9 - Klijent server arhitekture

S A D R Ž A J

9.1 Klijent server arihtekture

- Izbor tipa arhitekture vrlo je važan kod jednog klijent server sistema.
- Glavni zadaci jedne aplikacije za krajnjeg korisnika – klijenta mogu se predstaviti kroz sledeće tri komponente:
    1. **prikupljanje i čuvanje** podataka-informacija,
    2. **obrada** tih podataka,
    3. njihovo adekvatno **predstavljanje**.
- Način na koji su ove komponente obrađene, podela na aplikacije koje ih obrađuju, kao i mesto obrade u mrežnom sistemu, tačno definišu i određuju tip klijent serverske arihitekture
- Izbor odgovarajuće arhitekture zavisi i od: broja korisnika i računara na mreži, vrste razvojnih okruženja i programskih alata, modela i obima baze podataka kao i složenosti programskih procedura.
- Postoji više načina kako se ta podela vrši: dvoslojna (***two-tier***), troslojna (***three-tier***) arhitektura i višeslojna (***multi-tier***).
- Zadnjih godina zbog velikog povećanja broja računara kao i zahteva koji se postavljaju klijent server tehnologiji sve više su prisutne **višeslojne arhitekture** povezivanja klijenata i servera

9.1 Klijent server arihtekture

- Izbor arhitekture u mnogome zavisi i od vrste aplikacije koja povezuje klijente i servere
- Sve aplikacije mogu se podeliti (jednostavna apstrakcija) na 4 dela:

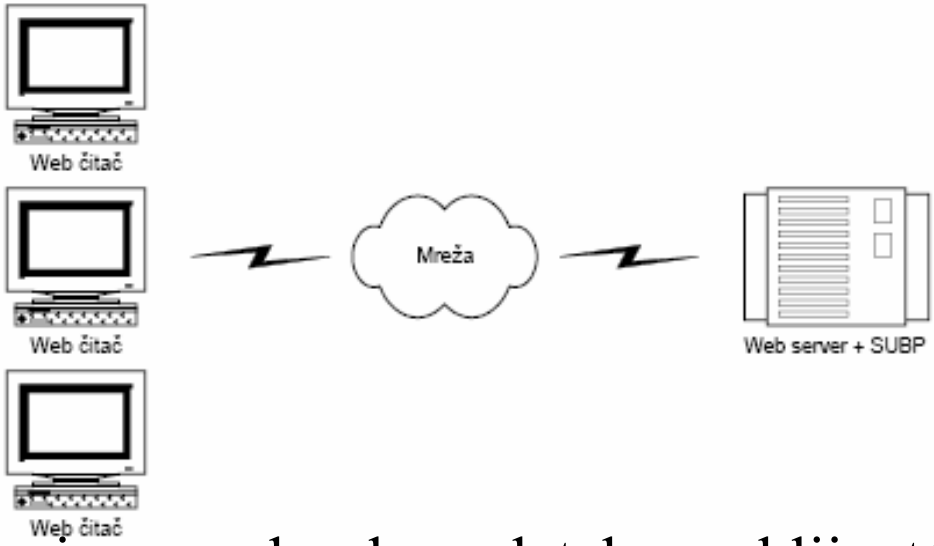
1. **Smeštanje podataka** (***data storage***) – većina informacionih sistema zahteva smeštanje podataka u datoteci ili u složenoj bazi podataka. Podaci su definisani u modelu podataka ***Entity Relationship Diagram***

2. **Pristup podacima** (***data access logic***) – deo aplikacije kojim se podaci upisuju i pretražuju, najčešće SQL upitima iz relacionih baza podataka.

3. **Logika aplikacije** (***application logic***) – deo aplikacije koji izvodi funkcionalnosti definisane u procesnom modelu ***Data Flow Diagram***, koje se odnose u slučajevima korišćenja i funkcionalnim zahtevima.

4. **Prezentacijska logika** (***presentation logic***) – deo aplikacije koji pruža podatke korisniku i od korisnika prihvata ulazne informacije i naredbe.

9.1 Klijent server arihtekture



- Kod klasičnih sistemima za obradu podataka po klijent/server modelu, mogu uočiti tri osnovne klase komponenti i to:

1. **server** – ima zadatak da optimalno upravljanja zajedničkim resursima što su najčešće podaci, da upravlja bazom podataka kojoj pristupa više korisnika, da kontroliše pristup i bezbednost podataka i da omogući centralizovano obezbeđenje integriteta podataka za sve aplikacije
2. **klijent** - vrši upravljanje korisničkim interfejsom i izvršava deo logike aplikacije.
3. **mreža** – ili komunikacioni posrednik koji ima zadatak da omogući prenos podataka između klijenta i servera.

9.2 Dvoslojna Arhitektura

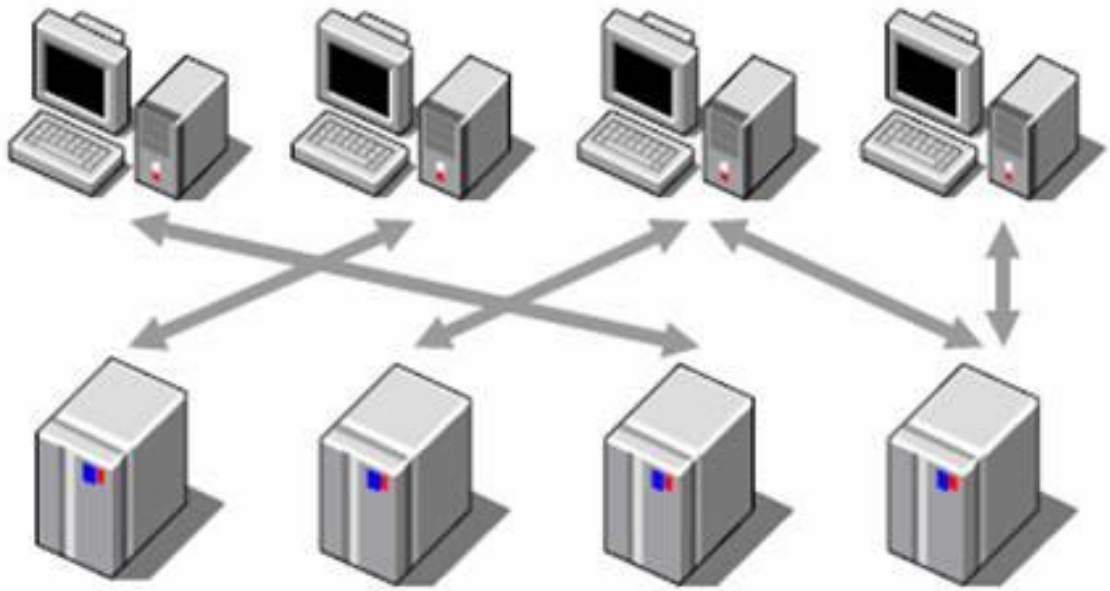
- Početak razvoja jedne ovakve arhitekture smatra se **1980** godina.
- Predstavlja prirodni nastavak dotadašnje ***file-server*** arhitekture.
- **Prednosti** koje su je izdvajale od klasične file-server aritekture bile su:
  - ✓ **Upotrebljivost** (*Usability*) - korišćenjem različitih formi za korisnički interfejs moguće je iste podatke prikazati na više različitih načina.
  - ✓ **Skalabilnost** (*Scalability*) - za razliku od fajl-server sistema gde su istovremeno mogli da rade 10 korisnika a da se performanse sistema drastično ne naruše, ovde je taj broj znatno veći, oko **100** korisnika.
  - ✓ **Fleksibilnost** (*Flexibility*) - mogućnost da podaci budu deljivi za veliki broj korisnika koji rade na heterogenim računarskim sistemima.
- Osnovne komponente svake dvoslojne klijent server arhitekture su:
  1. Korisnički interfejs (*User System Interface*) - način predstavljanja podataka, sesije, unos teksta, dijaloški prozori, prikaz na ekranu.
  2. Upravljanje procesima (*Processing Management*) - obrada podataka, generisanje, izvođenje i nadgledanje procesa i neophodnih resursa
  3. Upravljanje podacima (*DataBase Management*) - servisi vezani za čuvanje i deljenje podataka i datoteka

9.2 Dvoslojna Arhitektura

DVA SLOJA

Korisnički interfejs
+deo za upravljanje procesima

Upravljanje podacima (DBM)
+deo za upravljanje procesima



9.2 Dvoslojna Arhitektura

- Sve tri komponente koje su bitne za jednu aplikaciju (podaci, njihova obrada i predstavljanje) podeljene su između dva softverska entiteta (*tiers*): klijent aplikacija i servera baze podataka.
- Robustan jezik za razvoj klijentske aplikacije kao i prilagodljivi mehanizam za prenos klijentskih upita ka serveru su 2 najvažnije stvari
- Predstavljanje podataka se isključivo vrši na klijentskoj strani
- Njihova obrada se deli između klijenta i servera.
- Pristup podacima kao njihovo čuvanje je na serveru.
- Najveću odgovornost za funkcionalnost jedne aplikacije na sebe preuzima klijent jer se na njemu odvija najveći deo te aplikacije.
- Od njega i polaze svi zahtevi za nekim podacima i kod njega se završavaju svi podaci koji se prikazuju u različitim formama prikaza.
- Serverska strana je odgovorna za integritet i ispravnost tih podataka, prihvatanje zahteva putem redova (*query capabilities*) i pravovremeno rešavanje i slanje odgovora na te zahteve.
- Jedan od najraširenijih jezika koji je danas postao gotovo standard u slanju zahteva za nekim podacima je **SQL-jezik**.

9.2 Dvoslojna Arhitektura

- Slanje SQL zahteva od strane klijenta ka serveru zahteva čvrstu povezanost između dva sloja koji čine dvoslojnu arhitekturu.
- Da bi poslao jedan SQL zahtev klijent mora dobro da poznaje sintaksu serverske komponente ili pak da se putem odgovarajućeg API-a taj zahtev prevede da bi postao razumljiv serverskoj strani.
- Takođe klijent mora da zna **lokaciju servera**, kako su podaci na njemu **organizovani**, kao i **nazive** tih podataka.
- Upućeni zahtevi mogu da iskoriste prednosti logike koja je upamćena i koja se izvršava na serveru a koja centralizuje globalne zadatke koji se izvršavaju na serveru kao što su: validacija podataka, njihov integritet kao i sigurnost tih podataka.
- Traženi podaci se vraćaju klijentu i sada je na njemu kako će ih on prikazati, da li će izvršiti novu selekciju ili izvršiti neke nove operacije
- Velika prednost dvoslojne arhitekture je brzina razvoja nove aplikacije.
- Veliki razvoj PC baziranih rutina omogućio je klijentu da veoma lako razvije novu aplikaciju koja će koristiti veliki broj već razvijenih rutina za pozivanje podataka i njihovo predstavljanje

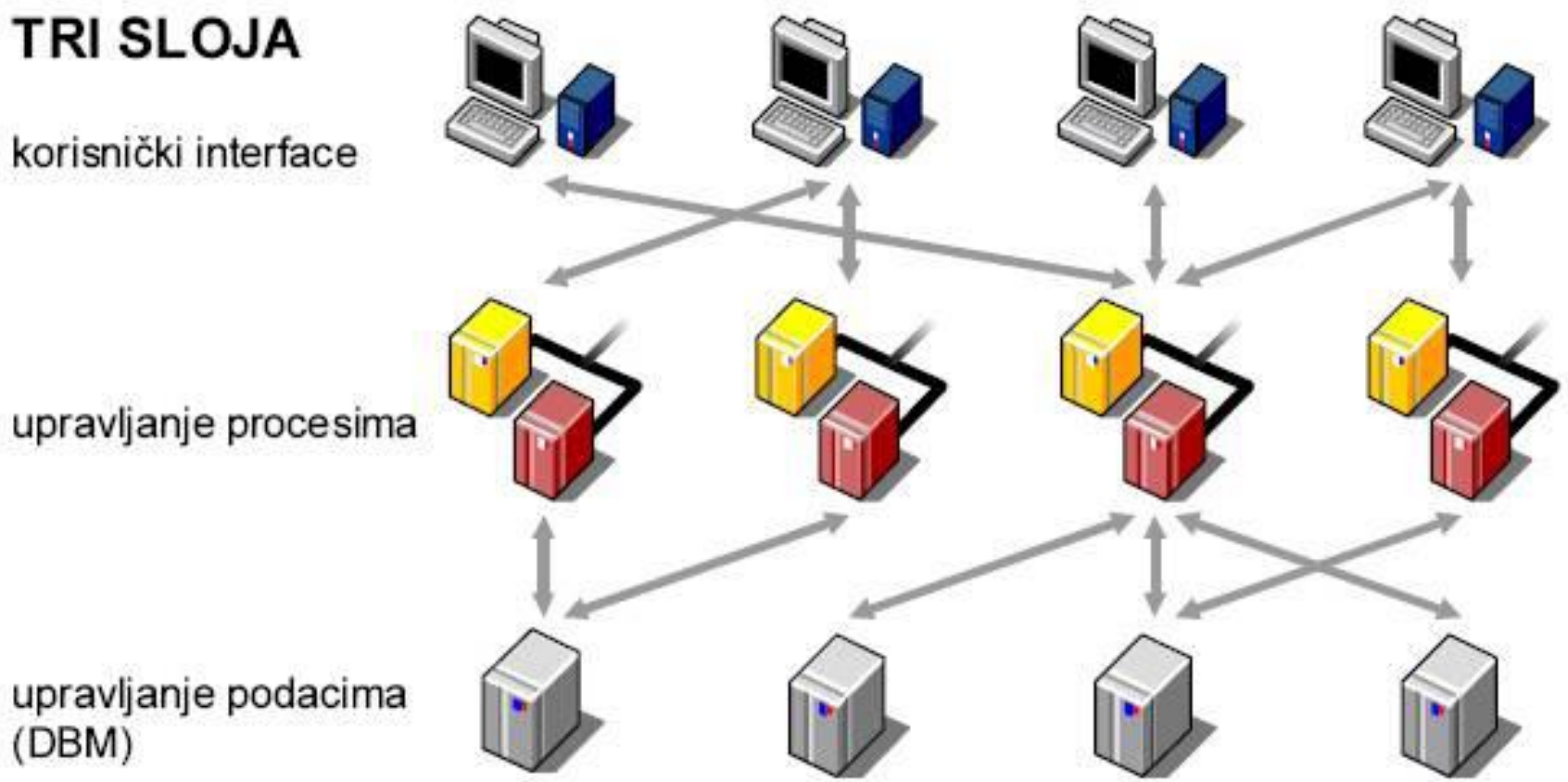
9.2 Dvoslojna Arhitektura

- Jedna od osnovnih karakteristika klijent/server sistema je distribuirana obrada podataka - logika aplikacije se deli između klijenta i servera.
- To znači da se prezentacija podataka kao i provera ulaznih podataka vrši u okviru klijent-aplikacije, dok se rukovanje podacima, u smislu njihovog fizičkog smeštaja i kontrole pristupa, vrši na serveru.
- Kod dvoslojne arhitekture klijent pokreće aplikaciju koja preko mreže pristupa serveru baze podataka a zatim pokreće aplikaciju koja izvršava logiku i prikazuje tražene rezultate na ekranu.
- Glavne prednosti ovakvog modela obrade podataka su: centralizovano upravljanje resursima i jednostavno obezbeđivanje njihove sigurnosti
- Osnovni problem je **nedostatak skalabilnosti** - pad performansi
- Dvoslojna arhitektura je pogodna za sisteme sa **malim brojem korisnika** jer se ona dobro ponaša za manje sredine do **100 klijenata**.
- Kada se korisniku šalje ogromna količina podataka dolazi do zagušenja što utiče na usporavanje protoka informacija kroz mrežu.
- Koristi se za **homogene sredine** gde imamo tačno definisana pravila koja se ne menjaju tako često.

