Diplomski studij

Informacijska i komunikacijska tehnologija:

Telekomunikacije i informatika

Računarstvo:

Programsko inženjerstvo i informacijski sustavi

Računarska znanost

Ak.god. 2008./2009.

Raspodijeljeni sustavi

2.

Modeli raspodijeljene obrade (2.dio):
Java RMI, komunikacija porukama, objavi-pretplati, JMS

Sadržaj predavanja

- Java RMI
- Komunikacija porukama
- Model objavi-pretplati
- Java Message Service (JMS)

Java Remote Method Invocation

- Sunovo rješenje za komunikaciju raspodijeljenih objekata na načelu poziva udaljene metode
- Oblikovan isključivo za programski jezik Java: omogućuje jednostavniju komunikaciju objekata koji se izvode u različitim JVM (*Java Virtual Machine*)
- Implementacija koristi TCP kao transportni protokol

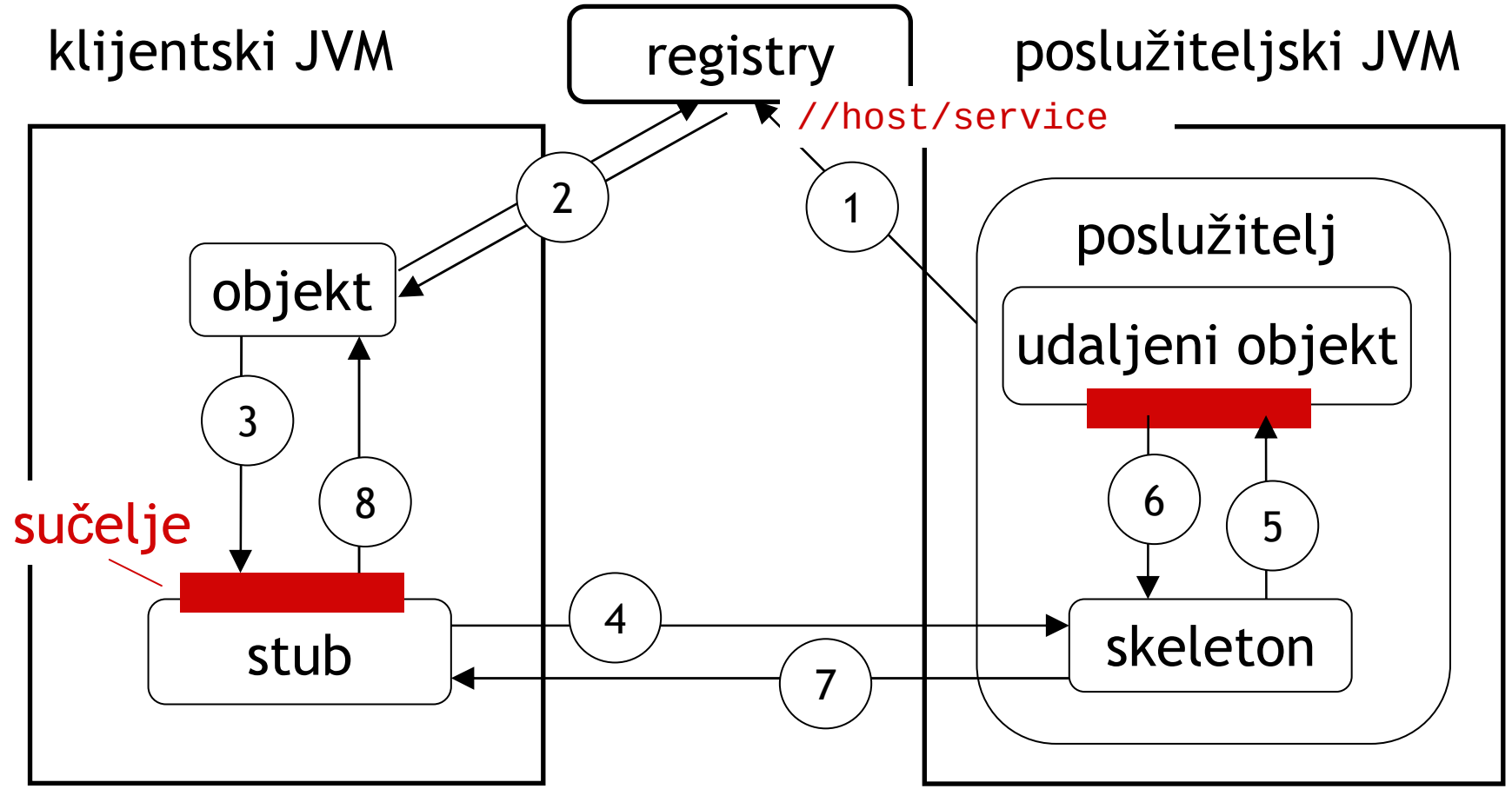
Javin model objekta

- transparentnost pristupa udaljenim objektima
  - referenca na udaljeni objekt istovjetna je referenci na lokalni objekt, no moraju implementirati sučelje `java.rmi.Remote`
- sučelja udaljenog objekta omogućuju komunikaciju s udaljenim objektom
- sučelje udaljenog objekta implementira *stub* (*proxy*) u adresnom prostoru klijentskog računala
- klase *stub* i *skeleton* generiraju se iz implementacije, a ne iz sučelja udaljenog objekta

Prenošenje parametara udaljenoj metodi

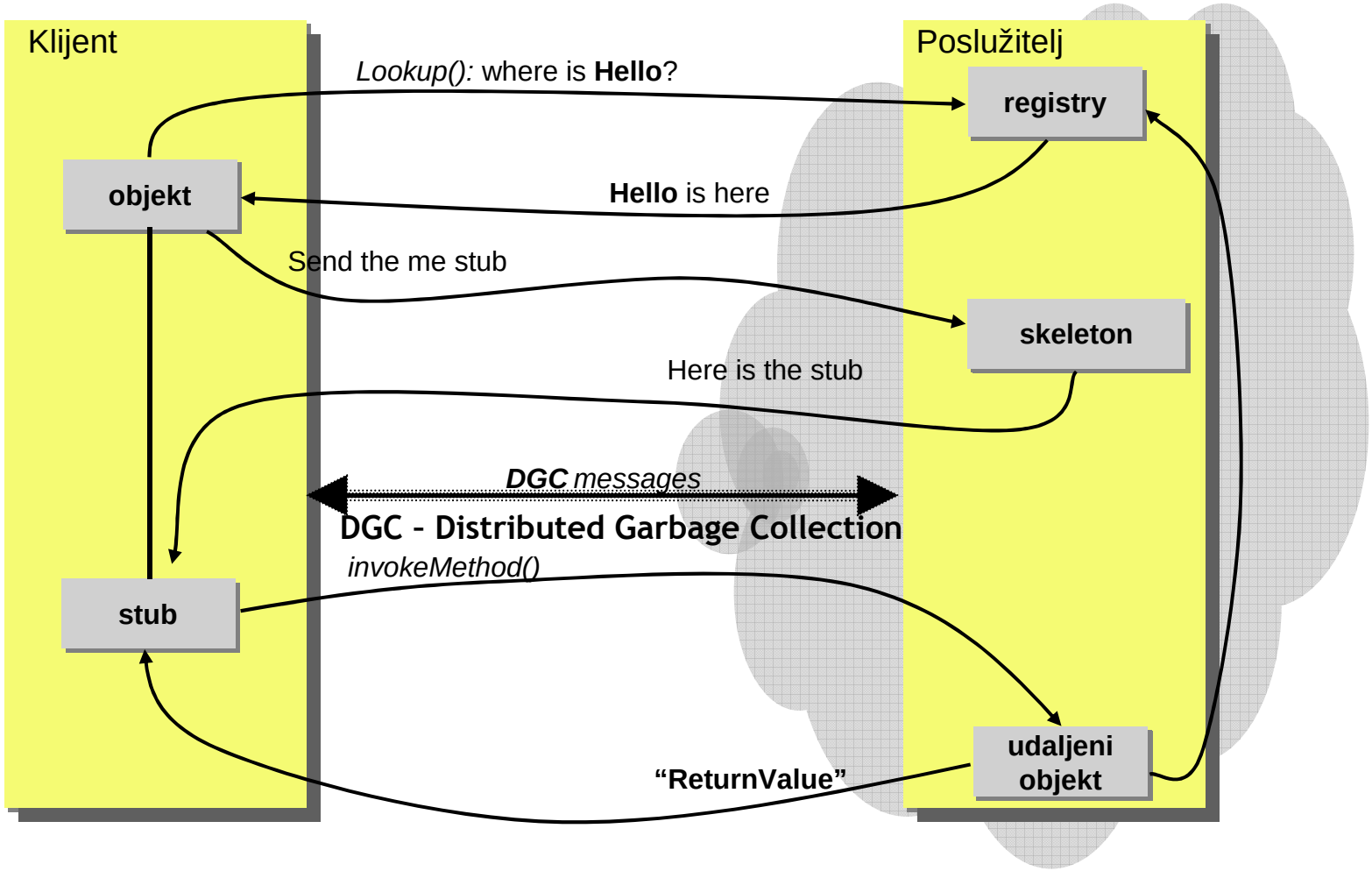
- lokalni objekti moraju se serijalizirati i prenosi se njihova vrijednost (*pass by value*)
  - implementiraju sučelje `Serializable`
- udaljeni se objekti prenose koristeći referencu (*pass by reference*)
  - implementiraju sučelje java.rmi.Remote i pravilno su eksportirani `UnicastRemoteObject.exportObject()`
  - referenca = adresa računala + port + identifikator udaljenog objekta
  - referenca udaljenog objekta je jedinstvena u raspodijeljenom sustavu

