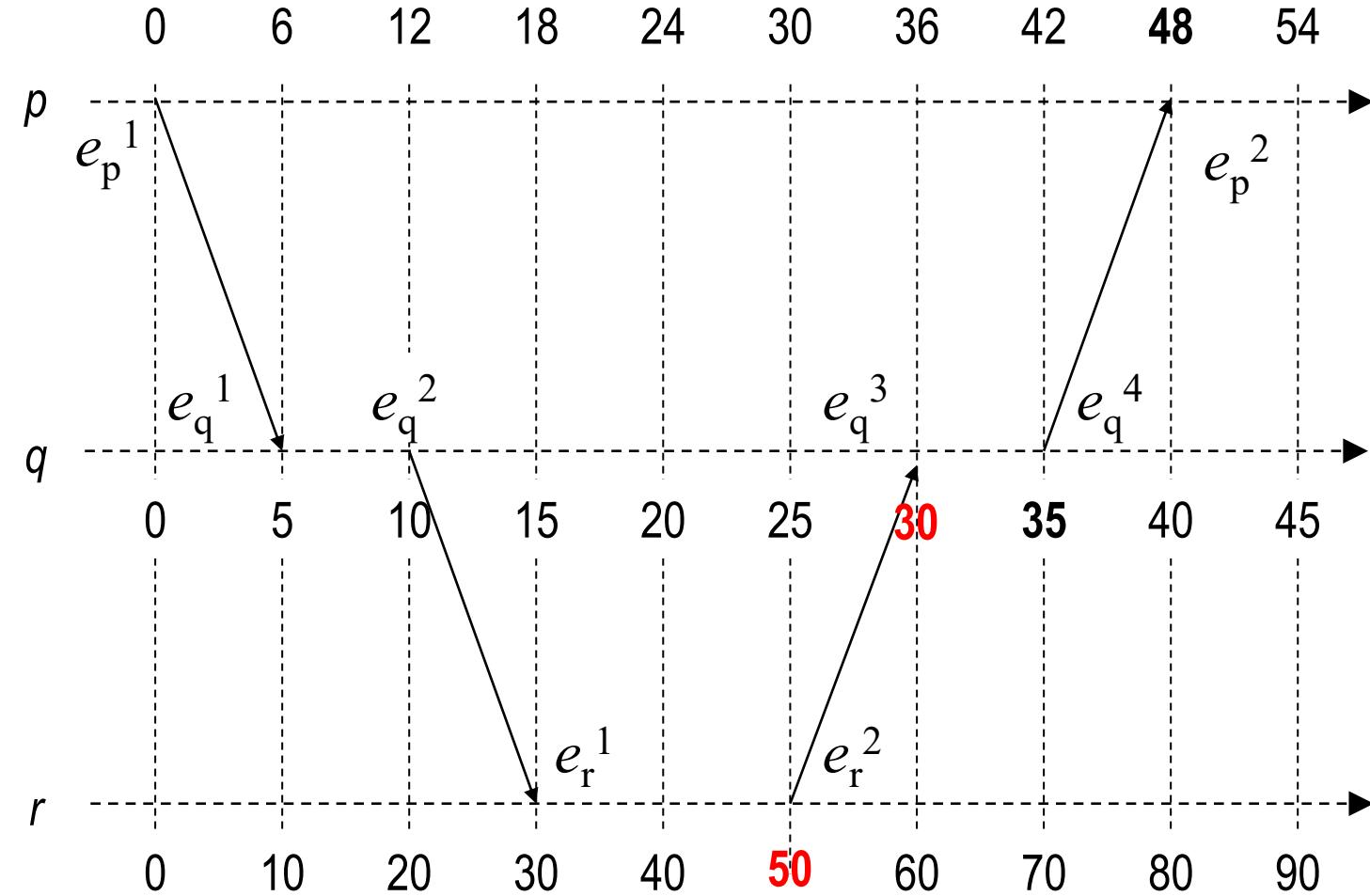
- Definira arhitekturu usluge za sinkronizaciju satnih mehanizama u raspodijeljenoj okolini i protokol za isporuku informacija o vremenu u Internetu
- Hjerarhijska organizacija NTP servisa
  - *primary servers*: povezani direktno na izvor sinkroniziran na UTC (Coordinated Universal Time)
  - *secondary servers*: sinkroniziraju se u odnosu na *primary servers*, itd.
  - preciznost: ~10 ms za računala na javnom Internetu, ~1 ms za računala u LAN-u
- RFC 5905: Network Time Protocol Version 4: Protocol and Algorithms Specification, 2010
- **Nedostatak fizičkog sata:** fizički satni mehanizmi su potpuno neovisni, stoga samo primjenom fizičkih satnih mehanizama nije moguće odrediti odnos događaja u vremenu (npr. redoslijed aktivnosti)

# Primjer nedostatka fizičkog sata

$$e_p^1 \rightarrow e_q^1 \quad e_q^2 \rightarrow e_r^1$$

$$e_r^2 \rightarrow e_q^3 \quad e_q^4 \rightarrow e_p^2$$

$$\mathbf{T(e_r^2) > T(e_q^3)}$$



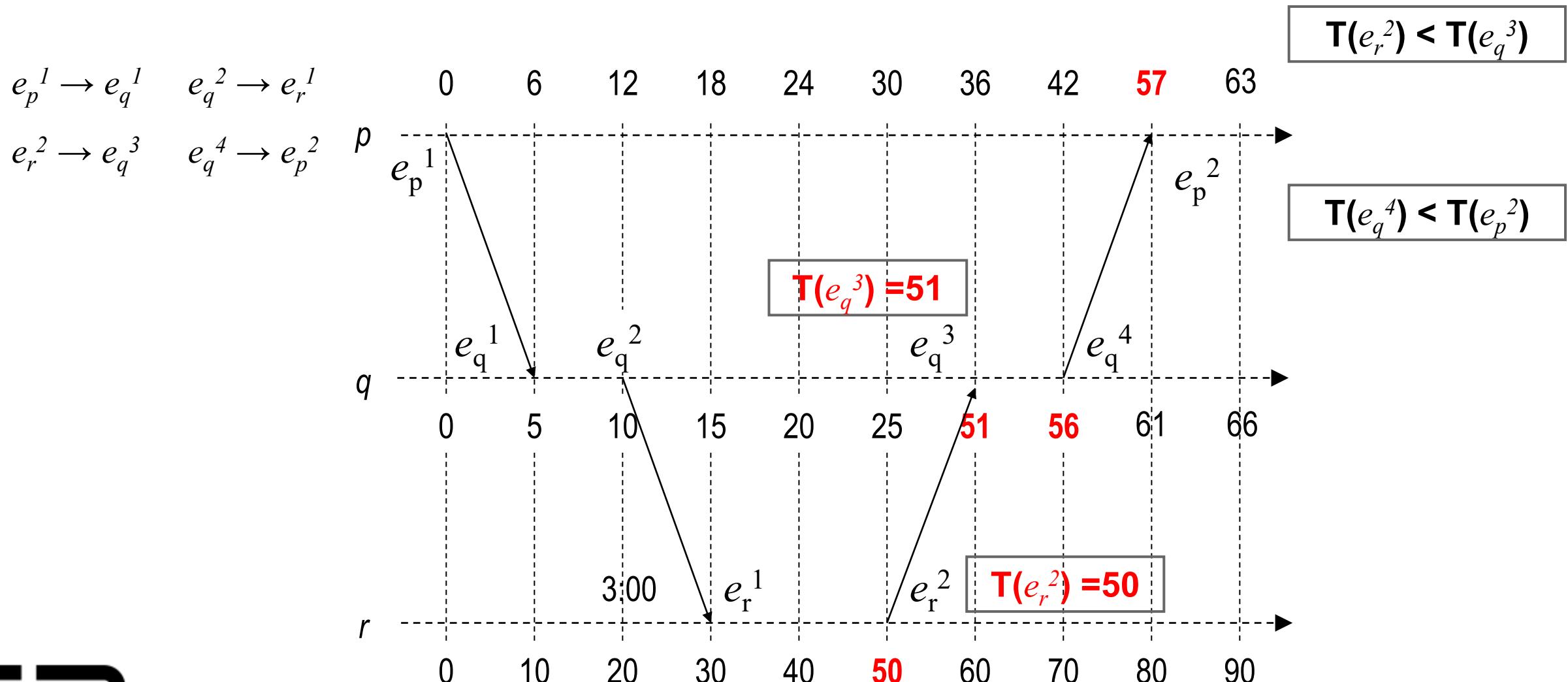
# Logičke oznake vremena

- **Usklađivanje globalnog tijeka vremena**
  - Primjena logičkih oznaka vremena (*timestamps*)
- **Vrste logičkih oznaka**
  - Skalarne oznake vremena
  - Vektorske oznake vremena

# Skalarne oznake vremena

- **Globalno logičko vrijeme**
  - Sva računala na jednak način bilježe tijek globalnog logičkog vremena
- **Oznake logičkog vremena**
  - Svakoj akciji  $a$  koju provode procesi u raspodijeljenoj okolini pridružena je jedinstvena oznaka vremena  $T(a)$
  - Ako za događaj  $a$  i  $b$  vrijedi uzročna relacija  $a \rightarrow b$  tada vrijedi da je akcija  $a$  ostvarena u vremenu prije akcije  $b$  [  $T(a) < T(b)$  ]

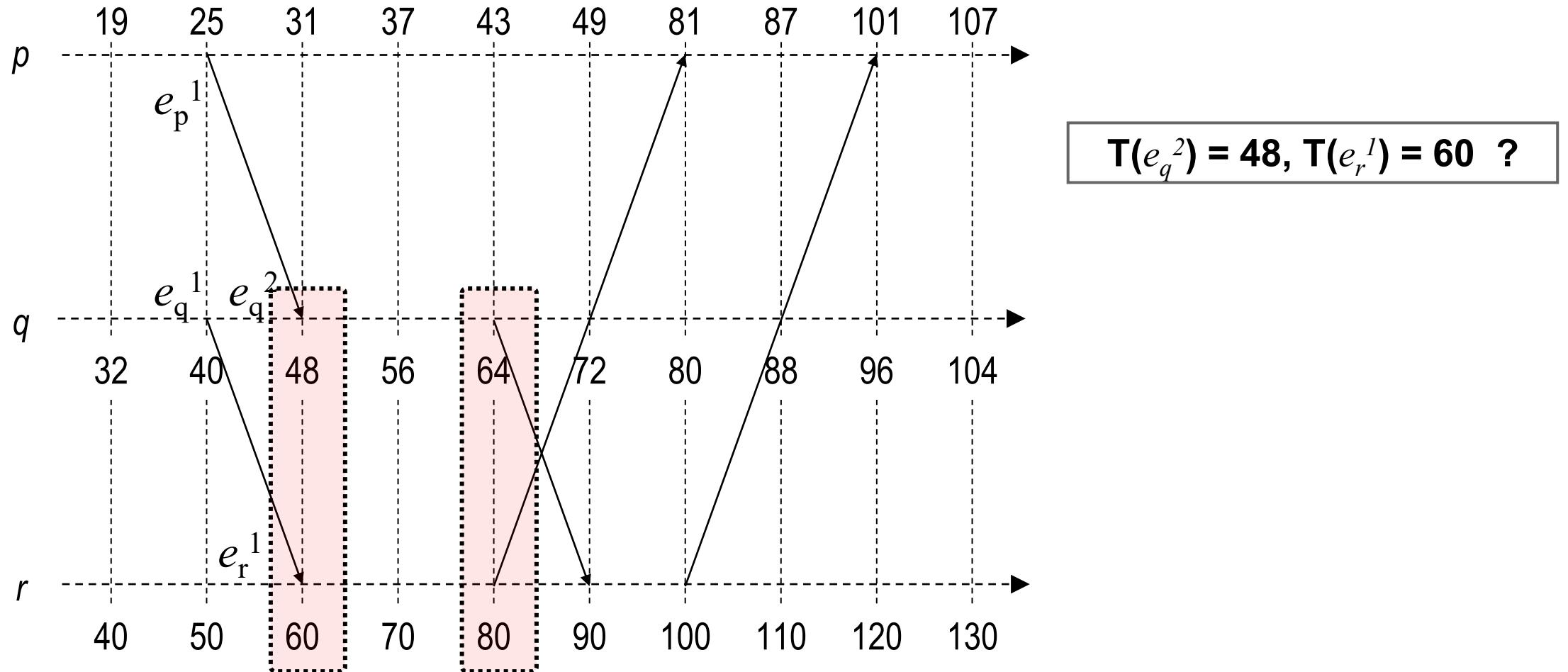
# Primjer uporabe skalarnih oznaka vremena



