JMS
-- extra reading materials

University of Hong Kong

JMS: Messages Explained

- A message typically consists of a header and a body.
- The message header contains vendor-specified values, but could also contain application-specific data as well.
  - Headers are typically name/value pairs.
- The body contains data; the type of the data is defined by the specification.
  - Text
  - A serialized Java object
  - One of a number of other types of data.

Publisher Sample

1. Perform a JNDI API lookup of the TopicConnectionFactory and topic
   - `topic = (Topic) jndiContext.lookup(topicName);`
2. Create a connection and a session
   - `topicConnection = topicConnectionFactory.createTopicConnection();`
   - `topicSession = topicConnection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);`
3. Create a TopicPublisher
   - `topicPublisher = topicSession.createPublisher(topic);`
4. Create a TextMessage
   - `Message = topicSession.createTextMessage();`
   - `message.setText("This is message " + (i + 1));`
5. Publishe one or more messages to the topic
   - `topicPublisher.publish(message);`
6. Close the connection, which automatically closes the session and TopicPublisher

Subscriber Sample

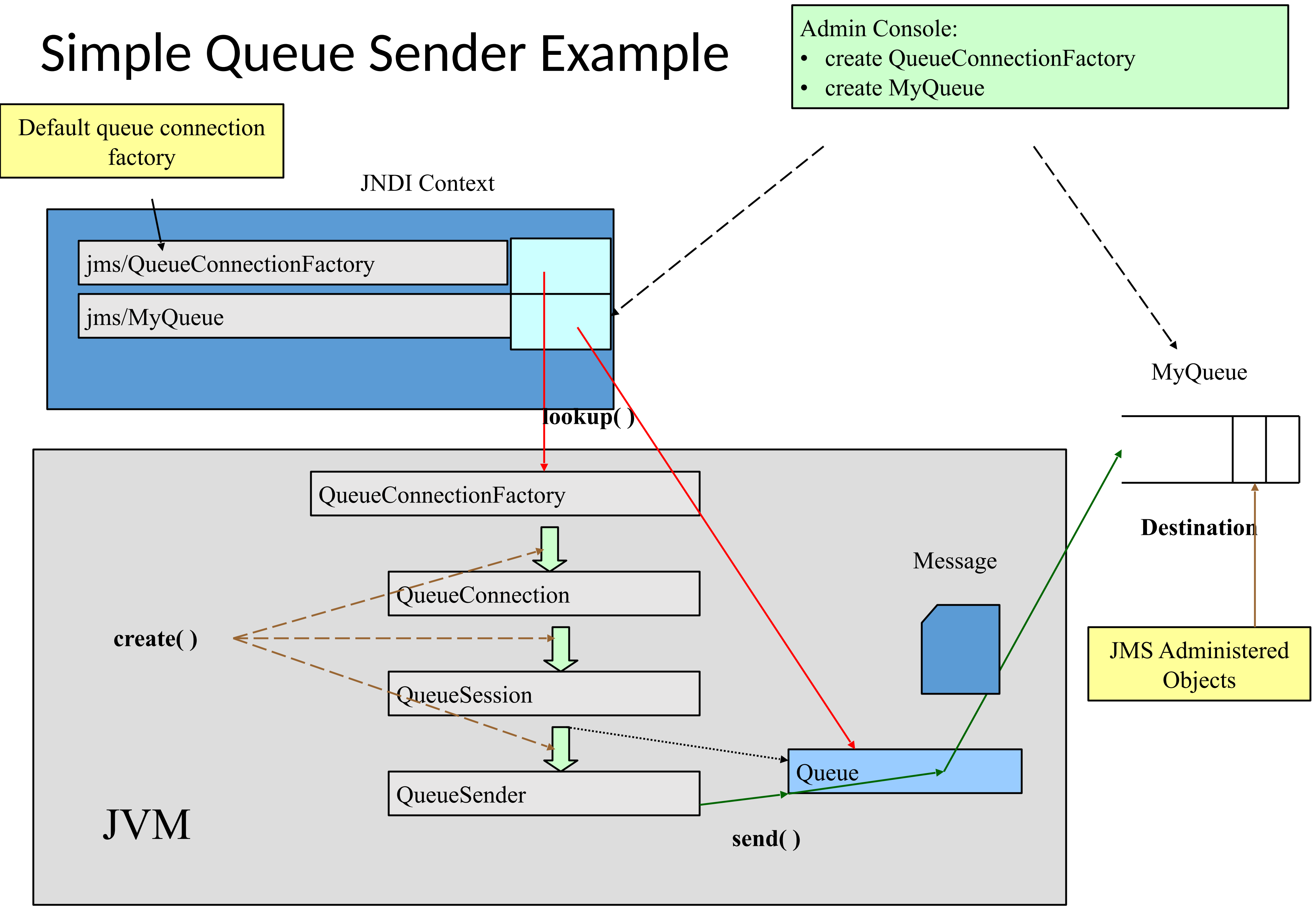
1. Perform a JNDI API lookup of the TopicConnectionFactory and topic (same as publisher)
2. Create a connection and a session (same as publisher)
3. Create a TopicSubscriber
   - `topicSubscriber = topicSession.createSubscriber(topic);`
4. Create an instance of the TextListener class and registers it as the message listener for the TopicSubscriber
   - `topicListener = new TextListener();`
   - `topicSubscriber.setMessageListener(`**`topicListener`**`);`
5. Start the connection, causing message delivery to begin
   - `topicConnection.start();`
6. Close the connection, which automatically closes the session and TopicSubscriber
   - `topicConnection.close();`