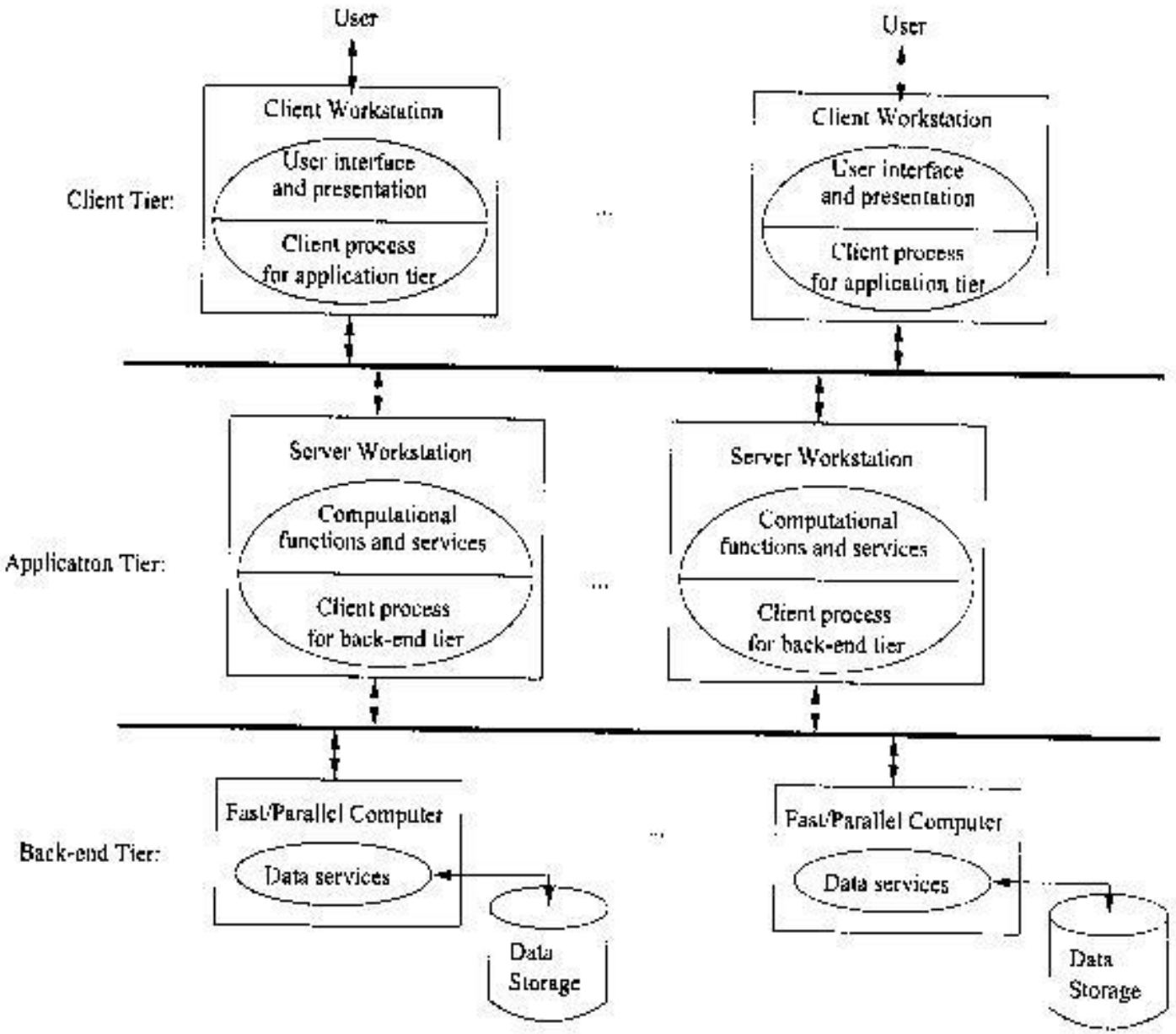
9.3 Troslojna arhitektura

- Troslojna arhitektura je takva arhitektura kod koje je potrebno uvesti još jedan sloj tzv. **srednji sloj** u kome se nalazi aplikativni server.
- Početak razvoja ovakvih arhitektura datira od 1990 godine i za razvoj ovakve arhitekture moguće je primeniti više različitih tehnologija.
- Međutim ono što je zajedničko za ovakvu arhitekturu je pozivni mehanizam koji je jedinstven za svaku vezu klijent server a to je **RPC**
- Kod većine dvoslojnih arhitektura primenjuje se **SQL** mehanizam poziva a taj mehanizam nije zatupljen kod troslojnih arhitektura
- Standardni RPC poziv ima mnogo veću fleksibilnost u odnosu na SQL
- RPC poziv može da sadrži parametre upita koji će specificirati strukturu podataka u kojoj žele da se prihvate traženi podaci.
- Korisnički sloj sada ne mora da zna SQL jezik jer on u RPC pozivu diktira u kom obliku želi da dobije povratne informacije.
- Ako dođe do promene podataka u sloju podataka tada nije potrebno da se menja korisnička aplikacija već se to radi samo u **srednjem sloju**.
- Podaci su organizovani **hijerahijski**, **relaciono** i u **vidu objekata** što omogućuje veliku fleksibilnost u pristupu i uvođenju novih tehnologija

- Troslojna arhitektura se sastoji iz tri sloja i to :
  1. **korisnički sloj** (*User interface*),
  2. **srednji sloj – upravljanje procesima** (*Comutational function* ili *Middle tier server*) i
  3. **sloj podataka – upravljanje podacima** (*Data access*).



9.3 Troslojna arhitektura



9.3 Troslojna arhitektura

- Namena joj je da prevaziđe neke probleme i ograničenja dvoslojne arhitekture time što je potpuno odvojila podatke, obradu i prezentaciju
- Isti tipovi rutina za predstavljanje podataka koji su korišćeni kod dvoslojne arhitekture, mogu da se koriste i ovde
- Kada se zahteva neka obrada tih podataka ili pristup novim podacima, upućuje se novi poziv srednjem sloju koji izvršava obradu podataka ili se generiše novi zahtev, sada kao klijent, ako se traže novi podaci.
- Aplikativni server komunicira sa *database* serverom, podaci dobijeni sa *database* servera se preko aplikativnog servera prosleđuju klijentu.
- Poslovna logika se izvršava na aplikativnom serveru.
- Aplikacija na aplikativom serveru tj. srednjem sloju, je višenitna (***multithread***) i omogućava istovremeni pristup više različitih klijenata.
- Aplikacioni server kreira konekciju sa bazom i rezultate vraća klijentu.
- Radi na principu *Connection Pooling*, zahtevi se stavljaju u jedan *Pool*
- Kada konekcija bude slobodna dodeljuje se sledećem korisniku u zavisnosti od prioriteta korisnika koji čeka u *Pool*-u.
- *Pooling* se može realizovati **IIS**(*Internet Information Server*) i **COM+**

9.3 Troslojna arhitektura

- Podeljenost na zasebne celine omogućava paralelan razvoj softverskih entiteta koji sada mogu da budu potpuno nezavisno razvijani i to na potpuno različitim hardverskim i softverskim platformama.
- Zbog ove osobine ova arhitektura je primenljiva za razvoj aplikacija koje se zasnivaju na distribuiranim bazama podataka.
- Tri ključne stvari koje su nam ovom arhitekturom omogučene su **zaključavanje**, **koegzistentnost** i **replikacija**.
- Veza se dinamički menja u zavisnosti od korisničkih zahteva
- Srednji sloj omogućuje praćenje i upravljanje procesima, njihovo aktiviranje, kontrolu rada, uspavljivanje i obnovu pojedinih procesa.
- Sistemi sa troslojnom arhitekturom (*three-tier architecture*) predstavljaju sisteme sa **tri**, u velikoj meri nezavisna, podsistema:

1. **podsistem za interakciju sa korisnikom** (f-je korisničkog interfejsa);
2. **podsistem za implementaciju osnovnih funkcija sistema** - poslovna logika sistema
3. **podsistem za rukovanje podacima** – zadužen za fizičko smeštanje podataka ili podsistem za upravljanje bazama podataka

9.3 Troslojna arhitektura

- Za razliku od dvoslojnog modela obrade podataka, gde je logika aplikacije bila podeljena između klijenta i servera, u troslojnom modelu ona se nalazi koncentrisana u tzv. ***aplikacionom serveru***
- Njegova namena je da izvršava programski kod koji implementira logiku aplikacije.
- Klijent aplikacija je namenjena samo za implementaciju korisničkog interfejsa, a funkcija sistema za upravljanje bazom podataka je isključivo fizičko rukovanje podacima
- Troslojni koncept je doveo do podele programskog koda na segmente koji implementiraju tačno određene funkcije sistema.
- Tako organizovan sistem je **jednostavniji za održavanje**, jer je moguće nezavisno razvijati korisnički interfejs i logiku aplikacije.
- Za potrebe fizičkog rukovanja podacima najčešće se koristi neki od **komercijalno dostupnih servera** za tu namenu.
- Troslojne arhitekture sistema podrazumevaju oslanjanje na standarde u odgovarajućom oblastima, zasnovane na Internet tehnologijama.

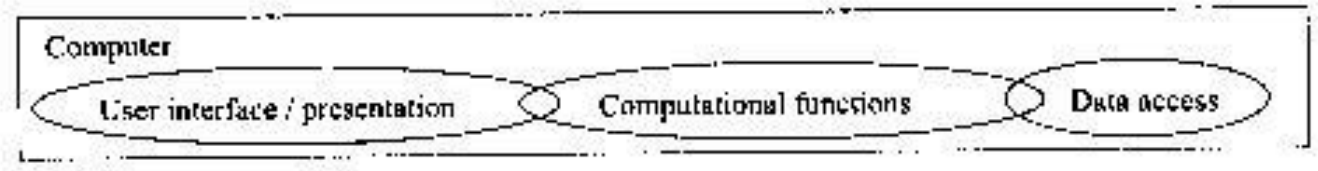
9.3 Troslojna arhitektura

Prednosti troslojne arhitekture:

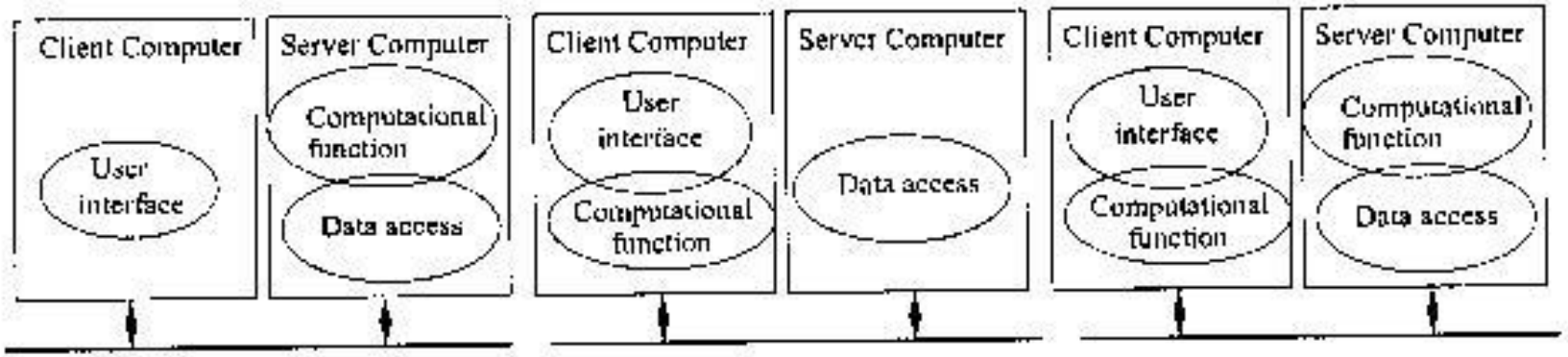
1. Važna karakteristika troslojnih sistema je **skalabilnost**.
   a) jednostavno povećavanje broja klijenata
   b) povećavanje propusne moći i brzine odziva servera srednjeg sloja je moguće kroz dodavanje novih serverskih mašina.
2. Povećana je **pouzdanost** i **fleksibilnost** **(**veći broj servera**)**
3. **Logika aplikacije** se može menjati i u toku rada sistema
4. Moguće je efikasno vršiti **balansiranje opterećenja** serverskog podsistema

- Omogućava integraciju heterogenih sistema u pogledu korišćene hardverske i softverske opreme.
- Ova arhitektura se koristi u **heterogenim sredinama** koje zahtevaju povezivanje heterogenih baza podataka.
- Primenjije se za **povezivanje velikog broja klijenata** gde je potrebno povezati više od **1000** klijenata.
- Prilagodljivost brzim promenema, kako u **korisničkom** (poslovnom), tako i u **implementacionom** (tehnološkom) okruženju.

