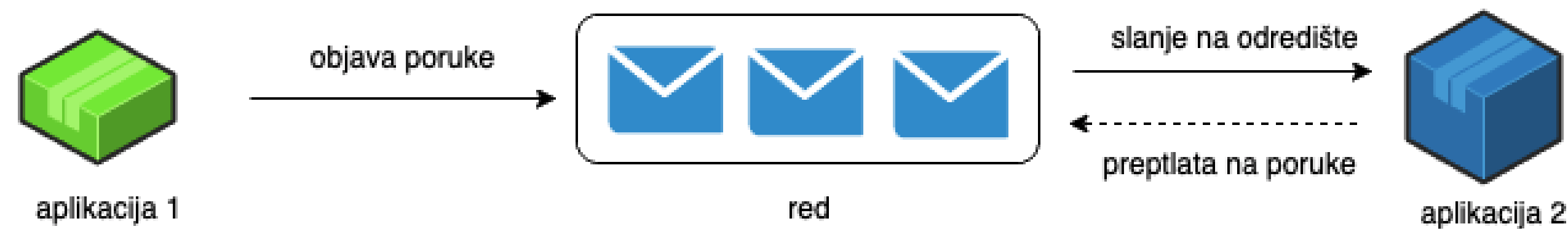
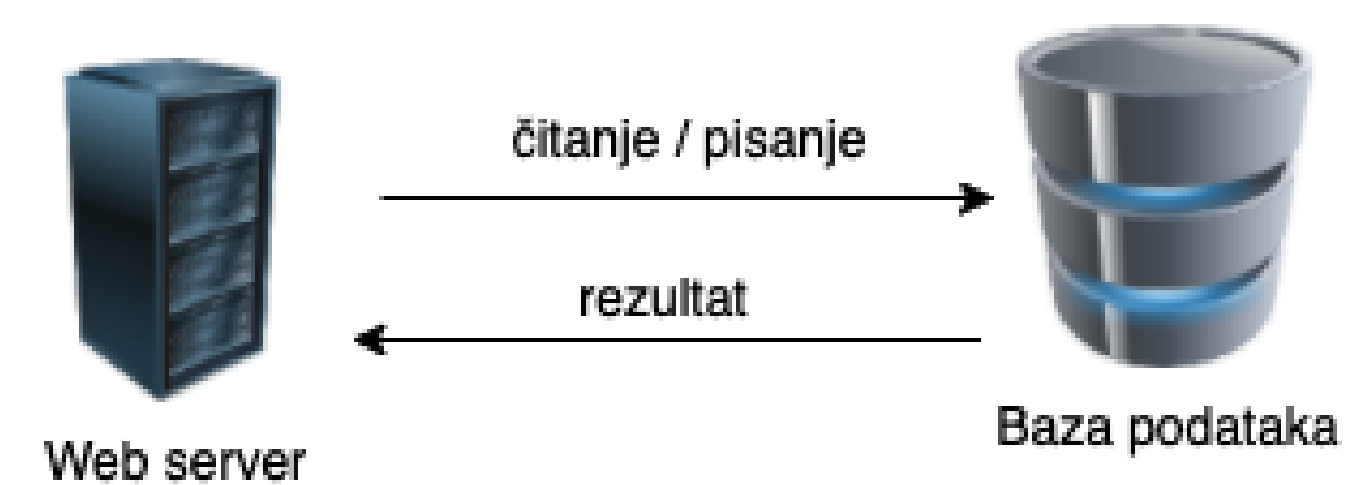
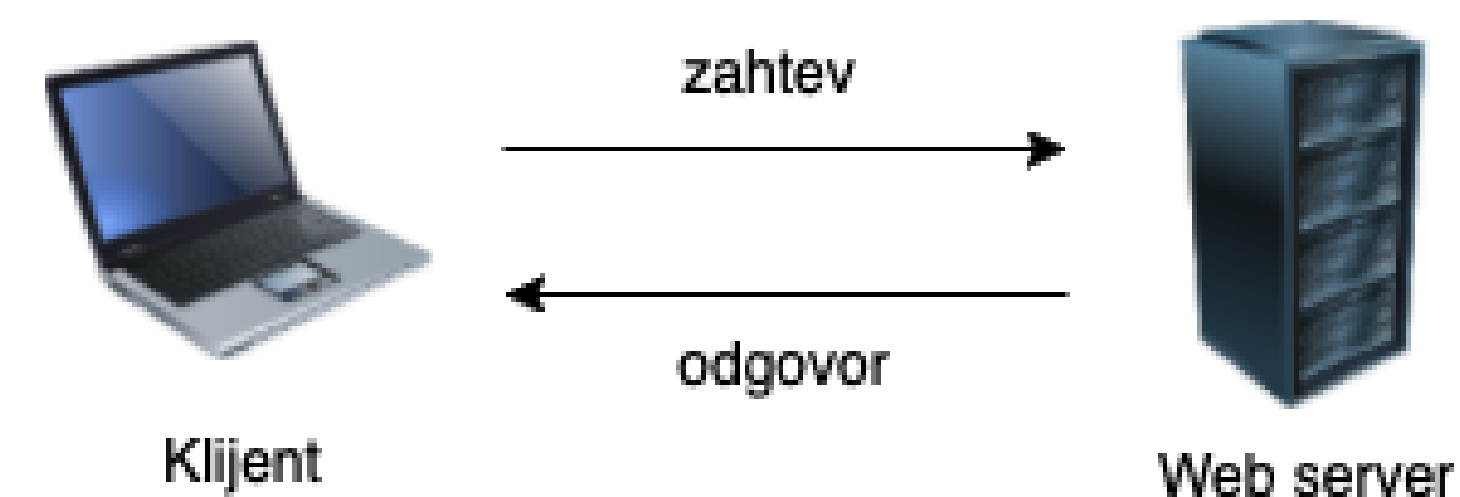