TextListener Sample

```
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 of wrong type: " +
                message.getClass().getName());
        }
    } catch (JMSException e) {
        System.out.println("JMSException in onMessage(): " + e.toString());
    } catch (Throwable t) {
        System.out.println("Exception in onMessage():" + t.getMessage());
    }
}
```

Simple Queue Sender Example



Simple Queue Sender: env setup

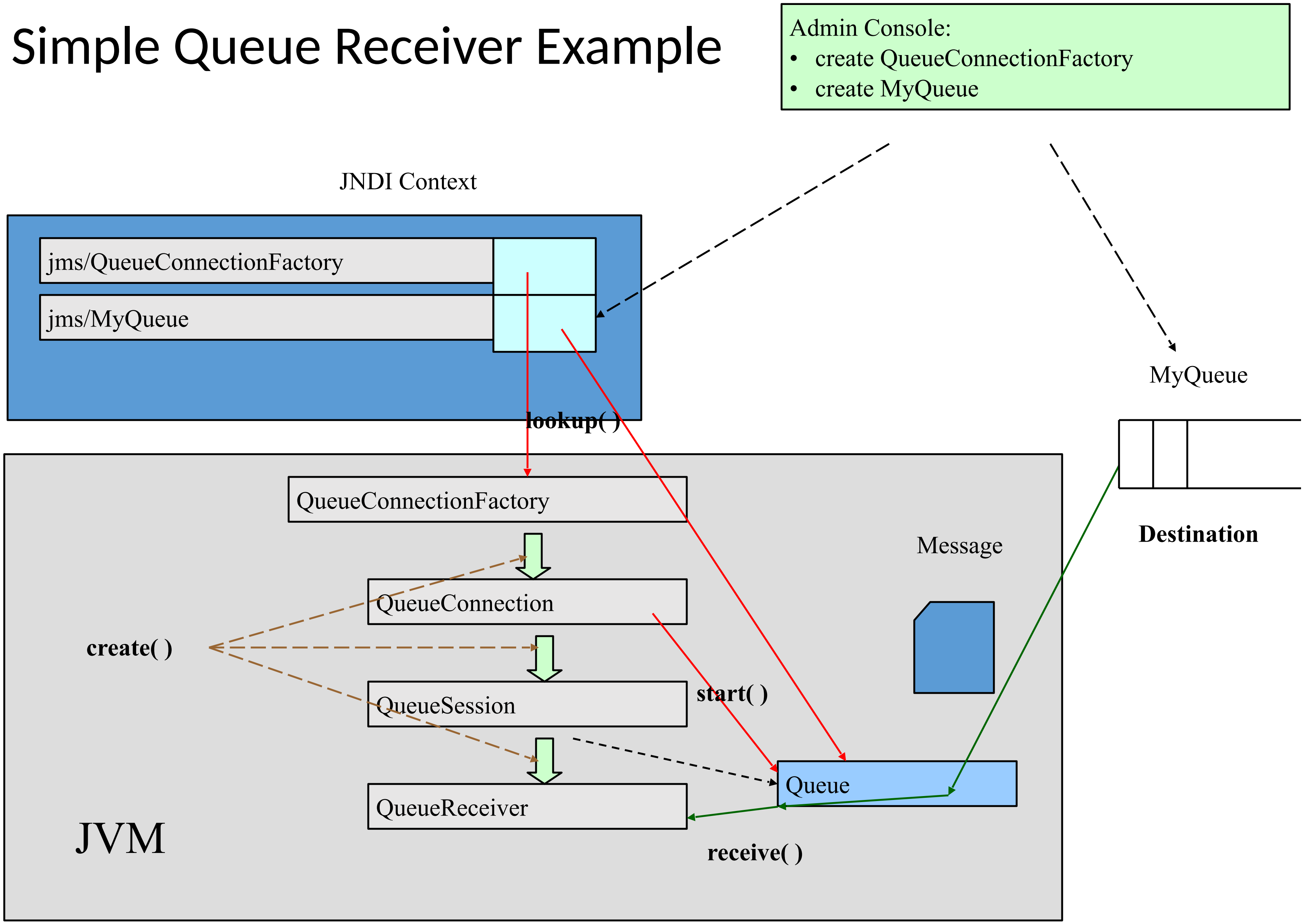
```
// import javax.jms.*,  javax.naming.*;
public class SimpleQueueSender {
     public static void main(String [ ] args) {
          String queueName = “jms/MyQueue”;
          Context ctx = null;
          QueueConnectionFactory qcf = null;
          QueueConnection qc = null;
          QueueSession qsess = null
          Queue q = null;
          QueueSender qsender = null;
          ctx = new InitialContext( );
          qcf = (QueueConnectionFactory)ctx.lookup(“jms/QueueConnectionFactory”);
          q = (Queue) ctx.lookup(queueName);
          qc = qcf.createQueueConnection();
          qsess = qc.createQueueSession(false, Session.AUTOACKNOWLEDGE);
          qsender = qsess..createSender(q);
```

