AOT LAB

Agent and **O**bject **T**echnology **Lab**
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma

Distributed and Agent Systems

JMS

Prof. Agostino Poggi

What is JMS?

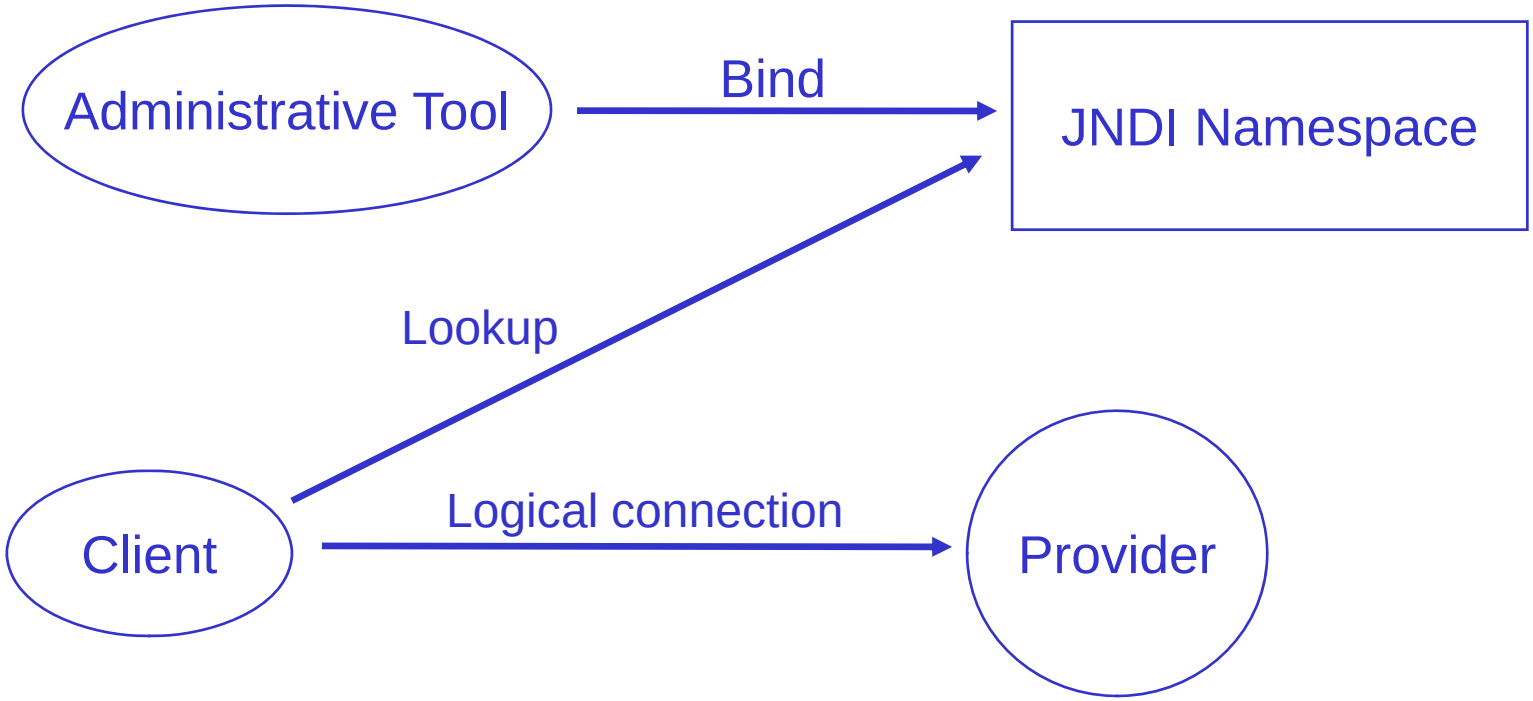
- Acronym of Java Messaging System
- A standardized and system independent Java API for development of heterogeneous, distributed applications based on message exchange
- Minimizes the set of concepts a programmer must learn to use messaging, but provides enough features to support sophisticated applications
- Maximizes the portability of applications across JMS providers in the same messaging domain

