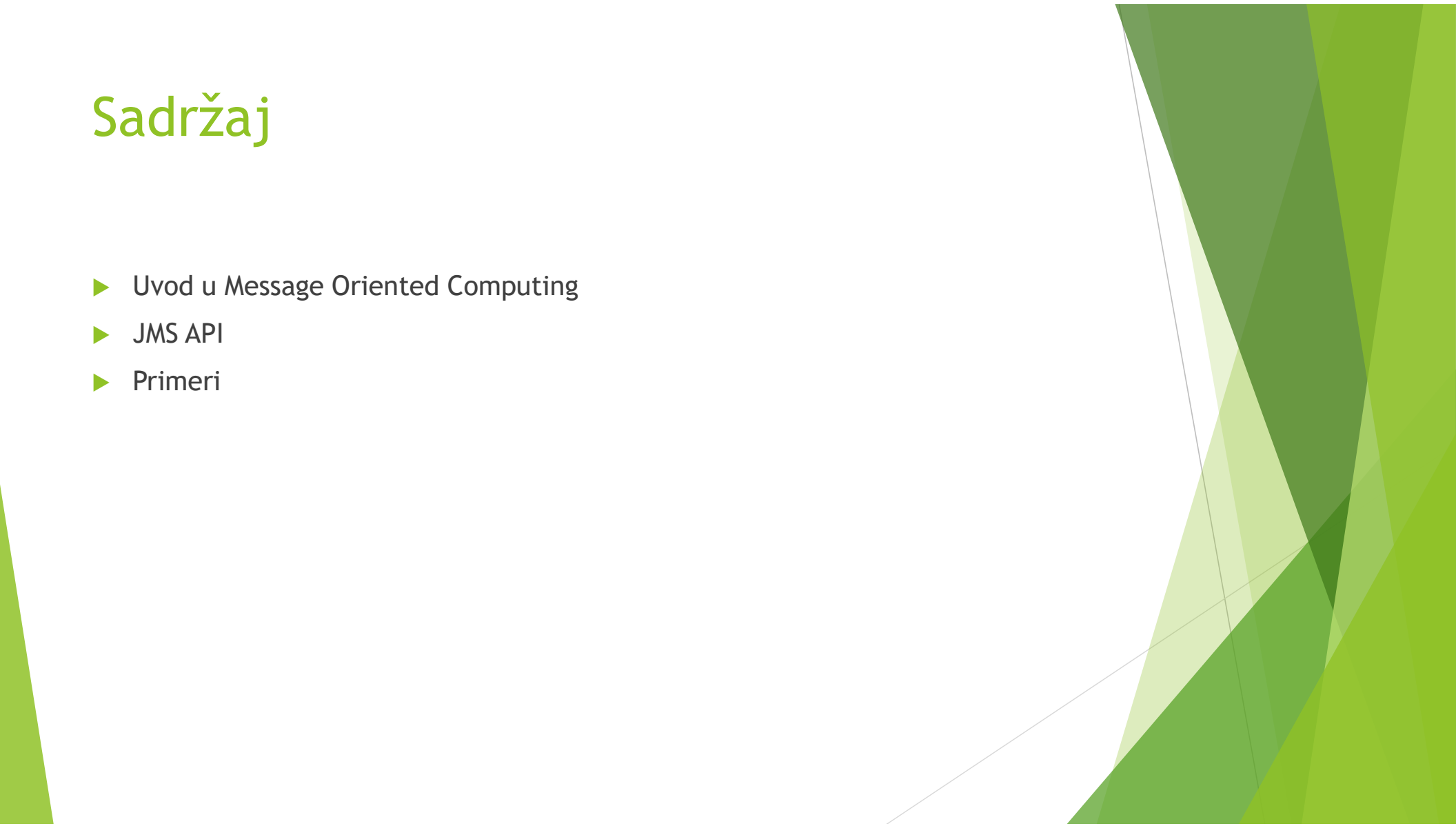
Distribuirni sistemi

java message service (JMS)