# **ASINHRONO PROCESIRANJE PORUKA**

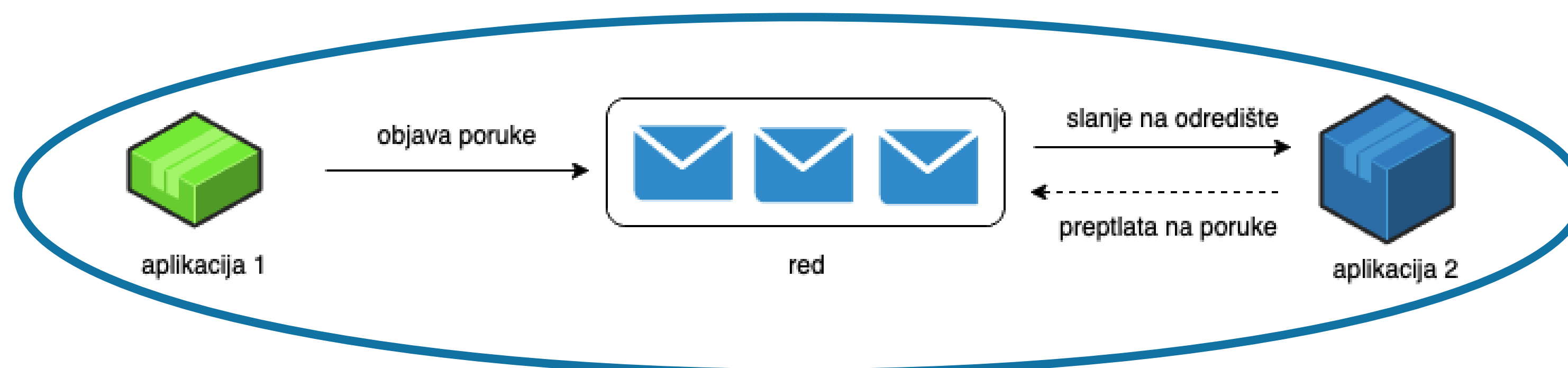
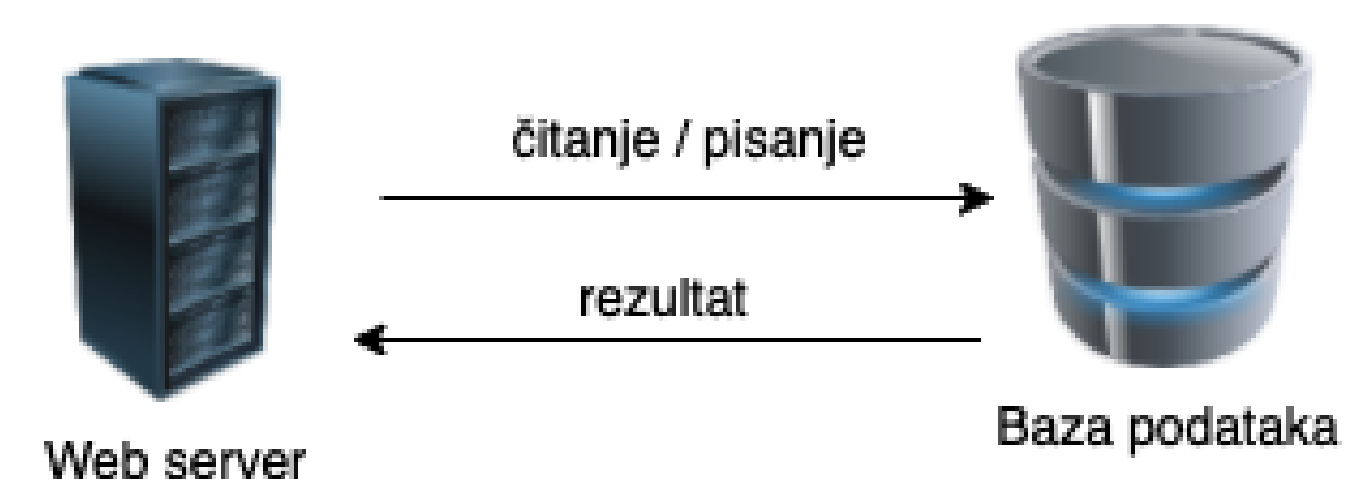
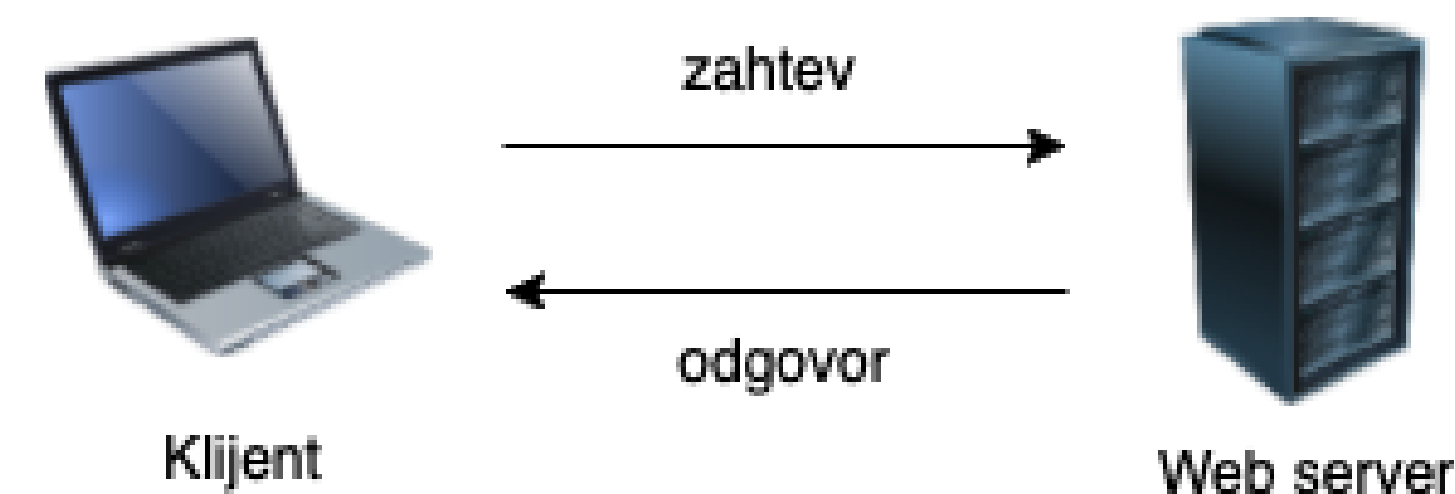
## TRI NAJČEŠĆA SCENARIJA

