

Arhitektura softvera u distribuiranim sistemima

Arhitekturalni stilovi

- Arhitekturalni stil se formuliše u vidu:
 - (zamenljivih) **komponenti** sa dobro definisanim interfejsima
 - **konektora** tj. načina na koji se komponente međusobno povezuju
 - **podataka** koji se razmenjuju između komponenti
 - načina kako se komponente i konektori zajedno konfigurišu u **sistem**
- **Komponenta** (engl. *component*) je modularna jedinica sa dobro definisanim interfejsima koja je zamenljiva unutar svog okruženja
- **Konektor** (engl. *connector*) je mehanizam koji vrši medijaciju prilikom komunikacije, koordinacije i saradnje između komponenti

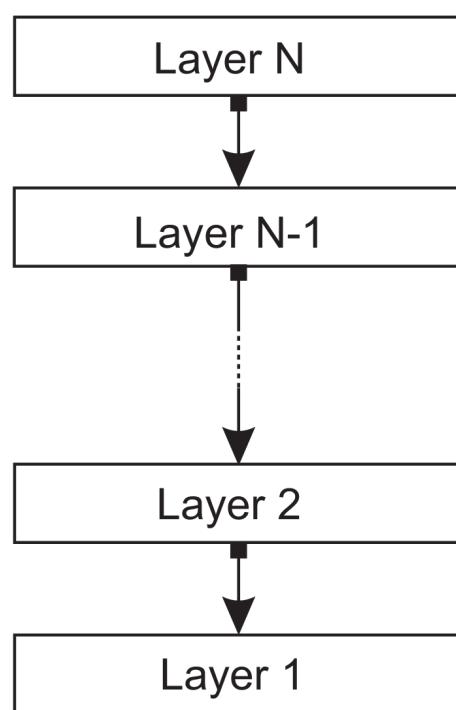
Arhitekturalni stilovi

- **Slojevite arhitekture** (engl. *layered architectures*)
- **Arhitekture zasnovane na objektima** (engl. *object-based architectures*)
- **Arhitekture zasnovane na resursima** (engl. *resource-based architectures*)
- **Arhitekture zasnovane na događajima** (engl. *event-based architectures*)

Slojevite arhitekture

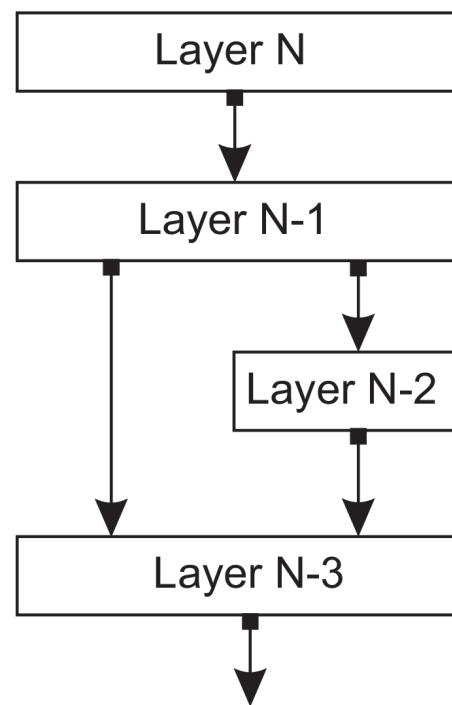
(a) Čista slojevita

Request/Response
downcall

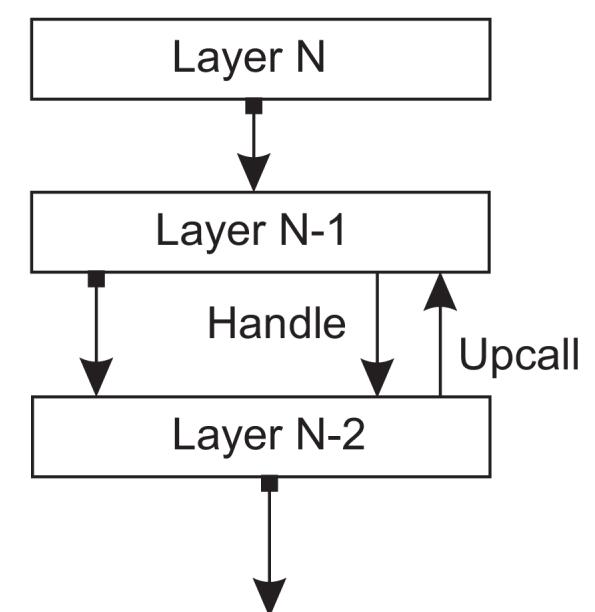


(b) Mešani slojevi

One-way call



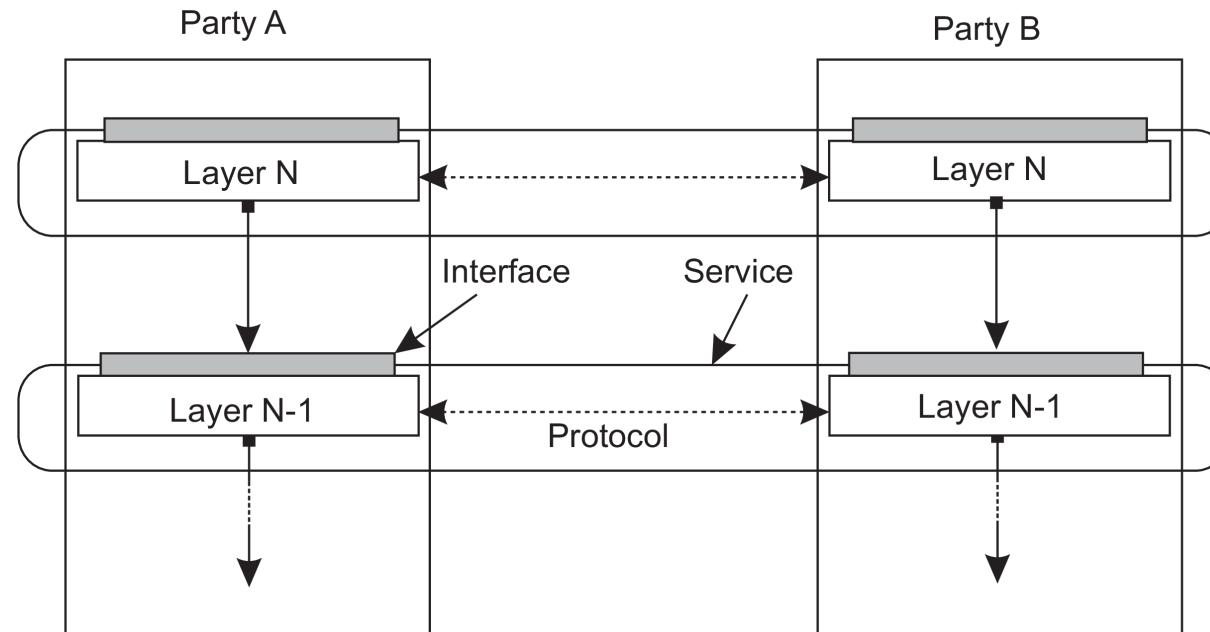
(c) Slojevita sa
pozivima nagore
(primer: ručke kod OS)



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

Primer: komunikacioni protokoli

- **Protokol** je skup pravila koje strane moraju ispoštovati kako bi razmenile informacije
- **Interfejs** specificira koje se funkcije mogu pozivati
- **Servis** pruža usluge slanja podataka do jednog ili više odredišta



- Važno je praviti razliku između **servisa** koje nudi sloj, **interfejsa** pomoću kojih su servisi dostupni, i **protokola** koje sloj implementira kako bi uspostavio komunikaciju

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

Dvostrana komunikacija: echo server

Server

```
1 from socket import *
2 s = socket(AF_INET, SOCK_STREAM)
3 (conn, addr) = s.accept() # returns new socket and addr. client
4 while True: # forever
5     data = conn.recv(1024) # receive data from client
6     if not data: break # stop if client stopped
7     conn.send(str(data)+"*") # return sent data plus an "*"
8 conn.close() # close the connection
```

Client

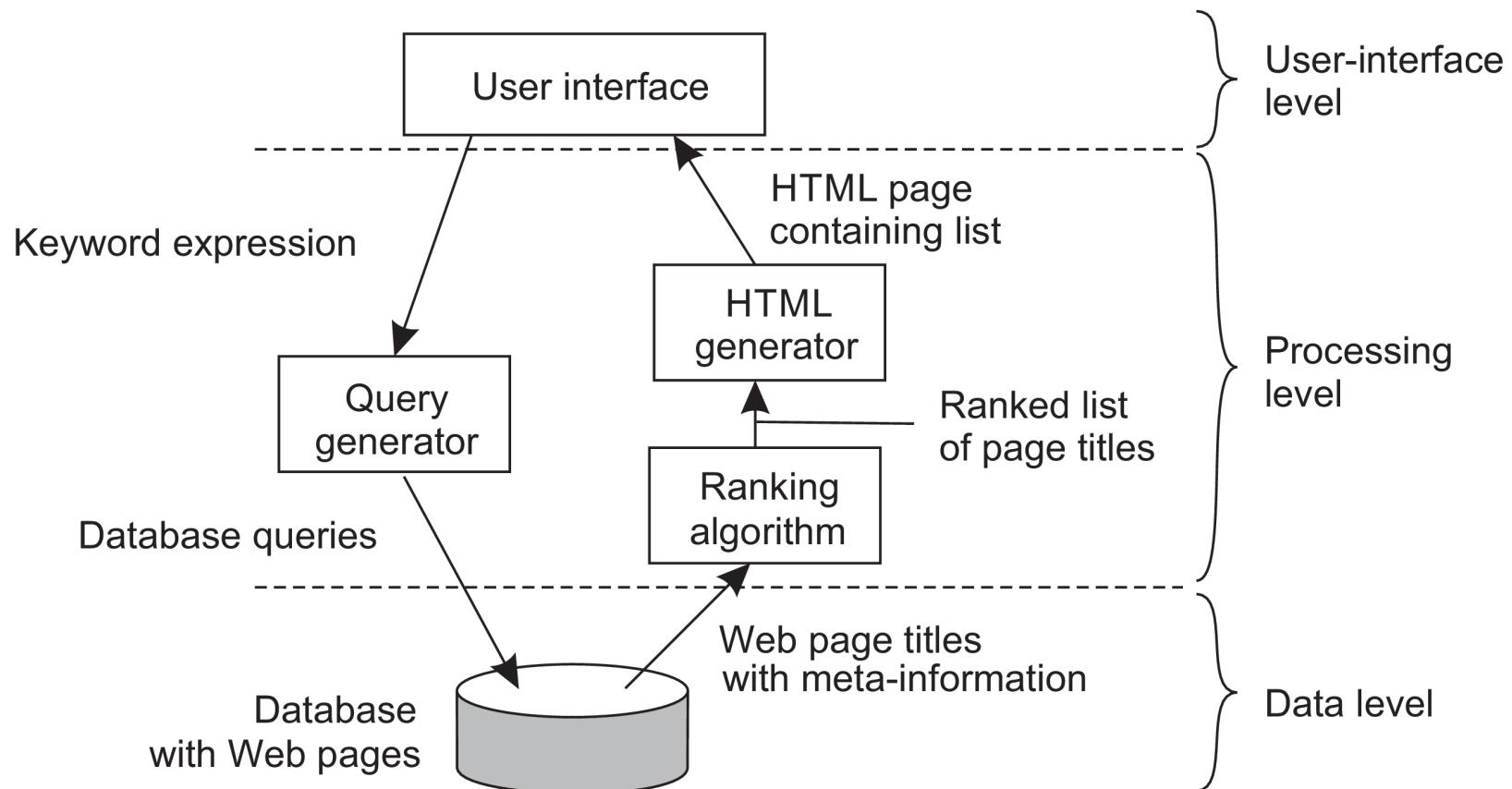
```
1 from socket import *
2 s = socket(AF_INET, SOCK_STREAM)
3 s.connect((HOST, PORT)) # connect to server (block until accepted)
4 s.send('Hello, world') # send some data
5 data = s.recv(1024) # receive the response
6 print data # print the result
7 s.close() # close the connection
```

Aplikacioni slojevi

- Tradicionalni troslojni logički pogled:
 - **Sloj interfejsa aplikacije** (engl. *application-interface layer*) sadrži jedinice interfejsa ka korisnicima ili eksternim aplikacijama
 - **Sloj obrade** (engl. *processing layer*) sadrži funkcije aplikacije bez specifičnih podataka
 - **Sloj podataka** (engl. *data layer*) sadrži podatke kojima klijent želi da manipuliše putem komponenti aplikacije
- Opažanje:
 - Tradicionalna slojevita arhitektura se tipično nalazi u mnogim distribuiranim informacionim sistemima, koristeći pri tom klasičnu tehnologiju baza podataka i pratećih aplikacija

Aplikacioni slojevi

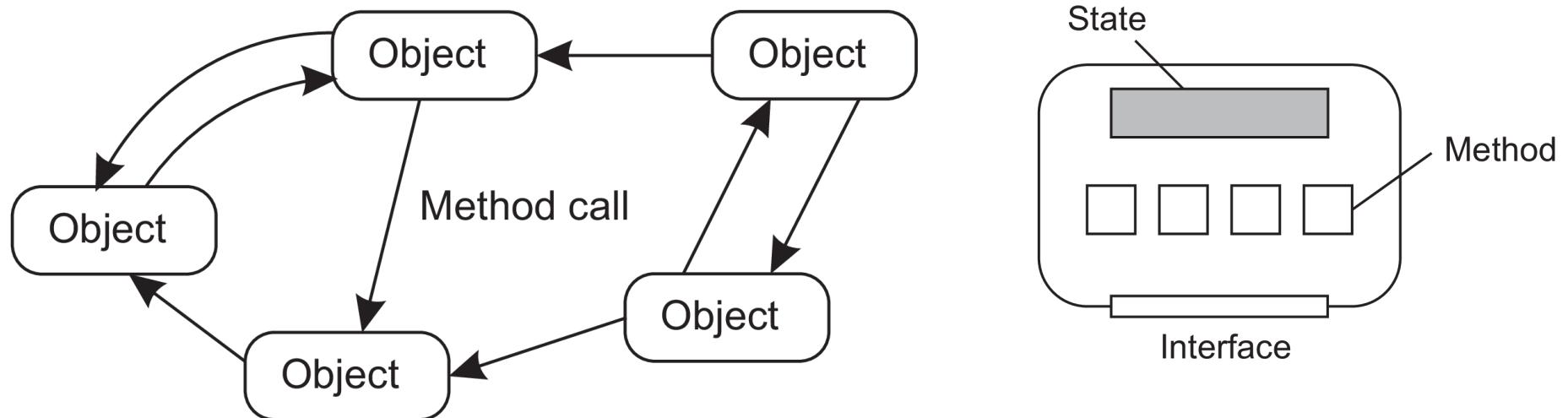
- Primer: jednostavan sistem za pretraživanje (engl. search engine)



