



SVEUČILIŠTE U ZAGREBU



Fakultet  
elektrotehnike i  
računarstva

# Raspodijeljeni sustavi

**Diplomski studij**

**Informacijska i  
komunikacijska tehnologija:**

Telekomunikacije i informatika

**Računarstvo:**

Programsko inženjerstvo i  
informacijski sustavi

Računarska znanost

**7. Mikrousluge (mikroservisi)**

Ak. god. 2020./2021.

# Creative Commons



- slobodno smijete:

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



- pod sljedećim uvjetima:

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



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

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

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

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

# Sadržaj predavanja

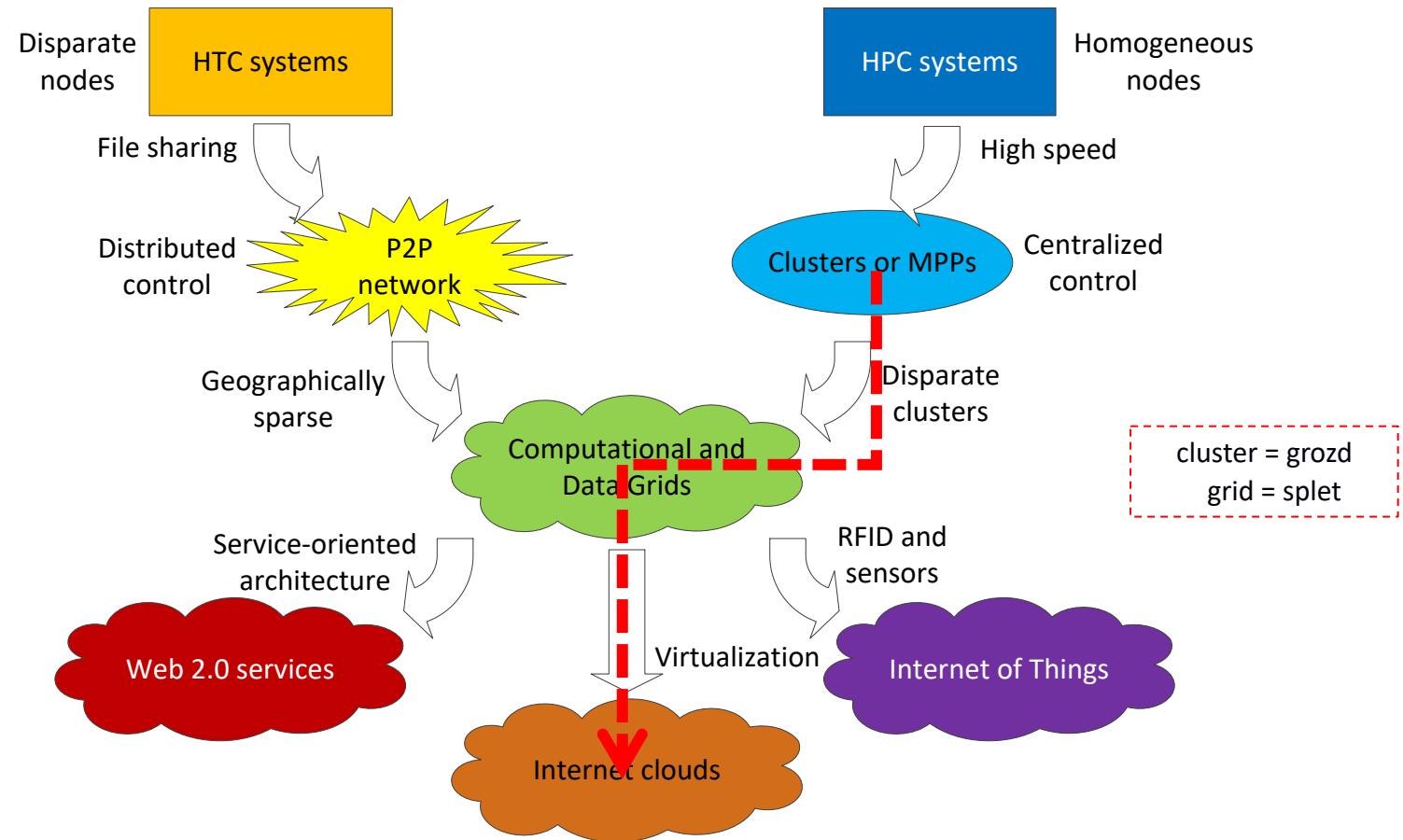
- Uvod
- Grozd računala
- Splet računala
- Računarstvo u oblaku
- Mikrousluge
- Trendovi i smjernice

# Aplikacije koje zahtijevaju obradu velikog broja zadataka

Domena	Primjena
Znanstvene aplikacije	Znanstvene simulacije, analiza genoma, predviđanje potresa, modeliranje globalnog zagrijavanja, vremenska prognoza, itd.
Industrijske aplikacije	Telekomunikacije, isporuka sadržaja, e-poslovanje, bankarstvo, burze, kontrola zračnog prometa, isporuka električne energije, udaljeno učenje, zdravstvo, telemedicina, itd.
Internetske aplikacije	Tražilice, podatkovni centri, nadgledanje prometa, računalna sigurnost, digitalizacija javne uprave, društvene mreže, itd.
Kritične aplikacije	Upravljanje kriznim situacijama, kontrola i zapovijedanje u vojnim zadacima

# Evolucija raspodijeljenog i paralelnog računarstva

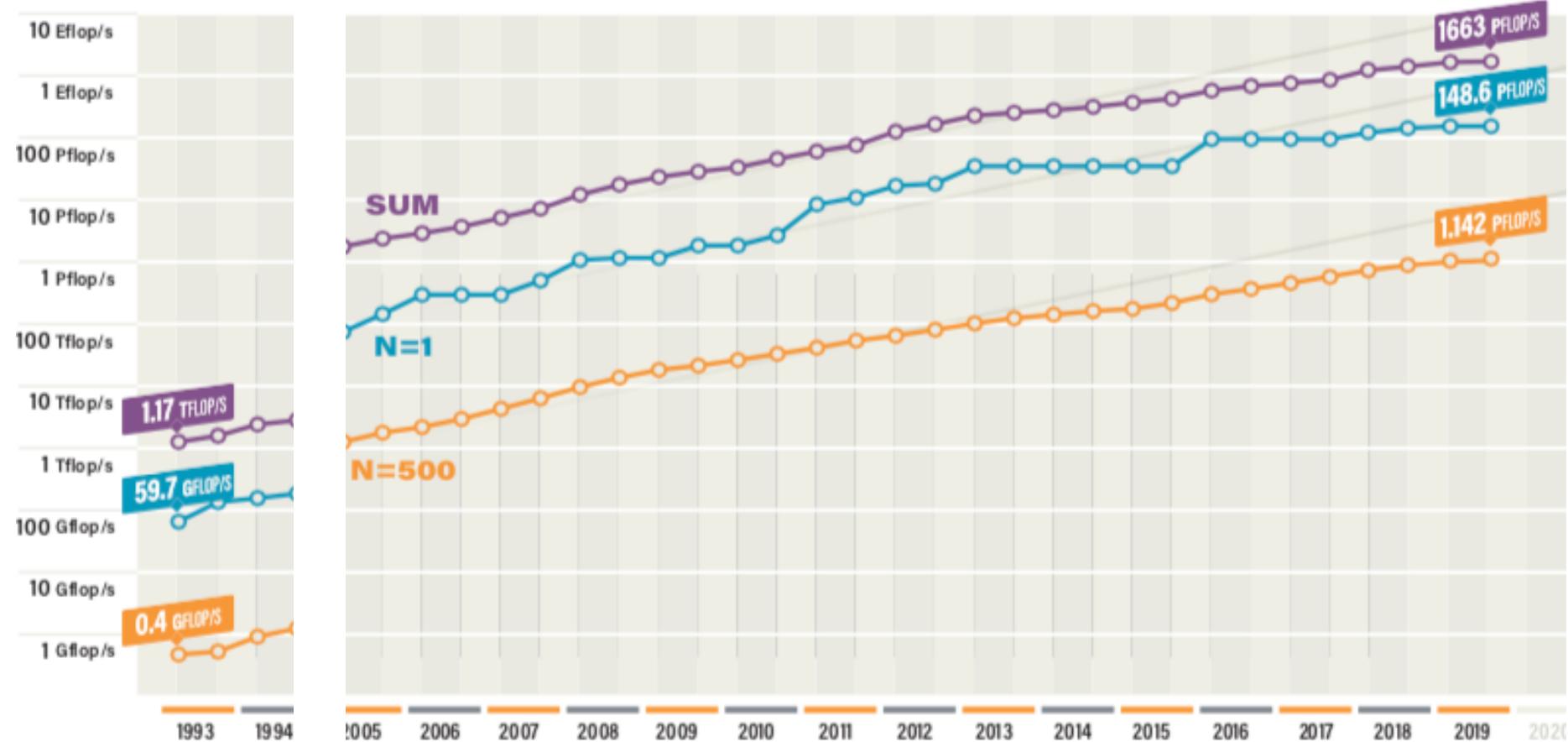
- HTC (High-Throughput Computing)
  - kontinuirana obrada velike količine raznovrsnih i nepovezanih zadataka
- HPC (High-Performance Computing)
  - paralelna obrada velike količine sličnih i povezanih zadataka u kratkom periodu
- MPP (Massively Parallel Processors )
  - centralizirana superračunala



# Usporedba grozdova, spletova i računarstva u oblaku

	Grozd ( <i>Cluster</i> )	Splet ( <i>Grid</i> )	Oblak ( <i>Cloud</i> )
Arhitektura	Skup računala povezanih brzom lokalnom mrežom	Skup udaljenih računala (ili grozdova) povezanih putem Interneta	Virtualizirani grozd računala koja se nalaze u jednom ili više podatkovnih centara
Resursi	Identična ili vrlo slična računala	Raznolika računala	Identična ili vrlo slična računala
Aplikacije	HPC, tražilice	Raspodijeljeno rješavanje problema	Uslužno računarstvo ( <i>utility computing</i> ), pohrana podataka
Primjeri	Google search engine, Cray XK7, BlueGene/Q, itd.	Folding@home, BOINC, SETI@home, itd.	Microsoft Azure, Amazon EC2, Google App Engine, itd.

# Evolucija performansi superračunala



# Evolucija arhitekture superračunala



SIMD – Single Instruction Multiple Data

MPP – Massively Parallel Processors

SMP – Symmetric multiprocessing

Constellations – više procesora na čvoru nego čvorova

# Sadržaj predavanja

- Uvod
- Grozd računala
  - Evolucija superračunala
  - Definicija i vrste grozda računala
  - Primjeri grozda računala
- Splet računala
- Računarstvo u oblaku
- Mikrousluge

# Definicija grozda računala (*computer cluster*)

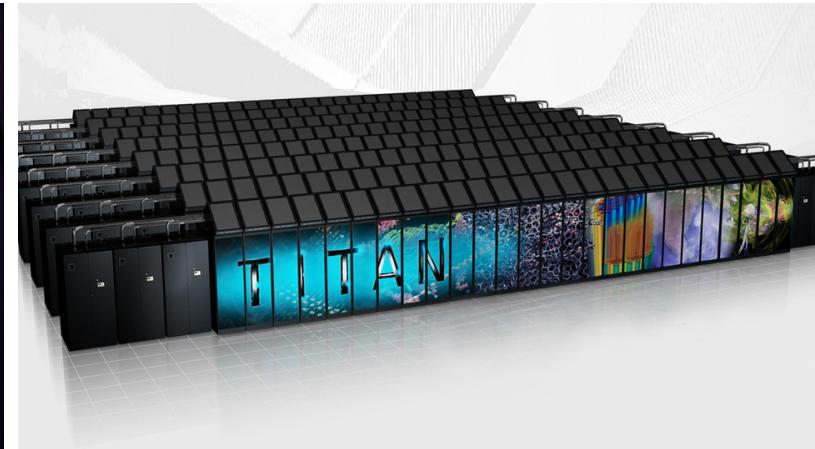
- Skup samostalnih računala
- Međusobno su povezana brzom lokalnom mrežom
- Rade kooperativno
- Čine jedan jedinstveni (integrirani) računalni resurs



JUQUEEN, Njemačka



Tianhe-2 (MilkyWay-2), Kina



Cray

# Vrste i primjene grozda računala

- Računarski grozd (*compute cluster*)
  - Upotrebljava se za kolektivno i kooperativno izvođenje jednog zahtjevnog posla (npr. simulacija vremenskih uvjeta)
  - Vrlo malo korištenje I/O operacija
- Grozd visoke raspoloživosti (*high-availability cluster*)
  - Redundantna računala preuzimaju posao uslijed neispravnosti drugih
  - Ne postoji jedinstvene točke ispada grozda
  - Koristi se za raspodijeljene baze podataka i ostale poslovne aplikacije
- Grozd za raspodjelu opterećenja (*load-balancing cluster*)
  - Raspodjelom opterećenja postiže visoke performanse
  - Međuoprema vrši raspodjelu opterećenja i migraciju procesa između računala

# Najsnažniji grozdovi u svijetu i Europi (kraj 2019.)

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	<b>Summit</b> - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
2	<b>Sierra</b> - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	<b>Sunway TaihuLight</b> - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371

# Grozdovi u HR

- Isabella na Srcu
  - <https://www.srce.unizg.hr/isabella/>
- Superačunalo Bura na Sveučilištu u Rijeci
  - HPC
  - pokrenuto 2016.
  - <https://www.hpc.uniri.hr/>
- HR-ZOO – Hrvatski znanstveni i obrazovni oblak
  - dovršetak projekta sredina 2021.
  - <https://www.srce.unizg.hr/hr-zoo>



# Isabella

- 135 računala (270 CPU procesora)
- 3.100 CPU procesorskih jezgri
- 16 TB radne memorije
- 756 TB dijeljenog podatkovnog prostora
- lokalna mreža visokog stupnja propusnosti
  - 2 x HP ProCurve 2848 1 Gb/s Ethernet preklopnik (96 pristupa)
  - 1 x Voltaire Grid Director QDR (40 Gb/s) Infiniband preklopnik (36 pristupa)
  - 6 x Mellanox SX6025 (56 Gb/s) Infiniband preklopnik (36 pristupa)
- temelji se na sustavima:
  - 28 x HP ProLiant SL250s
  - 12 x HP ProLiant SL250s (12)
  - 8 x HP ProLiant SL230s Gen8
  - 8 x HP ProLiant SL230s Gen8
  - 76 x Lenovo NeXtScale nx360 M5
  - 3 x Dell EMC PowerEdge C4140



# Sadržaj predavanja

- Uvod
- Grozd računala
- Splet računala
  - Definicija i vrste spleta računala
  - Primjeri spleta računala
- Računarstvo u oblaku
- Mikrousluge
- Trendovi i smjernice

# Splet računala (grid)

- Ideja: Ukoliko nam je za izvođenje posla potreban slabo povezan grozd (samo neka računala često komuniciraju pri izvođenju posla) posao možemo raspodijeliti među geografski udaljenim grozdovima – ***virtualno superračunalo***
- Definicija spleta računala
  - Skup međusobno udaljenih heterogenih računalnih sredstava
  - Međusobno su (slabo) povezana Internetom
  - Rade na zajedničkom zadatku radi postizanja zajedničkog cilja
- Računalna sredstava su najčešće pod različitim administrativnim domenama (koje pripadaju različitim organizacijama)

# Vrste i primjene spleta računala

Vrsta	Primjena	Primjeri
Računarski i podatkovni splet računala	Spremanje i obrada velike količine podataka, znanstveni eksperimenti, modeliranje, simulacije, itd.	TeraGrid (SAD), EGEE (EU), DataGrid (EU), ChinaGrid (Kina), NAS (SAD - NASA), LCG (Švicarska - Cern), itd.
Informacijski splet računala	Semantički web i upravljanje znanjem	Semantic Grid, Ontology Platform, D4Science, Information Power Grid (SAD - NASA), itd.
Poslovni splet računala	Poslovne web aplikacije, dubinska analiza velike količine poslovnih podataka	BElngGrid (EU), HP eSpeak, Oracle Grid Engine, itd.
Volonterski splet računala	Znanstvene analize	SETI@Home, Einstein@Home, BOINC, Folding@Home, itd.

