Message Oriented Middleware

Daniel Hagimont

IRIT/ENSEEIHT
2 rue Charles Camichel - BP 7122
31071 TOULOUSE CEDEX 7

Daniel.Hagimont@enseeiht.fr
http://hagimont.perso.enseeiht.fr

Message based model Introduction

- Client-server model
  - Synchronous calls
  - Appropriate for tightly coupled components
  - Explicit designation of the destination
  - Connection 1-1
- Message model
  - Asynchronous communication
  - Anonymous designation (e.g.: announcement on a newsgroup)
  - Connection 1-N

Message based model Introduction

- Application example
  - Supervision of equipments in a network
  - E.g. average load on a set of servers
- Client-server solution
  - Periodic invocation
- Message based solution
  - Each equipment notifies state changes
  - Administrators subscribe notifications

Message based services ... used everyday

- Electronic forums (News)
  - Pull technologies
  - consummers can subscribe to a forum
  - producers can publish information in a forum
  - consummers can login and read the content anytime
- Electronic mail
  - Push technologies
  - mailing lists (multicast - publish/subscribe)
  - consummers can subscribe a mailing list
  - producers can send emails to a mailing list
  - Consummers receive emails without having to perform any specific action

- Asynchronous
- Anonymous
- 1-N

Message based middleware Principles

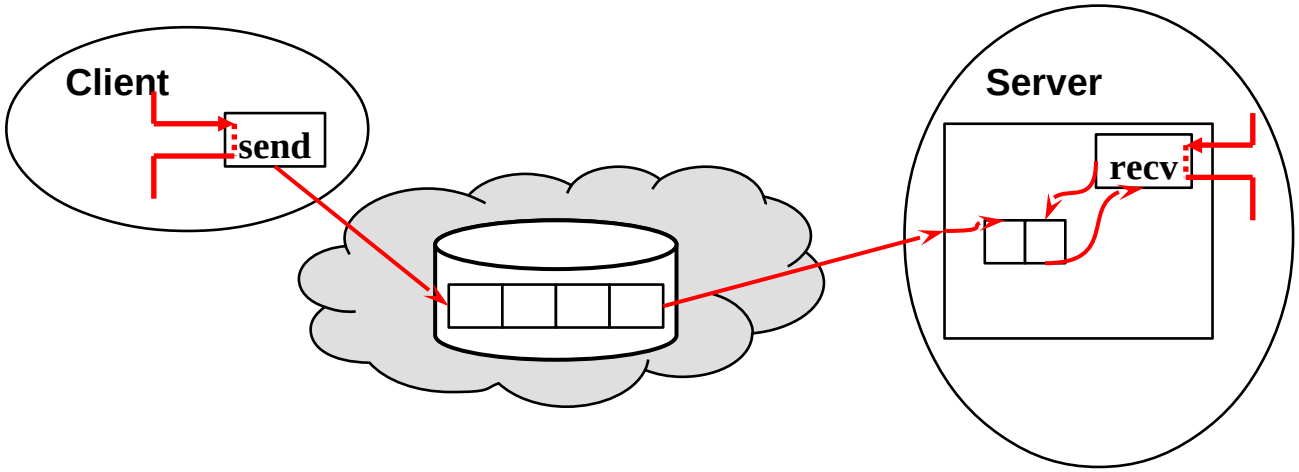
- Message Passing (communication with messages)
- Message Queuing (communication with persistent message queues)
- Publish/Subscribe (communication with subscriptions)
- Events (communication with callbacks)

Message based middleware
Message passing

- Communication with message
  - In a classical architecture: sockets
  - In a parallel programming environnement: PVM, MPI
  - Other environnements: ports (e.g. Mach)

Message based middleware
Message Queuing

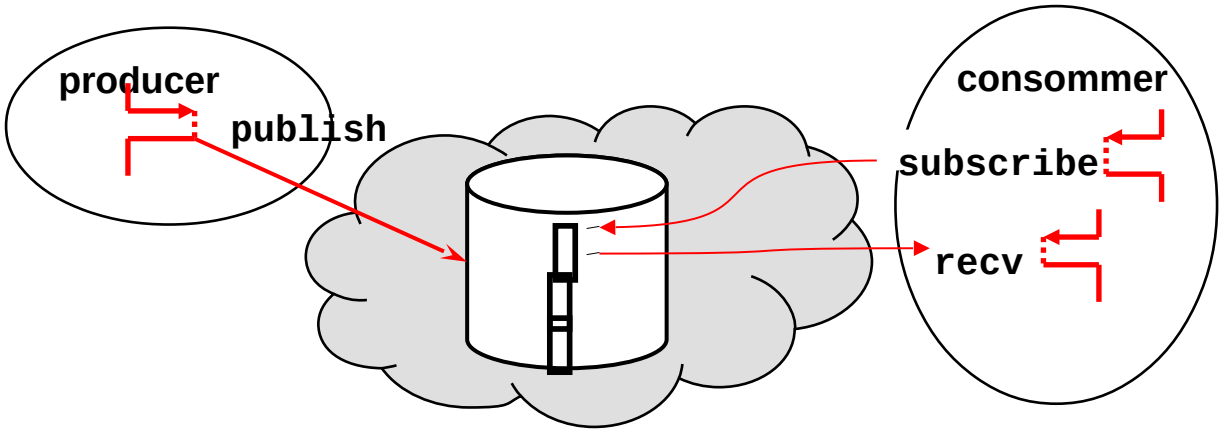
- Queue of messages
  - persistent ⇒ asynchronism and reliability



