Implementing a Service Bus with MassTransit

Roland Guijt
INDEPENDENT SOFTWARE DEVELOPER AND TRAINER

@rolandguijt   www.rmgsolutions.nl

Module Overview

Sending and receiving

Service bus concepts

Type support

Scheduling

Monitoring

Dependency Injection

Failure

Request/Response

Service Bus Framework for .NET

Endpoints and queues

Not like Biztalk

Gateway to transport

Multiple transports

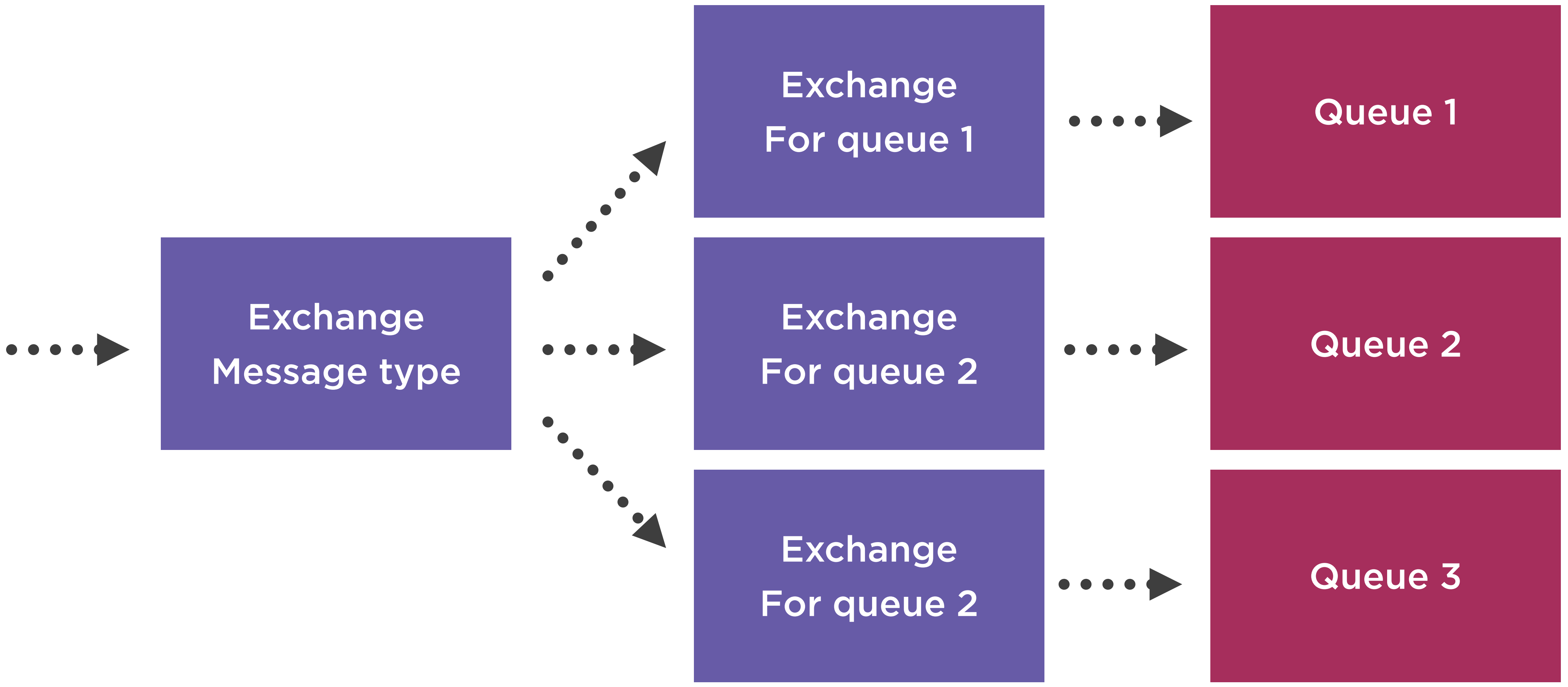
Optimized for testing

Built-in features

Service Bus vs Native Transport API

Service Bus	Native Transport API
A ready to go framework for messaging	Low level: Create framework yourself
Hides complexity of transport	Exposure to complexity of transport
Part of transport features supported	Full transport feature support
More geared towards type system	Heavy use of strings
Supports multiple transports	One transport supported
Easy to unit test	Not designed with unit testing in mind

MassTransit and RabbitMQ



Scheduling Messages

Delivery of messages at a later time

Quartz.net

MassTransit Quartz service

In memory

Scheduling in Code

```
cfg.UseMessageScheduler(new Uri(“rabbitmq://localhost/quartz”));
or
cfg.UseInMemoryMessageScheduler();


context.ScheduleMessage(destination, when, message);
or
schedulerEndpoint.ScheduleSent(destination, when, message);
```