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



## TRI NAJČEŠĆA SCENARIJA

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





# KAKO APLIKACIJE KOMUNICIRAJU?

4

## ◆ POTREBNE KOMPONENTE

- Interfejs
  - Protokol
  - Format poruke

## ◆ OBEZBEĐENA NEZAVISNOST SISTEMA

- Dok god se dva sistema dogovore oko oblika poruka koje razmenjuju moći će da komuniciraju međusobno bez obzira kako su sami sistemi implementirani (programski jezik, framework,...)



# KAKO APLIKACIJE KOMUNICIRAJU?

5

## ◆ **SINHRONO PROCESIRANJE**

- Tradicionalni pristup rada softvera gde klijent (nit, proces, aplikacija,...) šalje zahtev za izvršavanje i čeka odgovor pre nego što nastavi sa daljim radom
- Klijent zavisi od rezultata operacije i obično ne može da nastavi sa daljim radom

## ◆ **ASINHRONO PROCESIRANJE**

- Klijent šalje zahtev i procesiranje se nastavlja bez daljeg blokiranja klijenta



# PROIZVOĐAČI PORUKA (MESSAGE PRODUCERS)

6

## ◆ MESSAGE PRODUCERS/PUBLISHERS

- Proizvođači poruka su komponente koje iniciraju asinhrono procesiranje
- Nemaju puno odgovornosti:
  - Da formiraju validnu poruku (ispravan format, obavezna polja, korektni podaci)
  - Da je objave (publish) na odgovarajući kanal/temu/red
- Proizvođači ne znaju ko će poruku obraditi niti kako će se ona obraditi
  - Time se postiže slaba sprega (loose coupling)
- Jedna aplikacija može imati više proizvođača poruka, npr. po funkcionalnim celinama (nalozi, plaćanja, notifikacije, audit logovi)
- Tipično ne sadrže poslovnu logiku obrade, već samo iniciranje događaja ili komandi



