



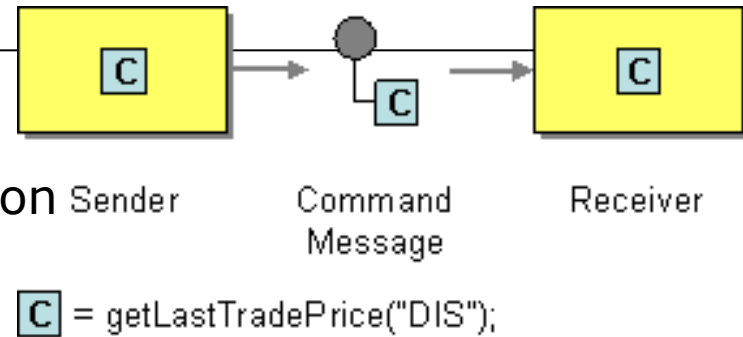
Message Construction

Systems Integration
PBA Softwareudvikling/BSc Software Development
Tine Marbjerg
Fall 2017

Message Issues

- **Message intent**
 - *Command Message* (145) invoke function
 - *Document Message* (147) send data
 - *Event Message* (151) send notification
- **Returning a response**
 - *Request-Reply* (154) want a reply
 - *Return Address* (159) where to put reply
 - *Correlation Identifier* (163) link request to reply by id
- **Large amounts of data**
 - *Message Sequence* (170) break data into manageable chunks
- **Slow messages**
 - *Message Expiration* (176) put deadline on time-sensitive messages
- **Design data format**
 - *Format Indicator* (180) specification of message format

Command Message

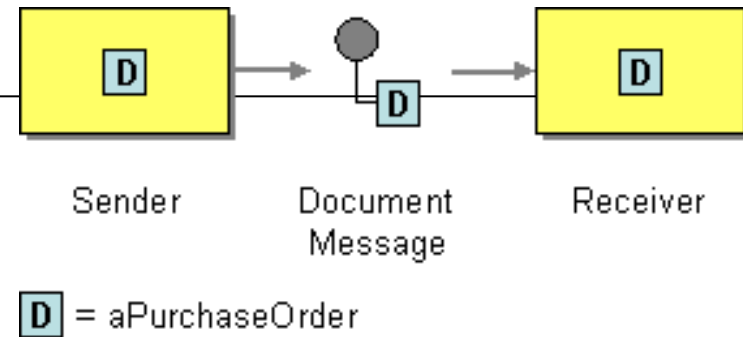


- Invoke functionality in other application
- Sender Command Message Receiver
- C** = getLastTradePrice("DIS");
- SOAP and WSDL example (EIP p. 146)
 - RPC-style SOAP message is example of *Command Message* pattern

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
  <soap:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </soap:Body>
</soap:Envelope>
```

Document Message

- Transfer data to other application

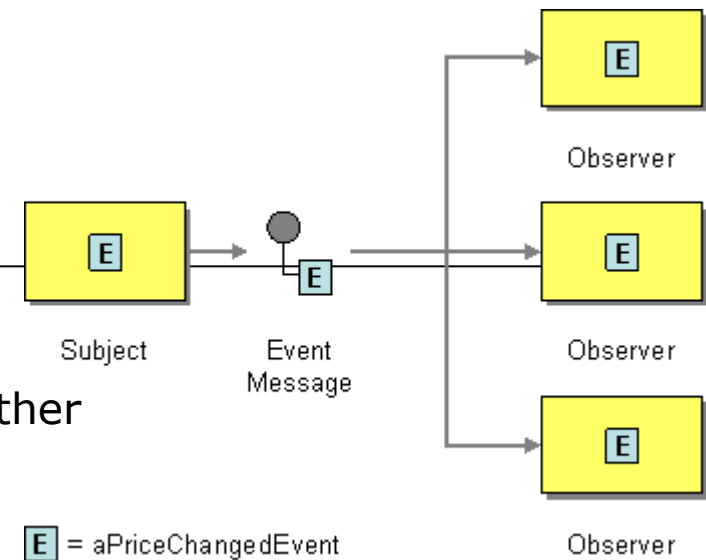


- SOAP and WSDL example (EIP p. 150)

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
  <soap:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <symbol>113.75</symbol>
    </m:GetLastTradePriceResponse>
  </soap:Body>
</soap:Envelope>
```

