# What a modern mailman should know when delivering messages!

_

BY ROB VAN PAMEL

XX_

## Rob Van Pamel

- .NET Consultant for AXXES
- Developer since 2007
- AWS Community Builder since 2023
- Started in Winforms application ... till cloud
- Greenfield project: open vision AWS Selected

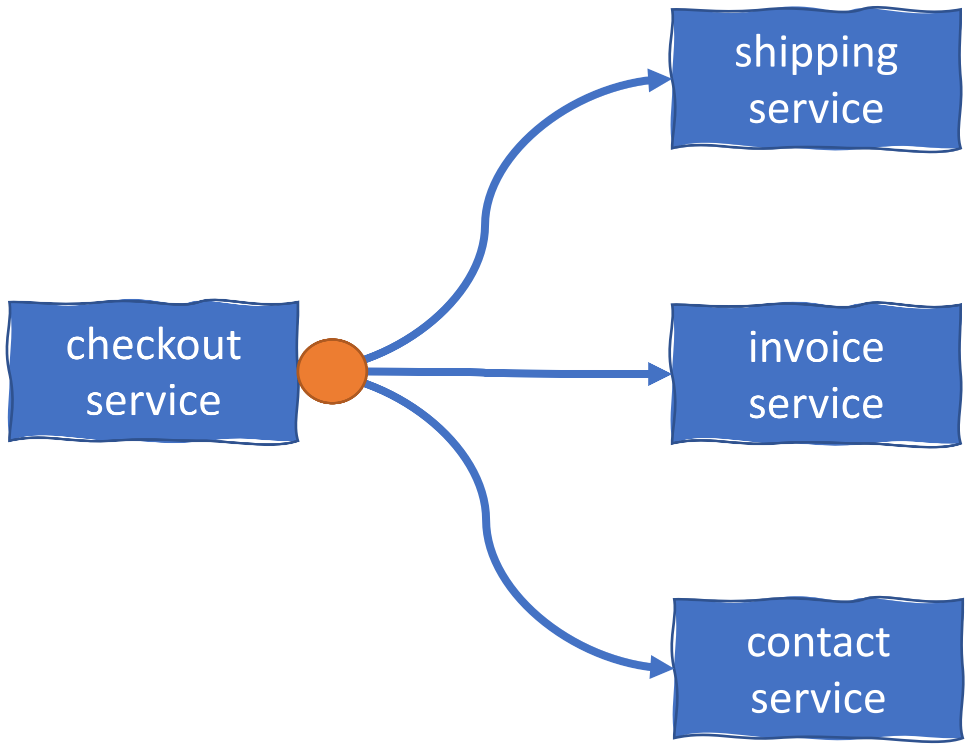
rob.vanpamel@gmail.com

@robvanpamel

AXXES_

## Synchronious communication

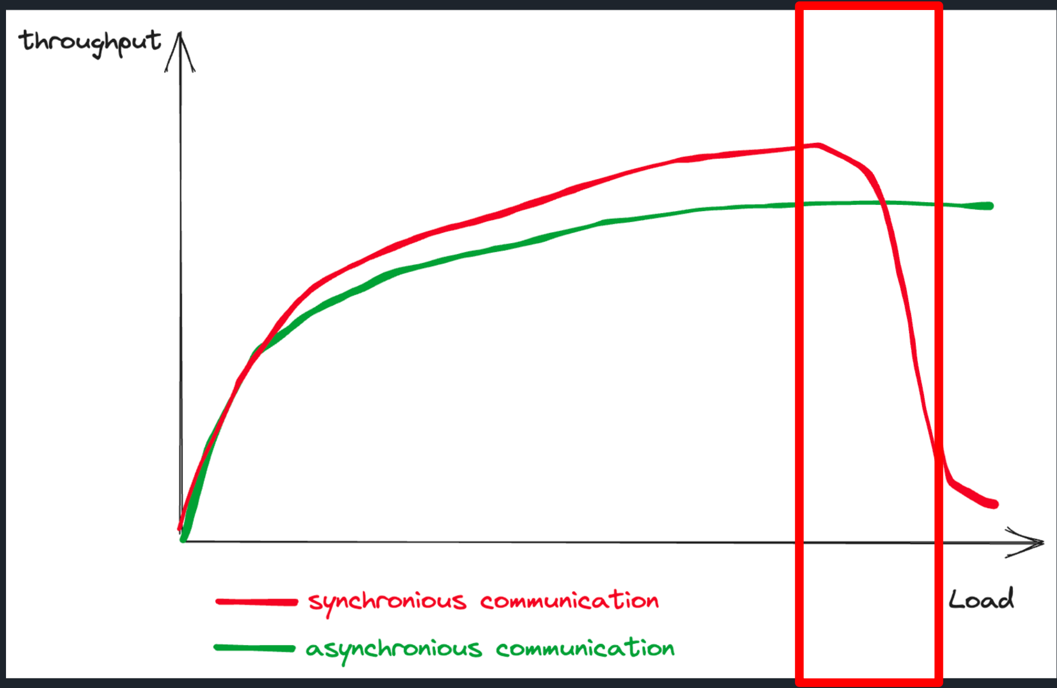
- Request-response style
  - Create the Invoice!
- Services are coupled
- More services are added over time
- More responsibility for teams
- Latency increases



AXXES_

## Why bother with async communication?

- Evolution from monolitic applications to service oriented
- Use of REST / gRPC with retry mechanisms
- Scalabillity
- Performance / Throughput
  - Sync communication collapses after a given point
  - *Unable to connect to remote host: Connection refused*
- Decoupling