# BROKERI PORUKA (MESSAGE BROKERS)

7

## ◆ MESSAGE BROKERS

- Predstavljaju specijalizovane aplikacije namenjene za brzo i fleksibilno prihvatanje, rutiranje i isporuku poruka
- Najčešće se implementiraju kao nezavisne aplikacije jer mogu da imaju dodatnih odgovornosti poput kontrole pristupa, oporavka u slučaju grešaka, itd.
- Obično je dovoljno brokere samo konfigurisati, a ne proširivati implementaciju dodatnim kodom
- Osnovna (centralna) komponenta brokera je red (queue):
  - Poruke se u njega smeštaju od strane proizvođača
  - Poruke se iz njega preuzimaju od strane potrošača
  - Red omogućava razdvajanje brzine proizvodnje i obrade poruka





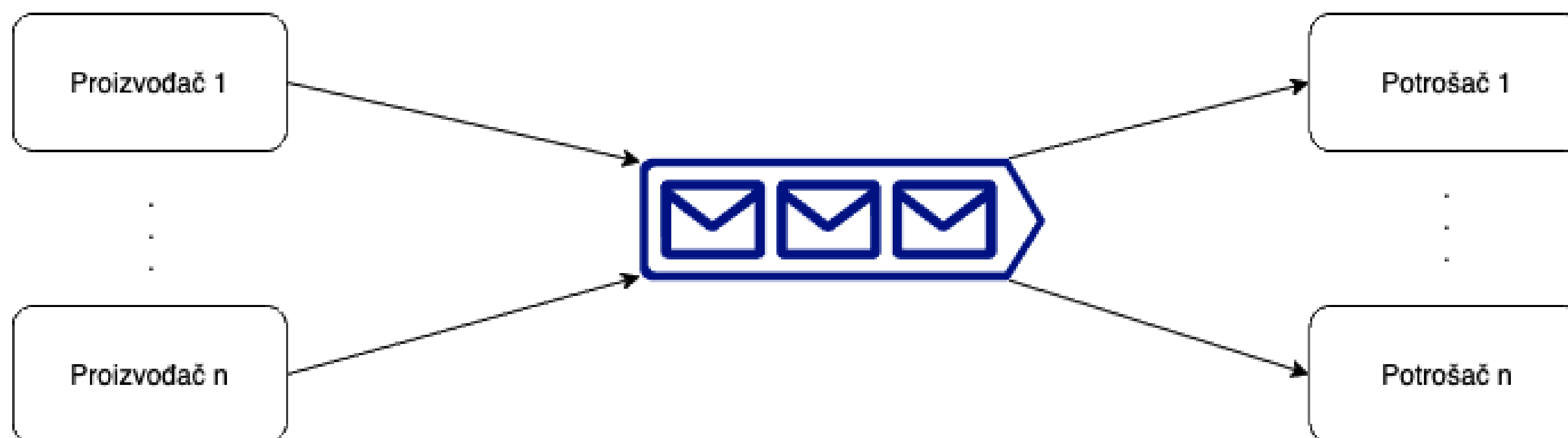
## ◆ MESSAGE CONSUMERS/SUBSCRIBERS

- Potrošači/primaoci poruka su komponente koje prihvataju i asinhrono procesiraju poruke iz reda
- Obično se nalaze na odvojenim serverima od proizvođača poruka kako bi bilo lakše skaliranje
- Dva najčešća pristupa implementaciji potrošača poruka su:
  - *Cron like, pull model* – potrošači se periodično konektuju na red, proveravaju status i konzumiraju poruke dok god ih ima u redu
    - Pogodno za jednostavne sisteme, manju frekvenciju poruka, situacije gde nije potrebna trenutna obrada
  - *Daemon like, push model* – potrošači su permanentno konektovani sa brokerom i čekaju da broker pošalje poruku njima
    - Pogodno za *real-time* ili *near-real-time* obradu, veće opterećenje, sisteme sa zahtevima za niskom latencijom



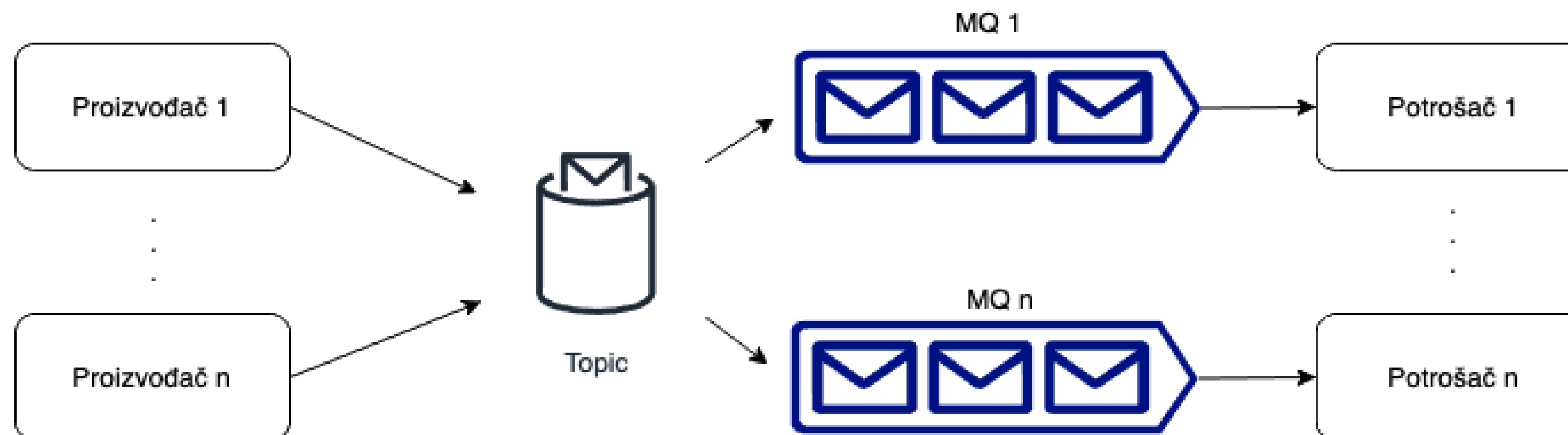
## ◆ DIREKTNE PORUKE (POINT-TO-POINT)