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

Arhitekture zasnovane na objektima

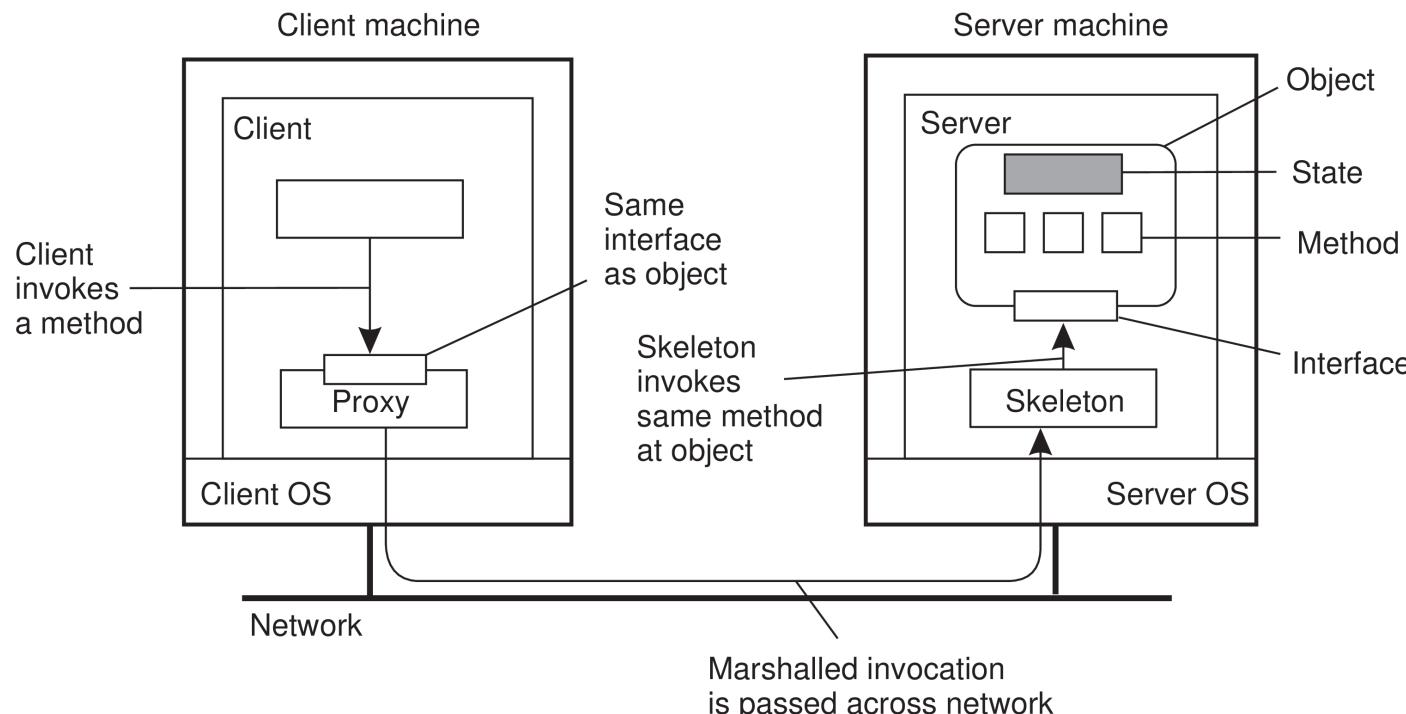
- **Komponente su objekti, međusobno povezani pozivima procedura.** Objekti se mogu nalaziti na različitim mašinama, s toga se pozivi mogu izvršavati preko mreže; **interfejs** može biti **na jednoj**, a **objekat na drugoj mašini – distribuirani objekat**
- **Enkapsulacija:** objekti **enkapsuliraju podatke** i nude **metode za rad sa tim podacima** bez otkrivanja interne implementacije



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

Arhitekture zasnovane na objektima

- Tipična organizacija udaljenog objekta sa proksijem na klijentskoj strani
 - **proksi** je implementacija interfejsa objekta koja se učitava u klijentski adresni prostor (odgovara klijent stub-u kod RPC), na serverskoj strani je **skeleton** koji omogućava serverskom midlveru da pristupi korisničkim objektima, **maršaling** je proces transformacije memorijske reprezentacije objekta u format podataka pogodan za slanje



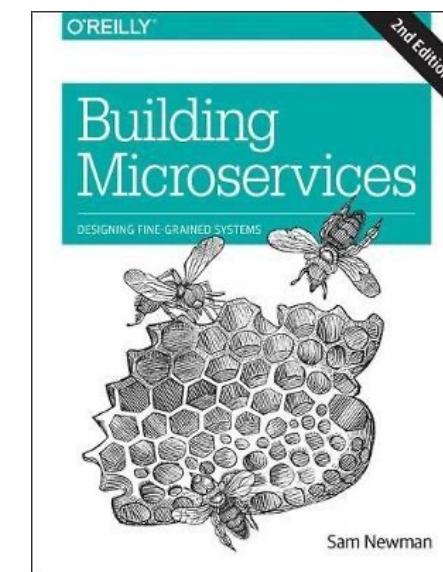
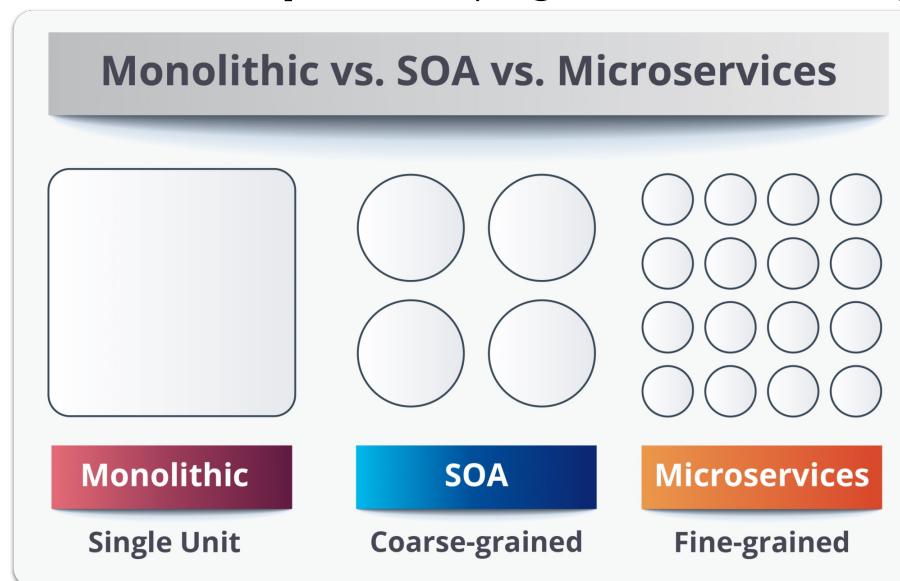
Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017/>

Servisno-orientisane arhitekture

- Arhitekture zasnovane na objektima pružaju osnovu za **enkapsulaciju servisa u nezavisne jedinice**
- **Enkapsulacija** ovde igra ključnu ulogu: servis kao celina se realizuje kao samostalni entitet, iako može koristiti i druge servise
- Jasnim razdvajanjem različitih servisa tako da mogu da rade nezavisno, dolazi se do **servisno-orientisanih arhitektura (SOA – engl. service-oriented architectures)**
- Kod SOA, distribuirana aplikacija ili sistem se u suštini konstruiše putem slaganja različitih servisa, koji se mogu nalaziti u različitim organizacijama
- Na ovaj način, problem razvoja distribuiranog sistema se delom svodi na **slaganje servisa** (engl. *service composition*) i osiguravanje da ti servisi rade u harmoniji. Ovaj problem je potpuno analogan pitanjima kod integracije poslovnih aplikacija (engl. *enterprise application integration*)

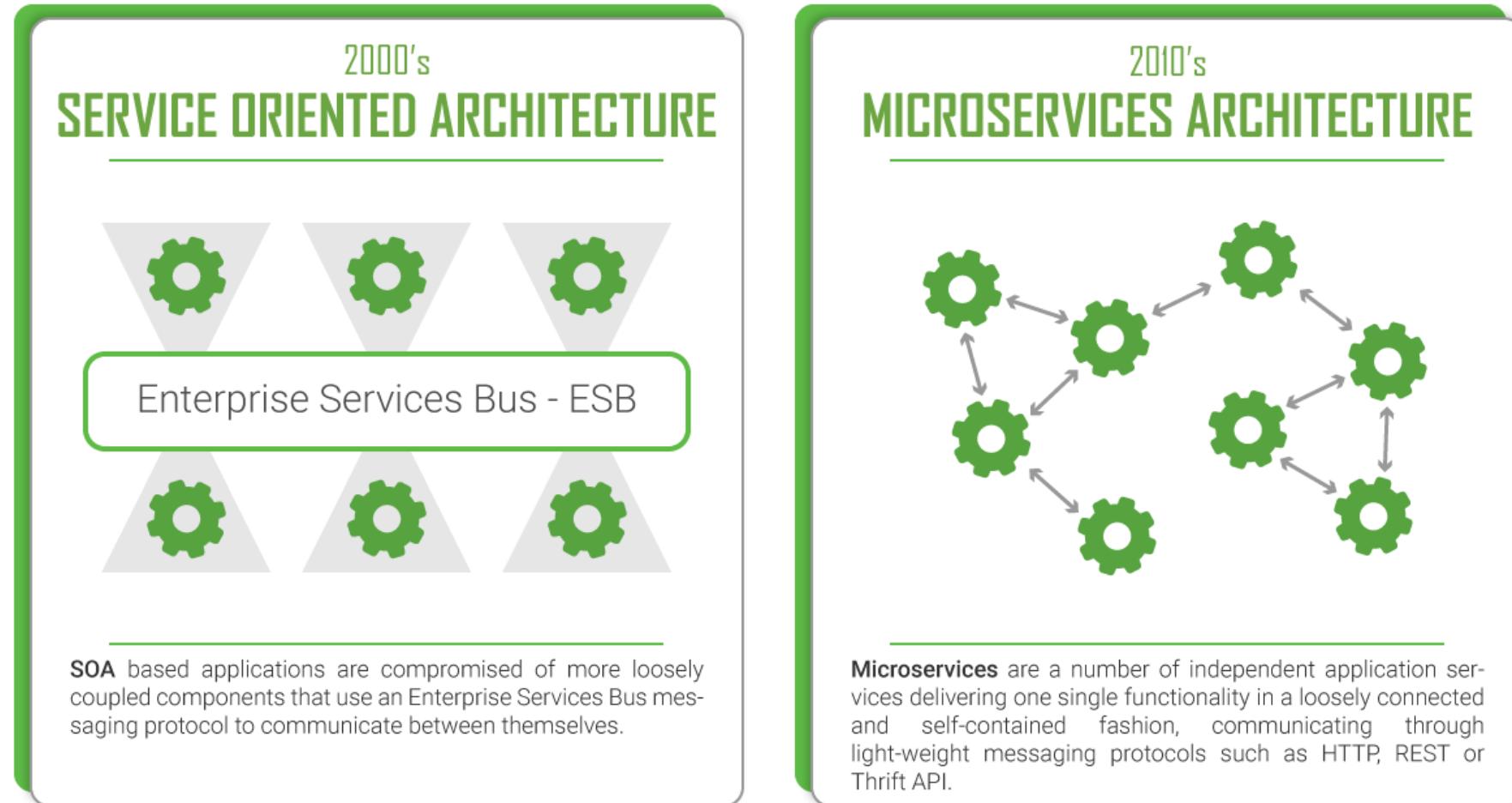
SOA i mikroservisi

- **Mikroservisi su tehnika razvoja softvera**, podvrsta SOA arhitekturalnog stila, kojom se aplikacija strukturira kao **skup slabo spregnutih servisa**. U mikroservisnim arhitekturama, servisi su fine rezolucije (engl. *fine-grained*), a protokoli su lagani (engl. *lightweight*)
- Mikroservisna arhitektura omogućava da mali autonomni timovi paralelno i nezavisno razvijaju, isporučuju i skaliraju svoje servise – **kontinualna integracija i kontinualna isporuka** (engl. *continuous integration/continuous delivery CI/CD*)



Izvor: <https://dzone.com/articles/microservices-vs-soa-is-there-any-difference-at-all>

SOA i mikroservisi



Izvor: <https://dzone.com/articles/microservices-vs-soa-is-there-any-difference-at-all>

Arhitekture zasnovane na resursima

- **Distribuirani sistem** se posmatra kao **skup resursa** kojima **pojedinačno upravljaju komponente**
- **Resursi** se mogu dodavati, brisati, pribavljati i modifikovati od strane (udaljenih) aplikacija – **Representational State Transfer (REST)**, karakteristike **RESTful arhitektura**:
 1. Resursi se identifikuju putem **jedinstvene šeme imenovanja** (engl. *naming scheme*)
 2. **Svi servisi pružaju isti interfejs**
 3. **Poruke poslate ka i od servisa su potpuno samo-opisujuće**
 4. Nakon izvršenja operacije od strane servisa, komponenta zaboravlja sve o pozivaocu (engl. *caller*) – **izvršavanje bez pamćenja stanja** (engl. *stateless execution*)
- **Osnovne operacije:**

Operacija	Opis
PUT	Stvaranje novog resursa
GET	Pribavljanje stanja resursa u nekoj reprezentaciji
DELETE	Brisanje resursa
POST	Izmena resursa prenosom novog stanja

Primer:Amazon Simple Storage Service (S3)

- **Objekti** (tj. fajlovi) se smeštaju u **kofe** (engl. *buckets*), tj. direktorijume.
Kofe se ne mogu smestiti u kofe
- Kako bi se izvršile operacije nad objektom ImeObjekta u kofi ImeKofe
neophodan je sledeći identifikator:

`http://ImeKofe.s3.amazonaws.com/ImeObjekta`
- Sve **operacije** se izvode **slanjem HTTP zahteva**:
 - Kreiranje kofe ili objekta: PUT zajedno sa URI-jem (engl. *Uniform Resource Identifier*) – **uniformni identifikator resursa** je string koji jednoznačno identificuje neki resurs
 - Listanje objekata: GET nad imenom kofe
 - Čitanje objekta: GET nad celim URI-jem