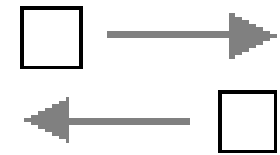
Event Message

- Transmit event from one application to another
- Many event msg. are empty; their mere occurrence tells the observer to react



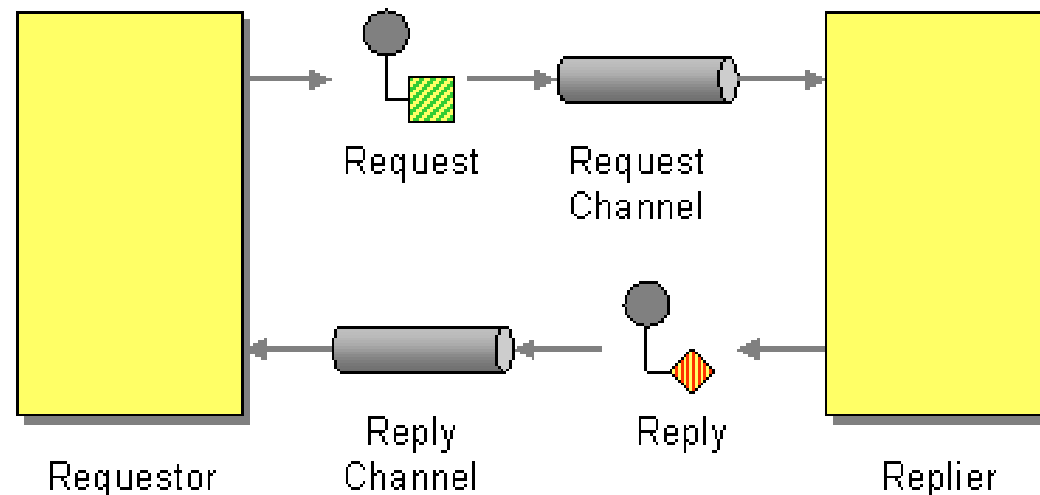
Observer Pattern

- The **push model** sends information about the change as part of the update
 - combined *Document/Event message*
- The **pull model** sends minimal information and observers can afterwards request state from the subject
 1. *Event Message* to notify observer about **update**
 2. *Command Message* send from observer to subject (**state request**)
 3. *Document Message* send from subject to observer (**state reply**)

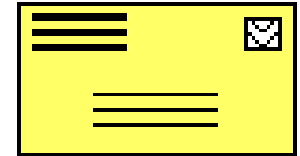


Request-Reply

- ***When an application sends a message, how can it get a response from the receiver?***
- Send a pair of *Request-Reply* messages, each on its own channel