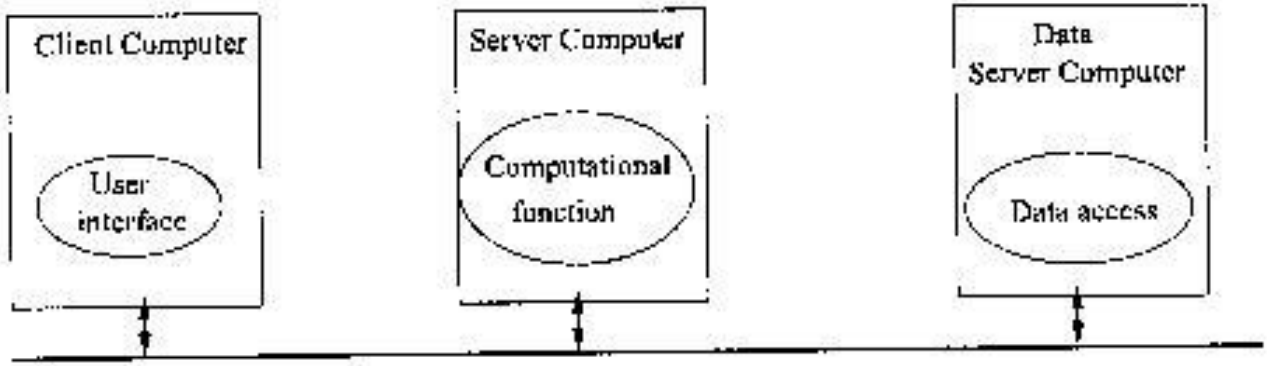
9.3 Vrste klijent server arhitektura



(a) Centralised configuration



(b) Two-tier configurations



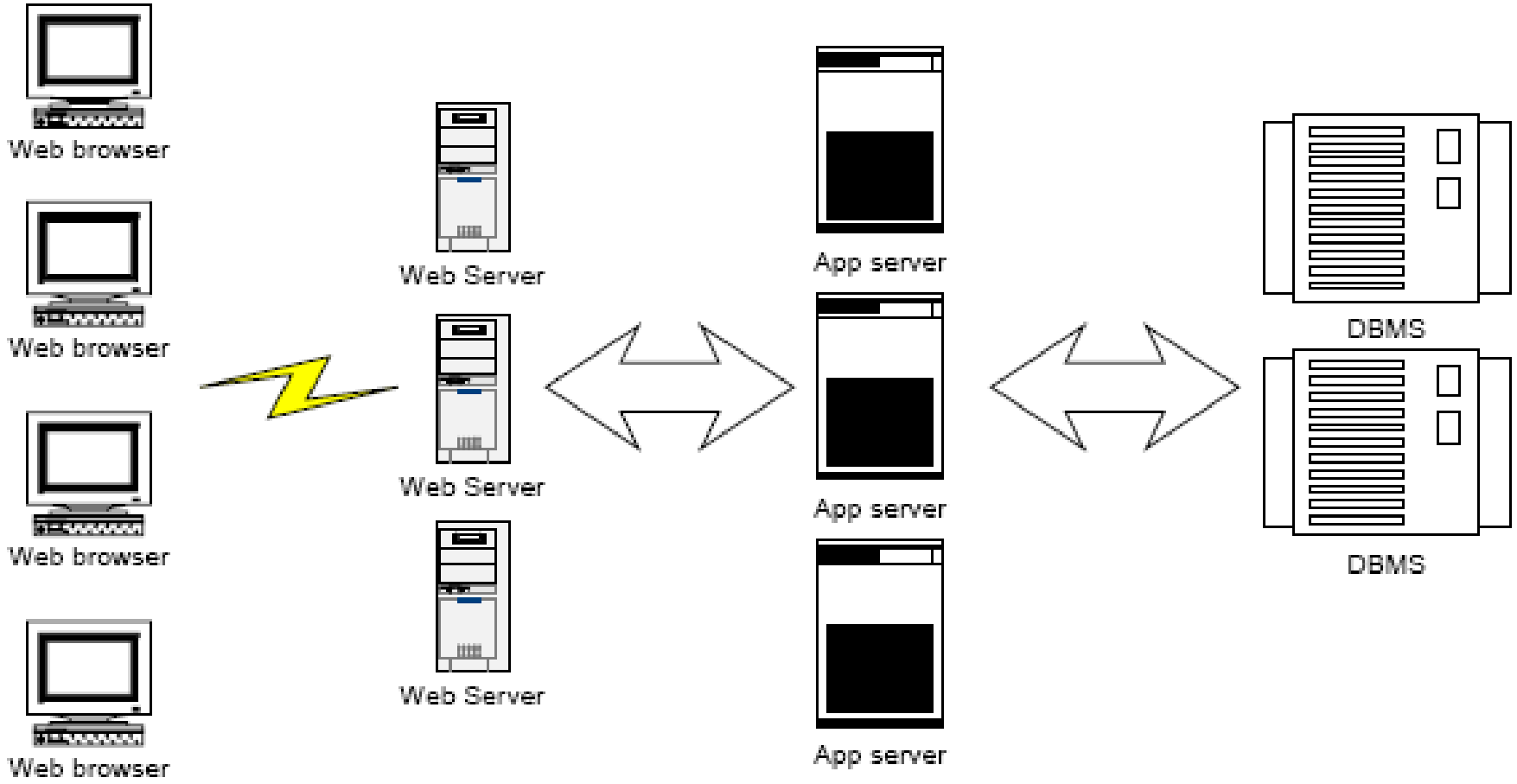
(c) A three-tier configuration

Figure 3. Examples of three-tier configurations

9.4 Višeslojne arhitekture

- Troslojna arhitektura je generička za višeslojne arhitekture
- Dalja podela na komponente u okviru srednjeg sloja sa ciljem još većeg povećanja skalabilnosti, odnosno performansi.
- Srednji sloj se deli na dva sloja:
  1. namenjen za opsluživanje Web klijenata,
  2. vrši implementaciju poslovne logike sistema.
- Srednji sloj se deli na dva ili više slojeva u zavisnosti od aplikacije
- To se javlja kod nekih Internet aplikacija koje imaju tankog klijenta pisanog u HTML jeziku a aplikacioni serveri su pisani u C++ ili JAVA.
- Postoje velike razlike između ovih tehnologija koje predstavljaju neprolaznu prepreku da bi se aplikacije međusobno direktno spojile.
- Zato postoji međusloj (***intermediate layer***), obično WEB server , koji je implementiran u nekom ***script*** jeziku.
- Taj sloj prihvata zahteve od klijenata i generiše HTML dokument koristeći servise koje pozajmljuje od poslovnog (*business*) sloja.
- Povećavanjem broja slojeva, odnosno većeg stepena indirekcije, omogućava se veća **modularnost**, **heterogenost** i **elastičnost** sistema

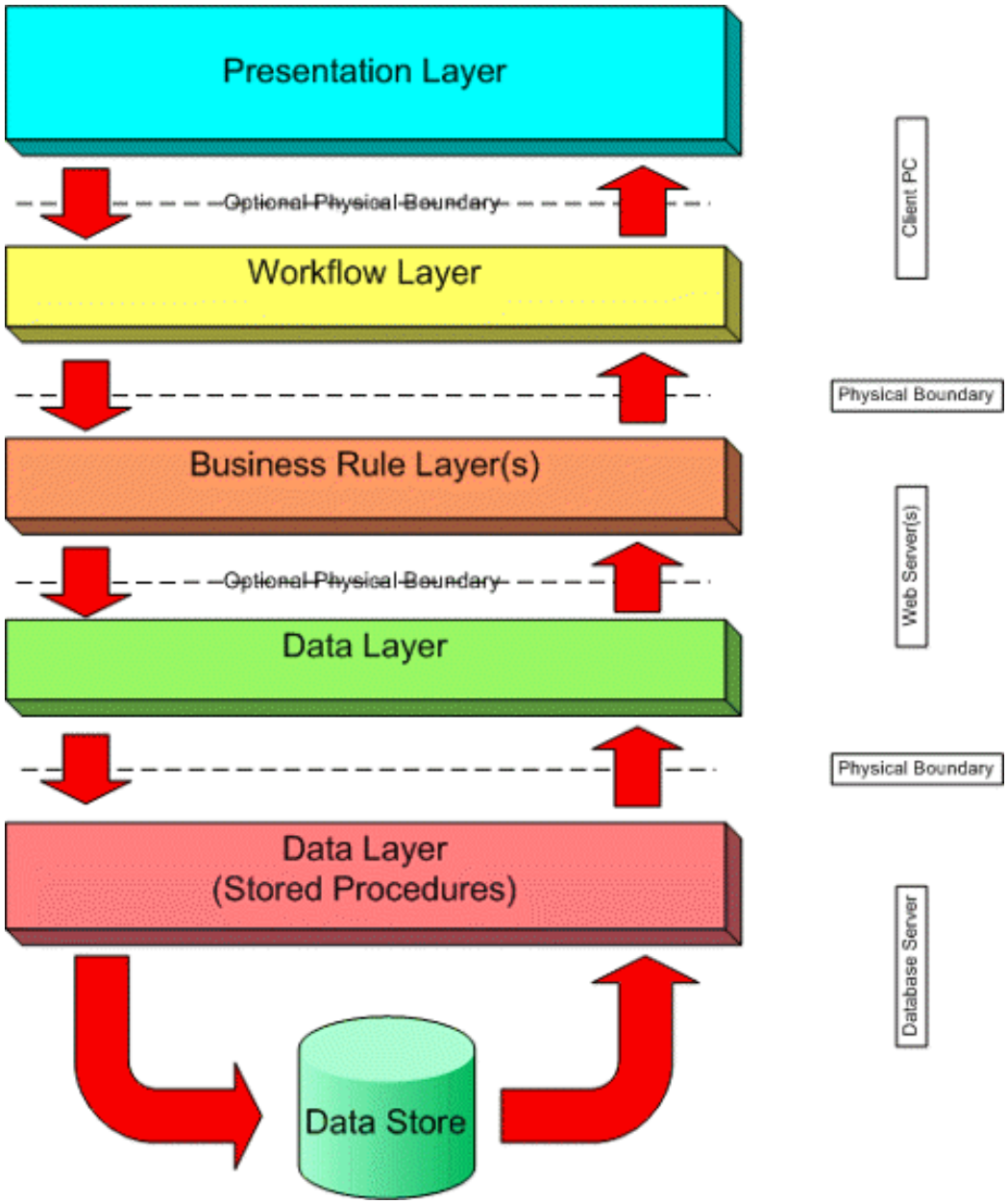
9.4 Višeslojna arhitektura



Prednosti: Obrada podataka se širi na razne servere čime se izjednačava opterećenje među pojedinim serverima i povećava skalabilnost sistema

Nedostatci: Veće opterećenje mreže i znatno je teža za razvoj i testiranje aplikacija koje rade u višeslojnoj arhitekturi

9.4 Višeslojna arhitektura



9.4 Java tehnologije kod višeslojnih arhitektura

- Interakciju sa korisnikom u ovakvom sistemu obavljaju klijenti koji imaju standardan Web interfejs.
- U pitanju su Web čitači koji prikazuju HTML stranice.
- Komunikacija između Web čitača i Web servera se odvija putem standardnog HTTP protokola, uz dodatak ***cookie*** - podataka kojima se prati korisnička sesija dok se on kreće po tom Web sajtu.
- Stranice koje prikazuju klijenti su najčešće generisane dinamički.
- Dinamičko generisanje Web sadržaja na osnovu podataka iz ostatka sistema vrše **servleti** ili se za tu namenu koriste **J***ava* **S***erver* **P***ages*
- Za ažuriranje podataka u sistemu servleti ili JSP stranice pristupaju objektima u okviru aplikacionih servera koji su dostupni kao **CORBA** ili **EJB** (*Enterprise JavaBeans*) komponente.
- Protokol za komunikaciju između ova dva sloja je **JRMP** (*Java Remote Method Protocol*) ili **IIOP** (*Internet Inter-ORB Protocol*)
- CORBA/EJB komponente za potrebe skladištenja podataka u bazi podataka pristupaju serveru za upravljanje bazama podataka preko standardnog **JDBC** (*Java Database Connectivity*) interfejsa.

9.5 Izbor klijent server arhitekture

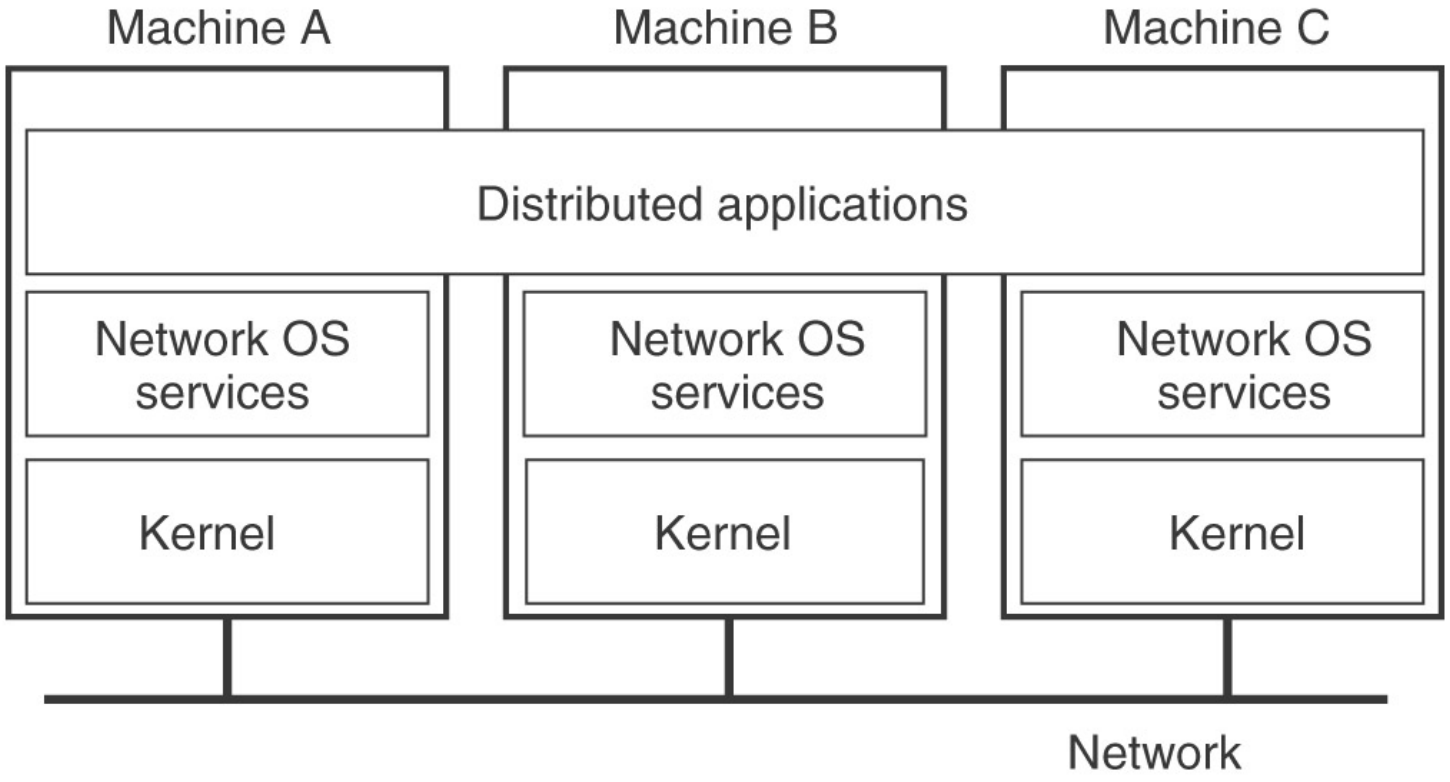
- Sve arhitekture softverskog sistema imaju svoje prednosti i nedostatke.
- Proces izbora arhitekture pored funkcionalnih zahteva treba da uzme u obzir i nefunkcionalne zahteve:

1. **Operativni** (*operational*) zahtevi: specificiraju okolinu u kojoj treba da radi aplikacija kao i kako se ona tokom vremena menja
2. **Zahtevi za performansama** (*performance requirements*): odnose se na performanse sistema kao što su **pouzdanost** (*reliability*), **kapacitet**, **vreme odziva** (*responese time*) i sl.
3. **Sigurnosni** (*security*) zahtevi: predstavljaju stepen zaštite podataka od oštećenja ili gubitka prouzrokovanog namernim postupkom (hakeri) ili slučajnim događajem (elementarna nepogoda)
4. **Kulturološki/politički** (lokalizacijski) (*cultural/political*) zahtevi: važe za pojedine države ili pojedina geografska područja

- Prvi korak je prerada nefunkcionalnih zahteva iz faze analize u detaljan opis nefunkcionalnih zahteva iz kojih se može onda izabrati arhitektura.
- Nakon toga se iz detaljnih nefunkcionalnih zahteva i dizajna arhitekture radi specifikacija potrebnog hardvera i softvera

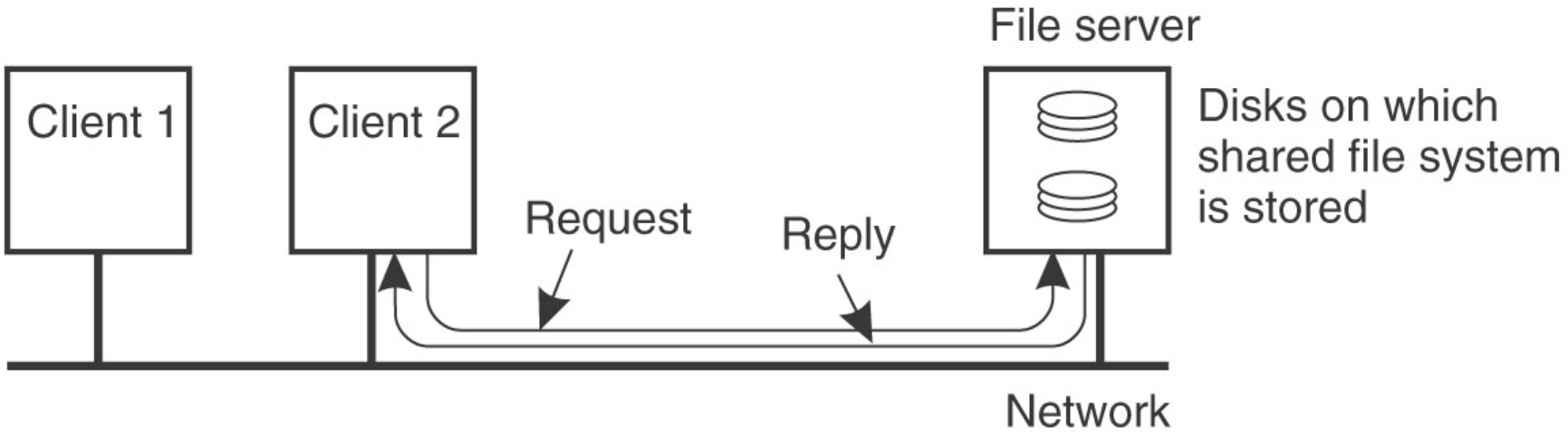
9.6 Mrežni operativni sistemi

- Prilagođavaju se OS svakog pojedinačnog računara i ne stvaraju sliku jedinstvenog sistema kao kod DS-a.
- Mrežni operativni sistemi svaki računar vide posebno i pružaju mrežne servise OS na svakom mrežnom računaru.



