Advanced NServiceBus Messaging and Configuration

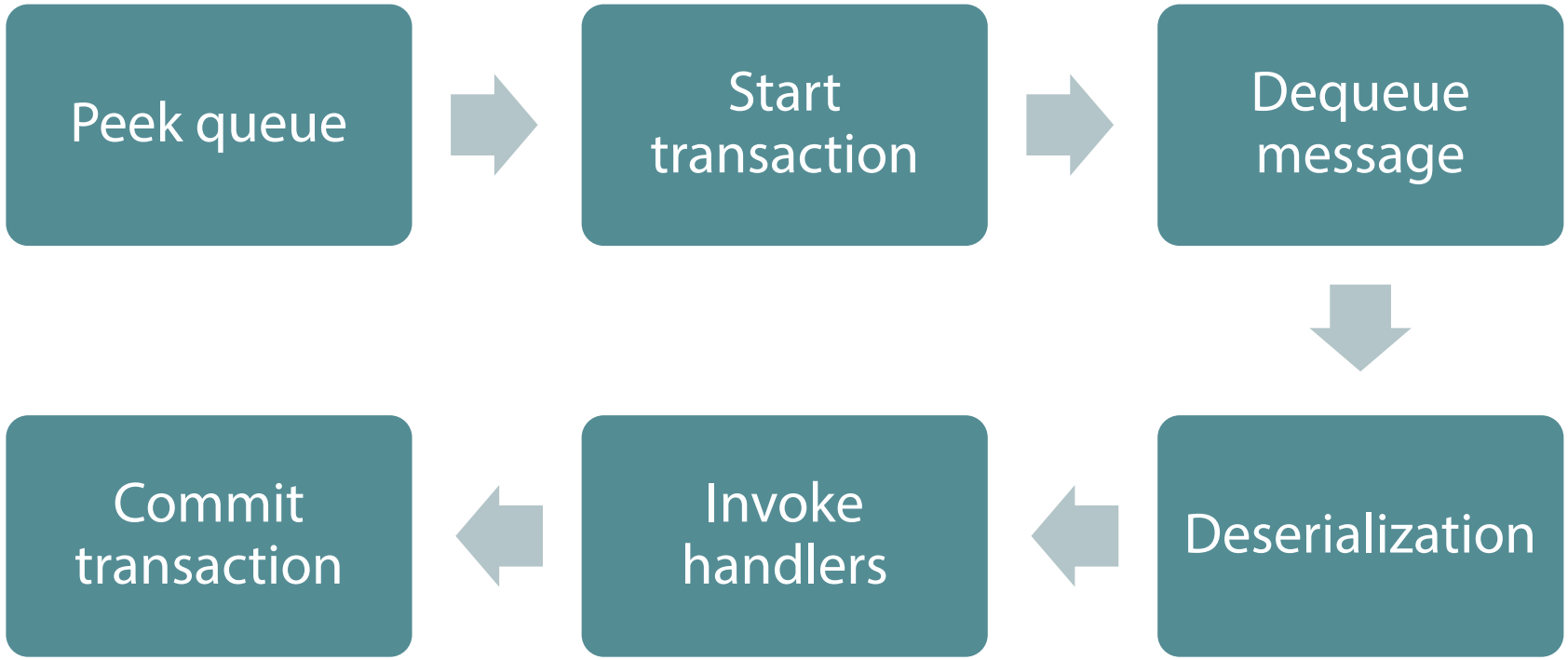
Roland Guijt

@rolandguijt | www.rmgsolutions.nl

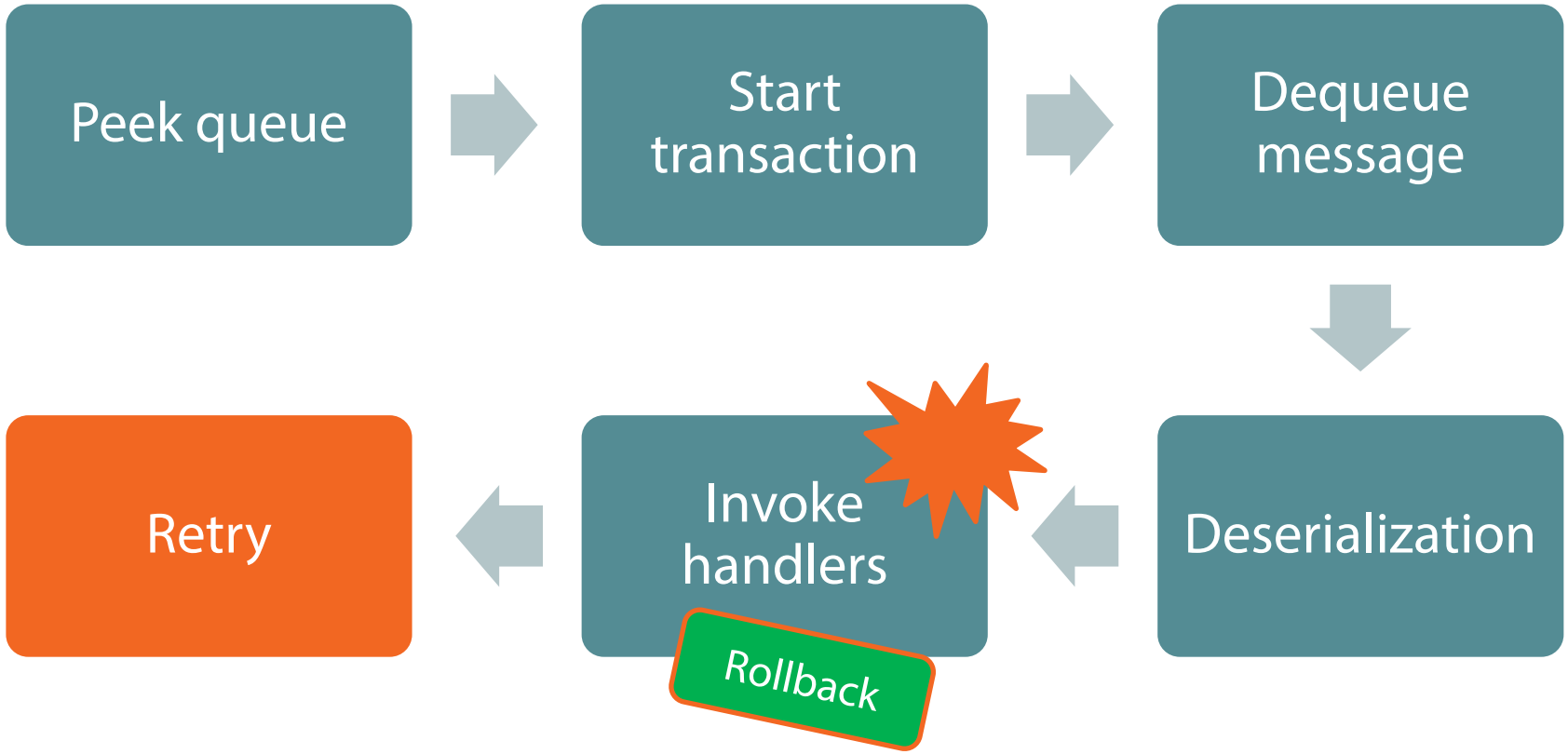
Overview

- Distributed transactions
- Additional features of messaging
- The message pipeline
- Monitoring messages
- Scaling services
- Unit testing handlers and sagas

