



SVEUČILIŠTE U ZAGREBU



Fakultet
elektrotehnike i
računarstva

Diplomski studij

**Informacijska i
komunikacijska tehnologija:**

Telekomunikacije i informatika

Računarstvo:

Programsko inženjerstvo i
informacijski sustavi

Računarska znanost

Raspodijeljeni sustavi

4. Procesi i komunikacija:
komunikacija porukama, model objavi-
pretplati, dijeljeni podatkovni prostor

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 licencom koja je ista ili slična ovoj.



U slučaju daljnjeg 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/>

„Neizravna” komunikacija

- engl. *indirect communication*
- komunikacija među procesima raspodijeljenog sustava putem posrednika bez direktne interakcije pošiljatelja i primatelja
- područja primjene
 - pokretne mreže i okoline
 - tokovi podataka (npr. financijski sustavi)
 - aplikacije u području Interneta stvari (senzori kontinuirano generiraju podatke)
- osigurava prostornu i vremensku neovisnost procesa (engl. *space uncoupling and time uncoupling*)

Sadržaj predavanja

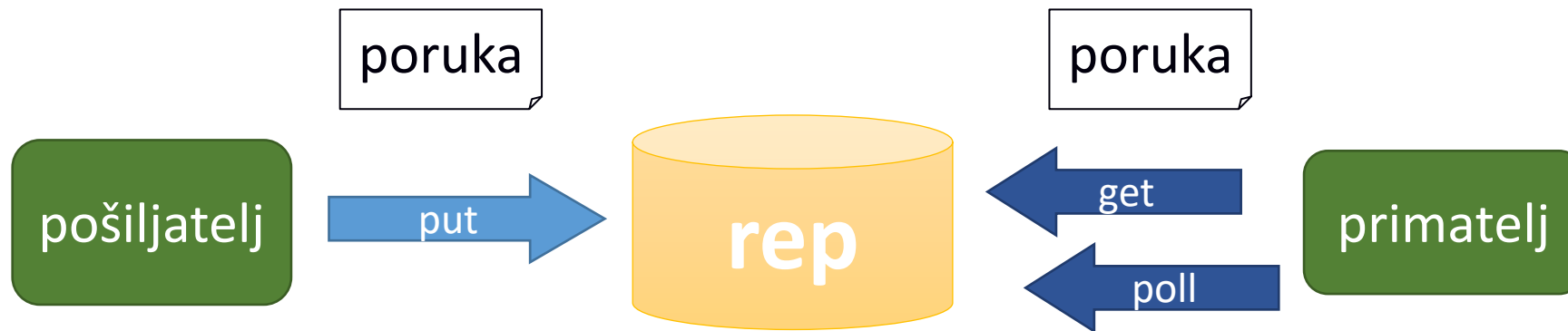
- **Komunikacija porukama**
- Model objavi-pretplati
- Primjeri protokola za komunikaciju porukama: JMS i AMQP
- Dijeljeni podatkovni prostor

Komunikacija porukama

engl. *message-queuing systems*, *Message-Oriented Middleware* (MOM)

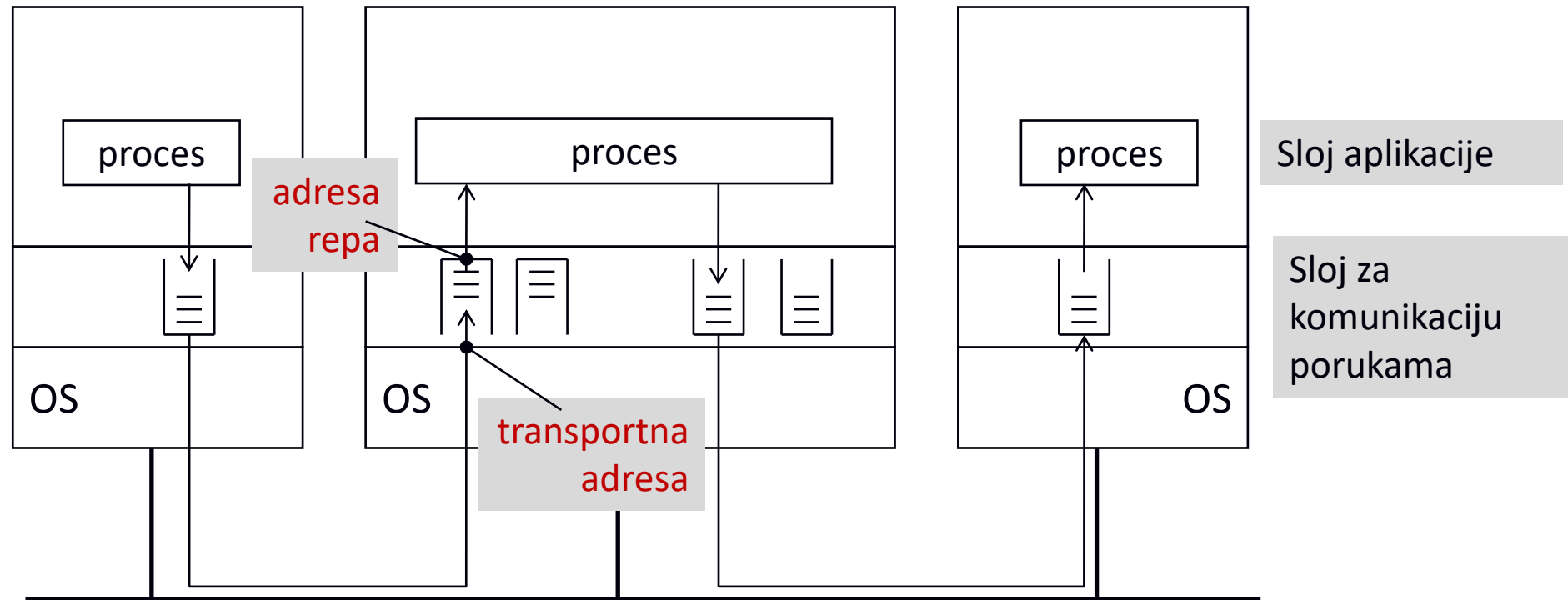
- Procesi/objekti komuniciraju razmjenjujući poruke.
- U komunikaciji sudjeluju izvor (pošiljalac poruke) i odredište (primatelj poruke).
- Izvor šalje poruku, poruka se pohranjuje u rep koji je pridijeljen odredištu.
- Odredište čita poruku iz repa.
- Poruke sadrže podatke, važna je adresa odredišnog repa.
- Adresiranje se izvodi najčešće na nivou sustava, svaki rep ima jedinstven identifikator u sustavu.

Izvođenje komunikacije porukama



- `put` – dodaj poruku u rep
- `get` – pročitaj poruku iz repa, primatelj je blokiran ako je rep prazan
- `poll` – provjeri postoje li poruke u repu i pročitaj prvu poruku ako takva postoji, primatelj nije blokiran

Arhitektura sustava za komunikaciju porukama



Obilježja komunikacije porukama

- vremenska neovisnost
 - primatelj i pošiljatelj ne moraju istovremeno biti aktivni, poruka se sprema u rep
- pošiljatelj mora znati identifikator odredišta, tj. njegovog repa
- komunikacija je perzistentna
- asinkrona komunikacija
 - pošiljatelj šalje poruku i nastavlja obradu neovisno o odgovoru od strane primatelja
- pokretanje komunikacije na načelu *pull*
 - primatelj provjerava postoji li poruka u repu

Komunikacija moguća i na načelu *push*

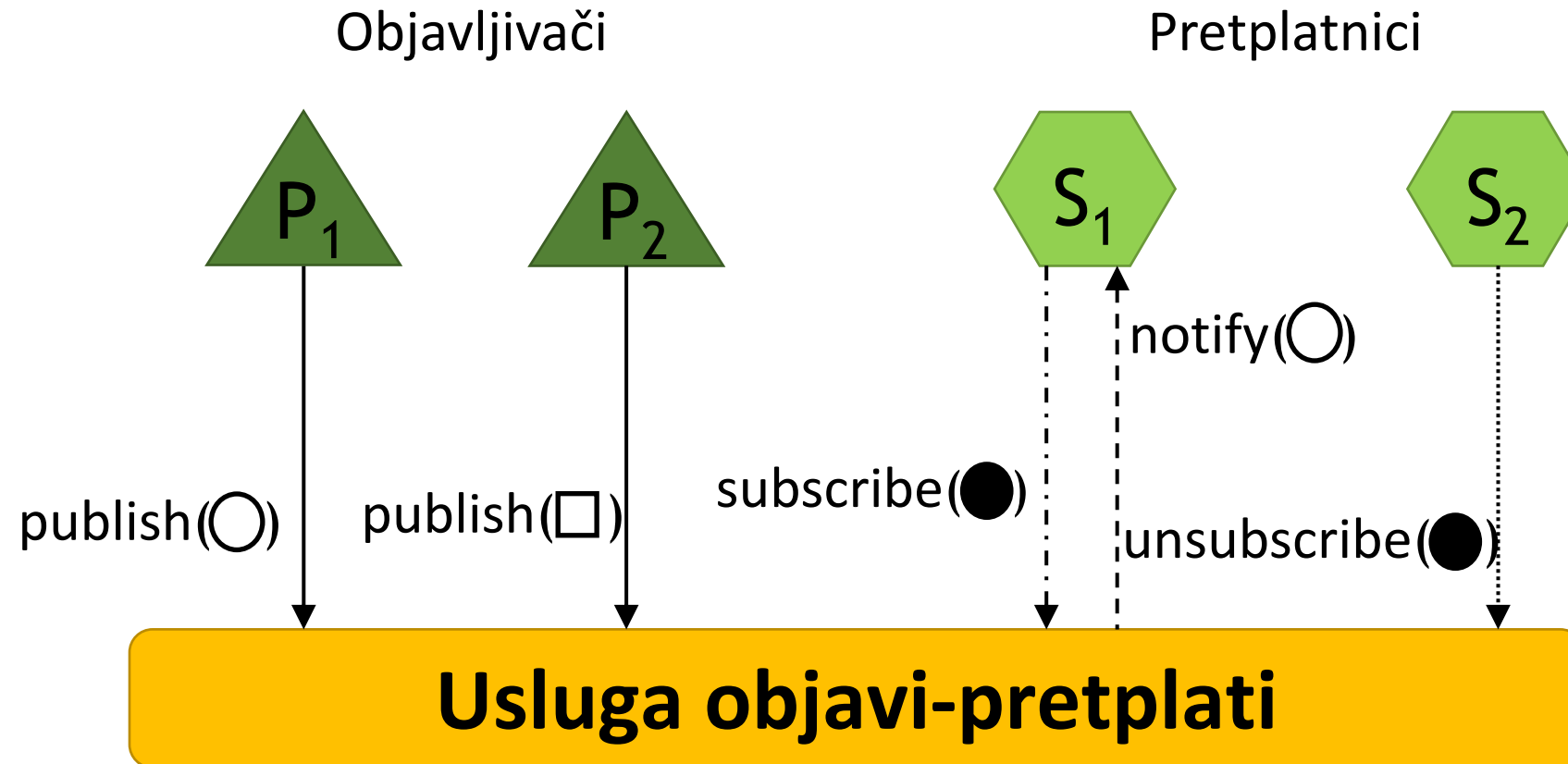


- notify – aktivna isporuka poruke iz repa primatelju po primitku poruke (na strani primateljskog procesa nužan je *listener thread*)

