Procesi

Osnovni pojmovi

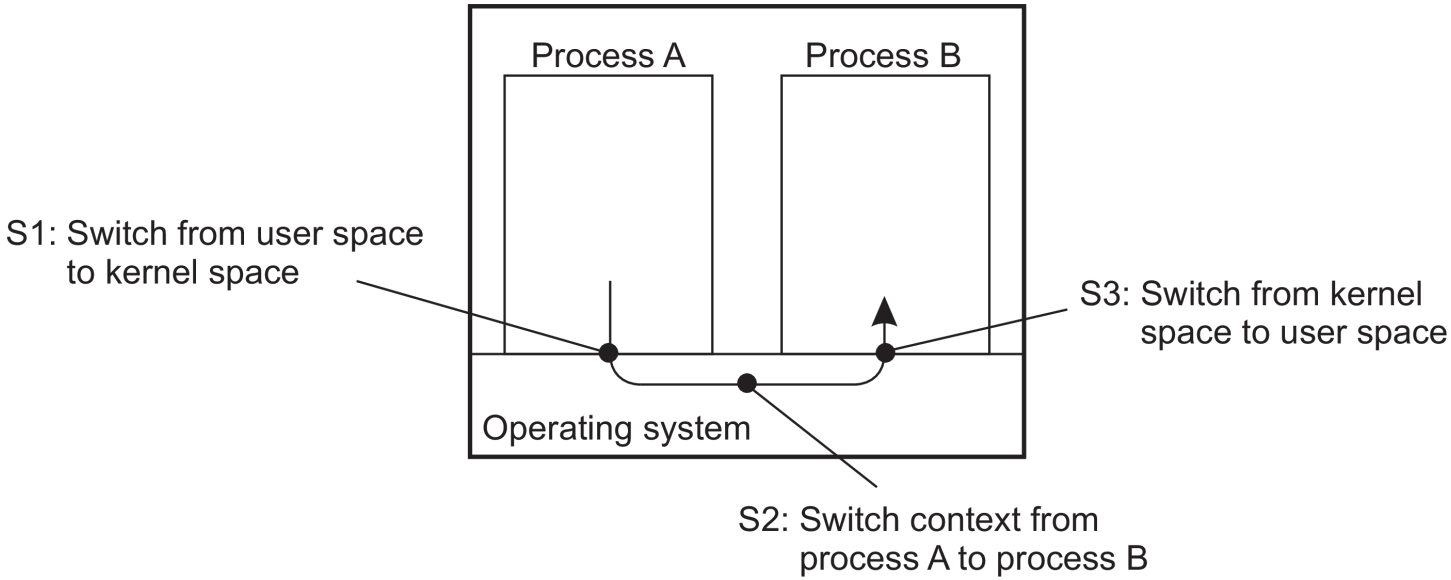
- Kako bi izvršio programe, operativni sistem kreira nad fizičkim procesorima posebne **virtuelne procesore za svaki od programa koji se izvršava** (**procesa**):
  - **Procesor** pruža skup instrukcija uz sposobnost automatskog izvršavanja niza ovih instrukcija
  - **Nit** (engl. *thread*) je minimalni 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

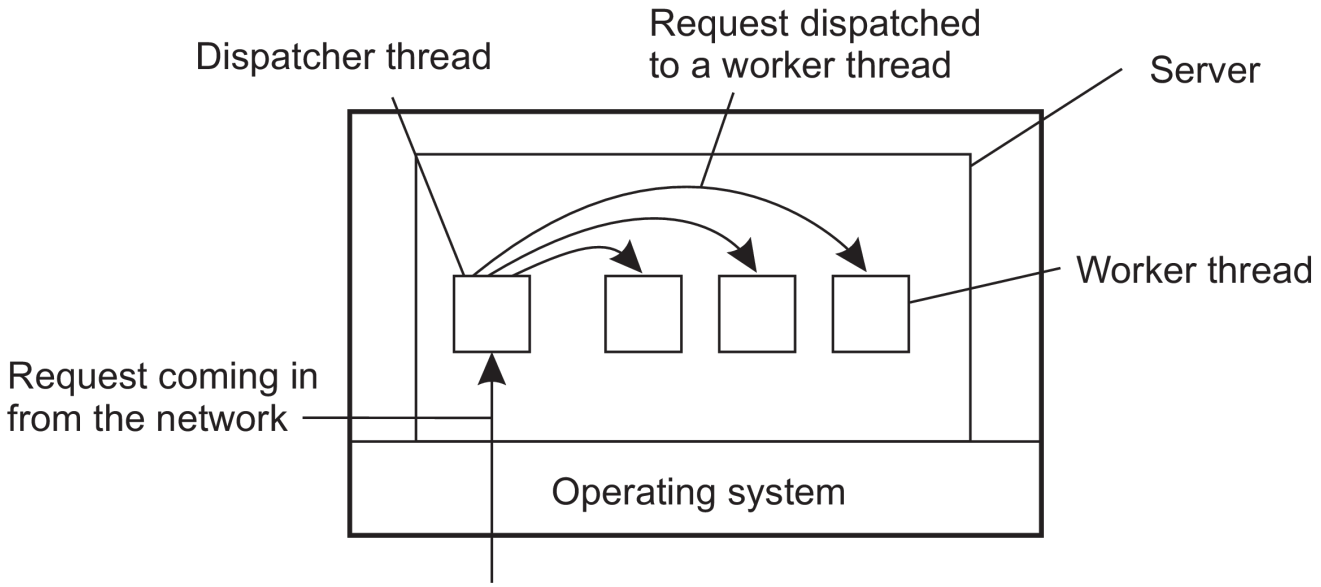
- Primena niti omogućava blokiranje sistemskih poziva bez blokiranja čitavog procesa u okviru koga se nit izvršava
- Na ovaj način **niti olakšavaju opis komunikacije u distribuiranom okruženju** kada se održava više logičkih konekcija u isto vreme
- **Latencija komunikacija** se obično **skriva** tako 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**

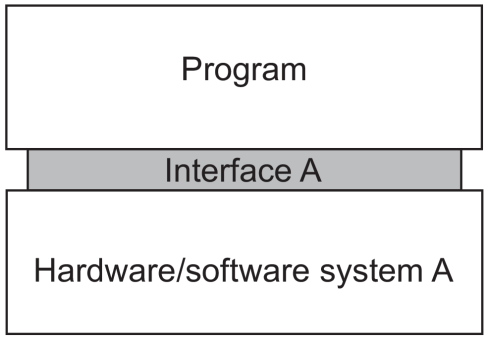


Model servera	Osobine
Višenitni proces	paralelno, blokirajući sistemski pozivi
Jednonitni proces	sekvencijalno, blokirajući sistemski pozivi
Jednonitni konačni automat	paralelno, neblokirajući sistemski pozivi

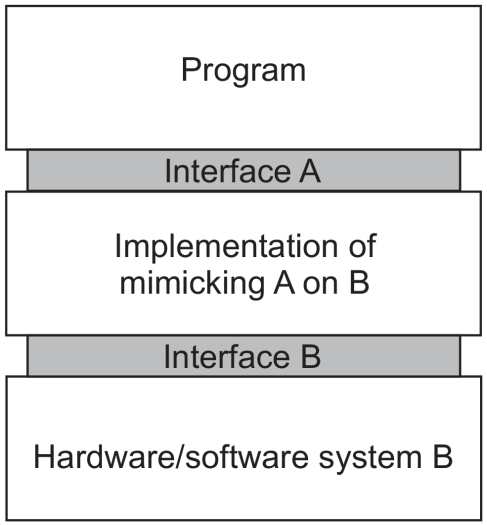
Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