- Proizvođači i potrošači poruka moraju da znaju samo naziv reda
- Svaka poruka koja dođe u red biće isporučena samo jednom potrošaču
- Ova metoda je pogodna za distribuciju zadataka koji dugo traju na više mašina (*worker-a*)
- Primeri korišćenja - slanje e-maila, procesiranje slika i video snimaka



## ◆ PUBLISH/SUBSCRIBE

- Proizvođači poruke šalju u *topic*, ne red
- Svaka poruka koja dođe u *topic* može biti isporučena proizvoljnom broju potrošača koji su pretplaćeni na poruke koje se tu objave
- Potrošači se pretplaćuju na poruke koje pristižu na određeni *topic*, kada poruka pristigne, kopira se u red koji je rezervisan samo za jednog potrošača (interna implementacija može biti različita u zavisnosti od proizvođača)
- Primer korišćenja – objava poruka o svakoj kupovini, o promeni akcija na berzi, distribucija informacija po geografskim lokacijama





## ◆ **NAJČEŠĆE KORIŠĆENI PROTOKOLI**

- Protokoli definišu kako se klijentska biblioteka povezuje sa brokerom i kako se poruke prenose
  - AMQP (Advanced Message Queuing Protocol)
  - STOMP (Streaming Text-Oriented Messaging Protocol)
  - MQTT (Message Queuing Telemetry Transport)



## ◆ ASINHRONO PROCESIRANJE

- Uz pomoć MQ možemo odblokirati klijenta i odložiti procesiranje vremenski zahtevnih zadataka
- Moguće primene:
  - Komunikacija sa udaljenim serverima
  - Zadaci za čiju obradu su potrebni veći resursi od onih kojima raspolaže klijent
  - Sklanjanje zadataka niskog prioriteta sa kritične putanje sistema
  - Nezavisno procesiranje zadataka različitih prioriteta



# PREDNOSTI KORIŠĆENJA MQ

14

## ◆ LAKŠE SKALIRANJE

- Aplikacije koje koriste MQ lakše se mogu skalirati jer su proizvođači nezavisni od potrošača
- Za kreiranje “skupih” poruka možemo imati više proizvođača poruka koji će ih slati brokerima u paraleli
- Dodavanjem više potrošača možemo obraditi više poruka u zadatom vremenskom periodu
- Skaliranje proizvođača i potrošača ne mora uopšte da utiče na konfiguraciju brokera



## ◆ NEOMETAN RAD PRI POVEĆANOM SAOBRAĆAJU

- U slučaju povećanog saobraćaja na aplikaciji, broker može da nastavi da prihvata zahteve
- Iako proizvođač kreira poruke mnogo brže nego što ih potrošač obrađuje, poruke se čuvaju u redu i proizvođač ne mora biti pogođen povećanim brojem zahteva
- Krajnji korisnik neće videti razliku, jer može dobiti instant odgovor da će zahtev biti obrađen (iako se stvarna obrada neće još neko vreme obaviti)
- Na ovaj način je povećana dostupnost sistema



