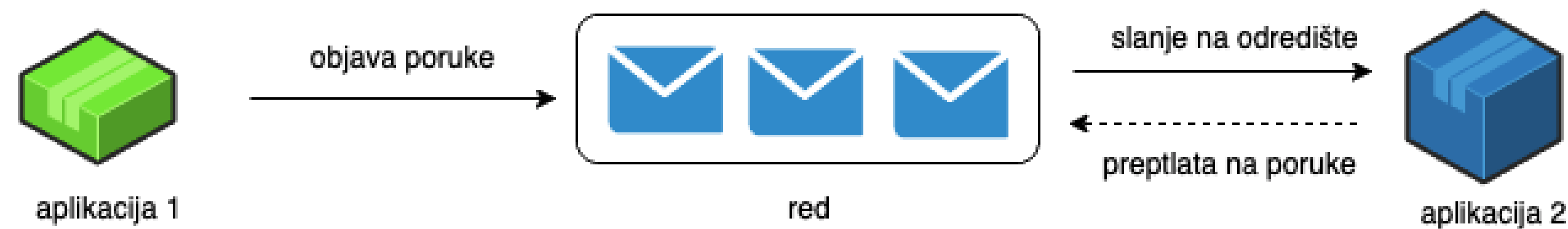
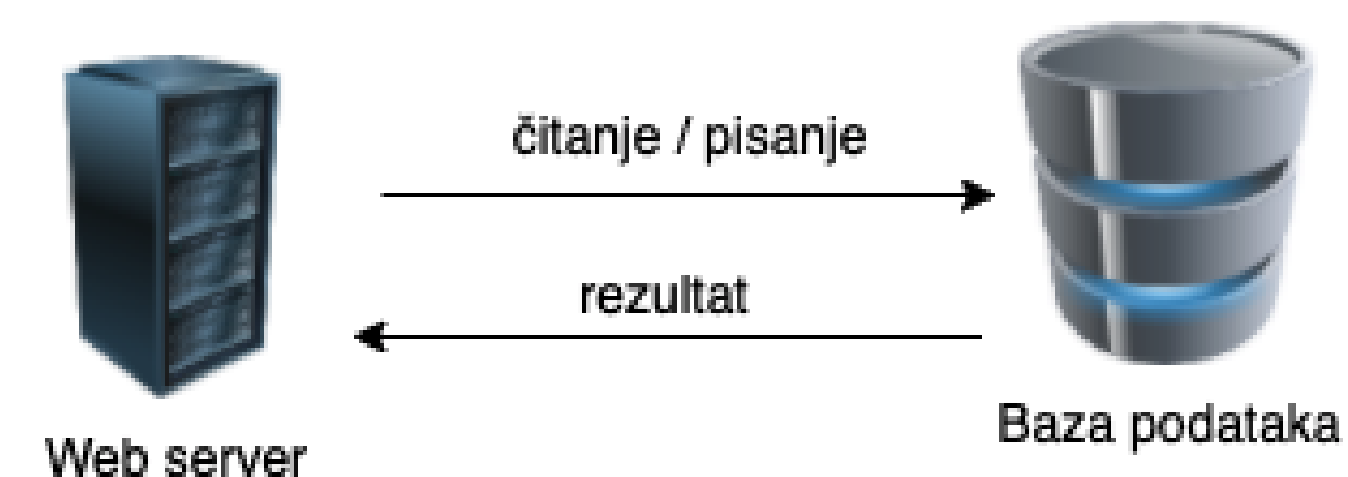


KOMUNIKACIJA SA BAZOM PODATAKA

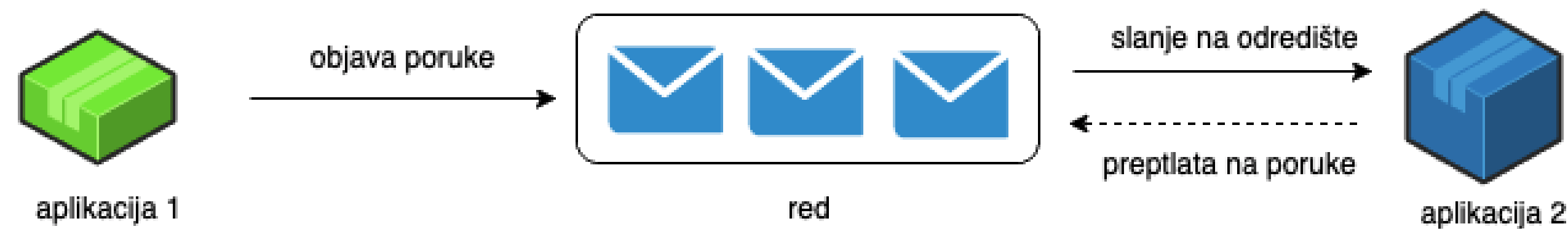
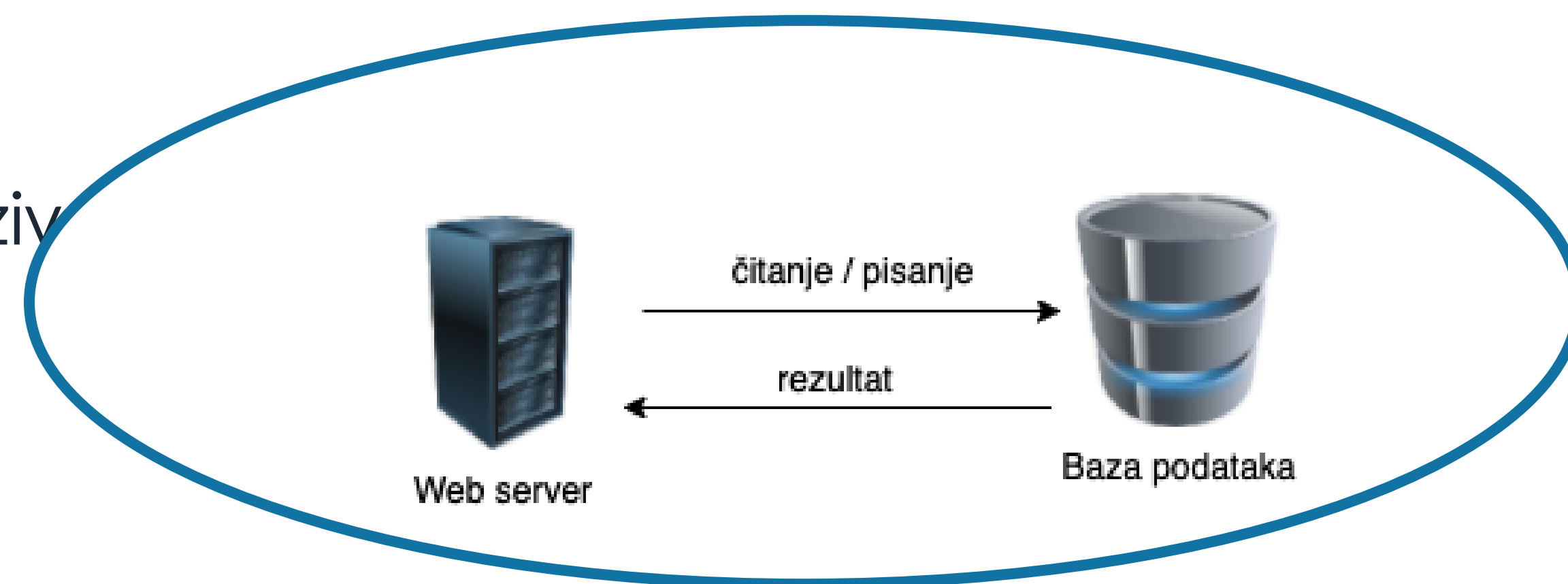
TRI NAJČEŠĆA SCENARIJA

