Микросервисни обрасци

У изради

Проф. др Игор Дејановић (igord at uns ac rs)

Креирано 2024-12-04 Wed 11:25, притисни ESC за мапу, Ctrl+Shift+F за претрагу, "?" за помоћ

Садржај

Увод

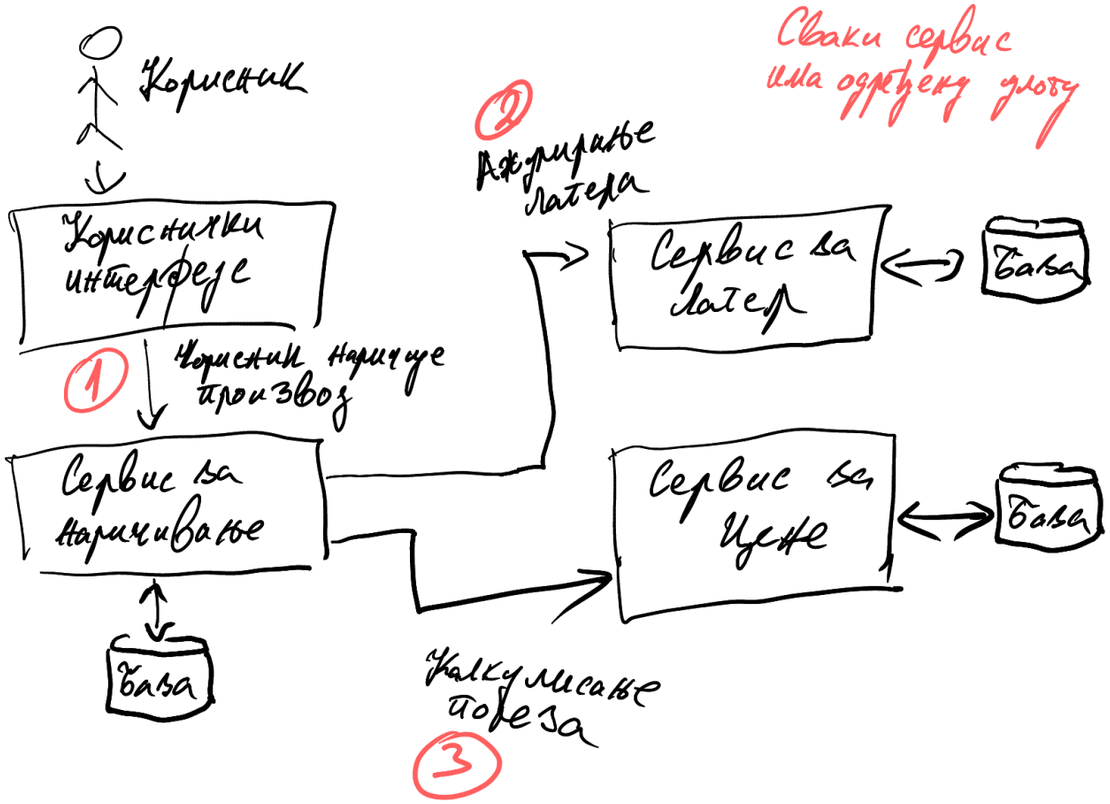
Микросервиси (*Microservices*)

- Софтверске компоненте.
- Висока кохезија, слаба спрега.
- Комуникација се обавља путем стандардних технолошки независних протокола (најчешће HTTP употребом REST стила уз JSON формат порука).
- Могу бити имплементирани у различитим програмских језицима и технологијама.
- Мали су, ограниченог контекста. Независно се развијају и распоређују.
- Имају јасно дефинисане интерфејсе путем којих комуницирају.
- *Unix* филозофија:

> *Do one thing and do it well.*

Архитектура базирана на микросервисима

- Архитектонски стил где се апликација гради као скуп слабо спрегнутих "малих" сервиса (микросервиса) који сарађују.
- Варијанта *Service-Oriented Architecture* (SOA) али сервиси су "мали" и протоколи за комуникацију су једноставни (*light-weight*).
- Микросервиси пружају услуге и/или користе друге микросервисе.
- У циљу независне миграције микросервиса на нове верзије интерфејси се верзионирају и омогућава се клијентима да користе старе интерфејсе у прелазном периоду.



Предности у односу на монолитну архитектуру

- Тимови могу бити технолошки хетерогени.
- Распоређивање (*deployment*) се обавља у малим инкрементима (*fine-grained*).
- Модуларност, декомпозиција. Лакше разумевање, развој и тестирање. Отпорност на "ерозију архитектуре".
- Боља скалабилност. Боља еластичност. Лако додавање нових микросервиса по потреби.
- Боља отпорност на отказе. Уколико један микросервис "падне" остатак апликације наставља да ради.
- Лакша миграција на нове технологије. Могућа постепеним заменама микросервиса.
- Интеграција хетерогених и "старих" система (*legacy*).
- Континуална интеграција и достава (*Continuous Integration/Delivery*)

Мане у односу на монолитну архитектуру

- Више "покретних делова". Захтева боље алате за распоређивање и надзор.
- Теже дебаговање. Дебаговање захтева праћење захтева кроз више микросервиса који се извршавају често на различитим физичким/виртуелним рачунарима.
- Додатни трошкови (*overheads*) услед комуникације.
- Додатни трошкови у случају потребе за дељењем података.

Обрасци за рад са базама података

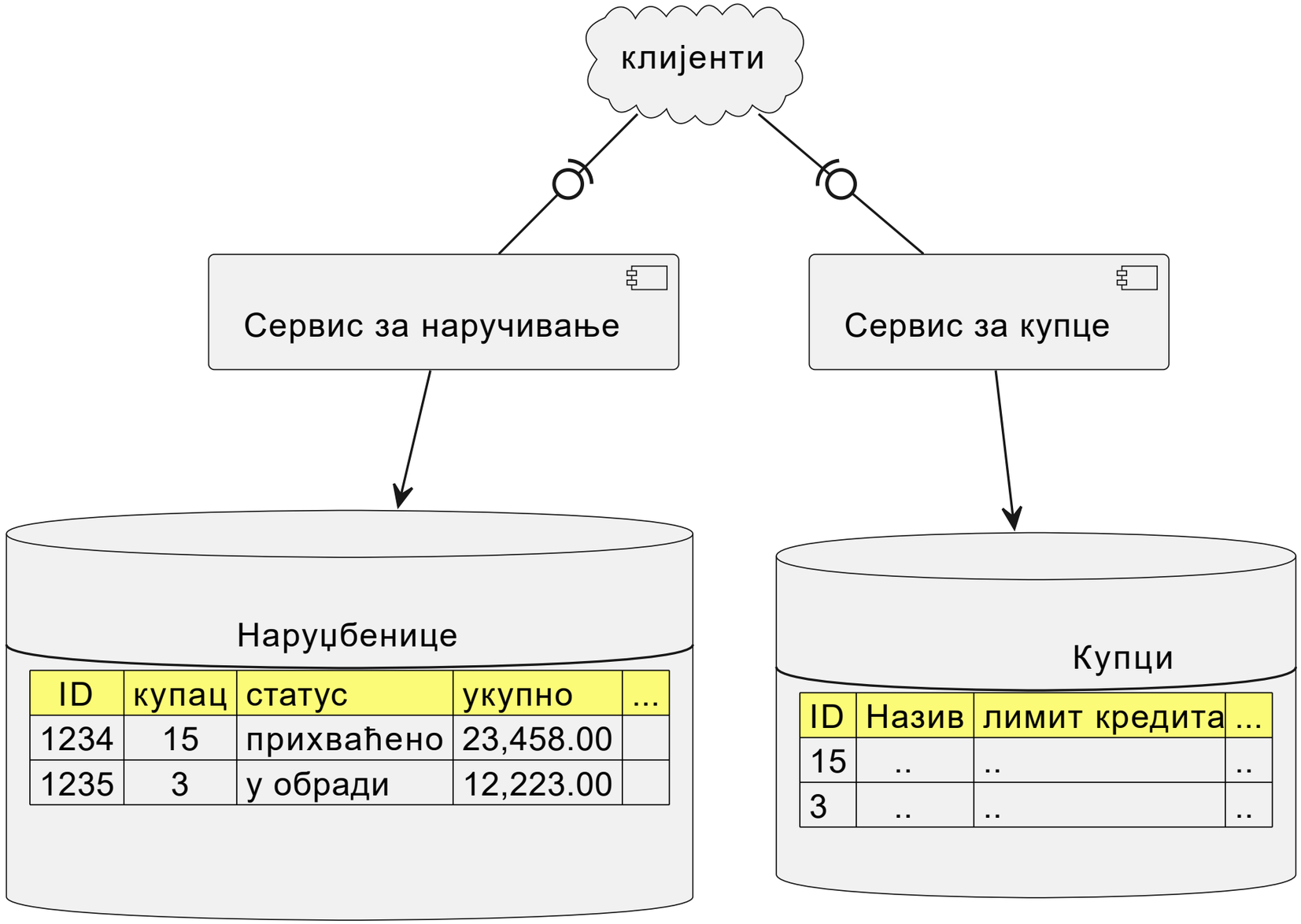
База података по сервису (*Database per service*)

- У циљу слабог спрезања сервиса подаци над којима сервиси оперишу се имплементирају као приватни.
- Други сервиси не могу приступити подацима директно већ само кроз интерфејс сервиса.

https://microservices.io/patterns/data/database-per-service.html

Структура

```
cloud "клијенти" as klijenti
component "Сервис за наручивање" as servis1
database baza1 [

                      Наруџбенице
----
<#FBFB77>|   ID | купац | статус     | укупно    | ... |
| 1234 |    15 | прихваћено | 23,458.00 |     |
| 1235 |     3 | у обради   | 12,223.00 |     |


]
servis1 --> baza1
servis1 -up0)- klijenti
component "Сервис за купце" as servis2
database baza2 [

                              Купци
----
<#FBFB77>| ID | Назив | лимит кредита| ... |
| 15 |    .. | .. | .. | ..    |
| 3  |    .. | .. | .. | ..    |

]
servis2 --> baza2
servis2 -up0)- klijenti
```


клијенти
Сервис за наручивање
Сервис за купце
Наруџбенице
ID
купац
статус
укупно
...
1234
15
прихваћено
23,458.00
1235
3
у обради
12,223.00
Купци
ID
Назив
лимит кредита
...
15
..
..
..
3
..
..
..