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

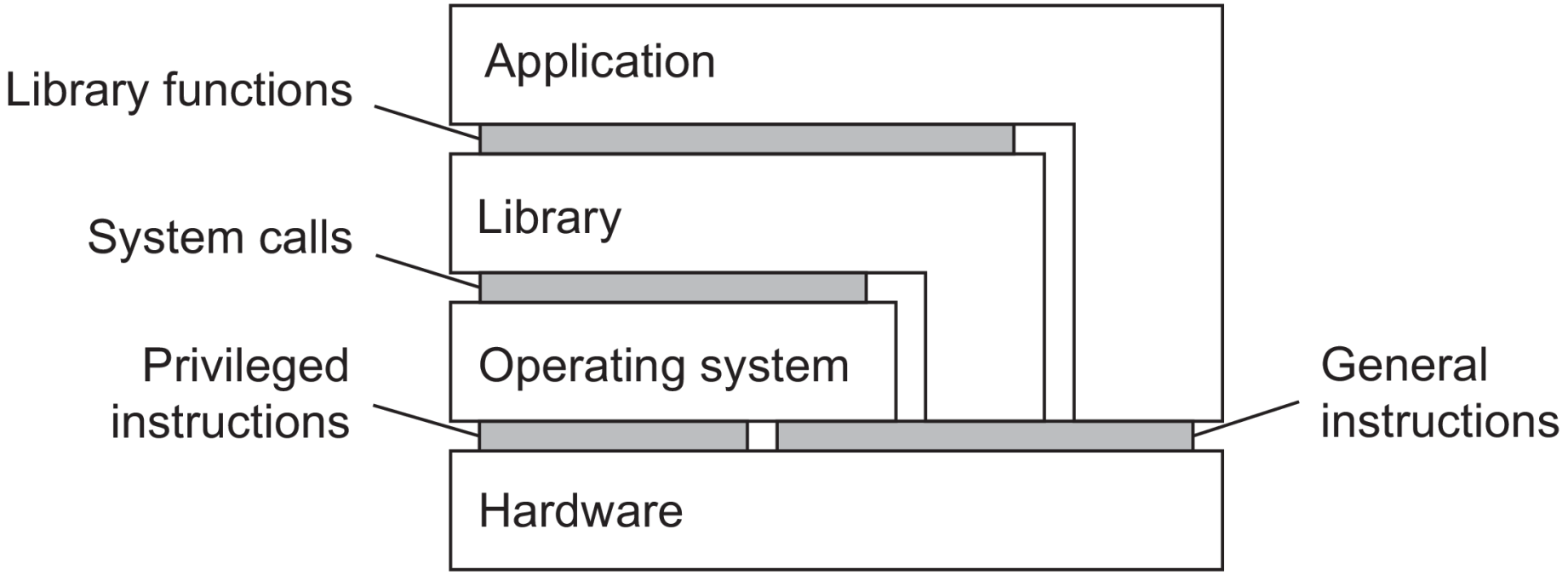
Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

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

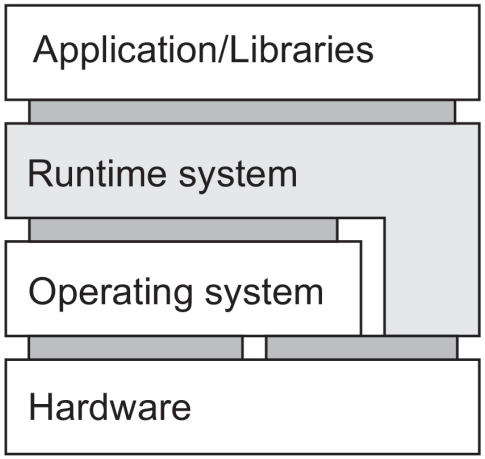
- **Različiti interfejsi u računarskim sistemima:**



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

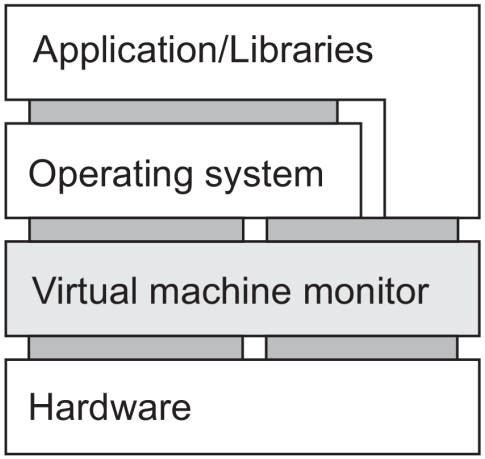
Načini virtuelizacije

(a) Procesna virtuelna mašina



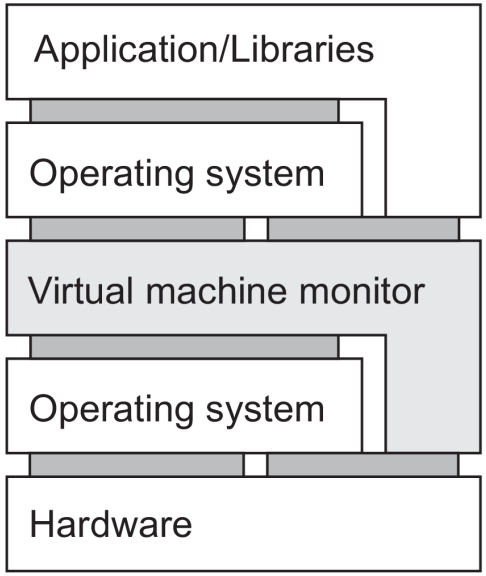
Za jedan proces, poseban set instrukcija, interpreter/ emulator, radi iznad OS, npr. JVM, emulacija programa

(b) Nativni monitor VM – nativni hipervizor



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

(c) Hostovani monitor VM – hostovani hipervizor

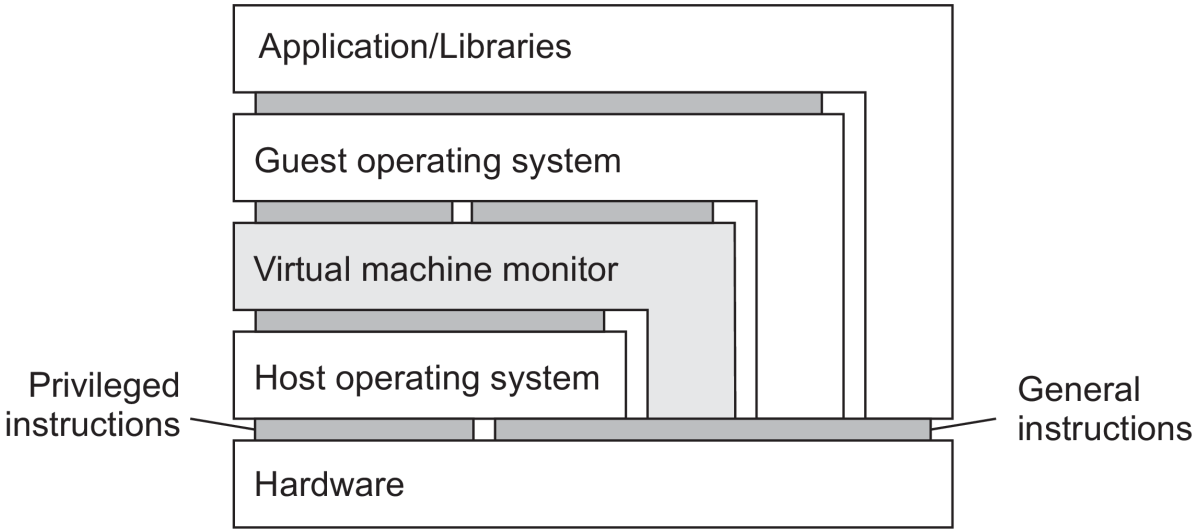


Instrukcije niskog nivoa, ali najveći deo posla se delegira kompletnom host OS, npr. VirtualBox