# Obilježja skalarne oznake vremena

- **Prednosti primjene skalarnih oznaka**
  - Tijek vremena zasnovan je na jednostavnom modelu
  - Svi procesi usklađeni su s globalnim tijekom vremena
  - Usuglašeni su vremenski trenutci nastupanja akcija u raspodijeljenoj okolini
- **Nedostatci primjene skalarnih oznaka**
  - Ako za događaje  $a$  i  $b$  vrijedi da je vremenska oznaka od  $a$  manja od vremenske oznake od  $b$ , to ne povlači nužno da je događaj  $a$  nastupio u vremenu prije događaja  $b$
  - $T(a) < T(b)$  ne povlači  $a \rightarrow b$

# Primjer nedostatka skalarnih oznaka



# Vektorske oznake vremena (1/2)

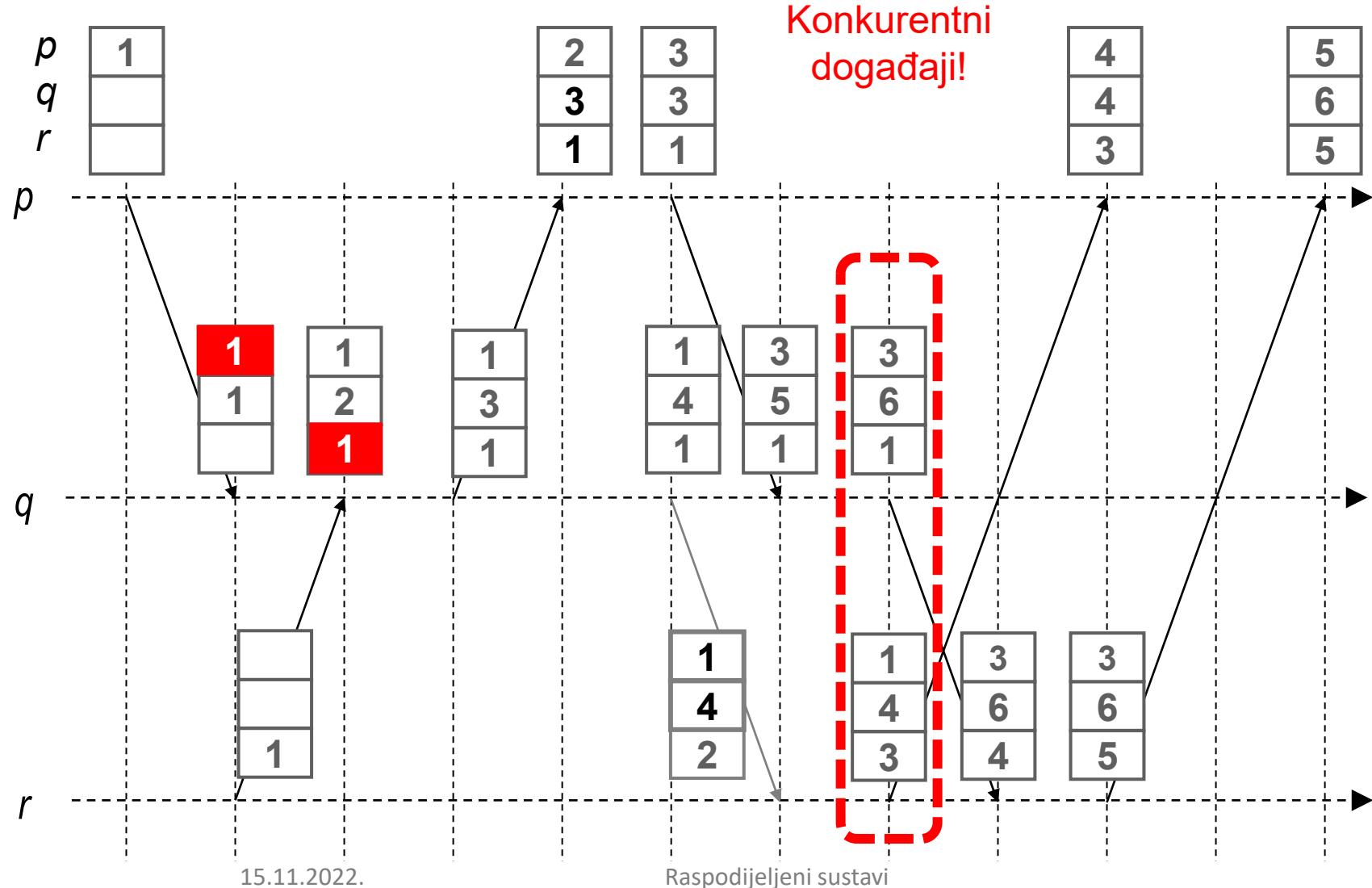
- Vektorska oznaka opisuje uzročno-posljedične veze između događaja u vremenu
  - Polje elemenata  $V[N]$  opisuje broj akcija (unutarnja akcija, slanje poruke, prijam poruke) provedenih od  $N$  procesa u raspodijeljenoj okolini
  - Procesi razmjenjuju vektorske oznake tijekom razmjene poruka
- Vektorska oznaka
  - $V_p[p]$  broj akcija koje je ostvario proces  $P_p$
  - $V_p[m]$  broj akcija za koje proces  $P_p$  zna da su ostvarene od strane procesa  $P_m$



# Vektorske oznake vremena (2/2)

- **Primjena vektorskih oznaka**
  - Ako za događaje  $a$  i  $b$  vrijedi  $V(a) < V(b)$  tada vrijedi da je događaj  $a$  nastupio u vremenu prije događaja  $b$ ,  $a \rightarrow b$
- **Za dvije vektorske oznake  $V_i$  i  $V_j$  vrijedi  $V_i < V_j$  ako:**
  - postoji barem jedan  $k$  za koji vrijedi  $V_i[k] < V_j[k]$ ,
  - za sve ostale  $l \neq k$  vrijedi  $V_i[l] \leq V_j[l]$ ,
  - $i, j, k, l \in [0, N-1]$  i
  - broj procesa u raspodijeljenoj okolini je  $N$

# Primjer uporabe vektorskih oznaka



# Vektorske oznake vremena

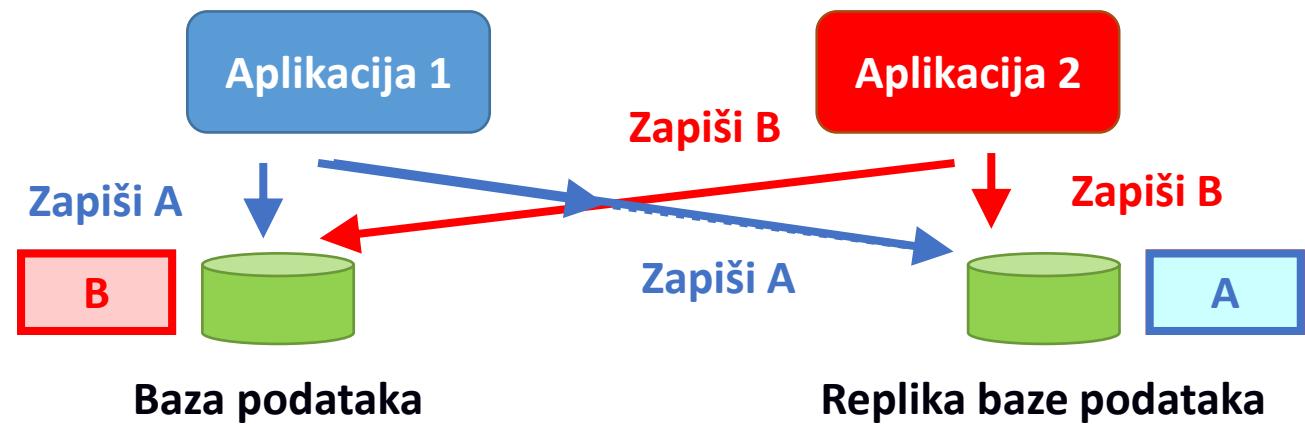
**Koraci algoritma za održavanje vektorskih oznaka:**

- 1) Početne vrijednosti svih vektorskih oznaka su postavljene na 0.
- 2) Za svaku unutarnju akciju procesa  $p$  uvećaj vremensku oznaku na procesu  $p$  pridijeljenu tome procesu za 1, tj.  $V_p[p]+1$ .
- 3) Prije slanja poruke na procesu  $p$  uvećaj oznaku  $V_p[p]$  za 1 i poslanoj poruci pridruži izgrađeni vektor  $V_p$ .
- 4) Nakon primitka poruke od procesa  $p$  na procesu  $k$  uvećaj oznaku  $V_k[k]$  za 1. Za ostale oznake  $i \neq k$  postavi  $V_k[i] = V_p[i]$  ako je  $V_k[i] < V_p[i]$ .



# Primjena sata u raspodijeljenoj okolini

- Uređena razmjena poruka
  - Primjena skalarnih logičkih oznaka vremena
  - Svi procesi na isti način vide redoslijed događaja
- Održavanje konzistentnosti
  - Bez vremenskih oznaka nije moguće odrediti pravilni redoslijed akcija u vremenu



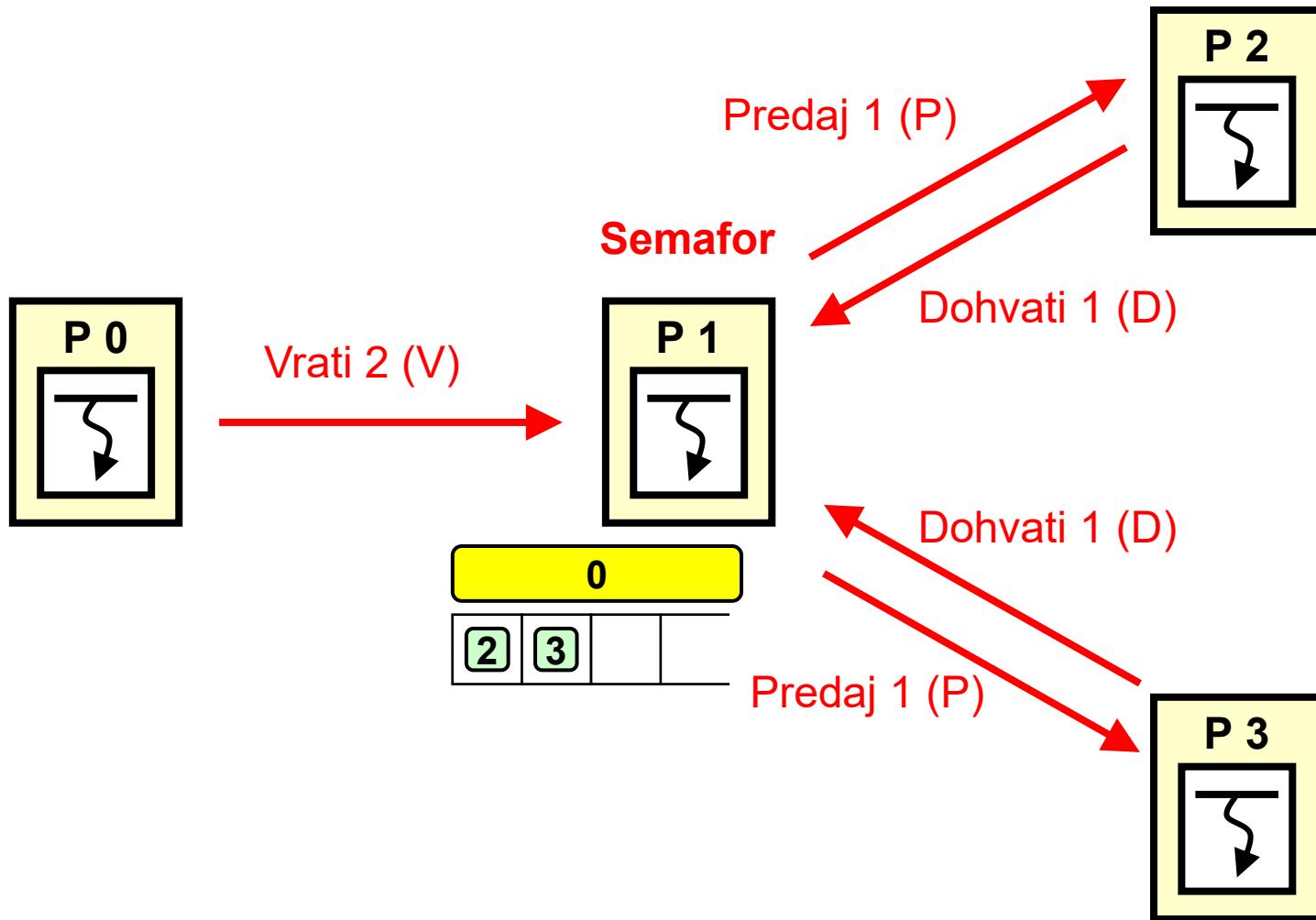
# Sadržaj predavanja

- Potreba za sinkronizacijom procesa
- Primjena sata u jednoprocesorskoj okolini
- Primjena sata raspodijeljenoj okolini
- Sinkronizacija tijeka izvođenja procesa
  - Primjena semafora u raspodijeljenoj okolini
  - Sinkronizacija zasnovana na razmjeni obavijesti
- Međusobno isključivanje procesa