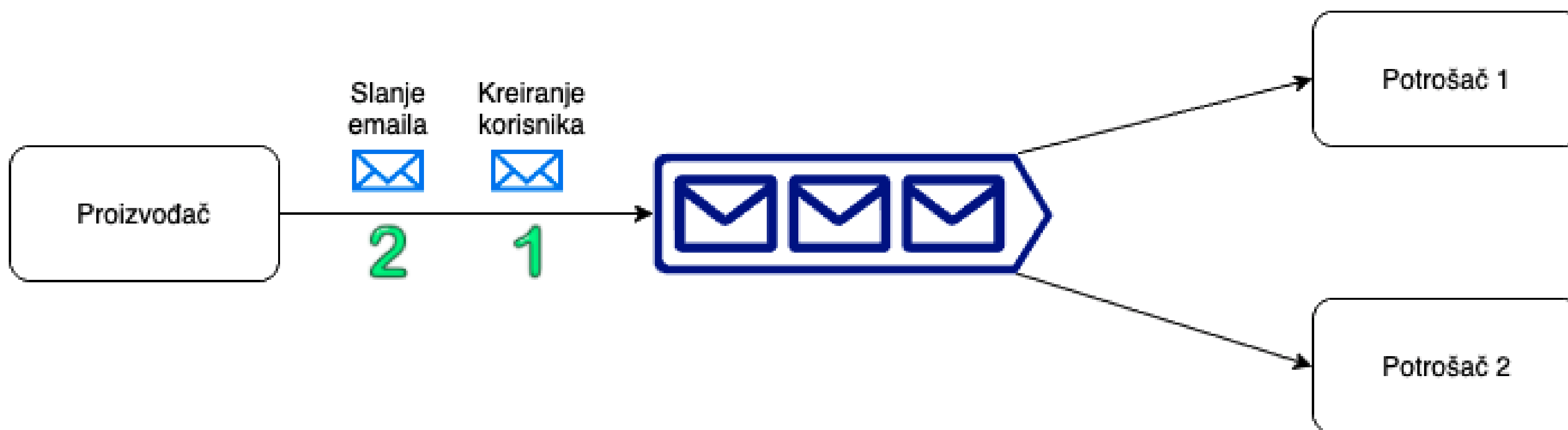
## ◆ ODVAJANJE PROIZVOĐAČA OD POTROŠAČA (DECOUPLING)

- Prema definiciji arhitekture MQ i uvođenja broker komponente, proizvođači i potrošači se razdvajaju
- Krajnji cilj je da postignemo potpunu nezavisnost proizvođača od potrošača tako što ne moraju znati jedan za drugog već samo za broker kome trebaju da se obrate
- Na ovaj način ako dođe do greške pri kreiranju ili obradi poruke druga strana ne mora da brine o problemima koje ima prva strana i obrnuto



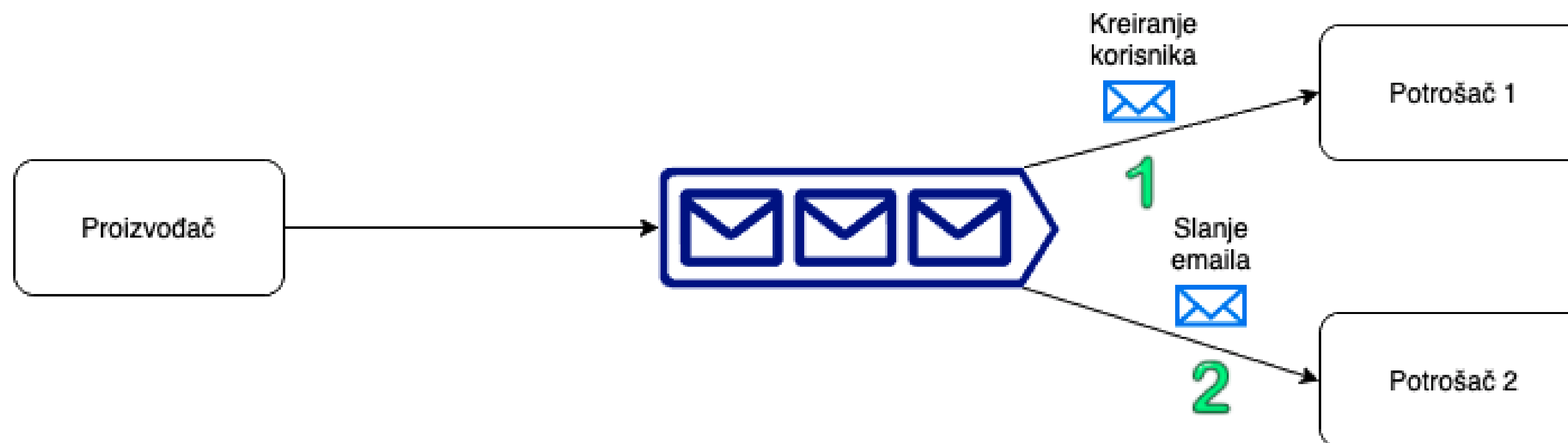
## ◆ NEPOSTOJANJE REDOSLEDA PORUKA

- Redosled poruka nije garantovan jer se poruke procesiraju u paraleli i ne postoji sinhronizacija između potrošača poruka
- Svaki potrošač radi sa jednom porukom u jednom trenutku i nema znanje o drugim potrošačima koji paralelno obrađuju svoje poruke
- Pošto potrošači rade u paraleli, bar jedan može da postane spor ili da se sruši i dosta je teško obezbediti da se poruke dostavljaju u ispravnom redosledu