- Independence between the emitter and the receiver
  - The receiver is not necessarily active

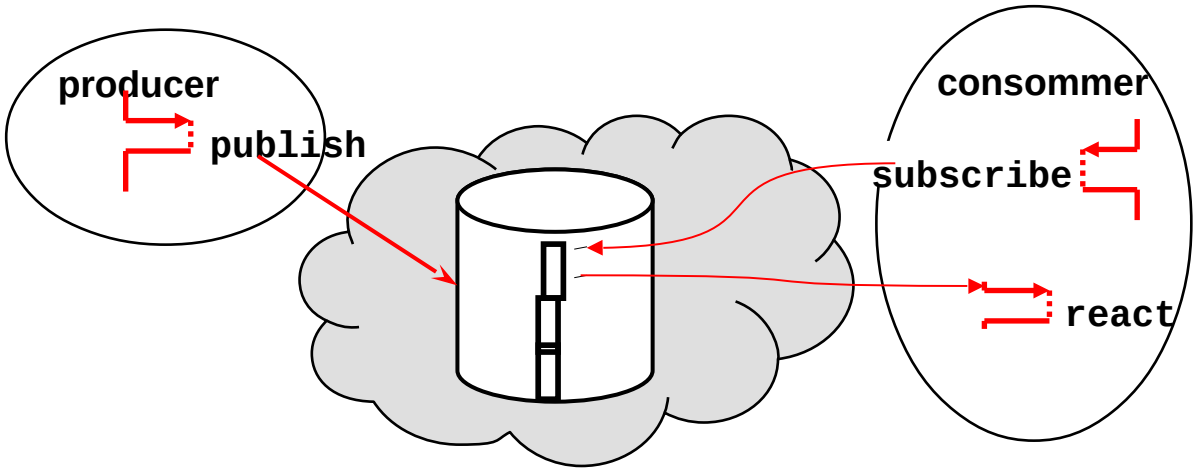
Message based middleware Publish/Subscribe

- Anonymous designation
  - The emitter sends a message
    - Subject-based
    - Content-based
  - The receiver subscribes (to a subject or a content)
- Communication 1-N
  - Several receivers may subscribe



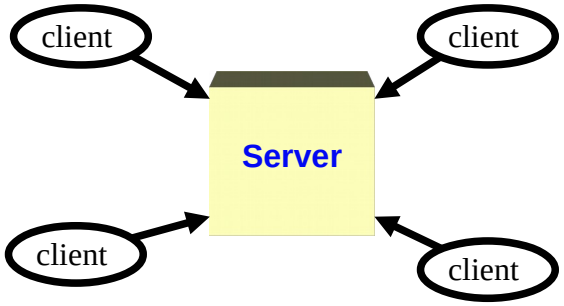
Message based middleware Events

- Basic concepts: events, reactions (handling associated with event reception)
- Attachement: association between an event type and a reaction

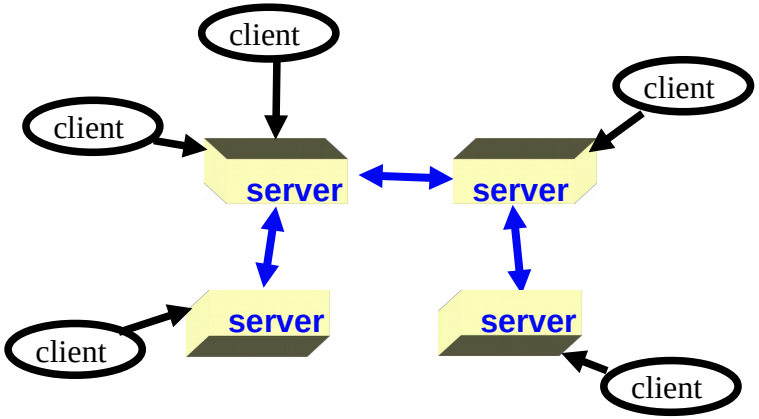


- Exists for all forms of messaging (Message Passing, Message Queuing, Publish/Subscribe)