Happy Path



Failure of Handler(s)



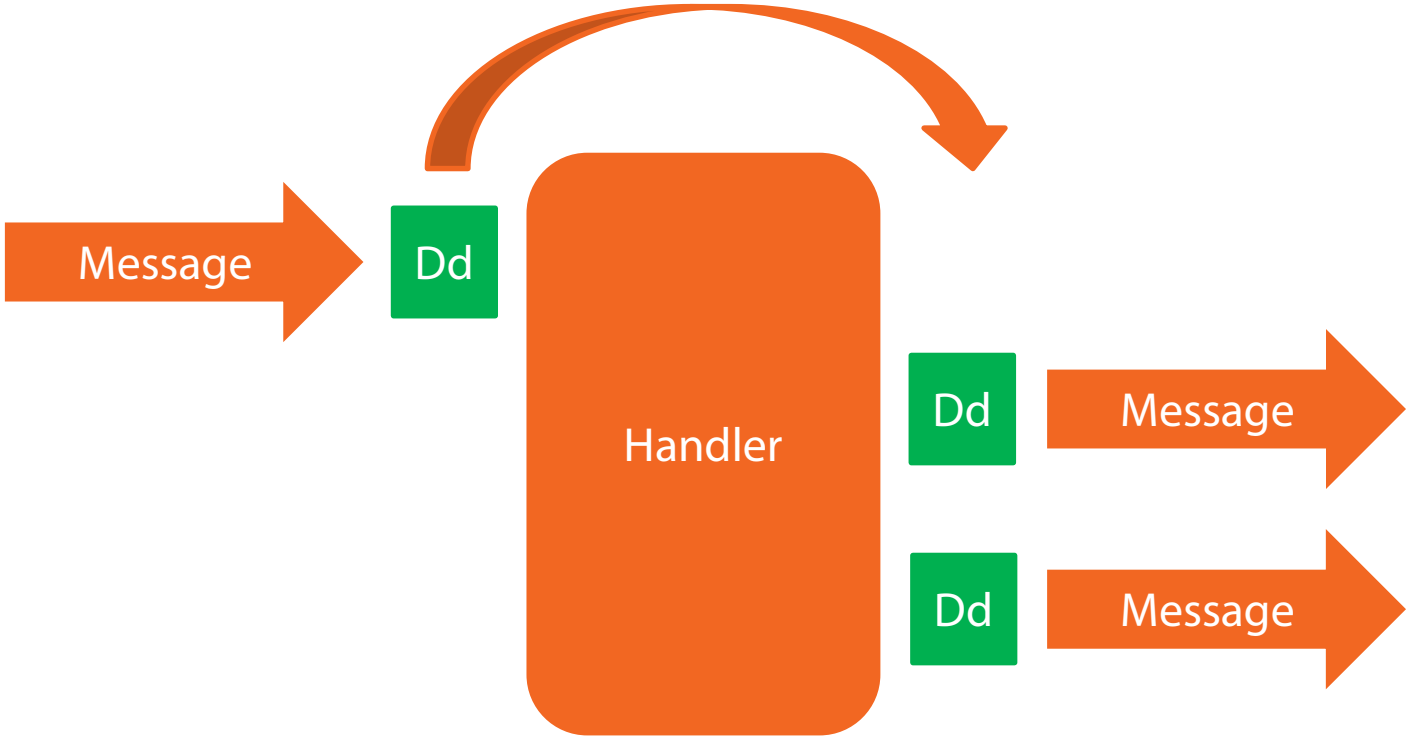
Distributed Transactions

- DTC (Distributed Transaction Coordinator)
- Default for MSMQ transport
- Configured by platform installer
- Resource managers
- Two-phase commit

Transports Not Supporting DTC

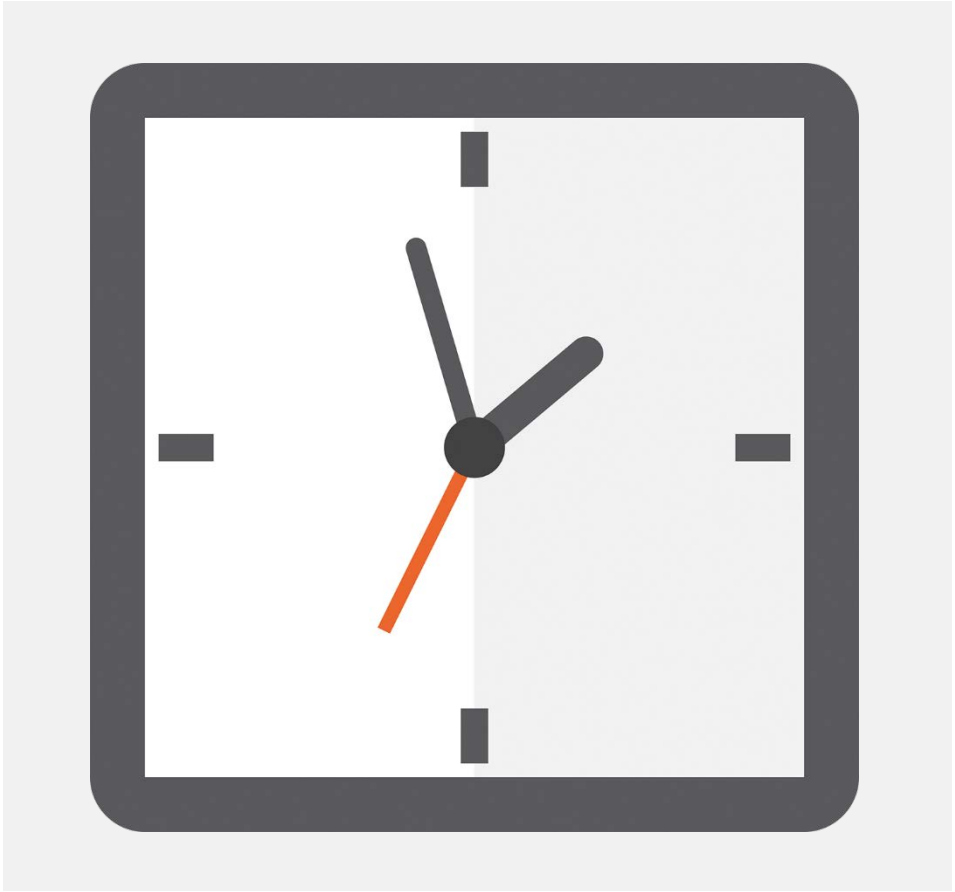
- Outbox
- Mimics DTC behavior
- Deduplication
- Uses data storage
- Kept for 7 days
- Enabled by default for RabbitMQ