Exceptions handling are not included

Simple Queue Sender: sending messages

```
        TextMessage message = null;
        message = qsess.createTextMessage();
        for ( int i=0; i<NUM_MSGS; i++ ) {
            message.setText(“This is message” + (i+1) );
            System.out.println(“Sending message:” + message.getText() );
            qsender.send(message);
        }
        // Send a non-textual control msg for end of messages
        qsender.send(qsess.createMessage());
    } }
```

Simple Queue Receiver Example



Simple Queue Receiver: env setup

```
// import javax.jms.*,  javax.naming.*;
public class SimpleQueueReceiver {
    public static void main(String [ ] args) {
        String queueName = “jms/MyQueue”;
        Context ctx = null;
        QueueConnectionFactory qcf = null;
        QueueConnection qc = null;
        QueueSession qsess = null
        Queue q = null;
        QueueSender qsender = null;
        ctx = new InitialContext( );
        qcf = (QueueConnectionFactory)ctx.lookup(“jms/QueueConnectionFactory”);
        q = (Queue) ctx.lookup(queueName);
        qc = qcf.createQueueConnection();
        qsess = qc.createQueueSession(false, Session.AUTOACKNOWLEDGE);
        qReceiver = qsess..createReceiver(q);
        qc.start( );
```

Exceptions handling are not included

Simple Queue Receiver: receiving messages

```
    TextMessage msg = null;
    while (true) {
        Message m = qReceiver.receive(1);
        if ( m != null ) {
            if ( m instanceof TextMessage) {
                msg = (TextMessage) m;
                System.out.println(“Reading message” + msg.getText() );
            }
            else { break; }
        }
    }
} }
```

JMS Destinations

- A provider-independent representation of a message delivery point
- The object a client uses to specify the target of messages it produces and the source of messages it consumes
- Created using Sun Java System Application Server Admin Console
- With 2 destination subtypes: Queue and Topic
  - Queues: destinations in point-to-point (PTP) messaging domain
  - Topics: shared, subscriber-based topic in pub/sub messaging domain
- Destinations can be looked up by JNDI:

  Topic myTopic = (Topic)ctx.lookup(“jms/*topic_name*”);

  Queue myQueue = (Queue)ctx.lookup(“jms/*queue_name*”)

JMS Sessions

- A session is a *single-threaded* context for producing and consuming messages
- Client program uses session to create message producers, message consumers and messages
- A session provides a *transactional context*: group a set of sends and receives into an atomic unit of work
  - If any one of the operations fails, the transaction can be rolled back, i.e. all produced messages are destroyed and all consumed messages are recovered and redelivered
  - If all the operations succeed, the transaction can be committed, i.e. all messages are sent and all consumed messages are acknowledged
- 2 forms: QueueSession and TopicSession
- E.g.

```
QueueSession queueSession = queueConnection.createQueueSession(true,0);
TopicSession topicSession = topicConnection.createTopicSession(true,0);
```

Session is transacted
Message acknowledgement not needed

JMS Message Producers/Consumers

- Message producer: an object created by a session that is used for sending messages to a destination:
  - PTP: message producer implements the QueueSender interface
  - Pub/sub: message producer implements the TopicPublisher interface
- E.g.

```
QueueSender qs = queueSession.createSender(myQueue);
qs.send(message);
```

- Message consumer: an object created by session that is used for receiving messages sent to a destination:
  - PTP: message producer implements the QueueReceiver interface
  - Pub/sub: message producer implements the TopicSubscriber interface
- E.g.

```
QueueReceiver qr = queueSession.createReceiver(myQueue);
queueConnection.start();
Message m = qr.receive();
```