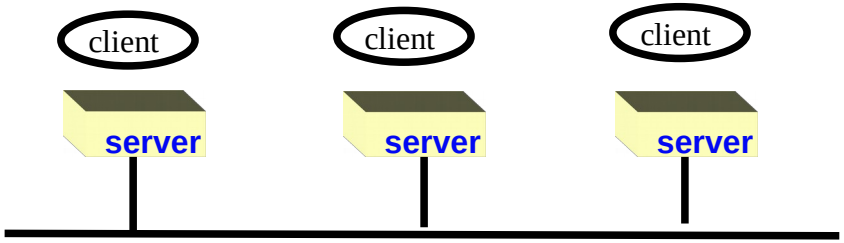
Message based middleware Implementations



Centralized server
(Hub & spoke)



Distributed servers
(Snow flake)



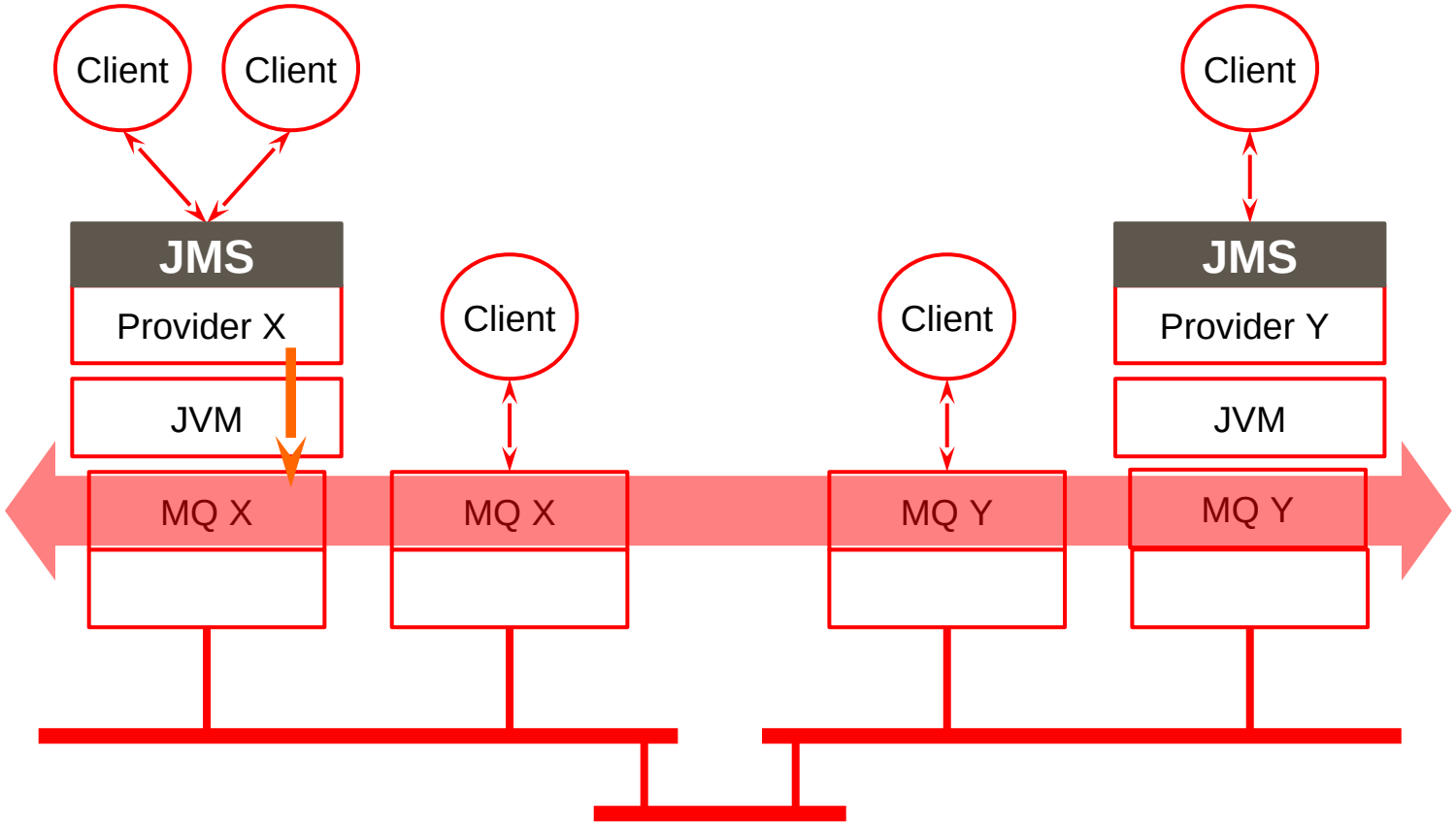
Software bus

Java Message Service

- JMS: Java API defining a uniform interface for accessing messaging systems
  - IBM (WebSphere MQ), Oracle (WebLogic)
  - Apache ActiveMQ, RabbitMQ