Предности

- Подаци су део имплементације сервиса.
- Имплементација приватне базе се може мењати независно од остатка система.
- Могуће је користити хетерогене технологије.

Мане

- Отежано извођење трансакција које се обављају између података различитих сервиса. Видети образац Saga.
- Отежани сложени упити који обухватају податке више сервиса. Видети образац CQRS.

Начини имплементације

- Употреба једне инстанце базе за све сервисе:
  - приватне табеле по сервису,
  - приватна шема по сервису.
- Посебна инстанца базе по сервису.

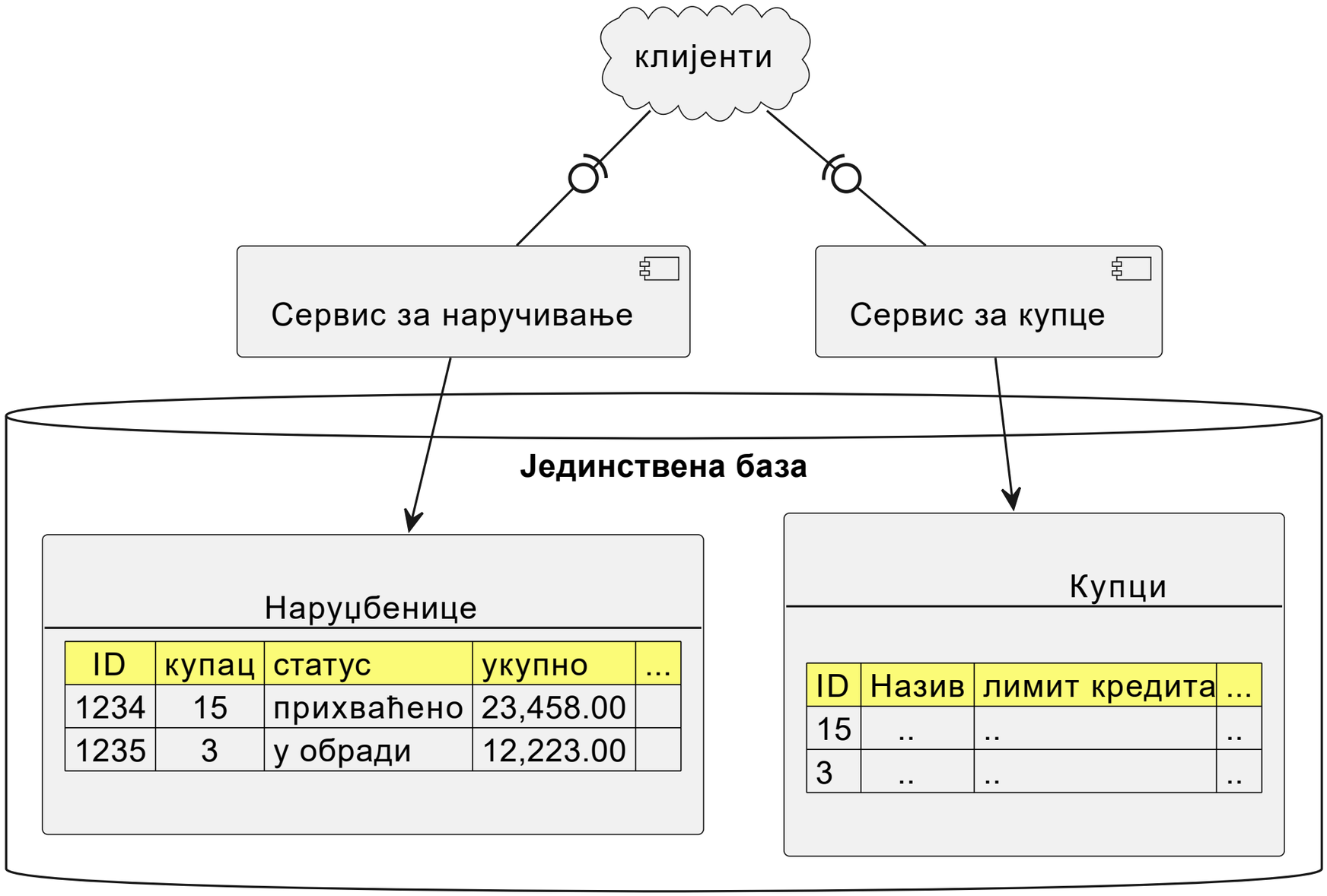
Дељена база података (*Shared database*)

- У циљу подршке за ACID трансакције сервиси деле исту базу података и могу слободно да приступају подацима других сервиса.

https://microservices.io/patterns/data/shared-database.html

Структура

```
cloud "клијенти" as klijenti
component "Сервис за наручивање" as servis1
database "Јединствена база" {

card narudzbenica [

                    Наруџбенице
---
<#FBFB77>|   ID | купац | статус     | укупно    | ... |
| 1234 |    15 | прихваћено | 23,458.00 |     |
| 1235 |     3 | у обради   | 12,223.00 |     |

]

card kupci [

                              Купци
---

<#FBFB77>| ID | Назив | лимит кредита| ... |
| 15 |    .. | .. | .. | ..    |
| 3  |    .. | .. | .. | ..    |

]

}
servis1 --> narudzbenica
servis1 -up0)- klijenti
component "Сервис за купце" as servis2
servis2 --> kupci
servis2 -up0)- klijenti
```


клијенти
Сервис за наручивање
Сервис за купце
Јединствена база
Наруџбенице
ID
купац
статус
укупно
...
1234
15
прихваћено
23,458.00
1235
3
у обради
12,223.00
Купци
ID
Назив
лимит кредита
...
15
..
..
..
3
..
..
..

Предности

- Једноставније за имплементацију и операцију.
- Једноставније трансакције (ACID) и упити (нпр. могућ JOIN између табела различитих сервиса).

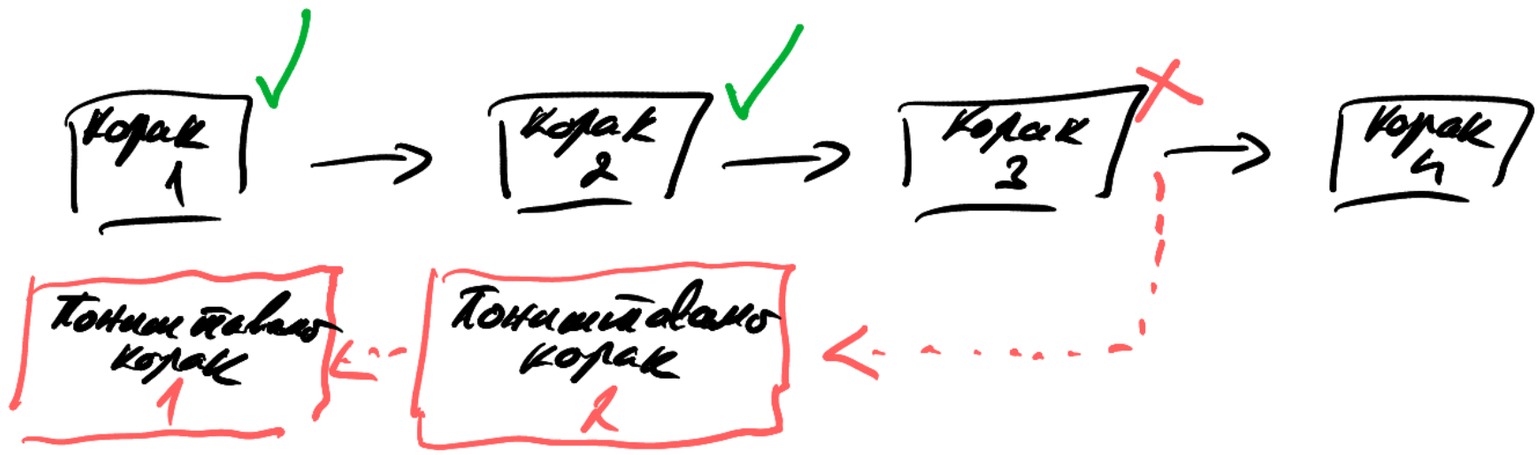
Мане

- Јача спрега између сервиса у време развоја (нпр. одржавање шеме базе мора бити координисано између тимова) и у време извршавања (нпр. један сервис може да закључа табелу и спречи друге сервисе да приступе).
- Сервиси могу да мењају податке других сервиса. Ово је могуће административно регулисати уколико база подржава.
- Иста база можда неће задовољити потребне функционалне и нефункционалне особине захтеване од стране неких сервиса.

Обезбеђивање конзистенције

Saga

- Вид дистрибуиране трансакције. Мање ригидна од *two-phase commit* (2PC).
- Очување конзистенције података између сервиса.
- Користи се када је у употреби Database per service образац за имплементацију трансакција.
- Низ локалних трансакција (*ACID*) које објавом поруке/догађаја иницирају следећу трансакцију у ланцу. Уколико нека од трансакција не успе, извршава се поништавање.

