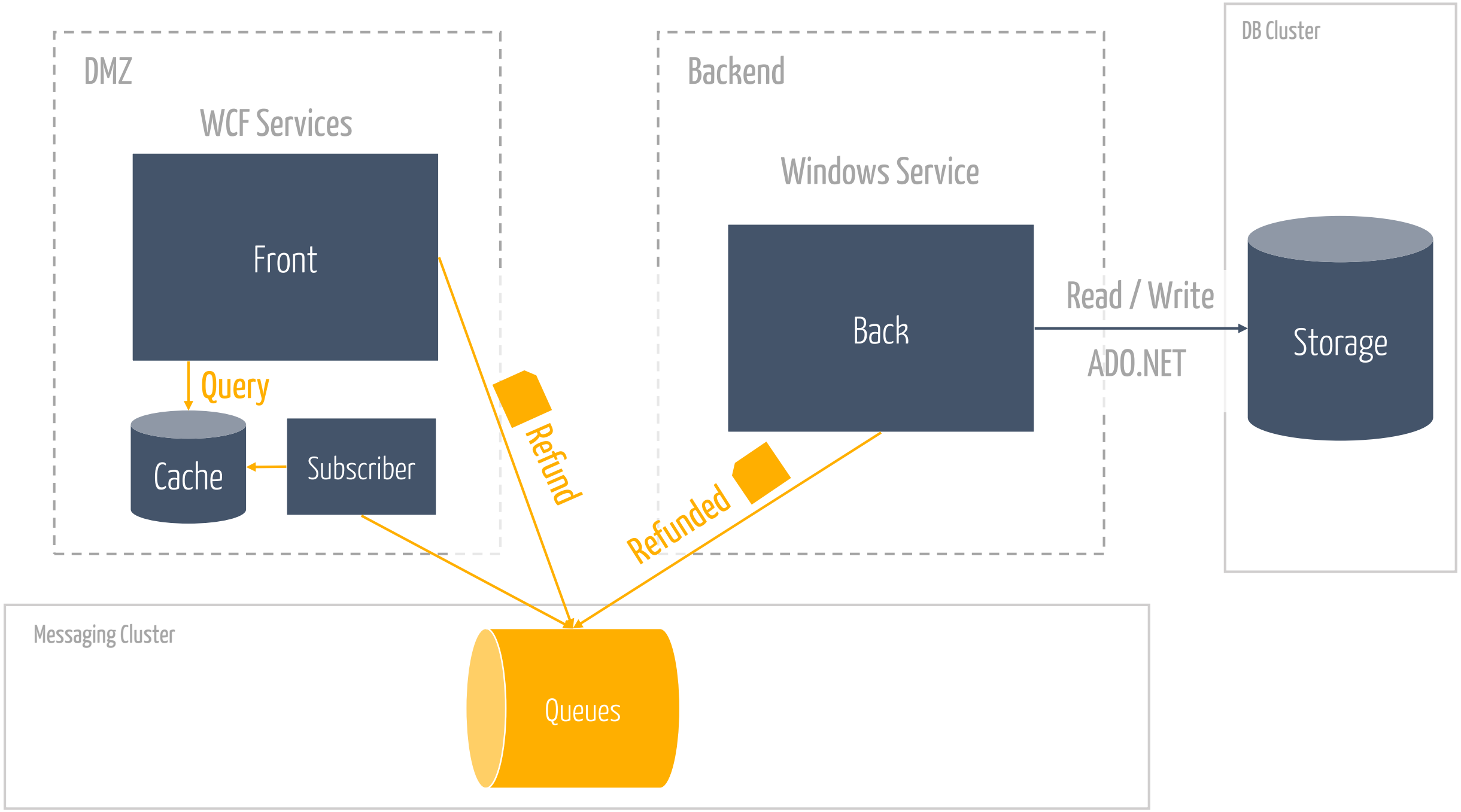
Particular
Software

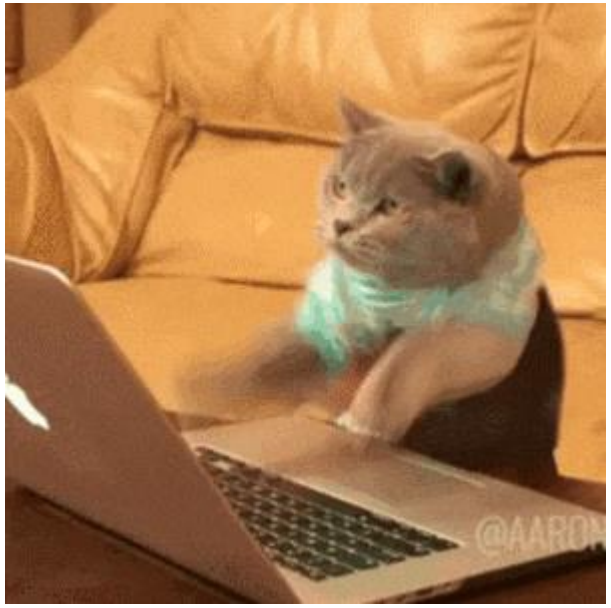
DIY Async Message Pump

Lessons from the trenches

@danielmarbach

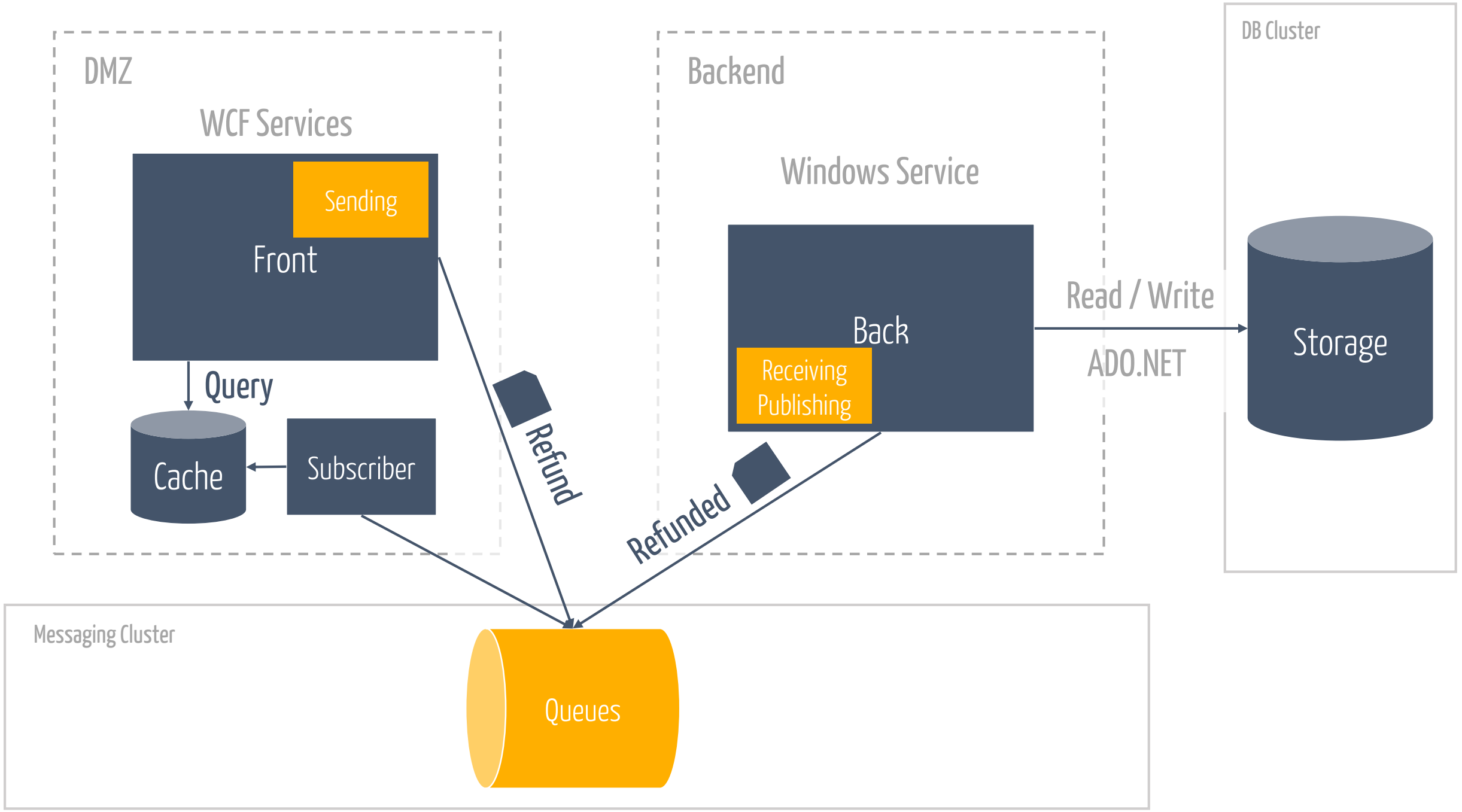

DMZ
WCF Services
Front
Query
Cache
Subscriber
Refund
Backend
Windows Service
Back
Refunded
Read / Write
ADO.NET
DB Cluster
Storage
Messaging Cluster
Queues