REST i SOAP interfejsi

- RESTful pristup je popularan zato što je interfejs ka servisima vrlo jednostavan. Nedostatak je što zahteva puno posla u parametarskom prostoru
- Tradicionalni pristup – Amazon S3 Simple Object Access Protocol (SOAP) interfejs, za komunikaciju se koristi XML
- Amazon S3 SOAP interfejs:

Operacije nad kofama	Operacija nad objektima
ListAllMyBuckets	PutObjectInline
CreateBucket	PutObject
DeleteBucket	CopyObject
ListBucket	GetObject
GetBucketAccessControlPolicy	GetObjectExtended
SetBucketAccessControlPolicy	DeleteObject
GetBucketLoggingStatus	GetObjectAccessControlPolicy
SetBucketLoggingStatus	SetObjectAccessControlPolicy

REST i SOAP interfejsi

- Upotreba interfejsa bucket koji ima operaciju `create`, koja zahteva ulazni string `mybucket`, kako bi se kreirala kofa pod nazivom “`mybucket`” primenom SOAP i REST interfejsa:

SOAP

```
import bucket  
bucket.create("mybucket")
```

RESTful

```
PUT "http://mybucket.s3.amazonaws.com/"
```

- Prvim pristupom greške se mogu uočiti u vreme prevodenja, dok je kod drugog to moguće samo u vreme izvršavanja. Specifikacija semantike operacije je lakša kod SOAP, dok se REST generički pristup lakše prilagođava promenama

Arhitekture zasnovane na događajima

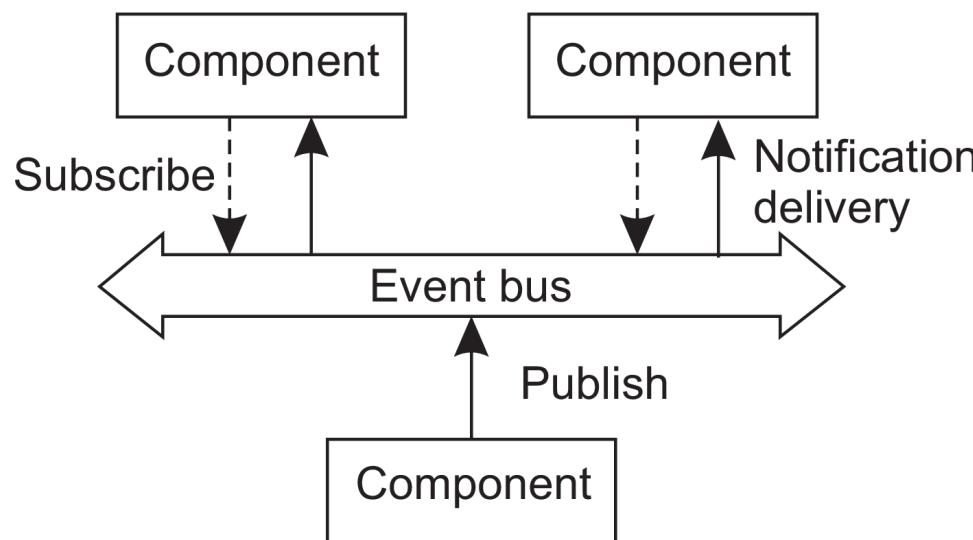
- Arhitekture zasnovane na događajima poznate su i kao **objavi-preplati se ili izdavač-preplatnik** (engl. *publish-subscribe*) **arhitekture**
- Distribuirani sistemi sa arhitekturom kod koje postoji jako razdvajanje između **obrade** i **koordinacije**, ideja je da se **sistem** posmatra kao **skup procesa** koji se **nezavisno izvršavaju**, **koordinacija** ovde obuhvata **komunikaciju i koordinaciju** procesa
- Tipovi koordinacije između procesa:

	Temporalno spojeni	Temporalno razdvojeni
Referencijalno spojeni	direktna	sanduče
Referencijalno razdvojeni	zasnovana na događajima	sa deljenim prostorom podataka

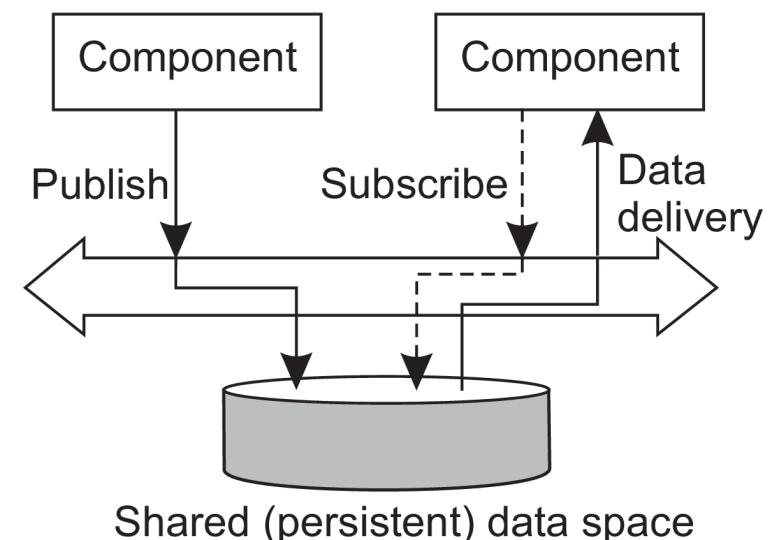
Arhitekture zasnovane na događajima

- Koordinacija kod referencijalno razdvojenih procesa:

(a) **zasnovana na događajima**



(b) **sa deljenim prostorom podataka**



- Procesi komuniciraju putem torki (engl. *tuples*)
- Asocijativni mehanizam pretrage torki

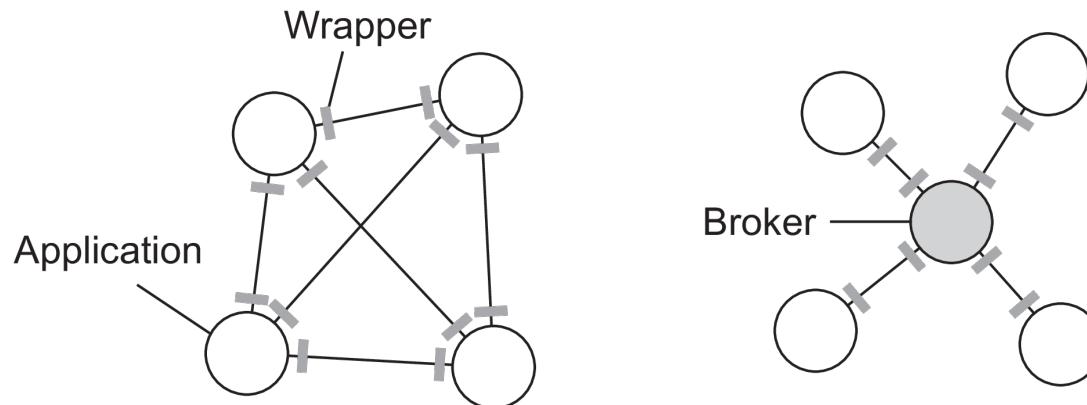
Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017/>

Organizacija midlvera

- Dva glavna tipa **projektnih obrazaca** koja se koriste prilikom **organizacije midlvera** su **omotači** (engl. *wrappers*) i **presretači** (engl. *interceptors*)
- Omotači i presretači omogućavaju **proširenje i adaptaciju midlvera**, **promenljivost midlvera** je jako važna u savremenim distribuiranim sistemima
- Glavni cilj organizacije midlvera je **otvorenost**, koja se u najboljem slučaju postiže slaganjem midlvera u vreme izvršavanja
- Tipičan problem je da interfejs stare komponente najverovatnije ne odgovara svim aplikacijama
- Rešenje je primena omotača tj. adaptera koji nudi interfejs prihvatljiv klijentskoj aplikaciji, funkcije adaptera se transformišu u one dostupne na komponenti i time se rešava problem nekompatibilnih interfejsa

Omotači

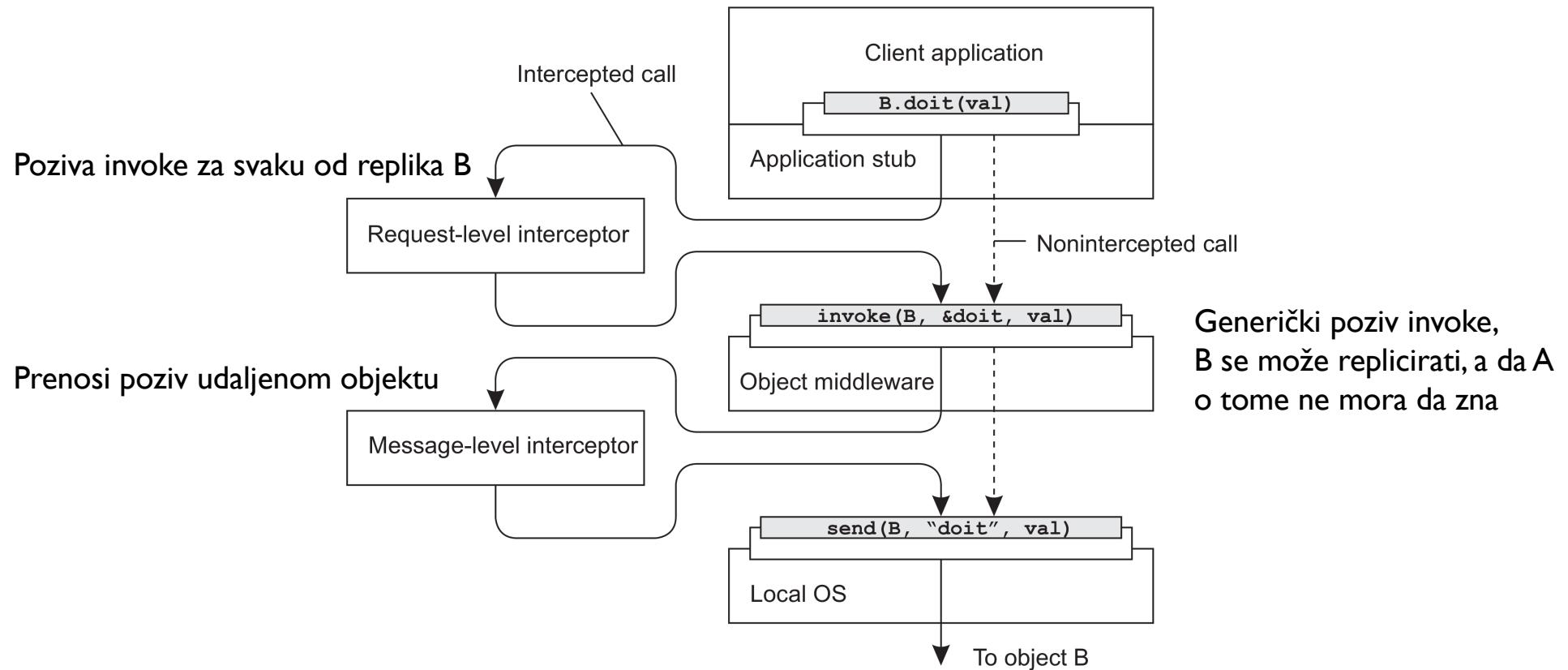
- Dva moguća pristupa: I-na-I ili putem brokera
 - I-na-I zahteva da svaka aplikacija ima omotač za svaku drugu aplikaciju
 - putem brokera: aplikacije šalju zahteve sa informacijom šta im je potrebno, broker ima sve relevantne informacije o svim aplikacijama, kontaktira odgovarajuću aplikaciju, po potrebi transformiše i kombinuje odgovore i vraća rezultat inicijalnoj aplikaciji
 - Složenost:
 - Kod I-na-I neophodno je $O(N^2)$ omotača, dok broker omogućava $O(N)$ omotača



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

Presretači

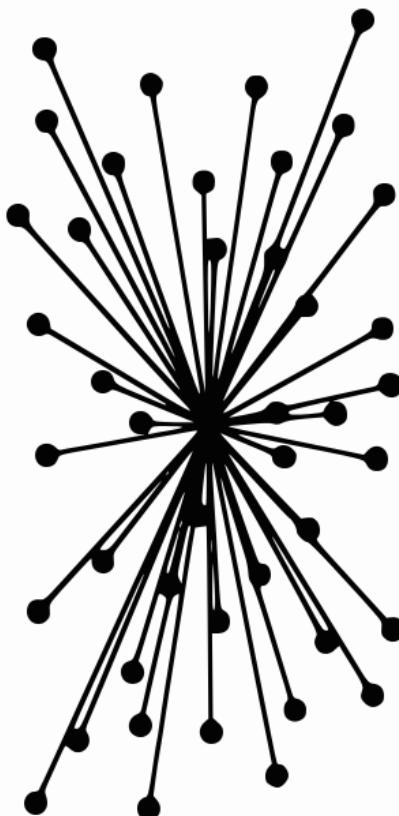
- Sredstvo za **adaptaciju midlvera specifičnim potrebama** aplikacija
- Objekat A može da pozove metodu koji pripada objektu B, iako se nalaze na različitim mašinama



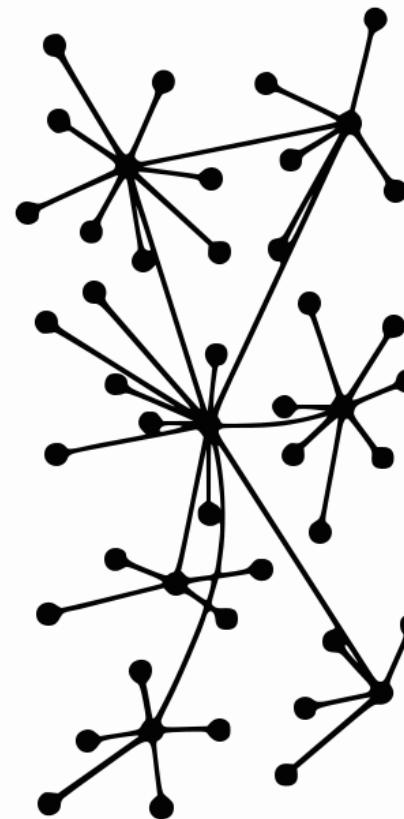
Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017/>