Monitoring

RabbitMQ management plugin

Observer interfaces

Performance counters

Observer Interfaces

Intercept messages

Read only

IReceiveObserver

IConsumeObserver

IConsumeMessageObserver<T>

ISendObserver

IPublishObserver

Message Observers in Code

```
public interface ISendObserver
{
    Task Presend<T>(SendContext<T> context);
`   Task PostSend<T>(SendContext<T> context);
    Task SendFault<T>(SendContext<T> context, Exception exception);
}

var observer = new SendObserver();
bus.ConnectSendObserver(observer);
```

Bus Observer

Observes bus activities

Implement IBusObserver

Register with cfg.BusObserver

Performance Counters

```
bus.EnablePerformanceCounters();
```

Dependency Injection

Avoid using bus object

Extra NuGet package

Adds extension methods to IReceiveEndpointConfigurator

Autofac, Ninject, StructureMap, Unity, Castle Windsor

Dependency Injection in Code

```
//create container
//register consumers
cfg.ReceiveEndpoint(“queuename”, e =>
{
    e.LoadFrom(container);
    or
    e.Consumer<ConsumerType>(container);
}
```

Responding to Failure

Connection management

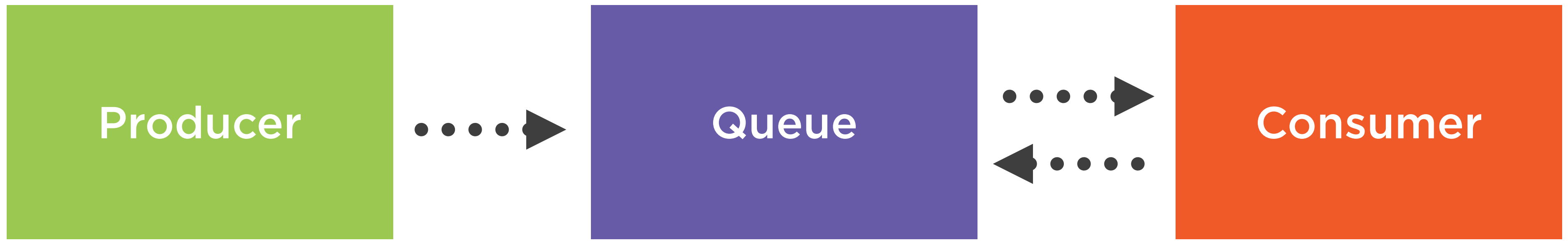
Skipped queue

Retries

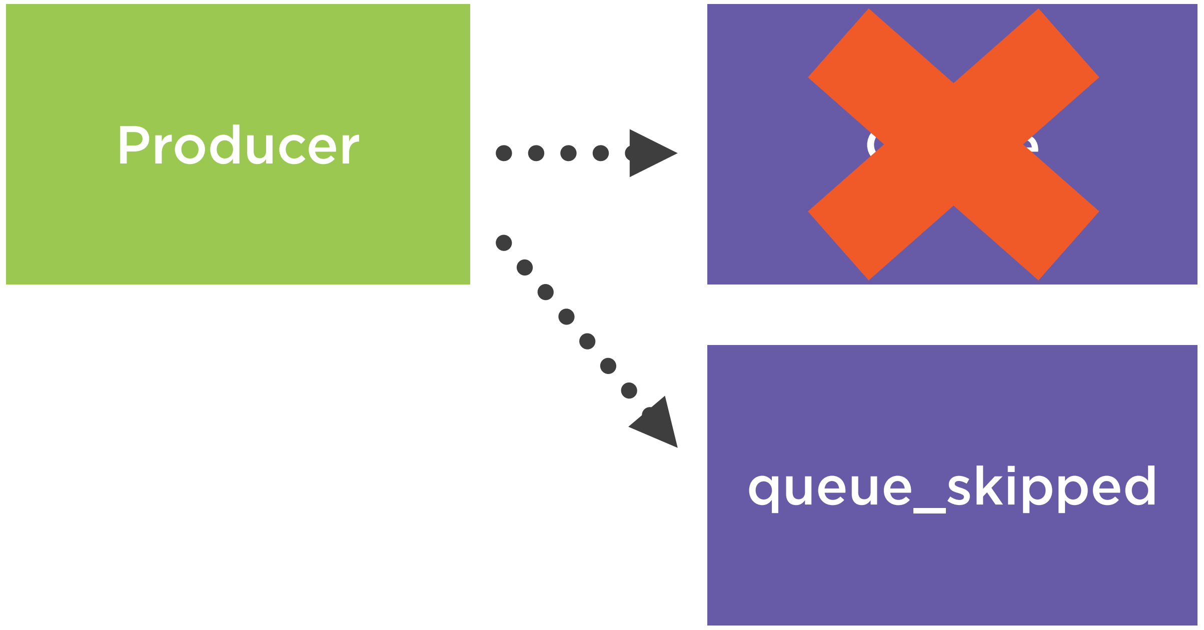