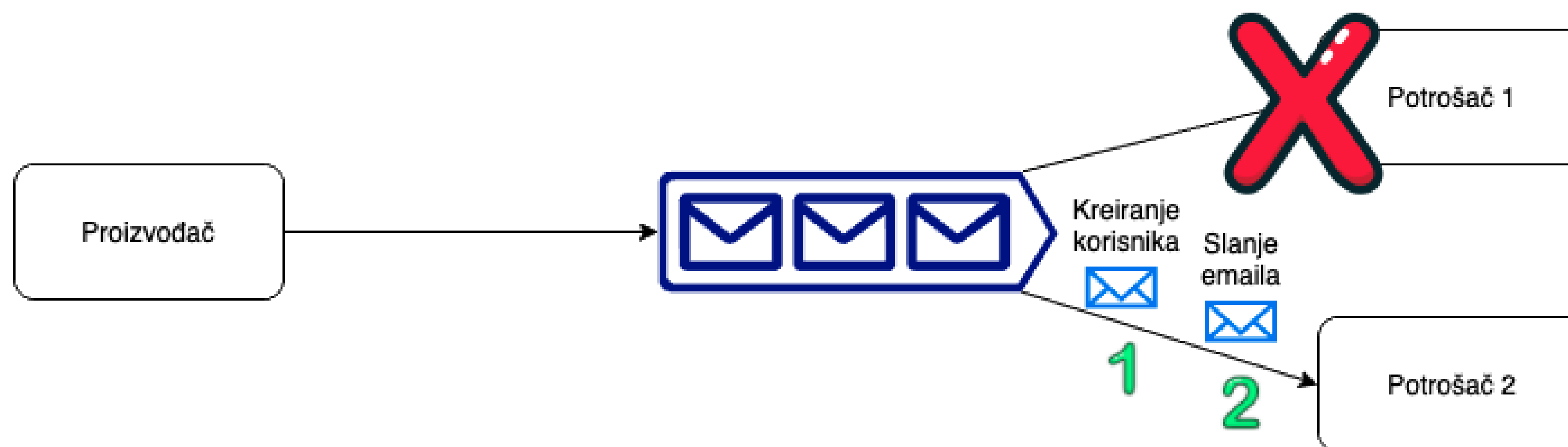
## ◆ NEPOSTOJANJE REDOSLEDA PORUKA

- Redosled poruka nije garantovan jer se poruke procesiraju u paraleli i ne postoji sinhronizacija između potrošača poruka
- Svaki potrošač radi sa jednom porukom u jednom trenutku i nema znanje o drugim potrošačima koji paralelno obrađuju svoje poruke
- Pošto potrošači rade u paraleli, bar jedan može da postane spor ili da se sruši i dosta je teško obezbediti da se poruke dostavljaju u ispravnom redosledu



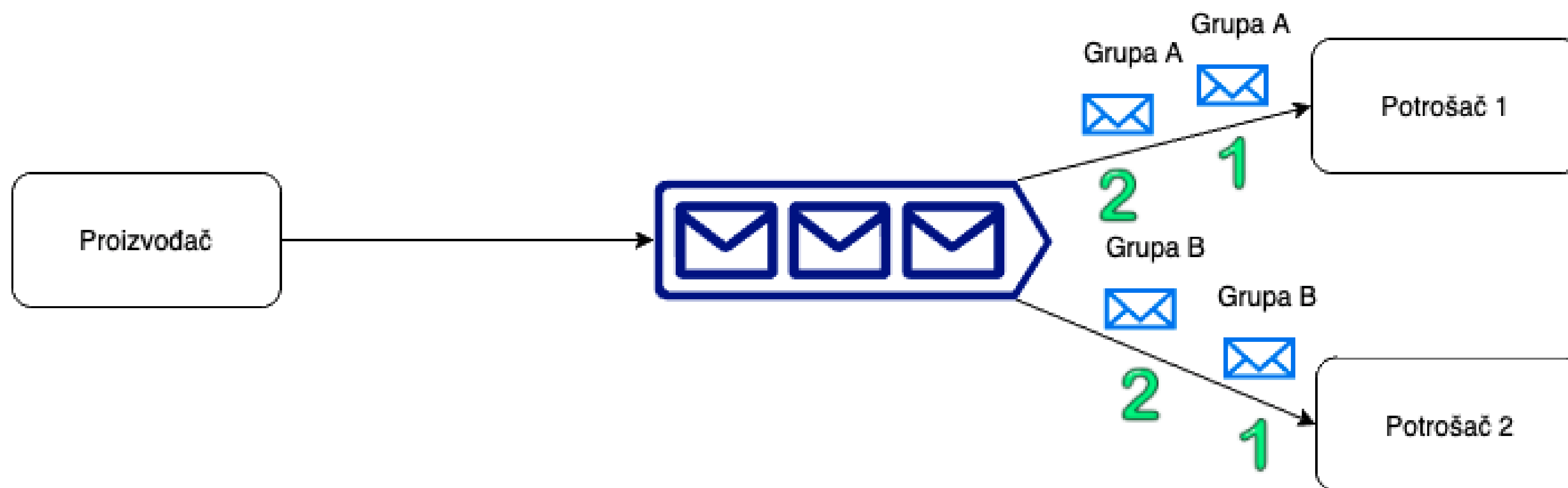
## ◆ NEPOSTOJANJE REDOSLEDA PORUKA

- Redosled poruka nije garantovan jer se poruke procesiraju u paraleli i ne postoji sinhronizacija između potrošača poruka
- Svaki potrošač radi sa jednom porukom u jednom trenutku i nema znanje o drugim potrošačima koji paralelno obrađuju svoje poruke
- Pošto potrošači rade u paraleli, bar jedan može da postane spor ili da se sruši i dosta je teško obezbediti da se poruke dostavljaju u ispravnom redosledu