# Primjeri spletova računala

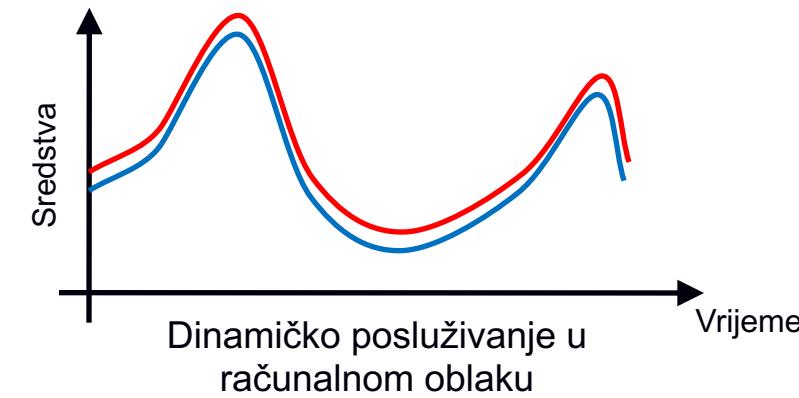
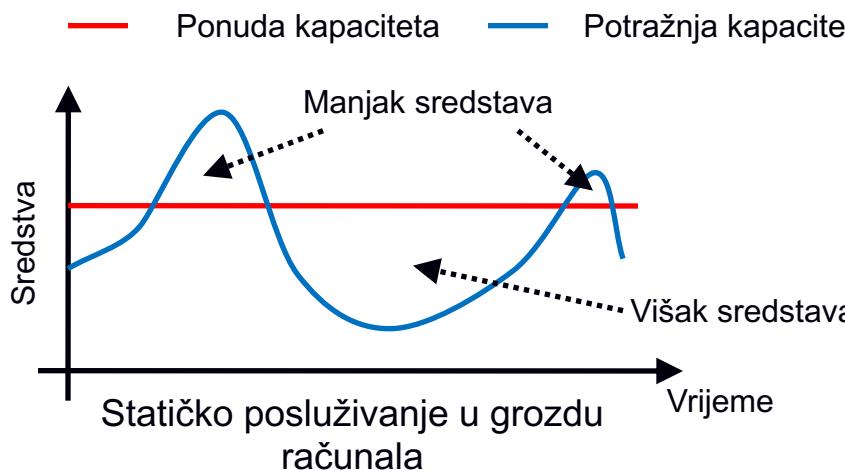
- [Bitcoin Network](#) - 80.000.000 PFLOPS (2019.)
- [BOINC](#) – 22 PFLOPS (2018.)
- [Folding@home](#) – 101 PFLOPS (2016.)
- [Einstein@Home](#) – 3,489 PFLOPS (2018.)
- [SETI@Home](#) – 0,890 PFLOPS (2018.)
- [MilkyWay@Home](#) – 0,941 PFLOPS (2018.)
- [GIMPS](#) – 0,558 PFLOPS (2019.)

# Sadržaj predavanja

- Uvod
- Grozd računala
- Splet računala
- **Računarstvo u oblaku**
  - Definicija i vrste računalnog oblaka
  - Modeli usluga i stranke u računalnom oblaku
  - Prednosti i nedostaci računalnog oblaka
  - Primjeri računalnog oblaka
  - Kontejneri
- Mikrousluge
- Trendovi i smjernice

# Računarstvo u oblaku

- Uslužno računarstvo (*utility computing*)
- Računalni resurs se ne kupuje, već se iznajmljuje po potrebi
- Plaća se onoliko računalnih resursa koliko se doista i koristi po principu „*pay as you go*”
- Privid neograničenosti računalnih resursa u oblaku
- Kapitalni trošak (CapEx) kupovine računalne opreme postaje operativni trošak (OpEx) iznajmljivanja resursa u oblaku
- Učinkovito korištenje računalnih resursa



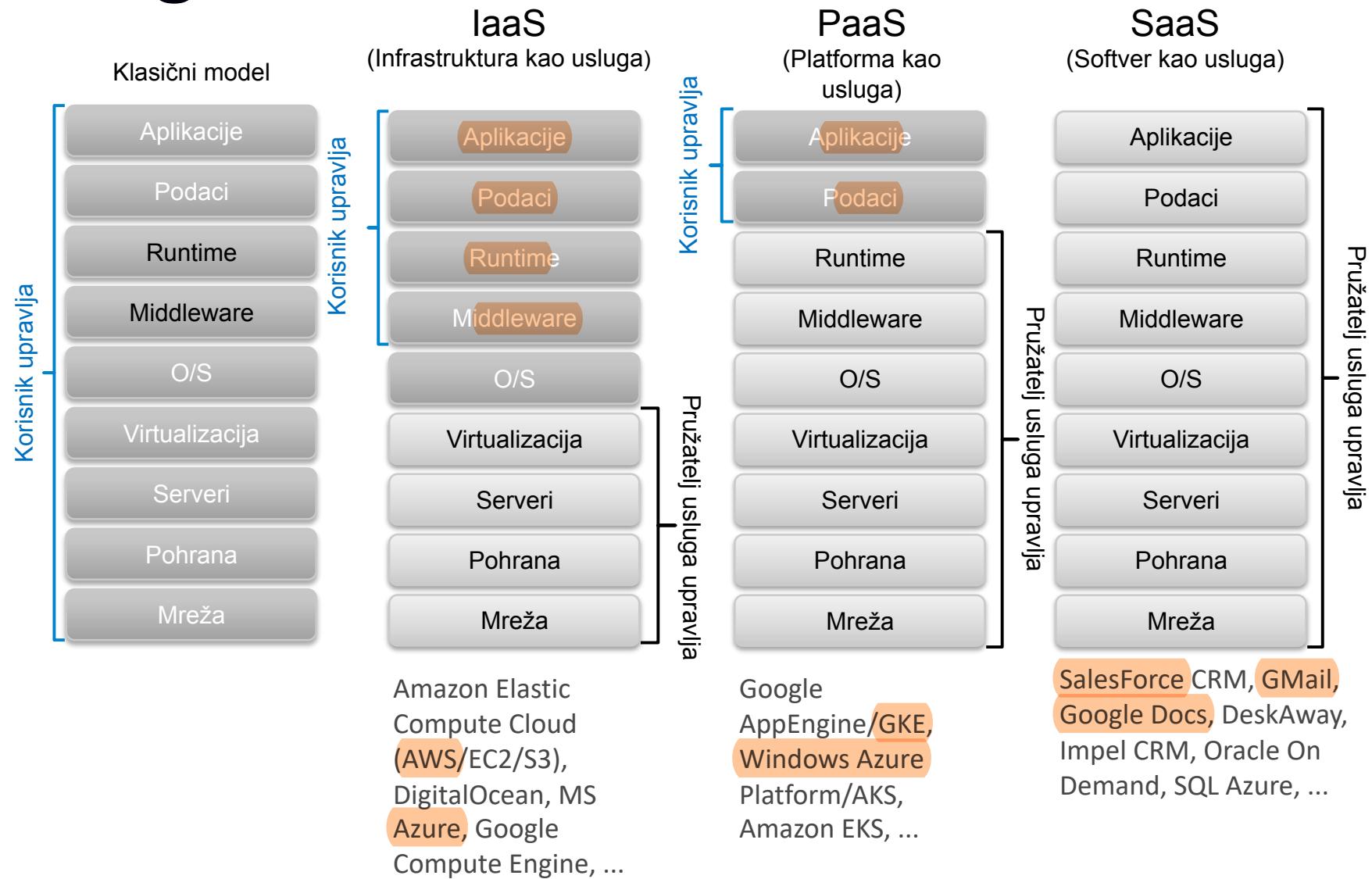
# Vrste računalnog oblaka prema smještaju i svrsi

- Javni oblak (*public*)
  - Iznajmljuje se za **javnu uporabu**
  - U vlasništvu je organizacije koja prodaje usluge u oblaku
- Privatni oblak (*private*)
  - U **privatnom vlasništvu poduzeća i samo** to poduzeće ga koristi
  - Ne smatra pravim „oblakom“
- Zajednički oblak (*community*)
  - Nekoliko **organizacija dijeli jednu infrastrukturu**
- Hibridni oblak (*hybrid*)
  - **Kompozicija** 2 ili više oblaka različitih vrsta

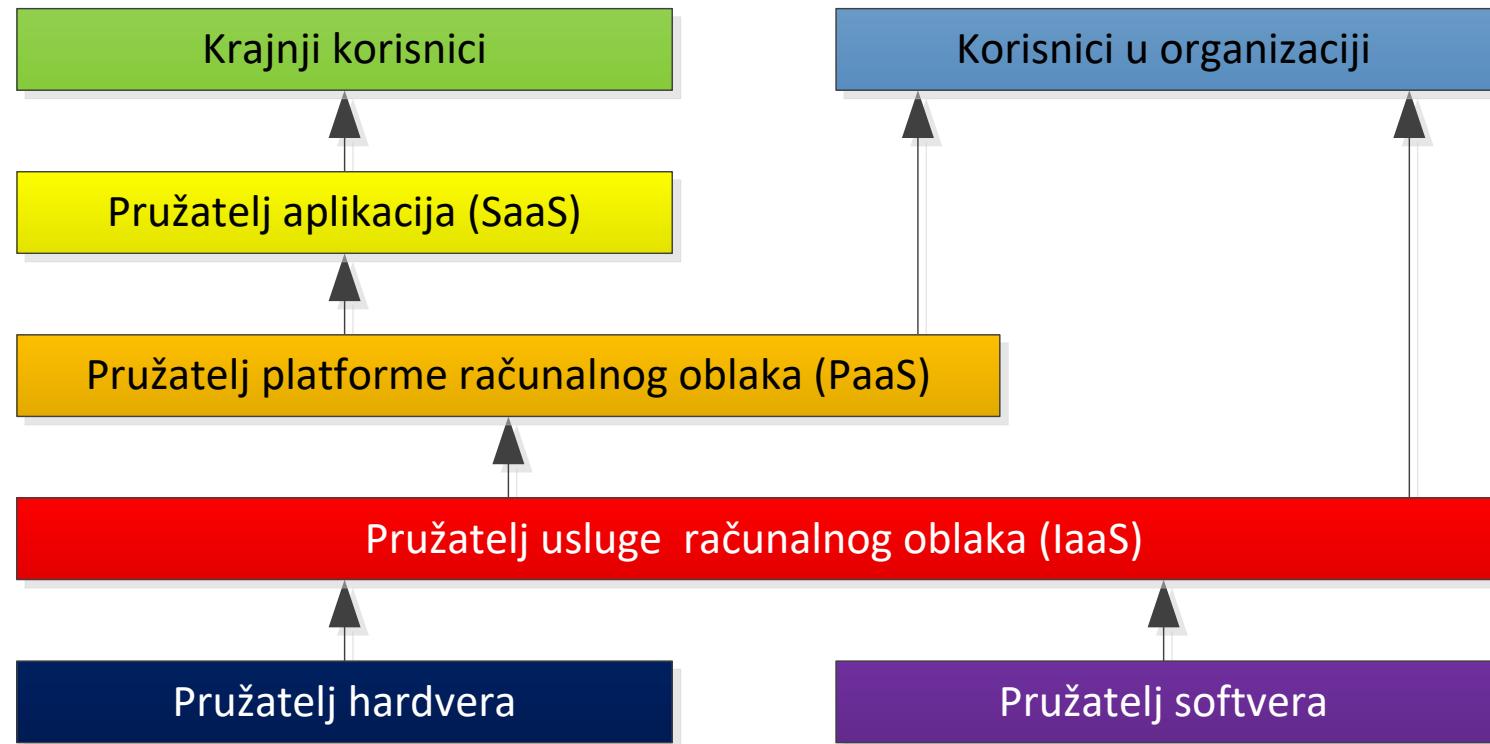
# Najvažniji koncepti

- Apstrakcija implementacije
  - Korisnik i razvijatelj ne znaju specifikaciju sustava na kojem će se aplikacije i usluge izvoditi
  - Podaci spremljeni na lokacije koje nisu poznate
  - Administracija sustava nije pod kontrolom razvijatelja
  - Pristup aplikacijama i uslugama je omogućen putem Interneta
- Virtualizacija
  - Omogućuje izvršavanje više operacijskih sustava na jednom fizičkom ili na više fizičkih računala
  - Isto vrijedi za pohranu podataka
  - Resursi se mogu dijeliti ili udruživati.
  - Računala mogu imati podršku za virtualizaciju (tehnologija hipervizora)
  - Primjeri: Xen, VMware, Wine, ...

# Modeli usluga



# Stranke u računalnom oblaku



Izvor: [1]

# Prednosti računarstva u oblaku u odnosu na vlastitu opremu

- plaćanje prema korištenju
- nadogradnja resursa na zahtjev
- elastičnost resursa
  - „automatsko“ proširivanje mrežnih i računalnih kapaciteta ovisno o opterećenju
- ne zahtijevaju se kapitalni troškovi za izgradnju infrastrukture
- povećana pouzdanost
- jednostavnije upravljanje održavanjem i nadogradnjom sustava
- usluge dostupne preko web-preglednika

# Nedostaci računarstva u oblaku

- nemogućnost prilagodbe klijentu kao u slučaju kada on sam upravlja sam svojim fizičkim računalima
- aplikacije koje su izvan oblaka obično imaju veće mogućnosti, nego aplikacije u oblaku (SaaS)
  - iz godine u godinu se smanjuju razlike, a kod nekih usluga i obrnuto
- ako aplikacija zahtjeva prijenos velikih količina podataka oblak nije najbolje rješenje
- ovisnost o jednom pružatelju usluga u oblaku
  - ovo ovisi o sustavu i tehnologijama koje se koriste
- usvajanje novog načina razvoja aplikacija
- privatnost i sigurnost podataka
  - neznanje o lokaciji spremanja podataka (sada se zna regija - GDPR)
  - lokalnih zakona o sigurnosti podataka (GDPR)

# Primjeri računalnog oblaka



- <https://aws.amazon.com>



Analytics



Application Integration



AR & VR



AWS Cost Management



Blockchain



Business Applications



Compute



Customer Engagement



Database



Developer Tools



End User Computing



Game Tech



Internet of Things



Machine Learning



Management & Governance



Media Services



Migration & Transfer



Mobile



Networking & Content  
Delivery



Quantum Technologies



Robotics



Satellite



Security, Identity &  
Compliance



Storage



See All Products

# Primjeri računalnog oblaka



- <https://cloud.google.com>

GOOGLE CLOUD PLATFORM

## AI and Machine Learning

[Speech-to-Text](#) · [Vision](#) · [Translation](#) · More

## API Management

[Apigee API Platform](#) · [Cloud Endpoints](#) · More

## Compute

[Compute Engine](#) · [Cloud GPUs](#) · More

## Hybrid and Multi-cloud

[Anthos](#) · [Migrate for Anthos](#) · [GKE](#) · More

## Data Analytics

[BigQuery](#) · [Cloud Datalab](#) · More

## Databases

[Cloud SQL](#) · [Cloud Firestore](#) · More

## Developer Tools

[Cloud Build](#) · [Cloud Code](#) · [Cloud SDK](#) · More

## Migration

[Data Transfer](#) · [VM Migration](#) · More

## Networking

[DNS](#) · [CDN](#) · [Virtual Private Cloud](#) · More

## Security and Identity

[Shielded VMs](#) · [Cloud IAM](#) · More

## Serverless Computing

[Cloud Run](#) · [App Engine](#) · [Cloud Functions](#) · More

## Storage

[Cloud Storage](#) · [Persistent Disk](#) · More

# Primjeri računalnog oblaka



- <https://azure.microsoft.com/en-us/>

⚡ Featured      Internet of Things

AI + Machine Learning      Management

Analytics      Media

Blockchain      Migration

Compute      Mixed Reality

Containers      Mobile

Databases      Networking

Developer Tools      Security

DevOps      Storage

Hybrid      Web

Identity      Windows Virtual Desktop

Integration

**Featured**  
Explore some of the most popular Azure products

**Virtual Machines**  
Provision Windows and Linux virtual machines in seconds

**Azure SQL Database**  
Managed, intelligent SQL in the cloud

**Azure Cosmos DB**  
Globally distributed, multi-model database for any scale

**Azure Kubernetes Service (AKS)**  
Simplify the deployment, management, and operations of Kubernetes

**Cognitive Services**  
Add smart API capabilities to enable contextual interactions

**Windows Virtual Desktop**  
The best virtual desktop experience, delivered on Azure

**App Service**  
Quickly create powerful cloud apps for web and mobile

**PlayFab**  
The complete LiveOps back-end platform for building and operating live games

**Azure Functions**  
Process events with serverless code

**Azure Quantum**  
Experience quantum impact today on Azure

# Primjeri računalnog oblaka



- <https://www.digitalocean.com>

## FEATURED PRODUCTS



### Droplets

Scalable compute services



### Kubernetes

Managed Kubernetes clusters



### Databases

Worry-free setup & maintenance



### Spaces

Simple object storage

## COMPUTE

Droplets

Kubernetes

## STORAGE

Spaces Object Storage

Volumes Block Storage

## MANAGED DATABASES

MySQL

PostgreSQL

Redis™

## DEVELOPER TOOLS

API

CLI

Monitoring

Teams

## NETWORKING

Cloud Firewalls

Load Balancers

Floating IPs

DNS

# Računalni oblaci u HR

- <https://ictmarketplace.hr> (t-com)