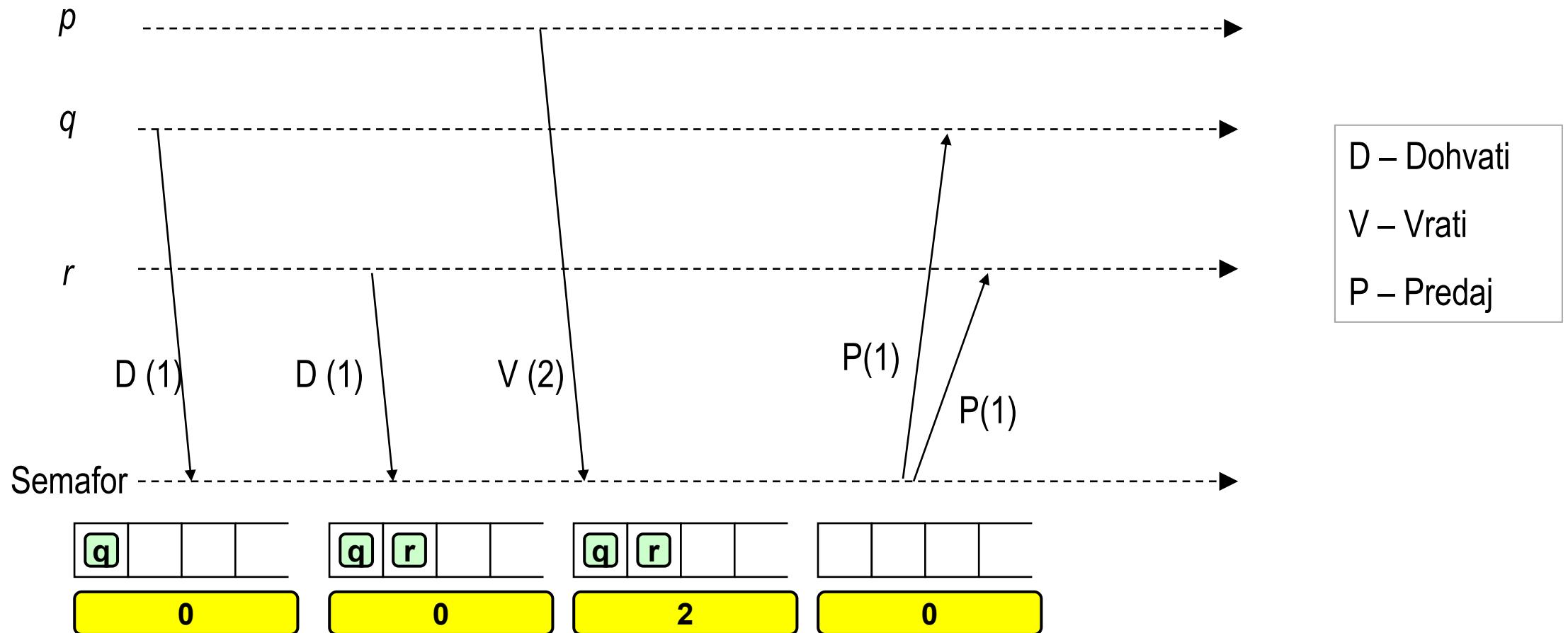
# Primjena semafora u raspodijeljenoj okolini

- **Semafor**
  - Proces koji u spremniku čuva  $N$  znački (*token*)
  - Rep čekanja zasnovan na posluživanju zahtjeva prema redoslijedu prispjeća (*FIFO*)
- **Korisnici**
  - Procesi šalju poruke *zahtjev za dohvat* (*D*)  $n$  znački
  - Ako u spremniku postoji traženi broj znački, proslijede se *potvrda za predaju* (*P*)
  - Ako u spremniku ne postoji traženi broj znački, zahtjev se stavlja u rep čekanja
  - Nakon završetka obrade, procesi vraćaju preuzete značke slanjem poruke *vrati* (*V*)

# Semafor u raspodijeljenoj okolini



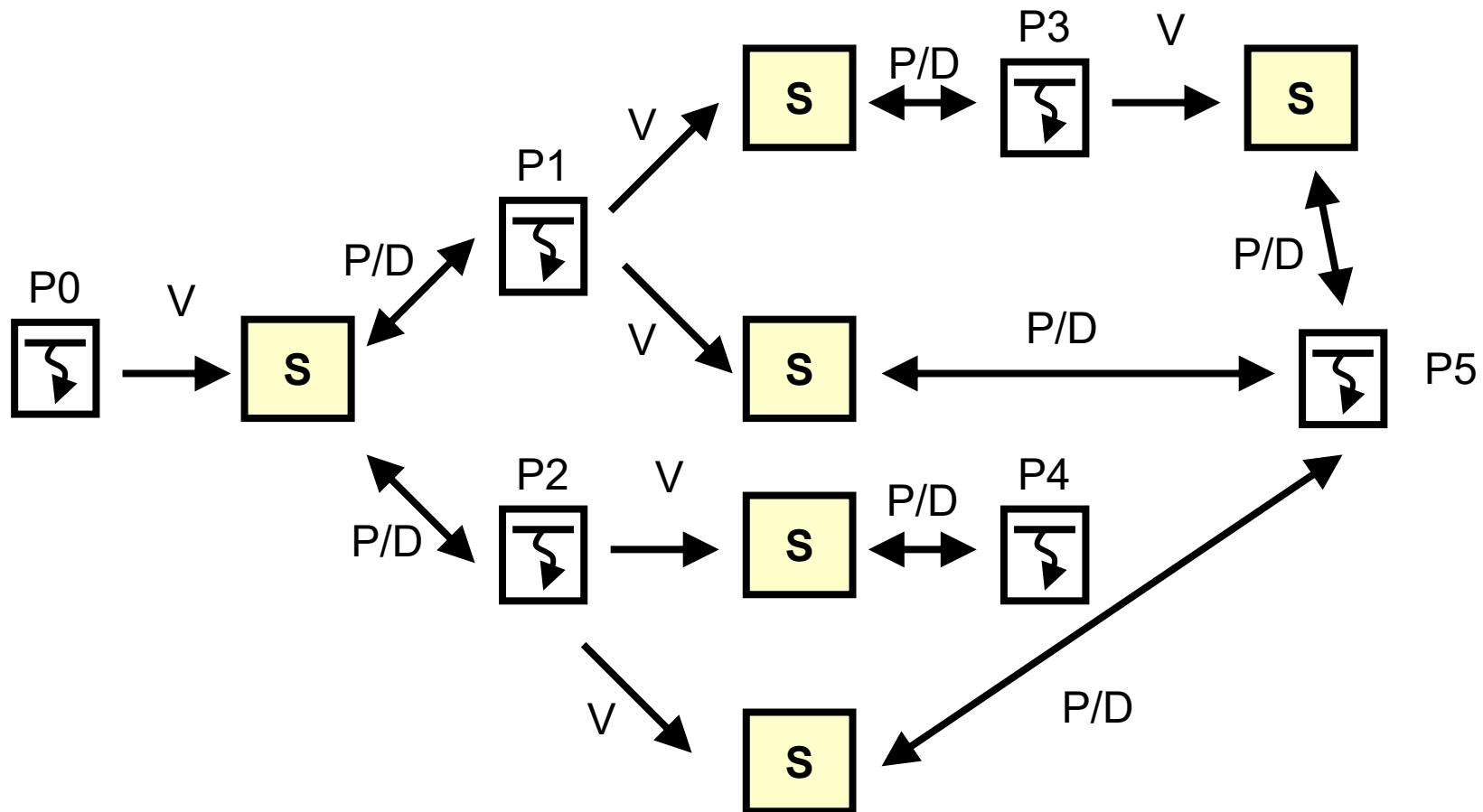
# Primjer sinkronizacije tijeka izvođenja



# Složeni obrasci sinkronizacije

- Semafor je osnovni element za ostvarivanje složenih obrazaca sinkronizacije
- Graf raspodijeljenog tijeka izvođenja procesa
  - Grananje tijeka izvođenja
  - Spajanje tijeka izvođenja
  - Ponavljanje tijeka izvođenja

# Graf raspodijeljenog tijeka izvođenja

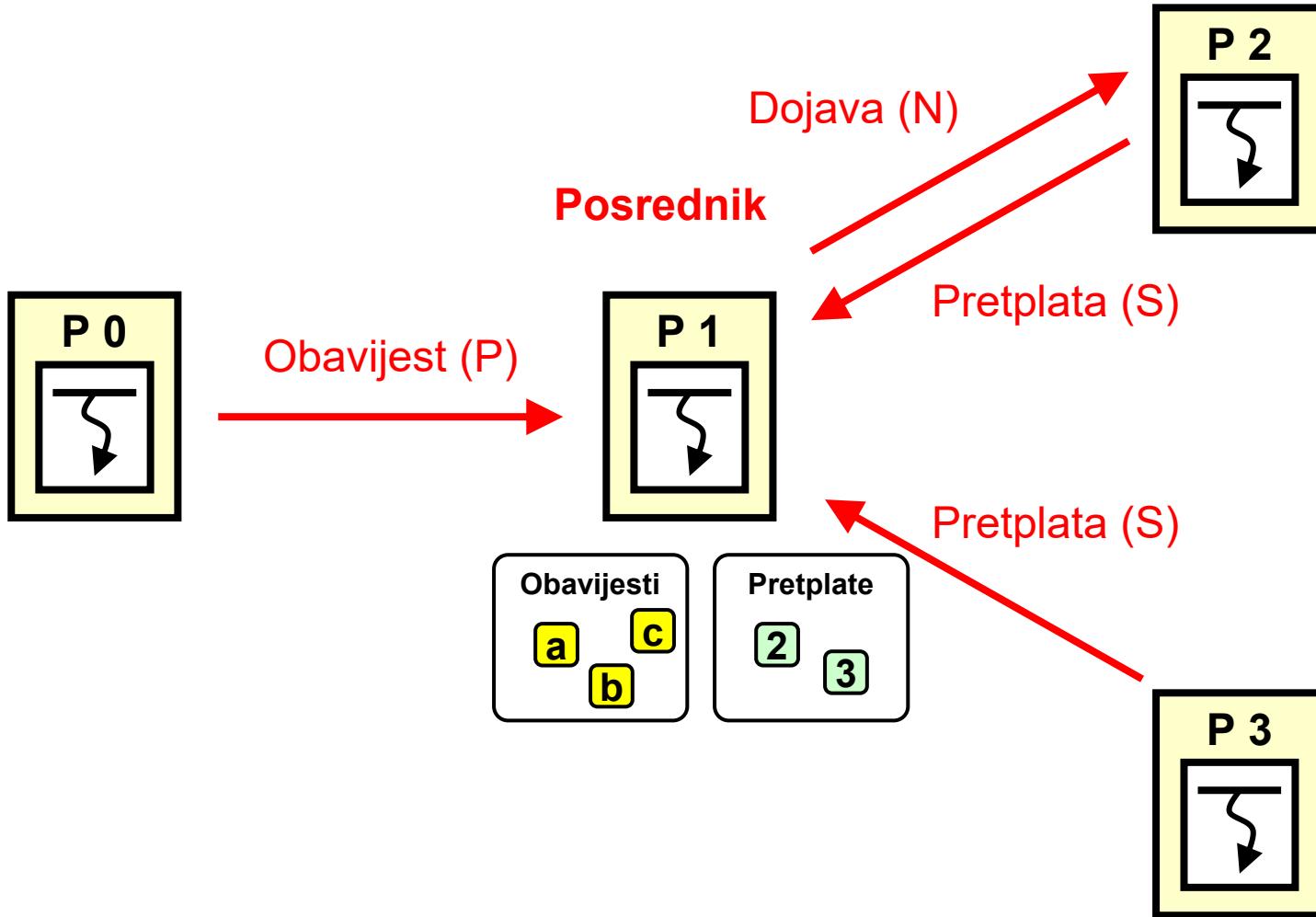


D – Dohvati  
V – Vrati  
P – Predaj

# Sinkronizacija razmjenom obavijesti

- **Posrednik**
  - Sadrži spremnik s obavijestima i spremnik pretplata na obavijesti
  - Ostvaruje postupak usporedbe obavijesti i pretplata prema modelu objavi - pretplati
- **Korisnici**
  - Procesi šalju posredniku pretplate (*S*)
  - Procesi šalju posredniku obavijesti (*P*)
  - Ako posrednik ima aktivnu pretplatu na obavijest, ona se prosljeđuje procesu pretplatniku u poruci dojave (*N*)

