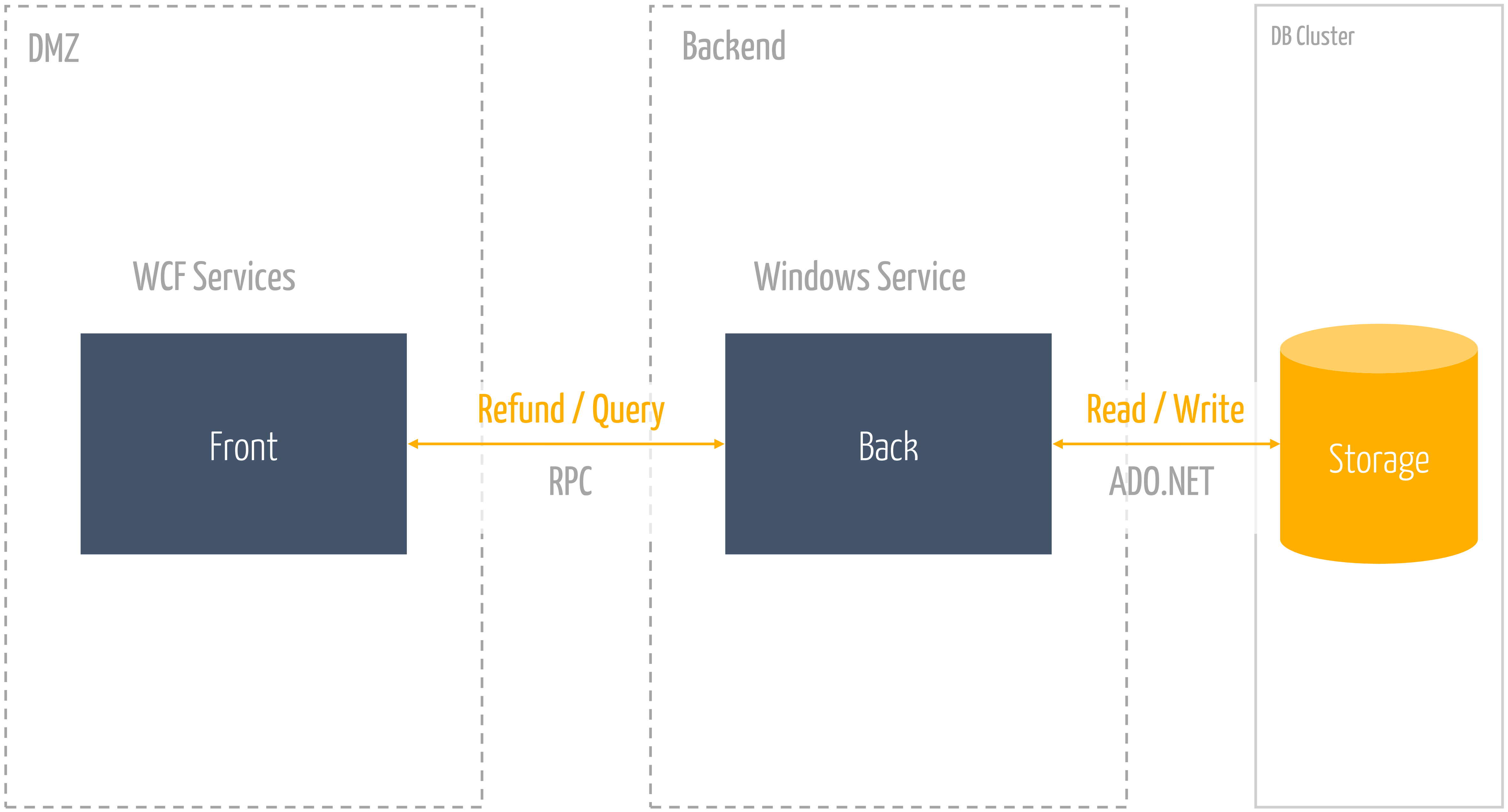
Particular Software

Do it yourself. A message pump

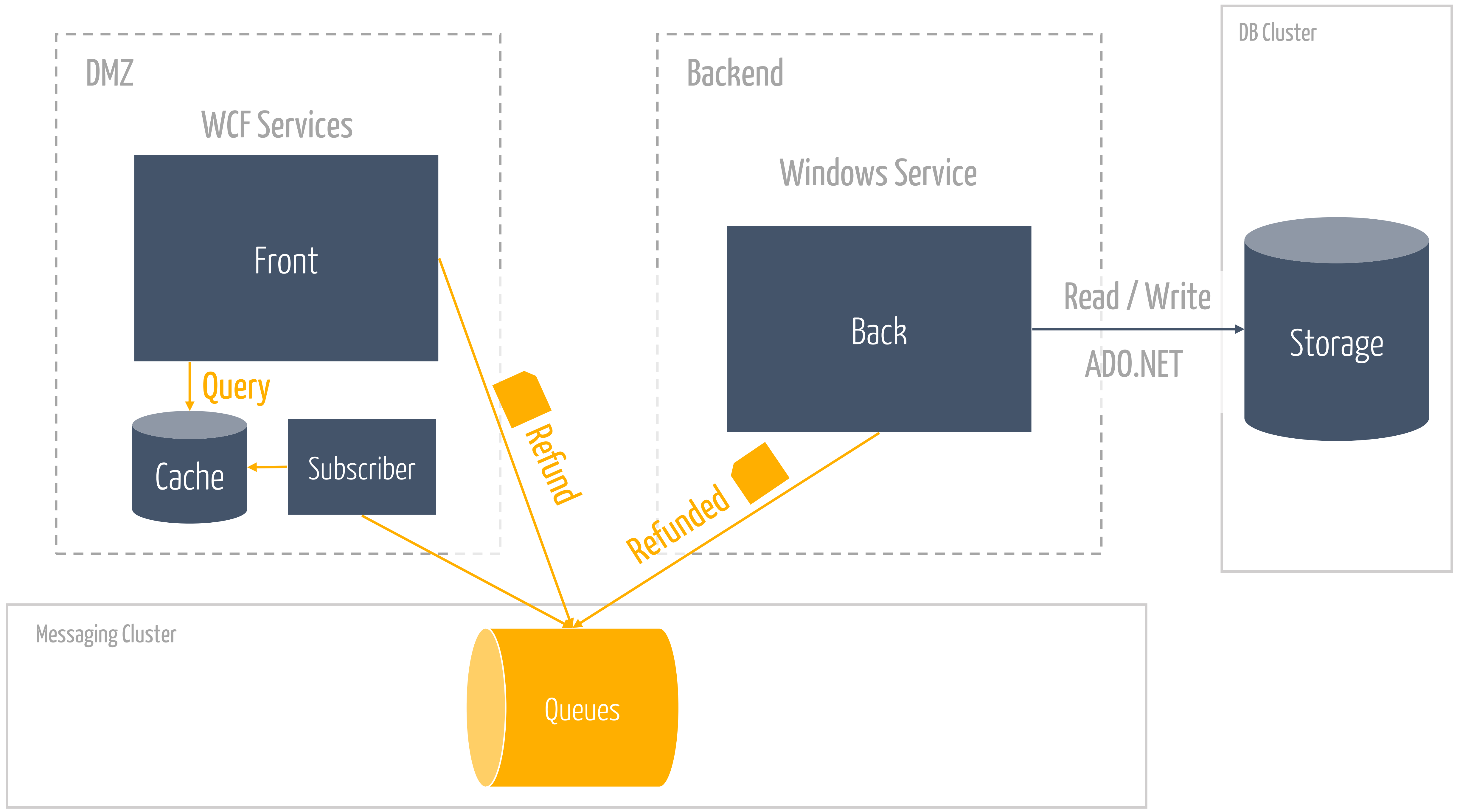
that kicks ass


DMZ
Backend
DB Cluster
WCF Services
Windows Service
Front
Refund / Query
RPC
Back
Read / Write
ADO.NET
Storage