- <https://sysportal.carnet.hr>



- <http://www.megatrend.com/usluge/cloud-usluge/>

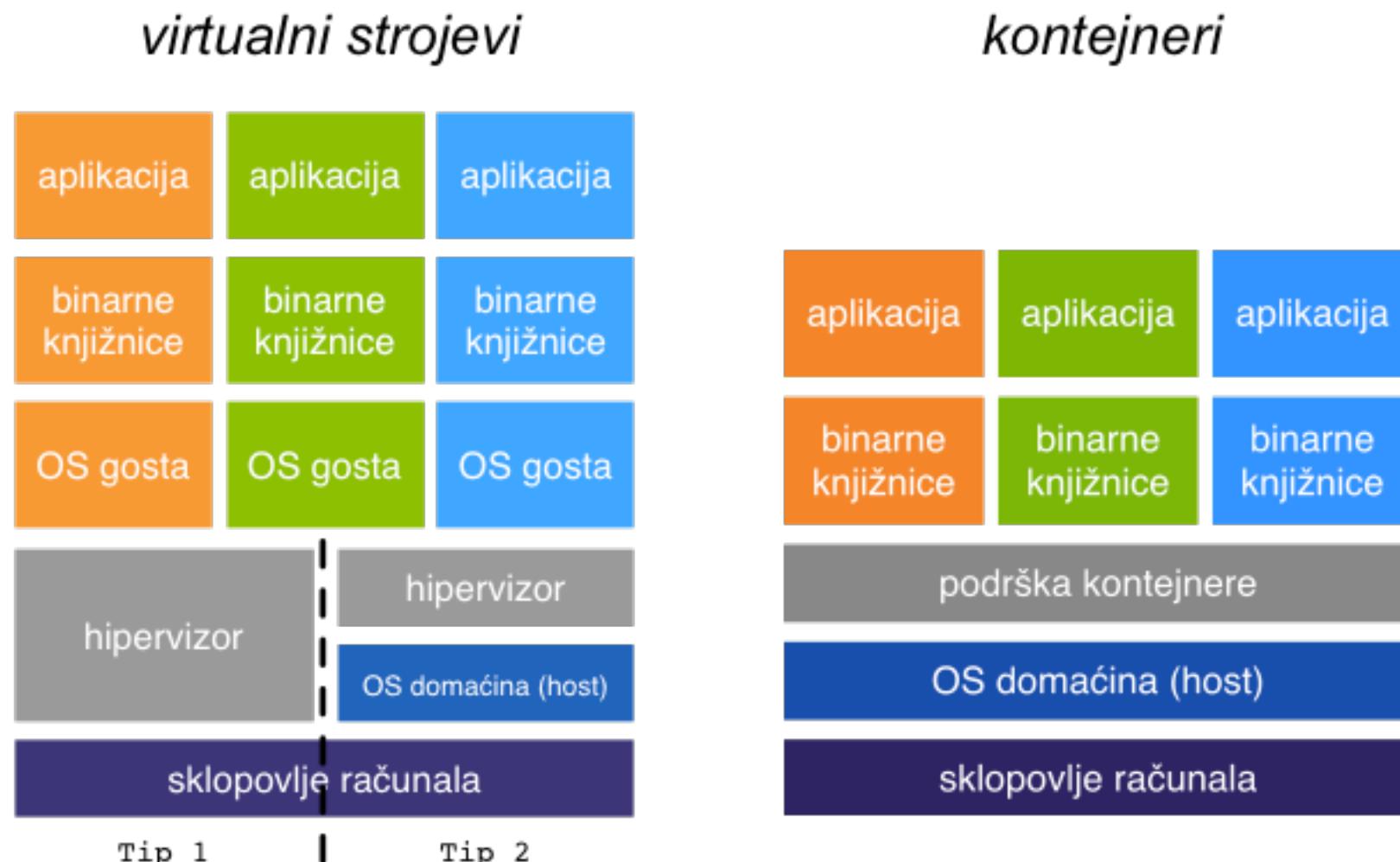


- <https://www.a1.hr/poslovni/ict-rjesenja/aplikacije>



# Kontejneri

# Razlika između kontejnera i virtualnog stroja



# Prednosti i mane kontejnera

- Prednosti:
  - ne pokreće čitav operacijski sustav
  - brže se pokreće
  - koristi manje resursa (procesora, memorije i diska)
- Mane:
  - slabija kontrola korištenja resursa
  - slabija izolacija između kontejnera
    - potencijalni sigurnosni problem
  - svaki kontejner mora koristiti isti kernel domaćinovog OS-a

# Primjeri implementacija

- Imunes – <http://imunes.net> (UniZG – FER – ZZT)
  - prvenstveno za IP mreže
- Docker – <https://www.docker.com>
  - najpoznatiji
- runC – <https://github.com/opencontainers/runc>
- rkt – <https://github.com/coreos/rkt>
  - razvija ga [CoreOS](#)
- containerd, LXC/LXD, OpenVZ, systemd-nspawn, machinectl, qemu-kvm, lkvm, ...
- Više se može naći [ovdje](#).

# Open Container Initiative specification

- <https://www.opencontainers.org>
- Osnovana 2015. na inicijativu Dockera i ostalih
- Definira:
  - izvršnu specifikaciju
  - specifikaciju slika

# Tehnologije potrebne za izvršavanje kontejnera

- Kontejner je zapravo skup izoliranih procesa u korisničkom prostoru operacijskog sustava (Linux/Windows)
- Tehnologije za izvršavanje na Linuxu:
  - **Namespaces** - ograničava što proces vidi od okoline u kojoj se izvršava:
    - Unix Timesharing System (*hostname*), Process IDs, Mounts, Network, User IDs, ...
  - **chroot**
    - promjena korijenskog direktorija nekog procesa
  - **Cgroups** - limitira korištenje resursa
    - memorija, procesor, I/O (količina podataka prenesena), broj procesa, ...

# Docker - [www.docker.com](http://www.docker.com)



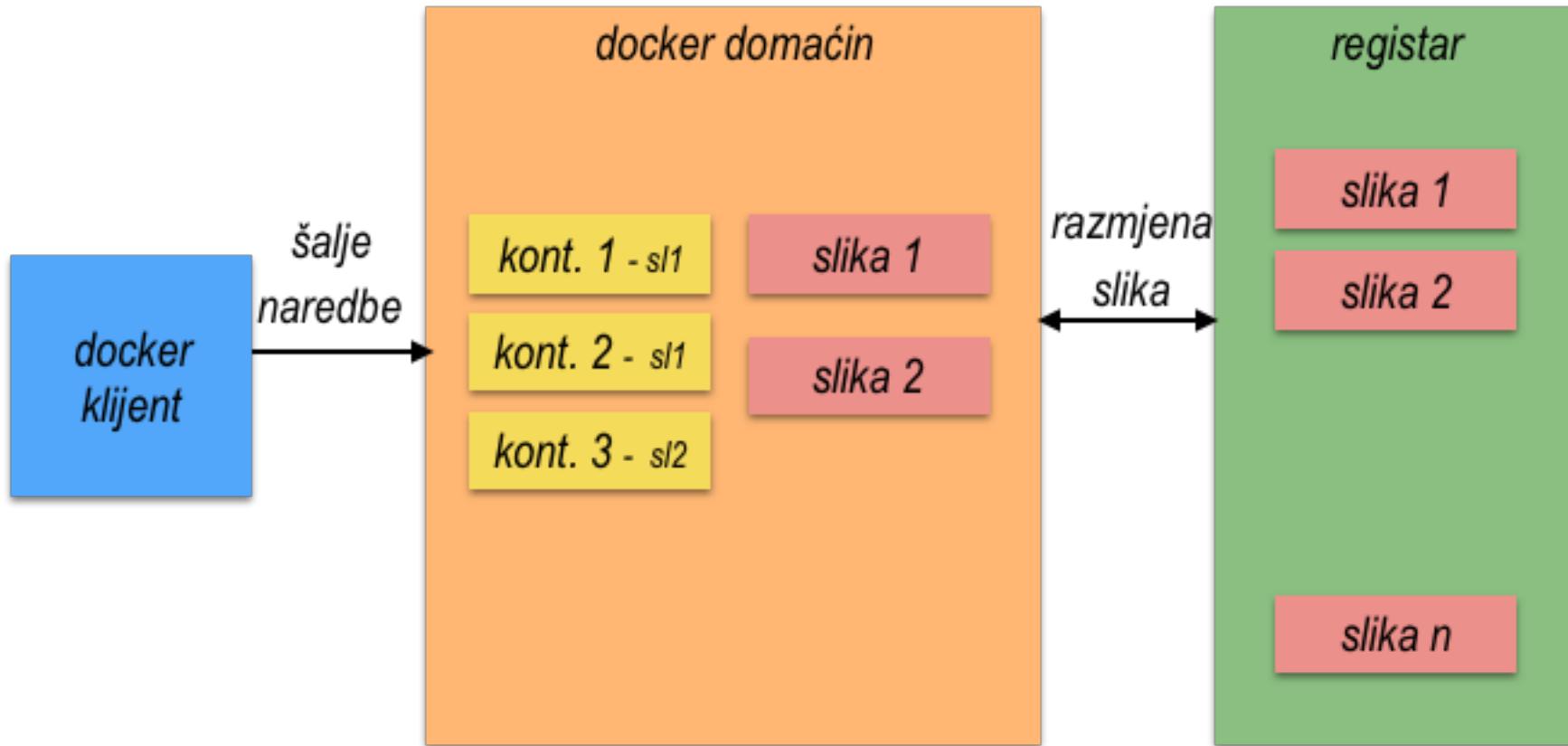
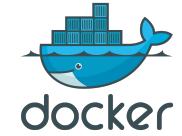
- Pojmovi:
  - *image* - slika OS-a
  - *container* - pokrenuta slika OS-a
  - *docker host* - računalo na kojem je pokrenut sustav docker
  - *docker hub* - portal za razmjenu slika (registar)
    - postoje i drugi: quay.io, gcr.io, registry.redhat.io
  - Dockerfile - tekstualna datoteka s konfiguracijom za izgradnju slike i za pokretanje kontejnera
- Detaljne video materijale možete naći [ovdje](#).

# Aspekti Dockerovog uspjeha

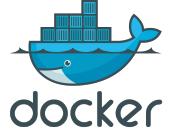


- sve su napravili da se može koristiti jednostavno
- 3 aspekta Dockera:
  - izrada slika
    - svu konfiguraciju stavljamo u datoteku (Dockerfile)
    - izgrađujemo sliku na temelju konfiguracije
    - sliku možemo spremiti u datoteku
    - slike se sastoje od nepromjenjivih slojeva (Union file system)
  - isporuka slika
    - sliku možemo podijeliti na docker hubu
    - lagano možemo pretraživati i skidati slike
    - imenovanje slika:
      - slike pojedinaca <repozitorij>/<proizvod>:<tag>
      - službene slike <proizvod>:<tag>
  - pokretanje slika
    - kontejner je pokrenuta slika

# Arhitektura i izvršavanje



# Izvršavanje Dockera na različitim OS-ovima



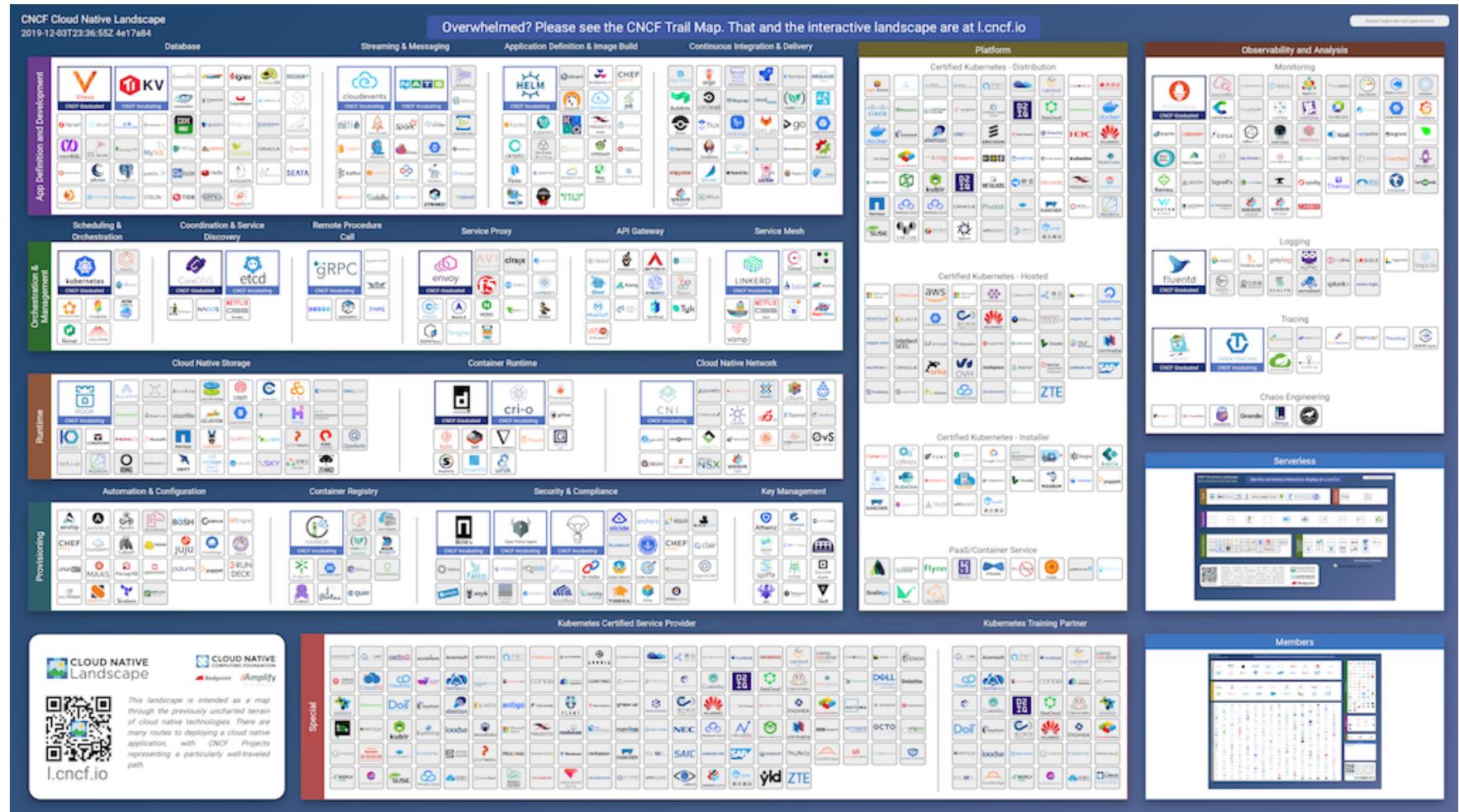
- Docker se izvršava na Linuxu
  - Od 2017. Windows aplikacije se mogu staviti u docker kontejner i pokretati na Windows strojevima
- Za izvršavanje na Windowsima ili Macu je potreban virtualni stroj s linuxom:
  - Docker Machine koji koristi VirtualBox za stvaranje stroja (boot2docker Linux distribution) - danas se rijetko koristi
    - konfigurira varijable okoline da bi se klijent mogao spojiti na taj stroj
  - **Docker Desktop**
    - na Macu koristi *native* Hypervisor.framework
    - na Windowsima koristi Microsoft Hyper-V

# Što kada imamo puno kontejnera?

- Potreba za upravljanjem kontejnerima na puno virtualnih računala:
  - Docker Swarm, Apache Mesos, Kubernetes, ...
- **Kubernetes (K8s):**
  - Automatsko postizanje i održavanje zadanog stanja aplikacije
  - Optimalno korištenje dostupnih resursa računalnog grozda
- *Service mesh*
  - Infrastruktura koja služi za povezivanje usluga u kontejnerima
  - Mogućnosti: otkrivanje usluga, upravljanje opterećenjem, praćenje usluga, autentikacija i autorizacija, šifriranje prometa, upravljanje ispadima
  - Implementacije: **Istio**, Lyft, Linkerd, Consul, ...
- Multicloud
  - Kontejneri se nalaze u više oblaka

# Cloud Native Computing Foundation ([CNCF](#))

- [CNCF Landscape](#)



# Kubernetes (K8s)



- Nastao u Googleu na temelju Borga (2014.)
- Danas ga održava CNCF
- Služi za orkestraciju kontejnera
- Neovisan o oblaku u kojem se izvršava
- Većina pružatelja oblaka pruža K8s kao PaaS ili IaaS
- Mi definiramo krajnje stanje, a K8s se brine da ga održi
- Svojstva:
  - Horizontalno skaliranje
  - Otkrivanje usluga
  - Uravnoteženje opterećenja
  - Samoizlječenje (*self-healing*)
  - Zamjena verzija bez zaustavljanja produkcije
  - Upravljanje konfiguracijama
  - Upravljanje tajnama (zaporke, ...)