9.6 Mrežni operativni sistemi

- Komunikacija u mrežnim operativnim sistemima usmerena je na prenos poruka između klijent računara i računara namenjenih pružanju usluga drugim računarima (*Data server*).

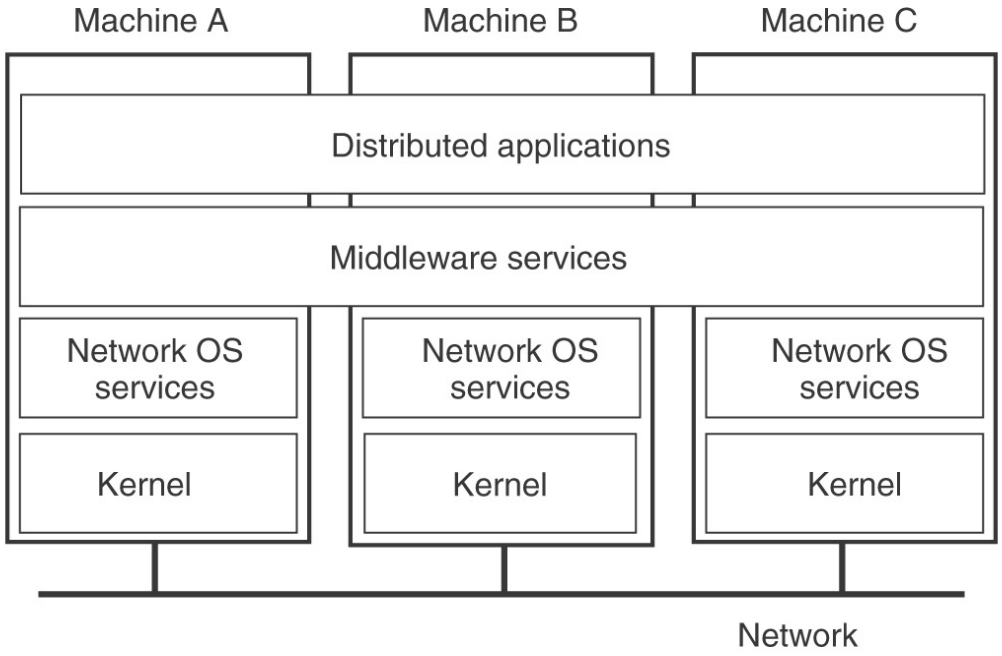


9.6 Mrežni operativni sistemi

- **Middleware** je dodatni sloj softvera koji se pozicionira između distribuiranih aplikacija i mrežnih operativnih sistema.
- Middleware je namenjen usklađivanju različitosti između mrežnih operativnih sistema.
- Middleware se primenjuje u distribuiranom sistemu podataka, zatim pri izvršavanju poziva na udaljenom računaru (*Remote Procedure Call – RPC*) ili za rad distribuiranih objekata.

Usluge middleware-a su:

1. Komunikacijske usluge
2. Imenovanje
3. Postojanost podataka
4. Distribuirane transakcije
5. Sigurnost



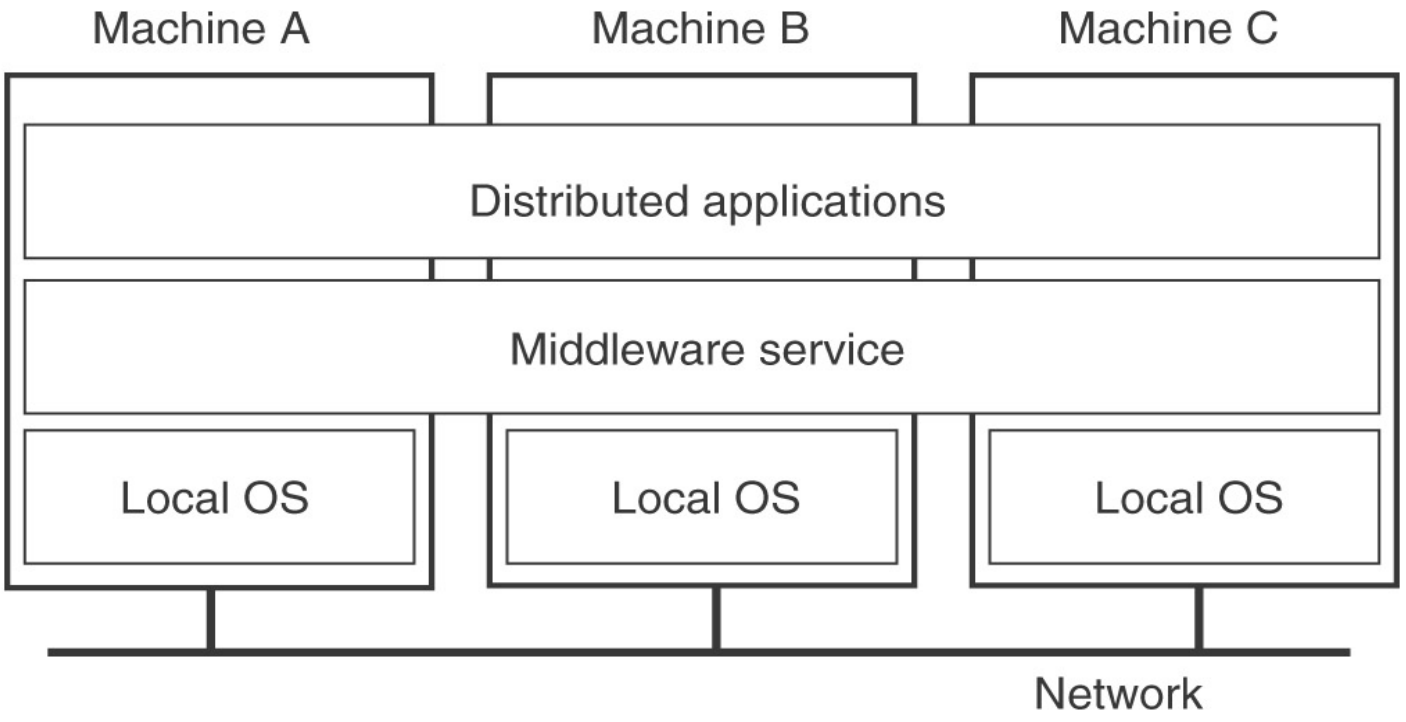
9.6 Razlozi za uvođenje DSa

- Ukoliko želimo povećati kvalitet sistema pri porastu broja korisnika nailazimo na niz ograničenja:

1. Primena centraliziranog servera za sve korisnike uslovljava veliki stepen opterećenja servera, ali u nekim situacijama nije opravdana upotreba više servera iste namene (jedan od razloga je sigurnost).
2. Pristup centraliziranim podacima je problematičan ako je u pitanju velika količina podataka jer dolazi do velikog opterećenja servera na kome su podaci smešteni. Ukoliko se koristi više računara mora se osigurati jednakost podataka na svim serverima.
3. Primena centraliziranih algoritama nije dobro rešenje jer se komunikacija izvodi slanjem i primanjem velikog broja paketa pa je poželjno koristiti različite mrežne linije u cilju ubrzanja izvođenja algoritma. Za to je potrebno imati informacije o raspoloživim putanjama i odabrati najpovoljniju putanju za prenos paketa.

9.6 Pojam distribuiranih sistema

- Distribuirani sistemi (DS) predstavljaju jedan vid mrežnih sistema koji omogućavaju računarima istovremeno izvođenje obrade podataka korišćenjem svih raspoloživih mrežnih resursa u toj mreži.
- Upotrebom DS-a klijent može pristupati svim mrežnim resursima kao resursima lokalnog računara tako da DS predstavlja skup nezavisnih računara koji su klijentu predstavljeni kao jedan koherentan sistem.



9.6 Prednosti distribuiranih sistema

1. **Ubrzanje obrade podataka** - obrada se istovremeno izvodi na više računara pa je ukupno vreme obrade kraće u odnosu na jedan računar
2. **Veća pouzdanost** -u slučaju nekorektne obrade izvodi se ponavljanje obrade na računaru na kome je greška nastala, dok se obrade izvedene na ostalim računarima ne ponavljaju; u slučaju izvođenja na jednom računaru nužno je ponavljanje svih obrada.
3. **Ravnomerno opterećenje računarske mreže** - ako je neki od računara u mreži jako opterećen nekom obradom, moguće je izvršiti njegovo rasterećenje prenosom obrade na manje opterećene računare.
4. **Smanjeni zahtevi za hardverom** - dovoljno je imati nekoliko računara sa nadprosečno kvalitetnim hardverskim osobinama, a zatim se sve hardverski zahtevnije obrade preusmeravaju na te računare.
5. **Smanjene zahteva za softverom** - skupi softverski proizvodi instaliraju se na nekoliko računara, a zatim se sve potrebne obrade preusmeravaju na jedan od tih računara; na taj način se smanjuje potreban broj licenci i cena upotrebe softverskih paketa.

9.6 Ciljevi distribuiranih sistema

1. **Povezivanje korisnika i resursa** - omogućuje korisniku pristup resursima svih računara u mreži i njihova upotreba kao lokalnih resursa računara na koje je korisnik spojen. DS-i koji se korisniku i aplikacijama prikazuju u formi jednog sistema su transparentni.
2. **Transparentnost** – u DS prikazuje se u sledećim oblicima:
   a. Transparentnost pristupa (***access***) – prikrivanje razlike u prikazu podataka (različite hardverske osnove računara koriste različite formate u prikazu podataka) i načina korišćenja resursa.
   b. Transparentnost lokacije (***location***) - korisnik ne treba da zna fizičku lokaciju resursa koji sadrži podatke koje on upotrebljava
   c. Transparentnost promene mesta (***migration***) – promena fizičke lokacije resursa ne utiče na način pristupa resursima.
   d. Transparentnost relokacije (***realocation***) - omogućava promenu lokacije resursa u trenutku kada se resurs koristi.
   e. Transparentnost replikacije (***replication***) – mogućnost postojanja više kopija originalnih podataka a da korisnik to ne zna.

9.6 Ciljevi distribuiranih sistema

f. Transparentnost konkurentnosti (***concurrency***) - mogućnost korišćenja skupa podataka za više korisnika istovremeno, pri čemu to korisnici ne vide i ne znaju za to.

g. Transparentnost greške (***failure***) – korisnici ne opažaju nefunkcionalnost određenog resursa kao ni postupak oporavka sistema u slučaju kvara.

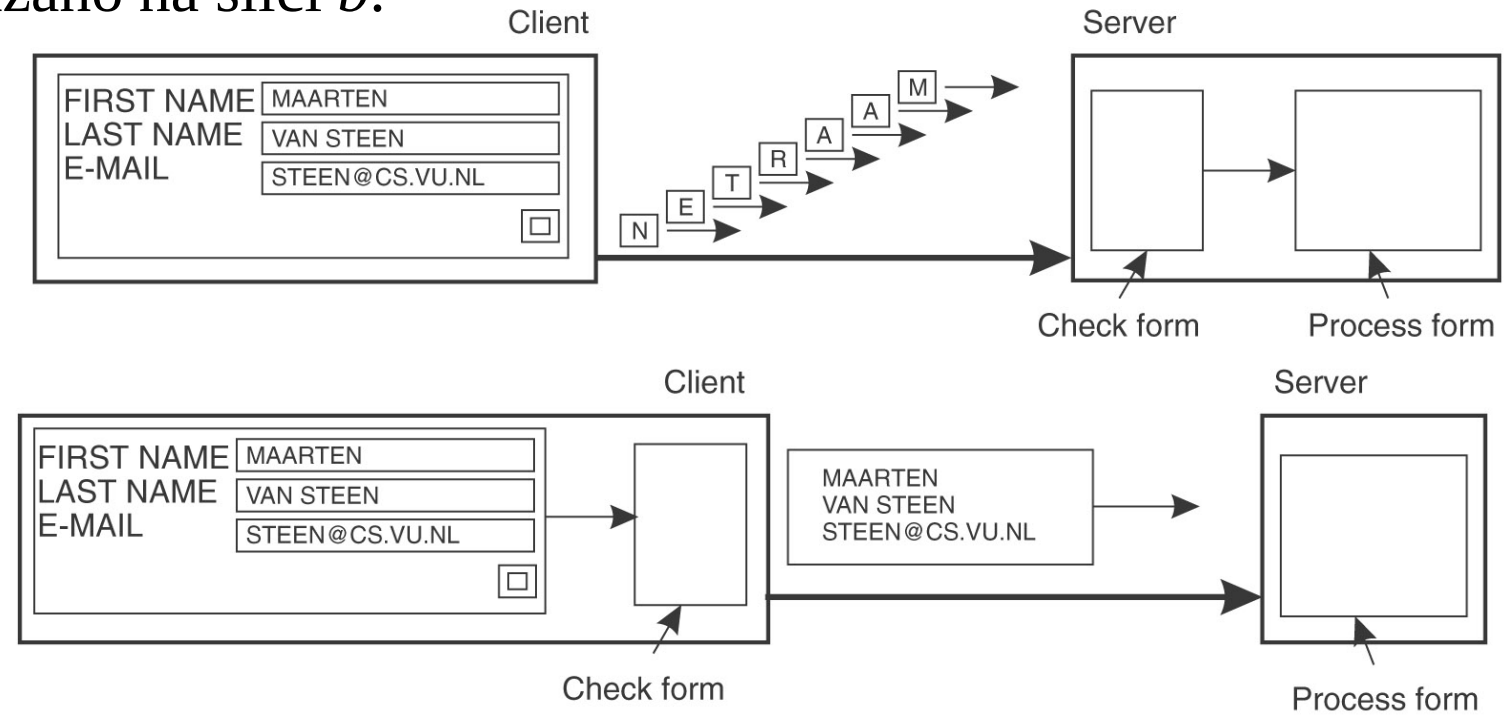
h. Transparentnost postojanosti (***persistance***) – korisnik nije upoznat sa trenutnom lokacijom podataka koji koristi (on taj postupak vidi kao direktnu operaciju u bazi podataka).

