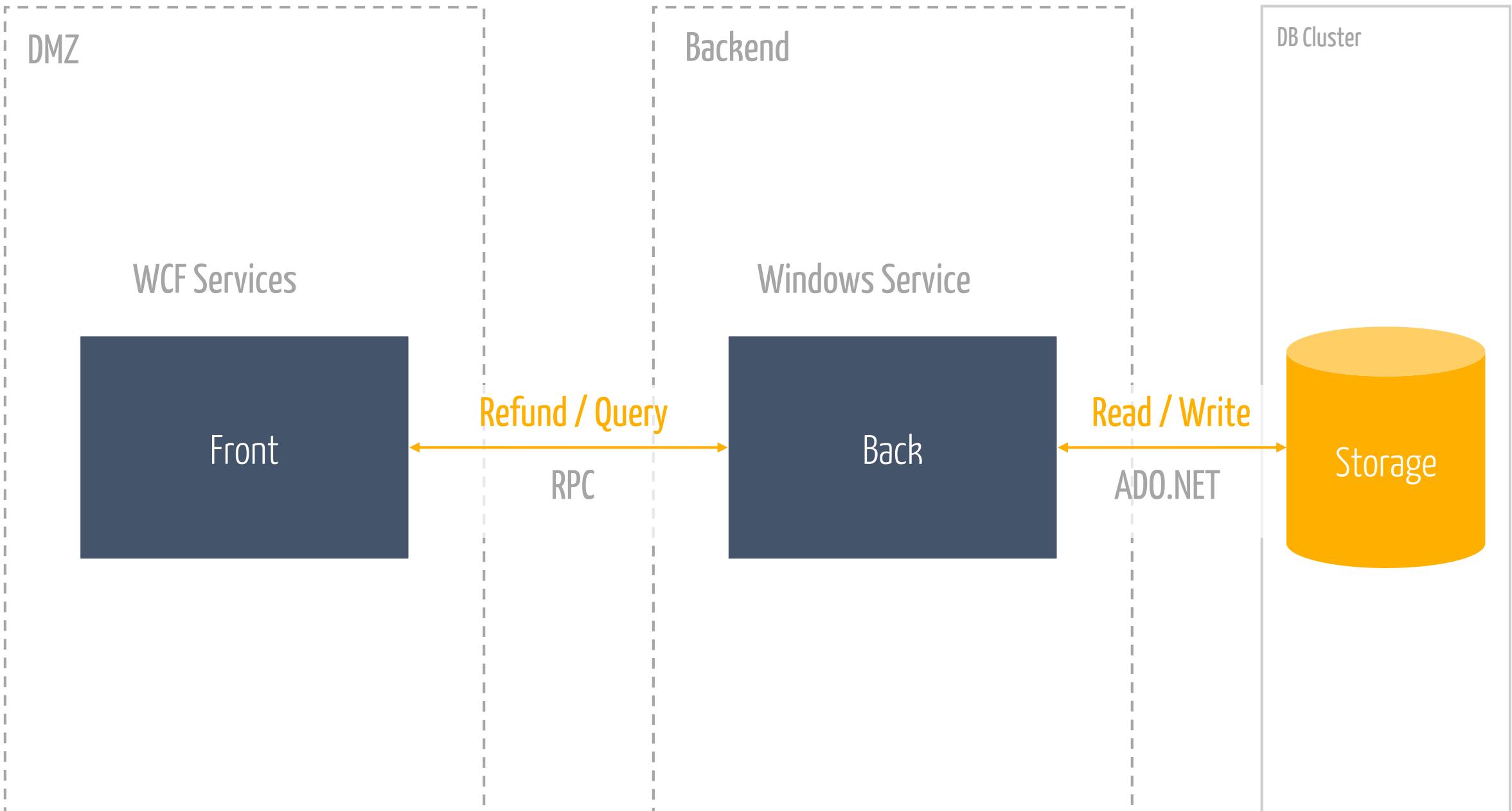




Do it yourself. A message pump
that kicks ass

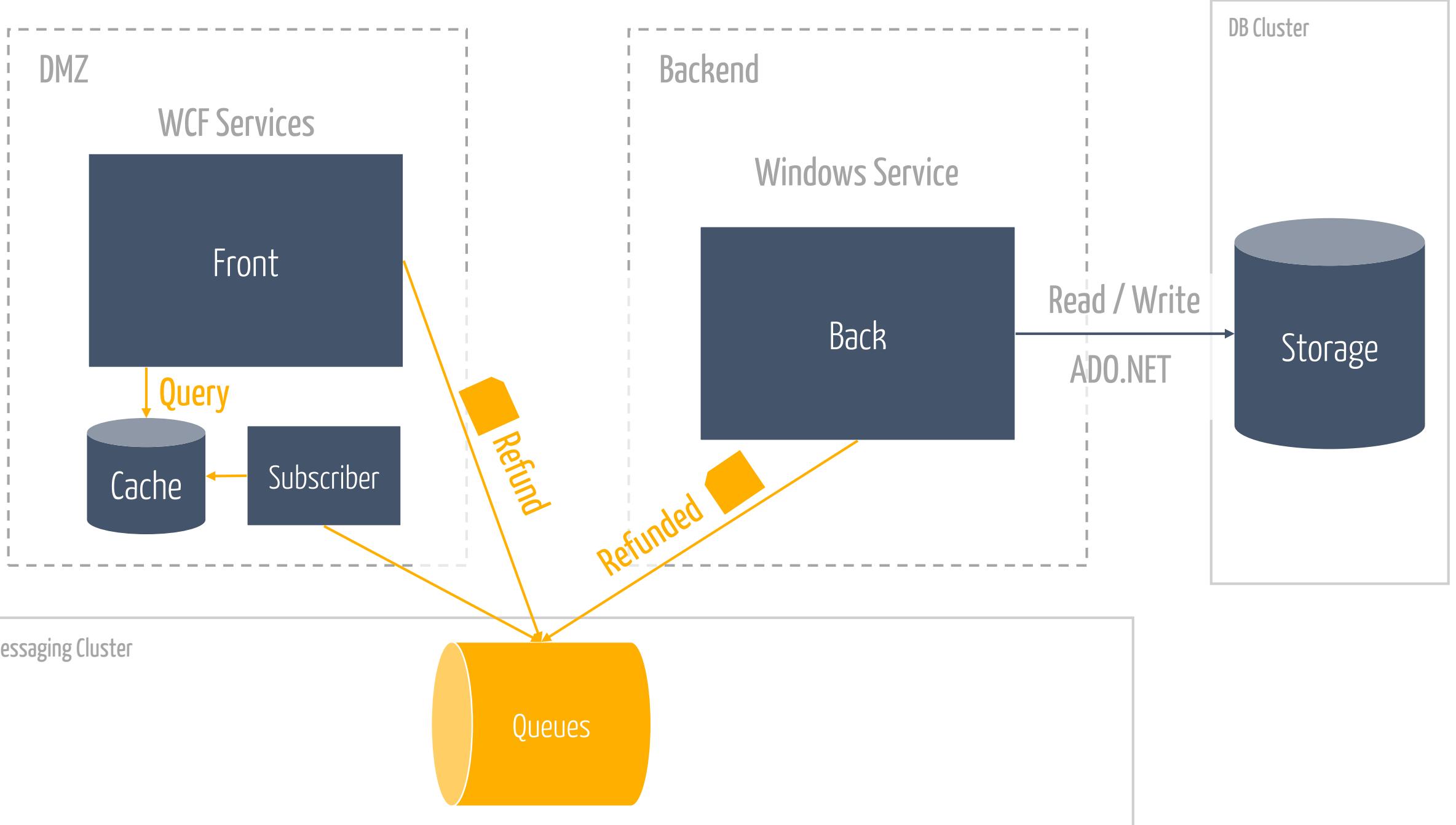