Sender and Receiver on the same Machine: Running the Example

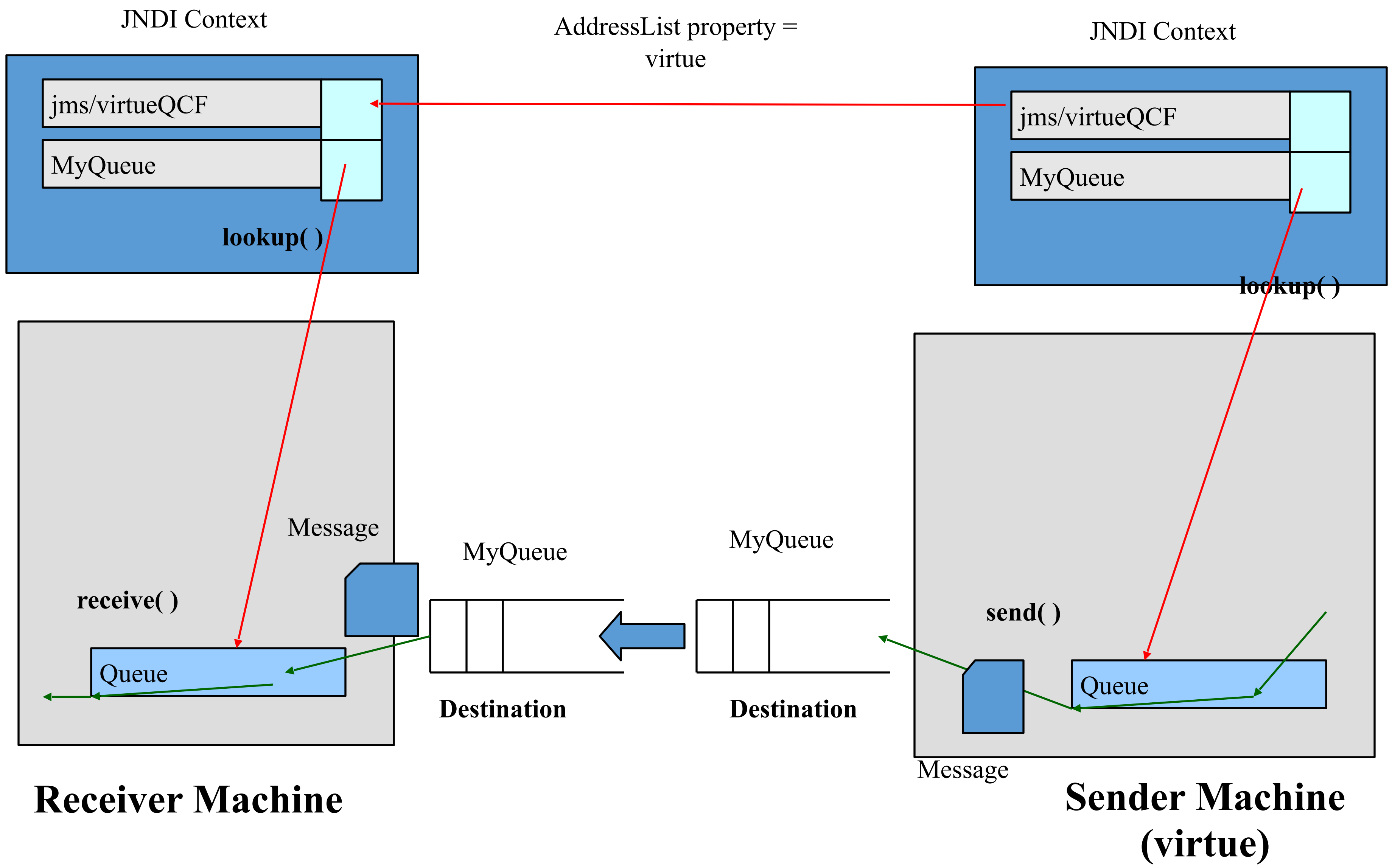
- Compile the source files:

  ```
  javac SimpleQueueSender.java
  javac SimpleQueueReceiver.java
  ```

- Start JMS provider, i.e. the Sun Java System Application Server
- Start the Sun Java System Application Server Admin Console
- Create JMS administered objects:
  - QueueConnectionFactory: jms/QueueConnectionFactory
  - Queue: jms/MyQueue
- Run the PTP clients: SimpleQueueSender
- Run the PTP clients: SimpleQueueReceiver

- Guideline for install GlassFish on unbutu: https://www.howtoforge.com/how-to-install-glassfish-on-ubuntu-22-04/

Sender and Receiver on 2 Machines


JNDI Context
AddressList property = virtue
JNDI Context
jms/virtueQCF
MyQueue
lookup( )
jms/virtueQCF
MyQueue
lookup( )
Message
MyQueue
MyQueue
receive( )
send( )
Queue
Queue
Destination
Destination
Message
Receiver Machine
Sender Machine (virtue)