# Kubernetes (K8s) – koncepti

- *node* (čvor) – jedno virtualno računalo na kojem se izvršava k8s
- *pod* (kapsula) – najmanji koncept koji se isporučuje, u sebi može imati više kontejnera
- *replica set* – pravila za repliciranje, pazi da određeni broj replika imamo u sustavu
- *deployment* – pravila za čitavu aplikaciju, brine se za verzije slika
- *service* – kada se izvana pristupi usluzi onda se promet preusmjeri na neki od kapsula i na određena vrata (tj. kontejner unutar te kapsule)
- *namespace* – odvaja dijelove k8s grozda
  - može se definirati ograničenje korištenja resursa (*quota*)
  - mogu se definirati prava korisnika
  - obično devops tim definira *namespace* i daje nam prava
- *label* – oznaka nekog koncepta
- *selector* – definira na koje koncepte se odnosi pojedini dio konfiguracije. Odabire se na temelju oznaka (*label*)

# Kubernetes (K8s) – arhitektura



- Dvije vrste čvorova:
  - Glavni (*master*) čvor:
    - **etcd** – raspodijeljena baza podataka sa svim elementima grozda
    - **kube-apiserver** – prima naredbe od kubectl i provodi ih
    - **kube-scheduler** – respordeđuje kapsule na radne čvorove (memorija, procesor, konflikti, ...)
    - **kube-controller-manager** - izvršava ono što je zapisano u etcd
    - Za pouzdani rad trebamo 3-5 instanci
  - Čvor radnik (*worker*):
    - **kapsule (pods)** – u njima se izvršavaju kontejneri
    - **kublet (Node Agent)** – prati što se događa i komunicira s glavnim čvorom (kontroler manager)
    - **kube-proxy (networking component)** – omogućuje uslugama da se neka vrata vide izvana
    - **container runtime (CRI - docker, rkt, ...)** – služi za izvršavanje kontejnera
    - na njima se izvršavaju naši programi, obično ih ima više/puno

# Istio



- Automatsko upravljanje opterećenjem za protokole: HTTP, gRPC, WebSocket i TCP
- Fina kontrola prometa: pravila usmjerenja, ponovno slanje zahtjeva, preusmjerenje u slučaju grešaka, ...
- Definiranje politika: prava pristupa, ograničenja (*limits, quota*)
- Automatska metrika, logovi, *traces*
- Sigurna komunikacija između usluga sa zaštitom identiteta i kriptiranjem komunikacije
- Arhitektura: [ovdje](#)

# Istio – komponente i arhitektura



- Komponente:
  - Envoy proxy – izvršava se u svakoj kapsuli (*side car*)
    - sva komunikacija između usluga ide preko njega
  - Mixer – komunicira s envoyem
    - šalje pravila po kojima envoy radi
  - Pilot
    - otkrivanje usluga
    - upravljanje prometom za inteligentno usmjeravanje (npr. A/B testovi, canary, ...)
    - otpornost na kvarove (*resiliency*): vrijeme odziva, ponovno slanje zahtjeva, osigurač, ...
  - Citadel
    - sigurnost, certifikati, identiteti
  - Galley
    - upravljanje konfiguracijama

# Mikrousluga

# Motivacija

- Aplikacije se nalaze na svakom pokretnom uređaju (telefoni, tablet), u oblaku na tisućama računala
- Očekivanja:
  - vrijeme odgovora je u milisekundama (za ljude i za strojeve)
    - utjecaj na mrežu i procesiranje
  - usluga radi stalno (100% vremena)
    - bez obzira na to koliko korisnika ju trenutno koristi

# Usporedba arhitekture usluga (2005-2017)

~2005

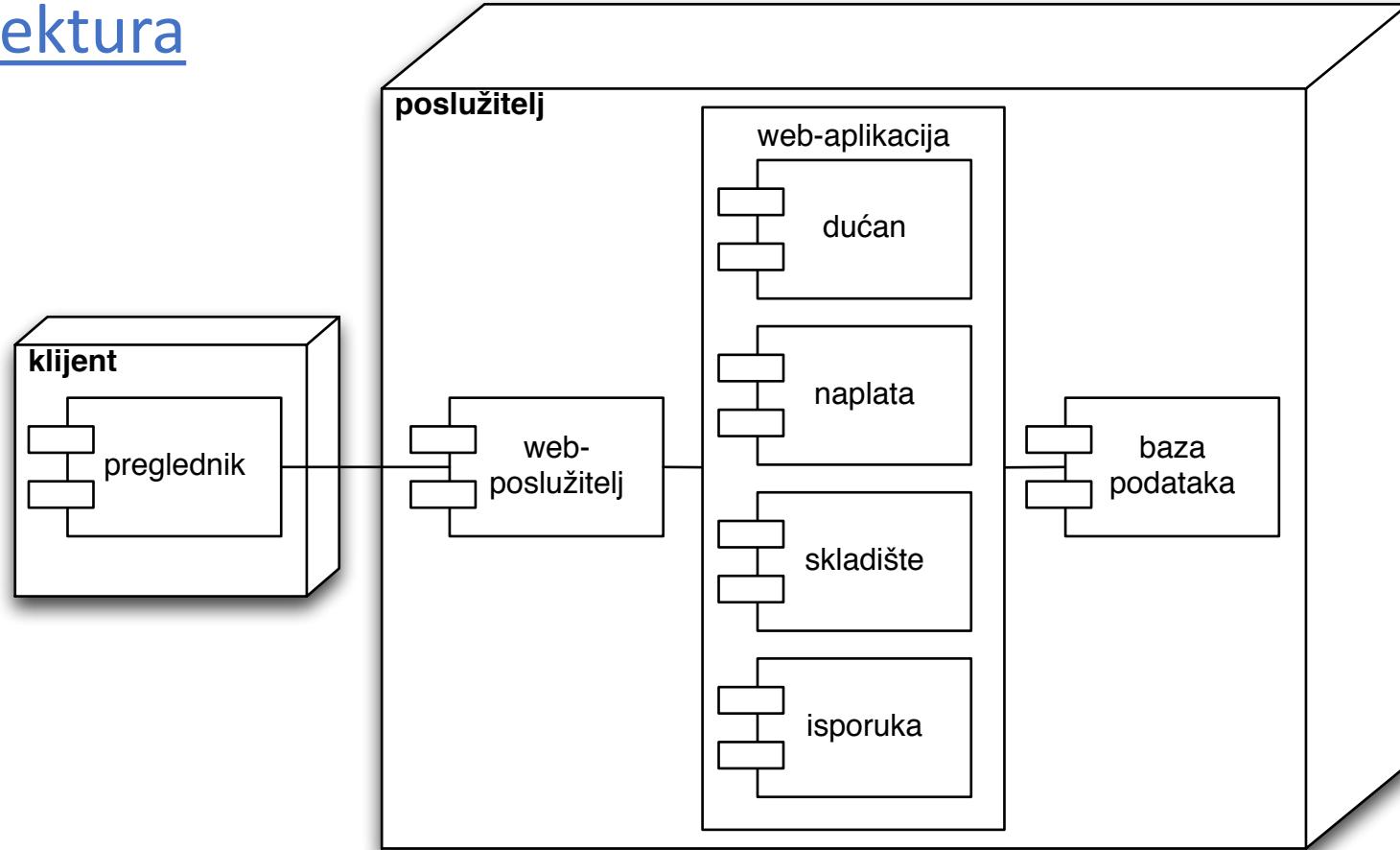
- Jedno računalo
- Jedna jezgra
- Skup RAM
- Skup disk
- Spora mreža
- Nekoliko konkurentnih korisnika
- Mala količina podataka
- Kašnjenje u sekundama
- Monolitne aplikacije
- Isporuka svakih nekoliko mjeseci

~2017

- Grozd računala
- Više jezgri
- Jeftin RAM
- Jeftin disk
- Brza mreža
- Puno konkurentnih korisnika
- Velika količina podataka
- Kašnjenje u milisekundama
- Pristup s mikrouslugama
- Isporuka X puta (10-100x) dnevno (DevOps)

# Tradicionalna monolitna aplikacija

- monolitna arhitektura



# Prednosti monolitne arhitekture

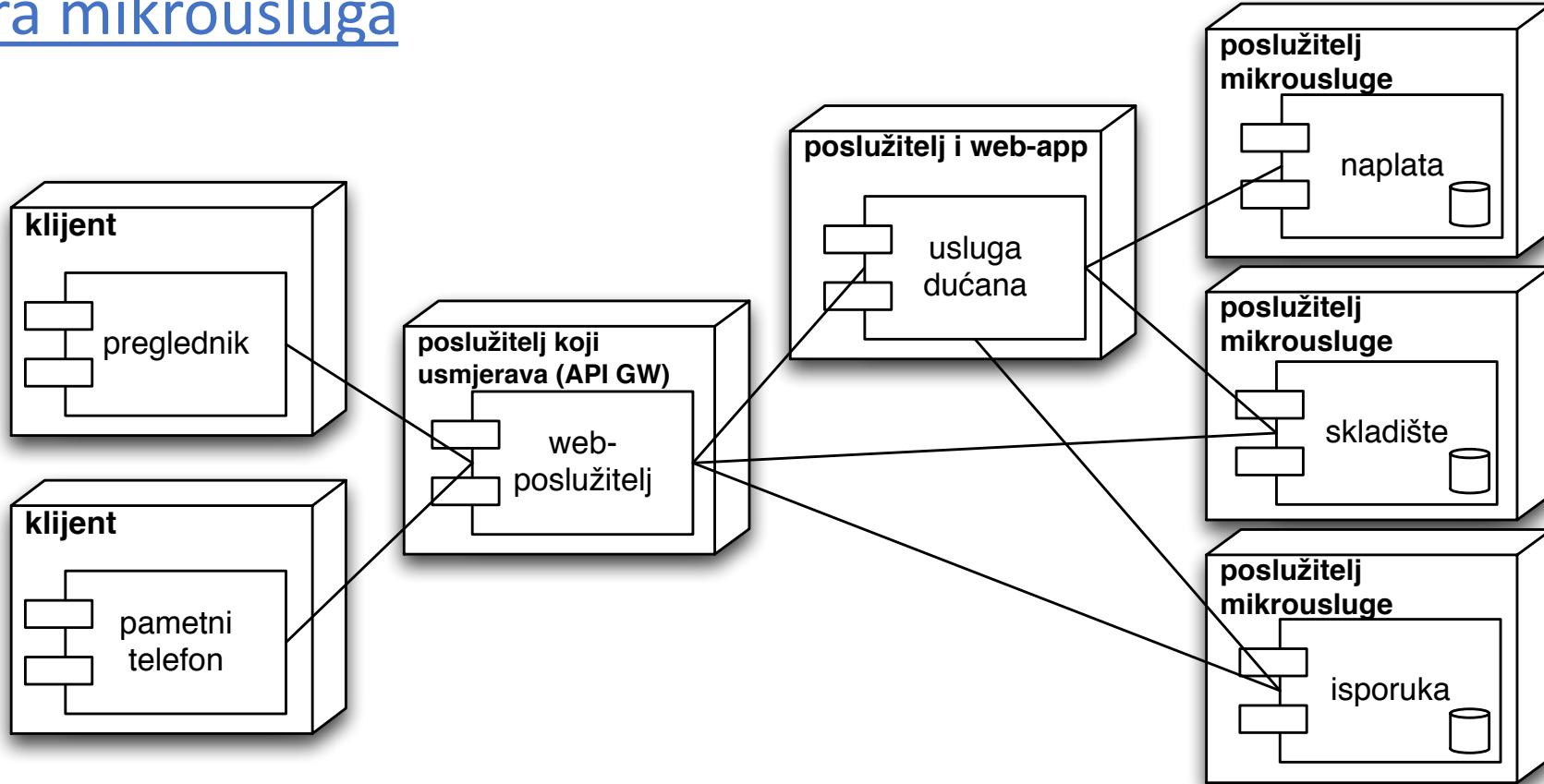
- Jednostavno koristiti druge module
  - Sve u jednom jeziku
  - Sve u jednom procesu pa je korištenje zapravo pozivanje funkcija/metoda što je brzo
- Jednostavno isporučiti – na jedan poslužitelj
- Jednostavno pratiti promjene u kodu (jedan izvorni kod)
- Jednostavno skalirati – jednu aplikaciju pokrenemo više puta
- Jezični konstrukti i radni okviri upravljuju kompleksnošću

# Nedostaci monolitne arhitekture

- Vezanost za programski jezik i programske okvire
  - Sve se radi u jednoj tehnologiji
  - Ne možemo koristiti nove pristupe za pojedine probleme
    - Npr. reaktivno programiranje, noSQL baze, ...
- Shvaćanje cijele aplikacije
  - Jedan razvijatelj ne može znati čitav kod
  - Jedan tim ne može upravljati čitavom aplikacijom
    - Svaki tim je za pojedini modul
  - Veliki timovi (stotinjak članova koji su zapravo puno malih timova)
- Isporuka kao jedna jedinica (npr. war datoteka)
  - Zbog male promjene moramo isporučiti čitavu aplikaciju
  - Jedna promjena može uzrokovati lanac promjena u čitavoj aplikaciji

# Mikrousluge

- arhitektura mikrousluga



# Prednosti mikrousluga

- jednostavno shvatiti jednu uslugu (kod i ulogu), ali ne i čitav sustav
- manji timovi za razvoj jedne usluge
- kontinuirana isporuka, brži razvojni ciklus/česte isporuke
- mogućnost korištenja novih/različitih tehnologija u različitim uslugama
  - baze, programski jezici, poslužitelji, ...
- fleksibilniji raspored usluga u sustavu
- raspodijeljeni podaci (različite baze podataka - SQL, noSQL)
- horizontalna skalabilnost – jednostavno pokrenuti nove instance pojedine usluge
- veća pouzdanost – ako imamo više instanci iste usluge
- moguća brza regeneracija sustava nakon pada performansi
- neovisnost o programskom jeziku, radnom okviru, infrastrukturi oblaka

# Nedostaci mikrousluga

- prednosti ne dolaze same po sebi
- kompleksnost prebačena sa aplikacije na integraciju
  - više održavanja
  - više znanja i vještina trebaju imati razvijatelji i održavatelji (*operations/administratori*)
  - teže testiranje čitavog sustava
- složenost raspodijeljenog sustava
  - usluge mogu biti nedostupne (greška u programu, računalu, mreži)
- asinkronost u komunikaciji je teško kombinirati
- treba više pratiti stanje sustava (automatsko promatranje)
  - potrebna je dodatna infrastruktura
- nema raspodijeljenih transakcija
  - upotrijebiti eventualnu konzistentnost
- razvijatelji moraju biti svjesni raspodijeljenosti sustava

# Mikrousluge – definicije eksperata

“...the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.”