JMS Components

- Provider
  - A messaging system that implements JMS interfaces, handles routing and delivery of messages and provides administrative and control features
- Clients
  - Programs or components written in Java that produce or consume messages
- Messages
  - Objects containing data exchanged between clients
- Administered objects
  - Preconfigured JMS objects created for the management of clients and their interactions

AOT LAB

JMS Components Interaction

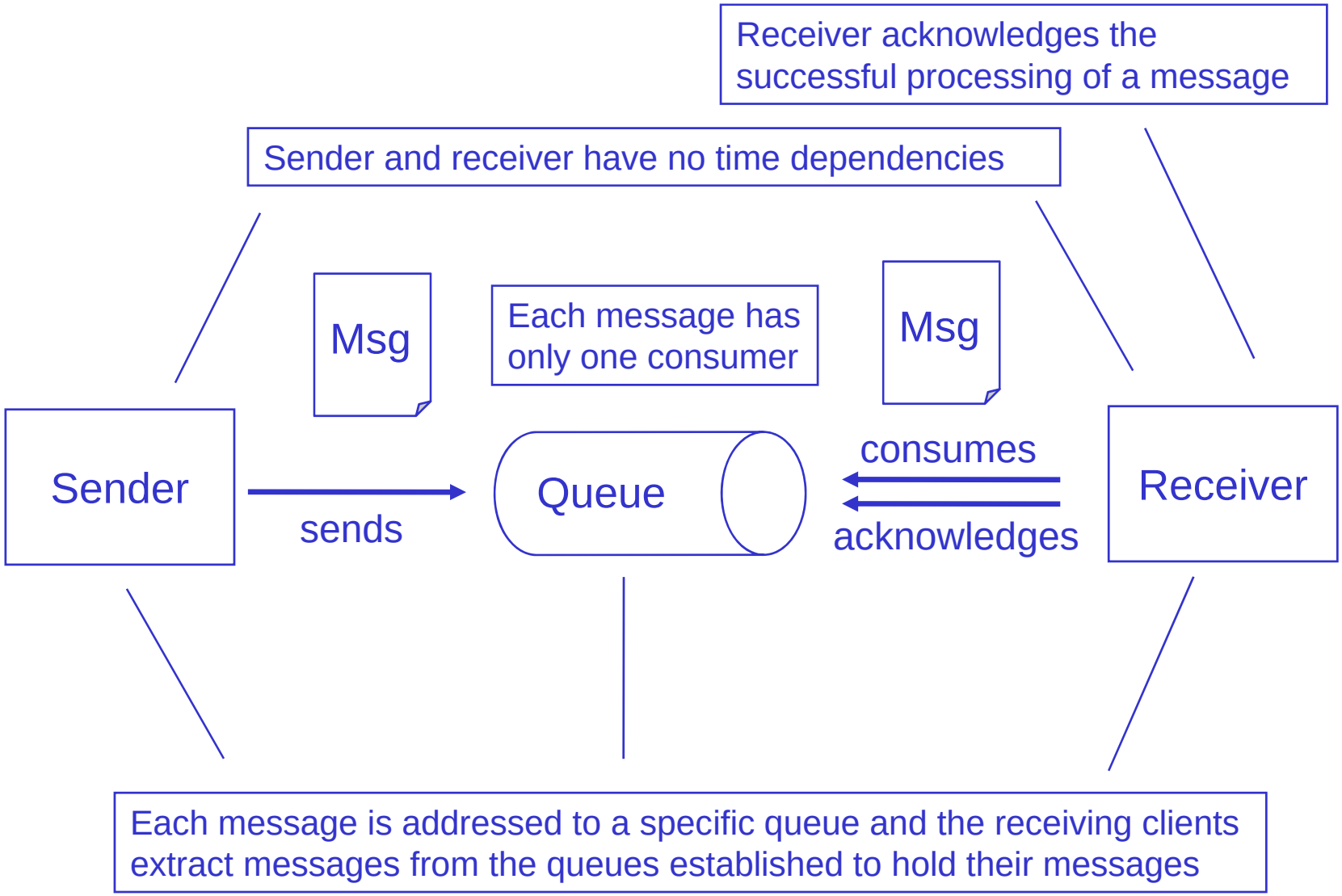


Client – Client Communication

- Communication between clients is not only loosely coupled but also:
  - Asynchronous
    - JMS provider can deliver messages to a client as they arrive and the client does not have to request messages in order to receive them
  - Reliable
    - JMS API can ensure that a message is delivered once and only once. Lower levels of reliability are available for applications that can afford to miss messages or to receive duplicate messages