  - Message Queue
  - Publish/Subscribe
  - Event

JMS: an interface (portability, not Interoperability)



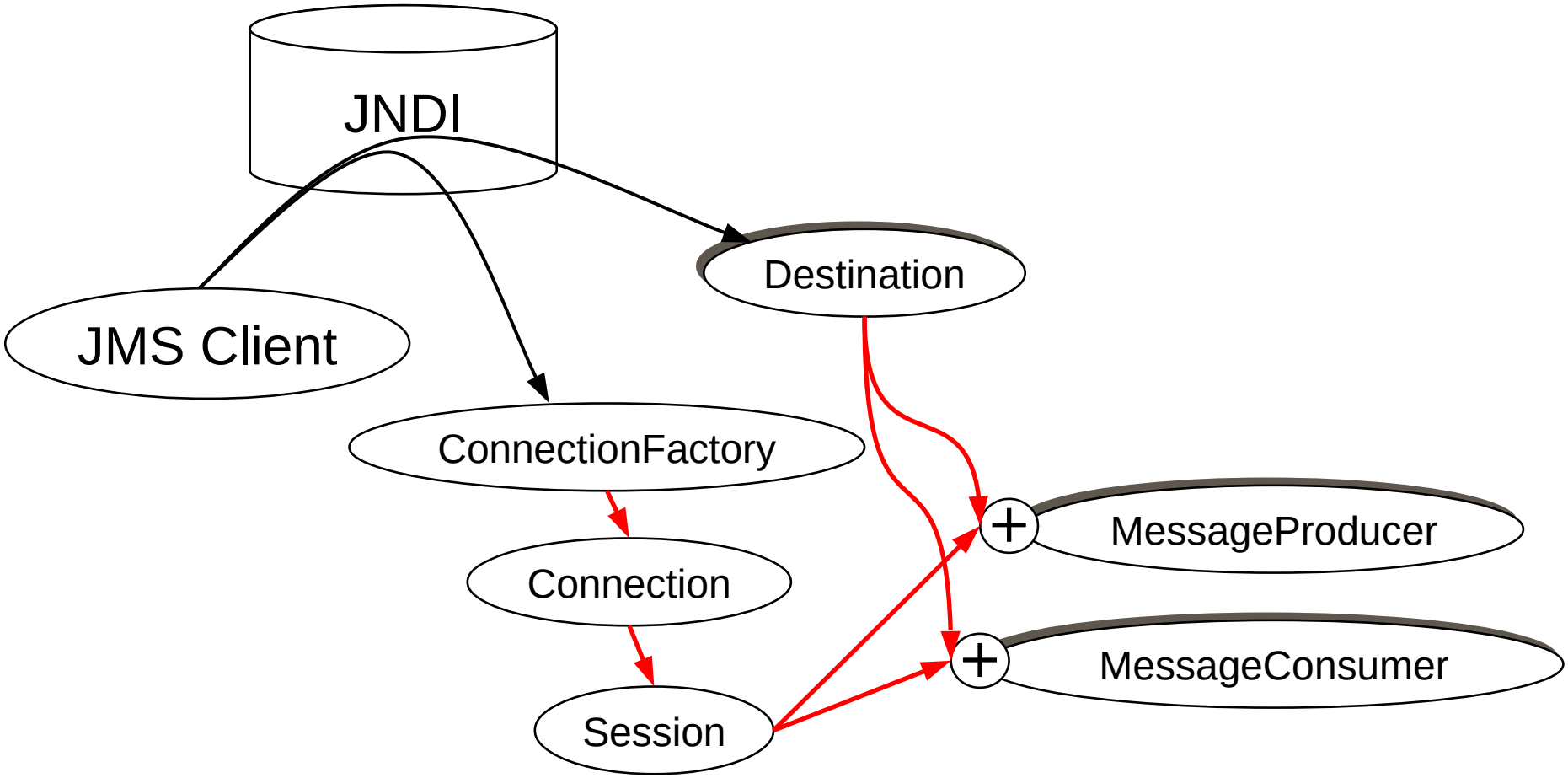
Interoperability : AMQP (Advanced Message Queuing Protocol)

JMS interface

- *ConnectionFactory*: factory to create a connection with a JMS server
- *Connection*: an active connection with a JMS server
- *Destination*: a location (source or destination)
- *Session*: a single-thread context for emitting or receiving
- *MessageProducer*: an object for emitting in a session
- *MessageConsummer*: an object for receiving in a session

- Implementations of these interface are specific to providers ...