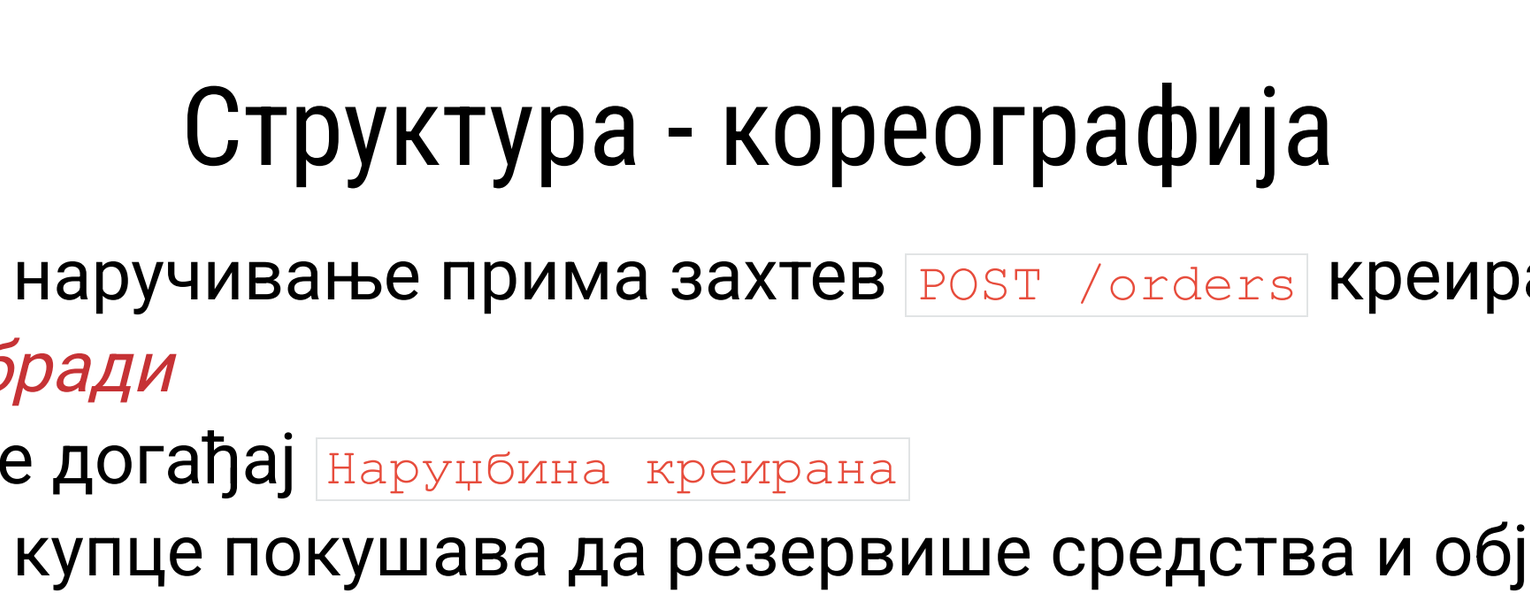


Приступи

- Два приступа у координацији трансакције:
  - Базиран на кореографији (*Choreography*) - после сваке локалне трансакције објављује се догађај који иницира извршавање следеће трансакције у низу.
  - Базиран на оркестрацији (*Orchestration*) - оркестратор (посебан објекат) је задужен да обавести учеснике да започну или да пониште трансакцију.

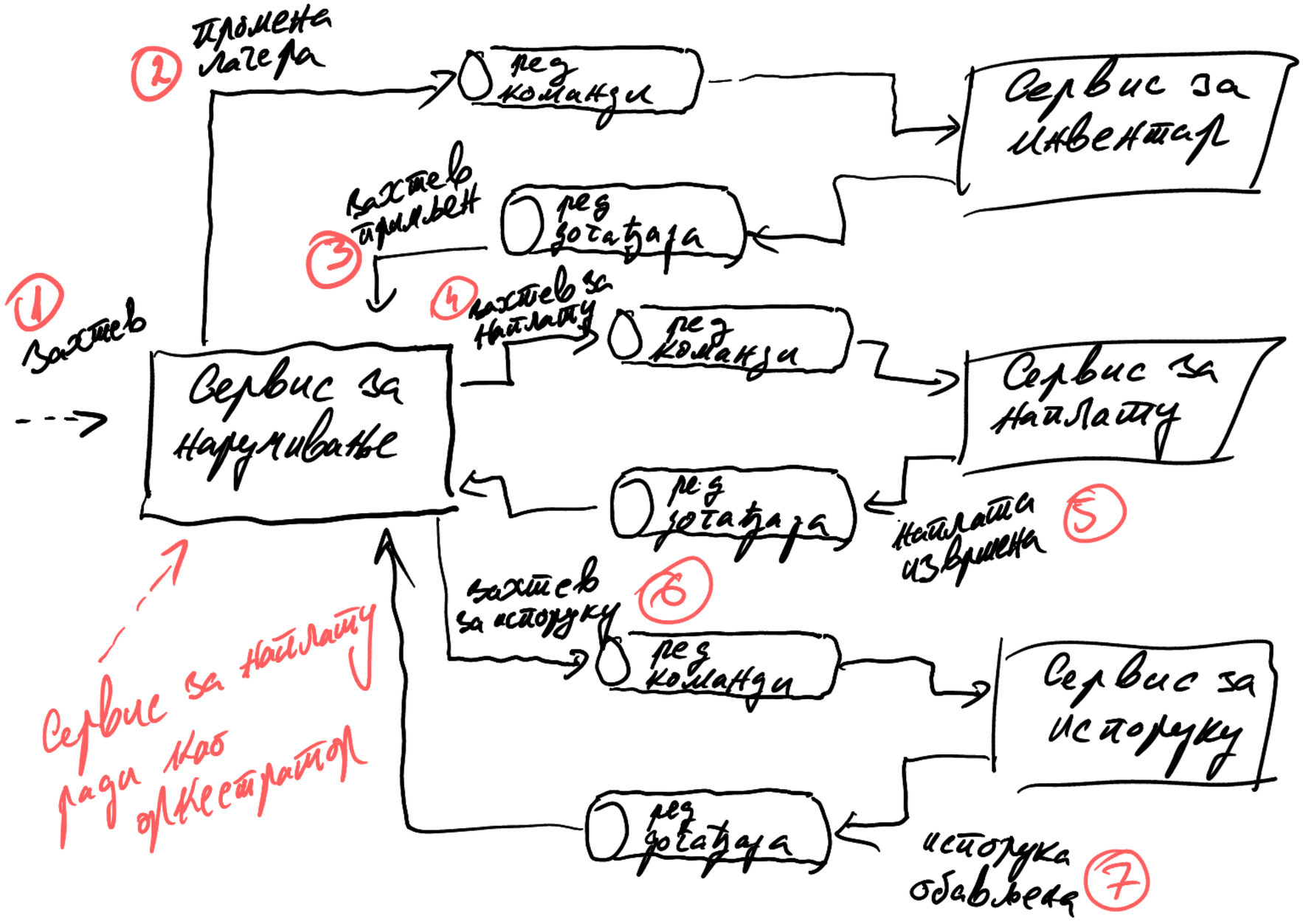
Структура - кореографија

1. Сервис за наручивање прима захтев `POST /orders` креира наруџбину у стању *у обради*
2. Прослеђује догађај `Наруџбина креирана`
3. Сервис за купце покушава да резервише средства и објављује догађај који представља резултат операције: средства су резервисана или је кредит прекорачен.
4. Сервис за наручивање прихвата или одбија наруџбину.



Структура - оркестрација

- Сервис за наручивање је оркестратор тј. задужен је за координацију целокупног процеса наручивања.



Предности

- Слабија спрега. Мање ригидан систем.
- Боља скалабилност.