- Serverska aplikacija <-> Baza podataka
- Direktna komunikacija klijent <-> server kroz poziv servisa
- Asinhrona komunikacija razmenom poruka preko reda poruka (message queue)



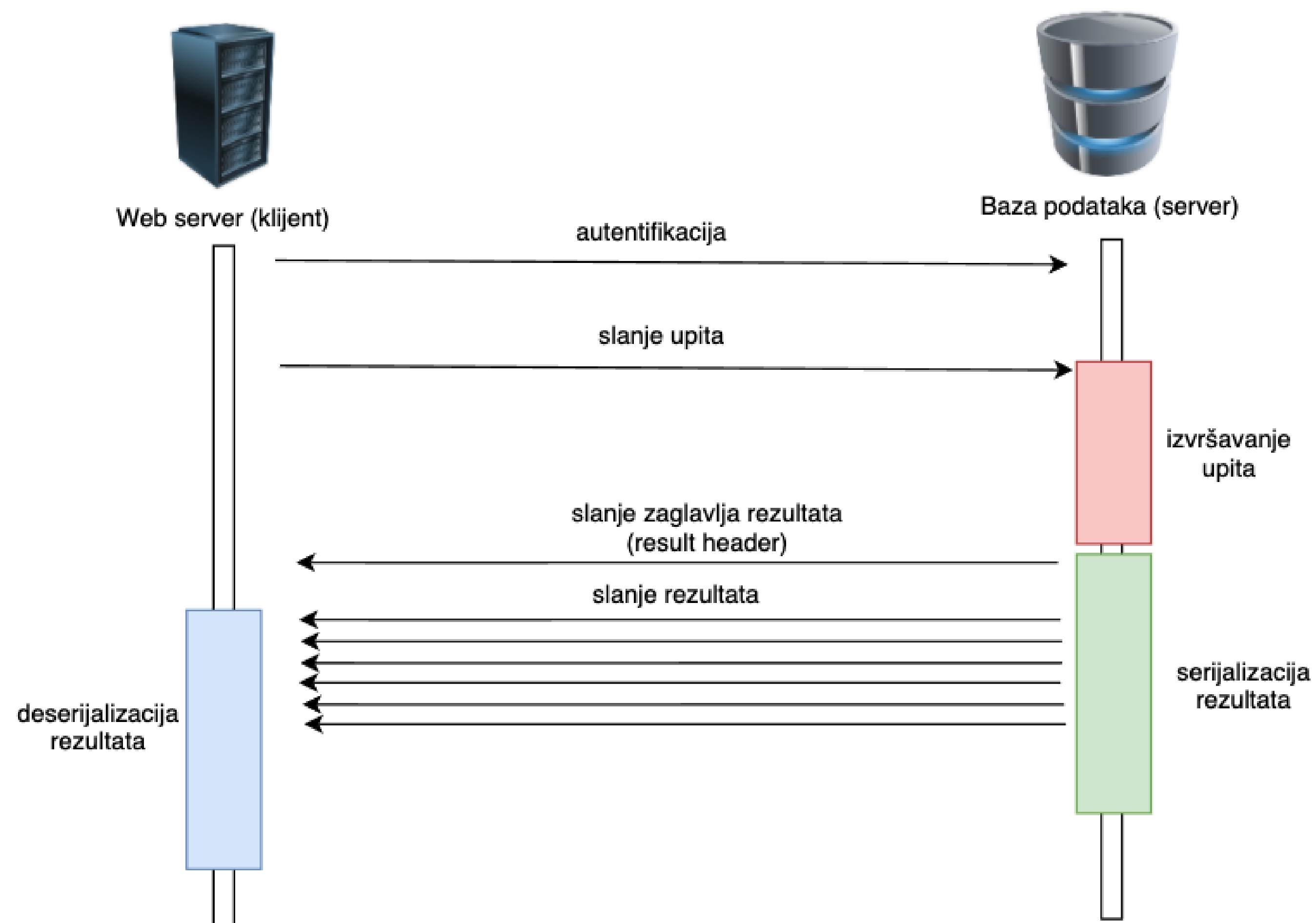
TRI NAJČEŠĆA SCENARIJA

- Serverska aplikacija <-> Baza podataka
- Direktna komunikacija klijent <-> server kroz poziv servisa
- Asinhrona komunikacija razmenom poruka preko reda poruka (message queue)