Build



or

Buy


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

Building the

pump

```
class ThePump : NotInteresting
{
    2 references
    Task pumpTask;
    4 references
    CancellationTokenSource tokenSource;

public void Start() {
    tokenSource = new CancellationTokenSource();
    var token = tokenSource.Token;
        public async Task Stop() {
            tokenSource.CancelAfter(TimeSpan.FromSeconds(5));
1
a
            await pumpTask.ConfigureAwait(false);                    ;

}
            tokenSource.Dispose();
        }
    }))();
}
```

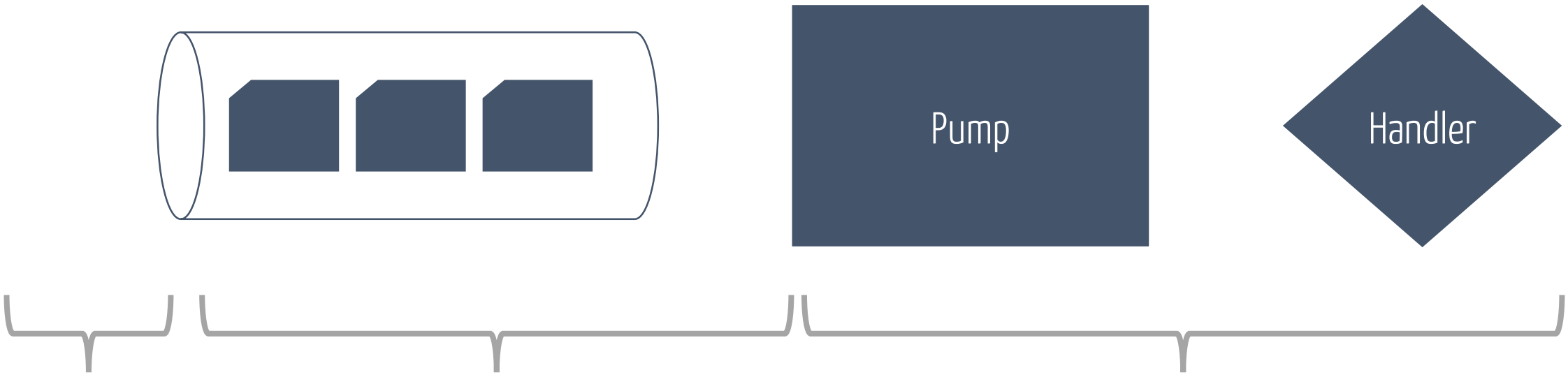
```
        await pumpTask.ConfigureAwait(false);

        tokenSource.Dispose();
    }
}
```


sending
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
Network
send time
Queue
wait time
Processing
time

```
C:\p\MessagePump
> dotnet run
```

Let's throw in some

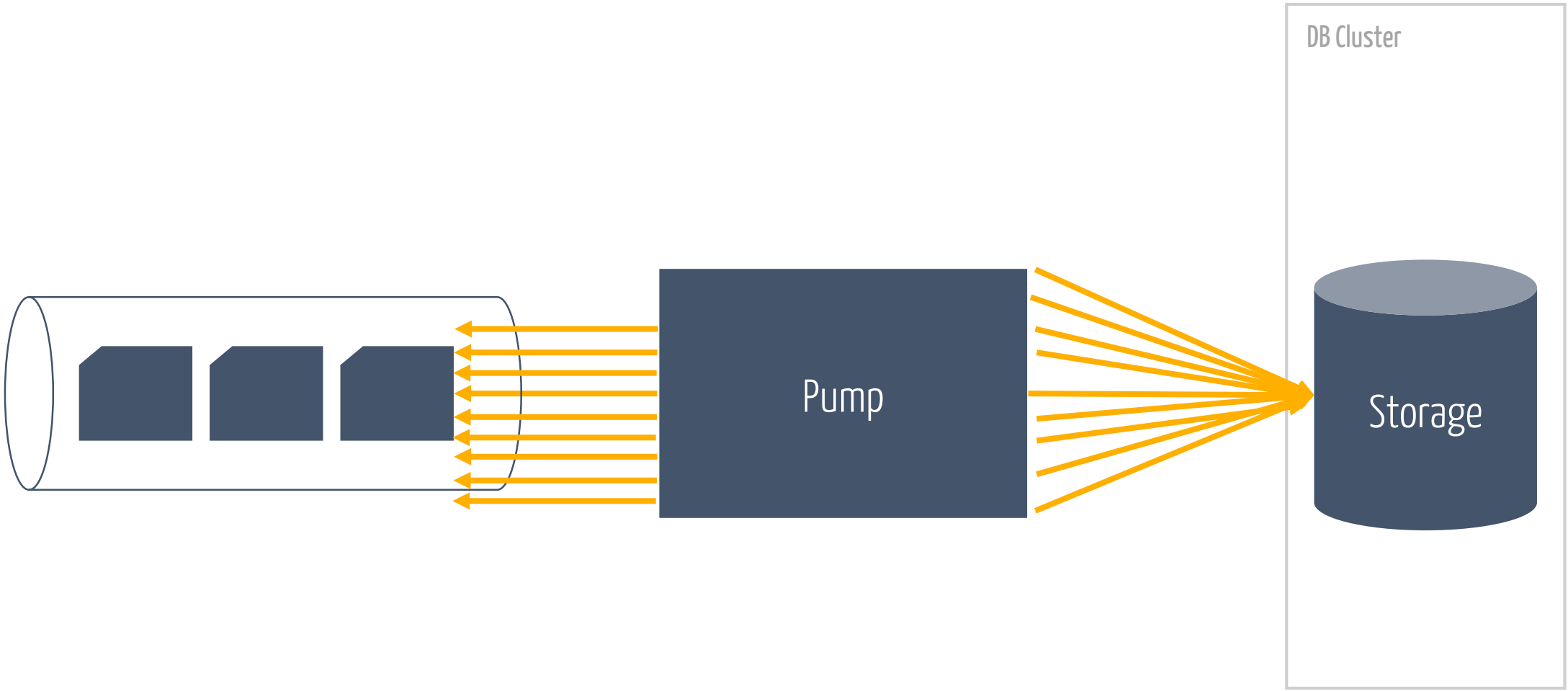
concurrency

```
class TheConcurrencyPump : NotInteresting
{
    2 references
    Task pumpTask;
    4 references
    CancellationTokenSource tokenSource;

    1 reference
    public void Start() {
        tokenSource = new CancellationTokenSource();
    }

    public async Task Stop() {
        tokenSource.CancelAfter(TimeSpan.FromMilliseconds(100));

        await pumpTask.ConfigureAwait(false);

        WriteLine("Are we done yet?");

        tokenSource.Dispose();
    }
}
```

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

It worked until


Rush hour


DB Cluster
Pump
Storage

```
C:\p\MessagePump
> dotnet run
ThePump executing
Pumping 1...
Pumping 2...
Pumping 3...
Pumping 4...
Pumping 5...
Press to continue...
```

Better limit

concurrency

```
class TheLimitingConcurrencyPump : NotInteresting
{
    #region Hide ···

    1 reference
    public void Start()
    {
        tokenSource = new CancellationTokenSource();
        semaphore = new SemaphoreSlim(MaxConcurrency);
```

```
    public void Start()
async Task FetchAndHandleAndRelease(SemaphoreSlim semaphore, CancellationToken token = default) {
    public async Task Stop() {
        tokenSource.CancelAfter(TimeSpan.FromSeconds(5));

        await Graceful(pumpTask).ConfigureAwait(false);

        while (semaphore.CurrentCount != MaxConcurrency) {
}           await Task.Delay(50).ConfigureAwait(false);
1 r     }
as
        tokenSource.Dispose();
}   }
    }
```

```
        await Graceful(pumpTask).ConfigureAwait(false);

        while (semaphore.CurrentCount != MaxConcurrency) {
            await Task.Delay(50).ConfigureAwait(false);
        }

        tokenSource.Dispose();
    }
}
```

Pumping 62577...
Pumping 62576...
Pumping 62575...
Pumping 62574...
Pumping 62573...
Pumping 62572...
Pumping 62571...
Pumping 62570...
Pumping 62569...
Pumping 62568...
Pumping 62567...
Pumping 62566...
Pumping 54400...
Pumping 54476...
Pumping 54519...
Pumping 54518...
Pumping 54391...
Pumping 54401...

Semaphore controls floodgate

Ship it!