Arhitekture distribuiranih sistema

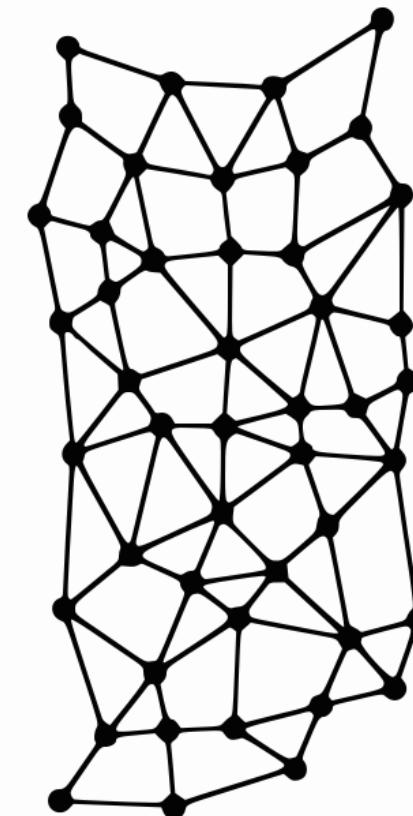
Organizacija sistema



Centralized



Decentralized



Distributed

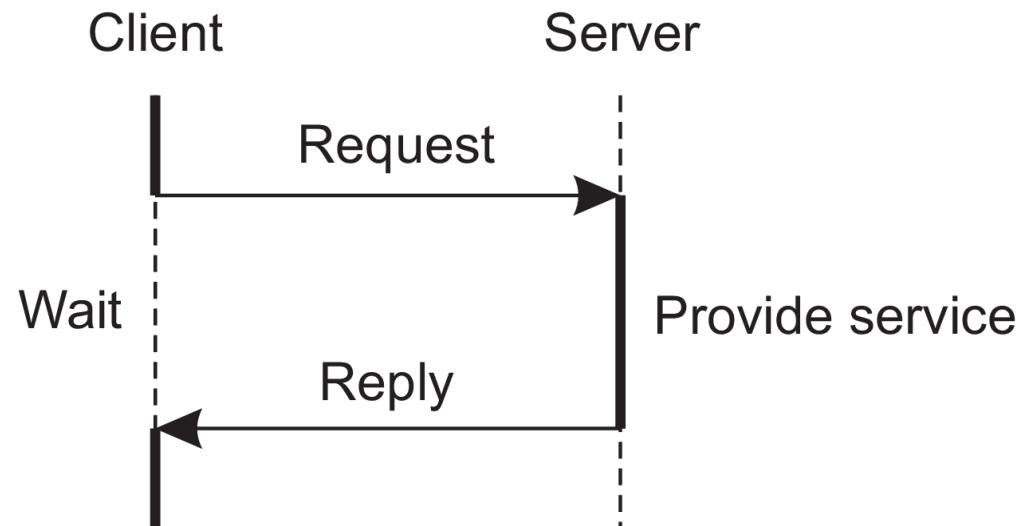
Izvor: Centralized, decentralized and distributed network models, Paul Baran (1964)

Arhitektura sistema

- Arhitektura sistema je instanca softverske arhitekture sa određenim softverskim komponentama, njihovom interakcijom i rasporedom
- **Centralizovane organizacije**
 - Klijent-server arhitektura
 - Višeslojne (engl. *multitiered*) arhitekture
- **Decentralizovane organizacije – peer-to-peer (P2P) sistemi**
 - Strukturirani P2P sistemi (distribuirane heš tabele)
 - Nestrukturirani P2P sistemi
 - Plavljenje (engl. *flooding*)
 - Slučajne šetnje (engl. *random walks*)
 - Hijerarhijski organizovane P2P mreže (primer: Skype)
- **Hibridne organizacije**
 - Sistemi sa serverima na ivici (engl. *edge-server*)
 - Kolaborativni distribuirani sistemi (primer: BitTorrent)

Jednostavna klijent-server arhitektura

- Postoje procesi koji nude usluge (**serveri**)
- Postoje procesi koji koriste usluge (**klijenti**)
- Klijenti i serveri mogu da se nalaze na različitim mašinama
- Klijenti prate zahtev/odgovor model u odnosu na servise

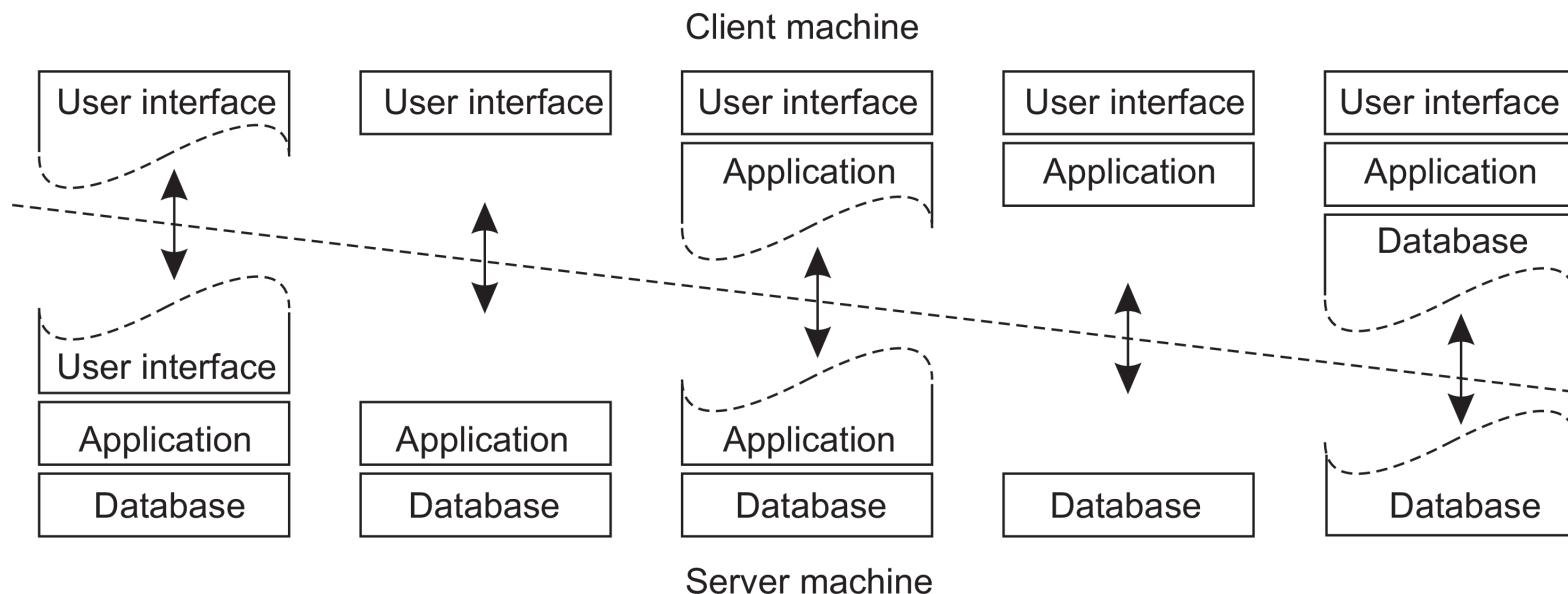


Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017/>

Višeslojne arhitekture

- Tradicionalne fizičke organizacije:
 - **Jednoslojna**: konfiguracija glupi terminal/mejnfrejm
 - **Dvoslojna**: konfiguracija klijent/jedan server
 - **Troslojna**: svaki sloj na posebnoj mašini

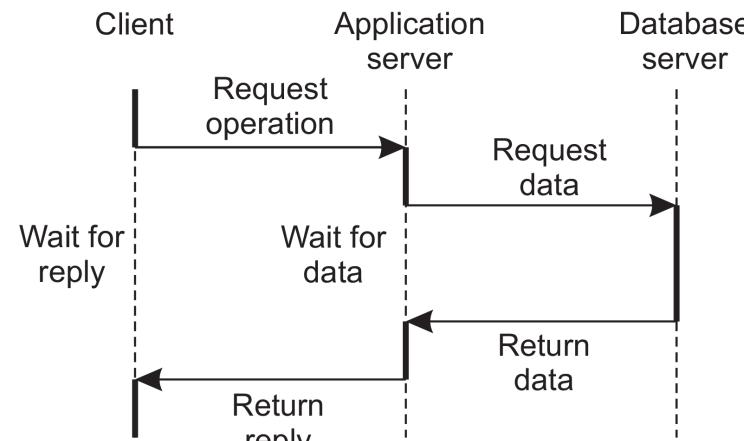
(Fizički) dvoslojna konfiguracija



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

Troslojne arhitekture

- Mašine mogu biti istovremeno i klijenti i serveri
- Primer je arhitektura za **obradu transakcija**, poseban proces (monitor za praćenje obrade transakcija) koordiniše sve transakcije koje se izvršavaju na više servera
- Primer je i **organizacija veb sajtova**. **Veb server** je ulazna tačka za sajt, prosleđuje zahteve **aplikacionom serveru** koji vrši obradu i interaguje sa **serverom baze podataka**
- Primer servera koji se ponaša kao klijent:



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

Decentralizovana organizacija

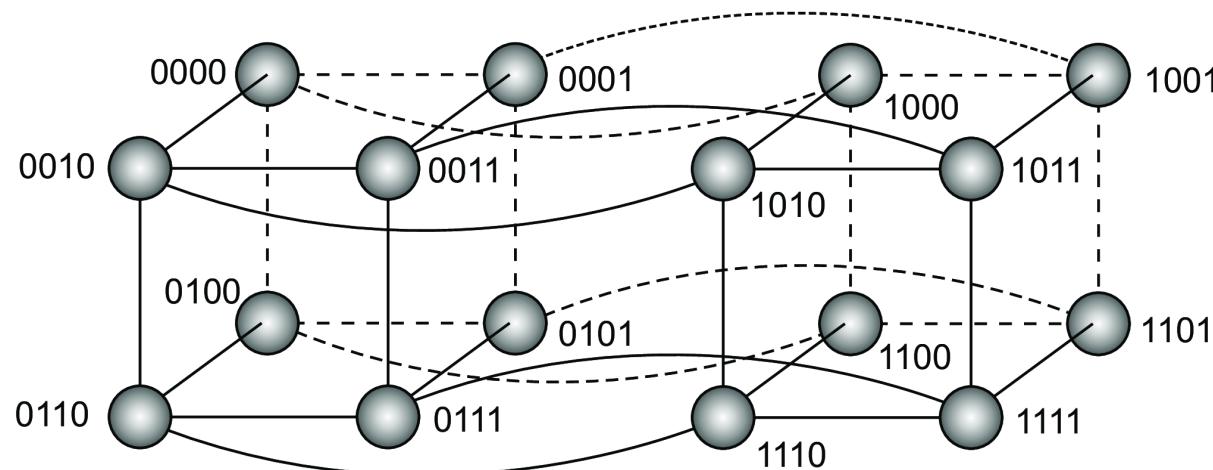
- **Vertikalna distribucija** proizilazi iz podele distribuiranih aplikacija na tri logička sloja i izvršavanja komponenti iz svakog od slojeva na drugom serveru
- **Horizontalna distribucija** omogućava da se klijent ili server mogu fizički podeliti u logički ekvivalentne delove, ali da svaki od delova radi na sopstvenom delu ukupnog skupa podataka – **peer-to-peer (P2P)** sistemi
- Kod **P2P arhitektura svi procesi su ravnopravni**, funkcije koje je neophodno izvršiti predstavljene su svakim od procesa, tj. svaki od procesa ponaša se istovremeno **i kao klijent i kao server** (ponaša se kao **sluga** – engl. *servant*), ključno kako organizovati procese u **prekrivajuću mrežu** (engl. *overlay network*)
- **Čvorovi** prekrivajuće mreže su **procesi**, a **potezi** predstavljaju moguće **komunikacione kanale** (koji se obično realizuju kao TCP konekcije)

Strukturirani P2P sistemi

- Koriste se **semantički slobodni indeksi** (engl. *semantic-free index*): svaka od **jedinica podataka** je na **jedinstven način** pridružena **ključu**, te se koristi kao **indeks**. Obično se koriste **heš funkcije**
$$\text{ključ}(\text{podatak}) = \text{heš}(\text{vrednost podatka})$$
- **P2P sistem** je odgovoran za **čuvanje parova** (**ključ, vrednost**). Svaki od čvorova dobija identifikator iz istog skupa svih mogućih heš vrednosti i svaki od čvorova postaje odgovoran za čuvanje podatak pridruženih određenom podskupu ključeva. Sistem praktično implementira **distribuiranu heš tabelu** (engl. *distributed hash table – DHT*) – primer Chord protokol
- Ovakav P2P sistem pruža efikasnu implementaciju *lookup* funkcije koja mapira **ključ** na **postojeći čvor**: **postojeći čvor** = $\text{lookup}(\text{ključ})$, pri čemu topologija strukturiranog P2P sistema igra značajnu ulogu jer je neophodno efikasno rutiranje *lookup* zahteva do čvora odgovornog za čuvanje podataka pridruženih datom ključu

Strukturirani P2P sistemi

- Jednostavan primer: **hiperkocka** (engl. *hypercube*) – n -dimenzionalna kocka koja povezuje fiksni broj čvorova
- P2P sistem organizovan kao četvoro-dimenzionalna hiperkocka:



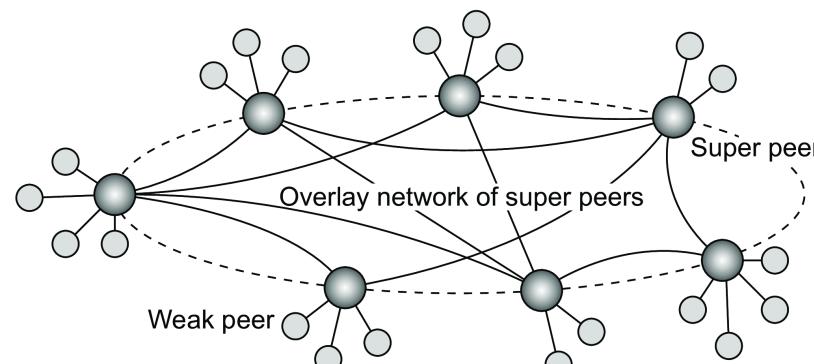
- Traženje d sa ključem $k \in \{0, 1, 2, \dots, 2^4 - 1\}$ podrazumeva rutiranje zahteva čvoru sa identifikatorom k