The RPC callstack of

doom

praise the lords of

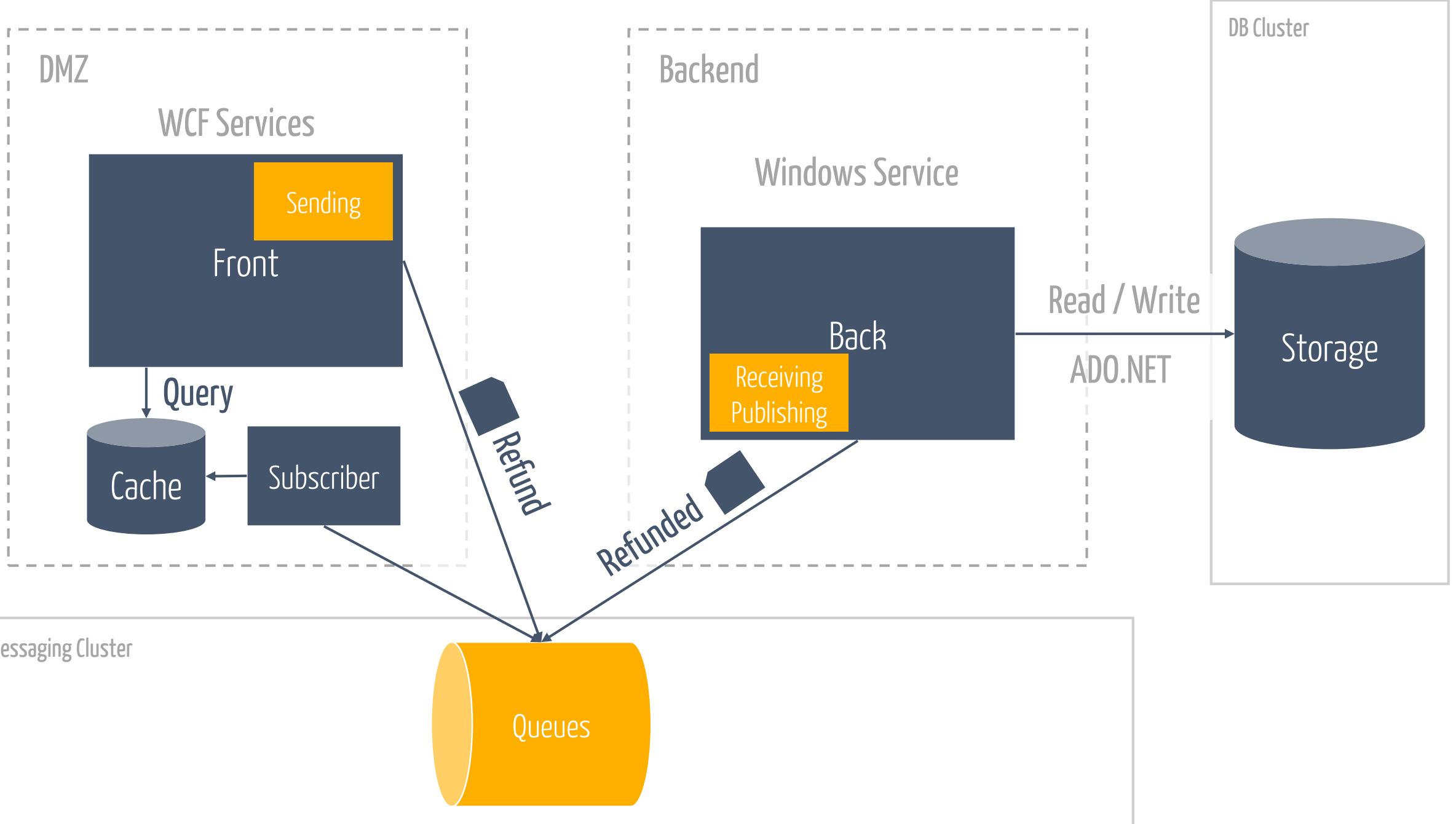
messaging



Build

or

Buy



Building the

pump



TPL handwaving

Cooporative cancellation 101

Async / Await

Ship it!