It worked

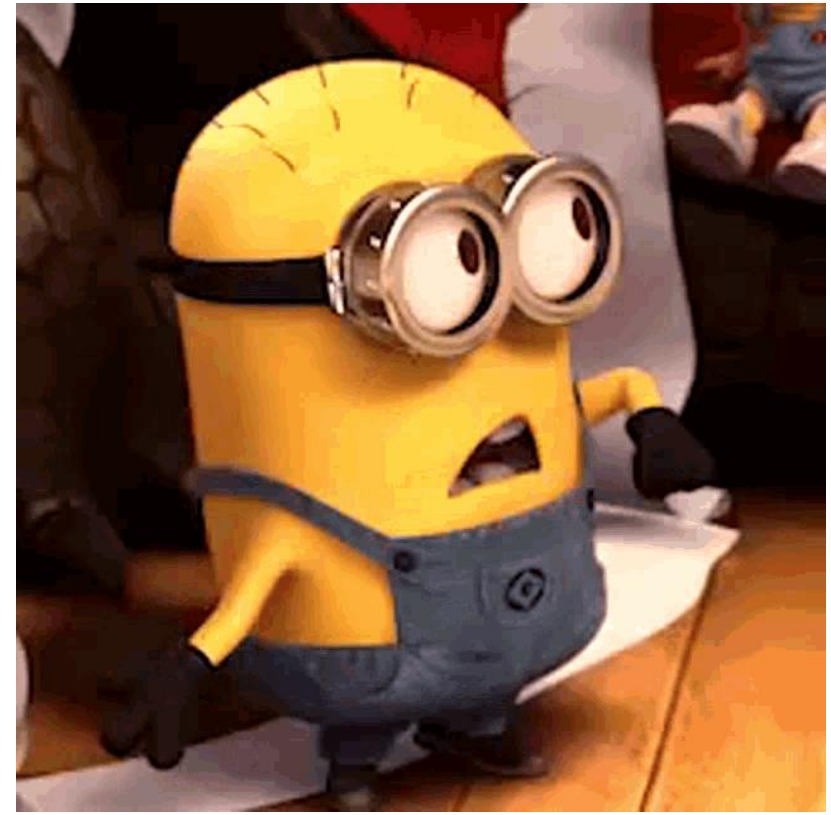
As time passes

Make it

flextensible

```
public class ThePumpHowItProbablyLooksLikeInReality : NotInteresting
{
    2 references
    private const int MaxConcurrency = 3;
    4 references
    private CancellationTokenSource tokenSource;
    4 references
    private SemaphoreSlim semaphore;
    2 references
    private Task pumpTask;

    0 references
    public void Start() {
        tokenSource = new CancellationTokenSource();

        var token = tokenSource.Token;

        semaphore = new SemaphoreSlim(MaxConcurrency);

        pumpTask = ((Func<Task>)(async () => {
            while (!token.IsCancellationRequested)
            {
                await semaphore.WaitAsync(token);

                FireAndForget(FetchAndHandleAndReleaseWithMiddleware(semaphore, token));
            }
        }))();
    }

    0 references
    public async Task Stop() {
        tokenSource.CancelAfter(TimeSpan.FromSeconds(5));
        await Graceful(pumpTask).ConfigureAwait(false);

        while (semaphore.CurrentCount != MaxConcurrency) {
            await Task.Delay(50).ConfigureAwait(false);
        }

        tokenSource.Dispose();
    }
```

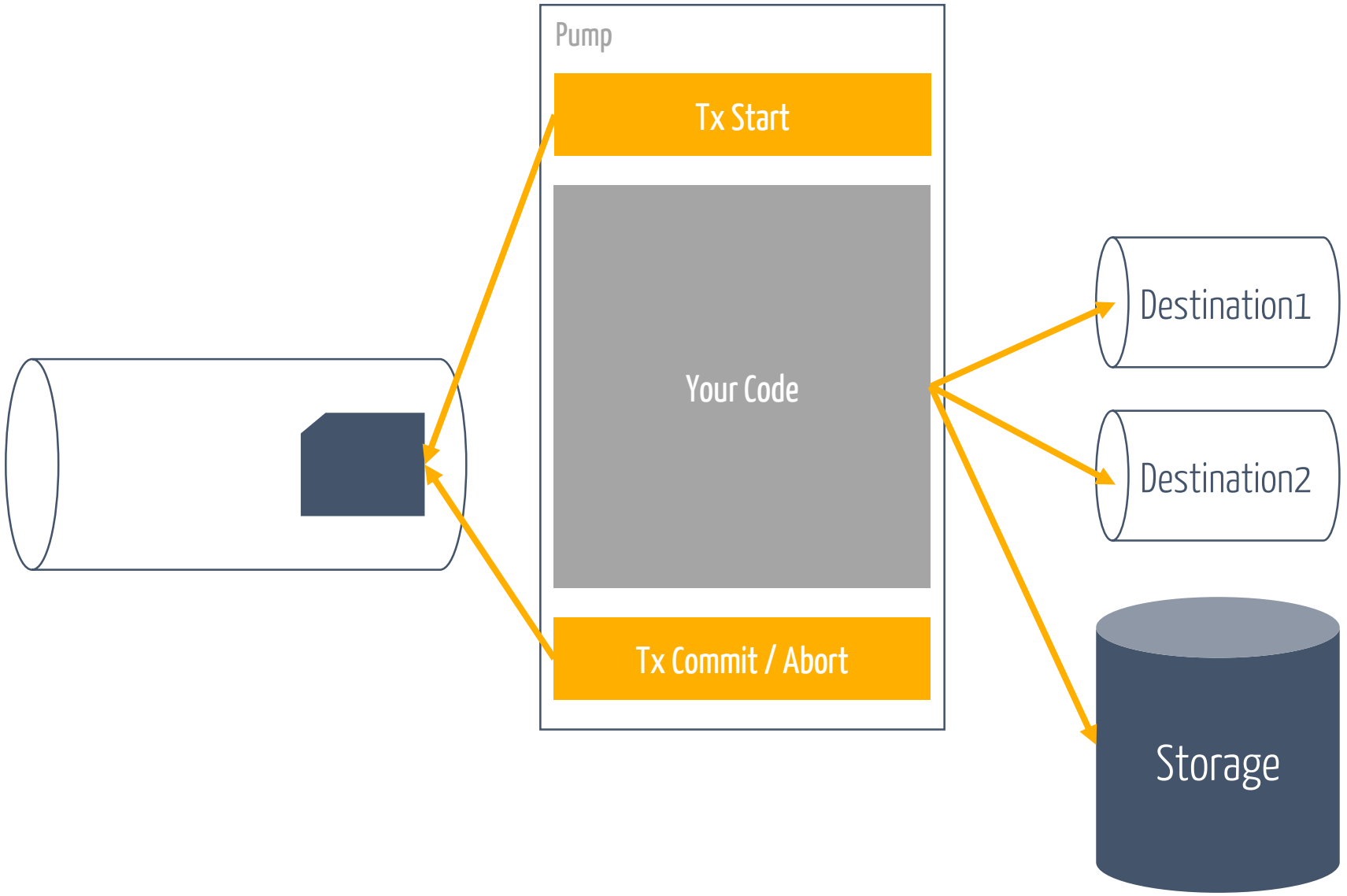
```
1 reference
async Task FetchAndHandleAndReleaseWithMiddleware(SemaphoreSlim semaphore, CancellationToken token = default) {
    using (var transaction = CreateTransaction()) {
        var (payload, headers) = await ReadFromQueue(token, transaction).ConfigureAwait(false);
        var message = Deserialize(payload, headers);

        using (var childServiceProvider = CreateChildServiceProvider()) {
            try {
                var middlewareFuncs = new Func<HandlerContext, Func<HandlerContext, CancellationToken, Task>, CancellationToken, Task>[] { Middleware1, Middleware2 };
                var middleware = FlextensibleMiddleware(childServiceProvider, middlewareFuncs, token);

                await middleware(message).ConfigureAwait(false);

                transaction.Complete();
            }
            catch (Exception) {
                // Just log?
            }
            finally {
                semaphore.Release();
            }
        }
    }
}

1 reference
async Task Middleware1(HandlerContext context, Func<HandlerContext, CancellationToken, Task> next, CancellationToken token = default) {
    await next(context, token);
}

1 reference
async Task Middleware2(HandlerContext context, Func<HandlerContext, CancellationToken, Task> next, CancellationToken token = default) {

    var handlers = context.Provider.Resolve<IHandleMessage<Message>>();

    foreach (var handler in handlers) {
        await handler.Handle(context.Message, context, token).ConfigureAwait(false);
    }

    await next(context, token).ConfigureAwait(false);
}

0 references
class MessageHandler : IHandleMessage<Message>
{
    1 reference
    public async Task Handle(Message message, HandlerContext context, CancellationToken token = default)
    {
        await Task.Delay(1000).ConfigureAwait(false);
        Pumping(message);
        await context.Send(new Message()).ConfigureAwait(false);
    }
}
```