Sadržaj predavanja

- Komunikacija porukama
- Model objavi-pretplati
- Primjeri protokola za komunikaciju porukama: JMS i AMQP
- Dijeljeni podatkovni prostor

Interakcija objavi-pretplati



1 izvor : n odredišta

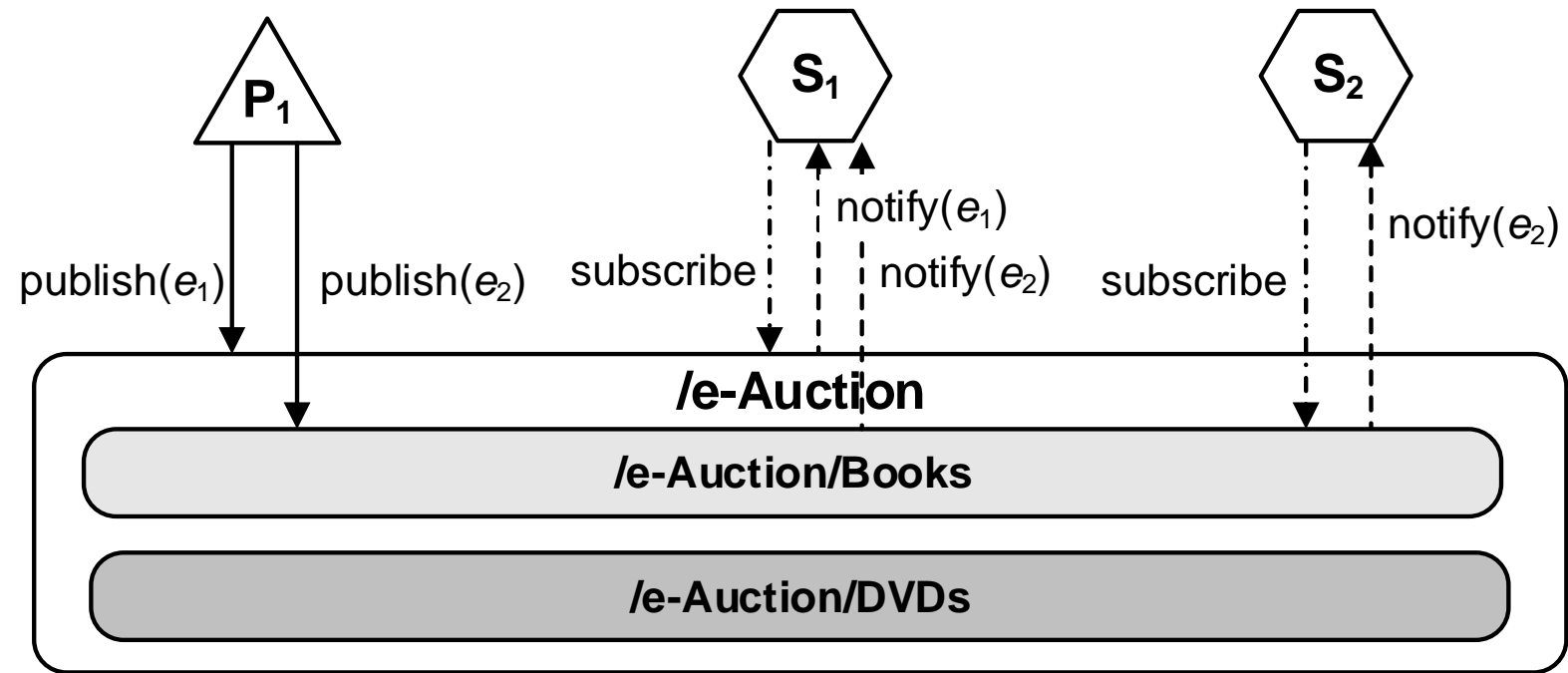
Osnovni pojmovi

- objavljiivači (*publishers*)
 - definiraju obavijesti (*notifications*)
- pretplatnici (*subscribers*)
 - pretplatama (subscriptions) i odjavama pretplata (unsubscriptions) izražavaju namjeru primanja određenog skupa obavijesti
- usluga objavi-pretplati:
 - sustav za obradu događaja (*event service* – ES)
 - obrađuje i pohranjuje primljene obavijesti/pretplata/odjave pretplata
 - isporučuje obavijesti pretplatnicima prema njihovim aktivnim pretplatama
 - omogućuje perzistentnu komunikaciju između objavljiivača i pretplatnika

Pretplate

- „kontinuirani upiti”
- pretplata na kanal/temu (engl. *topic-based subscription*)
 - kanal – logička veza između izvora i odredišta koja služi za tematsko grupiranje obavijesti (npr. vrijeme, sport, itd.)
 - hijerarhijski odnos kanala (npr. vrijeme u Europi, Hrvatskoj, Zagrebu)
- pretplata na sadržaj (engl. *content-based subscription*)
 - pretplata se definira ovisno o svojstvima i sadržaju obavijesti (skup atributa i vrijednosti)

Pretplata na kanal



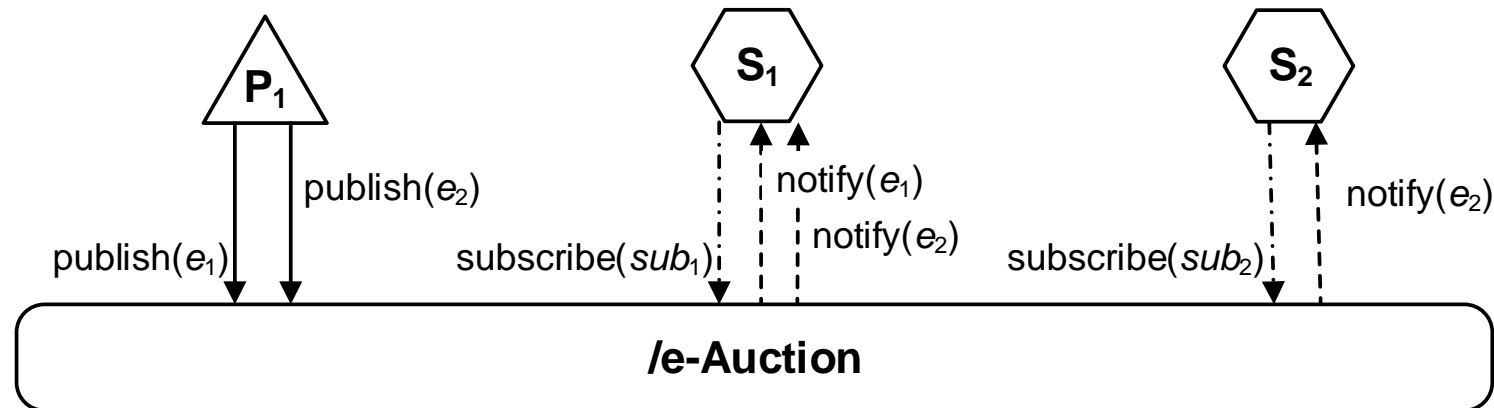
Pretplata na sadržaj

$e_1 = (\text{category} = \text{"books"} \\ \& \text{author} = \text{"D. Adams"} \\ \& \text{title} = \text{"The Hitchhiker's Guide through the Galaxy"} \\ \& \text{price} = 9.99 \text{ EUR})$

$e_2 = (\text{category} = \text{"books"} \\ \& \text{author} = \text{"J.R.R. Tolkien"} \\ \& \text{title} = \text{"The Lord of the Rings"} \\ \& \text{price} = 19.99 \text{ EUR})$

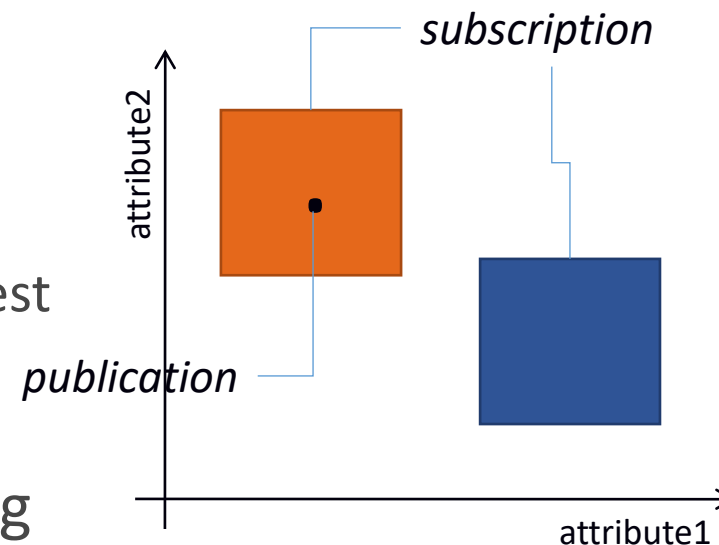
$sub_1 = (\text{category} == \text{"books"} \\ \& \text{price} < 20 \text{ EUR})$

$sub_2 = (\text{category} == \text{"books"} \& \\ \text{author} == \text{"J.R.R. Tolkien"} \\ \& \text{price} < 20 \text{ EUR})$



Primjer obavijesti/pretplata (strukturirani podaci)

- **obavijest** je najčešće točka u višedimenzionalnom prostoru
 - npr. očitavanje senzora, cijena dionice, oglas, vijest
 - objavljiivači kontinuirano objavljuju nove obavijesti (često ograničene valjanosti)
- **pretplata** je potprostor višedimenzionalnog prostora
 - definira se kao Booleova funkcija nad parom (obavijest, pretplata)
 - za pretplatu kažemo da **prekriva** obavijest kada obavijest zadovoljava uvjete pretplate, tj. $f(o_i, p_j) = T$