Outbox



Message Expiration

- Message might be irrelevant or in the way after a certain amount of time
- Limit the lifespan of unhandled messages with the TimeToBeReceived attribute
- Error queue and audit queue contain handled messages



Message Expiration Example

```
[TimeToBeReceived("00:01:00")] // Discard after one minute
public class MyMessage: IMessage { }
```

Handler Order

Handlers are not run in any particular order by default

But you can specify an order

```
// >= v6
busConfiguration.ExecuteTheseHandlersFirst(typeof(HandlerB),
typeof(HandlerA), typeof(HandlerC));

// < v6
busConfiguration.LoadMessageHandlers(First<HandlerB>.
Then<HandlerA>().AndThen<HandlerC>());
```

```
IBus.DoNotContinueDispatchingCurrentMessageToHandlers();
```

Stopping Messages

Stop message in current handler and all remaining handlers

Message is considered successfully processed

Transaction is committed

```
IBus.HandleCurrentMessageLater();
IBus.Defer(TimeSpan delay, object message);
```

Deferring Messages

Put message back in the queue

Be careful for loops

Or specify a TimeSpan or DateTime

Transaction is committed

```
IBus.ForwardCurrentMessageTo(string destination);
```

Forwarding Messages

Send message to another queue

Doesn’t stop handler execution

Property Encryption

- When a property in a message contains sensitive data
- WireEncryptedString
- Rijndael algorithm is default
- Use shared configuration
- IEncryptionService for custom encryption

Configuration of Property Encryption

```
public class ProvideConfiguration :
    IProvideConfiguration<RijndaelEncryptionServiceConfig>
{
    public RijndaelEncryptionServiceConfig GetConfiguration()
    {
        return new RijndaelEncryptionServiceConfig { Key =
            "gdDbqRpQdRbTs3mhdZh9qCaDaxJXl+e6", ExpiredKeys =
                new RijndaelExpiredKeyCollection { new
                    RijndaelExpiredKey { Key =
                "abDbqRpQdRbTs3mhdZh9qCaDaxJXl+e6" }
                }
            };
} }
```

DataBus: Supporting Large Messages

Size of a message is limited depending on transport

A large message could cause performance problems

Store properties large in size separately

busConfiguration.UseDataBus

FileShareDataBus or AzureDataBus

IDataBus for custom DataBus storage

A DataBus Property

```
[TimeToBeReceived("1.00:00:00")]
public class MessageWithLargePayload: IMessage
{
    public string SomeProperty { get; set; }
    public DataBusProperty<byte[]> LargeBlob { get; set; }
}
```

Conventions and Unobtrusive Mode

- IMessage, ICommand, IEvent
- Require reference to NServiceBus assembly
- Real POCO
- Define conventions
- TimeToBeReceived, DataBus and Encryption also supported

Defining Conventions

```
BusConfiguration config = new BusConfiguration();
var conventions = config.Conventions();

conventions.DefiningCommandsAs(t => t.Namespace ==
"MyNamespace" && t.Namespace.EndsWith("Commands"));

conventions.DefiningTimeToBeReceivedAs(t =>
t.Name.EndsWith("Expires") ? TimeSpan.FromSeconds(30)
: TimeSpan.MaxValue);
```

Auditing Messages

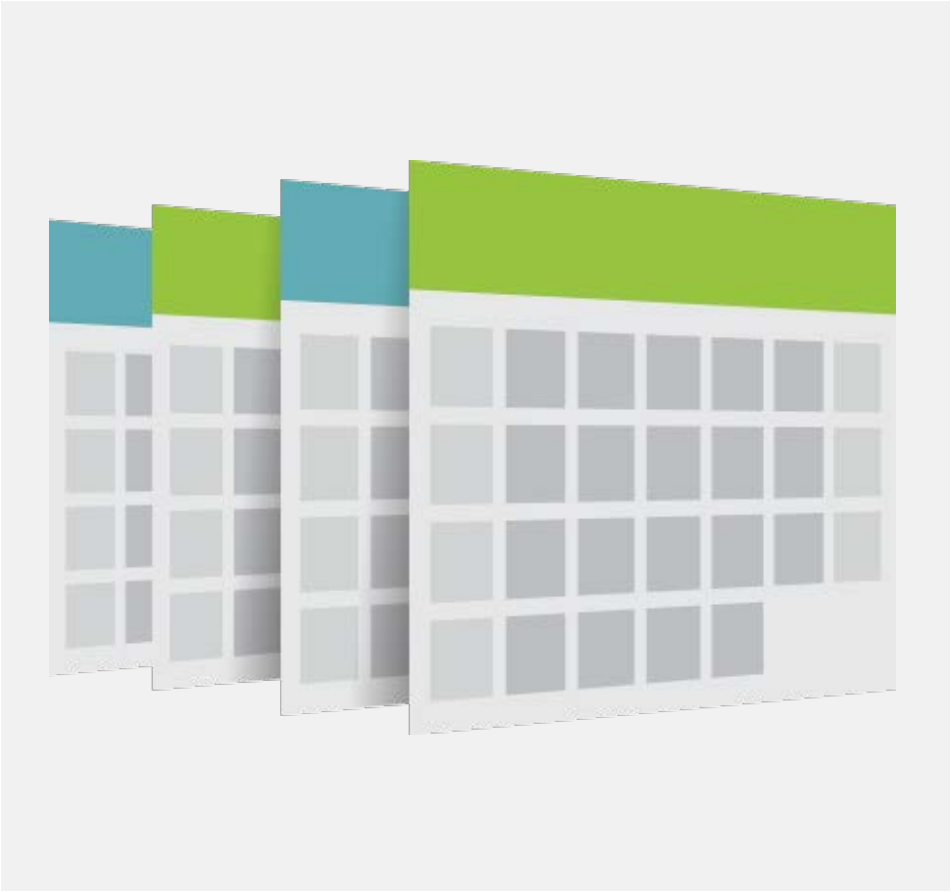
- Debugging of services more difficult
- Copy of all messages to audit queue
- One queue for all endpoints
- Particular platform tooling

Configuring Auditing

```
<configuration>
<AuditConfig QueueName="auditqueue@adminmachine"
                OverrideTimeToBeReceived="00:10:00"/>
</configuration>
```