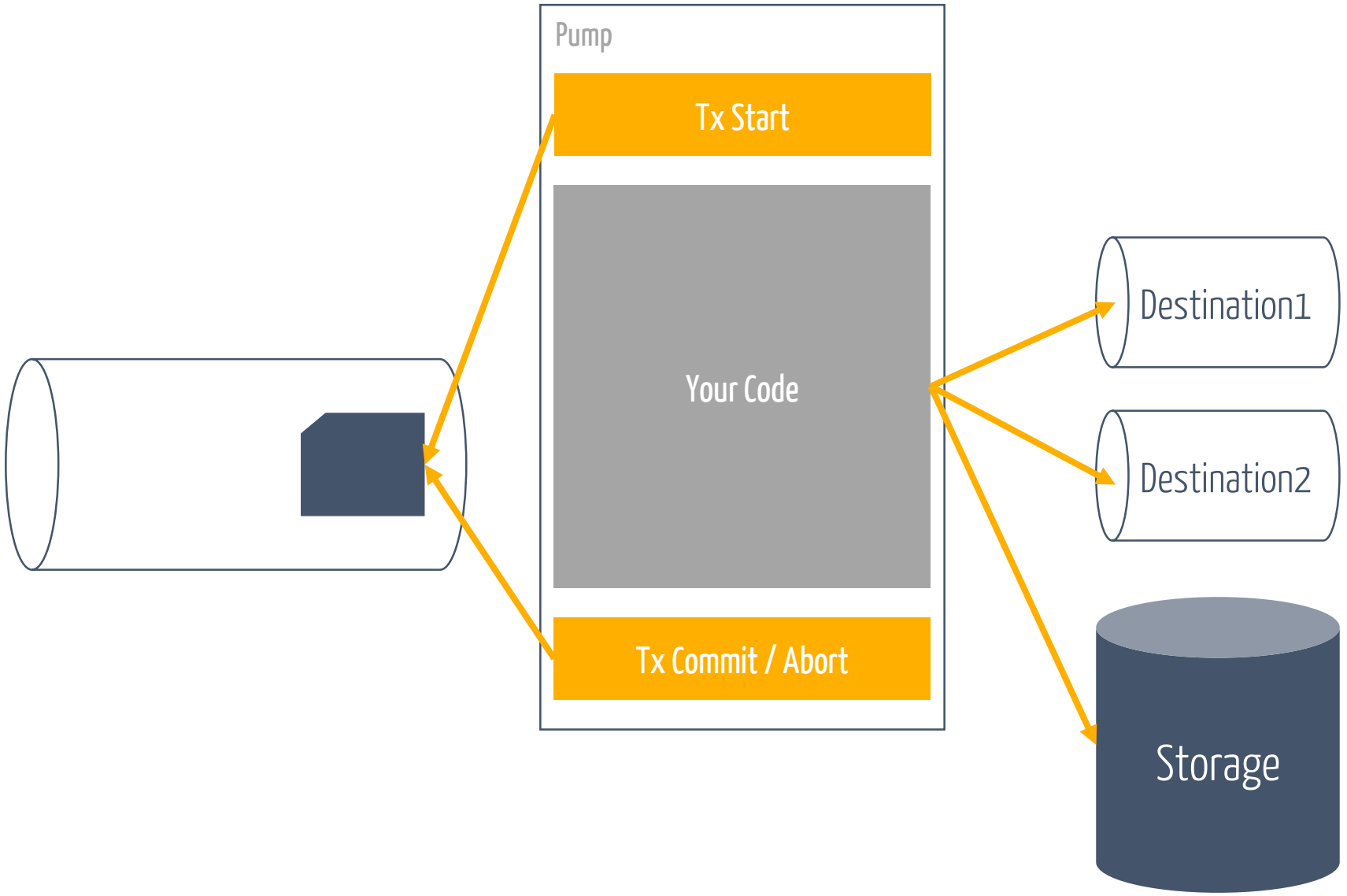


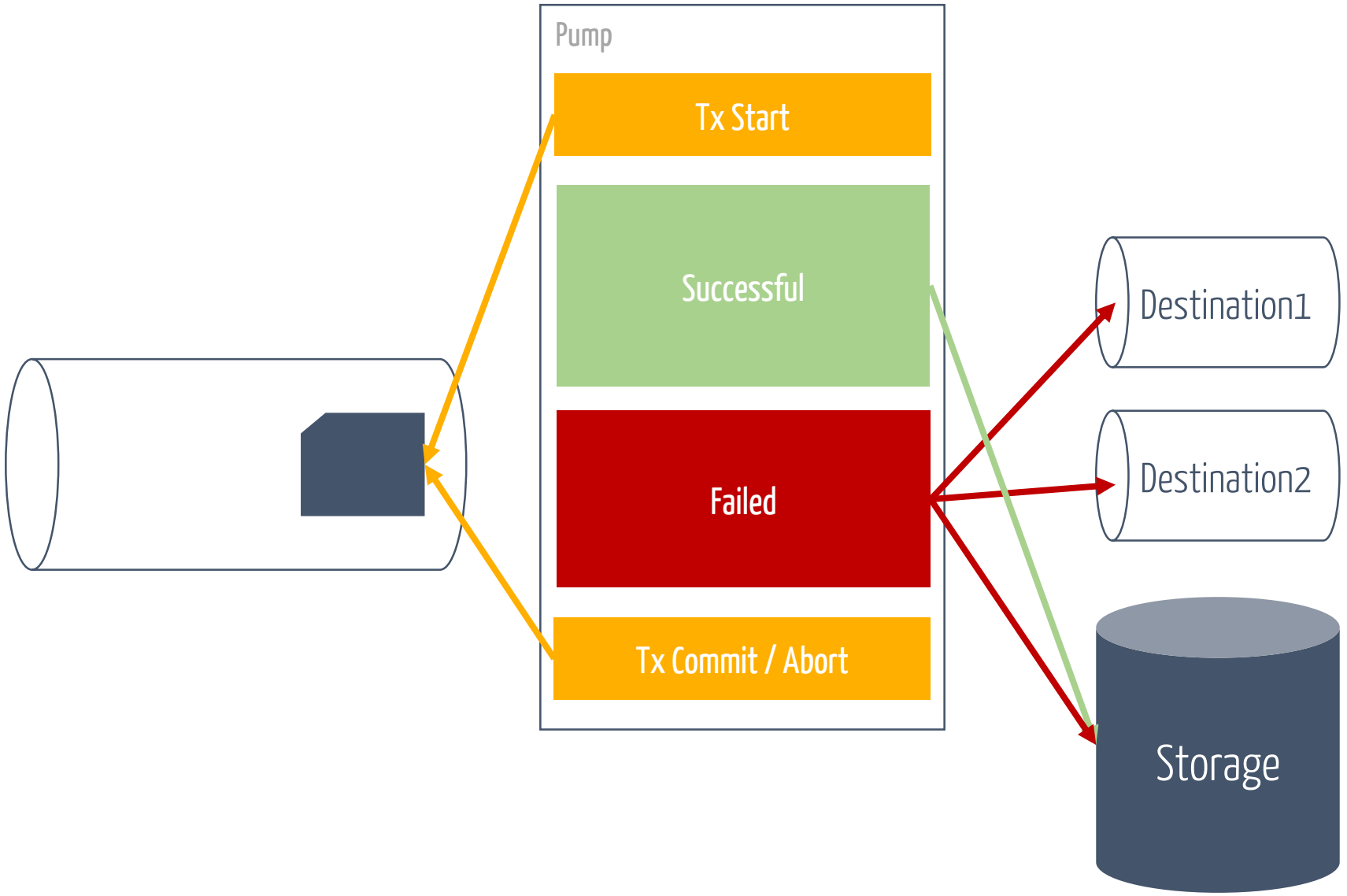
https://www.americanmuscle.com/gallery/image/1754-crazy-car-hi-res/