The RPC callstack of doom

praise the lords of

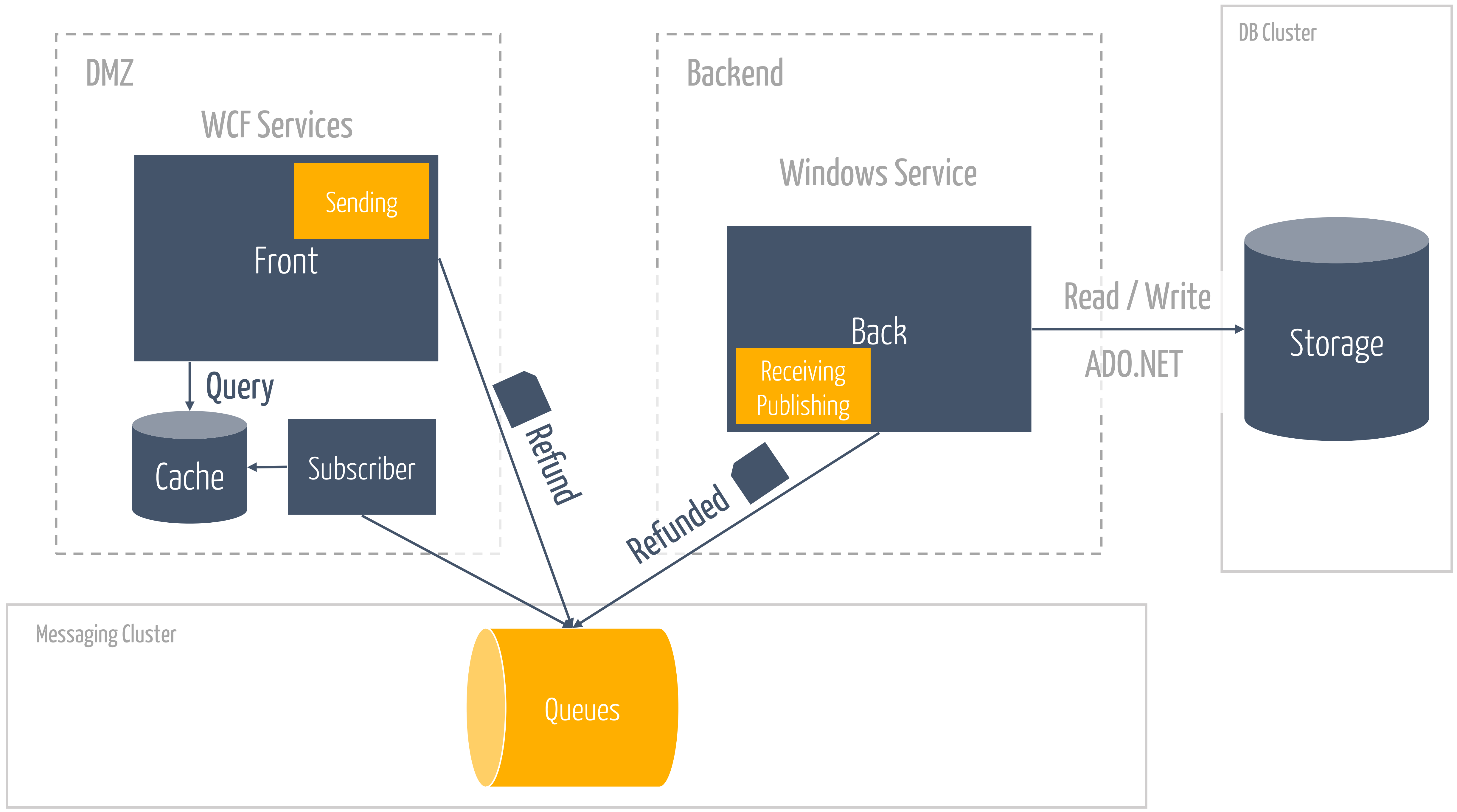
messaging


DMZ
WCF Services
Front
Query
Cache
Subscriber
Refund
Backend
Windows Service
Back
Refunded
Read / Write
ADO.NET
DB Cluster
Storage
Messaging Cluster
Queues

Build

or

Buy


DMZ
WCF Services
Sending
Front
Query
Cache
Subscriber
Refund
Backend
Windows Service
Back
Receiving
Publishing
Refunded
Read / Write
ADO.NET
DB Cluster
Storage
Messaging Cluster
Queues

Building the

pump

```
var (payload, headers) = Serialize(message);
await queue.Send(payload, headers);
```

Easy!