◆ APLIKACIJE PRISTUPAJU BAZI PODATAKA (SUBP, DBMS) KROZ API





◆ **APLIKACIJE PRISTUPAJU BAZI PODATAKA (SUBP, DBMS) KROZ API**

- Direktan pristup (specifičan za konkretan DBMS)
- Open Database Connectivity (ODBC)
- Java Database Connectivity (JDBC)

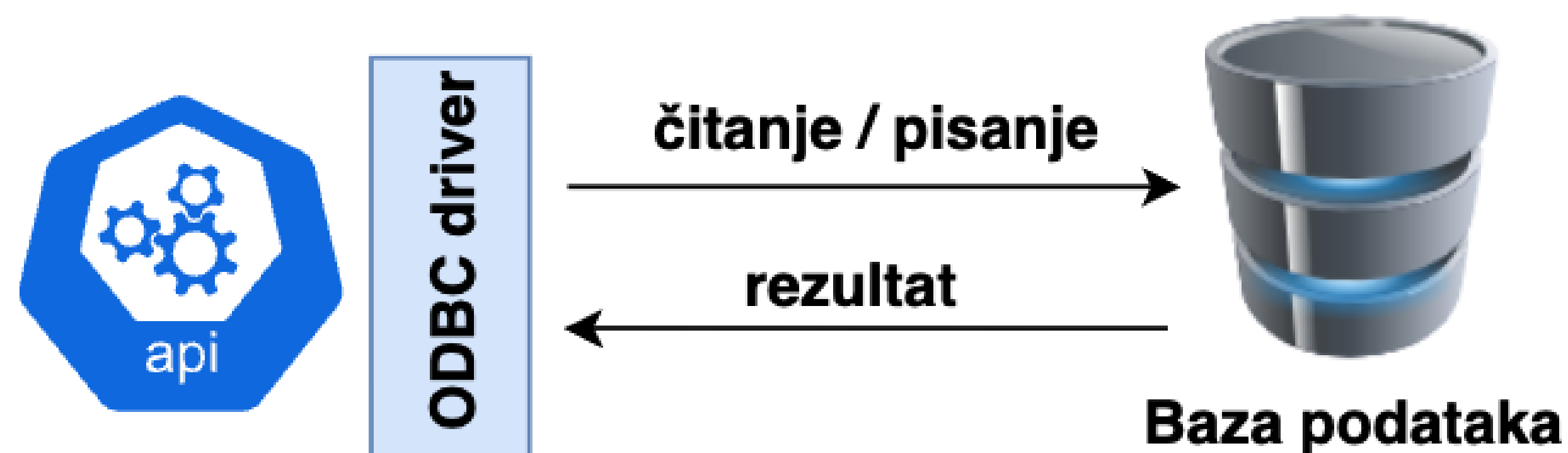


◆ ŠTA JE ODBC?

- Predstavlja standardni API za pristup DBMS
- Dizajniran da bude nezavisan od DBMS i OS
- Microsoft i Simba Technologies su ga razvili ranih 1990ih
- Svaki ozbiljniji DBMS ima ODBC implementaciju

◆ ŠTA JE ODBC?

- ODBC se bazira na „device driver“¹ modelu
- Drajver enkapsulira logiku koja je potrebna za konverziju standardnog skupa komandi u DBMS specifične pozive



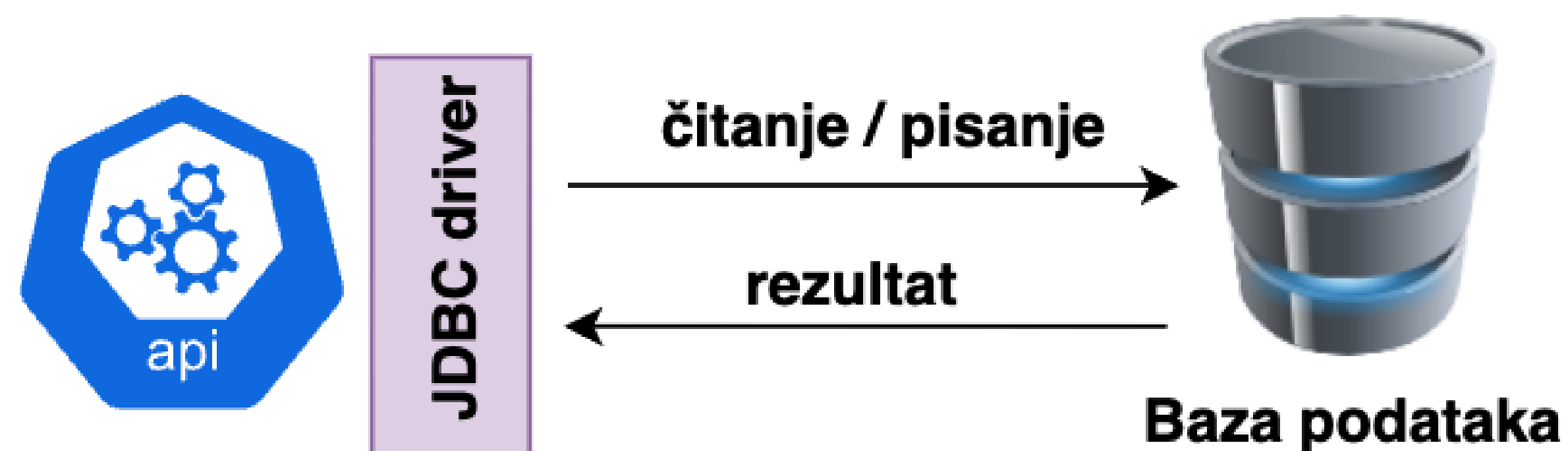


JAVA DATABASE CONNECTIVITY

8

◆ ŠTA JE JDBC?