# Okolina posrednika obavijesti



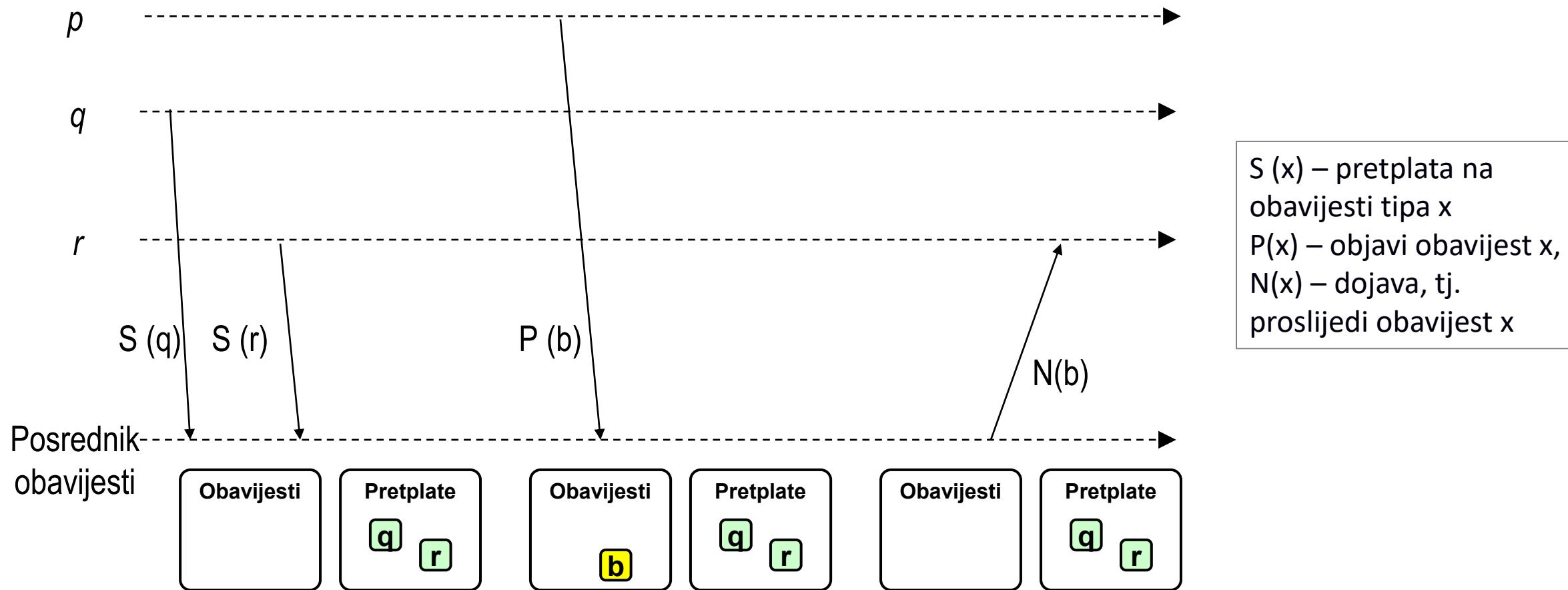
Npr:

Proces P2 se pretplatio (*subscribe*) na obavijesti tipa A

Proces P0 objavljuje (*publish*) obavijest a posredniku

Posrednik proslijeđuje obavijest (*notify*) a procesu P2

# Primjer sinkronizacije razmjenom obavijesti



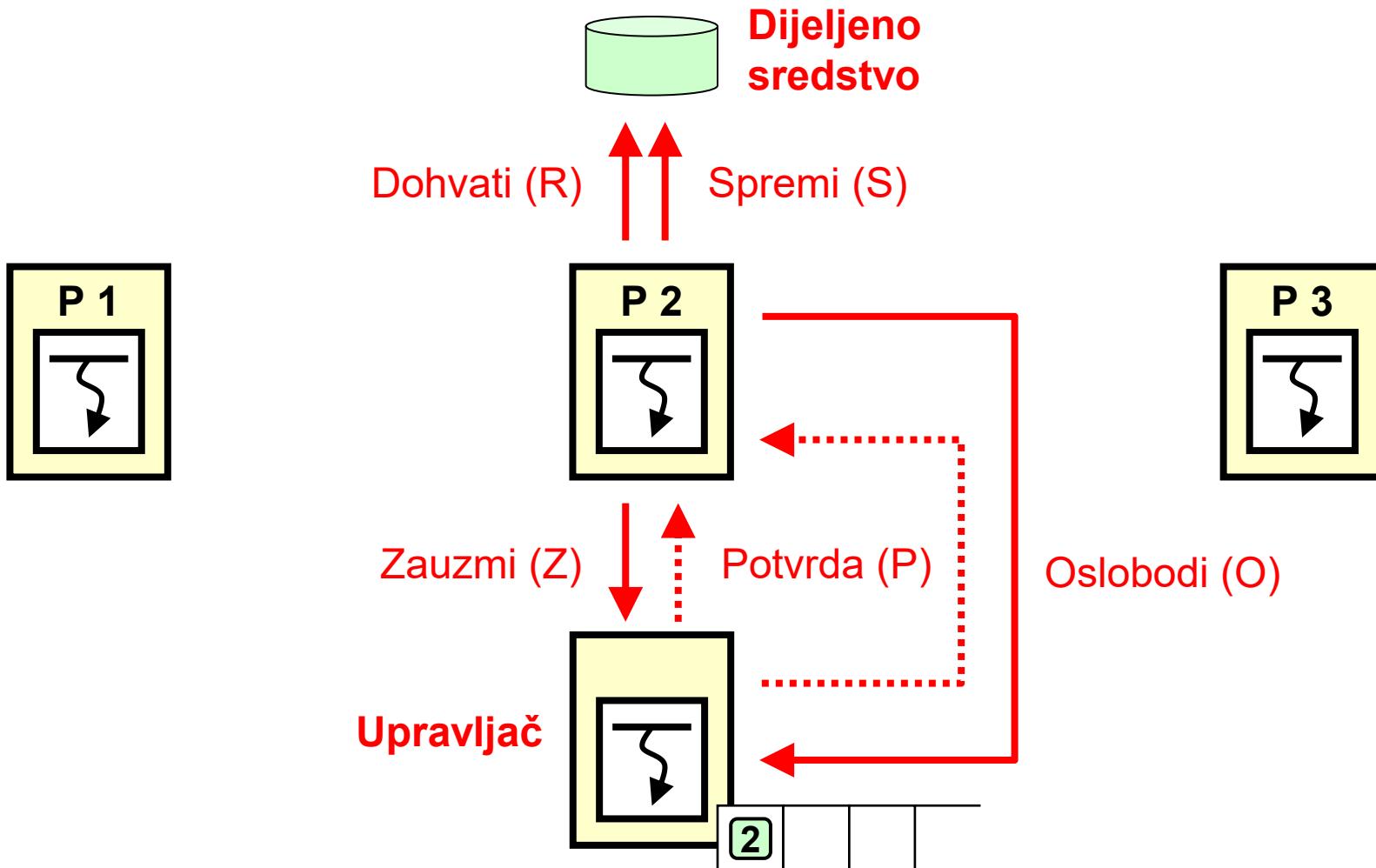
# Sadržaj predavanja

- Potreba za sinkronizacijom procesa
- Primjena sata u jednoprocesorskoj okolini
- Primjena sata raspodijeljenoj okolini
- Sinkronizacija tijeka izvođenja procesa
- Međusobno isključivanje procesa
  - Središnji upravljač s repom čekanja
  - Decentralizirano međusobno isključivanje
  - Isključivanje zasnovano na primjeni prstena

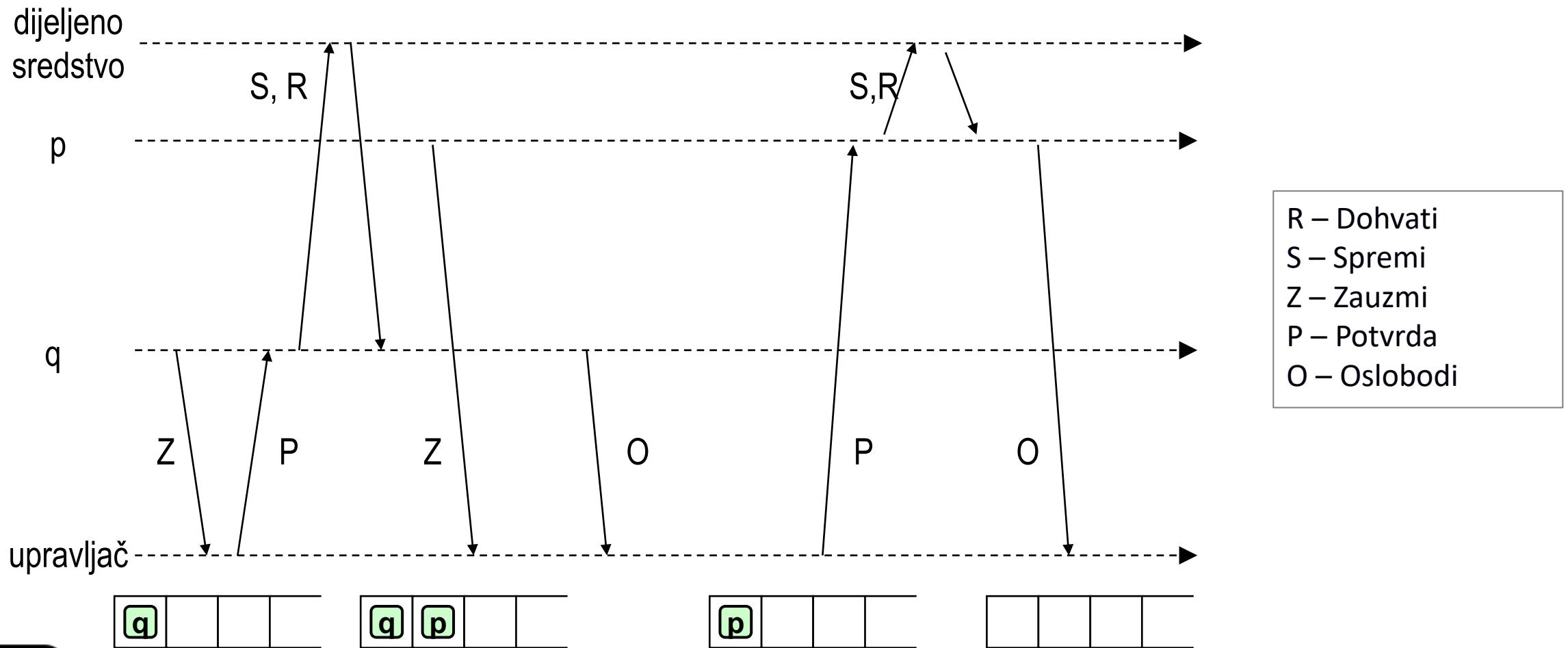
# Međusobno isključivanje procesa

- **Središnji upravljač s repom čekanja**
  - Proces koji čuva stanje repa čekanja
  - Rep čekanja zasnovan na posluživanju zahtjeva prema redoslijedu prispijeća (*FIFO*)
- **Korisnici**
  - Procesi šalju poruke sa *zahtjevom za zauzimanje* (*Z*) tj. pristup sredstvu
  - Procesi ostvaruju pristup sredstvu nakon primitka poruke *potvrde* (*P*), te dohvaćaju (*R*) i/ili spremaju (*S*) podatke na dijeljeno sredstvo
  - Nakon završetka obrade, procesi otpuštaju zauzeto sredstvo slanjem poruke *oslobodi* (*O*)