3. **Otvorenost** - omogućuje izvođenje usluga prema prihvaćenim standardima. Standardi za DS sadržani su u **interfejsu** (*interface*) koji se opisuje sa jezikom za definisanje interfejsa (*Interface Definition Languge* - **IDL**). IDL sadrži imena funkcija sa popisom parametara, rezultata izvođenja, mogućih razlika itd. Popis svih funkcija čini *specifikaciju*. Osobine dobre specifikacije su kompletnost i neutralnost.
4. **Merljivost** (*scalability*) - osobina koje se primenjuje na tri načina: prema **veličini**, prema **lokaciji** i prema **administriranju**.

9.6 Ciljevi distribuiranih sistema

Tehnike za realizaciju scalability-a su:

1. **Asinhrona komunikacija** - nakon slanja poruke aplikacija pokreće druge aktivnosti, tj. ne čeka na prijem odgovora od servera, a kada odgovor stigne on se prekidom dojavljuje aplikaciji te poseban proces (***handler***) izvodi obradu pristiglog odgovora. Nekada je bolje većinu obrade izvesti na klijent strani kako bi se rasteretio server kao što je prikazano na slici *b*.

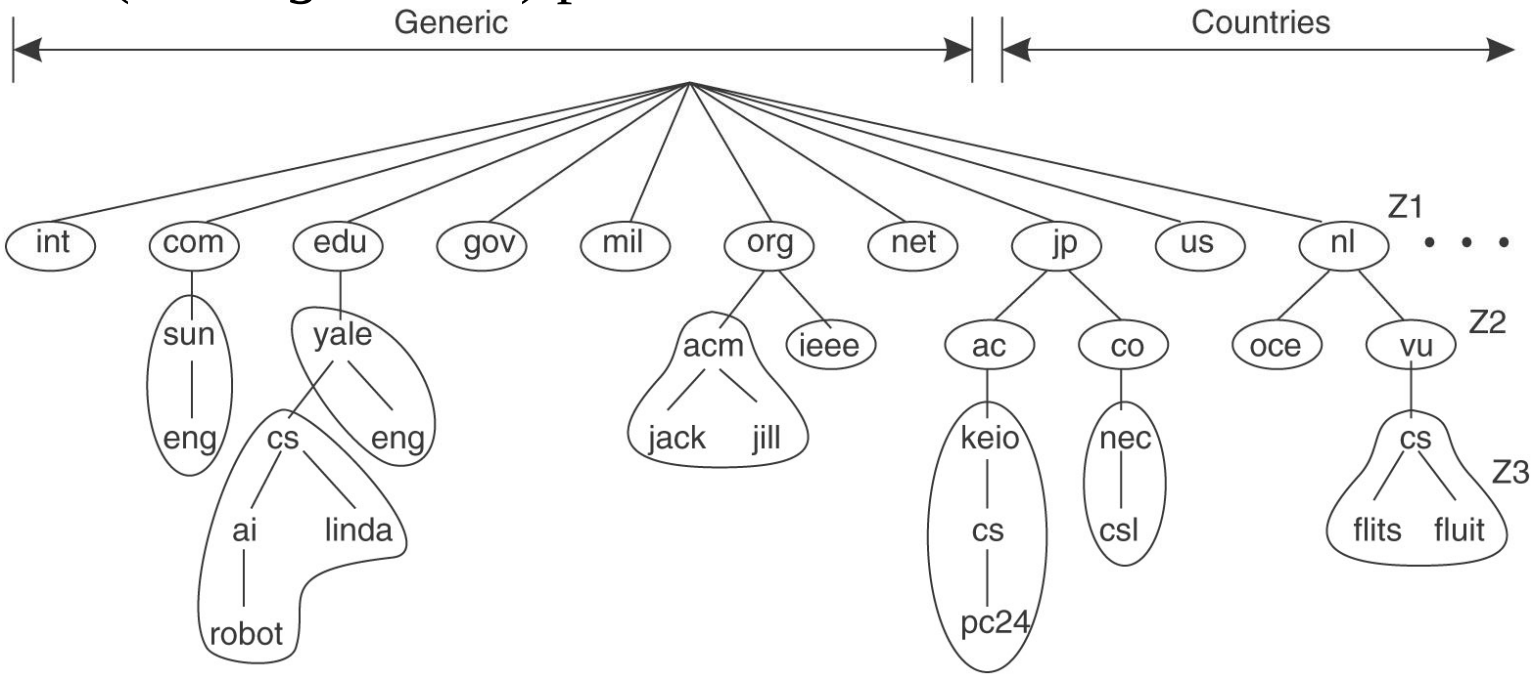


(b)

9.6 Ciljevi distribuiranih sistema

2. **Distribucija** - znači slanje određene komponente podeljene na manje delove i njihova distribucija putem mreže kako bi se na odredištu svi delovi ponovo spojili u celinu.

- Primer je rad DNS servera gde se primenom zona izvodi usluga dodele imena (*naming service*) primenom više decentraliziranih servera



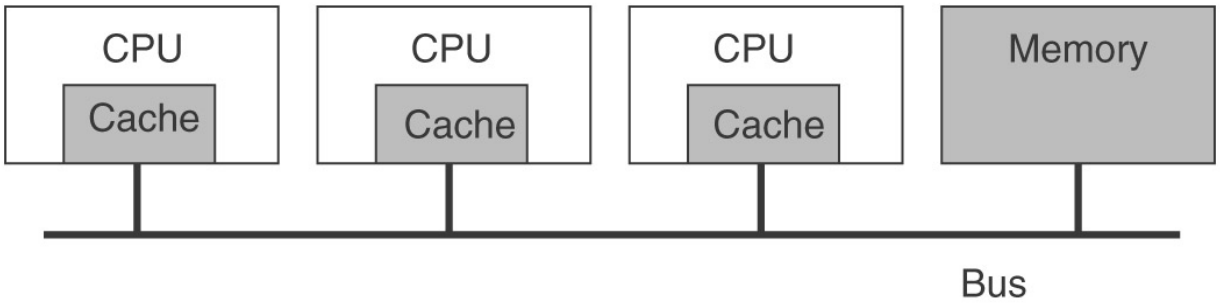
3. ***Caching*** – to je specijalna forma kopije (replike) podataka, a njena upotreba definisana je potrebama klijenta i servera koji ima originalnu verziju podataka.

9.7 Hardverska realizacija DS

- DS se realizuju sa različitim hardverskim konfiguracijama računara.
- Različitost računara uslovljava nivo složenosti distributivnog sistema

Dele se na:

1. **Multiprocesorske računarske sisteme -** jedan adresni prostor.
   - ✓ Ukoliko se promena vrednosti memorijske lokacije od jednog CPU-a može prikazati na drugom CPU za **1μs** imamo **koherentan sistem**.
   - ✓ Problem je opterećenje magistrale i smanjene performanse sistema.
   - ✓ Koristi se ***cache*** memorija koja rasterećuje magistrala, ali se pojavljuje problem ažuriranja vrednosti ***cache*** memorije ukoliko dva ili više CPU pristupaju istoj memorijskoj lokaciji

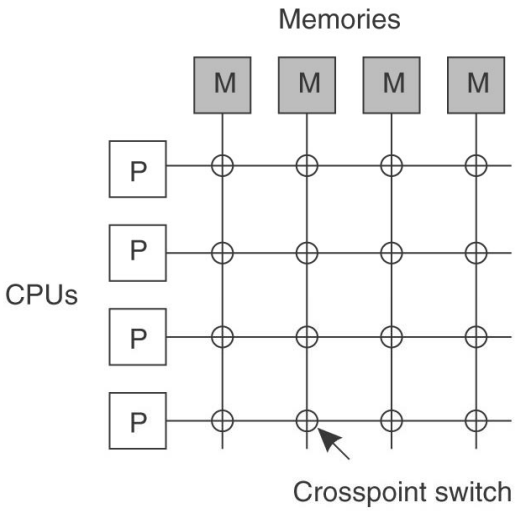


9.7 Hardverska realizacija DS

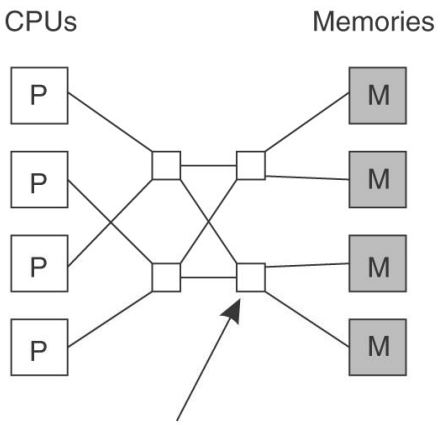
- Primena velikog broja procesora uslovljava podelu memorije u module kako bi se poboljšale osobine sistema.
- Pri radu sa memorijom svaki procesor može pristupati memorijskim modulima korišćenjem prekidača koji mogu biti:

a) ***Crossbar*** prekidači omogućuju direktno spajanje svakog procesora sa svakim memorijskim modulom, pružaju veliku brzinu spajanja ali uslovljavaju velik broj prekidača.

b) Manji broj prekidača postiže se upotrebom ***omega*** prekidača gde se njihovom upotrebom bira putanja pristupa memorijskim modulima.



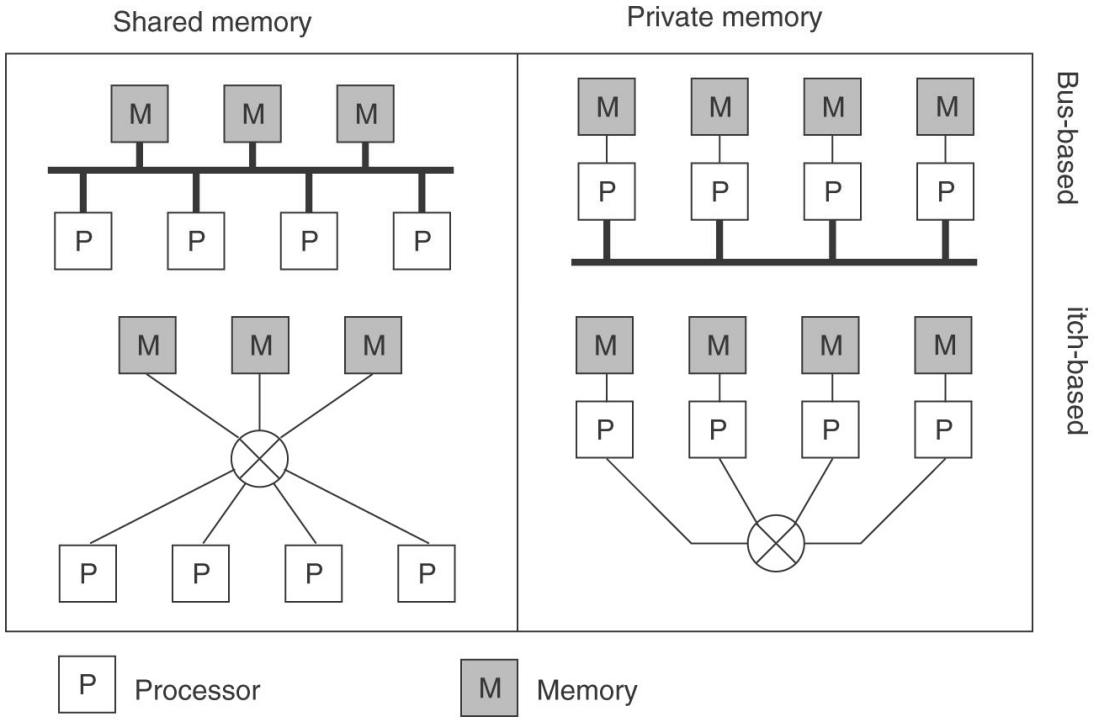
(a)



(b)

9.7 Hardverska realizacija DS

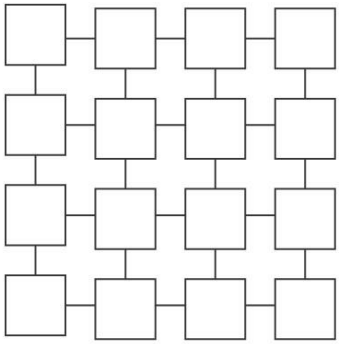
2. **Multiračunarske računarske sisteme** – zasebni memorijski prostori
   - ✓ Arhitektura se zasniva na **magistrali**(*bus*) ili na **prekidačima**(*switch*)
   - ✓ Multiračunarski sistemi mogu biti **homogeni** i **heterogeni**.
   - ✓ **Homogeni** sistemi imaju skup računara istih karakteristika jer koriste iste CPU (struktura je identična) i imaju jednu mrežnu tehnologiju.
   - ✓ **Heterogeni** sistemi sastoje se od velikog broja različitih računara koji su spojeni na različite mreže.



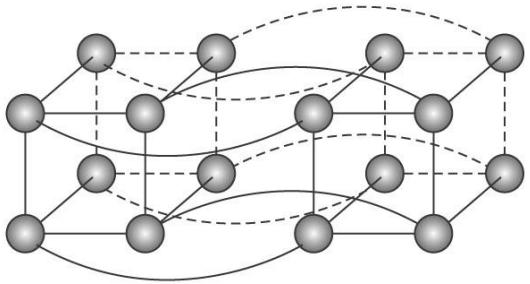
9.7 Hardverska realizacija DS

Homogeni multiračunarski sistemi

- Svi računari imaju istu strukturu pa je njihova upotreba jednostavnija
- Problem način povezivanja i komunikacije računara.
- Korisiti se struktura **rešetke** (*grid*) ili **hiperkocke** (*hypercube*).



(a)



(b)

Hetrogeni računarski sistemi

- Sastoje se od većeg broja različitih računara spojenih na mrežu putem različitih mrežnih struktura.
- Primenom DS-a generiše se srednji (***middleware***) sloj koji je nezavisan od hardvera i omogućava da se izvrši komunikacija između računara.
- Premošćuju se problemi koji nastaju zbog različitosti računara.