- Sun Microsystems ga je razvio 1997.
- Pruža standardni API za konekciju Java programa sa DBMS
- Možemo ga posmatrati kao verziju ODBC pisanu u Javi umesto u C programskom jeziku





◆ PRISTUPI ZA IMPLEMENTACIJU

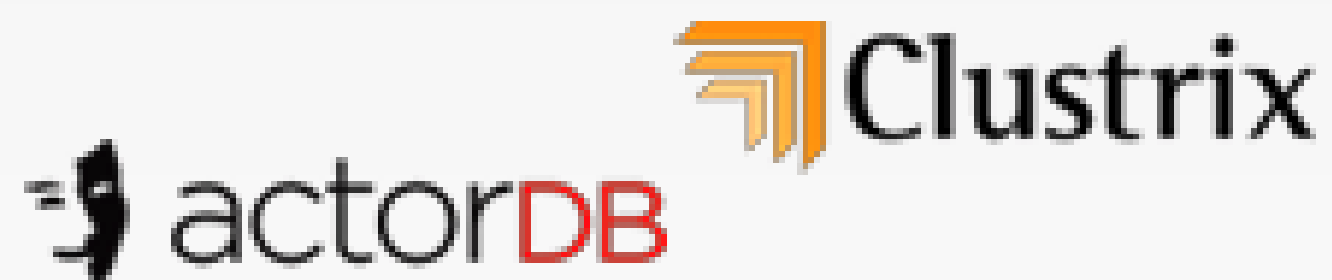
- JDBC-ODBC Bridge (Type 1)
 - Konvertuje JDBC poziv metode u ODBC poziv funkcije (od JDK 1.8 nije podržan pristup)
- Native-API Driver (Type 2)
 - Konvertuje JDBC poziv metode u nativni poziv DBMS API
- Network-Protocol Driver (Type 3)
 - Drajver se konektuje na *middleware* koji konvertuje JDBC poziv u DBMS specifični poziv
- Database-Protocol Driver/Thin Driver (Type 4)
 - Najbrži pristup jer je čista Java implementacija koja konvertuje JDBC poziv u DBMS specifični



◆ PRISTUPI ZA IMPLEMENTACIJU

- Svi veći proizvođači DBMS implementiraju svoje mrežne protokole preko TCP/IP
- Noviji DBMS implementiraju neke od open-source DBMS mrežnih protokola
- To im omogućava da koriste postojeće drajvere bez potrebe da razvijaju i održavaju svoje