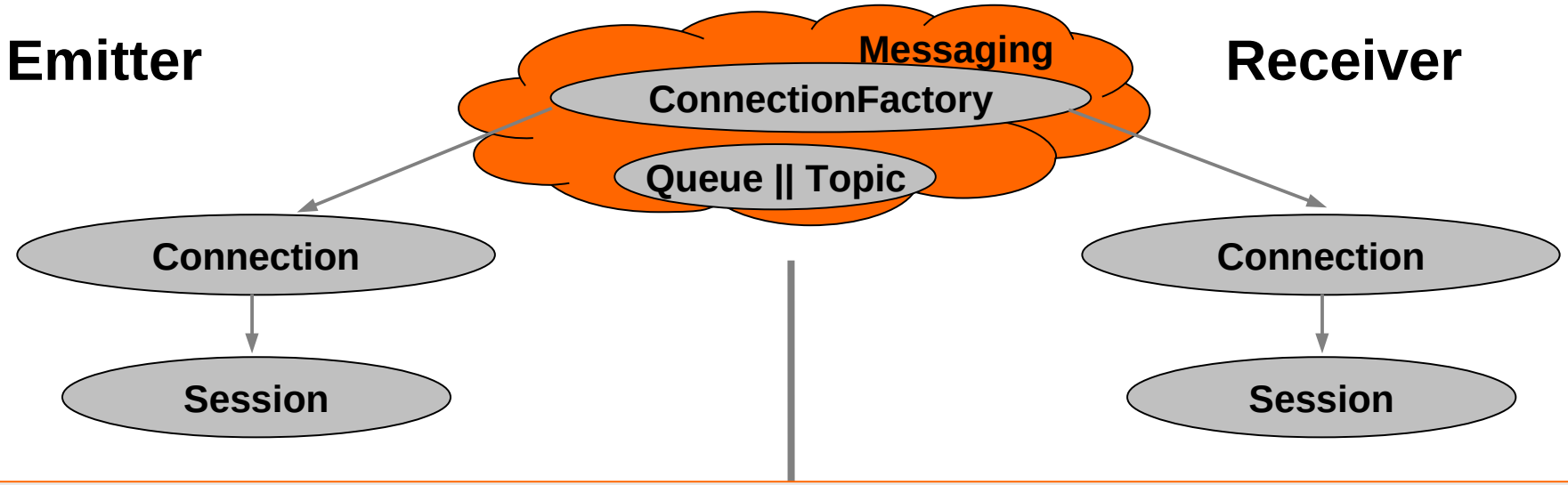
JMS - Architecture



Interfaces PTP et P/S

	Point-To-Point	Publish/Subscribe
ConnectionFactory	*QueueConnectionFactory*	*TopicConnectionFactory*
Connection	*QueueConnection*	*TopicConnection*
Destination	*Queue*	*Topic*
Session	*QueueSession*	*TopicSession*
MessageProducer	*QueueSender*	*TopicPublisher*
MessageConsumer	*QueueReceiver*	*TopicSubscriber*

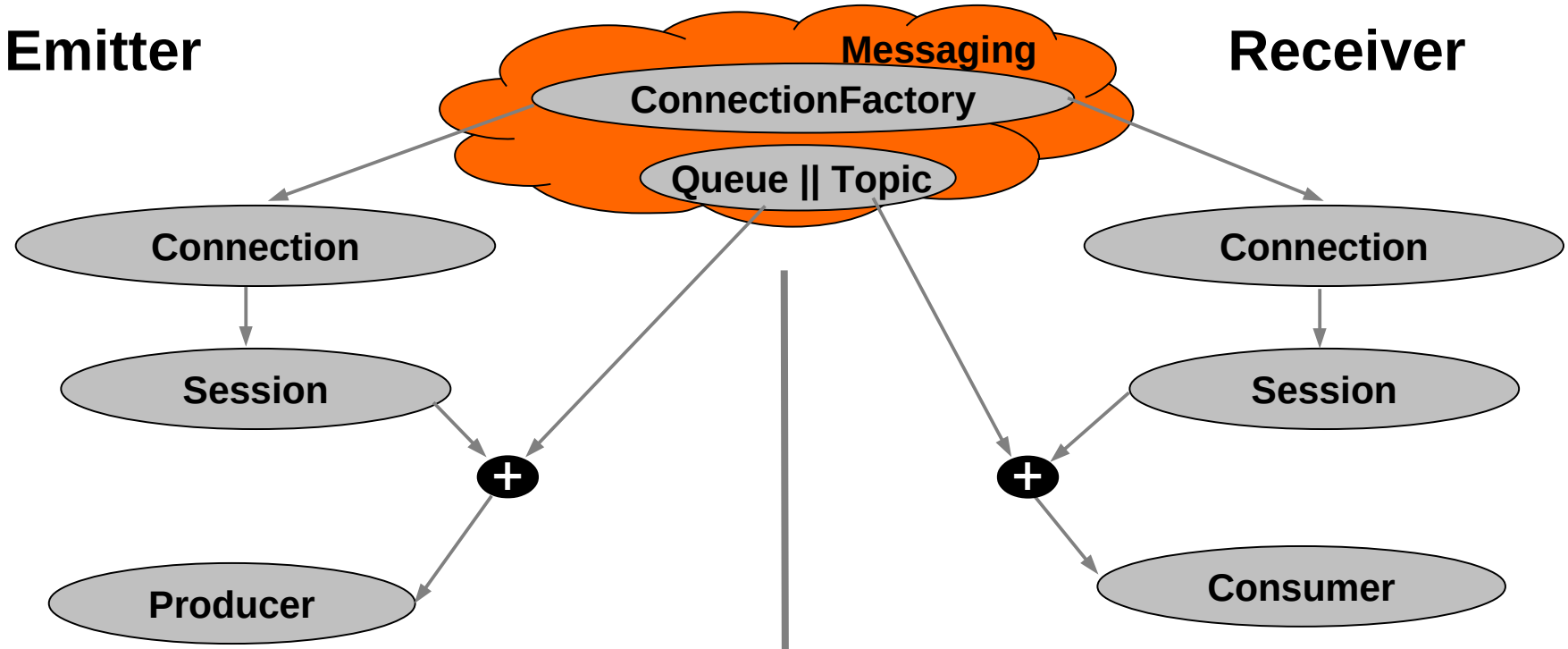
JMS - initialization




```
ConnectionFactory connectionFactory =
        new ActiveMQConnectionFactory(ActiveMQConnection.DEFAULT_BROKER_URL);
Connection connection = connectionFactory.createConnection();
connection.start();
Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);

Destination destination = session.createQueue("myQueue");
Destination destination = session.createTopic("MyTopic");
```