# Središnji upravljač s repom čekanja



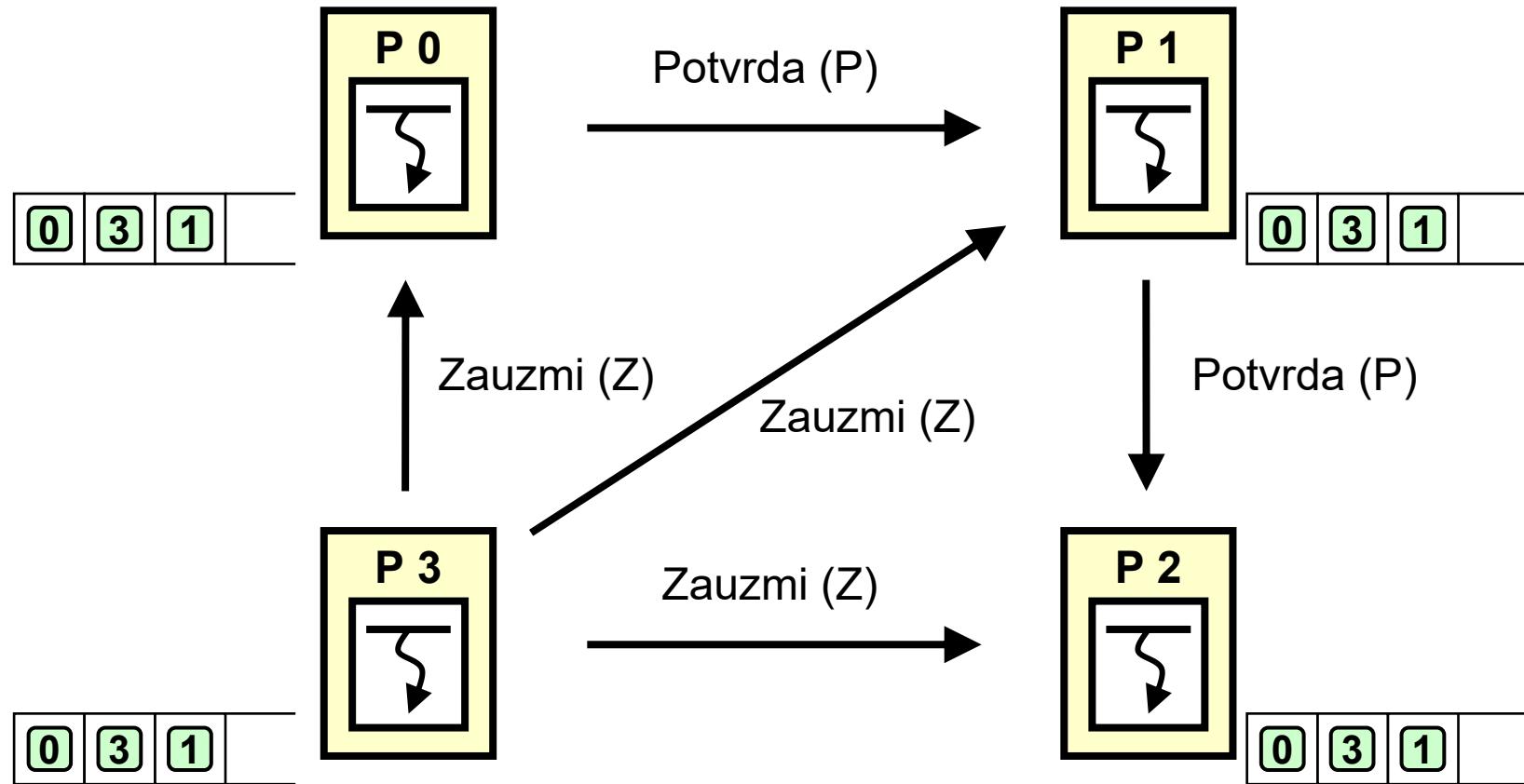
# Isključivanje putem središnjeg upravljača



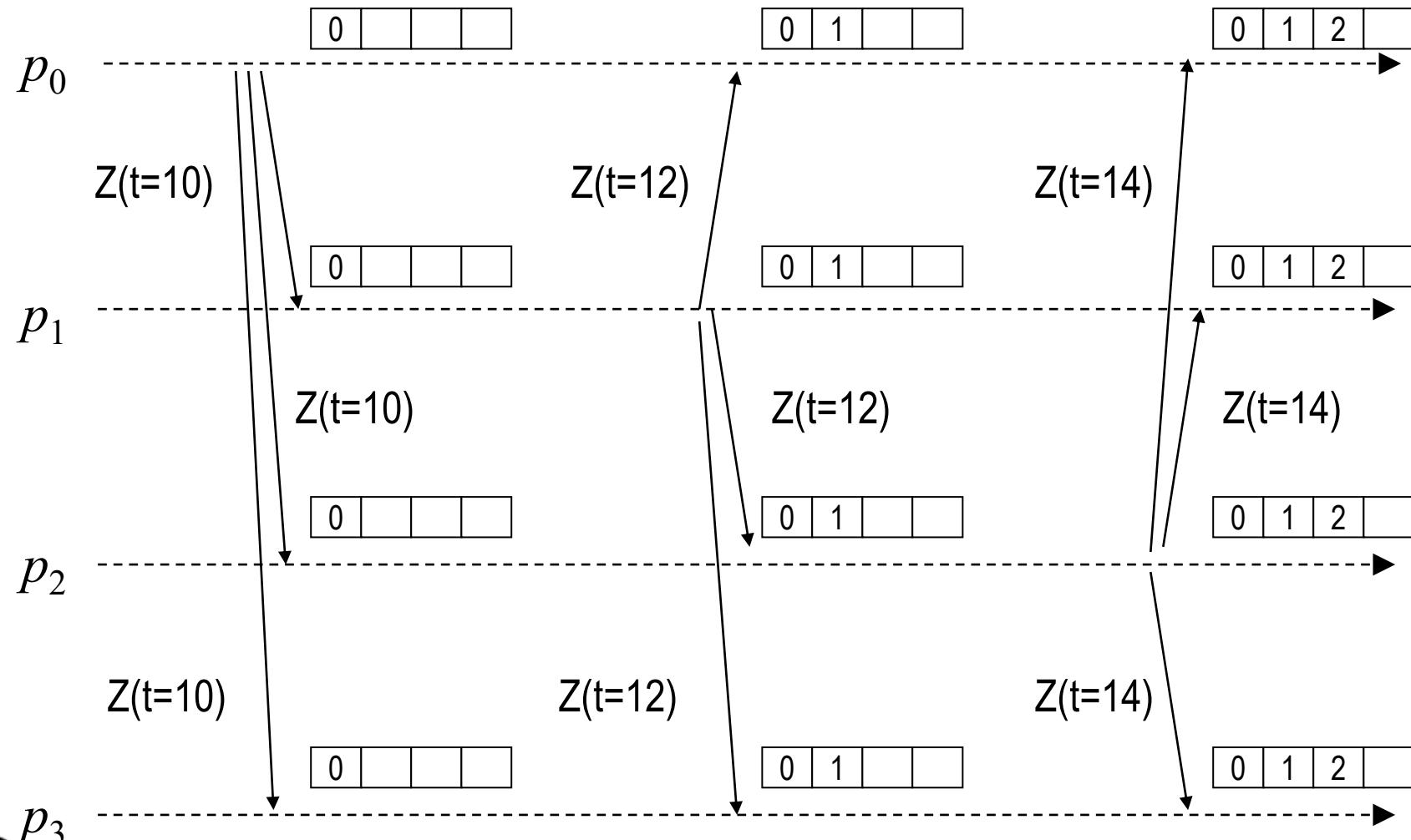
# Decentralizirano međusobno isključivanje

- **Raspodijeljeni rep čekanja**
  - Svaki proces ima lokalni rep čekanja
  - Procesi razmjenjuju informacije potrebne za usklađivanje stanja svih repova čekanja u sustavu
- **Pretpostavke**
  - Svaki proces ima lokalni satni mehanizam koji je usklađen s ostalim procesima
  - Svaki zahtjev za pristup sredstvu uključuje oznaku trenutka u kojem je proces uputio zahtjev
  - Procesi ostvaruju pristup u skladu s vremenskim oznakama upućivanja zahtjeva

# Elementi decentraliziranog isključivanja



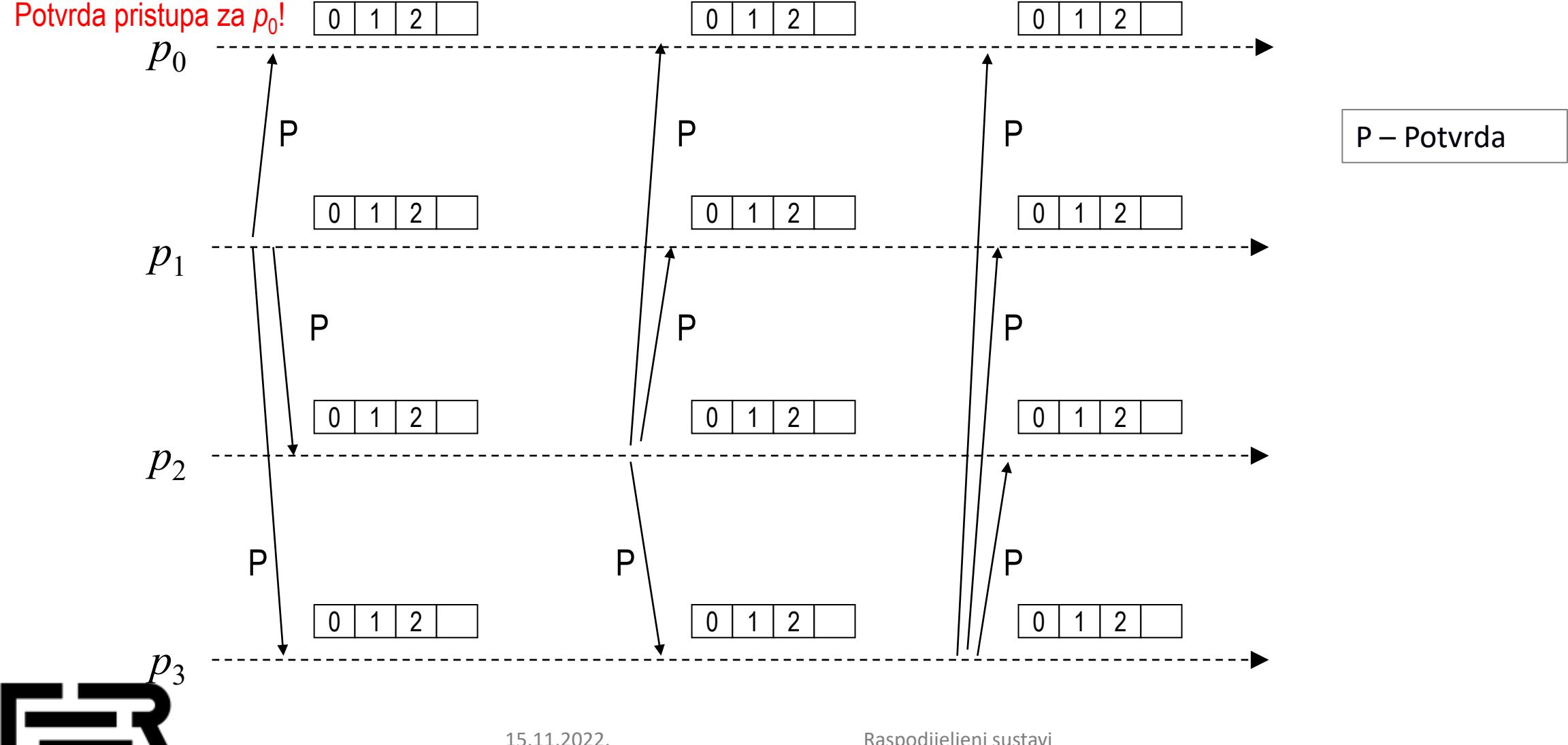
# Decentralizirano isključivanje (1/4)