EXISTING PROTOCOLS



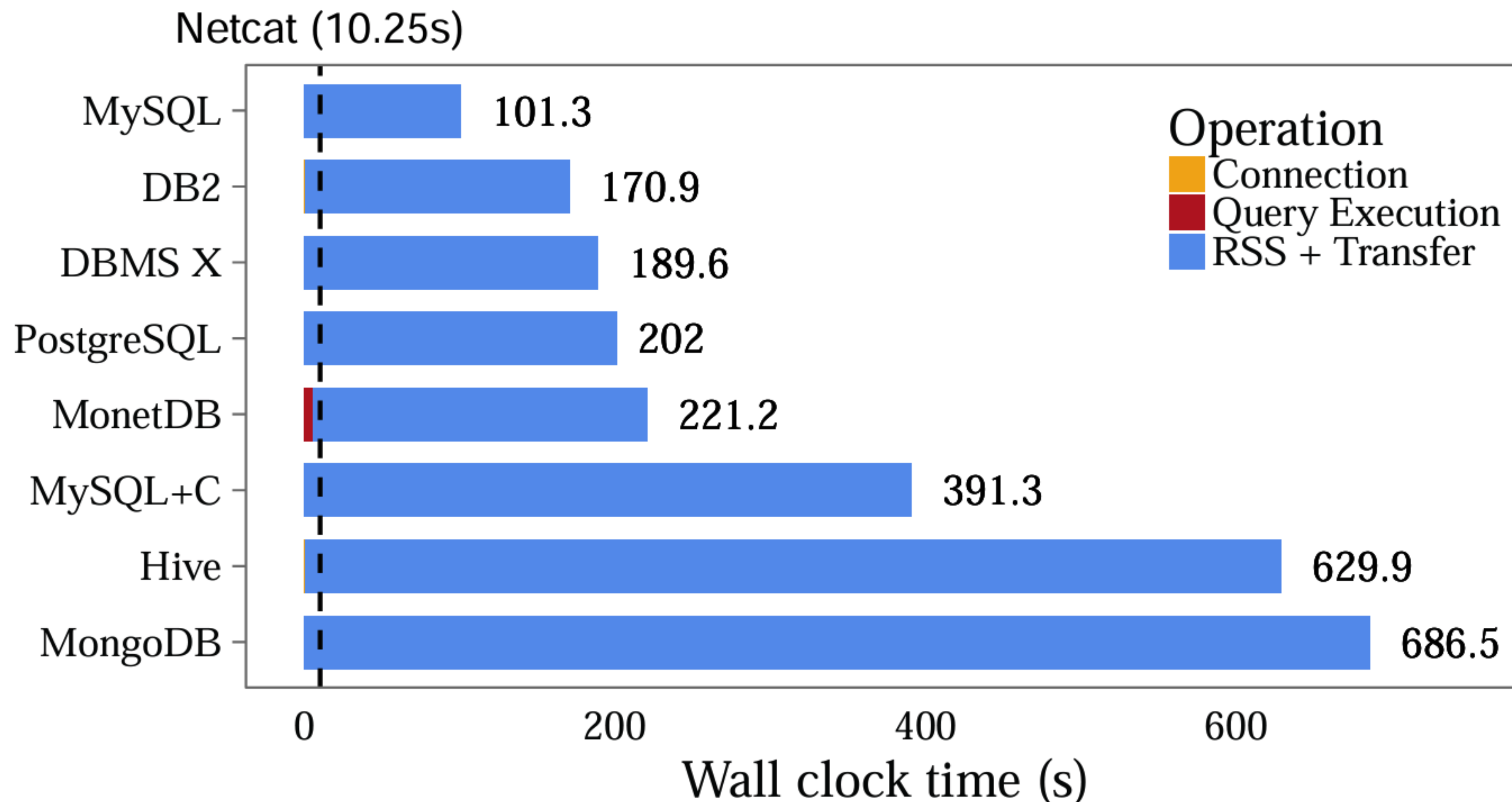


◆ PRISTUPI ZA OPTIMIZACIJU

- Row Layout problem
 - ODBC/JDBC su API-ji orijentisani ka redovima iz tabela gde server pakuje torke u poruke jednu po jednu, red po red i klijent deserijalizuje podatke red po red
 - Potencijalno rešenje – slati podatke u vektorima (u *batch*-u, *column layout*)
- Kompresija podataka pre slanja klijentu (Snappy, Zstd, Zlib, ...)
- Serijalizacija podataka
 - Binarni format (protobuf, flatbuffers, Thrift, ...)
 - Tekstualni format
- Upravljanje stringovima
 - Null termination – dodavanje "\0" na kraj
 - Prefiks dužine - dodavanje dužine stringa na početku bajtova koji se prenose
 - Fiksna dužina - za sve stringove se radi *padding* da se dostigne maksimalna veličina podatka