TPL handwaving

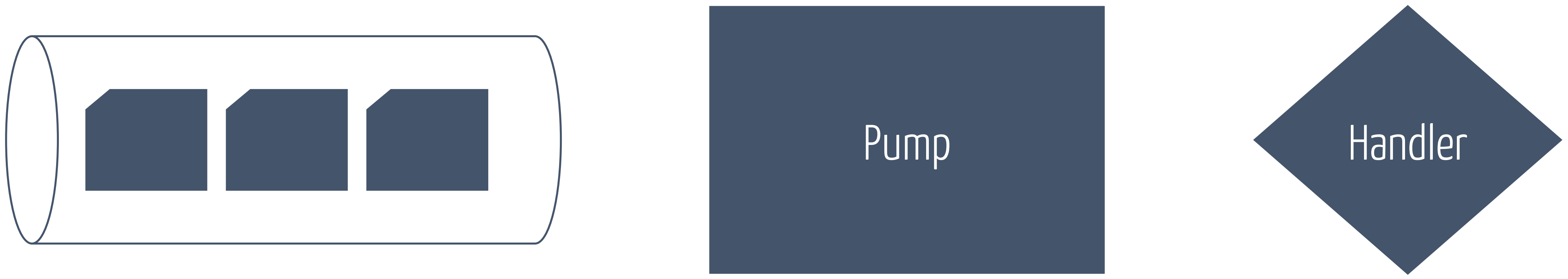
Cooporative cancellation 101

Async / Await

Ship it!

It worked until

Rush hour


Pump
Handler

Let's throw in some

concurrency

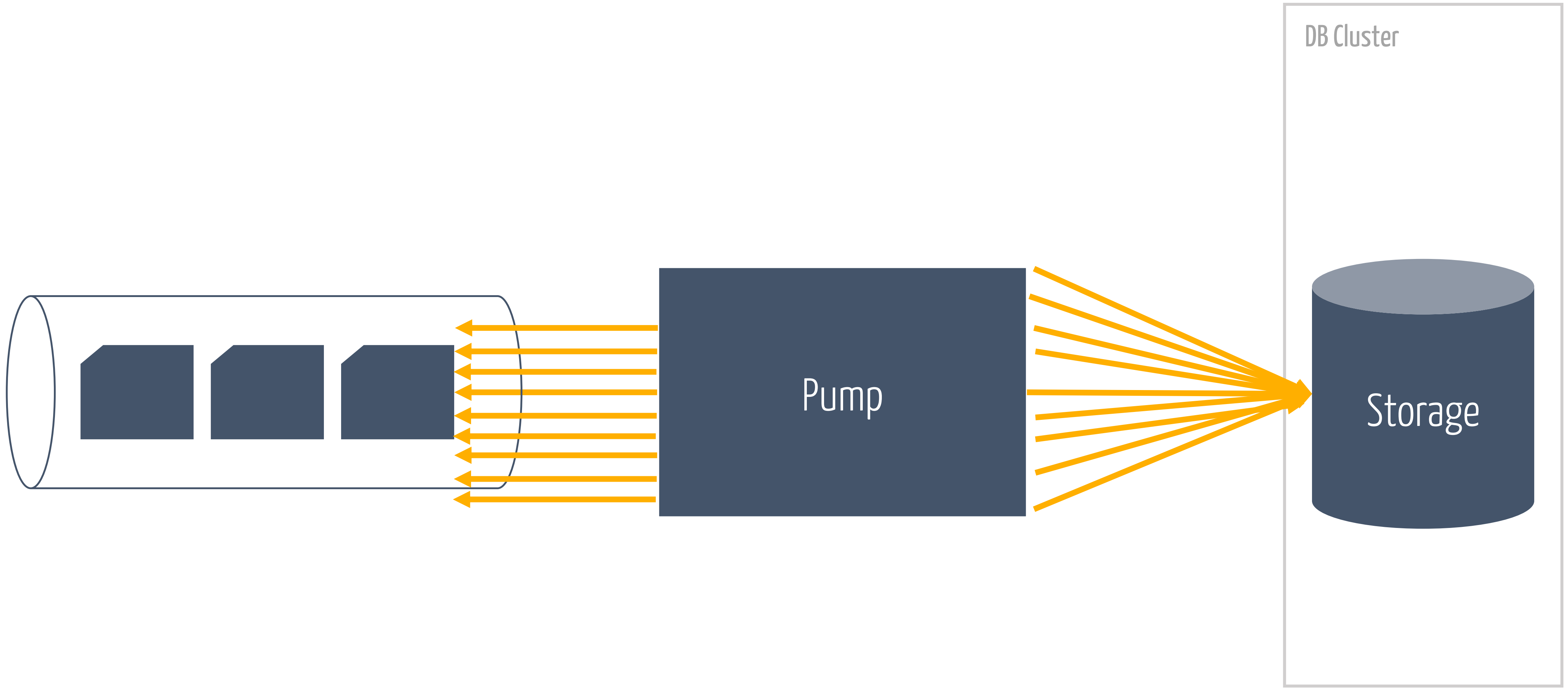
Just a tiny change...