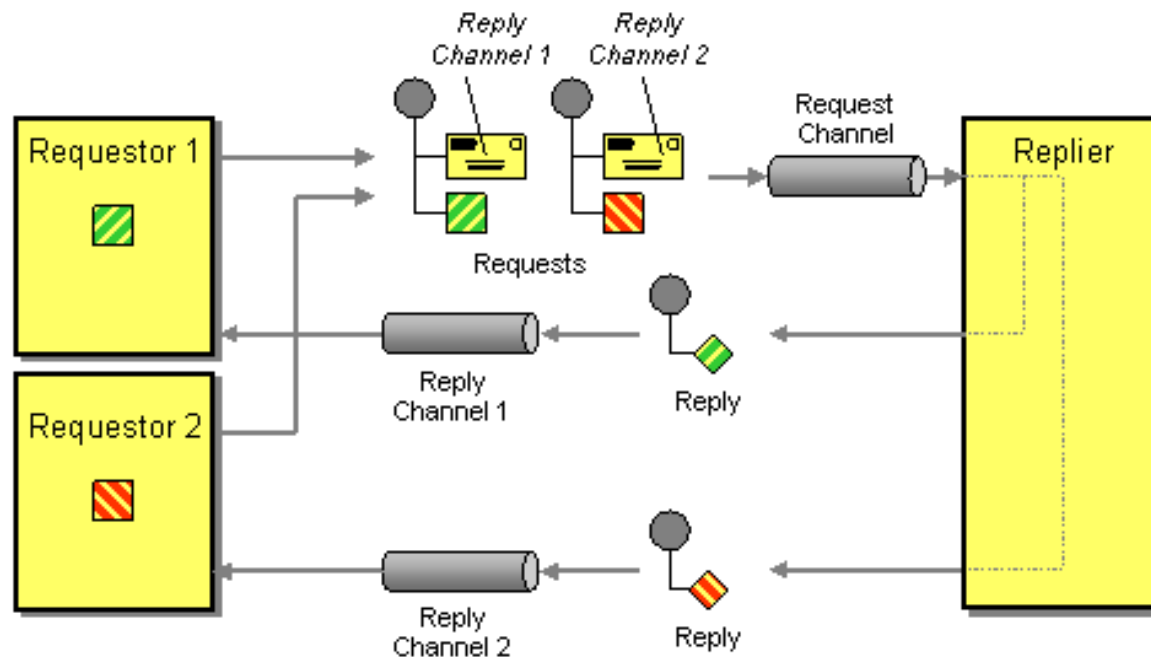


- The request channel can be a *Point-to-Point Channel* or a *Publish-Subscribe Channel*
- The reply channel is almost always point-to-point– the reply should only be returned to the requestor



Return Address

- **How does a replier know where to send the reply?**



- The request message should contain a *Return Address* that indicates where to send the reply message.

JMS Return Address Example (EIP p. 161)

JMS Sender code

```
Queue requestQueue = // Specify request destination
Queue replyQueue = // Specify reply destination
Message requestMessage = // Create request message
requestMessage.setJMSReplyTo(replyQueue);
MessageProducer requestSender = session.createProducer(requestQueue);
requestSender.send(requestMessage);
```

JMS Receiver code

```
Queue requestQueue = // Specify request destination
MessageConsumer requestReceiver = session.createConsumer(requestQueue);
Message requestMessage = requestReceiver.receive();
Message replyMessage = // Create reply message
Destination replyQueue = requestMessage.getJMSReplyTo();
MessageProducer replySender = session.createProducer(replyQueue);
replySender.send(replyMessage);
```

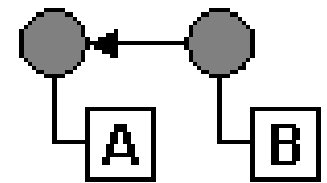

.NET Return Address Example (EIP p. 161 c+ chap 6)

.NET Sender code

```
...  
Message requestMessage = new Message();  
requestMessage.Body = "Hello world";  
requestMessage.ResponseQueue = replyQueue;  
requestQueue.Send(requestMessage);
```

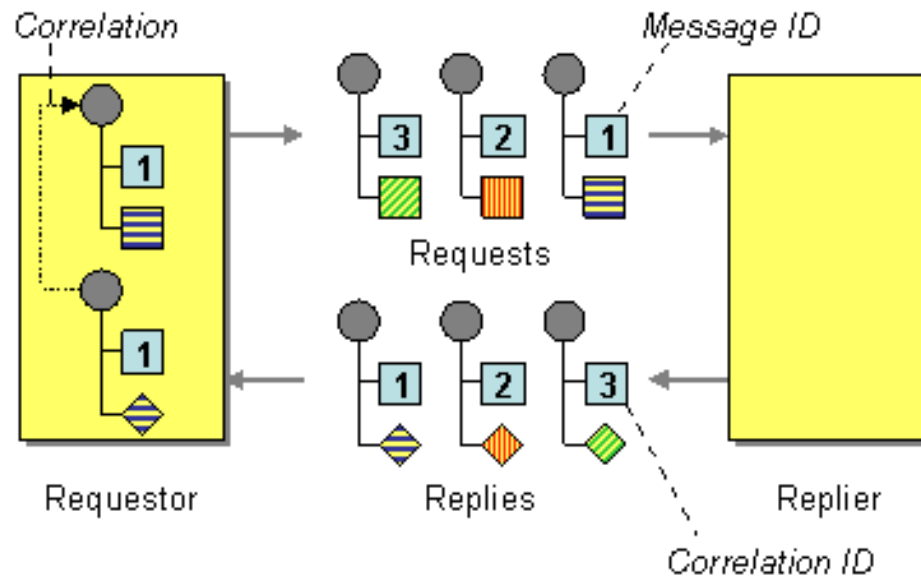
.NET Receiver code

```
...  
MessageQueue replyQueue = requestMessage.ResponseQueue;  
Message replyMessage = new Message();  
replyMessage.Body = // specify message  
replyQueue.Send(replyMessage);
```



Correlation Identifier

- How does a requestor that has received a reply know which request this is the reply for?