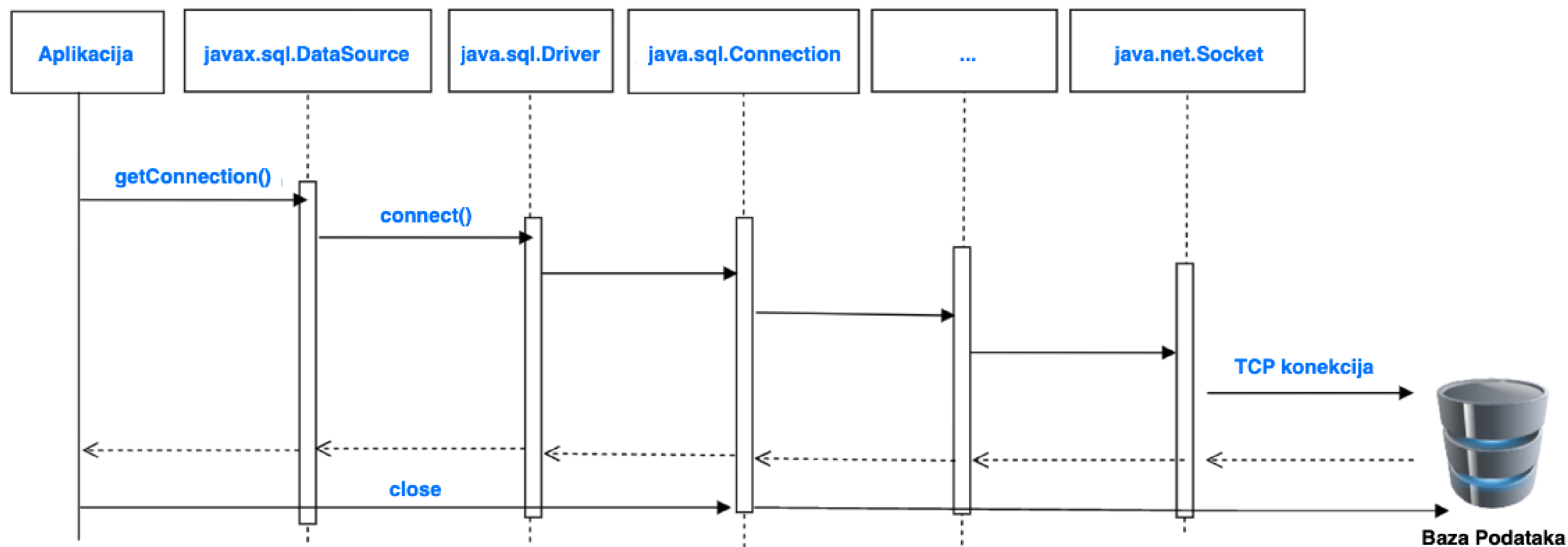


PROTOKOLI ZA RAD ZA BAZOM



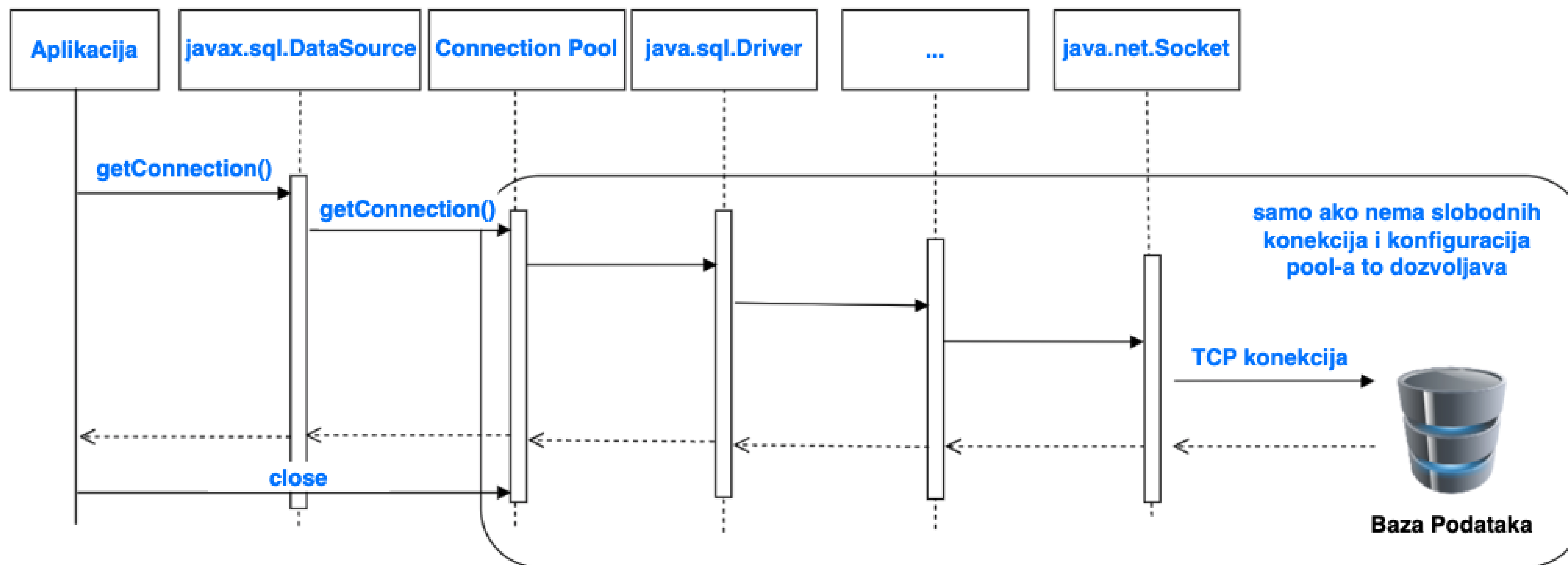
- Vreme da se izvrši **SELECT * FROM lineitem**¹ preko ODBC konektora iz različitih baza podataka gde su i klijent i server (baza) na istoj mašini
- Isprekidana linija je vreme za koje **netcat** prebaci preko soketa istu količinu podataka u CSV formatu

◆ KAKO SE OSTVARUJE KONEKCIJA SA BAZOM?