Protokol Java RMI (1)

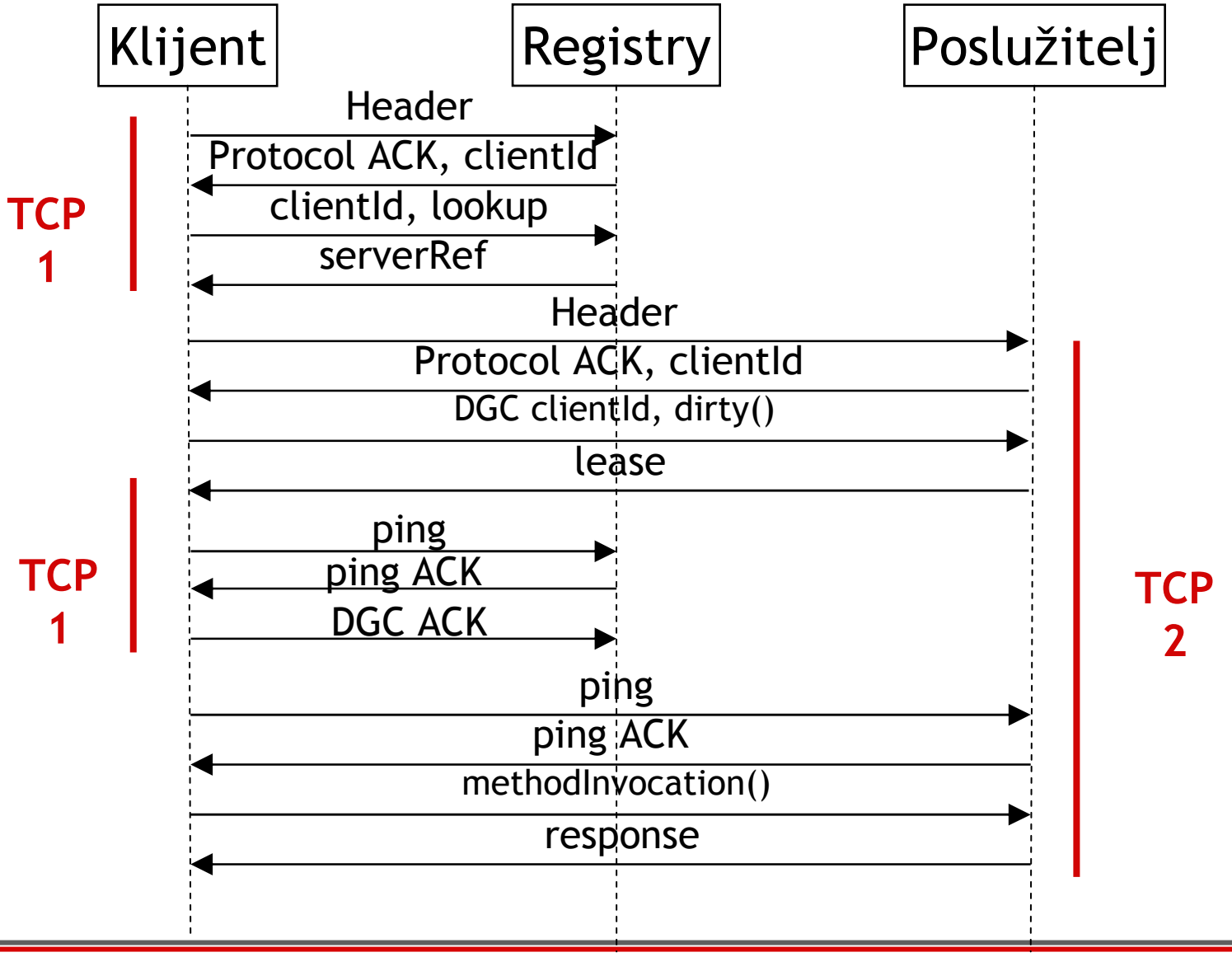


Pretpostavka: stub postoji na strani klijenta

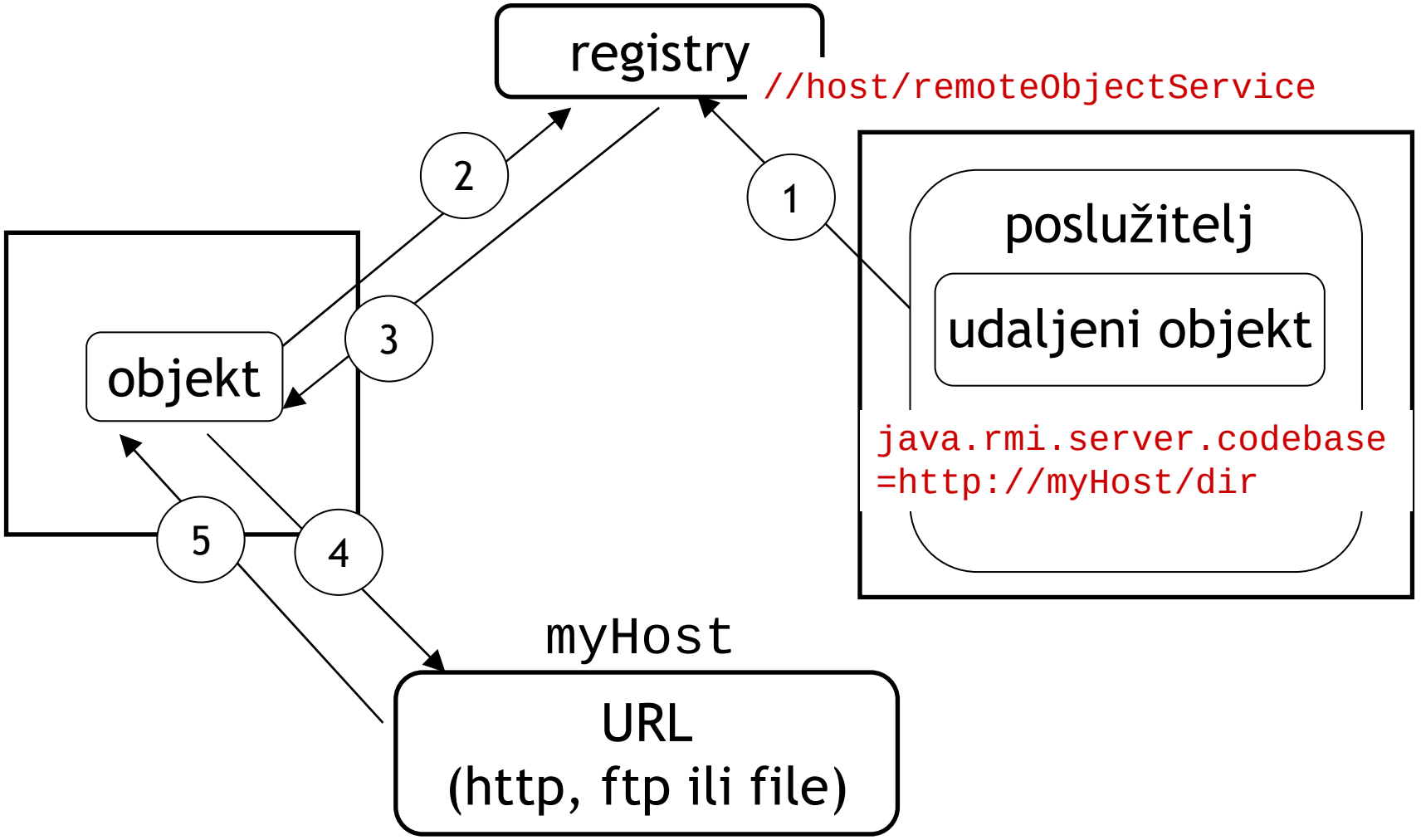
Protokol Java RMI (2)



Protokol Java RMI (3)



Dinamičko učitavanje klase stuba (1)



Primjer RMI sučelja


```
import java.rmi.RemoteException;
import java.rmi.Remote;

/**
 * Remote object offers the service of converting a
  string
 * to upper case.
 */
public interface UpperCase extends Remote {

  public String toUpperCase
    (String originalString) throws RemoteException;

}
```

Primjer RMI poslužitelja (1)


```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;

public class UpperCaseImpl extends UnicastRemoteObject
   implements UpperCase {

   private static String rmiUrl =
       new String("rmi://localhost:1099/UpperCase4U");

   public UpperCaseImpl() throws RemoteException {
      super();
   }

   public String toUpperCase( String originalString )
       throws RemoteException {
       return( originalString.toUpperCase() );
   }...
```

Primjer RMI poslužitelja (2)


```
...
  public static void main(String[] args) {
   try {
       if (System.getSecurityManager() == null) {
              System.setSecurityManager(new
   RMISecurityManager());

       UpperCaseImpl serverObject = new UpperCaseImpl();
       Naming.rebind(rmiUrl, serverObject);
       System.out.println("UpperCase object bound to " +
   rmiUrl);

   } catch (Exception e) {
      e.printStackTrace();
  }
}}
```