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

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

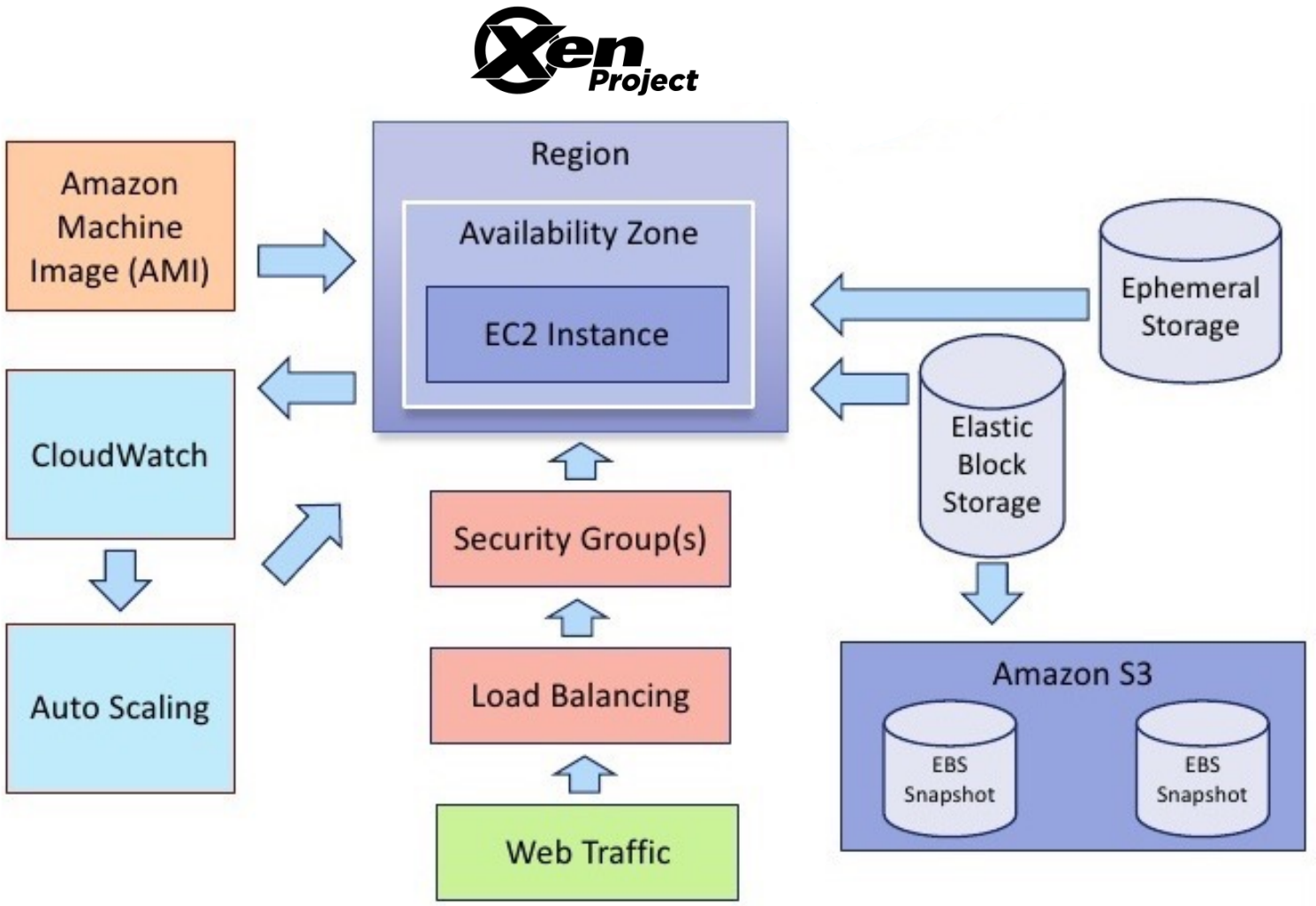
Virtuelne mašine i računarstvo u oblaku

- Tri različita tipa servisa kod računarstva u oblaku:
  - **infrastruktura-kao-servis** (engl. *Infrastucture-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: amazon web services EC2

- **Amazon Elastic Compute Cloud** (**EC2**) je veb servis koji omogućava kreiranje okruženja sa više umreženih virtuelnih servera, koji zajedno čine osnovu **kompletnog 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 eksplicitno 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

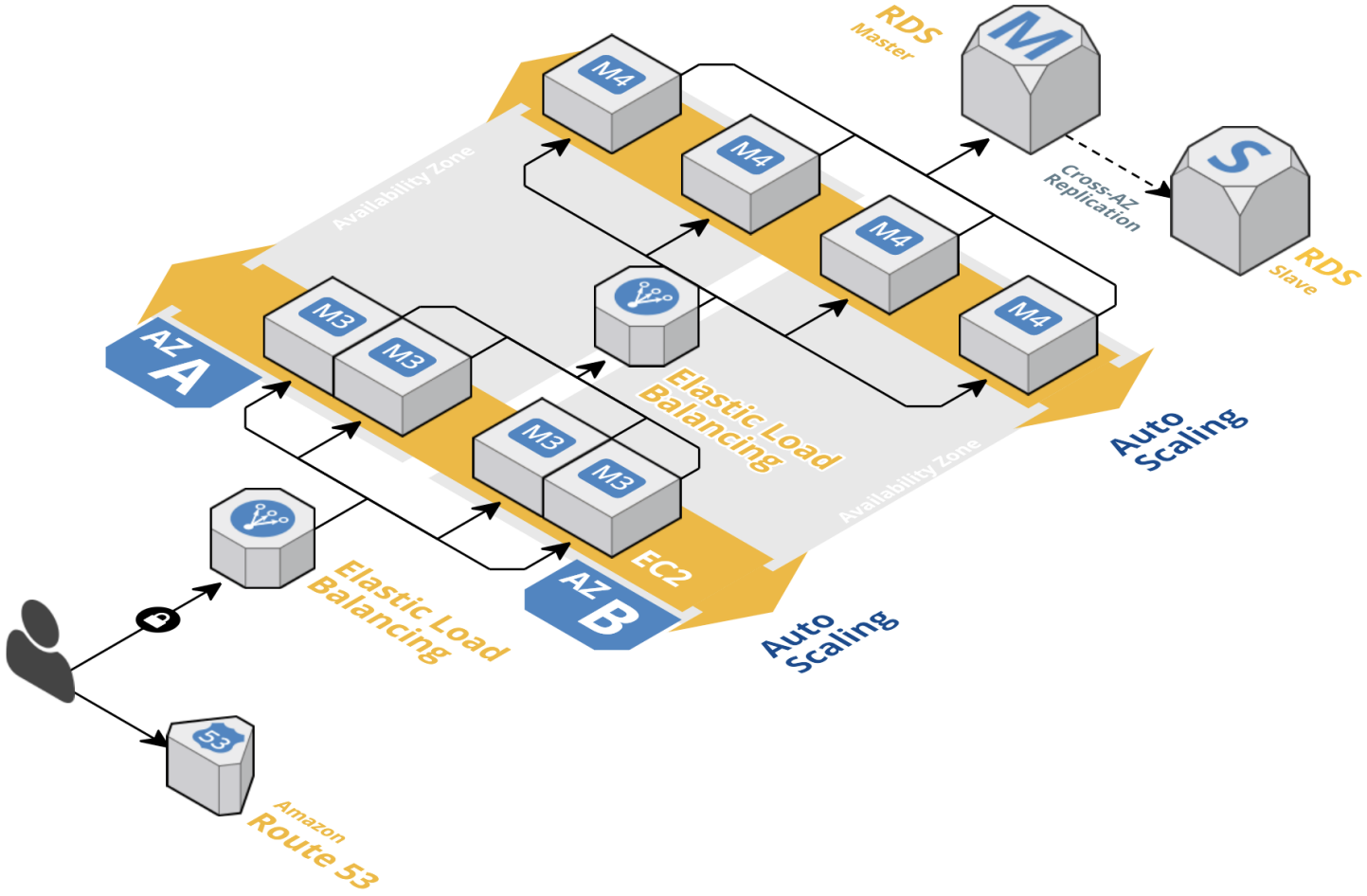
Primer: arhitektura amazon web services EC2



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

Primer: arhitektura amazon web services™ EC2

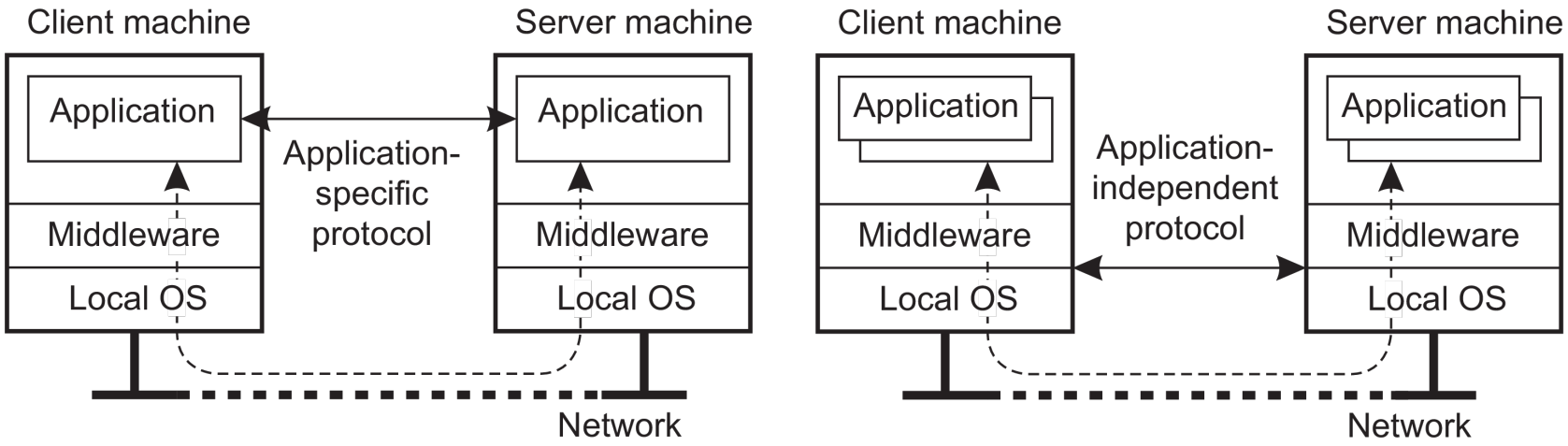
- Balansiranje opterećenja i automatsko skaliranje:



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

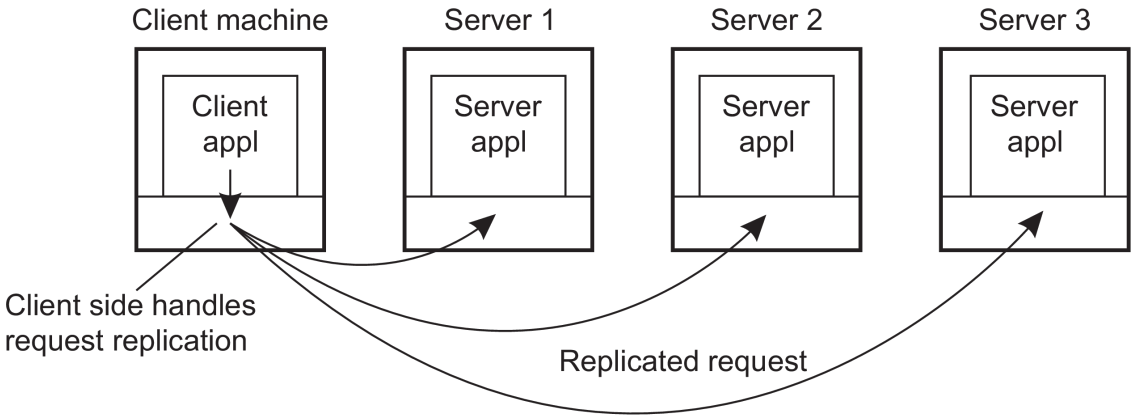
(b) Opšte rešenje za pristup udaljenim aplikacijama

Tanki klijent (engl. *thin-client approach*) kao terminal

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

Softver na klijentskoj strani

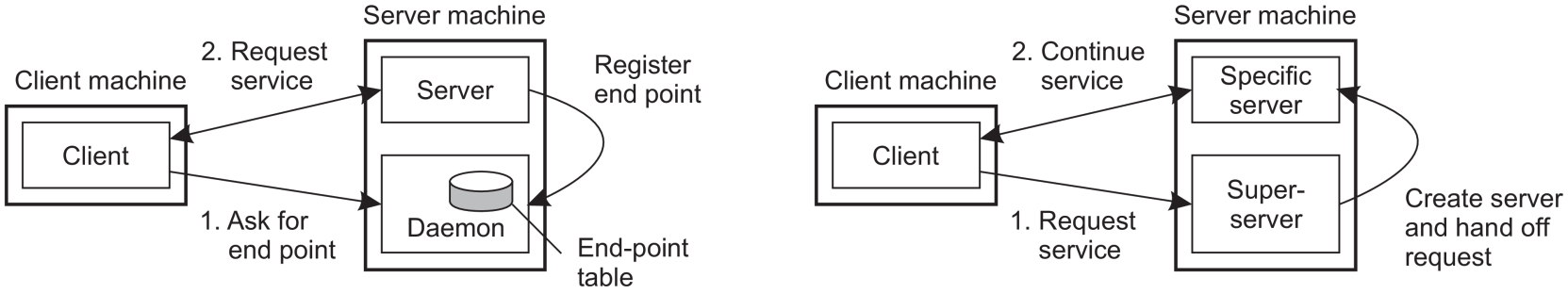
- Glavni cilj **transparentnost distribuiranosti**
  - **transparentnost pristupa**: klijentski isečak koda (engl. *stubs*) se generiše na osnovu definicije interfejsa servera (za RPC), pretvara lokalne pozive u poruke i obratno, skriva razlike u arhitekturi mašina i samu komunikaciju
  - **transparentnost lokacije/migracije**: klijentski sotfver (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**



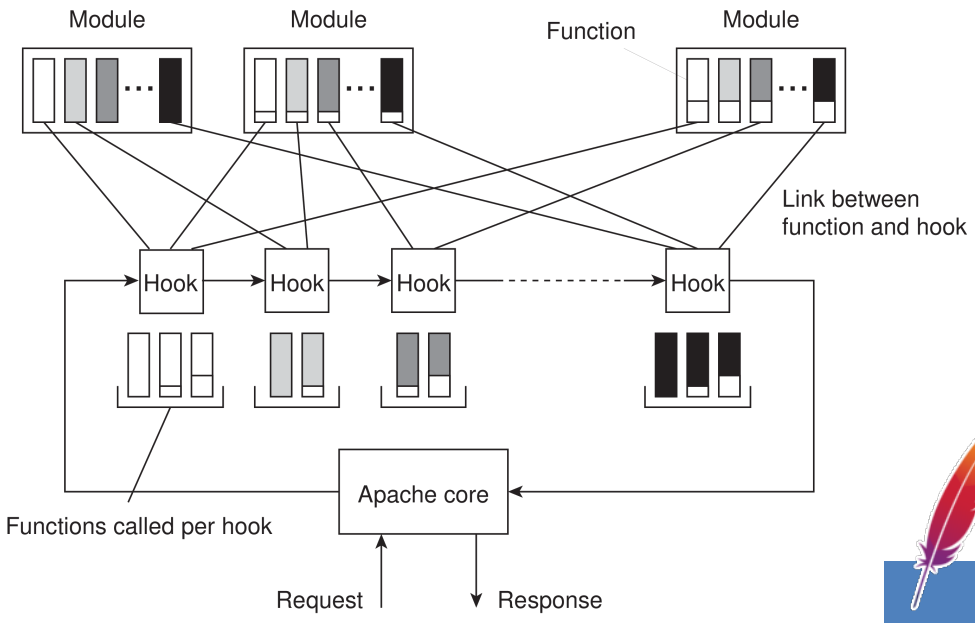
Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

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



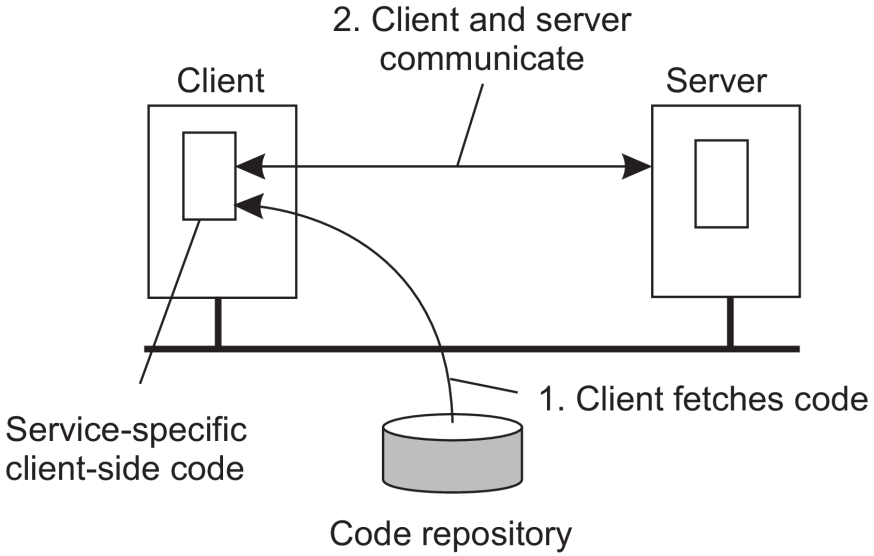
Kuka – grupa sličnih akcija koje su deo obrade zahteva
Primeri kuka:

- prevođenje 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)