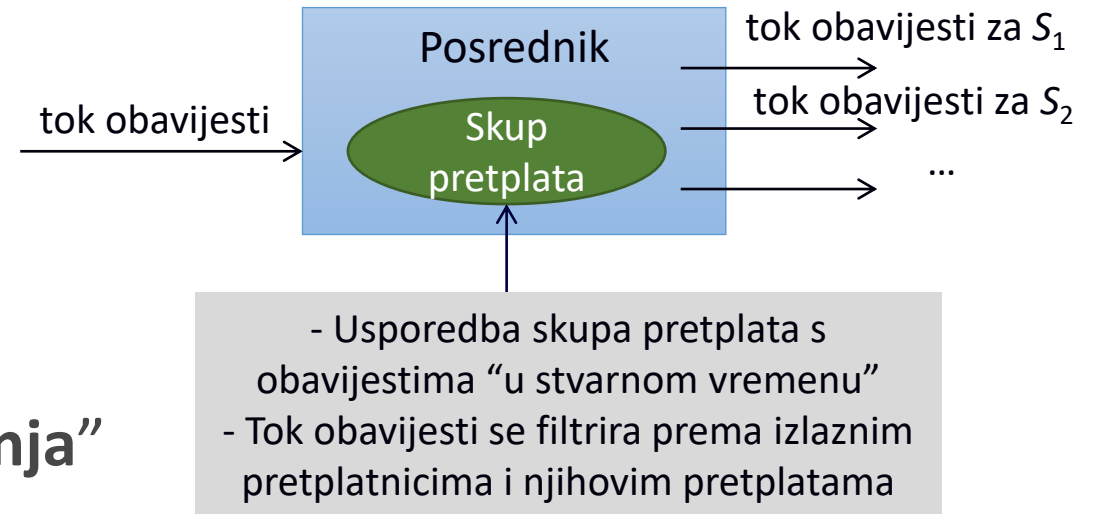
Poseban implementacijski izazov: učinkovita usporedba objave sa skupom pretplata jer je u stvarnom vremenu potrebno odrediti **podskup pretplata** koje **prekrivaju** obavijest kako bi se isporučila svim zainteresiranim pretplatnicima

Usporedba obavijesti sa skupom pretplatama

- Sustav objavi-pretplati održava skup pretplata koje se uspoređuju s novoobjavljenom obavijesti

- Usporedba ispituje svojstvo “**prekrivanja**” obavijesti pretplatom

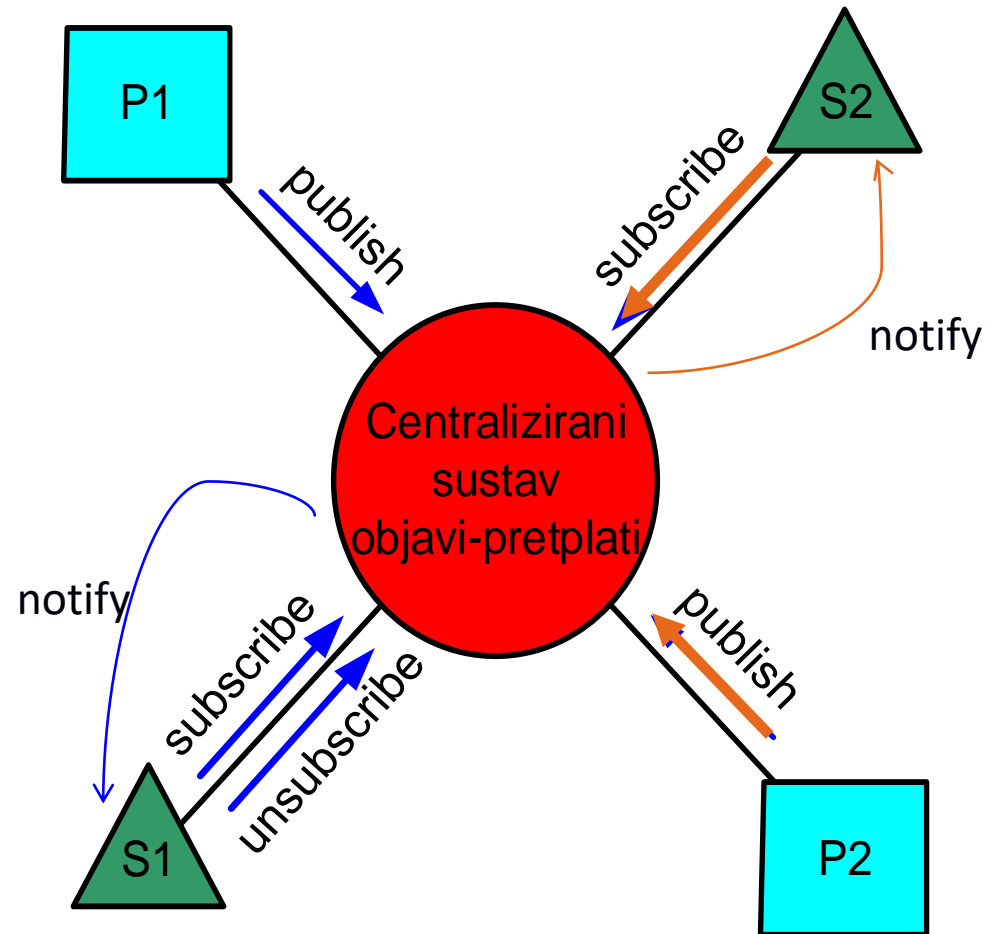
- Pretplata “prekriva” obavijest kada obavijest zadovoljava sve uvjete definirane pretplatom
- Pretplata $[a < 10, b \leq 20]$ prekriva obavijest $[a = 5, b = 20]$, ali ne prekriva obavijest $[a = 5, b = 25]$
- Pretplata $[sveučilište == Zagreb, fakultet == FER]$ prekriva obavijest $[sveučilište = Zagreb, fakultet = FER, vijest = Proslavljen dan FER-a!]$



Kvaliteta usluge za komunikaciju porukama

- Vezana uz garanciju isporuke poruke
 - najviše jednom (*at-most-once*) – ne postoje mehanizmi koji osiguravaju isporuku poruke u slučaju ispada
 - barem jednom (*at-least-once*) – postoje mehanizmi koji će u slučaju ispada ponoviti operaciju, moguće je da će primatelj primiti poruku više puta
 - sigurno jednom (*exactly once*) – primatelj će primiti poruku samo jednom
- Poruke mogu biti perzistentne (imaju vremenski definiran period valjanosti) i neperzistentne poruke („vrijede” u trenutku u kome su definirane)

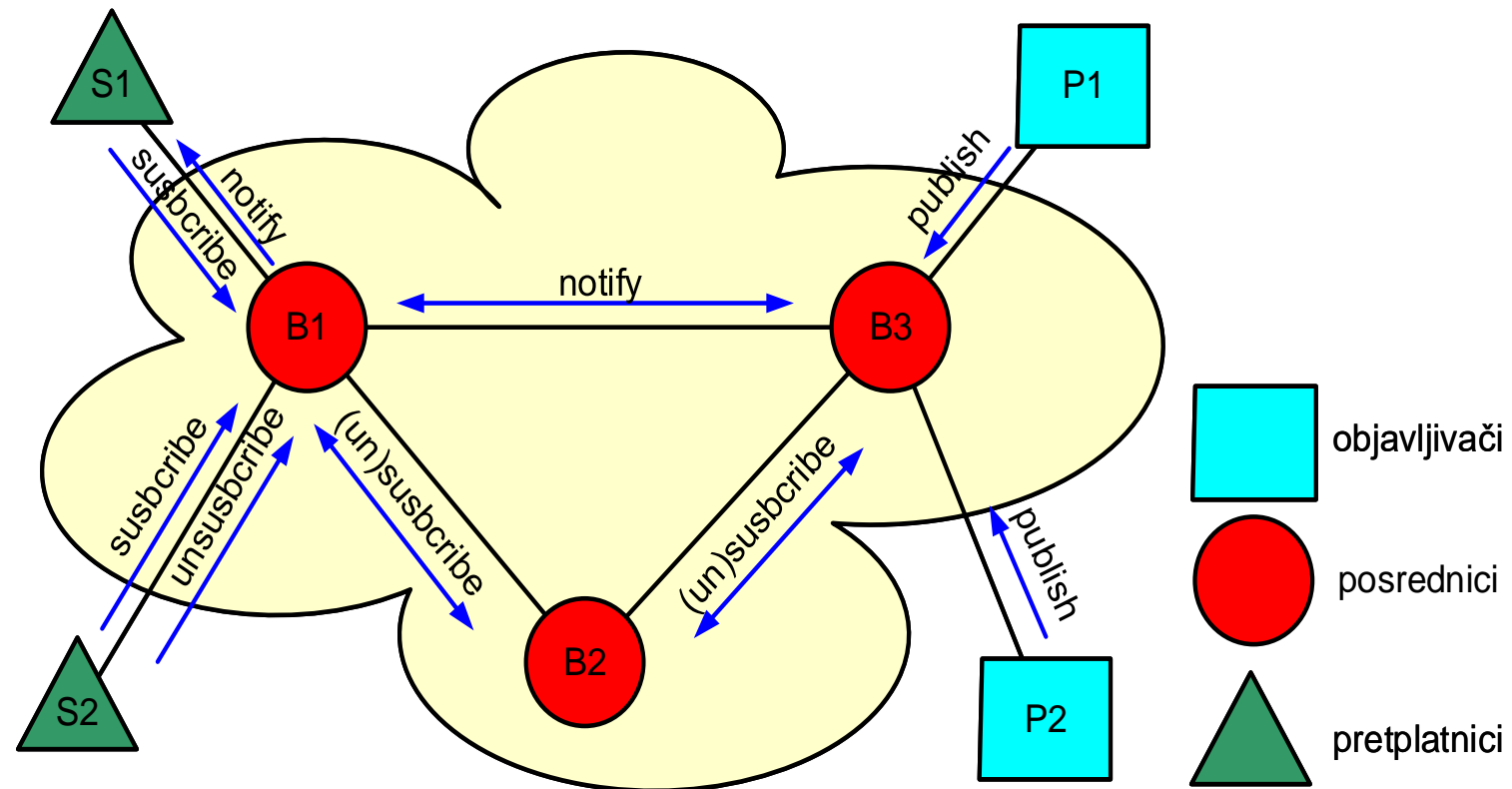
Centralizirana arhitektura



Arhitektura usluge objavi-pretplati

- Centralizirana
 - svi objavljiivači i pretplatnici razmjenjuju obavijesti i definiraju pretplate preko jednog poslužitelja posrednika
 - poslužitelj pohranjuje sve pretplate i prosljeđuje obavijesti
- Raspodijeljena
 - skup poslužitelja, svaki je poslužitelj zadužen za objavljiivače i pretplatnike u svojoj domeni
 - algoritmi za usmjeravanje informacija o pretplatama i usmjeravanje obavijesti