Error queue

Fault<T> message

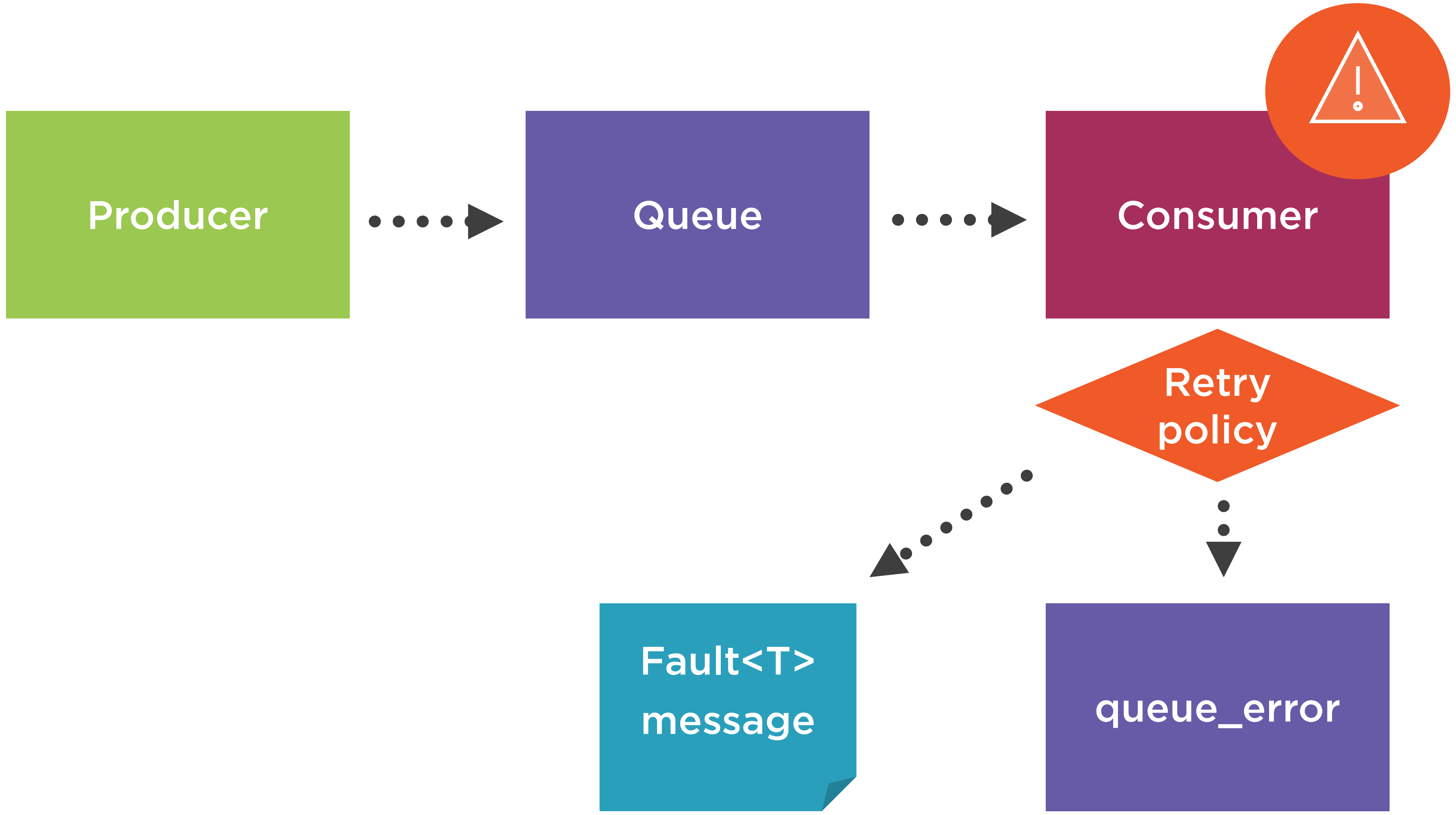
Happy Flow



Delivery Problem



Consumer Problem



How to Specify Fault and Response Addresses

```
context.Publish<IOrderRegisteredEvent>(orderRegisteredEvent,
    c => c.FaultAddress = urlToEndpoint);

context.Publish<IOrderRegisteredEvent>(orderRegisteredEvent,
    c => c.ResponseAddress = urlToEndpoint);
```

How to Set a Retry Policy

```
cfg.ReceiveEndpoint(host, queuename, e =>
{
    e.UseRetry(Retry.Except<ArgumentException>().Immediate(20));
    e.Consumer<OrderRegisteredConsumer>();
});
```

Exception Selectors

- **Except**
- **Selected**
- **All**
- **Filter**

Retry Policies

Immediate

Intervals

Exponential

Incremental

Request/ Response

Producer waits until consumer replies

Against asynchronous nature of Microservices

Supported with MessageRequestClient

Awaitable

Request/Response in Code

```
var client = new MessageRequestClient<commandMessageType,
    resultMessageType>(bus, address, requestTimeout);


var result = await client.Request(commandObject);
```

Summary

How to send and receive messages

Service bus pros and cons

MassTransit features

Failure

Request/Response