$Z(t=x)$  – Zauzmi,  
vremenska oznaka x

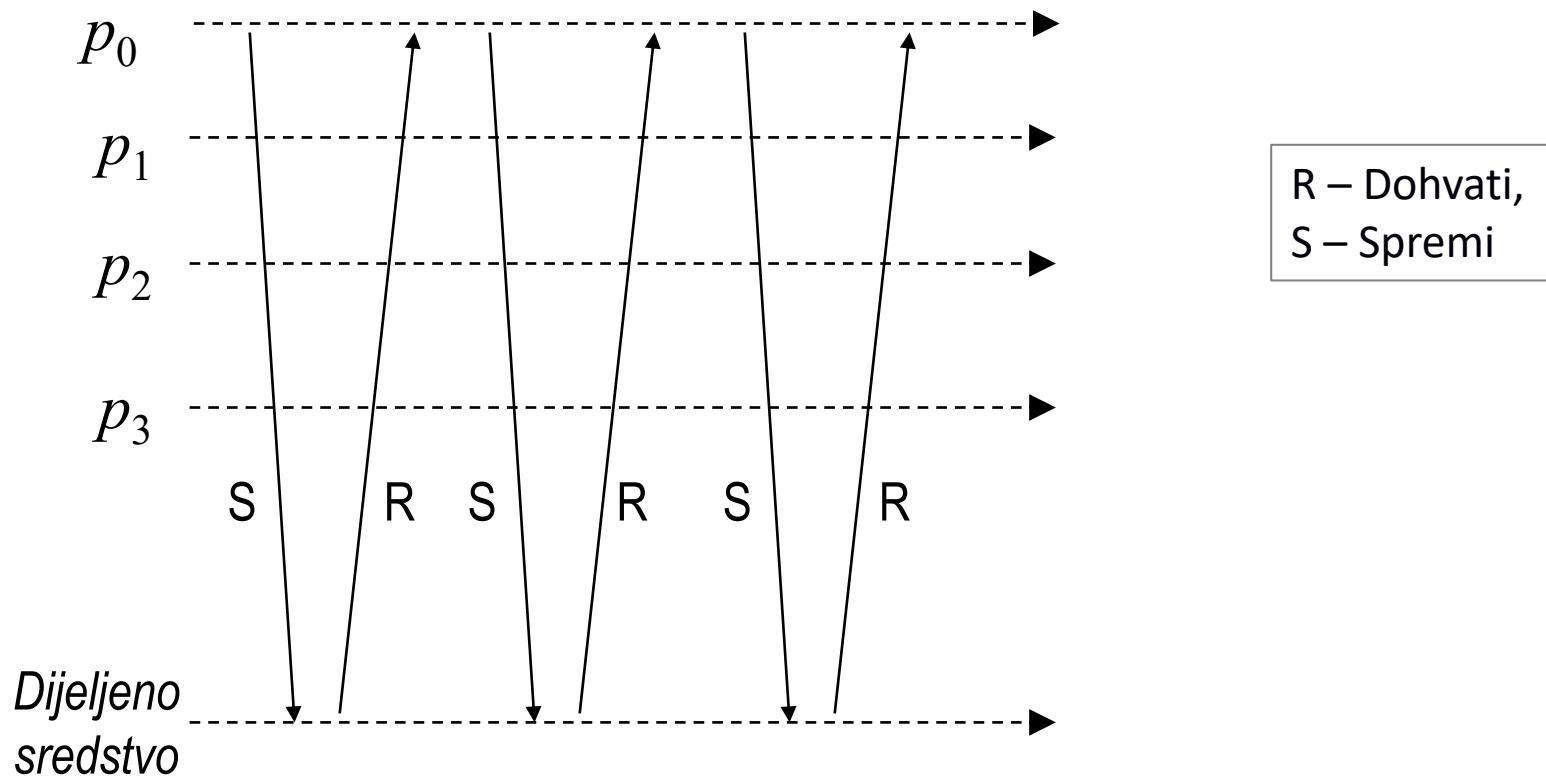
# Decentralizirano isključivanje (2/4)

Potvrda pristupa za  $p_0$ !



# Decentralizirano isključivanje (3/4)

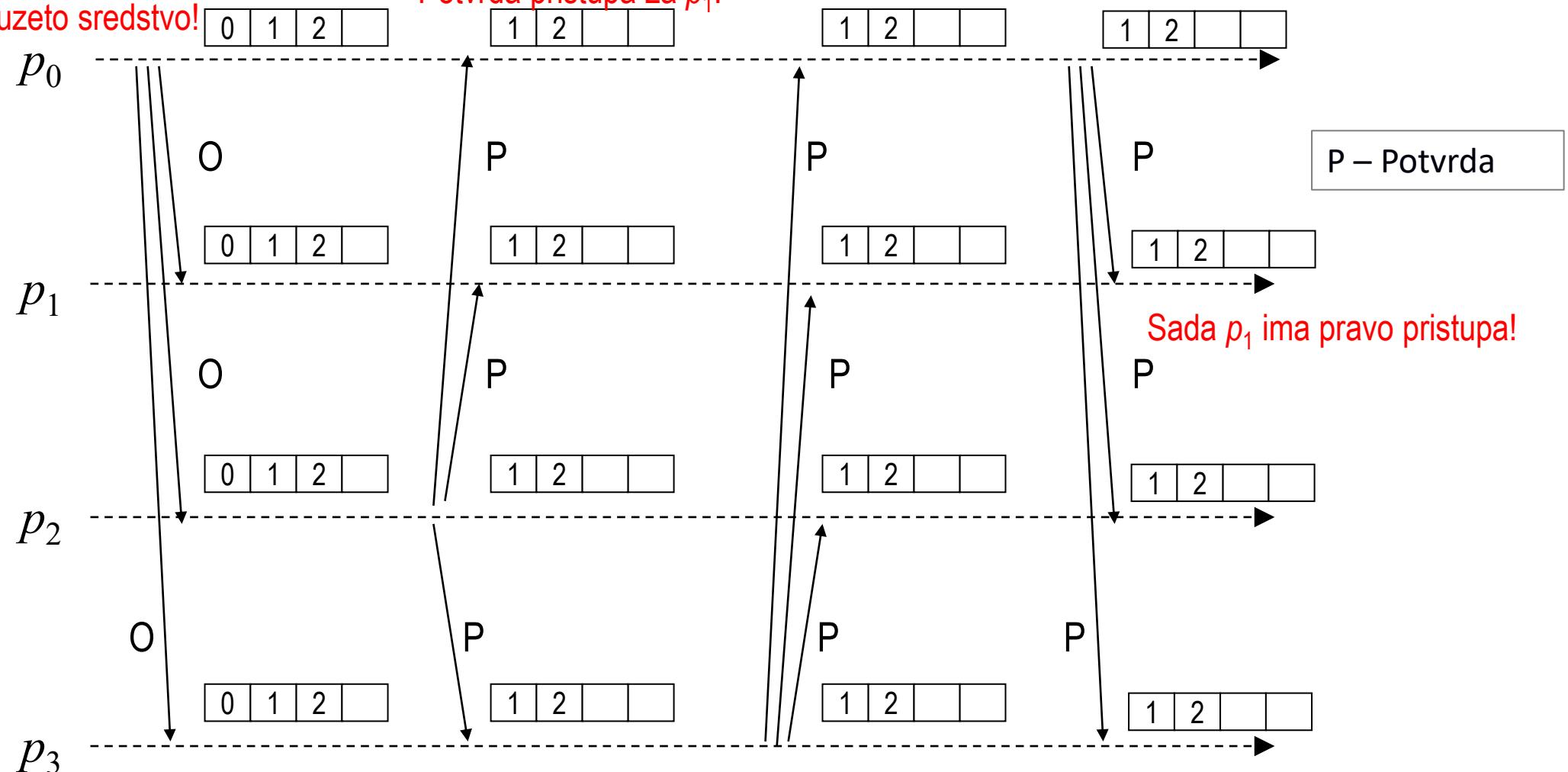
Pristup  
dijeljenom  
sredstvu!



# Decentralizirano isključivanje (4/4)

Otpusti zauzeto sredstvo!

Potvrda pristupa za  $p_1$ !



# Međusobno isključivanje primjenom prstena

- **Struktura prstena procesa**
  - Procesi su povezani u logičku mrežu zasnovanu na prstenu
  - Primjenjuju se identifikatori procesa za formiranje prstena
  - Duž prstena ostvaruje se razmjena jedne značke
  - I ovo je rješenje decentralizirano!
- **Pristup dijeljenom sredstvu**
  - Pristup ima samo proces koji u određenom trenutku ima značku
  - Nakon završetka pristupa, proces proslijeđuje značku susjednom procesu u prstenu

# Akcije procesa u prstenu (1)

## Proces $n$ prima značku

- 1) Čitanje podataka iz spremnika
- 2) Pisanje podataka u spremnik
- 3) Prosljeđivanje značke procesu (  $n-1$  )

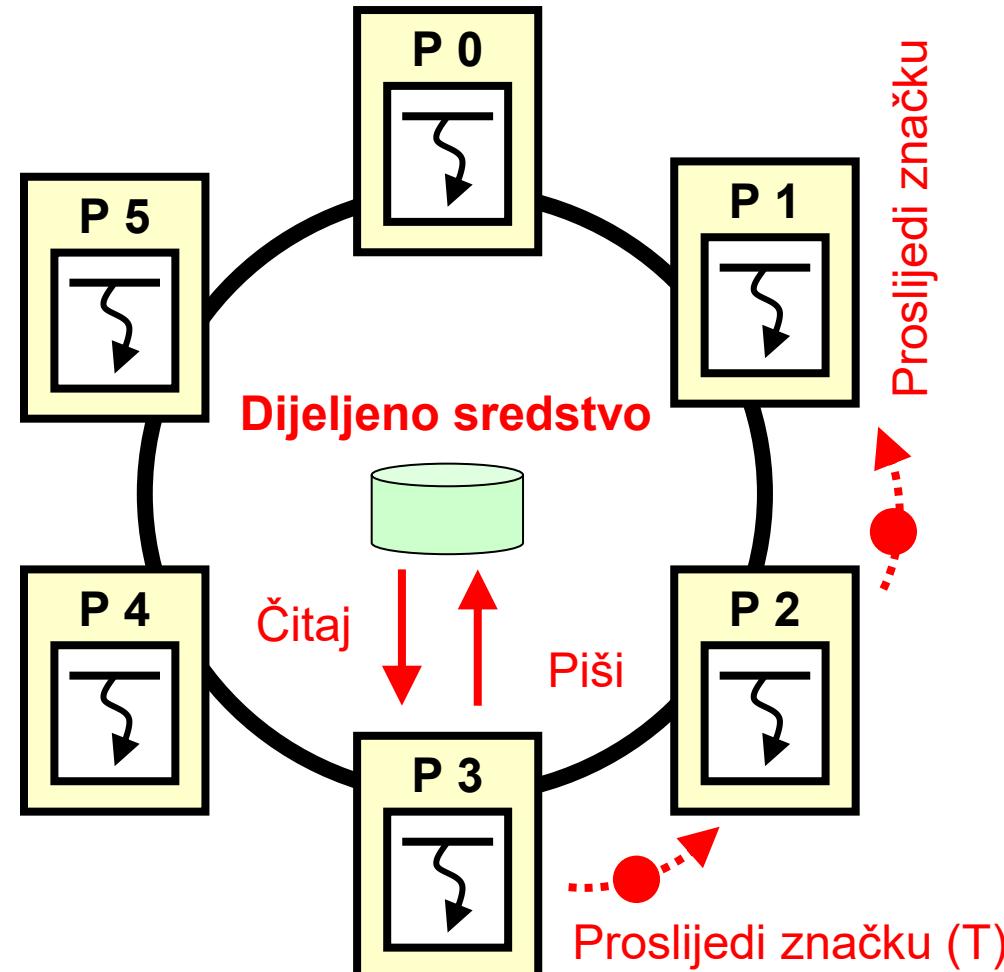
## Proces ( $n-1$ ) prima značku

- 4) Proces ne zahtjeva pristup spremniku
- 5) Prosljeđivanje značke procesu (  $n-2$  )