AXXES_

Join at menti.com use code 6666 0244

Mentimeter

## Why are you using messaging?

Waiting for responses

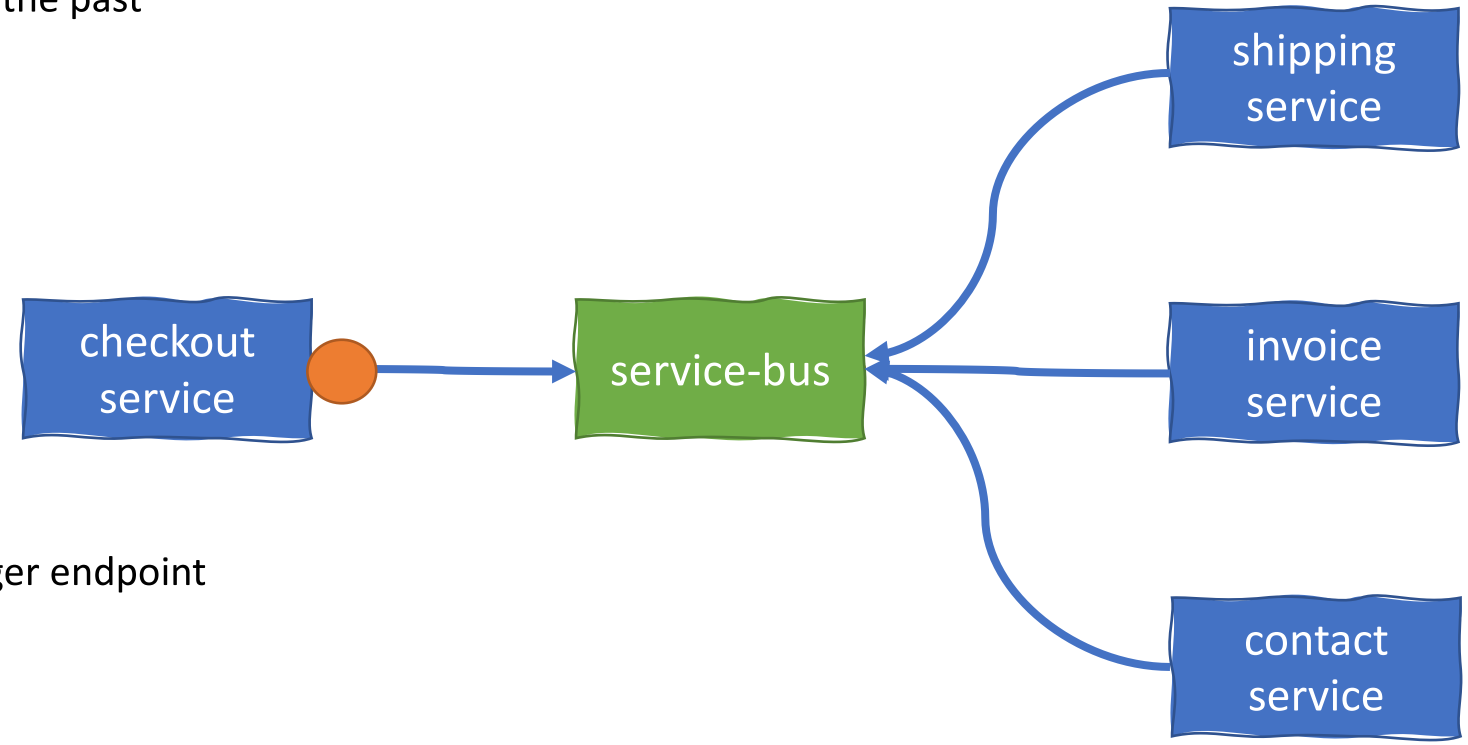
GO TO
menti.com
ENTER THE CODE
6666 0244

## Risks of sync communication ?

- Fallacies of Distributed Systems (Dr. Harvey)
    - The network is reliable
    - Latency is zero
    - Bandwidth is infinite
    - The network is secure
    - Topology doesn't change
    - There is one administrator
    - Transport cost is zero
    - Network is homogeneous

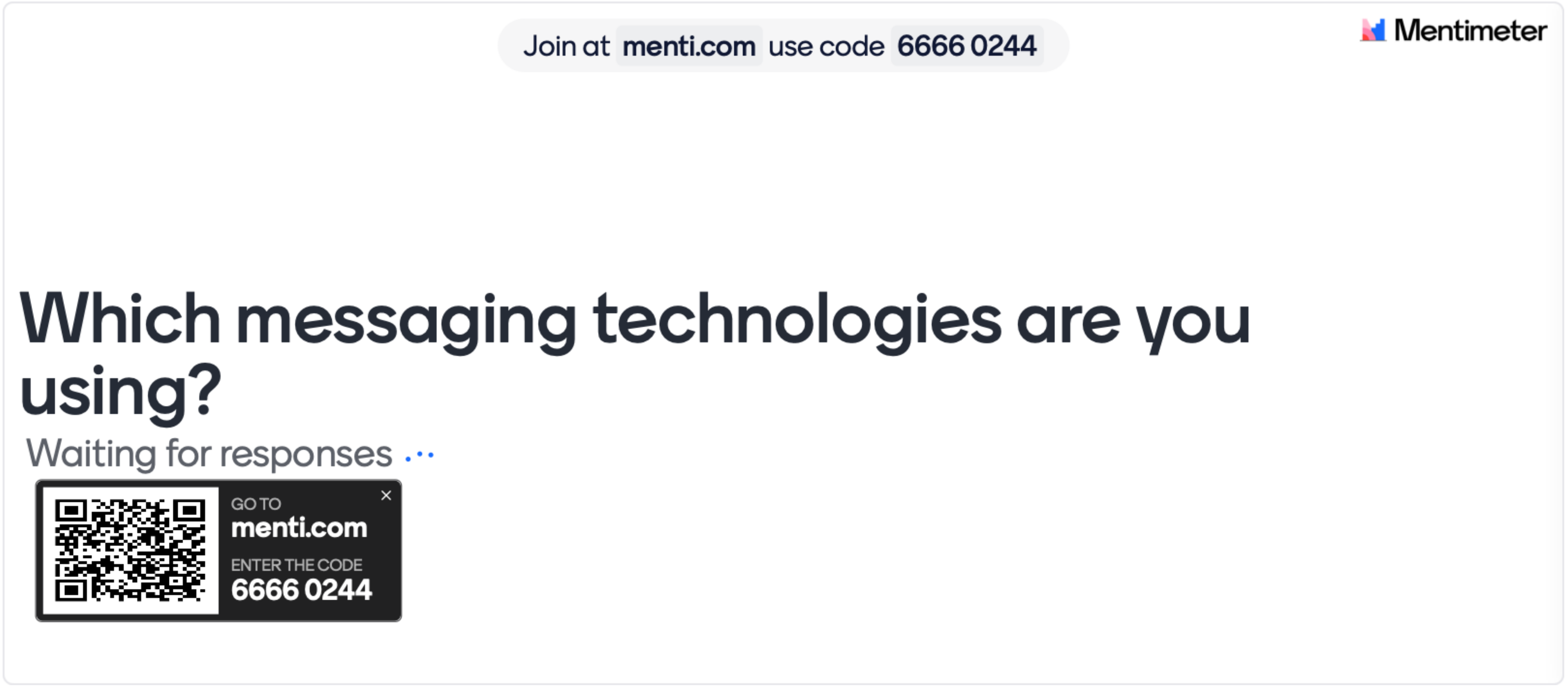
## Message Async Driven Architecture

- Introduce events
  - Change in state that happened in the past
  - Async
  - Decouple Services
- Focus on communication
  - Reverse the flow
  - Eg a service bus
  - ‘Broadcast the message’ and trigger endpoint



AXXES_

## Which messaging technology are you using ?



Join at menti.com use code 6666 0244

Mentimeter

Which messaging technologies are you using?