Nestrukturirani P2P sistemi

- Svaki od čvorova održava **ad hoc listu suseda**. Rezultujuće prekrivanje podseća na **slučajni graf** (engl. *random graph*), **potez** $\langle u, v \rangle$ postoji samo **sa određenom verovatnoćom** $P[\langle u, v \rangle]$. U idealnom slučaju ova verovatnoća je jednaka za sve parove čvorova, ali u praksi se nalazi širok spektar distribucija
- Traženje podataka ne može ići unapred određenom rutom kao kod strukturiranih P2P, već moramo vršiti **pretragu**:
 - **Plavljenjem** (engl. *flooding*): čvor u predaje zahtev d svim susedima. Zahtev se ignoriše od strane primajućeg čvor ako ga je već video ranije. U suprotnom, v traži lokalno d (rekurzivno)
 - **Slučajnom šetnjom** (engl. *random walk*): čvor u predaje zahtev d slučajno izabranom susedu v . Ako v ne poseduje d , prosleđuje zahtev jednom od svojih slučajno izabranih suseda i tako redom
- **Slučajna šetnja je komunikaciono efikasnija**, ali joj može trebati **više vremena da pronađe rezultat**

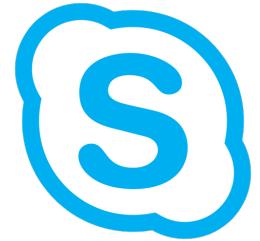
Hijerarhijski organizovane P2P mreže

- Ponekad ima smisla razbiti simetriju u P2P mrežama:
 - Kada pretražujemo u nestrukturiranim P2P mrežama, **indeks serveri** značajno unapređuju performanse
 - Odluka o tome gde treba čuvati podatke može se efikasnije doneti primenom **brokera**
- Uvode se posebni čvorovi – **super peerovi**, koji održavaju indeks podataka ili se ponašaju kao brokeri
- U ovakvoj organizaciji, svaki od **slabih peerova** (engl. weak peers) povezuje se kao klijent na super peer



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

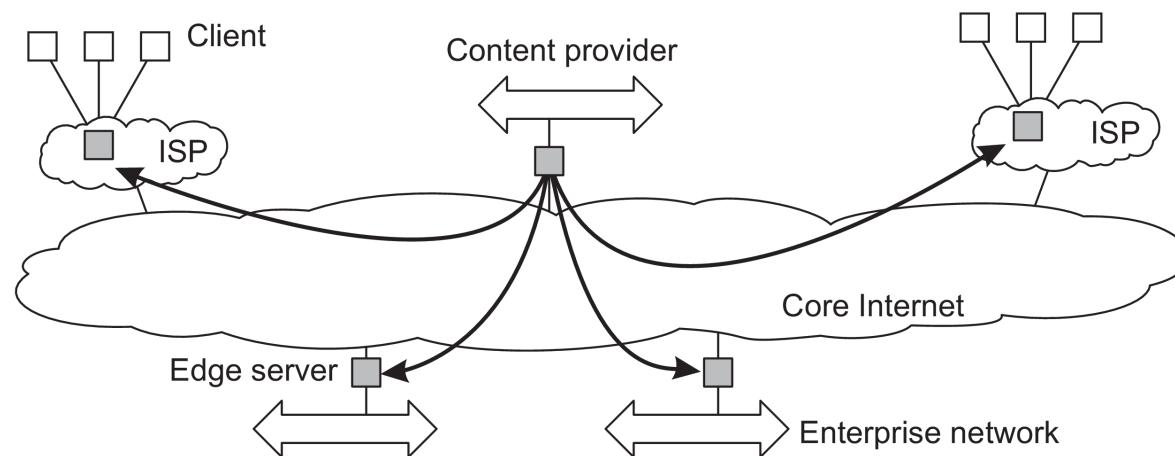
Primer: Skype VOIP mreža



- Postoji dodatni centralizovani Skype login server, super peer-ovi su Skype super čvorovi
- Kada A želi da kontaktira B preko Skype-a:
 - Ako su A i B na javnom Internetu
 - Uspostavlja se TCP konekcija za kontrolne pakete između A i B
 - Sam poziv se realizuje putem UDP paketa između dogovorenih portova
 - Ako se A nalazi iza firewall-a, a B je na javnom Internetu
 - A uspostavlja TCP konekciju (za kontrolne pakete) ka super peer-u S
 - S uspostavlja TCP konkeciju (za preusmeravanje kontrolnih paketa) ka B
 - Sam poziv se realizuje putem UDP paketa između dogovorenih portova
 - Ako se i A i B nalaze iza firewall-a
 - A se povezuje na onlajn super peer S putem TCP
 - S uspostavlja TCP konekciju ka B
 - Za sam poziv, drugi super peer se kontaktira kako bi radio kao relej R: A uspostavlja konekciju ka R, kao i B
 - Sav video saobraćaj se prosleđuje preko dve TCP konekcije putem R

Hibridne organizacije

- **Sistemi sa serverima na ivici** (engl. *edge-server systems*) su sistemi povezani na Internet pri čemu se serveri nalaze na ivici mreže, tj. na granici između poslovne (engl. *enterprise*) mreže i samog Interneta koju npr. pružaju Internet servis provajderi (ISP)
- Klijenti se povezuju na Internet putem edge servera čija je glavna uloga da pruža sadržaje, eventualno nakon primene funkcija filtriranja i transkodiranja
- Pogled na Internet kao kolekciju edge servera:



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

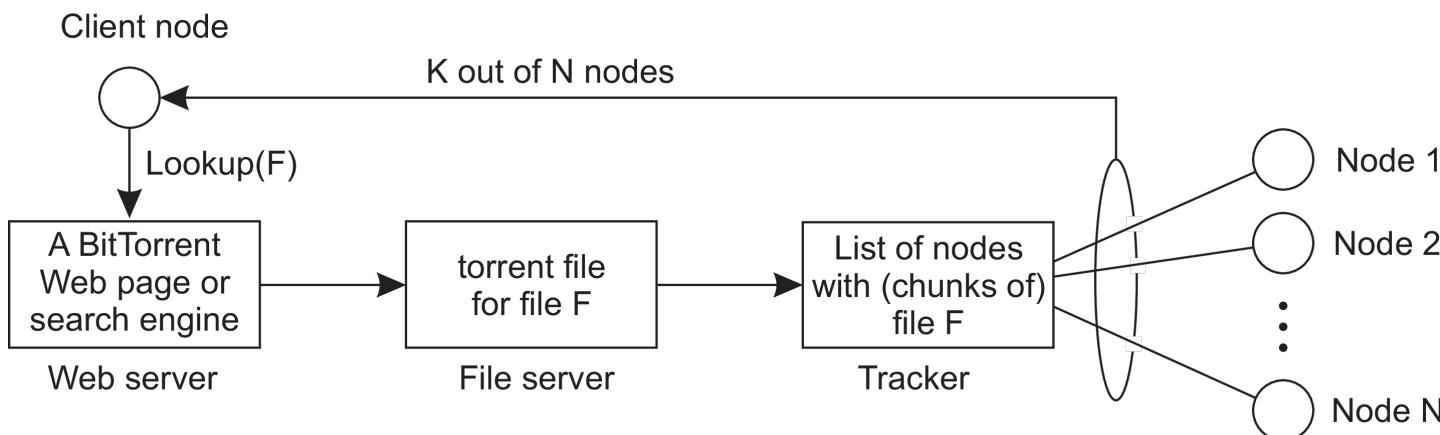
Hibridne organizacije

- Kod **kolaborativnih distribuiranih sistema** tipično se koristi klasični klijent-server pristup kako bi se uspostavila komunikacija, ali čim se čvor priključi sistemu uspostavlja se potpuno decentralizovana šema za saradnju
- Tipičan primer je **BitTorrent sistem za deljenje fajlova**
 - Glavna ideja je da kada korisnik traži neki fajl, on preuzima komade (engl. *chunks*) fajla od drugih korisnika dok god se od preuzetih komada ne može sastaviti kompletan fajl
 - Kako bi se sprečila slobodna vožnja (engl. *free riding*), kod BitTorrent-a se fajl može preuzimati samo kada klijent i pruža sadržaj nekom drugom
 - U implementaciji BitTorrent-a, čvor se može pridružiti i posebnom strukturiranom P2P sistemu u vidu DHT koji pomaže u praćenju preuzimanja fajla. Pri tome se opterećenje centralnog pratioca distribuira na uključene čvorove



Primer: BitTorrent

- Princip: traženje fajla F
 - fajl se traži u globalnom direktorijumu koji potom vraća odgovarajući **torrent fajl**
 - torrent fajl sadrži reference ka **pratiocu** (engl. tracker) koji predstavlja server koji održava tačan spisak aktivnih čvorova koji imaju (delove) F
 - P može da se pridruži roju (engl. swarm), dobije komad besplatno i potom zameni kopiju tog komada sa peerom Q , koji se takođe nalazi u roju, za neki drugi



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

Primer: BitTorrent

- Glavni detalji rada:
 - Pratilac za fajl F vraća skup svih njegovih procesa za preuzimanje – aktuelni roj
 - A komunicira samo sa podskupom roja: skupom suseda N_A
 - Ako je $B \in N_A$ tada je i $A \in N_B$ (simetričnost susedstva)
 - Skupovi suseda se redovno ažuriraju od strane pratioca
- Blokovi za razmenu:
 - Fajl se deli u komade iste veličine (tipično 256 KB)
 - Peerovi razmenjuju blokove komada, tipično 16 KB
 - A može da pošalje blok d komada D , samo ako ima komad D
 - Sused B pripada potencijalnom skupu P_A , ako B ima blok koji je potreban A
 - Ako je $B \in P_A$ i $A \in P_B$, onda su A i B u poziciji da zamene blok

- Glavni detalji rada:
 - Pratilac za fajl F vraća skup svih njegovih procesa za preuzimanje – aktuelni roj
 - A komunicira samo sa podskupom roja: skupom suseda N_A
 - Ako je $B \in N_A$ tada je i $A \in N_B$ (simetričnost susedstva)
 - Skupovi suseda se redovno ažuriraju od strane pratioca
- Blokovi za razmenu:
 - Fajl se deli u komade iste veličine (tipično 256 KB)
 - Peerovi razmenjuju blokove komada, tipično 16 KB
 - A može da pošalje blok d komada D , samo ako ima komad D
 - Sused B pripada potencijalnom skupu P_A , ako B ima blok koji je potreban A
 - Ako je $B \in P_A$ i $A \in P_B$, onda su A i B u poziciji da zamene blok



Primer: BitTorrent

- Tri faze rada:

1. Faza podizanja (engl. *bootstrap*)

- A je upravo dobio prvi komad (kroz mehanizam optimističnog odglavlјivanja (engl. *unchoking*) kojim čvor iz N_A nesebično pruža blokove komada novoprdošlom čvoru kako bi mu omogućio da se pokrene), čime može da počne sa trgovinom

2. Faza trgovine (engl. *trading*)

- U ovoj fazi je $|P_A| > 0$ što znači da uvek postoji peer sa kojim A može da razmenjuje blokove. U praksi je u ovoj fazi preuzimanje fajla jako efikasno

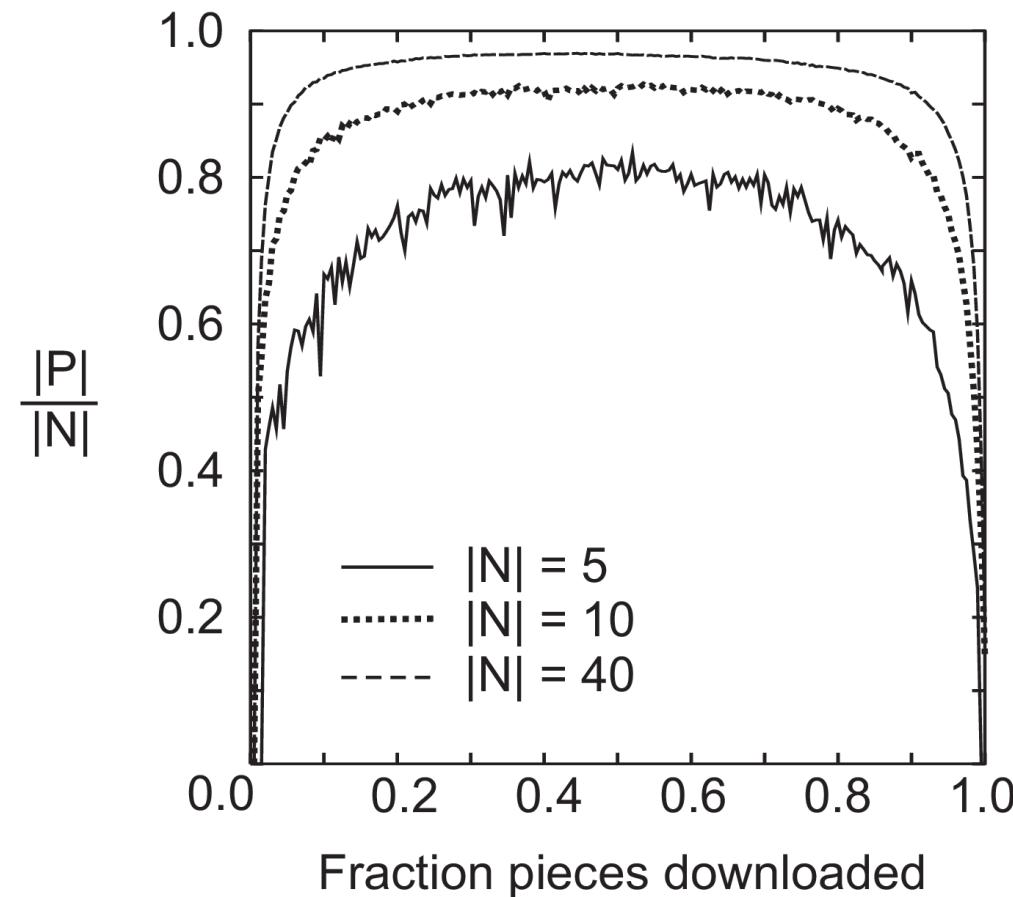
3. Faza poslednjeg preuzimanja (engl. *last download*)

- $|P_A| = 0$ zbog čega A zavisi od novoprdošlih peerova u N_A kako bi dobio poslednje nedostajuće komade. N_A se može promeniti samo od strane pratioca



Primer: BitTorrent

- Razvoj potencijalnog skupa veličina $|P|$ u odnosu na veličinu susedstva $|N|$:



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

Procesi

Osnovni pojmovi

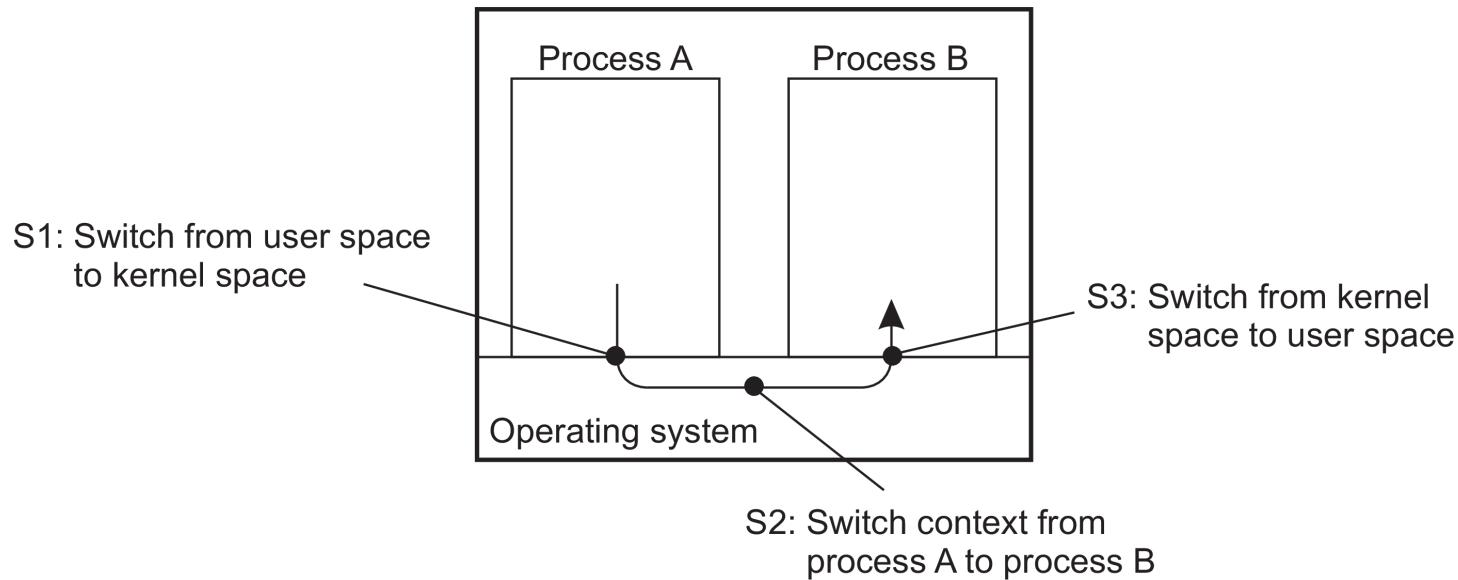
- Kako bi izvršio programe, operativni sistem kreira nad fizičkim procesorima posebne **virtuelne procesore za svaki od programa koji se izvršava (procesa)**:
 - **Procesor** pruža skup instrukcija uz sposobnost automatskog izvršavanja niza ovih instrukcija
 - **Nit** (engl. *thread*) je minimalni softverski procesor u čijem kontekstu se može izvršiti niz instrukcija. Čuvanje **konteksta niti** podrazumeva zaustavljanje trenutnog izvršavanja i čuvanje svih podataka koji su neophodni kako bi se izvršavanje kasnije nastavilo
 - **Proces** (engl. *process*) je **program koji se izvršava**, tj. **softverski procesor** u čijem kontekstu se može izvršavati jedna ili više niti. Izvršavanje niti podrazumeva izvršavanje niza instrukcija u kontekstu neke niti

Kontekst (engl. context)

- **Kontekst procesora** je minimalni skup vrednosti u registrima procesora koji se koristi prilikom izvršavanja instrukcija (npr. programski brojač, adresni registar, pokazivač steka)
- **Kontekst niti** je minimalni skup vrednosti u registrima i memoriji koji se koristi prilikom izvršavanja instrukcija (tj. kontekst procesora, osnovni podaci za upravljanje nitima)
- **Kontekst procesa** je minimalni skup vrednosti u registrima i memoriji koji se koristi prilikom izvršavanja niti (tj. kontekst niti, ali najmanje i vrednosti u registrima MMU – *memory management unit*, kao i TLB – *translation lookaside buffer*)

Zamena konteksta (engl. context switching)

- **Zamena procesa je skupa** jer je neophodno angažovanje operativnog sistema, tj. prelazak iz korisničkog (*user*) u privilegovani (*kernel*) režim
- **Primenom niti izbegava se zamena procesa** – složene aplikacije se organizuju kao skupovi niti, a ne procesa, izbegava se skupa interprocesna komunikacija (IPC) – UNIX/Linux mehanizmi cevi (engl. *pipes*), redovi poruka (engl. *message queues*), deljeni segmenti memorije



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

Zamena konteksta

- Kako niti dele isti adresni prostor, **zamena konteksta** između **niti** može se izvršiti potpuno **nezavisno od operativnog sistema**, ali je i više podložna greškama nego zamena procesa, jer nema OS podrške za zaštitu pristupa memoriji
- **Zamena konteksta, stvaranje i uništavanje niti** je dosta **jeftinije** nego što je to slučaj sa procesima
- **Izbegava se nepotrebno blokiranje** – kod višenitnih procesa, kada jedna od niti čeka na I/O, blokiranje se može izbeći
- **Koristi se paralelizam** – niti se mogu izvršavati paralelno na multiprocesoru ili višejezgarnom procesoru

Niti u distribuiranim sistemima

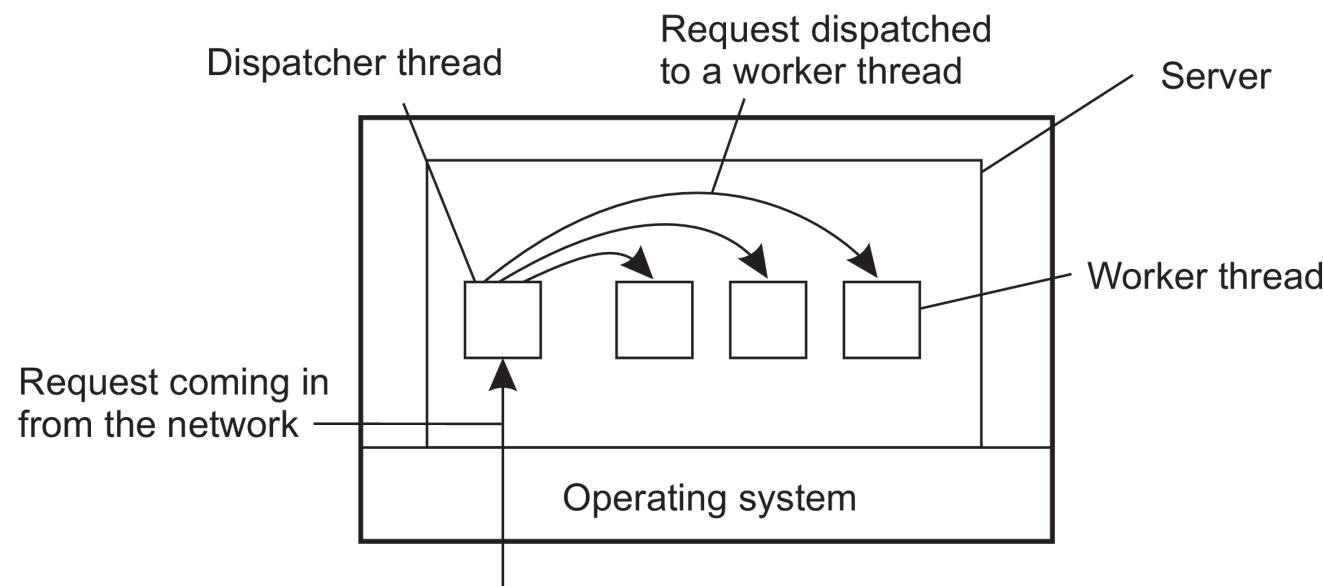
- Primena niti omogućava blokiranje sistemskih poziva bez blokiranja čitavog procesa u okviru koga se nit izvršava
- Na ovaj način **niti olakšavaju opis komunikacije u distribuiranom okruženju** kada se održava više logičkih konekcija u isto vreme
- **Latencija komunikacija** se obično **skriva** tako što proces inicira komunikaciju i onda nastavi sa drugim zadacima
- **Primer – višenitni veb klijent:** pretraživač prikazuje delove HTML stranice dok se ostatak preuzima, ako treba preuzeti više fajlova, svaki od fajlova preuzima posebna nit putem blokirajućeg HTTP zahteva, ako je veb server repliciran, onda višenitni pristup omogućava i bolje raspoređivanje opterećenja (engl. *load balancing*) i paralelno preuzimanje kada se, ako su zahtevi ka različitim serverima, dobija linearno ubrzanje
- Pretraživači tipično imaju TLP između 1.5 i 2.5 – niti se koriste pre svega za logičku organizaciju rada pretraživača

Niti u distribuiranim sistemima

- **Glavne prednosti rada sa nitima u distribuiranim sistemima nalaze se na serverskoj strani – unapređene performanse i bolja struktura**
- **Unapređenje performansi servera:**
 - kreiranje niti je **jeftinije** od kreiranja procesa
 - jednonitni server, za razliku od višenitnog, ne može dobro **skalirati** na multiprocesorskim sistemima
 - kao i na klijentskoj strani, **latencija** se može **pokriti** obradom narednog zahteva dok se ne pribave podaci za prethodni
- **Bolja struktura servera:**
 - većina servera ima visoke zahteve po pitanju I/O, primena dobro poznatih blokirajućih poziva uprošćava strukturu sistema
 - **višenitni programi** su **kraći i lakši za razumevanje** usled pojednostavljene kontrole toka

Niti u distribuiranim sistemima

- **Model fajl servera po principu dispečer/radnik**



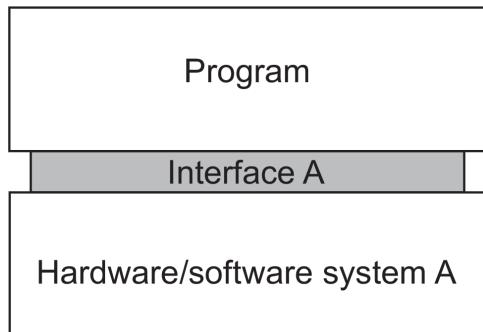
Model servera	Osobine
Višenitni proces	paralelno, blokirajući sistemski pozivi
Jednonitni proces	sekvencijalno, blokirajući sistemski pozivi
Jednonitni konačni automat	paralelno, neblokirajući sistemski pozivi

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

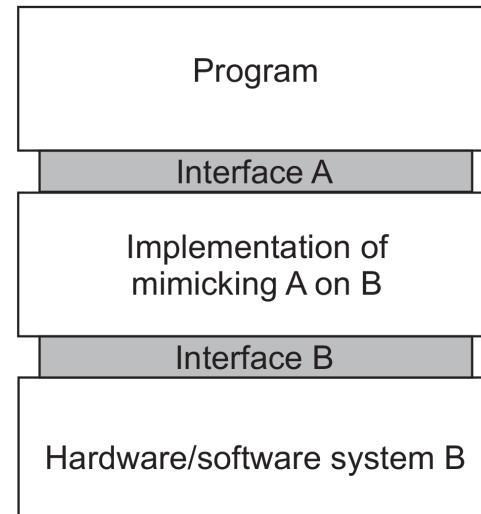
Virtuelizacija



- **Virtuelizacija** je važna za **portabilnost, pouzdanost i bezbednost** u distribuiranim sistemima, bazira se na principu **imitacije interfejsa**
 - virtuelne mašine (VM) rešavaju problem **brže promene softvera od hardvera**
 - VM olakšavaju **portabilnost** i migraciju koda i celog okruženja
 - VM **izoluje** otkaze ili napadnute komponente



(a) Opšta organizacija programa, interfejsa i sistema



(b) Opšta organizacija virtuelizacije sistema A na sistemu B

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

Tipovi virtuelizacije

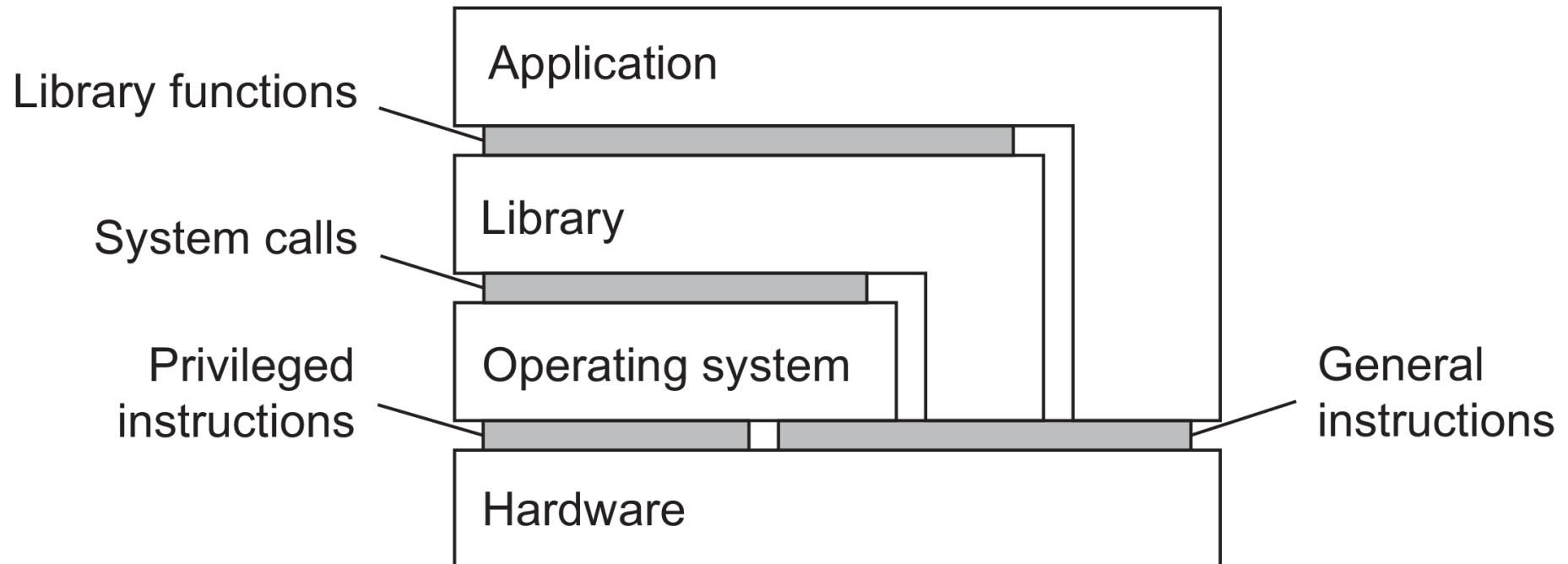


- Računarski sistemi nude **četiri različita tipa interfejsa na tri različita nivoa:**
 - **arhitektura skupa instrukcija** (engl. *instruction set architecture* – ISA) je skup mašinskih instrukcija, predstavlja **interfejs između hardvera i softvera**, može se podeliti na dva podskupa:
 - **privilegovane instrukcije** koje može izvršavati samo operativni sistem
 - **neprivilegovane instrukcije** koje može izvršavati bilo koji program
 - **sistemski pozivi** (engl. *system calls*) su interfejs koji nudi operativni sistem
 - **pozivi biblioteka** (engl. *library calls*) čine interfejse poznate i kao **programski interfejsi aplikacija** (engl. *application programming interface* – API), često su sistemski pozivi sakriveni pozivom API-ja