James Lewis, Martin Fowler, ([poveznica](#))



**adrian cockcroft**  
@adrianco

Follow

@kellabyte @mamund I used to call what we did "fine grain SOA". So microservices is SOA with emphasis on small ephemeral components

([poveznica](#))

# Praktična definicija

- Aplikacija sastavljena od puno malih usluga
- Svaka usluga se izvršava u neovisnom procesu
  - Ne moraju biti na istom računalu
- Mikrousluge komuniciraju otvorenim protokolima
  - Npr. HTTP/REST, AMPQ, gRPC, Thrift, ...
- Svaka mikrousluga može biti napisana, isporučena, skalirana i održavana na drugačiji način
  - Npr. koriste se različiti programski jezici i različite baze podataka
- Svaka mikrousluga enkapsulira neki poslovni aspekt
  - Ne programske konstrukte (biblioteka, modul, paket, ...)
- Svaka mikrousluga je neovisna, zamjenjiva i može se samostalno nadograditi

# Što mikrousluge nisu?

- Nije isto što i SOA
  - SOA integrira poslovne (*enterprise*) aplikacije
  - Mikrousluge su namijenjene za dekompoziciju jedne aplikacije
- Nije najbolje rješenje za svaki problem
  - **Korištenje mikrousluga ima svoje nedostatke i rizike**

# Zablude programiranja u raspodijeljenoj okolini (čitaj oblak)

- Mreža je pouzdana
- Nema kašnjenja
- Propusnost je neograničena
- Mreža je sigurna
- Topologija se ne mijenja
- Cijena transporta ne postoji
- Mreža je homogena
- Postoji jedan administrator

# Komponentizacija u mikrousluge

- Melvyn Conway, 1967 (Conway's Law):
  - Organizacija dizajnira sustav koji je kopija komunikacije u toj organizaciji
- Komponente/mikrousluge nisu konstrukti jezika ili radnog okvira
- Izolacija između komponenti je vrlo bitna i utječe na dizajn
  - Pažljivo treba definirati sučelja (npr. REST) koje će druge usluge koristiti
    - Pristup *Consumer Driven Contracts*
  - Ako se sučelja ne mijenjaju
    - Svaka mikrousluga se može mijenjati kada je ona spremna
    - Promjena jedne mikrouslugе **ne smije** zahtijevati promjenu drugih mikrousluga
  - Ako se sučelja mijenjaju treba ih napraviti tako da promjene ne uzrokuju trenutnu promjenu kod drugih usluga
  - Omogućuje lakše: skaliranje, praćenje, *debugging*, neovisno testiranje pojedine usluge
- Kako podijeliti aplikaciju u usluge?
  - Primarni način je prema poslovnoj funkcionalnosti
  - Paziti da se poštuje princip jedne odgovornosti ([Single Responsibility Principle](#)) – slično kao unix alati
  - [Ograničeni kontekst](#)
- Obrasci: [prikupljeni obrasci](#), [najčešće korišteni obrasci](#)

# Komunikacija i upravljanje

- Protokoli: HTTP, TCP, UDP, poručivanje (*messaging*)
  - Sadržaj: JSON, BSON, XML, Protocol Buffers (gRPC), Thrift, ...
- Poručivanje: koristimo događaje, gledamo u povijest, drugačiji pristup
- Tjera nas na definiranje jasnih sučelja
- Komuniciranje ide preko API-a, a ne preko baze podataka
- Treba koristiti najbolji alat za problem koji se rješava (programski jezik, radni okvir, baza podataka)
- Mikrousluge se mijenjaju i isporučuju različitim brzinama
- Mikrousluge mogu mijenjati IP adresu (kod skaliranja ili kod restarta) – potrebna je infrastruktura za otkrivanje usluga

# Problemi

- Što ako je takvih usluga 100?
  - Kako znamo na kojim je vratima (*port*) koja usluga?
  - Kako znamo da je svaka usluga pokrenuta?
  - Što ako trebamo promijeniti konfiguraciju nekih usluga?
  - Što ako neka usluga ne radi (praćenje performansi i grešaka - *monitoring*)?
  - Što ako je neka usluga preopterećena pa ne odgovara na vrijeme, nego sa zakašnjnjem koje nije prihvatljivo?
  - Kako pratiti kompleksnost sustava?
  - Kako znati koji *build*, koja verzija sw-a je na kojem čvoru (*traceability*)?
  - Kako doći do dnevničkih zapisa (raspodijeljeni *logging*)?
- Potrebno je veliko znanje u organizaciji (ulaganje u ljude)
- Svi testovi moraju biti automatizirani (DevOps)

# Aplikacije s 12 faktora

1. Svaka mikrousluga ima jedan repozitorij (npr. git) za kod i iz njega se sve prati i stvaraju izvršne verzije
2. Ovisnosti - eksplisitno definirane ovisnosti o drugima
3. Konfiguracija – u okolini, ne u kod
4. Podržavajuće usluge (*backing service*) – tretirati ih kao resurse koji se mogu konfigurirati
5. Izgradnja, objava, pokretanje – odvojiti faze i definirati pravila za izdavanje
6. Usluge se izvršavaju kao procesi bez stanja
7. Objavljuvanje usluga preko definiranih vrata (*port*)
8. Skaliranje pomoću pokretanja novih procesa
9. Povećanje robusnosti pomoću brzog pokretanja i pristojnog gašenja
10. Paritet između razvojne i produkcijske okoline
11. Tretirati zabilješke (*log*) kao tok događaja
12. Administratorski procesi bi se trebali izvršavati u istoj okolini kao i produkcijski

# Rješenja nekih problema (1)

- konfiguriranje
  - jednostavna primjena
  - jednostavno vraćanje na prethodnu konfiguraciju u slučaju grešaka
- lokacijska transparentnost
  - kada se neka usluga ugasi i druge pokrene na novoj IP adresi i/ili vratima (*port*) usluga koja ju koristi bi ju automatski trebala početi koristiti
- praćenje u radu
  - metrike (opterećenje, memorija, broj zahtjeva, kašnjenje)
    - spremnost za primanje zahtjeva
    - je li usluga pokrenuta ili se srušila
  - *tracing* – praćenje jednog zahtjeva kroz usluge (korelacijski identifikator)
  - dnevnički zapisi (*log*) – skupljanje na jednom mjestu i analiza
  - alarmiranje u slučaju nepredviđenog ponašanja

# Rješenja nekih problema (2)

- otpornost na kvarove
  - kada neka usluga prestane raditi ili ima veliko kašnjenje usluga koja ju koristi to treba detektirati i imati alternativu
    - obrazac osigurača
    - označavanje usluge neispravnom i izbaciti ju iz uravnoteženja opterećenja
    - kada se obnovi potrebno je automatsko uključivanje usluge u sustav
  - verzioniranje
    - potrebno je imati informacije koja verzija usluge se nalazi na kojem resursu kako bi se kod detekcije kvara u usluzi znalo koju verziju treba popraviti
  - skaliranje
    - povećanje/smanjenje broja instanici
    - želimo automatsko
    - skaliranje do 0 instanci
    - problem skaliranja usluga sa stanjem
  - [Chaos engineering \(npr. Netflix Simian Army\)](#)

# Konfiguracije

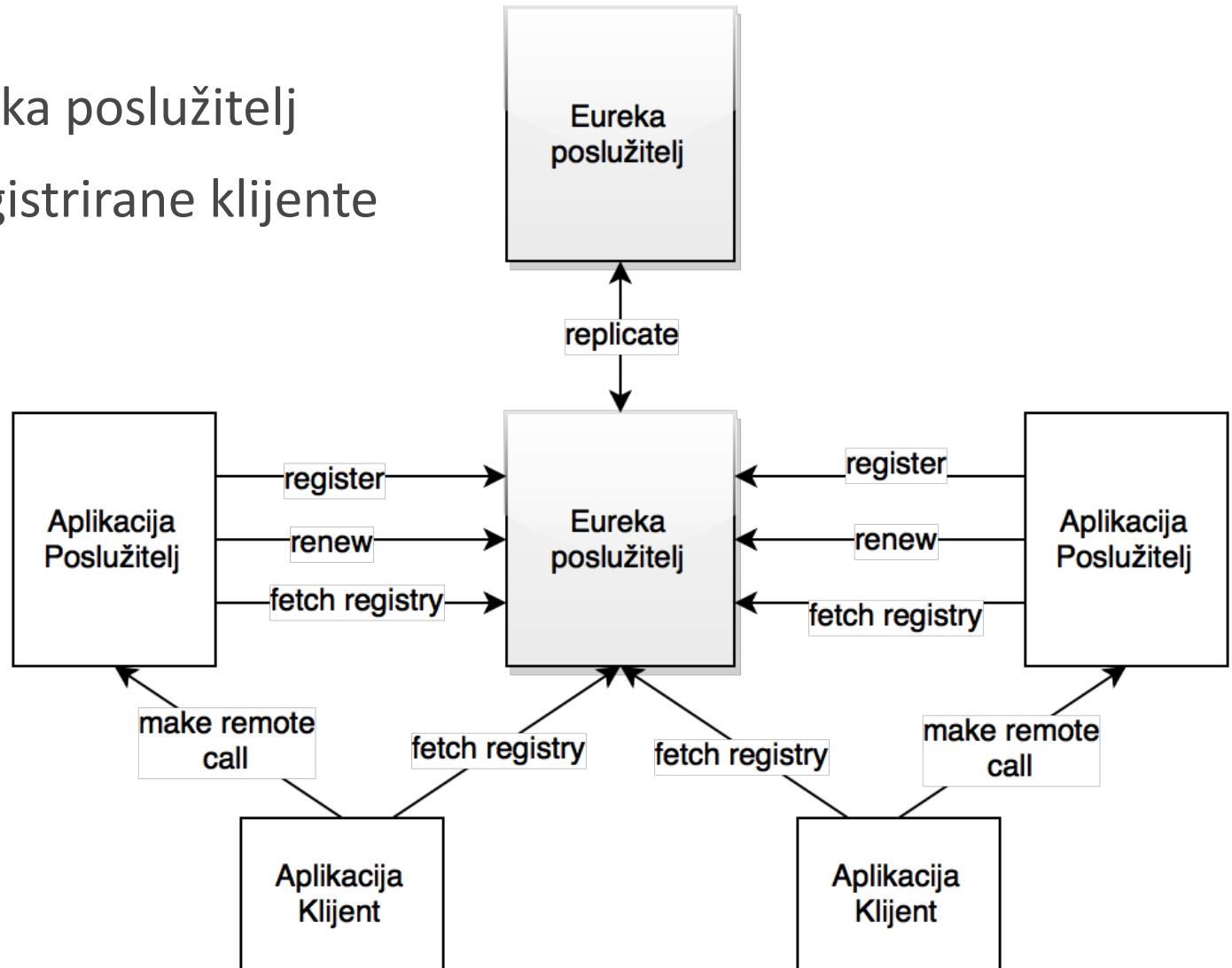
- Svaka aplikacija ima konfiguraciju
  - Npr. na koju bazu podataka se spaja, gdje su resursi, na kojim vratima (*port*) sluša, ...
- Konfiguracija nije u kodu, nego u nekoj datoteci kako bi se lakše moglo instalirati u drugoj okolini
- Gdje se konfiguracija može nalaziti:
  - U arhivi koja se isporučuje – ponovna isporuka u slučaju promjene
  - Konfiguracijske datoteke – na poznatom mjestu na disku (teško je osigurati takvo mjesto u okolini oblaka npr. AWS)
  - Variable okoline – različito na različitim OS-ovima, teško upravljati kada imamo puno podataka
  - Svaki oblak ima svoja rješenja za to – vezani smo za taj oblak
- Kubernetes ima mehanizam dijeljenja varijabli okoline
- Spring: Spring Cloud Config, Spring Cloud Bus (knjižnice)

# Otkrivanje usluga

- Otkrivanje s poslužiteljske strane
  - Komponenta za uravnoteženje opterećenja se brine o otkrivanju gdje se nalazi neka usluga
- Otkrivanje s klijentske strane
  - Klijent se brine o otkrivanju gdje je neka usluga
- Primjeri registara usluga:
  - etcd, consul, Apache Zookeeper, Netflix Eureka
- Spring Cloud Netflix Eureka
  - Aplikacije se same registriraju
  - Aplikacije mogu pretraživati registar
- Kubernetes – koristi DNS i uravnoteženje opterećenja
- Istio – koristi *proxy*

# Eureka

- Mikrousluga se registrira na Eureka poslužitelj
- Svaka usluga može pregledati registrirane klijente



# Uravnoteženje opterećenja

- Spring Cloud Netflix Ribbon (knjižnica)
  - Uravnoteženje opterećenja na klijentu
  - Automatski se integrira s Eurekom (kada je uključena u projekt)
    - Automatski dohvaća konkretnu IP adresu i vrata registrirane usluge
  - Može privremeno spremati rezultate (*caching*)
  - Podržava više protokola (HTTP, TCP, UDP)
  - Postoji više strategija (podrazumijevano *round-robin*)
- Kubernetes:
  - Posebna komponenta u grozdu koja radi uravnoteženje ravnomjerno s obzirom na broj instanci
- Istio:
  - koristi *proxy* kod svake usluge
  - puno finije mogućnosti usmjeravanja prometa

# Metrika mikrousluge

- zahtjev/sec., memorija, procesor, vrijeme odgovora, probe (*liveness, readiness*)
- Spring Boot Actuator (knjižnica)
  - usluga ima URI za različite metrike gdje se to može dohvaćati
- Kubernetes
  - koristi metrike od usluga
- Istio
  - proxy prikuplja podatke
- Tehnologije:
  - ELK stack
    - Elasticsearch - pretraživanje i analitika nad podacima
    - Logstash - prikuplja podatke o metriči sa različitih usluga
    - Kibana - vizualizacija podataka (grafovi, pite, ...)
  - Prometheus - sustav za praćenje i alarmiranje
  - Grafana - vizualizacija podataka

# Zašto je otpornost na kvarove bitna?

- Kada imamo veliki broj mikrousluga neka od njih neće biti dostupna zbog kvarova:
  - Računalo, mreža, preopterećenje, ...
- Zbog toga što jedna usluga poziva drugu velika mogućnost je da postoji kaskada usluga koje ne rade
  - Prepostavimo da sustav ima pouzdanost 99,95% - [Amazon EC2 Service Level Agreement](#)
  - Npr. [Netflix ima 13,000 kontejnera u produkciji \(2016.\)](#)
  - Monolitna aplikacija: ne radi 21,6 minuta mjesечно
  - Sustav mikrosuluga koji su povezane:
    - 10 povezanih mikrousluga: ne radi 3,59 sati mjesечно
    - 50 povezanih mikrousluga: ne radi 17,78 sati mjesечно
    - 100 povezanih usluga: ne radi 35,12 sati mjesечно

# Otpornost na kvarove

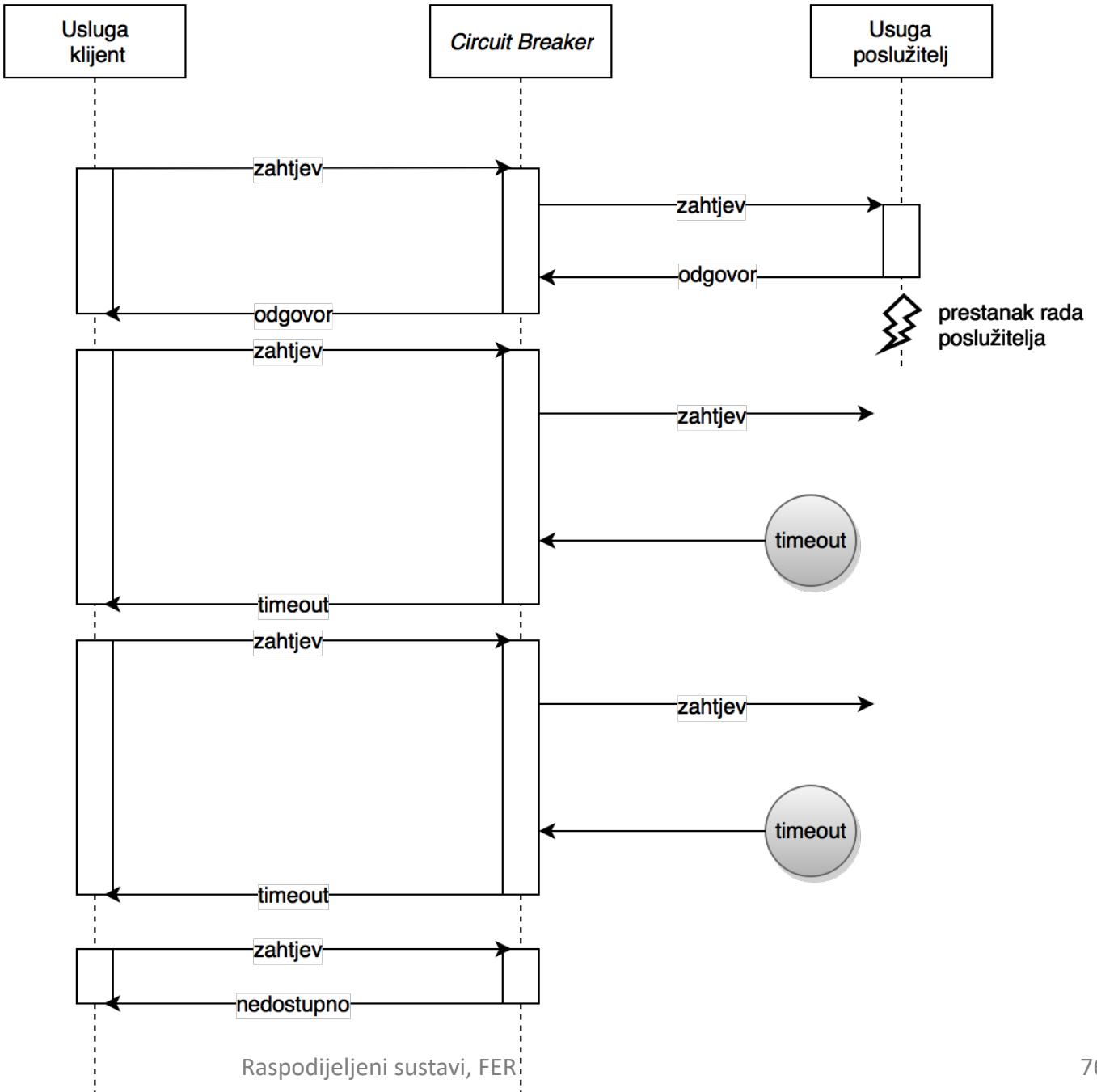
- Redundancija usluga
  - Više usluga može istovremeno raditi isti posao
- Vruća redundancija:
  - Na dva poslužitelja pristižu zahtjevi i oba rade isti posao, ali je jedan glavni. Ako glavni prestane raditi pomoći postaje glavni.
  - Nakon ponovnog uključivanja u rad potrebna je sinkronizacija podataka (ako se podaci spremaju)
- Princip osigurača:
  - Zahtjevi se uravnotežuju i jedna usluga prestane raditi
  - Nakon toga se zahtjevi ne šalju na tu uslugu
  - Npr. Hystrix
- Hijerarhija supervizora
  - Kada neka usluga ne radi supervizor ju gasi i ponovno pokreće
  - Postoji hijerarhija supervizora (npr. Erlang OTP ili Akka Framework)