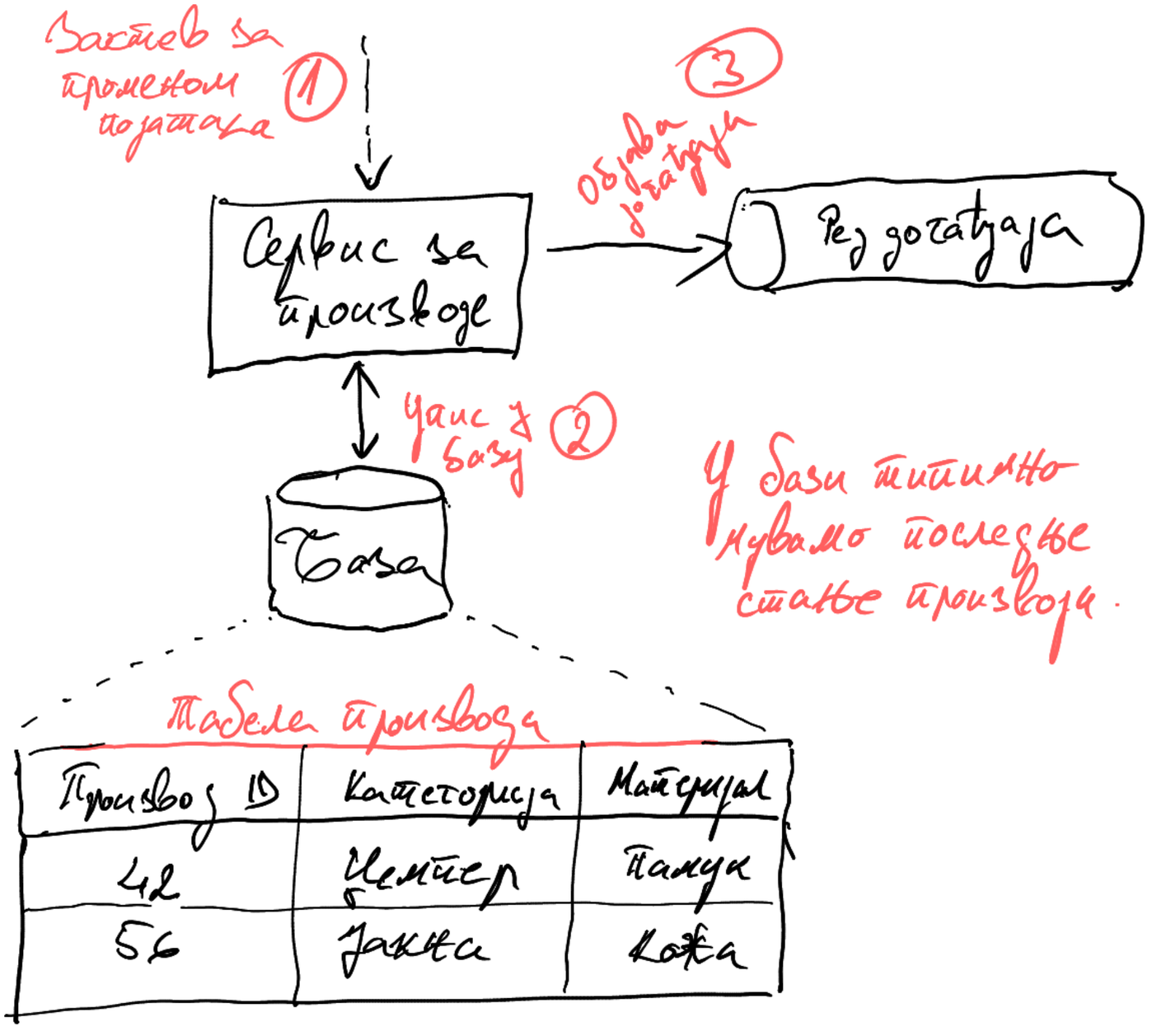
Мане

- Мора се пажљиво моделовати редослед операција због потенцијалног поништавања.
- Шта радити у ситуацији када је операција неуспешна због техничког проблема (нпр. сервис није тренутно доступан)?
- Модел конзистентности - коначна конзистентност (*Eventual Consistency*) - немамо гаранције да ће прочитани подаци бити право стање система тј. да ће бити конзистентни.
- BASE семантика (*Basically-Available, Soft-state, Eventual consistency*)
  - *Basically-Available* - добра доступност података али без гаранције на конзистентност
  - *Soft-state* - одређена вероватноћа да знамо стање јер систем можда још није конвергирао.
  - *Eventual consistency* - ако изменимо стање после одређеног времена измена ће бити видљива свим клијентима.

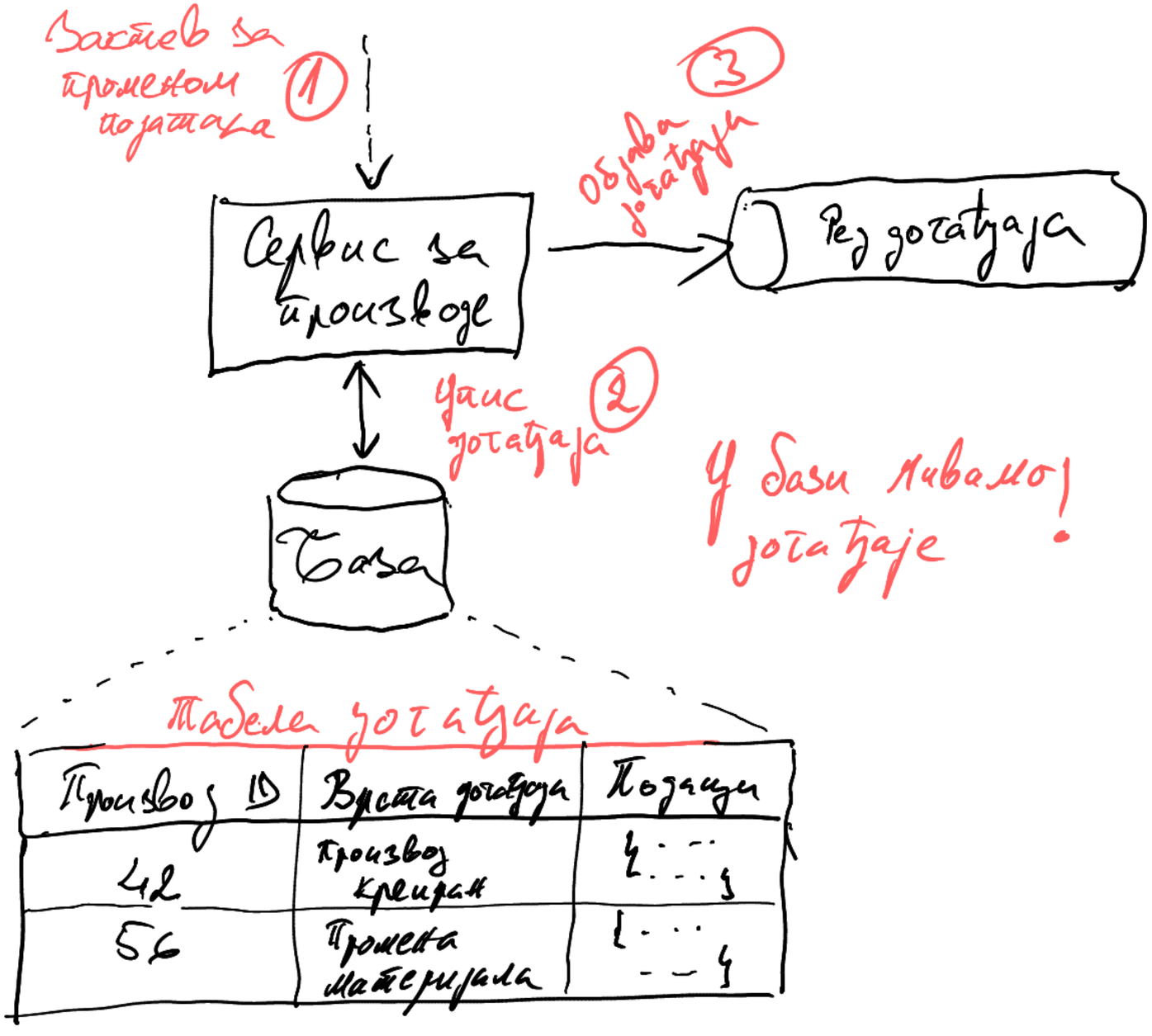
Event sourcing (ES)

- Користи се код архитектуре вођене догађајима (*Event-Driven*).
- Уместо чувања тренутног стања ентитета чува се низ догађаја који су мењали ентитет.
- Текуће стање се може добити применом свих догађаја до садашњег тренутка.
- Природно се користи са обрасцем CQRS.

Структура код класичног приступа



Структура код *ES* приступа



Напомене

- Ток догађаја треба да буде једини извор текућег стања.
- Предности:
  - различити модели се могу изградити применом тока догађаја у будућности.
  - природно садржи пуну историју измена што омогућава ревизију и контролу.
- Мана: немогућност постављања упита над током догађаја - због тога се користи у синергији са CQRS.
- Додатно можемо користити брокере порука (*message brokers*) уместо базе података.

Постављање упита

API композиција (*API Composition*)

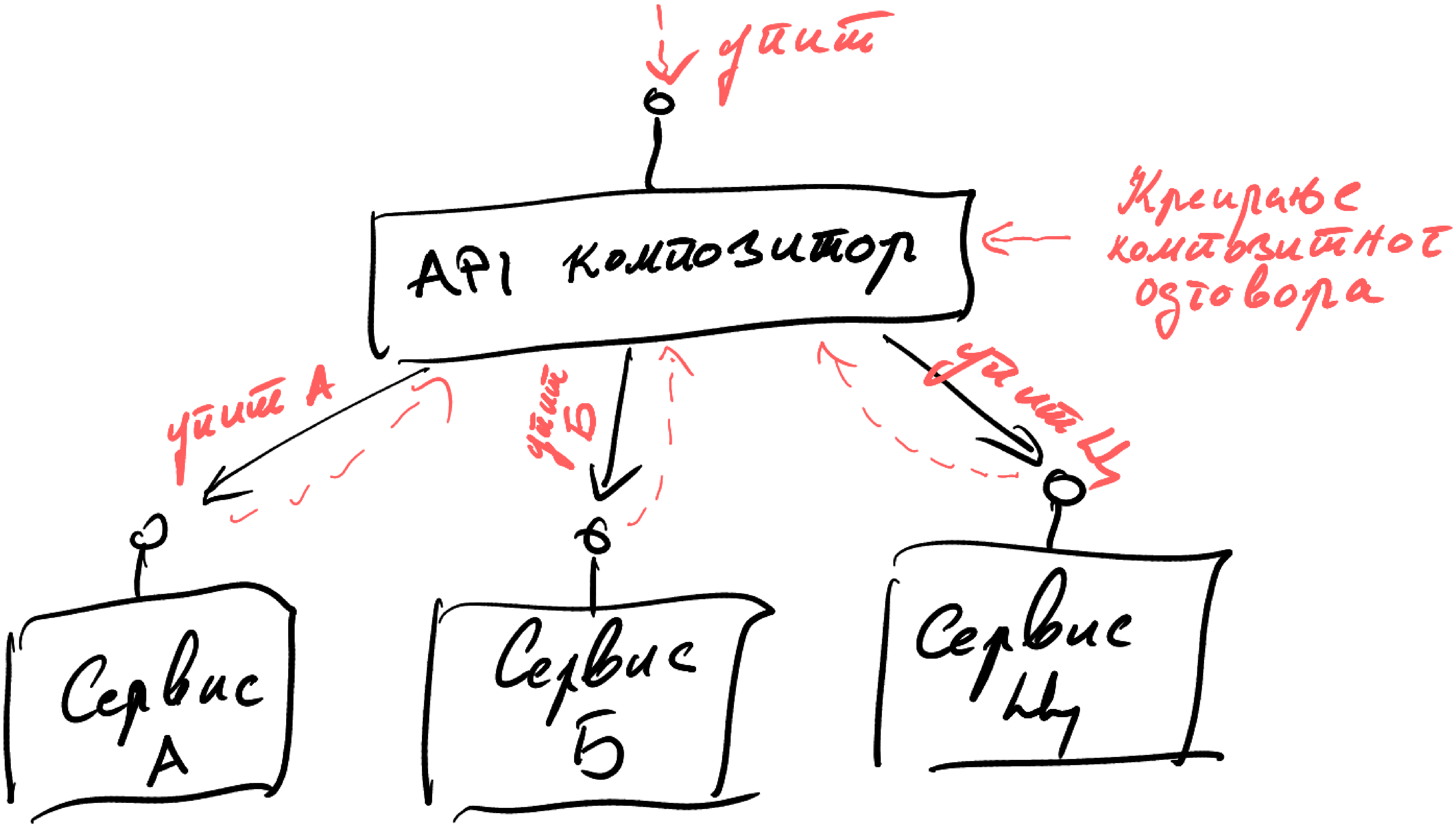
- У случају употребе обрасца *Database per service* поставља се питање како имплементирати упите који удружују податке из различитих микросервиса.

https://microservices.io/patterns/data/api-composition.html

Решење

- Креирати сервис који поставља појединачне упите и затим комбинује податке у меморији и враћа јединствени одговор са удруженим подацима.