Tipovi interfejsa



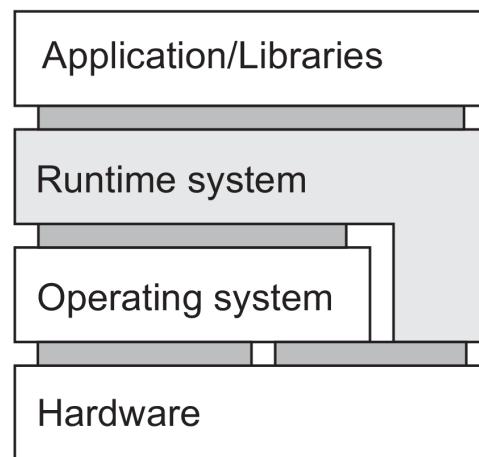
- **Različiti interfejsi u računarskim sistemima:**



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

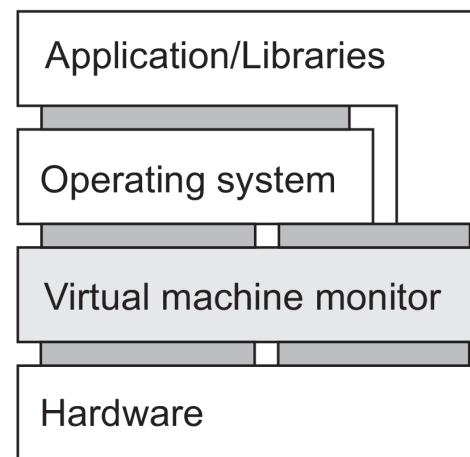
Načini virtuelizacije

(a) Procesna
virtuelna mašina



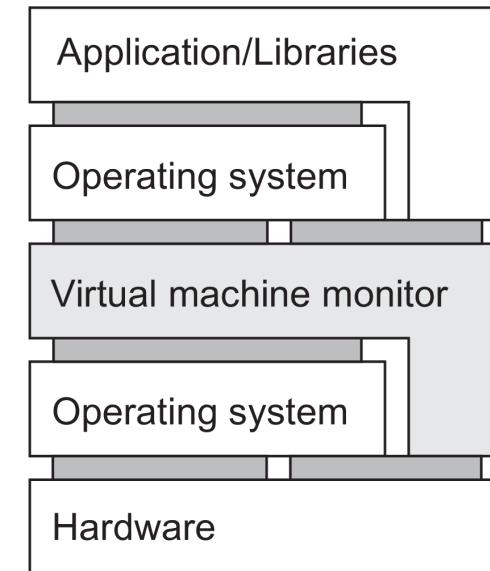
Za jedan proces, poseban set instrukcija, interpreter/ emulator, radi iznad OS, npr. JVM, emulacija programa

(b) Nativni monitor
VM – nativni
hipervizor



VMM instrukcije niskog nivoa,
kao i minimalni OS, npr. Xen,
VMware ESX

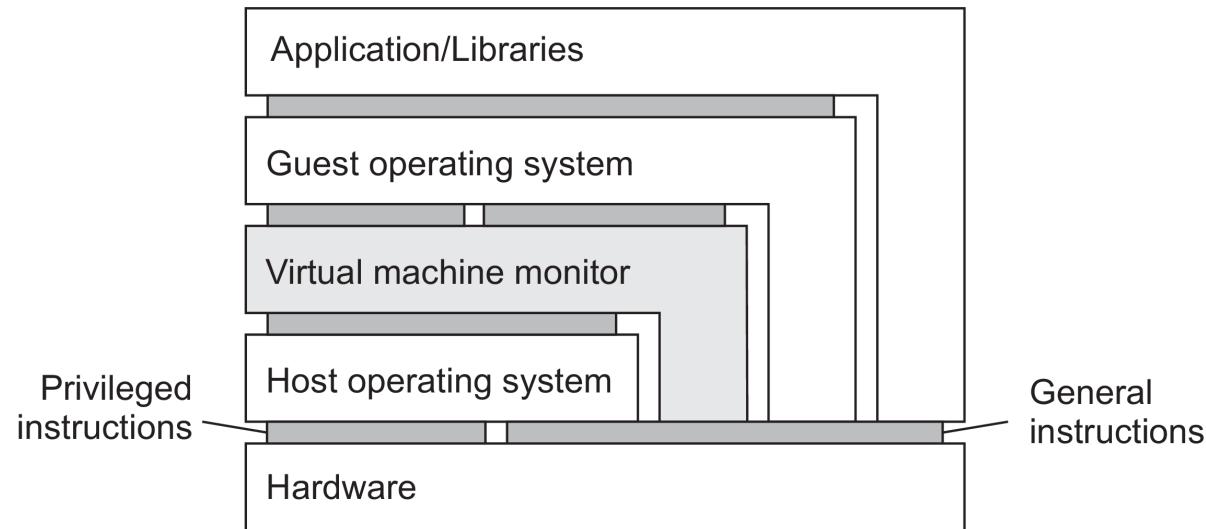
(c) Hostovani monitor
VM – hostovani
hipervizor



Instrukcije niskog nivoa, ali
najveći deo posla se delegira
kompletnom host OS, npr.
VirtualBox

Performanse virtuelnih mašina

- Performanse VM vrlo bliske direktnom izvršavanju programa na OS – veliki deo koda sa **neprivilegovanim instrukcijama** izvršava se **direktno u hardveru**
- Ako i samo ako se privilegovane instrukcije pozovu iz korisničkog režima, izazvaće **zamku** (engl. trap) kod OS, sve ostale instrukcije su neprivilegovane



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

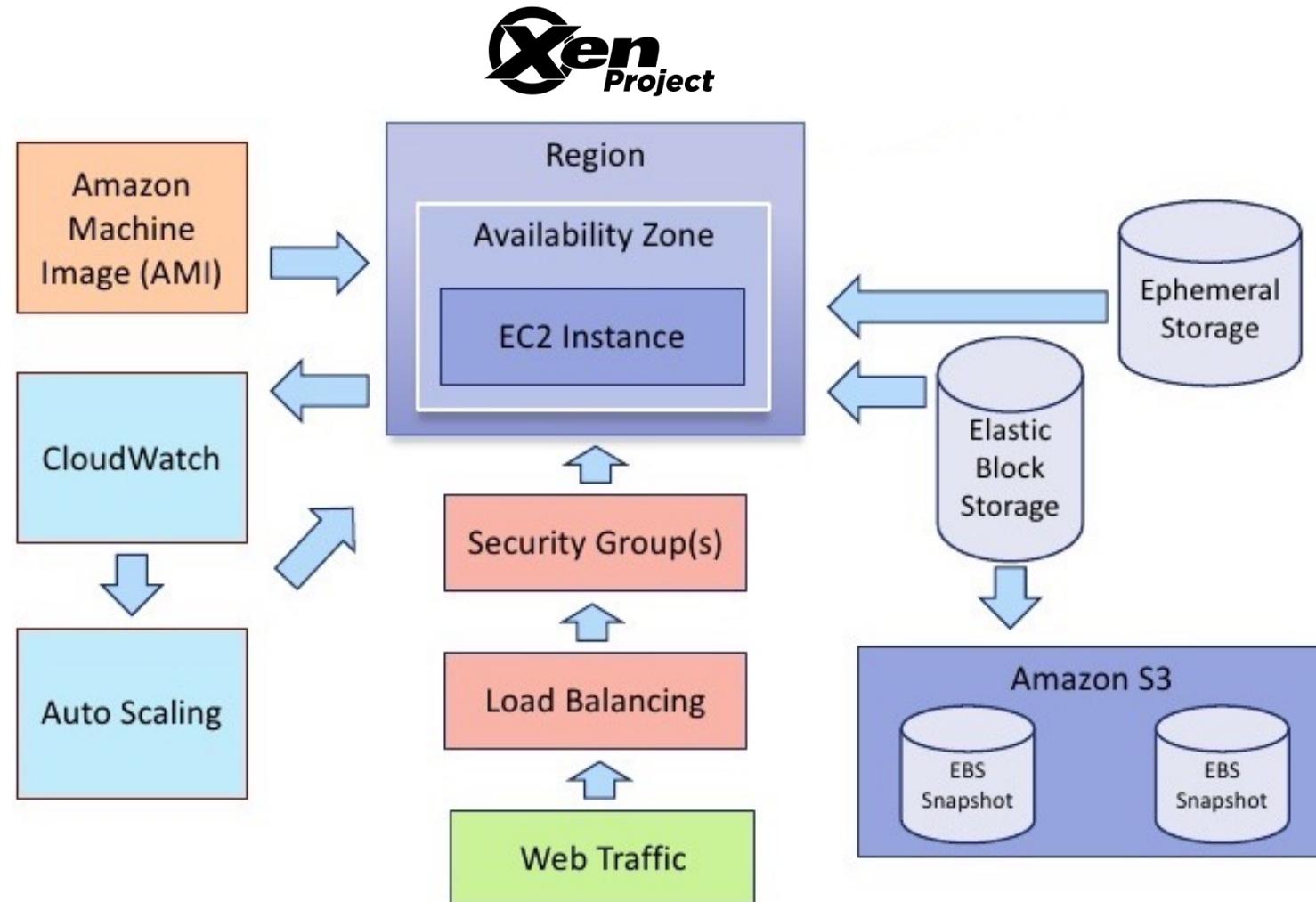
Virtuelne mašine i računarstvo u oblaku

- Tri različita tipa servisa kod računarstva u oblaku:
 - **infrastruktura-kao-servis** (engl. *Infrastructure-as-a-Service* – **IaaS**) pokriva osnovnu infrastukturu, **virtuelizacija** ovde igra **ključnu ulogu**, cloud provajder umesto fizičke mašine iznajmljuje VM (ili MVM) koja može deliti fizičku mašinu sa drugim klijentima, izolacija između klijenata ne postoji samo u smislu performansi, kada su performanse ključne ostaje opcija tzv. bare-metal mašina ili virtuelni privatni oblak
 - **platforma-kao-servis** (engl. *Platform-as-a-Service* – **PaaS**) pokriva servise na nivou sistema
 - **softver-kao-servis** (engl. *Software-as-a-Service* – **SaaS**) sadrži same aplikacije

Primer: EC2

- **Amazon Elastic Compute Cloud (EC2)** je veb servis koji omogućava kreiranje okruženja sa više umreženih virtuelnih servera, koji zajedno čine osnovu **kompletног distribuiranog sistema**
 - postoji veliki broj unapred konfigurisanih slika mašina poznatih kao **Amazon Machine Images** – AMIs. AMI je softverski paket koji se može instalirati a sadrži kernel OS-a i skup servisa, npr. **LAMP** – Linux kernel, Apache veb server, MySQL baza, PHP biblioteke, koristi se XEN hipervizor
 - pokretanje AMI stvara **EC2 instancu** – samu virtuelnu mašinu na kojoj se izvršavaju klijentske aplikacije, svaka instanca ima javnu (pristup sa Interneta) i privatnu (komunikacija između instanci) IP adresu, EC2 pruža monitoring, automatsko skaliranje i balansiranje opterećenja (engl. *load balancing*)
 - EC2 okruženje nudi različite nivoe servisa za CPU, memoriju, disk, platformu (32 i 64 bita) i mrežu (propusni opseg)
 - kada se instanca zaustavi, svi lokalni podaci se gube, neophodno eksplisitno čuvanje podataka u perzistentnom sistemu – **Amazon S3** ili **Amazon Elastic Block Storage (EBS)** virtuelni blok uređaj koji se može mauntovati kao disk na bilo koju od instanci

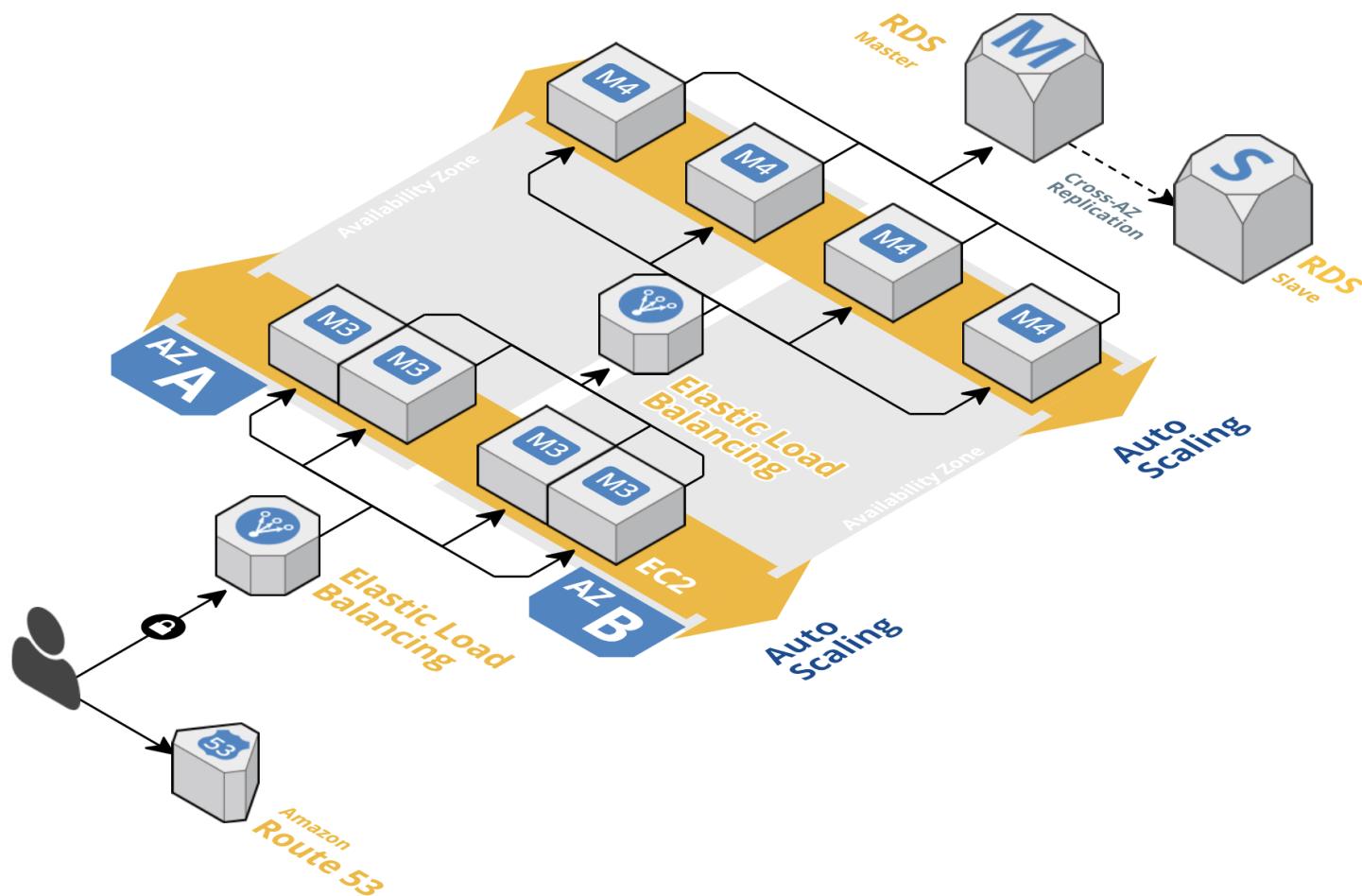
Primer: arhitektura



Izvor: <https://linuxmasterswiki.com/2017/07/11/amazon-ec2-amazon-elastic-compute-cloud>