# Izolacija neispravne usluge

- Kada neki dio ne radi treba ga što prije izolirati i ne slati zahtjeve na taj dio
- Na taj način se sprječavaju kaskadni ispadi
- **Hystrix** (knjižnica) je implementacija mehanizma osigurača između usluga
  - Detektira kvarove (iznimka ili kašnjenje) i izolira nedostupnu uslugu
  - Kvar se detektira statistički – npr. 20 kvarova u 5 sekundi
  - Omogućuje preusmjeravanje na pričuvnu uslugu dok se kvar ne popravi
  - Automatsko preusmjeravanje na popravljenu uslugu kada se vrati u funkciju
    - Nakon definiranog intervala provjere
- **Kubernetes:**
  - točke koje koristi za provjeru ispravnosti usluge povezuje s uravnoteženjem opterećenja
- **Istio**
  - *proxy* detektira ispade i dojavljuje komponenti za konfiguracije koja to proslijedi ostalima

# Hystrix

- implementira obrazac *circuit breaker*
- služi za povećanje otpornosti sustava na greške



# Strategije uvođenja novih verzija usluga

- *Recreate*
  - ugasiti staru verziju i pokrenuti novu
- *Ramped*
  - kada imamo više replika onda dodamo jednu novu verziju, pa ugasimo staru, i tako dok sve ne zamijenimo (Kubernetes)
- *Blue/Green*
  - instaliramo novu verziju (*blue*) i testiramo da radi, nakon toga samo sav promet preusmjerimo na novu verziju (*switch*)
- *Canary*
  - slično Blue/Green samo postepeno postotak sa stare verzije prebacujemo na novu (Istio)
- *A/B Testing*
  - samo neke korisnike prebacimo na novu verziju, primjeri kriterija: beta testeri, geografsko područje, korisnici pojedinog preglednika, jezik, OS, veličina ekrana, ... (Facebook, Istio)
- *Shadow (Dark Launch)*
  - instaliramo obje verzije i zahtjeve šaljemo na obje, ali samo od stare se odgovori vraćaju korisniku (testiranje u produkciji)

# Skaliranje usluge bez podataka

- Koriste se mehanizmi kao kod kvarova:
  - Redundancija usluga, usmjeravanje zahtjeva, dinamičko otkrivanje usluga, supervizor
- Bitna je lokacijska transparentnost
  - Kada se pokrene nova usluga da ju je lako otkriti
- Novi trend – Serverless Architecture - *Functions as a service*
  - Primjeri: [AWS Lambda](#), [Google Cloud Functions](#), [Azure Functions](#), [Project fn](#) (otvoreni kod), [Knative](#) (dodatak za K8S)
  - Ideja:
    - Kada pristigne zahtjev onda se pokreće usluga, proslijeđuje joj se zahtjev i kada ga obradi gasi se
  - Bitno svojstvo: brzo pokretanje
    - u desetinama sekundi, a ne u minutama

# Veličina i brzina pokretanja usluga

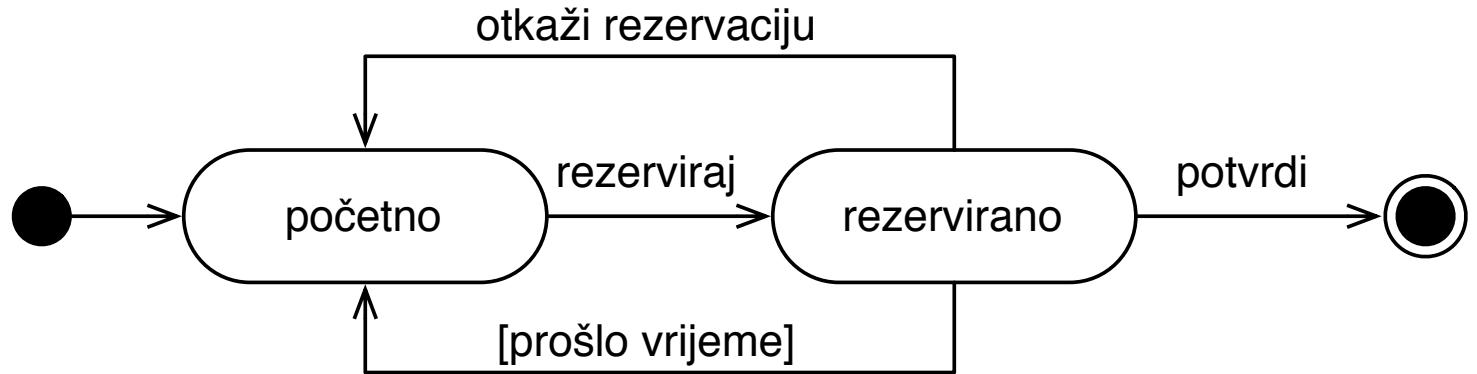
- Potrošnja memorije i brzina pokretanja aplikacija:
  - Node: 92MB, 18s
  - Quarkus: 82MB, 16s
  - Go: memorija slična Quarkusu, 19s
  - SpringBoot: 150-200MB, ~1min
- Quarkus - Supersonic Subatomic Java
- Izvori:
  - [Supersonic, Subatomic Java with Quarkus. Burr Sutter, Red Hat](#)
  - [Kubernetes Native Spring apps on Quarkus by Georgios Andrianakis](#)

# Skaliranje usluge s podacima

- Problem je kako osigurati podatke različitiminstancama usluge.
- Pristupi:
  - Replikacija podataka (*cache*)
  - Raspodijeljene baze podataka:
    - Segmentacija podataka (tablice na različitim računalima, podjela tablice na segmente)
    - Raspodijeljene transakcije (protokol dvofaznog izvršavanja – 2PC, standard Open XA)
  - P2P sustavi – npr. mehanizam Chord
  - Bilježenje događaja (*Event Logging*)
  - Raspodijeljena Saga

# Transakcije

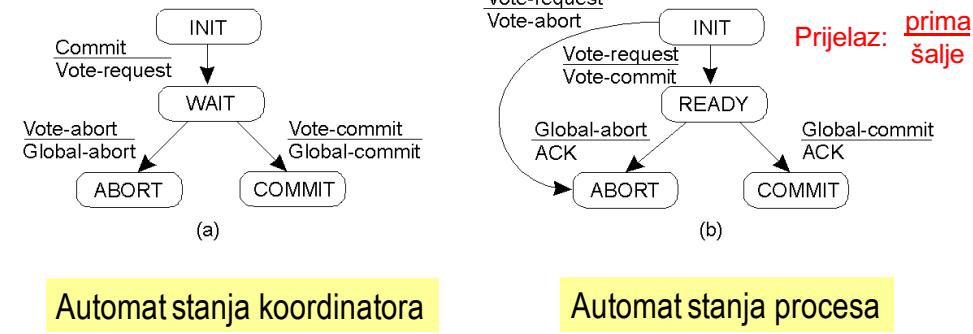
- Model transakcije
- Dva stanja:
  - Početno
  - Rezervirano



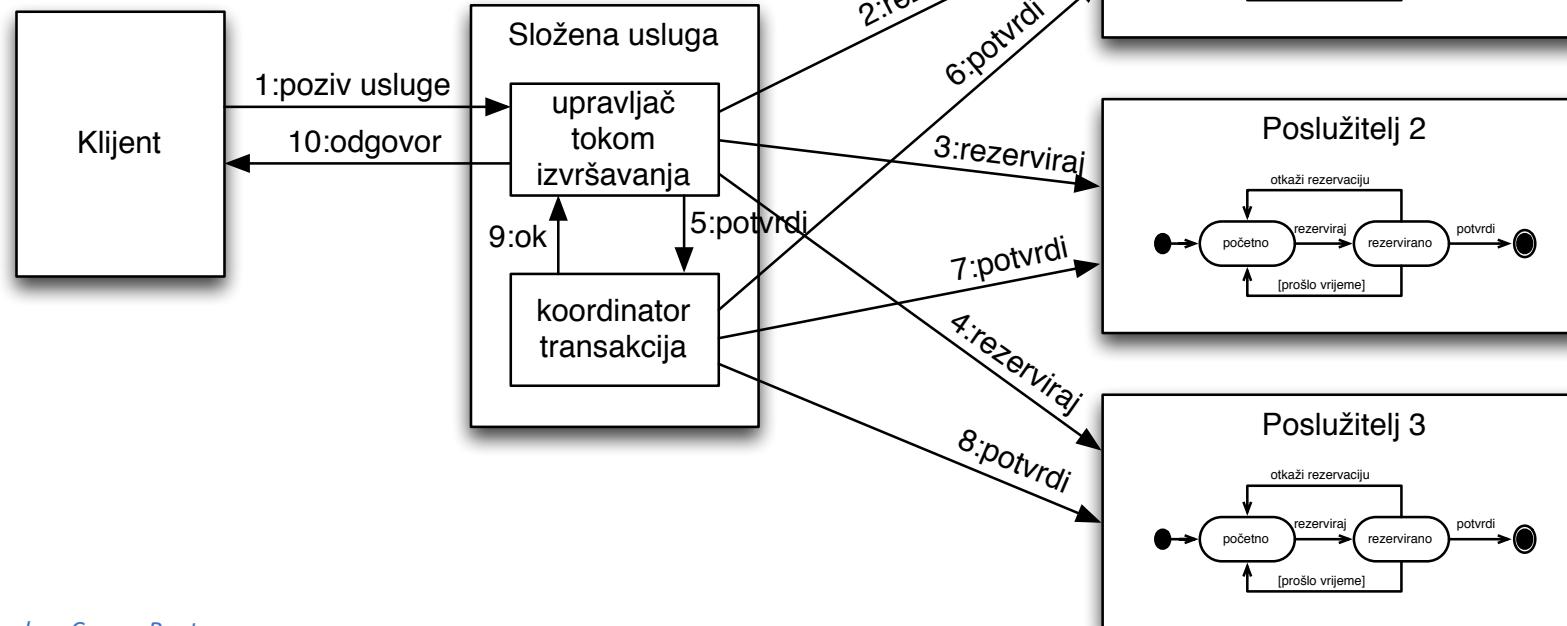
# ACID (izvor: kolegij Baze podataka)

- **Atomicity** – nedjeljivost transakcije (atomarnost) - transakcija se mora obaviti u cijelosti ili se uopće ne smije obaviti
- **Consistency** - konzistentnost- transakcijom baza podataka prelazi iz jednog konzistentnog stanja u drugo konzistentno stanje
- **Isolation** - izolacija - kada se paralelno obavljaju dvije ili više transakcija, njihov učinak mora biti jednak kao da su se obavljale jedna iza druge
- **Durability** - izdržljivost - ako je transakcija obavila svoj posao, njezini efekti ne smiju biti izgubljeni ako se dogodi kvar sustava, čak i u situaciji kada se kvar desi neposredno nakon završetka transakcije

# Raspodijeljene transakcije



- Primjena 2PC-a
  - $O(n^2)$  poruka
  - Koordinator točka ispada
  - Reducirana propusnost



Guy Pardon Cesare Pautasso  
REST: From Research to Practice

# ACID 2.0

- **Associative** – asocijativnost – grupiranje poruka ne utječe na rezultat (mat.  $a(bc) = (ab)a$ ), a omogućuje obradu naknadnu obradu u pozadini
  - **Commutative** – komutativnost – redoslijed poruka ne utječe na rezultat (mat.  $ab = ba$ )
  - **Idempotent** – idempotentnost – dupliciranje poruka ne utječu na rezultat
  - **Distributed** – raspodijeljenost – poslovi se mogu izvršavati raspodijeljeno
- 
- Primjena kod oblikovnog obrasca Saga
  - Primjer: CRDT (*Conflict-free Replicated Data Types*), ...

Pat Helland, Dave Campbell: “[Building on Quicksand](#)”, CIDR 2009

Joy BanerjeeShail Aditya Gupta: Method and system for event state management in stream processing, [patent US9680893B2](#)

VICTOR GRISHCHENKO: CID 2.0 - Associative, Commutative, Idempotent, Distributed, [poveznica](#)

# Alternativa transakcijama - model BASE (usluge/baze)

- ***Basically Available*** – ne garantira se teorem CAP, svaki zahtjev će dobiti odgovor, ali odgovor može biti “greška”. Slično kao u banci kada se čeka potvrda prijenosa sredstava.
- ***Soft state*** – stanje sustava se može promijeniti kroz vrijeme i u slučaju kada nema zahtjeva. To nazivamo “meko” stanje.
- ***Eventual consistency*** – Sustav će nakon nekog vremena nakon zadnjeg zahtjeva doći u konzistentno stanje (eventualna konzistentnost).
- Primjena kod: *Event Sourcing* i CQRS-a

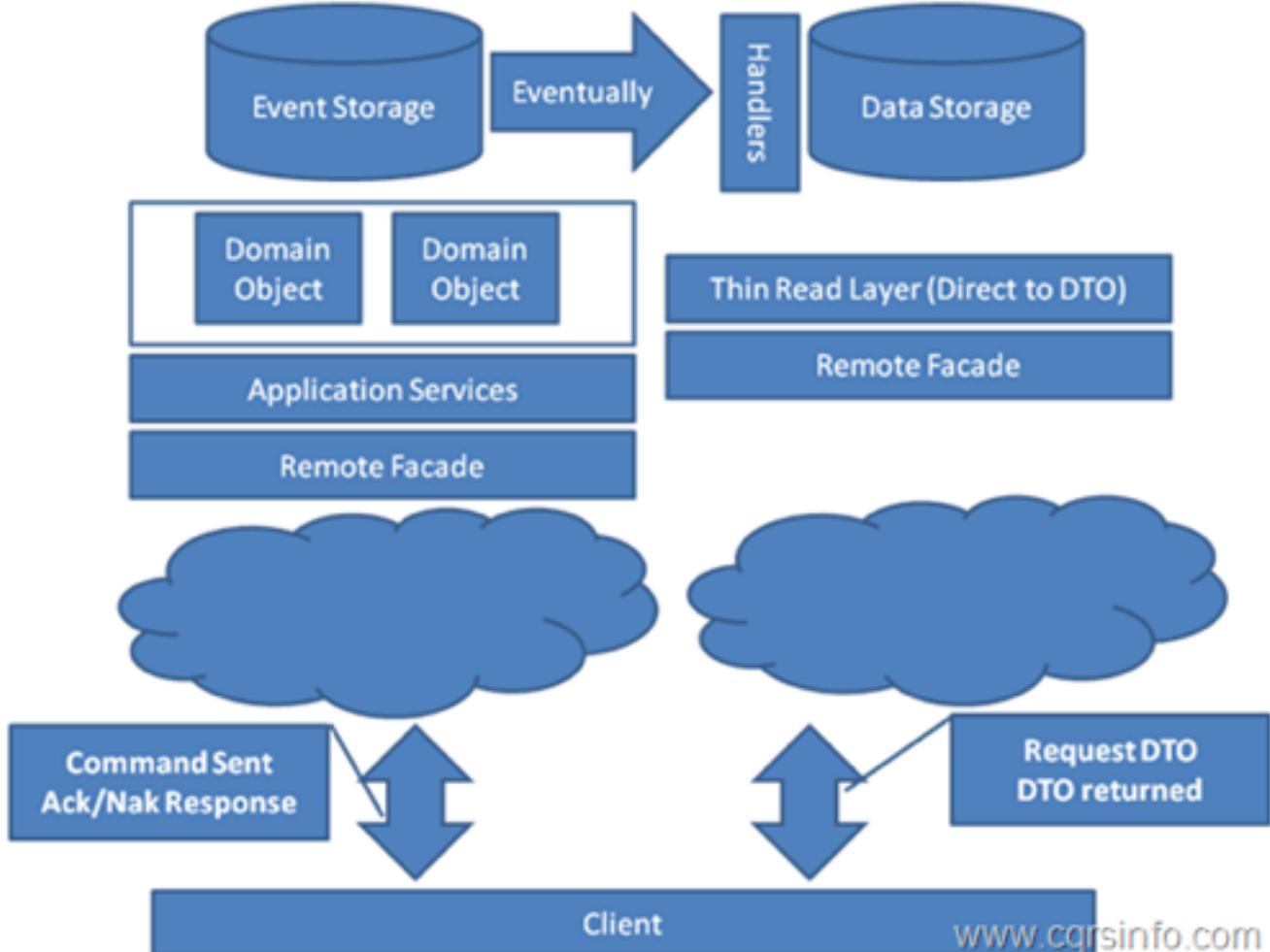
Dan Pritchett: "[Base: An Acid Alternative](#)", Queue, vol. 6, br. 3, str. 48-55