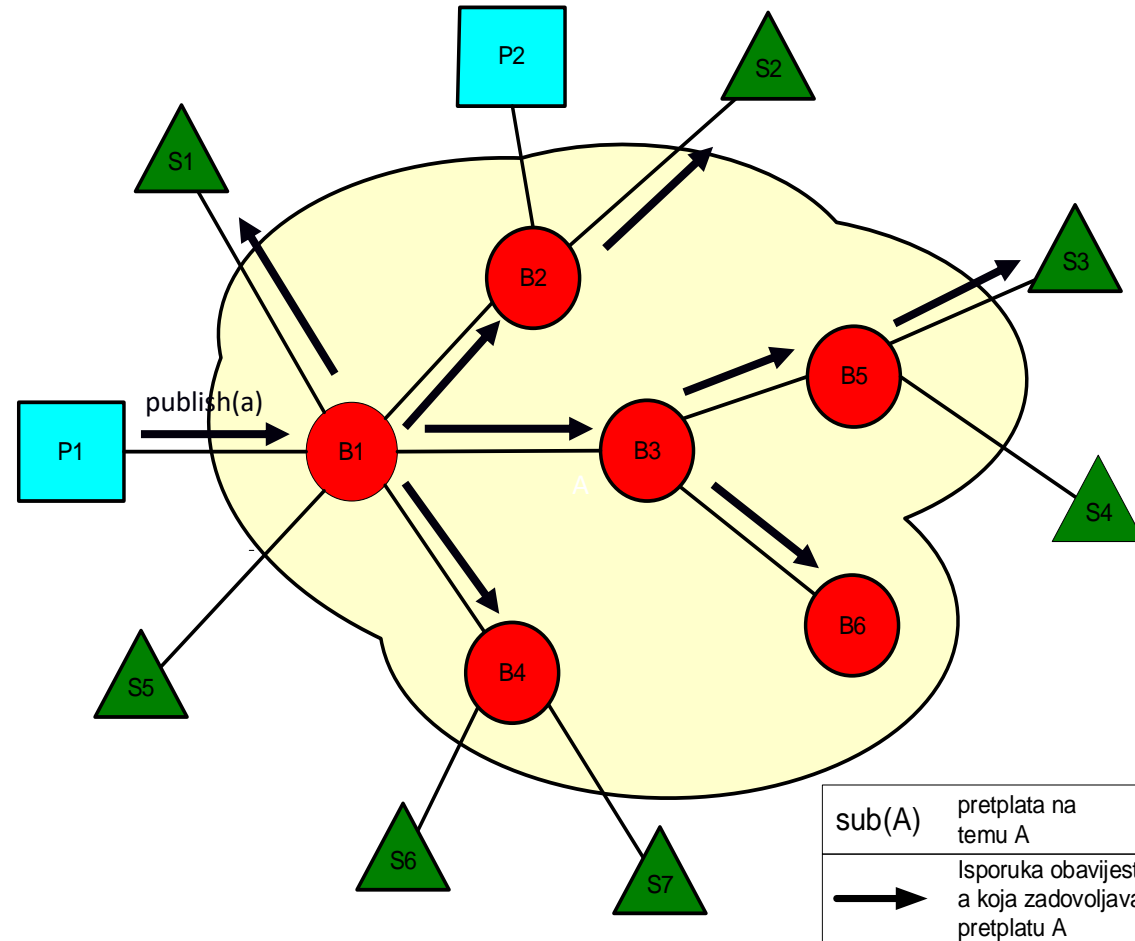
Raspodijeljena arhitektura



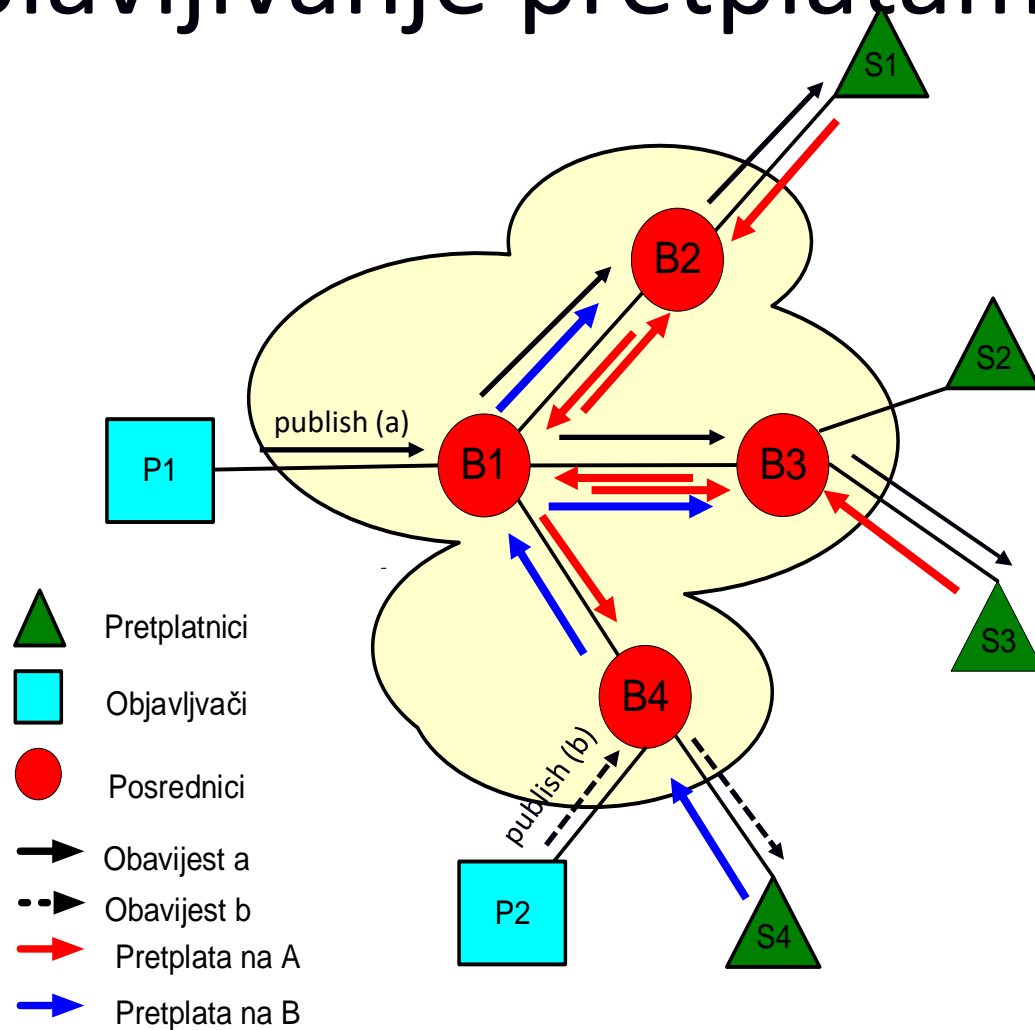
Osnovna načela usmjeravanja

- preplavlivanje
 - svaka primljena poruka (obavijest, pretplata ili odjava pretplate) prosljeđuje se svim susjedima osim onome od koga je poruka primljena
 - posrednik posjeduje tablicu usmjeravanja koja sadrži informacije o svim susjednim posrednicima i lokalnim pretplatnicima
- filtriranje poruka
 - filtriranje poruka se izvodi usporedbom obavijesti s aktivnim pretplatama koje definiraju svojstva obavijesti za koje je pretplatnik zainteresiran
 - osnovni cilj je isporuka samo onih obavijesti koje pretplatnika zanimaju
 - omogućuje i smanjenje prometa u mreži posrednika zbog sprječavanja širenja obavijesti “nezainteresiranim” posrednicima

Preplavljivanje obavijestima



Preplavljanje pretplatama



Za obavijest na temu	Šalji prema
A	B2,B3
B	B4

Tablica usmjeravanja posrednika *B1*

Obilježja modela objavi-pretplati

- **vremenska neovisnost**
 - objavljiivači i pretplatnici ne moraju istovremeno biti aktivni, posrednik pohranjuje poruku
- objavljiivač ne mora znati identifikator pretplatnika (**anonimnost**), o tome se brine posrednik – **prostorna neovisnost**
- komunikacija je **perzistentna**
- **asinkrona komunikacija**
 - objavljiivač šalje poruku i nastavlja obradu neovisno o odgovoru od strane odredišta – **vremenska neovisnost**
- pokretanje komunikacije na načelu **push**
 - objavljiivač šalje poruku posredniku koji je prosljeđuje pretplatnicima bez prethodnog eksplicitnog zahtjeva

Obilježja modela objavi-pretplati (2)

- personalizacija primljenog sadržaja
 - filtriranje objavljenih poruka prema pretplatama
- proširivost sustava
 - dodavanje novog objavljiivača ili pretplatnika ne utječe na ostale strane u komunikaciji
- skalabilnost
 - raspodijeljena arhitektura

Sadržaj predavanja

- Komunikacija porukama
- Model objavi-pretplati
- **Primjeri rješenja za komunikaciju porukama: JMS i AMQP**
- Dijeljeni podatkovni prostor

JMS

Java Message Service

JMS 2.0, Java Community Process, 21.05.2013.

<https://java.net/projects/jms-spec/pages/JMS20FinalRelease>

Specifikacija otvorenog protokola za komunikaciju porukama i komunikaciju na načelu objavi-pretplati.

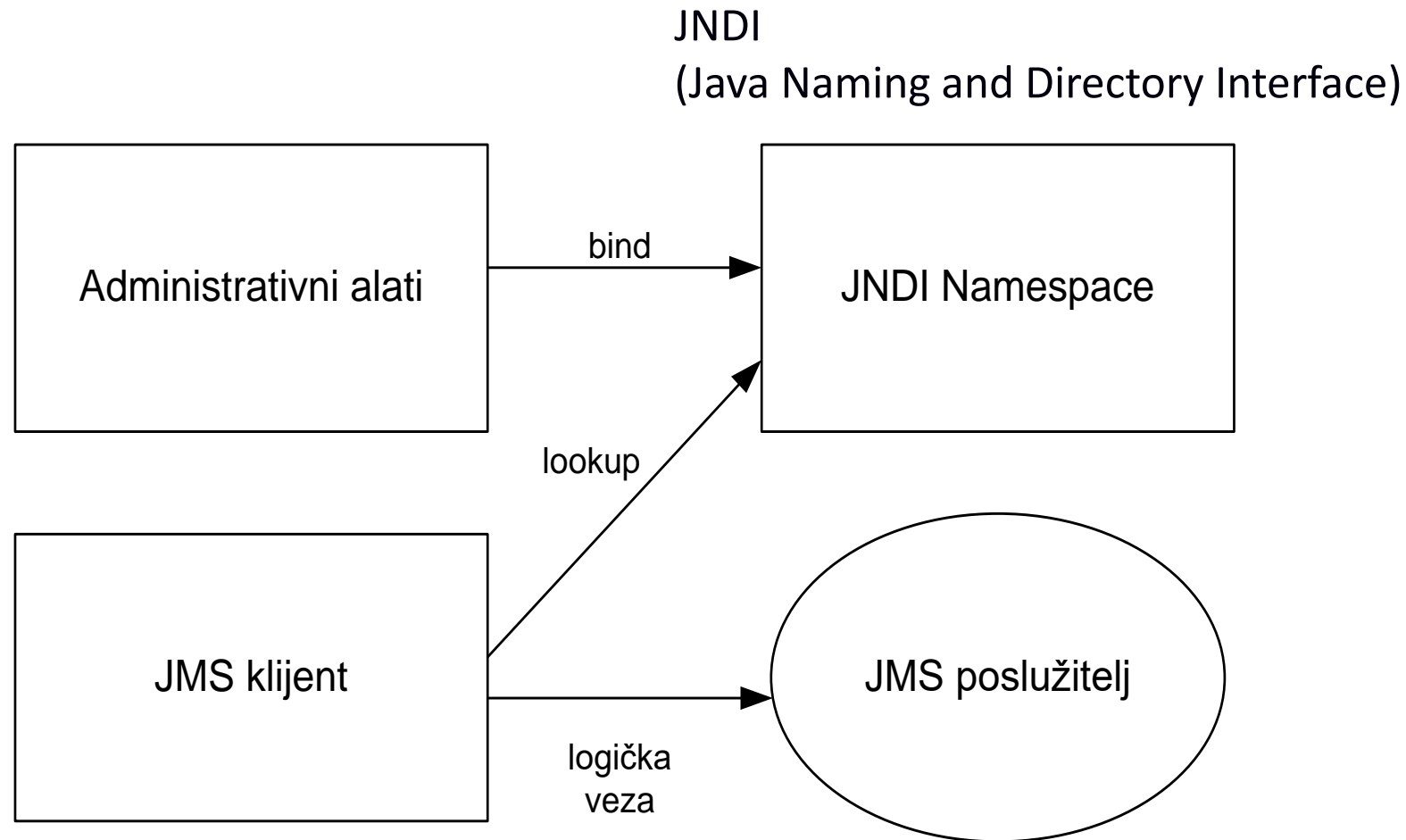
JMS API definira skup sučelja i pripadajuću semantiku koja omogućuje programima pisanim u Javi komunikaciju razmjenom poruka i na načelu objavi-pretplati.