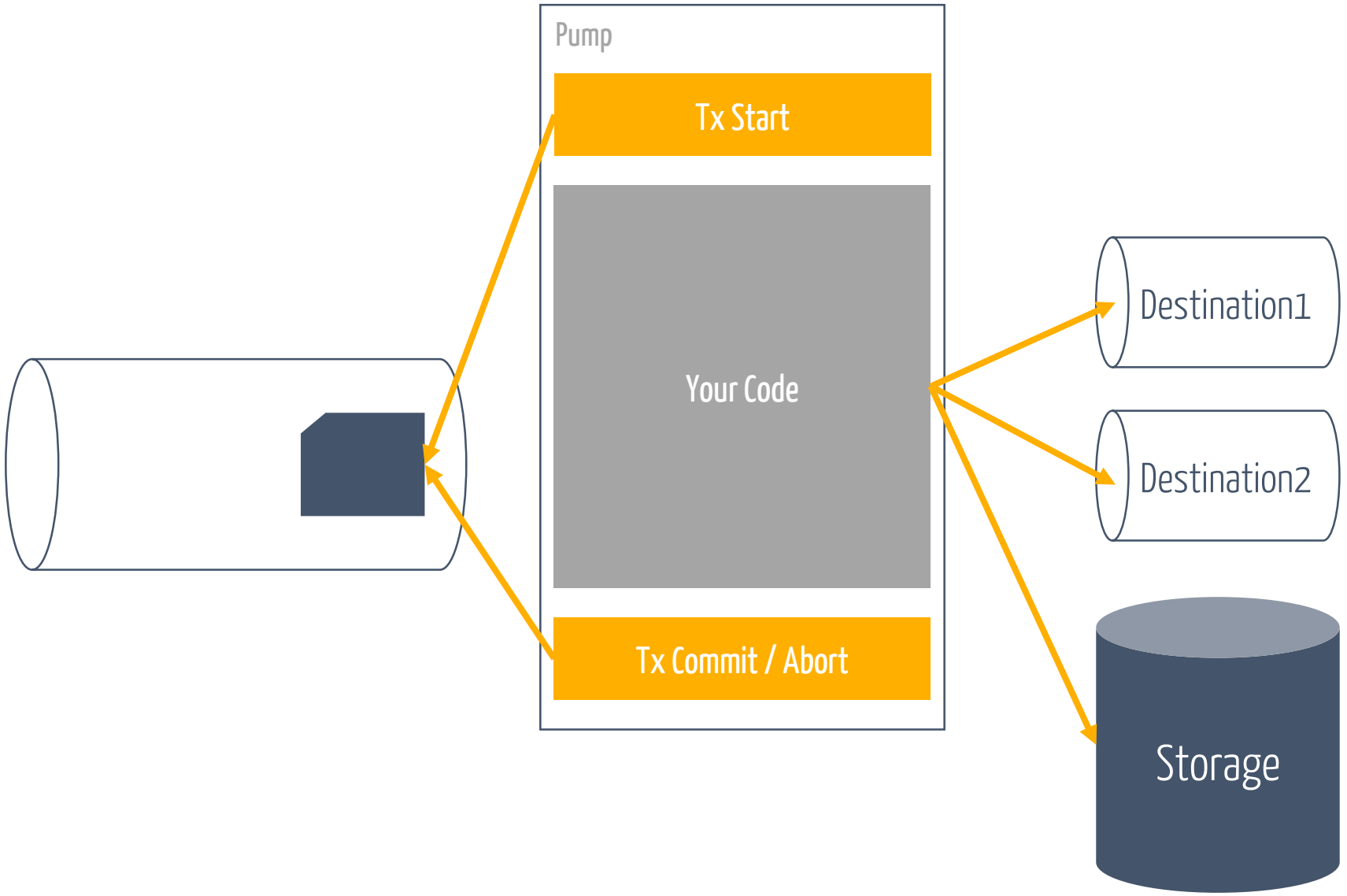
Life beyond

transactions


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


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

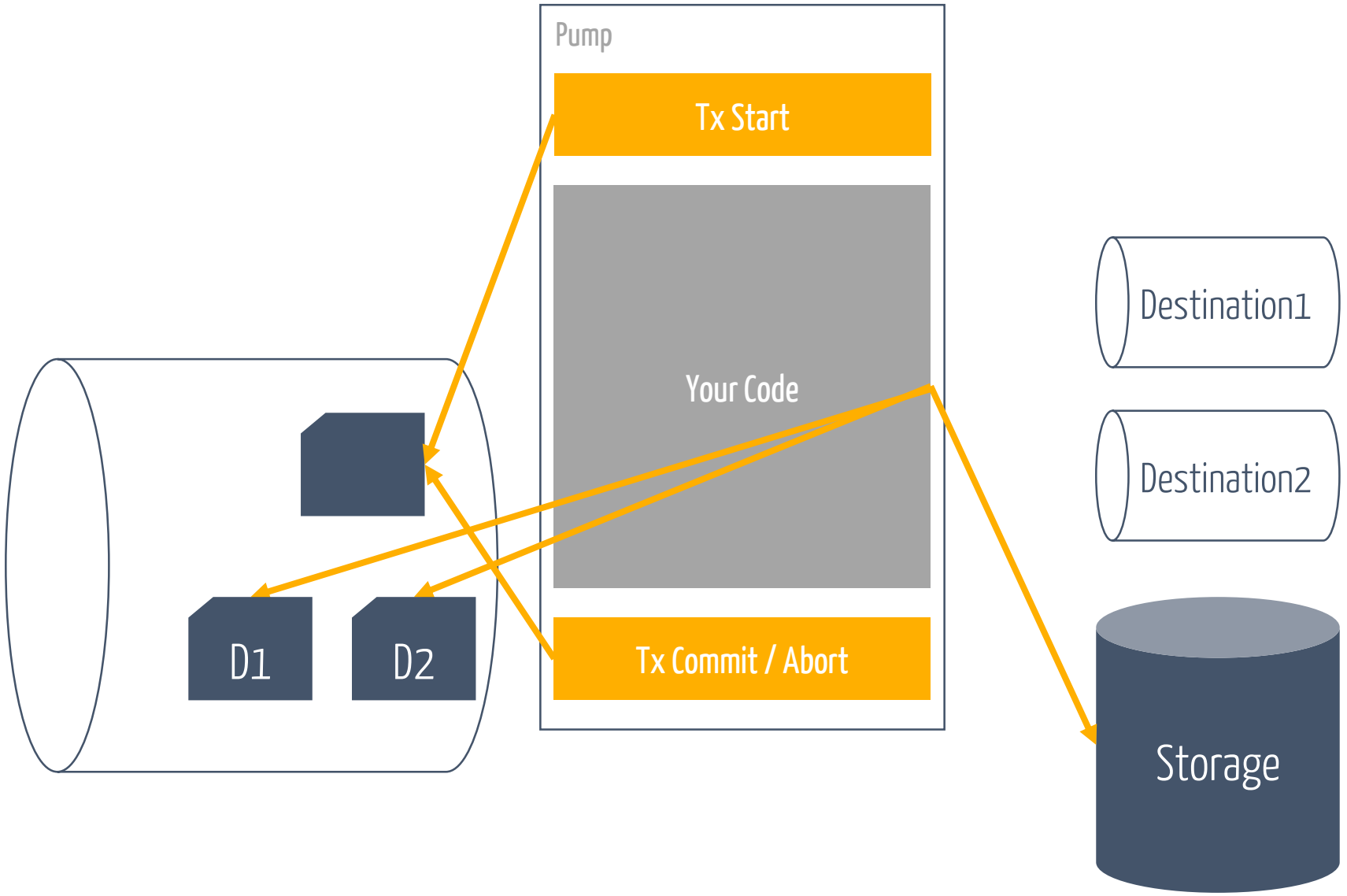

Pump
Tx Start
Successful
Failed
Tx Commit / Abort
Destination1
Destination2
Storage