Primjer RMI klijenta (1)


```
import java.rmi.RemoteException;
import java.rmi.NotBoundException;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;

public class UpperCaseClient {
   private String rmiUrl =
       new String( "rmi://localhost:1099/UpperCase4U" );
   private UpperCase uc = null;
   public UpperCaseClient() {
       try {
          uc = (UpperCase) Naming.lookup( rmiUrl );
          System.out.println( "Found remote object " +
              uc.toString() );
       } catch( Exception e ) {
              e.printStackTrace();
       }
    }...
```

Primjer RMI klijenta (2)


```
...
   public static void main(String[] args) {

       if (System.getSecurityManager() == null)
          System.setSecurityManager(new
   RMISecurityManager());

       UpperCaseClient client = new UpperCaseClient();
       try {
              String any = new String( "Any string...");
              System.out.println( "Sending\t" + any );
              System.out.println("Received\t"
                     + client.uc.toUpperCase(any));
       }
       catch (Exception e) {
              e.printStackTrace();
       }
   System.exit(0);
   }}
```

Serijalizacija objekta

- omogućuje kodiranje objekata u niz bitova te dekodiranje niza bitova i stvaranje originalnog objekta
- koristi se u Javi za prijenos objekata kod komunikacije *socketima* ili za RMI
- serijalizirani objekt se može zapisati na disk i nakon toga ponovo kreirati deserijalizacijom

Sučelje `Serializable`

- klasa mora implementirati sučelje `Serializable` da bi se njeni objekti mogli serijalizirati
- sučelje `Serializable` nema definirane metode
- postoji standardni način na koji Java serijalizira objekte
  - sve varijable unutar objekta moraju se moći serijalizirati
  - naslijeđena klasa mora se moći serijalizirati
- za posebne slučajeve se mogu definirati pravila serijaliziacije i deserijalizacije

Pravila serijalizacije

- private void writeObject(java.io.ObjectOutputStream out) throws IOException
  - definira pravila za serijalizaciju
  - konstante su dio klase i one se ne serijaliziraju
  - varijable koje se ne žele serijalizirati deklariraju se kao `transient`
- private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException
  - definira pravila za deserijalizaciju
  - kreirani objekt mora biti jednak objektu koji nastaje pozivanjem konstruktora

Loš primjer serijalizacije


```
public class StringList implements Serializable {
   private int size = 0;
   private Entry head = null;

   private static class Entry implements Serializable {
       String data;
       Entry next;
       Entry previous;
   }...
}
```

Dobar primjer serijalizacije (1)


```
import java.io.*;

public class StringList implements Serializable {
    private transient int size   = 0;
    private transient Entry head = null;

    private static class Entry {
        String data;
        Entry  next;
        Entry  previous;
    }

    public void add(String s) {
        Entry e = new Entry();
        e.data = s;
        e.next = head;
        if (head != null)
            head.previous = e;
        head = e;
    }...
```

Dobar primjer serijalizacije (2)


```
...
     private void writeObject(ObjectOutputStream s)
             throws IOException {
         s.defaultWriteObject();
         s.writeInt(size);
         // Write out all elements in the proper order.
         for (Entry e = head; e != null; e = e.next)
             s.writeObject(e.data);
     }

     private void readObject(ObjectInputStream s)
             throws IOException, ClassNotFoundException {
         s.defaultReadObject();
         int size = s.readInt();
         // Read in all elements and insert them in list
         for (int i = 0; i < size; i++)
             add((String)s.readObject());
     }
}
```

Obilježja Java RMI-a

- pozitivna svojstva
  - visok nivo transparentnosti
  - poziv udaljene metode ima jednaku sintaksu pozivu lokalne metode
  - podržava dinamičko učitavanje klasa
  - jednostavna i brza implementacija distribuiranog sustava
  - jednostavniji i čitljiviji kod programa
- negativna svojstva
  - performanse: poziv udaljene metode je puno sporiji od poziva metode lokalnog objekta, čak i ako su udaljeni objekt i klijent na istom računalu (TCP + dizajn protokola s velikim brojem ping paketa)

Paket `java.rmi`

http://java.sun.com/javase/6/docs/api/java/rmi/package-summary.html

- http://www.unix.com.ua/orelly/java-ent/jenut/ch13_01.htm
- The Java Tutorials, **Trail: RMI**
  http://java.sun.com/docs/books/tutorial/rmi/index.html
- **jGuru: Remote Method Invocation (RMI)**
  http://java.sun.com/developer/onlineTraining/rmi/RMI.html
- **Discover the secrets of the Java Serialization API**
  http://java.sun.com/developer/technicalArticles/Programming/serialization/

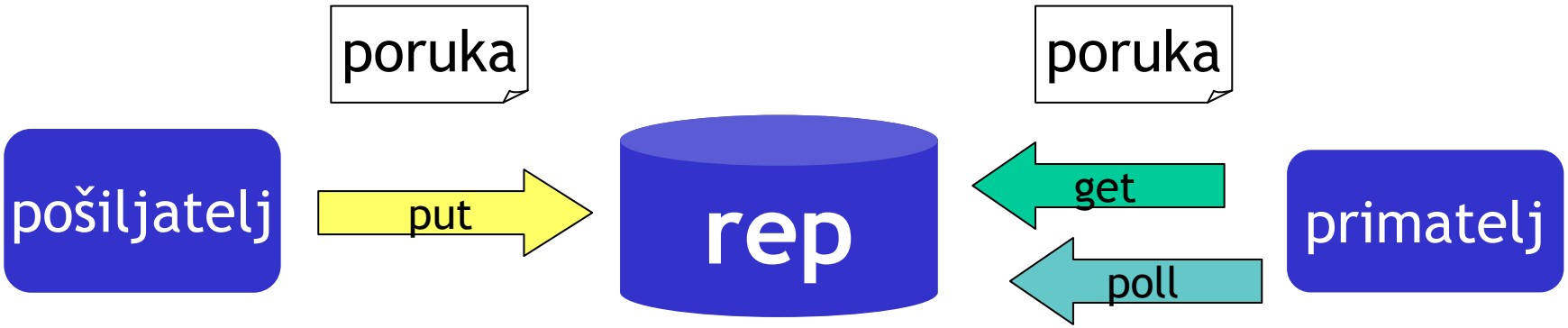
Sadržaj predavanja

- Java RMI
- Komunikacija porukama
- Model objavi-pretplati
- Java Message Service (JMS)

Komunikacija porukama

- Procesi/objekti komuniciraju razmjenjujući poruke.
- U komunikaciji sudjeluju izvor (pošiljatelj poruke) i odredište.
- 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 (1)



1 izvor : 1 odredište

Izvođenje komunikacije porukama (2)