Структура


упит
API композитор
Креирање композитног одговора
упит А
упит Б
упит Ц
Сервис А
Сервис Б
Сервис Ц

Предности и мане

- Предности:
  - Поједностављење сложених упита.
  - Једно место за ажурирање сложених упита.
- Мане:
  - Поједини упити могу бити неефикасни јер се велика количина података преноси преко мреже и удружује у меморији.

Пример

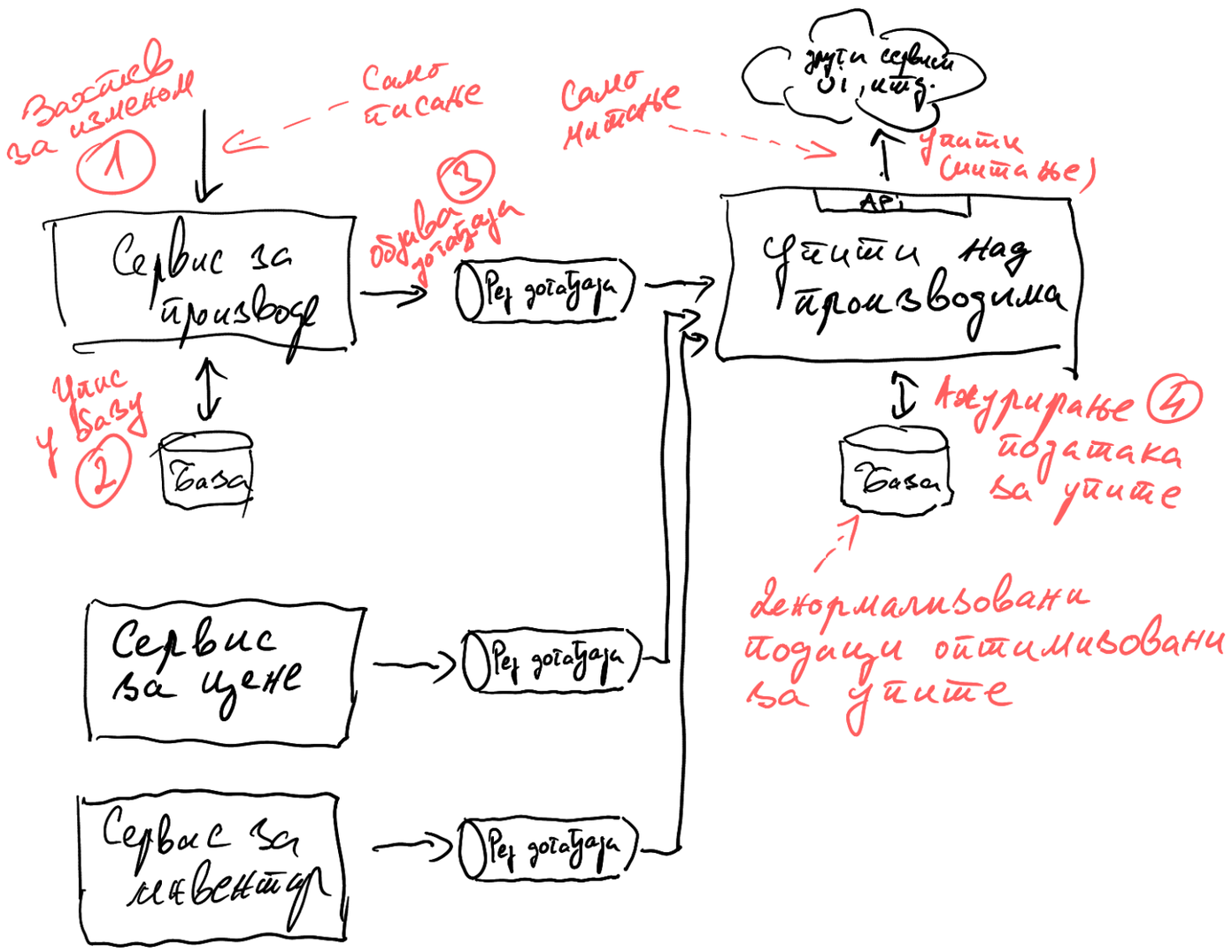
- Образац *API Gateway* често ради API композицију.

CQRS

- *Command Query Responsibility Segregation* (*CQRS*) се користи у ситуацији када имамо обрасце *Database per service* и *Event sourcing* имплементиране и желимо да подржимо упите који удружују податке из више микросервиса.
- Базиран на идеји поделе захтева на оне који мењају стање и оне који само читају тј. немају бочне ефекте. Еквивалентно са *REST* методама за читање (`GET, HEAD`) и измену стања (`POST, PUT, PATCH...`).
- Креирање базе која је само за читање и која се континуално ажурира обрадом догађаја који се емитују при промени података.

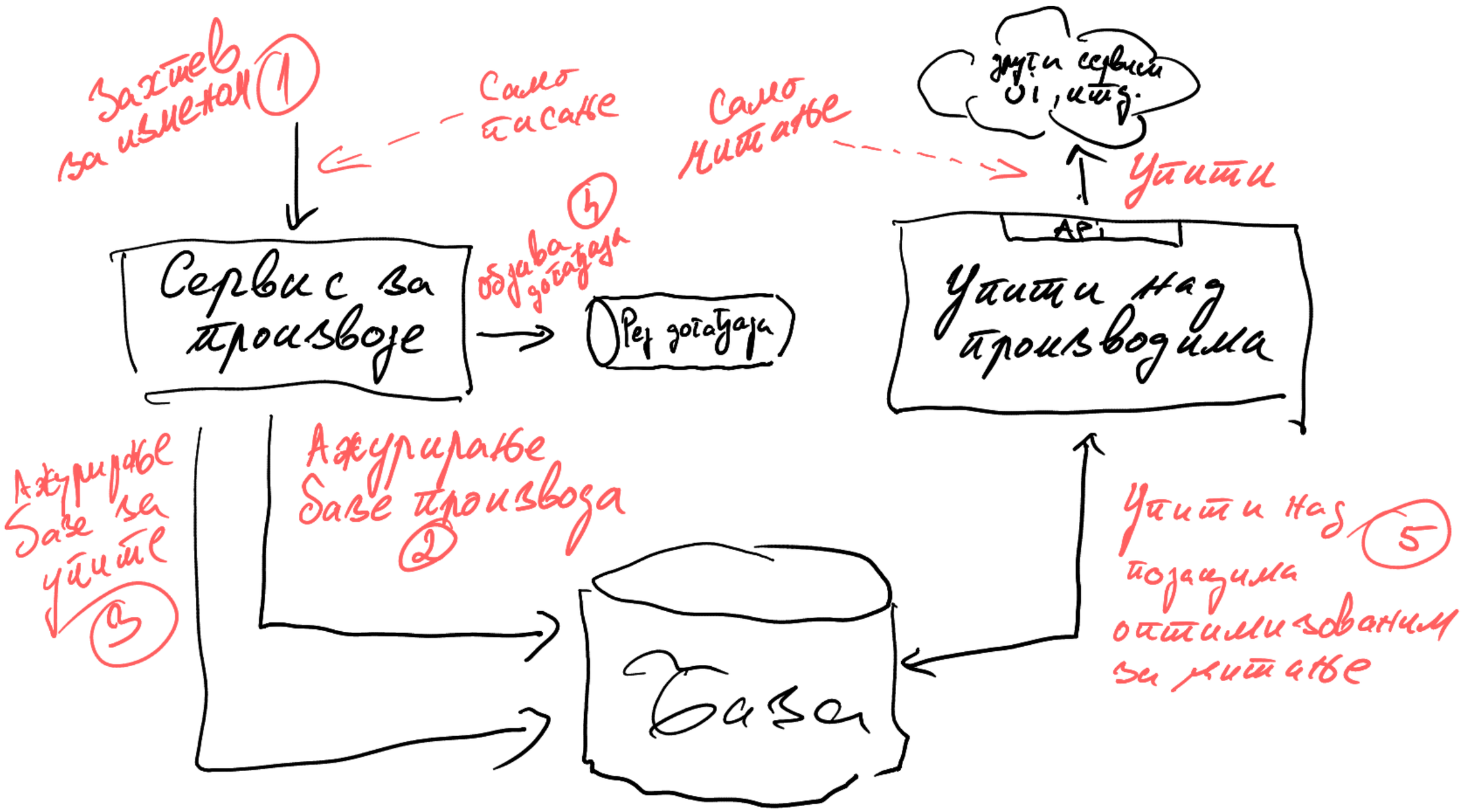
Структура - посебне базе

- Захтев за изменом производа `(1)` уписује текуће стање у локалну базу `(2)` и објављује догађај `(3)`.
- Сервис за упите, на основу догађаја, ажурира `(4)` свој интерни модел оптимизован за упите који чува у локалној бази.



Структура - јединствена база

- Захтев за изменом производа `(1)` уписује тренутно стање у локалну базу `(2)` и ажурира модел за упите `(3)` и затим објављује догађај `(4)`.
- Сервис за упите чита ажуран модел за упите `(5)`.



Предности

- Неминован код употребе обрасца *Event sourcing*.
- Боља подела надлежности.
- Једноставнији упитни модел. Боље перформансе упита. Подаци су најчешће денормализовани у циљу постизања оптималних перформанси.

Мане

- Увећана сложеност.
- Кашњење у репликацији. Коначна конзистентност (*Eventual Consistency*).
- Дуплирање података. Постоји могућност неконзистенције.