9.8 Softverska podrška dist.sistemima

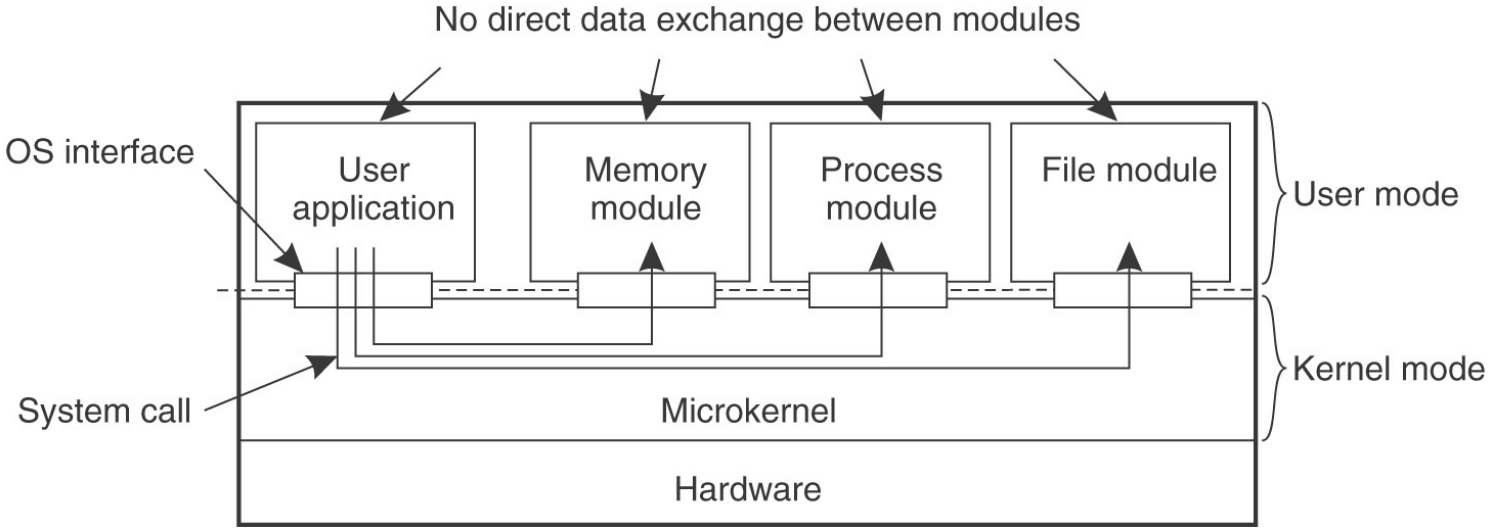
- Pri analizi softvera razlikujemo mrežne računare koji koriste OS koji:
  1. ima kontrolu nad svim resursima mreže (***tightly-coupled system***) i
  2. kontroliše samo resurse lokalnog računara (***loosely-coupled system***).
- Za prvu grupu koristi se **distribuirani** OS, a za drugu **mrežni** OS.

Tabela prikazuje razlike između ovih operativnih sistema:

Sistem	***Opis***	***Ciljevi***
Distribuirani OS	*Tightly-couped* sistemi; primena za višeprocesorske i homogene sisteme	prikrivaju način upotrebe resursa
Network OS	*Loosely-couped* sistemi; primena za heterogene sisteme	pružaju lokalne usluge za udaljene klijente
Middleware	dodatni sloj na NOS za implementaciju usluga opšte namene	osigurava transparentnost

9.8 Softverska podrška dist.sistemima

- Razlikujemo multiprocesorske DS i multiračunarske DS.
- Primena mikrokernela u dizajnu OS jednog računara zasnovanog na klijent-server strukturi određuje način rada DS-a.



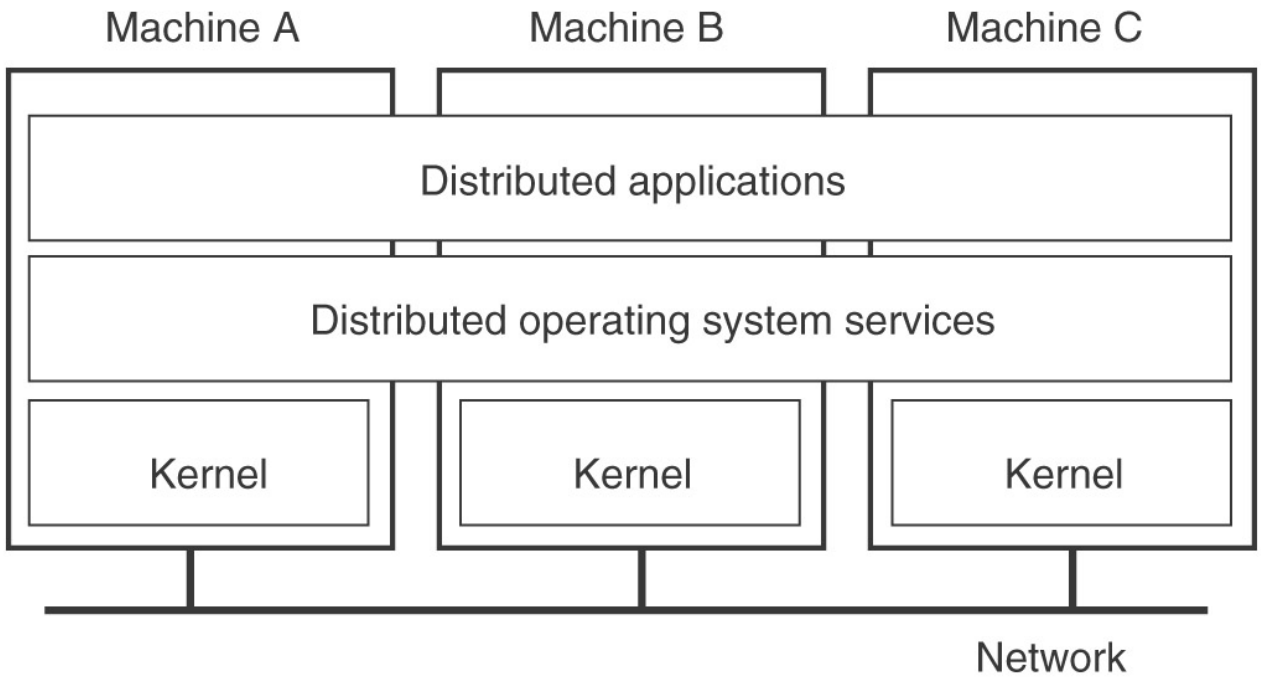
Multiprocesorski DS - omogućuju povećanje performansi sistema.

- ✓ Aplikacije koje pokrećemo ne uočavaju upotrebu više CPU
- ✓ Komunikacija u višeprocesorskom sistemu ostvaruje se manipulacijom podataka između deljenih memorijskih lokacija za više procesora.
- ✓ Uslov koji mora biti zadovoljen je onemogućavanje istovremenog pristupa jednoj memorijskoj lokaciji dva ili više CPU istovremeno.
- ✓ Omogućeno je primenom semafora, monitora i uslovnih promenljivih.

9.8 Softverska podrška dist.sistemima

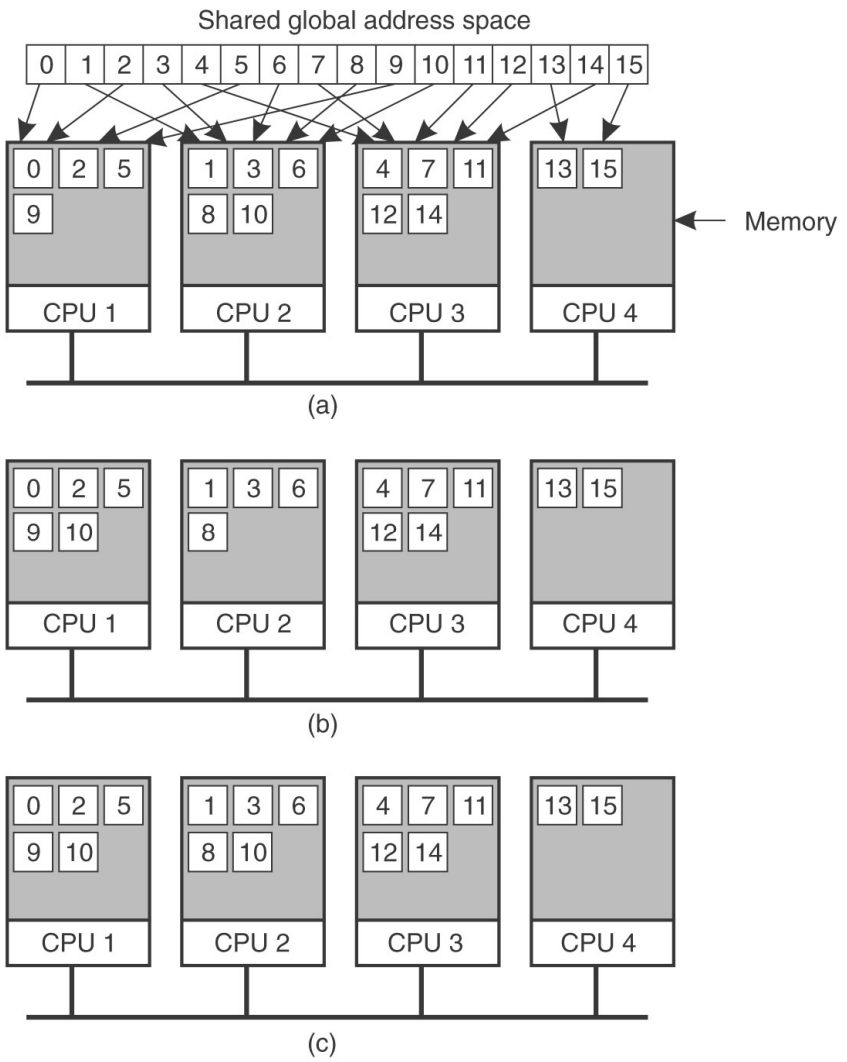
Multiračunarski DS

- ✓ Omogućuju rad potpuno različitih računarskih sistema
- ✓ Njihova kompleksnost znatno je veća u odnosu na multiprocesorske
- ✓ DS za multiračunarske sisteme se zasnivaju na slanju poruka.
- ✓ Za prenos poruka bitna je usklađenost pošiljaoca i primaoca pri slanju poruka, a pri tome se koriste ***bufferi***.



9.8 Softverska podrška dist.sistemima

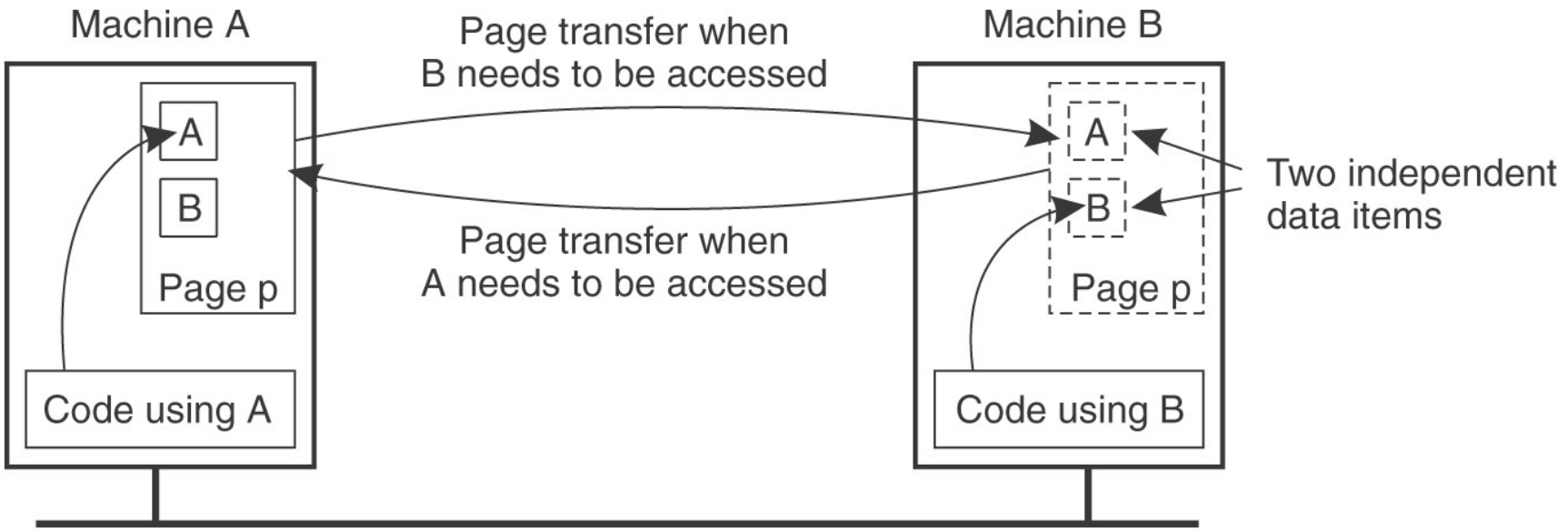
- **DS** sa deljenom memorijom zasnivaju svoj rad na deljenju globalnog memorijskog prostora računarske mreže.
- Adresni prostor je podeljen na stranice veličine 4 KB ili 8 KB
- Princip je da svaki računar koristi stranice u svojoj memoriji, a kada je potrebno koristiti stranicu koja nije u lokalnoj memoriji generiše se zahtev za učitavanjem tražene stranice.
- Princip sličan klasičnom straničenju ali se za virtuelnu memoriju ne koristi sekundarna memorija, nego RAM memorija drugih računara u mreži.



a) početno stanje
b) CPU 1 referencira stranicu 10
c) stanje nakon replike stranice 10

9.8 Softverska podrška dist.sistemima

- Pri radu sa zajedničkom memorijom pojavljaju se problemi uzrokovani obradom podataka unutar jedne stranice koja se koristi na različitim CPU
- Istovremene promene nezavisnih promenljivih rezultirali bi gubitkom promena za jedan od procesa koji se izvršavaju na različitim CPU.
- U tom slučaju promena promenljive u jednoj stranici nije vidljiva u drugoj stranici i može doći do greške (***false sharing***)



9.9 Komunikacija u distrib.sistemima

- Komunikacije su ključni element svih distribuiranih sistema.
- Najvažniju, ključnu ulogu u komunikaciji igraju **protokoli**.
- Prema modelu realizacije njih možemo podeliti na:
  1. protokole zasnovane na slojevima (*layers*) (OSI model i ATM)
  2. klijent-server model protokola.
- Osnovna razlika DS i lokalnog sistema je međuprosesna komunikacija
- Zasniva se na primeni poruka niskog nivoa (bliskom hardveru računara) korišćenjem postojeće mrežne strukture.
- Komunikacija dva računara izvodi se primenom strukture slojeva.
- Svaki sloj izvodi set aktivnosti i čini osnovu za primenu višeg sloja.
- Niži slojevi usmereni su prema hardveru, a viši prema aplikacijama.