- `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

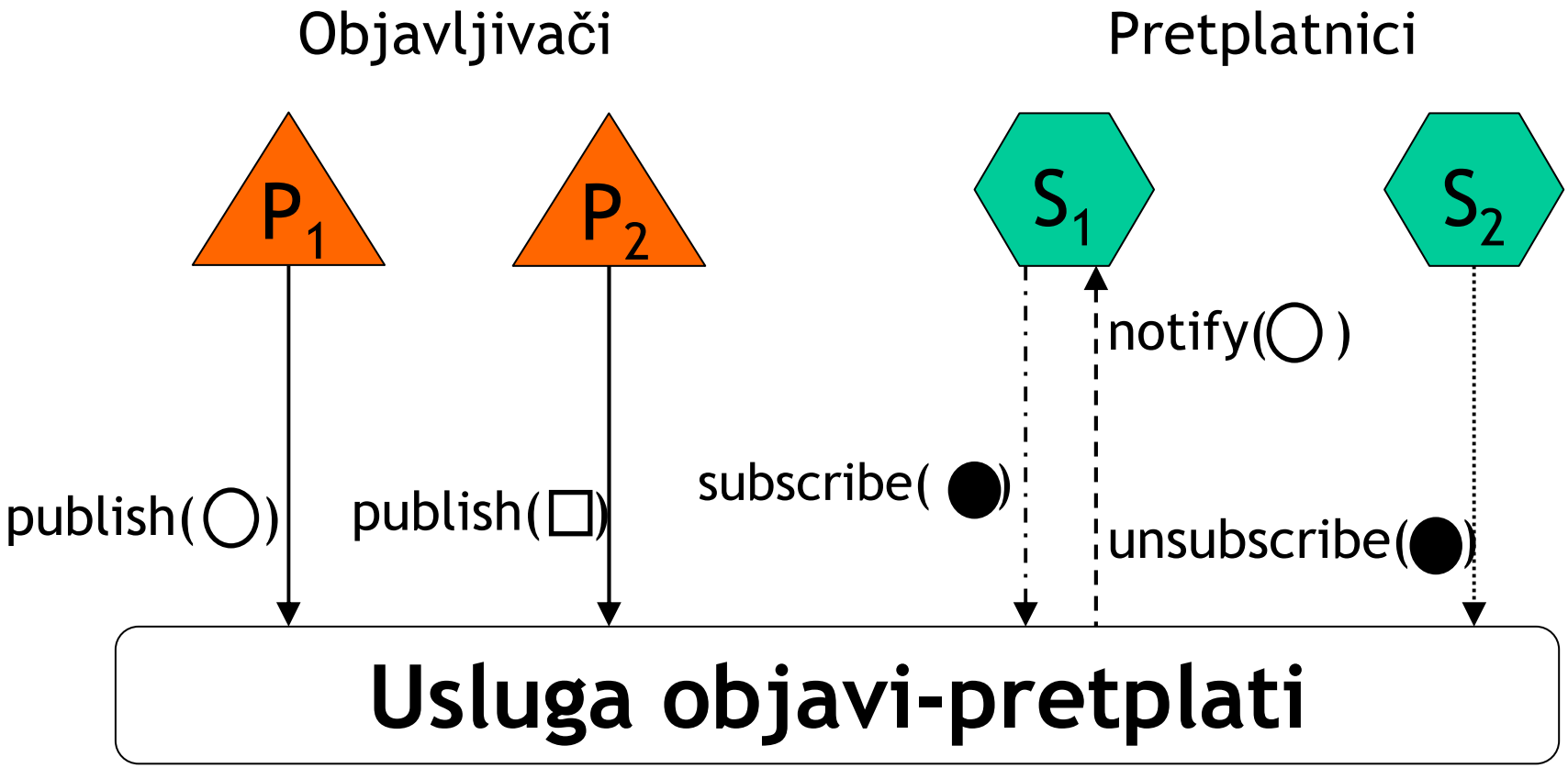
Karakteristike komunikacije porukama

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

Sadržaj predavanja

- Java RMI
- Komunikacija porukama
- Model objavi-pretplati
- Java Message Service (JMS)

Interakcija objavi-pretplati



Osnovni pojmovi (1)

- objavljivači (*publishers*)
- pretplatnici (*subscribers*)
- usluga objavi-pretplati:
  - sustav za obradu događaja (*event service* – ES)
- objavljivači i pretplatnici komuniciraju razmjenjujući sljedeće entitete preko usluge objavi-pretplati
  - obavijesti (*notifications*) – definiraju objavljivači,
  - pretplate (*subscriptions*) – definiraju pretplatnici i
  - odjave pretplata (*unsubscriptions*) – definiraju pretplatnici.

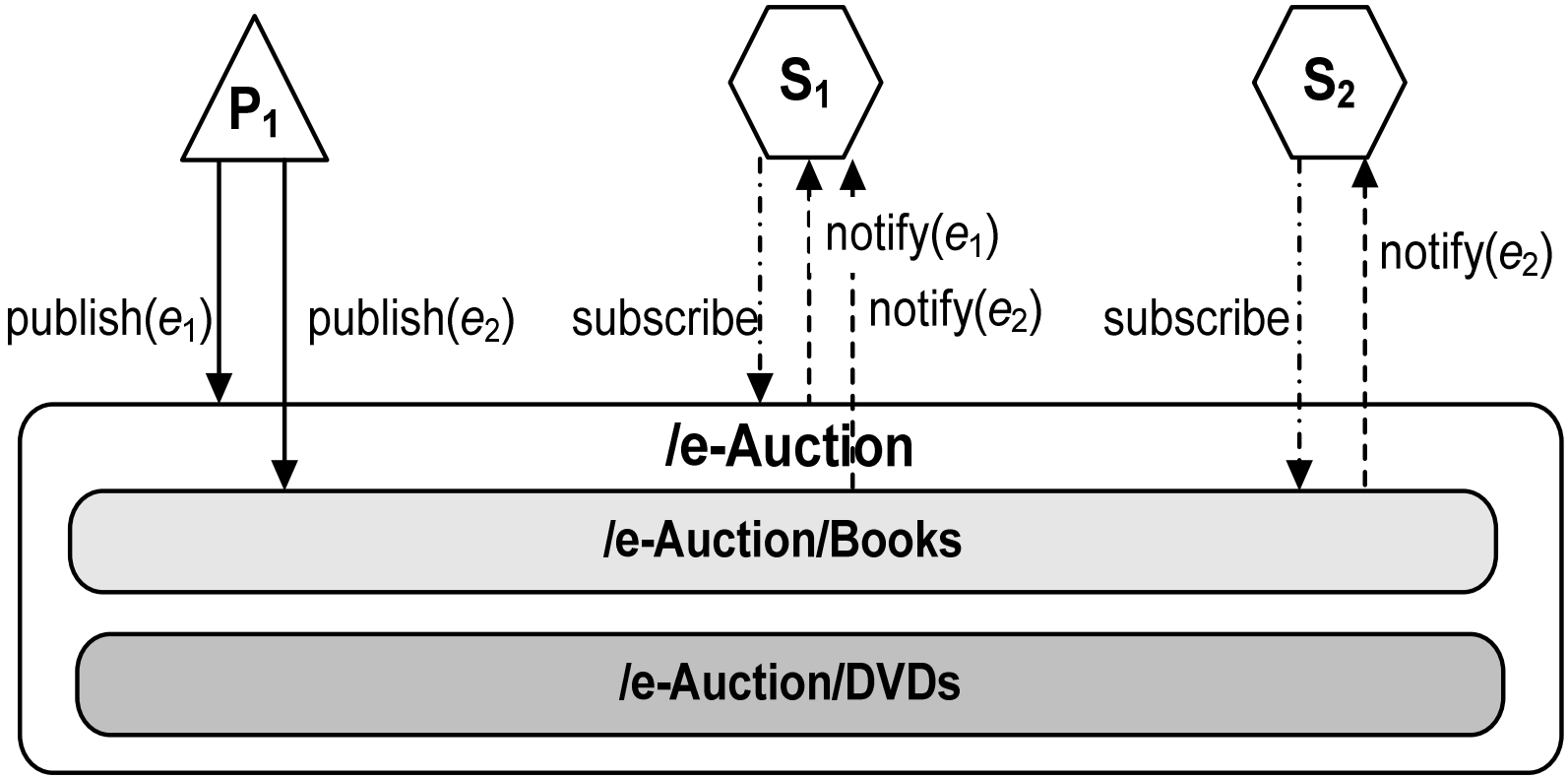
Osnovni pojmovi (2)

- objavljivači (*publishers*)
  - definiraju obavijesti
- pretplatnici (*subscribers*)
  - pretplatama i odjavama pretplata izražavaju namjeru primanja određenog skupa obavijesti
- usluga objavi-pretplati
  - obrađuje i pohranjuje primljene obavijesti/pretplate/odjave pretplata
  - isporučuje obavijesti pretplatnicima prema njihovim aktivnim pretplatama

Vrste pretplate

- Pretplata na kanal
  - tematsko grupiranje obavijesti (npr. vrijeme)
  - hijerarhijski odnos kanala (npr. vrijeme u Europi, Hrvatskoj, Zagrebu)
  - kanal – logička veza između izvora i odredišta
- Pretplata na sadržaj
  - pretplata se definira ovisno o svojstvima i sadržaju obavijesti (skup atributa i vrijednosti)

Pretplata na kanal



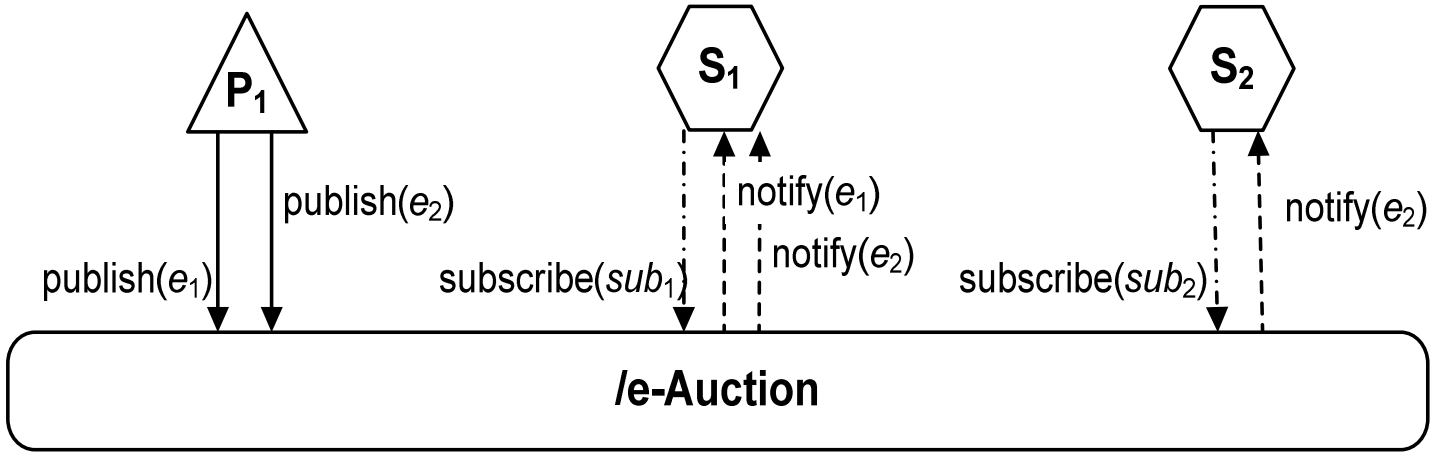
Pretplata na sadržaj

e_1 = (category = “books”
& author = “D. Adams”
& title = “The Hitchhiker's Guide through the Galaxy”
& price = 9.99 EUR)

e_2 = (category = “books”
& author = “J.R.R. Tolkien”
& title = “The Lord of the Rings”
& price = 19.99 EUR)

sub_1 = (category == “books”
& price < 20 EUR)