Primer: arhitektura EC2

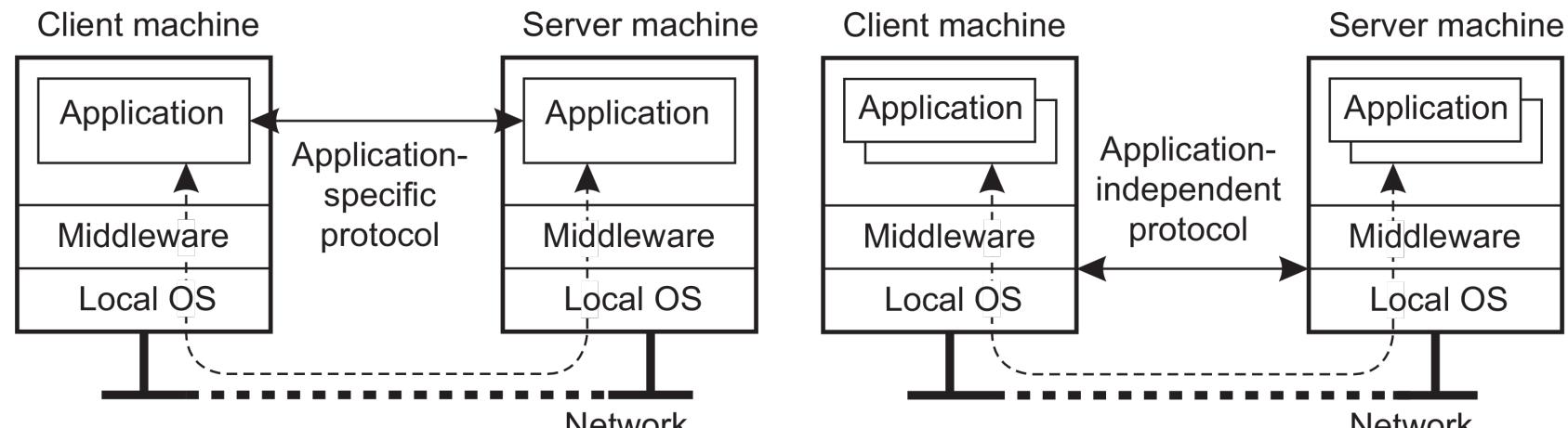
- Balansiranje opterećenja i automatsko skaliranje:



Izvor: <https://linuxmasterswiki.com/2017/07/11/amazon-ec2-amazon-elastic-compute-cloud>

Klijent-server interakcija

- Dva načina interakcije između klijenta i servera:

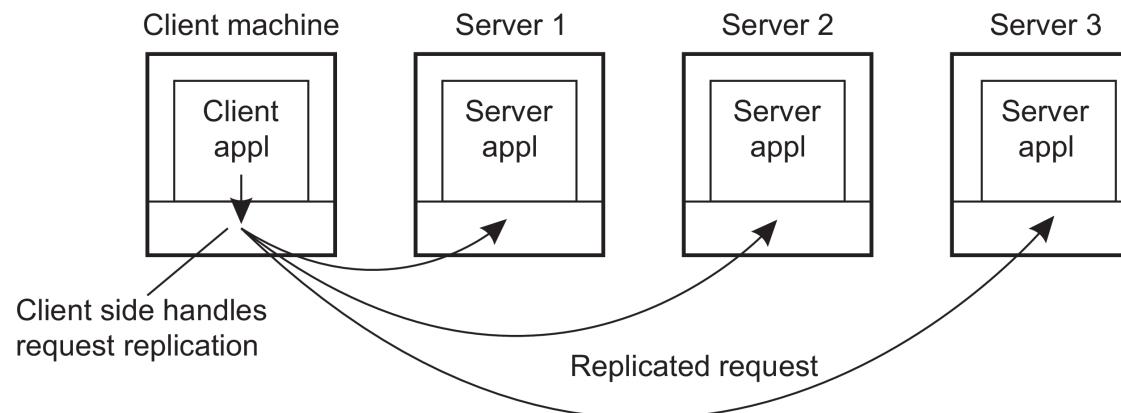


(a) Mrežna aplikacija sa sopstvenim protokolom
Primer: kalendar

(b) Opšte rešenje za pristup udaljenim aplikacijama
Tanki klijent (engl. *thin-client approach*) kao terminal

Softver na klijentskoj strani

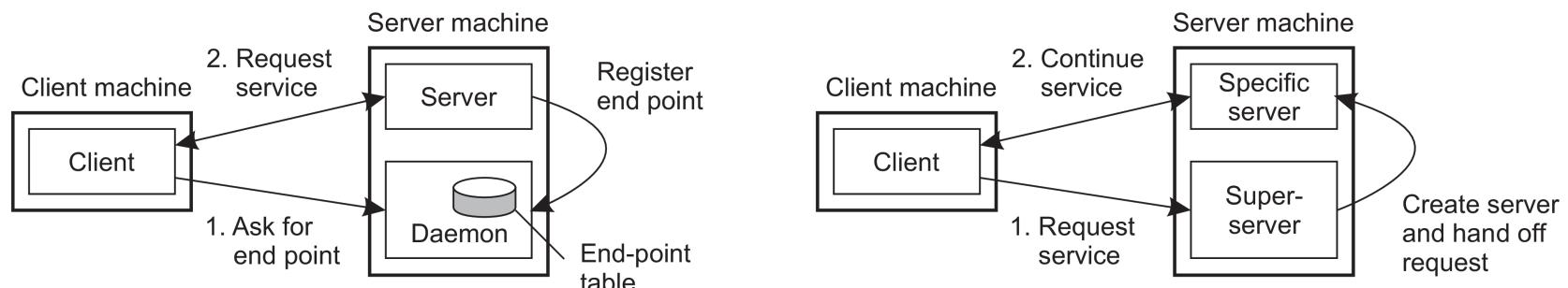
- Glavni cilj **transparentnost distribuiranosti**
 - **transparentnost pristupa:** klijentski isečak koda (engl. *stubs*) se generiše na osnovu definicije interfejsa servera (za RPC), pretvara lokalne pozive u poruke i obratno, skriva razlike u arhitekturi mašina i samu komunikaciju
 - **transparentnost lokacije/migracije:** klijentski softver (obično *midlver*) vodi evidenciju o stvarnoj lokaciji
 - **transparentnost replikacije:** višestruke pozive rešava klijentski isečak



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

Server

- **Server** je proces koji implementira određeni servis u ime skupa klijenata
 - server čeka na dolazeći zahtev od klijenta i potom osigurava da se pripremi i pošalje odgovor na zahtev, nakon čega čeka na sledeći zahtev
- **Dva tipa servera:**
 - **iterativni:** server obrađuje zahteve jedan po jedan
 - **konkurentni:** koristi **dispečer** koji prihvata dolazeći zahtev koji se onda prosleđuje posebnom procesu ili niti – danas standard jer mogu obraditi više zahteva i kada ima blokirajućih operacija (obraćanje disku ili drugim serverima)
- Klijenti šalju ka **krajnjoj tački** (engl. **end point**), tj. **portu** na mašini gde se izvršava server (npr. FTP port 21, HTTP 80), **dinamička dodela porta**



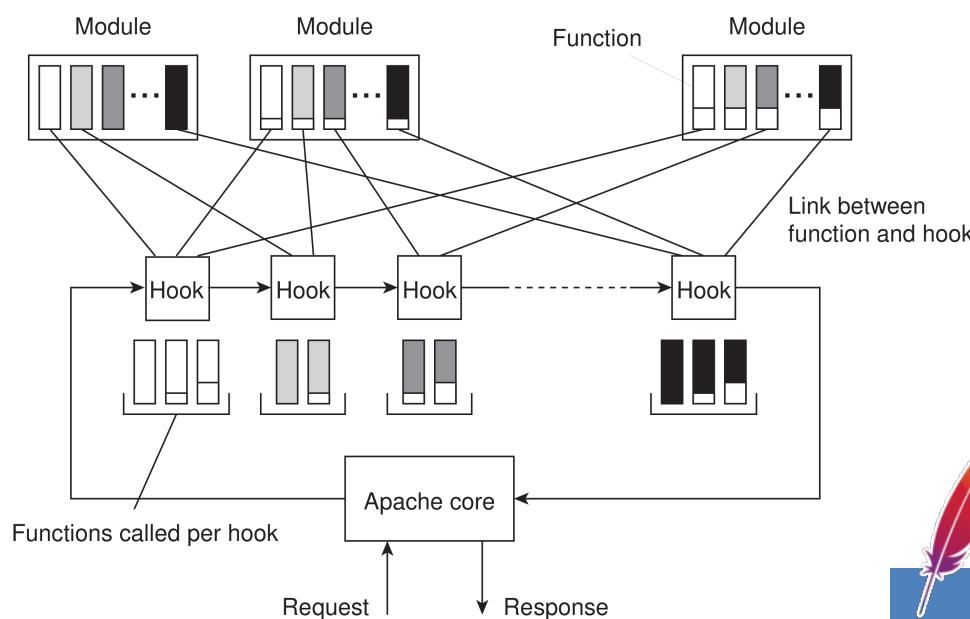
Izvor: <https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017>

Serveri i stanje

- **Serveri bez i sa pamćenjem stanja – stateless i stateful**
- **Bez pamćenja stanja (engl. stateless):**
 - nikada ne čuvaju tačne informacije o status klijenta nakon obrade zahteva (ne pamte da li je fajl bio otvoren, ne garantuju invalidaciju klijentskog keša, ne vodi evidenciju o svojim klijentima), npr. veb server
 - **klijenti i serveri su potpuno nezavisni, nekonzistentnosti stanja** zbog pada klijenta ili servera su redukovane, moguć **gubitak performansi** zato što server ne može predvideti ponašanje klijenta
- **Sa pamćenjem stanja (engl. stateful):**
 - **vodi evidenciju o svojim klijentima**, npr. fajl server (pamti da li je fajl otvoren kako bi se vršilo preuzimanje unapred (engl. *prefetching*), zna koje podatke su klijenti keširali i omogućava klijentima da čuvaju lokalne kopije deljenih podataka), nudi **visoke performanse**

Primer: Apache web server

- Visoko **konfigurabilan** i **proširiv** server, **nezavisan od platforme**
- Apache Portable Runtime (APR) je biblioteka sa platformski-nezavisnim interfejsom za upravljanje fajlovima, mrežom, nitima, zaključavanjem, itd.
- Koncept **kuke** (engl. hook) – čuvar mesta za određenu grupu funkcija, zahtevi se obrađuju u fazama od kojih svaka sadrži određeni broj kuka



Kuka – grupa sličnih akcija koje su deo obrade zahteva
Primeri kuka:

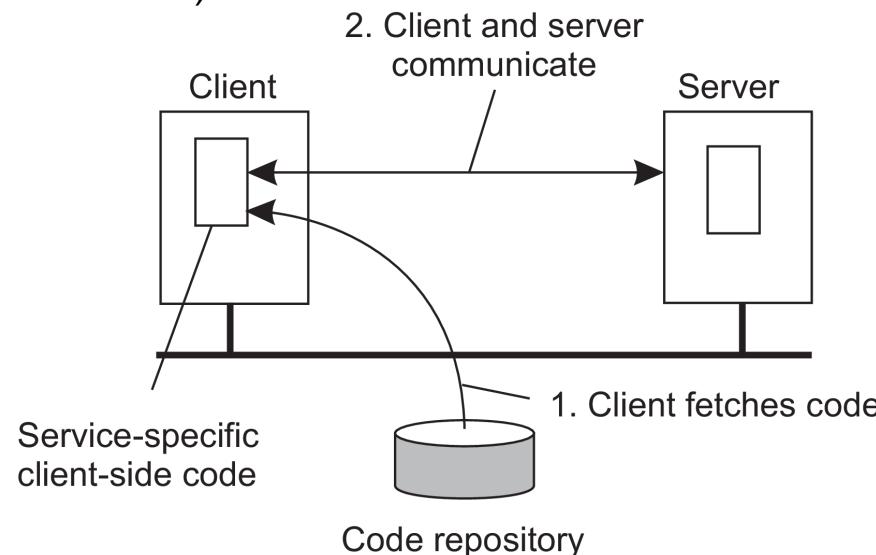
- prevodenje URL u naziv lokalnog fajla
- logovanje informacija
- provera prava pristupa
- provera identiteta klijenta



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

Migracija koda

- Motivacija **poboljšanje performansi, dinamička konfiguracija ds:**
 - **Balansiranje opterećenja**, osigurava da su serveri u data centru dovoljno opterećeni (energetska efikasnost), **minimizacija komunikacije** osigurava da se izračunavanje izvodi blizu podacima
 - **Fleksibilnost**, kod se po zahtevu dinamički prebacuje kod klijenta, **osnova dinamičkog veba** (princip dinamičke konfiguracije klijenta za komunikaciju sa serverom)



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