Introduce Fire & Forget

Ship it!

It worked until

Rush hour


Pump
DB Cluster
Storage

Better limit

concurrency

Semaphore controls floodgate

Ship it!

It worked

As time passes

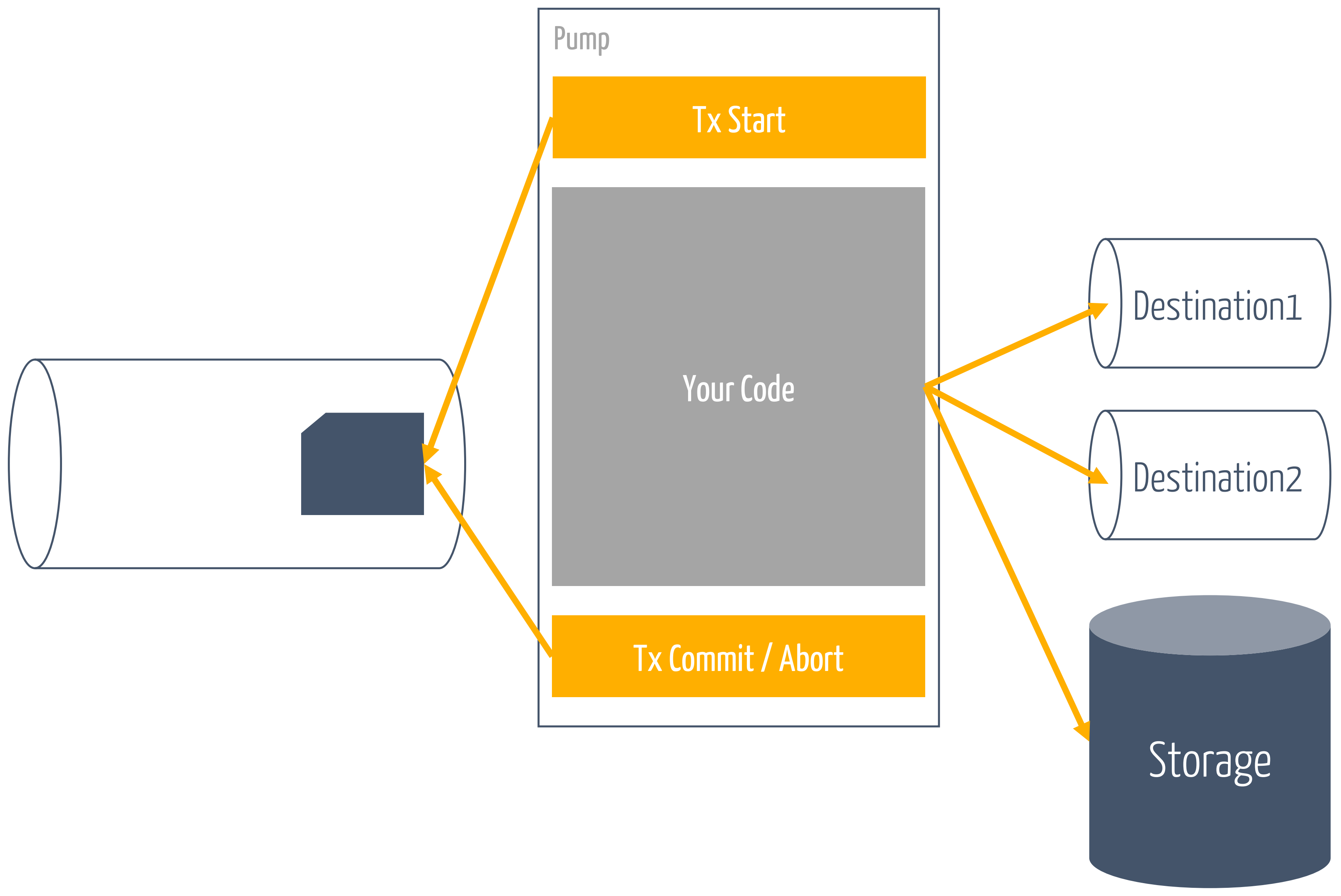
Make it

flextensible

PARSONS
FRUIT STAND
KEREMEOS B.C.