Scheduling

- Execute a task every given amount of time
- In memory
- Timeout Manager



Scheduling a Task

```
schedule.Every(TimeSpan.FromMinutes(5), () =>
bus.Send(new MyMessage()));
```

```
schedule.Every(TimeSpan.FromMinutes(5),
"MyCustomTask", SomeCustomMethod);
```

Polymorphic Message Dispatch

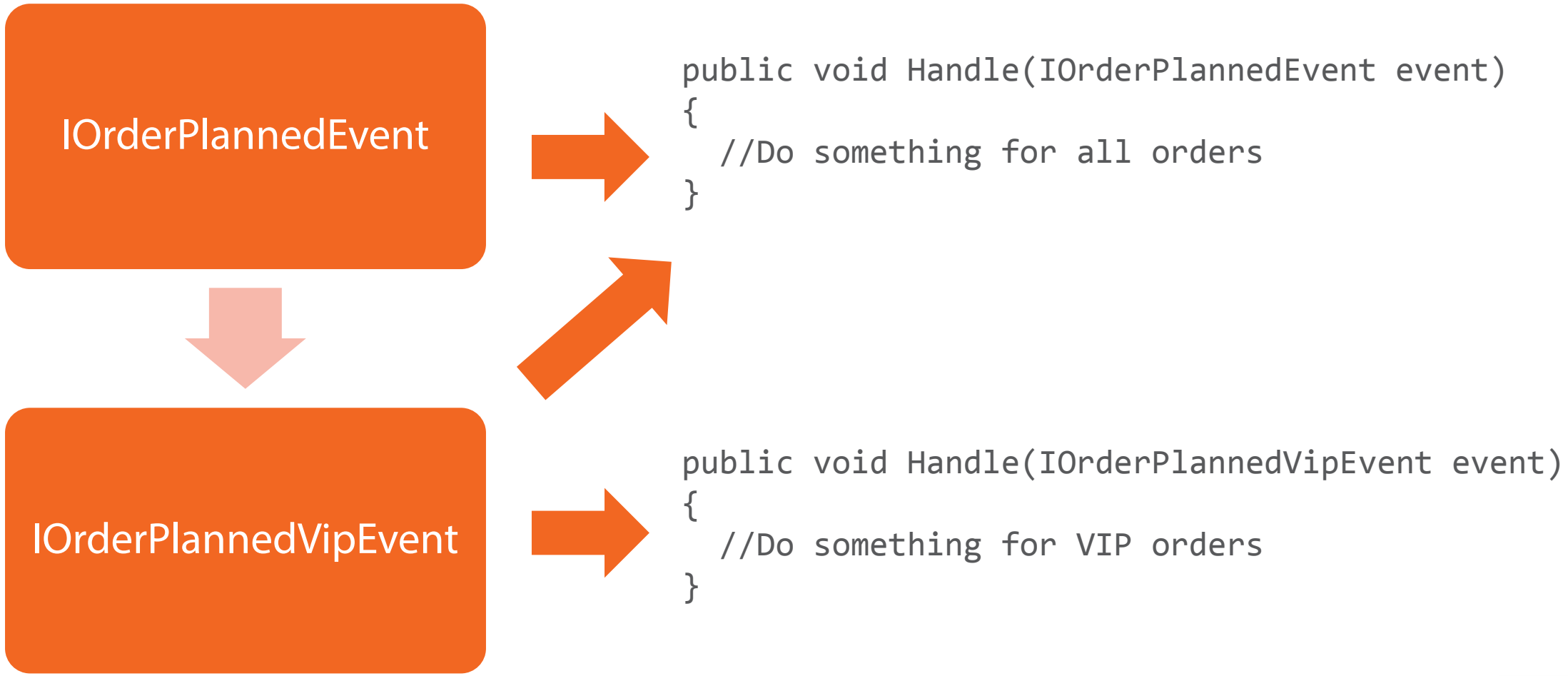
```
namespace v1
{
    public interface IOrderPlannedEvent: IEvent
    {
        int Weight { get; set; }
        string Address { get; set; }
    }
}
public interface IOrderPlannedEvent: v1.IOrderPlannedEvent
{
    DateTime OrderPlacedAt { get; set; }
}
```

Great For:

Versioning

Polymorphic event handling

Polymorphic Event Handling



Headers

Secondary information contained in the message

Similar to HTTP headers

Should contain metadata only

Can be written and read in behaviors, mutators and handlers

Examples of Headers

Message Interaction Headers

- NServiceBus.MessageId
- NServiceBus.CorrelationId
- NServiceBus.MessageIntent
- NServiceBus.ReplyToAddress

Audit Headers

- NServiceBus.ProcessingStarted
- NServiceBus.ProcessingEnded
- NServiceBus.ProcessingEndpoint
- NServiceBus.ProcessingMachine

```
public void Handle(MyMessage message)
{
    IDictionary<string, string> headers =
        bus.CurrentMessageContext.Headers;

    string nsbVersion = headers[Headers.NServiceBusVersion];
    string customHeader = headers["MyCustomHeader"];
}
```

Reading Headers

To get the dictionary in a mutator: use transportMessage.Headers (MutateIncoming)

In a behavior: context.GetPhysicalMessage().Headers (Invoke)

```
public void Handle(MyMessage message)
{
  SomeOtherMessage someOtherMessage = new SomeOtherMessage();
  bus.SetMessageHeader(someOtherMessage, "MyCustomHeader",
    "My custom value");
  bus.Send(someOtherMessage);
}
```