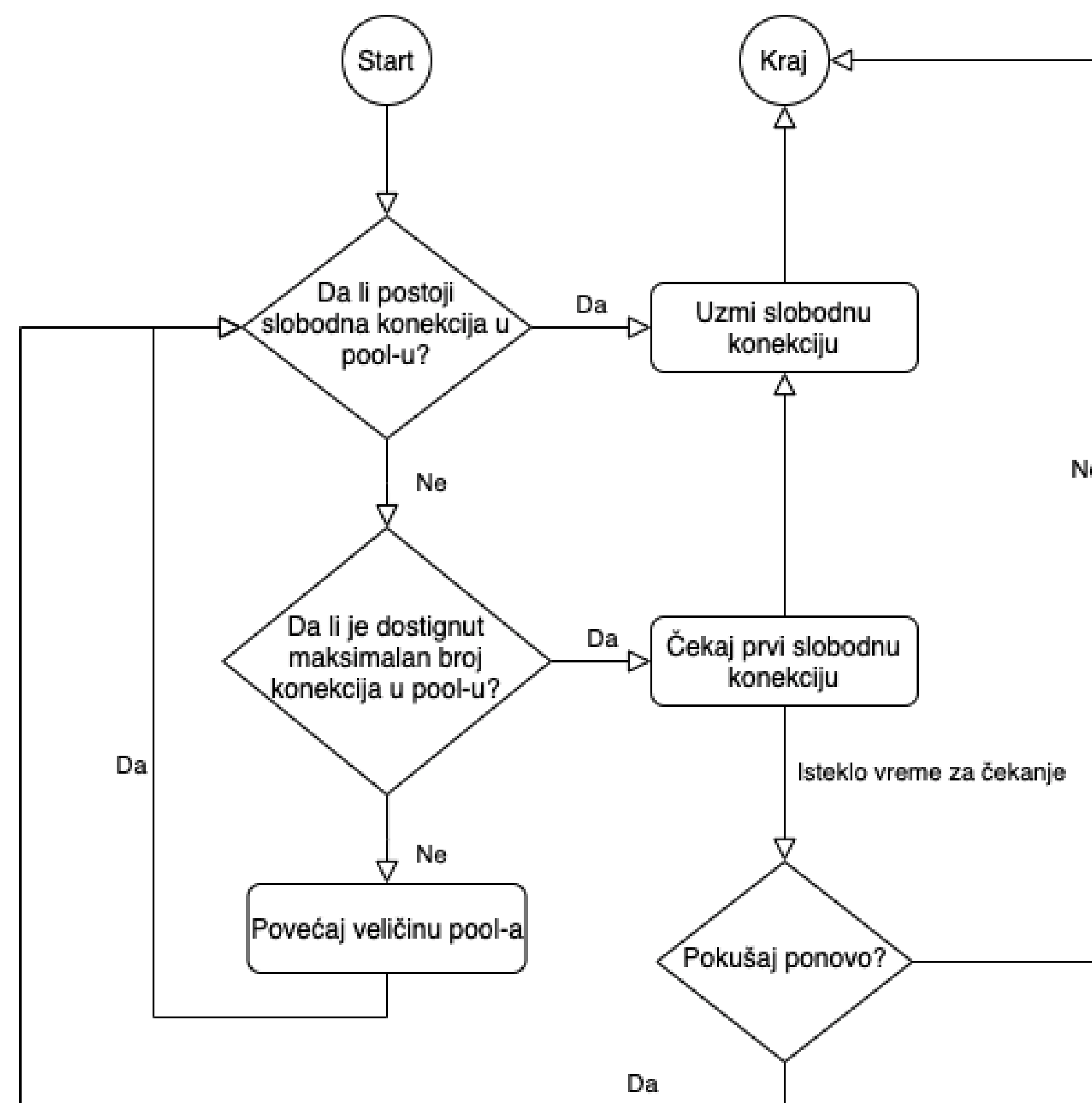


◆ KAKO DA SMANJIMO I/O OVERHEAD IZMEĐU APLIKACIJE I DBMS?

- Korišćenjem connection pool-a koji predstavlja keš za konekcije koje se mogu iznova koristiti bez potrebe kreiranja novih



◆ KAKO IZGLEDA LOGIKA PRIBAVLJANJA KONEKCIJE?





CONNECTION POOL

17

- ◆ **ŠTA SU USKA GRLA RADA SA BAZOM PODATAKA?¹**
- CPU
 - Disk
 - Mrežni protok
 - Memorija (pravi nekoliko redova veličine manju razliku u odnosu na prethodna 3)

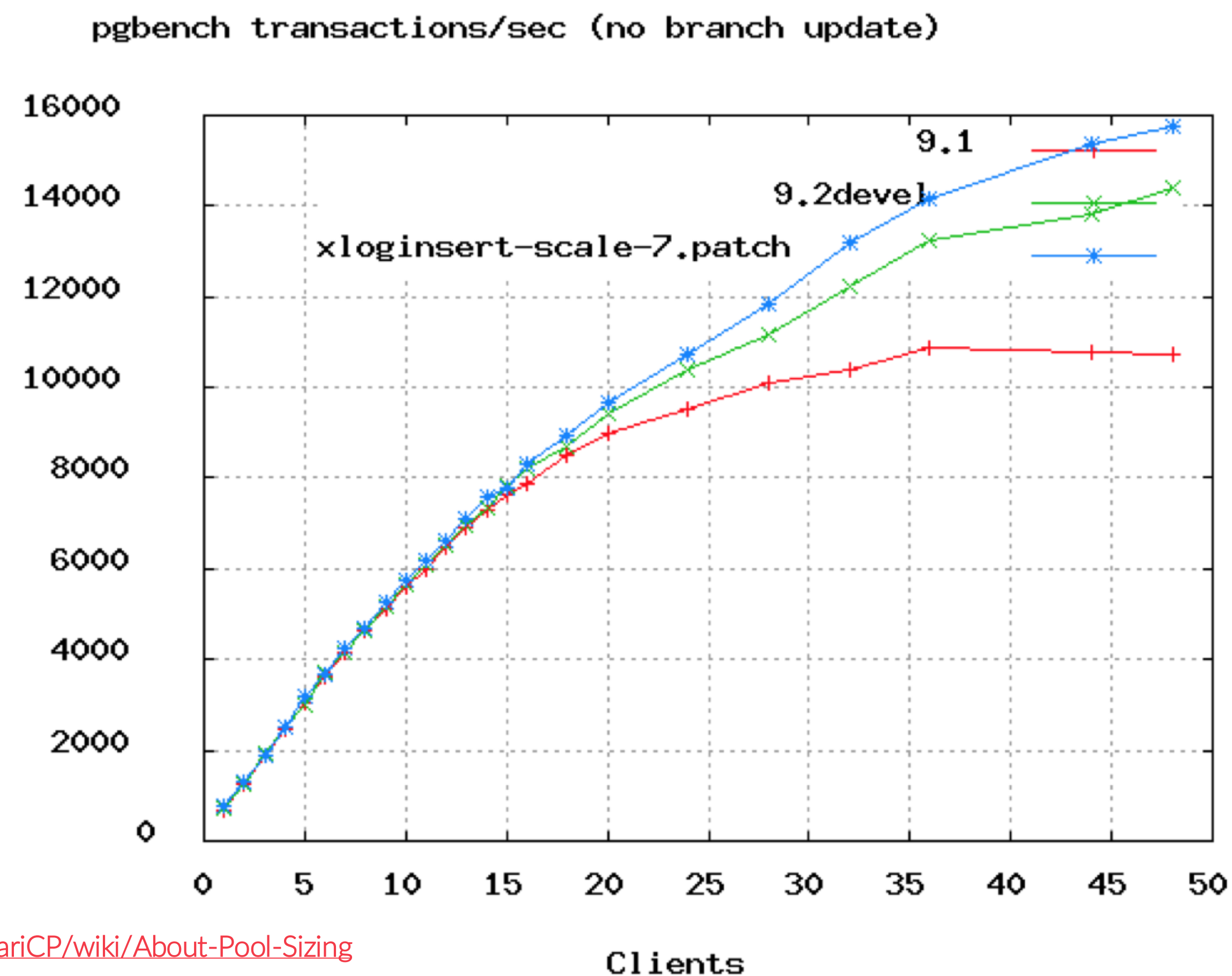
¹ <https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing>



CONNECTION POOL

18

◆ ŠTA SU USKA GRLA RADA SA BAZOM PODATAKA?