Sadržaj

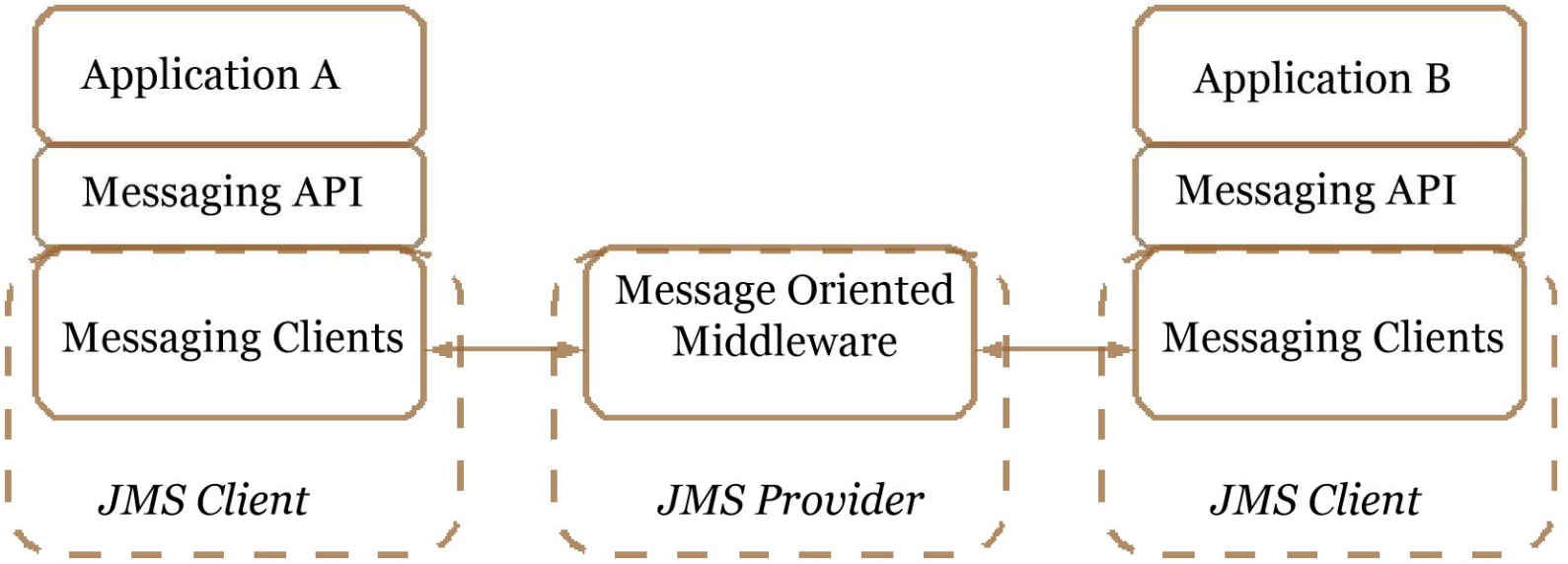
- Uvod u Message Oriented Computing
- JMS API
- Primeri



Message Oriented Middleware (MOM)

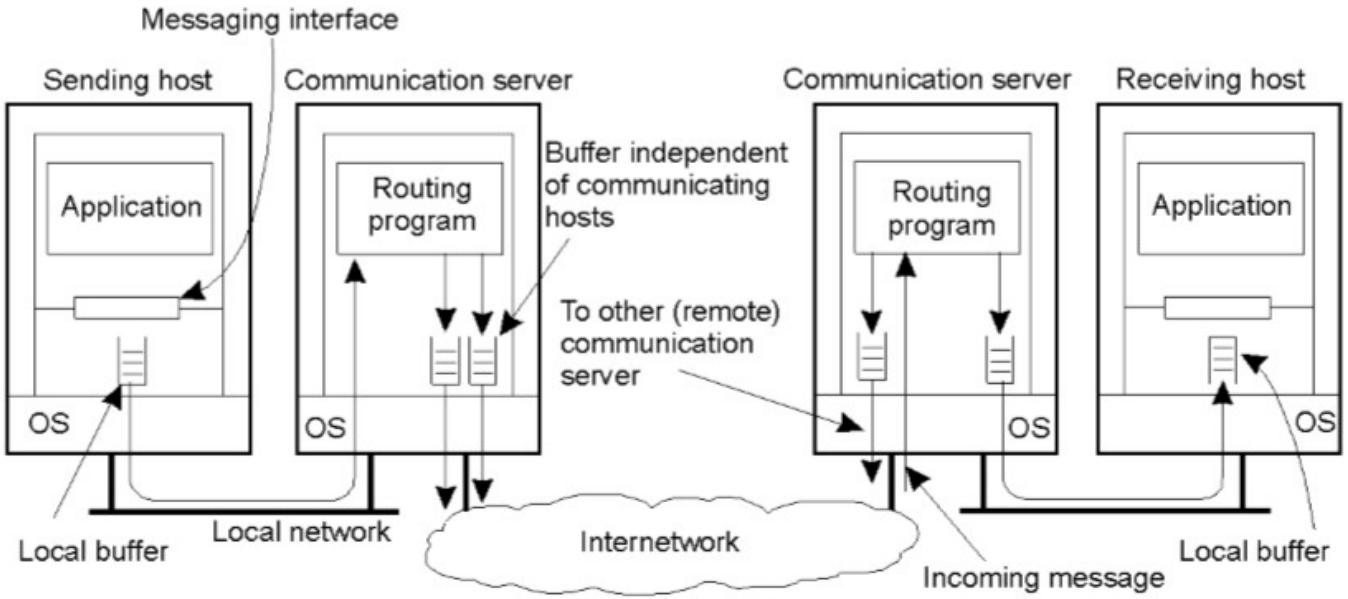
- Problemi integracije u Informacionim Sistemima
  - Asinhrona komunikacija
- Šta je MOM?
- Zašto bi smo koristili MOM?
  - Laka integracija heterogenih sistema
  - Dobro rešenje za uska grla u sistemskom dizajnu
  - Povećava ukupnu propusnu moć sistema
  - Poboljšava fleksibilnost sistemske arhitekture
  - Omogućava izgradnju geografski distribuirane sisteme

Message Oriented Middleware



MOM referentni model

- Infrastruktura slanja poruke je podržana kroz aplikativni sloj, preko nekoliko „komunikacijskih servera“.