Writing Headers for Version < 6

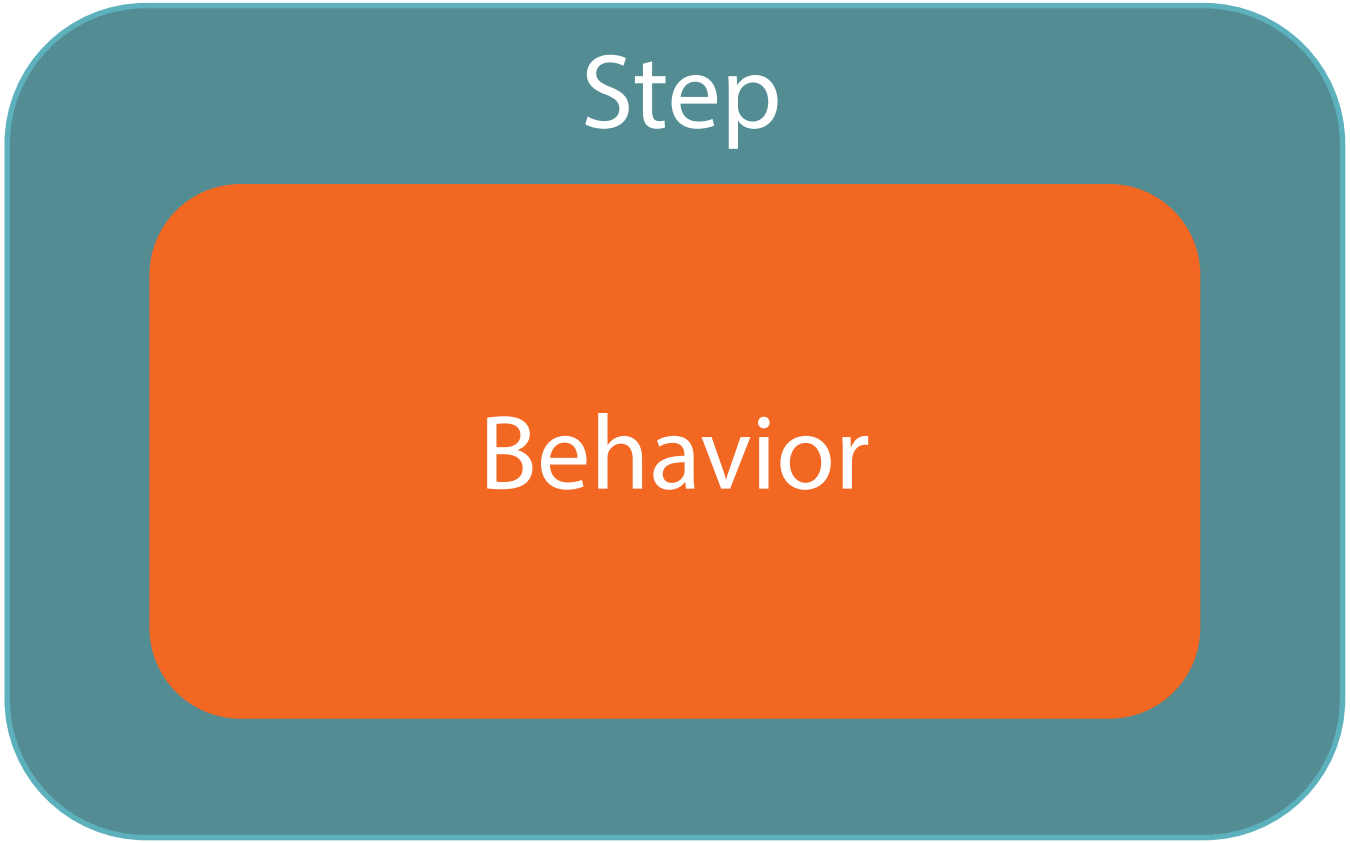
For behaviors and mutators, just write to the dictionary

```
public void Handle(MyMessage message)
{
    SendOptions sendOptions = new SendOptions();
    sendOptions.SetHeader("MyCustomHeader",
       "My custom value");
    bus.Send(new SomeOtherMessage(), sendOptions);
}
```

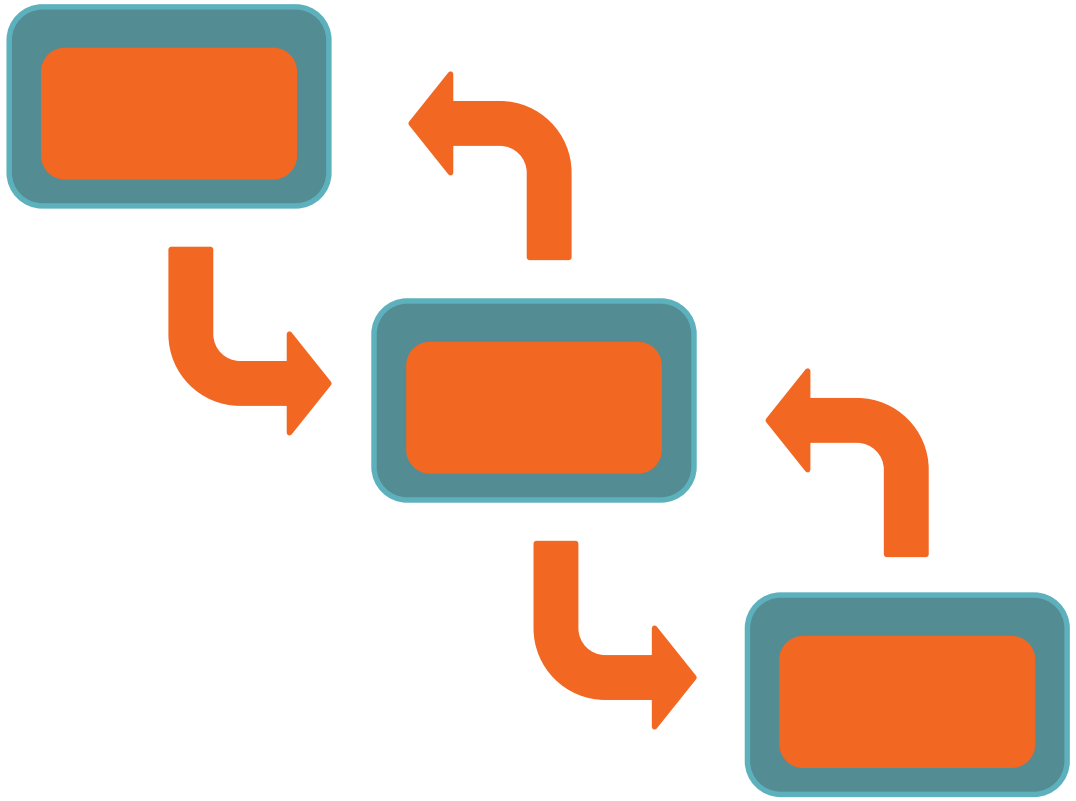
Writing Headers for Version >= 6

For behaviors and mutators, use context.SetHeader (Invoke and MutateOutgoing)