Popularne implementacije: Apache ActiveMQ, IBM WebSphereMQ, HornetQ, OpenJMS

Arhitektura JMS-a (1)

- JMS poslužitelj
 - sustav za razmjenu poruka koji implementira JMS sučelja i nudi administrativne i kontrolne usluge
- Klijent
 - bilo koji objekt, proces ili aplikacija koja stvara ili konzumira poruke
- Poruka (*message*)
 - objekt koji se sastoji od zaglavlja koje prenosi identifikacijske i adresne informacije i tijela koje prenosi podatke
- Odredište (*destination*)
 - objekt koji sadrži informacije o odredištu poruke

Arhitektura JMS-a (2)



Modeli JMS-a

JMS implementira sljedeće modele za komunikaciju porukama i obavijestima

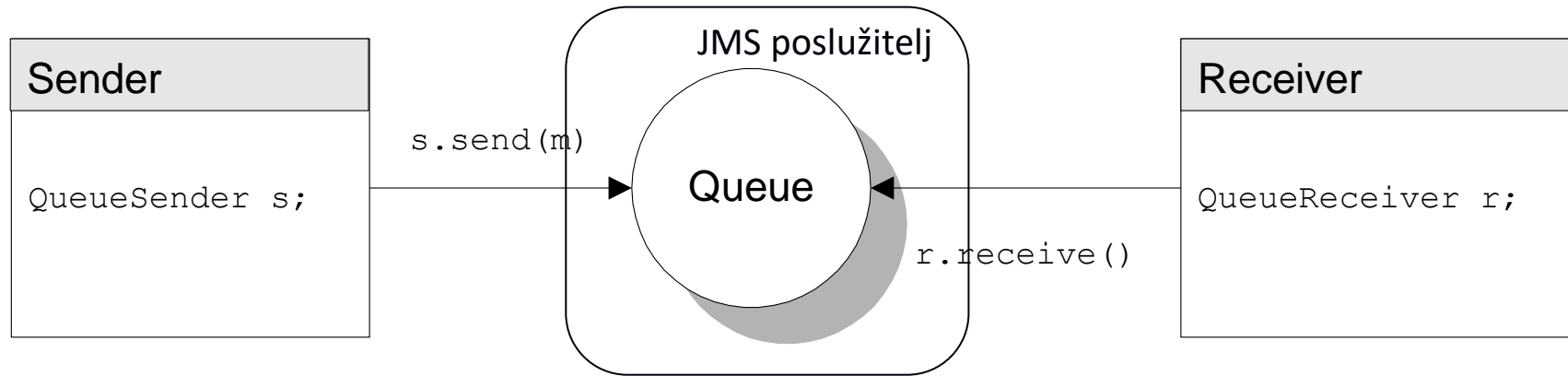
- *Point-to-point*

- komunikacija porukama, jedna poruka za jedno odredište

- *Publish/subscribe*

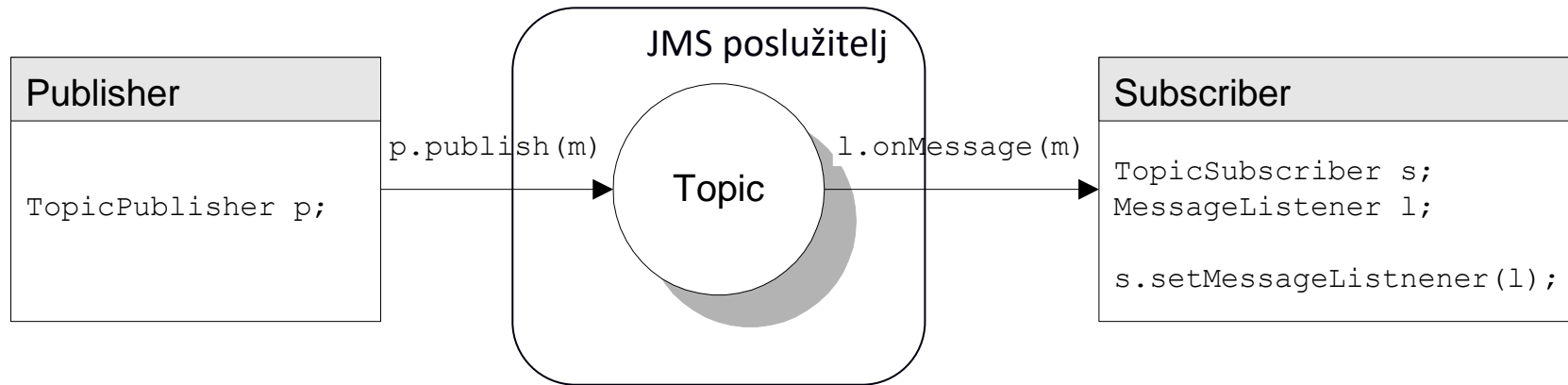
- objavi-pretplati, jedna poruka za skup zainteresiranih pretplatnika

Point-to-point



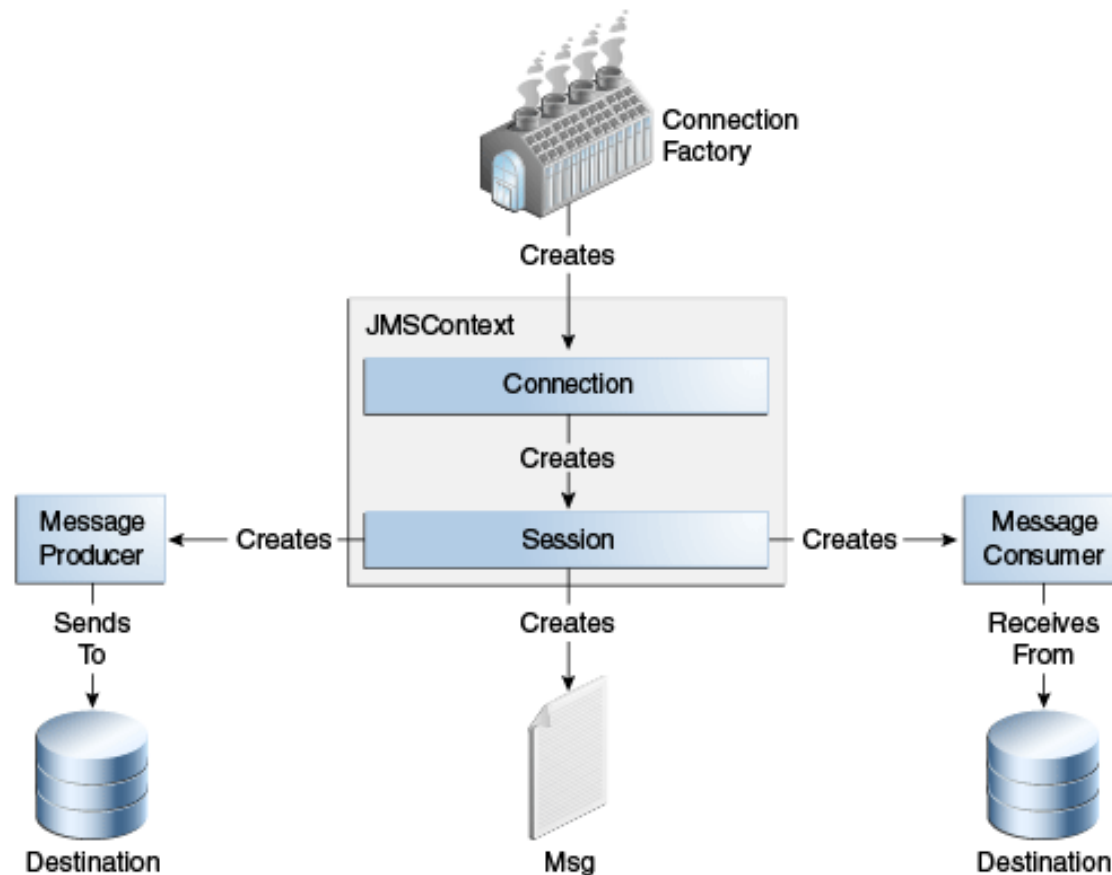
1. Klijent `s` koji šalje poruku `m` poziva `s.send(m)`. Poruka se sprema u rep.
2. Klijent koji prihvaća poruku mora provjeriti postoji li poruka u repu. Poziva `r.receive()`.
3. Poruka se briše iz repa i šalje klijentu.

Publish/subscribe



1. Tijekom inicijalizacije pretplatnik registrira instancu klase koja implementira sučelje `MessageListener` pozivajući `s.setMessageListener(l)`. `Topic` pamti sve pretplate.
2. Izvor objavljuje poruku `m` sa `p.publish(m)`.
3. `Topic` isporučuje poruku pretplatniku pozivajući `l.onMessage(m)`.

Programski model JMS API-ja



Izvor: **The Java EE 7 Tutorial**

Poglavlje 45: Java Message Service Concepts

Sučelja JMS-a (1)

Nad-sučelje	Point-to-point	Publish/subscribe
Destination	Queue	Topic
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection

◆ Destination

- administrirani objekt
- predstavlja odredište - identitet ili adresu repa/teme.