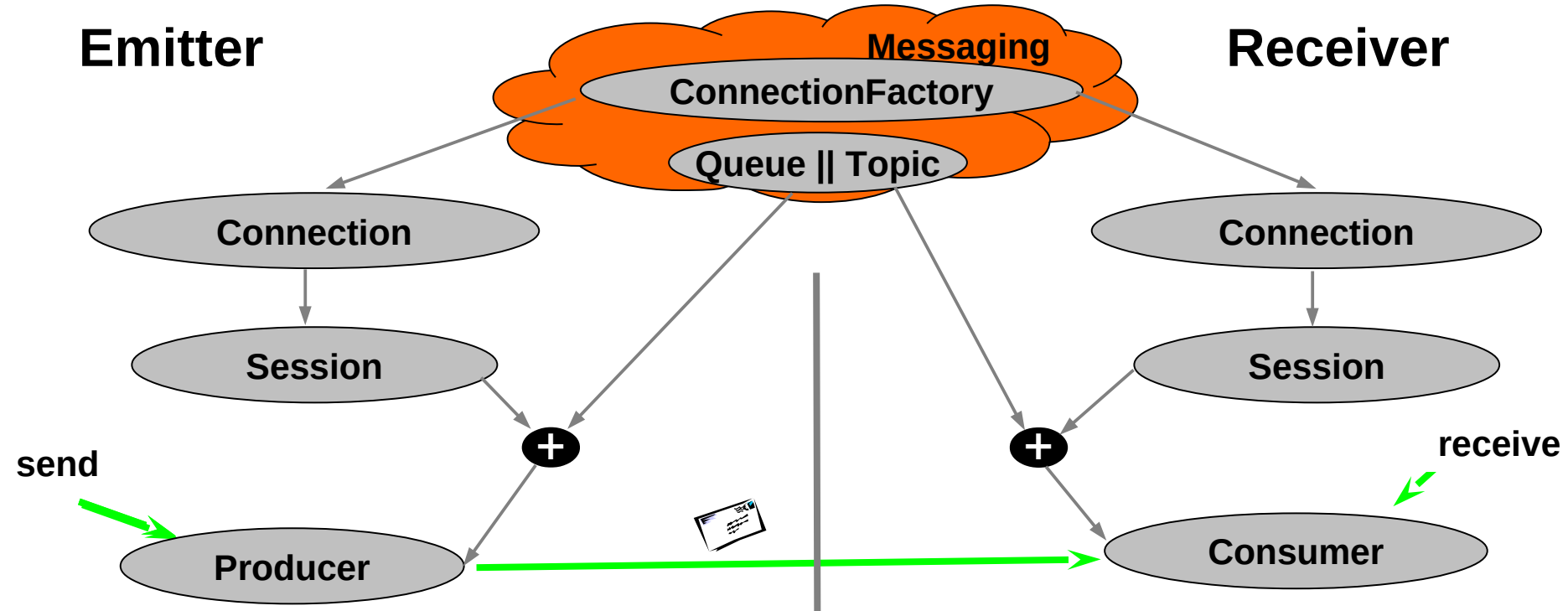
JMS – producer / consumer




```
MessageProducer producer =
      session.createProducer(destination);
```

```
MessageConsumer consumer =
    session.createConsumer(destination);
```