Message Pipeline



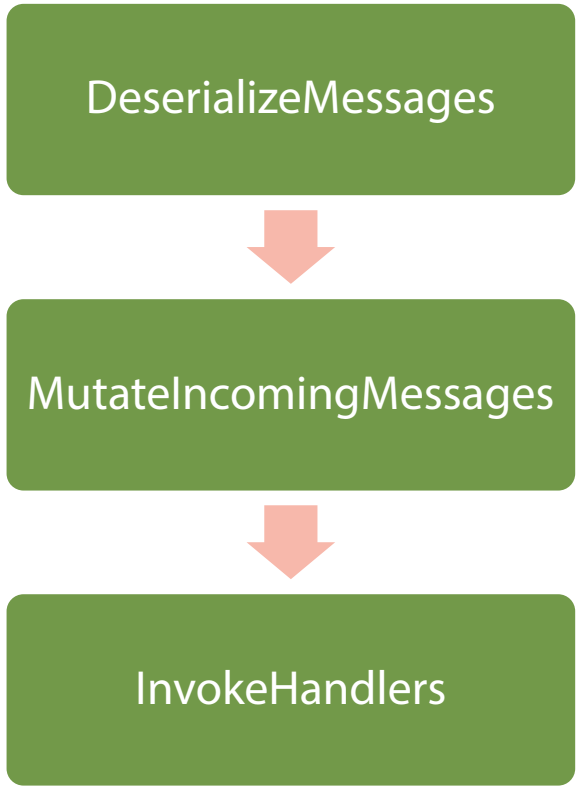
Message Pipeline



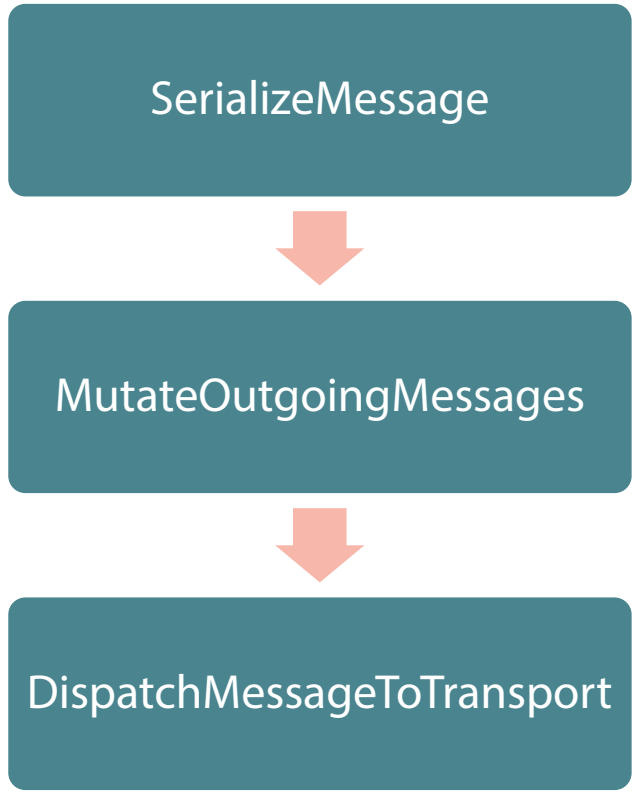
```
public void Invoke(context, next)
{
  //Do something before other behaviors
  next();
  //Do something after other behaviors
}
```

Default Pipeline

Incoming



Outgoing



Implementing New Behavior

```
public class SampleBehavior : IBehavior<IncomingContext>
{
    public void Invoke(IncomingContext context, Action next)
    {
      using(var dataContext = new DataContext())
      {
        context.Set(dataContext);
        next();
      }
    }
}
```

Creating a New Step

```
class NewStepInPipeline : RegisterStep
{
  public NewStepInPipeline(): base("NewStepInPipelineId",
    typeof(SampleBehavior), “Description")
  {
    InsertBefore(WellKnownStep.InvokeHandlers);
  }
}
```

Registering the Step

```
class NewStepInPipelineRegistration : INeedInitialization
{
  public void Customize(BusConfiguration busConfiguration)
  {
      busConfiguration.Pipeline
         .Register<NewStepInPipeline>();
  }
}
```

Replacing the Behavior of a Built-in Step

```
busConfiguration.Pipeline.Replace(“existing step id",
    typeof(SampleBehavior), "Description");
```

Message Mutators

Applicative

- Message as an object

- IMutateIncomingMessages
- IMutateOutgoingMessages
- IMessageMutator

Transport

- Raw bytes of the message

- IMutateIncomingTransportMessages
- IMutateOutgoingTransportMessages
- IMutateTransportMessages

Registering a Message Mutator

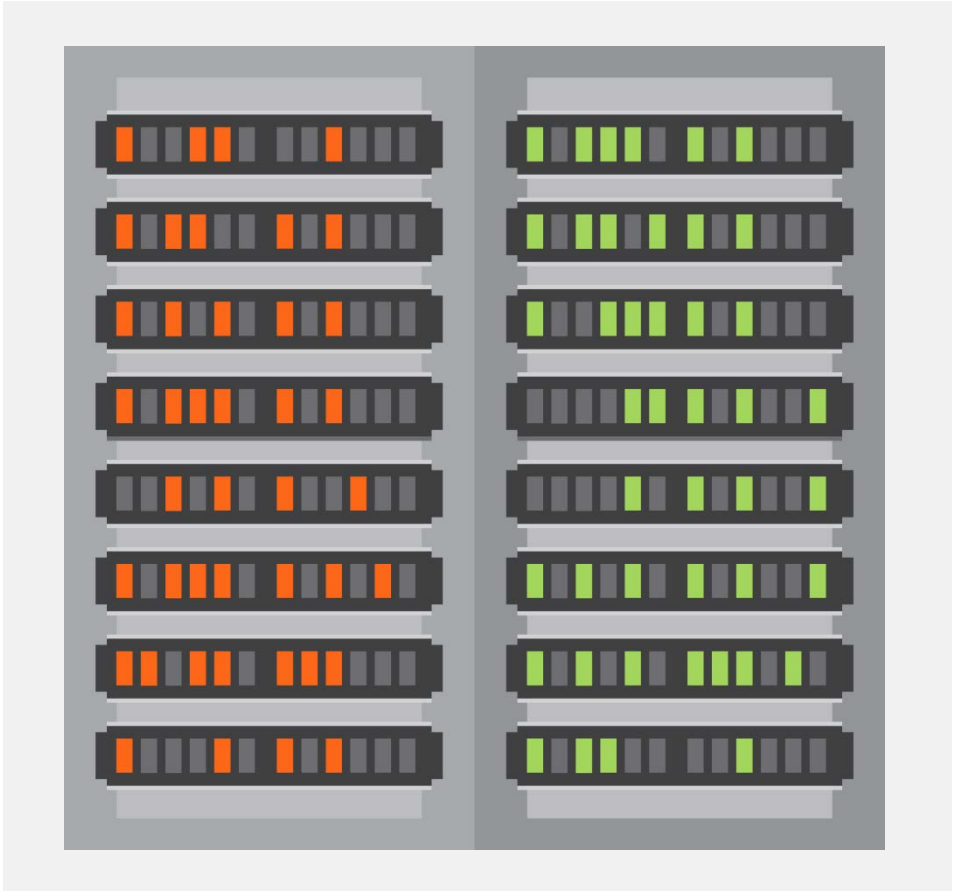
```
busConfiguration.RegisterComponents(components =>
{
    components.ConfigureComponent<ValidationMessageMutator>
        (DependencyLifecycle.InstancePerCall);
});
```