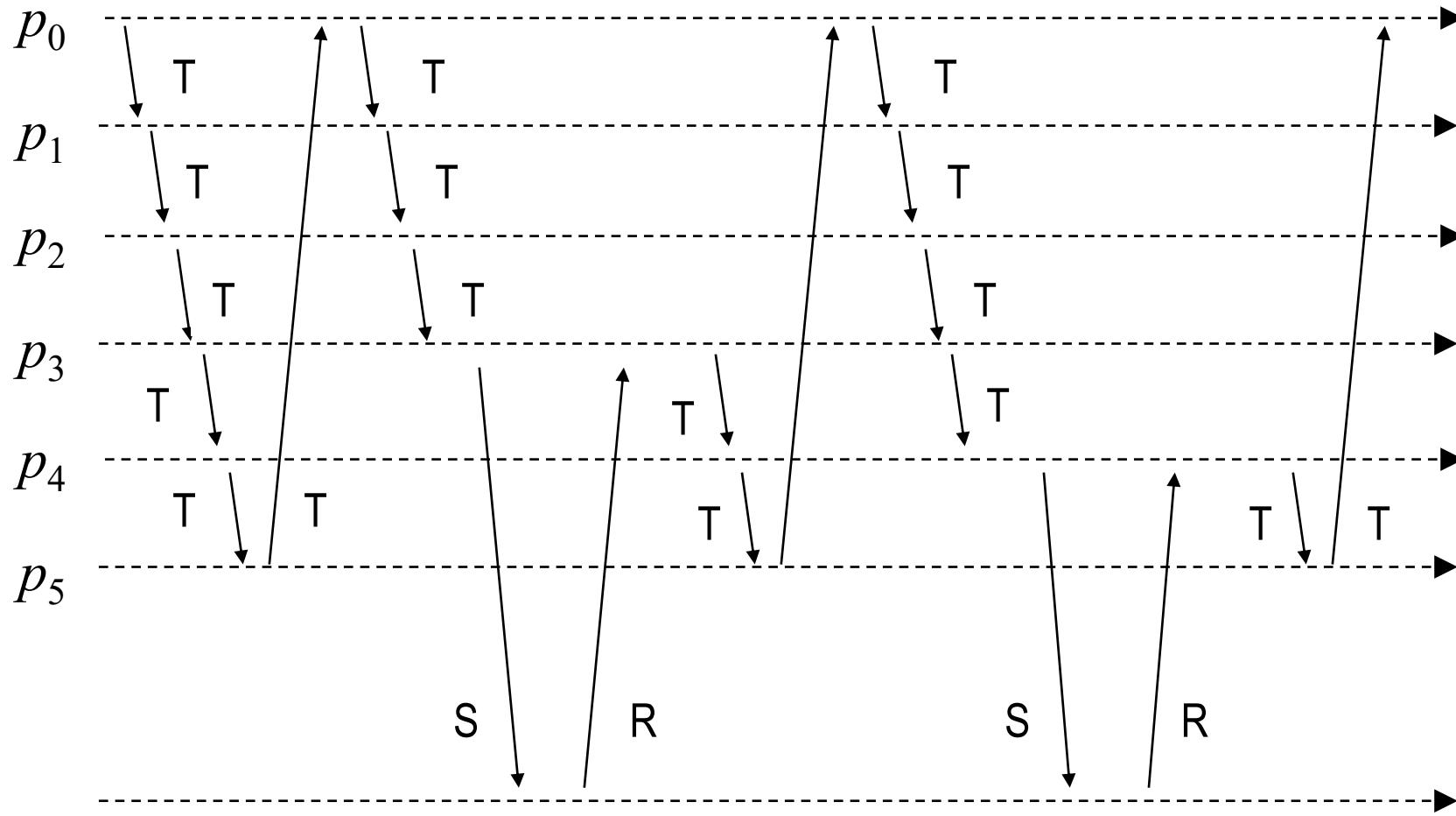
## Proces ( $n-2$ ) prima značku

- 6) ...

# Akcije procesa u prstenu (2)



# Akcije procesa u prstenu (3)



T – Prijenos tokena,  
S – Spremi,  
R – Dohvati

# Primjeri sustava za sinkronizaciju

## Usluga ZooKeeper

- Usluga za pouzdanu koordinaciju tijeka izvođenja skupa procesa u raspodijeljenoj okolini
- Dodatne informacije: <http://zookeeper.apache.org>

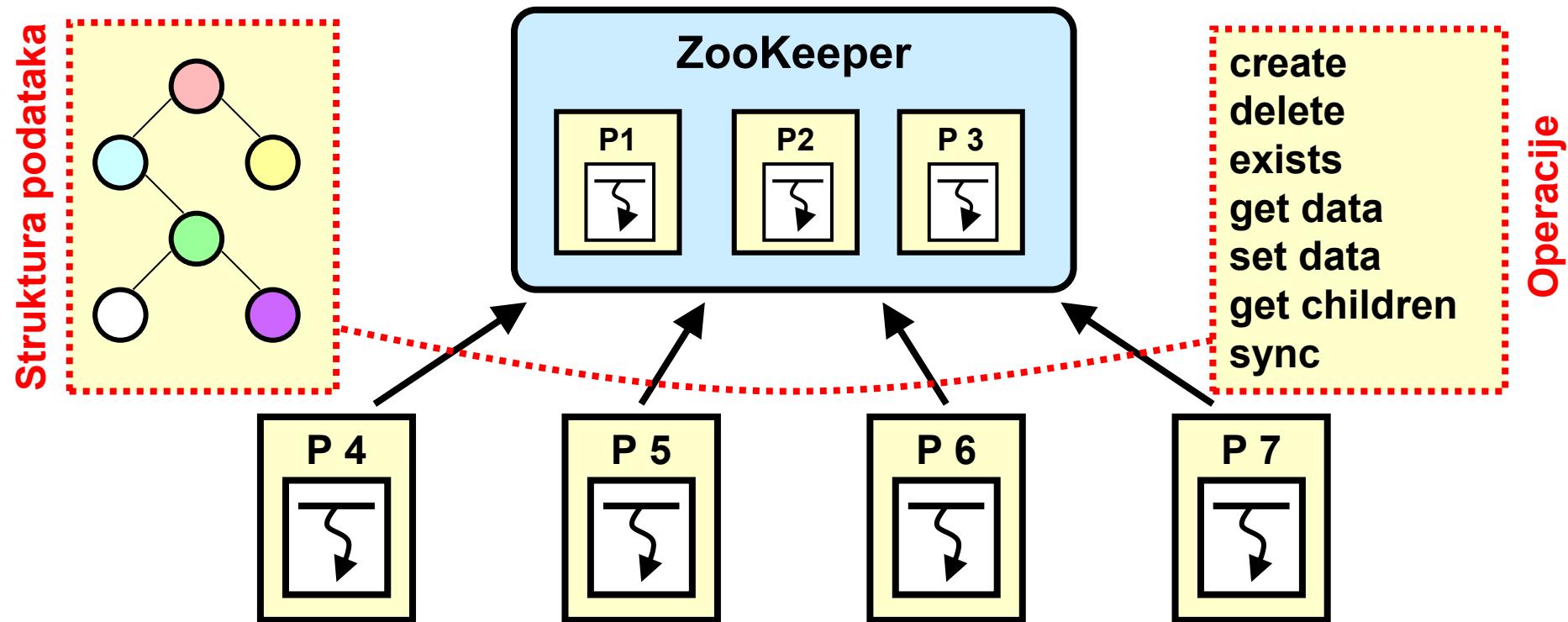
## Okružje Hadoop

- Programsко okružje za provođenje paralelne obrade velike količine podataka (*Big Data*)
- Postoji potreba za sinkronizacijom procesa *map* i *reduce*
- Dodatne informacije: <http://hadoop.apache.org>

# Usluga ZooKeeper (1)

Usluga opće namjene za koordiniranje skupa procesa u raspodijeljenoj okolini

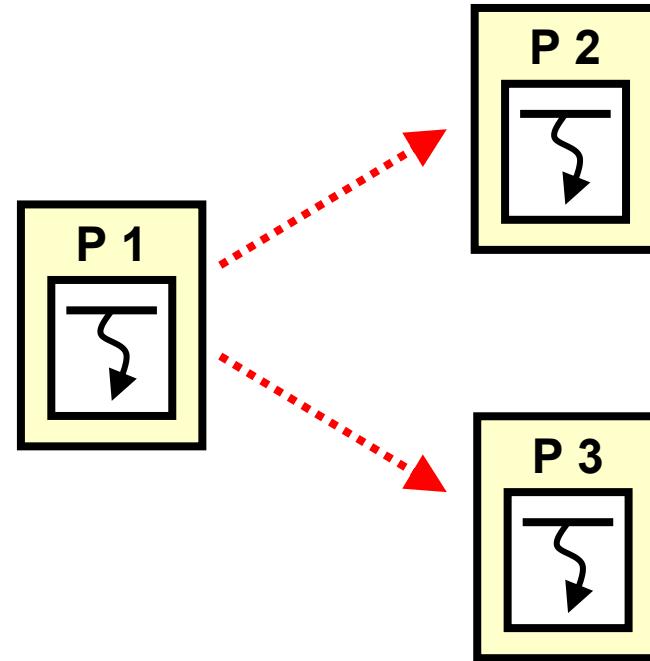
Imenovanje, sinkronizacija, upravljanje grupama, repovi, donošenje odluka, zaključavanje sredstava