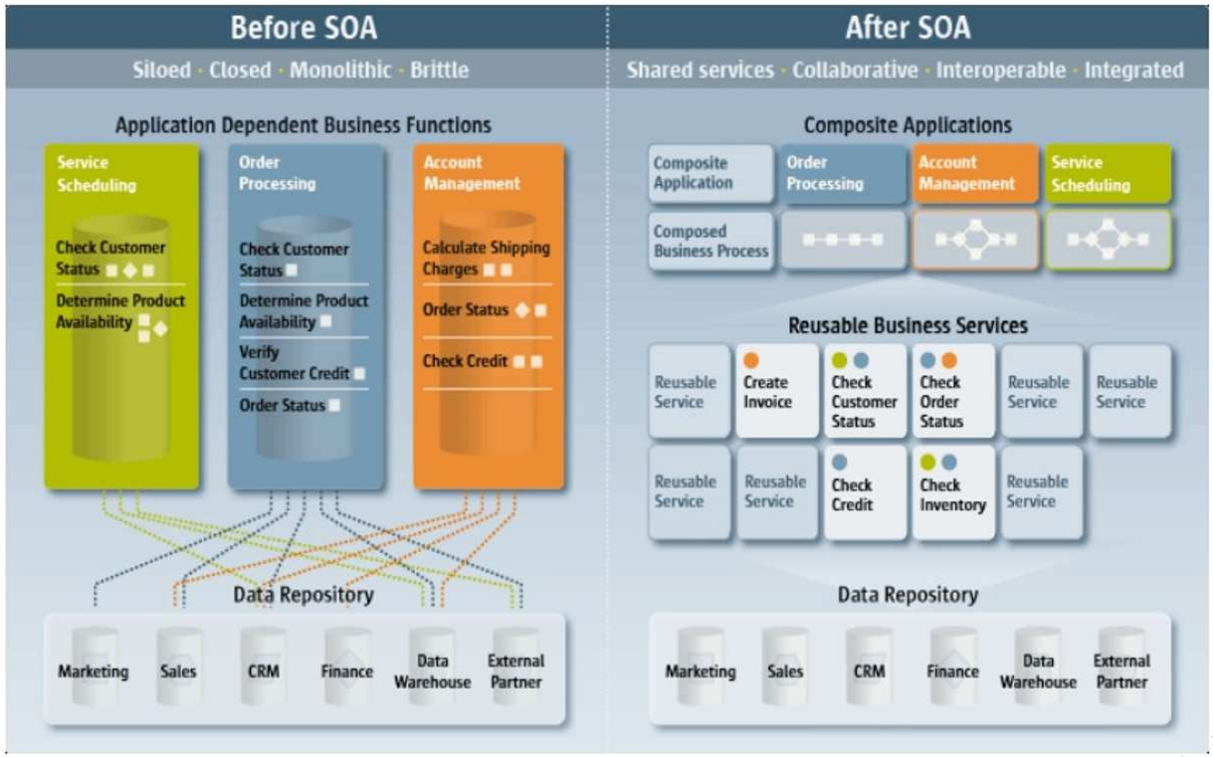
MOM – tipovi komunikacije

- Sinhroni – Asinhroni:
  - Sinhroni: pošiljilac je blokiran dok primalac ne dobije poruku (primi ili obradi),
  - Asinhroni: pošiljilac nastavlja sa radom nakon slanja poruke.
- Tranzijentni – Perzistentni:
  - Privremeni: pošiljilac i primaoc moraju biti aktivni da bi se poruka dostavila,
  - Trajni: poruka se čuva u komunikacionom sistemu sve dok njeno dostavljanje ne bude bilo moguće.
- U praksi postoje i alternative (kombinacije)