◆ ConnectionFactory

- administrirani objekt koji sadrži konfiguracijske parametre
- klijenti ga koriste za stvaranje objekta *Connection*.

◆ Connection

- predstavlja aktivnu konekciju prema JMS poslužitelju
- klijenti ga koriste za stvaranje sesije (*Session*).

Sučelja JMS-a (2)

Nad-sučelje	Point-to-point	Publish/subscribe
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver	TopicSubscriber

♦ Session

- dretva u kojoj se primaju odnosno šalju poruke
- klijenti koriste sesiju da stvore jedan ili više *MessageProducer* ili *MessageConsumer* objekata

♦ MessageProducer

- objekt za slanje poruka odredištu

♦ MessageConsumer

- objekt za primanje poruka koje su poslane odredištu

Poruke JMS-a

- zaglavlje
 - skup definiranih polja koja sadrže vrijednosti koje identificiraju i usmjeravaju poruku
- svojstva poruke
 - opcionalni parovi ime-vrijednost, a vrijednost može biti *boolean*, *byte*, *short*, *int*, *long*, *float*, *double* ili *String*
- tijelo poruke
 - *TextMessage* sadrži *java.lang.String*. (npr. za slanje XML dokumenata)
 - *StreamMessage* za niz Javinih primitiva.
 - *MapMessage* kada tijelo sadrži skup parova ime-vrijednost.
 - *ObjectMessage* sadrži Java objekt.
 - *ByteMessage* za tijelo koje sadrži niz neinterpretiranih *byte*-ova.

Literatura: JMS

The Java EE 7 Tutorial

- Chapter 45: Java Message Service Concepts
<https://docs.oracle.com/javaee/7/tutorial/jms-concepts.htm>
 - What's New in JMS 2.0, Part One: Ease of Use
<http://www.oracle.com/technetwork/articles/java/jms20-1947669.html>
 - What's New in JMS 2.0, Part Two—New Messaging Features
<http://www.oracle.com/technetwork/articles/java/jms2messaging-1954190.html>
 - Enterprise Integration Patterns
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/toc.htm>
- !

Primjer 1: JMS

1. Perform a JNDI lookup of the ConnectionFactory and Queue:

```
/* Create a JNDI API InitialContext object if none exists yet. */
Context jndiContext = null;
try {
    jndiContext = new InitialContext();
} catch (NamingException e) {
    System.out.println("Could not create JNDI API " + "context: " + e.toString());
    System.exit(1);
}

/* Look up connection factory and destination. If either does not exist, exit. */
QueueConnectionFactory connectionFactory = null;
Queue queue = null;
try {
    connectionFactory = (QueueConnectionFactory)
        jndiContext.lookup("jms/QueueConnectionFactory");
    queue = (Queue) jndiContext.lookup("queue");
} catch (Exception e) {
    System.out.println("JNDI API lookup failed: " + e.toString());
    e.printStackTrace();
    System.exit(1);
}
```

Queue Sender (2)

2. Create a Connection and a Session:

```
QueueConnection connection =  
    connectionFactory.createQueueConnection();  
QueueSession session = connection.createQueueSession(false,  
    Session.AUTO_ACKNOWLEDGE);
```

3. Create a QueueSender and a TextMessage:

```
QueueSender sender = session.createSender(queue);  
TextMessage message = session.createTextMessage();
```


Queue Sender (3)

4. Send one or more messages to the queue:

```
for (int i = 0; i < NUM_MSGS; i++) {  
    message.setText("This is message " + (i + 1));  
    System.out.println("Sending message: " +  
        message.getText());  
    sender.send(message);  
}
```

5. Send an empty control message to indicate the end of the message stream. Sending an empty message of no specified type is a convenient way to indicate to the consumer that the final message has arrived.

```
sender.send(session.createMessage());
```

Queue Sender (4)

6. Close the connection in a finally block, automatically closing the session and QueueSender:

```
} finally {  
    if (connection != null) {  
        try {  
            connection.close();  
        } catch (JMSEException e) {}  
    }  
}
```

Queue Receiver (1)

1. Performs a JNDI lookup of the ConnectionFactory and Queue.
2. Creates a Connection and a Session.
3. Creates a QueueReceiver:

```
QueueReceiver receiver = session.createReceiver(queue);
```

4. Starts the connection, causing message delivery to begin:

```
connection.start();
```

Queue Receiver (2)

5. Receives the messages sent to the destination until the end-of-message-stream control message is received:

```
while (true) {  
    Message m = receiver.receive();  
    if (m != null) {  
        if (m instanceof TextMessage) {  
            message = (TextMessage) m;  
            System.out.println("Reading message: " +  
                message.getText());  
        } else {  
            break;  
        }  
    }  
}
```

- Since the control message is not a TextMessage, the receiving program terminates the while loop and stops receiving messages after the control message arrives.
6. Closes the connection in a finally block, automatically closing the session and QueueReceiver.

TopicPublisher

1. Perform a JNDI lookup of the TopicConnectionFactory and Topic.
2. Create a TopicConnection and a TopicSession.
3. Create a TopicPublisher and a TextMessage.
4. Send one or more messages to the topic.
5. Send an empty control message to indicate the end of the message stream.
6. Close the connection in a finally block, automatically closing the session and TopicPublisher.

TopicSubscriber (1)

1. Perform a JNDI lookup of the TopicConnectionFactory and Topic.
2. Create a TopicConnection and a TopicSession.
3. Create a TopicSubscriber.
4. Create an instance of the TextListener class and registers it as the message listener for the TopicSubscriber:

```
listener = new TextListener();  
subscriber.setMessageListener(listener);
```
5. Start the connection, causing message delivery to begin.

TopicSubscriber (2)

6. Listen for the messages published to the topic, stopping when the user types the character q or Q:

```
System.out.println("To end program, type Q or q, " +  
    "then <return>");  
InputStreamReader = new InputStreamReader(System.in);  
while (!(answer == 'q') || (answer == 'Q')) {  
    try {  
        answer = (char) inputStreamReader.read();  
    } catch (IOException e) {  
        System.out.println("I/O exception: "  
            + e.toString());  
    }  
}
```

7. Close the connection, which automatically closes the session and TopicSubscriber.

Message Listener

1. When a message arrives, the onMessage method is called automatically.
2. The onMessage method converts the incoming message to a TextMessage and displays its content. If the message is not a text message, it reports this fact:

```
public void onMessage(Message message) {
    TextMessage msg = null;

    try {
        if (message instanceof TextMessage) {
            msg = (TextMessage) message;
            System.out.println("Reading message: " +
                               msg.getText());
        } else {
            System.out.println("Message is not a " +
                               "TextMessage");
        }
    } catch (JMSEException e) {
        System.out.println("JMSEException in onMessage(): " +
                           e.toString());
    } catch (Throwable t) {
        System.out.println("Exception in onMessage(): " +
                           t.getMessage());
    }
}
```


AMQP

Advanced Message Queuing Protocol

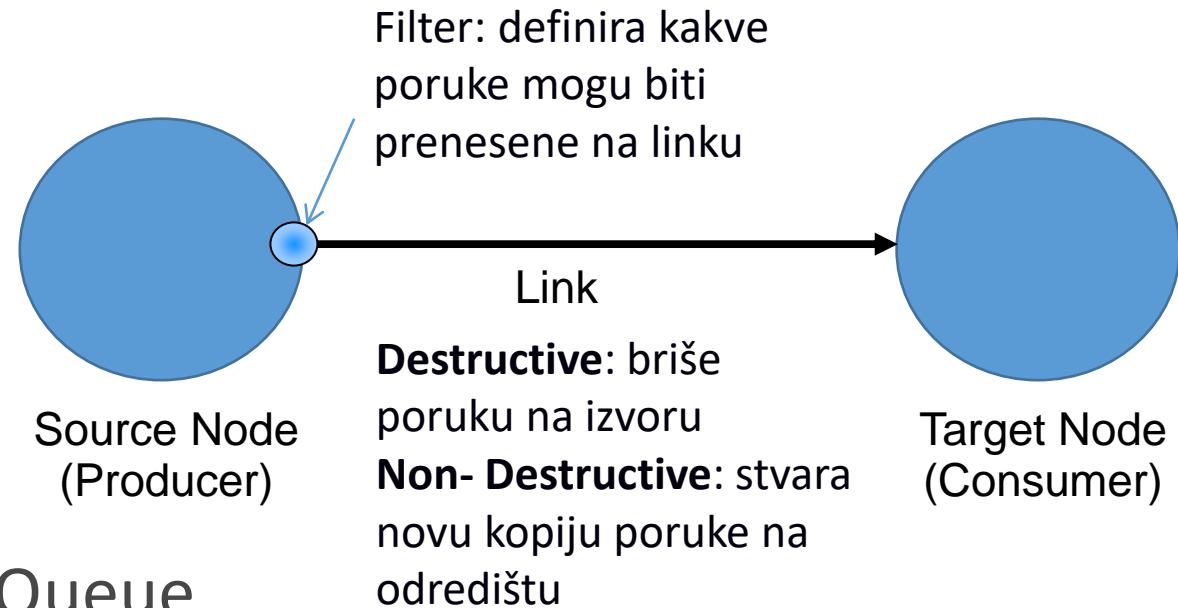
Version 1.0, OASIS Standard, 29.10.2012.

<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html>

Specifikacija **otvorenog protokola za** komunikaciju porukama i komunikaciju na **načelu objavi-pretplati**.

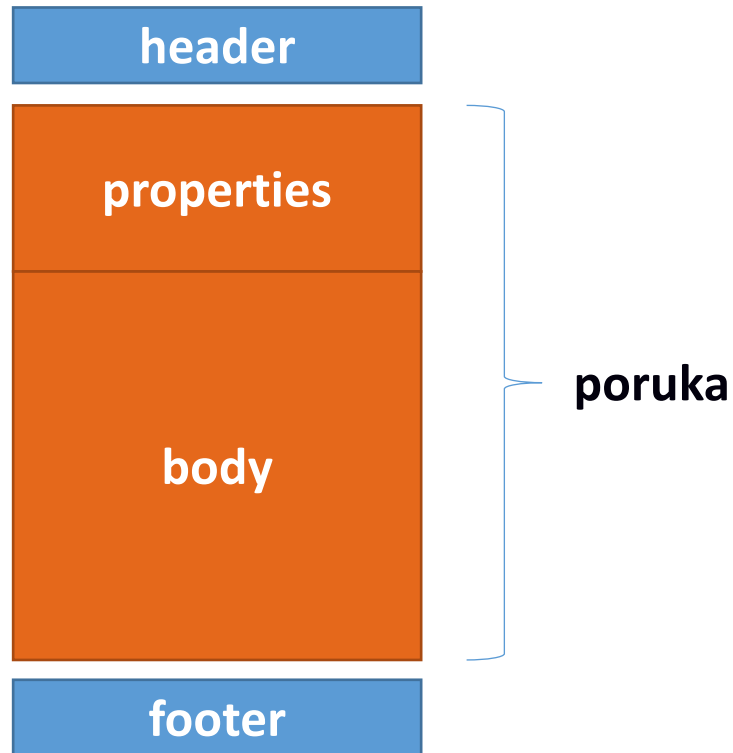
Popularan protokol i raširena primjena zbog implementacije RabbitMQ (**podrška AMQP 0.91**, v1.0 sadrži značajne razlike u odnosu na v0.91, potreban [plugin](#)), OpenAMQ, StortMQ, Apache QPid...