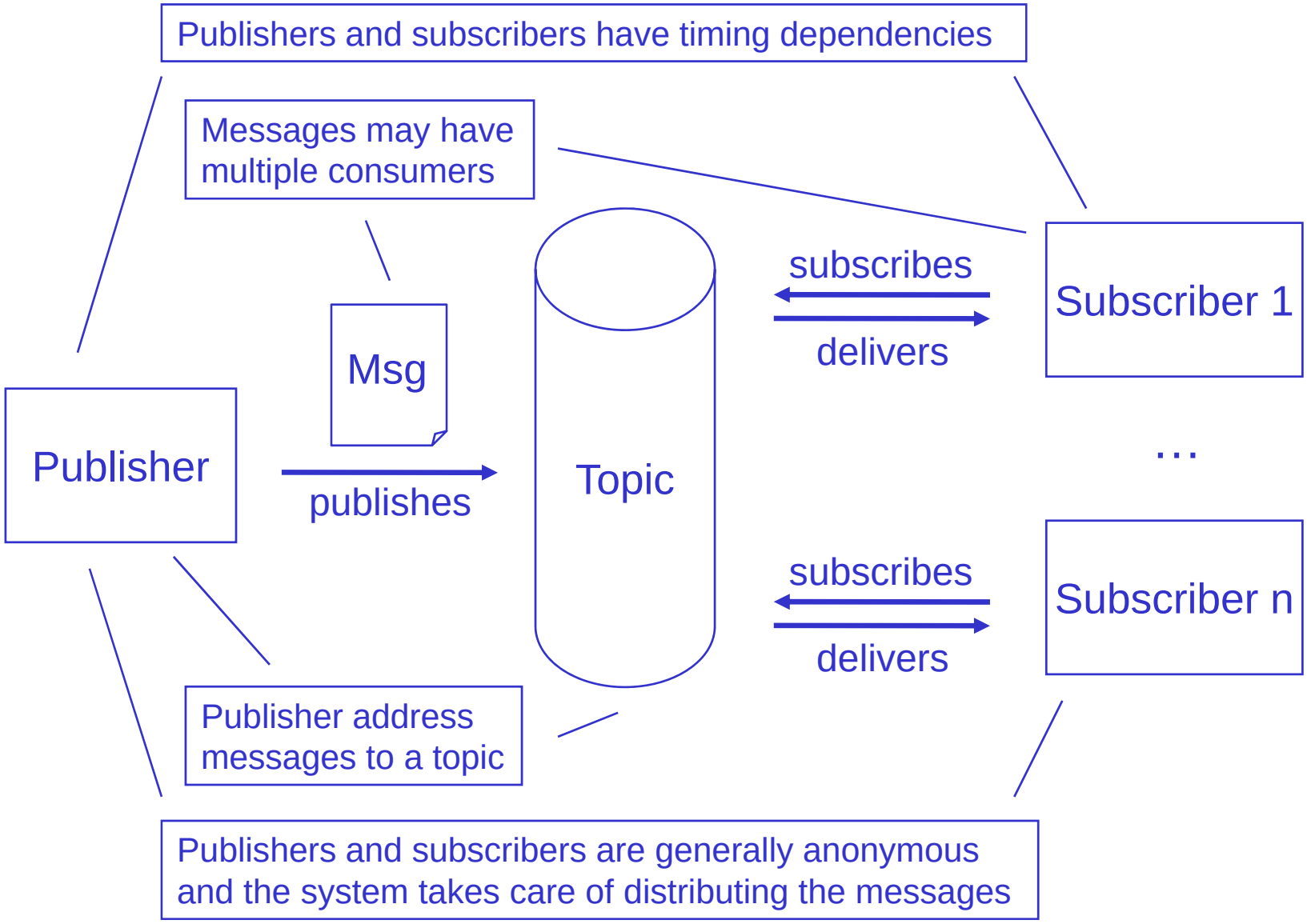
AOT LAB

Point-to-Point Messaging



AOT LAB

Publish/Subscribe Messaging



Message Consumptions

- Synchronously
  - A subscriber or a receiver explicitly fetches the message from the destination by calling the receive method
  - The receive method can block until a message arrives or can time out if a message does not arrive within a specified time limit
- Asynchronously
  - A client can register a message listener with a consumer
  - Whenever a message arrives at the destination, the JMS provider delivers the message by calling the listener's onMessage() method

Message Components

- A header containing a set of fields describing the message and its scope