DekaGanesh Chandra: "BASE analysis of NoSQL database", Future Generation Computer Systems, Volume 52, November 2015, Pages 13-21

# Alternativa transakcijama

- *Bilježenje događaja (Event Logging)*
  - spremanje svih događaja u sustavu
  - ne spremi se stanje
  - nema brisanja
- *Event Sourcing*
  - spremjeni događaji se koriste za izračunavanje stanja sustava u nekom vremenu
  - događaji su izvor istine u sustavu
- Arhitekturni obrazac CQRS - *Command Query Responsibility Segregation*
  - autor: Greg Young
  - komande - stvaranje, promjena, brisanje
  - upiti - vraća pripremljeni pogled ili se pretplati na događaje
- Arhitekturni obrazac Saga

# CQRS i Event Sourcing



<https://cqrss.info/documents/cqrs-and-event-sourcing-synergy/>

# Bilježenje događaja (Event Logging)

- U log se bilježe svi događaji
  - Nema brisanja
- Događaji nisu stanja, već zahtjevi ili komande
  - Predstavlja činjenicu koja se već dogodila u prošlosti
- Takav log se koristi kao jedina ispravna istina
  - Primjena obrasca *Event Sourcing*
  - Omogućuje agregiranje događaja u sažetke ili stanje u pojedinom trenutku (obično u memoriji)
- Omogućuje efikasno korištenje obrasca CQRS (*Command Query Responsibility Segregation*) – autor Greg Young
  - Odvajanje model čitanje i pisanja
    - Komande - stvaranje, promjena, brisanje
    - Upiti - vraća pripremljeni pogled ili se pretplati na događaje

# Oblikovni obrazac: Saga

- Primjena u dugim transakcijama u bazama podataka (ne raspodijeljenim)
- Smanjuje vrijeme zadržavanja ključa (kod mehanizma zaključavanja)
- Definicija:
  - Saga je transakcija koja dugo traje, a može se zapisati kao niz transakcija kojima nije bitan redoslijed.
  - Sve transakcije ili:
    - uspješno završe i tada je Saga uspješno završila ili
    - ako jedna ne završi uspješno onda postoje kompenzirajuće transakcije koje se moraju izvršiti za svaku uspješnu transakciju i tako vratiti sustav u prijašnje stanje. Tada je saga neuspješna.
- Ograničenja:
  - Transakcije ne ovise jedna o drugoj tj. ne smije postojati redoslijed.
  - Svaka transakcija u Sagi mora imati kompenzirajuću transakciju koja semantički radi poništavanje transakcije.
- Nema atomarnosti, ali dobijemo uporabljivost sustava (nije zaključan)

Hector Garcia-Molrna, Kenneth Salem: "Sagas", 1987.

# Saga na jednoj bazi

- Postoji komponenta SEC (Saga Execution Coordinator)
  - Nalazi se unutar procesa baze podataka
  - Upravlja Sagom
- Imamo Saga Log sa sljedećim porukama:
  - Akcije: početak (B), kraj (E), prekid (A)
  - Akcije nad: saga (S), transakcija (T), kompenzirajuća transakcija(C)
- Primjeri oznaka:
  - BT2 – početak transakcije 2
  - EC1 – kraj kompenzirajuće transakcije 1
  - AS – prekid sage

# Uspješna saga

- Npr. u sagi imamo 3 transakcije: T1, T2, T3
- Log uspješne sage:
  - BS
  - BT1
  - ET1
  - BT2
  - ET2
  - BT3
  - ET3
  - ES

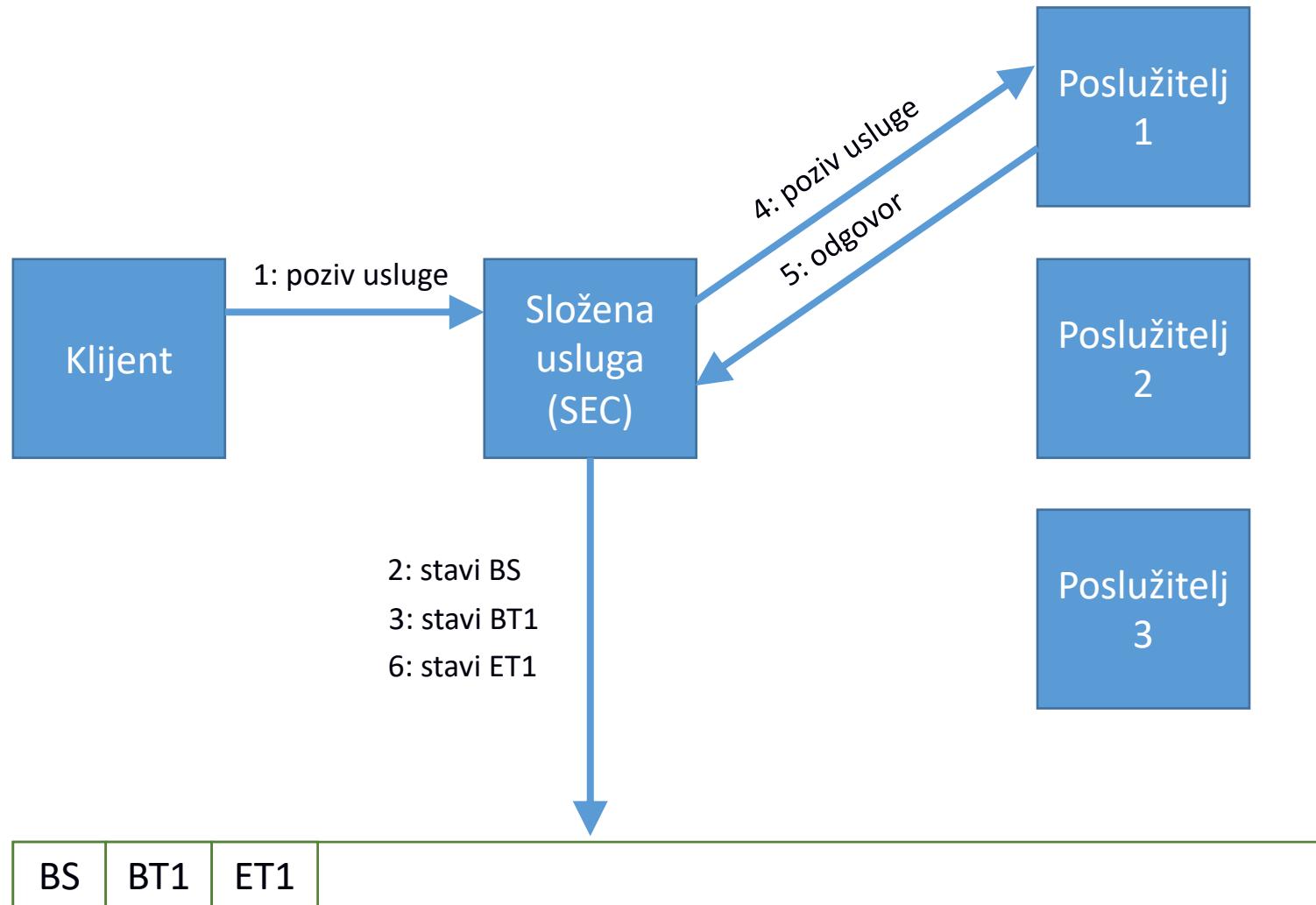
# Neuspješna saga

- Nekoliko načina oporavka:
  - Unatrag, unaprijed, ...
- Ovdje je oporavak unatrag (najčešće se koristi)
- Log neuspješne sage:
  - BS
  - BT1
  - ET1
  - BT2
  - **AS** – transakcija T2 se nije mogla izvršiti
  - **BC1**
  - **EC1**
  - **ES**

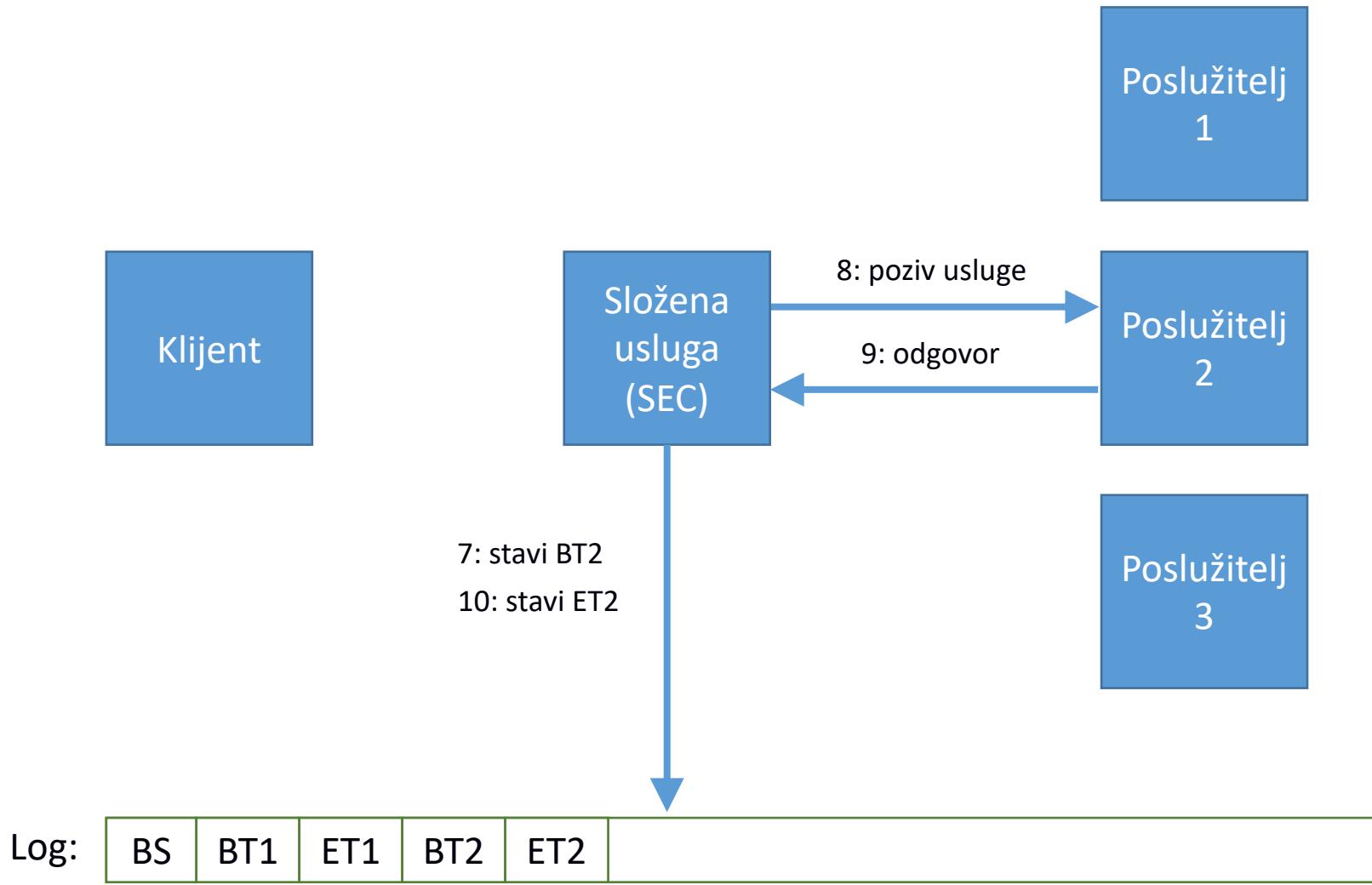
# Raspodijeljena saga

- Semantika je malo drugačija jer nemamo transakcije, nego zahtjeve
  - Obrada zahtjeva može i ne mora podržavati ACID. To ovisi o usluzi koju koristimo.
- Saga Log mora biti raspodijeljen i izdržljiv: npr. RabbitMQ, Kafka, ESB, Blok lanac, ...
  - Zahtjev se treba isporučivati najviše jednom
  - Kompenzirajući zahtjev se mora isporučiti najmanje jednom
- SEC (*Saga Execution Coordinator*) je poseban proces
  - Upravlja sagom
  - Ne pamti stanje, to je u logu
  - Može se pokvariti u bilo kojem trenutku
- Kompenzirajući zahtjevi moraju biti idempotentni i moraju se moći izvršiti.
- Nema atomarnosti niti izolacije

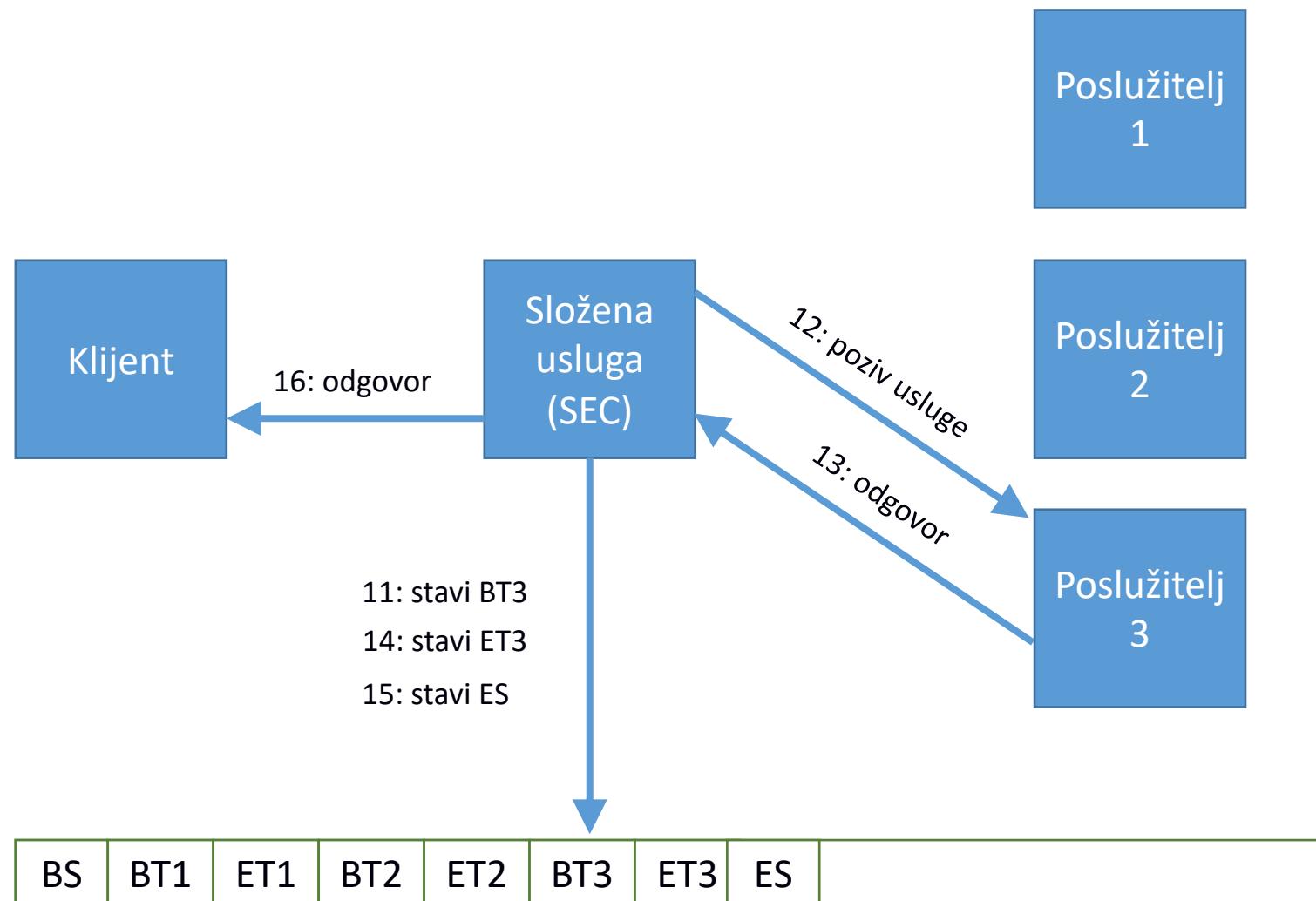
# Uspješna raspodijeljena saga (1)