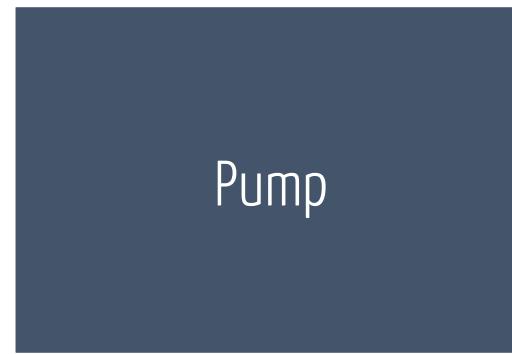
It worked until





Rush hour

The word "Rush" is rendered in a large, bold, orange sans-serif font. The letter "R" is partially overlaid by a vertical stack of five smaller, semi-transparent orange bars of varying heights, creating a sense of motion or depth. The word "hour" follows in a similar bold, orange font.



Let's throw in some

concurrency

Just a tiny change...
Introduce Fire & Forget
Ship it!

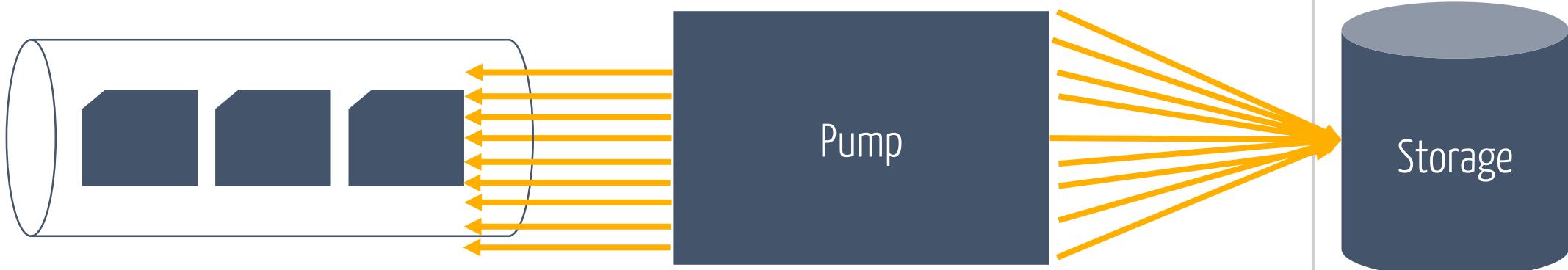
It worked until





Rush hour

The word "Rush" is rendered in a large, bold, orange sans-serif font. The letter "R" is partially overlaid by a vertical stack of five smaller, semi-transparent orange bars of varying heights, creating a sense of motion or depth. The word "hour" follows in a similar bold, orange font.



DB Cluster

Storage

Better limit

concurrency

Semaphore controls floodgate
Ship it!

It worked



A black and white photograph of a white car parked on a street. The car is positioned in the lower half of the frame, angled slightly towards the viewer. It is surrounded by dense, dark foliage from a large tree above, which obscures the upper portion of the image. The lighting creates strong shadows and highlights on the car's body.