- An optional body whose contents depends on the message type
- Some optional properties defined by the client

AOT LAB

Message Header

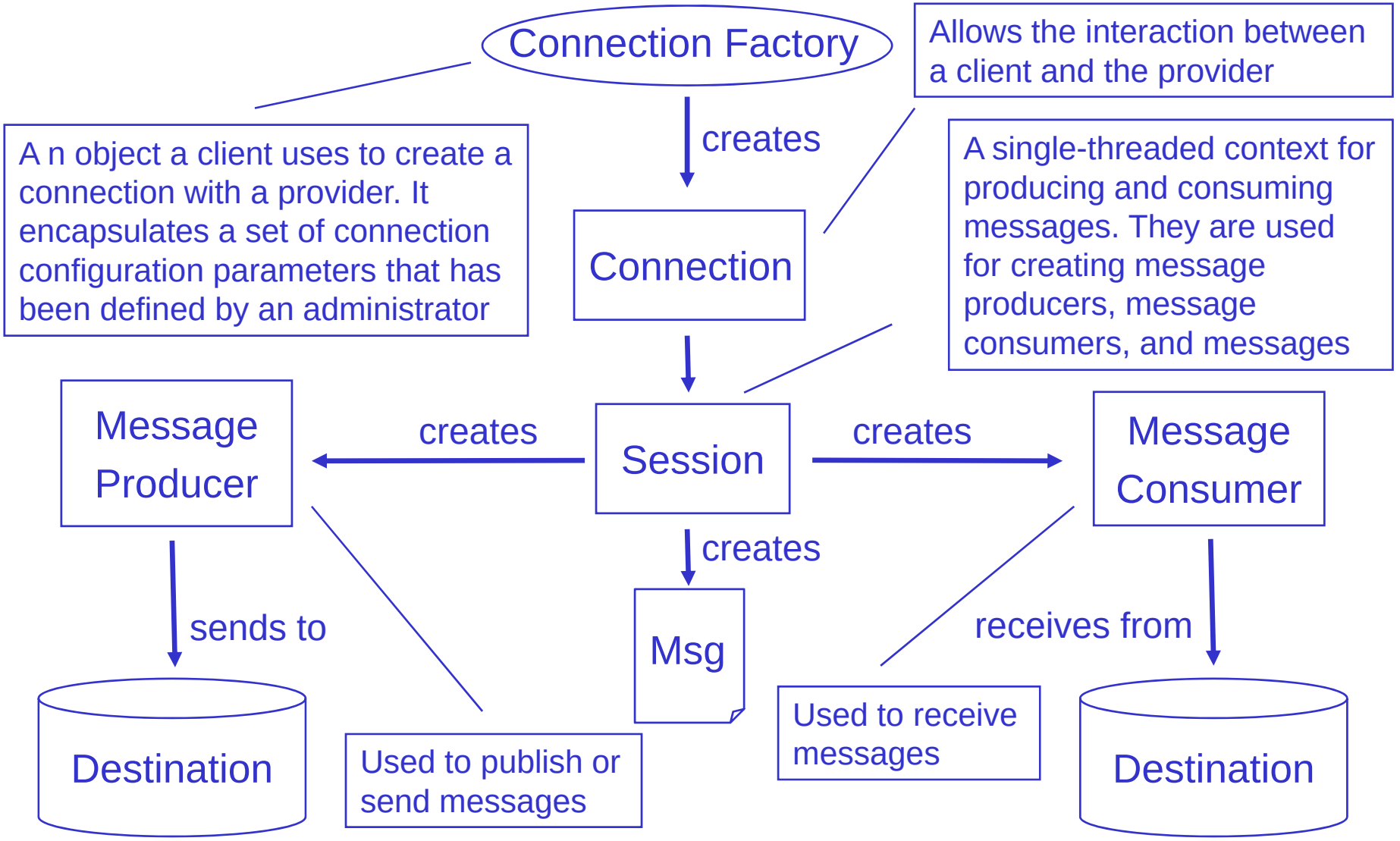
Field	Set By	Meaning
JMSDestination	Send / publish method	Queue or Topic
JSMDeliveryMode	Send / publish method	(non-) persistent
JMSExpiration	Send / publish method	Expiration time
JMSPriority	Send / publish method	From 0 (low) to 9 (high)
JMSMessageID	Send / publish method	Unique identifier
JMSTimestamp	Send / publish method	Hand off time
JMSCorrelationID	Client	Messages correlation
JMSReplyTo	Client	Destination of the reply
JMSType	Client	Contents type
JMSRedelivered	Provider	Retransmitted message