## ◆ REŠENJA

- Limitiranje broja potrošača na jednu nit po redu – nije skalabilno rešenje
- Pravljenje sistema pod pretpostavkom da će poruke uvek pristizati nasumičnim redosledom
- Upotreba brokera poruka koji garantuje isporuku poruka u parcijalnom redosledu (partial message ordering)





## ◆ **PONOVNO VRAĆANJE PORUKA U RED**

- Ako dođe do greške pri slanju poruke, ista se može ponovo poslati
- Na ovaj način sistem postaje robusniji
- Da bi ovaj pristup uspeo, potrošači poruka moraju biti idempotentni (što može biti izazovno ostvariti)

## ◆ **POVEĆANA KOMPLEKSNOŠT SISTEMA**

- Razdvajanjem komponenti na proizvođače i potrošače poruka zarad veće autonomije može dovesti do povećanja kompleksnosti sistema
- Dokumentovanje svih promena i razloga za uvođenje dodatne kompleksnosti je izuzetno važno za dalje održavanje sistema



## ◆ TRETIRANJE MQ KAO DA JE TCP SOCKET

- Neki brokeri dozvoljavaju da se kreira povratni kanal kojim potrošač poruka vraća poruke proizvođaču
- Ako se previše često koristi, aplikacija postaje više sinhrona nego asinhrona
- Idealno bi bilo da uvek postoji jednosmerni kanal gde se poruke samo šalju (“pošalji i zaboravi”)
- Postojanje kanala za prihvatanje odgovora vodi ka strogom spajanju komponenti koje razmenjuju poruke i sprečavaju namenu asinhronog procesiranja





## TRETIRANJE MQ KAO BAZE PODATAKA

- MQ ne bi trebalo tretirati kao nadogradnju baze podataka
- Ne bi trebao da postoji nasumični pristup porukama u redu koje bi se brisale ili menjale jer to može dovesti do povećane složenosti sistema
- Najbolje je tretirati redove kao append-only FIFO

## ◆ ČVRSTA SPREGA IZMEĐU PROIZVOĐAČA I POTROŠAČA PORUKA

- Poruke ne treba da budu formulisane tako da se tačno zna koja komponenta će ih konzumirati
- Potrebno je tumačiti brokera kao krajnju tačku gde ta poruka ide i strukturu poruke kao ugovor koji treba da se ispoštuje
- Ako nešto nije eksplicitno navedeno u poruci, znači da je deo implementacije i ne bi trebalo potrošača poruka da dotiče
- Poruke ne treba da imaju logiku ili kod koji treba da se izvrši, već jednostavan niz bajtova koji mogu da kreiraju i očitaju obe strane nezvezano od tehnologije u kojima su implementirani



## ◆ NEDOSTATAK OBRADE NEVALIDNIH PORUKA

- Ne treba uvek pretpostavljati da su sve poruke validne
- "Poruke smrti" ili "otrovne poruke"<sup>1</sup> su poruke koje mogu da izazovu da se potrošač poruka ponaša nepredvidivo ili sruši
- Rešenje – *Dead Letter Queue*
  - Za poruke koje se šalju na red koji ne postoji
  - Za poruke koje prekoračuju dužinu reda u koji se šalju
  - Za poruke čija je veličina prekoračena
  - Za poruke koje su odbijene od drugog reda
  - Za poruke koje su odbijene za procesiranje određeni broj puta
  - Za poruke koje nisu obrađene određeno vreme
  - Za poruke koje nisu obrađene uspešno

<sup>1</sup> Poison message [https://en.wikipedia.org/wiki/Poison\\_message](https://en.wikipedia.org/wiki/Poison_message)



# REFERENCE

27

- ◆ **KORAB J. UNDERSTANDING MESSAGE BROKERS.**  
<https://www.oreilly.com/library/view/understanding-message-brokers/9781492049296/>
- ◆ **EJSMOJNT A. WEB SCALABILITY FOR STARTUP ENGINEERS.**  
<https://www.oreilly.com/library/view/web-scalability-for/9780071843669/>
- ◆ **RABBIT MQ. AMQP 0-9-1 MODEL EXPLAINED.**  
<https://www.rabbitmq.com/tutorials/amqp-concepts.html>
- ◆ **MICROSOFT. SERVICE BUS QUEUES, TOPICS, AND SUBSCRIPTIONS.**  
<https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions>

**KOJA SU VAŠA  
PITANJA?**