As time passes

Make it

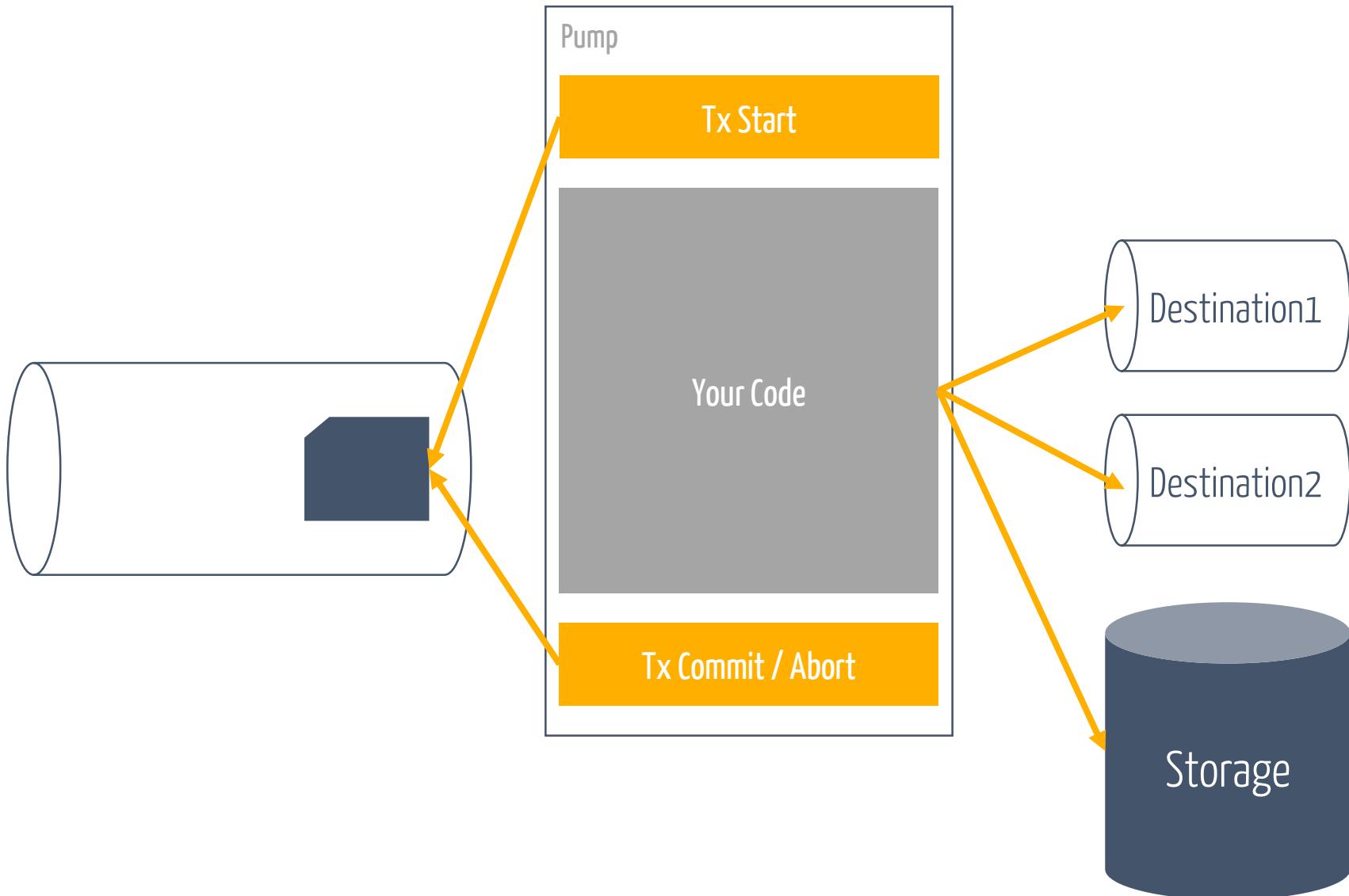
flextensible



PARSONS
FRUIT STAND
KELLOGG'S BC.