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

Cloudy with a chance of

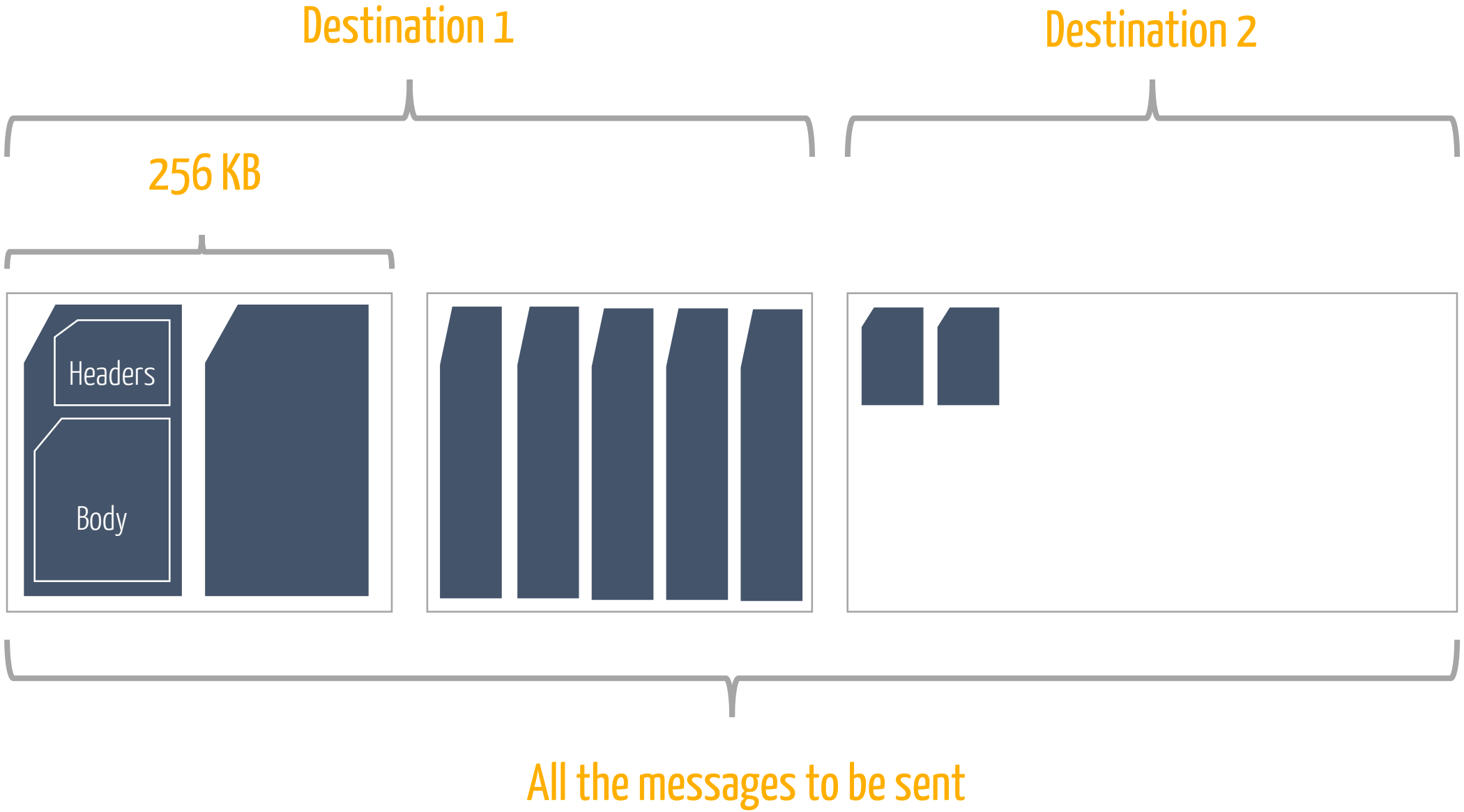

Pump


Handler


Pump
Tx Start
Your Code
Tx Commit / Abort
D1
D2
Destination1
Destination2
Storage

Penny Pinching

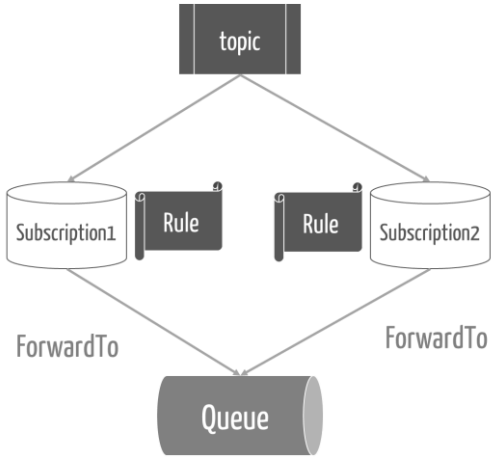
```
foreach(var message in messsagesToBeSent) {

    await SendMessage(message);

}
```