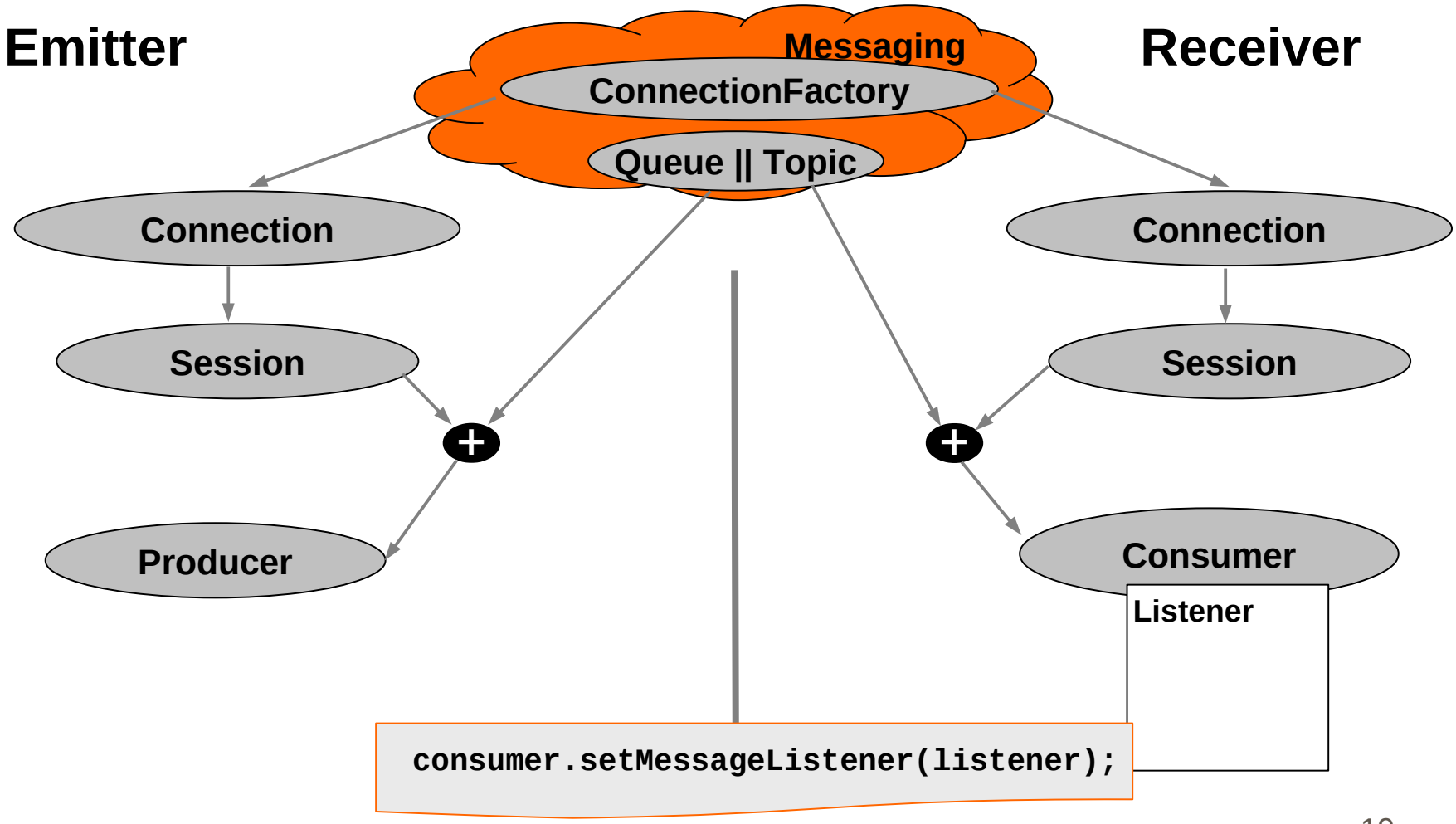
JMS - communication



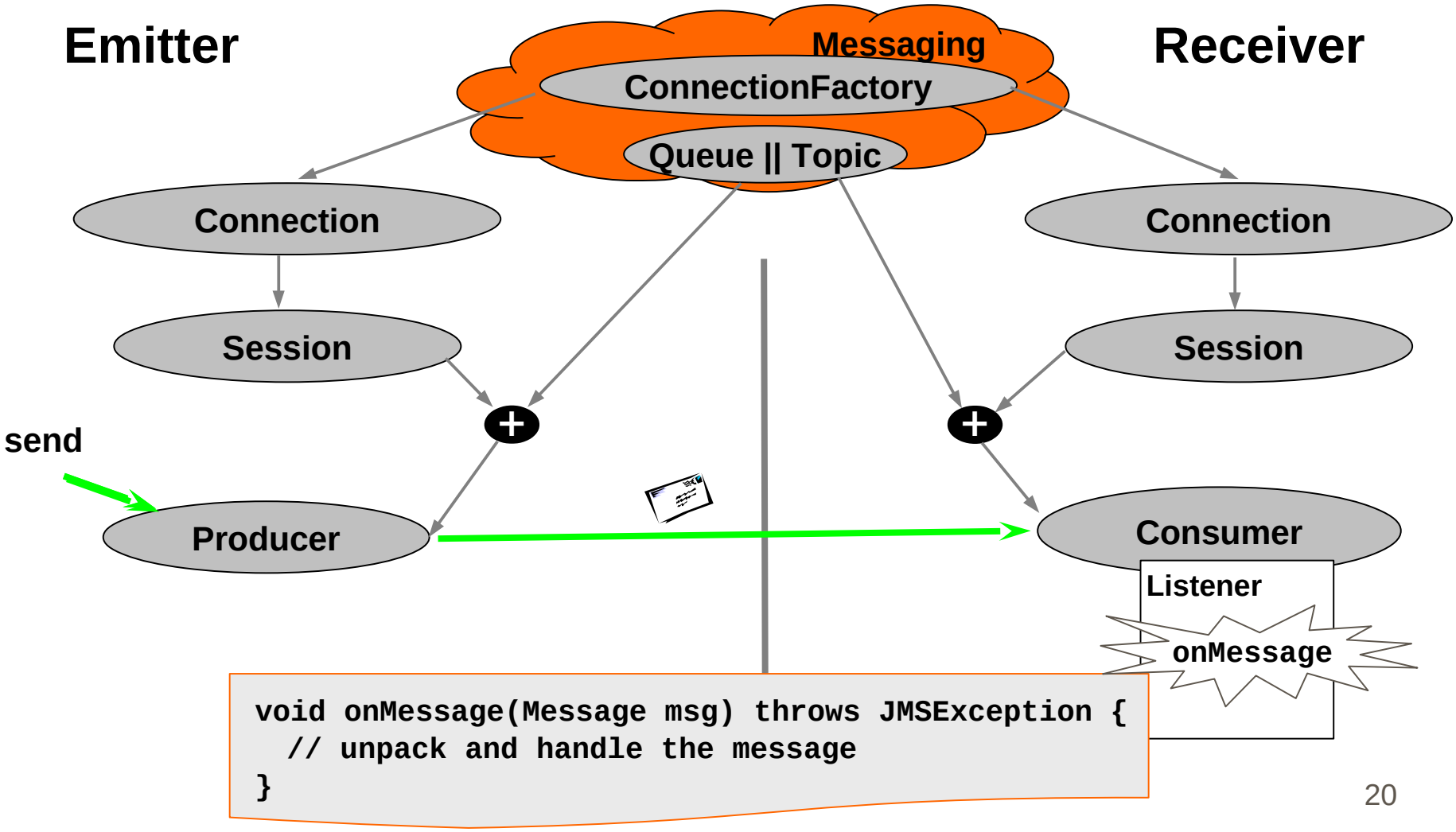

```
TextMessage msg =
    session.createTextMessage("...");
sender.send(msg);
```

```
TextMessage m =
   (TextMessage)consumer.receive();
```

JMS - Listener



JMS - Listener



JMS – messages

- TextMessage (a character string)

```
String data;
TextMessage message = session.createTextMessage();
message.setText(data);
```

```
String data;
data = message.getText();
```

- BytesMessage (bytes array)

```
byte[] data;
BytesMessage message = session.createByteMessage();
message.writeBytes(data);
```

```
byte[] data;
int length;
length = message.readBytes(data);
```

JMS – messages

- MapMessage (sequence of key-value pair)
  - A value is a primitive type

```
MapMessage message = session.createMapMessage();

message.setString("Name", "…");
message.setDouble("Value", doubleValue);
message.setLong("Time", longValue);
```

```
String name = message.getString("Name");
double value = message.getDouble("Value");
long time = message.getLong("Time");
```

JMS – messages

- StreamMessage (sequence of values)
  - A value is a primitive type
  - Reading should respect the sequence order to writing

```
StreamMessage message = session.createStreamMessage();

message.writeString("…");
message.writeDouble(doubleValue);
message.writeLong(longValue);
```

```
String name = message.readString();
double value = message.readDouble();
long time = message.readLong();
```

JMS – messages

- ObjectMessage (serialized objects)

```
ObjectMessage message = session.createObjectMessage();

message.setObject(obj);
```

```
obj = message.getObject();
```

Conclusions

- Communication with messages
  - Simple programming model
  - Many extensions, variants …
    - Message software bus, actors models, multi-agent systems …
  - Widely used for interconnecting tools, existing, developed independently
- However… it is only apparently simple
  - Propagation and report of errors
  - Development tools