Life beyond

transactions



Cloudy with a chance of

failura

Penny Pinching

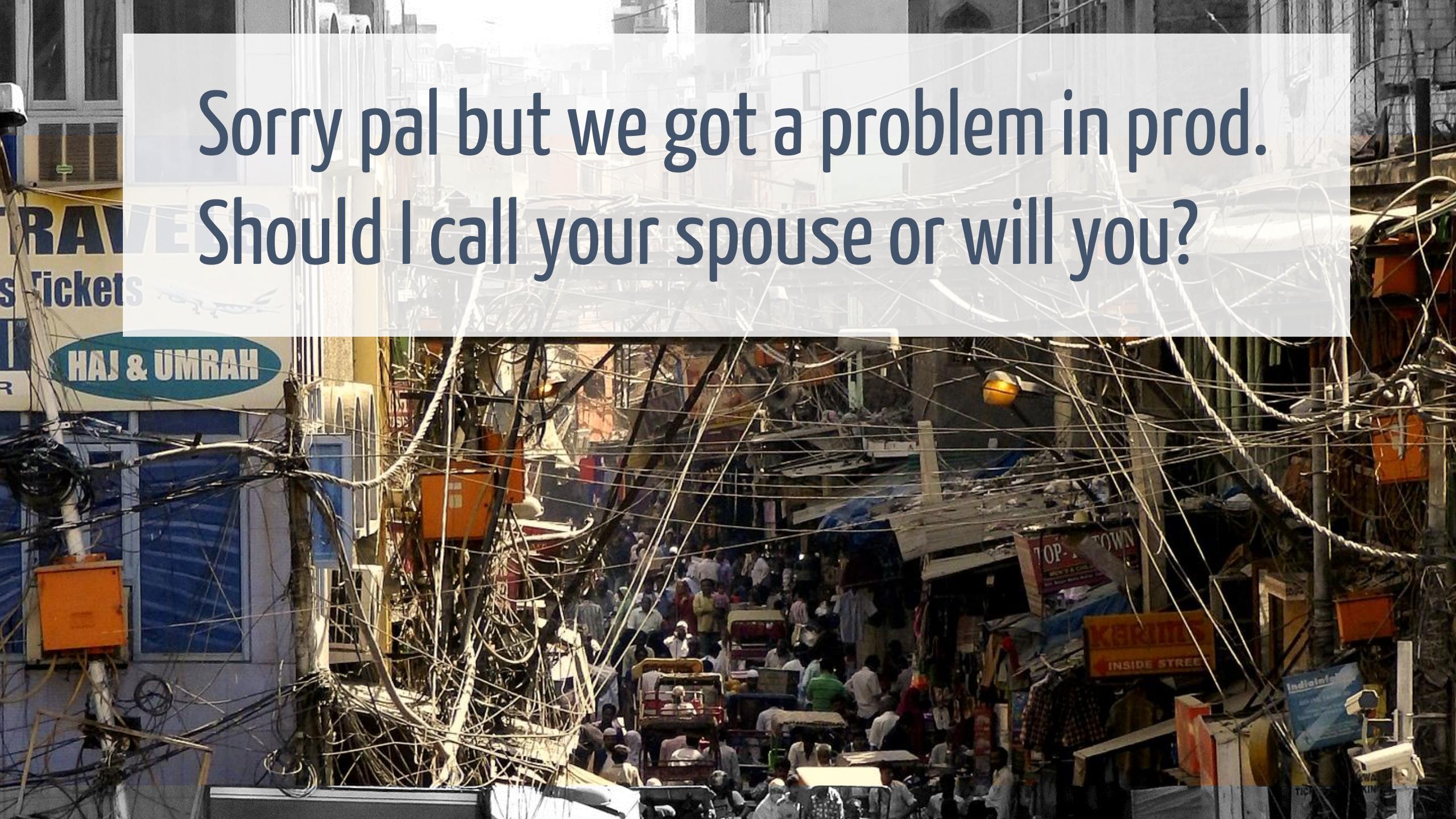
Now we have the

basic bits

Living on the edge of sanity



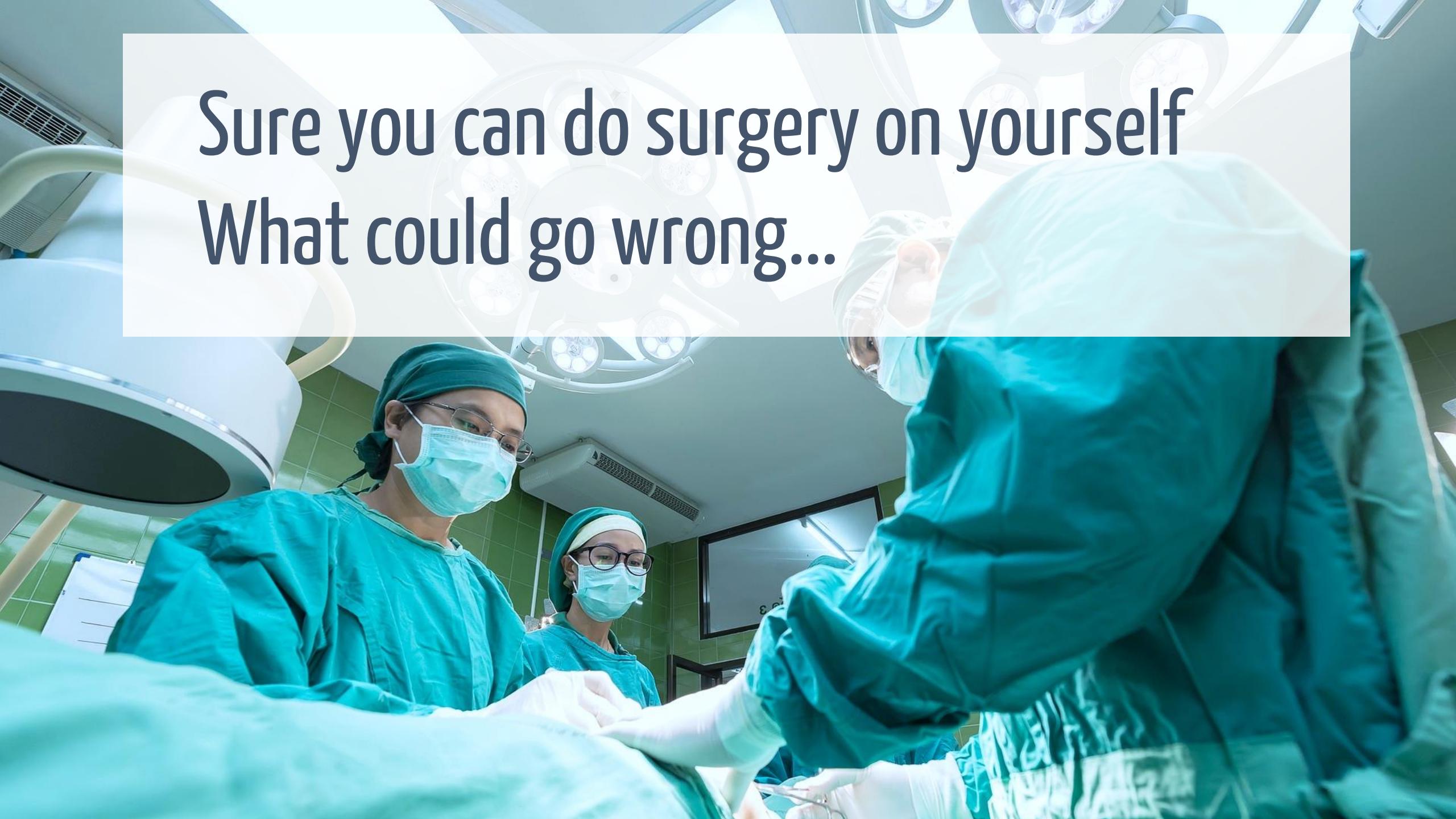
Sorry pal but we got a problem in prod.
Should I call your spouse or will you?



A pair of binoculars is positioned diagonally across the frame, serving as a background for the text. The binoculars are