Serverski orijentisana arhitektura

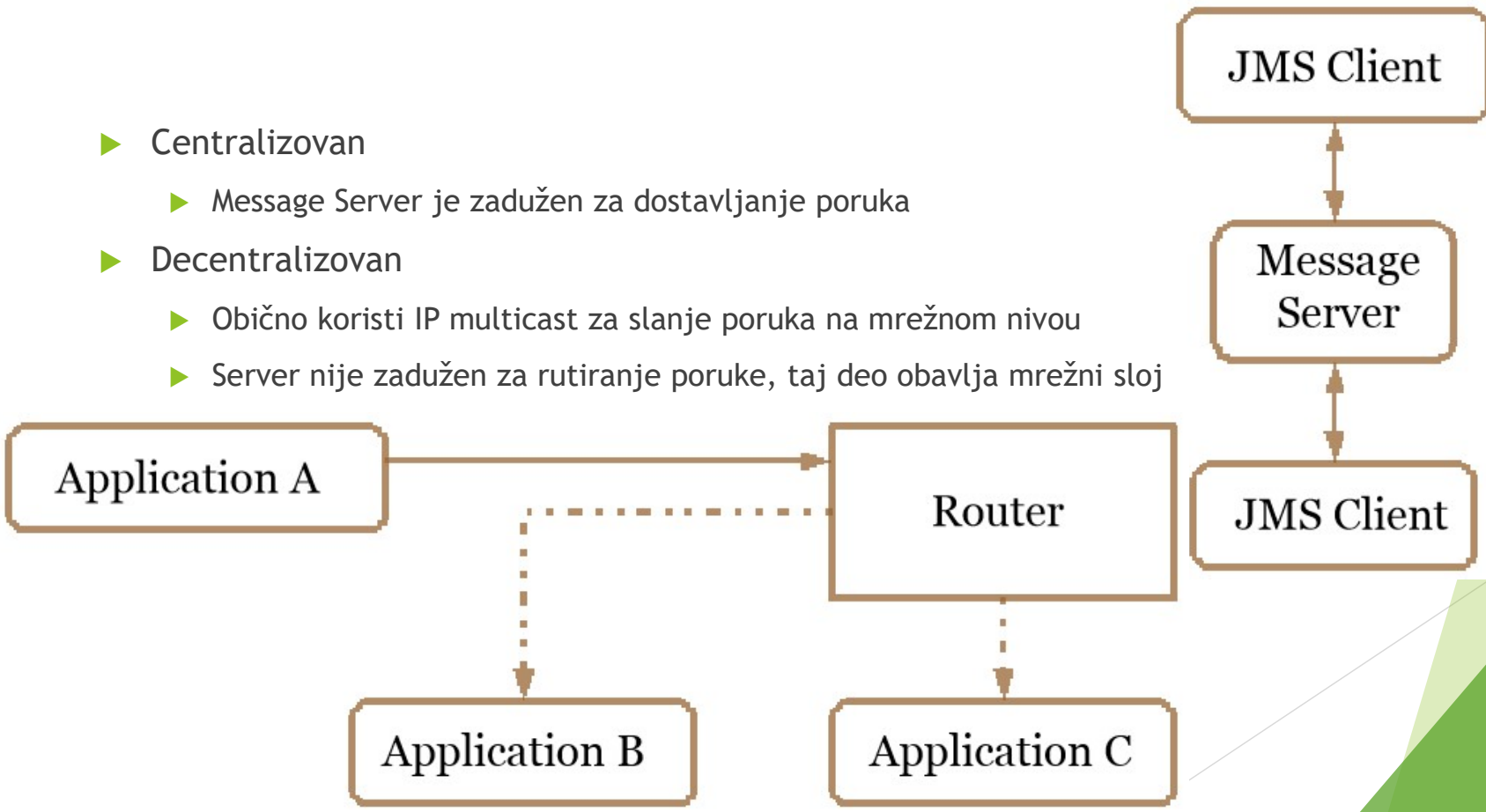
- Enterprise Service Bus (ESB) pristup
  - Poruke se razmenjuju asinhrono
  - Kositi se API za kreiranje poruka i slanje kroz MOM
  - Poruke predstavljaju autonomne celine, sadrže sve podatke i stanja neophodna za rad biznis logike
- Event-driven - pristup zanovan na događajima
  - Komunikacija se obavlja po asinhronoj šemi
  - Poruke se šalju na efikasan i robustan način
  - Self-described poruke – sadrže sve neophodne informacije koje omogućavaju „*prijemniku*“ da izvrši nezavisnu obradu
  - Loosly-coupled – sve komponente sistema su slabo spregnute

SOA primer


Before SOA
Siloed · Closed · Monolithic · Brittle
Application Dependent Business Functions
Service Scheduling
Check Customer Status
Determine Product Availability
Order Processing
Check Customer Status
Determine Product Availability
Verify Customer Credit
Order Status
Account Management
Calculate Shipping Charges
Order Status
Check Credit
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner
After SOA
Shared services · Collaborative · Interoperable · Integrated
Composite Applications
Composite Application
Order Processing
Account Management
Service Scheduling
Composed Business Process
Reusable Business Services
Reusable Service
Create Invoice
Check Customer Status
Check Order Status
Reusable Service
Reusable Service
Reusable Service
Reusable Service
Check Credit
Check Inventory
Reusable Service
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner

Arhitektura SOA

- Centralizovan
  - Message Server je zadužen za dostavljanje poruka
- Decentralizovan
  - Obično koristi IP multicast za slanje poruka na mrežnom nivou
  - Server nije zadužen za rutiranje poruke, taj deo obavlja mrežni sloj

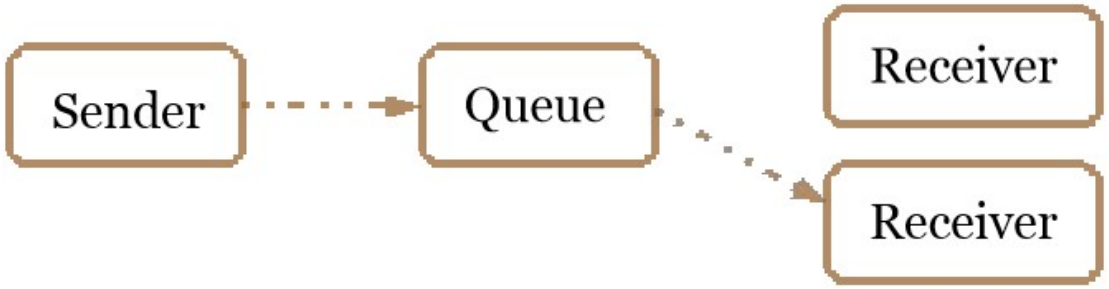


Modeli komunikacije

- Sinhrona komunikacija
  - Oba učesnika u komunikaciji moraju biti aktivni,
  - Pošiljilac dobija potvrdu o prijemu od prijemnika,
  - Blokirajući pozivi,
  - Podržava scenario gde je autorizacija obavezna (npr. Upotreba kreditnih kartica)
- Asinhrona komunikacija
  - U toku komunikacije ne moraju oba učesnika da budu aktivna,
  - Konfirmacija nije neophodna,
  - Nema blokirajućih poziva,
  - Korisno kada se zahteva procesiranje masovne komunikacije,
  - Omogućava efikasnu upotrebu hardverskih resursa.