Message Types

	Contains	**Methods**
TextMessage	String	getText, setText, …
MapMessage	Name and value pairs set	getString, setString, getLong, setLong, …
BytesMessage	Uninterpreted bytes stream	readBytes, writeBytes, …
StreamMessage	Primitive values stream	readString, writeString, readLong,writeLong, …
ObjectMessage	Serialize object	getObject, setObject, …

AOT LAB

JMS Programming Model



What is a Naming Service?

- A naming service associates names with objects (references) and allows the retrieval of the objects (references) through their names
  - A DNS maps machine names to IP addresses
  - A file system maps a filename to a file reference
- A context is a set of name-object bindings and provides a naming convention
- A name in one context object can be bound to another context object (sub-context)
  - The directory /usr defines a context in a Unix file system
  - The directory /usr/etc define a sub-context of /usr

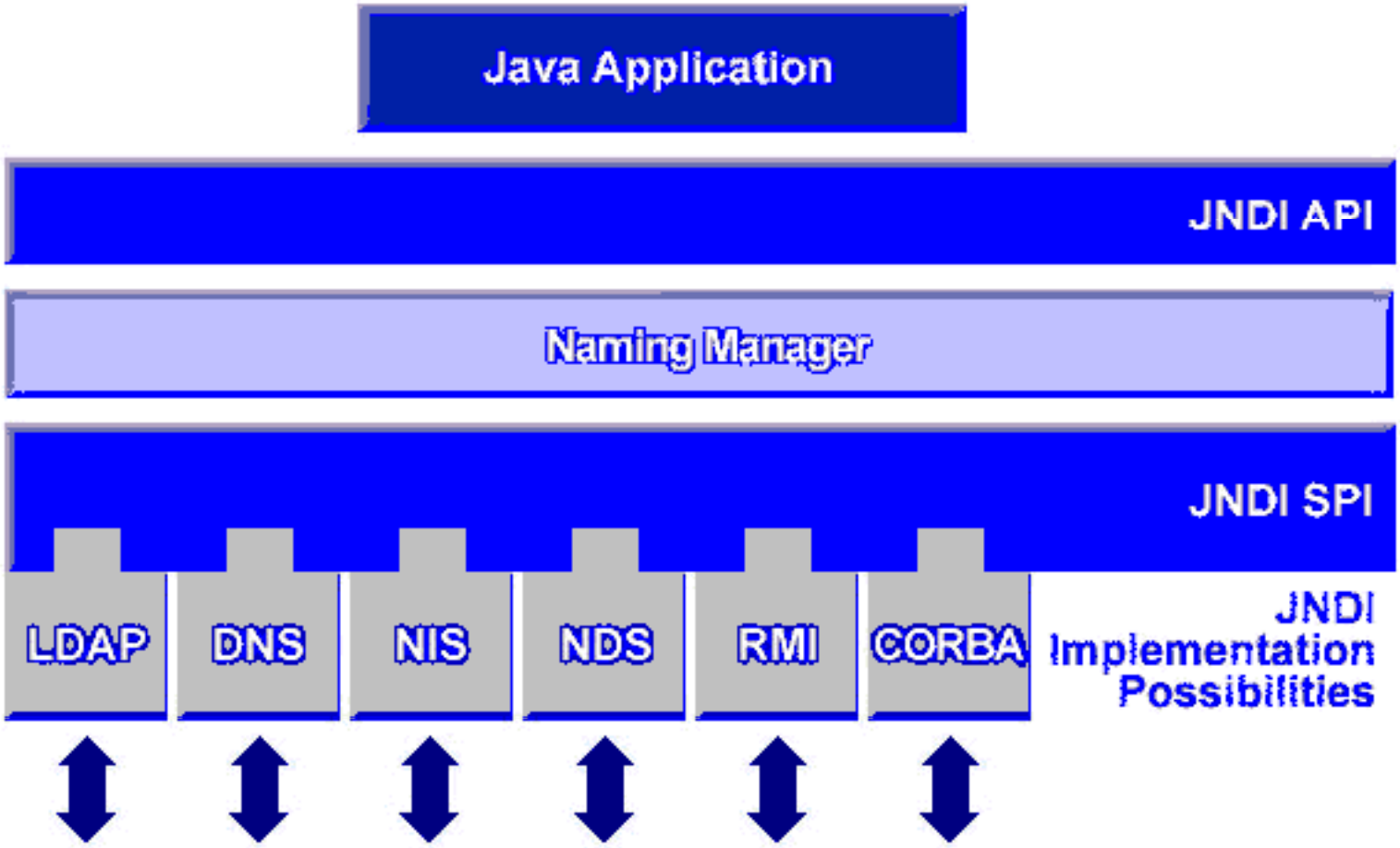
What is a Directory Service?

- A directory service extends a name service by associating names with objects and such objects with some attributes
  - a printer might be represented by an object that has as attributes its speed, resolution, and color
- A directory is a connected set of directory objects
- A directory service is a service for managing information about objects and searching them through such information
- A directory service can be combined with a naming service for arranging all directory objects in a tree

What is JNDI?

- The acronym of Java Naming and Directory Interface (JNDI)
- Provides naming and directory functionality to Java applications and is defined to be independent of any specific directory service implementation
- Consists of an API and a service provider interface
  - Java applications use the JNDI API to access a variety of naming and directory services
  - The service provider interface enables a variety of naming and directory services to be plugged in transparently, thereby allowing the Java application using the JNDI API to access their services

JNDI Architecture



JNDI Packages

- **javax.naming** provides the means for accessing naming services
- **javax.naming.directory** is an extension of javax.naming to provide access to directories
- **javax.naming.events** supports event notification in naming and directory services
- **javax.naming.ldap** support the use of features that are specific to LDAP
- **javax.naming.spi** provides the means for realizing the “plug-in” that allow the use of other naming and directory service providers through JNDI