Unit of Work

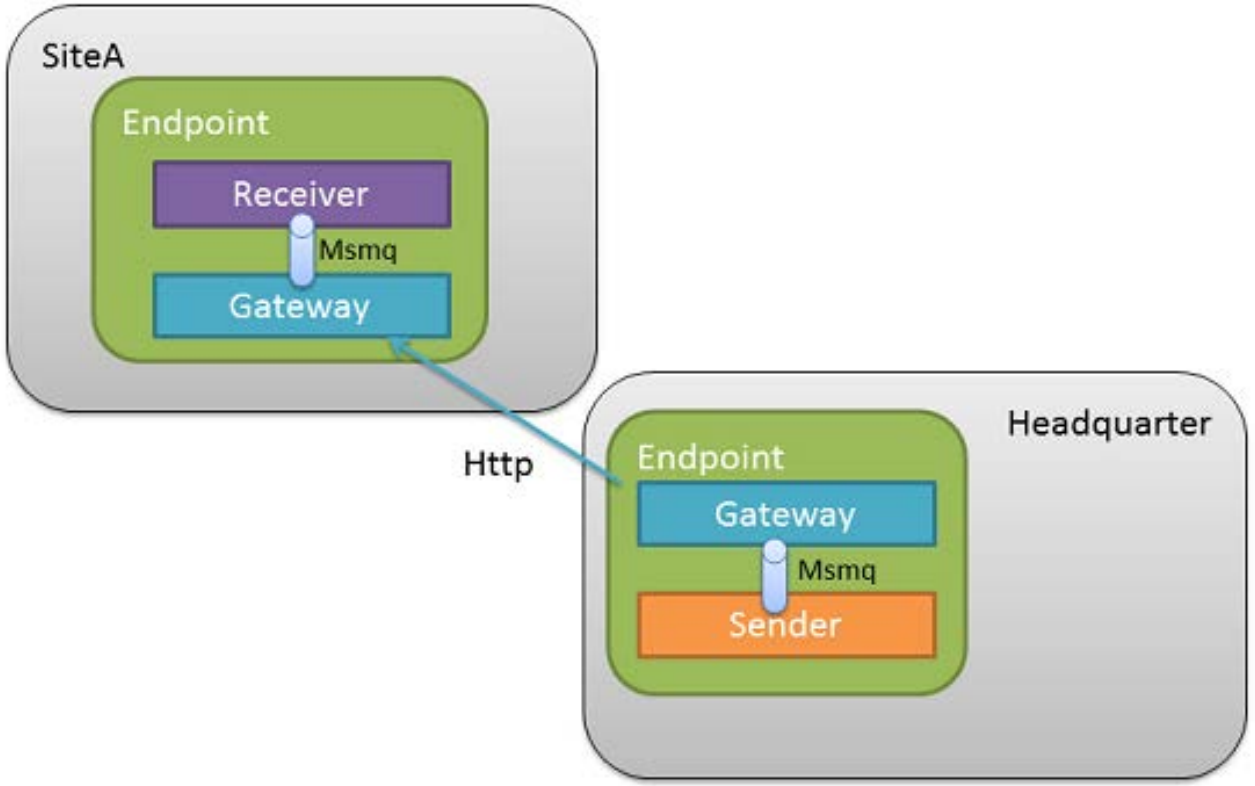
```
public class MyUnitOfWork : IManageUnitsOfWork
{
    public void Begin()
    {
    }
    public void End(System.Exception ex = null)
    {
    }
}
```

Messaging Across Multiple Sites

- VPN sometimes not an option
- Gateway
- Not for replication
- Messages have to be explicitly sent to the gateway
- Events not supported
- HTTP or custom channel



Gateway



Configuring Sites

```
<GatewayConfig>
    <Sites>
      <Site Key="SiteA"
            Address="http://SiteA.mycorp.com/"
            ChannelType="Http"/>
    </Sites>
</GatewayConfig>
```

Enabling Gateway in the Endpoint

```
busConfiguration.EnableFeature<Gateway>();
```

Sending a Gateway Message

```
Bus.SendToSites(new[] { "SiteA", "SiteB" },
    new MyMessage());
```