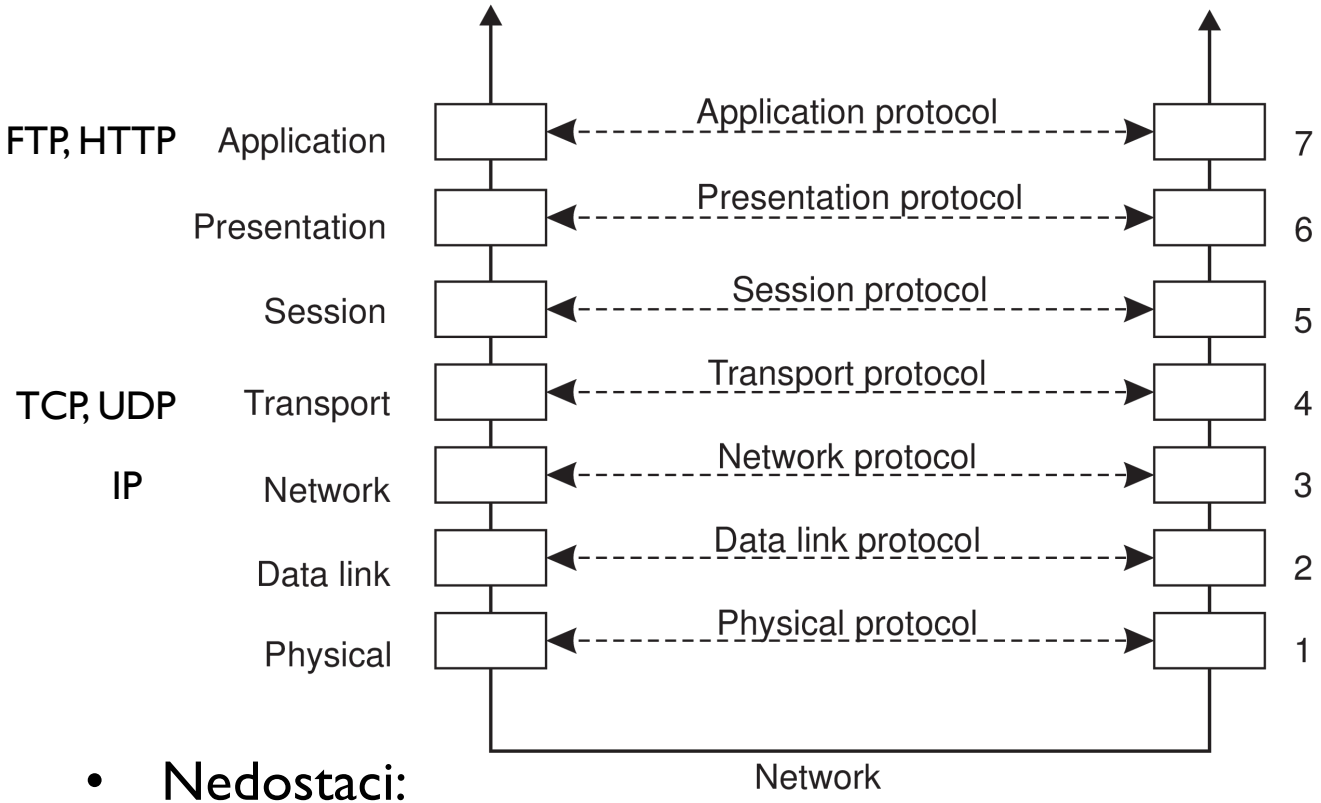
Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Komunikacija

Komunikacija

- **Međuprocesna komunikacija** (engl. **interprocess communication – IPC**) predstavlja **srce svih distribuiranih sistema**
- Savremeni distribuirani sistemi mogu biti sastavljeni od hiljada ili miliona procesa, raštrkanih širom mreže sa nepouzdanom komunikacijom kao što je Internet
- Skup pravila koja moraju poštovati procesi kako bi komunicirali predstavlja **komunikacioni protokol**, tipično **slojevito organizovan** (ISO OSI referentni model)
- Dva modela komunikacije:
  - **poziv udaljenih procedura** (engl. *Remote Procedure Call* – RPC)
  - **komunikacija orijentisana ka porukama** (soketi, MPI, MOM)

OSI referentni model



Transportni sloj pruža komunikaciona sredstva koja koristi većina distribuiranih Sistema

TCP: sa konekcijom, pouzdan, komunikacija sa tokovima

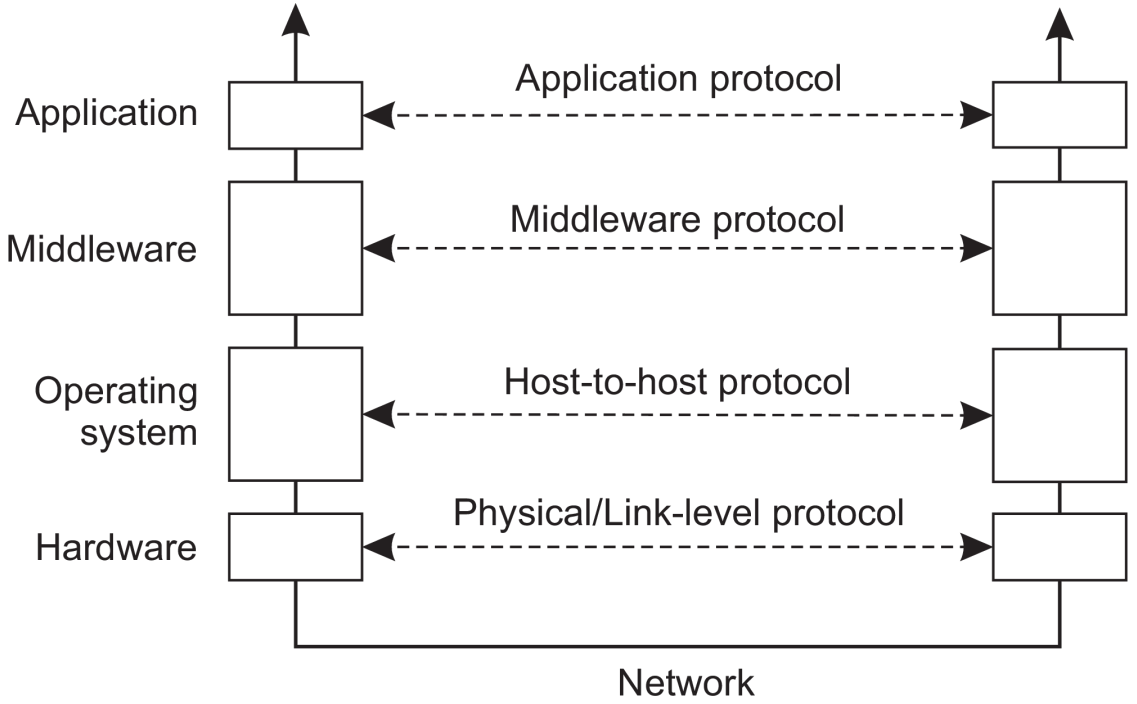
UDP: bez konekcije, nepouzdana (best-effort) datagram komunikacija

- Nedostaci:
  - fokus isključivo na slanju poruka
  - pruža često nepotrebne ili neželjene funkcionalnosti
  - krši transparentnost pristupa

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

Adaptirana slojevita šema

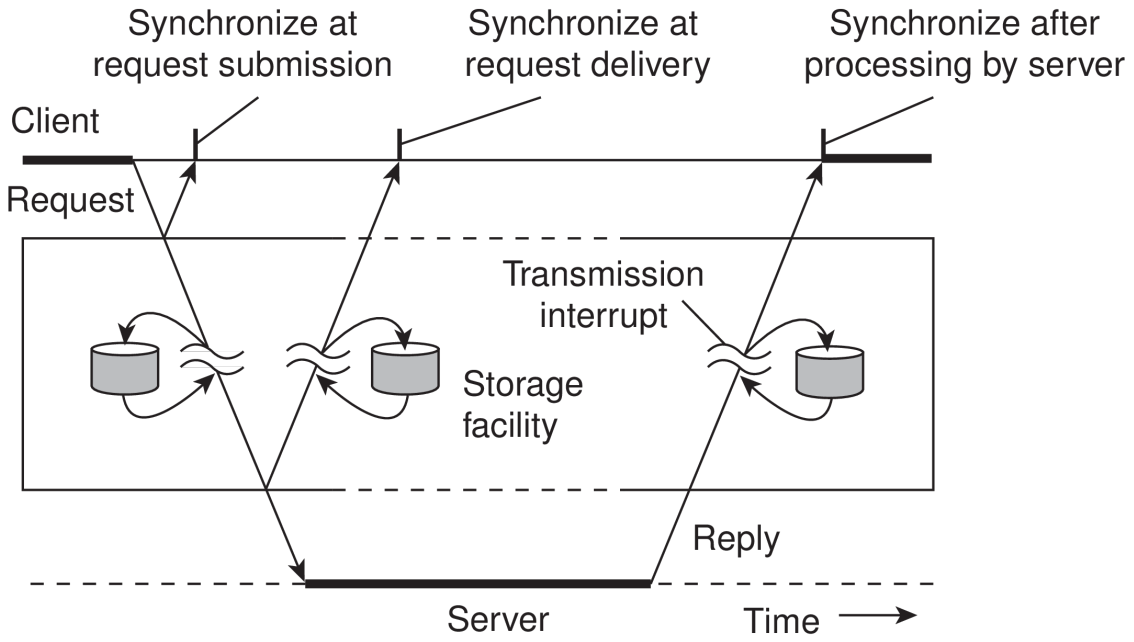
- **Midlver** je uveden u cilju pružanja **zajedničkih servisa** i **protokola različitim aplikacijama**
  - sadrži bogat skup **komunikacionih protokola**, (an)maršaling podataka, **protokoli za imenovanje** kako bi se lakše delili resursi, **bezbedonosni protokoli**, mehanizmi za **skaliranje** kao što su **replikacija** i **keširanje**