# Usluga ZooKeeper (2)

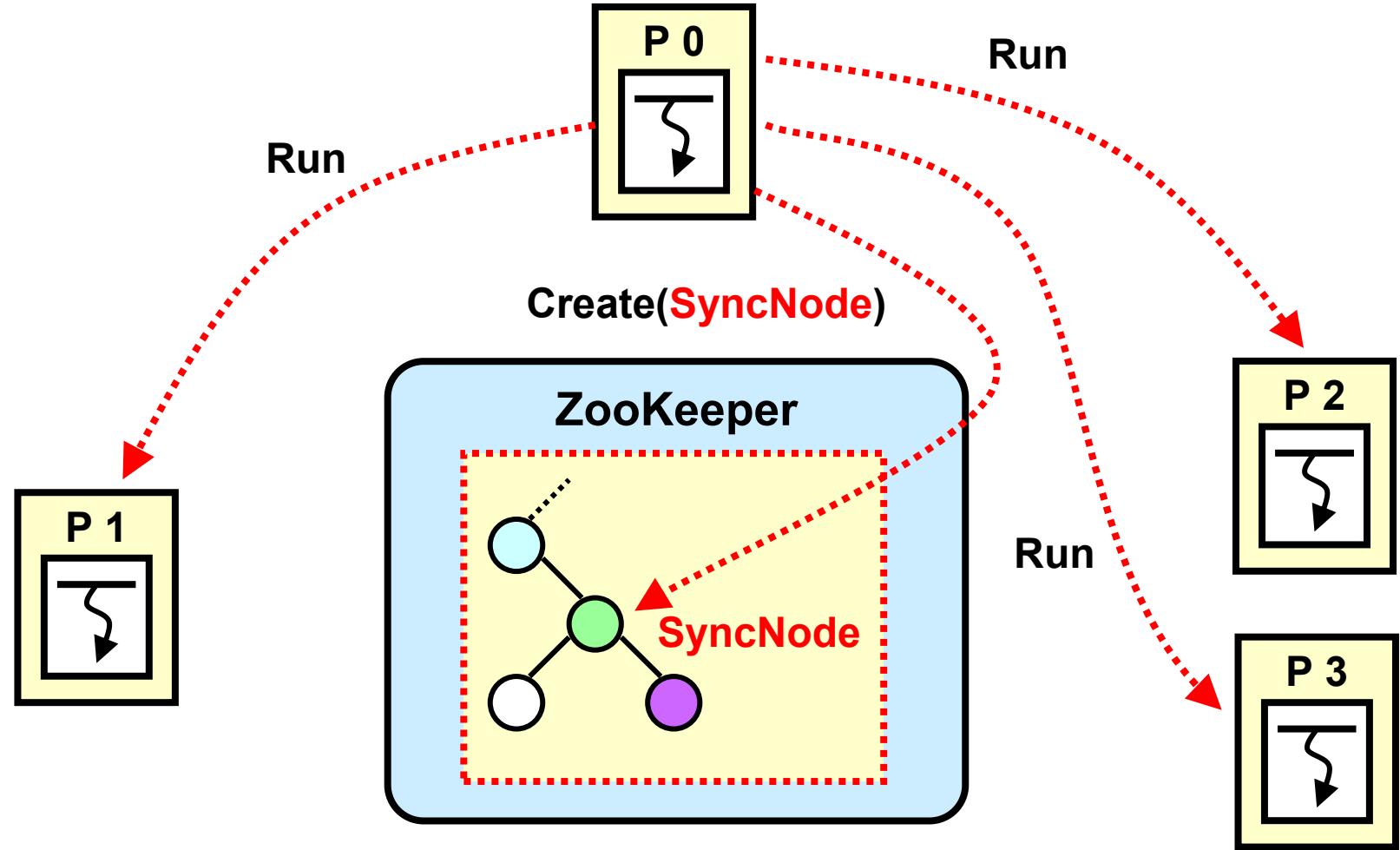
## Primjer: Sinkronizacija procesa

Procesi P2 i P3 započinju s izvođenjem tek nakon što je proces P1 završio s izvođenjem



# Usluga ZooKeeper (3)

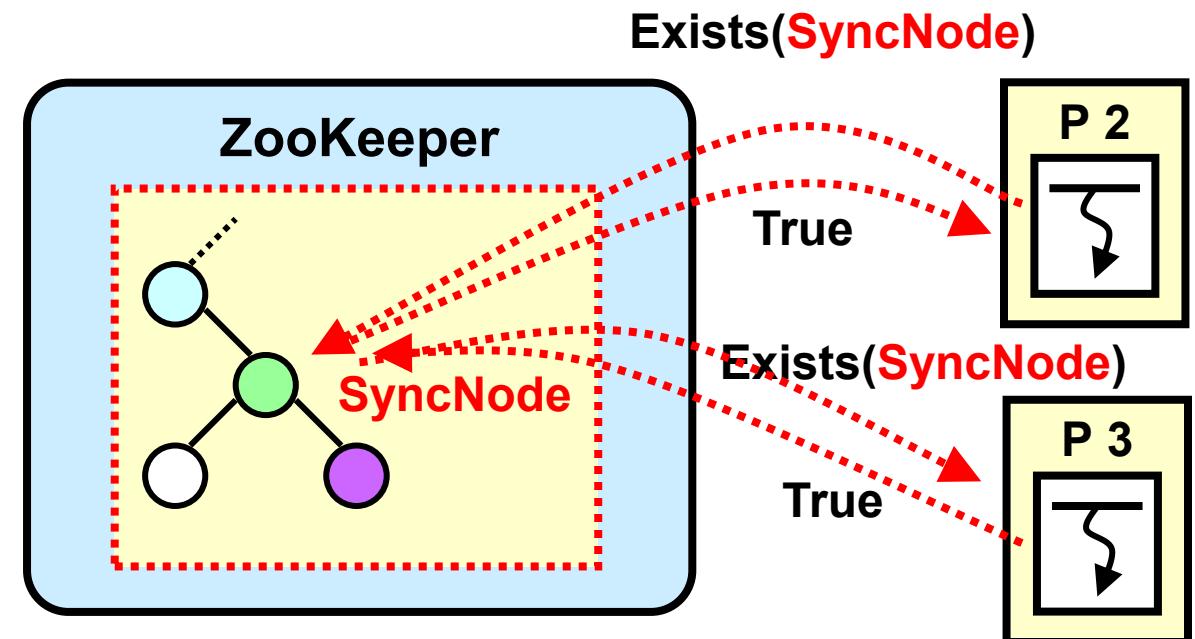
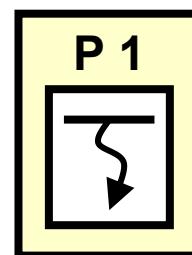
P0 je upravljački proces koji stvara sinkronizacijski čvor **SyncNode**



# Usluga ZooKeeper (4)

Procesi P2 i P3 ispituju postoji li sinkronizacijski čvor **SyncNode**

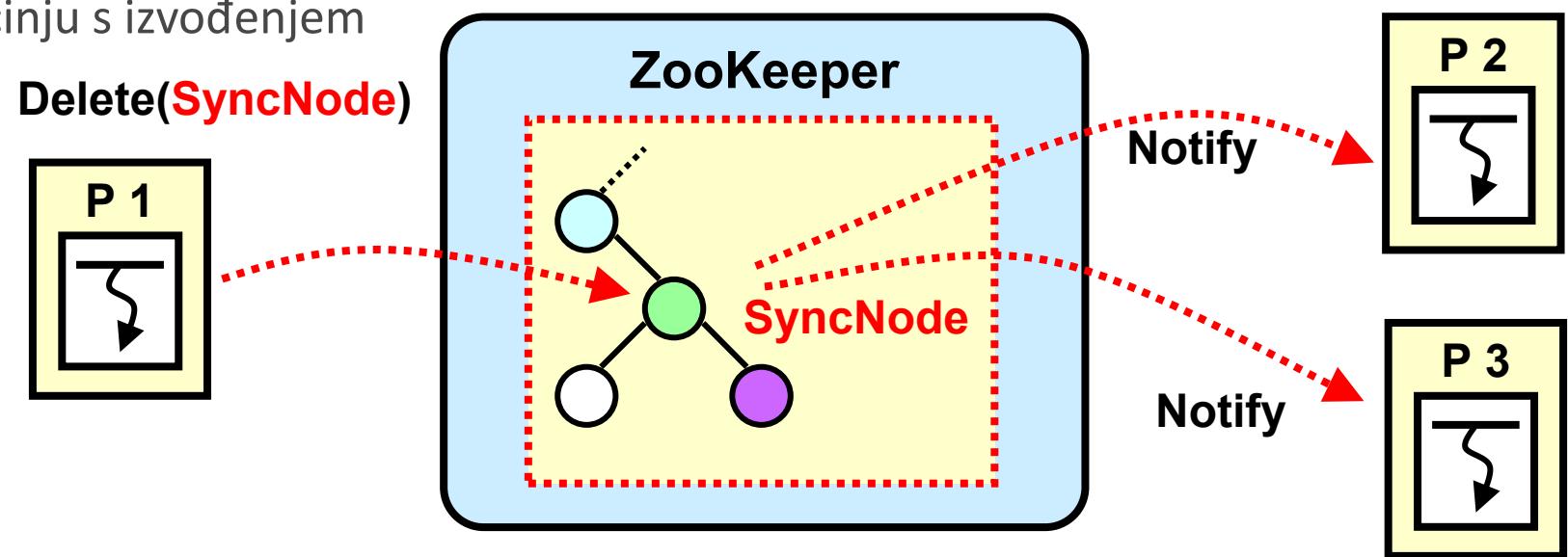
- Ako čvor postoji, čekaju na dojavu o brisanju čvora
- jer kada se čvor izbriše, oni započinju izvođenje



# Usluga ZooKeeper (5)

## Proces P1 završava s izvođenjem

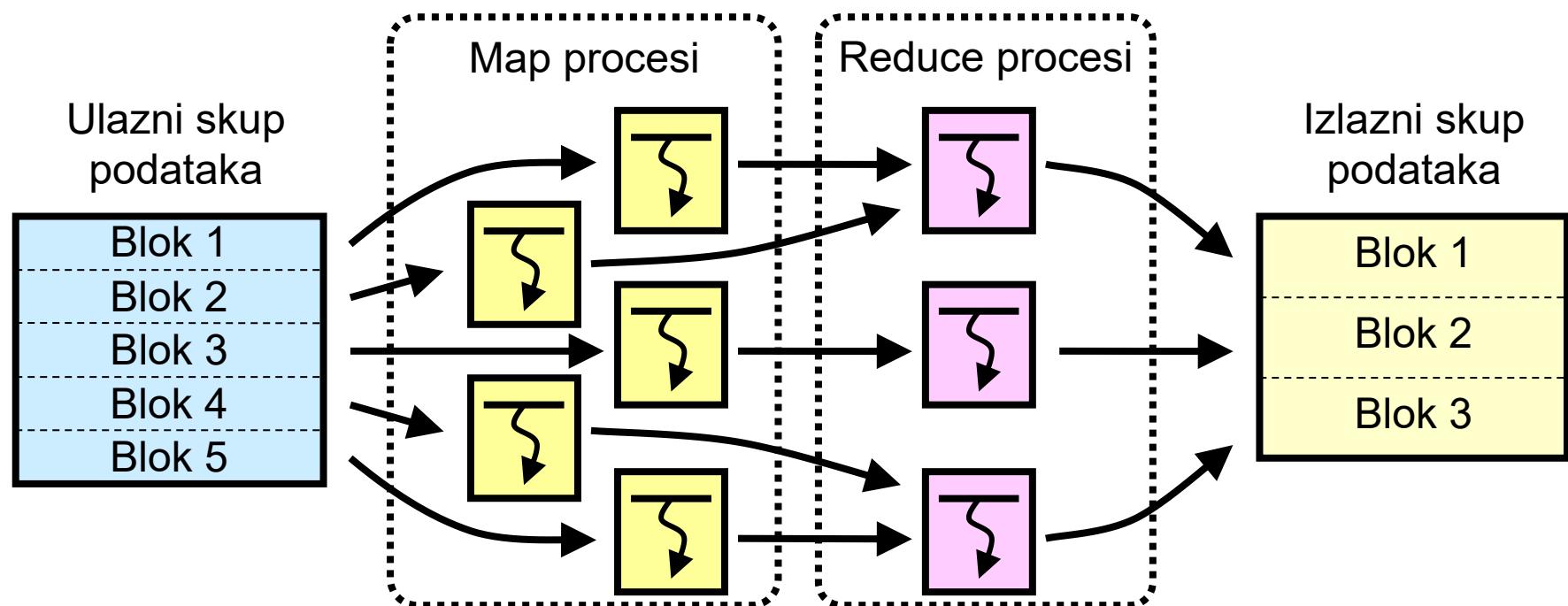
- Briše sinkronizacijski čvor **SyncNode**
- Usluga ZooKeeper dojavljuje brisanje čvora i stoga
- procesi **P2** i **P3** započinju s izvođenjem



# Okružje Hadoop

## MapReduce: analiza i obrada velikih skupova podataka

- Ulazni skup podataka dijeli se na blokove koje paralelno obrađuju nezavisni procesi
- Rezultati obrade grupiraju se u odredišni skup podataka



# Literatura

1. Maarten van Steen, Andrew S. Tanenbaum (2017.), *Distributed Systems 3<sup>rd</sup> edition*, Createspace Independent Publishing Platform poglavlje 14 (bez 14.5 i 14.6)
2. H. Attya, J. Welch: “**Distributed Computing: Fundamentals, Simulations, and Advanced Topics**”, Wiley, 2004. (Poglavlje: *Causality and Time*)
3. N. A. Lynch: “**Distributed Algorithms**”, Morgan Kaufmann, 1997. (Poglavlje: *Logical Time*)

# Dodatne informacije

- Kolegij na FER-u
  - **Raspodijeljena obrada velikih podataka, 3. semestar**
  - <https://www.fer.unizg.hr/predmet/rovp>
  - Sadržaj kolegija
    - Obrada velike količine podataka: MapReduce
    - Obrada nestrukturiranog teksta
    - Obrada toka podataka