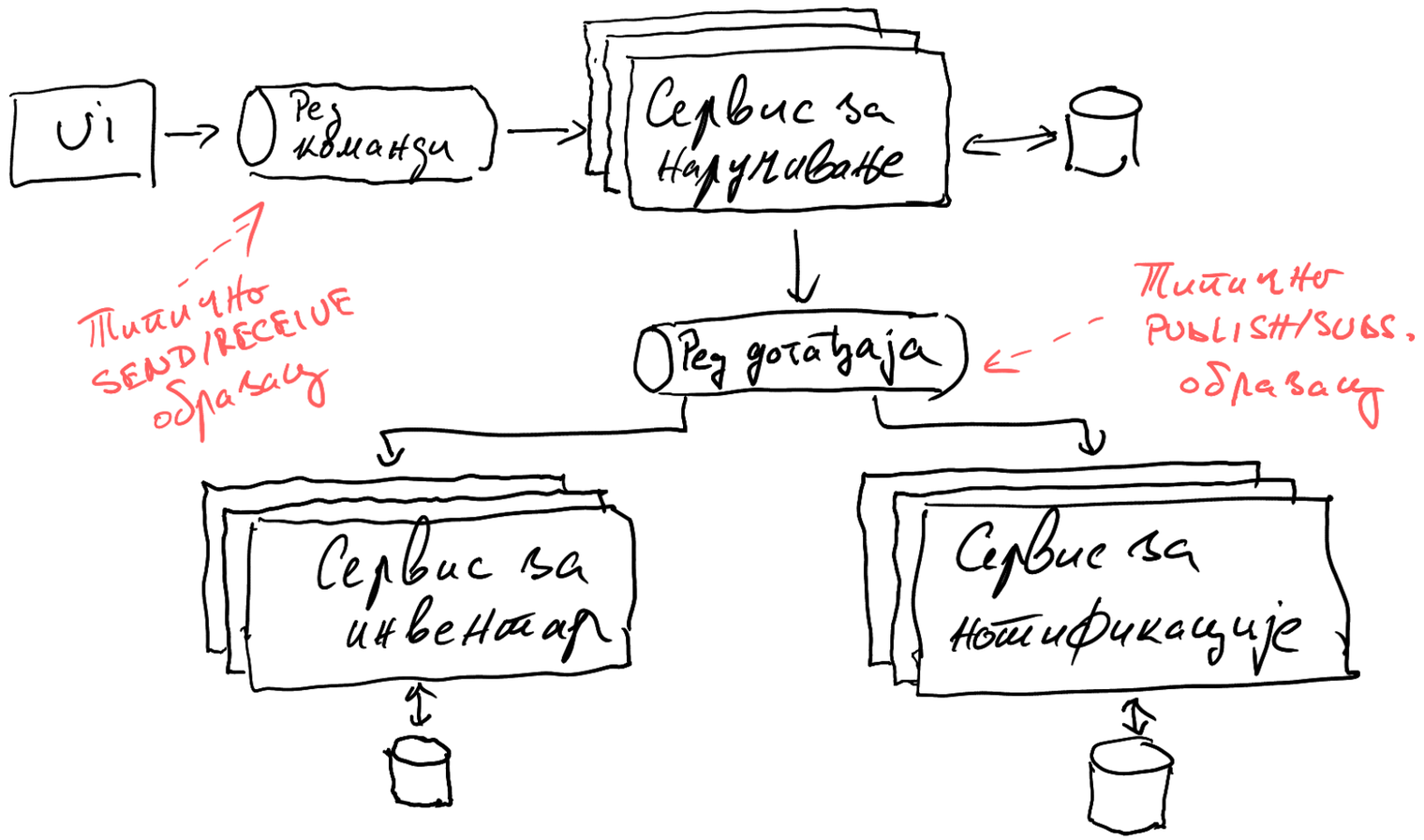
Комуникација

Messaging

- Слање порука је у основи архитектуре вођене догађајима.
- Асинхрона комуникација, слабо спрезање микросервиса.
- Посредник (*message broker*), који мора бити високо доступан, омогућава баферовање и перзистенцију порука.

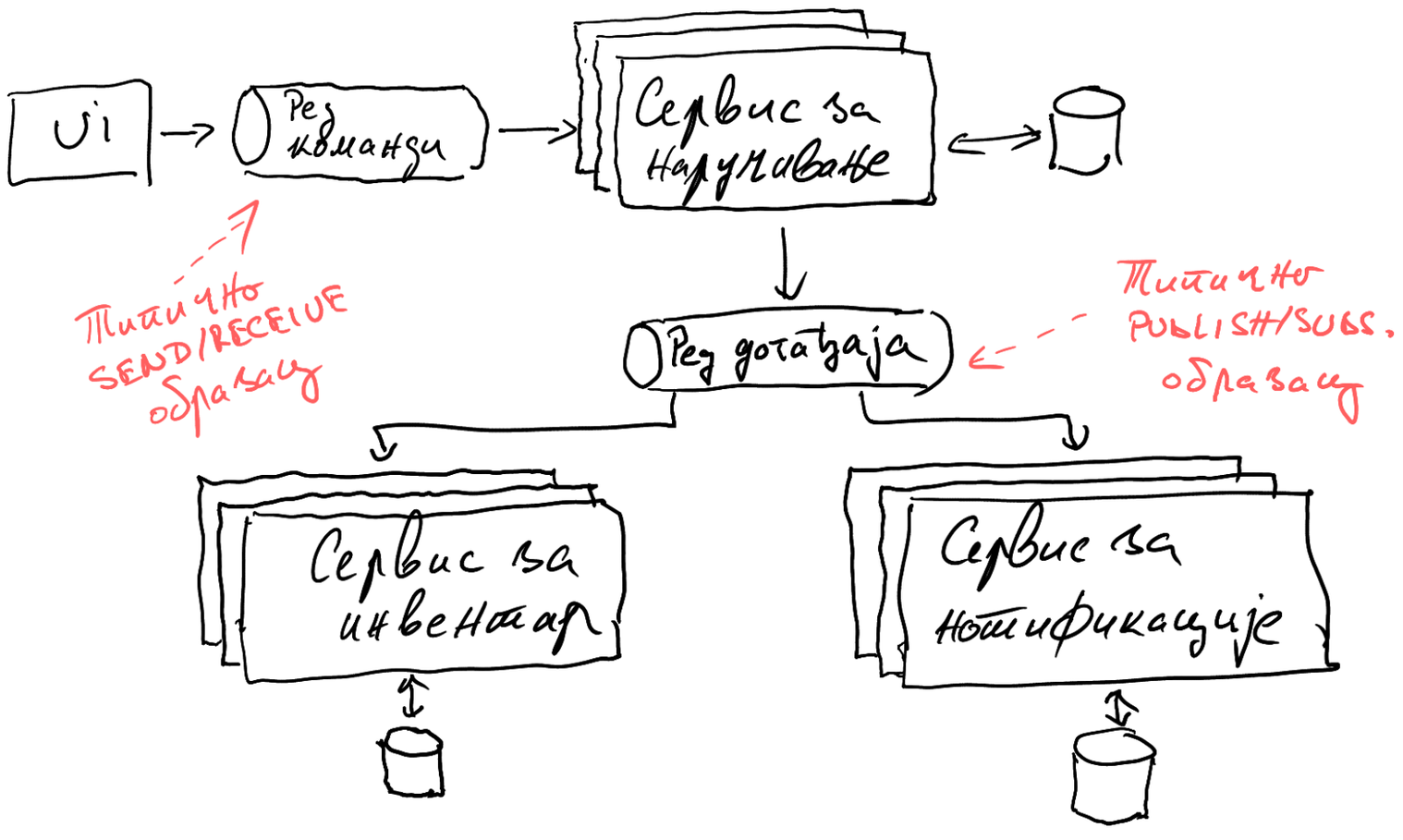
Send/Receive образац

- Обично представља комуникацију између два учесника (*point-to-point*) са специфичном наменом, најчешће извршење акције над циљним сервисом.
- Овај облик је коришћен типично од стране команди.
- Мора се обезбедити да само циљни сервис реагује на поруку.