Waiting for responses

GO TO
menti.com
ENTER THE CODE
6666 0244

AXXES_

EventBridge

NServicebus

Brighter

Amazon SQS

HTTP

Sagas

MQTT

RabbitMQ

QOS

Outbox

Amazon MQ

Amazon SNS

MSMQ

AMQP

Rebus

ActiveMQ

Idempotence

MassTransit

EventBridge

NServicebus

Brighter

Amazon SQS

HTTP

Sagas

MQTT

RabbitMQ

QOS

Outbox

Amazon MQ

Amazon SNS

MSMQ

AMQP

Rebus

ActiveMQ

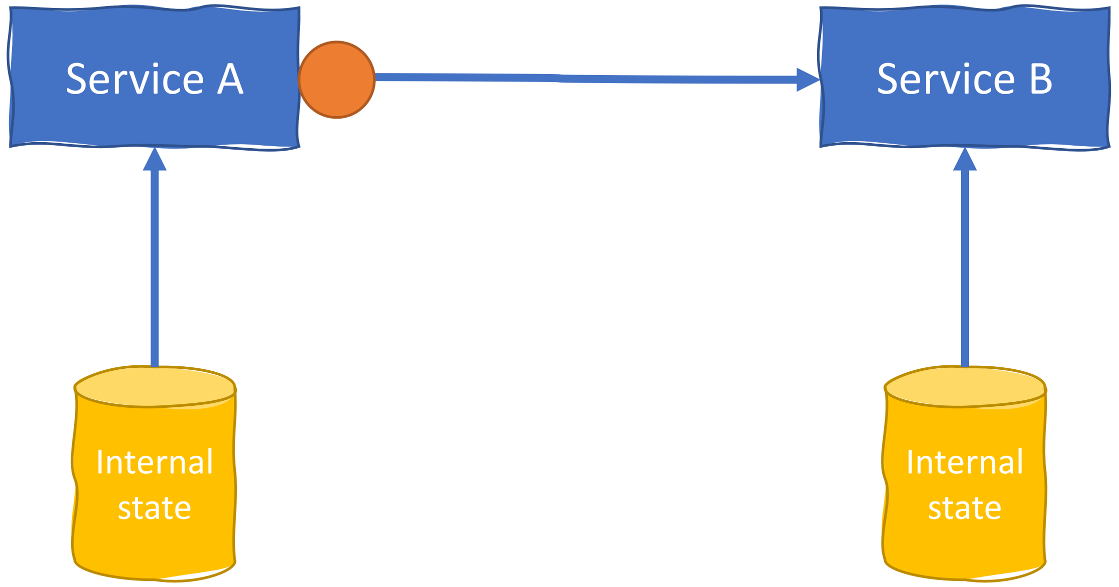
MassTransit

Idempotence

## Patterns and principles

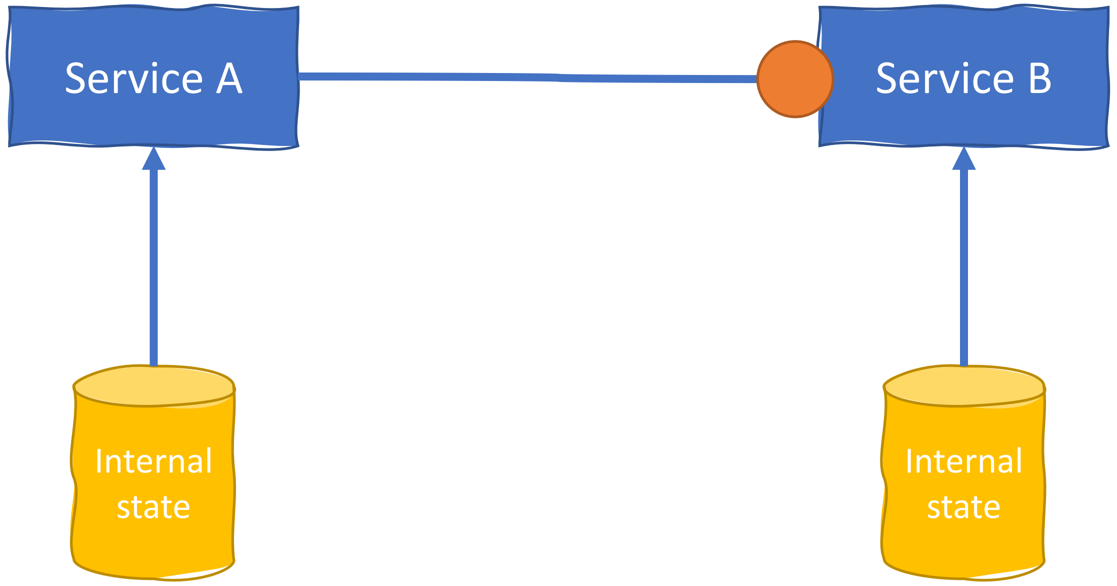
–

## Quality Of Service (QOS)



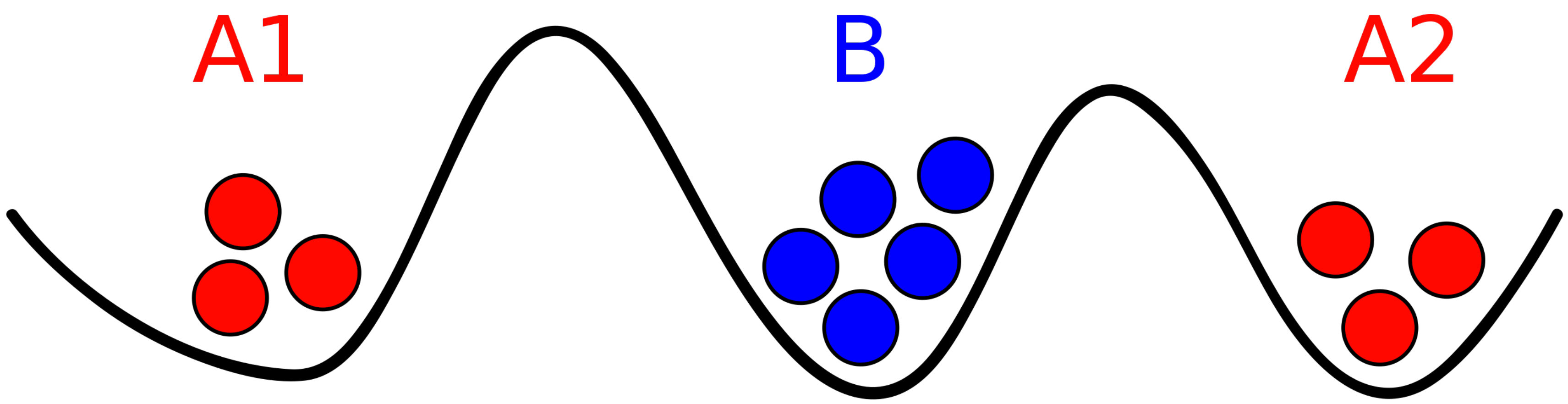
## Quality Of Service (QOS)

- Exactly Once ( QOS 2 )