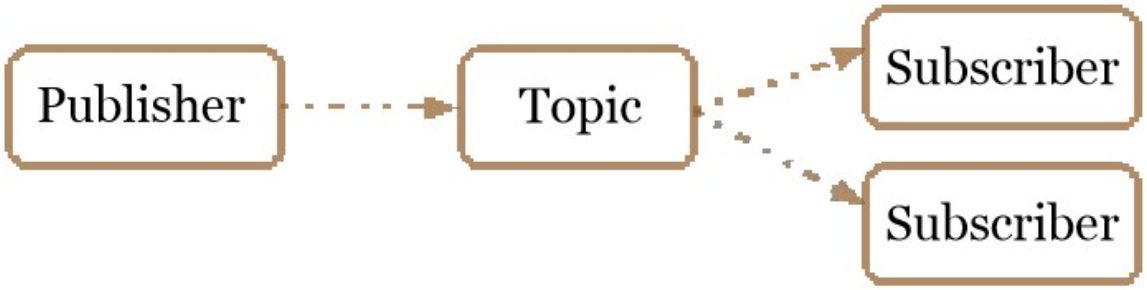
Point-to-Point

- Pošiljioci (*senders*) i primaoci(*receivers*) komuniciraju preko virtualnih kanala, poznatih pod nazivom redovi (*queues*), na asinhron i sinhron način,
- Poruku dobija isključivo jedan primaoc, komunikacija 1-1.
- Pošiljioc može zahtevati novu poruku u bilo kom trenutku.
- Servisi su jače spregnuti, pošiljioc obično zna primaoca i koje informacije primaoc očekuje.



Publish-and-Subscribe

- Poruke se objavljuju (*publish)* preko virtuelnih kanala nazvanih teme (*topics*),
- Oni koji objavljuju podatke (*producer*) nazivaju se *publishers*, dok oni koji koriste (obrađuju) podatke (*consumer*) nazivaju se *subcribers*,
- Poruke se brodkastuju svim *subcribers*-ima, svaki *subcriber* dobija kopiju poruke, komunikacija 1-više
- Servisi su slabije spregnuti u odnosu na P2P model



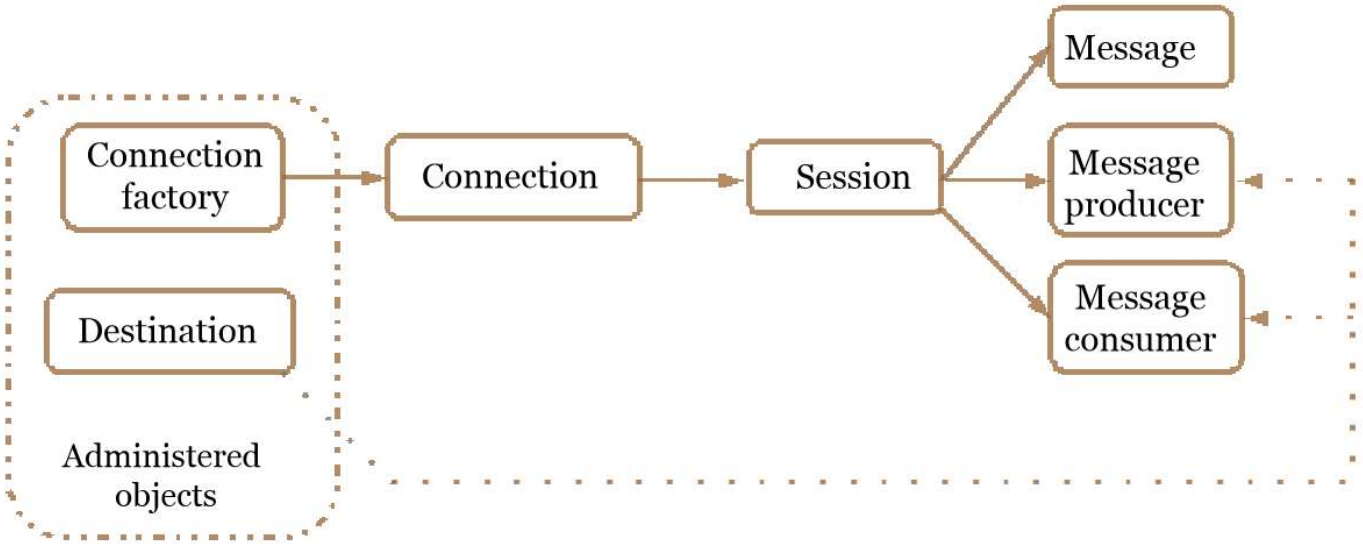
Java Message Service (JMS)

- JMS predstavlja API za kreiranje poruka,
- Napravljen od strane *Sun Microsystem*,
- Predstavlja abstrakt API, ne kompletan sitem.

- API sadrži:
  - General API (za komunikaciju i sa queues i sa topics)
  - Point-to-Point API,
  - Publish-and-Subscribe API.