A Point-to-Point Application

1. Write a sender and a receiver class
2. Compile the two classes
3. Start a JMS provider
4. Create the JMS administered objects (a queue)
5. Run the sender and the receiver (in any order)
6. Delete the queue
7. Stop the JMS provider

Sender Code

```
import javax.jms.*;
import javax.naming.*;
public class SimpleQueueSender {
  public static void main(String[] args) {
    String queueName = null;
    Context jndiContext = null;
    QueueConnectionFactory queueConnectionFactory = null;
    QueueConnection queueConnection = null;
    QueueSession queueSession = null;
    Queue queue = null;
    QueueSender queueSender = null;
    TextMessage message = null;
    final int NUM_MSGS;
    …
  }
}
```

Sender Main Code 1

```
if ( (args.length < 1) || (args.length > 2) ) { System.exit(1); }
queueName = new String(args[0]);
if (args.length == 2)
  NUM_MSGS = (new Integer(args[1])).intValue();
else
  NUM_MSGS = 1;
try {
  jndiContext = new InitialContext();
} catch (NamingException e) { System.exit(1); }
try {
  queueConnectionFactory = (QueueConnectionFactory)
  jndiContext.lookup("QueueConnectionFactory");
  queue = (Queue) jndiContext.lookup(queueName);
}
catch (NamingException e) { System.exit(1); }
```

Sender Main Code 2

```
try {
  queueConnection = queueConnectionFactory.createQueueConnection();
  queueSession = queueConnection.createQueueSession(
      false, Session.AUTO_ACKNOWLEDGE);
  queueSender = queueSession.createSender(queue);
  message = queueSession.createTextMessage();
  for (int i = 0; i < NUM_MSGS; i++) {
    message.setText("This is message " + (i + 1));
    queueSender.send(message);
  }
  queueSender.send(queueSession.createMessage());
} catch (JMSException e) {
} finally {
  if (queueConnection != null) {
    try { queueConnection.close(); } catch (JMSException e) {} }
}
```

Receiver Code

```
import javax.jms.*;
import javax.naming.*;
public class SimpleQueueReceiver {
  public static void main(String[] args) {
    String queueName = null;
    Context jndiContext = null;
    QueueConnectionFactory queueConnectionFactory = null;
    QueueConnection queueConnection = null;
    QueueSession queueSession = null;
    Queue queue = null;
    …
  }
}
```

Receiver Main Code 1

```
if (args.length != 1) { System.exit(1); }
queueName = new String(args[0]);
try {
  jndiContext = new InitialContext();
} catch (NamingException e) { System.exit(1); }
try {
  queueConnectionFactory = (QueueConnectionFactory)
  jndiContext.lookup("QueueConnectionFactory");
  queue = (Queue) jndiContext.lookup(queueName);
}
catch (NamingException e) { System.exit(1); }
```

Receiver Main Code 2

```
try {
  queueConnection = queueConnectionFactory.createQueueConnection();
  queueSession = queueConnection.createQueueSession(
      false, Session.AUTO_ACKNOWLEDGE);
  queueReceiver = queueSession.createReceiver(queue);
  queueConnection.start();
  while (true) {
    Message m = queueReceiver.receive(1);
    if (m != null) {
      if (m instanceof TextMessage) {
        System.out.println("Message: " + ((TextMessage) m).getText());
      } else { break; } } }
} catch (JMSException e) {
} finally {
  if (queueConnection != null) {
    try { queueConnection.close(); } catch (JMSException e) {} }
}
```

A Publisher/Subscriber Application

1. Write a publisher, a subscriber and a message listener class
2. Compile the three classes
3. Start a JMS provider
4. Create the JMS administered objects (a topic)
5. Run the subscriber and the publisher
6. Delete the queue
7. Stop the JMS provider

Publisher Code

```
import javax.jms.*;
import javax.naming.*;
public class SimpleTopicPublisher {
  public static void main(String[] args) {
    String topicName = null;
    Context jndiContext = null;
    TopicConnectionFactory topicConnectionFactory = null;
    TopicConnection topicConnection = null;
    TopicSession topicSession = null;
    Topic topic = null;
    TopicPublisher topicPublisher = null;
    TextMessage message = null;
    final int NUM_MSGS;
    …
  }
}
```