- Each reply message should contain a *Correlation Identifier*, a unique identifier that indicates which request message this reply is for

JMS Correlation ID Example (EIP p. 167)

The replying message can use the requesting message's message id as correlation id:

JMS Correlation ID code

```
Message requestMessage = // Get the request message
Message replyMessage = // Create the reply message
String requestID = requestMessage.getJMSMessageID();
replyMessage.setJMSCorrelationID(requestID);
```

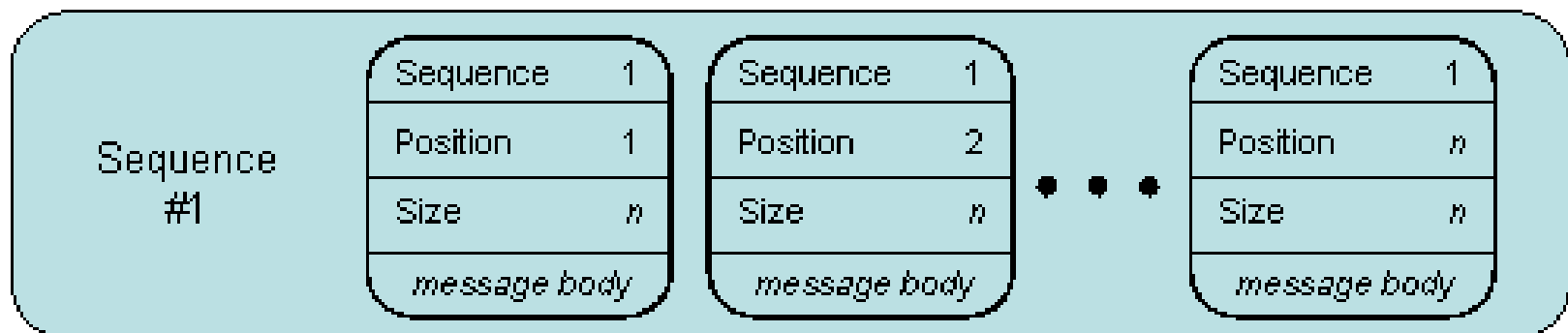
.NET

Each message in .NET has a `CorrelationId` property

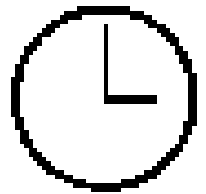


Message Sequence

- **How can messaging transmit an arbitrarily large amount of data?**

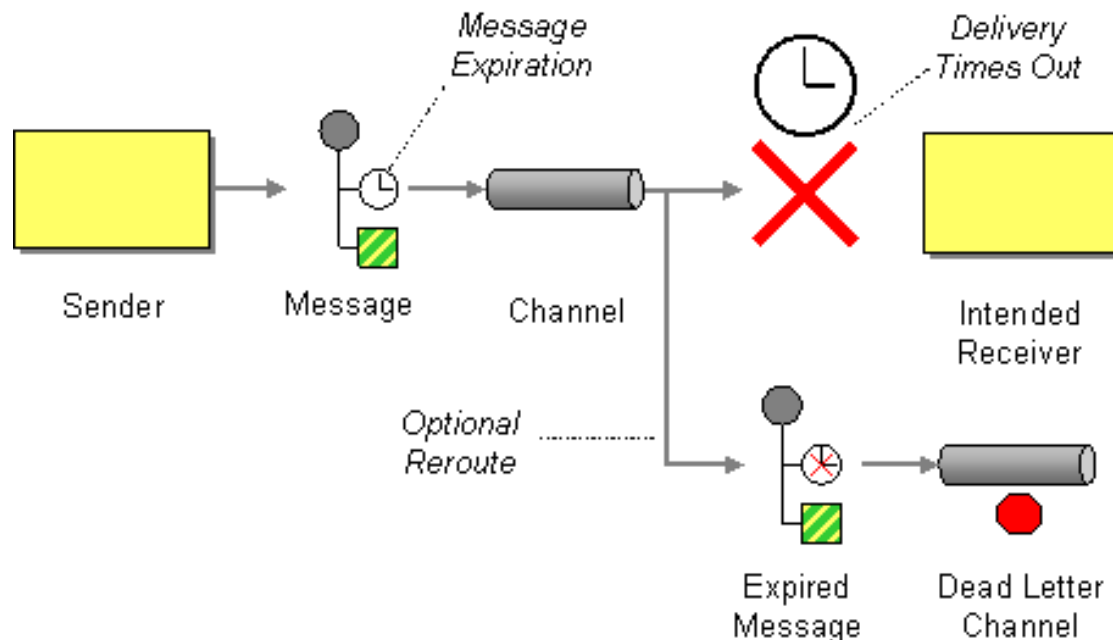


- Whenever a large set of data may need to be broken into message-size chunks, send the data as a *Message Sequence* and mark each message with sequence identification fields.



Message Expiration

- **How can a sender indicate when a message should be considered stale and thus shouldn't be processed?**



- Set the *Message Expiration* time stamp on the message
 - Like the expiration date on a milk carton ☺

Format Indicator

- **How can a message's data format be designed to allow for possible future changes?**
- Design a data format that includes a *Format Indicator*, so the message can specify what format it is using.
- Enables the sender to tell the receiver the format of the message.
- A receiver expecting several possible formats knows which one a message is using and therefore how to interpret the message's contents.

Alternative Implementations

- **Version Number.** Number or string that uniquely identifies the format
- **Foreign Key.** Unique ID (filename, URL etc.) that specifies a format document