JMS General API

- Glavni interfejsi:
  - ConnectionFactory, Destination, Connection, Session, Message, MessageProducer, MessageConsumer



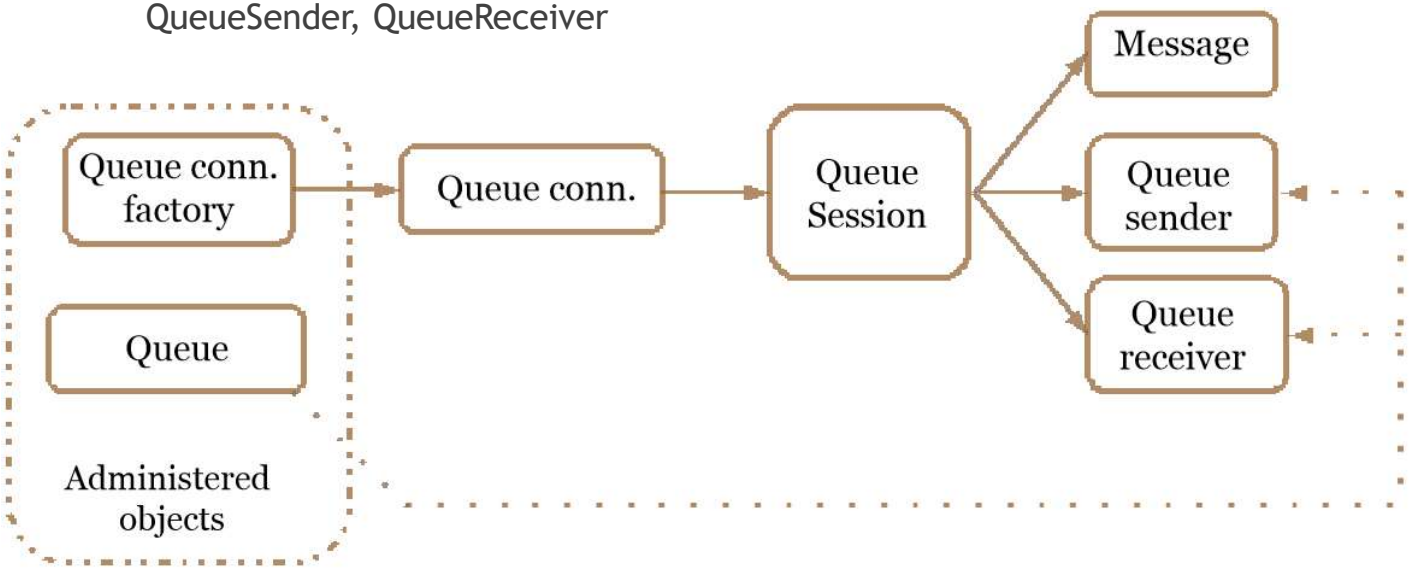
Primer 1.1

```
import javax.jms.*; import javax.naming.*;
public class Producer {
 static Context ictx = null;
 public static void main(String[] args) throws Exception {
  ictx = new InitialContext();
  Queue queue = (Queue) ictx.lookup("queue");
  Topic topic = (Topic) ictx.lookup("topic");
  ConnectionFactory cf = (ConnectionFactory) ictx.lookup("cf");
  ictx.close();
  Connection cnx = cf.createConnection();
  Session sess = cnx.createSession(false,
Session.AUTO_ACKNOWLEDGE);
  MessageProducer prod = sess.createProducer(null);
  TextMessage msg = sess.createTextMessage();
  int i;
  for (i = 0; i < 10; i++) {
   msg.setText("Test number " + i);
   prod.send(queue, msg);
   prod.send(topic, msg);
  }
 cnx.close();
 }
}
```

```
import javax.jms.*; import javax.naming.*;
public class Subscriber {
 static Context ictx = null;
 public static void main(String[] args) throws Exception {
  ictx = new InitialContext();
  Queue queue = (Queue) ictx.lookup("queue");
  Topic topic = (Topic) ictx.lookup("topic");
  ConnectionFactory cf =       (ConnectionFactory) ictx.lookup("cf");
  ictx.close();
  Connection cnx = cf.createConnection();
  Session sess = cnx.createSession(false, Session.AUTO_ACKNOWLEDGE);
  MessageConsumer recv = sess.createConsumer(queue);
  MessageConsumer subs = sess.createConsumer(topic);
  recv.setMessageListener(new MsgListener("Queue"));
  subs.setMessageListener(new MsgListener("Topic"));
  cnx.start();
  System.in.read();
  cnx.close();
 }
}
```

JMS P2P API

- Glavni interfejsi:
  - QueueConnectionFactory, Queue, QueueConnection, QueueSession, Message, QueueSender, QueueReceiver



JMS P2P Implementacija

- Producer
  - Obtain reference to QueueConnectionFactory,
  - Get reference to Queue,
  - Create QueueConnection,
  - Create QueueSession,
  - Create QueueSender,
  - Create Message,
  - Send Message.

- **Consumer**
  - Obtain reference to QueueConnectionFactory,
  - Get reference to Queue,
  - Create QueueConnection,
  - Create QueueSession,
  - Create QueueReceiver,
  - Wait for message, implement interface MessageListener.