Publisher Main Code 1

```
if ( (args.length < 1) || (args.length > 2) ) { System.exit(1); }
topicName = new String(args[0]);
if (args.length == 2)
  NUM_MSGS = (new Integer(args[1])).intValue();
else
  NUM_MSGS = 1;
try {
  jndiContext = new InitialContext();
}
catch (NamingException e) { System.exit(1); }
try {
  topicConnectionFactory = (TopicConnectionFactory)
  jndiContext.lookup("TopicConnectionFactory");
  topic = (Topic) jndiContext.lookup(topicName);
}
catch (NamingException e) { System.exit(1); }
```

Publisher Main Code 2

```
try {
  topicConnection = topicConnectionFactory.createTopicConnection();
  topicSession = topicConnection.createTopicSession(
      false, Session.AUTO_ACKNOWLEDGE);
  topicPublisher = topicSession.createPublisher(topic);
  message = topicSession.createTextMessage();
  for (int i = 0; i < NUM_MSGS; i++) {
    message.setText("This is message " + (i + 1));
    topicPublisher.publish(message);
  }
} catch (JMSException e) {
} finally {
  if (topicConnection != null) {
    try { topicConnection.close(); } catch (JMSException e) {} }
}
```

Subscriber Code

```
import javax.jms.*;
import javax.naming.*;
public class SimpleTopicSubscriber {
  public static void main(String[] args) {
     String topicName = null;
     Context  jndiContext = null;
     TopicConnectionFactory  topicConnectionFactory = null;
     TopicConnection topicConnection = null;
     TopicSession topicSession = null;
     Topic topic = null;
     TopicSubscriber topicSubscriber = null;
     TextListener  topicListener = null;
     TextMessage message = null;
     InputStreamReader  inputStreamReader = null;
     char answer = '\0';
     … } }
```

Subscriber Main Code 1

```
if (args.length != 1) { System.exit(1); }
topicName = new String(args[0]);
try {
  jndiContext = new InitialContext();
}
catch (NamingException e) { System.exit(1); }
try {
  topicConnectionFactory = (TopicConnectionFactory);
  jndiContext.lookup("TopicConnectionFactory");
  topic = (Topic) jndiContext.lookup(topicName);
}
catch (NamingException e) { System.exit(1); }
```

Subscriber Main Code 2

```
try {
  topicConnection = topicConnectionFactory.createTopicConnection();
  topicSession = topicConnection.createTopicSession(
      false, Session.AUTO_ACKNOWLEDGE);
  topicSubscriber = topicSession.createSubscriber(topic);
  topicListener = new TextListener();
  topicSubscriber.setMessageListener(topicListener);
  topicConnection.start();
  System.out.println("To end program, enter Q or q, " + "then <return>");
  inputStreamReader = new InputStreamReader(System.in);
  while (!((answer == 'q') || (answer == 'Q'))) {
    try {  answer = (char) inputStreamReader.read(); }
    catch (IOException e) { }  }
} catch (JMSException e) {
} finally {
  if (topicConnection != null) {
    try { topicConnection.close(); } }
}
```

Message Listener Code

```
import javax.jms.*;
public class TextListener implements MessageListener {
  public void onMessage(Message message) {
    TextMessage msg = null;
     try {
       if (message instanceof TextMessage) {
         msg = (TextMessage) message;
         System.out.println("Message: " + msg.getText());
       } else {
         System.out.println("Wrong type: " + message.getClass().getName());
       }
    } catch (JMSException e) {
    } catch (Throwable t) { }
  }
}
```

JMS Providers

- activeMQ from: http://activemq.apache.org
- openJMS from: http://openjms.sourceforge.net
- JEE from: http://java.sun.com
- Different other open sources and commercial implementations (BEA, IBM, Jboss, ORACLE, …)