Gateway Features

Automatic retries

De-duplication

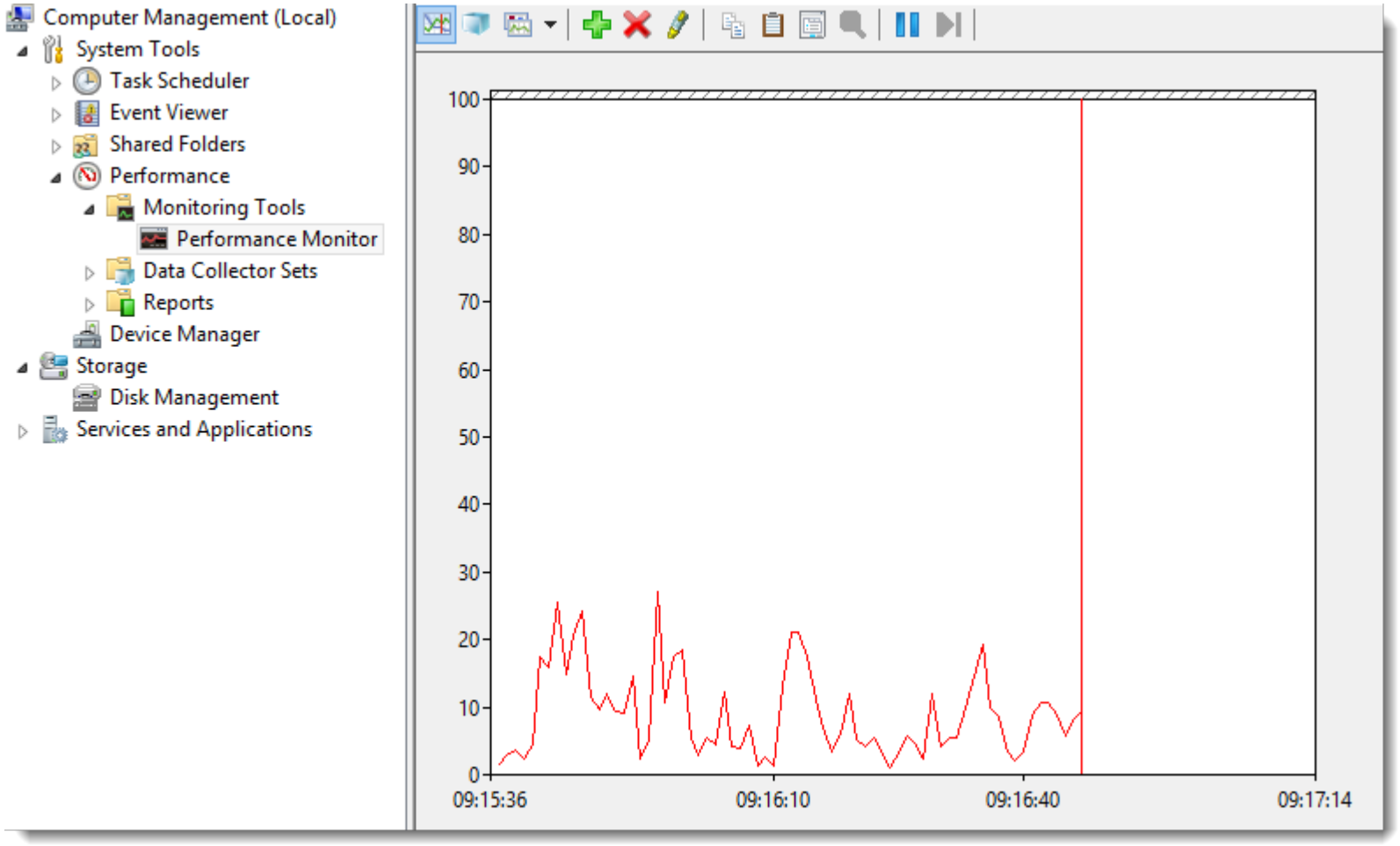
SSL

Data bus

Multiple channels

Extensible

Performance Counters

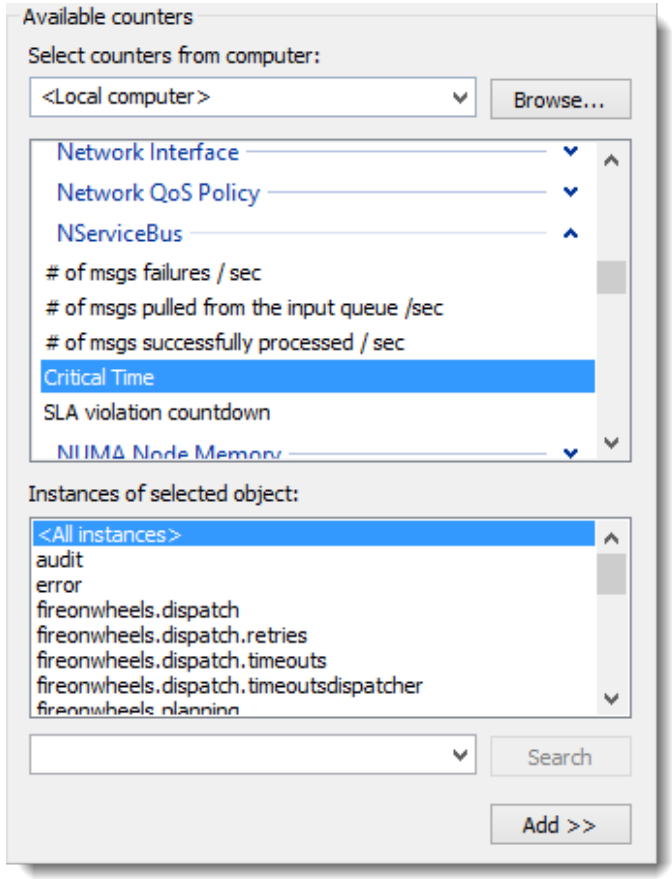


Performance Counters for Messaging

MSMQ has many

NServiceBus' counters focus on performance

Installed when running setup



NServiceBus Performance Counters

Successful message processing rate

Queue message receive rate

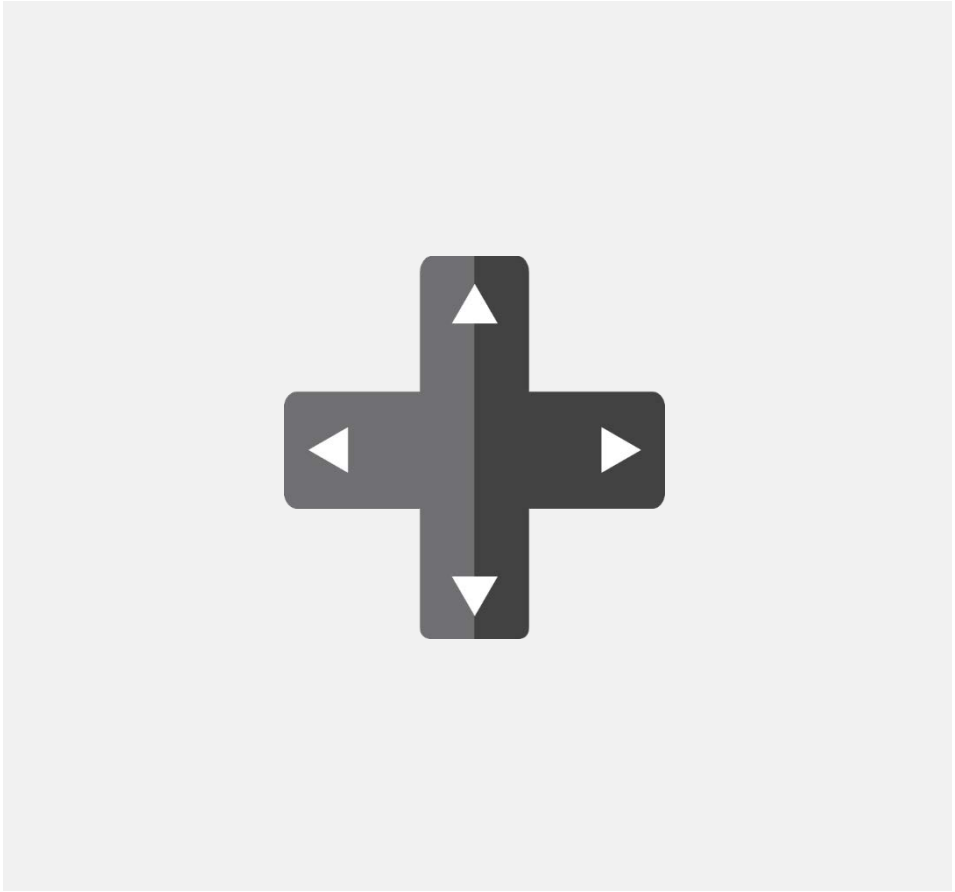
Failed message processing rate

Critical time

SLA violation countdown

Scaling Services

- Messages take too long to process
- Scale up
- Scale out



How to Scale Out?

Broker style transport

- Deploy extra instance of service

MSMQ transport

- Distributor feature
- Workers
- Master
- Performance
- Single point of failure

Unit Testing

- Common practice
- Hard without help
- NServiceBus.Testing NuGet package
- For handlers and sagas
- Works with every test framework

Demo: Unit Testing

Multiple developers

Many changes

Handlers and saga should be unit tested

Summary

- Different transactions mechanisms
- Many message features
- Adjustable pipeline
- Monitoring in a standard way
- Scaling out support
- Unit testing is easy