silver or chrome-colored with a textured eyecup. The background behind them is a soft, out-of-focus grey.

I want
insights

Sure you can do surgery on yourself
What could go wrong...





Thanks

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.

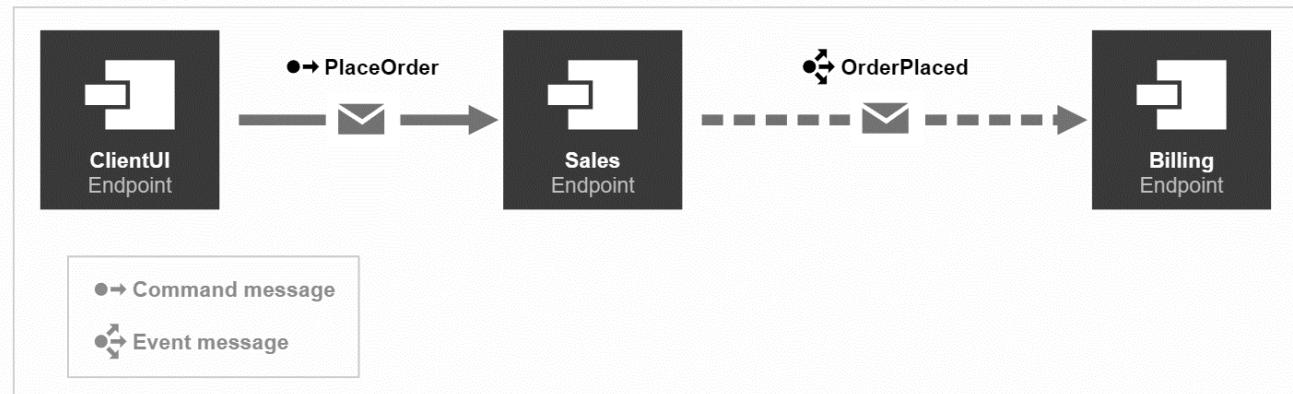
To get started, download the solution, extract the archive, and then open the **RetailDemo.sln** file with Visual Studio 2017[©].