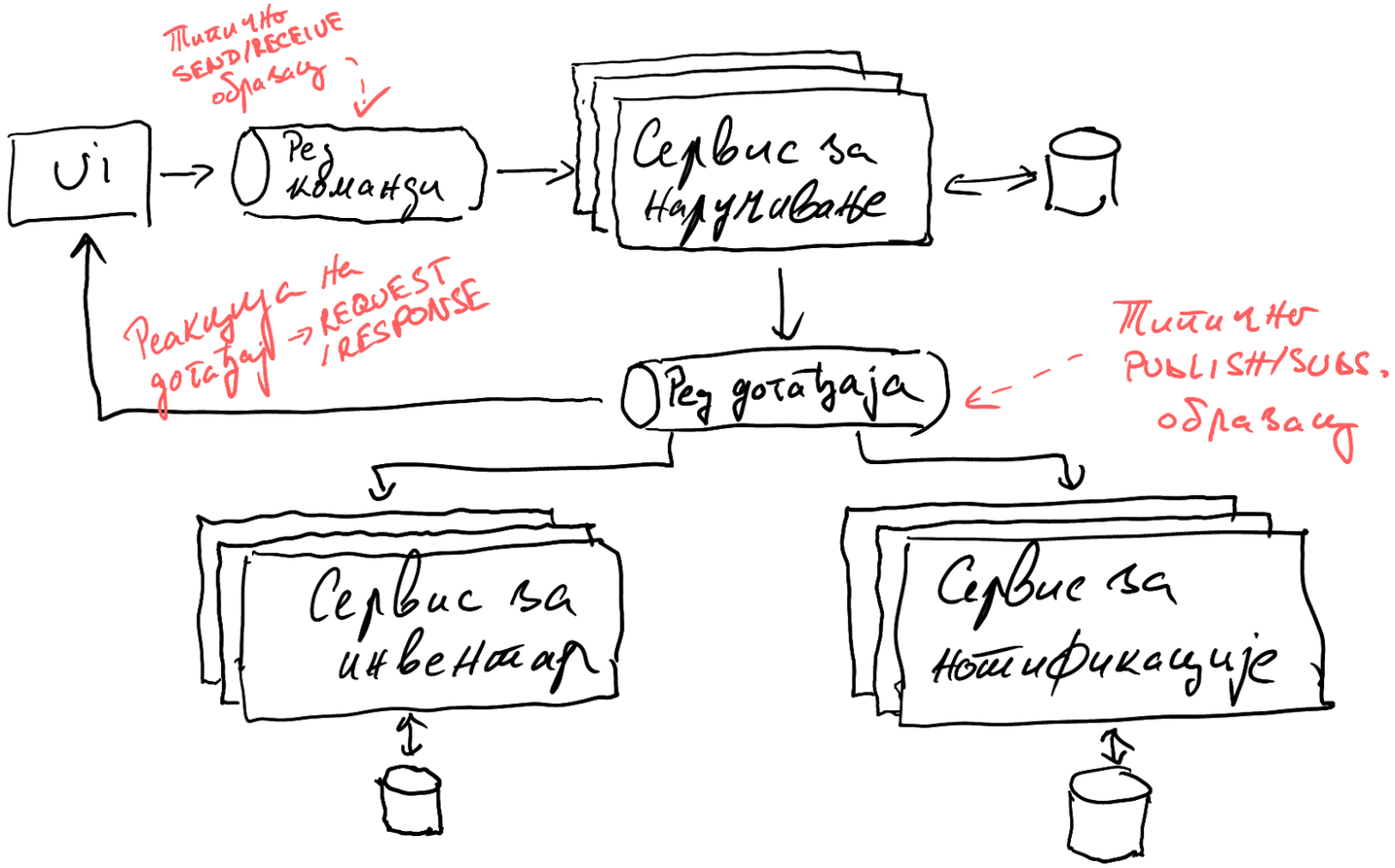
Publish/Subscribe образац

- Комуникација "један на више".
- Сервиси заинтересовани за одређене поруке се региструју (*subscribe*).
- Сервиси примају поруке и обрађују их у паралели различитом брзином.
- Основа хоризонталног скалирања.



Request/Response образац

- Имплементира се када је потребна повратна информација, обично при имплементацији *Send/Receive* образца.
- Порука/команда и одговор на команду могу бити корелирани одређеним идентификатором.



Remote Procedure Invocation (RPI)

- Сервиси често морају тесније сарађивати да би обрадили одређени захтев.
- Понекад је синхрони начин комуникације бољи. Тада користимо *RPI*.

Предности

- Једноставан вид комуникације. Синхрона варијанта *Request/Response* образца.

Мане

- Јако темпорално спрезање сервиса. Морају бити доступни истовремено.

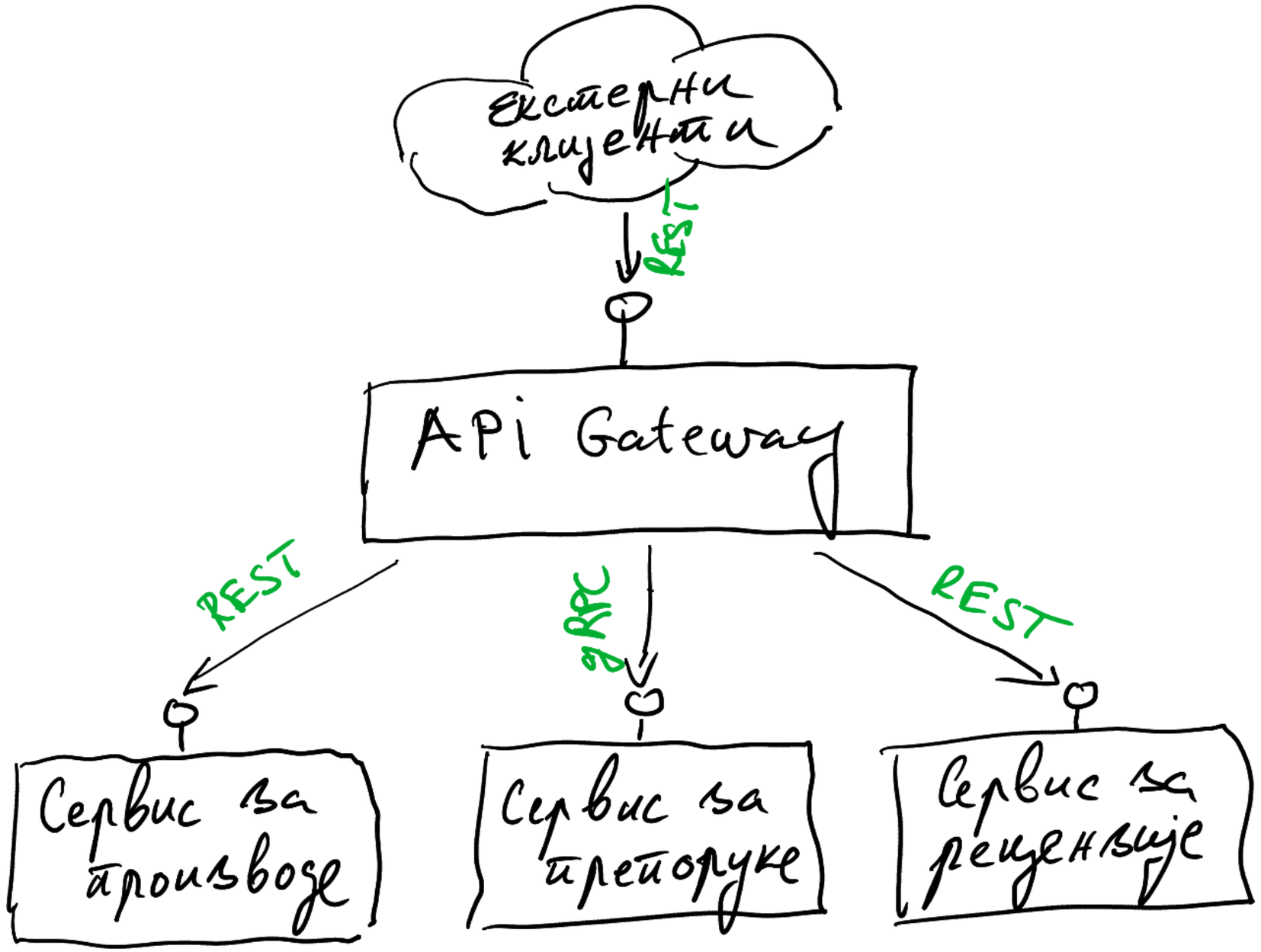
Приступи

- REST
  - Добро познат приступ. Основа комуникације на вебу.
  - Најчешће се користи у комбинацији са текстуалним порукама, нпр. JSON.
- gRPC
  - Развијен у Гуглу.
  - Акценат на перформансама.
  - Бинарне поруке базиране на технологији Protocol Buffers.
- Apache Thrift
  - Развијен у Фејсбуку.
  - Различити формати порука и транспортни протоколи.

API Gateway

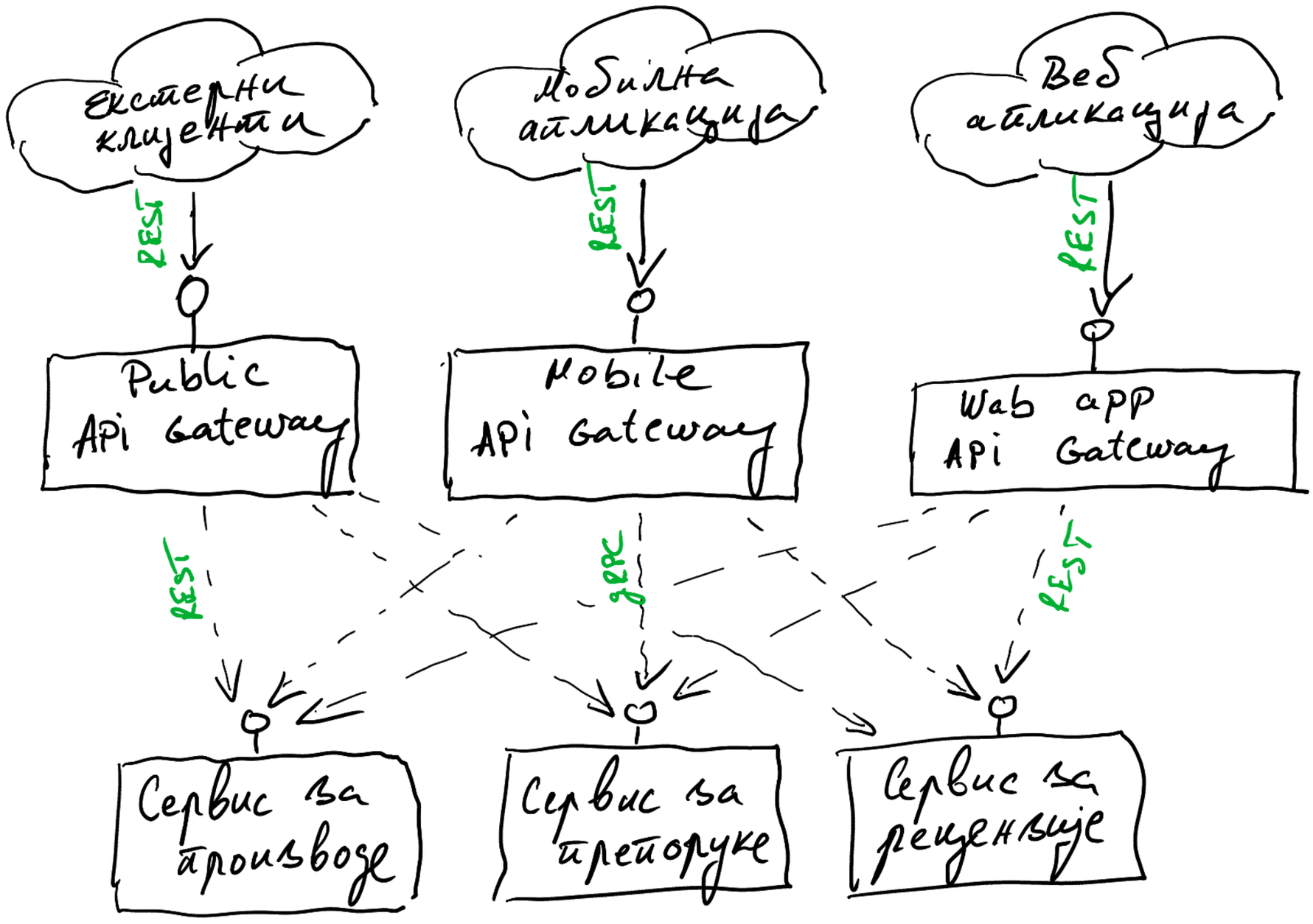
- Инстанца класичног ОО обрасца *Facade*. Скривање интерне сложености.
- Посредник за спољне клијенте.
- Специјализација обрасца API композиције за екстерне клијенте. Интеграција података са више микросервиса.
- Може имплементирати додатне функционалности, нпр. ауторизацију.