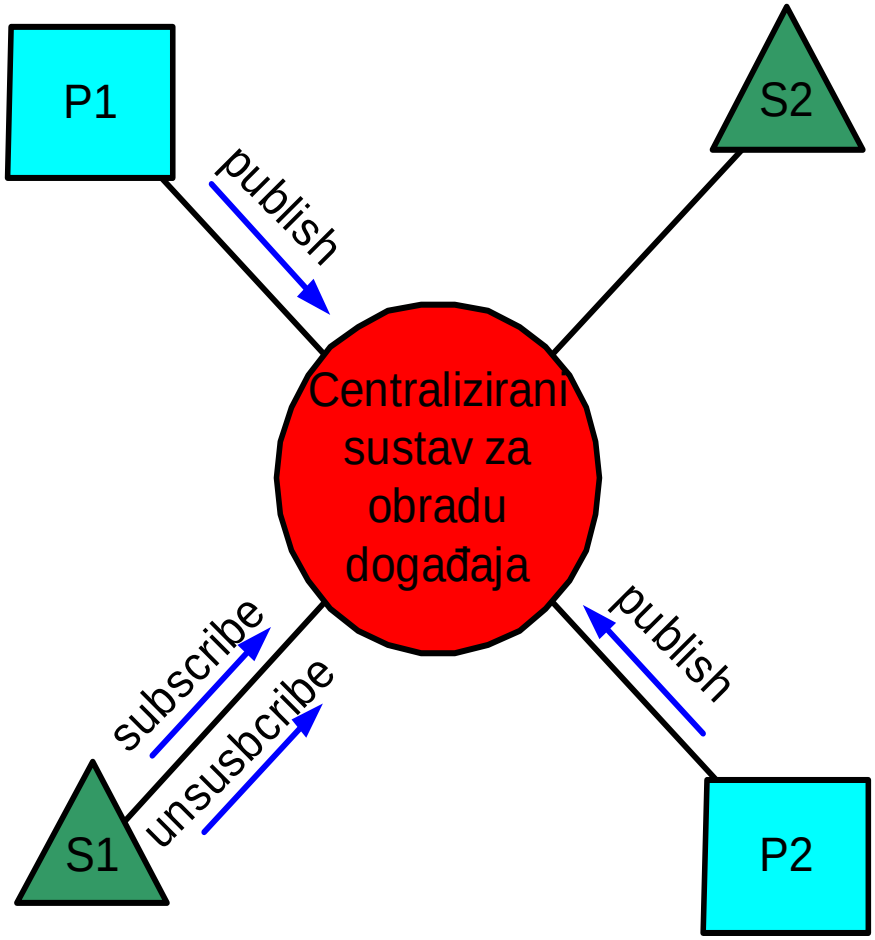
sub_2 = (category == “books” &
author == ”J.R.R. Tolkien”
& price < 20 EUR)