Primenjujemo sledeće modele komuniciranja:

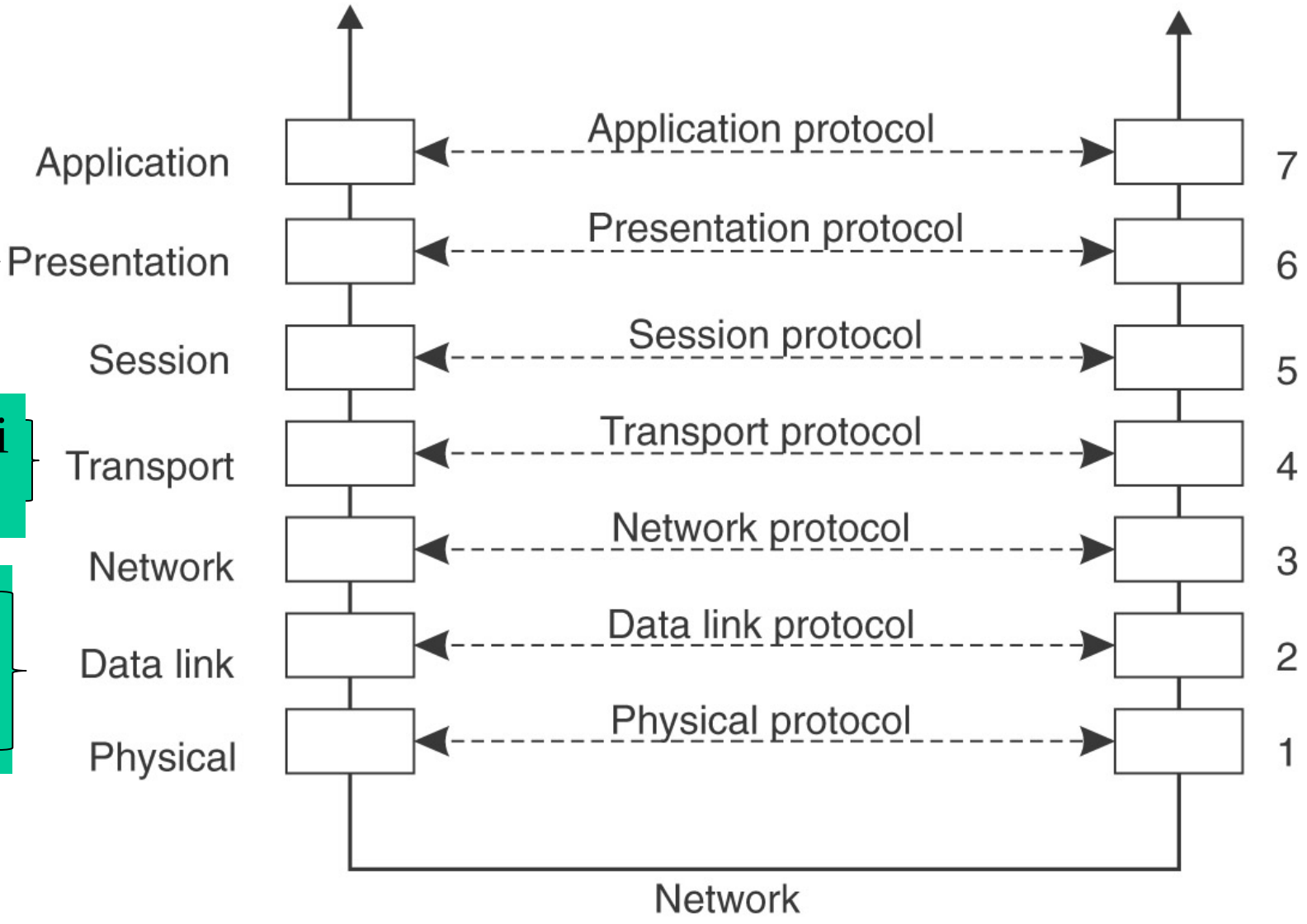
1. **Poziv procedure na udaljenom računaru** (***R****emote* ***P****rocedure* ***C****all*)
2. **Poziv metode na udaljenom računaru** (***R****emote* ***M****ethod* ***I****nvocation*)
3. **Međusloj usmeren prenosu poruka** (***M****essage* ***O****riented* ***M****iddleware*)
4. **Primena toka** (*Stream*).

9.9 Komunikacija u distrib.sistemima

Protokoli visokog nivoa

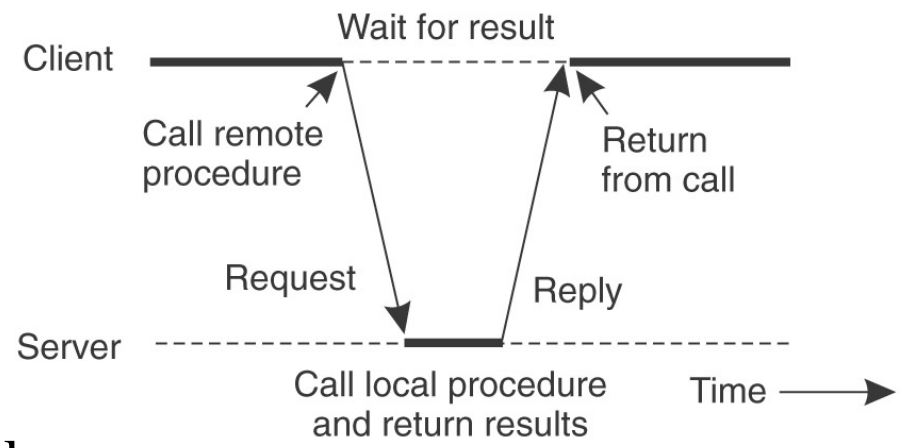
Transportni protokoli

Protokoli niskog nivoa



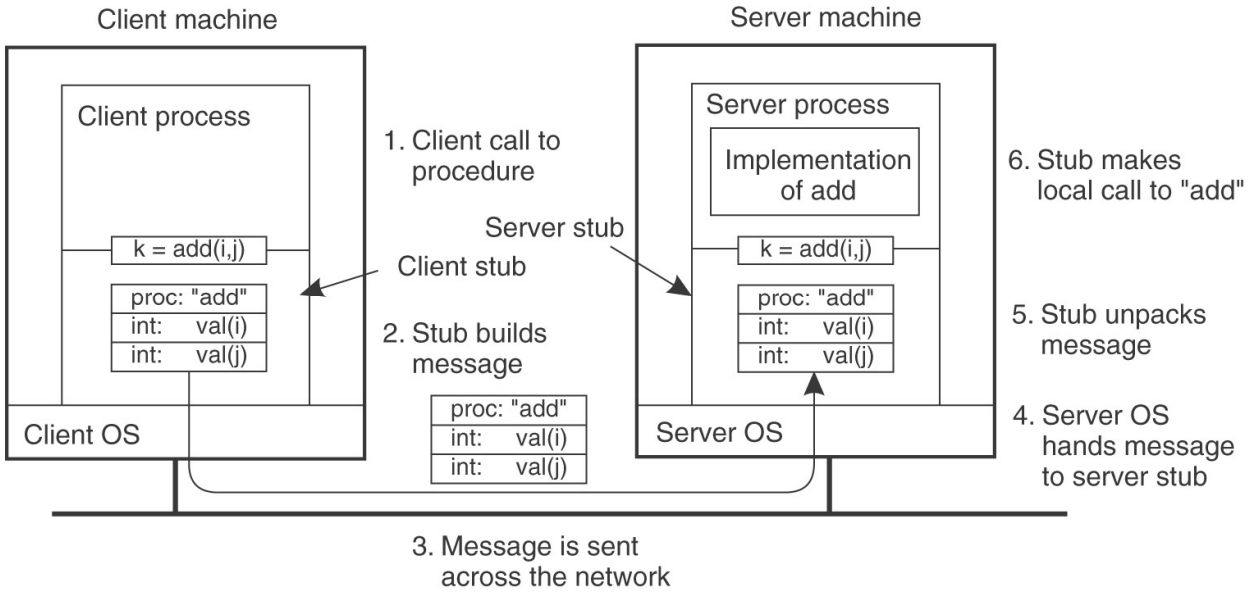
9.9 Remote Procedure Call

- Poziv procedure na udaljenom računaru omogućuje prenos obrade na drugi računar pokretanjem odgovarajuće procedure.
- Za vreme izvođenja procedure na drugom računaru proces koji je inicirao poziv je suspendovan i čeka rezultate izvođenja procedure
- Kada pozvana procedura vrati rezultat, proces na lokalnom računaru nastavlja izvođenje, pri čemu je ceo postupak nevidljiv za programera.
- Primenom RPC poziva omogućavamo da se pozvana procedura na udaljenom računaru izvede na način kao da lokalni proces izvodi proceduru na lokalnom računaru.
- Ostvaruje se upotrebom posebnog tipa zapisa procedura (***client stub***)
- Kada udaljeni računar primi poruku sa pozivom procedure prosleđuje je posebnom zapisu (***server stub***) u *stack* memoriji.
- Kada se procedura izvede sledi prenos rezultata klijentu i nastavak izvođenja procesa na klijent računaru.



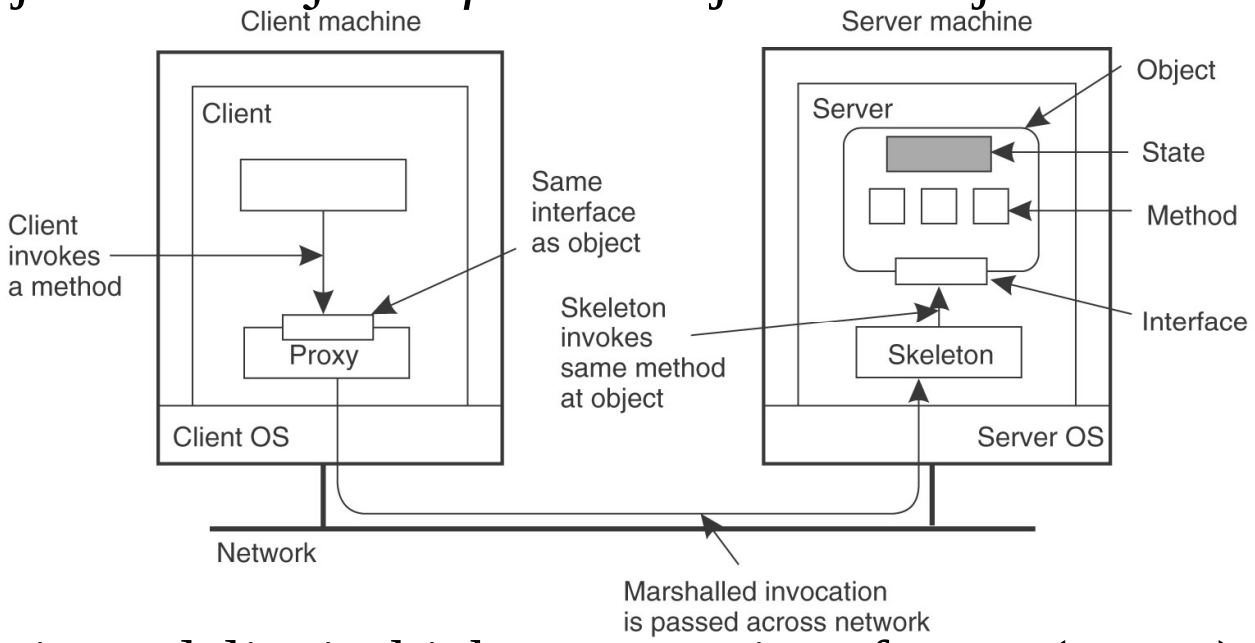
9.9 Remote Procedure Call

1. klijent proces poziva proceduru i generiše klijent stub,
2. klijent stub generiše poruku i prosleđuje je lokalnom OS,
3. klijent OS šalje poruku serveru,
4. OS na serveru prima poruku i prosleđuje je server stubu,
5. server stub analizira parametre poruke i geneše sistemski poziv
6. server izvodi poziv i vraća rezultat server stubu,
7. server stub pakuje poruku i prosleđuje je server OS,
8. server OS šalje poruku klijentu
9. klijent OS poruku prosleđuje klijent stubu,
10. klijent stub raspakuje poruku i prosleđuje rezultat klijent procesu.



9.9 Remote Method Invocation

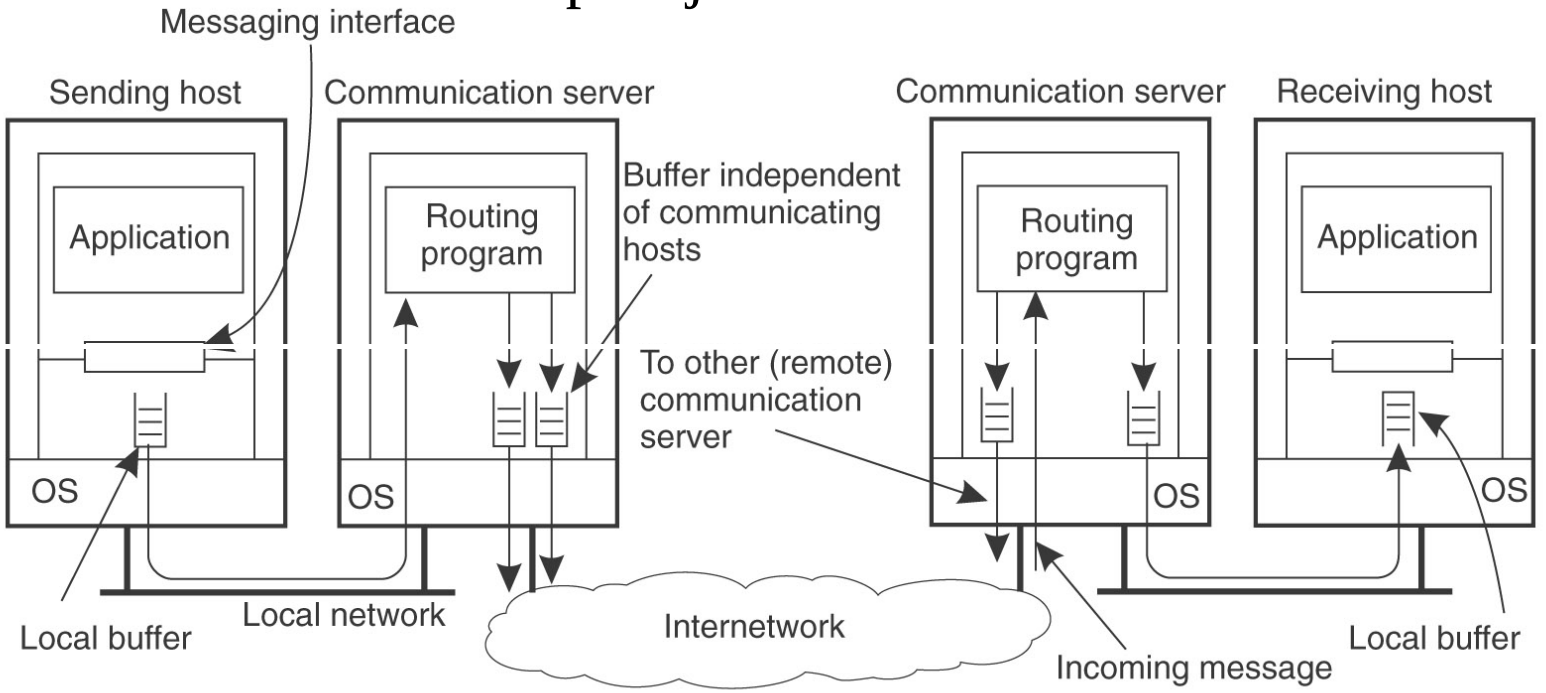
- Zasniva se na primeni distribuiranih objekata koji su opisani podacima koje oni sadrže (***state***) i operacijama nad tim podacima (***method***)
- Metode su DS dostupne primenom veze interfejsa (***interface***).
- DS dozvoljava smeštaj *interface*-a na jedan a objekta na drugi računar



- Klijent poziva udaljeni objekat putem interface-a (***proxy***), koji zahtev prosleđuje serveru gde se primenom ***skeleton***-*a* prosleđuje objektu.
- Nakon što klijent pristupi objektu on pokreće izvođenje metode
- *Interface*-i mogu biti definisani u trenutku izrade aplikacije na klijentu-***static invocation*** a ako nisu unapred poznati - ***dinamic invocation***

9.9 Međusloj za prenos poruka

- Porukama se izbegava ograničenje blokiranja klijent procesa pri RPC ili RMI pozivu na serveru.
- Primer primene poruka je slanje elektronske pošte
- Aplikacija generiše poruku koja se prosleđuje putem lokalne mreže prema komunikaciskom serveru (*mail* serveru).
- Komunikaciski server prosleđuje poruku drugom komunikaciskom serveru koji prihvata poruku i prosleđuje aplikaciji korisnika u trenutku kada se korisnik priključi na komunikaciski server.