Life beyond

transactions


Pump
Tx Start
Your Code
Tx Commit / Abort
Destination1
Destination2
Storage

Cloudy with a chance of

Ⅎailurə

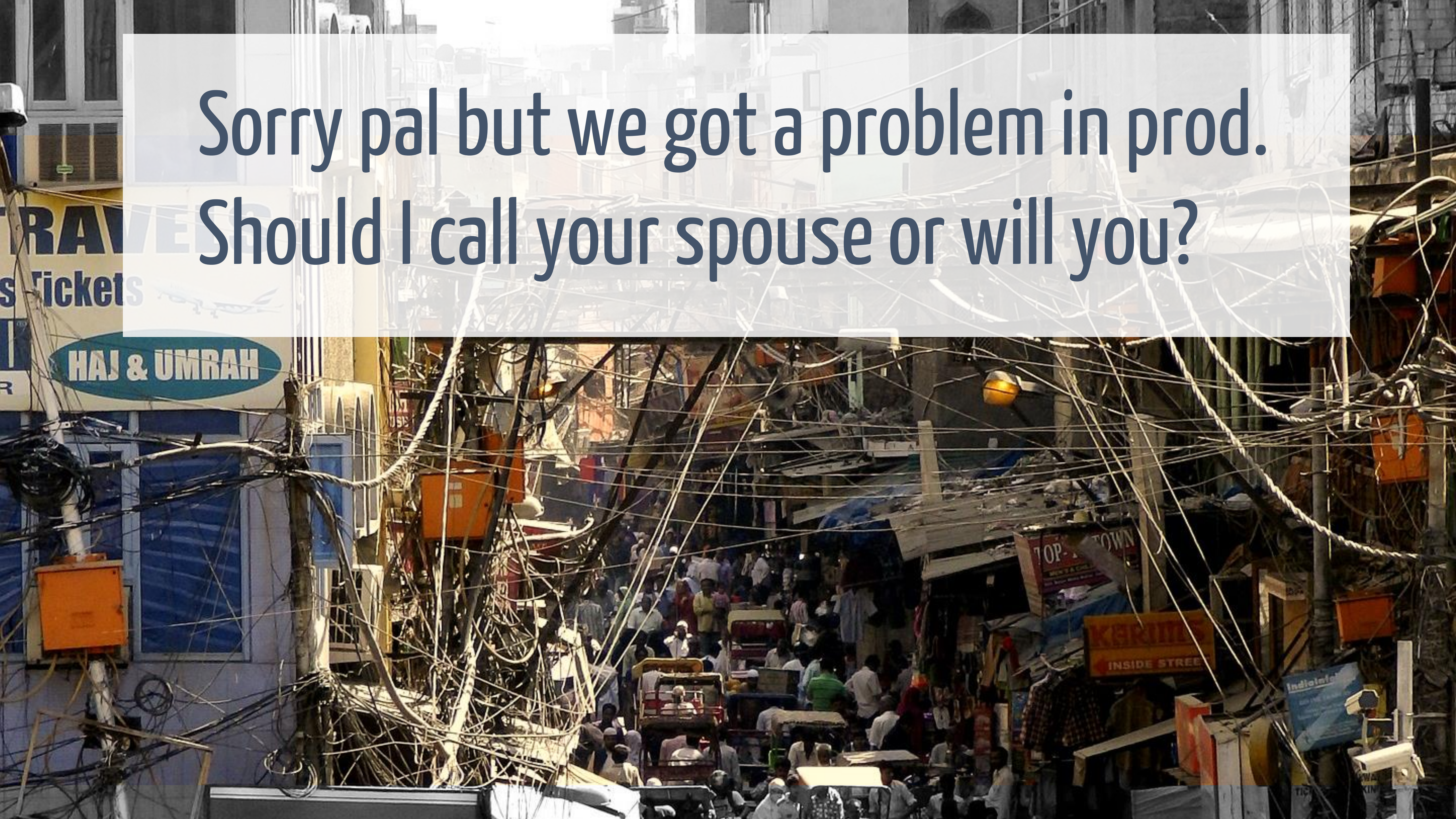
Penny Pinching

Now we have the

basic bits

Placeholder

Sorry pal but we got a problem in prod.
Should I call your spouse or will you?