## Quality Of Service (QOS)

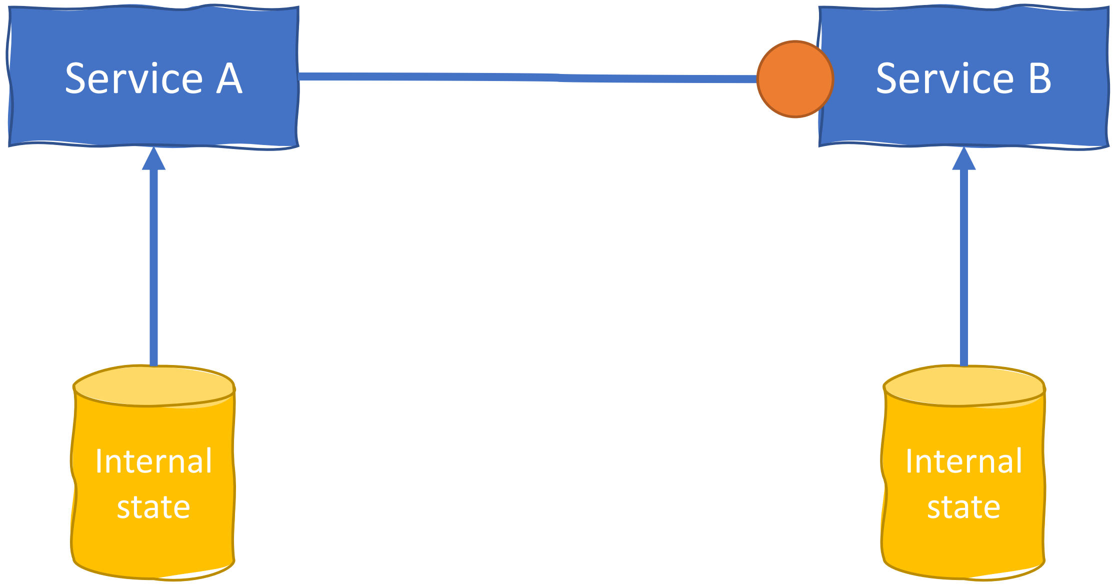
- Exactly Once ( QOS 2 )
- Problem of 2 generals



AXXES_

## Quality Of Service (QOS)

- Exactly Once ( QOS 2 ) ?
- At least once ( QOS 1)
- At most once  (QOS 0)



AXXES_

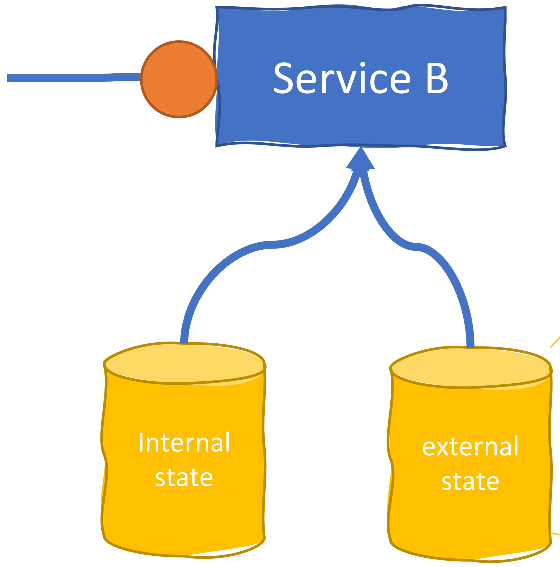
## Idempotence

–

is the property of *certain operations* in mathematics and computer science, that can *be applied multiple times without changing the result beyond the initial application*.

## Exactly once processing – Idempotence (Receiving side)

- Internal state vs external state
- Adding unique message identifier



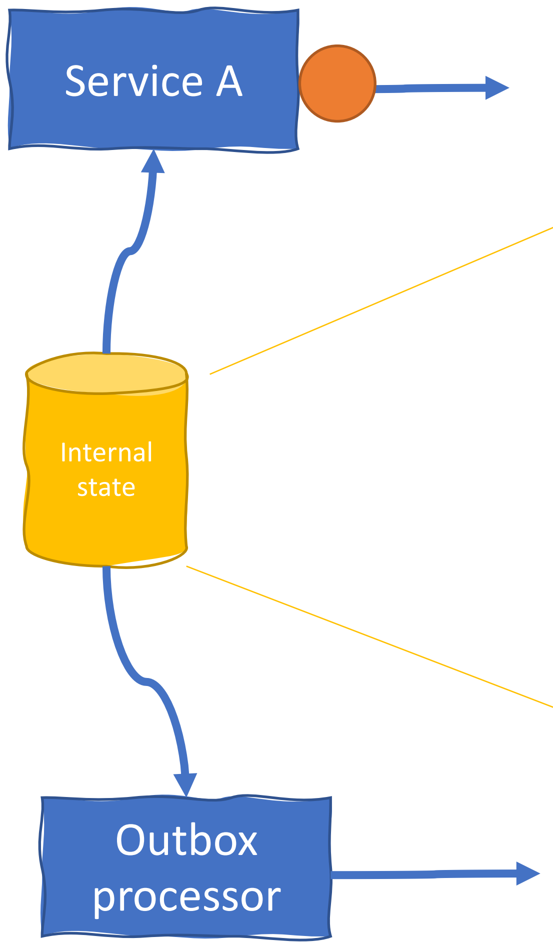
| MessageId | Outgoing Messages |
|---|---|
| F79a6822f5…7b6aa | [{ messageid: 2d73602b6ca31e messagetype: orderconfirmed body: { orderid: 23 productid: 89440 quantiy:4 } }] |
| 96a6bc22f5…7b6aa | |
| C4616c6e04be3c…fb06d | [{ messageid: 2d73602b6ca31e messagetype: ordercancelled body: { orderid: 23 } }] |

AXXES_

## Exactly once processing – Outbox (Sender side)

Outbox

- Process BL and send message atomically
- Single responsibility
- Store messages in along with BL
    - Identifier
    - Event type
    - OccuredAt
    - Payload
    - Sent
- Outbox processor publishes messages



| Id | payload | eventtype | occuredAt | sent |
|---|---|---|---|---|
| 456 | { orderid: 23 productid: 89440 quantiy:4 } | orderconfirmed | ... | 1 |
| 457 | body: { orderid: 23 } | ordercancelled | ... | 1 |
| 458 | { orderid: 24 productid: 34523 quantiy:2 } | orderconfirmed | ... | 0 |