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

Tipovi komunikacije

- Vrste komunikacije:
  - **prolazna** (engl. *transient* – server odbacuje poruke ako ih ne može proslediti) i **perzistentna** (engl. *persistent* – server čuva poruku dok god je ne isporuči), **sinhrona** i **asinhrona**
- **Midlver** kao **posredni** (**distribuirani**) **servis** u **komunikaciji** na nivou **aplikacija**:



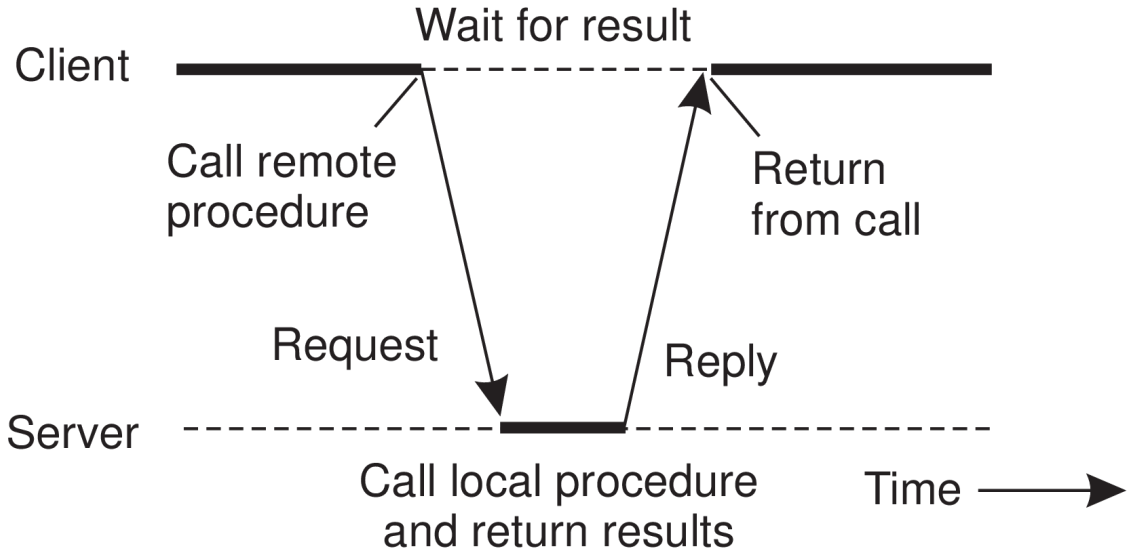
Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Tipovi komunikacije

- **Klijent/server** model se bazira na **prolaznoj sinhronoj komunikaciji**:
  - klijent i server moraju biti aktivni u vreme komunikacije
  - klijent šalje zahteve i blokira se dok ne dobije odgovor
  - server čeka na dolazeće zahteve i obrađuje ih
  - **mane sinhrone komunikacije**: klijent je blokiran dok čeka na odgovor, otkazi se moraju odmah rešavati jer klijent čeka, sam model nekad ne odgovara (mejl, vesti)
- **Midlver orijentisan ka porukama** (MOM) se bazira na **perzistentnoj asinhronoj komunikaciji**:
  - procesi šalju poruke koje se smeštaju u red, pošiljalac ne mora da čeka na odgovor, midlver osigurava otpornost na greške

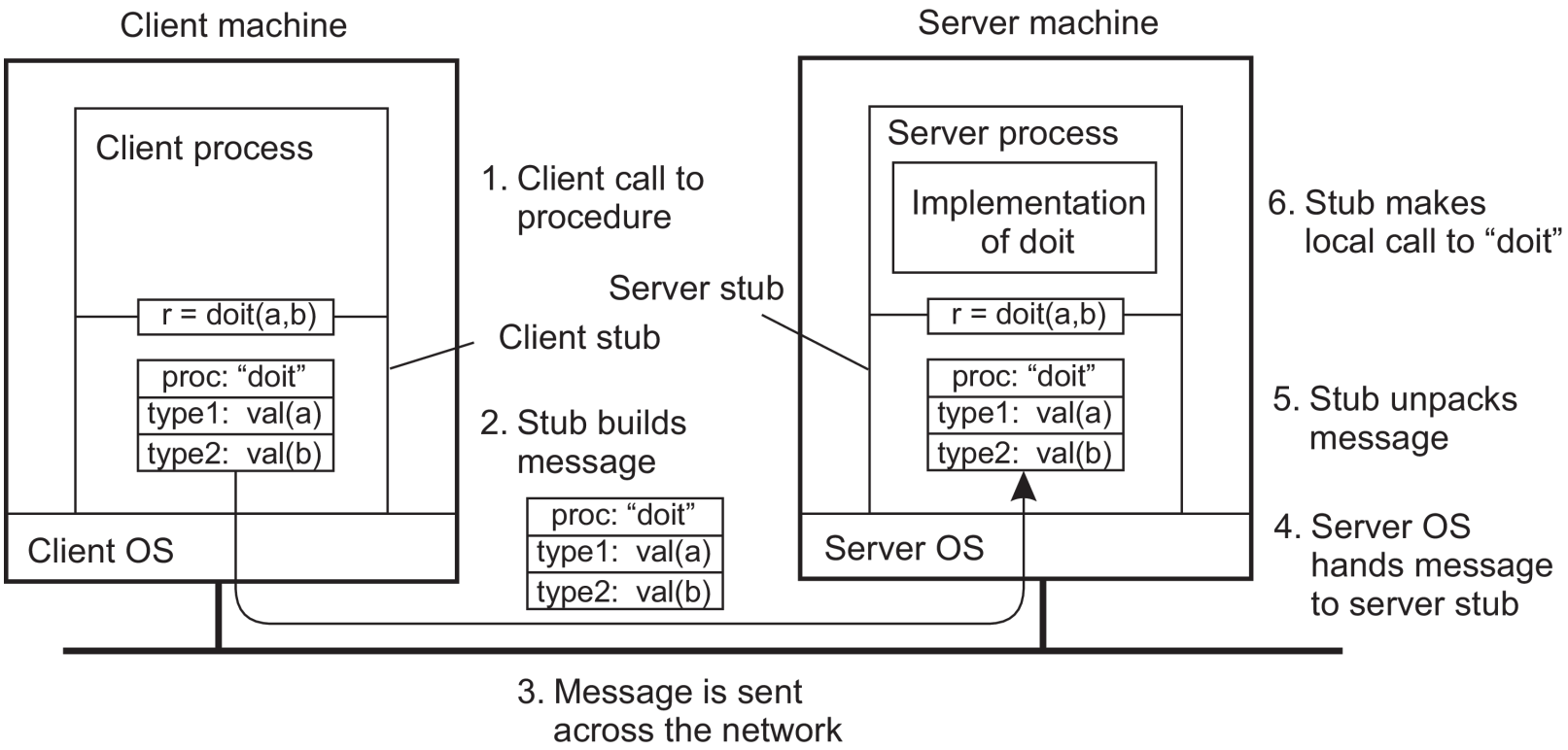
RPC

- Programeri su dobro upoznati sa jednostavnim modelom poziva procedura
- Dobro napravljene procedure rade izolovano (princip crne kutije) – nema razloga da se ne mogu izvršavati na posebnim mašinama
- Komunikacija između pozivaoca i pozvanog može se sakriti mehanizmom poziva procedura



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

Osnovne RPC operacije



1. klijentska procedura poziva klijentski isečak
2. isečak pravi poruku i poziva lokalni OS
3. lokalni OS šalje poruku udaljenom OS-u
4. udaljeni OS prosleđuje poruku isečku
5. isečak raspakuje parametre i poziva server
6. server pravi lokalni poziv i vraća rezultat isečku
7. isečak pravi poruku i poziva lokalni OS
8. OS servera šalje poruku OS-u klijenta
9. OS klijenta prosleđuje poruku isečku
10. isečak raspakuje rezultat i vraća ga klijentu