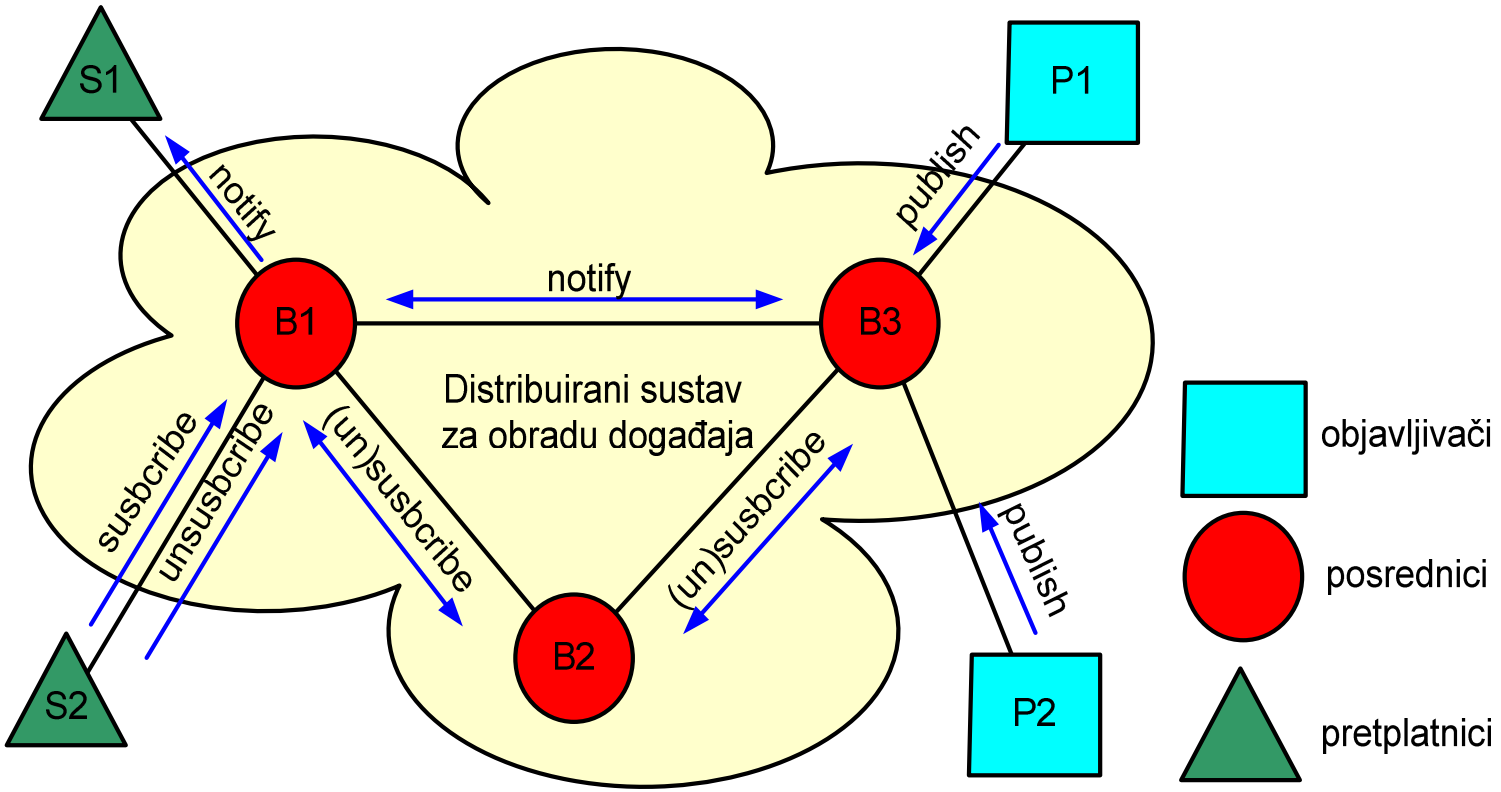
Arhitektura usluge objavi-pretplati

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

Centralizirana arhitektura



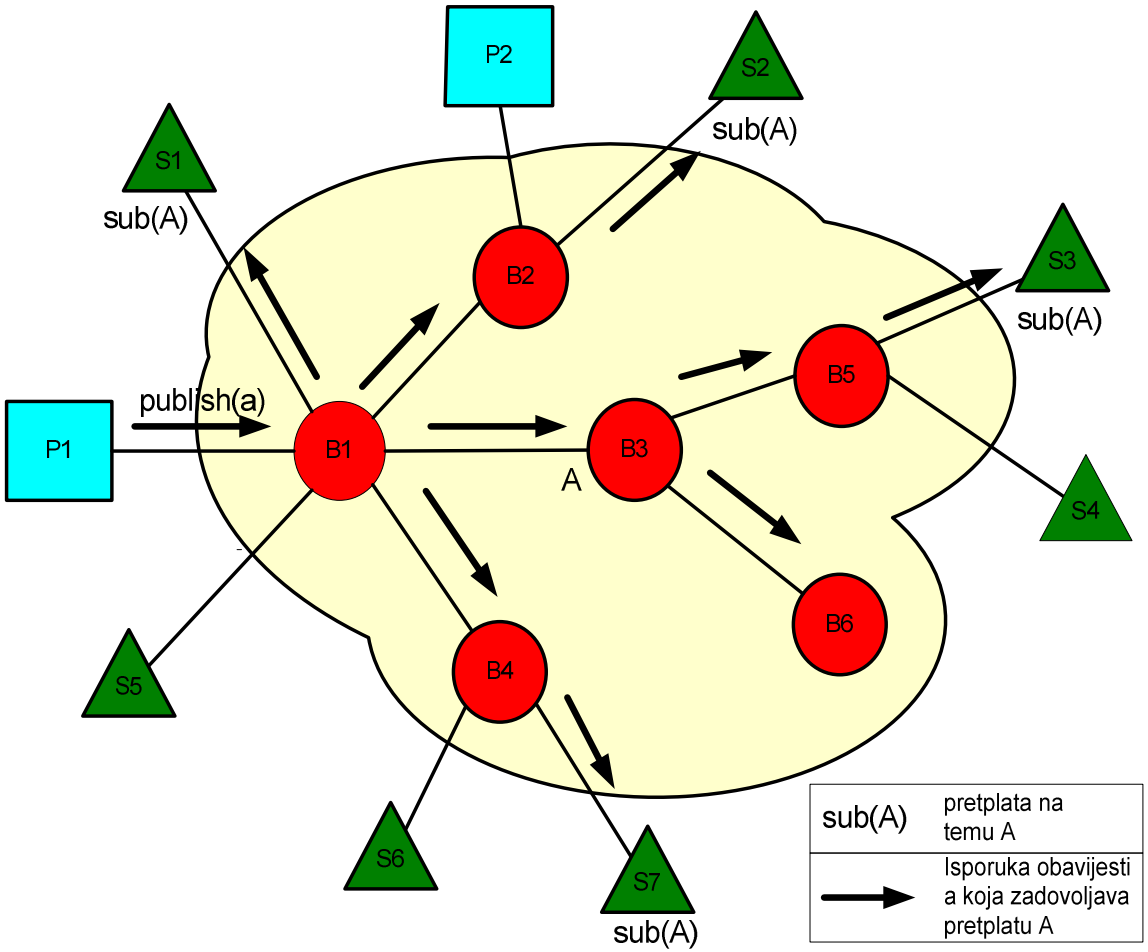
Distribuirana arhitektura



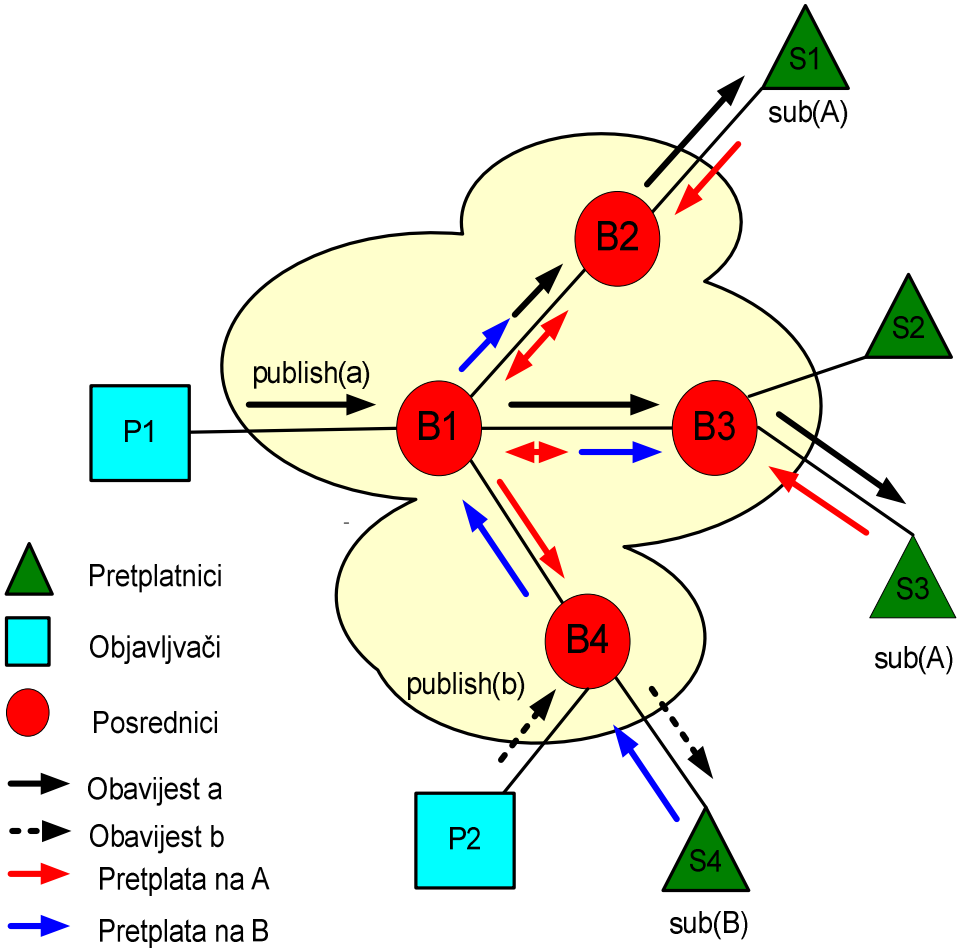
Osnovna načela usmjeravanja

- preplavljivanje
  - svaka primjena 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 zanimanju
  - omogućuje i smanjenje prometa u mreži posrednika zbog sprječavanja širenja obavijesti “nezainteresiranim” posrednicima

Preplavljivanje obavijestima



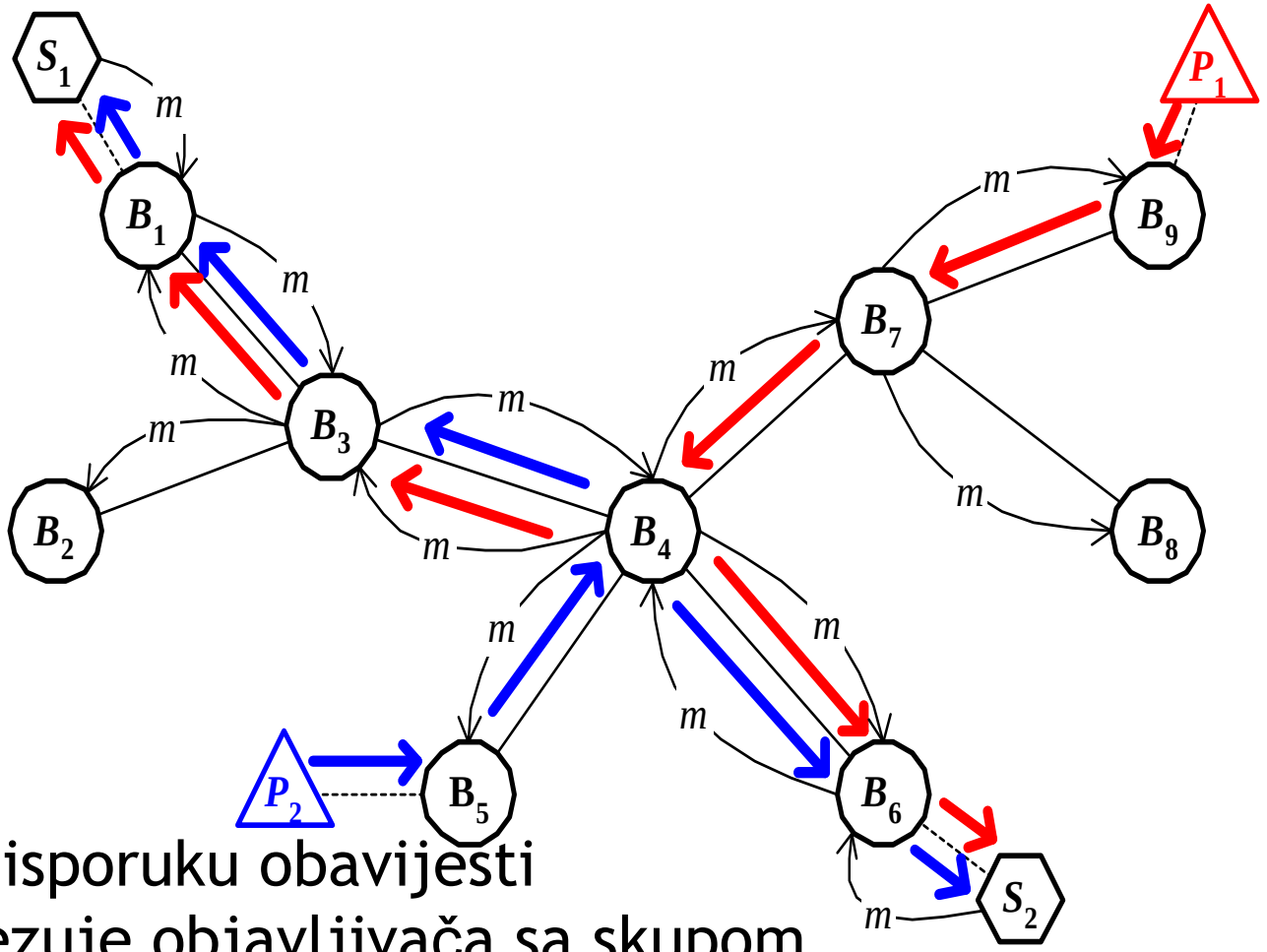
Preplavljivanje pretplatama



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

Tablica usmjeravanja posrednika *B*1

Reverse path forwarding



stablo za isporuku obavijesti

- povezuje objavljivača sa skupom pretplatnika
- 1 stablo za svakog objavljivača

Obilježja modela objavi-pretplati

- vremenska neovisnost
  - objavljivači i pretplatnici ne moraju istovremeno biti aktivni, posrednik pohranjuje poruku
- objavljivač ne mora znati identifikator pretplatnika (anonimnost), o tome se brine posrednik
- komunikacija je perzistentna
- asinkrona komunikacija
  - objavljivač šalje poruku i nastavlja procesiranje neovisno o odgovoru od strane odredišta
- pokretanje komunikacije na načelu *push*
  - objavljivač 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 objavljivača ili pretplatnika ne utječe na ostale strane u komunikaciji
- skalabilnost
  - distribuirana arhitektura

Sadržaj predavanja

- Java RMI
- Komunikacija porukama
- Model objavi-pretplati
- Java Message Service (JMS)

Java Message Service

- Sunova specifikacija 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.