AXXES_

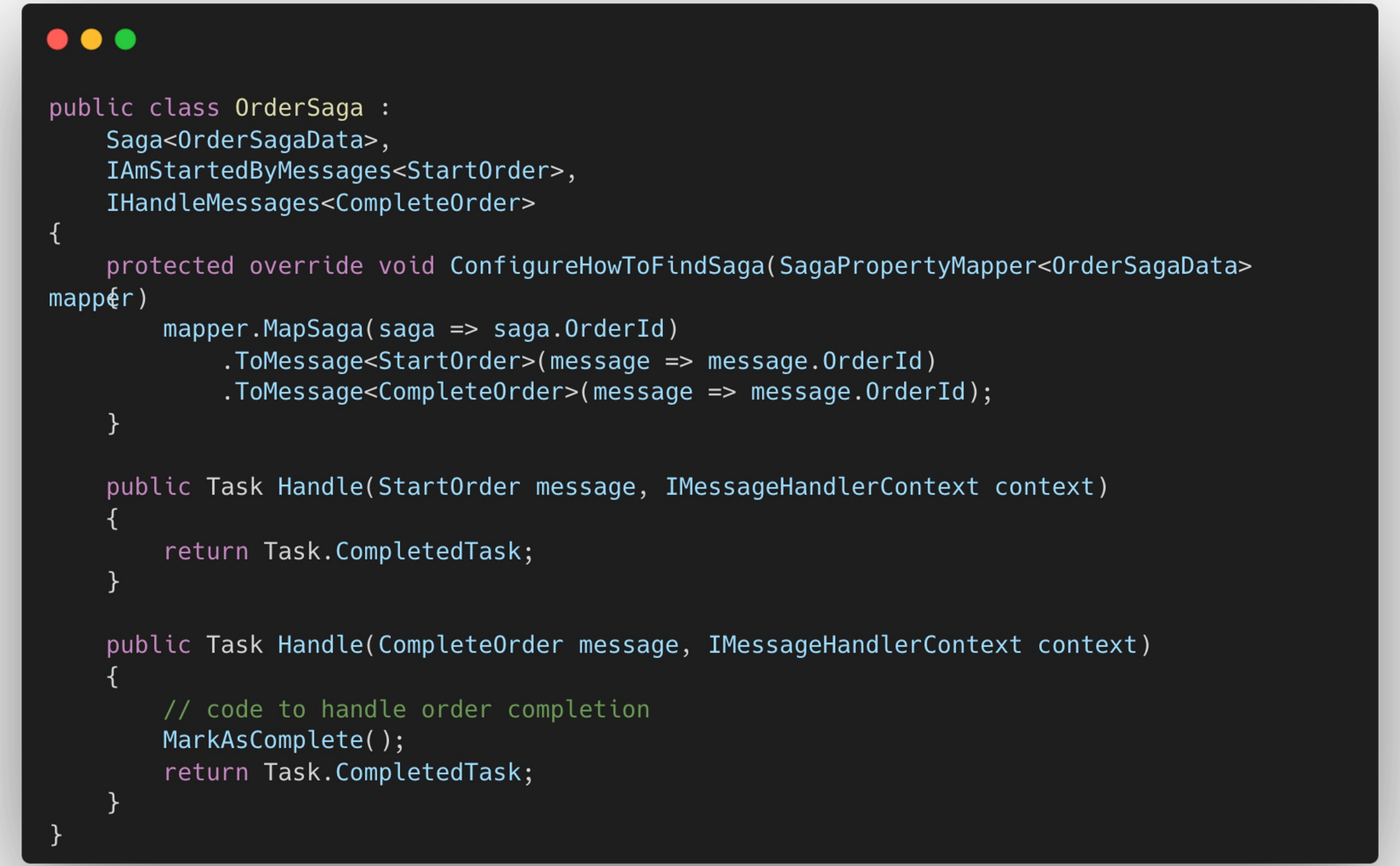
## Saga’s

–

## Sagas: maintain intigrity over multiple services

- Long-running business processes - Business processes that span multiple services
- No distributed transactions ( 2PC )
- State machine
- Require storage for saga state
- Compensating actions for failures
- AWS Step Functions

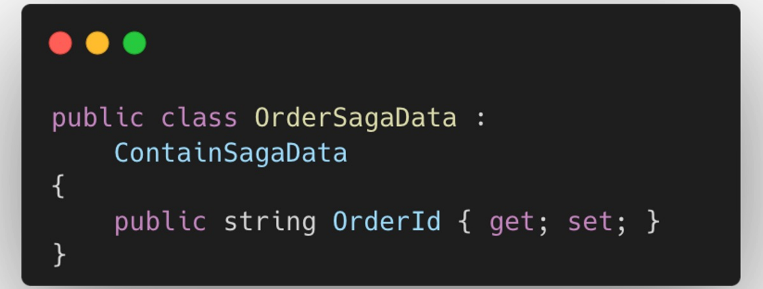
## Sagas



```
public class OrderSaga :
    Saga<OrderSagaData>,
    IAmStartedByMessages<StartOrder>,
    IHandleMessages<CompleteOrder>
{
    protected override void ConfigureHowToFindSaga(SagaPropertyMapper<OrderSagaData>
mapper)
        mapper.MapSaga(saga => saga.OrderId)
            .ToMessage<StartOrder>(message => message.OrderId)
            .ToMessage<CompleteOrder>(message => message.OrderId);
    }

    public Task Handle(StartOrder message, IMessageHandlerContext context)
    {
        return Task.CompletedTask;
    }

    public Task Handle(CompleteOrder message, IMessageHandlerContext context)
    {
        // code to handle order completion
        MarkAsComplete();
        return Task.CompletedTask;
    }
}
```



```
public class OrderSagaData :
    ContainSagaData
{
    public string OrderId { get; set; }
}
```