◆ POSTOJI LI IDEALNA FORMULA ZA BROJ KONEKCIJA?

$$\text{connections} = ((\text{core_count} * 2) + \text{effective_spindle_count})$$

*“Formula koja se prilično dobro držala u mnogim benčmarcima godinama jeste da za optimalnu propusnost broj aktivnih konekcija treba da bude negde blizu $((\text{broj_jezgara} * 2) + \text{efektivni_broj_hdd_osovina})$. Broj jezgara ne bi trebalo da uključuje HT niti, čak i ako je hyperthreading omogućen. Efektivni broj HDD osovina je nula ako je aktivni skup podataka u potpunosti keširan i približava se stvarnom broju osovina kako stopa pogodaka keša pada. ... Do sada nije bilo nikakvih analiza koliko dobro formula funkcioniše sa SSD diskovima.”¹*

¹ https://wiki.postgresql.org/wiki/Number_Of_Database_Connections



CONNECTION POOL

20

◆ POPULARNE IMPLEMENTACIJE

- Hikari-CP¹
- C3P0²
- Tomcat-JDBC³
- Apache common DBCP⁴

1 <https://github.com/brettwooldridge/HikariCP>

2 <https://www.mchange.com/projects/c3p0/>

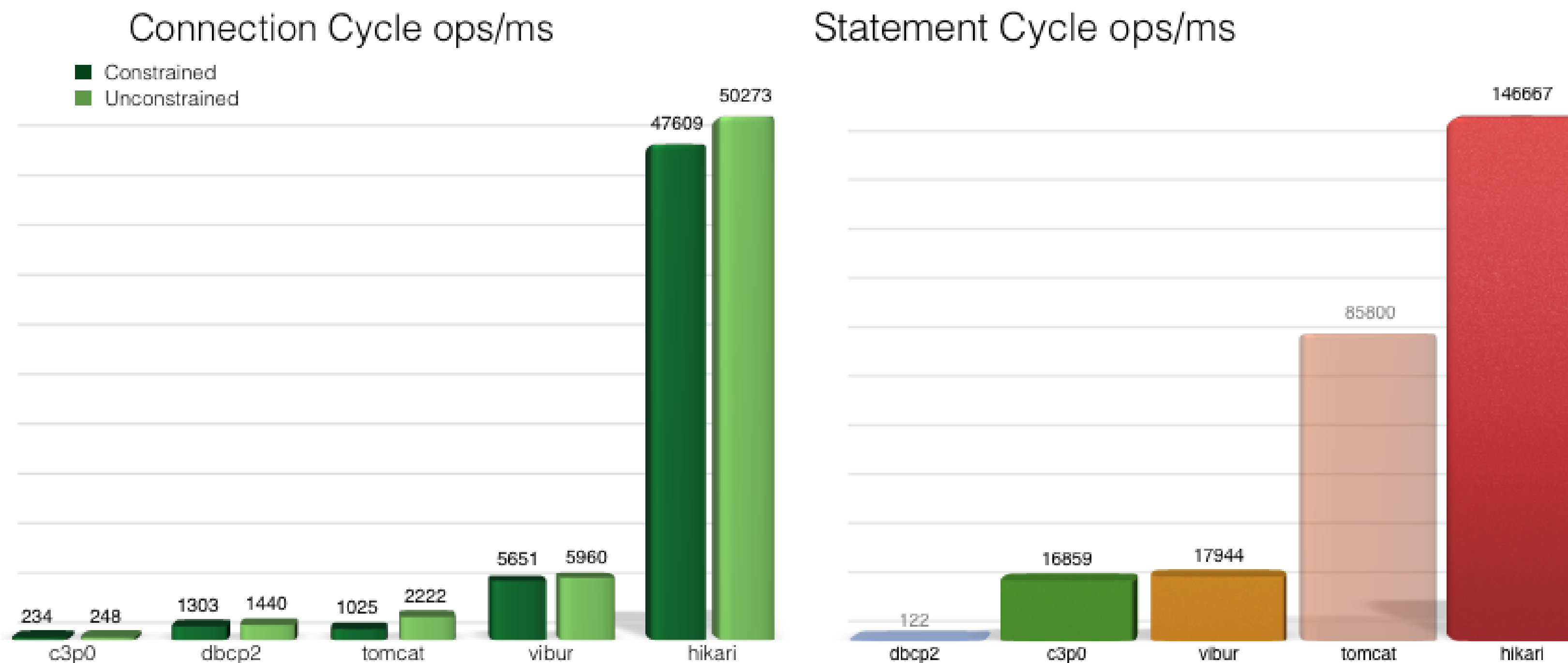
3 <http://tomcat.apache.org/tomcat-7.0-doc/jdbc-pool.html>

4 <http://commons.apache.org/proper/commons-dbcp/>



CONNECTION POOL

◆ KAKO SE POREDE POPULARNE IMPLEMENTACIJE?



- Connection Cycle predstavlja jednu konekciju od `DataSource.getConnection()` do `Connection.close()`
- Statement Cycle predstavlja jednu operaciju `Connection.prepareStatement()`, `Statement.execute()`, `Statement.close()`



REFERENCE

22

- ◆ **PRIMERI PO UZORU NA** <https://github.com/mbranko/isa19/tree/master/06-pooling>
- ◆ **PAVLO A., CMU. ADVANCED DATABASE SYSTEMS – NETWORKING**
<https://15721.courses.cs.cmu.edu/spring2020/slides/11-networking.pdf>
- ◆ **RAASVELDT M., MUHLEISEN H., DON'T HOLD MY DATA HOSTAGE – A CASE FOR CLIENT PROTOCOL REDESIGN**
<https://15721.courses.cs.cmu.edu/spring2020/papers/11-networking/p1022-muehleisen.pdf>
- ◆ **MIHALCHEA V. THE ANATOMY OF CONNECTION POOLING**
<https://vladmihalcea.com/the-anatomy-of-connection-pooling/>
- ◆ **HIKARI-CP. ABOUT POOL SIZING** <https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing>

**KOJA SU VAŠA
PITANJA?**