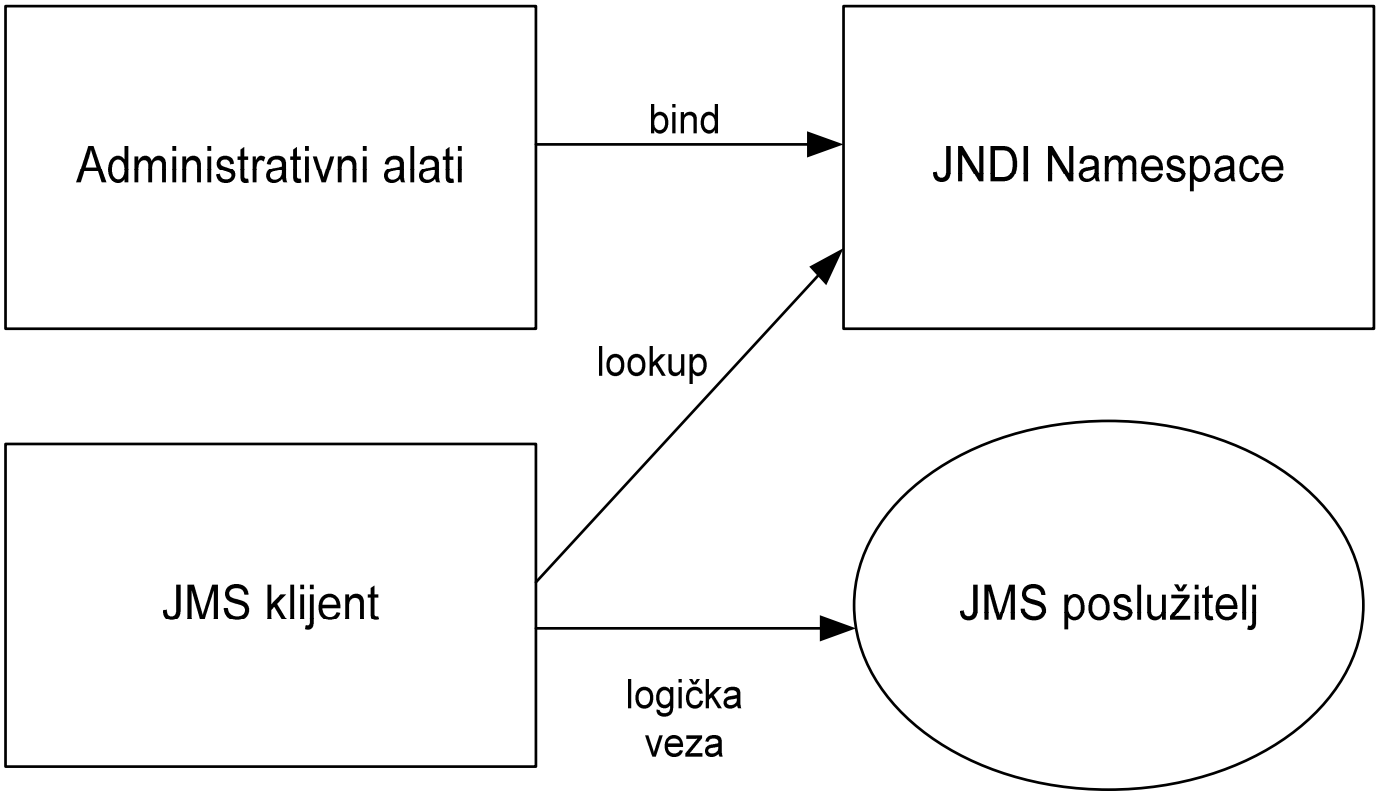
Arhitektura JMS-a (1)

- JMS poslužitelj (JMS provider)
  - 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
- Administrirani objekti

Administrirani objekti

- Kreira ih administrator
- Koriste ih klijenti
- Postoje dva tipa
  - ConnectionFactory
    - klijent ih koristi za uspostavljanje veze s JMS poslužiteljem
  - Destination
    - klijent ih koristi za specificiranje odredišta poruke koju šalje
- Administrator ih smješta u JNDI namespace (*Java Naming and Directory Interface*)

Arhitektura JMS-a (2)

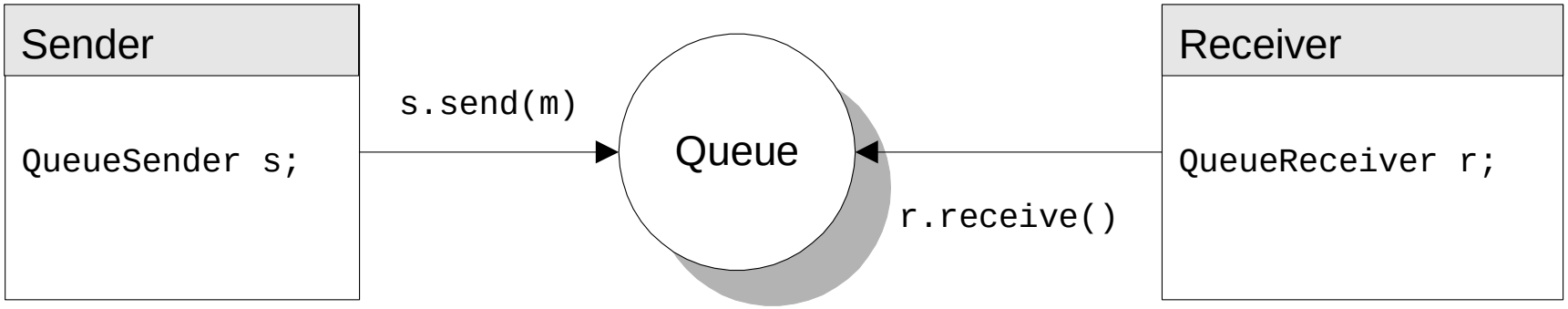


Modeli JMS-a

JMS implementira sljedeće modele za komunikaciju porukama i obavijestima

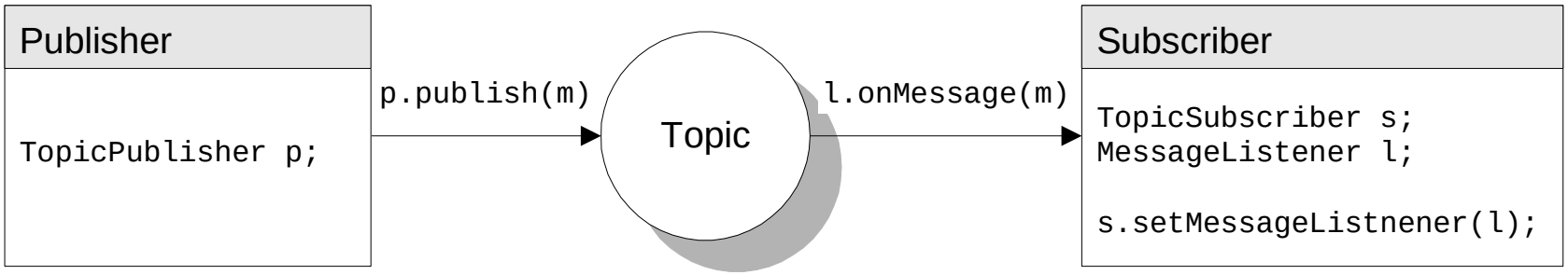
- *Point-to-point*
  - komunikacija porukama, jedna poruka za jedno odredište
- *Publish/subscribe*
  - objavi-pretplati, jedna poruka za skup zainteresiranih pretplatnika

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 da li u repu postoji poruka. Poziva `r.recieve()`.
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)`. Moguće je dodatno filtrirati poruke.