EventBridge

NServicebus

Brighter

Amazon SQS

HTTP

MQTT

RabbitMQ

Amazon MQ

Amazon SNS

MSMQ

AMQP

Rebus

ActiveMQ

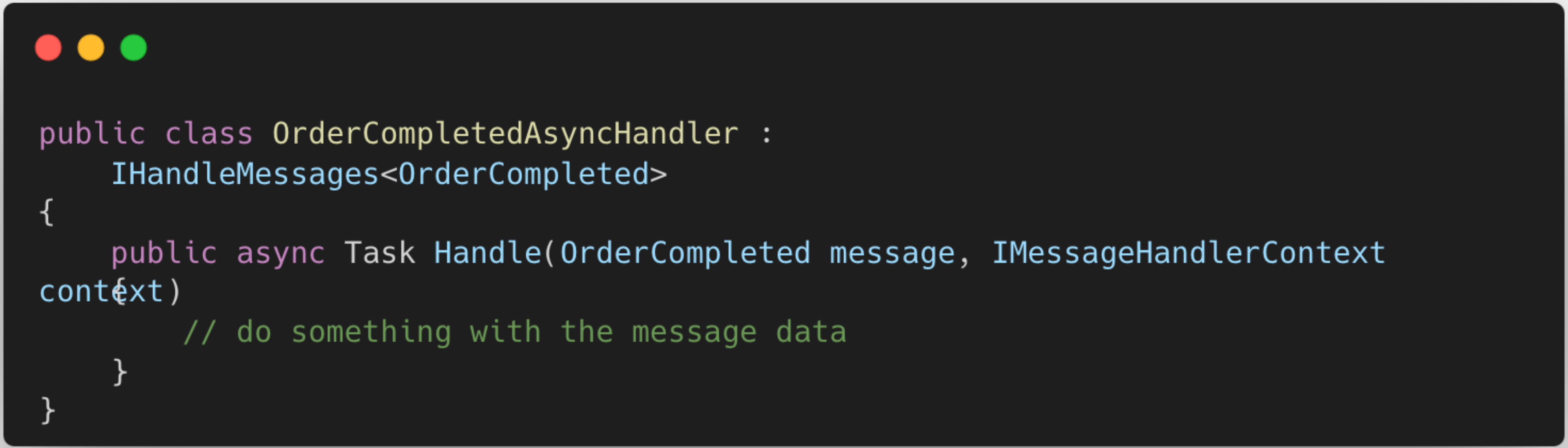
MassTransit

## Application Messaging Frameworks

–

## Application Messaging Frameworks

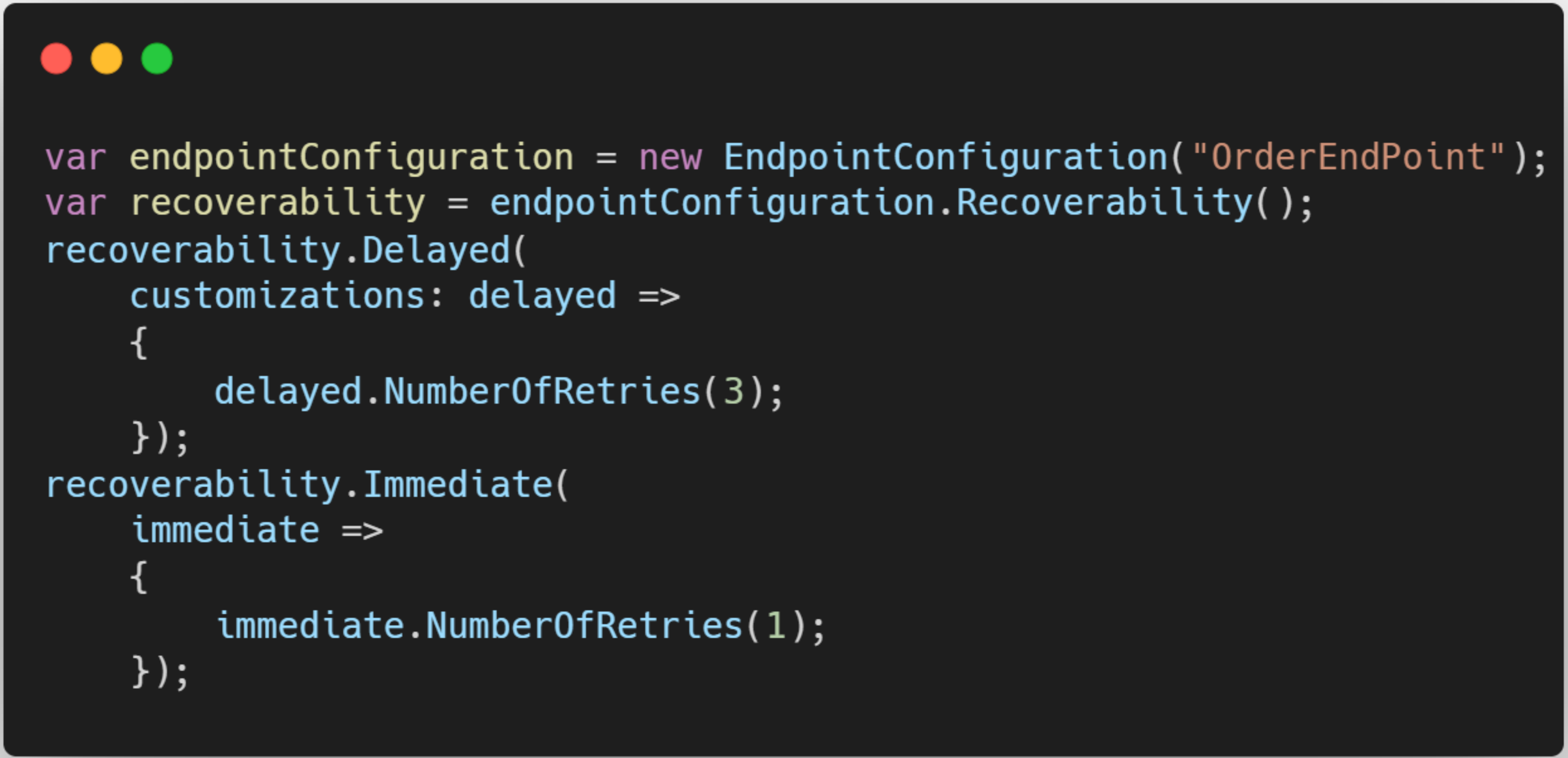
- Abstraction layer for message exchange
    - Transport details
    - Message handling



```
public class OrderCompletedAsyncHandler :
    IHandleMessages<OrderCompleted>
{
    public async Task Handle(OrderCompleted message, IMessageHandlerContext
context)
    {
        // do something with the message data
    }
}
```

## Application Messaging Frameworks

- Abstraction layer for message exchange
  - Transport details
  - Message handling
  - Routing
  - Serialisation
  - Exception Management
  - Retries & Poison Messages



```
var endpointConfiguration = new EndpointConfiguration("OrderEndPoint");
var recoverability = endpointConfiguration.Recoverability();
recoverability.Delayed(
    customizations: delayed =>
    {
        delayed.NumberOfRetries(3);
    });
recoverability.Immediate(
    immediate =>
    {
        immediate.NumberOfRetries(1);
    });
```

## Application Frameworks: NServiceBus

- Transport
  - RabbitMQ
  - SQS
  - MSMQ
  - Azure Service Bus / Azure Storage Queues
  - SQL Server
- Patterns
  - Outbox
  - Sagas
  - Idempotence
- Commercial License

## MASSTRANSIT

- Transport
  - RabbitMQ
  - SQS
  - Azure Service Bus
  - ActiveMQ
  - Kafka
- Patterns
  - Outbox
  - Sagas
  - Idempotence
- Apache 2.0 License

## REBUS

- Transport
  - RabbitMQ
  - MSMQ
  - SQS
  - SQSAndSNS
  - Azure Service Bus
- Patterns
  - Outbox
  - Sagas
  - Idempotence
- Apache 2.0 License

## Brighter

- Transport
  - RabbitMQ
  - Ports and Adapters
- Patterns
  - Outbox
  - Sagas
  - Idempotence
- Apache 2.0 License