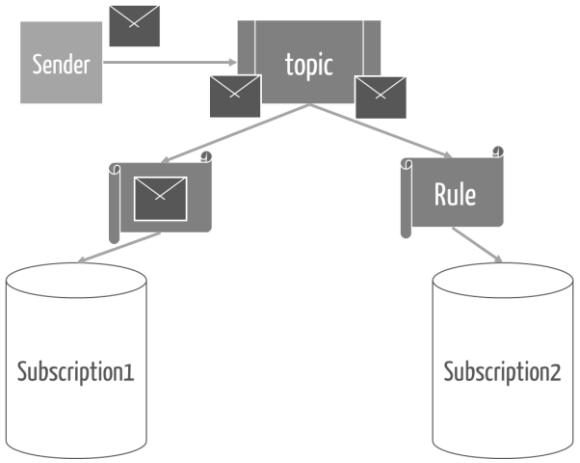

Destination 1
Destination 2
256 KB
Headers
Body
All the messages to be sent

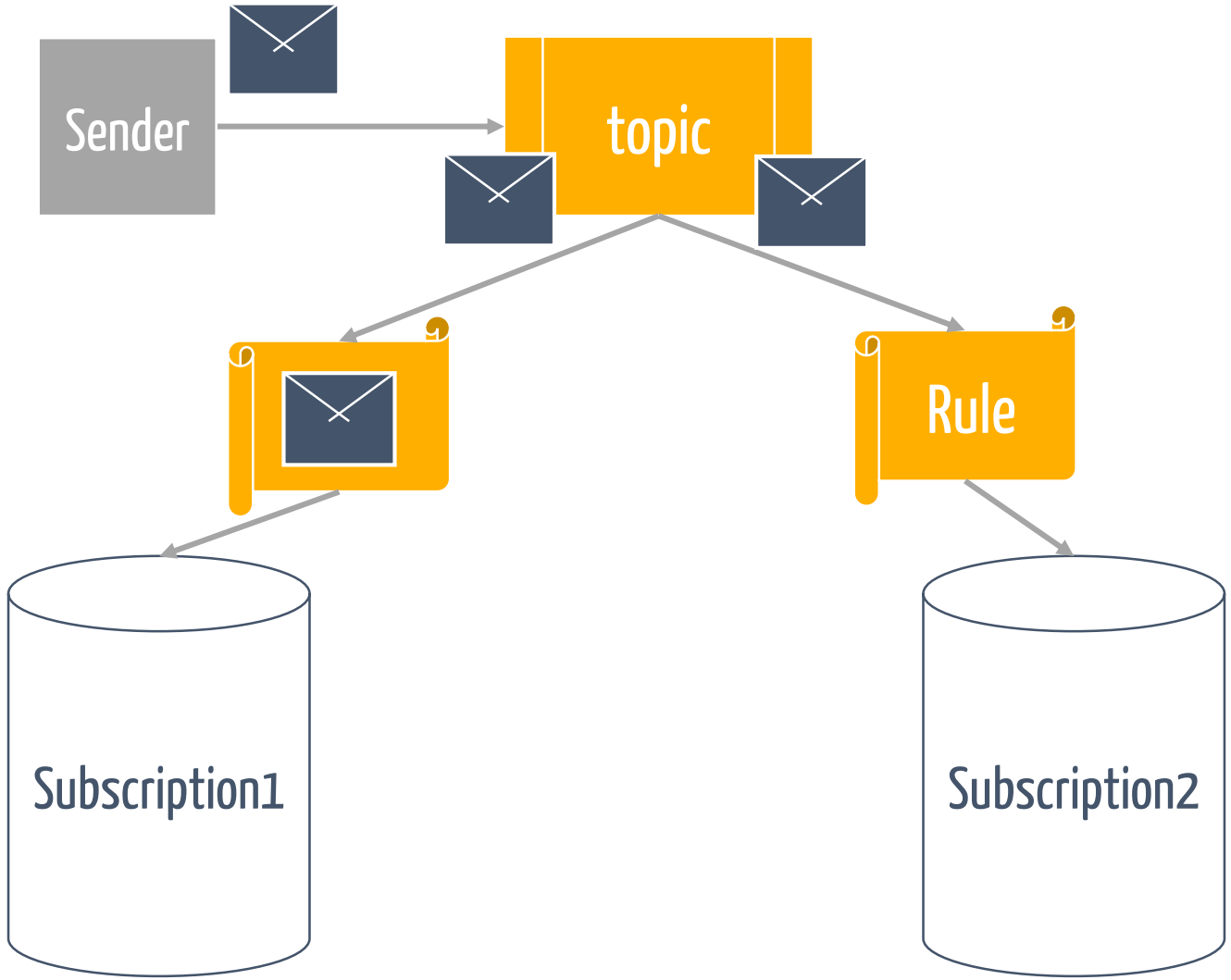
Now we have the

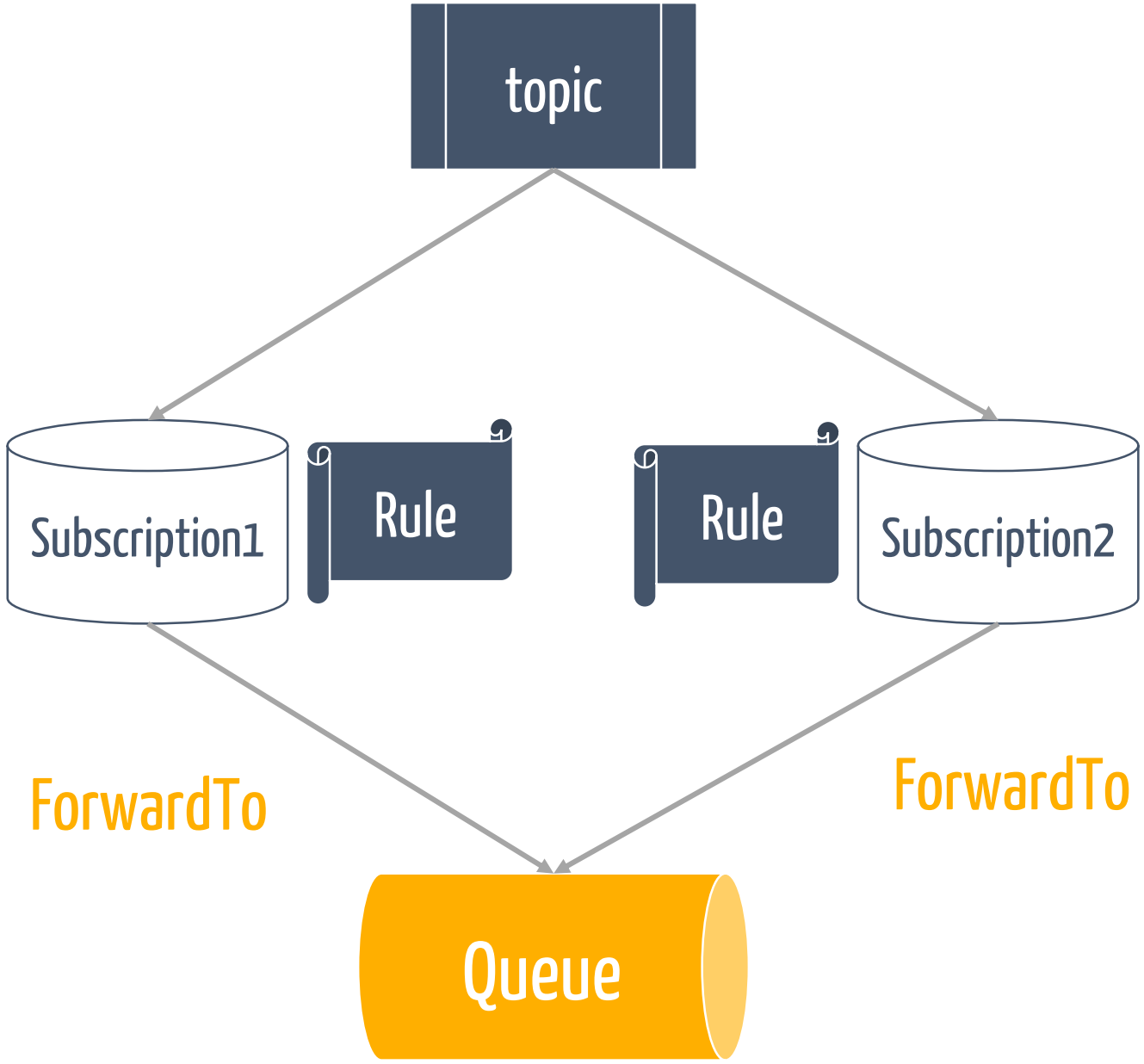
basic bits

Pub / Sub

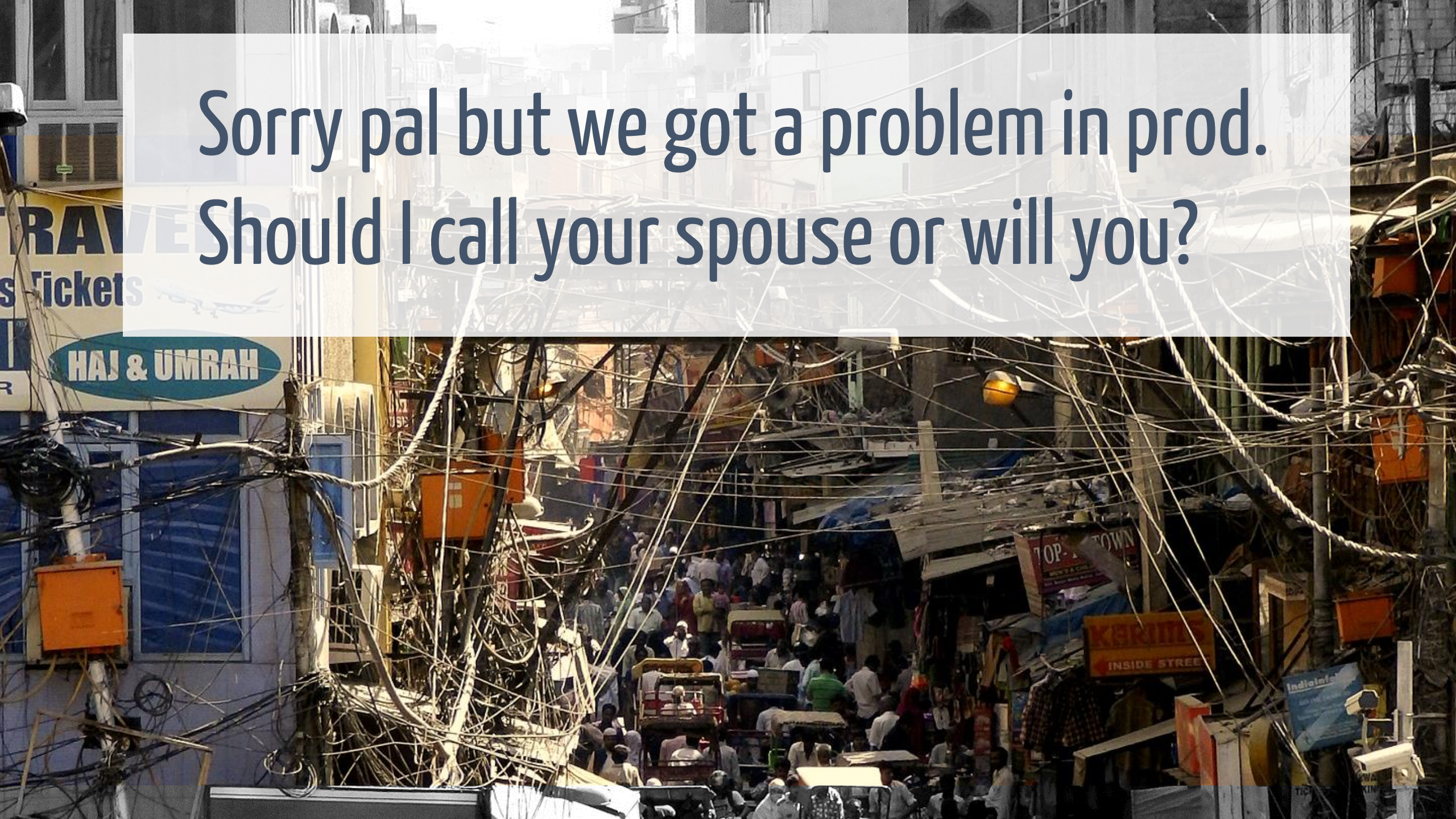

topic
Subscription1
Rule
Rule
Subscription2
ForwardTo
ForwardTo
Queue


Sender
topic
Rule
Subscription1
Subscription2

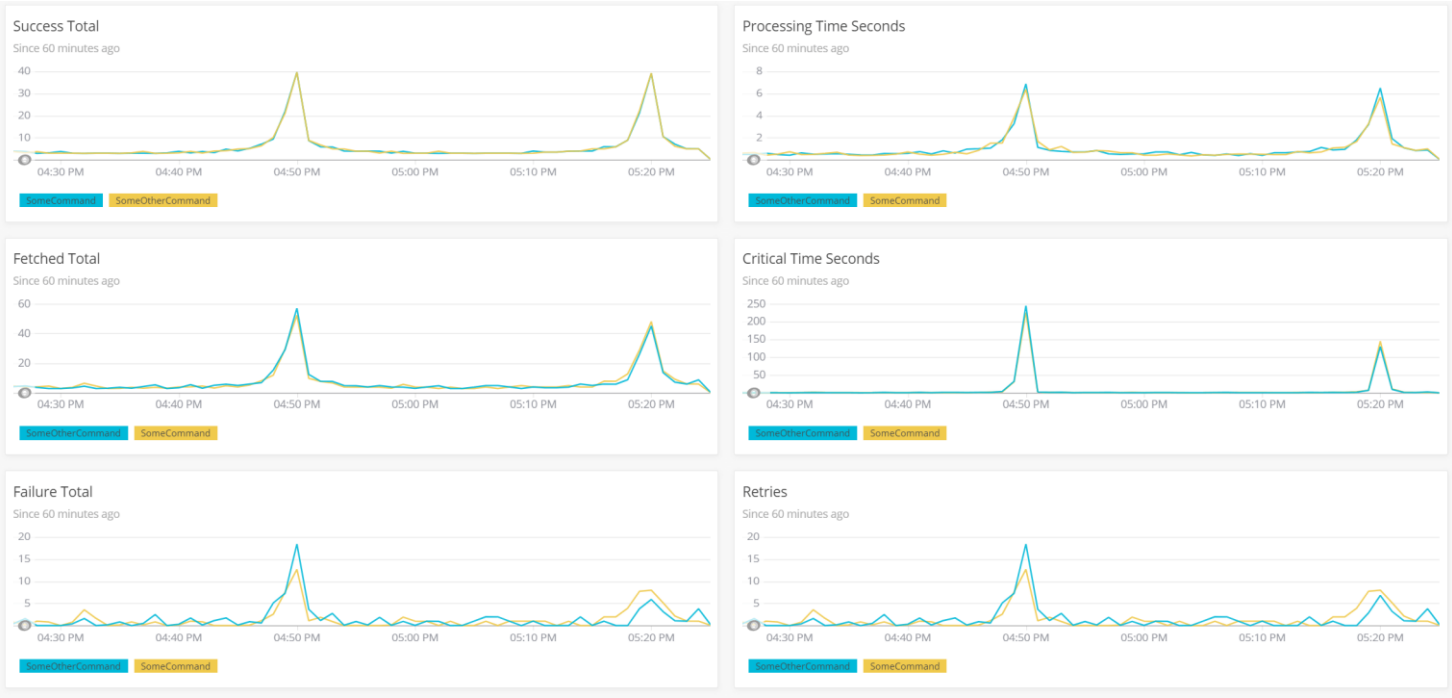

Sender
topic
Rule
Subscription1
Subscription2

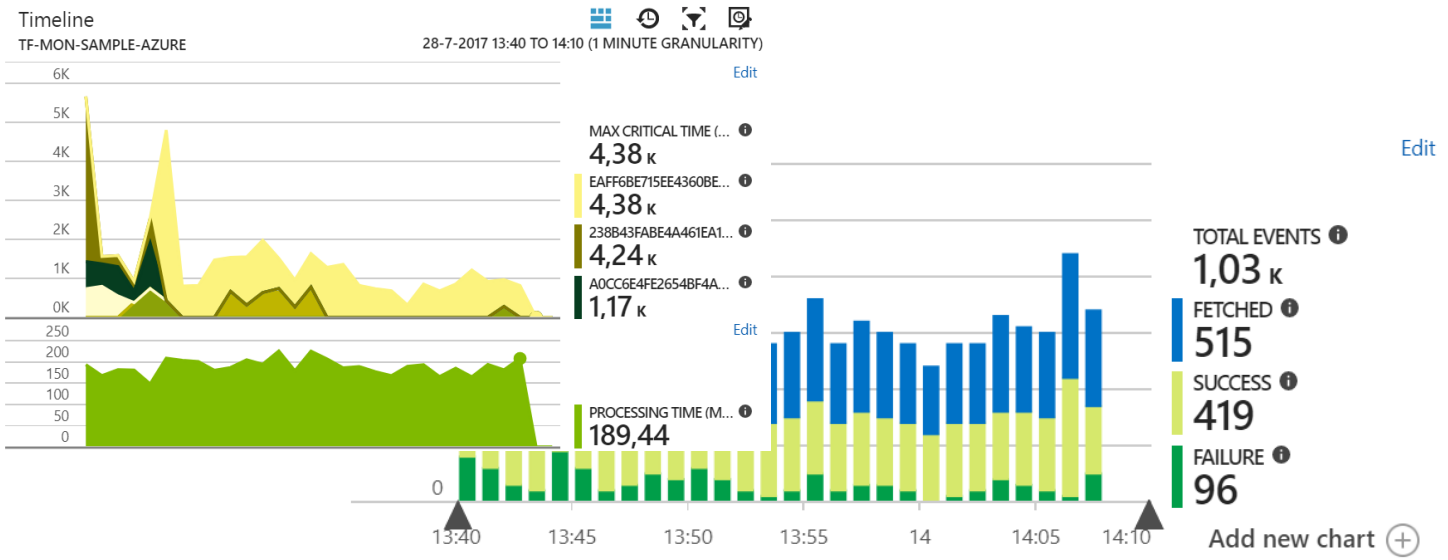

topic
Subscription1
Rule
Rule
Subscription2
ForwardTo
ForwardTo
Queue

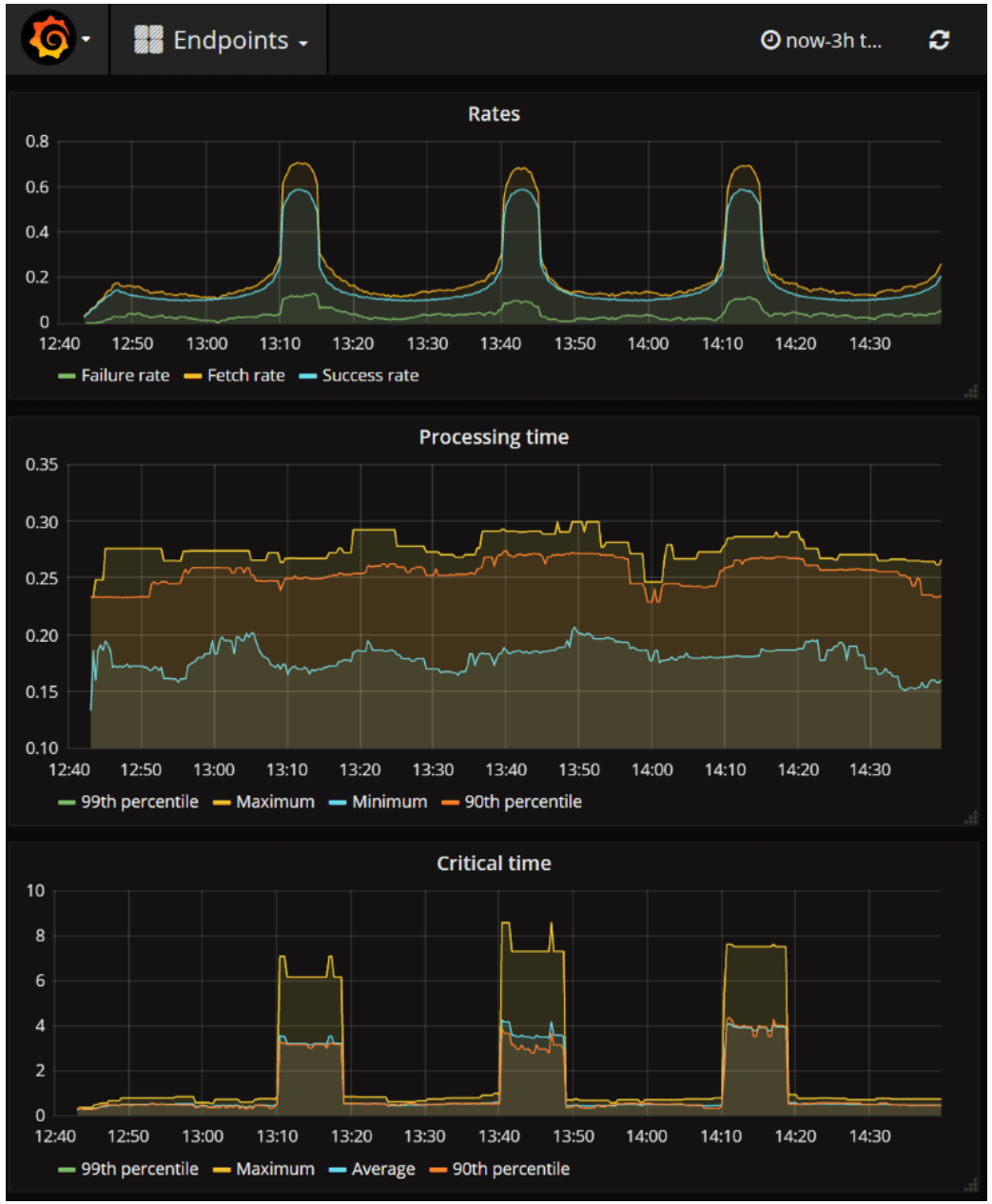
Living on the edge of sanity



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


Success Total
Since 60 minutes ago
Processing Time Seconds
Since 60 minutes ago
Fetched Total
Since 60 minutes ago
Critical Time Seconds
Since 60 minutes ago
Failure Total
Since 60 minutes ago
Retries
Since 60 minutes ago
SomeCommand
SomeOtherCommand


Timeline
TF-MON-SAMPLE-AZURE
28-7-2017 13:40 TO 14:10 (1 MINUTE GRANULARITY)
Edit
MAX CRITICAL TIME (...
4,38 K
EAFF6BE715EE4360BE...
4,38 K
238B43FABE4A461EA1...
4,24 K
A0CC6E4FE2654BF4A...
1,17 K
PROCESSING TIME (M...
189,44
TOTAL EVENTS
1,03 K
FETCHED
515
SUCCESS
419
FAILURE
96
Add new chart


Endpoints
now-3h t...
Rates
Failure rate
Fetch rate
Success rate
Processing time
99th percentile
Maximum
Minimum
90th percentile
Critical time
99th percentile
Maximum
Average
90th percentile


Try it yourself

Particular
Software

Thanks

Slides, Links...

github.com/danielmarbach/MessagePump

Particular Software

Software Engineer
Microsoft MVP

@danielmarbach
particular.net/blog
planetgeek.ch

Particular
Software

Thanks