9.9 Tok (Stream)

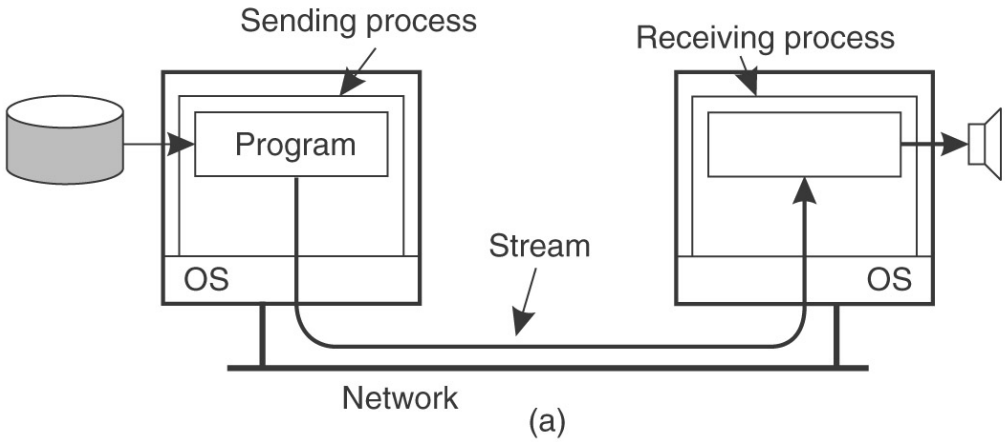
- Tok (*stream*) se koristi u radu sa multimedijalnim dokumentima gde je vrlo važna usklađenost između sadržaja i načina interpretacije sadržaja
- Razlikujemo kontinualne i diskretne medije.
- Za prikaz kontinualnih sadržaja važna je brzina prenosa podataka dok za diskretne medije brzina nema ključnu važnost.
- Za implementaciju kontinualnih medija u distribuiranim sistemima koristi se tok podataka (*data stream*).
- Za asinhroni transmisioni način prenošenja podaci se prenose jedan za drugim a koriste se kada je prenos završen (prenos slika-fotografija).
- Za sinhroni transmisioni način prenošenja podaci se prenose jedan za drugim i moraju se preneti pre određenog intervala vremena.
- Za izohrone transmisione načine prenošenja podaci se prenose jedan za drugim, imaju tačno određen vremenski interval prenosa, tako da se podaci ne smeju preneti posle ali ni pre dozvoljenog intervala vremena

Tok može biti:

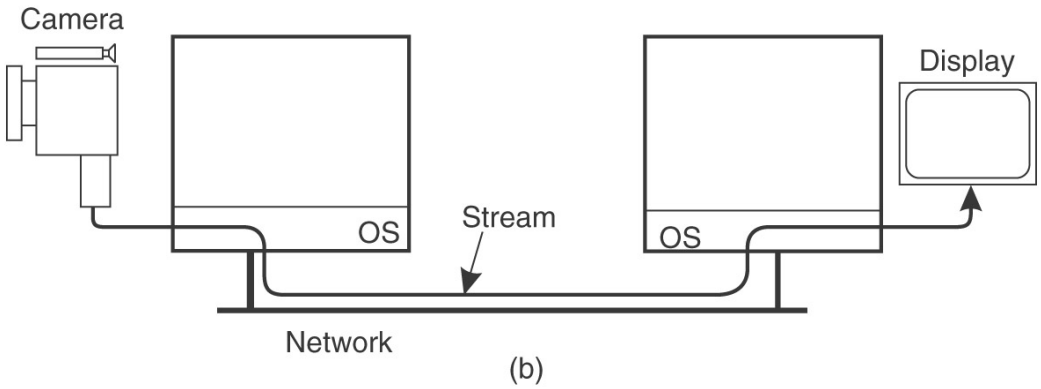
1. **jednostavan** – jedan niz podataka
2. **složen** – nekoliko nizova podataka (kombinacija slike i zvuka).

9.9 Tok (Stream)

Tok može biti izveden korišćenjem dva procesa,

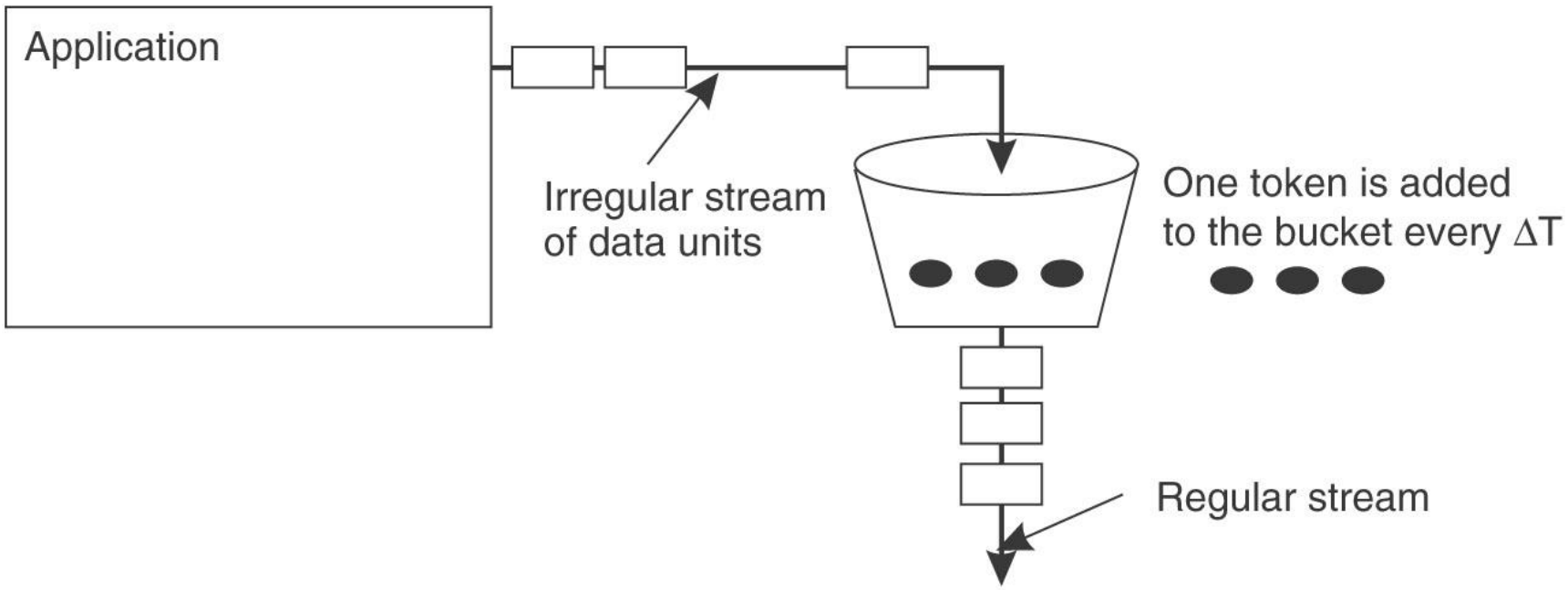


a može biti realizovan i direktno



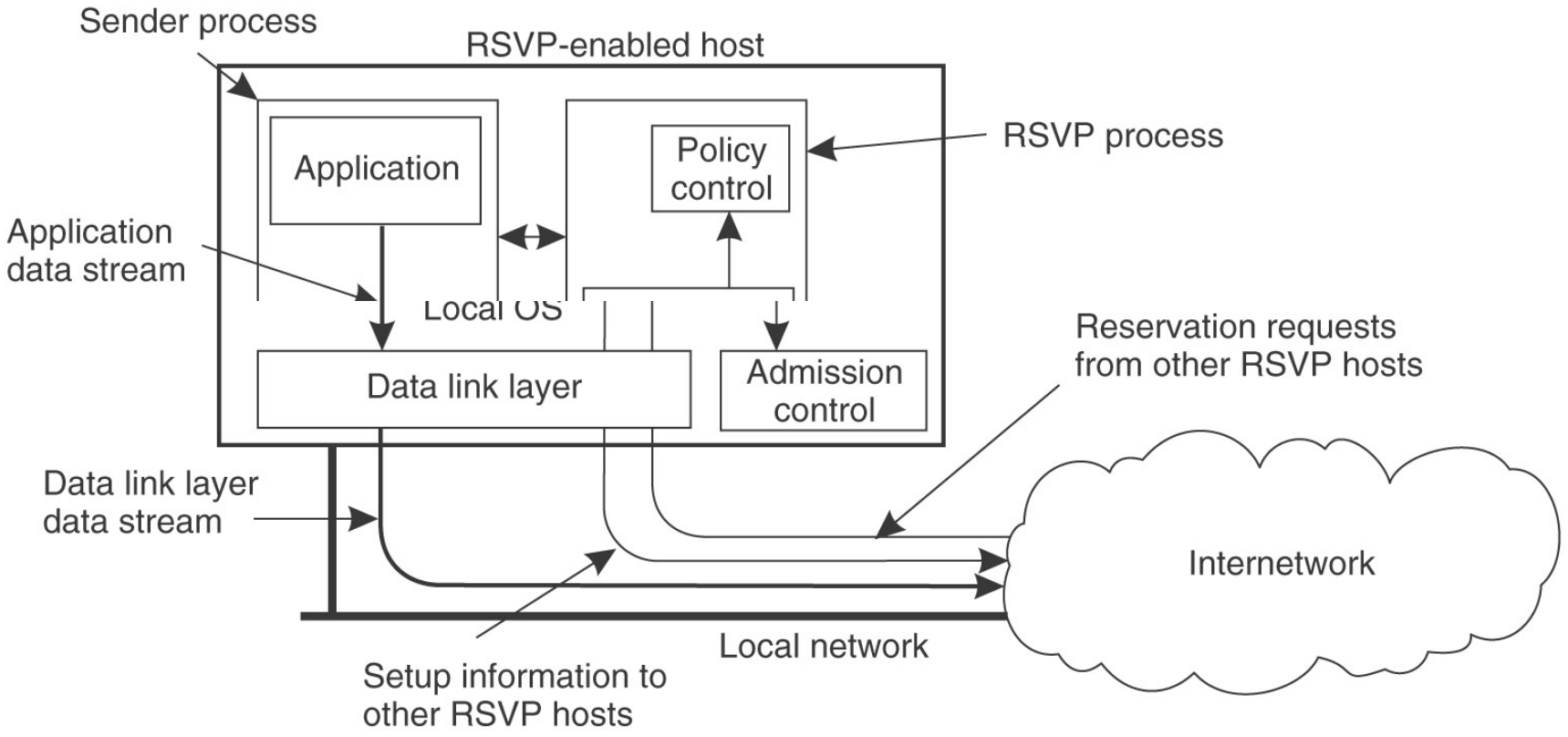
9.9 Tok (Stream)

- Kvalitet toka definiše se specifikacijama za prenos podataka sa podacima o brzini prenosa, problemima pri prenosu itd.
- Primer poboljšanja kvaliteta toka je ***Token bucket*** algoritam
- Aplikacija generiše podatke za tok u nepravilnim vremenskim intervalima, a filter korišćenjem brojača vremena DT šalje podatke u tačnim vremenskim intervalima.



9.9 Tok (Stream)

- Postavljanje toka podataka izvodi se primenom ***R**esurse re**S**er**V**ation **P**rotocol*-*a* (RSVP)
- On služi kao kontrolni protokol koji je smešten u transportni sloj.
- RSVP sadrži sve podatke potrebne za realizaciju toka podataka i preuzima kontrolu nad tokom podataka

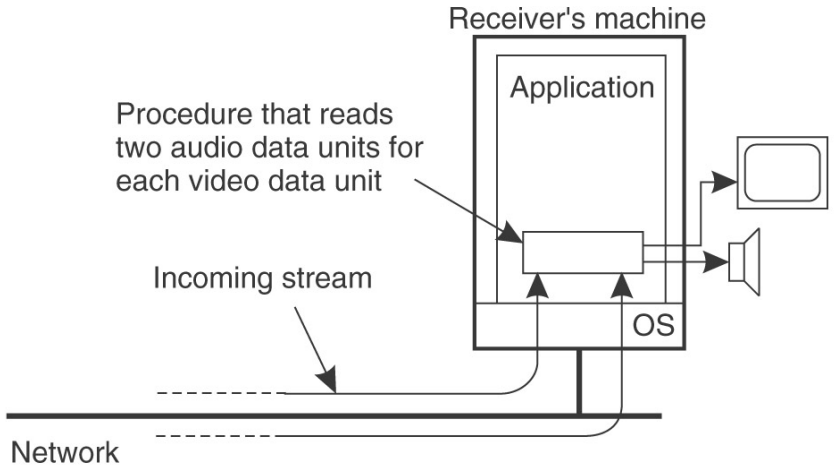


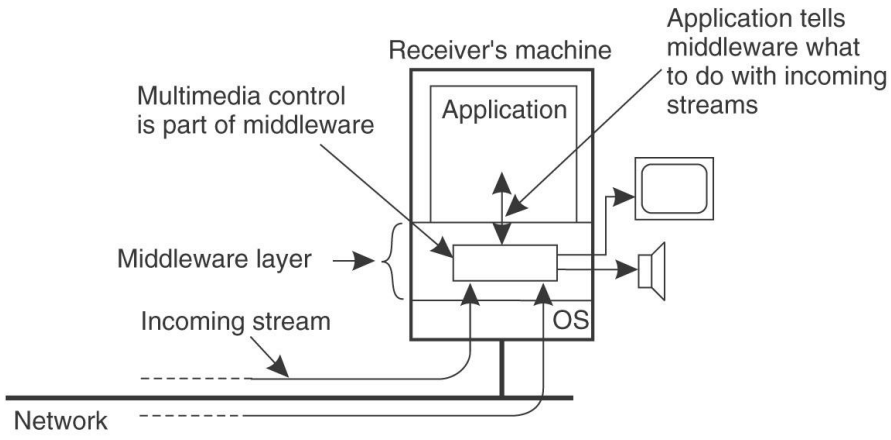
9.9 Tok (Stream)

- Za tok podataka bitna je sinhronizacija, pa je obavezno koristiti neki sinhronizacioni mehanizam.
- Zadatak mehanizma je da prihvati dolazni tok podataka i da vremenski izvrši usklađivanje toka podataka koji se prosleđuje prema primaocu.

Kontrolni mehanizam može biti:

1. **Jednostavan** - prihvata podatke i vrši sihronizaciju pri prosleđivanju podataka primacu
2. **Složen** - koristi se poseban program (aplikacija) koja je namenjena kontroli toka podataka





Hvala na pažnji !!!

Pitanja

? ? ?