## Application Messaging Frameworks

–

Demo

EventBridge

Amazon SQS

HTTP

MQTT

RabbitMQ

Amazon MQ

Amazon SNS

MSMQ

AMQP

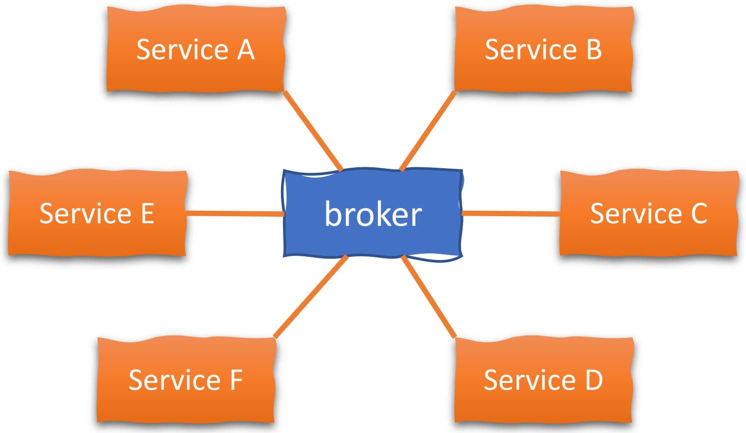
ActiveMQ

## Transports

–

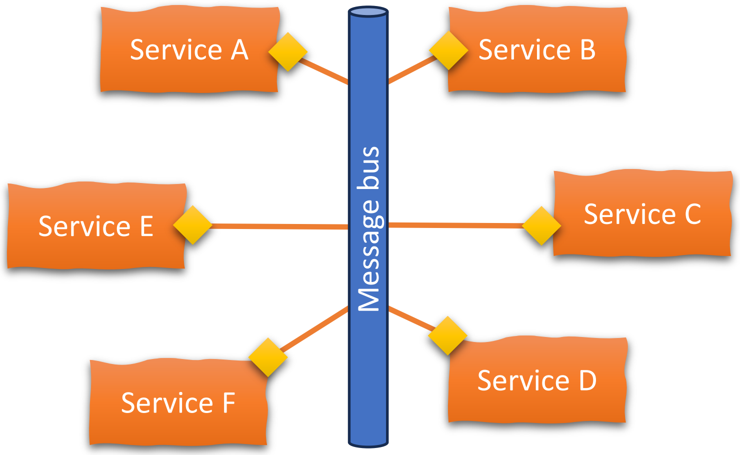
## Broker

- Centralised messaging system
- Receive from multiple sources
- Distribute to different system
- Routing included
- Always-on



## Bus

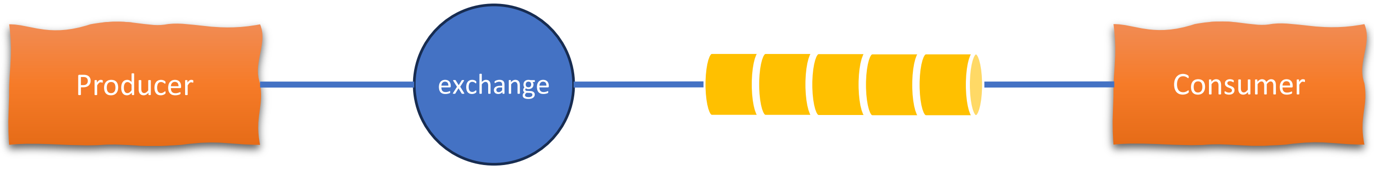
- Distributed
- Shared message agreed upon message schema
- Smart endpoint - dumb pipes
- Application Frameworks take care of routing
- Serverless



AXXES_

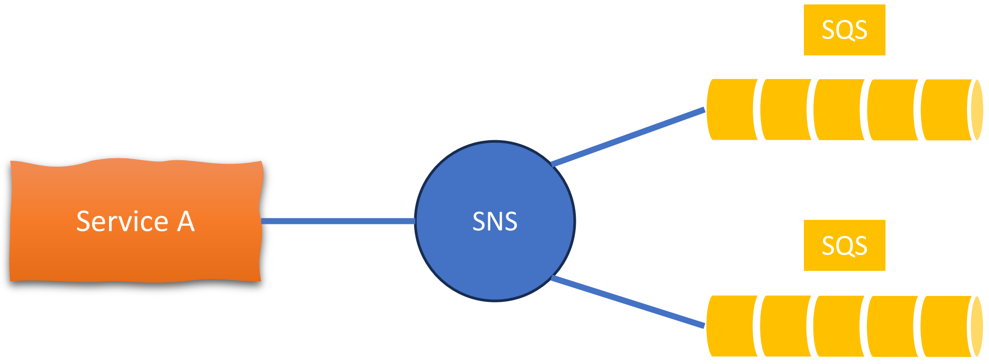
## Amazon MQ

- Managed service for broker systems RabbitMQ or ActiveMQ
- RabbitMQ
  - Supports multiple protocols*
  - AMQP 0-9-1 core protocol
  - Support for multiple scenario
    - Publish / Subscribe : Exchange type=Fanout
    - Routing: Exchange type=direct
    - Topics : Exchange type=topic
    - Competing consumers
    - Request / Reply
- ActiveMQ
  - Extensive Java support
- Clustering support



## Amazon SQS and SNS

- Bus systems
- Simple Notification System
  - Publish / Subscribe notification
  - Fan out system
  - Endpoints:
    - HTTP
    - SQS
    - Email
    - SMS
- Simple Queue System
  - Queue
  - Subscriptions
- Managed services by AWS
- Mostly used in combination ( Fanout by SNS – Queue Store by SQS )



AXXES_

## Amazon EventBridge

- Serverless bus systems built on top of CloudWatch Events
- Event Bus – pipeline
    - Routing and filtering
    - Default custom vs partner
- Events
- Source and Targets
- Rules
- EventSchema

HTTP

MQTT

MSMQ

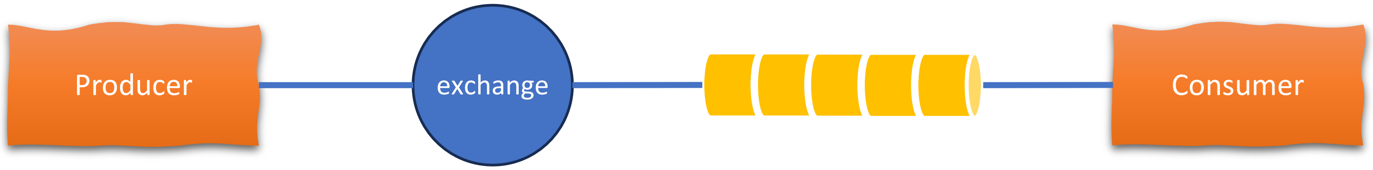
AMQP

## Protocols under the hood

–

# Async Messaging Transports - Advanced Message Queuing Protocol (AMQP)

- Built specific for enterprise messaging
- Built in the banking industry
- Binary protocols
- Support for multiple scenario
  - Publish / Subscribe : Exchange type=Fanout
  - Routing: Exchange type=direct
  - Topics : Exchange type=topic
  - Competing consumers
  - Request / Reply