Mreža AMQP



- *Node*: Producer, Consumer, Queue
- *Container*: Broker ili Client
- *Node* postoji unutar *Containera*,
- 1 *container* može sadržavati više čvorova

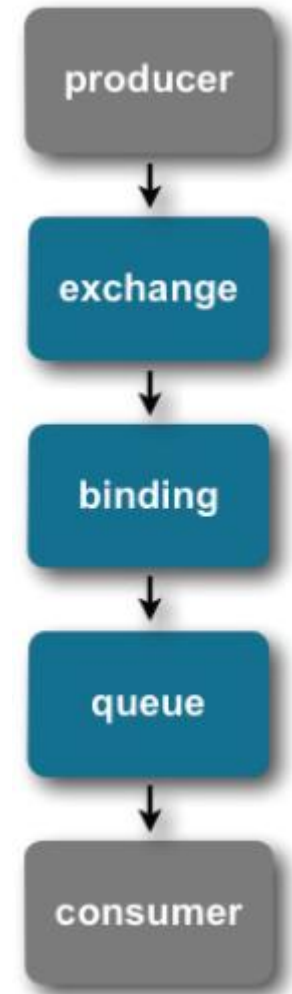
AMQP poruke



- Mreža ne smije mijenjati poruku
- Header i footer se mogu mijenjati u mreži
- Svaka poruka dobiva jedinstveni ID
- Na čvoru smije postojati samo jedna kopija poruke
- Body type: byte message

RabbitMQ: implementira AMQP v0.9

- *Producer*: šalje poruke u *exchange* i dodaje *routing key* uz poruku
- Exchange je povezan s repom putem poveznice (*binding*)
- Binding definira *consumer* (*consumer-driven messaging*), a specificira kakve poruke trebaju biti usmjerene iz *exchangea* do repa
- Consumer je vezan uz rep i prima poruke iz repa
- Uspoređuje se *routing key* i *binding*, ako je uvjet zadovoljen, poruka se isporučuje repu s definiranim *bindingom*

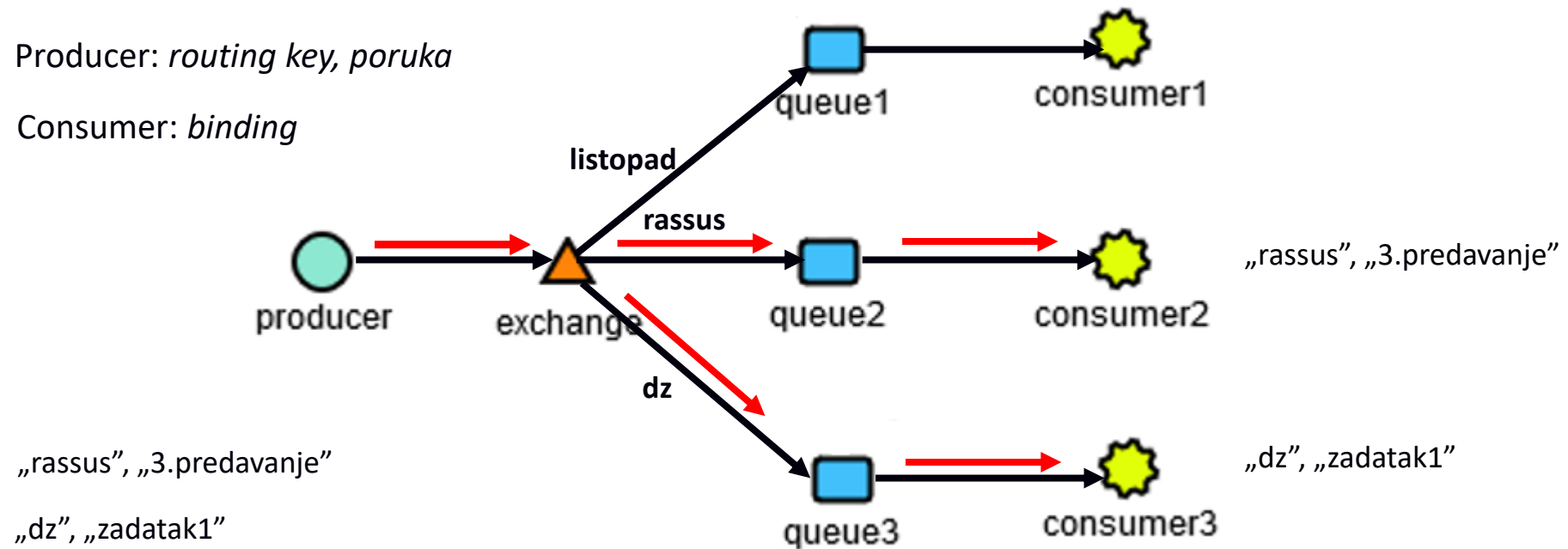


RabbitMQ: različiti komunikacijski modeli

Exchange types

- *Direct Exchange*: odgovara *point-to-point* modelu JMS-a, razlika je u tome što poruka može biti isporučena u više repova ako *binding* odgovara *routing key*-u
- *Fanout Exchange* i *Topic Exchange*: odgovara *publish/subscribe* modelu JMS-a
 - *Fanout* šalje sve poruke na sve repove spojene na exchange
 - *Topic* omogućuje filtriranje poruka i definiranje bindinga uporabom specijalnih znakova # i *

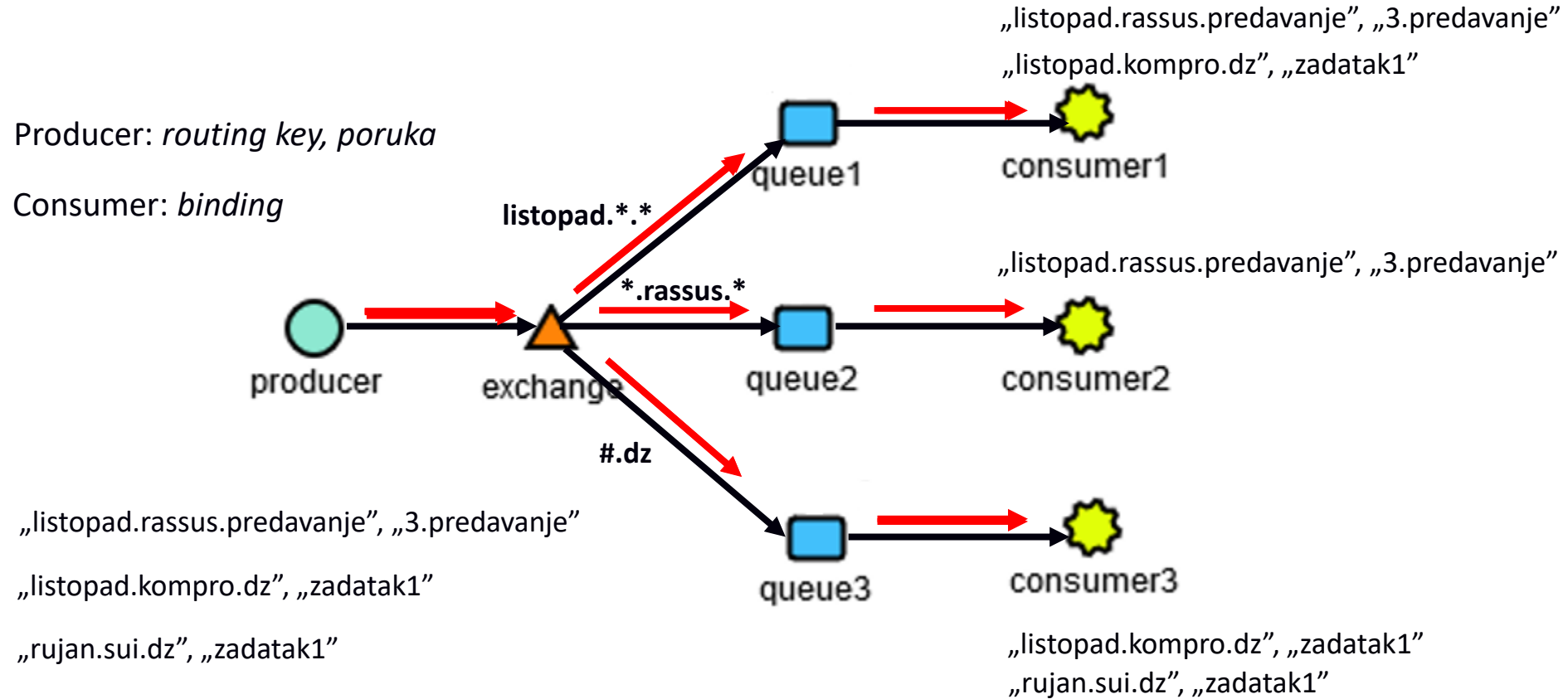
RabbitMQ: *Direct Exchange*



RabbitMQ: *Topic Exchange*

Producer: *routing key, poruka*

Consumer: *binding*



Literatura: AMQP

RabbitMQ Simulator, <http://tryrabbitmq.com/>

- Mark Richards: Understanding the Differences between AMQP & JMS, 2011
<http://www.wmrichards.com/amqp.pdf>
- RabbitMQ Tutorials, <https://www.rabbitmq.com/getstarted.html>
- Spring messaging with RabbitMQ
<https://spring.io/guides/gs/messaging-rabbitmq/>

Primjer 2: AMQP - *Producer*

RabbitMQ: producer šalje poruku na exchange pod nazivom "MyExchange", veže *routing key* uz poruku

```
private final static String EXCHANGE_NAME = "MyExchange";

public static void main(String[] args) throws Exception {
    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

    channel.exchangeDeclare(EXCHANGE_NAME, BuiltInExchangeType.TOPIC);
    String routingKey = "listopad.rassus.predavanje";
    String message = „3.predavanje";
    channel.basicPublish(EXCHANGE_NAME, routingKey, null, message.getBytes());

    channel.close();
    connection.close();
}
```