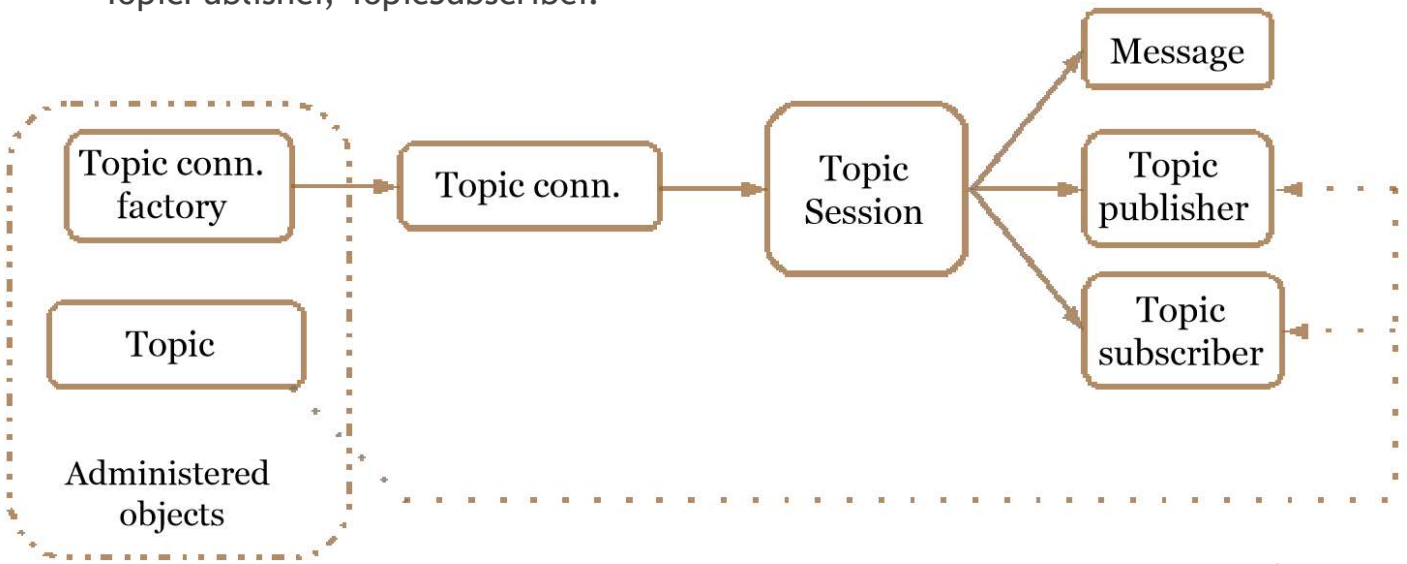
Primer 2.1

```
import javax.jms.*; import javax.naming.*;
public class Sender {
 static Context ictx = null;
 public static void main(String[] args) throws  Exception {
  ictx = new InitialContext();
  Queue queue = (Queue) ictx.lookup("queue");
  QueueConnectionFactory qcf = (QueueConnectionFactory)
ictx.lookup("qcf");
  ictx.close():
  QueueConnection qc = qcf.createQueueConnection();
  QueueSession qs = qc.createQueueSession(false,
Session.AUTO_ACKNOWLEDGE);
  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); }
  qc.close();
 }
}
```

```
import javax.jms.*; import javax.naming.*;
public class Receiver {
 static Context ictx = null;
 public static void main(String[] args) throws Exception {
  ictx = new InitialContext();
  Queue queue = (Queue) ictx.lookup("queue");
  QueueConnectionFactory qcf = (QueueConnectionFactory) ictx.lookup("qcf");
  ictx.close();
  QueueConnection qc = qcf.createQueueConnection();
  QueueSession qs =
qc.createQueueSession(false,Session.AUTO_ACKNOWLEDGE);
  QueueReceiver qrec = qs.createReceiver(queue);
  TextMessage msg;
  qc.start():
  int i;
  for (i = 0; i < 10; i++) {
   msg = (TextMessage) qrec.receive();
   System.out.println("Msg received: " + msg.getText());  }
  qc.close();
 }
}
```

JMS Publish-and-Subscribe API

- Glavni interfejsi:
  - TopicConnectionFactory, Topic, TopicConnection, TopicSession, Message, TopicPublisher, TopicSubscriber.



JMS Publish-and-Subscribe Implementacija

- Producer
  - Obtain reference to *TopicConnectionFactory*,
  - Get reference to *Topic*,
  - Create *TopicConnection*,
  - Create *TopicSession*,
  - Create *TopicPublisher*,
  - Create *Message*,
  - Send *Message*.
- **Consumer**
  - Obtain reference to *TopicConnectionFactory*,
  - Get reference to *Topic*,
  - Create *TopicConnection*,
  - Create *TopicSession*,
  - Create *TopicSubscriber*,
  - Wait for message, implement interface *MessageListener*.