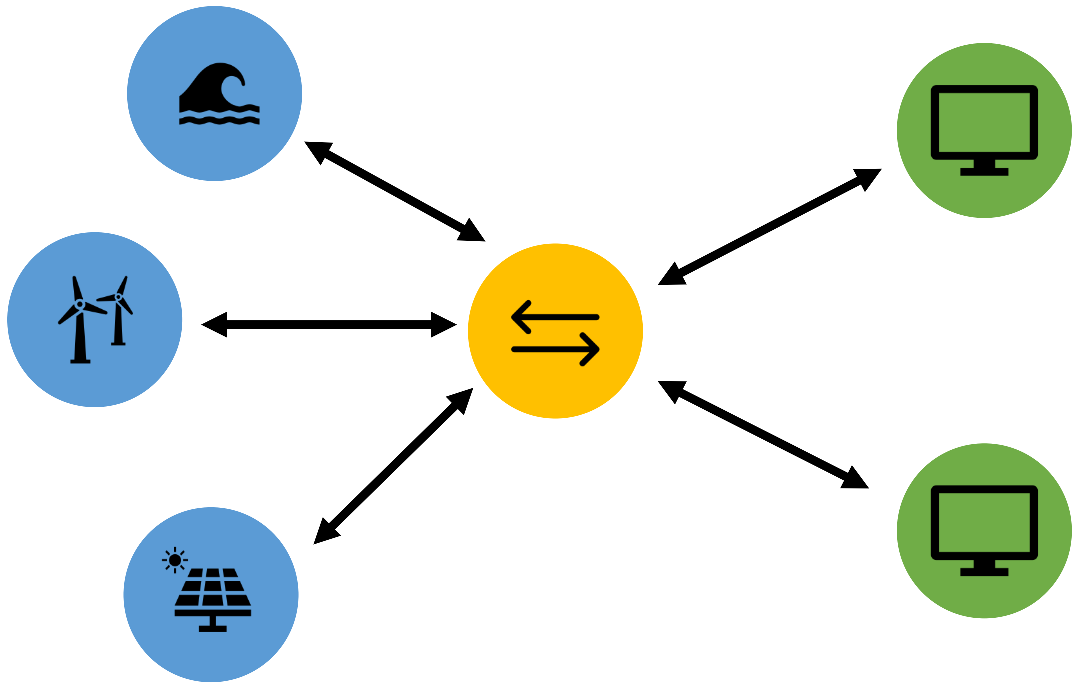
AXXES_

## Async Messaging Transports - HyperText Transfer Protocol (HTTP)

- Built “to serve the internet”
- Request – Response
- The oldest protocol (1991)
- Based on TCP by default
- Version 3
  - Major improvements
  - TLS improvements
  - Use of UDP and QUIC

## Async Messaging Transports - Message Queuing Telemetry Transport (MQTT)

- Built specific for Internet of Things (IoT) in the gas and oil industry
- Lightweight publish subscribe messaging protocol / Small devices
- Small bandwidth
- Requires broker eg AWS IoT



AXXES_

## Async Messaging Transports – Microsoft Message Queuing (MSMQ)

- Message queue supported since Windows NT 4. ( Windows Only )
- Support for transactions (Distributed)
- Early “store and forward” principle ( similar to outbox )
- *Not recommended – discontinued*

AXXES_

EventBridge

NServicebus

Brighter

Amazon SQS

HTTP

Sagas

MQTT

RabbitMQ

QOS

Outbox

Amazon MQ

Amazon SNS

MSMQ

AMQP

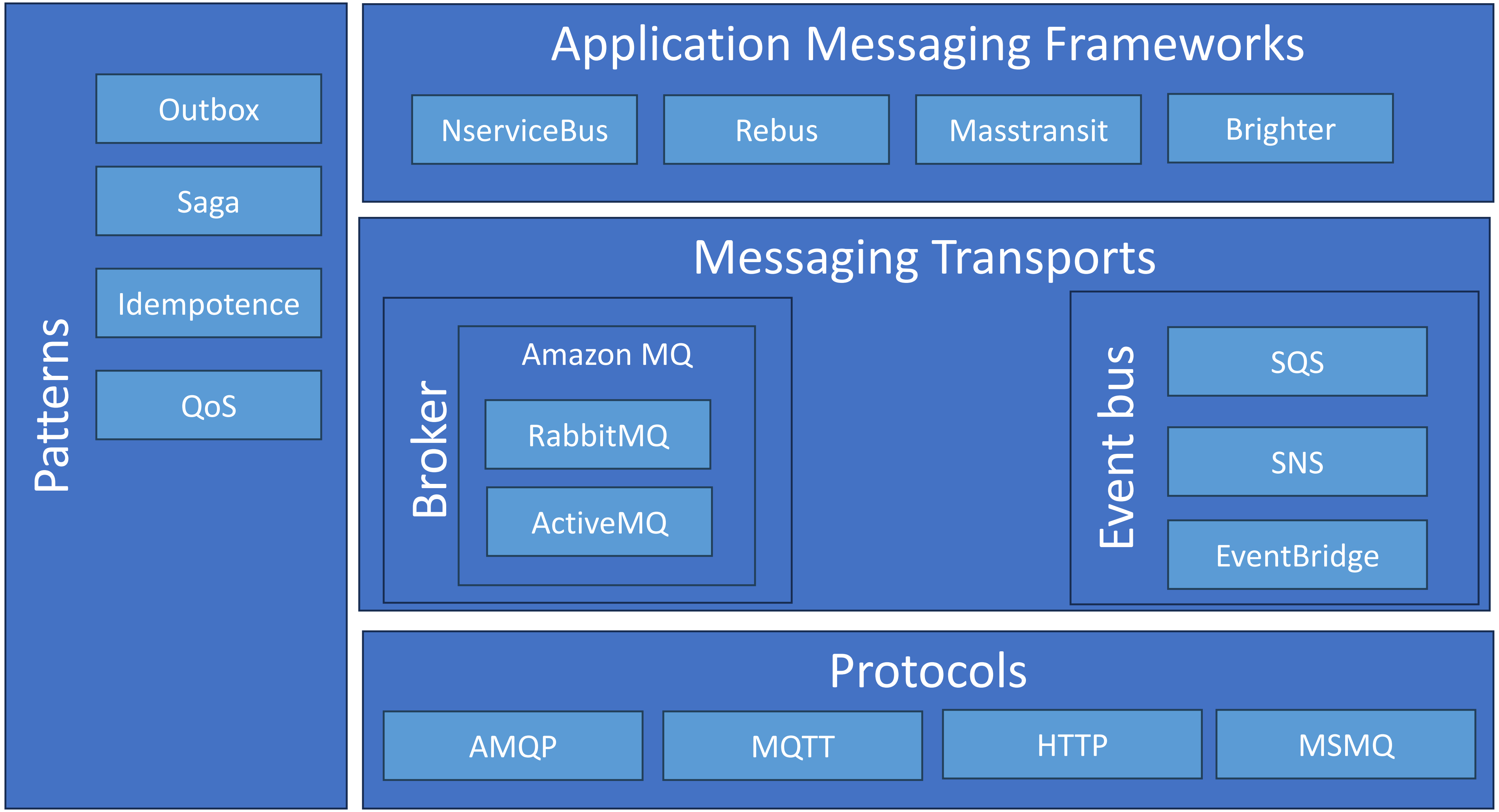
Rebus

ActiveMQ

MassTransit

Idempotence

## Summary


Patterns
Outbox
Saga
Idempotence
QoS
Application Messaging Frameworks
NserviceBus
Rebus
Masstransit
Brighter
Messaging Transports
Broker
Amazon MQ
RabbitMQ
ActiveMQ
Event bus
SQS
SNS
EventBridge
Protocols
AMQP
MQTT
HTTP
MSMQ

# What a modern mailman ought to know when delivering messages!

–

ROB VAN PAMEL

rob.vanpamel@gmail.com

@robvanpamel