AMQP - *Consumer*

Consumer definira rep i povezuje ga s *exchangeom*, kada binding odgovara *routing key-u*, poruka se isporučuje

```
...
channel.exchangeDeclare(EXCHANGE_NAME, BuiltInExchangeType.TOPIC);
String queueName = channel.queueDeclare().getQueue();

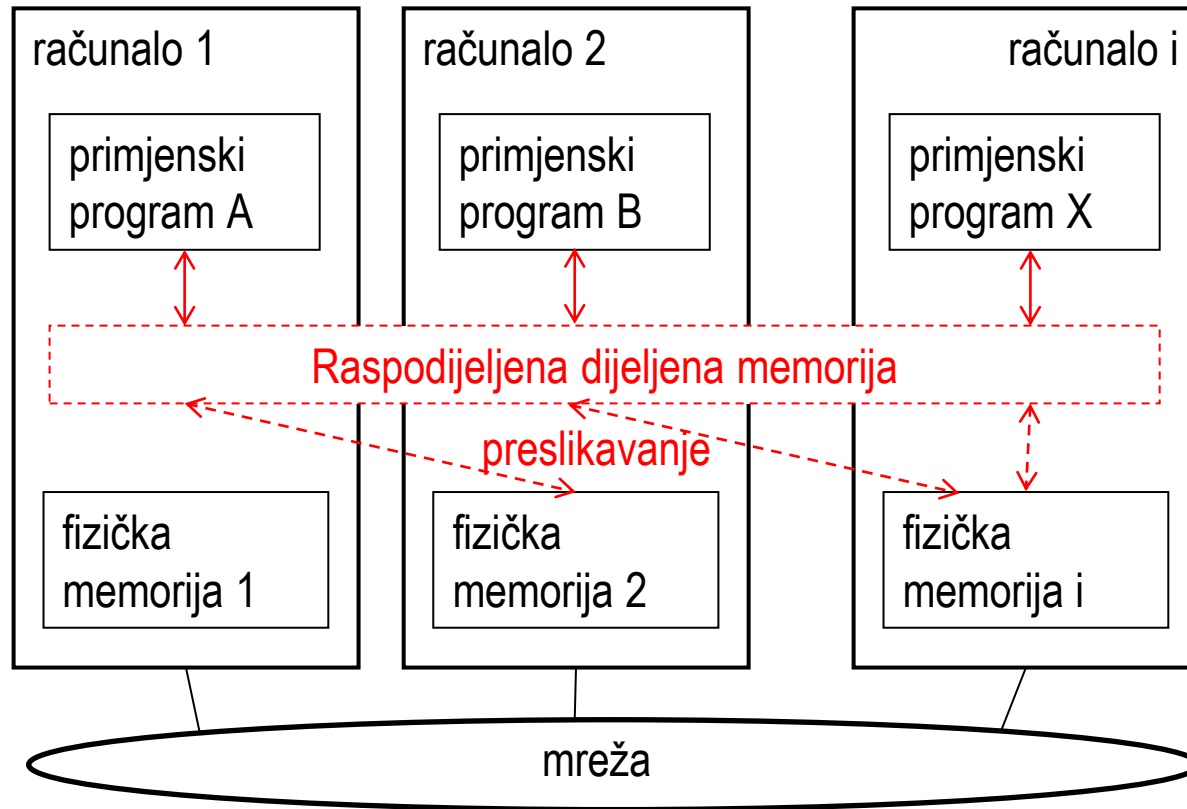
String binding = "/*.rassus.*";
channel.queueBind(queueName, EXCHANGE_NAME, binding);

Consumer consumer = new DefaultConsumer(channel) {
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
        AMQP.BasicProperties properties, byte[] body) throws IOException{
        String message = new String(body, "UTF-8");
        System.out.println("Received: " + message);
    }
};
channel.basicConsume(queueName, true, consumer);
```

Sadržaj predavanja

- Komunikacija porukama
- Model objavi-pretplati
- Primjeri protokola za komunikaciju porukama: JMS i AMQP
- **Dijeljeni podatkovni prostor**

Raspodijeljena dijeljena memorija



- Posrednički sloj koji nudi transparentan pristup dijeljenoj memoriji računala bez zajedničke fizičke memorije

Distributed Shared Memory (DSM)

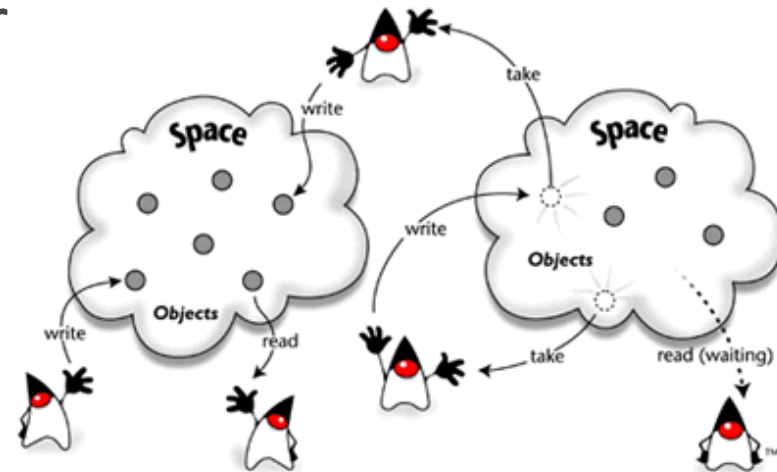
- omogućuje transparentan pristup fizičkoj memoriji na drugim računalima (primjenski program ima dojam da pristupa vlatitoj fizičkoj memoriji)
- upravlja replikama podataka, na računalu i čuvaju se lokalne kopije podataka kojima je nedavno pristupao primjenski program X

Dijeljeni podatkovni prostor

engl. *shared data/tuple spaces*

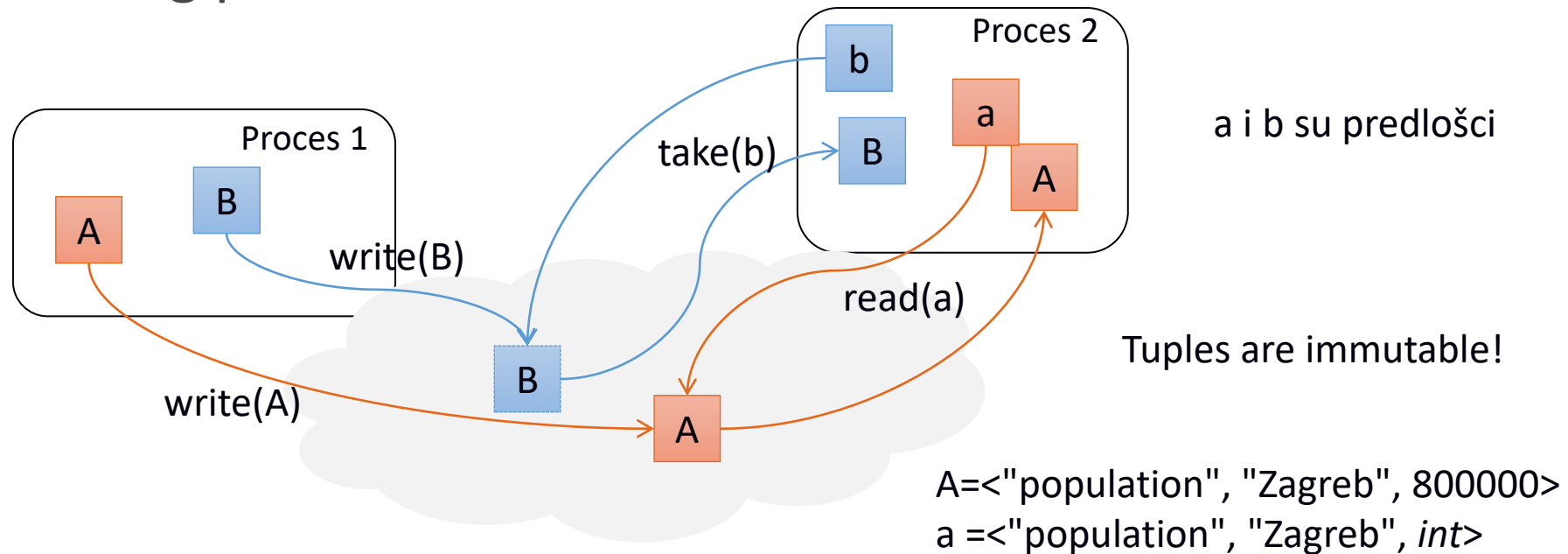
- arhitektura temeljna na podacima (*content-addressable memory*)
- procesi mogu **dodati**, **čitati** i “**izvaditi**” *tuple* iz zajedničkog dijeljenog podatkovnog prostora (*tuple space*)
- *tuple*: slijed podataka, za svaki je definiran
- primjeri: Linda, JavaSpaces, TSpaces

Izvor: "JavaSpaces Principles, Patterns, and Practice"
<http://java.sun.com/developer/Books/JavaSpaces/introduction.html>



Operacije

- **write** (A) – dodaj *tuple* A u raspodijeljeni podatkovni prostor
- **read** (a) $\rightarrow A$ – vraća *tuple* A koji odgovara predlošku a
- **take** (b) $\rightarrow B$ – vraća *tuple* B koji odgovara predlošku b i biše ga iz podatkovnog prostora



Primjer aplikacije



Obilježja dijeljenog podatkovnog prostora

- **vremenska neovisnost**
 - procesi ne moraju istovremeno biti aktivni radi komunikacije, dijeljeni podatkovni prostor pohranjuje poruku
- **anonimna** komunikacija (temelji se na sadržaju podataka)
- komunikacija je **perzistentna**
- **asinkrona komunikacija**
 - proces dodaje podatak u podatkovni prostor i nastavlja obradu
- pokretanje komunikacije na načelu ***pull***
 - proces eksplicitno šalje zahtjev za čitanje podatka iz dijeljenog podatkovnog prostora

Pitanja za učenje i ponavljanje

- Objasnite značenje vremenske i prostorne neovisnosti za komunikaciju procesa. Navedite jesu li komunikacija porukama i komunikacija na načelu objavi-pretplati vremenski i prostorno ovisne ili neovisne.
- Navedite sličnosti i razlike komunikacije na načelu objavi-pretplati i dijeljenog podatkovnog prostora.
- Usporedite pretplatu u sustavima objavi-pretplati i predložak u sustavima s dijeljenim podatkovnim prostorom. Zašto je moguće realizirati tzv. vremenski i prostorno neovisnu komunikaciju?
- Gdje se filtriraju obavijesti u raspodijeljenom sustavu objavi-pretplati koji koristi preplavlivanje obavijestima?

Literatura

1. G. Coulouris, J. Dollimore, T. Kindberg: Distributed Systems: Concepts and Design, 5th edition, Addison-Wesley, 2012
poglavlje 6
2. A. S. Tanenbaum, M. Van Steen: Distributed Systems: Principles and Paradigms, Second Edition, Prentice Hall, 2007
poglavlje 4.3