[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.

NServiceBus

Getting Started
Service Platform
Quick Start Tutorial
Messaging Basics Tutorial
Getting Started
Sending a command
Multiple endpoints
Publishing events
Retrying errors
Message Replay Tutorial
Step by Step Sample
Configuration choices Sample
Concepts
Platform Installer
License
Extensions
Contributing
Architectural Principles
Bus vs. Broker

Upgrade Guides

Messaging

Hosting

Handlers and Sagas

Testing

Recoverability

Pipeline

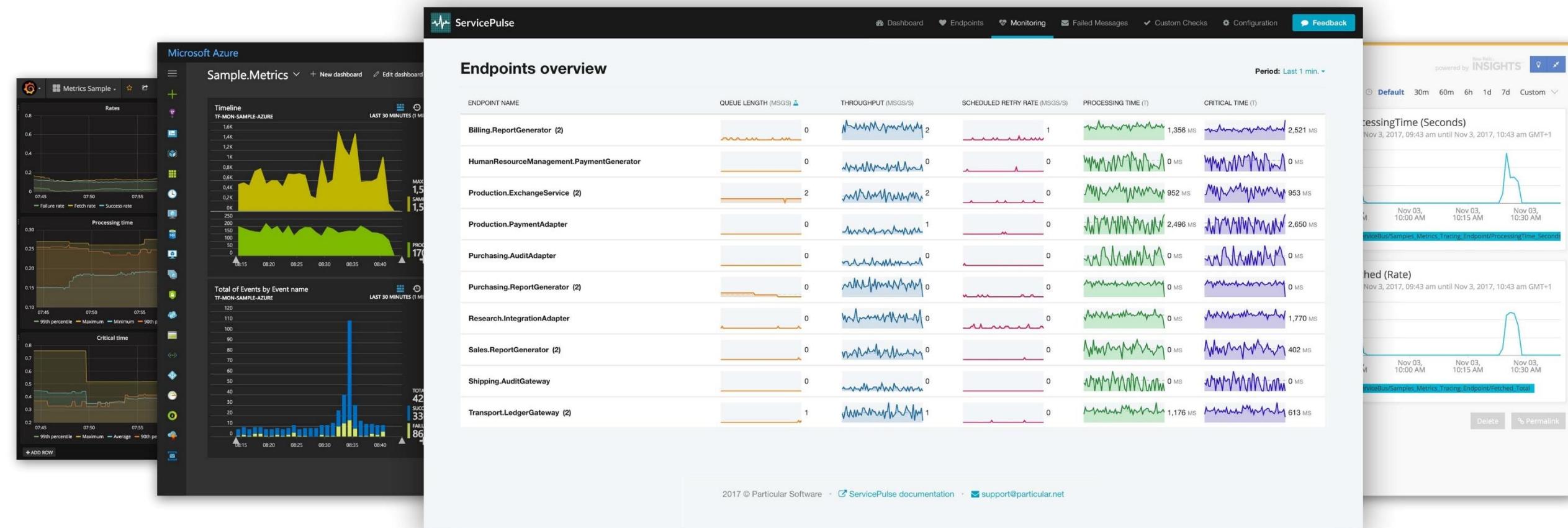
Serialization

Containers

Logging

Security

Operations



[particular.net/real-time-monitoring](https://www.particular.net/real-time-monitoring)

Slides, Links...

github.com/danielmarbach/MessagePump



Q & A



Software Engineer
Enthusiastic Software Engineer
Microsoft MVP

@danielmarbach
particular.net/blog
planetgeek.ch



Thanks