Структура


Екстерни клијенти
REST
API Gateway
REST
gRPC
REST
Сервис за производе
Сервис за препоруке
Сервис за рецензије

Варијанта - *Backends for Frontends*

- По један гејтвеј за сваки фронтенд.
- Специјализација API-ја.



Напомена

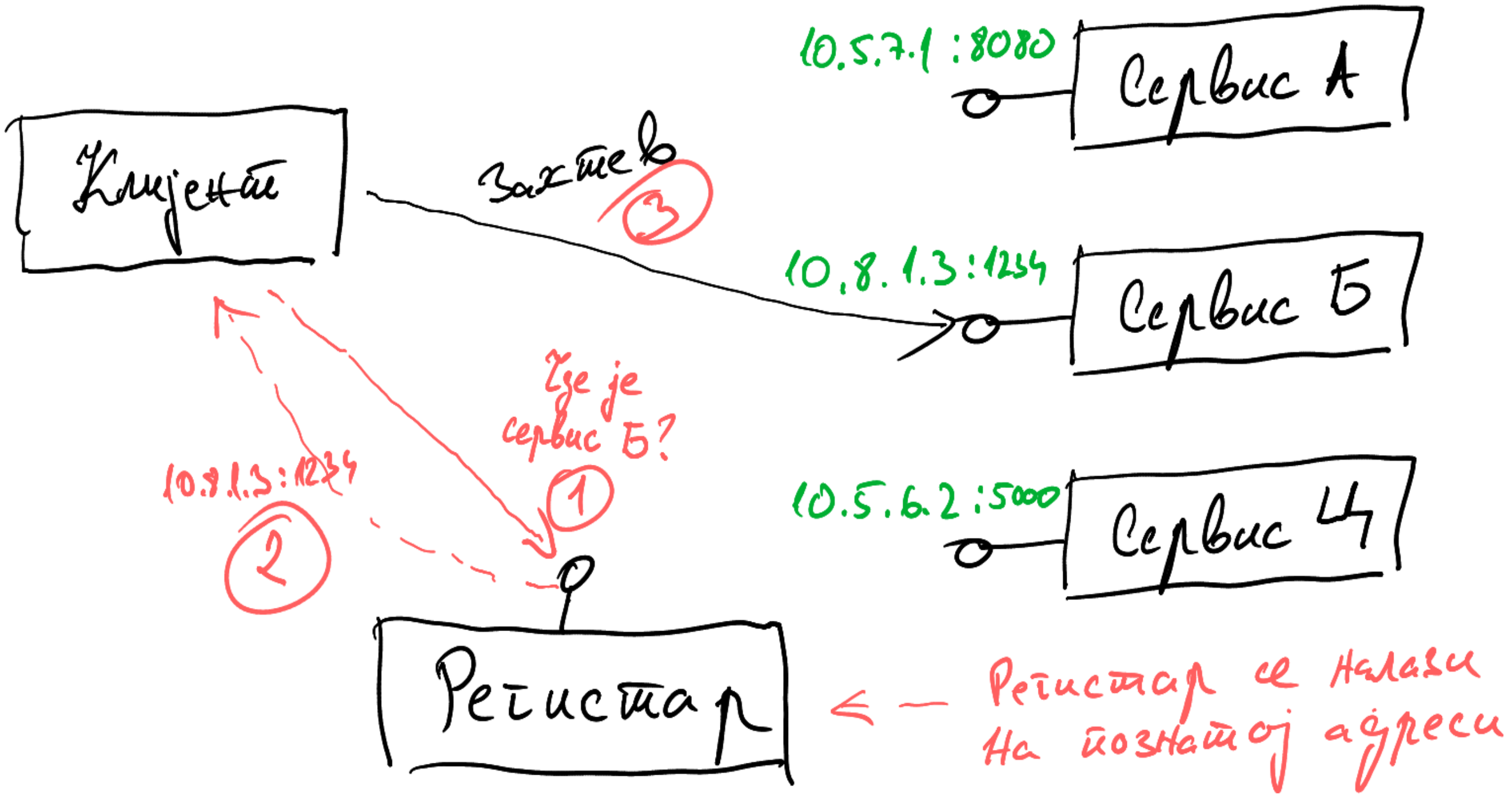
- Једна тачка отказа. Обезбедити високу доступност.

Откривање сервиса

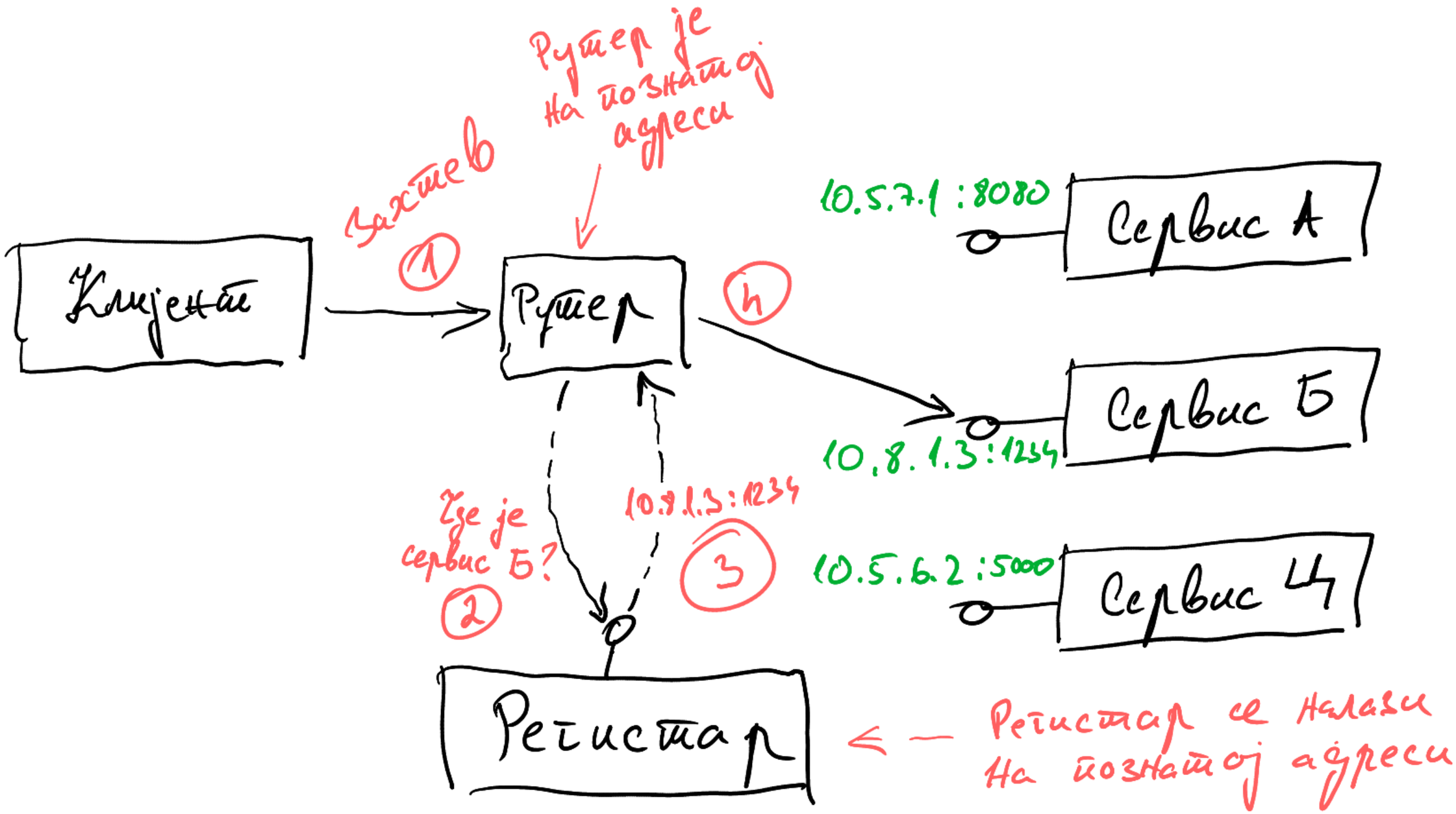
Service Registry

- За разлику од класичних дистрибуираних система код микросервисних архитектура сервиси нису увек на истој мрежној адреси.
- Како клијент сазнаје где се сервис налази?
- Специјални сервис који је увек на истој локацији и који има информације о локацијама свих других сервиса.
- Два приступа:
  - *клијентски* - код којег клијент сервис сам пита регистар,
  - *серверски* - код којег имамо посредника (*рутер*) који поставља питање регистру.

Структура - клијентски



Структура - серверски



Литература

- Hugo Filipe Oliveira Rocha, Practical Event-Driven Microservices Architecture, Apress, 2022.
- Microsoft, Cloud Design Patterns
- Chris Richardson, Microservice Architecture
- Wikipedia, Microservices