# Uspješna raspodijeljena saga (2)



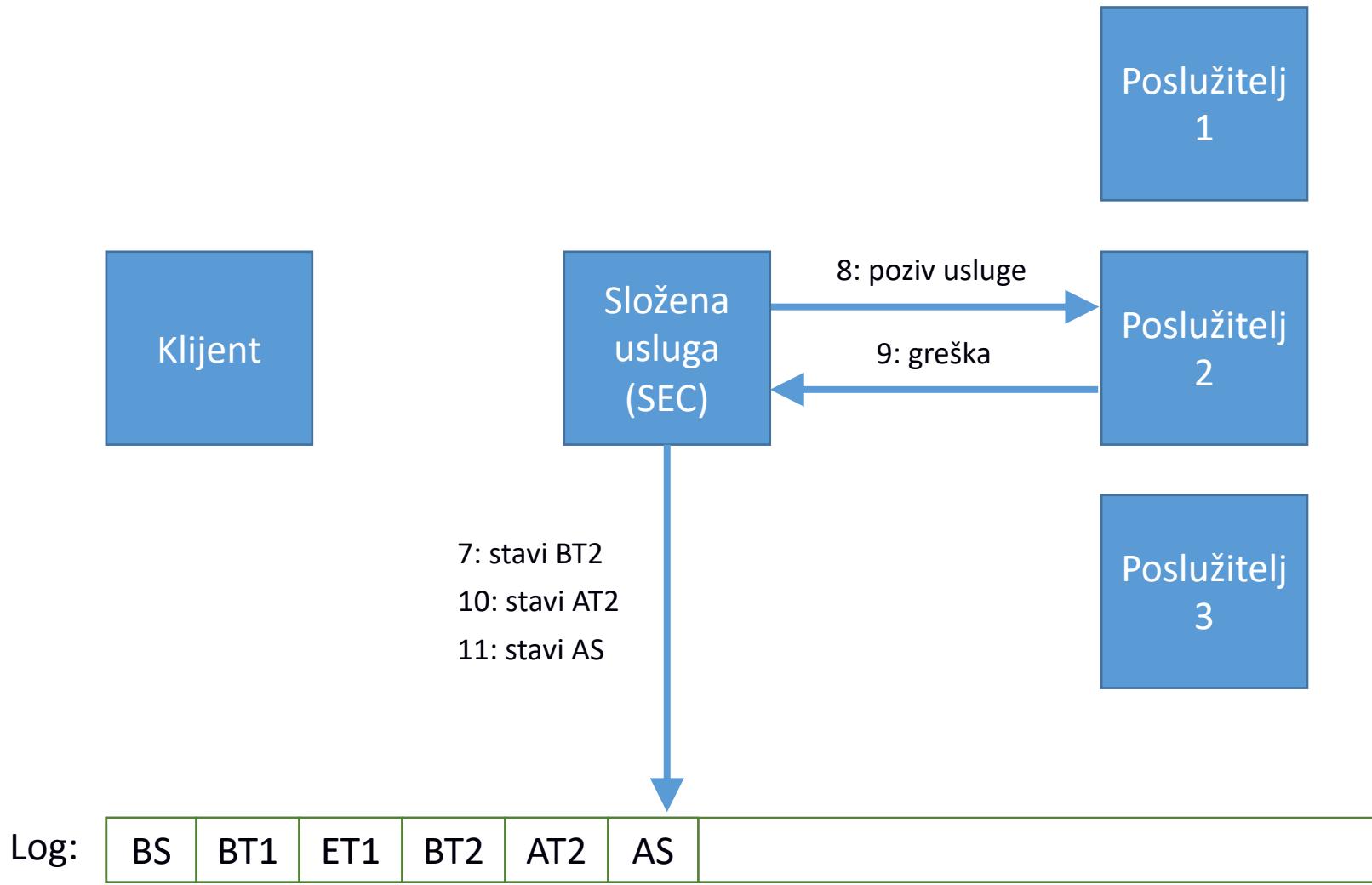
# Uspješna raspodijeljena saga (3)



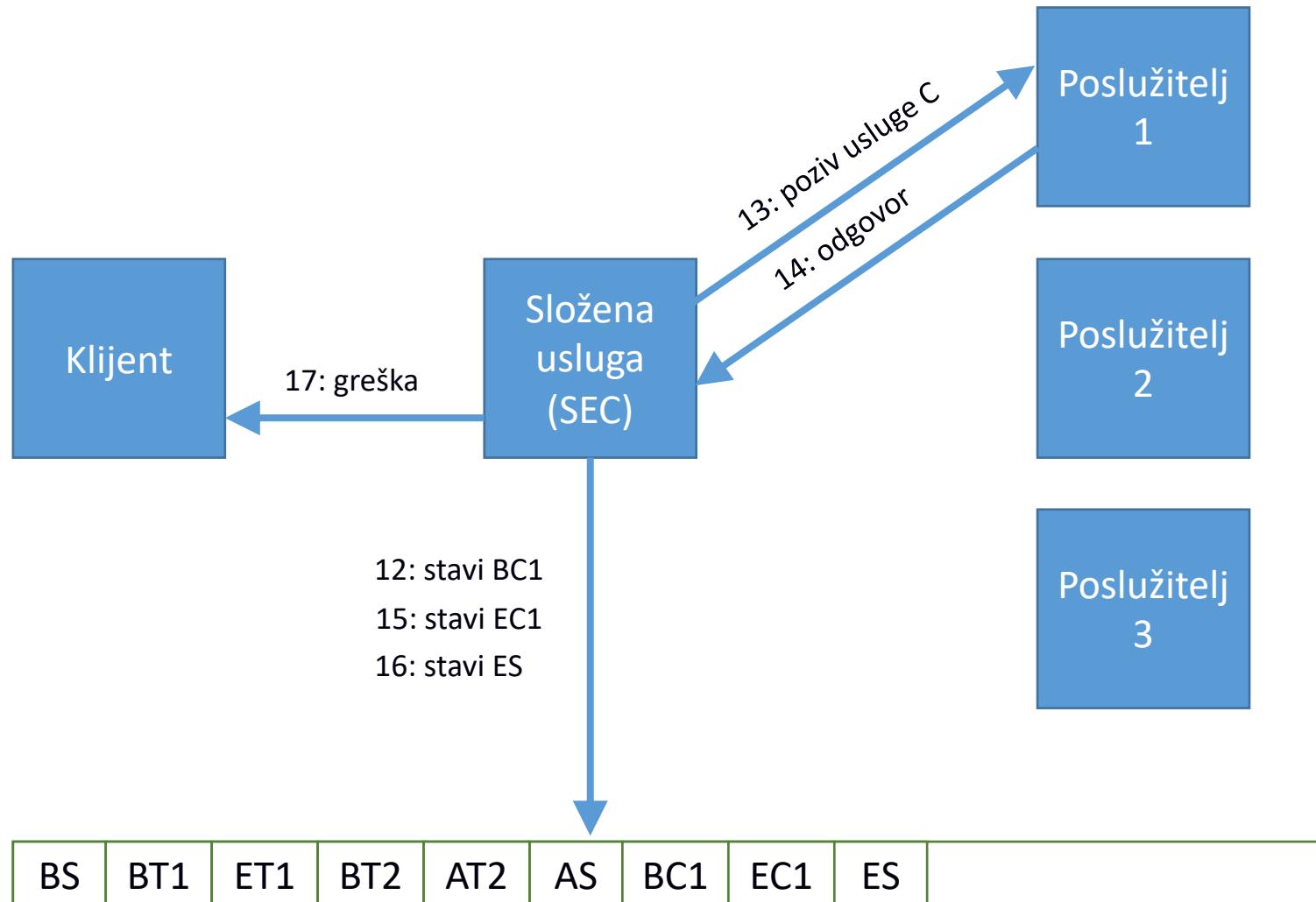
# Prekidi sage

- Korisnik prekida sagu
- Poslužitelj ne može izvršiti upit
- Poslužitelj se sruši
- SEC se sruši (nesigurno stanje)

# Poslužitelj 2 ne može izvršiti upit (1)



# Poslužitelj 2 ne može izvršiti upit (2)



# Što kada kompenzirajući zahtjev ne prođe?

- Zato kompenzirajući zahtjev mora biti idempotentan
- Ponavlja se kompenzirajući zahtjev dok se ne izvrši
- Ako je sistemska greška pa nikad kompenzirajući zahtjev neće proći treba ugraditi mehanizam vremenskog ograničenja.

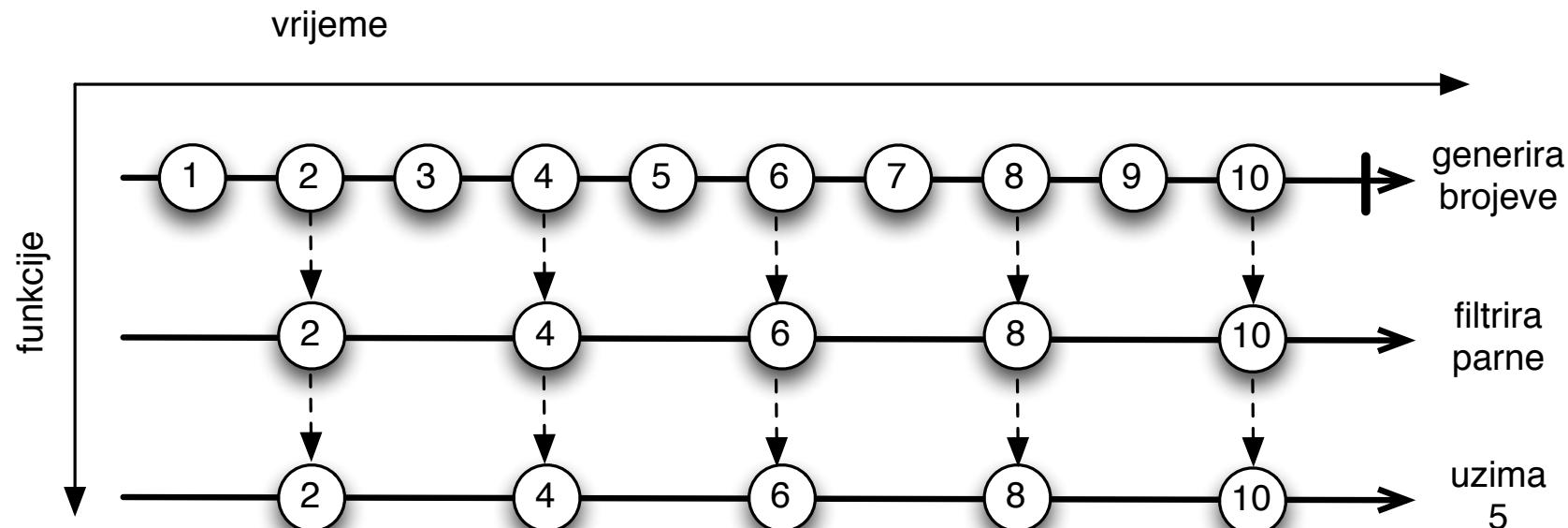
# Što kada se sruši SEC?

- Ponovno se pokreće SEC koji analizira log sage
- Analiza loga:
  - Ako su svi zahtjevi u logu započeli i završili (imamo B i E poruke u logu), SEC se nalazi u sigurnom stanju
    - Staviti AS i započeti s kompenzirajućim zahtjevima za izvršene zahtjeve ili
    - Nastaviti sa izvršavanjem ostalih zahtjeva
  - Inače se nalazi u nesigurnom stanju
    - Staviti AS
    - Započeti s kompenzirajućim zahtjevima za sve zahtjeve u logu uključujući i zahtjeve koji nemaju poruke o kraju

# Trendovi i smjernice

# Reaktivno programiranje

- Ideja: varijable koje se mijenjaju kroz vrijeme promatramo kao tok podataka
- Funkcije definiraju relacije između varijabli i kada se neka promijeni pokreće se izvršavanje funkcija
- Koristi se modificirani oblikovni obrazac *Observer*
  - Ima povratnu vezu
- Modeliranje se vrši pomoću [Marble dijagrama](#)



# Reaktivno programiranje

- Kako je nastalo?
  - Rx.NET - Microsoft Research 2007.-2009. (Jafar Husain)
    - Odgovor na složenost asinkrone obrade
  - Netflix - 2011. Jafar prelazi u Netflix
    - 2013. objavljen RxJava kako otvoreni kod
  - Inicijativa [Reactive Streams](#) (2013.) – Netflix, Pivotal, Typesafe
    - Definirali sučelja (2015.)
    - Implementacije - JSR 166 (Java 9), RxJava 2.0, Project Reactor (Pivotal),...
- Gdje se koristi?
  - Biblioteke: Rx (Reactive Extension) .NET, RxJava (Java), Reactor Project (Spring Java), Typesafe web framework, Bacon.js (javascript), RxJS (javascript)
  - Sustavi: Hystrix, node.js, različite noSQL baze imaju takve *drive*re
- Više o tome na kolegiju: Konkurentno programiranje

# Računarstvo na rubu

- Računarstvo na rubu (*edge computing*)
  - seli obradu podataka bliže izvoru (gdje se podaci generiraju) tj. što bliže rubu sustava
  - rub ima različite „definicije“ ovisno o tome tko govori o tome:
    - za oblak je rub sve što je izvan oblaka
    - za Cisco je rub usmjeritelj na rubu mreže
    - za telekomunikacijske kompanije je rub obrada na baznim stanicama
    - za IoT je rub krajnji uređaj ili mrežni prilaz (gateway) ili računala u industrijskom postrojenju
- svojstva:
  - smanjuje se količina podataka koji se prenose mrežom jer se ne moraju svi podaci slati u oblak
  - povećana sigurnost podataka jer podaci ne izlaze iz lokalne mreže (GDPR)
  - brže vrijeme odziva jer je manje kašnjenje u mreži

Više: [From Cloud Computing to Edge Computing](#), [Eclipse ioFog](#), [Edge computing - the way forward for Eclipse IoT](#)

# Računarstvo u magli

- Računarstvo u magli (*fog computing*)
  - standard koji definira kako bi računarstvo na rubu trebalo raditi
  - OpenFog Consortium započeo standardizaciju 2015.
    - osnivači: Cisco, Intel, Microsoft, Princeton University, Dell i ARM
    - 2016. objavljaju „OpenFog Reference Architecture“ white paper, a 2017. standard
    - 2018. IEEE Standards Association prihvaćaju arhitekturu
    - 2019. se pridružuju Industrial Internet Consortium
      - bave se standardizacijom u području IIoT (Industrial IoT) i Industry 4.0
- primjenjuje koncepte iz oblaka na mrežu uređaja na rubu
- to je horizontalna arhitektura koja raspoređuje i upravlja resursima i uslugama računanja, spremanja podataka, kontrole i umrežavanja
- više: Eclipse fog05,

# Put prema brzim podacima – *fast data*

- Tri faze:
  1. Obrada velike količine podataka “preko noći” – “*data at rest*”
    - Obrada podataka u pozadini
    - Kašnjenja rezultata u satima
    - Npr. Hadoop
  2. Reakcija u realnom vremenu – “*data in motion*”
    - Tok podataka koji se obrađuje u sekundama
    - Rezultat se vraća natrag u sustav i koristi se za druge stvari
    - Potrebne hibridne arhitekture s dva sloja:
      - “*speed layer*” – za odgovor u realnom vremenu
      - “*batch layer*” – za obradu u pozadini
    - Npr. lambda arhitekture kao što su AWS Lambda, ...
  3. Samo “*data in motion*”
    - Kompletno izbacivanje obrade u pozadini
    - “Čista” obrada tokova podataka
    - Rasподijeljeni sustavi kao što su: Flink, Spark streaming, Google Cloud Data flow
- Više o tome na kolegijima:
  - Rasподijeljena obrada velike količine podataka
  - Analiza velikih skupova podataka

# Literatura

- [1] K. Hwang, J. Dongarra, and G. C. Fox, "Distributed and cloud computing: From parallel processing to the internet of things," Morgan Kaufmann, 2011.
- [2] N. R. Adiga, G. Almasi, Y. Aridor, R. Barik, D. Beece, R. Bellofatto, G. Bhanot, et al., "An Overview of the BlueGene/L Supercomputer," SC2002, Baltimore, USA, 2002, p. 60.
- [3] <http://www.top500.org/>
- [4] <http://en.wikipedia.org/wiki/Folding@home>
- [5] S. Venugopal, R. Buyya, K. Ramamohanarao: "A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing", ACM Computing Surveys
- [6] Jonas Bonér: "Reactive Microsystems - The Evolution of Microservices at Scale", O'Reilly, 2017., [technical report](#)