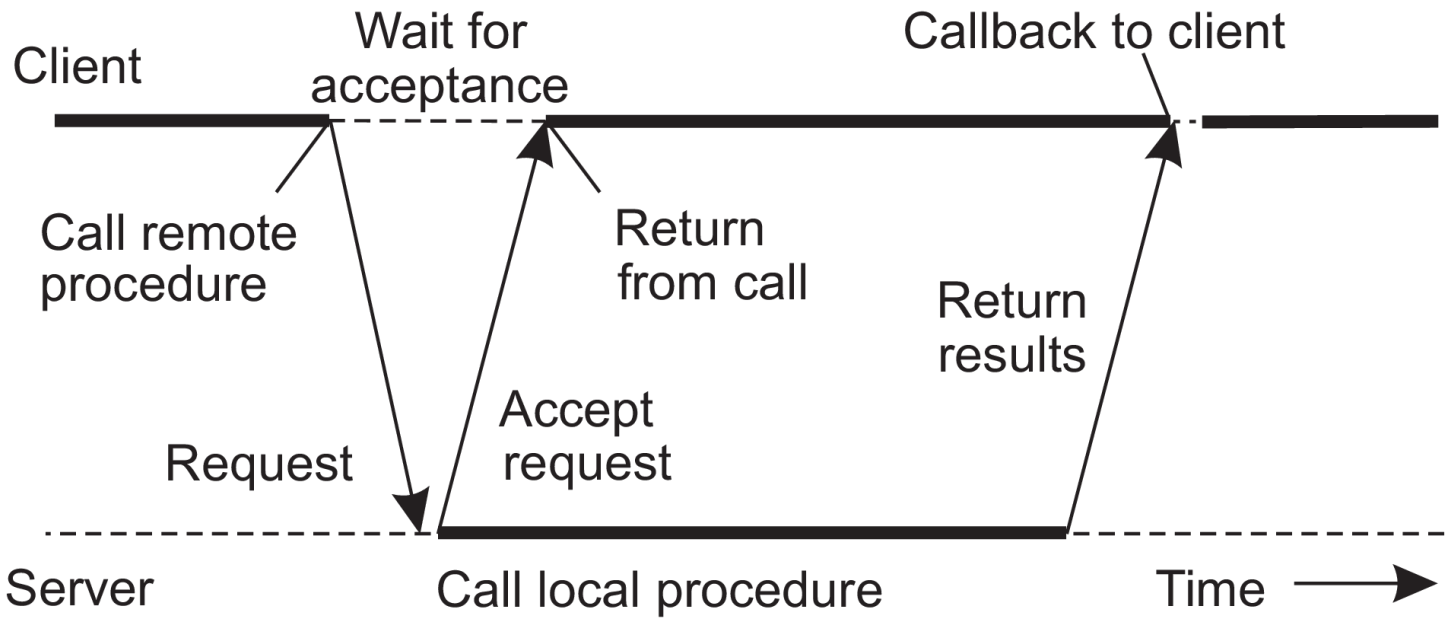
Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

RPC: prosleđivanje parametara

- Nije dovoljno samo spakovati parametre u poruke:
  - klijentska i serverska mašina mogu imati **različite reprezentacije podataka** (npr. različito uređenje bajtova – Intel little endian, ARM big endian)
  - pakovanje parametara podrazumeva **transformaciju vrednosti u sekvencu bajtova**
  - klijent i server se moraju **složiti oko određenog kodiranja**
  - kako se predstavljaju **osnovni tipovi podataka** (integer, float, character)
  - kako se predstavljaju **složeni tipovi podataka** (nizovi, strukture, unije)
- Neophodno je da **klijent i server ispravno interpretiraju poruke**, transformišući ih u odgovarajuće mašinski zavisne reprezentacije

Asinhroni RPC

- Izbegava se striktno ponašanje tipa zahtev-odgovor, klijent može da nastavi sa radom bez čekanja na odgovor od strane servera



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

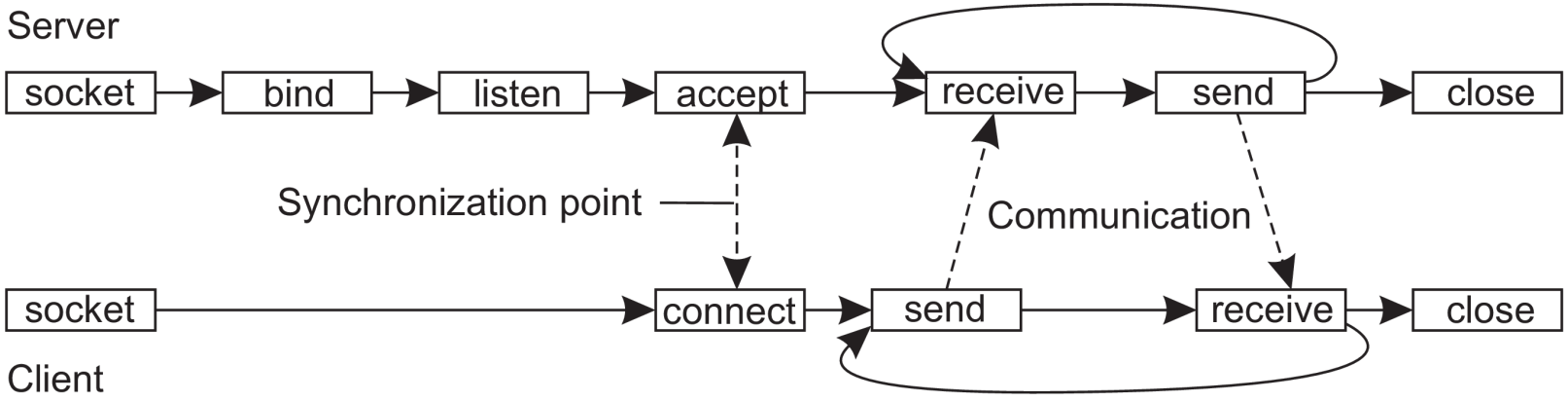
Komunikacija orijentisana ka porukama

- **Soket** (engl. *socket*) interfejs je uveden 1970-tih u Berkley UNIX-u, i kasnije je usvojen kao POSIX standard, podržavaju prolaznu (engl. *transient*) komunikaciju
- **Soket** je **krajnja tačka** (engl. *end point*) **u komunikaciji** na kojoj aplikacije mogu upisivati podatke za slanje putem mreže i pre kojih mogu primati dolazeće podatke,
- **Soket je apstrakcija stvarnog porta** kojeg koristi operativni sistem za određeni transportni protokol
- Kod TCP/IP, soketu se pridružuje **adresa soketa** koja se sastoji od para (*IP adresa*, *broj porta*) na lokalnom čvoru, neophodna je i adresa soketa i na drugom čvoru

```
Socket socket = getSocket(type = "TCP")
connect(socket, address = "1.2.3.4", port = "80")
send(socket, "Hello, world!")
close(socket)
```

Soketi

Soket operacija za TCP/IP	Opis
`socket`	kreiraj novu komunikacionu krajnju tačku
`bind`	pridruži lokalnu adresu soketu
`listen`	reci OS koji je max dozvoljeni broj čekajućih zahteva za konekciju
`accept`	blokiraj pozivaoca dok ne stigne zahtev za konekcijom
`connect`	aktivno pokušaj da uspostaviš konekciju
`send`	pošalji određene podatke preko konekcije
`receive`	primi određene podatke preko konekcije
`close`	prekini konekciju



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

MPI

- **Message Passing Interface** (**MPI**), standard za slanje poruka, projektovan za paralelnu obradu i prolaznu komunikaciju, pruža fleksibilnost u HPC primenama

Operation	Description
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until transmission starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

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

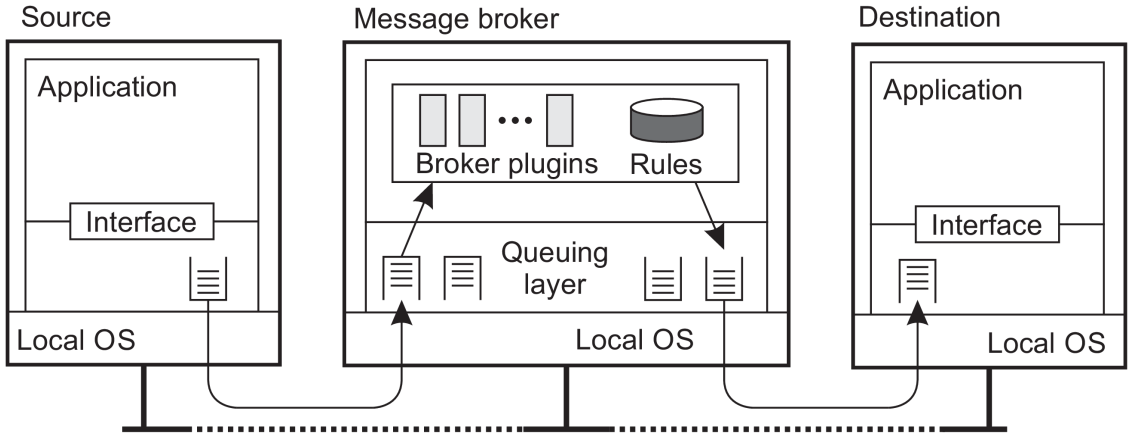
Midlver orijentisan ka porukama (MOM)