- **Format Document.** Schema that describes the data format. Embedded in the message –not referenced by a number or a key
- Version Number or Foreign Key can be stored in the header field and has to be agreed upon

Pros & Cons

- Version number
 - Pro: Don't have to share repository
 - Con: Each part must know what descriptor is indicated and where to access it
- Foreign key
 - Pro: Foreign key is compact
 - Con: format document might need to be retrieved from remote resource
- Format Document
 - Pro: messages are self-contained
 - Con: Message traffic increases

.NET Message Formatting Examples

`IMessageFormatter` produces a stream to be written to or read from the message body.

`XmlMessageFormatter`
`ActiveXMessageFormatter`
`BinaryMessageFormatter`
`MessageQueue.Formatter`

Examples




See: [https://msdn.microsoft.com/en-us/library/system.messaging.messagequeue.formatter\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.messaging.messagequeue.formatter(v=vs.110).aspx)

XML formatter on queue

```
myQueue.Formatter = new XmlMessageFormatter(new Type[]  
    { typeof(MyProject.Order) } );
```

XML formatter on message

```
myMessage.Formatter = new XmlMessageFormatter(new String[]  
    { "System.String,mscorlib" } )
```

	Name	Description
	<code>XmlMessageFormatter()</code>	Initializes a new instance of the <code>XmlMessageFormatter</code> class, without target types set.
	<code>XmlMessageFormatter(String[])</code>	Initializes a new instance of the <code>XmlMessageFormatter</code> class, setting target types passed in as an array of (fully qualified) string values.
	<code>XmlMessageFormatter(Type[])</code>	Initializes a new instance of the <code>XmlMessageFormatter</code> class, setting target types passed in as an array of object types.