Sučelja JMS-a (1)

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

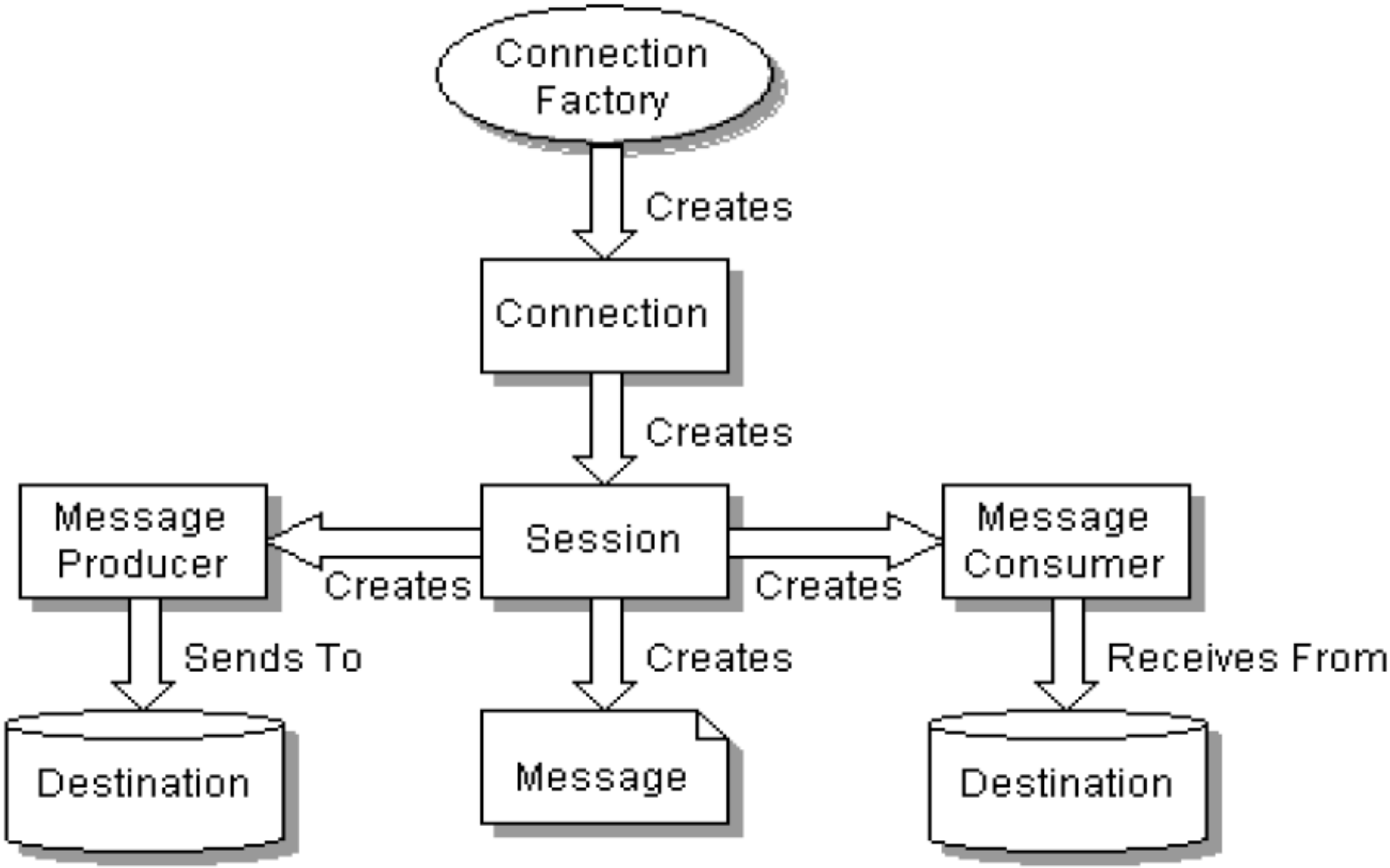
- **Destination**
  - administrirani objekt
  - predstavlja odredište - identitet ili adresu repa/teme.
- **ConnectionFactory**
  - administrirani objekt koji sadrži konfiguracijske parametre
  - klijenti ga koriste za stvaranje objekta *Connection*.
- **Connection**
  - predstavlja aktivnu konekciju 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**
  - jednostruka nit 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

Sučelja JMS-a (3)



Poruke JMS-a

- zaglavlje
  - skup definiranih polja koja sadrže vrijednosti za identificiranje i usmjeravanje poruke
- svojstva poruke
  - opcionalni parovi ime-vrijednost
  - standardni, specifični za aplikaciju, specifični za poslužitelja
  - 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* sadrži niz Javinih primitiva.
  - *MapMessage* kada tijelo sadrži skup parova ime-vrijednost.
  - *ObjectMessage* sadrži serijalizirani Java objekt.
  - *ByteMessage* za tijelo koje sadrži niz neinterpretiranih *byte*-ova.

Kvaliteta usluge

- 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
- JMS podržava perzistentne i neperzistentne poruke
- posebni JMS pretplatnici (*durable subscriber*) se mogu odjaviti iz sustava i ponovo prijaviti u sustav, primiti će sve perzistentne poruke objavljene u međuvremenu

Literatura - JMS

- J2EE, **Java Message Service (JMS)**
  http://java.sun.com/products/jms/
- **Java Message Service Tutorial**
  http://java.sun.com/products/jms/tutorial/

Queue Sender (1)


```
import javax.jms.*;
import javax.naming.*;

/**
 * Sends messages on the queue.
 */
public class Sender
{
  static Context ictx = null;

  public static void main(String[] args) throws Exception
  {
    System.out.println("Sends messages on the queue...");

    ictx = new InitialContext();
    Queue queue = (Queue) ictx.lookup("queue");
    QueueConnectionFactory qcf = (QueueConnectionFactory)
   ictx.lookup("qcf");
    ictx.close();
```

Queue Sender (2)


```
    QueueConnection qc = qcf.createQueueConnection();
    QueueSession qs = qc.createQueueSession(true, 0);
    QueueSender qsend = qs.createSender(queue);
    TextMessage msg = qs.createTextMessage();

    int i;
    for (i = 0; i < 10; i++) {
      msg.setText("Test number " + i);
      qsend.send(msg);
    }

    qs.commit();
    System.out.println(i + " messages sent.");
    qc.close();
  }
}
```

Queue Receiver (1)


```
import javax.jms.*;
import javax.naming.*;

/**
 * Requests messages on the queue.
 */
public class Receiver
{
  static Context ictx = null;

  public static void main(String[] args) throws Exception
  {
    System.out.println("Requests to receive messages...");

    ictx = new InitialContext();
    Queue queue = (Queue) ictx.lookup("queue");
    QueueConnectionFactory qcf = (QueueConnectionFactory)
   ictx.lookup("qcf");
    ictx.close();
```

Queue Receiver (2)


```
    QueueConnection qc = qcf.createQueueConnection();
    QueueSession qs = qc.createQueueSession(true, 0);
    QueueReceiver qrec = qs.createReceiver(queue);
    Message msg;
    qc.start();

    int i;
    for (i = 0; i < 10; i++) {
      msg = qrec.receive();
      if (msg instanceof TextMessage)
        System.out.println("Msg received: " + ((TextMessage)
  msg).getText());
      else if (msg instanceof ObjectMessage)
        System.out.println("Msg received: "
                           + ((ObjectMessage) msg).getObject());
      else
        System.out.println("Msg received: " + msg);
    }
    qs.commit();
    System.out.println(i + " messages received.");
    qc.close();
  }
}
```

Publisher (1)


```
import javax.jms.*;
import javax.naming.*;
/**
 * Publishes messages on the topic.
 */
public class Publisher
{
  static Context ictx = null;

  public static void main(String[] args) throws Exception
  {
    System.out.println();
    System.out.println("Publishes messages...");

    ictx = new InitialContext();

    Topic topic = (Topic) ictx.lookup("topic");
    TopicConnectionFactory tcf = (TopicConnectionFactory)
   ictx.lookup("tcf");
    ictx.close();
```

Publisher (2)


```
    TopicConnection tc = tcf.createTopicConnection();
    TopicSession ts = tc.createTopicSession(true, 0);
    TopicPublisher tpub = ts.createPublisher(topic);
    TextMessage msg = ts.createTextMessage();

    int i;
    for (i = 0; i < 10; i++) {
      msg.setText("Test number " + i);
      tpub.publish(msg);
    }

    ts.commit();

    System.out.println(i + " messages published.");

    tc.close();
  }
}
```

Subscriber (1)


```
import javax.jms.*;
import javax.naming.*;

/**
 * Subscribes and sets a listener to the topic.
 */
public class Subscriber
{
  static Context ictx = null;

  public static void main(String[] args) throws Exception
  {
    System.out.println();
    System.out.println("Subscribes and listens to the topic...");

    ictx = new InitialContext();
    Topic topic = (Topic) ictx.lookup("topic");
    TopicConnectionFactory tcf = (TopicConnectionFactory)
   ictx.lookup("tcf");
    ictx.close();
```

Subscriber (2)


```
    TopicConnection tc = tcf.createTopicConnection();
    TopicSession ts =
      tc.createTopicSession(false,
  javax.jms.Session.AUTO_ACKNOWLEDGE);
    TopicSubscriber tsub = ts.createSubscriber(topic);

    tsub.setMessageListener(new MsgListener());

    tc.start();

    System.in.read();

    tc.close();

    System.out.println();
    System.out.println("Subscription closed.");
  }
}
```

Message Listener (1)


```
import javax.jms.*;

/**
 * Implements the <code>javax.jms.MessageListener</code>
   interface.
 */
public class MsgListener implements MessageListener
{
  String ident = null;

  public MsgListener()
  {}

  public MsgListener(String ident)
  {
    this.ident = ident;
  }
```

Message Listener (2)


```
public void onMessage(Message msg){
  try {
    if (msg instanceof TextMessage) {
      if (ident == null)
        System.out.println(((TextMessage) msg).getText());
      else
        System.out.println(ident + ": " + ((TextMessage)
 msg).getText());
    }
    else if (msg instanceof ObjectMessage) {
      if (ident == null)
        System.out.println(((ObjectMessage) msg).getObject());
      else
        System.out.println(ident + ": " + ((ObjectMessage)
 msg).getObject());
    }
  }
  catch (JMSException jE) {
    System.err.println("Exception in listener: " + jE);
  }
 }
}
```