- **MOM** omogućava **asinhronu perzistentnu komunikaciju** putem **redova** (engl. *queues*) na **nivou midlvera**, **redovi** odgovaraju **baferima na komunikacionim serverima**, za razliku od soketa i MPI, MOM sistemi za prenos poruka omogućavaju da transfer traje minutima
- **Osnovna ideja u sistemima sa redovima poruka** (engl. *message-queuing systems*) je da aplikacije komuniciraju ubacivanjem poruka u redove, redovima upravljaju **menadžeri redova** koji izvlače poruke iz lokalnih redova i rutiraju ih, osnovni interfejs za rad sa redom:

Operacija	Opis
`put`	dodaj poruku u određeni red
`get`	blokiraj dok specificirani red nije prazan, ukloni prvu poruku
`poll`	proveri da li specificirani red ima poruke i ukloni prvu, nikada ne blokira
`notify`	instaliraj upravljač koji se poziva kada se poruka smesti u specificirani red

Brokeri poruka

- **Sistemi sa redovima poruka** podrazumevaju **zajednički protokol za poruke**, sve aplikacije moraju se složiti oko formata poruke (strukture i reprezentacije podataka)
- **Broker poruka** (engl. *message broker*) transformiše dolazeće poruke tako da ih odredišne aplikacije mogu razumeti
  - ponaša se kao kapija (engl. *gateway*) na aplikacionom nivou
  - pruža rutiranje prema temama (engl. *subjects*) (tj. objavi-pretplati se mogućnosti)
  - omogućava naprednu integraciju poslovnih aplikacija (EAI), odgovoran i za uparivanje aplikacija na osnovu poruka koje se razmenjuju



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

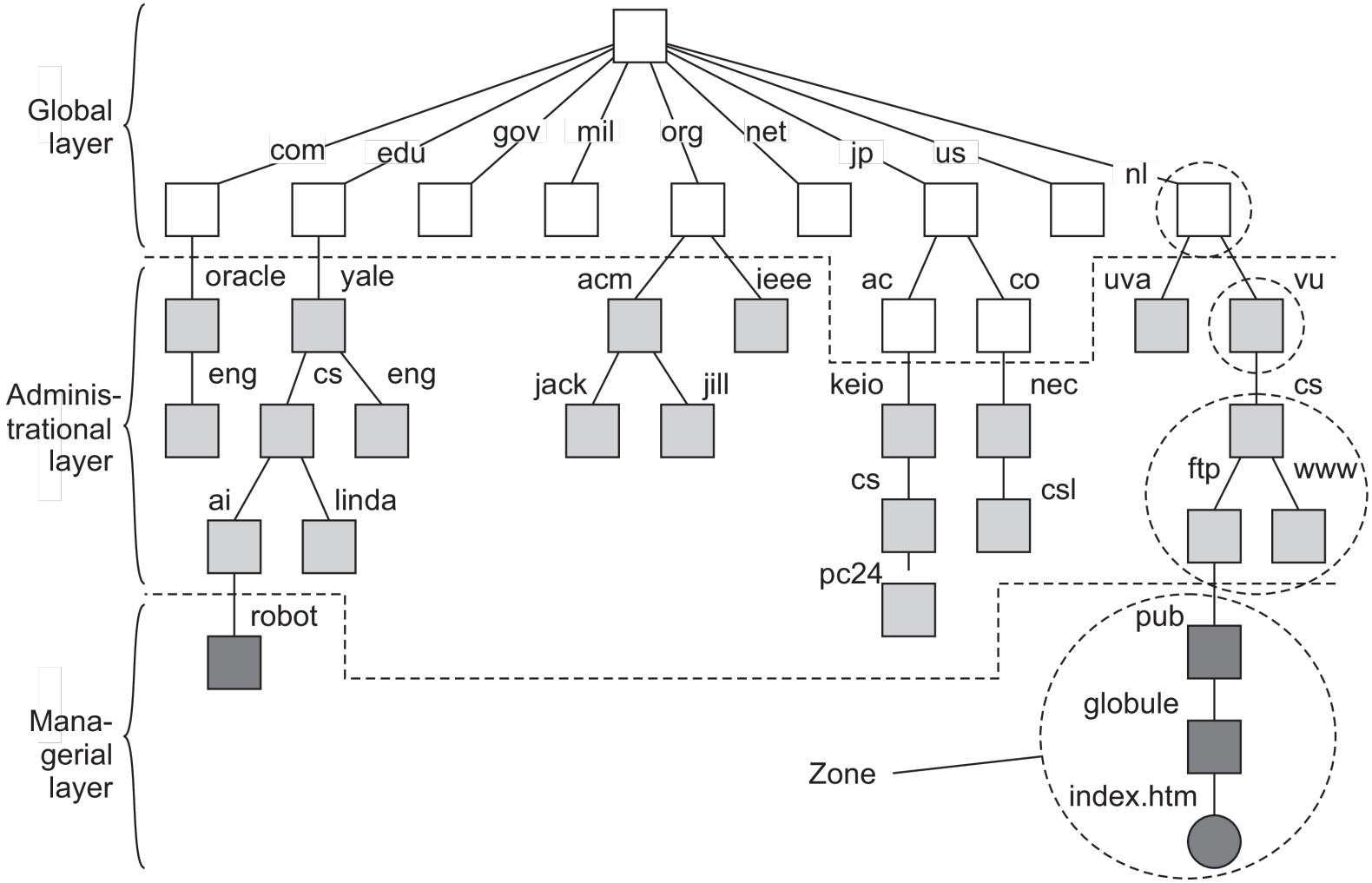
Multikast

- **Multikast** (engl. *multicast*) komunikacija podrazumeva **slanje podataka višestrukim primaocima**, podrazumeva organizaciju čvorova u distribuiranom sistemu u **prekrivajuću mrežu**
- Ako svaki čvor u prekrivajućoj mreži treba da primi poruku, onda se radi o **brodkastu** (engl. *broadcast*)
- Vrste multikasta:
  - **Multikast na nivou aplikacija zasnovan na stablima i mrežama**, za diseminaciju podataka koriste se stabla (engl. *tree*) čvorova ili meš mreža (engl. *mesh networks*) kod koje je neophodan neki vid rutiranja
  - **Multikast zasnovan na plavljenju** (engl. *flooding*), čvor *P* šalje poruku *m* svim svojim susedima, ako ranije nije video poruku *m*, svaki od suseda prosleđuje dalje *m* svim svojim susedima osim *P*
  - **Multikast zasnovan na ogovaranju** (engl. *gossip*), tzv. **epidemijski protokoli** (engl. *epidemic protocols*), lenja propagacija, na kraju svako ažuriranje dospeva do svake od replika, dve vrste epidemija: **anti-entropija** i **širenje glasina** (engl. *rumor spreading*)

Imenovanje u distribuiranim sistemima

- **Ime** u distribuiranom sistemu je **niz bitova ili karaktera** koji se koristi za **obraćanje nekom entitetu** kao što su **resursi** (host, printer, disk, fajl), **procesi**, **poruke**, **konekcije**, …
- **Pristupna tačka** (engl. *access point*) za rad sa entitetima je posebna vrsta entiteta u distribuiranim sistemima čiji naziv je poznat kao **adresa**
- **Identifikatori** su posebna vrsta imena koja na jedinstven način identifikuju entitete. Svaki identifikator odnosi se na najviše jedan entitet, na svaki od entiteta referiše se samo jedan identifikator, identifikator se takođe uvek odnosi na isti entitet
- **Tri klase sistema za imenovanje** (engl. *naming systems*):
  - **ravno imenovanje** (engl. *flat naming*), povezivanje identifikatora sa adresom pridruženih entiteta, pogodno za mašine, primer DHT kod Chord
  - **strukturirano imenovanje** (engl. *structured naming*), imenski prostori, primer DNS
  - **imenovanje zasnovano na atributima** (engl. *attribut-based naming*), koriste parove (*atribut*, *vrednost*), poznati i kao **directory services**, primer LDAP, **directory information base** i **tree** (DIB i DIT)

Primer: DNS implementacija imenskog prostora



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