MSMQ Messaging Demo + Exercise 1

Get some basic C# message code up running on your own computer:

- **be able to send and receive messages from the same app on the same channel (dk: "hul igennem prototype")**
- see Demo.cs (C#-MSMQ example) on next slide for inspiration
- In VS project, you need to add reference to `System.Messaging` assembly (also remember `"using ..."` in *.cs file)
- After having built and run your program:
 - Find the queue that you created programmatically in your C# code via Windows computer administration under the Message Queuing service (see how on later slide)
- You must have MSMQ service up running on your computer (the Message Queuing component MSMQ is part of Windows operating system, but is not installed by default)
 - Select Add or Remove Programs from the Control Panel (see more in Coffeeshop exercise, if your are in doubt about this).

```

using System;
using System.Messaging;
namespace QueueApplication
{
    class Demo
    {
        private MessageQueue mq;
        private string myText = "Not initialized";

        private void GetChannel() {
            if (MessageQueue.Exists(@".\Private$\MyQueue1"))
                mq = new System.Messaging.MessageQueue(@".\Private$\MyQueue1");
            else mq = MessageQueue.Create(@".\Private$\MyQueue1");
            Console.WriteLine(" Queue Created ");
        }

        private void Populate() {
            Message msg = new System.Messaging.Message();
            myText = "Body text";
            msg.Body = myText;
            msg.Label = "Tine Marbjerg";
            mq.Send(msg);
            Console.WriteLine(" Posted in MyQueue1");
        }

        private string GetResult() {
            Message msg;
            string str = "";
            string label = "";
            try {
                msg = mq.Receive(new TimeSpan(0, 0, 50));
                msg.Formatter = new XmlMessageFormatter(new String[] { "System.String,mscorlib" });
                str = msg.Body.ToString();
                label = msg.Label;
            }
            catch { str = " Error in GetResult()"; }
            Console.WriteLine(" Received from " + label);
            return str;
        }

        static void Main(string[] args) {
            Demo d = new Demo();
            d.GetChannel();
            d.Populate();
            string result = d.GetResult();
            Console.WriteLine(" send: {0} ", d.myText);
            Console.WriteLine(" receive: {0} ", result);
            Console.ReadLine();
        }
    }
}

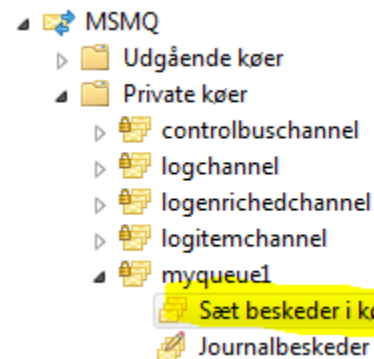
```

C# MSMQ example
I'm aware the code lacks line shifts here and there for readability, but I wanted to squeeze it into one slide 😊

Computer Administration of Message Service

- To see all message channels defined on your computer and how many messages they hold, do the following:
 1. Right-click on "My Computer" and chose Manage
 2. Expand the node "Services and Applications"
 3. Expand the node "Message Queuing"
 4. Expand the node "Private Queues"

Right-clicking on "Sæt beskeder i kø" shows all the messages in that particular queue (or you can empty the queue)



MSMQ Messaging Exercise 2

Make a [Rock-paper-scissors](#) game with two applications that represent each their hand.

The applications communicate via MSMQ.

You can use "Player One" project for inspiration

- be aware that you need two components running – one for each hand
- Extra: It would be nice if the game can run in a loop and not just once (find inspiration for how to do that in exercise 3)

MSMQ Messaging Exercise 3

Implement publisher-subscriber pattern with Multicasting.

See example here:

<https://www.codeproject.com/Articles/871746/Implementing-pub-sub-using-MSMQ-in-minutes>

MSMQ Messaging Exercise 4A

Make an application that can put an Order object in a message and put it on a message queue.

You can specify a formatter on either the queue or the message.

XML formatter on queue

```
myQueue.Formatter = new XmlMessageFormatter(new Type[]  
    { typeof(MyProject.Order) } ) ;
```

XML formatter on message

```
myMessage.Formatter = new XmlMessageFormatter(new String[]  
    { "System.String,mscorlib" } )
```

MSMQ Messaging Exercise 4B

Make a request-reply messaging implementation using the following patterns (look for inspiration in [EIP chap 6](#)):

1. *Return Address* specify reply channel on request message
2. *Correlation-id* specify id on reply message for identification
3. *Format Indicator* Specify data format on message
4. *Request-reply* Receiver app and Reply app must run as two separate components.

When an order message sent by client app is received by order app, the order app sends response with delivery time. The client app picks up answer on reply channel and outputs the delivery time on the screen. You need min. two queues 😊