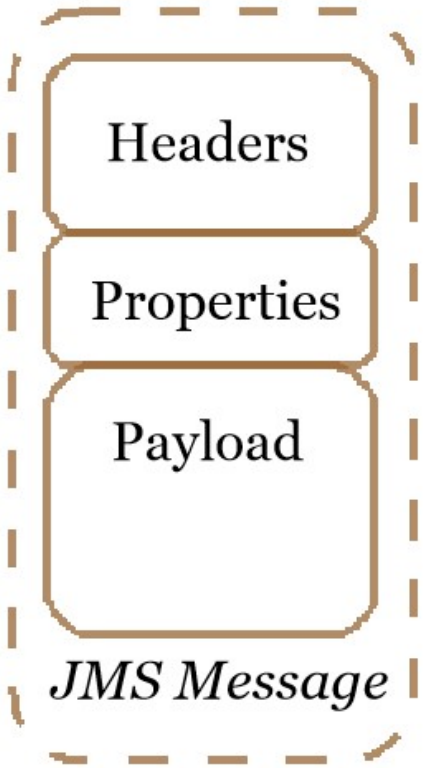
Primer 3.1

```
import javax.jms.*; import javax.naming.*;
public class Publisher {
 static Context ictx = null;
 public static void main(String[] args) throws Exception {
  ictx = new InitialContext();
  Topic topic = (Topic) ictx.lookup("topic");
  TopicConnectionFactory tcf = (TopicConnectionFactory)
ictx.lookup("tcf");
  ictx.close();
  TopicConnection tc = tcf.createTopicConnection();
  TopicSession ts = tc.createTopicSession(true,
Session.AUTO_ACKNOWLEDGE);
  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();
  tc.close();
 }
}
```

```
import javax.jms.*; import javax.naming.*;
public class Subscriber {
 static Context ictx = null;
 public static void main(String[] args) throws Exception {
 ictx = new InitialContext();
 Topic topic = (Topic) ictx.lookup("topic");
 TopicConnectionFactory tcf = (TopicConnectionFactory) ictx.lookup("tcf");
 ictx.close();
 TopicConnection tc = tcf.createTopicConnection();
 TopicSession ts = tc.createTopicSession(true, Session.AUTO_ACKNOWLEDGE);
 TopicSubscriber tsub = ts.createSubscriber(topic);
 tsub.setMessageListener(new MsgListener());
 tc.start();
 System.in.read();
 tc.close();
 }
}
class MsgListener implements MessageListener {
 String id;
 public MsgListener() {id = "";}
 public MsgListener(String id) {this.id = id;   }
 public void onMessage(Message msg) {
  TextMessage tmsg = (TextMessage) msg;
  try { System.out.println(id+": "+tmsg.getText());  }
  catch (JMSException jE) { jE.printStackTrace(); }
 }
}
```

JMS Message API

- Osnovna i najvažnija klasa
  - Svi podaci i događaji se prenose preko *Message* objekata,
  - NAPOMENA: poruka ne kaže primaocu šta da radi!
- Sastoji se iz tri dela:
  - Zaglavlje,
  - Propertiji,
  - Podaci.



JMS Zaglavlje poruke

- Osnovne informacije:
  - Potoje dve grupe propertija, podeljene po odgovornosti: podešavaju diveloperi, podešava JMS automatski,
  - Obe grupe su dostupne preko *set* i *get* metoda.
- Podrazumevano zaglavlje:
  - JMSDestination – odredište poruke,
  - JMSDeliveryMode – definiše nečin dosave (persistent/not-persistent),
  - JMSPriority – podešava *producer*, 0-4 normalan and 5-9 neodložan.
- Podisivo (custom) zaglavlje:
  - JMSReplyTo – definiše odredište odgovora,
  - JMSType – opciono zaglavlje, definiše tip poruke.

JMS Propertiji poruke

- Osnovne informacije:
  - Tri tipa: Aplikacijsko specifični, JMS defnisani, Provider specifični,
  - Koriste se kao dodatna zglavlja poruci,
  - Vrednosti propertija mogu biti: Sting, boolean, bzte, double, int, long ili float.
- Aplikacijsko specifični:
  - Definiše bilo koje dodatne podatke koja mogu biti pridodata poruci.
- JMS defnisani propertiji:
  - Automatski setovani od strane JMS providera,
  - JMSXGroupID, JMSXGroupSeq, JMSXUserID, JMSXAppID ...
- Provider specifični propertiji:
  - Automatski setovani od strane JMS providera,
  - Dostavlja lične informacije JMS providera.

JMS Teret poruke (Payload)

- Osnovne informacije:
  - JMS Provider mora da podrži 6 tipova poruka: Message, TextMessage, StreamMessage, MapMessage, ObjectMessage i BytesMessage,
  - Interfejs *Message* može biti proširen da podrži i druge tipove poruka (npe XML).
- Može se slati i čist Message tip
  - Ukoliko je potrebno da se pošalje događa.

JMS Filtriranje poruke (Message Selectors)

- Osnovne informacije:
  - Filtriranje poruka omogućava smanjenje broja poslatih poruka,
  - Moguće je i filtriranje na *producer* strani umesto na klijentskoj.

```
...
topic = (Topic)ctx.lookup(topicName);
...
String filter = "your condition";
TopicSubscriber subscriber =    session.createSubscriber(topic, filter, true);
...
```

- Filtriranje kod P2P:
  - Kod *queues*, jednom filtrirana poruka se uklnja iz reda i nije više dostupna ostalima,
  - Mogu se koristiti prioriteti, prvo se filtriraju poruke sa višim prioritetom.

JMS Filtriranje poruke (Message Selectors)

- Filteri se mogu primeniti na *consumer*-e:
  - QueueReceiver, QueueBrowser, or TopicSubscriber,
  - Zaglavlje poruka i propertiji mogu se koristiti u generisanju filtera,
  - Bez pristupa samom sadržaju poruke.
- Generisanje filtera
  - Za generisanje filtera koristi se SQL-92 sintaksa uslovnih (kondicionalih) izraza,
  - Promenljive za poređenje se koriste iz zaglavlja i propertija

    (npr. Name = 'distribuirani' AND JMSPriority > 5),
  - Literali su hardkodirani u filter,
  - Poređenje uvek vraća **Boolean** vrednost. Podržava sve algebarske komparatore, kao i operatore **LIKE**, **BETWEEN**, **IN**, **NOT** i **IS NULL**.

Primer 4.1

Primer 4.2