I want

insights

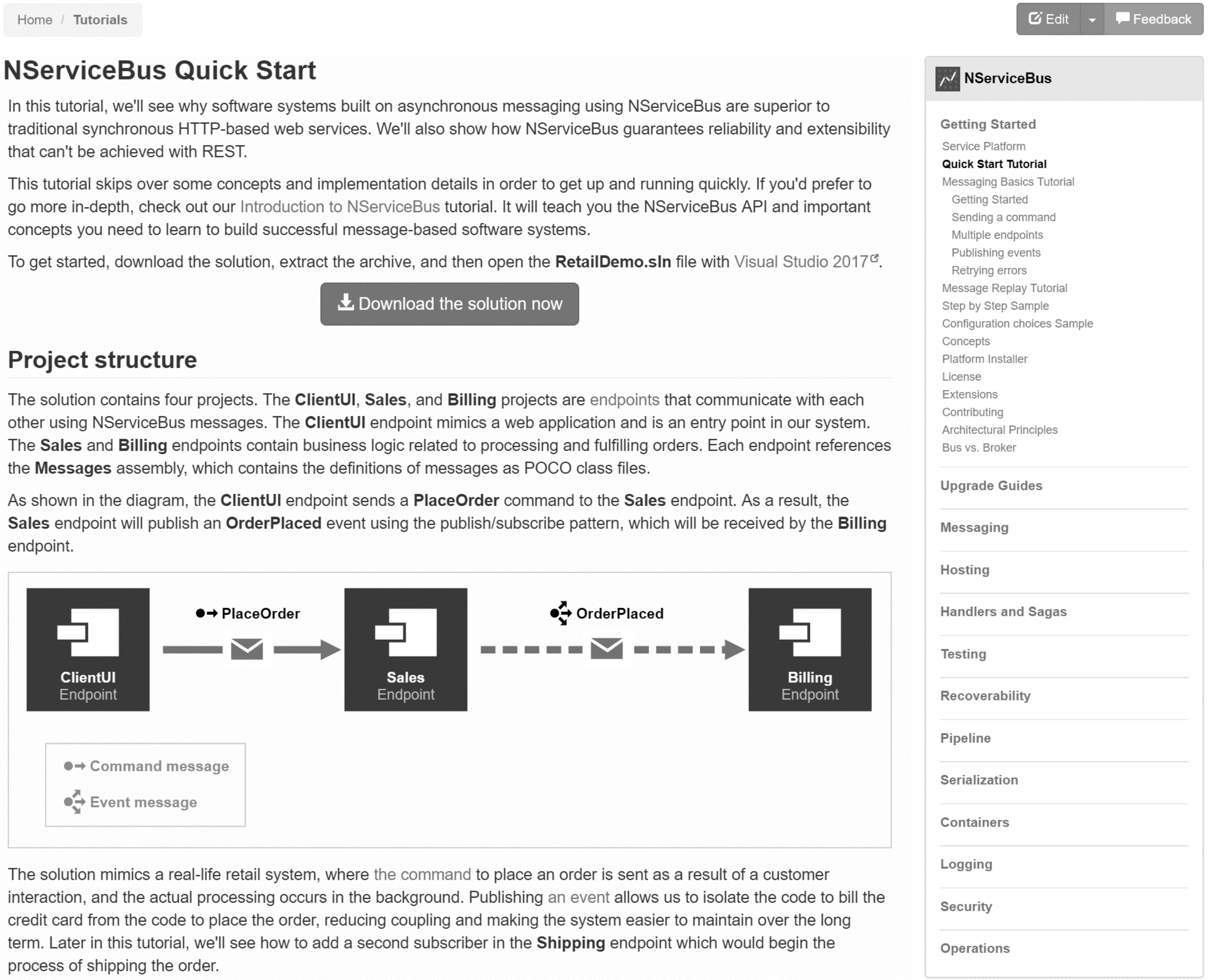
Placeholder

Goals

Premise

Recap

docs.particular.net/
tutorials/quickstart



Edit ▾ Feedback

NServiceBus Quick Start

In this tutorial, we'll see why software systems built on asynchronous messaging using NServiceBus are superior to traditional synchronous HTTP-based web services. We'll also show how NServiceBus guarantees reliability and extensibility that can't be achieved with REST.

This tutorial skips over some concepts and implementation details in order to get up and running quickly. If you'd prefer to go more in-depth, check out our Introduction to NServiceBus tutorial. It will teach you the NServiceBus API and important concepts you need to learn to build successful message-based software systems.

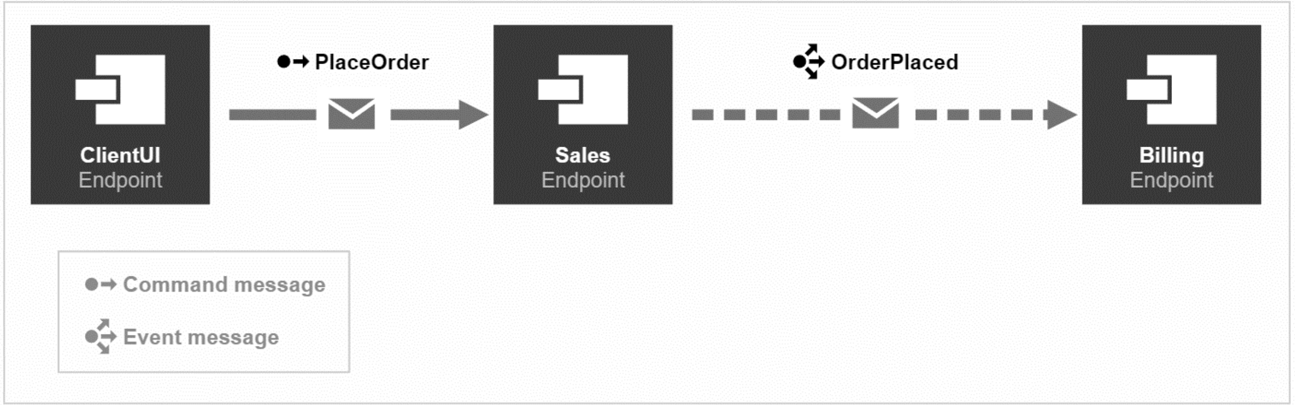
To get started, download the solution, extract the archive, and then open the **RetailDemo.sln** file with Visual Studio 2017.

Download the solution now

Project structure

The solution contains four projects. The **ClientUI**, **Sales**, and **Billing** projects are endpoints that communicate with each other using NServiceBus messages. The **ClientUI** endpoint mimics a web application and is an entry point in our system. The **Sales** and **Billing** endpoints contain business logic related to processing and fulfilling orders. Each endpoint references the **Messages** assembly, which contains the definitions of messages as POCO class files.

As shown in the diagram, the **ClientUI** endpoint sends a **PlaceOrder** command to the **Sales** endpoint. As a result, the **Sales** endpoint will publish an **OrderPlaced** event using the publish/subscribe pattern, which will be received by the **Billing** endpoint.



The solution mimics a real-life retail system, where the command to place an order is sent as a result of a customer interaction, and the actual processing occurs in the background. Publishing an event allows us to isolate the code to bill the credit card from the code to place the order, reducing coupling and making the system easier to maintain over the long term. Later in this tutorial, we'll see how to add a second subscriber in the **Shipping** endpoint which would begin the process of shipping the order.

Slides, Links...

github.com/danielmarbach/MessagePump

Particular
Software

Q&A

Software Engineer
Enthusiastic Software Engineer
Microsoft MVP

@danielmarbach
particular.net/blog
planetgeek.ch

Particular
Software

Thanks